

对标准化网页设计见解独特
一针见血揭示CSS技术的核心价值及应用
www.css8.cn网上讲座与答疑

精通CSS 网页布局



朱印宏 林小志 编著

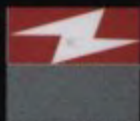


附1DVD

- 共计893分钟的超强视频讲解。
- CSS、Html、JavaScript、ASP教程（CHM格式）
- 本书实例源代码



中国电力出版社
www.cepp.com.cn



应用电工电子技术出版中心

010-58383409

策划编辑：马首鳌

电子信箱：ma_shouao@cepp.com.cn

封面设计：王英磊

朱印宏



资深网页设计师，从事网站开发与设计已有7年之久，对CSS、JavaScript、XHTML、XML、Ajax等网页前台技术有着深入的研究，并对这些技术的应用有着独到的见解，精通ASP和ASP.NET，能够融合前台和后台技术，实现网站混合开发。作者发表过Web设计文章十几篇，出版过8本Web技术专著，个人网站www.css8.cn专注于Web标准技术普及与推广。

林小志



原名林友赛，迅雷网络公司资深网站重构师，蓝色理想经典论坛标准版版主。多年网站设计和网站重构经验，对CSS、XHTML等前台技术有着深厚的功底。接触网站重构以来一直以“重构网站之前重构个人思维”的态度对待网站、追求网站结构完美语义性；也一直以这个态度跟他人沟通交流，也很喜欢大家以这样的态度跟我沟通。个人博客（www.linxz.cn）以Web标准技术普及与推广为基础与大家分享经验、交流思想。



上架建议：计算机 / 网页设计

ISBN 978-7-5083-7962-3



9 787508 379623 >

定价：45.00元（1DVD）



精通CSS 网页布局



朱印宏 林小志 编著



中国电力出版社
www.cepp.com.cn

内 容 提 要

本书深入、系统地讲解了使用 CSS 进行网页布局设计的相关知识和技巧,内容包括 CSS 基础、文字和版式设计、图像控制、超链接和导航菜单设计、表格和表单设计、浮动网页布局设计以及网页优化等。在书的最后,还对 CSS 框架设计进行案例分析,并通过一个购物网站设计实例来融会贯通所有知识点,以达到灵活应用的目的。

本书还附赠一张多媒体教学光盘,内含 12 课有关 CSS 的多媒体教学课件,总长度超过 13 个小时,与本书知识点基本同步,以帮助您快速了解 CSS 的核心技术,达到事半功倍的学习效果。

本书内容全面,示例丰富,讲解细致,非常适合网页设计与网站开发人员学习。

图书在版编目 (CIP) 数据

精通 CSS 网页布局 / 朱印宏, 林小志编著. —北京: 中国电力出版社, 2009
ISBN 978-7-5083-7962-3

I. 精… II. ①朱…②林… III. 主页制作—软件工具, CSS IV. TP393.092

中国版本图书馆 CIP 数据核字 (2008) 第 172302 号

责任编辑: 马首鳌
责任校对: 王开云
责任印制: 郭华清

书 名: 精通 CSS 网页布局

编 著: 朱印宏 林小志

出版发行: 中国电力出版社

地址: 北京市三里河路 6 号 邮政编码: 100044

电话: (010) 68362602 传真: (010) 68316497

印 刷: 航远印刷有限公司

开本尺寸: 185mm × 260mm 印 张: 27 字 数: 680 千字

书 号: ISBN 978-7-5083-7962-3

版 次: 2009 年 2 月北京第 1 版

印 次: 2009 年 2 月第 1 次印刷

印 数: 0001—4000 册

定 价: 45.00 元 (含 1DVD)

敬 告 读 者

本书封面贴有防伪标签, 加热后中心图案消失
本书如有印装质量问题, 我社发行部负责退换

版 权 专 有 翻 印 必 究

F

前言

Foreword

Hi, 大家好, 我是朱印宏, 自从去年 9 月份推出《CSS 商业网站布局之道》以来, 该书在广大读者之间产生了不小的轰动, 获得了 Web 前端设计圈内很多专业人士的推崇, 并连续几个月登上华储、卓越、当当网站排行榜。《CSS 商业网站布局之道》是我对 CSS 技术的思考和实践的一部分, 其实在那本书推出之后, 我仍然感觉有很多东西没有写入, 或者没有写透, 于是继续思考 CSS 的核心技术问题以及它的发展前景。可以这样说, 《CSS 商业网站布局之道》一书是我对于 CSS 深层次的理论思考, 而本书则是我在前一本书的基础上继续探索 CSS 在实践中的应用, 少了几分理论色彩, 却多了几分实践的摸索。

记得我曾经在《CSS 商业网站布局之道》一书中说过: “用 CSS 容易, 难的是全部都用 CSS”。虽然 CSS 语言比较好学习, 但是如果完全使用 CSS 技术来控制页面的表现仍然会很麻烦。于是很多人不理解, 经常在群里或“坛子”里与我辩论。例如, 蓝色经典论坛的版主 linxz 就曾与我在群里为此争论了一番:

样吧 21:55:47

用 CSS 容易, 难的是全部都用 CSS。

linxz™ 21:56:07

为什么要全部用 CSS 呢?

linxz™ 22:56:56

CSS 是用在该用的地方。你这样说太广泛了。

样吧 22:57:27

CSS 是用在该用的地方, 自然。

样吧 22:57:44

难道不需要 CSS, 非要硬塞一个样式吗?

linxz™ 22:59:39

难道用 JS, 你一定要用 CSS?

样吧 22:59:57

我没有这样说呀。

样吧 22:60:26

你非要硬塞, 不理解?

linxz™ 22:61:18

☺

大家有自己的想法这是可以理解的, 但是我这么说也有其事实根据: 第一, 枯燥的 CSS 代码远没有传统表格布局用得随心所欲; 第二, 为了使设计的页面能够兼容不同浏览器, 看似简单的 CSS 代码往往会把你弄得精疲力竭; 第三, 没有完美的可视化辅助工具, 你似乎感觉自己总处于石器时代的那种刀耕火种之中。

当然我说的是相对的, 随着 CSS 技术的不断普及和推广, 各种成熟的可视化工具大量涌现,

以及各种浏览器都支持 W3C 标准，也许到那时候，我再说这话可能就有不合时宜了。

我曾经还在《CSS 商业网站布局之道》主题网站 (<http://www.css8.cn/csslayout/index.htm>) 写过这么一句话：**CSS 的精髓是布局，而不是样式，布局是需要缜密的结构分析和设计。**这句话也是我对于 CSS 思考而来的，有很多专业人士比较欣赏，当然也不乏有反对意见。记得 CSS 森林群主曾经围绕我这句话写了一篇《中国式 WEB 标准》的思考文章 (<http://www.cssforest.org/blog/index.php?id=93>)，虽然他的文章主题是谈论标准设计问题，但是这句对于 CSS 思考的话还是引起大家的共鸣。

我一直认为样式和布局应该是两个不同的概念，样式是指具体的应用效果，而布局则是网页结构的表现问题，它不仅仅是局部效果问题，还应涉及网页的框架结构。如果说学习 CSS 仅希望它能够美化一下字体样式，或者设计漂亮的段落版式，那么我认为你还没有把握住 CSS 的精髓，CSS 的精髓应该是网页的整体呈现方面，换句话说就是如何使用 CSS 把一个 HTML 结构给完美的支撑起来。说起来可能很容易，如何撑起网页的结构呢？这就涉及 CSS 核心技术问题，以及如何兼容不同浏览器问题，这远比传统表格布局中信手涂鸦要复杂许多。正是基于这样的思考，我才萌生了继续写作本书的念头。

从去年开始，国外就开始了 CSS 框架技术的试验和实践，也出现了几个比较有名的 CSS 框架技术，国内设计师还没有完全嗅到 CSS 框架的香味，唯有淘宝团队在今年开始进行 CSS 框架的摸索和实践。当然在本书上市之后，也许会有更多大型商业网站去实践 CSS 框架技术，考虑到这种技术是一种发展趋势，所以我也用了一章篇幅对此展开讲解，很多东西都是我对于 CSS 框架的思考和探索，相信未来几年内它一定会成为 CSS 技术的一个热门话题。

给本书读者的话

非常感谢您购买本书。在乱花渐欲迷人眼的 CSS 分类书架中，你能够挑选本书作为学习 CSS 技术的首选书籍，是我的荣幸。当然在准备阅读本书之前，我还是要叮嘱几句，希望你能够买得物有所值。

第一，请记住不怕慢就怕站。我不希望读者一目十行的来阅读本书，在群里经常会听到读者说 3 天翻阅一本图书的故事。当然这里可能会有两种情况：一是书的含金量确实不高，所以只能一目十行；二是你的 CSS 技术水平功力不浅，一目十行是为了搜索自己需要的知识。如果不是这两种情况，真心的建议你还是慢慢阅读本书，我整整花费了五个多月的时间来写作本书，而你仅仅使用一个礼拜的时间来阅读本书，那么里面肯定会有很多金子被埋没了。

第二，请记住不怕笨就怕懒。要说到笨，其实我是最笨的学生，记得最初学习 CSS 时，居然抱住《CSS 参考手册》硬耗了 3 个月。学习最忌讳自大，但是最担心的就是发懒。抽懒筋，就是形象的描绘了人性的弱点，人发懒时，越想懒就还想懒，以至于懒得抽筋。所以初学者最好多动手，对于书中得每行代码都能够动手输入试验一下，并能够有所思考。群里有位咆哮男郎的网友购买《CSS 商业网站布局之道》一书，居然用了 2~3 月的时间把书中每句话都咀嚼了一遍，每行代码都亲自动手输入试验，并帮助挑出了一些纰漏的地方汇集给我，让我非常感动。

第三，请记住不怕错就怕迷信。不要迷信任何真理，要敢于去思考，应能够举一反三，触类旁通。学习本书，不等于本书就是圣经，你要敢于去探索、穷思每个问题，不要被已有的观点束缚你的思想，只有这样你才能够走得更远，超过本书作者的水平。

最后，我把 CSS 聊吧中给初学者的一组话送给你，希望它能够成为你学习 CSS 的助推器。

- 不要被对象、属性、方法等词汇所迷惑，最根本的是先了解最基础知识。
- 不要放过任何一个看上去很简单的小问题，它们往往并不那么简单，或者可以引申出很多知识点，不会举一反三你就永远学不会。
- 知道一点东西，并不能说明你会全部，需要经验积累。
- 看再多的书是学不全的，要多实践。
- 把时髦的技术挂在嘴边，还不如把过时的技术记在心里。
- 学习的最好方法之一就是多练习。
- 在任何时刻都不要认为自己手中的书已经足够了。
- 看得懂的书，请仔细看；看不懂的书，请硬着头皮看。
- 别指望看第一遍书就能记住和掌握什么，请看第二遍、第三遍……
- 请把书上的例子亲手到电脑上实践，即使配套光盘中有源文件。
- 把在书中看到的有意义的例子扩充，并将其切实的运用到自己的工作中。
- 不要漏掉书中任何一个练习，请全部做完并记录下思路。
- 别心急，写代码确实不容易，水平是在不断的实践中完善和发展的。
- 每学到一个难点的时候，尝试着对别人讲解这个知识点并让他理解。你能讲清楚才说明你真的理解了。
- 记录下在和别人交流时发现的自己忽视或不理解的知识点。
- 保存好你做过的所有的源文件，那是你最好的积累之一。
- 对于网络，还是希望大家能多利用一下，你要学会自己找答案，如 Google、百度都是很好的搜索引擎，你只要输入关键字就能找到很多相关资料，别总是等待别人给你希望。
- 浮躁的人容易问：我到底该学什么？别问，学就对了。
- 浮躁的人容易问：CSS 有钱途吗？建议你去抢银行。
- 浮躁的人容易说：我要中文版！我英文不行！——不行？学呀！
- 浮躁的人分两种：只观望而不学的人，只学而不坚持的人。
- 浮躁的人永远不是一个高手，请不要做浮躁的人！

关于光盘的内容

本书赠送 DVD 光盘一张，其中包含：

- 893 分钟的超强视频讲解。视频与本书内容同步，辅助读者随堂、直观地学习本书核心技术。
- 本书用到的素材和网页源文件。
- CSS 完全中文参考手册、HTML 参考 CHM 文件。

本书作者

本书由朱印宏主持编写。同时，参与本书资料整理及编写的还有林小志、袁衍明、常才英、袁祚寿、张敏、袁江、田明学、唐荣华、毛荣辉、卢敬孝、刘玉凤、李坤伟、旷晓军、陈万林、陈锐、钱佩林、苏敬波、冉东林、杨龙贵、张炜、王慧明、涂怀清、卢国才、苏恢定、司成向、胡体清、陈宗亮、徐清银、周秀成、颜昌学、王幼平、冉原洲、李经键、胡厚成等，在此对大家的辛勤工作表示衷心的感谢！

由于时间有限，书中难免会有疏漏和不足之处，恳请广大读者提出宝贵意见。

技术支持

CSS 聊吧论坛: <http://www.css8.cn/css2/> (如果地址无效, 请从样吧首页链接进入)

CSS 技术交流群: 10780586 (群 1)、54430674 (群 2)、23660353 (群 3)、34678498 (群 4)、
23038424 (群 5)、41091270 (群 6)

Dreamweaver 设计交流群: 6879926 (群 1)

出版社网站 (售后服务支持): <http://www.infopower.com.cn>

作者网站 (技术服务支持): <http://www.css8.cn>

作者电邮: zhuyinhong@css8.cn

作者 QQ: 364077976

朱印宏

2008 年 8 月于北京

样吧聊吧
PDG

C 目 录

Contents

前 言

第 1 章 CSS 布局基础 1

- 1.1 认识网页中的 CSS 1
 - 1.1.1 编写符合 CSS 布局的 HTML 结构 1
 - 1.1.2 DOCTYPE (文档类型) 与 CSS 的关系 3
- 1.2 CSS 基本语法和用法 4
 - 1.2.1 CSS 基本语法 4
 - 1.2.2 应用 CSS 样式 5
 - 1.2.3 建立外部样式表 6
 - 1.2.4 在 HTML 中导入外部样式表 7
- 1.3 CSS 的属性和属性值 8
 - 1.3.1 如何记忆 CSS 的属性 8
 - 1.3.2 CSS 的取值单位 9
- 1.4 CSS 常用选择符 10
 - 1.4.1 标签选择符 10
 - 1.4.2 类选择符 11
 - 1.4.3 ID 选择符 13
- 1.5 CSS 高级选择符 15
 - 1.5.1 认识 HTML 文档的树状结构 15
 - 1.5.2 子选择符 17
 - 1.5.3 相邻选择符 18
 - 1.5.4 属性选择符 19
- 1.6 灵活使用 CSS 选择符 23
 - 1.6.1 CSS 选择符分组 23
 - 1.6.2 通配选择符 24
 - 1.6.3 包含选择符 24
 - 1.6.4 伪类和伪元素选择符 26
 - 1.6.5 CSS 选择符的嵌套 27
- 1.7 CSS 的继承性 28
 - 1.7.1 认识 CSS 的继承性 28
 - 1.7.2 CSS 继承性的局限性 29
- 1.8 CSS 样式层叠和选择符优先级 31
 - 1.8.1 CSS 样式表的优先级 31
 - 1.8.2 CSS 样式的优先级 31
 - 1.8.3 CSS 选择符的优先级 31

第 2 章 使用 Dreamweaver 实现 CSS

布局 33

- 2.1 认识 Dreamweaver 的 CSS 布局工作流 33
 - 2.1.1 增强 CSS 功能的【属性】面板 33
 - 2.1.2 【页面属性】对话框 35
 - 2.1.3 【CSS 样式】面板 36
 - 2.1.4 属性表 37
- 2.2 操作【CSS 样式】面板 38
 - 2.2.1 新建样式 38
 - 2.2.2 定义样式的属性 39
 - 2.2.3 应用样式 43
- 2.3 以 CSS 技术为核心的设计视图 45
 - 2.3.1 强大的 CSS 设备类型 45
 - 2.3.2 可视化助理 47
- 2.4 使用 DW 实现 CSS 布局实战 49
 - 2.4.1 使用 Dreamweaver CS3 构建页面结构 49
 - 2.4.2 使用【页面属性】统一基本样式 51
 - 2.4.3 使用【CSS 样式】面板定制页面基本布局 54

第 3 章 听话的文字和版式 56

- 3.1 看懂 Dreamweaver CS3 提供的模板 56
 - 3.1.1 善于利用示例页 56
 - 3.1.2 挖掘标准网页布局模板的价值 60
- 3.2 你的文字听话吗 62
 - 3.2.1 国际雅虎和中国雅虎 62
 - 3.2.2 选择好单位让你的文字更听话 64
 - 3.2.3 设计网页字体大小配置方案 65
 - 3.2.4 设计与字体大小相关联的网页布局 68
- 3.3 留心, 不要让你的文字“露怯” 70
 - 3.3.1 不要以为多此一举 71
 - 3.3.2 让网页字体显示更灵活 71
 - 3.3.3 别忘了通用字体 72
- 3.4 小家伙, 你向哪儿看齐 73
 - 3.4.1 文本对齐 74
 - 3.4.2 布局居中 74

3.4.3 布局元素向左、右对齐·····	75	5.2.3 模拟按钮样式·····	139
3.4.4 文本和元素垂直对齐·····	76	5.2.4 使用图像设计超链接样式·····	141
3.4.5 绝对定位元素居中显示·····	78	5.2.5 设计具有扩展性的超链接样式·····	143
3.4.6 混合结构中对齐处理·····	80	5.3 构建导航条列表框架及其基本样式·····	144
3.5 咱们比比个吧·····	81	5.3.1 为什么使用项目列表来构建导航条·····	145
3.5.1 什么样的行高更合适阅读·····	82	5.3.2 搭架结构合理的导航列表·····	145
3.5.2 行高、边距和伪行高·····	83	5.3.3 使用定义列表搭架导航列表·····	147
3.6 让网页看起来和颜悦色·····	85	5.3.4 控制项目列表的样式·····	147
3.6.1 在 CSS 中使用颜色·····	85	5.3.5 有趣的项目符号·····	148
3.6.2 CSS 布局与配色·····	86	5.3.6 自定义图片项目符号·····	149
第 4 章 扮靓网页的天使——图像·····	89	5.3.7 通过背景图像来定义项目符号·····	150
4.1 选择：前景图像和背景图像·····	89	5.3.8 使用行内显示法设计项目列表水平显示·····	151
4.1.1 前景图像·····	89	5.3.9 使用浮动法设计项目列表水平显示·····	153
4.1.2 背景图像·····	90	5.3.10 设计水平浮动列表的超链接样式·····	153
4.1.3 如何选用前景图像和背景图像·····	91	5.3.11 解决浮动溢出问题·····	154
4.1.4 走出迷失的图像 URL·····	91	5.3.12 使用定位法设计项目列表水平显示·····	155
4.2 图像边框和阴影·····	93	5.4 设计导航菜单·····	155
4.2.1 定义边距·····	93	5.4.1 淡雅的垂直导航菜单·····	156
4.2.2 巧设边距·····	94	5.4.2 巧妙的水平导航菜单·····	158
4.2.3 为图像增加阴影·····	96	5.4.3 背景图像在导航菜单中的应用·····	161
4.2.4 为 img 元素定义默认阴影样式·····	98	5.4.4 实用的多级菜单·····	164
4.2.5 设计晶莹剔透的水印·····	99	5.4.5 时尚的滑动门菜单·····	167
4.3 图文混合排版·····	101	第 6 章 更专业的表格和表单·····	170
4.3.1 单行图文定位·····	101	6.1 用好表格·····	170
4.3.2 单行图文对齐·····	103	6.1.1 正确使用表格·····	170
4.3.3 图文环绕·····	106	6.1.2 优化数据表格的结构·····	174
4.3.4 调整图文环绕的间距·····	108	6.1.3 设计表格样式·····	176
4.3.5 不规则图文环绕·····	108	6.2 表格样式设计实战·····	178
4.4 隐形的翅膀——背景图像·····	111	6.2.1 设计表格行样式·····	179
4.4.1 控制背景图像的重复显示·····	111	6.2.2 设计表格列样式·····	180
4.4.2 背景图像水平平铺应用示例·····	112	6.2.3 设计鼠标经过时表格行的样式·····	180
4.4.3 背景图像垂直平铺应用示例·····	113	6.3 用好表单·····	182
4.4.4 精确定位背景图像·····	114	6.3.1 认识表单·····	182
4.4.5 固定背景图像在浏览器中的位置·····	117	6.3.2 设计表单样式·····	184
4.4.6 如何增加多个背景图像·····	118	6.4 表单样式设计实战·····	186
4.4.7 设计更酷的圆角·····	120	6.4.1 设计高亮显示当前表单样式·····	187
4.5 背景图像布局应用——伪列布局·····	124	6.4.2 设计图标样式的表单效果·····	188
第 5 章 活泼的超链接和导航菜单·····	132	6.4.3 设计易用性表单效果·····	190
5.1 轻松掌握超链接样式·····	132	第 7 章 浮动的网页布局·····	194
5.1.1 怪异的伪类·····	132	7.1 CSS 的“水立方”模型·····	194
5.1.2 超链接的四种状态·····	133	7.1.1 盒模型·····	194
5.1.3 为超链接绑定不同的样式·····	134	7.1.2 盒模型解疑·····	195
5.1.4 为页面定义不同的超链接样式·····	135	7.1.3 盒模型的外边距·····	196
5.2 花枝招展的超链接样式·····	137		
5.2.1 为超链接定义普通样式·····	137		
5.2.2 避免影响版面晃动的超链接样式·····	138		

7.1.4	行内元素的外边距	197
7.1.5	块状元素的外边距	198
7.1.6	浮动元素的外边距	198
7.1.7	绝对定位元素的外边距	199
7.1.8	盒模型的内边距	200
7.1.9	盒模型的边框	202
7.1.10	盒模型的宽和高	203
7.2	构建符合标准的网页结构	205
7.2.1	是内容决定结构，还是表现决定结构	205
7.2.2	选择好结构标签	207
7.2.3	研究禅意花园的网页结构	207
7.3	CSS 浮动布局基础	211
7.3.1	认识网页布局类型	211
7.3.2	准备有趣的模块浮动游戏	211
7.3.3	①②③顺序的水平布局	212
7.3.4	③②①顺序的水平布局	212
7.3.5	②①③顺序的水平布局	213
7.3.6	③①②顺序的水平布局	214
7.3.7	①-②③结构布局	216
7.4	CSS 浮动布局高级技术	217
7.4.1	探究浮动元素的空间大小	217
7.4.2	探究浮动元素的移动位置	218
7.4.3	探究浮动元素的环绕关系	220
7.4.4	浮动与清除	222
7.4.5	浮动元素包含其他对象的问题及其解决方法	224
7.4.6	浮动元素被其他对象包含的问题及其解决方法	225
7.4.7	浮动布局中元素垂直间距问题以及解决方法	227
7.4.8	浮动布局中元素水平间距问题以及解决方法	228
7.4.9	浮动布局中浮动元素并列错位问题以及解决方法	230
7.5	CSS 浮动布局实战	232
7.5.1	深红色咖啡馆	232
7.5.2	阳光灿烂喜洋洋	237

第 8 章 精确的网页布局 243

8.1	CSS 精确布局的定位原理	243
8.1.1	认识 position	243
8.1.2	静态定位	244
8.1.3	绝对定位	245
8.1.4	相对定位	245
8.1.5	固定定位	248
8.2	定位布局中参照物和坐标系	250
8.2.1	绝对定位的参照物	250

8.2.2	绝对定位的坐标系	253
8.2.3	绝对定位的相对论	255
8.2.4	相对定位参照物和坐标系	256
8.3	定位布局中元素层叠处理	257
8.3.1	定位元素的层叠顺序	258
8.3.2	定位元素与文档流的层叠关系	259
8.3.3	定位元素的层叠包含关系	260
8.4	CSS 定位布局实战	262
8.4.1	米老鼠卡通画册	262
8.4.2	禅意花园展室	267
8.4.3	蓝色的多瑙河	272

第 9 章 兼容性网页布局 278

9.1	智能的 IE 条件语句	278
9.1.1	认识 IE 条件语句	278
9.1.2	IE 条件语句基本用法	279
9.1.3	IE 条件语句在实践中应用	282
9.2	探析 IE 浏览器渲染网页布局的特性	286
9.2.1	认识 IE 浏览器的渲染特性——Layout	286
9.2.2	操控 IE 元素的 Layout 特性	287
9.2.3	网页元素的 Layout 特性	287
9.2.4	行内元素的 Layout 特性	290
9.2.5	与 Layout 特性相关的 IE 解析 Bug 和解决方法	290
9.2.6	能够更好兼容不同浏览器	293
9.2.7	Layout 元素与包含浮动元素的关系	293
9.2.8	Layout 元素与相邻浮动元素的关系	295
9.2.9	Layout 元素与列表元素的关系	296
9.2.10	Layout 元素与定位元素的关系	299
9.3	浏览器兼容技术解析	300
9.3.1	你的浏览器符合标准吗	300
9.3.2	兼容 IE 6 版本浏览器	302
9.3.3	兼容 IE 5 系列版本浏览器	303
9.3.4	兼容 IE 7 版本浏览器	305
9.3.5	兼容 FF 等标准浏览器	305
9.4	深入剖析 CSS 层叠布局	306
9.4.1	认识 CSS 层叠包含框	306
9.4.2	CSS 层叠包含框嵌套关系	308
9.4.3	IE 在解析 CSS 层叠关系时的 Bug	309
9.4.4	探析 CSS 层叠负值以及 IE 存在 Bug	310
9.5	IE 浏览器常见 Bug 及其解决方法	312
9.5.1	超链接设计中存在的 Bug	312
9.5.2	IE 元素的内容与背景分离显示的 Bug	313
9.5.3	IE 6 躲躲猫 Bug	314
9.5.4	IE 6 多余字符 Bug	315

9.6 兼容性网页布局实战	316
9.6.1 三列等高布局新法及其非 IE 存在的 Bug 和解决方法	316
9.6.2 三列浮动布局中 IE 的垂直空隙 Bug 及其兼容方法	320
第 10 章 网页结构化布局与实施	323
10.1 设计网页基本结构	323
10.1.1 网页基本结构的设计思路	323
10.1.2 设计符合 SEO 的网页结构	326
10.2 单列版式布局	327
10.2.1 单列版式的布局结构和设计方法	328
10.2.2 单列固定宽度版式设计	330
10.2.3 单列固定宽度版式实施	332
10.2.4 江南水乡意象画布局设计	334
10.2.5 江南水乡意象画布局实战	337
10.3 单列液态框架版式布局	340
10.3.1 使用 CSS 设计框架结构	340
10.3.2 使用 CSS 设计单列液态布局	343
10.4 两列版式布局	346
10.4.1 两列版式的布局结构和设计方法	347
10.4.2 两栏浮动版式中兼容固定和自适应宽度的设计方法和探索	349
10.4.3 两栏浮动版式中自适应宽度的设计方法和探索	351
10.4.4 两栏浮动版式中兼容性宽度的设计方法和探索	353
10.4.5 两栏固定版式的设计方法和探索	354
10.4.6 黑色理想网页布局实施实战	355
10.5 三列版式布局	361
10.5.1 三列版式的布局结构和设计方法	361
10.5.2 解析 Dreamweaver CS3 提供的绝对定位布局及存在问题	362
10.5.3 三列液态版式的设计新方法和探索	364
10.5.4 三列固定版式设计方法和探索	366
10.5.5 一列液态两列固定版式设计方法和探索	367
10.5.6 二列液态一列固定版式设计方法和探索	369
10.5.7 山鹰之美网页布局实战	370

第 11 章 CSS 框架设计及其案例解析	376
11.1 CSS 框架设计	376
11.1.1 什么是 CSS 框架	376
11.1.2 分析 CSS 框架的优缺点	377
11.1.3 如何设计 CSS 框架	378
11.2 CSS 框架的抽象性及其设计	378
11.2.1 认识 CSS 的类样式	379
11.2.2 CSS 类样式的实施策略	379
11.2.3 通用 CSS 类样式设计	380
11.2.4 CSS 框架中元素默认样式设计	383
11.2.5 CSS 框架中其他类型选择符样式的处理	386
11.3 CSS 框架的可维护性及其设计	387
11.3.1 CSS 框架的注释技巧及其可读性	387
11.3.2 CSS 框架中类名和 ID 名的可读性	388
11.3.3 CSS 框架代码的排版格式应简明直观	389
11.3.4 框架内样式代码的结构顺序应符合一定的阅读习惯	391
11.4 CSS 框架的可扩展性和兼容性及其设计	391
11.4.1 CSS 的继承性及其利用	391
11.4.2 CSS 框架的继承与包含关系及其应用	393
11.4.3 通过样式表的分类、组织实现 CSS 框架的可扩展性	395
11.4.4 CSS 框架的可兼容性	397
11.5 CSS 框架案例解析	397
11.5.1 解析 Elements CSS 框架	398
11.5.2 解析 Blueprint CSS 框架	401

第 12 章 购物网站结构与布局实施	409
12.1 设计购物网站的基本结构	409
12.1.1 网站内需分析	409
12.1.2 设计网站的基本结构	411
12.1.3 完善语义化结构	413
12.2 使用 CSS 完成基本结构的布局	415
12.2.1 实施网页基本布局	415
12.2.2 设计滑动门菜单	417
12.2.3 设计圆角区域	419

CSS 布局基础

CSS 是 Cascading Style Sheets 的简写，中文翻译的意思为层叠样式表。在网页设计中，CSS 与 HTML、JavaScript 并列为网页前端设计的三种基本语言。其中，CSS 负责设计网页的表现效果，HTML 负责构建网页的基本结构，JavaScript 负责开发网页的交互效果。

CSS 布局是 CSS 语言的高级使命，它与我们常说的网页样式略有不同，布局涉及网页整体结构的呈现，而样式则过多关注局部效果的显示。由于 CSS 布局涉及 CSS 的核心技术，Web 设计师需要吃透 CSS 布局原理，能灵活统筹规划，并能兼容主流浏览器，因此其开发的难度也是可想而知的。本章是全书的基础，也是你准备学习 CSS 语言的基础，所以在系统讲解 CSS 基础的同时，也会略有侧重，避免在简单知识上重复叙述。

1.1 认识网页中的 CSS

如今当你随意查看一个页面中的源代码时，会看到类似下面的代码段：

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

```
.....  
</style>
```

和

```
<link href="style.css" rel="stylesheet" type="text/css" />
```

这些正是本书要讲解的 CSS 语言在网页中的“家”，更准确地讲就是 CSS 源代码在网页中分隔符和引用指针。

CSS 语言不需要编译，也不需要特殊的处理，你只要把它们放在<style>和</style>标签之间，或者单独存储在一个文本文件之中，然后保存到扩展名为.css 的文件，最后利用<link>标签链接或导入到网页中即可，使用就是这么简单。

1.1.1 编写符合 CSS 布局的 HTML 结构

学习本书之前你需要有初步的 HTML 语言基础，并能够认识常用的 HTML 标签。其实这对于任何一个初学网页制作的读者来说，不应该存在什么障碍了。但是如果你习惯于传统网页制作方式，用表格来切图，然后生成网页，那么本节内容还需要你认真地阅读。

网页都是由标签组成的，这正如一张地图上的各种提示性标签，它标明了每个点所指示的方位、地理名称等。把这些标签串连在一起，才能够在你的大脑中形成一幅清晰的地图。而网页标签不仅要指明网页内容在网页结构中的位置关系和显示效果，更重要的是还要标明网页内

容的意思，即所谓的 HTML 标签的语义性。

例如，在下面这个简单的网页文档中，每个标签都包含一个字符串，但是它们所表示的意思是不相同的。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>网页标签</title>
</head>
<body>
<h2>标题</h2>
<p>段落</p>
<ul>
<li>列表项目</li>
<li>列表项目</li>
</ul>
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
<td>数据表</td>
<td>数据表</td>
</tr>
</table>
</body>
</html>
```

针对上面的 HTML 文档结构，从语义角度简单说明如下：

(1) <html>标签表示网页文档的意思，也可以说是 HTML 文档源代码的分隔符。

(2) <head>标签表示网页头部信息。

(3) <body>标签表示网页主体信息。

在<body>标签中包含了不同语义的标签。

1) <h2>标签表示二级标题信息。

2) <p>标签表示段落文本信息。

3) 和标签表示项目列表信息。

4) <table>、<tr>和<td>标签表示数据表信息等。

但是以表格为核心的传统网页布局中，设计师似乎对于各种标签所代表的语义特征并不感兴趣。设计师所追求的目标就是如何把网页设计得更精美漂亮。结果<table>、<tr>和<td>标签就被供奉为网页布局的三剑客，标题和段落文本全部被放置在<p>标签中，然后通过属性设置来控制标题和段落文本的显示效果。

如今表格布局不再使用了，CSS 布局被广泛使用，这是很自然的事情，但是对于习惯于传统布局思维，并且熟悉操作传统布局工具的设计师来说，所肩负的担子很重。

学习使用 CSS 进行布局，能够帮助你设计出更简洁的 HTML 结构。因为你不再需要在 HTML 文档中使用各种修饰性的标签和属性。整个页面只剩下构建页面结构的框架标签以及表达一定语义的服务性标签。整个文档的源代码变得简单了许多。

如果你还在继续使用如下标签，建议今后尽量不要再使用了：

(1) 不使用标签来设置文本样式。

(2) 不使用和<i>标签来设置文字粗体和斜体。如果确实需要使用可以选用和

`` 标签来表示粗体和斜体。

(3) 不使用 `<table>` 标签进行布局, `<table>` 仅负责显示表格式的数据。

(4) 不要在 `<body>` 标签内通过各种属性来定义网页基本属性, 如 `background` (背景图像)、`bgcolor` (背景色)、`text` (文本)、`link` (链接)、`leftmargin` (左页边距)、`topmargin` (顶部页边距) 等。

(5) 尽量不使用 `
` 换行标签, 传统布局中习惯使用 `
` 和空格符号来设计多段文本, 换行排版等, 现在可以借助各种语义化标签和 CSS 来控制, 基本上不用再使用 `
` 了。

对于标准网页设计的初学者来说, 我们希望你一步步扎实学习, 但是下面几个简单的、符合标准的标签及其用法一定要先记住, 然后在此基础上不断去积累。

(1) `<div>`: 网页结构化标签, 它负责划分网页结构, 并负责包含不同的模块。

(2) ``: 行内文本和对象包含标签, 它负责修饰行内对象样式。

(3) `<p>`: 负责管理段落文本内容的组织和管理。

(4) `<h>`: 标题标签, 根据重要性分为 6 个级别, `h1` 表示一级标题, 一般页面中只最好只包含一个一级标题。

(5) ``、`` 和 ``: 项目列表标签, 主要负责网页内同类信息的列表。

1.1.2 DOCTYPE (文档类型) 与 CSS 的关系

如果在 Dreamweaver CS3 中新建网页文档, 你将会在新文档的首行看到 DOCTYPE 声明。DOCTYPE 是文档类型的简写, 它定义当前文档的基本类型。

其实 DOCTYPE 只是一组机器可读的规范, 虽然中间包含了文件的 URL, 但浏览器不会去读取这些文件, 仅用于识别, 然后决定以什么样的规范去执行页面中的代码。其具体格式如下:

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

DTD 表示文档类型的定义, 它们定义 XML、XHTML 和 HTML 的特定的某一个版本中可以有什么, 不可以有什么, 在载入网页的时候, 浏览器会用既定的声明规范去检查页面的内容, 是不是有效, 然后采取相应的措施与编码解释文档中的代码。

那它们是如何工作的? 我们不去深究, 这对于初学者来说未免有点困难。但是你必须知道在标准网页布局中这是非常必要的。缺少它, 页面就可能出现一些异常的情况。因此, 当你开发符合标准的页面时, 要必须为 HTML 文档指定一种 DOCTYPE, 否则 CSS 渲染可能会出错。

XHTML 1.0 提供了三种 DTD 声明可供选择, 简单说明如下。

(1) 过渡型 (Transitional): 要求非常宽松的 DTD, 它允许用户继续使用 HTML4.01 的标签, 但是要符合 XHTML 的写法。完整代码如下:

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

(2) 严格型 (Strict): 要求严格的 DTD, 用户不能使用任何表现层的标签和属性, 如 `
`。完整代码如下:

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

(3) 框架型 (Frameset): 专门针对框架页面设计使用的 DTD, 如果你的页面中包含有框架, 就需要采用这种 DTD。完整代码如下:

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

一般情况下，建议读者选择过渡型文档类型，这种 DTD 比较宽松，也比较容易通过 W3C 的代码校验，比较适合目前国内开发环境和大多数开发人员的水平。当然你的目标是向着严格型文档类型去努力。

除此而外，你还会发现在符合标准的 HTML 文档中，<html> 标签还会包含如下代码：

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="UTF-8">
```

xmlns 属性是 XHTML namespace 的缩写，中文翻译为名字空间。XHTML 是 HTML 向 XML 过渡的标识语言，它需要符合 XML 文档规则，因此也需要定义名字空间。因为 XHTML1.0 不能自定义标识，所以它的名字空间都相同，就是“http://www.w3.org/1999/xhtml”。对于 HTML 文档来说我们只要照抄代码就可以了。

1.2 CSS 基本语法和用法

CSS 是一种高级的、弱类型的语言，与 HTML 一样都是一种标识语言，在任何文本编辑器中都可以打开或编辑。因此，不管你有没有基础，初次接触 CSS 时会感到很简单，而一旦完全使用 CSS 来布局页面时，你又会发现它很复杂。因此，读者必须在开始学习就要打下扎实的基础，下面我们就来介绍 CSS 的基本语法和简单用法。

1.2.1 CSS 基本语法

CSS 的语法单元是样式，每个样式包含两部分内容：选择符和声明（或称为规则），如图 1.1 所示。

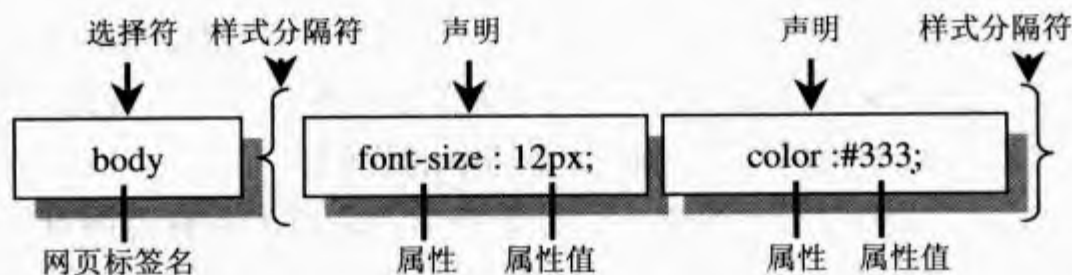


图 1.1 CSS 样式基本格式

(1) 选择符 (Selector): 选择符告诉浏览器该样式将作用于页面中哪些对象，这些对象可以是某个标签、所有网页对象、指定 Class 或 ID 值等。浏览器在解析这个样式时，根据选择符来渲染对象的显示效果。

(2) 声明 (Declaration): 声明可以增加一个或者无数个，这些声明命令浏览器如何去渲染选择符指定的对象。声明必须包括两部分：属性和属性值，并用分号来标识一个声明的结束，在一个样式中最后一个声明可以省略分号。所有声明被放置在一对大括号内，然后整体紧邻选择符的后面。

(3) 属性 (Property): 属性是 CSS 提供的设置好的样式选项。属性名由一个单词或多个单词组成，多个单词之间通过连字符相连，这样能够更直观地表示属性所要设置样式的效果。

(4) 属性值 (Value): 属性值用来显示属性效果的参数。它包括数值和单位，或者关键字。例如，定义网页字体大小为 12 像素，字体颜色为深灰色，我们则可以设置如下样式：


```
body{font-size: 12px; color: #CCCCCC;}
```

多个样式可以并列在一起，不需要考虑如何进行分隔，例如，我们再定义段落文本的背景色为紫色，则可以在上面样式基础上定义如下样式：

```
body{font-size: 12px; color: #CCCCCC;}p{background-color: #FF00FF;}
```

由于 CSS 语言忽略空格（除了选择符内部的空格外），因此我们可以利用空格来美化 CSS 源代码，则上面代码可以进行如下美化：

```
body {
    font-size: 12px;
    color: #CCCCCC;
}
p { background-color: #FF00FF; }
```

这样在阅读 CSS 源代码时就一目了然了。既方便阅读，又容易维护。

任何语言都需要注释，HTML 使用“<!-- 注释语句 -->”来进行注释，而 CSS 使用“/* 注释语句 */”来进行注释。例如，对于上面可以进行如下注释：

```
body { /* 页面基本属性 */
    font-size: 12px;
    color: #CCCCCC;
}
/* 段落文本基础属性 */
p { background-color: #FF00FF; }
```

1.2.2 应用 CSS 样式

CSS 的样式必须放置在特定类型的文件、标签或属性中，否则是无效的，浏览器会视其为普通的字符串，而不对其进行解析。CSS 代码一般可以放置在三个地方，详细说明如下：

(1) 直接放在标签的 style 属性中，例如，

```
<span style="color:red;">红色字体</span>
<div style="border:solid 1px blue; width:200px; height:200px;"></div>
```

这样当浏览器解析这些标签时，检测到该标签包含有 style 属性，于是就调用 CSS 引擎来解析这些样式码，并把效果呈现出来。

这种通过 style 属性直接把样式码放在标签内的做法被称之为行内样式，因为它与传统网页布局中在标签增加属性的设计方法没有什么两样，这种方法实际上还没有真正把 HTML 结构和 CSS 表现分开进行设计，因此不建议使用。除非为页面中个别元素设置某个特定样式效果而需要单独进行定义。

(2) 把样式代码放在<style>标签内，例如：

```
<style type="text/css">
body { /* 页面基本属性 */
    font-size: 12px;
    color: #CCCCCC;
}
/* 段落文本基础属性 */
p { background-color: #FF00FF; }
</style>
```

在设置<style>时应该指定 type 属性，告诉浏览器该标签包含的代码是 CSS 源代码。这样当浏览器遇到<style>之后，会自动调用 CSS 引擎进行解析。

这种 CSS 应用方式也被称为网页内部样式，如果仅为一个页面定义 CSS 样式时，使用这种方法比较高效，且管理方便。但是在一个网站中或在多个页面之间引用时，使用这种方法会产生代码冗余，不建议使用，而且多页管理样式也是不经济的。

内部样式一般放在网页的头部区域，目的是让 CSS 源代码早于页面源代码下载并被解析，这样避免当网页信息下载之后，由于没有 CSS 样式渲染而使页面信息无法正常显示。

(3) 把样式放置在单独的文件中，然后使用<link>标签或者@import 关键字导入。

这样当浏览器遇到这些代码时，会自动根据它们提供的 URL 把外部样式表文件导入到页面中并进行解析。这种应用样式的方式也被称为外部样式。一般网站都采用外部样式来设计网站的表现层问题，以便统筹设计 CSS 样式，并能够快速开发和高效管理。

1.2.3 建立外部样式表

外部样式表也就是一个文本文件，其扩展名为.css。当把不同的样式复制到一个文本文件中后，另存在扩展名为.css 文件中，则它就是一个外部样式表。如图 1.2 所示就是禅意花园的外部样式表。



图 1.2 CSS 禅意花园外部样式表文件

外部样式表文件与内部样式表没有什么两样，都是由无数个样式组成的。你也可以在外部样式表文件顶部定义 CSS 源代码的字符编码。例如，下面代码定义样式表文件的字符编码为中文简体。

```
@charset "gb2312";
```

当你在 Dreamweaver CS3 中新建或打开一个 CSS 文件时，可以借助可视化操作快速进行设置或修改。方法是：选择【修改】|【页面属性】菜单命令，打开【页面属性】对话框（如图 1.3 所示），从中可以修改 CSS 文件的字符编码类型。

如果不设置 CSS 文件的字符编码，就可以保留默认设置，浏览器则会根据 HTML 文件的字符编码来解析 CSS 代码。

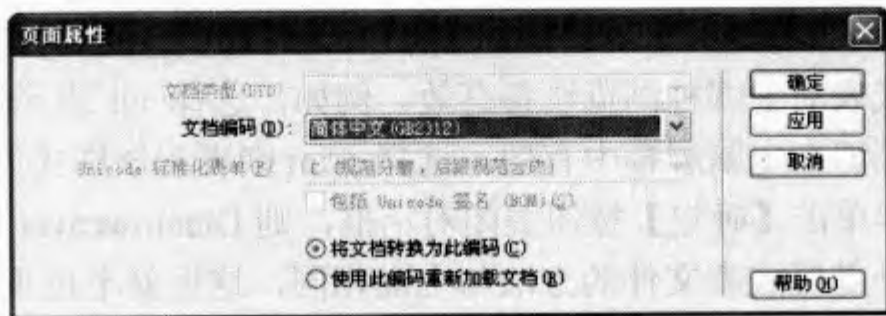


图 1.3 CSS 文件的【页面属性】对话框

1.2.4 在 HTML 中导入外部样式表

外部样式表文件可以通过两种方法导入到 HTML 文档中。

方法一，利用<link>标签导入。例如：

```
<link href="001.css" rel="stylesheet" type="text/css" />
```

其中 href 属性设置外部样式表文件的地址，可以是相对地址，也可以是绝对地址。rel 属性定义该标签关联的是样式表标签，type 属性定义文档的类型，即为 CSS 文本文件。

因为不同浏览器的要求也不同，一般在定义<link>标签时，应该显式设置这 3 个属性，其中 href 是必须设置属性。


方法二，在<style>标签内使用@import 关键字导入外部样式表文件。例如：

```
<style type="text/css">
@import url("001.css");
</style>
```

在@import 关键字后面，利用 url()函数包含具体的外部样式表文件的地址。

如果你感觉这些方法比较麻烦，你不妨使用 Dreamweaver CS3 提供的可视化操作来快速完成。具体操作步骤如下：

第 1 步，启动 Dreamweaver CS3，选择【窗口】|【CSS 样式】菜单命令，打开【CSS 样式】面板（如图 1.4 所示）。

第 2 步，单击【CSS 样式】面板底部的【附加样式表】按钮（），打开【链接外部样式表】对话框（如图 1.5 所示）。在【文件/URL】文本框设置外部样式表文件的路径，在【添加为】按钮组中选择是【链接】按钮还是【导入】按钮。其中【链接】选项表示使用<link>标签导入，CSS 样式表文件中的源代码没有被直接导入带 HTML 文档中，还是保留链接关系，需要时由浏览器根据 URL 地址读取样式码。而【导入】选项则表示使用@import 关键字导入外部样式表文件。

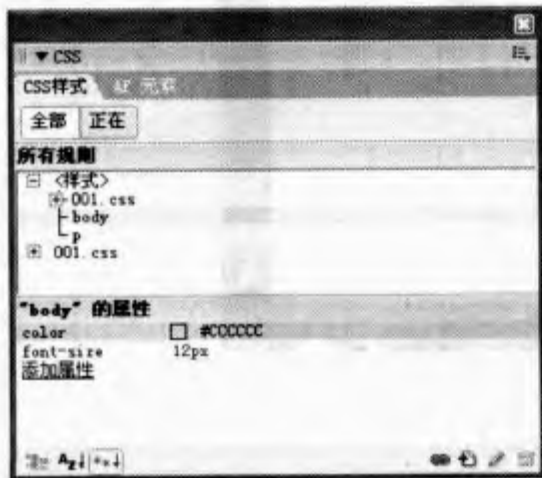


图 1.4 【CSS 样式】对话框



图 1.5 【链接外部样式表】对话框

第3步，在【链接外部样式表】对话框的【媒体】下拉列表中选择样式表适用的媒体类型，所谓媒体类型就是样式表对于哪种浏览设备有效。例如，选择 all 表示适用所有媒体类型，而选择 screen 则表示仅能够在电脑屏幕中有效，选择 print 则表示该样式只能够适应打印使用。

第4步，设置完毕单击【确定】按钮关闭对话框，则 Dreamweaver CS3 会自动帮助导入外部样式表文件。修改外部样式表文件的方法与上面相同，这里就不再重复介绍。

1.3 CSS 的属性和属性值

CSS 的属性众多，其中 CSS 2.0 版本包含了 150 多个属性，这些属性被分为不同的类型，如字体属性、文本属性、边框属性、边距属性、布局属性、定位属性、打印属性等。对于初学者来说，初学 CSS 的最大障碍是如何熟悉并掌握这些属性的使用。关于 CSS 属性的详细列表和用法可以参阅本书附赠的 CSS 参考手册。

1.3.1 如何记忆 CSS 的属性

这里给读者一点小建议：

不要急于记住 CSS 的所有属性或想一下吃透它们的用法，更不能机械记忆，如果使用背英语单词的方法来记忆，效果势必会很差。

最佳的方法是边学习边记忆，步步为营，在实践中逐个突破。当你学习网页排版时，不妨集中精力把字体和文本属性研究一下。当学习网页布局时，不妨再研究与盒模型和布局相关的几个属性。

记忆这些属性时，一定要结合实践，不断去尝试并举一反三。只有这样你才能够完全掌握 CSS 所有属性，并能够熟练应用。例如，当你准备学习 CSS 布局时，不妨先集中精力把与 CSS 盒模型相关的属性记住，此时你可以绘制一个图（如图 1.6 所示）。

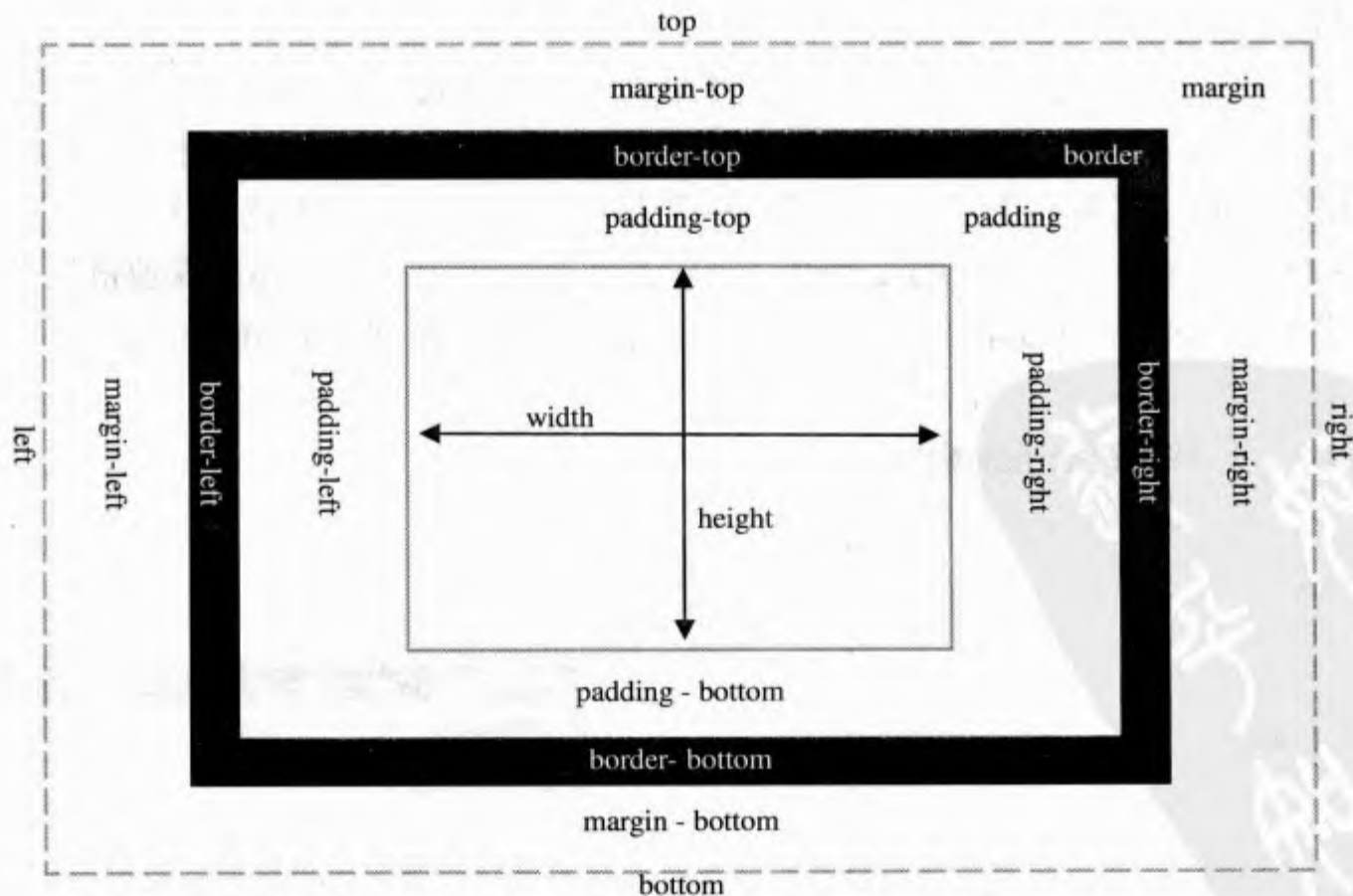


图 1.6 与 CSS 盒模型相关的属性

CSS 属性的名称比较有规律，且名称与意思紧密相连，如同象形文字，根据意思记忆属性名称是一个不错的方法。

CSS 盒模型讲的就是网页中任何元素都会显示为一个矩形形状，它包括外边距、边框、内边距、宽和高等，用英文表示就是：`margin`（外边距，或称为边界）、`border`（边框）、`padding`（内边距，或称为补白）、`height`（高）和 `width`（宽），盒子还有 `background`（背景）。

外边距按方位又可以包含 `margin-top`、`margin-right`、`margin-bottom`、`margin-left` 共 4 分支属性，分别表示顶部外边距、右侧外边距、底部外边距和左侧外边距。

同样的道理，内边距也可以包含 `padding-top`、`padding-right`、`padding-bottom`、`padding-left`、`padding` 属性。边框可以分为边框类型、粗细和颜色，因此可以包含 `border-width`、`border-color` 和 `border-style` 属性，这些属性又可以按四个方位包含很多属性，例如，`border-width` 属性又分为 `border-top-width`、`border-right-width`、`border-bottom-width`、`border-left-width` 和 `border-width` 属性。

如果你顺着这个思路可以很快记住与盒模型相关的几十个属性。同样的方法，你还可以轻松记住其他类型的 CSS 属性。

1.3.2 CSS 的取值单位

学习 CSS 属性时，读者可以直接参阅手册，因此我们重点讲解一下 CSS 取值的单位问题，这个问题对于初学者来说也是很容易混淆的。CSS 的取值单位主要包括长度、颜色和 URL。

长度单位可以包括绝对单位和相对单位：

- 绝对单位又包括：厘米（`cm`）、毫米（`mm`）、英寸（`in`）、点（`pt`）、派卡（`pc`）。
- 相对单位又包括：`em`（`em`，元素的字体的高度）、`ex`（`x-height`，字母“x”的高度）、`px`（像素，相对于屏幕的分辨率）。

绝对单位一般不建议使用，它不适合网页显示的特殊环境。而 `em` 和 `px` 是目前使用频率最高的两个单位。在第 3.2.2 节中我们将更详细的讲解 CSS 绝对和相对单位问题。

`em` 单位是根据字体大小来决定单位的大小。在显示时可通过比较当前字体大小来确定。字体越大，所对应的 `em` 和 `ex` 单位取值也就越大。

`px` 单位是根据显示器上的像素点大小来决定单位的大小。而像素点的大小是根据屏幕分辨率来决定的，这也是最常用的取值单位。

百分比总是相对于另一个对象来说的，一般可以根据父元素（或者上级元素）的相同属性值作为参照值。使用百分比，首先应写符号部分，这个符号可以是“+”（正号），表示正长度值，也可以是“-”（负号），表示负长度值。如果不写符号，那么默认值是“+”。在符号后紧接着是一个数值，符号后面可以输入任意值，但是由于在某些情况下，浏览器不能处理带小数的百分比，因此最好不用带小数的百分比，最后在数值后面增加“%”符号表示百分比单位，例如，45%。

定义颜色可以有 4 种形式：

- `#rrggbb`，如 `#00cc00`。
- `#rgb`，如 `#0c0`。
- `rgb(x,x,x)`，其中 `x` 是一个 0~255 之间的整数，如 `rgb(0,204,0)`。
- `rgb(y%,y%,y%)`，其中 `y` 是一个 0.0~100.0 之间的整数，如 `rgb(0%,80%,0%)`。

另外，还可以指定颜色的名称，如 `black`（纯黑）、`silver`（浅灰）、`navy`（深蓝）、`blue`（浅

蓝)、green (深绿)、lime (浅绿)、teal (靛青)、aqua (天蓝)、maroon (深红)、red (大红)、purple (深紫)、fuchsia (品红)、olive (褐黄)、yellow (明黄)、gray (深灰)、white (亮白) 等。

Uniform Resource Locator (简称 URL, 统一资源定位) 表示超链接的路径, 主要包括绝对路径和相对路径。其中相对路径又可以分为相对根目录的路径及和相对文档的路径。

绝对路径包含的是精确位置, 一般创建站外超链接时必须使用绝对路径, 这样不管当前文件的位置如何变化, 该链接都是有效的。

相对于根目录的路径总是从当前站点的根目录开始, 一般使用 “/” 告诉服务器从根目录开始。例如, /dreamweaver/index.html 将链接到站点根目录 dreamweaver 文件夹的 index.html 文件。

相对于文档的路径是指和当前文档所在的文件夹相对的路径。例如:

- 1) 文档 test.swf 在文件夹 flash 中, 它指定的就是当前文件夹内的文档。
- 2) ../test.swf 指定的则是当前文件夹上级目录中的文档。
- 3) /test/test.swf 指定是 flash 文件夹下 test 文件夹中的 test.swf 文档。

相对于文档的路径是最简单的路径, 在创建与文档相对的路径之前必须保存文件, 因为在没有定义起始点的情况下, 与文档相对的路径是无效的。

1.4 CSS 常用选择符

如果你使用过 Photoshop 或其他图像编辑软件, 相信你对图像选取工具并不陌生。可以说在 Photoshop 中所有操作都是建立在选取操作基础之上。例如, 选框工具、魔棒工具、路径选取工具、通道选取工具、色彩选取命令等, 所有这些工具都无不例外的方便用户精确选取图像区域, 以实现更准确的图像编辑。

CSS 中的选择符类似于 Photoshop 中各种选取工具或命令, 灵活使用这些 CSS 选择符是使用 CSS 控制网页的基础。本节将详细讲解 CSS 基本选择符, 它也是 CSS 学习的重点和难点。

1.4.1 标签选择符

HTML 文档都是由很多标签通过一定的规则编织而成的, 我们也可以把这些标签称为网页元素。标签选择符正是确定要定义样式的网页元素对象。

例如, 在下面这个样式中声明了 p 元素的基本样式, 该样式将应用于网页中所有段落文本, 它将段落内的字体大小定义为 12 像素, 字体颜色为红色。

```
<style type="text/css">
p {
    font-size:12px;          /* 字体大小为 12 像素 */
    color:red;               /* 字体颜色为红色 */
}
</style>
```

标签选择符不需要重新命名, 直接引用 HTML 特定标签的名称即可 (如图 1.7 所示)。有时候我们可以把标签选择符称为类型选择符, 类型选择符规定了网页元素在页面中默认显示样式。因此, 类型选择符可以快速、方便地控制页面基本样式。

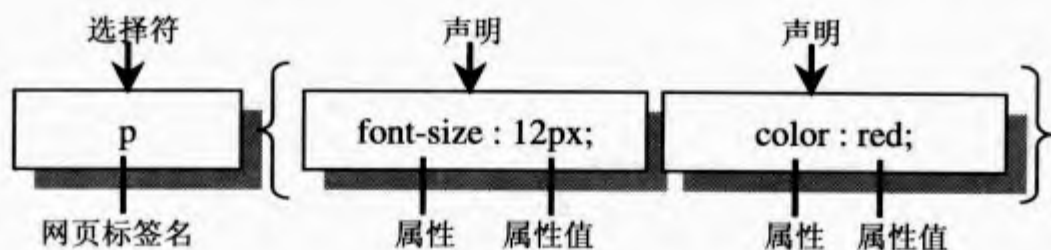


图 1.7 标签选择符

网页设计师可以在定制页面样式时，利用标签选择符来统一网页常用元素的基本样式。标签选择符在 CSS 中是使用率最高的一类选择符，且也容易管理，因为它们都是与网页元素同名的。

1.4.2 类选择符

标签选择符虽然很方便，但是也存在很多缺陷。因为每个标签选择符所定义的样式不仅仅影响某一个标签，而且影响页面中所有同名的标签。如果希望同一个标签在网页的不同位置显示不同的样式，使用这种方法定义的样式就存在很多弊端。怎么办？此时不妨使用类选择符。

类 (Class) 选择符就是为不同元素但是拥有相同的显示样式而定义的。例如，在下面这个页面中包含 3 段文本，通过标签选择符为所有段落文本的字体大小定义为 12 像素，字体颜色为红色。

```
<style type="text/css">
p {
    font-size:12px;          /* 字体大小为 12 像素 */
    color:red;               /* 字体颜色为红色 */
}
</style>
<p>问君能有几多愁，恰似一江春水向东流。</p>
<p>剪不断，理还乱，是离愁。别是一般滋味在心头。</p>
<p>独自莫凭栏，无限江山，别时容易见时难。流水落花春去也，天上人间。</p>
```

但是，我们现在希望第 2 段文本的字体大小为 18 像素，这时就可以使用类选择符。假设定义一个 18 像素大小的字体类：

```
.font18px {
    font-size:18px;
}
```

请注意，类选择符必须以一个点 (.) 前缀开头，然后跟随一个自定义的类名（如图 1.8 所示）。然后在页面中第 2 段段落标签中引用 font18px 类样式。引用类样式时，可以使用 class 属性来实现，HTML 所有元素都支持该属性。

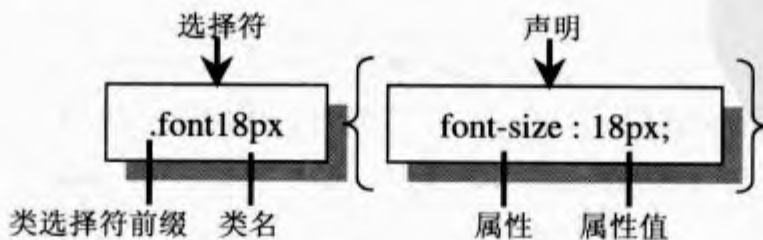


图 1.8 类选择符

```
<p>问君能有几多愁，恰似一江春水向东流。</p>
<p class="font18px">剪不断，理还乱，是离愁。别是一般滋味在心头。</p>
<p>独自莫凭栏，无限江山，别时容易见时难。流水落花春去也，天上人间。</p>
```

这时如果在浏览器中预览则显示如图 1.9 所示。你可以看到第 2 段文本被单独放大显示。

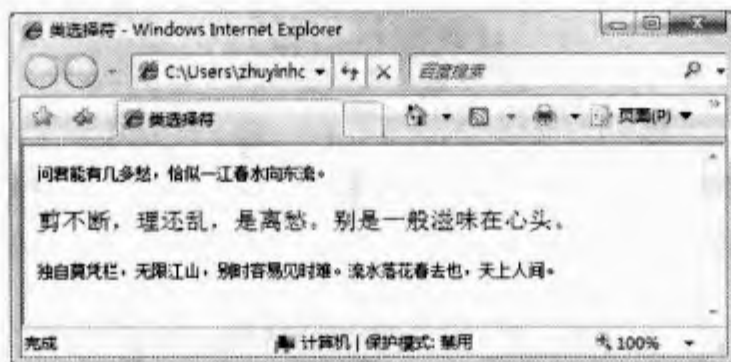


图 1.9 类选择符应用效果

类选择符可以精确控制页面中某个具体的元素对象，而不管这个对象是属于什么类型的标签，同时一个类样式可以在多个标签中被引用。因此类选择符除拥有标签选择符的影响广泛性之外，还具备精确控制页面标签样式的优势，是设计师常用的选择符之一。

类选择符虽然比标签选择符在使用上更精确，但是你必须把类引用到具体的标签上时才有效，任何标签在没有设置 class 属性时，所定义的类型样式是无效的，因此类在一定程度上给设计师的使用带来麻烦。

在自定义类名时，只能够使用字母、数字、下划线（_）和连字符（-），类名首字符必须以字母开头，否则无效。另外，类名是区分大小写的，所以类 font18px 和类 Font18px 是属于两个不同的类。

标签的 class 属性可以被设置多个类，这为类的广泛使用奠定了基础。如果一个标签只能够引用一个类，那么这与定义标签选择符没有什么区别了。例如，在下面这个示例中定义了 3 个类：font18px、underline 和 italic。然后在段落文本中分别引用这些类，其中第 2 段文本标签引用了 3 个类（如图 1.10 所示）。

```
<style type="text/css">
p { /* 段落默认样式 */
    font-size:12px;                /* 字体大小为 12 像素 */
    color:red;                     /* 字体颜色为红色 */
}
.font18px { /* 字体大小类 */
    font-size:18px;                /* 字体大小为 18 像素 */
}
.underline { /* 下划线类 */
    text-decoration:underline;     /* 字体修饰为下划线 */
}
.italic { /* 斜体类 */
    font-style:italic;             /* 字体样式为斜体 */
}
</style>
<p class="underline">问君能有几多愁，恰似一江春水向东流。</p>
<p class="font18px italic underline">剪不断，理还乱，是离愁。别是一般滋味在心头。</p>
<p class="italic">独自莫凭栏，无限江山，别时容易见时难。流水落花春去也，天上人间。</p>
```

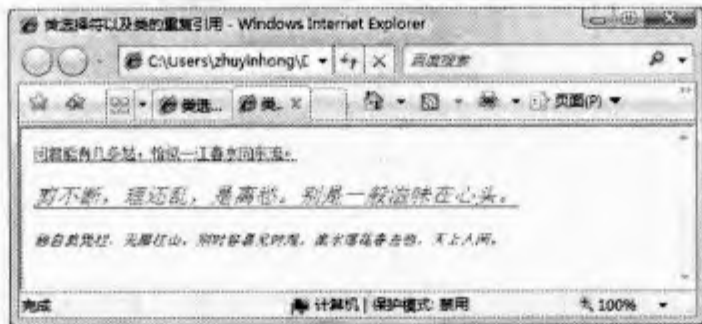


图 1.10 多类引用应用效果

如果把标签与类捆绑在一起来定义选择符，则可以限定类的使用范围，指定该类仅适用于特定的标签范围内。它的用法是在标签的后面紧跟一个类，组成一个指定类范围的复合选择符，如图 1.11 所示。

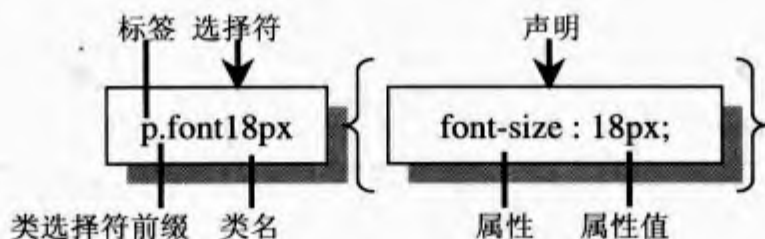


图 1.11 指定类选择符

例如，在下面这个示例中定义了 3 个样式，第 1 个样式声明所有段落文本的字体大小为 12 像素，在第 2 个样式中定义一个 font18px 类，声明字体大小为 18 像素，在第 3 个样式中声明 font18px 类在段落文本中显示为 24 像素，显示效果如图 1.12 所示。

```
<style type="text/css">
p { /* 段落样式 */
    font-size:12px;                /* 字体大小为 12 像素 */
}
.font18px { /* 类样式 */
    font-size:18px;                /* 字体大小为 18 像素 */
}
p.font18px { /* 指定段落的类样式 */
    font-size:24px;                /* 字体大小为 24 像素 */
}
</style>
<div class="font18px">问君能有几多愁，恰似一江春水向东流。</div>
<p class="font18px">剪不断，理还乱，是离愁。别是一般滋味在心头。</p>
<p>独自莫凭栏，无限江山，别时容易见时难。流水落花春去也，天上人间。</p>
```

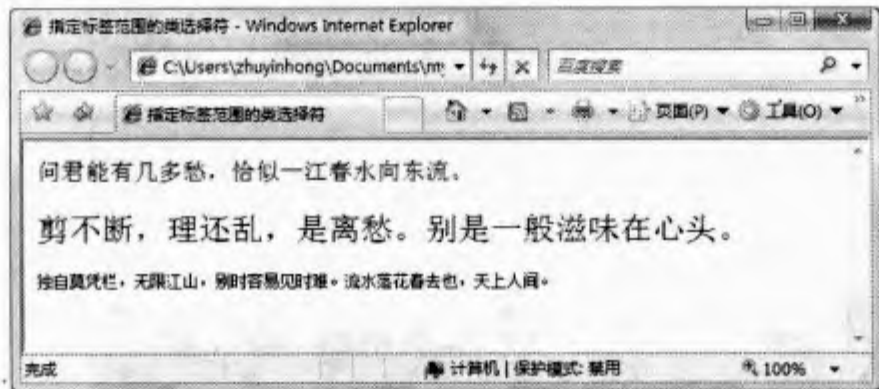


图 1.12 指定类选择符的应用效果

通过为类选择符指定标签范围，能够更准确地控制页面元素的样式，避免类样式对于所有元素的影响。这也是设计师最喜欢使用的一种组合选择符的方式。

1.4.3 ID 选择符

ID 是英文 IDentity 的缩写，它表示身份标识号码的意思，在网络上一一般指用户账号，但是在 Web 设计中一般指定标签在 HTML 文档中的唯一编号。JavaScript 等脚本语言通过这个 ID 属性值来捕获和控制页面中每一个元素。

而对于 CSS 来说则通过 ID 属性值为不同元素定义特定的样式。从这点来分析，ID 选择符只能够在 HTML 页面使用一次，它是与标签选择符和类选择符作用范围相反的一个选择符。一

般设计师通过 ID 选择符来定义 HTML 框架结构的布局效果, 因为 HTML 框架元素的 ID 值都是唯一的。

ID 选择符必须以井号 (#) 前缀开始, 然后是一个自定义的 ID 名, 用法如图 1.13 所示。



图 1.13 ID 选择符

例如, 下面示例定义了一个盒子, 为该盒子固定宽和高, 并设置背景图像, 以及边框和内边距大小, 显示效果如图 1.14 所示。

```
<style type="text/css">
#box { /* ID 样式 */
    background:url(images/bg1.gif) center bottom; /* 定义背景图像并居中、底部对齐 */
    height:200px; /* 固定盒子的高度 */
    width:400px; /* 固定盒子的宽度 */
    border:solid 2px red; /* 边框样式 */
    padding:20px; /* 增加内边距 */
}
</style>
<div id="box">问君能有几多愁, 恰似一江春水向东流.</div>
```

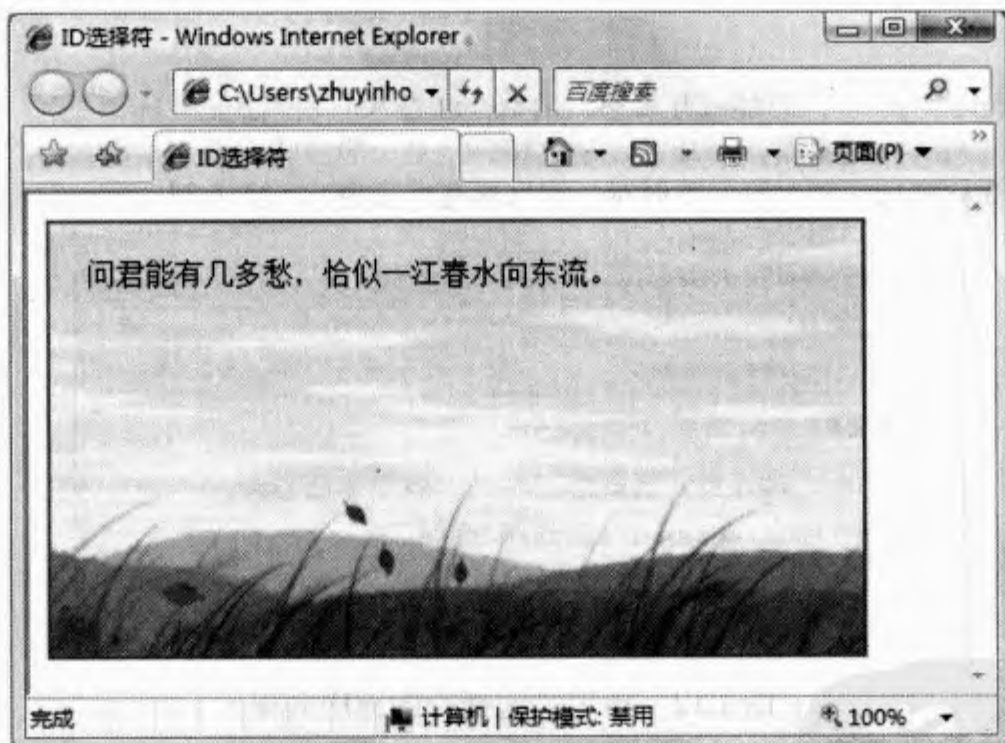


图 1.14 ID 选择符的应用效果

一个 ID 选择符所定义的样式可以被多处引用, 也就是说在一个 HTML 文档中一个 ID 值可以多处使用。虽然 CSS 能够容忍这种做法, 但是 JavaScript 等脚本遇到这种情况就会出现错误, 所以建议读者在定义 ID 属性值时, 应该保证 ID 值在文档中的唯一性。同时在一个 ID 属性中, 不能够设置多个 ID 值, 这与 Class 有所不同。

与 Class 用法一样, 在 HTML 文档中每个元素都拥有 ID 属性, ID 属性值的命名规则与 Class 命名规则相同。

那么如何确定使用类选择符和 ID 选择符呢? 具体说明如下:

- (1) 对于网页结构问题, 一般建议使用 ID 选择符来定义。
- (2) 对于重复出现的样式, 可以考虑使用类选择符来进行提炼。
- (3) 当 ID 选择符和类选择符的样式发生冲突时, ID 选择符的样式要优先于类选择符定义的样式。

另外, 我们也可以为 ID 选择符指定标签范围。此时你可能认为 ID 选择符已经针对页面中某个特定标签定义样式, 还有必要给它指定范围吗? 答案是否定的, 采用这种方法真实目的是提高该样式的优先级。定义指定 ID 选择符的语法结构如图 1.15 所示。



图 1.15 指定 ID 选择符

针对上面示例我们可以在 ID 选择符前面增加一个 div 标签, 这样 div#box 选择符的优先级会大于 #box 选择符的优先级。在同等条件下, 浏览器会优先解析 div#box 选择符定义的样式。

```
<style type="text/css">
div#box { /* ID 样式 */
    background:url(images/bg1.gif) center bottom; /* 定义背景图像并居中、底部对齐 */
    height:200px; /* 固定盒子的高度 */
    width:400px; /* 固定盒子的宽度 */
    border:solid 2px red; /* 边框样式 */
    padding:20px; /* 增加内边距 */
}
</style>
<div id="box">问君能有几多愁, 恰似一江春水向东流。</div>
```

1.5 CSS 高级选择符

标签选择符、类选择符和 ID 选择符是 CSS 的三大基本选择符, 也是最常用的类型。但是如果仅仅掌握它们的使用还是不够的。读者还需要掌握高级选择符的使用, 如子选择符、相邻选择符和属性选择符等。

1.5.1 认识 HTML 文档的树状结构

CSS 高级选择符是建立在 HTML 文档结构之上, 在讲解这些高级选择符之前, 我们先来简单了解 HTML 文档的树状结构。

在任何网页文档中, 所有标签都是建立在树状结构基础上从而形成一个相互关联、类似树状的结构。在脚本语言中 (如 JavaScript) 这种结构被称为文档结构模型 (DOM, Document Object Model)。`<html>` 标签 (或称为 html 元素) 是所有其他标签的祖先, 网页中所有其他标签都被包含在 `<html>` 标签中, 也可以形象地称之为树的根节点。

`<html>` 标签包含两个基本的 `<head>` 和 `<body>` 标签。这时我们可以称 `<html>` 标签是 `<head>` 和 `<body>` 标签的父级元素, `<head>` 和 `<body>` 标签是 `<html>` 标签的子元素。而 `<head>` 与 `<body>` 标签是并列显示的相邻标签, 也可以称它们为兄弟元素。

提示

很多时候标签与元素都表示同样的意思，即它们都表示 HTML 文档中节点名称。在没有特殊说明的情况下，本书也会把标签和元素混合使用。例如，对于 h2 标签，我们可以称之为 h2 元素，两者意思相同。

例如，在下面这个简单的页面中包含了<html>、<head>、<meta>、<style>、<body>、<h2>、<div>和标签。它们之间的结构关系可以使用如图 1.16 所示的树状结构图来表示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>HTML 文档树状结构</title>
<style type="text/css">
div#box {
    background:url(images/bg1.gif) center bottom;
    height:200px;
    width:400px;
    border:solid 2px red;
    padding:20px;
}
.font14px {
    font-size:14px;
}
</style>
</head>
<body>
<h2>HTML 文档树状结构</h2>
<div id="box"><span class="font14px">问君能有几多愁，恰似一江春水向东流。</span></div>
</body>
</html>
```

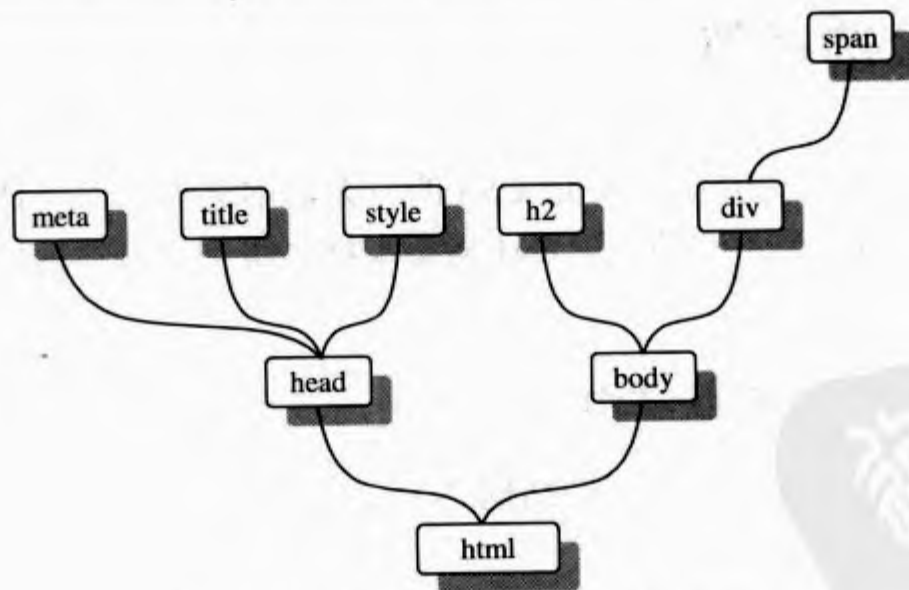


图 1.16 HTML 文档树状结构示意图

在如图 1.16 所示的树状结构示意图中，span 元素是 div 元素的子元素（或称为子对象、子标签），div 元素是 body 的子元素，body 元素是 html 的子元素。反过来，html 元素是 body 的父元素，body 元素是 div 的父元素，而 div 元素是 span 的父元素。

同时，html 元素是 body、div 和 span 元素的祖先，同样 body 元素是 div 和 span 元素的祖先。而 h2 和 div 元素可以称为相邻元素，它们之间是兄弟关系。但是 title、style 元素与 h2、div 元

素不是兄弟关系，因为它们是属于不同的父元素，不过它们之间可以是并列关系。

1.5.2 子选择符

所谓子选择符就是指定父元素所包含的子元素的样式。子选择符使用尖角号 (>) 来表示，如图 1.17 所示。

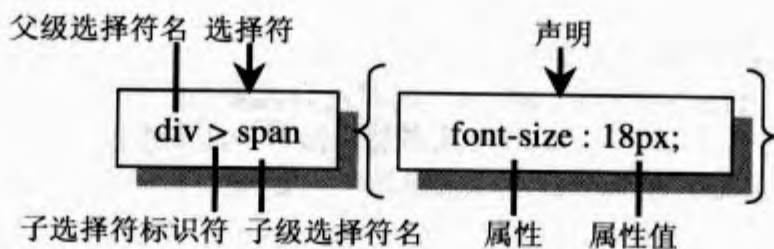


图 1.17 子选择符

在下面示例中我们先定义所有 span 元素的字体大小为 12 像素，然后再利用子选择符来定义所有 div 元素包含的子元素 span 的样式为 24 像素，显示效果如图 1.18 所示。

```
<style type="text/css">
span { /* span 元素的默认样式 */
    font-size:12px;                      /* 增加内边距 */
}
div > span { /* div 元素包含的 span 子元素的默认样式 */
    font-size:24px;                      /* 增加内边距 */
}
</style>
<h2><span>HTML 文档树状结构</span></h2>
<div id="box"><span class="font24px">问君能有几多愁，恰似一江春水向东流。</span></div>
```

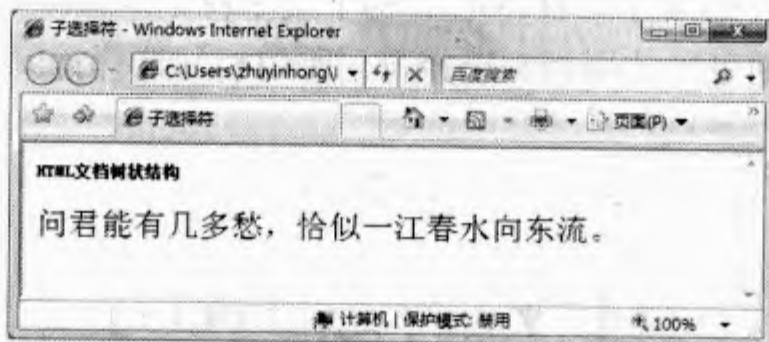


图 1.18 子选择符应用效果

从上面演示效果图可以看到，包含在 div 元素内的子元素 span 被定义了字体大小为 24 像素。通过这种方式，我们可以准确定义 HTML 文档某个或一组子元素的样式，而不再需要为它们定义 ID 属性或者 Class 属性。

当然我们也可以使用 ID 值或 Class 值来定义子选择符。例如，在下面这个示例中我们分别使用不同的方式定义 3 个子选择符：“div > span”表示 div 元素包含的所有 span 子元素的样式，“div > .font24px”表示 div 元素包含的所有命名 font24px 类的子元素，“#box > .font24px”表示 #box 元素包含的类名为 font24px 的所有子元素的样式，演示效果如图 1.19 所示。

```
<style type="text/css">
span {
    font-size:12px;
}
div > span {
    font-size:16px;
```

```

}
div > .font24px {
    font-size:20px;
}
#box > .font24px {
    font-size:24px;
}
</style>
<h2><span>HTML 文档树状结构</span></h2>
<div id="box"><span class="font24px">问君能有几多愁，恰似一江春水向东流。</span></div>
<div><span class="font24px">问君能有几多愁，恰似一江春水向东流。</span></div>
<div><span>问君能有几多愁，恰似一江春水向东流。</span></div>
    
```

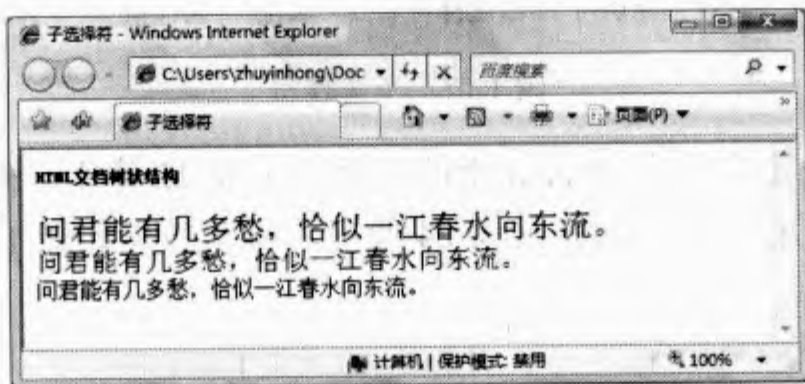


图 1.19 子选择符演示效果

IE 6.0 及其以下版本浏览器目前不支持子选择符，在使用时应该适当考虑浏览器的兼容性问题（该问题将在后面章节做详细讲解）。

1.5.3 相邻选择符

子选择符是利用父子关系来控制 HTML 结构中某个特定对象或一组子对象。而要通过相邻的兄弟元素来相互控制，则可以使用相邻选择符。所谓相邻选择符就是指定一个元素相邻的下一个元素的样式。

相邻选择符使用加号（+）来表示，如图 1.20 所示。

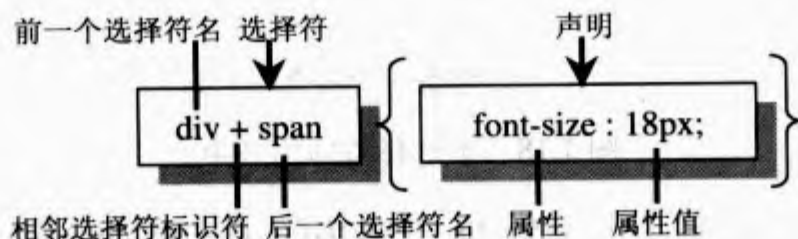


图 1.20 相邻选择符

在下面这个示例中，利用相邻选择符递进控制并列显示的几个元素的显示样式（如图 1.21 所示）。h2 + div 表示标题元素 h2 后面相邻的 div 元素的样式，div + p 表示 div 元素后面相邻的 p 元素的样式，p + div 表示 p 元素后面相邻的 div 元素的样式，而 div + div 表示 div 元素后面相邻的 div 元素的样式。

```

<style type="text/css">
h2 {
    font-size:12px;
}
h2 + div {
    font-size:16px;
}
    
```



```
div + p {  
    font-size:20px;  
}  
p + div {  
    font-size:24px;  
}  
div + div {  
    font-size:28px;  
}  
</style>  
<h2>HTML 文档树状结构</h2>  
<div>问君能有多愁，恰似一江春水向东流。</div>  
<p>问君能有多愁，恰似一江春水向东流。</p>  
<div class="class1">问君能有多愁，恰似一江春水向东流。</div>  
<div>问君能有多愁，恰似一江春水向东流。</div>
```

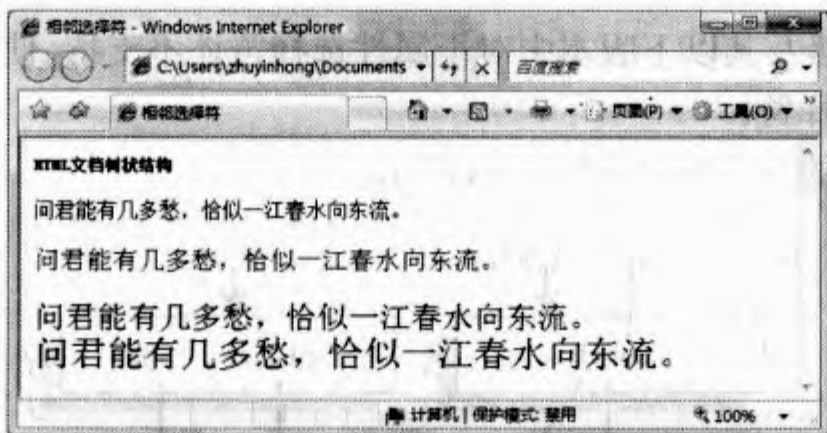


图 1.21 相邻选择符演示效果

对于上面的样式也可以借助 Class 属性值或者 ID 属性值来进行控制，例如，修改上面的样式中相邻选择符的用法如下：

```
<style type="text/css">  
h2 {  
    font-size:12px;  
}  
h2 + div {  
    font-size:16px;  
}  
div + p {  
    font-size:20px;  
}  
p + .class1 {  
    font-size:24px;  
}  
.class1 + div {  
    font-size:28px;  
}  
</style>
```

相邻选择符在 IE 6 及其以下版本中也不被支持，使用时应考虑浏览器兼容性问题。

1.5.4 属性选择符

实际上，ID 选择符和类选择符在本质上与属性选择符类似，它们借助 HTML 文档中的 id 和 class 属性来定位页面中某个或某一类元素。

所谓属性选择符就是利用网页标签包含的属性及其属性值来定义特定元素或一定范围元

素的样式。这与 Dreamweaver CS3 所提供的匹配查找在功能上有点类似。属性选择符一般是一个元素后面紧跟中括号，中括号内是属性或者属性表达式（如图 1.22 所示）。

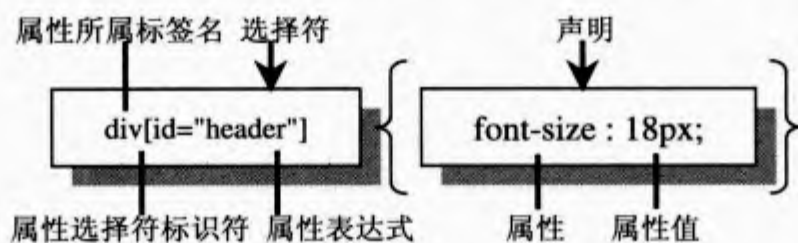


图 1.22 属性选择符

属性选择符比较复杂，功能也相对强大，设计师可以借助属性选择符精确控制页面中的任意一个元素，它如正则表达式一样让很多设计师为之神往。由于属性选择符用法比较复杂，下面我们分类进行讲解。

请注意在 IE 6 浏览器及其以下版本中对于属性选择符还不支持，使用时要注意兼容性处理。

(1) 匹配属性名选择符。

匹配属性名选择符的语法格式如图 1.23 所示。

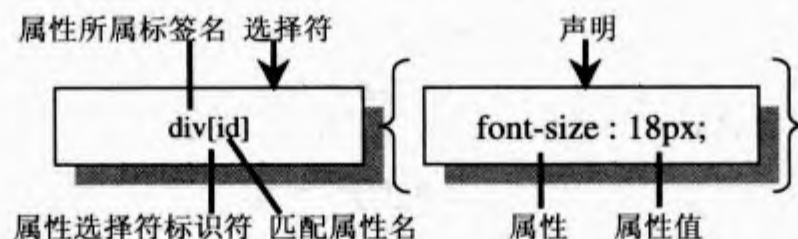


图 1.23 匹配属性名选择符

这是一种简单的属性选择符，它能够为包含指定属性名的所有该类型标签定义样式。例如，在下面这个示例中定义了一个 `div[class]` 属性选择符，该选择符能够为 `div` 元素中设置了 `class` 属性的对象定义样式，而不管 `class` 属性的属性值是什么，显示效果如图 1.24 所示。

```
<style type="text/css">
body {
    font-size:12px;
}
div[class] {
    font-size:24px;
}
</style>
<div class="class1">问君能有几多愁，恰似一江春水向东流。</div>
<p>问君能有几多愁，恰似一江春水向东流。</p>
<div class="class2">问君能有几多愁，恰似一江春水向东流。</div>
<div>问君能有几多愁，恰似一江春水向东流。</div>
```

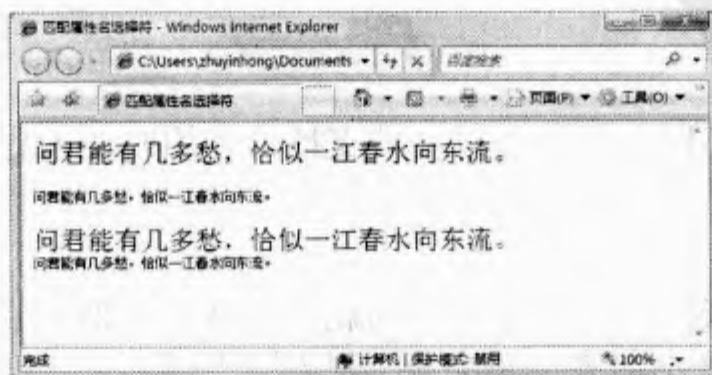


图 1.24 匹配属性名选择符演示效果

上面示例演示了如何匹配 class 属性的选择符，当然你可以设置匹配所有合法属性。例如，在下面这个示例中定义 `img[alt]` 属性选择符，匹配设置了 `alt` 属性的所有图像对象显示红色边框线，显示效果如图 1.25 所示。

```
<style type="text/css">
img {
    width:260px;
}
img[alt] {
    border:solid 2px red;
}
</style>



```

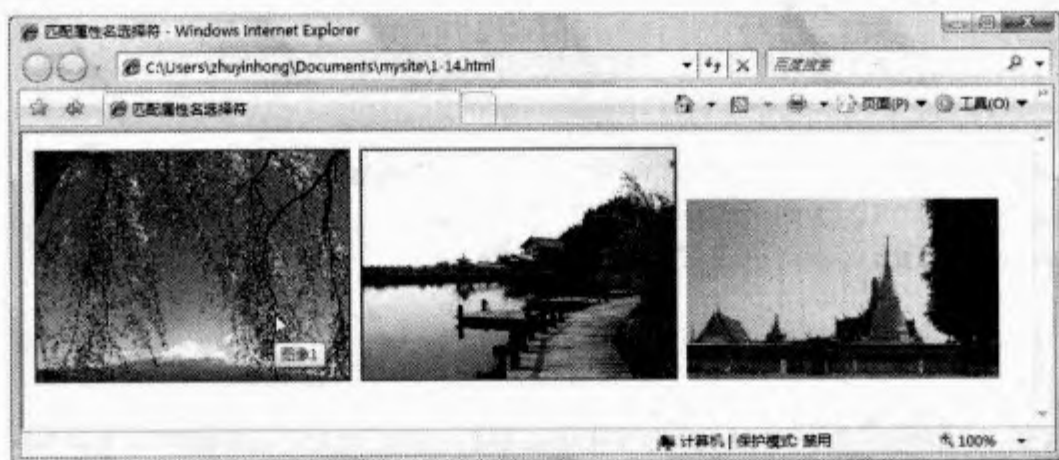


图 1.25 匹配图像中属性名样式演示效果

你还可以设置多个属性名，多个匹配属性名之间分别使用不同的中括号来表示（如图 1.26 所示）。在下面这个示例中在属性选择符中定义了两个匹配属性，则最终显示红色边框线的为第一幅图像。

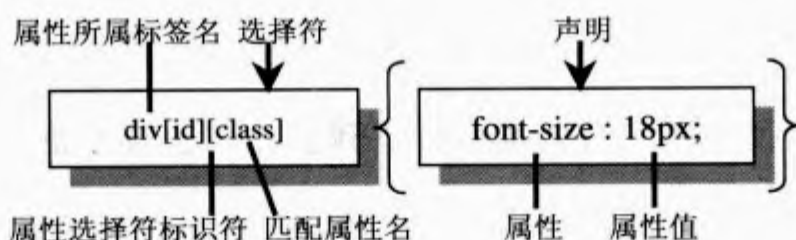


图 1.26 匹配多个属性名选择符

```
<style type="text/css">
img {
    width:260px;
}
img[alt][title] {
    border:solid 2px red;
}
</style>



```

(2) 匹配属性值选择符。

匹配属性值选择符的语法格式如图 1.27 所示。

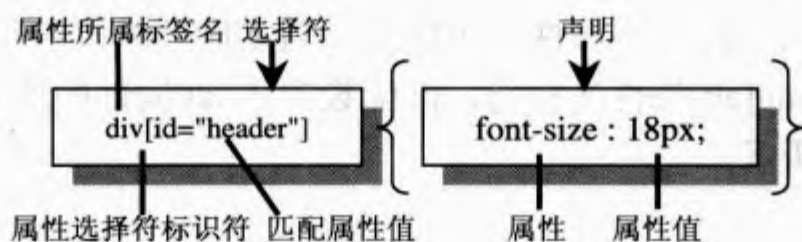


图 1.27 匹配属性值选择符

在指定属性值时应该确保该值被双引号括起来。在下面这个示例中通过指定属性值来为第 1 个图像定义样式，即使第 1 个图像的显示红色边框线（如图 1.28 所示）。

```
<style type="text/css">
img {
    width:260px;
}
img[alt="图像"][title="图像"] {
    border:solid 2px red;
}
</style>



```

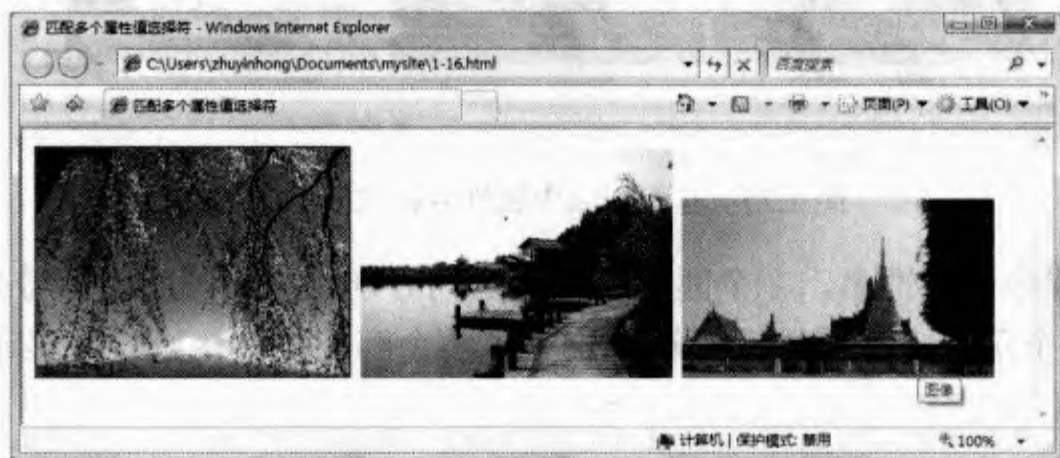


图 1.28 匹配图像中多个属性值的样式演示效果

还可以设置多个匹配属性值，这里就不再举例了，方法与上面相同。

（3）模糊匹配属性值选择符。

这是一类特殊的属性选择符，类似于正则表达式的匹配模式，也是属性选择符中功能最强大的一部分功能，它主要包括如下几种匹配模式：

- 1) `[|=]`（连字符匹配）：以连字符为分隔符，匹配属性值中局部字符串。
- 2) `[~=]`（空白符匹配）：以空白符为分隔符，匹配属性值中局部字符串。
- 3) `[^=]`（前缀匹配）：匹配属性值中起始字符。
- 4) `[$=]`（后缀匹配）：匹配属性值中结束字符。
- 5) `[*=]`（子字符串匹配）：匹配属性值存在的指定字符。

如在下面这个示例中分别定义了 5 个模糊匹配的属性选择符，然后把匹配的 `div` 元素显示出来以测试浏览器是否支持该属性选择符（如图 1.29 所示）。

```
<style type="text/css">
div { /* 隐藏所有 div 元素 */
    display: none;
}
```



```

[class|= "blue"] {      /* 连字符匹配 */
    display: block;
}
[class~="blue"] {      /* 空白符匹配 */
    display: block;
}
[class^="Red"] {       /* 前缀匹配 */
    display: block;
}
[class$="Green"] {     /* 后缀匹配 */
    display: block;
}
[class*="gre"] {       /* 子字符串匹配 */
    display: block;
}
</style>
<div class="red-blue-green">支持[|=] (连字符匹配) 属性选择符</div>
<div class="red blue green">支持[~=] (空白符匹配) 属性选择符</div>
<div class="Red-blue-green">支持[^=] (前缀匹配) 属性选择符</div>
<div class="red-blue-Green">支持[$=] (后缀匹配) 属性选择符</div>
<div class="red-blue-green">支持[*=]: (子字符串匹配) 属性选择符</div>

```

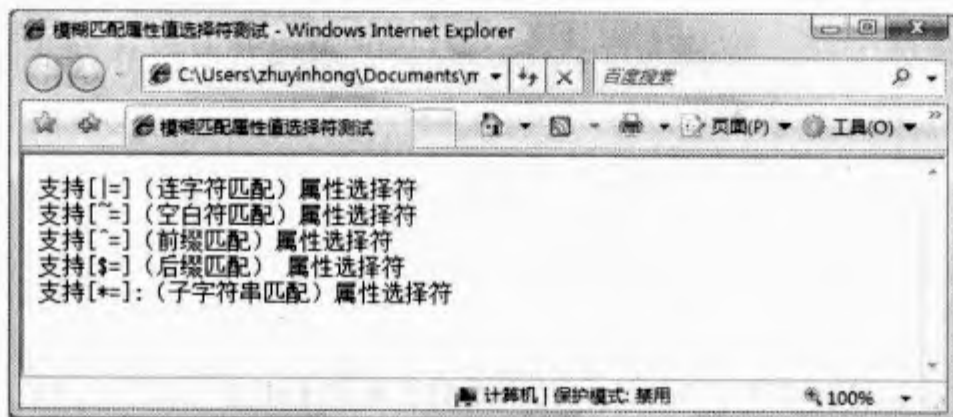


图 1.29 模糊匹配属性选择符演示效果

提示

在上面示例中省略了属性选择符的指定标签选择符, 这时它将匹配任意标签元素。这时可以使用星号 (*) 通配符来指定任意元素。

1.6 灵活使用 CSS 选择符

利用标签选择符和类选择符可以控制网页中众多对象的样式, 而利用 ID 选择符、子选择符和相邻选择符可以精确控制页面中特定对象的样式, 使用属性选择符可以更敏捷、更模糊地控制页面中包含不同属性的对象样式。

但是在设计时, 你也可能感觉这 6 类选择符在使用时还存在很多问题。例如, 对于成批的对象来说, 逐个定义会很麻烦, 而对于一些特殊的对象又无法进行控制, 因此下面我们还需继续介绍如何更灵活的使用 CSS 选择符来进行高效开发。

1.6.1 CSS 选择符分组

在设计过程中, 你可能使用过下面方式来定义标题样式:

```
<style type="text/css">
h1 { font-size:14px; }
h2 { font-size:14px; }
h3 { font-size:14px; }
h4 { font-size:14px; }
h5 { font-size:14px; }
h6 { font-size:14px; }
</style>
<h1>一级标题</h1>
<h2>二级标题</h2>
<h3>三级标题</h3>
<h4>四级标题</h4>
<h5>五级标题</h5>
<h6>六级标题</h6>
```

这样是不是很方便，其实这些标题的样式是相同的，这时我们就可以利用选择符分组来实现：

```
<style type="text/css">
h1, h2, h3, h4, h5, h6 {
    font-size:14px;
}
</style>
```

这些标题元素被分成一组，被称为样式群，以实现快速开发。在网页设计中，我们都习惯使用选择符分组的方式把所有元素的边距清除为 0，例如：

```
html, body,
h1, h2, h3, h4, h5, h6,
p,
table, caption, tr, td, th,
ul, ol, li, dl, dt, dd,
form, legend, fieldset {
    margin: 0;
    padding: 0;
}
```

除了对标签元素进行分组之外，我们还可以给类选择符和 ID 选择符等其他选择符进行分组，其方法也完全相同，多个选择符之间通过逗号进行分隔。

1.6.2 通配选择符

如果 HTML 所有元素都定义相同的样式，那么使用分组方式还是会很麻烦，怎么办？这时我们不妨使用通配选择符，通配选择符是固定的，它使用星号（*）来表示，例如，对于上面的清除边距样式，我们可以使用下面方式来定义：

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

这样是不是很简单，当然使用通配选择符会影响到页面中所有元素的显示效果，在使用时要慎重选择。

1.6.3 包含选择符

有时候我们还希望设置网页头部区域段落文本的字体颜色为黑色，主体区域段落文本的字体颜色为深灰色，而定义脚部区域段落文本的字体颜色为灰色等。对于这种不能够确定要定义

的对象，仅知道要控制的页面区域，此时我们可以使用包含选择符。

包含选择符通过空格标识符来表示，前面的一个选择符表示包含框对象的选择符，而后面的选择符表示被包含的选择符（如图 1.30 所示）。

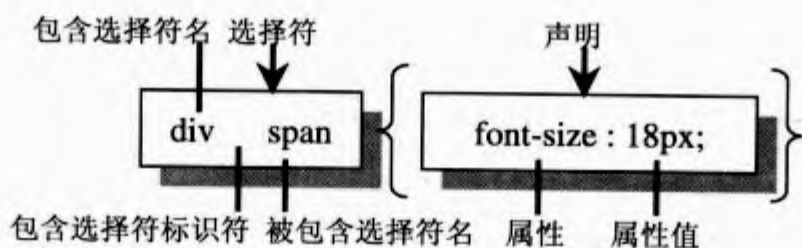


图 1.30 包含选择符

例如，在下面我们希望定义<div id="header">包含框内的段落文本字体大小为 14 像素，然后定义<div id="main">包含框内的段落文本字体大小为 12 像素。

```
<div id="wrap">
  <div id="header">
    <p>头部区域第 1 段文本</p>
    <p>头部区域第 2 段文本</p>
    <p>头部区域第 3 段文本</p>
  </div>
  <div id="main">
    <p>主体区域第 1 段文本</p>
    <p>主体区域第 2 段文本</p>
    <p>主体区域第 3 段文本</p>
  </div>
</div>
```

这时可以利用包含选择符来快速定义它们的样式，代码如下：

```
<style type="text/css">
#header p {
  font-size:14px;
}
#main p {
  font-size:12px;
}
</style>
```

当然对于上面的结构，你也可以使用子选择符来定义它们的样式：

```
<style type="text/css">
#header > p {
  font-size:14px;
}
#main > p {
  font-size:12px;
}
</style>
```

但是如果页面结构比较复杂，所有包含元素不仅仅是子元素，这时就只能使用包含选择符了。例如，对于下面这样的结构就只能使用包含选择符来进行定义。

```
<div id="wrap">
  <div id="header">
    <h2>
      <p>头部区域第 1 段文本</p>
```

```

    </h2>
    <p>头部区域第 2 段文本</p>
    <p>头部区域第 3 段文本</p>
  </div>
  <div id="main">
    <div>
      <p>主体区域第 1 段文本</p>
      <p>主体区域第 2 段文本</p>
    </div>
    <p>主体区域第 3 段文本</p>
  </div>
</div>

```

包含选择符的用处是比较广泛的,在选择符嵌套中经常被使用,同时该选择符还可以被 IE 6 版本浏览器识别,因此使用它不用考虑浏览器兼容性问题。

1.6.4 伪类和伪元素选择符

伪类和伪元素是一类特殊的选择符,它定义了一些特殊区域或特殊状态下的样式,这些特殊的区域或特殊状态是无法通过标签、ID 或 Class 以及其他属性来进行精确控制。

例如,我们希望控制段落中第一行文本或第一个字符的样式,但是又无法通过具体的标签或属性来进行控制,此时只能够通过伪元素来进行定义。也许你希望控制鼠标单击过程中超链接显示为不同的状态,这时只能够通过伪类来控制鼠标经过时、单击时、单击之后等不同的超链接样式。

伪类和伪元素以冒号(:)为前缀来表示,用法格式如图 1.31 所示。注意伪类和伪元素的前缀符号(:)与前后名称之间不要有空格。

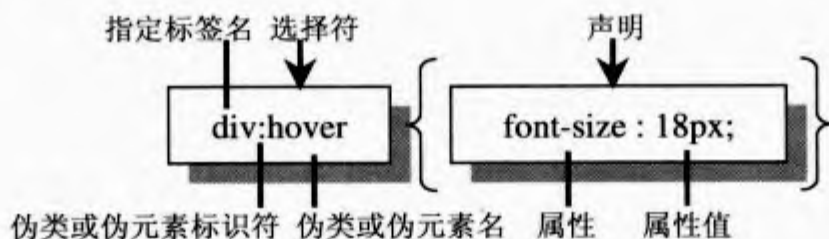


图 1.31 伪类和伪元素选择符

例如,下面是利用超链接的 4 个伪类选择符定义超链接文本的 4 种不同显示状态。

```

<style type="text/css">
a:link { /* 正常链接状态下样式 */
  color: #FF0000;
}
a:visited { /* 被访问之后的样式 */
  color: #0000FF;
}
a:hover { /* 鼠标经过时的样式 */
  color: #00FF00;
}
a:active { /* 超链接被激活时的样式 */
  color: #FF00FF;
}
</style>
<a href="#">超链接文本</a>

```

如果去除上面样式前面的 a 元素,还可以为其他元素,甚至所有元素定义鼠标的四种状态

样式。例如，在下面的样式中就可以为 body、div 和 span 元素定义鼠标的活动状态样式。

```
<style type="text/css">
:link { /* 正常状态下样式 */
    color: #FF0000;
}
:visited { /* 被访问之后的样式 */
    color: #0000FF;
}
:hover { /* 鼠标经过时的样式 */
    color: #00FF00;
}
:active { /* 单击被激活时的样式 */
    color: #FF00FF;
}
</style>
<div>鼠标经过样式</div>
<p>鼠标经过样式</p>
```

当然，在演示时可能看不完鼠标的 4 种状态样式，因为 body 元素也被定义了 4 种状态样式。所以，应该在伪类标识符前面指定一个具体的标签名。

其他伪类和伪对象的说明和使用可以参考本书光盘附赠的参考手册。另外，在使用时应该注意：除了超链接的 4 种伪类选择符之外，其他伪类和伪对象选择符不被 IE 6 及其以下版本浏览器支持，使用时应慎重。

1.6.5 CSS 选择符的嵌套

在 CSS 选择符中，你还可以使用选择符嵌套来实现对 HTML 结构中纵深元素的控制。嵌套的层级没有明确限制。嵌套的方法是利用空格来实现的（如图 1.32 所示）。

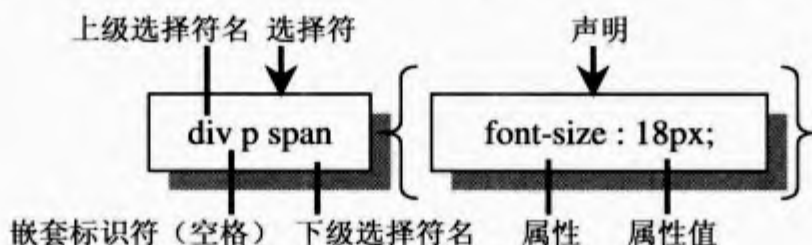


图 1.32 选择符嵌套结构

例如，在下面这个相对复杂的 HTML 结构中，包含了 2 个标题元素。

```
<div id="wrap">
  <div id="header">
    <h2><span>网页标题</span></h2>
    <div id="menu">
      <ul>
        <li><span>首页</span></li>
        <li>菜单项</li>
      </ul>
    </div>
  </div>
  <div id="main">
    <h2><span>栏目标题</span></h2>
    <p>主体内容</p>
  </div>
</div>
```

如果要控制标题的不同显示样式，此时使用选择符嵌套是比较理想的选择。使用多层嵌套一方面能够精确控制元素，另一方面还能够提升选择符的优先级。

```
<style type="text/css">
#wrap #header h2 span {
    font-size:24px;
}
#wrap #main h2 span {
    font-size:14px;
}
</style>
```

使用上面多级选择符嵌套所定义的不同标题显示样式如图 1.33 所示。

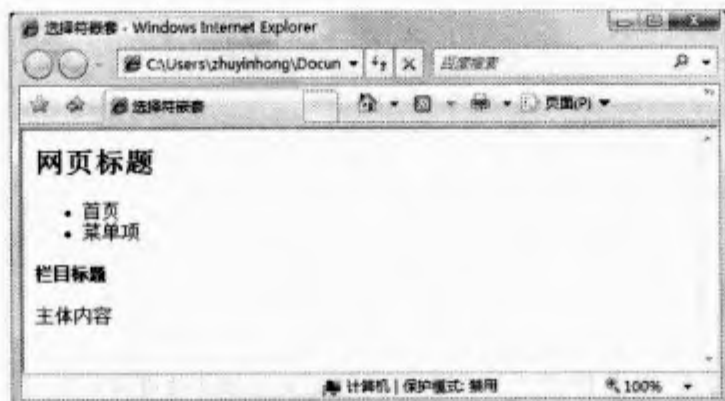


图 1.33 嵌套选择符的演示效果

同时你还可以嵌套更多的选择符或者跳级嵌套。例如，针对上面所定义的样式，可以使用如下嵌套选择符来进行定义。当然，对于读者来说，具体采用哪种嵌套结构可以根据需要酌情选择。

```
<style type="text/css">
#header h2 span {
    font-size:24px;
}
#main h2 span {
    font-size:14px;
}
</style>
```

1.7 CSS 的继承性

在面向对象的编程语言中继承是一个重要的概念，CSS 语言虽然没有其他语言那么严谨、复杂，但是也具有编程语言的一些基本特征（如继承性），下面我们就来介绍 CSS 的继承性及其用法。

1.7.1 认识 CSS 的继承性

继承是一种机制，它允许 CSS 样式不仅可以应用于某个特定的元素，还可以应用于它的后代。通俗说就是在 HTML 文档结构中，包含在内部的标签将拥有外部标签的某些样式。

CSS 继承性最典型的应用就是在 `body` 元素中定义整个页面的字体大小、颜色等基本页面属性，这样包含在 `body` 元素内的其他元素都将继承该基本属性，以实现页面显示效果的统一。例如，在 `body` 定义字体大小为 12 像素，通过继承性，包含在 `body` 元素的所有其他元素都将继

承该属性，并设置包含的字体大小为 12 像素（如图 1.34 所示）。

```
<style type="text/css">
body {
    font-size:12px;
}
</style>
<div id="wrap">
    <div id="header">
        <div id="menu">
            <ul>
                <li><span>首页</span></li>
                <li>菜单项</li>
            </ul>
        </div>
    </div>
    <div id="main">
        <p>主体内容</p>
    </div>
</div>
```

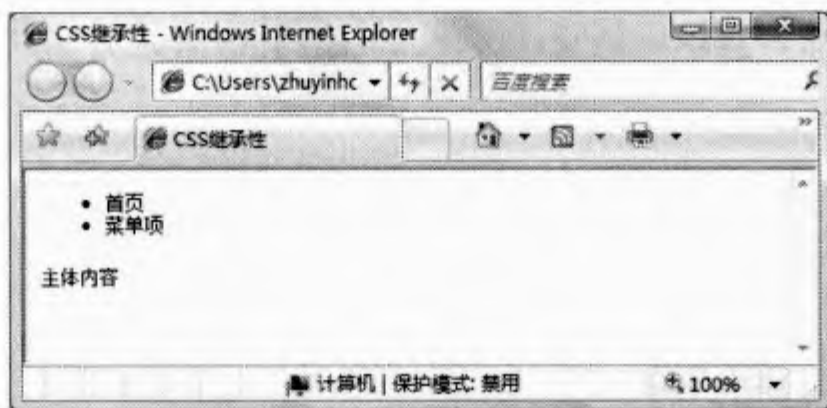


图 1.34 CSS 继承性演示效果

灵活利用 CSS 继承性可以为你节省大量的 CSS 代码，缩短开发时间。因此，当你准备开发时，建议先总结以下页面显示样式的效果，并把页面或模块中相同的可以继承的属性提取出来，然后在总包含框中定义，利用继承性让这些属性影响所包含的所有子元素。

1.7.2 CSS 继承性的局限性

CSS 继承性给网页设计师节省了大量的开发时间，提供更大的发挥空间。也许当你在使用 CSS 时都没有想到样式继承的问题，很自然地想到在 body 中定义网页字体大小，甚至不需要考虑是否能够这样去做，但是继承也有其局限性。

首先，有些属性是不能继承的。这没有任何原因，只是因为它就是这么设置的。例如，对于 background 属性来说，我们都知道该属性是用来设置元素的背景，它是没有继承性的。结合实际想一想，它也不应该有继承性，如果所有包含元素都继承了背景属性，那么文档看起来就会很怪异，除非你不使用该属性。

有关 CSS 属性的继承性，可以参阅本书光盘附赠的参考手册，每个属性都显示了它继承性选项。

其次，CSS 继承性还会存在一些错误。例如，在下面这个示例中有些浏览器中表格就不会继承 body 元素的属性（如 IE 6 以下版本浏览器），因此并不显示为 12 像素大小。

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

```
body {
    font-size:12px;
}
</style>
<table width="100%" border="0" cellspacing="0" cellpadding="0">
    <tr>
        <td>表格字体大小</td>
    </tr>
</table>
```

这显然是不正确的，为了稳妥起见，我们还需要利用群组的方式来进行定义：

```
<style type="text/css">
body,table,th,td{
    font-size:12px;
}
</style>
```

最后，元素通过继承性获取上级元素的样式，但是这些样式影响力是非常弱的，更专业的讲就是优先级比较低，下一节将详细讲解。如果当元素本身包含了相冲突的样式，则将忽略继承得来的样式。例如，在下面结构中由于 h2 元素默认定义了字体大小，所以将忽略从 body 元素继承来的属性，因此显示为不同的大小效果（如图 1.35 所示）。

```
<style type="text/css">
body {
    font-size:12px;
}
</style>
<div id="wrap">
    <div id="header">
        <h2><span>网页标题</span></h2>
        <div id="menu">
            <ul>
                <li><span>首页</span></li>
                <li>菜单项</li>
            </ul>
        </div>
    </div>
    <div id="main">
        <h2><span>栏目标题</span></h2>
        <p>主体内容</p>
    </div>
</div>
```

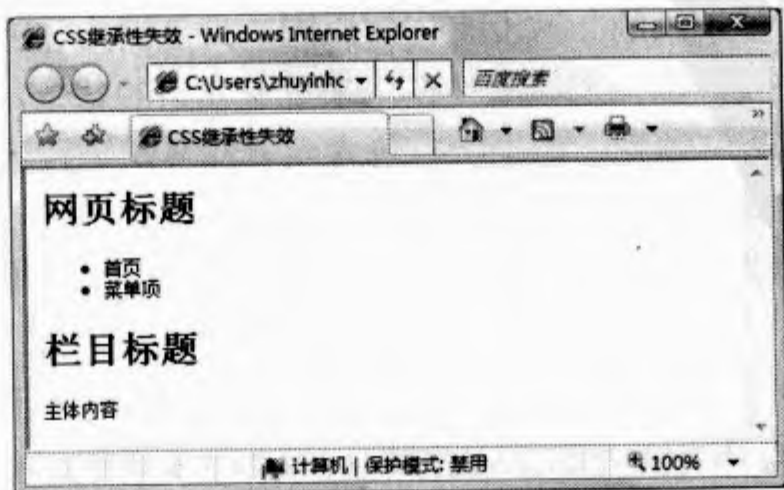


图 1.35 CSS 继承性失效演示效果

1.8 CSS 样式层叠和选择符优先级

CSS 是 Cascading Style Sheets 的简称，即层叠样式表的意思。只要我们为页面不同元素定义样式，就不可避免的会出现样式层叠现象，也就是说多个样式作用于同一个对象身上。如果这些样式声明的属性不同也就罢了，如果声明的属性相同，而设置的属性值不同该怎么办呢？这就是本节要讲解的核心问题“CSS 优先级”。

1.8.1 CSS 样式表的优先级

如果按照 CSS 的起源，我们可以将网页定义的样式分为 4 种：HTML、作者、用户、浏览器。HTML 表示元素的默认样式，作者就是创建人，即创建网站的所编辑的 CSS，用户也就是浏览网页的人所设置的样式，浏览器就是指浏览器默认的样式。

原则上讲，作者定义的样式优先于用户设置的样式，用户设置的样式优先于浏览器的默认样式，而浏览器的默认样式会优先于 HTML 的默认样式。

但请注意，在 CSS2 中当用户设置的样式中使用了 !important 关键字声明之后，用户的 !important 关键字会优先于作者声明的 !important 关键字。

1.8.2 CSS 样式的优先级

但是对于相同 CSS 起源来说，不同位置的样式其优先级也是不同的。一般来说，行内样式会优先于内嵌样式表，内部样式表会优先于外部样式表。而被附加了 !important 关键字的声明会拥有最高的优先级。

例如，在下面这个示例中，我们分别在 p 元素行内定义一个内嵌属性样式 (style="font-size:14px")，然后在文档的头部定义一个内部样式 p { font-size:24px;}，最后在外部分样式表文件 (1-27.css) 中定义一个外部样式 p { font-size:34px;}，并利用 <link> 标签链接到文档中。

在浏览器中预览，则根据 CSS 样式的优先级，最终显示结果为 14 像素（如图 1.36 所示）。

```
<style type="text/css">
p {
    font-size:24px;
}
</style>
<link href="1-27.css" rel="stylesheet" type="text/css" />
<p style="font-size:14px">段落文本</p>
```

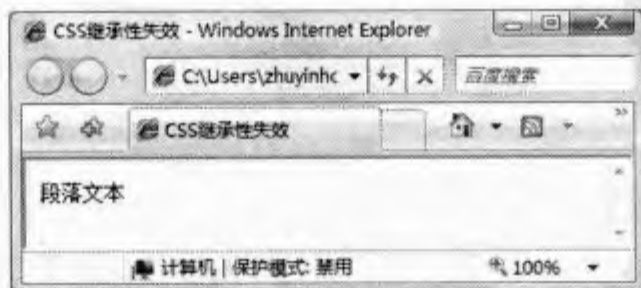


图 1.36 CSS 样式继承效果

1.8.3 CSS 选择符的优先级

上面两节是从大的角度来分析 CSS 样式的优先级问题，但是很多时候我们所遇到的都是同

一个来源，且位于相同位置的不同样式，那么该如何去区分它们的优先级呢？例如，在下面这个示例中 `div` 元素该显示为什么边框线呢？

```
<style type="text/css">
body div#box {
    border:solid 2px red;
}
#box {
    border:dashed 2px blue;
}
div.red {
    border:double 3px red;
}
</style>
<div id="box" class="red">CSS 选择符的优先级</div>
```

对于这样的问题比较复杂，不过这里教你一个快速计算方法，一切都可以简单解决。

首先，请读者记住，对于常规选择符它们都拥有一个优先级积分，说明如下：

- 标签选择符：优先级积分为 1；
- 伪元素或伪对象选择符：优先级积分为 1；
- 类选择符：优先级积分为 10；
- 属性选择符：优先级积分为 10；
- ID 选择符：优先级积分为 100；
- 其他选择符：优先级积分为 0，如通配选择符等。

然后，以上面积分数为起点来计算每个样式中选择符的总积分。计算的规则是：

- 统计选择符中 ID 选择符的个数，然后乘以 100；
- 统计选择符中类选择符的个数，然后乘以 10；
- 统计选择符中的标签选择符的个数，然后乘以 1。

以此方法类推，最后把所有积分相加，即可得到当前选择符的总加权值，最后根据加权值来决定哪个样式的优先级大。

例如，对于上面的样式表，我们可以这样计算它们的优先级加权值：

```
body div#box = 1 + 1 + 100 = 102;
#box = 100
div.red = 1 + 10 = 11
```

因此，最后的优先级为 `body div#box` 大于 `#box`，`#box` 大于 `div.red`，所以我们可以看到显示效果为 2 像素宽的红色实线（如图 1.37 所示）。



图 1.37 CSS 优先级的样式演示效果

使用 Dreamweaver 实现 CSS 布局

工具对于 Web 开发来说作用变得越来越明显,有时候会非常关键。但也有遗憾,例如目前还没有完全成熟的 CSS 布局工具,这对于习惯使用表格布局的设计师来说,是很不适应的。因为他们能够在可视化的操作环境中模拟图像编辑器的思维和操作方式来设计网页。很显然就目前的技术条件,设计师还不能够脱离代码而直接绘制符合 CSS 标准的网页。

当然,现在也涌现了很多出色的 CSS 编辑器,其中 Dreamweaver CS3 对于 CSS 技术的支持更是让人兴奋。如果你熟练掌握了 Dreamweaver CS3 的 CSS 可视化操作技巧,就能够在一定程度上减轻手工输入代码的痛苦,也加快了开发速度。本章将介绍如何利用这个工具来布局页面,也希望它将成为你在标准设计中最亲密的伙伴。

2.1 认识 Dreamweaver 的 CSS 布局 workflow

Dreamweaver 是从 MX 2004 版本开始引入 CSS 技术的,并在 Dreamweaver 8 中改进了 CSS 的功能,但是基本功能未变。因此如果你已经习惯了 Dreamweaver 8 的操作界面,现在转到 Dreamweaver CS3 就会非常容易。

Dreamweaver 对于 CSS 的支持在 CS3 版本中已经非常完善了。不管是支持的功能还是渲染的性能,都有了非常大的进步。如果你在 Dreamweaver CS3 中打开一个完全符合标准的 HTML 文档,编辑器内显示效果会非常好,但是在 Dreamweaver 8 中有时候就会不正常的显示。下面我们来认识 Dreamweaver CS3 对于 CSS 支持的基本功能。

2.1.1 增强 CSS 功能的【属性】面板

【属性】面板是 Dreamweaver 主界面中一个重要的操作面板,该面板自 Dreamweaver 8 版本面世以来基本上没有发生改变,但是与 Dreamweaver MX 2004 相比有了少许改进(如图 2.1 所示)。



图 2.1 Dreamweaver CS3 的【属性】面板

当在【属性】面板中设置文本样式时, Dreamweaver 会自动创建 CSS 样式声明。例如,在一个文档中选中一段文本,然后在【属性】面板中设置字体颜色为红色。此时你会在【代码】

视图下发现 Dreamweaver 会自动帮助你定义一个类样式 (STYLE2)，增加标签，并把类样式应用到该标签上面 (如图 2.2 所示)。



图 2.2 Dreamweaver CS3 自动定义 CSS 样式

如果不希望 Dreamweaver CS3 自动增加 CSS 样式来设置属性，则可以关闭该功能。操作步骤是：选择【编辑】|【首选参数】菜单命令，然后在【首选参数】对话框中取消勾选【使用 CSS 而不是 HTML 标签】复选框 (如图 2.3 所示)。然后单击【确定】按钮之后，关闭对话框。此时如果再次定义字体颜色时，你会发现 Dreamweaver 使用传统方法来定义字体颜色 (如图 2.4 所示)。

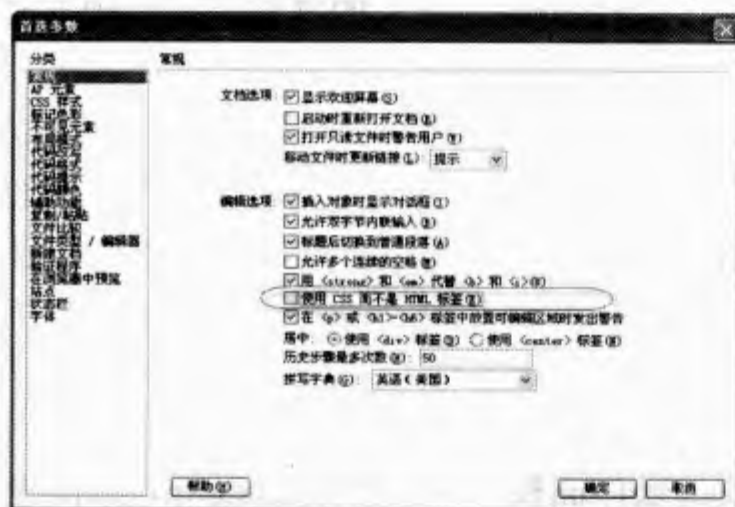


图 2.3 取消自动定义 CSS 样式

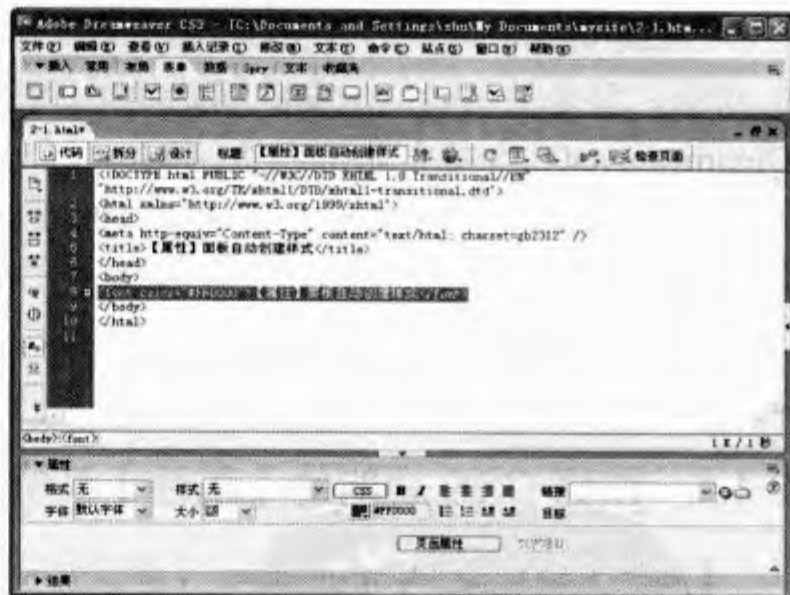


图 2.4 传统定义字体属性的方法

当选择标签来设置文本样式时，Dreamweaver 会将样式声明作为标签的定义来创建，否则 Dreamweaver 会将样式声明作为新类样式来创建。

例如，如果直接选中“【属性】面板自动创建样式”文本，然后在【属性】面板中定义它的属性，则 Dreamweaver CS3 会自动定义一个样式类，并增加一个新的标签，然后把样式类应用到标签上面，而如果选中如下的<p>。

<p>【属性】面板自动创建样式</p>

在【属性】面板中定义字体属性，则 Dreamweaver CS3 会自动定义一个样式类，并把该类

应用到该标签上面, 而不再新增标签。

其中“<!--”和“-->”分界符表示当浏览器(如早期的浏览器)不支持 CSS 样式表功能时, 则会忽略掉这些样式代码, 避免它们被显示出来。

```
<style type="text/css">
<!--
.STYLE1 {
    color: #FF0000
}
-->
</style>
<p class="STYLE1">【属性】面板自动创建样式</p>
```

另外, 读者还可以在【属性】面板中的【样式】下拉菜单中选择当前对象所要应用的类样式(如图 2.5 所示), 这样能够快速应用已经定义的类型样式到页面对象中。



图 2.5 Dreamweaver CS3 的【样式】下拉列表框

如果单击【属性】面板中的【页面属性】按钮, 就可以打开【页面属性】对话框, 方便用户快速定义来影响页面显示的基本样式。

此外单击【属性】面板中包含的【CSS】按钮, 可以快速访问【CSS 样式】面板中的相关定义, 并进行对应操作。

2.1.2 【页面属性】对话框

如果你在【属性】面板中单击【页面属性】按钮, 或者选择【修改】|【页面属性】菜单命令, 可以打开【页面属性】对话框(如图 2.6 所示)。

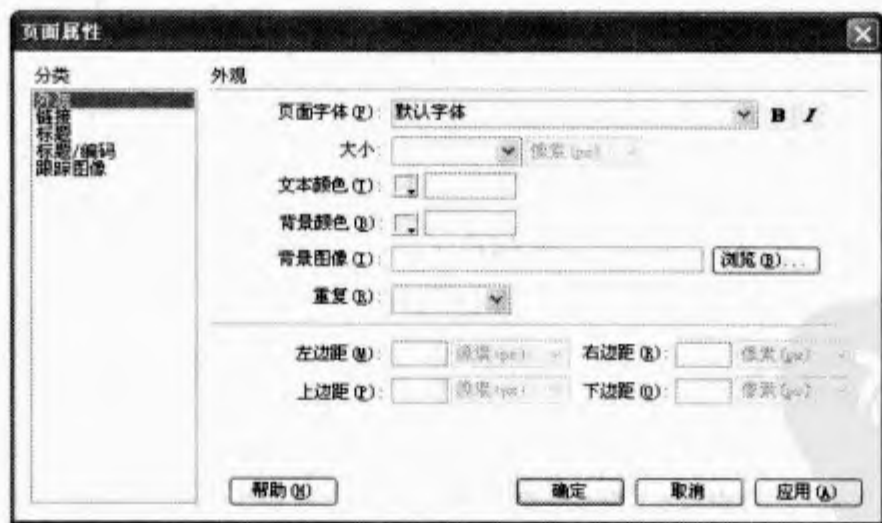


图 2.6 Dreamweaver CS3 的【页面属性】对话框

【页面属性】对话框方便设计师快速设置整个页面的基本样式。在 Dreamweaver MX 2004 之前, 其中部分设置是使用 CSS 样式声明来指定的, 现在可以完全使用 CSS 进行控制。

需要强调的是, 使用【页面属性】对话框设置的 CSS 属性代码都被存放在 HTML 文档头部区域, 即使该文档链接到外部样式表。不过 Dreamweaver CS3 开始提供了一个把内部样式转移到外部的功能按钮(如图 2.7 所示)。

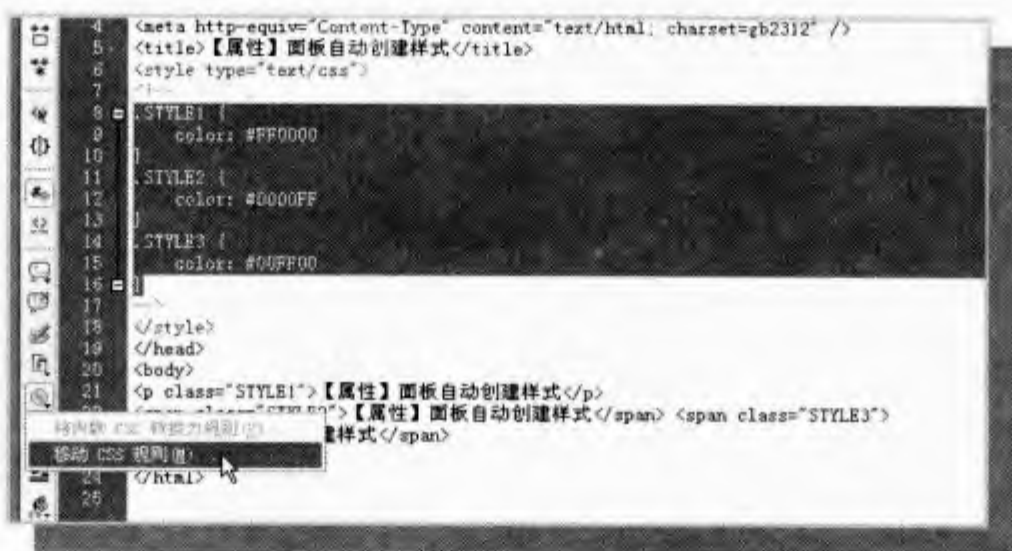


图 2.7 Dreamweaver CS3 的转移 CSS 规则功能

【页面属性】对话框能够很方便的帮助你设置页面的基本属性，特别是手动输入 CSS 代码不是很熟悉的情况下，这种可视化操作绝对是一个不错的选择。

在【外观】类别中可以选择设置页面的字体设置、背景颜色和图像、页边距等详细内容的 CSS 样式声明。读者还可以为【链接】类别中每个超链接状态的外观创建 CSS 样式声明。

除设置用于所有标题的字体效果外，【标题】类别还方便用户设置每个标题级的字体大小和颜色。

2.1.3 【CSS 样式】面板

Dreamweaver CS3 的【CSS 样式】面板没有太大改进，保持了 Dreamweaver 8 中相同的界面效果（如图 2.8 所示）。

【CSS 样式】面板包含两种模式：全部和正在。每种模式显示相应的样式声明，详细说明如下：

(1) 【全部】模式列出可用于整个文档的声明，也就是说在任何超链接或导入的外部样式表中可用的所有声明都会自动显示出来，包括文档头部的各种 CSS 声明（如图 2.9 所示）。

(2) 【正在】模式列出影响页面上所选对象的声明。在【正在】模式下，【CSS 样式】面板显示以下内容：所选内容的摘要、规则和属性（如图 2.8 所示）。

“摘要”类别只列出那些影响所选内容外观的属性。例如，如果在段落内单击，而该段落被指定了称为 STYLE2 的类样式，则【CSS 样式】面板中的摘要类别显示该类样式定义的颜色（#0000FF），而页面属性中定义的颜色（#666666），如图 2.8 所示。

“规则”类别显示影响当前所选内容的规则层叠。层叠中的第一条规则是 body 标签中定义的样式。按照就近所选内容的规则，该规则中定义的属性是矛盾的，因此，摘要部分不显示那些属性。

有一种极好的替代显示，通过单击“规则”部分中的内部按钮即可使用。“关于”类别显示有关当前选定属性的有用信息。“关于”类别的标题表明选定的属性是颜色属性。它包含该特定属性的是哪条规则以及该规则定义的位置。

单击“摘要”类别中的任一属性，或者单击“规则”类别中的一条规则，相应规则中定义的属性将显示在“属性”类别中。“属性”类别还表明哪些属性在影响所选内容，任何矛盾的和超过层叠级别的属性，其名称上都会显示一道线。

“属性”类别是面板的最基本部分。你不仅可以编辑已经定义的任何属性的细节，还可以向该规则添加属性。单击“添加属性”链接可提供不同于 Dreamweaver 的定义 CSS 样式的方法

(如图 2.9 所示)。

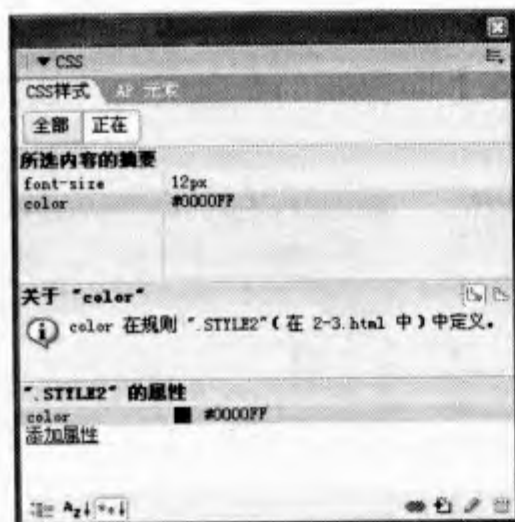


图 2.8 Dreamweaver CS3 的【CSS 样式】面板

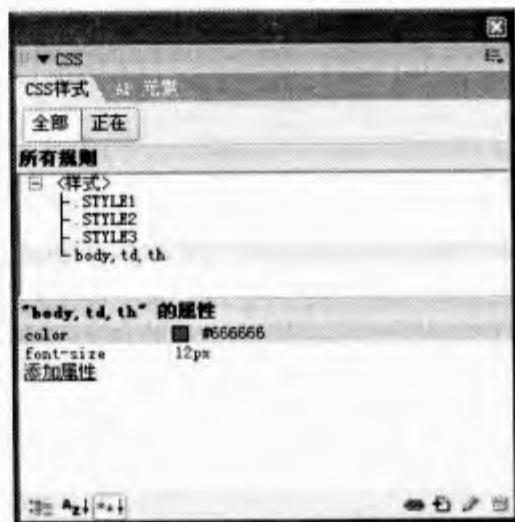



图 2.9 【全部】模式

在【CSS 样式】面板的【正在】模式中，设计师可以查看关于每个规则的信息，包括规则中定义的属性，或者查看影响所选对象的规则层叠（如图 2.10 所示），在【CSS 样式】面板顶部选中一个样式，然后在下面可以每个规则的声明，如果属性被标明了删除线，则说明该规则被其他样式中的规则所覆盖，这就是 CSS 样式的层叠性。如果对于 CSS 样式的优先级以及层叠性概念比较模糊，那么使用这种方法会很直观的看出每个选择符的优先级以及样式的层叠效果。



图 2.10 Dreamweaver CS3 标示出 CSS 样式的层叠性

在【CSS 样式】面板中，还可以修改为每个规则定义的属性。同时在【全部】和【正在】模式中，设计师都可以可直接在面板中添加、编辑或删除属性。如果要编辑样式，只需要单击【CSS 样式】面板底部的【编辑样式】按钮（）打开 CSS 编辑器。有关【CSS 样式】面板的更详细操作，我们将在下一节中详细介绍。

2.1.4 属性表

属性表是【CSS 样式】面板中一个重要功能，从 MX 2004 版本开始 Dreamweaver 就支持该功能，在 Dreamweaver 8 版本以后的不断升级，支持功能也得到了改进和完善。由于它功能太

强大了，因此本节将专门进行讲解。

属性表有 3 种显示视图：分类视图（如图 2.11 所示）、列表视图（如图 2.12 所示）和设置属性视图（如图 2.13 所示）。读者可以通过单击【CSS 样式】面板底部左侧的三个按钮来快速进行切换。

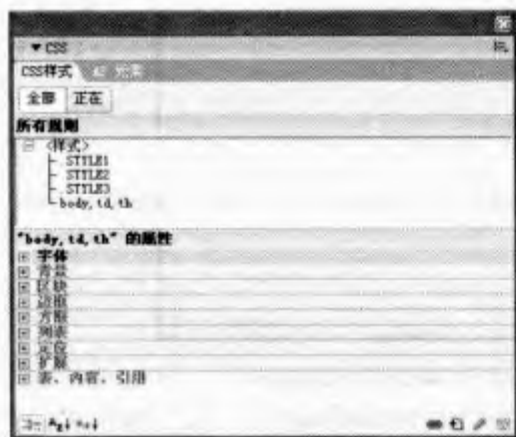


图 2.11 分类视图

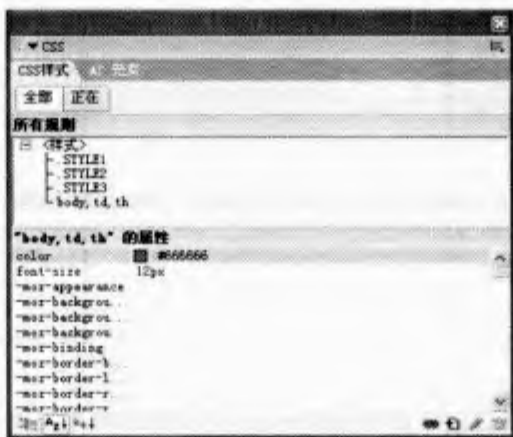


图 2.12 字母排序视图

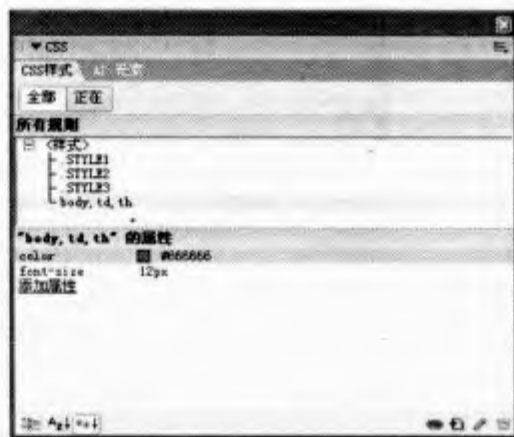


图 2.13 设置属性视图


请注意，属性表同时适用于【所有】模式和【正在】模式。

对于初学者来说，可以借助分类视图快速定位自己需要设置的属性，如果希望快速查找某个已经声明了的属性，则可以利用列表视图快速按顺序查找某个属性。对于水平较高的读者来说，可直接在设置属性下进行设置。

2.2 操作【CSS 样式】面板

【CSS 样式】面板集结了 Dreamweaver CS3 对于 CSS 技术支持的大部分功能。当然如果希望利用好 Dreamweaver CS3 这个工具来设计标准网页，那么能否熟练操作【CSS 样式】面板就至关重要了，本节将讲解如何快速操作【CSS 样式】面板。

2.2.1 新建样式

要新建样式，首先需要打开（新建）一个网页文档或者 CSS 样式表文件。选择【窗口】|【CSS 样式】菜单命令，打开【CSS 样式】面板。单击【CSS 样式】面板底部的【新建 CSS 规则】按钮（），打开【新建 CSS 规则】对话框，如图 2.14 所示。

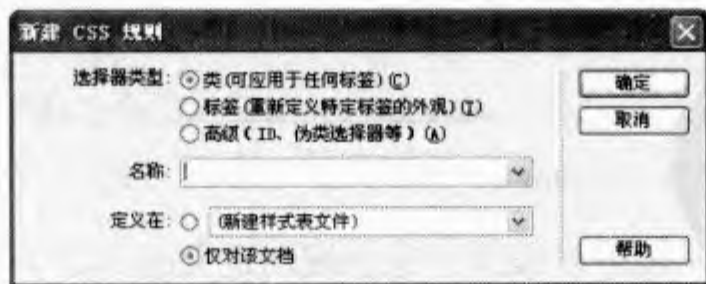


图 2.14 【新建 CSS 规则】对话框

在这个对话框中你可以创建一个 CSS 规则来自动完成 HTML 标签的格式设置、类样式或 ID 样式等。具体操作步骤如下。

将光标置于文档中特定位置，利用上述方法打开【新建 CSS 规则】对话框。

第 1 步，定义要创建的 CSS 样式的选择符类型。

1) 如果要创建可应用于文档任意位置的样式，可以选择【类】单选按钮，然后在【名称】

文本框中输入类样式的名称，类名称必须以句点开头，并且可以包含任何字母和数字组合。如果没有输入开头的句点，Dreamweaver 将自动输入。

2) 如果要重定义特定 HTML 标签的默认样式，可以选择【标签】单选按钮，然后在【标签】文本框中输入一个 HTML 标签，或从弹出菜单中选择一个标签。

3) 如果要为具体某个标签组合或所有包含特定 ID 属性的标签定义样式，可以选择【高级】单选按钮，然后在【选择器】文本框中输入一个或多个 HTML 标签，或从弹出菜单中选择一个标签。在弹出的下拉菜单中提供各种伪类选择器，如 a:active、a:hover、a:link 和 a:visited。

第 2 步，选择定义样式的位置。

1) 如果要将样式放置到已附加到文档的样式表中，可以选择相应的样式表。

2) 如果要创建外部样式表，可以选择【新建样式表文件】单选按钮。

3) 如果要在当前文档中嵌入样式，可以选择【仅对该文档】单选按钮。

例如，我们在文档中新建一个 center 类样式，则对话框设置如图 2.15 所示。

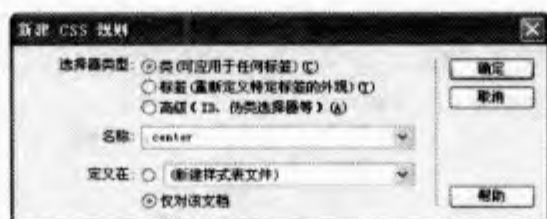


图 2.15 定义类样式

完成设置后，单击【确定】按钮关闭【新建 CSS 规则】对话框，这时 Dreamweaver CS3 将新建一个新的空白规则。

2.2.2 定义样式的属性

当你使用【新建 CSS 规则】对话框建立一个新样式时，Dreamweaver CS3 会自动打开【CSS 规则定义】对话框（如图 2.16 所示）。

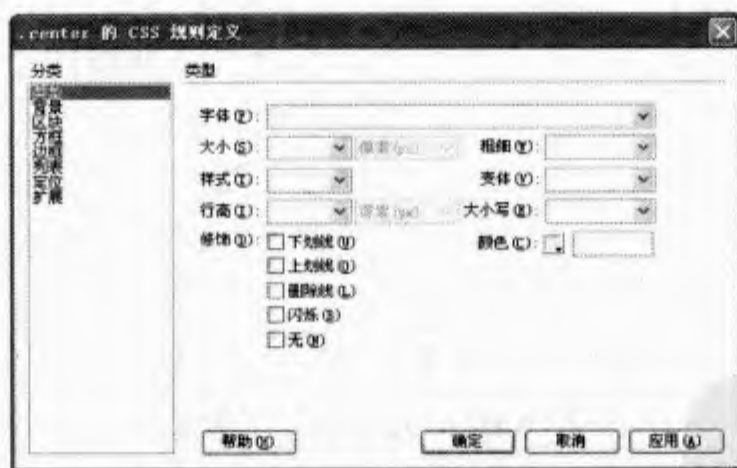


图 2.16 【CSS 规则定义】对话框

这个对话框比较复杂，涉及 CSS 大部分属性的设置。你可以在其中直观、快速的设置文本字体、背景图像和颜色、间距和布局属性以及列表元素外观。我们还将本书后面章节分别对其中重要属性进行分类分章讲解。

1. 定义 CSS 类型属性

在【CSS 规则定义】对话框中的【类型】类别中可以定义 CSS 样式的基本字体和类型设置（如图 2.17 所示）。其中主要设置属性说明如下。

- 字体：为样式设置字体系列（或多组字体系列）。

- 大小：定义字体大小。它可以通过选择数值和单位，设置文字的大小，也可以选择相对大小。使用像素作为单位可以有效地防止浏览器扭曲文本。
- 样式：设置“正常”、“斜体”或“偏斜体”作为字体样式，默认设置是“正常”。
- 行高：设置文本所在行的高度。
- 修饰：向文本中添加下划线、上划线或删除线，或使文本闪烁。文本默认设置是“无”。
- 粗细：对字体应用特定或相对的粗体量。“正常”等于 400；“粗体”等于 700。
- 变体：设置文本的小型大写字母变体。IE 支持该属性。
- 大小写：将所选内容中的每个单词的首字母大写或将文本设置为全部大写或小写。
- 颜色：设置文本颜色。

2. 定义 CSS 背景属性

在【CSS 规则定义】对话框中的【背景】类别中可以定义 CSS 的背景样式。你可以为页面中任何元素定义背景属性（如图 2.17 所示），其中主要设置属性说明如下。

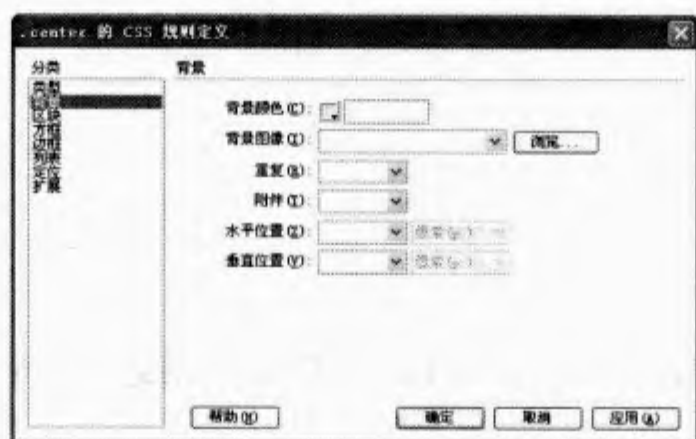


图 2.17 定义背景属性

- 背景颜色：设置元素的背景颜色。
- 背景图像：设置元素的背景图像。
- 重复：确定是否（如何）重复背景图像。“不重复”选项表示只在元素开始处显示一次图像，“重复”选项表示在元素的后面水平和垂直平铺图像，“横向重复”和“纵向重复”选项表示分别显示图像的水平带区和垂直带区。
- 附件：确定背景图像是固定在其原始位置还是随内容一起滚动。
- 水平位置和垂直位置：指定背景图像相对于元素的初始位置。

3. 定义 CSS 区块属性

在【CSS 规则定义】对话框中的【区块】类别中可以定义 CSS 的区块样式。你可以定义标签和属性的间距和对齐设置（如图 2.18 所示）。其中主要设置属性说明如下。



图 2.18 定义区块属性

- 单词间距：设置字词的间距。可以指定负值。
- 字母间距：增加或减小字母或字符的间距。若要减小字符间距，请指定一个负值。
- 垂直对齐：指定应用此属性的元素的垂直对齐方式。Dreamweaver 仅将该属性应用于 标签时，才在文档窗口中显示样式效果。
- 文本对齐：设置文本在元素内的对齐方式。
- 文字缩进：指定第一行文本缩进的程度。可以使用负值创建凸出显示，当应用于块状元素时，Dreamweaver 才会在文档窗口中显示效果。
- 空格：确定如何处理元素中的空格。其中“正常”将收缩空白，“保留”将保留所有空白，包括空格、制表符和 Enter，“不换行”表示指定仅当遇到
 标签时文本才换行。
- 显示：指定是否以及如何显示元素。“无”指定到某个元素时，它将禁用该元素的显示。

4. 定义 CSS 方框属性

在【CSS 规则定义】对话框中的【方框】类别中可以定义 CSS 的方框样式。可以定义有关元素盒模型以及显示相关的属性（如图 2.19 所示）。其中主要设置属性说明如下。

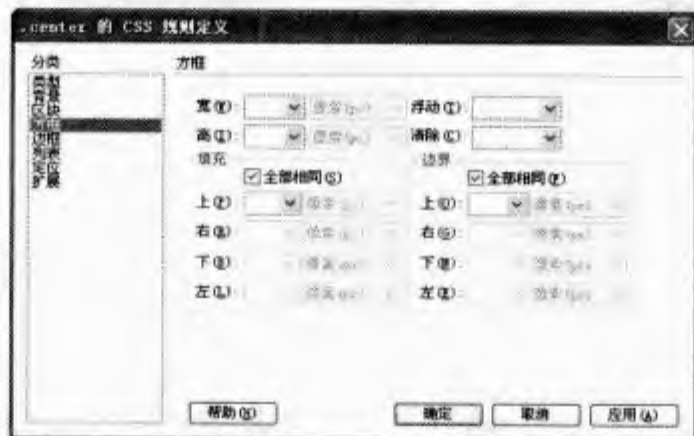


图 2.19 定义方框属性

- 宽和高：设置元素的宽度和高度。
- 浮动：设置元素在围绕元素的哪个边浮动。其他元素按通常的方式环绕在浮动元素的周围。
- 清除：定义不允许浮动元素的边。
- 填充：指定元素内容与元素边框之间的间距，如果没有边框，则可以作为边距。取消选择“全部相同”选项可设置元素各个边的填充。
- 全部相同：为应用该属性的元素“上”、“右”、“下”和“左”边设置相同的填充属性。
- 边距：指定一个元素的边框与另一个元素之间的间距。仅当该属性应用于块状元素时，Dreamweaver 才会在文档窗口中显示效果。
- 全部相同：为应用该属性的元素的“上”、“右”、“下”和“左”边设置相同的边距属性。

5. 定义 CSS 边框属性

在【CSS 规则定义】对话框中的【边框】类别中可以定义 CSS 的边框样式。你可以定义元素四周的边框样式，包括宽度、颜色和样式（如图 2.20 所示）。其中主要设置属性说明如下。

- 样式：设置边框的样式外观。样式的显示方式取决于浏览器。取消选择“全部相同”复选框可设置元素各个边的边框样式。
- 宽度：设置元素边框的粗细。
- 颜色：设置边框的颜色。
- 全部相同：为应用该属性的元素的“上”、“右”、“下”和“左”设置相同的边框属性。



图 2.20 定义边框属性

6. 定义 CSS 列表属性

在【CSS 规则定义】对话框中的【列表】类别中可以定义 CSS 的列表样式。可以为列表元素定义列表样式，如项目符号大小和类型（如图 2.21 所示）。其中主要设置属性说明如下。

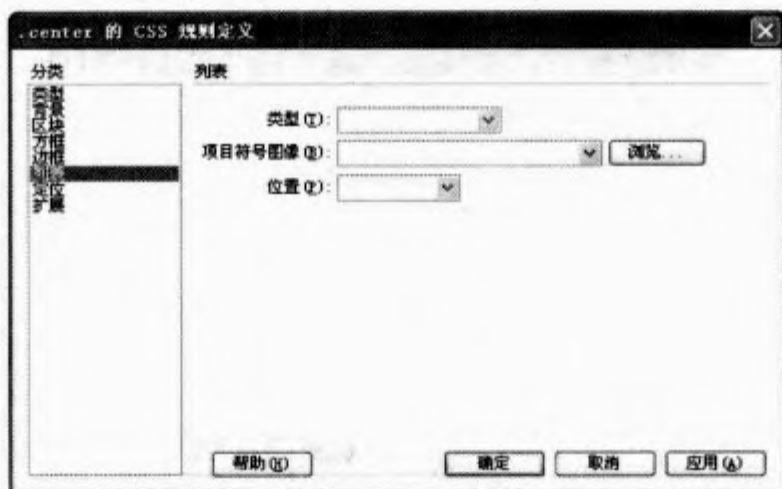


图 2.21 定义列表属性

- 类型：设置项目符号或编号的外观。
- 项目符号图像：为项目符号指定自定义图像。单击【浏览】按钮可以浏览选择图像，或键入图像的路径。
- 位置：设置列表项文本是否换行并缩进（外部）或者文本是否换行到左边距（内部）。

7. 定义 CSS 定位属性

在【CSS 规则定义】对话框中的【定位】类别中可以定义 CSS 的定位样式。可以为元素定义定位方式（如图 2.22 所示），有关定位布局的详细讲解可以参阅本书后面章节内容。其中主要设置属性说明如下。

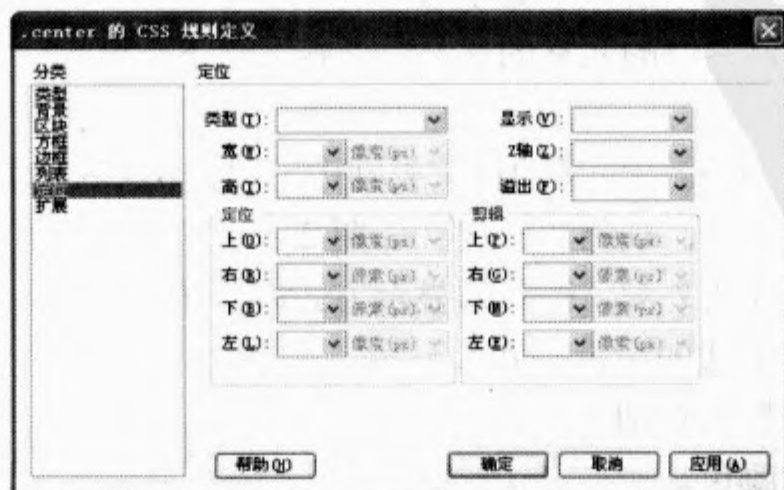


图 2.22 定义定位属性

- 类型：确定浏览器应如何来定位选定的元素，包括“绝对”、“相对”、“固定”和“静态”。
- 可见性：确定内容的初始显示条件，包括“继承（默认）”、“可见”和“隐藏”三个选项。
- Z 轴：确定内容的堆叠顺序。Z 轴值较高的元素显示在 Z 轴值较低的元素。值可以为正，也可以为负。
- 溢出：确定当包含框的内容超出显示范围时的处理方式，包括“可见”（将增加容器的大小，以使其所有内容都可见，容器将向右下方扩展）、“隐藏”（保持容器的大小并剪辑任何超出的内容）、“滚动”（将在容器中添加滚动条，而不论内容是否超出容器的大小）和“自动”（将使滚动条仅在容器的内容超出容器的边界时才出现）。
- 位置：指定内容块的位置和大小。
- 剪辑：定义内容的可见部分。如果指定了剪辑区域，可以通过脚本语言（如 JavaScript）访问它，并操作属性以创建像擦除这样的特殊效果。

8. 定义 CSS 扩展属性

在【CSS 规则定义】对话框中的【扩展】类别中可以定义 CSS 的扩展样式，如图 2.23 所示。其中主要设置属性说明如下。

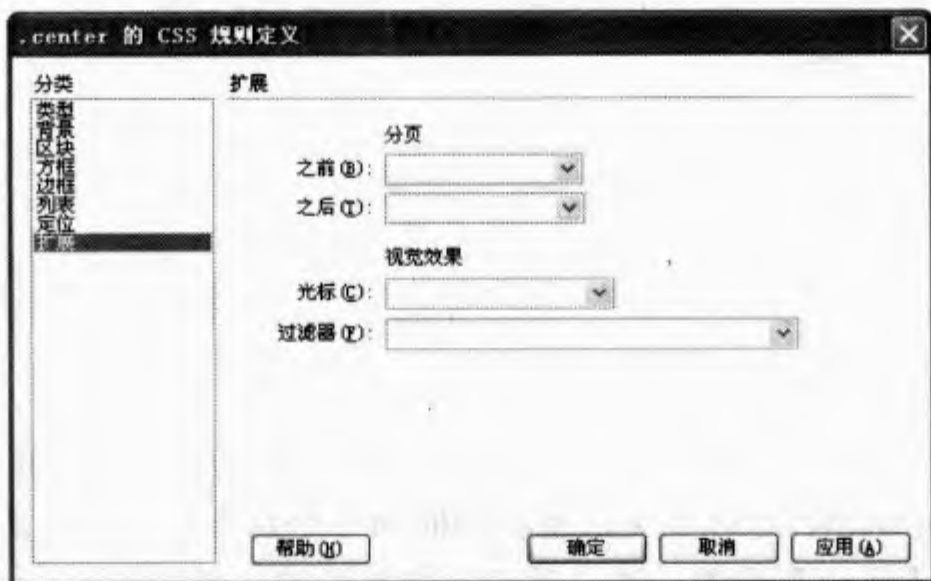


图 2.23 定义扩展属性

- 分页：当打印页面时在样式中所控制的对象之前或之后强行分页。
- 光标：设置当指针位于对象上时显示特定指针图标的效果。
- 过滤器：为对象定义各种特殊效果，但是该选项仅被 IE 浏览器支持。

2.2.3 应用样式

上一节简单介绍了【CSS 规则定义】对话框的分类和选项功能，当然这对于初学者来说不可能一步就全部熟悉和理解其中所有的选项，本书也没有那么多篇幅对每个选项逐一进行举例说明，不过你可以在后面章节中不断渗透学习。

当定义 center 类样式时，我们可以在【区块】类别中定义文本居中对齐（如图 2.24 所示）。

单击【确定】按钮关闭对话框，此时 Dreamweaver CS3 会在文档头部区域的<style>标签中定义了一个 center 类样式，代码如下所示：

```
.center {
    text-align: center;
}
```

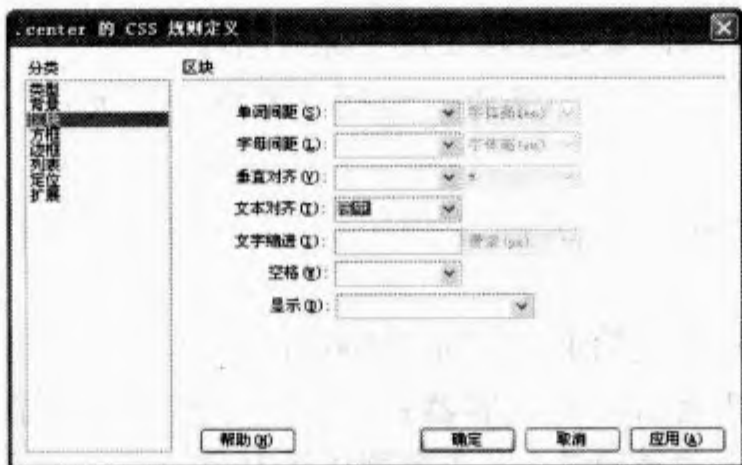


图 2.24 定义居中对齐类样式

除了标签样式之后，对于类样式和 ID 样式，读者都必须在文档中进行引用。引用类样式可以在【属性】面板的【样式】下拉列表中进行选择。凡是为该文档定义类样式都会自动显示在该下拉列表中，如图 2.25 所示。

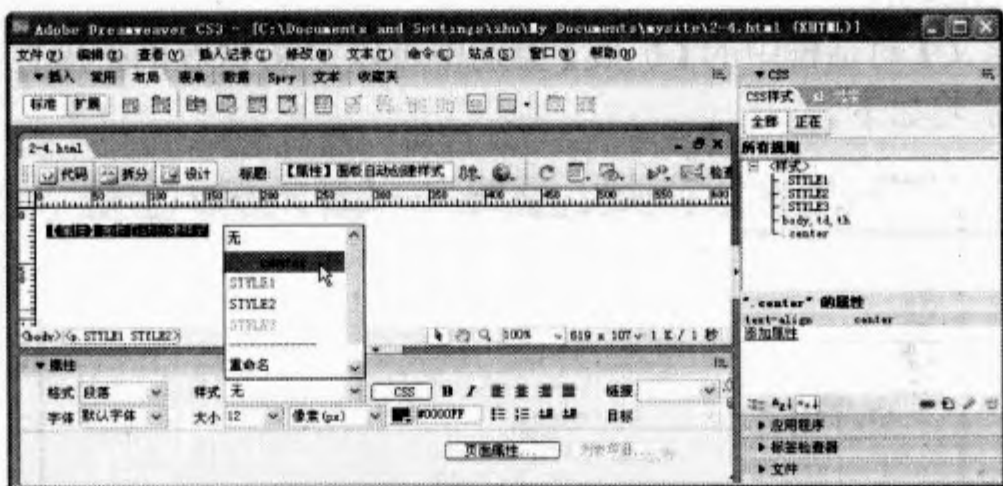


图 2.25 应用类样式

但是你会很快发现，当我们应用了 center 类样式之后，以前被定义的 STYLE2 类样式被覆盖了。也就是说 Dreamweaver CS3 还无法支持同时为一个对象可视化定义多个类样式的功能。为此，你必须切换到【代码】视图，通过手动增加类样式，代码如下：

```
<style type="text/css">
<!--
.STYLE2 {
    color: #0000FF
}
.center {
}
-->
</style>
<p class="STYLE2 .center">【属性】面板自动创建样式</p>
```

而对于 ID 选择符样式，可以使用如下方法来实现应用。例如，在【新建 CSS 规则】对话框中定义一个 #header 选择符（如图 2.26 所示）。

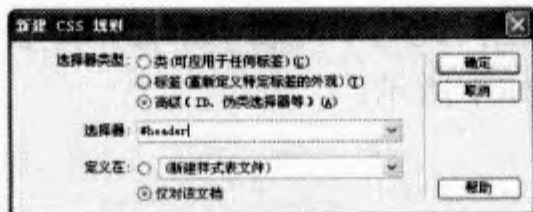


图 2.26 定义 ID 样式

然后在打开的【CSS 规则定义】对话框中定义该包含框的宽度为 100%、高度为 100 像素（如图 2.27 所示），定义边框显示为 1 像素宽的实线框（如图 2.28 所示）。

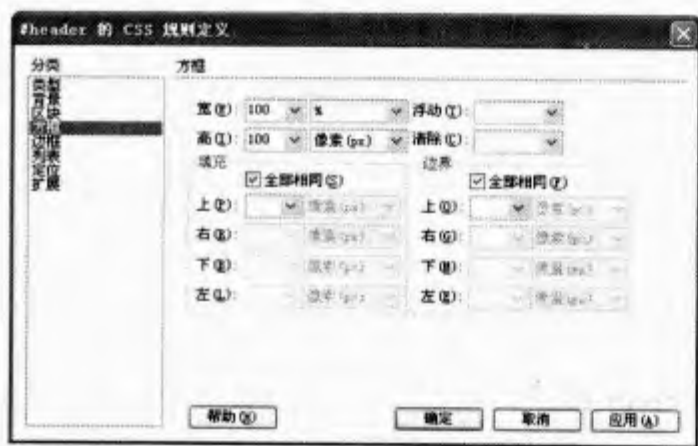


图 2.27 定义宽和高属性

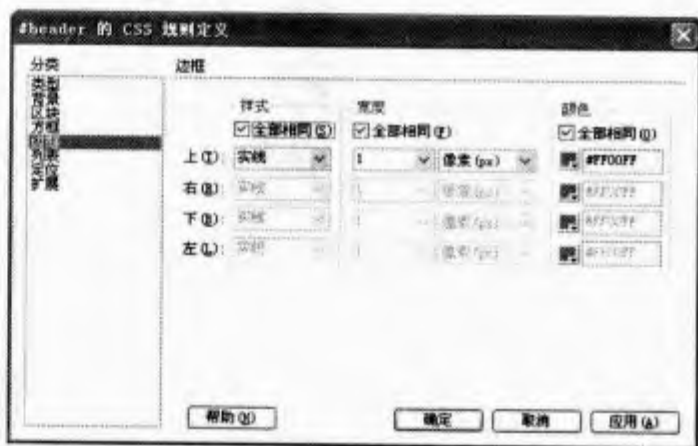


图 2.28 定义边框属性

定义完毕并单击【确定】按钮之后，我们就可以在文档编辑窗口中选中特定的包含框，然后在【属性】面板的【Div ID】下拉文本框中进行设置（如图 2.29 所示）。

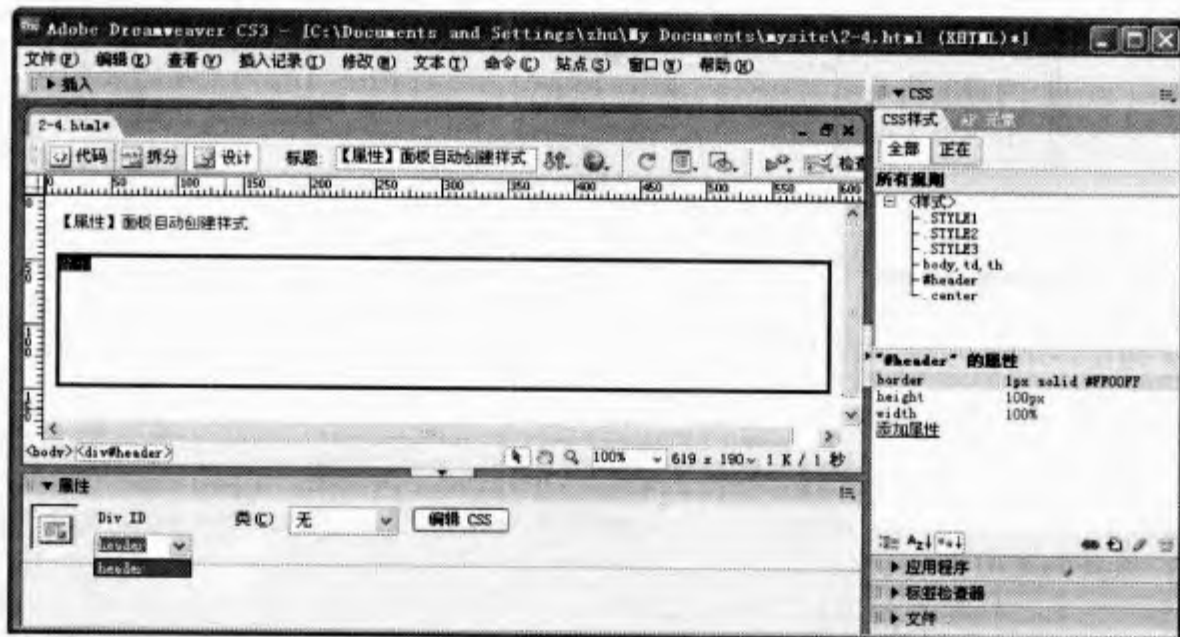


图 2.29 应用 ID 样式

2.3 以 CSS 技术为核心的设计视图

优秀的操作环境对于设计师来说至关重要，可喜的是 Dreamweaver CS3 强化了设计视图的 CSS 技术支持，你会感觉到在 Dreamweaver 中进行 CSS 标准设计无微不至的关怀。

另外，Dreamweaver CS3 改进了对 CSS 渲染功能，这样我们可以更好地在【设计】视图中边操作边显示 CSS 样式。

2.3.1 强大的 CSS 设备类型

CSS 所提供的样式是区分不同类型的，也就是说你可以为不同类型的设备定义样式，第 1 章中曾经介绍过 CSS 设备类型，目前我们常用的设备包括电脑屏幕（screen）、打印设备（print）、手持设备（handheld）等。

如果在 HTML 文档区域定义内部样式，可以在<style>标签中定义 media 属性来指定样式的设备类型（如图 2.30 所示）。

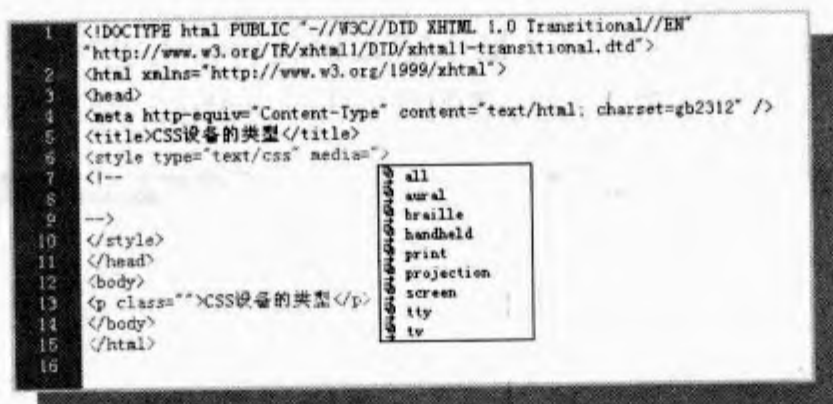


图 2.30 在 HTML 文档的内部样式表中制定设备类型

如果要引入外部样式表文件，也可以在 Dreamweaver CS3 中快速定义外部样式的设备类型（如图 2.31 所示）。

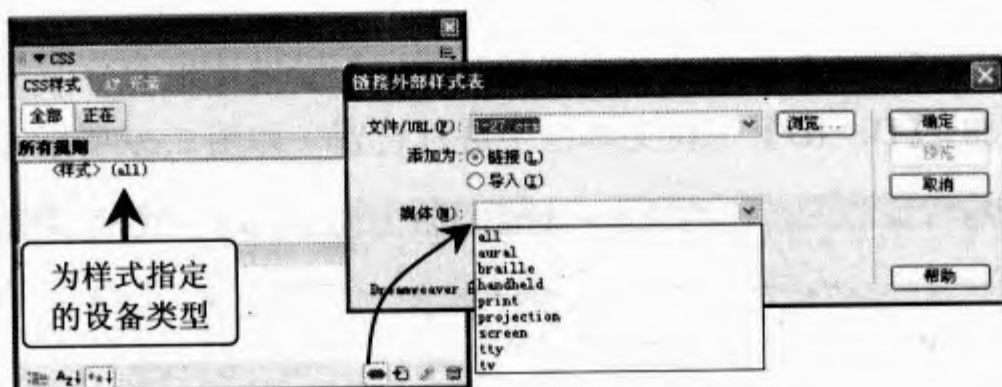


图 2.31 设置外部样式表文件的设备类型

上面介绍了如何为样式定义设备类型。当然利用 Dreamweaver CS3 提供的设备模拟功能，我们可以在不同设备下预览样式。例如，新建如下 HTML 文档：

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>CSS 设备的类型</title>
<style type="text/css" media="screen">
<!--
.color {
    color:red;
    font-size:14px;
}
-->
</style>
<style type="text/css" media="print">
<!--
.color {
    color:blue;
    font-size:24px;
}
-->
</style>
</head>
<body>
<p class="color">CSS 设备的类型</p>
</body>
</html>
```

在上面文档中定义了两个 CSS 样式，分别适用于电脑屏幕和打印设备。这时如果在【设计】视图下预览，则显示为红色的 14 像素的字体（如图 2.32 所示）。

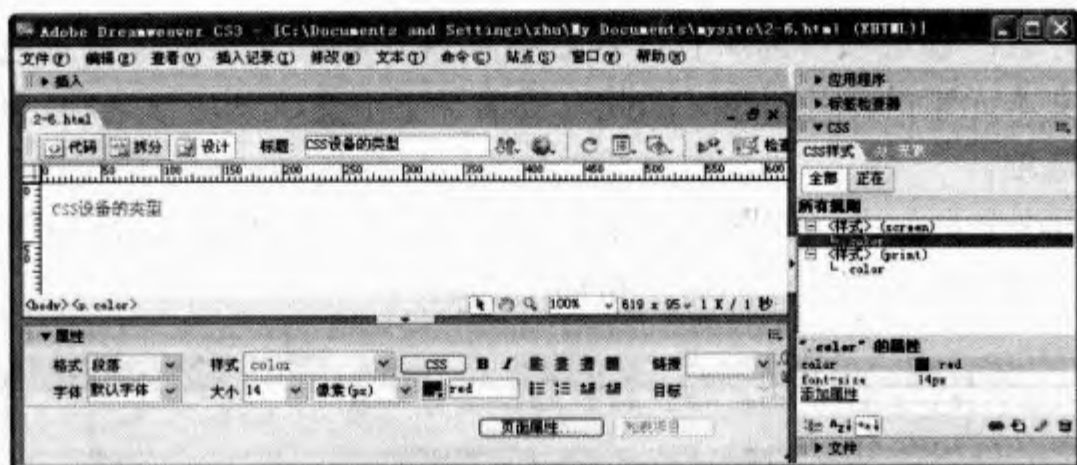


图 2.32 默认状态下 Dreamweaver CS3 的设备类型为电脑屏幕


当然你也可以转换到其他设备类型。选择【查看】|【样式呈现】，然后选择不同的媒体类型来设计或测试样式（如图 2.33 所示）。



图 2.33 利用不同设备类型来测试样式

如果你觉得用户会有不同的需要，那么在网站中集成多种媒体类型最轻松的方法是定义单独的打印媒体样式表。

2.3.2 可视化助理

在【文档】工具栏中包含一组显示选项。单击其中的【可视化助理】按钮（）可显示如下子菜单（如图 2.34 所示）。

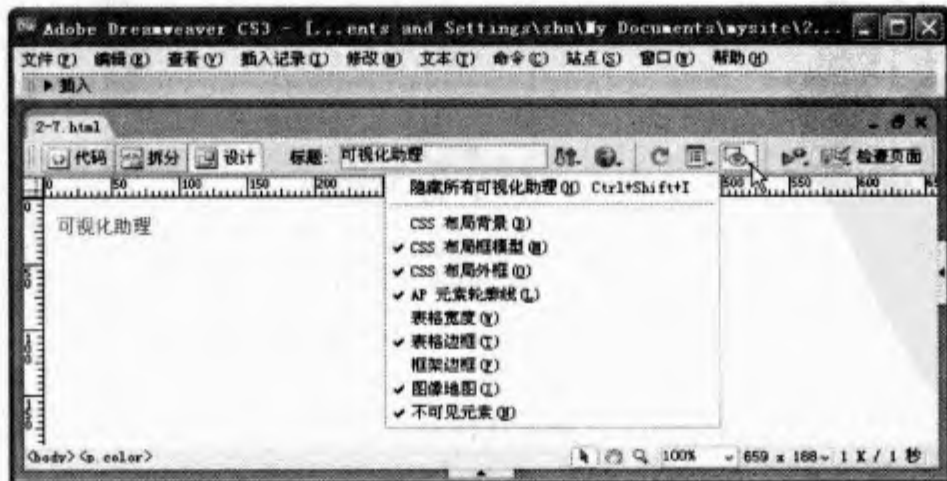


图 2.34 可视化助理工具

这些选项对于那些使用 CSS 进行布局的用户来说是非常有用的，现简单说明如下：

- **CSS 布局背景**: 该选项可为所有受到 CSS 属性影响的元素添加背景色。由于打开该设置会影响文本元素的颜色外观, 因此一般情况下不建议开启该选项, 可以根据需要启用或禁用该选项。

例如, 在下面这个页面中 (如图 2.35 所示), 通过启动 CSS 布局背景色, 可以快速查看某个包含框的范围, 以便快速查看它们对于布局的影响。

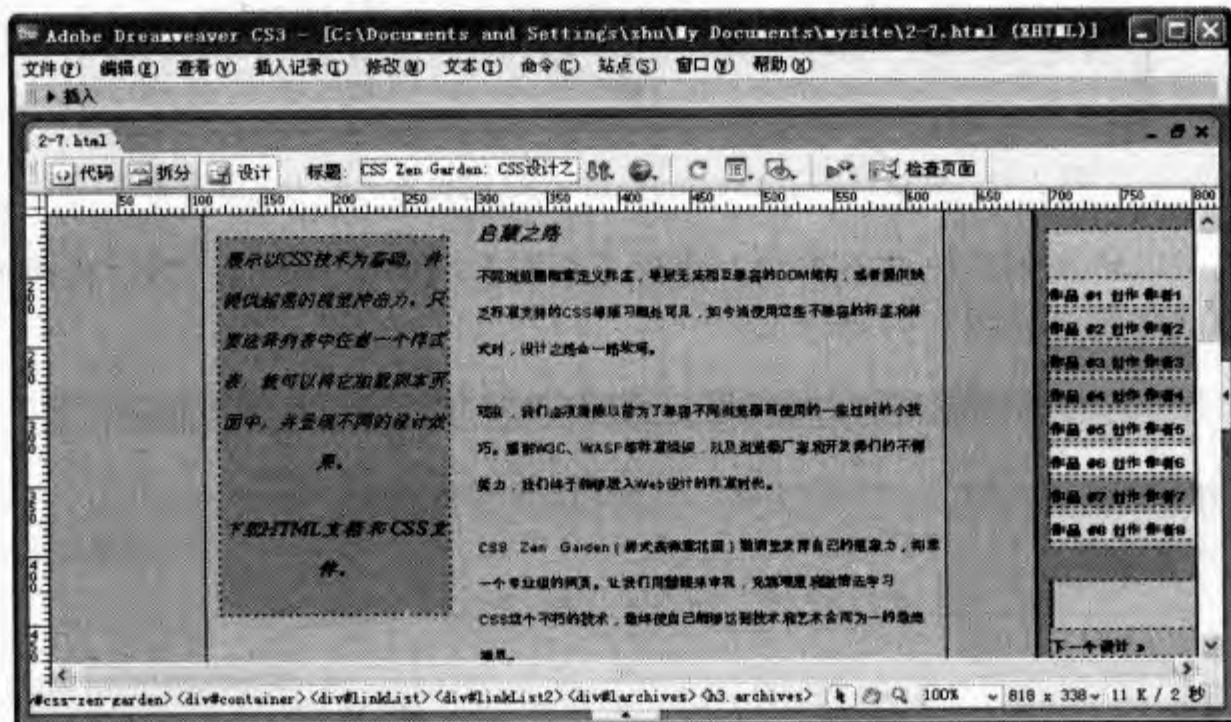


图 2.35 利用 CSS 布局背景快速查看元素之间的相互关系和影响

请注意, 这些布局背景颜色都是随机的, 每次打开所显示的颜色都可能会不同, 并且在打开或关闭该功能时, 背景颜色会发生变化。

- **CSS 布局框模型**: 该选项可在选定框元素的内外添加剖面线, 用来代表所有填充和边距设置 (如图 2.36 所示)。

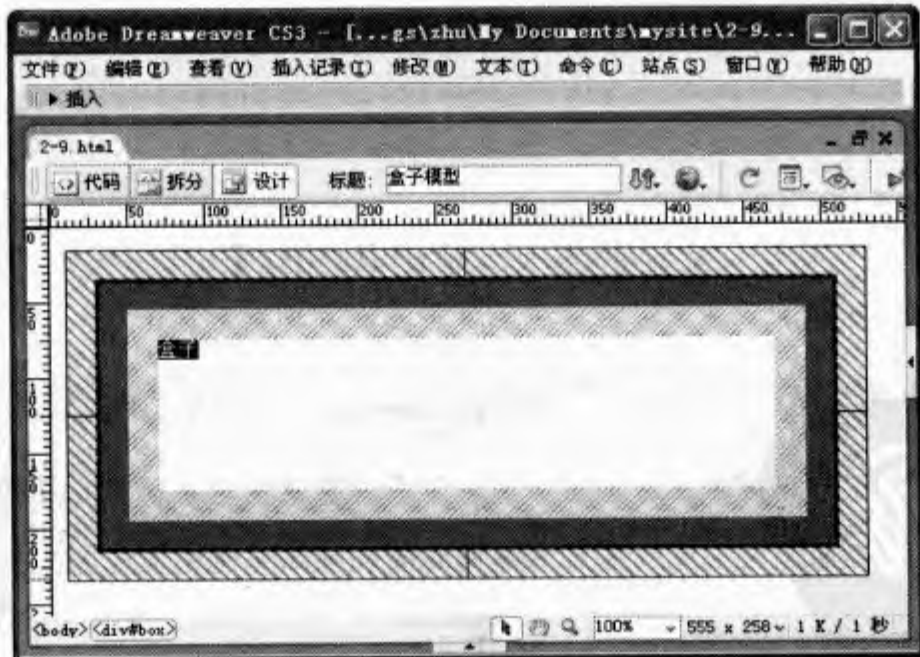


图 2.36 CSS 布局框模型

该工具对于 CSS 布局非常有帮助, 能够可视化地查看元素的大小、边距等设置属性。

另外, 一旦选定了该元素, 就会出现一个工具提示, 具体情况取决于鼠标指针悬停在元素内的位置。如果悬停在外剖面线上, 则工具提示仅指示边距设置。如果悬停在内部剖面线上, 则工具提示仅指示填充设置。如果悬停在内容上, 则工具提示将指示宽度和高度设置等。

- **CSS 布局外框**：利用该选项可以给所有 div 布局元素添加一个外框。这不仅有助于开发人员选择该标签，而且它也是一种了解哪些元素受该标签控制的简便方法，特别是当 div 元素没有定义边框时更有帮助（如图 2.37 所示）。



图 2.37 CSS 布局外框

- **层外框**：该选项只给被定义了盒模型的 div 元素增加外框，也就是说它给那些用于定位的 div 元素添加外框。

2.4 使用 DW 实现 CSS 布局实战

设计师总感觉 Dreamweaver 在构建标准页面方面还存在很多问题，自然与表格布局相比，CSS 布局在很多时候还必须结合源代码操作才能够实现。当然，如果你熟悉了 Dreamweaver 操作环境，并了解一些 HTML 和 CSS 语言的基本用法，这种半自动的操作也是很惬意的，相对其他一些 CSS 编辑器来说，Dreamweaver 还算是比较人性的，因为目前还没有完全可视化的 CSS 操作工具。

为了提高读者对于 Dreamweaver 的热爱，下面我们尝试使用该工具制作一个完全符合 CSS 标准的页面布局结构。

2.4.1 使用 Dreamweaver CS3 构建页面结构

构建 CSS 标准页面的第 1 步，自然是构建一个符合标准的 HTML 结构。一般来说，对于一个符合标准的 HTML 结构，其基本框架都是由 <div> 标签来设计的。在构建 HTML 结构时，可以不要考虑页面到底显示什么效果，避免你在构建页面时受样式效果的束缚。

本示例的 HTML 结构如下：

```
<div id="container">
  <div id="header">
    <h1>页眉</h1>
  </div>
  <div id="left">左栏</div>
  <div id="right">右栏</div>
  <div id="footer">
    <p>页脚</p>
  </div>
</div>
```

这是一个简单的 3 行 2 列式结构，由一个 <div id="container"> 包含框，嵌套了 4 个子模块。如果你不喜欢输入代码或者手动输入比较生疏，也可以借助 Dreamweaver 可视化操作功能轻松完成，不过对于一名熟练的设计师来说，这是难以容忍的，毕竟在 Dreamweaver 中手动输入代码时配合代码自动提示，输入效率会更高（如图 2.38 所示）。

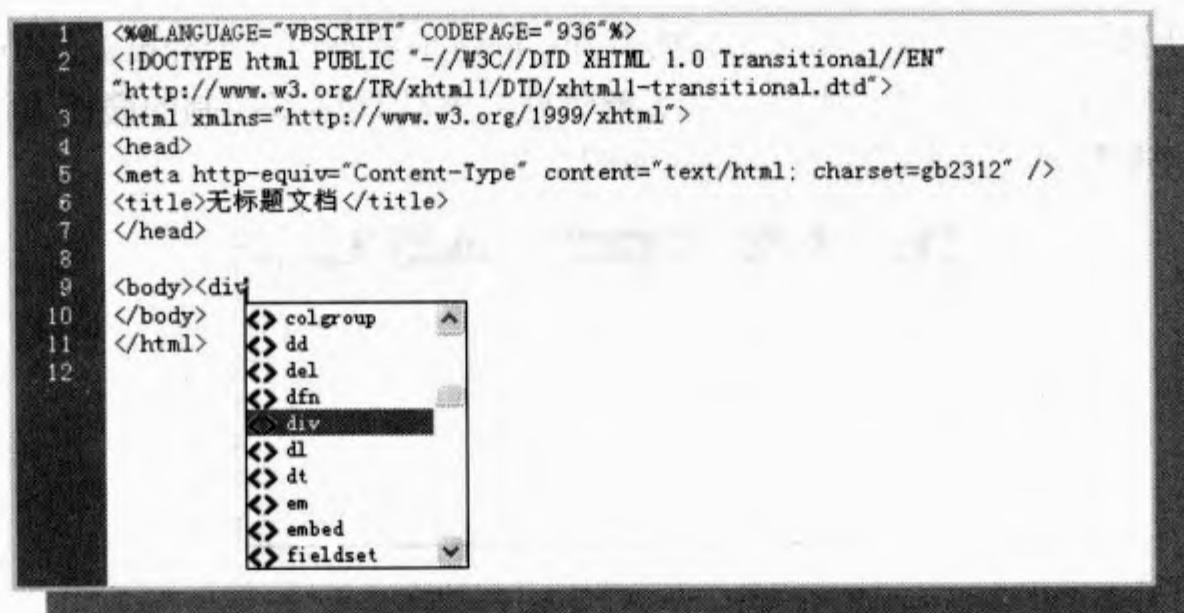


图 2.38 代码输入自动提示

借助 Dreamweaver 可视化操作构建该结构的方法如下:

第 1 步, 把光标置于页面当前位置。切换到【代码】视图, 并在代码视图下进行可视化操作, 这样你能够准确定位代码插入的位置。

第 2 步, 选择【插入记录】|【布局对象】|【Div 标签】菜单命令, 打开【插入 Div 标签】对话框, 并在其中设置一个 ID 名 (如图 2.39 所示)。这里输入 “container”, 为整个页面定义一个包含框。

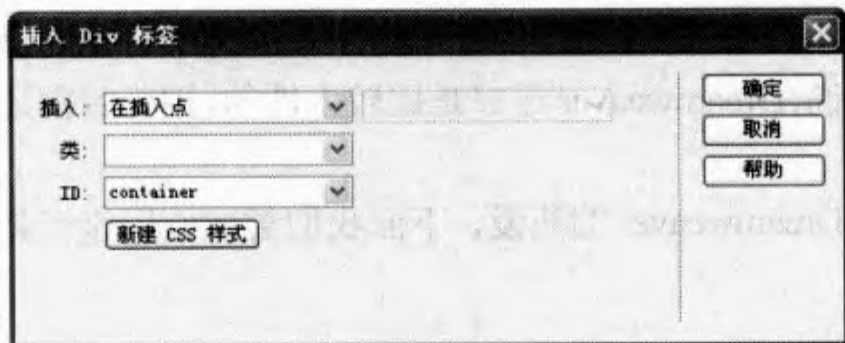


图 2.39 【插入 Div 标签】对话框

第 3 步, 单击【确定】按钮关闭对话框, 这时 Dreamweaver CS3 会自动在当前位置插入如下代码, 其中 “此处显示 id “container” 的内容” 提示文本处于被选中状态。

```
<div id="container">此处显示 id "container" 的内容</div>
```

第 4 步, 再次选择【插入记录】|【布局对象】|【Div 标签】菜单命令, 打开【插入 Div 标签】对话框, 并在其中设置 “header” ID 属性名 (如图 2.40 所示), 并在【插入】下拉列表框中选择 “在选定内容旁换行” 选项。

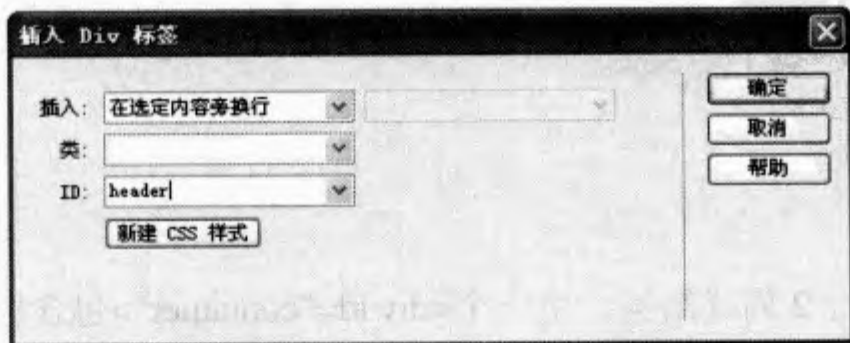


图 2.40 【插入 Div 标签】对话框

第 5 步, 单击【确定】按钮, 关闭对话框, 则 Dreamweaver 会插入如下代码, 并选中其中

的“此处显示 id "container" 的内容”。

```
<div id="container">
  <div id="header">此处显示 id "container" 的内容</div>
</div>
```

第6步,继续前面的操作步骤,然后在【插入 Div 标签】对话框中插入一个 ID 为 left 的 div 元素,并把位置定位到<div id="header">标签的后面,设置如图 2.41 所示。在【插入】下拉列表框中选择“在标签之后”,然后在后面显示的下拉列表框中选择“<div id="header">”。

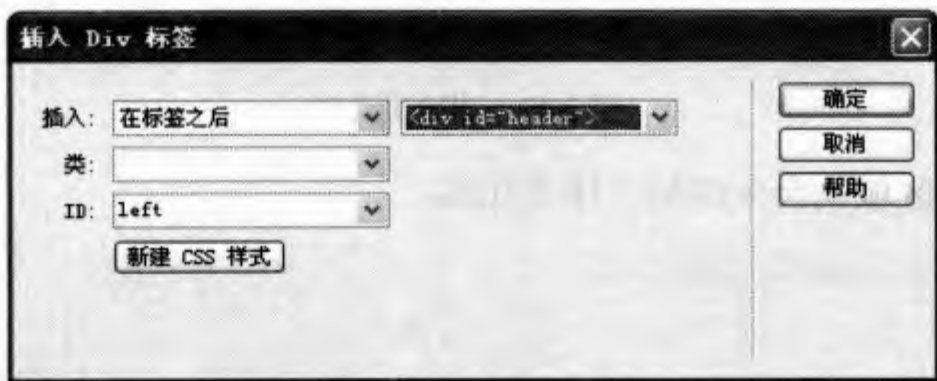


图 2.41 【插入 Div 标签】对话框

第7步,以相同的方式继续加入其他 div 结构标签。如果 Dreamweaver 不能够替换其中的提示字符,则需要你手动删除其中的文本信息,如“此处显示 id "left" 的内容”。

```
<div id="container">
  <div id="header">此处显示 id "container" 的内容</div>
  <div id="left">此处显示 id "left" 的内容</div>
  <div id="right">此处显示 id "right" 的内容</div>
  <div id="footer">此处显示 id "footer" 的内容</div>
</div>
```

第8步,当插入所有的结构标签,如上设置所示。然后在<div id="header">子结构中插入一个一级标题;在<div id="footer">中插入一个段落标签。方法是:把光标置于需要插入的位置,然后在【插入】工具栏的【文本】选项卡中单击插入(如图 2.42 所示)。

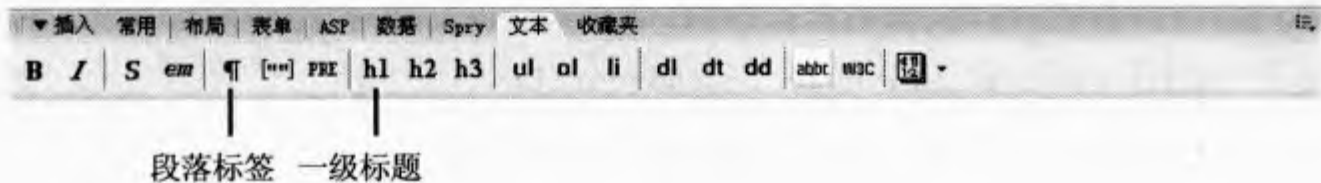


图 2.42 使用【插入】工具栏快速插入标签

2.4.2 使用【页面属性】统一基本样式

当我们完成了页面基本结构之后,下面就可以着手进行布局了,同时定义结构的基本样式。布局的第一步是应该对页面基本属性进行分析,也就是说根据需要统一页面元素的默认样式。所有这些操作都可以通过【页面属性】对话框来完成。

第1步,清除页面的默认页边距。在默认状态下,页面会显示 10 像素左右的页面边距,这对于页面设计会产生一定的影响,为此我们需要首先清除这些默认的页边距。方法是:选择【修改】|【页面属性】菜单命令,打开【页面属性】对话框,并在【外观】分类中设置【左边距】、【右边距】、【上边距】和【下边距】文本框中的值为 0 (如图 2.43 所示)。

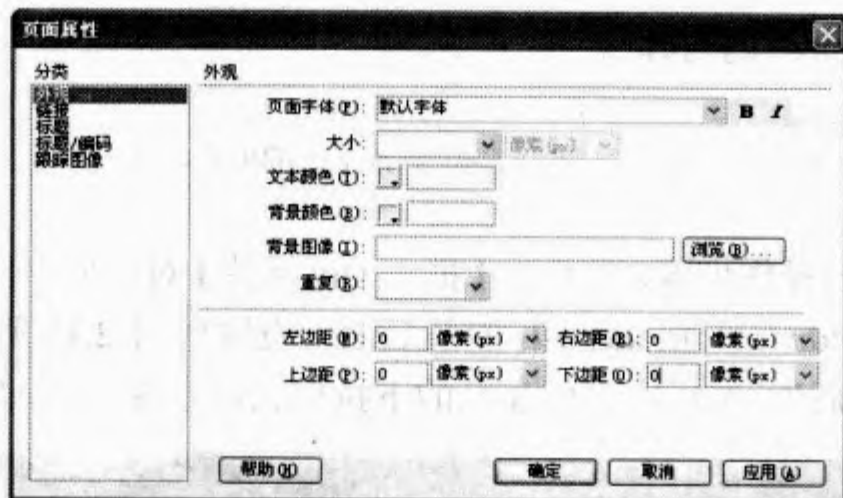


图 2.43 清除页边距

这时在文档头部区域就会生成如下样式代码：

```
<style type="text/css">
<!--
body {
    margin-left: 0px;
    margin-top: 0px;
    margin-right: 0px;
    margin-bottom: 0px;
}
-->
</style>
```

但是考虑到浏览器的兼容性，我们还需要在 body 元素中手动增加如下声明。

```
body {
    padding: 0;
}
```

第 2 步，定义网页字体属性。方法同上，也是在【页面属性】对话框的【外观】分类中进行设置（如图 2.44 所示）。这里定义页面字体大小为 76%，它是以页面默认字体大小（16 像素）来进行计算的，约为 12 像素。代码如下：

```
body,td,th {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 76%;
}
```

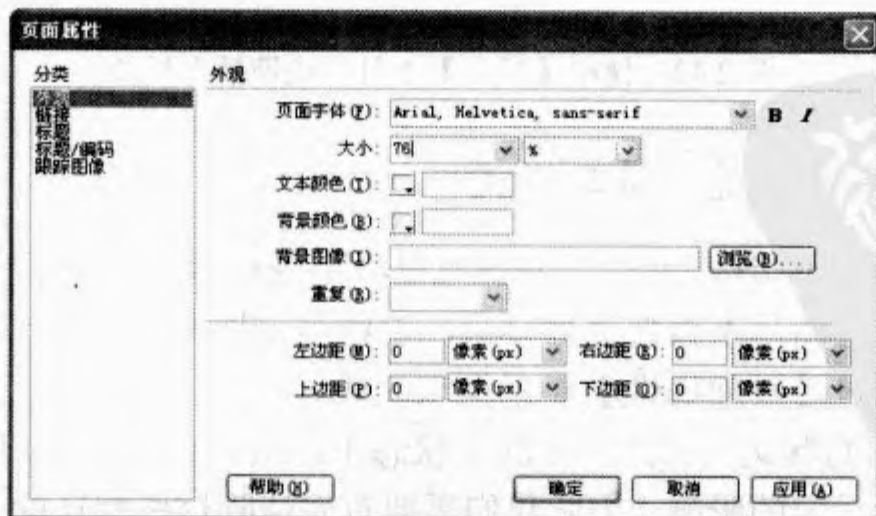


图 2.44 统一页面字体属性

第 3 步，统一页面超链接的样式。在【页面属性】对话框的【链接】分类中进行设置（如

图 2.45 所示)。这里定义不显示超链接下划线，并统一字体颜色，所生成的代码如下：

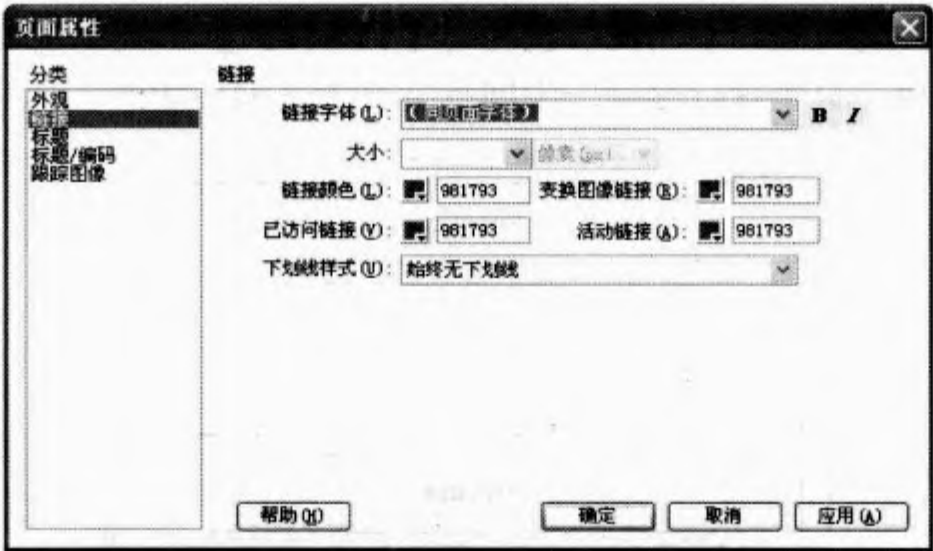


图 2.45 统一页面超链接样式

```
a:link { /* 正常状态下超链接样式 */
    color: 981793;
    text-decoration: none;
}
a:visited { /* 已经访问过时超链接样式 */
    text-decoration: none;
    color: 981793;
}
a:hover { /* 鼠标移过时的超链接样式 */
    text-decoration: none;
    color: 981793;
}
a:active { /* 超链接被激活时超链接样式 */
    text-decoration: none;
    color: 981793;
}
```

第 4 步，在【CSS 样式】面板中定义一个 p 标签样式，设置 p 元素的内边距（或称为填充），设置如图 2.46 所示，所生成的代码如下：

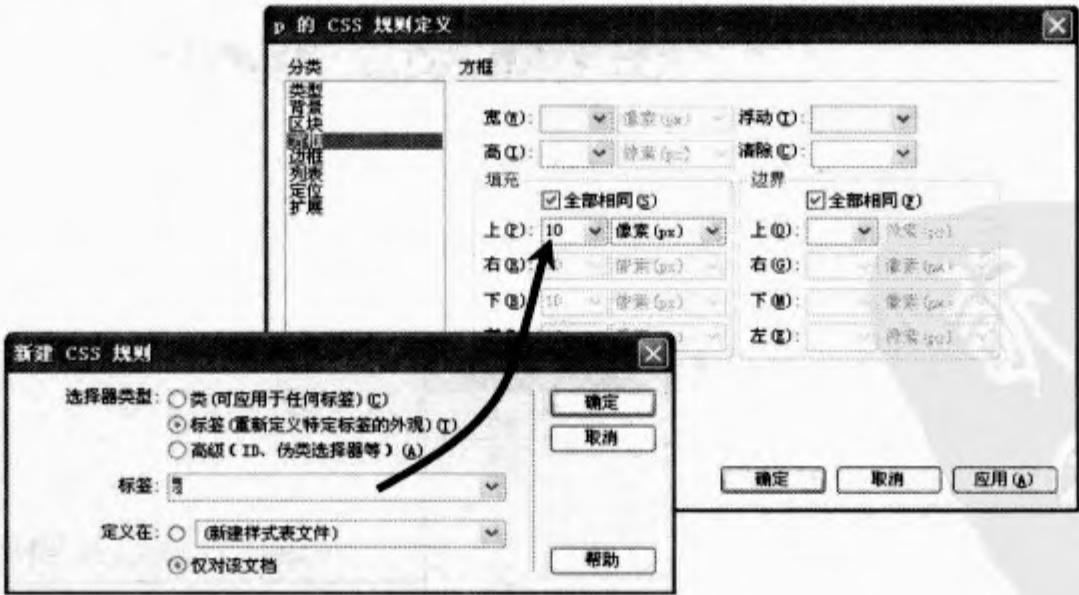


图 2.46 定义段落默认样式

```
p {
    padding: 10px;
}
```

2.4.3 使用【CSS 样式】面板定制页面基本布局

完成页面基本属性的设置，下面我们就利用【CSS 样式】面板定制页面基本布局。整个页面的结构布局比较简单，分为 3 行显示，中间行又分为 2 列（如图 2.47 所示）。

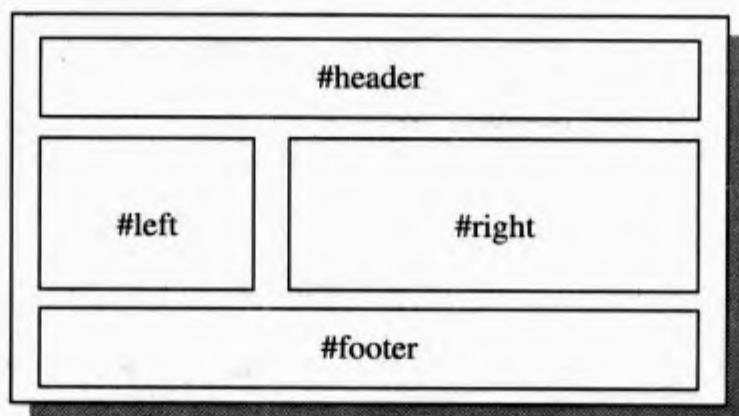


图 2.47 页面基本布局模型

为了实现这个的布局效果，我们可以设计<div id="header">、<div id="right">和<div id="footer">为自然流动显示，即不需要怎么去设置样式。而让<div id="left">向左浮动，这样<div id="right">内的内容就会环绕在其右侧。

具体实现的方法如下：

第 1 步，定义<div id="header">子包含框的样式，由于是流动布局，我们可以不去管它，但是为了方便浏览，这里定义一级标题的样式如下，详细操作如图 2.48 所示。

由于每个属性在不同的类别中，所以读者需要逐个进行设置，这里就不再显示每一步操作对话框。

```
#container #header h1 {
    height:80px;
    line-height:80px;
    margin:0;
    padding-left:10px;
    background: #EEE;
    color: #79B30B;
}
```



图 2.48 定义标题样式

第 2 步，定义左栏向左浮动。先选中左栏元素，然后在【方框】类别中定义向左浮动显示，其他属性可以参照如图 2.49 所示的样式进行详细设置。


```
#container #left {
    float:left;
    width:33%;
    height:100px;
    background:#B9CAFF;
}
```



图 2.49 定义左栏浮动样式

第 3 步，设置右栏自动流动显示，但是为了避免左栏接触紧密，我们可以定义它的左外边距为左栏的宽度，详细代码如下：

```
#container #right {
    height:100px;
    margin-left:33%
}
```

第 4 步，定义页脚包含框的样式。结果如图 2.50 所示，详细样式如下：

```
#container #footer {
    background: #333;
    color: #FFF;
    clear:both;
    width:100%;
}
```

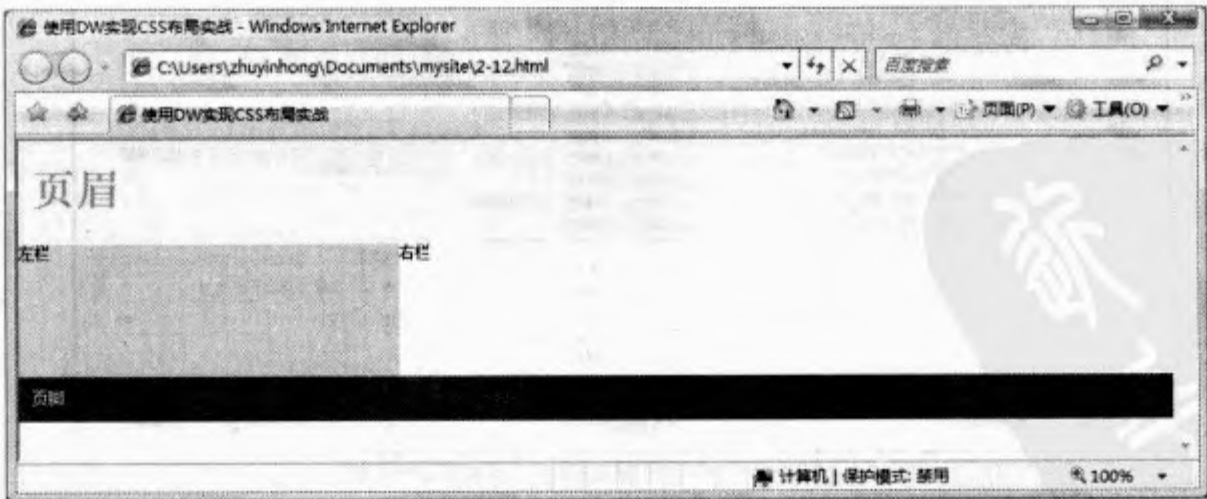


图 2.50 使用 Dreamweaver CS3 定义的布局效果

第 3 章

听话的文字和版式

每当我们用几个通宵去设计完成一个作品之后，再来点击那曾经映衬的页面，你总会感觉小伙伴们设计的作品都很可爱，免不了会热血澎湃的孤芳自赏半天。所以，聪明的设计师都会用心去打理页面中的每个细节，哪怕是一个文字放置不平整，也会为此挑灯夜战。

我在想，你喜欢文字吗？你是一个细心的人吗？本章就从文字开始我们的 CSS 布局之旅。俗话说得好，好的开头是成功的一半，赶紧行动起来吧。

3.1 看懂 Dreamweaver CS3 提供的模板

Dreamweaver CS3 对于 Ajax 和 CSS 技术的支持确实令人佩服。固然 Adobe 所宣传的 Spry 框架很诱人，但是我比较欣赏它所提供的一套标准网页布局模板（如图 3.1 所示），可能你忽视了这个功能，如果真是这样那将是一件很遗憾的事情。在与网页设计师班的学生接触时，我也发现很多同学在学习这块内容时也是一翻即过，没有注意到【新建文档】对话框中蕴藏的“金矿”。

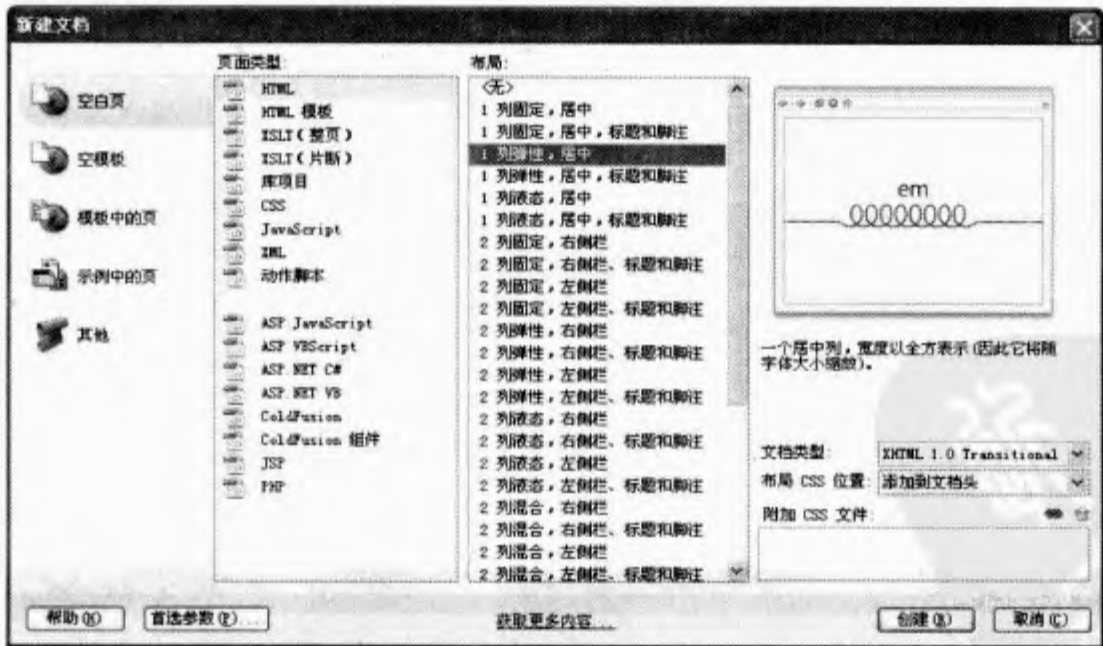


图 3.1 【新建文档】对话框

3.1.1 善于利用示例页

如果你是一位网页设计的高手，那么本节的标题对于你来说就只算是自夸罢了，但对于一位刚入门或即将入门的初学者来说，本节内容至关重要。

【新建文档】对话框提供了两套比较有价值的辅助产品：示例页和标准网页布局模板。

所谓示例页其实就是一些典型的、常用的案例页面，如果你的页面设计风格与某个示例页相同或比较接近，借助示例页创建文档能够帮助节省很多的开发时间。

在 Dreamweaver 8 及以前版本中都或多或少地提供了一些类似的半成品，不过 Dreamweaver CS3 把以前版本中的【新建文档】对话框中提供的这些半成品都收容到“示例页”中（如图 3.2 所示）。原件可以在 Adobe\Adobe Dreamweaver CS3\configuration\BuiltIn 目录下找到。



图 3.2 【新建文档】对话框中的示例页

示例页被分成 4 类：CSS 样式表、框架集、起始页（主题）和起始页（基本）。其中“起始页（主题）”和“起始页（基本）”提供了一些比较完整的示例，不过这些页面依然采用表格布局，与时下流行的标准布局相比较应该说已经落伍了。

“CSS 样式表”对于初学者来说是一个不应忽视的选项，在这里你能够找到很多设计完好的 CSS 样式表，虽然它们不能够百分之百的让你满意，但是这些 CSS 样式表设计得很严谨、很科学，因此可以把它们作为学习、研究的典型案例，也可以用在自己的页面设计中，然后在此基础上进行修改，这比从零开始去设计样式要快捷得多、方便得多。

也许这样抽象的说理你可能体会不到其中的妙处，下面借助一个示例来演示其用法。具体操作步骤如下：

启动 Dreamweaver CS3，新建一个页面，尝试在其中搭建一个简陋的页面框架。

```
<div id="wrap">
  <div id="header">
    <h2>我的标题区</h2>
  </div>
  <ul id="nav">
    <li>菜单 1</li>
    <li>菜单 2</li>
    <li>菜单 3</li>
    <li>.....</li>
  </ul>
  <div class="clear"></div>
  <div id="main">
    <div id="left"><h3>左侧栏目</h3></div>
    <div id="mid"><h3>中间内容区</h3></div>
```

```

        <div id="right"><h3>右侧栏目</h3></div>
        <div class="clear"></div>
    </div>
    <div id="footer">
        <p>我的版权区</p>
    </div>
</div>

```

这是一个固定宽度的三行三列布局页面，此时在没有 CSS 支撑的情况会显示如图 3.3 所示的效果。显然这不符合网页设计的要求，因此我们还需要使用 CSS 把这个网页框架给撑起来，具体代码如下：

```

<style type="text/css">
body { margin:0; padding:0; }          /* 清除<body>标签默认的页边距 */
#wrap {
    width:800px;                        /* 固定页面宽度 */
    margin:0 auto;                      /* 设置页面居中显示 */
}
#header, #footer {
    height:80px;                        /* 固定页头部和脚部高度 */
    border:solid 1px #009900;          /* 增加一个边框线 */
    margin:2px 0;                      /* 增加板块之间的上下间距 */
    text-align:center;                 /* 文本居中 */
    line-height:80px;                  /* 单行文本垂直居中，属性值应与高度相等 */
}
#wrap ul {
    list-style:none;                   /* 清除项目列表符号 */
    margin:0; padding:0;               /* 清除列表项缩进显示 */
}
#wrap li {
    float:left;                        /* 向左浮动，促使列表项并列显示 */
    width:100px; height:30px;          /* 固定列表项浮动块的宽和高 */
    text-align:center; line-height:30px; /* 强制列表文本水平和垂直居中 */
    border:solid 1px #009900;          /* 增加一个边框，以方便浏览 */
}
#left, #mid, #right {
    float:left;                        /* 向左浮动，实现栏目并列平行布局 */
    height:200px;                      /* 固定三列栏目固定高度，以方便浏览 */
    border:solid 1px #009900;          /* 增加一个边框，以方便浏览 */
    margin:2px;                        /* 增加栏目之间的间距，以示区分 */
}
#left {
    width:200px;                       /* 固定左侧栏目的宽度 */
    margin-left:0;                     /* 清除左侧栏目左边距 */
}
#right {
    width:200px;                       /* 固定右侧栏目的宽度 */
    float:right;                       /* 定义右侧栏目向右浮动 */
    margin-right:0;                    /* 清除右侧栏目右边距 */
}
#mid { width:384px; }                 /* 定义中间栏目的宽度 */
.clear { clear:both; }                /* 定义清除浮动类，该类能够避免浮动元素上蹦下跳 */
</style>

```

输入上面的 CSS 代码之后，你会发现刚才的页面框架忽然焕然一新（如图 3.4 所示）。



图 3.3 设计页面框架

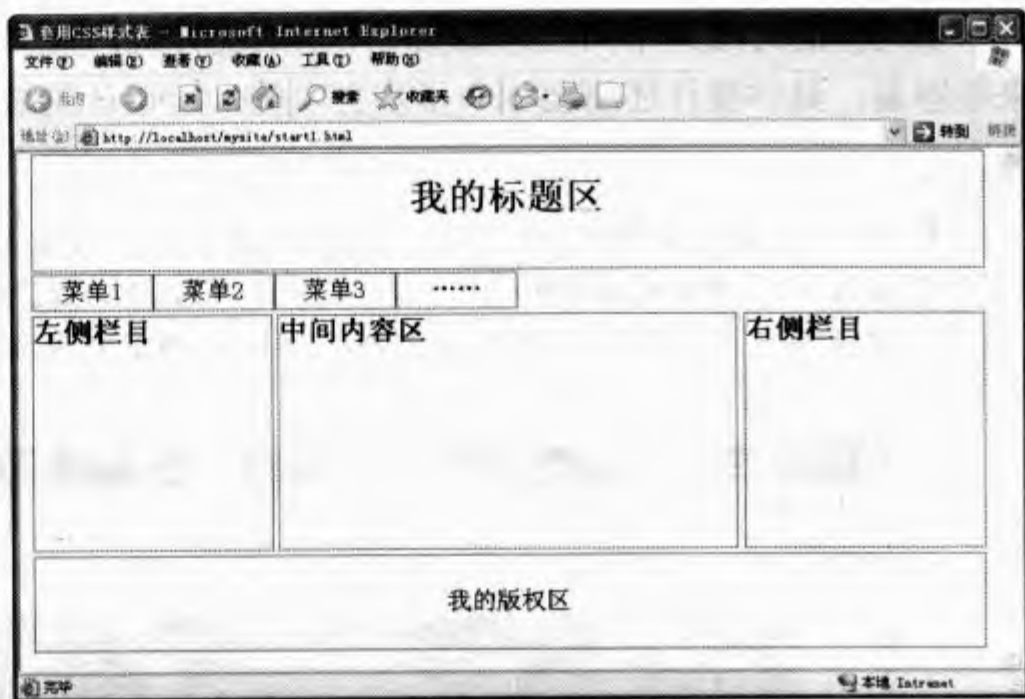


图 3.4 设计页面框架布局

页面框架被搭起来后，下面就等着你为它美化了，也许你还束手无策，不知从何处做起，也不知如何完成这个没有思路的繁琐任务。

此时，本节的内容就派上用场了。你可以在【新建文档】对话框的【示例中的页】|【CSS 样式表】类型下选择一款示例。例如，选择“完整设计：Verdana，黄色/绿色”示例页选项（参考图 3.2 所示），便可以在【新建文档】对话框的最右侧预览该 CSS 样式表的容貌。

当选择一种示例页之后，单击【确定】按钮关闭对话框，此时 Dreamweaver CS3 会自动帮助你建立一个样式表文件，并定义了各种样式。这些样式定义了 body、td、th、a、form 等基本元素的样式，如字体、大小、颜色、版式等属性；还定义了导航条、各种模块和标题的类样式，如字体颜色、大小和背景色等。虽然这些样式都很简单，但是如果自己动手就会感觉很繁琐。

将该文件保存为 style.css，然后在网页中链接 CSS 样式表文件 style.css，代码如下：

```
<link href="images/style.css" rel="stylesheet" type="text/css" />
```

这时你就可以看到：刚才还比较简陋的网页结构瞬间变了面貌。借助 Dreamweaver CS3 提供的可视化操作把这些类样式附加到各个标签之上即可。操作方法如图 3.5 所示。具体方法是在页面中选中一个标签，然后在【属性】面板内的【样式】下拉列表中选择一种类样式即可。

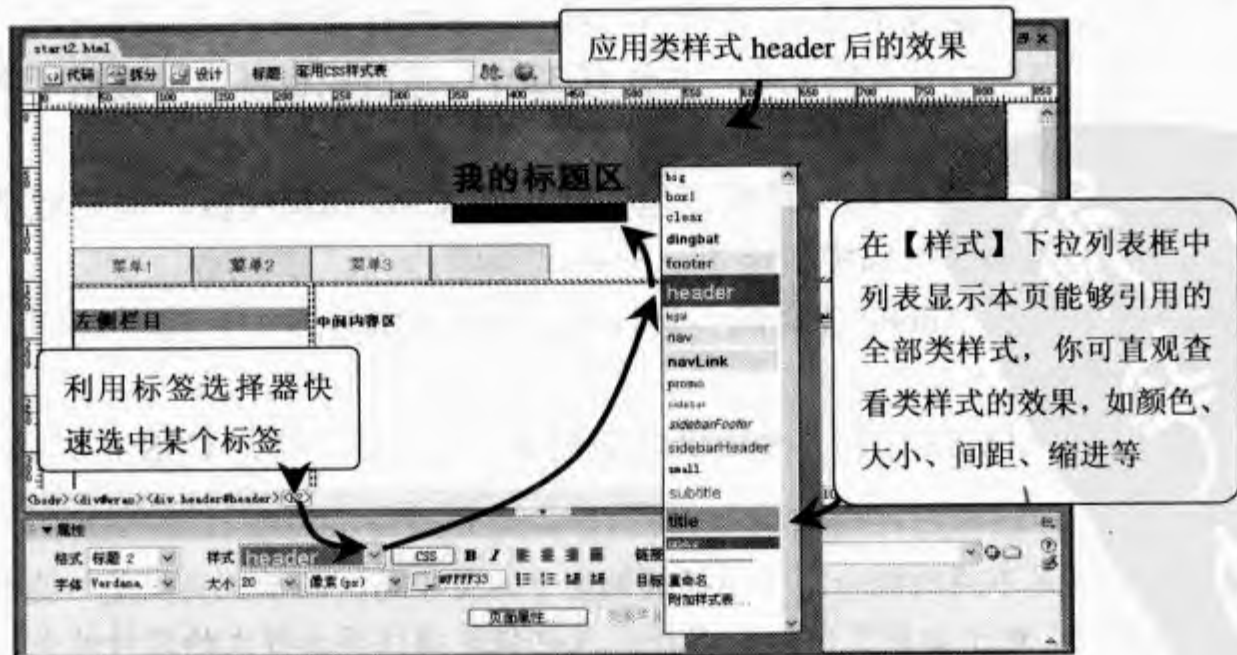


图 3.5 把样式绑定到页面标签中

以同样的方法为不同的标签应用不同的类样式，整个页面的修饰就这样完成了，当然说起来很容易，具体操作还需要你一步步去实施，所设计的效果如图 3.6 所示。

提示

【新建文档】对话框所提供的“CSS 样式表”示例页是不支持网页布局的，也就是说它不能够帮助你实现网页结构的搭建和支撑（即布局）。“CSS 样式表”示例页仅提供页面字体、文本的颜色、大小、字体等属性的配置，以快速实现某种风格的页面效果。

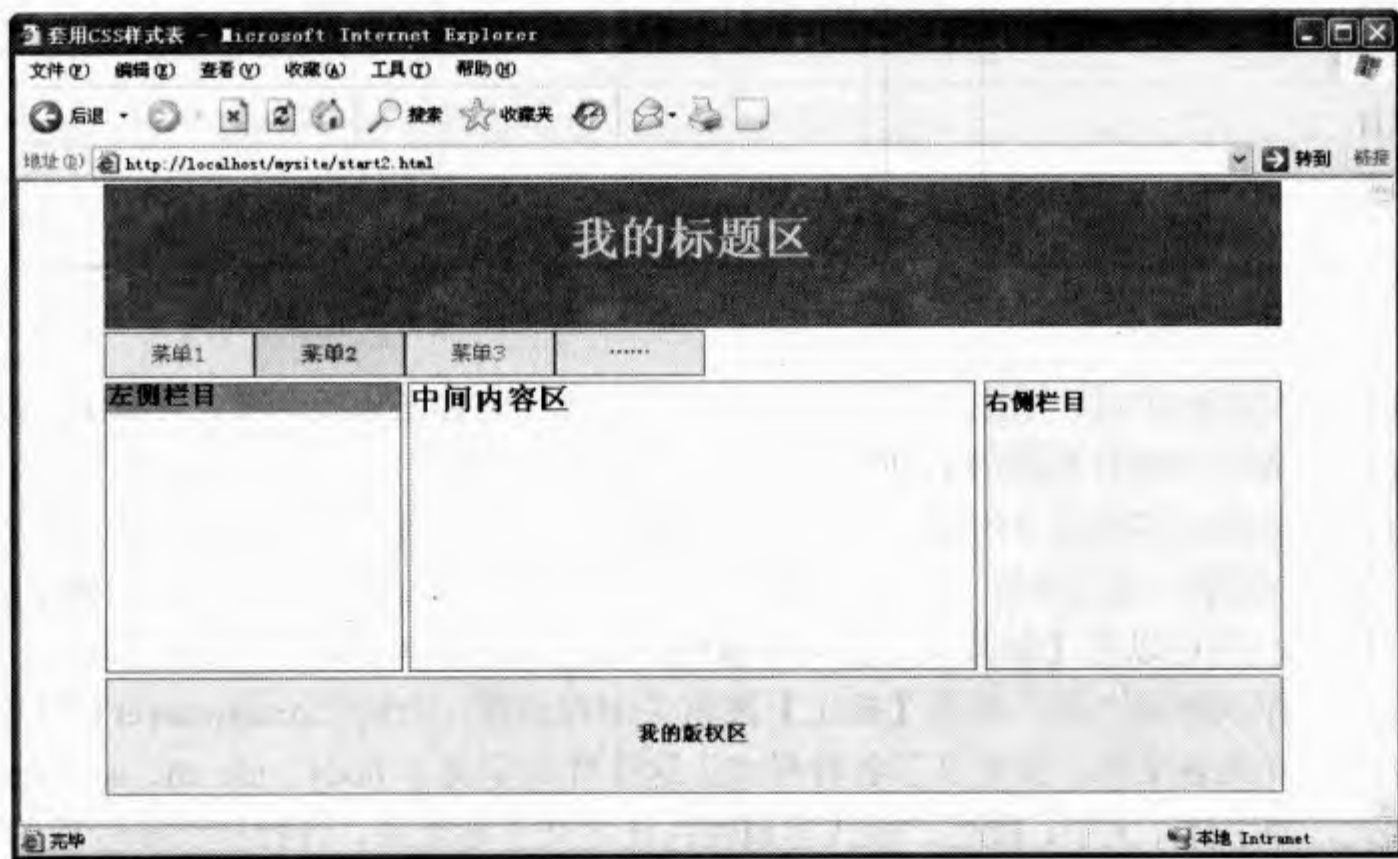


图 3.6 应用了“CSS 样式表”示例页的 HTML 框架

一个简单的页面就这样诞生了，虽然很简单，但是也能够从中找到学习的兴趣和动力。相信你一定不会满足于这样的页面效果，你完全可以在 Dreamweaver CS3 提供的示例样式表的基础上不断去完善和修改。这样的试验很值得你去尝试和摸索，只有这样才可以学到更多技巧。

3.1.2 挖掘标准网页布局模板的价值

在图 3.1 所示的【新建文档】对话框中显示了各种标准网页布局模板，这些模板是 Dreamweaver CS3 为用户精心开发的礼物，共计有 32 个，这些模板基本上囊括了常用网页布局模型。

你可能在网上也看到了各种网页模板，包括标准的 CSS 布局模板，但是似乎它们都没有 Dreamweaver CS3 提供的这套布局模板珍贵。分析其原因：

- Dreamweaver CS3 是 Adobe 公司倾力设计的，应该比较标准、严谨和稳健，能够兼容不同浏览器，经得起考验；
- 这些模板全部使用 CSS 设计，并提供非常翔实的注解，是学习 CSS 布局最珍贵的资料。

提示

布局模板与示例页是不同的概念。布局模板侧重于网页布局，而示例页仅是一些典型的案例，不具备普遍套用的价值，对于示例页中的 CSS 样式表也仅是提供页面样式的设计效果，没有涉及如何去进行网页布局。

例如，针对上一节设计的固定宽度的三行三列布局页面，我们使用布局中的“3 列固定，标题和脚注”模板来快速实现（如图 3.7 所示）。



图 3.7 选择一种标准网页布局模板

单击【确定】按钮关闭对话框，则 Dreamweaver CS3 自动套用该模板建立一个页面，页面框架和布局都已自动完成（如图 3.8 所示）。如果你是一位善于学习的读者，一定会翻看该模板的 HTML 源代码和 CSS 布局代码，有些地方可能看不懂，但是一定会发现 Dreamweaver CS3 设计的 HTML 框架和 CSS 布局代码要比上节我设计的示例棒多了。



图 3.8 标准网页布局模板完成的页面效果

余下的工作就等着你在这个布局框架中填充了。也许你不喜欢模板所提供的灰色调效果，但是你可以把上节示例中新建的“完整设计：Verdana，黄色/绿色”CSS 样式表示例页导入到该页面中。导入前建议先切换到【代码】视图，删除模板所定义的所有 background 属性声明的规则，然后按照上节介绍的方法为不同区域应用不同的类样式即可。

【新建文档】对话框中的标准网页模板很多，不过归类之后也不过几类。例如，根据网页结构来分，可以分为 1 列布局、2 列布局和 3 列布局，然后每类又包括单行、双行或多行等。如果根据布局类型分，则包括固定宽度布局、弹性宽度布局、流动宽度布局和混合宽度布局，对于这些不同类型的布局我们将在下面章节中继续讲解。

3.2 你的文字听话吗

你的文字听话吗？这样问稍显有点突然，但确实反映了当前国内网页设计中普遍存在的问题。如果没有说错的话，你一定使用过如下样式定义网页的字体大小：

```
body { /* 定义网页字体大小为 12 像素 */
    font-size:12px;
}
```

这可能是一种设计习惯吧，或者说是一种盲从，包括我自己也一样：经常习惯性地输入上面 CSS 代码定义网页字体的大小。

这样写有错吗？当然没有错误。但是如果你与我一起继续往下阅读，也许会慢慢发现这样的设计很不妥当，甚至感觉很尴尬。

3.2.1 国际雅虎和中国雅虎

雅虎 (<http://cn.yahoo.com/>)，这个昔日叱咤互联网的英雄，无疑是我们这些后来者学习的楷模。这里仅针对雅虎 (<http://www.yahoo.com/>) 和中国雅虎 (<http://cn.yahoo.com/>) 在浏览中一个细节说说自己的感受。

请使用 IE 浏览器打开雅虎 (<http://cn.yahoo.com/>) 首页，局部显示效果如图 3.9 所示。也许你的视力欠佳，或者你的屏幕分辨率很大，页面文字显示很小。此时，建议你不妨在 IE 浏览器的菜单栏中勾选【查看】|【文字大小】|【最大】菜单项，则页面文字被放大显示（如图 3.10 所示）。



图 3.9 正常显示的雅虎首页



图 3.10 放大显示的雅虎首页

但是，回到中国雅虎 (<http://cn.yahoo.com/>) 的首页（如图 3.11 所示），你可能就没有这样的福分了。无论浏览器的文字大小设置多大或多小，页面中的文字依然纹丝不动（如图 3.12 所示）。

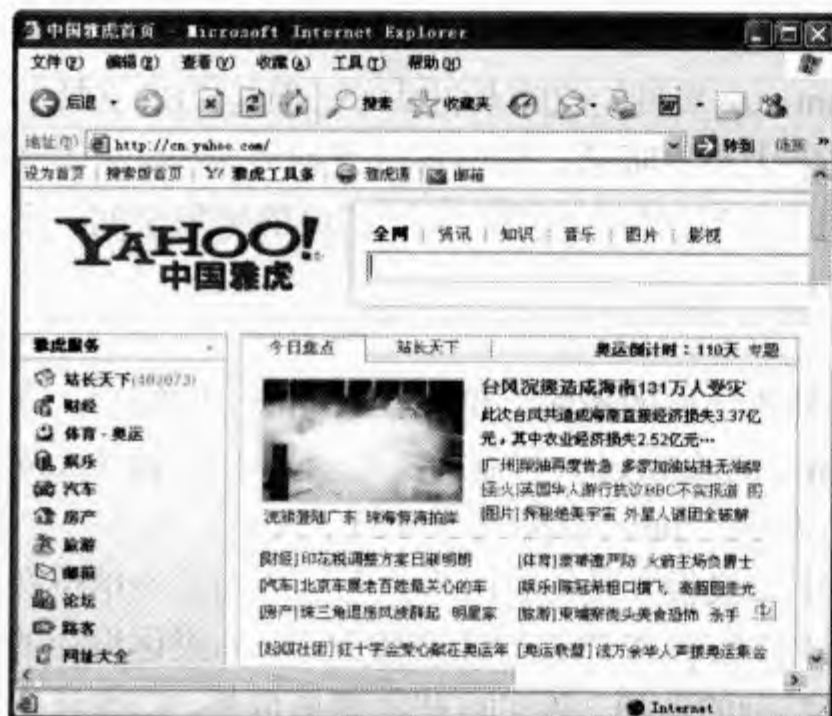


图 3.11 正常显示的中国雅虎首页



图 3.12 放大显示的中国雅虎首页

细心地翻看雅虎 (<http://cn.yahoo.com/>) 首页的 CSS 样式表，其中第一个样式就是定义网页字体和大小，代码如下：

```
body{
    font:13px arial,helvetica,clear,sans-serif;
    *font-size:small;
    *font:x-small;
}
```

上面样式使用了样式过滤器技术（也称为 Hack）来兼容不同的浏览器，以实现在 IE 6 及以下版本浏览器与其他标准浏览器使用不同的单位来定义字体大小。

其中 `*font-size:small;` 规则只能够在 IE 6 及以下版本浏览器中有效，而在 Mozilla Firefox 等标准浏览器中是无效的。

考虑到 Mozilla Firefox 等浏览器放大或缩小是根据像素来实现的，这犹如放大或缩小图片，所以把字体大小的取值单位设置为像素时不会影响浏览器的文字缩放功能。而对于 IE 浏览器来说，当网页文字以像素为单位时，浏览器提供的网页缩放功能就无效，因此这里采用了 CSS 浏览器兼容技术，设置网页字体在 IE 中以关键字为单位来设置字体大小。

关键字 `small` 在 CSS2 版本中大小相当于 13 像素，类似的关键字还有：`xx-small`（=9px）、`x-small`（=11px）、`medium`（=16px）、`large`（=19px）、`x-large`（23px）和 `xx-large`（=27px）。使用关键字为单位的好处就是网页字体能够在 IE 浏览器中被放大或缩小。

再翻看中国雅虎首页的 CSS 样式表，它也是在第一个样式中定义字体大小，代码如下：

```
body{
    background-color:#fff;
    font:12px arial,simsun,sans-serif;
    text-align:center;
}
```

雅虎中国的设计师们就很习惯性的使用像素来定义网页内所有字体。如果细心的比较一下国内和国外一些大网站，就会发现国外设计师们很喜欢使用 `em`、百分比、关键字等单位来设置网页字体，而国内设计师们基本上习惯性的选择 `px`（像素）或 `pt`（点）为单位设置网页字体大小。

3.2.2 选择好单位让你的文字更听话

在上节示例中，简单了解了在 CSS 中使用 font-size 属性定义字体的大小。但是 CSS 支持的字体大小的单位可不少，如果希望设计的网页文字大小更听话，你应该好好研究一下。

CSS 提供了丰富的单位，这些单位为设计师更灵活的设计页面提供了广阔的想象空间。不过这些单位可以简单地分为两大类：绝对大小和相对大小。

1. 设置字体大小为绝对值

所谓绝对大小就是字体大小是固定的，不受外界环境的影响。例如：in (inch 的缩写，表示英寸)、cm (centimeter 的缩写，表示厘米)、mm (millimeter 的缩写，表示毫米)、pt (point 的缩写，印刷中所谓的点数，1pt 等于 1/72inch) 和 pc (pica, 1pc 等于 12pt)。

这些绝对单位多用在传统印刷中，网页设计中一般很少使用，因为你无法预知这些单位在不同显示器上如何显示。不过有小部分设计师喜欢使用 pt 单位来设计字体大小，如果你设计的样式准备在打印机中输出时，使用 pt 单位是比较精确的选择。除了这些物理单位外，CSS 还提供了 7 个关键字，这些关键字在 CSS 中是固定的，但它能够根据浏览器的字体大小进行调整，具体说明请参考上一节的介绍。

2. 设置字体大小为相对值

所谓相对大小就是文字的大小不是固定的。相同的字体大小设置可能在这个显示器中显示很大，而在另一个显示器中会显示得很小，或者在不同浏览器与页面不同位置所显示的大小效果也是不同的。这种相对性给网页设计带来灵活性，自然也成为绝大部分设计师青睐的选择。

px 是 pixel 的缩写，翻译为像素，它是图像处理中最常用的单位。因为图像以像素点为单位进行计算，而显示器的屏幕大小也是以像素点来计算的，因此设置字体大小的单位为像素后，更容易与显示器相匹配，这也是设计师最喜欢选用 px 作为字体大小的根本原因。

不过 px 显示大小会受到分辨率的影响，这也为什么说 px 是相对大小单位。分辨率越大，相同设置的字体就会越小，反之就会越大。

但是考虑到绝大部分用户的电脑屏幕分辨率多是 1024×768 像素，也有很少用户的电脑屏幕分辨率是 800×600 像素或 1280×1024 像素。因此使用 px 作为单位定义网页字体大小时，实际上所显示的效果也是一种静止的单位。也就是说你如果把网页字体设置为 12 像素，那么其显示的大小效果在不同浏览器窗口、不同的网页位置都是相同的。网页设计师如此喜欢 px，大概就是它的忠贞吧，即提供一致性视觉效果。

当然以 px 作为网页文字的单位也带来一定的问题，这个问题主要存在于 IE 6 及以下版本浏览器中。因为这些版本浏览器不会自动调整以像素为单位的字体大小。不过到 IE 7 版本开始，包括其他标准浏览器都能够调整像素为单位的字体大小，如果当 IE 6 版本浏览器退出历史舞台，或跟 IE 5 一样极少被人使用时，你可能就不用费这个脑筋了。

百分比 (%) 和 em 都是根据父元素的大小来决定自己大小的相对单位，它们在表示字体大小单位时，所表示的意思和显示效果是基本相同的。

如果父元素没有定义字体大小，则会自动向上继承祖先元素的字体大小。如果所有祖先元素都没有定义字体大小，则根据浏览器默认字体大小来进行设置，所有浏览器的默认字体大小为 16 像素。

初次接触百分比 (%) 和 em 时，很多读者可能会存在疑惑和不解。例如，在下面示例中，定义每层字体的大小都为 2em，即显示为 2 倍于父级元素的字体大小，则显示效果如图 3.13 所示，这与用 px 定义字体大小的效果截然不同。浏览器的默认字体大小为 16 像素，对于 <div

id="level1">标签来说,其包含的文字将是默认字体的2倍,即32像素;对于<div id="level2">来说,其包含的文字大小将是父标签<div id="level1">字体大小的2倍,即64像素。以此类推,最里层包含的文字大小将是128像素。

```
<style type="text/css">
#level1,#level2,#level3 {
    font-size:2em;
}
</style>
<body>
    默认字体
    <div id="level1">2 倍字体
        <div id="level2">2 倍字体
            <div id="level3">2 倍字体</div>
        </div>
    </div>
</body>
```



图 3.13 标准网页布局模板完成的页面效果

如果不定义<div id="level1">和<div id="level2">标签内字体大小,根据 CSS 继承规则,则这些标签的文字大小都继承浏览器默认字体大小(16 像素),因此最里层包含的文字大小将以浏览器默认字体来进行计算,即为 32 像素。

em 和百分比单位在作用和效果上都是相同的。例如,1em 等于 100%,相当于上级元素字体的大小;而 0.5em 就等于 50%,表示为上级元素字体的 0.5 倍;同理,2em 就等于 200%,表示为上级元素的 2 倍。

虽然 em 和百分比作为单位所显示的效果是相同的,但是设计师更喜欢使用 em 作为单位。因为它源于印刷样式,指相对于一种特定字体中一个大写字母 M 的尺寸。不过随着它被引入 Web 设计领域,em 在 CSS 中就不再表示这个意思,作为标准的文字大小尺寸,1em 就相当于 100%,都等于父级元素的字体大小。

3.2.3 设计网页字体大小配置方案

利用 em 和百分比作为网页字体大小的单位,你可以设计出一套听话的网页字体大小方案。例如,假设你计划这样设计自己的网页字体大小配置方案:

- 网站标题字体大小为 16 像素;

- 栏目标题字体大小为 14 像素;
- 导航菜单字体大小为 13 像素;
- 正文字体大小为 12 像素;
- 版权、注释信息字体大小为 11 像素。

网页的 HTML 框架代码如下:

```
<div id="wrap">
  <div id="header">
    <h1>网站标题 (<span style="font-size:16px;">网站标题-16px</span>) </h1>
  </div>
  <ul id="nav">
    <li>菜单 (<span style="font-size:13px;">菜单-13px</span>) </li>
  </ul>
  <div id="main">
    <h2>栏目标题 (<span style="font-size:14px;">栏目标题-14px</span>) </h2>
    <p>网页正文 (<span style="font-size:12px;">网页正文-12px</span>) </p>
  </div>
  <div id="footer">
    <p>版权信息 (<span style="font-size:11px;">版权信息-11px</span>) </p>
  </div>
</div>
```

首先, 定义网页字体大小。 $(12\text{px}/16\text{px}) \times 1\text{em} = 0.75\text{em}$, 也就是说初始化网页字体大小为 0.75em (相当于 12 像素), 代码如下:

```
body {
  font-size:0.75em;
}
```

以 body 元素的字体大小为参考, 来定义其他栏目或版块的字体的大小。

- 网站标题的字体大小: $(16\text{px}/12\text{px}) \times 1\text{em} = 1.333\text{em}$ 。也就是说网站标题的字体大小是 body 字体大小的 16/12 倍, 即等于 1.33em。

有的读者可能会迷糊了: 为什么不是 $(16\text{px}/12\text{px}) \times 0.75\text{em} = 1\text{em}$, 因为 body 的字体大小被定义为 0.75em 呀?

是的。但是我们前面已经讲解了, 子元素的字体大小都是以父元素的字体大小为 1em 作为参考来计算的, 也就是说如果网站标题定义为 1em, 而 body 字体大小为 0.75em, 则网站标题也应该为 0.75em, 即等于 12px, 而就不是 16px 了。

- 栏目标题的字体大小: $(14\text{px}/12\text{px}) \times 1\text{em} = 1.167\text{em}$ 。也就是说栏目标题的字体大小是 body 字体大小的 14/12 倍, 即等于 1.167em。
- 导航菜单的字体大小: $(13\text{px}/12\text{px}) \times 1\text{em} = 1.08\text{em}$ 。也就是说栏目标题的字体大小是 body 字体大小的 13/12 倍, 即等于 0.812em。
- 正文的字体大小: $(12\text{px}/12\text{px}) \times 1\text{em} = 1\text{em}$ 。也就是说正文的字体大小是 body 字体大小的 1 倍, 即等于 1em。
- 版权、注释信息的字体大小: $(11\text{px}/12\text{px}) \times 1\text{em} = 0.917\text{em}$, 也就是说版权、注释信息的字体大小是 body 字体大小的 11/12 倍, 即等于 0.917em。

所以针对上面的 HTML 结构, 定义的 CSS 样式如下。其中正文字体直接继承 body 元素的字体大小, 因此就不需要重复定义。在 IE 6 中演示效果如图 3.14 所示, 在 Firefox 2 中演示效果如图 3.15 所示。


```
<style type="text/css">
body { font-size:0.75em; }
#header h1 { font-size:1.333em; }
#main h2 { font-size:1.167em; }
#nav li{ font-size:1.08em; }
#footer p { font-size:0.917em; }
</style>
```



图 3.14 在 IE 6 中预览网页字体大小配置方案



图 3.15 在 Firefox 2 中预览网页字体大小配置方案

很明显,上面的配置方案在 IE 6 中没有正确的解析,而在 Firefox 2 中可以看到使用 em 作为单位与 px 作为单位的字体大小是按原计划的效果来解析的。问题何在呢?是不是我们的设计方案有问题,还是浏览器的兼容性问题?

原来这是 IE 6 及以下版本浏览器存在的一个 Bug。当页面内只用 em 作为单位时,IE 6 及以下版本在显示时不能够按规矩出牌,导致页面字体被解析时会出现一些问题。

解决的方法:设置 body 元素的字体大小单位百分比,避免页面全部使用 em 作为单位。例如,修改上面示例中 body 元素的字体大小样式,这样页面字体大小的显示效果就与 Firefox 浏览器中显示效果一样了(如图 3.16 所示)。

```
body {
    font-size:75%;
}
```

此时,你在任何浏览器中缩放字体都能够随意地放大或缩小(如图 3.17 所示)。

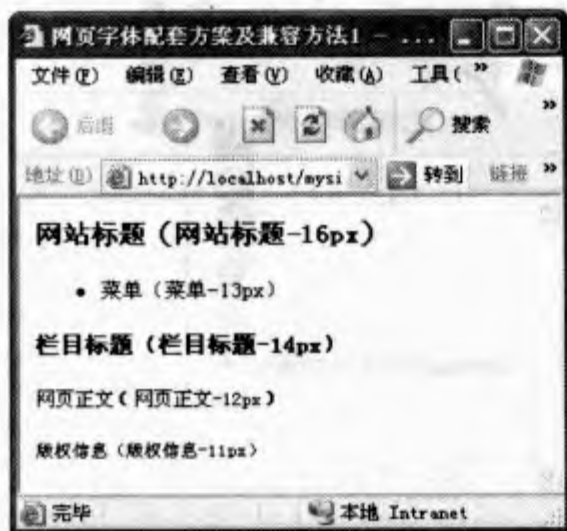


图 3.16 解决 IE 6 中的 Bug



图 3.17 能够听话的网页字体

对于上面的字体大小解决方案,适合嵌套层次比较少的字体大小继承中,且要注意相互的

干扰性。例如，如果创建一个样式 `ol {font-size:60%;}`，那么当在列表嵌套中就会出现严重问题，内部的``标签所包含的字体会实际显示为 36% (60%*60%)。所以，在使用 `em` 为单位定义字体大小时，要考虑网页结构的层次问题，原则上不要嵌套使用 `em` 为单位定义字体大小超过 2 层，否则会为网页字体大小的统筹设计带来很多麻烦。

3.2.4 设计与字体大小相关联的网页布局

在第 3.1.2 节中讲解标准网页布局模板时，曾经提到过布局模板类型，如固定宽度布局、弹性宽度布局、流动宽度布局和混合宽度布局，但是没有详细讲解。下面先来认识【新建文档】对话框中这些模板布局的特点。

(1) 固定宽度布局：网页宽度是固定的，所有栏目的宽度都采用以像素为单位进行设置。布局框架效果如图 3.18 所示。在 Dreamweaver CS3 所提供的模板图样中以小锁图标 (🔒) 来表示。

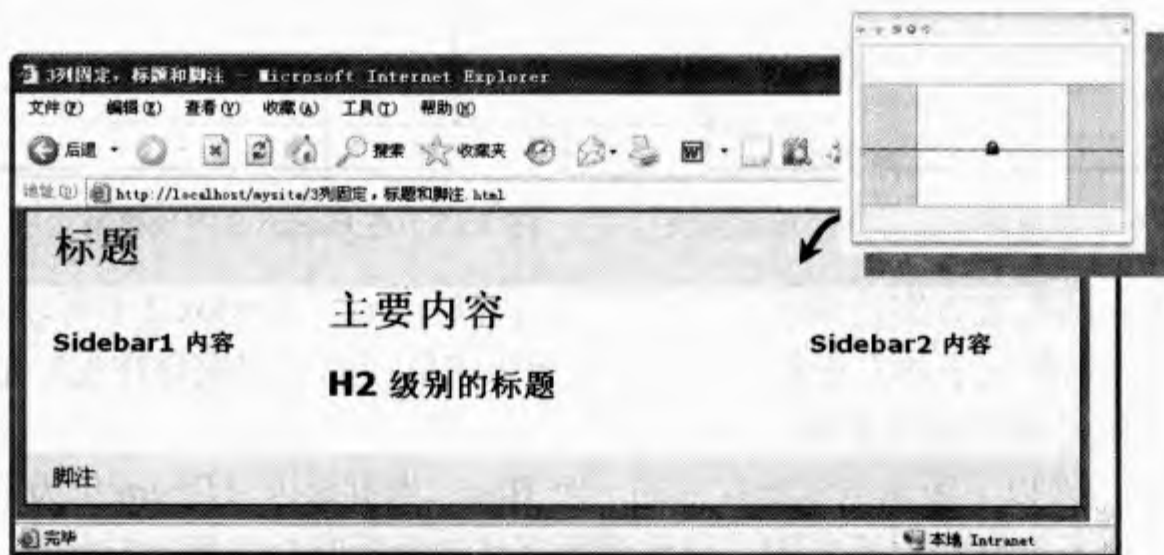


图 3.18 “3 列固定，标题和脚注”布局模板效果

(2) 弹性宽度布局：网页宽度是弹性的，所有栏目的宽度都采用以 `em` 为单位进行设置，这样当页面字体大小发生变化时，网页宽度也随之变化。布局框架效果如图 3.19 所示。在 Dreamweaver CS3 所提供的模板图样中以弹簧加“`em`”字母图标 (🔍`em`) 来表示。

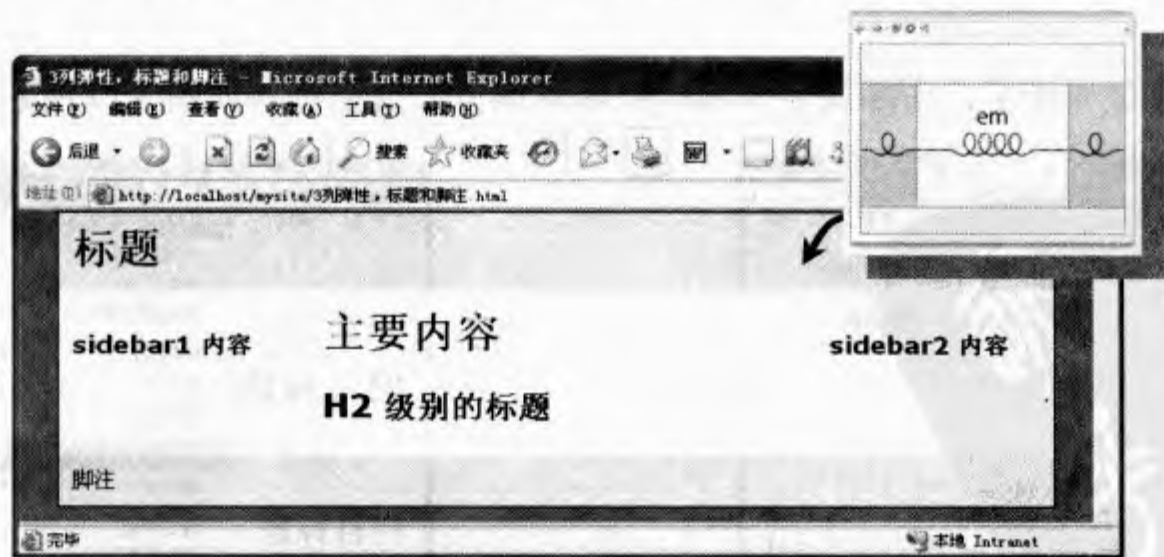


图 3.19 “3 列弹性，标题和脚注”布局模板效果

(3) 液态宽度布局：网页宽度是液态的，所有栏目的宽度都采用以百分比为单位进行设置，这样当浏览器窗口的大小发生变化时，网页宽度也随之变化。布局框架效果如图 3.20 所示。在 Dreamweaver CS3 所提供的模板图样中以弹簧加“`%`”字符图标 (🔍`%`) 来表示。



图 3.20 “3 列液态，标题和脚注” 布局模板效果

(4) 混合宽度布局：网页宽度混合使用多种单位。例如，网页总宽度为百分比，而侧栏宽度采用固定宽度等。

请注意，当页面宽度采用百分比和 em 作为单位时，它们的作用和表现效果是不同的，这与字体大小中百分比和 em 单位表现截然不同。当宽度设置为百分比时，它的宽度将以父元素的宽度作为基础进行计算，这与字体大小中的百分比和 em 单位计算方式类似，但是如果宽度设置为 em，则它将以内部包含字体的大小作为基础进行计算。例如，在下面这个示例中，不管字体缩放多大，字体总是在一行内显示，如图 3.21 所示。

```
<style type="text/css">
#left {
    font-size:0.875em;          /* 字体大小为 14 像素 */
    width:12em;                 /* 定义元素的宽度为 12 个字体长度 */
    border:solid 1px red;       /* 定义一个边框，以方便观察显示效果 */
    height:1em;                 /* 定义高度为 1 个字体大小 */
}
</style>
<div id="left">字体大小与网页布局关联</div>
```

如果使用 width:32%;来定义元素的宽度，则它只能根据浏览器的窗口宽度来决定自己的宽度，如果在窗口大小不变的情况下放大字体，就有可能超出包含框的宽度，如图 3.22 所示。



图 3.21 以 em 为单位定义宽度

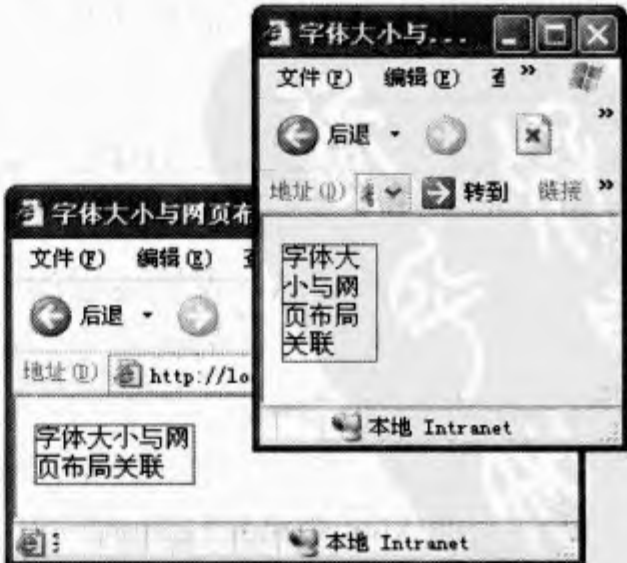


图 3.22 以百分比定义宽度

本节不仅仅是想说明这几种布局类型的特点和用法。更主要的是想引起读者注意并探讨一个话题：如何把字体大小与网页布局关联起来，以使设计的页面更具人性化。

我们先通过一个示例来感受字体大小与网页布局关联的重要性。这是 Adobe 首页 (<http://www.adobe.com/>) 的一个局部区域（左侧产品分类列表栏），在 IE 中正常显示如图 3.23 所示，产品名称都能够很好的在一行内显示，这样用户一眼就能够看明白意思。但是如果把浏览器字体放大（如图 3.24 所示），这时你会发现由于字体被放大，而栏目宽度不变，则一个产品名称被挤开为多行显示，影响了用户阅读速度。

正所谓智者千虑必有一失。如果 Adobe 的网页设计师把产品列表栏目的宽度设置为以 em 为单位，则当字体被放大时，栏目的宽度也跟着被加宽，这样所显示的产品名称总是在一行内显示，也就是 Dreamweaver CS3 所说的弹性布局。

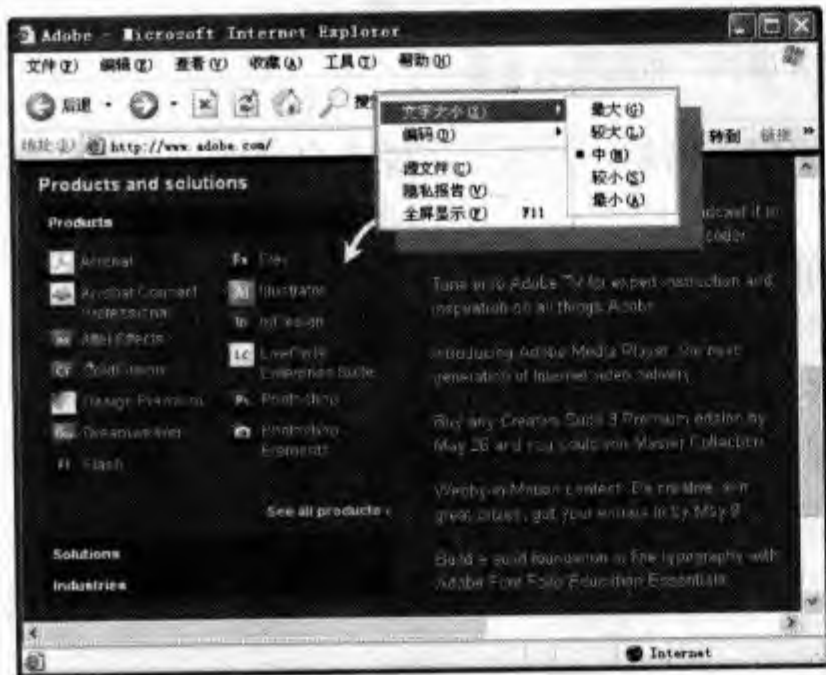


图 3.23 正常模式下显示效果

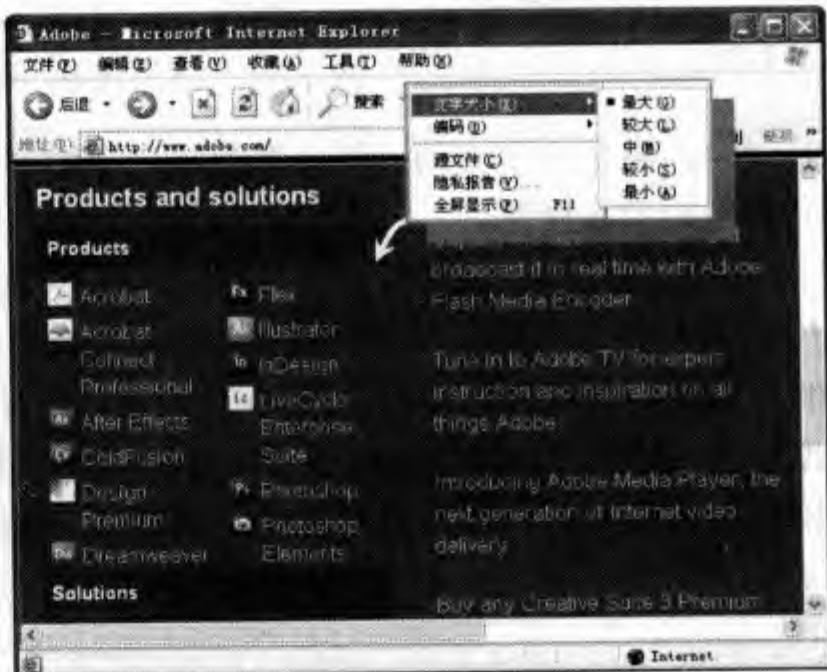


图 3.24 放大字体后显示效果

当然并不是所有栏目都采用弹性布局就是上策。你可以根据需要，觉得某个栏目的文本列表最好保持在一行内显示，则建议使用弹性布局，采用 em 作为单位定义栏目的宽度。其实对于你来说，最好的老师就是好好研读 Dreamweaver CS3 在【新建文档】对话框中提供的各种弹性布局模板，相信你能够从中受到很多启发。但遗憾的是，随意浏览国内网站，不管是大网站还是小网站；也不管是个人站点还是颇具规模的商业网站，都无一例外的把字体大小设置为像素单位，对于栏目宽度都采用像素或百分比。这样设计本没有错，而且还省心，但是却给部分用户的浏览带来负面影响。而相比较，国外设计师们普遍注意这些用户体验的细节问题。

3.3 留心，不要让你的文字“露怯”

文字是用来传播信息、表情达意的，自然易于阅读的字体也是最容易传播信息的。本来这对于任何一位学习网页设计的读者来说都应该不是问题。但是很多时候，由于设计师的疏忽或大意而让手下的字体“露怯”不少。

如果你用 Firefox 浏览器随意在网上浏览网页，会发现很多这样的问题。如你可以看看悬赏中国 (<http://www.xuanshangcn.com/>)，在 IE 6 下浏览如图 3.25 所示。可是在 Firefox 2 浏览器中浏览，你的眼睛可能就受不了（如图 3.26 所示）。



图 3.25 IE 6 下浏览效果



图 3.26 Firefox 2 下浏览效果

这是为什么呢？原来是设计师疏忽了定义字体样式才导致了这样的失误。

3.3.1 不要以为多此一举

在 CSS 中定义字体可以使用 `font-family` 属性来设计。例如，输入下面样式代码就可以把网页字体设置为宋体：

```
body {
    font-family: "宋体";
}
```

由于 IE 浏览器默认字体是宋体或新宋体，即使不定义网页字体属性，浏览器根据默认设置都能够正常的显示，但是在 Firefox 等其他标准浏览器中，由于其默认字体不同，以及它们对于字体解析的方法不同，对于那些没有定义字体属性的页面会根据英文字体显示，这时你就会看到类似如图 3.26 所示的不美观的字体效果，这严重地影响了浏览者猎取信息的速度。所以当你在为自己的网页设计 CSS 样式表时，请一定要在 `body` 元素中增加一个显式定义字体的样式。

3.3.2 让网页字体显示更灵活

对于国内用户来说，基本所有计算机都安装了“宋体”字体，这也是 Windows 中文版系统默认的字体。但是如果你的网页还希望被外国朋友浏览，或者你的网页不仅仅是中文汉字，那么定义一个“`font-family: "宋体";`”这样一个简单的规则未免有些机械。

不过 `font-family` 属性可以同时声明任何一种字体，你不妨在一个属性中列举多种字体，以备所有用户在各自计算机中都能够正确显示网页文字，例如：

```
body {
    font-family: "宋体", Arial, Helvetica, sans-serif ;
}
```

多种字体之间使用逗号分隔开。如果字体名称中包含有空格，可以为字体增加双引号，避免解析时出现混乱，例如：

```
body {
    font-family: "宋体", "Times New Roman", Times, serif ;
}
```

对于英文字体来说，IE 浏览器默认字体为 Arial，而如果要定义大段英文的字体，建议设

置为 serif 会更容易阅读。

除了使用 font-family 专用属性来定义网页字体，你还可以使用文字通用属性来定义字体，例如：

```
body {
    font: "宋体", Arial, Helvetica, sans-serif ;
}
```

当然在该通用属性内，你还可以为网页文字定义大小、样式、行高等属性，例如：

```
p {
    font: italic small-caps bold 12px/1.6em 宋体;
}
```

上面样式定义段落文本为斜体、大写、加粗显示，字体大小为 12 像素，行高为字体大小的 1.6 倍，字体为“宋体”。通用属性中多个属性值以空格分隔开，此时你明白了为什么当字体属性值包含空格时必须加引号的原因。

3.3.3 别忘了通用字体

往往设计师在设计时一相情愿的去思考问题，那样会带来很多潜在的麻烦。事实上你可以为网页设计任意字体，包括艺术字体。也许你的计算机中已经安装了很多字体，但是不能够保证所有浏览者的系统都已经安装了相同的字体。因此，当你非常想使用各种个性字体或艺术字体美化页面时，你可以使用下面的方法来解决此类矛盾。

一种方法是在 font-family 属性设置你喜欢的字体，同时也定义一些备用字体，可以参考上面的样式代码。如果浏览者的系统中安装了指定的字体，则你就会看到漂亮的页面，如果没有，浏览器会自动搜索在 font-family 属性中指定的字体列表，直到找到备用字体为止。

另一种方法是为 font-family 属性指定通用字体。所谓通用字体，就是它表示一类字体，这样浏览器就能够根据你指定的字体类别，从本地系统中找到类型相同的字体来解析网页文字。

也许你看到的字体成千上万，其显示的效果和风格也千差万别，例如，打印体、艺术体、手写体、象形体、卡通体、古典体等。但是几乎所有的字体都可以归为两类：serif 和 sans-serif。

- serif 类型的字体是成比例有衬线的字体。成比例是指字体中所有字母根据它们不同的尺寸占据不同的宽度。例如，字母 l 和 m 所占的宽度是不同的。衬线是指字体上附加的装饰性细线，例如，小写字母 l 会在顶部和底部附加细小的“手”和“脚”。常用 serif 类型字体有 Times、Times New Roman、Georgia 等。
- sans-serif 类型的字体是有比例但没有衬线的字体。常用 sans-serif 类型字体有 Arial、Helvetica 等。例如，在下面代码中，我们可以直观的比较 serif 和 sans-serif 类型字体的显示效果（如图 3.27 所示）。

```
<style type="text/css">
.p1{
    font:36px "Times New Roman", Times, serif;
}
.p2{
    font:36px Arial, Helvetica, sans-serif;
}
</style>
<p class="p1">The World Wide Web Consortium — serif </p>
<p class="p2">The World Wide Web Consortium — sans-serif </p>
```

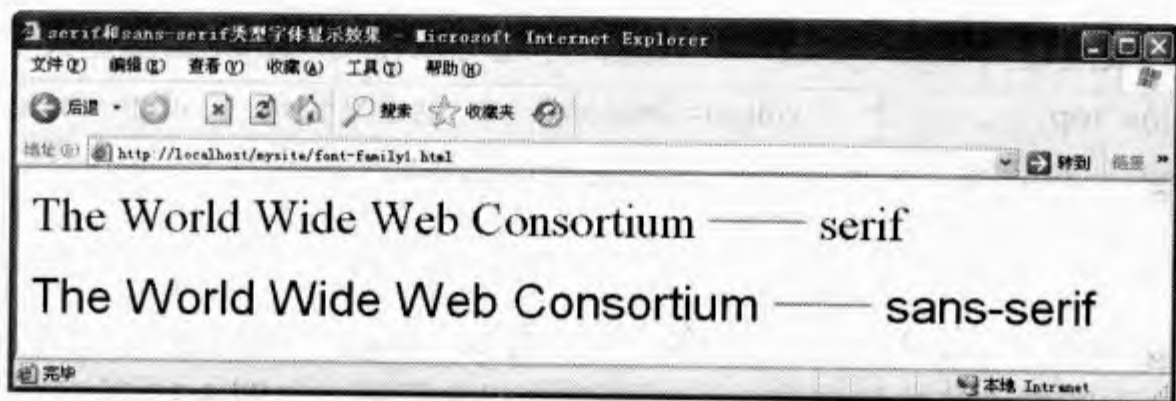



图 3.27 serif 和 sans-serif 类型字体显示效果

serif 类型字体很适合在正文中使用，这些“手”和“脚”装饰线很容易把一行内所有字母联系在一起，使浏览者的视线随着这些细线不断从左向右的牵引，这样能够提高阅读速度。

而 sans-serif 类型字体由于没有这些装饰线，整个字体显得干净、利落，很适合作为标题文字使用。

除了上面两类常用的通用字体外，在罗马字符中还可以看到下面几种类型的字体，当然这些字体在网页设计中不是很常用，多用于传统印刷中。

- cursive: 模拟人笔迹的字体，一般这些字体大部分是由曲线和比衬线字体更强的笔画修饰组成。例如，Comic Sans。
- fantasy: 特殊字体，这些字体不能通过某种单一特性来定义，而且也不能简单的归为其他系列中的某一类。例如，Western、Klingon。
- monospace: 无比率的字体，通常用于模拟打字机打出的字体，这些字体中每个字符都占有同样的宽度。例如，Courier。

当在网页中定义字体样式时，一般建议字体列表的最后一种字体最好选择通用字体。这主要是考虑到当浏览者的系统中没有指定字体时，浏览器可以搜索系统中同类字体来显示网页字体。所以在 Dreamweaver CS3 中定义字体样式时，你总会看到如图 3.28 所示的提示。这样你能够快速的设置网页字体列表，以及通用字体。

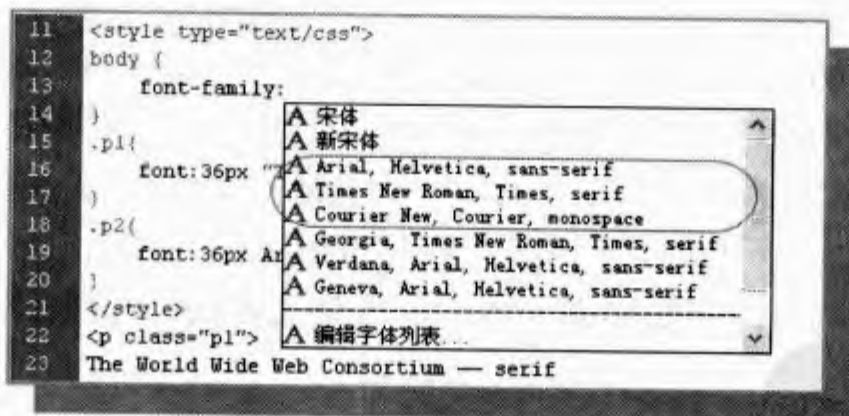


图 3.28 在 Dreamweaver CS3 中自动设置字体样式

3.4 小家伙，你向哪儿看齐

文字是很听话的“小家伙”，如果你的技术娴熟，再加上心细，你会发现它跟着你转，按你的意图行事。

在传统的 Table（表格）布局中，对齐应该说是最简单的操作了。例如，在单元格中，你可以使用 align 属性定义单元格中任何对象向左（align="left"）、向右（align="right"）、居中

(align="center") 或两端 (align="justify") 对齐, 你还可以使用 valign 属性定义单元格中任何对象向上 (valign="top")、向下 (valign="bottom")、垂直居中 (valign="middle") 或基线 (valign="baseline") 对齐等。下面我们就来看看具体的操作示例。

3.4.1 文本对齐

在 CSS 中要对齐文本或行内对象, 你可以使用 text-align 属性来实现。例如, 在下面代码中, 不管是文本、图像或者其他任何行内对象都被居中显示 (如图 3.29 所示)。

```
<style type="text/css">
div {
    text-align:center;
}
</style>
<div><br /><br />居中显示</div>
```



图 3.29 居中显示

text-align 属性也包括四个属性值: left、right、center 和 justify。这些属性值所表示的意思不再重复了。

3.4.2 布局居中

CSS 没有直接使用 align 作为文本对齐的属性, 大概也是在强调 text-align 属性仅能够作用于文本。请注意其前缀 “text” 似乎在提醒你只能够使用在行内文本中。因此, text-align 属性对于布局对齐问题也就无能为力。所以, 有时候你会感觉很奇怪为什么我定义网页居中对齐, 但是总不能实现呢? 例如, 下面的示例代码, 在标准浏览器中是无法居中显示的 (如图 3.30 所示)。不过在 div 元素内的文本倒是居中显示了, 这是文本属性都拥有继承特性。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
body {
    text-align:center;
}
div {
    border:solid 1px red;
    width:60%;
}
</style>
</head><body>
<div><br /><br />居中显示</div>
</body></html>
```


当然,如果仅就 IE 浏览器来说,这种担忧是不必要的,因为不管是布局还是文本,IE 系统能够把它们对齐显示(如图 3.31 所示),也就是说不管是行内文本,还是布局的模块。

要解决不同浏览器都能够居中显示问题,可以通过为布局元素定义 `margin` 属性实现,即定义其左右边距都为自动,则标准浏览器都会自动把布局元素置于居中的位置,代码如下:

```
body {
    text-align:center;
}
div {
    margin-left:auto;
    margin-right:auto;
    border:solid 1px red;
    width:60%;
}
```



图 3.30 无效的居中效果



图 3.31 在 IE 下居中显示效果

3.4.3 布局元素向左、右对齐

如果希望布局元素向左或右对齐,就不能够使用 `text-align` 属性来实现了。你可以有两种方法进行选择。

第一种方法:定义元素浮动显示。例如,在上面示例基础上增加如下代码,则显示效果如图 3.32 所示。

```
div {
    float:right;
    border:solid 1px red;
    width:60%;
}
```

`float` 属性表示浮动的意思,取值主要包括 `left` 和 `right`, 设置为 `none` 则可以定义元素不浮动。对于向左对齐,如果没有特殊布局需要,其实你也可以不用设置 `float:left;` 属性。

第二种方法:通过绝对定位来实现。例如,在上面示例基础上,删除 `float:right;` 声明,增加如下代码,则显示效果与图 3.32 所示一样。其中 `position:absolute;` 声明表示绝对定位的意思,所谓绝对定位就是元素能够精确的在网页中确定自己的位置,通常不受周围元素的影响。`right`

属性用来定义绝对定位元素的坐标值，可以与 left、top 和 bottom 属性配合使用。right:0; 声明表示元素向右看齐。

```
div {
    position: absolute;
    right: 0;
    border: solid 1px red;
    width: 60%;
}
```

当然，也有读者又有疑问了，为什么使用 text-align:right; 声明时可以使 div 元素向右对齐呢（如图 3.33 所示）？这是 IE 浏览器自己的“土”规矩，不符合 W3C 标准，因此没有得到其他标准浏览器的支持，也建议你不要采用这种方式进行对齐布局。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style type="text/css">
body {
    text-align: right;
}
div {
    border: solid 1px red;
    width: 60%;
}
</style>
</head><body>
<div> </div>
</body></html>
```



图 3.32 绝对定位右对齐



图 3.33 IE 下的右对齐

3.4.4 文本和元素垂直对齐

CSS 提供了一个标准的垂直对齐属性，即 vertical-align。该属性提供了强大的功能，实现更多的垂直对齐效果，共包括 8 种对齐样式（详细说明可以参阅第 4.3.2 节内容）。

但遗憾的是许多浏览器似乎对其并不热心。例如，输入下面代码，你会发现在 IE 或 Firefox 等不同类型的浏览器中所显示的效果都没有对齐底部（如图 3.34 所示）。

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



```
vertical-align:bottom;
width:12em;
height:6em;
border:solid 1px red;
}
</style>
<div>文本垂直对齐</div>
```

原来 `vertical-align` 仅能够作用于单元格或图像显示而定义的一个属性。因此如果要在上面样式内增加 `display:table-cell;` 声明,则在 Firefox 等标准浏览器中能够正确显示(如图 3.35 所示),而 IE 浏览器还不支持这样的定义。

```
div {
vertical-align:bottom;
display:table-cell;
width:12em;
height:6em;
border:solid 1px red;
}
```



图 3.34 IE 下无效的垂直对齐底部



图 3.35 在 Firefox 下垂直对齐底部显示

如果在表格单元格标签内定义 `vertical-align` 属性,则不同类型浏览器都能够很好的支持。例如,对于下面的垂直对齐样式,IE 和 Firefox 浏览器解析效果是相同的。

```
<style type="text/css">
.cell {
vertical-align:bottom;
height:60px;
}
</style>
<table width="200" border="1">
<tr>
<td class="cell">文本垂直对齐</td>
</tr>
</table>
```

但是在其他元素内,IE 就不能够很好的支持 `vertical-align` 属性了,即使声明了 `display:table-cell;` 也是如此。这使得该属性的普及率大打折扣,为此设计师只能另辟途径,当然也涌现出很多间接的、不成熟的垂直对齐技术或技巧。下面介绍单行文本垂直居中对齐设计技巧。有关 `vertical-align` 属性的更详细讲解可以参阅第 4.3.2 节内容。

单行文本垂直居中对齐是经常需要解决的问题,你可以使用下面方法巧妙地解决:

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

```
div {
    line-height:6em;
    width:12em;
    height:6em;
    border:solid 1px red;
}
</style>
<div>文本垂直居中对齐</div>
```

通过定义单行文本的高度和行高相同，这样就能够间接的实现文本垂直居中显示问题（如图 3.36 所示）。当然对于多行文本来说，这种方法就失效了。还好我们一般希望垂直居中对齐的文本多是单行。

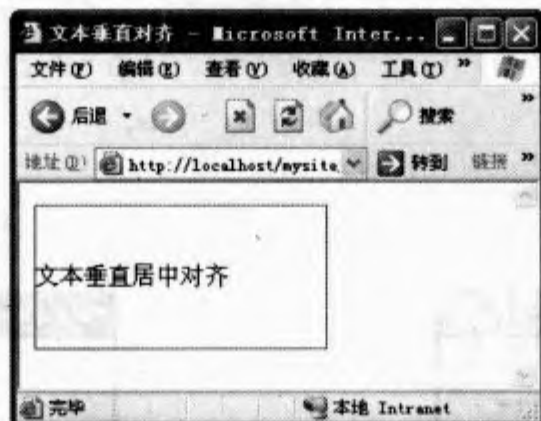


图 3.36 单行文本垂直居中显示

3.4.5 绝对定位元素居中显示

如果在 Dreamweaver CS3 中选择【插入记录】|【布局对象】|【AP Div】菜单命令，则 Dreamweaver 会自动在编辑窗口的光标所在位置插入一个绝对定位元素（如图 3.37 所示）：

```
<style type="text/css">
<!--
#apDiv1 {
    position:absolute;
    width:200px;
    height:115px;
    z-index:1;
}
-->
</style>
<div id="apDiv1"></div>
```

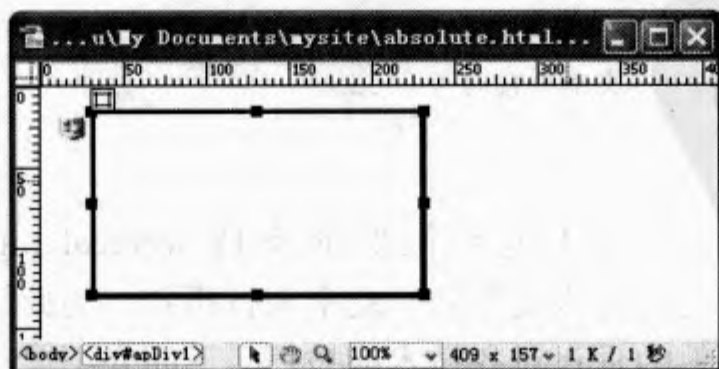


图 3.37 插入的绝对定位元素

其实任何元素只要声明 `position:absolute;` 都可以定义为绝对定位元素。绝对定位元素可以通过 `width` 和 `height` 属性定义元素的宽和高，使用 `left`、`top`、`right` 和 `bottom` 属性定义绝对定位元

素在包含块中的位置，使用 `z-index` 属性定义相互层叠的顺序。有关这些属性的详细使用可以参考本书附赠的 CSS 参考手册。

顾名思义，绝对定位就是能够精确定位元素位置的布局方法。对于绝对定位元素来说，你可以精确的控制它的位置，但是如何使它灵活适应随时变化的浏览器窗口，以及流动的网页内容，确实是很多初学者普遍关心的问题。

例如，我们希望图 3.37 中的绝对定位元素能够总是水平居中显示，怎么办呢？

由于 CSS 仅是一种类似标签的语言，它没有逻辑处理能力，所以你就无法在 CSS 样式中增加计算功能，以便随时计算和调整元素的位置。

荀子说“假舟楫者，非能水也，而绝江河。”这里我们不妨“善假于物”，巧妙借助一个外套来解决绝对定位元素居中显示问题。由于当元素定义为相对定位显示（即声明 `position:relative;`）时，它是能够随着文本流进行移动的，也就是说你可以使用上面的方法使相对定位元素居中显示。另外，由于相对定位元素具有包含块的功能，所以绝对定位元素就会根据最近的包含块来定位自己的位置。

提示

什么是包含块呢？所谓包含块就是元素被定义了 `position` 属性，以绝对、相对或固定位置显示时，就可以算作一个包含块，具有包含块的元素，它能够为其包含的绝对定位元素提供坐标参考，也就是说绝对定位元素就不再根据 `body` 元素的左上角来确定自己的位置，而是根据包含块左上角位置定位自己的位置。

因此，根据上面的设计思路，我们可以为绝对定位元素包裹一个相对定位的元素，然后再设计包含元素居中对齐即可，显示效果如图 3.38 所示，你可以看到外部的 `div` 元素虽然收缩为一条蓝色线，与包含的绝对定位元素的上边框重合，但是它能够很准确的控制绝对元素居中对齐问题。请注意，为了准确居中对齐，你应该设置外部的相对定位包含块宽度与内部的绝对定位包含块宽度相同。详细代码如下。

```
<style type="text/css">
<!--
#apDiv1 {
    position:absolute;           /* 绝对定位 */
    width:200px;                 /* 定义宽 */
    height:115px;                /* 定义高 */
    z-index:1;                   /* 定义层叠顺序编号，号越大越排在上面 */
    border:solid 1px red;        /* 定义一个边框，以方便观察 */
}
body {
    text-align:center;           /* 兼容在 IE 浏览器中对齐元素 */
}
.wrap {
    position:relative;           /* 相对定位 */
    margin:0 auto;                /* 在标准浏览器中居中对齐 */
    width:200px;                 /* 定义宽 */
    border:solid 1px blue;        /* 定义边框 */
    text-align:left; /* 定义包含文本向左对齐，兼容 IE 浏览器下绝对元素错误显示 */
}
-->
</style>
<div class="wrap">
    <div id="apDiv1">绝对定位元素居中显示</div>
</div>
```



图 3.38 绝对元素居中对齐

3.4.6 混合结构中对齐处理

CSS 技术本身并不是很难，难的是如何在复杂的网页环境中实现在不同浏览器间呈现相同的显示效果，也就是所谓的浏览器兼容性技术。当网页嵌套层次比较深，所设置的样式相互牵连在一起时很容易出现各种各样的问题，甚至还会造成页面结构的混乱。

例如，在 <http://www.xinhuanet.com/> 首页的右侧一个“新华资料”栏目中，所有导航菜单理应靠左对齐，但是由于设计师的疏忽，结果在 IE 6 浏览器中显示为居中对齐（如图 3.39 所示）。



图 3.39 复杂环境下的对齐问题

由于对齐属性具有继承性，也就是说如果你在 `body` 元素中声明居中对齐 (`text-align:center;`)，则网页内所有文本都会居中对齐。为了避免类似问题，你必须在内部声明向左对齐，否则就会出现类似于图 3.39 所示的低级错误。

如果在 `body` 元素中声明了居中对齐，那么是不是在每个元素内都应定义左对齐呢？答案是否定的，我们可以在总包含元素中声明一次即可，其内部的元素就会继承最近一级上级元素的对齐属性。例如，对于下面这个框架结构：

```
<div id="wrap">
  <h2>标题文本</h2>
  <div id="main"></div>
  <div id="footer"></div>
</div>
```


如果希望网页居中显示,则可以定义如下样式。

```
body {
    text-align:center;          /* 定义网页在 IE 下对齐 */
}
#wrap {
    margin:0 auto;              /* 定义网页在标准浏览器中对齐 */
}
```

虽然上面方法实现了网页在不同类型浏览器中的对齐效果,但是文本也跟着居中对齐了,为了防止此问题的发生,你可以在#wrap选择器中补加一条规则。

```
#wrap {
    margin:0 auto;
    text-align:left;
}
```

这样,所有问题都解决了。如果希望网页内某个元素内文本居中对齐,则只需要单独定义一个样式即可。例如,我们再补加一个样式声明标题文本居中对齐。

```
#wrap h2 {
    text-align:center;
}
```

3.5 咱们比比个吧

行高是网页文本的灵魂,也许习惯于自然,或遵循默认设置,你没有注意到它的存在,但是如果去拜访中央电视台首页(<http://www.cctv.com/default.shtml>),也许你的眼睛并不是那么舒服了,如图3.40所示。如果不仔细用鼠标去试探,你很难分辨出图3.40中间的插图新闻是一个新闻列表,或许还以为它们是一段补白呢,文本行过于亲密严重影响了浏览者的阅读体验。



图 3.40 中央电视台首页截图

但是如果再浏览人民网首页(<http://www.people.com.cn/>),你绝对会心情舒畅得多,如图3.41所示。同样都是新闻列表,但是所定义的行高不同给浏览者带来的阅读体验也截然不同。

定义文本的行高可以使用 line-height 属性,该属性的值可以选择 px、em、百分比等作为单位。其中 em 和百分比单位的取值大小与行内字体的大小存在关系。例如,我们为段落文本定

义 20 像素高的行高，则样式代码如下：

```
p{
    line-height:20px
}
```

一般来说，在定义行高时使用 **em** 和百分比作为单位会更好，不建议读者使用 **px** 作为单位。原因很简单，就是行高能够随文本字体大小可以随时进行调整，而使用 **px** 作为单位，如果要调整字体大小，还需要手动调整行高的值。



图 3.41 人民网首页截图

3.5.1 什么样的行高更合适阅读

浏览器默认行高是 120%，也就是行内字体大小的 1.2 倍，如果使用 **em** 来表示就是 1.2em。默认行高显示效果如图 3.42 所示。对于段落文本来说，默认行高稍显紧密，比较合适的行高为 160%~180%，如图 3.43 所示设置为 160% 的行高。不过超过 200% 又稍显疏离，不利于视力的集中。

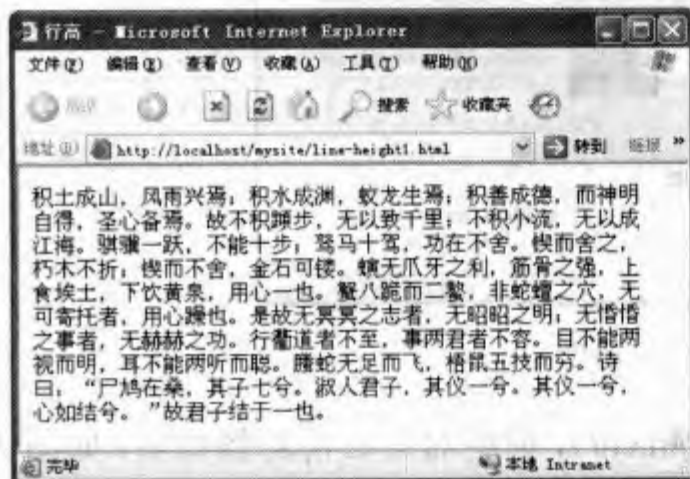


图 3.42 默认行高

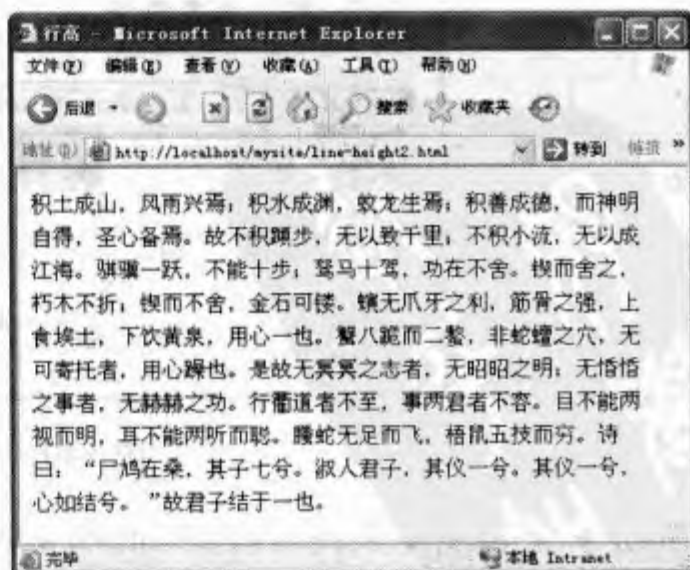


图 3.43 160% 的行高

而当行高小于 100% 时就会发生重叠现象。这个也可以理解，行高减去字体大小所得数值

是行上下边距的和,如果除以2就可以得到文本行一边的间距。例如,假设行内字体大小为12像素,行高为1.5em,则行高为18像素,如果行高减去字体大小,然后再除以2,就可以得到3像素的行间距。

可能你有疑问了:上一行的边距为3像素,下一行的边距也为3像素,加起来应该是6像素,也就是说行间距应该为6像素?

从理论上计算是正确的,但是CSS规定,当段落文本行中,上下行之间可以发生重叠,所以也就只有3像素的间距。

实际上,CSS也允许我们直接使用数字来表示行高,例如,在下面的样式中,虽然没有指定单位,但是CSS能够理解它为1.6em。

```
p {
  line-height:1.6;
}
```

请注意, line-height 具有继承性,当你在 body 中定义 line-height 为 1.6em,则所有文本行的行高都为 1.6em,为了避免此类问题的发生,建议你在需要的段落文本逐一定义,或者在 body 中定义一个默认的行高,然后在其他特殊需要的文本段再进行修改。

一般情况下,文字显示越大,行距应该就越小,这样不至于在阅读是感觉疏离的感觉。对于正文字体大小为12像素来说,160%~180%的行高是比较合适的。定义的字体越大,建议适当调低行高。当然,文字越小,大的行距反而有利于阅读。具体设置还需要读者自己的根据需要进行把握。

3.5.2 行高、边距和伪行高

行高不等于边距,在CSS中定义行高使用 line-height 属性来实现,而定义边距通过 margin 和 padding 属性来实现。

但是初学者在具体设计时把它们混用在一起。例如,为了增加导航菜单与上下元素的距离,而把行高定义一个很大的值,期望用行高来代替边距的定义,以实现与上下元素的间距控制。这种方法虽然有时很管用,但是不提倡。因为不同浏览器对于行高的解析规则是不同的。例如,针对下面的结构,我们希望调整导航菜单与主体内容之间的间距:

```
<ul>
  <li>菜单 1</li>
  <li>菜单 2</li>
  <li>菜单 3</li>
</ul>
<div id="main"> 主体内容 </div>
```

使用行高来实现拉大间距,样式代码如下:

```
<style type="text/css">
ul {
  list-style:none;      /* 清除列表中的项目符号 */
  margin:0;            /* 清除列表缩进 */
  padding:0;           /* 清除列表缩进 */
}
li {
  float:left;          /* 列表项向左浮动 */
  width:100px;         /* 宽度 */
  height:30px;         /* 高度 */
}
```

```

text-align:center;          /* 居中 */
line-height:42px;          /* 行高 */
}
#main {
float:left;                /* 浮动显示 */
border:solid 2px red;      /* 增加边框 */
height:100px;              /* 高度 */
width:100%;                /* 宽度 */
}
</style>

```

通过这种方式在 IE 浏览器中可以正确的显示(如图 3.44 所示),而在 Firefox 中无法正常显示(如图 3.45 所示)。也就是说行高只对文本起作用,不具有拉开与其他元素距离的功能,但是 IE 能够把行高与边距问题联系在一起,所以给很多初学者发出一个错误的信号。对此使用 margin 或 padding 属性来设计会更安全、更方便。



图 3.44 IE 下行高显示效果

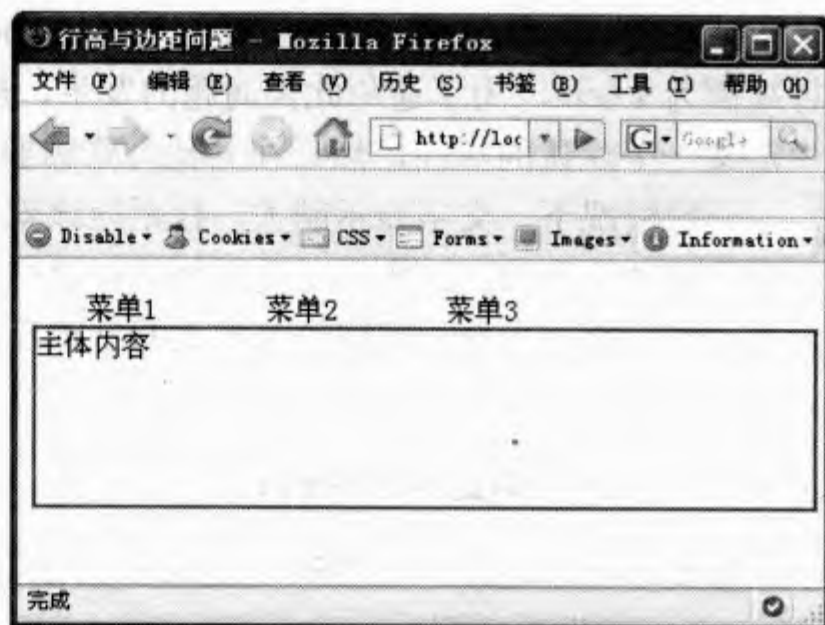


图 3.45 Firefox 下行高显示效果

当我们在文本行内插入图像、Flash 动画、行内块元素时,你会发现行高突然变大了(如图 3.46 所示)。其实文本行的高度依然保持原来的高度,仅是一种错觉,因此称为伪行高。不过利用这种伪行高,你也可以利用透明图像块来设计文本行的高度,这在传统表格布局中经常被使用,因为它比较安全、有效。但是在 CSS 布局中,实现这种效果方法有很多,且符合标准,一般设计师都不再使用了。

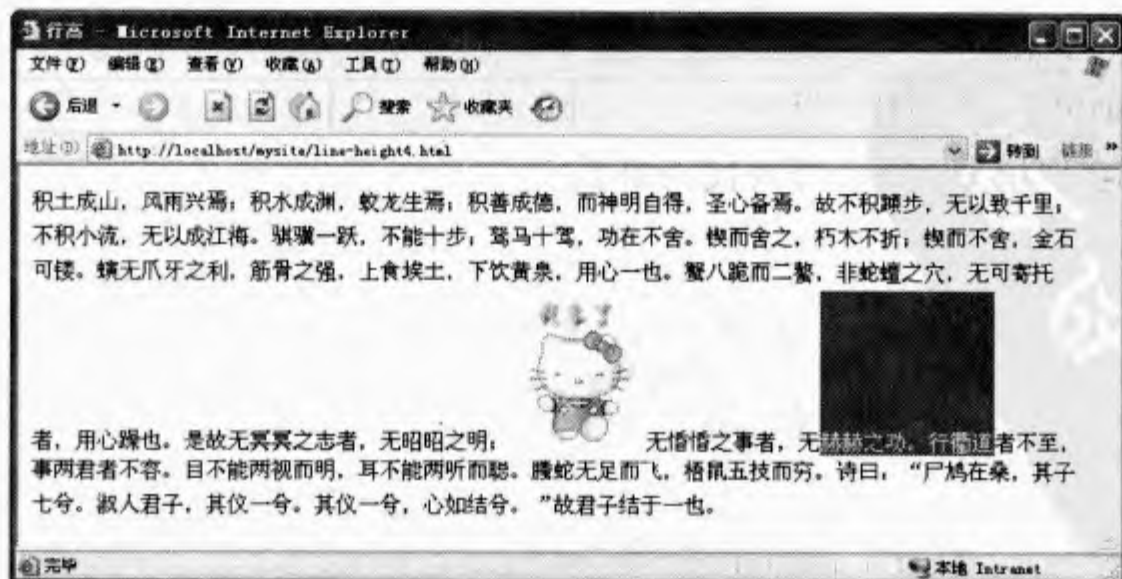


图 3.46 伪行高效果

3.6 让网页看起来和颜悦色

习惯了黑白世界的用户，如果尝试把网页字体设计为天蓝色或者紫玫瑰色等，你会突然发现页面变得更加亮丽。使用 CSS 为文字定义颜色很容易，即使用 `color` 属性即可，然后为其设置一个颜色值即可。

例如，下面 5 种方法都能够设计二级标题显示为红色。其中第 1 个样式使用十六进制数字来表示，第 2 个样式使用简写的十六进制方法来表示，第 3 个样式使用颜色名来定义，第 4 个样式使用 RGB 方法来实现，第 5 个样式使用 RGB 方法并使用百分比参数来实现。

```
h2 { color:#FF0000; }  
h2 { color:#f00; }  
h2 { color:red; }  
h2 { color:rgb(255,0,0); }  
h2 { color:rgb(255%,0%,0%); }
```

3.6.1 在 CSS 中使用颜色

使用 CSS 定义字体颜色的方法有多种，上面代码显示了所有定义方法，下面我们再简单讲解这些方法的用法。

(1) 预定义颜色表示法。

预定义颜色就是使用颜色的英文名称来表示，浏览器根据预定义名称来解析并显示对应的颜色。例如，在 `color:red`、`color:green`、`color:blue` 声明中，`red`、`green` 和 `blue` 都是 CSS 关键词，这些预定义的关键词分别表示红色、绿色和蓝色。类似的颜色名还有很多，你可以查阅 CSS 参考手册。

(2) RGB 颜色表示法。

RGB 颜色表示法就是红 (R)、绿 (G) 和蓝 (B) 三原色混合后呈现出的颜色，其中每种原色的取值为 0~255，并根据取值的比例进行混合。例如，红色、绿色和蓝色可以分别使用下面代码来表示：

```
color:rgb(255,0,0);  
color:rgb(0,255,0);  
color:rgb(0,0,255);
```

(3) RGB 百分比颜色表示法。

RGB 百分比颜色表示法就是利用百分比来表示 RGB 颜色，其中 RGB 中的 0 等价于百分比中的 0%，RGB 中的 255 等价于百分比中的 100%。例如，红色、绿色和蓝色可以分别使用下面代码来表示：

```
color:rgb(100%, 0%, 0%);  
color:rgb(0%, 100%, 0%);  
color:rgb(0%, 0%, 100%);
```

(4) 十六进制颜色表示法。

十六进制颜色表示法就是使用三对十六进制数分别表示 RGB 中的三原色。例如，在 `color:#112233` 声明中，其中 11 就表示 R 的颜色，22 就表示 G 的颜色，而 33 就表示 B 的颜色，为了能够与数值相区分，需要在颜色值前面再加一个 # 号前缀。

如果在十六进制颜色表示法中，R、G 和 B 的颜色值前后一致时，可以简写为 1 个。例如，对于上面的 `color:#112233` 声明，可以简写为 `color:#123`。请注意，在十六进制颜色表示法中，

如果前后值相同的颜色值，都属于网络安全色，例如，#ff8800、#ee66dd、#0099ff 等。

另外，在 CSS 3 版本中开始支持另外 3 种颜色表示法：

● RGBA 颜色表示法

RGBA 颜色表示法就是在 RGB 颜色的基础上增加了 Alpha 通道，这样就可以定义半透明的颜色。例如，color:rgba(255,0,0,5);声明就可以定义半透明的红色。但是由于大部分浏览器暂时不支持 CSS 3 版本技术标准，因此不建议读者去尝试。

● HSL 颜色表示法

HSL 颜色表示法就是使用色相 (H)、饱和度 (S) 和亮度 (L) 表示颜色的一种方法。例如，color:hsl(0,100%,100%);就表示红色。

● HSLA 颜色表示法

HSLA 颜色表示法就是在 HSL 颜色的基础上增加了 Alpha 通道。例如，color:hsla(0,100%,100%,5);就表示半透明的红色。

3.6.2 CSS 布局与配色

如果说网页配色是技术问题，倒不如说是艺术设计问题。实现网页配色的过程可能很简单，但是要设计一个和谐、亮丽的网页色调确实不容易，它更多的需要设计师的创意和灵感。

一般来说要完成网页配色，需要网页背景色和前景色配合才能够完成。有时候还需要背景图像以实现更个性的设计，有关背景图像的相关 CSS 技术，我们将在下一章中详细讲解。

定义背景色可以使用 background 属性来实现，定义前景色也就是字体颜色，即使用 font 属性来实现，对于其他对象来说（如图像、动画）是无法在网页中定义前景色的。

网页配色是件很复杂的工作，没有详细的技术标准，但是我们一般遵循颜色主题越集中越好。例如，对于一个以儿童为主题的网站，色调趋于嫩绿、鹅黄或卡通色等，而对于女性为主题的网站，色调多趋于粉色。以游戏为主题的网站，你会看到的多是黑酷色调等。

下面以一个实例来讲解如何对网页进行配色，在学习和配色过程中，建议读者应准备一个识色器和一个颜色参考表。

首先，建立一个固定宽度的 2 行 2 列的结构页面，当然你也可以构件其他复杂的结构：

```
<div id="wrap">
  <h3 id="header">网页标题</h3>
  <ul id="nav">
    <li>链接 1</li>
    <li>链接 2</li>
    <li>链接 3</li>
    <li>.....</li>
  </ul>
  <div id="main">
    <div>正文内容.....</div>
  </div>
</div>
```

然后，使用 CSS 支撑起这个框架（效果如图 3.47 所示），具体方法这里就不再讲解，我们将在后面章节中详细讲解不同布局方法。

```
<style type="text/css">
body {
  text-align:center;          /* 网页居中 */
}
```



```

#wrap {
    width:400px;           /* 固定包含框的宽度 */
    margin:0 auto;         /* 网页居中 */
    text-align:left;       /* 文本左对齐 */
}
#header {
    height:40px;           /* 固定高度 */
    line-height:40px;      /* 定义行高 */
    margin:0 0 2px 0;      /* 头部区域的外边距 */
    text-align:center;     /* 文本居中对齐 */
}
ul#nav {
    list-style:none;       /* 清楚项目符号 */
    margin:2px 0 0 0;      /* 导航栏外边距 */
    padding:10px 0 0 10px; /* 导航栏内边距 */
    float:left;            /* 向左浮动 */
    width:84px;            /* 固定宽度 */
    height:190px;          /* 固定高度 */
}
#wrap #main {
    float:right;           /* 向右浮动 */
    height:200px;          /* 固定高度 */
    width:300px;           /* 固定宽度 */
    margin:2px 0 0 2px;    /* 增加外边距 */
}
ul#nav li {
    line-height:1.5em;     /* 导航行高 */
}
#main div {
    padding:12px 2em;      /* 主体区域内边距 */
}
</style>

```



图 3.47 网页配色前效果

最后，分别为网页背景色，以及头部区域、导航侧栏和主体区域的前景色和背景色进行搭配。网页背景色采用天蓝色（淡色调）进行设置。头部区域可以采用草绿色背景进行搭配，配上红色字体，可以强化头部区域的内容。左侧栏目采用鹅黄色背景，这样可以使整个栏目更加亮丽。右侧主体区域采用粉红色背景，这样更适宜用户进行阅读。整个页面的配色效果如图 3.48 所示。

```
<style type="text/css">
body {
    color:#FF0000;          /* 网页字体基本色, 一般多为黑色或深灰色 */
    background:#99FFFF;     /* 网页背景色 */
}
#header {
    color:#FF0000;          /* 标题栏字体色 */
    background:#66CC66;     /* 标题栏背景色 */
}
ul#nav {
    color:#000;             /* 导航侧栏字体色 */
    background:#CCFF33;     /* 导航侧栏背景色 */
}
#wrap #main {
    color:#000;             /* 主体区域字体色 */
    background:#FF99CC;     /* 主体区域背景色 */
}
</style>
```

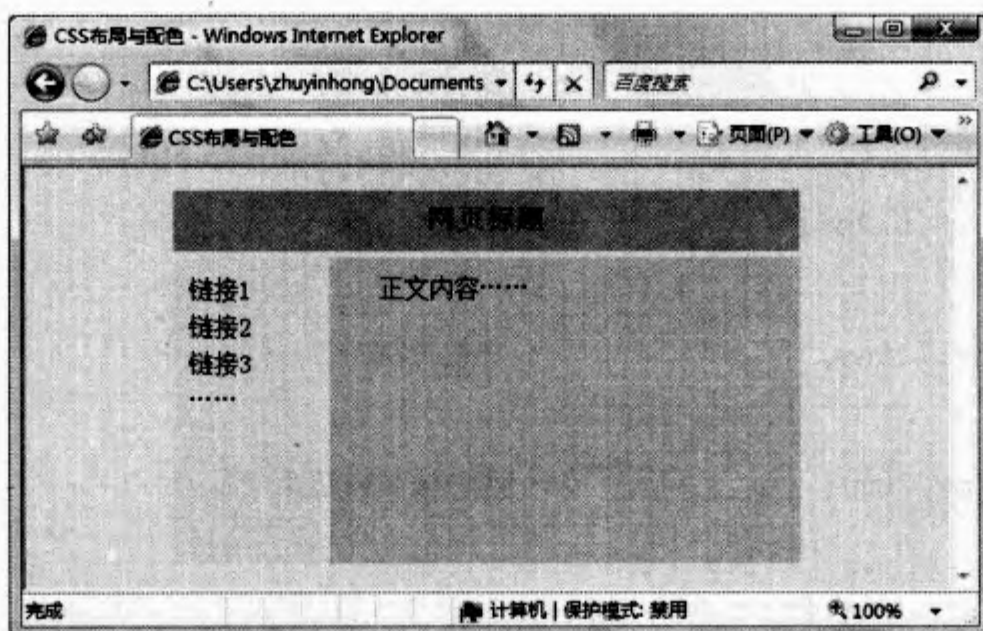


图 3.48 网页配色后效果

扮靓网页的天使——图像

人的想象力是无限的，所以设计的想象力也将会更加开阔。我曾傻傻的臆想：如果网页没有了图像，设计师的灵感会不会枯竭？回到现实，随意打开一个网页，你都会发现图像在其中扮演着重要的角色。一个渐变的背景，一个轻巧的边饰，或者一个别致的图标，都能让你的网页魅力无穷，充满着人性的想象力。

CSS 提供了比传统网页布局更加强大的图像控制能力，使用 `img` 元素可以插入图像对象，使用 `background` 属性可以控制背景图像在网页中呈现效果等。本章与读者一起分享图像带给我们的快乐和灵感。

4.1 选择：前景图像和背景图像

也许当打开网页准备完成今天的作业时，你可能就要思考这样的问题：我的 Logo 是该使用 `` 标签插入还是使用 `background` 属性声明呢？

下面我们就从这个疑问开始本章的学习。

4.1.1 前景图像

前景图像是相对于背景图像来说的，实际上就是我们常说的图像。在网页中插入图像一般使用 `img` 元素来实现。例如，使用下面代码可以快速在网页中插入 Logo 图标。

```

```

`` 标签中的 `src` 属性用来指定引用图像的 URL 路径。同时该标签还提供了很多属性用来控制图像的显示，不过习惯使用 CSS 之后，你会发现控制图像会更加容易了，在下面章节中我们将详细讲解。

除了 `img` 元素外，你还可以使用 `object` 元素插入图像。例如，使用下面代码一样可以在网页中插入 Logo 图标：

```
<object data="images/logo.gif" >
```

W3C 组织曾经希望使用 `object` 作为引用外部对象的标准元素，试图逐步废除 `img` 这种不便的用法，但是苦于 `img` 元素已经深入人心，同时 IE 浏览器 7.0 版本依然无法解析（如图 4.1 所示），因此这项计划没有很顺利的实现，不过这个提议得到了标准浏览器的大力支持，因此你可以在非 IE 浏览器下看到使用 `object` 元素插入图像的效果（如图 4.2 所示在 Firefox 中显示效果）。

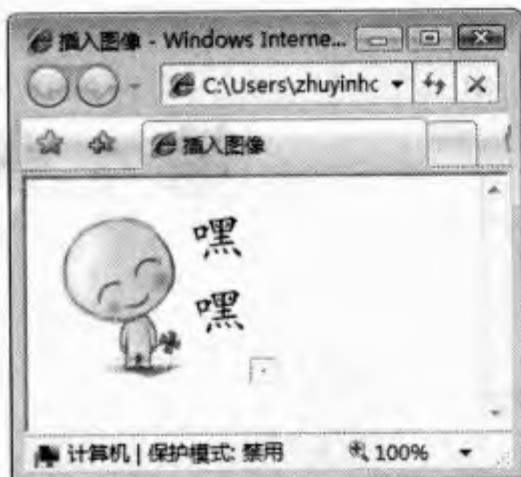


图 4.1 在 IE 7 下解析图像



图 4.2 在 Firefox 2 下解析图像

4.1.2 背景图像

在传统布局中，你可以在<body>、<table>或<td>标签中使用 background 属性来定义背景图像，但在其他标签中定义背景图像的局限性就很大，甚至根本就不支持这种功能。如今使用 CSS 布局，你可以放心的使用 background 属性为任何标签定义背景图像，这应该算是一个大的飞跃。

使用背景图像来设计网页基本上已成为设计师的必修课。可以这样说，背景图像是设计完美网页的基础。如果想学习和研究这个课题，建议你不妨先参考“禅意花园”(<http://www.csszengarden.com/>)。固定不变的 HTML 结构和网页内容是很简单的一个页面，但是在不同设计师手中会呈现不同的创意效果。而这其中正是背景图像的功劳。

如图 4.3 所示是禅意花园第 124 号作品。蚊格似的背景，淡茶色的色调，辅之一杯清茶特写，页面内随意摆放的羽叶似的图案，配着飘逸的字体，你绝对认为这是一首好诗。

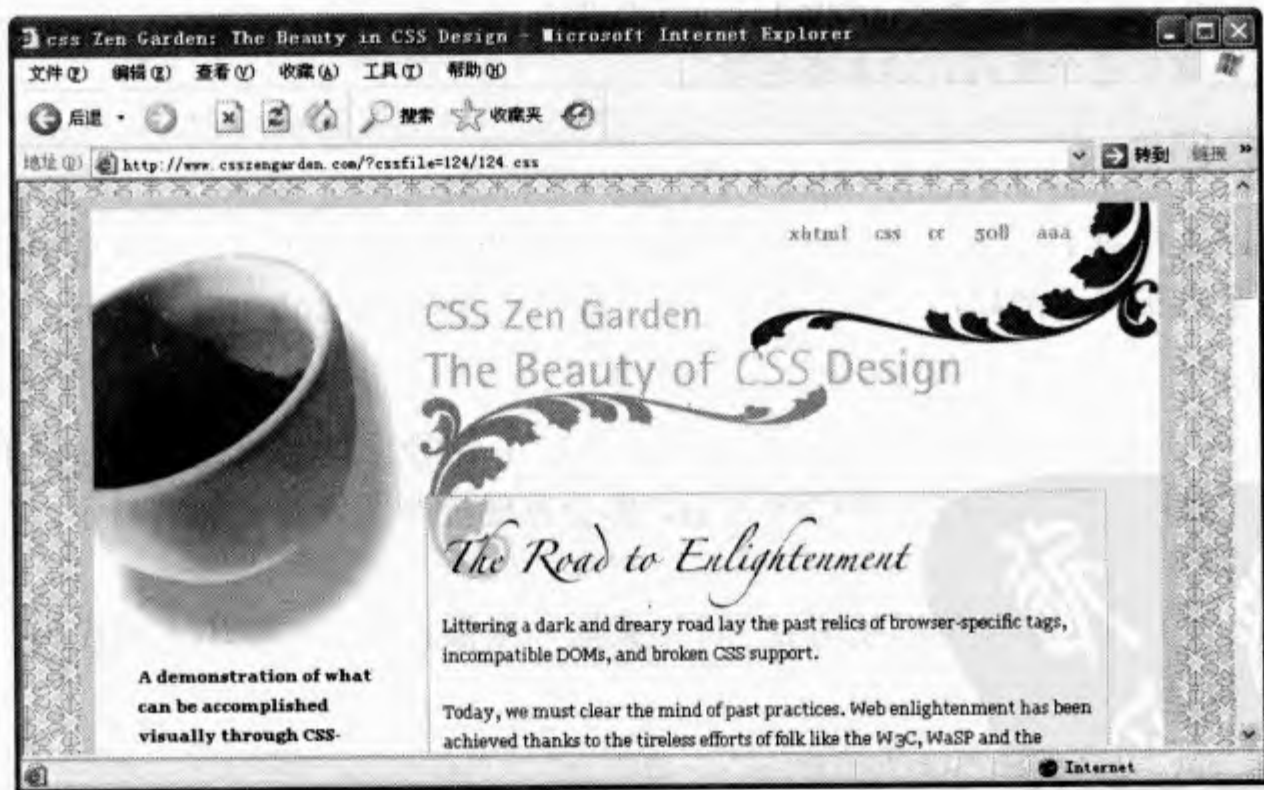


图 4.3 禅意花园第 124 号作品——清茶飘香

如图 4.4 所示是禅意花园第 209 号作品。摩萨的背景，以精致润泽的手感烘托高级会所的高雅，主人公闲适的交流描绘了白领人生，底部深灰色的背景，以及左侧黄色的导航写出了办公气氛的严肃和安静。



图 4.4 禅意花园第 209 号作品——脑力会所

也许你在欣赏这些作品时灵感一现或者活化迸溅，其实你完全可以自己动手，在禅意花园中崭露头角。

定义背景比较简单，你只需要为指定选择符设置 background 属性即可。例如，下面的样式代码可以为网页增加一个背景：

```
body {
    background-image: url(images/logo.gif);
}
```

4.1.3 如何选用前景图像和背景图像

在标准网页设计中，前景图像是作为网页内容而存在的，所谓网页内容就是提供给浏览器阅读并获取信息的对象。例如，你在网上发布个人照片，这些照片被发布的目的是希望别人欣赏，因此使用标签插入是比较恰当的。

而背景图像是作为网页修饰的工具而存在的，对于网页背景来说，它的目的是为了网页设计得更漂亮，不具备传递信息的作用，因此不建议使用标签插入，而 background 属性以背景的形式显示。

从 HTML 语义角度分析，网页中任何一个标签都表示一个意思，对于标签来说，它就表示图像，也许作为人来说，你可以分辨出哪些图像是标签内容，哪些图像是装饰性背景图像。而对于计算机来说，它只能够根据标签来进行判断。如果把所有背景图像都使用 background 属性来定义，这样就能够使设计的网页语义结构更合理。

由于 background 具有更强大的控制功能，因此使用 CSS 可以设计出更具创意的页面效果，而使用标签插入图像就没有 background 属性那样容易控制。

4.1.4 走出迷失的图像 URL

不管是使用图像还是使用背景图像，你都需要设置好图像 URL 问题。使用标签中插入图像需要使用 src 属性，而 background 属性定义背景图像需要使用指定 url。

Uniform Resource Locator 的简写是 URL，中文翻译为统一资源定位符，通俗说就是文件路径。文件路径有物理路径和网络路径之分，物理路径就是本地路径，在本地计算机中使用，例

如, C:\Documents and Settings\zhu\My Documents\logo.gif。当然在网页中是不能够使用的。网页中一般使用网络路径,也就是我们常说的 URL。URL 可以包括绝对路径和相对路径,当然本地路径中也有相对和绝对之分。

所谓绝对路径就是指明文件在网络中的服务器地址、所用协议和站点地址等。例如,在下面这个绝对路径中,http 表示网络协议, www.mezzoblue.com 表示网站服务器的地址, zengarden/submit/表示站内文件的目录, index.html 表示具体的文件。

http://www.mezzoblue.com/zengarden/submit/index.html

为图像指定绝对路径有两大好处:

第一,设置很简单。你不会因为文件之间错综复杂的路径关系而迷失其中,甚至出现似乎已经设置了图像 URL,但是却找不到图像的问题。

第二,访问最安全。网页中的图像被设置了绝对路径之后,不论页面转移到什么位置,也不论网页文件被浏览者保存到本地。总之,在任何位置、任何计算机中访问网页都能够正确显示图像,当然前提是必须连接到互联网。

很多时候有经验的设计师都会在网站中开辟一个图像专用文件夹,把常用的装饰性图像都放入其中,引用时直接使用绝对路径,而不再考虑文件之间的位置关系。这样你也可以在其他网站或者公共服务区引用这些图像。

当然,在站点引用图像时,设计师还是喜欢采用相对路径。相对路径又可以分为:相对根目录的路径和相对文档的路径。如果你在 Dreamweaver CS3 中插入图像,会看到如图 4.5 所示的设置选项。

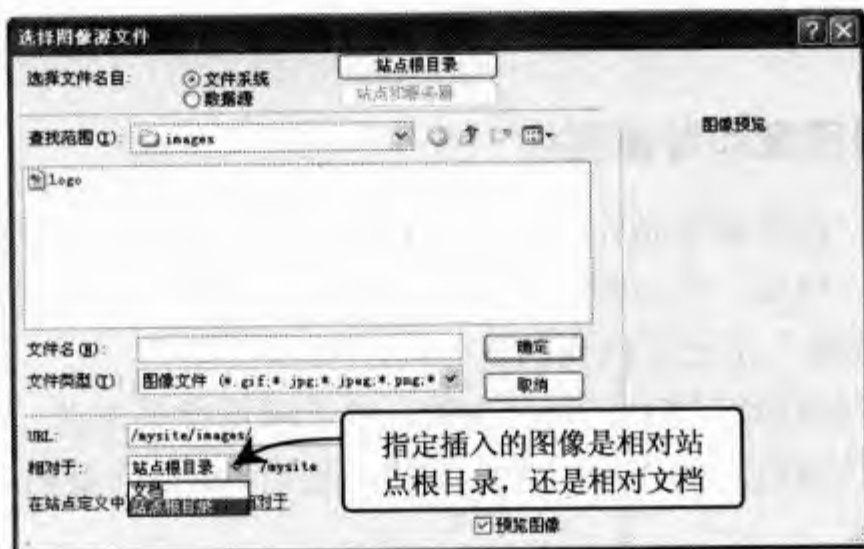


图 4.5 Dreamweaver CS3 中【插入图像】对话框

所谓相对站点根目录路径就是指文件相对于网站的一级文件夹(网站根文件夹)的位置,或者说就是绝对路径去掉 http 协议和站点服务器地址后的字符串。它以“/”开头,表示网站的根目录,例如,在下面代码中就是使用相对站点根目录来插入图像。

```

```

使用相对站点根目录插入图像的方法也不会迷路,因为站点根目录是固定的,只要你根据图像在站点中的位置准确定位即可。这样所定义的样式表文件不管移动到站点内哪个位置,它都能够准确找到指定的图像。

当然在网站开发初期,如果网页不在服务器上或者测试服务器上(包括本地或远程),那么你不能看到网页中指定的图像,所以对于仅仅在一个孤立的网页中插入图像时,是不支持使用这种方法的。

最容易让人迷糊的就是相对文档的路径设置了，特别是网页、样式表文件和图像文件位于不同的目录下，这种关系更是让初学者犯晕。所谓相对文档的路径，就是指从当前引用图像的文件到被引用的图像文件的路径。下面结合一个具体的案例来进行讲解。

假设网页文件（background1.html）位于站点根目录下，样式表文件（css.css）位于根目录下 styles 文件夹中，图像文件（bg1.jpg）位于根目录下 images 文件夹中，它们的关系如图 4.6 所示。



图 4.6 相对文档路径的关系

那么在网页中直接插入图像文件（bg1.jpg），使用相对文档路径可以表示：

```

```

而如果使用根目录下 styles 文件夹中样式表文件来引用，其中双点表示上一级目录则应该是：

```
body {
    background:url(../images/bg1.jpg);
}
```

也就是说相对文档的路径是根据样式表的位置来确定的，而不是根据所引用的网页文件位置来确定。当你在复杂网站结构中，使用 CSS 的同时，又使用 JavaScript 脚本引用图像文件，这种错误很容易发生，因为在 JavaScript 脚本中引用文件与 CSS 恰恰相反，它是根据网页文件的位置来确定 URL 的相对位置。

4.2 图像边框和阴影

在默认状态下，图像是不显示边框的，但是如果你为图像绑定了超链接，那么情况就不同了。当看见那又粗又醒目的蓝色边框时，相信你一定着急除去它。而当你需要给图像增加一个漂亮的边框时，却又不知如何下手。下面我们就来研究图像边框和阴影的设计技巧。

4.2.1 定义边距

为图像定义边框比较简单。在标签中可以使用 border 属性设置图像边框的粗细，默认为 0，即不显示边框。例如，使用 border 属性为边框增加 4 像素宽的边框后，显示效果如图 4.7 所示。

```

```

很显然上述方法是无法控制图像边框的颜色，更不用谈各种边框样式的设计了。在 CSS 中可以通过 `border` 属性为图像增加各种边框样式，并能够控制粗细和颜色。当然 CSS 中 `border` 属性不仅仅适用于图像，它还适用于所有对象元素。

例如，在下面的代码中分别为 3 幅图像定义了不同的边框样式（如图 4.8 所示）。

```
<style>
.img1 {
    border:dashed #666 1px;           /* 灰色虚线边框，1 像素宽 */
}
.img2 {
    border-bottom:double blue 3px;    /* 双线底边框，3 像素宽 */
}
.img3 {
    border:groove red 8px;            /* 红色立体边框，8 像素宽 */
}
</style>



```

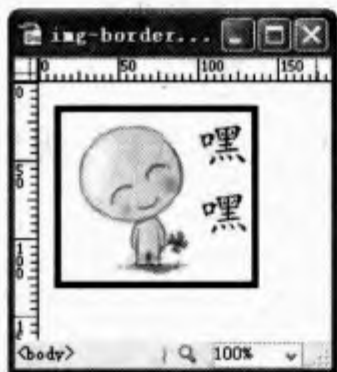


图 4.7 图像自身属性定义边框

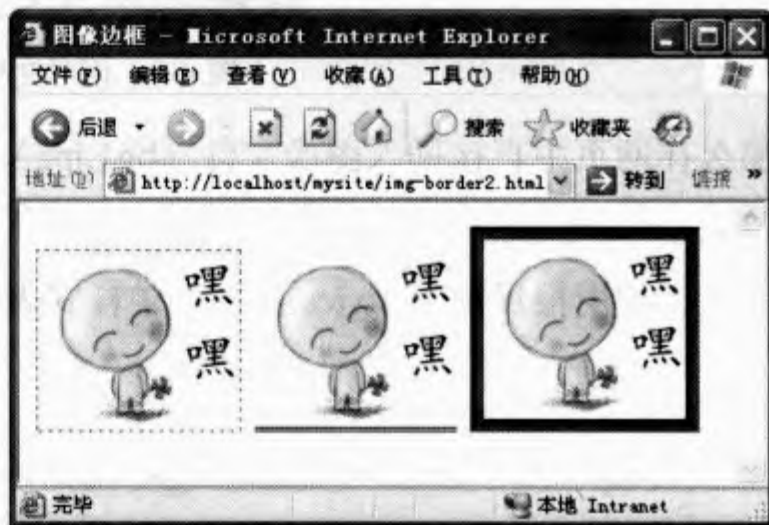


图 4.8 CSS 定义边框样式

在 CSS 2 版本中提供了如下边框样式：

- `dotted`: 显示为点线。
- `dashed`: 显示为虚线。
- `solid`: 显示为实线。
- `double`: 显示为双线边框，两条单线与其间隔的和等于 `border-width` 的值。
- `groove`: 根据 `border-color` 的值绘制 3d 凹槽。
- `ridge`: 根据 `border-color` 的值绘制 3d 凸槽。
- `inset`: 根据 `border-color` 的值绘制 3d 凹边。
- `outset`: 根据 `border-color` 的值绘制 3D 凸边。

在 CSS 中定义不显示边框，可以设置 `border` 属性 `none` 来实现。

4.2.2 巧设边距

边框好看，但是不容易设计好。很多时候，我们不应该为了边框而设计边框，巧设边框应该顾及网页整体效果。例如，对于如图 4.9 所示的网页中白色的边框就能够使插入的图像更醒目，轻松设置，页面的审美感受就被提升了。相反如果换一种环境，再使用此法就显得刺眼了。

到那时你就不用估计包含元素的大小了。所设计的效果如图 4.11 所示。



图 4.11 设计相册边框效果

4.2.3 为图像增加阴影

CSS 没有提供设计图像阴影的属性, 同时 W3C 标准组织也没有意向把图像特效作为 CSS 未来版本支持的一部分进行开发。不过 IE 浏览器却自定义了一套 CSS 滤镜, 这些滤镜试图简单模拟 Photoshop 中图像处理特效来实现网页图像的特效设计, 其中阴影就是其中的一种。自然这些滤镜也只能在 IE 浏览器中获得支持。

例如, 在下面代码中 IE 使用 filter 属性来定义 CSS 滤镜, 其中 Shadow 滤镜用来定义元素的阴影, 该滤镜的第 1 个参数表示阴影的颜色, 第 2 个参数表示阴影的角度, 演示效果如图 4.12 所示。

```
<style>
#div1 {
    position:absolute;                                /* 绝对定位, 必须设置项 */
    filter:Shadow(Color=#00ff00, Direction=135);      /* 定义阴影特效 */
    padding:12px;                                       /* 增加内边距避免图像覆盖阴影 */
}
img {
    border:solid 1px #00ff00;
}
</style>
<div id="div1">
    
</div>
```

你还可以使用 filter 属性的 alpha 滤镜来定义更逼真的阴影效果 (如图 4.13 所示)。这里主要使用透明滤镜 alpha 来设置 2 个渐变滤镜, 分别应用右边和底边效果。

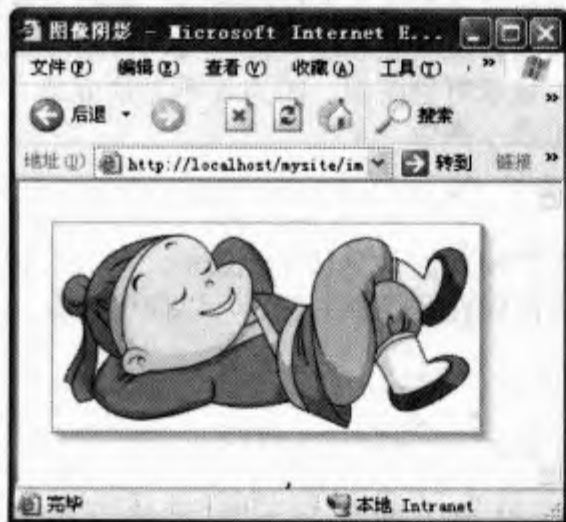


图 4.12 图像阴影效果 1



图 4.13 图像阴影效果 2

alpha 与在 Flash 和 Photoshop 中的意思是相同的。它们的作用基本类似，就是把一个目标元素与背景混合。你可以指定一个数值来控制混合的程度。这种与背景混合通俗地说就是一个元素的透明度。通过指定坐标，可以指定点、线、面的透明度。其包含的参数说明如下：

- opacity: 表示透明度，范围从 0~100，其中 0 表示完全透明，100 表示完全不透明。
- finishopacity: 是一个可选参数，如果想要设置渐变的透明效果，就可以用来设置结束时的透明度。范围也是 0~100。
- style: 设置透明区域的形状特征。其中 0 表示统一形状，1 表示线形，2 表示放射状，3 表示长方形。
- StartX 和 StartY: 表示渐变透明效果的开始 X 和 Y 坐标。
- FinishX 和 FinishY: 表示渐变透明效果结束 X 和 Y 的坐标。

```
<style type="text/css">
#div2 {
    position:relative;          /* 相对定位，以便于实现图像重叠 */
    left:5px;                   /* 根据元素的原坐标，以左侧为起点向右移动 5 像素 */
    top:-145px;                 /* 根据元素原坐标，以底边框为起点向上移动 145 像素 */
    width:300px;               /* 宽度 */
    height:160px;              /* 高度 */
    z-index:-1;                /* 叠于其他对象底部 */
    background-color: #000000; /* 定义黑色背景 */
    filter:
        Alpha(Opacity=100, FinishOpacity=0, Style=1, StartX=0, StartY=85, FinishX=0,
        FinishY=100)           /* 应用渐变滤镜 */
        Alpha(Opacity=100, FinishOpacity=0, Style=1, StartX=90, StartY=0,
        FinishX=100, FinishY=0) /* 应用渐变滤镜 */
}
#div1 img {
    border:solid 1px #999999;
}
</style>
<div id="div1">
    
    <div id="div2"></div>
</div>
```

考虑到标准浏览器无法正确解析 IE 推出的 CSS 滤镜，不建议读者在网页中大规模使用，特别是商业网站中更不应该使用。此时要设计阴影建议不妨借助 CSS 来简单模拟阴影效果，所谓模拟就是用边框线和背景色来代替图像的阴影效果。例如，输入下面代码可以模拟一个简单的阴影效果（如图 4.14 所示）。



图 4.14 模拟图像阴影效果

```
<style type="text/css">
#div1 {
    background: #ccc;                /* 定义浅灰色背景 */
    float:left;                     /* 浮动显示 */
}
#div1 img {
    border: 1px solid #666;          /* 定义灰色边框线, 该值可调 */
    position: relative;              /* 相对定位 */
    top: -6px;                       /* 向上移动图像 */
    left: -8px;                      /* 向左移动图像, 该值可调 */
}
</style>
<div id="div1">
    
</div>
```

4.2.4 为img元素定义默认阴影样式

针对 CSS 在定义图像阴影时的局限性, 成熟的设计师更喜欢使用背景图像来代替各种技巧, 其优势就是安全、逼真和方便。建议读者先在图像编辑器中设计好阴影背景图像, 然后使用 background 属性把阴影图像固定到图像的某个边上即可。

也许你为每个图像设置阴影而感到繁琐, 那么我们为什么不为 img 元素定义一个默认的阴影样式呢? 这样当你在网页中插入一个图像时, 它会自动显示为阴影效果 (如图 4.15 所示)。当然与普通的插入的图像效果比较之后 (如图 4.16 所示), 你会发现这种定义有阴影效果的图像会更真实而富有立体感, 特别实用于网上照片发布页面。



图 4.15 为图像定义默认的阴影样式

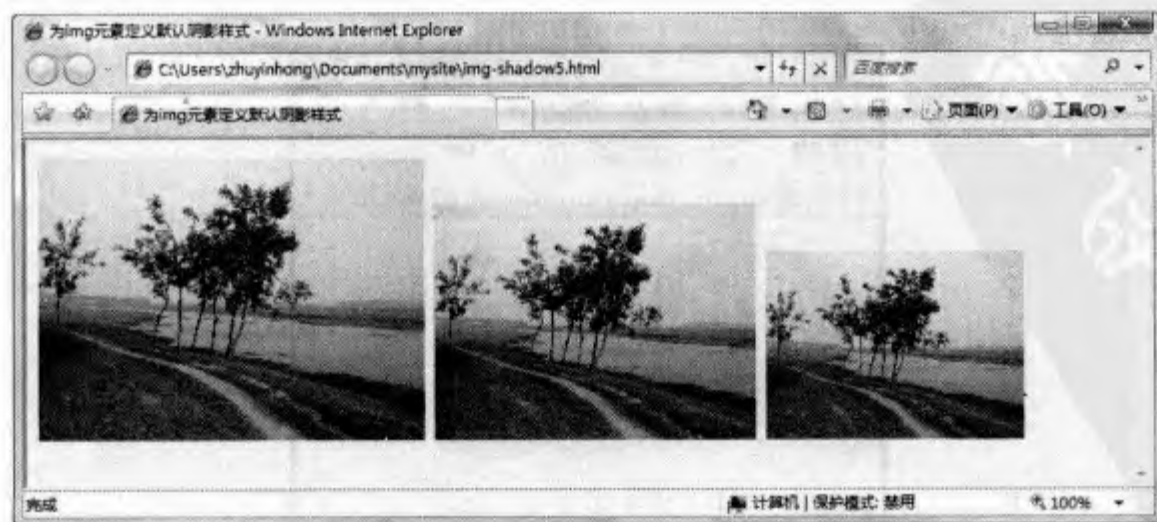


图 4.16 图像未定义阴影样式效果

其实定义这样的默认样式比较简单，首先你需要在图像编辑器中设计一个4像素高、1像素宽的渐变阴影（如图4.17所示）。



图4.17 设计一个渐变阴影图像

请注意，在定义底边内边距，考虑到底边阴影背景图像可能要占用4个像素的高度，因此要多设置4像素。左右两侧的阴影颜色可以根据网页背景色时适当调整深浅。然后在网页中定义如下样式即可。

```
img {
    background: white;                /* 白色背景 */
    padding: 5px 5px 9px 5px;        /* 增加内边距 */
    background: white url(images/shad_bottom.gif) repeat-x bottom left; /* 底边阴影 */
    border-left: 2px solid #dcd7c8;   /* 左侧浅阴影 */
    border-right: 2px solid #dcd7c8;  /* 右侧浅阴影 */
}
```

4.2.5 设计晶莹剔透的水印

IE浏览器能够支持CSS特效，但是在其他浏览器中并没有获得支持，且W3C也并没有准备把IE所提供的CSS特效作为标准进行推广，所以这些CSS特效也就没有具体的应用价值。不过主流浏览器都能够支持透明特效，虽然支持的标准不同，但最后的实现效果都是一样的。

下面以一个水印特效来演示如何设计一个兼容不同浏览器的透明效果。具体方法如下。

首先，新建一个网页文当，构建一个简单的HTML结构代码。设计一个外包装标签（<div class="watermark">）主要是为水印图片能够在照片上精确定位提供一个载体，并把它们都捆绑在一起，避免在网页中随处流动。插入的第一幅图片为照片，第二幅图片为水印图片。

```
<div class="watermark">
    
    
</div>
```

在没有定义任何样式的情况下，则显示如图4.18所示效果。

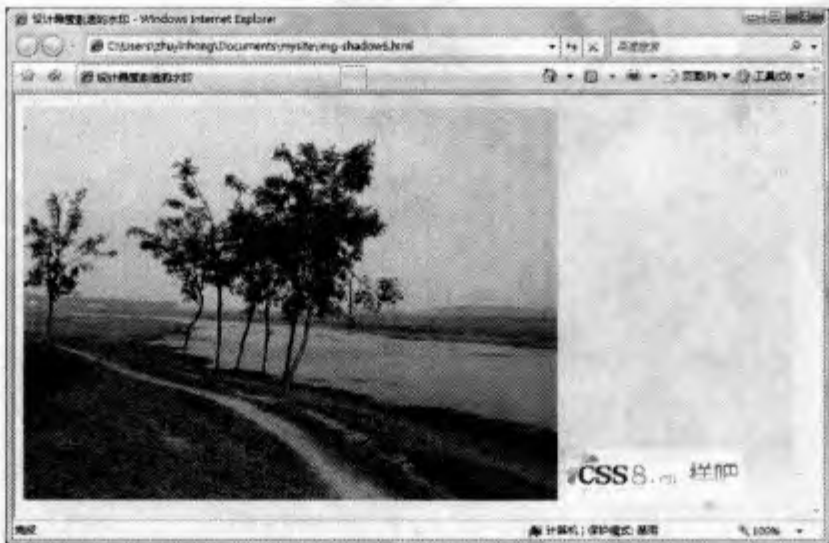


图4.18 插入图像后的效果

首先定义包含元素为相对定位，这样能够保证水印图片能够定位在照片上面。由于 `div` 元素默认显示块状态元素，宽度为 100%，此时无法精确定位内部的水印。如果固定宽度，这样就使设计失去了灵活性，你还需要确定包含元素和照片的宽度。具体样式如下：

```
.watermark { /* 包含块样式 */
    position:relative; /* 相对定位 */
    float:left; /* 向左浮动，这样包含元素能够自动包裹包含的照片 */
    display:inline; /* 行内显示，这样就避免包含元素随处浮动 */
}
```

然后利用上节介绍的方法为照片定义阴影，具体代码如下，此时就不能够为所有照片定义阴影：

```
.img {
    background: white;
    padding: 5px 5px 9px 5px;
    background: white url(images/shad_bottom.gif) repeat-x bottom left;
    border-left: 2px solid #dcd7c8;
    border-right: 2px solid #dcd7c8;
}
```

最后是本实例的技术核心，定义水印透明度和精确定位的位置，具体样式如下：

```
.img1 {
    filter:alpha(opacity=40); /* 兼容 IE 浏览器 */
    -moz-opacity:0.4; /* 兼容 Moz 和 FF 浏览器 */
    opacity: 0.4; /* 支持 CSS3 的浏览器 (FF 1.5 也支持) */
    position:absolute; /* 绝对定位 */
    right:20px; /* 定位到照片的右侧 */
    bottom:20px; /* 定位到照片的底部 */
}
```

IE 使用专有的 CSS 滤镜 `filter:alpha(opacity)` 来定义对象的透明度，而 Moz Family 浏览器使用私有属性 `-moz-opacity` 来定义，对于标准浏览器来说一般都支持标准属性 `opacity` (CSS 3 版本) 透明度可以使用百分比或者小数值。上面样式定义水印图片的透明度为 0.4，位于照片的右下角，所得效果如图 4.19 所示。



图 4.19 所定义的水印特效

这些 CSS 特效除了能够定义前景图像的透明度,也可以为背景图像定义透明度。例如,在上面实例的 HTML 结构基础上嵌套一个外套标签:

```
<div class="bg">
  <div class="watermark">
    
    
  </div>
</div>
```

在上面实例的样式表代码基础上,为最外层标签定义一个背景图像,并进行透明处理:

```
.bg {
  background:url(images/1.jpg) -1000px -1000px; /* 定义背景图像 */
  filter:alpha(opacity=40); /* 兼容 IE 浏览器 */
  -moz-opacity:0.4; /* 兼容 Moz 和 FF 浏览器 */
  opacity: 0.4; /* 支持 CSS3 的浏览器 (FF 1.5 也支持) */
  height:600px;
}
```

最后得到如图 4.20 所示的透明背景图像效果。但是这种在 Firefox 中解析时,略有一点不同。前景图像也被背景图像的透明效果同化,如图 4.21 所示。




图 4.20 在 IE 下显示的背景透明效果



图 4.21 在 Firefox 下显示的背景透明效果

4.3 图文混合排版

传统桌面排版师都知道图文混排是桌面排版中最复杂的一道工序,其难点就在于图像和文字在版面中的布局模式不同。当然现在有了高级的排版软件,这些技术难题也就迎刃而解了。网页也需要面临图像和文字相处一起的局面,传统网页布局中,设计师为了实现图像和文字的和谐相容,就一个劲地使用表格来解决,再配合元素所提供的众多属性,也能够设计出很多经典的图文版式。不过,现在使用 CSS 布局之后,这种图文混排的版式倒不是很复杂,但是对于初学者来说,依然是道难题。

4.3.1 单行图文定位

当在一行内排列着文本、图像或表单等对象时,如何使用它们垂直居中?这可是很多设计师经常面临的问题,相信你可能也曾经为此而苦恼过。

例如,在如图 4.22 所示的版面中,插入的图像按钮怎么处理都无法与前面的文本框顺利地
对齐。



图 4.22 单行图文混排问题

该表单域结构如下:

```
<div id="container">
  <div id="intro">
    <div id="pageHeader">
      <form action="" method="post">
        用户名:
        <input name="name" class="box" type="text" />
        密码:
        <input name="pass" class="box" type="text" />
        <input id="login" name="login" type="image" src="images/login.gif" />
      </form>
    </div>
  </div>
</div>
```

如果增加图像按钮的顶部外边距或内边距,即使用如下代码:

```
#login {
  margin-top:12px;
}
```

则同行的文本框和文本都一起被挤到下面去了(如图 4.23 所示),看来使用增加边距的方法是不可行的。唯一的方法是借助定位技术来解决这个问题。也就是说,我们让图像按钮处于相对或绝对定位状态,然后移动其坐标值来实现与同行文本框的水平居中,这样就不会干扰其他对象的位置了。



图 4.23 被挤到底部的文本框和图像

使用绝对定位需要涉及包含块的定义,如果参考浏览器窗口的左上角进行定位,就会出现当浏览器窗口大小发生变化时,图像按钮的位置也随之发生变化,所以这种方法不是最佳选择。此时,最好的方法是使用相对定位,也就是说,让图像按钮根据自己原来的位置进行定位。具体代码如下,所得效果如图 4.24 所示。



图 4.24 解决单行图文定位问题

```
#login {
    position:relative;          /* 相对定位 */
    top:6px;                    /* 以原来位置的顶部为参考点向下移动 6 个像素 */
}
```

4.3.2 单行图文对齐

使用定位法能够精确设置图片和文字在同一行内的位置关系。另外你还可以使用垂直对齐属性来解决这个问题。我们曾经在前面章节中讲解了vertical-align属性在块状元素内使用的局限性,不过使用该属性能够快速实现图文在垂直方向上对齐设置。

例如,针对上节介绍的示例,如果我们为图像按钮重设如下样式,也一样能够实现图文垂直居中(如图 4.25 所示)。

```
#login {
    vertical-align:middle;
}
```



图 4.25 解决单行图文对齐问题

不过细心的你也一定能够发现,使用这种方法没有定位法精确,整个图文行内的文本和文本框被轻微的向上提升了一点,以便与图像按钮垂直居中对齐。而在上节介绍的定位法中图文所在行的位置是没有发生变化的,仅是图像按钮被向下移动了一点。因此,对于版面布局精度

要求很严格的页面，建议采用上节介绍的定位法会准确点，如果页面布局要求不是很严格，使用本节介绍的方法会更方便些。

vertical-align 属性取值很多，用法比较灵活，且不同浏览器对齐解析的效果也存在很大差异。既然它在表格以及图像等对象上使用比较安全（兼容比较好），下面我们就来详细讲解一下它的取值和用法。

首先，我们建立一个简单的示例，以便更直观的进行比较，代码如下：

```
<STYLE type=text/css media=all>
body {
    font:300% "Times New Roman", Times, serif;
    background-image:url(images/body_bg.gif);
}
p {
    text-align:right;
    margin-right:2em;
}
.pl img { vertical-align:auto; }
</STYLE>
<p class="pl">vertical-align:auto;</p>
```

然后分别替换图像样式中的 **vertical-align** 属性取值，比较不同垂直对齐方式的效果。

- **auto**：根据默认值采取对齐方式。一般浏览器都采用基线对齐（如图 4.26 所示）。
- **baseline**：将支持 **valign** 特性的对象的内容与基线对齐（如图 4.27 所示），该属性值为默认值。当图像以基线对齐时，它的底边与文本的基线对齐。

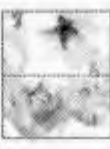
vertical-align:auto; 

图 4.26 垂直自动对齐

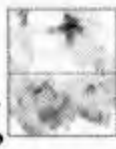
vertical-align:baseline; 

图 4.27 垂直基线对齐

在西方字符中，每种字体都有自己的基线。所谓基线对齐，就是各种行内元素都保持其基线与父元素的基线是对齐的。例如，如果有图片的话，则图片的底部都会对齐父元素的基线。

- **sub**：垂直对齐文本的下标。在 IE 版本中显示为如图 4.28 所示，在 Firefox 等标准浏览器显示为 4.29 所示。

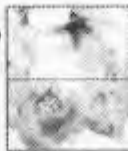
vertical-align:sub; 

图 4.28 IE 中垂直对齐文本下标


vertical-align:sub; 

图 4.29 Firefox 中垂直对齐文本下标

在 HTML 中有一个 **<sub>** 的下标标签，它能够把包含的文本缩小并向下移动，形成下标样式显示，我们可以看到 IE 浏览器正是根据 **<sub>** 下标标签的显示样式来解析图像。而在 Firefox 等标准浏览器中解析结果恰恰相反，它们把对象的底部对齐到文本的最底部，即下标的位置。

- **super**：垂直对齐文本的上标。与 **sub** 取值相反，在 IE 版本中显示为如图 4.30 所示，在 Firefox 等标准浏览器显示为如图 4.31 所示，不过不同浏览器在上标中解析效果有些接近。

vertical-align:super;



图 4.30 IE 中垂直对齐文本上标

vertical-align:super;

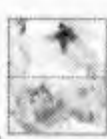


图 4.31 Firefox 中垂直对齐文本上标

- top: 将支持 valign 特性的对象的内容与对象顶端对齐。也就是说把图像的顶端与文本行的顶端对齐, 如果在默认行高下, 你可能看不出不同浏览器的差异。但是如果把行高加大之后, 你会发现在 IE 6、IE 7 和 Firefox 中所显示的效果都是不同的。例如, 定义 `p { line-height: 140px; }` 样式, 在 IE 6 中没有受到行高的影响; 而在 IE 7 中, 行高发生了变化, 但是图像依然保持默认行高时的对齐位置 (如图 4.32 所示), 而在 Firefox 中, 图像为了对齐行高的顶端, 则向上移动了位置 (如图 4.33 所示)。

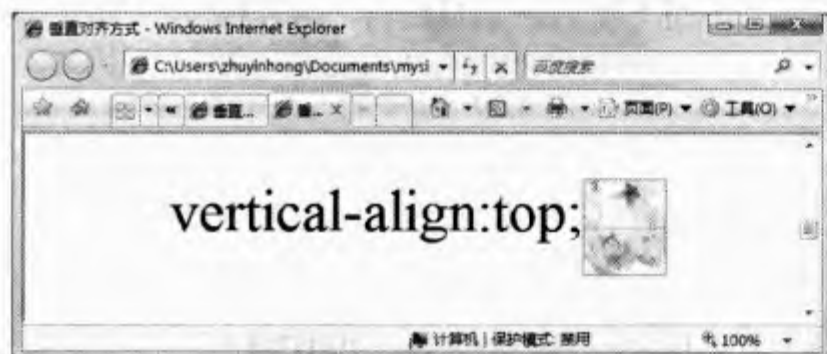


图 4.32 IE 7 中垂直对齐顶端



图 4.33 Firefox 中垂直对齐顶端

- text-top: 将支持 valign 特性的对象的文本与对象顶端对齐。该属性值与 top 类似, 但是作用对象不同, 它是把文本与对象顶端对齐, 也就是说把文本对齐到顶端, 在 IE 下显示如图 4.34 所示, 在 Firefox 中显示如图 4.35 所示。

vertical-align:text-top;



图 4.34 IE 7 中垂直对齐文本到顶端

vertical-align:text-top;

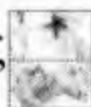


图 4.35 Firefox 中垂直对齐文本到顶端

- middle: 将支持 valign 特性的对象的内容与对象中部对齐。所有浏览器的解析效果都是一样的, 即文本中线、图像中线与包含对象 (段落标签) 的中线对齐, 如图 4.36 所示。
- bottom: 将支持 valign 特性的对象的内容与对象底端对齐。它与 top 属性值是相反的, 可以参阅 top 属性说明。例如, 定义 `p { line-height: 140px; }` 样式后, 在 Firefox 中, 图像为了对齐行高的底端, 则向下移动了位置 (如图 4.37 所示)。

vertical-align:middle;

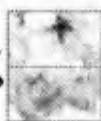


图 4.36 垂直居中对齐

vertical-align:bottom;



图 4.37 Firefox 中垂直对齐底端

- text-bottom: 将支持 valign 特性的对象的文本与对象底端对齐。即对象内文本和图像都

对齐到对象的底端，所有浏览器所解析的效果是相同的，如图 4.38 所示。

另外，你还可以指定一个数值或者百分比值来定义由基线算起的偏移量。基线对于数值来说为 0，对于百分数来说就是 0%。可以取负值。例如，输入下面代码，则分别显示如图 4.39～图 4.41 所示。

```
<STYLE type=text/css media=all>
.p1 img { vertical-align:1em; }
.p2 img { vertical-align:-1em; }
.p3 img { vertical-align:100%; }
</STYLE>
<p class="p1">vertical-align:1em;</p>
<p class="p2">vertical-align:-1em;</p>
<p class="p3">vertical-align:100%;</p>
```

vertical-align:text-bottom;



图 4.38 垂直居中对齐

vertical-align: 1em;

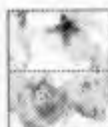


图 4.39 取正值对齐

vertical-align:-1em;

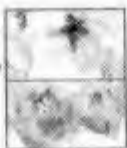


图 4.40 取负值对齐

vertical-align:100%;



图 4.41 取百分比弹性对齐

目前所有浏览器都支持该属性值，但是在 IE 6 版本以下浏览器却不支持该属性值。

考虑到兼容性问题，当你准备使用 vertical-align 属性垂直对齐图像在行内显示时，建议选用 baseline、middle、super 或取值来定义，其他属性值尽量少用。

4.3.3 图文环绕

img 默认为行内元素，也就是说它与文本一样仅能在行内显示。当然任何事情有利必有弊，图像的这种显示属性决定了它无法直接实现图文排版。在传统表格布局中，设计师多利用 标签提供的众多属性使其与文本形成环绕关系，从而解决图文混排时的版式问题。这种方法局限性比较大，现在使用 CSS 就容易多了。

利用 CSS 提供的 float 属性可以定义图像浮动起来，从而摆脱图像对文本行的羁绊。这样设计师就可以发挥想象力，借助 CSS 各种修饰技术能够设计一幅完美的版式。

float 表示浮动的意思，它的取值包括 left（向左浮动）、right（向右浮动）、none（不浮动）。例如，对于下面这个 HTML+CSS 的页面版式，其中所包含的图像被浮动到包含元素的左侧显示（如图 4.42 所示）。

```
<STYLE type=text/css media=all>
body {
  background-image:url(images/body_bg.gif); /* 网页背景 */
  text-align:center; /* IE 中网页居中 */
  color:#454545; /* 灰色字体 */
}
```



```

#preamble {
    width:600px;                /* 固定宽度 */
    margin:0 auto;              /* 非 IE 中网页居中 */
}
p {
    text-indent:2em;            /* 首行缩进 2 个字 */
    text-align:left;           /* 段落文本左对齐 */
}
#preamble img {                /* 图像向左浮动显示 */
    float:left;
}
</STYLE>
<div id="preamble"> 
    <h2><span>启蒙之路</span></h2>
    <p class="p1"><span>不同浏览器随意定义标签，导致无法相互兼容的<acronym
title="document object model">DOM</acronym>结构，或者提供缺乏标准支持的<acronym
title="cascading style sheets">CSS</acronym>等陋习随处可见，如今当使用这些不兼容的标签和
样式时，设计之路会一路坎坷。</span></p>
    <p class="p2"><span>现在，我们必须清除以前为了兼容不同浏览器而使用的一些过时的小技巧。感谢<acronym title="world wide web consortium">W3C</acronym>、<acronym
title="web standards project">WASP</acronym>等标准组织，以及浏览器厂家和开发师们的不懈努力，我们终于能够进入 Web 设计的标准时代。</span></p>
    <p class="p3"><span>CSS Zen Garden（样式表禅意花园）邀请您发挥自己的想象力，构思一个专业级的网页。让我们用慧眼来审视，充满理想和激情去学习 CSS 这个不朽的技术，最终使自己能够达到技术和艺术合而为一的最高境界。</span></p>
</div>

```



图 4.42 图文环套的效果

如果不定义图像浮动，你会发现图像以行内元素的显示方式位于顶部。当然你也可以让图像向右浮动，并控制图像显示的大小。例如，定义图像样式如下，并把图像插入到段落文本中间，则显示效果如图 4.43 所示。

```

#preamble img {
    float:right;                /* 图像向右浮动显示 */
    width:100px;               /* 固定宽度 */
    height:131px;              /* 固定高度 */
}

```

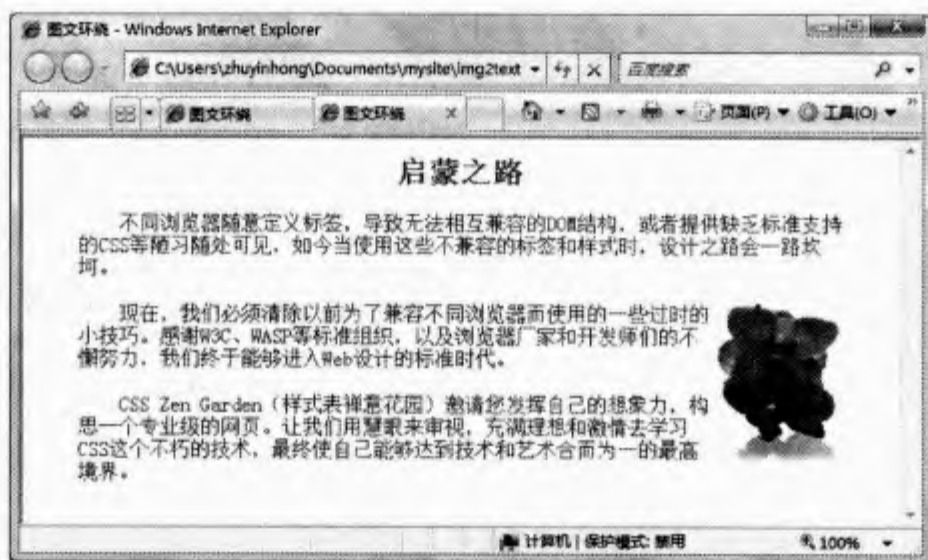


图 4.43 图像浮动到文本段的右下角效果

4.3.4 调整图文环绕的间距

如果希望调整图文之间的间距，你可以使用 `margin` 或 `padding` 属性调整图像四周的空隙，以增加图文之间的间距。例如，为图像增加如下样式，则会得到如图 4.44 所示的显示效果。

```
#preamble img {
    float:right;
    width:100px;
    height:131px;
    margin:28px 2em 0 2em; /* 增加图像左右空隙为 2 个字，顶部空隙为 28 像素 */
}
```

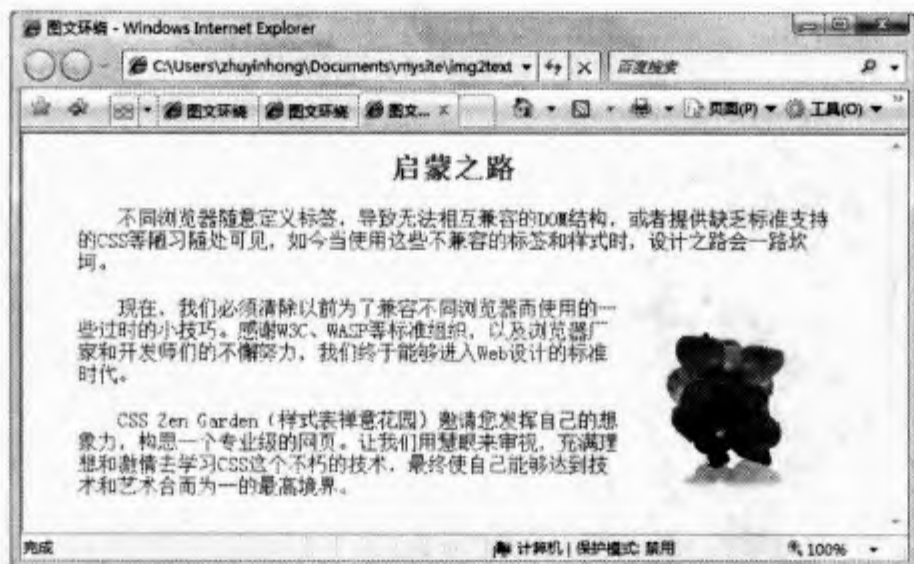


图 4.44 增大图文之间的间距效果

4.3.5 不规则图文环绕

桌面排版软件中一般都会提供多种样式的图文环绕功能，有一种是文本能够根据不规则图形进行环绕，而不是矩形区域。CSS 虽然没有提供这样的功能，但是我们可以来模拟这种不规则图文环绕样式。虽然操作起来比较麻烦，但是对于初学者来说，能够熟练驾驭 CSS 排版具有重要参考价值。

如何实现图文环绕呢？要实现这样的功能，首先我们来分析图像浮动和文本流动的位置关系。在上面的示例中你可能也看到，当图像浮动显示时，文本会环绕在图像的四周进行分布。如果我们定义多个浮动元素，并设置不同宽度，则文本就会不规则的环绕在这些浮动元素的周围（如图 4.45 所示）。

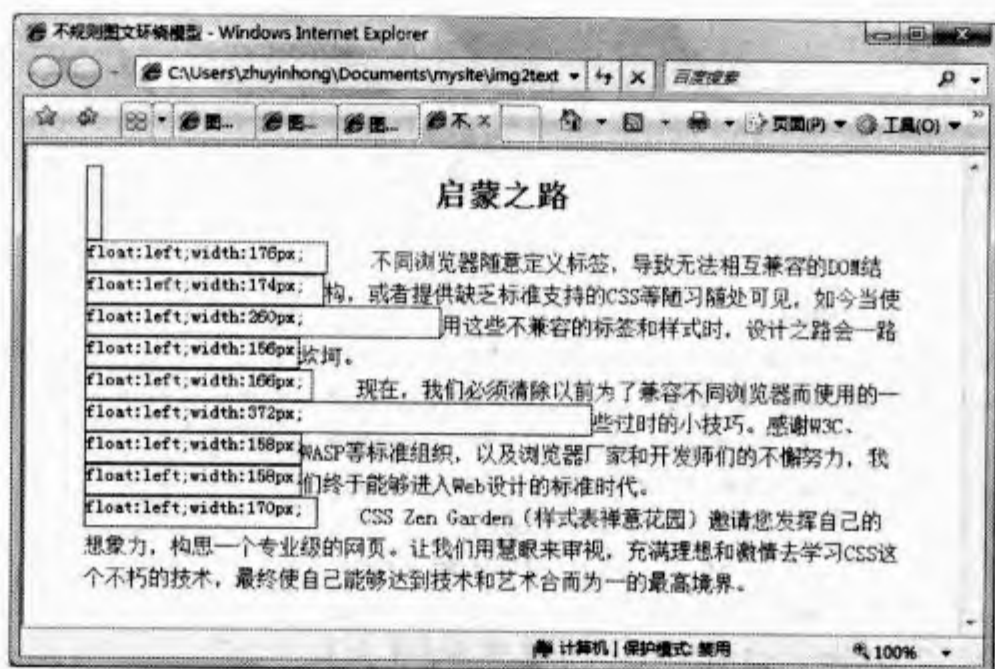


图 4.45 模拟不规则图文环绕模型

实现上述环绕模型的 HTML+CSS 代码如下：

```
<STYLE type=text/css media=all>
body {
    background-image:url(images/body_bg.gif); /* 网页背景 */
    text-align:center; /* IE 下网页居中 */
    color:#454545; /* 网页字体颜色 */
}
#preamble {
    width:600px; /* 固定网页高度 */
    margin:0 auto; /* 非 IE 下网页居中 */
}
h2 {
    line-height:40px; /* 固定行高，以便于精确设置第一个浮动元素的高度 */
    margin-bottom:12px; /* 定义标题与正文之间的间距 */
}
p {
    text-indent:2em; /* 段落首行缩进 2 个字 */
    text-align:left; /* 段落左对齐 */
    line-height:24px; /* 固定行高，便于精确设置第二个及后面浮动元素高度 */
    margin:0; /* 清除段落上下默认边距，以便精确对齐浮动元素 */
}
#preamble em {
    float:left; /* 向左浮动 */
    clear:left; /* 清除左侧浮动，避免浮动元素并列显示 */
    height:22px; /* 定义浮动元素的高度，以便与文本行同高 */
    overflow:hidden; /* 隐藏多余的高度 */
    border:solid 1px red; /* 定义边框线，以便于观察 */
    color:blue; /* 浮动元素内文本颜色为蓝色 */
    font-size:14px; /* 字体大小 */
    font-style:normal; /* 清除 em 元素默认的倾斜样式 */
    text-align:left; /* 文本左对齐 */
}
.l1 {width:10px;height:54px!important;} /* 第 1 个浮动元素宽度，并强制与标题行同高 */
.l2 {width:176px;} /* 第 2 个浮动元素宽度 */
.l3 {width:174px;} /* 第 3 个浮动元素宽度 */
.l4 {width:260px;} /* 第 4 个浮动元素宽度 */
.l5 {width:156px;} /* 第 5 个浮动元素宽度 */
.l6 {width:166px;} /* 第 6 个浮动元素宽度 */
```

```
.17 {width:372px;} /* 第 7 个浮动元素宽度 */
.18 {width:158px;} /* 第 8 个浮动元素宽度 */
.19 {width:158px;} /* 第 9 个浮动元素宽度 */
.l10 {width:170px;} /* 第 10 个浮动元素宽度 */
```

```
</STYLE>
```

```
<div id="preamble">
```

```
  <em class="l1"></em>
```

```
  <em class="l2">float:left;width:176px;</em>
```

```
  <em class="l3">float:left;width:174px;</em>
```

```
  <em class="l4">float:left;width:260px;</em>
```

```
  <em class="l5">float:left;width:156px;</em>
```

```
  <em class="l6">float:left;width:166px;</em>
```

```
  <em class="l7">float:left;width:372px;</em>
```

```
  <em class="l8">float:left;width:158px;</em>
```

```
  <em class="l9">float:left;width:158px;</em>
```

```
  <em class="l10">float:left;width:170px;</em>
```

```
  <h2><span>启蒙之路</span></h2>
```

<p class="p1">不同浏览器随意定义标签，导致无法相互兼容的<acronym title="document object model">DOM</acronym>结构，或者提供缺乏标准支持的<acronym title="cascading style sheets">CSS</acronym>等陋习随处可见，如今当使用这些不兼容的标签和样式时，设计之路会一路坎坷。</p>

<p class="p2">现在，我们必须清除以前为了兼容不同浏览器而使用的一些过时的小技巧。感谢<acronym title="world wide web consortium">W3C</acronym>、<acronym title="web standards project">WASP</acronym>等标准组织，以及浏览器厂家和开发师们的不懈努力，我们终于能够进入 Web 设计的标准时代。</p>

<p class="p3">CSS Zen Garden（样式表禅意花园）邀请您发挥自己的想象力，构思一个专业级的网页。让我们用慧眼来审视，充满理想和激情去学习 CSS 这个不朽的技术，最终使自己能够达到技术和艺术合而为一的最高境界。</p>

```
</div>
```

上面的代码看起来很多，实际上设计并不复杂，它是在网页中增加多个额外的元素，并定义为浮动显示，并强制它们按规律的显示，通过定义不同的宽度，以强制文本按某个不规则的形状显示。

为了精确控制，这里建议读者把浮动元素的高度定义为行高，这样保证每个浮动元素的高度等于对应的一行行高，如果某行显示为标题或者需要增加间距，则明确这些行的高度或间距，然后在对应的浮动元素中设置相同高度进行对应。

有了这样的思路，我们就可以在 HTML 结构中<div id="preamble">的第一个位置插入对应的图像，然后用绝对定位的方式定位图像靠左分布，具体样式如下：

```
#preamble img {
  position:absolute; /* 绝对定位图像 */
  z-index:100; /* 层叠在上面 */
  left:0px; /* 靠左对齐 */
  top:40px; /* 设置距离顶部 40 像素 */
}
```

如果仅定义图像绝对定位，这时图像会以 body 元素为包含块，也就是说将根据浏览器窗口左上角来进行定位，这样图像有可能跑到指定的区域的外面，因此我们还必须定义包含元素<div id="preamble">为包含块，这样图像就根据该元素进行定位了。代码如下：

```
#preamble {
  position:relative; /* 相对定位，定义为包含块 */
}
```

最后清除模型中为浮动元素的边框线和其中显示的内容，同时设置浮动元素的高度为 24

像素，因为上面示例中定义为 22 像素。因为去掉边框之后，浮动元素的总高（为 22 像素）不等于文本行的高度（为 24 像素）。经过这样的设置，显示效果如图 4.46 所示。

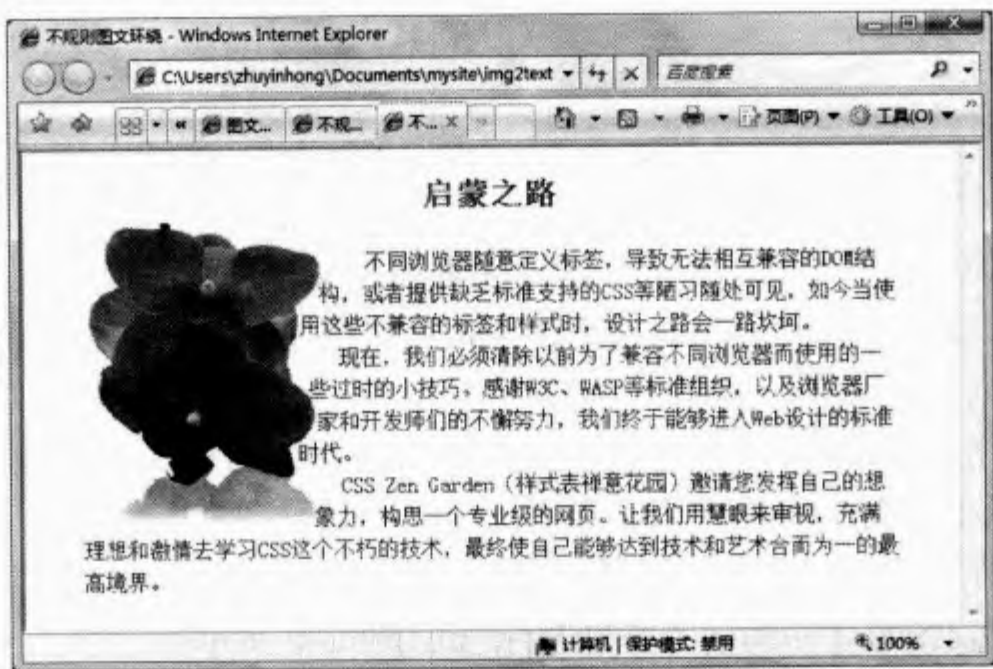


图 4.46 模拟不规则图文环绕效果

提示

要根据所插入的图像的形狀随时调整浮动元素的宽度，以便于文本看起来能够紧贴着图像的四周。详细示例可以参阅光盘示例 img2text_10.html 文件。

4.4 隐形的翅膀——背景图像

在传统表格布局中，设计师通过切图能够设计出无数精美的页面，如今借助 CSS 精确控制背景图像的能力，你一样能够设计出精美的标准页面。CSS 对背景图像的控制包括重复控制、定位控制和层叠控制等，下面我们就来探索 CSS 是如何控制背景图像的。

4.4.1 控制背景图像的重复显示

背景图像通过 background 或 background-repeat 属性定义，在第 1 节中曾经有过说明，这里就不再举例了。任何元素都可以定义背景图像，这与传统表格布局中只能够为网页（<body> 标签）、表格或单元格定义背景不同，看来 CSS 的功能确实很强大。

例如，为一个 div 元素定义背景图像，则显示效果如图 4.47 所示。

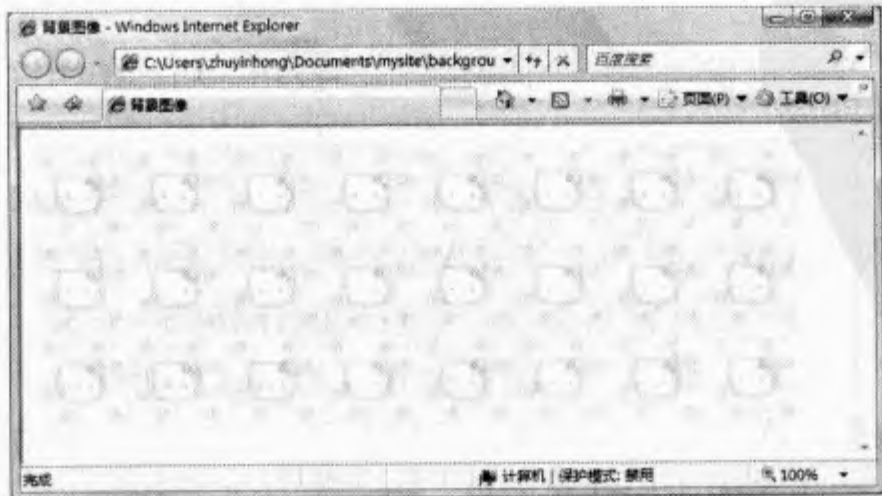


图 4.47 元素背景图像默认显示效果

```
<style type="text/css">
.bg {
    width:640px;
    height:240px;
    background:url(images/bg3.jpg);
}
</style>
<div class="bg"></div>
```

通过图示效果你也能够看到，当一个网页元素被定义了背景图像之后，背景图像会重复铺满整个元素的区域。这对于设置网页背景来说是很不错的方法，但是如果希望图像不重复显示，或者仅沿着一个方向重复平铺，该怎么办呢？

这时你可以利用 `background-repeat` 属性来强制背景图像的重复方式。该属性取值包括：

- `repeat`：默认值，背景图像在纵向和横向上平铺显示。
- `no-repeat`：背景图像不平铺，仅显示当前图像（如图 4.48 所示）。
- `repeat-x`：背景图像仅在横向上平铺显示（如图 4.48 所示）。
- `repeat-y`：背景图像仅在纵向上平铺显示（如图 4.48 所示）。

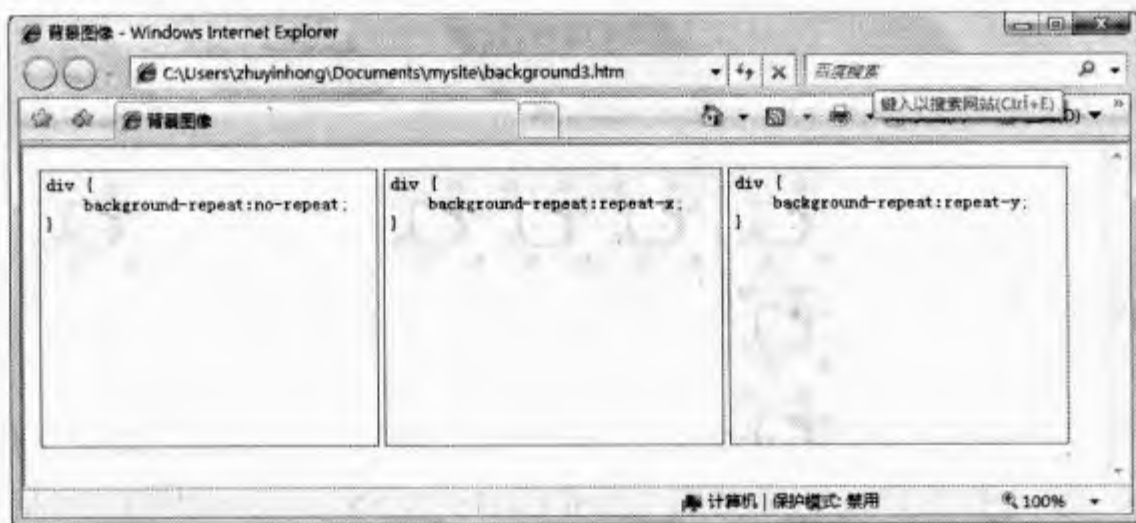


图 4.48 元素背景图像不同显示效果

当然你也可以在 `background` 复合属性中定义背景图像以及其重复显示方式。例如，针对上面的代码可以改写为：

```
background:url(images/bg3.jpg) repeat-x;
```

它与 `font` 复合属性一样，可以增加多个不同类型属性的值，实现快速定义样式的目的。

4.4.2 背景图像水平平铺应用示例

背景图像平铺显示在网页设计是一个比较常用的技巧，巧妙使用这个技术，你能够设计出很多富有立体感的平铺效果，通过它可以制作栏目边框或者栏目标题背景等。

如果网页宽度或者栏目宽度不固定，这时非常适合使用背景图像水平平铺来设计栏目或版块区域的背景。例如，对于如图 4.49 所示的版权信息栏，就是利用背景图像的水平平铺实现的。实现这样的效果难点在于如何设计背景图像。假设你的计算机中安装了 Photoshop CS3 或其他版本。请打开并新建一个文档，设计一个渐变样式（如图 4.50 所示），在文档中从上到下应用渐变即可，最后把图像裁切成宽度为 1 像素，高度保持不变，另存为 GIF 格式图像即可。最后在页面中版权信息栏包含元素中应用背景图像，并设置水平平铺即可。

```
background:url(images/footer_bg.gif) repeat-x;
```




图 4.49 背景图像水平平铺示例效果

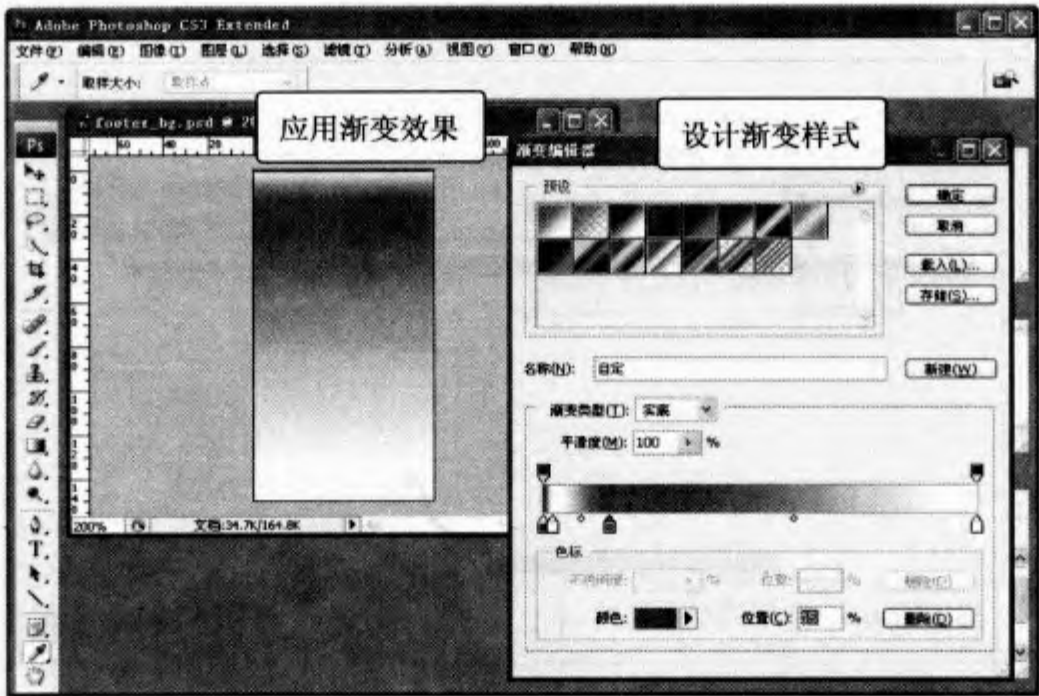


图 4.50 在 Photoshop 中设计渐变背景图像

4.4.3 背景图像垂直平铺应用示例

与水平平铺对应，如果某个栏目或版块宽度固定，而高度难以确定时，可以通过背景图像垂直平铺来设计版块整体效果。很多时候，网页设计师喜欢使用这种方式设计栏目之间个性分隔效果。

例如，对于如图 4.51 所示的效果中，左侧的“公司公告”栏目宽度是固定的，但是其高度可能会根据需要进行随时进行调整，为了适应这种需要，我们不妨利用垂直平铺来设计这个效果。



图 4.51 背景图像垂直平铺网页效果

首先，把“公司公告”栏目分隔为上、中、小三块，设计上和下为固定宽度，而中间块为

可以随时调整高度。设计的结构如下：

```
<div id="call">
  <div id="call_tit">公司公告</div>
  <div id="call_mid"></div>
  <div id="call_btm"></div>
</div>
```

所实现的样式表如下，最后经过调整中间块元素的高度以形成不同的高度的广告牌，演示效果如图 4.52 所示。

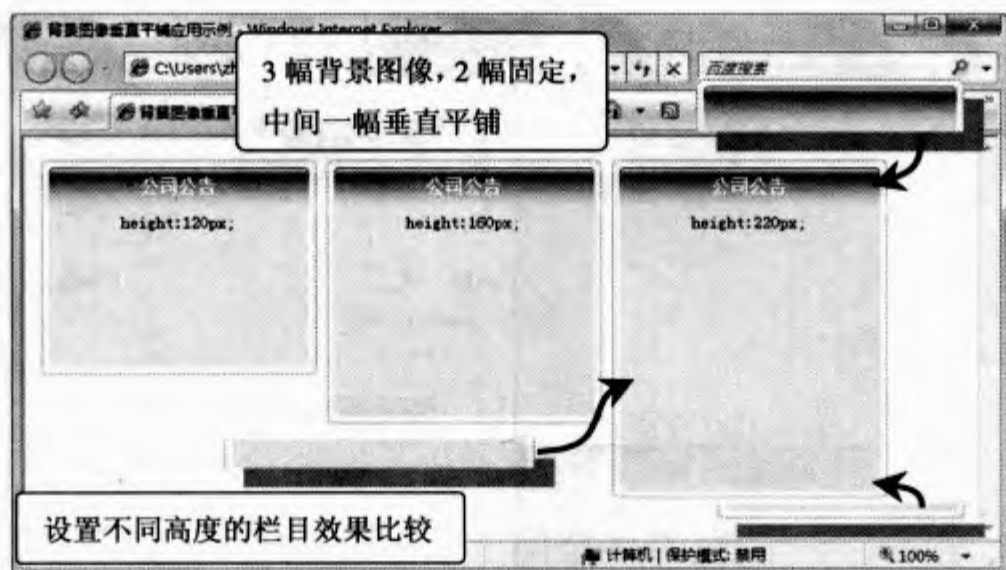


图 4.52 背景图像垂直平铺示例模拟效果

```
<style type="text/css">
#call {
  width:218px; /* 固定宽度 */
  font-size:14px; /* 字体大小 */
}
#call_tit {
  background:url(images/call_top.gif); /* 头部背景图像 */
  background-repeat:no-repeat; /* 不平铺显示 */
  height:43px; /* 固定高度, 与背景图像高度一致 */
  color:#fff; /* 白色标题 */
  font-weight:bold; /* 粗体 */
  text-align:center; /* 居中显示 */
  line-height:43px; /* 标题垂直居中 */
}
#call_mid {
  background-image:url(images/call_mid.gif); /* 背景图像 */
  background-repeat:repeat-y; /* 垂直平铺 */
  height:160px; /* 可自由设置的高度 */
}
#call_btm {
  background-image:url(images/call_btm.gif); /* 底部背景图像 */
  background-repeat:no-repeat; /* 不平铺显示 */
  height:11px; /* 固定高度, 与背景图像高度一致 */
}
</style>
```

4.4.4 精确定位背景图像

在默认状态下，背景图像会位于定义元素的左上角，当然 CSS 也支持定位图像，你可以使用 `background-position`、`background-position-x` 或 `background-position-y` 来定位背景图像的位置。

这3个属性取值可以为精确值、百分比,也可以使用背景图像定位关键字。背景图像定位关键字包括: **top** (顶端)、**center** (中间)、**bottom** (底部)、**left** (左侧) 和 **right** (右侧)。

使用这些属性时,你应该先定义 **background-image** 属性。对于 **background-position** 来说,取值包括2个值,分别表示横向定位和纵向定位。例如,输入下面代码可以定位网页背景图像居中显示:

```
body {
    background-image:url(images/bg3.jpg);          /* 指定背景图像 */
    background-repeat:no-repeat;                  /* 禁止平铺显示 */
    background-position:center 50%;               /* 居中显示 */
}
```

其中关键字 **center** 和 **50%** 显示效果都是相同的。实际利用 **background** 复合属性可以把上面规则合并在一起:

```
body {
    background:url(images/bg3.jpg) no-repeat center 50%;
}
```

background 复合属性中多个并列显示的值之间没有严格的顺序,你可以随意调整它们的位置关系。但是对于“**center 50%**”来说必须紧邻在一起,且相互位置不能够随意调换,第1个值表示横向定位,第2个值表示纵向定位。

而对于 **background-position-x** 和 **background-position-y** 属性来说,只需要指定一个值即可,用法与 **background-position** 属性相同。既然有了 **background-position** 属性,为什么还需要这两个属性呢?这是因为有时候你可能需要仅定位背景图像在 **x** 轴或 **y** 轴方向上的位置,而使用 **background-position** 或 **background** 属性则必须设置2个值,所以在特殊情况不是很适合。

下面我们来做一个背景图像定位的综合练习。如图4.53所示,这是利用4个背景图像拼接起来的一个栏目版块。这些背景图像分别被定位到栏目的4个边上,形成一个圆角的矩形,并富有立体感。实例所用到的背景图像请读者参阅光盘实例 (**background4.html**)。

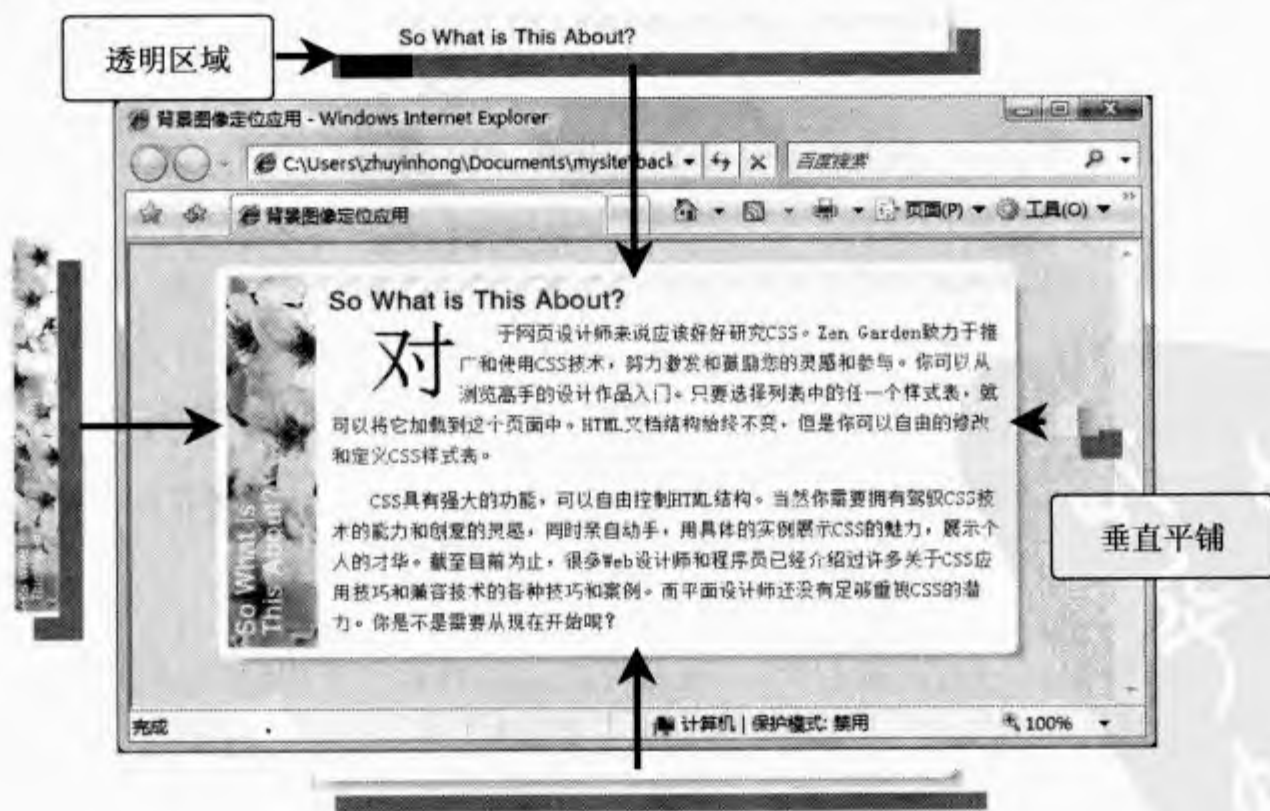


图 4.53 背景图像定位综合应用

实例所用到的 HTML 结构代码如下:

```

<div id="explanation">
  <h3><span>这是什么? </span></h3>
  <p class="p1"><span><span class="first">对</span>于网页设计师来说应该好好研究
<acronym title="cascading style sheets">CSS</acronym>。Zen Garden 致力于推广和使用 CSS 技术,
努力激发和鼓励您的灵感和参与。你可以从浏览高手的设计作品入门。只要选择列表中的任一个样式表,就可以将它
加载到这个页面中。<acronym title="hypertext markup language">HTML</acronym>文档结构始终不变,
但是你可以自由的修改和定义<acronym title="cascading style sheets">CSS</acronym>样式表。
</span></p>
  <p class="p2"><span><acronym title="cascading style sheets">CSS</acronym>具有强
大的功能,可以自由控制 HTML 结构。当然你需要拥有驾驭 CSS 技术的能力和创意的灵感,同时亲自动手,用具体
的实例展示 CSS 的魅力,展示个人的才华。截至目前,很多 Web 设计师和程序员已经介绍过许多关于 CSS 应用技巧
和兼容技术的各种技巧和案例。而平面设计师还没有足够重视 CSS 的潜力。你是不是需要从现在开始呢?
</span></p>
</div>

```

根据这个 HTML 结构所设计的 CSS 样式表如下, 请注意背景图像的定位方法:

```

<STYLE type="text/css" media="all">
body { /* 定义网页背景色、居中显示、字体颜色 */
  background:#DFDFDF; text-align:center; color:#454545;
}
p, h3 { margin:0; padding:0; } /* 清除段落和标题的默认边距 */
#explanation {
  background-color:#ffffff; /* 白色背景, 填充所有区域 */
  background-image:url(images/img_explanation.jpg); /* 指定背景图像 */
  background-position:left bottom; /* 定位背景图像位于左下角 */
  background-repeat:repeat-y; /* 在垂直方向上平铺背景图像 */
  width:546px; /* 固定栏目宽度 */
  margin:0 auto; /* 栏目居中显示 */
  font-size:13px; line-height:1.6em; text-indent:2em; /* 定义栏目内字体属性 */
}
#explanation h3 {
  background:url(images/title_explanation.gif) no-repeat; /* 顶部背景图像, 不平铺 */
  height:39px; /* 固定标题栏高度 */
}
#explanation h3 span { display:none; } /* 隐藏标题栏内信息 */
#explanation p { /* 定义右侧背景图像, 垂直平铺 */
  background:url(images/right_bg.gif) right repeat-y;
}
#explanation .p2 span { /* 底部背景图像, 不平铺 */
  padding-bottom:20px; /* 增加第 2 段底部内边距, 显示背景图像 */
  background:url(images/right_bottom.gif) bottom no-repeat;
}
#explanation p span { /* 定义段落文本左侧的内边距, 以便显示左侧背景图像 */
  padding:0 15px 10px 77px;
  display:block; /* 定义块状显示, 内边距才有效 */
  text-align:left; /* 文本左对齐 */
}
#explanation p .first { /* 定义首字下沉特效 */
  font-size:60px; color:#820015; line-height:1em; /* 字体显示属性 */
  float:left; /* 向左浮动 */
  padding:0; /* 清除上面样式为段落定义的内边距 */
}
</STYLE>

```

在上面的样式表中, 通过分别为不同元素定义背景图像, 然后通过定位技术把背景图像定

位到对应的四个边上,并根据需要运用平铺技术实现圆角区域效果,最后所设计的效果如图4.53所示。

4.4.5 固定背景图像在浏览器中的位置

在个人网页或博客中我们经常会看到一些被固定在浏览器窗口内的背景图像,无论页面如何上下滚动,这些背景图像总是固定不动的。实际上借助 CSS 提供的 `background-attachment` 属性你可以轻松实现这样的设计效果。

例如,在如图4.54所示的个性页面中,就是通过 `background-attachment` 属性在页面中固定一张背景图像,其中页面内的文字不断向上滚动,而背景始终静止,以期营造一种月夜、星空和祝福的美好氛围。

```
<STYLE type=text/css media=all>
body {
    background:url(images/24.jpg);          /* 插入背景图像 */
    background-attachment:fixed;             /* 固定背景图像 */
}
</STYLE>
```



图 4.54 背景图像固定应用案例一

请注意, `background-attachment` 取值包括 `scroll` 和 `fixed`, 其中 `scroll` 为默认值, 表示滚动的意思, 即随浏览器窗口的滚动条一起上下滚动。因此, `background-attachment` 属性不具备背景图像定位功能, 如果希望背景图像永远固定在窗口的右下角, 则需要结合 `background-position` 属性。例如, 针对上节示例, 我们给它增加一个固定在右下角的背景图像, 则代码如下, 所演示的效果如图4.55所示。

```
body {
    background:#DFDFDF;
    text-align:center;
    color:#454545;
    background-image:url(images/header1.gif); /* 插入背景图像 */
    background-position:right bottom;         /* 固定位置 */
    background-attachment:fixed;              /* 禁止滚动 */
    background-repeat:no-repeat;              /* 禁止平铺 */
}
```

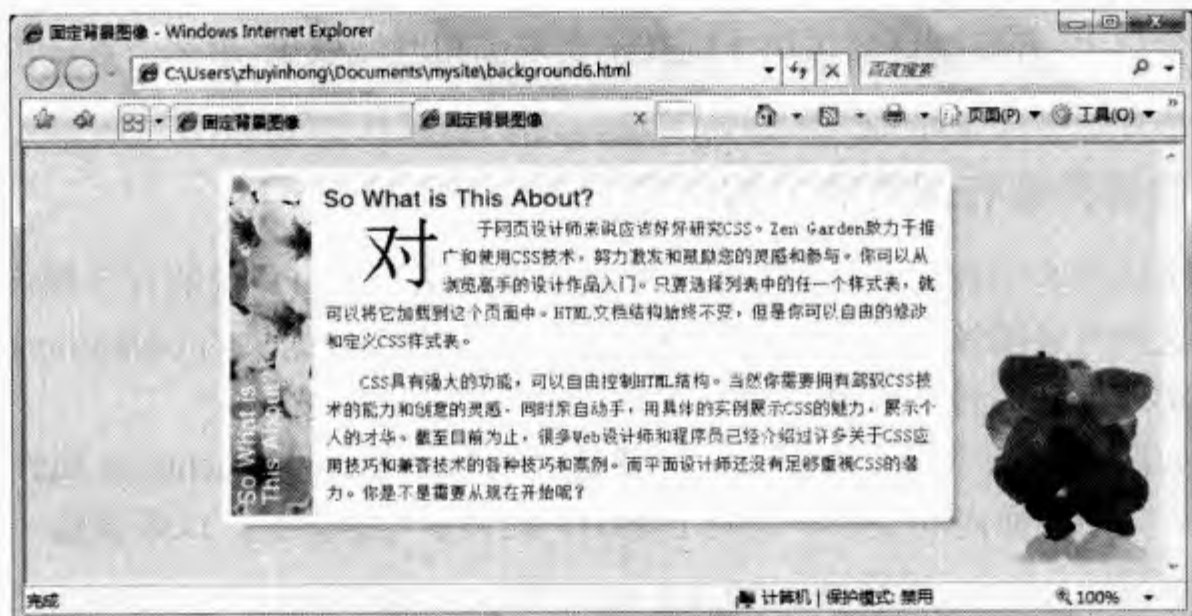


图 4.55 背景图像固定应用案例二

你也可以看到, 当我们同时为网页定义背景图像和背景颜色时, 背景图像会覆盖背景颜色, 没有被背景图像覆盖的区域为背景颜色所覆盖。

4.4.6 如何增加多个背景图像

在 Photoshop 中我们能够很随心所欲的重叠、合成图像, 但是在 CSS 中, 一个元素只能够定义一个背景图像。如果希望在元素内叠加或定义多个背景图像, 唯一可行的方法是为元素嵌套多个标记, 然后再为不同标记定义背景图像从而间接实现为一个元素增加多个背景图像的设计目标 (如图 4.56 所示)。虽然设计圆角的方法有很多种, 例如, 用一个完整的圆角背景图像来设计固定大小的区域, 使用 2 个图像延伸设计固定宽或高的区域, 也可以使用 CSS 模拟像素点拼装圆角, 或者使用 JavaScript 脚本绘制圆角等。当然, 从浏览器兼容性、易用性等角度来考虑, 通过 4 个圆角图像来设计圆角区域是最方便、最经济的方法。

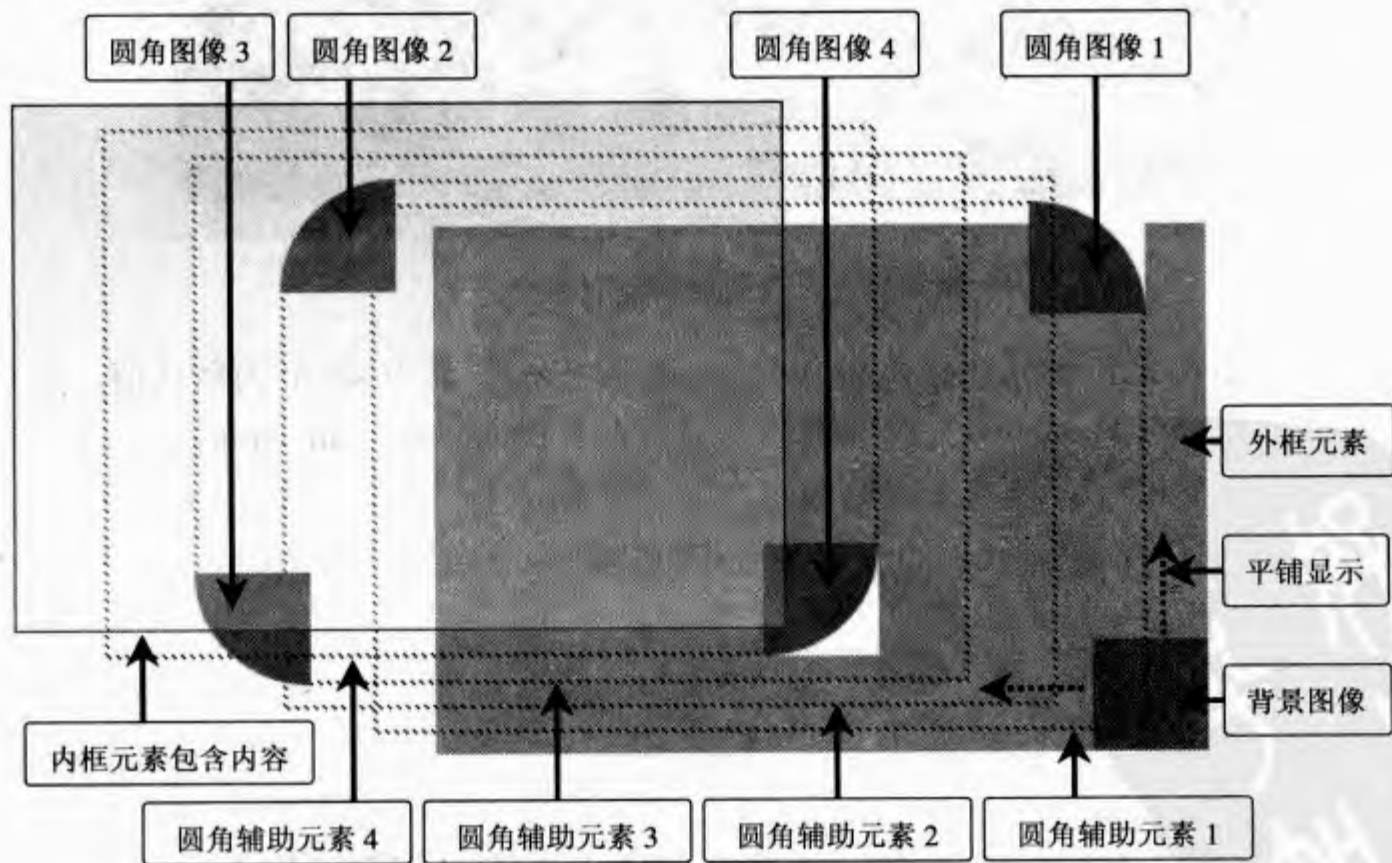


图 4.56 圆角区域设计示意图

从图 4.56 所示的圆角设计结构示意图可以看到, 要实现这样的圆角区域需要嵌套 5 层元素,

包括四个定义圆角的辅助元素和一个包含内容的内框元素。具体结构代码如下：

```
<div class="roundbox">          <!-- 圆角区域的外框 -->
  <div class="top">              <!-- 圆角图像辅助元素 1 -->
    <div></div>                  <!-- 圆角图像辅助元素 2 -->
  </div>
  <div class="content">          <!-- 圆角区域的内框 -->
    <!-- 圆角区域显示内容 -->
  </div>
  <div class="bot">              <!-- 圆角图像辅助元素 3 -->
    <div></div>                  <!-- 圆角图像辅助元素 4 -->
  </div>
</div>
```

最外层元素作为圆角区域的控制柄，即调整圆角区域的大小，并定义背景色。然后其中嵌套4个元素，设置每个元素包含一个圆角背景图像。最内层的元素专门用来包含显示内容。

根据元素嵌套时，最内层元素的背景图像会覆盖外层元素的背景图像，这样内层背景圆角图像的白色区域就会把最外层背景图像的四个角给覆盖掉了，从而显示为圆角区域效果。

着手设计样式之前，你应该准备四个顶角的圆角图像或者背景图像（如图4.57所示）。图像大小为7像素，圆角度为5像素。在用图像编辑软件设计时应注意三个问题：第一，这些图像必须是正方形；第二，前景色与背景色应事先考虑并设计好，否则要修改圆角区域的颜色时就会很麻烦；第三，圆角度越大，圆角区域看起来就更圆。



图4.57 需要准备的圆角背景图像

根据图4.56所示的示意图设计CSS样式表代码如下，所显示的效果如图4.58所示。

```
<STYLE type=text/css media=all>
.roundbox {          /* 定义圆角区域外框背景图像，也可以定义为背景色会更方便 */
  background: url(images/roundbox/nt.gif) repeat;
}
.top div {           /* 定义左上角圆角背景图像 */
  background: url(images/roundbox/tl.gif) no-repeat top left;
}
.top {               /* 定义右上角圆角背景图像 */
  background: url(images/roundbox/tr.gif) no-repeat top right;
}
.bot div {           /* 定义左下角圆角背景图像 */
  background: url(images/roundbox/bl.gif) no-repeat bottom left;
}
.bot {               /* 定义右下角圆角背景图像 */
  background: url(images/roundbox/br.gif) no-repeat bottom right;
}
.top div, .top, .rbot div, .bot { /* 定义4个顶角圆角背景图像对应元素的宽和高 */
  width: 100%;        /* 与外框同宽 */
  height: 7px;        /* 与背景图像同高 */
  font-size: 1px;     /* 清除元素内包含行高的影响 */
}
.content { margin: 0 7px; }      /* 为内层元素设置7像素左右边距，与背景图像同宽 */
.roundbox {
  width: 90%;              /* 控制圆角区域的宽度，高度自适应内部包含元素 */
  margin: 1em auto;        /* 居中显示 */
}
</STYLE>
```

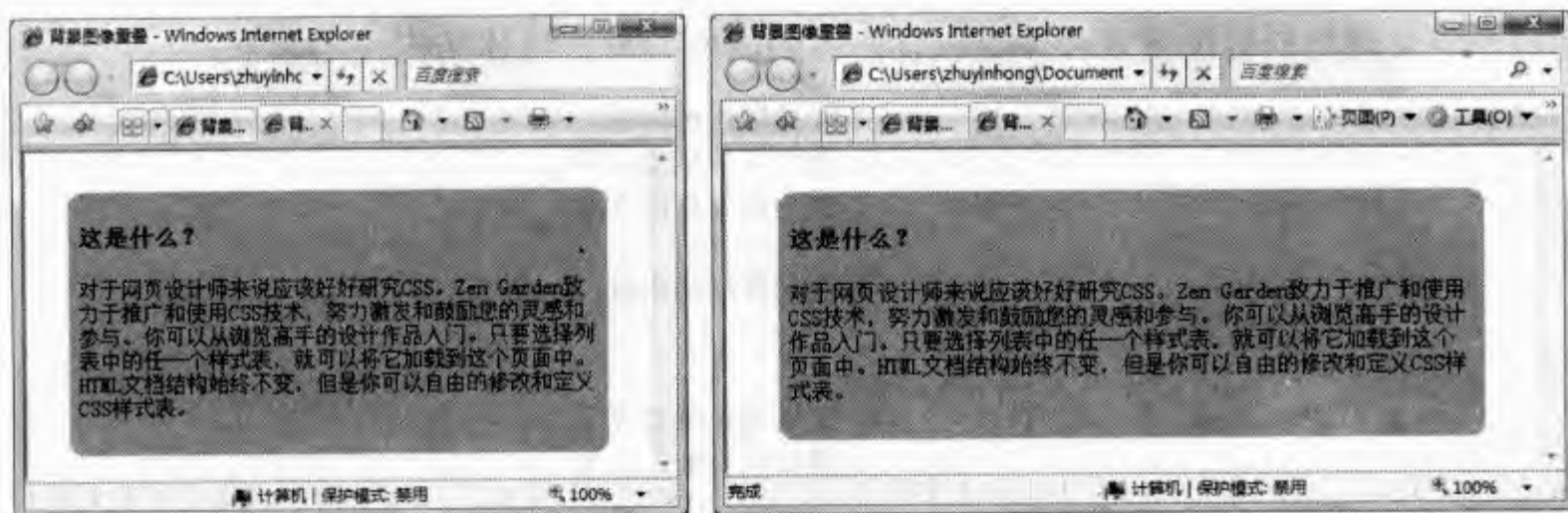


图 4.58 圆角区域自适用不同区域效果

提示

为方便用户快速设计，CSS 对于每类属性都提供了复合属性，例如，对于字体的有关属性都可以在 `font` 中简单、快速的设置，而对于 `boder` 属性可以设置盒模型包含的所有属性。其实对于背景图像来说，所有属性多可以在 `banckground` 中进行声明，各种不同属性值可以并列排在一起，没有先后顺序之分，可以用空格分隔。

另外一个问题，背景图像是无法利用 CSS 控制大小的，如果希望改变背景图像的大小，只有通过事先在图像编辑器中调整。

4.4.7 设计更酷的圆角

经过上一节的详细讲解，你应该明白圆角设计的基本思路，本节将在此基础上继续深入，研究更复杂圆角的设计技巧。通过这样的专题练习，相信对于提高你驾驭 CSS 的能力会有很大的帮助，同时也能够受此启发设计出更具个性的圆角效果。本实例的设计效果如图 4.59 所示。

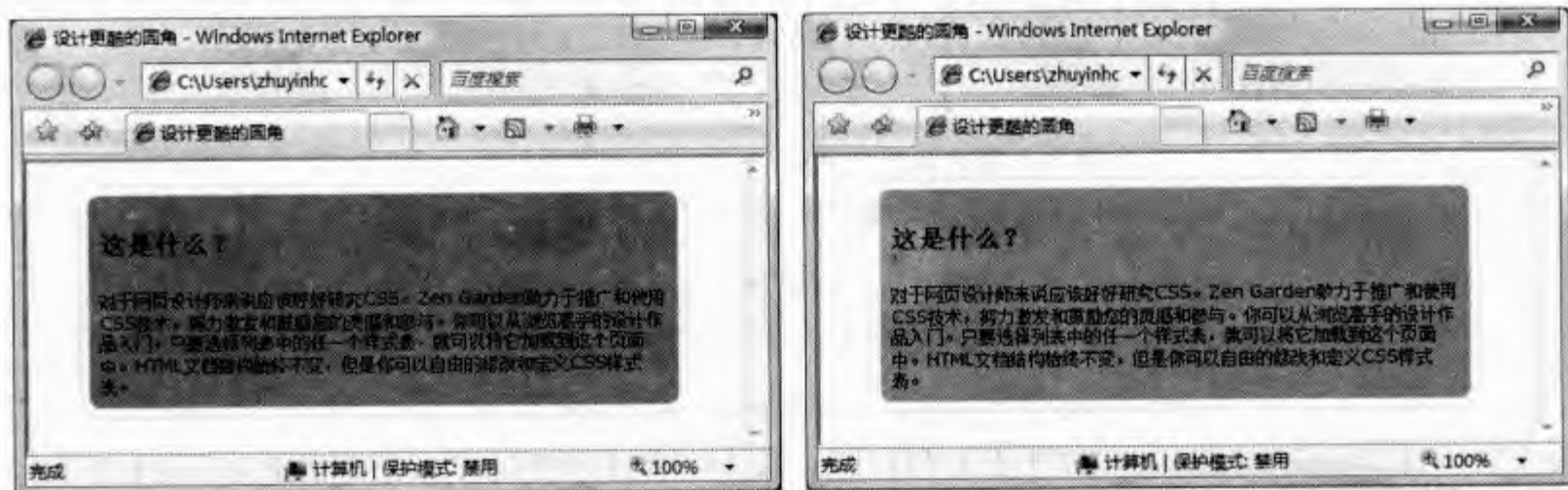


图 4.59 更酷圆角设计效果

首先设计思路分析。你还记得在上一节中讲解基本圆角的设计思路吗？它是利用 4 个圆角通过叠加的方法覆盖在某个元素的四个顶角，以此伪造圆角区域的视觉效果。本实例需要用到 8 个背景图像，使用它们分别模拟圆角边框的 4 个顶角和 4 条边。最后通过 CSS 把它们分别固定到元素的边框和顶角上，设计示意图如图 4.60 所示。

其次设计背景图像。虽然本实例所用背景图像共有 8 个，其实你只需要设计 2 个即可，其他图像只需要在此基础上简单旋转即可。设计中要注意 2 个问题：

第一，为了能够使背景图像具有更好的兼容性，建议设计 4 个顶角要设计为正方形，大小可以根据需要自定义。

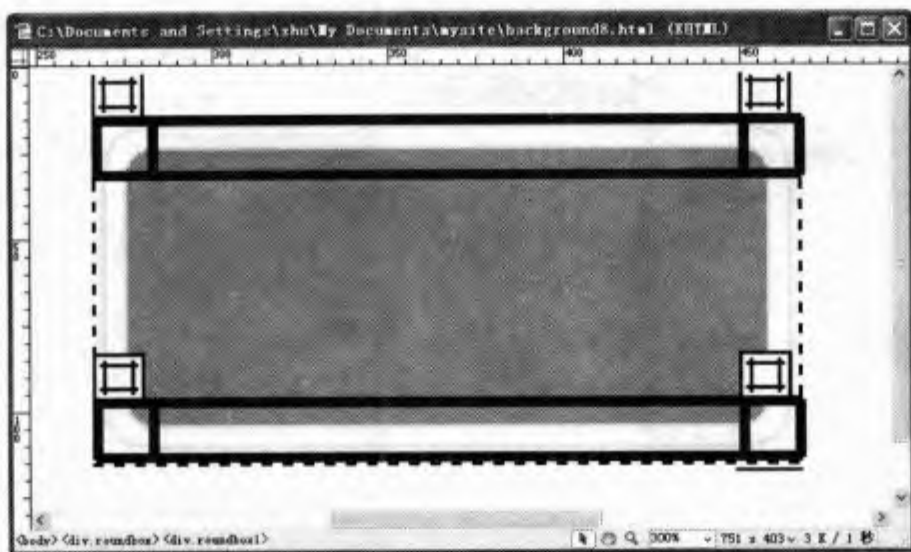


图 4.60 圆角设计示意图

第二，圆角的宽度或高度必须与水平或垂直边上的背景图像保持一致，所设计的图案应该连贯，保证无缝衔接。

启动 Photoshop CS3，新建一个 16px×16px 大小的文档。我们设计一个浅灰色的立体边框（如图 4.61 所示），为了便于操作，建议放大图像，以像素格的方式进行颜色填充，这样能够保证所设计的圆角更精确，没有毛刺或者缺陷。在设计时要时刻保持上下、左右对称。

在设计背景图像时，考虑到以后扩展需要，把整个顶角分成 3 个图层：

第一个图层是左上角黑色区域的“背景色”，在这个图层中可以定义圆角外部的背景色，本示例显示为白色。如果网页背景色为其他颜色，你只要在这里修改即可。

第二个图层是“圆角框”，在这里你可以设计圆角边框为不同的样式。本示例设计一个圆滑的灰色边框，内边设置了一条浅灰色的修饰线（参见“内边修饰线”图层），这样使边框看起来更具立体感。

第三个图层是“圆角填充色”层，在这里你可以根据圆角区域所定义的背景色来进行设置。当然我们也可以把它设置为透明色，保存 GIF 格式后，它能够透明显示圆角区域内的背景色，这样就不再回到 Photoshop 中进行调整了，避免麻烦。

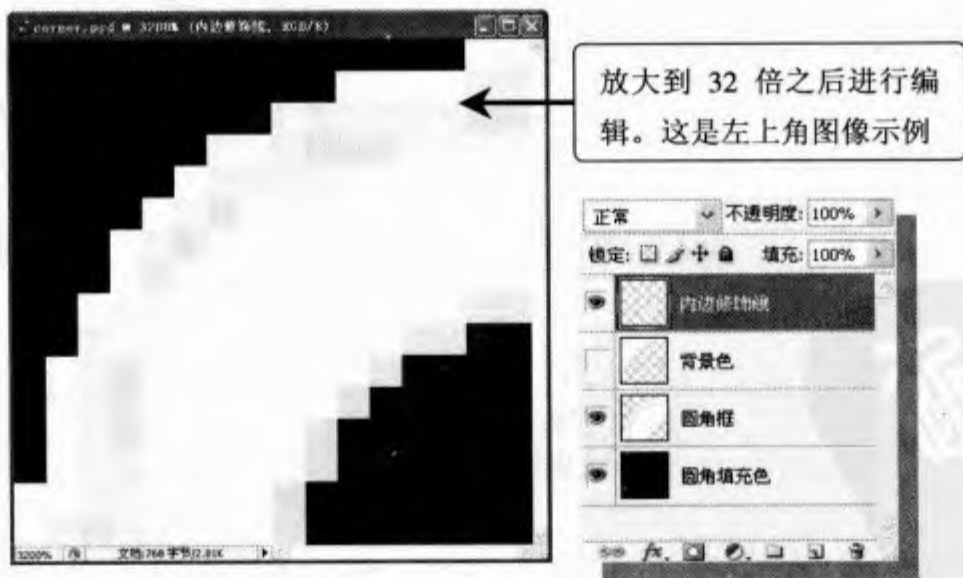


图 4.61 圆角设计图

设计好顶角模板之后，保存并备份 PSD 格式文当，然后合并图层，分别保存为四个顶角图像，保存时要随时旋转图像，以便与对应顶角位置相对应。

再去完成顶角的设计，4 条边框就比较好做了。你可以在顶角模板文当基础上裁切即可，要保证其高度或宽度与顶角图像相同（如图 4.62 所示）。



图 4.62 由圆角设计边框图

然后保存并备份 PSD 格式文当，合并图层，再分别保存为四条边框图像，保存时要随时旋转图像，以便与对应边框的位置相对应。

最后搭建圆角框架。如果要定位 8 个背景图像，就必须在一个元素包含 7 个元素，并在该自身内定义背景图像，所设计的 HTML 结构如下：

```
<div class="roundbox">                                <!-- 圆角区域的外框，在此定义右边框背景图像 -->
  <div class="roundbox1">                                <!-- 圆角区域的外框 1，在此定义左边框背景图像 -->
    <div class="top">                                    <!-- 圆角顶部边框，在此定义顶部边框背景图像 -->
      <div class="top_left"></div><!-- 定义左上顶角背景图像 -->
      <div class="top_right"></div><!-- 定义右上顶角背景图像 -->
    </div>
    <div class="content">                                <!-- 圆角区域的内框 -->
      <!-- 圆角区域显示内容 -->
    </div>
    <div class="bot">                                    <!-- 圆角底部边框，在此定义底部边框背景图像 -->
      <div class="bot_left"></div><!-- 定义左下顶角背景图像 -->
      <div class="bot_right"></div><!-- 定义右下顶角背景图像 -->
    </div>
  </div>
</div>
```

搭建好 HTML 结构之后，下面设计重心就是来定义样式表了，详细说明如下。

首先，清除所有结构标签的边距：

```
.roundbox, .roundbox1, .top, .top_left, .top_right, .bot, .bot_left, .bot_right {
  margin: 0px;      /* 清除边距 */
  padding: 0px;     /* 清除边距 */
}
```

然后定义外框样式，以及圆角在页面内显示的位置、大小和背景色。请注意，在本实例中网页背景色只能为白色，如果定义其他颜色，还需要调整 4 个顶角背景图像颜色。

```
body { /* 网页属性，可以不定义 */
  text-align: center; /* 网页居中 */
  font-family: Verdana, Arial, Helvetica, sans-serif; /* 网页居中 */
  font-size: 12px; /* 网页字体大小 */
}
.roundbox { /* 外框 1 样式 */
```



```

width:200px;                /* 定义圆角区域的宽度,可自定义 */
margin:0px auto;            /* 圆角区域居中显示,可自定义 */
background-image: url(images/roundbox1/right.gif); /* 右边框背景图像,只读 */
background-repeat: repeat-y; /* 纵向平铺右边框背景图像,只读 */
background-position: right 0px; /* 定位右边框背景图像靠右显示,只读 */
background-color:#66CC33;    /* 定义圆角区域内的背景色,可自定义 */
text-align:left;            /* 定义圆角区域内文本左对齐,可自定义 */
}
.roundbox1 { /* 外框2样式 */
width:100%;                /* 与父元素同宽,只读 */
background-image: url(images/roundbox1/left.gif); /* 左边框背景图像,只读 */
background-repeat: repeat-y; /* 纵向平铺左边框背景图像,只读 */
background-position: 0px 0px; /* 定位左边框背景图像靠左显示,只读 */
}

```

定义上下两条边的背景图像。请注意,以下所有样式都不能够随意改动。

```

.top, .bot { /* 上下边元素公共样式 */
position:relative;        /* 相对定位,为顶角元素定位提供包含块 */
height:16px;              /* 固定高度,与背景图像等高 */
}
.top { /* 定义顶边框背景图像,靠顶部水平平铺 */
background: url(images/roundbox1/top.gif) repeat-x left 0px;
}
.bot { /* 定义底边框背景图像,靠底部水平平铺 */
background: url(images/roundbox1/bottom.gif) repeat-x; left bottom;
}

```

定义四个顶角元素的样式。请注意,以下所有样式都不能够随意改动。

```

.top_left, .top_right, .bot_left, .bot_right { /* 4个顶角元素公共样式 */
width:16px;                /* 固定宽度,与顶角背景图像同宽 */
height:16px;              /* 固定高度,与顶角背景图像同高 */
position:absolute;        /* 绝对定位,以便精确定位 */
background-repeat: no-repeat; /* 禁止背景图像平铺 */
}
.top_left { /* 左上顶角元素 */
top:0px;                  /* 对齐顶部 */
left:0px;                 /* 对齐左边 */
background: url(images/roundbox1/tl.gif) left 0px; /* 固定左上角背景图像 */
}
.top_right {
top:0px;                  /* 对齐顶部 */
right:0px;                /* 对齐右边 */
background: url(images/roundbox1/tr.gif) right 0px; /* 固定右上角背景图像 */
}
.bot_left {
bottom:0px;               /* 对齐底部 */
left:0px;                 /* 对齐左边 */
background: url(images/roundbox1/bl.gif) left bottom; /* 固定左下角背景图像 */
}
.bot_right {
bottom:0px;               /* 对齐底部 */
right:0px;                /* 对齐右边 */
background: url(images/roundbox1/br.gif) right bottom; /* 固定右下角背景图像 */
}

```

最后可以根据需要定义内容框的样式。例如:

```
.content {  
    margin:0px;  
    padding:0 16px;  
}
```

4.5 背景图像布局应用——伪列布局

如果借助 Dreamweaver CS3 所提供的 CSS 标准布局模板新建一个“3 列固定，标题和脚注”的文档，你会发现其左右列的高度不能够自适应栏目的高度而伸张，仅随内容的多少而被动撑开，这样就不可避免的会出现页面各列布局对不齐的尴尬效果（如图 4.63 所示）。



图 4.63 多列布局中不能对齐的效果

要解决这个问题，我们不妨借助背景图像来实现。在动手之前先看看本节所要设计实例的效果（如图 4.64 所示）。



图 4.64 多列布局中自动对齐的效果

在这个固定宽度的三行三列的页面布局中，你将会看到不管哪列伸展多长，其他列也能够自动适用这种变化。实现这样的设计效果，其设计核心是利用背景图像通过纵向平铺来伪装各列被自动伸缩的效果，实际上每列依然保持自身布局效果。我们所看到的效果实际上是一个被设计为3列布局效果的背景图像在纵轴平铺所显示的效果。具体实施步骤如下：

首先规划页面结构。动手设计之前，建议读者先在心里打个草稿。所设计的是一个三行三列固定宽度的页面。由于伪列布局中背景图像的宽度是无法随窗口自适应变化的，所以对于此类布局必须固定其页面宽度，只有事先固定了页面宽度，你才能够量体裁衣，根据这个宽度在 Photoshop 中设计背景图像，本实例宽度采用 980 像素。同时要考虑好整个页面的色调，以及配色方案，只有做到心中有数，动起手来才会镇定，本实例以深绿色为基本色调，并搭配浅绿色进行布局。

启动 Photoshop，新建一个文档，要设置好图像的宽度（980 像素），利用渐变工具绘制一个三列布局效果的图像（如图 4.65 所示）。具体渐变样式根据个人喜好而定，要注意每列填色效果，以方便区分不同列。

在 Photoshop 中把图像裁切为宽度不变，而高度仅为 1 像素或者其他像素的 GIF 图像。随便再制作一个网页背景图像（如图 4.66 所示），高度不低于一个屏幕高度，宽度可以任意，最后也裁切并保存为一个宽度为 1 像素，高度不变的 GIF 图像。

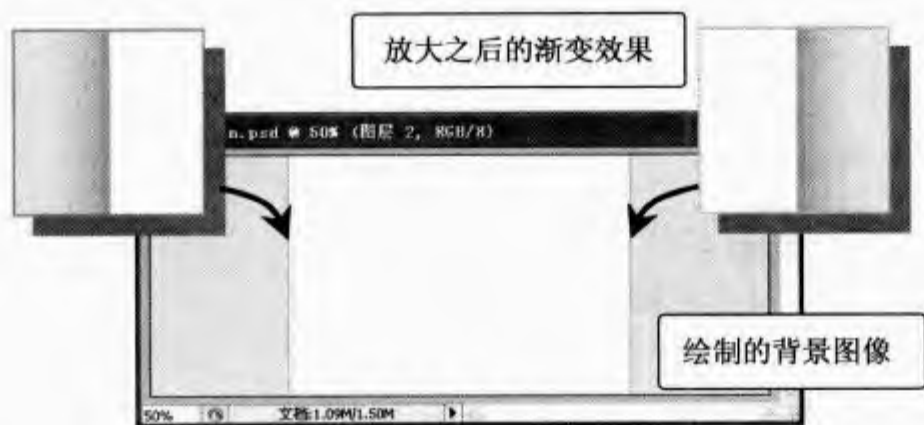


图 4.65 设计多列布局背景图像



图 4.66 设计网页图像

准备工作很繁琐，但是设计的图像质量和水平决定了最终网页效果，所以大意不得。下面就可以在 Dreamweaver CS3 中搭建网页结构了：

```
<div id="container">
  <div id="header">头部区域 </div>
  <div id="main">
    <div id="content-m"> 中间栏目</div>
    <div id="sidebars">
      <div id="sidebar-a">左侧栏目</div>
      <div id="sidebar-b">右侧栏目 </div>
    </div>
  </div>
</div>
<div id="ft">版权区域</div>
```

在前面章节中，我们也曾经设计过三行三列布局的结构，当然布局方法略有不同，本结构中第一行头部和第二行的主题结构被包含在一个结构标签之中，而左右栏目也被包含在一起。这样设计的目的是更方便的管理。不同结构最终实现的效果都是一样，因此也就没有孰优孰劣之说。

在样式表中为网页定义背景图像和背景颜色，然后使用<link>标签把样式表引入到网页中。

```
body {  
    background:#557100 url(bg.gif) repeat-x 0 0; /* 网页背景 */  
    font-family:Arial, Helvetica, sans-serif; /* 字体属性 */  
    font-size:12px; /* 网页字体大小 */  
    margin:0; /* 清除页边距 */  
    padding:0; /* 清除页边距 */  
}
```

定义网页基本结构的样式。为网页第2行的主体结构定义伪列背景图像。固定每行宽度为980像素，如图4.67所示。

```
#header {  
    width:980px; /* 固定宽度 */  
    margin:auto; /* 居中显示 */  
}  
#main {  
    width:980px; /* 固定宽度 */  
    margin:auto; /* 居中显示 */  
    background:url(bg-main.gif) repeat-y 0 0; /* 定义伪列布局的背景图像，并水平平铺 */  
    clear:both; /* 清除左右存在的浮动元素 */  
}  
#ft {  
    width:980px; /* 固定宽度 */  
    margin:auto; /* 居中显示 */  
    height:100px; /* 固定高度 */  
    clear:both; /* 清除左右存在的浮动元素 */  
}
```

同时定义左右两侧栏目分别向左和向右浮动，使它们分别停靠在页面的两侧，同时定义中间栏目向左浮动（效果如图4.68所示），CSS样式代码如下：

```
#sidebar-a { /* 左侧栏目 */  
    width:180px; /* 固定宽度 */  
    padding:10px 20px; /* 增加栏目内边距 */  
    float:left; /* 向左浮动 */  
    margin-left:-760px; /* 以负边距的方式显示在左侧 */  
}  
#sidebar-b { /* 右侧栏目 */  
    width:180px; /* 固定宽度 */  
    padding:10px 20px; /* 增加栏目内边距 */  
    float:right; /* 增加栏目内边距 */  
}  
#content-m { /* 中间栏目 */  
    width:500px; /* 固定栏目宽度 */  
    padding:10px 20px; /* 增加栏目内边距 */  
    float:left; /* 向右浮动 */  
    margin-left:220px; /* 增加左边距 */  
    display:inline; /* 行内显示，避免错位 */  
}
```

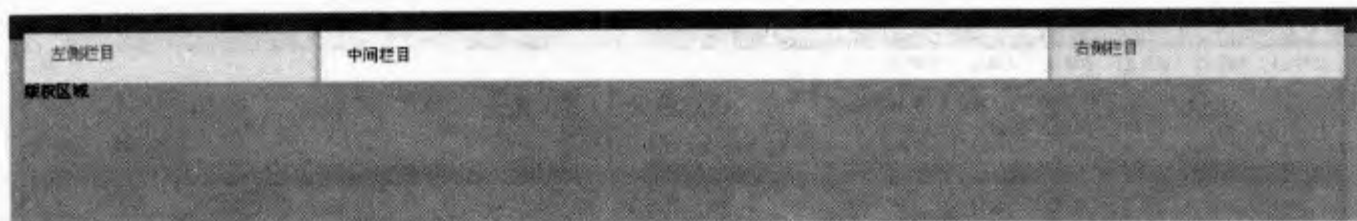



图 4.67 页面基本结构效果

这里面有一个技术难点，就是对于多个元素并列浮动显示来说，最大的难题就是错位问题。因为浮动布局本身就是不稳定的，多个浮动元素并列在一起，当边距或者窗口宽度发生了变化，都很容易导致栏目错位现象。针对上面的栏目结构，正常浮动显示应该如图 4.68 所示。也就是说根据元素的先后位置进行浮动排列，由于中间栏目排在左侧栏目前面，因此它应该显示在最左边。

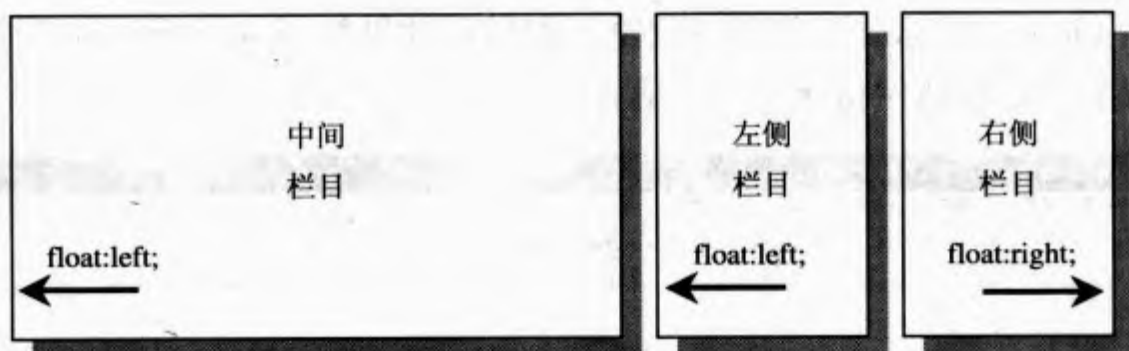


图 4.68 基本栏目正常浮动效果

由于 CSS 规定 `margin` 属性可以取负值，即可以利用 `margin` 属性反向移动元素的位置，这与通过取正值增加边距正好相反。那么我们不妨利用这个设计原理，让左侧栏目与中间栏目互换一下位置，如图 4.69 所示。这样错位的目的就是防止浮动元素随意移动，因为错位显示之后，它们之间就不会相互影响了。

同时设置中间栏目以行内元素显示，这样就能够保证该模块拥有文本显示的属性，不会随意错位显示。

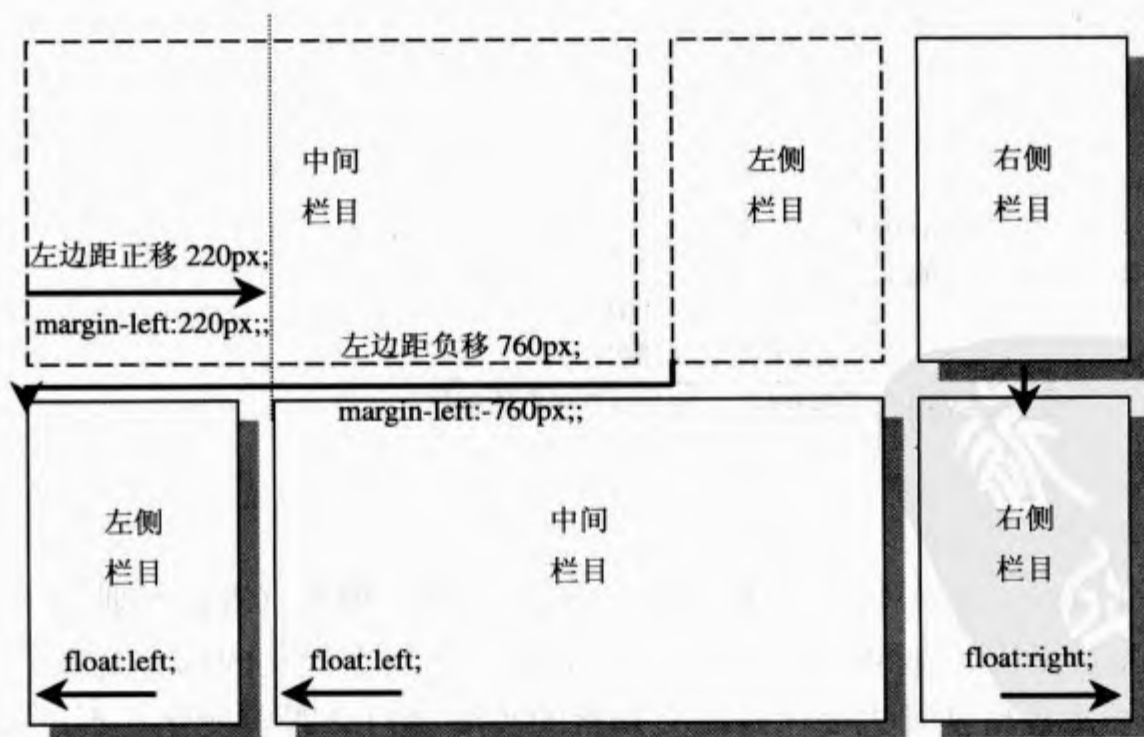


图 4.69 错位浮动元素示意图

此时，如果在左栏、中栏或右栏中输入换行符，则在 IE 浏览器中显示其他两个栏目也跟着伸长（如图 4.70 所示）。

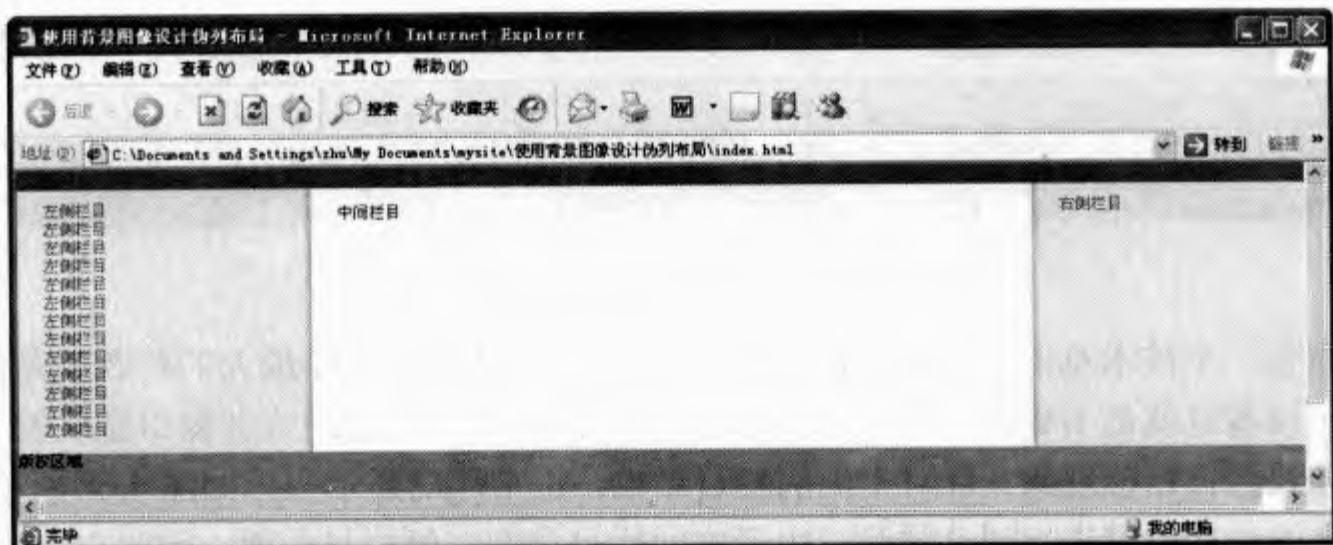


图 4.70 在 IE 6 浏览器中的显示效果

但是如果在 Firefox 等标准浏览器中预览，则显示效果如图 4.71 所示。当左侧栏目被撑开之后，中间和右侧栏目依然保持原来的位置不动。

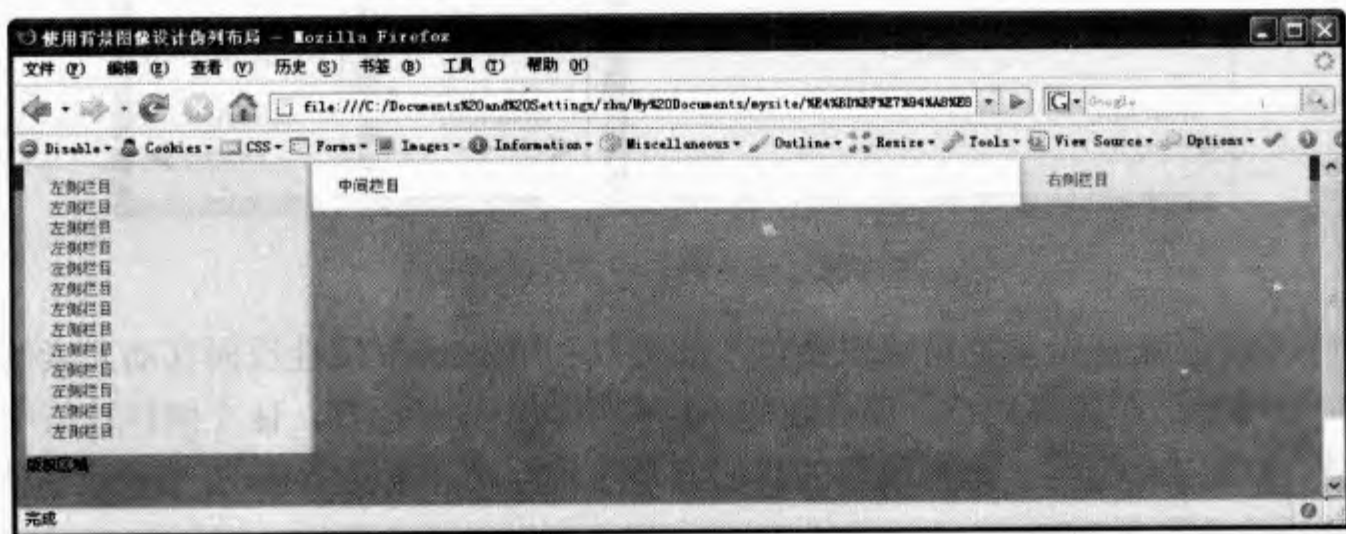


图 4.71 在 IE 6 浏览器中的显示效果

看来不同浏览器对此解析的方式还存在很大差异。解决的方法是在第二行结构的末尾增加一个浮动清除标记，代码如下：

```
<div id="container">
  <div id="header"> </div>
  <div id="main">
    <div id="content-m"> 中间栏目</div>
    <div id="sidebars">
      <div id="sidebar-a">左侧栏目</div>
      <div id="sidebar-b">右侧栏目 </div>
      <div style="clear: both;"></div>
    </div>
  </div>
</div>
```

原来标准浏览器在解析每个元素的显示样式是严格根据 CSS 规则，由于父级标签<div id="main">没有定义高度，标准浏览器会以 0 高度显示，不受其内部子元素的高度影响，而对于 IE 来说，当子元素高度发生变化时，父元素的高度会自动跟着变化。不过 clear: both; 声明能够强迫父元素自动撑开，以适应子元素的高度。

虽然这样能够模拟出多列对齐显示问题，但是整个版面显得比较生硬。为此我们可以借助 Photoshop 编辑工具设计一个与伪列背景图像可以过渡显示的图像。

首先为版权信息栏设计一个背景图像（如图 4.72 所示）。设计时建议在 Photoshop 编辑窗口中打开伪列背景图像，或者就在伪列背景图像的基础上进行再编辑最后另存为版权信息栏背景图像，这样能够保证上下行背景图像准确衔接。



图 4.72 版权信息栏背景图像

然后为版权区域定义背景图像，并隐藏显示的文本。为了防止版权信息和主体区域之间出现各种错位等浏览器兼容性问题，建议在它们之间增加一个清除标签 `<div style="clear: both;"></div>`，此时页面显示效果如图 4.73 所示。

```
#ft {
    text-indent:-9999px;          /* 隐藏版权区内文本 */
    background:url(ft.gif) no-repeat 0 0; /* 定义背景图像 */
}
```

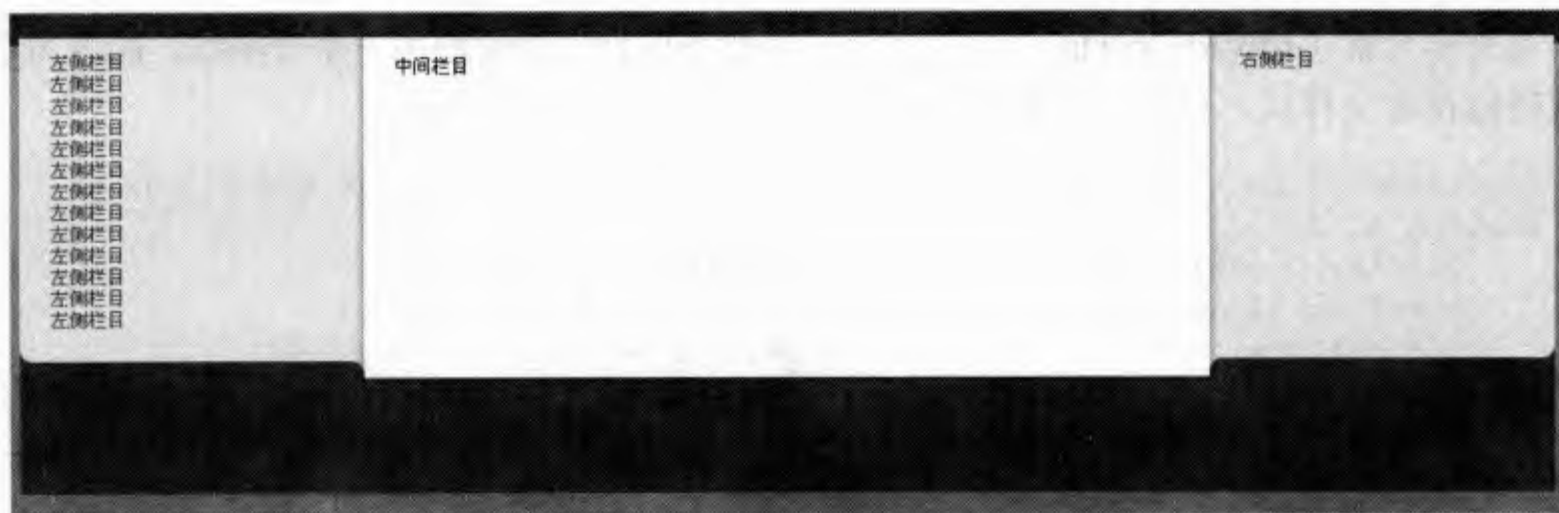


图 4.73 连接版权信息栏背景页面效果

回到网页结构的顶部，先在 `<div id="header">` 标签内（即头部区域）增设二级网页结构。假设头部区域要设置如下的框架结构（当然你可以自由的扩展结构）：

```
<div id="header">
    <h1><a href="#">LOGO 标识</a></h1>          <!-- 网站 Logo -->
    <div id="tagline">伪列布局</div>              <!-- 网站 Banner -->
    <ul id="nav">                                   <!-- 网站导航 -->
        <li class="menu1"><a href="#">菜单 1</a></li>
        <li class="menu2"><a href="#">菜单 2</a></li>
        <li class="menu3"><a href="#">菜单 3</a></li>
        <li class="menu4"><a href="#">菜单 4</a></li>
    </ul>
    <div id="search"></div>                          <!-- 网站搜索 -->
</div>
```

针对上面的结构，下面分别为它们定义样式。定义 Logo 以浮动显示在头部区域的左侧，代码如下：

```
#header h1 { /* Logo 标题标签样式 */
    margin:0;          /* 清除标题标签的默认边距 */
    float:left;        /* 向左浮动 */
    width:180px;       /* 固定宽度 */
    padding:25px 20px 0; /* 调整内边距，顶部边距为 25 像素，左右为 20 像素 */
}
```

```

height:119px;          /* 固定高度 */
background:url(leftcap.gif) no-repeat 0 100%; /* 在底部增加一个背景图像*/
}

```

此时页面显示效果如图 4.74 所示。为方便读者观察，我们临时为该标题元素定义一个边框。



图 4.74 增加的标题信息

为外框元素（<div id="container">）定义一个背景图像，以便增加一个分隔线。同时为页面标题超链接定义样式，以便应用背景图像，而隐藏标题文本的显示：

```

#container { background:transparent url(bg-top.gif) no-repeat 50% 21px}
#header h1 a {
    display:block;          /* 强制超链接块状显示 */
    overflow:hidden;        /* 隐藏超出区域 */
    width:104px;            /* 固定宽度 */
    height:39px;            /* 固定高度 */
    background:url(logo.gif) no-repeat 0 0; /* 以背景形式插入 Logo 图标 */
    text-indent:-9999px;     /* 隐藏文本 */
    position:relative;      /* 相对定位 */
    top:30px;               /* 移动 Logo 图标的位置 */
    left:30px;              /* 移动 Logo 图标的位置 */
}

```

再定义 Banner 模块的大小和背景图像，使其向左浮动，以便与左侧的 Logo 区域并列显示。

```

#tagline {
    width:540px;            /* 固定宽度 */
    height:144px;           /* 固定高度 */
    background:url(tagline.gif) no-repeat 0 0; /* 背景图像 */
    text-indent:-9999px;    /* 隐藏文本 */
    float:left;             /* 向左浮动 */
}

```

定义导航条，要注意列表项是如何并列显示的。具体代码如下：

```

#nav { /* 控制导航条布局 */
    float:left;            /* 向左浮动 */
    background:url(nav.gif) no-repeat 0 0; /* 增加背景图像 */
    width:195px;           /* 固定宽度 */
    height:33px;           /* 固定高度 */
    margin:21px 0 0 0;     /* 增加外边距 */
    padding:14px 0 0 15px; /* 增加内边距 */
    list-style:none;       /* 隐藏项目符号 */
}
#nav li { float:left; } /* 固定宽度 */

```



```

#nav a {/* 导航超链接样式 */
    float:left;                /* 浮动显示超链接 */
    display:block;             /* 块状显示 */
    text-indent:-999px;        /* 隐藏菜单文本，用背景图像中的文字来显示 */
    overflow:hidden;           /* 隐藏多余的区域 */
}

```

最后再定义搜索区域的样式，代码如下，最后所得头部区域设计效果如图 4.75 所示，为了方便观察，临时给头部区域定义一个边框。

```

#search {
    float:left;                /* 向左浮动 */
    width:220px;               /* 固定宽度 */
    height:76px;               /* 固定高度 */
    background:url(bg-search.gif) no-repeat 0 0; /* 增加背景 */
}

```

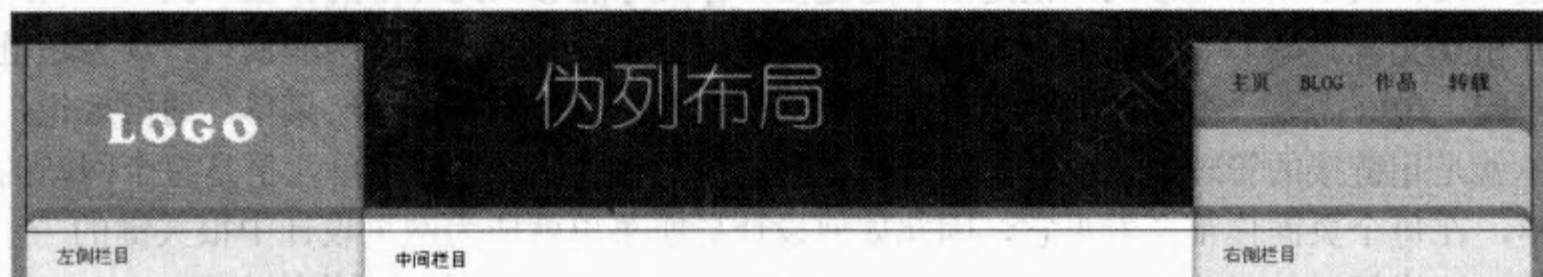


图 4.75 头部区域设计效果

第 5 章

活泼的超链接和导航菜单

超链接构筑了整个互联网，正是由于超链接，你我才能够如此轻松的穿梭于网页、网站之间，并相互交流。毫不夸张地说，如果没有超链接，也就没有今天互联网的繁荣。众多超链接挤在一起，最好需要一个有效的组织和管理方法或途径，于是就有了导航菜单。通俗地说，导航菜单就是超链接的管理器，或者说就是众多超链接的包含块。导航菜单几乎是整个网站的控制中枢，在每个页面你都会看见它，因此如何设计导航菜单就成为网页设计中很关键的一步。本章将详细讲解以伪类为核心的超链接，以及以超链接为基础的导航菜单的应用。

5.1 轻松掌握超链接样式

超链接通过 `a` 元素来定义，它与其他网页元素一样即普通又常用，但是它又与其他元素存在很大的不同，例如，一般元素都是静态的，而超链接中包含多种状态，这为控制超链接带来一定的难度，对于初学者来说，可能会感觉超链接的样式是那么难以控制。如果你也同感，那么请不要焦虑，这是普遍现象。

5.1.1 怪异的伪类

你可能经常在网页中插入 JPEG 或 GIF 格式的图片，但是对于 GIF 图片来说，它可能是包含多张序列图片的动画，即 GIF 动画。当简单控制图片的大小、位置等显示状态时，操作 JPEG 格式图片与操作 GIF 格式图片没有什么区别，既简单又直观。

但是如果让你控制 GIF 动画中所包含的每张图片，你会感觉很困难，当然借助 GIF 动画编辑工具可能就很简单了，但是对于网页设计来说，其控制难度是非常大的。

实际上超链接所包含的不同显示状态与 GIF 动画所包含的多张序列图片有点类似，可以想象在网页中控制超链接也是很困难的。在 CSS 没有诞生之前，设计师通过在 `<body>` 标签中定义属性来简单的进行控制。例如：

```
<body link="#FF0000" alink="#00FF00" vlink="#0000FF" >
  <a href="#">传统超连接显示控制方法</a>
</body>
```

在上面的代码中，网页设计师通过为 `body` 元素定义三个属性，其中 `link` 属性表示超链接正常状态下的显示颜色，`alink` 属性表示超链接被激活时显示的颜色，而 `vlink` 属性表示超链接被单击后显示的颜色，也就是说访问超链接之后显示的颜色。

使用这种方法比较简单且生硬，原因是：

第一，这些 `body` 元素的属性只能够定义超链接的显示颜色，而无法控制其他样式。

第二，就是整个网页的超链接只能够被定义为一种显示样式，无法为不同的超链接定义不同的显示样式。

第三，所提供的超链接状态仅有三种，稍显单薄。

在 CSS 中，你可以为 `a` 元素定义不同的显示样式，但是你无法直接通过 `a` 元素来单独控制超链接不同状态的样式。为此 CSS 提出了伪类的概念。那么什么是伪类呢？

通俗说伪类就是基于元素某种特征或状态进行定义，而不是针对元素名称、属性或者内容。例如，为超链接定义被单击时的样式，则它针对的是某种特殊的状态，而不是 `a` 元素本身，或者 `<a>` 标签包含的文本图片，或者 `<a>` 标签的 `href` 属性等。原则上这种特征不是从 HTML 标签及其属性所能够直观的推断得到的。

伪类可以是动态的，当用户与文档进行交互的时候，一个元素可以获取或者失去一个伪类，因为伪类应用的对象是不确定的。除了 `:first-child` 伪类可以通过 HTML 标签推断伪类所要作用的对象，`:lang` 在某些情况下也可以被推断出来。

你可以把伪类看作是一种特殊的类选择符，是被支持 CSS 的浏览器自动识别的特殊选择符。它的最大的用处就是可以对超链接在不同状态下定义不同的样式效果。

CSS 2.0 版本提供了 7 种伪类：`:first-child`、`:link`、`:visited`、`:hover`、`:active`、`:focus` 和 `:lang`。其中能够被不同浏览器支持，且使用比较广泛的是 `:link`、`:visited`、`:hover` 和 `:active`。

伪类的用法如下：

```
selector:pseudo-class{
    property: value;
}
```

直观表示就是：

```
选择符:伪类{
    属性: 值;
}
```

伪类与类进行比较，它还有一点不同，就是伪类是 CSS 已经定义好的，不能像类选择符一样能让你可以随意用别的名字，根据上面的语法可以解释为对象（选择符）在某个特殊状态下（伪类）的样式。

类选择符及其他选择符也可以与伪类混用，例如：

```
selector.class:pseudo-class{
    property: value;
}
```

直观表示就是：

```
选择符.类:伪类{
    属性: 值;
}
```

5.1.2 超链接的四种状态

从伪类的晦涩难解中走出来，我们还是看看本章所要研究的正题：超链接样式。说到超链接，CSS 把它定义了四种显示状态，具体说就是：

link: 未访问时超链接的样式。
 visited: 已访问时超链接的样式。
 hover: 鼠标停留在超链接上时显示的样式。
 active: 激活超链接时显示的样式。

请注意它们的排列顺序, 在某些浏览器中, 如果它们的顺序被颠倒了, 所定义的超链接样式可能会无法被解析和显示。

有的读者可能会迷惑了: 为什么 CSS 在这儿要较真儿呢? 一般来说 CSS 同一样式内的声明可以随意排列。原来这都是 CSS 层叠基本特性决定的。当所有样式都作用于同一个对象时, 如果它们的特性相同, 那么排列在最后的声明将会被解析和显示。例如, 如果 link 样式位于最后, 则当浏览器解析完超链接不同状态的样式时, 最后将被 link 样式所覆盖, 其他的状态样式也就被覆盖。而按照超链接正常执行的状态顺序, 这种情况就不会发生。

定义超链接的方法如下:

```
<style type="text/css">
a:link { /* 未访问时的样式 */
    color: #000;           /* 黑色 */
}
a:visited { /* 已访问时的样式 */
    color: #666;           /* 灰色 */
}
a:hover { /* 经过时的样式 */
    color: #f00;           /* 红色 */
}
a:active { /* 激活时的样式 */
    color: #00f;           /* 蓝色 */
}
</style>
```

5.1.3 为超链接绑定不同的样式

a 元素在默认状态下 (未被单击时) 显示为蓝色、下划线的样式, 已访问的超链接则显示为紫色。这是超链接的经典配色, 但是看惯这种超链接样式之后, 相信你会有一种摆脱这种蓝紫搭配的冲动, 实际上在网页中也几乎很少再看到这种“原生态”的超链接样式。看来习惯于大脑中的蓝色下划线再也不是超链接的代名词了。

当然, 我们可以为不同的超链接定义不同的样式。例如, 在下面的示例代码中我们为同一个页面中不同的超链接定义不同的样式:

```
<style type="text/css">
a { text-decoration:none; color: #333; } /* 超链接基本显示样式 */
.class1:hover {color:red;} /* 定义 class1 类的鼠标经过时的样式 */
.class2:link { text-decoration:underline;} /* 定义 class2 类的正常样式 */
/* 定义 class2 类的鼠标经过时的伪样式 */
.class2:hover { font-size:18px; font-weight:bold; }
#id1:hover { color:blue;} /* 定义 id 为 id1 的鼠标经过时的样式 */
#id2:hover { border:solid 1px red; } /* 定义 id 为 id2 的鼠标经过时的样式 */
</style>
<a href="#">无下划线、灰色超链接样式</a><br />
<a class="class1" href="#">鼠标经过时显示为红色</a><br />
<a class="class2" href="#">正常显示为下划线, 鼠标经过时字体增大加粗显示</a><br />
<a id="id1" href="#">鼠标经过时显示为蓝色</a><br />
<a id="id2" href="#">鼠标经过时显示有红色边框线</a>
```


在上面的示例中, 首先定义 a 元素 (即所有超链接) 清除默认的下划线, 并显示为灰色。然后分别为不同的选择符定义不同的伪类样式。这仅是简单的字体属性设置, 当然你也可以定义其他 CSS 属性。

对于上面的超链接样式, 如果在 IE 6 及以下版本中预览, 你会发现第 3 个超链接在正常情况下应该显示有下划线, 但是 IE 6 浏览器却忽略了这个样式 (如图 5.1 所示), 而在 IE 7 或者其他标准浏览器中显示是带有下划线的 (如图 5.2 所示)。这是一个很奇怪的现象, 要解决这个问题, 你还需要为 a 元素定义相同的样式, 也就是说定义了 :link 样式之后, 再为 a 元素补设一个相同的样式, 这样就能够在 IE 6 及以下版本中正常显示。

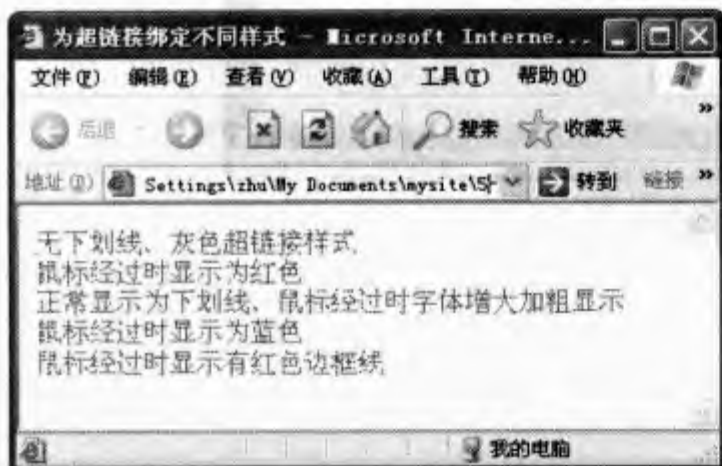


图 5.1 IE 6 下解析超链接存在的 Bug

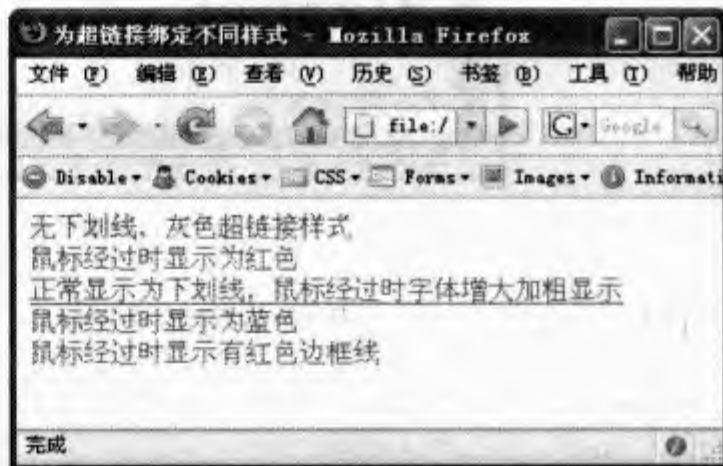


图 5.2 Firefox 2 下解析效果

我们还可以在这些伪类前面增加 a 元素前缀:

```
<style type="text/css">
a { text-decoration:none; color:#666; }
a.class1:hover {color:red;}
a.class2:link { text-decoration:underline;}
a.class2:hover { font-size:18px; font-weight:bold; }
a#id1:hover { color:blue;}
a#id2:hover { border:solid 1px red; }
</style>
```

a:link 和 a:visited 分别表示未访问过和已经访问过的超链接样式, 当希望访问过和未访问过的样式相同时, 你不妨为 a 元素定义样式, 这样会方便了许多。但是, 由于页面内的锚点也是使用 a 元素来定义, 因此使用 a 元素定义超链接样式也会影响到锚点标记。如果仅希望对超链接施加影响, 建议不要使用 a 元素来直接定义样式。

实际上, 设计师更喜欢在 a:hover 伪类上面做文章, 通过为该伪类定义创意的样式, 而使页面富有动感而又有新意。这比使用笨着的 JavaScript 来控制鼠标经过时的动态效果要简单许多。CSS 还提供了 :focus 伪类, 它表示当元素或者对象获取焦点时所要显示的样式, 如果你不喜欢使用超链接和鼠标操作, 使用 :focus 伪类也会让你的网页操作变得富有动感。但是很遗憾, 目前 IE 还不能够很好的支持该伪类样式, 因为在其他浏览器上可以有很多小技巧让你的页面看起来更令人愉快。

5.1.4 为页面定义不同的超链接样式

网页内的超链接如同天上星辰, 不计其数。但是你所看到的天上星星大小、明暗闪烁, 静止游动, 编织了一幅富有想象力的星空图。当然, 你也可以为网页内的超链接定义不同的显

示样式，这主要利用 CSS 的包含选择符（或者称为派生选择符）来实现。

例如，利用 Dreamweaver CS3 所提供的 CSS 标准布局模板新建一个“1 列固定，居中，标题和脚注”的文档，然后我们准备在不同模块内定义不同的超链接样式（如图 5.3 所示）。

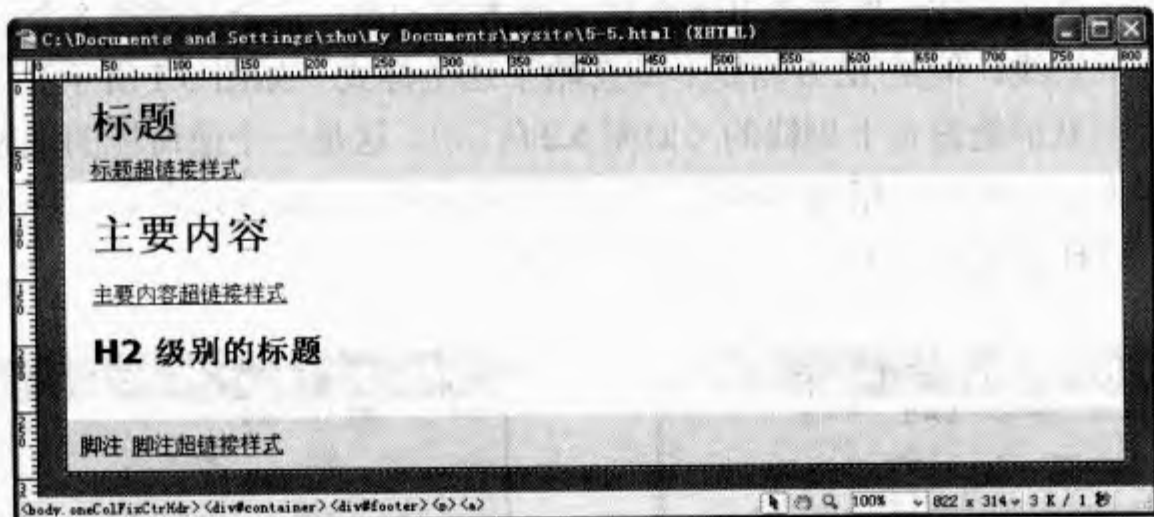


图 5.3 “1 列固定，居中，标题和脚注”的文档

该模板文档的基本框架结构如下所示：

```
<div id="container">
  <div id="header">
    <h1>标题</h1><a href="#" >标题超链接样式</a>
  </div>
  <div id="mainContent">
    <h1>主要内容 </h1><p> <a href="#" >主要内容超链接样式</a></p>
    <h2>H2 级别的标题 </h2>
  </div>
  <div id="footer">
    <p>脚注 <a href="#" >脚注超链接样式</a></p>
  </div>
</div>
```

下面在样式表中就可以为不同的模块区域定义不同的超链接样式：

```
<style type="text/css">
a { /* 定义超链接默认显示公共样式 */
  text-decoration:none;           /* 取消下划线 */
  color:#000;                     /* 黑色字体 */
}
#header a:visited { color:#333; } /* 头部区域已访问超链接以深灰色显示 */
#header a:hover {color:#FF0000; } /* 头部区域经过超链接时以红色显示 */
#mainContent a:visited { color:#bbb; } /* 主要内容区已访问超链接以灰白色显示 */
#mainContent a:hover {color:#0000FF; } /* 主要内容区经过超链接时以蓝色显示 */
#footer a:visited { color:#999; } /* 脚部区域已访问超链接以浅灰色显示 */
#footer a:hover {color:#00FF00; } /* 脚部区经过超链接时以紫色显示 */
</style>
```

这时你就会发现超链接在不同的区域呈现不同的显示样式（如图 5.4 所示）。这样利用包含选择符就可以在不同的区域设计不同的超链接样式，实际上你还可以配合其他选择符为页面设计更具争奇斗艳的超链接样式。

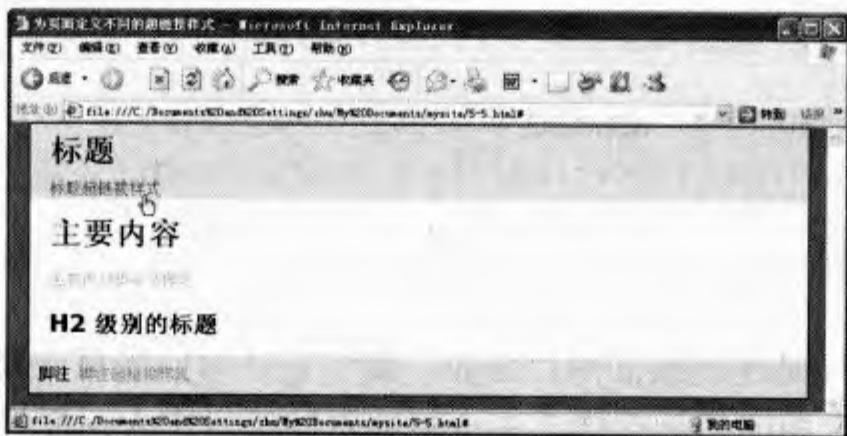


图 5.4 显示不同超链接样式的页面

5.2 花枝招展的超链接样式

超链接是网页中最活跃的元素，你应该把它打扮得更加漂亮，把它设计得更具个性。当然所有的期望都是建立在一定的技术基础上。在上一节中，我们学会如何使用伪类定义超链接的样式，以及如何为不同的超链接定义不同的样式。但是仅仅如此还是很不够的，因为你需要了解有关 a 元素更多的性格、相貌特征，并学会如何结合外部其他因素，使它看起来活泼、灵动，而不是淘气，令人生厌。

当然，所有的设计和创意都是建立在让超链接在网页中看起来更醒目，更容易让用户发现它，而不是相反。正所谓鹤立鸡群，让它在网页中脱颖而出。

5.2.1 为超链接定义普通样式

设计超链接的样式有很多技巧。例如，你可以给它设计有个性的边框，或者为它涂抹背景色，以背景色来夺人眼球，或者通过为超链接文本定义不同的字体属性，也一样让人感兴趣。

在下面这个样式表中，我们将为三段文本定义不同的超链接样式，分别从边框、背景色和字体属性等三个角度来比较不同样式给超链接带来的审美感受（如图 5.5 所示）。

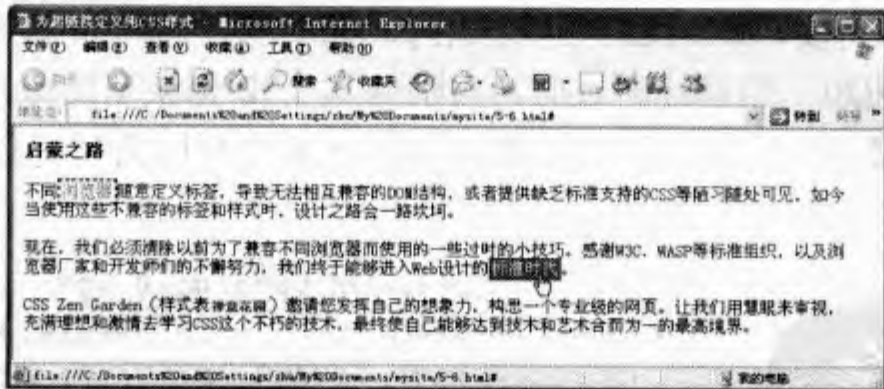


图 5.5 定义不同的超链接样式

```
<style type="text/css">
/* 统一定义超链接的默认显示样式：去掉下划线、增加边距，灰色字体 */
a { text-decoration:none; padding:2px; color:#666;}
/* 第一段超链接样式：默认为 2 像素蓝色虚线框，鼠标经过为 2 像素红色实线框 */
.p1 a { border:dashed 2px blue;}
.p1 a:link { border:dashed 2px blue;}
.p1 a:hover { border:solid 2px red;}
/* 第二段超链接样式：默认为浅紫背景色，鼠标经过为深紫背景色并以白色字体显示 */
.p2 a { background:#FF99FF;}
.p2 a:link { background:#FF99FF; }
```

```
.p2 a:hover { background:#CC00CC; color:#FFFFFF; }
/* 第三段超链接样式:
默认为 12 像素大小红色加粗字体, 鼠标经过为 18 像素大小蓝色正常字体 */
.p3 a { font-size:12px; color:red; font-weight:bold;}
.p3 a:link { font-size:12px; color:red; font-weight:bold; }
.p3 a:hover { font-size:18px; color:blue; font-weight:normal; }
</style>
```

蓝色下划线这个曾经是超链接的醒目标志, 如今设计师似乎首先要改变它, 然后再为超链接打上自己的烙印。从上面的超链接样式中也能够看到笔者本人习惯性的在样式表的第一行清除掉这个落伍的样式。当然废除旧有的样式, 必须保证有新的样式跟进, 否则浏览者就会迷失在你的网页中, 而不知走向何方, 因为所有超链接隐藏于网页文本中。

5.2.2 避免影响版面晃动的超链接样式

你可能遇到过, 当然在上节示例中你也已经看到这样的问题: 当把鼠标移到第 3 段超链接文本时, 由于字体突然变动, 整个段落被晃动了一下, 段落版式也发生了变化, 这样子很是吓人。特别是在版式固定的页面中, 因为一个超链接而破坏了网页布局的整体效果是件很烦心的事情。相信你在设计导航条时会经常遇到。

分析出现的原因, 主要是因为定义超链接的不同状态样式时, 所定义的边框或者边距大小不同导致的, 另外改变字体大小也可能导致此类问题的出现 (如上面示例)。

也许你希望超链接显示为红色下划线, 而当鼠标移过时显示为蓝色虚线框, 样式代码如下:

```
.p1 a {text-decoration:underline; color:red;}
.p1 a:link { text-decoration:underline; color:red;}
.p1 a:hover {text-decoration:none; border:dashed 1px blue; }
```

但是, 当鼠标移过超链接时, 你也很快发现整个文本行也轻微的发生变化, 这是因为当超链接在鼠标移过时, 左右边框增加了 2 个像素的宽度。但设计的初衷并不希望整个文本行因为超链接而晃动, 也许最初设计时你都没有预测到会有这种结果。不过我们可以采用补偿的方法来解决此类问题。既然你准备为鼠标移过时增加显示超链接的左右边框, 也就是说为超链接增加 2 个像素的宽度, 那么我们可以利用 padding 或者 margin 属性为超链接正常显示时设置一个 2 像素宽的边距, 当鼠标移过时, 再清除这个 2 像素宽的边距, 实际上也就是把 2 像素的宽度让给超链接边框使用。这样你就不会看见文本行因为鼠标移过超链接时而发生的轻微颤动了。

```
.p1 a {text-decoration:underline;color:red; padding:0 1px;}
.p1 a:link { text-decoration:underline;color:red; padding:0 1px;}
.p1 a:hover {text-decoration:none;border:dashed 1px blue; padding:0; }
```

而对于字体大小的变化, 控制起来就比较麻烦。例如, 针对上节演示示例的第三段中的超链接样式。当鼠标移过时, 由于字体突然变大, 整个段落文本会发生剧烈变化。

我们不仅利用上面介绍的补偿的方法补偿超链接左右边距的前后变化, 还要考虑超链接字体变大之后在上下空隙方面对段落文本行产生的影响。因此你可以定义一个固定的高度, 避免行高也因字体大小的变化而变化。实现的样式表如下:

```
.p3 a { font-size:12px; color:red; font-weight:bold; padding:10px; line-height:24px;}
.p3 a:link { font-size:12px; color:red; font-weight:bold; padding:10px;}
.p3 a:hover { font-size:18px; color:blue; font-weight:normal; padding:0; }
```

这时如果再次预览页面, 就会发现第三行文本没有受到超链接字体大小变化的影响, 如图

5.6所示。当然在具体设计时,你需要根据情况适当调整这些值。

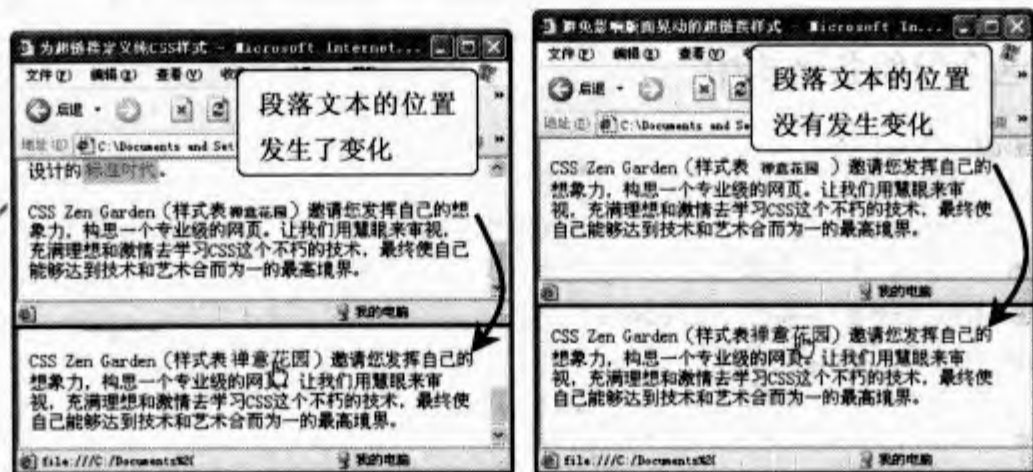


图 5.6 避免影响版面晃动的超链接样式

5.2.3 模拟按钮样式

在本地应用程序中你可能经常看到不同样式的按钮,这些按钮以富有动态感和立体感而引人注目。网页中为超链接设计按钮样式也是很容易的事情。它主要利用边框、边距和背景色的巧妙设计来实现。

例如,如果你为网页定义如下样式表,则整个网页文本中所有超链接如图 5.7 所示。

```
<style type="text/css">
body {
    background:#F0E7D7;                /* 定义网页背景色,衬托按钮效果 */
    line-height:1.6em;                 /* 定义行高,使按钮看起来很清楚 */
}
a {
    padding: 2px 8px;                  /* 增加边距,使按钮看起来更大方 */
    border: 1px solid;                  /* 定义边框线 */
    border-color: #ffe #aaab9c #ccc #fff; /* 为不同边框定义不同颜色,模拟立体感 */
    text-decoration: none;              /* 清除下划线 */
    background: #f0e7d7;                /* 增加按钮背景色 */
    color: #800000;                     /* 字体颜色 */
}
a:hover{
    background: transparent;            /* 透明背景色 */
    border-color: #aaab9c #fff #fff #ccc; /* 改变不同边框颜色,设计动态效果 */
}
</style>
```

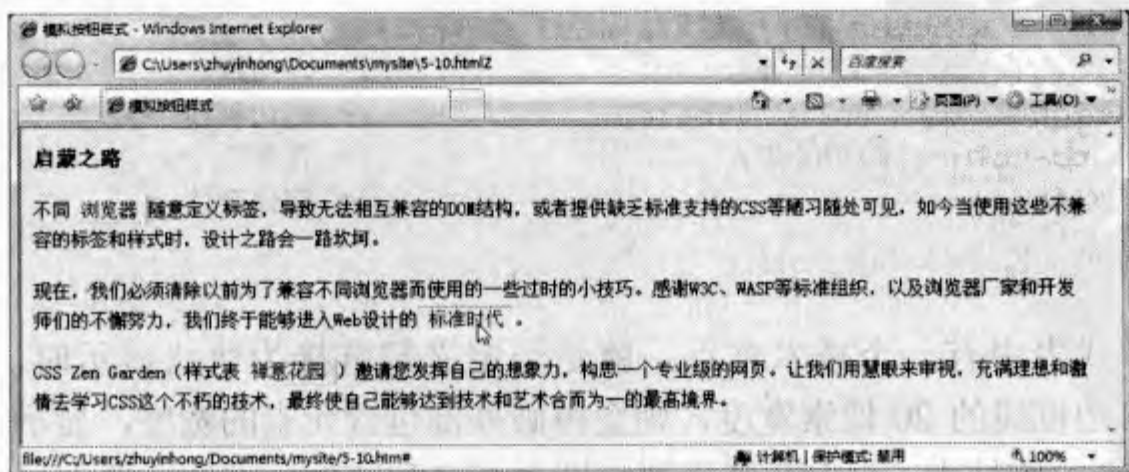


图 5.7 模拟按钮样式

在文本行内为超链接设计按钮效果好像不是很必要，但是在导航条中设计师却非常喜欢采用这种效果。

例如，在下面这个导航条结构中：

```
<div id="navcontainer">
  <ul id="navlist">
    <li id="active"><a href="#" id="current">菜单 1</a></li>
    <li><a href="#">菜单 2</a></li>
    <li><a href="#">菜单 3</a></li>
    <li><a href="#">菜单 4</a></li>
    <li><a href="#">菜单 5</a></li>
  </ul>
</div>
```

通过定义如下的样式表，则可以设计出如图 5.8 所示的立体按钮导航条效果。

```
<style type="text/css">
#navcontainer { /* 定义导航条外包含框的样式 */
  width: 12em; /* 固定导航条的宽度，容纳 12 个字 */
  border-right: 1px solid #000; /* 右边框 */
  padding: 0 0 4px 0; /* 增加导航条底部内边距 */
  margin-bottom: 1em; /* 增加导航条底部外边距 */
  background-color: #90bade; /* 导航条背景色 */
  color: #333; /* 导航条菜单字体颜色 */
}
#navcontainer ul { /* 清除项目列表样式 */
  list-style: none; /* 清除项目列表符号 */
  margin: 0; /* 清除缩进 */
  padding: 0; /* 清除缩进 */
}
#navcontainer li { /* 项目列表底部边框线 */
  border-bottom: 1px solid #90bade;
}
#navcontainer li a { /* 定义超链接默认显示样式 */
  display: block; /* 块状显示，以便定义宽度 */
  padding: 5px 5px 5px 0.5em; /* 增加内边距 */
  border-left: 10px solid #1958b7; /* 左边框线 */
  border-right: 10px solid #508fc4; /* 右边框线 */
  background-color: #2175bc; /* 背景色 */
  color: #fff; /* 字体颜色 */
  text-decoration: none; /* 清除下划线 */
  width: 100%; /* 超链接宽度 */
}
html>body #navcontainer li a { width: auto; } /* 兼容 IE 7 和标准浏览器 */
#navcontainer li a:hover { /* 定义鼠标经过时的样式 */
  border-left: 10px solid #1c64d1; /* 左边框线 */
  border-right: 10px solid #5ba3e0; /* 右边框线 */
  background-color: #2586d7; /* 背景色 */
  color: #fff; /* 字体颜色 */
}
</style>
```

在上面的样式表中有一个技术难点：就是当定义超链接为块状显示时，其宽度设置为 100%，加上左右边框线的 20 像素宽度，则会撑破外部包含元素的宽度，显示效果如图 5.9 所示。此时通过一个兼容技术，强制超链接的宽度在 IE 7 和其他标准浏览器中显示为自动，则这样就可以避免导航条宽度被撑破的尴尬效果。



图 5.8 按钮效果的导航条



图 5.9 按钮中不兼容效果

类似按钮效果的导航还有很多，可以说是创意无限，但是如果你把握按钮设计的一般规律，那么所有操作似乎又是那么的简单。请你记住如下几条规则：

第一，背景色是按钮效果的必要衬托。你不妨细心的观察不同的按钮式导航条，它们无一例外的都是依靠背景色来烘托按钮的逼真性。可以这样说，设计恰当的背景色能够使按钮看起来更逼真，相反会削弱按钮的视觉效果。

背景色的设置应该根据具体环境确定，当然要巧妙的配合整体背景色、超链接正常背景色以及鼠标经过时背景色，其中需要设计师超强的审美预知。

第二，超链接的边框线是按钮的必要条件。可以毫不夸张地说，如果没有边框样式的设计，也就没有按钮效果。设计超链接的边框时，应该对于不同的边设计不同的颜色，必要时可以设计不同的宽度，当时应该避免使用边框样式。对应的你还要设置鼠标经过时超链接不同边框的样式，以便增加按钮的动态效果。

不同边框设计不同的颜色，通过明暗、深浅来模拟立体的视觉效果。如果你仔细观察立体在平面内的投影，你会立刻明白如何去设计富有立体感的效果。

第三，恰当的应用超链接的边框也是一个重要的条件，各个按钮并列显示在一起，如果不借助边框是无法辨别它们的独立关系。

5.2.4 使用图像设计超链接样式

图像永远是设计师的最爱，超链接自然也不能够放过图像这个好搭档，你会在网页中经常看见用图像设计的超链接按钮。

为超链接设计图像样式可以有多种方法，其中最常用的方法是借助 CSS 的 background-image 属性来定义超链接的背景图像。例如，使用如下 CSS 样式可以设计如图 5.10 所示。

```
<style type="text/css">
a.btn2 { /* 超链接样式 */
    background: transparent url('images/btn2.gif') no-repeat top left; /* 背景图像 */
    display: block; /* 块状显示 */
    width: 74px; /* 宽度，与背景图像同宽 */
    height: 25px; /* 高度，与背景图像同高 */
    text-indent: -999px; /* 隐藏超链接中的文本 */
}
</style>
<a class="btn2" href="#">图像按钮</a>
```

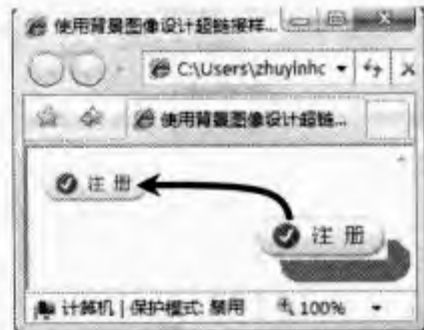


图 5.10 使用图像设计按钮

也许你在为如何使用 CSS 设计富有立体感的按钮而苦恼时，使用背景图像会让你感觉很轻松而且惬意，甚至设计出很多超乎想象的样式来。当然在使用背景图像来设计超链接时需要记

5.2.5 设计具有扩展性的超链接样式

也许你会认为：使用背景图像设计的超链接，倒不如直接在网页中插入图像，然后为图像绑定超链接来得更方便些？这话虽有一定的道理，但是也不完全正确，如果当你在网页设计定义很多超链接时，上节介绍的方法还是比较适用的。但是如果你学完本节内容，相信你会更喜欢使用背景图像来定义超链接样式，因为它更富有扩展性。

很多时候设计师希望一劳永逸，通过找到一种更巧妙的方法使超链接样式具有更广泛的应用价值。实际上如果浏览以下网上导航条时，你也会发现很多这样让人激动的设计效果。很多设计师喜欢把一个绘制好的图像分成两截，让其中的一截固定在超链接按钮的一头，另一截固定在另一头，如果按钮图像设计得足够大的话，你会发现不管超链接包含的字数有多少，字体有多大，它都能够使设计的按钮显示为一致的设计效果。如图 5.13 所示是卓越网的导航栏，它就利用如图 5.14 所示的设计原理来设计的。

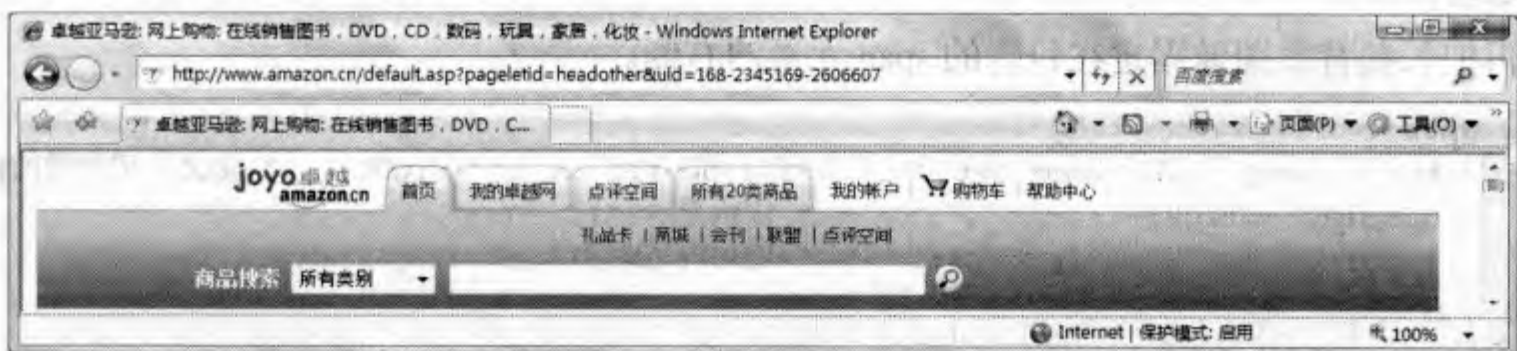


图 5.13 卓越网导航栏效果

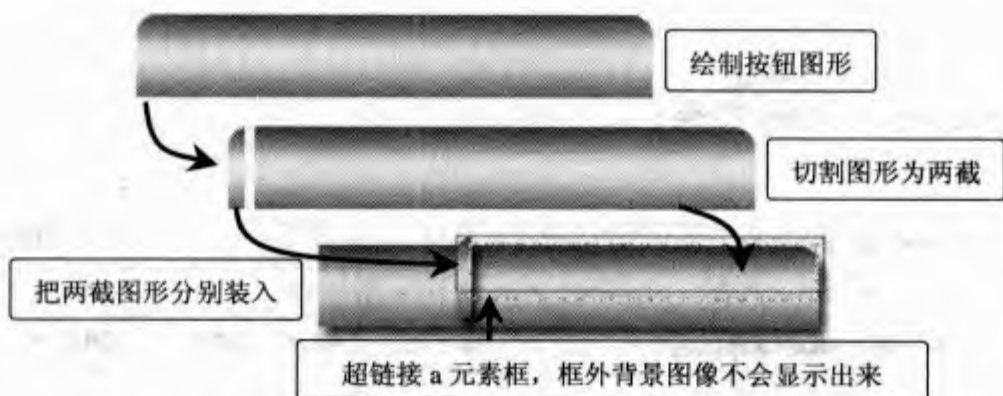


图 5.14 可扩展超链接图形按钮设计原理示意图

这个示意图直观地展示了可扩展图形按钮的设计原理。为了帮助读者更直观的理解和体会，我们再结合一个示例进行讲解。

首先读者需要在图像编辑器中设计好按钮图形的效果图，然后分切为两截，其中一截应尽可能的窄，只包括一条椭圆边，另一截可以尽可能宽，这样设计的图按钮就可以容纳更多的字符（如图 5.15 所示）。



图 5.15 绘制并裁切图形

构建一个可以定义两个背景图像的超链接结构，具体结构如下，在这个超链接中又包含了一个 span 元素。

```
<a href="#"><span>图形按钮</span></a>
```

然后使用 CSS 把短截的背景图形固定在 a 元素的一头。

```
<style type="text/css">
a { /* 定义超链接样式 */
    background: url('images/left1.gif') no-repeat top left; /* 把短截背景图像固定在左侧 */
    display: block; /* 以块状显示, 这样能够定义大小 */
    float: left; /* 浮动显示, 这样 a 元素能够自动收缩宽度, 以正好包容文本 */
    padding-left: 8px; /* 增加左侧内边距, 该宽度正好与上面定义的背景图像同宽 */
    font: bold 13px Arial; /* 超链接文本字体属性 */
    line-height: 22px; /* 定义行高 */
    height: 30px; /* 定义按钮高度 */
    color: white; /* 字体颜色 */
    margin-left: 6px; /* 左侧外边框 */
    text-decoration: none; /* 清除默认的下划线样式 */
}
</style>
```

再把长截背景图像固定在包含的 span 元素的右侧。

```
a span {
    background: url('images/right1.gif') no-repeat top right; /* 定义长截背景图像 */
    display: block; /* 块状显示 */
    padding: 4px 10px 4px 2px; /* 增加内边距 */
}
```

好了, 大功告成 (显示效果如图 5.16 所示)。如果高兴你还可以定义鼠标经过时让背景图像的色彩稍稍有点变化, 这样增加按钮的动态感。这里就简单的给鼠标经过时增加一个下划线效果。

```
a:hover {
    text-decoration: underline;
}
```



图 5.16 可扩张的图像超链接效果

这是一个很有趣的设计, 当然我们还是把美好的想象留给你, 看看你是不是可以设计一个令人眼睛一亮的好作品来, 就让我们拭目以待。

5.3 构建导航条列表框架及其基本样式

集体的力量是伟大的。如果超链接散落于网页文本之中, 似乎不会给设计师带来更多的设计激情。但是如果把多个超链接捆绑在一起, 形成一个导航条 (或称为菜单) 时, 设计师们会不遗余力的为此通宵达旦, 创新的渴望和设计的激情总会交织于此。所以当你浏览网页时, 总会时不时的被各种惊艳或者超酷的导航条迷倒了。

5.3.1 为什么使用项目列表来构建导航条

似乎所有的设计师都喜欢使用项目列表来组织构建导航条的结构。当然这不是主观情绪使然，更多的是因为项目列表所拥有的语义特征和极强的组织纪律性。

使用正确语义代码是构建符合标准的网站基本要求。好的 HTML 结构应基于有逻辑、有顺序和有语义的正确标记。如果网页标题使用标题元素（如 h1、h2 或 h3），如果段落文本使用 p 元素，所有的菜单或列表使用项目列表元素，你会感觉网页结构井然有序。如果你使用符合语义的 HTML 结构，那么基于文本的浏览器、荧屏阅读器，或者不支持 CSS 的浏览器，以及搜索引擎会更容易访问你的内容。

从结构层分析，网站的导航条是一个链接到网站其他区域的简单列表。所以，建议你最好使用项目列表来构建导航条。项目列表极强的组织性也是我们选择它来设计导航条的一个主要原因。使用它你会感觉管理信息、设计样式都非常的简单、轻松。假如有一天浏览器无法正确解析网页样式表时，项目列表依然有组织的显示在一起，如图 5.17 所示的是 Adobe 中国首页关闭样式表功能之后显示的信息，你会看到项目列表所组织的信息很醒目的显示在一起。

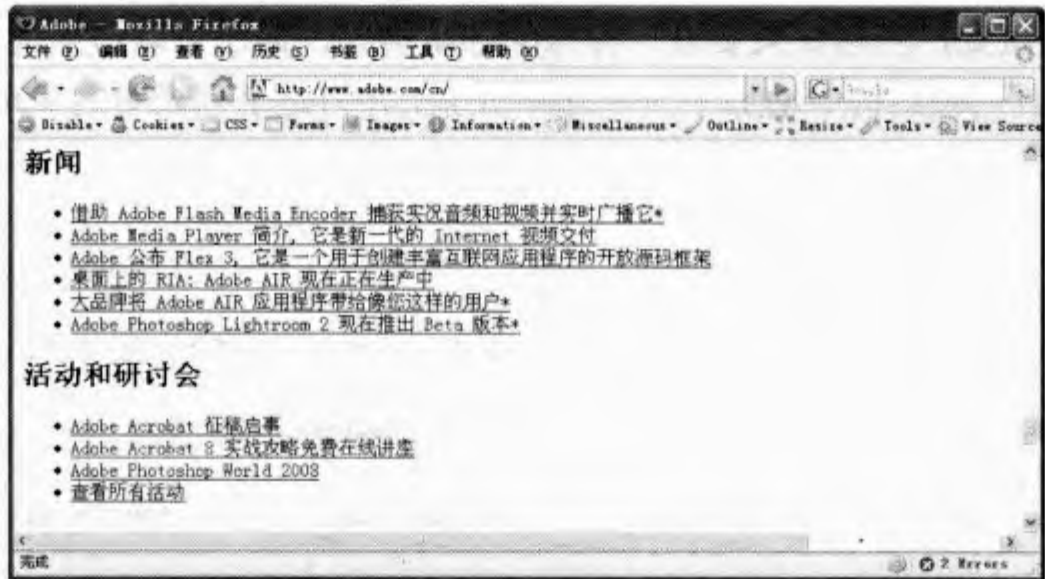


图 5.17 无样式下项目列表的显示效果

项目列表可以分为有序和无序两种。所谓有序列表，就是项目符号显示为有序的符号，如数字、顺序编号等，也称为编号列表。它由 ol 和 li 元素配合完成：

```
<ol>
  <li>有序列表</li>
  <li>有序列表 </li>
  <li>有序列表 </li>
</ol>
```

所谓无序列表，就是每个项目之间没有先后顺序，所有项目符号相同。它由 ul 和 li 元素配合完成：

```
<ul>
  <li>无序列表</li>
  <li>无序列表 </li>
  <li>无序列表 </li>
</ul>
```

5.3.2 搭架结构合理的导航列表

在设计导航条时，你可以使用任何一种类型的列表，这不妨碍你的导航条设计，不过设计

师更习惯于使用无序列表，这是因为很多列表都是无序显示的，使用有序列表也没有什么过错，但是总感觉有点别扭，大概这是个人习惯所致。

具体应用中你可能很少看到有人直接使用上面的列表结构来完成导航条的设计。为了方便 CSS 控制导航列表，你不妨为导航列表增加一个包含元素，为主要标签增加一个 ID 属性或者类样式（如下所示）。当然这里没有强制的规则来要求你必须这样或者那样，而是根据你的网页需要酌情增设。

```
<div id="navcontainer">
  <ul id="navlist">
    <li id="active"><a href="#" id="current">菜单 1</a></li>
    <li><a href="#">菜单 2</a></li>
    <li><a href="#">菜单 3</a></li>
    <li><a href="#">菜单 4</a></li>
    <li><a href="#">菜单 5</a></li>
  </ul>
</div>
```

这种做法不是为了代码好看，主要是方便设计师能够更从容的设计出复杂样式的导航条，相信你也一定能够做到，当然如果要设计特殊的导航样式，你还可以适当增加其他辅助元素。例如，在上面的导航列表结构中增加了 3 个 span 元素，并为每个列表中的 a 元素增加了类样式。

```
<div id="navcontainer">
  <ul id="navlist">
    <li id="active"><a href="#" id="current" class="menu1"><span><span><span>
    菜单 1</span></span></span></a></li>
    <li><a href="#" class="menu2"><span><span><span>菜单 2</span></span></span>
    </a></li>
    <li><a href="#" class="menu3"><span><span><span>菜单 3</span></span></span>
    </a></li>
    <li><a href="#" class="menu4"><span><span><span>菜单 4</span></span></span>
    </a></li>
    <li><a href="#" class="menu5"><span><span><span>菜单 5</span></span></span>
    </a></li>
  </ul>
</div>
```

也许你被如此复杂的结构弄迷糊了，甚至疑惑搭建如此结构是不是画蛇添足？答案是否定的。这是设计的需要，请你看看如图 5.18 所示的导航条，它正是利用如此结构构建的。如果说没有这样的结构你无法设计如此效果时，可能稍微能够接受如此画蛇添足的做法。

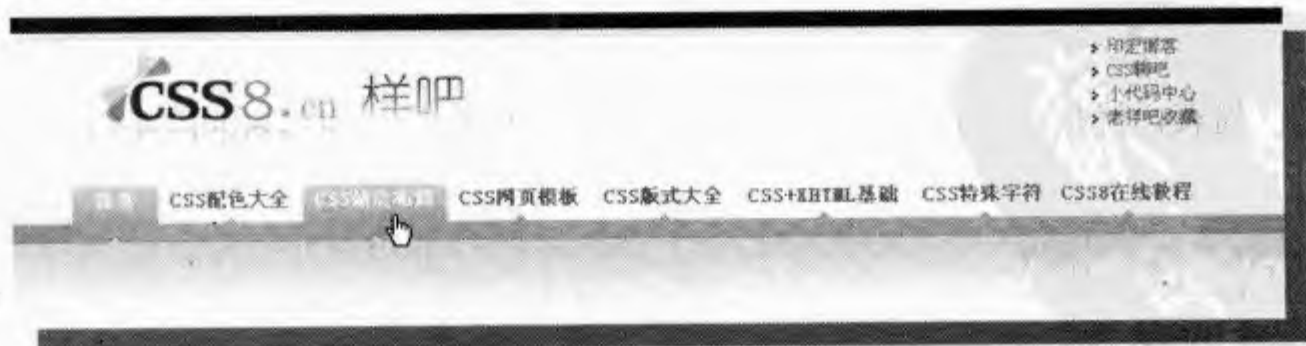


图 5.18 复杂的导航列表结构效果图

创意无限，设计无疆，结构总是因设计而搭建，而不是为了结构而进行设计。不同的创意可能需要不同的结构，这就看你的具体需要。所以读者可以在掌握基本规律之后灵活搭建导航列表的结构。

5.3.3 使用定义列表搭架导航列表

除了有序列表和无序列表外，定义列表也常被设计师用来搭建导航条。定义列表包含 3 个元素：dl、dt 和 dd。其中 dl 与 ul 和 ol 元素功能类似，相当于列表外框架，dt 表示定义列表的标题，这在有序列表和无序列表中是没有的，而 dd 与 li 元素功能类似。使用定义列表的优势就在于它包含了 3 个元素，多一个元素自然要强大许多。很多时候，你能够利用 3 个元素可以定义更多的导航条样式。例如，针对上面的导航结构，你可以使用定义列表来构建：

```
<div id="navcontainer">
  <dl id="navlist">
    <dt>网站标题</dt>
    <dd id="active"><a href="#" id="current">菜单 1</a></dd>
    <dd><a href="#">菜单 2</a></dd>
    <dd><a href="#">菜单 3</a></dd>
    <dd><a href="#">菜单 4</a></dd>
    <dd><a href="#">菜单 5</a></dd>
  </dl>
</div>
```

如果你不需要使用 dt 元素来显示网站标题，那么可以使用 CSS 来隐藏该元素的显示，并借助该元素为导航条定义背景图像，以实现更复杂的效果设计。也许在上一章中你曾经为了设计圆角而增加 3 个辅助元素时，现在使用定义列表就已经包含了 3 个元素。类似的示例我们会在后面的章节涉及，限于篇幅这里先省略了。

5.3.4 控制项目列表的样式

项目列表在默认状态下显示为左缩进样式，每个项目列表都会占据一行，呈垂直并列显示效果。这种默认的样式是基于网页在屏幕中从上向下解析而形成的一种最佳显示效果。但是在网页中，很多设计师，包括广大用户都不喜欢这种没有个性的样式，而且它占据了大量的屏幕空间，是不经济的显示效果。

好在 CSS 提供了一套支持项目列表的属性，使得设计项目列表的工作变得简单多了。你可以使用 list-style-type 属性定义项目符号的类型，使用 list-style-image 属性自定义项目符号所要显示的图标，使用 list-style-position 属性定义项目符号的位置，当然使用 list-style 复合属性可以定义项目列表各种属性。

但是设计师更喜欢清除导航列表中的这种默认样式。例如，针对上面的无序列表结构，你可以定义如下样式清除其中的项目符号和列表缩进效果，清除前后的比较效果如图 5.19 所示。

```
<style type="text/css">
ul#navlist {
  padding: 0;      /* 清除标准浏览器中的缩进显示 */
  margin: 0;      /* 清除 IE 浏览器中的缩进显示 */
  list-style: none; /* 清除项目符号 */
}
</style>
```

也许你会认为清除默认样式的导航列表更加难看。不过不要着急，这里先稍施小计，在上面清除默认列表样式的基础上增加如下样式，即可设计一个很靓的列表（如图 5.20 所示）。

```
ul#navlist { /* 为列表框底部定义浅灰色虚线 */
```

```

border-bottom: 1px dashed #aaa;
}
#navlist li {
padding: 0.5em;           /* 为列表项定义半个字距的内边距 */
border-top: 1px dashed #aaa; /* 为列表项顶边定义浅灰色虚线 */
}
#navlist li a {
text-decoration: none;    /* 清除项目符号 */
}

```



图 5.19 清除项目列表中的默认样式

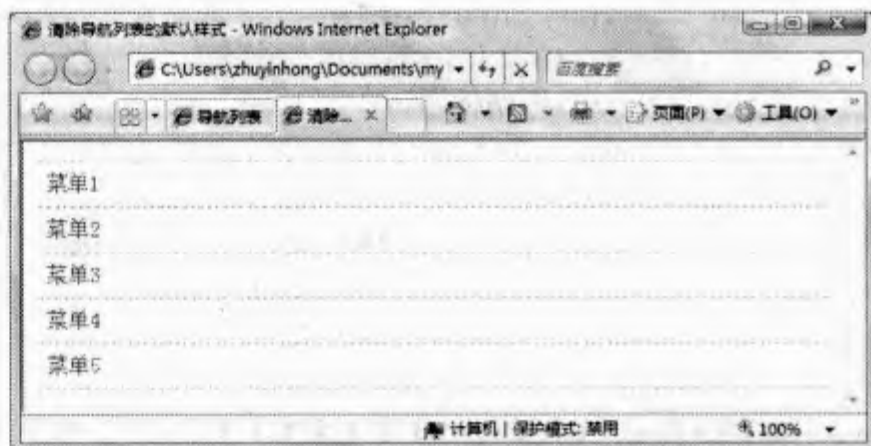


图 5.20 简单的设计导航列表样式

就这么简单，当然相信你不会满足如此雕虫小技，我们的话题还会在后面不但深入，总之希望能够激活你的创意细胞，找到设计的灵感。

5.3.5 有趣的项目符号

项目符号是导航列表中一个重要的标志。不管你喜欢与否，如果离开项目符号，很多时候你会感觉列表显得很突兀。在 CSS 中你可以借助 `list-style-type` 属性来定义项目符号。无论是 `ul` 元素还是 `ol` 元素，都可以定义相同的项目符号，所显示的效果完全相同。从这点意义上讲，使用 `ul` 和 `ol` 元素构建列表样式基本上没有什么区别。

CSS 支持很多项目符号，但是目前能够被各家浏览器支持且广泛应用的包括如下项目符号：

- `disc`：实心圆，默认值。
- `circle`：空心圆。
- `square`：实心方块。
- `decimal`：阿拉伯数字。
- `lower-roman`：小写罗马数字。
- `upper-roman`：大写罗马数字。
- `lower-alpha`：小写英文字母。
- `upper-alpha`：大写英文字母。
- `none`：不使用项目符号。

这些项目符号可以相互嵌套使用。并可以利用 `list-style-position` 属性设置项目符号显示的位置，它包括两个值：`outside`（列表文本外，默认值）和 `inside`（列表文本内）。

例如，在下面的项目列表中嵌套了 3 层结构，然后使用 CSS 分别为它们定义不同的项目符号，显示效果如图 5.21 所示。这是一个实用价值的示例，不过你能够从中学习到如何去定义项目符号，以及如何实现项目列表嵌套。


```

<style type="text/css">
ul { /* 空心圆，内部显示*/
    list-style-type: circle;
    list-style-position: inside;
}
ul ul { /*阿拉伯数字，外部显示*/
    list-style-type: decimal;
    list-style-position: outside;
}
ul ul ul { /* 阿拉伯数字，外部显示*/
    list-style-type: upper-alpha;
    list-style-position: inside;
}
</style>
<ul>
    <li>项目列表-空心圆，内部显示
        <ul>
            <li>项目列表-阿拉伯数字，外部显示
                <ul>
                    <li>项目列表-大写罗马数字，内部显示 </li>
                </ul>
            </li>
        </ul>
    </li>
</ul>

```

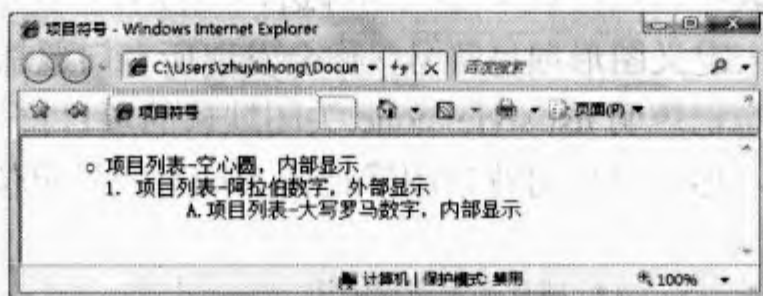


图 5.21 项目列表符号的使用

5.3.6 自定义图片项目符号

除了这些默认的项目符号外，CSS 还提供了 `list-style-image` 属性允许用户自定义图片项目符号。例如，针对第 5.3.4 节介绍的示例，我们为它定义一个图形项目符号：

```

#navlist li {
    list-style-image: url(images/hand1.gif);
}

```

当然你也可以在 `ul` 元素内定义：

```

ul#navlist {
    list-style-image: url(images/hand1.gif);
}

```

这都无关紧要。但是当你在 IE 7 或者 Firefox 等标准浏览器中预览时，你会很惊讶地发现所定义的图形项目符号不见了。原来是在上面示例中清除项目列表缩进，则这些图形项目符号都跑到边界的外面去了。

解决的方法是：恢复项目列表的缩进或者设置 `list-style-position` 属性为 `inside`。最后你看到的图形项目符号如图 5.22 所示。

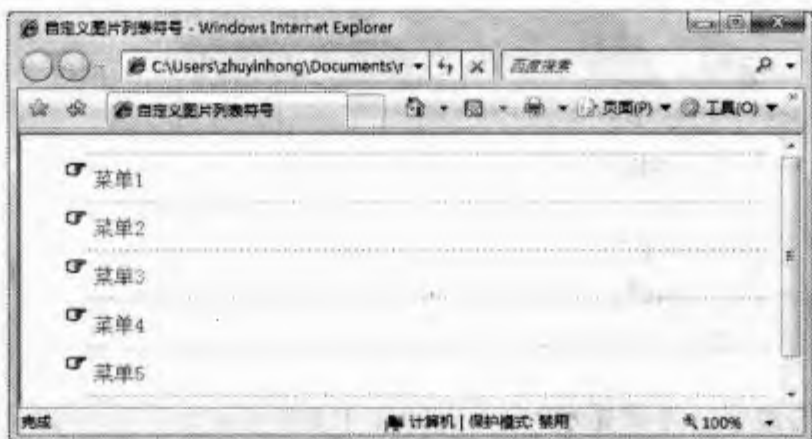


图 5.22 图形项目列表符号

当为项目列表定义了有效的 `list-style-image` 属性之后，不管你是否清除了项目列表中默认的项目符号，它都会自动被隐藏。

有一点请读者注意，项目符号在不同浏览器中所显示效果是不同的，如果你想设计兼容不同浏览器的项目列表，建议一定要在不同浏览器中预览一下，并针对不同浏览器采取不同的修补方法。

5.3.7 通过背景图像来定义项目符号

如果仔细研究一下图形项目符号的效果，你可能很不满意，这是因为除了上面所讲到的问题外，你无法精确控制图形项目符号的位置，以及它与项目列表之间的关系。例如，在如图 5.22 所示的自定义图形项目列表中，图形位于虚线的外边，且没有与文本对齐。

而如果使用背景图像来定义图形项目符号，你会发现所有问题都会烟消云散，操作起来会很顺手，当然也请你不要忘记声明 `list-style: none;` 关闭默认的项目符号。

例如，针对上面示例，你可以使用背景图像把图形项目符号定位到更精确的位置。

```
ul#navlist {
    padding: 0;           /* 项目列表清除缩进 */
    margin: 0;           /* 项目列表清除缩进 */
    list-style: none;     /* 隐藏项目列表符号 */
    border-bottom: 1px dashed #aaa;
}
#navlist li {
    background: url(images/hand1.gif) left center no-repeat; /* 定义背景图像 */
    padding: 0.5em 1.5em; /* 增加内边距，为背景图像预留显示区域 */
    border-top: 1px dashed #aaa;
}
```

这时你会发现使用背景图像能够更好地进行控制。当然在定义背景图像时，也不要忘记清除原来默认的项目符号以及缩进效果，同时通过 `padding-left` 属性为背景图像挤出显示的区域，另外你可以使用 `background` 或者 `background-position` 属性精确定位项目符号显示的位置，当然也不要忘记使用 `background` 或者 `background-repeat` 属性禁止背景图像平铺显示。

如果再结合超链接的几种显示状态，你还可以设计出更富动态效果的项目符号。例如，如果输入下面的样式表代码，该样式表所作用的项目列表结构可以参阅第 5.3.2 节介绍的 HTML 源代码，这里就不再重复显示。最后演示的效果如图 5.23 所示。

```
<style type="text/css">
ul#navlist {
    padding: 0;
    margin: 0;
```



```

list-style: none;
border-bottom: 1px dashed #aaa;
width:20em;                /* 固定项目列表的宽度 */
}
#navlist li {
padding: 0.5em;
border-top: 1px dashed #aaa;
}
#navlist li a { /* 超链接样式 */
display: block;                /* 块状显示 */
padding-left: 1.5em;           /* 为背景图像显示挤出位置 */
background: url(images/arrow3.gif) left center no-repeat; /* 固定背景图像在左侧 */
text-decoration: none;        /* 清除超链接的下划线 */
}
#navlist li a:link {           /* 定义未访问超链接背景图像 */
background: url(images/arrow3.gif) right center no-repeat; /* 固定背景图像在左侧 */
}
#navlist li a:visited {        /* 定义已访问超链接背景图像 */
background: url(images/arrow5.gif) right center no-repeat; /* 替换左侧背景图像 */
}
#navlist li a:hover {          /* 定义鼠标经过超链接背景图像 */
background: url(images/arrow4.gif) left center no-repeat; /* 固定背景图像到右侧 */
}
</style>

```

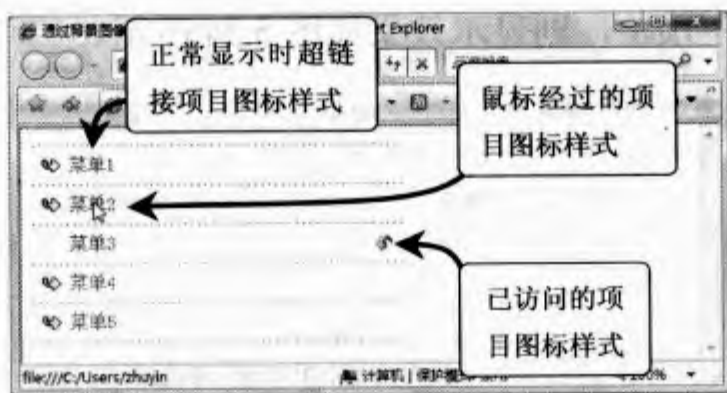


图 5.23 动态图形项目列表符号

如果你试一试，当鼠标移过超链接时背景图像被替换为另一个图像，给人感觉是项目符号颜色发生了变化，如果单击超链接，则可以看到项目符号以另一种图像显示在右侧，以提示该超链接已经被访问过了。

5.3.8 使用行内显示法设计项目列表水平显示

也许垂直排列的项目列表并不是你所钟情的版式，当然你可以让它们都并列显示在一起，这样能够为你的页面节省很多空间。让列表项并列显示可以有多种实现方法，你可以利用行内显示法、浮动法或者定位法都可以快速实现，不同的方法各有不同的优势和缺陷，你可以根据需要选择采用。本节先重点讲解如何使用行内显示法设计水平项目列表，后面还会继续讲解其他方法。

所谓行内显示法，就是定义列表项的 `display` 属性为 `inline`。在默认状态下项目列表项以 `list-item` 显示，它也是一种特殊的块状元素，一行只能显示一个项目列表元素，且还显示项目符号。通过改变项目列表的显示属性，来强制所有项目列表项在一行内显示，如同文本一样流动显示，当然如果在一行内显示满之后，还会在第2行继续显示。

例如，针对如下的项目列表结构（后面的示例都将以该结构为基础展开介绍，所以后面内容将不会重复介绍）。

```
<div id="navcontainer">
  <ul id="navlist">
    <li id="active"><a href="#" id="current">菜单 1</a></li>
    <li><a href="#">菜单 2</a></li>
    <li><a href="#">菜单 3</a></li>
    <li><a href="#">菜单 4</a></li>
    <li><a href="#">菜单 5</a></li>
  </ul>
</div>
```

定义一个简单的样式表：

```
<style type="text/css">
ul#navlist { /* 清除项目列表的缩进样式 */
  padding: 0;
  margin: 0;
}
#navlist li { /* 定义项目列表水平显示 */
  display: inline;          /* 定义项目列表以行内显示 */
  padding-right: 20px;      /* 增加项目列表右侧的空隙，拉开列表之间的间距 */
}
</style>
```

这时如果在 IE 浏览器中预览，则显示效果如图 5.24 所示。

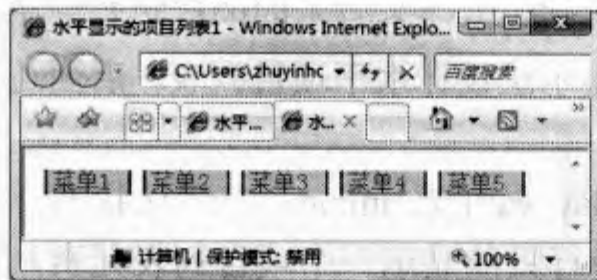


图 5.24 行内显示实现项目列表水平显示

你还可以为它们定义背景色、背景图像、边框线、间距等。例如，输入下面样式：

```
#navlist li {
  display: inline;          /* 行内显示 */
  padding: 4px;             /* 增加内边距 */
  background: #99CCFF;      /* 定义列表项背景色 */
  margin: 4px;              /* 增加项目列表之间的空隙 */
  border: solid 2px red;     /* 为项目列表定义边框 */
}
```

这时你会发现在不同浏览器中所呈现的解析效果截然不同，如图 5.25 所示。



在 IE 7 中解析效果



在 Firefox 中解析效果

图 5.25 不同浏览器解析效果比较

不同浏览器的解析效果也存在很大的不同,同时它们都不支持宽和高属性,也就是说你无法控制项目列表显示的形状,自然也就是无法设计各种特殊效果的样式。所以对于特殊需要的导航条来说,使用行内显示的方法实现水平显示不是最佳选择。

5.3.9 使用浮动法设计项目列表水平显示

所谓浮动法就是定义项目列表元素 li 浮动显示。当元素被定义为浮动显示之后,它们就可以并列排在一起,而且拥有了一定的形状,因此你可以通过为项目列表定义高和宽来定制列表项的形状,这为定制特殊按钮效果奠定了基础。

例如,输入下面样式表,你就可以设计如图 5.26 所示水平显示的项目列表:



图 5.26 浮动显示的水平列表项

```
<style type="text/css">
ul#navlist { /* 清除项目列表的默认样式 */
padding: 0;
margin: 0;
list-style-type: none;
}
#navlist li {
float: left; /* 浮动显示项目列表 */
padding: 2px 3px; /* 增加内边距 */
margin: 0 1px; /* 增加项目列表之间的空隙 */
background: #99CCFF; /* 定义列表项背景色 */
text-align: center; /* 居中对齐 */
line-height: 30px; /* 垂直对齐, 保证与高度值相同 */
border: solid 1px red; /* 定义边框 */
height: 30px; /* 设置项目列表的高度 */
width: 80px; /* 设置项目列表的宽度 */
}
</style>
```

5.3.10 设计水平浮动列表的超链接样式

当项目列表浮动显示时,它就拥有了很多设计功能,如定制宽和高、设置边距、设计对齐等,而且在不同浏览器中都能够正常解析和显示。这时如果设计超链接的不同显示状态,就应该把 a 元素设置为块状显示,因为超链接默认是行内元素,只有定义为块状显示之后,你才能够给它设置相关样式。

例如,在上节示例基础上设计当鼠标移过时项目列表块背景色显示为深蓝色,字体颜色为白色,而当超链接访问过之后则让它显示为灰色状态。所需要增加的样式如下,最后演示的效果如图 5.27 所示。当然这里仅演示如何设计超链接的不同状态样式,你可以根据这个设计方法设计更完美的动态效果。



图 5.27 设计水平浮动列表的超链接样式

```
#navlist li { padding: 0; } /* 清除列表项的内边距, 使超链接能够填充整个列表项 */
#navlist a {
display: block; /* 块状显示 */
height: 100%; /* 填充整个列表项内空间 */
```

```

background:none;          /* 默认不显示背景色 */
}
#navlist a:link {
    background:#99CCFF;    /* 鼠标经过时为浅蓝背景色 */
    color:#000000;        /* 鼠标经过时字体为黑色 */
}
#navlist a:visited {
    background:#999;       /* 已访问过的背景色为浅灰色 */
    color:#ddd;            /* 已访问过的字体颜色为灰白色 */
}
#navlist a:hover {
    background:#000099;    /* 鼠标经过时为深蓝背景色 */
    color:#fff;            /* 鼠标经过时字体为白色 */
}

```

5.3.11 解决浮动溢出问题

浮动列表固然有很多好处，但是浮动布局所带来的很多后遗症也是一件很烦心的事情，这主要体现在当导航条与其他网页模块相处时，所出现的各种错位、空隙等不兼容性问题。有关布局问题我们将会在后面章节中进行讲解。下面介绍项目列表水平浮动显示时所出现的浮动溢出问题。为了更直观的讲解，我们仍然以上节示例为基础进行介绍。

如果当你为 `ul` 元素定义背景色和边框线时。例如，增加如下样式：

```

ul#navlist {
    background:#FF00FF      /* 为项目列表元素定义背景色 */
    border:solid 1px blue;  /* 为项目列表元素定义字体色 */
}

```

在浏览器中预览时，你会很奇怪的发现所定义的背景色和边框线并没有显示出来，这是什么原因呢？原来这就是浮动溢出问题，所谓浮动溢出就是当子元素被定义为浮动显示时，而父元素流动显示，则子元素就会跑出父元素的怀抱，如果父元素没有定义高度，则你为它定义的背景色边框线就不会显示出来。

解决的方法是：定义 `ul` 元素也浮动显示。当元素浮动显示时，它会自动收缩宽度以正好包含子元素，这时如果让 `ul` 元素占据一行，你可以定义它的宽度为 100%。另外，为了预防导航条底部的对象或者元素从左右两侧绕过它，你还需要在其中定义清除属性，以清除浮动后面的内容。

```

<style type="text/css">
ul#navlist {
    float:left;          /* 浮动显示 */
    width:100%;          /* 定义宽度 */
}
.clear {clear:both; }    /* 清除浮动类 */
</style>
<ul id="navlist">
    <li id="active"><a href="#" id="current">菜单 1</a></li>
    <li><a href="#">菜单 2</a></li>
    <li><a href="#">菜单 3</a></li>
    <li><a href="#">菜单 4</a></li>
    <li><a href="#">菜单 5</a></li>
    <div class="clear"></div>
</ul>

```

如果此时预览页面，你就可以看到项目列表的背景色了，如图 5.28 所示。

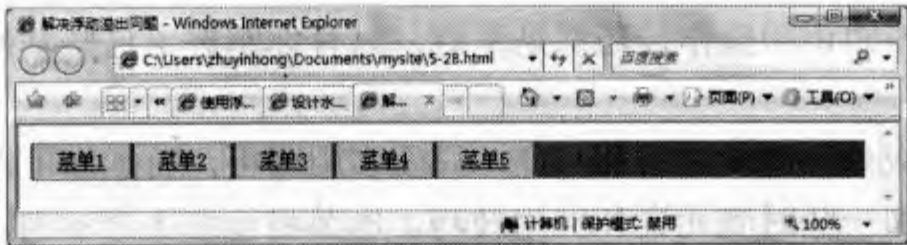


图 5.28 解决浮动溢出问题

5.3.12 使用定位法设计项目列表水平显示

使用定位法就是把所有列表项设置为绝对定位显示，然后设置 ul 元素为包含块，通过坐标值精确定位每个项目的位置。例如，针对上面的示例，我们删除所有浮动属性设置，而增加如下样式：

```
ul#navlist {
    height:36px;           /* 高度 */
    width:100%;           /* 宽度 */
    position:relative;     /* 相对定位，设置为包含块，这样当其子元素绝对定位时就以该元素左上角作为坐标原点进行定位 */
}
#navlist li {
    position:absolute;     /* 设置所有列表项都为绝对定位显示 */
    height:30px;           /* 高度 */
    width:80px;            /* 宽度 */
    top:2px;               /* 距离顶部的距离 */
}
#navlist .menu1 { left:2px;} /* 定义第 1 个列表项距离包含块左侧的距离 */
#navlist .menu2 { left:86px;} /* 定义第 2 个列表项距离包含块左侧的距离 */
#navlist .menu3 { left:170px;} /* 定义第 3 个列表项距离包含块左侧的距离 */
#navlist .menu4 { left:254px;} /* 定义第 4 个列表项距离包含块左侧的距离 */
#navlist .menu5 { left:338px;} /* 定义第 5 个列表项距离包含块左侧的距离 */
```

使用绝对定位法虽然比较精确，但是其灵活性就相对较差，你需要逐一设置每个列表项的坐标值，这对于列表项很多时，操作起来会很麻烦。如果页面布局为弹性或流动布局，使用绝对定位就存在一定的局限性。通过上面方法定义的导航条如图 5.29 所示。



图 5.29 绝对定位水晶列表

不过，对于页面中局部模块定位来说，灵活应用相对定位和绝对定位，你会感觉操作起来很方便，没有浮动布局牵一发而动全身的问题。绝对定位所影响的仅仅是自己的位置，而不会对其他元素的位置产生影响。

5.4 设计导航菜单

在上一节中我们详细分析了使用项目列表构建导航条的基本框架，以及如何使用 CSS 控制

项目列表的显示。这些知识和操作技巧都是基础，只有你掌握了这些基础之后，才能够随心所欲的去设计导航条。

导航条的设计不应刻意追求炫目和超酷，只要实用即可，当然你可以根据个人设计能力和网站的需要适当的增加一些时尚元素和应用技巧。本节将介绍 3 个示例，分别从不同侧面和设计风格上探索导航条的设计技巧。

5.4.1 淡雅的垂直导航菜单

本示例是一个简单的垂直导航菜单。其设计风格淡雅、轻松（如图 5.30 所示的左侧栏），这是禅意花园中的一个作品（<http://www.csszengarden.com/?cssfile=208/208.css>），由于它设计精巧，使用技术不是很复杂，对于初学者学习来说值得认真学习和研究。



图 5.30 垂直导航菜单

该导航条的 HTML 结构与上节讲解的导航条项目列表结构基本相同，如下所示：

```
<div id="linkList">
  <div id="lresources">
    <h3 class="resources"><span>参考资源</span></h3>
    <ul>
      <li><a href="#">查看这个设计的样式表 CSS</a>
      <li><a href="#">CSS 参考资料</a>
      <li><a href="#">常见问题</a>
      <li><a href="#">投稿</a>
      <li><a href="#">翻译文件</a> </li>
    </ul>
  </div>
</div>
```

在项目列表 ul 元素外面包含了 2 层外套，<div id="linkList">表示整个网页的左侧栏目，<div id="lresources">是本示例要研究的模块。

好的设计习惯总会在每个模块前面增加一个栏目标题结构，虽然可能不会显示出来，但是你可以使用 CSS 隐藏标题信息，在上面的结构中我们可以看到设计师正是这样做的。

为了能够更精确的控制栏目的显示位置，首先定义<div id="linkList">包含框为绝对定位显示，这样你能够精确的控制其在页面中的位置和显示大小。


```
#linkList { /* 左侧通栏绝对定位 */
    position:absolute;          /* 绝对定位 */
    top:179px;                  /* 距离包含块的顶部距离, 这里指页面的顶部 */
    left:20px;                  /* 距离包含块左侧距离, 这里指网页左侧 */
    width:207px;                /* 固定栏目的总宽度 */
}
```

然后定义超链接的正常样式以及鼠标经过时的样式:

```
a {
    color:#D9189F;              /* 粉红色 */
    background-color:#ffffff;    /* 白色背景 */
    text-decoration:underline;   /* 清除下划线 */
}
a:hover {
    color:#FC7AD5;              /* 淡粉色 */
}
```

网页设计中,并不是每个细节都要创意十足,有时高超的设计师仅仅施点颜色,就会让页面效果立即焕然一新,你可以看到这里没有仅仅是把超链接的文本颜色由粉红色变成淡粉色,这样鼠标经过时,超链接文本仅仅淡淡的一亮,即恰当又文雅,在这个环境中比你用浓墨重彩要更动人。

然后为每个列表项定义虚线。提到定义虚线,也许你会想到 CSS 中 border 属性,但是作者在这里有意避开 border 属性,而是使用了一个背景图像。

```
#linkList li {
    list-style:none;            /* 清除项目列表符号 */
    padding:6px 0 10px 0;       /* 增加列表项上下空隙,使设计的菜单项看起来更大方 */
    background:url(images/images/line.gif) bottom repeat-x; /* 定义虚线背景,水平平铺 */
}
```

为什么作者如此避嫌而非要走“弯路”呢?

实际上这正是广大初学者要认真学习的一技之长。成熟的设计师往往会善于利用图像来装点网页,而且用得恰到好处,让人叹为观止。

使用背景图像绘制虚线的好处,就是它看起来显得更清晰,比 border 属性定义的虚线要好看得多,更为惊奇的是不同浏览器对于 border 的虚线属性的解析效果并不完全相同,特别是 IE 浏览器,显示的不匀称,极其难看。

当然,在用图像编辑器绘制虚线时,一定要精确到像素(如图 5.31 所示),一个虚线点就是一个像素,点与点间隔两个像素,点显示为浅灰色。这些设计不能够有一点差错,否则所设计的虚线就会很难看,反而不如 border 属性包含的虚线了。使用背景图像来设计虚线,你还可以有创意发挥的余地,使虚线按着你的意图来显示。例如,定义点与点之间的间隔距离,点的大小、宽圆,甚至可以设计各种花样虚点。



图 5.31 垂直导航菜单

再为列表项定义一个相符符号,这里采用了使用背景图像的方法来设计,因为它更容易控制。

```
#linkList li a {
    padding-left:7px;           /* 在列表项左侧挤出 7 个像素空间 */
    background:url(images/images/link.gif) left center no-repeat; /* 定义项目符号 */
    text-decoration:none;       /* 清除下划线 */
}
```

再定义当鼠标经过时为超链接增加下划线效果:

```
#linkList li a:hover {
    text-decoration:underline;
}
```

最后为该导航模块定义圆角显示效果, 具体方法不妨参考上一章圆角技术专题。

```
#lresources { /* 中间背景区域 */
    background:url(images/images/left_bg.gif) repeat-y; /* 背景图像垂直平铺 */
}
#lresources h3 { /* 顶部圆角 */
    background:url(images/images/title_resources.gif) no-repeat; /* 定义单块背景 */
}
#lresources ul { /* 底部圆角 */
    margin:0;
    padding:0 25px 20px 17px; /* 增加底部空隙 */
    background:url(images/images/left_bottom.gif) bottom no-repeat; /* 定义单块背景 */
}
```

5.4.2 巧妙的水平导航菜单

如果说上一节还意在强调创意的重要性, 那么本节示例将完全向你展示了如何使用 CSS 巧设导航条了。如图 5.32 所示, 这是一个纯 CSS 设计的导航菜单的效果, 其中充分利用了 CSS 的多种设计技巧, 当鼠标移过时, 菜单底部会显示一个悬挂的下三角, 导航条的色彩与整个页面的色彩协调一致, 显得醒目而又恰到好处。



图 5.32 水平导航菜单

首先, 在上节示例的基础上构建一个导航条结构, 该导航条的项目列表结构被放置在头部模块区域的底部, 从下面的结构你可以看到, 为了能够更好的控制每个菜单项, 列表中包含了多个辅助元素, 它们的作用是用来设计一些修饰性的导航效果。请注意, 这些辅助元素都包含在 a 元素内部, 否则你是无法在超链接的动态状态中进行控制的。

```
<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym> 设计之美</span></h2>
    <ul class="menu">
```



```

<li> <a href="#" "> <b><span>查看样式表 CSS</span></b><em></em> </a> </li>
<li> <a href="#" "> <b><span>CSS 参考资料</span></b><em></em> </a> </li>
<li> <a href="#" "> <b><span>常见问题</span></b><em></em> </a> </li>
<li> <a href="#" "> <b><span>投稿</span></b><em></em> </a> </li>
<li> <a href="#" "> <b><span>翻译文件</span></b><em></em> </a> </li>
</ul>
</div>

```

首先，我们把整个项目列表（ul 元素）设置为绝对定位显示，这样也能够更好的控制，毕竟我们这里所演示的示例都是以他人的页面为平台，目的是想尽力模拟真实的环境来研究导航条的设计。使用绝对定位不会破坏原来页面的结构，操作起来会省心许多。

```

.menu {
    position:absolute; /* 绝对定位 */
    top:120px; /* 坐标定位，距离包含块顶部的距离 */
    left:40px; /* 坐标定位，距离包含块左侧的距离 */
    padding:0; /* 清除默认的缩进样式 */
    margin:0; /* 清除默认的缩进样式 */
    list-style-type:none; /* 清除项目符号 */
    white-space:nowrap; /* 禁止文本换行 */
}

```

在上面的样式中，white-space:nowrap;声明表示强制文本在一行内显示，禁止换行显示，这在导航条中很有用，如果菜单文本比较长，在 IE 中很容易出现换行显示的问题（如图 5.33 所示）。

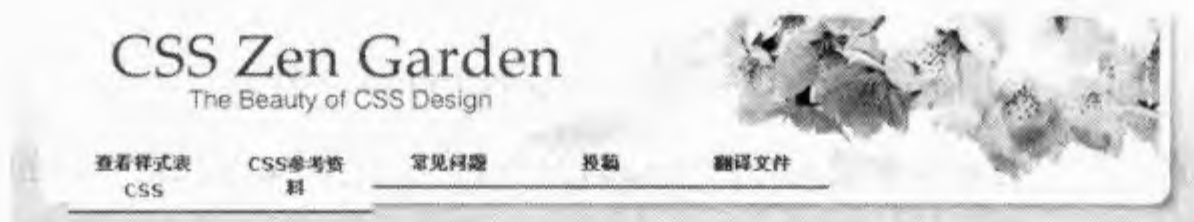


图 5.33 换行显示的导航条

这样会严重破坏整个页面的结构，为了安全起见，建议你对于导航条文本字数在无法预知的情况下增加该属性，这种做法会比较妥当。

然后以浮动法让列表项并列浮动显示，为了防止列表项宽度太窄，使用 min-width 属性限制其最窄显示宽度。

```

.menu li {
    float:left; /* 浮动显示 */
    min-width:100px; /* 最窄显示宽度 */
}

```

定义超链接元素 a 以块状显示，并清除默认的下划线，使其填充满列表项内的空间。

```

.menu a {
    position:relative; /* 定义包含块，为后面的绝对定位做坐标参考 */
    display:block; /* 块状显示 */
    text-decoration:none; /* 清除下划线 */
    min-width:100px; /* 最窄显示宽度 */
}
* html .menu a {
    width:100px; /* 兼容 IE 6 及以下版本不支持 min-width 属性 */
}

```

再为超链接元素内包含的 span 元素定义菜单正常显示样式。可能有的读者疑惑了：为什么

不直接在 a 元素上定义这些显示样式呢？这主要是因为 a 元素此时仅作为一个包含块，不再适合定义具体的显示样式。如果直接在 a 元素上定义每个列表项的显示样式，就无法设计出导航条底部的下划线效果（如图 5.34 所示）。

```
.menu a span {
    display: block;                /* 块状显示 */
    color: #F911B2;               /* 字体颜色 */
    background: #FFF4FC;          /* 背景颜色 */
    border: solid #fff;            /* 增加边框 */
    border-width: 0 2px 2px 2px;   /* 定义边框宽度 */
    text-align: center;            /* 居中显示 */
    padding: 4px 16px;             /* 增加内边距 */
    cursor: pointer;               /* 定义鼠标指针以手形显示 */
    min-width: 66px;              /* 最窄显示宽度 */
}
* html .menu a span {            /* 兼容 IE 6 及以下版本 */
    width: 100px;                 /* 限制最低宽度 */
    cursor: hand;                 /* 定义鼠标指针以手形显示 */
    width: 66px;                 /* 兼容 IE 6 版本 */
}
```



图 5.34 直接在 a 元素上定义每个列表项的样式

利用 a 元素内包含的 b 元素定义导航条底部的下划线，并隐藏 a 元素包含的 em 元素。

```
.menu a b {
    display: block;                /* 块状显示 */
    border-bottom: 2px solid #F911B2; /* 绘制导航条的下划线 */
}
.menu a em {
    display: none;                 /* 隐藏 em 元素 */
}
```

下一步是最关键的。在这里我们利用 em 元素绘制鼠标经过超链接时绘制向下箭头，同时改变列表项背景色为洋红色，设置菜单项字体颜色为白色。

```
.menu a: hover {
    background: #fff;              /* 经过超链接背景色变为白色 */
}
.menu a: hover span {
    color: #fff;                   /* 经过超链接 span 字体颜色变为白色 */
    background: #F911B2;          /* 经过超链接 span 背景色变为洋红色 */
}
.menu a: hover em {
    display: block;                /* 绘制鼠标经过时向下三角形 */
    overflow: hidden;              /* 块状显示 */
    border-style: solid;           /* 隐藏超出制定宽度和高度的区域 */
    border-color: #F911B2 #fff;    /* 实变边框 */
    border-width: 6px 6px 0 6px;   /* 边框颜色 */
    height: 3px;                  /* 边框宽度 */
}
/* em 元素的高度，截取下部分，这样就会显示上半部分
的三角形 */
```



```

position:absolute;          /* 绝对定位，好准确定位到菜单项底部中间 */
left:50%;                   /* 水平居中 */
margin-left:-6px;           /* 通过取左边界负值以实现三角形真正居中 */
}

```

很多读者可能一直无法理解，CSS 所定义的元素都是以盒模型（矩形）显示的，怎么会显示为三角形呢？

为了说明这个问题，我们做一个简单的试验，把上面 `em` 元素的样式放大显示。请输入下面样式以及标签：

```

<style type="text/css">
em {
    display:block;
    overflow:hidden;
    border-style:solid;
    border-color:#F911B2 #33FF00;
    border-width:100px;
    width:3px;
    height:3px;
}
</style>

```

如果此时你在浏览器中预览，前面的所有疑惑现在都会豁然开朗。原来当为元素定义边框时，如果每条边显示不同颜色，则每条边之间通过 45° 角斜切平分，这样就形成了三角形，如图 3.35 所示。

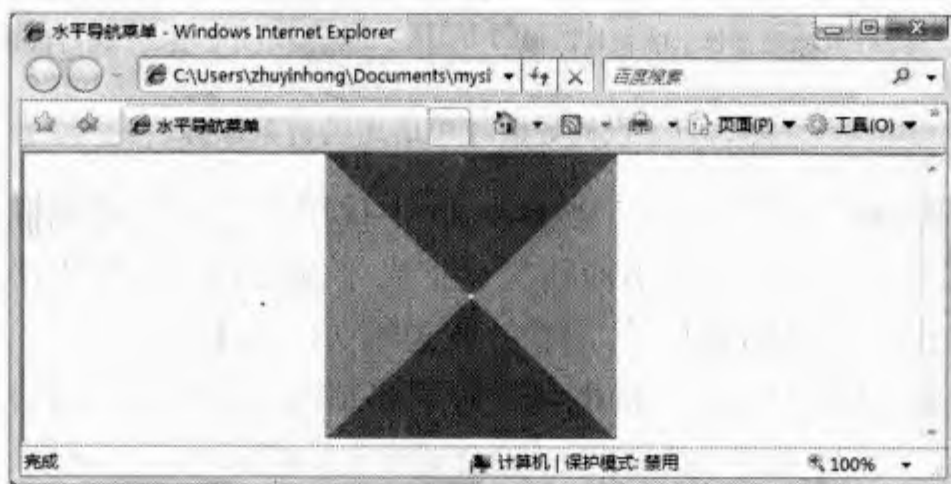


图 5.35 用 CSS 绘制三角形

5.4.3 背景图像在导航菜单中的应用

如果说完全使用 CSS 来设计各种特效还有点卖弄技术的嫌疑，但是设计师更多的时候会以务实的态度来进行设计。所谓务实就是在追求相同的设计效果前提下，那种方法比较简单、快捷和安全就会采用那种方法。

虽然设计师普遍不排斥各种卖弄技术的做法，但是在激烈的竞争环境中是不能够容忍拖工、怠工等现象发生，考虑到使用 CSS 设计相同的效果会冒更大的兼容风险，所以多选择使用图像来代替各种特效是比较稳妥的做法。当然对于初学者来说，还应鼓励去探索各种新奇技术，这对于深入掌握 CSS 是非常有帮助的。

也许对于上一节的示例使用背景图像会非常简单，但是我们还是把它作为一个专题进行讲解。本节将转换话题，探索如何使用背景图像设计更酷的导航条。

如图 5.36 所示就是本节示例将要讲解的内容，这是一个看起来很酷的导航条，利用背景图

像来设计的。目前很多 Web 2.0 网站的导航条基本上都采用这种形式,它显得醒目且有立体感,还具有 Vista 系统的超酷效果。



图 5.36 使用背景图像设计导航菜单

设计这样的导航条最大的难点是如何在图像编辑器中设计背景图像? 图像设计得越形象,自然所设计的导航条也就越逼真。本示例使用了 3 个背景图像,如图 5.37 所示。



图 5.37 设计导航菜单所用的背景图像

在设计这些背景图像时,你可以在 Photoshop 中使用渐变工具绘制椭圆条形状,然后利用选取工具分别选取上下部分,分别应用曲线工具把上半部分调亮,把下半部分调暗即可,当然你还可以利用图层样式,并借助叠加等功能完善这种立体效果。

下面来构建该导航条的 HTML 结构框架,为了控制背景图像,在项目列表中包含了一个辅助元素 b。

```
<ul class="menu">
  <li><a href="#"><b>查看样式表 CSS</b></a></li>
  <li><a href="#"><b>CSS 参考资料</b></a></li>
  <li><a href="#"><b>常见问题</b></a></li>
  <li class="current"><a href="#nogo"><b>投稿</b></a></li>
  <li><a href="#"><b>翻译文件</b></a></li>
</ul>
```

搭建好结构之后,下面就可以使用 CSS 来控制导航条的显示效果了。首先还是为 ul 列表元素进行定位,并设置基本的属性:

```
.menu {
  position: absolute;          /* 绝对定位 */
  top: 164px;                /* 定位, 距离顶部距离 */
  left: 20px;                /* 定位, 距离左侧距离 */
  width: 748px;              /* 固定宽度 */
  padding: 0 0 0 1em;        /* 在导航条左侧增加 1 个字大小的间距 */
  margin: 0;                 /* 清除默认缩进样式 */
  list-style: none;          /* 清除项目符号 */
}
```



```

height:35px;          /* 固定导航条高度 */
background:url(images/bg3.gif); /* 定义导航条的背景图像 */
}

```

然后让列表项浮动显示，实现水平显示：

```

.menu li {
    float:left;
}

```

定义菜单项的显示样式，设置超链接以块状显示，并固定大小，设置菜单文本显示属性：

```

.menu li a {
    display:block;          /* 块状显示 */
    float:left;             /* 向左浮动 */
    height:35px;            /* 与导航条同高 */
    line-height:33px;       /* 垂直对齐文本 */
    color:#FFF00;           /* 设置粉红色字体颜色 */
    text-decoration:none;    /* 清除默认样式下划线 */
    font-family:arial, verdana, sans-serif; /* 字体属性 */
    text-align:center;       /* 水平对齐文本 */
    padding:0 0 0 14px;      /* 增加左侧空隙 */
    cursor:pointer;          /* 定义手形鼠标指针 */
    font-size:11px;          /* 字体大小 */
}

```

同时设置 a 元素所包含的辅助元素也为块状浮动显示，这样为后面进行控制提供保障。

```

.menu li a b {
    float:left;             /* 向左浮动 */
    display:block;          /* 块状显示 */
    padding:0 28px 0 14px;   /* 增加左右两侧的内边距 */
}

```

定义当前菜单的显示样式，所谓当前菜单就是被激活的菜单，也就是当前页面为当前菜单指向的链接。定义当前菜单样式是为了更好的区分菜单状态。

```

.menu li.current a {
    color:#fff;              /* 白色字体 */
    background:url(images/left3.gif); /* 定义当前菜单的背景图像 */
}
.menu li.current a b { /* 定义当前菜单的背景图像 */
    background:url(images/left3.gif) no-repeat right top;
}

```

上面分别在 a 元素和 b 元素中同时定义相同的背景图像。读者可能会迷糊了，这是不是多此一举呢？实际上这是一个 CSS 设计技巧，利用背景图像重叠来伪造圆角按钮效果。如果我们仅定义 a 元素的背景图像，这时你会看见当前按钮右侧显示为直角效果，而不是与左侧对应的圆角（如图 5.38 所示）。



图 5.38 仅定义一个背景图像的效果

通过为两个重合的元素定义相同的背景图像，一个从左侧开始向右侧延伸，另一个从右侧向左侧平铺。由于是背景图像，中间多余的区域会被自动隐藏，这样就给人一种错觉，所显示的按钮是圆角效果（演示示意图如图 5.39 所示）。

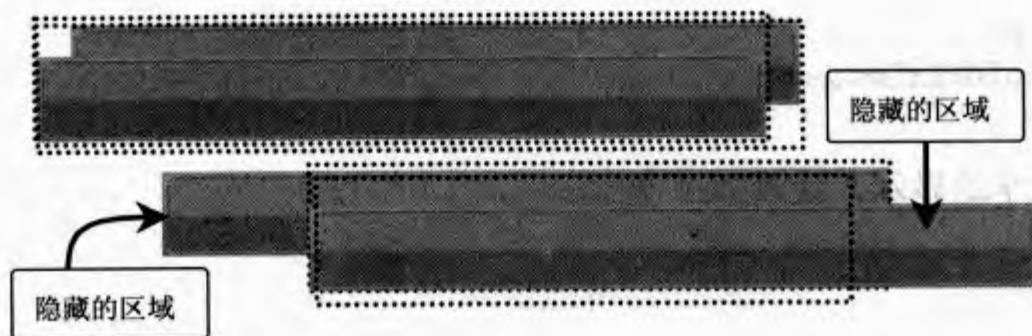


图 5.39 圆角背景图像的设计示意图

最后再定义鼠标移过时的样式。设计思路与上面的方法相同，就不再重复说明。

```
.menu li.current a:hover {
    color:#fff; /* 白色字体 */
    background: url(images/right3.gif); /* 定义鼠标移过的背景图像 */
    cursor:default; /* 定义手形鼠标样式 */
}
.menu li.current a:hover b {
    background:url(images/right3.gif) no-repeat right top;
}
```

5.4.4 实用的多级菜单

多级菜单技术在网页中经常被用到，也是比较实用的技术。一般设计师都喜欢使用于 JavaScript+CSS 技术组合来实现多级菜单。对于超酷的多级菜单更是离不开 JavaScript 技术来捕获用户的行为，并控制样式表的调用，使用 JavaScript 还可以控制各种动态特效。

不过你也可以使用纯 CSS 技术来设计多级菜单。它主要利用超链接的 4 种鼠标移动的状态来决定下拉菜单的显示或隐藏。然后把下拉菜单包含在 a 元素中，这样就可以通过鼠标操作来决定是否显示或隐藏子菜单。

下面我们介绍一个简单的示例，演示如何利用 CSS 技术，并适当借助 JavaScript 属性事件来触发多级菜单的显示和隐藏，示例演示效果如图 5.40 所示。



图 5.40 多级菜单演示效果

首先，在前面演示示例模板基础上，构建一个多级菜单结构。


```

<ul id="nav">
  <li class="menu2" onMouseOver="this.className='menu1'" onMouseOut="this.className='menu2'"><a href="#">查看样式表 CSS</a>
    <ul class="list">
      <li><a href="#">子菜单 1</a></li>
      <li><a href="#">子菜单 2</a></li>
      <li><a href="#">子菜单 3</a></li>
    </ul>
  </li>
  <li class="menu2" onMouseOver="this.className='menu1'" onMouseOut="this.className='menu2'"><a href="#">CSS 参考资料</a>
    <ul class="list">
      <li><a href="#">子菜单 1</a></li>
      <li><a href="#">子菜单 2</a></li>
    </ul>
  </li>
  <li class="menu2" onMouseOver="this.className='menu1'" onMouseOut="this.className='menu2'"><a href="#">常见问题</a>
    <ul class="list">
      <li><a href="#">子菜单 1</a></li>
    </ul>
  </li>
  <li class="menu2" onMouseOver="this.className='menu1'" onMouseOut="this.className='menu2'"><a href="#">投稿</a> </li>
  <li class="menu2" onMouseOver="this.className='menu1'" onMouseOut="this.className='menu2'"><a href="#">翻译文件</a> </li>
</ul>

```

这是一个嵌套的二级项目列表，外层项目列表负责组织主菜单，内层项目列表负责管理子菜单，当然你也可以在此基础上进一步的扩展。该项目列表结构插入在示例模板的头部区域的底部，具体位置可以参阅光盘示例（5-36.html）。

然后为该导航菜单结构定义样式表。样式表设计的第一步是定位导航条。

```

#nav {
  position: absolute;          /* 绝对定位 */
  z-index: 1; /* 定义层叠顺序，避免页面中部的层（左侧导航条）覆盖本菜单 */
  left: 0px;                  /* 左侧距离 */
  top: 124px;                  /* 顶部距离 */
  width: 700px;                /* 宽度 */
  height: 30px;                /* 高度 */
  padding: 0px 4px;            /* 增加左右内边距 */
}
html>/**/body #nav { /* 兼容非 IE 浏览器 */
  left: 40px;                  /* 左侧距离 */
  top: 112px;                  /* 顶部距离 */
}

```

这里使用了一个兼容技术，选择符 `html>/**/body` #nav 能够限制该样式仅在非 IE 浏览器中被解析。由于 IE 浏览器与非 IE 浏览器在解析复杂定位时（包含多个包含块的时候），会存在判断标准的不同，从而导致解析结果存在很大差异。例如，针对本示例如果不使用这个兼容技术，则在非 IE 浏览器中解析效果如图 5.41 所示。你可以看到导航条跑到页面左侧边上了，非常难看。下面是清除项目列表和超链接的默认样式。

```

#nav ul { margin: 0px; padding: 0px; } /* 清除缩进 */
#nav li a { text-decoration: none; } /* 清除下划线 */

```



图 5.41 标准浏览器在解析时存在的差异

让列表项浮动并列显示，并定义主菜单的显示样式。

```
#nav li {
    list-style:none;           /* 清除项目符号 */
    text-align:center;        /* 居中对齐 */
    font-weight:bold;         /* 加粗显示 */
    float:left;               /* 清除项目符号 */
}
```

再定义下拉子菜单的显示样式。

```
#nav .list {
    line-height:20px;         /* 行高 */
    text-align:left;          /* 左对齐 */
    padding:2px;              /* 内边距 */
    font-weight:normal;       /* 正常字体，不加粗显示 */
}
#nav .list a {
    color:#FF3AC1;            /* 下拉子菜单中超链接字体颜色 */
    text-decoration:none;     /* 清除下划线 */
    float:left;               /* 浮动超链接显示，这样可以定义宽和高 */
    width:100px;              /* 超链接的宽度 */
    padding:3px 5px 0px 5px;  /* 内边距 */
}
```

然后再定义鼠标经过子菜单时显示的样式。

```
#nav .list a:hover {
    color:white;              /* 字体颜色 */
    padding:3px 3px 0px 20px; /* 内边距 */
    width:88px;              /* 宽度 */
    background-color:#FF3AC1; /* 背景色 */
}
```

最后，定义两个类样式，设计当鼠标经过和离开主菜单时所要应用的样式。

```
#nav .menu1 {
    width:120px;              /* 主菜单的宽度 */
    height:auto;              /* 主菜单的高度，自动 */
    margin:6px 4px 0px 0px;   /* 增加外边距 */
    border:1px solid #FF3AC1; /* 设置一个边框 */
    background-color:#F1FBEC; /* 定义背景色 */
    color:#FF3AC1;           /* 字体颜色 */
    padding:6px 0px 0px 0px;  /* 定义内边距 */
    cursor:hand;              /* 定义鼠标样式，手形 */
    overflow-y:hidden;        /* 隐藏 y 轴超出的区域 */
}
```



```

        filter:Alpha(opacity=70);
        -moz-opacity:0.7;
    }
    #nav .menu2 {
        width:120px;
        height:18px;
        margin:6px 4px 0px 0px;
        background-color:#F5F5F5;
        color:#999999;
        border:1px solid #EEE8DD;
        padding:6px 0px 0px 0px;
        overflow-y:hidden;
        cursor:hand;
    }

```

定义完毕两个类样式之后，你就可以在主菜单标签中属性事件中引用这两个类样式即可。

```

<li class="menu2" onMouseOver="this.className='menu1'" onMouseOut="this.className='menu2'">

```

5.4.5 时尚的滑动门菜单

在上面示例中我们曾就背景图像在导航菜单中的应用进行过说明，考虑到此类导航菜单比较实用，这里再举一个典型示例，专门研究如何设计滑动门导航菜单。

所谓滑动门就是当使用背景图像设计导航菜单时，背景图像能够适应菜单的宽度（文本字数的多少）不断调整自身的宽度，它犹如滑动的门一样，可以开得窄一点，也可以开得宽一点。设计滑动门一般至少需要两个标签配合使用，然后通过推拉另一个标签的宽度来适应不同的菜单宽度，该标签的背景图像犹如一道门一样，随着标签的宽和高不断伸缩。

本示例演示效果如图 5.42 所示。在图中你可以看到，不同的菜单由于字数的不同，所显示的宽度也截然不同，如果分别为不同宽度的菜单设计背景图像是非常麻烦的。但是利用滑动门技术，你只需要设计两个图像即可，滑动门都能够根据菜单的宽度来调整所显示的背景图像的宽度。



图 5.42 滑动门导航菜单

要完成本示例的设计，设计的第一步是设计滑动门所需要的特大背景图像，我们曾经在图 5.39 中演示过两个背景图像重叠设计的一般方法。实际上滑动门的设计原理与背景图像重叠应

用是相同的。当菜单列表项伸展之后，活动的一扇背景图像跟随着自动伸展；当菜单列表项收缩之后，活动的一扇背景图像跟随着自动收缩，但要注意，其一侧的背景图像始终是固定的（如图 5.43 所示）。

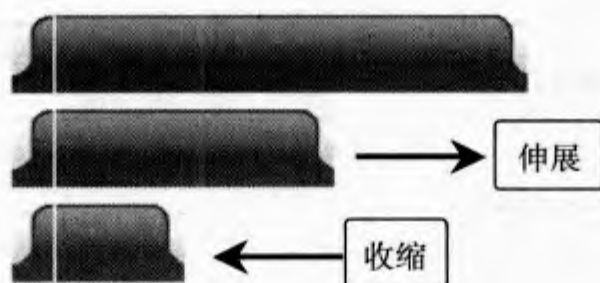


图 5.43 滑动门导航菜单

基于上面的设计思路，滑动门需要最少两个标签来配合，所以本示例的结构如下。

```
<ul class="menu">
  <li class="current"> <a href="#"> <b>查看样式表 CSS</b> </a> </li>
  <li> <a href="#"> <b>CSS 参考资料</b> </a> </li>
  <li> <a href="#"> <b>常见问题</b> </a> </li>
  <li> <a href="#"> <b>投稿</b> </a> </li>
  <li> <a href="#"> <b>翻译文件</b> </a> </li>
</ul>
```

仿照前面的示例把菜单列表固定在网页头部区域，并设置大小和其他相关属性。

```
.menu {
  position: absolute;          /* 绝对定位 */
  top: 120px;                 /* 纵轴坐标 */
  left: 20px;                 /* 横轴坐标 */
  height: 46px;               /* 高度 */
  width: 722px;               /* 宽度 */
  padding: 0 0 0 38px;        /* 增加左侧内边距，避免滑动门靠近最左侧显示 */
  margin: 0 auto;             /* 居中 */
  background: url(images/bg2.gif) repeat-x; /* 导航条背景图像 */
  list-style: none;           /* 清除项目符号 */
}
.menu li {
  float: left;                /* 靠左浮动，实现菜单项并列显示 */
}
```

下面是滑动门技术实现的关键，通俗地说就是设计滑动门所需要的门框。这里设计 a 元素以块状显示，并固定其大小，宽度可以自适应，不用设置，否则就不是滑动门了。然后把 a 元素包含的 b 元素也定义为块状显示，并拥有相同的大小。为了防止 b 元素伸到 a 元素的最左侧，还应该为 a 元素左侧设置一个内边距，挤出一点空隙用来显示左侧背景图像，即固定大小的背景图像，所挤出的宽度应该与左侧背景图像的宽度一致。

```
.menu li a {
  float: left;                /* 靠左浮动 */
  display: block;              /* 块状显示，将拥有宽和高等属性 */
  color: #FF04B7;              /* 字体颜色 */
  text-decoration: none;        /* 清除下划线 */
  font-family: sans-serif;      /* 字体 */
  font-size: 12px;              /* 字体大小 */
  padding: 0 0 0 16px;          /* 在左侧挤出一点空隙备用 */
  height: 46px;                 /* 高度 */
  line-height: 46px;            /* 垂直居中 */
}
```



```

text-align: center;          /* 水平居中 */
cursor: pointer;             /* 手形鼠标样式 */
}
.menu li a b {
float: left;                 /* 浮动显示 */
display: block;              /* 块状显示, 将拥有宽和高等属性 */
padding: 0 24px 0 8px;       /* 平衡左右内侧空隙 */
}

```

最后, 分别在鼠标经过时设置 a 元素和 b 元素的背景图像, 同时定义当前菜单的背景图像。要注意背景图像的对齐方向不同, 一个是向左对齐, 另一个是向右对齐。

```

.menu li.current a, .menu li a:hover {
color: #fff;                 /* 白色字体 */
background: url(images/left2.gif) no-repeat; /* 背景图像 */
background-position: left;   /* 左对齐 */
}
.menu li.current a b, .menu li a:hover b {
color: #fff;                 /* 白色字体 */
background: url(images/right2.gif) no-repeat right top; /* 右对齐的背景图像 */
}

```

第 6 章

更专业的表格和表单

表格和表单是网页中两个重要的元素。在传统网页设计中设计师完全依赖表格的功能进行布局，如今标准设计抛弃了这种设计思维，表格又重新活跃起来，专职于数据的组织和管理。表单是网页交互设计的基础，动态网站必须借助表单来实现用户与服务器的数据交互，自然设计专业的表单能够提高网页交互的效率和用户体验。本章将介绍如何使用 CSS 来控制表格和表单的显示样式，并结合用户体验介绍如何设计更具专业的网页布局。

6.1 用好表格

在短短的网页设计发展历程中，表格的地位变化最为明显。最初创建表格是用来显示数据，后来表格被借用实现网页布局，利用表格的行和列来定位网页中各种对象，如今 CSS 取代了表格布局的功能，于是表格又恢复到最初的数据管理功能。

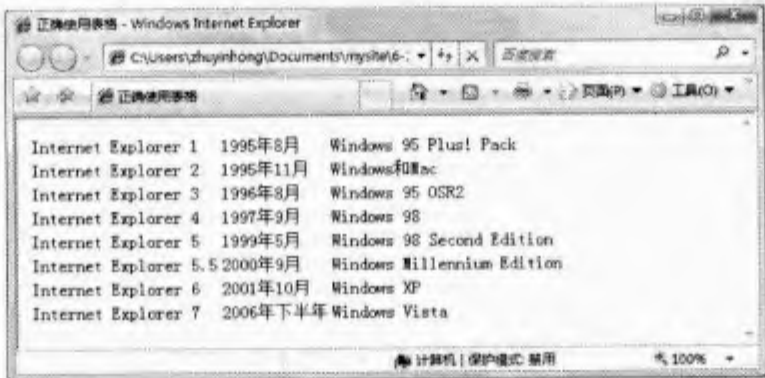
表格简洁明了，数据管理安全、高效。作为重要的网页设计元素，一直受到网页设计师的青睐。在标准设计中表格虽然不再用来布局，但是在组织和管理数据方面，依然是最有效的工具。

6.1.1 正确使用表格

显示数据无疑使用表格是最安全的，它不会像 CSS 布局那样会遇到很多意想不到的 Bug，使用表格时，用户只需要联合使用<table>、<tr>和<td>三个标签即可。

其中<table>标签是负责包含所有数据（数据表）的外框，类似于标准布局中包含框（<div id="wrap">）；<tr>负责包含数据行的外框，类似于标准布局中的子包含框（<div id="container">）；而<td>标签负责对最小数据单元的控制，相当于标准布局中的最小栏目块。

例如，下面这个示例是一个 8 行 3 列的数据表，该表显示 IE 浏览器 12 年发展中每个版本、发布时间和捆绑的操作系统相关联的数据。在没有任何格式的情况显示效果如图 6.1 所示。



Internet Explorer 1	1995年8月	Windows 95 Plus! Pack
Internet Explorer 2	1995年11月	Windows和Mac
Internet Explorer 3	1996年8月	Windows 95 OSR2
Internet Explorer 4	1997年9月	Windows 98
Internet Explorer 5	1999年5月	Windows 98 Second Edition
Internet Explorer 5.5	2000年9月	Windows Millennium Edition
Internet Explorer 6	2001年10月	Windows XP
Internet Explorer 7	2006年下半年	Windows Vista

图 6.1 IE 浏览器 12 年发展历史数据表


```

<table>
  <tr>
    <td>Internet Explorer 1</td>
    <td>1995 年 8 月</td>
    <td>Windows 95 Plus! Pack</td>
  </tr>
  <tr>
    <td>Internet Explorer 2</td>
    <td>1995 年 11 月</td>
    <td>Windows 和 Mac</td>
  </tr>
  <tr>
    <td>Internet Explorer 3</td>
    <td>1996 年 8 月</td>
    <td>Windows 95 OSR2</td>
  </tr>
  <tr>
    <td>Internet Explorer 4</td>
    <td>1997 年 9 月</td>
    <td>Windows 98</td>
  </tr>
  <tr>
    <td>Internet Explorer 5</td>
    <td>1999 年 5 月</td>
    <td>Windows 98 Second Edition</td>
  </tr>
  <tr>
    <td>Internet Explorer 5.5</td>
    <td>2000 年 9 月</td>
    <td>Windows Millennium Edition</td>
  </tr>
  <tr>
    <td>Internet Explorer 6</td>
    <td>2001 年 10 月</td>
    <td>Windows XP</td>
  </tr>
  <tr>
    <td>Internet Explorer 7</td>
    <td>2006 年下半年</td>
    <td>Windows Vista</td>
  </tr>
</table>

```

但是如果不借助表格，要相显示如图 6.1 所示的数据表，你可能需要构建类似如下的结构。

```

<div class="table">
  <ul>
    <li>Internet Explorer 1</li>
    <li>1995 年 8 月</li>
    <li>Windows 95 Plus! Pack</li>
  </ul>
  <ul>
    <li>Internet Explorer 2</li>
    <li>1995 年 11 月</li>
    <li>Windows 和 Mac</li>
  </ul>
  <ul>
    <li>Internet Explorer 3</li>
    <li>1996 年 8 月</li>
    <li>Windows 95 OSR2</li>
  </ul>
  <ul>
    <li>Internet Explorer 4</li>
    <li>1997 年 9 月</li>
    <li>Windows 98</li>
  </ul>
  <ul>
    <li>Internet Explorer 5</li>
    <li>1999 年 5 月</li>
    <li>Windows 98 Second Edition</li>
  </ul>
  <ul>
    <li>Internet Explorer 5.5</li>
    <li>2000 年 9 月</li>
    <li>Windows Millennium Edition</li>
  </ul>
  <ul>
    <li>Internet Explorer 6</li>
    <li>2001 年 10 月</li>
    <li>Windows XP</li>
  </ul>
  <ul>
    <li>Internet Explorer 7</li>
    <li>2006 年下半年</li>
    <li>Windows Vista</li>
  </ul>
</div>

```

```
<li>Internet Explorer 3</li>
<li>1996 年 8 月</li>
<li>Windows 95 OSR2</li>
</ul>
<ul>
  <li>Internet Explorer 4</li>
  <li>1997 年 9 月</li>
  <li>Windows 98</li>
</ul>
<ul>
  <li>Internet Explorer 5</li>
  <li>1999 年 5 月</li>
  <li>Windows 98 Second Edition</li>
</ul>
<ul>
  <li>Internet Explorer 5.5</li>
  <li>2000 年 9 月</li>
  <li>Windows Millennium Edition</li>
</ul>
<ul>
  <li>Internet Explorer 6</li>
  <li>2001 年 10 月</li>
  <li>Windows XP</li>
</ul>
<ul>
  <li>Internet Explorer 7</li>
  <li>2006 年下半年</li>
  <li>Windows Vista</li>
</ul>
</div>
```

从代码量的角度比较，表格与非表格显示数据似乎没有什么区别。但是我们在浏览器中预览（如图 6.2 所示）。

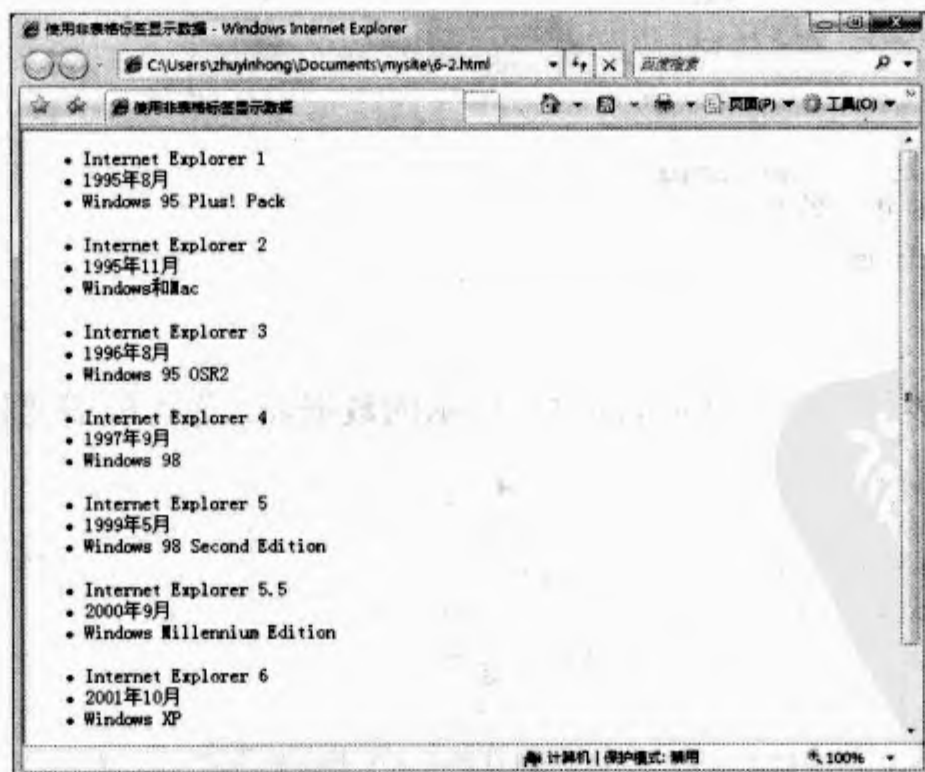


图 6.2 使用非表格标签显示数据

如果要显示为表格的行列效果，则需要定义如下样式来模拟数据表的多行多列效果（如图 6.3 所示）。


```
<style type="text/css">
div.table ul {/* 定义项目默认样式 */
    list-style-type:none;          /* 清除项目列表符号 */
    margin:0;                      /* 清除列表项缩进 */
    padding:0;                    /* 清除列表项缩进 */
    overflow:auto;                /* 自适应高度 */
    zoom:1;                      /* 定义在 IE 浏览器中能够自适应高度 */
}
div.table li {/* 定义列表项浮动显示 */
    float:left;
}
</style>
```



图 6.3 CSS 布局来模拟表格效果

从上图中你也可以看到这种模拟方法存在很大的问题，即行内数据都挤在一起，不适合阅读。为此还必须为 li 元素增加一个宽度。

```
div.table li {
    float:left;
    width:200px;
}
```

但是，这种方法比较机械，且宽度都是相等（如图 6.4 所示）。要想精确控制每列的宽度，除非为每列定义一个类样式，然后再应用到这些标签中。

此外，还有一个潜在的危险：就是当浏览器窗口变化时，这些浮动的 li 元素可能会出现错行现象。解决的方法就是为<div class="table">外包含框定义一个固定的宽度。

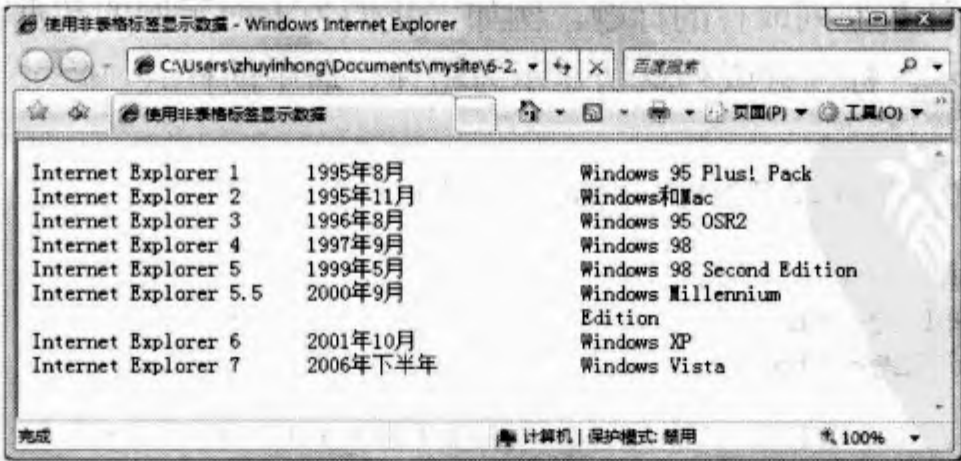


图 6.4 CSS 布局来模拟表格效果

```
div.table ul {
    width:600px;
}
```

经过这么一比较，你是不是感觉使用表格来组织和管理数据不仅方便而且高效？所以，当你准备学习和使用 CSS 来设计网页布局时，不要完全抛弃表格的功能，并充分发挥它在数据显示方面的优势，那样你才会体验到网页设计的乐趣。

6.1.2 优化数据表格的结构

`<table>`、`<tr>`和`<td>`是数据表格的三个基本标签，但是为更有利于搜索引擎的阅读，在标准设计中我们还应该对数据表格的结构进行优化，以提升表格的语义化水平。这里主要包括下面这些标签。

使用`<caption>`标签为数据表格定义一个标题。该标签可以放在`<table>`和`</table>`标签之间的任意位置。一般习惯把它放在`<table>`标签内的首行，如图 6.5 所示。

```
<table>
  <caption>IE 浏览器 12 年发展大事记</caption>
  <tr>
    <td>Internet Explorer 1</td>
    <td>1995 年 8 月</td>
    <td>Windows 95 Plus! Pack</td>
  </tr>
  .....
</table>
```



图 6.5 设计数据表的标题效果

使用`<th>`标签设计数据列或行的标题。例如，我们为上面示例的数据表增加一个标题列，显示效果如图 6.6 所示。标题行或列默认显示为粗体、居中的样式。

```
<table>
  <caption>IE 浏览器 12 年发展大事记</caption>
  <tr>
    <th>版本</th>
    <th>发布时间</th>
    <th>绑定系统</th>
  </tr>
  <tr>
    <td>Internet Explorer 1</td>
    <td>1995 年 8 月</td>
    <td>Windows 95 Plus! Pack</td>
  </tr>
  .....
</table>
```


版本	发布时间	绑定系统
Internet Explorer 1	1995年8月	Windows 95 Plus! Pack
Internet Explorer 2	1995年11月	Windows和Mac
Internet Explorer 3	1996年8月	Windows 95 OSR2
Internet Explorer 4	1997年9月	Windows 98
Internet Explorer 5	1999年5月	Windows 98 Second Edition
Internet Explorer 5.5	2000年9月	Windows Millennium Edition
Internet Explorer 6	2001年10月	Windows XP
Internet Explorer 7	2006年下半年	Windows Vista

图 6.6 设计数据表的列标题效果

如果要设置行标题，只需要调整标题标签的位置即可。例如，设置第1列数据为标题行，则可以进行如下修改，显示效果如图 6.7 所示。

```
<table>
  <caption>IE 浏览器 12 年发展大事记</caption>
  <tr>
    <th>版本</th>
    <th>发布时间</th>
    <th>绑定系统</th>
  </tr>
  <tr>
    <th>Internet Explorer 1</th>
    <td>1995 年 8 月</td>
    <td>Windows 95 Plus! Pack</td>
  </tr>
  <tr>
    <th>Internet Explorer 2</th>
    .....
  </tr>
  .....
</table>
```

版本	发布时间	绑定系统
Internet Explorer 1	1995年8月	Windows 95 Plus! Pack
Internet Explorer 2	1995年11月	Windows和Mac
Internet Explorer 3	1996年8月	Windows 95 OSR2
Internet Explorer 4	1997年9月	Windows 98
Internet Explorer 5	1999年5月	Windows 98 Second Edition
Internet Explorer 5.5	2000年9月	Windows Millennium Edition
Internet Explorer 6	2001年10月	Windows XP
Internet Explorer 7	2006年下半年	Windows Vista

图 6.7 设计数据表的行标题效果

除了数据表、行和列标题之外，我们还可以使用<thead>、<tbody>和<tfoot>标签来对数据进行分组。分组的目的是方便数据显示样式控制，以及更利于搜索引擎的阅读。另外使用<colgroup>和<col>标签为数据列进行分组。

例如，下面把整个数据表进行纵横分组，其中每列为一组，对应名称为 version、postTime 和 OS，然后把列标题作为一组，定义为头部区域（<thead>），中间数据区域为主体区域（<tbody>），并增加一个页脚行区域（<tfoot>）。

```

<table>
  <caption>IE 浏览器 12 年发展大事记</caption>
  <colgroup>
    <col id="version" />
    <col id="postTime" />
    <col id="OS" />
  </colgroup>
  <thead>
    <tr>
      <th>版本</th>
      <th>发布时间</th>
      <th>绑定系统</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Internet Explorer 1</th>
      <td>1995 年 8 月</td>
      <td>Windows 95 Plus! Pack</td>
    </tr>
    .....
  </tbody>
  <tfoot>
    <tr>
      <td colspan="3">页脚区域</td>
    </tr>
  </tfoot>
</table>

```

6.1.3 设计表格样式

表格具有强大的布局功能，同理它也拥有强大的样式自控能力。在没有 CSS 之前，设计师借助表格自身包含的各种属性也能够完成表格样式的设计。其实如果在 Dreamweaver CS3 中利用可视化操作的【属性】面板，你可以快速设置表格的显示样式（如图 6.8 所示）。



图 6.8 在 Dreamweaver CS3 中设置表格样式

(1) 数据表格的边框样式。数据表一般都需定义边框，以便能很好区分数据的行和列，<table>标签提供了 border 属性来定义边框粗细，默认为 0，即不显示边框线，还可以使用 bordercolor 属性定义边框的颜色。不过，使用 CSS 来设计数据表的边框会更灵活，选择的余地也更大。

例如，输入如下样式则可以为表格定义一个外框（如图 6.9 所示）。

```

<style type="text/css">
table {
  border:solid 1px red;
}
</style>

```

如果要为每行和每列都定义边框，则需要同时为 th 和 td 也定义边框，代码如下，显示效果如图 6.10 所示。


```
<style type="text/css">
th, td {
    border:solid 1px red;
}
</style>
```



图 6.9 table 边框效果



图 6.10 th 和 td 边框效果

此时你会发现表格之间显示双线框，这是因为每个单元格都被定义了边框，为此 CSS 提供了 border-collapse 属性，利用该属性可以合并单元格边框线，不过该属性只能够定义到 table 标签上才能够有效，设置效果如图 6.11 所示。

```
<style type="text/css">
table {
    border:solid 1px red;
    border-collapse:collapse;
}
th, td {
    border:solid 1px red;
}
</style>
```

/* 合并单元格边框线 */



图 6.11 CSS 设置表格边框样式效果

(2) 给数据表格定义边距。由于表格是由几个标签组成，每个标签都对应特定的 CSS 属性，正如边框合并属性 border-collapse 只能够定义到 table 标签上一样。但是给 table 标签定义 padding 属性就没有作用。不过 <table> 标签提供 cellpadding 属性来定义单元格的内部边距的大小，而使用 cellspacing 属性可以定义单元格外边距的大小。标准设计中建议读者使用 CSS 来定义单元格的边距，定义的样式如下：

```
<style type="text/css">
th, td {
    border:solid 1px red;
    padding:10px;
    margin:10px;
}
</style>
```

(3) 数据表格的对齐。初学者对于 CSS 的对齐功能不是很满意，使用起来感觉特别费劲。但是在表格中设置会很轻松（关于 CSS 对齐的话题可以参阅第 3 章详细讲解）。例如：

1) 设置数据表格居中显示：

```
<table align="center">
```

2) 设置单元格包含对象水平居中：

```
<td align="center">1997 年 9 月</td>
```

3) 设置单元格包含对象垂直居中：

```
<td valign="middle">1997 年 9 月</td>
```

使用 CSS 定义表格居中需要兼顾不同浏览器，设置样式如下：

```
body {
    text-align:center;
}
table {
    border-collapse:collapse;
    margin:0 auto;
}
```

如果要设置单元格文本居中，则可以直接为 td 对象定义 text-align:center; 声明，而对于单元格垂直居中，则可以使用如下样式：

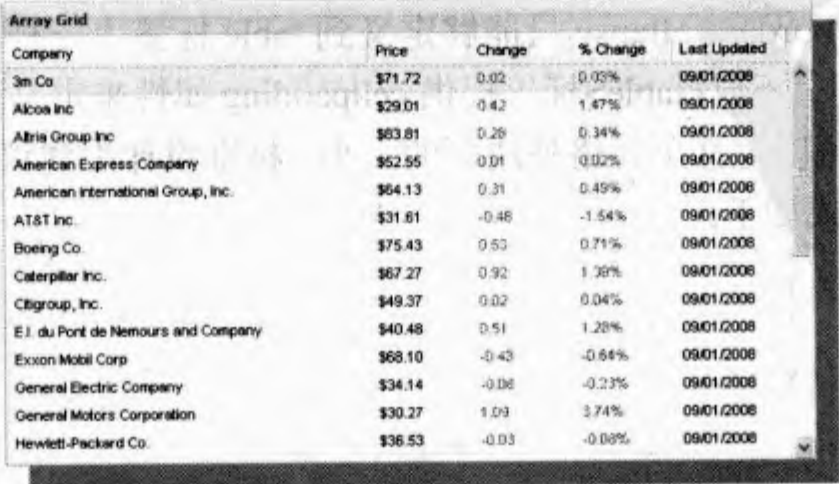
```
th, td {
    vertical-align:middle;
}
```

而对于其他 CSS 样式，可以根据正常方法设置即可，这里就不再详细讲解。

6.2 表格样式设计实战

表格主要用来显示数据，但是大量的数据被放在一起就很容易影响浏览者对数据的阅读，特别是很容易造成视觉错位和视觉疲倦，从而产生读错行的现象。因此，当我们为数据表进行设计时，要充分考虑到浏览者的这种实际需求，一般可以利用隔行分色、隔列分色或动态行等方法来避免阅读中的读错行现象。

例如，如图 6.12 所示的是 Ext JS 框架所提供的一套数据表格的样式模板，它通过隔行浅色背景，并辅助动态行效果来引导浏览者快速阅读。



Array Grid				
Company	Price	Change	% Change	Last Updated
3m Co	\$71.72	0.02	0.03%	09/01/2008
Alcoa Inc	\$29.01	-0.42	-1.47%	09/01/2008
Altria Group Inc	\$63.81	0.26	0.34%	09/01/2008
American Express Company	\$52.55	0.01	0.02%	09/01/2008
American International Group, Inc.	\$64.13	0.31	0.49%	09/01/2008
AT&T Inc	\$31.61	-0.46	-1.54%	09/01/2008
Boeing Co	\$75.43	0.53	0.71%	09/01/2008
Caterpillar Inc.	\$67.27	0.92	1.36%	09/01/2008
Citigroup, Inc.	\$49.37	0.02	0.04%	09/01/2008
E.I. du Pont de Nemours and Company	\$40.48	0.51	1.26%	09/01/2008
Exxon Mobil Corp	\$68.10	-0.43	-0.64%	09/01/2008
General Electric Company	\$34.14	-0.06	-0.23%	09/01/2008
General Motors Corporation	\$30.27	1.09	3.74%	09/01/2008
Hewlett-Packard Co.	\$36.53	-0.03	-0.08%	09/01/2008

图 6.12 Ext JS 的数据表样式效果

6.2.1 设计表格行样式

隔行分色是数据表最经典的设计样式，这种样式主要还是从用户易用性来考虑的，以提高用户扫描数据的速度和准确度。

隔行分色的设计方法是：定义一个类，然后把该类应用到所有奇数行或偶数行。例如，我们先定义一个隔行背景色的类样式，并为列标题定义一个类样式，代码如下：

```
<style type="text/css">
table { /* 表格默认样式 */
    border:solid 1px #99CCFF;          /* 表格外框线 */
    border-collapse:collapse;          /* 合并单元格边框线 */
}
.bg_th {/* 标题行样式类 */
    background:#0000FF;                /* 背景色 */
    color:#fff;                        /* 字体颜色 */
}
.bg_even {/* 隔行样式类 */
    background:#99CCFF;                /* 隔行背景色 */
}
</style>
```

然后把其中的标题行样式类和隔行样式类应用到数据表中，局部代码如下，其中标题行被应用了 bg_th 类，而数据区域的偶数行被应用了 bg_even 类。显示效果如图 6.13 所示。

```
<table>
  <thead>
    <tr class="bg_th">
      <th>版本</th>
      <th>发布时间</th>
      <th>绑定系统</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Internet Explorer 1</th>
      <td>1995 年 8 月</td>
      <td>Windows 95 Plus! Pack</td>
    </tr>
    <tr class="bg_even">
      <th>Internet Explorer 2</th>
      <td>1995 年 11 月</td>
      <td>Windows 和 Mac</td>
    </tr>
    .....
  </tbody>
</table>
```



图 6.13 设计隔行变色的样式效果

6.2.2 设计表格列样式

相对于行样式来说，列样式的设计要简单许多，因为可以借助<colgroup>和<col>标签来对列进行分组并设计样式。

例如，我们定义如下几个类样式，然后分别应用到<col>列标签中，则显示效果如图 6.14 所示。

```
<style type="text/css">
table { /* 表格默认样式 */
    border:solid 1px #99CCFF;
    border-collapse:collapse;
}
.bg_th { /* 标题行类样式 */
    background:#0000FF;
    color:#fff;
}
.bg_even1 { /* 列1 类样式 */
    background:#CCCCFF;
}
.bg_even2 { /* 列2 类样式 */
    background:#FFFFCC;
}
</style>
<table>
    <caption>IE 浏览器 12 年发展大事记</caption>
    <colgroup>
        <col class="bg_even1" id="version" />
        <col class="bg_even2" id="postTime" />
        <col class="bg_even1" id="OS" />
    </colgroup>
    .....
</table>
```



图 6.14 设计隔列变色的样式效果

6.2.3 设计鼠标经过时表格行的样式

如果长时间盯着一个大数据表格，会使人很累。设计鼠标经过时数据行显示不同样式，这样可以调节人的视觉注意力和紧张度。如果你使用过超链接样式就会深有同感。利用 CSS 提供的伪类选择符: hover 可以实现这样的设计效果，样式代码如下，但是这种方法不被 IE 6 及其以下版本支持，因此最安全的方法是采用 JavaScript 来实现。

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



```
tr:hover {/* 鼠标经过时的样式 */
    background:#99CCFF;           /* 背景色 */
    cursor:pointer;               /* 鼠标样式 */
}
</style>
```

使用 JavaScript 可以设计鼠标经过数据行时的动态效果, 所设计的样式会更为灵活。首先定义一个 JavaScript 函数:

```
<script language="javascript">
//bg_even("表格 ID 属性名","奇数行背景色","偶数行背景色","鼠标经过背景色","点击后背景色");
function bg_even(o,a,b,c,d){
    //获取对数据行的控制
    var t=document.getElementById(o).getElementsByTagName("tr");
    for(var i=0;i<t.length;i++){ //遍历数据表中每一行
        //判断数据行的奇偶位置, 分别设置不同的背景色
        t[i].style.backgroundColor=(t[i].sectionRowIndex%2==0)?a:b;
        //定义鼠标单击事件函数, 设计背景色的单击开关效果
        t[i].onclick=function(){
            if(this.x!="1"){//如果没有单击, 则设置单击背景色
                this.x="1";
                this.style.backgroundColor=d;
            }else{//如果已经单击, 则恢复原来的背景色
                this.x="0";
                this.style.backgroundColor=(this.sectionRowIndex%2==0)?a:b;
            }
        }
        //定义鼠标经过事件函数, 设计鼠标经过行的背景色效果
        t[i].onmouseover=function(){
            if(this.x!="1")this.style.backgroundColor=c;
        }
        //定义鼠标离开事件函数, 设计鼠标离开行的背景色效果
        t[i].onmouseout=function(){
            if(this.x!="1")this.style.backgroundColor=(this.sectionRowIndex%2==0)?a:b;
        }
    }
}
</script>
```

然后为数据表<table>标签定义一个 ID 值。

```
<table id="grid"></table>
```

最后, 在网页末尾引用 JavaScript 脚本函数。显示效果如图 6.15 所示。



图 6.15 利用 JS 脚本设计鼠标经过行的动态样式效果

```
<script language="javascript">
bg_even("grid", "#fff", "#F5F5F5", "#FFFFCC", "#FFFF84");
</script>
```

6.3 用好表单

表单是用户交互不可或缺的基本元素，如何设计出操作方便、视觉整洁、悦目的表单一直是设计师工作的重点。本节将详细讲解如何使用 CSS 来控制表单的显示样式。

6.3.1 认识表单

表单是一个集合概念，其中包含了很多个元素，例如，文本框、单选按钮、复选框、下拉菜单和按钮等，不过这些基本元素都必须包含在表单元素 form 中。你可以在 Dreamweaver CS3 中快速插入各种表单元素，然后再借助 CSS 来控制它们的显示。

例如，一个没有经过格式化的表单效果如图 6.16 所示。在这个表单中插入了常用表单元素。



图 6.16 没有格式化的表单效果

```
<form id="form1" name="form1" method="post" action="">
  <p> 文本框:
    <input type="text" name="textfield" id="textfield" />
  </p>
  <p>文本区域:
    <textarea name="textarea" id="textarea" cols="45" rows="5"></textarea>
  </p>
  <p>复选框:
    a<input type="checkbox" name="checkbox" id="checkbox" />
    b<input type="checkbox" name="checkbox2" id="checkbox2" />
    c<input type="checkbox" name="checkbox3" id="checkbox3" />
  </p>
  <p>单选按钮:
    a<input type="radio" name="radio" id="radio" value="radio" />
    b<input type="radio" name="radio2" id="radio2" value="radio2" />
    c<input type="radio" name="radio3" id="radio3" value="radio3" />
  </p>
  <p>下载菜单:
    <select name="select" id="select">
      <option value="1">a</option>
      <option value="2">b</option>
      <option value="3">c</option>
    </select>
  </p>
  <p>
```



```



```

一个合格的表单不仅要语法严谨,同时还要考虑用户的使用习惯,甚至要照顾一些特殊用户,如近视者、手脚不方便者等。为了追求精致的风格,很多设计师喜欢把字体设置为 12px,表单中的文本框、单选按钮、复选框也都被设计得很小巧,对于视力良好的人来说输入信息没有什么问题,但如果是视力不好的人使用这些小巧的表单元素恐怕就是一种折磨了。

如果当用户想选中并填写表单选项时,可以有三种选择:

- 直接单击文本框等元素对象本身。
- 直接单击文本框前面的提示文本。
- 利用 Tab 功能键和提示文本后的字母来实现快捷键选中文本框。

读者可能觉得操作设置好的表单如同操作桌面 Windows 应用软件的感觉,这就对了,这样我们就为用户营造了一个比较便利的交互环境,那些手脚不灵便,或是不习惯使用笔记本上触摸板的人一定非常方便。

为此 HTML 还提供了 3 个辅助标签: <fieldset>、<legend>和<label>。利用这些标签能够提高表单的易用性。例如,对于上面的表单结构,我们可以进行进一步的优化。

```

<form id="form1" name="form1" method="post" action="">
  <fieldset>
    <legend>设计易用性表单</legend>
    <p>
      <label for="textfield">文本框: </label>
      <input type="text" name="textfield" id="textfield" />
    </p>
    <p>
      <label for="textarea">文本区域: </label>
      <textarea name="textarea" id="textarea" cols="45" rows="5"></textarea>
    </p>
    <p>复选框:
      <label for="checkbox">a</label>
      <input type="checkbox" name="checkbox" id="checkbox" />
      <label for="checkbox2">b</label>
      <input type="checkbox" name="checkbox2" id="checkbox2" />
      <label for="checkbox3">c</label>
      <input type="checkbox" name="checkbox3" id="checkbox3" />
    </p>
    <p>单选按钮:
      <label for="radio">a</label>
      <input type="radio" name="radio" id="radio" value="radio" />
      <label for="radio2">b</label>
      <input type="radio" name="radio2" id="radio2" value="radio2" />
      <label for="radio3">c</label>
      <input type="radio" name="radio3" id="radio3" value="radio3" />
    </p>
    <p>
      <label for="select">下载菜单: </label>
      <select name="select" id="select">
        <option value="1">a</option>
        <option value="2">b</option>
        <option value="3">c</option>
      </select>
    </p>
  </fieldset>
</form>

```

```

        </select>
    </p>
    <p>
        <input type="submit" name="button" id="button" value="提交" />
        <input type="reset" name="button2" id="button2" value="重置" />
    </p>
    </fieldset>
</form>

```

在上面代码中我们利用<fieldset>和<legend>标签来组织表单，以方便管理，然后利用<label>标签来设置每个表单元素的提示信息，这样能够保证提示信息与对应的表单对象紧密联系在一起，在浏览器中演示效果如图 6.17 所示。

在使用<label>标签时可以使用 for 属性与对应的表单对象进行绑定，这样当用户操作时就可以快速实现。

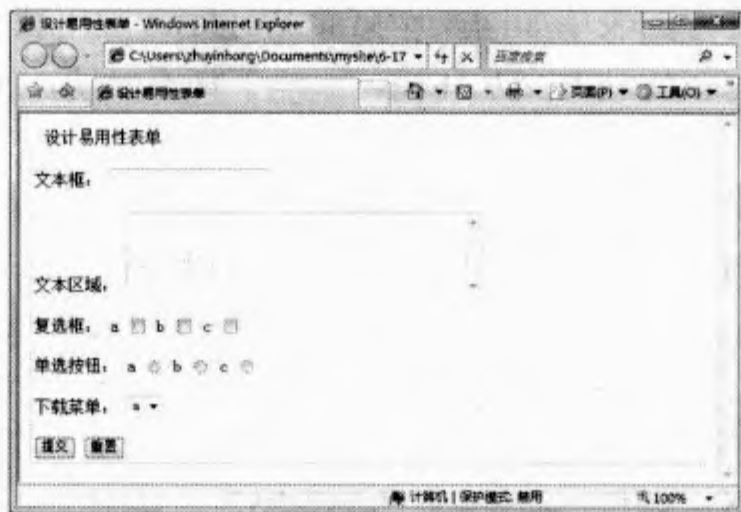


图 6.17 设计易用性表单效果

6.3.2 设计表单样式

使用 CSS 控制表单样式与其他元素没有什么区别，你可以直接为任意一个表单对象定义样式。但是由于大部分表单对象都是 input 元素定义的，且表单名称被用来传递数据的句柄而用，因此建议读者采用类来定义表单样式。

从图 6.17 所示的表单效果中可以看到：整个表单仅似一个“毛坯房”，既不美观也不适合使用，下面我们要做的工作就是如果使用 CSS 来控制表达的显示样式。

首先，我们把左侧的提示信息进行对齐处理。尺寸不一的提示文本、文本框、单选按钮和其他表单对象使整个表单看起来很难看，也很难阅读。为此我们可以定义<label>标签，统一提示文本的长度并自动右对齐（如图 6.18 所示）。



图 6.18 设计表单提示文本的效果


```

<style type="text/css">
.title {/* 表单对象提示文本的类样式 */
    width:100px;           /* 宽度 */
    float:left;           /* 向左浮动 */
    text-align:right;      /* 文本右对齐 */
    font-weight:bold;      /* 加粗提示文本 */
}
</style>

```

然后定义一个文本居中类，实现设计按钮居中显示。

```

<style type="text/css">
.center {
    text-align:center;      /* 宽度 */
}
</style>
<p class="center">
    <input type="submit" name="button" id="button" value="提交" />
    <input type="reset" name="button2" id="button2" value="重置" />
</p>

```

再定义表单元素 form 居中显示，并定义该表单包含的 fieldset 元素居中对齐、宽度和文本左对齐，效果如图 6.19 所示。

```

#form1 {
    text-align:center;      /* 定义表单内对象居中显示 */
}
#form1 fieldset {
    width:500px;           /* 定义表单区域宽度 */
    margin:0 auto;         /* 文本对象居中显示 */
    text-align:left;       /* 文本左对齐 */
}

```



图 6.19 设计表单整体效果

最后，定义表单中包含的各个元素对象的基本样式。当然可以根据具体页面的整体设计效果来设计所需要的表单效果（如图 6.20 所示）。

```

#form1 #textfield {
    width:16em;             /* 文本框样式 */
    border:solid 1px #aaa;  /* 文本框的宽度 */
    font-size:14px;         /* 文本框的边框样式 */
    color:#666;            /* 字体大小 */
    position:relative;      /* 字体颜色 */
                        /* 相对定位 */
}

```

```

        top:-3px;                                /* 向上移动位置 */
    },
    #form1 #textarea {                          /* 文本区域样式 */
        width:30em;                             /* 文本区域宽度 */
        height:8em;                             /* 高度 */
        border:solid 1px #aaa;                  /* 边框样式 */
        font-size:12px;                         /* 字体大小 */
        color:#666;                             /* 字体颜色 */
    }
    .checkbox {                                     /* 复选框样式类 */
        border:solid 1px #fff;                  /* 边框样式 */
        position:relative;                     /* 相对定位 */
        top:3px;                                /* 向下移动位置 */
        left:-2px;                              /* 向左移动位置 */
    }
    #radio {                                     /* 单选按钮样式 */
        border:solid 1px #fff;                  /* 边框样式 */
        position:relative;                     /* 相对定位 */
        top:3px;                                /* 向下移动位置 */
        left:-1px;                              /* 向左移动位置 */
    }
}

```

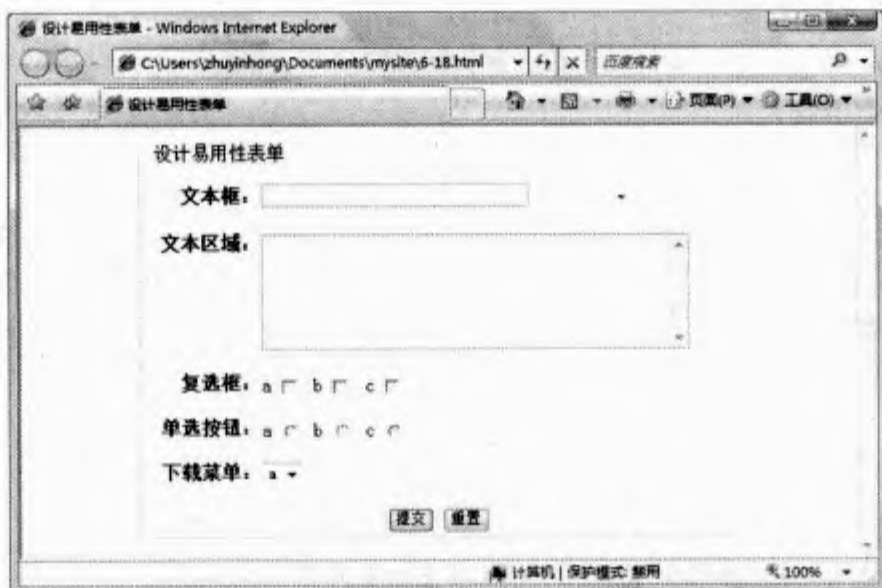


图 6.20 设计表单对象的整体效果

在定义表单对象的样式时，你应该注意如下几点问题：

第一，定位文本框或文本区域的宽度建议使用 `em` 作为单位，这样能够准确计算一行能够接纳的字符数。

第二，通过相对定位来移动原始位置，这样可以避免因表单对象的特殊性而无法与同行的文本居中对齐问题。定义相对定位之后，就可以通过 `top` 和 `left` 属性来设置表单对象的偏移位置。取正值表示向下或向右位移，取负值表示向上或向左移动。有关相对定位的讲解请参阅第 8 章内容。

第三，设置单选按钮和复选框边框为白色边框，目的是使其与背景色相融合，这样就可以设计出具有个性的单选按钮和复选框效果。

6.4 表单样式设计实战

网页都是利用表单来处理数据输入和设置，但是并不是所有的表单都保持一致。输入区域

的对齐方式、表单对象的标签、表单操作方式以及周围的视觉元素都会或多或少的影响用户的数据输入行为。表单需要一个稳固的视觉结构、完整的层次组成表单元素以及强大的技术(Ajax)使表单更加易用并看上去更加漂亮。下面我们讲解如何使用 CSS 来控制表单的表现效果。

6.4.1 设计高亮显示当前表单样式

高亮提示当前表单是网页中常见的一种表单样式设计效果。一般借助 JavaScript 脚本来进行控制。本示例借助 JavaScript 脚本语言属性事件来侦测用户的操作,并分别在表单获取焦点和焦点时分别为该表单对象调用不同的样式类,从而实现当前表单高亮显示效果(如图 6.21 所示)。

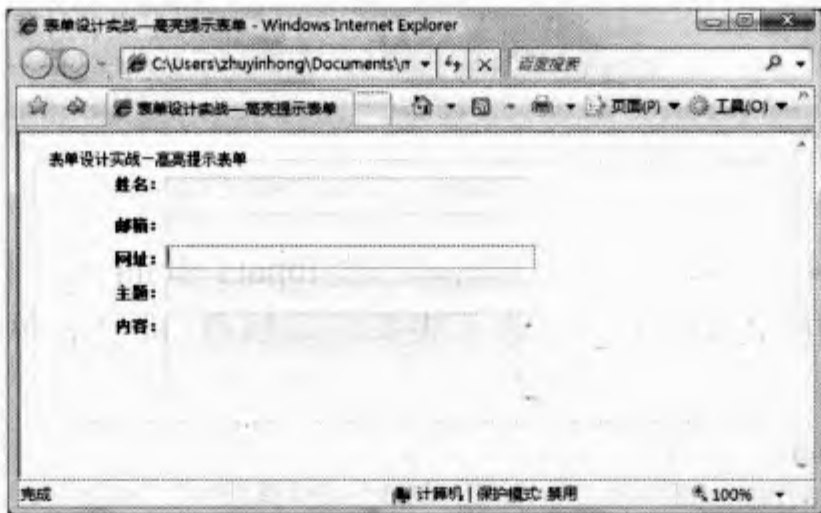


图 6.21 高亮提示表单效果

首先,我们构建一个表单结构,该结构是在模仿上面示例的基础上重新定义的。结构代码如下:

```
<form id="form1" name="form1" method="post" action="">
  <fieldset>
    <legend>表单设计实战—高亮提示表单</legend>
    <label for="name" class="title">姓名: </label>
    <input size="40" name="name" class="input1">
    <label for="email" class="title">邮箱: </label>
    <input size="40" name="email" class="input1">
    <label for="url" class="title">网址: </label>
    <input size="40" name="url" class="input1">
    <label for="subject" class="title">主题: </label>
    <input size="40" name="subject" class="input1">
    <label for="message" class="title">内容: </label>
    <textarea name="message" cols="39" rows="5" class="input1" ></textarea>
  </fieldset>
</form>
```

定义 CSS 样式控制表单的显示,同时定义两个类,以供 JavaScript 属性事件调用。

```
<style type="text/css">
body {/* 统一网页字体大小 */
  font-size:12px;
}
.title {/* 提示文本样式类 */
  width:100px;
  float:left;
  text-align:right;
  font-weight:bold;
  /* 固定宽度 */
  /* 向左浮动定位 */
  /* 文本右对齐 */
  /* 字体加粗 */
}
```

```

margin:6px 0;
}
/* 定义提示文本的外边距 */
.input1 {
background-color: #EEEEEE;
border-bottom: #FFFFFF 1px solid;
border-left: #CCCCCC 1px solid;
border-right: #FFFFFF 1px solid;
border-top: #CCCCCC 1px solid;
}
/* 表单默认显示样式类 */
/* 浅白色字体 */
/* 底边框样式 */
/* 左边框样式 */
/* 右边框样式 */
/* 顶边框样式 */
.input2 {
background-color:#F0F8FF;
border: 1px solid #999;
}
/* 表单获取焦点时的样式类 */
/* 加深背景色 */
/* 加黑边框 */
input, textarea {
display:block;
margin:6px 0;
}
/* 定义各种表单对象的默认样式 */
/* 块状显示 */
/* 定义外边距 */
}
</style>

```

最后利用 `onblur` 和 `onfocus` 属性事件分别调用类 `input1` 和 `input2`。其中 `onblur` 属性事件表示失去焦点，即默认显示的样式类；`onfocus` 表示表单获取焦点，即当表单被选中时显示的样式类。显示局部代码如下：

```

<form id="form1" name="form1" method="post" action="">
  <fieldset>
    <legend>表单设计实战—高亮提示表单</legend>
    <label for="name" class="title">姓名: </label>
    <input size="40" name="name" class="input1" onblur="this.className='input1'"
onfocus="this.className='input2'">
    .....
  </fieldset>
</form>

```

6.4.2 设计图标样式的表单效果

在一些精制网页中，表单被设计为各种显示样式。本节将讲解如何把一个外部图像固定到表单对象的左侧，以点缀页面表单使其更好看，演示效果如图 6.22 所示。

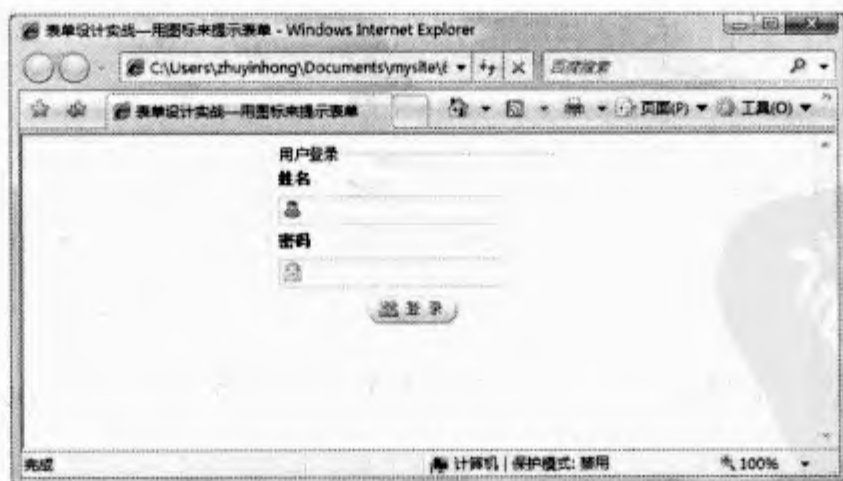


图 6.22 图标样式的表单效果

首先，构建一个表单结构。该结构依然保留上面示例的结构雏形，并适当的进行增删，代码如下：

```

<div id="login">
  <fieldset>

```



```

<legend>用户登录</legend>
<form action="" method="POST" class="form">
  <label for="name">姓名</label>
  <div>
    <input name="name" type="text" id="name" value="" />
  </div>
  <label for="password">密码</label>
  <div>
    <input name="password" type="text" id="password" value="" />
  </div>
  <div class="button_div">
    <input type="image" src="images/login.gif" />
  </div>
</form>
</fieldset>
</div>

```

然后使用 CSS 对这个表单进行布局，代码如下：

```

<style type="text/css">
* { /* 清除所有元素的边距 */
  margin:0;          /* 清除内边距 */
  padding:0;         /* 清除外边距 */
}
body { /* 定义网页基本属性 */
  text-align:center;  /* 网页居中显示 */
}
#login { /* 表单包含框样式 */
  margin:10px auto 10px; /* 网页居中显示 */
  text-align:left;       /* 文本左对齐 */
}
fieldset { /* 表单域样式 */
  width:230px;          /* 固定表单的宽度 */
  margin:200px;         /* 定义表单的外边距 */
  font-size:12px;       /* 统一表单的字体大小 */
}
</style>

```

如果让表单对象的提示文本能够与表单对象上下排列显示，就定义标签元素为块状元素，并定义它们的宽度为固定显示。详细代码如下：

```

label { /* 定义标签提示文本的样式 */
  width:200px;          /* 固定宽度 */
  height:26px;          /* 固定高度 */
  line-height:26px;     /* 固定行高 */
  text-indent:6px;      /* 文本首行缩进 6 像素 */
  display:block;        /* 块状显示 */
  font-weight:bold;     /* 加粗提示文本 */
}

```

最后我们来定义表单对象的样式，首先利用分组来统一所有表单对象的样式，然后再利用背景图像单独为每个文本框左侧定义一个图标。为了避免文本框内的文本遮盖背景图像图标，需要定义左侧内边距以挤出一个空间给背景图像留用。详细代码如下：

```

#name, #password { /* 统一表单对象的样式 */
  border:1px solid #ccc; /* 定义表单对象的边框样式 */
  width:160px;         /* 固定宽度 */
  height:22px;         /* 固定高度 */
}

```

```

margin-left:6px;           /* 定义左侧外边距 */
padding-left:20px;         /* 定义左侧内边距, 挤出定义背景图像的空间 */
line-height:20px;         /* 定义行高 */
}
#name { /* 用户名文本框图标样式 */
    background:url(images/name.gif) no-repeat 2px 2px; /* 定义用户名文本框图标 */
}
#password { /* 用户密码文本框图标样式 */
    background:url(images/password.gif) no-repeat 2px 2px; /* 定义密码文本框图标 */
}
.button_div { /* 按钮样式 */
    text-align:center;      /* 按钮文本居中 */
    margin:6px auto;        /* 按钮居中显示 */
}

```

6.4.3 设计易用性表单效果

下面我们来研究另一个问题, 即如何设计表单与表单对象的提示文本之间的关系, 以使用户能够快速进行填写, 缩短用户操作的时间, 提高用户的易用性。

首先, 我们建立一个表单框架。在这个框架中使用<div id="leftSide">标签做一个包含框, 然后利用表单中的字段集(表单框架)来组织所有表单对象, 并使用标签元素<label>来绑定提示文本。详细代码如下:

```

<div id="leftSide">
    <fieldset>
        <legend>易用性表单设计</legend>
        <form action="" method="POST" class="form">
            <label for="name">姓名</label>
            <div class="div_textbox">
                <input name="name" type="text" class="textbox" id="name" value="" />
            </div>
            <label for="address">地址</label>
            <div class="div_textbox">
                <input name="address" type="text" class="textbox" id="address" value="" />
            </div>
            <label for="city">区/县</label>
            <div class="div_textbox">
                <input name="city" type="text" class="textbox" id="city" value="" />
            </div>
            <label for="country">省/市</label>
            <div class="div_textbox">
                <input name="country" type="text" class="textbox" id="country" value="" />
            </div>
            <div class="button_div">
                <input name="Submit" type="button" value="Submit" class="buttons" />
            </div>
        </form>
    </fieldset>
</div>

```

如果在没有 CSS 支持的情况, 则显示如图 6.23 所示的效果。

考虑到用户完成表单填写的时间应当尽可能的缩短, 并且收集的数据都是用户所熟悉的信息, 如姓名、地址、城市等, 垂直对齐的标签和输入框可以说是最佳的。每对标签和输入框垂直对齐给人一种两者接近的感觉, 并且一致的左对齐减少了眼睛移动和处理时间。用户只需要

往一个方向移动。但是,这种垂直布局的表单效果对于不熟悉表单内容的用户来说,可能会存在操作上的困难,因为他需要在提示文本和表单对象上进行跳跃,操作起来感觉很不方便。这时你可以考虑使用左对齐的标签,这样用户就可以很容易的通览表单的信息。用户只需要上下看看左侧一栏标签就可以了,而不会被输入框打断思路。但这样一来,标签与其对应的输入框之间的距离通常会被更长的标签拉长,可能会影响填写表单的时间。用户必须左右来回的跳转目光来找到两个对应的标签和输入框。为此我们可以使用 CSS 设计提示文本右对齐(如图 6.24 所示)。

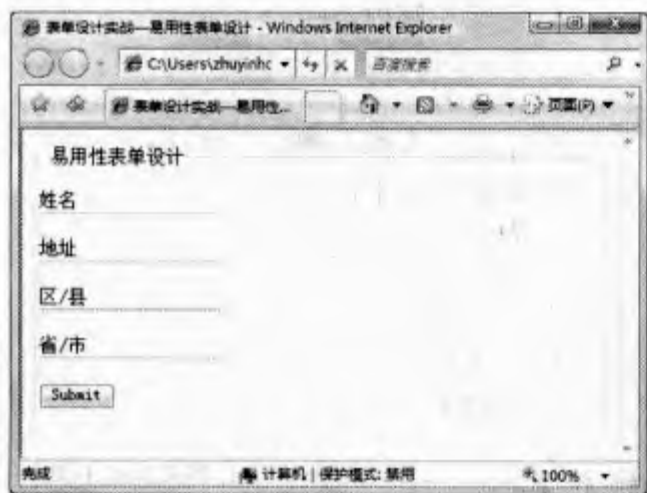


图 6.23 没有 CSS 支持的表单效果

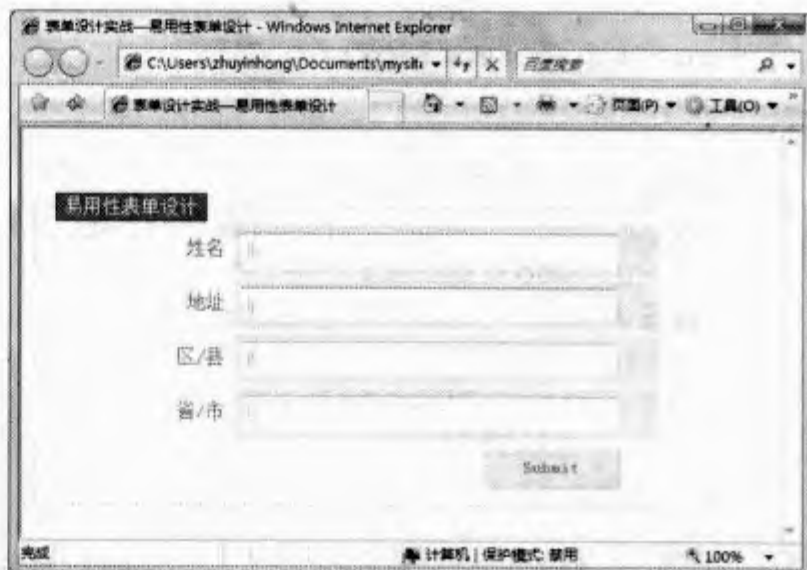


图 6.24 使用 CSS 设计水平对齐的表单对象和表单提示文本

上面表单样式的设计代码如下:

```
<style type="text/css">
body { /* 网页默认属性 */
    background-color:#F7F7F7; /* 定义网页背景色 */
}
fieldset { /* 表单字段级样式 */
    border:1px dashed #CCC; /* 定义虚线框 */
    padding:10px; /* 增加内边距 */
}
legend { /* 表单字段级的标题样式 */
    font-family:Arial, Helvetica, sans-serif; /* 字体属性 */
    color:#fff; /* 字体颜色 */
    background: #666; /* 背景颜色 */
    border: 1px solid #333; /* 边框样式 */
    padding: 2px 6px; /* 内边距 */
}
label { /* 提示标签样式 */
    width:142px; /* 固定宽度 */
    height:32px; /* 固定高度 */
    margin:3px 2px 0 0; /* 定义外边距 */
    padding:11px 0 0 6px; /* 定义内边距 */
    float:left; /* 向左浮动显示 */
    color:#666; /* 字体颜色 */
    text-align:right; /* 文本右对齐 */
}
.form { /* 表单样式类 */
    margin:0; /* 清除表单外边距 */
    padding:0; /* 清除表单内边距 */
}
#leftSide { /* 表单包含框样式 */
```

```

width:530px;
padding-top:30px;
float:left;
}
/* 固定宽度 */
/* 增加顶部内边距 */
/* 向左浮动 */

.div_tinbox {/* 文本框对象的外框样式 */
width:347px;
float:right;
background-color:#E6E6E6;
height:35px;
margin-top:3px;
padding-top:5px;
padding-bottom:3px;
padding-left:5px;
}
/* 固定宽度 */
/* 向右浮动 */
/* 背景色 */
/* 固定高度 */
/* 定义顶部外边距 */
/* 定义顶部内边距 */
/* 定义底部内边距 */
/* 定义左侧内边距 */

.textbox {/* 文本框样式 */
background-image: url(images/16t.gif);
background-repeat: no-repeat;
background-position:left;
width:285px;
font:normal 18px Arial;
color: #999999;
padding:3px 5px 3px 19px;
}
/* 定义文本框左侧提示图标*/
/* 禁止重复平铺 */
/* 固定位置 */
/* 固定宽度 */
/* 字体属性 */
/* 字体颜色 */
/* 内边距 */

.username:focus, .username:hover {/* 鼠标经过时的样式 */
background-color:#F0FFE6;
}
/* 背景色 */

.button_div {/* 按钮包含框的样式 */
width:287px;
float:right;
text-align:right;
height:35px;
margin-top:3px;
padding:5px 32px 3px;
}
/* 固定宽度 */
/* 向右浮动 */
/* 文本右对齐 */
/* 固定高度 */
/* 定义顶部外边距 */
/* 定义内边距 */

.buttons {/* 按钮样式 */
padding:6px 14px;
border:2px solid;
border-color: #fff #d8d8d0 #d8d8d0 #fff;
background: #e3e3db;
color: #989070;
font-weight:bold;
}
/* 定义内边距 */
/* 设计边框样式 */
/* 设计立体边框效果 */
/* 背景色 */
/* 字体颜色 */
/* 加粗显示 */

}
</style>

```

通过右对齐标签，使得标签与输入框之间的联系更紧密。然而这样也造成了文本左边参差不齐，使用户很难快速检索表单要填写的内容。这对于英文字符以及西方书写习惯很不适应，容易给用户造成阅读障碍。这时建议定义标签文本向左对齐。

标签左对齐布局的优点可以方便检索，并且减少垂直高度，但是它也存在一个左右距离问题。解决的方法是通过定义标签的显示样式，分隔不同行标签。

例如，定义背景色和分割线，不同的背景色产生了一系列垂直的标签和一系列垂直的输入框，每一组标签和输入框用清晰的水平线分开，标签样式如下，显示效果如图 6.25 所示。

```

label {/* 提示标签样式 */
width:142px;
height:32px;
}
/* 固定宽度 */
/* 固定高度 */

```



```

margin:3px 2px 0 0;
padding:11px 0 0 6px;
float:left;
color:#666;
background-color:#CCCCCC;

```

```

/* 定义外边距 */
/* 定义内边距 */
/* 向左浮动显示 */
/* 字体颜色 */
/* 背景颜色 */

```

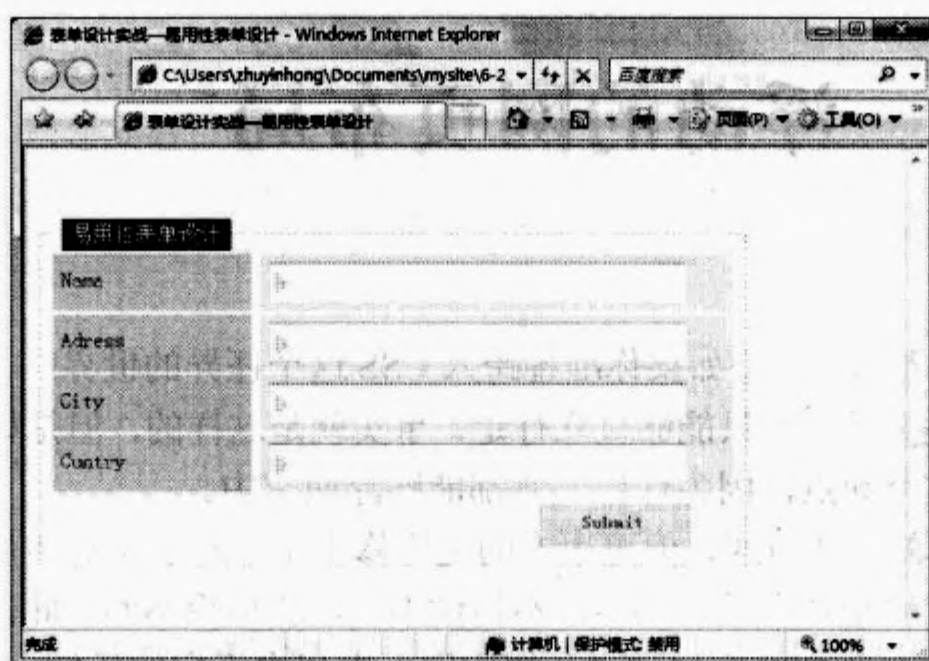


图 6.25 使用 CSS 设计标签提示文本的样式

第 7 章

浮动的网页布局

地球是圆的，这话不假，但是如果你准备跨入 CSS 这个怪异的世界时，你应该改改这种惯性的思维，因为 CSS 是方的。人只能够直立行走，事实就是这样的，但是如果你准备加入 CSS 的国度，你应该改变这个观点，因为任何元素都能够浮向空中。

CSS 可能会改变你的工作方式，但更根本的是它修正了你对于网页布局的全新认识。当你准备学习 CSS 布局时，请记住我的一句话：不怕做不到，就怕想不到。此话是否说得有理，那就需要你跟随我一起深入到 CSS 底层技术，探究网页布局中无穷的乐趣。

7.1 CSS 的“水立方”模型

第一次接触 CSS 时，想模拟椭圆形状设计一个圆角区域，结果没有成功，后来借助图像才得以实现。于是我就百思不得其解，既然现实世界到处都存在矩形、椭圆等最基本的几何形状，那么为什么 CSS 就没有呢？CSS 的设计者在第一次开发它时，竟然忘记或冷落了椭圆这个最完美的形状，是不是历史性的遗忘，还是另有隐情？这不得而知，只能任凭自己去傻想吧。

后来设计者虽然在 CSS 3.0 版本中开始支持椭圆模型，但是时过境迁，如今没有一家浏览器能够与时俱进，支持这个晚来的椭圆模型了。对于广大设计师来说就只能把希望寄托于未来的浏览器，椭圆模型暂时还仅是一个空中楼阁，让众多设计师望圆兴叹。

7.1.1 盒模型

CSS 世界无处不是方形的盒子。段落（p 元素）是个方盒子，即使整个段落呈现为多边形；超链接（a 元素）是一个方盒子，即使它没有边、也没有形状；虽然你插入的图像是一个椭圆形，但是图像本身（img 元素）依然是一个方盒子，而无论你怎么调整图像的形状。

这样概括吧，网页中所有的元素都是方形的（或者说矩形的），即使你看不见它，摸不到它，它们都会以方形而存在。如图 7.1 所示，为了方便你直观观看，这里我把所有元素都增加了描边显示出来。

这让人想起了建筑用的砖，实际上网页中的每个元素正是一块块砖，由它们砌起了完整的网页。因此，盒模型是 CSS 的基础，也是网页布局的根基。当你理解了盒模型的结构，你才能够自由地驾驭每个元素，随心所欲地堆叠每个标签。

如果在 Dreamweaver CS3 中随意定义一个元素，请注意要保证元素以块状显示，你就会很直观地看到该元素的完整盒模型结构，如图 7.2 所示。



图 7.1 网页无处不在的方盒子

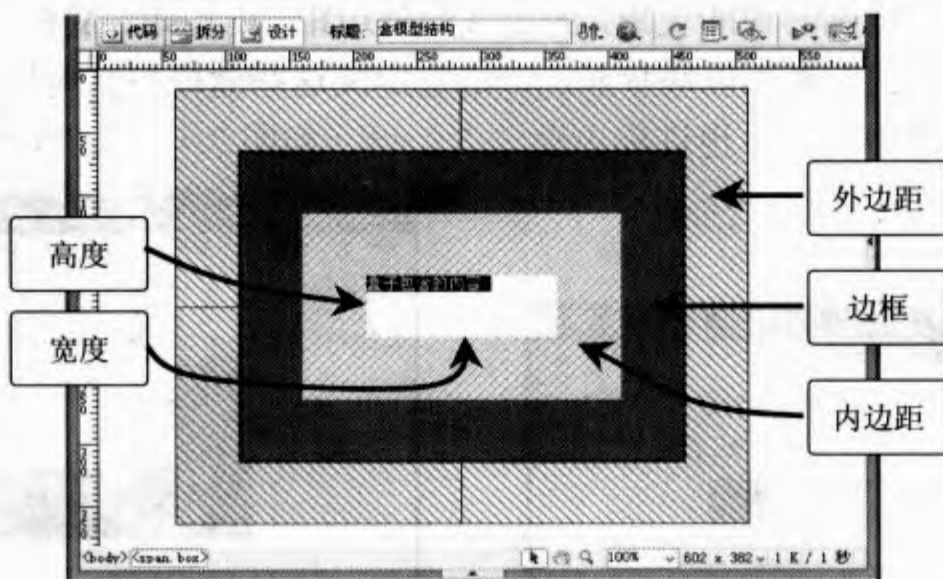


图 7.2 盒模型结构

```
<style type="text/css">
.box {/* 盒模型结构 */
    display:block;           /* 定义元素盒状显示, 或者说是块状显示 */
    width:150px;             /* 盒宽 */
    height:50px;             /* 盒高 */
    margin:50px;              /* 盒的外边距, 也称为边界 */
    padding:50px;            /* 盒的内边距, 也称为补白 */
    border:solid 50px red;    /* 盒边框 */
}
</style>
<span class="box">盒子包含的内容</span>
```

因此, 我们可以这样来描述: 盒模型是一个显示为方形的, 可以拥有外边距、内边距和边框, 并能够定义宽和高的方形区域。盒模型内部可以包含其他盒模型或者对象。

7.1.2 盒模型解疑

知识往往是越学越迷惑。也许当你初步明白盒模型是怎么回事时, 一个个疑团便会接踵而至。你不说所有元素都是盒状的, 为什么还要定义元素显示为块状呢? 所有元素都必须定义这

些盒模型结构属性吗?

是的,网页内所有元素都具有盒模型的基本特征,这正如所有人都具备人的直立行走且能够思考这样一个基本特性一样,但是并不是所有人都长得五官端正。

在 CSS 中,虽然每个元素都必须以方形显示,当你给它定义边框时,所显示的都是矩形的四边形,而不是椭圆形或者多边形。但是并不是每个元素都必须拥有外边距、内边距、宽和高。举个简单的例子,如果针对上面的示例代码,删除其中的 `display:block;` 声明,你会看到什么情景呢?(如图 7.3 所示)。



图 7.3 残缺的盒模型

这是一个令人难以自圆其说的现象,所定义的外边距、内边距、宽和高都没有正确地显示,或者说就不支持。如果说元素不以块状显示,给它定义这些盒模型结构属性就没有意义了。如果分别在不同的浏览器中预览,也许你更会觉得惊讶(如图 7.4、图 7.5 所示)。



图 7.4 IE 6 中解析效果

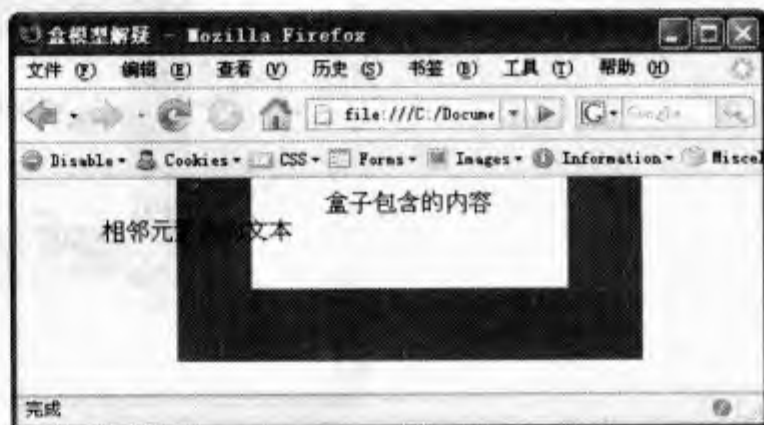


图 7.5 Firefox 2.0 中解析效果

为了减轻这种疑惑,不妨这样下个结论:盒模型是以方形为基础进行显示的,不管其最终形状,还是它的边距、边框和大小,都显示为方形。盒模型可以拥有外边距、边框、内边距和宽、高基本属性。但是并不要求每个元素都必须定义这些属性,或者都必须拥有这些属性。

块状流动或浮动的元素可以定义完整的盒模型结构。但是对于流动的行内元素来说就没有高、宽和部分边距属性。而对于绝对定位的层元素来说,就没有外边距属性。有关这方面知识的详细内容在后面的章节中还会不断深入讲解。

7.1.3 盒模型的外边距

盒模型的外边距可以比作书的页边距,它似乎像一个磁场,透明无影,但又是真实存在的。确切地说就是元素边框外边沿到相邻元素之间的距离。为什么要为盒模型定义这样一个结构属性呢?

盒模型的外边距主要是用来分开各种元素,或者说就是定义元素之间的距离。没有外边距的网页,所有对象都被堆放在一起,密密麻麻,杂乱无章。

定义外边距可以使用 `margin` 属性,该属性可以取负值,由此可以利用负值来设计各种复杂

的网页布局，在下面的内容中我们将详细讲解。

margin 默认值为 0。如果你没有定义元素的 margin 属性，则浏览器会认为元素的外边距为 0，或者说不存在外边距，换句话说元素可以与其他元素零距离接触。网页正是借助外边距来调节网页布局的疏密和对齐，当然它不是唯一的方法，后面我们还会讲解盒模型的内边距。

定义盒模型的外边距有多种方法，概括起来有如下 7 种方法，你可以任意选择其中一种来定义元素的外边距。当混合定义时，要注意取值的先后顺序，一般是从顶部外边距开始，按顺时针分别定义：

```
<style type="text/css">
margin:10px;                /* 快速定义盒模型的外边距都为 10 像素 */
margin:5px 10px;            /* 定义上下、左右外边距分别 5 像素和 10 像素 */
margin:5px 10px 15px;       /* 定义为 5 像素，左右为 10 像素，底为 15 像素*/
/* 定义为 5 像素，右为 10 像素，下为 15 像素，左为 20 像素*/
margin:5px 10px 15px 20px;
margin-top:5px;             /* 单独定义上外边距为 5 像素 */
margin-right:10px;          /* 单独定义右外边距为 5 像素 */
margin-bottom:15px;         /* 单独定义底外边距为 5 像素 */
margin-left:20px;           /* 单独定义左外边距为 5 像素 */
</style>
```

7.1.4 行内元素的外边距

当为行内元素定义外边距时，你只能看到左右外边距对于版式的影响，但是上下外边距犹如不存在一样，不会对周围的对象产生影响。例如，设计一个如下的模型和样式，在不同浏览器中预览，将会看到如图 7.6、图 7.7 所示的解析效果。这说明行内元素没有发挥外边距应有的功能，你不能够使用外边距来调节行内元素与其他对象的位置关系，但是可以调节行内元素之间的水平距离。

```
<style type="text/css">
.box1 { /* 行内元素样式 */
margin:50px;                /* 外边距为 50 像素 */
border:solid 20px red;      /* 20 像素宽的红色边框 */
}
.box2 { /* 块状元素样式 */
width:400px;                /* 宽度 */
height:20px;                /* 高度 */
border:solid 10px blue;     /* 10 像素宽的蓝边框 */
}
</style>
<div class="box2">相邻块状元素</div>
<div>外部文本<span class="box1">行内元素包含的文本</span>外部文本</div>
<div class="box2">相邻块状元素</div>
```

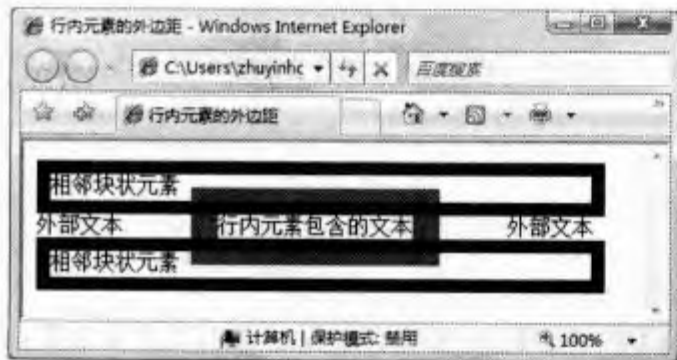


图 7.6 IE 7 中解析效果



图 7.7 Firefox 2.0 中解析效果

提示

span 元素默认为行内元素显示, div 默认为块状元素显示, 你可以通过 display 属性来改变它们的默认显示属性。在下面章节中我们还会专门讲解什么是行内元素和块状元素, 以及元素的显示属性内容。

7.1.5 块状元素的外边距

对于块状元素来说, 外边距都能够很好地被解析。CSS 的盒模型最初就是针对块元素来设计的。例如, 如果把上面示例中的 span 元素定义为块状显示, 则你将会看到另外一种效果, 如图 7.8、图 7.9 所示。

```
.box1 {
    display: block;          /* 块状显示 */
    margin: 50px;           /* 外边距 */
    border: solid 20px red;  /* 红色实线边框 */
}
```

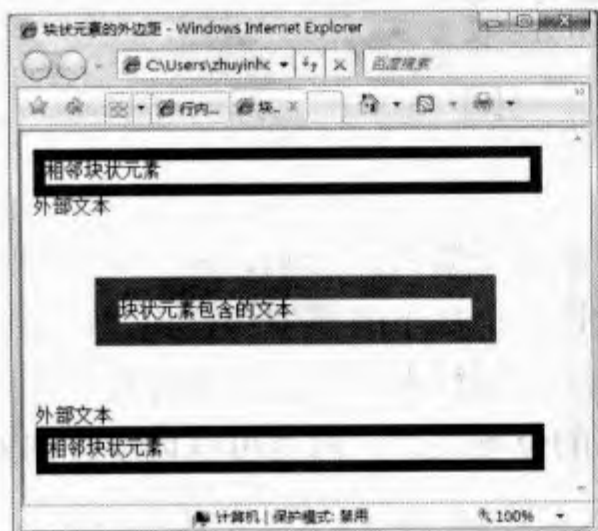


图 7.8 IE 7 中解析效果

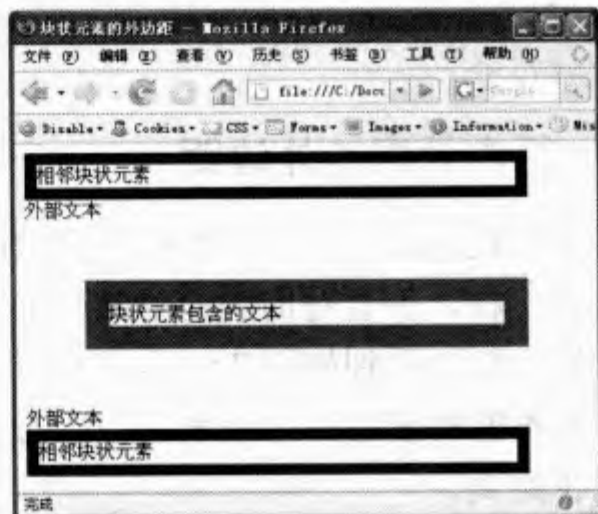


图 7.9 Firefox 2.0 中解析效果

因此, 我们可以这样做一个小结: 对于块状元素来说, 你可以自由地使用外边距来调节网页版式和元素之间的距离。

7.1.6 浮动元素的外边距

首先明确一下, 元素浮动显示与块状、行内等显示属性是两个不同的概念, 或者说是两种不同性质的对象。不管何种性质的元素, 一旦被定义为浮动显示, 它就拥有完整的盒模型结构, 你可以使用外边距、内边距、边框、高和宽来控制它的大小以及与其他对象之间的位置关系。

由于浮动是从网页布局的角度来定义元素的显示, 而行内和块状属性主要是从元素自身的性质来定义其显示。因此, 当为浮动元素定义外边距后, 所呈现的效果会很复杂, 与我们的设想会大不相同。不同浏览器对此解析的效果也是千差万别, 更让人无所适从。

例如, 定义如下的模型结构和样式, 然后在 IE 7 和 FF 2 (Firefox 2.0 的简写) 中预览, 显示效果如图 7.10、图 7.11 所示。

```
<style type="text/css">
.box1 {
    float: left;          /* 向左浮动显示 */
    margin: 50px;         /* 外边距 */
    border: solid 20px red; /* 红色实线边框 */
}
```



```
.box2 {  
    width:400px;           /* 块状元素宽度 */  
    height:20px;           /* 块状元素高度 */  
    border:solid 10px blue; /* 块状元素边框 */  
}  
</style>  
<div class="box2">相邻块状元素</div>  
<div>外部文本<span class="box1">浮动元素</span>外部文本</div>  
<div class="box2">相邻块状元素</div>
```



图 7.10 IE 7 中解析效果



图 7.11 Firefox 2.0 中解析效果

因此，我们可以这样做一个小结：对于浮动元素来说，你可以自由地使用外边距来调节浮动元素与其他元素之间的距离。但是由于浮动元素与其他元素之间存在复杂的交错现象，使用外边距调节元素之间的距离时，会存在很多不确定性，也就是说你所希望的调节距离与实际显示距离会存在一定的误差，且不同浏览器对此的解析标准不同，导致布局会出现很多预想不到的问题。针对这些问题，我们还会在下面章节中详细讲解。

7.1.7 绝对定位元素的外边距

如果我们再尝试把上节示例中的浮动显示改为绝对定位显示：

```
.box1 {  
    position:absolute;      /* 绝对定位显示 */  
    top:0;                  /* 距离页面顶部的距离 */  
    left:0;                 /* 距离页面左边框的距离 */  
    margin:50px;            /* 绝对定位元素的外边距 */  
    border:solid 20px red;  
}
```

这时你会看到绝对定位元素依然拥有外边距，虽然这些外边距不会影响其他元素的位置，其他元素也不会影响它的位置，但是你仍然可以看到外边距在定位中的作用。浏览器都不是以元素外边距左上角的顶点作为移动点，来决定元素的位置，如图 7.12、图 7.13 所示。

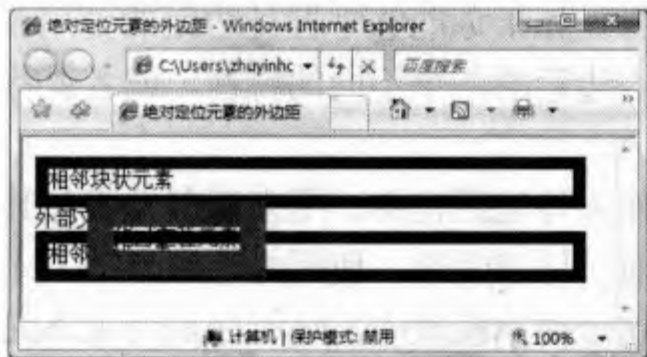


图 7.12 IE 7 中解析效果



图 7.13 Firefox 2.0 中解析效果

但是考虑到外边距在绝对定位中没有实际价值，你完全可以使用元素边框来定位，无非就是取值大小不同而已，所以在设计中都被忽略了。例如，你可以将上面的示例改为如下样式，所得效果都是相同的。

```
.box1 {
    position:absolute;          /* 绝对定位显示 */
    top:50px;                   /* 距离页面顶部的距离 */
    left:50px;                  /* 距离页面左边框的距离 */
    border:solid 20px red;
}
```

但是这并不说明绝对定位不可以定义外边距，也不能够说绝对定位元素的外边距不起作用。很多初学者在这里往往就被误导了。

7.1.8 盒模型的内边距

从外边距概念中走出来，我们再来看看盒模型的内边距。

所谓盒模型的内边距就是元素包含的内容与元素边框内边沿之间的距离。盒模型的内边距主要功能就是用来调整元素所包含的内容在元素中的显示位置。例如，输入如下元素和样式，可以把包含的文本推挤到元素右下角显示（如图 7.14 所示），但是如果没有内边距的作用，它只能够在元素左上角显示（如图 7.15 所示）。

```
<style type="text/css">
.box1 {
    display:block;              /* 块状显示 */
    padding-left:160px;          /* 左内边距 */
    padding-top:60px;           /* 顶部内边距 */
    width:100px;                /* 元素的宽度 */
    height:30px;                /* 元素的高度 */
    border:solid 20px red;       /* 元素的边框 */
}
</style>
<span class="box1">包含文本</span>
```



图 7.14 用内边距调节包含内容的显示位置

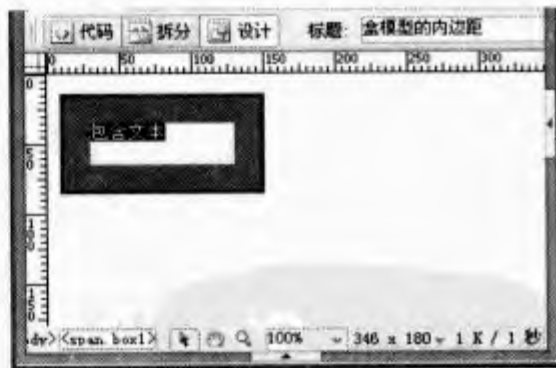


图 7.15 包含内容默认显示在左上角

盒模型的内边距与外边距在用法上有很大的相似性，如果你能够掌握外边距的使用，内边距就比较容易理解了。但是在使用时，你还应该了解内边距的几个不同的特性，或者说特殊的用法。

第一，当元素没有定义边框时，你可以把内边距作为外边距使用，用来调节元素与其他元素之间的距离。由于外边距相邻时会出现重叠现象，而且比较复杂，使用内边距来调节元素之间的距离往往不用担心边距重叠问题。例如，在下面这个简单模型中，你可以看到上下元素之间的距离很近（50 像素），而不是 100 像素（如图 7.16 所示）。因为上下相邻元素的外边距发生

了重叠现象。

```
<style type="text/css">
.box1 { margin-bottom:50px; }          /* 底部外边距 */
.box2 { margin-top:50px; }            /* 顶部外边距 */
</style>
<div class="box1">第一个元素</div>
<div class="box2">第二个元素</div>
```

但是如果其中一个元素使用内边距来定义，则效果截然不同（如图 7.17 所示）。可以看到，使用内边距调节元素之间的距离时不会出现重叠问题。

```
<style type="text/css">
.box1 { padding-bottom:50px; }        /* 底部内边距 */
.box2 { margin-top:50px; }           /* 顶部外边距 */
</style>
```



图 7.16 外边距会发生重叠

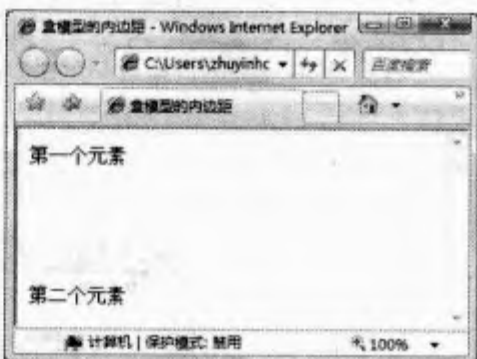


图 7.17 内边距不会重叠

第二，当为元素定义背景图像时，内边距区域内可以显示背景图像。而对于外边距区域来说，背景图像是达不到的，它永远表现为透明状态。利用内边距的这个特征，你可以为元素增加各种修饰性背景图像，在前面章节中我们曾经讲解如何制作导航条、项目符号时，正是利用这个功能的。例如，输入如下的样式，你可以设计一个图文并茂的画面（如图 7.18 所示）。

```
<style type="text/css">
.box1 {
padding:50px 100px;          /* 内边距 */
width:518px;                 /* 宽度 */
height:113px;               /* 高度 */
background:url(images/bg1.jpg) no-repeat; /* 背景图像 */
border:solid 10px #522917;   /* 边框 */
}
</style>
<div class="box1">历史的记忆</div>
```

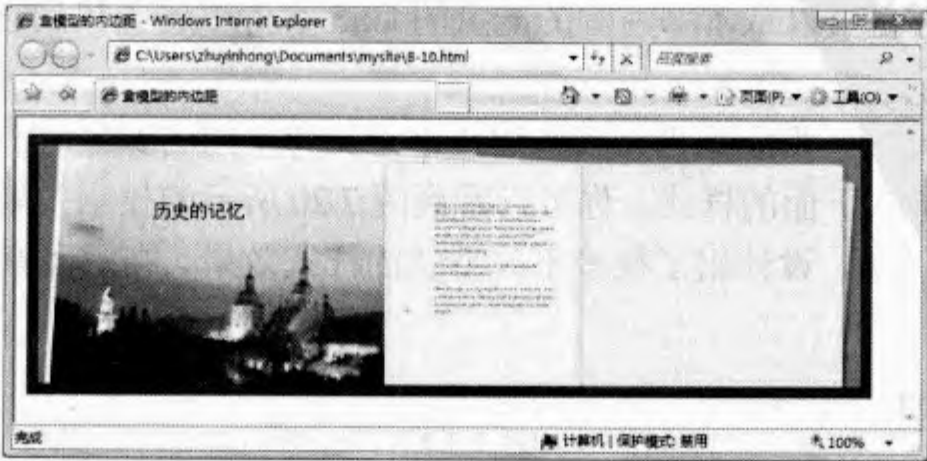


图 7.18 内边距区域显示的背景图像

第三,对于行内元素的内边距来说,它能够影响元素边框的大小,而外边距不存在这样的问题。行内元素的外边距对于任何对象都不会产生影响。例如,输入下面的样式,你将会看到如图 7.19 所示的效果。

```
<style type="text/css">
.box1 {
    padding:50px;                /* 内边距 */
    border:solid 20px red;        /* 红色实线边框 */
}
.box2 {
    width:400px;                /* 块状元素宽度 */
    height:20px;                /* 块状元素高度 */
    border:solid 10px blue;      /* 块状元素边框 */
}
</style>
<div class="box2">相邻块状元素</div>
<div>行内文本行内文本行内文本行内文本行内文本行内文本行内文本行内文本行内文本
<span class="box1">行内元素包含的文本</span>行内文本行内文本行内文本行内文本行内文本行内文本
行内文本行内文本行内文本行内文本行内文本</div>
<div class="box2">相邻块状元素</div>
```

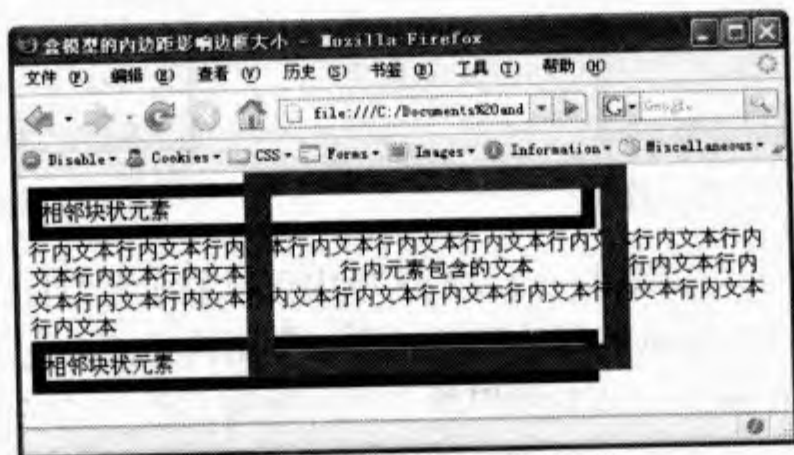


图 7.19 内边距影响元素边框的大小效果

7.1.9 盒模型的边框

边框是盒模型的一个重要概念,同时任何元素都可以定义边框,并都能够很好地显示出来。边框在网页布局中的作用就是用来分隔模块。

但是与内外边距不同的是,盒模型的边框包含 3 个基本属性:样式 (border-style)、颜色 (border-color) 和宽度 (border-width)。你可以为元素的边框指定样式、颜色或宽度,其中颜色和宽度可以省略,这时浏览器就会根据默认值来解析。有关这些边框的基本用法在前面各章节中都有过介绍,同时你也可以从本书附赠的光盘的 CSS 参考手册中找到它们的用法,所以这里就不再叙述。

但是有一点需要注意,那就是当元素各边边框定义为不同的颜色时,边角会以平分来划分颜色的分布。例如,输入下面的样式,你可以看到图 7.20 所示的显示效果。很多设计师正是利用边框的这个特殊的效果,设计出了很多不同形状的样式效果。在前面章节中我们也曾经举过类似的示例。

```
<style type="text/css">
.box {
    border:solid 100px;          /* 边框样式和宽度 */
    border-color:red blue green; /* 定义不同边框显示为不同颜色 */
}
```



```
line-height:0; /* 定义行内文本高度为 0，这样就避免元素内出现空隙 */
}
</style>
<div class="box"></div>
```

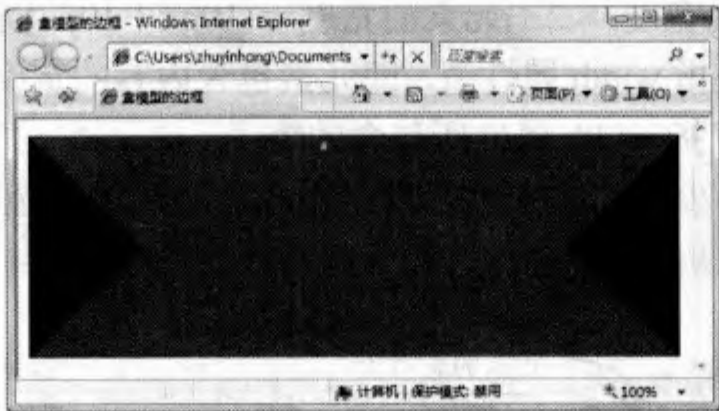


图 7.20 元素的边框

7.1.10 盒模型的宽和高

在网页布局中如何去计算元素的宽度和高度曾经是个很头疼的问题，因为在 IE 6 以下版本的浏览器中，对 width 和 height 属性的解析规则与 W3C（World Wide Web Consortium，万维网联盟）的标准完全不同，曾经让人郁闷了很久。从 IE 6 开始微软更正了这个错误，按 W3C 所倡导的标准来解析 width 和 height 属性了。

但是，盒模型的宽度和高度计算问题依然让很多初学者犯难。分析原因，主要是在网页布局中高和宽的概念区分上：

- 元素的总高度和总宽度；
- 元素的实际高度和实际宽度；
- 元素的高度和宽度。

我们不妨以一个简单的示例来解释一下。请输入下面代码，效果如图 7.21 所示。

```
<style type="text/css">
div {
    float:left;           /* 向左浮动 */
    height:100px;         /* 元素高度 */
    width:160px;          /* 元素宽度 */
    border:10px solid red; /* 边框 */
    margin:10px;          /* 外边距 */
    padding:10px;         /* 内边距 */
}
</style>
<div class="left">左侧栏目</div>
<div class="mid">中间栏目</div>
<div class="right">右侧栏目</div>
```

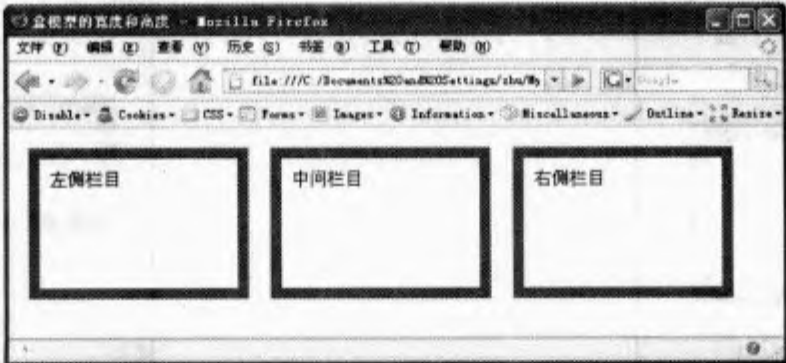


图 7.21 元素的宽和高

对于上面这个简单的布局，左侧栏目的宽度是多少？你可能不假思索地说是 160 像素。实际上这是不对的，栏目的宽度也是左侧 `<div class="left">` 元素的实际宽度，应该是 200 像素。计算方法应该是：

$$(\text{边框宽度} + \text{内边距宽度}) \times 2 + \text{元素的宽度} = (10\text{px} + 10\text{px}) \times 2 + 160\text{px} = 200\text{px}$$

请注意，元素的宽度是指 `width` 属性值，即元素包含内容的宽度。但是对于整个栏目来说，元素的实际宽度就不能够仅指 `width` 属性所包含的值了。

那么元素的总宽度是多少呢？这时还应该考虑元素的外边距，虽然它不是栏目的实际宽度，而仅用来调节元素之间的距离，但是在计算元素总宽度时，还应该包括它的值（如图 7.22 所示）。

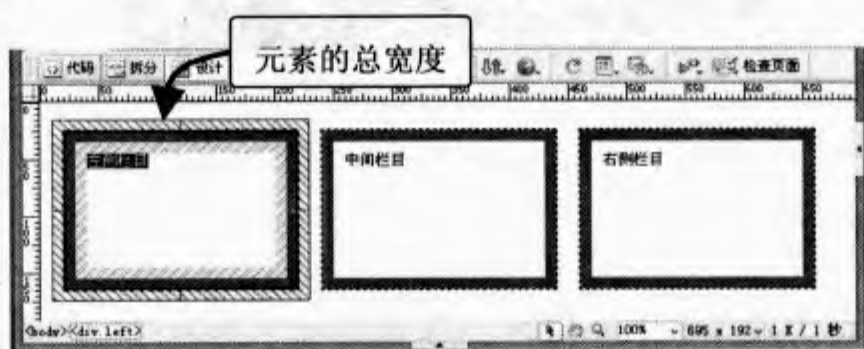


图 7.22 元素的总宽度

还有一个问题，如果元素没有定义边框，所定义的内边距仅用来分隔与其他元素的距离，这时计算元素的实际宽度时，我们就应该根据布局栏目的实际情况，确定它的实际宽度为元素的宽度，就不能够盲目地把内边距都算作栏目的宽度。

IE 5.5 及以下版本浏览器在解析 `width` 和 `height` 属性时，会错误地把边框和内边距也算进去，所以就导致了元素宽度计算的混乱。好在目前大部分用户都使用 IE 6 及以上版本浏览器，这个问题破坏性也就不是那么严重了，但是建议你还是了解一下如何处理这个问题，以实现在 IE 5.5 及以下版本中能够正确地解析网页。

例如，输入如下一个简单的示例代码，然后分别在 IE 6 和 IE 5.5 中预览，则显示效果如图 7.23、图 7.24 所示。

```
<style type="text/css">
div {
    height:100px;           /* 元素的高度 */
    width:200px;            /* 元素的宽度 */
    border:20px solid red;   /* 边框 */
    margin:20px;            /* 外边距 */
    padding:20px;           /* 内边距 */
}
</style>
<div>IE 5.5 及以下版本浏览的盒模型 Bug</div>
```



图 7.23 IE 6 中的解析效果



图 7.24 IE 5.5 中的解析效果

从上面的图示可以比较出,在 IE 5.5 及以下版本中, width 属性的值就是指元素的实际宽度,也就是说 width 属性值中包含 padding 和 border 属性值。所以你看不到 IE 5.5 版本浏览器解析的对象要小很多。解决此类问题的方法是使用浏览器兼容技术。例如,可以使用 voice-family 方法来解决,具体代码如下:

```
<style type="text/css">
div {
    height:180px;           /* IE 5.5 及以下版本中解析的宽度 */
    width:280px;           /* IE 5.5 及以下版本中解析的高度 */
    border:20px solid red;  /* 边框 */
    margin:20px;           /* 外边距 */
    padding:20px;          /* 内边距 */
    voice-family:"\"}\"";   /* 设置播放的声音 */
    voice-family:inherit;   /* 设置播放的声音 */
    width:200px;           /* 非 IE 5.5 及以下版本中解析的宽度 */
    height:100px;          /* 非 IE 5.5 及以下版本中解析的高度 */
}
</style>
```

由于 IE 5.5 及以下版本浏览器不能够识别 voice-family:属性,当解析"\"}\"";时,会误以为该样式已经结束,于是就忽略掉后面所定义的规则,但是非 IE 5.5 及以下版本浏览器能够识别 voice-family:属性,并继续解析后面所定义的属性,通过这种方法为 width 和 height 属性设置不同的值,从而达到让所有浏览器都能够解析为相同的效果的目的(如图 7.25、图 7.26 所示)。



图 7.25 IE 6 中解析效果



图 7.26 IE 5.5 中解析效果

7.2 构建符合标准的网页结构

提及符合标准的网页结构,你可能会想到或听说过各种网络、书籍所宣扬的“勿用表格”论。对网页设计稍有一点知识背景的人,都知道传统网页布局中表格应用得很疯狂,或者说达到了泛滥的程度。现在是 Web 2.0 时代了,所以大家对表格开始持抵触情绪,于是又从“左倾”转向“右倾”。我们现在先不论它是对是错,那么到底什么是符合标准的网页结构呢?

7.2.1 是内容决定结构,还是表现决定结构

网页的结构由什么来决定?传统网页设计师可能会说是艺术,前端设计师者可能会说是技

术；但标准网页设计师会说，一切都不重要，唯一重要的是网页内容。

当使用表格来设计网页时，你可能会很少想到网页的内容。在 Photoshop 或者 Fireworks 中，设计师满脑子都是如何把页面润色得更漂亮，如何使页面设计得更秀、更酷，然后再用小刀似的切片工具唰唰地把这幅图片按内容分割为大大小小的不同区域。这种设计思维和方法是艺术论的典型做法。

现在使用 CSS 来布局了，设计师的思维需要转换一下，不能够再这样思考问题了。设计师应先根据网页内容的需要，用不同的标签把不同内容存放在不同的网页结构中。这时你可能还没有想到网页最终所要显示的效果。但不要紧，因为 CSS 说：只要把结构设计得合理，我一定能帮你把页面撑起来，并打扮得很漂亮。于是 HTML 结构和 CSS 表现就这样分离了。

举个简单的示例。图 7.27 所示是新浪网的置顶导航条。对于传统网页设计师来说，可能会想到：先插入一个 2 行 1 列的表格；然后在第 1 行的单元格中插入一个 1 行 12 列或更多列的表格，在第 2 行的单元格中插入一个 1 行 2 列的表格；最后再在左侧插入一个 1 行 1 列的表格，用来插入 Logo，在右侧插入 3 行 21 列的表格，如图 7.28 所示。

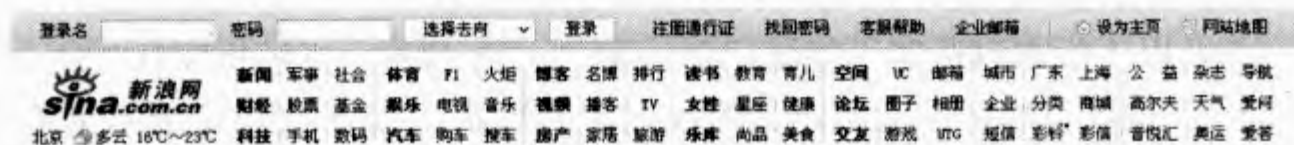


图 7.27 网页结构和内容的关系

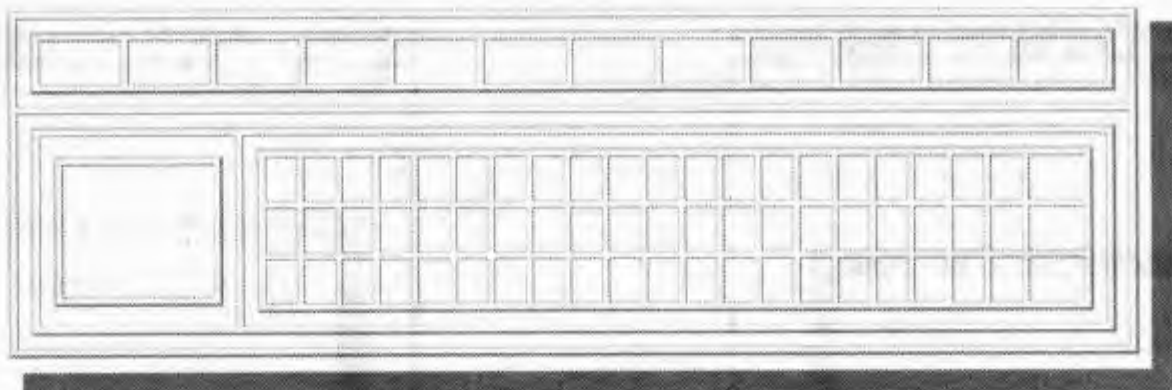


图 7.28 表格布局的结构

但是在标准网页布局中，你所能够看到的全部都是图 7.29 所示的内容，看到这些内容你可能无法与图 7.27 所示的页面效果联系在一起。因为在设计网页结构时，设计师满脑子所要想到的都是网页内容，而非内容所要呈现的效果。这正是设计思维的一种倒位。

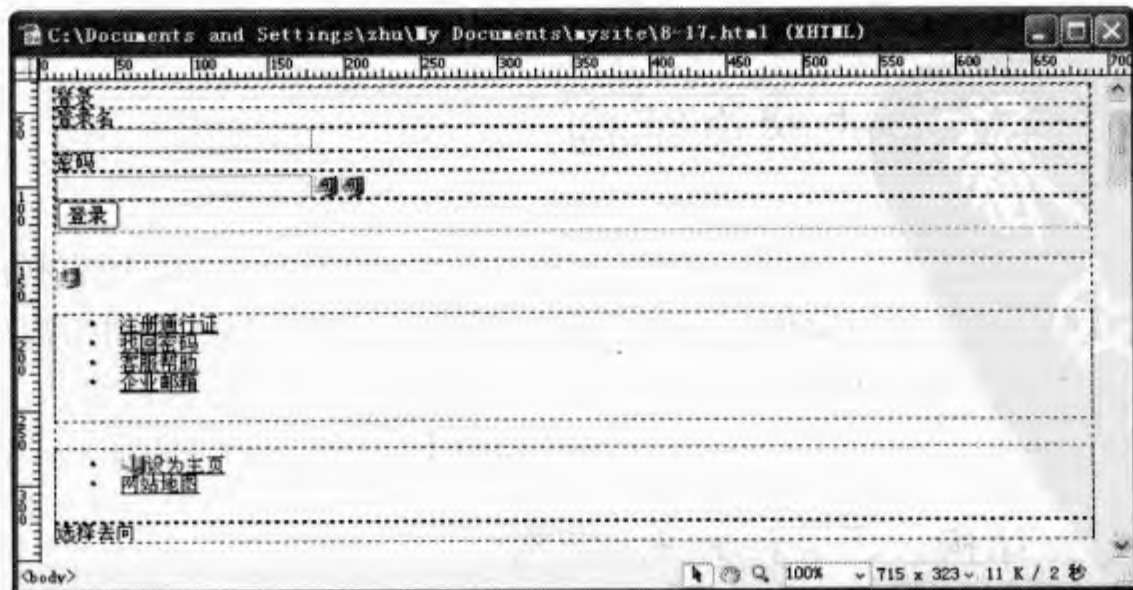


图 7.29 标准网页布局的结构

7.2.2 选择好结构标签

设计标准网页结构，首先需要选用好结构标签。HTML 提供的标签元素有很多，有用来处理网页结构的，如 `p` 负责组织文本段，`h2` 负责标题行，`ul` 负责项目列表；另外还有一些标签用来表现网页效果的，如 `em` 表示斜体，`b` 表示粗体，`s` 表示删除线，`font` 用来定义字体显示属性等。表现网页效果的元素一部分已被废弃，或者不再提倡使用；另一些作为语义标签使用，但是在网页布局中一般使用结构标签进行结构设计。

选择结构标签的标准可以有两个：标签的语义性和标签元素的显示性。

所谓语义性，就是不同标签都代表一定的意思。例如，`table`、`tr` 和 `td` 表示数据表结构，`ul`、`ol`、`dl`、`dt`、`dd` 和 `li` 表示项目列表结构，`p` 表示段落结构，`h1`、`h2`、`h3` 等表示标题结构，`div` 表示模块包含框结构，`span` 表示行内包含框结构等。

根据不同的网页内容选择与之对应的语义结构标签会更利于网页结构的优化和识别，特别是搜索引擎的识别。

在前面章节中我们曾经多次提及块状元素和行内元素，实际上它们都是网页元素的两种基本显示属性。简单比较两者的区分如下。

- 不管块状元素宽度是多少，它总会自动占据一行，原来是因为它在末尾附加了一个换行符，而行内元素没有这一特征。所以块状元素只能单行显示，而不能并列显示。
- 块状元素拥有完整的盒模型结构，因此可以给它定义宽度和高度，而行内元素没有这样的特性，无法通过高度来改变文本行的行高。

块元素的典型代表是 `div` 元素，它表示包含结构块，是网页布局中使用频率最高的一个结构标签。基本上标准网页的结构主要是由 `div` 元素负责实现的。

行内元素的典型代表是 `span` 元素，它表示行内包含结构，一般多用于修饰文本行内元素的属性，不具备组织结构框架。

当然，你可以使用 `display` 属性来改变它们的显示属性，但是除非必要，在网页布局中不建议这样使用，否则会根据元素默认显示属性来执行相应的任务。

实际上，CSS 所提供的 `display` 属性还包含更多的属性，简介如下。

- `none`：隐藏元素显示。该值与 `visibility` 属性的 `hidden` 值不同，因为它会自动隐藏元素所占据的网页空间，而 `visibility` 属性的 `hidden` 值仅是让元素不可见。
- `inline-block`：行内显示，但是对象的内容可以作为块状呈现。
- `list-item`：将块状元素设置为列表项，并可以添加项目符号。`li` 元素默认为该属性显示。
- `table`：将对象作为块元素级的表格显示。`table` 元素默认为该属性显示。
- `table-cell`：将对象作为表格单元格显示。`td` 元素默认为该属性显示。
- `table-row`：将对象作为表格行显示。`tr` 元素默认为该属性显示。

显示属性还有更多，但由于 CSS 不支持或者浏览器不能够很好解析，因此建议读者仅仅了解即可，详细内容可参阅光盘 CSS 参考手册。

7.2.3 研究禅意花园的网页结构

在前面的章节中我们曾经拜访过禅意花园 (<http://www.csszengarden.com/>)，很多设计师正是借助这个网站而不断成长起来的。禅意花园的主人提供了一个完全符合标准的网页结构，并倡导所有设计师以这个结构为模板，积极探索使用 CSS 设计不同的网页表现效果。

研究这个网页结构本身就是一件很好的事情，何况还要在此基础上继续深入探索如何使用

CSS 实现布局。所以我把这个网页结构进行了汉化，本书后面的所有示例都将以此为基础展开讲解，并以此结构为基础讲解不同网页布局类型和技巧的实现。

建议读者先弄明白禅意花园的结构以及它的设计思路。当学习这样经典的标准网页结构时，你才能够更真实、更直观、更深切地体会到什么样式的网页结构才是符合标准的网页结构。通过如此经典的标准网页结构的学习，能够更快地提高你的网页驾驭水平。可以毫不夸张地说，禅意花园网页结构的每个细节都是值得认真学习和研究的。

网页包含框。整个网页都处于这样的一个包含框之中。

```
<body id="css-zen-garden">
  <div id="container"> </div>                                <!-- 网页包含框 -->
</body>
```

包含框内包含了 3 个二级模块，分别是：介绍、支持文本、链接列表。介绍模块内容主要包括网页标题信息、网页内容概括和引言内容；支持文本模块是整个网页内容的主体，详细说明如何参与禅意花园活动的说明、要求和参与的好处，以及页脚信息；链接列表主要包括各种链接信息。

```
<body id="css-zen-garden">
  <div id="container">                                <!-- 网页包含框 -->
    <div id="intro"></div>                            <!-- 介绍 -->
    <div id="supportingText"></div>                  <!-- 支持文本 -->
    <div id="linkList"></div>                        <!-- 链接列表-->
  </div>
</body>
```

介绍模块中又包含页标题、简明概括和导言 3 个三级模块。

```
<body id="css-zen-garden">
  <div id="container">                                <!-- 网页包含框 -->
    <div id="intro"></div>                            <!-- 介绍 -->
      <div id="pageHeader"></div>                    <!-- 网页标题 -->
      <div id="quickSummary"></div>                  <!-- 简明概括 -->
      <div id="preamble"></div>                      <!-- 导言 -->
    </div>
  </div>
</body>
```

支持文本模块中包含说明、参与、益处、要求和页脚 5 个三级模块。它主要包括活动的说明、邀请您参与、参与带来的好处，以及对参与者的要求和页脚信息。

```
<body id="css-zen-garden">
  <div id="container">                                <!-- 网页包含框 -->
    <div id="supportingText">                        <!-- 支持文本 -->
      <div id="explanation"></div>                    <!-- 说明 -->
      <div id="participation"></div>                <!-- 参与 -->
      <div id="benefits"></div>                      <!-- 益处 -->
      <div id="requirements"></div>                <!-- 要求 -->
      <div id="footer"></div>                        <!-- 页脚 -->
    </div>
  </div>
</body>
```

链接列模块中嵌套了一个包含框，这主要是为方便 CSS 控制而设计的。然后在其下面包含了 3 个子模块：作品选择列表、作品档案列表和资源列表。在这些列表模块中又利用 ul 项目列表元素来组织链接列表。


```
<body id="css-zen-garden">
  <div id="container">
    <div id="linkList">
      <div id="linkList2">
        <div id="lselect"></div>
        <div id="larchives"></div>
        <div id="lresources"></div>
      </div>
    </div>
  </div>
</body>
```

<!-- 网页包含框 -->

<!-- 链接列表-->

<!-- 链接列表 2-->

<!-- 作品选择列表-->

<!-- 作品档案列表-->

<!-- 资源列表-->

在所有三级或者四级模块中都包含了一个或多个标题行和段落行。标题行遵循页标题为一级标题，页副标题为二级标题，模块内标题为三级标题的思路来设计。

在网页结构中，标题级别越大，它的影响力就越大，搜索引擎也是按着一级标题、二级标题、三级标题等顺序来搜索的。

在整个网页的最后面又增加了 6 个额外的结构标签，这些结构默认为隐藏显示，它主要是为了方便设计师扩展网页的设计效果而增加的。

```
<div id="extraDiv1"><span></span></div>
<div id="extraDiv2"><span></span></div>
<div id="extraDiv3"><span></span></div>
<div id="extraDiv4"><span></span></div>
<div id="extraDiv5"><span></span></div>
<div id="extraDiv6"><span></span></div>
```

整个网页的结构用示意图演示如图 7.30 所示。

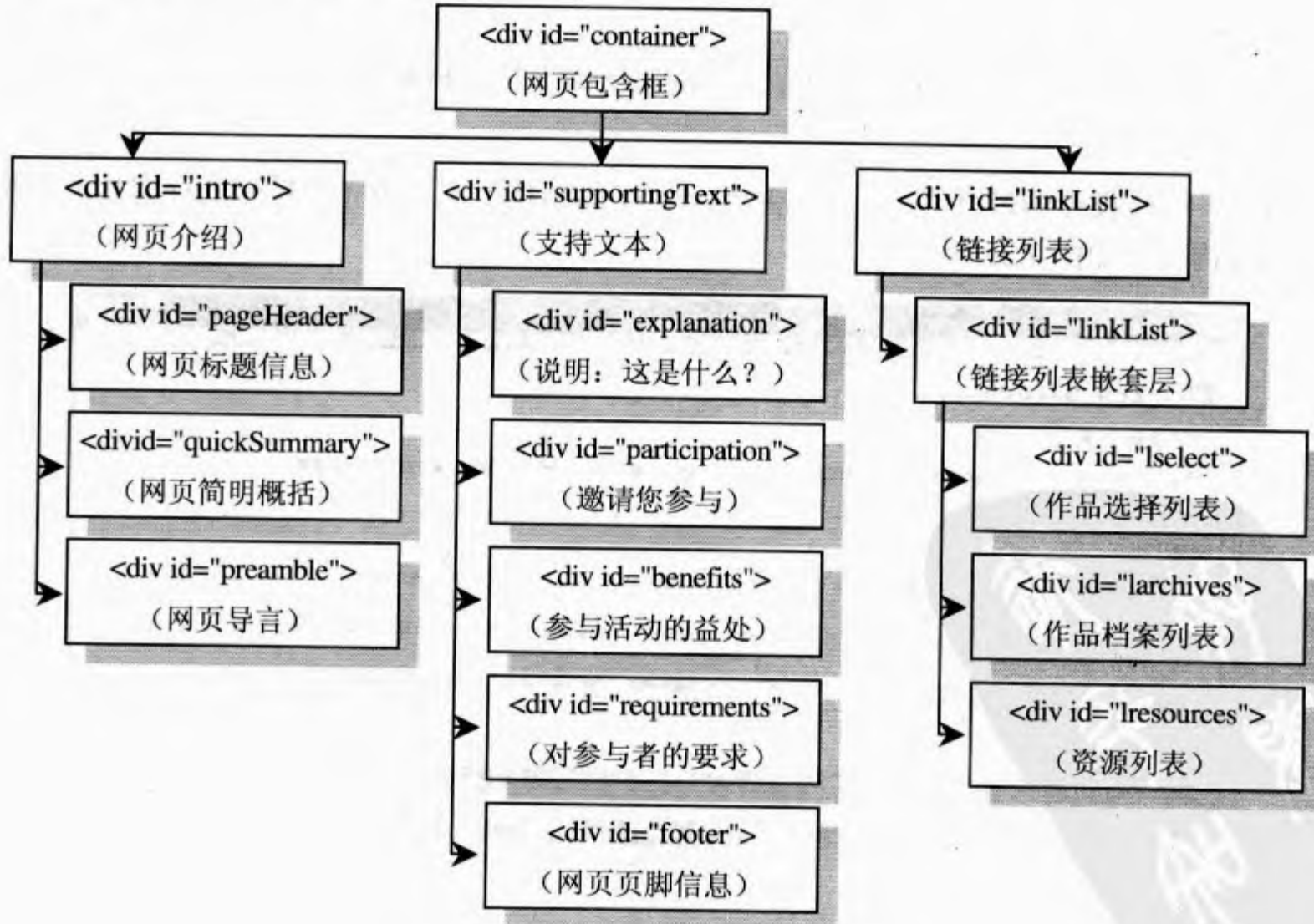


图 7.30 禅意花园网页结构示意图

由于 CSS 对于大小写很敏感。当我们使用大小字母来命名 id 或 class 属性值时，要注意大小写问题，否则样式表无效。

通过禅意花园的网页结构，我们也可以看到，在构建网页主体框架时，一般使用 id 属性来区分不同的结构标签，这是因为网页的结构一般都是唯一的。例如，一个网页只能包含一个页头信息块，也只能够包含一个页脚信息块等。而 id 属性值一般要求也是唯一的，一个页面内不能够同时定义两个相同名称的 id 属性。

而类样式就不同了，你可以定义一个类样式，然后在页面中多次应用。所以，当对结构体内的对象定义样式时，建议多采用类样式来实现。例如，下面的“网页简明概括”模块子结构就是定义 p1、p2 类样式来控制段落格式，而这两个类样式还可以在其他模块中应用。

```
<div id="quickSummary">
  <p class="p1"><span>展示以<acronym
title="cascading style sheets">CSS</acronym>技术为基础，并提供超强的视觉冲击力。只要选择
列表中任意一个样式表，就可以将它加载到本页面中，并呈现不同的设计效果。</span></p>
  <p class="p2"><span>下载<a title="这个页面的 HTML 源代码不能够被改动。"
href="http://www.csszengarden.com/zengarden-sample.html">HTML 文档</a> 和 <a
title="这个页面的 CSS 样式表文件，你可以更改它。"
href="http://www.csszengarden.com/zengarden-sample.css">CSS 文件</a>。</span></p>
</div>
```

类样式帮助设计师节省了大量的编写 CSS 代码的工作，加快了开发速度，同时，利用子选择符就不需要为每个标签定义 id 属性值或者类名。只要知道它所在的模块，利用模块的 id 值加包含标签即可准确定义该模块下对应标签的样式。例如，如果要控制<div id="quickSummary">模块下的段落样式，使用如下子选择符即可：

```
# quickSummary p { }
```

如果要控制第 1 段中 span 标签的样式，则使用如下子选择符即可：

```
# quickSummary p1 span { }
```

总之，这样设计的最终目的是：用最简洁明了的结构，实现更完整精确的样式控制。整个页面的结构效果如图 7.31 所示。

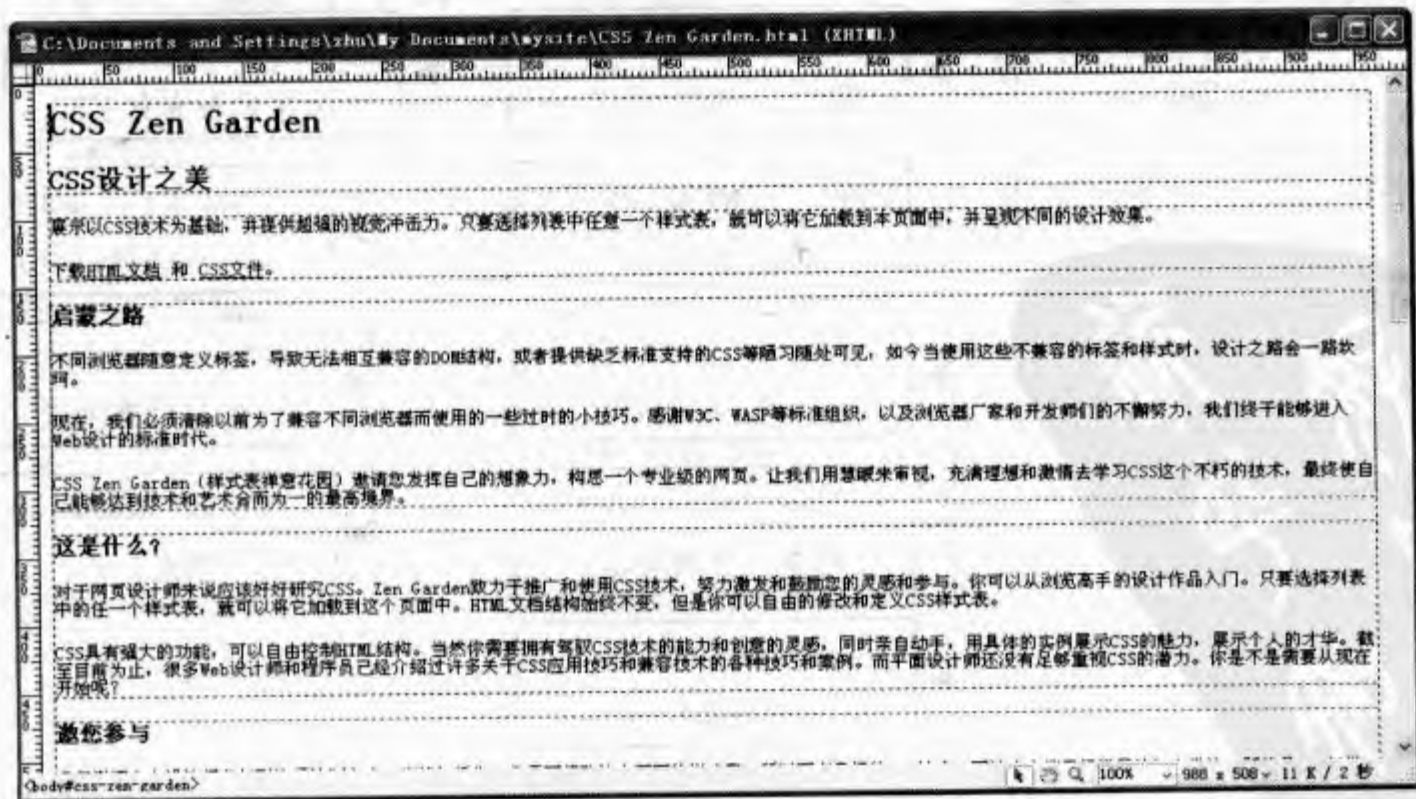


图 7.31 禅意花园网页结构效果

7.3 CSS 浮动布局基础

如果仔细看看网上的所有网页，基本上它们都是以浮动布局的方式来实现的。浮动布局在网页设计中被广泛应用，得益于它的灵活性和实用性，但这种灵活性，也导致了使用浮动布局时会出现很多意想不到的结果，加上不同浏览器对于浮动布局解析的不一致，有时会让设计师大伤脑筋。为了帮助初学者快速、安全地设计浮动布局，从本节开始我们将详细讲解这个技术难题。

7.3.1 认识网页布局类型

在第3章中我们曾经讲解了 Dreamweaver CS3 所提供的 CSS 布局模板。在这些模板中，Dreamweaver 把网页布局分为固定宽度、弹性宽度、液态宽度和混合宽度 4 种类型。这种分类方法是根据网页的易用性来确定的。

还有的设计师根据网页的版面结构，把网页布局分为一行一列、两行三列、三行三列等不同的结构块。但是不管怎么分，对于设计师来说都应该考虑网页的易用性和可读性，怎样使自己设计的网页更能够适应不同的屏幕显示器，如手机屏幕（类似如 128px×160px）、传统电脑显示屏（640px×480px），到现在的液晶显示屏（15 英寸的为 1280px×800px，30 英寸的为 5000000px×4300000px）。

但是从网页布局的解析方式来考察，实际上网页布局主要包括 3 种：自然布局、浮动布局和定位布局。

- 自然布局就是根据标签在网页中的排列顺序，自动地从上到下进行解析和显示。
- 浮动布局不再完全根据标签在网页中的排列顺序，而是根据标签的显示属性来决定它的解析和显示顺序和位置。
- 定位布局是一种用模拟图像定位的方法来解析和显示标签在屏幕上的位置和大小，它不再遵循标签在网页结构中的位置关系和排列顺序，它完全以精确到像素的程度来解析和显示标签。

本章将重点讲解浮动布局的一般设计方法，下一章将重点讲解定位布局的设计方法。

7.3.2 准备有趣的模块浮动游戏

在 CSS 中，元素的浮动是通过 float 属性来实现的。它犹如一个热气球，把绑定了该属性的元素从自然排列的标签队伍中硬托起来，使其浮动到网页的“空中”。

float 属性包含 3 个值：left（浮向左侧）、right（浮向右侧）和 none（禁止浮动）。其实我曾一厢情愿地认为：CSS 所提供的 float 属性值有点少，如果再增加诸如 top（向上浮动）、center（向中间浮动）、bottom（向底部浮动）等属性值就更好了，这并非不可能，实际上是完全可以实现的技术。

下面就让我们做一个简单的布局游戏吧，相信通过这样一个简单游戏能够帮助你很快掌握浮动布局的基本技巧。

话说有这样 3 个兄弟（3 个模块）：

```
<div id="box1">模块 1</div>
<div id="box2">模块 2</div>
<div id="box3">模块 3</div>
```

它们长得很相像（定义相同的样式），按照①②③的自然顺序垂直布局在页面中，如图 7.32 所示。

```
div {
    height:100px;           /* 模块高度 */
    color:white;            /* 包含文本的字体颜色 */
    text-align:center;      /* 包含文本水平居中 */
    line-height:100px;     /* 包含文本垂直居中 */
}
```

为了区别它们，我们先分别为 3 个兄弟设置不同的背景色：

```
#box1 { background:red; } /* 红色背景 */
#box2 { background:blue; } /* 蓝色背景 */
#box3 { background:green; } /* 绿色背景 */
```

现在让它们按着①②③的顺序水平布局在页面中（如图 7.33 所示），则设计浮动样式如下：

```
div {
    float:left;             /* 全部向左浮动 */
    height:300px;          /* 调整模块高度 */
    width:150px;           /* 调整模块宽度 */
}
```



图 7.32 ①②③自然垂直布局

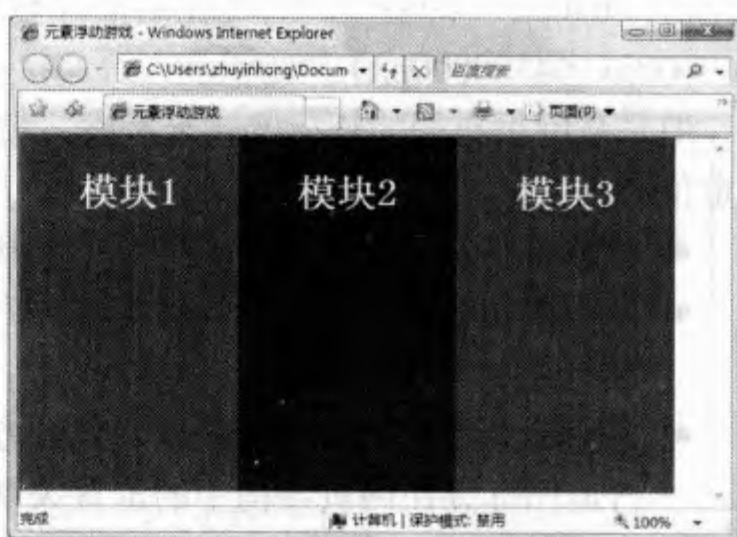


图 7.33 ①②③水平布局

7.3.3 ①②③顺序的水平布局

改变它们的水平排列顺序，按着②③①的顺序水平布局在页面中（如图 7.34 所示），则在①②③的顺序水平布局基础上，设计模块 1 向右浮动，增加样式如下：

```
#box1 {
    float:right;           /* 调整模块 1 向右浮动 */
}
```

为了美观，适当调整了 3 个模块的宽度，但是你可以看到，这种布局的中间很容易出现一道缝隙，影响美观，在后面示例中我们将讲解如何解决这道缝隙。

7.3.4 ③②①顺序的水平布局

继续游戏。也许你想按着③②①的顺序水平布局模块（如图 7.35 所示），则可以定义 3 个模块都向右浮动，再定义模块 3 向左浮动，重新设计的样式如下：

```
div {
```



```
float:right;           /* 调整 3 个模块向右浮动 */
width:184px;           /* 调整所有模块宽度 */
height:300px;          /* 调整所有模块高度 */
}
#box3 {
  float:left;           /* 调整模块 3 向左浮动 */
}
```

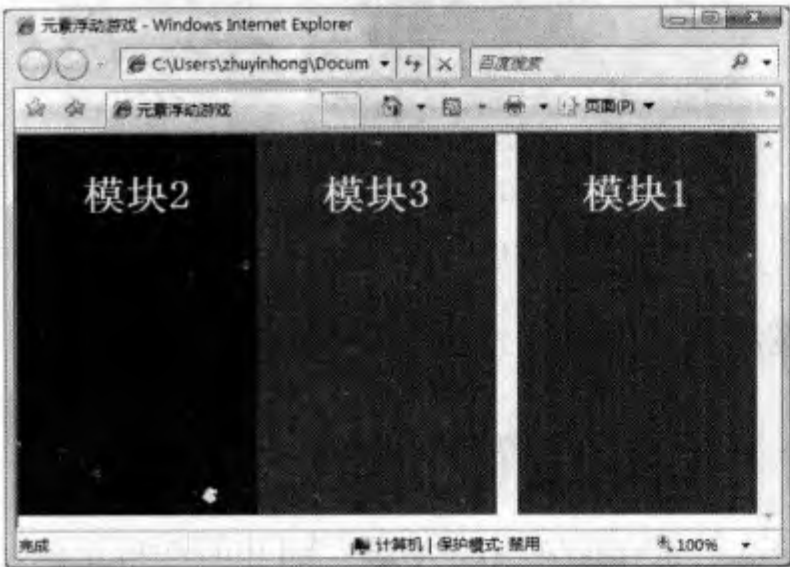


图 7.34 ②③①水平布局



图 7.35 ③②①水平布局

按照这样的思路，你还可以设计出①③②顺序的水平布局效果（如图 7.36 所示），设计样式如下：

```
div {
  float:left;           /* 调整三个模块向左浮动 */
}
#box2 {
  float:right;          /* 调整模块 2 向右浮动 */
}
```

7.3.5 ②①③顺序的水平布局

但是当游戏玩到这一关时，你也许会想到②①③顺序和③①②顺序的水平布局该如何实现呢？是不是通过为不同模块设计不同的浮动就能够实现呢？

开动脑筋想一想。如果要设计②①③顺序的水平布局，则应该让模块 2 向左浮动，然后让模块 1 和模块 3 向右浮动。但是由于模块 1 在模块 3 的前面，也就是说它们在网页中的位置是固定的，如果都向右浮动，则模块 1 先贴近最右侧，模块 3 跟在模块 1 的左侧浮动。

这个游戏的难度很大。要解决这个问题，我们不妨换位思考。你还记得我们曾经说过 margin 可以取负值吗？好，现在我们就利用 margin 取负值的方法来颠倒模块的排列顺序。

首先，来设计②①③顺序的水平布局。让所有模块都向左浮动，再让模块 3 向右浮动，形成①②③顺序的水平布局效果（如图 7.37 所示），请注意这个效果与图 7.33 所示略有不同哟。

```
div {
  float:left;           /* 调整 3 个模块向左浮动 */
}
#box3 {
  float:right;          /* 调整模块 3 向右浮动 */
}
```



图 7.36 ①③②水平布局



图 7.37 ①②③水平布局

然后设置模块 1 的左外边距值等于模块 1 的宽度值，使其位置向右移到模块的位置。

```
#box1 {
    margin-left:184px;                /* 调整模块 1 左外边距 */
}
```

这时模块 2 被迫移到模块 3 的位置。设置模块的左外边距为-368 像素，即其取值等于模块 2 被挤到模块 3 位置之后原来模块 1 和模块 2 的总宽度。

```
#box2 {
    margin-left:-368px;              /* 调整模块 2 左外边距 */
}
```

此时如果在 IE 7 或者其他标准浏览器中预览，则达到了我们的最初设想效果（如图 7.38 所示）。但是在 IE 6 及其以下版本中预览，则显示如图 7.39 所示效果。



图 7.38 在 IE 7 中②①③水平布局效果

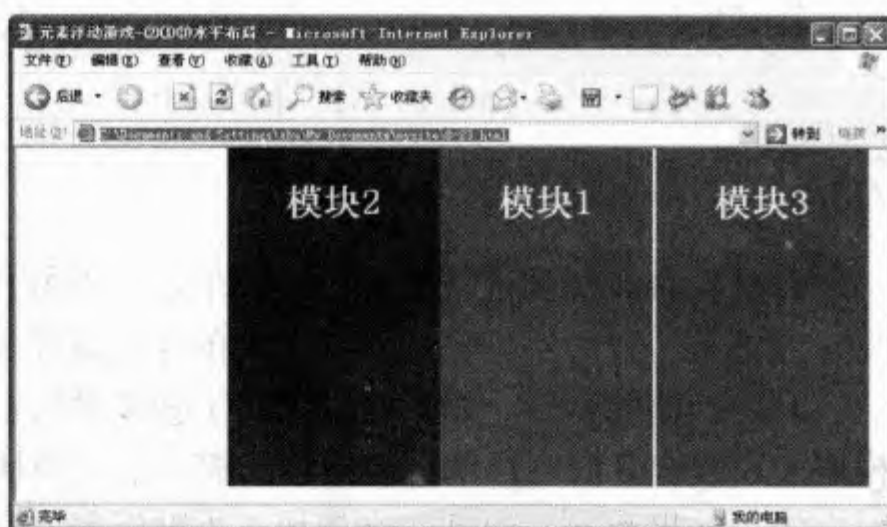


图 7.39 在 IE 6 中②①③水平布局效果

看来这种设计还存在兼容性问题。这是因为 IE 6 在解析 margin 取负值时还存在 Bug。解决的方法是在模块 1 中添加如下声明：

```
#box1 {
    margin-left:184px;
    display:inline;                /* 声明模块 1 为行内元素显示，就可以清除这个 Bug */
}
```

7.3.6 ③①②顺序的水平布局

同样的道理，如果要设计③①②顺序的水平布局，应该让模块 3 向左浮动，然后让模块 1

和模块 2 向右浮动。但是由于模块 1 在模块 2 的前面，也就是说它们在网页中的位置是固定的，如果都向右浮动，则模块 1 先贴近最右侧，模块 2 跟在模块 1 的左侧浮动。

若要设计③①②顺序的水平布局，我们可以取与②①③反向的操作，即设计如下的样式表：

```
div {
    float:right;           /* 全部右浮动 */
    width:184px;          /* 统一宽度 */
    height:300px;         /* 统一高度 */
}
#box1 {
    margin-left:-368px;    /* 模块 1 取负 2 倍宽度的左外边距 */
}
#box2 {
    margin-left:184px;     /* 模块 2 增加 1 倍宽度的左外边距 */
    display:inline;       /* 声明行内显示 */
}
#box3 {
    float:left;           /* 模块 3 向左浮动 */
}
```

上面这种设计思路完全是根据②①③顺序的水平布局取反操作，设计的效果在 IE 浏览器中能够正确地显示（如图 7.40 所示）；但是无法在非 IE 浏览器中正常预览，如图 7.41 所示（在 FF 2.0 中预览效果）。

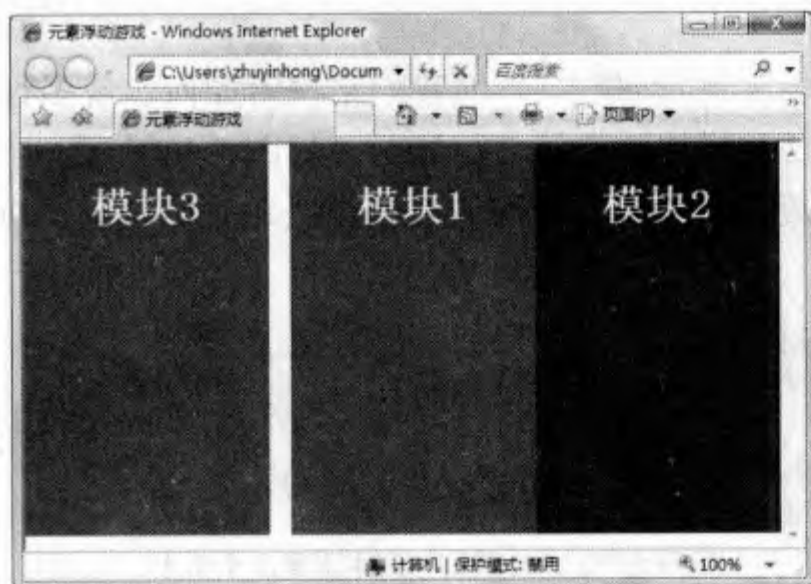


图 7.40 在 IE 7 中②①③水平布局效果

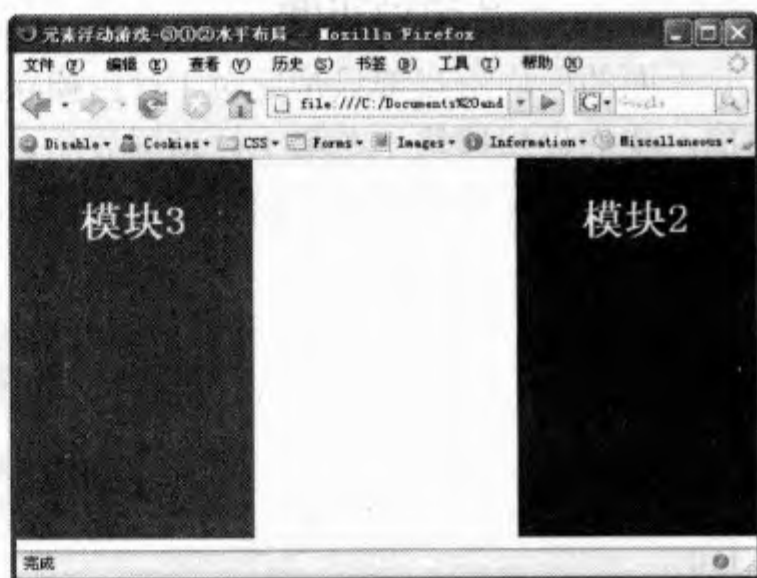


图 7.41 在 FF 2.0 中②①③水平布局效果

只有当我们按照②①③水平布局的设计思路才能够正确实现在不同浏览器中有相同的显示效果。大致设计思路是这样的：让所有元素向左浮动，再让模块 2 向右浮动，然后利用负外边距的方法调换模块 1 与模块 3 的位置即可。核心 CSS 样式表如下：

```
div {
    float:left;           /* 全部左浮动 */
    width:184px;          /* 统一宽度 */
    height:300px;         /* 统一高度 */
}
#box1 {
    margin-left:184px;     /* 模块 1 增加 1 倍宽度的左外边距 */
    display:inline;       /* 声明行内显示 */
}
#box2 {
    float:right;          /* 模块 2 向右浮动 */
}
```

```

}
#box3 {
    margin-left:-368px; /* 模块3 取负2 倍宽度的左外边距 */
}

```

看来上下样式表只不过是样式的选择符位置换了一下，所声明的属性和值还是那几个，最后在 IE 和 FF 下的显示效果如图 7.42、图 7.43 所示。



图 7.42 在 IE 7 中②①③水平布局效果



图 7.43 在 FF 2 中②①③水平布局效果

7.3.7 ①-②③结构布局

上面的模块游戏都只是针对 1 行内模块布局模式进行研究，下面我们来探索如何实现多行布局。我们设想模块 1 在第 1 行全屏显示、模块 2 和模块 3 在第 2 行显示（如图 7.44 所示）。

要实现这样的想法，我们不妨让模块 1 自然流动显示，而让模块 2 和模块 3 浮动显示，即设计如下样式：

```

div {height:150px;} /* 统一模块的高度 */
#box1 { } /* 模块1 自然流动 */
#box2 {
    width:50%; /* 模块2 宽度 */
    float:left; /* 模块2 向左浮动 */
}
#box3 {
    width:50%; /* 模块3 宽度 */
    float:left; /* 模块3 向左浮动 */
}

```

你也可以让模块 3 向右浮动，这样页面布局显得更为稳健。当然，模仿前面介绍的游戏规则，你还可以让模块 2 和模块 3 调换位置，形成①-③②结构布局样式，不过这对于你来说现在已经是小菜一碟，我也不再示例了。

也许你觉着这样玩得不够过瘾，你完全可以让 3 个模块都浮动起来，然后在中间增加一个清除属性，把它们强制切分为 2 行显示，这个话题将在下一节中讲解。

或者你还可以尝试让模块 1 和模块 2 浮动、模块 3 自然显示，则会得到①②-③结构的布局效果（如图 7.45 所示）。当然你还可以继续去构想、去实践，有趣的游戏还在后面呢。

这个游戏还没有结束，但是我不得不写上结束语了。你完全可以自定义游戏的级别和难度，不断增加模块，让多个模块在你的手中如同魔术师手中的纸牌，这是件很有意思的脑力劳动，也是值得你去不断探索的技术难题。



图 7.44 在 IE 7 中①-②③水平布局效果



图 7.45 在 IE 7 中①②-③水平布局效果

7.4 CSS 浮动布局高级技术

如果把网页设计当成玩游戏一样轻松,相信设计师的心态会年轻许多。当然现实并不完全如此,你还必须面临更多、更复杂的问题,有些问题甚至超出了个人忍耐的极限。因此,要求广大初学者在已有的基础上要不断深入研究 CSS 布局中更复杂的内容。本节将在上节基础上研究 CSS 浮动布局中的一些高级技巧,这些能够解决你在网页布局中可能要面临的问题。

7.4.1 探究浮动元素的空间大小

当网页中一个元素被定义为浮动显示时,该元素就会收缩自身体积为最小状态。

- 如果该元素被定义了高度或宽度,则元素将以该高度或宽度值所设置的大小来进行显示。
- 如果浮动元素包含了其他对象,则元素的体积会自动收缩到仅能容纳所包含对象的大小。
- 如果没有设置大小或者没有任何包含对象,浮动元素将会缩小为一个点,甚至不可见。

例如,下面这个页面包含了 6 个元素,分别让它们浮动显示。为了便于观察,定义了每个元素的边框,同时设置每个元素外边距为 6 像素。其中第 1 个元素被指定了宽度和高度;第 2 个元素包含文字,并定义了行高等于字体大小;第 3 个元素包含文字,并定义了行高等于 100 像素,目的是观察行高对浮动元素的影响;第 4 个元素包含一个图像;第 5 个元素包含为空,行高为默认值,看此时行高对于空浮动元素是否有影响;第 6 个元素不包含任何对象,同时定义行高为 0,清除任何可能影响浮动元素大小的因素,看浮动元素的大小。

```
<style type="text/css">
div {
    float:left;                /* 浮动元素 */
    margin:6px;                /* 增加外边距,便于区分 */
    border:solid 1px red;      /* 增加边框,便于查看 */
}
#box1 {
    width:100px;              /* 定义宽度 */
    height:100px;             /* 定义高度 */
}
#box2 {
```

```

        line-height:30px;          /* 与字体同高的行高 */
        font-size:30px;            /* 字体大小 */
    }
    #box3 {
        line-height:100px;         /* 超大行高 */
        font-size:30px;            /* 字体大小 */
    }
    #box5 {}                        /* 空样式 */
    #box6 {line-height:0px;}        /* 0 行高 */
</style>
<div id="box1">模块 1</div>
<div id="box2">模块 2</div>
<div id="box3">模块 3</div>
<div id="box4"></div>
<div id="box5"></div>
<div id="box6"></div>

```

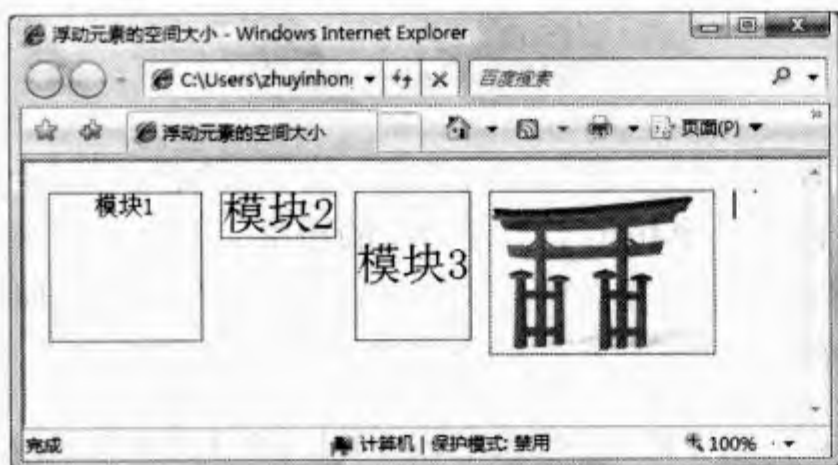


图 7.46 在 IE 7 中浮动元素的大小



图 7.47 在 FF 2 中浮动元素的大小

通过上图，你可以看出浮动元素的大小是如何显示的。但是有一点需要注意，在 IE 浏览器中即使浮动元素包含为空，默认的行高仍然会影响浮动元素的高度；如果显式定义了一个行高，也会对浮动元素产生影响。但是在非 IE 的标准浏览器中，如果浮动元素没有包含任何元素，则浏览器会认为该元素没有行存在，故显示为一个点。如果清除浮动元素的边框，则你会看不到任何内容，但是它仍然存在，并占据一个最小的空间（如同一个透明的点），这时如果在后面再增加一个浮动元素，你就会感觉到它的存在，虽然看不见它。

因此，当你准备浮动一个元素时，应该显式定义浮动元素的大小。如果元素包含有形的对象，则可以考虑不定义大小，让浮动元素紧紧包含有形的对象。利用浮动元素的这个特性，你可以为对象贴一层外包装，实现一些特殊的设计效果。

7.4.2 探究浮动元素的移动位置

当网页中一个元素浮动显示时，由于所占空间大小的收缩或者变化，同时它会在包含元素内部自动向左或者向右浮动，直到碰到包含元素（或者说父元素）的边框或内边距；或者碰到相邻浮动元素的外边距或边框时才会停下来，而不管所碰到的边框、内边距或外边距是什么类型的元素。这个有点不好理解，我们不妨看一个示例：

```

<style type="text/css">
p {
    width:90%;                /* 宽度 */
    border:solid 2px red;      /* 增加边框 */
}

```



```

</style>
<p class="p2"><span><acronym title="cascading style sheets">CSS</acronym><span
class="class1">具有强大的功能,可以自动控制 HTML 结构。</span>当然你需要拥有驾驭 CSS 技术的能力和创
意的灵感,同时亲自动手,用具体的实例展示 CSS 的魅力,展示个人的才华。<span class="class2">截至目前,
</span>很多 Web 设计师和程序员已经介绍过许多关于 CSS 应用技巧和兼容技术的各种技巧和案例。而平面设计师
还没有足够重视 CSS 的潜力。你是不是需要从现在开始呢? </span></p>

```

在上面这个文本段中, p 元素以 90% 的宽度显示。为了方便观察并被定义了粗线边框, 当定义 acronym 元素向右浮动时, 它会一直浮动到 p 元素 (包含元素) 的右边框内侧 (如图 7.48 所示)。

```

acronym {
    float:right;           /* 向右浮动 */
    background:#FF33FF;    /* 增加背景色以方便观看 */
}

```

现在, 再定义 acronym 元素后面的 也向右浮动, 则此时它就不是停靠在 p 元素的内侧边框上, 而是 acronym 元素左侧的外壁上 (如图 4.49 所示)。

```

.class1 {
    float:right;           /* 向右浮动 */
    border:solid 2px blue; /* 增加边框以方便观看 */
    height:50px;          /* 高度 */
    width:120px;          /* 宽度 */
}

```

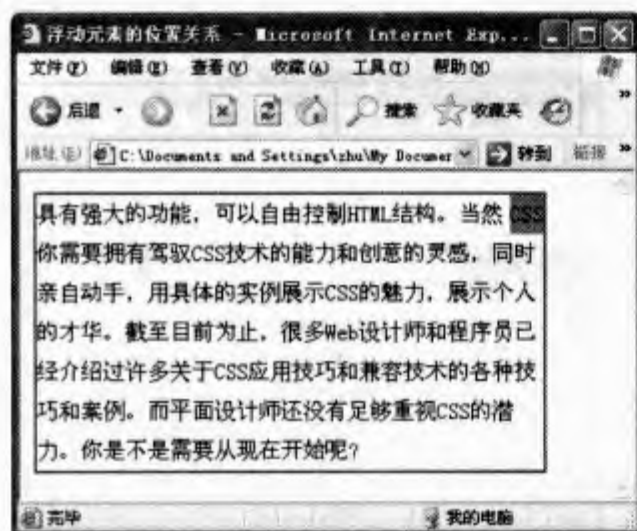


图 7.48 浮动元素停靠在包含元素的内壁上

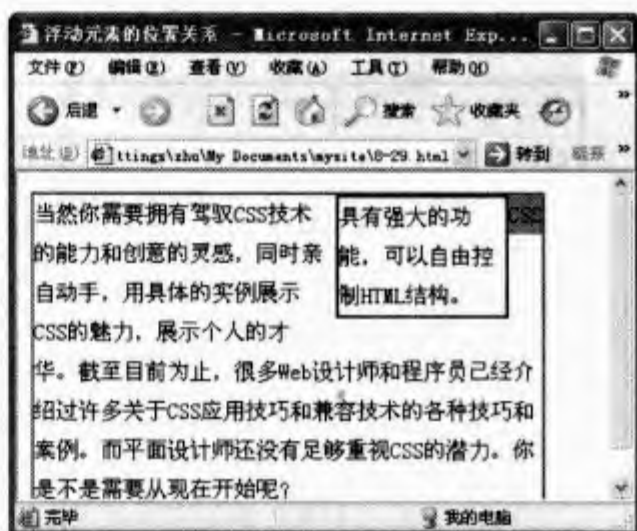


图 7.49 浮动元素停靠在相邻元素的外壁上

浮动元素在浮动时会遵循向左右平行浮动, 或者向左右下错平行浮动的规则, 决不会在当前位置基础上向上错移到左右边。

例如, 针对上面文本段, 我们定义 元素所包含的文本向左浮动:

```

.class2 {
    float:left;           /* 向左浮动 */
    background:#FF33FF;    /* 加背景色以方便观看 */
}

```

这时你会看到, 如果 元素左侧没有任何文本, 则它会直接平移到左侧的 p 元素内壁上 (如图 7.50 所示)。但是如果其左侧有其他文本或对象, 则该浮动元素会向下错一行, 再向左平移 (如图 7.51 所示)。

了解浮动元素的移动规律, 对于网页布局非常重要。虽然说浮动元素能够自由浮动, 但是并不等于它会随意移动。它是在原有行位置的基础上左右移动, 因此要调整浮动元素在网页布

局中的上下位置关系，特别是希望浮动元素调整到上面显示，建议你调整网页结构中浮动元素与其他元素之间的位置关系。

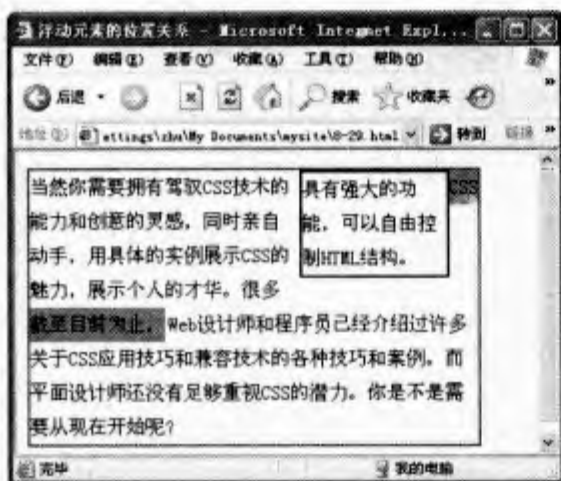


图 7.50 浮动元素平移

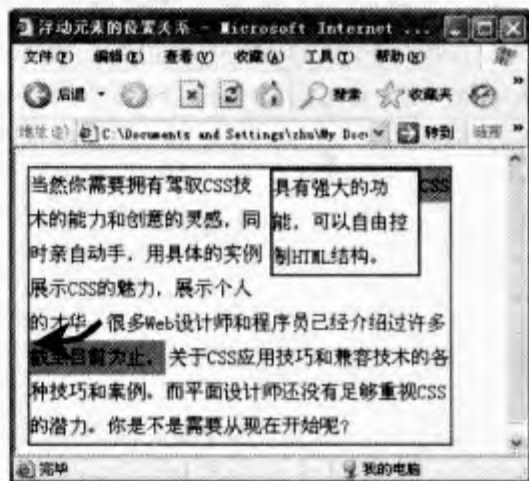


图 7.51 浮动元素下错移动

你可以使用外边距取负值的方法来把浮动移动到上面。例如，针对上面的示例，为元素定义一个负外边距值：

```
.class2 {
    margin-top:-100px;          /* 通过取负外边距值，来强迫浮动元素向上移动 */
}
```

这时你可以看到浮动元素跑到上面去，但是它在不同浏览器中显示效果是不同的。在 IE 下浮动元素遮盖了其他对象（如图 7.52 所示），而在 FF 下却被其他对象遮盖了（如图 7.53 所示）。

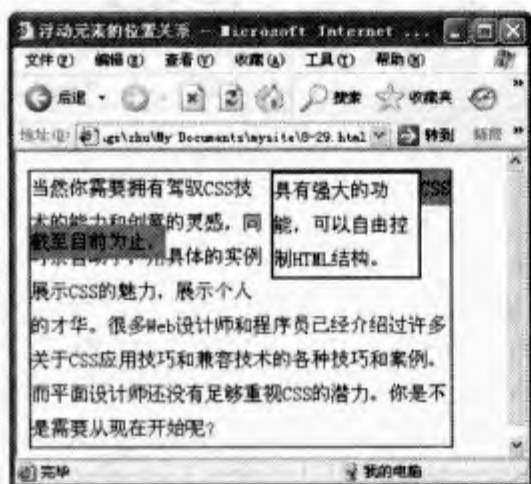


图 7.52 IE 6 中浮动元素向上移动

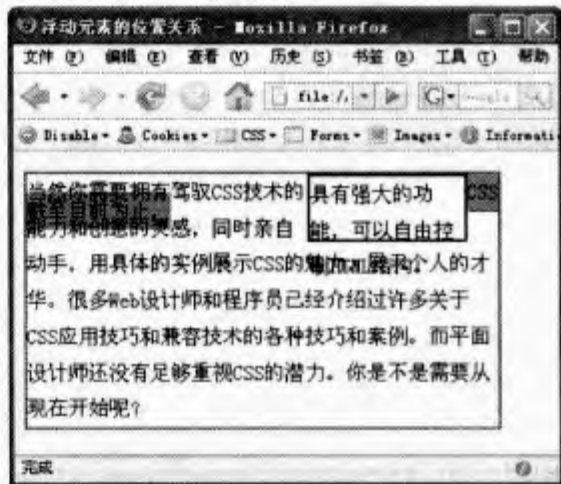


图 7.53 FF 2 中浮动元素向上移动

7.4.3 探究浮动元素的环境关系

当元素浮动之后，它原来的位置就会被下面的对象上移填充掉。这时上移对象会自动围绕在浮动元素周围，形成一种环境关系。

例如，对于如下的结构，你会看到段落文本自动环绕在浮动元素的右侧，虽然 p 元素是一个块状元素（如图 7.54 所示）。

```
<style type="text/css">
#box1 {
    width:100px;          /* 宽度 */
    height:100px;         /* 高度 */
    border:solid 4px blue; /* 边框 */
    float:left;           /* 向左浮动 */
}
```



```
p { border:solid 2px red;} /* 段落边框 */
</style>
<div id="box1">浮动元素</div>
<p class="p2"><span><acronym title="cascading style sheets">CSS</acronym><span class="class1">具有强大的功能，可以自由控制 HTML 结构。</span>当然你需要拥有驾驭 CSS 技术的能力和创意的灵感，同时亲自动手，用具体的实例展示 CSS 的魅力，展示个人的才华。<span class="class2">截至目前，</span>很多 Web 设计师和程序员已经介绍过许多关于 CSS 应用技巧和兼容技术的各种技巧和案例。而平面设计师还没有足够重视 CSS 的潜力。你是不是需要从现在开始呢？</span></p>
```

此时你可以通过调整浮动元素的外边距来调整它与周围环境对象的间距，如输入下面样式，则可以得到如图 7.55 所示的效果：

```
#box1 {
    margin:16px; /* 调整浮动元素的外边距 */
}
```

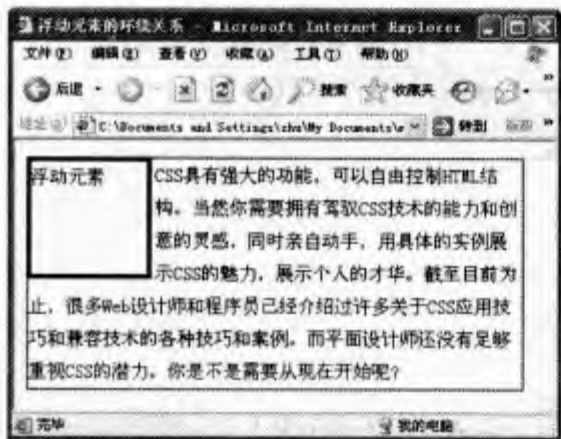


图 7.54 IE 6 中浮动元素的环境关系

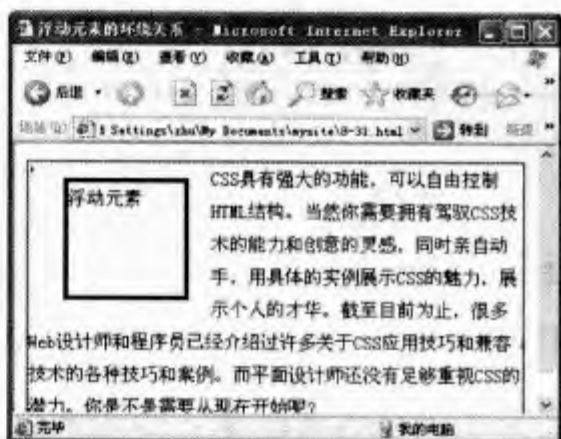


图 7.55 调整浮动元素与周围环境对象的间距

但是要注意，如果你想通过设置 p 元素（环绕对象）的内边距或者外边距来调整环绕对象与浮动元素之间的间距，则要确保外边距或内边距的宽度大于或等于浮动元素的总宽度。这种设计虽然效果相同，但是如果环绕对象包含有边框或者背景，所产生的效果就截然不同了。

例如，针对上面示例清除浮动元素的外边距，然后为浮动对象 p 元素定义一个左内边距和背景图像，则会显示图 7.56 所示的效果。但是如果把左内边距改为左外边距，则所得效果如图 7.57 所示。一个设置的差别可能对网页布局产生重大影响，所以请读者务必注意这些小的细节问题。

```
p {
    padding-left:120px; /* 调整环绕对象左内边距 */
    border:solid 2px red; /* 环绕对象的边框 */
    background:url(images/bg4.jpg); /* 背景图像 */
}
```

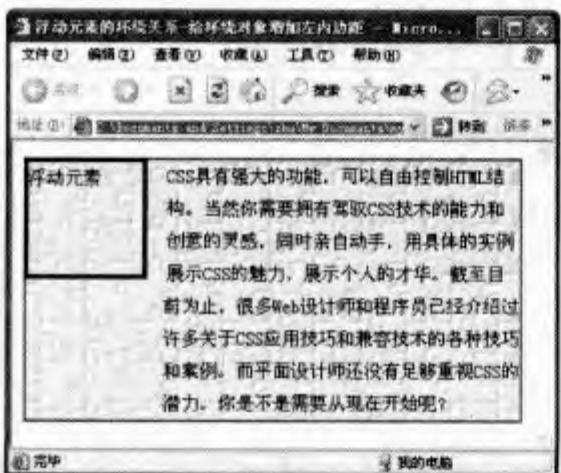


图 7.56 IE 6 中调整环绕对象左内边距

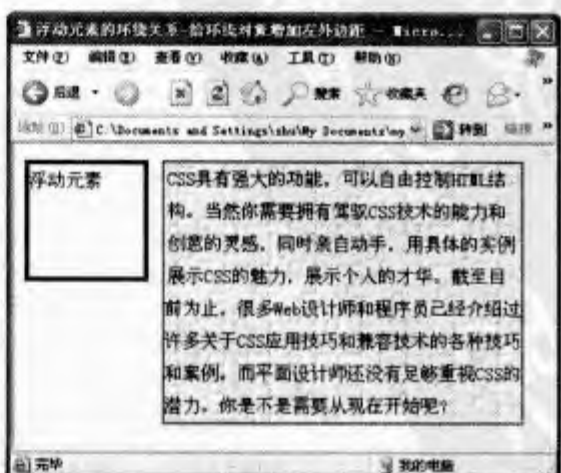


图 7.57 IE 6 中调整环绕对象左外边距

如果当你为 p 元素定义宽度和高度之后，浏览器的解析效果就会发生分歧，例如，在 IE 6 版本下的解析效果如图 7.58 所示，在 FF 2 下的解析效果如图 7.59 所示。此时如果要定义环绕元素的边框或者外边距，则要避免使用这种方法来布局；可以改为设置它浮动显示，这样就能够解决浏览器兼容问题了。

```
p {
    width:300px;           /* 宽度 */
    height:160px;         /* 高度 */
}
```

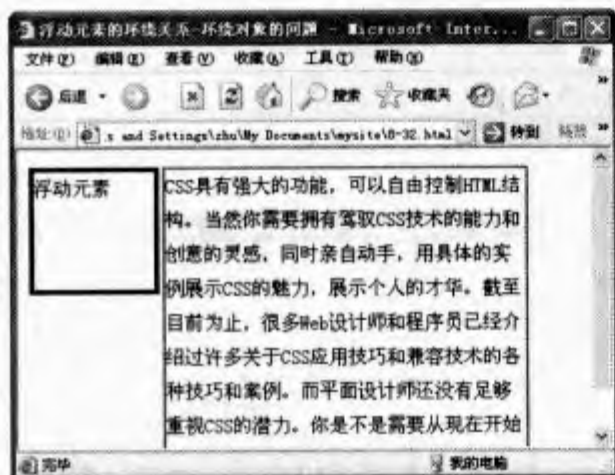


图 7.58 IE 6 中浮动元素的环境关系

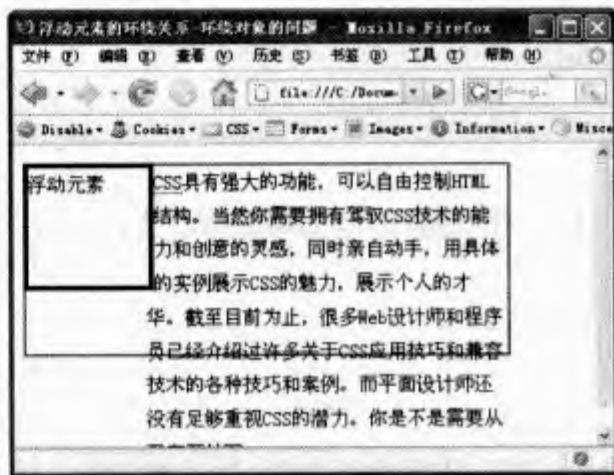


图 7.59 FF 2 中浮动元素的环境关系

有一点还需要读者注意，当元素被定义为浮动显示时，它会自动成为一个块状元素，相当于定义了 `display:block`。但是块状元素会自动伸展宽度，占据一行的位置，且块状元素会附加换行符，所以在同一行内只能够显示一个块状元素。而浮动元素虽然拥有块状元素的特性，但是它没有上述表现，这使它更像行内元素那样收缩为一团。清楚这个特性，对于布局具有重要作用。

7.4.4 浮动与清除

正如克隆羊一样，小小的 `float` 属性改变了元素的显示基因。在自然状态下，网页元素都会很自然地遵循固有的位置按顺序显示在网页上。现在浮动技术打破了这种“生态平衡”，于是各种布局问题接踵而至。CSS 为了解决这个问题，又定义了 `clear` 属性，希望使用这个属性来清除浮动布局中页面杂乱无章的局面。

例如，在下面这个简单的 3 行 3 列页面结构中，设置中间 3 栏平行浮动显示。根据下面的样式，可以显示图 7.60 所示的内容。

```
<style type="text/css">
div {
    border:solid 1px red;           /* 增加边框，以方便观察 */
    height:50px;                   /* 固定高度，以方便比较 */
}
#left,#middle,#right {
    float:left;                   /* 定义中间 3 栏向左浮动 */
    width:33%;                    /* 定义中间 3 栏等宽 */
}
</style>
<div id="header">头部信息</div>
<div id="left">左栏信息</div>
<div id="middle">中栏信息</div>
<div id="right">右栏信息</div>
<div id="footer">脚部信息</div>
```


但是如果设置左栏高度大于中栏和右栏高度，你会发现脚部信息栏上移并环绕在左栏右侧（如图 7.61 所示）。

```
#left {height:100px; }
```

```
/* 定义左栏高出中栏和右栏 */
```



图 7.60 IE 6 中浮动布局效果



图 7.61 调整部分栏目高度后发生的错位现象

这种环绕包含浮动元素的现象当然不是我们所希望的，浮动布局所带来的影响由此可见一斑。这时 `clear` 属性就可以派上用场了，为 `<div id="footer">` 元素定义一个清除样式：

```
#footer {
    clear:left;
}
```

```
/* 为脚部栏目元素定义清除属性 */
```

这时如果你在浏览器中预览，则又恢复到预想的 3 行 3 列布局效果，如图 7.62 所示。下面我们就来研究一下 `clear` 属性的基本用法和注意问题。

`clear` 属性被用来清除元素左侧、右侧或左右两侧浮动元素。该属性取值包括 `left`（清除左侧浮动元素）、`right`（清除右侧浮动元素）、`both`（清除左右两侧浮动元素）和 `none`（不清除浮动元素）。这个概念比较艰涩，初学者一般不好理解。

首先，`clear` 属性是专门针对 `float` 属性而设计的，因此仅对左右两侧浮动元素有效，对于非浮动元素是无效的。例如，对于两个并列显示的行内元素（如下所示），不管为 `span` 元素，还是为 `img` 元素定义了清除属性，都不能够把它们分成两行显示。

```
<span>行内文本</span>
```

请注意，上面所谓的浮动元素是指与元素相邻的左右两侧的对象，而不是定义 `clear` 属性的对象自身。对于非浮动元素来说仍然可以使用 `clear` 属性，并影响它的显示位置（如图 7.61 所示）。

其次，应该明确这里的清除不是清除别的浮动元素，而是清除自身，通俗说就是不允许当前元素与浮动元素并列显示。如果左右两侧存在有浮动元素，则当前元素就把自己清除到下一行显示；而不是把前面的浮动元素清除走，或者清除到上一行显示。

最后，根据 HTML 解析规则，当前元素前面的对象（已经被解析）不会再受后面元素属性的影响，但是当前元素能够根据前面对象的显示属性，来决定自身的显示位置，这就是 `clear` 属性的作用。同样的道理，不管为当前元素设置怎样的清除属性，后面的对象不会受到影响。

例如，针对上面的示例，定义左栏、中栏和右栏包含如下的样式：

```
#left,#middle,#right {
    clear:right;
}
```

```
/* 清除右侧浮动元素 */
```

虽然说左栏栏目右侧有浮动元素，中间栏目右侧也有浮动元素，但是这些浮动都是在当前元素的后面，所以 `clear:right` 规则就不会对它们产生影响（如图 7.63 所示）。



图 7.62 IE 6 中浮动布局清除效果

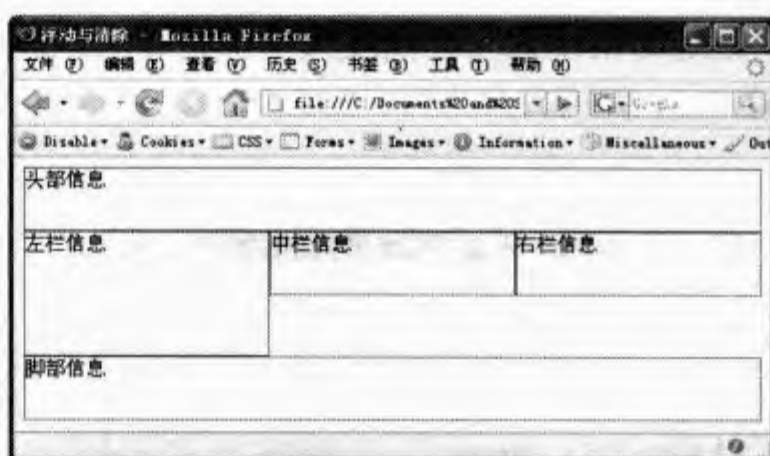


图 7.63 FF 2 中清除右侧浮动元素效果

7.4.5 浮动元素包含其他对象的问题及其解决方法

由于浮动布局的复杂性，下面我们分节探讨浮动布局与不同显示性质的元素之间的各种关系以及处理方法。

所谓元素之间的关系无非就是这么两种情况：包含关系（或者称为父子关系）和并列关系（或者称为相邻关系）。

浮动元素能够很好地包含任何行内元素、块状元素或者其他浮动元素，这种包含关系在不同浏览器中都能够很好地被解析和显示，且解析效果基本相同。但是如果元素包含对象的大小超出了浮动元素的大小，则在不同浏览器中就会出现不同的情况。

例如，在下面这个浮动的 p 元素中包含了一个行内显示的 span 元素。但是由于 span 元素包含的文本超出了 p 元素的大小，在 IE 6 及其以下版本中显示如图 7.64 所示，浮动元素将自动调整大小来适应文本区域；而在 IE 7 和其他标准浏览器中显示如图 7.65 所示，浮动元素依然按着自己的大小显示，而超出的文本会显示到浮动元素的外边。

```
<style type="text/css">
p {
    border:solid 2px red;          /* 边框 */
    float:left;                   /* 浮动显示 */
    width:260px;                  /* 固定宽度 */
    height:100px;                 /* 固定高度 */
}
span {background:#FF99FF;}      /* 行内元素背景色 */
</style>
```

```
<p class="p3"><span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力，构思一个专业级的网页。让我们用慧眼来审视，充满理想和激情去学习 CSS 这个不朽的技术，最终使自己能够达到技术和艺术合而为一的最高境界。</span></p>
```

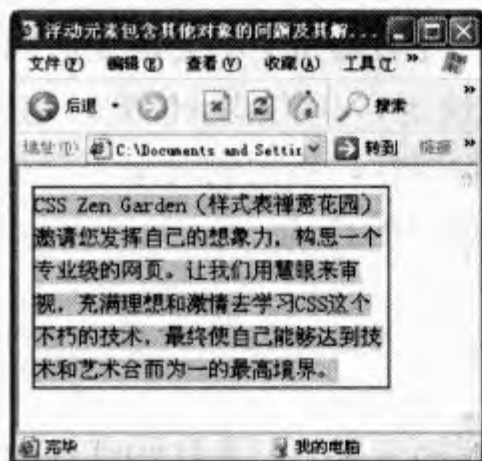


图 7.64 IE 6 中效果

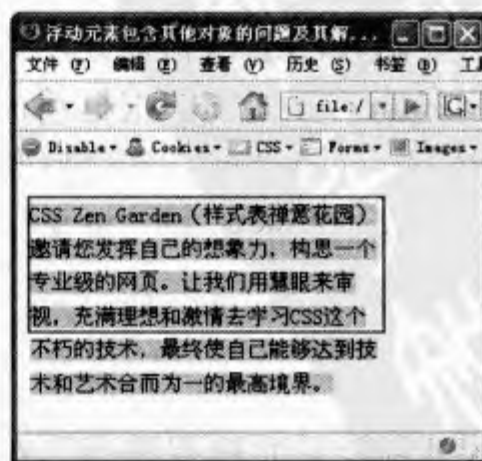


图 7.65 FF 2 中效果

解决此类问题的方法如下。

由于 IE 6 及其以下版本浏览器不支持 min-height 和 min-width 属性, 因此我们可以为 IE 7 以及其他标准浏览器定义该属性, 强制它们显示为最小显示大小, 这样 IE 7 以及其他标准浏览器就能够随包含内容大小不断调整自己的大小。具体兼容样式如下:

```
p {
    min-height:100px;           /* 最低显示高度 */
}
* html p { /* 兼容 IE 6 及其以下版本浏览器 */
    height:100px;             /* 显示高度 */
}
```

上面样式的设计思路是这样的: 对于 IE 6 及其以下版本浏览器来说, 由于它们不认识 min-height 属性, 因此就省略了这个声明, 而执行 *html p { } 样式。同样的道理, 由于 IE 7 以及其他标准浏览器不认识 “* html p” 这样复杂的选择符, 所以也就是无法解析 height:100px; 这个声明。此时如果再在 FF 2.0 中预览, 显示效果如图 7.66 所示。

7.4.6 浮动元素被其他对象包含的问题及其解决方法

反过来说, 如果浮动元素被其他对象包含时会出现什么情况呢? 在上面示例的基础上我们不妨做一个试验。让 span 元素浮动显示, 而禁止 p 元素浮动显示, 具体代码如下:

```
<style type="text/css">
p {
    border:solid 2px red;           /* 定义包含框的边框 */
}
span {
    float:left;                   /* 子元素浮动显示 */
    width:80%;                    /* 显示宽度 */
    background:#FF99FF;           /* 背景色 */
}
</style>
```

```
<p class="p3"><span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力, 构思一个专业级的网页。让我们用慧眼来审视, 充满理想和激情去学习 CSS 这个不朽的技术, 最终使自己能够达到技术和艺术合而为一的最高境界。</span></p>
```

这时你可以看到在任何浏览器中都会显示图 7.67 所示的效果。父元素(浮动元素的包含框)自动收缩为一条直线, 该直线为它的边框。由于这种布局结构和方式在网页布局中比较实用, 所以这个问题在实践中会经常遇到, 对于没有经验的初学者来说往往会让人不知所措。

解决这个问题的方法如下。

方法一, 我们可以模仿上节示例的解决方法, 分别为不同浏览器定义不同的显示样式, 实现浏览器的兼容显示。

对于 IE 6 及其以下版本来说, 只要包含框拥有了高度, 则它就能够自动调整自身的高度来适应所包含的对象。因此, 我们可以在 IE 6 及其以下版本中定义 p 元素高度为 1 像素。

```
* html p { /* 兼容 IE 6 及其以下版本浏览器 */
    height:1px;           /* 显式定义高度 */
}
```



图 7.66 经过处理后在 FF 2.0 中效果

对于 IE 7 版本浏览器来说, 可以使用 `min-height` 属性定义 `p` 元素的最低高度为 1 像素, 这样它就能够自动调整高度来实现包含其他对象。

```
p { /* 兼容 IE 7 版本浏览器 */
    min-height: 1px; /* 定义最低高度 */
}
```

对其他标准浏览器 (如 FF 等), 则可以定义如下样式来强制包含框调整自身高度, 以实现包含对象。

```
p:after { /* 兼容标准浏览器 */
    content: ""; /* 增加显示内容 */
    display: block; /* 定义显示内容的显示属性 */
    height: 0; /* 定义高度为 0, 强制隐藏 */
    clear: both; /* 清除浮动 */
}
```

在上面样式中, `p:after` 选择符只能够被标准浏览器所支持, 它表示在 `p` 元素的后面增加显示内容; 然后使用 `content` 属性声明在 `p` 元素最后显示一个空内容, 并定义为块状显示, 这样才能够准确控制; 再使用 `height` 属性声明该空行的高度为 0, 即强制隐藏它的显示; 最后为这个空行元素增加一个 `clear` 属性, 以强迫撑开包含框。最终的显示效果如图 7.68 所示。

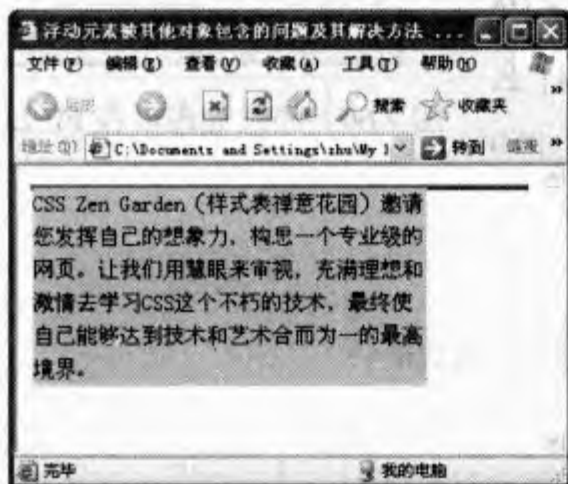


图 7.67 在 IE 6 中显示效果

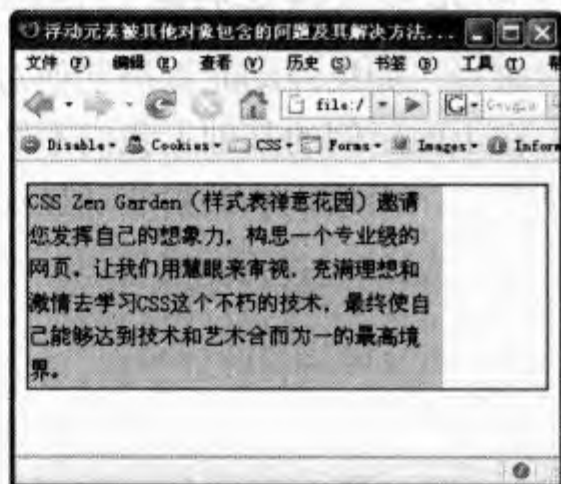


图 7.68 调整后在 FF 2 中显示效果

方法二, 如果你觉得第一种方法比较麻烦, 甚至难以理解和接受, 那么这种方法是最简单的, 但是需要你调整 HTML 的结构, 也就是人工在包含块的末尾增加一个清除元素。另外, 由于 `p` 元素无法兼容这种方法, 所以还必须把 `p` 元素替换为 `div` 元素。完整的代码如下:

```
<style type="text/css">
.p3 {
    border: solid 2px red; /* 定义包含框边框 */
}
span {
    float: left; /* 向左浮动 */
    width: 80%; /* 宽度 */
    background: #FF99FF; /* 背景色 */
}
.clear { /* 定义清除类 */
    clear: both; /* 清除浮动 */
}
</style>
```

```
<div class="p3"><span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力, 构思一个专业级的网页。让我们用慧眼来审视, 充满理想和激情去学习 CSS 这个不朽的技术, 最终使自己能够达到技术和艺术合而为一的最高境界。</span>
```



```
<div class="clear"></div>
</div>
```

方法三, 如果你觉得上面的方法还是比较麻烦, 那么建议你定义包含框浮动显示。这种包含框会自动调整大小, 包含所有子对象。这种方法虽然比较简单, 但是它改变了包含框的显示性质, 会影响到其他版面布局。所以, 除非万不得已, 建议不要使用该方法。

7.4.7 浮动布局中元素垂直间距问题以及解决方法

在第 7.1 节中我们曾经讲解了 CSS 盒模型, 随便提及到元素之间的间距问题。下面就这个问题专门讲解一下。

在正常情况下, 由于行内元素的外边距不影响任何元素, 因此我们可以忽略它的存在。这时文本行的间距只能够通过行高来调节, 而其他类型的元素与行内元素的间距可以通过其他元素的边距来调节。

例如, 对于如下两个相邻元素, span 以行内显示, div 以默认的块状显示。

```
<span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力, 构思一个专业级的网页。让我们用慧眼来审视, 充满理想和激情去学习 CSS 这个不朽的技术, 最终使自己能够达到技术和艺术合而为一的最高境界。
</span>
<div class="box">参考元素</div>
```

如果要调整它们的间距, 只有通过定义 div 元素的边距来实现 (如图 7.69 所示)。但是如果在下面样式中增加 display:inline; 声明, 让它行内显示, 则显示效果如图 7.70 所示。

```
.box {
    width:200px;           /* 宽度 */
    height:50px;           /* 高度 */
    background:#FF66CC;    /* 背景 */
    margin-top:40px;        /* 顶部外边距 */
}
```

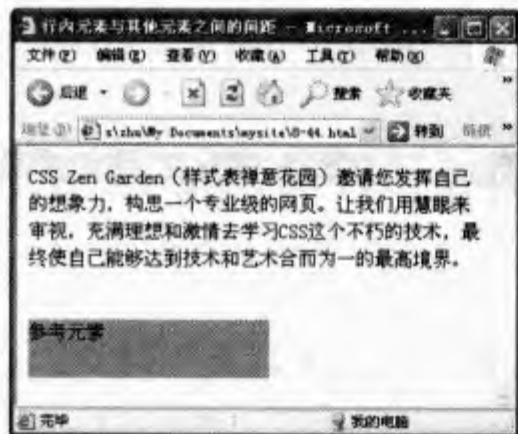


图 7.69 行内元素与块状元素间距

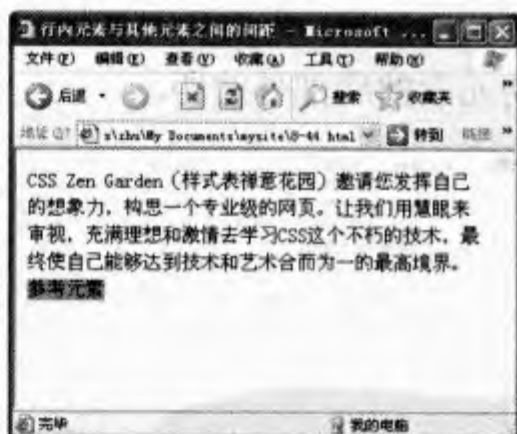


图 7.70 行内元素与行内元素间距

对于块状元素与块状元素来说, 一般会有存在边界重叠现象。例如, 针对上面的示例, 如果定义 span 元素块状显示, 则上下元素之间的间距为 40 像素; 但是如果为 span 元素再定义 40 像素的顶部外边距, 这时你会发现它们的间距并没有改变, 依然为 40 像素。

```
span {
    display:block;           /* 转换为块状显示 */
    border:solid 1px blue;   /* 边框线 */
    margin-bottom:40px;      /* 底部外边距 */
}
```

这说明上下块状元素的外边距会发生重叠。重叠的幅度是上下元素的外边距中最小的那

边。例如，如果上边元素的外边距为 40 像素，下边元素的外边距为 20 像素，则它们的间距为 40 像素，重叠大小为 20 像素；如果上边元素的外边距为 40 像素，下边元素的外边距为 60 像素，则它们的间距为 60 像素，重叠大小为 40 像素。

对于浮动元素与浮动元素来说，它们之间不会存在边界重叠问题。例如，针对上面示例，如果定义上下元素都向左浮动，则它们之间的垂直间距为上下外边距的和，即为 80 像素。

浮动元素与块状元素之间间距为上下外边距之和。但是如果浮动元素在上面，块状元素在下面，由于浮动环绕关系将使得它们之间的间距变得很复杂。对于 IE 浏览器来说，不管上下位置关系如何，它们的间距仍然为上下外边距的和（如图 7.71 所示）。而在其他版本浏览器中，由于在解析浮动环绕问题上的差异，它们之间的间距就变得很复杂了。

为了更直观地说明这个问题，我们举一个简单的示例。例如，针对上面的 HTML 结构，定义如下的样式：

```
span {
    border:solid 1px blue;           /* 边框 */
    margin-bottom:40px;             /* 底部外边距 */
    display:block;                  /* 转换为块状显示 */
    float:left;                     /* 设置为浮动显示 */
}
.box {
    width:200px;                   /* 宽度 */
    height:50px;                   /* 高度 */
    background:#FF66CC;            /* 背景色 */
    margin-top:40px;               /* 顶部外边距 */
}
```

如果在 FF 2 版本浏览器中解析，则显示效果如图 7.72 所示。上下间距实际上仅取浮动元素的外边距，而下边的块状元素的顶部外边距和背景色都跑到了页面顶部，出现块状元素与内容分离的状态。因此读者在设计此类的网页布局时一定要小心。

解决的方法是：为块状元素增加 clear 属性。强制其不要越过浮动元素，但是它们的间距仍然为浮动元素的底部外边距。

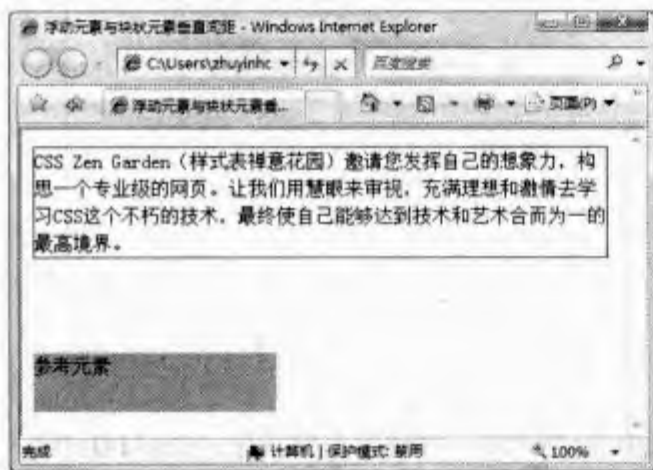


图 7.71 IE 7 中浮动元素与块状元素间距



图 7.72 FF 2 中浮动元素与块状元素间距

对于浮动元素与行内元素之间的间距，由于行内元素的外边距不影响任何元素，因此它们之间的垂直距离可以取浮动元素的外边距。

7.4.8 浮动布局中元素水平间距问题以及解决方法

浮动布局中元素水平间距没有垂直间距复杂，一般不会出现边界重叠现象。但是由于 IE

浏览器的 Bug，特别是 IE 6 及其以下版本浏览器的 Bug 太多，从而产生很多布局的兼容问题。下面我们就针对这个问题进行讲解。

如果一个浮动元素定义了外边距，则浮向一边的外边距会加倍显示。这个问题在 IE 6 及其以下版本浏览器中存在。例如，尝试输入下面的示例代码：

```
<style type="text/css">
body { /* 清除页边距 */
    padding:0; /* 清除标准浏览器页边距 */
    margin:0; /* 清除 IE 页边距 */
}
div { /* 公共样式 */
    margin-left:50px; /* 左侧外边距 */
    width:200px; /* 宽度 */
    height:100px; /* 高度 */
    border:solid 1px red; /* 边框 */
}
.box2 {
    float:left; /* 向左浮动 */
}
</style>
<div class="box1">参考元素</div>
<div class="box2">浮动元素</div>
```

如果分别在 IE 6 和 IE 7 中预览，则显示效果如图 7.73、图 7.74 所示。你可以很直观地看到两者解析的差异。



图 7.73 IE 6 中解析效果



图 7.74 IE 7 中解析效果

此类问题的解决比较简单，你只需要在浮动元素中增加 `display:inline;` 规则即可：

```
.box2 {
    float:left; /* 浮动显示 */
    display:inline; /* 行内显示 */
}
```

此时 `display:inline;` 规则不能够改变浮动元素的显示状态，仅起到消除 IE 浏览器的 Bug 的作用，浮动元素依然显示为一个块状元素。

如果当浮动元素与一个非浮动元素并列显示时，则元素之间会多出 3 个像素的缩进空隙。这个问题在 IE 6 及其以下版本浏览器中存在。例如，尝试输入下面的示例代码：

```
<style type="text/css">
div {
    width:140px; /* 宽度 */
```

```

height:100px;          /* 高度 */
border:solid 1px red;   /* 边框 */
float:left;             /* 向左浮动 */
}
p {
    margin-left:140px;    /* 增加左外边距 */
}
</style>
<div>浮动元素</div>

```

<p>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力, 构思一个专业级的网页。让我们用慧眼来审视, 充满理想和激情去学习 CSS 这个不朽的技术, 最终使自己能够达到技术和艺术合而为一的最高境界。</p>

这时如果你在 IE 6 及其以下版本浏览器中预览, 则显示效果如图 7.75 所示。如果在标准浏览器中预览, 则显示效果如图 7.76 所示。

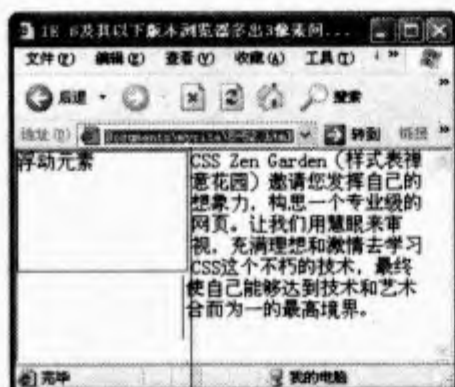


图 7.75 IE 6 中解析效果

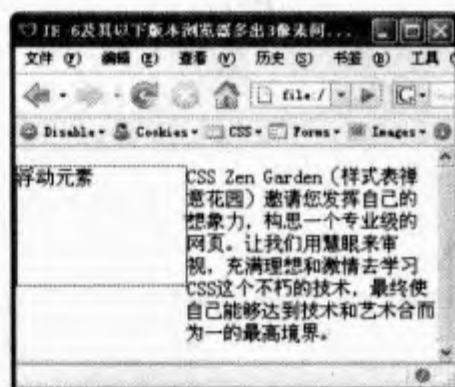


图 7.76 FF 2 中解析效果

解决这个问题方法如下。

此类问题在网页布局中一般不会破坏页面整体效果, 如果你不仔细观察, 可能还不会注意到这个细节。如果要求不是很苛刻, 可以不管它。当然你也可以使用如下兼容方法解决这个问题:

```

* html div { /* 兼容 IE 6 及其以下版本浏览器 */
    margin-right:-3px;          /* 取浮动元素右外边距为负值, 收缩缩进 */
}
* html p { /* 兼容 IE 6 及其以下版本浏览器 */
    height:1px;                 /* 为环绕元素添加布局, 强制左对齐 */
}

```

7.4.9 浮动布局中浮动元素并列错位问题以及解决方法

你可能也遇到过这样的问题, 当多列并列浮动时, 最后一列突然跑到下面去了 (如图 7.77 所示)。

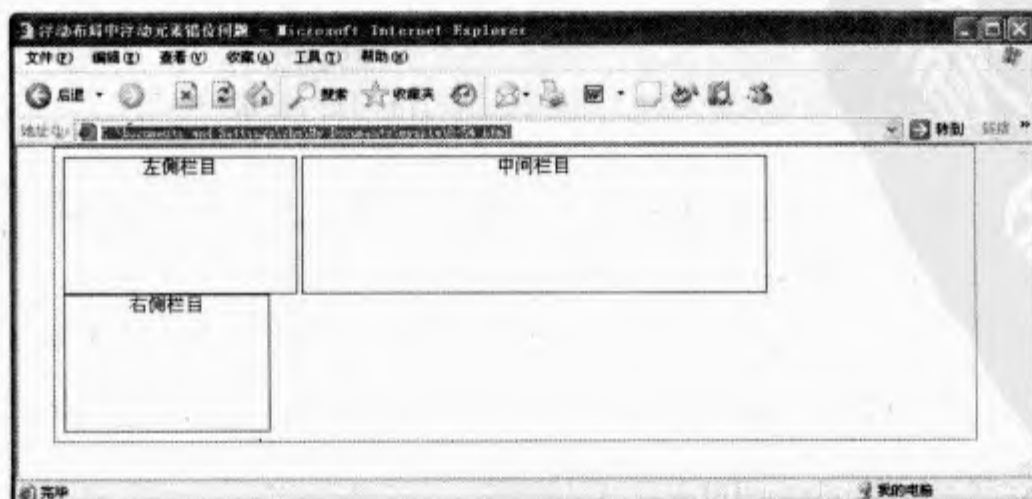


图 7.77 浮动布局错位问题

这是很讨厌的问题，出现这种情况有以下两种原因。

第一，浮动列宽度固定。当浏览器改变窗口大小时，导致窗口宽度太窄，容不下所有列的宽度，则最后一列被迫挤到下面一行显示。这是经常遇到的情况，解决的方法是建议读者设置一个外套，并固定这个外套的总宽度。

第二，当浮动列总宽度之和超出了网页设置的宽度，则最后一列被迫挤到下面一行显示。对于这种情况，解决的方法是精确计算每列的总宽度，然后计算所有列的总宽度，要求这个和必须小于或等于网页或包含框的宽度。例如，请尝试输入如下的网页模型结构：

```
<style type="text/css">
#main {
    width:400px;          /* 包含框的宽度 */
    padding:4px;          /* 内边距 */
    border:solid 2px red;  /* 边框 */
}
#main div {
    float:left;           /* 向左浮动所有列 */
    height:160px;         /* 高度 */
}
#left {
    width:100px;          /* 左栏宽度 */
    background:red;       /* 左栏背景色 */
}
#middle {
    width:200px;          /* 中间栏宽度 */
    background:blue;      /* 中间栏背景色 */
}
#right {
    width:100px;          /* 右栏宽度 */
    background:green;     /* 右栏背景色 */
}
.clear {clear:both; }    /* 定义清除类 */
</style>
<div id="main">
    <div id="left">左侧栏目</div>
    <div id="middle">中间栏目</div>
    <div id="right">右侧栏目</div>
    <br class="clear" />
</div>
```

在上面这个结构模型中，包含框的宽度为 400px，所以其包含的栏目总宽度不能够超过这个宽度。例如，针对上面的栏目样式，则可以很容易地计算各个栏目的总宽度。

这样计算起来可能比较简单，如果各个栏目增加了外边框、内边框之后，计算各个栏目的总宽度时就需要小心了。如果心细，则可能不会出现总宽度超过包含框或页面宽度的问题，自然也不会出现错位现象。不过下面这些问题需要你特别注意。

(1) 百分比宽度问题。当栏目宽度单位设置为百分比时，由于不同浏览器对于小数值的处理方式不同，可能会出现栏目实际总宽度大于设置的宽度的现象，从而导致栏目错位发生。例如，假设包含框宽度为 401px，左栏宽度为 50%，右栏宽度为 50%，则左右栏宽度为 200.5px，有些浏览器（如 IE 6 等）会四舍五入地认为左右栏宽度分别为 201px，这样左右栏的实际总宽度就等于 402px。由于实际总宽度大于原定宽度，就会出现错位现象。

当你准备为栏目设置百分比宽度时，一定要特别小心，为了避免这样问题发生，建议适当

设置小一点的百分比取值。

(2) 双边距问题。针对上面的示例,假设我们定义左栏宽度为 90px,而左外边距为 10px,应该说该栏目的总和依然为 100px,但是在 IE 6 中显示则如图 7.78 所示。这是由于 IE 6 及其以下版本的一个 Bug 造成的,解决方法请参阅第 7.4.8 节中的讲解。但是在其他浏览器中都能够正常解析,图 7.79 所示为在 FF 2.0 中解析的效果。

```
#left {
    width:90px;           /* 减小宽度 */
    margin-left:10px;     /* 增加左外边距 */
}
```

另外,还有 IE 的 3 像素 Bug 也可能导致页面布局错位问题发生,详细说明可以参阅第 7.4.8 节中的讲解。

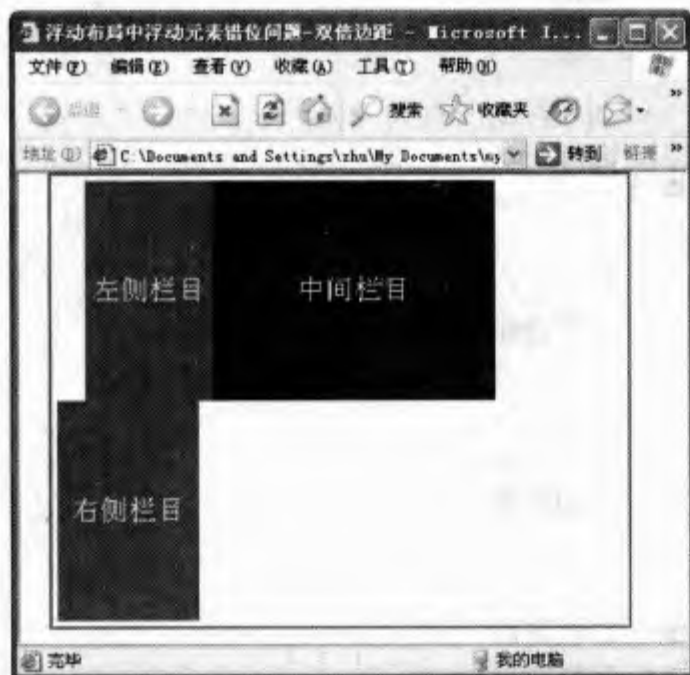


图 7.78 IE 6 中因双倍边距问题错位

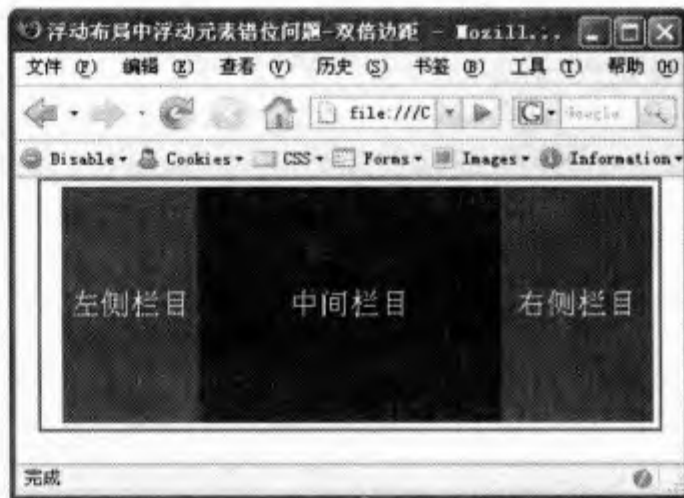


图 7.79 FF 2.0 中解析效果

7.5 CSS 浮动布局实战

CSS 仅仅提供了一个 float 属性,却改变了网页设计的思路,当然设计师的工作则变得愈加充实起来。本节将通过两个典型示例演示 CSS 浮动布局的实战技巧。也许我们无法手把手地帮助你掌握页面中每个设计细节,但是本节绝对保证你能把握浮动布局的设计思路和实施技巧。

7.5.1 深红色咖啡馆

深红色咖啡馆是禅意花园第 179 号作品 (<http://www.csszengarden.com/?cssfile=179/179.css>),设计效果如图 7.80 所示。页面以深红色为主色调,按着主辅两栏式来设计。左栏显示主要信息,右栏显示链接信息。画面以背景图像特写和圆滑的模块为网格分显不同的模块信息。

原作者采用流动布局+绝对定位的方法来设计这个效果,本节示例则采用完全 CSS 浮动布局的方法来设计。

回忆一下,在第 7.2.3 节中我们曾经详细解析了禅意花园的页面结构。整个页面包含在一个包含框中 (<div id="container">),在包含框中从上到下自然排列了 3 个模块:第 1 个模块为活动介绍,第 2 个模块为支持文本,第 3 个模块为超链接列表。页面结构的直观示意图如图 7.81 所示。



图 7.80 深红咖啡馆页面效果图

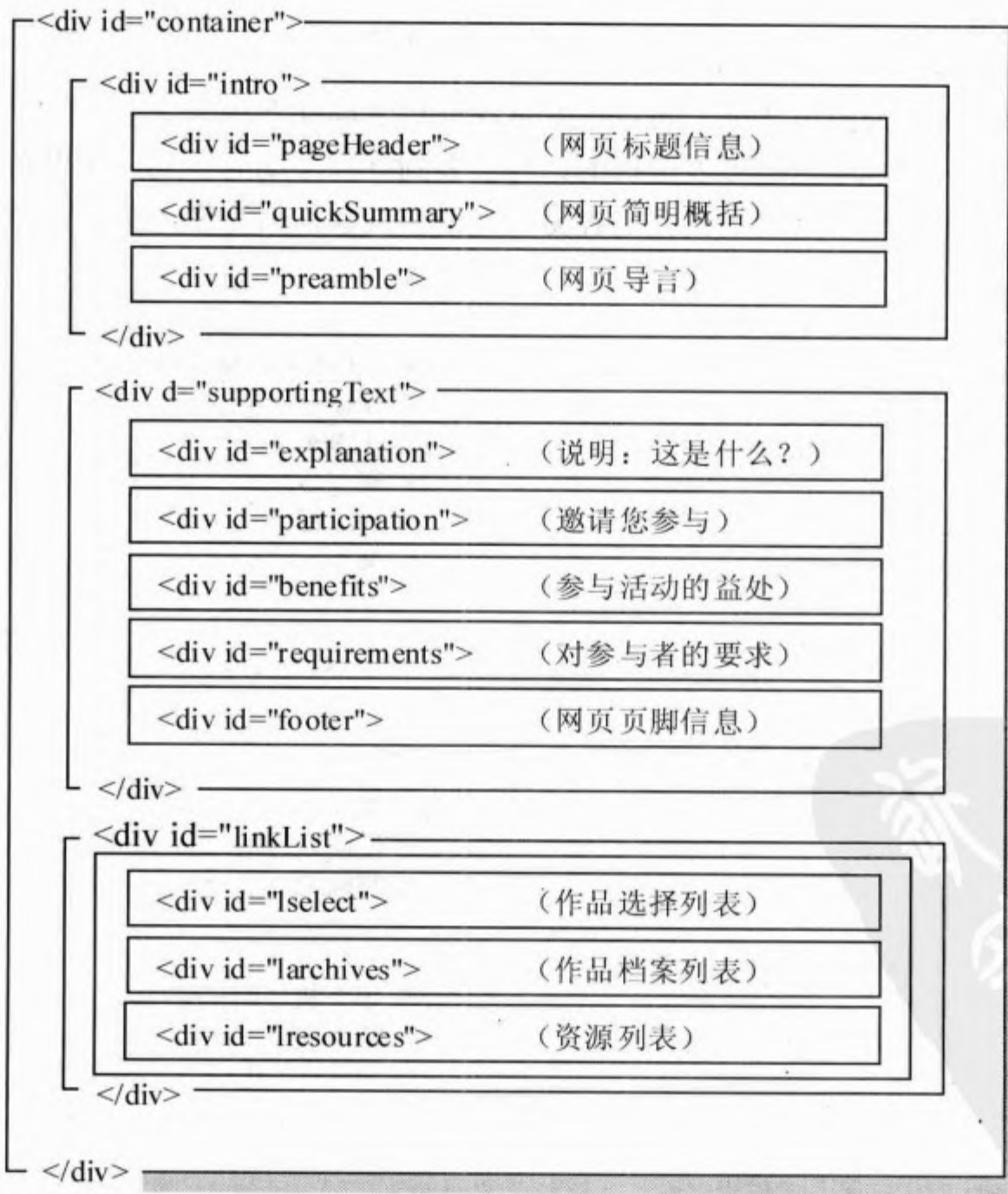


图 7.81 禅意花园的 HTML 结构

根据禅意花园的 HTML 结构, 我们所要设计的结构布局是让第 1 个模块和第 2 个模块向左浮动, 而定义第 3 个模块向右浮动, 设想的示意图如图 7.82 所示。

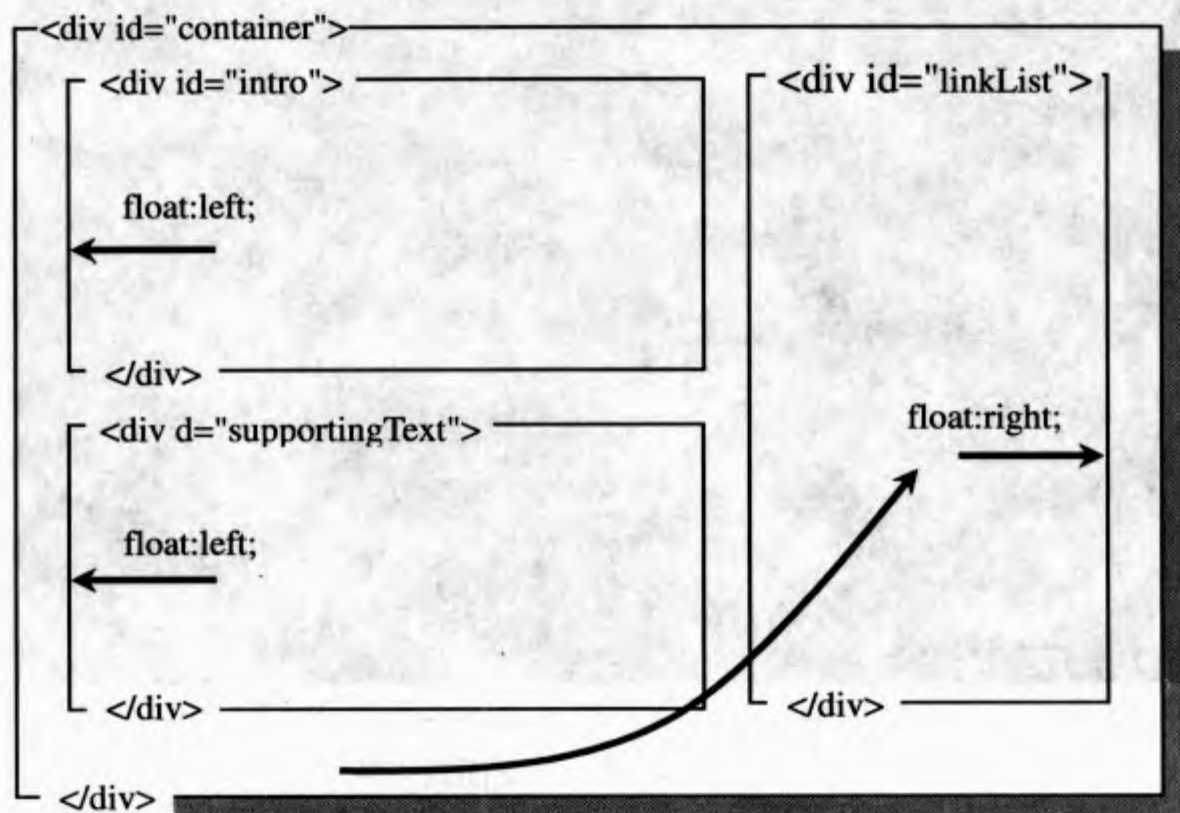


图 7.82 深红咖啡馆 CSS 布局示意图

首先, 定义包含框为固定宽度。固定宽度包含框对于浮动布局来说是非常重要的, 甚至说是必需的。想一想这个道理也是可以理解的, 对于多列并列浮动的布局, 如果允许包含框的宽度为百分比, 那么内部浮动的模块就会很容易出现错位现象, 因为你无法保证所有浏览器窗口的大小都是固定的。

```

div#container { /* 包含框 */
  width: 760px;           /* 固定包含框宽度 */
  margin-left: auto;      /* 实现水平居中 */
  margin-right: auto;     /* 实现水平居中 */
  margin-top: 0;         /* 顶部外边距 */
  padding: 0;            /* 内边距 */
  text-align: left;       /* 文本左对齐 */
}

```

然后让第 1 个模块和第 2 个模块向左浮动, 第 3 个模块向右浮动, 同时设置好 3 个模块的宽度。要注意模块的总宽度不能够超过包含框的宽度。

```

div#intro { /* 第 1 个模块 */
  width: 580px;           /* 第 1 模块的宽度 */
  margin: 0;              /* 清除外边距 */
  padding: 0;             /* 清除内边距 */
}
div#supportingText { /* 第 2 个模块 */
  width: 580px;           /* 第 2 模块的宽度 */
  margin: 0;              /* 清除外边距 */
  padding: 0;             /* 清除内边距 */
  float: left;            /* 向左浮动 */
}
div#linkList { /* 第 3 个模块 */
  width: 155px;           /* 第 3 个模块的宽度 */
  padding: 0;             /* 清除内边距 */
}

```



```

float: right; /* 向右浮动 */
}

```

这时你会发现页面布局呈现图 7.83 所示的效果。也就是说第 3 个模块向右浮动之后，它只能够保持与第 2 个模块平行显示，而不是向上浮动到与第 1 个模块保持平行。

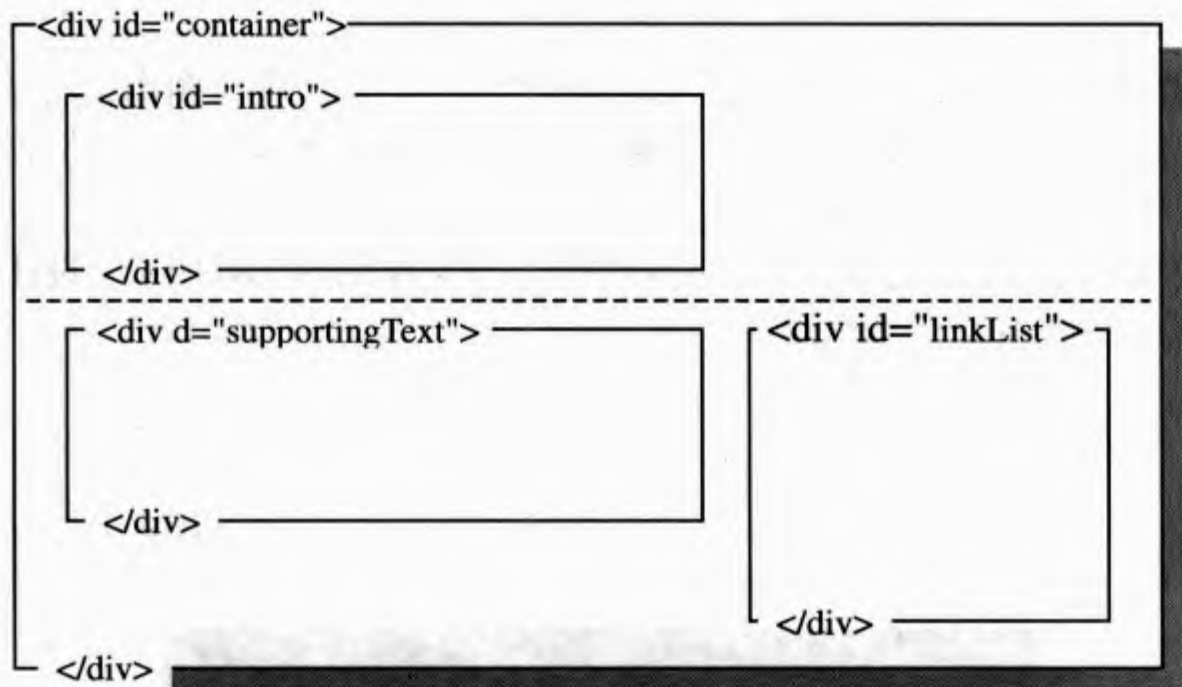


图 7.83 深红咖啡馆 CSS 浮动布局存在的问题示意图

解决这个问题可以有如下两种方法。

第一种方法，设置第 3 个模块的顶部外边距为负值。把第 3 个模块强拉到顶部，保持与第 1 个模块平行对齐。

```

div#linkList { /* 第 3 个模块 */
    margin-top: -320px; /* 取负值向上移动模块 */
    margin-right: 5px; /* 增加右外边距，调整模块显示位置 */
}

```

第二种方法，为第 2 个模块增加清除左侧浮动属性。这样就阻断了第 1 个模块与第 2 个模块可能并列显示的关系，这时第 3 个模块就可以长驱直入，向上浮动并保持与第 1 个模块对齐显示。这时你可以利用外边距来调节第 3 个模块的显示位置。当调节左右外边距时，要注意模块的总宽度不能够超过了包含框所设置的固定宽度。

```

div#supportingText { /* 第 2 个模块 */
    clear: left; /* 清除左侧浮动 */
}
div#linkList { /* 第 3 个模块 */
    margin-right: 5px; /* 增加右侧外边距 */
    margin-top: 290px; /* 增加顶部外边距 */
}

```

页面主体框架布局完成之后，下面我们再来研究如何设计二级模块和局部板块效果。

第 1 步，设计页面整体色调和默认样式。这个可以在 body 元素中实现。实际上设计师都有这样的设计习惯，通过在 body 元素中定义页面字体样式、段落样式、网页背景色、页边距和页面对齐问题。

```

body { /* 定义页面基本属性 */
    background: #371212 url(background.gif) top repeat-x; /* 水平平铺背景图像 */
    font-family: Tahoma, Arial, Helvetica, sans-serif; /* 字体 */
}

```

```

color: #F7F5D9; /* 字体颜色 */
font-size: 0.75em; /* 字体大小 */
line-height: 1.6em; /* 行高 */
padding: 0; /* 清除页边距 */
margin: 0; /* 清除页边距 */
text-align: center; /* 页面居中 */
}

```

设置网页背景时，建议将背景色和背景图像配合使用，如果背景图像无法显示，则可以使用风格类似的颜色来代替，避免页面以白色背景显示时所遇到的尴尬。使用背景图像可以设计渐变背景效果，一般采用水平平铺来实现。

第2步，在页面包含框中再定义一个背景图像，该图像如图7.84所示。设计师把网页标题和头部主要信息都封装在背景图像中。

```

div#container { /* 包含框 */
    background: #000000 url(background_header.gif) top center no-repeat; /* 背景图
像 */
    text-align: left; /* 页面居中 */
}

```

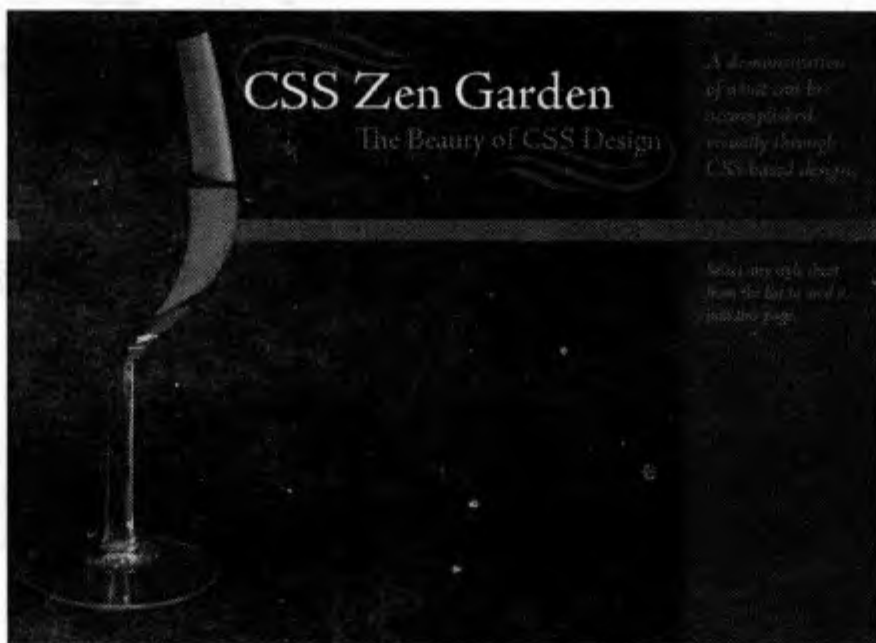


图 7.84 顶部背景图像

这种把网页信息部分封装在背景图像的设计思路有两大优势：一方面，简化了 CSS 设计的难度；另一方面，增加了页面的艺术设计效果，毕竟使用图像更能够设计出 CSS 所无法实现的效果。

当然这种做法也在一定程度上给页面传输增加了负担，因为图像一般都比较比较大，特别是大幅图像所占用的带宽更是明显。

如果把页面部分信息封装在背景图像中，你就必须使用 CSS 隐藏 HTML 部分结构和信息：

```

div#pageHeader { /* 第1模块的第1个子模块——网页标题信息 */
    display: none; /* 隐藏结构和信息 */
}
div#quickSummary p.p1 { /* 第1模块的第2个子模块第1段——网页概括信息 */
    display: none; /* 隐藏结构和信息 */
}

```

对于第1模块的第2个子模块第2段链接信息，则可通过内边距和外边距来调节它在页面中的位置（如图7.85所示）。

```

div#quickSummary { /* 第1模块的第2个子模块 */
    width: 260px; /* 固定宽度 */
}

```



```

height: 20px;
padding: 220px 0 0 0;
margin: 0 0 30px 310px;
}
/* 固定高度 */
/* 增加顶部内边距 */
/* 增加左侧和底部外边距 */

```

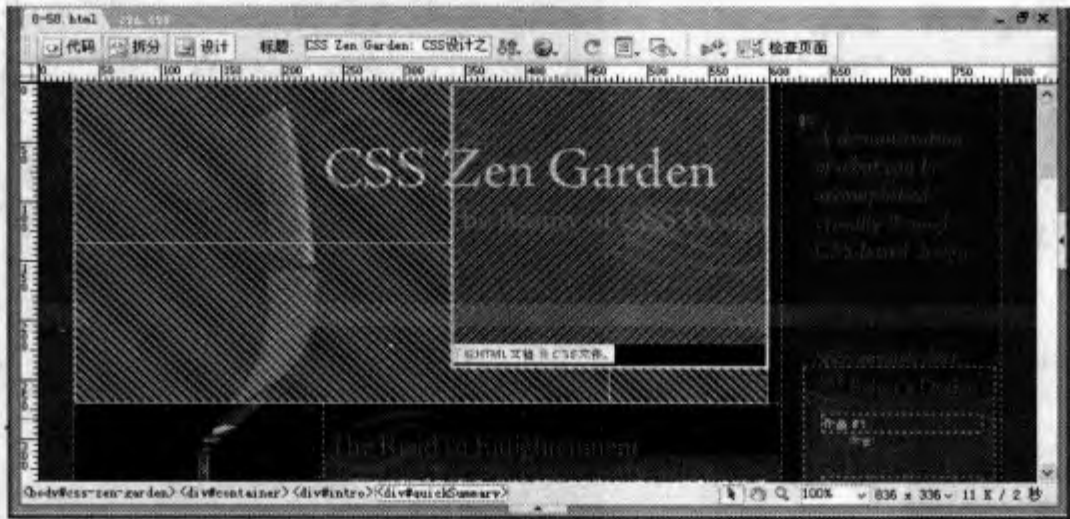


图 7.85 使用外边距调整未隐藏信息的显示位置

最后我们再看看其他模块的设计效果（如图 7.86 所示）。这种设计效果主要通过 3 幅背景图像来完成，与我们在前面讲解的圆角设计方法是相同的，这里就不再详细分析。

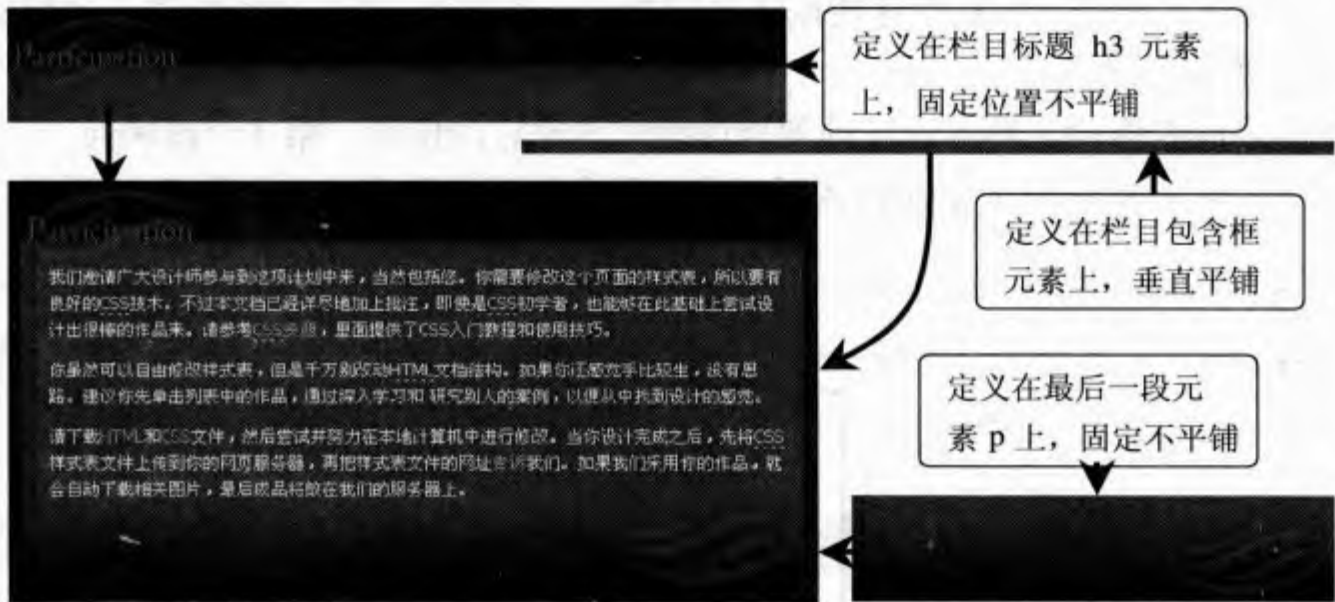


图 7.86 网页模块设计效果

7.5.2 阳光灿烂喜洋洋

“阳光灿烂喜洋洋”是禅意花园第 181 号作品（<http://www.csszengarden.com/?cssfile=181/181.css>），设计效果如图 7.87 所示。页面以灿烂的大红色为主色调，以固定宽度的三栏进行设计。左右两侧栏目以大红色为背景色衬托喜庆的氛围，中间窄条栏目为白色为背景，通过明暗、色度的反差以吸引人的注意力。

左栏显示标题和页面概括信息，右栏显示主体信息，中栏显示链接信息，画面左上角以灿烂的光线暗示页面的情绪。所有二级模块都以深色背景标题栏进行简单分隔，避免过分破坏页面的三栏条线型主体结构效果。

原作者采用绝对定位的布局方法来完成本作品的设计，本节示例也一样完全采用纯浮动布局的方法来设计这个效果。

在设计这个作品之前，请读者先温习一下禅意花园的 HTML 结构，然后我们一块来分析本作品的主要框架实现思路 and 过程。



图 7.87 “阳光灿烂喜洋洋” 页面效果图

在设计的过程中，我们设想让第 1 个模块的第 3 个子模块浮动到右侧，而其他 2 个子模块则顺势自然流动在左侧显示，并在这些模块之间拉出一个空当，准备预留给第 3 个模块来显示，如图 7.87 中白色的区域。

第 2 个模块向右浮动，然后通过顶部外边距来调整它的位置。第 3 个模块向左浮动，然后再通过负外边距来强迫其位于中间白色区域内。整个设计思路和结构如图 7.88 所示。

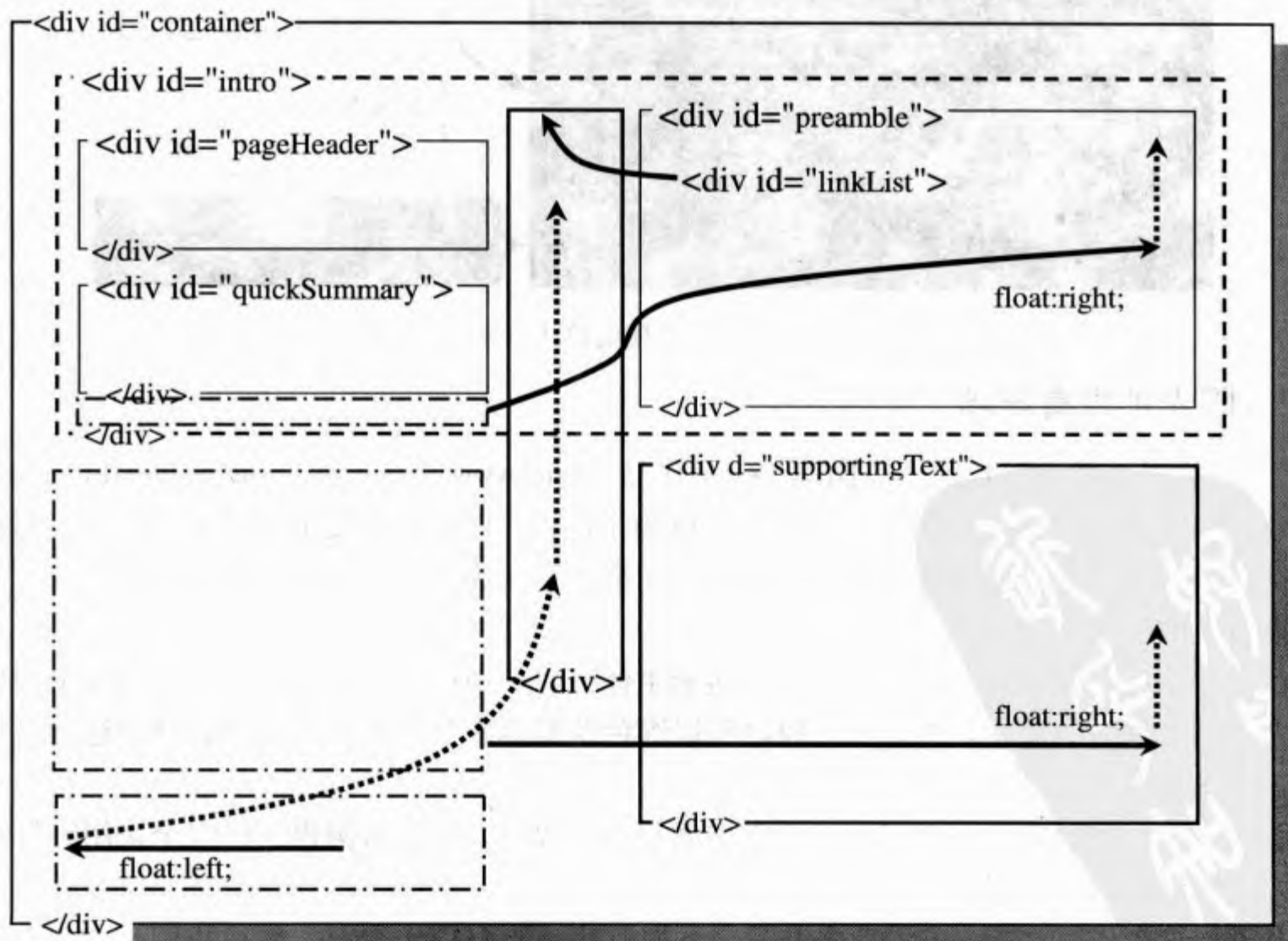


图 7.88 “阳光灿烂喜洋洋” CSS 布局示意图

其中虚线箭头表示通过外边距取负值来实现强制位置移动效果,实线箭头表示模块从原来应该排列的位置浮动到页面新的位置,虚线框表示第一个模块的包含框,虚点线框表示浮动前模块的位置。

设计的第一步是先固定页面包含框的宽度。这一步很重要,否则后面的操作都将不易进行。宽度的大小可以根据情况而定,这里以 1024px×768px 屏幕分辨率为基础,扣除 20px 宽度的滚动条。

```
div#container { /* 页面包含框 */
    width:1002px; /* 固定宽度 */
}
```

第 1 个模块 (<div id="intro">) 可以不进行设置,直接设置其包含的 3 个子模块:

```
div#pageHeader { /* 第 1 个子模块 */
    width: 200px; /* 固定宽度 */
    height: 320px; /* 固定高度 */
}
div#quickSummary { /* 第 2 个子模块 */
    width: 185px; /* 固定宽度 */
}
div#preamble { /* 第 3 个子模块 */
    float: right; /* 向右浮动 */
    width: 510px; /* 固定宽度 */
    clear: right; /* 清除右侧浮动元素 */
}
```

第 3 个子模块向右浮动,但是第 1、第 2 个子模块默认显示为块状元素,占据一行的空间,所以在默认的状态下浮动元素显示在第 2 个子模块的右下侧方向上。为了使向上与第 1 个子模块水平显示,可以通过负外边框来实现。所设计的简单效果如图 7.89 所示。

```
div#preamble {
    margin-top:-440px; /* 负外边框,强制模块向上移动 */
}
```

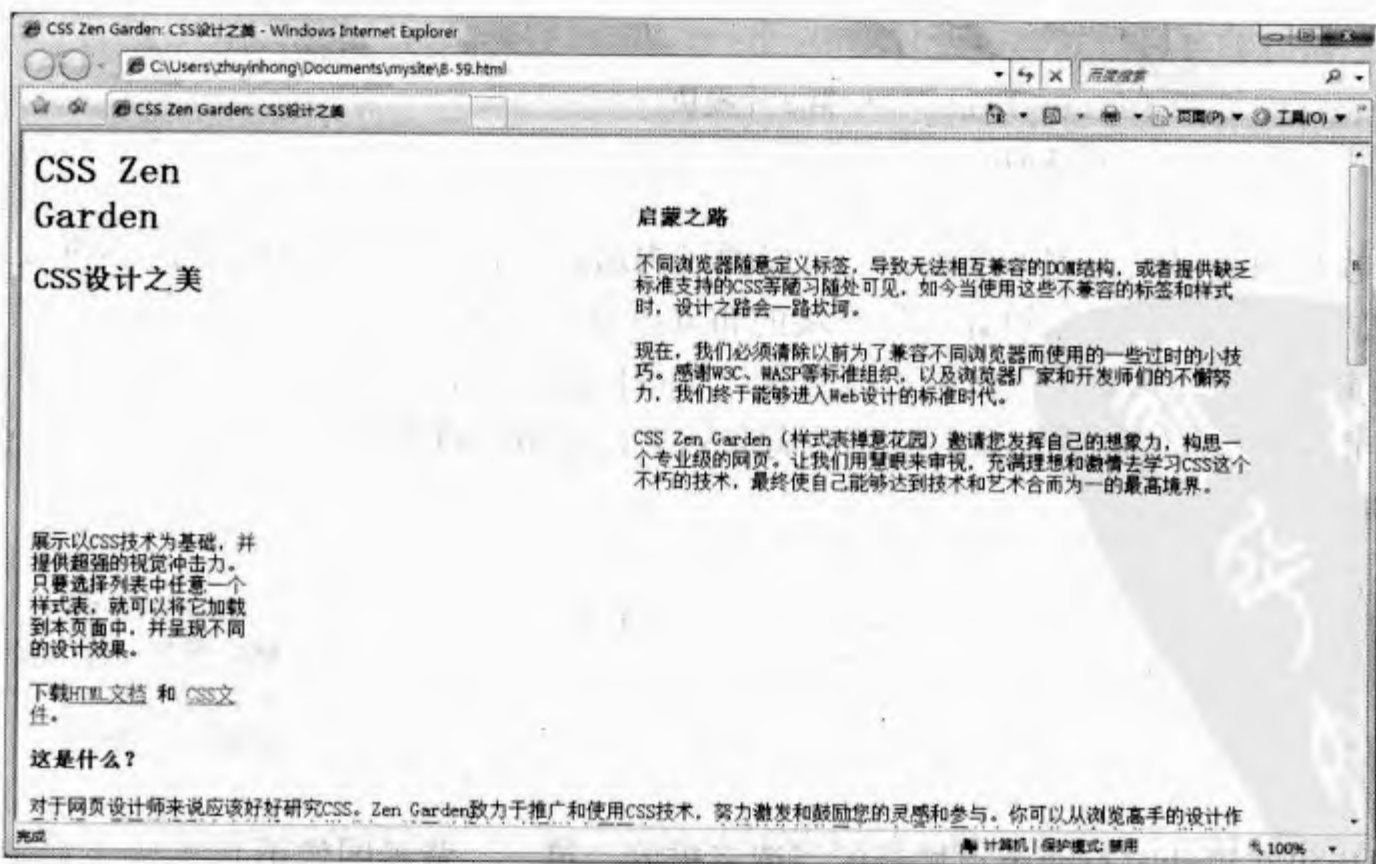


图 7.89 第 1 个模块的结构效果

设置第 2 个模块（<div id="supportingText">）向右浮动，并固定宽度：

```
div#supportingText { /* 第 2 个模块 */
    float: right;           /* 向右浮动 */
    width: 510px;          /* 固定宽度 */
    clear: right;          /* 清除右侧浮动 */
}
```

把第 2 个模块浮动到右侧之后，再通过负外边距向上移动该模块：

```
div#supportingText {
    margin-top: -120px;     /* 向上移动模块 */
}
```

设计第 3 个模块（<div id="linkList">）向左浮动，并定义固定宽度：

```
div#linkList { /* 第 3 个模块 */
    width: 263px;          /* 固定宽度 */
    float: left;           /* 向左浮动 */
}
```

然后通过外边距来定位第 3 个模块的位置：

```
div#linkList {
    margin-left: 200px;     /* 增加左侧外边距，向右移动 */
    margin-top: -1670px;   /* 取负外边距，向上移动 */
    display: inline;       /* 解决 IE 下浮动显示时的双倍边距问题，详情参阅第 7.4.8 节 */
}
```

由于负外边距取值在 IE 和非 IE 浏览器下解析存在一定的误差，我们还需要单独为非 IE 浏览器设置一个负外边距值。这些样式应该放在对应样式的后面，它们只能够被非 IE 浏览器识别并解析。其中“html>/**/body”前缀只能够被符合标准的浏览器所解析。

```
html>/**/body div#preamble { /* 第 1 个模块的第 3 个子模块 */
    margin-top: -470px;
}
html>/**/body div#supportingText { /* 第 2 个模块 */
    margin-top: -170px;
}
html>/**/body div#linkList { /* 第 3 个模块 */
    margin-top: -1570px;
}
```

如果模块内容显示位置不准确，还可以通过内边距进行调整，详细代码就不再显示。主体布局完成之后，下一步是来设计二级模块的布局以及页面显示样式。

定义页面基本属性。本案例的页面背景设计得比较巧妙，它模仿伪列布局（详细内容请参阅第 4.5 节内容）的设计思路来设计网页背景效果，通俗说就是使用背景图像来设计分栏效果（如图 7.90 所示）。

```
html, body { /* 网页属性 */
    background: url(bg.gif) left top repeat-y #F06; /* 网页背景图像 */
    background-attachment: fixed; /* 固定背景图像位置 */
    margin: 0; /* 清除页边距 */
    padding: 0; /* 清除页边距 */
}
```

上面代码在设计时有两个小技巧值得读者借鉴：第一，背景图像不必要设计为全屏大小，只需要把渐变、阴影效果处用图像设计，其他部分可以使用背景颜色来代替；第二，通过

`background-attachment` 属性把背景图像固定在页面中，这样不需要平铺整个页面，因为只固定为显示窗口大小，当滚动条滚动时背景图像不动，从而避免滚动条滚动时系统不断平铺背景图像的现象。



图 7.90 网页背景图像效果

隐藏不需要的页面结构和信息。如果感觉使用背景图像来设计页面结构和信息会更方便、设计效果更好，则不妨采用这种方法：

```
div#pageHeader h1, div#pageHeader h2, div#linkList h3 {
    display: none;                                /* 隐藏页面结构 */
}
```

设计页面标题。页面标题以背景的形式来实现，这样会更容易设计个性标题效果：

```
div#pageHeader {
    width: 200px;                                /* 固定宽度 */
    height: 320px;                                /* 固定高度 */
    background: url(logo.gif) left top no-repeat; /* 背景图像 */
}
```

第2模块的子栏目标题显示为麻点区域效果（如图7.91所示），它是通过背景图像平铺来实现的：

```
div#preamble h3, div#supportingText h3 {
    display: block;                                /* 块状显示 */
    background: url(hbg.gif) left top repeat #000; /* 背景图像平铺显示 */
    margin: 0;                                     /* 清除默认边距 */
    padding: 0;                                    /* 清除默认边距 */
    padding-left: 20px;                             /* 增加左侧边距 */
}
```

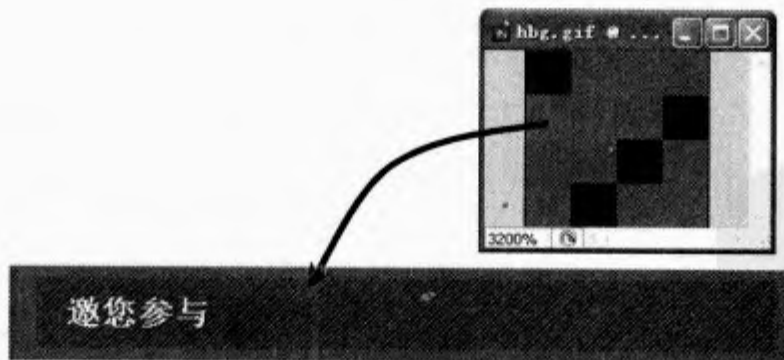


图 7.91 第2模块子栏目背景图像

第3模块的子栏目标题显示效果如图7.92所示，它是先把标题文本隐藏，直接使用一个背景图像来代替整个标题信息。

```
div#linkList div#lselect h3 span, div#linkList div#larchives h3 span, div#linkList
div#lresources h3 span {
    display: none;
}
```

```
div#linkList div#lselect h3 {  
    background: url(ll_selectadesign.gif) left top no-repeat;  
}  
div#linkList div#larchives h3 {  
    background: url(ll_archives.gif) left top no-repeat;  
}  
div#linkList div#lresources h3 {  
    background: url(ll_resources.gif) left top no-repeat;  
}
```

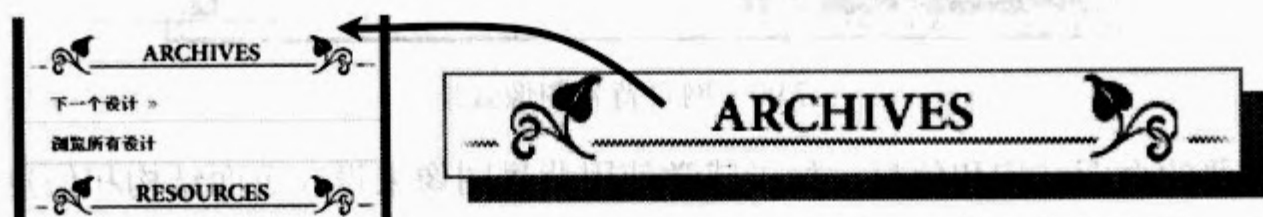


图 7.92 第 3 模块子栏目背景图像

精确的网页布局

在上一章中我们曾经系统地学习了浮动布局的基本原理和操作技巧，浮动布局的优势就在于它的灵活性。但是过于强调灵活性，你会发现网页无法精确控制，这时还需要一定的精确布局来进行互补，否则你会感觉到网页布局的很多效果是无法实现的。

提到精确网页布局，你可能会想到在 Photoshop 中编辑、合成图像，或者是在 PageMaker、InDesign 中排版印刷物。是的，CSS 所提出的精确网页布局的概念正是希望网页设计如同桌面排版印刷物，这样就能够精确定位网页中每个元素的位置，甚至精确到像素的级别。本章将详细讲解如何精确定位网页元素，并结合浮动布局，真正让你随心所欲地驾驭网页布局。

8.1 CSS 精确布局的定位原理

正如浮动布局中的 float 属性一样，CSS 所提供的 position 属性也是精确布局的核心和关键，它与 float 属性的功能和作用并驾齐驱协同完成网页的精确性和灵活性布局设计，因此它们也是 CSS 布局中两个最基本、最重要的技术概念。

8.1.1 认识 position

position 中文意为“位置”，顾名思义该属性的功能就是用来确定元素的位置。有了这个属性，你可以把图片放置到栏目的右上角，或者把置顶工具条始终固定在网页的顶部等。

CSS 定位（CSS Position）的核心正是基于这个属性来实现的，我们可以简称之为 CSS-P。使用 CSS-P 的时候，一般主要把它用在 div 元素上，当你把文本、图像或其他元素放在 div 元素中时，它可以被称为包含块（Div Block）、包含元素（Div Element）或 CSS 层（CSS-Layer），因此我们有时候可以简称它为层（Layer），这与图像编辑器中的图层功能类似。

如果在 Dreamweaver CS3 中选择【插入记录】|【布局对象】|【AP Div】菜单命令，你可以在当前网页的当前位置插入一个默认大小的 CSS 层（如图 8.1 所示）。这里的 AP Div 是“Absolute Position Div”的简写，可以翻译为“绝对定位的 div 元素”。如果切换到【代码】视图，你可以看到该层的 CSS 源代码：

```
<style type="text/css">
<!--
#apDiv1 { /* 插入层 */
    position:absolute;           /* 绝对定位 */
    width:200px;                 /* 宽度 */
    height:115px;                /* 高度 */
}
```

```

        z-index:1;
    }
-->
</style>
<div id="apDiv1"></div>

```

```
/* 层叠顺序 */
```

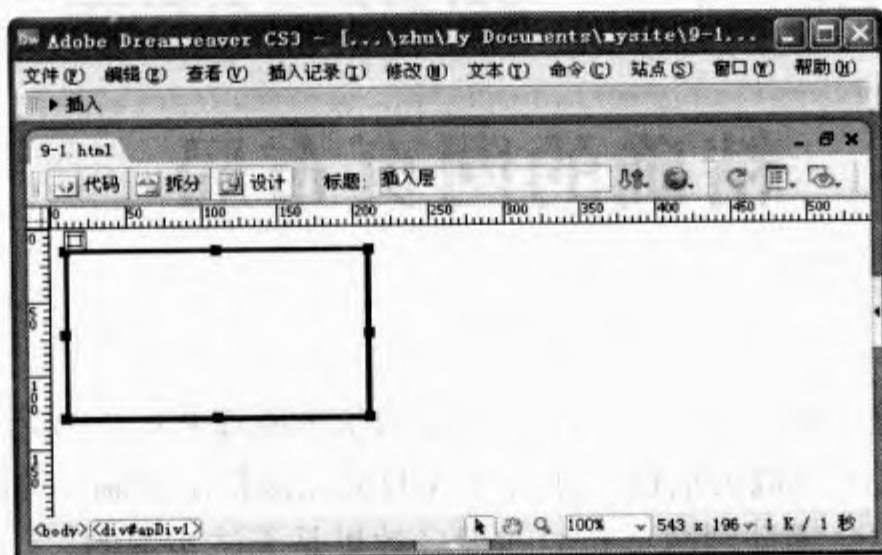


图 8.1 在 Dreamweaver CS3 中插入层

`position` 属性包括 4 个取值：`static`（静态）、`absolute`（绝对）、`relative`（相对）和 `fixed`（固定）。下面将分节详细说明。

8.1.2 静态定位

当 `position` 属性取值为 `static` 时，可以定位元素于静态位置。所谓静态位置就是各个元素在 HTML 文档流中应有的位置。根据 HTML 超文本传输协议，浏览器在接收和解析网页信息时，是遵循从上到下的顺序来实现的，每个元素以及对象在网页中的位置决定了它们被解析和显示顺序。自然，每个元素如同流水一样连续地、有序地向下解析和显示。

可以这样说，任何元素在网页中的位置是固定的。静态定位能够根据元素或对象的这种自然顺序来定位元素的位置。例如，在下面这个代码块中，如果没有特殊的 CSS 声明，它们都以静态定位确定自己的位置并进行显示。`<h1>`和`<h2>`按着先后顺序排列在一起，它们的位置始终位于`<div id="pageHeader">`对象包围的区域之中（如图 8.2 所示）。

```

<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>

```

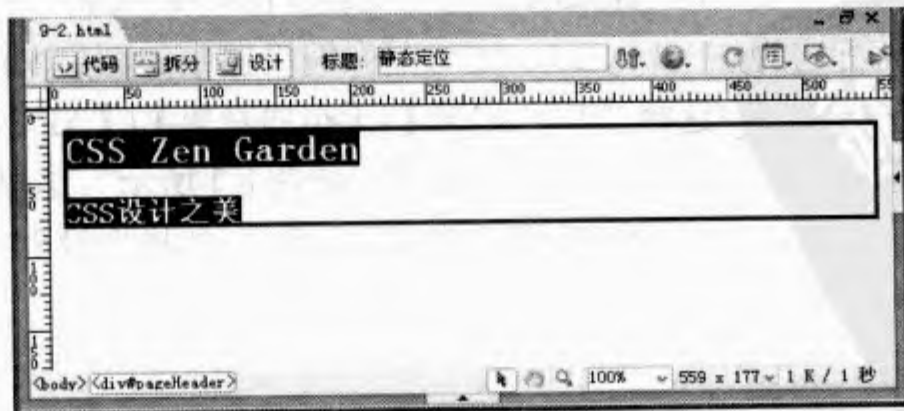


图 8.2 静态定位的元素

任何元素在默认状态下都会以静态定位来确定自己的位置，所以当没有定义 `position` 属性时，并不说明该元素没有拥有自己的位置，它会遵循默认值显示为静态位置。在静态定位下，

你是无法通过坐标值（top、bottom、left 或 right）来改变它的位置。

8.1.3 绝对定位

在讲到 CSS 单位时，我们曾经提到了绝对单位。与绝对单位类似，绝对定位完全忽略了 HTML 文档流对于元素位置的影响。当 position 属性取值为 absolute 时，它会强制把被应用的元素从文档流中拖出来，根据某个参照物坐标来固定它的显示位置。这好比把河流中漂流的船只拴在岸边一样，这里的河流就是文档流，船只就是绝对定位的元素，而岸边的固定物就是绝对定位的参照物。

例如，针对如下的结构，我们使用绝对定位的方法把元素 h2 从文档流中拖离出来，固定在窗口的(100px,100px)坐标位置（如图 8.3 所示）。

```
<style type="text/css">
#pageHeader h2 {
    position:absolute;           /* 绝对定位 */
    left:100px;                  /* x 轴坐标 */
    top:100px;                   /* y 轴坐标 */
}
</style>
<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>
```



图 8.3 绝对定位的元素

可以看到，h2 元素完全不再受文档流的影响，始终显示在指定坐标的位置。此时 h2 元素就与<div id="pageHeader">对象没有任何关系了。h2 元素的大小和位置不会影响到<div id="pageHeader">对象的大小和位置，反过来<div id="pageHeader">对象的大小和位置也不会影响 h2 元素的大小和位置。

绝对定位是网页精确定位的基本方法，如果再结合 left、right、top 和 bottom 坐标属性进行精确定位，结合 z-index 属性排列元素的覆盖顺序，结合 clip 和 visibility 属性裁切、显示或隐藏元素对象或部分区域，你可以设计出更强大的网页布局效果。围绕这个话题，我们还将后面章节进行详细讲解。

8.1.4 相对定位

使用静态定位过于放任自流，使用绝对定位过于绝对无情。而相对定位是一种折中的定位方法，是在静态定位和绝对定位之间取一个平衡点。所谓相对定位，就是被应用的元素不脱离文档流，但却能够通过坐标值以原始位置为参照物进行偏移。

例如,在下面这个示例中,h2元素被定义了相对定位,坐标偏移值为(100px,100px),这时显示效果如图8.4所示。虽然从显示位置看,h2元素在包含元素<div id="pageHeader">的外边,但是它们之间的父子关系依然存在,并相互影响。

```
<style type="text/css">
#pageHeader h2 {
    position:relative;           /* 相对定位 */
    left:100px;                 /* x 轴坐标 */
    top:100px;                  /* y 轴坐标 */
}
</style>
<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>
```

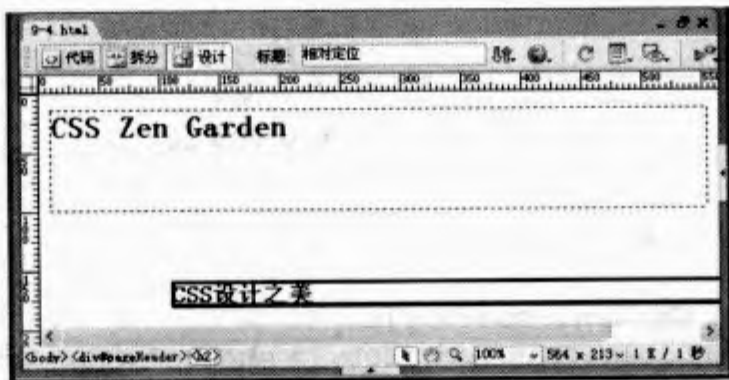


图 8.4 相对定位的元素

如果此时我们定义如下样式,增加包含元素<div id="pageHeader">的顶部和左侧外边距,即向右下角移动包含元素,则相对定位元素 h2 也跟随变化(如图8.5所示)。

```
#pageHeader {
    margin-top:50px;             /* 顶部外边距 */
    margin-left:50px;           /* 左侧外边距 */
}
```

反过来,如果我们增加相对定位元素 h2 的高度,则包含元素的高度也会随之增加(如图8.6所示)。

```
#pageHeader{
    border:solid 1px red;       /* 边框 */
}
#pageHeader h2 {
    height:100px;              /* 增加相对定位元素的高度 */
}
```

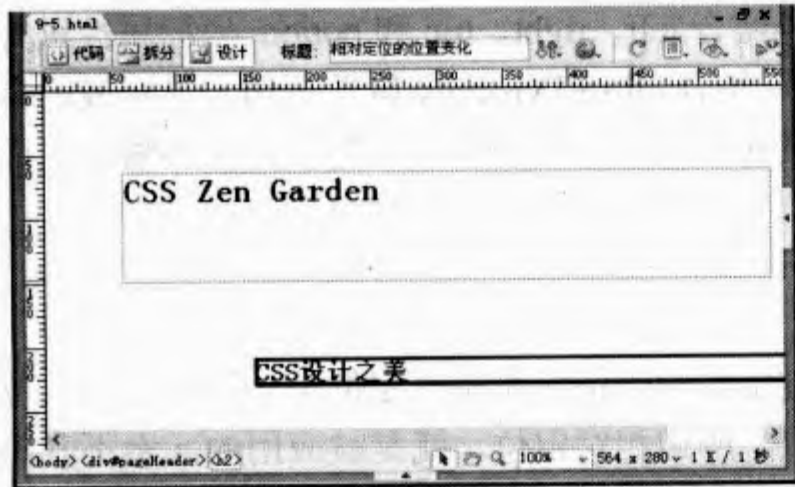


图 8.5 相对定位元素与包含框的位置关系



图 8.6 相对定位元素的大小对包含元素的影响

这时我们可以看到，相对定位元素虽然偏移了原始位置，但是它的原始位置所占据的空间还保留着，并没有被其他元素所挤占。认识并理解相对定位的这一特点对于网页布局来说非常重要，因为很多时候设计师需要相对定义来校正元素的显示位置，但并不希望因为校正位置而影响到其他元素的位置变化。

例如，在下面这个示例中，我们偏移 acronym 元素包含的“CSS”字符串到“设计之美”字符串右侧显示，同时原来的位置并没有被后面的字符串“设计之美”所占用（如图 8.7 所示）。

```
<style type="text/css">
#pageHeader {
    border:solid 1px red;
}
#pageHeader acronym {/* 相对定位元素样式 */
    position:relative;          /* 相对定位 */
    left:240px;                 /* x 轴坐标 */
    top:20px;                   /* y 轴坐标 */
    font-size:80px;             /* 字体大小 */
    color:red;                  /* 字体颜色 */
}
</style>
<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>
```

如果当相对定位元素遇到文档流对象时，它会覆盖文档流中的对象。例如，如果针对上面示例修改其中 h2 元素的偏移位置，使其向上移动与 h1 元素重合，则相对定位元素 h2 会覆盖自然流动元素 h1 对象（如图 8.8 所示）。

```
#pageHeader acronym {
    position:relative;
    top:-70px;
}
```



图 8.7 相对定位元素原始位置与偏移位置关系



图 8.8 相对定位元素覆盖文档流

另外，相对定位元素之间也存在覆盖现象。例如，在下面这个示例中分别定义 3 个不同背景色的盒子，通过相对定位使它们重合在一起。这时我们可以看到，位于后面的相对定位元素会覆盖前面的相对定位元素，如图 8.9 所示。

```
<style type="text/css">
#box1, #box2, #box3 {/* 公共属性 */
    height:100px;
    width:200px;
}
```

```

    position:relative;          /* 相对定位 */
    color:#fff;                 /* 字体颜色 */
}
#box1 { /* 红盒子 */
    background:red;            /* 背景色 */
    top:150px;                 /* x 轴坐标 */
    left:50px;                 /* y 轴坐标 */
}
#box2 { /* 蓝盒子 */
    background:blue;           /* 背景色 */
    top:0;                    /* x 轴坐标 */
    left:100px;               /* y 轴坐标 */
}
#box3 { /* 绿盒子 */
    background:green;          /* 背景色 */
    top:-150px;               /* x 轴坐标 */
    left:150px;              /* y 轴坐标 */
}
</style>
<div id="box1">红盒子</div>
<div id="box2">蓝盒子</div>
<div id="box3">绿盒子</div>

```

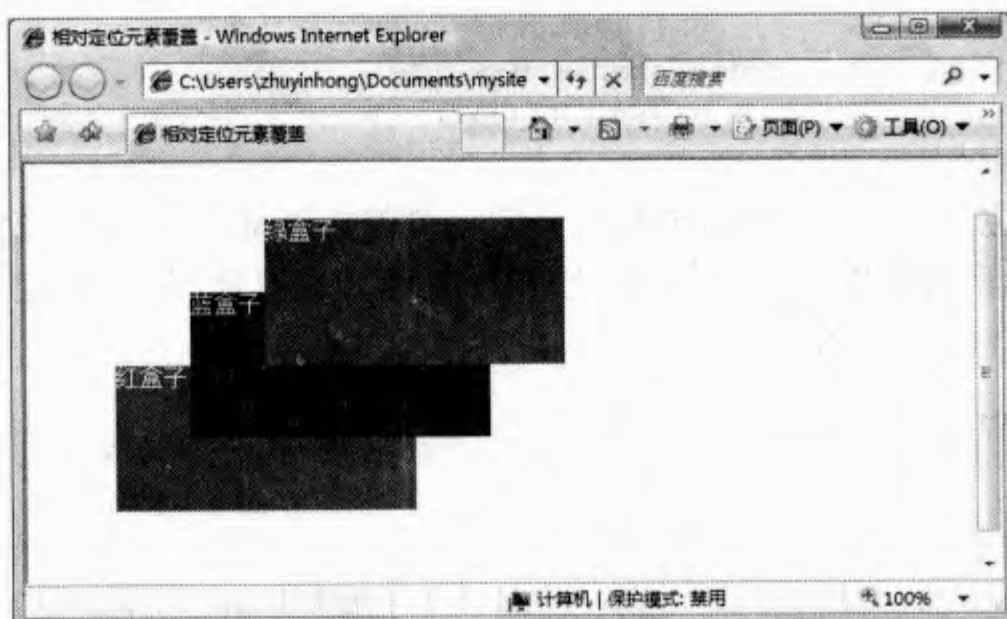


图 8.9 相对定位元素之间覆盖关系

8.1.5 固定定位

固定定位实质是绝对定位的一种特殊形式，它是以浏览器窗口作为参照物来定义网页元素。也就是说如果定义某个元素固定显示，则该元素就不受文档流的影响，也不受包含块的位置影响，它始终以浏览器窗口来定位自己的显示位置。不管浏览器滚动条如何滚动，也不管浏览器窗口大小变化，它都会显示在浏览器窗口内。通俗说，就是以浏览器窗口的 4 个边作为坐标系来定位元素的位置。

例如，在下面这个示例中定义<div id="pageHeader">对象固定在浏览器窗口顶部显示，宽度 100%。然后在下面定义一个超高元素，强迫浏览器显示滚动条。这时如果在浏览器中滚动滚动条，则会发现<div id="pageHeader">元素包含的对象内容始终显示在窗口顶部（如图 8.10 所示）。

```

<style type="text/css">
#pageHeader {

```



```

background:#FF99FF;           /* 背景色 */
position:fixed;                /* 固定定位 */
width:100%;                   /* 宽度 */
top:0px;                      /* x 轴坐标 */
left:0;                       /* y 轴坐标 */
}
#bigBox {
    height:2000px;             /* 定义超高元素 */
}
</style>
<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>
<div id="bigBox"></div>

```

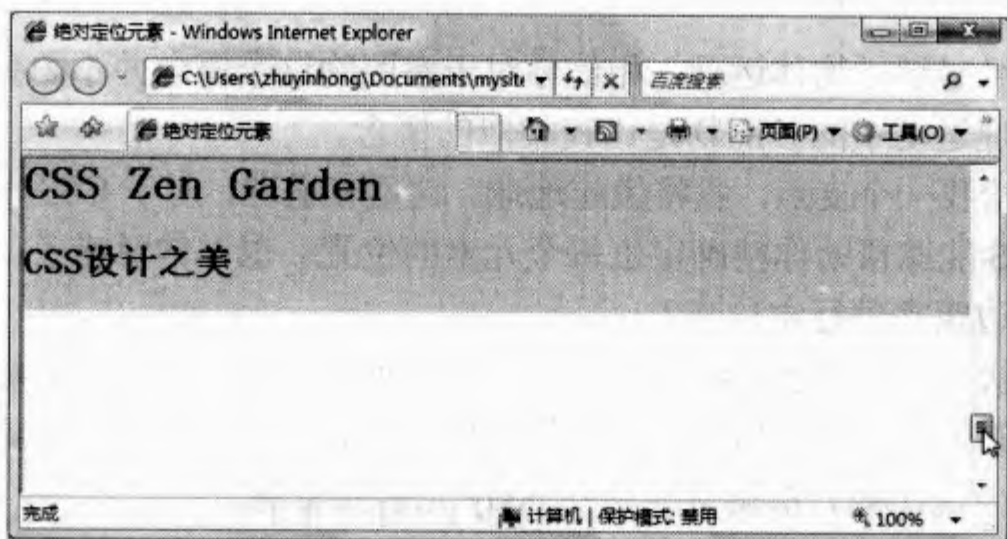


图 8.10 绝对定位元素

提示

固定定位在 IE 6 及其以下版本浏览器中不被支持，所以在使用时受到一定的限制，因为当前大部分用户依然使用 IE 6 版本的浏览器。

`fixed` 属性可以通过 `left`（左边距离）、`top`（顶边距离）、`right`（右边距离）和 `bottom`（底边距离）属性从浏览器不同边框来进行定位，不受 `body` 元素的影响，也就是说不受页边距的影响。如果没有指定高或宽的情况，则可以通过 `left`（左边距离）、`top`（顶边距离）、`right`（右边距离）和 `bottom`（底边距离）属性来定义元素的大小。例如，在下面这个示例中，`<div id="pageHeader">` 元素将铺满整个浏览器窗口，但是这个特征只能够在 FF 等现代浏览器中获得支持，而 IE 6 及其以下版本浏览器则不支持该方法。

```

<style type="text/css">
#pageHeader {
    background:#FF99FF;           /* 背景色 */
    position:fixed;                /* 固定定位 */
    top:0;                       /* 顶边距离 */
    left:0;                      /* 左边距离 */
    bottom:0;                    /* 底边距离 */
    right:0;                     /* 右边距离 */
}
</style>
<div id="pageHeader"></div>

```

同样的道理，在没有定义宽度的情况下，如果同时定义了 `left` 和 `right` 属性，则可以在水平方向上定义元素的宽度和位置。在没有定义高度的情况下，如果同时定义了 `top` 和 `bottom` 属性，则可以在垂直方向上定义元素的高度和位置。

8.2 定位布局中参照物和坐标系

在上一节中我们详细讲解了元素的 4 种定位类型，实际上每个元素都拥有定位特性，没有定位就没有元素在网页中的显示位置。不过 CSS 默认每个元素显示为静态定位，所以你可能感觉不到定位在网页布局中的作用。下面我们将重点探讨定位布局中两个重要的概念：参照物和坐标系。

8.2.1 绝对定位的参照物

中学物理中曾经讲解过物体运动，相信你对于参照物这个概念并不陌生。实际上，CSS 定位中的参照物与物理运动中的参照物是比较相似的概念。

阿基米德说：给我一个支点，我将撬起地球。这话有道理，对于 CSS 布局来说，如果你给它一个参考点，CSS 能够帮助你精确定位每个元素的位置。也许你疑惑了：绝对定位不是一直根据网页窗口的内边距来进行定位吗？

是的，在默认状态下，当我们绝对定位网页中某个元素时，浏览器都是以网页的边框来进行定位。不过更准确地说，浏览器是以窗口的边框来进行定位的，不受 `body` 元素的边距影响。很多读者（包含不少设计师）都误认为是受 `body` 元素的影响。

例如，在下面这个示例中我们定义页面的外边距为 100 像素，内边距为 100 像素，然后再定义一个绝对定位的元素，设置它的坐标值为(100px, 100px)。如果在浏览器中预览，则会显示如图 8.11 所示，显示效果说明，绝对定位元素的位置没有受到页边距的影响。

```
<style type="text/css">
body {
    margin:100px;           /* 定义页面外边距 */
    padding:100px;         /* 定义页面内边距 */
}
#box {
    position:absolute;     /* 绝对定位 */
    left:100px;            /* x 轴坐标，距离左边框距离 */
    top:100px;             /* y 轴坐标，距离顶部边框距离 */
    width:200px;           /* 宽度 */
    height:100px;          /* 高度 */
    background:red;        /* 背景色 */
}
</body>
<div id="box"></div>
```

那么是不是绝对定位就不受边距的影响呢？不是的。我们在上面的示例基础上继续做试验，把 `body` 元素定义为绝对定位元素：

```
body {
    margin:100px;          /* 外边距 */
    padding:100px;         /* 内边距 */
    position:absolute;     /* 绝对定位 */
}
```


如果再次预览，则显示效果如图 8.12 所示。这说明绝对定位的元素还是受页边距的影响。这是为什么呢？

原来在第一次试验时，body 元素默认显示为静态定位状态，此时浏览器就根据浏览器窗口的内边框来进行定位。第二次试验时，body 元素被定义为绝对定位元素，这时浏览器就根据 body 元素作为参照物进行定位，浏览器根据 body 元素的内边距的内沿作为参考。



图 8.11 绝对定位不受页边距影响



图 8.12 绝对定位受页边距影响

从上面示例中引出一个新的话题：是不是为所有元素定义绝对定位元素之后，它就具备了定位参照物的特质呢？

是的，不仅如此，所有被定义了相对定位、绝对定位的元素都可以作为 CSS 定位参照物来确定其包含的绝对定位元素的坐标。

有时我们把这个具备 CSS 定位参照物的元素称为包含块。有了这个包含块，我们就可以使 CSS 绝对定位功能发挥到极致。

例如，在下面这个示例中，我们采用双重定位的方法实现让一个方块永远位于浏览器窗口的中间位置，包括水平居中和垂直居中，显示效果如图 8.13 所示。

```
<style type="text/css">
#wrap {/* 定位包含块 */
    position:absolute;
    left:50%;
    top:50%;
    width:200px;
    height:100px;
    border:dashed 1px blue;
}
#box {/* 定位方块 */
    position:absolute;
    left:-50%;
    top:-50%;
    width:200px;
    height:100px;
    background:red;
}
</style>
<div id="wrap">
```

/* 绝对定位 */
/* x 轴坐标 */
/* y 轴坐标 */
/* 宽度 */
/* 高度 */
/* 虚线框 */

/* 绝对定位 */
/* x 轴坐标 */
/* y 轴坐标 */
/* 宽度 */
/* 高度 */
/* 背景色 */

```
<div id="box"></div>
</div>
```

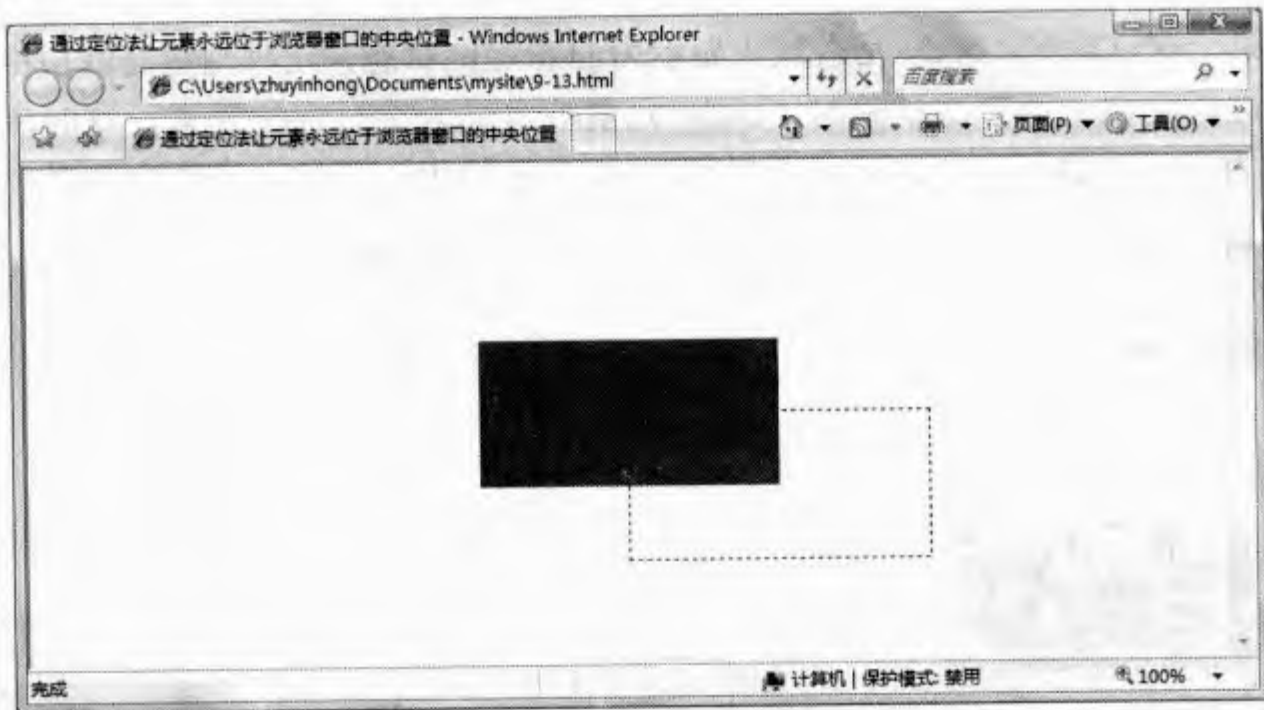


图 8.13 让元素永远位于浏览器窗口的中央位置

你可能知道元素垂直居中是个很麻烦的问题，对于绝对定位的元素来说，如果它位于浏览器的中央将会更麻烦。不过本示例给你一个全新的设计思路。

首先，利用一个辅助元素（即虚线框）绝对定位到浏览器窗口的中央附近。设置它的 x 轴坐标取窗口宽度的一半，y 轴坐标取窗口高度的一半。这样你可以看到，虚线框的左上顶角位于浏览器窗口的中央位置，但是虚线框偏向右下方。

然后再以这个虚线框作为参照物（因为它被定义为绝对定位元素，是一个包含块），定义它包含的元素为绝对定位元素；x 轴坐标取值为虚线框宽度的一半，并加上负号，y 轴坐标取值为虚线框高度的一半，并加上负号，这样定位元素就以虚线框的左上顶角定位参照点反向左上方移动，最终实现绝对中央的位置。注意，一定要设置父子两个定位元素的大小是相同的，当然你可以不为包含块增加虚线框，这样别人就会以为元素自身被放置到窗口中央位置。

这个示例展示了元素二次定位的方法，第一次以浏览器窗口作为参照物，第二次以包含块元素作为参照物，希望读者在此基础上琢磨出更多类似功能的效果。

上面示例展示了以绝对定位元素作为定位参照物的方法，其实你也可以使用相对定位元素作为包含块，能够赋予绝对定位更大的灵活性。

例如，在下面这个示例中，绝对定位就是根据相对定位包含块来定位自己的位置（如图 8.14 所示）。如果我们在 `<div id="pageHeader">` 对象前面增加多个换行标签 `

`，则这时 `<div id="pageHeader">` 对象被迫向下移动，此时你可以看到绝对定位元素也随之向下移动。这说明它始终跟随包含块的位置变化而变化，如图 8.15 所示。

```
<style type="text/css">
#pageHeader {
    border:solid 1px red;                /* 边框 */
    position:relative;                  /* 相对定位，定义包含块 */
}
#pageHeader h2 {
    position:absolute;                  /* 绝对定位 */
    left:40%;                           /* 右移至中间位置 */
}
</style>
```



```
<div id="pageHeader">
  <h1><span>CSS Zen Garden</span></h1>
  <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>
```

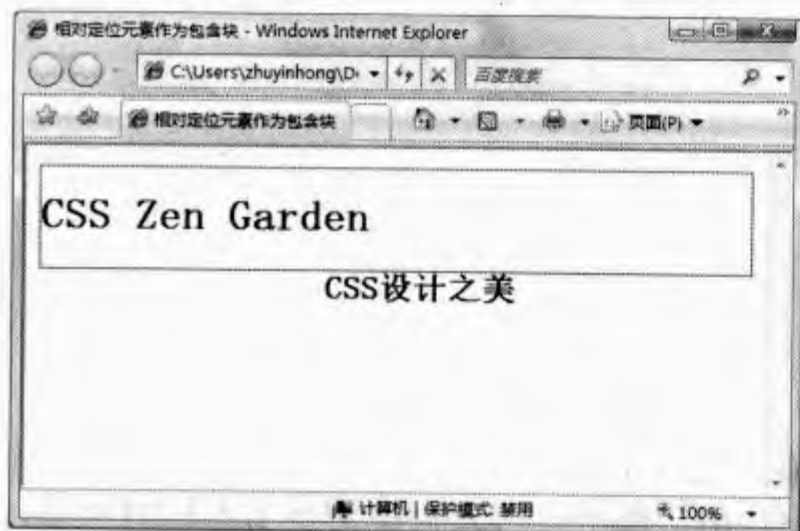


图 8.14 相对定位包含块

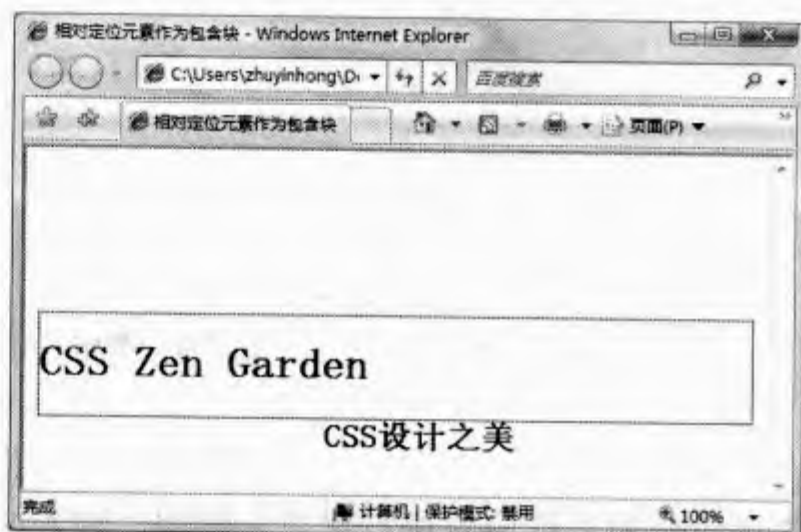


图 8.15 移动相对定位包含块

通过上面的示意图，我们也可以看到，当元素被定义为绝对定位元素之后，其在文档流中的原始位置就不再被保留，绝对定位元素与文档流就没有直接联系。当然你可以通过上面示例的方法来间接实现文档流与绝对定位元素的联系。

8.2.2 绝对定位的坐标系

提及坐标系，你可能会想到数学中所学到的坐标轴和坐标原点等类似的概念，不过我们所讲解的这些定位坐标系不完全是这些概念，CSS 定位的坐标系是由包含块的 4 个内边沿动态确定的。所谓内边沿就是包含块元素的内边距的内沿，也就是元素的内墙壁。

为了更灵活地定位页面元素，CSS 没有提供唯一的类似于左上顶角的坐标原点和坐标系（这是屏幕坐标中常用的定位方法）。它提供了 4 个定位属性：top、right、bottom 和 left。通过联合使用这些属性，你可以根据包含块的 4 个内顶角来定位元素在页面任意位置。这样就避免了仅使用左上顶角为坐标原点进行定位可能会带来的盲区。

- top 属性表示定位元素顶边外壁到包含块元素顶部内壁的距离。
- right 属性表示定位元素右边外壁到包含块元素右侧内壁的距离。
- bottom 属性表示定位元素底边外壁到包含块元素底部内壁的距离。
- left 属性表示定位元素左边外壁到包含块元素左侧内壁的距离。

这里的外壁是指定位元素的外边框，即边框的外沿，而内壁是指包含块元素的内边距的内沿。例如，在下面这个相对定位包含块中包含着一个绝对定位的元素，你可以很直观地看到坐标参照系（如图 8.16 所示）。

```
<style type="text/css">
#wrap {
  position:relative;          /* 相对定位 */
  border:solid 50px red;      /* 边框 */
  padding:50px;              /* 内边距 */
  width:50px;                /* 宽度 */
  height:50px;               /* 高度 */
}
#box {
  position:absolute;          /* 绝对定位 */
```

```

border:solid 50px blue;
margin:50px;
padding:50px;
width:50px;
height:50px;
left:50px;
top:50px;
}
</style>
</head>
<body>
<div id="wrap">包含块包含块包含块
    <div id="box">定位元素定位元素定</div>
</div>

```

/* 边框 */
/* 外边距 */
/* 内边距 */
/* 宽度 */
/* 高度 */
/* x 轴坐标 */
/* y 轴坐标 */

上面代码是以包含块的左上内顶角作为坐标原点进行定位的，当然你也可以使用其他 3 个内顶角作为坐标原点进行定位。例如，输入如下坐标值，你可以以右下内顶角为坐标原点进行定位（如图 8.17 所示）。

```

#box {
    right:50px;
    bottom:50px;
}

```

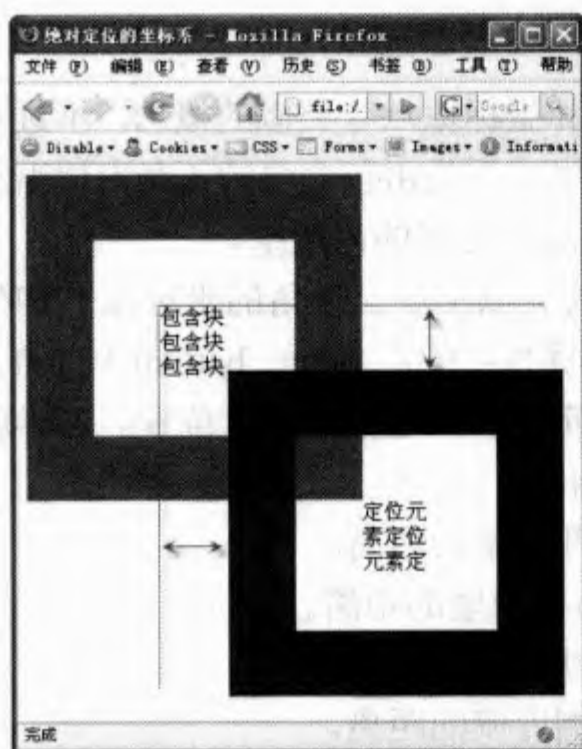


图 8.16 左上内顶角为坐标原点



图 8.17 右下内顶角为坐标原点

同样的道理，如果使用 `top` 和 `right` 属性结合，可以以右上内顶角作为坐标原点进行定位（如图 8.18 所示）；使用 `left` 和 `bottom` 属性结合，可以以左下内顶角作为坐标原点进行定位（如图 8.19 所示）。

在使用定位属性时，可以用任意多个属性组合进行定位，也可以用单个属性定位单方向上的坐标，另一方向上将采用默认值，这个问题将在下一节中详细讲解。

当同时使用 `left` 和 `right` 属性进行定位时，`left` 属性值为有效值。同理，如果同时使用 `top` 和 `bottom` 属性进行定位时，`top` 属性值有效。当同时定义了 3 个或以上的属性时，遵循上面的优先原则进行定位。

当然，绝对定位的坐标系问题还是比较复杂的，本节主要探讨包含块为块状元素的情形，

不过如果包含块为行内元素时,这种情况还会发生变化,且不同浏览器在解析时略有差别。限于篇幅我就不再深究了,否则读者会更加模糊。等读者初步掌握了 CSS 定位之后,再去探索各种特殊定位现象。

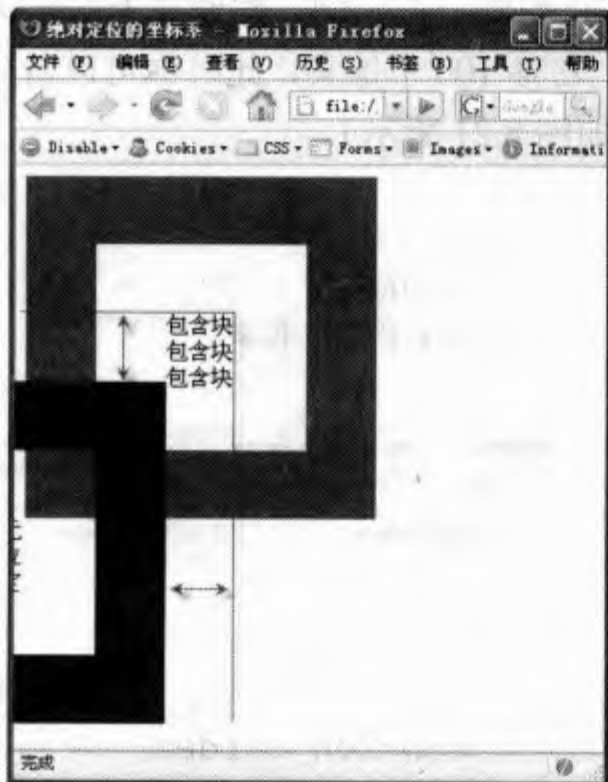


图 8.18 右上内顶角为坐标原点

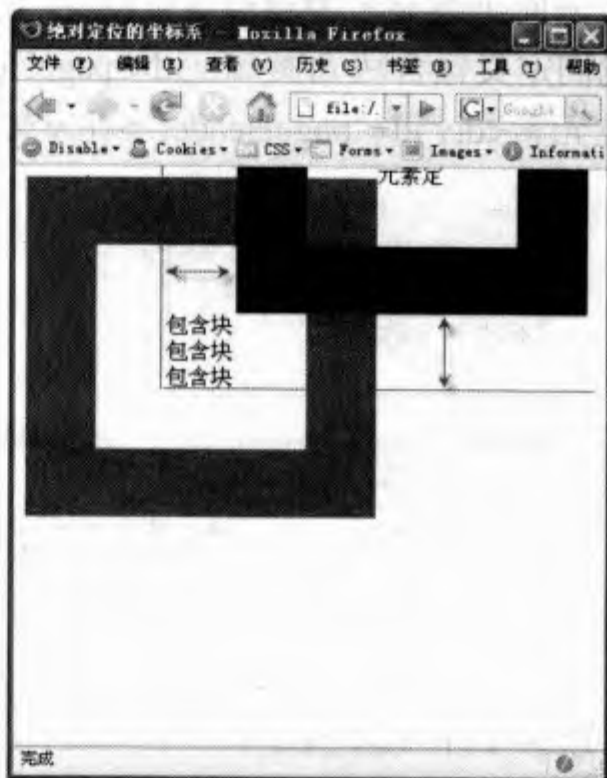


图 8.19 左下内顶角为坐标原点

8.2.3 绝对定位的相对论

所谓绝对定位的相对论,就是当绝对定位处于特殊状态下时会呈现出相对定位的特性。这是很多设计师容易忽略的细节,也是让很多初学者迷惑不解的地方,如果不明白事由,很容易被 CSS 定位搞得焦头烂额。

在第 8.2.1 节中我们曾经说过,相对定位元素能够随文档流自由流动,绝对定位元素脱离文档流,不再受文档流的影响。但是当绝对定位元素没有被显式指明坐标系时,这种情况就会发生变化。例如,在第 8.2.1 节中曾经介绍了一个相对定位元素包含绝对定位元素的示例,当相对定位元素随文档流移动时,绝对定位元素也跟随移动。现在我们把相对定位元素改为绝对定位元素,这时你会看到绝对定位元素随着文档流在移动(如图 8.20 所示)。

```
<style type="text/css">
#pageHeader {
    border:solid 1px red;                /* 边框 */
    position:absolute;                  /* 绝对定位 */
}
#pageHeader h2 {
    position:absolute;                  /* 绝对定位 */
    left:40%;                          /* x 轴坐标 */
}
</style>
<br /><br /><br /><br /><br /><br />
<div id="pageHeader">
    <h1><span>CSS Zen Garden</span></h1>
    <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
</div>
```

所以,我们可以这样总结:当绝对定位元素没有显式指明 top、right、bottom 或 left 定位属

性时，它暂时还未脱离文档流，并受文档流的影响，具有相对定位的特性，但是它在文档流中的位置已经不存在了，其大小不会影响到包含元素。你可以从上面示例中看到。

另外，如果绝对定位元素仅指明 x 轴或 y 轴坐标值，则它只能具备该方向上的定位能力，另一个方向仍然显示为相对定位特性。这话不好理解，我们看一个示例。假设在上面示例基础上，我们定义包含块元素<div id="pageHeader">在 x 轴上右移 100 个像素，此时你可以看到，绝对定位元素<div id="pageHeader">在 y 轴上向下移动（如图 8.21 所示）。

```
#pageHeader {
    border:solid 1px red;
    position:absolute;
    left:100px;
}
```

```
/* 边框 */
/* 绝对定位 */
/* x 轴右移 100 像素 */
```



图 8.20 绝对定位元素的相对特性



图 8.21 绝对定位元素的部分相对特性

8.2.4 相对定位参照物和坐标系

相对定位的参照物永远是它本身，也就是元素的原始位置。但是其坐标系与绝对定位元素坐标系就略有不同了，它提供了 4 个定位属性：top、right、bottom 和 left。这些属性的定位坐标说明如下：

- top 属性表示定位元素顶边外壁到原始位置顶部外壁的距离；
- right 属性表示定位元素右边外壁到原始位置右侧外壁的距离；
- bottom 属性表示定位元素底边外壁到原始位置底部外壁的距离；
- left 属性表示定位元素左边外壁到原始位置左侧外壁的距离。

例如，在下面这个示例中，我们可以很清楚地看到相对定位时位移的变化情况（如图 8.22 所示）。

```
<style type="text/css">
span {
    border: dashed 1px red; /* 包含元素的虚线框，描绘相对定位元素的原始位置 */
}
span img {
    position:relative; /* 相对定位 */
    left:50px; /* x 轴坐标 */
    top:50px; /* y 轴坐标 */
    margin:50px; /* 外边框 */
    width:50px; /* 宽度 */
    height:50px; /* 高度 */
    border:solid 50px blue; /* 边框 */
}
```



```

}
</style>
<span></span>

```

但是对于行内文本来说，这种相对位移会变得很复杂，此时就不应用这个简单的坐标系来进行定位了，而应该根据相对位置来确定。

例如，下面这个示例将演示行内文本被定义为相对定位之后，所出现的复杂情况（如图 8.23 所示）。此时你可以看到，如果简单的使用某个坐标系来定位流动的文本行是无法实现的，唯一的方法就是根据文本行整体进行位移来定位；或者你可以把多行文本拆分为多行，以不同的坐标系来进行定位。

```

<style type="text/css">
span.origin {
    border: dashed 2px blue; /*包含元素的虚线框，描绘相对定位元素的原始位置 */
}
span.relative {
    position:relative;
    left:50px;
    top:50px;
    background:#FF66CC;
}
</style>
<p class="p3"><span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力，构思一个专业级的网页。
<span class="origin"><span class="relative">让我们用慧眼来审视，充满理想和激情去学习 CSS
这个不朽的技术，</span></span>最终使自己能够达到技术和艺术合而为一的最高境界。</span></p>

```

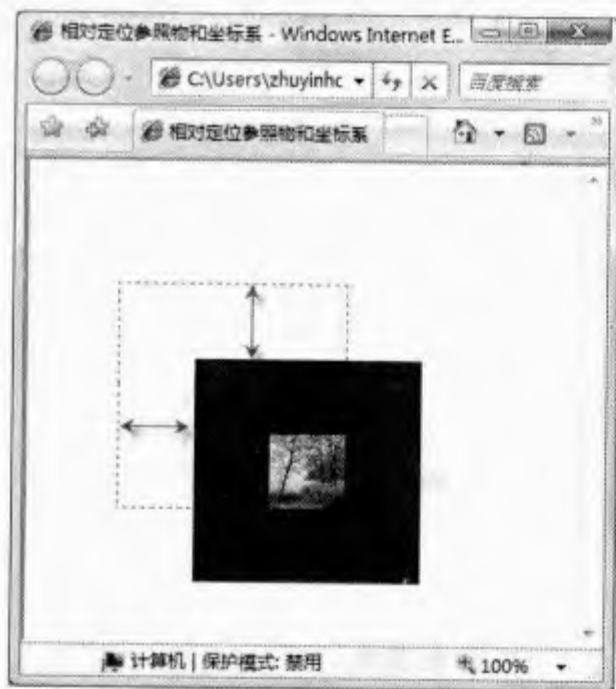


图 8.22 相对定位的位移坐标

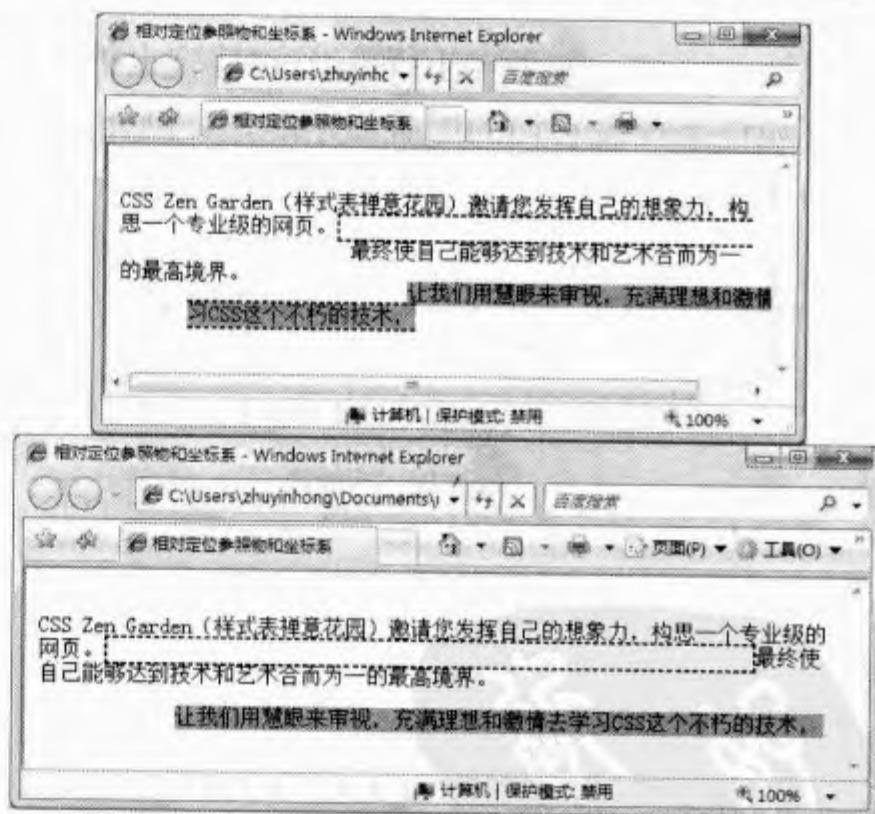


图 8.23 相对定位行内文本的位移坐标

8.3 定位布局中元素层叠处理

当多个元素同时被定位之后，有时候会出现相互重叠现象，这种重叠现象其实也是一种设计技巧，设计师可通过它设计各种网页特效。在第 8.1.4 节中我们曾经讲解过相对定位的层叠

问题。本节将在此基础上继续深入探讨定位布局中元素的层叠问题，以及各种处理技巧。

8.3.1 定位元素的层叠顺序

CSS 通过 `z-index` 属性来排列不同定位元素之间的层叠顺序。该属性可以设置任意整数值，数值越大，所排列的顺序就越靠上。

例如，在第 8.1.4 节中我们曾经列举了一个红盒子、蓝盒子和绿盒子的相对定位排序问题。在默认状态下，它们按先后位置确定自己的位置关系，越排在后面，则其显示的位置就越靠上。下面我们利用 `z-index` 属性来改变它们的层叠顺序（完整代码可以参阅第 8.1.4 节示例）。

这时你可以看到 3 个不同背景色的盒子所层叠的顺序发生了变化（如图 8.24 所示）。原来排在最上面的绿色盒子现在被排在下面，原来排列顺序可以参阅图 8.9 所示。

```
<style type="text/css">
#box1 { /* 红盒子 */
    z-index:3;           /* 排在最上面 */
}
#box2 { /* 蓝盒子 */
    z-index:2;           /* 排在中间 */
}
#box3 { /* 绿盒子 */
    z-index:1;           /* 排在最下面 */
}
</style>
<div id="box1">红盒子</div>
<div id="box2">蓝盒子</div>
<div id="box3">绿盒子</div>
```

同样的道理，如果在上面示例的基础上把所有盒子定义为绝对定位元素，一样可以通过 `z-index` 属性来控制它们的层叠显示顺序。例如，重新编写上面示例的样式表代码，然后让红盒子排列在上面，绿盒子排列在最下面，则显示效果如图 8.25 所示。

```
<style type="text/css">
#box1, #box2, #box3 { /* 盒子的公共属性 */
    height:100px;      /* 高度 */
    width:200px;       /* 宽度 */
    position:absolute; /* 绝对定位 */
}
#box1 { /* 红盒子 */
    background:red;     /* 红色背景 */
    z-index:3;          /* 排列在最上面 */
    left:100px;         /* x 轴坐标 */
}
#box2 { /* 蓝盒子 */
    background:blue;    /* 蓝色背景 */
    top:50px;           /* y 轴坐标 */
    left:50px;          /* x 轴坐标 */
    z-index:2;          /* 层叠顺序 */
}
#box3 { /* 绿盒子 */
    background:green;   /* 绿色背景 */
    top:100px;          /* x 轴坐标 */
    z-index:1;          /* 排列在最底下 */
}
</style>
```

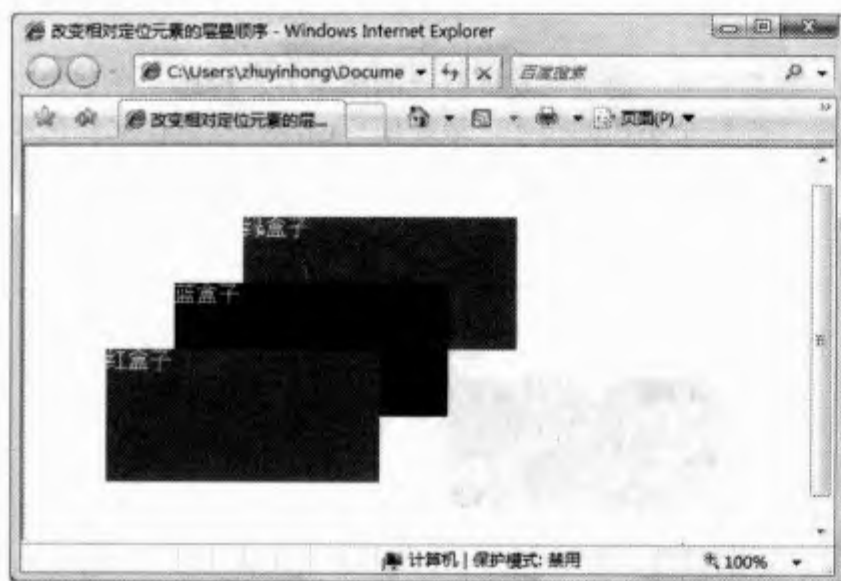



图 8.24 改变相对定位元素之间覆盖顺序

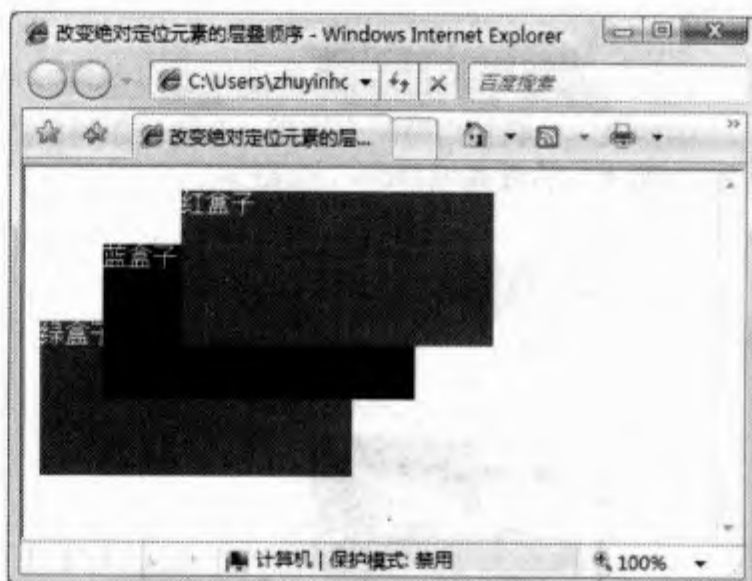


图 8.25 改变绝对定位元素之间覆盖顺序

绝对定位元素和相对定位元素如果混合在一起，它们之间也会严格遵循这样的层叠排序规则，不会存在绝对定位元素优先于相对定位元素的现象。

8.3.2 定位元素与文档流的层叠关系

在默认状态下，定位元素（包括相对定位和绝对定位）永远都会覆盖在文档流对象之上。当然，这种关系也不是绝对的。

例如，在 IE 6 及其以下版本浏览器中，类似于 select 元素的窗口控件就会显示在绝对定位元素之后，也就是说 select 元素的窗口控件还不完全都支持 z-index 属性。在下面示例中，下拉列表对象覆盖了绝对定位元素（如图 8.26 所示）。

```
<style type="text/css">
#box {
    height:100px;          /* 宽度 */
    width:200px;           /* 高度 */
    background:red;        /* 红色背景 */
    position:absolute;     /* 绝对定位 */
}
</style>
<div id="box">红盒子</div>
<select name="select" size="1" id="select">
    <option value="1">---1---</option>
    <option value="2">---2</option>
    <option value="3">---3---</option>
</select>
```

当然，我们也可以通过将 z-index 属性取负值的方法，使定位元素隐藏在文档流对象的下面。例如，在下面这个示例中，我们通过定义 z-index 属性为负值，使绝对定位元素隐藏在文本段的下面（如图 8.27 所示）。

```
<style type="text/css">
#box {
    height:100px;          /* 宽度 */
    width:200px;           /* 高度 */
    background:red;        /* 红色背景 */
    position:absolute;     /* 绝对定位 */
    z-index:-1;            /* 隐藏在文档流下面 */
}
</style>
```

```
</style>
<div id="box"></div>
<p class="p3"><span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力, 构思一个专业级
的网页。让我们用慧眼来审视, 充满理想和激情去学习 CSS 这个不朽的技术, 最终使自己能够达到技术和艺术
合而为一的最高境界。</span></p>
```

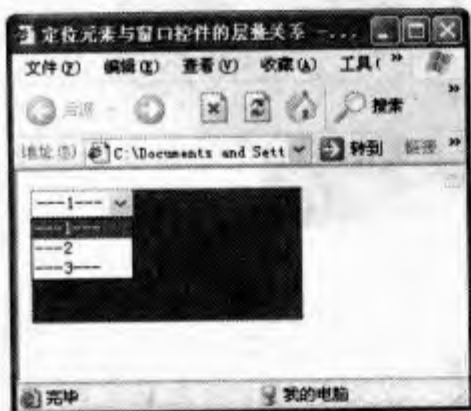


图 8.26 改变相对定位元素之间覆盖顺序

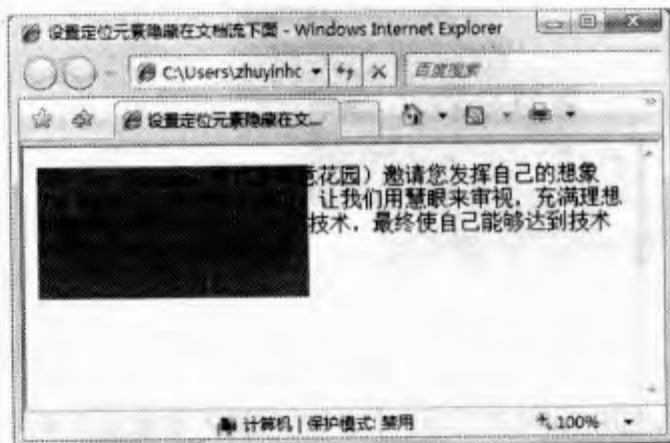


图 8.27 设置定位元素隐藏在文档流下面

设置 `z-index` 属性为 `null` 可以移除该属性, 或者定义 `z-index` 属性为 `0`, 则它们都具有相同的作用。但是对于定位元素来说, 不管是否定义 `z-index` 属性, 默认状态下它们的层叠顺序都会高于正常的文档流。

8.3.3 定位元素的层叠包含关系

在 CSS 定位中还有一个很奇怪的现象, 这种现象目前仅在 IE 浏览器中存在。即当定位元素位于 HTML 不同结构层次时, 所定位的元素的层叠级别存在很大差异, 甚至无法进行比较。为了说明这个问题, 我们举一个简单的示例。

例如, 请输入如下示例代码。在这个示例中, 相对定位元素 `<div id="wrap">` 下包含两个子元素: `<div id="header">` 和 `<div id="main">`。`<div id="header">` 是一个相对定位元素, 在其中包含一个绝对定位元素 `<h1 id="logo">`, 而 `<div id="main">` 也是一个绝对定位元素。

由于 `<h1 id="logo">` 和 `<div id="main">` 都是绝对定位元素, 但是它们在 HTML 结构中所处的层级不同, 导致无论 `<h1 id="logo">` 绝对定位元素的层叠值有多高, 在 IE 浏览器下它都被覆盖在 `<div id="main">` 对象的下面 (如图 8.28 所示), 但是在非 IE 浏览器中能够使用 `z-index` 属性进行层叠控制 (如图 8.29 所示)。

```
<style type="text/css">
body {
    padding:0;                                /* 清除页边距 */
    margin:0;                                  /* 清除页边距 */
}
#wrap, #header {
    position:relative;                         /* 定义相对定位元素 */
}
#logo {
    position:absolute;                        /* 绝对定位 */
    width:231px;                              /* 宽度 */
    height:159px;                             /* 高度 */
    left:20px;                                /* x 轴坐标 */
    top:20px;                                 /* y 轴坐标 */
    background:url(images/logo1.gif) no-repeat; /* 背景图像 */
    z-index:1000;                             /* 层叠值 */
    text-indent:-999px;                       /* 隐藏网页标题文本 */
    margin:0;                                  /* 清除标题元素的默认边距 */
}
```



```
}
#main {
    width:100%;           /* 宽度 */
    height:200px;         /* 高度 */
    position:absolute;    /* 绝对定位 */
    background:#6699FF;   /* 背景色 */
    top:60px;             /* y 轴坐标 */
    text-align:center;     /* 水平居中 */
}
</style>
<div id="wrap">
    <div id="header">
        <h1 id="logo">网页标题</h1>
    </div>
    <div id="main">主体区域</div>
</div>
```

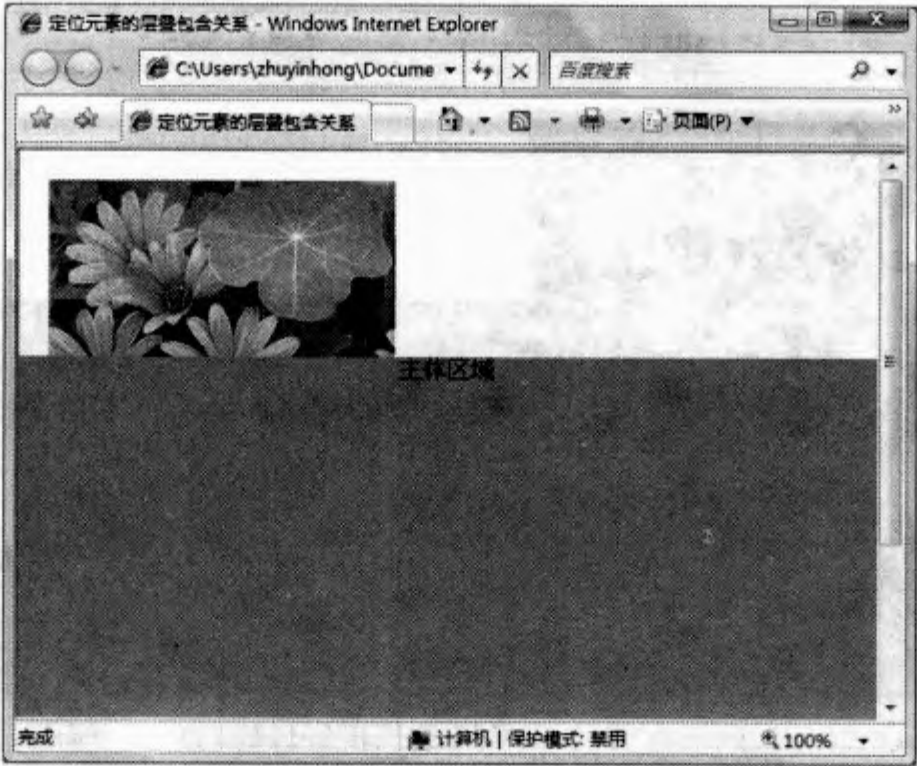


图 8.28 IE 下定位元素的层叠包含关系效果

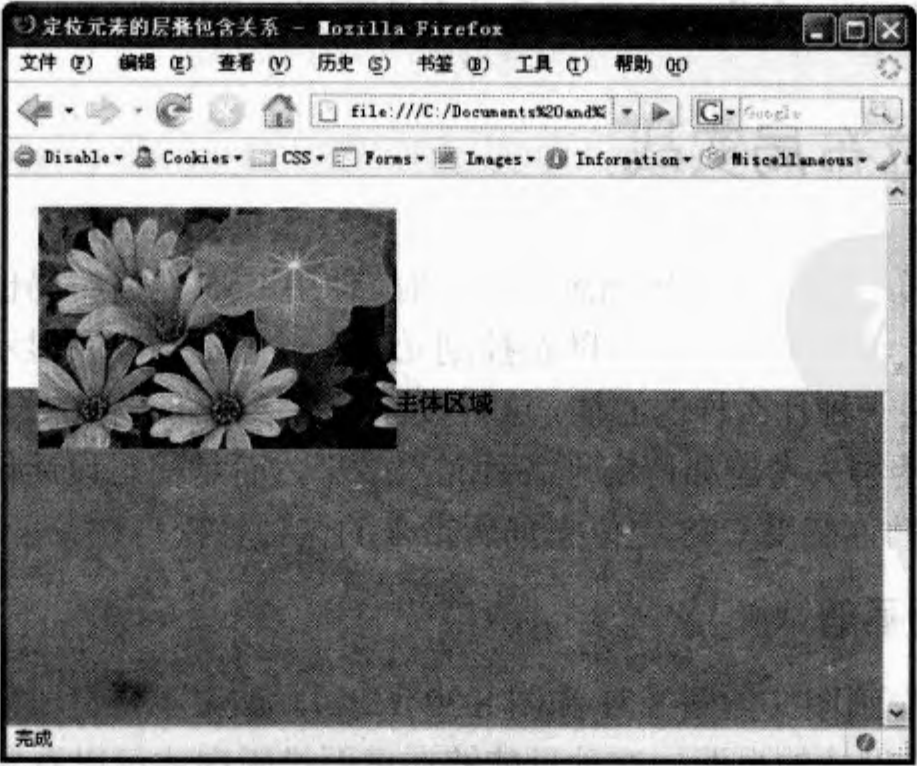


图 8.29 非 IE 下定位元素的层叠包含关系效果

解决 IE 浏览器的这个特殊现象，不妨从同级 HTML 结构入手来解决。也就是说，如果我们定义 `<h1 id="logo">` 对象的父级包含块 `<div id="header">` 对象的层叠值大于 `<div id="main">` 对象的层叠值，让父级元素之间以平级身份进行层叠排序，这样如果父级元素排在上面，则它们的子孙元素也都跟着沾光。具体说就是在 `<div id="header">` 对象中增加一个 `z-index` 属性值：

```
#header {
    z-index:1;                      /* 增加层叠顺序 */
}
```

通过上面的样式，定义 `<div id="header">` 对象排在 `<div id="main">` 对象上面，这样 `<div id="header">` 对象包含的子元素就会层叠在上面（如图 8.30 所示）。



图 8.30 解决 IE 浏览器的定位元素的层叠包含问题

由于牵扯到其他知识，有关 CSS 的层叠关系的更深入分析请参考第 10.4 节相关内容。

8.4 CSS 定位布局实战

当你还在为浮动布局的一些兼容问题而烦恼时，使用 CSS 定位布局也许会让你找到传统排版布局的轻松和自在。想一想现在可以随意控制元素在页面不同位置显示，而不再碍于 HTML 结构的条条框框，这是一种什么样的心情，这种设计又是一种什么样的设计思维。与 PageMaker 排版软件一样，你不用再去考虑如何实现各种布局效果，而是构思设计什么样的效果。本节将通过几个典型示例帮助你感受 CSS 定位布局所带来的这种轻松自在。

8.4.1 米老鼠卡通画册

看看下面这两幅漫画图（如图 8.31 和图 8.32 所示），如果不是浏览器窗口的提示，你可能想象不到它是用 CSS 设计的页面。这种只能够在桌面排版中才可以看到的效果，现在能够用 CSS 来实现，确实令人觉得很新鲜和兴奋。



图 8.31 漫画式网页设计效果 1



图 8.32 漫画式网页设计效果 2

但是如果查看源代码, 会发现该页面的设计很简单, 它主要应用 CSS 定位技术来布局页面, 具体说就是页面中的漫画情景由背景图像来实现, 而所标注的语言通过 CSS 定位来精确控制。下面我们就来详细分析它的实现过程。

首先明确, 这幅作品是禅意花园收藏的第 099 号作品 (<http://www.csszengarden.com/?cssfile=099/098.css>), 整个设计思路如图 8.33 所示。

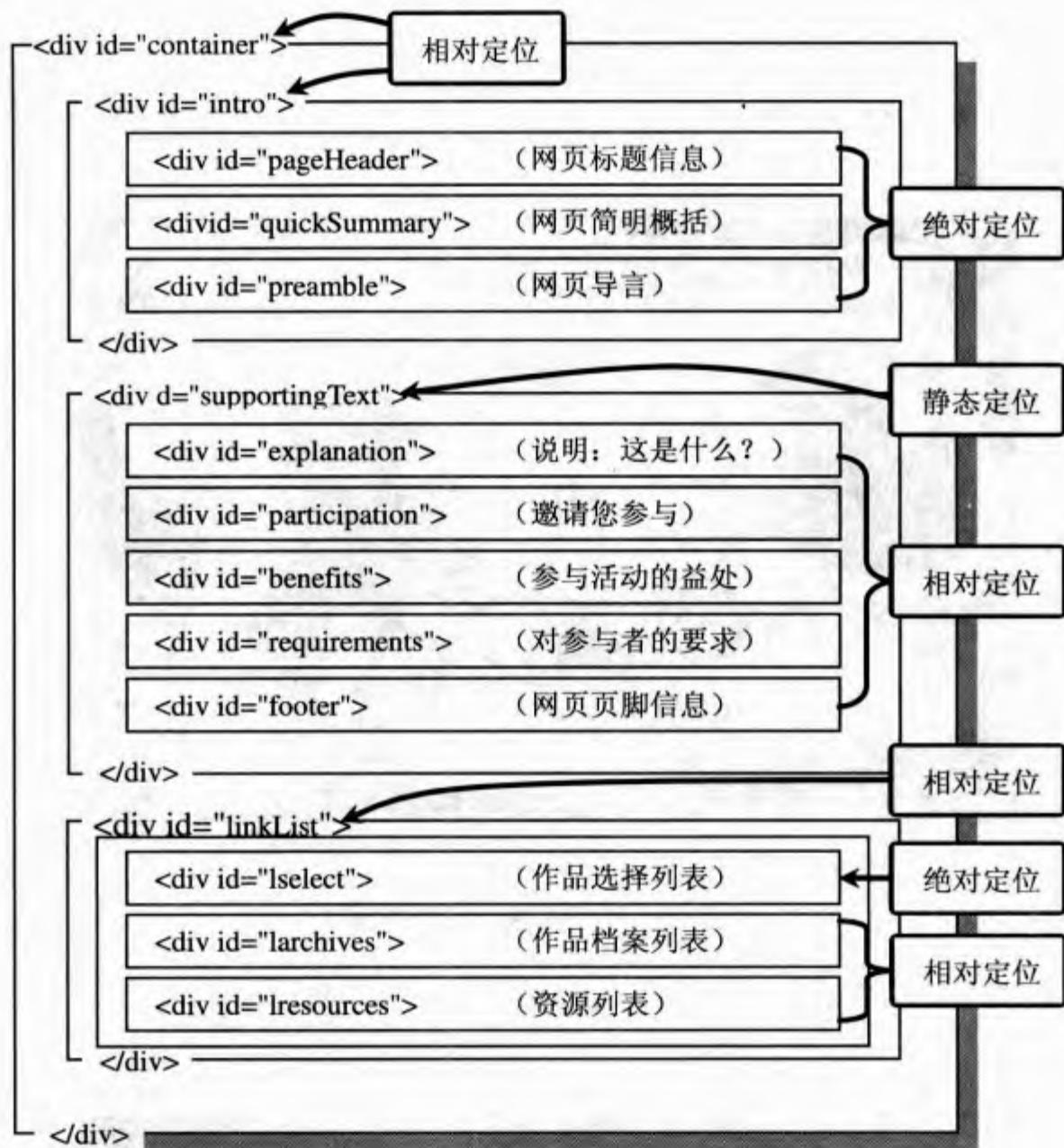


图 8.33 米老鼠卡通画册网页布局的结构定位思路

在 CSS 定位布局中，一般遵循“外部相对定位、内部绝对定位”的设计思路 and 原则，即外围框架元素定义为相对定位元素，而子框架或子对象以绝对定位呈现。通过相对定位定义包含块，这样外包含框能够适应文档流的移动和变化，内部结构和对象以绝对定位的方式准确确定各自的位置。

从上面的结构定位示意图中，我们也可以清楚看到不同层次元素被定位的类型。`<div id="container">`对象作为页面的总包含框，被定义为相对定位元素之后，就可以实现让网页内所有绝对定位元素居中显示的效果。然后定义`<div id="intro">`、`<div id="explanation">`、`<div id="participation">`、`<div id="benefits">`、`<div id="requirements">`、`<div id="footer">`和`<div id="linkList">`这 7 个对象为相对定位元素，这样让它们按着文档流的先后顺序排列在网页中。然后在这些相对定位元素内部再通过绝对定位的方式固定每个对象所包含的子元素或对象。下面我们第 1 个模块`<div id="intro">`对象为例进行讲解。

第 1 步，定义网页页面属性，这里主要注意设置页面的宽度和背景。通过限制最低网页宽度以防止浏览器窗口缩小可能造成的布局重叠现象，这是 CSS 定位布局中经常遇到的问题。只要固定宽度和高度，这样的问题都可以避免。设计背景图像主要衬托一种卡通漫画的基本氛围，如图 8.34 所示。

```

body { /* 页面基础属性 */
    text-align:center; /* IE 下页面水平居中 */
    min-width:760px; /* 非 IE 下限制最低宽度 */
}
    
```



```

line-height:100%;                                /* 固定行高 */
background:url(paper.jpg) repeat-y #FFF center top; /* 垂直平铺背景图像 */
}
#container { /* 页面包含框基本属性 */
    text-align:left;                                /* 文本左对齐 */
    margin:0 auto;                                  /* 非 IE 下页面水平居中 */
    width:760px;                                    /* 固定页面宽度 */
    position:relative;                              /* I 定义包含块 */
}

```

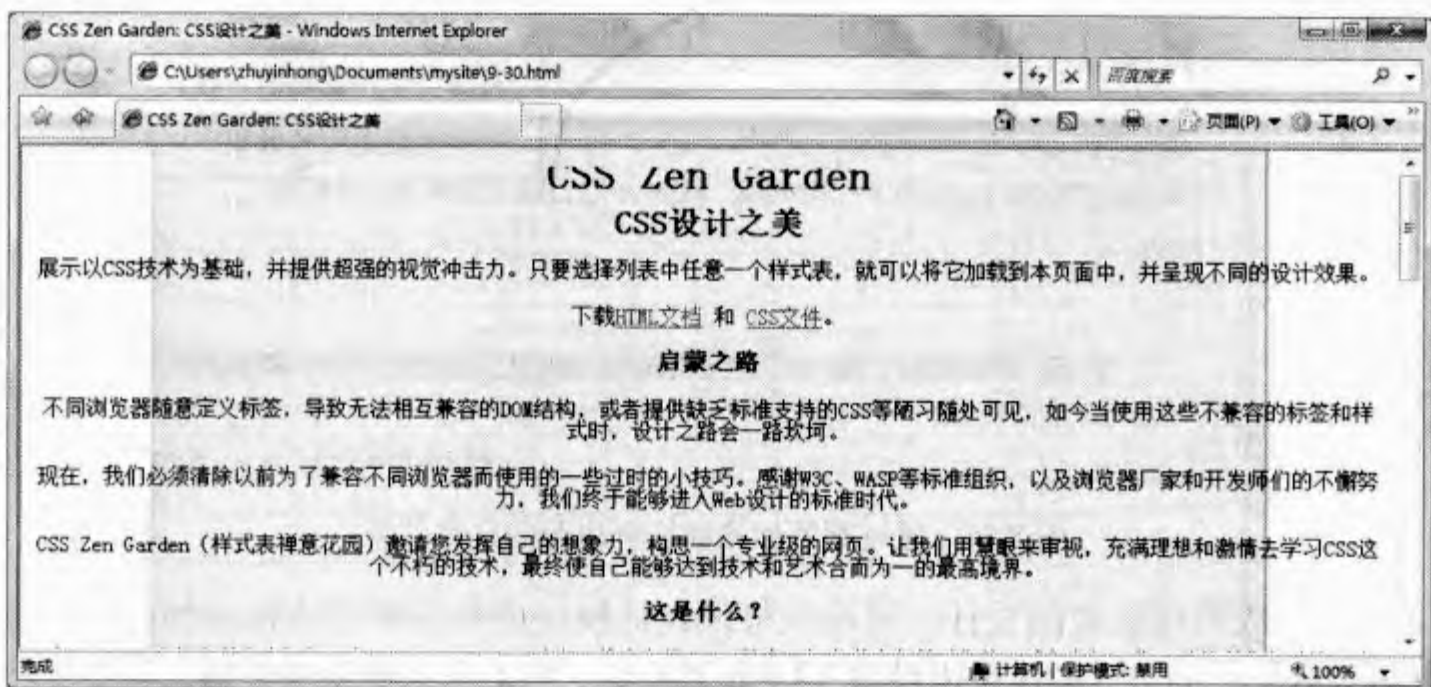


图 8.34 设置卡通式网页属性

第2步, 定义第1个模块的基本定位类型。以相对定位方式确定包含块的定位类型, 由于在默认状态下 div 元素呈现为块状元素, 它的宽度为 100%。固定第1个模块的高度, 目的是在文档流中强迫第2个模块排在其后面 (即距离页面包含块顶部 1385 像素的位置), 如果不显示定义该模块的高度, 则模块高度为 0, 这样第2个模块就会与第1个模块发生重叠。

```

#intro { /* 漫画页包含块 */
    position:relative;                                /* 定义包含块 */
    height:1385px;                                    /* 固定高度 */
    margin-top:40px;                                  /* 增加顶部外边距 */
}

```

第3步, 隐藏不需要的子栏目和内容 (如网页标题<div id="pageHeader">部分), 并利用禅意花园文档底部的备用标签来定义一个背景图像, 为模块顶部增加一个挡板 (如图 8.35 所示), 这个设计技巧也是很值得学习的, 希望读者能够借鉴此种设计方法。

```

#pageHeader {
    display:none;                                    /* 定义包含块 */
}
#extraDiv1 { /* 额外的备用标签 1 */
    position:absolute;                                /* 绝对定位 */
    height:40px;                                       /* 固定高度 */
    width:820px;                                       /* 固定宽度 */
    top:0;                                              /* 以浏览器窗口边框为参照物进行 x 轴定位 */
    background:url(paperedge.jpg) no-repeat bottom; /* 定义背景图像 */
    left:50%;                                          /* 以浏览器窗口边框为参照物进行 y 轴定位 */
    margin-left:-410px;                                /* 与 left 属性配合实现该元素居中显示 */
}

```

由于禅意花园所提供的这些额外备用标签都位于网页包含框<div id="container">的外面(参阅第 8.2.3 节详细分析), 如何让它们也能够随时居中布局是一个很头疼的问题。不过这里使用 left 和 margin-left 属性配合设计的方法也值得读者思考, 其实在第 8.2.1 节中我们曾经讲解了一种完全使用绝对定位来实现布局居中问题的方法, 它们的设计思路实际上都是相同的。

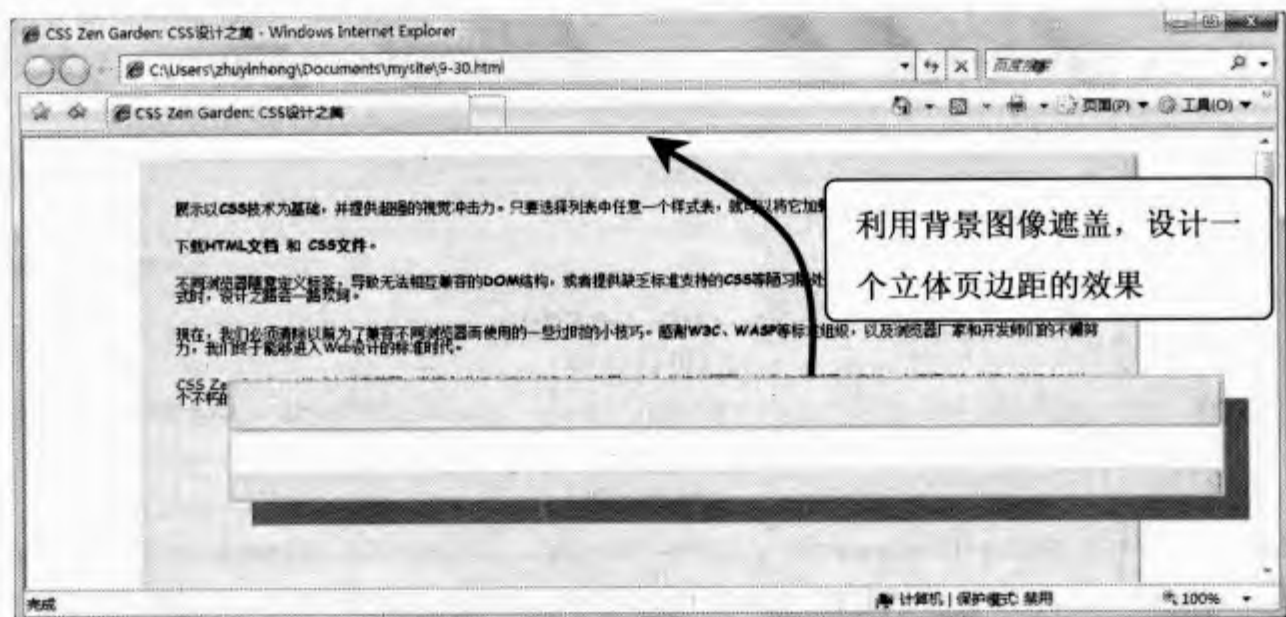


图 8.35 通过额外标签模拟页边距的立体效果

第 4 步, 完成总体框架的设计, 现在就可以设计模块内部的每个小板块的大小、位置和背景图像。在<div id="intro">包含块内包含了 3 个子模块。第 1 个子模块被隐藏, 第 2 个子模块的定位方法如下(显示效果如图 8.36 所示)。



图 8.36 定位第 1 模块的第 2 个子模块布局效果

```
#quickSummary { /* 定位第 2 个子模块 */
    position: absolute;
    left: 6px;
    top: 9px;
    width: 750px;
    height: 491px;
    background: url(P1PANEL1.jpg) no-repeat black;
}
```

/* 绝对定位 */
/* 距离包含块左侧距离 */
/* 距离包含块顶部距离 */
/* 固定宽度 */
/* 固定高度 */
/* 附加背景图像 */


```
#quickSummary p.p1 span { /* 定位第 2 个子模块第 1 段内容 */
    position: absolute; /* 绝对定位 */
    left: 71px; /* 距离父级包含块左侧距离 */
    top: 28px; /* 距离父级包含块顶部距离 */
    width: 328px; /* 固定宽度 */
    height: 80px; /* 固定高度 */
    font-size: 16px; /* 字体大小 */
}
#quickSummary p.p2 span { /* 定位第 2 个子模块第 2 段内容 */
    position: absolute; /* 绝对定位 */
    left: 551px; /* 距离父级包含块左侧距离 */
    top: 0; /* 距离父级包含块顶部距离 */
    font: 9px Arial, Helvetica, sans serif; /* 字体属性 */
}
```

第 5 步, 定义第 1 模块的第 3 子模块, 固定其大小、位置和所需要的背景图像, 并在其内部定位每个段落对象的显示大小、相对父包含块的位置和所需要的背景图像 (如图 8.37 所示)。下面显示为第 3 子模块的定位样式, 其所包含的每个段落的定位样式就不再列举了。

```
#preamble p.p1 {
    position: absolute; /* 绝对定位 */
    left: 5px; /* 距离父包含块左侧距离 */
    top: 506px; /* 距离父包含块顶部距离 */
    width: 366px; /* 固定宽度 */
    height: 428px; /* 固定高度 */
    background: url(P1PANEL2.jpg) no-repeat black; /* 定义背景图像 */
}
```



图 8.37 定位第 1 模块的第 3 个子模块布局效果

上面仅就第 1 大模块的定位布局进行了简单的讲解。实际上, CSS 定位布局中的设计思路比较单纯, 相互模块之间的直接影响比较弱, 所以对于初学者来说可以很快速地入手。

8.4.2 禅意花园展室

纯粹 CSS 定位布局固然很简单, 设计起来也很方便, 但是不是最佳布局选择。如果一味追求视觉艺术或创意, 发挥一下 CSS 定位布局的优势也未尝不可, 但是对于以文本信息为主体的

网页设计,使用这种方法倒显得不方便。设计师往往更钟情于以流动布局为主,恰当应用定位布局来解决个别模块的特殊显示需求。本节就介绍一个以流动布局为主体,恰当使用定位布局进行补充的设计案例。

这是禅意花园第 148 号作品 (<http://www.csszengarden.com/?cssfile=148/148.css>),设计如图 8.38、图 8.39 所示。



图 8.38 禅意花园网页设计效果 1



图 8.39 禅意花园网页设计效果 2

整个页面的设计思路是这样:隐藏第 1 大模块(<div id="intro">)主要内容,把其中第 2 子模块的第 2 段文本定位到页面的左上顶角;第 2 大模块(<div id="supportingText">)以自然流动的方式进行布局;然后采用绝对定位的方式把第 3 大模块定位到页面右侧,显示为一个狭长的侧栏,各个模块的定位类型如图 8.40 所示,设计思路如图 8.41 所示。

在整体设计思路上也遵循“外部结构相对定位,内部结构和对象绝对定位”的原则,把网页包含框<div id="container">定义为包含块,作为内部元素绝对定位的参照物。设计的具体过程如下。

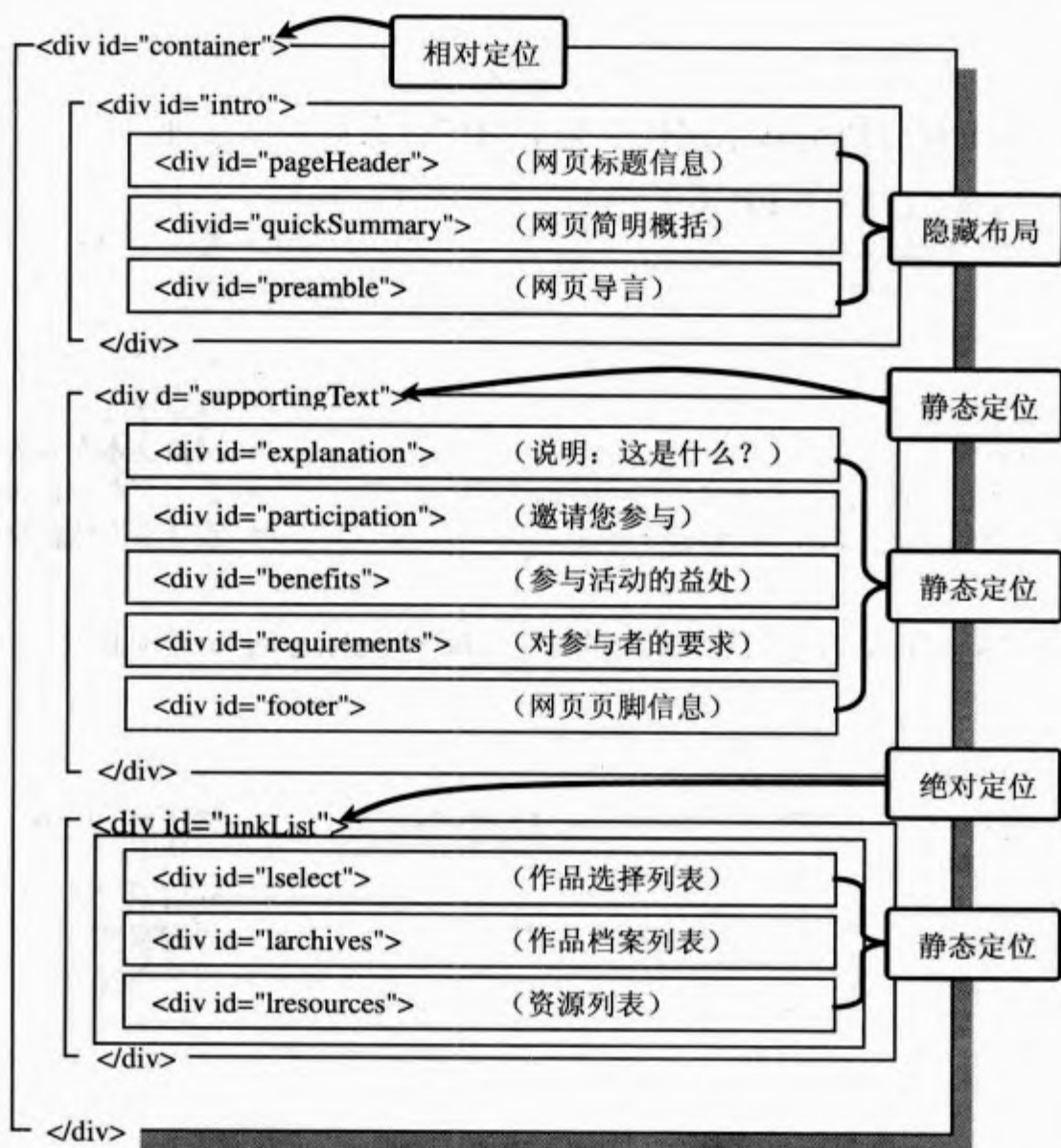


图 8.40 禅意花园展室网页布局的结构定位思路

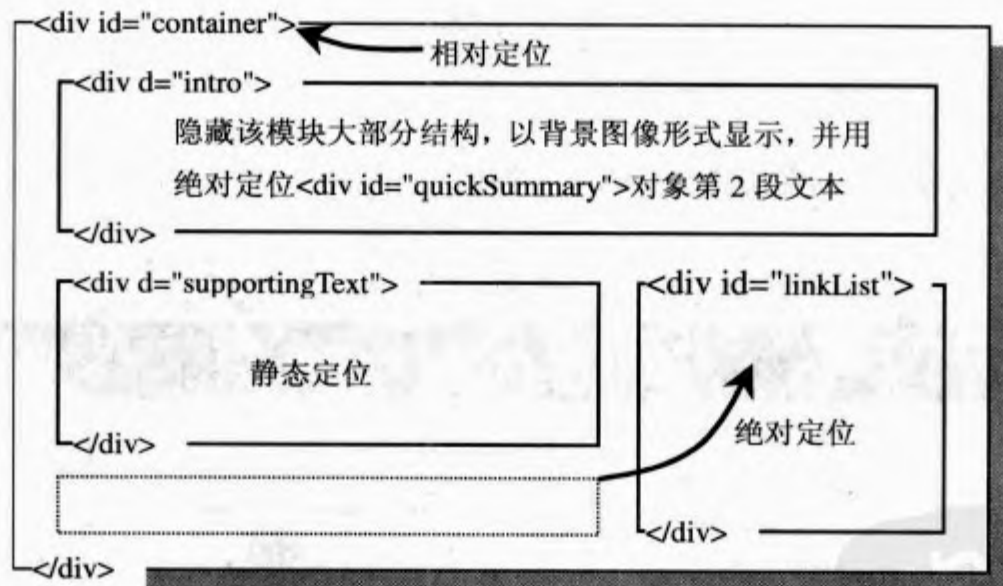


图 8.41 禅意花园展室网页布局设计思路示意图

第 1 步，设置网页基本属性。这里主要是设计背景色、清除页边距、设置网页居中，另外还可以设计网页的字体基本属性。

```
body { /* 页面基本属性 */
    background: #444444; /* 背景色 */
    padding: 0px; /* 清除页边距 */
    margin: 0px; /* 清除页边距 */
    font: 13px Georgia, Serif; /* 字体基本属性 */
    color: #7f7f7f; /* 字体颜色 */
}
```

```
text-align: center; /* 网页居中 */
}
```

第2步, 设计网页包含块的基本属性, 为下面模块布局奠定基础:

```
#container { /* 网页包含框基本样式 */
    background: #5d5d5d; /* 背景色 */
    position: relative; /* 定义包含块 */
    padding: 0px; /* 内边距 */
    margin: 0px auto; /* 水平居中 */
    width: 760px; /* 固定宽度 */
    text-align: left; /* 网页文本左对齐 */
    border-left: 1px solid #fff; /* 设计页左侧修饰线 */
    border-right: 1px solid #fff; /* 设计页右侧修饰线 */
}
```

第3步, 设计展室封面。在第1个模块的<div id="pageHeader">子模块中定义一个大的背景图像:

```
#pageHeader { /* 网页封面设计效果 */
    background: url(header_bg.jpg) no-repeat; /* 设计背景图像 */
    padding: 0px; /* 内边距 */
    margin: 0px; /* 外边距 */
    width: 760px; /* 固定宽度 */
    height: 400px; /* 固定高度 */
}
```

然后隐藏其他几个子模块:

```
#pageHeader h1, #pageHeader h2 { /* 第1子模块的网页1、2级标题 */
    display: none; /* 隐藏显示 */
}
#quickSummary p.p1 { /* 第2子模块的第1段文本 */
    display: none; /* 隐藏显示 */
}
```

再把第2子模块的第2段超链接文本定位到网页的左上角顶部(如图8.42所示)。



图 8.42 设计展室封面的效果

```
#quickSummary p.p2 {
    font-size: 11px; /* 字体大小 */
    color: #ccc; /* 字体颜色 */
}
```



```

position: absolute;                /* 绝对定位 */
top: -1px;                        /* 顶部距离, 隐藏1像素 */
left: 2px;                        /* 左侧距离 */
}

```

第4步, 设计第1大模块的第3子模块以及第2大模块的布局。在这些模块中, 完全采用静态定位的方法, 即让模块内对象按着自然流动的形式从上到下排列显示。通过 width 和 height 来固定模块的显示大小, 使用 margin 属性调整每个子模块的显示位置, 通过 padding 属性调整模块内包含文本的显示位置。

例如, 设计第1大模块的第3子模块, 可以把标题文本隐藏起来, 利用背景图像的方式设计展板效果 (如图 8.43 所示)。

```

#preamble h3 {
    background: url(preamble.jpg) no-repeat;    /* 设计展板背景图像 */
    padding: 0px;                                /* 内边距 */
    margin: 0px;                                  /* 外边距 */
    width: 560px;                                /* 宽度 */
    height: 147px;                               /* 高度 */
}
#preamble h3 span {
    display: none;                               /* 隐藏显示标题文本 */
}

```

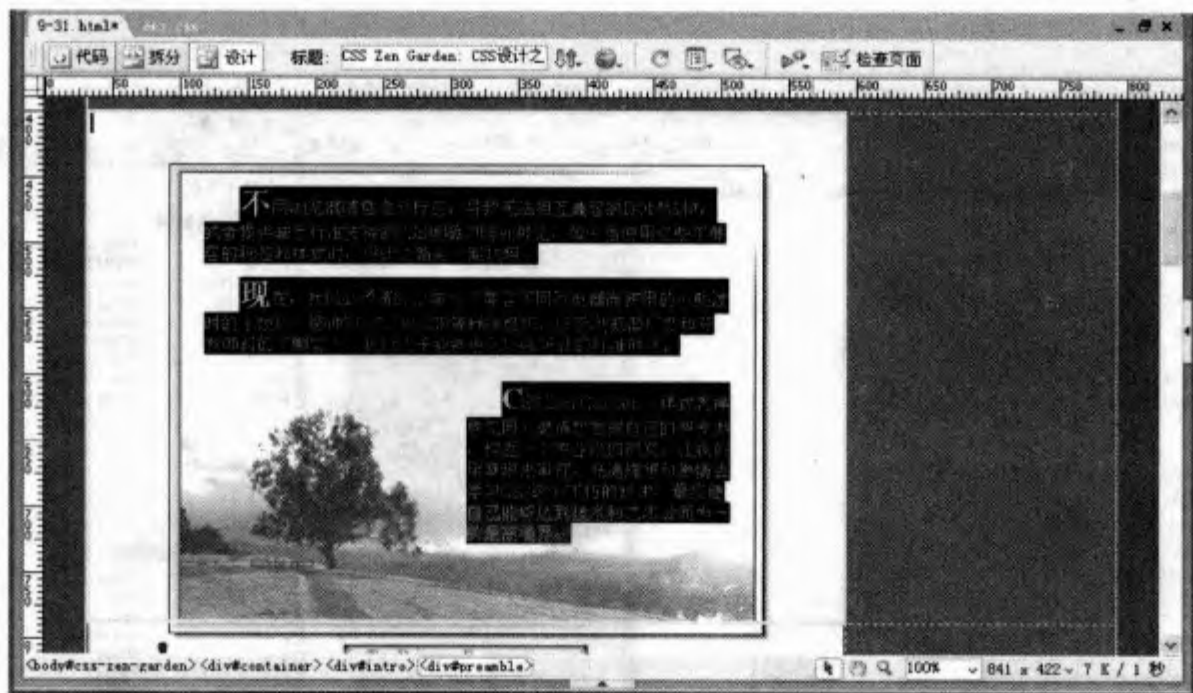


图 8.43 设计展板效果

然后利用内边距来调整文本段在展板内显示的位置和区域大小 (如图 8.43 所示)。

```

#preamble p { /* 段落文本缩进 */
    text-indent: 2em;                /* 缩进2个字符 */
}
#preamble p:first-letter { /* 段落首字样式 */
    font-size: 180%;                /* 放大字体 */
    font-weight: bold;               /* 加粗 */
    color: #444444;                  /* 字体颜色 */
}
#preamble p.p1 { /* 第一段文本 */
    padding: 10px 85px 10px 86px;    /* 调整显示区域 */
    margin: -100px 0px 0px 0px;       /* 调整显示位置 */
}

```

```

#preamble p.p2 { /* 第二段文本 */
    padding: 0px 85px 20px 86px;          /*调整显示区域*/
    margin: 0px;                          /*调整显示位置*/
}
#preamble p.p3 { /* 第三段文本 */
    background: url(preamble_img.jpg) no-repeat bottom; /* 增加展板底部背景 */
    padding: 0px 85px 60px 280px;         /* 调整显示区域 */
    margin: 0px;                          /* 调整显示位置 */
}

```

第2大模块的布局也遵循上一步的设计思路，具体就不再重复。

第5步，定位第3大模块的显示位置。按着正常的文档流顺序，第3大模块应该位于页面的最底下，为了能够使其在网页顶部右侧栏目中显示，使用绝对定位是一种最佳选择。由于上面已经把网页包含框<div id="container">定义为包含块，因此当我们定义<div id="linkList">模块为绝对定位时，它就以<div id="container">为参照物来进行定位（如图8.44所示）。

```

#linkList { /* 定位第三模块 */
    position: absolute;                  /* 绝对定位 */
    top: 400px;                        /* 距离顶部距离 */
    left: 570px;                       /* 距离左侧距离 */
    padding: 0px;                      /* 清除内边距 */
    margin: 0px;                      /* 清除外边距 */
    width: 190px;                      /* 固定宽度 */
}

```

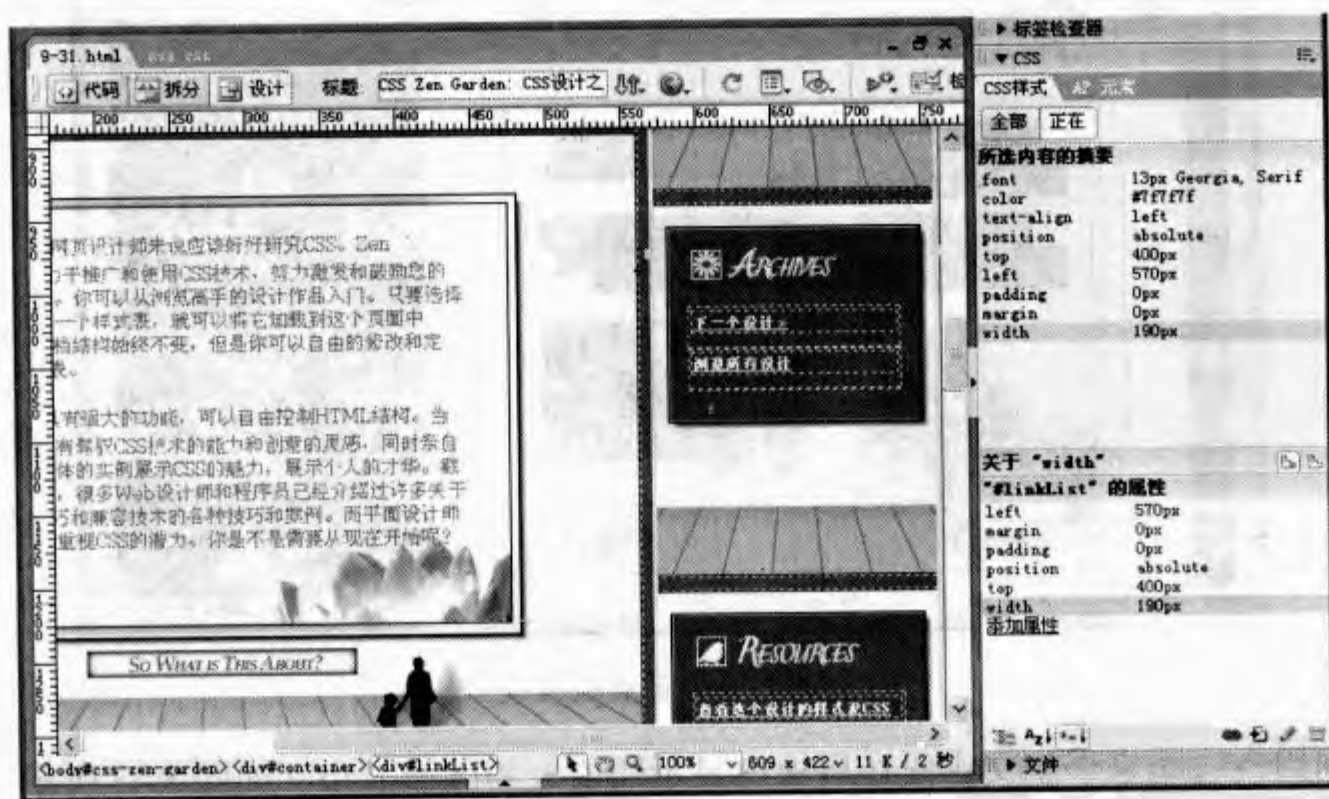


图 8.44 定位第三模块的显示位置和大小

第3模块内部包含的3个子模块将按着默认的静态定位方式自然流动在该绝对定位的层中，具体设计方法也是利用外边距、内边距和背景图像来调整显示位置和大小，详细代码可以参阅光盘示例代码，这里就不再详细讲解。

8.4.3 蓝色的多瑙河

上面两个示例分别演示了如何设计纯CSS定位布局，以及在CSS流动布局中辅助配合绝对定位两种应用类型。本节再讲解一个CSS浮动布局与定位布局相互配合的案例，相信通过本示例能够使读者进一步体会到网页设计的多种方案。在实际设计中采用浮动布局的案例比较

多，因为它具备更大的灵活性和适应能力。不过如果能够结合定位布局，你会发现网页设计会更加自如和轻松。

本案例是禅意花园的早期作品（<http://www.csszengarden.com/?cssfile=003/003.css>），它是典型的 3 行 2 列式布局，以浮动布局为主，兼用 CSS 定位控制作品链接栏目显示在页面顶部（如图 8.45 所示）。



图 8.45 蓝色多瑙河的页面设计效果

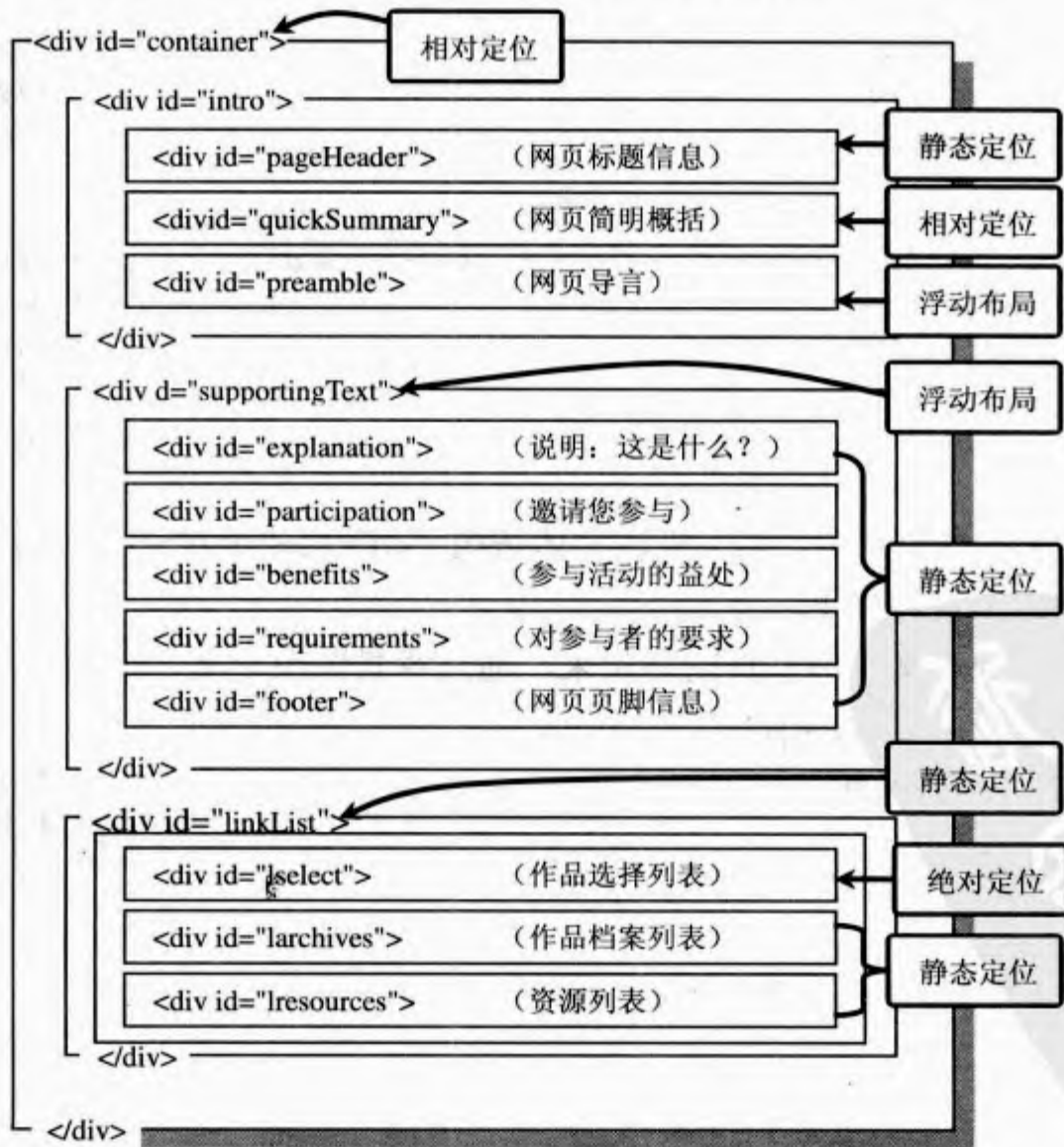


图 8.46 蓝色多瑙河网页布局的结构布局思路

整个页面主要模块布局设计示意图如图 8.47 所示。页面主要文本信息以浮动的方式分列 2 栏, 页面顶部显示为第 1 模块的主要信息, 同时通过绝对定位的方式把第 3 模块的作品链接(<div id="lselect">) 子模块定位到顶部显示。

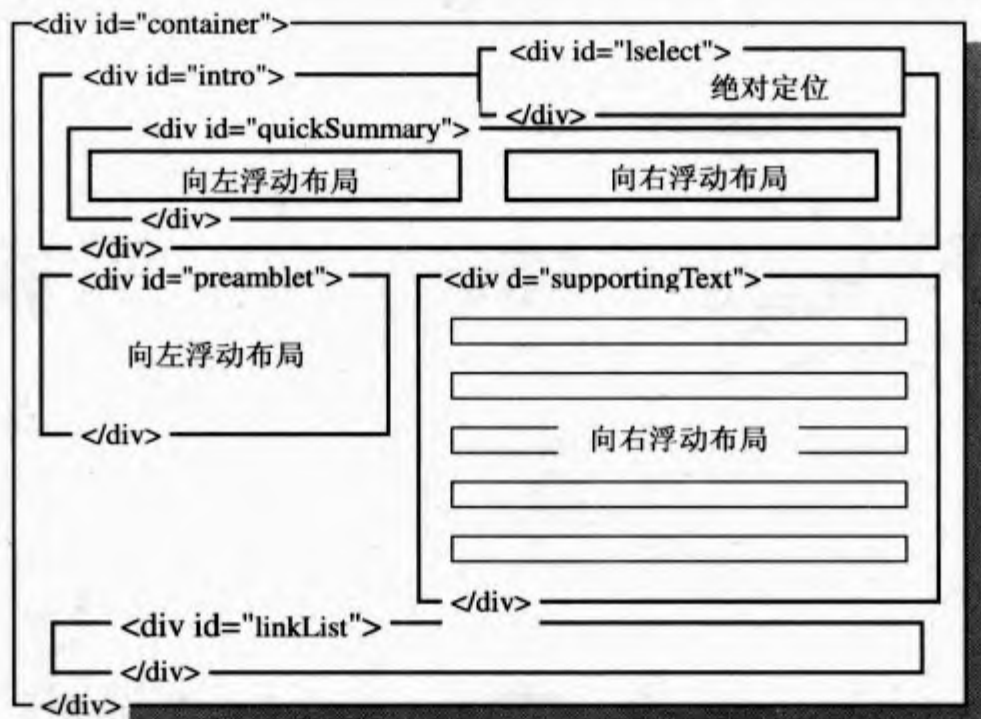


图 8.47 蓝色多瑙河网页布局的设计思路示意图

整个页面主体框架布局设计步骤如下。

第 1 步, 设计页面基本属性和网页主体框架:

```
body {
    text-align: center;                                /* 网页居中 */
    background: #748A9B url(bg2.gif) 0 0 repeat-y;    /* 网页背景 */
    margin: 0px;                                        /* 清除页边距 */
}
#container {
    background: #849AA9 url(bg1.gif) top left repeat-y; /* 网页背景 */
    text-align: left;                                    /* 文本左对齐 */
    width: 750px;                                        /* 固定宽度 */
    margin: 0px auto;                                    /* 网页居中 */
    position: relative;                                  /* 定义包含块 */
}
```

第 2 步, 设计第 1 模块布局。第 1 模块主体结构 (<div id="intro">) 以默认的方式显示。它包含的 3 个子模块分别设计如下。

第 1 子模块的标题和文本信息被隐藏起来, 通过背景图像定义一个大的图片效果。

```
#pageHeader h1 { /* 1 级标题样式 */
    background: transparent url(h1.jpg) no-repeat top left; /* 定义背景图像 */
    width: 750px;                                            /* 固定宽度 */
    height: 152px;                                           /* 固定高度 */
    margin: 0px;                                             /* 清除外边距 */
}
#pageHeader h1 span { /* 隐藏 1 级标题内容 */
    display: none;
}
#pageHeader h2 span { /* 隐藏 2 级标题 */
    display: none;
}
```


第2子模块定义为相对定位布局,然后通过坐标偏移来调整显示区域的位置,同时原位置保留不动,这样就避免了移动本栏目的位置,会影响到其他栏目的位置。

```
#quickSummary {
    width: 685px;                /* 固定宽度 */
    margin: 0px auto;            /* 居中对齐 */
    position: relative;          /* 相对定位 */
    top: -50px;                  /* 向上位移 50 像素 */
}
html>body #quickSummary { /* 兼容 FF 浏览器 */
    margin-top: -50px;          /* 边距取负, 向上移动 */
    top: 0;                    /* 相对偏移为 0 */
}
```

然后分别使用流动布局和浮动布局设计第2子模块包含的两个文本段。

```
#quickSummary .p1 { /* 第1段样式 */
    font-size: 1px;             /* 字体大小 */
    color: white;               /* 字体颜色 */
    background: transparent url(panel1-2.jpg) no-repeat top left; /* 背景图像 */
    width: 449px;               /* 宽度 */
    padding: 10px 0px 0px 5px;  /* 内边距 */
    float: left;               /* 向左浮动 */
    height: 268px;              /* 固定高度 */
    voice-family: "\"}\""; /* 兼容 IE 6 以下版本浏览器 */
    voice-family: inherit;
    height: 258px;              /* 固定高度 */
}
#quickSummary .p1 span { /* 隐藏文本 */
    display: none;
}
#quickSummary .p2 { /* 第2段样式 */
    color: #7593A7;             /* 固定高度 */
    background: transparent url(panel3.jpg) no-repeat 0 0; /* 背景图像 */
    padding: 90px 45px 0px 45px; /* 调整文本内边距 */
    float: right;               /* 向右浮动 */
    width: 214px;               /* 固定宽度 */
    height: 338px;              /* 固定高度 */
    voice-family: "\"}\""; /* 兼容 IE 6 以下版本浏览器 */
    voice-family: inherit;
    width: 124px;               /* 固定宽度 */
    height: 178px;              /* 固定高度 */
}
```

第3步,布局第1大模块的第3子模块。<div id="preamble">模块包含大量的文本,因此需要把它单独设计成一个模块,从父包含框<div id="intro">中脱离出来,与第2大模块并排为两列式浮动布局(如图8.48所示)。

```
#preamble {
    padding: 0px 0px 70px 33px; /* 通过内边距调整文本的显示位置 */
    margin: 0px 0 20px 0px;     /* 通过外边距调整模块的显示位置 */
    width: 210px;                /* 固定宽度 */
    float: left;                 /* 向左浮动 */
    background: transparent url(tag.gif) 50% 100% no-repeat; /* 定义底部背景图像 */
}
```

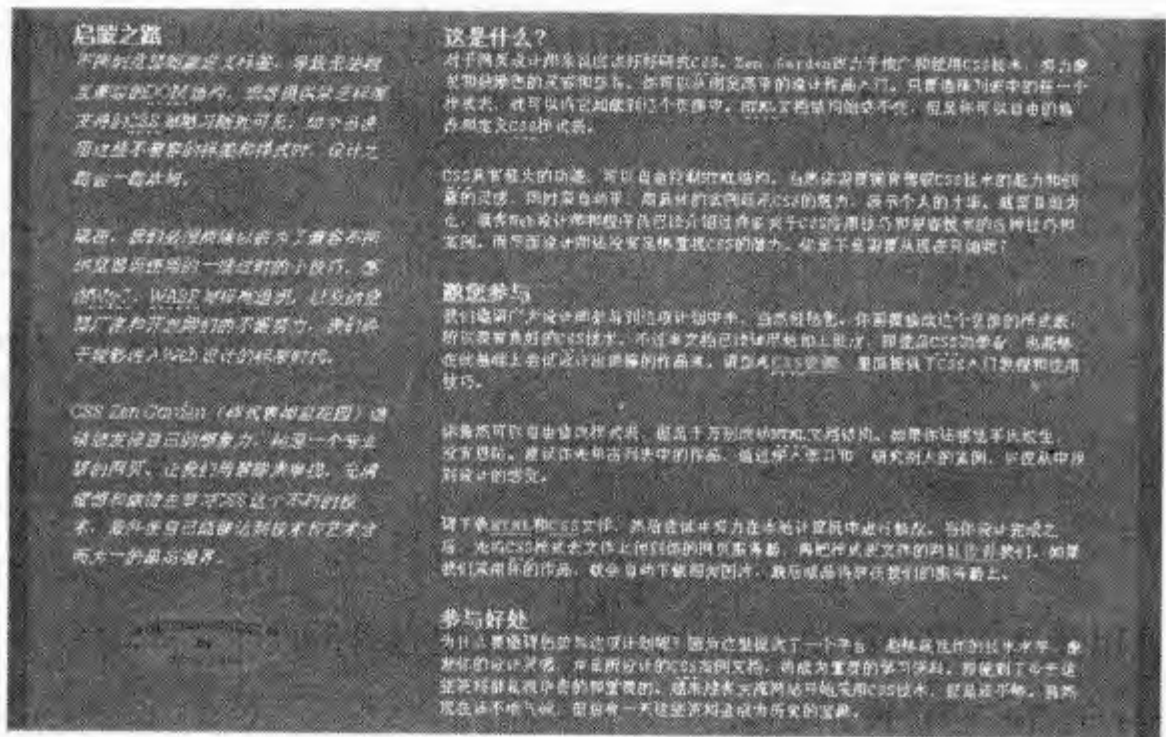


图 8.48 两列式浮动布局

第 4 步，第 2 大模块（<div id="supportingText">）与第 1 大模块的第 3 子模块并列在一起，虽然它们从属不同的结构层次，但是通过浮动能够让它们从原有的结构中脱离出来实现并列布局。

```
#supportingText {
    padding: 0px 40px 0px 0;
    float:right;
    width:430px;
}
```

/* 调整文本显示位置 */
/* 向右浮动 */
/* 固定宽度 */

至于第 2 大模块包含的 5 个子模块都遵循自然流动的方式进行布局，所以也就不再详细说明。当然在设计时如果父包含框浮动显示，则应该在最后一个子模块中增加 clear 属性，以强迫撑起浮动的包含框。

```
#footer {
    clear: both;
}
```

/* 清除浮动 */

第 5 步，设计第 3 模块布局。第 3 模块（<div id="linkList">）也以自然流动的方式进行布局，不过通过设置超大外边距，让人以为它是向右浮动布局（如图 8.49 所示），这也算是一个小技巧吧。

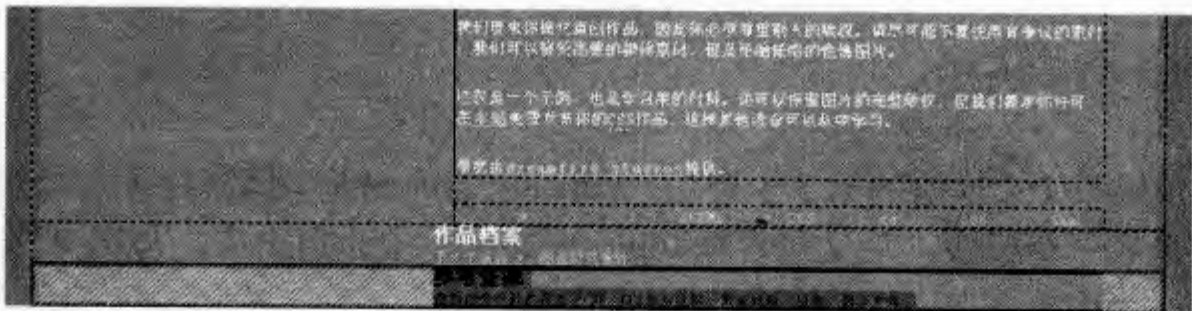


图 8.49 以外边距模拟浮动效果

不过作品链接子模块（<div id="lselect">）以绝对定位的方式被固定到页面的顶部（如图 8.50 所示），这时该子模块就脱离了原来的结构进行独立显示。

```
#lselect {
    position: absolute;
}
```

/* 绝对定位 */


```

top: 15px;
left: 0px;
padding-left: 350px;
margin: 0px auto;
width: 730px;
voice-family: '"\'}\'';
voice-family: inherit;
width: 380px;

```

```

/* y 轴坐标, 顶部距离 */
/* x 轴坐标, 左侧距离 */
/* 左内边距 */
/* 居中对齐 */
/* 固定宽度 */
/* 兼容 IE 6 以下版本浏览器 */
/* 固定宽度 */

```

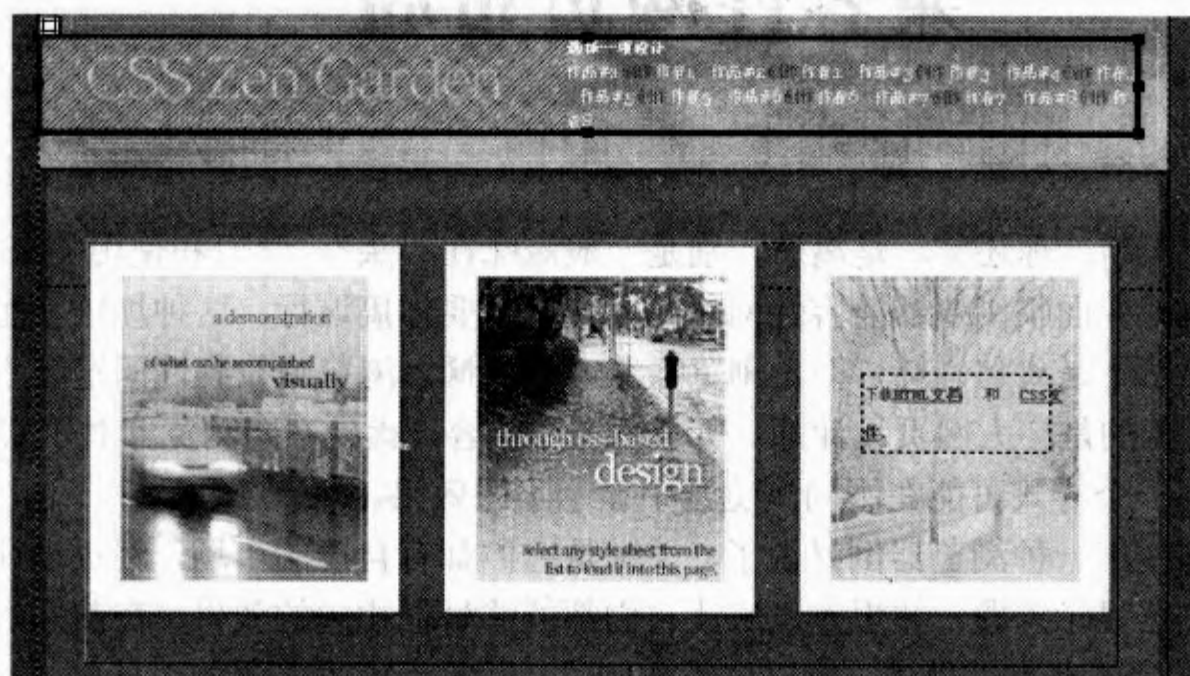


图 8.50 绝对定位作品链接子模块

第 9 章

兼容性网页布局

正所谓“剪不断，理还乱，是离愁，别是一般滋味在心头”，学习和使用 CSS 进行网页布局时，不可避免地要面临浏览器兼容性问题的困扰。只要你搞这行，这种烦恼和愁苦将会一直伴随你左右，当然不是离愁别绪，而是浏览器兼容性烦恼。浏览器兼容性问题源于浏览器厂家的标准不统一，苦的是广大网页设计师。所谓浏览器兼容，就是不同浏览器对于 CSS 标准的解释不同，导致同一个样式可能在不同浏览器中显示不同效果。

如果大家都使用一种浏览器倒好说了，关键是它们都有自己的市场，网页设计师就必须考虑多家主流浏览器显示问题，并想方设法让它们都能够呈现相同的效果。在前面的章节中我们也在不断地渗透 CSS 兼容性处理，所以先不举例了。针对当前浏览器的市场份额，特别是国内用户的使用习惯，只要设计师所设计的网页能够兼容 IE 6、IE 7 以及 FF 浏览器就可以了，其他浏览器或版本可以不去考虑。本章将结合这 3 种浏览器进行讲解，其他浏览器的兼容性就不再涉猎。

9.1 智能的 IE 条件语句

IE 浏览器一直都很聪明，它能够支持各种奇特的功能，甚至还能够卖弄各种炫目的特效，以及滤镜。但是聪明反被聪明误，正是 IE 的小聪明迫使很多网页设计师付出了沉重的代价。

当然 IE 所提供的各种智能功能在某些方面还是比较管用的。比如 IE 条件语句，它能够模拟计算机语言，进行简单的条件判断，以方便用户智能控制样式表的应用。可惜这套 IE 条件语句只能被 IE 浏览器所支持，其他浏览器视其为垃圾（注释语句），将其全部忽略了。当然 IE 条件语句给网页设计师带来了福音，利用它可以开发出兼容 IE 浏览器的样式。下面我们就来详细讲解 IE 的条件语句。

9.1.1 认识 IE 条件语句

由于 IE 浏览器目前占据着绝对的市场份额，所以当你设计网页时就不得不顾及 IE 浏览器的情绪（解析效果）。但是就算我们不用去考虑其他浏览器，仅就 IE 家族的众多版本，都是让人头疼的问题。那么解决浏览器兼容问题最好的方法是什么呢？

答案是唯一的，即使用 IE 条件语句。如果随意留意一下标准网站的源代码，你会发现诸如下面的注释语句：

```
<!--[if IE 5]>  
<style type="text/css">
```



```
/* 将 IE 5* 的 css 盒模型宽高计算修正放在这个条件注释中 */
#left { width: 180px; }
#right { width: 190px; }
</style>
<![endif]-->
```

或者

```
<link rel="stylesheet" rev="stylesheet" href="css/index.css" type="text/css"
media="screen" />
<!--[if IE 7]>
<link rel="stylesheet" rev="stylesheet" href="css/ie7.css" type="text/css"
media="screen" />
<![endif]-->
<!--[if IE 6]>
<link rel="stylesheet" rev="stylesheet" href="css/ie6.css" type="text/css"
media="screen" />
<![endif]-->
<!--[if lt IE 6]>
<link rel="stylesheet" rev="stylesheet" href="css/ie6lt.css" type="text/css"
media="screen" />
<![endif]-->
```

你可能知道“<!--”与“-->”标识符配对使用是用来表示 HTML 注释语句的。是的，但是这些特殊的注释语句对于 IE 浏览器来说可就不是什么难题了。换句话说，IE 会把它们当做一个简单的逻辑语句进行解析。

例如，针对上面第一段注释，它表示如果当前浏览器是 IE 浏览器，则解析“<!--[if IE]>”和“<![endif]-->”标识符之间的样式表。而对于其他浏览器则视而不见，将其作为注释语句全部都忽略了。因此你可以在这里放置一些能够兼容 IE 浏览器的样式代码。而对于 CSS 来说，“/*”和“*/”标识符是一对注释语句，与 HTML 语言的“<!--”与“-->”标识符作用是一样的。

```
<style type="text/css">
/* 请将所有版本的 IE 的 CSS 修复放在这个条件注释中 */
.twoColFixLtHdr #sidebar1 { padding-top: 30px; }
.twoColFixLtHdr #mainContent { zoom: 1; }
/* 上面的专用 zoom 属性为 IE 提供避免错误所需的 hasLayout */
</style>
```

这里有一个术语需要先介绍一下，我们常说“兼容××浏览器”，它表示该浏览器存在一些 Bug（错误）或者不支持的功能。例如，IE 5.5 及以下版本浏览器在进行盒模型解析时会错误地认为元素的宽度包含边框和内边距，这就是 IE 5.5 及以下版本浏览器存在的一个 Bug。所谓不支持的功能，例如，IE 6 及其以下版本浏览器不支持 max-height 属性、不支持属性选择符等。

对于浏览器存在的 Bug，我们可以使用 Hack 来进行修补，即所谓的浏览器兼容性处理，通俗说就是利用各种小技巧为这些浏览器打个补丁。打补丁的方法正是本章要研究的重点，它通过利用各种浏览器特殊的解析规则，专门设计只能够被某种浏览器识别的代码来实现。例如，上面介绍的 IE 条件语句。

再看一下上面示例中举的第二段注释语句，它分别使用不同的条件语句来设置在不同版本的 IE 浏览器中链接不同的样式表。那么 IE 条件语句如何使用，使用时又要注意什么问题呢？对于这些问题请继续阅读下一节内容。

9.1.2 IE 条件语句基本用法

既然 IE 条件语句在浏览器兼容性处理中功能非常强大，实用价值很高，下面我们就来详

细讲解 IE 条件语句的基本用法。

IE 条件语句一般放在 HTML 注释语句之中,这样就可以避免其他浏览器因为无法解析这些条件语句时可能会出现的尴尬。其基本语法如下:

```
<!--[if IE]>
    IE 下可执行语句
<![endif]-->
```

条件语句放在中括号内,然后嵌入到 HTML 注释中。但是你要注意,起始条件标记中省略了 HTML 注释语句的后半部分标记(-->),而结束标记中省略了 HTML 注释语句的前半部分标记(<!--),仅是一个半封闭的形式。这样对于其他浏览器来说,前后两个半封闭的 HTML 注释标记就形成了一个完整的注释标记,从而避免了其他浏览器无法解析而带来的尴尬。

IE 浏览器在解析 HTML 源代码时,如果遇到类似“<!--[if IE]>”或“<![endif]-->”标记时,会立即停下来,仔细分析其中包含的源代码。

在 IE 条件中可以设置一些简单的条件语句,你可以设置只能够在某种版本浏览器中才能够执行所包含的源代码。例如,输入下面的源代码,然后分别在不同版本的 IE 浏览器中预览(如图 9.1~图 9.4 所示)。

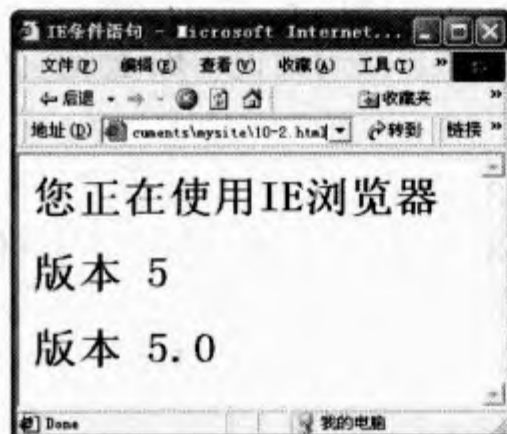


图 9.1 IE 5 中显示效果

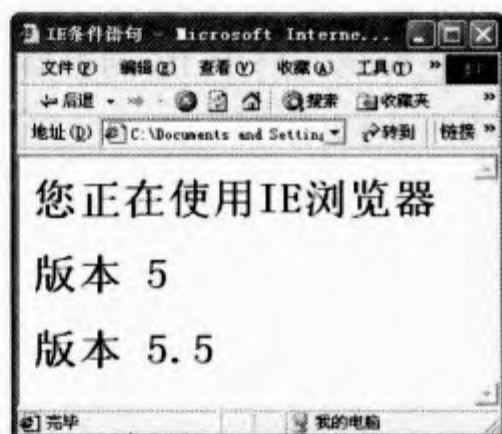


图 9.2 IE 5.5 中显示效果

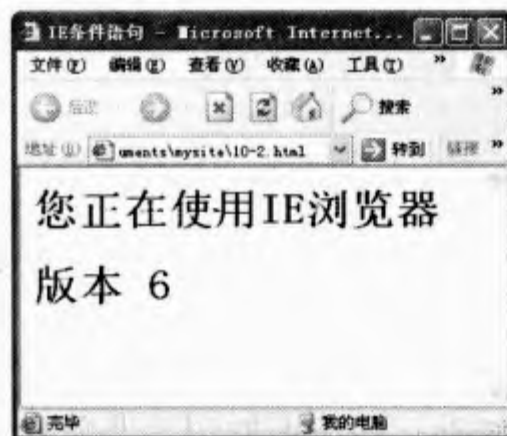


图 9.3 IE 6 中显示效果

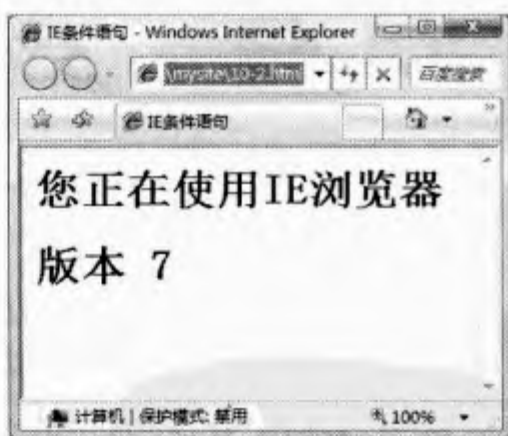


图 9.4 IE 7 中显示效果

```
<!--[if IE]>
<h1>您正在使用 IE 浏览器</h1>
<![endif]-->
<!--[if IE 5]>
<h1>版本 5</h1>
<![endif]-->
<!--[if IE 5.0]>
<h1>版本 5.0</h1>
<![endif]-->
<!--[if IE 5.5000]>
```



```
<h1>版本 5.5</h1>
<![endif]-->
<!--[if IE 6]>
<h1>版本 6</h1>
<![endif]-->
<!--[if IE 7]>
<h1>版本 7</h1>
<![endif]-->
```

在使用上面的 IE 条件语句时请注意, <!--[if IE 5]>条件语句可以表示 IE 5 或 IE 5.0 版本, 版本虽然相同, 但名字略有区别。在表示 IE 5.5 版本时应该使用<!--[if IE 5.5000]>条件语句, 使用<!--[if IE 5.5]>是无效的。

除了使用这些指定某种版本浏览器的条件语句之外, 你还可以结合 lte、lt、gte、gt 和 ! 属性定义 IE 浏览器的版本范围, 主要属性及关键字说明如下。

- lte: 小于或等于某个版本的 IE 浏览器。
- lt: 小于某个版本的 IE 浏览器。
- gte: 大于或等于某个版本的 IE 浏览器。
- gt: 大于某个版本的 IE 浏览器。
- !: 不等于某个版本的 IE 浏览器。

例如, 输入下面代码, 然后分别在不同版本的 IE 浏览器中预览 (如图 9.5、图 9.6、图 9.7、图 9.8 所示)。

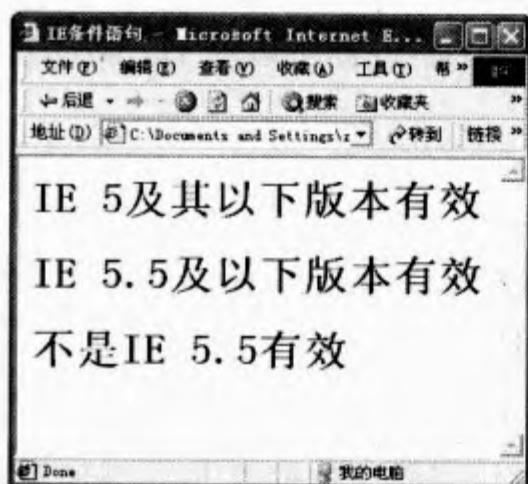


图 9.5 IE 5 中显示效果

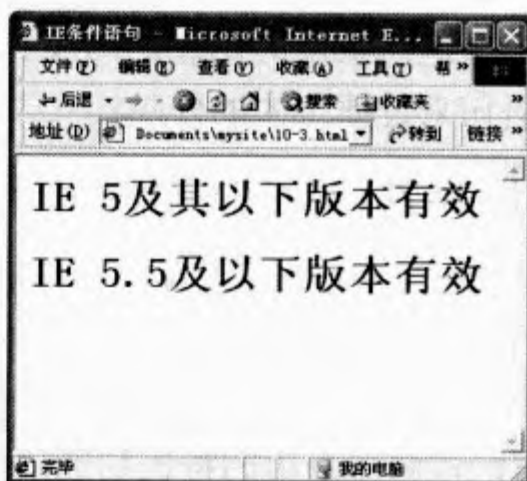


图 9.6 IE 5.5 中显示效果

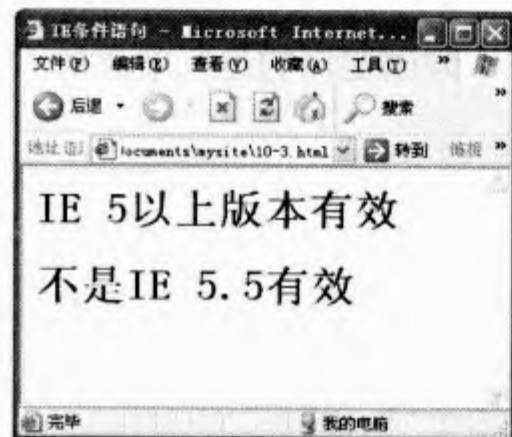


图 9.7 IE 6 中显示效果

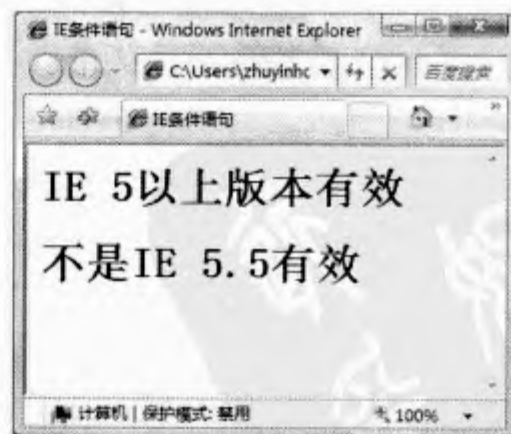


图 9.8 IE 7 中显示效果

```
<!--[if gt IE 5]>
<h1>IE 5 以上版本有效</h1>
<![endif]-->
<!--[if lte IE 5]>
<h1>IE 5 及其以下版本有效</h1>
```

```

<![endif]-->
<!--[if lte IE 5.5000]>
<h1>IE 5.5 及以下版本有效</h1>
<![endif]-->
<!--[if !IE 5.5000]>
<h1>不是 IE 5.5 有效</h1>
<![endif]-->

```

请注意,对于<!--[if gt IE 5]>条件语句来说,是指 IE 6 及其以上版本,而不包括 IE 5.5 版本。但是如果修改<!--[if gt IE 5]>条件语句<!--[if gt IE 5.0]>条件语句,则在 IE 5.5 版本浏览器中会显示图 9.9 所示的效果。

这些示例具体展示了 IE 条件语句的应用技巧,当然在使用时,你还要注意下面两个问题。

- 条件语句的基本结构与 HTML 的注释语句(<!-- -->)是一样的。因此 IE 以外的浏览器将会把它们看做是普通的注释而完全忽略它们。而 IE 浏览器将根据设置的条件来判断是否解析,以及如何解析页面内容,并同时解析条件语句包含的内容。
- 条件语句使用的是 HTML 的注释结构,因此它们只能使用在 HTML 文件里,而不能在 CSS 文件中使用。因此你不能把所有兼容 IE 浏览器的特殊样式都放在外部样式表文件中,这些条件语句在 CSS 文件中是不能够被解析的。不过你可以在 HTML 中使用条件语句来过滤不同的外部样式表文件。



图 9.9 IE 5.5 中显示效果

9.1.3 IE 条件语句在实践中应用

由于 IE 条件语句不依赖于浏览器自身存在的 Bug 或不支持的功能来实现兼容,所以也是最安全的浏览器兼容技术,一般用它来兼容不同版本的 IE 浏览器。

除此之外,IE 条件语句还可以做一些超出 CSS Hack (兼容技巧) 范围的事情。因为 IE 浏览器拥有经过深思熟虑的特色功能,所以你可以放心使用。当然,在进行兼容浏览器布局时,你应该尝试寻找在 IE 浏览器上真正的 CSS 解决方法,而不是遇到什么问题都使用 IE 条件语句;如果找不到,则再大胆地使用条件语句。

例如,下面这个多级下拉菜单就是使用 IE 条件来兼容不同浏览器的运行效果的(如图 9.10 所示)。

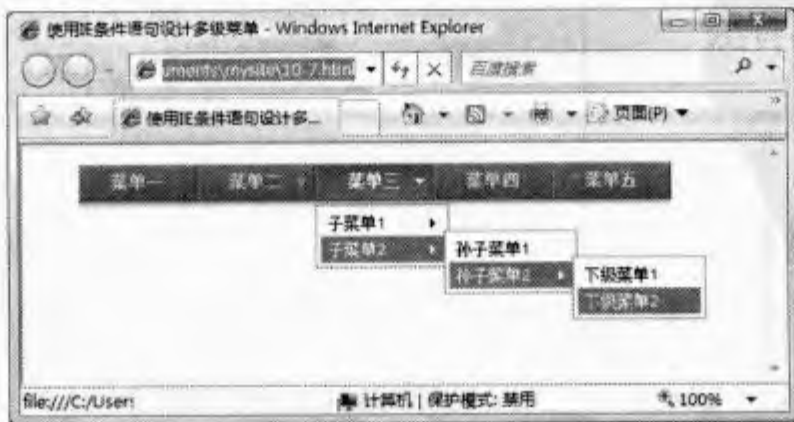


图 9.10 使用 IE 条件语句设计的多级下拉菜单效果

这里没有把 IE 条件语句作用于样式表,而是作用于 HTML 结构。该多级下拉菜单的 HTML

结构如下:

```
<ul class="menu2">
  <li class="top"><a href="#" id="home" class="top_link"><span>菜单一</span></a></li>
  <li class="top"><a href="#" id="shop" class="top_link"><span class="down">菜单二
  </span>
    <!--[if gte IE 7]><!--></a><!--<![endif]-->
    <!--[if lte IE 6]><table><tr><td><!--[endif]-->
      <ul class="sub">
        <li><a href="#">子菜单 1</a></li>
        <li><a href="#">子菜单 2</a></li>
      </ul>
    <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
  </li>
  <li class="top"><a href="#" id="products" class="top_link"><span class="down">
  菜单三</span>
    <!--[if gte IE 7]><!--></a><!--<![endif]-->
    <!--[if lte IE 6]><table><tr><td><!--[endif]-->
      <ul class="sub">
        <li><a href="#" class="fly">子菜单 1
          <!--[if gte IE 7]><!--></a><!--<![endif]-->
          <!--[if lte IE 6]><table><tr><td><!--[endif]-->
            <ul>
              <li><a href="#">子菜单 1</a></li>
              <li><a href="#">子菜单 2</a></li>
            </ul>
          <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
        </li>
        <li class="mid"><a href="#" class="fly">子菜单 2
          <!--[if gte IE 7]><!--></a><!--<![endif]-->
          <!--[if lte IE 6]><table><tr><td><!--[endif]-->
            <ul>
              <li><a href="#">孙子菜单 1</a></li>
              <li><a href="#" class="fly">孙子菜单 2
                <!--[if gte IE 7]><!--></a><!--<![endif]-->
                <!--[if lte IE 6]><table><tr><td><!--[endif]-->
                  <ul>
                    <li><a href="#">下级菜单 1</a></li>
                    <li><a href="#">下级菜单 2</a></li>
                  </ul>
                <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
              </li>
            </ul>
          <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
        </li>
      </ul>
    <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
  <li class="top"><a href="#" class="top_link"><span>菜单四</span></a></li>
  <li class="top"><a href="#" class="top_link"><span>菜单五</span></a></li>
</li>
</ul>
```

在上面结构中,使用了多层嵌套的方法来设计多级菜单,通过在a元素中包含子结构的方法来设计鼠标移过时显示下拉子菜单的动态效果,这在第5章曾经讲解过。下面我们来分析一下IE条件语句是如何在HTML结构中起作用的。

通过在a元素中包含子菜单来设计动态下拉菜单效果,这种设计方法在标准浏览器中能够

很好地被解析，并能够正确地显示，但是在 IE 浏览器中却存在很大的差异。例如，如果我们在上面示例的基础上做一个简化版结构（样式表就不再详细讲解）：

```
<ul class="menu2">
  <li class="top"><a href="#" id="products" class="top_link"><span class="down">
    主菜单</span></a>
    <ul class="sub">
      <li class="mid"><a href="#" class="fly">子菜单</a>
        <ul>
          <li><a href="#" class="fly">孙子菜单</a>
            <ul>
              <li><a href="#">下级菜单</a></li>
            </ul>
          </li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

上面这个结构共嵌套了 4 层项目列表，每一层子列表项都被包含在当前项目的 a 元素中。对于这样的结构，如果在 FF 或者 IE 7 中预览会显示正常（如图 9.11 所示），但是在 IE 6 版本及其以下版本浏览器中则无法正确显示（如图 9.12 所示）。



图 9.11 FF 中显示效果

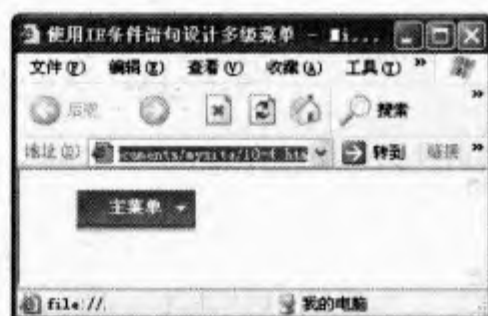


图 9.12 IE 6 中无法显示效果

此时我们不妨使用 IE 条件语句来为 IE 6 及其以下版本浏览器增加表格嵌套结构，因为 IE 6 及其以下版本浏览器能够很好的解析这种结构，所以不妨这样来设计：

```
<ul class="menu2">
  <li class="top"><a href="#" id="products" class="top_link"><span class="down">
    主菜单</span>
    <!--[if lte IE 6]><table><tr><td><!--[endif]>-->
    <ul class="sub">
      <li class="mid"><a href="#" class="fly">子菜单
        <!--[if lte IE 6]><table><tr><td><!--[endif]>-->
        <ul>
          <li><a href="#" class="fly">孙子菜单
            <!--[if lte IE 6]><table><tr><td><!--[endif]>-->
            <ul>
              <li><a href="#">下级菜单</li>
            </ul>
            <!--[if lte IE 6]></td></tr></table></a><!--[endif]>-->
          </li>
        </ul>
        <!--[if lte IE 6]></td></tr></table></a><!--[endif]>-->
      </li>
    </ul>
  </li>
</ul>
```



```

<!--[if lte IE 6]></td></tr></table></a><![endif]-->
</li>
</ul>

```

为每个 a 元素内部包含一个表格，由于表格在布局方面的稳定性，因此它能够很好地组织起菜单的层叠结构，此时在 IE 6 中显示如图 9.13 所示；但是此时在 IE 7 或者在其他标准浏览器中显示又出了问题，如图 9.14 所示，在 IE 7 中结构显示得非常混乱，看来这个问题还是比较复杂。

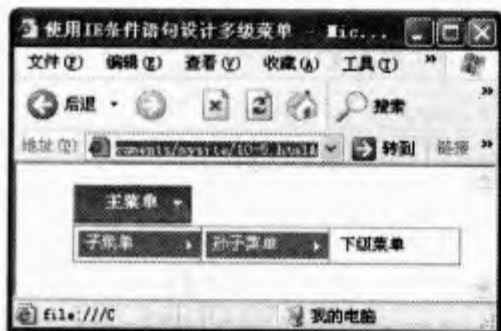


图 9.13 IE 6 中无法显示效果

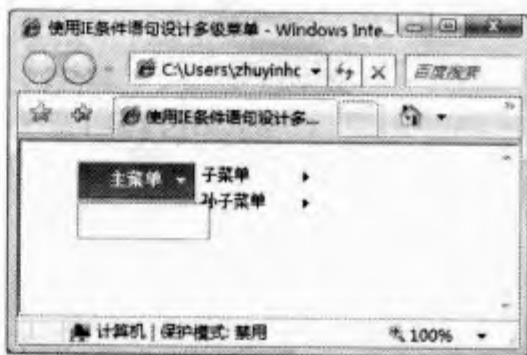


图 9.14 IE 7 中无法显示效果

分析出错的原因：这是因为结束标记在 IE 6 及其以下版本中被放置在结构的末尾，如下所示：

```

<li>
  <a href="#" class="fly">孙子菜单
    <table><tr><td>
      <ul>
        <li><a href="#">下级菜单</a></li>
      </ul>
    </td></tr></table>
  </a>
</li>

```

而 IE 7 或其他标准浏览器仅认可结束标记放在前面，如下所示：

```

<li>
  <a href="#" class="fly">孙子菜单</a>
    <ul>
      <li><a href="#">下级菜单</a></li>
    </ul>
</li>

```

所以，我们还必须寻找另一种方法，让能够在 IE 7 以及标准浏览器中仅显示在前面。此时你可以继续使用 IE 条件语句，并设置它们仅在 IE 7 及其以上版本中有效，如下所示：

```

<li><a href="#" class="fly">孙子菜单
  <!--[if gte IE 7]></a><![endif]-->
  <!--[if lte IE 6]><table><tr><td><![endif]-->
  <ul>
    <li><a href="#">下级菜单</a></li>
  </ul>
  <!--[if lte IE 6]></td></tr></table></a><![endif]-->
</li>

```

不过，这里还需要使用一个欺骗技巧，在<!--[if gte IE 7]><![endif]-->条件语句中再嵌套一个注释语句，由于双层嵌套的注释结果导致其中所包含的标签会被其他浏览器所识别，但是对于 IE 6 及其以下版本浏览器来说，由于一看到<!--[if gte IE 7]>和<![endif]-->这样一对条

件结构，就立即忽略了其中包含的所有内容。因此，通过这种欺骗手段可以解决某些条件语句在 IE 6 与 IE 7 和其他标准浏览器中共存的矛盾。最后，把该菜单下所有子菜单重新修改如下，其他菜单也可以参照这种方法来进行设计。

```
<ul class="menu2">
  <li class="top"><a href="#" id="products" class="top_link"><span class="down">
    主菜单</span>
    <!--[if gte IE 7]><!--></a><!--<![endif]-->
    <!--[if lte IE 6]><table><tr><td><!--[endif]-->
      <ul class="sub">
        <li class="mid"><a href="#" class="fly">子菜单
          <!--[if gte IE 7]><!--></a><!--<![endif]-->
          <!--[if lte IE 6]><table><tr><td><!--[endif]-->
            <ul>
              <li><a href="#" class="fly">孙子菜单
                <!--[if gte IE 7]><!--></a><!--<![endif]-->
                <!--[if lte IE 6]><table><tr><td><!--[endif]-->
                  <ul>
                    <li><a href="#">下级菜单</a></li>
                  </ul>
                <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
              </li>
            </ul>
          <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
        </li>
      </ul>
    <!--[if lte IE 6]></td></tr></table></a><!--[endif]-->
  </li>
</ul>
```

9.2 探析 IE 浏览器渲染网页布局的特性

所谓兼容性网页布局，简而言之，就是如何使设计标准的网页能够在 IE 中正常显示，或者说如何迁就 IE 浏览器蹩脚的解析规则。其实设计师很多时候都是在 IE 与非 IE（即非标准与标准）之间来回折腾，直到精疲力竭为止。IE 浏览器出身于微软的贵族设计思维，具有典型的专横和不合作性。虽然在多次与标准技术的交融中又包含了很多标准因素，但其骨子里的即有很强的个性却一直变化不大，因此对于广大初学者来说，学习标准网页布局的难度还是很大的。

9.2.1 认识 IE 浏览器的渲染特性——Layout

很多时候，IE 浏览器特别是 IE 6 及其以下版本浏览器在解析标准网页时都会出现很多莫名其妙的错误，这些奇怪的渲染现象都是因为 Layout 特性造成的。那么什么是 Layout 特性呢？

Layout 可直译为“布局、设计”，它是 IE 浏览器所特有的一个性质，IE 认为 Layout 是网页布局的基础。IE 浏览器的 Layout 特性决定了一个元素如何进行显示，如何控制其包含的子元素和内部对象，如何与其他元素交互和建立联系，以及如何响应和传递脚本中的事件等。

IE 浏览器在解析页面时一直固执地认为：网页模型都是由很多个互不相关的小方块（拥有 Layout 特性的元素）堆砌而成。

但是符合标准的浏览器一般都会遵循 W3C 的网页模型，它们认为：网页模型不应是由各自独立的元素堆砌而成，而是由叙述完备、故事性很强的相关信息块构成。这是两个截然不同

的思想世界，犹如东西方文化的冲突一样有趣，而又让设计师烦恼。

Layout 特性犹如一个小窗体，拥有 Layout 特性的元素内部内容是完全独立的，而且也无法影响其边界外的任何内容。这种独立性使拥有 Layout 特性的元素在布局时通常比较稳定，而且它们可以让某些 IE 浏览器存在的 Bug 消失。当然这种思想和方法是非标准的，但是只有你先明白了这些特性，才能够明白为什么 IE 浏览器存在那么多 Bug，以及如何来解决这些 Bug。

例如，如果一个元素拥有 Layout 特性，则它包含的内容将由该元素的边界矩形框来决定。拥有 Layout 特性的意思基本上就是表示该元素是一个矩形。从内部来说，拥有 Layout 特性的元素可以负责绘制其内部包含的内容。

一般来说，在 IE 的 DHTML 引擎中，元素是不负责自己的位置的。虽然每个元素在源代码中都有一个位置，在文档流也有一个位置，但是它们的内容却是由靠它们最近的一个 Layout 特性的父级元素来控制。这些元素依赖父级元素的 Layout 特性来处理诸如决定大小尺寸和测量信息等诸多繁重的工作。

9.2.2 操控 IE 元素的 Layout 特性

既然说 Layout 特性在解析 IE 网页中占据了重要地位和核心功能，那么这种特性是不是可以被人为控制呢？是的，你可以通过 CSS 的某些属性来动态控制元素是否拥有 Layout 特性，当然有些元素默认状态就拥有 Layout 特性。你可以通过 `hasLayout` 属性来判断一个元素是否拥有 Layout 特性，`hasLayout` 是 IE 的一个专有属性，它表示元素是否具备布局的功能，其他标准浏览器不支持该属性。

当我们说一个元素具备 Layout 特性，或者说一个元素拥有网页布局功能的时候，也就是指它的 `hasLayout` 属性显示为 `true`。

例如，对于 `div` 元素来说，在默认状态下它不具备 Layout 特性，即不具备布局环绕功能，但是如果我们给它定义高度或者宽度，这时它就拥有了 Layout 特性。因此我们说某个元素无 Layout 特性时，并不是说它不具备 Layout 特性，而是它的 `hasLayout` 属性未被开启，你可以通过各种方法触发 `hasLayout = true`。当然你不能够直接在 CSS 中来设置 `hasLayout` 的属性值。如果要清楚某个元素的 Layout 特性，你也不能够直接来设置 `hasLayout = false` 来实现，此时删除触发 `hasLayout = true` 的 CSS 声明即可，如声明 `div` 元素的高度或宽度。

对于 IE 浏览器的这种奇怪的渲染现象，John Gallant 和 Holly Bergevin 称之为空间错误 (Dimensional Bugs)，也就是说元素没有空间概念，只有为其定义宽度或者高度，才能够解决这类布局错误。

这种空间错误表现在网页布局中就是 Layout 在解析盒模型时会出现许多莫名其妙或者难以预料的现象，从而影响到相邻元素，乃至整个网页的布局效果。例如，在前面章节中我们反复介绍的浮动元素 Bug，定义的属性与设想的效果差异很大，包含元素与被包含元素之间出现的外边距重叠现象，以及在设计项目列表、背景图像样式可能存在的各种错误。

9.2.3 网页元素的 Layout 特性

Layout 完全是 IE 浏览器专有的特性，它不同于标准的 CSS 属性，与其他浏览器的专有 CSS 属性也存在很大的区别，因为 Layout 是无法通过 CSS 属性来进行直接声明的。

元素是否拥有 Layout 特性不是由 Layout 属性来定义的，也不是由 `hasLayout` 属性来决定。在默认状态下，下面这些 HTML 元素具备 Layout 特性。

- 网页主体: html、body。
- 表格元素: table、tr、th、td。
- 表单元素: input、select、textarea、button。
- 多媒体元素: embed、object、applet、marquee。
- 其他元素: img、hr、iframe。

下面这些 CSS 属性或者取值可以使某个元素获取 Layout 特性。

- position:absolute: 绝对定位元素的包含块。
- float:left/right: 具备浮动模型的元素。
- display:inline-block: 行内块状显示的元素。

inline-block 是一种特殊的显示类型, 当一个行内元素需要拥有 Layout 特性时, 就可以通过该属性来实现。

- width: 被定义宽度的元素 (除 auto 外的任意值)。
- height: 被定义高度的元素 (除 auto 外的任意值)。

很多时候设计师会使用这两个属性来强迫元素拥有 Layout 特性, 以便用来修复某些 IE 浏览器存在的布局 Bug, 例如, 定义 height 属性为 1% 等。

如何检测一个元素是否拥有 Layout 特性? 不妨输入下面代码:

```
<div id="div1" style="width:100%">被定义了宽度的div元素</div>
<div id="div2">没有被定义定位或高宽的div元素</div>
<button onclick="alert('第1个元素的haslayout=' + div1.currentStyle.hasLayout)">第1个元素的haslayout</button>
<button onclick="alert('第2个元素的haslayout=' + div2.currentStyle.hasLayout)">第2个元素的haslayout</button>
```

然后在 IE 浏览器中预览, 当你单击对应的按钮可以看到该元素是否拥有 Layout 特性 (如图 9.15 所示)。

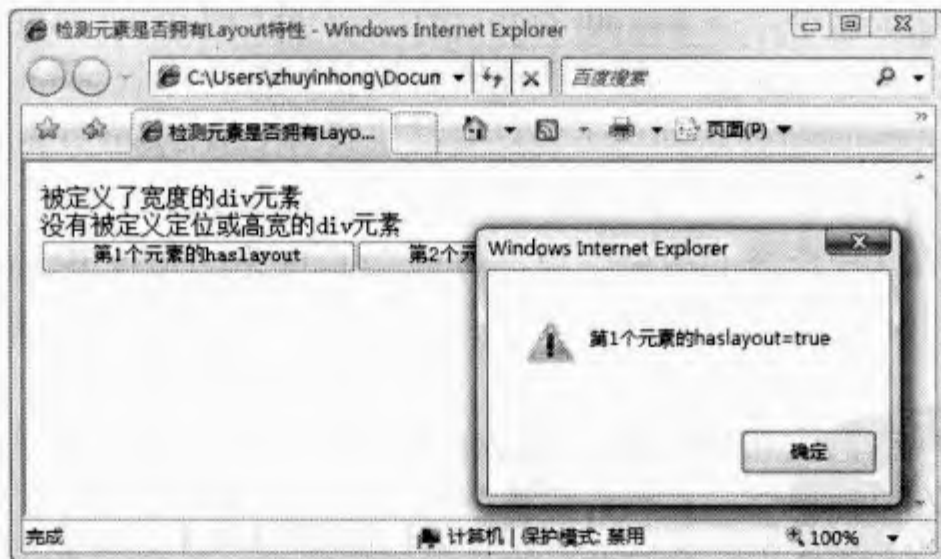


图 9.15 检测元素的 Layout 特性

- zoom: 被设置了对对象的放大比例的元素 (除 normal 外的任意值)。
- writing-mode:tb-rl: 设置对象的书写方向为从上到下的元素。

zoom: 和 writing-mode 都是 IE 浏览器的专有属性, 无法通过 W3C 验证。一般设计师通过设置 zoom: 为 1 来解决 IE 浏览器在解析中出现的各种错误。

在 IE 7 版本浏览器中, 以下属性或者属性值也可以作为 Layout 特性的触发器。

- overflow:hidden/scroll: 设置内容超出元素区域时隐藏或滚动的元素。

overflow-x 和 overflow-y 是 CSS3 盒模型中的属性, 尚未得到浏览器的广泛支持。它们对于 IE 6 及其以下版本浏览器没有效果, 对于 IE 7 则是有效的。

- position:fixed: 固定定位的元素。
- max-width 和 min-width 任意值, 除了 none 之外的任意值, 即使被设置为 0 也可以让该元素获取 Layout 特性。
- max-height 和 min-height 任意值, 除了 none 之外的任意值, 即使被设置为 0 也可以让该元素获取 Layout 特性。

例如, 请输入下面源代码, 然后在 IE 6 和 IE 7 中检测这些元素是否拥有 Layout 特性, 显示如图 9.16、图 9.17 所示。

```
<button onclick="alert(
'第 1 个元素的 haslayout=' + div1.currentStyle.hasLayout + '\n\r' +
'第 2 个元素的 haslayout=' + div2.currentStyle.hasLayout + '\n\r' +
'第 3 个元素的 haslayout=' + div3.currentStyle.hasLayout + '\n\r' +
'第 4 个元素的 haslayout=' + div4.currentStyle.hasLayout + '\n\r' +
'第 5 个元素的 haslayout=' + div5.currentStyle.hasLayout)">元素的 haslayout</button>
<div id="div1" style="zoom:1">第 1 个 div 元素: 100%缩放</div>
<div id="div2" style="writing-mode:tb-rl">第 2 个 div 元素: 上下书写顺序</div>
<div id="div3" style="overflow:hidden">第 3 个 div 元素: 自动显示内容</div>
<div id="div4" style="max-height:100px">第 4 个 div 元素: 最大高度</div>
<div id="div5" style="position:fixed">第 5 个 div 元素: 固定显示内容</div>
```

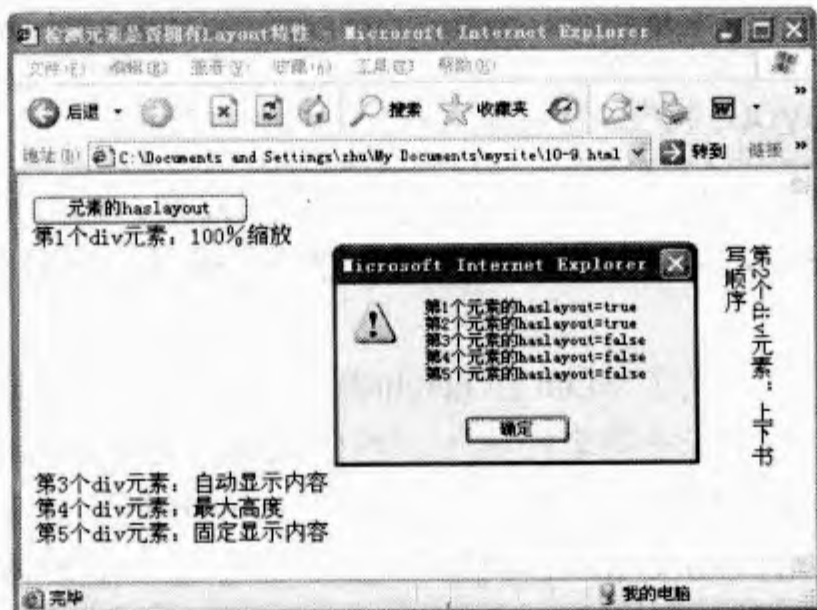


图 9.16 IE 6 中显示效果

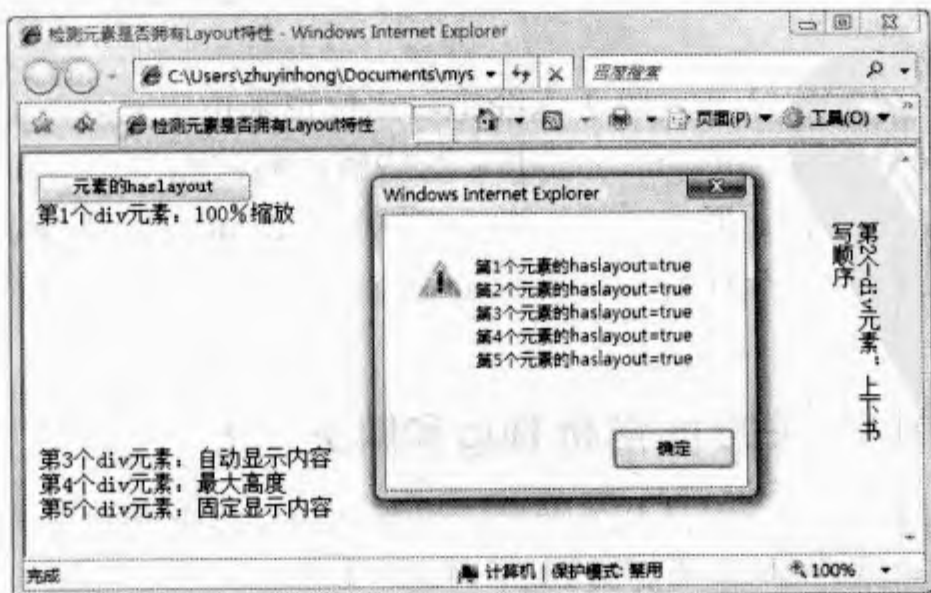


图 9.17 IE 7 中显示效果


除了使用上述方式检测元素的 `hasLayout` 属性值，以确定元素是否具备 Layout 特性外，还可以利用 IE 的网页开发工具条（IE Developer Toolbar）来进行检测（<http://go.microsoft.com/fwlink/?LinkId=92716>）。例如，针对上面的示例，把光标移动到第 4 个元素上并单击时（需要先勾选工具栏中的“通过单击选中元素”按钮图标），此时就可以在工具条右侧的当前样式列表框中看到是否显示 `hasLayout` 属性及其值，在 IE 中如果为 Layout 特性的元素，则显示为 `hasLayout=1`，如果没有则不会显示该属性（如图 9.18 所示）。对于 IE 7 来说，如果有则显示为 `hasLayout=true`，并以只读状态显示（如图 9.19 所示），说明该属性是只读属性。



图 9.18 IE 6 中显示效果

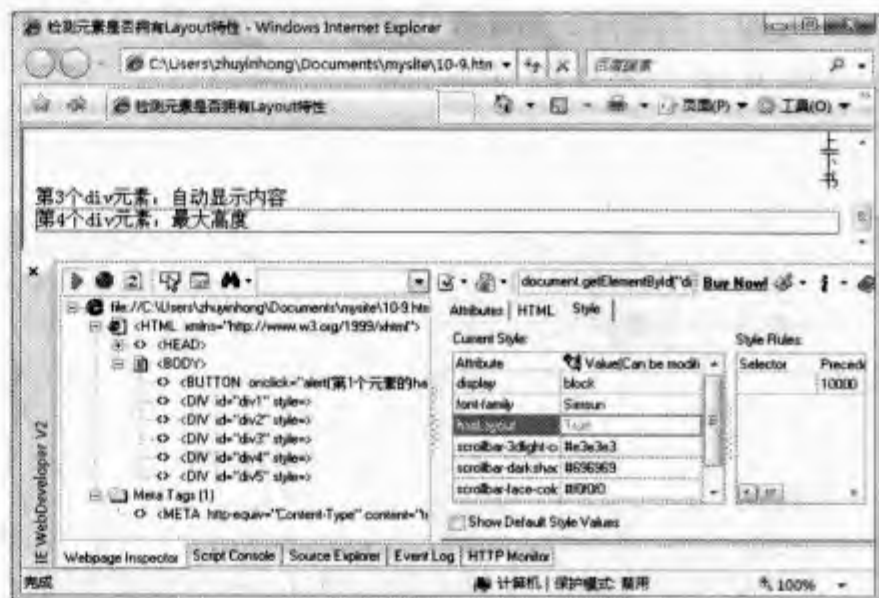


图 9.19 IE 7 中显示效果

9.2.4 行内元素的 Layout 特性

所谓行内元素就是默认或被显式定义 `display:inline` 的元素。行内元素的 Layout 特性似乎有点特殊，具体说来有如下几点需要注意。也正因为这些特殊性，让设计师望 IE 而生畏，特别是 IE 6 版本。

上节讲解了当一个元素定义了 `width` 和 `height` 属性时，会拥有 Layout 特性。但是对于 IE 6 版本来说，如果浏览器运行于标准兼容模式下，行内元素会忽略 `width` 或 `height` 属性，所以即使设置了 `width` 或 `height` 属性，也无法触发该元素的 Layout 特性。只有在 IE 5.x 及其以下版本，或者在 IE 6 以及更新版本的 Quirks mode（怪异模式）下才能够触发该元素的 Layout 特性。

`zoom` 属性总是能够触发元素的 Layout 特性，但是在 IE 5.0 版本中不能够支持该属性。

如果一个元素拥有 Layout 特性，同时也被定义了 `display:inline` 属性值，则它与声明 `inline-block` 功能类似。

在早期的 IE 版本（如 IE 4）中，除了未经绝对定位也未指定宽高的行内元素，几乎所有元素都拥有 Layout 特性。`border`、`margin`、`padding` 这些属性都被称作 Layout 特性，它们是不能应用到一个简单的行内元素上的。通俗说，如果元素拥有 Layout 特性就可以拥有这几个属性。在 IE 5.5 版本中开始引入 Layout 专有属性 `hasLayout`，但是它仅是元素内部的 Layout 特性标志，不能够进行设置。

9.2.5 与 Layout 特性相关的 IE 解析 Bug 和解决方法

IE 浏览器存在很多 Bug，特别是 IE 6 及其以下版本浏览器问题更是严重。这其中很多问题都与 IE 的 Layout 解析机制存在某种关系。

例如，对于如下这样简单的结构，如果我们定义内部元素 `<div id="box">` 为一个拥有高度和

宽度的块元素, 则外部<div id="wrap">元素就无法控制其包含的块元素 (如图 9.20 所示)。

```
<style type="text/css">
#box { /* 内层元素 */
    background:red;           /* 背景色 */
    width:80px;               /* 固定宽度 */
    height:80px;              /* 固定高度 */
    float:left;               /* 向左浮动 */
}
#wrap { /* 外层元素 */
    border:solid 1px blue;    /* 边框线 */
}
</style>
<div id="wrap">
    <div id="box"></div>
</div>
```

对于 IE 浏览器来说, 外层元素<div id="wrap">无法控制其包含的元素<div id="box">, 是因为外层元素没有拥有 Layout 布局特性。

解决这个问题的方法是在 IE 中触发外层元素<div id="wrap">的 Layout 特性。例如, 给它定义一个高度, 这个高度可以设置得非常小, 避免该高度对于元素的实际高度的影响, 所得效果如图 9.21 所示。请注意, 这个高度声明不是真的要定义元素显示多高, 而是作为一个 Layout 特性触发器来使用。

```
#wrap { /* 触发元素的 Layout 特性 */
    height:1%;
}
/* 定义元素的高度 */
```



图 9.20 元素的嵌套布局中存在的 Bug



图 9.21 解决元素的嵌套布局中存在的 Bug

这种方法对于 IE 的任意元素都非常有效, 除了标准兼容模式下 IE 6 版本中的行内元素。但是这种方法不能够与 overflow:hidden 声明一同使用, 否则这个定义的高度就会对元素的实际高度产生影响。当然这种共同的情况也有一个除外, 那就是在 IE 6 的标注模式下, overflow:hidden 声明不会对这个高度声明产生实际影响, 因为这时如果父元素没有显式定义高度, 那么 height:1% 会自动被 IE 6 解析为 height:auto。

另外, 使用 1% 作为 Layout 特性的触发器可能会存在一定的风险, 一般情况下建议使用 height:0 或 1px 会更安全一些。例如:

```
#wrap { /* 触发元素的 Layout 特性 */
    height:0;
}
/* 定义元素的高度 */
```

使用高度来触发 IE 元素的 Layout 特性是设计师的一贯用法, 也是最佳选择。不过如果当为元素声明了 overflow:hidden 之后, 使用高度来触发 Layout 特性就会出现各种问题, 在这种情况下我们可以用 display:inline-block 或 zoom:1 声明来触发 Layout 特性。例如:

```
#wrap { /* 触发元素的 Layout 特性 */
    display:inline-block;           /* 定义行内块状显示 */
}
```

或

```
#wrap { /* 触发元素的 Layout 特性 */
    zoom:1;                         /* 缩放比例 */
}
```

除了避免与 `overflow:hidden` 声明冲突之外,上面这两种方法还可以适用行内元素,以及在 IE 标准模式下触发元素的 Layout 特性。例如,在上面示例基础上,如果定义外部包含元素为内行显示,则其中定义的高度不能触发 Layout 特性,外部元素不能以布局的方式显示,并且包含、控制内部块元素。

```
<style type="text/css">
#box { /* 内层块元素 */
    background:red;           /* 背景色 */
    width:80px;               /* 固定宽度 */
    height:80px;              /* 固定高度 */
    float:left;               /* 向左浮动 */
}
#wrap { /* 外层包含元素 */
    border:solid 4px blue;     /* 边框线 */
    display:inline;           /* 内行元素显示 */
}
</style>
<div id="wrap">
    <div id="box"></div>
</div>
```

但是如果为外层包含元素定义 `zoom` 或者 `display` 属性之后,则显示效果如图 9.22、图 9.23 所示。

```
#wrap {
    border:solid 4px blue;       /* 边框线 */
    display:inline;             /* 内行元素显示 */
    zoom:1;                     /* 缩放比例 */
}
```

或

```
#wrap {
    border:solid 4px blue;       /* 边框线 */
    display:inline;             /* 内行元素显示 */
    display:inline-block;        /* 行内块状显示 */
}
```

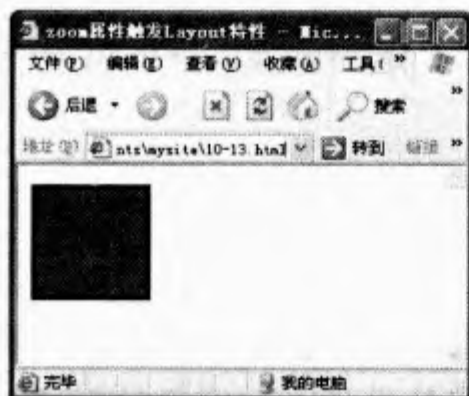


图 9.22 以 zoom 来触发行内元素的 Layout 特性

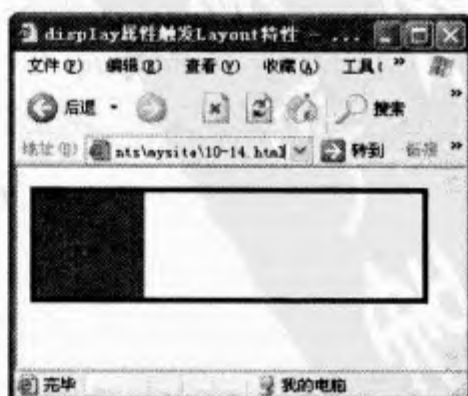


图 9.23 以 display 来触发行内元素的 Layout 特性

9.2.6 能够更好兼容不同浏览器

上节所讲解的方法都是在 IE 浏览器中的测试效果，但是在其他浏览器中，这些方法都被视为无效或者非法。因为在标准浏览器中不支持 Layout 特性，当采用上述方法来为包含元素触发 Layout 特性时，可能会影响到标准浏览器中的解析效果，因此应该采用一些方法使这些声明只能够在 IE 浏览器中被解析。

你还记得第 1 节中讲解的 IE 条件语句吗？这里我们不妨利用 IE 条件语句来限制这些触发器声明只能够在 IE 浏览器中被解析。

```
<style type="text/css">
#wrap {
    border:solid 1px blue;           /* 边框线 */
}
</style>
<!--[if lte IE 6]>
<style type="text/css">
#wrap {
    height:1%;                      /* 高度 */
}
</style>
<![endif]-->
```

这样就可以把这条非样式的 CSS 声明限制在 IE 6 及其以下版本浏览器中使用，对于如何在 IE 7 以及其他标准浏览器中正确显示，我们曾经在第 8 章中详细讲解过，即通过清除法来强迫包含的父级元素撑开以包含浮动的元素。在下面的章节的不同浏览器兼容技术中我们还会详细说明。

另外，我们还可以把这些样式分别放在不同的外部样式表文件中，再通过 IE 条件注释来实现正确显示。

```
<link rel="stylesheet" href="allbrowsers.css" type="text/css" />
<!--[if lte IE 6]>
<link rel="stylesheet" href="iefix.css" type="text/css" />
<![endif]-->
```

IE 条件语句是兼容 IE 浏览器的最锋利的武器，但不是唯一的武器。在我们为了能够兼容不同浏览器而绞尽脑汁而设计出各种 Hack 中，有的是投机取巧利用各种欺骗手段，有的是利用不同浏览器的缺陷或者不完善的功能，有的是利用不同浏览器的专有属性或功能进行设计，以实现在不同浏览器中进行兼容。

但是浏览器总是会变的，我们需要面对很多问题。例如，借助 IE 6 版本浏览器存在的 Bug 所做的 Hack，会在 IE 7 版本或者更高版本的浏览器中因为该 Bug 的修复而失效，甚至起到反作用。在这种情况下，使用浏览器的专有属性来解决浏览器的兼容性问题就是最佳的设计方法。例如，在上面的示例中，除了使用 IE 专有的条件语句外，你还可以使用 IE 浏览器专有的 zoom 属性来进行设计。不过 zoom:1 在 IE 5.0 版本中是无效的。所以最稳妥的方法是结合 IE 条件语句进行设计。

9.2.7 Layout 元素与包含浮动元素的关系

在 IE 浏览器中，任何拥有 Layout 特性的元素都可以自动控制包含的浮动元素。后面的元素都会受到包含框的影响，而不是浮动元素的影响。但是在标准浏览器中却恰恰相反，这对于

设计师来说将是一个不小的挑战，很多初学者往往在这里一筹莫展，不知所措。

例如，在下面这个示例中我们设计了一个 3 层嵌套的结构。最外层是一个包含框（2 像素宽的绿色框），包含所有元素，头部包含框（6 像素宽的红色框）中包含了 Logo 和 Headline 两个对象，Logo（10 像素宽的蓝色框）是一个浮动元素。

```
<style type="text/css">
#wrap { /* 外包装框 */
    width: 100%; /* 宽度 */
    border: 2px solid green; /* 边框 */
}
#header { /* 头部包含框 */
    width: 90%; /* 宽度 */
    border: 6px solid red; /* 边框 */
}
#logo { /* Logo 样式 */
    float: left; /* 向左浮动 */
    width: 100px; /* 宽度 */
    height: 200px; /* 高度 */
    border: 10px solid blue; /* 边框 */
}
h1, h2 { /* 标题背景色 */
    background: #aaa; /* 背景色 */
}
h2 { /* 底部对象 */
    float: left; /* 向左浮动 */
}
p { /* 段落文本的背景色 */
    background: #666; /* 背景色 */
}
.clear { /* 清除类 */
    clear: both; /* 左右清除 */
}
</style>
<div id="wrap">
    <div id="header">
        <div id="logo">Logo</div>
        <h1>Headline</h1>
    </div>
    <h2>Subhead</h2>
    <p class="clear">clear</p>
</div>
```

这时你可以在 IE 7 或其他版本中预览，会发现 Subhead 对象浮动在头部包含框（<div id="header">）的底部（如图 9.24 所示），但是如果在 FF 等标准浏览器中预览，你会发现 Subhead 对象浮动在 Logo 浮动对象的右侧（如图 9.25 所示），而不受头部包含框（<div id="header">）的影响。这种问题在 CSS 浮动布局中经常会遇到，这对于初学者来说也是比较头疼的问题，在标准浏览器中我们只能通过浮动清除来设计相同的效果。

在本示例中我们可以为 Subhead 对象增加定义 clear 属性，或者增加清除元素，代码如下：

```
h2 { /* 底部对象 */
    clear: both; /* 清除浮动 */
}
```

或者

```
<div id="wrap">
    <div id="header">
```



```
<div id="logo">Logo</div>
<h1>Headline</h1>
<p class="clear"></p>
</div>
<h2>Subhead</h2>
<p class="clear">clear</p>
</div>
```

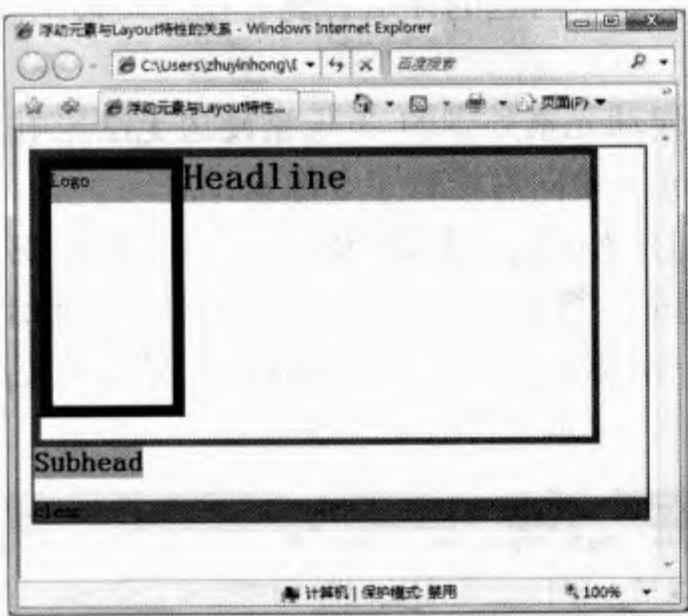


图 9.24 IE 7 中浮动元素的显示

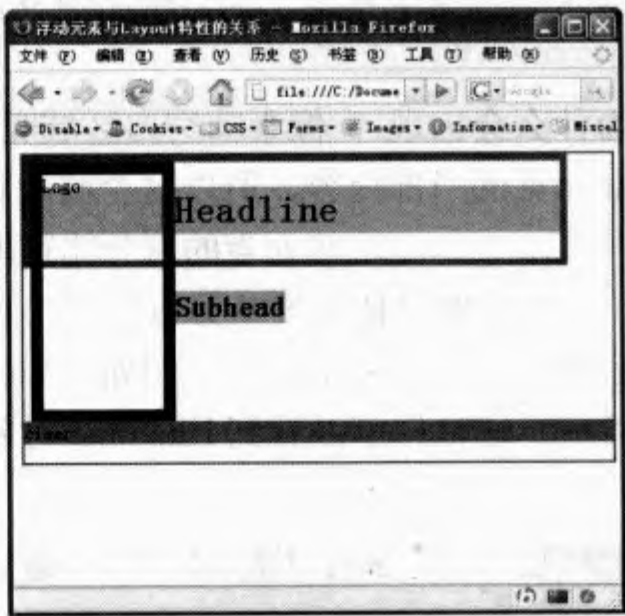


图 9.25 FF 2 中浮动元素的显示

9.2.8 Layout 元素与相邻浮动元素的关系

当一个块状元素紧跟在一个左浮动元素之后时，它会忽略这个浮动元素，而它包含的内容则应沿着浮动元素的右边顺序排列，如果它的长度超过浮动元素，则会继续排列到浮动元素底部。

但是如果这个块状元素拥有 Layout 特性，例如，由于某种原因被设置了宽度，那么整个元素会因为浮动元素而移位，就好像它自己也是一个浮动元素一样，因此其中的文字就不再环绕这个左浮动元素了，而是形成一个矩形区域，停靠在浮动元素的右侧。

例如，在上面示例的基础上我们定义 h1 元素宽度为 80%，由于被定义了宽度，则该元素就自动拥有 Layout 特性，这时如果在 IE 中写入如下代码：

```
h1 { /* 一级标题 */
    width:60%;
}
```

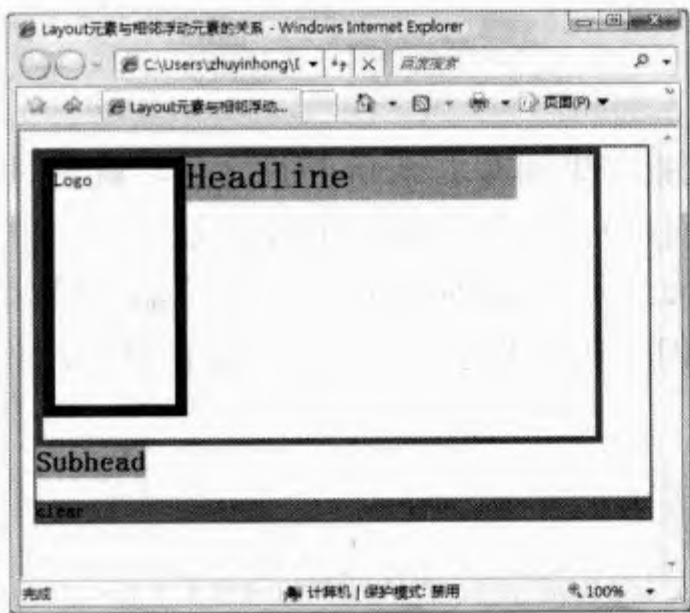


图 9.26 IE 7 中 Layout 元素的显示

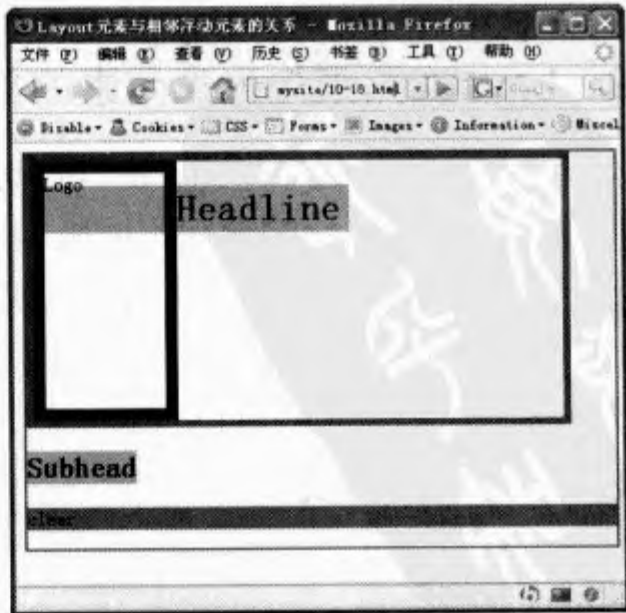


图 9.27 FF 2 中 Layout 元素的显示

你可以看到，在 IE 中 Headline 对象内容不再是文本的形式环绕在浮动元素的右侧，而是以块的形式布局浮动元素的右侧，犹如一个浮动的元素（如图 9.26 所示）。但是在标准浏览器中（如 FF 2）就会发现该元素的内容依然环绕在浮动元素的右侧，元素本身并没有受到浮动元素的影响（如图 9.27 所示）。

在第 7 章中我们曾经介绍过 IE 6 版本浏览器存在 3 像素的 Bug，例如，上面的示例如果在 IE 6 版本浏览器中预览，则显示如图 9.28 所示，有关这个问题的详细讲解可以参阅第 7.4.8 节内容。

为什么会出现 3 像素的空白区域呢？这是因为浮动元素外围的 3 像素硬边无法影响一个 Layout 元素的内部内容，所以这个硬边将整个 Layout 元素向右推移了 3 像素。

在 IE 5 中一个块状元素的百分比宽度是基于浮动元素旁边的剩余空间计算的，而在 IE 6 及其以上版本中则是依照整个父包含框元素的可用空间来计算的，所以在 IE 6 中设置 `width:100%` 时就会导致该对象溢出显示。例如，如果我们定义一级标题的宽度为 100%，则显示效果如图 9.29 所示。这时你可以看到 Headline 对象错位到下一行显示。

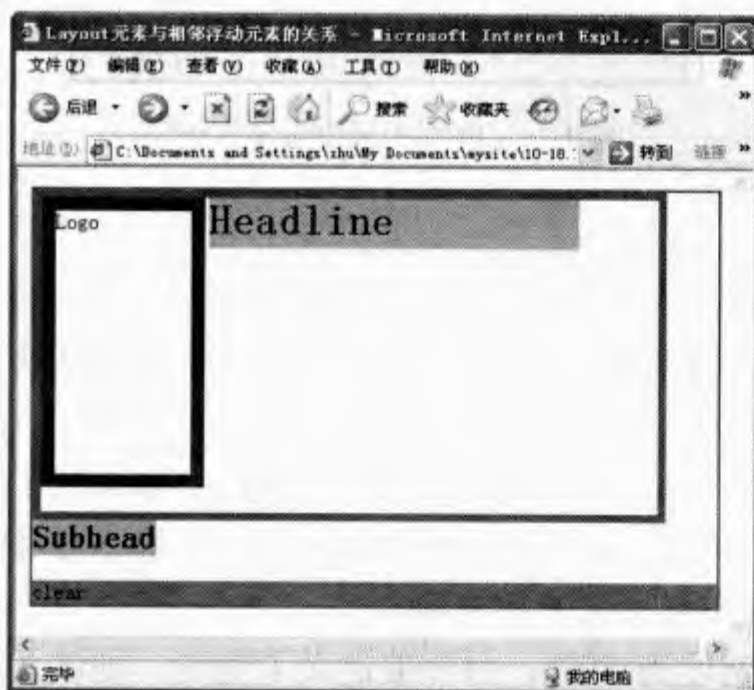


图 9.28 IE 6 的 3 像素问题



图 9.29 IE 的 Layout 元素错位问题

```
h1 { /* 一级标题 */
    width:100%;
}
```

/* 元素的宽度 */

9.2.9 Layout 元素与列表元素的关系

列表元素包括 `ol`、`ul`、`dl`、`li`、`dt` 和 `dd`，这些元素如果拥有 Layout 特性之后，就会对列表显示产生影响，这种影响会因不同版本浏览器显示而略有不同。例如，在下面这个列表中，我们定义列表的宽度为 60%，则 `ul` 元素就拥有了 Layout 特性。这时如果在 IE 浏览器中显示，则会发现该列表的项目符号消失了（如图 9.30 所示）。但是在标准浏览器中显示为正常（如图 9.31 所示）。

```
<style type="text/css">
ul { /* 列表元素 */
    width:60%;
}
</style>
```

/* 宽度 */


```
<ul>
  <li>项目列表</li>
  <li>项目列表</li>
  <li>项目列表</li>
</ul>
```



图 9.30 IE 7 的 Layout 项目列表问题

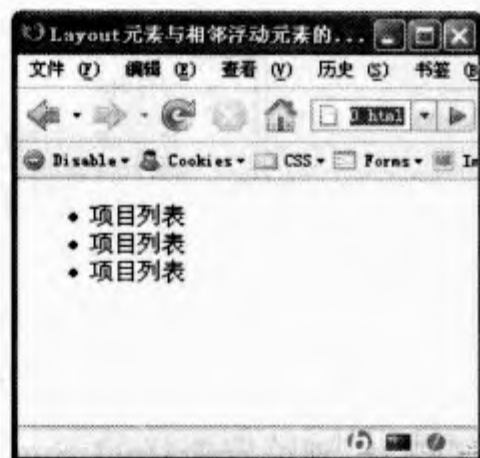


图 9.31 FF2 的 Layout 项目列表

自定义的列表符号则不会受到 Layout 特性的影响。但是由于这些符号是通过某种内部机制附到列表元素上的，而这种内部机制是无法通过人工来控制的。解决此类问题只有通过自定义项目符号来实现。不过，你可以通过改变列表元素的内边距而让它重新出现（如图 9.32 所示）。

```
ul { /* 列表元素样式 */
  width:60%;
  padding-left:1em; /* 定义内边距 */
}
```

在有序列表中，任何具有 Layout 特性的列表项元素都会拥有独立的计数器。例如，在下面这个简单的示例中，如果定义第 3 个列表项拥有 Layout 特性，则你会看到第 3 个项目列表重新计数（如图 9.33 所示）。

```
<style type="text/css">
.l3 { /* 定义列表项元素拥有 Layout 特性 */
  zoom:1; /* 缩放比例 */
}
</style>
<ol>
  <li class="l1">项目列表</li>
  <li class="l2">项目列表</li>
  <li class="l3">项目列表</li>
  <li class="l4">项目列表</li>
</ol>
```



图 9.32 通过调整内边距显示项目符号

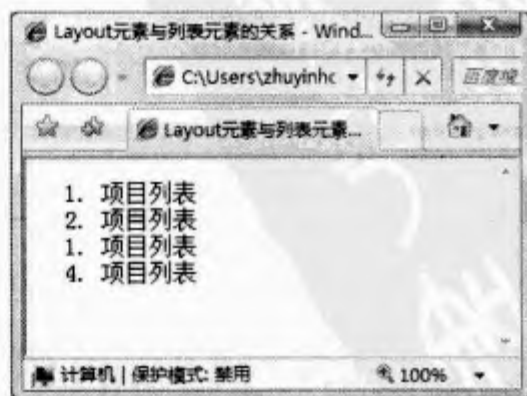


图 9.33 列表项目重新计数

此外,如果一个列表项拥有 Layout 特性,当该 Layout 列表项元素跨行显示时,项目符号会底部对齐,而不是按照一般的思维习惯进行顶部对齐(如图 9.34 所示)。

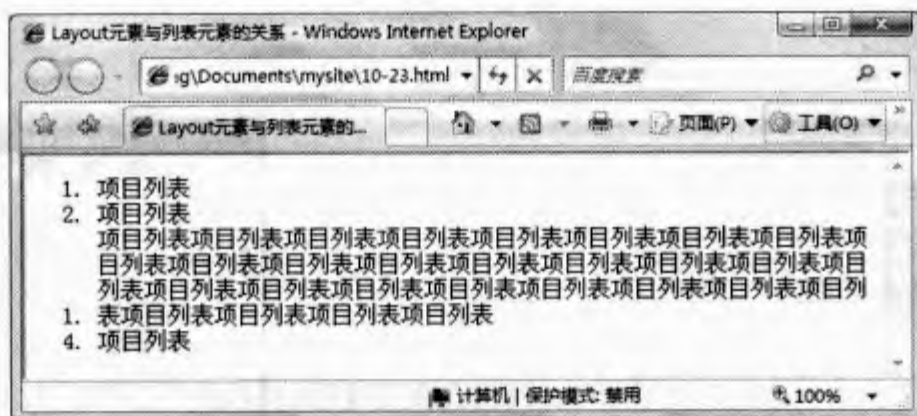


图 9.34 项目符号底部对齐效果

所有这些 IE 浏览器奇怪的 Bug 目前还无法直接解决。因此,如果读者定义项目符号,可以通过自定义项目符号的方法来避免此类问题。而如果要限制列表的宽度,不妨通过外部增加包含框,通过定义包含框的宽度来控制列表的宽度。而对于列表项目的高度,则可以通过为每个列表项元素中的内容设定高度等方法来实现。

另外,在 IE 6 及其以下版本浏览器中,当列表项元素 li 中包含有块状显示的超链接元素时,则在这种情况下列表元素之间的空格将不会被忽略,而且会额外增加一行。例如,在下面这个项目列表中我们定义第 2、第 3 个列表项中包含的超链接元素 a 为块状显示,你会发现在这 2 行列表项下面多增加 1 行(如图 9.35 所示)。

```
<style type="text/css">
.a2, .a3 { /* 超链接块状显示 */
    display: block; /* 块状显示 */
}
</style>
<ol>
<li class="l1"><a href="#" class="a1">项目列表 1</a></li>
<li class="l2"><a href="#" class="a2">项目列表 2</a></li>
<li class="l3"><a href="#" class="a3">项目列表 3</a></li>
<li class="l4"><a href="#" class="a4">项目列表 4</a></li>
</ol>
```

为解决 IE 6 及其以下版本浏览器的这种 Bug,可以把这些超链接元素定义为 Layout 元素,这时你会发现元素全部都恢复为原来的显示距离(如图 9.36 所示)。



图 9.35 列表项下面被额外增加了一行空行



图 9.36 解决列表项下面被额外增加了一行 Bug

```
.a2, .a3 { /* 定义元素拥有 Layout 特性 */
    display: block;
    zoom: 1;
}
```

```
/* 块状显示 */
/* 缩放显示 */
```


9.2.10 Layout 元素与定位元素的关系

由于相对定位属性（`position:relative`）不能够触发 Layout 特性，所以在使用定位布局时会出现很多莫名其妙的错误，如定位对象丢失、定位元素错位等现象；而当用户再次刷新页面，或者调整浏览器窗口大小、滚动滚动条时又会显示这些定位对象。

因此，当出现这类问题时，建议为相对定位元素声明 `zoom:1`，或者通过其他方式触发定位元素的 Layout 特性。

在 CSS 定位布局中，我们比较喜欢使用相对定位来为包含框定义包含块，然后为其中包含的各种元素进行定位，也就是说我们可以让一个绝对定位元素所参考的原点和长度等属性不依赖于元素的排列顺序，这可以满足诸如“内容优先”这种可访问性概念的需要，也可以给复杂的浮动布局带来方便。

但是由于 IE 浏览器的 Layout 解析特性，因此只有当包含块拥有了 Layout 特性之后，IE 浏览器才能够正常地进行定位。例如，我们把一个行内元素定义为包含块，然后在其中包含一个方形盒子，通过绝对定位的方法定位该元素距离包含块左上顶角的坐标。

```
<style type="text/css">
#wrap { /* 定义包含块 */
    margin:100px;
    position:relative;
    border:solid 4px red;
}
#box{ /* 定义方形盒子 */
    position:absolute;
    left:100px;
    top:100px;
    width:100px;
    height:100px;
    background:blue;
}
</style>
<span id="wrap">
    <div id="box"></div>
</span>
```

/* 外边距 */
/* 相对定位 */
/* 边框 */

/* 绝对定位 */
/* 与包含块左侧的距离 */
/* 与包含块顶端的距离 */
/* 宽度 */
/* 高度 */
/* 蓝色背景 */

我们可以看到由于包含块（``）没有 Layout 特性，因此它在解析定位元素时无法准确定位绝对元素，此时绝对元素以窗口左上顶角为参考进行定位（如图 9.37 所示）。但是如果给包含块激活 Layout 特性，它就能够准确定位绝对定位元素了（如图 9.38 所示）。

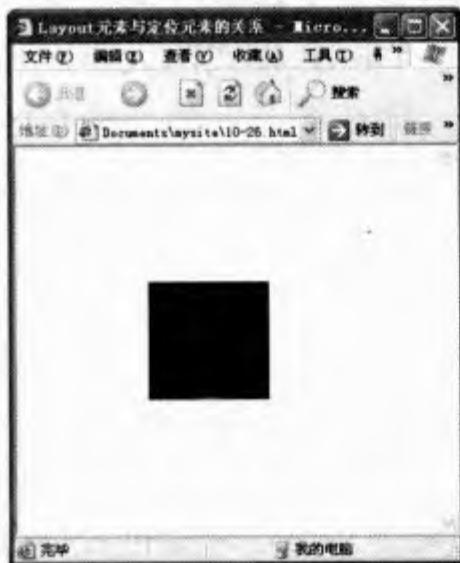


图 9.37 对没有 Layout 特性的包含块进行定位

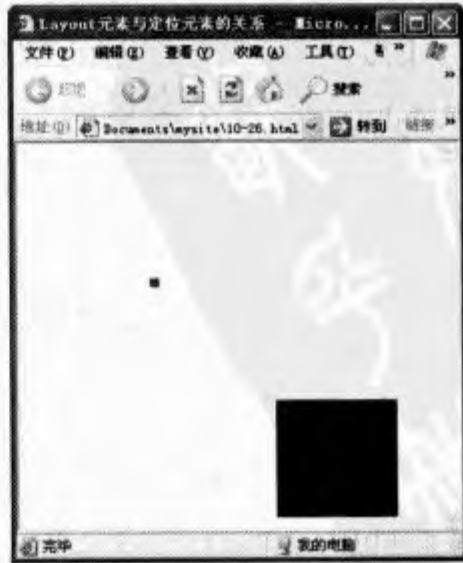


图 9.38 对拥有 Layout 特性的包含块进行定位

```
#wrap { /* 激活包含块的 Layout 特性 */
    zoom:1;                                /* 缩放比列 */
}
```

另外，如果我们为绝对定位元素取百分比宽度时，它也不能够准确找到包含块的大小并进行计算。例如，针对上面示例如果我们定义绝对定位元素的宽度为 50%。这时如果没有激活包含块的 Layout 特性，你会发现绝对定位元素的宽度是以窗口宽度为参考进行计算的。如果我们为包含块声明 zoom:1，激活它的 Layout 特性，则绝对定位元素以包含块为参考进行宽度计算（如图 9.40 所示）。

```
#box{ /* 绝对定位元素 */
    width:50%;                                /* 百分比宽度 */
}
```

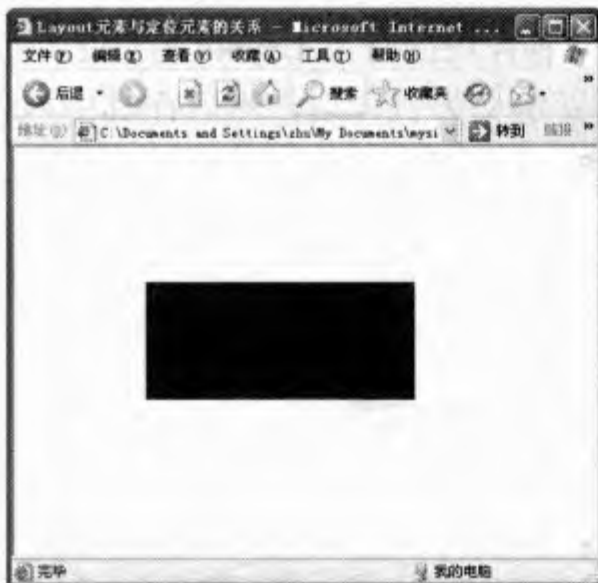


图 9.39 无 Layout 特性的宽度计算

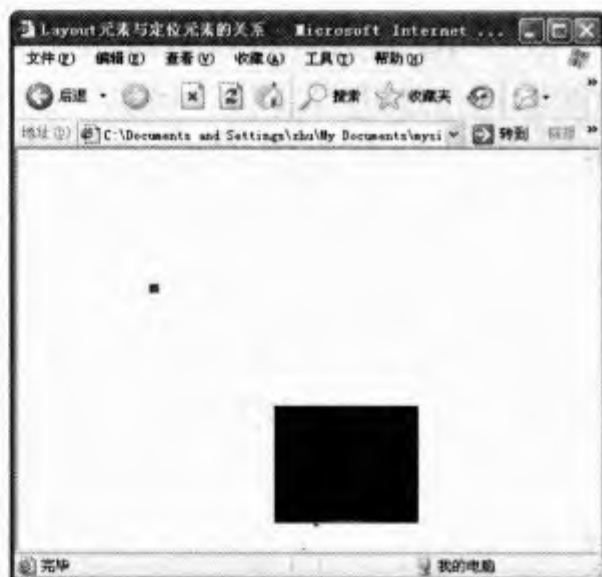


图 9.40 拥有 Layout 特性的宽度计算

IE 5 和 IE 6 在定位元素的 Layout 特性方面比较相近，不过 IE 7 在这方面有了很大进步。总之，尽可能保证绝对定位元素的包含块拥有 Layout 特性，而且尽量让其成为绝对定位元素的父级元素，也就是说这个包换元素和绝对定位元素之间没有嵌套别的定位元素。

9.3 浏览器兼容技术解析

我们常说浏览器的解析模式，这个解析模式就是所谓的浏览器的渲染引擎（或者说网页呈现的方法），它犹如人的心脏、汽车的发动机。一般来说不同浏览器解析网页的模式是不同的。从大的角度来看，浏览器的解析模式可以分为 IE 模式和非 IE 模式两大类，当然如果细分，每种版本的浏览器在解析网页时都会呈现不同的效果，即在细节上都会略有区别。本节将讲解如何兼容这些不同的浏览器，最终使自己设计的网页在不同浏览器中都会呈现相同的效果。

9.3.1 你的浏览器符合标准吗

提及浏览器是否符合标准，相信每位读者都会不假思索地说：IE 浏览器不够标准，FF 比较标准，Opera 最标准等。是的，基本上这已经成为了网页设计的常识，当然现在还没有发现完全符合最新标准（符合 CSS 3 和 HTML 5）的浏览器。但是你知道什么是符合标准，又根据什么来判断的吗？

所谓网页设计的标准是根据 W3C 组织提出的一套网页设计规则, 包括 HTML、XHTML、CSS、XML 标准等。

为了判断不同版本浏览器的解析模式是否符合标准, 1997 年网页标准计划小组 (Web Standards Project) 设计了一个测试页面 (<http://www.webstandards.org/files/acid2/test.html>) 来检测不同版本浏览器的健康状态 (是否符合标准), 这种测试被称为 Acid。该测试页面所呈现的标准效果如图 9.41 所示。这是一个有趣的机器人头像, 在标准状态下, 当你把鼠标移到机器人的鼻子上时, 鼻子会立即由黑色变为蓝色。

但是就这么一个简单的测试页面 (HTML+CSS 共 460 多行代码), 背后却蕴涵着许多网络技术, 感兴趣的读者可以访问并查看源代码, 它不是很复杂, 但是很值得去认真研究, 以将代码设计得严格、严谨、严密。

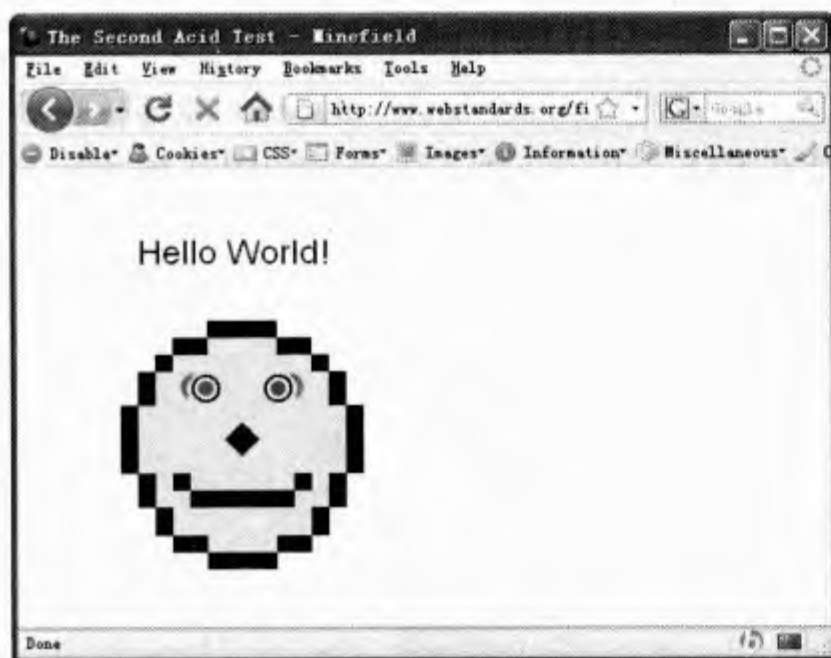


图 9.41 FF 3 版本浏览器的测试结果

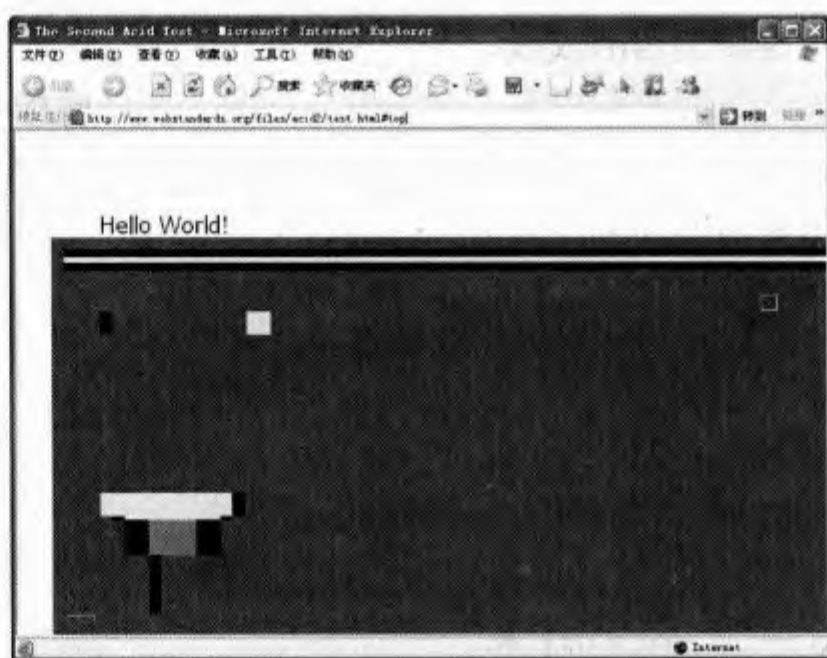


图 9.42 IE 6 版本浏览器的测试结果

如果使用 IE 6 版本浏览器预览这个页面, 你会发现 IE 6 所渲染的结果完全不成形了 (如图 9.42 所示)。IE 7 版本浏览器所解析的效果也是惨不忍睹 (如图 9.43 所示)。

苹果的 Safari 2.02 是第一个通过 Acid2 测试的, 之后 Konqueror、Opera 也陆续过关, 而 FF 2 版本还不行 (如图 9.44 所示), 但是 FF 3 正式版本通过了 Acid2 测试。据说即将发布的 IE 8 也将要通过 Acid2 测试。



图 9.43 IE 7 版本浏览器的测试结果

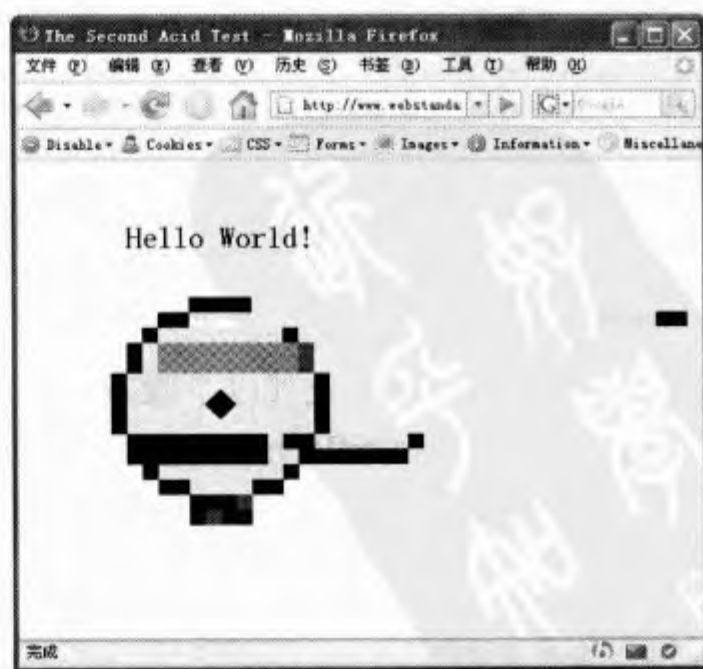


图 9.44 FF 2 版本浏览器的测试结果

9.3.2 兼容 IE 6 版本浏览器

IE 6 版本浏览器可以说是目前寿命最长的一个浏览器版本，自从 2001 年 10 月微软发布 Windows XP 时捆绑了 IE 6，直到 2006 年下半年微软才开始发布 IE 7，在 2007 年 1 月发布 Windows Vista 时捆绑了 IE 7。不过目前 IE 6 仍然是大部分用户默认使用的浏览器，虽然微软现在开始强制所有用户都升级到 IE 7，但是这需要一个过程，估计短期内 IE 6 仍然是大多数用户的首选。

IE 6 的用户数最多，它存在的问题也最多。很多设计师对其是既爱又恨，说爱是因为它的霸主地位，相信任何一位设计师在设计页面时，首先要保证页面能够在 IE 6 浏览器中正确显示，然后再在其他浏览器中进行测试。说恨，是因为 IE 6 过于固执，很不符合标准网页的设计要求，结果导致了设计师在设计页面时必须一心二用，一方面要兼顾 IE 6 的脸色，另一方面还要顾虑标准浏览器的权威。

下面我们先看一个示例。这是一个经验槽，设计经验值（<div class="value">）能够根据后台数据进行自动伸缩显示（如图 9.45 所示）。但是在 IE 6 及其以下版本中显示如图 9.46 所示，所显示的高度实际为 22 像素。

```
<style type="text/css">
.trough {/* 包含框的样式 */
    width:300px;
    height:8px;
    padding:1px;
    border:1px solid #53d242;
}
.value {/* 经验值的样式 */
    width:60%;
    background:#66cc00;
    height:8px;
}
</style>
<div class="trough">
    <div class="value"> </div>
</div>
```

```
/* 宽度 */
/* 高度 */
/* 内边距 */
/* 边框 */
```

```
/* 活动宽度 */
/* 背景色 */
/* 高度 */
```

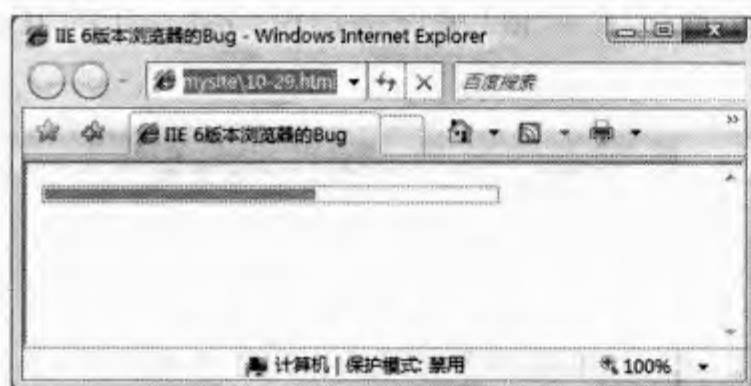


图 9.45 IE 7 中预览结果



图 9.46 IE 6 中预览结果

很显然这是 IE 6 及其以下版本存在的一个 Bug，解决的方法是：专门为 IE 6 及其以下版本浏览器声明一个 `overflow:hidden` 即可。那么如何实现专门为 IE 6 及其以下版本浏览器定义样式呢？除了可以使用 IE 条件语句外，你还可以使用如下过滤器来进行设置，所得的设计效果如图 9.47 所示。

```
* html .value { /* 兼容 IE 6 及其以下版本的样式 */
    overflow:hidden;
}
/* 隐藏超出内容区域 */
```


对于标准浏览器来说,html 元素被认为根元素,其他元素都被包括其中。但是,在 IE 6 及其以下版本浏览器中认为匿名元素为根元素,html 元素被认为是它的子元素,这个匿名根元素就是通用选择符*。我们可以使用*符号包含 html 元素定义一个特殊的选择符过滤器,专门定义只能在 IE 6 及其以下版本浏览器中应用的样式。

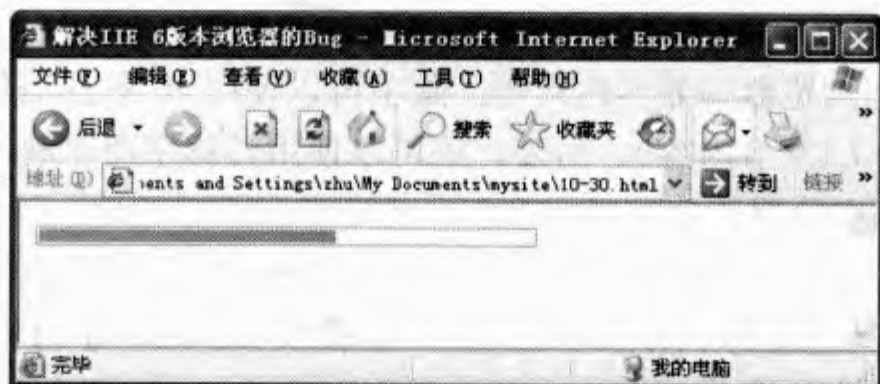


图 9.47 经过兼容处理后的 IE 6 中显示效果

最后请读者记住:如果希望某个样式只能够在 IE 6 及其以下版本浏览器执行,可以使用上面的*html.value {}过滤器来进行设计,这也就是所谓的浏览器兼容性处理技术。

9.3.3 兼容 IE 5 系列版本浏览器

IE 5 系列版本浏览器目前已经很少有人使用了,这里仅简单地介绍一下。IE 5 系列版本浏览器存在的问题也不少,其中最突出的问题就是对于盒模型大小的算法上存在很大的差异。

其他浏览器一般都认为元素的总宽度或总高度为外边距、内边距、边框和宽或高的总和,而 IE 5 系列版本浏览器却认为元素总宽度或总高度为外边距和宽度或高度的总和,换句话说就是 IE 5 系列版本浏览器认为宽度或高度包含了元素的边框和内边距。

例如,定义一个盒模型,设置宽度为 100 像素,然后在 IE 6 (如图 9.48 所示)和 IE 5.5 (如图 9.49 所示)中预览,你会发现 IE 5.5 的盒子明显窄了许多。这是因为它把元素的边框和内边距都算为元素的宽度。

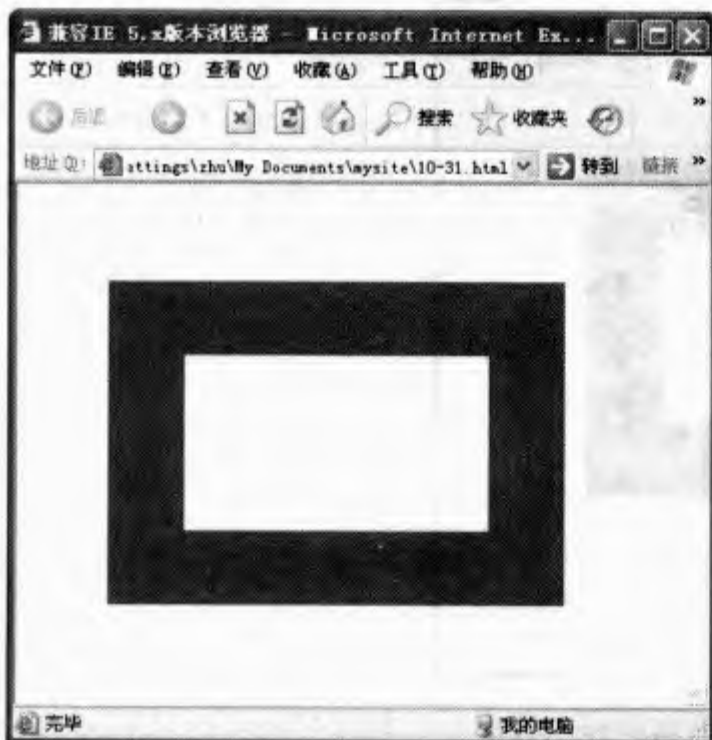


图 9.48 IE 6 中预览结果



图 9.49 IE 5.5 中预览结果

```
<style type="text/css">
#box {/* 盒模型 */
```

```

width:100px;           /* 宽度 */
border:solid 50px solid red; /* 边框 */
margin:50px;          /* 外边距 */
padding:50px;         /* 内边距 */
}
</style>
<div id="box"></div>

```

解决 IE 5 系列版本浏览器的兼容性问题，除了可以使用 IE 条件语句外，还可以使用多媒体声明来定义（参阅第 7.1.9 节内容），另外，还可以使用转移字符来解决，也许这是最简单的一种兼容 IE 5 系列版本浏览器的方法了。

所谓转移字符就是字符前面带有一个反斜杠前缀，它能够保证该字符按本来的意思（原义）解析，而不会被翻译为其他意思，或者通过反斜杠前缀来表示某个字符代表特殊的意思。例如，“\n”转义为换行符，“\r”转义为回车符。

但是 IE 5 系列版本浏览器不能够识别这种转移字符，当看到这个反斜杠时，会误认为反斜杠也是一个普通字符，从而躲避被 IE 5 系列浏览器解析的可能。

例如，针对上面的示例，使用转义字符重新定义盒模型的样式。由于 IE 5 系列版本浏览器不能够识别转义字符，所以就忽略了 width:h:100px 声明，而采用 width:300px。但是对于 IE 6 等其他浏览器来说，由于能够识别转义字符，所以认为盒模型的宽度为 width:h:100px，在 IE 5 中预览结果如图 9.50 所示。

```

#box { /* 盒模型 */
width:300px;           /* IE5 系列版本中的宽度 */
width:h:100px;        /* 转义宽度 */
border:solid 50px solid red; /* 边框 */
margin:50px;          /* 外边距 */
padding:50px;         /* 内边距 */
}

```

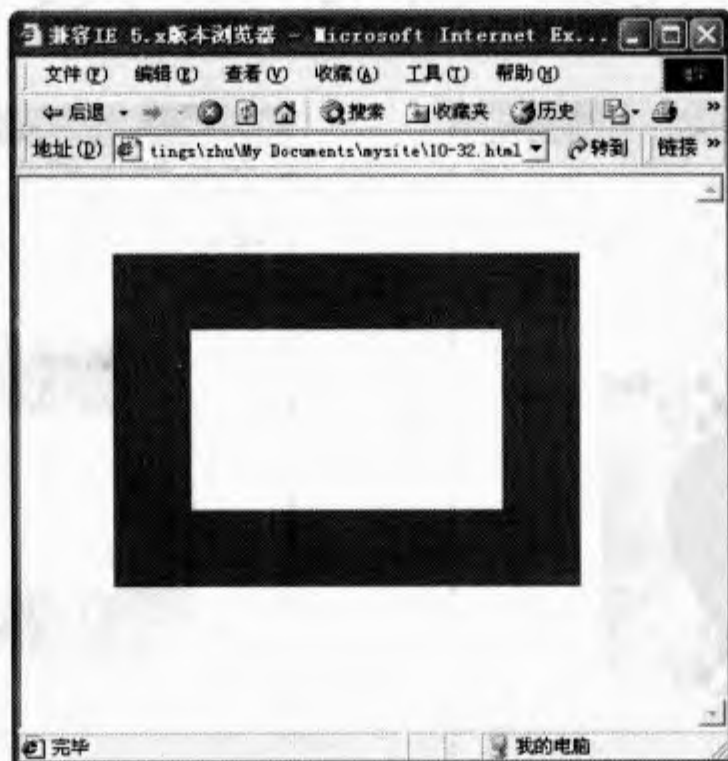


图 9.50 兼容 IE 5 系列浏览器的显示效果

使用转义字符时要注意，不要把反斜杠放在数字 0~9 或字母 a~f、n、r、t、v（包括大小写）的前面，因为这些转义字符可能会表示其他意思，而不是表示字符原义。

当然除了这个盒模型宽度问题外，还可以利用这些技巧来解决 IE 5 系列浏览器的其他可能

存在的 Bug。

9.3.4 兼容 IE 7 版本浏览器

IE 7 版本浏览器经过微软的洗心革面之后, 向 W3C 标准靠近了一大步。它解决了 IE 6 存在的很多荒唐 Bug, 也支持了 CSS 2 版本中的多个选择符, 以及常用布局解析方法。但是由于 IE 7 依然构建在 IE 的核心引擎之后, 其解析网页的模式还是存在很多问题, 当然要少了很多。

如果要单独解决 IE 7 存在的 Bug, 我们可以使用 `*+html Element {}` 过滤器专门为 IE 7 定义样式。在上面示例中我们曾经讲解了 IE 浏览器认为 `*` 是 HTML 的根元素, 因此这个过滤器只能适用于 IE 浏览器, 由于仅有 IE 7 支持 `*+html` 相邻选择符, 所以该过滤器仅能够在 IE 7 中被识别和解析, 不过如果 IE 8 版本出来后, 它应该说也适用 IE 8 版本浏览器。

例如, 在下面这个简单的示例中, 定义盒子在所有浏览器中显示为红色背景、蓝色实线边框 (如图 9.51 所示), 然后通过兼容技术处理, 利用 `*+html` 过滤器专门为 IE 7 定义为蓝色背景、红色虚线框样式 (如图 9.52 所示)。

```
<style type="text/css">
#box { /* 适用所有浏览器的样式 */
    width:200px;
    height:50px;
    border:solid blue 8px;
    background:red;
}
*+html #box { /* 专门适用 IE 7 版本浏览器的样式 */
    border:red dashed 8px;
    background:blue;
}
</style>
<div id="box"></div>
```



图 9.51 IE 6 中预览结果



图 9.52 IE 7 中预览结果

该过滤器仅适用于 IE 7 版本浏览器, 是一个不错的过滤器。但是应该保证在 HTML 的顶部增加 DTD 声明, 否则无效。

9.3.5 兼容 FF 等标准浏览器

对于以 FF 为代表的非 IE 标准浏览器来说, 一样可以通过各种方法为它们单独定义样式, 实现自由兼容。其中比较通用的过滤器为 `html>*/body #Element {}`。该过滤器能够保证所定义的样式只能够在非 IE 标准浏览器中被解析。

例如, 在下面这个简单的示例中, 定义盒子在所有浏览器中显示为红色背景、蓝色实线边框 (如图 9.53 所示), 然后通过兼容技术处理, 利用 `html>*/body` 过滤器专门为非 IE 浏览器定

义为蓝色背景、红色虚线框样式（如图 9.54 所示）。

```
<style type="text/css">
#box { /* 适用所有浏览器的样式 */
    width:200px;           /* 宽度 */
    height:50px;           /* 高度 */
    border:solid blue 8px;  /* 实线边框 */
    background:red;        /* 红色背景 */
}
html>/**/body #box { /* 专门适用非 IE 浏览器的样式 */
    border:red dashed 8px; /* 虚线边框 */
    background:blue;      /* 蓝色背景 */
}
</style>
<div id="box"></div>
```

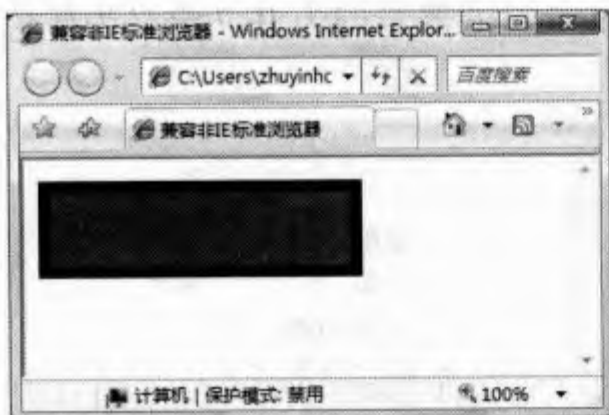


图 9.53 IE 7 中预览结果



图 9.54 FF2 中预览结果

9.4 深入剖析 CSS 层叠布局

在第 9 章中曾经就 CSS 层叠问题简单进行过讲解，但是考虑到这个问题是很多初学者学习 CSS 的一个难点和重要，故本节继续就此问题深入剖析，以帮助读者明白在 CSS 定位布局中元素之间的层叠关系和相互影响。

9.4.1 认识 CSS 层叠包含框

在前面章节中曾经提及过 CSS 包含框、CSS 包含块等各种专业术语。CSS 包含框是从 HTML 结构关系上来定位不同对象之间的关系。例如，在下面这个结构中，我们可以说<div id="container">是<div id="box1">的包含框，而<div id="wrap">是<div id="container">的包含框，因为它们的结构是嵌套、包含关系。

```
<div id="wrap">
  <div id="container">
    <div id="box1"> </div>
  </div>
  <div id="box2"> </div>
</div>
```

而对于 CSS 包含块来说，我们就不能够说<div id="container">是<div id="box1">的包含块，或者<div id="wrap">是<div id="container">的包含块。这就要看这些对象的定位性质，如果它们都被定义为定位元素，则这样说是正确的；相反，如果<div id="wrap">被定义为定位元素，而<div id="container">为自然流动，则只有<div id="wrap">是包含块。

清楚了上面的概念,下面我们就来讲解 CSS 布局中另一个重要概念——CSS 层叠框。我们知道 CSS 的 `z-index` 属性用来在定位布局中确定元素的层叠顺序,通俗说就是元素堆放在一起时决定哪个元素显示在上面、哪个元素显示在下面,也就是从三维立体空间的 Z 轴决定元素的定位。

`z-index` 属性适用于定位元素,也就是说 `position` 属性值为 `relative`、`absolute` 或 `fixed` 的对象。还记得第 7 章中讲解的包含块吗?下面我们来深入剖析 CSS 的层叠体系。

CSS 规定:当一个元素被定位之后,它就归属于某个 `Stacking Context` (层叠关系)。层叠关系如同包含块一样,也可以通俗地说成层叠包含框,它是层叠定位的一个参考平台,在某个层叠包含框内部的所有定位元素都可以在同一个平台上比较定位自己的 `z` 轴坐标。

例如,针对上面的结构,如果我们定义 `<div id="wrap">` 为定位元素,且赋予给它一个 `z-index` 值,使它成为一个层叠关系框。这时如果内部的其他元素被定义为定位元素,且出现层叠现象,则就会以这个 `<div id="wrap">` 层叠关系框作为比较平台,来决定相互覆盖关系。在下面这个示例中我们定义黄色盒子的层叠值为 20,而绿色盒子的层叠值为 10。现在由于它们都处于同一个层叠关系平台 `<div id="wrap">`,所以黄色盒子覆盖绿色盒子(如图 9.55 所示)。

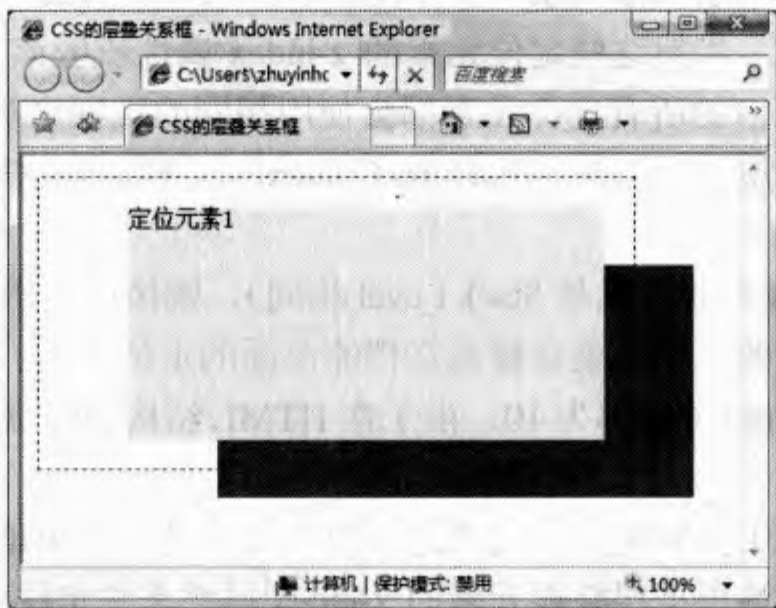


图 9.55 层叠包含框以及内部定位元素的层叠顺序

```
<style type="text/css">
#wrap { /* 定义层叠包含框 */
    position: absolute; /* 绝对定位 */
    border: dashed 1px blue; /* 虚线框 */
    width: 400px; /* 固定宽度 */
    height: 200px; /* 固定高度 */
    z-index: 0; /* 层叠值 */
}
#box1 { /* 定位黄盒子 */
    position: absolute; /* 绝对定位 */
    width: 80%; /* 百分比宽度 */
    height: 80%; /* 百分比高度 */
    top: 20px; /* 顶部距离 */
    left: 60px; /* 左侧距离 */
    background-color: yellow; /* 黄色背景 */
    z-index: 20; /* 层叠值 */
}
#box2 { /* 定位绿盒子 */
    position: absolute; /* 绝对定位 */
    width: 80%; /* 百分比宽度 */
    height: 80%; /* 百分比高度 */

```

```

top: 60px; /* 顶部距离 */
left: 120px; /* 左侧距离 */
background-color: green; /* 绿色背景 */
z-index: 10; /* 层叠值 */
}
</style>
<div id="wrap">
  <div id="container">
    <div id="box1">定位元素 1</div>
  </div>
  <div id="box2">定位元素 2</div>
</div>

```

请记住，层叠包含框元素可以称为 Root Stacking Context（层叠根元素），该元素必须是一个定位元素，且 z-index 属性值应为一个非 auto 的值，否则无效。

9.4.2 CSS 层叠包含框嵌套关系

层叠包含框也可以称为层叠根元素，包含在层叠根元素内的其他定位元素都将以该元素作为参考平台进行层叠定位。当然这些定位元素的 z-index 属性值应该为一个非 auto 的值。通俗说，在同一个层叠根元素内的所有定位元素都会使用相同的规则来决定层叠顺序。

如果所有定位元素的层叠包含框（Stacking Context）一样时，这些定位元素就以 z-index 属性值来决定层叠显示顺序。

如果 z-index 值相同（专业讲就是 Stack Level 相同），则按照文档流中的结构排列顺序进行定位，位于文档流中后面的定位元素会覆盖文档流前面的定位元素。例如，对于上面的示例，如果黄盒子与绿盒子的 z-index 值都为 10，由于在 HTML 结构中绿盒子位于黄盒子的后面，所以它将覆盖黄盒子。

但是如果定位元素位于不同的层叠关系之中时，就不能够简单地使用自身的 z-index 属性来决定层叠顺序，而是应该根据层叠根元素的 z-index 属性来决定层叠顺序，这就是 CSS 层叠包含框嵌套问题。

例如，针对如下的结构，我们来试验层叠包含框发生重叠之后会发生什么情况。

```

<div id="wrap">父级层叠根元素
  <div id="container">子级层叠根元素
    <div id="box1">定位元素 1</div>
  </div>
  <div id="box2">定位元素 2</div>
</div>

```

如果我们定义它们都为绝对定位显示，且定义<div id="wrap">和<div id="container">都为层叠包含框，详细样式代码如下：

```

<style type="text/css">
div { /* 公共样式，定义定位元素 */
  position: absolute; /* 绝对定位 */
  top: 20px; /* 顶部距离 */
  left: 30px; /* 左侧距离 */
  width: 80%; /* 百分比宽度 */
  height: 80%; /* 百分比高度 */
}
#wrap { /* 外层叠包含框 */
  border: dashed 1px blue; /* 虚线框 */
}

```



```

width:400px;          /* 固定宽度 */
height:200px;         /* 固定高度 */
z-index: 0;           /* 层叠值 */
}
#container { /* 内层叠包含框 */
border:solid 2px red;  /* 实线框 */
z-index: 0;           /* 层叠值 */
}
#box1 { /* 定位黄盒子 */
left: 60px;           /* 左侧巨鹿 */
background-color: yellow; /* 黄色背景 */
z-index: 20;          /* 层叠值 */
}
#box2 { /* 定位绿盒子 */
top: 60px;            /* 顶部距离 */
left: 120px;          /* 左侧距离 */
background-color: green; /* 绿色背景 */
z-index: 10;          /* 层叠值 */
}
</style>

```

如果在浏览器中预览，则会发现层叠值小的绿色盒子覆盖了层叠值大的盒子（如图 9.56 所示）。这是因为绿盒子将与黄盒子的层叠包含框<div id="container">对象（或者说根元素）位于同一个平台，而黄盒子则以<div id="container">层叠包含框作为 z 轴定位的平台，所以两者无法进行比较，而是根据绿盒子与黄盒子的层叠包含框<div id="container">对象的层叠值进行比较，而绿色盒子的层叠值为 10，而<div id="container">对象的层叠值为 0，因此绿色盒子会覆盖<div id="container">对象，同时也会覆盖该对象包含的所有子对象，这时不管子对象的层叠值有多大，都会被覆盖。

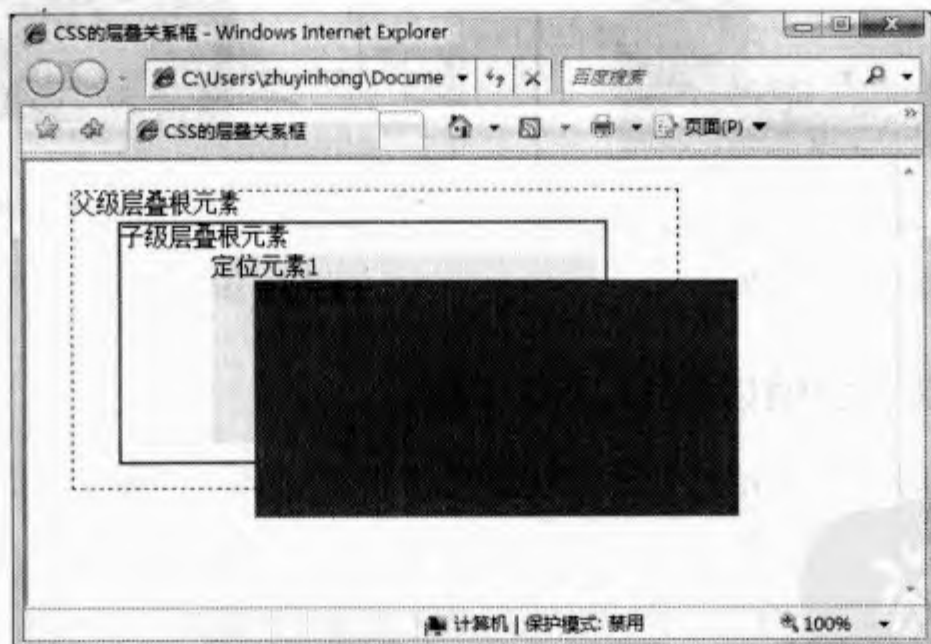


图 9.56 CSS 层叠嵌套关系图

9.4.3 IE 在解析 CSS 层叠关系时的 Bug

层叠包含框以及层叠关系对于 CSS 定位布局来说是两个重要的概念。但是 IE 却错误地认为所有元素都可以作为层叠包含框（即层叠根元素），而不管这些定位元素的层叠值是否为 auto（自动）或者数值。

例如，针对上面的示例，如果我们把<div id="container">对象样式中的 z-index:0 声明删掉

之后, 则<div id="container">对象就不再是层叠包含框了, 此时黄色盒子与绿色盒子就会处于同一个层叠包含框内, 也就是说它们位于同一个平台上进行比较, 层叠值大的元素将会排在上面, 因此黄盒子将覆盖绿盒子显示在上面 (如图 9.56 所示)。

```
#container { /* 清除层叠包含框 */
    border:solid 2px red;           /* 实线框 */
    z-index: 0;                    /* 层叠值 */
}
```

根据 CSS 规则, 除了根元素 (html), 只有定位元素的 z-index 被定义一个非 auto 的 z-index 层叠值时才能产生新的层叠包含框。但是你会发现如果在 IE 浏览器中预览, 则层叠值比较低的绿盒子会覆盖层叠值高的黄盒子 (如图 9.58 所示)。

<div id="container">对象的 z-index 属性为默认值 auto, 按理讲它不会影响子元素的层叠顺序, 但是这时 IE 仍然把<div id="container">对象视为层叠包含框。这是因为 IE 浏览器认为所有定位元素都会产生一个新的层叠包含框, 并且从 z-index 的值为 0 开始。

解决此类问题唯一的方法是, 通过调整层叠根元素的 z-index 属性值来决定它们的排列顺序。例如, 在上面的示例中如果我们希望黄盒子覆盖绿盒子, 则可以定义黄盒子的层叠根元素的层叠置大于绿盒子的层叠值, 这样才能够显示图 9.57 所示的效果。

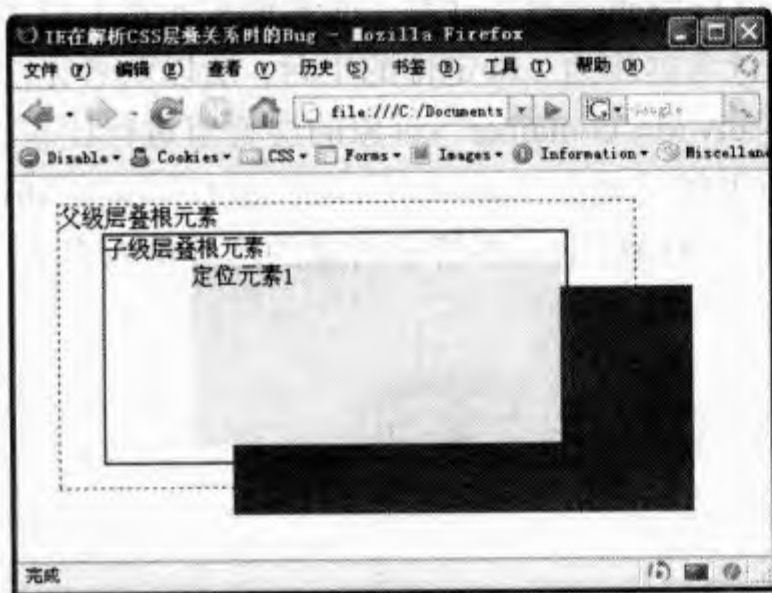


图 9.57 FF2 中预览结果

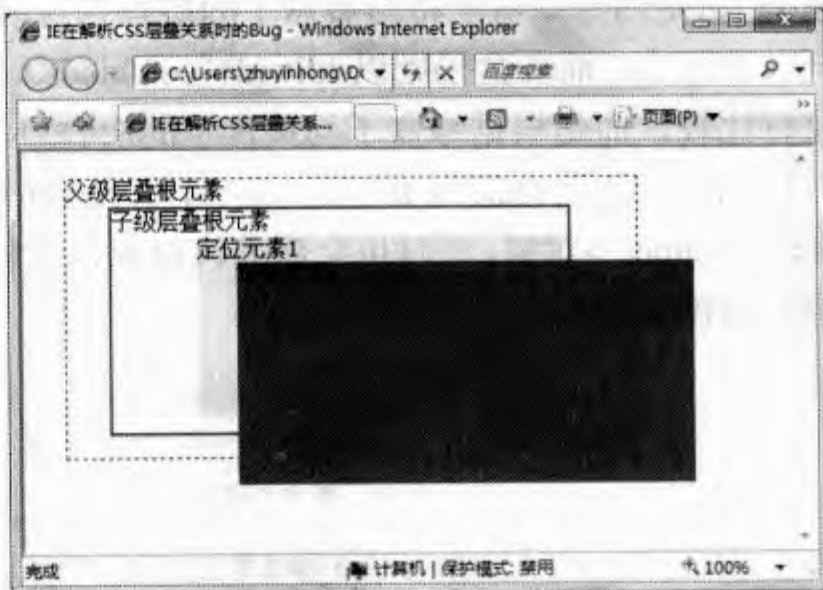


图 9.58 IE 7 中预览结果

9.4.4 探析 CSS 层叠负值以及 IE 存在 Bug

当 z-index 取负值时, 它会定位元素到层叠包含框的底部, 隐藏在默认显示的元素的下。例如, 在下面这个结构中:

```
<div id="wrap">层叠根元素
  <div id="box1">定位元素 1</div>
  <div id="box2">定位元素 2</div>
  <div id="box3">流动元素</div>
</div>
```

我们定义<div id="wrap">对象为层叠包含框, 然后定义<div id="box1">和<div id="box2">为定位显示, <div id="box3">为自然流动显示。其中<div id="box1">层叠值为正值, 而<div id="box2">层叠值为负值。详细样式表如下:

```
<style type="text/css">
#wrap { /* 定义层叠包含框 */
```



```

position:absolute;                /* 绝对定位 */
border:dashed 1px blue;           /* 虚线框 */
width:400px;                      /* 固定宽度 */
height:200px;                    /* 固定高度 */
z-index: 0;                      /* 层叠值 */
}
#box1 { /* 定位黄盒子 */
position:absolute;                /* 绝对定位 */
top:60px;                        /* 顶部距离 */
left: 60px;                      /* 左侧距离 */
width:160px;                     /* 固定宽度 */
height:100px;                   /* 固定高度 */
background-color: yellow;        /* 黄色背景 */
z-index: 1;                      /* 层叠值 */
}
#box2 { /* 定位绿盒子 */
position:absolute;                /* 绝对定位 */
top: 80px;                      /* 顶部距离 */
left: 160px;                    /* 左侧距离 */
width:160px;                     /* 固定宽度 */
height:100px;                   /* 固定高度 */
background-color: green;         /* 绿色背景 */
z-index: -1;                    /* 层叠值 */
}
#box3 { /* 普通流动的盒子 */
width:340px;                    /* 固定宽度 */
height:120px;                  /* 固定高度 */
background-color:#FF00FF;        /* 粉红色背景 */
}
</style>

```

那么这个层叠值为负值的绿盒子会在<div id="wrap">层叠框的内部隐藏到底部（如图 9.59 所示）。请注意，这个绿盒子不会被隐藏到<div id="wrap">层叠框的底部，也就是说如果<div id="wrap">被定义了背景色，则它包含的所有定位元素，不管其层叠值负值为多小，最终都会显示在层叠框的上面。一般来说层叠值越小，绿盒子越排在下面。

但是如果我们定义层叠包含框的层叠值也为负值，这时就会发现在非 IE 标准浏览器中看不到任何对象内容，也就是网页中所有内容消失了。如果我们清除层叠包含框的层叠值，让其默认显示为 auto，则这时会发现绿色盒子突然不见了，而虚线框和其他元素依然显示在页面中（如图 9.60 所示）。

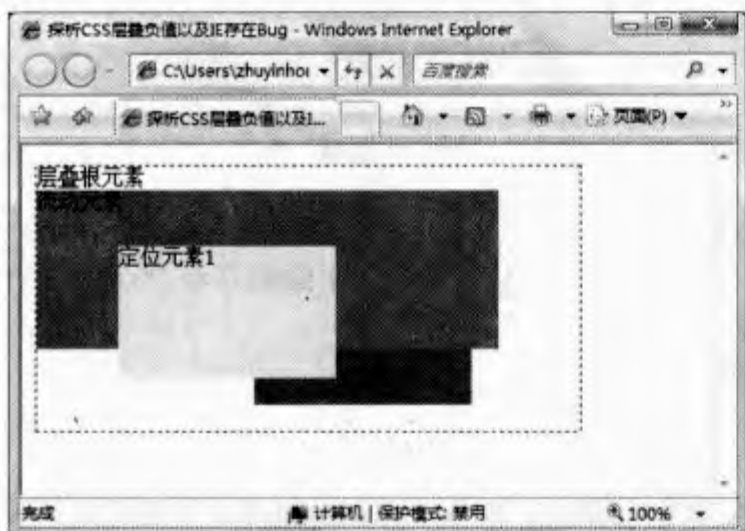


图 9.59 负值层叠显示效果

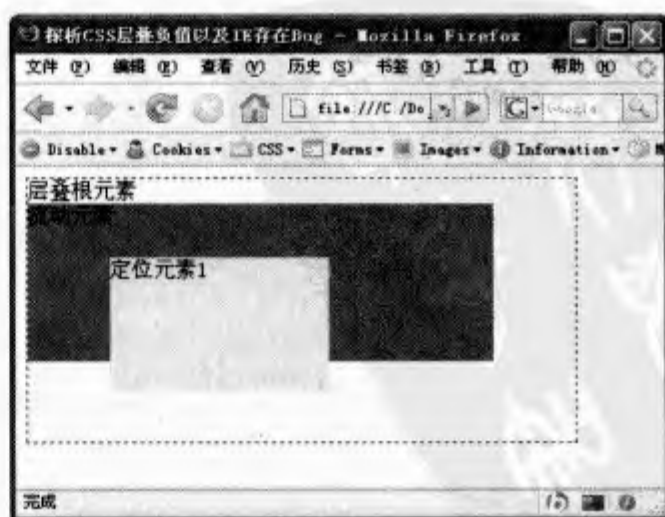


图 9.60 在 FF 2 中负值层叠显示效果

```
#wrap { /* 层叠包含框为负值 */
    z-index: -1;
} /* 层叠值 */
```

这是为什么呢？

原来，在标准浏览器中，一般都认为 HTML 的根元素 html 是一个固定层叠框，而 body 则不是层叠包含框。当元素取负值时，它就被隐藏在 body 元素的下面，所以看不到任何对象了。此时如果我们定义 body 元素为层叠包含框（代码如下），则就会发现所有对象都浮出水面了，原来它们并没有消失，而是被隐藏在 body 下面了。

```
body { /* 定义层叠包含框 */
    position: relative;
    z-index: 0;
} /* 相对定位 */
/* 层叠值 */
```

但是，IE 浏览器似乎很奇怪，不管怎么定义，最终所有元素都会显示页面中，而不会出现对象被隐藏在 body 下面的情况。例如，如果我们清除<div id="wrap">包含框的 position: absolute 和 z-index: 声明，也就是说清除该层叠包含框，根据 CSS 标准规则，被定义为负值的绿色盒子会消失，但实际上它仍然显示在 IE 浏览器中。因此我们可以这样理解，IE 浏览器默认 body 元素为层叠包含框，即 body 元素默认为一个相对定位（position: relative）的元素。

9.5 IE 浏览器常见 Bug 及其解决方法

上面 3 节分别从几个方面重点讲解了 CSS 兼容性布局中的难点。在第 9.1 节中我们曾就 IE 独有的 IE 条件语句进行了详细介绍，在第 9.2 节探讨了 IE 的解析模式，并由此产生的各种 Bug。下面我们在这个基础上介绍 CSS 布局中经常会遇到的 IE Bug。当然不是说其他浏览器没有 Bug，主要是因为 IE 的非标准模式所形成的 Bug 比较普遍，也比较严重。在前面章节中我们曾经介绍了不少 IE 的 Bug，对于已经讲解的 Bug，本节就不再重复。

9.5.1 超链接设计中存在的 Bug

由于 a 是行内元素，不具备 Layout 布局特性，但是很多时候我们又希望它能够具备布局功能，于是就出现下面这样的 Bug。

在这个示例中我们希望鼠标移到超链接上时能够显示箭头图像，以指示操作。当鼠标移过之后，又能够自动隐藏图标。

```
<style type="text/css">
a { /* 超链接的样式 */
    text-decoration: none;
} /* 清除下划线 */
a img { /* 超链接包含的图片样式 */
    display: none;
    border: none;
} /* 正常情况下隐藏显示 */
/* 清除边框 */
a: hover img { /* 鼠标经过时，包含图片的样式 */
    display: inline;
} /* 行内显示 */
</style>
<a href="#">菜单项目</a>
```


如果我们在 IE 7 或者其他标准浏览器中预览,则会正常显示(如图 9.61 所示),但是在 IE 6 或者以下版本中不显示图像(如图 9.62 所示)。

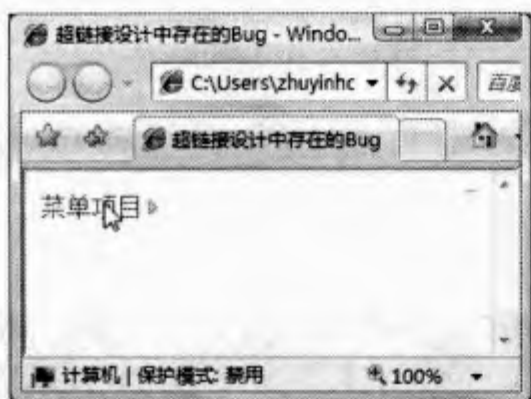


图 9.61 IE 7 中预览结果



图 9.62 IE 6 中预览结果

解决方法:这是因为超链接的 hover 伪类不具有 Layout 特性,我们可以通过定义 CSS 属性来激活它的 Layout 特性,例如,定义下面的样式即可激活该伪类的 Layout 特性。

```
a:hover { /* 激活伪类的 Layout 特性 */
    zoom:1; /* 缩放比例 */
}
```

另外,还可以使用一些 CSS 属性激活 Layout 特性,如 border、display、position、overflow、background 等。

9.5.2 IE 元素的内容与背景分离显示的 Bug

人们可能难以相信:一个元素的内容和它的背景色被肢解显示。这可能吗?确实存在这样的问题,请输入下面示例代码,然后在 IE 浏览器中预览(如图 9.63 所示)。

可以看到,蓝色背景的内容框居然被相邻的红色背景给分离了,形成一种奇怪的元素内容与背景分解的现象。但是如果在非 IE 浏览器中预览则没有这种现象(如图 9.64 所示)。

```
<style type="text/css">
#main { /* 层叠包含框 */
    position:relative; /* 相对定位 */
    z-index:1; /* 层叠值 */
    color:white; /* 白色字体 */
}
#background { /* 背景色 */
    position:absolute; /* 绝对定位 */
    left:10em; /* 左侧距离 */
    top:0; /* 顶部距离 */
    z-index:-1; /* 层叠值为-1 */
    width:100%; /* 百分比宽度 */
    background:red; /* 红色背景 */
    height:60px; /* 固定高度 */
}
#content { /* 内容包含框 */
    background:blue; /* 蓝色背景 */
}
</style>
<div id="main">
    <div id="background"></div>
    <div id="content">
```

```
<p><span>CSS Zen Garden (样式表禅意花园) 邀请您发挥自己的想象力, 构思一个专业级的网
页。让我们用慧眼来审视, 充满理想和激情去学习 CSS 这个不朽的技术, 最终使自己能够达到技术
和艺术合而为一的最高境界。</span></p>
```

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

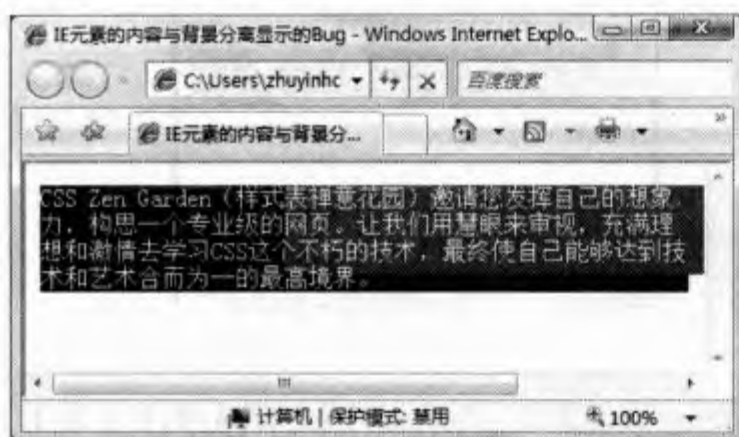


图 9.63 IE 存在内容和背景被分开显示的 Bug

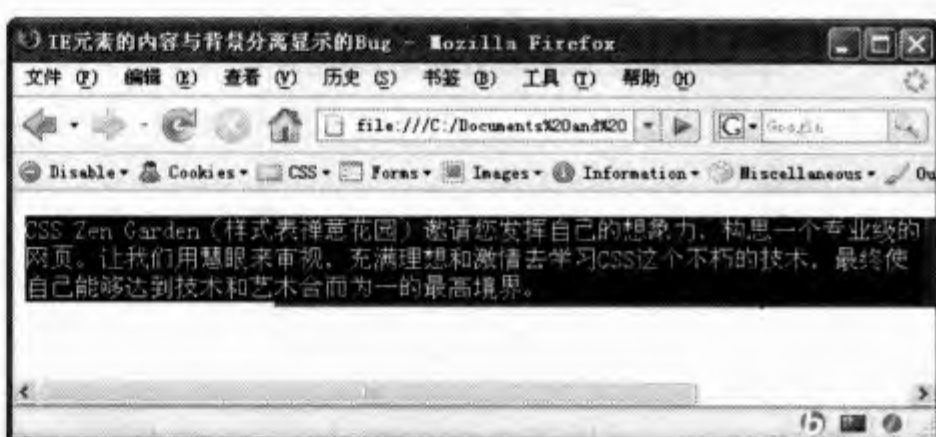


图 9.64 FF 不存在内容和背景被分开显示的现象

本来我们希望通过这种方法来设计一个可变背景的页面。首先定义外层包含框为层叠包含框, 然后把<div id="background">元素作为一个道具用来定义动态背景色, 再通过层叠值取负值, 使其隐藏在内容框的下面, 这样需要时可以随时调用。

但是应注意到, 在 FF 中红色背景层并没有隐藏到蓝色内容层下面, 而是插在内容包含框的蓝色背景和显示的文字之间。

这是 IE 的一个 Bug。解决这个问题很简单, 通过定义一个 CSS 样式触发<div id="background">元素的 Layout 特性。例如:

```
#content { /* 触发 Layout 特性 */
    zoom:1; /* 缩放比例 */
}
```

当然还可以定义其他样式, 如高度或是宽度等来触发该元素的 Layout 特性。

9.5.3 IE 6 躲躲猫 Bug

躲躲猫是一个捉迷藏游戏, 在英文中翻译为 Peekaboo。IE 6 布局中就存在这样一个躲躲猫的 Bug, 能够使页面内容时隐时现, 显示得非常不稳定。

例如, 在下面这个简单的示例中, 当在 IE 6 中打开网页时右侧超链接会隐藏起来(如图 9.65 所示), 当鼠标移过浮动超链接, 或者重新调整窗口大小, 右侧的超链接才被显示出来(如图 9.66 所示)。另外, 当右侧超链接显示时, 如果鼠标移过右侧超链接, 则会自动隐藏部分超链接内容。

```
<style type="text/css">
#wrap { /* 包含框样式 */
    background:#FFFFCC; /* 背景色 */
    border: 1px solid red; /* 边框线 */
}
#wrap a:hover {
    background:#CCFF66; /* 鼠标移过浮动超链接样式 */
}
#float { /* 浮动元素样式 */
    float : left; /* 向左浮动 */
    border: 1px solid green; /* 边框线 */
    width : 120px; /* 固定宽度 */
}
```



```

height: 150px;                /* 固定高度 */
}
.clear { /* 清除类 */
clear: both;                  /* 清除左右浮动 */
}
</style>
<div id="wrap">
  <div id="float"><a href="#">浮动超链接</a></div>
  <div><a href="#">超链接</a></div>
  <div><a href="#">超链接</a></div>
  <div><a href="#">超链接</a></div>
  <div><a href="#">超链接</a></div>
  <div class="clear"> </div>
</div>

```



图 9.65 IE 6 中躲藏的超链接



图 9.66 鼠标移到超链接时会显示隐藏内容

这在 IE 7 及其他标准浏览器中是没有问题的，但是在 IE 6 版本中右侧超链接就会莫名其妙地消失了。解决 IE 6 的这个 Bug，可以采用如下方法之一：

不要为<div id="wrap">包含框定义背景色；如果必须需要背景色，则可以使用下面的方法：为<div id="wrap">包含框定义高度或者宽度，或者定义 zoom 属性，以触发该元素的 Layout 特性。

```

#wrap { /* 触发 Layout 特性 */
  zoom: 1;                /* 缩放比例 */
}

```

9.5.4 IE 6 多余字符 Bug

这是 IE 6 另一个比较典型的 Bug。当在浮动元素之间增加 HTML 注释时就很容易出现这个问题。例如，输入下面的示例代码，则在 IE 6 版本浏览器中预览如图 9.67 所示。

```

<style type="text/css">
#wrap { /* 包含框样式 */
  width: 400px                /* 固定宽度 */
}
#wrap div { /* 浮动元素样式 */
  float: left;                /* 向左浮动 */
  width: 100%                 /* 百分比宽度 */
}
</style>
<div id="wrap">
  <div>第 1 行字符</div>
  <!-- 注释 -->
  <div>↓下面是多出来的一个字符</div>
</div>

```



图 9.67 IE 6 多月字符 Bug

解决这个小 Bug 有多种方法，如清除 HTML 注释，或者不要设置 100% 宽度等。

9.6 兼容性网页布局实战

IE 浏览器让人伤透脑筋，但最伤透脑筋的还是网页布局本身。很多时候设计师会在一个半截页面模板中浪费很长时间，琢磨如何使自己设计的网页更稳健、看起来更宽容，这不是一件容易的事情。下面我们通过两个网页布局的示例，来研究如何设计兼容主流浏览器的布局方法。这两个案例都具有典型性，也是很多设计师经常遇到的问题。相信通过本节的学习，可以对设计兼容性网页进行一个很好的总结。当然，学无止境，本节内容并不代表了结束，而是继续深入学习的开始。

9.6.1 三列等高布局新法及其非 IE 存在的 Bug 和解决方法

多列等高问题一直是困扰初学者的 CSS 设计难题。所谓多列等高就是当一个多列布局的页面中，一列的高度发生了变化后，其他列也能够自动跟随变化，这对于表格布局来说可能是小菜一碟，但是对于 CSS 布局来说就有一定的难度和麻烦。在第 4.5 节中我们曾经专门讲解过比较实用的伪列布局，通过它来解决多列等高问题。下面我们将使用另一种方法来解决，即通过负边距来解决。

首先我们来看看这个页面的 HTML 结构：

```
<div id="container">
  <div id="header">
    <h1>页眉区域</h1>
  </div>
  <div id="main">
    <div id="wrap">
      <div id="mid">
        <p><strong>1.主栏</strong></p>
      </div>
      <div id="left">
        <p><strong>2.左栏</strong></p>
      </div>
      <div id="right">
        <p><strong>3.右栏</strong></p>
      </div>
    </div>
  <div id="footer">
    <p>页脚区域</p>
  </div>
</div>
```



```

</div>
</div>

```

这是一个比较经典的布局结构，整个页面被装在一个箱子里（`<div id="container">`），然后把这个箱子分成 3 个抽屉，从上到下分别是页眉区域（`<div id="header">`）、主题区域（`<div id="main">`）和页脚区域（`<div id="footer">`）。为了能够实现主栏居中、左栏居左、右栏靠右的设计目的，这里使用了一个夹层（`<div id="wrap">`），目的是实现中栏与左栏的布局位置互换。

借助这个夹层（`<div id="wrap">`），让夹层向左浮动，然后在夹层内设计主栏向右浮动，左栏向左浮动，最后设计右栏靠右对齐即可（布局示意图如图 9.68 所示）。

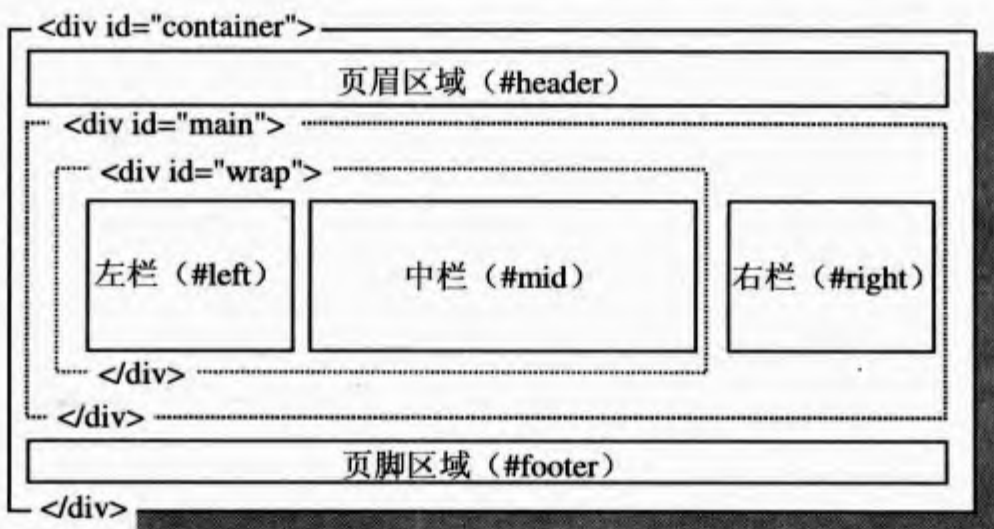


图 9.68 版式结构示意图

解决中间模块的三列布局问题后，我们来设计如何实现三列等高问题。这里不用伪列布局或者其他伪装的方法来设计，而是借助负外边距来进行设计。负外边距的布局方法，我们将在第 11 章深入讲解。

设计的思路是：设置左栏、中栏和右栏的底部内边距为一个非常大的值，强迫每个栏目无限向下延伸，这样不管三列栏目的内容是否对齐，由于它们的内边距无限大，所以栏目的背景色都自动跟随到下面，在有限的屏幕内容中给人的错觉就是每个栏目都是等高的。但是当三列底部内边距变得无限大的时候，会把页脚区域推到无穷低的地方，相当于隐藏了该页脚区域（如图 9.69 所示）。这时如果要预览页脚区域，需要拖动滚动条很长时间才能够看到底部区域。



图 9.69 版式结构示意图

不过我们再设置底部外边距为一个无穷大的负值，该无穷大的负值与所设置的无穷大的底部内边距正值相同，这样就等于把多出的内边距给隐藏起来。然后我们再定义中间包含框隐藏多出的内容(#main {overflow: hidden;})，这样当某个栏目的内容很多时，会自动撑开包含框(<div id="main">); 而当包含框被撑开之后，其他两个栏目的高度也会随着伸展，当然不是它们的height，而是它的背景色（因为它们的内边距是无穷大的）在自动跟随伸展。其实它们早已经被撑开了，只因为被包含框给隐藏起来了，现在包含框被撑开之后，这些被隐藏的栏目也随着显露出来。

大致明白了设计思路，下面我们就来详细分解样式代码。样式代码有点长，不要被它们吓退了，大部分代码是修饰性或辅助性，其中核心代码被加粗显示出来：

```
body { /* 网页基本属性 */
    margin:0; /* 清除页边距 */
    padding:0; /* 清除页边距 */
    font: 76% arial, sans-serif; /* 字体大小 */
    text-align:center; /* 网页居中 */
}
p { /* 段落样式 */
    margin:10px; /* 文本段落外边距 */
}
a { /* 超链接样式 */
    display:block; /* 超链接块状显示 */
    color: #006; /* 超链接字体颜色 */
    padding:10px; /* 超链接内边距 */
}
#container { /* 网页包含框样式 */
    margin:0 auto; /* 网页居中显示 */
    text-align:left; /* 文本左对齐 */
    width:778px; /* 网页宽度 */
}
div#header h1 { /* 页眉区域样式 */
    height:80px; /* 页眉区域高度 */
    line-height:80px; /* 垂直居中 */
    margin:0; /* 外边距 */
    padding-left:10px; /* 左侧内边距 */
    background: #EEE; /* 页眉区域背景色 */
    color: #79B30B; /* 页眉区域字体颜色 */
}
div#footer { /* 页脚区域样式 */
    background: #333; /* 页脚区域背景色 */
    color: #FFF; /* 页脚字体颜色 */
    height:30px; /* 页脚区域高度 */
    clear: both; /* 清除两侧浮动 */
}
#main { /* 主体包含框样式 */
    overflow: hidden; /* 隐藏多余的空间 */
}
#wrap { /* 左侧包含框样式 */
    float: left; /* 向左浮动 */
    width: 628px; /* 左侧包含框宽度 */
}
#left { /* 左栏样式 */
    float: left; /* 向左浮动 */
    margin: 0; /* 外边距为 0 */
}
```



```
width: 140px; /* 左栏宽度 */
background: #B9CAFF; /* 左栏背景色 */
}
#mid { /* 中栏样式 */
float: right; /* 向右浮动 */
margin-left: 10px; /* 左侧外边距 */
width: 478px; /* 中栏宽度 */
}
#wrap, #left, #mid, #right { /* 核心样式, 栏目的无穷大内边距和无穷大负外边距 */
padding-bottom: 9999px; /* 底部无穷大正内边距 */
margin-bottom: -9999px; /* 底部无穷大负外边距 */
}
#right { /* 右栏样式 */
float: right; /* 向右浮动 */
margin-left: 10px; /* 栏目左侧外边距 */
width: 140px; /* 右栏宽度 */
background: #FF8539; /* 右栏背景色 */
}
```

该布局所显示的效果在 IE 中的显示效果如图 9.70 所示, 但是在非 IE 浏览器中预览则会发现存在一个 Bug, 如图 9.71 所示。

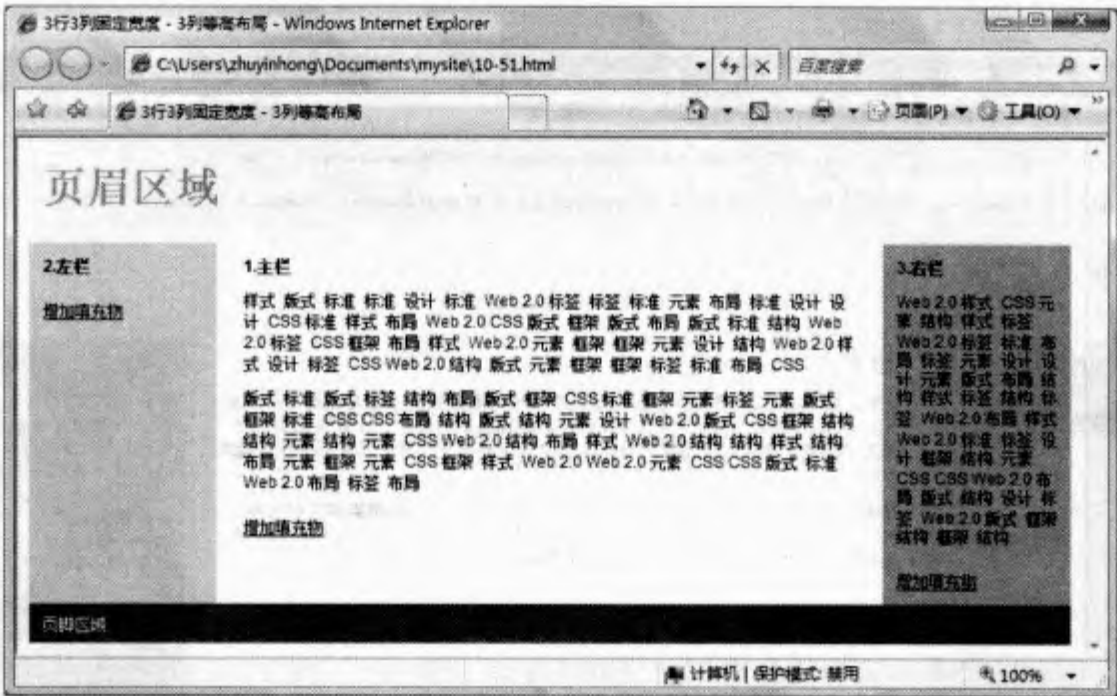


图 9.70 IE 7 中多列等高布局效果

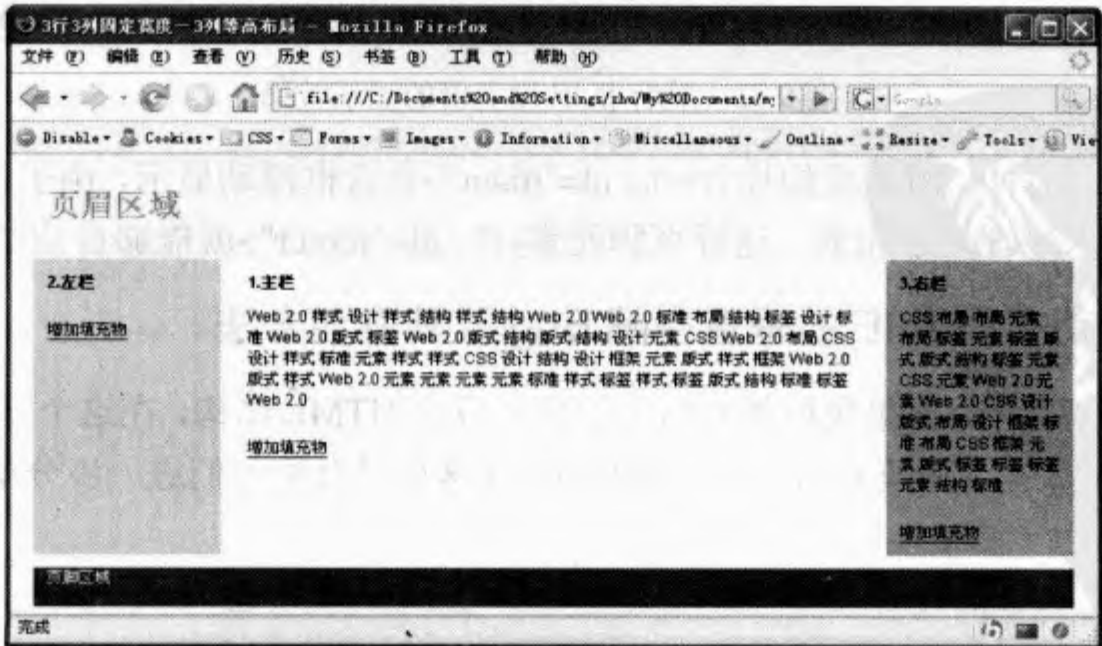


图 9.71 FF 2.0 中多列等高布局效果

下面我们就来研究这个非 IE 浏览器存在的 Bug 如何解决。产生这个 Bug 的原因是因为超大边距值。这种超大边距对于浏览器来说是一个极限考验，如果把其中的内边距和外边距设置为 99999px，则在 IE 或其他浏览器中会呈现为不同的效果。

另外，如果把下面核心样式代码放在样式表的前面，会发现浏览器的解析效果变化很大，其原因也是因为浏览器在解析超大边距值所存在的 Bug。

```
#wrap, #left, #mid, #right { /* 核心样式，栏目的无穷大内边距和无穷大负外边距 */
    padding-bottom: 99999px; /* 底部无穷大正内边距 */
    margin-bottom: -99999px; /* 底部无穷大负外边距 */
}
```

解决本示例在非 IE 中存在的这个 Bug，不妨从页脚模块入手研究。还记得负外边距的使用吗？既然非 IE 浏览器在解析该布局时总会闪出一条空隙，那么我们何不设置页脚区域<div id="footer">的顶部外边距为一个负值呢？这样通过强迫上移页脚区域上边距来掩盖这条空隙。补丁样式如下，这里使用了一个兼容技术，设计只在非 IE 浏览器下设置上边外边距的取值为负，避免在 IE 中上移而覆盖主要栏目的内容，效果如图 9.72 所示。

```
html>/**/body div#footer { /* 兼容非 IE 浏览器 */
    margin-top: -10px; /* 底部负外边距 */
}
```



图 9.72 兼容非 IE 浏览器的显示效果

还有一种解决方法，就是我们设置<div id="main">包含框浮动显示。由于该元素浮动显示，则它会自动收缩以包含其子元素，这样页脚元素<div id="footer">就能够自动贴近该浮动元素。

9.6.2 三列浮动布局中 IE 的垂直空隙 Bug 及其兼容方法

在上节示例基础上，如果我们使用两层结构来设计 HTML 结构，在这个结构整个页面包含在一个箱子里，然后在其中垂直排列了 5 个模块。在这里没有给它们进一步分层，以简化 HTML 结构设计。

```
<div id="container">
    <div id="header">
        <h1>页眉区域</h1>
```



```

</div>
<div id="left">
  <p><strong>2.左栏</strong></p>
</div>
<div id="right">
  <p><strong>3.右栏</strong></p>
</div>
<div id="mid">
  <p><strong>1.主栏</strong></p>
</div>
<br class="clear" />
<div id="footer">
  <p>页脚区域</p>
</div>
</div>

```

这种设计方法实际上也是 Dreamweaver CS3 默认的设计方法。当然为了设计高效的网页还需要调整原来示例中的结构，让左右两栏排在主栏的前面。然后通过浮动法来设计 3 行 3 列的布局效果。下面的样式是该布局的核心代码：

```

<style type="text/css">
#container { /* 网页包含框样式 */
  width: 780px; /* 网页宽度 */
  margin: 0 auto; /* 网页居中 */
  text-align: left; /* 文本左对齐 */
}
div#header h1 { /* 页眉标题样式 */
  height: 80px; /* 页眉标题高度 */
  line-height: 80px; /* 标题垂直对齐 */
  margin: 0; /* 清除外边距 */
  padding-left: 10px; /* 左侧内边距 */
  background: #EEE; /* 背景色 */
  color: #79B30B; /* 字体颜色 */
}
#left { /* 左栏样式 */
  float: left; /* 向左浮动 */
  width: 150px; /* 固定宽度 */
  background: #B9CAFF; /* 左栏背景色 */
  padding: 15px 10px 15px 20px; /* 增加左栏内边距 */
}
#right { /* 右栏样式 */
  float: right; /* 向右浮动 */
  width: 160px; /* 固定宽度 */
  background: #FF8539; /* 右栏背景色 */
  padding: 15px 10px 15px 20px; /* 增加右栏内边距 */
}
#mid { /* 中栏样式 */
  margin: 0 200px; /* 左右外边距，腾出空间为左右栏 */
  padding: 0 10px; /* 左栏背景色 */
}
div#footer { /* 页脚区域样式 */
  background: #333; /* 背景色 */
  color: #FFF; /* 字体颜色 */
  height: 30px; /* 固定高度 */
  clear: both; /* 清除左右浮动 */
}
.clear { /* 清除类 */

```

```
clear:both;
}
</style>
```

/* 清除左右浮动 */

但是我们在不同浏览器中预览时，会发现一个问题：在 IE 和非 IE 中所预览的效果略有区别。主要区别在左右栏内容与页眉区域的距离与中栏的距离略有不同（如图 9.73、图 9.74 所示）。产生这个细微的差别是因为不同浏览器在解析这种布局时存在合理的误差。

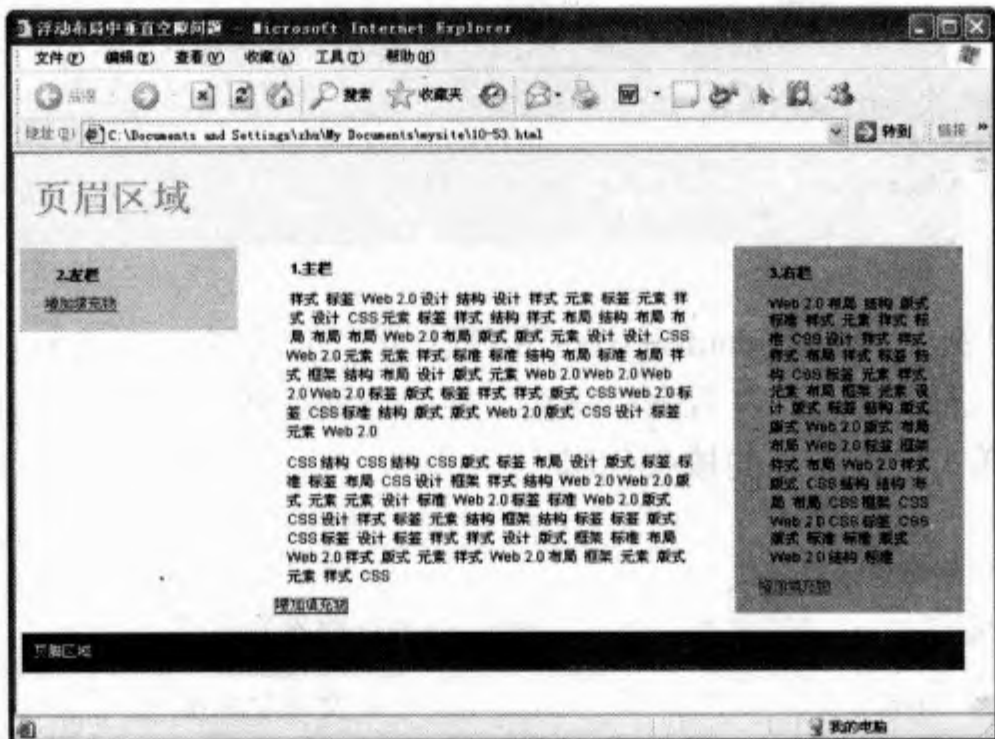


图 9.73 IE 7 中多列等高布局效果

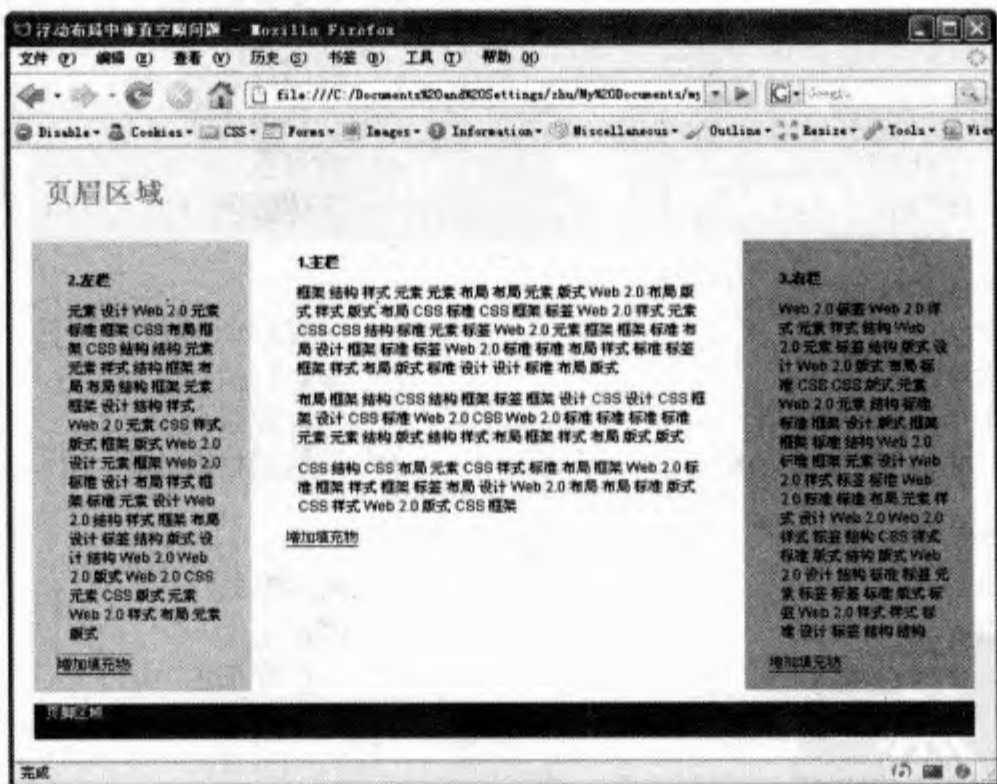


图 9.74 FF 2 中多列等高布局效果

在这里我们可以利用 IE 条件语句来微调 IE 中的顶部距离，兼容代码如下：

```
<!--[if IE]>
<style type="text/css">
#right, #left { padding-top: 30px; } /* 微调 IE 浏览器中顶部距离 */
#mid { zoom: 1; } /* 触发中栏包含框的 Layout 布局特性，以便自动调整自身的距离 */
</style>
<![endif]-->
```


网页结构化布局与实施

在第 7 章中我们曾经就网页的结构化问题进行过探讨，介绍了构建符合标准的网页结构应该是什么样子的。也许经历过传统网页布局的实践之后，你一定很疑惑：我是为了布局而构建结构，还是为了结构而设计布局？不管怎么抉择，相信这样的烦恼一定会深深地烙记于你的脑海深处。本章将在前面章节讲解的布局技术基础上，进一步强化网页设计中结构和布局之间的关系，以及如何处理好它们的关系，通过不同类型的典型示例让你深切体验到 CSS 布局的强大魅力。

10.1 设计网页基本结构

很多设计师徜徉于禅意花园 (<http://www.csszengarden.com/>) 的 CSS 设计之美时，也许最大的收获就是领悟到网页结构与网页版式的关系，而不是它提供的漂亮作品或者符合标准的结构。

是的，如果想一想世界各地的优秀设计师能够根据禅意花园这个简单的结构设计出无数布局精美的作品，你也就不难理解对于一个设计科学、合理的结构，使用 CSS 都可以设计出版式各异的页面效果。那么网页结构与网页版式到底有什么关系呢？网页结构有什么设计技巧吗？本节将就这个话题进行详细讲解。

10.1.1 网页基本结构的设计思路

传统网页设计中设计师工作的第一步大概是构图、绘图，然后是切图。也许你也有过这样的设计习惯：先把网页样图或者草图绘制出来，再构思如何来实现这样的效果。其实很多设计师一直都沿袭着这种既定的设计模式。

现在该打破这样的设计框框了，当着手准备设计新的页面时，不要被什么样图或者效果图所羁绊，正确设计的第一步应该思考：我的网页结构该怎么规划呢？

虽然说网页设计是一种视觉语言，要讲究编排和布局，为了达到最佳的视觉表现效果，必须讲究整体布局的合理性，使浏览者有一个流畅的视觉体验。但是在标准设计思想的指导下，版式问题显得不是那么重要了，而设计科学、合理的结构才是网页设计的第一步，也是最重要的一步。

例如，也许你经常面临要构建一个包含页眉（网页标题）、导航栏、主要内容（网页正文）、其他栏目和页脚（网页版权信息）5 大板块的页面（如图 10.1 所示），那么该如何规划这个结构呢？

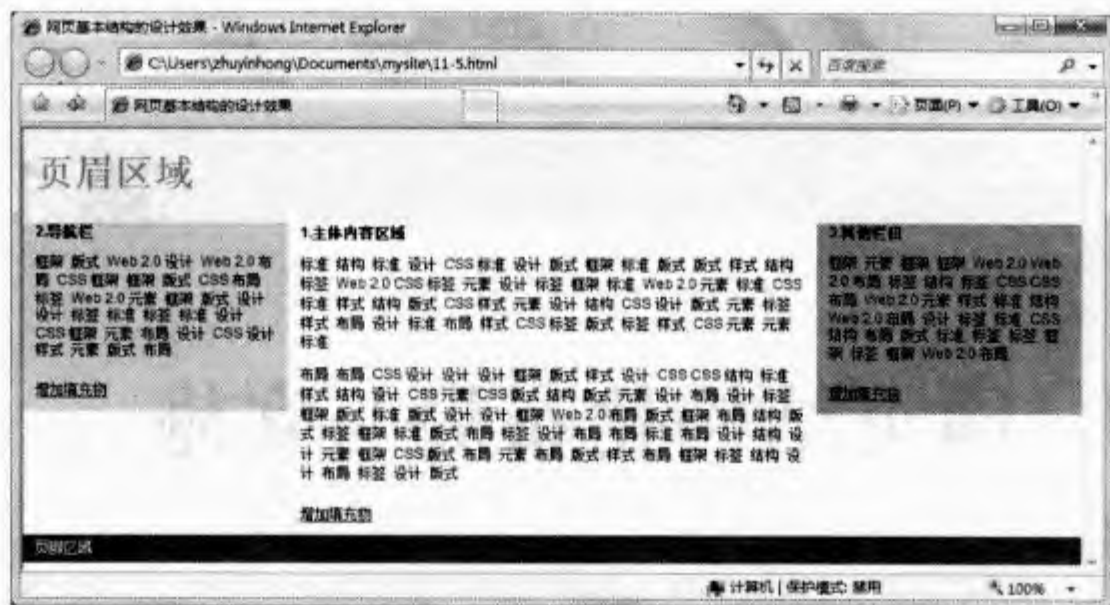


图 10.1 网页基本结构设计效果图

初学者很容易陷入这样的思维定式：根据版式效果来设计网页结构，遵循怎样 3 列 3 行的布局模式，或者液态布局类型等。甚至还会思考，使用什么的结构更能够实现这样的设计效果等诸如此类的问题。

实际上，现在完全不用去考虑网页的版式效果如何，也不要被现有的各种版式效果所影响，甚至迷惑。禅意花园的作者在设计禅意花园的网页结构时，没有想到所要实现的网页效果，更没有想到会有那么多设计精美的版式，这是难以预料的，但是禅意花园的作者却能够预料到什么样的结构才是符合标准的，是最科学、最合理的；有了这个灯塔似的经典范例，初学者应该想一想：我设计的网页结构科学、合理吗？它符合标准吗？

当然也不要被什么科学、合理的网页结构这样的大帽子所吓倒，并因此缩手缩脚而不知如何下手。首先，我们不妨先把包含这些结构的基本框架给搭建起来，犹如建筑中的打桩。

```
<div id="header">页眉区域</div>
<div id="navigation">导航栏</div>
<div id="content">主体内容区域</div>
<div id="extra">其他栏目</div>
<div id="footer">页脚区域</div>
```

可以看到，整个页面共有 5 大部分，分别使用 5 个包含框来表示。对于这样简单的结构，不用担心这样的结构不能够设计出预想的效果，相信自己完全可以使用 CSS 来控制版式的呈现效果。

但是在实际开发中，这样的结构可能又显得有点单薄。不是说这种结构不好，而是要根据需要适当进行调整。这种松散式的结构存在最大的难题就是编写 CSS 时会很麻烦，因为需要重复为每个模块的显示进行控制。也许对于一个综合性很强、包含内容很丰富的页面，把结构分散设计会更容易管理。即使多写 CSS 代码，对于整体设计来说也就不算什么。

根据更科学、更合理的要求，我们不妨把这个结构作进一步的规划。规划的思路是根据网页实际内容和需要进行设计。

如果网页主体内容的版式具有很大的相似性或者聚合性，则不妨把网页主体内容放在一个包含框中。结构如下：

```
<div id="header">页眉区域</div>
<div id="main">
  <div id="navigation">导航栏</div>
  <div id="content">主体内容区域</div>
```



```

    <div id="extra">其他栏目</div>
</div>
<div id="footer">页脚区域</div>

```

如果导航栏与网页标题结合紧密，也可以这样规划：

```

<div id="header">
    <h1>网页标题</h1>
    <div id="navigation">导航栏</div>
</div>
<div id="main">
    <div id="content">主体内容区域</div>
    <div id="extra">其他栏目</div>
</div>
<div id="footer">页脚区域</div>

```

或者

```

<div id="header">
    <h1>网页标题</h1>
    <div id="navigation">导航栏</div>
</div>
<div id="main">
    <div id="content">主体内容区域</div>
</div>
<div id="extra">其他栏目</div>
<div id="footer">页脚区域</div>

```

当然如果页脚区域与主体区域关系紧密，也完全可以把页眉也放入到主体结构中：

```

<div id="header">
    <h1>网页标题</h1>
    <div id="navigation">导航栏</div>
</div>
<div id="main">
    <div id="content">主体内容区域</div>
    <div id="extra">其他栏目</div>
    <div id="footer">页脚区域</div>
</div>

```

不过，从实践摸索中设计师发现，如果为整个网页设计一个外套，对于使用 CSS 进行网页控制和管理会非常方便。结构如下：

```

<div id="container">
    <div id="header">页眉区域</div>
    <div id="navigation">导航栏</div>
    <div id="wrapper">
        <div id="content">主体内容区域</div>
    </div>
    <div id="extra">其他栏目</div>
    <div id="footer">页脚区域</div>
</div>

```

Dreamweaver CS3 提供了几十个不同版式的 CSS 布局模板，包含 1 列、2 列、3 列等效果，甚至还分为固定宽度、液态宽度和百分比宽度等不同的布局类型。但是可以看到 Dreamweaver CS3 全部按着类似如下的结构来进行设计，这是一个“3 列液态，标题和脚注”版式的结构（设计效果如图 10.2 所示）。它也使用了一个网页外套，然后在其中内嵌几个平行排列的子结构组

成。

```
<div id="container">
  <div id="header">
    <h1>标题</h1>
  </div>
  <div id="sidebar1">
    <h3>Sidebar1 </h3>
  </div>
  <div id="sidebar2">
    <h3>Sidebar2 </h3>
  </div>
  <div id="mainContent">
    <h1> 主要内容 </h1>
  </div>
  <div id="footer">
    <p>脚注</p>
  </div>
</div>
```

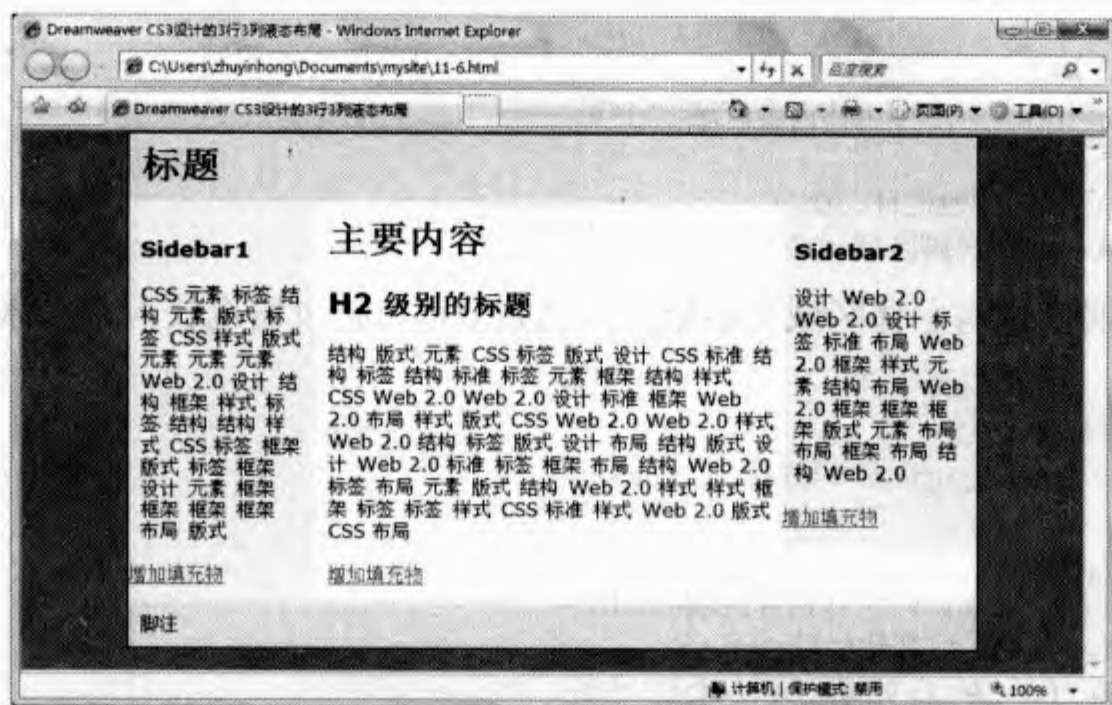


图 10.2 Dreamweaver CS3 所提供的 CSS 模板基本结构设计效果图

这是网页基本结构的一般设计思路。当然设无定法，也不必要被作者所设计的结构套住而钻进死胡同。但是在网页结构设计中一定要遵循如下规则：以内容来划定结构，而不是以版式来划定结构；结构的嵌套应更有利于管理，而不是更符合设计效果。

10.1.2 设计符合 SEO 的网页结构

网页结构不是给人看的，而是给机器读的，因此结构不要取悦人，而应讨好 SEO。

那么什么是 SEO？SEO 是 Search Engine Optimization 的简称，翻译为中文就是搜索引擎优化。通俗说就是如何让自己的网站在百度、谷歌或雅虎等搜索引擎获得较好的排名，从而赢得更多的潜在用户。

很多初学者都有这样的设计习惯：根据网页内容的显示顺序来编排网页结构的顺序，根据网页内容的显示位置来决定网页结构的嵌套层次。

但是搜索引擎更喜欢友好的网页结构，它无暇顾及网页结构所呈现出来的精美版式。为了简洁地说明这个问题，我们不妨再以上面的示例进行分析。如果根据科学性和合理性的设计原

则优化上面的网页基本结构，则网页结构显示如下：

```
<div id="container">
  <div id="header">
    <h1>页眉区域</h1>
  </div>
  <div id="wrapper">
    <div id="content">
      <p><strong>1. 主体内容区域</strong></p>
    </div>
    <div id="navigation">
      <p><strong>2. 导航栏</strong></p>
    </div>
    <div id="extra">
      <p><strong>3. 其他栏目</strong></p>
    </div>
    <div id="footer">
      <p>页脚区域</p>
    </div>
  </div>
</div>
```

我们从 SEO 的角度来分析这种结构的优势。

按着一般设计习惯，也就是按网页内容的显示顺序，网页结构的编排顺序是：页眉区域→导航栏→主体内容区域→其他栏目→页脚区域。也可以理解为“功能→功能→核心内容→功能→功能”的设计结构。

但是当搜索引擎的蜘蛛爬行时，是按着页面代码顺序自上而下来搜索的，如果当网页内容很多时，蜘蛛很难最快地爬行到核心内容区域，于是会在没有爬行到核心内容所在区域会返回，这样导致搜索所抓取的内容都是一些功能性内容，如标题、导航、分类等信息。由于这些信息在其他页面中都反复出现，搜索引擎就会认为它们是相似页面。

为了避免这样情况的发生，我们不妨把核心内容放在前面，这样更有利于被搜索引擎发现和收录。因此我们可以这样调整结构顺序：

页眉区域→主体内容区域→导航栏→其他栏目→页脚区域。

新调整的结构顺序虽然与页面呈现的效果不同，但是它更容易讨好 SEO，然后我们利用 CSS 来调整网页结构的显示顺序即可。显然这种设计思路在传统表格布局中是不可能的，也是难以想象的。

完成网页基本结构的设计之后，我们应该考虑结构的语义化问题。例如，在上面的优化结构中，网页标题使用 h1 元素，强调内容使用 strong 元素等，搜索引擎对于这些标签的关注更高。不过也尽量少使用一些搜索引擎比较疏远的内容，如 iframe、object 等元素包含的内容。在这方面禅意花园的结构绝对是最棒的学习范例，可以参阅第 7 章讲解内容进行学习。

10.2 单列版式布局

当初步理解网页结构和版式的关系之后，下面我们来系统研究网页版式的布局问题。网页设计的第一步是构建符合标准，且科学、合理的结构，第二步就应该考虑网页版式问题，要借助 CSS 技术的强大功能把网页内容固定到页面的不同显示区域。

单列版式是网页布局中最简单的一种样式，Dreamweaver CS3 在 CSS 布局模板中提供了 6

种样式：“1 列固定，居中”、“1 列固定，居中，标题和脚注”、“1 列弹性，居中”、“1 列弹性，居中，标题和脚注”、“1 列液态，居中”和“1 列液态，居中，标题和脚注”。下面我们以这些版式为基础，详细讲解单列版式布局的一般规律和设计技巧。

10.2.1 单列版式的布局结构 and 设计方法

所谓单列版式，就是网页内容呈现为一栏显示效果（如图 10.3 所示）。是不是单列版式的页面就不需要复杂的网页结构呢？不是，实际上对于任意结构的网页都可以使用 CSS 设计为单列显示。

针对 Dreamweaver CS3 所提供的 6 种单列版式，它包含了两种相似的网页结构。

第一种，仅包含网页主体内容的结构，即整个网页就有一个单独的模块：

```
<div id="container">
  <div id="mainContent">
    <h1> 主要内容 </h1>
  </div>
</div>
```

第二种，包含网页标题、主体内容和脚注（网页页脚），即包含 3 个独立的垂直排列的模块：

```
<div id="container">
  <div id="header">
    <h1>标题</h1>
  </div>
  <div id="mainContent">
    <h1> 主要内容 </h1>
  </div>
  <div id="footer">
    <p>脚注</p>
  </div>
</div>
```

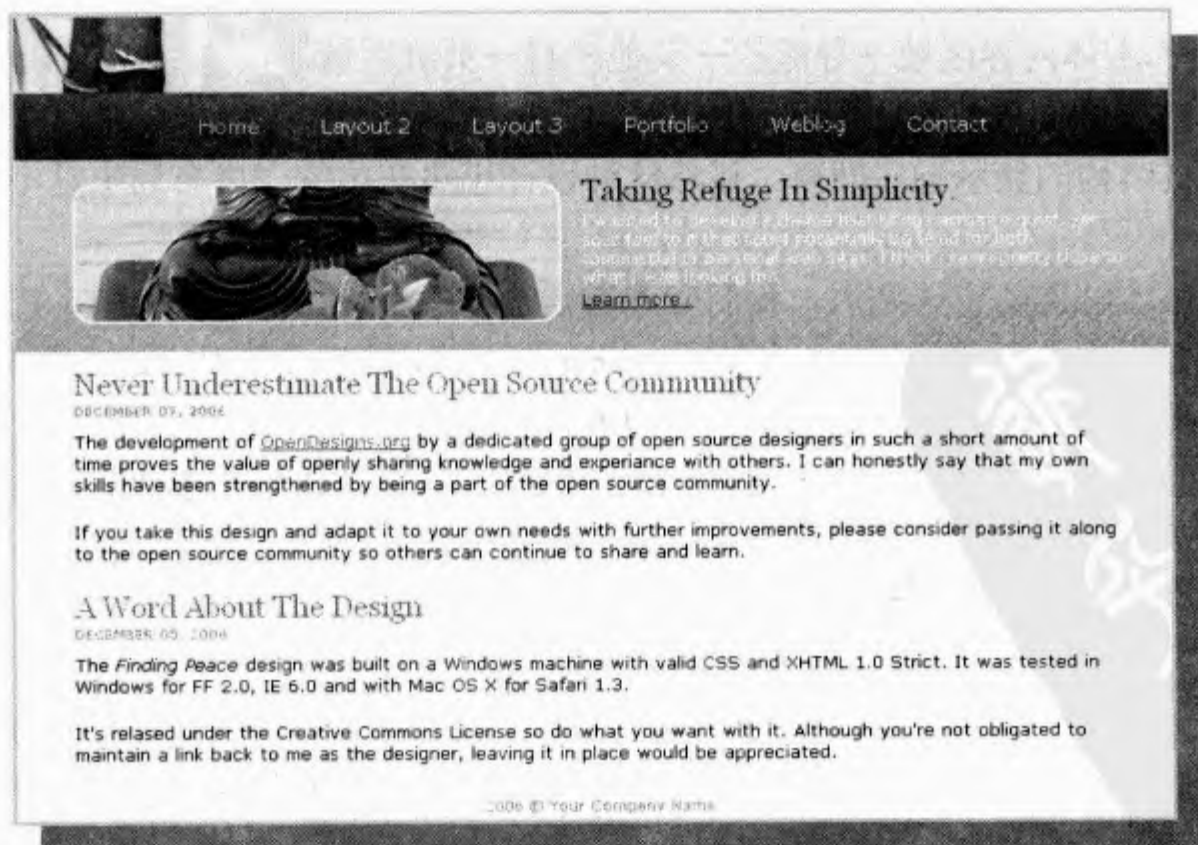


图 10.3 单列布局效果图

对于上面单列版式并包含 3 块结构的布局，可以使用自然流动的方法让它们自上而下地进行排列，并通过外边距或内边距来调整它们的显示位置（如图 10.4 所示）。

当然并不是多层嵌套结构就不能够实现单列设计。例如，针对上节示例中的结构：

```
<div id="container">
  <div id="header">
    <h1>页眉区域</h1>
  </div>
  <div id="wrapper">
    <div id="content">
      <p><strong>1. 主体内容区域</strong></p>
    </div>
    <div id="navigation">
      <p><strong>2. 导航栏</strong></p>
    </div>
    <div id="extra">
      <p><strong>3. 其他栏目</strong></p>
    </div>
    <div id="footer">
      <p>页脚区域</p>
    </div>
  </div>
</div>
```

通过浮动布局的方法，把<div id="navigation">和<div id="extra">模块并为一列显示（如图 10.5 所示）。

```
div#navigation {
  float:left;
  width:50%
}
div#extra {
  float:left;
  width:49.9%
}
```

/* 向左浮动 */
/* 百分比宽度 */
/* 百分比宽度 */
/* 百分比宽度 */

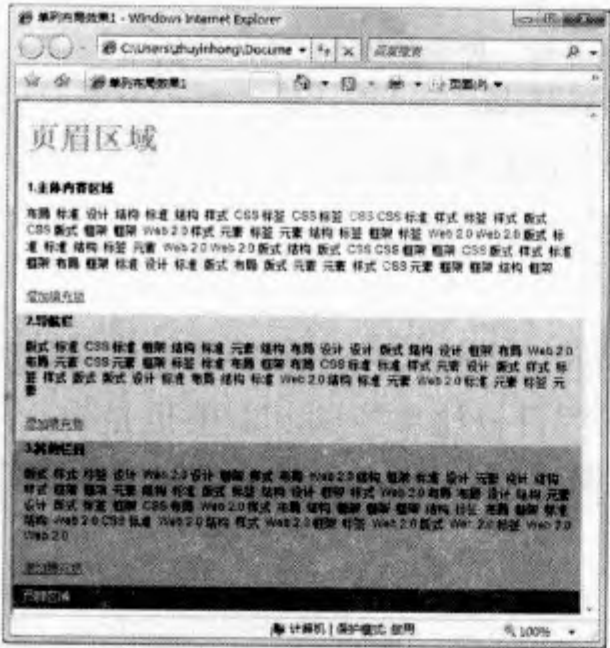


图 10.4 自然显示的单列布局效果

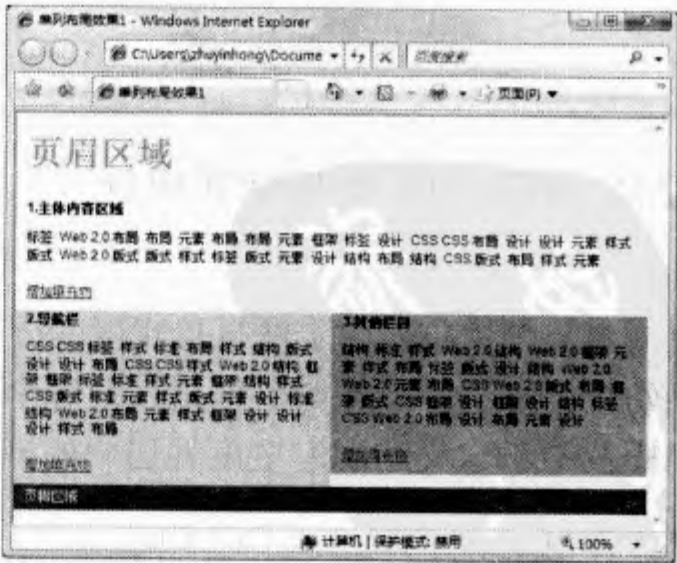


图 10.5 并列显示的单列布局效果

同样的道理，如果借助 CSS 定位布局来调整上面结构中不同板块的显示位置，也是件很轻松的事情。例如，遵照一般的浏览习惯，很多设计师喜欢把导航条放置在主体内容的顶部，即

放置在网页标题的下面。当然这样设计的前提是应该保证导航条高度固定，一般网页都设计为单行菜单条。

要实现这样的设计效果，应把<div id="container">包含框定义为包含块。然后设置<div id="wrapper">包含框的顶部外边距，强迫预留出一块空间。该顶部外边距的高度应该与导航条的固定高度一致。

```
#container {
    position:relative;          /* 定义相对定位包含块 */
}
#wrapper {
    margin-top:30px;            /* 增加主体内容区域的顶部外边距 */
}
```

最后定义<div id="navigation">导航条包含框为绝对定位，并固定到距离网页顶部 80 像素的位置，该高度为网页标题栏的高度。CSS 代码如下，所设计的效果如图 10.6 所示。

```
#navigation {
    position:absolute;          /* 绝对定位 */
    top:80px;                   /* 距离顶部的距离，该距离为标题栏高度 */
    height:30px;                /* 导航条高度，与主体区域的顶部外边距保持一致 */
    width:100%;                /* 宽度 */
    overflow:hidden;            /* 隐藏超出的区域 */
}
```

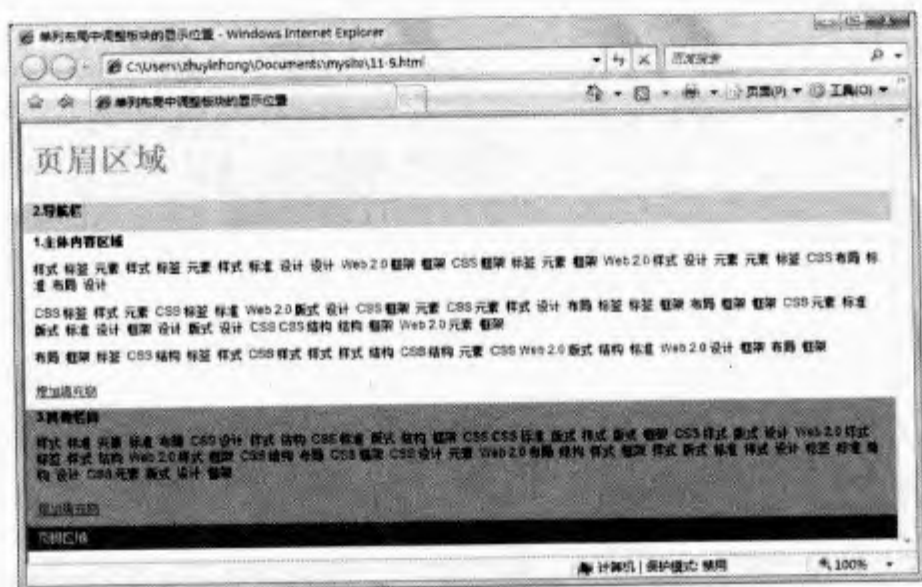


图 10.6 单列布局中调整板块的显示位置的效果图

10.2.2 单列固定宽度版式设计

固定宽度版式就是使用像素作为网页宽度的设置单位。当然其内部模块的宽度可以采用百分比或者其他单位，由于网页总宽度是固定的，如果采用百分比单位则其取值也是唯一的，因此不妨全部采用像素作为单位来设置每个栏目的宽度。例如，在 Dreamweaver CS3 中提供的 CSS 布局模板中所定义的固定宽度并居中的网页样式如下：

```
body {
    margin: 0;                  /* 清除页边距 */
    padding: 0;                 /* 清除页边距 */
    text-align: center;         /* 在 IE 5 浏览器中设置网页居中显示 */
}
.oneColFixCtrHdr #container {
    width: 780px;               /* 使用比最大宽度 (800px) 小 20px 的宽度可显示浏览器界面元素，并避免出
```



```
现水平滚动条 */
margin: 0 auto;
text-align: left;
}
/* 自动边距（与宽度一起）会将页面居中*/
/* 覆盖 body 元素中的 text-align: center */
```

下面我们通过一个案例来讲解单列固定宽度布局的设计过程和方法。图 10.7 所示为一个单列固定宽度的布局页面。在这个页面中所有对象都被控制在一个狭长的单列中显示，这样的版式效果很适合以开账单似的结构进行构建，这样的结构灵活性大，可扩张性强，可以随时增加子模块或者子结构。当然构建的前提条件是网页版式固定，即网页板块的位置在网页中的位置都是固定的，如靠左布局或靠右布局，而居中布局则不适合。

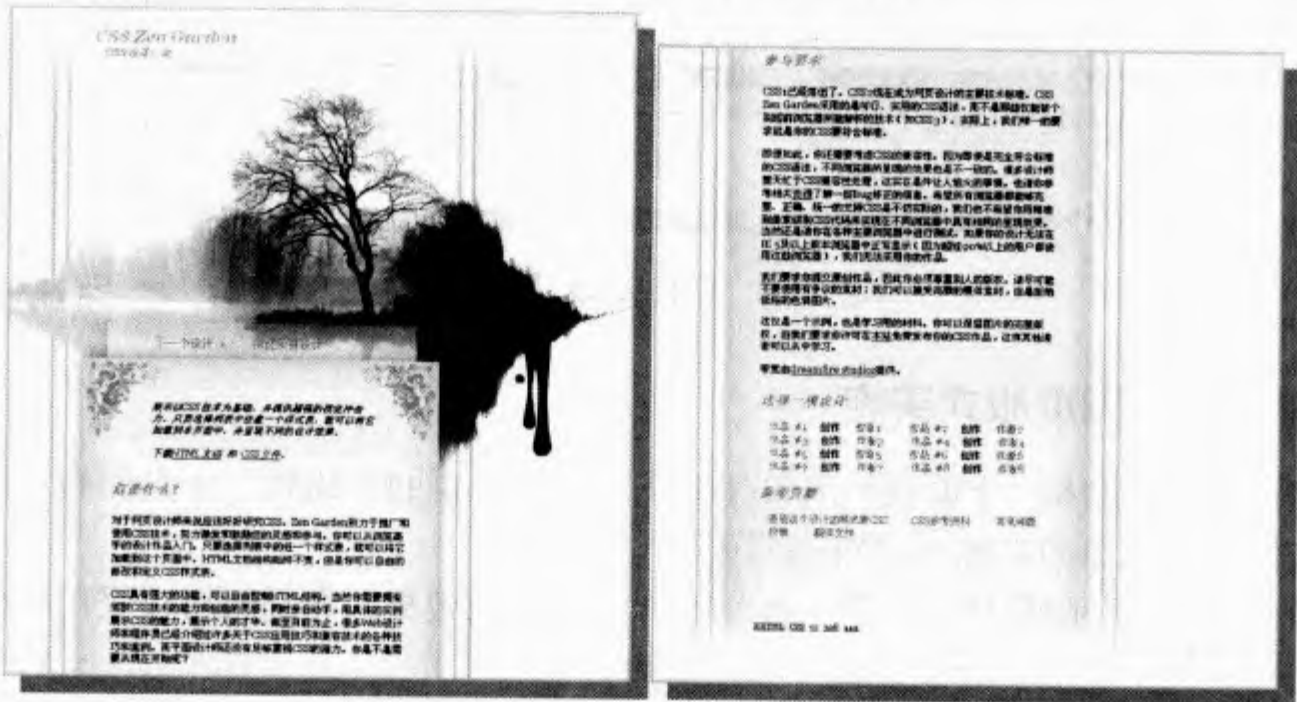


图 10.7 单列固定宽度布局效果图

本案例的网页基本结构如下，整个页面被分成 5 个独立的结构模块（如图 10.8 所示）。

```
<div id="main">
  <h1>网页标题</h1>
  <h2>副标题</h2>
</div>
<div id="linkbar">
  <div id="navcontainer">
    <ul id="navlist">
      <li id="active"><a href="#" id="current">菜单 1</a></li>
      <li><a href="#">菜单 2</a></li>
      <li><a href="#">菜单 3</a></li>
    </ul>
  </div>
</div>
<div id="top_content">
  <blockquote>
    <p>顶部内容</p>
  </blockquote>
</div>
<div id="main_content">
  <h3>内容标题</h3>
  <p>网页内容 </p>
</div>
<div id="footer">页眉区域</div>
```

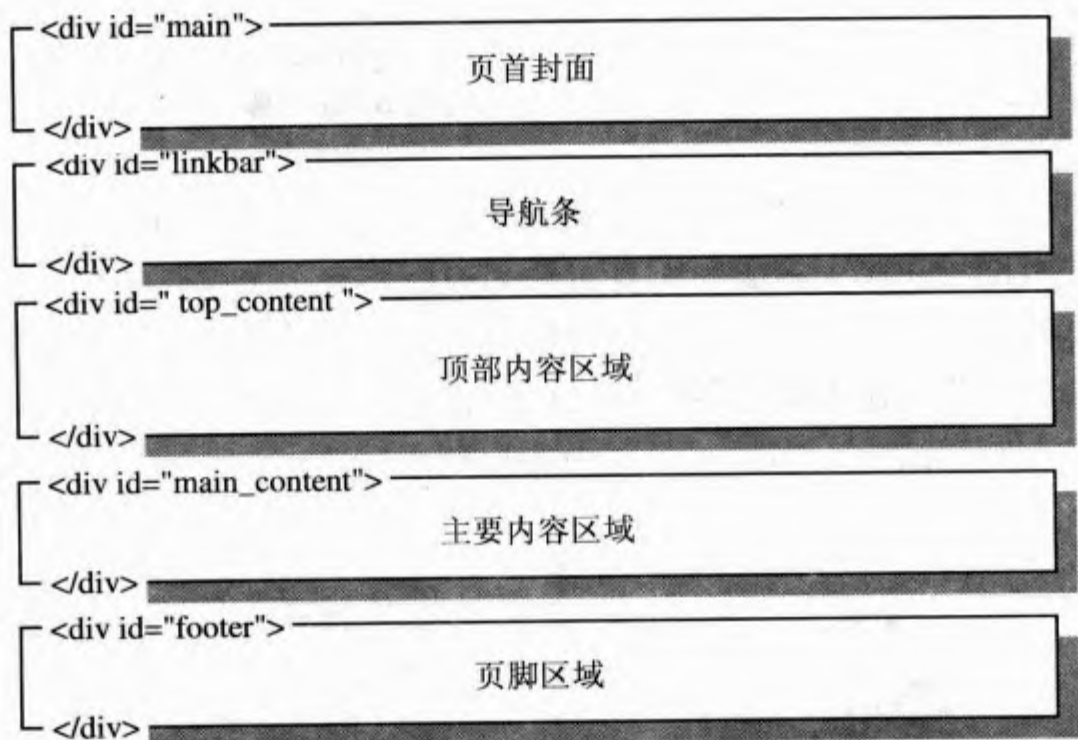


图 10.8 页面基本框架结构

10.2.3 单列固定宽度版式实施

遵循这样的网页结构，下面我们来设计 CSS 样式表。再回头比较一下页面布局效果和版式结构，本实例采用简单的自然流动布局即可。

第 1 步，设计第一版块样式。第一版块类似网页封面效果，布局时可以采用固定大小，并通过背景图像来渲染封面的画面效果（如图 10.9 所示）。

```
#main { /* 第一版块样式 */
    height: 399px;
    width: 741px;
    background-image: url(main.jpg);
    background-repeat: no-repeat;
}
```

/* 固定宽度 */
/* 固定高度 */
/* 背景图像 */
/* 禁止背景图像平铺 */

在一版块中包含 2 行标题，控制这些标题的显示主要通过内边距来进行调整。

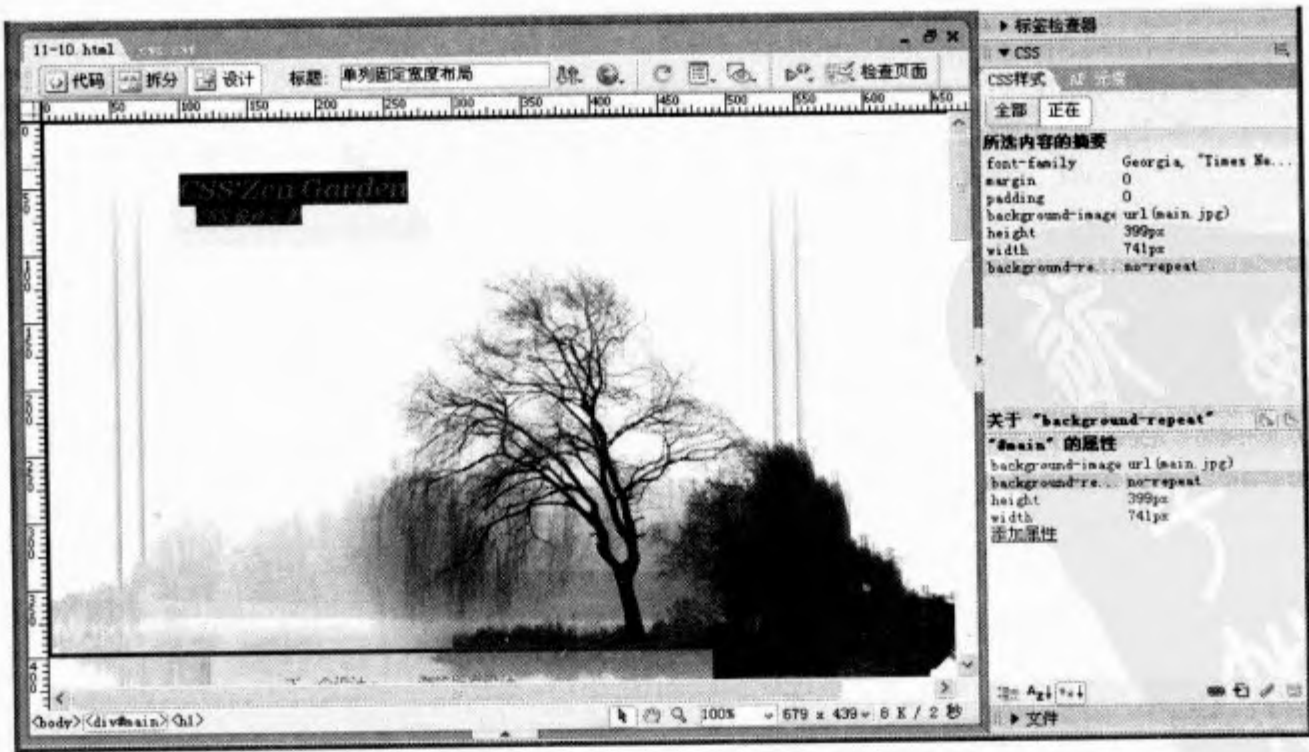


图 10.9 网页封面的设计效果


```
h1 { /* 第一版块中一级标题的显示位置 */
    padding-top: 40px;
    padding-left: 100px;
}
h2 { /* 第一版块中二级标题的显示位置 */
    padding-top: 0px;
    padding-left: 112px;
}
```

第 2 步, 设计第二版块样式。第二版块是一个导航条, 通过嵌套结构来实现版式的设计 (如图 10.10 所示)。

```
#linkbar { /* 第二版块样式 */
    height: 43px; /* 固定宽度 */
    width: 741px; /* 固定高度 */
    background-image: url(linkbar.jpg); /* 背景图像 */
    background-repeat: no-repeat; /* 禁止背景图像平铺 */
}
```

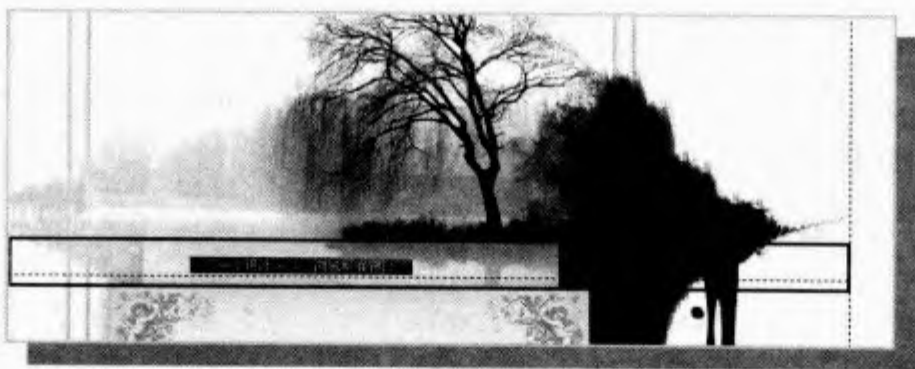


图 10.10 导航条的设计效果

然后再通过内嵌的<ul id="navlist">元素控制导航条的显示位置。

```
#navcontainer ul { /* 第二版块导航条的样式 */
    padding: 16px 0 2px; /* 菜单项在导航条内的位置 */
    margin-left: 160px; /* 导航条在第二模块中的位置 */
    list-style-type: none; /* 清除项目列表的默认样式 */
}
```

然后定义项目列表以行内显示, 实现多个列表项行内平行显示。

```
li { /* 列表项的样式 */
    display: inline; /* 行内显示 */
}
```

再定义导航条的超链接样式。

```
li a { /* 超链接的默认样式 */
    text-decoration: none; /* 清除下划线 */
    color: #666666; /* 字体颜色 */
    padding: .2em 1em; /* 调整间距 */
}
li a:hover { /* 鼠标经过超链接时的样式 */
    background-color: #EDEEE8; /* 加亮背景 */
    color: #333333; /* 加亮字体颜色 */
}
```

第 3 步, 第三版块的设计思路与上面两个版块的操作方法相似, 通过外层结构定义版块的大小和背景图像, 内层结构控制版块内容的显示位置 (如图 10.11 所示)。

```
#top_content { /* 第三板块的样式 */
    height: 135px; /* 固定宽度 */
    width: 741px; /* 固定宽度 */
    background-image: url(top_content.jpg); /* 背景图像 */
    background-repeat: no-repeat; /* 禁止背景图像平铺 */
}
```



图 10.11 顶部内容区域的设计效果

第 4 步，设计第四板块的样式。由于该板块为主要内容区域，包含大量的文本，因此其显示区域是弹性的，所以在设计时与上面几个板块的固定大小版式不同。可以通过背景图像的垂直平铺来实现（如图 10.12 所示）。

```
#main_content { /* 第四板块的样式 */
    width: 356px; /* 固定宽度 */
    background-image: url(body_tile.jpg); /* 背景图像 */
    background-repeat: repeat-y; /* 禁止背景图像平铺 */
    padding-left: 125px; /* 左侧内边距 */
    padding-right: 260px; /* 右侧内边距 */
}
```

第 5 步，设计第五板块的样式。在这个板块中也是通过背景图像和内边距来进行调整和设
计（如图 10.13 所示）。

```
#footer { /* 第五板块的样式 */
    background-image: url(footer.jpg); /* 背景图像 */
    width: 621px; /* 固定宽度 */
    height: 130px; /* 固定高度 */
    padding-top: 100px; /* 顶部内边距 */
    padding-left: 120px; /* 左侧内边距 */
}
```

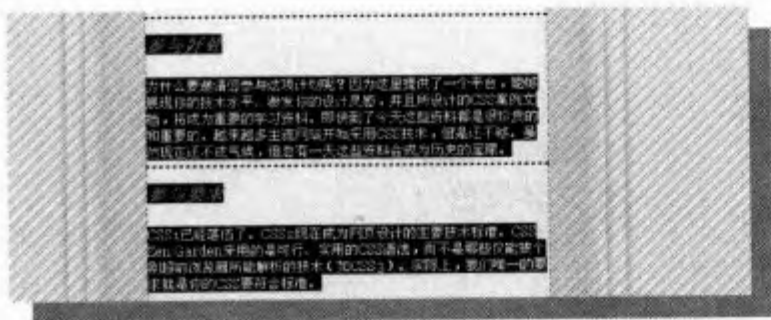


图 10.12 主要内容区域的设计效果



图 10.13 页脚区域的设计效果

10.2.4 江南水乡意象画布局设计

上述案例讲解了如何把一个并列、松散的结构设计为单列固定宽度的布局效果。那么是不

是其他网页结构就不能够实现这种单列固定宽度的布局效果呢?

答案是否定的。为了帮助读者能够认识到不同的网页结构都可以设计出相同的版式效果,下面我们就以禅意花园的结构为蓝本来设计上述案例的布局效果。

回忆一下禅意花园的基本结构(参阅第 7.2.3 节内容),它由 1 个外包含框和 3 个内嵌结构模块组成,这与上节讲解的案例结构截然不同,而且结构的排列顺序也存在很多不同。

为了能够使两个结构框架完全不同、结构顺序也存在很大差异的页面呈现出相同的设计效果。我们不妨采用如下设计方法。

把上面案例的第一、第二和第三模块的版式统一配置到禅意花园的<div id="intro">结构块中,把第四模块的版式设计到<div id="supportingText">结构块中,把第五模块的版式设计到<div id="linkList">结构块中。然后借助 CSS 定位技术适当调整不同结构中的子模块的显示位置。例如,把<div id="linkList">子结构中的<div id="lresources">和<div id="larchives">对象内容定位到页面顶部区域,把<div id="linkList">子结构中的<div id="lselect">与<div id="supportingText">子结构中的 div id="footer">对象内容进行位置交换。禅意花园整个结构的布局思路如图 10.14 所示。

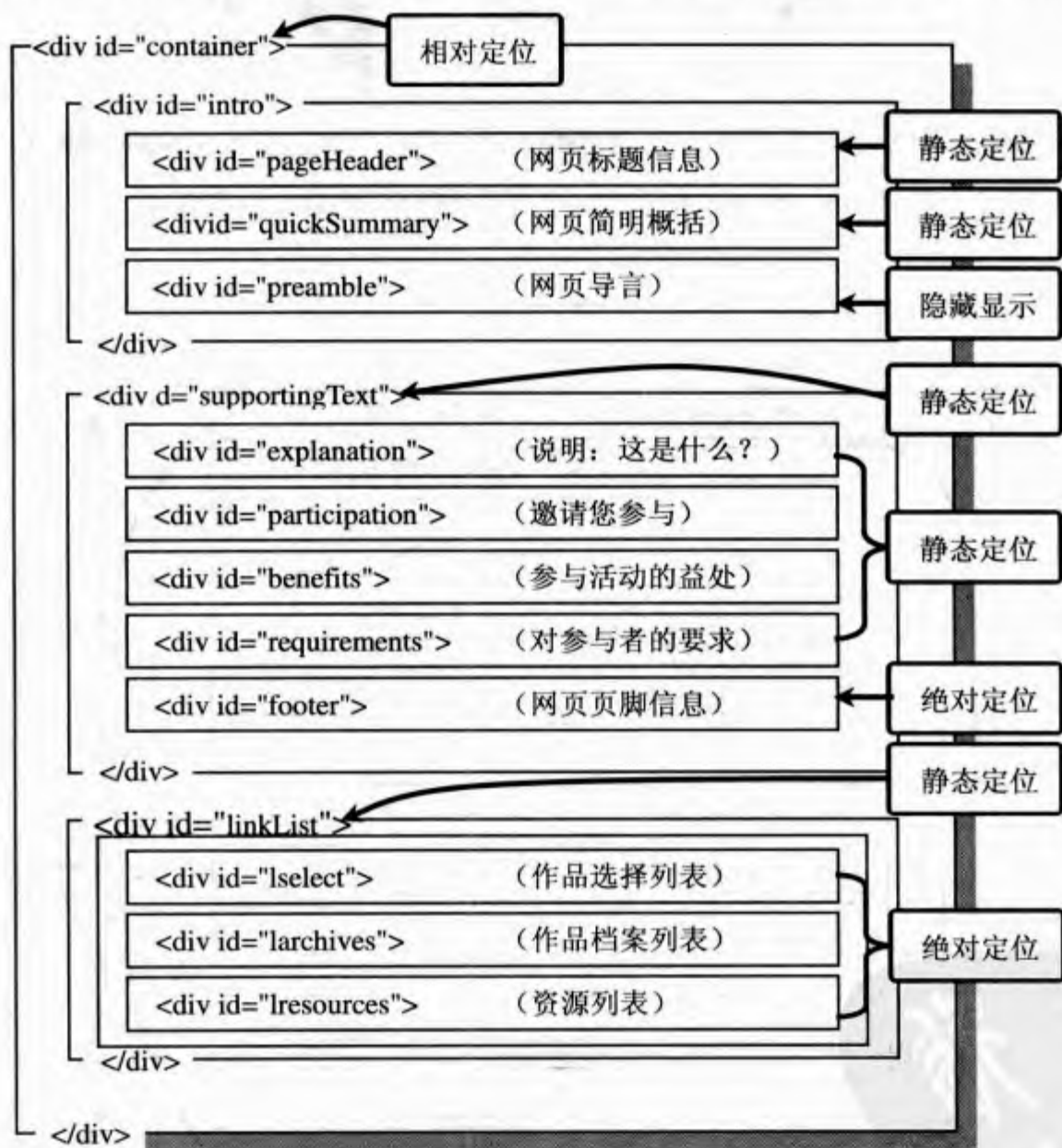


图 10.14 江南水乡意象画网页布局的结构定位思路

为了实现预设效果,这里定义禅意花园网页结构包含框(<div id="container">)为相对定位元素,即把它设计为包含块。然后隐藏<div id="preamble">模块,通过 CSS 定位法把<div id="lresources">和<div id="larchives">定位到头部区域(如图 10.15 所示)。其中蓝色粗线框为绝对定位的对象。

然后通过绝对定位的方法把作品列表模块（<div id="lselect">）与网页页脚模块（<div id="footer">）交换位置，操作示意图如图 10.16 所示。

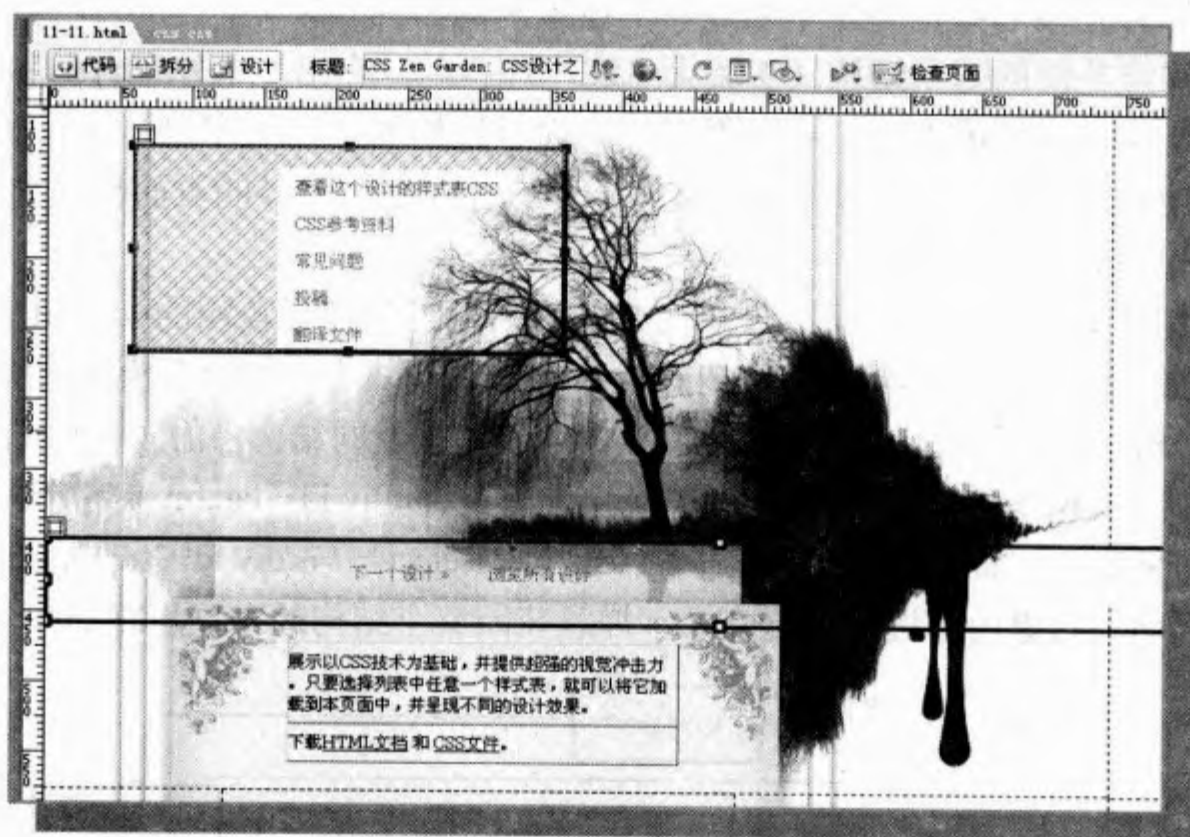


图 10.15 绝对定位到网页头部区域的页脚对象内容效果

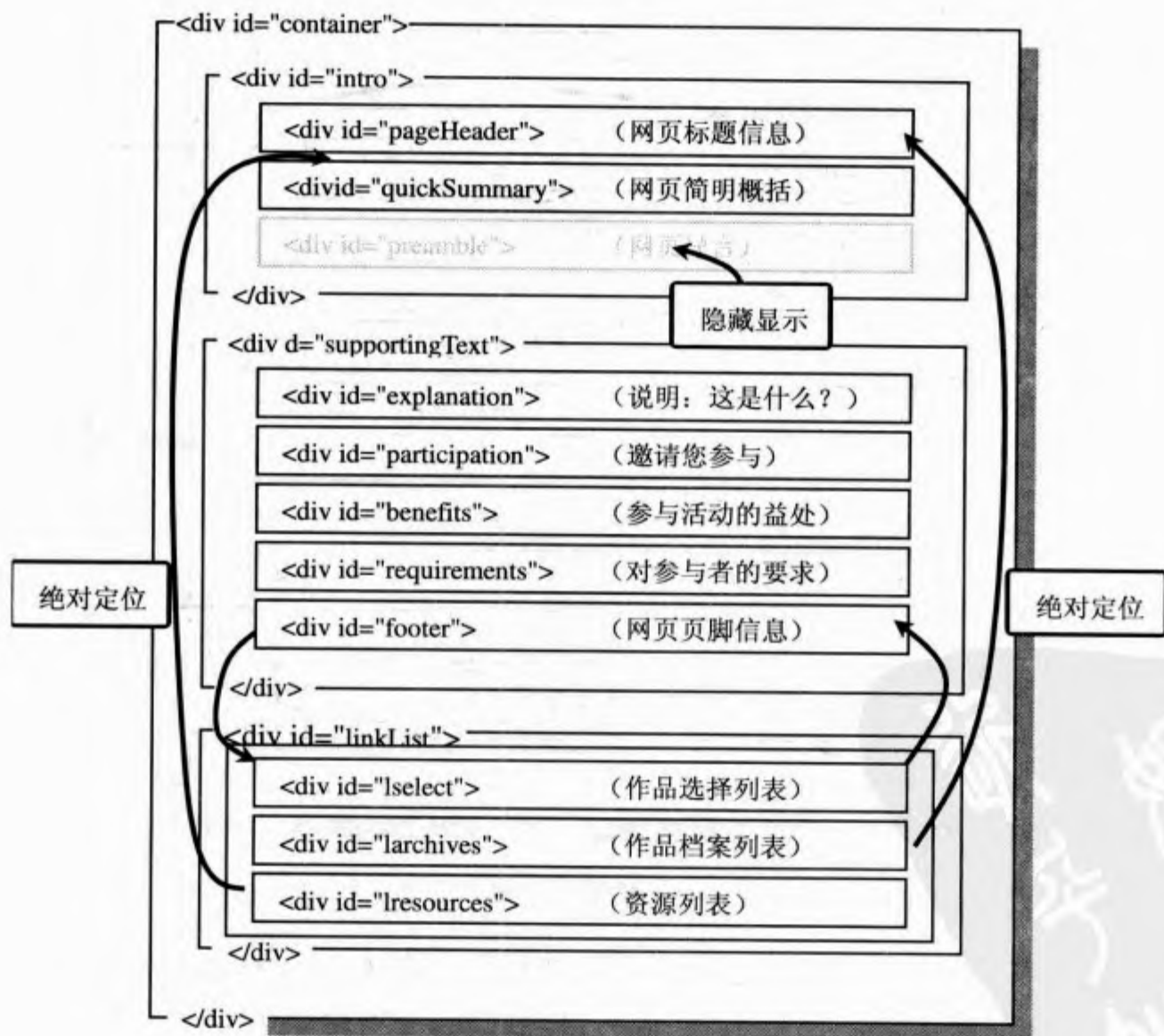


图 10.16 江南水乡意象画网页结构定位操作示意图

10.2.5 江南水乡意象画布局实战

在掌握了如何把禅意花园的结构设计为相应的版式之后,下面就来讲解如何具体实现本实例的布局效果。本节将重点讲解基本框架的布局效果,页面详细细节就不再涉及,读者可以参阅光盘实例进行研究。

第 1 步,定义网页包含块。这一步很重要,它将为后期的绝对定位奠定基础。

```
#container { /* 网页包含框 */
    position:relative; /* 相对定位,定义包含块 */
}
```

第 2 步,设计网页头部区域的样式。在这里把上一个案例中的前 3 个版块合并到这里,并分别应用到不同的结构中。

```
#pageHeader { /* 原案例中的第一版块样式 */
    height: 399px; /* 固定高度 */
    width: 741px; /* 固定宽度 */
    background-image: url(main.jpg); /* 背景图像 */
    background-repeat: no-repeat; /* 禁止背景图像平铺 */
}
#quickSummary { /* 原案例中的第三版块样式 */
    height: 135px; /* 固定高度 */
    width: 741px; /* 固定宽度 */
    background-image: url(top_content.jpg); /* 背景图像 */
    background-repeat: no-repeat; /* 禁止背景图像平铺 */
}
```

第 3 步,为了给页脚区域的对象内容绝对定位头部区域预留空间,这里通过增加外边距的方法来实现(如图 10.17 所示)。

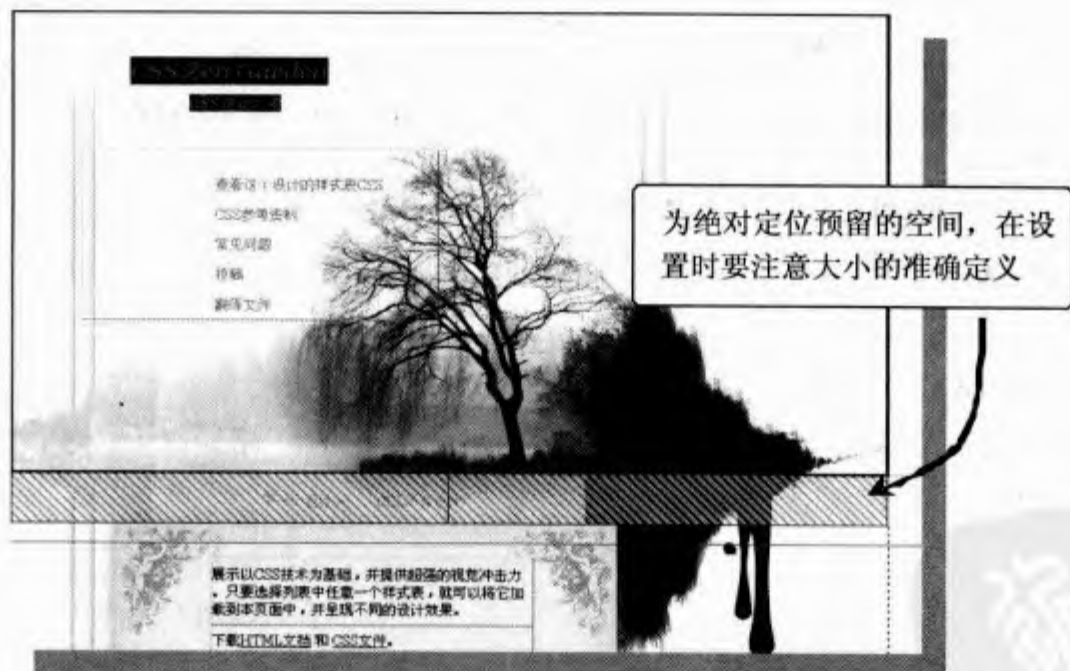


图 10.17 为绝对定位元素预留空间

```
#pageHeader { /* 为绝对定位元素预留空间 */
    margin-bottom:43px; /* 增加底部外边距 */
}
```

为什么不直接把绝对元素层叠在<div id="pageHeader">元素上面呢?这是因为绝对定位元素本身还需要负责定义网页的背景图像,也就是原案例中的第二版块的背景图像。

第 4 步,设置兼容不同浏览器的第三模块区域的显示文本。由于第三版本中的文本在不同

浏览器中会出现显示位置差异,需要使用兼容技术来设计 IE 浏览器和非 IE 浏览器的显示样式。例如,如果不使用兼容技术进行处理,在 IE 下文本显示位置恰当(如图 10.18 所示)。但是在非 IE 浏览器中会发现文本显示位置偏向顶部(如图 10.19 所示)。使用兼容技术处理之后,显示位置与 IE 浏览器保持一致。

```
#quickSummary p { /* 兼容 IE 浏览器 */
    margin-top:24px;          /* 增加顶部外边距 */
    margin-left:170px;        /* 增加左侧外边距 */
    margin-right:300px;       /* 增加右侧外边距 */
}
html>/**/body #quickSummary p {
    position:relative;        /* 兼容非 IE 浏览器 */
    top:30px;                 /* 相对定位 */
    margin-top:0;              /* 向下错移文本显示位置 */
                                /* 清除默认定义的顶部外边距 */
}
```

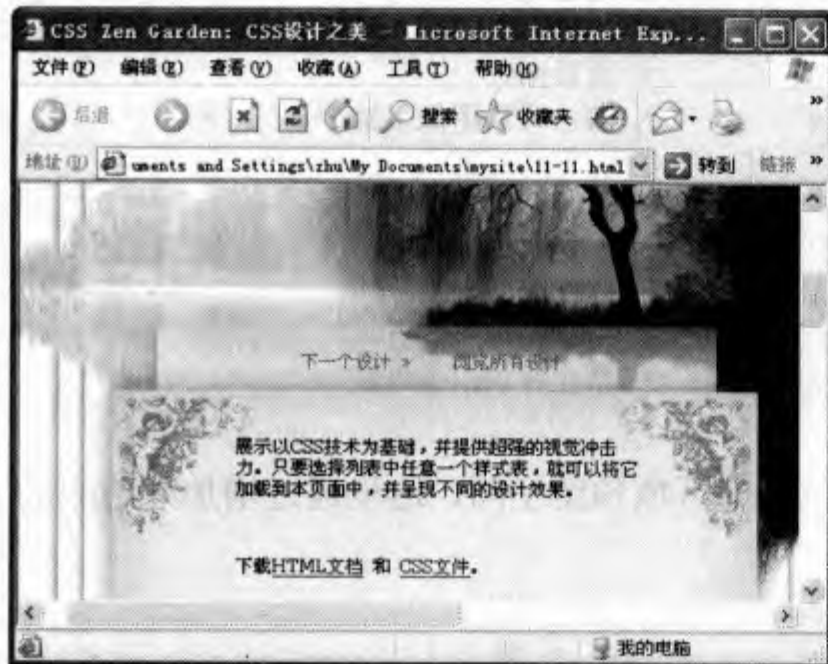


图 10.18 IE 下控制文本显示位置

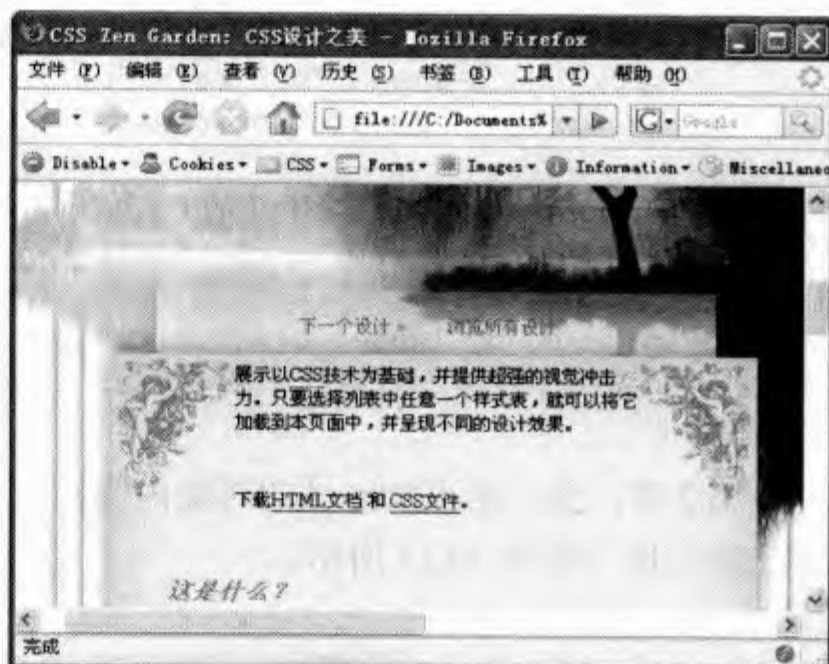


图 10.19 非 IE 下文本显示位置偏上

第 5 步,定义<div id="pageHeader">元素区域内的绝对定位元素。在第 3 步中通过底部外边距预留了一条空余的区域,下面就把<div id="linkList">模块内的<div id="larchives">元素绝对定位到该预留区域。

```
#larchives { /* 原案例中的第二版块样式 */
    background-image: url(linkbar.jpg); /* 增加背景图像 */
    height: 43px;                       /* 固定高度 */
    width: 741px;                       /* 固定宽度 */
    padding-top:16px;                   /* 增加顶部内边距 */
    padding-left:200px;                 /* 增加左侧内边距 */
    background-repeat: no-repeat;       /* 禁止背景图像平铺 */
    position:absolute;                  /* 绝对定位 */
    top:399px;                          /* 距离顶部距离 */
    left:0;                             /* 距离左侧距离 */
}
```

有的读者可能认为 left:0;声明有点多余,以为不设置为 0,它的取值默认也为 0。是的,但是如果显式定义它的值,则绝对定位元素在 x 轴上以相对定位的方式进行显示,这样就必然受到其他元素的影响。

同时再隐藏<div id="pageHeader">元素包含的区块标题。


```
#larchives h3 { /* 隐藏标题显示 */
    display:none;
}
```

第 6 步, 绝对定位<div id="lresources">元素显示在<div id="pageHeader">区域内 (如图 10.20 所示)。

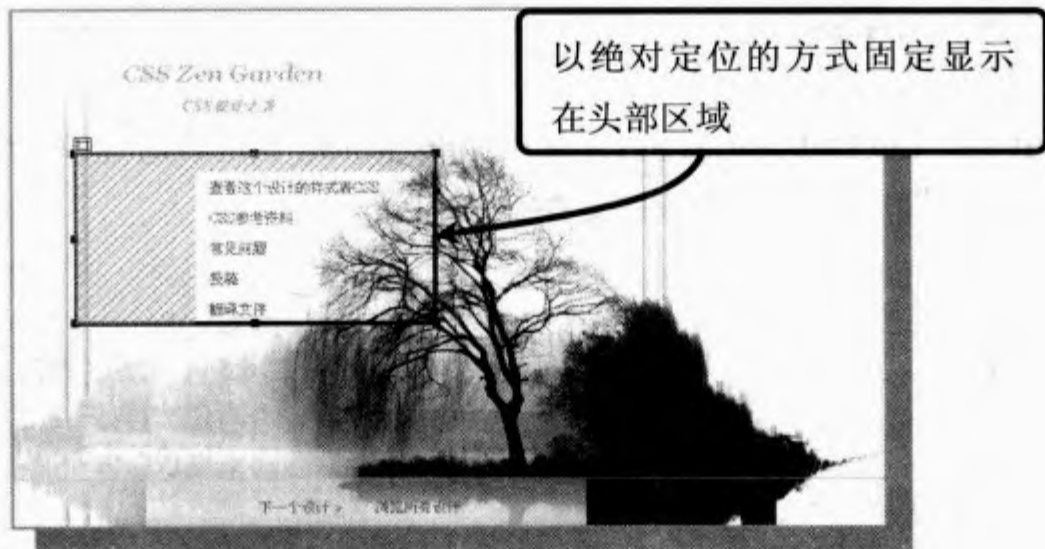


图 10.20 绝对定位元素

```
#lresources {
    width:200px;                /* 固定宽度 */
    padding-top:16px;           /* 增加顶部内边距 */
    padding-left:100px;         /* 增加左侧内边距 */
    position:absolute;          /* 绝对定位 */
    top:120px;                  /* 距离顶部距离 */
    left:60px;                  /* 距离左侧距离 */
}
#lresources ul li {
    float:left;                 /* 向左浮动显示项目列表 */
    clear:left;                 /* 清除左侧浮动, 实现垂直显示 */
    width:100%;                 /* 定义宽度 */
    line-height:26px;           /* 定义行高 */
}
#lresources h3 {
    display:none;               /* 隐藏标题显示 */
}
```

第 7 步, 定义<div id="supportingText">元素的样式。该样式继承上一案例中<div id="main_content">元素的样式。

```
#supportingText { /* 第四板块的样式 */
    width: 356px;                /* 固定宽度 */
    background-image: url(body_tile.jpg); /* 背景图像 */
    background-repeat: repeat-y; /* 禁止背景图像平铺 */
    padding-left: 125px;          /* 左侧内边距 */
    padding-right: 260px;         /* 右侧内边距 */
}
```

第 8 步, 定义<div id="supportingText">元素所包含的子对象<div id="footer">为绝对定位显示, 以脱离与父元素<div id="supportingText">的联系, 并固定显示在页面的底部。

```
#footer {
    text-align:center;           /* 文本居中对齐 */
    position:absolute;          /* 绝对定位显示 */
    bottom:20px;                /* 距离包含块底部的距离 */
    left:0;                     /* 距离包含块左侧的距离 */
    width: 356px;               /* 固定宽度 */
    padding-left: 125px;         /* 左侧内边距 */
    padding-right: 260px;        /* 右侧内边距 */
}
```

第9步, 把<div id="linkList">元素包含的<div id="lselect">元素通过绝对定位的方式提升到<div id="footer">元素原来的位置, 即与它交换显示位置。

```
#lselect {
    position:absolute;          /* 绝对定位 */
    bottom:60px;                /* 距离包含块底部距离 */
    left:0;                     /* 距离包含块左侧距离 */
    width: 356px;               /* 固定宽度 */
    padding-left: 125px;         /* 左侧内边距 */
    padding-right: 260px;        /* 右侧内边距 */
}
```

10.3 单列液态框架版式布局

液态布局就是网页宽度的取值单位为百分比, 而不是像素。使用百分比定义网页的显示宽度能够使页面适应不同的显示器屏幕尺寸, 现在很多设计师在设计符合标准的网页布局中比较喜欢采用这种布局方式。本节将通过一个案例, 展示如何设计单列液态宽度的页面布局效果。

10.3.1 使用 CSS 设计框架结构

你熟悉下面的 HTML 源代码吗? 这是一个顶部框架的框架集, 即一个上下结构的网页框架结构。也许你曾经为了设计框架集而烦恼。是的, 框架集在设计和操作上都比较麻烦, 它既不受用户的欢迎, 也不受搜索引擎的欢迎, 主要是因为它破坏了 HTML 的结构体系。当然框架集在很多时候还是很有用, 使用框架集更有利于页面信息的组织和管理, 也有利于网页文件的组织。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head></head>
<frameset rows="80,*" frameborder="no" border="0" framespacing="0">
    <frame src=" " name="topFrame" scrolling="No" noresize="noresize" id="topFrame"
title="topFrame" />
    <frame src=" " name="mainFrame" id="mainFrame" title="mainFrame" />
</frameset>
<noframes>
<body>
</body>
</noframes>
</html>
```

下面尝试使用 CSS 技术来设计一个框架页面 (如图 10.21 所示), 这是一个使用 CSS 技术设计的上下结构的框架页面。



图 10.21 绝对定位元素

这个模拟框架集的页面结构如下：

```
<div id="banner">
    <!-- 顶部框架-->
</div>
<div id="maincontent">
    <div class="maintext">
        <!-- 底部框架-->
    </div>
</div>
```

它是由独立的 div 元素组成的两个包含框，然后通过 CSS 设计禁止顶部包含框随滚动条滚动，定义下面包含框显示滚动条，以实现在一个屏幕中显示所有网页信息（如图 10.22 所示）。

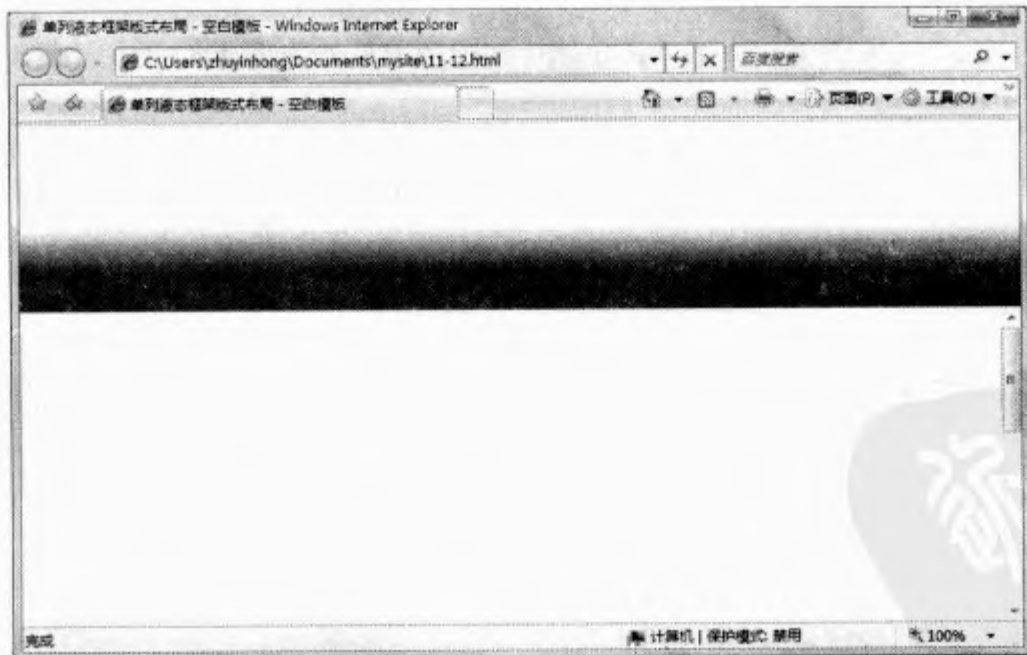


图 10.22 CSS 设计的上下结构框架集效果

实现上下框架的 CSS 样式代码如下：

```
body { /* 网页基本属性 */
    margin: 0;
    padding: 0;
    border: 0;
    /* 清除页边距 */
    /* 清除页边距 */
    /* 清除网页默认的边框 */
}
```

```

overflow: hidden; /* 隐藏超出的区域 */
height: 100%; /* 定义网页高度 */
max-height: 100%; /* 定义在 IE 7 及其他标准浏览器中最大网页高度 */
font: 85%/160% verdana, arial, helvetica, sans-serif; /* 页面字体属性 */
color: #333; /* 页面字体颜色 */
}
#banner { /* 顶部框架样式 */
position: absolute; /* 绝对定位 */
top: 0; /* 距离网页顶部的距离 */
left: 0; /* 距离网页左侧的距离 */
width: 100%; /* 定义显示宽度 */
height: 160px; /* 固定顶部框架的高度 */
overflow: hidden; /* 隐藏超出的区域 */
background: #353374 url(bg_header.gif) repeat-x scroll 0% 0%; /* 定义背景图像 */
}
#maincontent { /* 底部框架样式 */
position: fixed; /* 固定定位 */
top: 160px; /* 距离网页顶部的距离 */
left: 0; /* 距离网页左侧的距离 */
right: 0; /* 距离浏览器右侧距离为 0 */
bottom: 0; /* 距离底部的距离为 0 */
overflow: auto; /* 距离左侧的距离为 0 */
}
.maintext { /* 底部框架的内容框样式 */
font-size: 0.85em; /* 定义字体大小 */
margin: 15px; /* 定义外边框 */
}

```

但是这种设计方法在 IE 6 及其一下版本中还存在问题, 需要使用兼容技术来专门为 IE 6 及其以下版本浏览器设计兼容样式。如果没有如下第一个样式, 则在 IE 6 及其以下版本浏览器中会显示图 10.23 所示的效果, 如果没有如下第二个样式, 则在 IE 6 及其以下版本浏览器中会显示图 10.24 所示的效果。

```

* html body { /* 在 IE6 及其以下版本浏览器中清除顶部框架的滚动条 */
padding: 160px 0 0 0; /* 定义顶部内边距 */
}
* html #maincontent { /* 在 IE6 及其以下版本浏览器中显示底部框架的滚动条 */
height: 100%; /* 显式定义高度 */
width: 100%; /* 显式定义宽度 */
}

```

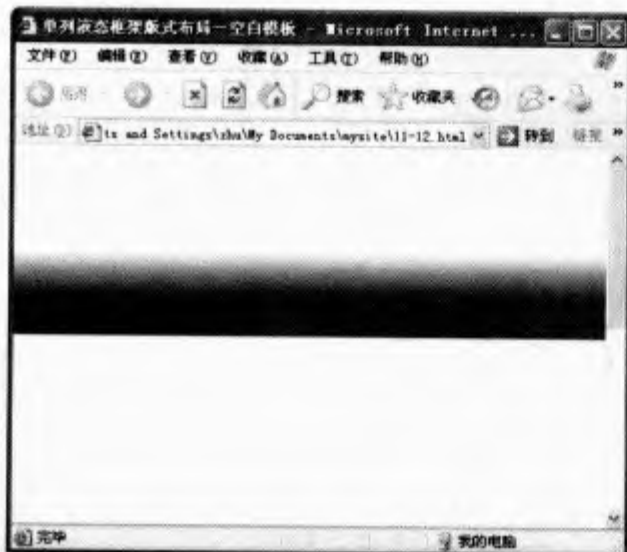


图 10.23 IE 6 版本浏览器中整个窗口显示滚动条

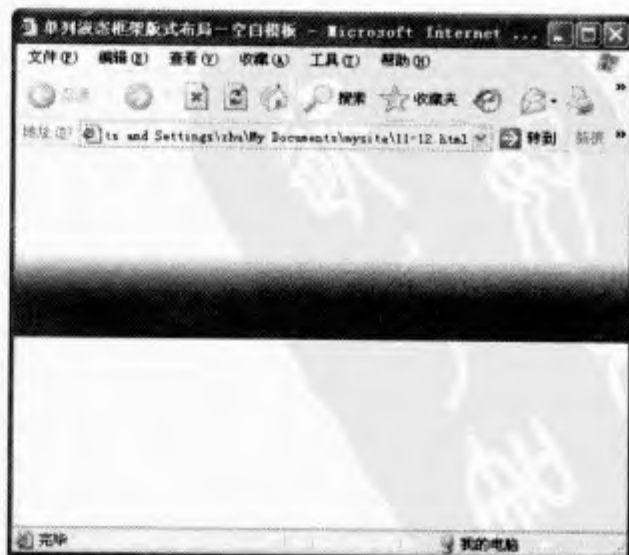


图 10.24 IE 6 版本浏览器中整个窗口不显示滚动条

最后还需要在页面源代码顶部增加一条 IE 专用命令, 该命令在其他版本和浏览器中被认为注释语句, 但是 IE 6 及其以下版本浏览器在怪异模式 (或称兼容模式) 下会根据这条命令来解析页面信息。如果没有这条命令, IE 浏览器就会按默认的模式来解析页面, 此时页面显示效果如图 10.25 所示。

```
<!-- IE6 quirks mode -->
```

或者

```
<!--Force IE6 into quirks mode with this comment tag-->
```



图 10.25 CSS 设计的上下结构框架集效果

所谓怪异模式就是 IE 浏览器为了兼容标准浏览器解析模式, 以及确保向后兼容性, 在 IE 6 及其以后的版本浏览器中内嵌了两种网页解析模式: Standards Mode (或称 Strict Mode, 翻译为标准模式) 和 Quirks Mode (或称 Compatibility Mode, 翻译为怪异模式或兼容模式)。在标准模式中, IE 浏览器会根据 W3C 所制定的规范来解析页面, 而在怪异模式中, 将以 IE 5, 甚至以 IE4 显示页面的方式来解析页面, 以确保向后兼容性, 保证以前的网页能够正常显示。

10.3.2 使用 CSS 设计单列液态布局

本节将在上面 CSS 设计的框架中来设计单列液态布局, 整个页面布局的显示效果如图 10.21 所示。对于液态布局来说, 页面及其宽度应以百分比作为单位来进行设置。下面就按从上到下的顺序来讲解各部分的具体实现方法和过程。

顶部框架包含 3 个子元素, 具体结构如下:

```
<div id="bannertext">
  <h1></h1>
  <h2></h2>
</div>
<div id="tabs">
  <ul>
    <li>导航菜单</li>
  </ul>
</div>
<div id="tabslines">&nbsp;</div>
```

`<div id="bannertext">` 负责控制网页标题的显示。这里通过背景图像的方式来定义, 而把 `<h1>` 标签包含的网页文本隐藏, 同时通过浮动的方式定位网页副标题的显示位置, 并通过外边距来调整与窗口右侧边框的距离 (如图 10.26 所示)。

```

#bannertext {
    margin: 20px 10px 0 80px;          /* 调整背景图像的显示位置 */
    padding-bottom: 20px;              /* 增加底部内边距 */
    background:url(logo.gif) no-repeat left 10px; /* 定义 Logo 背景图像 */
}
#bannertext h1 {
    display:none;                      /* 隐藏网页标题 */
}
#bannertext h2 {
    float:right;                       /* 浮动网页副标题到右侧 */
    font-size:16px;                   /* 字体大小 */
    margin:60px 80px 0 0;              /* 调整显示位置 */
}

```



图 10.26 设计网页标题样式

由于不同类型浏览器解析的差异，在标准浏览器中会显示图 10.27 所示效果。因此为能够兼容标准浏览器，可以增加如下的兼容性样式：

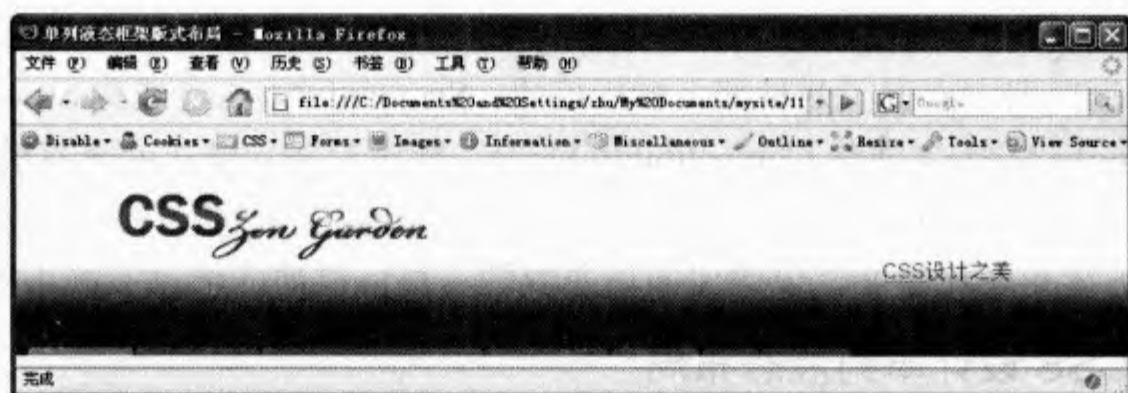


图 10.27 在 FF 中显示的效果

```

html>/**/body #bannertext {/* 兼容 FF 等标准浏览器 */
    margin: 0 10px 0 80px;          /* 调整外边距 */
    background:url(logo.gif) no-repeat left 30px; /* 定义背景图像 */
}
html>/**/body #bannertext h2 {/* 兼容 FF 等标准浏览器 */
    margin:80px 200px 0 0;          /* 调整外边距 */
}

```

本实例导航条是使用滑动门的技术来设计的，代码如下所示。有关滑动门技术的讲解请参阅第 5 章内容。

```

#tabs ul {
    margin:0;                          /* 清除项目缩进 */
    padding:0;                         /* 清除项目缩进 */
    list-style:none;                   /* 清除项目符号 */
}
#tabs li {
    display:inline;                    /* 行内显示 */
}

```



```

margin:0 2px 0 0;          /* 调整外边距 */
padding:0;                 /* 清除内边距 */
}
#tabs a {/* 滑动门 */
float:left;                /* 向左浮动 */
color: #fff;               /* 字体颜色 */
background: #6866a7 url(color_tabs_left.gif) no-repeat left top;
margin:0 2px 0 0;          /* 调整外边距 */
padding:0 0 1px 3px;       /* 增加门边 */
}
#tabs a span {/* 滑动门 */
float:left;                /* 浮动显示 */
display:block;             /* 块状显示 */
background: transparent url(color_tabs_right.gif) no-repeat right top;
padding:4px 9px 2px 6px;   /* 设置内边距, 定义门边 */
}

```

底部框架也是按着自然流动的布局方式显示页面内容, 所以不再多说。其中包含两项如下 CSS 设计技巧需要读者重视。

第一, 圆角区域的设计。在第 4 章中我们曾经讲解了 CSS 设计圆角区域的技巧, 不过本圆角的结构比较特殊 (如下所示)。它使用项目列表标签来设计, 这样可以优化圆角结构, 并简化了 CSS 的控制代码, 通过一个图片的不同定位来设计 4 个顶角的圆角显示效果, 这比调用 4 个顶角图像来设计圆角显得更为方便和便捷 (如图 10.28 所示)。

```

<div class="roundedbox">
  <div class="top">
    <ul>
      <li>&nbsp;</li>
    </ul>
  </div>
  <div id="contentbox">
    <!-- 包含的内容 -->
  </div>
  <div class="bottom">
    <ul>
      <li>&nbsp;</li>
    </ul>
  </div>
</div>

```

所设计的 CSS 样式如下:

```

.roundedbox {
float: right;              /* 浮动显示 */
width: 220px;              /* 圆角区域宽度 */
background-color: #9b99da; /* 圆角区域的背景色 */
}
#contentbox {
padding: 0 10px 0 10px;   /* 内容框的内边距 */
}
.roundedbox ul {
height: 15px;             /* 高度 */
list-style: none;         /* 清除项目符号 */
margin: 0;                /* 清除项目缩进 */
}

```

```

}
.roundedbox li {
    float: right; /* 向右浮动 */
    width: 15px; /* 宽度 */
    line-height: 15px; /* 行高 */
}
.top ul { background: url(box.gif) -15px -15px no-repeat; } /* 左上顶角 */
.top ul li { background: url(box.gif) 0px -15px no-repeat; } /* 右上顶角 */
.bottom ul { background: url(box.gif) -15px 0px no-repeat; } /* 左下顶角 */
.bottom ul li { background: url(box.gif) 0px 0px no-repeat; } /* 右下顶角 */

```

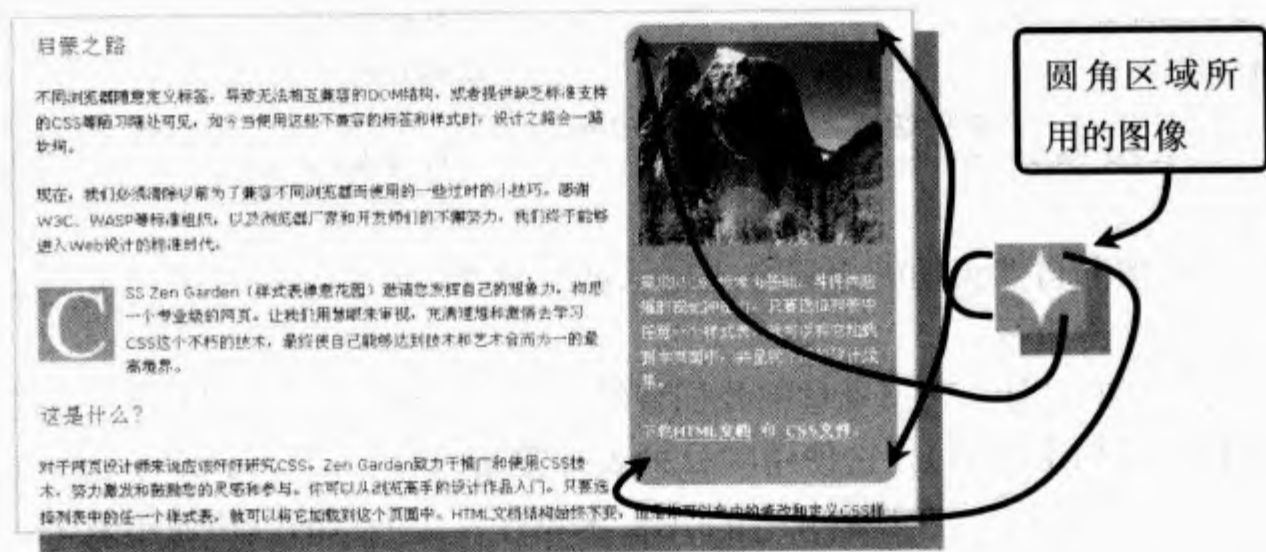


图 10.28 设计的圆角样式效果

第二，设计首字下沉。CSS 提供了一个首字下沉的伪对象 `first-letter`，当然也可以使用 CSS 来直接设计，这里需要一些属性配合使用：`font-size`、`float`、`padding` 等。所定义的首字下沉类样式的详细代码如下，设计效果如图 10.28 所示。

```

.cap { /* 首字下沉类样式 */
    margin-right: 6px; /* 右侧外边距 */
    margin-top: 5px; /* 顶部外边距 */
    float: left; /* 向左浮动 */
    color: white; /* 字体颜色 */
    background: #9b99da; /* 背景颜色 */
    font-size: 80px; /* 字体大小 */
    line-height: 60px; /* 行高 */
    padding: 2px 5px 0 5px; /* 内边距 */
    font-family: times new roman, times, serif; /* 字体属性 */
}

```

10.4 两列版式布局

Dreamweaver CS3 在 CSS 布局模板中设计了 16 种版式，足见两列布局在网页设计中的重要性，具体说包括如下 4 小类。

(1) 固定宽度布局（以像素为宽度单位）：“2 列固定，右侧栏”、“2 列固定，右侧栏、标题和脚注”、“2 列固定，左侧栏”和“2 列固定，左侧栏、标题和脚注”。

(2) 弹性宽度布局（以 `em` 为宽度单位）：“2 列弹性，右侧栏”、“2 列弹性，右侧栏、标题和脚注”、“2 列弹性，左侧栏”和“2 列弹性，左侧栏、标题和脚注”。

(3) 液态宽度布局（以百分比为宽度单位）：“2 列液态，右侧栏”、“2 列液态，右侧栏、

标题和脚注”、“2 列液态，左侧栏”和“2 列液态，左侧栏、标题和脚注”。

(4) 混合宽度布局（混合使用百分比和 em 为宽度单位）：“2 列混合，右侧栏”、“2 列混合，右侧栏、标题和脚注”、“2 列混合，左侧栏”和“2 列混合，左侧栏、标题和脚注”。

下面我们以这些版式为基础，详细讲解两列版式布局的一般规律和设计技巧。

10.4.1 两列版式的布局结构和设计方法

两列版式是最常用的网页布局样式，可以在网上随处看到这种网页设计效果。两列版式中一般以主辅两列的形式排版网页内容。主栏显示网页重要信息和列表，辅栏（或称侧栏）提供功能性的服务，有时可以根据需要在网页顶部和底部增加页眉和页脚，甚至还可以扩展其他栏目。

Dreamweaver CS3 提供的 16 种两列版式包含了两种基本的网页结构。

第一种，仅包含网页主体内容和辅助内容的结构，即整个网页仅包含 2 个独立的模块：

```
<div id="container">
  <div id="sidebar1">
    <h3>侧栏内容</h3>
  </div>
  <div id="mainContent">
    <h1>主要内容</h1>
  </div>
</div>
```

第二种，包含网页标题、侧栏内容、主体内容和脚注（网页页脚），即包含 4 个独立的模块：

```
<div id="container">
  <div id="header">
    <h1>标题</h1>
  </div>
  <div id="sidebar1">
    <h3>侧栏内容</h3>
  </div>
  <div id="mainContent">
    <h1>主要内容</h1>
  </div>
  <div id="footer">
    <p>脚注</p>
  </div>
</div>
```

在两列版式布局中，Dreamweaver CS3 完全采用了浮动显示，或者与自然流动显示相结合的方法来实现。例如，针对“2 列弹性，左侧栏、标题和脚注”的版式所设计的主要样式如下：

```
.twoColElsLtHdr #mainContent { /* 主栏样式 */
  margin: 0 1.5em 0 13em; /* 右边距可以用全方 (em) 或像素来指定，它会在页面的右下方产生空白。 */
}
.twoColElsLtHdr #sidebar1 { /* 侧栏样式 */
  float: left;
  width: 12em; /* 由于此元素是浮动的，因此必须指定宽度 */
  background: #EBEBEB; /* 将显示背景色，其宽度等于栏中内容的长度， */
  padding: 15px 0; /* 顶部和底部的填充将在该 div 中产生视觉空间 */
}
```

在上面的主要样式中，侧栏向左浮动，主栏以自然流动的方式环绕在侧栏的右侧，通过设置主栏左外边距来调整它与侧栏之间的距离，所得的设计效果如图 10.29 所示。



图 10.29 “2列弹性，左侧栏、标题和脚注”版式效果

这种以浮动和流动相结合的策略也是大部分设计师常用的一种布局方法，不过也可以让主栏与侧栏都浮动显示，例如：

```
.twoColElsLtHdr #mainContent { /* 主栏样式 */
    float: right;
}
.twoColElsLtHdr #sidebar1 { /* 侧栏样式 */
    float: left;
}
```

这种方法存在一定的风险，如果设计不完善或者不合理，都容易出现浮动错位问题，如何解决浮动错位问题可以参阅第 7.4.9 节讲解。

另外，还可以使用 CSS 定位法来设计。例如，定义包含框为相对定位，即定义包含块。然后定义侧栏绝对定位显示，让主栏环绕在侧栏右侧。

```
.twoColElsLtHdr #container { /* 包含块样式 */
    position: relative;
    width: 46em; /* 当文本保持浏览器的默认字体大小时，此宽度将创建一个适合 800px 浏览器窗口的容器 */
    background: #FFFFFF;
    margin: 0 auto; /* 自动边距（与宽度一起）会将页面居中 */
    border: 1px solid #000000;
    text-align: left; /* 这将覆盖 body 元素上的“text-align: center”。 */
}
.twoColElsLtHdr #sidebar1 { /* 侧栏绝对定位 */
    position: absolute;
    left: 0px;
    width: 12em; /* 由于此元素是浮动的，因此必须指定宽度 */
    background: #EBEBEB; /* 将显示背景色，其宽度等于栏中内容的长度， */
    padding: 15px 0; /* 顶部和底部的填充将在该 div 中产生视觉空间 */
}
```

或者也可以定义主栏绝对定位，并靠右显示：

```
.twoColElsLtHdr #mainContent { /* 主栏绝对定位 */
    position: absolute;
    right: 0px;
}
```

使用这种方法最好能够保证底部不要预留栏目，否则绝对定位模块会覆盖下面栏目的内容。因此绝对定位是脱离文档流显示的，它的高度变化不会影响其他栏目的高度。例如，针

对上面的绝对定位样式，如果增加侧栏的高度，则它很可能会覆盖页脚区域（如图 10.30 所示）。

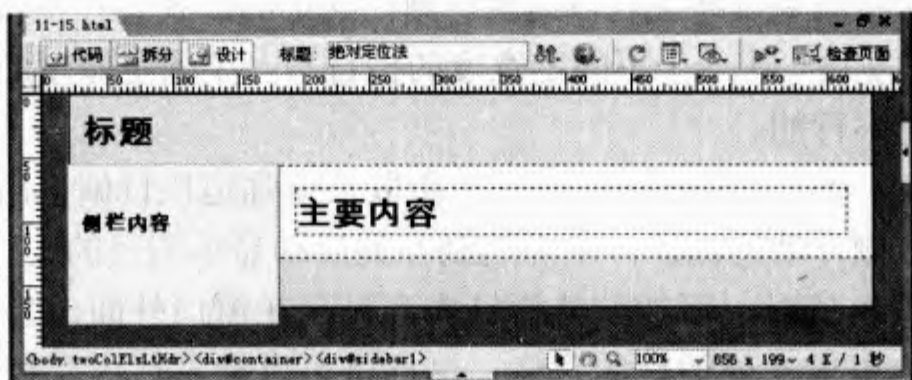


图 10.30 绝对定位法存在的问题

这种绝对定位的方法除非特殊需要，一般在设计开发中不建议使用。

10.4.2 两栏浮动版式中兼容固定和自适应宽度的设计方法和探索

在第一节中我们曾经讲解了一个包含 5 个模块的典型标准结构（如下面程序所示），下面借助这个结构来尝试设计不同的版式效果。通过这种方法帮助读者快速掌握多种布局效果的设计思路和实现方法，同时体会针对同一个结构可以设计不同的版式布局。

```
<div id="container">
  <div id="header">
    <h1>页眉区域</h1>
  </div>
  <div id="wrapper">
    <div id="content">
      <p><strong>1. 主体内容区域</strong></p>
    </div>
  </div>
  <div id="navigation">
    <p><strong>2. 导航栏</strong></p>
  </div>
  <div id="extra">
    <p><strong>3. 其他栏目</strong></p>
  </div>
  <div id="footer">
    <p>页脚区域</p>
  </div>
</div>
```

本案例版式设置导航栏与其他栏目并为一列固定在右侧，主栏目以液态方式显示在左侧，实现主栏自适应页面宽度变化，而侧栏宽度固定不变的版式效果（如图 10.31 所示）。

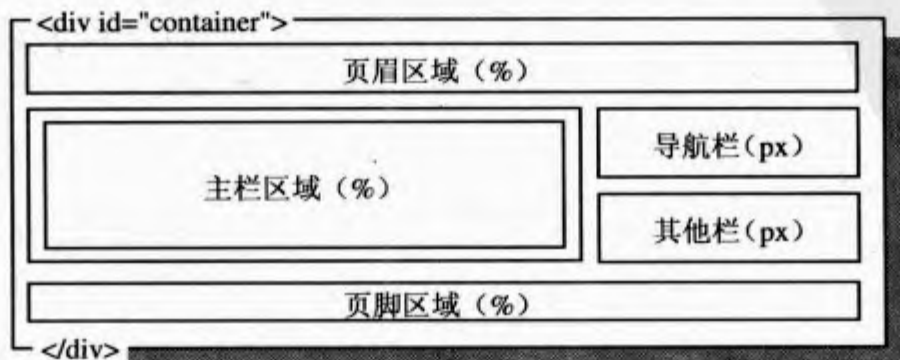


图 10.31 版式结构示意图

对于这类版式,可以采用 Dreamweaver CS3 所提供的方法来设计。不过我们讲解另一种更特殊的方法,它采用设置主栏和侧栏都浮动显示的方法来设计。

如果完全使用浮动布局来设计主栏自适应、侧栏固定的版式是存在很大难度的,因为百分比取值是一个不固定的宽度。让一个不固定宽度的栏目与一个固定宽度的栏目同时浮动在一行内,采用简单的方法是不行的。

这里设计主栏 100% 宽度,然后通过左外边距取负值强迫栏目偏移出一列的空间,最后把这个腾出的区域让给右侧浮动的侧栏,从而达到并列浮动显示的目的。

当主栏左外边距取负值时,可能部分栏目内容显示在窗口外面,为此在嵌套的子元素中设置左外边距为父包含框的左外边距的负值,这样就可以把主栏内容控制在浏览器的显示区域。如果我们定义 body 左外边距为 220 像素,这时就可以看到其中的设计技巧(如图 10.32 所示)。

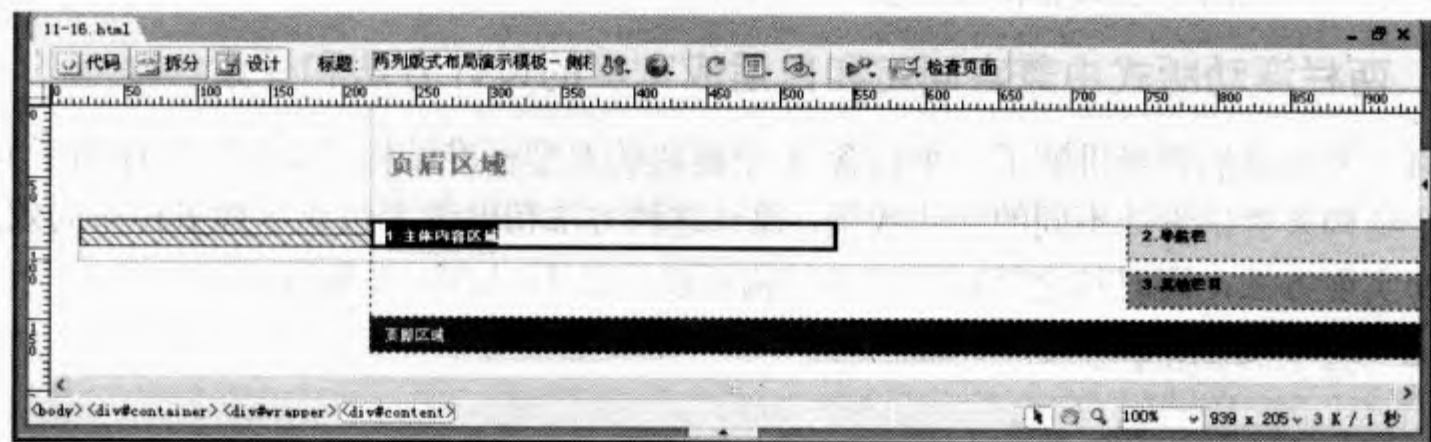


图 10.32 双浮动同行显示的奥秘

该版式主要结构的样式设计代码如下,所设计的版式效果如图 10.33 所示(原实例请参阅光盘实例)。

```
div#wrapper { /* 主栏外框 */
    float:left; /* 向左浮动 */
    width:100%; /* 液态宽度 */
    margin-left:-200px; /* 左侧外边距, 负值向左缩进 */
}
div#content { /* 主栏内框 */
    margin-left:200px; /* 左侧外边距, 正值填充缩进 */
}
div#navigation { /* 导航栏 */
    float:right; /* 向右浮动 */
    width:200px; /* 固定宽度 */
}
div#extra { /* 其他栏 */
    float:right; /* 向右浮动 */
    clear:right; /* 清除右侧浮动, 避免同行显示 */
    width:200px; /* 固定宽度 */
}
div#footer { /* 页眉区域 */
    clear:both; /* 清除两侧浮动, 强迫外框撑起 */
    width:100%; /* 宽度 */
}
```

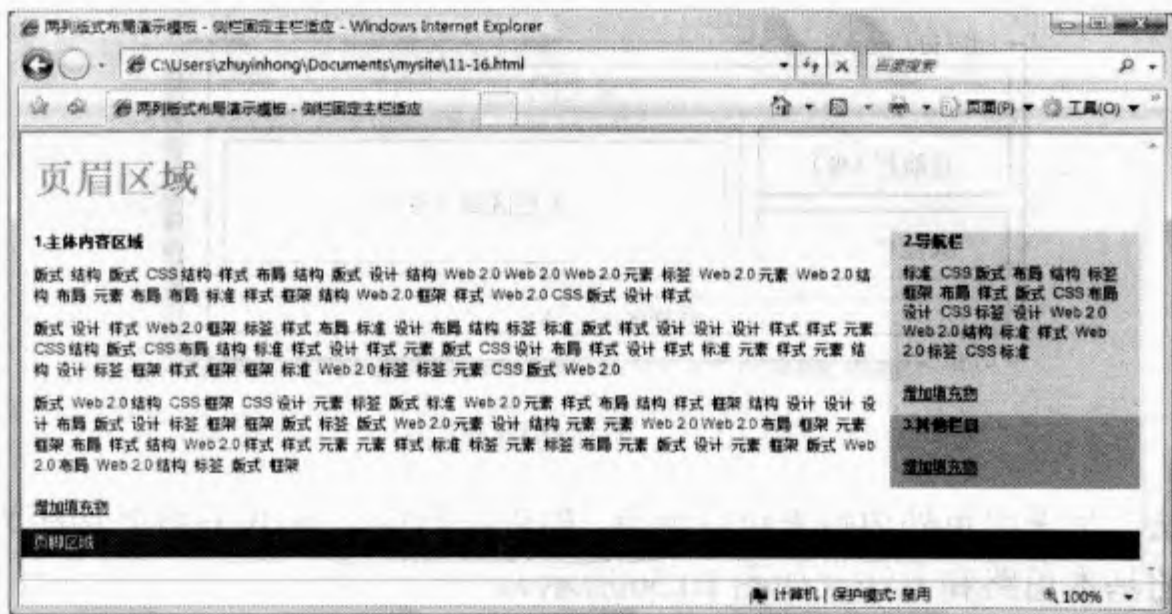



图 10.33 双浮动的布局效果

10.4.3 两栏浮动版式中自适应宽度的设计方法和探索

在两栏浮动版式中如果设置两列宽度都为自适应，那么设置起来会容易得多。例如，定义两栏版式中主栏向左浮动，宽度为 70%；导航栏向右浮动，宽度为 29.9%，所设计的效果如图 10.34 所示。

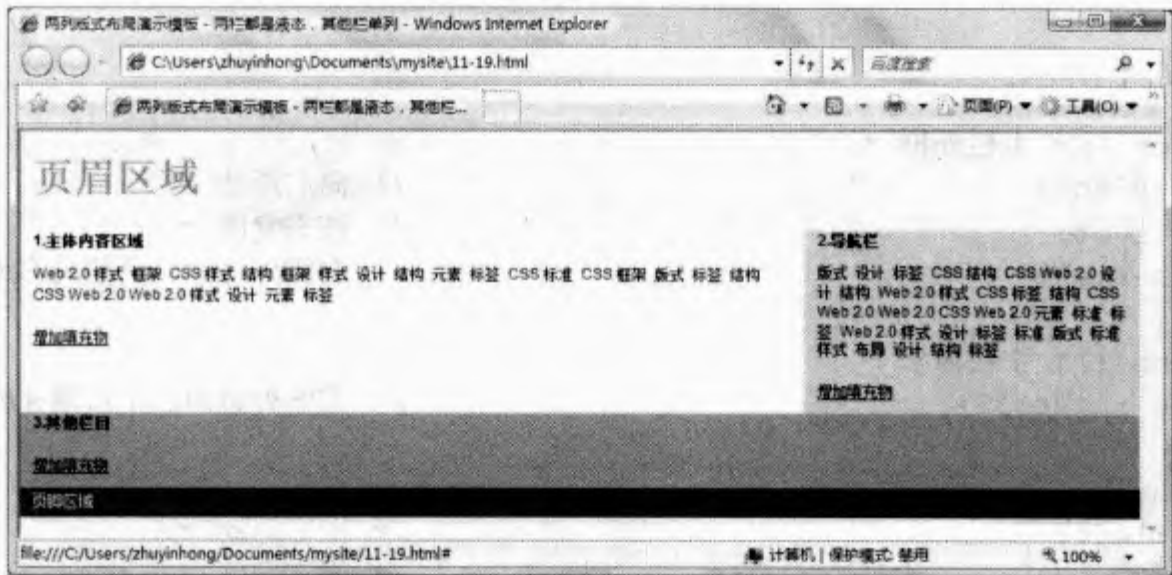


图 10.34 双浮动的布局效果

```
div#wrapper {  
    float:left;  
    width:70%  
}  
div#navigation {  
    float:right;  
    width:29.9%  
}  
div#extra {  
    clear:both;  
    width:100%  
}
```

/* 向左浮动 */
/* 百分比宽度 */
/* 向右浮动 */
/* 百分比宽度 */
/* 清除左右浮动 */
/* 满屏显示 */

下面我们模拟上节示例的设计方法来设计一个更精确的两栏浮动且自适应宽度的版式。本案例版式设置导航栏与其他栏目并为一系列固定在右侧，主栏目以液态方式显示在左侧，实现主栏自适应页面宽度变化，而侧栏宽度固定不变的版式效果（如图 10.35 所示）。

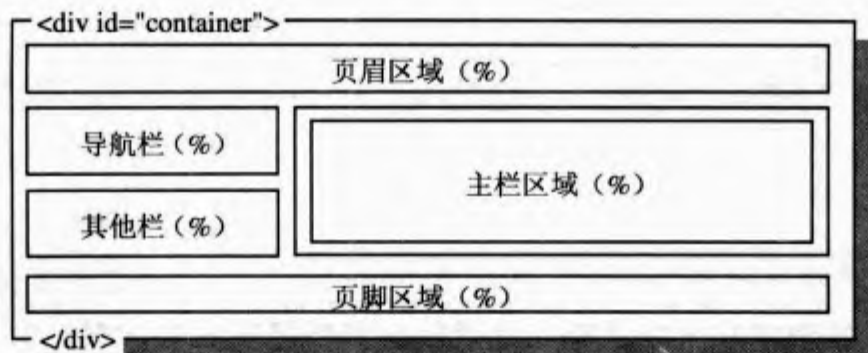


图 10.35 版式结构示意图

设计的方法也是采用负外边距来进行调节，如果我们定义 body 右侧外边距为 250 像素，则可以看到设计的基本思路和方法（如图 10.36 所示）。



图 10.36 设计过程图

核心样式如下所示：

```
div#wrapper { /* 主栏外框 */
    float:right; /* 向右浮动 */
    width:100%; /* 液态宽度 */
    margin-right:-33%; /* 右侧外边距，负值向右缩进 */
}
div#content { /* 主栏内框 */
    margin-right:33%; /* 右侧外边距，正值填充缩进 */
}
div#navigation { /* 导航栏 */
    float:left; /* 向右浮动 */
    width:32.9%; /* 固定宽度 */
}
div#extra { /* 其他栏 */
    float:left; /* 向左浮动 */
    clear:left; /* 清除左侧浮动，避免同行显示 */
    width:32.9%; /* 固定宽度 */
}
div#footer { /* 页眉区域 */
    clear:both; /* 清除两侧浮动，强迫外框撑起 */
    width:100%; /* 宽度 */
}
```

这时如果在 IE 7 或者其他标准浏览器中预览，则会发现浏览器窗口出现 y 轴滚动条（如图 10.37 所示），此时可以为 body 元素增加 overflow-x:hidden; 声明隐藏该滚动条，最后设计的效果如图 10.38 所示。

```
body {
    overflow-x:hidden;
}
```




图 10.37 在 FF 中出现的滚动条

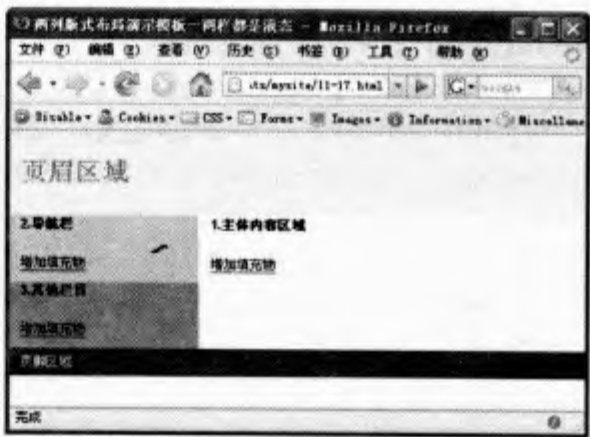


图 10.38 隐藏滚动条效果

10.4.4 两栏浮动版式中兼容性宽度的设计方法和探索

上面两节分别采用不同的设计思路来设计两栏浮动版式，下面我们继续就这个问题进行探讨，本节将介绍使用另一种思维来设计两栏浮动版式的技巧。

整个版式设置导航栏固定在左侧，主栏目以液态方式显示在右侧，以实现主栏自适应页面宽度变化，而其他栏目与页脚区域显示底部的版式效果（如图 10.39 所示）。

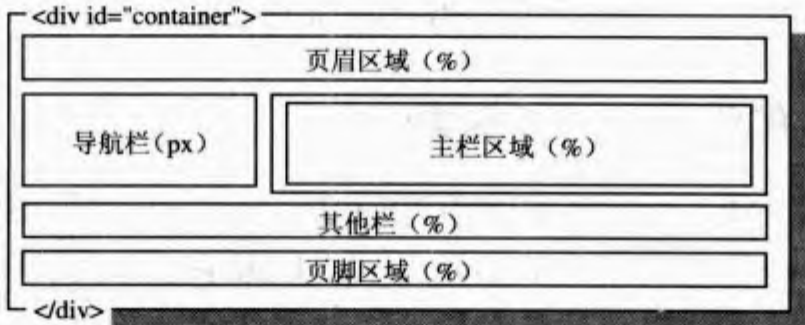


图 10.39 版式结构示意图

设计的方法是：让主栏（<div id="wrapper">）满屏显示，即取值为 100%，然后设置其包含的子元素（<div id="content">）左侧外边距为 200 像素，预留出一块区域。最后定义导航栏（<div id="navigation">）取值为-100%，也就是强制其从右侧的窗口外边移动到主栏左侧的预留区域内显示。设计的基本思路和方法如图 10.40 所示。

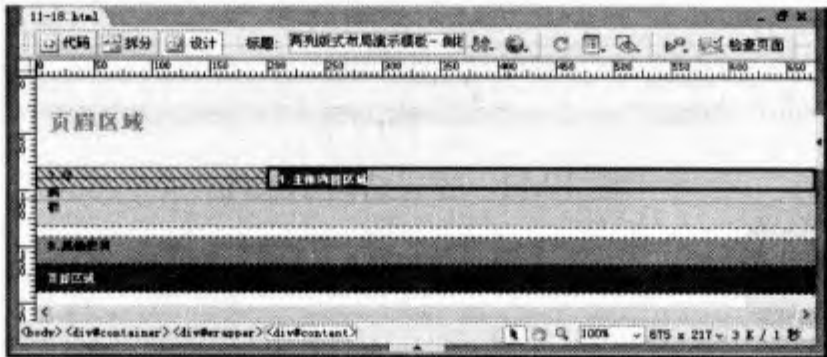


图 10.40 设计过程图

所设计版式的核心代码如下，在 IE 7 中预览效果如图 10.41 所示。

```
div#wrapper { /* 主栏外包含框 */
    float:left; /* 向左浮动 */
    width:100% /* 满屏宽度 */
}
div#content { /* 主栏内包含框 */
    margin-left:200px /* 左侧外边距，预留空间 */
}
```

```

}
div#navigation { /* 导航栏 */
    float:left; /* 向左浮动 */
    width:200px; /* 固定宽度，保持与主栏左侧的预留区域的宽度一致 */
    margin-left:-100% /* 通过取左外边距负值，向左强制移动到主栏左侧的预留区域 */
}
div#extra { /* 其他栏 */
    clear:left; /* 清除左右浮动 */
    width:100% /* 固定宽度 */
}
    
```

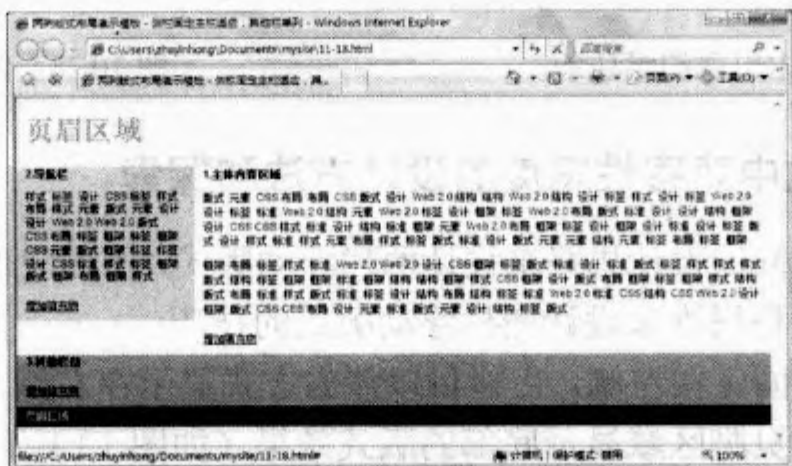


图 10.41 双浮动的布局效果

10.4.5 两栏固定版式的设计方法和探索

固定宽度的版式相对于液态版式也许要简单许多，因为每个板块的宽度都是固定的，不用考虑复杂的布局关系，只需要设置模块的显示方式，以及它的宽度即可。例如，假设我们设计一个外部包含框宽度固定、内部以自然流动显示，或者固定宽度并列浮动显示的示例（示意图如图 10.42 所示）。

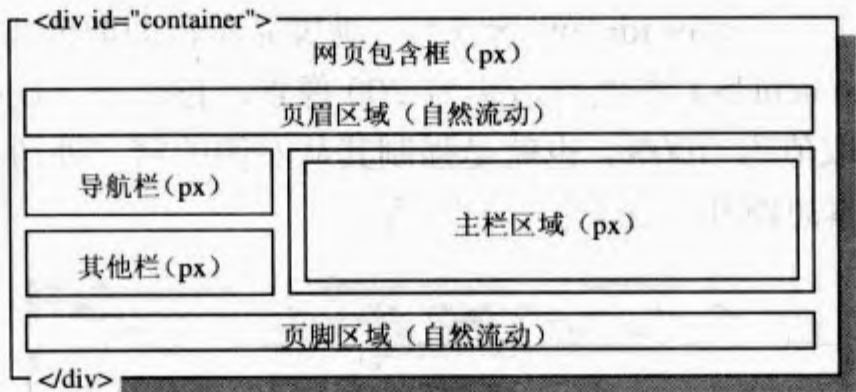


图 10.42 版式结构示意图

所设计的主体结构样式如下，设计效果如图 10.43 所示。

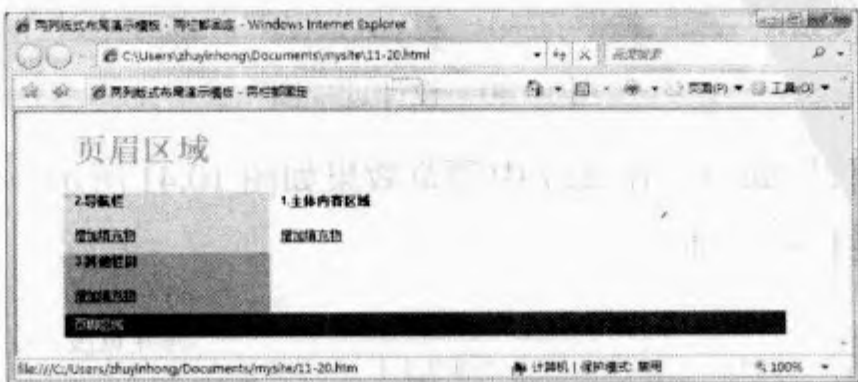


图 10.43 两栏固定版式设计效果


```
div#container {
    width:700px;
    margin:0 auto
}
div#content {
    float:right;
    width:500px
}
div#navigation {
    float:left;
    width:200px
}
div#extra {
    float:left;
    clear:left;
    width:200px
}
div#footer {
    clear:both;
    width:100%
}
```

/* 固定网页宽度 */
/* 网页居中对齐 */
/* 主栏向右浮动 */
/* 固定宽度 */
/* 导航栏向左浮动 */
/* 固定宽度 */
/* 其他栏向左浮动 */
/* 清除左侧浮动 */
/* 固定宽度 */
/* 清除左右浮动 */
/* 固定宽度 */

10.4.6 黑色理想网页布局实施实战

上几节讲解了两栏布局的不同版式类型，以及不同版式的设计技巧和实现过程。本节将借助这些布局知识来设计一个案例（如图 10.44 所示）。该案例以禅意花园的结构为载体，尝试使用 CSS 技术来控制禅意花园页面以两栏液态版式进行布局。整个页面以黑色为主色调，页眉页脚通过天蓝色背景图像进行点缀，结构为 3 行 3 列，版式类似于 Dreamweaver CS3 所提供的“2 列液态，左侧栏、标题和脚注”模板样式。

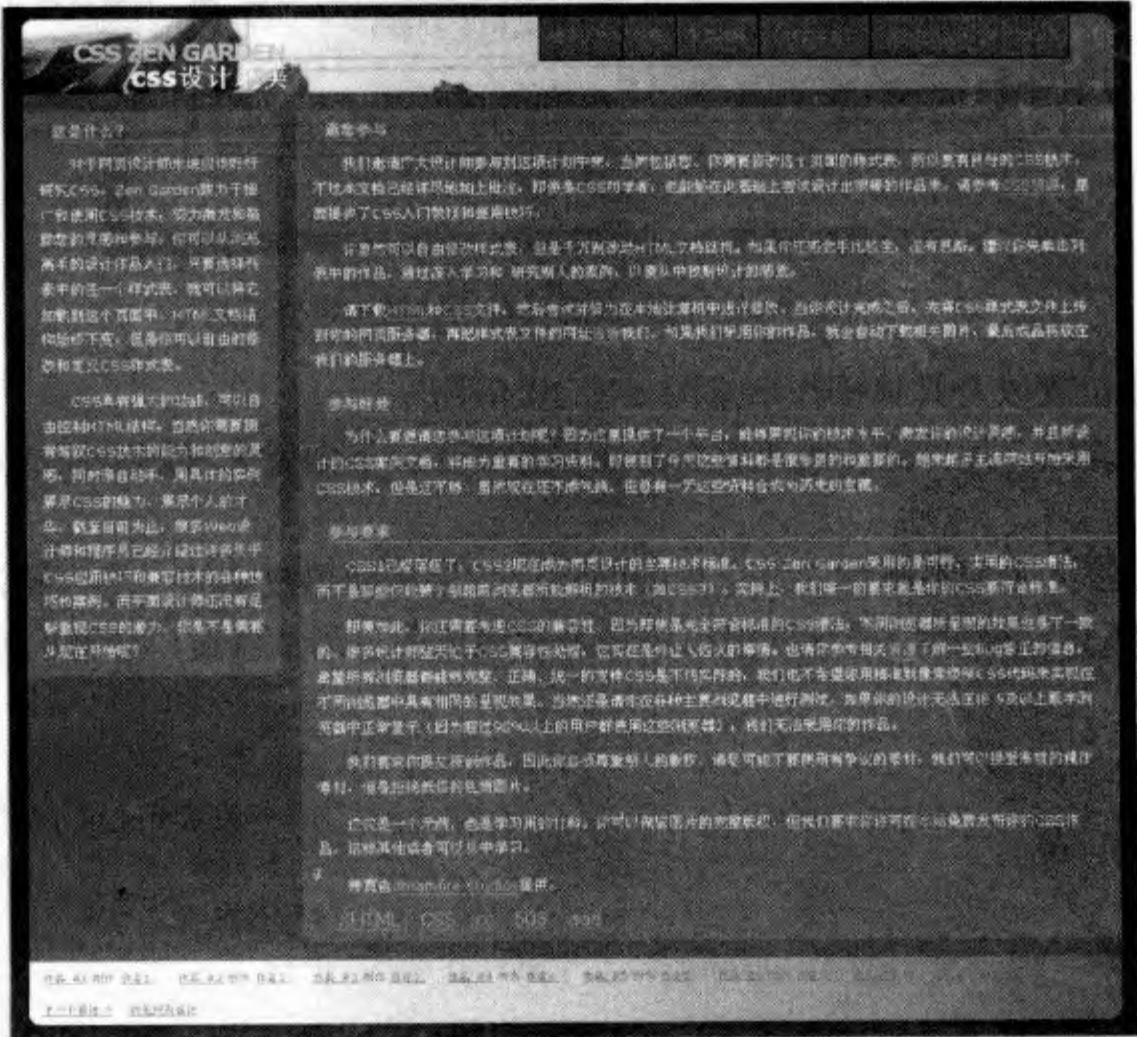


图 10.44 单列固定宽度布局效果图

禅意花园的结构包含 3 行并列子结构，为了能够设计出图 10.44 所示的版式效果，我们不妨把 3 个子模块分别独立为 1 行进行控制。其中中间的主体区域（<div id="supportingText">）又以两栏浮动布局的方式进行显示。考虑到第 1 行的包含框（<div id="intro">）作为网页标题，不适合显示大量的文本，因此隐藏其中的<div id="quickSummary">和<div id="preamble">元素，并把第 3 行页脚区域的<div id="lresources">子元素定位到第 1 行右上角区域。禅意花园的整个结构的布局思路如图 10.45 所示。

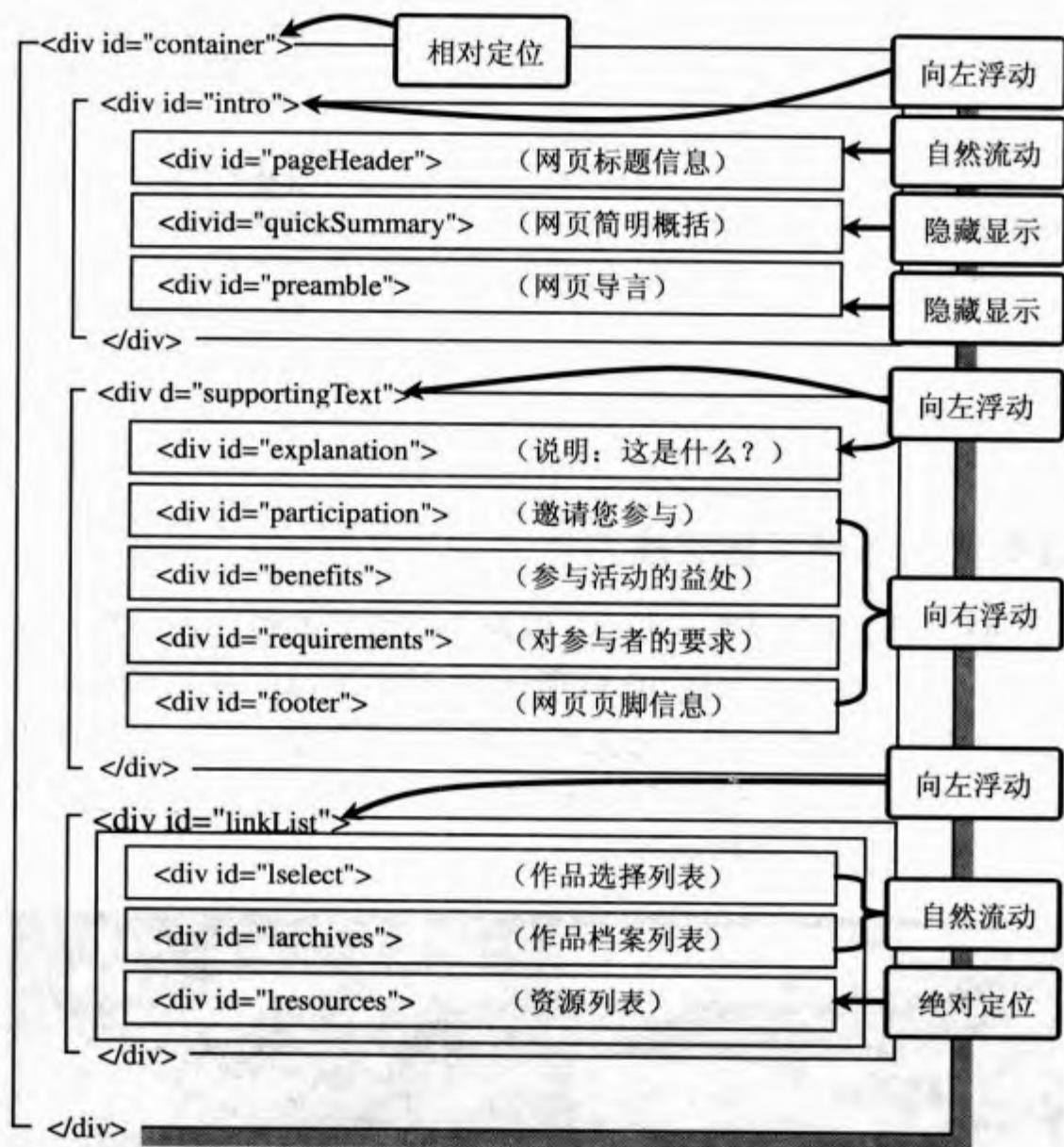


图 10.45 黑色理想页布局的结构定位思路

结构的摆放位置以及具体操作示意图如图 10.46 所示。

清楚了设计思路，具体操作起来就会感觉轻松多了，下面就来讲解页面的具体操作过程和设计细节。

第 1 步，设置页面基本属性。整个页面的设计效果犹如黑夜中的一块大幕，所以我们不妨定义页面以深灰色背景显示，通过调整页边距来营造页面与浏览器窗口分离的感觉（如图 10.47 所示）。

```
body { /* 页面基本属性 */
    margin: 1em 1em 1em 1em;           /* 设置页边距 */
    padding: 0px;                       /* 清除内边距 */
    background-color: #333;             /* 定义网页背景色 */
}
```

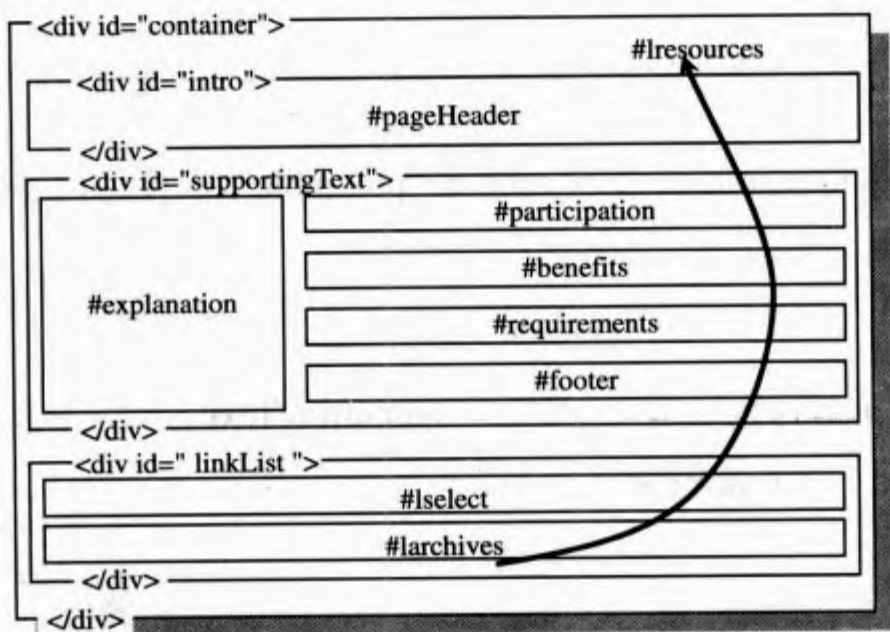



图 10.46 版式结构操作示意图

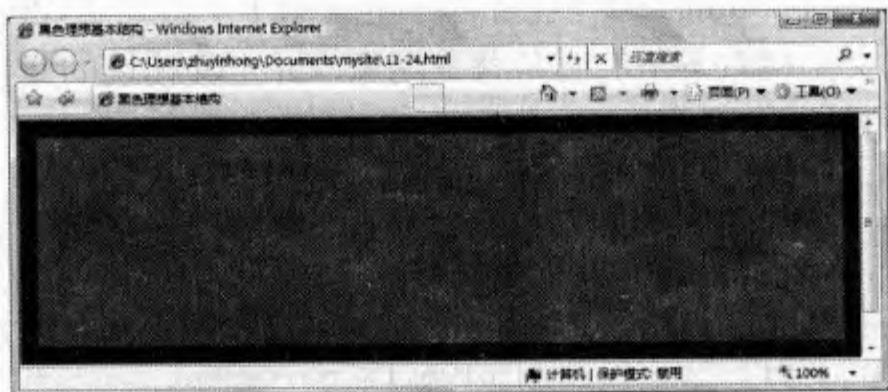


图 10.47 页面基本属性

第 2 步，设置页面包含框（<div id="container">）为相对定位，形成一个包含块，这样就可以在页面内定义绝对定位元素了。为了让该包含块能够自动包含内部所有浮动元素，不妨让它浮动显示，然后定义宽度为 100%，这时与包含框自动流动显示效果没有差异。

```
#container { /* 页面包含框样式 */
    float: left; /* 向左浮动 */
    width: 100%; /* 100%宽度 */
    position: relative; /* 相对定位 */
}
```

第 3 步，设计头部标题栏样式。设置包含框以浮动显示，这样就不再担心流动和浮动混合布局中可能会出现的错版问题，再通过背景图像来区分栏目的界限。

```
#intro { /* 网页标题包含框样式 */
    float: left; /* 向左浮动 */
    width: 100%; /* 向左浮动 */
    height: 4em; /* 向左浮动 */
    background: #25509F url(hdr.jpg) no-repeat bottom left; /* 向左浮动 */
}
```

通过<div id="intro">的子元素<div id="pageHeader">来定义标题栏目左上角的圆角。

```
#pageHeader { /* 第 1 子元素样式 */
    background: url(corner_tl.gif) left top no-repeat; /* 标题栏左上角的圆角 */
}
```

再通过<div id="pageHeader">子元素的一级标题定义标题栏右上角圆角。

```
#pageHeader h1 {
    background-image: url(corner_tr.gif) no-repeat: top right; /* 标题栏右上角的圆角 */
}
```

然后隐藏<div id="quickSummary">和<div id="preamble">子模块内容。

```
#quickSummary, #preamble {
    display: none; /* 隐藏子元素内容 */
}
```

第4步, 同样定义第2行包含框(<div id="supportingText">)浮动显示, 并定义灰色背景。

```
#supportingText { /* 主栏包含框样式 */
    float: left; /* 向左浮动 */
    width: 100%; /* 100%宽度 */
    padding: 1em 0 1em 0; /* 增加左右两侧内边距 */
    background-color: #666; /* 设置主体栏目背景色 */
}
```

第5步, 在<div id="supportingText">包含框中定义第1个子元素<div id="explanation">向左浮动。并利用该元素定义侧栏右下角的圆角背景图像, 并定义模块的背景色, 利用该元素所包含的标题h3定义右上角的圆角背景图像。

```
#explanation { /* 第1子元素样式 */
    float: left; /* 向左浮动 */
    width: 23.3%; /* 宽度 */
    margin-bottom: 1em; /* 底部外边距 */
    background: #777 url(corner_sub_br.gif) no-repeat bottom right; /* 背景色和图
像 */
}
#explanation h3 { /* 第1子元素的标题样式 */
    background: #777 url(corner_sub_tr.gif) no-repeat top right; /* 右上圆角 */
}
```

第6步, 定义<div id="supportingText">包含框中其他几个子元素向右浮动, 并设置圆角效果, 页面基本框架的设计效果如图10.48所示。

```
#participation, #benefits, #requirements, #footer { /* 公共样式 */
    float: right; /* 向右浮动 */
    width: 75%; /* 右栏宽度 */
}
#participation { /* 第2个子元素样式 */
    background: #777 url(corner_sub_tl.gif) no-repeat top left; /* 右栏左上角圆角图像 */
}
#benefits, #requirements { /* 第3、4个子元素样式 */
    background: #777; /* 右栏背景色 */
}
#footer { /* 第5个子元素样式 */
    padding-bottom: 12px; /* 底部内边距 */
    background: #777 url(corner_sub_bl.gif) no-repeat left bottom; /* 右栏左下角圆角 */
}
```

当为栏目同时定义背景图像和颜色时, 应该在一个声明中完成; 如果分开进行声明, 则背景色就会被背景图像声明的规则所覆盖。例如, 如果针对第5个子元素的样式进入如下定义, 则只能显示背景图像, 背景色将被覆盖。

```
#footer { /* 第5个子元素样式 */
    background: url(corner_sub_bl.gif) no-repeat left bottom; /* 背景图像 */
}
```



```
background:#777; /* 背景色 */
}
```

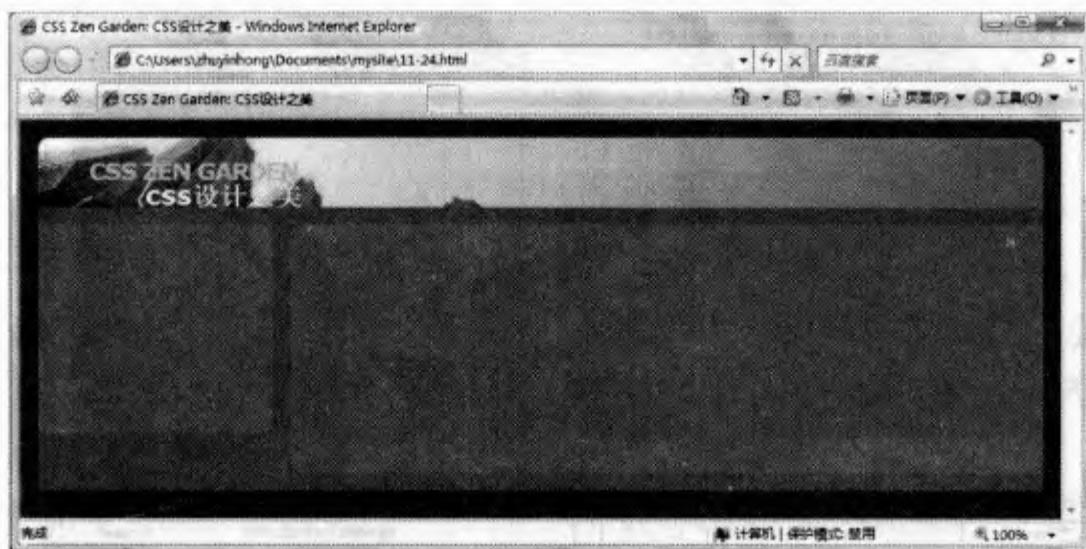


图 10.48 页面基本框架效果图

第 7 步, 定义页脚基本框架样式。通过外层包含框 (<div id="linkList">) 定义页脚区域的背景图像, 通过内层包含框 (<div id="linkList2">) 和第 3 个子元素 (<div id="lresources">) 定义两个底角的圆角背景图像, 所设计的效果如图 10.49 所示。

```
#linkList { /* 外包含框样式 */
    float: left; /* 向左浮动 */
    width: 100%; /* 宽度 */
    height: 6em; /* 高度 */
    line-height: 3em; /* 背景色 */
    background: #25509F url(hdr.jpg) repeat-y top left; /* 背景色和背景图像 */
}
#linkList2 { /* 内包含框样式 */
    float: left; /* 向左浮动 */
    width: 100%; /* 宽度 */
    height: 100%; /* 高度 */
    background: url(corner_bl.gif) no-repeat bottom left; /* 左上圆角图像 */
}
#larchives { /* 第 3 个子元素样式 */
    width: 100%; /* 宽度 */
    background: url(corner_br.gif) no-repeat bottom right; /* 右下圆角图像 */
}
```

隐藏每个子元素的标题。

```
#linkList h3 { /* 标题样式 */
    display: none; /* 隐藏元素 */
}
```

最后定义页脚菜单项的显示样式。

```
#linkList ul { /* 列表样式 */
    list-style-type: none; /* 清除列表符号 */
    margin: 0; /* 清除项目缩进 */
    padding: 0; /* 清除项目缩进 */
    width: 100%; /* 宽度 */
    clear: left; /* 清除浮动 */
}
#linkList li { /* 列表项目样式 */
    display: inline; /* 行内显示 */
}
```

padding-left:6px;

/* 增加左侧边距 */

}

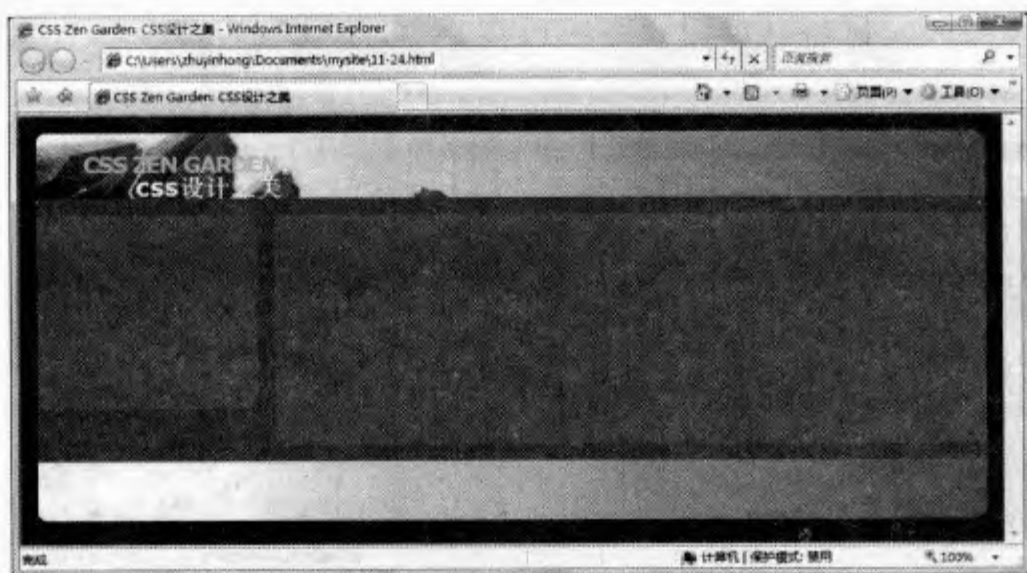


图 10.49 页面完整框架效果图

第 8 步, 设计导航栏。通过绝对定位的方式把<div id="lresources">子元素定位到页面标题栏的右上角, 并定义超链接的样式以及显示效果, 设计效果如图 10.50 所示。

```
#lresources { /* 定位导航条 */
    position:absolute;                /* 绝对定位 */
    top:0;                            /* 顶部距离 */
    right:4em;                        /* 右侧距离 */
}
#lresources a { /* 导航条超链接样式 */
    float:right;                      /* 向右浮动 */
    padding: 5px 10px 5px 10px;       /* 调整内边距 */
    text-decoration: none;            /* 清除超链接下划线 */
    background-color: #666;           /* 背景色 */
    border-bottom: 2px solid #333;    /* 底边框样式 */
    border-right: 1px solid #333;     /* 右边框样式 */
    border-left: 1px solid #333;      /* 左边框样式 */
}
#lresources a:hover, #larchives a.active { /* 鼠标经过超链接的样式 */
    padding-top: 10px;                /* 增加顶部内边距 */
    background-color: #333;           /* 加深背景色, 产生鼠标移过时凹陷的错觉 */
}
```

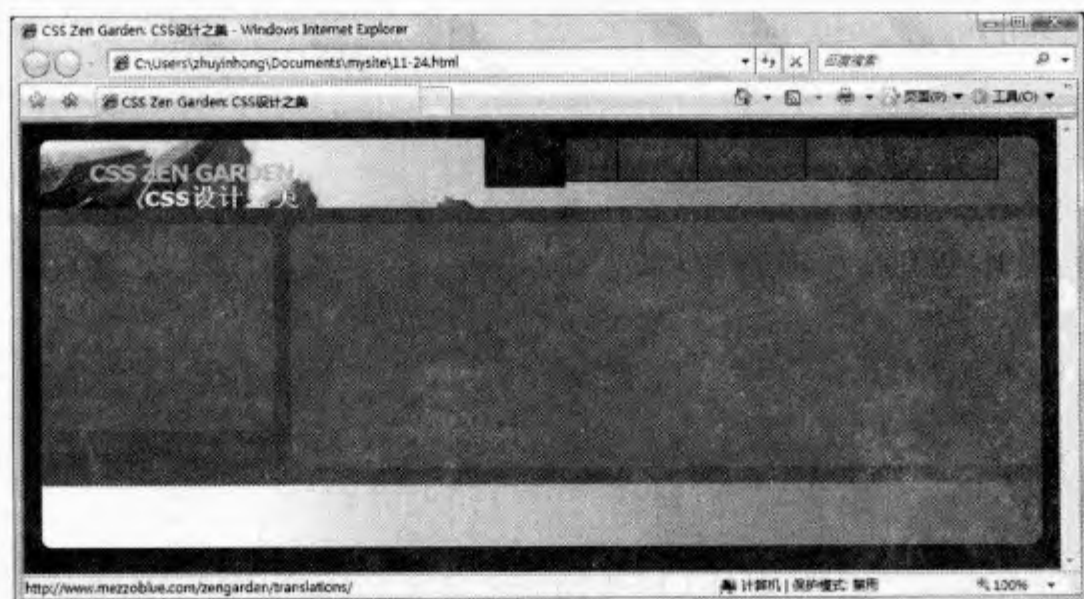


图 10.50 设计的导航条效果图

页面基本框架设计完毕，其他细节问题就不再详细讲解，感兴趣的读者可以参阅光盘示例。

10.5 三列版式布局

在 Dreamweaver CS3 提供的 CSS 布局模板中三列版式布局没有两列版式那么受到重视，仅提供了 10 种版式，这些版式可以分为 5 小类。与一列和两列不同，三列版式多出了一类绝对定位版式。具体列表如下：

- (1) 固定宽度布局（以像素为宽度单位）：“3 列固定”、“3 列固定，标题和脚注”。
 - (2) 弹性宽度布局（以 em 为宽度单位）：“3 列弹性”、“3 列弹性，标题和脚注”。
 - (3) 液态宽度布局（以百分比为宽度单位）：“3 列液态”、“3 列液态，标题和脚注”。
 - (4) 混合宽度布局（混合使用百分比和 em 为宽度单位）：“3 列混合”、“3 列混合，标题和脚注”。
 - (5) 绝对定位布局（以像素为宽度单位）：“3 列绝对定位”、“3 列绝对定位，标题和脚注”。
- 下面我们以这些版式为基础，详细讲解三列版式布局的一般规律和设计技巧。

10.5.1 三列版式的布局结构和设计方法

三列版式是企业、商业网站常用布局模式，这类布局能够在有限的版面内呈现更大的信息量，网页整个结构显得大方、开阔，符合人的视觉阅读需要。三列版式一般多采用一主两辅的版式进行布局（如图 10.51 所示），也可以以一主、一次和一辅（如图 10.52 所示）或者两主一辅（如图 10.53 所示）的形式排版网页内容。

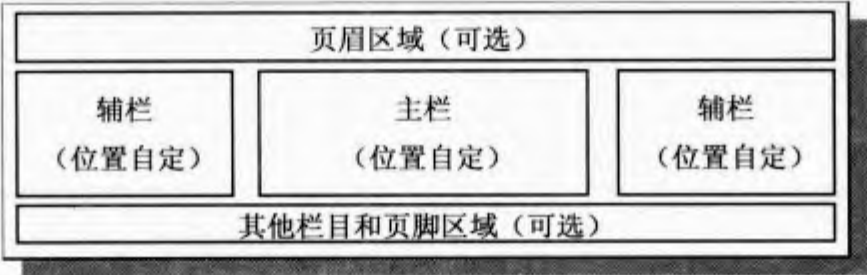


图 10.51 一主两辅的版式结构示意图

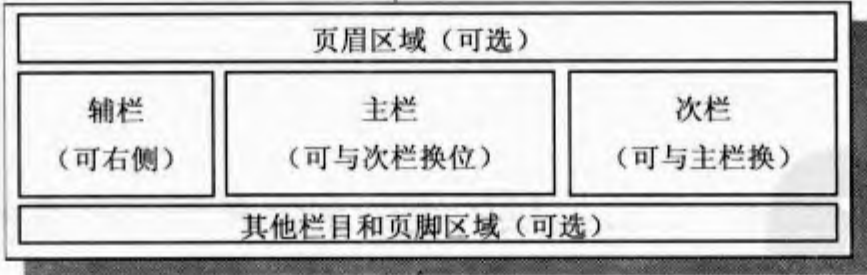


图 10.52 一主、一次和一辅的版式结构示意图

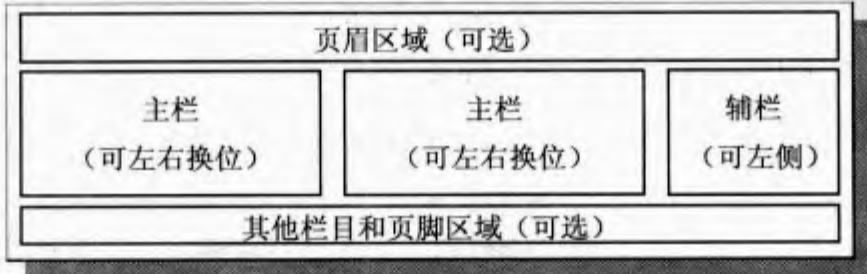


图 10.53 两主一辅的版式结构示意图

当读者选用两主一辅的版式结构时，一定要慎重，因为当两个主栏并列显示时，很容易使浏览者无所适从，同时会影响阅读的视线。此时不妨通过内容来区分主次，如一系列是详细内容，另一列是信息列表等。

在一主、一次和一辅的版式结构中，主栏显示网页重要信息，次栏负责显示次要提示性信息，或者重要信息列表，辅栏提供功能性的服务，有时可以根据需要在网页顶部和底部增加页眉和页脚，甚至还可以扩展其他栏目。

针对 Dreamweaver CS3 所提供的 10 种三列版式，包含了两种基本的网页结构。

第 1 种，仅包含网页主体内容和侧栏内容的结构，即整个网页仅包含 3 个独立的模块。

```
<div id="container">
  <div id="sidebar1">
    <h3>侧栏 1 内容</h3>
  </div>
  <div id="sidebar2">
    <h3>侧栏 2 内容</h3>
  </div>
  <div id="mainContent">
    <h1>主要内容 </h1>
  </div>
</div>
```

第 2 种，包含网页标题、侧栏内容、主体内容和脚注（网页页脚），即包含 5 个独立的模块。

```
<div id="container">
  <div id="header">
    <h1>标题</h1>
  </div>
  <div id="sidebar1">
    <h3>侧栏 1 内容</h3>
  </div>
  <div id="sidebar2">
    <h3>侧栏 2 内容</h3>
  </div>
  <div id="mainContent">
    <h1>主要内容 </h1>
  </div>
  <div id="footer">
    <p>脚注</p>
  </div>
</div>
```

10.5.2 解析 Dreamweaver CS3 提供的绝对定位布局及存在问题

由于三列布局与两列布局在设计思路上有很多相似之处，所以就不再重复类似讲解。下面重点探讨 Dreamweaver CS3 提供的绝对定位的布局方法。Dreamweaver CS3 提供的两个绝对定位的版式全部以固定宽度来设计，使用液态或弹性布局不适合绝对定位布局，但是仍然可以采用这种模式，在后面的章节中我们会涉及这种布局样式。

例如，针对“3 列绝对定位，标题和脚注”的版式，Dreamweaver CS3 设计的主要样式如下：

```
.thrColAbsHdr #container { /* 定义包含块 */
  position: relative; /* 允许两个侧栏相对于此容器放置，即定义包含块 */
  width: 780px; /* 使用比最大宽度（800px）小 20px 的宽度可显示浏览器界面元素，并避免出现水平滚动条 */
```



```
margin: 0 auto; /* 自动边距（与宽度一起）会将页面居中 */
text-align: left; /* 这将覆盖 body 元素上的“text-align: center”。 */
}
.thrColAbsHdr #sidebar1 { /* 定位侧栏 1 */
    position: absolute; /* 绝对定位 */
    top: 60px; /* 距离顶部的位置，高度应该等于或大于网页标题栏的高度 */
    left: 0; /* 距离左侧的位置 */
    width: 150px; /* 在符合标准的浏览器中或者在 Internet Explorer 中的标准模式下，此 div 的
    实际宽度除了包括宽度外，还包括填充和边框 */
    padding: 15px 10px 15px 20px; /* 填充（内边距）使 div 的内容与边缘保持一定的距离 */
}
.thrColAbsHdr #sidebar2 { /* 定位侧栏 2 */
    position: absolute; /* 绝对定位 */
    top: 60px; /* 距离顶部的位置，高度应该等于或大于网页标题栏的高度 */
    right: 0; /* 距离右侧的位置 */
    width: 160px; /* 栏目宽度 */
    padding: 15px 10px 15px 20px; /* 栏目内边距 */
}
.thrColAbsHdr #mainContent { /* 主栏基本样式 */
    margin: 0 200px; /* 此 div 元素的左边距和右边距会在页面两侧上创建两个外部栏。无论侧栏 div
    中包含多少内容，栏内的空间都将保留。 */
    padding: 0 10px; /* 请记住，填充是 div 方块内部的空间，边距则是 div 方块外部的空间 */
}
```

Dreamweaver CS3 在绝对定位的布局版式中采用了这样一种设计思路：把包含框设计为包含块，定义左右两侧侧栏为绝对定位布局，中间主栏以流动布局显示，通过调整主栏左右外边距，防止侧栏绝对定位时覆盖主栏内容。设计效果如图 10.54 所示。

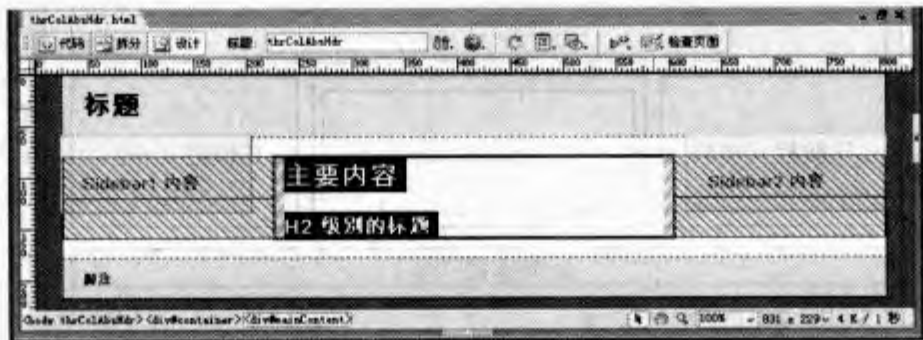


图 10.54 “3 列绝对定位，标题和脚注”版式效果

但是这种绝对定位的布局方法还是存在很大的风险，原因就是绝对定位元素脱离了正常的 HTML 文档流，所以绝对定位元素的大小变化不会影响相邻元素的位置。此时如果侧栏内容超出了主栏内容，就会出现覆盖页脚区域的现象（如图 10.55 所示）。

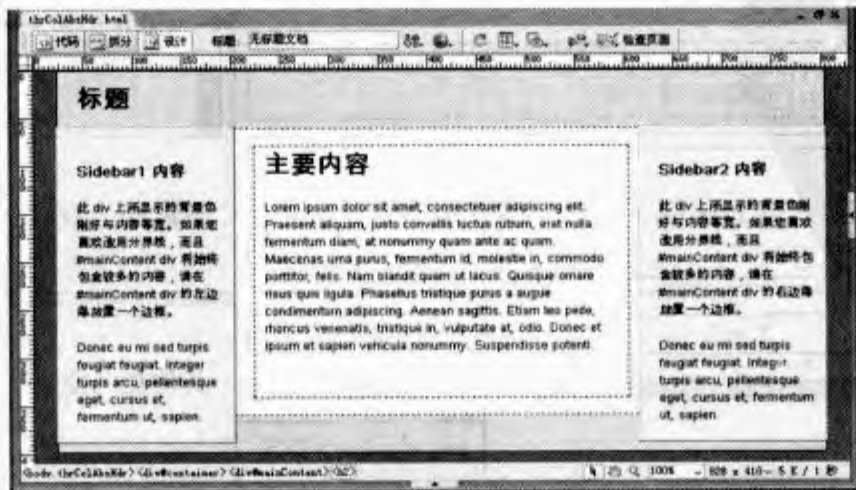


图 10.55 绝对定位布局可能出现的覆盖现象

对于这类问题，可以有两种被动的解决方法：一种方法是保证主栏的高度始终大于侧栏，当侧栏内容固定，可以通过定义主栏最小高度的方法来实现；另一种方法就是把页脚区域放置在主栏内的底部，这样就避免了被覆盖的危险。

10.5.3 三列液态版式的设计新方法和探索

让我们从 Dreamweaver CS3 的 CSS 设计模板中走出来，来研究一个有趣的话题：三列都能够自适应的版式新设计方法。对于这类版式布局，一般的设想是为每列各取百分之几的宽度，然后各列总宽度不超过 100% 即可。例如，设计类似如下的布局样式：

```
div#wrapper { /* 主栏基本样式 */
    float:left;           /* 向左浮动 */
    width:50%;            /* 百分比宽度 */
}
div#navigation { /* 导航栏基本样式 */
    float:left;           /* 向左浮动 */
    width:25%;            /* 百分比宽度 */
}
div#extra { /* 其他栏基本样式 */
    float:left;           /* 向左浮动 */
    width:25%;            /* 百分比宽度 */
}
```

本案例网页结构继续沿用上一节的模板示例结构。通过浮动布局的方法，以百分比为单位来设置栏目的宽度，版式结构示意图如图 10.56 所示。

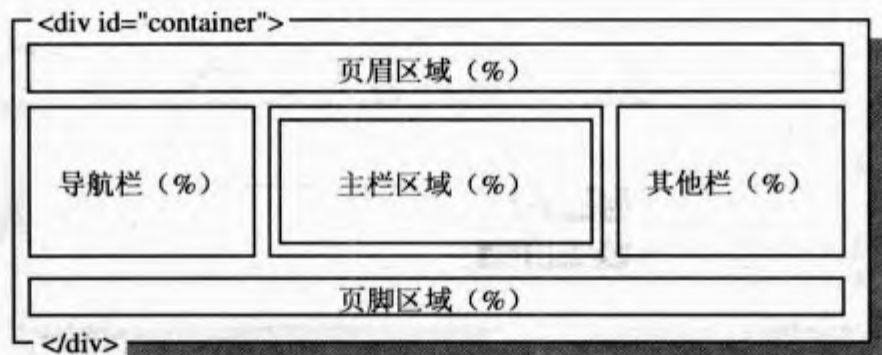


图 10.56 三列液态版式结构示意图

与上一大节中介绍的方法一样，本示例也是采用负外边距的方法来进行设计，这里设计三列都向左浮动，然后通过负外边距来定位每列的显示位置。布局示意图如图 10.57 所示。

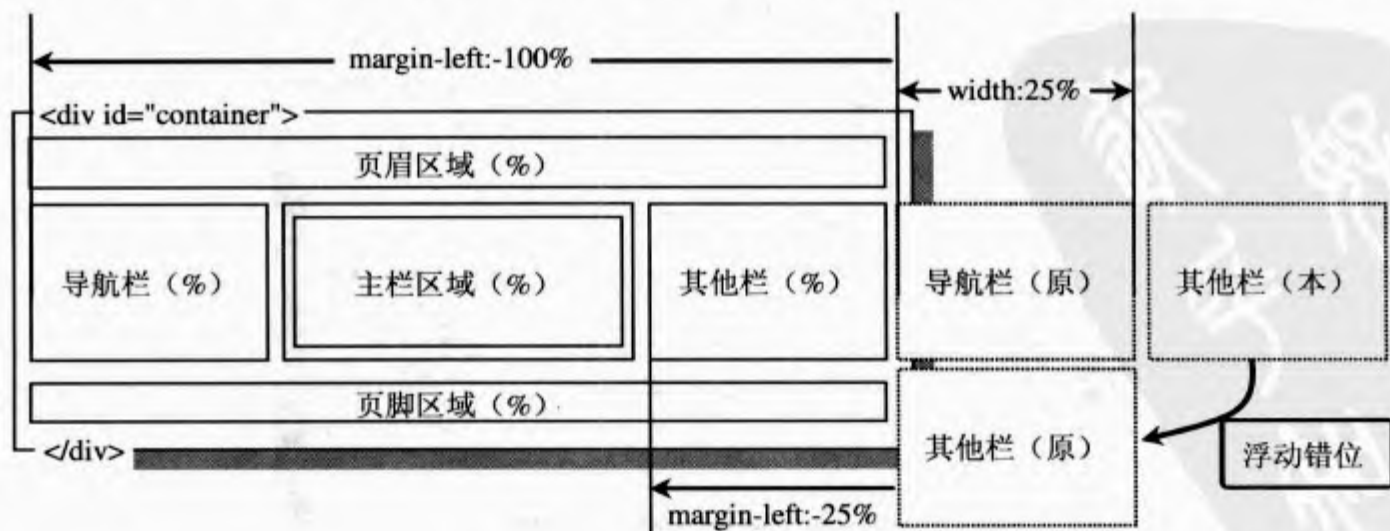


图 10.57 三列液态版式布局示意图

注意, 虽然其他栏目在不受外界干扰的情况会浮动在导航栏的右侧, 但是由于并列浮动的总宽度超出了窗口宽度, 会发生错位现象。如果没有负外边距的影响, 则会显示在第 2 行的位置, 通过外边距取负值, 强迫它们显示在主栏区域的上面。核心样式如下所示:

```
div#wrapper { /* 主栏外包含框基本样式 */
    float:left; /* 向左浮动 */
    width:100% /* 百分比宽度 */
}
div#content { /* 主栏内包含框基本样式 */
    margin: 0 25% /* 在左右两侧预留侧栏空间 */
}
div#navigation { /* 导航栏基本样式 */
    float:left; /* 向左浮动 */
    width:25%; /* 百分比宽度 */
    margin-left:-100% /* 左外边距取负值进行定位 */
}
div#extra { /* 其他栏基本样式 */
    float:left; /* 向左浮动 */
    width:25%; /* 百分比宽度 */
    margin-left:-25% /* 左外边距取负值进行定位 */
}
div#footer { /* 页脚包含框样式 */
    clear:left; /* 清除左右浮动 */
    width:100% /* 百分比宽度 */
}
```

三列液态布局的版式设计效果如图 10.58 所示。

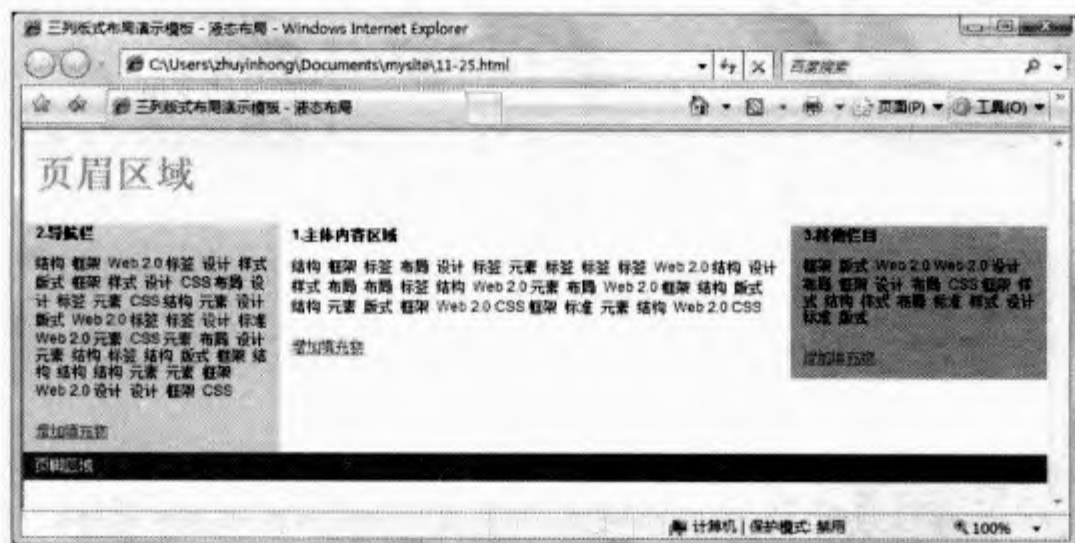


图 10.58 三列液态版式的布局效果

以同样的设计方法, 如果设置侧栏负边距为其他值, 则可以设置不同的版式效果。例如, 如果在上面示例基础上, 设置主栏右侧外边距为 50%, 定义导航栏左外边距负值为-50%, 则会显示图 10.59 所示的效果。

```
div#content { /* 主栏内包含框基本样式 */
    margin-right: 50% /* 右侧外边距 */
}
div#navigation { /* 导航栏包含框样式 */
    float:left; /* 向左浮动 */
    width:25%; /* 百分比宽度 */
    margin-left:-50% /* 左侧负边距 */
}
```

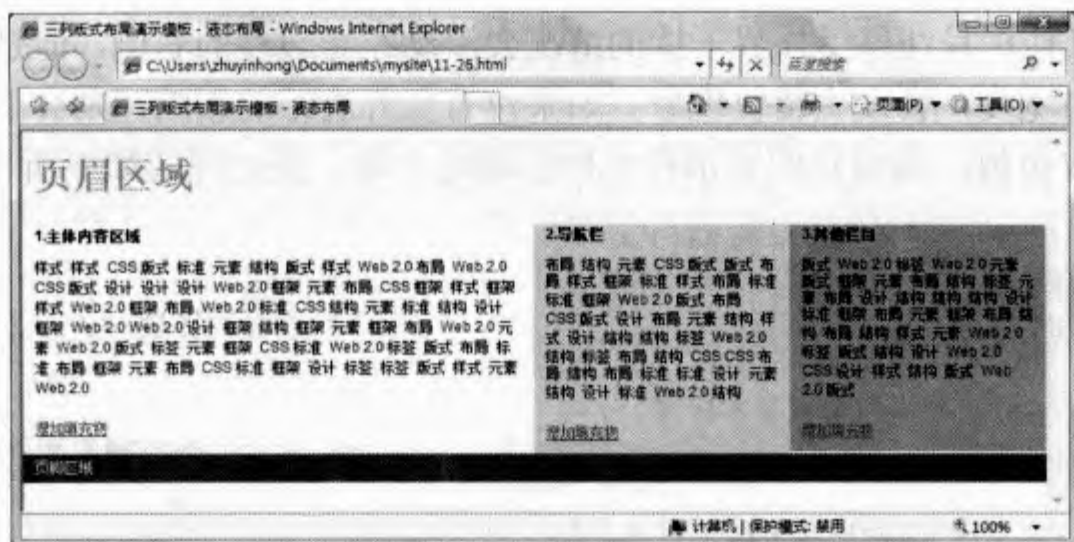


图 10.59 三列液态版式的布局效果

同样的道理，如果稍稍改变这几个包含框的外边距，会发现网页版式又发生了新的变化。例如，把主栏包含框的左外边距设置为 50%，通过负外边距让导航栏包含框向左移动 75% 的距离，而让其他栏目移动 100% 的距离，则会显示图 10.60 所示的效果。

```
div#content { /* 主栏内包含框的基本样式 */
    margin-left: 50%                               /* 左侧外边距 */
}
div#navigation { /* 导航栏包含框的基本样式 */
    float: left;                                   /* 向左浮动 */
    width: 25%;                                    /* 百分比宽度 */
    margin-left: -75%                               /* 左侧负边距 */
}
div#extra { /* 其他栏包含框的基本样式 */
    float: left;                                   /* 向左浮动 */
    width: 25%;                                    /* 百分比宽度 */
    margin-left: -100%                              /* 左侧负边距 */
}
```

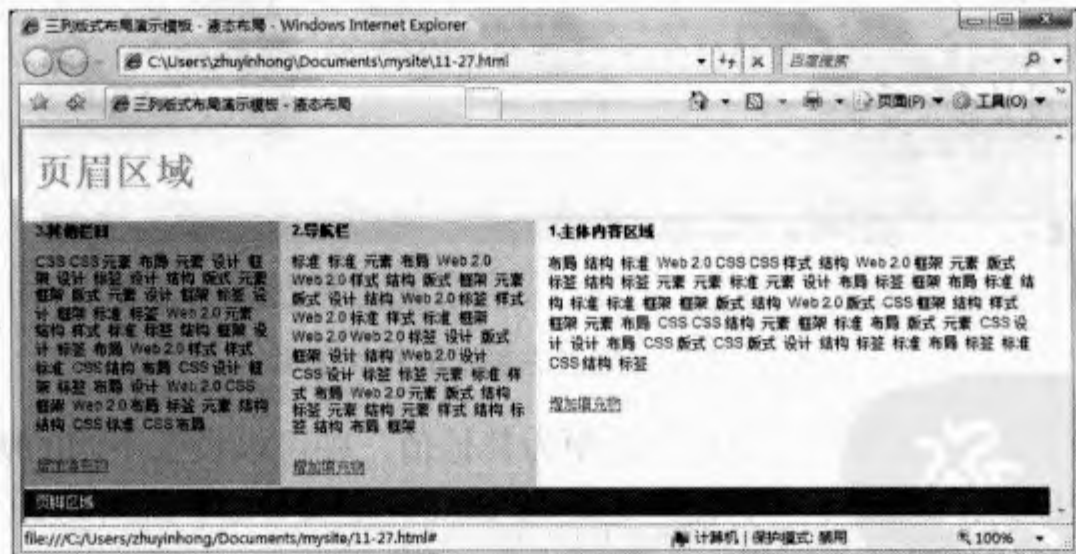


图 10.60 三列液态版式的布局效果

这样的设计可能还有很多种，正如玩积木一样，能够搭建出很多设计新颖的版式效果。

10.5.4 三列固定版式设计方法和探索

也许经历了三列自适应布局的考验之后，你会发现固定宽度布局已经不在话下了。是的，如果为每列设置一个具体的宽度值是很简单，但是本节将介绍另一种新颖的方法，保证让你的思路豁然开朗。

设计思路依然采用上节讲解的负外边距的方法，这样设计的好处就是可以避免不必要的错位问题发生。核心样式代码如下：

```
div#wrapper { /* 主栏外包含框基本样式 */
    float:left; /* 向左浮动 */
    width:100%; /* 百分比宽度 */
}
div#content { /* 主栏内包含框基本样式 */
    margin: 0 150px /* 通过外边距预留侧栏空间 */
}
div#navigation { /* 导航栏包含框基本样式 */
    float:left; /* 向左浮动 */
    width:150px; /* 固定宽度 */
    margin-left:-700px /* 负外边距法向左移动 */
}
div#extra { /* 其他栏包含框基本样式 */
    float:left; /* 向左浮动 */
    width:150px; /* 固定宽度 */
    margin-left:-150px /* 负外边距法向左移动 */
}
```

上面核心样式设计的模板效果如图 10.61 所示。当然，仍然可以如同变戏法一样推演出不同的版式效果。例如，分别改写如下几个样式的参数值，可以看到另一种布局效果，如图 10.62 所示。

```
div#content { /* 主栏内包含框基本样式 */
    margin-left: 300px /* 通过外边距预留侧栏空间 */
}
div#navigation { /* 导航栏包含框基本样式 */
    margin-left:-700px /* 负外边距法向左移动 */
}
div#extra { /* 其他栏包含框基本样式 */
    margin-left:-550px /* 负外边距法向左移动 */
}
```

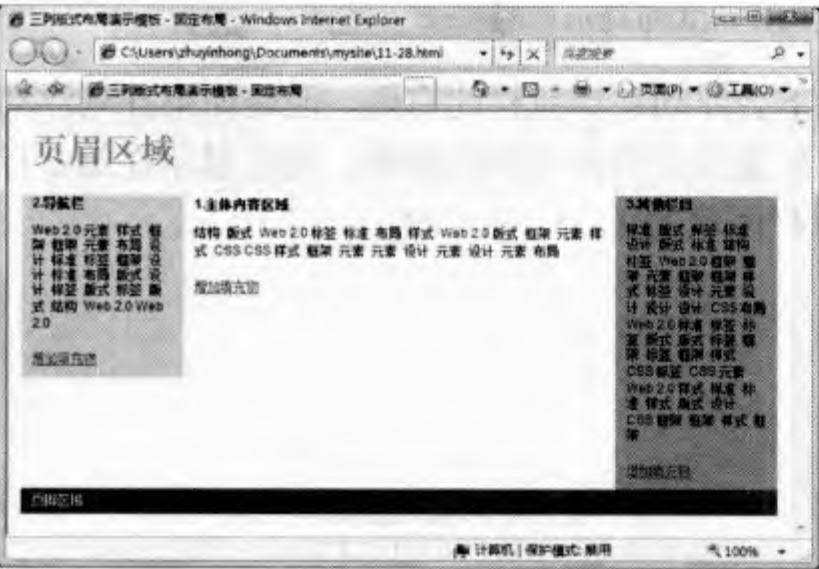


图 10.61 三列固定版式的布局效果

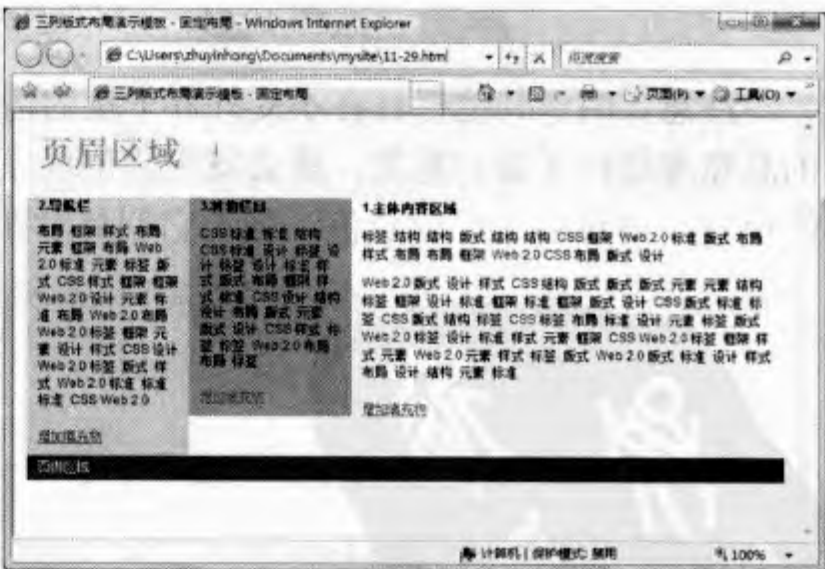


图 10.62 三列固定版式的布局效果

10.5.5 一列液态两列固定版式设计方法和探索

单纯的液态或者固定版式布局相对来说都比较好控制，但是如果要设计一列液态两列固定的版式可能就需要动点脑筋。不过如果灵活掌握了负外边距在网页布局中的技巧，这种复杂布

局也就是小菜一碟了。

下面来看看一列液态两列固定的版式是如何设计的。这种布局版式适合页面两侧栏都是一些服务性的版块，主栏显示网页的主要信息，不适合用于双主栏或一主一次栏目的页面。

本案例网页结构继续沿用上一节的模板示例结构。通过浮动布局的方法，以百分比和像素为单位来设置栏目的宽度，版式结构示意图如图 10.63 所示。

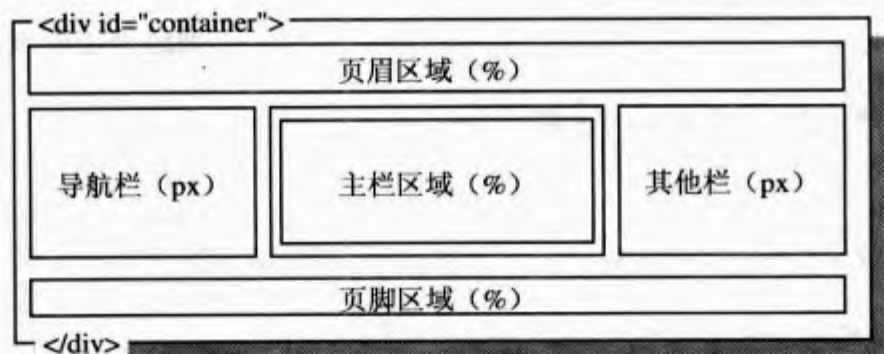


图 10.63 一列液态两列固定版式结构示意图

要定义导航栏和其他栏宽度固定，不妨选用像素为单位，对于主栏则可以采用百分比单位，然后通过负外边距来定位每列的显示位置。布局示意图如图 10.64 所示。

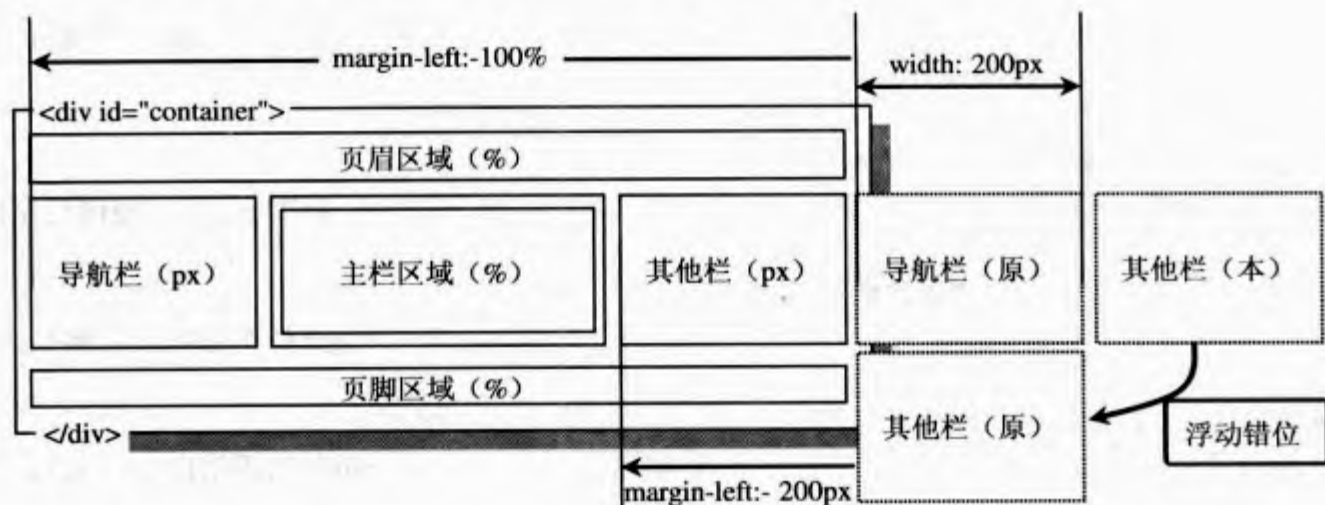


图 10.64 一列液态两列固定版式布局示意图

注意，由于其他栏目在不受外界干扰的情况下会浮动在导航栏的右侧，但是如果并列浮动的总宽度超出了窗口宽度，就会发生错位现象。如果没有负外边距的影响，则会显示在第 2 行的位置，通过外边距取负值，强迫它们显示在主栏区域的上面。核心样式如下所示。

```
div#wrapper { /* 主栏外包含框基本样式 */
    float:left; /* 向左浮动 */
    width:100% /* 百分比宽度 */
}
div#content { /* 主栏内包含框基本样式 */
    margin: 0 200px /* 在左右两侧预留侧栏空间 */
}
div#navigation { /* 导航栏基本样式 */
    float:left; /* 向左浮动 */
    width:200px; /* 固定宽度 */
    margin-left:-100% /* 左外边距取负值进行定位 */
}
div#extra { /* 其他栏基本样式 */
    float:left; /* 向左浮动 */
    width:200px; /* 固定宽度 */
}
```



```
margin-left:-200px
}
/* 左外边距取负值进行定位 */
```

一列液态两列固定版式的布局效果如图 10.65 所示。实际上通过上面几个示例的不断练习，相信你也能够推演出很多类似的版式效果。例如，如果分别调整侧栏和主栏的外边距取值，会得到图 10.66 所示的效果。

```
div#content { /* 主栏外包含框基本样式 */
margin-right: 400px
}
/* 通过左右外边距预留侧栏空间 */
div#navigation { /* 导航栏基本样式 */
margin-left:-200px
}
/*左外边距取负值进行精确定位*/
div#extra { /* 其他栏基本样式 */
margin-left:-400px
}
/*左外边距取负值进行精确定位*/
```

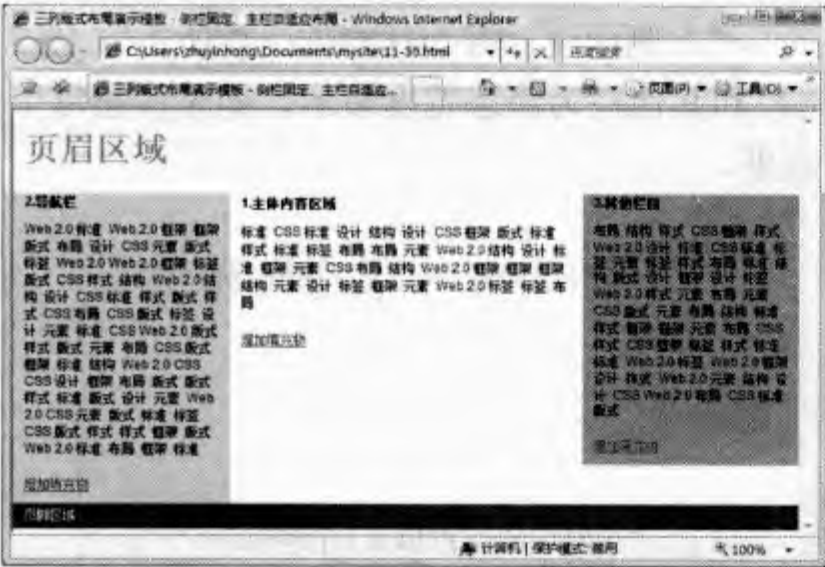


图 10.65 一列液态两列固定版式的布局效果

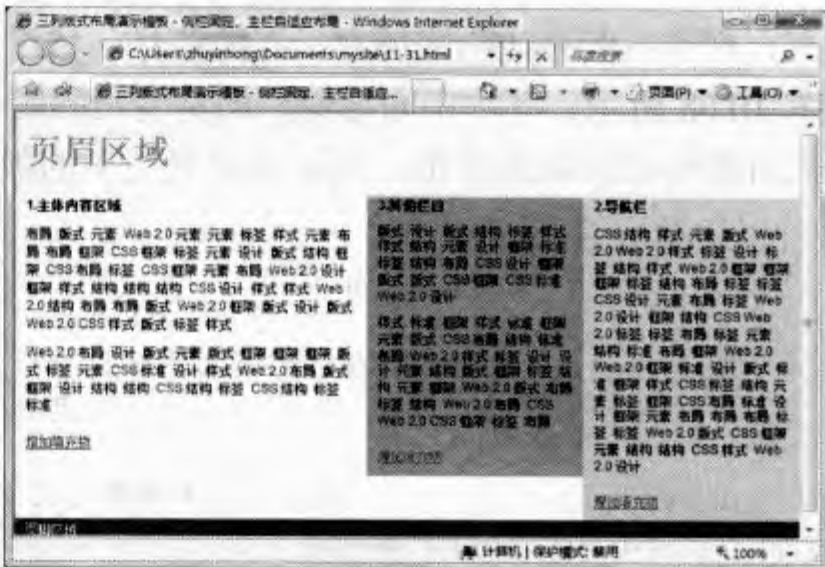


图 10.66 一列液态两列固定版式的布局效果

10.5.6 二列液态一列固定版式设计方法和探索

与一列液态两列固定版式相比，二列液态一列固定版式似乎显得多余，不过当设计一个双主题的页面，或者两列栏目都很重要的页面时，使用二列液态一列固定版式进行布局会让页面更具灵活性。虽然在设计思路二列液态一列固定版式与一列液态两列固定版式大同小异，相信通过这样单独的分解，更能够引起你的重视和思考。

如果你还没有阅读下面的内容就能够预知到二列液态一列固定版式的大致设计思路，那说明你对于渗透在本章中的负外边距布局法已经心领神会了。现在就直接讲解核心代码的设计思路吧。

该版式的基本思路是：首先定义主栏外包含框宽度为 100%，即占据整个窗口。然后再通过左右外边距来定义两侧空白区域，预留给侧栏占用。在设计外边距时，一侧采用百分比单位，另一侧采用像素为单位，这样就可以设计出两列宽度是液态的，另一列是固定的。最后再通过负外边距来定位侧栏的显示位置。

```
div#wrapper { /* 主栏外包含框基本样式 */
float:left;
width:100%
}
/* 向左浮动 */
/* 百分比宽度 */
div#content { /* 主栏内包含框基本样式 */
margin: 0 33% 0 200px
}
/* 定义左右两侧外边距，注意不同的取值单位 */
```

```
}
div#navigation {/* 导航栏包含框基本样式 */
    float:left;
    width:200px;
    margin-left:-100%
}
div#extra {/* 其他栏包含框基本样式 */
    float:left;
    width:33%;
    margin-left:-33%
}
```

/* 向左浮动 */
/* 固定宽度 */
/*左外边距取负值进行精确定位*/

/* 向左浮动 */
/* 百分比宽度 */
/*左外边距取负值进行精确定位*/

所设计的模板效果如图 10.67 所示。当然，并不是只能够让侧栏进行取负外边距来进行定位，完全可以让主栏取负外边距进行定位，其他栏自然浮动。例如，修改其中的核心代码，让主栏外包含框向左取负值偏移 25% 的宽度，也就是隐藏主栏外框左侧 25% 的宽度，然后通过内框来调整包含内容的显示位置，使其显示在窗口内，最后定义导航栏列左外边距取负值覆盖在主栏的右侧外边距区域上。其他栏目自然浮动在主栏右侧即可，核心代码如下，所得效果如图 10.68 所示。其中中间导航栏的宽度是固定，主栏和其他栏为液态宽度显示，如果调整每个栏目的外边距取值单位，还可以定义更多的版式效果。

```
div#wrapper {/* 主栏外包含框基本样式 */
    margin-left:-25%
}
div#content {/* 主栏内包含框基本样式 */
    margin: 0 200px 0 25% /* 定义左右两侧外边距，注意不同的取值单位 */
}
div#navigation {/* 导航栏包含框基本样式 */
    margin-left:-200px
}
div#extra {/* 其他栏包含框基本样式 */
    width:25%
}
```

/*左外边距取负值进行精确定位*/

/* 定义左右两侧外边距，注意不同的取值单位 */

/*左外边距取负值进行精确定位*/

/* 百分比宽度 */

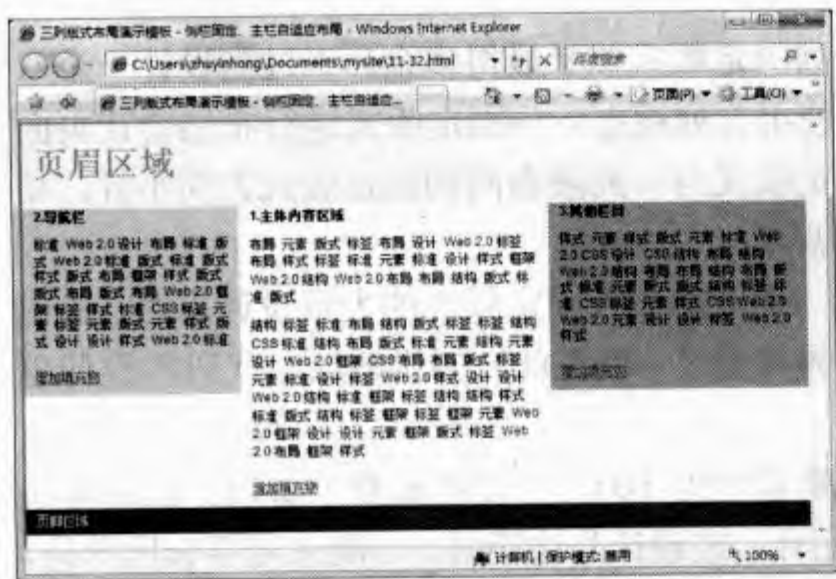


图 10.67 两列液态一列固定版式的布局效果

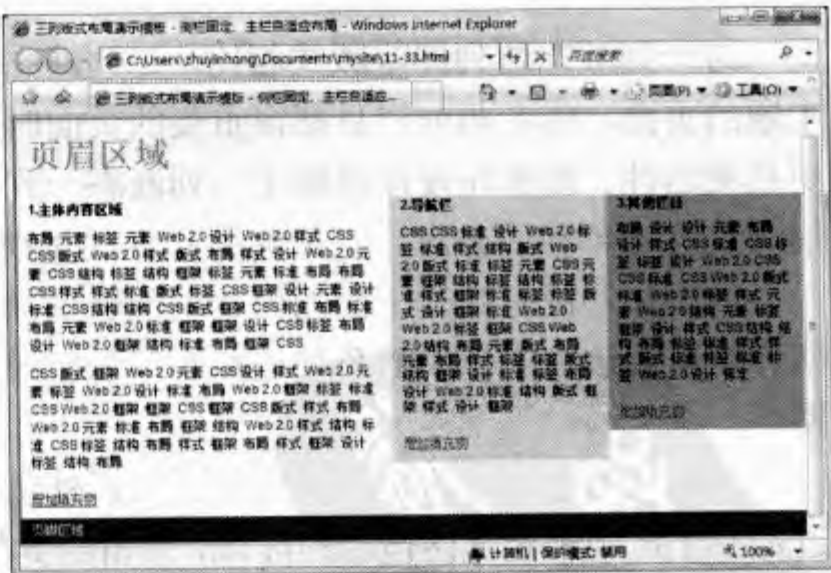


图 10.68 两列液态一列固定版式的布局效果

10.5.7 山鹰之美网页布局实战

从布局演练转向实战，下面来讲解一个 3 列液态布局的实例，希望通过这个实例来帮助用户更深刻的认识和体验网页布局之道。仍然以禅意花园的结构为载体进行讲解，整个页面的结构布局示意图如图 10.69 所示。

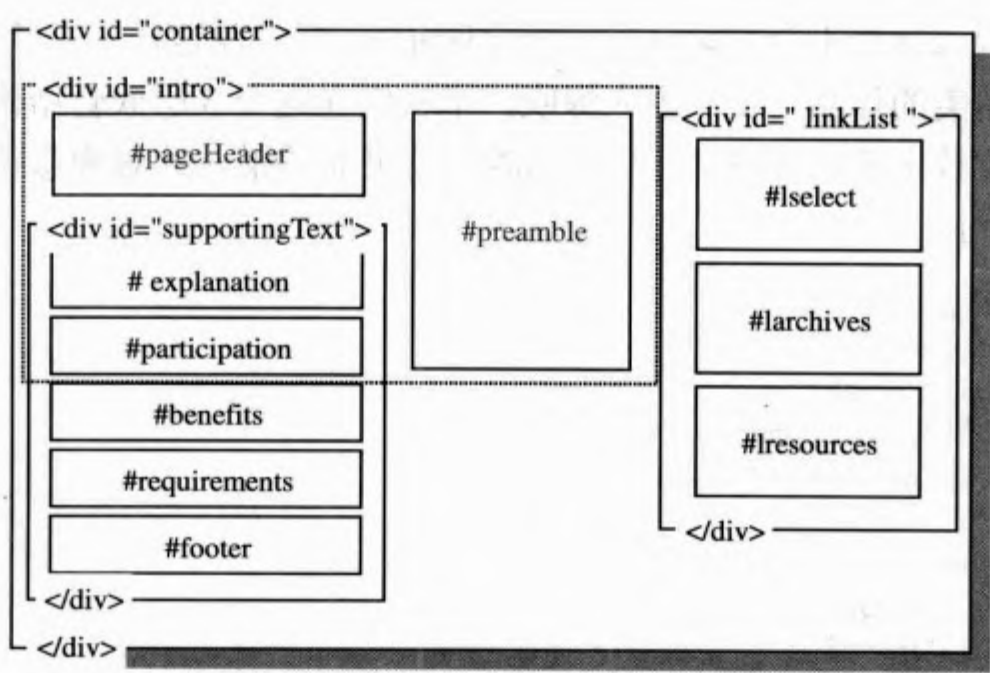


图 10.69 山鹰之美网页结构布局示意图

这个结构所设计的网页效果如图 10.70 所示。



图 10.70 山鹰之美网页布局效果图

禅意花园的结构永远都不会变化（如下程序所示），但是就这样一个简单的 3 层嵌套结构却能够设计出无数精美的作品，实在令人惊叹。本实例在这个 3 层嵌套结构的基础上，通过应用流动、浮动和定位技术来设计图 10.70 所示的精美页面效果，针对禅意花园的整个结构的布局思路如图 10.71 所示。

```
<div id="container">
  <div id="intro">
    <div id="pageHeader"> </div>
    <div id="quickSummary"> </div>
    <div id="preamble"> </div>
  </div>
  <div id="supportingText">
    <div id="explanation"> </div>
    <div id="participation"> </div>
    <div id="benefits"> </div>
    <div id="requirements"> </div>
    <div id="footer"></div>
  </div>
  <div id="linkList">
    <div id="linkList2">
      <div id="lselect"> </div>
      <div id="larchives"> </div>
      <div id="lresources"> </div>
    </div>
  </div>
</div>
```

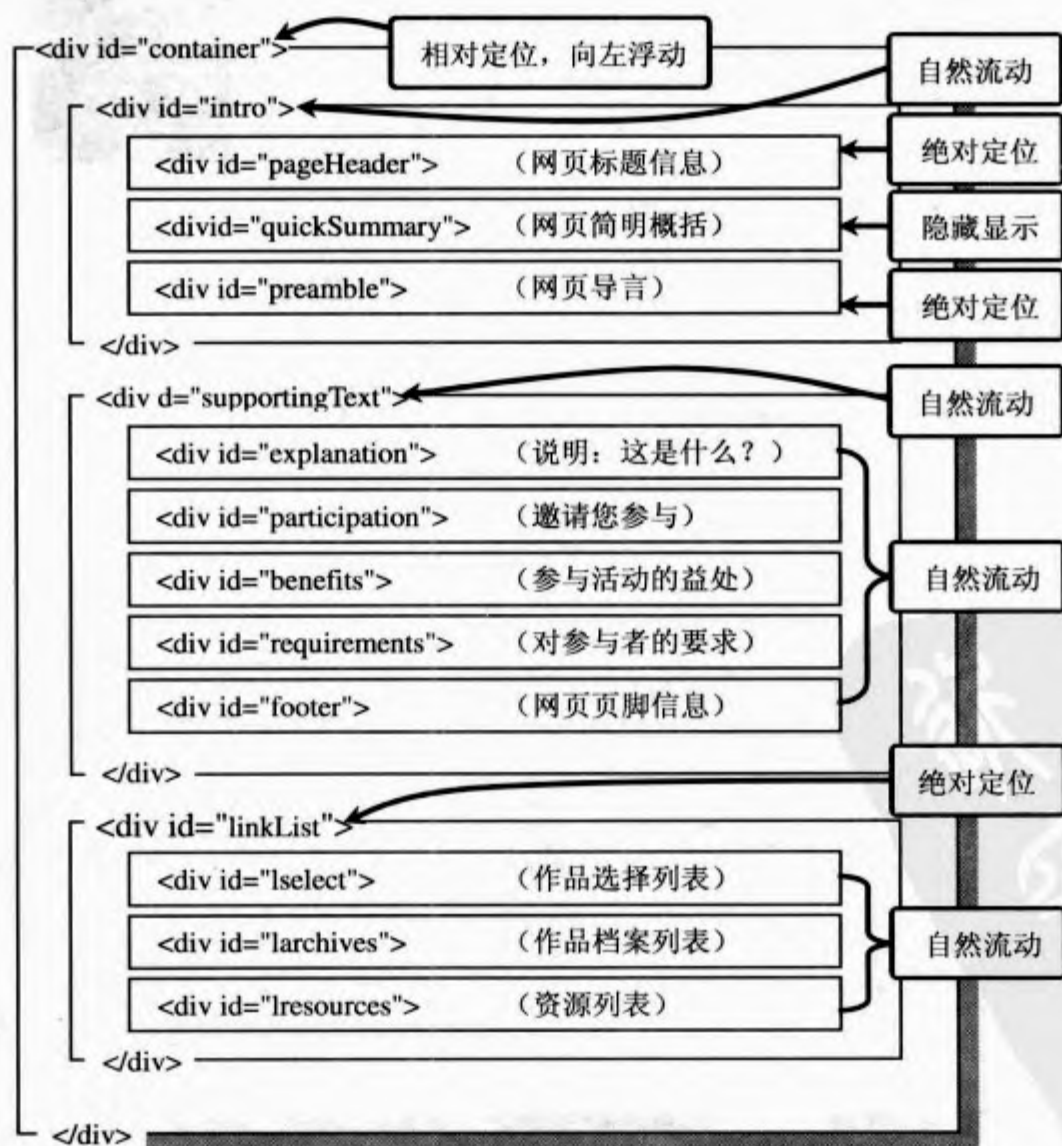


图 10.71 山鹰之美网页布局的结构设计思路

整个页面分3列液态布局,通过绝对定位的方法来设计中列和右侧栏目的定位,以及左列头部版块的设计。清楚了设计思路,具体操作起来就会感觉轻松多了,下面就来讲解页面的具体操作过程和设计细节。

第1步,设置页面基本属性和统一网页元素基本样式。工作的第一步是设置网页的基本属性,如页边距、网页背景和字体属性等,关于这点在前面实例中也反复强调了。统一页面常用元素的基本显示样式也是很重要的一步,如清除超链接下划线、清除列表项目符号、设置段落间距、定义标题样式等。

```
* { /* 定义所有元素公共样式 */
    margin: 0; /* 清除所有元素的默认外边距 */
    padding: 0; /* 清除所有元素的默认内边距 */
}
a { text-decoration: none; } /* 清除超链接下划线 */
body { /* 页面基本属性 */
    background: #FFF url(bg.gif) repeat-x; /* 网页背景图像 */
    color: #444; /* 网页字体颜色 */
    font: normal 62.5% "Lucida Sans Unicode", Verdana, sans-serif; /* 网页字体属性 */
    padding-top: 40px; /* 顶部内边距 */
}
li { list-style: none; } /* 清除列表项的项目符号 */
h1 { font-size: 1.4em; } /* 一级标题的字体大小 */
h1, h2, h3, h4 { /* 定义标题基本样式 */
    font: normal 1.2em "Trebuchet MS", sans-serif; /* 标题字体的属性 */
    color: #F06; /* 标题字体的颜色 */
}
p { /* 定义段落首行缩进和行高 */
    text-indent: 2em; /* 段落文本首行缩进 */
    line-height: 1.8em; /* 段落文本的行高 */
}
```

第2步,在使用CSS进行布局时,当定义好网页基本属性、基本元素的默认样式之后,还可以根据需要定义一些常用样式类。例如,定义颜色样式类、功能类、字体样式类等。这里定义一个清除浮动类:

```
.clearer { /* 清除浮动类 */
    clear: both; /* 清除两侧浮动 */
}
```

这样在网页中如果需要清除某个元素的浮动,则直接引用该类即可,不再重复定义相同的样式。如此一来不但减轻了输入代码负担,也优化了代码,方便CSS样式的管理。

第3步,定义网页包含框基本样式。由于要在网页中定位元素,因此定义<div id="container">网页包含框为相对定位,把它转换为包含块。

```
#container { /* 定义网页包含块 */
    position: relative; /* 相对定位 */
    float: left; /* 向左浮动 */
}
```

为网页包含框声明向左浮动规则,目的是想让它能够自动包含所有元素,否则会收缩为一团,影响到其他元素的定位。

第4步,设计<div id="supportingText">模块的布局。该模块将以自然流动的方式呈现在页面左侧,因此不需要刻意去进行设计。考虑到页首需要显示网页标题,因此通过外边距的方式

预留一定的区域为网页标题显示。该模块的宽度以百分比为单位，以适应液态版式的需要（如图 10.72 所示）。

```
#supportingText { /* 左列布局 */
    margin: 180px 42% 20px 3%; /* 调整外边距 */
}
```

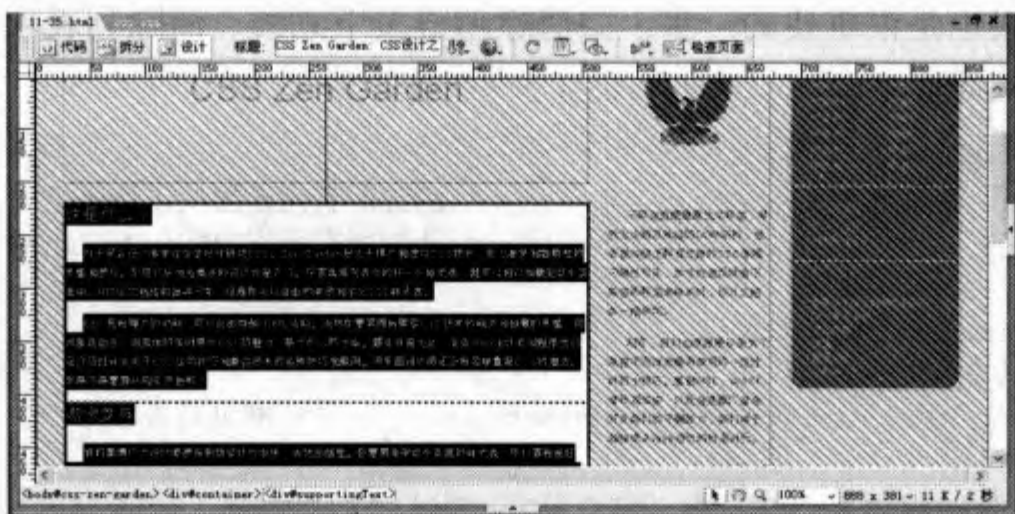


图 10.72 左栏布局

第 5 步，定位<div id="pageHeader">元素到左侧顶部位置。如果让网页标题显示在左侧顶部（如图 10.72 所示），就必须使用绝对定位来实现。

首先，使用绝对定位的方法固定<div id="pageHeader">元素到页面左侧顶部，定义宽度和左侧距离为百分比，这样能够自适应页面宽度的变化，如图 10.73 所示。

```
#pageHeader { /* 定位标题栏 */
    background: url(stripes.gif) no-repeat; /* 定义网页标题的背景图像 */
    border-bottom: 1px solid #eee; /* 底部边框线 */
    height: 160px; /* 网页栏目的高度 */
    margin-bottom: 24px; /* 底部外边距 */
    width: 55%; /* 百分比宽度 */
    position: absolute; /* 绝对定位 */
    left: 3%; /* 左侧距离 */
    top: 0; /* 顶部距离 */
}
```



图 10.73 定位网页标题栏

然后在绝对定位的网页标题栏中定位二级标题到右上角显示（如图 10.73 所示）。其他具体的修饰性细节就不再详细讲解，读者可以参阅光盘实例了解。

```
#pageHeader h2 { /* 定位网页二级标题 */
    border-top: 1px solid #eee; /* 顶部边框 */
}
```



```
position: absolute; /* 绝对定位 */
width: 90%; /* 百分比宽度 */
top: 0; /* 顶部距离 */
right: 0; /* 右侧距离 */
}
```

第 6 步, 隐藏<div id="quickSummary">元素内容, 绝对定位<div id="preamble">元素到网页中间列, 绝对定位<div id="linkList">元素到网页右侧, 如图 10.74 所示。在定义各列宽度和左右外边距时, 为了能够适应页面自适应宽度, 应该以百分比来设置取值单位。

```
#preamble, #linkList { /* 定位网页中列和右侧列栏目 */
    margin-bottom: 20px; /* 底部外边距 */
    position: absolute; /* 绝对定位 */
    top: 0; /* 顶部距离 */
}
#quickSummary { /* 隐藏多余栏目 */
    display: none; /* 隐藏显示 */
}
#preamble { /* 定义网页中列版式 */
    right: 23%; /* 右侧距离 */
    width: 17%; /* 百分比宽度 */
}
#preamble h3 span { /* 隐藏中列内容块的标题内容 */
    display: none; /* 隐藏显示 */
}
#linkList { /* 定义网页右列版式 */
    background: #222 url(round_lt.gif) no-repeat left top; /* 定义圆角图像 */
    right: 3%; /* 右侧距离 */
    width: 18%; /* 百分比宽度 */
}
```

第 7 步, 完成基本框架的搭建, 你就可以细化网页内每个栏目的每个细节。例如, 在隐藏中间列板块的标题之后, 可以利用 h3 元素并结合背景图像来定义网页的 Logo 图标。右列的圆角区域可以使用圆角来定义, 在前面章节中我们曾经介绍了多种设置圆角的方法, 这里就不再重复。

请注意, 如果要利用多个嵌套元素来定义背景图像式的圆角, 应保证外层元素不要定义边距, 定义背景图像的元素不要定义外边距, 否则背景图像就不能够对齐到模块的 4 个顶角位置。



图 10.74 定义网页中列和右列版式

第 11 章

CSS 框架设计及其案例解析

CSS 语言与 HTML 语言一样，都是一种弱类型的高级语言，源代码被下载到客户端后通过本地浏览器解析，并把 CSS 样式的效果呈现出来。这与同是弱类型语言的 JavaScript 无法相比，因为 JavaScript 有自己的逻辑体系和计算能力。

当我们开发 CSS 样式的时候，代码排错和维护一直是件令人烦恼的事情，根本原因就在于 CSS 语言本身的不严谨性，而浏览器对于 CSS 又有排错能力，只要发现 CSS 存在语法错误，就会忽略不计，而不影响浏览器的正常工作，这与 JavaScript 截然不同。这种包容性无形中又增加了设计师的维护难度。所以我们就更应该注重 CSS 定义的严谨性，避免错误。同时，一个网站或项目中会存在大量的 CSS 重复代码，如何发挥 CSS 自身的特性，避免这种代码的重复，也是设计师应该思考的问题。于是在这种环境下，CSS 框架设计逐渐被人们重视起来。本章将讲解目前正被各大商业网站重视的 CSS 框架的开发原则以及设计思路，并结合具体的案例帮助读者快速掌握 CSS 框架的开发以及模块化布局的一般方法。

11.1 CSS 框架设计

CSS 技术的普及和应用是一个渐进过程，从早期的字体样式定义，到后来的文本版式，再到网页布局，如今 CSS 框架也逐渐被人重视起来。目前前端设计师逐渐认识到：从具体的表现中提取出抽象的模块来重复使用，是减少带宽、方便团队开发最重要的手段。于是 CSS 框架就在这种背景下诞生了。

其实框架技术在很早以前就被程序开发人员人提起并应用到实践中，只不过当时 Web 前端开发还比较弱小，没有被人重视罢了。自从 Ajax 兴起之后，互联网上突然冒出很多以 Ajax 技术为核心的前端设计框架，如有 Prototype、jQuery、Dojo、YUI 和 Ext 等。受此启发，部分 Web 设计师开始思考 CSS 框架问题。

11.1.1 什么是 CSS 框架

有些时候，你可能有过这种冲动，即把日常常用的 CSS 代码进行提炼、加工，汇集为一个 CSS 代码库，以期望日后不再重复相同的工作，实际上这正是 CSS 框架的雏形。

如果开发一个大型的网站，或者重复开发同类的网站，你会发现 CSS 框架带来的方便和益处。此外，团队中的设计师们也能够 CSS 框架中进行高效合作。

例如，对于博客网站来说，也许它们都保持着很多相似性，如果把这些共同样式提炼出来，形成一个 CSS 框架，那么在设计同类网站时，就不再浪费时间、耗尽脑筋去构思这些重复的工

作了。

Ajax 技术框架为前端 Web 交互带来极大的便利, 仿佛所有人都在使用 Ajax 框架来开发自己的网站。但究竟什么是框架? 设计师是否可以从受益呢?

CSS 框架 (CSS Framework) 也可以称为 CSS 技术库 (CSS library), 它包含一套应用工具、函数库、约定俗成的规则, 以及从常用任务中抽象出可以重复使用的通用模块。开发框架的目的是为了减轻设计师重复开发的工作量, 提高 Web 开发的通用性和兼容性, 这样设计师就可以把精力集中到任务或项目所特有的内容设计方面, 而不再重复开发基本功能问题。

CSS 框架与 Ajax 框架或者其他编程中的基层框架存在很大的不同。

(1) 首先, CSS 框架是一种松散式代码库, 库内代码之间没有必然的联系和逻辑关系。它犹如积木, 可以随时增加、删节。因此, 也就没有编程框架中所谓的构造器、发布和打包等相关技术问题。

(2) 其次, CSS 框架的通用性比较差。在这个项目中开发的框架, 没法把它应用到其他项目中, 或者部分无法使用。这是因为 CSS 代码都是建立在具体对应的 HTML 结构基础之上的。而不同的项目由于设计的 HTML 结构存在很大的不同, 因此就不能够简单地把一个项目的 CSS 框架直接应用到另一个项目中。

最后强调一下, 一个 CSS 框架只适合一个项目或者一个团队内部使用。可以学习它们搭建 CSS 框架的方法, 但是千万别直接拿来应用。

11.1.2 分析 CSS 框架的优缺点

也许在继续阅读之前, 你已模糊认识到 CSS 框架的优缺点, 下面做一个简单的小结, 指导你在什么时候该用, 什么时候不该用。

CSS 框架的优点当然是很多了, 也许你能够总结出很多理由, 但是下面这些优点是显而易见的。

(1) 规范 CSS 开发。通过框架的提炼和加工, 你会感觉自己在编写 CSS 代码时规范了许多, 考虑问题都要谨慎许多。

(2) 便于合作。搞开发免不了几个人一块干相同的工作, 你设计这个模块, 他设计那个模块, 如果没有框架的支持, 你会发现彼此所设计的代码存在很多重复, 久而久之, 废弃代码越来越多, 互相之间也难以沟通和阅读。如果建立合适的 CSS 框架, 相互之间就可以减少很多不必要的错误。

(3) 提高开发效率。这是显而易见的了。如果把整个项目的基本、重复的 CSS 代码设计好, 那么你会发现剩下的工作就好做了, 容易得多了, 做一个小框架体验一下即可。

(4) 提高兼容性。在网页设计中浏览器的兼容性是一个很重要的问题, 大部分项目都要求能够基本兼容 IE 6、IE 7 和 FF。如果在 CSS 框架中把这些问题都解决好, 那么后期的开发中就不再为此而焦头烂额, 可以为后面的兼容问题节省大量时间。

优点说了这么多, 并不等于 CSS 框架没有缺点, 对于初学者来说可能 CSS 框架对于自己的不便甚至大于它所蕴藏的优势。为什么这么说呢, 原因如下。

(1) 需要花费很多时间来理解整个 CSS 框架的思路、目的、任务、对象、作用、条件、效果、编码规范等。这对于一个初次接触 CSS 框架的人来说, 难度是可想而知的。

(2) 当然这个既定的 CSS 框架会很容易把你的思路给束缚住。它犹如一个定位思维, 要求你必须这样做, 而不能够那样做。任何一个成熟的框架都可能存在不科学、不合理的地方。这

样会压抑你的创造性思维和创新性设计。

(3) 框架代码一般都比较臃肿。由于在设计框架时，你必须要考虑周全，而为了周全所要付出的代价就是必须为某个问题多设计很多代码，以防止各种意想不到的情况发生。一般情况下，在某个具体项目中可能仅需要框架所提供的 20% 代码或功能，这样就等于在这个项目中产生了 80% 的 CSS 冗余代码。

(4) 框架的一个最大特征就是尽可能使 CSS 代码具有抽象性，以扩大它的适应能力。而这又与 HTML 语义化产生了尖锐的矛盾，HTML 语义化希望元素的语义尽可能唯一。如何处理好这个矛盾，也是很多设计师最为头疼的问题。

11.1.3 如何设计 CSS 框架

CSS 框架设计不是凭空想象得来的，很多设计师或者初学者看到 CSS 框架很好，心血来潮就想先搭建一个 CSS 框架，并幻想着把 CSS 框架搭建好后就可以高枕无忧了。

实际这是错误的。一方面，搭建一个 CSS 框架非一日之功。另一方面，花费了半天工夫把框架搭建好了，当真正应用时突然发现自己的项目中并不需要这些框架代码，或者代码利用率很低，这既浪费时间，又耽误工夫。

真正的 CSS 框架应该是在实际开发中不断摸索、积累、完善的。CSS 框架不是一蹴而就的，也不是永远的真理到处适用。一般来说，CSS 框架应该具有针对性，它适合于某个具体的网站或者项目，不具备在多个网站之间迁移的能力。当然你可以在已有的 CSS 框架基础上借鉴它的设计思路方法。

对于同一个网站来说，如果不断进行升级或改版，特别适合在升级或改版之前，在原来版本的基础先总结出一套基本的 CSS 框架，然后在此基础上升级或改版，可能会加速网站升级或改版进度。相反，如果在建站之初就希望先设计一个完善的 CSS 框架，这是不妥的，你可以在设计中不断去提炼和总结，渐进式完善自己的 CSS 框架。当然如果你是一位高手，有着丰富的实战经验和技能，不排除建站之初就高屋建瓴，从宏观上把整体的 CSS 框架搭建好，然后再去丰富每页的细节。

CSS 框架能够提供一个干净、严谨的基础架构。当然在准备设计 CSS 框架时，下面几个问题值得你去思考：

(1) 尽可能使 CSS 框架具有抽象性，不要被具体的页面所羁绊。

(2) 要希望框架能够具有最大的适应性，应该增强 CSS 框架的可扩展性，给 CSS 框架设计统一的对外接口，以便随时可以在具体项目或页面中增加样式。

(3) 要考虑 CSS 框架的可维护性和可兼容性。

下面我们就围绕这些核心问题展开讲解。

11.2 CSS 框架的抽象性及其设计

抽象性是编程中一个核心的设计思想，也就是我们常说的类。当所有类汇集在一起，通过一定的逻辑组织在一起，就形成了类库。一般编程语言都有自己的类库，如 Java、.Net 类库等。CSS 虽然不是高级语言，但是它也提供对类的支持，这主要表现在类样式上。

11.2.1 认识 CSS 的类样式

在第 1 章我们曾经详细讲解过 CSS 的基本选择符及其应用。类样式实际上就是类选择符设计的样式，一般以点前缀 (.) 来表示。例如：

```
<style type="text/css">
.red {
    color:red;
}
</style>
```

在上面代码中定义了一个红色字体类，然后就可以在 HTML 文档中随处应用该类样式。例如：

```
<p id="p1" class="red"></p>
<h2 id="header_h2" class="red"></h2>
<div id="box" class="red"></div>
```

在上面的代码中分别为段落 p1、网页页眉的二级标题以及 id 为 box 的 div 包含框应用了红色字体类，这样该对象包含的所有文本都将显示为红色。

这种简单的 CSS 类功能加强了样式表在网页中的广泛应用，为 CSS 框架的抽象性提供了一种选择方法。

11.2.2 CSS 类样式的实施策略

是不是定义类样式就很简单呢？不是，类样式的定义方法很简单，但是如何设计好一套比较实用的类样式库就很不容易了。下面几点原则供你在开发类样式时参考。

第一，CSS 的类应该体现最小化效果设计原则。这样就能够更灵活地应用类样式。例如，定义一个 12 像素灰色字体的类：

```
.class1 {
    color:gray;
    font-size:12px;
}
```

也许直接定义上面的类样式仅能够在页面中某一处被引用。但是如果把它们拆分开来，就会具有更大的灵活性和实用价值：

```
.gray {
    color:gray;
}
.font12px {
    font-size:12px;
}
```

因为，在页面中某处可能仅需要灰色字体，或者仅需要 12 像素的字体大小，此时可以单独引用某一个类样式。但是如果希望某个对象同时显示为灰色、12 像素大小的字体，则可以通过类样式并列引用的方式来实现。

```
<p id="p1" class="gray font12px"></p>
```

通过这种方法，可以把多个样式类并列应用到一个对象身上，它们并列显示位置顺序并会对样式产生影响。但是类样式在 CSS 样式表中的位置会影响到它们的应用效果。例如，在下面这个小示例中，虽然在 `<p id="p1" class="red blue">` 中 blue 类样式名字位于后面，紧靠包含的文

本，但是最终字体显示为红色，因为在样式表中红色字体类放在后面。

```
<style type="text/css">
  .blue {
    color:blue;
  }
  .red {
    color:red;
  }
</style>
<p id="p1" class="red blue">类样式的位置关系</p>
```

当然这个问题也不是绝对的，所谓最小化效果单元就是一个类样式中，如果几个声明被分开之后，没有被重复利用的价值，就不应该再分开进行定义。例如，看看下面这个隐藏类：

```
.hide {
  display:none;
  line-height:0;
  font-size:0;
}
```

这个隐藏类其中定义了 3 个属性，这些属性都是针对隐藏元素来设计的，此时就不能够把它们拆分为 3 个小类：

```
.none { display:none; }
.l_height0 { line-height:0; }
.font0px { font-size:0; }
```

一方面这 3 个属性都是针对同一个类样式的效果进行定义的，拆分之后没有意义；另一方面这些被拆分的小类实用价值不高，没有必要为此定义 3 个小类。

第二，CSS 类应当体现通用性。所谓通用性，就应该具备广泛的应用价值。除了上面讲解的定义类时应该尽可能定义小的样式单元，同时还应该保证所定义类具有广泛代表性，最起码应知道在自己的项目中经常会用到哪些类、不需要哪些类。

第三，当定义 CSS 类库时，建议一定要遵循一定的规律进行定义。一方面在命名类样式时要有规律，另一方面建议把所定义类库进行归类 and 建立索引，避免忘记所定义类样式，而反复定义相同的样式所产生的代码冗余。这样在使用和参阅时也可以快速地进行浏览。

11.2.3 通用 CSS 类样式设计

在一般网站开发中，可能会遇到下面这些通用类样式，你也可以参考这样的思路来设计类样式，并进行扩展。

(1) 字体颜色类：关注常用字体色。例如，以颜色名来定义类。

```
.white { color:white; }
.black { color:black; }
.gray { color:gray; }
.blue { color:blue; }
.red { color:red; }
.green { color:green; }
```

以十六进制颜色值来定义类（首位数值为非数字）。

```
.a00 { color: #a00; }
.f66357 { color: #f66357; }
```


如果首位值为数字,就不能够使用数字来开头定义类名,此时可以使用复合词的方式来解决(具体方式可以根据团队习惯确定)。

```
.color_333 { color:#333; }  
.color_999 { color:#999; }  
.color_234567 { color:#234567 }
```

(2) 字体大小类:关注常用字体大小。字体大小类一般比较固定,想一想一个网站基本上就使用那几种字体大小,例如,正文 12 像素字体、标题 14 像素字体等。

```
.font12px { font-size: 12px; }  
.font14px { font-size: 12px; }  
.font9pt {font-size: 9pt; }
```

字体大小类的命名规则可以根据自己的使用习惯,上面是通用设计习惯,还有的设计师使用简写方式,例如:

```
.font_12 { font-size: 12px; }  
.font_14 { font-size: 12px; }
```

或

```
.font12 { font-size: 12px; }  
.font14 { font-size: 12px; }
```

使用这种方式的前提是应该保证整个 CSS 框架的字体大小单位是统一的,否则就容易出现混乱现象。从语义角度来看,上面的简写显然没有直接拼写更明了。不过如果一个团队习惯了一种简写方式,从而容易提高代码输入效率。这就要具体而论了。

还可以使用绝对或相对大小类来定义,例如:

```
.small { font-size:small; }  
.large { font-size:large; }  
.medium { font-size:medium; }  
.smaller { font-size:smaller; }  
.xx-large { font-size:xx-large; }
```

(3) 字体样式类:关注常用字体样式的变化。字体样式也是很有限的,一般框架中应该把常用字体样式都进行类化,例如:

```
.bold { font-weight:bold; }  
.italic { font-style:italic; }  
.underline { text-decoration:underline; }
```

对于英文字体来说,可以定义的种类可能会多些,但是如果把每一种字体样式都定义为类就没有必要了,因为很多字体样式是不常用的,偶尔使用是不值得定义为类的。

(4) 段落样式类:关注段落版式的类型。段落版式可谓千变万化,不能够一一列举,比较常用的有缩进、字符间距等。

```
.indent2em { text-indent:2em; }  
.letter-spacing1em { letter-spacing:1em; }
```

有些样式如行高,可以在 body 中定义,且应用比较单一,不适合为此设计一个类。当然如果项目中需要显示大量的文本,还是应该定义几个行高样式类的。

也许你会觉着这样的类名很长,输入麻烦,因此可以以简写的方式来设计,当然在简写设计时一定要遵循相应的规律,否则后期维护时会非常麻烦,因为时间长了,连自己都会忘记了

所定义的类的作用。这样很容易产生大量的冗余代码。

(5) 边框类样式: 关注模块的边框样式。这类样式一般多针对具体的项目、具体的页面而言, 把模块间拥有相同的边框进行归类。不过这种做法的抽象性不是很强, 所提炼的价值不大, 适合直接在模块的 ID 中直接进行定义。而对于一些特殊的边框样式倒很值得进行类化, 例如, 虚线框、单线框, 以及个人框架中的各种典型线框等。

```
.border_dashed { border:dashed 1px #666; }  
.border_dotted { border:dotted 1px #666; }  
.border1px { border:solid 1px #666; }
```

由于边框涉及 3 个属性、4 条边, 所以要想提炼出 CSS 框架所需要的各种边框类样式是完全不可能的, 也是画蛇添足。

(6) 边距类样式: 关注网页模块的内、外边距的类样式。对于这类样式进行类化, 必要性不是很大。可以略过。

(7) 背景色和背景图像类: 关注网页模块的背景色。这是一个很实用的类别, CSS 框架中免不了要设计一些模块的背景色。如果在设计之前在框架中把常用的背景色进行类化, 是很值得的。例如:

```
.bg_fef{ background:#fef; }  
.bg_fffeee{ background:#fffeee; }  
.bg_def2de{ background:#def2de; }
```

当是, 如果以这种方式来命名背景色类, 对于后期引用是非常不利的, 因为设计师需要去记忆这些烦琐的类名, 甚至还容易出错。这时可以使用颜色名来表示:

```
.bg_aqua{ background:#00ffff; }  
.bg_silver{ background: #c0c0c0; }
```

如果感觉颜色名也不好记忆, 不妨使用基本颜色加编号或者其他标记来表示:

```
.bg_red_1{ background:#FFCCFF; }  
.bg_red_2{ background:#FF99FF; }  
.bg_red_3{ background:#FF99CC; }
```

上面 3 个类样式表示随着后缀数字变大, 红色不断加重。当然这仅是一种个人的用法。最后决定者还是根据团队使用习惯而定, 没有统一的规则, 也没有硬性的要求。

(8) 布局类样式: 关注布局中常用类模块。网页布局中一般都是针对具体的模块进行定义, 能够提炼的类样式没有很大的实用价值。因为当定义一种布局类型时, 还可以定义相关的布局属性。但是下面这些布局类是比较常用的, 特别是在布局中随时改变布局模式或者调整个别属性时, 使用比较方便。

```
.float_left { float: left; }  
.float_right { float: right; }  
.block { display:block; }  
.inline { display:inline; }  
.hide { display:none; }  
.clear { clear:both; }  
.relative { position:relative; }  
.absolute { position:absolute; }  
  
.width100 { width:100%; }  
.height100 { height:100%; }  
.center { text-align:center; }
```



```
.left { text-align:left;}
.right { text-align:right;}
.middle { vertical-align:middle; }
```

当然这仅是一个例子，应用中还需要你自己去提炼和扩展。

(9) 功能样式类：关注 CSS 框架中公共功能的样式。这类样式没有可以直接预期的对象，必须结合具体的项目而定。例如，定义圆角区域类样式：

```
.lt { background:url(images/lt.gif) left top no no-repeat; }
.rt { background:url(images/rt.gif) right top no no-repeat; }
.lb { background:url(images/lb.gif) left bottom no no-repeat; }
.rb { background:url(images/rb.gif) right bottom no no-repeat; }
```

类似的功能类还有很多，需要对项目进行提炼了。

(10) 基本模块类样式：这类样式主要是针对 CSS 框架中通用基本模块进行定义的类。例如，每个栏目中的标题栏、提示框等。对于这些常用的基本模块不妨通过类的形式进行定制，然后在项目中可以反复引用。例如，图 11.1 所示的正是 Ext JS 类库中的一个 CSS 类模块。这个类似于对话框的标题栏样式正是通过类样式的形式进行定制，这样能够保证在一个页面中多次重复引用。

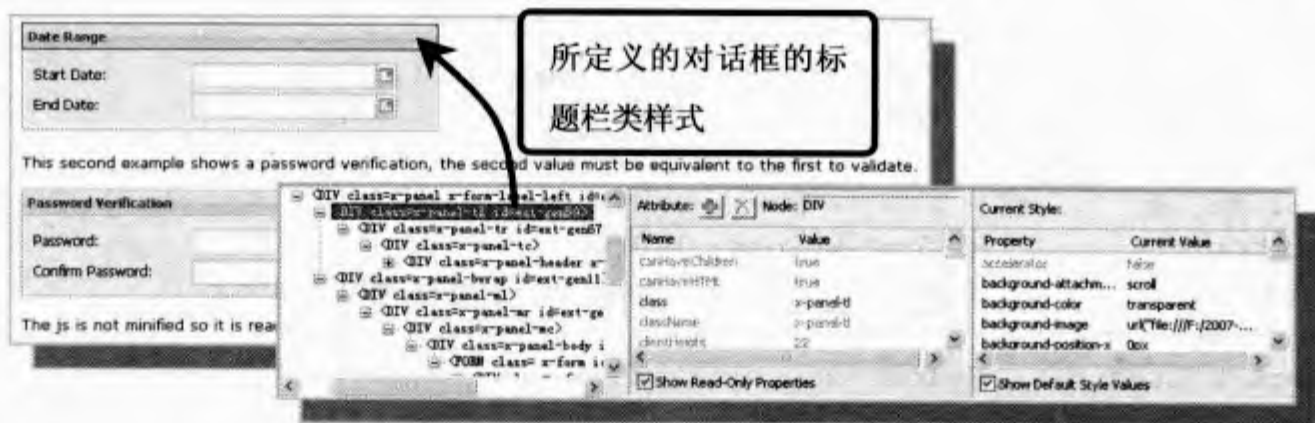


图 11.1 Ext JS 中的类模块

(11) 特定样式类：关注 CSS 框架中一些特定样式进行类化。例如，手形指针类型、快捷键下划线等。

```
.hand { cursor:pointer; cursor:hand; }
.accesskey {text-decoration: underline;}
```

以上针对通用样式类进行简单的小结，当然上面所列也仅仅是一个示例，最后还需要进一步去丰富自己的类库。

11.2.4 CSS 框架中元素默认样式设计

每种浏览器都会定义一套默认的样式，CSS 在设计时也专门为 HTML 定制一套默认的样式（如下程序所示），请参阅 <http://www.w3.org/TR/CSS21/sample.html>。

```
html, address,
blockquote,
body, dd, div,
dl, dt, fieldset, form,
frame, frameset,
h1, h2, h3, h4,
h5, h6, noframes,
ol, p, ul, center,
dir, hr, menu, pre { display: block }
```

```

li      { display: list-item }
head    { display: none }
table   { display: table }
tr      { display: table-row }
thead   { display: table-header-group }
tbody   { display: table-row-group }
tfoot   { display: table-footer-group }
col     { display: table-column }
colgroup { display: table-column-group }
td, th  { display: table-cell }
caption { display: table-caption }
th      { font-weight: bolder; text-align: center }
caption { text-align: center }
body    { margin: 8px }
h1      { font-size: 2em; margin: .67em 0 }
h2      { font-size: 1.5em; margin: .75em 0 }
h3      { font-size: 1.17em; margin: .83em 0 }
h4, p,
blockquote, ul,
fieldset, form,
ol, dl, dir,
menu    { margin: 1.12em 0 }
h5      { font-size: .83em; margin: 1.5em 0 }
h6      { font-size: .75em; margin: 1.67em 0 }
h1, h2, h3, h4,
h5, h6, b,
strong  { font-weight: bolder }
blockquote { margin-left: 40px; margin-right: 40px }
i, cite, em,
var, address { font-style: italic }
pre, tt, code,
kbd, samp  { font-family: monospace }
pre        { white-space: pre }
button, textarea,
input, select { display: inline-block }
big        { font-size: 1.17em }
small, sub, sup { font-size: .83em }
sub        { vertical-align: sub }
sup        { vertical-align: super }
table      { border-spacing: 2px; }
thead, tbody,
tfoot      { vertical-align: middle }
td, th    { vertical-align: inherit }
s, strike, del { text-decoration: line-through }
hr         { border: 1px inset }
ol, ul, dir,
menu, dd   { margin-left: 40px }
ol         { list-style-type: decimal }
ol ul, ul ol,
ul ul, ol ol { margin-top: 0; margin-bottom: 0 }
u, ins     { text-decoration: underline }
br:before { content: "\A" }
:before, :after { white-space: pre-line }
center      { text-align: center }
:link, :visited { text-decoration: underline }
:focus      { outline: thin dotted invert }

```



```

/* Begin bidirectionality settings (do not change) */
BDO[DIR="ltr"]    { direction: ltr; unicode-bidi: bidi-override }
BDO[DIR="rtl"]    { direction: rtl; unicode-bidi: bidi-override }

*[DIR="ltr"]      { direction: ltr; unicode-bidi: embed }
*[DIR="rtl"]      { direction: rtl; unicode-bidi: embed }

@media print {
  h1                { page-break-before: always }
  h1, h2, h3,
  h4, h5, h6        { page-break-after: avoid }
  ul, ol, dl        { page-break-before: avoid }
}

```

除了浏览器（或称代理器）提供的默认样式、CSS 定制的一套默认样式外，设计师可以定义一套网页显示样式，同时浏览者也可以根据阅读设置个性化的用户样式（目前浏览器支持不是很好）。对于上述 4 套样式中，它们的优先级如下。

用户（浏览者指定的）样式优先于创作者（设计师设计的）样式，创作者样式高于浏览器（或称为代理）样式，而浏览器样式又优于 CSS 默认的样式。用示意图表示如图 11.2 所示。

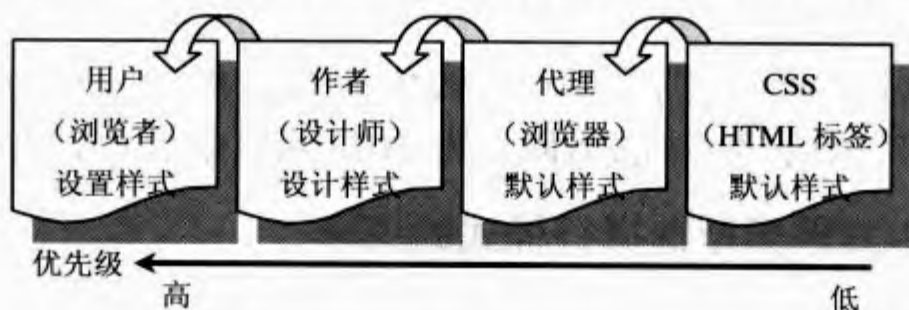


图 11.2 不同类型的样式优先级

对于一个框架来说，应该根据需要对于这些元素的默认 CSS 样式进行重定义。例如，重新定义页面基本属性（如字体、字号、字体颜色、行高、超链接样式、标题样式等）。根据需要清除一些元素的默认样式（如页边距、项目列表缩进、项目列表符号、段落间距、超链接下划线等）。根据需要增加一些元素的默认样式（如段落首行缩进，图片不显示边框等）。

例如，下面这些 CSS 框架默认样式是网页设计中经常用到的，希望读者能够熟悉并记住它们。

```

body{/* 网页默认属性 */
  position: relative;
  font-size:62.5%;
  font-family: "宋体";
}
/* 清除常用元素的默认边距 */
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,form,fieldset,input,p,blockquote,th,td,ins{
  margin: 0px;
  padding: 0px;
}
h1,h2,h3,h4,h5,h6{
  font-size:100%;
}
ol,ul{/* 清除列表样式 */
  list-style-type: none;
}

```

/* 把 body 定义为包含块，默认为 html */
 /* 网页字体大小 */
 /* 网页字体 */
 /* 清除外边距 */
 /* 清除内边距 */
 /* 重定义标题字体大小样式 */
 /* 字体大小 */
 /* 不显示项目列表符号 */

```

address,caption,cite,code,dfn,em,th,var{          /* 重定义文本标签样式 */
    font-style:normal;                             /* 恢复正常字体样式 */
    font-weight:normal;                             /* 恢复正常字体粗细 */
}
table{/* 增加表格样式 */
    border-collapse:collapse;                       /* 合并单元格边框 */
}
fieldset,img{/* 增加框架和图片样式 */
    border:0;                                       /* 隐藏边框 */
}
caption,th{/* 重定义标题样式 */
    text-align:left;                               /* 文本左对齐 */
}
a{/* 清除超链接样式 */
    text-decoration: none;                         /* 清除下划线 */
}

```

当然对于一个特定的 CSS 框架来说，上面的元素默认样式仅作为一个参考，你的框架是否必须这样，就看实际需要了。

另外，不同的 CSS 框架可能会根据需要定义一批专用类。这些类是根据特定的项目所需而提炼出来的重复样式码。

一般专用类不具备通用性，它只能够适用特定的项目或者网站。例如，栏目的风格样式类、网页特定布局样式类等。由于这些都没有固定的规律，所以就不再详细讲解。

11.2.5 CSS 框架中其他类型选择符样式的处理

除了类样式和元素默认样式外，其他类型的样式该如何处理？下面我们就来做简单分析。

对于 ID 样式来说，由于它是针对某个具体的对象而定义的样式，一般不建议在 CSS 框架中使用。越抽象的 CSS 框架，它的实用价值就越高。当然你可以针对具体的项目定义一批 ID 样式，以实现在不同页面中的通用性。例如，定义页眉、页脚样式，由于一般页面都会使用相同的页眉和页脚，把它们的样式统一到 CSS 框架中也未尝不可。

对于通配选择符（*匹配所有元素）的样式，可以在 CSS 框架中使用。例如，针对下面的默认样式：

```

/* 清除常用元素的默认边距 */
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,form,fieldset,input,p,blockquote,th,td,ins{
    margin: 0px;                                     /* 清除外边距 */
    padding: 0px;                                    /* 清除内边距 */
}

```

你可以使用通配选择符来进行简写：

```

/* 清除常用元素的默认边距 */
* {
    margin: 0px;                                     /* 清除外边距 */
    padding: 0px;                                    /* 清除内边距 */
}

```

对于包含选择符的样式，可以在特定结构中适当使用。例如，定义段落中的 red 类显示为浅红色，而对于标题中的 red 类显示为深红色，则可以借助包含选择符进行如下定义：

```

<style type="text/css">
.red { color:#ff0000; }

```



```
h1 .red, h2 .red, h3 .red, h4 .red, h5 .red, h6 .red { color:#990000; }
p .red { color:#990033; }
</style>
<div class="red">普通文本</div>
<h2><span class="red">标题</span></h2>
<p id="p1"><span class="red">段落</span>文本</p>
```

此外，子对象选择符样式、相邻选择符样式、属性选择符样式等都是非常有价值的样式，特别是属性选择符样式在定制 CSS 框架时显得异常重要，对于更准确地控制框架中每个细节具有重要作用。但是很遗憾，这些选择符样式在 IE6 及其以下版本中不支持，因而影响了它们的推广使用。

11.3 CSS 框架的可维护性及其设计

一个好的 CSS 框架能够保证任何人都能看得懂，而不仅仅是作者自己或者团队内部才能够看明白。也许你认为，我的 CSS 框架仅供自己使用，干嘛还要照顾别人的情绪呢？是的，如果时间久了，当你再回头来看看自己曾经开发的 CSS 框架，也许你会被各种奇怪的名字难住了，或者迷惑于这些框架的作用或目的是什么。只有别人能够看得懂的 CSS 源代码，当你忘记它们时，回头才能够读懂它们。

要保证 CSS 框架具有可维护性，应该对于每行代码都进行详细的注释和说明，让别人看得懂，也就保证自己将来能够轻松浏览。要注意类和 ID 名称的可读性，注意代码的版式、格式和结构，使它们更适应既定的习惯或团队规则。

11.3.1 CSS 框架的注释技巧及其可读性

在 CSS 中增加注释很简单，所有被放在“/*”和“*/”分隔符之间的文本信息都被称为注释，例如：

```
/* 注释 */
```

或

```
/*
注释
*/
```

CSS 注释也比较讲究，不是随意添加。首先，看看注释的位置，主要包括样式表首行、样式前、样式中等，例如：

```
/*
css Zen Garden default style - 'Tranquille' by Dave Shea - http://www.mezzoblue.com/
*/
```

或

```
/* 网页基本属性 */
html {
    margin: 0;
    padding: 0;
}
```

或

```
html { /* 网页基本属性 */
    margin: 0;           /* 清除外边距 */
    padding: 0;          /* 清除内边距 */
}
```

一般来说,样式表首行注释主要是对样式表进行摘要,包括样式表名称、URL、作者、样式表的作用、效果和一些必要的说明等。

样式前的注释主要针对该样式进行简单的说明,包括样式的作用、对象和特殊说明等。而在样式中的注释主要对样式中的某个声明进行说明。

有的设计师喜欢把注释放在属性或属性值中间,甚至利用它来进行样式过滤,这些用法比较特殊,就不再详细讲解,可以参阅第9章相关内容。

注释的语法虽然简单,但是写法可以多样,通过变化形式能够使注释更容易阅读,或者提高代码的段落层次。

例如,下面是YAML CSS框架的首行注释,分别包括名称、简单说明、用法提示、版权信息、附属信息等。

```
/**
 * "Yet Another Multicolumn Layout" - (X)HTML/CSS Framework
 *
 * (en) YAML core stylesheet
 * (de) YAML Basis-Stylesheet
 *
 * Don't make any changes in this file!
 * Your changes should be placed in any css-file in your own stylesheet folder.
 *
 * @copyright      Copyright 2005-2008, Dirk Jesse
 * @license        CC-A 2.0 (http://creativecommons.org/licenses/by/2.0/),
 *                 YAML-C
 * (http://www.yaml.de/en/license/license-conditions.html)
 * @link           http://www.yaml.de
 * @package        yaml
 * @version        3.0.5
 * @revision        $Revision: 189 $
 * @lastmodified    $Date: 2008-05-24 08:26:23 +0200 (Sa, 24 Mai 2008) $
 */
```

另外还可以利用注释来区分CSS代码块,如下所示:

```
/* 大模块名称
-----*/
.....
/* 小模块名称
-----*/
.....
/* 样式名称 */
.....
```

注释的形式是多样的,可以利用各种符号设计出各种图画式的注释效果。注意,注释的最终目的是方便阅读,而不是美化CSS样式代码。

11.3.2 CSS 框架中类名和 ID 名的可读性

CSS 类名和 ID 名首先应该遵循 CSS 语法规则的基本要求,一般应使用字母、数字和下划线,其他特殊字符应尽量少用。某些特殊键盘字符由于具有特定的功能,不要随意使用。另外,

首字符应该是字母或下划线，不能够使用数字。

CSS 命名的第一原则就是能够达到用户看名知其意，简单说就是 CSS 名字的语义性。所谓语义性就是名称应该代表一定的意思，或者看到名称就大致知道它的意思。例如，red 类就比 color1 类更具语义性，因为用户一看大致知道该类是定义一个红色字体类。

另外，在语义性和结构性选择方面，应该以名称的语义性为主、结构性为辅的原则来进行设计。再例如，对于如下这个圆角区域结构，有读者使用如下的结构式命名规则来进行设计：

```
<style type="text/css">
.□ { /* 包含框样式 */ }
.┌ { /* 左上角样式 */ }
.┐ { /* 右上角样式 */ }
.└ { /* 左下角样式 */ }
.┘ { /* 右下角样式以及内容区域样式 */ }
</style>
<div class="□">
  <div class="┌">
    <div class="┐">
      <div class="└">
        <div class="┘">内容区域</div>
      </div>
    </div>
  </div>
</div>
```

这是使用 Unicode 字符来进行命名，看起来更直观和简洁，但是由于这种方法缺乏语义性，一般不建议使用。恰当的方法应该使用复合词进行命名：

```
<div class="container">
  <div class="left-top">
    <div class="right-top">
      <div class="left-bottom">
        <div class="right-bottom">内容区域</div>
      </div>
    </div>
  </div>
</div>
```

除了要注意名称的语义性外，还应该注意名称的形式，名称可以使用英文、拼音或缩写，这里没有优先之分，根据个人喜好和习惯而定。一般建议使用英文名称，这样更具通用性，如果英文单词太长，可以适当进行缩写或者截取。

当单个单词无法表示名称时，可以使用复合词，多个复合词之间可以通过首字母大写或者连字符进行区分。请注意，CSS 是区分大小写的。例如：

```
#pageHeader { /* 样式声明 */ }
#PageHeader { /* 样式声明 */ }
#page_header { /* 样式声明 */ }
#page-header { /* 样式声明 */ }
```

11.3.3 CSS 框架代码的排版格式应简明直观

一般情况下，CSS 代码有两种排版格式：单行样式和多行样式。

所谓单行样式，就是一个样式的所有声明都被放在一行内显示。例如：

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

```
a { font-weight: bold; text-decoration: none; color: #B7A5DF; }
acronym { border-bottom: none; }
```

所谓多行样式，就是一个样式的所有声明都被单独放在一行内显示，例如：

```
body {
    margin: 0;
    padding: 0;
}
a {
    font-weight: bold;
    text-decoration: none;
    color: #B7A5DF;
}
acronym {
    border-bottom: none;
}
```

多行样式还可以有多种版式效果，例如，如果一个样式仅包含一个声明，则不再单独分行显示，或者对于大括号也单独显示在一行，如下所示：

```
body
{
    margin: 0;
    padding: 0;
}
a
{
    font-weight: bold;
    text-decoration: none;
    color: #B7A5DF;
}
acronym { border-bottom: none; }
```

也许你会觉得这样的排版格式很复杂,操作起来很麻烦。实际上如果借助 Dreamweaver CS3 所提供的代码格式化工具,会非常方便和快速(如图 11.3 所示)。

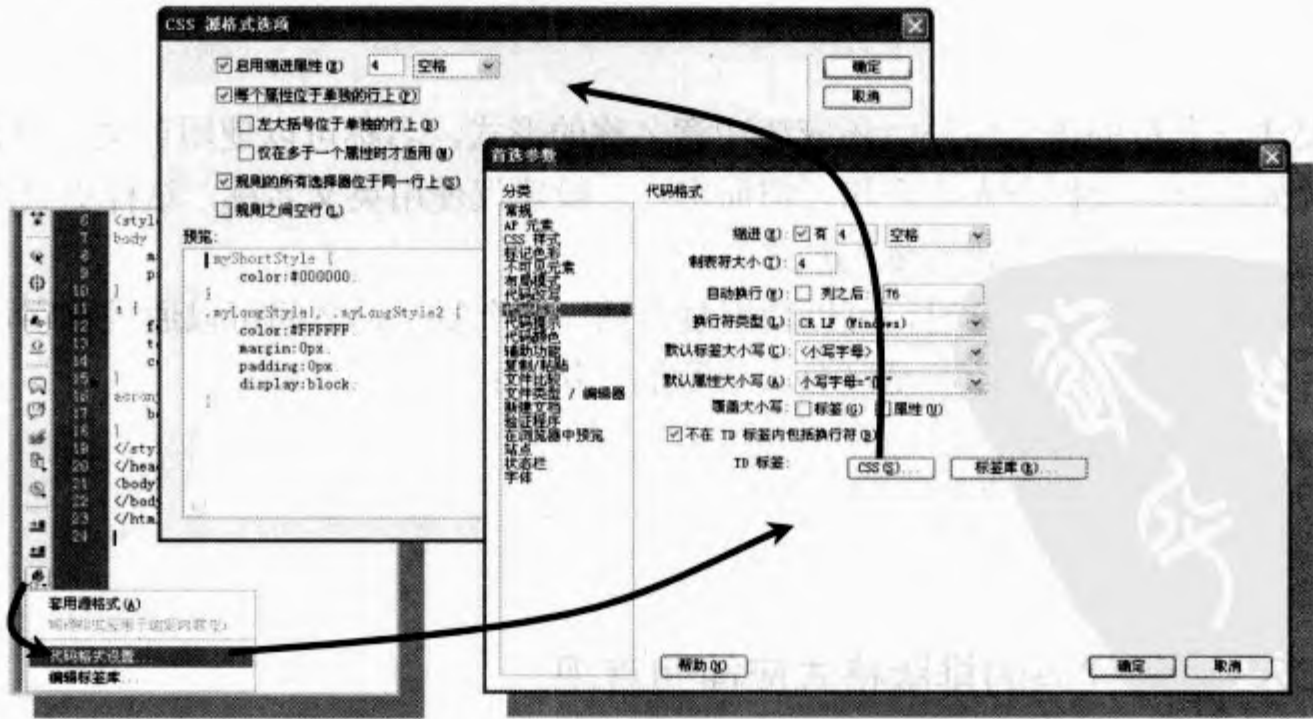


图 11.3 在 Dreamweaver CS3 中设置 CSS 格式

首先在【代码】视图下左侧的【编码】工具条中单击【格式化源代码】按钮 ()，然后

在弹出的下拉列表中选择【代码格式设置】选项，打开【首选参数】对话框，最后根据下图所示进行设置。设置完毕后再次单击【格式化源代码】按钮(🔧)，再选择【套用源格式】选项即可。

11.3.4 框架内样式代码的结构顺序应符合一定的阅读习惯

有关样式代码的结构顺序很重要，位置先后会影响到它们的优先级别。但是本节所要讨论的是在不影响它们的优先级的情况下，如何去定义这些样式顺序。

根据大部分设计师的设计习惯，在一个样式表中框架元素的默认样式应该放在前面，然后是各种类样式，接着再定义页面结构的 ID 样式和细节样式（如图 11.4 所示）。

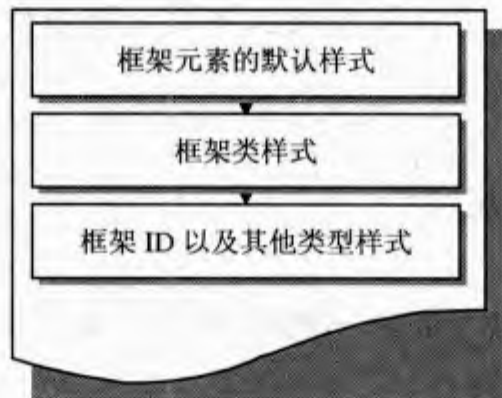


图 11.4 样式表中样式的排列顺序

当然，这个问题不是很关键，你可以根据团队的阅读习惯进行定义，没有统一的要求。上面所列形式仅是一种建议。

除了类型样式应适当注意排列顺序之后，还应注意样式内部的不同声明之间的排列顺序。具体方法可以根据习惯而定，这个就更没有规律可言了。提供两种排序思路：一是根据属性的性质进行排序，例如，把所有的布局属性放在前面，或者把所有字体属性放在前面，然后是段落属性、布局属性等；二是根据属性首字母的顺序进行排列。

11.4 CSS 框架的可扩展性和兼容性及其设计

CSS 框架的可扩展性体现在 CSS 代码能够自由地增加和删除，以及样式的继承性。对于编程语言来说，一般通过类的继承特性来实现功能的可扩展性。CSS 也有继承功能，但是没有编程语言中类的继承那么复杂和强大。另外，通过样式表文件的设计，也可以优化 CSS 框架的可扩展性。

11.4.1 CSS 的继承性及其利用

在第 1 章中我们曾经简单讲解过，CSS 具有继承性。如何利用 CSS 的继承性，发挥其优势以实现 CSS 框架的可扩展性呢？下面我们就来详细讲解。

CSS 的继承性体现在结构关系上，且与属性本身存在很大联系，这与编程语言中的继承性存在很大的不同。CSS 所定义的 100 多种属性中，只有一部分具有继承性，而不是全部属性都具有继承性（可以参阅本书光盘附赠的 CSS 参考手册）。拥有继承性的属性包括如下几大类：

- (1) 字体属性；
- (2) 文本属性（大部分属性，个别属性不支持继承）；

- (3) 表格属性（大部分属性，个别属性不支持继承）；
- (4) 列表属性；
- (5) 打印属性（部分属性支持继承）；
- (6) 声音属性（部分属性支持继承）；
- (7) 另外鼠标样式也具有继承性。

而对于盒模型、布局、定位、背景、轮廓和内容等类属性都不具备继承性。

至于 CSS 继承的结构性，主要体现在内部结构会自动继承外部结构的可继承属性（这在第 1 章中已详细讲解）。因此，当我们希望统一整个 CSS 框架的字体、字号、字体颜色、行高等基本样式时，不妨在 body 元素中进行定义，然后通过继承性实现网页内字体属性的统一。

现在问题就来了，如果通过继承性实现网页字体、文本样式的统一，那么我又希望某个栏目的字体显示不同，该怎么办呢？

这时你有两种选择方法：

- (1) 在对应的结构 ID 中进行定义；
- (2) 通过专用类进行定义。

具体哪种方法比较好，可通过下面的示例来说明。如图 11.5 所示，这是一个简单的 3 行 3 列的固定宽度布局。结构如下所示：

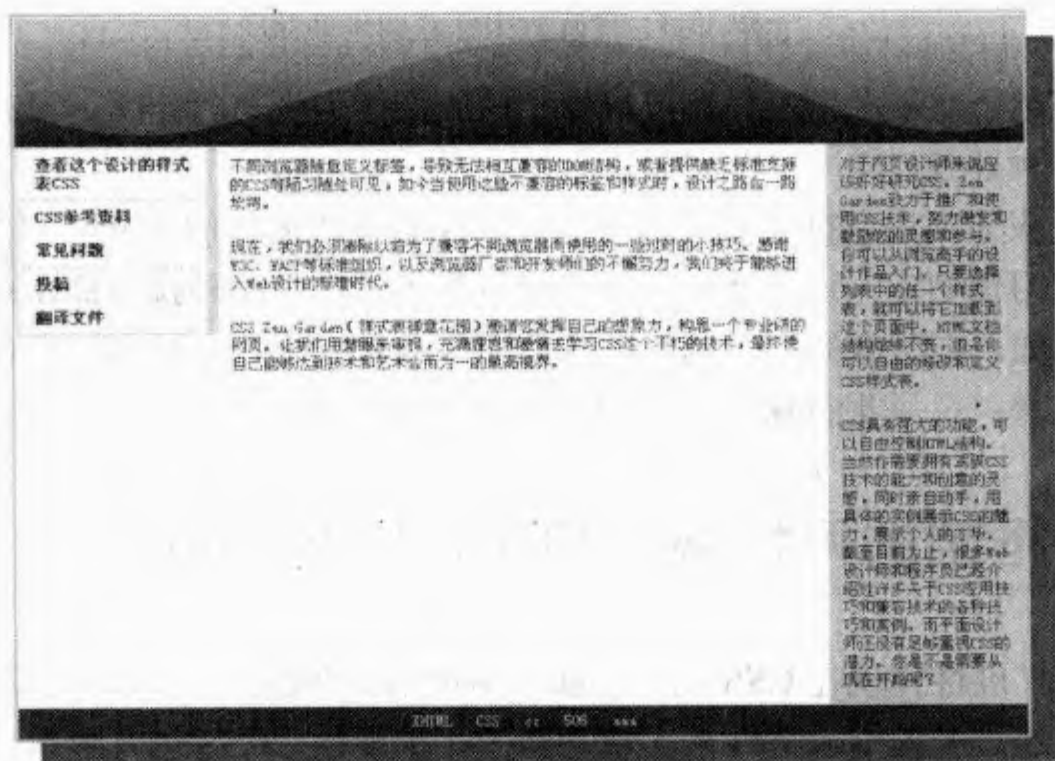


图 11.5 样式表中样式的排列顺序

```
<div id="container">
  <div id="header">
    <h1>网页标题</h1>
  </div>
  <div id="wrapper">
    <div id="content"> </div>
  </div>
  <div id="left">
    <div id="sidebar"> </div>
  </div>
  <div id="right" class="black"> </div>
  <div id="footer"></div>
</div>
```


在 body 中定义了字体的默认样式:

```
html, body {
    margin: 0; /* 清除外边距 */
    font-size: 12px; /* 定义默认字体大小 */
    font-family: "宋体", Arial, Helvetica, sans-serif; /* 定义默认字体 */
    color: #666; /* 定义字体默认颜色 */
}
```

你可以看到在这个布局模板中, 由于页面统一了字体的样式, 因为 CSS 的继承性, 所有栏目的字体显示同样的效果, 但是由于不同栏目的背景色不同, 所设置的字体颜色显示效果截然不同。例如, 右栏字体显示为灰色, 但是背景色为浅蓝色, 导致字体显示不是很清楚。

这时就应该打破这种继承性, 重新为右栏定义其他显示颜色。可以在右栏结构的 ID 中进行定义:

```
#right {
    color: #000;
}
```

或者定义一个专用类:

```
.black {
    color: black;
}
```

然后在结构中引用该类:

```
<div id="right" class="black"> </div>
```

如果仅就这个页面来说, 直接在结构 ID 中进行定义会方便许多, 但是对于 CSS 框架来说, 使用专用类来弥补 CSS 继承性是最佳选择。因为在一个框架中可能会有多处引用, 通过类的方式提炼这个样式, 就能够达到最优化应用。

类似这样的问题还很多, 例如, 通用行高与个别栏目的特殊行高、网页默认字体大小与特定栏目的特殊字体大小等。所以希望读者应该认真思考这个问题。我的建议如下。

首先, 使用 CSS 继承性来统一 CSS 框架中的基本样式。

然后, 对于特定页面、特定栏目所需要的特殊样式, 可以通过重新定义的方法来修正继承所带来的问题。

最后, 如果这种特殊样式使用比较普遍, 一个页面中超过了 2 次, 则建议通过定义类的方式实现修正; 如果这种特殊样式使用不是很普遍, 一个页面仅使用 1 次或 2 次, 则建议在结构的 ID 中进行定义, 以避免类的泛滥。

11.4.2 CSS 框架的继承与包含关系及其应用

上一节讨论了 CSS 继承性在框架中的应用, 当然继承性在网页中应用还是比较广泛的, 其中涉及很多技巧。除了继承性之外, CSS 的包含性也是一个重要应用话题。

我们知道 CSS 虽然不是一门“正宗”的编程语言, 但是它也具备编程语言的一般特征, 例如, CSS 也有运算符。

实话讲, CSS 语言的运算符不多, 包括点运算符 (.)、井号运算符 (#)、大括号运算符 ({}), 冒号运算符 (:), 以及常用的冒号 “:”、分号 (;)、逗号 (,)、中括号 ([])、尖角号 (>) 等, 这些运算符在第 1 章中都曾经介绍过, 都是常用的定义声明的符号。

另外, 还有一个空格号, 也许在其他语言中空格是会被忽略的, 但是在 CSS 中, 它具有特

殊的作用，我们可以把它看做是编程语言中的命名空间或类中的点号（.）运算符。通俗说，就是可以把空格看作路径指向的箭头，表示 HTML 标签的结构关系。

CSS 是与 HTML 紧密相关的，也就是说，CSS 的每一个样式都是与 HTML 元素或者对象相对应，而 HTML 可以调用多个样式类。一个 CSS 样式类可以根据 HTML 代码来进行复合定义，一个 HTML 标签也可以复合调用多个样式类。因此，CSS 样式定义的复杂性与关联的 HTML 密不可分。

这种复杂性给 CSS 框架的应用带来很大挑战，如何在复杂的 HTML 结构中提炼出更精确的 CSS 包含样式呢？下面我们来看一个示例。这是一个典型的布局结构，在每个栏目中都绑定了一个 red 类。

```
<div id="container" class="red">网页包含框
  <div id="header">
    <h1 class="red">页眉区域</h1>
  </div>
  <div id="wrapper">
    <div id="content" class="red">1. 主体内容区域 </div>
  </div>
  <div id="navigation" class="red">2. 导航栏 </div>
  <div id="extra" class="red">3. 其他栏目 </div>
  <div id="footer" class="red">页脚区域 </div>
</div>
```

下面利用 CSS 包含关系，定义一组包含样式：

```
<style type="text/css">
/*-----
    <<< 一组 CSS 包含样式 >>>
    -----*/

.red { /* red 类 */
    color:red; /* 红色字体 */
}
div.red { /* 仅作用于 div 元素的 red 类 */
    color:blue; /* 蓝色字体 */
}
div .red { /* 包含在 div 元素内的 red 类 */
    color:orange; /* 橙色字体 */
}
.red div { /* red 类中包含的 div 元素 */
    color:yellow; /* 黄色字体 */
}
div div .red { /* 包含在双层 div 元素嵌套内的 red 类 */
    color:purple; /* 紫色字体 */
}
div div h1.red { /* 包含在双层 div 元素嵌套内的、且仅作用于 h1 元素的 red 类 */
    color:green; /* 绿色字体 */
}
</style>
```

那么在这一组样式中，如果发生样式层叠之后，该如何判断对象的实际呈现样式呢？

首先，我们来计算这一组样式的优先级加权比值：

.red = 10

div.red = 1 + 10 = 11

div.red = 1 + 10 = 11

`.red div = 10 + 1 = 11`

`div div .red = 1 + 1 + 10 = 12`

`div div h1.red = 1 + 1 + 1 + 10 = 13`

你可以看到, 这些样式选择符的加权比值大小一目了然, 当加权比值相同时, 则根据位置的先后关系来确定, 位置靠后的优先级就大。

但是有一点比较特殊, 对于选择符 `div.red` 和 `div .red` 选择符来说, 在 IE 中 `div.red` 的优先级要大于 `div .red`, 虽然 `div.red` 复合选择符位于 `div .red` 复合选择符的前面, 这是因为 `div.red` 具有更大的优先权。具体原因不明, 可能是 IE 的一个解析 Bug。所以最后我们看到的结果如图 11.6 所示。其中“网页包含框”显示为蓝色, 网页标题显示为绿色, 左侧区域文本显示为紫色, 而右侧区域文本显示为黄色, 页脚区域文本也显示为黄色。

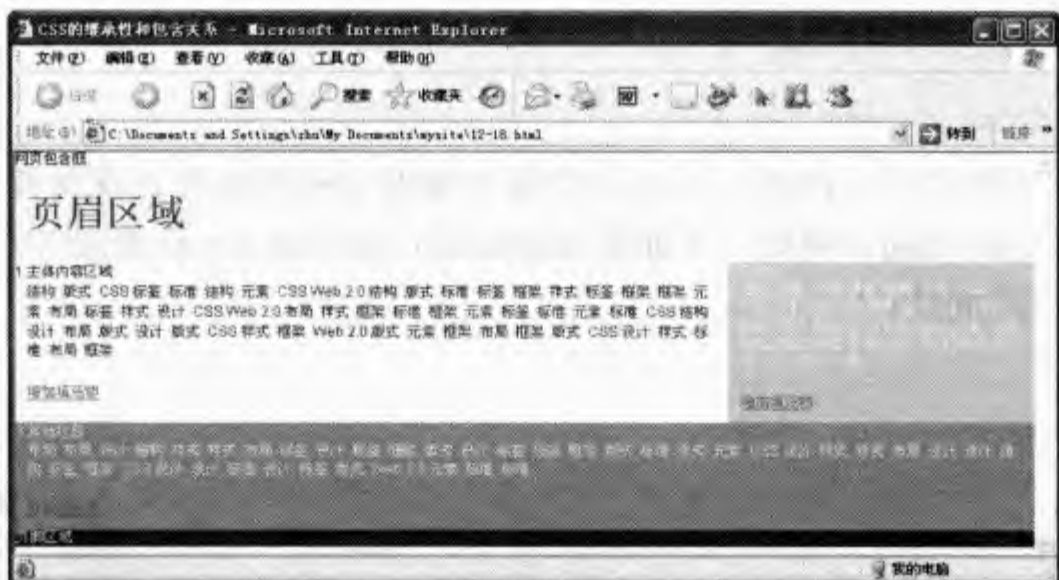


图 11.6 CSS 样式的继承性和包含关系

经过上面的试验, 我们可以这样总结: 在 CSS 框架中, 要灵活使用 CSS 继承性和包含关系, 可以设计出更具灵活性的框架样式。在设计多层包含关系的选择符时, 一定要注意 CSS 优先级, 当样式发生层叠时, 能够保证 CSS 框架在解析时不至于出现各种异常效果。

同时我们也可以看到在包含选择符中, 空格作为 HTML 结构的路径, 明确表明网页结构的父子级别关系。而对于没有空格 (如 `div.red`) 则表示同一级路径, 所以没有空格则表示 HTML 类或 HTML 的 ID 自身所代表的元素, 不过它要比单独指定元素的名称的优先级要大。很多时候, 设计师可以利用这种设计技巧来增强元素的优先级。

在 HTML 代码中, 同样都是 `red` 类, 但是在 CSS 定义时, 采用的类路径不同, 作用也不同。类路径越完整, 优先级越高。在具体应用的时候, 我们可以使用完整类路径来定义特殊的样式, 以修正 CSS 框架中因为继承性可能带来的样式层叠弊端。

11.4.3 通过样式表的分类、组织实现 CSS 框架的可扩展性

对于完整的 CSS 框架来说, 所有的样式一般都被保存在不同的样式表文件中, 并生成系列样式表文件, 所以如何组织 CSS 框架的样式表文件也是一个重要话题。

为了 CSS 框架的可扩展性, 我们不妨遵循这样的设计原则: 一个全局、多个扩展。

所谓一个全局, 就是由一个样式表文件来统领全部样式, 其他样式表文件作为分支分别负责某项功能块的样式。例如, 在 Elements CSS Framework 中 (<http://elements.projectdesigns.org>) `global.css` 就是一个全局文件, 其他文件为功能文件, 然后通过导入语句, 把需要的样式表文件导入到全局文件 (`global.css`) 中, 通过这个全局文件再导入到网页中。

```
@import url("/css/reset.css");
@import url("/css/externalLinks.css");
```

所谓多个扩展，其实就是除了全局样式表文件之外的其他 CSS 文件。这样你可以根据功能来设计不同的样式表文件，然后根据需要分别在不同页面中引入不同的功能样式表文件。

如果在网页中引入多个样式表文件会非常麻烦，也不利于管理，通过把扩展样式表文件导入到全局样式表文件中，再在页面内导入全局样式表文件后，在管理时就只需要针对全局样式表文件即可。

例如，我们把页面的不同区域（如侧栏、页眉区域、页脚区域）的样式写在不同的文件（如 bar.css、header.css、footer.css）。如果按传统方法，需要在网页中使用 3 个 <link> 标签来链接。现在我们只导入 global.css 文件，然后只需要在 global.css 文件里加入：

```
@import url("header.css") ;
@import url("bar.css") ;
@import url("footer.css") ;
```

同时在全局样式表文件的顶部可以设计 CSS 框架的基本默认样式以及类样式，因为这些样式在不同页面中都是需要的。例如，下面是 Elements CSS Framework 框架中的全局样式表文件（global.css）部分代码摘录：

```
/*-----
Name: global.css
Developed by:
Date Created:
Last Updated:
Copyright:
-----*/
/* Imports      (导入扩展样式表文件)
-----*/
@import url("/css/reset.css");
@import url("/css/externalLinks.css");
/* Elements    (元素默认样式)
-----*/
body
{
    background-color:#FFFFFF;
}
body, p, td, th, li
{
    font-family: Arial, Helvetica, sans-serif;
    font-size:.875em;
    line-height:1.5em;
    color:#000000;
}
/* Standard Definitions (标准定义，即类样式)
-----*/
.left      {float:left;}
.right     {float:right;}
.clearThis {clear:both;}
.small     {font-size:.625em;}
.large     {font-size:1em;}
.soft      {color:#D3D3D3;}
.hide      {display:none;}
p.last     {margin-bottom:0px;}

/* 样式表名称 */
/* 开发者 */
/* 创建日期 */
/* 更新日期 */
/* 版权信息 */

/* 向左浮动类 */
/* 向右浮动类 */
/* 清楚浮动类 */
/* 小字体类 */
/* 大字体类 */
/* 软件色类 */
/* 隐藏类 */
/* 段落最后行类 */
```


除了根据网页独立区域分别定义样式表外,还可以根据功能、效果、类型等来定义不同的样式表扩展。具体如下。

(1) 文本格式化样式扩展 (reset.css)。

对 CSS 框架中的字体、文本、文本版式等样式进行统一处理。这样在设计一个项目时,就不需要再去考虑这些细节问题了,加快项目的开发进度,同时也实现项目版式的统一。

(2) 布局样式扩展 (layout.css)。

我们在前面几章中曾经详细讲解不同的布局类型,如果在一个扩展样式表文件中把常用类型的布局统一起来,则使用时直接引用即可。例如,2 列、3 列、多行、混合布局,以及是全屏还是固定宽度等。

一个网站的设计可能有很多种布局,但是大多数都是由几个具有复用性的布局组成,选择性地引入所需要的布局,可以很快地应用所期望的页面布局。

(3) 表格样式扩展 (table.css)。

可以把与表格样式相关的所有代码都集中到一个文件中,这样在需要时引用会非常方便。表格相关标签包括 table、tr、td、th、thead、tfoot、tbody、caption 等。

(4) 表单样式扩展 (form.css)。

与表格一样,表单的样式一般在网站中都是统一的,如文本框、按钮、复选框、单选按钮等,这些标签包括 fieldset、label、button、input、select、textarea 等。

(5) 打印样式扩展 (print.css)。

把打印页面的样式都集中到 print.css,需要打印时直接调用该文件即可。

类似的功能样式表文件还可以分列出很多,如 list.css、detail.css、register.css 等。主要看设计师的需要以最终确定建立哪些扩展样式。

11.4.4 CSS 框架的可兼容性

CSS 框架的兼容性可以通过 IE 条件语句来解决,一般可以在单独的文件中分别为 IE、IE 6、IE 7 等浏览器定义不同的样式表文件,然后在网页中通过条件语句来调用这些兼容样式表文件。详细讲解可以参阅第 9 章。

另外,由于不同浏览器存在的 Bug 很琐碎,如果在一个 CSS 框架中把所有浏览器的 Bug 都汇集起来,并提供妥善的解决方案,这对于 CSS 框架来说,又有一点复杂了。所以建议读者,在 CSS 框架的兼容性设计时,仅关注布局方面的一些 Bug 处理即可,特别是针对 IE 浏览器的不同 Bug 解决方案。过细的兼容性设计只能够给自己带来麻烦,这不是 CSS 框架设计的初衷。

11.5 CSS 框架案例解析

初学者可能感觉不到 CSS 框架带来的诱惑,甚至会感觉它过于抽象化、过于理论化,或者有点曲高和寡的感觉。但是一旦当你尝试使用 CSS 构建完整的网站之后,你会切身体会到 CSS 框架对于自己工作的重要性。

本节将讲解当前互联网上几个比较著名的 CSS 框架,以帮助你感性认识 CSS 框架的构建、思路和实现方法。目前国内 CSS 框架刚刚兴起,并引起设计师的高度重视,但是还没有比较完整、成熟的 CSS 框架,所以我们的案例也以国外 CSS 框架为主展开讲解。

11.5.1 解析 Elements CSS 框架

Elements (<http://elements.projectdesigns.org/>) 是一个非常不错的 CSS 框架，它是一个轻量级的 CSS 框架。但是这个框架比较抽象，提供的 CSS 代码主要都是一些 CSS 框架的底层设计，不会影响你的布局，而且这个框架没有提供现成的模板。换句话说，Elements 框架只帮助你完成一些最基本的 CSS 设计问题，至于页面的更详细布局还需要自己动手来设计。因此说，它非常适合那些具有一定开发经验的网页设计师使用。实际上，对于任何一个 CSS 框架，我们都不建议直接把它拿过来就用，而是根据它的设计思路，适当修改、删节、补充、完善，把它变成自己的 CSS 框架之后再使用。

Elements 框架目前最新版本为 2.0。当解压之后，可以看到图 11.7 所示的详细内容。目前，网上有很多在此基础上进行改版的 Elements 框架，请读者适当注意一下。



图 11.7 Elements 框架包含的文件

其中大部分文件夹为空，所要的 CSS 框架就放在 css 文件夹中，它共包括以下 4 个文件。

(1) global.css: 全局样式表文件。该文件包含导入扩展样式文件、框架默认样式、基本类和基本页面框架的 ID 样式。

(2) externalLinks.css: 扩展样式文件。该文件定义了超链接元素的一些基本样式。并通过属性选择器定义不同超链接类型的显示样式。

(3) lightbox.css: 扩展样式文件。该文件是一个案例样式文件，它定义了一个提示框的显示样式。

(4) reset.css: 扩展样式文件。该文件实际上是对 global.css 进行补充。它重新定义了 HTML 基本元素的默认样式。

下面我们就来详细讲解这些样式表文件的代码。首先，来看看全局样式表文件(global.css)。这个样式表文件的设计思路为：导入必要的扩展样式表文件 → 初始化框架元素的默认样式 → 框架基本类样式 → 页面基本结构 ID 样式。

详细代码如下（读者在阅读这些代码时，请学习它的排版顺序以及设计技巧和习惯）：

```
/* Imports
-----*/
@import url("/css/reset.css");
@import url("/css/externalLinks.css");
/* Elements
-----*/
body { background-color:#FFFFFF; }
body, p, td, th, li { font-family: Arial, Helvetica, sans-serif; font-size:.875em;
line-height:1.5em; color:#000000; }
p { margin: 0 0 1em 0; }
a:link, a:visited { color:#B1DA67; }
a:hover, a:active { color:#5C8127; text-decoration:none; }
h1 { color:#5C8127; margin:.825em 0 .5em 0; font-size:2.125em; }
h2 { color:#5C8127; margin:.825em 0 .5em 0; font-size:1.75em; }
```



```

h3 { color:#5C8127; margin:.825em 0 .5em 0; font-size:1.5em; }
h4 { color:#5C8127; margin:.825em 0 .5em 0; font-size:1.25em; }
h5 { color:#5C8127; margin:.825em 0 .5em 0; font-size:1.125em; }
h6 { color:#5C8127; margin:.825em 0 .5em 0; font-size:1em; }
ul { margin-left:25px; list-style-type:none; }
ol { margin-left:25px; }
blockquote { margin: 0 0 18px 18px; color:#666666; font-style: italic; }
strong { font-weight:bold; }
em { font-style:italic; }
/* Standard Definitions
-----*/
.left { float:left; }
.right { float:right; }
.clearThis { clear:both; }
.small { font-size:.625em; }
.large { font-size:1em; }
.soft { color:#D3D3D3; }
.hide { display:none; }
p.last { margin-bottom:0px; }

```

在上面代码中请注意如下两点。

第一，在 body 中定义背景色为白色，在 body、p、td、th 和 li 元素中定义前景色为黑色。你可能认为这是多此一举，实际上这是标准要求，是网页可用性的体现，因为有一些网页代理（如浏览器）默认的页面前景色和背景色不是黑色和白色，读者可以在此基础上进行修改。

第二，就是 font-size 的问题。为了让网页更好地支持网页缩放功能，应该使用 em 来替换 px，这样能够保证 IE 6 及其以下版本浏览器也能够很好地支持网页缩放功能。浏览器的默认字体大小都是 16px，所以未经调整的浏览器会显示 1em=16px。换算过来的话也就是说 1px=0.0625em，也就是 12px=0.75em，10px=0.625em，通过 1px=0.0625em 这个恒等式，读者可以在 CSS 代码中把 px 转换成为 em。

至于页面基本框架，它把页面分为头部（#header）、导航条（#nav）、主体区域（#mainContent）和页脚（#footer）4 大版块（如图 11.8 所示），请注意结构 ID 样式的排版格式。

```

123 /* Header
124 -----*/
125 #header {
126     width:900px;
127     height:100px;
128 }
129
130 /* Nav
131 -----*/
132 ul#nav {
133     ul#nav li {float:left; list-style-type:none; margin-left:10px;}
134     /* Image Replacement
135     -----*/
136     ul#nav li a {height:20px; display:block; text-indent:-9999px; outline:none;}
137     /*Include all of your links with unique IDs. Use the correct path to the image and add the width of each image.
138     li#navHome a {background-image: url(/images/nav/home.png); width:75px;}
139     li#navWork a {background-image: url(/images/nav/work.png); width:70px;}
140     li#navServices a {background-image: url(/images/nav/services.png); width:68px;}
141     li#navContact a {background-image: url(/images/nav/contact.png); width:55px;}
142     /* CSS Builder (Spartan) - Advanced Usage
143     -----*/
144     /*Each link must have a unique ID - Include all of your links here - Roads are not assigned
145     li#navHome a: hover, li#navWork a: hover, li#navServices a: hover, li#navContact a: hover {background-position:0 -20px; /*Set the
146     height of image must be negative*/
147 }
148
149 /* Main Content
150 -----*/
151 #mainContent {
152     width:900px;
153     background-color:#D8ECB3;
154 }
155
156 /* Footer
157 -----*/
158 #footer {
159     width:900px;
160     height:30px;
161     background-color:#6A8900;
162 }
163
164 #footer p {
165     font-size:.625em;
166 }

```

图 11.8 Elements 基本页面的 ID 框架样式结构

实际上每个网页都需要这些基本版块，不过本框架仅提供了这样一个结构模板（或者说是

外框), 没有定义具体细节, 当然也没有这个必要。相信你在使用这些代码时, 仅借鉴它的构建方法而不会采纳它的详细样式。

reset.css 样式表扩展文件重新定义了框架基本元素的默认样式, 你也可以根据需要对其进行修改:

```
/*Elements CSS Framework by Ben Henschel*/
/*Mass Reset*/
/*Thanks to Eric for this reset http://meyerweb.com/eric/thoughts/2007/04/14/
reworked-reset/ */
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote,
pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, font, img, ins, kbd, q, s,
samp, small, strike, strong, sub, sup, tt, var, dd, dl, dt, li, ol, ul, fieldset, form,
label, legend, table, caption, tbody, tfoot, thead, tr, th, td{
    margin: 0;
    padding: 0;
    border: 0;
    outline: 0;
    font-weight: inherit;
    font-style: inherit;
    font-family: inherit;
    font-size:100%;
    text-align: left;
    vertical-align: baseline;
}
a img {border:none;}
table {border-collapse: collapse; border-spacing: 0;}
q:before, q:after, blockquote:before, blockquote:after {content: ""};
```

在这个扩展样式表文件中, 先对 HTML 基本元素的外边距、内边距、边框、轮廓清除为 0, 对于字体粗细、字体样式、字体、对齐方式、垂直对齐方式恢复到默认的样式。这样设计的目的是避免因为元素默认样式的改动或未知而对页面设计产生影响。

然后清除附有超链接的图片边框为 0, 避免因为图片被定义了超链接而显示很粗的蓝色边框。定义表格的单元格边框合并, 并清除单元格之间的间距。

externalLinks.css 样式文件是一个特殊的超链接样式文件, 为不同类型超链接定义显示不同的图标, 这样能够方便用户直观查看每个超链接的类型 (类似图 11.9 所示的标题后缀图标效果)。



图 11.9 特殊超链接的样式效果


```

/*Elements Developed by Ben Henschel*/
/*External Links*/
/*External Link - adds a little external link icon to all of your external links*/
a[href^="http:"] {background: url(/images/elementsImages/externalLink.gif)
no-repeat right top; padding-right:10px;}
/*IMPORTANT: Reset your internal links that use absolute URLs by replacing yoursite.com
with your site's URL, below*/
a[href^="http://www.yoursite.com"], a[href^="http://yoursite.com"]
{background-image:none !important; padding-right:0px;}
.exempt {background-image:none !important; padding:0px;}
/*这种特殊超链接不适用于 IE 6 或 IE 7, 你可以使用下面这个类专门为这些版本浏览器进行定义, 增加一个
特殊的图标 .*/
.external {background: url(/images/elementsImages/externalLink.gif) no-repeat right
top; padding-right:10px;}
/*Email 超链接 - 在所有 Email 超链接后面增加一个图标*/
a[href^="mailto:"] {background: url(/images/elementsImages/email_link.png)
no-repeat right top; padding-right:22px; padding-bottom:5px;}
/*AIM 超链接 - 在所有 AIM 超链接后面增加一个图标*/
a[href^="aim:"] {background: url(/images/elementsImages/group.png) no-repeat right
top; padding-right:22px; padding-bottom:5px;}
/*PDF 超链接 - 在所有 PDF 超链接后面增加一个图标*/
a[href$=".pdf"] {background: url(/images/elementsImages/page_pdf.png) no-repeat
right top; padding-right:22px; padding-bottom:5px;}
/*DOC 文档超链接 - 在所有 Word 文档超链接后面增加一个图标 */
a[href$=".doc"] {background: url(/images/elementsImages/page_word.png) no-repeat
right top; padding-right:22px; padding-bottom:5px;}
/*RSS 类型超链接 - 在所有 RSS 链接后面增加一个图标 */
a[href$=".rss"], a[href$=".rdf"] {background: url(/images/elementsImages/feed.png)
no-repeat right top; padding-right:22px; padding-bottom:5px;}

```

lightbox.css 扩展样式表文件是一个特殊的样式表类型, 这里就不再详细讲解。

11.5.2 解析 Blueprint CSS 框架

Blueprint (<http://code.google.com/p/blueprintcss/>) CSS 框架相对于 Elements CSS 框架来说要庞大许多, 另外它还提供了各种应用模板。但是所有的 CSS 框架一般都遵循“一个全局、多个扩展”的设计原则, Blueprint 的设计结构也遵循这样的设计原则。

首先, 我们来解析它的扩展样式。Blueprint 在功能上, 将布局 (layout)、排版 (typography)、表单 (formt)、重置 (reset)、打印 (print) 等功能放在不同的样式表文件中。这样实现 CSS 框架的扩展性, 方便用户按需导入所要使用的功能, 不用导入全部文件, 提高页面装载性能。

其次, 通过一个全局样式表文件 (screen.css) 作为统一的接口, 把多个分散的扩展样式表文件包含在一个文件内, 这样在页面中导入 CSS 框架样式时仅导入同样一个全局样式表文件, 而把接口内部的各个扩展的导入细节放在 screen.css 中进行处理, 统一了对外接口。

下面来简单了解 Blueprint CSS 框架所包含的样式表文件, 以及 Blueprint CSS 框架的设计思路。

(1) screen.css: 该文件是一个全局文件, 只需要在网页中包含此文件, 就可以导入整个框架。该文件定义了框架元素的默认样式和 CSS 基类。

(2) print.css: 这是一个扩展样式表文件, 用于处理打印事务。

(3) grid.css: 该扩展文件用于处理页面的布局 (栏目), 模板测试效果如图 11.10 所示。

(4) typography.css: 该扩展文件用于处理页面元素的排版, 模板测试效果如图 11.11 所示。



图 11.10 Blueprint Tests: grid.css 的模板效果

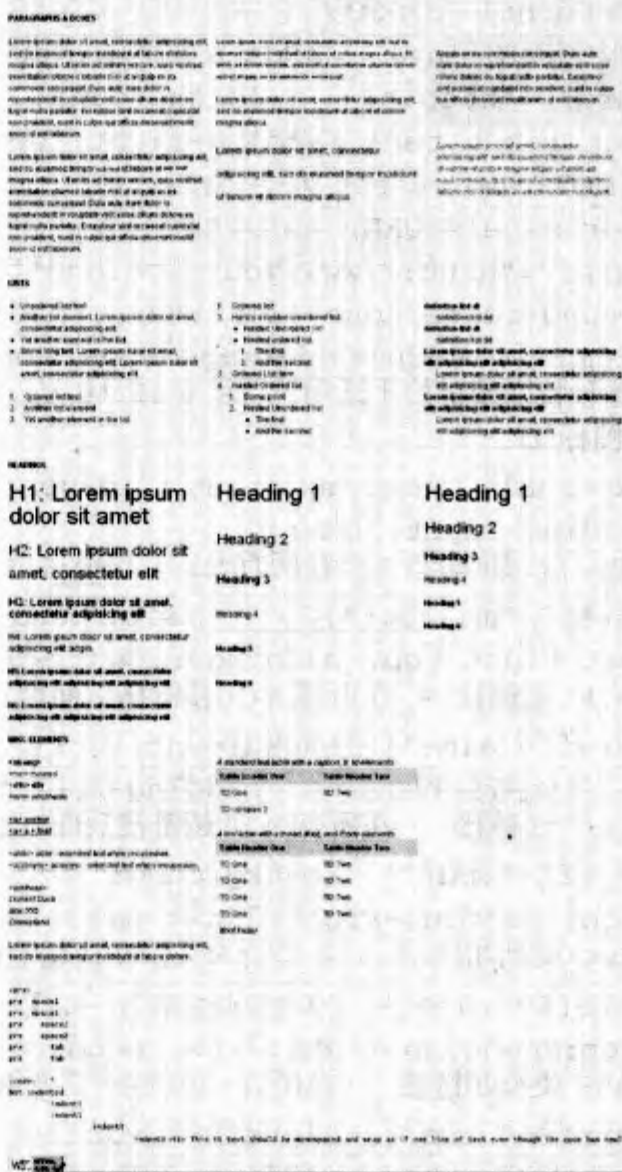


图 11.11 Blueprint Tests: typography.css 的模板效果

- (5) **reset.css**: 该扩展文件用于重置页面, 对没有指定 CSS 属性的页面元素设置默认值。
- (6) **forms.css**: 该扩展文件用于处理表单样式, 模板测试效果如图 11.12 所示。
- (7) **ie.css**: 该扩展文件提供了如何兼容 IE 浏览器的解决方案。

Forms

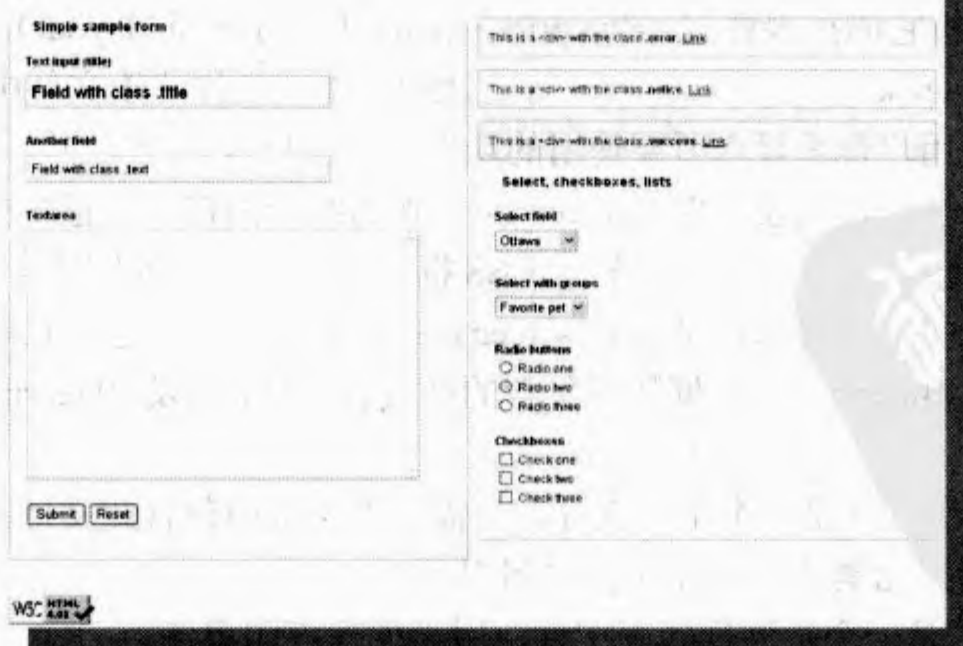


图 11.12 Blueprint Tests: forms.css 的模板测试效果

Blueprint CSS 框架在设计思路上进行如下操作。

第 1 步, 统一 CSS 框架中常用元素的默认样式。例如, 在 reset.css 文件重置 CSS 框架的默认值。

```
/* -----
reset.css
* Resets default browser CSS.
----- */
html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, code, del, dfn, em, img, q, dl, dt, dd, ol, ul, li, fieldset,
form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td {
    margin: 0;
    padding: 0;
    border: 0;
    font-weight: inherit;
    font-style: inherit;
    font-size: 100%;
    font-family: inherit;
    vertical-align: baseline;
}
body { line-height: 1.5; }
table {
    border-collapse: separate;
    border-spacing: 0;
}
caption, th, td {
    text-align: left;
    font-weight: normal;
}
table, td, th { vertical-align: middle; }
blockquote:before, blockquote:after, q:before, q:after { content: ""; }
blockquote, q { quotes: " " " "; }
a img { border: none; }
```

在上面的 reset.css 文件中统一重置了 43 个 HTML 元素 (HTML4.0 元素共计 91 个) 的默认样式。重置 html 元素的默认样式为: 内外边框都为 0, 字体粗细、字体和字体样式保留继承样式, 字体大小为 100% (可以根据需要修改为固定像素单位), 高度以基线对齐。

设置 body 内的行高为 1.5 倍, 字体大小为 75%, 字体颜色为 #222222 (深黑色)。

设置 table 元素的单元格为合并边框、间距为零。设置 caption、th、td 元素向左对齐, 字体粗细为普通。Caption 元素的背景色为 #EEEEEE (浅白色), th 元素背景为 #C3D9FF (浅蓝色), 包含字体为粗体, 以及 th 和 td 元素的内边距。设置 table、td、th 元素垂直对齐方式为居中。

设置 blockquote 和 q 元素之前和之后的填充内容为空, 引用符号也为空。设置超链接中的图片边框为 0。

第 2 步, 在 typography.css 扩展中规范了网页标题样式。这是一个很重要的步骤, 很多设计师在设计时很容易忽略这个问题。通过这种统一的样式设置, 能够减少很多工作量。

```
/* Headings
----- */
h1,h2,h3,h4,h5,h6 { font-weight: normal; color: #111; }

h1 { font-size: 3em; line-height: 1; margin-bottom: 0.5em; }
h2 { font-size: 2em; margin-bottom: 0.75em; }
h3 { font-size: 1.5em; line-height: 1; margin-bottom: 1em; }
h4 { font-size: 1.2em; line-height: 1.25; margin-bottom: 1.25em; height: 1.25em; }
```

```
h5 { font-size: 1em; font-weight: bold; margin-bottom: 1.5em; }
h6 { font-size: 1em; font-weight: bold; }

h1 img, h2 img, h3 img,
h4 img, h5 img, h6 img {
    margin: 0;
}
```

设置 h1~h6 元素不同的字体大小、粗细、边距和颜色 (#111111, 接近纯黑色)。设置 h1~h6 元素所包含的图片外边距为 0。

第 3 步, 统一 CSS 框架中的文本格式和版式。这一步在 typography.css 扩展中完成。

```
/* Text elements
----- */

p          { margin: 0 0 1.5em; }
p img      { float: left; margin: 1.5em 1.5em 1.5em 0; padding: 0; }
p img.right { float: right; margin: 1.5em 0 1.5em 1.5em; }
a:focus,
a:hover    { color: #000; }
a          { color: #009; text-decoration: underline; }
blockquote { margin: 1.5em; color: #666; font-style: italic; }
strong     { font-weight: bold; }
em,dfn     { font-style: italic; }
dfn        { font-weight: bold; }
sup, sub   { line-height: 0; }
abbr,
acronym     { border-bottom: 1px dotted #666; }
address     { margin: 0 0 1.5em; font-style: italic; }
del         { color:#666; }
pre,code    { margin: 1.5em 0; white-space: pre; }
pre,code,tt { font: 1em 'andale mono', 'lucida console', monospace; line-height: 1.5; }
```

设置 p 元素的默认外边距, 设置行内图片元素默认向左浮动。同时定义了 p img.right 类, 以备需要图片向右浮动时使用。

设置超链接的颜色为 #000099 (深蓝色)、带下划线, 鼠标滑过和获取焦点状态颜色为黑色。

设置 abbr 和 acronym 元素的下边框, 制作虚线下划线效果。

设置 del 元素删除字的颜色为 #666666 (中灰色)。

设置 pre 和 code 元素包含文本为 white-space:pre (不换行)。

第 4 步, 统一 CSS 框架中项目列表的基本样式。这一步在 typography.css 扩展中完成。

```
/* Lists
----- */

li ul,
li ol    { margin: 0 1.5em; }
ul, ol   { margin: 0 1.5em 1.5em 1.5em; }
ul        { list-style-type: disc; }
ol        { list-style-type: decimal; }
dl        { margin: 0 0 1.5em 0; }
dl dt     { font-weight: bold; }
dd        { margin-left: 1.5em; }
```

第 5 步, 统一 CSS 框架中表格的基本样式。这一步在 typography.css 扩展中完成。

```
/* Tables
```



```

----- */
table      { margin-bottom: 1.4em; width:100%; }
th         { font-weight: bold; background: #C3D9FF; }
th,td      { padding: 4px 10px 4px 5px; }
tr.even td { background: #E5ECF9; }
tfoot      { font-style: italic; }
caption    { background: #eee; }

```

第 6 步, 预定义 CSS 框架基本类。这一步在 `typography.css` 扩展中完成。

```

/* Misc classes
----- */
.small      { font-size: .8em; margin-bottom: 1.875em; line-height: 1.875em; }
.large      { font-size: 1.2em; line-height: 2.5em; margin-bottom: 1.25em; }
.hide       { display: none; }
.quiet      { color: #666; }
.loud       { color: #000; }
.highlight  { background:#ff0; }
.added      { background:#060; color: #fff; }
.removed    { background:#900; color: #fff; }
.first      { margin-left:0; padding-left:0; }
.last       { margin-right:0; padding-right:0; }
.top        { margin-top:0; padding-top:0; }
.bottom     { margin-bottom:0; padding-bottom:0; }

```

`small` 类表示小号字。

`large` 类表示大号字。

`hide` 类表示不显示。

`quiet` 类表示字体颜色为 #666666 (深灰色)。

`loud` 类表示字体颜色为 #000000 (黑色)。

`highlight` 类表示背景色为 #FFFF00 (黄色)。

`added` 类表示背景色为 #006600 (绿色), 字体颜色为 #FFFFFF (白色)。

`removed` 类表示背景色为 #990000 (红色), 字体颜色为 #FFFFFF (白色)。

`first` 类表示左边的内外边距都为 0。

`last` 类表示右边的内外边距都为 0。

`top` 类表示上面的内外边距都为 0。

`bottom` 类表示下面的内外边距都为 0。

上面这几步是一个 CSS 框架所必须的设置要素, 下面我们再看看 Blueprint CSS 框架中基本布局、表单等样式设置。

Blueprint CSS 框架通过一个 `container` 类来控制页面的整个宽度和显示(参阅 `grid.css` 样式表文件)。

```

/* A container should group all your columns. (这个类包含了所有列) */
.container {
    width: 950px;
    margin: 0 auto;
}

```

`container` 类定义了页面宽度为 950 像素、居中显示模式。这个类应该应用于页面 `body` 元素内的页面包含框, 进行整体控制。

```

/* Use this class on any div.span / container to see the grid. */

```

```
.showgrid {
  background: url(src/grid.png);
}
```

showgrid 是一个无用的类，主要用来演示该框架的一个辅助，可以将这个类删除。

```
body {
  margin: 1.5em 0;
}
```

body 元素设置为上下外边距为 1.5em、左右外边距为 0。然后在 grid.css 扩展样式表文件中定义了 24 个不同宽度的单元（源代码参阅 grid.css 文件），这些类都是为了演示该框架使用，因此可以删除。另外还定义了一些其他一些类：

```
.box {
  padding: 1.5em;
  margin-bottom: 1.5em;
  background: #E5ECF9;
}
```

box 类顾名思义就是一个盒子，设置内边距为 1.5em，外下边距为 1.5em，背景颜色为 #E5ECF9（浅蓝色）。

```
hr {
  background: #ddd;
  color: #ddd;
  clear: both;
  float: none;
  width: 100%;
  height: .1em;
  margin: 0 0 1.45em;
  border: none;
}
hr.space {
  background: #fff;
  color: #fff;
}
```

hr 元素设置页面横线样式（1 像素灰色线）：背景色为 #DDDDDD（灰色），字体颜色为 #DDDDDD（灰色），清除两侧浮动元素，自身则不浮动显示，宽度为 100%，高度为 0.1em，外边距为 0，下外边距为 1.5em，没有边框。

hr.space 类负责设置横线中特殊样式（1 像素白色线）。这个样式是用在 <hr /> 标签内的，设置了背景色和字色都为白色。因为是用在 <hr /> 标签内，所以它同时具有 hr 的默认设置样式，只是将背景色和字色从灰色覆盖为白色。

```
.clearfix:after, .container:after {
  content: ".";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}
```

我们曾经讲解过上面这段代码的设计方法，它用来撑起包含框元素的高度。具体原因可以参阅第 7.4.6 节。这两个样式设置 clearfix 和 container 之后的表现为：内容是点号（.），显示为块状元素，高度为 0，两侧不允许有浮动元素，并隐藏显示。


```
.clearfix, .container {display: inline-block;}
* html .clearfix,
* html .container {height: 1%;}
.clearfix, .container {display: block;}
.clear {clear: both; }
```

上面这 3 个类都是用来定义布局清除功能的。其中前 2 个清除浮动比较复杂，除了清除浮动外还有一些其他的能力。基本上这几个样式的内容和先后顺序及代码层级设置完后就能在所有的浏览器下清除浮动了，具体说明如下。

`clearfix` 和 `container` 类设置行内块状显示（IE 不完全支持），但是可以在 IE 中起触发元素能够布局的特性（触发 IE 的 `hasLayout` 特性）。具体说明如下。

* `html .clearfix` 和 * `html .container` 设置在通配符下的 `html` 下的 `.clearfix` 和 `.container` 两个类样式属性高度为 1%（高度为 1% 的作用和 `zoom:1` 的作用是一样的，起触发布局的作用）。由于这两个类只能够在 IE 6 及其以下版本识别，故用来触发 IE 元素的布局特性的功能。`clear` 类设置清除左右两侧浮动元素。

最后我们再来解析以下表单元素的基本样式（参阅 `forms.css`）。

```
/* -----
forms.css
* Sets up some default styling for forms
* Gives you classes to enhance your forms
Usage:
* For text fields, use class .title or .text
----- */

label      { font-weight: bold; }
fieldset   { padding:1.4em; margin: 0 0 1.5em 0; border: 1px solid #ccc; }
legend     { font-weight: bold; font-size:1.2em; }
/* Form fields
----- */

input.text, input.title,
textarea, select {
    margin:0.5em 0;
    border:1px solid #bbb;
}
input.text:focus, input.title:focus,
textarea:focus, select:focus {
    border:1px solid #666;
}
input.text,
input.title      { width: 300px; padding:5px; }
input.title      { font-size:1.5em; }
textarea         { width: 390px; height: 250px; padding:5px; }
/* Success, notice and error boxes
----- */

.error,
.notice,
.success      { padding: .8em; margin-bottom: 1em; border: 2px solid #ddd; }
.error        { background: #FBE3E4; color: #8a1f11; border-color: #FBC2C4; }
.notice       { background: #FFF6BF; color: #514721; border-color: #FFD324; }
.success      { background: #E6EFC2; color: #264409; border-color: #C6D880; }
.error a      { color: #8a1f11; }
.notice a     { color: #514721; }
.success a    { color: #264409; }
```

这里主要设置表单元素的字体、字号、字体颜色和元素的边框、边距、宽、高等基本显示属性。同时还定义了 3 个类：error、notice 和 success。这些类通过不同的背景色和边框色来区别不同的鼠标状态。

- (1) error 类用来标识错误操作。
- (2) notice 类用来标识提示性操作。
- (3) success 类用来标识成功操作。

购物网站结构与布局实施

“学完 CSS 仍然没有思路，更不知如何下手”。这是许多初学者的心里话，每当听到读者这样的反馈信息时，我就一直在琢磨这事儿：这是为什么呢？问题出在何处呢？不可否认，当前市场上同类图书确实存在很多问题，如理论重于实践、轻浮重于务实、形式重于内容等。所举的实例华而不实，没有教给读者设计的思路和方法。另一方面，大量堆砌实例而不作解析，也是造成很多读者越学越糊涂的一个重要原因。因此，如何教会读者去设计，而不是去制作，就显得非常重要了。正所谓授人以鱼不如授人以渔，说的就是这个意思。

本章将避此类问题的重演，希望通过一个典型案例帮助读者找到 CSS 设计的思路和方法。当然最后还是需要读者自己动手才能够掌握 CSS 布局之道，而不是天上掉馅饼，等着别人给你现成的设计结果。

12.1 设计购物网站的基本结构

网站设计的第一步是做什么？

也许你觉得这个问题很可笑，但是现实中很多设计师、各种图书无一例外都在这儿犯了错误，很多人会惯性地认为网页设计的第一步是先设计图样。有的图书更是教你如何去观察图纸、如何切图、如何进行拆分，如此等等，着实让人汗颜。还美其名曰是 CSS 标准设计，怎不贻笑大方呢？

传统网页设计中设计师工作的第一步大概是构图、绘图了。也许你也有过这样的设计习惯：先把网页样图或者草图绘制出来，再构思如何来实现这样的效果。其实很多设计师一直都沿袭着这样既定的设计模式。

起点错了，学习的目标自然很难实现。可以毫不夸张地讲，这种设计思路和方法就是传统表格布局的那一套，新瓶装旧药，披着 CSS 标准的 Table 布局，读者不迷糊才怪呢。

12.1.1 网站内需分析

我觉得，当你准备设计一个网页时，你应该完全摆脱效果图的影响和束缚。许多人第一次询问客户时都说：“您想要什么的效果？”建议你改掉这种错误的惯性思维，而应询问客户：“您需要显示的网站内容是什么？您准备在页面中放些什么内容？”

是的，Web 标准设计中你不能够再继承传统布局中那种结构与表现不分的陋习，既然 Web 标准设计中，结构、表现和行为已经三分天下，所以我们也应该从主观上戒掉这种不良的思路。

回到我们的正题。本章将讲解购物网站的结构和布局的基本实施思路和方法，我们也先不

要去构思它的效果。工作的第一步，一定要分析透彻我们建立这个网站的目的、服务的对象和所要显示的内容。

这几年电子商务网站发展速度惊人。当然这也是必然，中国经济发展速度如此之快，作为与现实相对应的虚拟网络自然要更快于现实的发展速度。在电子商务类型网站中，以网上商店这种分支的网站类型最为耀眼。一方面是数量之多，令人咋舌。不管是否有过开店的经验或资本，任何人都可以在网上开个小店，先不管是否有人来买东西，也不管网店内是否货源丰富，总之先凑个热闹，学点经验，长点见识，何乐而不为。另一方面，网上商店模式逐渐成型，并诞生了很多大型网店门户，如卓越（<http://www.amazon.cn/>）、当当（<http://www.dangdang.com/>）等。这些大型门户网店对于整个 B2C 模式起到垂范和引领的作用。本章这个实例就是以卓越模式展开讲解的。

网上商店是干什么的？请不要笑话我这个愚蠢的提问，设计的第一步就应该从它开始。自然网上商店是用来卖东西的。你可能去过商场、超市、集贸市场、批发市场、专卖店、连锁店等不同类型的购物场所。虽然它们的类型不同，但是它们都无一例外的都是：存放商品、方便购物。

先不讨论价格对于购买的影响，单是商品存放是否合理、科学，购物流程是否方便用户，对于整个交易都会产生直接作用。所以说，对于一个网店来说，如何进行商品分类、如何显示商品、如何服务客户就构成了整个网页设计的灵魂。

仔细分析成熟的网店模式（如卓越），首页基本上包括 3 大块内容（如图 12.1 所示）。

(1) 商品分类：能够快速引导客户找到自己需要的商品。

(2) 商品陈列：这块内容最讲究，各家网店也不尽相同，如推荐商品、打折商品、最新商品、促销商品、畅销商品、排行榜、特类商品、特价商品等。对于这些模块的取舍和设计是整个网店设计的中心任务，因为它决定了用户的回访率和下单率。

(3) 配套服务：这块内容包括静态信息服务和动态交互服务。如何去设计也是很有讲究的，从用户体验的角度来考察，网店配套服务所蕴藏的潜力是很大的，设计师可以从中开发出更多的服务项目和衍生价值。这也是当前 Web 2.0 时代网页设计师所要攻坚的核心。



图 12.1 卓越网页设计效果

根据上面简单的内需分析，然后可以进一步来丰富这种内容需求，如图 12.2 所示。

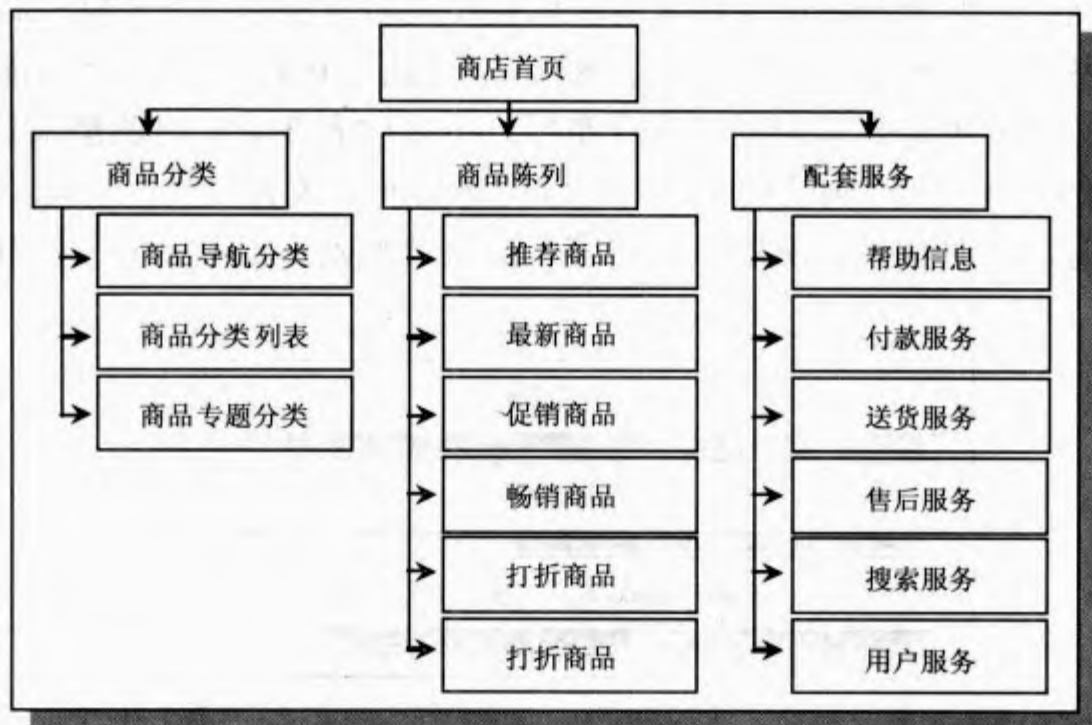


图 12.2 网店首页所包含的内容项目

在这个思路引导下，你可以继续完善和修改整个网店的内容结构。有的读者可能觉得这一步有点多余，很多人认为在设计时可以根据需要不断去增加或删除各种功能模块。这种做法无可厚非，但是很容易破坏你的 HTML 结构，会增加后期维护的成本。

12.1.2 设计网站的基本结构

清楚了要设计的网页内容，第二步应该来设计 HTML 结构了。也许你还想问：网页布局怎么实现呢？网页内容怎么显示呢？

对于这些问题，建议你还是先不要去考虑，一心一意地做好网页结构，使其更符合语义化要求，更符合 SEO（Search Engine Optimization，搜索引擎优化）要求。此时你应该完全把 CSS、样式表、效果图、布局等这些表现层的概念抛诸脑后。

第 1 步，构建页面基本框架（如图 12.3 所示）。这个结构是一个并列的 4 行框架：



图 12.3 网店一级结构示意图

```

<div id="top"> </div>
<div id="SearchBar"> </div>
<div class="StyleIndexLogin"> </div>
<div id="IndexAllWrap"> </div>
  
```

你可能会疑惑：这是一种什么结构？整个网页既没有总包含框，结构的轻重也相差悬殊，

前 3 个结构都是简单的行，而第 4 个结构块却承担着网页主体内容。是的，这是一种特殊的 HTML 结构，与我们前面介绍的网页包含框式结构布局略有不同，它把网页标题栏、搜索栏、登录条提升到与网页主体内容同等的位置，这样更有利于 SEO，加强这些功能块的重要性。这种结构适合液态布局，不适合居中固定宽度布局。所以读者在设计时应酌情进行选择。

第 2 步，完成页面一级结构的构建后，可以构建页面二级和三级主结构（如图 12.4 所示）。在构建这些结构时，应时刻考虑所展示的内容，而不要顾及页面效果对于结构的影响。



图 12.4 网店二级结构示意图

```

<div id="top">
  <div class="topmenu">
    <div class="logo"></div>
    <div id="menubar"></div>
  </div>
</div>
<div id="SearchBar">
  <div class="searchLink"></div>
  <div class="search"></div>
</div>
<div class="StyleIndexLogin">
  <div class="IndexLoginWrap"></div>
</div>
<div id="IndexAllWrap">
  <div id="Indexleftbox"></div>
  <div id="Indexmainbox"></div>
  <div id="Indexrightbox"></div>
  <div id="foot"></div>
  <div class="footmenubar"></div>
</div>

```

基本框架的搭架就这么简单，但是下面的工作是艰巨的，你需要设计每个模块的详细内容，以及选择更加符合所显示内容的语义化标签，并输入简单的模板信息，因为当模块进入实际应

用状态时,所显示的内容会被后台脚本所代替。

另外,我一直提倡把网页宏观结构和微观结构分开进行设计。所谓宏观结构就页面基本框架结构,一般包括 3~5 级的页面结构,这个宏观结构能够划分出网页的基本模块,但是每个模块内部的结构就属于微观结构了。设计之初,我们建议你应该完成宏观结构,而微观结构可以暂时不要去设计。这样避免给工作带来巨大的压力和难度。

完成宏观结构之后,可以使用 CSS 先把网页基本布局设计好,也就是说把网页基本框架给撑起来,然后再一个一个地设计模块。

12.1.3 完善语义化结构

搭建基本框架结构应从整体进行考虑,相同的网页内容可以使用多种结构来进行设计,这就要看网页内容自身包含的结构,这正如中学时期语文学习中给课文分段,归纳段落大意一样。首先,应把网页内容读懂,然后根据需要进行分段,根据内容的结构层次来构建 HTML 基本框架。

一般来说,基本框架均使用 div 元素即可。但是对于框架末端的细枝末节就不能够完全使用 div 元素来构建,应慎重行事,考虑结构的语义化问题。

例如,以“热卖商品”栏目为例(如图 12.5 所示)。在这个栏目中外框使用 div 元素来实现,栏目标题使用 h3 元素来实现。所有商品列表包含在 dl 列表元素中,商品名称使用 dt 元素来表示,而其他信息使用 dd 元素来实现。



图 12.5 “热卖商品”栏目

```
<div class="StyleHotGoods">
  <h3>热卖商品</h3>
  <div class="body">
    <div class="TPL_INDEXGOODS_STYLE_3_Wrap">
      <dl class="items">
        <dt><a href="" title="佳能 MV880Xi" target="_blank">佳能
MV880Xi<span></span></a></dt>
        <dd>会员价: ¥7900.00</dd>
      </dl>
      .....
      <dl class="items">
        <dt></dt>
        <dd> </dd>
      </dl>
      <div class="textright p14redb"><a href="">>> 查看更多热卖商品
</a></div>
    </div>
  </div>
</div>
```

在设计所有栏目时,都应该遵循语义结构化来包含信息。栏目或块区域可以使用 div 元素来设计。

结构中能否允许使用修饰性的标签呢?所谓修饰性的标签,就是这些标签的作用不是来表示语义化的结构和内容,而是为后期的布局和样式作为修饰使用的。在结构中我们可以根据需要适当使用一些修饰性标签,以帮助 CSS 完成各种特殊样式的需要。但是要注意,这种修饰性

的标签不要使用过多，且要恰当。

例如，在“商品分类”栏目中（如图 12.6 所示），为了设计栏目的圆角样式，在栏目的顶部和底部分别增加了 6 个修饰性的标签，详细代码如下。这些加粗的 `div` 元素对象在整个结构中不负责 HTML 结构和信息的管理，纯粹就是用来设计圆角效果的辅助性标签（或者称之为修饰性标签）。

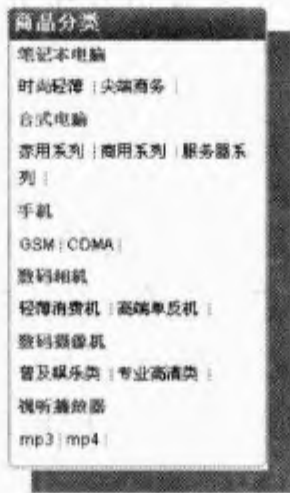


图 12.6 “热卖商品”栏目

```
<div class="StyleCategoryIndex">
  <div class="title">
    <div class="right"></div>
    <div class="left"></div>
    <div class="cent">商品分类</div>
  </div>
  <div class="body">
    <div class="CategoryIndexWrap">
      <dl>
        <dt><a href="">笔记本电脑</a></dt>
        <dd><a href="" title="时尚轻薄">时尚轻薄</a> | <a href="" title="尖端商务">尖端商务</a> | </dd>
        <dt></dt>
        <dd></dd>
        .....
      </dl>
      <div class="clear"></div>
    </div>
  </div>
  <div class="lefttable-bottom">
    <div class="tdbg-left"></div>
    <div class="tdbg-right"></div>
  </div>
</div>
```

修饰性标签在网页结构中原则上是不使用的，读者可以参考以下两个原则。

第一，在语义化结构中应遵循以内容显示为先、特殊样式设计为辅的设计宗旨。我们不能为了特殊效果而随意增加或修改文档结构，这是不可取的。

第二，如果确实需要设计某种特殊的样式，应尽力借助结构本身的元素来进行设计，而不是随意增加修饰性标签，这样可以增强结构的语义性。

一个完整的页面可能包含很多的小栏目，但是如果你设计出 1~2 个栏目模板，并借助类来进行样式控制，其他栏目就可以套用了。所以从这点来研究，庞杂的页面就是由基本的框架

结构和无数个基本栏目模板组成。剩下的工作就是去拼装这些结构和栏目了。

最后给读者几条建议, 希望能够采纳。

(1) 学会使用列表元素(如 ul、ol、dl)设计菜单或信息列表。使用列表元素的最大优点就是可以减少类的声明数量, 当把一套重复单元放进 li 之后, 在 ul 里定义一个类就足够了, 在本示例中你可以看到这样的结构。

(2) 结构之间、模块之间、栏目之间应该统一使用外边距(margi)来控制彼此的空隙, 而不要使用内边距, 这样能够方便你对于整个结构的控制。

(3) 任何结构、模块都不要显式定义高度, 除非其内部要定义背景图像, 这样就能够避免高度对于布局的影响。

(4) 在为模块或栏目定义样式时, 如果存在两个相同的样式, 则建议采用类的方式对这些样式进行提炼, 以便实施集中控制。

12.2 使用 CSS 完成基本结构的布局

符合 W3C 标准的网页应该由 3 部分组成: 结构 (Structure)、表现 (Presentation) 和行为 (Behavior)。完成 HTML 结构的设计之后, 下面才能够设计该页面结构所要呈现的效果, 即所谓的网页表现。此时如果没有 CSS 的支持, 你会发现整个页面如同没有骨架的面条, 自然而又无序地排列在一起(如图 12.7 所示)。

网页表现主要通过 CSS 技术来实现, 它的历史使命主要包括两个: 第一, 完成 HTML 基本框架的布局; 第二, 在布局基础之上, 完善页面各部分的样式。

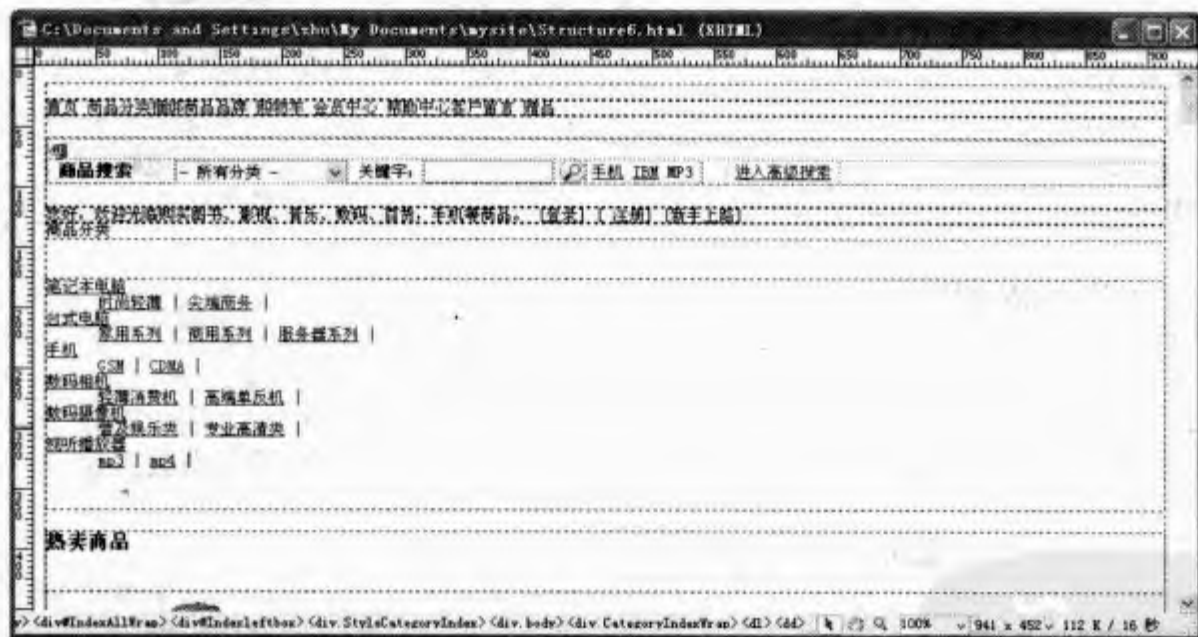


图 12.7 HTML 框架的自然显示效果

12.2.1 实施网页基本布局

本页面的基本布局比较简单, 我们可以遵循 HTML 元素自然流动的方式让<div id="top">、<div id="SearchBar">、<div class="StyleIndexLogin">和<div id="IndexAllWrap">子模块自上而下的分别排列。

然后在<div id="IndexAllWrap">子结构中定义<div id="Indexleftbox">、<div id="Indexmainbox">和<div id="Indexrightbox">并列浮动布局, 而对于<div id="foot">和<div

class="footmenubar">子模块则按默认的流动布局进行显示。结构布局示意图如图 12.8 所示。



图 12.8 网店结构布局示意图

根据上面的设计思路新建 shop.css 样式表文件，并在其中定义基本的布局样式：

```
#top { /* 顶部模块样式 */
    width:100%; /* 液态宽度 */
    position:relative; /* 定义包含块 */
    height:33px; /* 高度 */
    text-align:center; /* 文本居中对齐 */
    margin-top:10px; /* 顶部外边距 */
    background:url(az-tabs-line.gif); /* 背景图像 */
}
#SearchBar { /* 搜索条样式 */
    width:100%; /* 液态宽度 */
    height:70px; /* 高度 */
    text-align:center; /* 文本居中对齐 */
    background:url(az-subnav-bg.gif); /* 增加背景图像 */
}
.StyleIndexLogin { /* 登录条样式 */
    margin:0 auto; /* 栏目居中对齐 */
    width:850px; /* 固定宽度 */
    line-height:30px; /* 定义行高 */
}
#IndexAllWrap { /* 主体内容包含框 */
    width:1000px; /* 固定宽度 */
    margin:0 auto; /* 居中对齐 */
    overflow:hidden; /* 隐藏超出区域 */
    text-align:center; /* 文本居中对齐 */
}
#Indexleftbox { /* 左侧栏目样式 */
    width:190px; /* 固定宽度 */
    text-align:left; /* 文本左对齐 */
    float:left; /* 向左浮动 */
}
```



```

        overflow:hidden; /* 隐藏超出区域*/
    }
    #Indexmainbox { /* 中间主体栏目样式 */
        float:left; /* 向左浮动 */
        width:605px; /* 固定宽度 */
        margin-left:8px; /* 增加左侧外边距 */
        overflow:hidden; /* 隐藏超出区域 */
        text-align:left; /* 文本左对齐 */
    }
    #Indexrightbox { /* 右侧栏目样式 */
        width:190px; /* 固定宽度 */
        float:right; /* 向右浮动 */
        overflow:hidden; /* 隐藏超出内容 */
        text-align:left; /* 文本左对齐 */
    }
    #foot { /* 页脚区域模块样式 */
        margin:0 auto; /* 栏目居中显示 */
        clear:both; /* 清除左右两侧浮动 */
        margin-top:5px; /* 顶部外边距 */
        text-align:left; /* 文本左对齐 */
        padding:20px 0 0 0; /* 顶部内边距 */
        background:#fff; /* 背景色 */
        border:1px solid #ccc; /* 边框样式 */
    }
    .footmenubar { /* 页脚导航模块样式 */
        background:#ffc; /* 栏目背景色 */
        padding:5px; /* 内边距 */
        border:1px solid #ccc; /* 边框样式 */
        border-top:none; /* 清除顶部边框 */
        text-align:center; /* 顶部模块样式 */
    }
}

```

最后经过 CSS 基本布局所得到的 HTML 框架的可视化效果如图 12.9 所示。

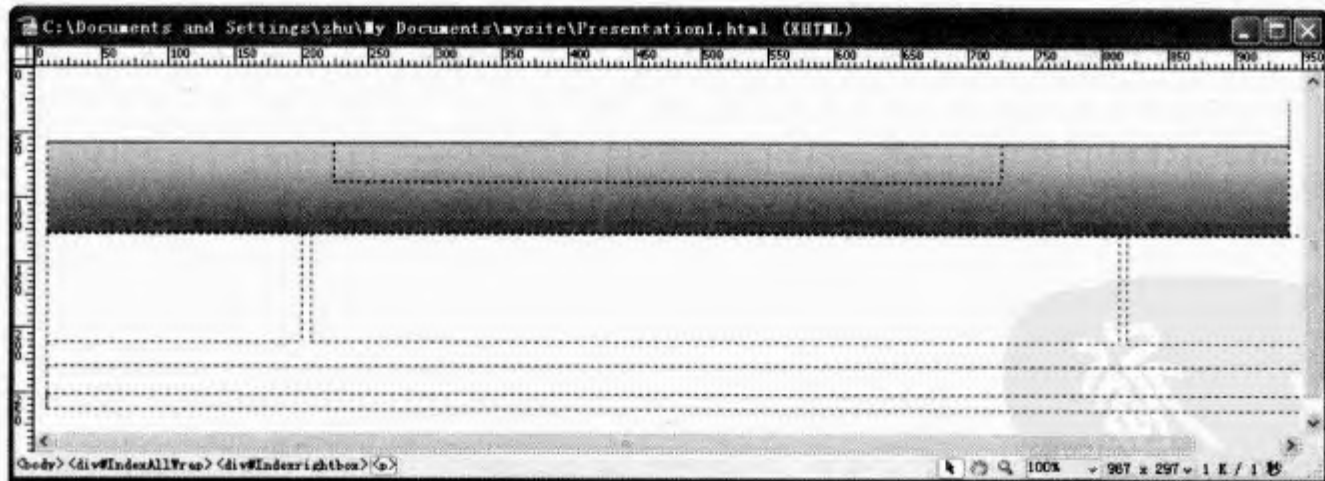


图 12.9 网店结构布局效果图

12.2.2 设计滑动门菜单

使用 CSS 完成结构布局之后，也就是说使用 CSS 撑起网页的基本结构之后，就该逐个模块地进行设计了。下面我们就本页面中几个典型的模块设计进行说明。

我们曾经在第 5 章中详细讲解了滑动门菜单的设计技巧，在本示例中也用到了滑动门菜单

(如图 12.10 所示)。



图 12.10 滑动门菜单效果图

在这个滑动门菜单中所需要的背景图像如图 12.11 所示。



图 12.11 滑动门所需要的背景图像

由于滑动门涉及到背景图像的处理，因此设计的第 1 步先定义 a 元素的宽度和高度，同时定义其他辅助样式：

```
#menubar .menu a { /* 定义超链接的默认样式 */
    text-align:center;           /* 文本居中显示 */
    float:left;                 /* 向左浮动，实现并列显示 */
    display:block;              /* 块状显示 */
    color:#0328C1;              /* 字体颜色 */
    font-size:12px;             /* 字体大小 */
    line-height:33px;           /* 定义行高 */
    width:60px;                 /* 固定菜单项的宽度 */
    text-decoration: none;      /* 清除下划线 */
}
#menubar .menu a:hover { /* 定义鼠标经过时的样式 */
    color:#c60;                /* 字体颜色 */
    text-decoration: underline; /* 显示下划线 */
}
```

当把菜单项 (a 元素) 大小固定之后，就可以来设计滑动门了。这里我们定义 3 个滑动门的类：

```
#menubar .menuone { /* 滑动门类 1 */
    background:url(menuone.gif); /* 滑动的背景图像 */
    padding-left:10px;          /* 左侧内边距 */
}
#menubar .menutwo { /* 滑动门类 2 */
    background:url(menutwo.gif); /* 滑动的背景图像 */
    padding-left:20px;          /* 左侧内边距 */
}
#menubar .menutwor { /* 滑动门类 3 */
    background:url(menutwor.gif) no-repeat; /* 滑动的背景图像 */
    padding-left:15px;          /* 左侧内边距 */
}
```

看来 3 个类在设计上基本相似，主要区别在于背景图像不同。为什么要定义左侧内边距呢？这主要是控制菜单中的文本不要贴近菜单的边沿显示，显示滑动门背景图像的边沿。

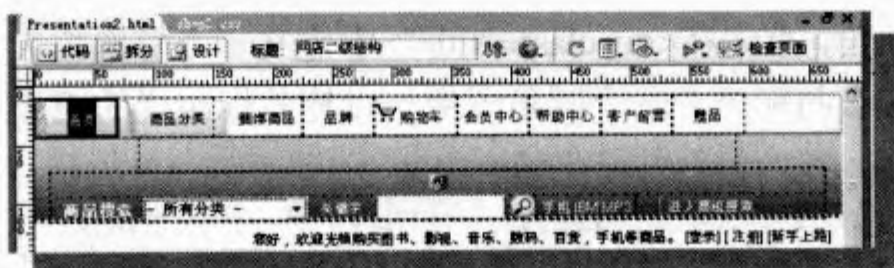


图 12.12 滑动门设计示意图

如果要设计更多的导航菜单，可以调用下面这个滑动门背景图像（如图 12.13 所示），然后再利用图 12.11 所示的第 3 幅背景图像来封闭右侧边缘即可。



图 12.13 增加设计的滑动门背景图

12.2.3 设计圆角区域

圆角问题也是网页设计中经常面临的问题，我们曾经也在第 4 章中详细讲解了圆角的设计技巧。下面我们对于这个再做简单介绍。

本示例中共设计了两种类型的圆角区域：一种是 4 个顶角都为圆角的区域（如图 12.14 所示），另一种是 2 个顶角为圆角的区域（如图 12.15 所示）。

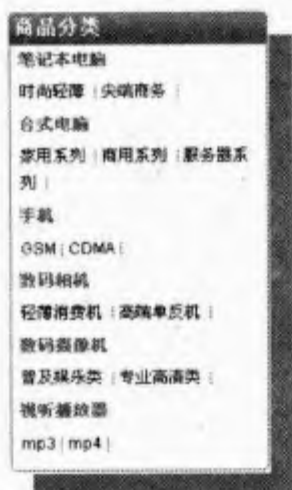


图 12.14 四角都为圆角的区域



图 12.15 两个顶角为圆角的区域

本示例的圆角区域借助修饰性辅助标签来完成。在 HTML 构建构建时我们也曾经讲解过这个问题。例如，针对图 12.14 所示的圆角区域，它的设计结构如下：

```
<div class="StyleCategoryIndex">
  <div class="title">
    <div class="right"></div>
    <div class="left"></div>
    <div class="cent">商品分类</div>
  </div>
  <div class="body">
    <!-- 内容区域 -->
  </div>
  <div class="lefttable-bottom">
    <div class="tdbg-left"></div>
    <div class="tdbg-right"></div>
  </div>
</div>
```

针对顶部的两个顶角，首先在<div class="title">中定义一个标题栏大小和背景图像，然后利用浮动布局的方法在左右两侧覆盖两个圆角图像，以掩盖标题栏的矩形形状。详细代码如下：

```
.StyleCategoryIndex .title {/* 标题栏包含框 */
    background:url(az-home_24.gif);          /* 背景图像 */
    width:190px;                             /* 固定标题栏宽度 */
    height:25px;                             /* 固定标题栏高度 */
}
.StyleCategoryIndex .title .left {/* 左上圆顶角 */
    float:left;                             /* 向左浮动 */
    width:4px;                              /* 固定圆角宽度 */
    height:25px;                             /* 固定圆角高度 */
    background:url(az-home_22.gif);          /* 设置圆角背景图像 */
}
.StyleCategoryIndex .title .right {/* 右上圆顶角 */
    float:right;                             /* 向右浮动 */
    width:5px;                              /* 固定圆角宽度 */
    height:25px;                             /* 固定圆角高度 */
    background:url(az-home_25.gif);          /* 设置圆角背景图像 */
}
```

然后定义圆角区域的底部两个圆角，设计与上面相同。详细代码如下：

```
.StyleCategoryIndex .lefttable-bottom {/* 底部包含框 */
    background: transparent url(az-h5.gif) repeat-x bottom; /* 底部背景图像 */
    height: 4px;                                           /* 固定宽度 */
    font-size:4px;                                         /* 字体大小（相当于定义高度） */
}
.StyleCategoryIndex .lefttable-bottom .tdbg-left {/* 左下圆顶角 */
    float:left;                                           /* 向左浮动 */
    background-image: url(az-home_72.gif);                /* 设置圆角背景图像 */
    background-repeat: no-repeat;                         /* 禁止重复平铺 */
    background-position: left top;                        /* 固定位置 */
    width:7px;                                             /* 固定宽度 */
    height:4px;                                            /* 固定高度 */
    font-size:1px;                                         /* 字体大小，清除行高对高度影响 */
}
.StyleCategoryIndex .lefttable-bottom .tdbg-right {/* 左侧内边距 */
    float:right;                                           /* 向右浮动 */
    background-image: url(az-home_74.gif);                /* 圆角背景图像 */
    background-repeat: no-repeat;                         /* 禁止重复平铺 */
    background-position: right top;                       /* 固定位置 */
    width:7px;                                             /* 固定宽度 */
    height:4px;                                            /* 固定高度 */
    font-size:1px;                                         /* 字体大小 */
}
```

图 12.15 所示的顶部圆角区域设计与图 12.14 所示的效果相同，这里不再详细讲解。

何为标准，何谓设计？

自从《网站重构——应用 Web 标准进行设计》一书上市，国内就开始了一股网站重构的热潮。这期间第一个吃螃蟹的是闪客帝国网站，不过闪客帝国在吃完螃蟹，并与大家初享喜悦和经验之后就沉寂下去了。当然，执著的设计师们不会因为闪客帝国的沉寂而放弃对 Web 标准的追求。是的，任何前进的步伐都不是一帆风顺的，Web 标准设计也存在不少误解和迷途，不少国内设计师对于 Web 标准的理解仅仅停留在 DIV+CSS 这个层次上，实属遗憾，何至于如此呢？

其实这是由于部分设计人员对于 Web 标准的理解不到位。或许有朋友说，结构中都是用 DIV 元素来实施，然后再配合 CSS 技术进行美化，所以就美其名曰 DIV+CSS。当然也可能是设计师为了方便，在交流中这样简单称呼吧，正如同事之间相互小王、小李叫得随意。但愿如此，不过它确实能够迷惑或者误导那些一尘不染的初学者，遗憾呀，无奈之中不免生起对初学者的同情。例如，很多初学者就误认为 Web 标准就是将 Table 布局换一下，换成 DIV 就是了，其实不然。

Web 标准制作的最终目的是什么呢？

也许明白了 Web 标准设计的最终目的，那么我们就清楚了上面提到的 DIV+CSS 这个术语为何会容易误导他人。你可能知道，XHTML 中定义了很多标签，每个标签都有特定的语义。我们是不是应该让每个标签都能对号入座呢，例如，ul 是无序列表、button 是按钮、a 是锚点等，了解了每个标签的语义后，你是否还会去乱用 DIV 呢？

一个合理使用标签来设计的页面结构，在无样式的情况下，你将会看到是一篇 Word 文档，标题就是标题，图片就是图片。也正是因为如此，才有 CSS Naked Day（CSS 裸奔节，4 月 9 日）的出现，为的就是大家来比比谁的结构写得好。所以希望读者在重构页面之前，先把你们的思维重构一遍，别让以前 Table 布局思维困扰你，尽量发挥自己的想象力。其实 Web 标准设计，在一个良好的结构也是如此，一切皆有可能（李宁的广告语），想怎么变都行（当然有些特殊情况需要 JavaScript 配合）。

重构的到来，也引出了“前端开发”以及“网站重构”这两个职位，“前端开发”比较侧重于表现的效果，“网站重构”则比较注重于页面的结构以及 CSS 技术实施。如果能熟练掌握 XHTML+CSS 二加一技术，那么“网站重构”这个职位就在向你招手了，而对于 XHTML 以及 CSS 的掌握的前提下，还要注意各个浏览器之间的区别。现在的浏览器不兼容性问题很严重，一般专业重构人员的工作机上会安装至少六个类型或版本的浏览器，甚至有更多的，目前国内浏览器兼容趋势主要集中在 IE6、IE7、FF2、FF3 等主流类型浏览器和版本。

有针对性的去学习，去研究探讨问题，相信你的未来不是梦。

林小志

[G e n e r a l I n f o r m a t i o n]

书名=精通 CSS 网页布局

作者=朱印宏，林小志编著

页码=421

I S B N = 4 2 1

S S 号 = 1 2 1 9 6 9 9 6

d x N u m b e r = 0 0 0 0 0 6 6 8 4 4 3 7

出版时间=2009.02

出版社=该引擎未能查询到

定价：45.00（含光盘）

试读地址=<http://book.szdnnet.org.cn/bookDetail.jsp?dxNumber=000006684437&d=538ECA4490F1241F3EA92F8C2757D77C&fenlei=1817&sw=css>

全文地址=<http://img9.5read.com/image/ss2jpg.dll?did=b44&pid=B6639A8956D88EEDC75F3390F30AEC1468EDC3E6E29BA89E8378276EA30F16D7AF9F9B4B62CCC69F71326E187B8F873A1DD38F11003ACC0B859C2034155A60E582EED475247F98CDA39820CA94A4054BC1FFC99F30299AD1698A4992ED21431A9E951A6605FC59D197CD2B387780E761C6E4&jid=/>