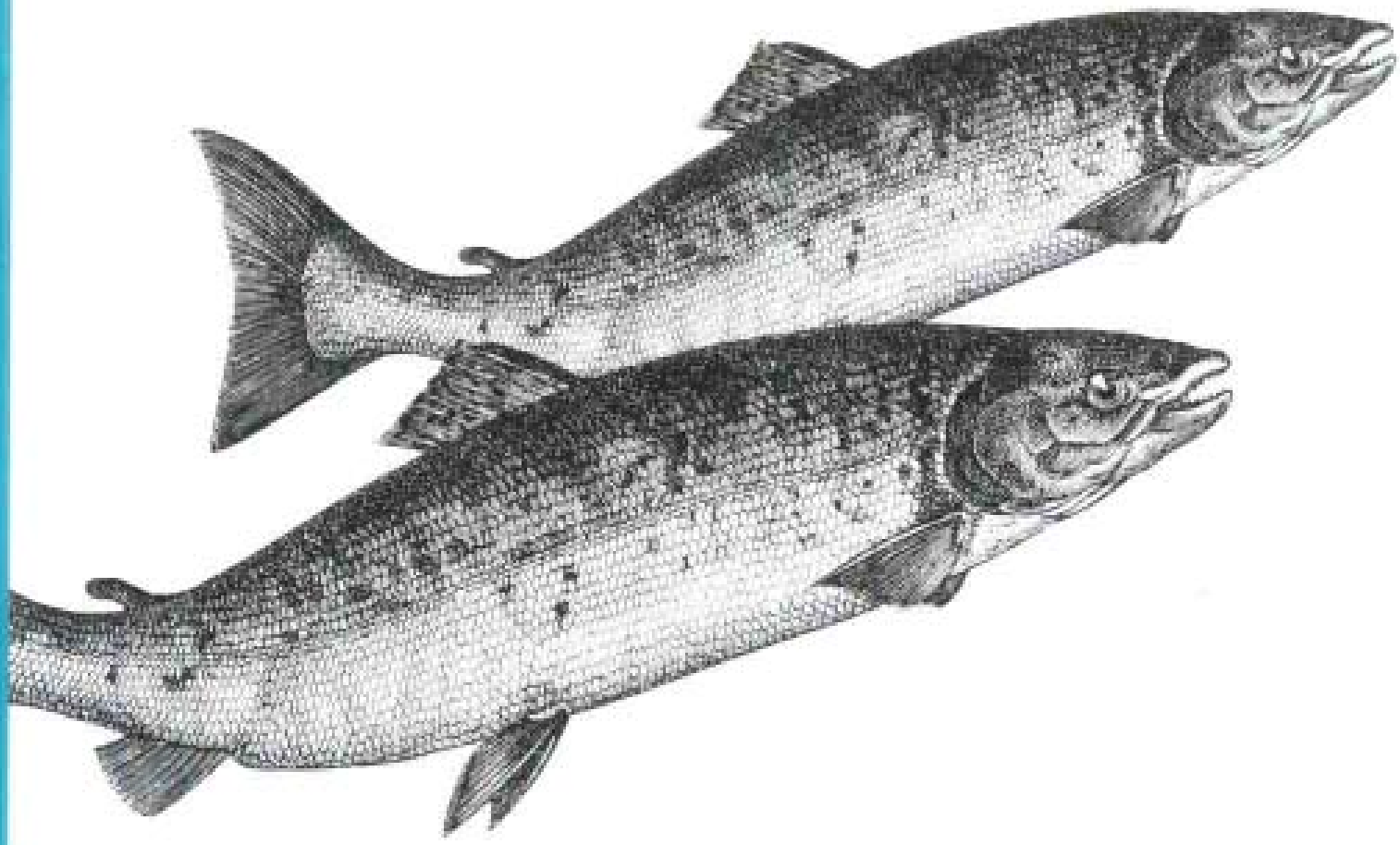
Cascading Style Sheets: The Definitive Guide

CSS权威指南



O'REILLY®

中国电力出版社

Eric A. Meyer 著

许勇 齐宁 译

CSS 权威指南

Eric A. Meyer 著

许勇 齐宁 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

中国电力出版社

图书在版编目（CIP）数据

CSS 权威指南 /（美）迈耶（Meyer, E.）编著；许勇，齐宁译．-北京：中国电力出版社，2001

书名原文：Cascading Style Sheets: The Definitive Guide

ISBN 7-5083-0560-4

Ⅰ.C… Ⅱ.①迈… ②许… ③齐… Ⅲ.主页制作－软件工具，CSS －指南

Ⅳ.TP393.092-62

中国版本图书馆 CIP 数据核字（2001）第 12491 号

北京市版权局著作权合同登记

图字：01-2001-2084 号

书　　名 / CSS 权威指南

书　　号 / ISBN 7-5083-0560-4

责任编辑 / 关敏

封面设计 / Ellie Volckhausen，张健

出版发行 / 中国电力出版社（www.infopower.com.cn）

地　　址 / 北京三里河路 6 号（邮政编码 100044）

经　　销 / 全国新华书店

印　　刷 / 北京市地矿印刷厂

开　　本 / 787 毫米 × 1092 毫米　16 开本　30.5 印张　450 千字

版　　次 / 2001 年 5 月第一版　2001 年 5 月第一次印刷

印　　数 / 0001-5000 册

定　　价 / 55.00 元（册）

译者序

自制定和发布以来，CSS规范便颇受世人的青睐。特别是对于专业网页设计者来说，CSS无疑是给他们的设计带来了新的希望。传统的HTML语言只是一种简单的结构标记语言，还不能够很好地控制网页的外观。而CSS的引入，正好弥补了这一缺陷。运用CSS，不仅能够设计出精美的网页效果，而且还能提高网页的可访问性、可维护性，从而为网页设计者节省大量的时间和精力。

本书的特点在于，它是第一本将CSS同当前浏览器的支持情况结合起来进行描述的读物，而不仅仅是讲述CSS理论上应该如何工作。它向Web创作者和编辑人员提供了高效使用CSS的全面指南。整本书以CSS的属性为主线，理论加实践，深入浅出地讲解了CSS的强大功能及最新特性，并配合适当的代码和警告提示，对某些重点难点或容易被忽略的地方加以强调和说明。而且在最后的附录部分，还列出了各种资源信息以方便读者查阅。当然在进行本书的学习之前，最好是对HTML有所了解，这样会更容易上手。

原书的作者Eric A. Meyer是万维网联合会（W3C）CSS&FP工作组的特邀专家和成员之一，他不仅对CSS规范相当熟悉，而且还亲自编写代码进行测试，积累了大量的实践经验。同时他还广泛听取了其他一些资深CSS专家的建议，从而使得本书的内容更全面，更有针对性。本书不仅可以作为专业网页制作人员和文档编辑人员的参考用书，而且可以当作一本很好的网页制作初学者的教科书。

本书由邓劲生组织翻译，由许勇、齐宁等进行主要章节的翻译，由孙兆林统校全书，灯芯工作室的其他工作人员完成全书的录入、校对和审稿等工作。由于书中涉及的知识和技术范围较广，尽管我们为此付出了许多努力，但由于时间紧迫，加上译者的水平有限，翻译中的不妥和错误之处在所难免，殷切希望广大读者和同仁批评指教。

译者

2001年3月

目录

前言

正如从封面可以猜到的那样，本书的主题是层叠样式表（Cascading Style Sheets，CSS）。CSS有两个“标准”，即CSS1和CSS2。它们之间的区别是：CSS2包含CSS1，而且还加入了很多其他的内容。本书涵盖了所有CSS1的内容，以及CSS2的一部分，即CSS定位（positioning）。而CSS2的其余部分则未包括在内，因为在写本书时，大部分CSS2的标准都还没有实现，所以与其包含太多的理论叙述，还不如将精力集中在当前实用的东西上。

对于网页设计者和文档作者来说，要想获得精美的页面风格、高的可访问性，并尽量节省时间和精力，那么本书是很适用的。在开始本书的学习之前，所有需要了解的相关知识就是HTML 4.0。当然，对HTML越熟悉，学习本书就会越得心应手。至于其他的相关知识，只需了解一点点就可以了。

请记住，对于网络标准和相关书籍而言，前者是在不停地发展变化的，然而后者则相对固定（至少在下一个版本还未出现之前是不变的）。就拿HTML和CSS来说，有很多变化正以文字的形式记录下来。例如，最近万维网联盟（W3C）推出的XHTML 1.0规范，就是万维网发展的一个重要的里程碑。对于CSS来说，可能还会出现更多的标准以扩展其样式化文档的能力；一些主要的网页浏览器也在向完全支持CSS1靠近、而且健壮的CSS2实现也初见端倪。所以，对于网页设计者来说，这是个令人兴奋的时刻，而且现在学习CSS也会为将来的工作打下一个很好的基础。

排版约定

本书使用下列排版约定:

等宽字体（Constant width）

用于代码示例，HTML标签和CSS元素。

等宽斜体（*Constant width*）

用于文本中的可替换元素。

斜体（*Italic*）

用于引入新的术语、文件名及路径名。

属性约定

在本书中，对于每个给定的CSS属性都用一个框将其取值或者相关的约定加以细化和说明。这些说明都已按照CSS规范逐字逐句重新编排过，但还是有必要对某些语法元素进行相应的解释。

每个属性所允许的取值都用下面的语法加以列举:

允许值:[<长度> | thick | thin]{1,4}
允许值:[<系列名称> ,]* <系列名称>
允许值: <url>? <颜色> [/ <颜色>]?
允许值: <url> || <颜色>

任何位于符号“<”和“>”中的文字都给定了一种类型的值，或者是对另一属性的引用。例如，font属性可以接受任何属于font-family属性的值，这是通过取值“<字体系列>”来表示的。任何以固定宽度文本出现的文字都是关键字，它必须按照实际的形式出现，不能引用。正斜杠（/）和逗号（,）也不能引用。

几个关键字排在一起意味着它们都必须同时出现——而且按照给定的顺序。例如，“help me”意味着这个属性必须按这样的前后顺序使用这些关键字。

如果一条单竖线（X | Y）分隔了两个选项，则它们中之一必须出现。双竖线（X || Y）意味着X或者Y，或者两者都出现，但它们可以按任何顺序出现。中括号[...]用于分组。分组优先于双竖线，而双竖线又优先于单竖线。这样，“V W | X || Y Z”就等同于“[V W] | [X || [Y Z]]”。

每个文字或括号必须紧跟以下修饰符中的一个：

- 星号（*）表示前面的值或者分组被重复零次或多次。这样，budket*表示文字“bucket”可以使用任意次，包括零次。对于被使用的次数上限没有明确的规定。
- 加号（+）表示前面的值或者分组被重复一次或多次。这样，mop+就表示文字“mop”至少必须使用一次，也隐含可以使用多次。
- 问号（?）表示前面的值或者分组是可选的。例如，[pine tree]?表示文字“pine tree”可以不使用（也可以使用，但它们必须按正确的顺序出现）。
- 大括号中的一对数字（{M,N}）表示前面的值或分组重复至少M次，最多N次。例如，ha{1,3}表示可以有一个，两个或者三个文字“ha”出现。

下面是一些例子：

```
give || me || liberty
```

至少必须使用三者之一，而它们可以按任何次序出现。例如，give liberty，give me，liberty me give和give me liberty都是有效的。

```
[ I | am ] ? the || walrus
```

文字I或者am可以使用，使用哪一个是可选的。而且，the或者walrus，或者它们一起都可以跟在后面。这样，可以组合成I the walrus，am walras the，am the，I walrus，walrus the等等。

```
koo+ka-choo
```

一个或者多个koo的后面必须跟一个ka-choo。当然，koo koo ka-choo，koo koo koo ka-choo和koo ka-choo都是合法的。koo数量可以是无限的，但在实现时总是有限的。

I really{1,4}* [love | hate][Microsoft | Netscape]

这是网页设计者发表意见的通用表达器。可以翻译成I love Netscape, I really love Microsoft，以及相似的一些句子。可以使用零到四个really。但必须在love和hate中选择其一，在本例中选择了love。

[[Alpha || Baker || Cray],]{2,3} and Delphi

这是个很长很复杂的表达式。可能的结果是Alpha, Cray, and Delphi。另一个可以是Alpha Baker, Cray Alpha, Baker Cray Alpha, and Delphi。逗号因其位置的特殊而放在中括号内。

建议与评论

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们:

美国:

O'Reilly & Associates,Inc.
101 Morris Street
Sebastopol, CA 95472

中国:

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

询问技术问题或对本书的评论，请发电子邮件到:

info@mail.oreilly.com.cn

可以访问本书的网站，这里有书中的实例、勘误表和对将来版本的计划:

http://www.oreilly.com/catalog/css/

最后，你可以在 WWW 上找到我们:

http://www.oreilly.com
http://www.oreilly.com.cn

致谢

写这本书是出于一个目标：就是使CSS易学、易懂、易于使用。当然，我们为此做了很多的工作试图将复杂的算法简要地翻译成易于理解的语言并不像听起来那么容易。

在编写本书的过程中，我发现自己好像是在CSS中不引人注意的角落里独自爬行，创建测试程序，苦思冥想所有的问题，而且还要兼顾网页浏览器的一些限制。在这期间，我遇到了很多热心的人，我应该感谢他们。

当然，首先是Opera Software的Håkon wium Lie和W3C的Bert Bos，由于他们努力创建了CSS，而且回答了我关于*WWW*样式邮件列表的很多问题，甚至是一些很愚蠢的问题。我同样要感谢W3C的Chris Lilley，他鼓励我将网页变得更加样式化。他对我先前的努力所给于的赞扬，对于我来说真是一副强心剂，而且也正是他使我加入W3C的CSS&FP工作组成为可能。

Tim O'Reilly给了我向专业出版物迈进的机会，为此我感激不尽。O'Reilly的编辑，Richard Koman，在我写作的过程中非常耐心，或许我该送去更多的稿件以表示我对他的感激。Tara McGoldrick，同样在O'Reilly工作，他在图形整理、语言沟通方面给了我许多帮助，使我的生活变得更容易。Songline工作室的Dale Dougherty给了我最初在专业写作方面的突破，而且Chuck Toporek（现在在O'Reilly工作）是我的第一个长期在《Web Review》的编辑，他认真负责地为我安排所有文章的提交日程。我的朋友、同事Peter Murray，非常大方地献出他的宝贵时间及精力帮助我创作其中的一个案例研究，以及Ron Ryan，他是我所知道的最棒的管理人员，基于此他获得了一颗金星和很多好评。

在过去的几年，来自Usenet组comp.infosystems.www.authoring.stylesheets的一群热心人既回答了我许多的问题又提供了反馈信息，给了我巨大的帮助。在这个组里，Todd Fahrner，Alan Flavell，Anthony Boyd和Jan Roland Eriksson为我提供了许多有用的线索。特别值得一提的是，Sue Sims勤勤恳恳地帮助（有时是保

护）她从未遇见过的人 —— 那就是我。Ian Hickson 和 David Baro 作为 CSS 方面的专家，奉献出了他们宝贵的精力以解释规范，寻找浏览器的 bug 和传播 CSS 带来的好处，同样要感谢他们。他们的评论及洞察力给本书的写作带来了极大的帮助。

当然，衷心的感谢要献给CSS的勇士们：Todd Fahrner，Liam Quin，David Baron，Ian Hickson，Sue Sims，Jan Roland Eriksson，John Alsopp和Braden McDaniel。是的，许多名字都有重复。毕竟，这只是一个小小的团体，但我还能有什么可说的呢？再一次感谢他们的大力支持。

请注意，他们中不应该有任何人会因本书中的错误而受到责备。相反，由于他们为我纠正了许多的错误而应受到我的赞许。

当然，为 Tim Berners-Lee 发明了 WWW 而感谢他，对我来说是合适的。我虽不认识他，但我觉得凡是和万维网相关的书籍的作者都有必要感谢 Tim 的巨大贡献。我认为这或许可以成为一项国际上的惯例；不妨就叫它Berners Convention。

最后，我个人还要致谢：Michelle，他是我患难与共的朋友；Randy 总是很乐意充当我的老师和密友；Steve 在大学及以后的几年里给了我很多的帮助；Dave 在这些年里给我带来了欢笑，特别是当我最需要它的时候，在我头脑发热时，又给了我许多让我清醒的言语；还有 Tina，在我想放弃的时候帮助我站了起来。

同时还要感谢我的妻子，Kathryn，她给了我无尽的支持，特别在紧急关头，她对我和我的能力的信任和肯定；还有我的父母，Art 和 Carol，姐姐，Julie，他们总是一如既往地给我支持。

再次感谢他们。

— Eric A. Meyer

2000 年 2 月 23 日

第一章

HTML与CSS

从很多方面来说，层叠样式表（Cascading Style Sheets，CSS）规范代表了万维网历史上一个独特的发展阶段。从最大限度地使结构文档样式化的根本能力上来说，CSS算是前进了一步，也算是后退了一步——但这是很好的一步后退，而且是必需的一步。要想知道其原因，首先有必要理解网络为何需要CSS这样的东西，以及CSS是怎样使网站更适合于网页制作者和网上冲浪者的。

网络的由盛到衰

追溯到网络发展的早期（1990－1993），HTML还是一种很贫乏的微型语言。它几乎全由结构化元素组成，这些元素对于描述段落、超链接、列表及标题非常有用。但像表格、框架或者是我们认为在创建网页时很有必要的一些复杂的标记却没有包含在内，甚至连边儿都未沾上。通常认为，HTML是一种结构化标记语言，用于描述文档的不同部分。至于这些部分应该怎样去显示，HTML却很少去关注。所以这种语言并不关心文档的外观，它仅仅是一种纯粹的小的标记系统。

然后Mosaic出现了。

突然，对于几乎所有上网超过十分钟的人来说，万维网的力量是显而易见的。从一个文档跳到另一个文档，只需让鼠标指向一个特殊颜色的文本或者是图像，然

后再轻轻点击鼠标即可。甚至，文本和图像可以在一起显示，而且要创建网页所需的所有工具只是一个纯文本编辑器。这就显得非常自由、非常开放，而且网页的外观也非常漂亮。

于是网站开始到处萌生。个人杂志网站、大学网站、公司网站及其他各种各样的网站纷至沓来。然而，随着网站数量的增加，对产生新的网页效果的HTML标签的需求也相应增加了。网页制作者也就开始有了使用粗体或者斜体文本的需求。

当时，HTML并不具备处理这类需求的能力。读者可能需要强调一小部分文本，但你无法保证它一定变为斜体——或许粗体，甚至是不同颜色的正常字体，这依赖于用户的浏览器及其设置。但是没有任何措施能保障制作者所创作的就是读者所看到的。

由于种种压力，像<B>和<I>这样的标记元素开始引入到语言中。突然，一种结构化语言开始出现了。

乱作一团

几年后，我们同时也继承了这个过程所带来的内在缺陷。例如，HTML 3.2和HTML 4.0的大部分内容都专注于外观上的考虑。至于通过FONT元素来控制文本颜色和大小，对文档和表格应用背景颜色和图像，对表格单元内容的分隔和填充以及使用闪烁的文本字符，所有这些都成了最初要求“尽量控制！”所带来的产物。

想知道为什么这是一件坏事吗？不妨对某公司网站的网页中的标记进行观察。把那些大量的标记同有用的信息进行比较，可能会令人大吃一惊。更坏的是，对于大多数网站来说，表格和FONT标签几乎组成了整个标记网页，但它们没有表达任何真正的意义。而从结构的立场来看，这些网页仅比随机的字符串好那么一点点。

例如，假设有个页标题，制作者使用的是FONT标签而不是像H1这样的标题标签，如下：

```
<FONT SIZE="+3" FACE="Helvetica" COLOR="red">Page Title</FONT>
```

从结构的观点来说，FONT标签毫无意义，这使得文档几乎无用。比如，FONT标签会给支持语音的浏览器带来什么好处呢？如果制作者使用标题标签而非FONT标签，那么语音浏览器还可以用某种语音格式来读取标题文本。但是对于FONT标签，浏览器却没有办法区分不同的文本。

制作者为什么要这样使用呢？因为他们想让读者看到的网页跟他们设计的一样。使用结构化的HTML标记将会放弃对网页外观的许多控制，而且理所当然地不允许过去几年中曾流行的那种密密麻麻的网页设计。

那么问题出在哪里呢？考虑下面的因素：

- 非结构化的网页使内容索引变得很难。一个真正强大的搜索引擎应该允许用户按页面标题搜索，或者是页面内的子标题，或者仅仅是段落文本，或仅仅是那些标记为重要的段落。然而，为了做到这一点，网页的内容必须包含在某些结构化标记内——这正是大多数网页缺乏的那种标记。
- 结构的缺乏降低了可访问性。设想一下，盲人主要通过语音浏览器来浏览网页。那他将如何做选择呢：是允许浏览器阅读章节标题，进而对感兴趣的章节进行选择呢？还是那种缺乏结构的，强迫阅读所有内容，而且对于哪些是标题，哪些是段落，哪些东西是重要的都毫无指示的网页呢？
- 高级的页面表现力只可能通过某种文档结构来达到。设想某个页面只显示了章节标题，而每个章节旁都带有一个箭头，那么用户就能够决定哪些章节适合他，从而点击它，打开相应章节的文本。
- 结构化标记易于维护。设想为了在某个浏览器中找寻弄乱网页的某个小错误，会花掉多少时间去搜索人家的或自己的HTML页？为了一个带白色超链的边框，会花多少时间去书写嵌套表格和FONT标签呢？为了在标题和其后的文本之间得到一个理想的间隔距离，又会插入多少的换行标签呢？通过使用结构化标记，就能使代码变得清晰，而很容易找到想要的东西。

诚然，全结构化的文档的确有些平淡。这都是基于这样一个事实：在20世纪末，结构化标记还不能动摇市场对时下流行的HTML页面的钟爱，真正需要的是寻找一种可以将结构化标记同丰富的页面表现相结合的方式。

这一观念也并不新鲜了。在过去的几十年里已经提出和创建过很多样式表技术。这些都是企图在不同的行业和协会内使用不同的结构化标记语言。这些语言也已被测试过、使用过，而且被证实在需要用结构来表现的环境中是适用的。但是还没有任何样式表能立即就适用于HTML。所以，为解决这一问题还有一些工作要做。

CSS是拯救之方

当然，用于表示的标记元素会影响到HTML的结构，而W3C也没有忽略这个问题。他们在早些时候就意识到这样的情形不能持久，而且急需一个好的解决方案。于是在1995年，W3C开始着手制定一个名为CSS的规范。到1996年，这一规范已同HTML本身处于同一地位，成为了一个正式的规范。

那么CSS能提供给我们什么呢？就本书而言，它提供给我们两个标准。第一个标准是层叠样式表Level 1（CSS1），它是1996年W3C的一个正式的规范。不久以后，W3C的层叠样式表和格式化属性（CSS&FP）工作组开始着手一个更高级的规范。1998年，当层叠样式表Level 2（CSS2）被当作一个正式的规范时，他们的工作便告一段落了。CSS2构建在CSS1之上，并且在没有大的变化的情况下，扩展了一些早期的工作。

将来CSS或许还会有更高的版本，但在那之前，还是让我们来看看已有的CSS吧。

丰富的样式

首先，CSS允许更为丰富的文档外观，即使从表示程度的高度上来看，它也比HTML更为丰富。CSS拥有设置文本颜色和背景颜色的能力；它允许为任何元素创建边框并调整边框与文本之间的距离；它允许改变文本的大小写方式、修饰方式（如加下划线）、文本字符间隔，甚至可以隐藏文本以及其他许多效果。

例如，网页上的一级标题（也是主题），通常就是本页的标题。正确的标记为：

```
<H1>Leaping Above The Water</H1>
```

现在假设要让此标题为暗红色，使用某种字体，斜体，带下划线，而且还带黄色的背景。如用HTML来做，就得在表格中放置H1标签，而且还要用到其他一大堆像FONT和U之类的标签。但用CSS，所需的仅仅是一条规则：

```
H1 {color: maroon; font: italic 1em Times, serif; text-decoration: underline;
   background: yellow;}
```

就这么简单。总之，能用HTML制作的都可以用CSS来制作。没有必要将自己约束在HTML的圈子里。例如：

```
H1 {color: maroon; font: italic 1em Times, serif; text-decoration: underline;
   background: yellow url(titlebg.png) repeat-x;
   border: 1px solid red; margin-bottom: 0; padding: 5px;}
```

这样一条规则就允许我们在H1的背景里按水平方向重复放置一个图像，为H1加上一个边框，它与文本的间隔至少为五个像素，而且还去掉了元素的下边界（空白的空间）。而HTML不能做如此细化的工作——这也算是对CSS的一个尝试了。

易于使用

如果CSS还不能令人信服，那么这应该可以了：样式表能彻底减轻网页设计者的工作负担。

样式表能够在恰当的地方集中一批命令，以实现某种可视效果，而不是将它们到处分散在整个文档中。比如，假设要让文档中的所有标题都显示为紫色。（我也不知道为什么要这样做，只是假设而已。）如果使用HTML就需在每个标题标签里放置一个FONT标签，如下：

```
<H2><FONT COLOR="purple">This is purple!</FONT></H2>
```

而且还必需对每个二级标题进行同样的设置。如果在文档中有40个这样的标题，那么我们就必须插入40个FONT标签，因为每个标题都需要一个！这么一个小小的效果所需的工作量却非常大。

不妨假设上面的工作也已做完，而且那些所有的FONT标签也都加上了。现在心情

应该很舒畅了——然而可能又决定（或许是老板决定）将标题变为暗绿色，而不是紫色。那么又得回到前面，修改每一个FONT标签。当然，只要此标题是文档中唯一的紫色文本，我们就可以用查找替换的编辑功能。但如果文档中还有另外的紫色FONT标签，就不能用查找替换操作了，因为查找替换会影响到其他元素。

运用单独的一条规则会更好：

```
H2 {color: purple;}
```

不仅键入快，而且易于改变。如果想由紫色变为暗绿色，只需对这条规则加以改变就可以了。

再回到前面的例子：

```
H1 {color: maroon; font: italic 1em Times, serif; text-decoration: underline;
   background: yellow;}
```

它的书写看起来比使用HTML更糟糕。但考虑这样一种情形：当某页需要12个同H1一样的H2元素时，如果使用HTML，那么这12个H2元素需要多少标记呢？非常多。另一方面，如使用CSS，则只需照下面这样做：

```
H1, H2 {color: maroon; font: italic 1em Times, serif; text-decoration: underline;
   background: yellow;}
```

现在这种样式同时应用于H1和H2元素，但只需多输入3个字符。

如果想改变H1和H2元素的显示方式，CSS带来的好处更是不言而喻了。考虑将所有H1和12个H2元素的HTML标记作改变所花的时间，和将前面的样式变为下面的样式所花的时间，并且进行比较：

```
H1, H2 {color: navy; font: bold 1em Helvetica, sans-serif;
   text-decoration: underline overline; background: silver;}
```

如果这二种方法都用计时表来计时，我敢打赌经验丰富的CSS作者一定会轻而易举地胜出HTML的操作者。

另外，大多数CSS规则都可以集中起来放到文档中的某个位置。也可以将每条规则同具体的元素联系起来，从而应用到整个文档，但通常还是将所有的样式规则

放到一张样式表中更有效。这样，创建（或修改）整修文档的外观时只在样式表文件中修改就可以了。

在多页上应用样式

上面提到过，不仅可以将所有样式信息集中到页面的一个地方，而且还可以为其创建一个样式表文件，它可以应用于多个页面。这可以通过这样一个过程来做到：先将一个样式表存储起来，然后在其他页中再引入这个样式表文档。运用这一特性，可以很快创建一个网页风格一致的网站。而且当需要改变网站的网页风格时，只需对这样一个样式表文件作修改，其变化就可以影响到整个网站服务器 —— 自动影响！

考虑这样一个网站，所有的标题都是在白色背景上显示灰色文本。它们将从下面这样一个样式表中获取颜色：

```
H1, H2, H3, H4, H5, H6 {color: gray; background: white;}
```

现在假设此网站有700个网页，每个网页都使用这样一个样式表。但由于某种原因，决定将标题变为在灰色背景上显示白色。因此网站的管理员可以像下面这样编辑样式表：

```
H1, H2, H3, H4, H5, H6 {color: white; background: gray;}
```

然后将其存盘，改动就算完成了。这当然要比将每个标题逐一放到表格和FONT标签里要简单多了，更何况是700个页面，难道不是吗？

层叠

CSS的特性还不只这些！它还对冲突规则做了约定；这些约定指的就是层叠。例如，在前面的例子中，将一个样式表引入到整个网站页面中。现在要另外再插入一组网页，这些网页共享了大部分样式，但还是有只适用于它们自己的特殊的样式规则。可以创建另一个样式表，将其引入到这些网页中，同时附加上已经存在的样式表，或者仅仅将这些特殊的样式放置到需要它们的网页里。

例如，有一页同其他700页都不同，它不是在灰色背景上显示白色，而是在深蓝色背景上显示黄色。在这个网页里，就可以插入这样一条规则：

```
H1, H2, H3, H4, H5, H6 {color: yellow; background: blue;}
```

而由于层叠的特性，这条规则将覆盖原来引入的“白-灰”标题。所以，理解了层叠的特性并很好地运用它们，就可以创作出许多高质量的样式表，从而赋予网页专业的、易于修改的外观样式。

这种能力并不仅限于制作者。网络冲浪者（或读者），也可以在某些浏览器上创建自定义的样式表（称为读者样式表，很奇妙吧），这些样式表可以层叠制作者的样式，和用浏览器层叠一样。这样，一位色盲的读者就可以创建一种样式，使超链接更为醒目：

```
A:link {color: white; background: black;}
```

读者样式表几乎可以包含任何东西：如果用户弱视，则可以使文本足够大以便于阅读；去除图像的规则以使阅读和浏览的速度更快；甚至可以将用户最喜欢的图片放到每个文档的背景上。（当然不推荐这样做，但这是可能的。）这样，读者不必去掉所有制作者的样式就能定制他们自己的网页风格。

由于其样式表的可引入、可层叠以及多样化的效果，使得CSS成为制作者和读者们非常有用的工具。

压缩文件大小

除了可视化及赋予制作者和读者的特殊能力之外，CSS还可以使文档足够小，以缩短下载的时间。那么它是怎么做的呢？正如我们前面所提到的，很多网页都使用表格和FONT标签以获得精美的视觉效果。不幸的是，这些方法产生了大量的HTML标记，从而使文件变大。通过将可视的样式信息分组为主要的几部分，然后用尽量压缩的语法去表示这些规则，就可以去掉FONT标签和其他一些常用标签。这样，CSS就可以保证下载时间缩短，从而令读者更满意。

为将来做准备

正如前面所提到的，HTML是一种结构化语言，而CSS是对它的一个补充：一种样式化的语言。正基于此，W3C这个讨论和批准网络标准的主要机构正开始着手去掉HTML中的样式标签。其原因就是样式表能够用来创建HTML标签所提供的任何效果，所以HTML就不再需要这些标签了。

在写本书时，HTML 4.0规范中有许多标签遭到了反对；也就是说它们会被分阶段地从该语言中删除，最终，它们将被标记为作废，即浏览器不再需要对它们提供支持。<FONT>、<BASEFONT>、<U>、<STRIKE>、<S>和<CENTER>就属于其中之列。随着样式表的出现，这些HTML标签会慢慢消失。

而且，HTML很可能会逐渐被可扩展标记语言（XML）所取代。XML比HTML更复杂，但也更强大、更灵活。尽管如此，XML自身并不提供任何方式来声明像<I>或<CENTER>这样的样式标签。取而代之的是，XML文档很可能会依赖样式表来决定文档的外观。然而XML使用的样式表可能不是CSS，但会是对CSS的某种继承，必定与CSS非常类似。所以现在学习CSS也会给制作者带来很多好处，特别是当发展到基于XML的网络的时候。

CSS的局限

本书中也有一些CSS1没有涉及到的领域，当然这些内容就不包含在详细讨论的范围内；有一些内容会放到第十章“CSS2展望”里来讨论。但是，即使是一个覆盖所有CSS1和CSS2的完全版的CSS也不能满足世界上每一个网页设计者的需要。所以跨越CSS的一些边界是必要的。

受限的初始范围

一旦掌握了正确的学习方法后，CSS1规范就显得并不那么复杂了。它包含大约70个属性，而且其所有的内容可以在100页内打印出来。但它仍然不失为一个成熟、精致的引擎，只是网页设计的某些领域却从CSS1中删去了。

首先，CSS1几乎不对表格作任何处理。读者可能会想到为表格单元设置边界这个例子——而且网络浏览器也允许这么做——但边界并不是在任何环境下都能应用于表格单元。CSS2为此引入了一套新的属性和行为以处理表格，但在写本书的时候，很少有浏览器支持这样的属性和行为。

注意：从某种意义上来讲，从CSS1中删去表格代表了多数人的意见，因为他们认为表格不应该用于网页的布局。浮动的和位置固定的元素就能代替所有表格的工作，而且其功能会更强大。但这种提法是否正确，我们在此不作任何评论。

同样，CSS1不包含位置的概念。当然也可以对元素的位置稍加移动，但多数情况是使用负边界和浮动来实现的。从某种意义上来讲，其中的所有元素都是相互关联的。然而，CSS2有三个章节用于讲述可视化建模，其中就包含位置元素。

CSS1也不提供可下载的字体。这就导致了大量关于如何适应用户系统配置和有效的字体的讨论。CSS2引入了某些字体处理。即便这样，问题仍未得到解决，其主要原因是缺乏对字体格式的广泛支持。或许可缩放向量图形（Scalable Vector Graphics，SVG）能够解决部分甚至所有的问题，但要对此下一个定论是不太可能的。

最后，CSS1缺乏对媒体类型的支持。换句话说，CSS1主要是一种屏幕设备语言，只能用于将内容输出到计算机的显示器上。但它也有一些对于页面媒体的支持，像打印输出，但不多。（尽管如此，CSS1并不是一个像素级的控制机制。）为了克服这一限制，CSS2引入了媒体类型，它可以根据文档的显示媒体创建各自的样式表。CSS2也针对页面媒体和语音媒体特别地引入了一些属性和行为。

实现

不幸的是，使用CSS最主要的缺陷就是它在最初未能得到完全的实现。由于各种误传、误解、混乱和低质量的控制，致使最初企图支持CSS的浏览器做得一团糟。

最典型的浏览器就是Microsoft Internet Explorer 3.x 和 Netscape Navigator 4.x。它们都提出了对CSS的最初的支持，但这些浏览器都做得不完美，而且错误百出，甚至常常是适得其反，所以就更不用提对CSS2的支持了。这些实现十分拙劣，以

致于很难分辨它们是否真的支持CSS。而且，当试图处理某些样式时，有些致命的缺陷往往会使浏览器崩溃，甚至锁住整个系统。

发展到Internet Explorer 4.x 和 5.x 时，情况有了一些改善。尽管不能说是完美，但这些版本的浏览器至少是剔除了许多IE3中的错误，而且还增加了对先前未接受的CSS1 和CSS2 中的一些属性的支持。

另一方面，Opera 3.5 因提供对CSS 的良好支持而为人所知。由于它本身就是基于CSS1 的，所以这个浏览器对属性的支持非常好，只有一些小的错误。当发行3.6版时，几乎所有这些小错误都被修正了，尽管提供的支持还是未能超出CSS1的范围。而在3.5版以前，Opera根本就不支持CSS。

至于Netscape的产品，Navigator 4.7 与4.0版差不多，对CSS提供的支持也没有得到显著的改善，但不那么容易崩溃了。而Netscape对CSS良好支持的唯一希望就落在了他们的Gecko引擎上。到写本书的时候，最近的Gecko版本已非常优秀，而且实际上我们就是用它（连同Macintosh 上的Internet Explorer 4.5 和 5.0）来创建了本书中的许多图。

既然CSS不打算对文档的显示提供完全的控制，但总应该让网页的内容在任何浏览器上都能够正确显示，至少它不应该成为用户使用CSS的障碍。然而，如果有用户正在使用过时的浏览器（Explorer 3.x，或许 Navigator 4.x），也应该通知他们检查其浏览器的设置，以禁用样式表。这样，他们至少能阅读网页的内容，即使这些页面并不是我们希望他们看到的那种样式化的风格。

CSS 和 HTML

迄今为止，我们一直将注意力放在HTML 文档的结构上。实际上，这也是当今网络发展所要面临的一个问题：多数人都忽略了文档的内部结构，而这种内部结构和我们平时所见的结构不一样。所以，为了在网络上创建最酷的网页，我们通常扭曲，甚至忽略了网页也应该包含结构信息这一思想。

然而，这种结构却是联系HTML和CSS的纽带；如果没有这个结构，它们之间根本就不会存在任何关系。为了更好地理解这一点，让我们来看一个HTML文档实例，其标记如下，结果如图 1-1 所示：

```
<HTML>
<HEAD>
    <TITLE>Eric's World of Waffles</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="sheet1.css" TITLE="Default">
    <STYLE TYPE="text/css">
        @import url(sheet2.css);
        H1 {color: maroon;}
        BODY {background: yellow;}
        /* These are my styles! Yay! */
    </STYLE>
</HEAD>
<BODY>
    <H1>Waffles!</H1>
    <P STYLE="color: gray;">The most wonderful of all breakfast foods is
    the waffle-- a ridged and cratered slab of home-cooked, fluffy
    goodness...
    </P>
</BODY>
</HTML>
```

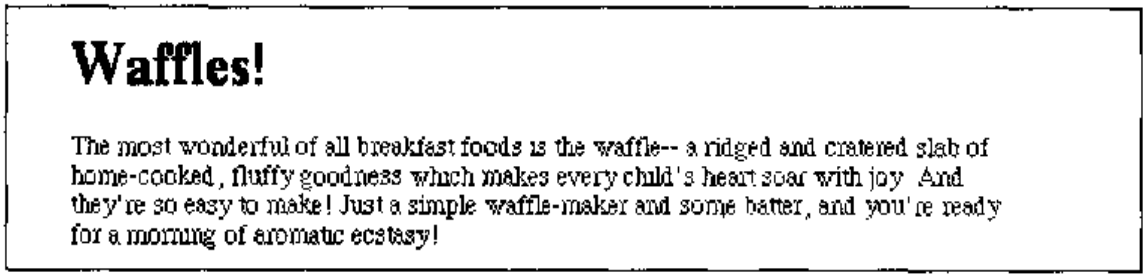


图 1-1 一个简单的文档

现在，让我们检查一下文档的每个组成部分。

LINK 标签

```
<LINK REL="stylesheet" TYPE="text/css" HREF="sheet1.css" TITLE="Default">
```

首先来看一下 LINK 标签的使用。LINK 标签很少引人注意，但它却是个非常有用的标签，而且它出现在 HTML 规范中已有多年了，有望发挥更好的作用。它最基本的用途是允许 HTML 制作者在一个文档中链接其他的文档。CSS1 用它来为 HTML 文档链接样式表；在图 1-2 中，名叫 *sheet1.css* 的样式表被链接到文档中。

虽然这些样式表并不是 HTML 文档的一部分，但仍然被当作外部样式表使用。它们对于 HTML 文档来说是外部的（如图）。

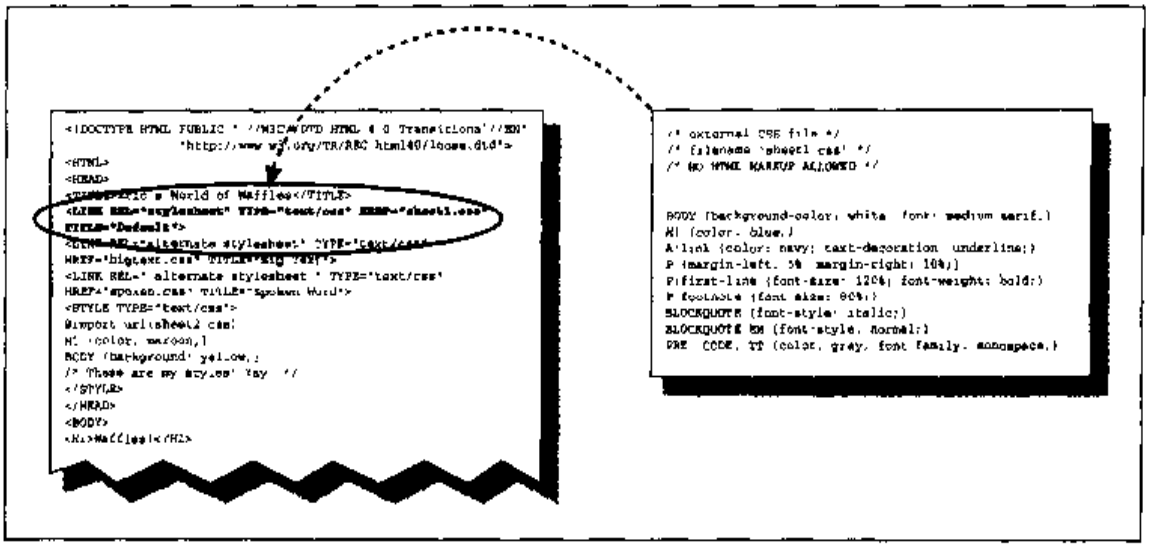


图 1-2　外部样式表怎样应用于文档

为了成功地载入一个外部样式表，LINK标签必须放在HEAD元素里，而不能放在任何其他元素里，即使是TITLE或STYLE这样的元素也不行。这样，网络浏览器就能定位和载入样式表，而且用它所包含的所有样式规则作用于HTML文档，就像图 1-2 显示的那样。

那么外部样式表的格式是怎样的呢？它只是一个简单的规则列表，就像在前面看到的那样，但在这种情形下，所有规则都存放在它们自己的文件里。请记住，HTML 或任何其他标记语言都不能包含在样式表里 —— 样式表里只能有样式规则。下面就是一个外部样式表的标记：

```
H1 {color: red;}
H2 {color: maroon; background: white;}
H3 {color: white; background: black; font: medium Helvetica;}
```

这就是所有的东西 —— 没有 STYLE标签，也没有任何 HTML 标签，只有平淡而简单的样式声明。这些规则存放在一个简易的文本文件里，而且通常给其一个扩展名 *.css*，如 *sheet1.css*。

文件的扩展名并不是必需的，但某些浏览器只能识别以 *.css* 为扩展名的样式表，即使网页的 LINK 元素里包含了正确的 text/CSS 类型。因此，一定要给样式表取个合适的名字。

LINK属性

对LINK标签而言，剩下的就是其属性和值了，而它们都很简单。REL是“关联”的意思，在这里就是指关联“样式表”。TYPE总是设置成text/css，这个值描述了使用LINK标签载入的数据类型。这样，浏览器就知道这是一个层叠样式表。实际上这也就决定了浏览器怎样处理所载入的数据。毕竟，将来或许还会有其他的样式语言。因此选用哪种语言就显得很重要了。

接下来是HREF属性。它的值表示样式表的URL。这个URL既可以是绝对的，也可以是相对的。当然，在我们的例子里URL是相对的。它也可以是http://www.style.org/sheet1.css这样的URL。

最后是TITLE属性。这个属性不常用，但将来可能变得很重要。为什么呢？当不只有一个LINK标签时它就显得重要了——而且很可能会有更多的标签。然而在这样的情况下，也只有那些REL为stylesheet的LINK标签会在文档的初始显示时被使用。这样，如果想用名为*basic.css*和*splash.css*来链接两个样式表，就应该这样去标记：

```
<LINK REL="stylesheet" TYPE="text/css" HREF="basic.css">
<LINK REL="stylesheet" TYPE="text/css" HREF="splash.css">
```

这会使浏览器载入两个样式表，然后组合它们的规则，再将其结果作用于文档(见图1-3)。在下一章中，我们可以清楚地看到样式表的组合，但现在暂且接受这样的组合。例如：

```
<LINK REL="stylesheet" TYPE="text/css" HREF="sheet-a.css">
<LINK REL="stylesheet" TYPE="text/css" HREF="sheet-b.css">

<P CLASS="a1">This paragraph will be gray only if styles from the
stylesheet 'sheet-a.css' are applied.</P>
<P CLASS="b1">This paragraph will be gray only if styles from the
stylesheet 'sheet-b.css' are applied.</P>
```

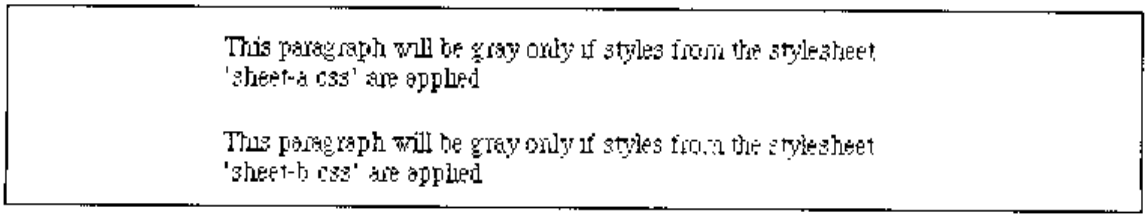


图1-3 组合链接的样式表

也可以定义可选择的样式表。它们是用一个alternate stylesheet的REL来标记的，而且只有当它们被读者选用后才能应用于文档。

可选择样式表

不幸的是，到写本书时，对许多浏览器来说，使用可选择的（alternate）样式表并不是一件容易的事。如果一个浏览器能够使用可选择的样式表，它会用TITLE属性的值来产生一系列的样式选项。因此可以像下面这样书写：

```
<LINK REL="stylesheet" TYPE="text/css"
 HREF="sheet1.css" TITLE="Default">
<LINK REL="alternate stylesheet" TYPE="text/css"
 HREF="bigtext.css" TITLE="Big Text">
<LINK REL=" alternate stylesheet " TYPE="text/css"
 HREF="spoken.css" TITLE="Spoken Word">
```

用户可以选择他们想要使用的样式，然后浏览器就从第一个样式（在本例中标号为“Default”）切换到读者所选用的样式上。图1-4显示了这种选择机制的一种方式。

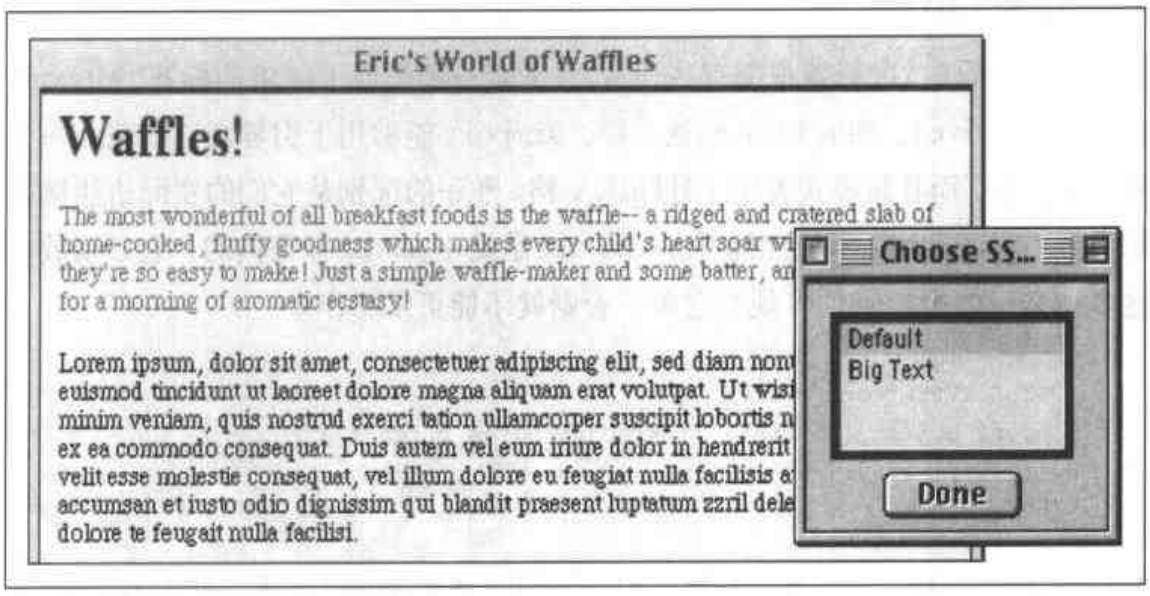


图 1-4　浏览器提供可供选择的样式表

警告：在写本书时，仅有一个浏览器支持可选择的样式表 —— Internet Explorer for Macintosh —— 而且只能配合JavaScript使用，而它并不是随浏览器一起载入的。三个常用的浏览器中没有一个支持可选择样式表（如图1-4所示）。

在写本书时，唯一能识别可选择样式表的浏览器（Internet Explorer for Macintosh）不会从任何REL的取值为alternate stylesheet的LINK元素中应用样式，除非用户选择了样式表。

STYLE元素

STYLE元素相对来说属于HTML中较新的元素，它也是定义一个样式表最常用的方式，所不同的是，样式本身也出现在文档中。STYLE总是使用TYPE属性；在一个CSS1文档里，正确的值为text/css，就像在LINK标签中的那样。因此，STYLE属性总是以 <STYLE TYPE="text/css"> 开始，紧跟后面的是一个或多个样式，最后以一个结尾标签 </STYLE> 为结尾。

在开始和结尾STYLE标签中的样式将作为文档样式表或嵌入样式表被引用，因为这种样式表是嵌入到文档里的。它包含应用于文档的样式，但也包含有使用@import指示引入的对多个外部样式表的链接。

@import指示

现在来看一看STYLE标签里都有些什么。首先是一种类似于LINK的标签：@import指示（directive）。如同LINK标签一样，@import能够用于引导浏览器载入一个外部样式表，而且将样式表用于HTML文档。唯一的区别是它们的实际语法格式和位置不同。正如所看到的那样，@import是处于STYLE标签内的，它必须放在这个地方，位于其他CSS规则之前，否则就不能正常工作。

```
<STYLE TYPE="text/css">
@import url(styles.css); /* @import comes first */
H1 {color: gray;}
</STYLE>
```

像LINK标签一样，在一个文档里可以有不只一条@import语句。然而与LINK不同的是，每一个@import指示的样式表都会被载入并使用。因此，像下面的这三条语句，所有的三个外部样式表都会被载入，而且所有样式规则都将用于此文档的显示：

```
@import url(sheet2.css);
```

```
@import url(blueworld.css);
@import url(zany.css);
```

警告： 只有 Internet Explorer 4.x/5.x 和 Opera 3.x 支持 @import；Navigator 4.x 忽略了这种将样式应用于文档的方法。实际上这可以为这些浏览器“隐藏”样式带来方便。更多细节可参看第十一章“CSS 应用”。

实际的样式

```
H1 {color: maroon;}
BODY {background: yellow;}
```

在例子中除@import语句外，我们还发现了一些普通的样式。它们的具体含义不属于本书讨论的范围，尽管读者可以猜到它们是将H1元素设成了赤褐色，将BODY元素的背景设置成了黄色。

这些样式构成了嵌入样式表的大部分——简单的、复杂的，长的、短的样式规则。而且一个文档的 STYLE 元素中不包含任何规则是很少见的。

如果想让老一点儿的浏览器也能访问当前的文档，那么这儿有个重要的警告。读者或许明白，浏览器会忽略它们不能识别的标记；例如，如果一个网页包含一个 BLOOPER 标签，浏览器会完全忽略这个标签，因为这不是它能识别的标签。

样式表同样如此。如果浏览器不能识别<STYLE>和</STYLE>，它就会一起忽略它们。然而在它们之间的声明却不会被忽略。

因为它们对浏览器来说是普通文本，因此这个样式声明会出现在网页的顶端！（当然浏览器会忽略这些文本，因为它们并不属于 BODY 元素的一部分，而这也不是我们想要得到的网页效果。）这一问题如图 1-5 所示。

为了解决这一问题，推荐读者在注释标签里封装样式声明。在我们下面的这个例子里，注释标签的开始符出现在 STYLE 开始标签的后面，结尾符出现在 STYLE 结果标签的前面：

```
<STYLE type="text/css"><!--
@import url(sheet2.css);
```

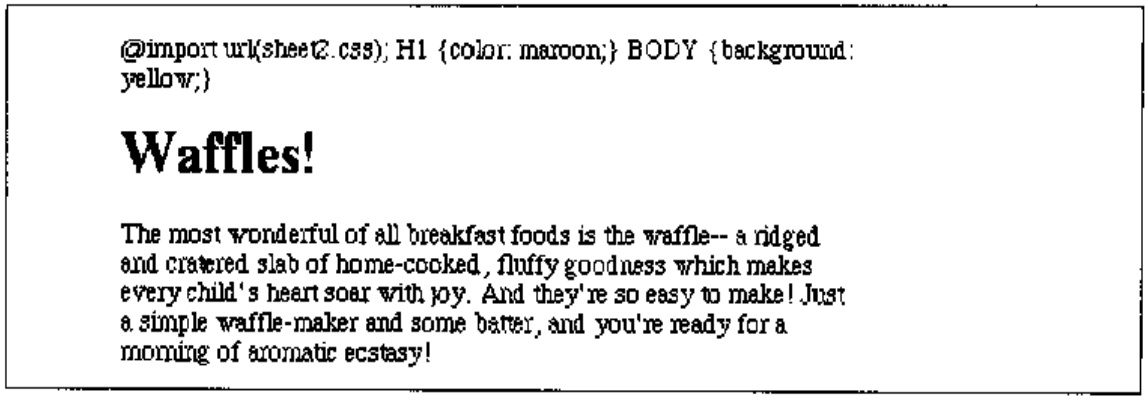


图 1-5 老式的浏览器会逐字逐句地显示样式表

```
H1 {color: maroon;}
BODY {background: yellow;}
--></STYLE>
```

这样就会使老式的浏览器完全忽略STYLE标签及其声明，因为HTML的注释是不会显示的。同时那些能识别CSS的浏览器仍然能够读取样式表。

警告：这种策略也有个缺点。有些老一点儿的浏览器，像非常早版本的Netscape Navigator和NCSA Mosaic，在注释方面还存在一些问题。这些问题可能导致花屏甚至使浏览器崩溃。但这只会发生在很少的浏览器版本中，而且这样的浏览器现在几乎没有人使用了。但还是有一些人还在使用这样的版本，如果使用注释标签则会对他们查看网页带来很大的问题。

CSS 注释

```
/* These are my styles! Yay! */
```

CSS可以使用注释，但语法完全不同。CSS注释非常类似于C/C++注释，都是用/*和*/括起来：

```
/* This is a CSS1 comment */
```

同C++一样，注释可以跨越多行：

```
/* This is a CSS1 comment, and it
can be several lines long without
any problem whatsoever. */
```

记住，CSS 注释不可以嵌套使用。因此下面的这个实例就是不正确的：

```
/* This is a comment, in which we find
another comment, which is WRONG
   /* Another comment */
and back to the first comment */
```

然而，几乎没有必要使用嵌套的注释，所以这个限制不是个大问题。

如果要将注释放在标记的同一行上，则要注意它的放置。例如，下面的方式就是正确的：

```
H1 {color: gray;}   /* This CSS comment is several lines */
H2 {color: silver;} /* long, but since it is alongside */
P {color: white;}   /* actual styles, each line needs to */
PRE {color: gray;} /* be wrapped in comment markers. */
```

对上例来说，如果不是每行都加注释标签，则大多数的样式表都会成为注释的一部分，这样就不能达到预期的目的：

```
H1 {color: gray;}   /* This CSS comment is several lines
H2 {color: silver;}   long, but since it is not wrapped
P {color: white;}   in comment markers, the last three
PRE {color: gray;}   styles are part of the comment. */
```

在这个例子中，只有第一个规则（H1 {color: gray;}）会应用于文档。剩下的规则都被当作注释的一部分，从而被浏览器所忽略了。继续研习我们的例子，在 HTML 标签里我们还会发现更多的 CSS 信息。

内联样式

```
<P STYLE="color: gray;">The most wonderful of all breakfast foods is
the waffle-- a ridged and cratered slab of home-cooked, fluffy goodness...
</P>
```

在某些情况下，可能只需要简单地将一些样式应用于某个独立的元素，而不需要嵌入或使用外部样式，可以使用 HTML 的 STYLE 属性来设置内联样式（*inline style*）。STYLE 属性对 HTML 来说是新的，但它可以同其他任何 HTML 标签一起用，除那些位于 BODY 之外的标签外（例如，HEAD 或者 TITLE）。

STYLE属性的语法也很普通。事实上，它看起来很像STYLE容器（container）内的声明，只是将花括号变成了双引号。因此<P STYLE="color: maroon; background: yellow;">会将文本颜色置为赤褐色，背景颜色置为黄色。但这一声明只对这个特定的段落有效，对文档的其他部分没有影响。

小结

为了提高结构化的HTML的显示效果，允许制作者指定文档的显示样式是必要的。CSS就满足了这一需求，而且远比HTML本身的一些表示元素做得还好。多年来头一次使得网页的结构更加清晰，而且外观更加风格化。

为了保证尽量平稳地过渡，HTML引入了一系列方式将HTML和CSS链接在一起，同时还保持了它们各自的特点。这样，网页制作者就能简化文档的外观布局，并充分发挥其效果，当然也使工作变得更容易了。为了下一步进入XML的世界，CSS对可访问性和位置文档方面所做的改进使其成为令人注目的一项技术。

关于对用户代理支持，LINK元素已经受到广泛的支持，而且STYLE元素和属性也同样被广泛采用。但@import却没有那么顺利，它被Navigator 4排除在外了。虽然这或许会让人感到苦恼，但它并不是CSS的主要策略，况且LINK元素也能够引入外部样式表。

为了完全理解CSS怎样做到这些，制作者需要牢牢掌握CSS是怎样处理文档结构的，某条规则是怎样按预期工作的，以及名字中的“层叠”到底是什么含义等等。

第二章

选择符与结构

网页设计者的生活有时是很艰苦的。为了一个新的设计，他们要埋头苦干，甚至要去参阅17个委员会制定的相关规范和4个主要修订版本的相关标准和信息。最后总算得到了一个令人满意的效果，而此时，一个副总裁却突然说："我正在考虑我们标题中所使用的绿色调，能否让我们看到一些较亮色调或者更暗色调的网站版本呢？"

好了，现在有必要召开另一个会议了。设计者又得重新回到电脑前设计一个新版本，需要将所有的<FONT COLOR="green">标签替换为另一种不同色调的标签。同时，其他管理员也要开始考虑他们自己的方式，以适应新的设计样式。或许标题应该是暗蓝色而不是绿色，或许边框的背景颜色不对，或许应该用公司的标志作为列表项前的项目符号，而不是像其他的那样使用小黑点。

因此，在下一次设计会议上，在取得大家的一致同意后，所有这些新版的修订思想就开始涌现了，所有的管理员都在点头称道。为什么要这样做呢？或许我们应该看到使用红色而非绿色的新版设计。于是一场螺旋式的战斗又打响了。

即使是工作在一个非常幸运的环境下，不需要忍受这样的无理取闹，但有时，自己也会提出类似的问题。蓝色的边框背景与黄色的链接对比度够吗？不同段落中的标题使用不同的字体会怎样呢？红色的标题是不是看起来更好呢？唯一的方法就是将FONT标签和BGCOLOR属性乱摆弄一通。如果文档很多或者设计很复杂，这

就会花掉很多时间。而且如果突然要转向一个不同的设计方向时，可能花在清理原有的残留物上的时间会与实际用于创作设计工作上的时间一样多。

而样式表能提供一种简单、方便、强有力的方式来摆脱这个困境。CSS给设计者的一个最大的便利，就是能将一套样式应用于同种类型和所有元素的这一能力。这可能说得还不够明白，但考虑：编辑CSS中的某一行就能够改变所有标准的颜色。不喜欢使用蓝色吗？改变一行代码，所有的标题都可变为紫色，或黄色，或褐色，或是其他任何喜欢的颜色。设计时间和繁琐的工作也因此而减少了，而且能更好地将注意力集中在创新上。下次开会时，如果某个人想看到不同颜色的标准，只需编辑网页的样式，再重新载入即可。哇！变化后的结果就摆在所有人的面前，仅仅花了几秒钟的时间就完成了，再也不需要开下一次会了！

当然，CSS不能解决所有的问题——例如，不能用它来改变GIF的颜色，但它能使工作变得比过去更容易。这是通过选择符（selector）和结构（structure）来实现的，选择符用于产生变化，而变化又充分利用了结构。

基本规则

CSS的主要功能就是将某些规则应用于文档中同一类型的元素，这样可以减少网页设计者大量的工作。例如，将所有H2元素的文本变成灰色。如果直接使用HTML，必须在所有H2元素中插入<FONT color="gray">...</FONT>标签，像下面这样：

```
<H2><FONT COLOR="gray">This is H2 text</FONT></H2>
```

如果文档中有许多H2元素，这可能会是很乏味的工作。如果后来又决定要将所有H2元素变为绿色而非灰色，那么事情就变得更复杂了，必须找到所有的H2元素，然后将其FONT标签的值修改为灰色。

在CSS里，可以避免这样的麻烦，用更加容易的方式能得到一样的效果。只需要在文档的样式表里定义下面这样一条规则即可：

```
H2 {color: gray;}
```

这就称为一条规则（*rule*），就这么一条简单的规则就能使所有H2元素的颜色变为灰色。如果想要变换成另外一种颜色，那么只需修改本条规则就可以影响文档中所有的H2元素：

```
H2 {color : silver;}
```

规则结构

为了理解得更为透彻，让我们来分析一下规则的结构。

每条规则有两个部分：选择符和声明（*declaration*）。每条声明实际上是属性和值的组合。每个样式表就是由一系列规则组成的，但规则并不总是出现在样式表里。

首先，让我们来分解下面的一条规则，如图2-1所示。

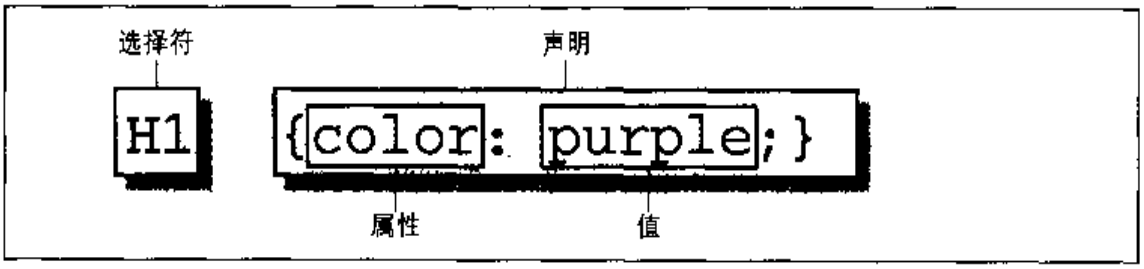


图2-1 规则的结构

正如图中所示，规则的左边就是选择符。所谓选择符就是规则中用于选择文档中要应用样式的那些元素。在本例中，H1元素被选中了。如果上例中的选择符为P，那么所有的P元素（段落）就会被选择，而H1元素不受影响。

规则的右面部分是声明。它是由CSS属性及其值组成的。在图2-1中，声明部分说明此规则会使文档的某部分变成紫色。而将应用此规则的元素就是选择符所指示的（本例中是所有的H1元素）。

简单的选择符

选择符通常是一个HTML元素，但也可以是其他的。例如，如果一个CSS文件包含XML文档的样式，它看起来可能如下所示：

```
QUOTE {color: gray;}
BIB {color: red;}
BOOKTITLE {color: purple;}
MYElement {color: red;}
```

换句话说，最基本的选择符是一个文档中的某个元素。在XML里，可以是任何东西。另一方面，如果是为HTML定制样式，常常是使用某个HTML的元素，像P，H3，EM，A或者是BODY元素，如下：

```
BODY {color: black;}
H1 {color: purple;}
H3 {color: gray;}
STRONG {color: red;}
EM {color: maroon;}
```

这种样式的结果如图2-2所示，带有明显的灰度级。

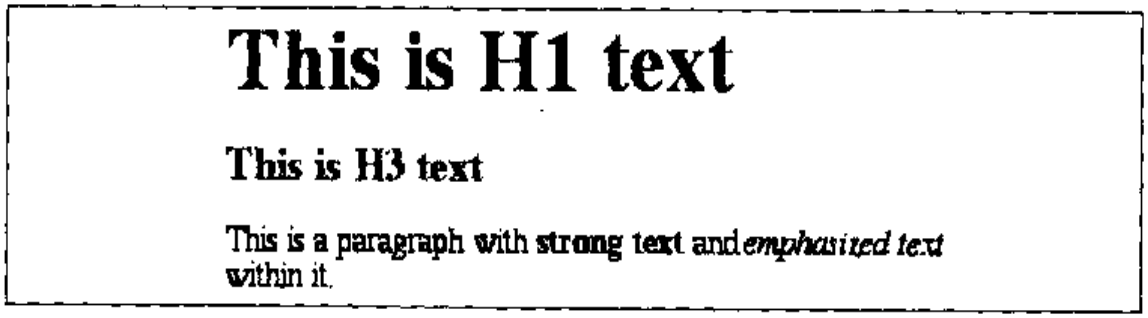


图2-2 一个文档的简单样式

将样式应用于元素的能力显然是很强大的，它可以更容易地使元素的样式由一种变成另一种。假设我们有个网页，其中所有H2元素都为灰色：

```
H2 {color: gray;}
```

看上去还不错，但看的次数越多，它就显得不那么好看了。最终还是决定让段落文本显示灰色而不是H2文本。没问题！所有要做的工作就是将选择符由H2改为P，这样就能将原来H2元素的样式转换到P（段落）元素：

```
P {color: gray;}
```

刚才我们讨论了规则的左边，在继续深入讨论选择符的一些附加特性之前，让我们来看看右边的声明部分。

声明

声明的格式是固定的，某个属性（*property*）后跟一个冒号，然后是其取值。最后，这条声明以分号（;）作为终结。而值可以是单个关键字（*keyword*）或者是由空白符所分开的一串关键字组成的。如果有一个不正确的属性被用于声明中，那么，整条声明将被忽略。而且后续的声明也将被忽略，因为即使值是正确的，属性也不正确：

```
brain-size: 2cm;
```

如果某个值不正确，多数情况下只有那个值被忽略——尽管它可能使整条规则作废。然而，这种可能性对多数浏览器来说是很小的，因为大部分都能忍受CSS中的一定的错误：它们只是丢掉不认识的值，使用声明中的剩余部分，而不是忽略整条规则（这听起来好像是做了件礼貌的事，但对于制作者来说也更容易培养坏的编写习惯）。

如果使用多个关键字作为一个属性的值，通常用空白符将它们分开。并不是每个属性都接受多个关键字，但大部分都可以：例如，font属性。如果想要使段落的文本采用中等尺寸的Helvetica字体，那么规则应该如下所示：

```
P {font: medium Helvetica;}
```

注意medium和Helvetica之间的空格，两边各为一个关键字（第一个是字体尺寸，第二个是字体名）。这一空格使得用户代理能分辨关键字，从而正确地使用它们。最后的分号表示规则结尾。

之所以要将这些以空格符分开的单词当作关键字，是因为它们全部合起来才形成属性的值。例如，下面这条虚构的规则：

```
rainbow: red orange yellow green blue indigo violet;
```

当然，没有像rainbow这样的属性，而且上面的许多颜色也是无效的，但它对于我们举例还是有用的。所以我们对于rainbow这个属性所得到的值就是red orange yellow green blue indigo violet。这七个关键字合在一起成为一个值。也可以重新定义其值：

```
rainbow: infrared red orange yellow green blue indigo violet ultraviolet;
```

现在rainbow就有了一个不同的值，它是由九个而不是七个关键字组成的。尽管这两个值看似相似，但它们就像0和1一样是不同的值。

在有一些例子中，关键字不是由空格分隔的。font就是一个很好的例子，而且只有这一个地方两个关键字由正斜杠（/）分开。如下：

```
H2 {font: large/150% sans-serif;}
```

斜杠分开了字体大小和行高。这是在font声明中斜杠唯一能出现的地方。所有其他关键字还是由空格来分开。

这些就是一个简单声明需要注意的地方，正如简单的选择符那样，没有什么可以多说的，尽管我们并不局限于这样简单的操作。下面就让我们来看看如何能得到更强大的CSS。

分组

迄今为止，事情就这么简单——如果只想将一个样式应用于某个选择符，就是这样轻而易举。毫无疑问，我们要求的不仅仅是这些；有时，我们或许需要让同一条规则应用于多个元素，也就是多个选择符，或者要将更多的样式应用于一个或一组元素。

分组选择符

假设文档中的H2元素和P元素都应是灰色的文本。最简便的方式就是像下面的这样：

```
H2, P {color: gray;}
```

通过将H2和P元素同时放在规则的左边并且用逗号分隔它们，右边为规则定义的样式（color: gray;），规则将被同时应用于两个选择符。逗号告诉浏览器在这一条规则中包含两个不同的选择符。如果漏掉了逗号，将会赋予这条规则完全不同的意义。我们将在“上下文选择符”一节中对此展开讨论。

也可以一次分组任意多个选择符。如果想让文档中的每个元素都为灰色，可以用下面的规则：

```
BODY, TABLE, TH, TD, H1, H2, H3, H4, P, RE, STRONG, EM, B, I, {color: gray;}
```

正如可以看到的那样，分组可以让作者大量地压缩某种类型的样式分配，否则它们会占用很长的样式表。下面的两种方式结果一样，但哪一种更为容易是很明显的：

```
H1 {color: purple;}
H2 {color: purple;}
H3 {color: purple;}
H4 {color: purple;}
H5 {color: purple;}
H6 {color: purple;}
```

或者使用：

```
H1, H2, H3, H4, H5, H6 {color: purple;}
```

分组可以有一些有趣的选择方式。例如，下面的所有样式表都是等价的——每一个样式表都显示了一种不同的分组方式——而它们所得到的效果都一样，如图 2-3 所示。

```
H1 {color: purple; background: white;}
H2 {color: purple; background: green;}
H3 {color: white; background: green;}
H4 {color: purple; background: white;}
B {color: red; background: white;}

H1, H2, H4 {color: purple;}
H2, H3 {background: green;}
H1, H4, B {background: white;}
H3 {color: white;}
B {color: red;}

H1, H4 {color: purple; background: white;}
H2 {color: purple;}
H3 {color: white;}
H2, H3 {background: green;}
B {color: red; background: white;}
```

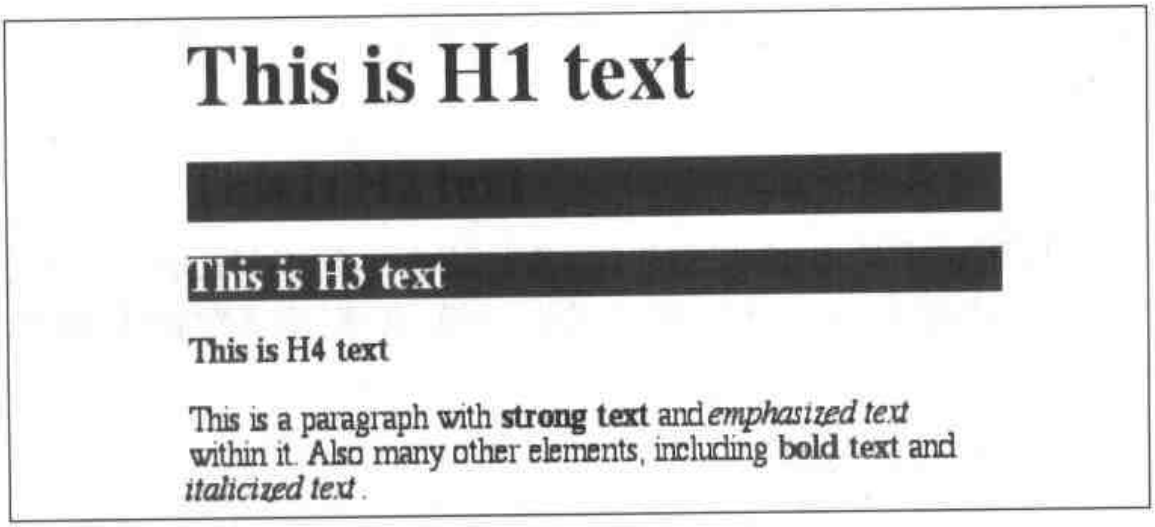


图 2-3 等价的样式表结果

分组声明

既然可以为一条规则中的选择符分组，那么声明也应该能够分组。在这里用分号作为一条声明的结束，这样可以显得更加清楚，因为一条规则里可能含有多个声明。这样还能使样式表更加紧凑、有组织、易于阅读。例如，要让某个背景的所有H1元素都具有紫色文本，而且用18磅的Helvetica字体，可以像下面这样书写样式：

```
H1 {font: 18pt Helvetica;}
H1 {color: purple;}
H1 {background: aqua;}
```

这样做效率并不高——试想如果要为某个元素分配10个或者15个样式，情形会怎样呢！所以，必须对声明分组：

```
H1 {font: 18pt Helvetica; color: purple; background: aqua;}
```

这同上面三行样式表所表达的效果是一样，其结果如图2-4所示。

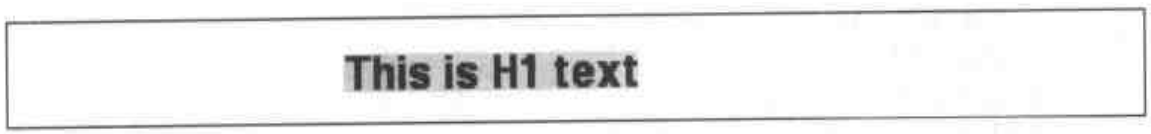


图 2-4 分组声明

当在一条规则中分组声明时，分号作为分隔符使用。这一点尤为重要，因为空白会被忽略掉。当然，像下面这样的格式也是可以的：

```
H1 {
  font:       18pt Helvetica;
  color:      purple;
  background:    aqua;
}
```

如果第二个分号被删掉，用户代理就会将其翻译成下面的样式：

```
H1 {
  font: 18pt Helvetica;
  color: purple background: aqua;
}
```

因为background:并不是个有效的颜色值，而且color只能被赋予一个关键字，所以整条语句都将被忽略。在这种情况下，用户代理就不能按预期的效果来显示了。既然只能有一个关键字赋给color，用户代理就会忽略整条声明，这也就意味着不能得到紫色的H1元素；进而得到的是没有背景的缺省颜色（通常为黑色）。声明font: 18pt Helvetica仍然有效，因为它以正确的分号作为结尾，但其他样式就不能工作了。

注意：从技术上来讲，没有必要在最后的声明后加个分号，但这不失为一个好的习惯，首先，它能保持用分号来中断一条声明的习惯，缺少这样一个分号常常是引起错误的主要因素。其次，如果想往一条规则里加入另一个声明，就不必担心会忘记插入一个特殊的分号了。最后，如果将最后的那个分号去掉的话，像IE 3.x这样较老的浏览器会出毛病。为了避免所有的这些问题，建议还是加上这个分号为好。

同选择符分组一样，声明分组是保持样式表短小精悍且易于维护的一种便利方式。下面的例子显示了用两种方式为H1元素分配六种不同的样式的情形。第一种是采用独立规则方式，第二种采用声明分组的方式。第二种方式的好处是只需编辑一个选择符，而不是六个，就能改变其样式。尽管两种方式都可以得到图2-5一样的结果：

```
H1 {color: gray;}
H1 {background: white;}
H1 {border: 1px solid black;}
```

```
H1 {padding: 0.5em;}
H1 {font: 20pt Charcoal,sans-serif;}
H1 {text-transform: capitalize;}

H1 {color: gray; background: white; border: 1px solid black; padding: 0.5em;
  font: 20pt Charcoal,sans-serif; text-transform: capitalize;}
```

图2-5 分组与不分组，结果一样

同时分组

现在，我们能分组选择符，也可以分组声明。将二者结合起来，就可以用一条语句来定义复杂的样式。假设我们需要为文档中的一些标题分配一些复杂的样式。如下：

```
H1, H2, H3, H4, H5, H6 {color: gray; background: white; padding: 0.5em;
  border: 1px solid black; font-family: Charcoal,sans-serif;}
```

分组选择符意味着右边的所有样式都应用于列出的标题元素，而分组声明表示所有列举的样式都被运用于左边的选择符。其结果如图2-6所示。

这种方法显然更可取，要不然像下面这样逐一定义就会比较繁琐，而且下面的还没有完全写出来。

```
H1 {color: gray;}
H2 {color: gray;}
H3 {color: gray;}
H4 {color: gray;}
H5 {color: gray;}
H6 {color: gray;}
H1 {background: white;}
H2 {background: white;}
H3 {background: white;}
```

当然，如果愿意的话，可以像这样来书写——但不推荐这样做。否则，当需要修改样式的时候，面临的问题将和满篇使用FONT标签一样多！

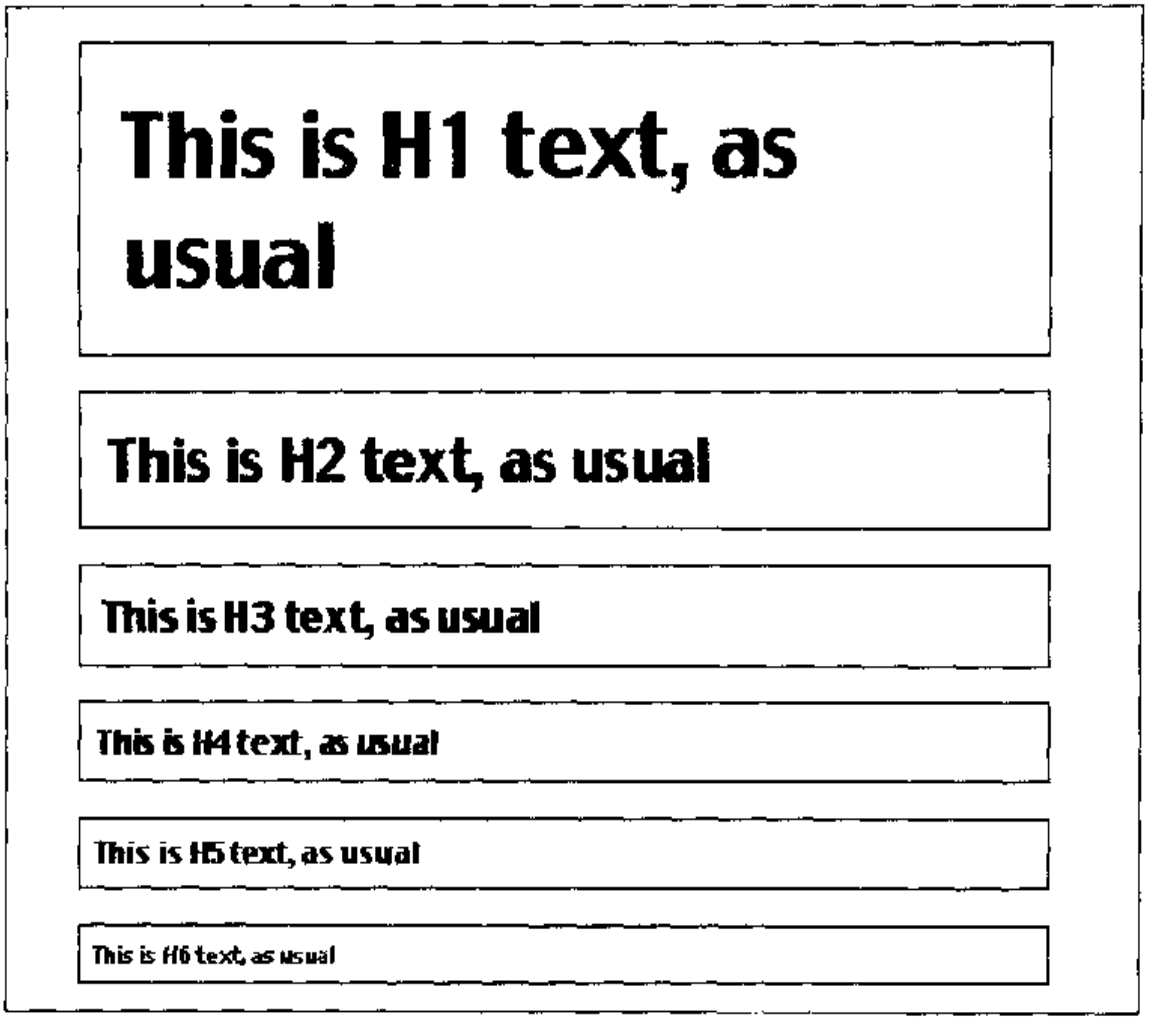


图 2-6　同时分组选择符和规则

幸亏有了分组才使CSS文件能像今天这样。下面是一个典型的样式表，其结果如图 2-7 所示：

```
BODY {background: white; color: gray;}
H1, H2, H3, H4, H5, H6 {font-family: Helvetica, sans-serif;
  color: white; background: black;}
H1, H2, H3 {border: 2px solid gray; font-weight: bold;}
H4, H5, H6 {border: 1px solid gray;}
P, TABLE {color: gray; font-family: Times, serif;}
PRE {margin: 1em; color: maroon;}
```

尽管我们已介绍了它强大的功能和复杂性，但对于选择符来说，还有值得研究的地方。可以在选择符中加入更多表达式，这样就可以根据信息类型对不同的元素应用不同的样式。当然要得到这样强大的功能，必须下更大的功夫，但这是值得的。

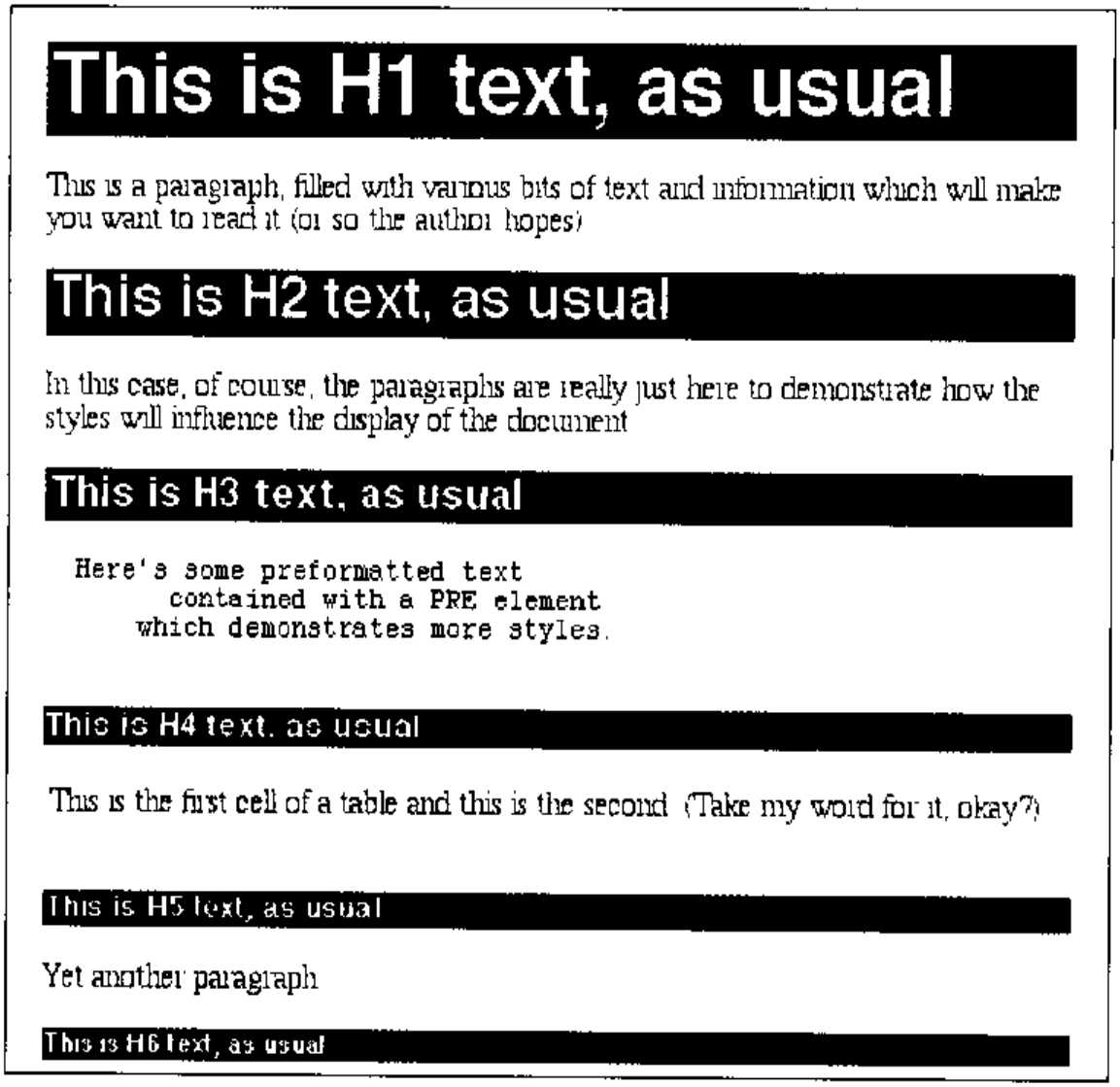


图 2-7 一个典型的样式表的结果

类选择符和 ID 选择符

迄今为止，我们已经以多种方式组合了选择符和声明，但选择符的运用还是相对比较简单的。它们只涉及到了文档元素，但这也只是通常所需的，还有更特殊的场合需要更专门的属性支持。

除基本的文档元素外，还有两个其他的选择符：类（*class*）选择符和ID选择符，它们允许以独立于文档元素的方式来分配样式规则的应用。这些选择符既可以独立运用，也可以和元素选择符合用。然而，只有当文档被正确标记后，它们才能发挥作用，所以使用它们时得事先考虑和计划好。

当然，读者或许想急于知道为什么要这样做。这样做的好处又是什么呢？让我们来看一个有关处理放射性物质钚的文档，在文档中很多是关于处理这一物质时应该引起足够重视的警告。而这些警告文本应该使用粗体以区别于其他的文本，使其更加醒目。

但是却不能具体确定这些警告是何种类型的元素。有时是一整个段落的警告文本，有时又只是一个一个的警告列表项，或者干脆是一个段落的某个部分。不论是何种情况，都不能为之选择一个简单的选择符加以描述。如果像这样：

```
P {font-weight: bold;}
```

那么所有的段落都会变成粗体，而不仅是包含警告的那个段落。所以，需要提供一种方式仅选择具有警告的那个段落，或者更精确，只选择那些属于警告的文本。

另一种情形就是对不同的链接采取不同的样式。或许可以为指向其他站点网页的链接设置不同的颜色。同样，也不能仅仅写下面一条规则：

```
A {color: maroon;}
```

因为这样会影响所有的链接，而不管它指向的网页位于本网站还是其他网站。

因此我们所需要的是将样式应用于文档中以某种方式标记过的那一部分，而同元素无关——这也就是CSS所要做的。

类选择符

最通常的方式就是使用类选择符。但在使用它之前得对实际文档作标记，这样才能使这个选择符发挥作用。为什么呢？既然我们不打算根据HTML元素来作选择，那么就需借助其他的手段。下面来看类属性：

```
<P CLASS="warning">While handling plutonium, care must be taken to avoid
the formation of a critical mass.</P>
<P>During this step, <SPAN CLASS="warning">the possibility of implosion is
very real, and must be avoided at all costs</SPAN>. This can be accomplished
by keeping the various masses separate...</P>
```

为了将类选择符的样式关联到一个元素，那个元素必须要设置合适的类属性值。

在上面的代码中，我们已经将一个类值warning分配给了两个元素：第一段和第二段中的SPAN元素。

现在所有要做的就是对这些标记了的元素应用样式。如下：

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

将它同前面的实例标记结合起来，这条简单的规则就会有如图2-8所示的效果。实际上，给定上面这样一个样式表后，font-weight: bold这个样式就会应用于任何类属性值为warning的元素。

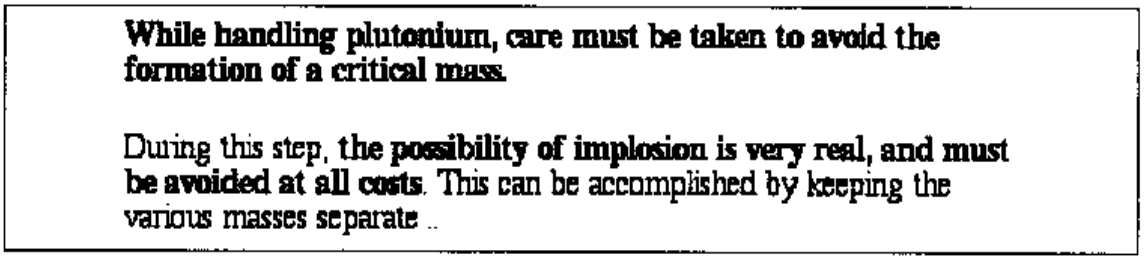


图2-8 使用类选择符的结果

正如所看到的，类选择符通过直接引用元素中类属性的值而产生效果。在这个引用前面总是一个句点（.），用它来标识一个类选择符。这个句点是必要的，因为它可以帮助类选择符与其他元素相分离，这些元素也可能和它组合在一起——像元素选择符一样。或许我们只想当整个段落都是警告文本时才使其为粗体。这样：

```
P.warning {font-weight: bold;}
```

如图2-9所示，只有第一段是粗体，然而第二段中的文本就不再是粗体，因为它不再匹配SPAN元素。而选择符P.warning表明："任何类属性的值为warning的段落都将采取统一的样式。"既然SPAN元素不是一个段落，这条规则就与它无关，因此它的文本就不会变为粗体。

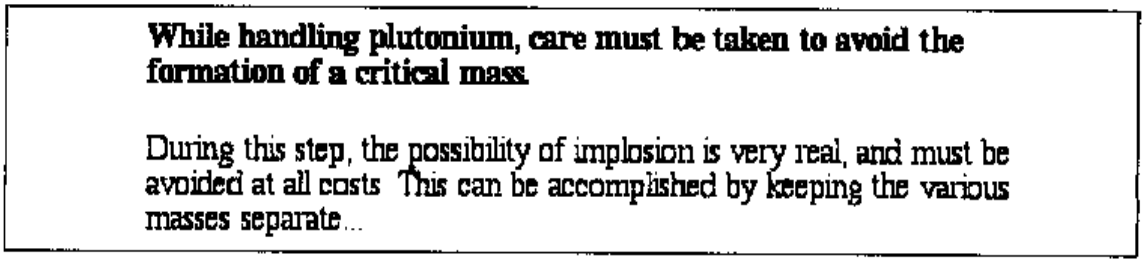


图2-9 将元素选择符与类选择符进行组合

当然，也可以使用SPAN.warning选择符来分配不同的样式。参见图2-10中这些样式的结果：

```
P.warning {font-weight: bold;}
SPAN.warning {font-style: italic;}
```

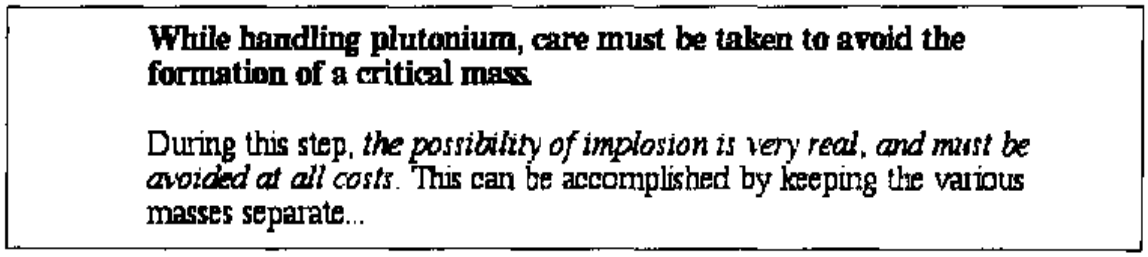


图2-10 使选择符更具体

在这种情况下，警告段落将以粗体显示，然而警告SPAN为斜体。每条规则只能应用于一个特定类型的类组合元素，而且不会影响到其他的元素。

另外，还可以使用一般类选择符或者特定元素的类选择符来使样式的效果更佳，如图2-11所示：

```
.warning {font-style: italic;}
SPAN.warning {font-weight: bold;}
```

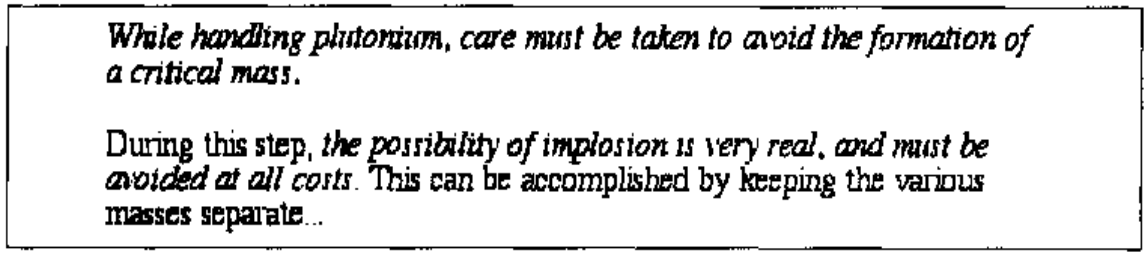


图2-11 使用一般的和具体的选择符来组合样式

现在所有警告文本都将是斜体，只有类为warning的SPAN元素内的文本才是粗体字。

ID选择符

在很多方面，ID选择符都类似于类选择符——但也有一些重要的区别。第一个区别就是ID选择符的前面是#号——而不是句点。看看下面的这条规则：

```
#first-para {font-weight: bold;}
```

这会使任何ID属性的值为first-para的元素样式为粗体文本。这就是第二个区别：ID选择符指向的是ID属性中的值，而不是引用类属性的值。下面是一个ID选择符的实例：

```
#first-para {font-weight: bold;}

<P ID="first-para">This is the first paragraph, and will be boldfaced.</P>
<P>This is the second paragraph, which will NOT be bold.</P>
```

如图2-12所示，段落中的文本显示正如上面所描写的那样：第一段为粗体，而第二段不是。

This is the first paragraph, and will be boldfaced.

This is the second paragraph, which will NOT be bold.

图2-12 使用ID选择符

注意first-para值并不局限于某个段落。它可以分配给文档中的任何元素。

类和ID有什么区别？

在前面，我们已经以相似的方式使用了类选择符和ID选择符。它们的使用或多或少地影响到HTML的结构属性，因为样式可以应用于任何元素，而不管它在文档中的结构功能如何。这并不是我们所希望的，因为理论上类和ID选择符可以让任何元素的行为类似于其他元素，而这些元素都是使HTML成为结构化语言而引入的。我们暂且将这些放到一边，那么到底类和ID间的区别在哪里呢？

首先，类可以分配给任何数量的元素，像我们前面所看到的——warning这个类同时应用于段落和SPAN元素，而且它可以应用于许多其他的元素。另一方面，ID却只能在某个HTML文档中使用一次。从这点来看，它们有点类似于表格元素INPUT中的NAME属性。每个NAME属性的值应该是唯一的，类似于ID。

注意： 在实际应用中，浏览器并不一定检查HTML文档中ID的唯一性。也就是说可以在HTML文档中设置多个元素，而这些元素对于它们的ID属性来说值相同，从而同一个样式可应用于多个元素。虽然这种用法是不正确的，但却常常被使用。顺便提一下，这在HTML文档中是不对的。其他的标记语言对ID可能没有这一约束，尽管我们无法直接得知，但去查一查它的语言规范就知道了。无论如何在HTML文档中，不允许某个ID的值同其他ID的值一样。

类和ID的另一个区别是：ID对给定元素应用何种样式比类具有更高的优先权。这将在后面的“层叠”一节详细讲述。

和类一样，ID的声明也能独立于HTML元素，尽管它们应该是唯一的，但这已无关紧要了。当然，可能在某些情况下能够预知某个ID值会出现在文档中，但却不知道是在哪个HTML元素上。所以能够声明独立的ID选择符是很有用的。例如，在某个给定的文档中，将有一个ID值为mostImportant的元素。但却不知道这个元素到底是段落、短语、列表项，还是标题，只知道它存在于这个文档中的某个元素上，而且出现不止一次。于是可以写出这样一条规则：

```
#mostImportant {color: red; background: yellow;}
```

这条规则可以匹配任何下列元素（像前面提到的那样，它们不应该出现在同一个文档中，因为它们具有相同的ID值）：

```
<H1 ID="mostImportant">This is important!</H1>
<EM ID="mostImportant">This is important!</EM>
<LI ID="mostImportant">This is important!</LI>
```

伪类和伪元素

更有趣的是伪类（*pseudo-class*）和伪元素（*pseudo-element*）选择符，至少在语法上讲是这样的。它们允许将样式应用于文档中不存在的结构上，或者是通过当前元素状态甚至是文档自身的状态而推断的某些东西上。换句话说，可以不依赖于文档结构，而且在不能简单地通过研究文档的标记来推断的情况下，将样式应用于文档的某个部分。

看起来好像我们是在随机地应用样式，其实并非如此。在这里，样式的应用是根

据不能提前预测的即时条件而确定的。而且，对能够应用样式的情形都很好地进行了定义。这就好比是一场体育赛事，不管主队在什么时候得分，观众都会为之喝彩。用不着去关心它什么时候到来，只要适当的条件出现了，观众们就会像预期的那样为之喝彩。虽然不知道到底会在哪一刻（或哪一局，哪一节）主队会得分，但是这并不会影响观众的预期行为。

伪类选择符

首先我们来看看伪类选择符。由于它们得到浏览器的良好支持，所以也得到了广泛的应用。让我们借助一个实例来看看它们是怎样运作的。在讲到实质性的东西前我们会花掉一些时间，因此请耐心地跟我来吧。

考虑定位锚标记，它用来在一个文档和另一个文档间建立链接。当然，定位锚只是一种标记，但某些定位锚指向那些已经访问过的网页，而另外一些则指向还没有访问过的网页。而且这不是简单地查看HTML标记就能判断出来的，因为在标记里，所有的定位锚都一样。只有将文档中的链接同用户浏览器的历史记录结合起来，才能判断哪些是已访问过的，哪些还没有。因此，从某种意义上讲，有两种基本类型的定位锚标记：访问过的和未访问过的。实际上，它们都属于伪类，而且所使用的选择符被称作伪类选择符。

要明白其含义，先让我们来看一下浏览器是怎样处理链接的。Mosaic的常规作法是让未访问过的链接显示为蓝色，已访问过的链接显示为红色（这在后来的一些浏览器，如Internet Explorer中已被改成紫色了）。如果要在定位锚中插入一个类，那么任何访问过的定位锚都应该有一个“已访问过的”类，这样就能使其变为红色：

```
A.visited {color: red;}

<A HREF="http://www.w3.org/" CLASS="visited">W3C Web site</A>
```

然而这样一种方法要在每次访问一个新网页时改变定位锚中的类，这就显得有些麻烦。取而代之的是，CSS定义的伪类可以使访问过的链接具有“已访问过的”类的特征。如下：

```
A:visited {color: red;}
```

就这么简单——任何指向已访问过的网页的定位锚都将是红色，而且不必为任何定位锚添加类属性。注意规则中的冒号（:）。分隔A和“visited”的这个冒号被称作伪类或伪元素的标记符。所有伪类和伪元素关键字都是以一个冒号开头的。

下面是另一个例子：

```
A:visited {color: silver;}

<A HREF="http://www.w3.org/">W3C Web site</A><BR>
<A HREF="http://www.nowhere.net/">Nowhere in particular</A>
```

如图2-13所示，第一个定位锚指向已访问过的网页，其颜色为银色，而第二个定位锚仍是蓝色，因为在这之前，浏览器还没有载入过这个网页。

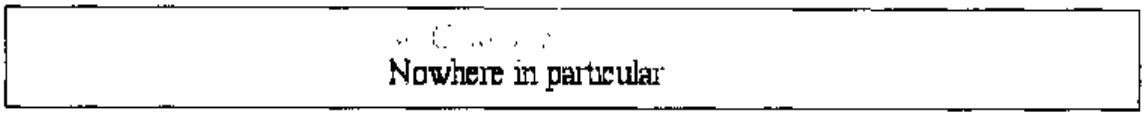


图2-13 运作中的伪类

在CSS1中，只有三个伪类。前面已经见过的:visited d，再加上:link和:active。每个都指代一种定位锚，如表2-1所示。

表2-1 伪类

名字	描述
:link	指超链接（也就是说具有HREF属性）是一个未访问过的地址。注意有些浏览器可能将:link翻译成指向任何超链接、访问过或未访问过的
:visited	指已经访问过的网页的定位锚
:active	指任何已经处于活跃状态（例如被点击）的定位锚。在CSS1里，这只用于超链接，然而在CSS2中，:active理论上可应用于任何元素

表2-1中的第一个伪类看起来似乎是多余的。毕竟，如果定位锚未被访问过，那么它肯定是未访问过的（unvisited）。因此，我们就只需像下面这样去做：

```
A {color: blue;}
A:visited {color: red;}
```

看上去似乎是合理的，但实际上并不完全如此。第一条规则不仅会应用于未访问过的链接，而且会应用于目的定位锚，如下：

```
<A NAME="section4">4. The Lives of Salmon</A>
```

上面的文本也会是蓝色的，因为A元素也匹配上面的规则。所以为了避免将链接样式应用于目的定位锚，应该使用:link伪类：

```
A:link {color: blue;}    /* unvisited links are blue */
A:visited {color: red;}   /* visited links are red */
A:active {color: yellow;}  /* anchors turn yellow while clicked */
```

或许读者已经意识到了:link,:visited和:active选择符在功能上等价于BODY属性的LINK，VLINK和ALINK。当然，对于CSS中的伪类来说，应用不仅仅是在颜色方面。

假设某人想创建一个网页，使所有未访问过的链接呈紫色，访问过的呈红色，当用户点击时呈蓝色，可以像下面这样来做：

```
<BODY LINK="purple" VLINK="red" ALINK="yellow">
```

在CSS中，可以这样来实现：

```
A:link {color: purple;}
A:visited {color: red;}
A:active {color: yellow;}
```

在这里我们又可以重温一下类选择符以及它们是怎样同伪类组合的。例如，假设要改变指向网站之外的链接的颜色。这很容易，可以为这些定位锚分配一个类。例如：

```
<A HREF="http://www.mysite.net/">My home page</A>
<A HREF="http://www.site.net/" CLASS="external">Another home page</A>
```

为了对外部链接应用不同的样式，只需一条规则即可：

```
A.external:link, A.external:visited {color: maroon;}
```

这就可以将前面的第二个链接的颜色设成褐色，而第一个链接将是缺省的链接颜色（通常是蓝色）。

实际问题

对于定位锚伪类有一些有趣的问题。例如，可以将访问过和未访问过的链接设置成相同大小的字体，而将活跃的链接的字体设得更大一些：

```
A:link, A:visited {font-size: 12pt;}
A:active {font-size: 18pt;}
```

正如从图2-14中可以看到的那样，用户代理增大了链接的字体。支持这种行为的用户代理在这一链接被点击后不得不重新显示此文档。然而，CSS规范特别陈述了这样一条规则：一旦在文档最初显示一次后，用户代理就不必重新显示了。但这也不是绝对的，因此设计时也要尽量避免对这一行为的依赖。

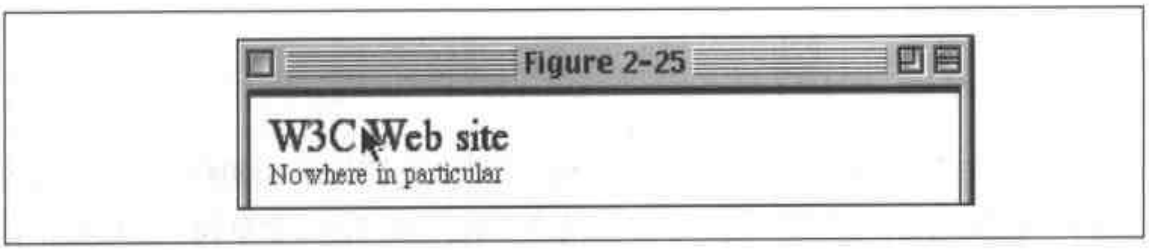


图2-14 根据状态重新调整元素的大小

警告：Navigator 4.x和Opera 3.x并不支持伪类，而IE 4.x和IE 5.x都支持它。

伪元素选择符

同伪类的方式类似，伪元素通过对插入到文档中的虚构元素进行触发，从而达到某种效果。

在CSS1里，有两个伪元素，即:first-letter和:first-line。它们将样式分别应用于首字母或首行，而首字母和首行位于像段这样的块级元素中。例如：

```
P:first-letter {color: red;}
```

这会使每个段落的首字符变为红色，是不是显得很简单？还可以使H2标题的首字母比其他的字母大两倍，结果如图2-15所示：

```
H2:first-letter {font-size: 200%;}
```

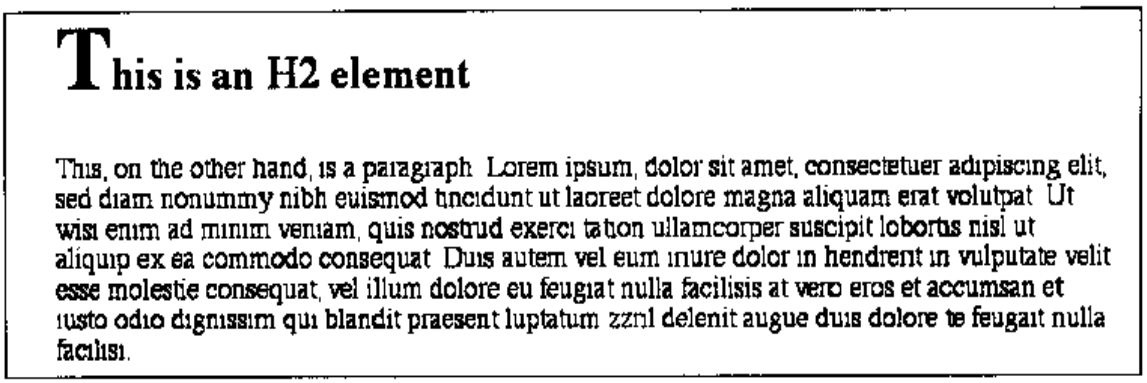


图 2-15 应用中的:first-letter 伪元素

同理,:first-line可用于元素中文本的首行。例如，可以让文档中每个段落的首行显示为灰色:

```
P:first-line {color: gray;}
```

在图2-16中可以看到样式应用于每个段落的首行。不管其显示的区域有多宽或多窄，样式都会准确地应用于首行。如果段落的首行只包含五个单词，则只有那五个单词会变灰。如果首行包含 30 个单词，那么所有的 30 个单词都会变灰。

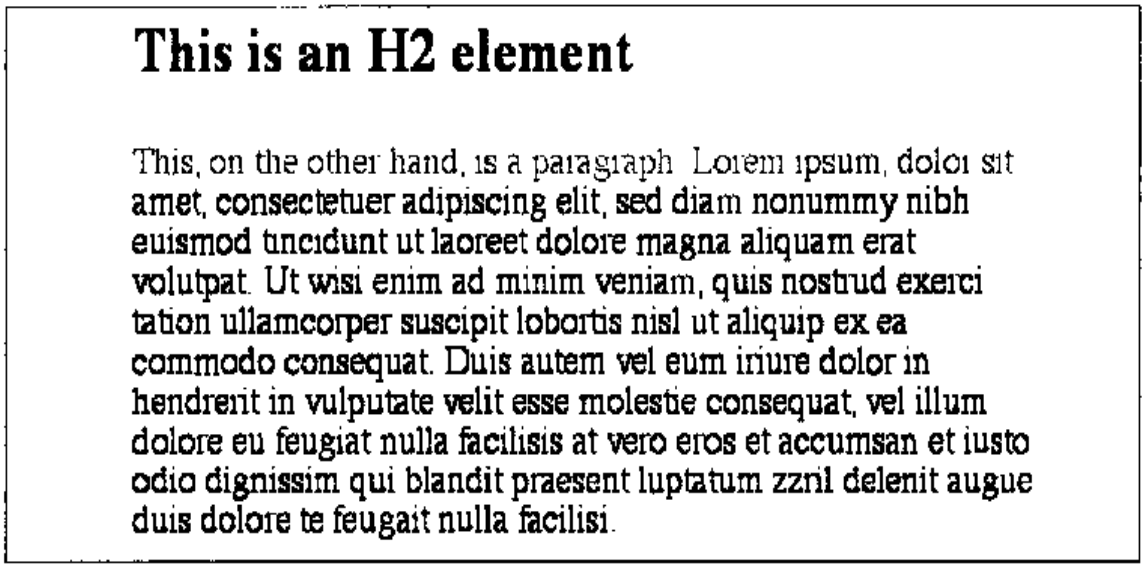


图 2-16 应用中的:first-line 伪元素

之所以:first-line和:first-letter被当作伪元素引用，是因为它们在效果上使文档中产生了一个临时的元素。这是应用“虚构标记”的一个最典型的实例，正如CSS 规范中所描述的那样。

假设有如下标记:

```
P:first-line {color: gray;}

<P>This is a paragraph of text which has only one style applied to it. That
style causes the first line to be gray. No other text in the paragraph is
affected by this rule (at least, it shouldn't be).</P>
```

进而假设用户代理显示的文本如下:

```
This is a paragraph of text which has only
one style applied to it. That style causes
the first line to be gray. No other ...
```

等等。既然从“This”到“only”的文本应该是灰色，那么用户代理就可以使用类似于下面的虚构标记:

```
<P><P:first-line>This is a paragraph of text which has only</P:first-line>
one style applied to it. That style causes the first line to be gray. No other ...
```

虚构的标签产生如图 2-17 所示的效果。

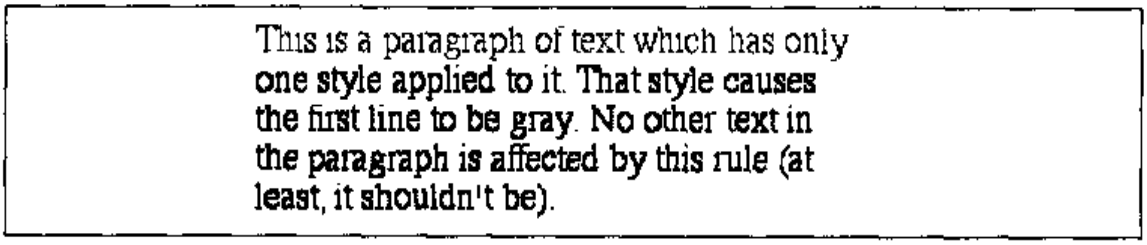


图 2-17 从理论上来讲，伪元素选择符:first-line 是如何工作的

而<P:first-line>元素并没有出现在源文档中，甚至于它根本就不是一个有效的元素。它的存在是建筑在用户代理之上的，其作用是将:first-line样式应用于合适的文本块上。换句话说，<P:first-line>不是一个真正意义上的元素，而是一个伪元素。记住，不必再添加任何新标签，用户代理会完成余下的工作。

:first-letter 伪元素的情况类似:

```
P:first-letter {font-weight: bold;}

<P><P:first-letter>T</P:first-letter>his is a paragraph of text which has
another style sheet applied to it. This time it uses a first-letter effect.
```

伪类和伪元素选择符的限制

应用于:first-line和:first-letter的CSS属性有一些限制，如表2-2所示。

表2-2 能用于伪元素的属性

:first-letter	:first-line
所有font属性	所有font属性
所有color和background属性	所有color和background属性
text-decoration	word-spacing
vertical-align（如果float属性设置成none）	letter-spacing
text-transform	text-decoration
line-height	vertical-align
所有margin属性	text-transform
所有padding属性	line-height
所有border属性	clear
float属性	
clear	

在CSS1下，伪类和伪元素不能组合在一个选择符中。这在CSS2里会做某些修改。但用类和ID选择符来组合伪类和伪元素是可能的，尽管其语法很严格。它们总是处于选择符的最后，紧接在元素、类和/或ID之后：

```
A.external:link {color: gray;}
A#link721:visited {color: purple;}
```

结构

前面已经提到，CSS之所以如此强大，是因为它采用HTML文档结构来决定其样式的应用。但这仅仅只是一方面，因为它只暗示了CSS之所以使用文档结构，仅仅是为了决定将不同的规则应用于不同的元素这一点。

事实上文档的结构在样式的应用中扮演着更为重要的角色。为了理解这一角色，

我们需要理解文档是怎样结构化的。对于下面列举的这个简单的HTML文档，图2-18显示了其对应的“树形视图”：

```
<HTML>
<HEAD>
<BASE HREF="http://www.meerkat.web/">
<TITLE>Meerkat Central</TITLE>
</HEAD>
<BODY>
<H1>Meerkat <EM>Central</EM></H1>
<P>
Welcome to Meerkat <EM>Central</EM>, the <STRONG>best meerkat web site
on <A HREF="inet.html">the <EM>entire</EM> Internet</A></STRONG>!</P>
<UL>
<LI>We offer:
<UL>
<LI><STRONG>Detailed information</STRONG> on how to adopt a meerkat</LI>
<LI>Tips for living with a meerkat</LI>
<LI><EM>Fun</EM> things to do with a meerkat, including:
<UL>
<LI>Playing fetch</LI>
<LI>Digging for food</LI>
<LI>Hide and seek</LI>
</UL>
</LI>
</UL>
<LI>...and so much more!</LI>
</UL>
<P> Questions? <A HREF="mail to:suricate@meerkat.web" >Contact us!</A>
</P>
</BODY>
</HTML>
```

CSS的大部分能力都是基于元素的父子关系。HTML文档，实际上大多数结构文档，都是基于元素的层次关系的，这种层次可以用图2-18中的树形结构来刻画。在这样的层次图中，每个元素都处于整个结构文档中的某个位置，而且每个元素或是另一个元素的父元素，或是子元素，或者既是父元素又是子元素。

一个元素包含了另一个元素，那么这个元素就是另一个元素的父元素。例如在图2-18中，P元素是EM和STRONG元素的父元素，而STRONG是另一个定位锚（A）元素的父元素，这个定位锚元素又是另一个EM元素的父元素。相反地，一个元素被另一个元素所包含，那么它就是子元素。这样，图中的链接元素就是STRONG元素的子元素，STRONG又是段落元素的子元素，依次类推。

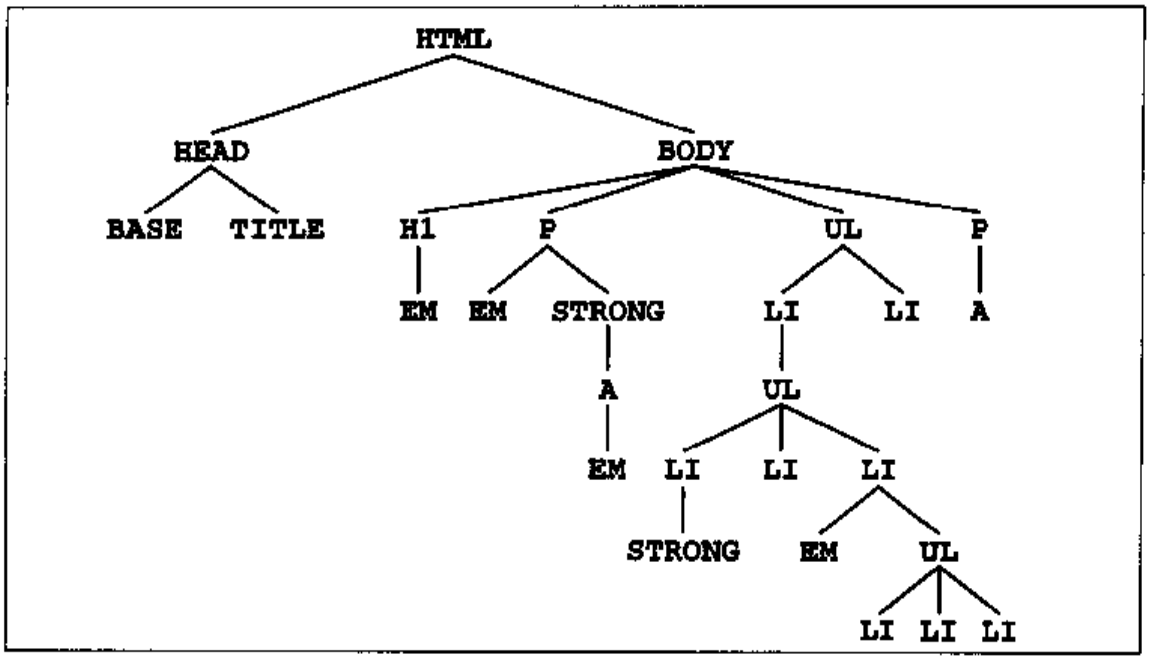


图2-18 一个简单的HTML文档的“树形视图”

父和子的术语常常被一般化为祖先和后代这两个术语。但也有区别：在树形图中，如果一个元素刚好处于另一个元素的上面一层，那么它们就是父子关系。（当然，子元素也是一个后代，父元素也是一个祖先。）但是，如果从一个元素到另一个元素的路径上跨越了两个或更多层次，那么它们就具有祖先-后代关系。在图2-18中，第一个UL元素是两个LI元素的父元素，但它也是第二个UL元素下所有元素的祖先元素，一直到最深一层的LI元素。

继续观察图2-18，可以看到定位锚元素是STRONG的子元素，同时也是段落元素的后代，还是BODY和HTML元素的后代。BODY元素是所有浏览器能显示的元素的祖先，当然HTML元素是所有元素的祖先。基于这个原因，HTML元素也被称为根元素。

上下文选择符

我们能从这种模型中得到的第一个好处就是定义上下文选择符（*contextual selector*）的能力，也就是在某种特定的结构环境下创建运行规则的行为。比如，想要设置EM文本为灰色，但它必须是在H1元素中才符合条件。那么可以在所有H1中的EM元素里放置一个类属性，但这几乎和使用FONT标签一样繁琐。显然，为符合上述条件的EM元素声明一条规则会更好。

于是可以如下书写规则：

```
H1 EM {color: gray;}
```

这条规则就可以使所有属于H1元素后代的EM元素的文本显示为灰色。其他的EM文本，如段落或引用块中的，都不会匹配此规则，如图2-19所示。

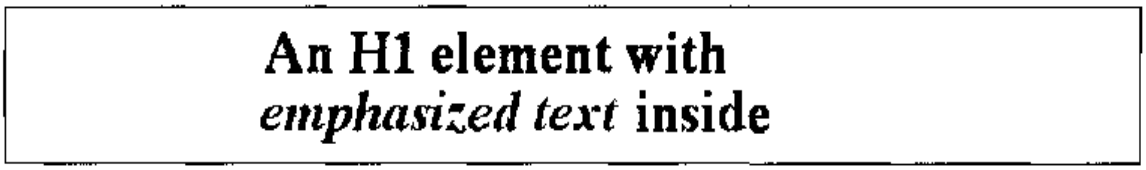


图2-19 根据其上下文选择元素

在一个上下文选择符里，规则的选择符方是由两个或更多个以空格符分隔的选择符组成的。每个空格可以翻译成“在…中发现的”、“是…的一部分”或者“它是…的后代”，但是前提是按从后至前的顺序读选择符。H1 EM可以译作“任何EM元素，它是H1元素的后代”（如果坚持要从前往后读，可以译作，“任何H1包含的EM元素”）。

当然不仅限于两个选择符。例如：

```
UL OL UL EM {color: gray;}
```

在这种情况下，如图2-20所示，任何无序列表下的有序列表下的无序列表内的强调文本将以灰色显示。很明显，这是一个特殊的选择循环。

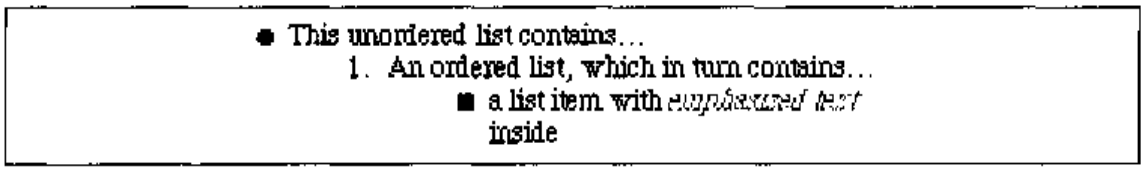


图2-20 一个具体的上下文选择符

上下文选择符的功能非常强大。它们使在HTML中不能做到的成为可能——至少不再需要繁琐的FONT标签。让我们考虑一个普通的例子。假设某个文档有个边栏和一片主区域。边栏是蓝色背景，主区域为白色背景。在边栏中有一系列链接，在主区域内也有链接，但不能将链接设成蓝色，因为边栏的背景也为蓝色（蓝色背景上的蓝色文本是毫无意义的）。

怎么办呢？上下文选择符。在这种情况下，可以为包含边栏的表格单元设置一个sidebar的类，为主区域设置一个main-page的类。然后像下面这样书写样式：

```
TD.sidebar {background: blue;}
TD.main-page {background: white;}
TD.sidebar A:link {color: white;}
TD.main-page A:link {color: blue;}
```

图2-21显示了其结果：蓝色背景的边栏中是白色的链接，白色背景的主区域中是蓝色的链接。

图2-21 使用上下文选择符为同一类型元素应用不同的样式

下面是另一个例子：要让BLOCKQUOTE元素中的B元素及普通段落中的粗体文本都显示为灰色：

```
BLOCKQUOTE B, P B {color: gray;}
```

结果如图2-22所示，段落和引用的粗体字为灰色，而列表中的粗体文本不是灰色。

注意：庆幸的是，上下文选择符的运用很容易上手——尽管在Navigator 4.x中会出现一些小的随机的怪现象。但是通常情况下，像分组、上下文选择符还是非常安全、稳定的操作。

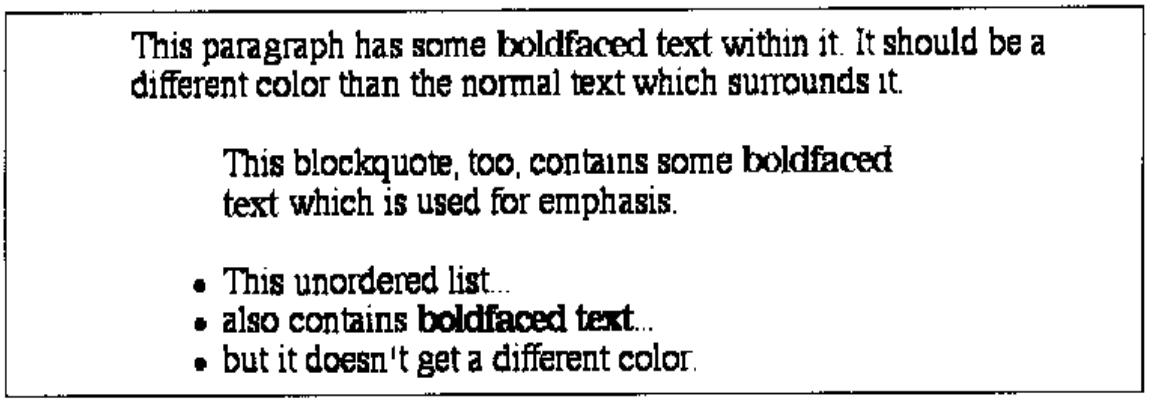


图2-22 上下文与样式

继承

将文档视为一棵树是很重要的，其中一个原因就是：CSS的一个主要特征就是继承（*inheritance*），它是依赖于祖先－后代关系的。继承其实是一种机制，它允许样式不仅应用于某个特定的元素，而且应用于其后代。例如，如果某种颜色应用于H1元素，那么这一颜色不仅会应用于所有H1中的文本，而且会应用于H1元素的所有子元素中的文本。

```
H1 {color: gray;}

<H1>Meerkat <EM>Central</EM></H1>
```

从图2-23中可见，H1文本和EM文本都是灰色的，因为颜色值被EM元素所继承了。这很符合制作者的意图，也是为什么继承是CSS的一部分的原因。

图2-23 继承样式

如果在某种情况下，继承不能正常工作，那么EM文本就会是黑色而不是灰色了。

另一个说明继承是如何工作的例子就是无序列表。假设要将color:gray这样一个样式应用于UL元素。我们所希望的是，这种样式不仅要应用于UL，而且包含其列表项，以及列表项目的所有内容。有了继承，很容易就能实现上面的要求，如图2-24所示：

```
UL {color: gray;}
```

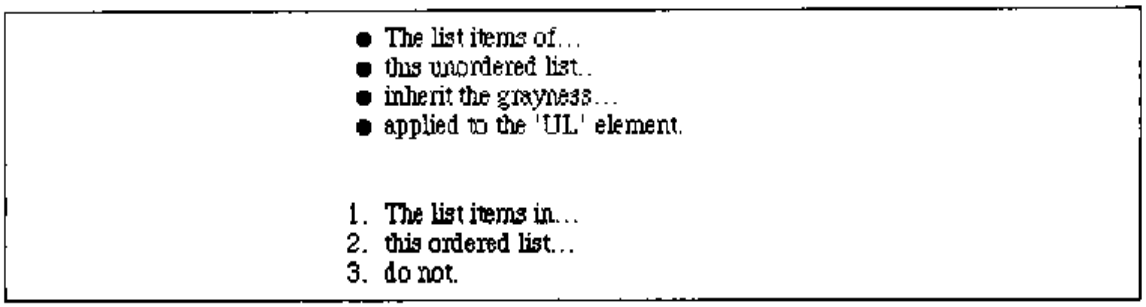


图 2-24 继承样式

可见，在CSS中，继承是一种非常基本而且很自然的行为，我们甚至不需要考虑是否能够这样去做——正如我们想当然地利用高速公路给我们带来的便利一样。然而，对于继承也有一些值得注意的地方。

继承的局限性

对任何完美的事物，当凑得足够近再去观察它时，总能发现一些瑕疵。首先，有些属性是不能继承的。这没有任何原因，仅仅是凭直觉。例如，boder 属性（用于设置元素的边框，很奇怪吧）就不能继承。看一看图2-25就知道为什么会这样。如果继承了边框，那么文档看起来就会很杂乱，除非作者再采取另外的措施关掉边框的继承。

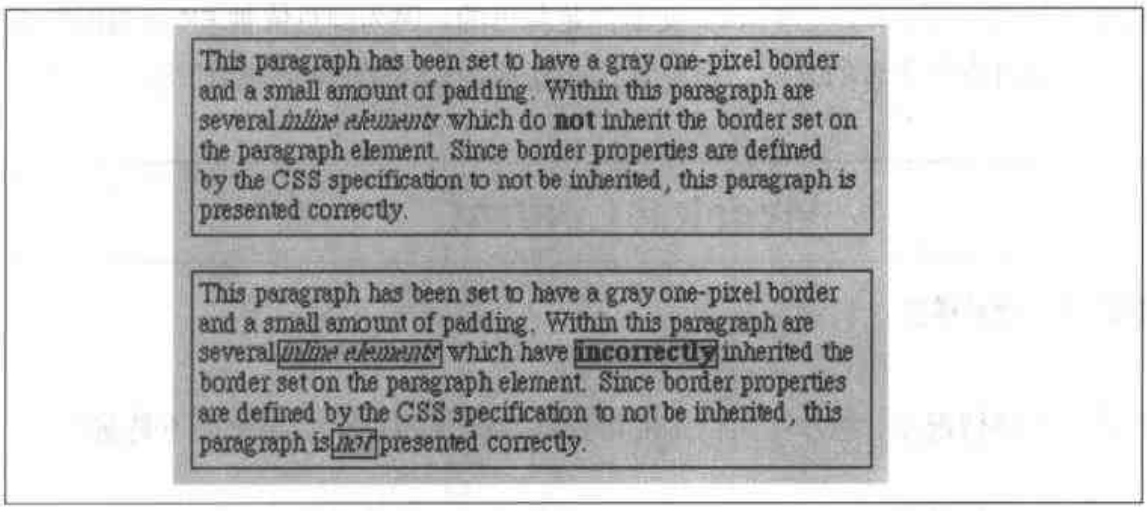


图 2-25 为什么边框不能继承

正如图中所示，多数边框类属性，包括边界、补白、背景和边框，都不能继承。

继承错误

不幸的是，继承也可能带来麻烦。多亏在浏览器实现时出现的问题，使制作者不可能在任何情形下都依赖于继承。例如，Navigator 4（以及IE4）不能在表格中继承样式。下面这条规则会使除表格之外的文本变成紫色：

```
BODY {color: purple;}
```

或许这并不一定正确，因为Navigator 4的一些版本在表格显示之后就会完全忘掉样式；所以表格之后的文本以及表格中的文本都不会变成紫色。从技术上来说，这是不正确的，但它的确存在，因此常常需要借助于某些技巧，例如：

```
BODY, TABLE, TH, TD {color: purple;}
```

这也只是很可能，而不是绝对会得到预期的效果。除去这些潜在的问题，其余的文本都会完全按照继承的机制来显示。而继承为什么会出现这样的情况呢？如果要将其原因和出错的方式写成完整的文档，那么就可以写一本书了。

特殊性

有了继承，读者可能想知道在下面这种情形下会发生什么：

```
.grape {color: purple;}
H1 {color: red;}

<H1 CLASS="grape">Meerkat <EM>Central</EM></H1>
```

既然选择符H1和.grape都能匹配上面的H1元素，那么到底应该使用哪一个呢？.grape是正确的答案，因此H1元素将会显示为紫色。这是因为两条规则的特殊性不一样，CSS规则必须处理这样的情形。

特殊性（*specificity*）描述了不同规则的相对权重（weight）。根据规范，一个简单的选择符（比如H1）具有特殊性1，类选择符具有特殊性10，而ID选择符具有特殊性100。下面规则的特殊性在其注释中说明：

```
H1 {color: red;}                    /* specificity = 1 */
P EM {color: purple;}               /* specificity = 2 */
.grape {color: purple;}             /* specificity = 10 */
P.bright {color: yellow;}           /* specificity = 11 */
P.bright EM.dark {color: brown;}    /* specificity = 22 */
#id216 {color: blue;}               /* specificity = 100 */
```

规则#id216具有更高的特殊性，因而更高的权重。当有多个规则都能应用于同一元素时，权重越高的样式将被优先采用。

继承和特殊性

在特殊性的框架下，被继承的值具有特殊性0，这就意味着任何显式声明的规则将会覆盖其继承样式。因此，不管一条规则具有多高的权重，如果没有其他规则能应用于这个继承元素，那么它也只是个被继承的规则而已。

例如，考虑下面的例子：

```
BODY {background: black;}
LI {color: gray;}
UL.vital {color: white;}
```

读者可能希望看到的是，除包含vital类的列表项显示为白色外，其余所有的列表项都应是灰色。然而，正如图2-26所示，并非如此。

图2-26 明显的不正确行为

为什么会这样呢？因为带选择符LI的显式声明的权值比从UL.vital规则那里继承过来的权值大。

下面再来仔细看看这个过程。若给定下列的标记，则强调文本将会是灰色的，而非黑色，因为EM规则的权值要大于从H1元素继承来的权值：

```
H1#id3 {color: black;}    /* specificity = 101 */
EM {color: gray;}         /* specificity = 1 */

<H1 ID="id3">Meerkat <EM>Central</EM></H1>
```

这是因为第二条规则的特殊性（1）比被继承的特殊性的值（0）要大。事实上规则H1#id3的原始特殊性101对其继承值没有影响，仍然为0。

如果想让H1始终为黑色，而EM文本在其他情况下为红色，那么下面的代码应该是个比较好的解决方法：

```
H1, H1 EM {color:black;}        /* specificity = 1, 2 */
EM  {color:red;}                /* specificity = 1*/
```

给定这些规则后，除在H1元素内的任何EM文本都是红色的。然而H1元素内的EM文本为黑色。由于其选择符分组，在第一条规则中就有两条有效的规则（一条是对H1的，另一条是对H1 EM的），也就有两个特殊性——每条规则一个。

带有STYLE的元素在CSS1下其权值定义为100，尽管是#id3这样的ID选择符也一样。实际工作中，这一特殊性的权值会更高一点，因为STYLE元素的值看起来要比多数普通规则的权值大，即使从技术角度来说它们具有更高的特殊性（例如H1#id3 EM）。换句话说，下列标记将会产生如图2-27所示的结果：

```
H1#id3 EM {color: gray;}

<H1 ID="id3">Meerkat <EM STYLE="color: black;">Central</EM>!</H1>
```

图2-27 内联样式具有高的特殊性

也可以让STYLE具有特殊性值1000，尽管CSS规范不支持这样的值，它也不值得信赖。最后，当计算特殊性时，伪元素将被忽略不计，但伪类被当作常规的类对待。

在特殊性方面，还有另外一种方式，用以覆盖整个特殊性机制。

重要性

曾经想过某种规则非常重要，以致于其权值比其他规则都要大这样一种情形吗？CSS就可以将某些规则标记为比其他的更重要。它们被称为重要规则（*important rule*）。这是根据其声明的方式和它们的自然属性而命名的。通过在一条规则的分号前插入!importan这样一个短语来标记一条重要规则：

```
P.dark {color: #333 !important; background: white;}
```

这里颜色值#333被标记为!important，而背景色white未被标记。如果希望标记二条规则都为重要的，那么每条规则都需要有自己的!important：

```
P.dark {color: #333 !important; background: white !important;}
```

正确地放置!important很重要，否则整个规则将无效。!important总是放在声明的最后，在分号之前。尤其是当规则中包含多个关键字的属性时，正确的位置显得更加重要：

```
P.light {color: yellow; font: 11pt Times !important;}
```

如果!important放在字体声明的其他任何地方，那么整个声明就很有可能会作废，进而没有样式可以应用了。

标记为!important的规则不必有特殊性值，不过可以假定它们具有很高的特殊性值，比如说10000——换句话说，是比其他的权值都要大的值。注意，虽然制作者定义的样式比用户定义的样式具有更高权值时（见本章的“层叠”一节），但!important规则恰好相反：重要的用户定义规则要比制作者定义的样式具有更高权值，即使是标记为!important的重要规则也一样。

!important规则会覆盖内联STYLE属性的内容。因此，给定下面一段代码，结果会是灰色文本而不是黑色：

```
H1 {color: gray !important;}

<H1 STYLE="color: black;">Hi there!</H1>
```

还有最后一种需考虑的情形。如下所示：

```
P#warn {color: red ! important;}
EM {color: black;}

<P ID="warn">This text is red, but <EM>emphasized text is black.</EM></P>
```

记住，继承值总是具有特殊性值0。即使是从带有！important的规则继承的值也同是一样。在匹配重要规则的元素之外，重要性也随之消失。

警告：当写本书时，很少有浏览器实现了!important。Internet Explorer 5和Opera 3.6可以允许有!important，但也仅此而已。另外，!important将在Navigator 6里得到支持。

层叠

还有一个重要的问题：当两条具有同样特殊性的规则应用于同一元素时，情况又如何呢？浏览器又如何解决这一冲突呢？

例如，有下面两条规则：

```
H1 {color: red;}
H1 {color: blue;}
```

那么到底哪条规则会胜出呢？它们都有特殊性值1，因此它们的权值相等，都应该被应用。当然不是这样，因为元素不可能既是红色又是蓝色，必须二选一。但是该选哪个呢？

还是让“层叠样式表”来解决吧。CSS是基于样式层叠这样一种方法的——这可以通过规则的继承和特殊性来实现。层叠规则简单：

1. 找出所有包含与给定元素匹配的选择符的声明。
2. 按应用于给定元素的所有声明的显式权重排序。标记为!important的规则具有更高的权值。然后再按声明的起源排序。有三种起源：制作者、读者和用户代理。在正常环境下，制作者样式优先于读者样式。在CSS1里，!important制作者样式优先于重要的读者样式，但在CSS2中，!important读者样式优先于其他样式。制作者或读者样式将覆盖用户代理样式。

3. 按应用于给定元素的所有声明的特殊性排序。高特殊性元素比低特殊性元素具有更大的权值。

4. 按应用于给定元素的所有声明的出现顺序排序。样式表或文档中越靠后的声明其权值越大。被引入样式表中的所有声明排在样式表中引入它们的所有声明之前，而 STYLE 属性中的声明排在文档的嵌入样式表中的声明之后。

为了更清楚地表达上面的意思，下面用例子来说明上面四步的具体含义。第一步很简单：找出所有的选择符。

在第二步中，如果两条规则同时应用于一个元素，而且有一个标记为 !important，那么这条规则优先。如下：

```
P {color: gray !important;}

<P STYLE="color: black;">Well, <EM>hello</EM> there!</P>
```

尽管在段落的 STYLE 属性中有一个颜色值，但 !important 规则仍然优先，所以段落显示为灰色，如图 2-28 所示，同时 EM 元素也继承了灰色。

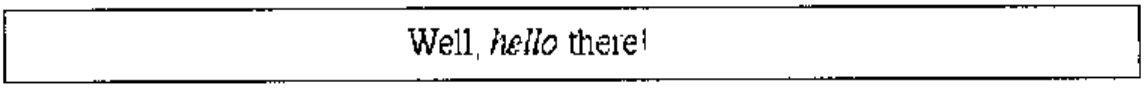


图 2-28 通过重要性排序样式

此外，再来考虑规则的起源。如果某个元素同时匹配制作者和读者的样式表，那么制作者的样式将被采用。例如，假设下面的样式来自注释中的起源：

```
P EM {color: black;}      /* author's style sheet */

P EM {color: yellow;}     /* reader's style sheet */
```

其结果图 2-29 所示，强调文本为黑色。

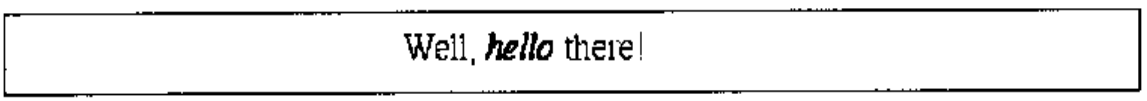


图 2-29 按起源排序样式

浏览器的缺省样式——通常由其设置而定——在这一步中也有影响。浏览器的缺省样式是所有样式中最不具有影响力的。当然制作者定义的样式会覆盖浏览器的缺省样式。如果读者想执行某些规则，那么必须在本地样式表中定义，而且声明它们为!important，这样它们才能具有高的权值。这是因为重要性使得读者定义的样式覆盖了制作者和浏览器的样式。

根据第三步，如果多个规则应用于一个元素，那么它们应该按特殊性排序，最特殊的规则优先。例如：

```
P#bright {color: silver;}
P EM {color: gray;}
P {color: black;}

<P ID="bright">Well, <EM>hello</EM> there!</P>
```

正如我们从图2-30中可以看到的那样，除EM文本为灰色外，段落文本的颜色为银色。为什么呢？因为P#bright的特殊性（101）高于P的特殊性（1），即使它们排在P#bright规则的后面。

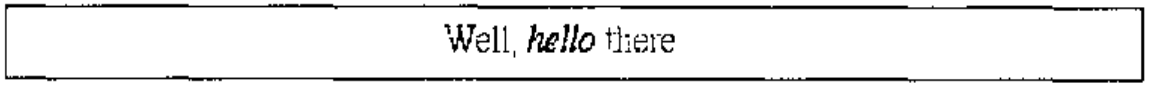


图2-30 按特殊性排序样式

然而，EM文本并没有继承silver值，因为它有一条显式的规则与它关联，而且这条规则的特殊性（2）覆盖了其继承的值。

这点比较重要。例如，假设有一个样式表使得工具栏里的文本为黑底白字：

```
#toolbar {color: white; background: black;}
```

只要ID为toolbar的元素内含有纯文本，那么就会显示黑底白字。然而当这个元素内的文本为超链接（A元素）时，用户代理的超链接样式就会被应用——尽管它们是被引入的样式，但它们也是显式的样式分配，因此会覆盖继承值。通常，超链接为蓝色，因为浏览器样式表一般都包含下列的规则：

```
A:link {color: blue;}
```

为了解决这个问题，制作者需要声明：

```
#toolbar A:link {color: white; background: black;}
```

直接对工具栏的A元素应用此规则，其结果如图2-31所示。

图2-31 用上下文选择符取消特殊性

最后，第四步，如果两条规则具有相同的权值、起源和特殊性，那么在样式表中最后出现的规则优先。让我们再回到前面的例子，在文档的样式表中包含下面两条规则：

```
H1 {color: red;}
H1 {color: blue;}
```

则文档中所有H1元素的值都会为蓝色，而不是红色。

任何位于文档中的规则都要比引入的规则的权值大，因而优先。如果规则是文档样式表的一部分，但不是元素STYLE属性的一部分，上面的原则同样适用。如下所示：

```
P EM {color: purple;}     /* from imported style sheet */

P EM {color: gray;}       /* rule contained within the document */
```

在这种情况下，因为第二条规则是文档样式表的一部分，它将覆盖引入的规则。

元素的STYLE属性所给出的样式被认为处在文档样式表的末尾，因而处于其他规则的后面。而且，STYLE属性和ID选择符具有相同的权值。这样，下面的代码就会产生黑色文本，如图2-32所示：

```
#hello {color: red;}

<P ID="hello" STYLE="color: black;">Hello there!</P>
```

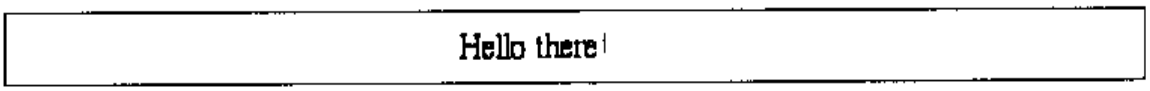


图2-32 按位置来排序样式

这里，内联样式的权值（100）与规则 #hello的权值（100）相等，因此靠后的规则将胜出。

元素分类

在前面，我们已经以层次图的形式讨论了文档中的元素。在最基本的层次上，块级元素包含其他的块级元素、内联元素和替换元素。这种层次模式同时也要依赖于这些元素的类型之间的关系；例如，内联元素可以是块级元素的子元素，但反过来就不成立了。

在CSS中，元素被分为三种类型:

块级元素

诸如段落、标题、列表、表格、DIV和BODY等元素都是块级（block-level）元素。图像和表单输入这样的替换元素，可以看作是块级元素，但通常都不这样看。每个块级元素都从一个新行开始显示，而且其后的元素也需另起一行进行显示。块级元素只能作为其他块级元素的子元素，而且需要一定的条件。

内联元素

如A、EM，SPAN元素及大多数的替换元素，如图像和表单输入元素。它们不必在新行上显示，也不强迫其他元素在新行上显示，而且可以作为任何其他元素的子元素。

列表项元素

在HTML中只包含LI元素。它们类似于书签，用于特殊的表示场合（如圆点、字母或数字）。如果它们出现在某种有序列表中，则具有顺序性。因此在有序列表中的列表项能依据它们的上下文自动编号。

这几种元素占据了display属性的四个值中的三个。

display

允许值	block | inline | list-item | none
初始值	block
可否继承	否
适用于	所有元素

不像其他的CSS元素，display一般不会取缺省值。给定元素的display值是在文档类型定义（document type definition，注1）DTD中定义的。所以在HTML里，H1和P元素都被定义成块级元素了。另一方面，A和EM是内联元素，LI是一个列表项元素。因此，这些元素的缺省的display值为：

```
H1, P {display: block;}
A, EM {display: inline;}
LI {display: list-item;}
```

替换元素是依据它们的上下文和在文档流中所处的位置来决定的。例如，一个浮动的图像通常被认为是块级元素，但通常图像都是内联元素。

理论上，display可能会扰乱标记语言的结构定义。在传统的HTML里，段落间总是有空白空间，而且两个段落不能出现在“同一行”。但这可以通过下面的规则来改变：

```
P {display: inline;}
```

用这样一条简单的声明，P元素就同SPAN元素没什么区别了。如果将其应用于文档中的几个段落，它们就会杂乱地挤在一起。如图2-33所示。

注1：文档类型定义（DTD）是对标记语言的正式描述。DTD提供一种严密的方式以定义元素的含义和它们在语言中所处的层次。它类似于用名词、动词、副词等来描述英语，但另外描述了各个部分的含义和它们之间的关联关系。若未经培训，DTD是不容易阅读的。

This is a paragraph, which is ordinarily block-level. This is a second paragraph. As you can see the paragraphs in this document are no longer block-level. Here's the third paragraph. The setting of `inline` for the `display` property has made the paragraphs all inline elements, which is why they're "running together." This is the fourth paragraph. There really are <P> elements in this document. The elements are still here, and the structure of the document is preserved. It's just that the display is not quite what we're used to.

图2-33 内联段落

注意，应用于段落的样式，如字体或颜色仍然有效。只有那些只应用于块级元素的属性在内联段落里才会无效。

将普通的内联元素换成块级元素会得到相反的效果。假如想让文档中所有的图像元素都显示在它们各自的行上，只需下面这条规则就可以得到如图2-34所示的效果：

```
IMG {display: block;}
```

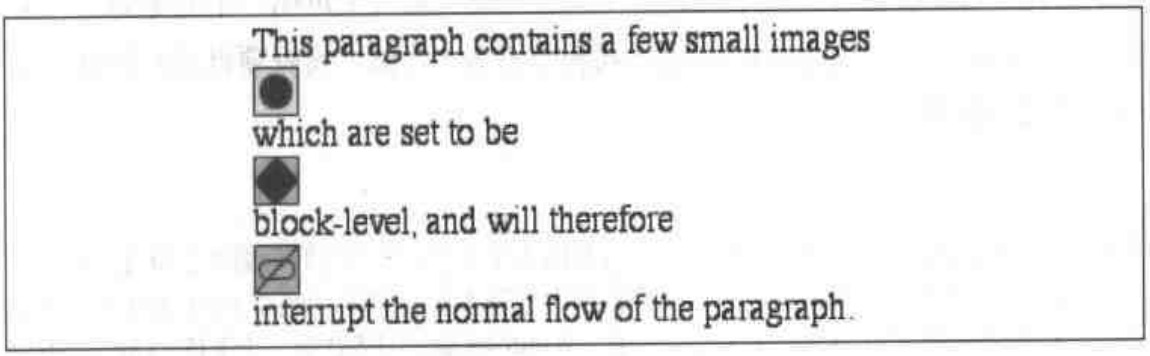


图2-34 块级图像

如果要想去掉文档中的所有图像，则可以使它们不显示：

```
IMG {display: none;}
```

如果将元素的display设成none，则元素的存在会被浏览器所忽略，如图2-35所示。

而且不只是不被显示，它所占用的空间也会被忽略。即使它在源文档中仍然存在，但在显示文档时就当它不存在一样：

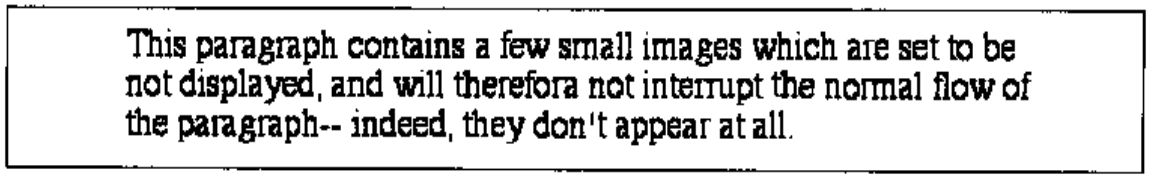


图 2-35 用 display: none 来禁止图像的显示

```
<P>This is the first paragraph in the document.</P>
<P STYLE="display: none;">This will not be displayed,
nor will it affect the layout of the document.</P>
<P>This is another paragraph in the document.</P>
```

下面的标记可以用来在非 CSS 的浏览器中显示一些小的警告:

```
<P STYLE="display: none;">This page was designed with CSS, and
looks best in a CSS-aware browser--which, unfortunately, yours is not.
However, the document should still be perfectly readable, since that's
one of the advantages of using CSS.</P>
```

这个警告只能在一个不认识display:none的浏览器中才可见 —— 换句话说，即为不支持 CSS 的浏览器。

对于display，还有另外一个值，即 list-item。它用于声明一个元素是一个列表项。理论上，这会使元素的行为同列表差不多。当然，列表项也已经将这一显示类型设为缺省了。

注意: 当 display 取 none 值时，多数浏览器都能正常工作，因为它们只需要忽略这一显示属性即可。（毕竟，销售商不会宣称他们的浏览器不支持 CSS。）所以 none 成了在所有支持 CSS 的浏览器中都能正常工作的值。Windows 版的 IE5 是第一个支持 display 取值的浏览器。尽管它不是很完美：好像在处理 list-item 时有点错。

为什么显示属性会存在?

值得注意的是，显示属性可能会完全扰乱 HTML 文档结构。下面就来看一看这个样式表造成的破坏:

```
H1, H3, P, DIV {display: inline;}
IMG, B, STRONG, EM, A {display: block;}
```

```
A:link {display: none;}
H2, I, TABLE {display: list-item;}
```

用当前的浏览器来描述这些样式是不可能的。那么为什么还要使用这个属性呢？首先，display:none非常有用。假如浏览器支持可选择的样式表，那么就可以定义一个称为“No Images”的样式表，其中设置这样一条规则：

```
IMG {display: none;}
```

如前所述，这条规则会去掉所有的图像显示。

更值得一提的是，当CSS链接到一个XML文档时，display属性是很有用的。XML不包含任何元素信息，因为它没有预定义的元素类型。因此，当写完一个XML文档后，可以为之写一个CSS样式表以描述哪些元素是块级，哪些是内联等等。如果没有这样一个样式表，浏览器就不知道哪些XML元素会作为列表项显示，哪些又作为块级元素显示等等。

CSS1不能描述表格部分，例如单元或者行。因此表格的格式不能用CSS1来描述。和表格相关的显示值在CSS2中被引入，在第十章“CSS2展望”中对此有简要的叙述。

小结

运用选择符，可以很轻易地创建CSS规则以应用于大量的相似元素，而且还能针对很苛刻的环境创建其应用规则。对选择符和声明分组的能力使样式表更紧凑、更灵活，从而使文件更小，下载速度更快。使用继承和层叠，便能够创建连锁的规则集合，使文档样式更灵活。所有这些都归功于CSS与文档结构的紧密结合。因为它使用这些结构来决定规则如何应用，哪些规则该应用，哪些元素该从其祖先元素那里继承样式等等。

选择符是用户代理必须支持的，因为如果不能正确地解释选择符就不能使用CSS，然而层叠和继承却不一样，而且在它们的实现上也存在一些缺陷。例如，Navigator 4.x对于表格的继承和列表这样的结构就支持得不好，但可以运用非常严格的HTML来清除许多这样的问题。

还有许多CSS规则用于描述长度、颜色及其他用特定的单位和值才能表达的属性——而这正是下一章的主题。

第三章

单位和值

我们将在本章研究几乎所有CSS属性的基础：影响颜色和距离的单位（unit）。没有单位，就不可能声明一个段落为紫色，或一个图像具有十个像素的边界，或具有某个尺寸的标题。通过对本章概念的理解，能够帮助我们更快地学会使用CSS的其他特性。

然而，本章也包含许多试图防止错误发生的说明、警告和对浏览器错误以及操作系统之间的不一致性的讨论。CSS并不希望成为很精确的布局语言——而且，本章中讨论的许多问题并不是CSS造成的错误，而是任何人在使用计算机时都会遇到的一些基本的问题。因此，读者一旦完成了本章的学习，不但可以很好地掌握CSS的单位，而且或许还能解决以前未曾明白的一些问题。

不管前面的路有多难，最重要的是坚持，坚持就是胜利。

颜色

几乎所有网页初学者都想知道这样一件事，“我该怎样为网页设置颜色？”在HTML里，有两种选择：使用有限的一些颜色名，像红色或者紫色；或者使用十六进制的颜色值。这两种描述颜色的方法在CSS中都能找到，而且CSS还有其他一些较为复杂的方法。

命名颜色

假设读者喜欢选择数量较少的、基本的颜色，那么最简单的方法就是直接使用颜色名字。它们就是下面要讨论的命名颜色。

与浏览器公司宣称的相反，其实只有一定数量的命名颜色是可用的。例如，“mother-of-pearl”就不可用，因为它并未定义。从技术角度说，根本就没有真正定义的颜色，但规范中推荐了16种颜色，而且主要的一些浏览器都能识别它们：

aqua（水绿）	gray（灰）	navy（深蓝）	silver（银）
black（黑）	green（绿）	olive（橄榄）	teal（深青）
blue（蓝）	lime（浅绿）	purple（紫）	white（白）
fuchsia（紫红）	maroon（褐）	red（红）	yellow（黄）

的确，有一些颜色名很怪异。那么它们是从哪里来的呢？这些颜色最初来源于16种基本的Windows VGA颜色，而且浏览器都能产生这样的颜色。或许它们是一个杂乱的颜色集合，但这就是我们所能使用的。

假如想让所有第一级标题为褐色。最好的声明是：

```
H1 {color: maroon;}
```

简单、直接，而且不易忘记，但也仅此而已。下面是更多的例子：

```
H1 {color: gray;}
H2 {color: silver;}
H3 {color: black;}
```

当然，读者或许见到过（或使用过）除上面的颜色之外的颜色，例如：

```
H1 {color: orange;}
```

尽管orange不在上面命名的颜色之列，但很可能会看到所有的H1元素都变为橙色。这是因为多数浏览器都能认识大约140种颜色名，其中包括16种标准色。但运用其他的颜色时有两个问题。第一就是并不是所有的浏览器都能识别它们。例如，Opera仍然坚持使用16种颜色，至少Opera 3.x还是这样的。但并不能说这就是个失败，因为这代表了对标准的严格支持，尽管它可能会迷惑或干扰许多网页设计人员。

第二个问题更为重要：这些其他的颜色名字没有统一标准的颜色值。声明一个元素为orange并不意味着所有的浏览器，或者不同平台上运行的同一种浏览器会产生同一颜色值的橙色。而对于16种标准色，至少有可能使它们尽量相似，因为毕竟它们是已定义的颜色值。除此之外，别无他求。浏览器可以为同一颜色名实现相似的色调，也可能不，其区别对人眼来说可能是很难察觉的，也可能是很明显的，或者根本就是不和谐的。

颜色的再生

颜色的再生是个很重要的问题。我们马上就会看到，所有的颜色都可以用某种固定的方式来指定，这看上去好像解决了不同浏览器显示同一颜色的问题。实际上，情况要比这复杂得多。首先，人的视觉是相对的。同一颜色在不同监视器上显示，可能由于光线、亮度、邻色和其他因素的改变而改变。读者可以通过改变电脑的桌面背景来观察这种颜色效果的变化，桌面上的图标颜色会随之发生轻微的变化。

像监视器这样的显示设备通常都有一个缺省的灰度值集合，这是调整颜色的一个重要因素。灰度通常是由操作系统设定的，尽管许多昂贵的监视器可能有自身的灰度设置。问题是不同的系统会有不同的灰度值。因此，如果为网页做一个彩色的背景，在同样的光线条件下分别在Windows和Macintosh机器上显示，其颜色会有所区别。对于网页图片也有同样的问题，在Windows中产生的图片在Macintosh上可能会显得暗一些，而在Macintosh上产生的图像在Windows里显示会变得更亮。

在打印颜色时，情况会更糟，因为打印机的分辨率、纸张的颜色，甚至是打印时的温度都会对打印的颜色产生影响。

所以，要想对文档的外观进行彻底的控制是不可能的。它是由不一样的操作系统设置和奇特的人类视觉而引起的客观反映，并不是计算机本身所能克服的障碍。

如果想使用命名的颜色，那么制作者自己得决定采取什么样的措施，但至少就16种标准颜色来说，还是有希望让它们显得比较一致。

这就是确定颜色的最简易方法——看起来虽有些危险，但不失为一个可行的方法。另外的四种方法要更为复杂一些。但使用这些方法的好处在于，可以在8位光谱里指定任意的颜色，而不只是16种命名颜色。这是利用计算机产生颜色的机理来实现的。

RGB颜色

计算机通过组合不同级别的红、绿和蓝色来产生新的颜色，这就是通常所说的*RGB*颜色。事实上，如果将计算机的监视器，或者电视机打开后，仔细观察射线管，就会发现有三个“电子枪”。(记住，在观察这些“电子枪”时注意安全。)这些电子枪射出不同亮度的光线，打到荧光屏上的某一点，而这一点就是RGB颜色中的一束。三条不同的光束同时打到一点，它们的亮度一经组合就形成了人们在屏幕上所看到的颜色。这一个点叫做一个像素，这是我们在后面要提到的一个术语。

有了这样一个产生颜色的机制，就可以通过直接给出不同的颜色级来确定颜色，即自己去组合三种光束的值。显然，这种方法有点复杂，但却是值得的，因为我们可以不再受限于命名颜色所带来的不便了。

百分比颜色

有四种方法可定义RGB颜色，首先介绍最易掌握的一种，它使用百分比。如下面的例子:

```
rgb(100%,100%,100%)
```

这一声明将红、蓝、绿都设到最大值，其组合值为白色。为了产生黑色——所有三个都设成0%。下面是更多的声明:

```
H1 {color: rgb(0%,0%,0%);}          /*black*/
H2 {color: rgb(50%,50%,50%);}       /*medium gray*/
H3 {color: rgb(25%,66%,40%);}
```

设定百分比颜色值的语法为:

```
rgb(color)
```

这里，color有两种方法来指定。第一种就是用百分比，另一种用数字，后一种方法将在后面讨论。

如果想让H1元素显示为红色和褐色之间的某种红色。因为红色为rgb(100%,0%,0%)，褐色为(50%,0%,0%)，所以可以这样来设定上面所需的颜色：

```
H1 {color: rgb(75%,0%,0%);}
```

这使得所取的红色值比褐色更亮，但比红色更暗一些。另外，如想要一种更淡的红色，那么可以增加其他两个值：

```
H1 {color: rgb(75%,50%,50%);}
```

最直观的方法就是创建一个灰度值表，而且灰度打印也是我们本书仅能够提供的印刷方式：

```
P.one {color: rgb(0%,0%,0%);}
P.two {color: rgb(20%,20%,20%);}
P.three {color: rgb(60%,60%,60%);}
P.four {color: rgb(80%,80%,80%);}
P.five {color: rgb(100%,100%,100%);}
```

图3-1显示了不同百分比的效果。

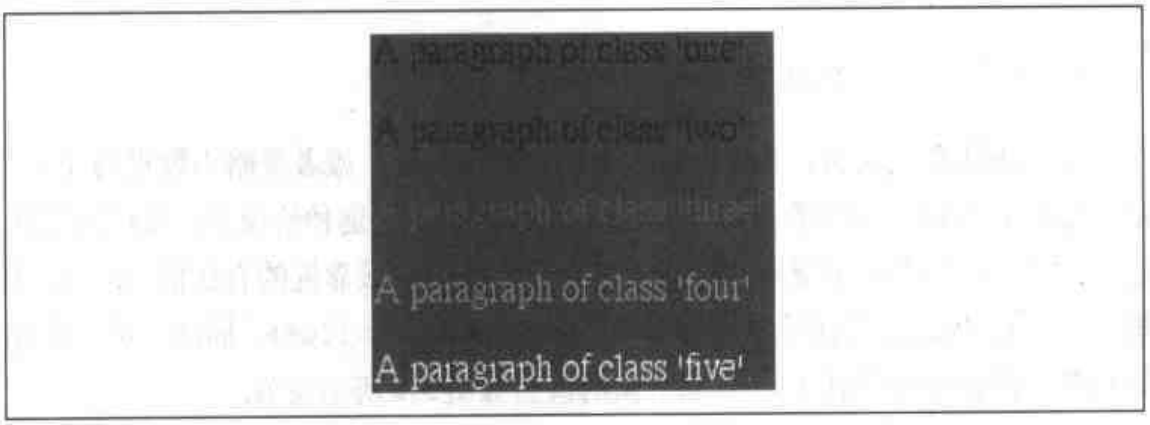


图3-1　灰度级

由于我们处理的是灰色调，每个语句中的百分比取值都一样。如果其中某个百分比不同，那么就会出现不同的颜色。例如，rgb(50%,50%,50%)被改成rgb(50%,50%,60%)后，其结果会是带一点儿蓝色的中灰度。

表 3-1 是百分比颜色的一个对照表。

表 3-1 普通颜色的百分比 RGB 颜色值

颜色名	对应的百分比值
红色	rgb(100%,0%,0%)
橙色	rgb(100%,40%,0%)
黄色	rgb(100%,100%,0%)
绿色	rgb(0%,100%,0%)
蓝色	rgb(0%,0%,100%)
靛蓝色	rgb(20%,0%,100%)
紫罗兰色	rgb(80%,0%,100%)
中灰色	rgb(50%,50%,50%)
深灰色	rgb(20%,20%,20%)
棕褐色	rgb(100%,80%,60%)
金黄色	rgb(100%,80%,0%)
紫色	rgb(100%,0%,100%)

也可以使用小数值。例如某种颜色由25.5%红色，40%绿色，和98.6%蓝色组成。这都不成问题:

```
H2 {color: rgb(25.5%,40%,98.6%);}
```

实际上，还是有问题的。某些浏览器不能识别小数值，或者忽略小数点的存在，错误地将上面的一句翻译成 rgb(255%,40%,986%)。在这种情况下，假设浏览器能正常工作，那么超出范围的值会被“修剪”成与之最靠近的合法值 —— 在这里为100%。所以最后浏览器的解释是 rgb(100%, 40%, 100%)。同理，对于所有的负数，都将被修剪成 0%。例如下面的值会如图 3-2 所示修剪:

```
P.one {color: rgb(300%,4200%,110%);}
P.two {color: rgb(0%,-40%,-5000%);}
```

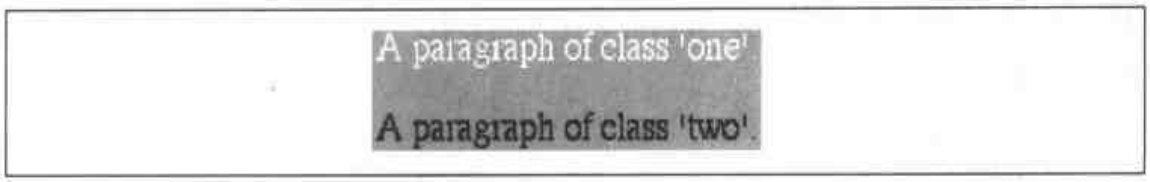


图 3-2　超出范围的值将被修剪

数字颜色

与百分比颜色相关的另一种方法是采用数字方法。数字范围从 0 到 255，即 rgb(0,0,0)代表黑色，而 rgb(255,255,255)表示白色。可能多数读者都会想到这些数字的另一起源：8 位二进制数的十进字表示。如果不知道这一点，那么只需知道在计算机里都是使用二进制（开／关）数，包括数字和颜色的表示，而且 255 也只是这种表示法所表示的数字之一。

数字颜色的设置同百分比值设置几乎完全一样，只是量度不一样：上限为 255 而不是 100%。表 3-2 里的颜色值对应于我们通常所见的颜色。

表 3-2　普通颜色的数字 RGB 颜色值

颜色名	对应的数字值
红色	rgb(255,0,0)
橙色	rgb(255,102,0)
黄色	rgb(255,255,0)
绿色	rgb(0,255,0)
蓝色	rgb(0,0,255)
靛蓝色	rgb(51,0,255)
紫罗兰色	rgb(204,0,255)
中灰色	rgb(128,128,128)
深灰色	rgb(51,51,51)
棕褐色	rgb(255,204,153)
金黄色	rgb(255,204,0)
紫色	rgb(255,0,255)

同样，在0～255之外的值也会被修剪，正如百分比一样——但在这里是被修剪成0或255。

```
H1 {color: rgb(0,0,0);}              /* black */
H2 {color: rgb(127,127,127);}        /* gray */
H3 {color: rgb(255,255,255);}        /* white */
P.one {color: rgb(300,2500,101);}    /* white */
P.two {color: rgb(-10,-450,-2);}     /* black */
```

如果读者喜欢用百分比，那也无妨，而且我们很容易在百分比和数字之间转换。如果已知每个RGB的百分比，那么只需将其应用于数字255，就可以得知对应的数字值。比如某种颜色为25%红色，37.5%绿色和60%蓝色，将它们分别乘上255就得到63.75，95.625和153。因为只允许整数出现，将其四舍五入得rgb(64,96,153)。百分比可以有小数，但数字值不能有小数。

当然，已知百分比后，要将其转换成数字就不难了。这种表示法对于使用Photoshop的人来说非常有用，因为Photoshop将产生数字颜色。或者对于那些已经习惯用数字0～255来表示颜色的人来说，尤为重要。

或许某些人对十六进制表示法会更熟悉，这是我们下面将要讲到的。

十六进制颜色

如果读者曾做过网页设计工作，而且在设计过程中设置过颜色，那么本部分将是一个缩影。对网页设计者来说，使用十六进制表示法来设置颜色是再熟悉不过的了：

```
H1 {color: #FF0000;}    /* set H1's to red */
H2 {color: #903BC0;}    /* set H2's to dusky purple */
H3 {color: #000000;}    /* set H3's to be black */
H4 {color: #808080;}    /* set H4's to be medium gray */
```

如果不熟悉这种表示法，这里有个初级读本。首先，十六进制意味着以十六为基数的计数，因此基本单位是16，而不是我们所习惯了的10。在十六进制数中，有效数字为0到9和A到F。一旦到达F，下一个数就是10。这样，类似于小孩数数的过程如下：

```
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F,
```

```
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F,
20, 21, 22, 23, ...
```

我自己也觉得用字母代替数字有些奇怪，但十六进制确实需要这样做。数字A到F实际上只是一种符号而已——没有其他任何含义，而且它比其他发明的符号易于记忆——加之也没有人会为字母虚构新的名字。

十六进制与十进制（基于10）的对应关系也是很直接的。05等于5，0C等于12，0F等于15，10等于16。但也不是真的直接对应，1F等于31，20等于32，等等。就像下面这样：

```
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F,
00, 01, 02, 03, 04, 05, 06 ,07, 08, 09, 10, 11, 12, 13, 14, 15,

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,

20, 21, 22, 23, ...
32, 33, 34, 35, ...
```

在计算机领域使用十六进制表示法已经有很长时间了，而且一般程序员都要学会其使用。多数程序员都觉得使用十六进制表示法很自然——甚至某些人还用十六进制进行思维——因此这成了CSS规范的一部分。为什么呢？因为规范是由程序员制定和编辑的。他们将颜色模式同十六进制关联起来就显得合情合理了。

因此，将三对十六进制组合成字符串，就可以设置颜色了。这种方法的更一般的描述如下：

```
#RRGGBB
```

这样看来，它与前面所讨论的方法就很像了——从0到255的数字。实际上，十进制的255等于十六制的FF，这样能解释很多关于此种方法如何工作的问题。它同最后一种方法一样：只是用了不同的计数体制。如果必须要二选一，那么使用任何一种都很方便。

因此，和使用三个0到255的数字来指定某种颜色的方法类似，可以用三对十六进制数来指代某种颜色。如果有个计算器用于在十进制和十六进制间转换，那么这会显得非常简单。若非如此，可能会显得稍微复杂一点。

我们再次列出一些颜色对照表，见表3-3。

表3-3 普通颜色的十六进制颜色值

颜色名	对应的十六进制值
红色	#FF0000
橙色	#FF6600
黄色	#FFFF00
绿色	#00FF00
蓝色	#0000FF
靛蓝色	#3300FF
紫罗兰色	#CC00FF
中灰色	#808080
深灰色	#333333
棕褐色	#FFCC99
金黄色	#FFCC00
紫色	#FF00FF

不管相信与否，还有一种设置颜色的方法，它包含更少的按键。

短十六进制颜色

现在是最后一种方法。我们先看一个例子，然后再来解释它：

```
H1 {color: #000;}  /* set H1s to be black */
H2 {color: #666;}  /* set H2s to be dark gray */
H3 {color: #FFF;}  /* set H3s to be white */
```

正如上面的标记所示，每个颜色值只有三位数字。既然00到FF之间的十六进制数需要两位，那么一共只有三位，这又是怎么回事呢？

答案是浏览器会对每一位进行复制。因此，#F00就等于#FF0000——而它的书写稍微简单一些。否则，这种方法就同前面的方法一样了，只是短了点儿。#6FA同#66FFAA一样，而#FFF同#FFFFF一样，即所谓的白色。这种方法有时也被称做短十六进制表示。

值得注意的是，十六进制法不像十进制方法，它没有定义修剪的方法。如果输入了无效数值，那么浏览器的动作将不可预测。一个成熟的浏览器应该执行修剪操作，以使超出范围的数变为某个极限值，但也不能完全依赖于此。例如，Navigator 4.x 不会忽略或修剪无效的颜色值，但会执行某种转化，从而产生完全不可预料的颜色。

几种颜色的对照

表 3-4 给出了我们所讨论过的几种颜色值的对照。某些颜色名（斜体表示）是合法的标准颜色声明值，而某些颜色名可能不为浏览器所识别，所以可以用RGB或十六进制值来定义（只要保证安全就行）。另外，还有些短十六进制值没有显示出来。在这种情况下，这些较长的值（六位）不能变短，因为它们并不重叠。例如，值 #880，扩展为 #888800，而非 #808000（否则就变为 olive 了）。当然也就没有 #808000 短十六进制了，表中的相应位置就为空。

表 3-4 颜色对照表

颜色名	百分比	数字	十六进制对	短十六进制
红色	rgb(100%,0%,0%)	rgb(255,0,0)	#FF0000	#F00
橙色	rgb(100%,40%,0%)	rgb(255,102,0)	#FF6600	#F60
黄色	rgb(100%,100%,0%)	rgb(255,255,0)	#FFFF00	#FF0
绿色	rgb(0%,100%,0%)	rgb(0,255,0)	#00FF00	#0F0
蓝色	rgb(0%,0%,100%)	rgb(0,0,255)	#0000FF	#00F
靛蓝色	rgb(20%,0%,100%)	rgb(51,0,255)	#3300FF	#30F
紫罗兰色	rgb(80%,0%,100%)	rgb(204,0,255)	#CC00FF	#C0F
水绿色	rgb(0%,100%,100%)	rgb(0,255,255)	#00FFFF	#0FF
黑色	rgb(0%,0%,0%)	rgb(0,0,0)	#000000	#000
紫红色	rgb(100%,0%,100%)	rgb(255,0,255)	#FF00FF	#F0F
灰色	rgb(50%,50%,50%)	rgb(128,128,128)	#808080	
浅绿色	rgb(0%,100%,0%)	rgb(0,255,0)	#00FF00	#0F0
褐色	rgb(50%,0%,0%)	rgb(128,0,0)	#800000	
深蓝色	rgb(0%,0%,50%)	rgb(0,0,128)	#000080	
橄榄色	rgb(50%,50%,0%)	rgb(128,128,0)	#808000	

表3-4 颜色对照表（续）

颜色名	百分比	数字	十六进制对	短十六进制
紫色	rgb(50%,0%,50%)	rgb(128,0,128)	#800080	
银色	rgb(75%,75%,75%)	rgb(192,192,192)	#C0C0C0	
深青色	rgb(0%,50%,50%)	rgb(0,128,128)	#008080	
白色	rgb(100%,100%,100%)	rgb(255,255,255)	#FFFFFF	#FFF
深灰色	rgb(20%,20%,20%)	rgb(51,51,51)	#333333	#333
棕褐色	rgb(100%,80%,60%)	rgb(255,204,153)	#FFCC99	#FC9

网络安全色

我们在前面讨论过，同一种颜色在不同的操作系统或浏览器上可能不一样。这里有一种方法可以局部改善这一问题，但它仍然依赖于颜色的选择。有216种颜色被认为是“网络安全的”，即对于所有计算机和浏览器都应该一样，不会有抖动或大的变化。注意我说的是“应该”——也不是绝对的，但在通常情况下可以这么认为。

所谓网络安全色指所有的RGB颜色值都为20%或51的倍数的那些颜色，当然十六进制为33。0%和0也是安全值。因此，如果用RGB百分比，那么这三个值要么为0%，要么能被20%整除。例如，rgb(40%,100%,80%)或rgb(60%,0%,0%)。如果在0～255中使用RGB值，那么这个值要么为0，要么能被51整除，如rgb(0,204,153)或rgb(255,0,102)。

对于十六进制对，相应的值为00、33、66、99、CC和FF。任何三个这些值的组合都是网络安全的。例如#669933，#00CC66和#FF00FF。也就是说0、3、6、9、C和F是短十六进制值的安全值。如#693、#DC6、#F0F等都是网络安全色。

定义颜色的方式就这么多了，我相信读者读完此节后，就不会老是用同一种方式去看彩虹了。下面我们将介绍测量的单位。

长度单位

为了正确显示不同的网页元素，许多CSS属性，比如页边界，依赖于长度的测量。而在CSS中，有很多种方法用于测量长度。

所有长度单位都可以通过正数或负数来表达，后面再跟上一个说明性短语——但某些属性只接受正值。它们也可以是实数，即带十进制小数的数，如10.5或4.561。所有长度单位后跟一两个字母的缩写，它代表实际的测量单位，如in（英寸），或pt（磅）。唯一例外的就是长度0，它不需要任何单位。

这些长度单位被分成两类：绝对单位和相对单位。

绝对单位

尽管绝对单位在网页设计时几乎没用，但我们仍从它开始，因为它最简单。五种类型的绝对单位如下：

英寸（in）

在美国，直尺上的单位就是英寸。规范中使用的也是这种单位，因为几乎全世界都用它作为测量单位，而且这或多或少还能反映出美国人对于互联网的浓厚兴趣。

厘米（cm）

在世界范围内都可以在直尺上发现这样的单位。1英寸等于2.54厘米，1厘米等于0.394英寸。

毫米（mm）

1厘米为10毫米，因此，25.4mm等于1英寸，1毫米就等于0.0394英寸。

磅（pt）

磅是标准的印刷上的量度，广泛应用于打印机和排字机，以及字处理程序。72个磅为1英寸，而且磅在使用公制之前就已经定义了。因此，12个磅的大写字母就为六分之一英寸大。如，`P {font-size: 18pt;}`就等于`P {font-size: 0.25in;}`。

pica（pc）

另一种印刷术语。1 pica（12 点活字）等于 12 磅，也即 6 pica 等于 1 英寸。同上，1 pica大小的大写字母就等于六分之一英寸高。例如，P {font-size: 1.5pc;}定义的文本大小就同上例中用磅定义的文本大小一样。

只有当浏览器知道所有关于监视器、打印机和其他用户代理的细节后，这些单位才真正有用。

例如，监视器的尺寸和分辨率就能影响网页的显示。当然，网页设计者对此也无能为力，但至少相互之间应保持单位的一致性，即 1.0 英寸的设置应该比 0.5 英寸的设置要大两倍，如图 3-3 所示。

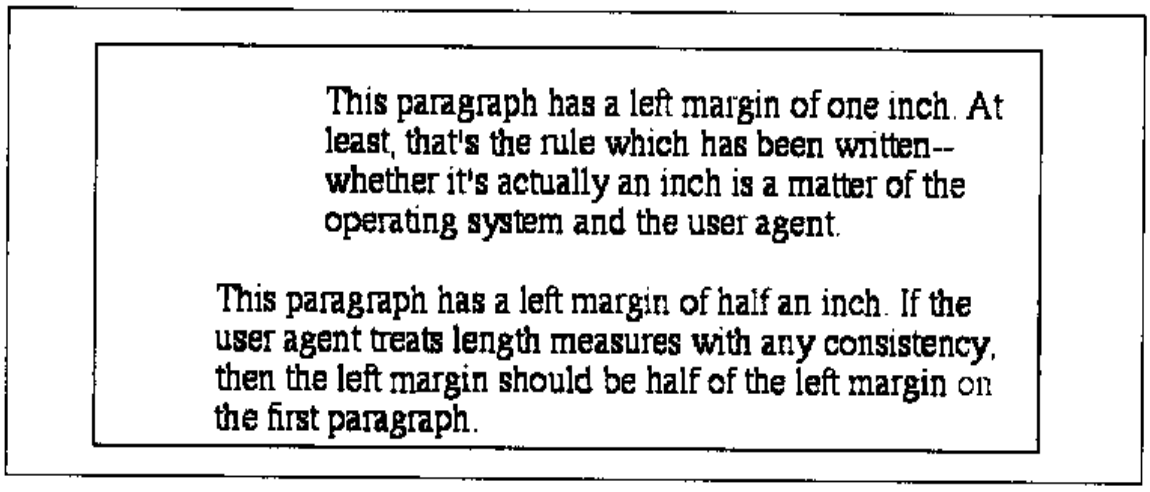


图 3-3　1.0 英寸的页边界（上）和 0.5 英寸的页边界（下）

使用绝对长度

Windows 用户可以设置显示程序使用的实际量度。点击“开始”按钮，选“设置”菜单，再点“控制面板”选项，然后在控制面板中双击“显示”属性。在“显示”对话框中有一个设置标签。点击它，然后再点“高级”按钮打开另一个对话框，对不同的计算机来说，这个对话框可能不一样。但应该能够看到标有“字体尺寸”的选项，在这个菜单中选择“其他”，然后将直尺贴近屏幕，拖动滚动条直到二者匹配为止。最后连续点击几次“OK”按钮即完成设置！

如果是 Macintosh 用户，就不能设置这一信息，这也没关系 —— Mac OS 已经假定了屏幕像素和绝对尺寸的对应关系，这是通过声明 72 像素等于 1 英寸来假定

的。这种假定是完全错误的，但它是嵌在操作系统内部的，所以不可避免，至少现在是这样。(将来的浏览器会包含一项定义像素和英寸的对应关系的选项，从而可以摆脱OS强加的限制。)

当然，在基于Macintosh的浏览器中，比如Navigator 4.x，Internet Explorer 3.x和4.x中，磅和像素是等价的——24pt的文本就是24像素高，而8pt的文本就等于8个像素高。不幸的是，这会显得有些小，因而不是很清晰，但在Windows的浏览器上还挺好。如图3-4所示。

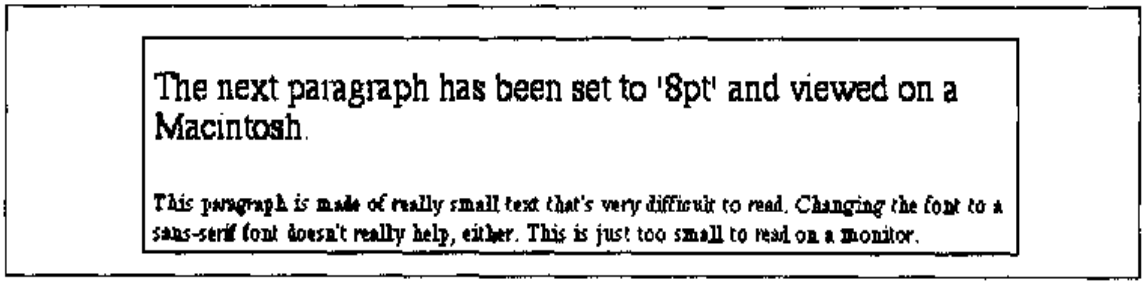


图3-4 小尺寸的字体在Macintosh上的显示效果不好

所以，在设计网页时，应避免使用磅作为单位。厘米、百分比和像素相对于磅来说，显示效果会好一些。

我们不妨假设计算机完全了解显示系统的细节，从而能够产生实际的尺寸。在这样一种情况下，就可以明确地声明一些规则，例如将上边界设为半英寸：`P {margin-top: 0.5in;}`。从而，规则中声明的尺寸也就是我们实际看到的物理尺寸。

绝对单位在为打印文档定义样式表时更为有用。在这里，使用英寸、磅和pica作为单位是很常见的。既然网页设计中使用绝对单位是危险的，那么让我们来看看相对单位。

相对长度单位

与自然界中爱因斯坦的相对论不同，这里要讨论的相对单位，是指它们在量度时要关系到的其他因素。它们所量度的实际距离（或绝对距离）可能会由于超越它们所控制的因素而改变，如屏幕分辨率、可视区域的宽度、用户的个人设置以及

其他的一些因素等。另外，对于某些相对单位，它们的尺寸几乎总是相对于某个元素而言的，而且有时还会随元素的变化而变化。

有三种相对的长度单位：em、ex和px。头两种代表“元素的字体的高度”和“字母"x"的高度”，它们是普通印刷上的量度；然而，在CSS中，它们还有出人意料的含义。最后一种长度px代表“像素”。一个像素就是计算机监视器上的一个点。之所以其值是相对的，是因为它依赖于显示设备的分辨率，我们在后面会详细讨论。

em和ex

现在来看em和ex。CSS中，一个“em”就是给定字体的font-size值，因此如果某个元素的字体尺寸为14个磅，那么对这个元素来说，一个“em”就同14个磅一样。换句话说，不管将字体尺寸设成什么，它总是这个元素1em的值。对于一个18磅文本的段落来说，1em的长度就为18磅。

很明显，这个值能随元素而变化。例如，给定H1的字体为24像素，H2为18个像素，而段落字体为12像素，我们就可以得到如图3-5所示的结果：

```
H1 {font-size: 24px;}
H2 {font-size: 18px;}
P {font-size: 12px;}
H1, H2, P {margin-left: 1em;}

<H1>Left margin = 24 pixels</H1>
<H2>Left margin = 18 pixels</H2>
<P>Left margin = 12 pixels</P>
```

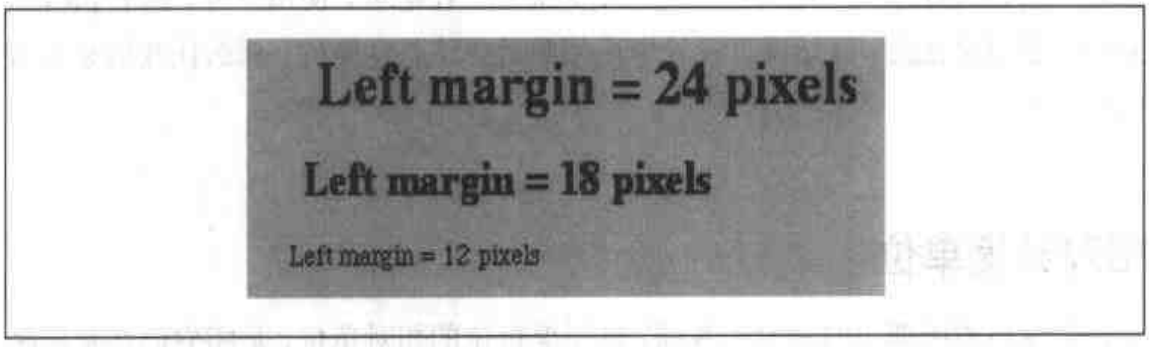


图3-5 页边界随字的大小而变化

对于这条规则，唯一的例外就是设置字体尺寸，这种情况下em的值是相对于父元素的。(如图3-6所示):

```
SMALL {font-size: 0.8em;}

<P>Although this text is the default size for the user agent, the
<SMALL>small-text element</SMALL> within the paragraph has a font
size which is 80% the rest of the text in the paragraph.</P>
```

Although this text is the default size for the user agent, the small-text element within the paragraph has a font size which is 80% the rest of the text in the paragraph.

图3-6 em设置字体尺寸是相对于父元素的

另一方面，ex是指所使用的字体中小写字母x的高度。所以在两个不同字体的24磅大小的文本段落里，ex的值是不一样的。这是因为不同的字体有不同的x高度，如图3-7所示。

Times: x

Garamond: x

Helvetica: x

Arial: x

Impact: x

Courier: x

图3-7 不同的字体、字母"x"具有不同的高度

实际应用中的em和ex

上面讲的都是理论上应该发生的。在实际应用中，许多用户代理通过将em的值除以2得到ex。为什么呢？在多数字体中都没有x的高度值，而且要计算它也很

困难。大多数小写字母的高度大约是大写字母高度的一半，所以假定1ex等于0.5em是一种很合理的假定。但我们希望，随着时间的推移，用户代理将开始使用ex的实际值，半个em的捷径也将渐渐消失。

像素

对像素来说，情况变得更为复杂。从表面来看，像素应该很直接。毕竟，当靠近监视器时，能看见它被分成非常细小的小方格。每个小方格就是一个像素。然而多少个小方格等于一个英寸呢？如果监视器的设置为1024像素宽，768像素高，而且其屏幕正好是14.22英寸宽，10.67英寸高。但这种情况很少出现，尤其是很难找到刚好这样尺寸的监视器。因此对大多数监视器来说，每英寸的实际像素数值（ppi）比72要大——有时会更大，甚至是120或更大。

更令人迷惑的是，CSS规范推荐一个“参照像素”，它大致是90ppi——实际上很少有操作系统会使用这一量度。而规范中假设90个像素等于1英寸，除此之外，别无其他，因此这也没什么帮助。通常，若声明font-size: 18px，那么就由用户代理去决定到底一个像素有多大。浏览器通常都会使用监视器的实际像素值，毕竟这是现成的，但对于其他显示设备，像打印机，情形就变得有点不确定了。

警告：这里有个例子，是在早期的CSS1的实现中由于像素而导致的问题。在Internet Explorer 3.x中，当文档打印时，IE3假定18px就为18个像素点，即为18/300，或6/100英寸——或者是0.06英寸。然而，这是非常小的文本！

另一方面；像素很适合表达图像的尺寸，即某个数量的像素高和宽。实际上，只有当需要将它们同文本尺寸相匹配时才采用其他单位来表示图像尺寸。这是一种很好的，有时也很有用的方法，而且当使用的是基于向量的图像，而非基于像素的图像时显得更为合理。（可缩放向量图像或许能成为现实。）

怎么办？

综上所述，最好的量度或许还是相对量度，尤其是em，还有px。而ex目前只是em量度的一部分，并不显得那么有用。如果用户代理能够支持实际的字母“x”的量度，那么ex或许会更有用。

这样，我们就结束了长度单位的讨论。本章的剩余部分就比较直观易懂了。

百分比值

同长度单位比较，百分比单位显得异常简单。正如读者所想像的那样——只是一个百分号（%）。例如：

```
H1 {font-size: 18pt;}
H1.tall {line-height: 150%;}
```

它用一个 tall 类来设置所有 H1 元素的行高比普通行高大一半，如图 3-8 所示。

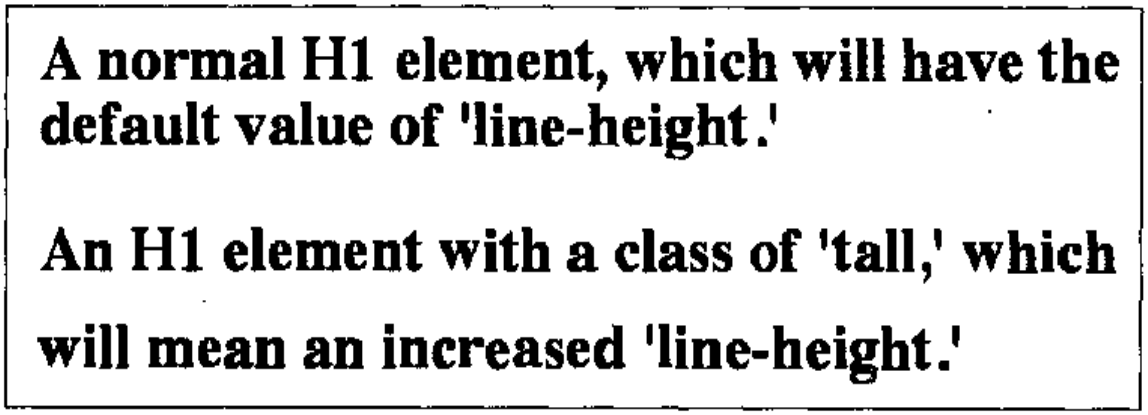


图 3-8　100% 的行高（上面）和 150% 的行高（底下）

百分比值总是通过另一个值来计算的——通常是一个长度单位。在这种情况下，行高为 27 磅（18pt 乘以 1.5）。它同设置行高为 1.5em 的结果一样，但说不准到底哪种方法更好。

通常，百分比可正可负。然而，有些属性只接受正值，而不允许负值（包括负的百分比值）。这些属性会在后续的章节中提到。

URL

如果读者写过网页，那么对于 URL 应该很熟悉。在样式表中，URL 并不常用，但如果必须要应用它——如在 @import 语句中，它用于载入一个样式表——那么一般的格式为：

```
url(http://server/pathname)
```

这个例子定义了一个绝对的URL。绝对就是指这个URL在任何网页中都可以找到，因为它在网络空间中定义了一个绝对的位置。假设有个服务器为www.waffles.org，在这个服务器上有个*pix*目录，而且在目录中有个图像为waffle22.gif。这种情况下，这个图像的绝对URL为http://www.waffles.org/pix/waffle22.gif。不管将它放在哪里都有效。而不论网页是在www.waffles.org上还是在web.pancakes.com上。

另外一种类型的URL是相对URL，这样取名是因为这类URL指定的位置是相对于使用它的文档来说的。如果要引用一个相对位置，例如与网页同在一个目录中的文件，其格式如下：

```
url(pathname)
```

只有当这个图像与包含这个URL的网页在同一个服务上才能工作。假设一个网页为http://www.waffles.org /syrup.html，而且有*waffle22.gif*这样一个图像位于本网页。在这种情况下，URL为*pix /waffle22.gif*。

这样，浏览器就知道将网页所在的位置加上这个相对URL，即服务器名http://www.waffles.org/ 加上路径*pix/waffles22.gif*就等于http://www.waffles.org/pix/waffles22.gif。

下面是另外两个例子：

```
@import url(http://css1.style.org/example.css);

BODY {background-image: url(hatch.gif);}
```

只要它们定义的是有效的位置，使用绝对或相对URL都无关紧要，哪种方便就用哪种。

另外需要注意的是相对URL是相对于样式表而不是相对于HTML文档。例如，在一个外部样式表中引入另一个样式表。如果使用相对URL来引入第二个样式表，那么它必须是相对于第一个样式表的。例如，有这样一个HTML文档http://www.waffles.org/toppings/tips.html，里面有个LINK链接一个样式表http://www.waffles.org/styles/basic.css:

```
<LINK REL="stylesheet" TYPE="text/css"
    HREF="http://www.waffles.org/styles/basic.css">
```

在文件 *basic.css* 里是一个 @import 语句指向另一个样式表:

```
@import url(special/toppings.css);
```

@import 语句会载入文件 http://www.waffles.org/styles/special/toppings.css，而不是http://www.waffles.org/toppings/special/toppings.css。如果要载入后一个样式表，那么 @import 应该像下面的这样:

```
@import url(http://www.waffles.org/toppings/special/toppings.css);
```

换句话说，就是用一个绝对 URL。

在任何情况下，这都不失为一个好主意。Navigator 4 将相对 URL 解释成相对于 HTML 文档，而不是样式表。这实际上是错误的，但 Navigator 4 确实是这么做的。因此最保险的方法就是用绝对 URL，因为 Navigator 至少能够正确处理它。

注意在 url 和开括弧之间不能有空格。如:

```
BODY {background: url(http://www.pix.web/picture1.jpg);}   /* correct */
BODY {background: url (images/picture2.jpg);}              /* incorrect */
```

如果包含空格，则整个规则无效，进而被忽略。

CSS2 中的单位

除了前面所讨论的之外，CSS2中还增加了一些新单位，几乎所有这些单位都同听觉样式表（用于支持语音的浏览器）相关。我们也将其简单地列出来:

角度值

用于定义发出声音的位置。有三种类型的角度: 度（deg）、梯度（grad）和弧度（rad）。例如，一个直角就是 90 deg，100grad 或 1.57rad; 每种值都可以翻译成 0 度到 360 度中的值。负值也一样是允许的。-90deg 就等同于 270 deg。

时间值

用于指明语音元素间的延迟，可以是毫秒（ms）或秒（s）。100ms和0.1s等价。时间值不能为负。

频率值

用于指定语音浏览器能够产生的声音的频率。频率值可以是赫兹（Hz）和兆赫（mHz），不能为负。区分大小写，因此10mHz和10mhz不一样。

除此之外，还有另外一个新名字。*URI*就是唯一资源标识符，它是唯一资源定位（URL）的另一个名字。大部分是属于语义上的差别，但许多人开始采纳将在线地址作为URI引用的习惯，而不是URL。但规范中仍规定URI作为url(...)来声明，因此很难理解在CSS2中包含如何使用URI而非URL这一节的意义何在。

小结

单位和值覆盖了很广的范围，从长度单位到颜色单位再到文件的位置（如图像文件）。对于大部分单位，用户代理都能正确识别；只有很小的一些错误或缺陷。例如，相对URL的解释使许多制作者感到困惑，也导致了对绝对URL的过分依赖。颜色单位几乎也能很好地同用户代理配合，只是有时会出现一些怪异的情况。然而，奇特的长度单位有时不会产生错误，这是任何制作者都需要解决的有趣的问题。

所有这些单位都有它们自身的优点和缺点，这要看它们所处的环境。有些部分我们已经讨论过了，至于在某些环境中的细微差别将在本书的剩余部分讨论。这将从CSS用于改变文本显示效果的属性开始。

第四章

文本属性

当网页设计的大部分精力都放在挑选合适的颜色和得到最酷的网页效果时，问题也变得多起来。我们可能会因为文本的放置和效果的显示而花掉大量的时间。这多半同<FONT>和<CENTER>这样的HTML标签有关系，因为它们能对文本的外观及位置加以控制。

正因为这样，大多数CSS都和以这样或那样的方式影响文本的属性相关联。在CSS1里，这些属性被分为两部分："文本属性"和"字体属性"。本章将讲述前者，后者将在第五章中解决——因为它们都很复杂，所以各用一章来介绍。

使用文本

或许读者想知道文本和字体的区别。简单地说，文本是内容，字体用于显示，它是一种改变文本外观的方法。在讨论字体前，读者可能已经了解了一些简单的可以影响文本外观的方式。另外，在本章讨论的某些东西在字体属性一章也是很重要的，因此，我们先重点讨论文本属性。

运用文本属性可以改变某行上文本的位置，而且可以实现上标、下划线以及改变大小写等，甚至还可以在某种程度上模拟打字机上的Tab键的功能。

缩进和水平对齐

首先讨论怎样才能影响同一行文本的水平位置。这不同于实际的位置，它是以页面本身为参照物的。试着考虑怎样影响文本行的布局，正如在创建一份时事通讯或写一篇报告时要做的那样。

文本缩进

在网页的文本格式中最重要的效果之一就是段落首行的缩进（indent）。（其次是消除段落间的“空白行”，将在第七章中讨论）。有些站点通过在段落的首字母前放置一个小型透明图像来解决这一问题，这样就可以将文本推后，还有的使用非标准的 SPACER 标签来得到同样的效果。但在 CSS 中有更好的方式。

text-indent

允许值	<长度> | <百分比>
初始值	0
可否继承	是
适用于	块级元素

注意：百分比是指相对于父元素的宽度。

使用 text-indent，任何元素都可以让首行以给定的长度缩进，长度甚至可以是负数。这一属性的最常用方式是段落的首行缩进：

```
P {text-indent: 0.25in;}
```

这条规则使任何段落的首行缩进四分之一英寸，如图 4-1 所示。

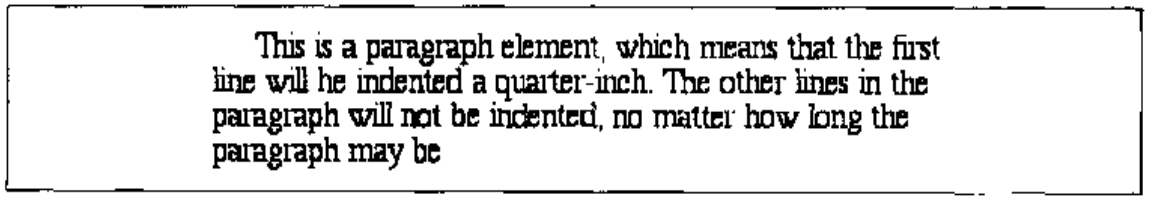


图 4-1 文本缩进

通常，text-indent可应用于任何块级元素，如PRE，H1或BLOCKQUOTE。但不能将它应用于内联元素，如STRONG或A，也不能应用于替换元素，如图像。但是，如果在段落的首行有个图像，它将同文本一样向右移，如图4-2所示。

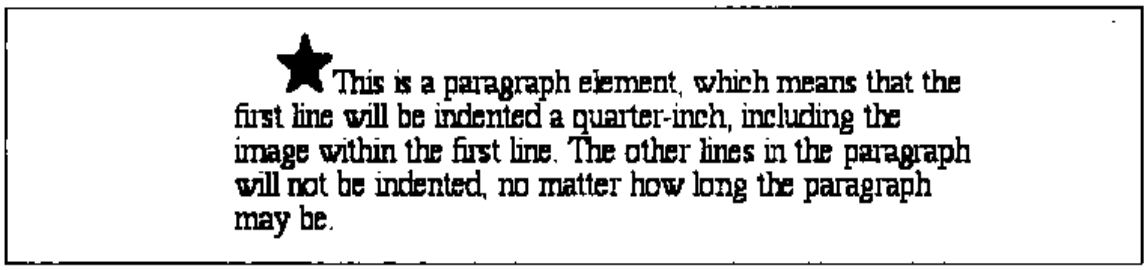


图4-2　文本缩进和图像

text-indent的值可以为负，而且能用于一些有趣的方式。最常用的就是“悬挂缩进(hanging indent)”，它使得首行悬挂在元素其他部分的左边，如图4-3所示：

```
P {text-indent: -4em;}
```

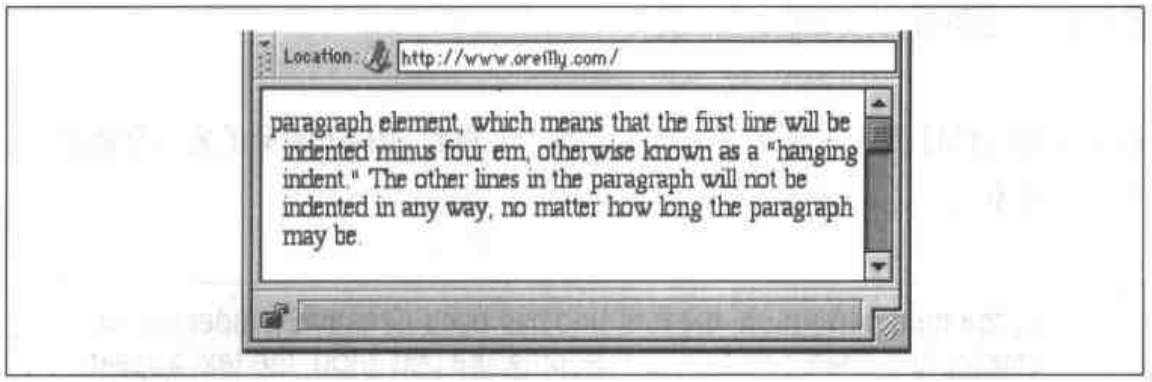


图4-3　负的文本缩进

正如在图4-3中看到的，设置负缩进值有个潜在的危险：头三个单词(“This is a”)被浏览器窗口的左边沿遮掉了。为了避免这种问题，可以使用边界或补白来配合负缩进，如图4-4所示：

```
P {text-indent: -4em; padding-left: 4em;}
```

负缩进有时很有用。考虑下面这个例子，如图4-5中演示的一样，它在文本中混合了一个浮动图像：

```
p.hang {text-indent: -30px;}
```

```
<P CLASS="hang"><IMG SRC="floater.gif" WIDTH="30px" HEIGHT="60px"
ALIGN="left" ALT="Floated">This paragraph has a negatively indented first
line, which overlaps the floated image which precedes the text. Subsequent
lines do not overlap the image, since they are not indented in any way.</P>
```

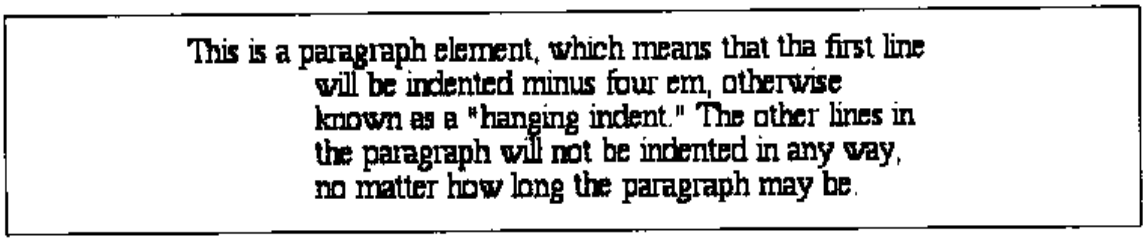


图 4-4 浮动图像和负文本缩进

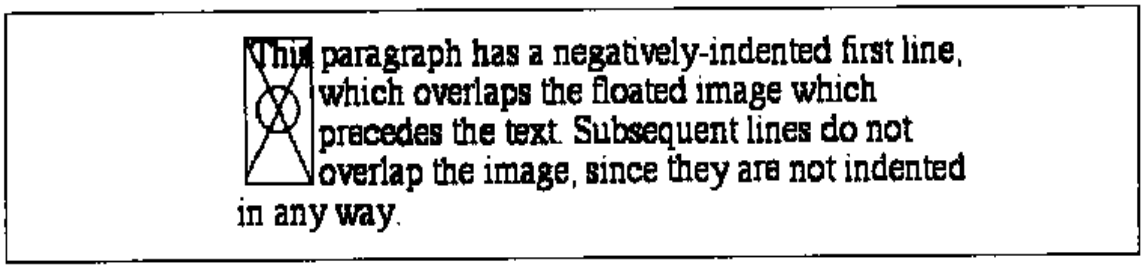


图 4-5 说明使用负文本缩进的原因

许多有趣的设计都可以通过这一简单的技术来得到。图4-6显示了另一个例子，这里首行文本为 -40px缩进。

图 4-6 负缩进和浮动图像

任何长度单位都可以用于 text-indent，而且，百分比值也是允许的。在这里，百分比是指缩进相对于父元素的宽度的值。这样，如设置 5% 的缩进值，那么此元素的首行将按其父元素的宽度的 5% 进行缩进，如图 4-7 所示:

```
DIV {width: 400px;}
P {text-indent: 5%;}

<P>This paragraph is contained inside a DIV which is 400px wide, so the
first line of the paragraph is indented 20px (400 * 5% = 20). This is because
percentages are computed with respect to the width of the parent element.</P>
```

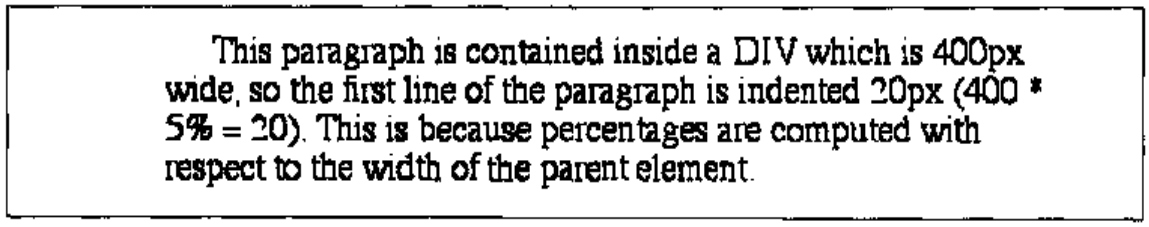


图4-7 百分比的文本缩进

注意，缩进只应用于元素的第一行，即使是插入了换行符也一样，如图4-8所示：

```
DIV {width: 400px;}
P { text-indent: 5%;}

<DIV>
<P>This paragraph is contained inside a DIV which is 400px wide, so the
first line of the paragraph is indented 20px (400 * 5% = 20). Subsequent
lines within the same element are not indented,<BR>
even if they follow a<BR>
line-break.</P>
<P>Once again, the first line of this paragraph is indented by 20px,
but other lines in the same element are not.</P>
</DIV>
```

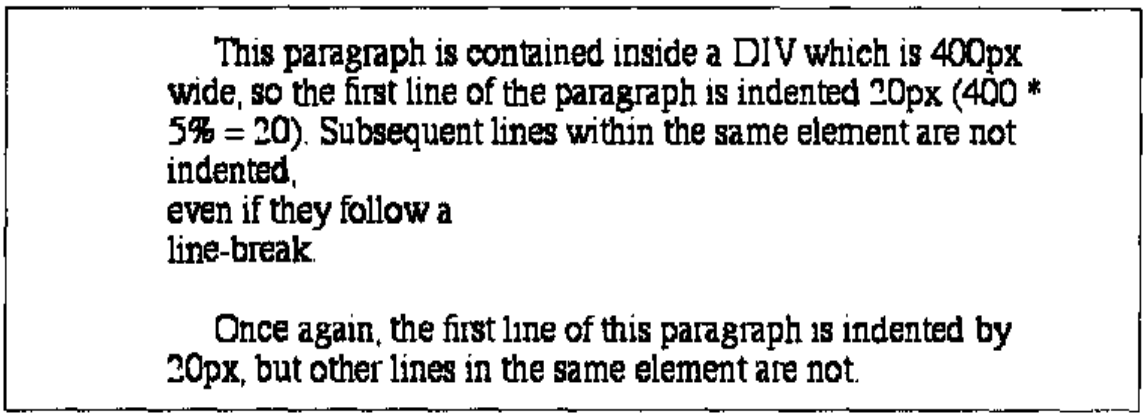


图4-8 文本缩进和换行

text-indent中最有趣的部分是继承，但继承的是计算值，而非声明值。如下面的标记：

```
BODY {width: 500px;}
DIV {width: 500px; text-indent: 10%;}
P {width: 200px;}

<DIV>
This first line of the DIV is indented by 50 pixels.
<P>This paragraph is 200px wide, and the first line of the paragraph
is indented 50px.  This is because computed values for 'text-indent'
are inherited, instead of the declared values.</P>
</DIV>
```

即使段落只有200像素宽，它的首行缩进还是50个像素（如图4-9所示），它是text-indent的继承值。

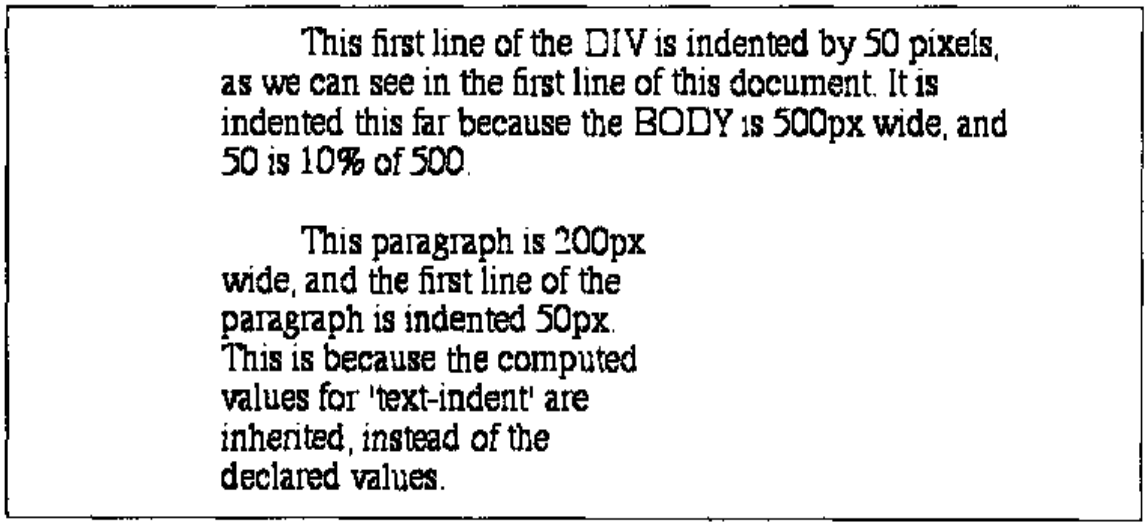


图4-9 继承文本缩进

文本对齐

比text-indent更为基本的是text-align属性，它用于元素中文本行的对齐方式的设置。它共有四个取值；前三个都很简单，但第四个就有些复杂了。

理解这些值的最快方式就是参看图4-10。

text-align是另一种只应用于块级元素的属性。居中某行中的一个链接，而不管行中剩余部分的对齐方式是行不通的（可能读者也不想这样做）。

text-align

允许值	left | center | right | justify
初始值	与用户代理有关
可否继承	是
适用于	块级元素

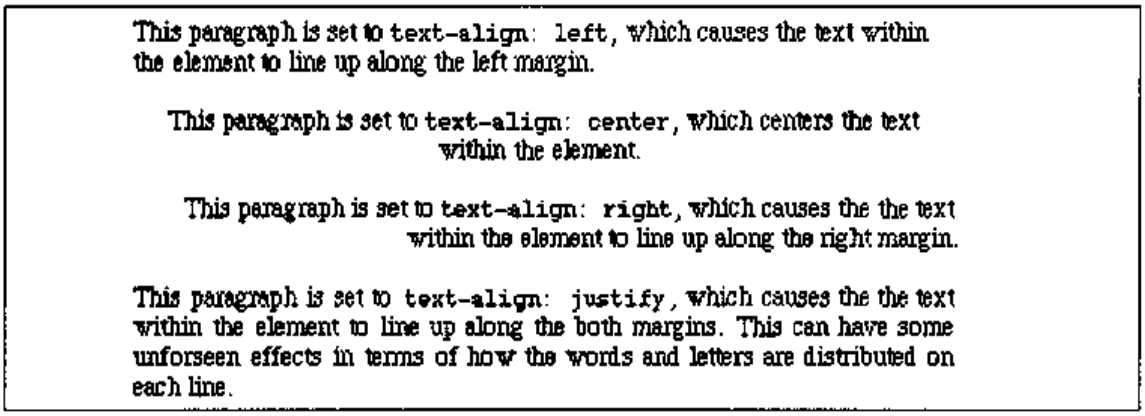


图 4-10 文本对齐属性的行为

规则 text-align: center 用于替换 CENTER 标签的操作，如图 4-11 所示：

```
H1 {text-align: center;}
```

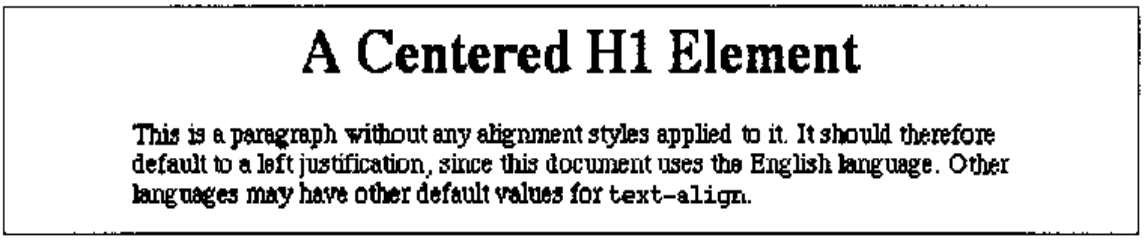


图 4-11 用文本对齐属性居中文本

也可以使元素居中，这会使元素中的文本和图像同时居中——同样，其行为类似于 CENTER 标签，如图 4-12 以及下面的标记所示：

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

```
<H1>An Un-centered H1 Element</H1>
<DIV CLASS="first">
<P>
This is a paragraph without any alignment styles applied to it. However, it is
contained within a DIV element which has alignment set, and this alignment will
inherit into the paragraph. This will also affect any images which appear within
the DIV, such as this one <IMG SRC="star.gif"> or the next one.
</P>
<IMG SRC="floater.gif">
</DIV>
```

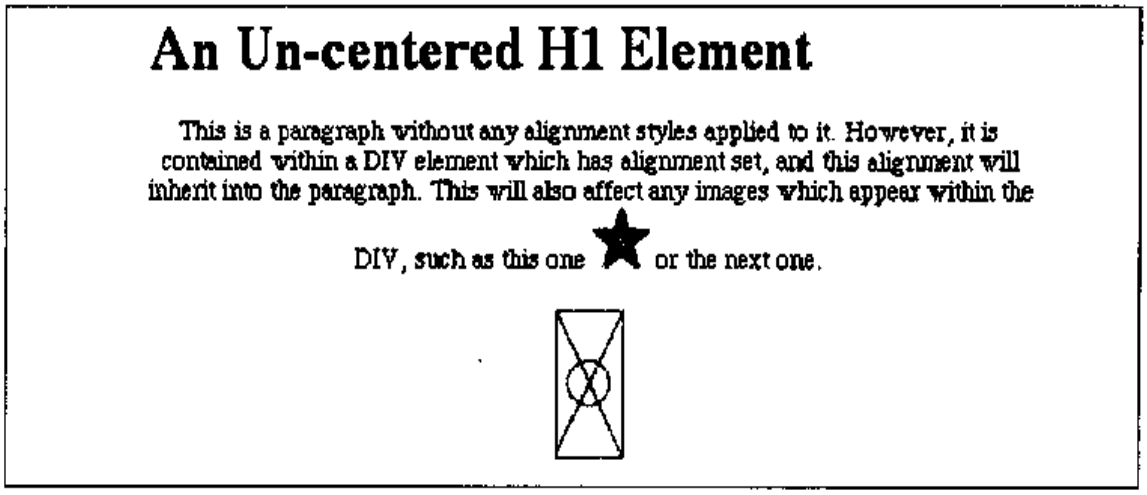


图 4-12 用文本对齐属性居中文本和内联图像

然而，如想使一个图像单独居中，用text-align就不合适了。按照下面的标记，会得到如图 4-13 所示的结果：

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

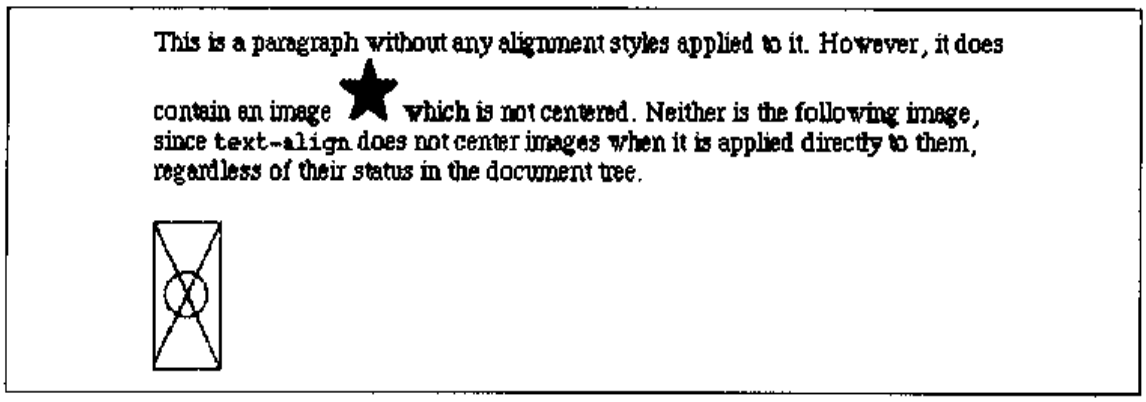


图 4-13 文本对齐属性和块级图像

图像不能居中的最基本原因就是它不是块级元素。用 text-align 居中图像的唯一方法是在图像周围包上一个 DIV，然后设置 DIV 的内容为居中：

```
<DIV STYLE="text-align: center;">
   <IMG SRC="shiny.gif" ALT="Shiny object">
</DIV>
```

如图 4-14 所示，图像居中了。

图 4-14 用包装方式来居中图像

对西方语言来说，是从左读到右，text-align 的缺省值为 left，即文本从左边空白处依次排列，右边可能有不定数目的空白（也可称之为“从左到右”文本）。有些语言，如希伯来语或阿拉伯语的缺省值为 right，因为这些语言是从右读到左的。center 的含义很明确，它使元素中的每行文本居于中间位置。

至于 justify，有一些问题需要考虑。如图 4-15 所示，两端对齐文本的格式为：文本行的两端被放置于父元素的内边缘。通过调整单词和字母的间距，使得每一行都刚好是同样的长度。这种效果同印刷体类似（例如本书），但在 CSS 里，还要考虑另外一些问题。

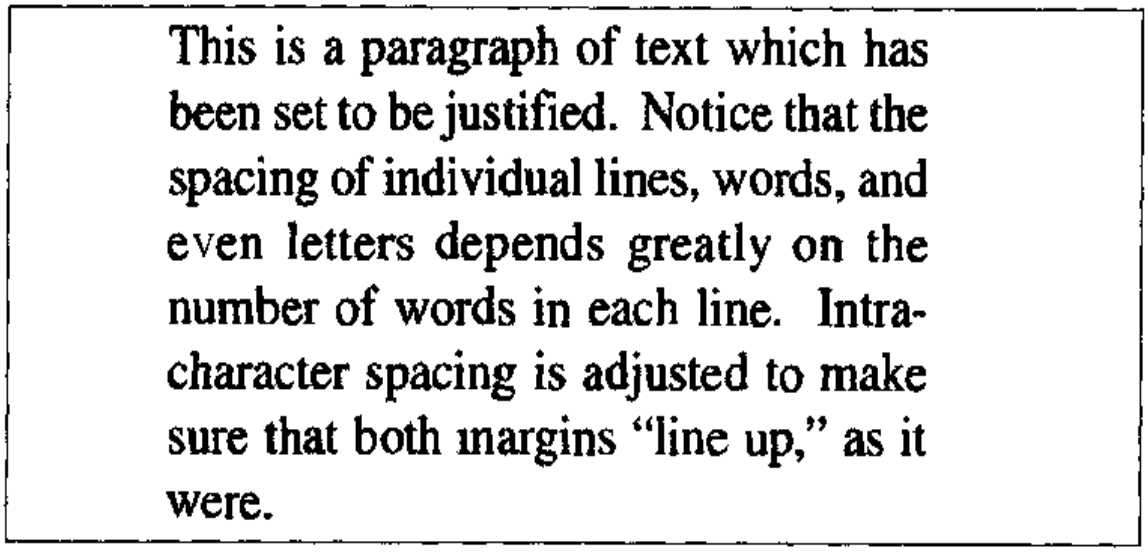


图 4-15 两端对齐文本

CSS并不指明两端对齐文本怎样去延伸，才能使其刚好填满父元素左右边缘间的空间，因此某些用户代理（user agent）只在单词间增加额外的空间。而其他用户代理会在字母间分布一些额外的空间。同理，某些用户代理在某些行上会去掉一些间隔，使文本显得比平时更加紧凑。所有这些不同的选择都会影响到元素的外观，甚至可能改变其高度，这都依赖于用户代理对文本行所做的调整（见图4-16）。

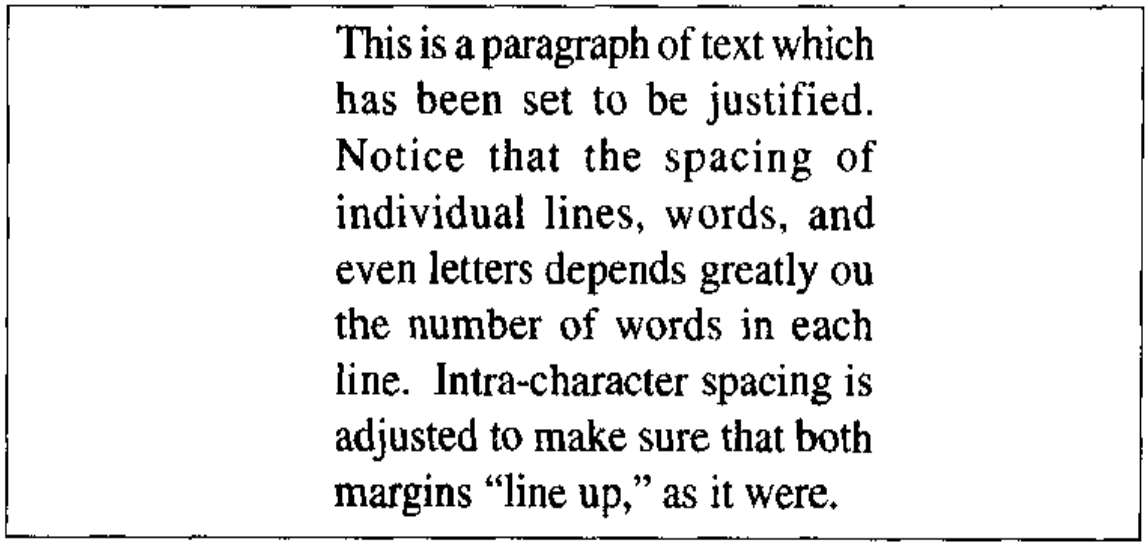


图4-16 不同形式的两端对齐

另外一个问题是CSS不会关心连字符号的处理。大多数两端对齐文本都使用连字符来连接跨两行的长单词，这样可以减少单词间的间隔，而且改进了文本行的外观（见图4-17）。

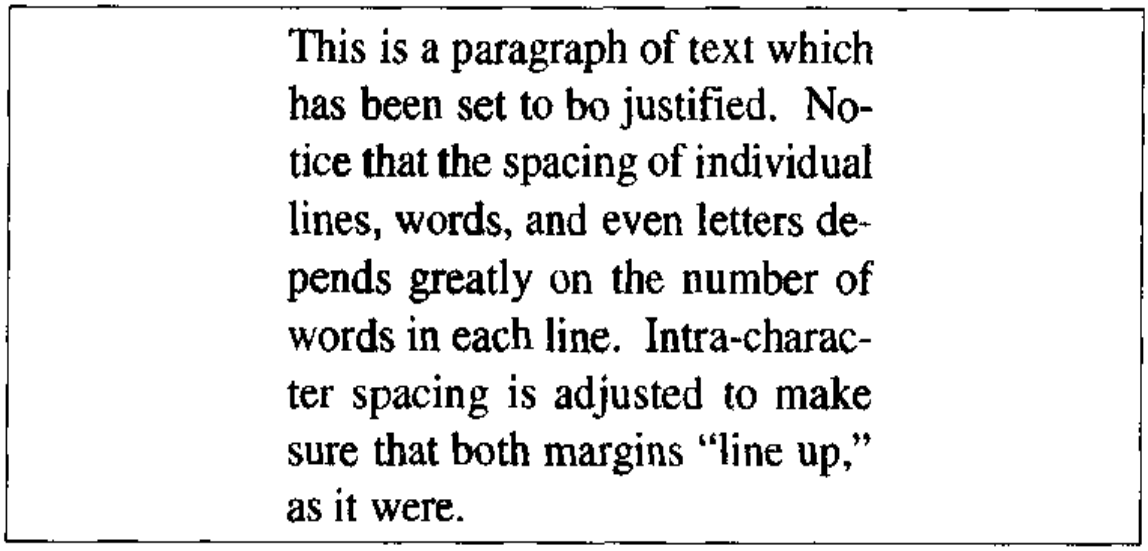


图4-17 带连字符的两端对齐

注意：在CSS中没有关于连字符的描述，而不同的语言往往有不同的连字符规则。而试图编造一套规则会显得很不完全，规范中简单地避免了这一问题。另外，还可以让用户代理使用它们自己的连字符规则，而且可以在不受CSS规范的约束下随时改进它们。

因为，在CSS中没有连字符，用户代理也就不可能自动执行任何连字符规则。这样，CSS下的两端对齐文本远没有它用于打印时那么有吸引力，尤其是当元素很窄，每行只能有少许单词时，如图4-18所示。当然，我们也可以使用两端对齐文本，但要考虑到其缺陷。

警告：几乎所有支持CSS的浏览器都能处理多数text-align的值，但在处理两端对齐时常常失败。Navigator 4支持两端对齐，但错误很多——常常在表格中出故障。Internet Explorer 4.x不支持两端对齐，然而IE5支持，Opera 3.6也支持。

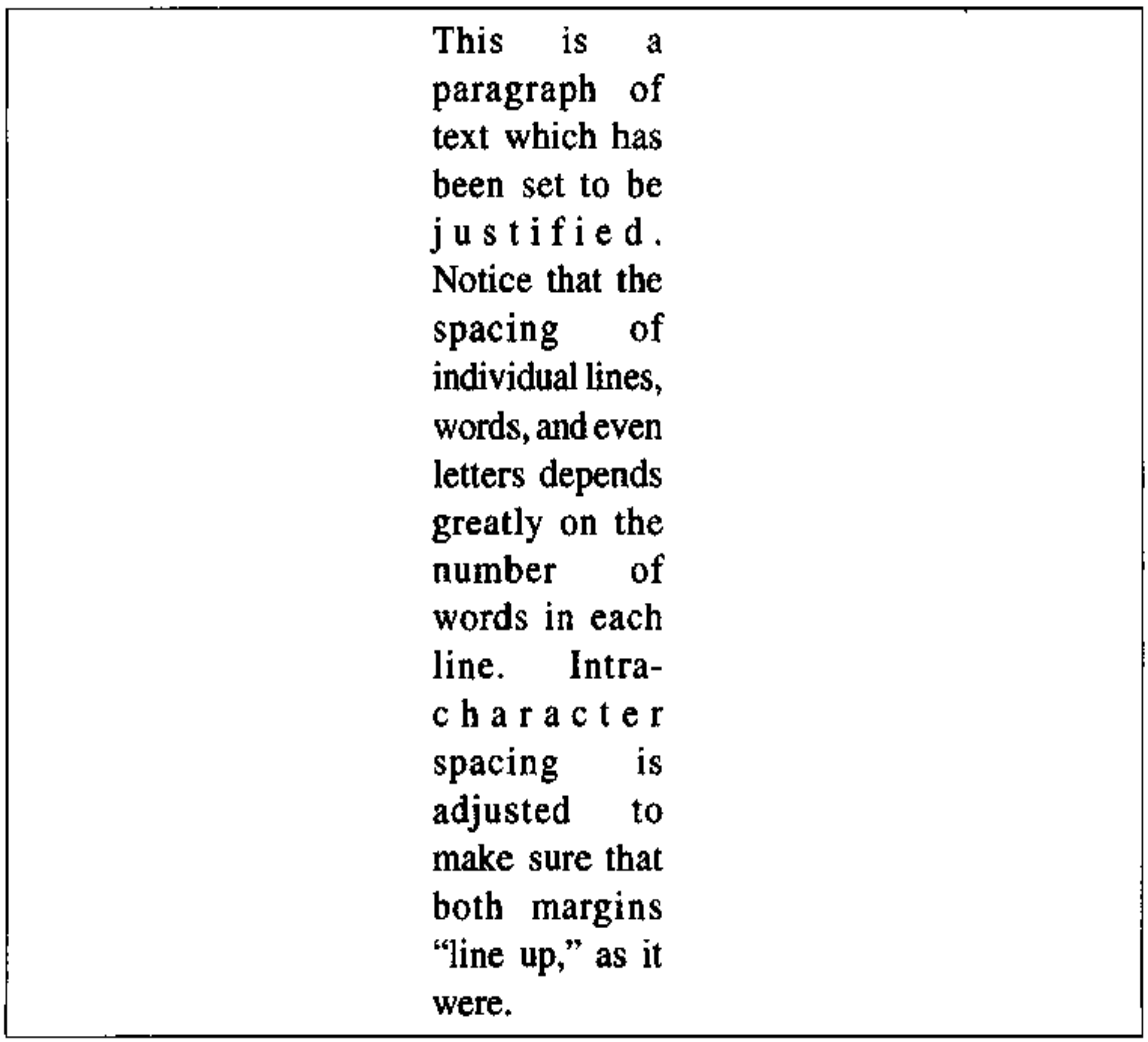


图4-18 在很窄的情况下不带连字符的两端对齐

处理空白

让我们来讨论 white-space（空白）属性，它对文本的显示有着重要的影响。

white-space	
允许值	pre | nowrap | normal
初始值	normal
可否继承	否
适用于	块级元素

运用这一属性，可以影响浏览器对单词和文本行间空白的处理方式。从某种程度上说，HTML 已经做了这一点：它将任何空白压缩成单个空白符。这样，下面的标记的效果如图 4-19 所示，在每个单词间只有一个空白符：

```
<P>This    paragraph    has      many
    spaces          in it.</P>
```

This paragraph has many spaces in it.

图 4-19 显示一个带很多空白的段落

也可以用下面的声明显式设置其行为：

```
P {white-space: normal;}
```

这条规则告诉浏览器忽略多余的空白。任何额外的空白符及回车符会被浏览器完全忽略。

如果设置 white-space 属性值为 pre，那么元素内的空白将不被忽略，就好像这个元素是一个 PRE 元素一样，如图 4-20 所示：

```
P {white-space: pre;}

<P>This    paragraph    has      many
    spaces          in it.</P>
```

This paragraph has many
spaces in it.

图4-20 保留标记中的空白符

当white-space取值为pre时，浏览器会注意额外的空白符和回车符。在这种情况下，也只有在此情况下，任何元素才能像 PRE 元素一样动作。

如果取值为nowrap，它将阻止文本被包裹在块级元素里，除非使用
 元素。这一点和设置一个表格单元不能通过 <TD NOWRAP> 来包裹很类似，除非white-space的值能应用于任何块级元素。这样可以得到如图4-21所示的效果：

```
<P STYLE="white-space: nowrap;">This paragraph is not allowed to wrap,
which means that the only way to end a line is to insert a line-break
element.  If no such element is inserted, then the line will go forever,
forcing the user to scroll horizontally to read whatever can't be
initially displayed <BR>in the browser window.</P>
```

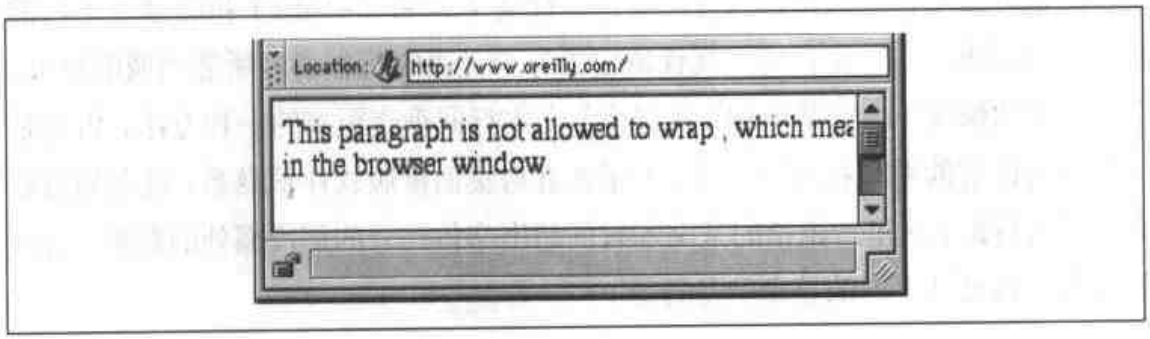


图4-21 用空白属性来避免文本包裹

实际上，可以用white-space来替换表格单元的nowrap属性，如图4-22所示。

```
TD {white-space: nowrap;}

<TABLE><TR>
<TD>The contents of this cell are not wrapped.</TD>
<TD>Neither are the contents of this cell.</TD>
<TD>Nor this one, or any after it, or any other cell in this table.</TD>
<TD>CSS prevents any wrapping from happening.</TD>
</TR></TABLE>
```

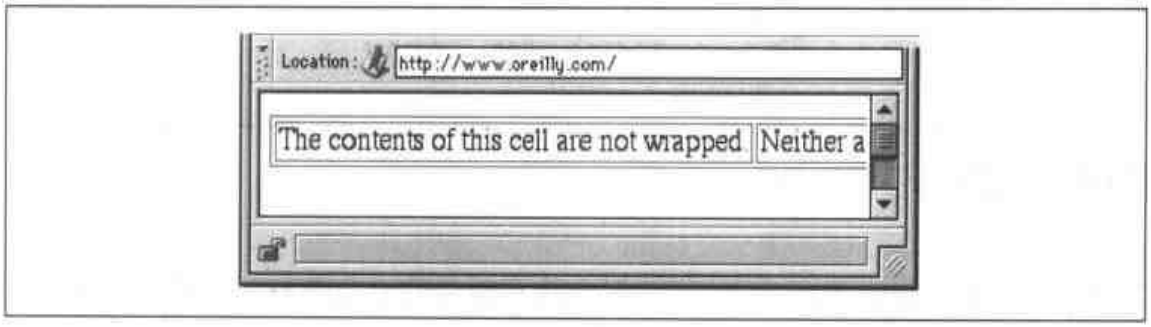


图 4-22 用 white-space: nowrap 属性来避免表格单元的包裹

警告：尽管 white-space 看起来很有用，但写本书时，除 IE5 for Macintosh 和 Netscape 6 的预览版外，没有浏览器支持它。同样，表格单元的 nowrap 属性也是不可用的，尽管它看上去是非常简单的行为，但仍未得到支持。

行高

与将在第五章中讨论的字体尺寸相对应，行高（line-height）指的是文本行的基线间的距离。实际上，这一属性决定了每个元素的行框高度所能增减的数值。在最简单的情况下，它是增加（或减少）文本行间垂直距离的一种方法，但这是一种容易误解的看待行高的方式。如果读者对桌面排版软件很熟悉，就会知道行高控制着行距，这里行距指的是文本行间超出字体尺寸的那段额外的距离。行高的值与字体尺寸之间的差别就是行距。

line-height

允许值	<长度> | <百分比> | <数字> | normal
初始值	normal
可否继承	是
适用于	所有元素

注意：百分比是指相对于元素的字体尺寸。

从技术角度来讲，一行中的每个元素都有一个内容区域，它是由字体尺寸决定的。

内容区域又包含一个内联框，在没有其他因素的情况下，内联框刚好等于内容区域。然而行距可以增加或减少内联框的高度。这是通过将行距分成两半，再将每个半行距分别应用于内容区域的顶部和底部来实现的。其结果就是内联框。例如，如果内容区域为14磅高，而行高设成18磅，那么其差别（4磅）将被分成两半，而且每一半将分别加到内联框的顶部和底部，这就得到一个18磅高的内联框。这听起来好像是用一种迂回的方式来描述行高，其实不然，在本书的后面将会明白这样描述行高的原因所在，在第八章“可视化格式编排”中有关于内联格式化模型的详细解释。

如果使用的是缺省值normal，行间垂直距离将会是用户代理的缺省值。它通常是字体尺寸的1.0到1.2倍，但也能随用户代理而改变。

除em和ex以外的长度都是些简单的量度（例如：0.125in）。这里有个小小的警告，即使是使用像4cm这样有效的长度单位时，浏览器（或操作系统）都可能得出一个不正确的实际量度值。这样，当在监视器上显示文档时，直尺上的行高的读数并不是正好4cm。要了解更多的细节，请参看第三章“单位和值”。

同em和ex一样，百分比的计算也是相对于元素的字体尺寸的。这样，下面的标记就显得相对直接一些，如图4-23所示：

```
BODY {line-height: 14pt; font-size: 13pt;}
P.one {line-height: 1.2em;}
P.two {font-size: 10pt; line-height: 150%;}
P.three {line-height: 0.33in;}

<P>This paragraph inherits a 'line-height' of 14pt from the BODY, as well as
a 'font-size' of 13pt.</P>
<P CLASS="one">This paragraph has a 'line height' of 16.8pt (14 * 1.2), so
it will have slightly more line-height than usual.</P>
<P CLASS="two">This paragraph has a 'line-height' of 15pt (10 * 150%), so
it will have slightly more line-height than usual.</P>
<P CLASS="three">This paragraph has a 'line-height' of 0.33in, so it will have
slightly more line-height than usual.</P>
```

然而，一旦行高属性从一个块级元素继承到另一个块的元素时，情况就变得不确定了。不管子元素中的字体尺寸如何设置，计算出的行高值都会被继承。这就可能导致一些问题，正如图4-24所示。

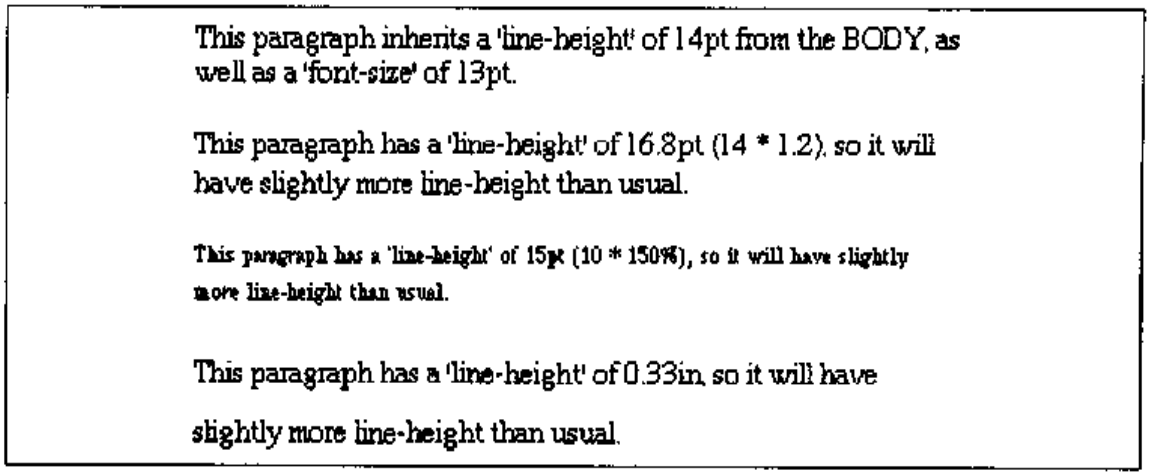


图 4-23 行高属性的简单计算

```
BODY {font-size: 10pt;}
DIV {line-height: 12pt;}
P {font-size: 18pt;}

<DIV>
<P>This paragraph's 'font-size' is 18pt, but the inherited 'line-height'
is only 12pt.  This may cause the lines of text to overlap each other by
a small amount.</P>
</DIV>
```

图 4-24 小的行高、大的字体尺寸会导致一些小问题

有两种解决办法。一种是显式地为每个元素设置行高，但这种方法不是很实用，因为可能有很多元素都需要这样的属性。另一种方法就是设定一个数，实际上是设定一个缩放因子：

```
BODY {font-size: 10pt;}
DIV {line-height: 1.5;}
P {font-size: 18pt;}
```

当使用这样一个数字时，继承的就不再是计算值，而是这个缩放因子。这个缩放因子应用于这个元素及其所有的子元素，因此每个元素都有一个关于它自己的字体尺寸的行高计算值，如图 4-25 所示：

```
BODY {font-size: 10pt;}
DIV {line-height: 1.5;}
P {font-size: 18pt;}

<DIV>
<P>This paragraph's 'font-size' is 18pt, and since the 'line-height'
set for the parent DIV is 1.5, the 'line-height' for this paragraph
is 27pt (18 * 1.5).</P>
</DIV>
```

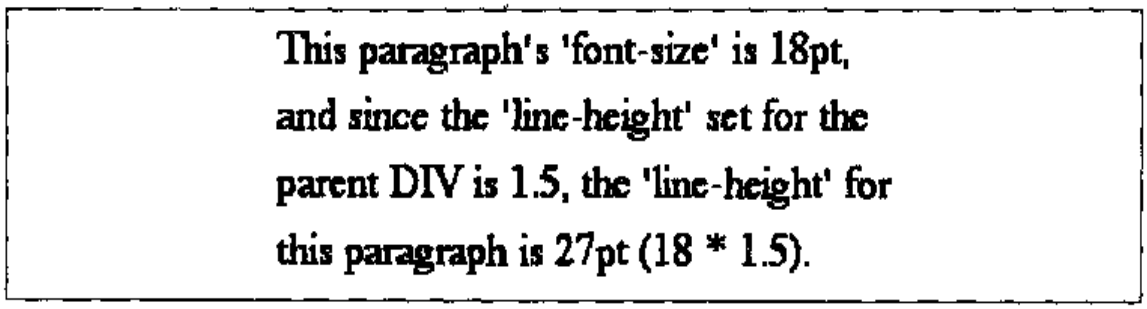


图4-25 使用行高因子来解决继承问题

正如前面介绍过的，尽管看上去好像是行高属性为每个文本行的顶部和底端分配了额外的空间，实际上它是从内联元素的顶部和底部加上（或减少）了某个固定的值。当某行有不同字体尺寸的元素时，这可能会导致一些奇怪的情况发生，但目前，我们还是坚持这样一种简单的情况。假设某个段落的缺省字体尺寸为12pt，考虑下面的规则：

```
P {line-height: 16pt;}
```

既然12磅文本的“固定”行高为12磅，上述规则就意味着在每行文本的周围有额外的4磅的空间。这一额外值被分成两半，一半在行上，另一半在行下。然而两行间的距离是16磅，这就说明了额外的距离是怎样分配的。

现在让我们来看一个稍微复杂一点的的例子。如下所示：

```
BODY {font-size: 10pt;}
P {font-size: 18pt; line-height: 23pt;}
BIG {font-size: 250%;}

<P>This paragraph's 'font-size' is 18pt, and the 'line-height' for this
paragraph is 23pt. However, a <BIG>larger element</BIG> within the
paragraph does not cause the value of 'line-height' to change, which can
lead to some interesting effects.</P>
```

图4-26中的结果看上去可能像是浏览器错误，其实不然。它正是例子中的标记所应该显示的。

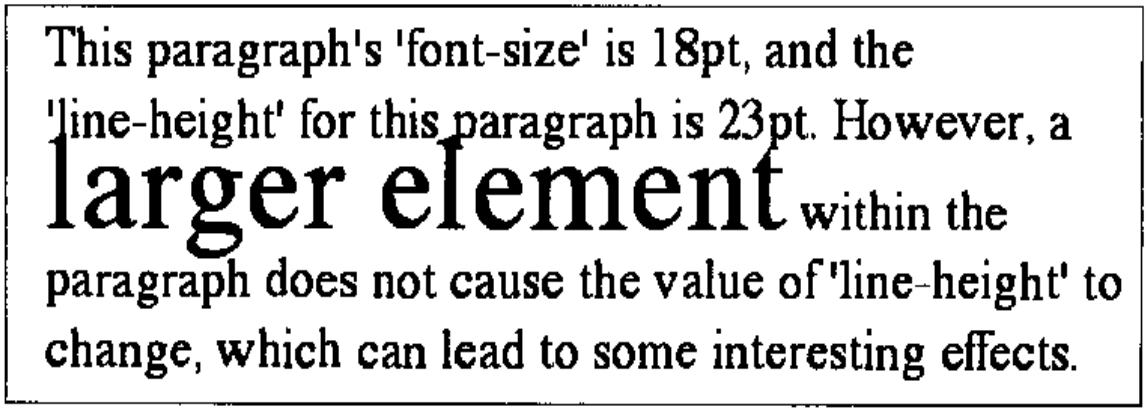


图4-26 不同尺寸的内联元素及行高属性可能引发的行为

这决不是行高所引起的唯一的怪现象。正如在图4-27中所看到的那样，元素中每一行的行高都一样，而不管它们实际上显得有多远。这在下一节，“纵向对齐”中还会提到。

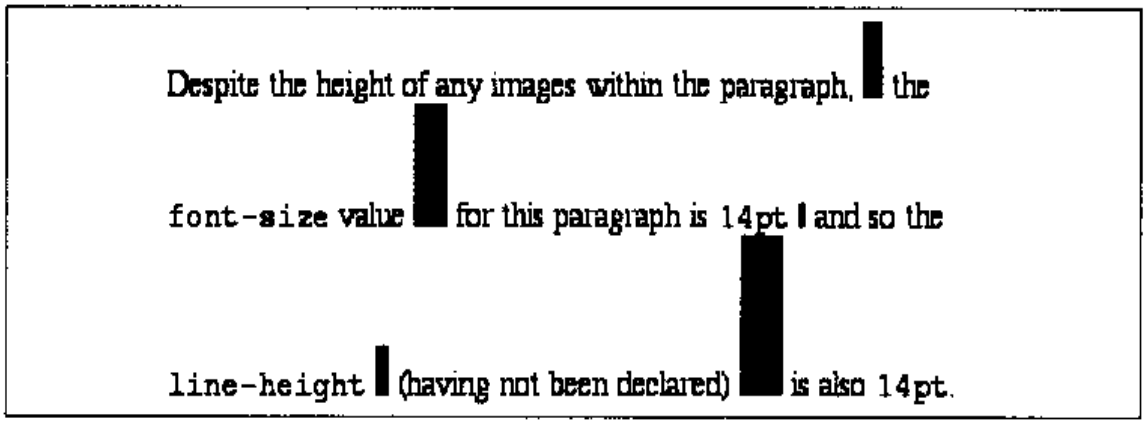


图4-27 高的图像并不改变行高

图4-27中的实际行高对每行来说都一样。而且对任何元素，包括内联元素都可以设置行高，我们正好可以利用这样一个事实，再次回到前面的那个例子，将其做一点修改，在BIG元素的样式中加一个行高。其结果如图4-28所示：

```
BODY {font-size: 10pt;}
P {font-size: 18pt; line-height: 27pt;}
BIG {font-size: 250%; line-height: 1em;}
```

```
<P>This paragraph's 'font-size' is 18pt, and the 'line-height' for this
paragraph is 27pt. A <BIG>larger element</BIG> within the paragraph does
not cause the line's height to change, but setting its 'line-height' does,
which leads to some interesting effects.</P>
```

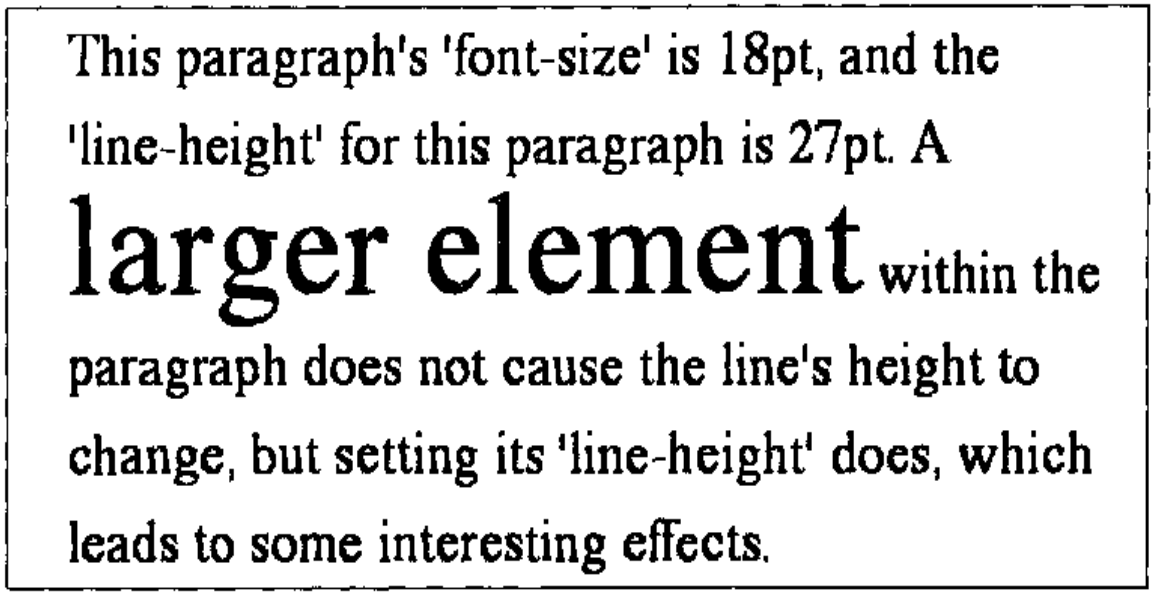


图 4-28 改变内联元素的行高

设置BIG元素的行高为1em将导致其行高同BIG文本本身的尺寸一样，而图像也具有同样的效果。它将文本行展得足够开，以致于其间的文本可以清楚地显示出来。在这种情况下，其效果是由于增加了内联元素BIG的行高而导致的，与图像本身的尺寸相对应，但其效果基本一样。

为什么会发生这种情况，其原因是很复杂的。如需了解更多的细节，请参见第八章中关于内联格式化模型的有关讨论。

实际问题

前面介绍过，缩放因子是避免由于行高的长度测量而引起继承问题的最好方法。因此，好像使用一个数字更加合算。其实也未必。Internet Explorer 3 会将缩放因子解释为一个像素值，从而得到与图 4-29 类似的结果。

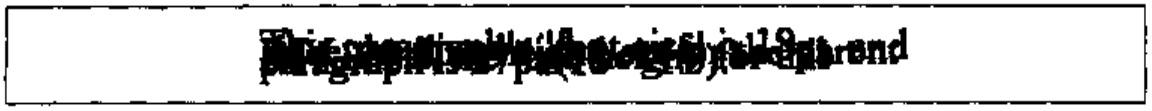


图 4-29 Internet Explorer 3 中的行高因子会带来大麻烦

根据CSS规范，用户代理可以为缺省的关键字normal设置任何最佳值，但推荐范围为1.0到1.2，这依赖于显示媒体和正在使用的字体。大多数浏览器似乎都使用接近1.2的值，但这会在新浏览器，甚至旧浏览器的新版本中加以改变。

纵向对齐

可能读者在某种程度上已经熟悉了文本的纵向对齐。如果读者以前使用过SUP和SUB元素（上标和下标元素），或者使用过<IMG SRC="foo.gif" ALIGN="texttop">标记的图像，那么其实就已经接触过基本的纵向对齐属性了。CSS的vertical-align（纵向对齐）属性允许内联元素的所有对齐方式。

警告：有件事要记住：vertical-align不能影响表格单元中的内容的对齐，对于块级元素中的内容也一样。在CSS1下，复制<TD valign="top">这样的标记是不允许的。（这会在CSS2中得到改变；请参见第十章“CSS2展望”中的“表格”一节。）

因此，让我们看看vertical-align能做些什么。这一属性只适用于内联元素，尽管它包含了诸如图像和表单输入等替换元素，而且它不能被继承。

vertical-align

允许值	baseline | sub | super | bottom | text-bottom | middle | top | text-top | <百分比>
初始值	baseline
可否继承	否
适用于	内联元素

注意：百分比是指相对于元素的行高。

vertical-align接受任何八个关键字之一，或者是百分比值，但不能同时取二者，关键字中有些我们很熟悉，有些则较陌生：baseline（缺省值）、sub、super、bottom、text-bottom、middle、top和text-top。我们将对其一一进行讲解。

基线对齐

vertical-align: baseline使元素的基线同父元素的基线对齐，这也是浏览器已经实现了的、常用的方式。例如，下面的标记将产生如图 4-30 所示的结果:

```
B {vertical-align: baseline;}

<P>The baseline of the <B>boldfaced text</B> is aligned with the baseline
of this paragraph.</P>
```

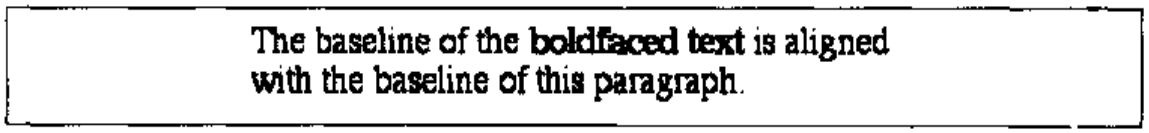


图 4-30 基线对齐

检查图 4-30，会发现它和想像中的差别不大，也不应该会有太大的差别。

如果纵向对齐的元素没有基线 —— 即，如果它是一个图像，一个表单输入，或是一个替换元素 —— 那么元素的底部就会同父元素的基线对齐，如图4-31所示:

```
IMG {vertical-align: baseline;}

<P>The image found in this paragraph <IMG SRC="dot.gif" ALT="a dot"> has its
bottom edge aligned with the baseline of the paragraph.</P>
```

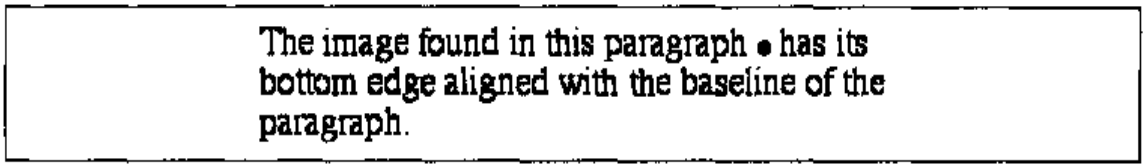


图 4-31 图像的基线对齐

上标和下标

声明vertical-align: sub使元素为“下标”。通常，这意味着元素的基线（或底端，如果是替换元素）相对于其父元素的基线被降低了。然而降低的距离在规范中没有定义。因此，它会随用户代理的不同而改变。注意sub并不隐含元素字体尺寸的变化。因此，它不会使变为下标的文本变小（或变大）。反而，任何下标元素中的文本，缺省情况下，都应该与段落中的文本尺寸一样，如图 4-32 所示:

```
SUB {vertical-align: sub;}

<P>This paragraph contains <SUB>subscripted</SUB> text.</P>
```

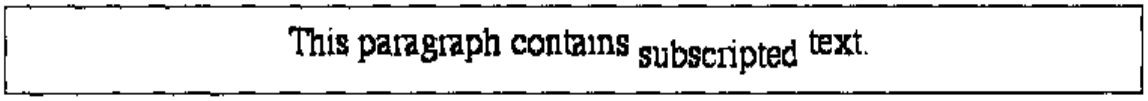


图 4-32 下标对齐

当然，可以用字体尺寸属性使文本变小，但必须自己动手。

上标（super）与下标类似，元素的基线（或替换元素的底端）相对于父元素的基线被升高了。同样，被升高的距离取决于用户代理，对字体尺寸没有隐含的变化。如图 4-33 所示：

```
SUP {vertical-align: super;}

<P>This paragraph contains <SUP>superscripted</SUP> text.</P>
```

图 4-33 上标对齐

底端对齐

vertical-align: bottom似乎很简单，元素的内联框的底端同行框的底端对齐。例如，下面的标记将产生图 4-34 所示的结果：

```
IMG.feeder {vertical-align: bottom;}

<P>This paragraph contains first a single fairly <IMG SRC="tall.gif" allgn="middle"
ALT="tall image"> which is tall, and then an image <IMG SRC="short.gif"
CLASS="feeder" ALT="short image"> which is not tall.</P>
```

正如从图 4-34 中看到的，段落的第二行包含两个图像，它们的底端互相对齐。（第一个图像用 HTML 的 align 属性设置成中间对齐。我们将在本章的后面来看用 CSS 取得类似效果所使用的方法。）

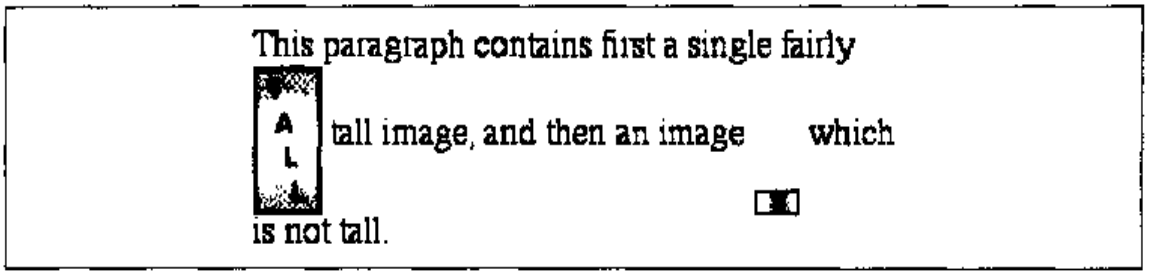


图4-34 底端对齐

vertical-align: text-bottom指文本行的底端。替换元素，或根本没有文本的任何元素都会被忽略。取而代之的是，“缺省”的文本框会被考虑。这个缺省的文本框源于父元素的字体尺寸。对齐元素的内联框的底端将与缺省文本框的底端对齐。于是下列标记得到如图4-35所示的结果：

```
IMG.tbot {vertical-align: text-bottom;}

<P>Here: a <IMG SRC="tall.gif" ALIGN="middle"
ALT="tall image"> tall image, and then a <IMG SRC="short.gif"
CLASS="tbot" ALT="short image"> short image.</P>
```

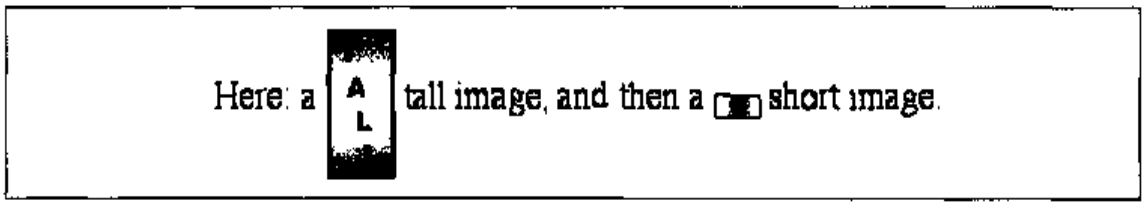


图4-35 文本底端对齐

顶端对齐

使用vertical-align: top与bottom的效果刚好相反。同样，-align: text-top同text-bottom相反。图4-36显示了下面标记所产生的结果：

```
IMG.up {vertical-align: top;}
IMG.textup {vertical-align: text-top;}

<P>Here: a <IMG SRC="tall.gif" ALIGN="middle"
ALT="tall image"> tall image, and then a <IMG SRC="short.gif"
CLASS="up" ALT="short image"> short image.</P>
<P> Here: a <IMG SRC="tall.gif" CLASS="textup"
ALT="tall image"> tall image, and then a <IMG SRC="short.gif"
CLASS="textup" ALT="short image"> short image.</P>
```

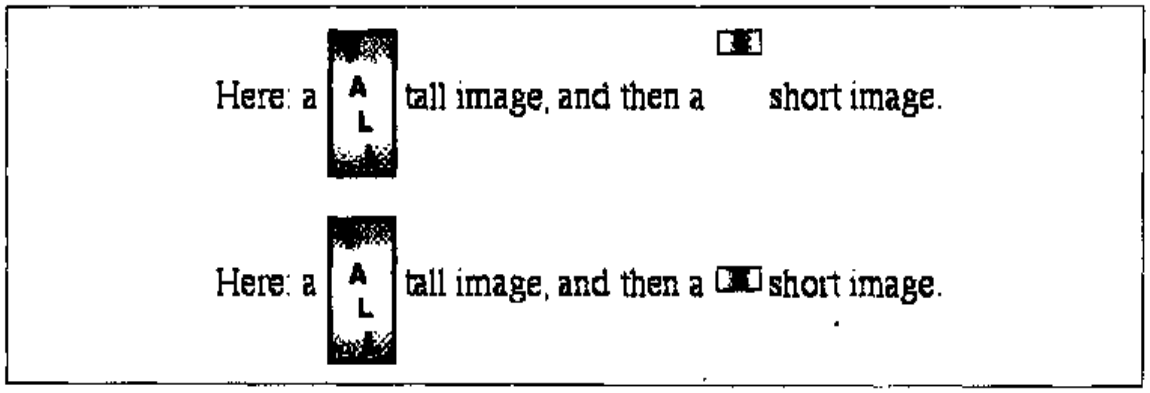


图 4-36 顶端对齐和文本顶端对齐

当然，对齐的准确位置依赖于行中的元素和它们的高度。

中间对齐

vertical-align: middle这个值通常适合于图像，因为它使元素的纵向的中点对齐本行的“中间”。行的中间定义为基线以上半个 ex 处的那个点，而 ex 的值源于父元素的字体尺寸。如图 4-37 中的例子：

```
IMG {vertical-align: middle;}
```

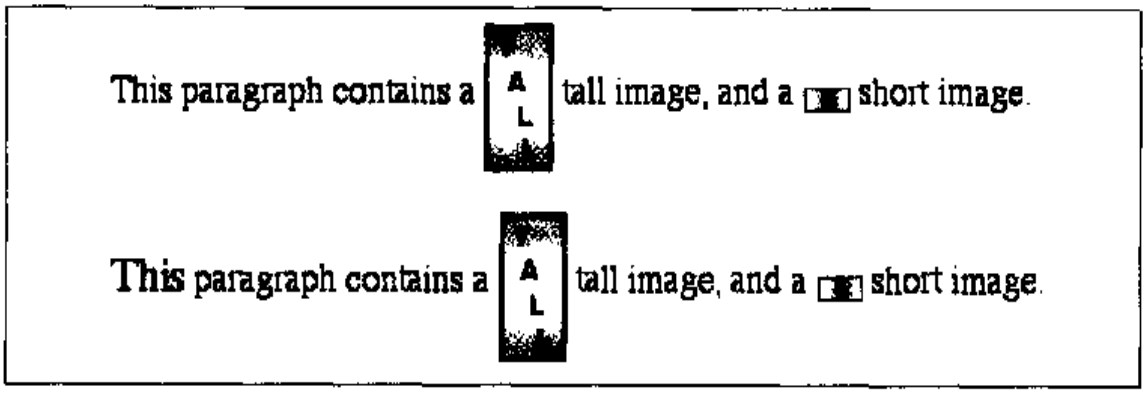


图 4-37 中间对齐

在实际应用中，因为大多数用户代理将 1ex 等同于 0.5em，middle 将会使元素的纵向中点与父元素基线以上四分之一 em 处的一个点对齐。详细情况见图 4-38。

这很像是在模仿 <IMG ALIGN="middle">，正如所看到的，使用百分比值或许会更相似。

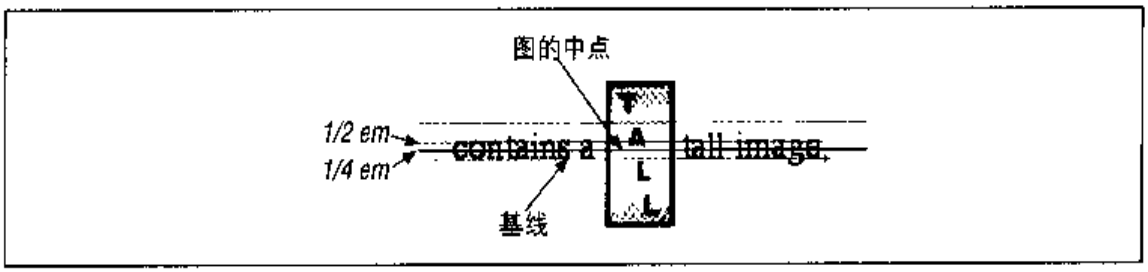


图 4-38 中间对齐的精确细节

百分比

对于图像，百分比不能用于模仿ALIGN="middle"，而设置vertical-align为百分比值会使元素的基线（或替换元素的底端）根据给定的值被升高或降低，当然是相对于父元素的基线。所指定的百分比是作为行高的值的百分数来计算的。正百分数使元素上升，而负百分数使元素降低。这可以使被上升和降低的元素好像是分别位于两个相邻的行上，如图4-39所示，但在使用百分比值时一定要小心：

```
B {vertical-align: 100%;}

<P>This paragraph contains some <B>boldfaced</B> text, which is raised
up 100%.  This makes it look as though it's on a preceding line.</P>
```

is the end of the previous paragraph.
This paragraph contains some **boldfaced** text, which is raised up 100%.
This makes it look as though it's on a preceding line

图 4-39 百分比值的纵向对齐

然而，认识到纵向对齐的文本并不是另一行的一部分十分重要。当没有其他文本时，它看起来是这样的，其实不然。如图4-40所示，这里，某些纵向对齐的文本出现在段落的中间了。

This paragraph contains a lot of text to be dislpayed, and a part of that **boldfaced text**
text is some which is raised up 100%. This makes it look as though the boldfaced text is on a preceding line, but notice thet the line is actually taller than ordinary.

图 4-40 百分比对齐能够影响行的高度

当然，这也可以带来某些有趣的视觉效果，如图 4-41 所示：

```
SUB {vertical-align: -100%;}
SUP {vertical-align: 100%;}

<P>We can either <SUP>soar to new heights</SUP> or, instead,
<SUB>sink into despair...</SUB></P>
```

既然百分比值是行高的百分数，那么当行中的一个特别高的图像使行的高度增加时会发生什么呢？

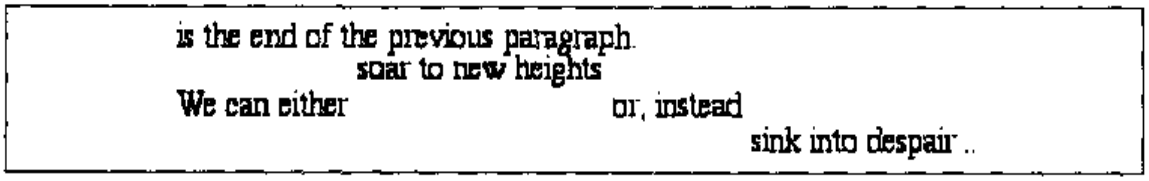


图 4-41 百分比和有趣的效果

如果再回忆一下前面介绍的，我们就会知道答案，不管这个图像有多高，它都不会使实际的行高增加，增加的是行框的高度。因此，如果行的高度为 14px，行中的某个元素的纵向对齐属性设置为 50%，而且在行中有一个 50 像素高的图像，那么会得到如图 4-42 所示的结果：

```
SUP {vertical-align: 50%;}
```

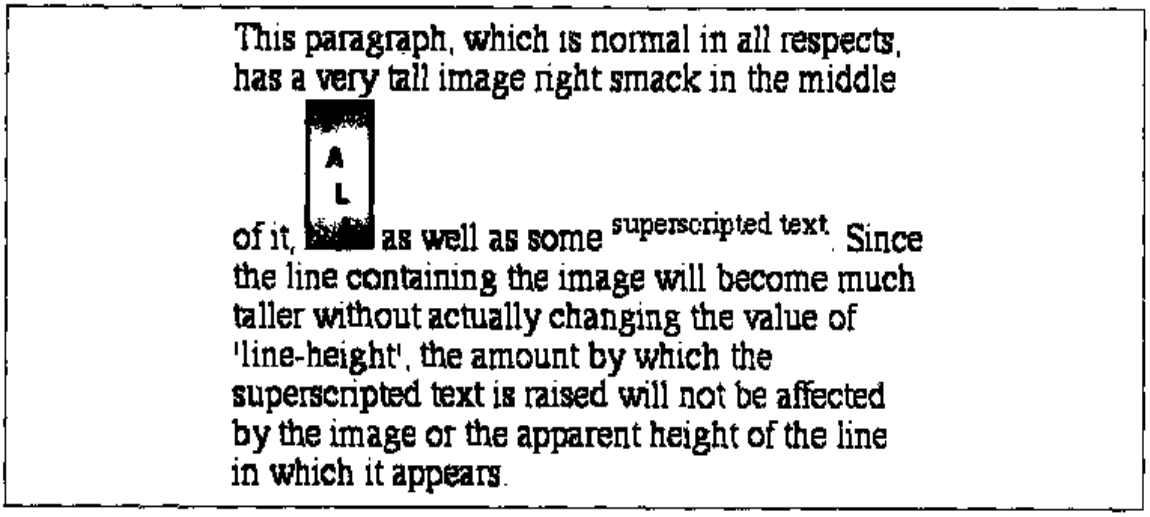


图 4-42 百分比的纵向对齐和高的图像

这个 50% 对齐的元素的基线上升了 7 个像素（14px的一半），而不是 25 个像素。

注意行框被扩展得足够大以放置这个图像。这实际上和内联框模型是一致的，因为替换元素具有这种效果。

如果有两个图像，其中一个是带百分比值的纵向对齐，那么我们对纵向对齐操作能看得更清楚。如图 4-43 所示，其结果依赖于行高，其值显式声明为 14px:

```
P {line-height: 14px;}
IMG.up {vertical-align: 50%;}

<P>Some <IMG SRC="tall.gif" alt="tall image">
<IMG SRC="short.gif" CLASS="up" ALT="short image"> text.</P>
```

较小图像的底端（或基线）上升了行高的一半（50%），或者说 7 个像素。如果设置 IMG.up的对齐属性为 -50%，那么较小的图像将降低 7 个像素。

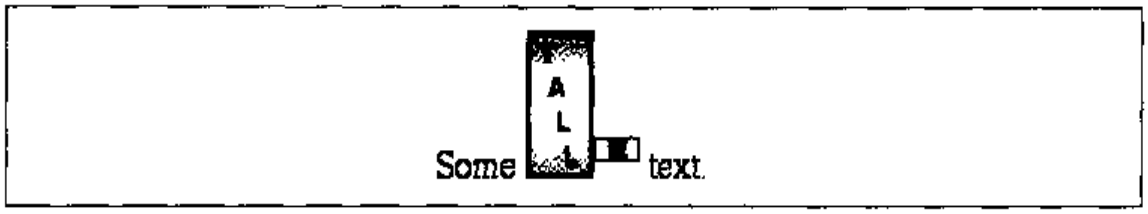


图 4-43 百分比的纵向对齐和两个图像

组合对齐

正如我们所看到的，给定某行文本，其显示方式主要依赖于那一行上各种元素的纵向对齐。基于这一事实，我们再次强调这一点。例如，某行包含一个中间对齐的图像，以及一个底端对齐的内联文本元素。那么，其结果将如图 4-44 所示:

```
IMG {vertical-align: middle;}
SUB {vertical-align: bottom;}
```

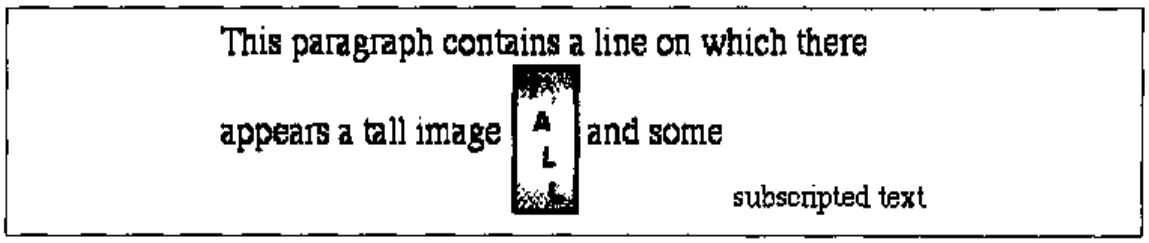


图 4-44 组合的纵向对齐

如果将文本元素改成文本底端对齐，则情况会发生根本性的变化，如图4-45所示：

```
IMG {vertical-align: middle;}
SUB {vertical-align: text-bottom;}
```

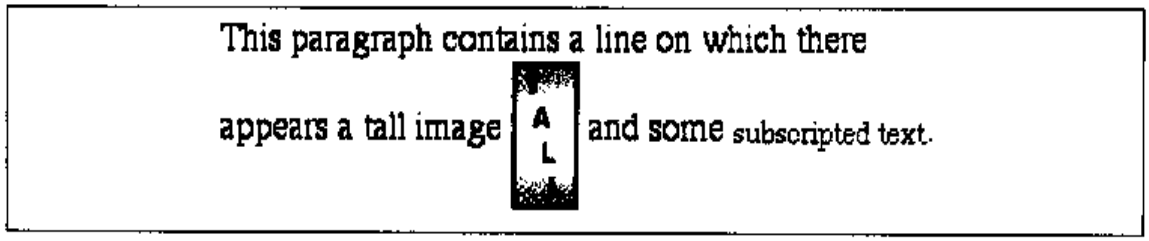


图4-45 不同方式的组合纵向对齐

文字间隔和字母间隔

它们的概念都非常简单：文字和字母间隔就是插入或删除文字间或字母间的间隔的方式。同样，对于这两个属性有一些非直观的问题需要考虑。首先让我们来讨论文字间的间隔。

文字间隔

word-spacing

允许值	<长度> | normal
初始值	normal
可否继承	是
适用于	所有元素

文字间隔（word-spacing）属性可以接受正的或负的长度值。这个值被加到文字的间隔中，这或许并不是读者所希望的。在效果上，文字间隔属性用作文字间间隔的一个调节器。因此，缺省值normal与设置一个零值（0）的效果一样，如图4-46所示：

```
P.base {word-spacing: normal;}
```

```
P.norm {word-spacing: 0;}
```

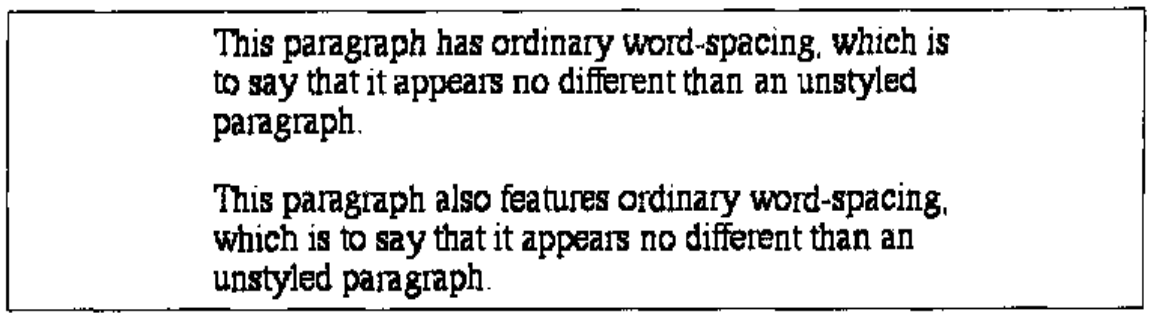


图 4-46 两种方式得到普通的文字间隔

所以，如果我们应用一个正值，文字间隔就会增大，如图 4-47 所示：

```
P {word-spacing: 0.2em;}

<P>The spaces between words in paragraphs will be increased by 0.2em.</P>
```

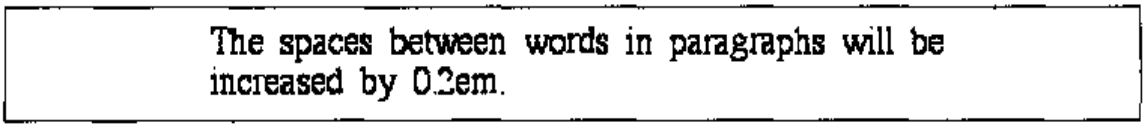


图 4-47 增加文字间间隔

通过设置负值，文字会变得更紧凑。如图 4-48 所示：

```
P {word-spacing: -0.4em;}

<P>The spaces between words in paragraphs will be decreased by 0.4em.</P>
```

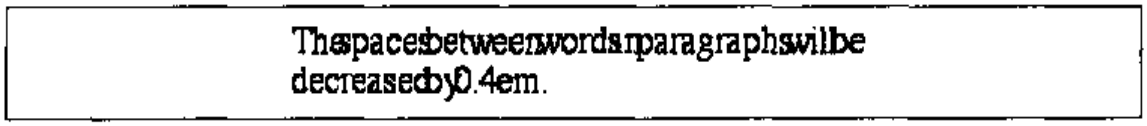


图 4-48 减小文字间间隔

到此为止，我们将给出“文字”的精确定义。在最简单的CSS的术语中，一个“文字”就是指任何不含空格符的字符串，而且这个字符串的两端都是空白。这个定义并没有真正的语义上的含义——它只不过假定当制作者要写作一个文档时，里面的每个字都要用空白符来进行分隔。很明显，支持CSS的用户代理不可能决定在某种给定的语言中，什么是有效的文字，什么不是。

这也就意味着任何使用象形文字或其他非罗马书写风格的语言是不可能利用这一属性的，而且这种“文字”的字义也不是用户代理需要遵从的；它只不过代表了定义文字的最基本方式，用户代理可以使用更为精确的方式来决定什么是文字，什么不是。

当然，这一属性可能会产生很难阅读的文档，如图 4-49 所示：

```
P {word-spacing: 1in;}
```

The spaces between
words in paragraphs will
be increased by a
whole inch. This would
make an ordinary
document almost completely
unreadable.

图 4-49 实在太宽的文字间隔

字母间隔

letter-spacing

允许值	<长度> | normal
初始值	normal
可否继承	是
适用于	所有元素

字母间隔同文字间隔一样有着同样的问题。它们之间唯一的区别是，字母间隔属性是字符或字母间的空白符数量的调节器。

它所允许的值为任何长度值和缺省关键字 normal（它同 letter-spacing: 0的效果一样）。任何长度值将会使字母间隔增加或减少所声明的数量。图 4-50 显示了下面的标记所产生的结果：

```
P {letter-spacing: 0;}    /* 同'normal'一样 */
P.spacious {letter-spacing: 0.25em;}
P.tight {letter-spacing: -0.25em;}

<P>The letters in this paragraph are spaced as normal.</P>
<P CLASS="spacious">The letters in this paragraph are spread out a bit.</P>
<P CLASS="tight">The letters in this paragraph are smooshed together a bit.</P>
```

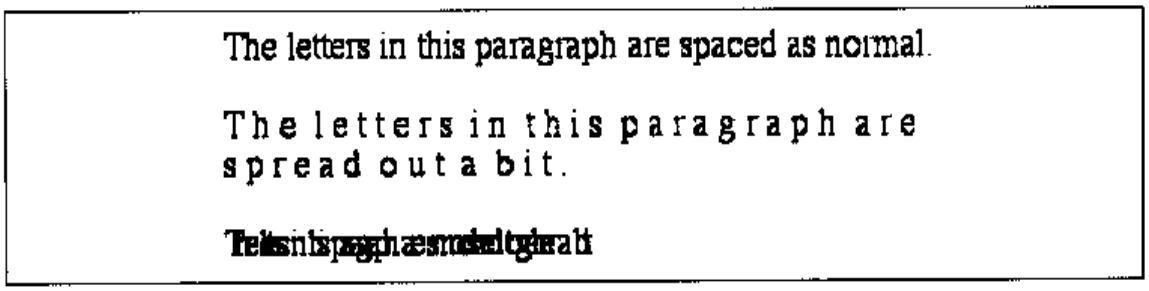


图 4-50 各种字母间隔

字母间隔的一个有趣的用法就是增加强调文本的字母间隔，这种技术在早些年很常用。可以用下面的声明来得到如图 4-51 所示的效果：

```
STRONG {letter-spacing: 0.2em;}

<P>This paragraph contains <STRONG>strongly emphasized text</STRONG>
which is spread out for extra emphasis.</P>
```

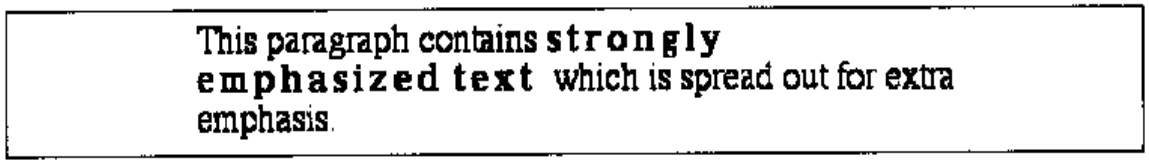


图 4-51 使用字母间隔来增加强调文本的间隔

间隔、对齐和字体尺寸

文字间隔和字母间隔都要受到文本对齐的值的影响。如果某个元素被设置成两端对齐，那么字母和文字间隔可以被改变，以允许两端完全对齐，反过来，它也可以改变制作者对文字间隔和字母间隔的声明值。CSS规范并未对这种情况下的间隔该如何计算做任何规定，因此用户代理可以自由地对此做出处理。

而且，元素的计算值通常被其子元素所继承。不像行高那样，对于文字间隔和字

母间隔没有办法为其定义一个缩放因子，以取代计算值而被继承。这样就会出现如图 4-52 所示的问题。

```
P {letter-spacing: 0.25em;}
SMALL {font-size: 50%;}

<P>This spacious paragraph features <SMALL>tiny text which is just
as spacious</SMALL>, even though the author probably wanted the
spacing to be in proportion to the size of the text.</P>
```

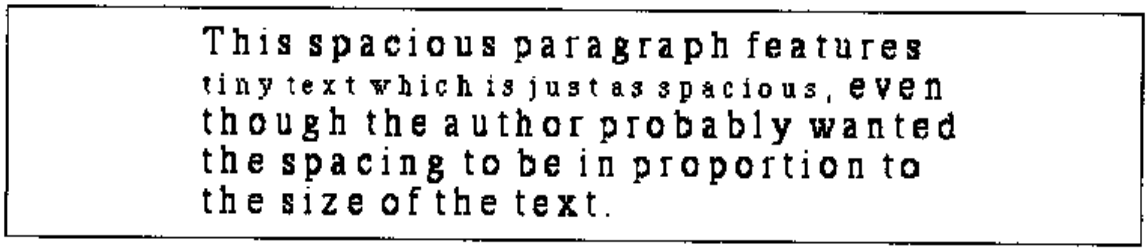


图 4-52 被继承的字母间隔

使字母间隔同文本尺寸相称的唯一方法就是显式地设置它们，下面标记的结果如图 4-53 所示：

```
P {letter-spacing: 0.25em;}
SMALL {font-size: 50%; letter-spacing: 0.25em;}

<P>This spacious paragraph features <SMALL>tiny text which is
proportionally spacious</SMALL>, which is what the author
probably wanted.</P>
```

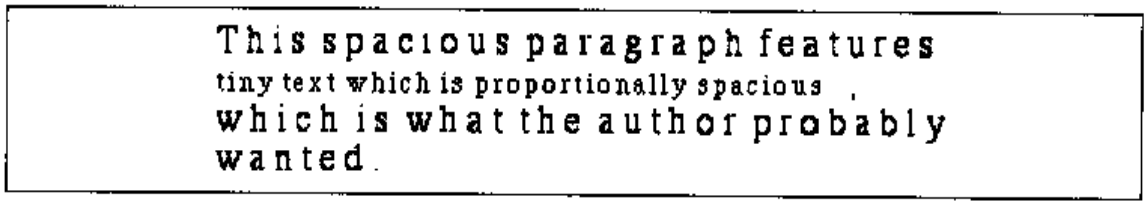


图 4-53 解决被继承的字母间隔的问题

文本转换

现在我们来看处理文本大小写的方法。这是通过文本转换（text-transform）属性来完成的。

text-transform

允许值	uppercase | lowercase | capitalize | none
初始值	none
可否继承	是
适用于	所有元素

缺省值none将使用源文档中已经存在的大小写。uppercase和lowercase将使整个文本变为大写或小写字母，正如它们的名字所指出的那样。最后，capitalize使每个文字的第一个字母大写。(注意在本属性中所提到的“文字”同文字间隔一节中的定义一样。)

图4-54以不同的方式显示了这些设置:

```
STRONG {text-transform: uppercase;}
H1, H2, H3 {text-transform: capitalize;}
P.cummings {text-transform: lowercase;}
P.raw {text-transform: none;}

<H1>The heading-one at the beginninG</H1>
<P>
By default, text is displayed in the capitalization it has in the source
document, but <STRONG>it is possible to change this</STRONG> using
the property 'text-transform'.
</P>
<P CLASS="cummings">
For example, one could Create TEXT such as might have been Written by
the late Poet E.E.Cummings.
</P>
<P CLASS="raw">
If you feel the need to Explicitly Declare the transformation of text,
that can be done as well.
</P>
```

注意，不同的用户代理决定文字首字母的方式可能不同，因此哪些字母该被大写可能也有不同。在图4-54中，H1元素里的“heading-one”文本可能以两种方式来显示:“Heading-one”或“Heading-One”。CSS对此未做规定，所以两种情况都可能出现。

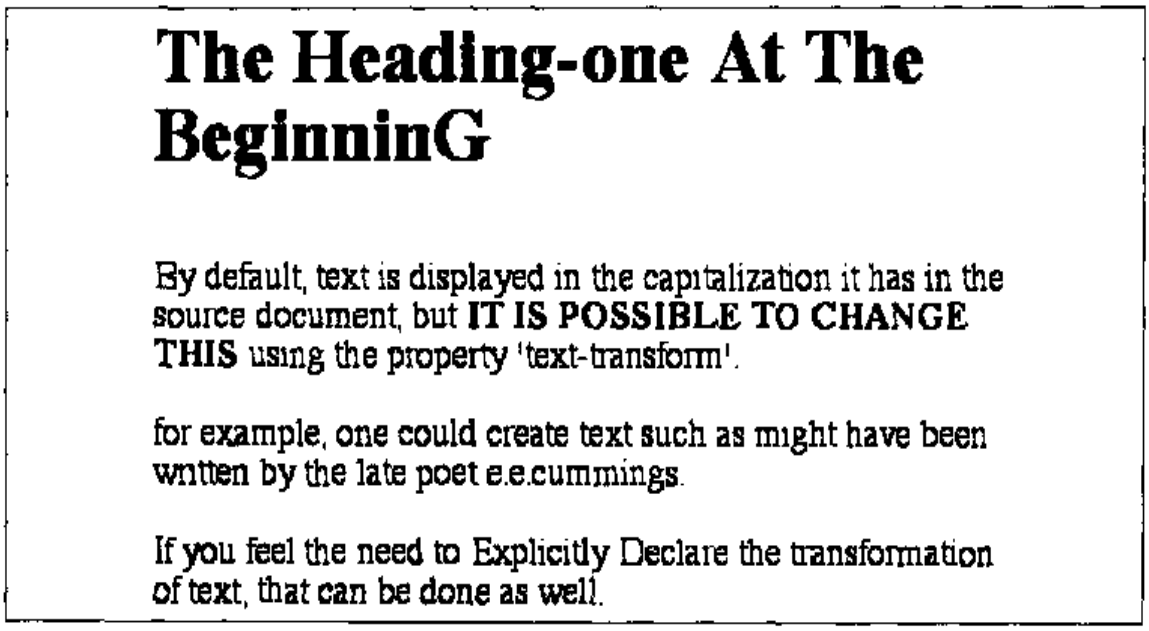


图 4-54 各种文本转换

同时注意在图 4-54 中，H1 元素的最后一个字母仍然是大写。这是正确的：当 text-transform取capitalize值时，CSS只需用户代理确定每个文字的首字母大写。它们不必对文字的剩余部分做任何改动。

作为一个属性，text-transform或许并不像它看起来那么有用。实际上，当决定要让所有的H1元素为大写时，它是非常有用的。用不着去改变所有H1元素的内容，只需用 text-transform来完成这一变化：

```
H1 {text-transform: uppercase;}

<H1>This is an H1 element</H1>
```

正如图 4-55 显示的那样，现在所有文本都为大写字母。

图 4-55 转换 H1 元素

有双重好处。首先，所有要做的只是写一条规则来做改动，而不是改变H1本身。

另外，如果后来决定将所有大写变为每个文字的首字母大写，这样的变化就更简单了，如图 4-56 所示：

```
H1 {text-transform: capitalize;}

<H1>This is an H1 element</H1>
```

This Is An H1 Element

图 4-56　以另一种不同的方式转换 H1 元素

文本修饰

最后我们来讨论文本修饰（text-decoration）属性，这是个很有吸引力的属性，同时也为浏览器带来很多麻烦。然而，首先让我们讨论在理论上它应该怎样工作。

text-decoratien	
允许值	none | [underline || overline || line-through || blink]
初始值	none
可否继承	否
适用于	所有元素

正如所期望的的那样，underline 就像 HTML 中的 U 元素一样，使元素带下划线。overline 正好相反，它使元素内的文本带上划线。line-through 在文本的中部划一条直线，也称之为删除线，等价于 HTML 中的 S 和 STRIKE 元素。blink 使文本闪烁，如同 Netscape 支持的 BLINK 标签一样。图 4-57 显示了这些值的结果：

```
P.emph {text-decoration: underline;}
P.topper {text-decoration: overline;}
P.old {text-decoration: line-through;}
P.annoy {text-decoration: blink;}
P.plain {text-decoration: none;}
```

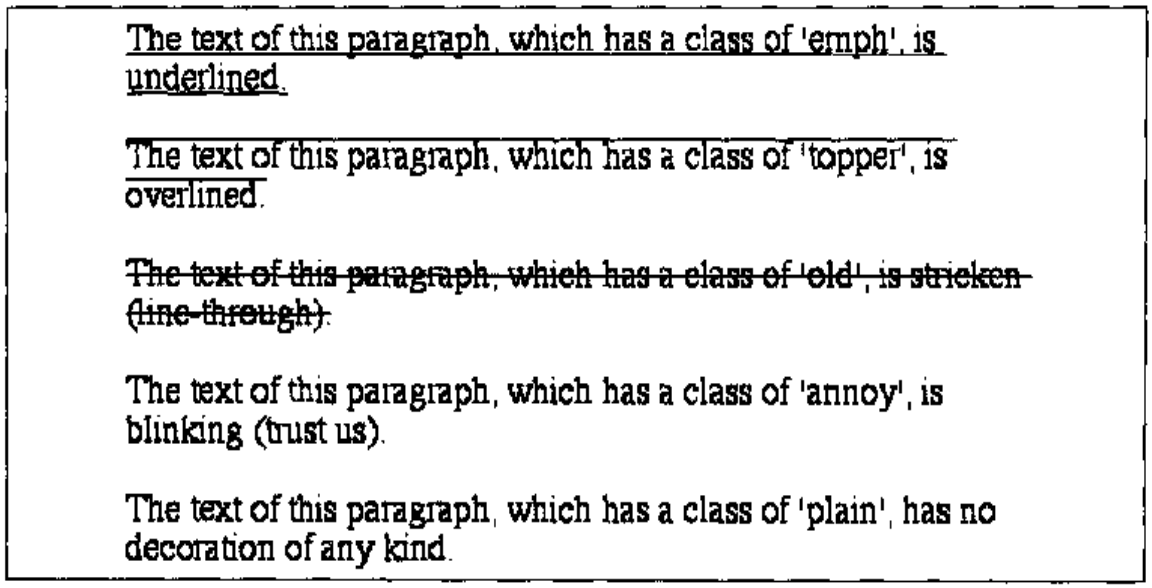


图 4-57 各种文本修饰

警告：当然不可能在打印中使用闪烁效果，这是很容易想到的。顺便说一句，用户代理不必支持闪烁，而在写本书时也只有 Navigator 4.x 支持闪烁。

正如图 4-57 的最后一部分所示，值 none 将关掉任何文本修饰。通常这是文本的缺省外观值，但也并不总是这样。例如，链接的缺省为下划线，如果想取消超链接的下划线，可以用下面这条 CSS 规则：

```
A:link {text-decoration: none;}
```

图 4-58 中的文本包含了三个超链接：三个列表项。因为我们显式地取消了链接的下划线，普通文本和链接之间能见到的区别就是颜色。

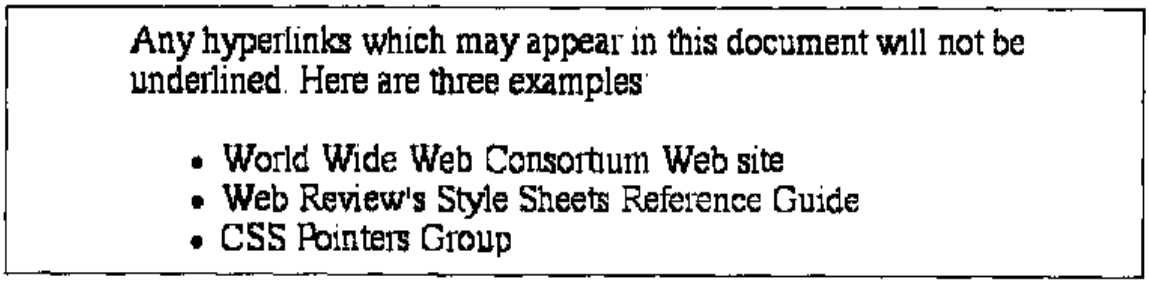


图 4-58 隐藏超链接的下划线

注意：尽管我个人对此毫不介意，但许多用户却因取消链接的下划线而感到恼火。很明显，这只是一个建议而已，用户可以选择适合自己的方式——但记住，如果链接的颜色与普通文本的对比不够明显，对于某些弱视的用户来说就很难找到文档中的超链接。

在一个规则中组合多个文本修饰也是可以的。例如，想让所有的超链接既有下划线又有上划线，可用下列规则：

```
A:link, A:visited {text-decoration: underline overline;}
```

这可使text-decortation具有速记的特性。必须注意这一点，因为，如果同时有两个不同的文本修饰匹配同一元素，那么，胜出的那条规则将会完全取代另一条规则。如下：

```
H2.stricken {text-decoration: line-through;}
H2 {text-decoration: underline overline;}
```

给定以上两条规则，那么任何带有stricken类的H2元素将只有删除线的文本修饰。下划线和上划线修饰就没有了，因为速记值会用一个替换另一个，而不是累积。

奇异的修饰

现在让我们来看一些关于文本修饰的奇怪的东西。首先，文本修饰不能被继承。这隐含着文本中所划的任何修饰线（要么在下面，要么在上面，或在中间）都应该同父元素的颜色一致，即使子元素是另一种颜色也一样，如图4-59所示：

```
P {text-decoration: underline; color: black;}
B {color: gray;}

<P>This paragraph, which is black and has a black underline, also contains
<B>boldfaced text</B> which has the black underline beneath it as well.</P>
```

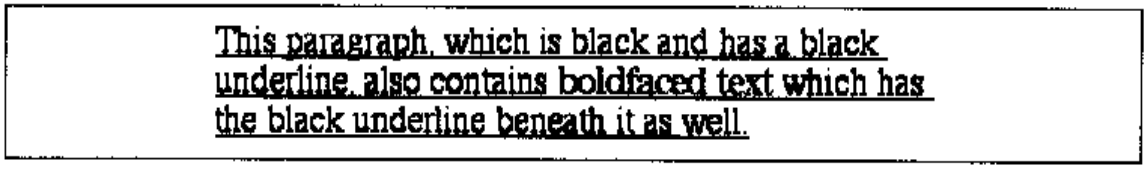


图4-59 下划线的颜色一致

为什么这样呢？因为文本修饰的值是不能被继承的，B元素有缺省值none，因此，B元素就没有下划线。当然，很清楚，B元素下面有一条线，因为它没有下划线看上去会很愚蠢。然而并非如此。B元素下面的线是段落的下划线，它“跨越”了B元素。通过改变粗体字元素的样式能使这更加明显。

```
P {text-decoration: underline; color: black;}
B {color: gray; text-decoration: none;}

<P>This paragraph, which is black and has a black underline, also contains
<B>boldfaced text</B> which has the black underline beneath it as well.</P>
```

显式声明B元素没有任何文本修饰后，其结果一样。换句话说，没有方法能够“去掉”元素里的下划线（上划线或删除线）。如果某个元素设置了一个文本修饰，那么它的所有子元素也会应用这样一个修饰。

如果将文本修饰同纵向对齐组合使用，会发生更奇怪的事情。图4-60显示了其中一个怪现象。虽然SUP元素没有任何文体修饰，但它却包含于另一个带上划线的元素中，以致于这条上划线从中穿过SUP元素：

```
P {text decoration: overline; font-size: 12pt;}
SUP {vertical-align: 50%; font-size: 12pt;}
```

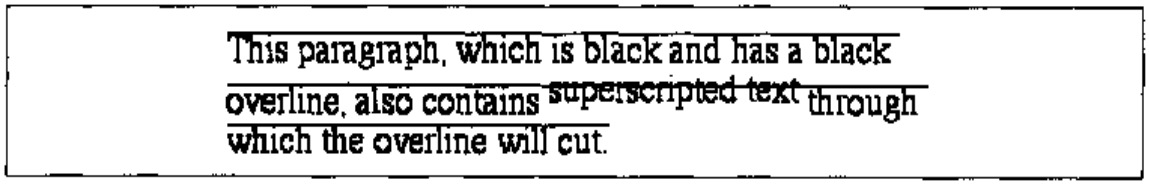


图4-60 正确的修饰行为，尽管有些奇怪

基于上面的这些问题，读者或许对使用文本修饰感到绝望了，觉得它使情况变得更糟了（可能是更好）。迄今为止，我们已经探究了规范中所描述的属性的工作方式。在实际工作中，许多浏览器确实会关掉子元素的下划线，即使它们实际不这样去做。它们违背规范的原因很简单：制作者的期望。

```
P {text-decoration: underline; color: black;}
B {color: gray; text-decoration: none;}

<P>This paragraph, which is black and has a black underline, also contains
<B>boldfaced text</B> which does not have black underline beneath it.</P>
```

如图4-61所示，网络浏览器关掉了B元素的下划线。如果有显式声明text-decoration: none来禁止上划线，那么Navigator，Explorer和Opera都会这样去做。这当然是出于制作者的期望，而且也是浏览器这样去做的原因。

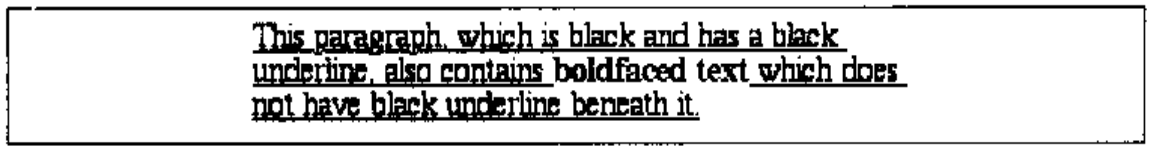


图4-61 浏览器在实际中的做法

或许将来的某一天，浏览器（或其他用户代理）会严格遵守规范。如果总是依赖于使用none来取消修饰，那么它可能会反过来成为束缚。而且，未来的CSS版本应该包含能关闭修饰的方法，而不是错误地使用none属性值，因此还是有希望的。

最后，还有一种在不违背规范的情况下改变修饰颜色的方法。回忆前面讲过的，设置元素的文本修饰属性意味着整个元素都会有同样的修饰颜色，即使有不同颜色的子元素。为了匹配各个元素的颜色修饰，需要像下面这样显式声明颜色修饰：

```
P {text-decoration: underline; color: black;}
B {color: gray; text-decoration: underline;}

<P>This paragraph, which is black and has a black underline, also contains
<B>boldfaced text</B> which has its own gray underline.</P>
```

在图4-62中，B元素被设置成灰色且有下划线。灰色的下划线覆盖了父元素的黑色下划线，因此修饰颜色也就与B元素的颜色相匹配。

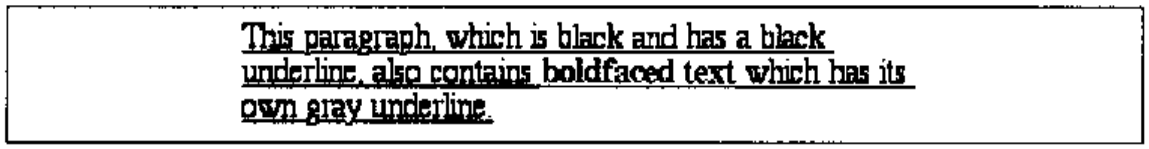


图4-62 解决下划线的缺省行为

小结

无需改变字体，就有许多方法可以改变文本的外观，其中包括传统的效果，如下

划线。当然，CSS 也给了我们在文本的上面和中间划线的能力；可以改变文字和字母间的间隔大小；段落（或其他块级元素）的首行缩进；文本的左或右对齐等等。我们甚至还可以改变文本行间的间隔大小，尽管这种操作很复杂，在第八章将介绍它的一些细节。

这些行为基本上都到了相对较好的支持，要不然就是根本不支持。文本的两端对齐就是其中一个未得到很好支持的属性，而且大多数老版本的用户代理在文本修饰和纵向对齐中会出错，行高的计算也是如此。另一方面，文字和字母间隔几乎能正常工作，而文本缩进在使用中也有一些小的毛病。大小写转换通常来说都得到了很好的支持。

当然，对于文本，除了改变其大小、粗细和对齐方式等外观外，还有一件制作者通常想做的事情，就是改变文本的字体。我们将在下一章里讨论这一问题。

第五章

字体属性

正如规范的制订者们认识到的，字体属性将是一个流行的CSS特性。有千千万万个网页中散乱着几十个，甚至上百个 <FONT face="...">标签。事实上，CSS1的“字体属性”一节是以这样一个句子开头的：“设置字体属性将是样式表中最普通的用法。”

事实是，到现在为止还没有一种方式能保证网络上字体使用的一致性，因为没有一种统一的方式用于描述字体及其变形。例如，字体Times、Times New Roman和TimesNR或许很相似，或者一样，但用户代理怎么会知道呢？制作者可能在文档中指明用TimesNR，但用户恰好没有安装这一字体怎么办呢？即使安装了Times New Roman字体，用户代理又怎能知道它们是可以互换的呢？所以，想强加给读者以某种字体，是行不通的，尽管CSS2有便利的可下载字体，但没有很好地实现，而且读者多半会由于性能原因而拒绝下载字体。CSS并不提供最终的字体控制，这一点类似于字处理程序：当一个Microsoft Office文档在另一台机器上打开时，它的显示将依据于这台机器中安装的字体。如果它没有文档中所指的相同字体，那么这个文档看起来就会不一样。这对于用CSS设计的文档是一样的道理。

字体命名问题比试图匹配字体名更为困难，而且在字体变形，如粗体或斜体文本方面，尤其令人迷惑。大多数人都知道斜体文本是什么样，但它同倾斜的文本又有什么不同呢？是的，有区别，但大部分人都很难描述出它们的区别。当然，它

们不是唯一指代这种文本的术语；还有像*Oblique*、*incline*（或*inclined*）、*cursive*和*kursiv*这些单词也可作为这种术语。这样，某种字体可以称之为*Times Italic*，而另一种字体的叫法可能是*Garamond Oblique*。尽管这两种叫法都同它们的"斜体形式"等价，但它们的标记却各不相同。类似地，字体的变形术语*Bold*、*Black*和*Heavy*或许是一回事，或许不是。

尽管不能提供一个完全的解决方案，但CSS1还是试图针对这些问题提供某些解决机制。CSS1中字体处理最复杂的部分是字体系列匹配和字体粗细匹配，以及字体尺寸计算。在CSS1中字体外观有风格（style）（例如斜体），和字体变形（如小写）；相对而言，它们显得更直接。所有这些特征都被糅合在字体（font）属性中，这将下面讨论。首先让我们来讨论字体系列，因为它们是为文档选择合适字体的最基础的一步。

字体系列

正如我们前面讲到的，标记同一种字体有很多方式，但CSS1仍试图帮助用户代理将它们分门别类。因为我们所考虑的"字体"是由许多描述其粗细、斜体等字体变形组合而成的。比如，读者可能对字体Times较熟悉。然而，Times实际上是许多变形的组合，包括TimesRegular、TimesBold、TimesItalic、TimesOblique、TimesBoldItalic、TimesBoldOblique等等。Times的每一种变形都是一个字体形式（*font face*），而我们通常所想像的Times就是这些字体形式的组合。换句话说，Times是一个字体系列，并不单单是一种字体，尽管大家都以为字体只是一种单一的实体。

除每个特定的字体系列，如Times、Verdana、Helvetica或Arial外，CSS还定义了五种一般字体系列（generic font family）：

Serif字体

成比例且有衬线（serif）的字体。一种字体有比例，是指字体中的所有字母根据它们不同的尺寸占据不同的宽度。这样，小写字母*j*和*m*就有不同的宽度（例如，本书中的缺省字体就是成比例的）。衬线特指加在字母上做装饰的细线，例如在小写字线*l*的顶端和底端，或者大写字母*A*两条腿底端的细线。衬线字体的例子有Times，Garamond和New Century Schoolbook。

Sans serif 字体

有比例但没有衬线的字体。这种字体的例子有Helvetica、Geneva、Verdant、Arial 和 Univers。

Mono space 字体

无比例的字体。它们通常用于模拟打字机打出的文本，或者老式点阵打印机和更老的影像显示终端的输出。在这些字体中，每个字符都占有同样的宽度，因此小写字母*i*和*m*所占的宽度一样。如果一种字体有统一的字符宽度，不管它有没有衬线，都可归为 monspace。例如 Courier 和 Andale Mono。

Cursive 字体

试图模拟人的笔迹的字体。通常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的。例如，大写字母*A*可能在其左腿的底端带一个小的卷曲。例如 Zapf Chancery，Author 和 Comic Sans。

Fantasy 字体

不能通过某种单一的特征来定义，而且也不能简单地归为其他系列的某一类的那些字体。这样的字体较少，如 Western 和 Klingon。

理论上，用户能够安装的每一种字体系列都应属于这些常见系列之一。实际上并非如此，但例外的情况（如果有的话）非常少。

使用一般的字体系列

通过使用字体系列（font-family）属性，可以在文档中使用任何一种字体系列。

font-family

允许值	[[<系列名> | <一般系列>],]* [<系列名> | <一般系列>]
初始值	与用户代理有关
可否继承	是
适用于	所有元素

如果读者希望在文档中使用一种sans serif（无衬线）字体，但也不特别介意是哪一种，那么正确的声明为：

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

这会使用户代理选择一种sans serif字体，如Helvetica，而且将其应用于BODY元素。由于继承，这将会应用于整个文档（除非有另外的选择符将其覆盖）。其结果与图5-1所示类似：

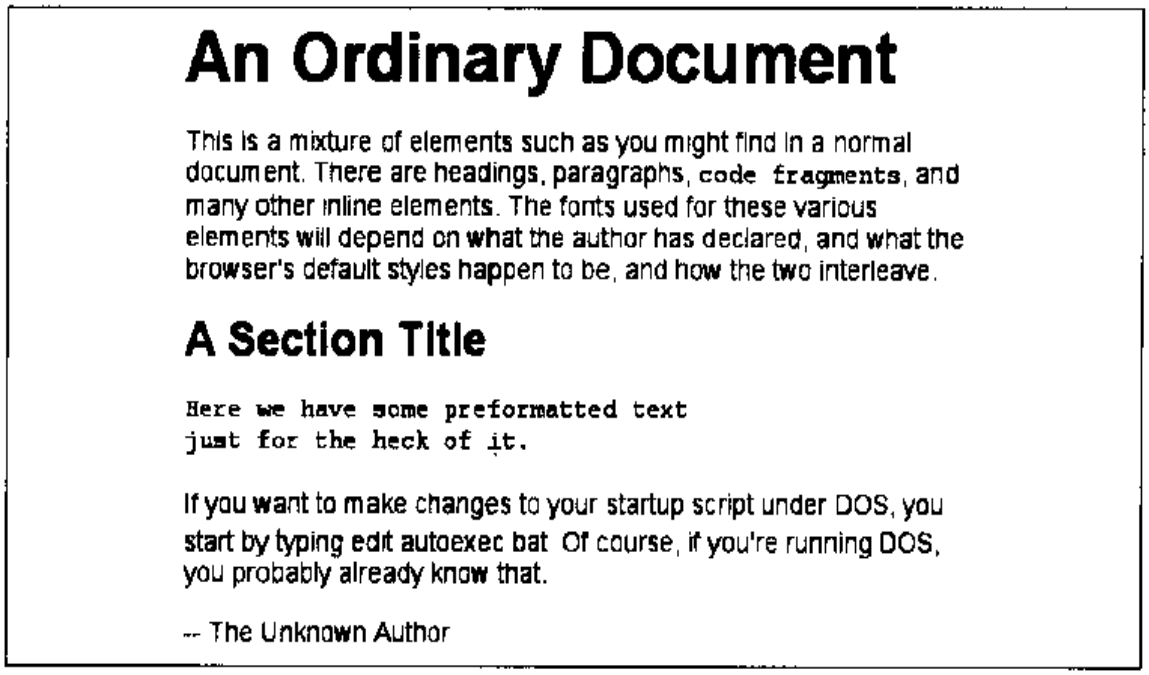


图5-1 使用sans serif字体

仅用这些一般字体系列，制作者就能够创建非常丰富的样式表。例如下面的一组规则，其结果如图5-2所示：

```
BODY {font-family: serif;}
H1, H2, H3, H4 {font-family: sans-serif;}
CODE, PRE, TT, SPAN.input {font-family: monospace;}
P.signature {font-family: cursive;}
```

这将使大部分文档设为serif（有衬线）字体，例如Times，而其中所有带signature类的段落为cursive字体，如Author。标题1到4将是sans serif字体，如Helvetica。而CODE、PRE、TT和SPAN.input将是monospace字体，如Courier——通常头三个元素都是选用这样的字体。

指定实际的字体名

另一方面，制作者可能对元素的显示字体还有更具体的设置；同理，用户可能想自己创建一个样式表以定义文档中所用的字体。对于这些情况，font-family仍然适用。

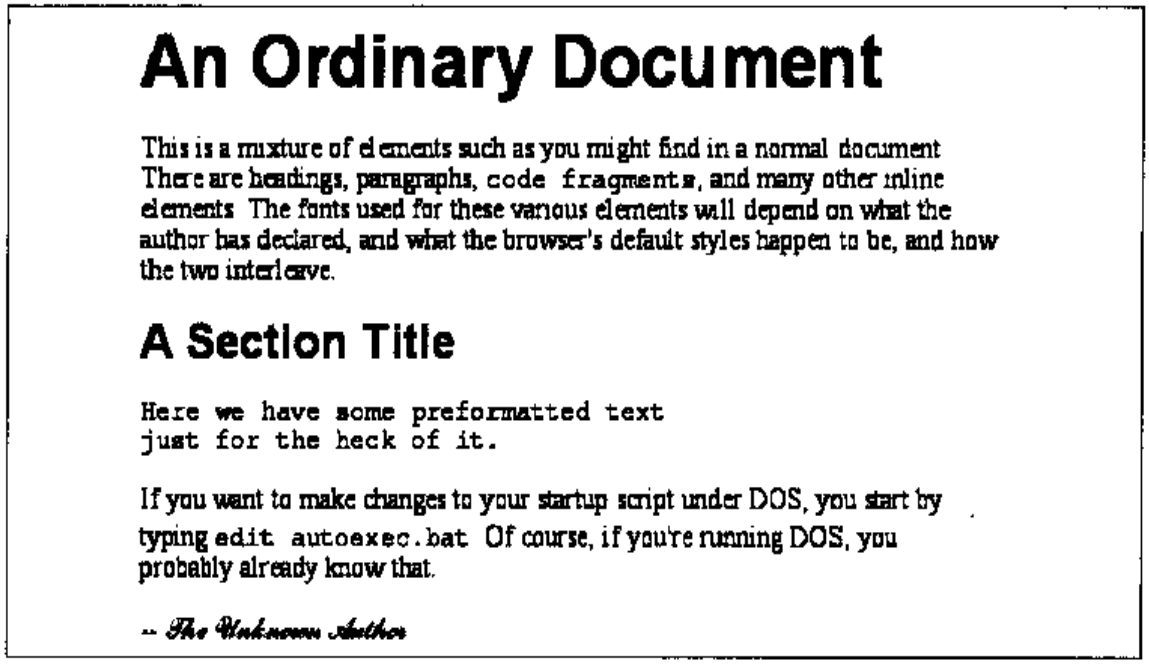


图 5-2 各种字体系列

假设现在要让所有 H1 的字体为 Garamond。最简单的规则如下：

```
H1 {font-family: Garamond;}
```

这会使用户代理用 Garamond 显示文档中的所有 H1 元素，如图 5-3 所示。

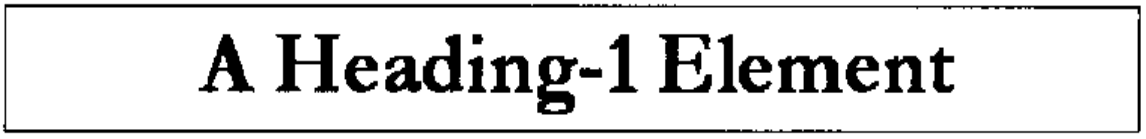


图 5-3 使用 Garamond 的 H1 元素

这里假设用户代理有Garamond字体可用。但如果因为某种原因没有这种字体，怎么办呢？遇到这种情况，用户代理就不能使用这条规则了。但它不会忽略这条规则，如果它不能找到一种叫“Garamond”的字体，那么这条规则就不会起任何作用。

然而，并不是就没办法了。通过将具体的字体名同一般字体系列组合，文档将能够尽量以接近制作者意图的方式显示。继续前面的例子，下面的标记告诉用户代理使用Garamond字体，但是如果没有这种字体，则用另外一种serif字体：

```
H1 {font-family: Garamond, serif;}
```

如果读者没有安装Garamond字体，而安装有Times，则用户代理就可能使用Times来显示H1元素，如图5-4所示。虽然这是非精确的匹配，但它能尽量接近指定的字体。

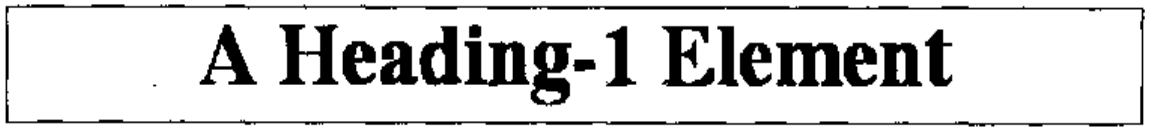


图5-4 使用浏览器选择的serif字体的H1元素

所以，制作者最好是将一般字体系列作为font-family规则的一部分。这样，可以让不能提供精确字体匹配的浏览器利用它们的反馈机制来选择另一种相似的字体。

尤其是在一种交叉平台的环境下，这种方式很有用，因为互相之间到底安装了哪些字体是不可知的。当然，全世界的Windows机器可能都安装了Arial和Times New Roman，但很多Macintosh则没有，而且Unix机器中可能没有上面的字体。相反，Chicago和Charcoal对Macintosh机器来说是很普通的，而Windows和Unix用户可能一种都没安装，同时安装了这两种字体的可能性就更小了。因此，包含这些字体或其他的字体声明，最好以一个一般字体系列作为结尾：

```
H1 {font-family: Arial, sans-serif;}
H2 {font-family: Charcoal, sans-serif;}
P {font-family: TimesNR, serif;}
ADDRESS {font-family: Chicago, sans-serif;}
```

再次强调，这不是必需的，但这是个不错的主意。

如果读者对字体很熟悉，那么也可以为给定的元素列出许多相似的字体以便使用。假如想要文档中所有段落均以Times显示，但同时也可以接受Times NR，

Garamand，New Ceutury Schoolbook 和New York（它们都属于serif字体）。首先，决定这些字体的设置顺序，然后像下面这样用逗号来分隔它们：

```
P {font-family: Times, TimesNR, 'New Century Schoolbook', Garamond,
      'New York', serif;}
```

基于这个列表，用户代理会依次去查找相应的字体。如果表中的字体都不可用，那么它将简单地挑选一个有效的serif字体。

使用引号

读者可能注意到前面例子中单引号的出现，我们以前都没见过。只有当字体名中有一个或更多空白时，引号才能在font-family的声明中使用，例如“New York”，或者字体名中包含了如“#”或“$”等符号时。在这两种情况中，整个字体名字都需要用引号括起来，以使用户代理不至于产生迷惑。（读者或许认为逗号足够了，其实不然。）这样，一个名叫Karrank%的字体就需要引号：

```
H2 {Wedgie, 'Karrank%', Klingon, fantasy;}
```

如果漏掉了引号，差别就大了，用户代理就会忽略整个字体名，尽管它仍会处理规则余下的部分。使用单个文字的字体名，像Garamond，不必使用引号，而且一般字体系列（“serif”，“monospace”等）决不要用引号。如果引用一个一般字体系列名，那么用户代理会假定制作者是想要一个指定的字体（例如“serif”），而不是一般字体系列。

至于使用哪种引号，单引号和双引号都可以。然而，如果是在STYLE属性中放置一条font-family规则，那么规则内就必须使用单引号了。如果重复使用双引号，它们就会介入属性的语法，如图5-5所示：

```
P {font-family: sans-serif;}  /* sets paragraphs to sans-serif by default */

<!-- the next example is correct (uses single-quotes) -->
<P STYLE="font-family: 'New Century Schoolbook', Times, serif;">...</P>

<!-- the next example is NOT correct (uses double-quotes) -->
<P STYLE="font-family: "New Century Schoolbook", Times, serif;">...</P>
```

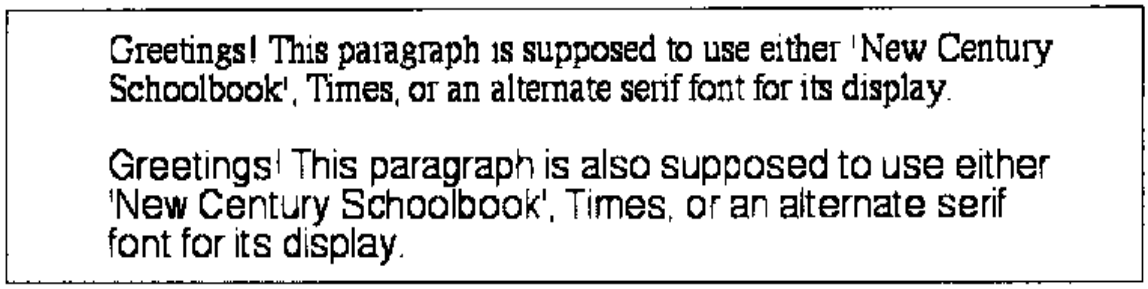


图 5-5 不正确使用引号带来的危险

实践

前面讲到过，CSS提供可选择字体：这些字体通常都来自于同一个一般字体系列，但也并不是一定要这样。可以将不同系列的字体混合在一起，而不是一定要是所有 serif 字体、sans serif 字体或 cursive 字体的字体列表。唯一的要求就是要在 font-family 声明的最后提供一个一般字体系列。

```
P.signature {Author99, ScriptTM, serif;}
```

在这里，如果 Author99 和 ScriptTM 都不可用，那么用户代理应该使用任何 serif 字体。为什么不能指定 cursive 作为一般字体系列呢？让我们将这个例子再扩展一下：

```
P {font-family: Verdana, sans-serif;}
P.signature {font-family: Author99, ScriptTM, cursive;}
```

假设这些样式应用于某一文档，而且浏览此文档的用户没有安装“signature”所列的两种字体，也没有任何 cursive 字体可用。在这种情况下，整个规则都会被用户代理忽略，而且 <P CLASS="signature"> 元素将以 Verdana 或者其他的 sans serif 字体显示，如图 5-6 所示。

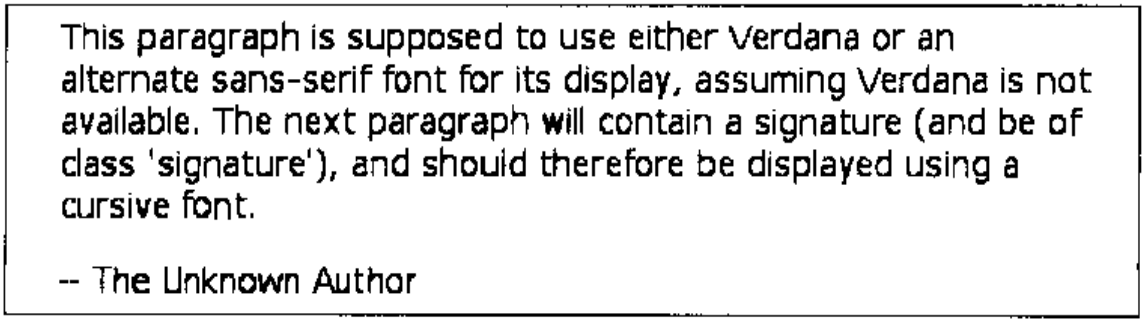


图 5-6 没有 cursive 字体的显示结果

这是因为本元素是个段落，既然它的规则不能使用，那么更一般的规则P {font-family: Verdana, sans-serif;}就应该被使用。为了避免出现这种情形，可以使用下面的规则：

```
P {font-family: Verdana, sans-serif;}
P.signature {font-family: Author99, ScriptTM, serif;}
```

这样，“signature”段落的字体就会与文档的其余部分不同，如图5-7所示。

This paragraph is supposed to use either Verdana or an alternate sans-serif font for its display, assuming Verdana is not available. The next paragraph will contain a signature (and be of class 'signature'), and should therefore be displayed using a cursive font.

-- The Unknown Author

图5-7 以防没有cursive字体

字体加粗

font-weight

允许值	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
初始值	normal
可否继承	是
适用于	所有元素

即使读者没有完全认识它，也肯定对字体加粗很熟悉了：粗体文本就是字体加粗的一个普通的例子。一般说来，字体显得越黑越粗，那么它就被认为是越重要。有很多方式可以用来标记文本的粗细。例如，被称作Zurich的字体系列就有很多种变形，如Zurich Bold、Zurich Black、Zurich UltraBlack、Zurich Light和Zurich Regular。它们都属同一基本字体，只是具有不同的粗细。

如果想在文档中使用Zurich字体，但又想充分利用它们不同的粗细。当然，可以在font-family属性中直接引用不同粗细的Zurich字体。但是其实并不需要这样去做。因为，像下面这样的样式表是很繁琐的：

```
H1 {font-family: 'Zurich UltraBlack', sans-serif;}
H2 {font-family: 'Zurich Black', sans-serif;}
H3 {font-family: 'Zurich Bold', sans-serif;}
H4, P {font-family: Zurich, sans-serif;}
SMALL {font-family: 'Zurich Light', sans-serif;}
```

而且它还必须要求每种字体都已安装，其实多数人都不会这样做。通过为文档指定一种字体系列，然后再为各种元素分配字体加粗，这会合理得多。为各种元素指定字体加粗用font-weight属性来设置。下面是一条典型的font-weight声明：

```
B {font-weight: bold;}
```

它的意义就是B元素将以粗体显示；或者说它的字体比文档中的普通文本要粗一些，如图5-8所示。当然，这也是我们平时所习惯的显示方式，因为B元素本身就使文本以粗体显示。

Within this paragraph we find a 'B' element,
which will contain **boldfaced text**, as expected.

图5-8 使B标记变粗

这实际上是字体的一种更粗的变形被用于B元素的显示。如果某个段落用的是Times字体，而且一部分是粗体，那么实际上是使用了两种字体：Times和TimesBold。普通的文本用Times字体显示，粗体文本用TimesBold。

字体加粗是怎样工作的

为理解用户代理怎样决定字体变形的粗细，得先从关键字100到900谈起，然后我们再来看它是如何继承的。这些数字关键字用于定义与字体的相关特征的映射关系，即字体的粗细被分成九个等级。例如，OpenType就使用了九个值的数字

级。字体有了这些级别之后，这些数字就直接映射到各个级，如100映射到最轻的字体变形，而900对应最重的字体变形。

事实上，在这些数字中并不存在本质的字体粗细的约定。CSS指出，每个数字对应的字体粗细不得小于它前面的数字所对应的字体粗细。这样，100，200，300和400或许都对应同样粗细的字体变形，而500和600可能对应到一个更粗的字体变形，700，800和900则对应于另一种更粗的字体变形。

这些数字被定义为同某些普通的字体变形名等价。400等价于normal，而700对应于bold。其他数字不对应任何font-weight属性的关键字，但它们可以对应于普通的字体变形名。如果某种字体变形标

记为“Normal”、“Regular”、“Roman”或“Book”，那么它便被分配给400，而且任何标记为“Medium”的字体变形就对应于500。然而，如果标记为“Medium”的字体变形是唯一可用的字体，那么它就不能同500相对应。

如果在某个给定的字体系列里少于九个字体粗细级，则用户代理需要做更多的工作。在这种情况下，它必须用一种预先定义的方式来填充其间的空隙：

- 如果值500未分配，它就被赋予同400一样的字体粗细。
- 如果300未分配，它就同比400稍轻的字体变形对应。如果没有更轻的字体可用，300就同400一样对应于某个级。这通常是在“Normal”和“Medium”情况下。同样的方法用于100和200。
- 如果600未分配，它便同比400稍重的字体变形对应。如果没有这样的字体可用，600就同500一样对应于某种字体变形。这种方法也用于700，800和900。

为了便于理解，让我们来看三个字体粗细分配的例子，首先假设字体系列Karrank%是OpenType字体，而且已经定义了九个相应的粗细级。在这里，这些数字分别对应于各个级，而关键字normal和bold各自分配给400和700。

在第二个例子里，我们考虑字体系列Zurich，它在本节的开始提到过。假设其字体变形也被分配了表5-1中所示的粗细值。

表5-1 对指定字体的假想粗细分配

字体形式	分配的关键字	分配的数字
Zurich Light		100, 200, 300
Zurich Regular	normal	400
Zurich Medium		500
Zurich Bold	bold	600, 700
Zurich Black		800
Zurich UltraBlack		900

头三个数字分配给最轻的字体。普通字体对应关键字400和normal。Medium字体分配给数字500。没有字体变形分配给600，因此将600和700一起对应于同一字体变形Bold字体。最后800和900分别分配给Black和UltraBlack字体变形。也只有当最上面的两个粗细级已经分配后，才会出现这样的情况。否则用户代理可能会忽略它们，并且将800和900分配给Bold字体，或者将它们分配给两个Black字体变形之一。

最后，让我们来看看Times字体，它只有两种字体变形，TimesRegular和TimesBold，如表5-2所示。

表5-2 Times字体的假想粗细分配

字体形式	分配的关键字	分配的数字
TimesRegular	normal	100, 200, 300, 400, 500
TimesBold	bold	600, 700, 800, 900

关键字normal和bold的分配相当直接。对于这些数字，100到300分配给Regular字体，但500怎么办呢？它被分配给Regular字体了，因为没有Medium字体；这样它就同400一样了。余下的，700总是分配给bold字体，而800和900，由于缺乏更粗的字体，也分配给Bold字体。最后，600被分配给下一个更粗的字体，当然，也只有bold字体了。

字体粗细可被继承，如果将段落设置为bold，则所有的子元素都会继承粗体，如图5-9所示。

```
P.one {font-weight: bold;}
```

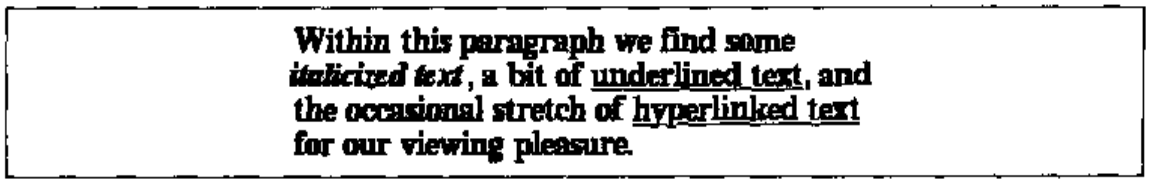


图 5-9 被继承的字体粗细

这并没有什么特别的，但是当使用最后两个值时，情况就会变得有趣了。下面我们将讨论这两个值：bolder和lighter。一般说来，这两个关键字的效果是可以想像的，它们使文本字体更粗或更细。先来看bolder值。

加粗

当设置元素的字体粗细值为bolder时，用户代理首先要决定从其父元素继承的字体粗细，然后再选择与其最接近的，而且更粗一些的字体。如果没可用的字体，那么用户代理将元素字体粗细值设置为下一个数字值。如果已到900，其值不变，仍为900。这样，可能会遭遇下列的情况，如图 5-10 所示：

```
P {font-weight: normal;}
P EM {font-weight: bolder;}  /* results in 'bold' text, evaluates to '700' */

H1 {font-weight: bold;}
H1 B {font-weight: bolder;}  /* if no bolder face exist, evaluates to '800' */

P {font-weight: 100;} /* assume 'Light' face exist ; see explanation */
P STRONG {font-weight: bolder;}  /* results in 'normal' text, weight '400' */
```

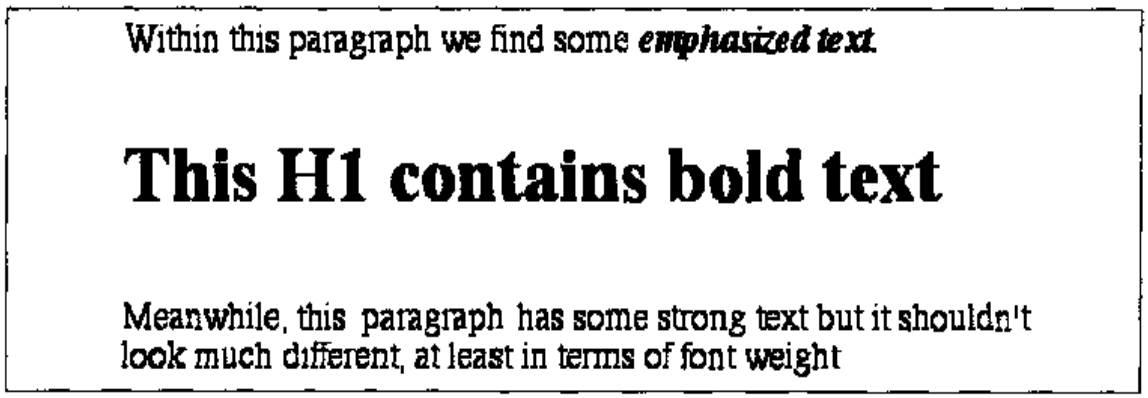


图 5-10 试图使文本变得更粗

在第一个例子中，用户代理将字体粗细从normal变为bold；用数字的方式表示就是从400改为700。在第二个例子中，H1文本已经设置成bold。如果没有更粗的字体可用，那么用户代理会将H1元素中的B文本设置成800，因为它比700（与bold等价的数字表示形式）稍大。由于800同700分配了同样的字体粗细，普通的H1文本和粗体H1文本在视觉上没有区别，但其字体粗细的数字值还是不一样的。

在最后一个例子里，段落字体被设置为最小可用的字体粗细值，假设是Light字体的某个变形。而且，这一字体系列里的其他字体为Regular和Bold。那么任何段落中的EM文本将为normal，因为这是本字体系列中下一个稍粗的字体。然而，字体系列中只有Regular和Bold字体时会怎么样呢？在这种情况下，其声明如下所示：

```
/*   assume only two faces for this example: 'Regular' and 'Bold' */
P {font-weight: 100;}   /* looks the same 'normal' text */
P SPAN {font-weight: bolder;}   /* maps to '700' */
```

100被分配给normal字体，但字体粗细的值仍为100。因此，SPAN文本（它被设置成bolder）将继承值100，然后选择下一个更粗的字体，即Bold字体，它具有数字值700。其结果如图5-11所示。

Within this paragraph we find some **SPANned text**.

图5-11 更粗的字体通常会有更显著的视觉效果

让我们再更进一步，再加两条规则和一些标记，以便说明这一切是怎样工作的（其结果如图5-12所示）：

```
/*   assume only two faces for this example: 'Regular' and 'Bold'   */
P {font-weight: 100;}   /* looks the same as 'normal' text */
P SPAN {font-weight: 400;}   /* so does this */
STRONG {font-weight: bolder;}   /* bolder than its parent */
STRONG B {font-weight: bolder;}   /* bolder still */
<P>
This paragraph contains elements of increasing weight: there is an
<SPAN>SPAN element which contains a <STRONG>strongly emphasized
element, and that contains a <B>boldface element</B></STRONG></SPAN>.
</P>
```

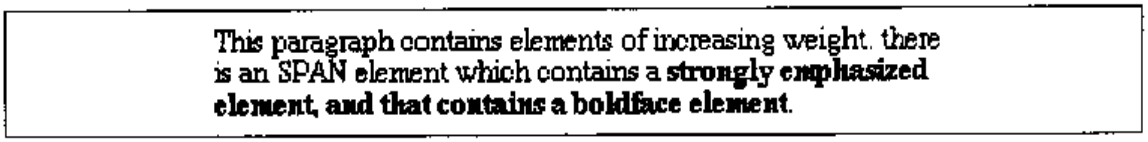


图 5-12 向上移动字体粗细级

在最后两个内联元素里，由于关键字 bolder 的使用，使字体粗细的计算值增加了。如果用每个元素的字体粗细的数字表示来代替段落中的文本，其结果将如图 5-13 所示:

```
<P>
100 <SPAN> 400 <STRONG> 700 <B> 800 </B></STRONG></SPAN>.
</P>
```

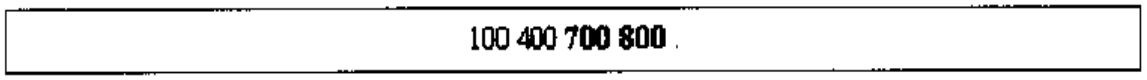


图 5-13 改变粗细，用数字值来说明

头两个值增加很多，从 100 跳到 400，另一个从 400 到 bold (700)。从 700 开始就没有更粗的字体了，因此用户代理只是简单地将字体粗细值向上移动了一个数字级 (800)。此外，如果在 B 元素中插入一个 STRONG 元素，其结果如图 5-14 所示:

```
<P>
100 <SPAN> 400 <STRONG> 700 <B> 800 <STRONG> 900
</STRONG></B></STRONG></SPAN>.
</P>
```

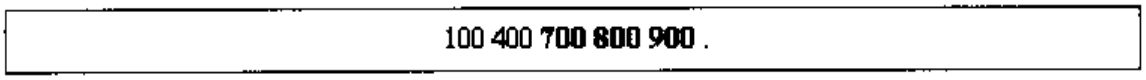


图 5-14 数字粗细值

如果还有另一个 B 元素插入到最内层的 STRONG 元素中，其字体粗细值将仍为 900，因为没有比 900 更大的了。假设我们只有两种字体可用，那么文本将只有两种显示方式，Regular 和 Bold 文本，如图 5-15 所示:

```
<P>
regular <SPAN> regular <STRONG> bold <B> bold
```

```
<STRONG> bold </STRONG></B></STRONG></SPAN>.
</P>
```

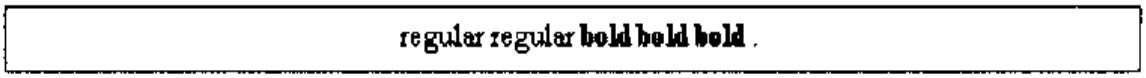


图 5-15 用描述符号表示的两种可用字体

字体变细

可以想到，ligter的方式同前面的bolder一样，只是将粗细级向下移动，而不是向上。将前面的例子稍加修改，我们就会看得很清楚：

```
/* assume only two faces for this example: 'Regular' and 'Bold' */
P {font-weight: 900;}   /* as bold as possible, which will look 'bold' */
P SPAN {font-weight: 700;}   /* this will also be bold */
STRONG {font-weight: lighter;}   /* lighter than its parent */
B {font-weight: lighter;}   /* lighter still */

<P>
900 <SPAN> 700 <STRONG> 400 <B> 300 <STRONG> 200
</STRONG></B></STRONG></SPAN>.
</P>
<!-- ...or, to put it another way... -->
<P>
bold <SPAN> bold <STRONG> regular <B> regular
<STRONG> regular </STRONG></B></STRONG></SPAN>.
</P>
```

暂且忽略直觉上的差异，我们从图5-16中可以看到，主段落文本粗细值为900，而SPAN元素为700。当STRONG元素文本设置为lighter时，它将选择下一个更细的字体，也即为regular字体，或400（与normal一样）。下一步是300，它同normal一样，因为没有更细的字体可用。从此处开始，用户代理只能每次向下移动一个数字级，直到100（例子中还未到100）为止。第二段显示了哪些文本该为bold，哪些为regular。

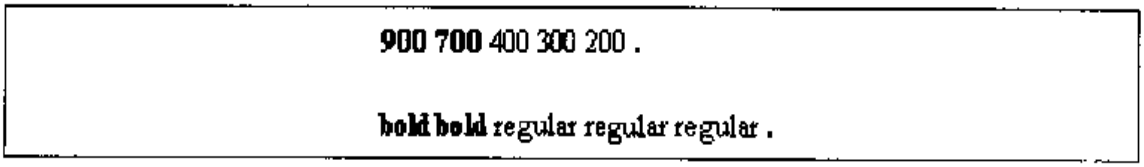


图 5-16 使文本更细

字体尺寸

决定字体尺寸的方法既让人熟悉，又与以往有所不同。

font-size

允许值	xx-small | x-small | small | medium | large | x-large | xx-large | smaller | larger | <长度> | <百分比>
初始值	medium
可否继承	是

注意：百分比是指相对于父元素的字体尺寸的百分比。

适用于	所有元素

同font-weight关键字bolder和lighter类似，font-size属性也有相对尺寸的关键字值，称为larger和smaller。同前面的相对字体粗细差不多，这些关键字使font-size的计算值在某个绝对值级别上下移动。这也是我们在探究larger和smaller之前需要理解的。首先，让我们来看字体的尺寸是如何定义的。

实际上，font-size属性同实际所看到效果之间的关系是由字体设计者来决定的。这一关系是由随同字体本身的全身方格（*em square*）（有些人也称之为全身方框）来设定的。这种全身方格，也即字体尺寸，并不参照任何字体本身的字符边界，而是参照在不带任何额外行高（CSS中的line-height）的情况下的基线之间的距离。对于字体而言，某些字符比缺省的基线间距离更高是很有可能的。这样，就可以定义字体，使其所有的字符都比其全身方格小。一些假想的例子如图5-17所示。

因此，字体尺寸的作用就是为给定字体提供某个尺寸的全身方格。这并不能保证任何实际字符将以这一尺寸显示。

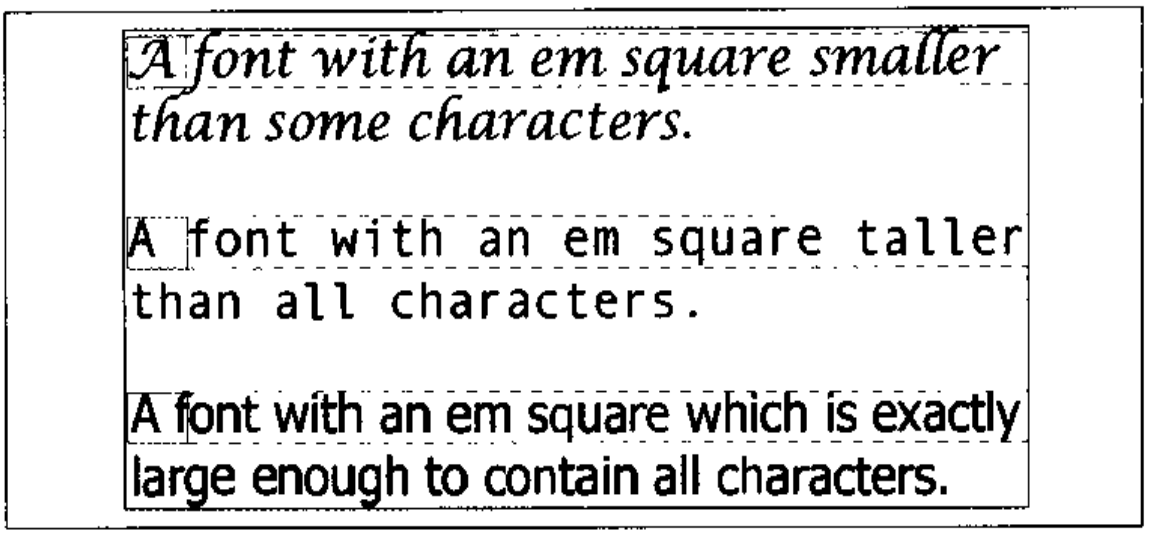


图5-17 字体及字体尺寸

绝对尺寸

建立了字体尺寸的概念后，让我们来看看绝对尺寸关键字。有七种绝对字体尺寸值：xx-small、x-small、small、medium、large、x-large和xx-large，这些都没有精确定义，只是相对而言的，如图5-18所示：

```
P.one {font-size: xx-small}
P.two {font-size: x-small;}
P.three {font-size: small;}
P.four {font-size: medium;}
P.five {font-size: large;}
P.six {font-size: x-large;}
P.seven {font-size: xx-large;}
```

根据CSS1规范，绝对尺寸之间的区别（或者缩放因子）是向上为1.5，向下为0.66。这样，如果medium为10px，那么large就是15px。另一方面，缩放因子并不一定要是1.5；不同的用户代理可能有所不同，但在CSS2中，它被规定为1.2。

假定medium为12px，对于不同的缩放因子，得到如表5-3所示的绝对尺寸（当然是估计值）。

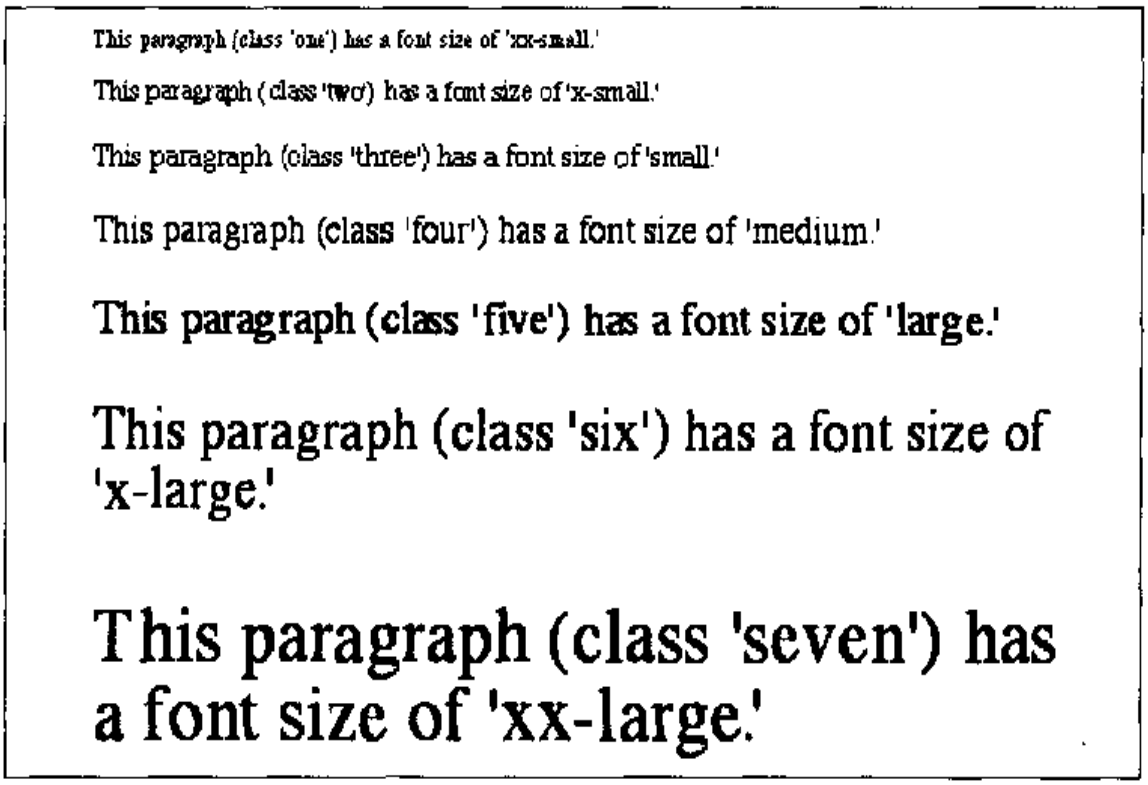


图5-18 绝对字体尺寸

表5-3 由缩放因子转换为磅

关键字	缩放因子：1.5	缩放因子：1.3	缩放因子：1.2
xx-small	4px	5px	6px
x-small	6px	7px	8px
small	8px	9px	10px
medium	12px	12px	12px
large	18px	16px	14px
x-large	27px	21px	17px
xx-large	42px	27px	20px

通常，这些尺寸在浏览器中被提前计算，而且一旦算出后就不会发生变化。因此，如浏览器将x-large认为是27px，那么不管发生什么它始终为这个值。其实这并不好，因为当用户修改浏览器的设置后，这些绝对尺寸应该重新计算，这样才显得更合理（而且和CSS的目标也更一致）。

更为复杂的情况是，不同的用户代理能为“缺省”字体尺寸分配不同的绝对关键字。拿Navigator 4来说：它使非样式文本尺寸为medium，而Internet Explorer则认为small尺寸的文本与非样式文本等价。尽管font-style的缺省值应该为medium，Internet Explorer的行为并不像它当初那样无法变通（注1）。

相对尺寸

同前面进行比较，关键字larger和smaller更容易理解：它们使元素的尺寸相对于其父元素的绝对尺寸变大或变小，而且使用和计算绝对尺寸时所用的一样的缩放因子。换句话说，如果浏览器使用1.2的绝对尺寸的缩放因子，那么它在应用相对尺寸关键字时也使用同样的缩放因子。如下：

```
P {font-size: medium;}
STRONG, EM {font-size: larger;}

<P>This paragraph element contain <STRONG>a strong-emphasis element
which itself contains <EM>an emphasis element that also contains
<STRONG>a strong element.</STRONG></EM></STRONG></P>

<P> medium <STRONG>large <EM> x-large
<STRONG>xx-large</STRONG></EM></STRONG></P>
```

由于font-size的操作方式的不同，字体的尺寸逐渐变大（或变小），如图5-19所示。

不像字体粗细的相对值，相对尺寸值不必约束于绝对尺寸的范围。因此，字体尺寸可以超出XX-small和XX-large的值。例如：

```
H1 {font-size: xx-large;}
EM {font-size: larger;}
```

注1：注意这里有七种绝对尺寸的关键字，正如有七种FONT尺寸（例如，<FONT SIZE="5">）一样。因为典型字体尺寸的缺省值为3，因此将CSS绝对尺寸列表中的第三个值用作字体尺寸的缺省值就显得比较合理。因为这个关键字是small，所以Explorer会做这样的假定。在技术上，微软已经包含了这一缺省样式：BODY {font-size: small;}。唯一覆盖它的方法只有显式声明medium值，但文档的字体尺寸在Internet Explorer和Navigator中就有所不同。

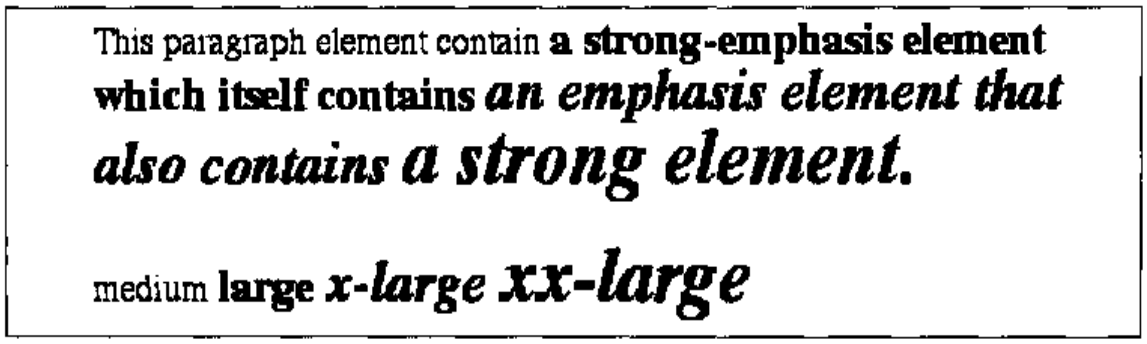


图5-19 相对字体尺寸

```
<H1>A Heading with <EM>Emphasis</EM> added</H1>
<P>This paragraph has some <EM>emphasis</EM> as well.</P>

<H1> xx-large <EM> xx-large </EM> xx-large </H1>
<P> medium <EM>large </EM> medium </P>
```

从图5-20中可以看到，H1元素中的强调文本比XX-large的尺寸还大。缩放的数量留给用户代理去决定，只是提出一个推荐缩放因子1.2。段落中EM文本的尺寸也相应地移到下一个绝对尺寸值（large）。

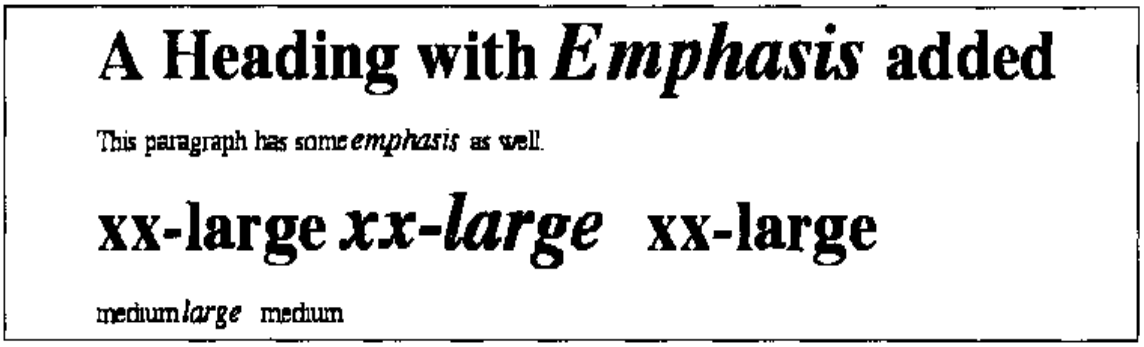


图5-20 处于绝对尺寸边缘的相对字体尺寸

百分比和尺寸

从某种程度上说，百分比值非常类似于相对尺寸的关键字，因为百分比值的计算总是与元素从父元素那里继承的尺寸相关。但百分比不像相对尺寸的关键字，允许对字体尺寸的计算值很好地进行控制，如图5-21所示：

```
P {font-size: 12px;}
EM {font-size: 120%;}
STRONG {font-size: 135%;}
SMALL, .fnote {font-size: 75%;}
```

```
<P>This paragraph contains both <EM>emphasis</EM> and <STRONG>strong
emphasis</STRONG>, both of which are larger than their parent element.
The <SMALL>small text</SMALL>, on the other hand, is smaller by a quarter.</P>
<P CLASS="fnote">This is a 'footnote' and is smaller than regular text.</P>

<P> 12px <EM> 14.4px </EM> 12px <STRONG> 16.2px </STRONG>  12px
<SMALL> 9px </SMALL> 12px </P>
<P CLASS="fnote"> 9px </P>
```

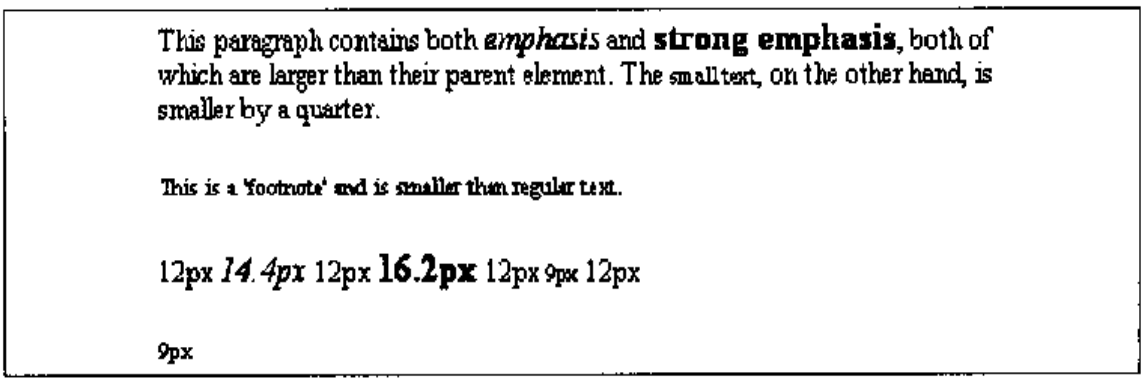


图 5-21 百分比与相对尺寸结合

在这个例子中，显示出来了确切的像素值。在实际应用中，网络浏览器一般不会舍入取整，如 14px，而是使用小数值尺寸。

顺便说一句，CSS 定义 em 的长度值与百分比值等价，因为 1em 和 100% 是一个意思。因此下列的规则会产生相同的结果：

```
P.one {font-size: 160%;}
P.two {font-size: 1.6em;}
```

当使用 em 来量度时，将应用与百分比一样的规则，例如计算值的继承等等。

字体尺寸和继承

在 CSS 中，尽管字体尺寸是可被继承的，但继承的是计算值，而不是百分比，如图 5-21 所示。因此，被 STRONG 元素所继承的值为 12px，而且这个值将因声明值为 135% 而重新计算为 16.2 像素（它可能会被舍入成 16 个像素）。

与相对尺寸关键字相比，百分比是可累积的。因此下面的标记将产生如图 5-22 所示的结果：

```
P {font-size: 12px;}
EM {font-size: 120%;}
STRONG {font-size: 135%;}

<P>This paragraph contains both <EM>emphasis and <STRONG>strong
emphasis</STRONG></EM>, both of which are larger than the paragraph text. </P>

<P> 12px <EM>14.4px <STRONG>19.44px</STRONG></EM> 12px  </P>
```

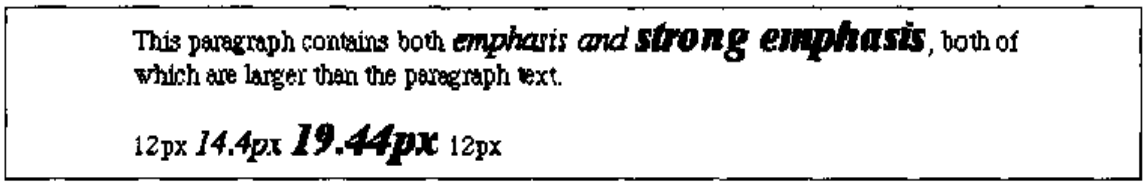


图 5-22 继承问题

图 5-22 中 STRONG 元素的尺寸是这样计算的:

12px × 120% = 14.4px
14.4px × 135% = 19.44px（可能舍入为 19px）

然而，在另外一种情况下，最后的值可能稍有不同。在这种情况下，用户代理将像素尺寸舍入，然而这些舍入值将被子元素正常地继承。因此，有下列计算方式:

12px × 120% = 14.4px [14.4px ≈ 14px]
14px × 135% = 18.9px [18.9px ≈ 19px]

如果用户代理每步都进行舍入，那么两种情况下所得的结果会一样: 19 个像素。然而，越来越多的百分比累积起来，舍入的误差就会增加。但这不应该是个主要问题 —— 毕竟，CSS 并不能保证对任何事情都能精确控制 —— 但它仍是个需要考虑的因素。

使用长度单位

字体尺寸可以使用在第三章“单位和值”中讲到的任何长度单位。在 72 磅每英寸（dpi）的环境下，下列所有字体尺寸的声明将等价:

```
P.one {font-size: 36pt;}   /* assuming 72dpi, these are all the same thing */
```

```
P.two {font-size: 3pc;}
P.three {font-size: 0.5in;}
P.four {font-size: 1.27cm;}
P.five {font-size: 12.7mm;}
```

要使其正确显示，用户代理必须知道在显示环境中使用的是多少点每英寸。不同的用户代理使用不同的值——某些基于操作系统，某些基于用户设置，而某些是基于写用户代理的程序员的假定。然而，这五行应该总是同一尺寸。因此，当结果和现实并不匹配时（例如，P.three的实际尺寸可能不到半英寸），它们之间的量度应该保持一致。

还有另外一个值与刚才讨论的那些尺寸一样，那就是36px，如果显示环境是72像素每英寸（ppi），它将和它们具有同样的物理尺寸。然而，很少还有监视器具有这样的设置。大多数设置会更高，从96ppi到120ppi。尽管如此，许多Macintosh网络浏览器趋向于将磅（pt）和像素点（px）同等对待，因此14pt和14px在Macintosh上看来差不多。但对于Windows和其他平台来说，情况就不是这样了，这也是为什么在文档设计中很难将点作为量度单位的主要原因之一。

操作系统之间的差异是作者选择使用像素值作为字体尺寸的主要原因。在文本和图像混合在一起的网页上，这种方法尤为吸引人，因为文本能够（理论上）和图像元素设置相同的高度。这可以通过声明font-size: 11px; 或其他相似的方式来达到，如图5-23所示。

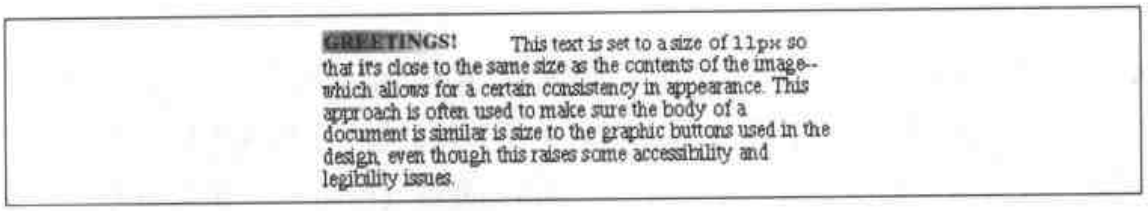


图5-23 用像素尺寸保持文本和图像的比例

使用像素量度，是获得字体尺寸“一致”的方法之一（实际上，不只是对字体尺寸，对于任何长度都可以），但要注意，当用户代理错误地将“像素”定义成“最小的点”时，会发生怎样的情况呢？在一个300dpi的环境下，它会使文本变成原来的二十分之一大。另外，像素尺寸文本在不同的监视器上、不同的分辨率下将有不同的物理尺寸，因此在使用px作为有效量度的文档中禁止使用诸如in和cm

这样的度量。实际上，在设计网页时，最好避免混用长度单位。本章中介绍的其他方法，例如关键字和百分比，都是更稳妥（或用户友好）的方法。

样式和变形

同前面所讨论的相比，本节实际上比较简单。这里讨论的属性非常直接、非常简单，读者在此可以稍稍放松。在讨论字体属性之前，先讨论font-style和font-Variant。

字体样式

font-style非常简单：它用于在normal文本、italic文本和oblique文本中做选择，就这么简单。稍微复杂一点儿的就是了解italic文本和oblique文本之间的区别，以及浏览器如何对这些字体样式予以支持。

font-style	
允许值	italic | oblique | normal
初始值	normal
可否继承	是
适用于	所有元素

font-style的缺省值为normal。它指代“直立”文本，或许可以描述为“非斜体的或非倾斜的文本”。例如，本书中的文本绝大部分都是直立的。

剩下的就只有italic和oblique文本间的区别了。对于这一点，最容易的方法就是参看图5-24，它很清楚地说明了其间的区别。

基本上，斜体文本还是保持了原有的字体，只是在每个字母结构上作了点儿小变化以获得外观的变化。衬线字体就是这样，除了文本字符倾斜外，它的衬线也可

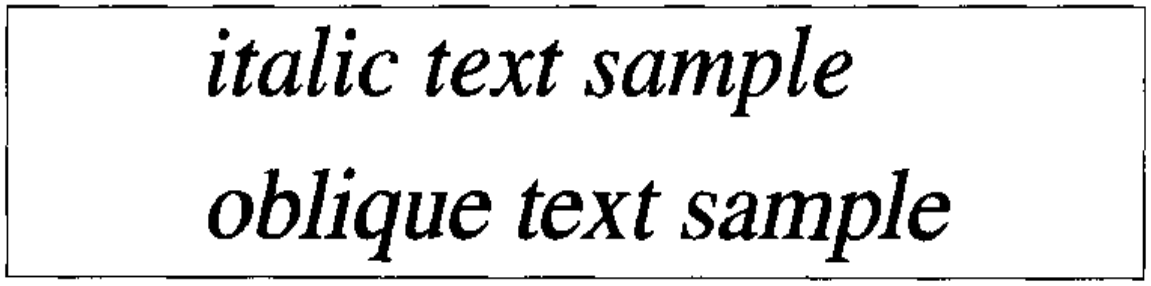


图5-24 斜体与倾斜的文本详细比较

以变为倾斜的。另一方面，倾斜的文本只是普通的直立文本的简单的倾斜。像标有Italic、Cursive和Kursiv的字体，通常都映射为italic关键字，而标有Oblique、Slanted和Incline的字体则与Oblique对应。

如果要在文档中使用斜体文本，可以像下面这样来写样式表：

```
P {font-style: normal;}
EM, I {font-style: italic;}
```

从图5-25可以看出，这些样式使段落以平常的直立文本显示，而EM和I元素则使用斜体文本。

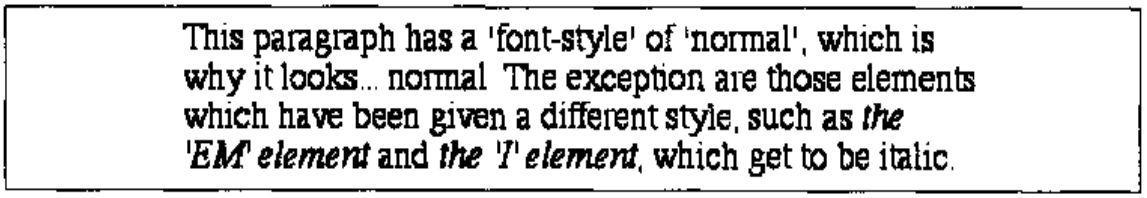


图5-25 带样式表的普通文档

另外，在EM和I元素之间或许应该有细微的区别。

```
P {font-style: normal;}
EM {font-style: oblique;}
I {font-style: italic;}
```

仔细观察图5-26，EM和I元素没有明显的区别。实际上，并不是每种字体都能将斜体字和倾斜字区分开来，而且当二种字型都存在时，很少有浏览器能区分它们之间的区别。

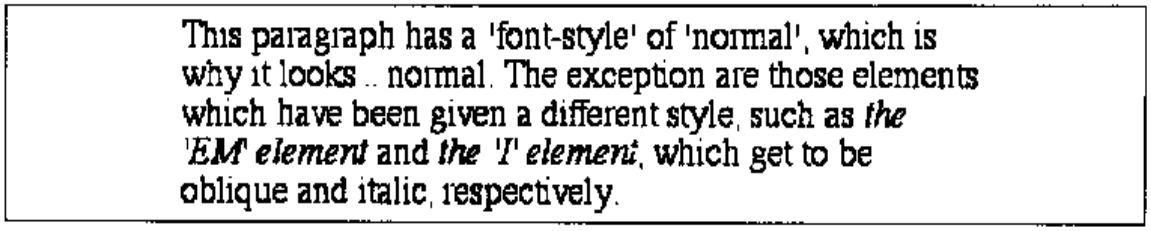


图 5-26 更多的字体样式

基于上述原因，如果没有Italic字型，只有Oblique字型，那么后者可代替前者使用。如果情形刚好相反——Italic字型存在，但没有Oblique字型——依据CSS规范，用户代理不能用前者来代替后者。最终，用户代理只好简单地将直立字体的某个倾斜的字体版本作为倾斜字体使用。事实上，在数字世界中经常发生这样的情况，因为只需简单计算就能很容易地倾斜某种字体。

而且，在某些操作系统中，已声明的italic字体可能依据其实际尺寸的大小变换为倾斜的字体。例如，Times字体在Macintosh上的显示如图5-27所示，其中唯一的区别就是字体尺寸。

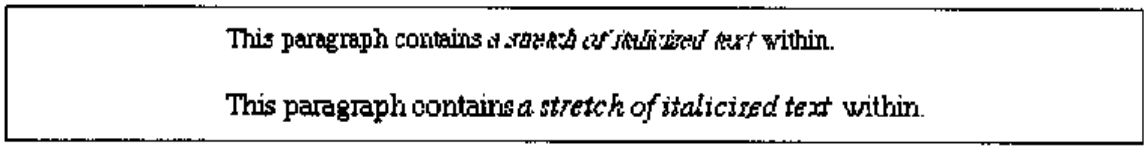


图 5-27 同样的字体、同样的模式、不同的尺寸

不幸的是，操作系统也没有更好的处理办法。通常情况下，在网络浏览器中斜体和倾斜字体看上去一模一样。

但是字体样式仍然有用。例如，引用块为斜体是一种很常用的印刷习惯，但引用中特殊的强调文本应该是直立文本。为得到这样的效果，如图5-28所示，可以用下面的样式：

```
BLOCKQUOTE {font-style: italic;}
BLOCKQUOTE EM, BLOCKQUOTE I {font-style: normal;}
```

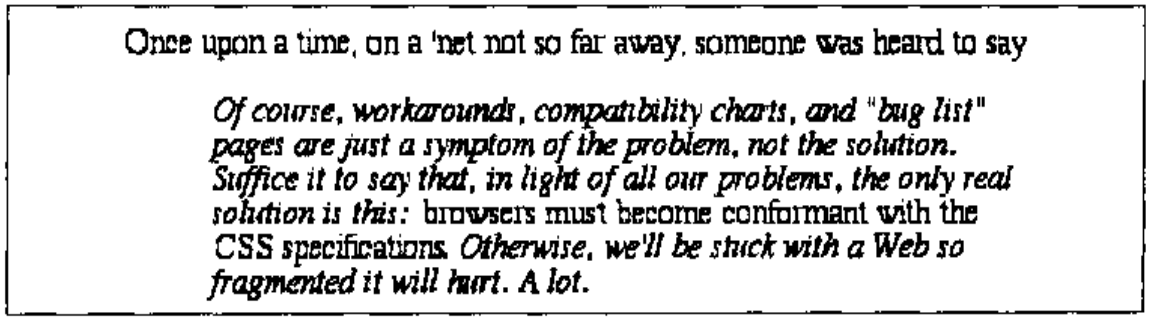


图 5-28 常用的印刷习惯

字体变形

除了尺寸和样式外，还有字体变形（variant）。CSS提供一种方式以声明常用的字体变形。

font-variant

允许值	small-caps | normal
初始值	normal
可否继承	是
适用于	所有元素

font-variant有两个值：缺省值normal，用于描述普通文本，和small-caps，它请求使用小型大写字母文本。如果读者不熟悉这种效果，它看起来应像这样：IT LOOKS SOMETHING LIKE THIS。不同于大小写字母，小型大写字母（small-caps）字体使用的是不同尺寸的大写字母。如图 5-29 所示：

```
H1 {font-variant: small-caps;}
P {font-variant: normal;}

<H1>The Uses of font-variant</H1>
<P>
The property <CODE>font-variant</CODE> is very interesting...
</P>
```

读者可能注意到了图中的H1元素，其中有稍大的大写字母，也有较小的大写字

母。它们之间唯一的区别就是大写字母的尺寸不一样，可能读者会联想到text-transform: uppercase。但用字体属性来声明small-caps的原因是某些字体有特殊的小型大写字母字体变形。因此，可以用一个字体属性来选择这种字体。

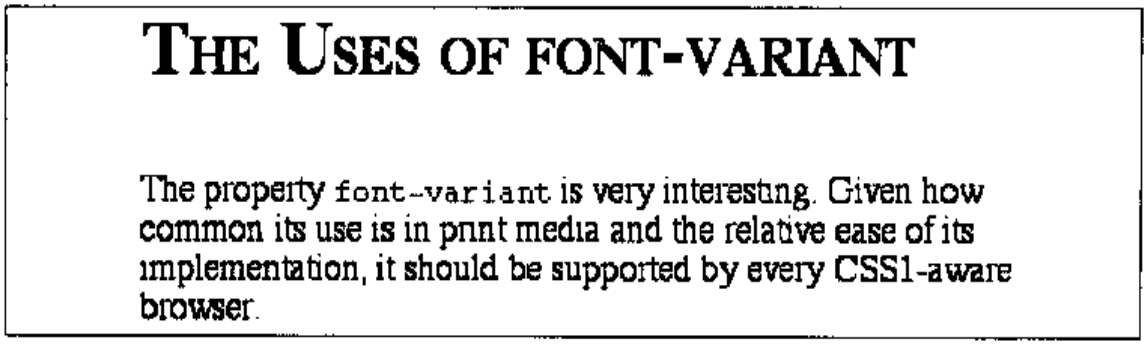


图5-29 小型大写字母

如果没有这样的字体会怎么样呢？在规范中提供了两个选项。第一，让用户代理通过缩放大写字母来创建小型大写字母的字体。另外，就是简单地让所有大写字母都是一个尺寸，就好像是用text-transform:uppercase作为声明一样，如图5-30所示。很明显，这并不是一种理想的解决方案，但这是可行的。

```
H1 {font-variant: small-caps;}
```

图5-30 可行的，但不是最优的小型大写字母

警告：在支持font-variant: small-caps的浏览器中（Explorer 4或5，以及Opera3.5），只有Opera和macintosh上的IE5会按制作者的意图显示文本，其他的Explorer版本都采用全大写字母的方式。

使用缩略方式：字体属性

当然，所有这些属性都很复杂，而综合运用它们就开始有点乏味了。

```
H1 {font-family: Verdana, Helvetica, Arial, sans-serif; font-size: 30px;
   font-weight: 900; font-style: italic; font-variant: small-caps;}
H2 {font-family: Verdana, Helvetica, Arial, sans-serif; font-size: 24px;
   font-weight: bold; font-style: italic; font-variant: normal;}
```

其中某些属性设置可以通过分组选择符来解决，然而，是不是将所有属性都组合到一个属性上更为简单呢？这就是字体属性（font），它是其他所有属性的一个缩略属性。

font

允许值	[<字体风格> || <字体变形> || <字体加粗>]? <字体尺寸> [/ <行高>]? <字体系列>
初始值	参见各个独立的属性
可否继承	是
适用于	所有元素

一般来说，font声明可以让每个字体属性都有一个值。因此前面的例子可以作简化，但其效果一样（如图 5-31 所示）：

```
H1 {font: italic 900 small-caps 30px Verdana, Helvetica, Arial, sans-serif}
H2 {font: bold normal italic 24px Verdana, Helvetica, Arial, sans-serif;}
```

图 5-31 典型的字体规则

因为其他的可能性很少，而font规则又可以用相对松散的方式来书写，这就使得前面的样式能够被简化。仔细观察前面的例子，会发现前三个值的顺序不一样。在H1规则中，前三个值的顺序为font-style、font-weight和font-variant，而第二条规则中的顺序为字font-weight、font-variant和font-style。这并没有错，因为它们可以按任意顺序书写。而且，它们中的normal值可以去掉：

```
H1 {font: italic 900 small-caps 30px Verdana, Helvetica, Arial, sans-serif;}
H2 {font: bold normal italic 24px Verdana, Helvetica, Arial, sans-serif;}
```

在这个例子中，H2规则中的normal值被去掉了，但效果同图5-31中的一样。

但这种自由的情况只能应用于font的前三个值，而后两个值的应用非常严格。不仅是字体尺寸和字体系列必须按固定的顺序出现在声明中，而且它们必须要在font的声明中出现。如果其中之一被去掉了，那么整条规则将无效，而且很可能会被用户代理完全忽略。因此下面的规则会得到如图5-32所示的结果。

```
H1 {font: normal normal italic 30px sans-serif;}   /* no problem here */
H2 {font: 1.5em sans-serif;}   /* also fine; omitted values set to 'normal' */
H3 {font: sans-serif;}  /* INVALID--no 'font-size' provided */
H4 {font: light 24pt;}  /* INVALID--no 'font-family' provided */
```

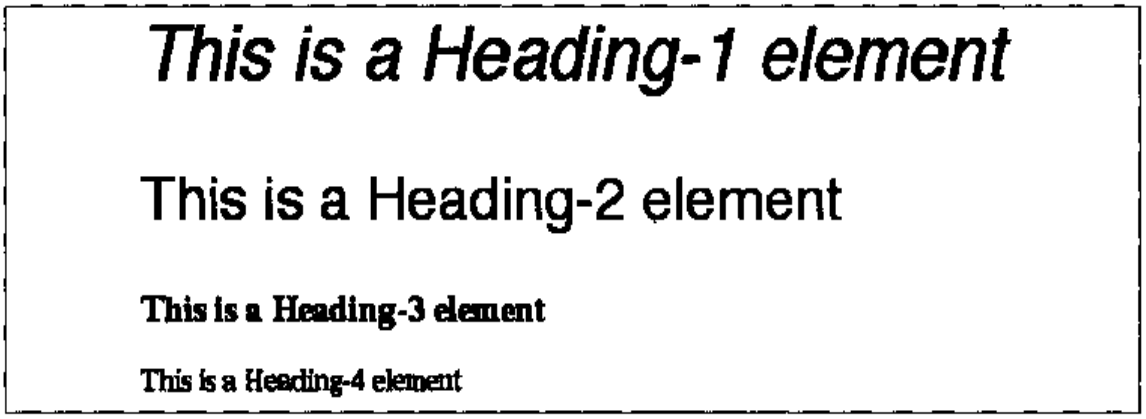


图5-32 字体尺寸和字体系列都是必需的

加上行高

迄今为止，font属性中只用了五个值，但也可以在其中设置行高（line-height），尽管行高是一个文本属性，而非字体属性。行高的加入方式为：将其附加在字体尺寸的值后，以正斜杠（/）分隔：

```
H2 {font:bold italic 200%/1.2 Verdana,Helvetica,Arial,Sans-serif;}
```

这条规则设置所有的H2元素为粗体和斜体，使用非衬线字体系列的某种字体，字体尺寸为24px，而且行高为30px。其结果如图5-33所示。

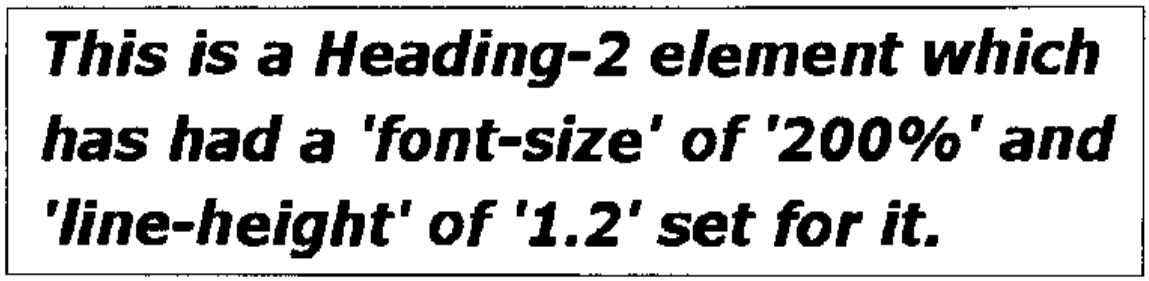


图5-33 加入行高

行高的加入是可选的，如同前三个值一样，只有在必要的时候才出现在字体属性的声明中。但要记住字体尺寸总是在行高前面的，而且还由一个正斜杠分隔开来。

字体匹配

正如我们所看到的，CSS允许匹配字体系列、粗细和变形。这都是通过字体匹配来完成它的，它是一个较复杂的过程。对于制作者来说，要想让用户代理为他们的文档选择更好的字体，理解字体匹配是很重要的。我之所以将它放到本章的末尾，是因为它对于字体属性的工作方式来说并不是必须要理解的，而且读者也可以跳过这一部分直接到第六章，“颜色和背景”。如果读者对此感兴趣，下面我们将讲述字体是如何匹配的。

第一步，用户代理创建或访问一个字体属性数据库。这个数据库中包含各种用户代理访问的CSS1的字体属性。典型地，它包含本机安装的所有字体，尽管还可能有其他字体（例如，用户代理可以有自己内置的字体）。如果用户代理遇到两个相同的字体，则简单地忽略其中之一。

第二步，用户代理取出一个元素以决定应用哪些字体属性，而且为元素的显示创建一个字体属性列表。基于此列表，用户代理选择一个初始的字体系列应用于元素的显示。如果有这样的系列可用，则使用它，否则，还需做其他的工作。

第三步，如果第二步中无匹配的字体，用户代理将查找同一系列中的可选字体。如果找到，则重复第二步动作。

第四步，假设一般的匹配条件已满足，但它不包含显示所需的每一样东西——例

如，缺少了版权证号的字体——那么用户代理又回到步骤三，它需要搜索另一种可选的字体进而再返回到第二步。

最后，如果所有可选字体都已试过，但还是未能匹配，用户代理会为给定的字体系列选取缺省字体，以尽量接近的方式显示此元素。

可以看出，在这个过程中，第二步访问量很大。可以将其分解为更细的步骤：

1. 首先匹配字体的font-style属性。关键字italic匹配任何标记为“italic”或“oblique”的字体。如果都不可用，则匹配失败。
2. 下一步是font-variant属性。任何非“small-caps”的字体都为normal。任何标记为“small-caps”的字体，或允许小型大写字母的字体，或需要将小写字母替换为大写字母的字体都匹配为small-caps。
3. font-weight属性的匹配。它必须在某个限度内进行匹配。由于这一属性在CSS1下的处理方式使得它的匹配不可能失败。
4. 最后是font-size的匹配。必须在某个限度内进行匹配，而这个限度留给用户代理来定义。因此，一个用户代理可能允许在20%的误差范围内匹配，而另一个用户代理可能只允许10%的尺寸误差。

整个过程又长又乏味，但它能帮助理解用户代理选择字体的过程。比如，读者或许想在文档中指定用Times字体或任何其他的衬线字体，如图5-34所示：

```
BOBY {font-family: Times, serf;}
```

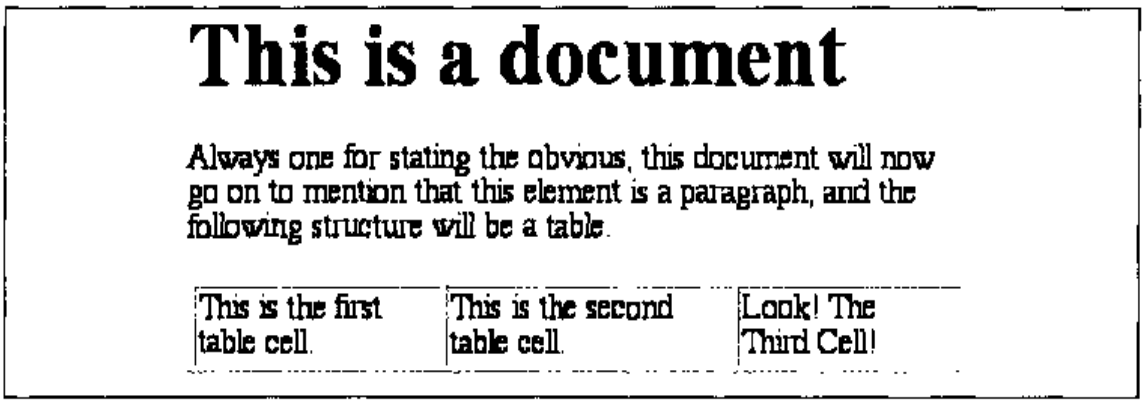


图5-34 字体匹配的最终结果

对于每一种元素，用户代理都要检查其中的字符以决定是否匹配Times字体。多数情况下，这都没有什么问题。但假设段落中有一个中文字符，Times不可能匹配这个字符，所以用户代理要么绕过这个字符，要么寻找另外一种能满足显示要求的字体。当然，西方字体几乎不可能包含中文字符，但总应该有一种这样的字体存在（让我们暂且称之为AsiaTimes）。然后用户代理就能用它来显示这个元素——或这个字符。因此，整个段落都可能用AsiaTimes来显示，或者除这个中文字符用AsiaTimes显示之外，其余字符用Times字体进行显示。

小结

尽管制作者不能指望在文档中使用某个具体的字体，但他们能够很容易指定使用一般字体系列。这种特殊的行为受到广泛的支持，因为用户代理不让制作者（或读者）指定字体，但它自己能很快地找到一种合适的字体。

至于字体操作的其他领域，其支持程度在不停地发生变化。改变字体尺寸通常工作得很好，但其实现却从最简单的到接近于正确的都有。令制作者感到沮丧的通常不是字体尺寸如何被支持，而是为何在不同的媒体，甚至是不同的操作系统和用户代理下使用同一单位时会产生不同的结果。使用点作为单位带来的危险很多，而且用长度单位来作网页设计通常不可取。百分比、em和ex通常适用于改变字体尺寸，因为这些单位在所有普通的显示环境下都很好用。

到此为止，我们已经讲完如何改变文本和字体属性，下面将转向如何改变包含文本的元素的风格。

第六章

颜色和背景

毫无疑问，多数读者可能都还记得第一次改变网页颜色时的情形。以前多是灰色背景、黑色字体再加上蓝色链接。突然，读者可以随心所欲地使用自己想要的颜色——或许是黑色的背景、浅蓝色的文本，再加上灰绿色的链接。进而，还可往背景里加入一些图像，而且能将背景图像与彩色文本适当组合，创作出非常精美的网页效果。

但一个网页上的所有文本都用同一种颜色已远远不能满足用户的要求了。于是就产生了<FONT COLOR="...">。没有这种属性设置，就不可能得到多彩的网页，比如使某些文本为黑色，某些为红色。

现在，由于CSS的强大功能，在不使用任何FONT标签的情况下，也可以为单一网页增加更多的颜色选择。这就赋予了制作者更强的网页创作能力，在白色主背景上显示黑色文本的同时，还可以在淡紫色背景上显示黑色的导航链接。而且创作空间几乎是无限的：定义红色的警告文本，用暗紫色使粗体文本更显眼，以不同的绿色调设置标题，等等。

当然，这就意味着在设计网页时，得事先考虑周全。尤其是在颜色方面，更应如此。例如，要将所有超链接设置为黄色，但这样会和文档其他部分的背景色发生冲突吗？使用太多的颜色是否会使用户感觉太凌乱了？如果改变了缺省链接颜色，

用户还能辨认出来吗？如将普通文本和超链接文本设置为同一种颜色，那么就很难认出链接——事实上，如果链接无下划线，几乎不可能认出来。

尽管有这么多的问题，改变元素颜色几乎是每个制作者都要使用的，或许会经常使用。只要使用得当，这些颜色确实能够增强文档的表现力。一旦计划好如何使用颜色后，就必须决定怎样去应用它，因为某些方法需要使用类以及一些简单的元素选择符。

让我们来看一个例子，假设网页中的所有H1元素为绿色，大多数H2元素为蓝色，而所有超链接为暗红色。然而，某些H2元素应该为暗蓝色，因为它们与不同类型的信息相关联。处理上述要求的最简单的方法就是在需要暗蓝色的H2元素上放置一个dkblue类，声明如下：

```
H1 {color: green;}
H2 {color: blue;}
H2.dkblue {color: navy;}
A:link {color: maroon;}   /* a good dark red color */
```

任何暗蓝色的H2元素都应标记为<H2 CLASS="dkblue">...</H2>。

注意：用对信息类型的描述作为类名要比直接选用试图取得的视觉效果作为类名要好一些。比如，我们想让所有作为子标题的H2元素为暗蓝色，用subsec或sub-section作为类名就要清楚得多。这样的名字既代表了某种含义——而且还和其他概念相独立。毕竟，或许后来决定将原来的暗蓝色换为暗红色，而此时语句H2.dkblue {color: maroon;}就显得有点傻了。

从这样一个简单的例子中，我们得到这样一个教训：当计划使用样式时，通常应该提前做好构思，并使用所有能用得上的工具。继续上面的例子，假设某个导航栏被加到网页中。在导航栏里，超链接为黄色，而非暗红。如果导航栏被标记为一个navbar的ID，那么只需加入这样一条规则：

```
#navbar A:link {color: yellow;}
```

这就会改变导航栏中的超链接颜色，而不会影响到文档中其他的超链接。

颜色

在CSS中，只有一种类型的颜色，即简单的纯色。如果将页面的背景设置成红色，那么整个背景都将是同一种红色。当然，这与迄今为止在HTML中运用的颜色没什么区别。当声明<BODY LINK="blue" VLINK="blue">时，不管超链接位于文档的什么地方，它们都将是同一种色调的蓝色。

在使用CSS时，同样不要改变这种思维方式。如果用CSS来设置所有超链接的颜色（访问或未访问过的）为蓝色，效果同上面的一样。同样，如果使用样式来设置页面背景为绿色，那么整个背景都将是同一色调的绿色。如果设置H1元素背景色为navy，那么每个H1元素的背景都为暗蓝色。

在CSS中，可以设置任何元素的前景色和背景色，从BODY元素到下划线和斜体标记，几乎所有的一切——列表项、整个列表、标题、超链接、表格单元、表单元素，甚至是图像（在有限的范围内）。为理解这种方式，首先应了解什么是前景。

什么是元素的前景？一般来说，前景是指元素中的文本，但也不尽然：元素的边框也被认为是前景的一部分。这样就有两种方式直接影响元素的前景色：使用颜色属性和使用边框属性中的一种来设置边框颜色。最基本的是边框颜色(border-color)属性，以及更具体的一些属性，如border-top，border-right，border-bottom，border-left和border。

元素的背景指的是所有前景后面的空间，直到边框的边缘，这样，内容框和填充都是元素背景的一部分。有两种方法设置背景色：background-color和background属性。

前景色

设置前景色的最简便方法是用color属性。

color	
允许值	<颜色>
初始值	与用户代理有关
可否继承	是
适用于	所有元素

本属性接受第三章“单位和值”中所讨论的任何一种有效颜色值，比如 #FFCC00 或 rgb(100%,80%,0%)。它同设置文本颜色具有同样的效果，如图 6-1 所示：

```
<P STYLE="color: gray;">This paragraph has a gray foreground.</P>
<P>This paragraph has the default foreground.</P>
```

This paragraph has a gray foreground

This paragraph has the default foreground

图 6-1 声明颜色与缺省颜色的对比

注意：在图 6-1 中，缺省的前景色为黑色。但也不是必须为黑色，因为用户可以为他们的浏览器（或其他用户代理）设置不同的前景（文本）色。如果缺省文本被设置成绿色，前面例子中的第二段就会是绿色，而不是黑色——但第一段仍然是灰色。

当然不必将自己约束于这一简单的操作。还有许多种方法可以使用颜色。比如在某些段落中可能包含对用户的一些警告文本。为了使这些文本更突出，可以用红色来显示。这只需要在包含警告文本的段落中放置一个 warn 类（<P CLASS="warn">）和如下所示的一条规则即可：

```
P.warn {color: red;}
```

如果想让警告段落中的链接以绿色显示。如下：

```
P.warn {color: red;}
P.warn A:link {color: green;}
```

然后，读者可能会改变主意，决定让警告文本为灰色，而警告段落中的链接为银色。那么只需将上述的规则变为下列规则：

```
P.warn {color: gray;}
P.warn A:link {color: silver;}
```

颜色的另一种用法是用于增加文本的吸引力。例如，粗体文本虽然已很明显，但可以给它赋予不同的颜色，使其更加显眼：

```
B {color: maroon;}
```

如果想让所有包含highlight类的表格单元中的文本为黄色，可如下设置：

```
TD.highlight {color: yellow;}
```

这将设置所有带highlight类的表格单元的前景色为黄色，如图6-2所示。

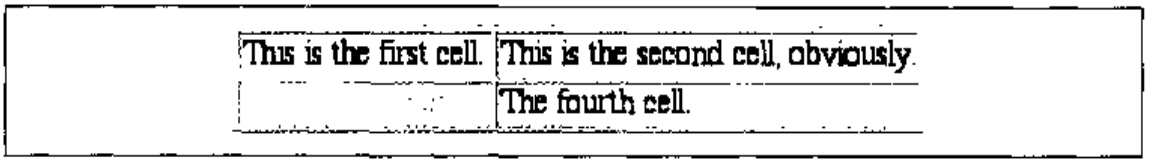


图6-2 高亮显示表格单元中的内容

BODY属性

color属性有很多种用途，最基本的就是用来代替BODY属性TEXT，LINK和VLINK。若和链接伪类连用，color可以完全代替BODY属性。下面例子的第一行可以用其后的CSS重写，而且它们的结果一样，如图6-3所示：

```
<BODY TEXT="black" LINK="#808080" ALINK="silver" VLINK="#333333">

BODY {color: black;}      /* replacement css */
A:link {color: #808080;}
A:active {color: silver;}
A:visited {color: #333333;}
```

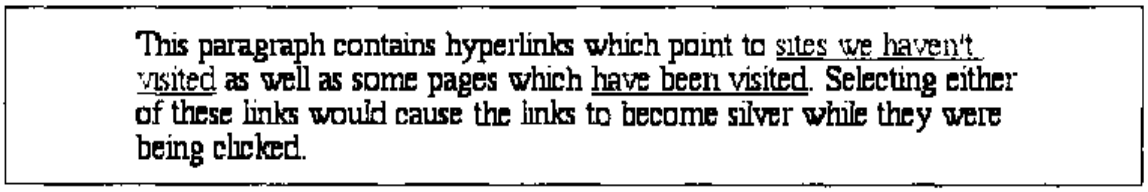


图 6-3 用 CSS 替换 BODY 属性

然而同老方法相比，这似乎增加了一些额外的输入量，且只能在文档级修改。例如，要使某些链接为中灰色，而其他的稍微暗一些，用 BODY 属性就不能这样做。因此，只好在每一个稍暗的链接上使用<FONT COLOR="#666666">。而 CSS 就不同，只需在所有需要改变颜色的链接元素中加入一个类，再对应地修改样式即可。结果如图 6-4 所示：

```
BODY {color: black;}
A:link {color: #808080;}        /* medium gray */
A.external:link  {color: silver;}
A:active {color: silver;}
A:visited {color: #333333;}     /* a very dark gray */
```

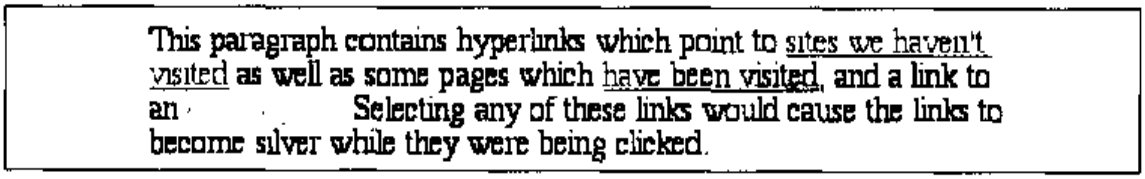


图 6-4 改变超链接的颜色

这就将所有包含 external 类（<A CLASS="external" HREF="...">）的链接设置为银色，而不是中灰色。当然，一旦它们被点击后，仍为暗灰色，除非再加一条特殊的规则：

```
BODY {color: black;}
A:link {color: #808080;}        /* medium gray */
A.external:link  {color: #666666;}
A:active {color: silver;}
A:visited {color: #333333;}     /* a very dark gray */
A.external:visited {color: black;}
```

这将使所有未访问过的外部链接为中灰色，访问后为黑色，而其他链接在访问后为暗灰色，未访问时为中灰色——如图 6-5 所示。

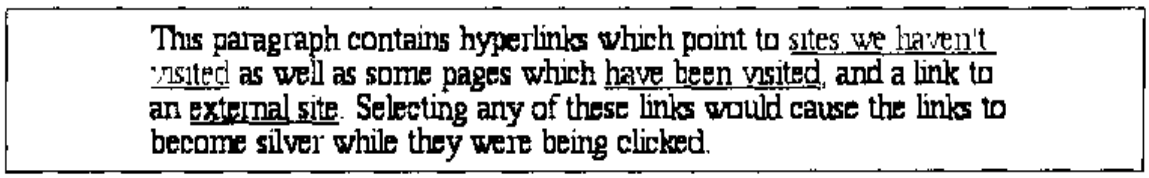


图 6-5　为不同的链接类设置不同的颜色

这对于老式的 BODY 属性来说是不可能的。如果要用 BODY 属性，就必须为每个文档中符合条件的元素定义其属性值。但是，当要改变这些值时——对，必须去编辑大量的文件，这合算吗？另一方面，如果将其建立为一个外部样式表，然后将所有页链接到这个样式表，那么就只需要编辑一个文件即可。

动人的边框

颜色值也能用于边框。假设已定义了下面的样式，其结果如图 6-6 所示：

```
P.aside {color: gray; border-style: solid;}
```

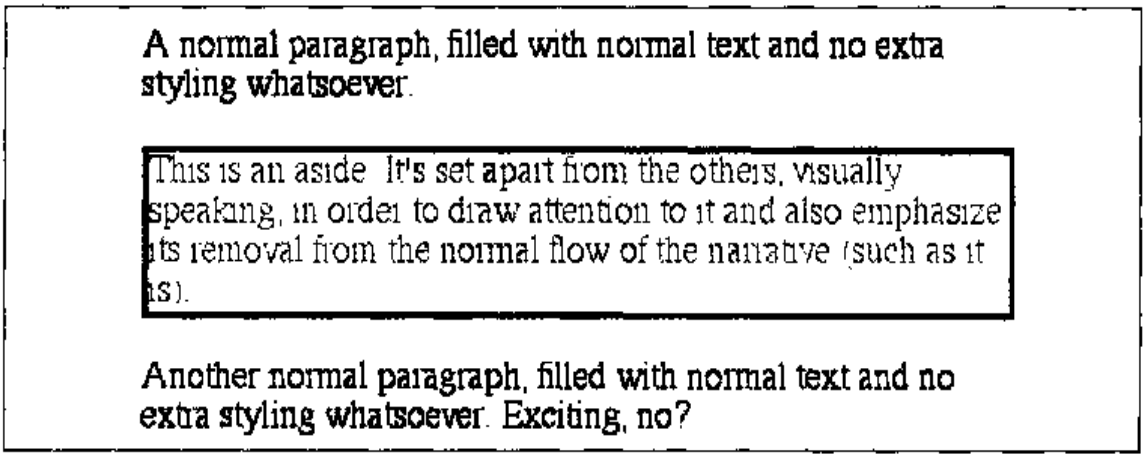


图 6-6　边框颜色取自文本内容的颜色

这使得 <P CLASS="aside"> 元素具有灰色文本和中等宽度的实边框。前景色被缺省应用于边框。覆盖它的方法是用 border-color 属性：

```
P.aside {color: gray; border-style: solid; border-color: black;}
```

这将使得文本为 gray，边框为 black，如图 6-7 所示。border-color 的任何值都会覆盖 color 的值。

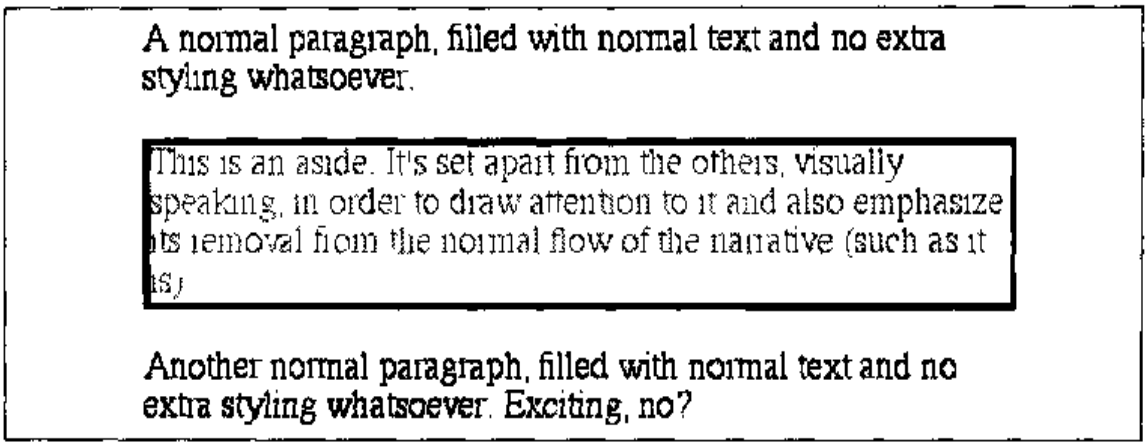


图 6-7 覆盖缺省的边框颜色

附带地，边框还可以影响图像的前景色。因为图像本身是由各种颜色组成的，因此就不能使用 color 来修改它们，但可以改变图像周围边框的颜色。这可以用 color 或 border-color 来实现。下面的规则对于类 type1 和 type2 的图像具有相同的效果，如图 6-8 所示：

```
IMG.type1 {color: gray; border-style: solid;}
IMG.type2 {border-color: gray; border-style: solid;}
```

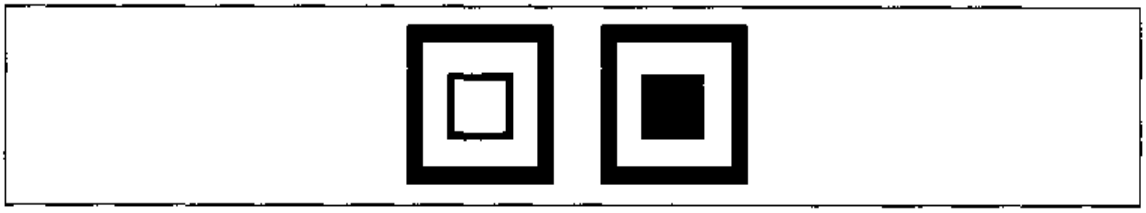


图 6-8 图像的边框颜色

边框颜色和边框在第七章“框与边框”中，讲述得更为详细。

继承颜色值

到现在为止，读者可能已经猜到 color 是可被继承的。这又给我们带来了便利，如果读者声明 P {color: maroon;}，那么段落中的任何文本都应是褐色，即使是强调或者粗体或其他的任何文本都一样。当然，如果想让某一元素的颜色不一样，这也很简单，如图 6-9 所示：

```
P {color: maroon;}
EM {color: #999999;}
```

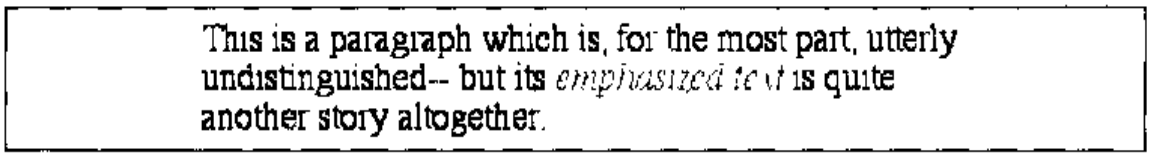


图 6-9　不同元素具有不同颜色

由于颜色的可继承性，理论上可以设置文档中的所有文本为某一颜色，比如声明 BODY {color: red;}使其为红色。这将使得除已经样式化的元素之外的所有文本为红色。但这不会影响水平线规则(HR)或图像元素。然而，早些版本的Explorer允许HR元素继承颜色值，而表单元素不能继承颜色值。这种继承问题仍然存在，实际上，直到现在还能发现某些浏览器预先定义表格颜色，以阻止BODY的颜色值被继承到表格单元里：

```
BODY {color: red;}
TABLE {color: black;}
```

如图 6-10 所示是制作者样式与浏览器内置样式组合的结果。

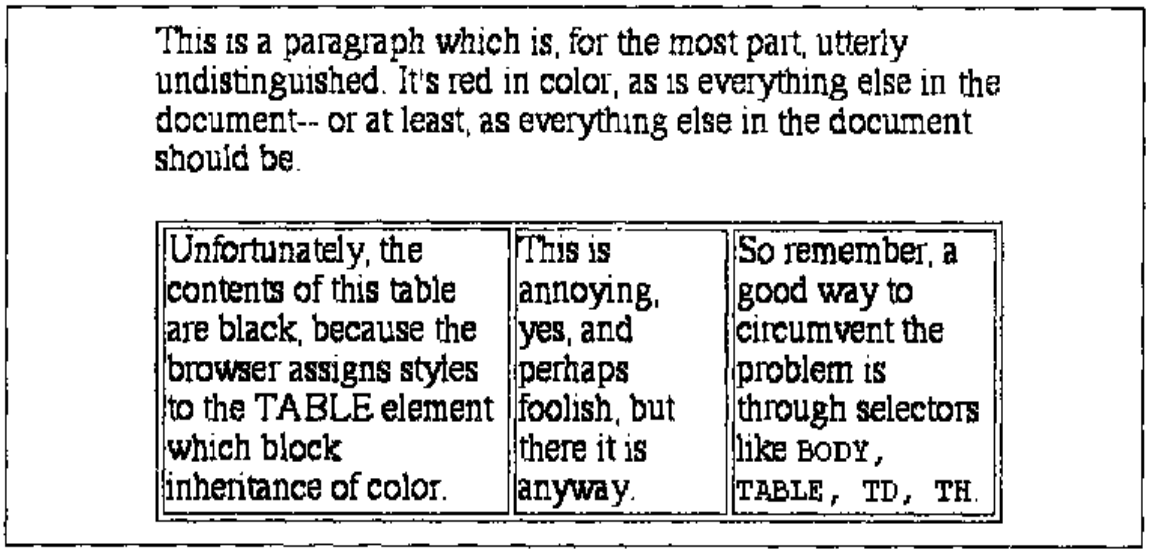


图 6-10　制作者样式与浏览器样式组合的结果

因为浏览器为TABLE元素预先定义了颜色值，它便将继承的颜色值覆盖了。这显得既麻烦又多余，但这个障碍是必须克服的。可以通过列举各种表格元素的选择符来克服它。例如，为使所有表格内容同文档主体一样呈红色，可以用下面这条规则：

```
BODY, TABLE, TD, TH {color: red;}
```

这通常能解决一些问题，但由于某些未知的原因，它也不是总能解决这个问题。Navigator 4 对于上述规则就会出现一些随机错误。所以CSS不得不提出一种方案以预测Navigator的行为。

警告： 还需注意的是Navigator 4遇到一个无效值时（如invalidValue），它可以按照它自己的机制来处理，并且使用某种颜色代替无效值。尽管这一行为不是随意的，也不是精确的，但效果实际上是一样的。例如，invalidValue结果变成了暗蓝色，而在CSS1中无效却在CSS2中有效的inherit值却变成了糟糕的黄绿色调。这些都是不正确的行为，所以在写样式时要注意这一点。

表单元素

颜色属性应该能够（至少理论上可以）应用于表单元素。如图6-11所示，声明SELECT元素使用暗灰色文本可以使用下面的规则：

```
SELECT {color: rgb(33%,33%,33%);}
```

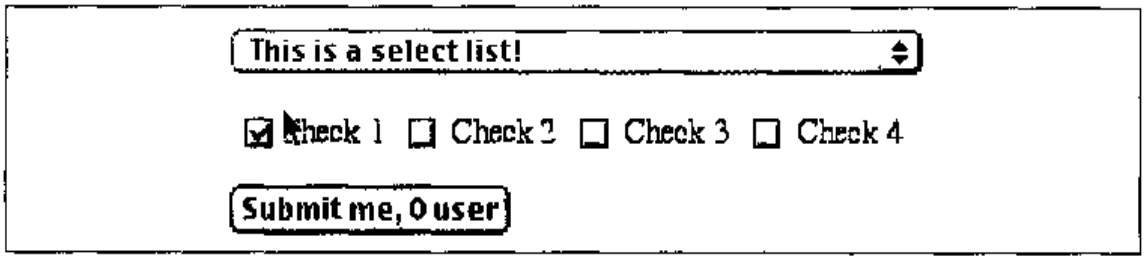


图6-11 设置表单元素的颜色

也可以设置INPUT元素的前景色，但这会影响所有输入的元素，从文本到单选按钮到复选框都会变成同一种颜色，如图6-12所示：

```
SELECT {color: rgb(33%,33%,33%);}
INPUT {color: gray;}
```

图中，复选框旁边的文本仍为黑色。这是由于我们只将这些样式应用于INPUT和SELECT元素，而没有应用于普通的段落（或其他）文本。

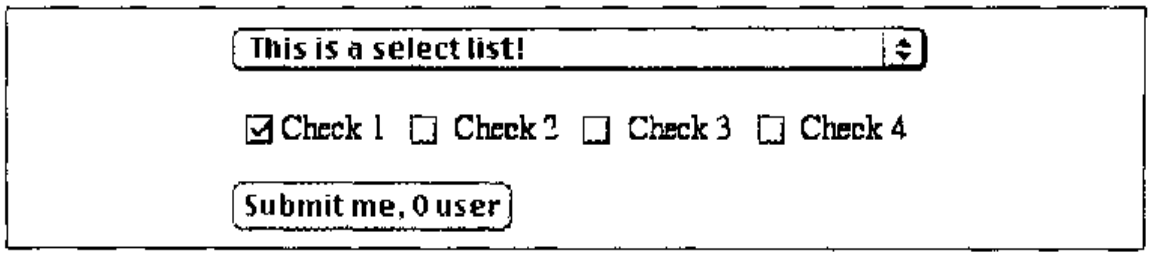


图 6-12 改变表单元素的前景色

CSS1 也有它的局限性，它不能分辨不同类型的 INPUT 元素。如果想要使复选框和单选按钮为不同的颜色，那么只好为它们分配不同的类以得到想要的结果（见图 6-13）：

```
INPUT.radio {color: #666666;}
INPUT.check {color: #CCCCCC;}

<INPUT TYPE="radio" NAME="r2" VALUE="A" CLASS="radio">
<INPUT TYPE="checkbox" NAME="c3" VALUE="one" CLASS="check">
```

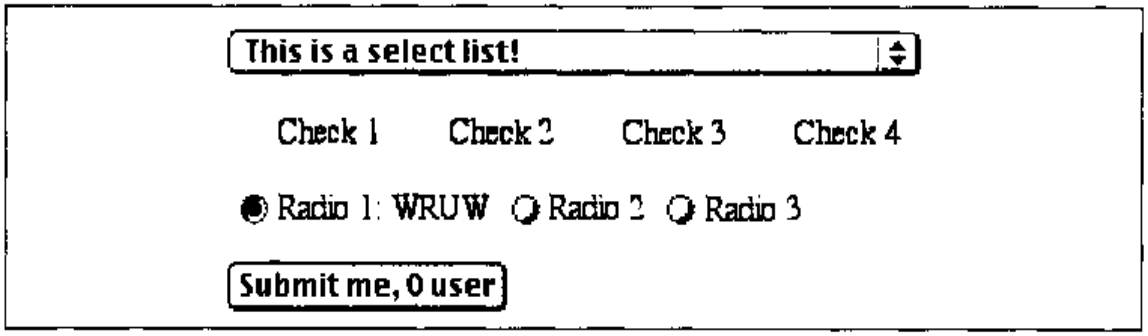


图 6-13 使用类来将不同的样式应用于不同的 INPUT 元素

在 CSS2 中，根据元素的属性来分辨不同的元素就容易得多。下面的规则将分别与其后的两个 INPUT 标签匹配：

```
INPUT[type="radio"] {color: #333333;}
INPUT[type="checkbox"] {color: #666666;}

<INPUT TYPE="radio" NAME="r2" VALUE="A ">
<INPUT TYPE="checkbox" NAME="c3" VALUE="one ">
```

这种匹配允许和类一起实施，在本例中就是这样。关于这种选择符更详细的信息，请参看第十章“CSS2 展望”。

警告：Navigator 4 不支持表单元素的颜色设置，但 Internet Explorer 4 和 5，以及 Opera 3.5 及后续版本都支持对表单元素设置颜色值。

背景颜色

类似于前景色，同样可以为元素的背景声明颜色。这可以使用background-color属性，它接受任何有效的颜色值。

background-color	
允许值	<颜色> | transparent
初始值	transparent
可否继承	否
适用于	所有元素

背景指的是内容框及补白（padding），总是位于前景的后面。当然，声明的背景色会同时应用于内容框和补白，如图 6-14 所示：

```
P {background-color: gray;}
```

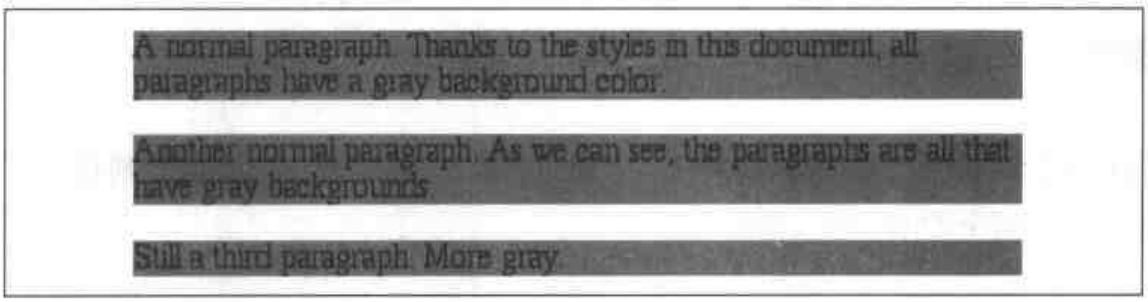


图 6-14　段落的背景为灰色

如果想让颜色从元素中的文本向外扩充一点，那么只需要添加一些补白，结果如图 6-15 所示：

```
P {background-color: gray; padding: 10px;}
```

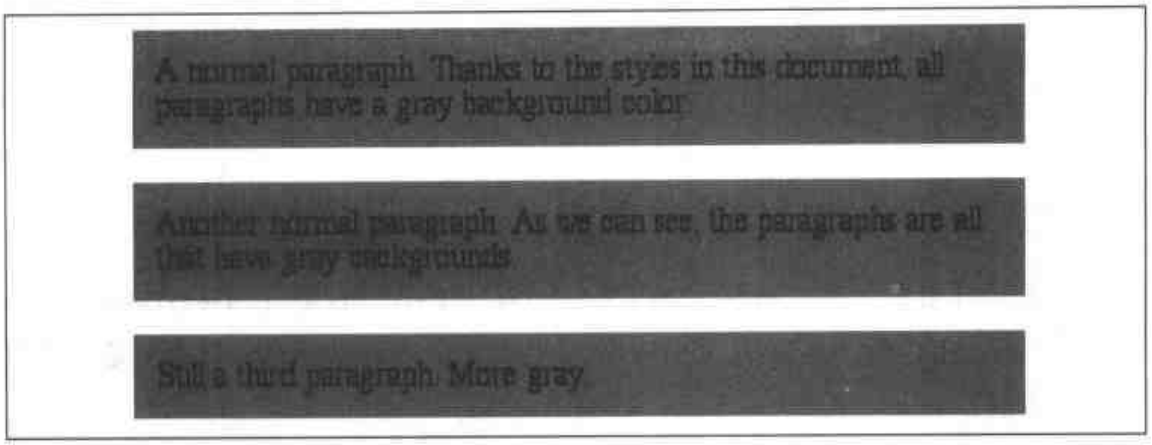


图 6-15 背景与补白

(补白将在第七章中详细讲述。)

任何元素都可以设置背景色，从BODY元素到EM和A这样的内联元素都可以。甚至表单元也可以，虽然某些用户代理不能正确处理这种情况。同样，background-color不能被继承。它的缺省值为transparent(透明)，这是因为，如果某个元素没有定义颜色，那么它的背景色将为透明，这样就能保证它的祖先元素是可见的。试想一下，如果缺省值为其他颜色，如银色，则总会出现与图6-16中类似的情况。如果浏览器也这么去做，就会出现问题，但庆幸的是，它们并没有这样做。

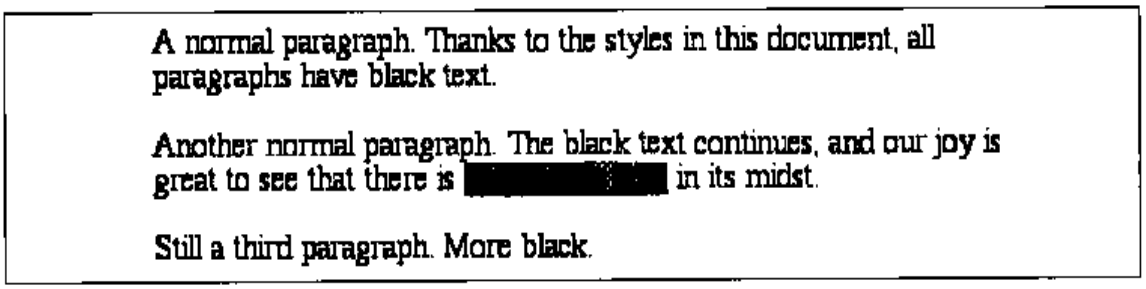


图 6-16 非透明背景

多数时候是不得不使用关键字transparent，但偶尔它也显得很有用。尽管它是缺省值，用户也可以将浏览器中的所有链接背景设置为白色。当读者为网页中的链接设置前景色为白色时，或许并不希望它有背景色。那么就需要声明下面的规则：

```
A:link {color: white; background-color: transparent;}
```

如果不管背景色，那么读者的白色前景将和用户的白色背景组合在一起，从而使得链接完全不可见。

实际问题

设置元素的背景色基本上就这么多了，但还有一个小小的警告：Navigator 4完全弄错了背景色的位置。它不是将背景色应用于整个内容框和补白，而只是将其显示于文本的后面，如图 6-17 所示。

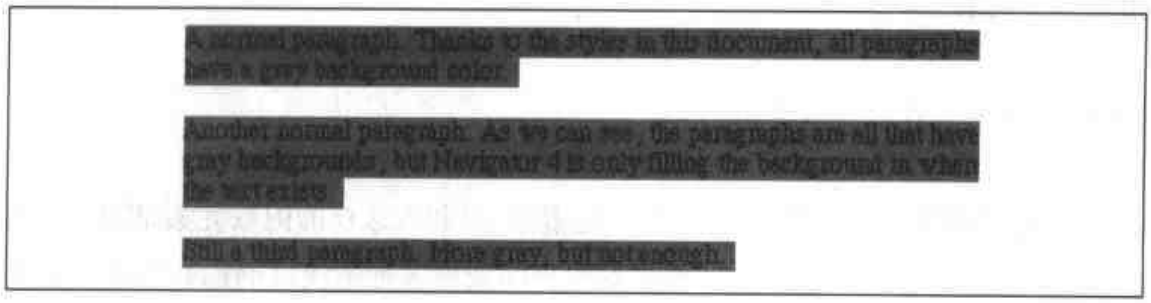


图 6-17 Navigator的不正确行为

再重复一次：这是完全错误的。然而，有一种办法可以解决这一问题，这就是设置元素的边框，也就是将边框的颜色设置成与背景一样：

```
BODY {background: silver;}
P {background-color: gray; padding: 1px; border: 0.1px solid gray;}
```

在这里，需要设置边框样式（border-style）。但不管是使用这个样式属性，还是简单地使用border属性的一个值，都无关紧要。

当然，这种设置关系到元素的边框，所以在其他的用户代理中也将使用这一设置。由于Navigator并不能很好地处理补白，因此在上面的例子中，会在内容框和边框之间出现一点儿黑色的间隔。总而言之，它显得并不完美。

特殊效果

有了color和background-color，就可以创作出很好的效果。如图 6-18 所示：

```
P {color: black;}
H1 {color: white; background-color: rgb(20%,20%,20%);}
```

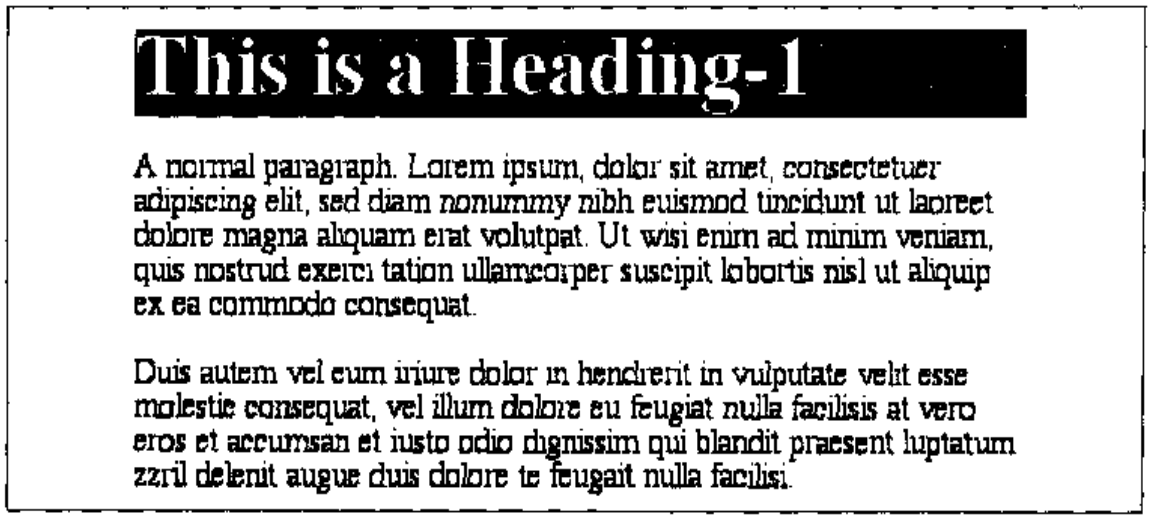


图 6-18 H1 元素的漂亮效果

这只是其中的一个例子，当然，可以像颜色那样有很多种组合，但我们不可能在此一一列举——只是介绍一些方法而已。下面就是一些简单的方法。

下面是一个简单的样式表，其结果如图 6-19 所示：

```
BODY {color: rgb(0%,50%,0%); background-color: #CCFFCC;}
H1, H2 {color: yellow; background-color: rgb(0,51,0);}
```

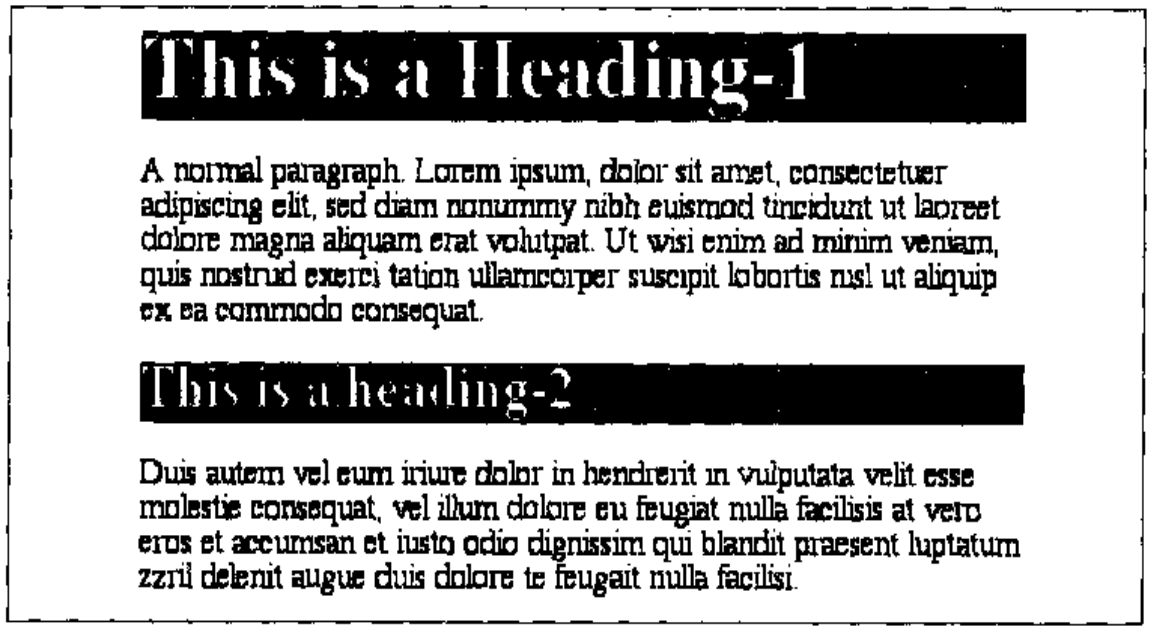


图 6-19 一个简单样式表的效果

下面的样式表稍微复杂一点儿（如图 6-20 所示）：

```
BODY {color: black; background-color: white;}
P {color: #333;}
PRE, CODE, TT {color: rgb(50%,50%,50%); background-color: #FFFFCC;}
A:link {color: blue; background-color: yellow;}
A:visited {color: navy; background-color: white;}
```

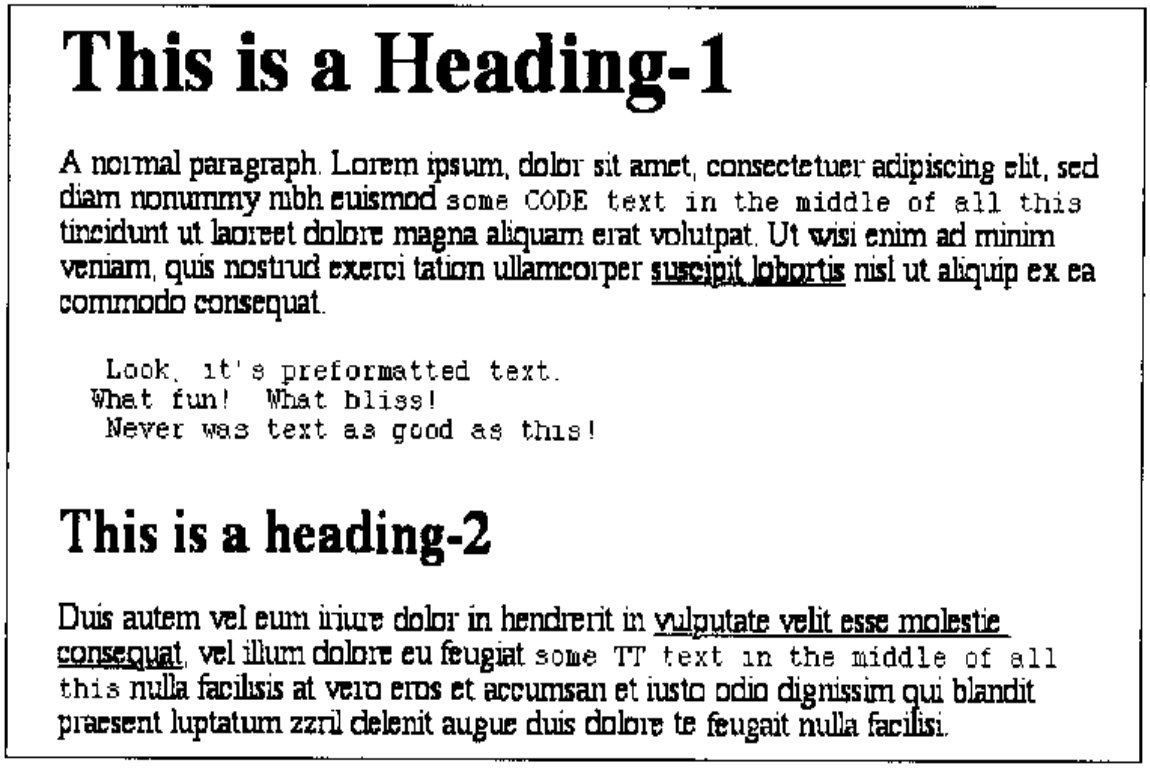


图 6-20 一个稍复杂的样式表的效果

这只是一个开端而已。读者可以设计自己的样式!

实践

读者可能已经注意到了，几乎在每种情况下，只要设置了前景色，同时也会设置其背景色。通常，这是一种好方法，因为我们不知道用户预先定义了什么样式，也不知道我们的样式将怎样和它们互相作用。可能还记得前面的那个例子吧，白色背景和白色前景，这就是我们想要避免的事情。

让我们再仔细研究一下。如下:

```
/* reader styles */
BODY {color: white; background-color: black;}
```

```
/* author styles */
BODY {color: black;}
```

在这种情况下，制作者样式会优先于读者样式 —— 总归在CSS1中是这样的 —— 那么本文档的新样式表将如下（结果如图6-21所示）：

```
/* combined styles */
BODY {color: black; background-color: black;}
```

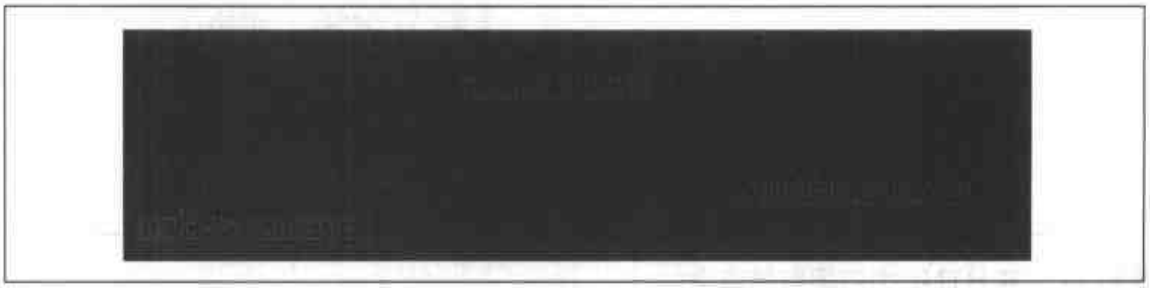


图6-21 黑色背景黑色前景

几乎不能阅读，是吧？只有超链接是可见的，这都要归于缺省样式。这就是为什么在同一个声明中组合前景和背景色是一种好方法的原因。所以，如果说声明一种颜色重要，无疑同时声明这两种颜色也很重要。

复杂背景

讨论了基本的前景和背景色后，我们再来看看背景图像。在HTML中，可以通过BODY属性BACKGROUND来将一幅图像同文档背景联系起来：

```
<BODY BACKGROUND="bg23.gif">
```

这将使用户代理载入文件bg23.gif，然后将其在水平和垂直方向重复地“平铺”于文档背景上，直到占满整个背景为止，如图6-22所示。

在CSS中也可以获得这样的效果，而且CSS还包含更多的平铺方式。让我们先从基础开始。

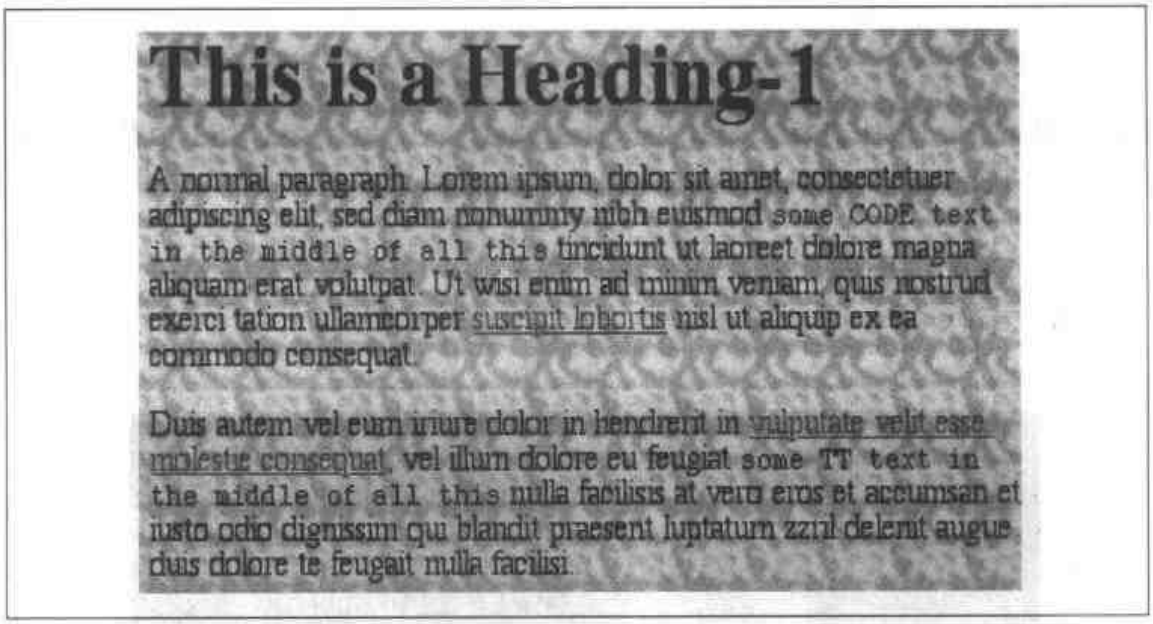


图 6-22 在 HTML 中应用背景图像

背景图像

要将图像贴于背景，使用 background-image 属性：

background-image	
允许值	<url> | none
初始值	none
可否继承	否
适用于	所有元素

缺省值 none 表示：没有图像会放置于背景。想要获得背景图像，只需给出属性的一个 URL 值：

```
BODY {background-image: url(bg23.gif);}
```

由于其他背景属性都取缺省值，这将使得图像 *bg23.gif* 平铺于文档的背景，如图 6-22 所示。

通常，同时指定背景色和背景图像是一种比较好的方式。但在本节中，我们并不打算这样做，我们只将注意力集中在background-image上。在本章的后面，我们再回过头来讨论这一问题。

背景图像可以应用于任何元素，不论是块级元素，还是内联元素，BODY是最常应用的元素，但我们的讨论不只局限于BODY元素。例如：

```
P.starry{background-image: url(http://www.site.web/pix/stars.gif);
   color: white;}

<P CLASS="starry">It's the end of autumn, which means the stars will be
brighter than  ever!  Join us...
```

在图6-23中，背景只应用于文档中的某个段落。

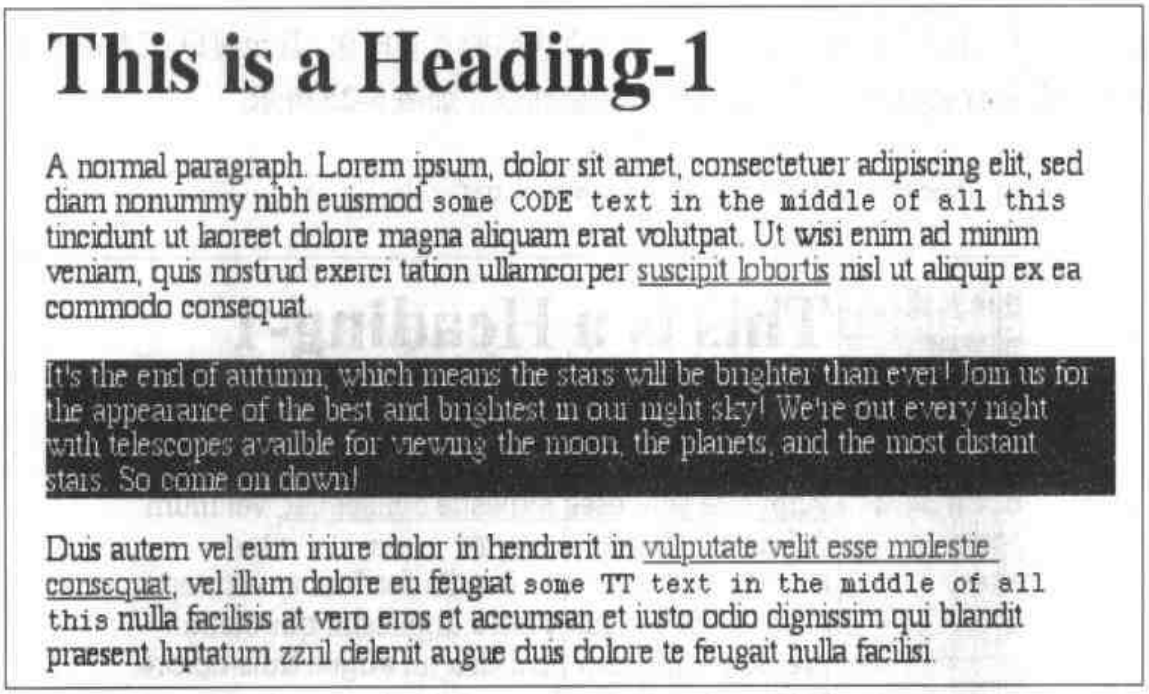


图6-23　为单个元素应用背景图像

它还允许将背景图像应用于内联元素，如超链接元素，如图6-24所示。当然，如果想要见到平铺的式样，需要很小的图像。毕竟，单个字符是不会很大的！

```
A.grid {background-image: url(smallgrid.gif);}

<P>This paragraph contains <A HREF="..." CLASS="grid">an anchor with a
background image</A> which is tiled only within the anchor.</P>
```

有很多方式可以应用背景图像。为了使其更加突出，可以放置一个图像在STRONG元素的背景里。

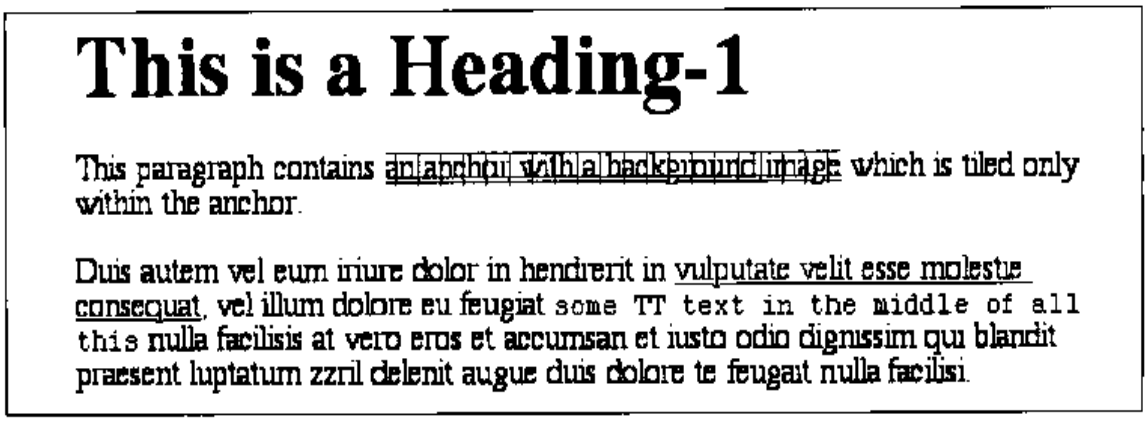


图 6-24 内联元素上的背景图像

也可以在标题背景里填上波浪式样或小点式样的背景图像，甚至可以在表格单元里填充各种样式以使其区别于页中的其他部分，如图 6-25 所示：

```
TD.nav {background-image: url(darkgrid.gif);}
```

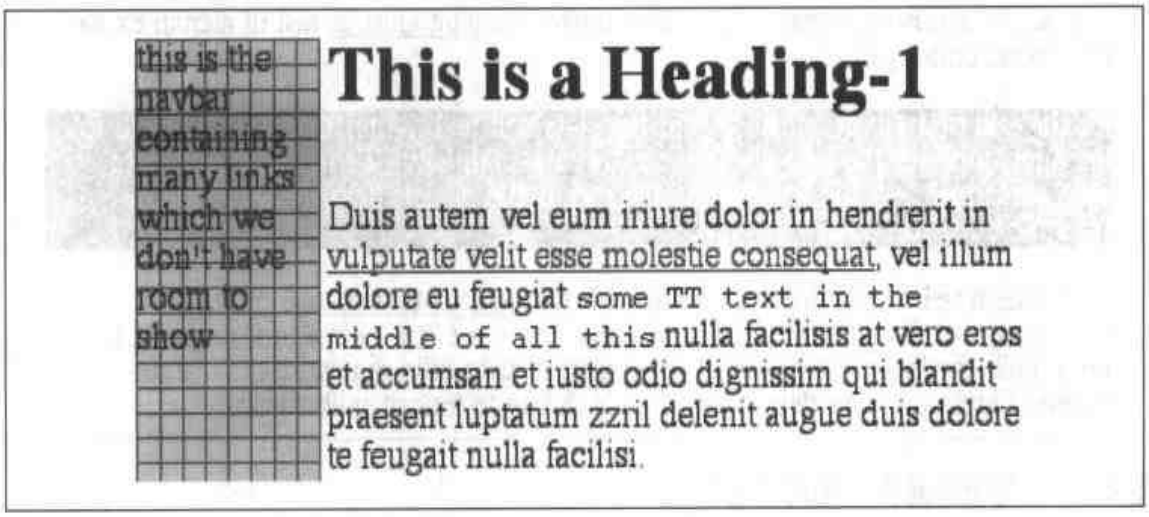


图 6-25 为表格单元设置背景图像

理论上，还可以将图像应用于像TEXTAREA和SELECT列表这样的替换元素的背景，但用户代理并不善于处理这种情况 —— 实际上，在写本书时，还没有浏览器能正确地将图像放置于表单元素的背景中。

警告：就像background-color，background-image不能被继承一样——实际上没有背景属性能被继承。当指定一个背景图像的URL时，同样要受url值的约束和影响：相对URL应该依据样式表来解释，但Navigator 4.x并不能正确地处理这一点，因此绝对URL应该更好一些。

背景实践

有趣的是，图像是放置于任何背景颜色之上的。如果完全平铺GIF，JPEG，或其他不透明的图像类型，那么颜色将被完全覆盖，因为背景图像会占满整个文档背景，而且它们都不透明，所以这些类型的图像带来的效果相同。然而，alpha通道的图像格式，如PNG，可以半透明或完全透明，这就可以将图像与背景色组合。另外，如果由于某种原因载入图像失败，那么用户代理会用背景色来代替图像的那部分位置。考虑下面例子中，如果载入图像失败，“布满星星的段落”会变成什么样。如图6-26所示。

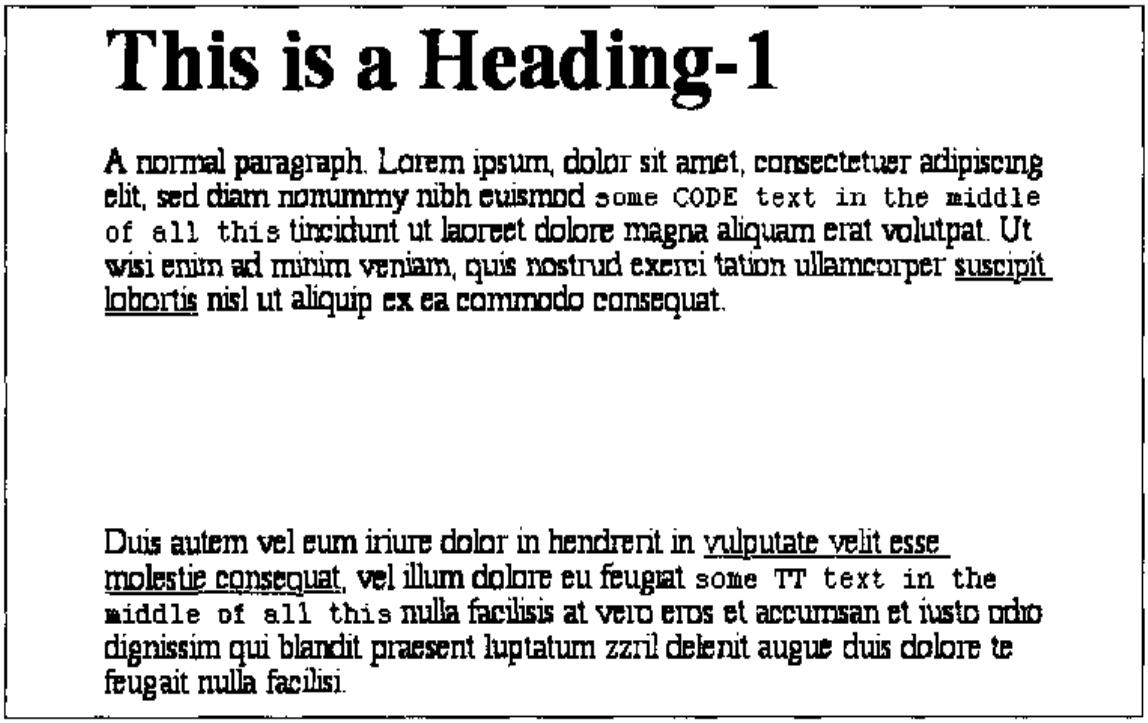


图6-26 缺少背景图像的结果

由于这一原因，在使用背景图像的时候，最好还是指定一种背景。这样，至少白色文本是可见的：

```
P.starry {background-image: url(http://www.site.web/pix/stars.gif);
   background-color: black; color: white;}

<P CLASS="starry">It's the end of autumn, which means the stars will be
brighter than ever!  Join us...
```

除了让图像完全覆盖整个文档背景外，还可以用另外的方式使用背景图像，这需要指定一种颜色用以覆盖图像之外的其他部分。

按方向重复图像

到此为止，我们只在背景中沿水平和垂直方向重复背景图像。如果想要得到“边栏”背景，则需要创建很短很宽的背景图像；也许这些图像的尺寸最好是10个像素高、2500像素宽。当然，此图像的大部分都是空白，只有左边大约100像素才包含“边栏”图像，图像的剩余部分也就浪费了。如图6-27所示。

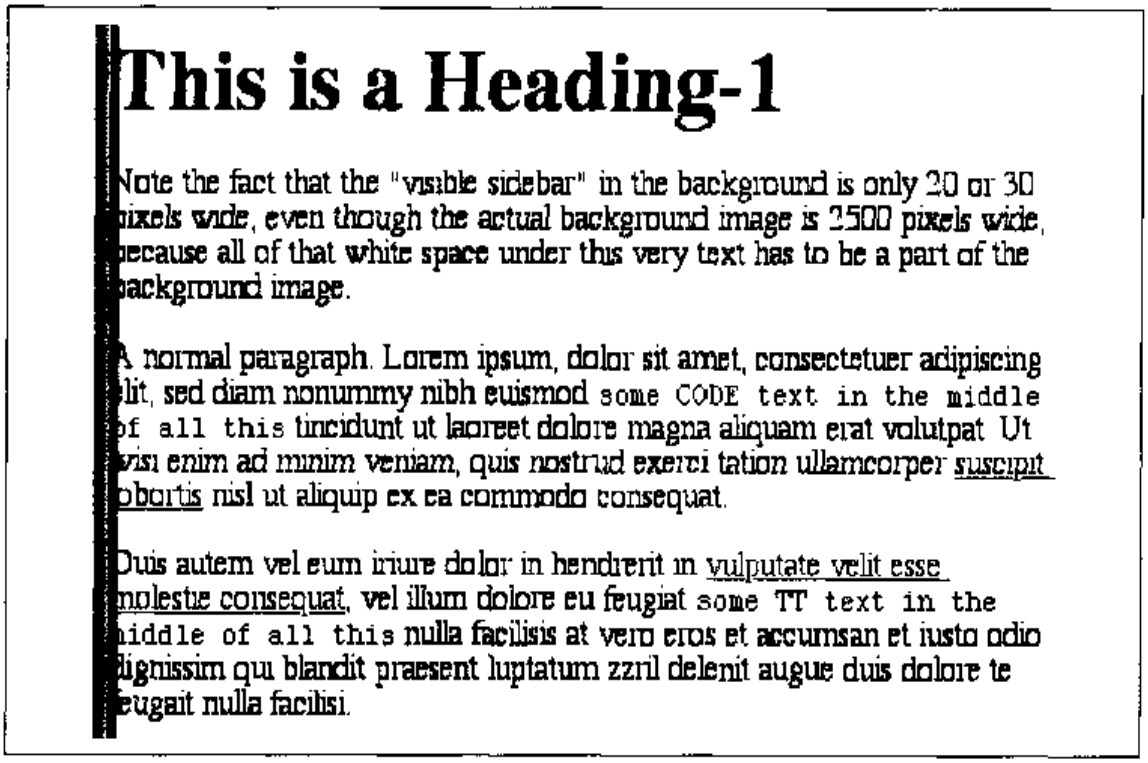


图6-27 为小范围的效果使用宽大图像

那么是不是创建一个10个像素高、100个像素宽的边栏图像会更好呢？也不用浪费空白位置，而且只将其在垂直方向上重复即可。这也会使设计工作更为简单，而且用户的下载时间也将减少许多。这就是background-repeat属性。

background-repeat	
允许值	repeat | repeat-x | repeat-y | no-repeat
初始值	no-repeat
可否继承	否
适用于	所有元素

repeat使图像在两个方向平铺，就像前面的背景图像那样。repeat-x和repeat-y使图像分别在水平和垂直方向上平铺，no-repeat禁止图像在任何方向上平铺。

缺省情况下，背景图像将从元素的左上角开始平铺。（我们会在后面讲到如何改变这种方式。）下面的规则将有如图6-28所示的效果：

```
BODY {background-image: url(yinyang.gif);
   background-repeat: repeat;}
```

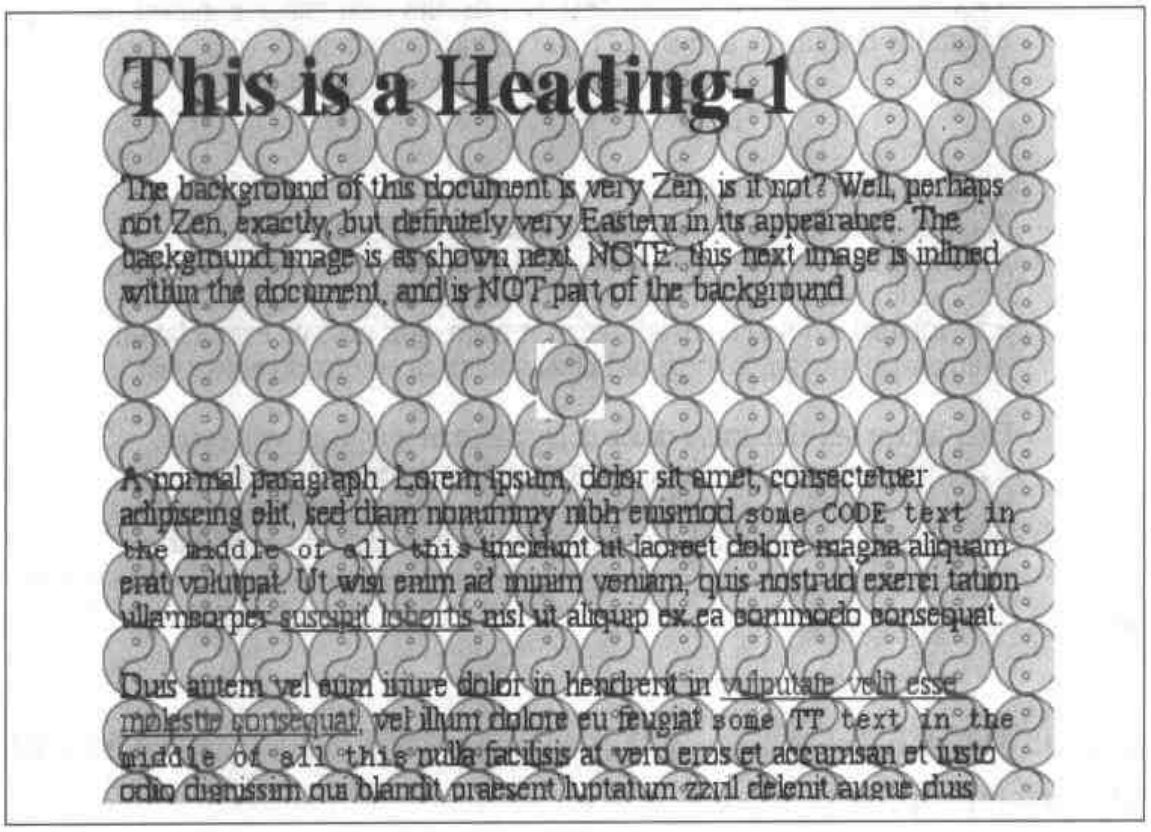


图6-28 在CSS中平铺背景图像

为了使规则尽量短，我们在这里忽略了背景色的设置，但要记住，在设置背景图像的同时要设置背景颜色。图 6-28 所示的效果与不使用 background-repeat 属性的效果一样，因为 repeat 是缺省值。

如果只想让图像往下平铺于文档的左边，那么不必再为此创建一幅特殊的图像，只需将规则稍做修改即可：

```
BODY {background-image: url(yinyang.gif);
   background-repeat: repeat-y;}
```

如图 6-29 所示，图像只是简单地沿着 Y 轴重复（即垂直方向）——在这种情况下，起点为浏览器窗口的左上角。

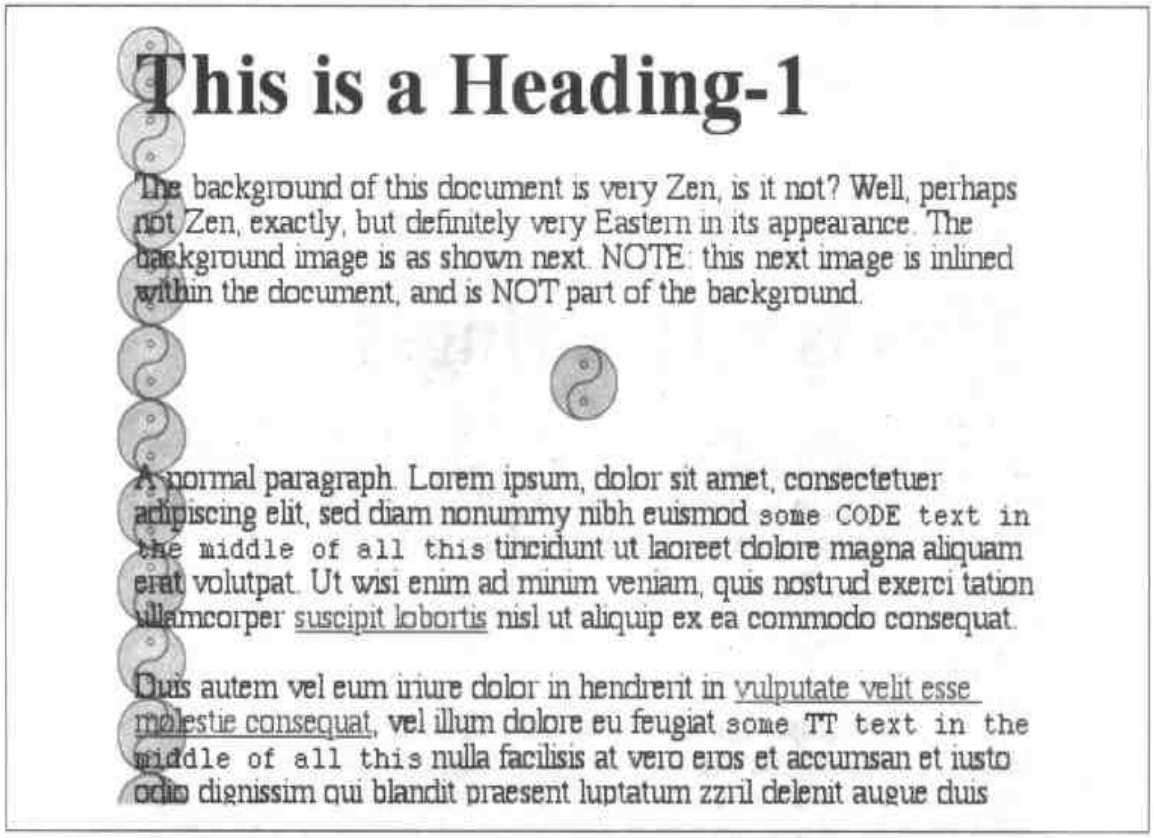


图 6-29 沿垂直方向平铺

在图中，只有垂直的一列图像。如果想要两列这样的图像，那么这个基本图像必须变为并排的两个符号，如图 6-30 所示。

将重复值变为 repeat-x：

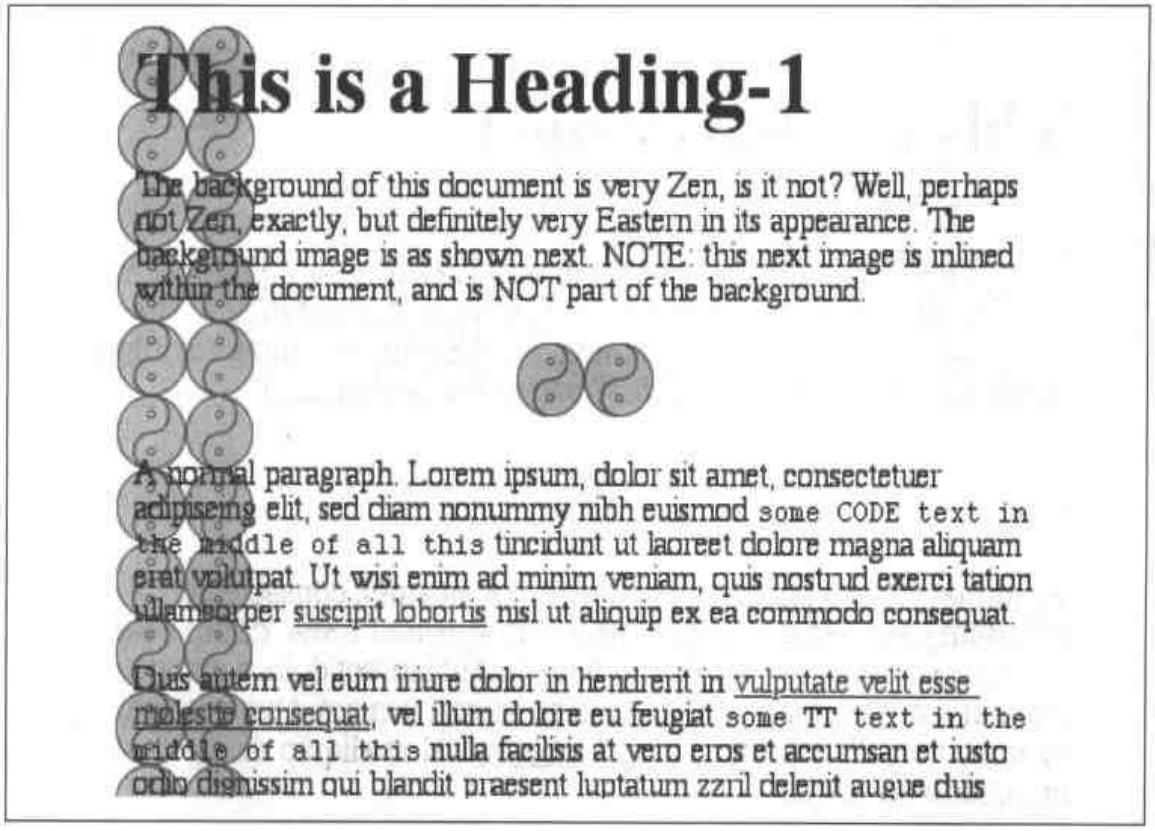


图 6-30　沿垂直轴平铺一幅稍大的图像

```
BODY {background-image: url(yinyang.gif);
   background-repeat: repeat-x;}
```

现在，图像将沿 x 轴（即水平方向）平铺，如图 6-31 所示。

当然，我们也可以选择不重复图像，即最后一种取值，no-repeat，其效果如图 6-32 所示：

```
BODY {background-image: url(yinyang.gif);
   background-repeat: no-repeat;}
```

这看起来似乎用处不大，因为只在文档的左上角出现了一个小小的符号。让我们来看一个更大一些的符号，其效果如图 6-33 所示：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;}
```

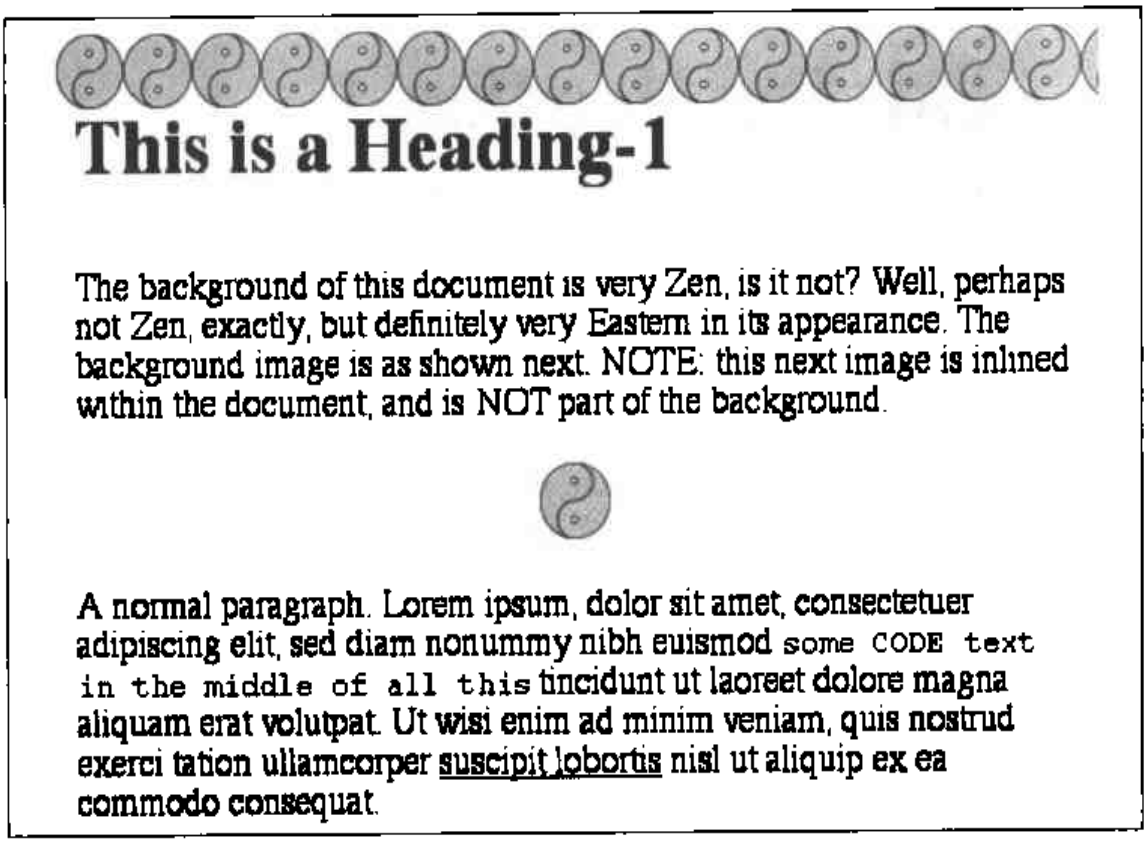


图 6-31 沿水平轴平铺

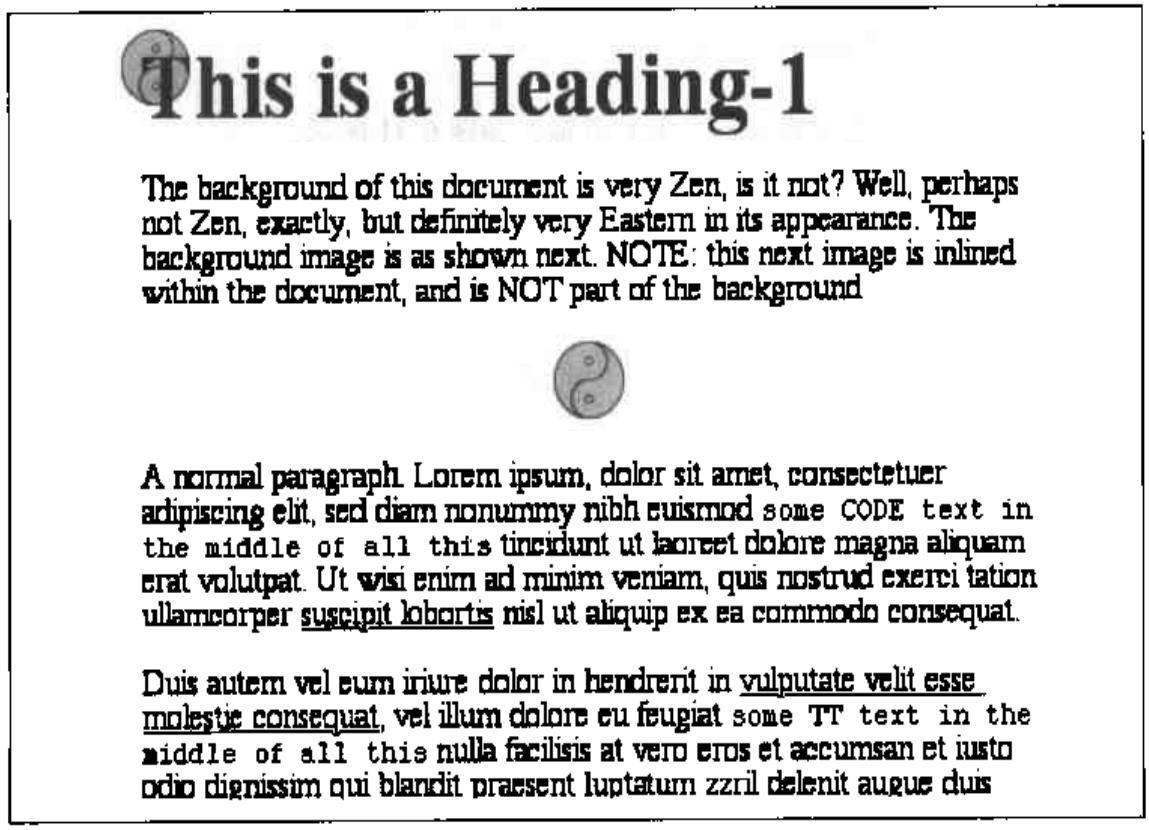


图 6-32 不平铺

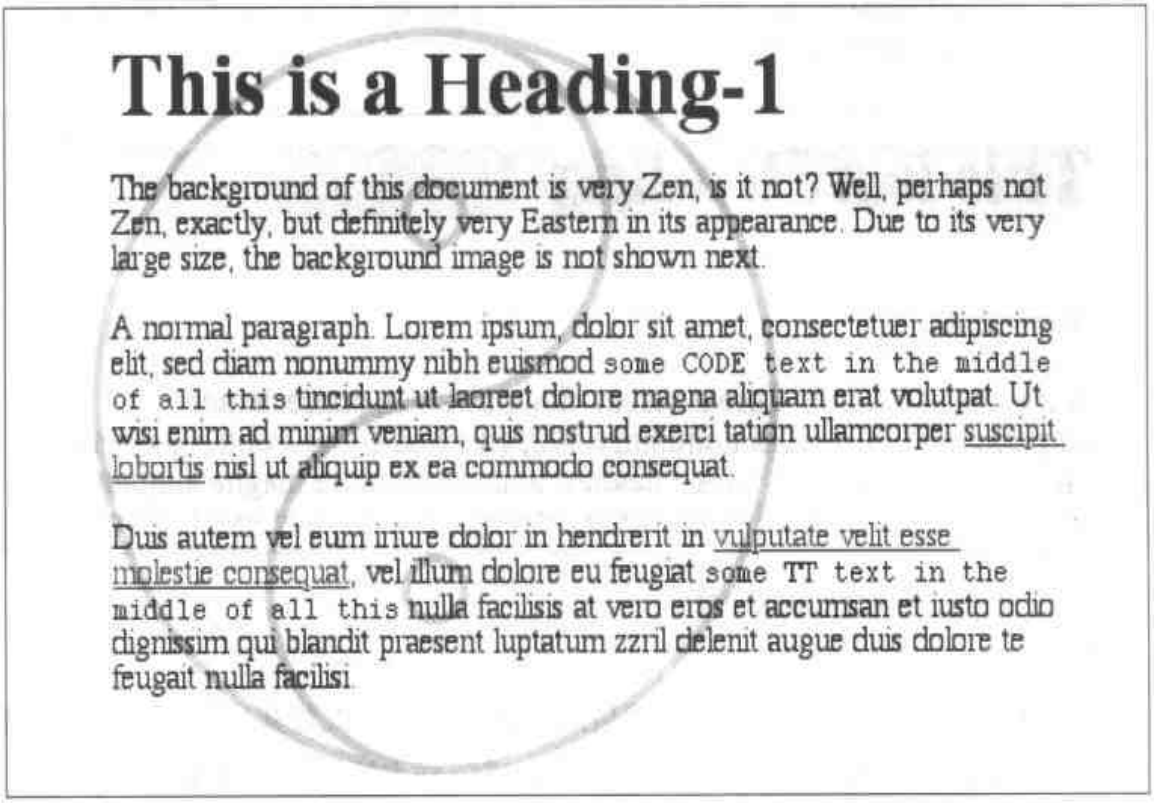


图 6-33　放置一个大的背景图像

实际问题

这种控制重复方向的能力在文档设计中能很好地扩展网页效果。例如，要在每个 H1 元素的左边放置一个三边框，那么可以如下声明：

```
H1 {background-image: url(triplebor.gif); background-repeat: repeat-y;}
```

如图 6-34 所示，小图像 triplebor.gif 在标题元素的左边沿垂直方向平铺，效果很好。

图 6-34　为 H1 元素创建一个“三边框”

甚至，我们可以沿着 H1 元素的顶端设置一个波浪框，效果如图 6-25 所示。图像的颜色同背景颜色相混合，产生出波浪效果：

```
H1 {background-image: url(wavybord.gif); background-repeat: repeat-x;
    background-color: #CCCCCC;}
```

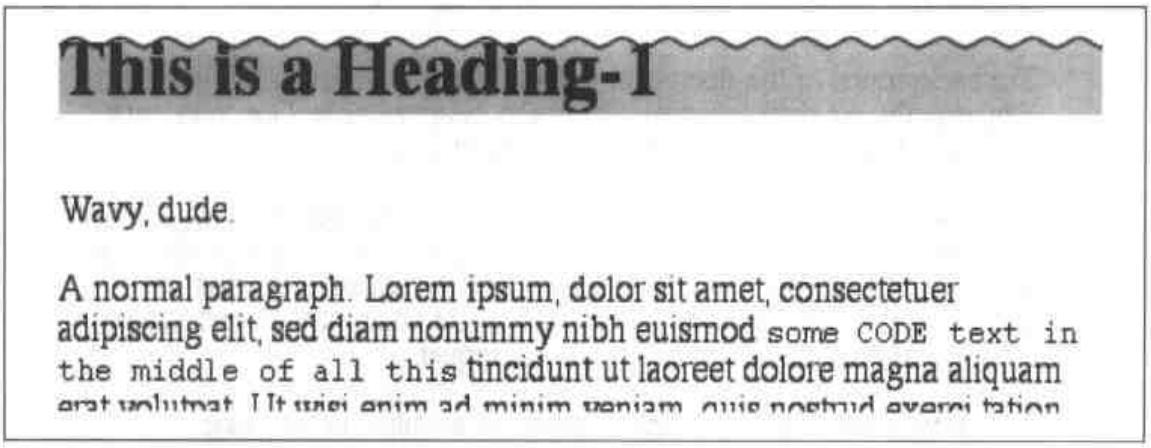


图 6-35 为 H1 元素设置波浪边框

只是简单地选择一些合适的图像，然后以某种新颖的方式将其应用于文档，就会带来一些令人惊奇的网页效果。现在我们知道怎样来控制背景图像的重复方式，那么又该怎样将其放置到背景的不同位置呢？

背景位置

使用background-repeat可以将一幅大图像放置于文档背景，而且可以禁止重复。于是我们可以先添加这幅图像，然后再改变图像在背景中的放置位置。

background-position

允许值	[<百分比> | <长度>]{1,2} | [top | center | bottom] | |[left | center | right]
初始值	0% 0%
可否继承	否
适用于	块级元素和替换元素

注意：百分比是指相对于元素和初始图像上的一个点的值(本章后面的“百分比值”一节将对此做解释)。

例如，可以将图像放于中心位置，如图6-36所示：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: center;}
```

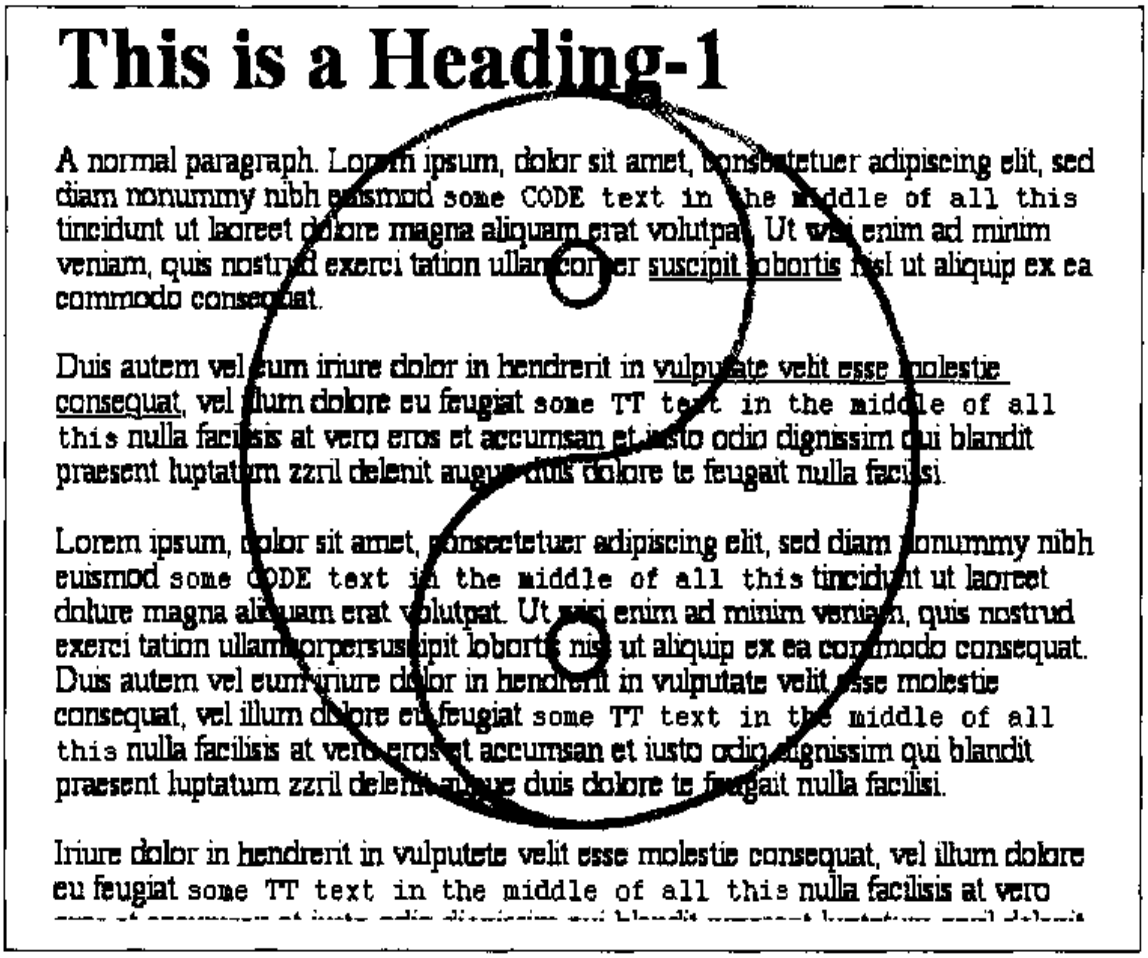


图6-36 居中图像

这种效果是由背景位置（background-position）属性来设置的，还有很多种方式可以将不同的值应用于这个属性。首先，关键字有top，bottom，left，right和center。通常它们都成对使用，但也有例外（如图6-36所示）。还可以是长度值，如50px或者2cm，最后是百分比值。每种值对于背景图像的放置都有着各不相同的效果。

关键字

最易懂的是关键字。它们的效果正如其名字一样。例如，top right使图像放于元素的右上方。看下面的例子：

```
BODY {background-image: url(yinyang.gif);
   background-repeat: no-repeat;
   background-position: top right;}
```

顺便说一句，图6-37中的效果同right top的设置一样。当使用位置关键字时，可以使用任何顺序，只要不超出两个关键字即可，一个用于水平方向，另一个用于垂直方向。

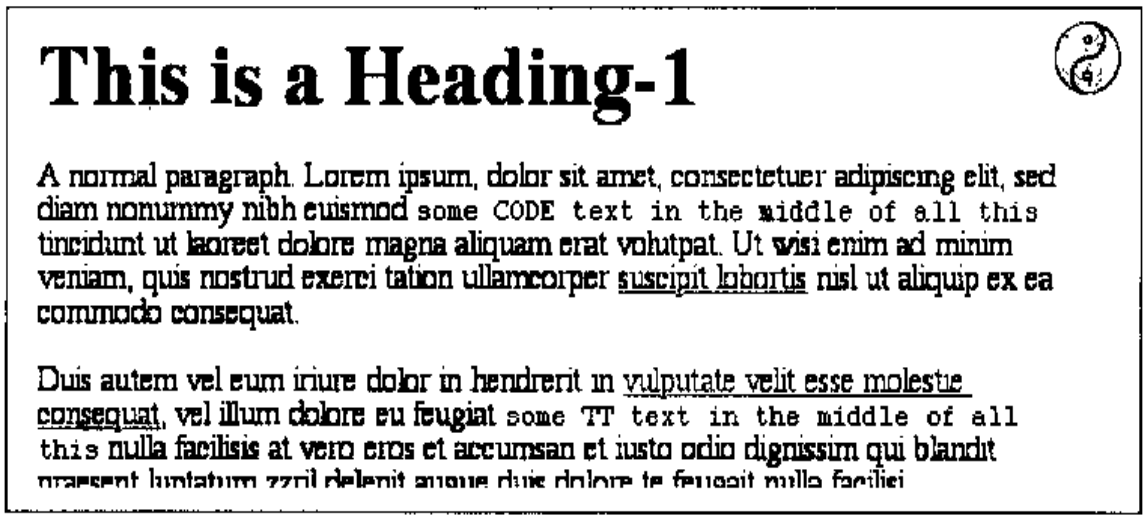


图6-37 将背景图像放置于浏览器窗口的右上角

如果只有一个关键字，那么另一个关键字缺省为center。表6-1两边的关键字等价。

表6-1 位置关键字对照表

单个关键字	等价的双关键字
center	center center
top	top center center top
bottom	bottom center center bottom
right	center right right center
left	center left left center

如果要让图像出现在每个段落的中上方，如图 6-38 所示，只需如下声明：

```
P {background-image: url(bg23.gif);
   background-repeat: no-repeat;
   background-position: top;}
```

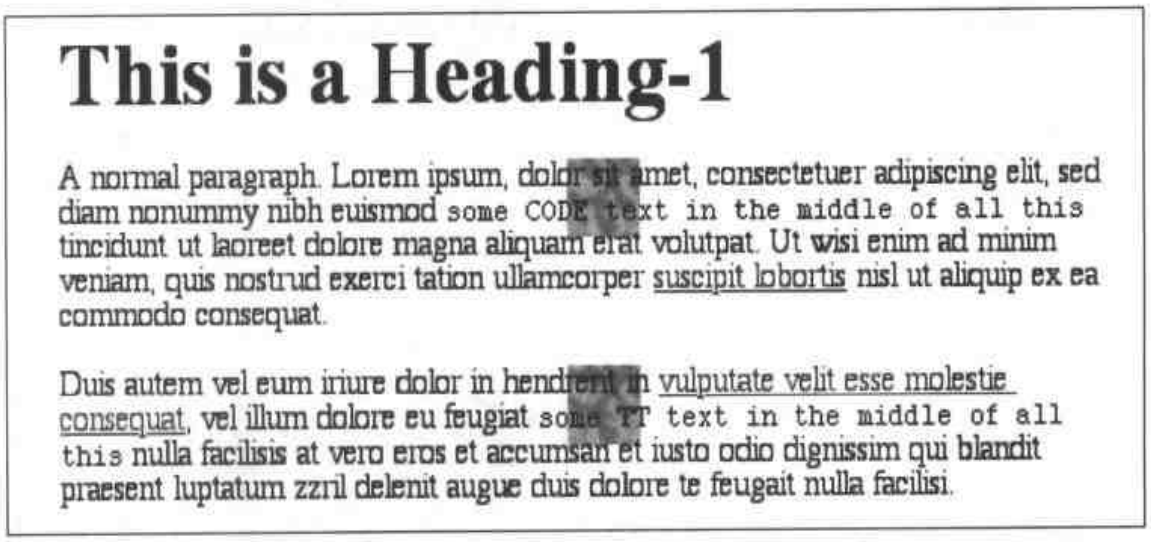


图 6-38　将背景图像置于段落的中上方

百分比值

百分比值同关键字很接近，但其操作方式不一样。用百分比值来居中一幅背景图像，也很简单：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: 50% 50%;}
```

这使得背景图像的中心同其父元素中心对齐，如图 6-39 所示。换句话说，百分比值同时应用于元素及其背景图像。

为理解这一概念，让我们来仔细观察其过程。当在某一元素里居中背景图像时，图像中被描述为 50% 50% 的点将与元素中同样描述的点对齐。如图 6-40 所示。

因此，要使背景图像横跨元素的三分之一，纵跨三分之二，可以按如下声明：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: 33% 66%;}
```

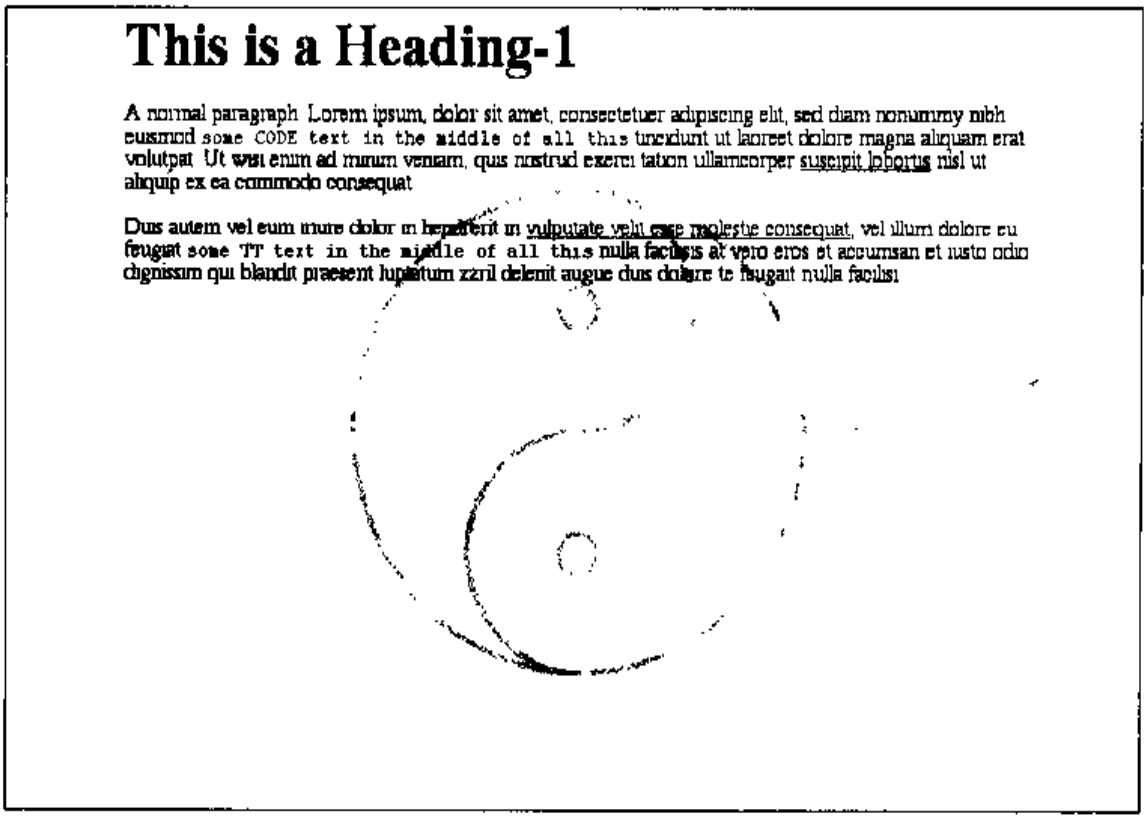


图 6-39 使用百分比值居中背景图像

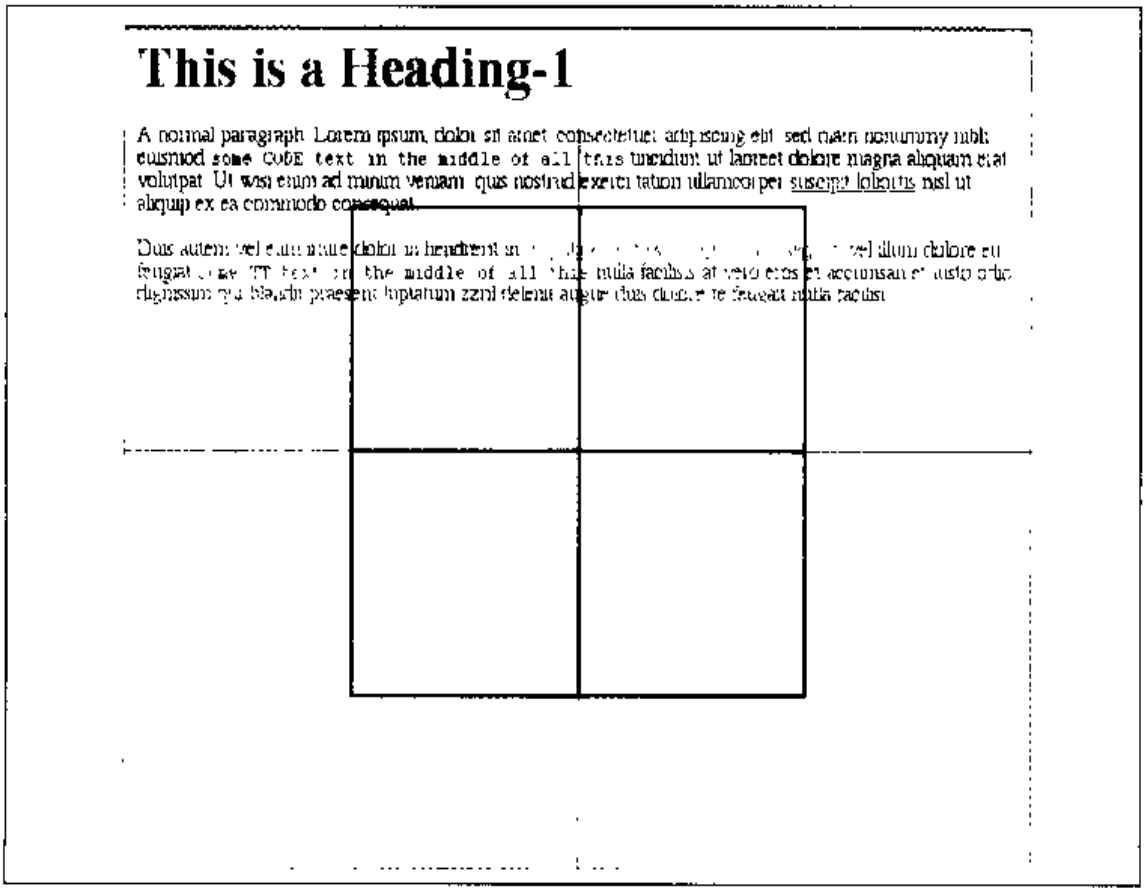


图 6-40 对齐图像中心和浏览器窗口中心

这就使得图像上相对于左上角水平为三分之一，垂直为三分之二的那个点与元素中的同样的点重合，如图 6-41 所示。

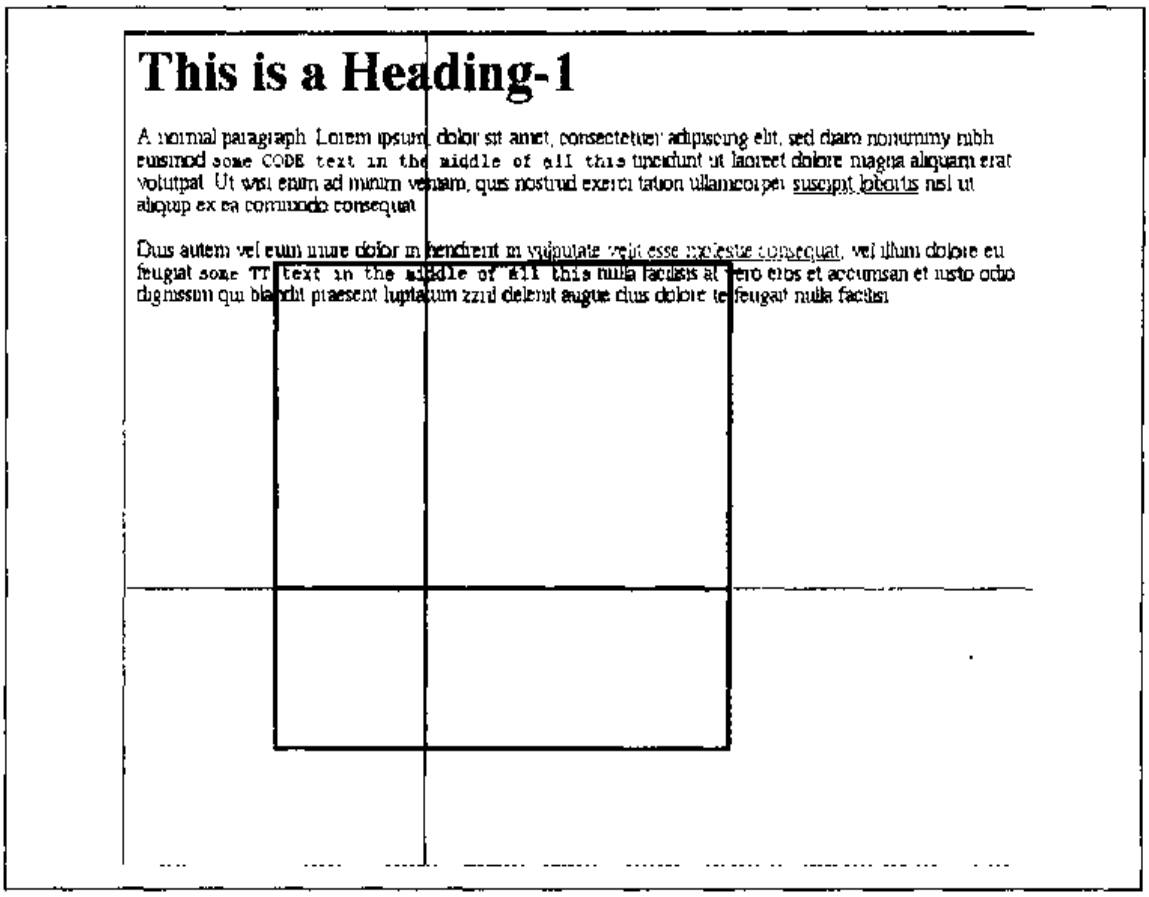


图 6-41 更多的百分比位置

注意，百分比的水平值总是在前面。如果将上例中的这两个百分比值交换位置，那么背景图像将被放于水平三分之二，垂直三分之一处。同样，当只给出一个百分比值时，这个值为水平值，而另一个垂直百分比值被假定为50%。这同关键字一样，当只有一个关键字给出时，另一个被假定为 center。如下：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: 33%;}
```

则背景图像会放置于水平三分之一，垂直二分之一处，如图 6-42 所示。

表 6-2 给出了关键字和百分比值的对照比：

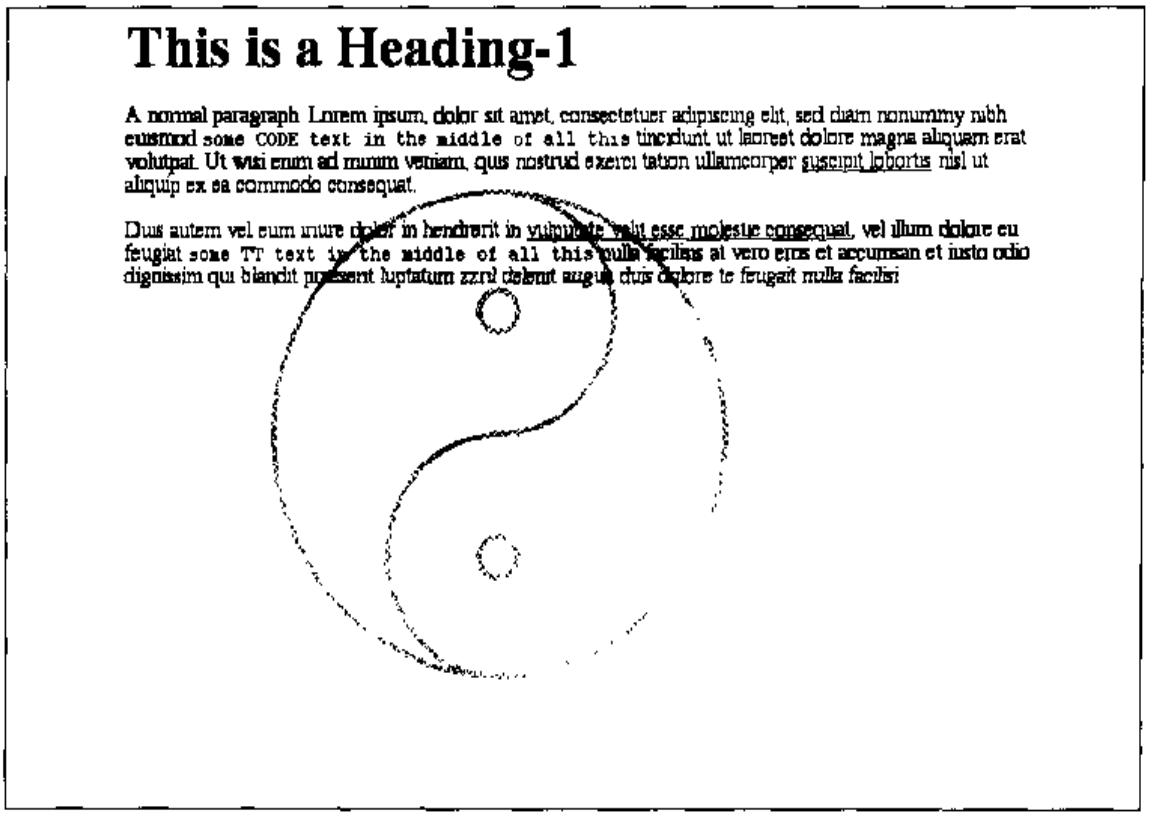


图 6-42 只有一个百分比值意味着垂直百分比为 50%

表 6-2 位置对照表

单一关键字	等价的双关键字	等价的百分比
Center	center center	50% 50%
Top	top center	50% 0%
	center top	
Bottom	bottom center	50% 100%
	center bottom	
Right	center right	100% 50%
	right center	100% 50%
left	center left	0% 50%
	left center	0%
	top left	0% 0%
	left top	
	top right	0% 100%
	right top	

表 6-2 位置对照表（续）

单一关键字	等价的双关键字	等价的百分比
	bottom right right bottom	100% 100%
	bottom left left bottom	0% 100%

background-position的缺省值为 0% 0%，和top left的功能一样。这也是为什么在设置百分比值时，图像总是相对于元素的左上角开始平铺的原因所在。

长度值

最后，我们来讨论长度值。当用长度值来决定背景图像的位置时，它们被解释为相对于元素左上角的偏移量。背景图像上的偏移点也为左上角的那个点。因此，如果设置为20px 30px，那么图像的左上角将位于元素左上角往右20个像素，往下30个像素的位置，如图 6-43 所示：

```
BODY {background-image: url(bg23.gif);
   background-repeat: no-repeat;
   background-position: 20px 30px;}
```

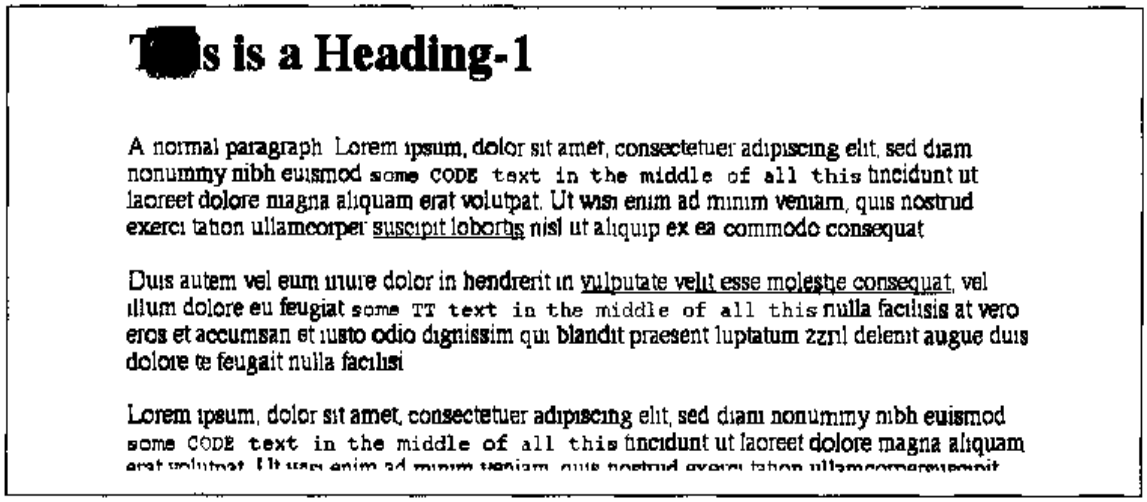


图 6-43 背景图像的偏移量

这不同于百分比值，因为偏移量只是简单地表示为一个左上角相对于另一个左上角的位移。换句话来说，是背景图像的左上角与声明中的偏移量所对应的元素上

的一个点对齐。也可以组合长度值和百分比值，以得到更棒的效果。比如，要想使一幅背景图像总是位于元素右边，且离顶端10个像素的位置，如图6-44所示，只需按如下声明：

```
BODY {background-image: url(bg23.gif);
   background-repeat: no-repeat;
   background-position: 100% 10px;}
```

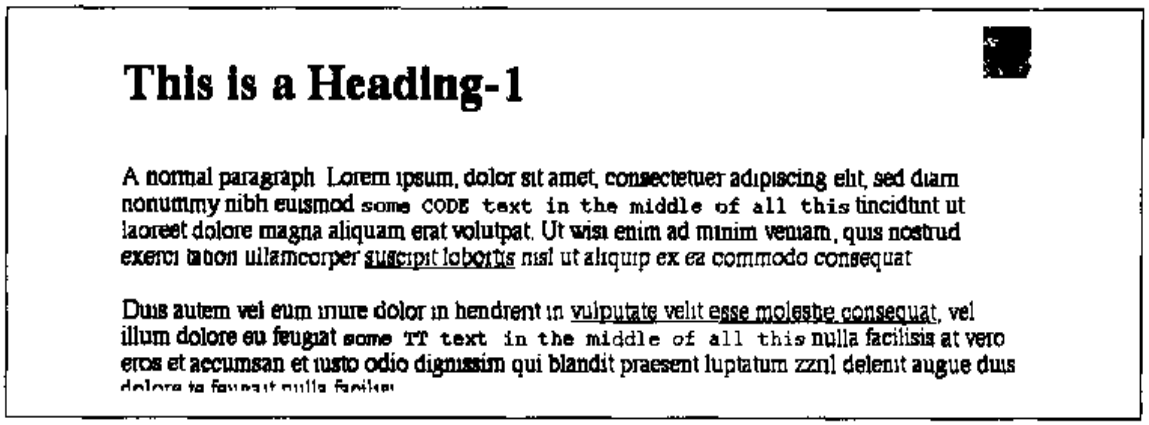


图 6-44 百分比与长度值组合

但是，不能将关键字同其他值组合。如，top 75%就是无效值。所以关键字只能单独使用，但百分比却可以和长度值组合使用。

而且，长度值和百分比值还可以为负，这将使图像以某种程度偏离元素。例如，我们想在背景中放置一个很大的阴阳八卦图，但只想让其一部分显示于元素的左上角，那该怎么办呢？没问题。假设图像为300像素高，300像素宽，而且假设只有其右下角三分之一是可见的。我们可以用下面的声明来得到想要的效果（如图6-45所示）：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: -200px -200px;}
```

或者，想看到更多的部分，如图6-46所示：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: -150px -100px;}
```

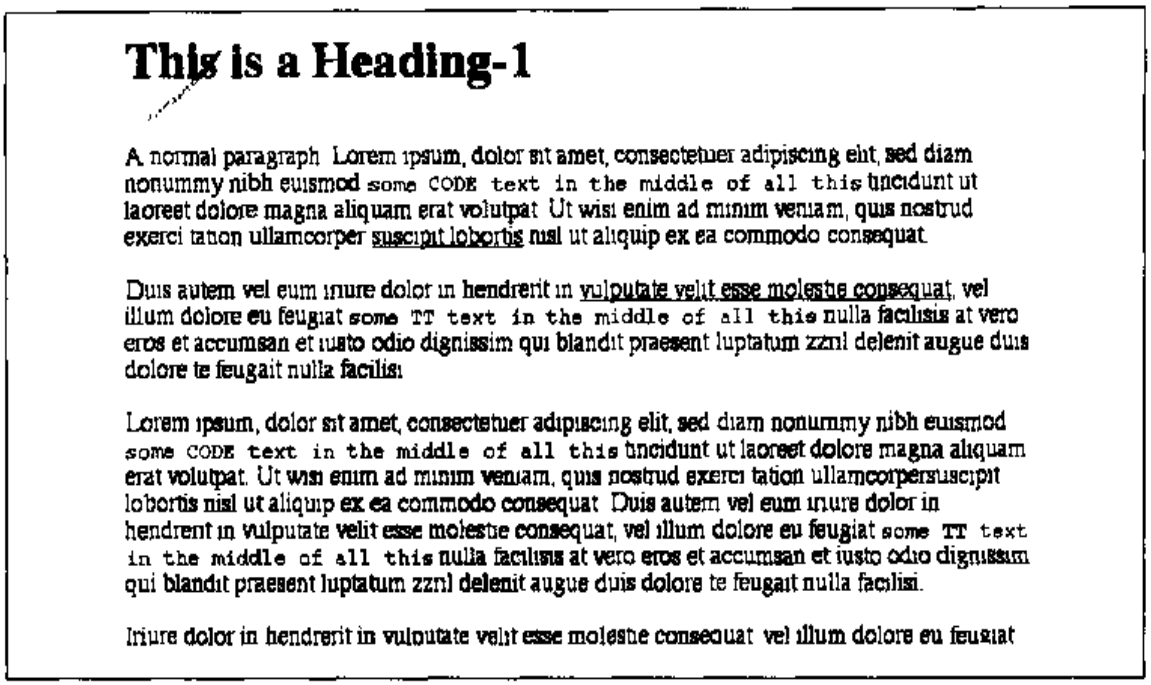


图 6-45 使用负长度值

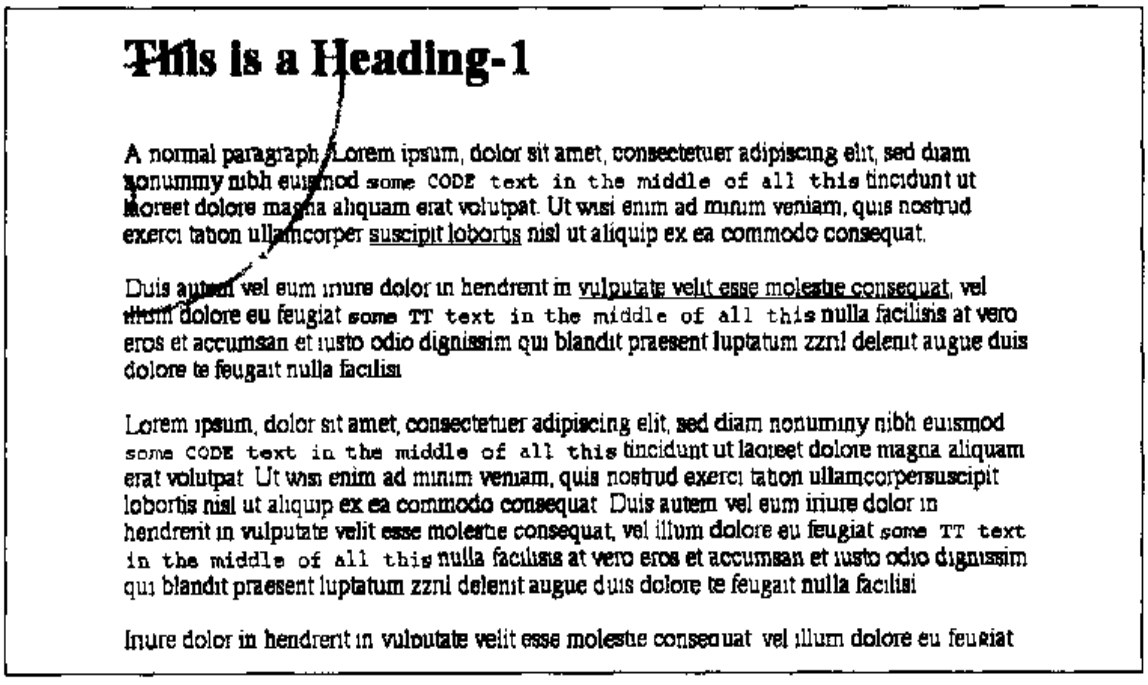


图 6-46 另一组负长度值

负百分比在理论上也是可行的，但有两个问题。第一就是受用户代理的限制，它可能识别不了负值，另外就是负百分比值的计算值不唯一。如图 6-47 所示。

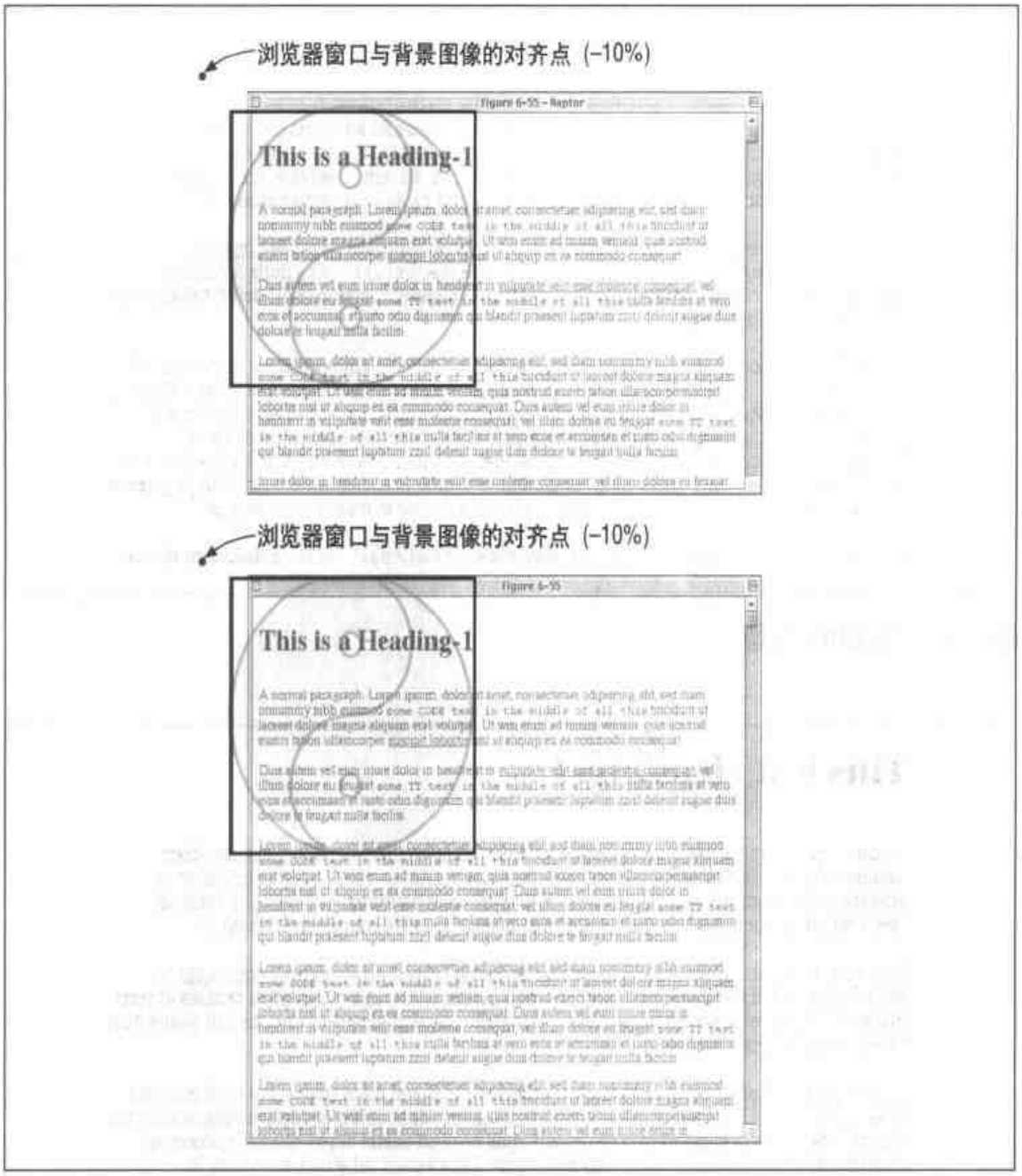


图 6-47　对齐负百分比点：两种情况

当然，并不是不能用负值，只是在使用时要注意这些问题。

警告： 尽管多数支持 CSS 的浏览器（Explorer 4.x 和 5.x 以及 Opera 3.5 和后续版本）都支持负的背景位置，但其效果却不可预测。甚至负长度值也可能导致不可预测的效果。如果读者想取得部分背景图像可见的效果，那么最好是先产生这个部分图像，再按常规的方式放置。

在本节中，所有的例子的重复值都为no-repeat。其原因很简单：当只有一幅背景图像时，更容易看清其放置效果。但我们也不必总是禁止图像的重复：

```
BODY {background-image: url(bigyinyang.gif);
   background-position: -150px -100px;}
```

从图6-48中可以看到，平铺式样是从background-position所指定的位置开始的。第一幅图像也被称作起始图像，这一点对于理解下一节相当重要。

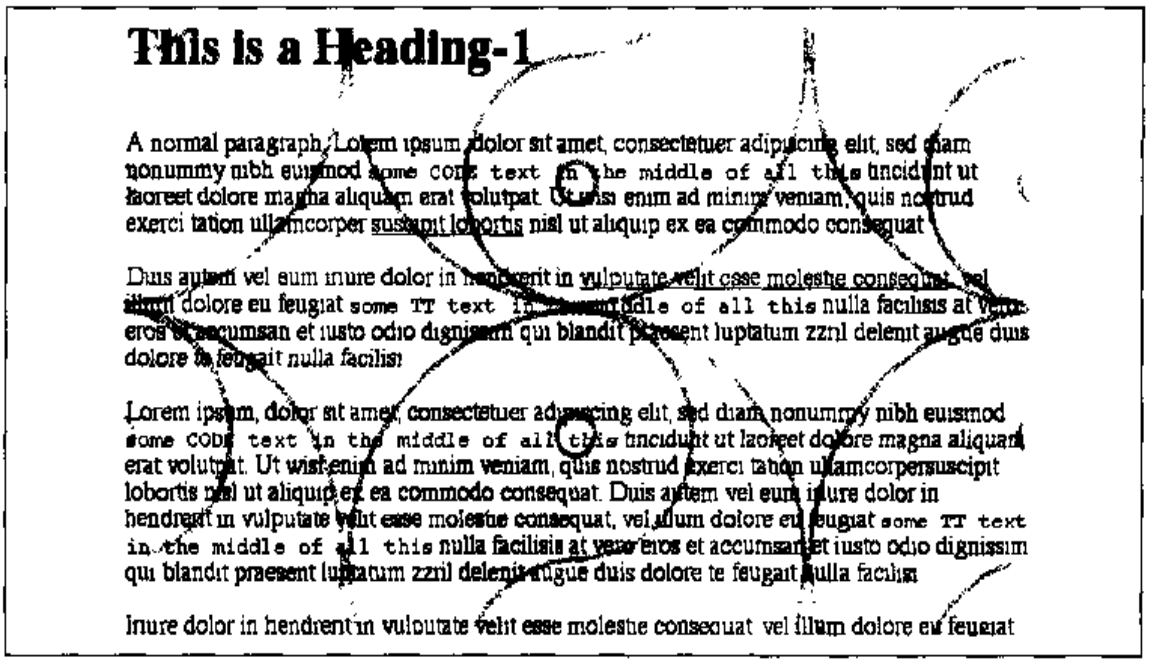


图6-48 背景位置属性用于设置平铺式样的起点

的确如此，背景的位置和我前面介绍的有些出入。background-position是唯一受约束的背景属性，因为它只能应用于块级元素和替换元素；背景图像位置不能在像超链接这样的内联元素中调整。

重复图像（再次讨论）

在前面，我们讨论了repeat-x、repeat-y和repeat，以及它们如何影响背景图像的平铺。而且在前面的例子中，平铺总是从元素（如BODY）的左上角开始。当然，并不是必须这样做。由于background-position的缺省值为0% 0%，因此，除非改变起始图像的位置，才能改变平铺图像的起始点。由于我们已经知道如何改变起始图像的位置，所以我们需要清楚用户代理是怎样处理这种情况的。

我们先来看一个例子，然后再来解释它。考虑下面的样式，其结果如图6-49所示：

```
BODY {background-image: url(bg23.gif);
   background-repeat: repeat-y;
   background-position: center;}
```

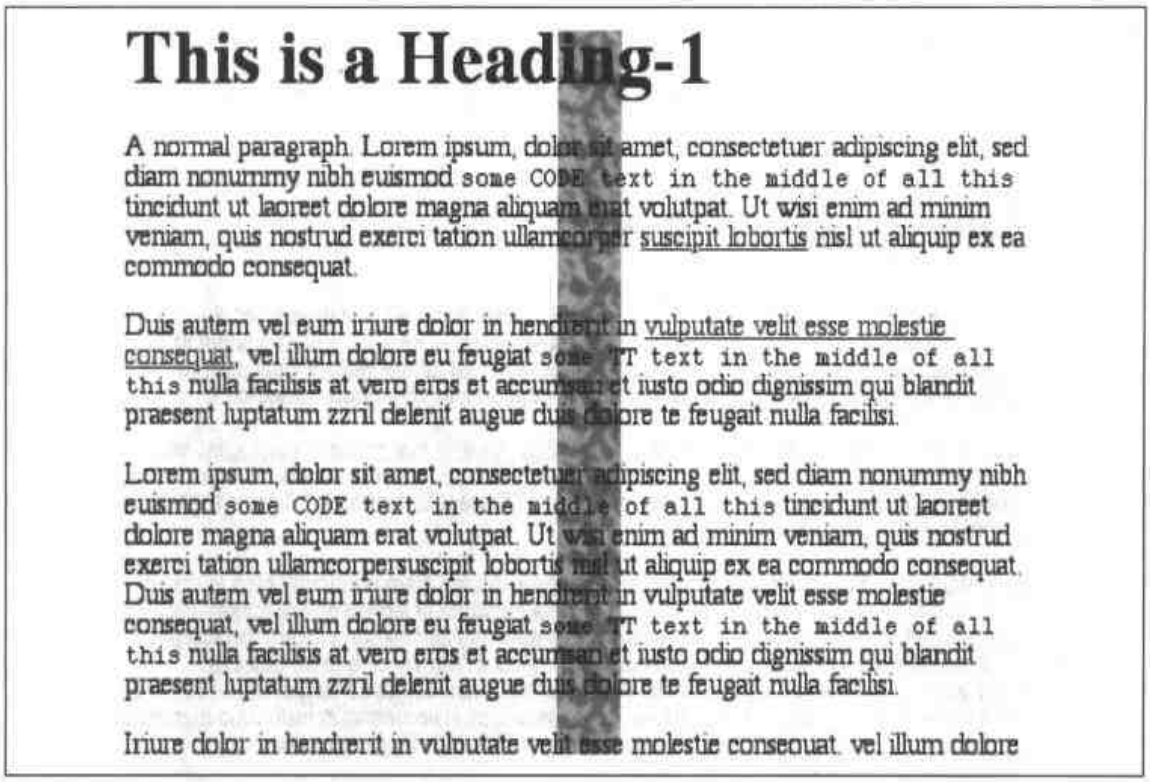


图 6-49　居中起始图像，再垂直方向重复

得到的效果是：文档正中间有一条竖条纹。看上去似乎有错，其实不然。

图 6-49 中的结果是正确的，因为起始图像已经被放置于 BODY 元素的中央，然后再沿 y 轴方向平铺开来 —— 即同时向上和向下平铺。同理，当为水平重复时，背景图像将沿左右方向平铺，如图 6-50 所示：

```
BODY {background-image: url(bg23.gif);
   background-repeat: repeat-x;
   background-position: center;}
```

因此，如在 BODY 元素中居中一幅大的图像，再让其重复，这将使它在四个方向上平铺开来。唯一不同的就是平铺的起始点。图 6-51 显示了从 BODY 中央与从 BODY 左上角开始平铺的不同效果。

注意浏览器窗口边沿的异同。当背景图像沿中央重复时，视野中的栅格也同时居中，结果导致四周边缘保持一致的“剪切”形式。这些变化是很巧妙的，但在设计过程中，你肯定有充分的理由来运用这样的方法。

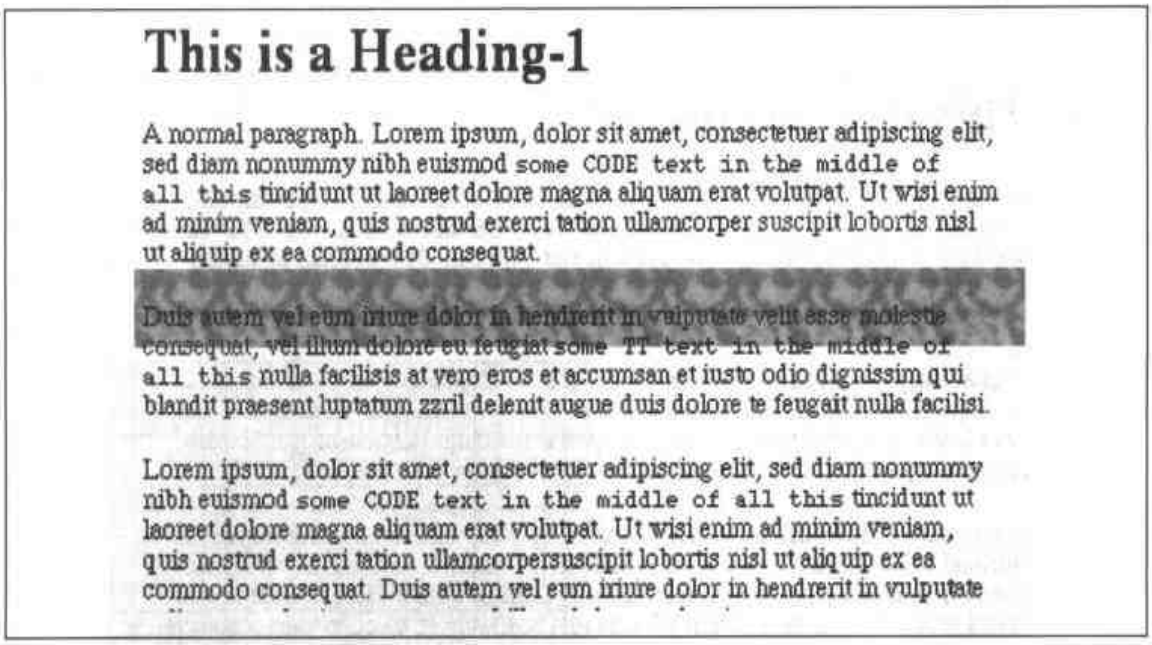


图 6-50　带水平重复的居中

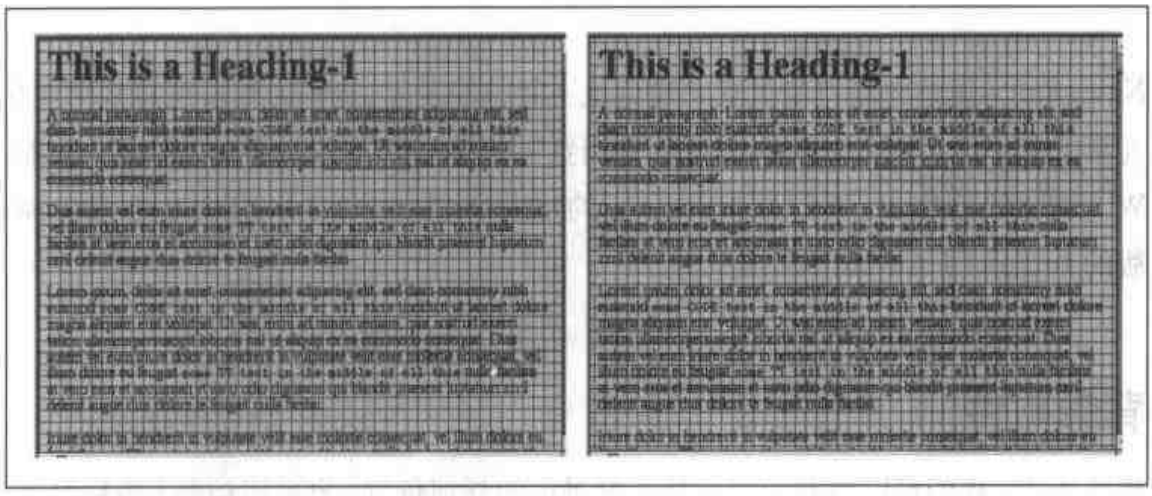


图 6-51　从左上角与从中央开始重复的区别

除此之外，再也没有别的方法能够控制重复方式了，比如说，repeat-left就无效，尽管可能在CSS的将来版本中会加进这一特性。但就目前来说，只有完全平铺、垂直平铺、水平平铺和不平铺四种方式。

重复中的实际问题

但就具体的浏览器而言，还有值得注意的地方。首先，Navigator 4 和 Internet Explorer 4 只支持向下和向右平铺。如果使用 Internet Explorer 4，而且在背景中居中一幅图像，然后再平铺，那么会得到如图 6-52 所示的效果。

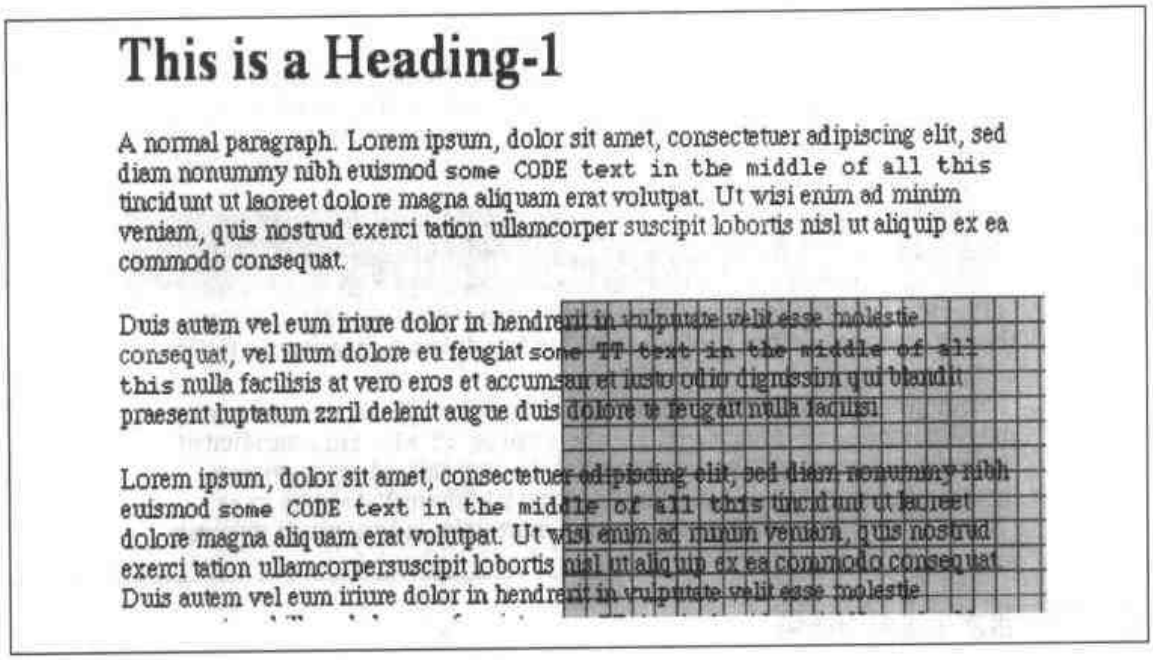


图 6-52 Internet Explorer 4 中的错误行为

Navigator 4通过忽略背景位置来避免这一问题的发生，也即起始图像总是出现在元素的左上方。在写本书时，只有四种浏览器能够正确处理重复及背景图像位置：Windows版的Opera 3.6，Macintosh版的Internet Exporer 4.5和5，以及Windows版的 Internet Explorer 5。

背景附件

到此为止，我们可以将起始图像放在背景中的任何位置，也可以控制（某种程序）它的平铺方式。但对很长的文档来说，如果居中一幅背景图像，那么在初始显示时，读者可能看不见背景图像。毕竟，浏览器只提供一个有限的文档窗口。如果文档太长，那么用户可以来回滚动，而文档中心可能要向下滚动两三屏才能见到，或者刚好就在浏览器窗口的下方，如图 6-53 所示。

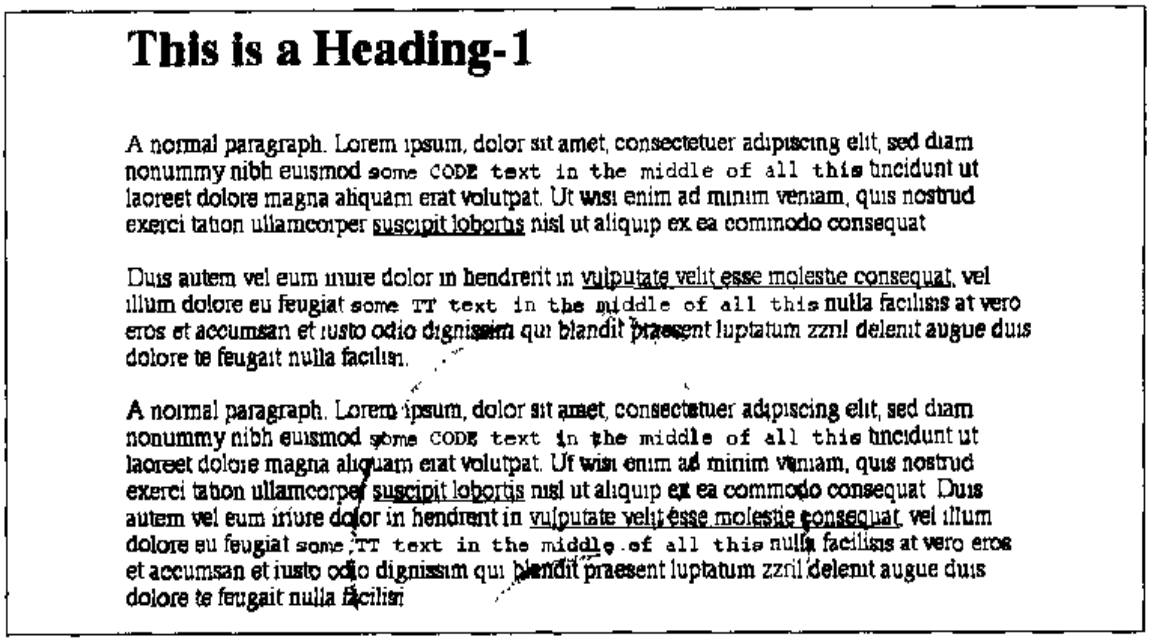


图 6-53 背景图像太低而不能完全可见

即使背景图像初始时可见，但还是会由于文档的滚动而出现图 6-54 所示的情形，这是读者所不愿见到的：

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: center;}
```

不要担心：有一种方法可以阻止这种滚动。

background-attachment

允许值	scroll | fixed
初始值	scroll
可否继承	否
适用于	所有元素

使用background-attachment属性，可以使背景图像固定在视野范围内，以避免出现因滚动而消失的效果：

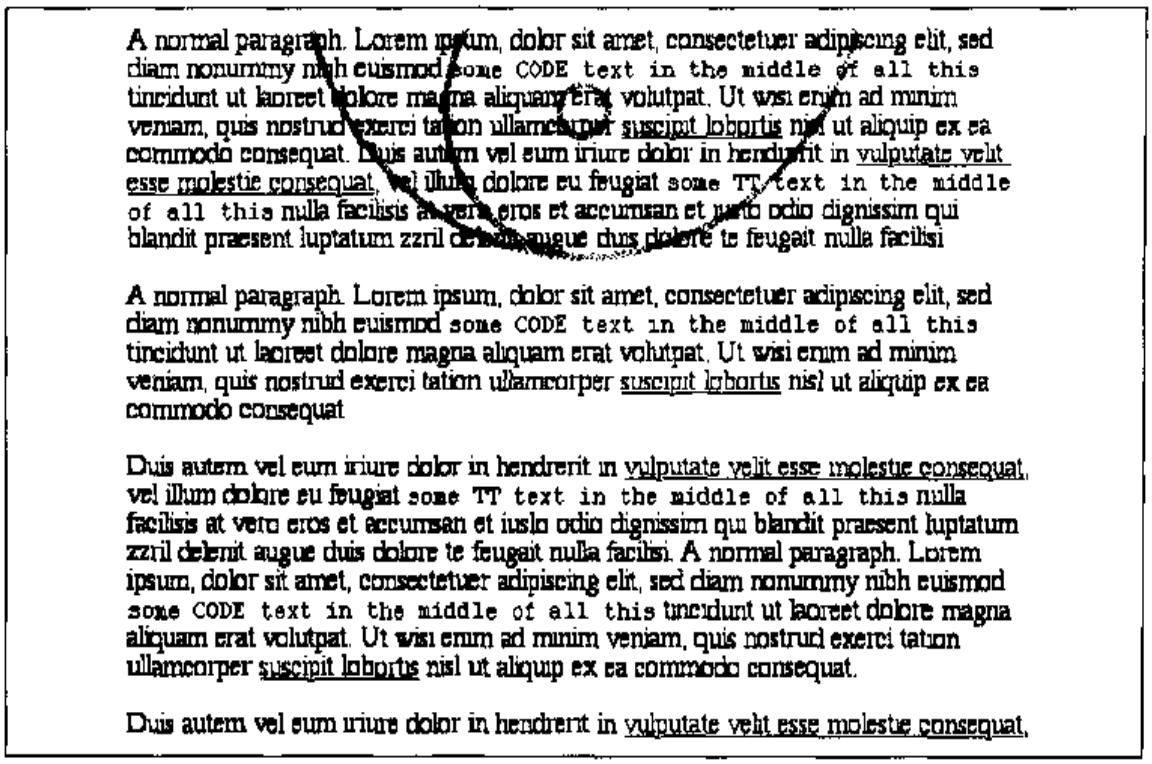


图 6-54 背景图像逐渐滚动出视野

```
BODY {background-image: url(bigyinyang.gif);
   background-repeat: no-repeat;
   background-position: center;
   background-attachment: fixed;}
```

这样做可以立即得到两种效果，如图6-55所示。第一就是背景不随文档滚动，第二就是图像的位置由视野范围决定，而不是由文档尺寸来决定。

当打印文档时，这两种效果依然有效，因为显示区域（打印纸）同文档尺寸一样。在浏览器中，可视范围可以通过重调窗口尺寸来改变。这会导致起始图像随窗口尺寸的变化而变化。图6-56描述了同一文档的几种情况。因此，从某种意义上来讲，图像并非固定不动，只是在可视范围固定的情况下，才保持固定状态。

background-attachment属性的另一个取值为scroll，也是缺省值。它使背景同文档一起滚动，而且当窗口尺寸变化时，不需变化起始图像的位置。如果文档宽度是固定的（或许是由BODY属性显式声明的），那么调整可视范围将不影响起始图像的位置。

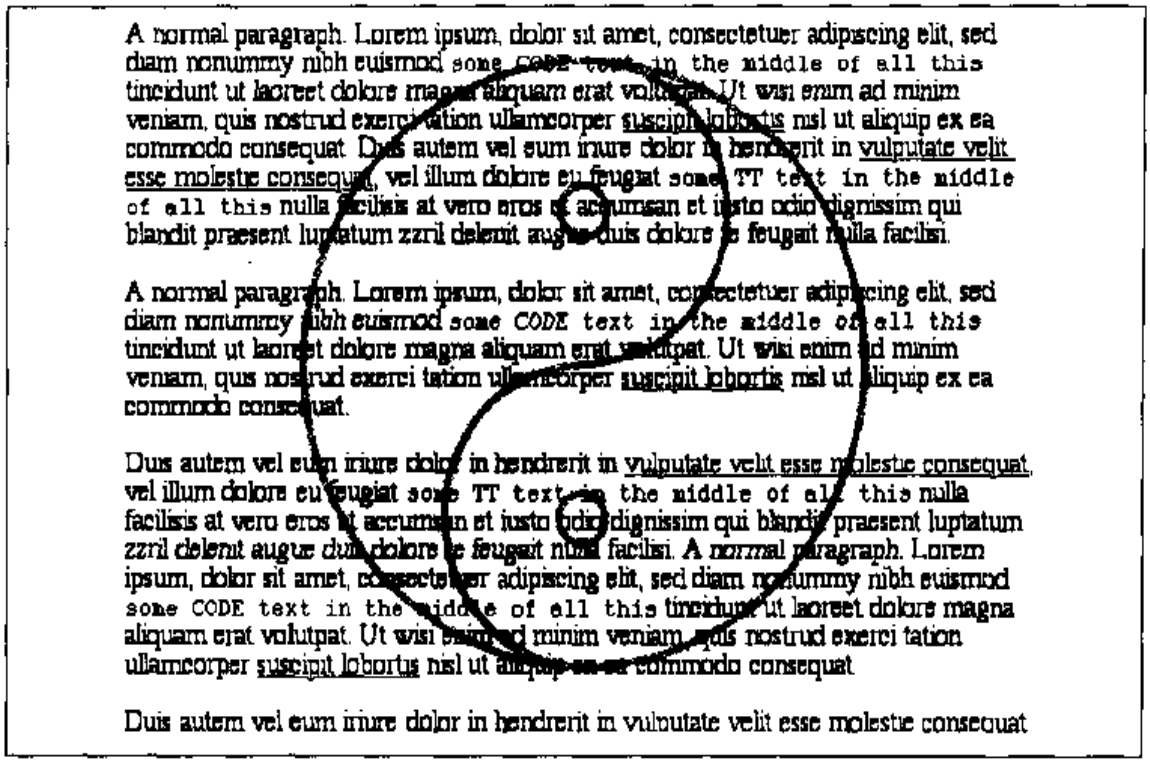


图 6-55 将背景固定在一个位置

有趣的效果

从技术上来讲，背景图像设为 fixed 是相对于可视范围而言的，而不是相对于包含它的元素本身。而背景图像只可能在包含它的元素内才是可见的。这就导致了很有趣的结果。

假设有一个平铺背景的文档，而且 H1 元素也有同样模式的平铺背景，只是颜色不同。BODY 和 H1 元素都被设置为固定背景，如图 6-57 所示：

```
BODY {background-image: url(tile1.gif);  background-repeat: repeat;
   background-attachment: fixed;}
H1 {background-image: url(tile2.gif);  background-repeat: repeat;
   background-attachment: fixed;}
```

这种对齐方式可能吗？当背景固定时，元素的位置也就定下了。因此，两个背景都从文档的左上角开始平铺。对于 BODY 元素，我们可以见到其整个重复模式。而对于 H1 元素，背景中可见的只是填充和 H1 元素的文本。因为它们的背景图像都是同一尺寸，而且它们的初始位置也一样，所以它们也似乎经过对齐，如图 6-57 所示。

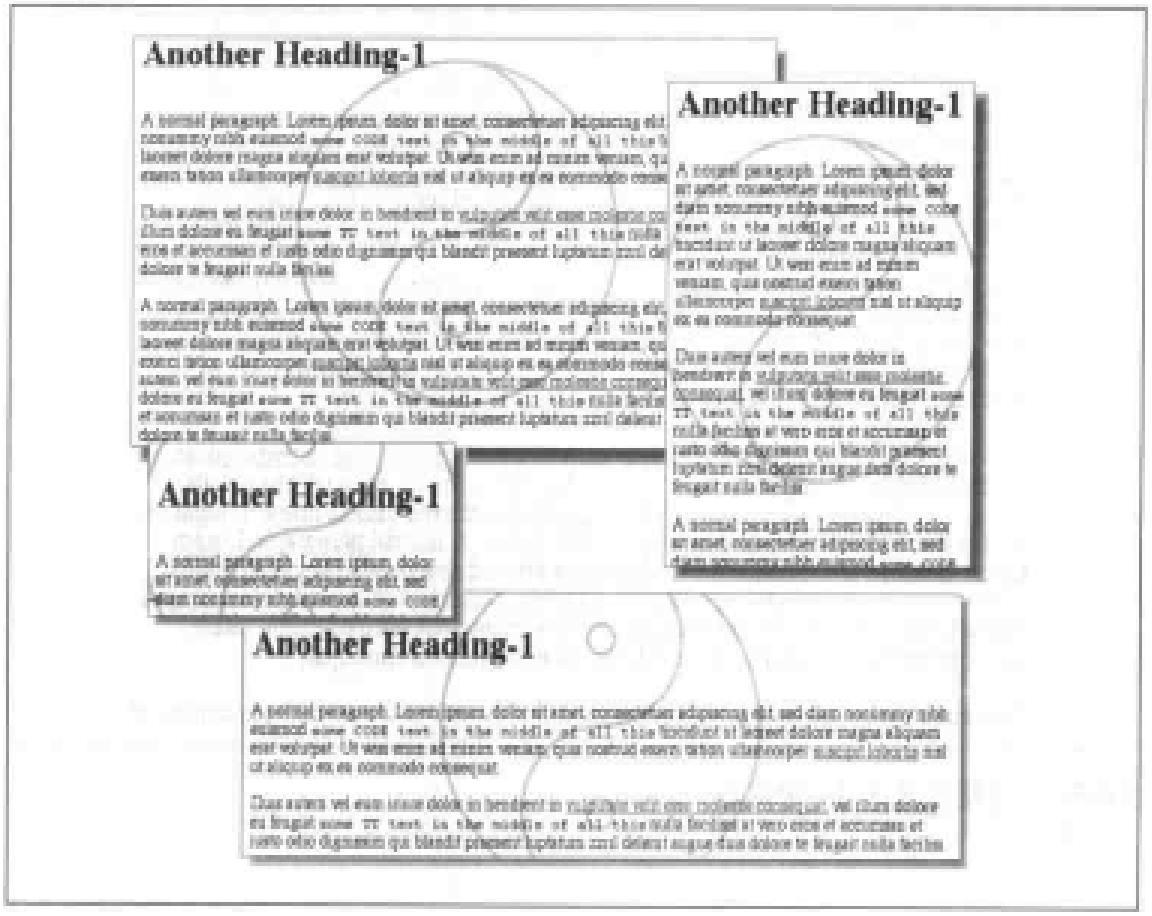


图 6-56 尽管图像被“固定”，但仍保持居中

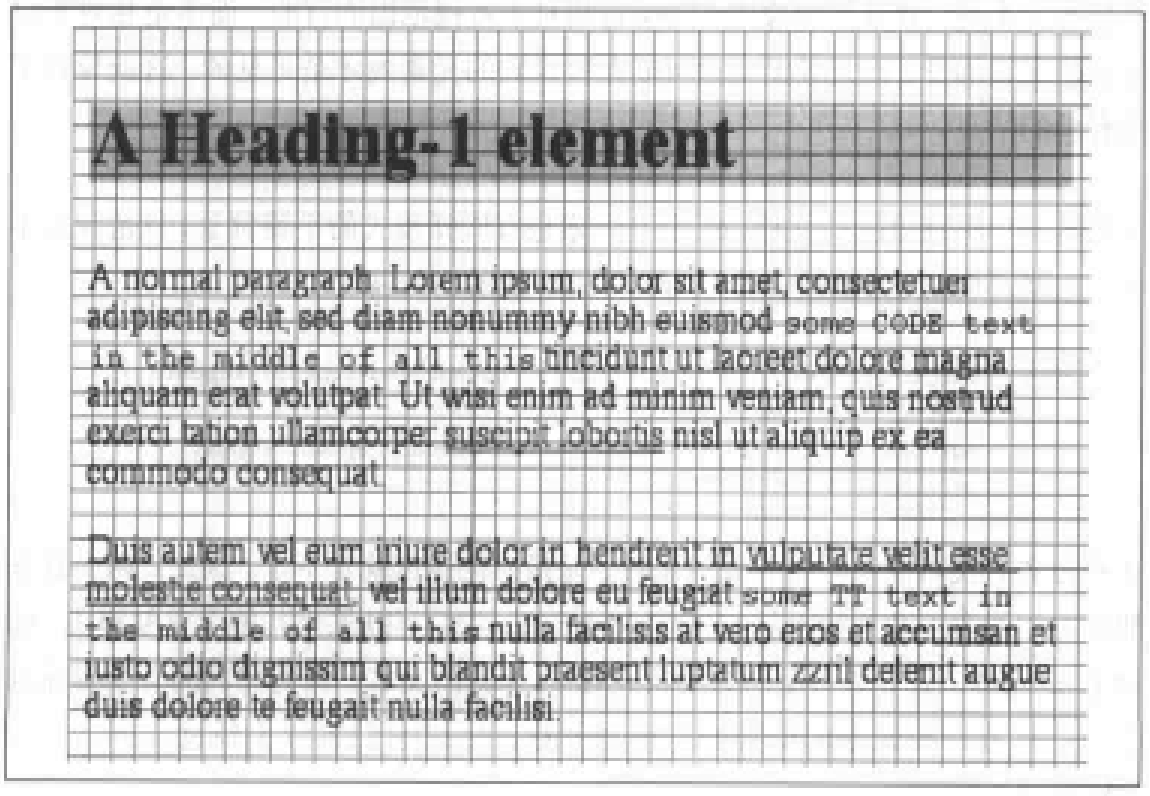


图 6-57 背景的完美对齐方式

警告：有一点需注意的是，在写作本书时，浏览器都不能很好地支持这种固定对齐的方式，因此，这个例子只是一个理想化的实践。

综合起来

类似于字体属性，背景属性也能够综合到一个缩略属性中：background。这一属性能从其他每个背景属性各取一个值，可按任意顺序排列。

background

允许值	<背景颜色>||<背景图像>||<背景重复>||<背景附件>||<背景位置>
初始值	参见各个属性
可否继承	否
适用于	所有元素

注意：在<背景位置>中允许出现百分比值。

因此，下列的各个语句等价，其结果如图 6-58 所示：

```
BODY {background-color: white; background-image: url(yinyang.gif);
   background-position: top left; background-repeat: repeat-y;
   background-attachment: fixed;}
BODY {background: white url(yinyang.gif) top left repeat-y fixed;}
BODY {background: fixed url(yinyang.gif) white top left repeat-y;}
BODY {background: url(yinyang.gif) white repeat-y fixed top left;}
```

实际上，背景属性取值的排列顺序只有一个小小的约束，即当背景位置有两个取值时，它们必须同时出现，水平值在前，垂直值在后。这可能并不奇怪，但一定得记住。

对于这一缩略属性，如果漏掉了某些值，对应属性的缺省值会自动补上。因此，下列两条规则等价：

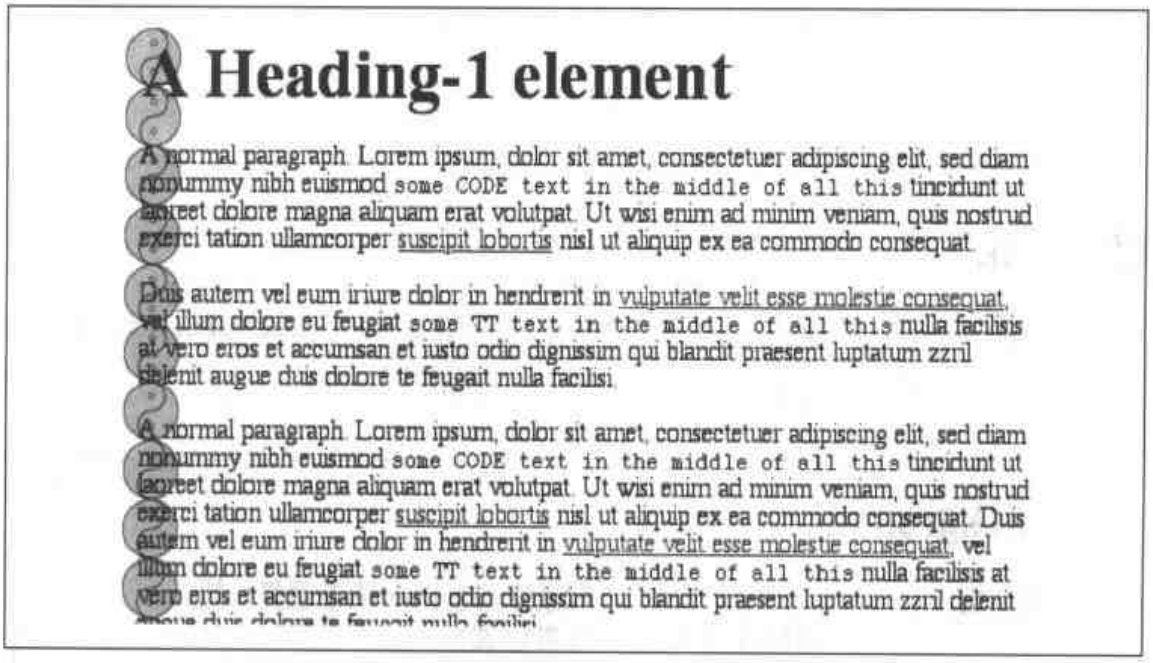


图 6-58 使用缩略属性

```
BODY {background: white url(yinyang.gif;}
BODY {background: white url(yinyang.gif) top left repeat scroll;}
```

而且，background属性没有必须出现的值——只要有一个值，就可以省略其他所有的值。因此，可以用这一简写属性来设置背景颜色：

```
BODY {background: white;}
```

这是合理的，而且显得更为简洁。另外，它还可以设置其他所有背景属性的缺省值，即表示background-image会被设为none。这能阻止其他的规则（例如，样式表）设置背景图像，从而保证可读性。

下面的所有规则都是合理的，如图 6-59 所示：

```
H1 {background: silver;}
H2 {background: url(h2bg.gif);}
P {background: url(parabg.gif);}
P.type1 {background: repeat-x left center;} /* BG from previous rule is lost */
P.type2 {background: center;}  /* same BG loss applies here as well */
```

注意没有图像的两段。这是因为type1段和type2段的样式没有包括背景图像的URL地址。

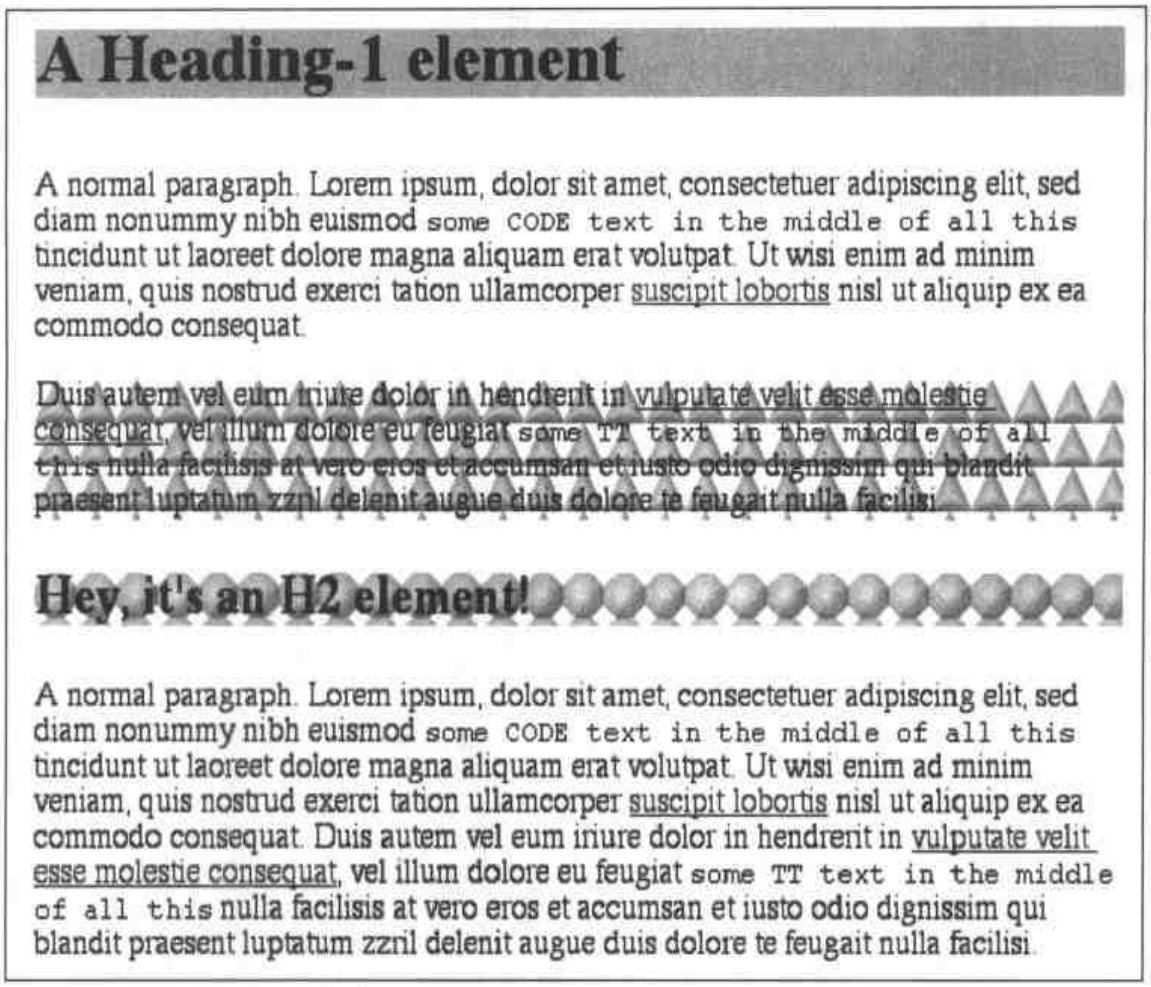


图 6-59 多个背景应用于一个文档

警告：一些老式浏览器，如 Navigator 4 的早些版本对 background 属性支持得较好，但对于一些单个属性的支持却不那么好，如 background-color。所以，应尽量使用 background 属性，避免运用具体的单个属性。

小结

CSS 中颜色和背景的设置给制作者带来了很大的便利。CSS 比传统方法优越就是因为颜色和背景能应用于文档的各个元素——而不仅仅是表格单元，或其他封装在 FONT 标签内的元素。尽管有一些实现中的小问题，像 Navigator 4 不能将背景应用于元素的整个内容区域等，但它们大都得到了广泛的应用。而且由于颜色在页面中最具渲染力，这使得它们的流行程度甚至于超乎我们的想像。

然而，在元素样式方面，CSS还有更多的属性设置：边框能应用于任何元素，包括边界和补白，甚至是“浮动”元素。这些将在下一章中介绍。

第七章

框与边框

大部分网页设计者对作为页面布局语言的HTML都很熟悉，即使他们不很了解它。回想一下那些页面设计，有多少是依赖表格以使页面元素出现在它应该出现的位置？读者可以和大多数网页设计者一样，所有网页均使用表格。这是因为表格可用于生成边栏，可以为整个页面的外观建立复杂的结构，它也可以做些简单的事情，如将文字放入有边框的彩色框中。

表面上看来，用表格来实现后一种效果比较简单。如果需要的只是一个有红色边框、黄色背景的段落，何必在段落周围包上一个单个单元的表格来获得这种效果呢？如果段落自身有边框及背景，忘掉那些表格标记，岂不是容易许多？

令人欣慰的是，CSS的作者也有这个感受，他们投入大量的精力，使得CSS拥有了为几乎所有在网页中出现的元素定义边框的能力。如段落、标题、DIV、定位锚、图像及各种类型的可加边框的元素。这些边框用于将一个元素同其他元素分开，或强调它的外观，或标记某种数据为被改动，以及诸如此类的行为。

除边框外，在元素周围还可以定义区域以控制边框相对于内容如何放置，以及与其他元素可以有多近的距离。在元素内容与边框之间，是元素的补白（*padding*），在边框的上方，是边界（*margin*）。这些属性影响着整个文档的布局，当然更重要的是，它们是影响一个给定元素的外观的重要因素。这是很多CSS样式模型的基础，为了理解它，我们需要看一下元素是如何构成的。

基本元素框

CSS中，所有文档元素都生成一个矩形框，称之为元素框。这个框描述了元素及其属性在文档布局中所占的空间大小，因此每个框都可以影响其他元素的位置及大小。例如，如果文档中第一个元素框为一英寸高，则下一个框将始于比文档顶部低一英寸的地方。如果第一个元素框变换为两英寸高，则其后的元素框变为下沉两英寸，第二个元素框将始于文档顶部低两英寸的地方。

由图7-1可以推知，整篇HTML文档是由大量分布的、不会相互重叠的方框组成。同时，在一定的约束条件下，这些框会占用尽可能少的空间，而且保留足够的空间以辨明内容属于哪个元素。

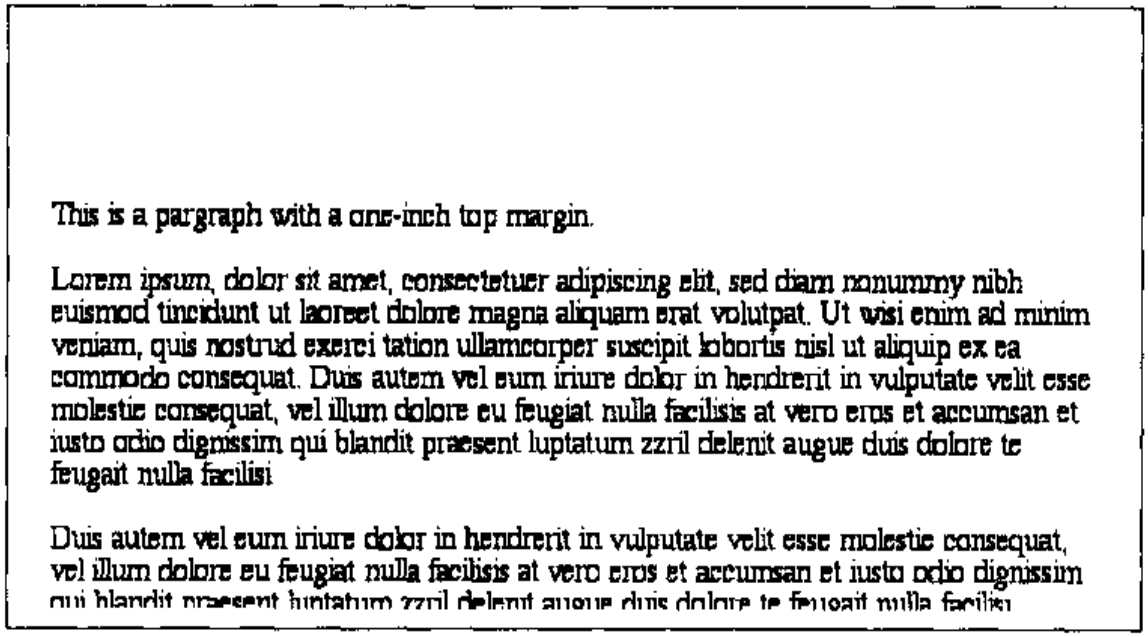


图7-1 一个元素如何影响其他元素

首先，网页制作者可以用基础的方式影响文本元素间的间隔——甚至可以使元素相互重叠！元素框的边界和补白是这个新能力的关键。

为完全理解边界、补白及边框是如何处理的，读者必须清楚地理解大量的边界和区域。它们在图7-2中有详细显示。

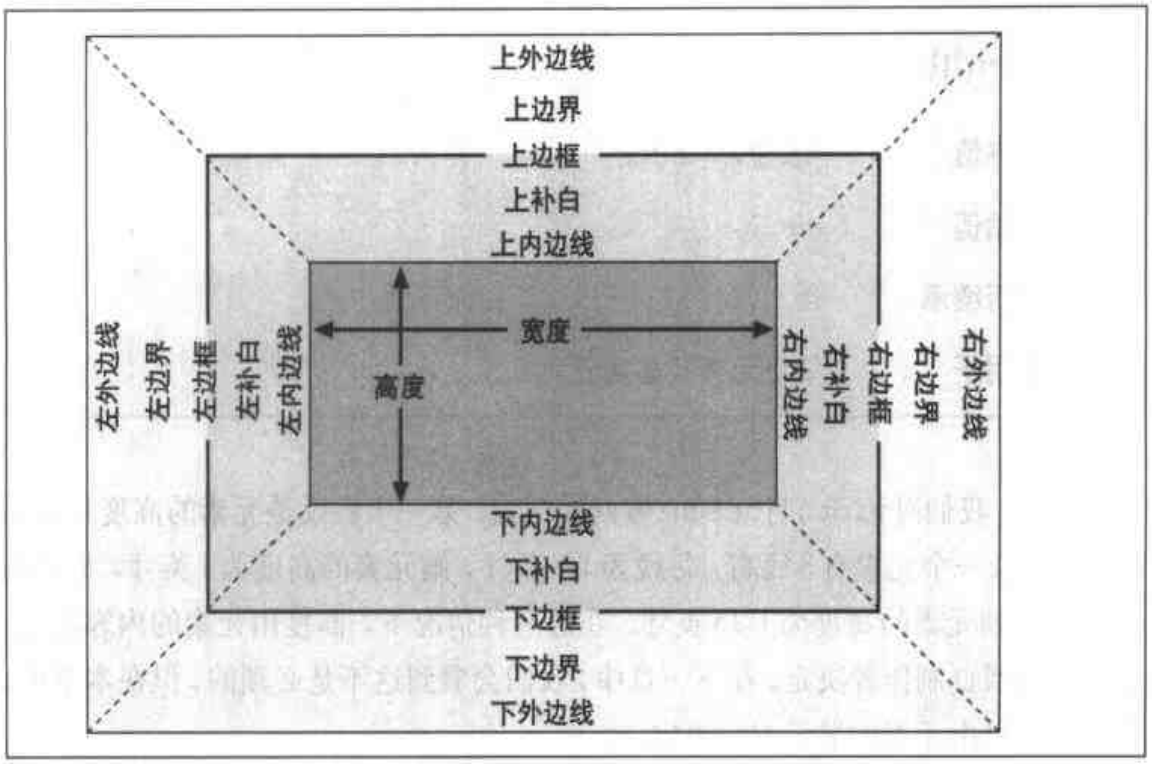


图 7-2　CSS 框模型

通常情况下，一个元素的 width 定义为左侧内部边线到右侧内部边线的距离，height 定义为上内边线到下内边线的距离。它们都是可应用于元素的特性。

width

允许值	<长度> | <百分比> | auto
初始值	auto
可否继承	否
适用于	块级元素和替换元素

注意：百分比是指相对于父元素的宽度。

与 width 相对应的是 height。

height	
允许值	<长度> | auto
初始值	auto
可否继承	否
适用于	块级元素和替换元素

在本章中，我们对width与height做两个假定。第一个假设是元素的高度将自动计算。如果一个元素有8线高，每线为1/8英寸，则元素的高度为1英寸。如果为10线高，则元素的高度为1.25英寸。在这两种情况下，高度由元素的内容决定，而不是由网页制作者决定。在下一章中，我们会看到这不是必须的，但在本章中，假定高度只由元素的显示方式决定。

第二个假定是元素的宽度刚好与其所需的宽度一致。在CSS下，所有元素框与父元素内容区域的宽度一致。这样，如果一个DIV的内容区域为两英寸宽，则那个DIV中任何段落的所有元素框也会是两英寸宽。图7-3详细显示了这一点。

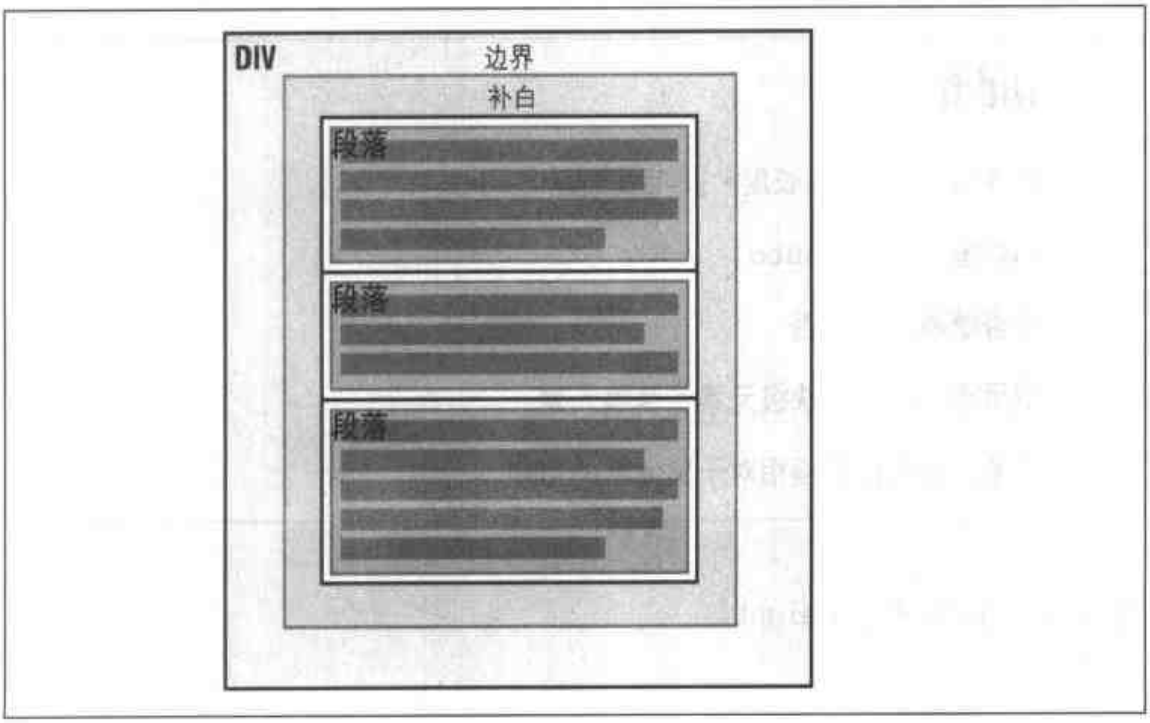


图7-3 元素宽度由父元素的宽度决定

在很多情况下，边界、补白及边框的宽度均为0，此时元素的宽度与父元素内容区域的宽度相同。如果有边界或补白，则它们的宽度将被加到元素的宽度上，以使得与父元素的内容区域宽度一致。在本章的所有例子中，我们都假定元素的宽度是保证同父元素内容区域宽度一致所需的宽度。

是边界，还是补白？

有三种方法在元素周围生成额外的空间。第一种是只对元素增加补白，第二种是只增加边界，第三种是既增加补白，又增加边界。在有些情况下，选择哪一种并不重要，但假如元素使用背景，则与选择有关了。这是因为背景将扩展到补白里，但不会扩展到边界中。这样，对一个元素设定的边界和补白的数量将影响到元素背景的结束位置。

如果为元素设置背景颜色，如图7-4所示，则区别就变得明显一些。有补白的元素有着额外的背景区，而那些有边界的元素则没有。

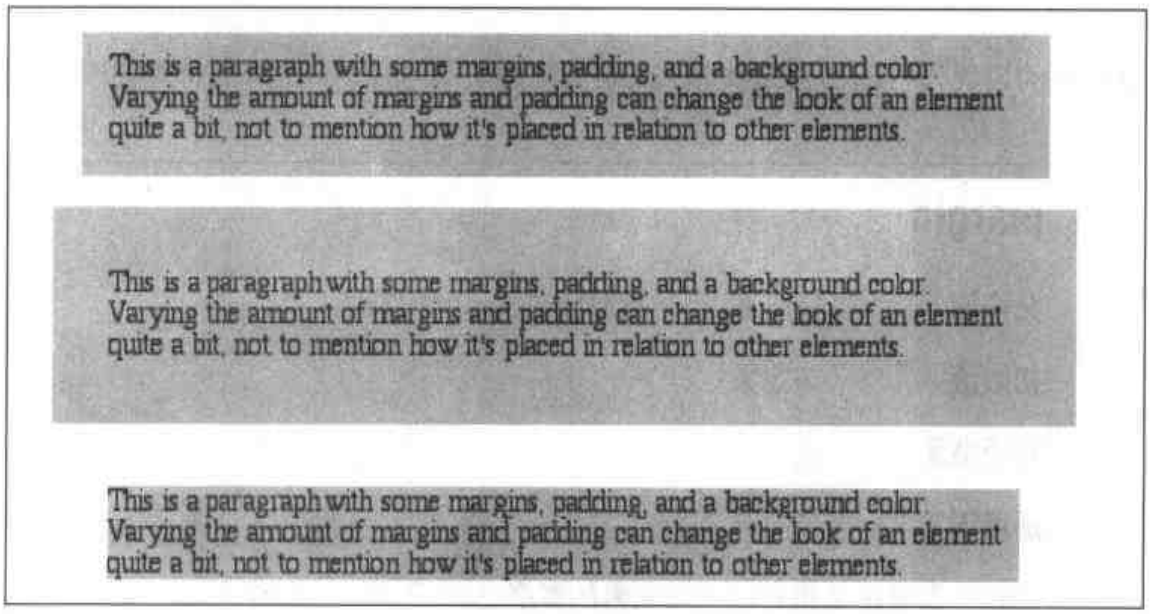


图7-4　用有不同边界和补白的带背景段落来显示差别

最终决定如何设置边界和补白的是设计者，他必须权衡各种可能的效果并做出最好的选择。当然，为了做好这些选择，知道哪些属性可用是有帮助的。

边界

最基本的、可加到元素上的是边界。它们在元素周围生成额外空白区。“空白区”通常是指其他元素不能出现且父元素背景可见的区域。图7-5显示了两个没有边界的段落与有边界的同样段落的差别。

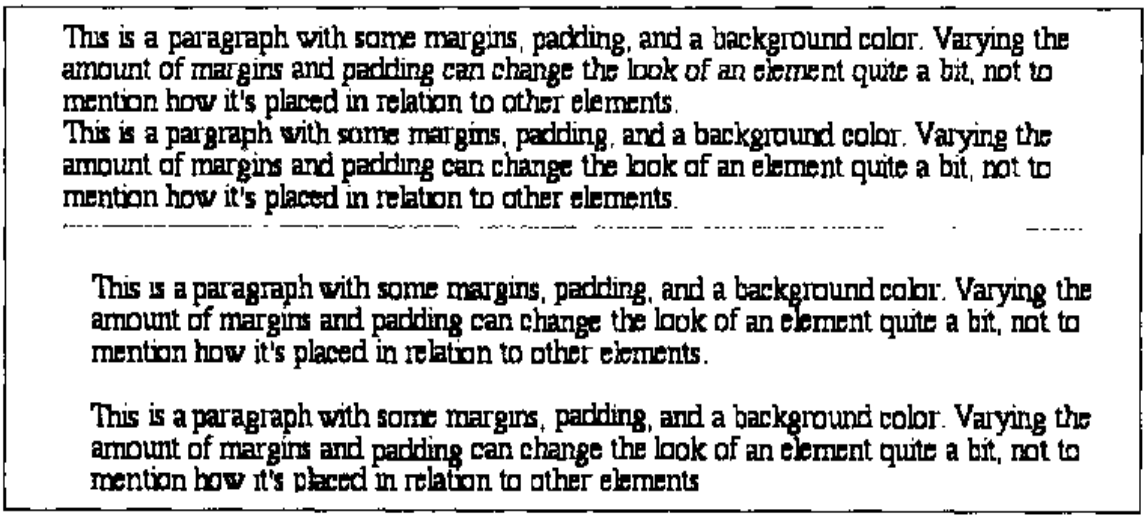


图7-5 有边界及没有边界的段落

最简单的设置边界的方法是使用margin属性。

margin

允许值	[<长度> | <百分比> | auto]{1,4}
初始值	未定义
可否继承	否
适用于	所有元素

注意：百分比是指相对于父元素的宽度。

假定我们要给H1元素设1/4英寸边界，如图7-6所示。（加入背景颜色以显示内容区域的边界）。

```
H1{margin: 0.25in; background-color: silver;}
```

图 7-6　为 H1 标题元素设置边界

这个语句为H1元素的各个边设置了1/4英寸的空白区。在图7-6中用虚线来表示。这条虚线只是用来说明的，在网页浏览器中不会真正出现。

margin可采用任何长度度量单位，如像素、英寸、毫米或 em。margin的缺省值是 0（zero），这意味着如果不声明一个值，那么在缺省时，将没有边界出现。

注意：在实际应用中，浏览器对很多元素都有预指定样式，边界也不例外。例如，在支持CSS的浏览器中，每个段落元素上方与下方的“空行”是由边界生成的。因此，如果没有对P元素定义边界，浏览器自身会使用边界。这说明未定义边界不代表没有边界。

还可以定义边界为 auto。这里，假定 auto 会给出一个自动计算出的值。这个值经常是0，但不总是为0。（幸运的是，auto不为0的环境已被很好地定义，下章将有详细的讨论。）

最后，还可以为 margin 设置百分比。该类型值将在下一节中讨论。

长度值与边界

如前所述，任何长度值均可用于元素边界的设置。如果想在段落元素周围设10像素空白区，非常简单。下列标记可生成一个普通段落及一个带 10 像素边界的段落，如图 7-7 所示：

```
P {background-color:silver;}
P.one {margin:10px;}
```

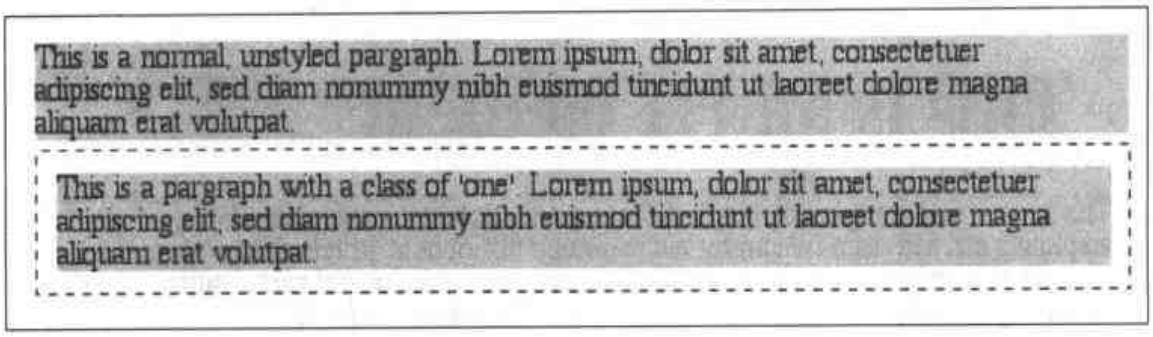


图 7-7 可比较段落

(背景颜色帮助显示内容区，虚线只起说明目的。) 如图 7-7 显示，10 像素空白加到内部部分的每个边。这与在 HTML 中使用 HSPACE 和 VSPACE 有些类似。还可以使用 margin 来为图像设定额外空白区。

假定想给所有图像周围加入 1em 空白:

```
IMG {margin:1em;}
```

就完成了所有的处理。

有时也许需要给元素的每个边加上不同的空白，这也很简单。如果想使所有的 H1 元素上边界为 10 像素，右边界为 20 像素，下边界为 15 像素，左边界为 5 像素，可使用下列标记:

```
H1 {margin:10px 20px 15px 5px;backgroud-color:silver;}
```

如图 7-8 所示，达到了预期的效果。值的顺序很重要，应遵照下列模式:

```
margin: top right bottom left
```

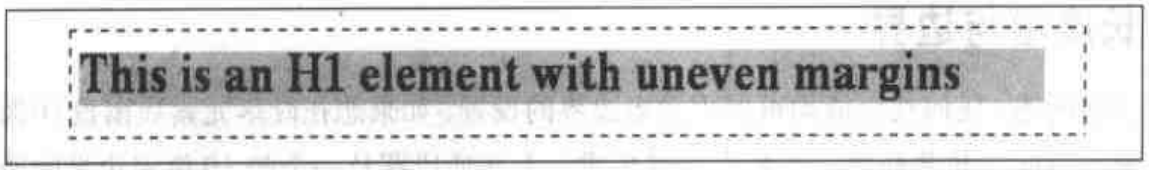


图 7-8 不一致边界

记住这个模式的好的途径是: 这 4 个值从元素顶端开始，顺时针围绕元素。值总是按这个顺序被使用，因此若想得到预期的效果，必须正确地排列它们。

注意：除了想着从顶端开始，按顺时针顺序旋转外，简单地记住边声明顺序的方式是：将各边按正确顺序排列可以帮助避开“trouble”——即TRBL，代表“Top Right Bottom Left”。

也可以混合使用长度类型。并不受限于只能用一种长度类型，如下所示：

```
H2{margin:14px 5em 0.1in 3ex;} /* value variety! */
```

图7-9显示了该声明的结果，图中有一些额外的注解（虚线部分）。

图7-9　混合值边界

百分比与边界

如前所述，可以用百分比来设定元素边界的值。百分比是与父元素的宽度相关的，因此，如果父元素的宽度变了，百分比也会随着发生变化。如图7-10所示：

```
P {margin: 10%;}

<DIV STYLE="width: 200px;">
<P>This paragraph is contained within a DIV which has a width of 200 pixels,
so its margin will be 10% of the width of the paragraph's parent (the DIV).
Given the declared width of 200 pixels, the margin will be 20 pixels on
all sides.</P>
</DIV>
<DIV STYLE="width: 100px;">
<P>This paragraph is contained within a DIV with a width of 100 pixels,
so its margin will still be 10% of the width of the paragraph's parent.
There will, therefore, be half as much margin on this paragraph as that
on the first paragraph.</P></DIV>
```

考虑元素未定义宽度的情况，此时元素的全部宽度（包括边界）依赖于父元素的宽度。

```
P{margin:10%}
```

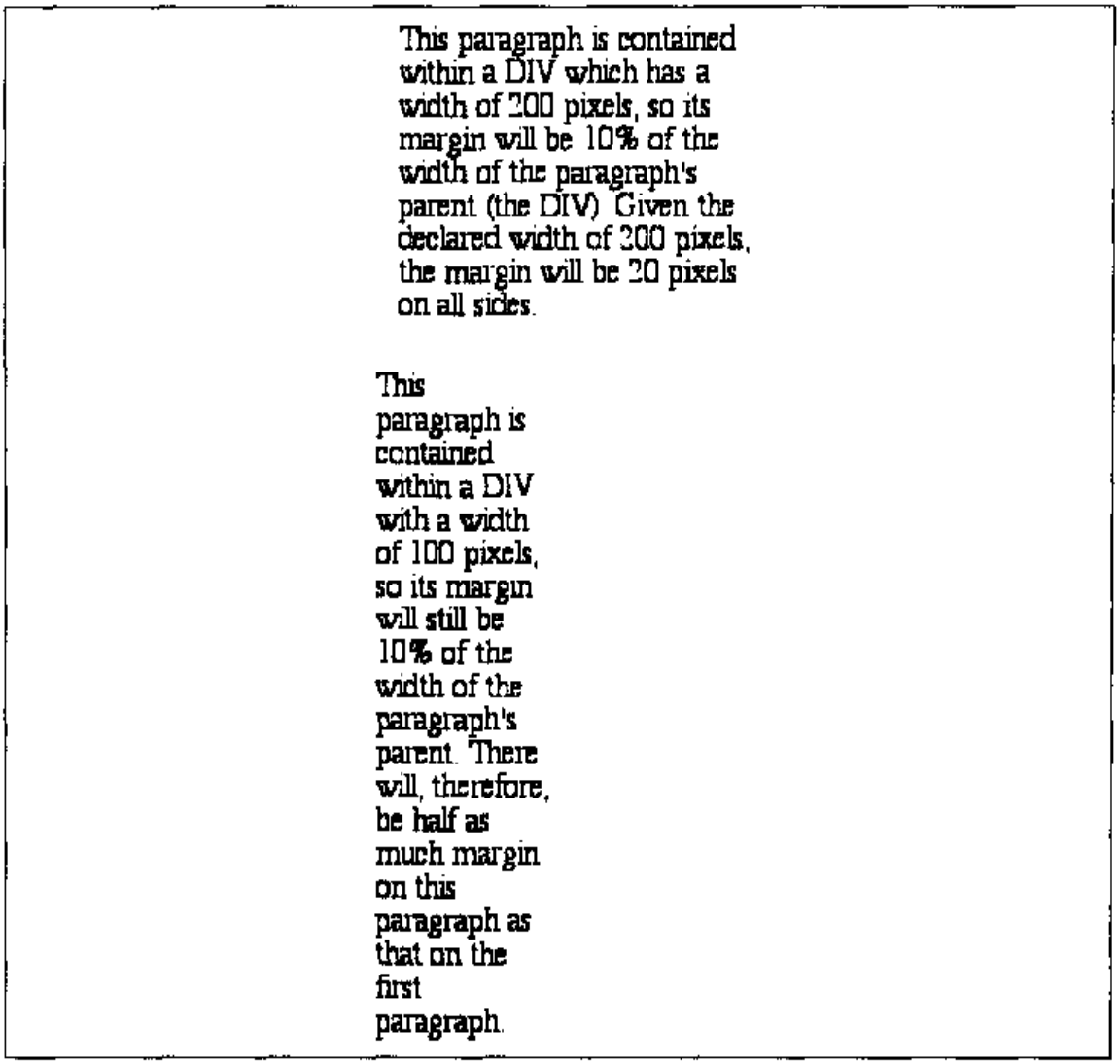


图 7-10 父元素宽度与百分比

图 7-11 为在两个不同大小的浏览器视窗中，段落的边界是如何变化的。

正如可以想像的那样，这将导致“流体”页面的产生，这种页面中元素的补白及边界可以放大或缩小以适应显示画面的实际大小。理论上，当用户改变浏览器视窗的宽度时，补白及边界会动态扩展或缩小，但不是所有的浏览器均支持这种行为。对margin和padding使用百分比是使页面样式能被多种媒体接受的最好方法。例如，文档在显示器上会有与打印效果一致的外观。

也可以组合使用百分比与长度值。为设定 H1 元素有 1/2em 的顶端、底端边界及浏览器视窗宽度 10% 的左右边界，可以用如下声明，图示见 7-12：

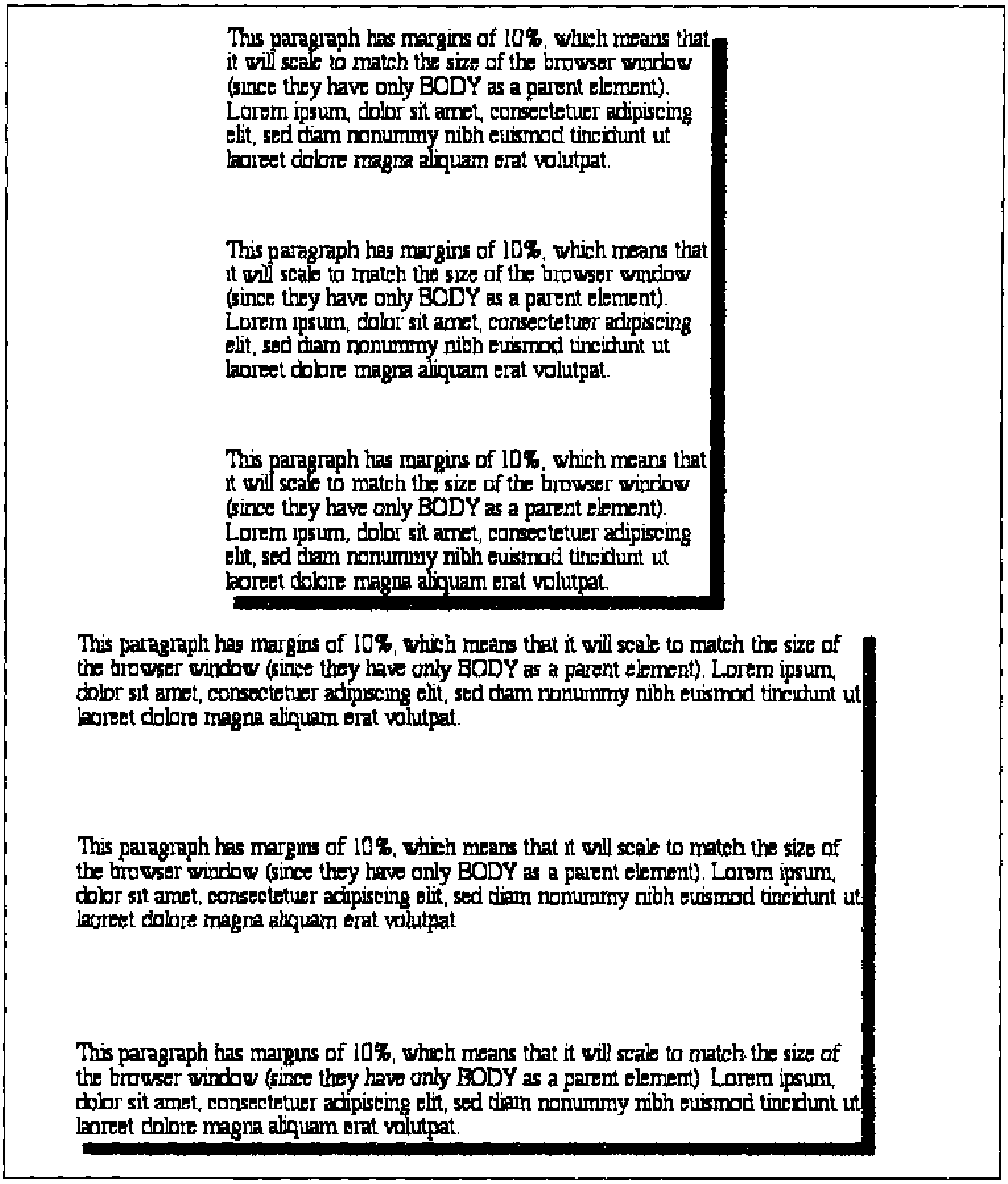


图 7-11　百分比边界及变换的环境

```
H1{margin:0.5em 10% 0.5em 10%;}
```

尽管顶端与底端边界在任何情况下都保持恒定，但左右边界会随浏览器视窗的宽度而变化。当然，这里假定所有 H1 元素都是 BODY 的子元素，且 BODY 元素的宽度为浏览器视窗的宽度。更确切地讲，H1 元素的左右边界宽度是 H1 父元素宽度的 10%。

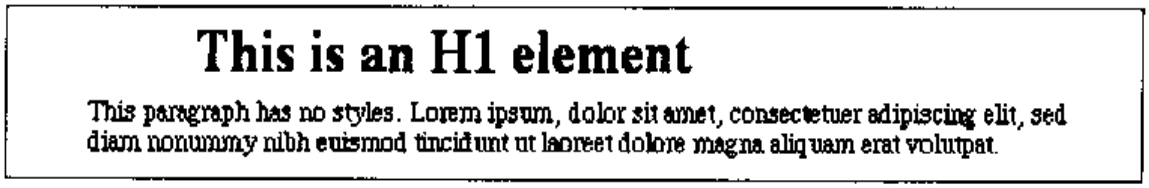


图7-12 混合式边界

让我们回顾一下本例:

```
H1{margin:0.5em 10% 0.5em 10%;}
```

看起来有些多余，不是吗？毕竟要两次输入相同的两个值。CSS提供了简单的方法以避免发生这种情况。

复制值

有时，为margin输入的值有些重复:

```
P{margin:0.25em 1em 0.25em 1em;}
```

不必这样重复输入一对值。可以用下面的标记来代替它:

```
P{margin:0.25em 1em;}
```

这样的两个值足以替代那四个值，是怎么做的呢?

CSS定义了几个步骤以接收少于四个的margin参数。

- 如果没有 *left* 值，则使用 *right* 值代替。
- 如果没有 *bottom* 值，则使用 *top* 值代替。
- 如果没有 *right* 值，则使用 *top* 值代替。

假如更喜欢直观的说明，请看如图7-13所示的图解。

换句话来说，如果给margin赋了三个值，则第四个(*left*)由复制第二个(*right*)得到。如果给出了两个，则第四个由复制第二个得到、第三个(*bottom*)由复制第一个(*top*)得到。最后，如果只给出了一个值，那么它将被复制到其他三个。

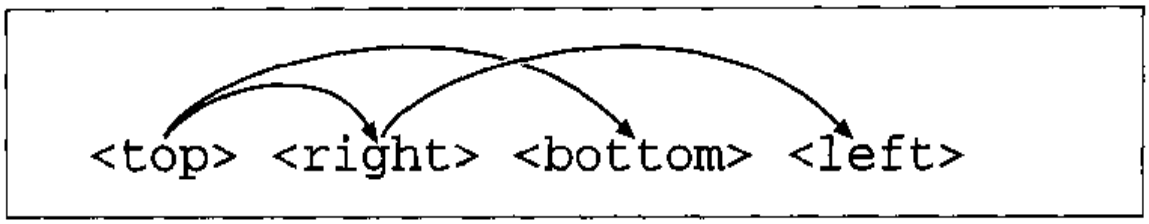


图 7-13 值复制模式

这个简化机制允许制作者只提供必须的值，如下所示：

```
H1{margin:0.25em 0 0.5em;} /* same as '0.25em 0 0.5em 0' */
H2{margin:0.15em 0.2em;}  /* same as '0.15em 0.2em 0.15em 0.2em' */
P{margin:0.5em 10px;}  /* same as '0.5em 10px 0.5em 10px' */
P close{margin:0.1em;}  /* same as '0.1em 0.1em 0.1em 0.1em' */
```

这种机制的缺点确实很小，但用户肯定会遇到它。假定希望为H1元素设置10像素的上边界与左边界，20像素的下边界与右边界。在这种情况下，必须写成：

```
H1{margin: 10px 20px 20px 10px;} /* can't be any shorter */
```

不幸的是，在这种情况下无法减少所需值的个数。

看另一个例子：一个元素，期望除3em的左侧边界外，其他的边界均设为auto：

```
H2{margin: auto auto auto 3em;}
```

我们得到了想要的效果。问题是auto的输入有些烦人。毕竟，我们只是想影响元素一侧的边界，如图7-14所示，它将带我们进入下一个话题。

图 7-14 只为左侧边界设新值

单侧边界属性

非常幸运，有一种方法可以指定元素单侧边界的值。假定我们只想将H2元素的左侧边界设为3em，不必输入margin所需的那些值，可以采用这种方法：

```
H2{margin-left:3em;}
```

margin-left是设置元素框的各侧边界的四属性之一。

margin-top, margin-right, margin-bottom, margin-left

允许值	<长度> | <百分比> | auto
初始值	0
可否继承	否
适用于	所有元素

注意：百分比是指相对于父元素的宽度。

使用这些属性中的每一个都可以只设定该侧的边界，而不会直接影响其他边界。

可以在一个规则中使用多于一个的单侧属性，例如：

```
H2{margin-left:3em; margin-bottom:2em; margin-right:0; margin-top:0;}
```

如图7-15所示，边界的设置与要求的一致。

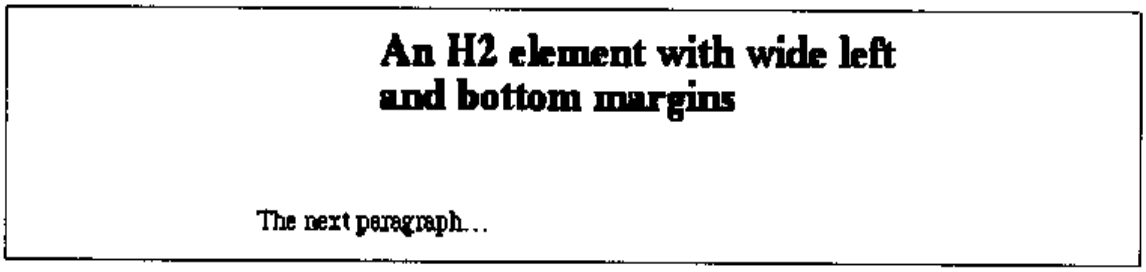


图7-15 多个单侧页空白

然而，在这种情况下，使用margin会更简单些：

```
H2 {margin: 0 0 2em 3em;}
```

结果与刚看到的完全一样，但输入更少了。通常情况下，如果试图设定多侧的边界，简单地使用margin更容易。从文档显示的角度来看，使用什么方法都不重要，所以自由选择容易的方法即可。

压缩边界

在将边界应用于块级元素时，有一个有趣的现象：纵向相邻边界的压缩。这种现象发生在布局中一个有边界的元素紧邻另外一个这样的元素时。

无序列表是一个合适的例子，在无序列表中，列表项一个接一个地排列。假定下面的标记是为一个包含5个列表项的列表声明的：

```
LI {margin-top: 10px; margin bottom: 15px;}
```

这样，每个列表项有10像素的上边界和15像素的下边界。当列表被绘制出来时，相邻列表项间的距离是15像素，而不是25像素。这是因为沿纵轴方向，相邻边界被压缩了。换句话说，两个边界中较小的那个被淘汰了。图7-16显示了压缩与非压缩边界的区别。

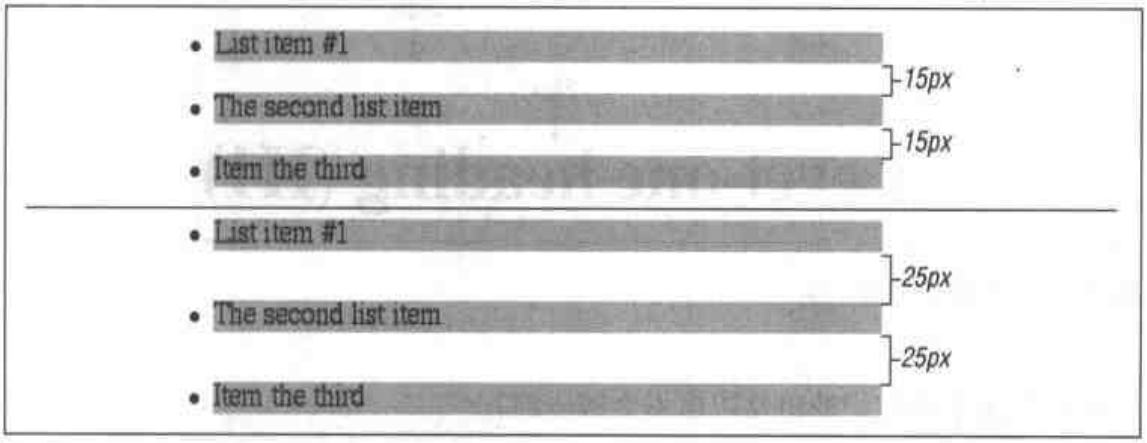


图7-16 压缩边界与非压缩边界的比较

正确实现的客户代理会压缩纵向相邻边界，如图7-16中第一个列表所示。第二个列表显示如果客户代理不压缩边界，会使列表项间有25像素的间隔。

如果你不喜欢“压缩”，可以用另一个词：“重叠”。尽管边界并非真的重叠起来，仍可用下述方法来直观模拟压缩。想像每个元素，如段落，是一小片纸，上面写有元素的内容。在纸的周围，是一定数量的透明塑料，塑料代表边界。第一片纸（H1）放在画布（浏览器视窗）上。第二片纸（段落）放在它的下面，然后滑动直至某一片的塑料边界邻接到另一片的内容边界。如果第一片底端有1/2英寸宽的塑料，第二片顶端有1/3英寸宽的塑料，那么当它们滑到一起时，第一片的塑料

会触到第二片纸的顶端。这样，当两片纸都放到画布上时，附加其上的塑料便会重叠到一起。

当多个边界相遇时，也会发生重叠，例如在一个列表的结尾有如下规则：

```
UL {margin-bottom: 10px;}
LI {margin-top: 10px; margin-bottom: 20px;}
H1 {margin-top: 28px;}
```

列表中的最后一项有20像素的底端边界，UL的底端边界为10像素，紧跟其后的H1有28像素的顶端边界。给定这些以后，如果边界被压缩（重叠），则LI结尾与H1开头之间的距离为28像素，如图7-17所示。

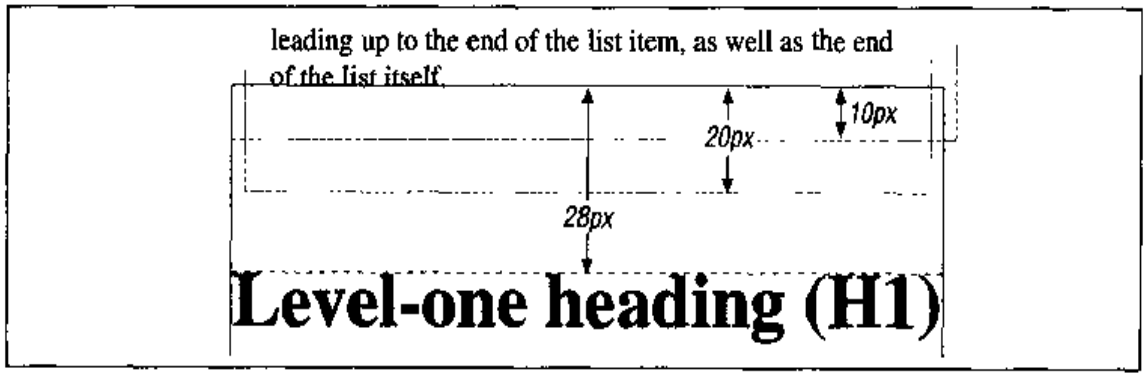


图7-17 压缩的详细说明

压缩只应用于边界，补白及边框不会被压缩。

负边界值

边界还有另外一侧：负侧。可以给边界设负值。假如用户代理完全支持负边界，会出现一些有趣的效果。

注意：根据CSS1规范，用户代理无需完全支持负边界，也说是说，“负数值是允许的，但可能在具体实现中受限”。在网页浏览器中，Navigator 4.x，Explorer 4.x/5.x及Opera 3.x都允许负边界。

负边界对垂直格式编排有影响，影响着边界如何压缩。如果有负的垂直方向的边界，那么浏览器会用最大的正边界的值减去负边界值中的最大绝对值。

如果只有两个边界待压缩，一正一负，则情况的处理会很简单。用正边界的值减去负边界的值的绝对值，或换一种说法，负数加到正数上，结果便是元素间的距离。

为明白这些含义，假定一段落有负的顶端边界，其他侧没有边界（使例子较简单）。另外，我们对段落加粗，以区分于其邻接元素。

```
<P STYLE="margin:-1.75em 0 0 0;font-weight:bold;">
This paragraph has a negative top margin...
</P>
```

在图7-18中可以看到，该段落位置上升了很多，以致于和前一段落的尾部重叠了起来。这就是所期望的效果。

Lorem ipsum, dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

This paragraph has a negative top margin, which will cause it to be "pulled upward," so to speak. Not only will this cause some overlap with the previous paragraph, but it will also pull the rest of the document upward with it.

Lorem ipsum, dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

图7-18 负上边界

用类似的方式，对其他侧也设置负值使它们超出普通的限制：

```
<P STYLE="margin: -2em 0 0 0;font-weight:bold;">...
```

图7-19显示得很清楚，段落溢出了浏览器视窗的边界，而且不仅上升很多，重叠到前一段落的尾部，还将下一个段落拉到与自己最后一行重叠的位置。

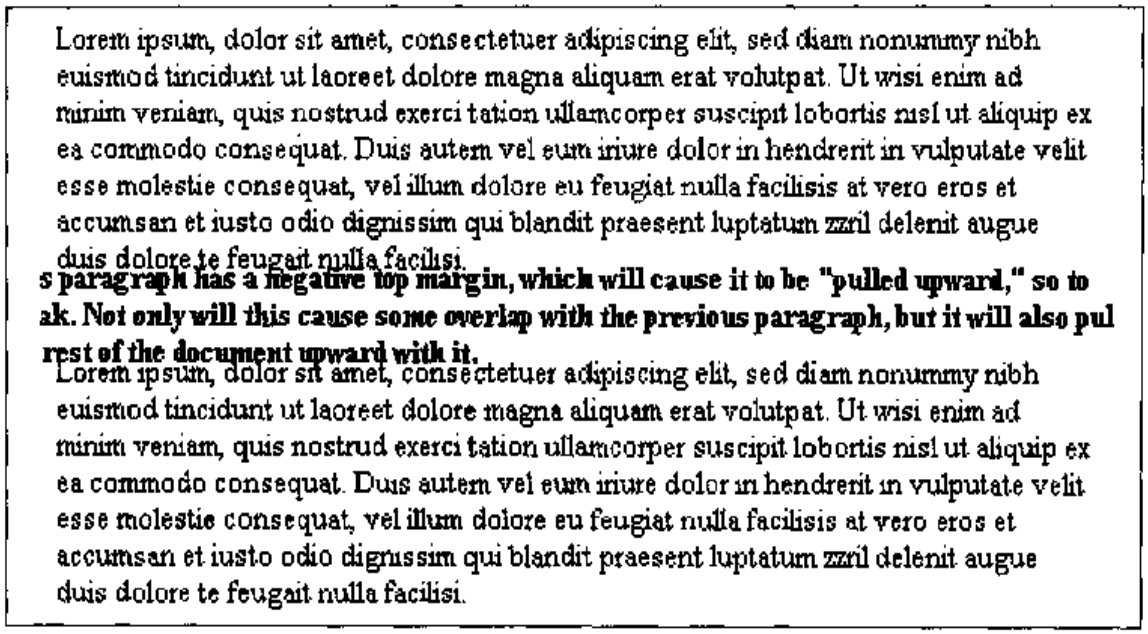


图7-19 负边界

也可以使用负百分比。其行为与负长度值一样，不同的是负数的大小依赖于父元素的宽度，像这样：

```
P {margin: -10%}
```

图7-20显示了该规则的结果，段落间相互重叠的数量及溢出浏览器视窗的程度完全依赖于视窗的宽度，视窗越大，情况越差。

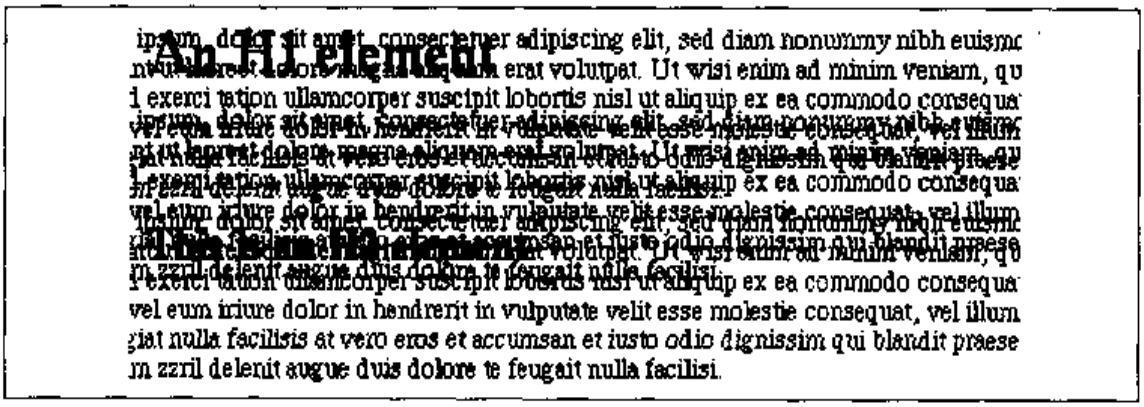


图7-20 文档宽度负边界的危险

将负边界用于块级元素是很危险的——但它也是有价值的。进行大量的练习，犯许多错误才能学会区分这两者。

边界及行内元素

迄今为止，我们只讨论了边界在块级元素上的应用，如段落、标题。边界也可以应用于内联元素，当然效果会有一些不同。

假定想给加粗文本设置顶端、底端边界，声明为：

```
B {margin-top: 25px; margin-bottom: 50px;}
```

这是规范所允许的，但显然它对行的高度没有影响，且由于边界效果是透明的，没有任何的视觉影响，如图 7-21 所示。

This paragraph contains a small amount of **boldfaced text** to which a margin has been applied.

图 7-21　边界应用于内联元素

这是因为边界应用于内联元素时不改变元素的行高度。(能够改变只含文本的行间距的属性只有 line-height，font-size，vertical-align。)

然而，这只是对内联元素的顶端和底端而言的，左侧和右侧则不同。考虑最简单的情况，一个位于单独行的小的内联元素，如图 7-22 所示。

This paragraph contains a small amount of **boldfaced text** to which a margin has been applied.

图 7-22　有左侧边界的单行内联元素

如果为左侧或右侧边界设值，则它们的效果可见，如图 7-23 所示：

```
B {margin-left:10px;backgroud:silver;}
```

This paragraph contains a small amount of **boldfaced text** to which a margin has been applied.

图 7-23　有左侧边界的内联元素

注意内联元素前面一个词的结尾与其背景边界之间的额外空间。如果希望的话，我们可以在内联元素的两端留出空白：

```
B {margin-left: 10px; backgroud: silver;}
```

如同所期望的那样，图7-24显示出内联元素的左侧与右侧均有额外空白，顶端及底端没有额外空白。

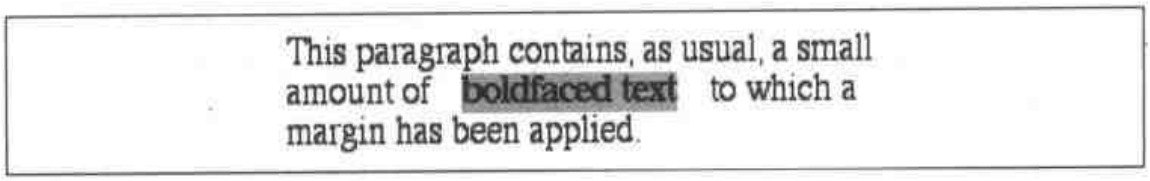


图7-24 有10像素边界的内联元素

这些看起来都很简单，但当加粗文本伸展到多行时，情况会变得有些奇特。首先，为内联元素设定的边界不能应用于行间断发生的位置。这些行间断是为了适合浏览器视窗或适应父元素。边界对行间断唯一的影响是在行内产生额外空白页以移动内容。这使得行在一个与通常不一样的地点中断。

图7-25为一个使用边界的内联元素多行显示时的情形：

```
B {margin: 10px; backgroud: silver;}
```

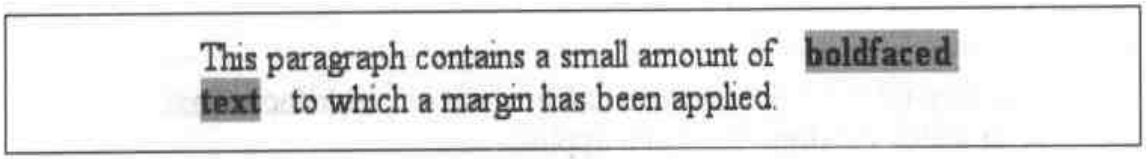


图7-25 有10像素边界的跨行文本内联元素

左侧边界应用于元素的开头，右侧边界应用于元素的结尾，边界没有应用在顶端与底端。而且，如果没有使用边界，则这一行会在“text”后而不是在“boldfaced”后面中断。这是边界影响行间断的唯一方式。

为了理解其原因，让我们回到上一节中的纸片与塑料的模拟。把内联元素考虑成一长条形纸片，旁边由塑料包围。跨行显示内联元素就如同把这一条切成较小的几条。然而，没有额外塑料附加到较小的纸条上。被使用的塑料只在纸片的两端，所以只出现在内联元素的首部与尾部。

边界：已发现的问题

尽管边界很有用，但使用它仍会带来很多问题。实际上，多到有理由单独列出一节来介绍，而不是只有一个小的警告框。

第一个是Navigator 4.x一般增加边界规则到它的内部边界，而不是取代其内部值。例如，希望消除H1元素与段落之间的空间，这是最简单的做法：

```
H1 {margin-bottom:0;}
P {margin-top:0;}
```

这是一种正确消除相邻元素间空间的方式。然而，Navigator 4.x会在显示时在元素间增加一个空行，如图7-26所示。这是因为它把0值加到自己的缺省边界上。

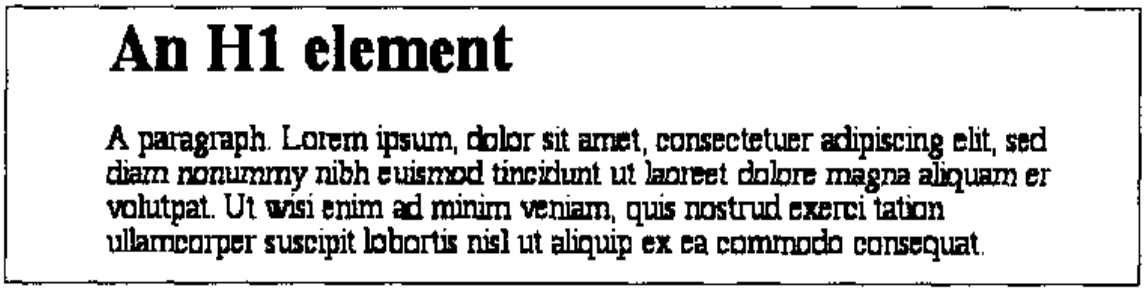


图7-26 Navigator 4.x和边界

如果想去掉这空间，可以使用负边界。这是一种可行的声明：

```
H1 {margin-bottom:0;}
P {margin-top:-1em;}
```

这种解决办法也会导致问题发生，当文档用Internet Explorer浏览时，显示如图7-27所示。文本重叠不是Explorer方面的错误，它能正确地按声明执行。基本上，没有简单的办法可以回避这个问题，有两种方法将在第十一章详细介绍。

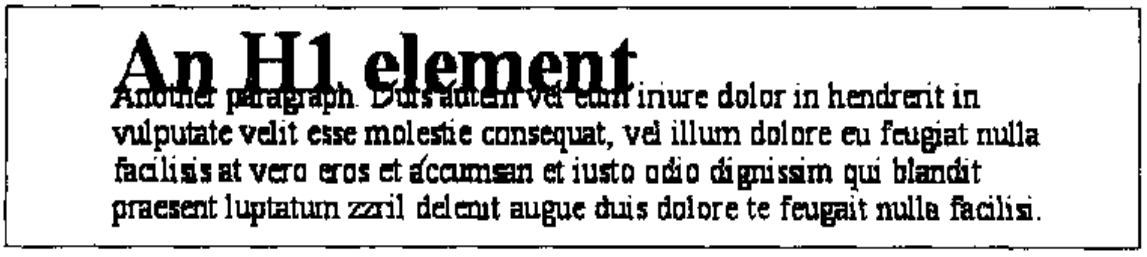


图7-27 在Explorer中文本重叠

更糟糕的是，如果将边界应用于内联元素，如前面讨论的那样，会在Navigator 4.x得到如图7-28所示的结果。

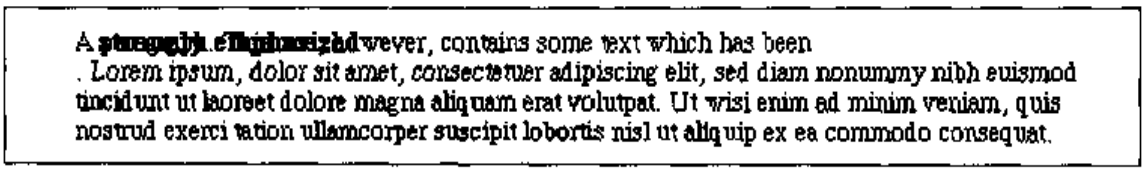


图7-28 边界、内联元素及Navigator4.x

用于生成图7-28的样式是：

```
STRONG  {margin-left: 10px;}
```

Navigator不是在STRONG元素的开始部分添加10个像素的空白空间，而是假定边界是参考浏览器视窗的左边沿，并把STRONG元素放置在相应的位置。这是彻底的、完全的错误。（有人推测Navigator将内联元素变换为块级元素，但它的布局却隐含着事情不是如此。所以这个问题很难确定。）事实证明，应用边界于内联元素是一个冒险的主张，不能轻率采纳。

边框

元素的边框（border）就是一条（有时是多条）围绕着元素内容及补白的线。元素的背景会结束于边框的外边沿，因为背景不延伸到边界，而边框恰恰位于边界的内部。CSS规范强烈暗示背景延伸到边框的外边界，因为它在谈及边框时，是边框画在“元素背景的顶部”，但看来不是所有浏览器都同意这一点。这一点很重要，因为有的边框是“间歇”的，如点与破折号式样，元素的背景应该在边框的可见部分之间出现。

每个边框都有三个特征：宽度或粗度，式样或外观，以及颜色。边框宽度的缺省值是medium，medium没有显示定义，通常为两个或三个像素。通常不会看到边框的原因是因其缺省样式为none，阻止了它们的出现。如果一个边框没有样式，那么它可以不出现。缺少边框式样也会重设置宽度，我们将稍后讨论这个问题。

最后，边框的缺省颜色是元素本身的前景色。如果没有声明边框颜色，那么，它

的颜色将与元素的文本颜色相同。另一方面，如果一个元素没有文本，例如一个只包含图像的表格，颜色是由继承得到的。表格边框的颜色是其父元素的文本颜色。其父元素可能是BODY、DIV或另一个TABLE。这样，如果一个图像有边框，且BODY是其父元素，给出如下规则：

```
BODY {color: purple;}
```

这样，缺省情况下，围绕图像的边框颜色为purple。当然，为了使边框显示出来，还必须先做一些工作。

带样式的边框

这里我们先讨论边框的样式，因为它是边框中最重要的部分。它之所以最重要不是由于它控制着边框的外观，尽管它确实控制，而是因为如果没有样式，根本就不会有边框。

border-style

允许值	none | dotted | dashed | solid | double | groove | ridge | inset | outset
初始值	none
可否继承	否
适用于	所有元素

注意：对CSS1来说，只有支持solid是必需的。

CSS1定义了九种不同样式的border-style属性，包括缺省值none。如图7-29所示。

注意： 最有趣的边框样式是double。它定义中的两条线的宽度加上两条线之间的空间宽度，与border-width的值（将在下一节中讨论）相同。然而，规范中并没有规定一条线是不是应该比另一条粗，还是应该有相同的宽度，两线间的空间应该比线粗还是细。所有这些都留给用户代理去决定。

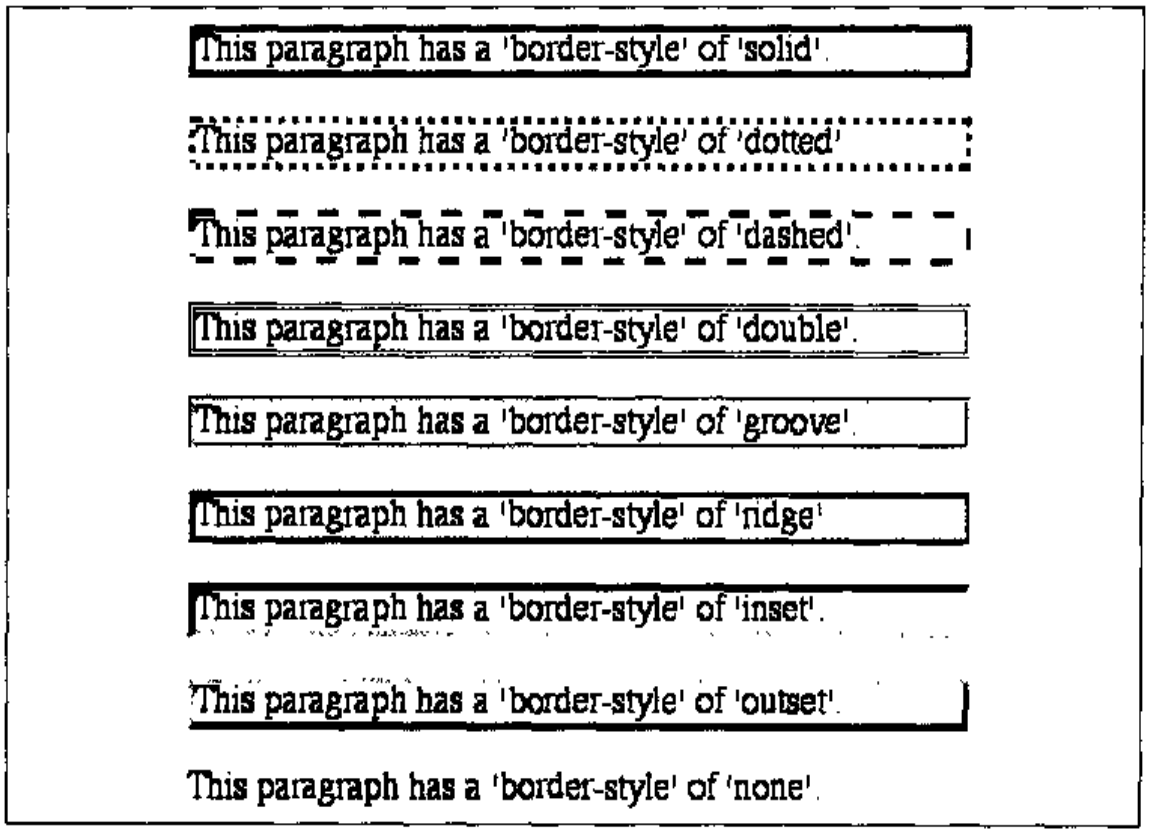


图 7-29 边框样式

图 7-29 中所有边框的颜色都是 gray，这样可以使总体效果更易见。边框样式的外观在一定程度上依赖于边框的颜色，尽管不同的用户代理会有不同的确切的策略。图 7-30 显示了绘制 inset 边框的两种不同方式。

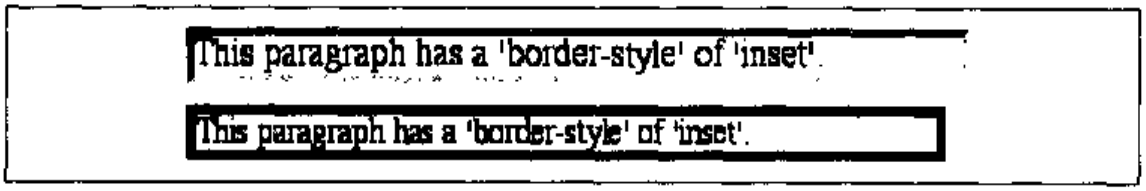


图 7-30 两种有效的绘制 inset 的方式

假定想为超链接中的图像定义一种边框样式，可以将它们设为 outset，这样图像就有一种“凸出按钮”的外观，如图 7-31 描述的那样：

```
A:link IMG {border-style: outset;}
```

边框的颜色基于元素的 color 值，在这个环境下为蓝色（尽管无法在打印中显示）。这是因为图像包含于超链接中，超链接前景色通常为蓝色。如果希望将颜色变为 silver，可以通过下面的标记来实现这变化：

图 7-31　对超链接图像使用 outset 边框

```
A:link IMG {border-style: outset; color: silver;}
```

如图 7-32 所示，边框颜色成为浅银灰色，即图像现在的前景色。尽管图像实际上不使用这个颜色，它还是被传送给边框。在以后的章节中，我们还会谈及另一种改变边框颜色的方法。

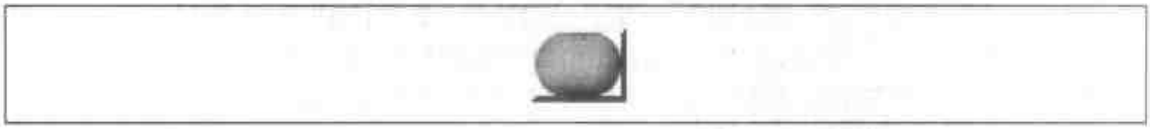


图 7-32　改变边框的颜色

多种样式

可以为给定边框定义多种样式。例如：

```
P.aside {border style: solid dashed dotted solid;}
```

结果如图 7-33 所示，是一个有实心线的上边框、长划线右侧边框、点状线下边框、实心左侧边框的段落。

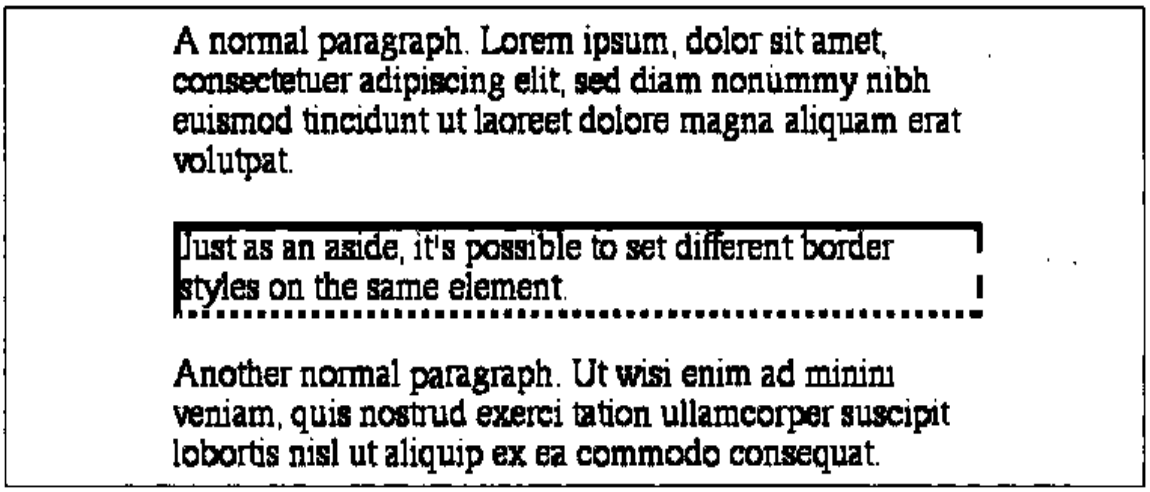


图 7-33　在一个元素上使用多种边框样式

又一次出现按top-right-bottom-left顺序排列的值。这和用多个值设定不同边界的能力类似。关于值复制的规则也应用于边框式样，如同它们作用于边界与补白。这样，下面的两个声明有相同的效果，如图7-34描绘的那样：

```
P.new1 {border-style: solid dashed none;}
P.new2 {border-style: solid dashed none solid;}
```

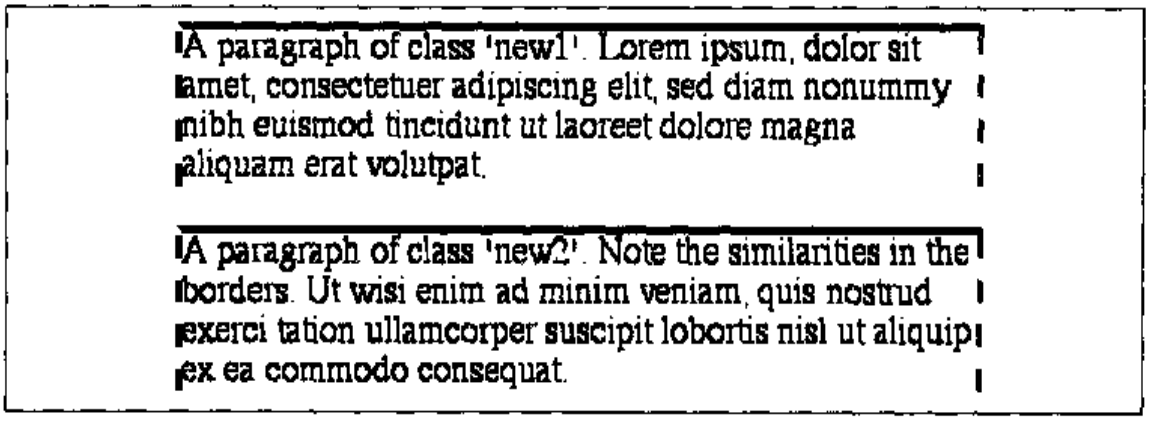


图7-34 等同的样式规则

也许读者很好奇，在CSS1中，为何没有办法用类似于border-top-style的方式直接为元素的一侧设定边框样式，因为在CSS1中没有这样的属性（尽管该属性及其他类似的属性在CSS2中都引入了）。当然，仍可以绕过这个限制，使用本章稍后讨论的缩略属性为给定边框声明样式。

回退到实心

关于CSS，有一个有趣的东西，它足以使制作者的工作变得艰难起来。在CSS1中，用户代理可以将border-style的任何值（包括none）解释为solid。由于该许可的存在，一个技术上与CSS1兼容的用户代理可能会将下面的边框全部显示为实心：

```
P.new3 {border-style: ridge dashed double;}
```

图7-35显示的结果肯定与制作者的构想不同，但在技术上是正确的。只要支持none和solid，且其他任何合法值都被解释为solid，就足以与CSS1兼容。于是，即便Navigator 4.x不能绘制dashed和dotted边框，而把边框按solid绘制了，它也没有什么可被指责的。

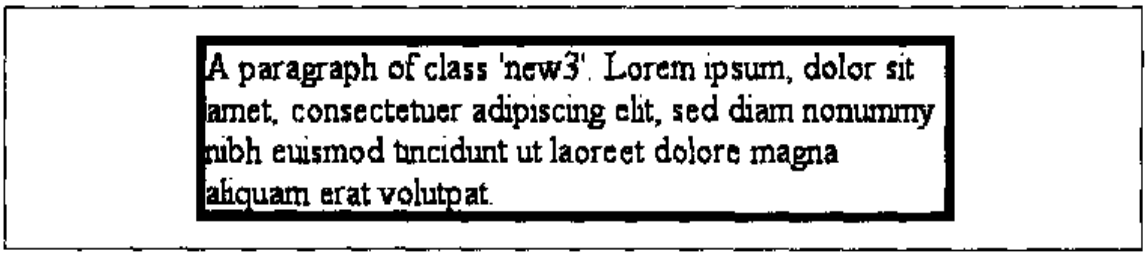


图7-35 用实心代换不支持的边框样式

读者可能注意到了，本节所有例子中边框的宽度都完全一样。这是因为未定义宽度，因此缺省为一个固定的值。接下来，我们要获知这个缺省值及其他更多的内容。

边框宽度

一旦指定了一个样式，定制边框的下一步就是给它一定的宽度。这个步骤使用border-width属性。也可以使用下列同等属性之一：border-top-width，border-right-width，border-bottom-width和border-left-width。

每个属性用于设定一特定侧的边框宽度，如同边界属性那样。

border-width

允许值	[thin | medium | thick | <长度>]{1,4}
初始值	未定义缩略属性
可否继承	否
适用于	所有元素

共有四种指定边框宽度的方法：赋一长度值如4px或0.1em，或使用三个关键字之一。这些关键字是thin，medium（缺省值）和thick。这些关键字不必与某特定的宽度对应，只是简单地定义了相互间的关系。根据规范，thick总比medium宽，medium总比thin宽。

border-top-width, border-right-width, border-bottom-width, border-left-width

允许值	thin | medium | thick | <长度>
初始值	medium
可否继承	否
适用于	所有元素

然而，由于未定义确切的宽度，所以一个用户代理可以将它们设为5px、3px、2px，而另一个用户代理可将它们设为3px、2px、1px。不论用户代理用什么宽度来代表关键字，在任何环境下，它在全文档中是一致的。如果medium等同于2px，则一个中等宽度边框总是两个像素宽，不论边框内的是H1元素还是P元素。图7-36描述了处理这三个关键字的方法，它们相互间的关系以及它们与所包围的内容的关系。

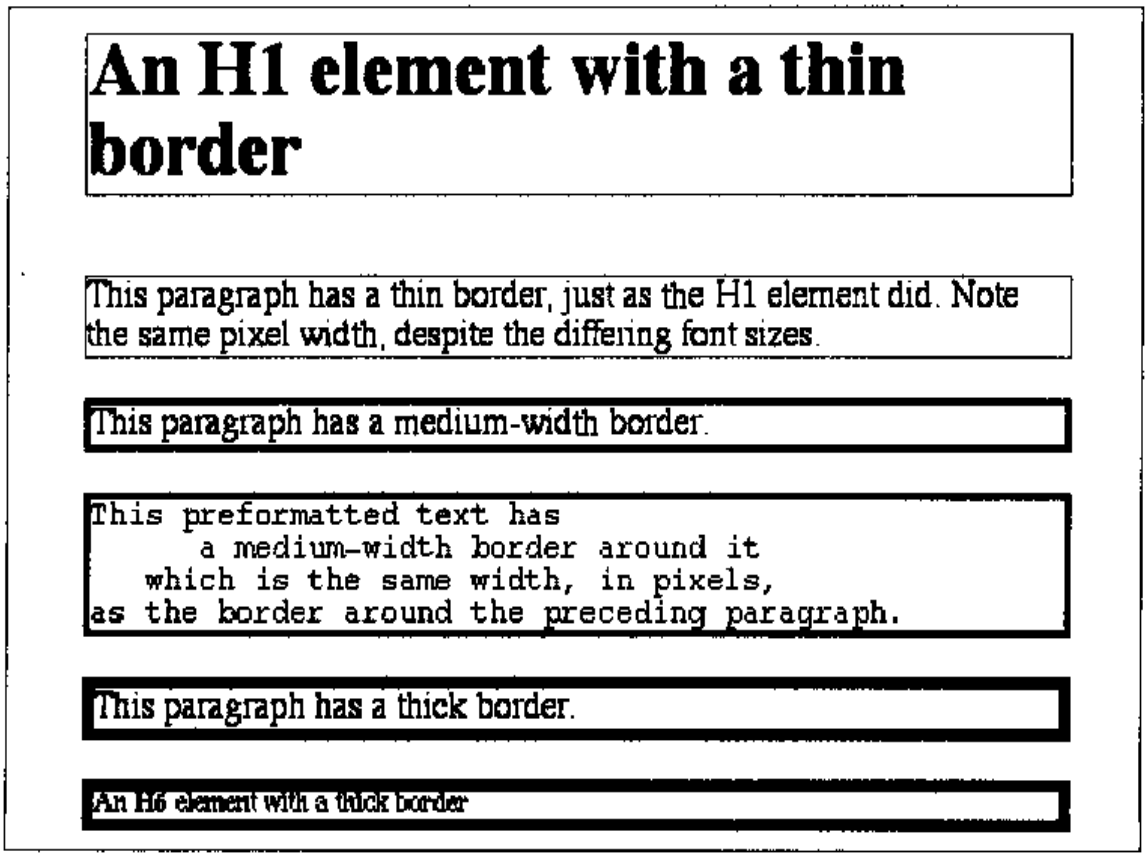


图7-36 边框宽度关键字的相互关系

假定一个段落有边界、背景颜色、边框样式，如图 7-37 所示：

```
P {margin: 5px; backgroud-color: silver; border-style: solid;}
```

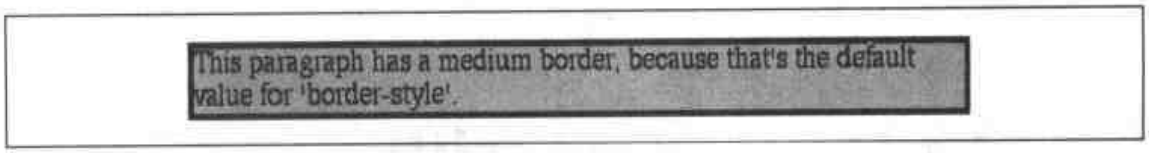


图 7-37　边界、背景颜色及边框

边框宽度缺省情况下为 medium，如图 7-37 所示。我们可以改变它，改变后的结果见图 7-38：

```
P {margin: 5px; background-color: silver;
  border-style: solid: border-width: thick;}
```

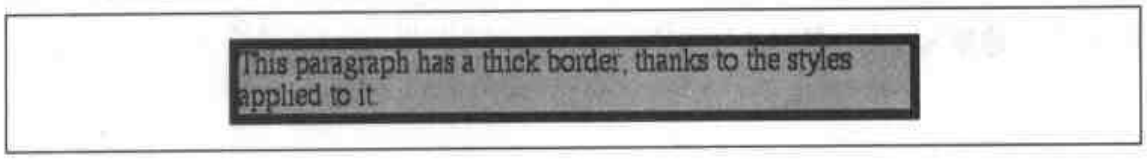


图 7-38　改变边框宽度

改变边框宽度可以使边框变得相当荒唐与框端，比如设置 20 像素的边框，如图 7-39 所绘：

```
P {margin: 5px; background-color: silver;
  border-style: solid: border-width: 20px;}
```

所期望的是：样式与宽度联合生成一个颜色依赖于元素前景色的边框。

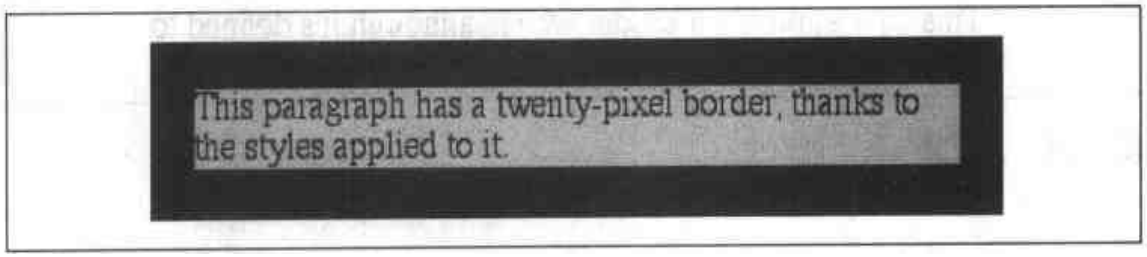


图 7-39　将边框宽度膨胀到不合理的级别

也可以为单独的一侧设定宽度。可用两种熟悉的方法来完成，第一种方法是使用

本节前面所讲述的特殊属性，如border-bottom-width。另一种方法是在border-width中使用值复制。这些在图7-40中有解释。

```
H1 {border-style: none none dotted; border-bottom-width: thin;}
P {border-style: solid; border-width: 15px 2px 7px 4px;}
```

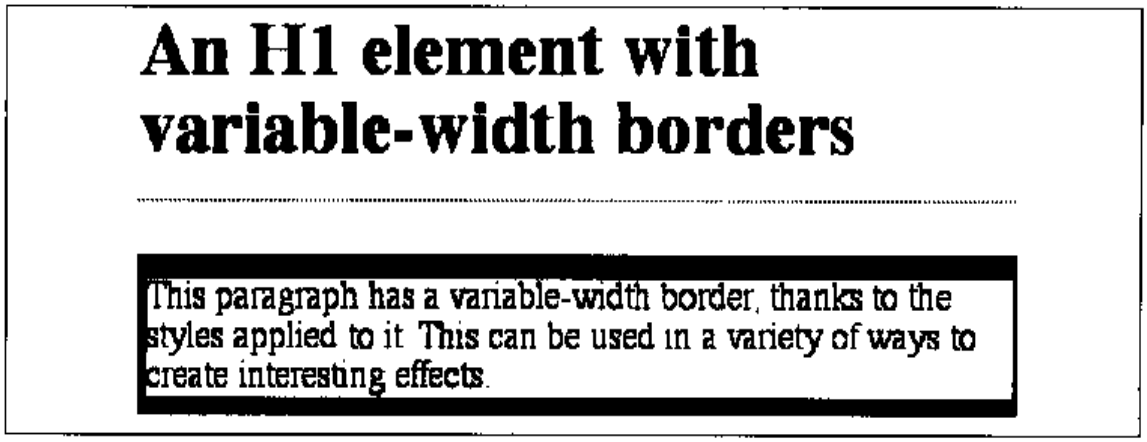


图7-40 值复制及不一致的边框宽度

无边框

到这里，我们还只是讨论了使用可见边框样式的情形，如solid或outset。当边框式样设为none时，事情变得有趣起来：

```
P {margin: 5px; border-style: none; border-width: 20px;}
```

如图7-41所示，尽管边框宽度被设为20px，当样式被设为none时，不仅边框样式没有了，边框宽度也消失了，为什么呢？

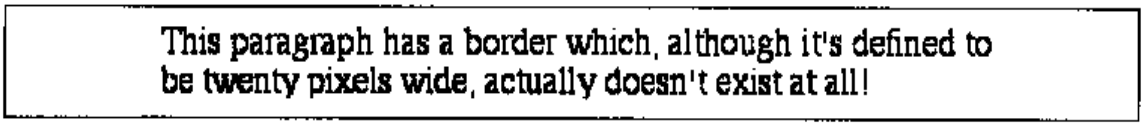


图7-41 难以置信的边框消失

如果还记得，前面节中使用的术语是：样式为none的边框不存在。那些词都是仔细挑选的，因为它们可以解释这里的事件。由于边框不存在，那么它就不能有宽度，因此宽度自动设为0（zero）。这看起来不好理解，但实际上很有意义。毕竟，如果一个水杯是空的，则不能将它描述为有半杯“nothing”，只能在杯中有一些

内容时才能讨论内容的深度。同样，只有在边框存在的情况下讨论边框宽度才有意义。

这一点要牢牢记住，因为常见的错误是忘记定义边框样式。这是一个第一眼看上去正确，但结果是一个无边框段落的声明，它甚至能难倒各类制作者：

```
P {margin: 5px; border-width: 20px;}
```

由于border-style的缺省值是none，声明边框失败等同于声明border-style为none。因此，若想一个边框出现，必须挑选一个样式并声明它。

边框颜色

同其他边框特征相比，设定颜色十分容易。在CSS1中，只有一个border-color属性，可以一次接收最多四个颜色值。

border-color

允许值	<颜色>{1,4}
初始值	元素的颜色
可否继承	否
适用于	所有元素

如果少于四个值，则需进行值复制。这样，如果想使H1元素有细、黑色顶端、底端边框和粗、灰色两侧边框，P元素有中等粗度、灰色边框，这些可充分达到要求，如图7-42所示：

```
H1 { border-style: solid; border-width: thin thick; border color: black gray;}
P {border-style: medium; border-color: gray;}
```

缺省状况下，只有一个颜色值时，该颜色应用于所有四个边上，如上例中的段落。另一方面，如果提供四个颜色值，则每个边都可以得到不同的颜色。任何类型的颜色值均可使用，包括命名的颜色、十六进制值及RGB值。

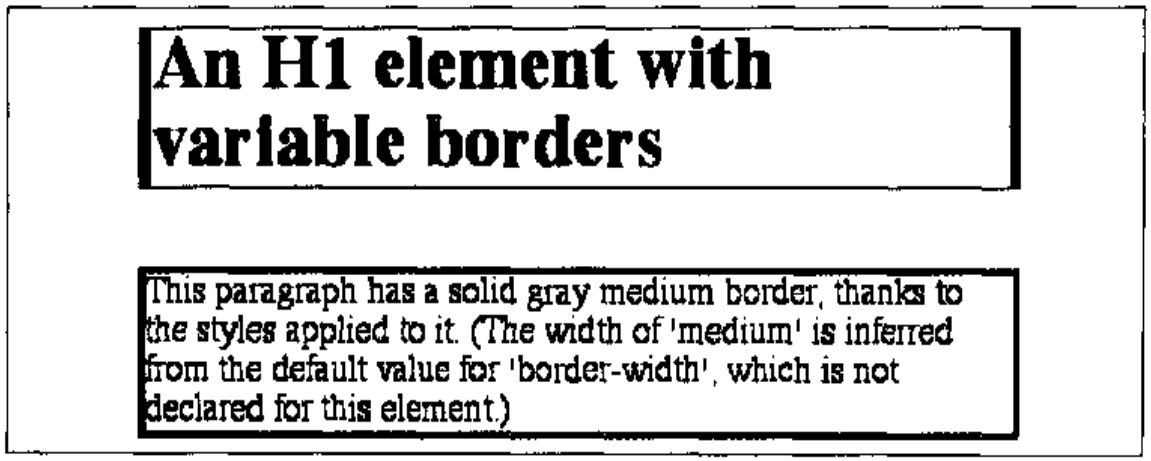


图 7-42 有多个特征的边框

```
P {border-style: solid; border-width: thick;
  border-color: black rgb(25% 25% 25%) #808080 silver;}
```

图 7-43 显示边框的不同灰色阴影。由于本书的印刷只支持灰度，所以例子中均用灰色阴影，其实各种颜色均可使用。如果希望 H1 元素有红、绿、蓝、黄边框，十分容易做到：

```
H1{border-style: solid; border-color: red green blue yellow;}
```

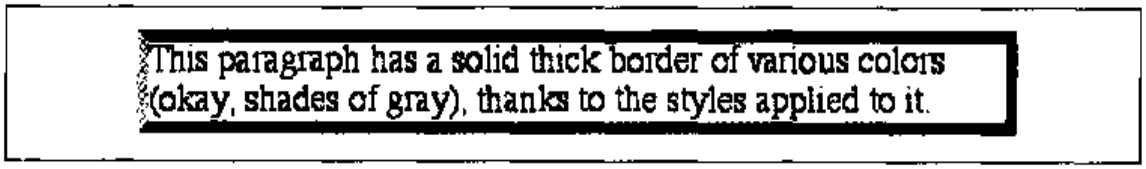


图 7-43 一个边框，多种颜色

如前所述，如果未定义颜色，则缺省颜色为元素的前景色。因此，以下声明会如图 7-44 那样显示：

```
P.shodel {border-style: solid; border-width: thick; color: gray;}
P.shode2 {border-style: solid; border-width: thick; color: gray;
   border-color: black;}
```

结果是第一段落有灰色边框，灰色是从段落自身的前景色得到的。第二段落有一个黑色边框，因为该颜色用 border-color 显示指定。

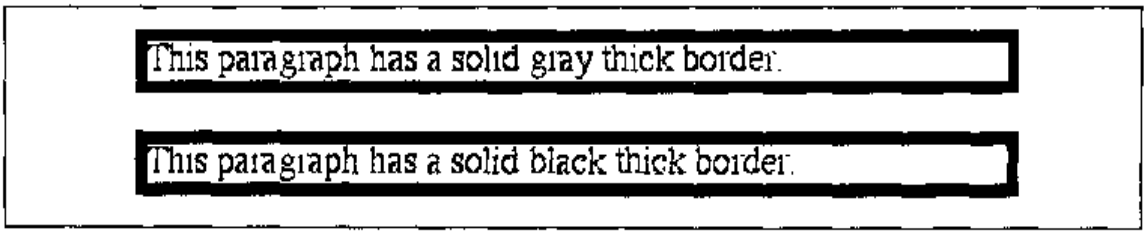


图7-44 边框颜色基于前景色（顶端）及基于border-color属性值（底端）

缩略边框属性

尽管有缩略属性如border-color及border-style，它们有时帮助不大。例如，若想设定所有H1元素有一个粗灰实心边框，但只在底端。有两种方法：

```
H1 {border-bottom-width:thick;    / *option #1 */
  border-style: none noen solid;
  border-color: gray;}
H1 {border-width: 0 0 thick;      /*option #2*/
  border-style: none noen solid;
  border-color: gray;}
```

两者都不方便，要完全输入。幸好，有一个好的解决方法：

```
H1 {border-bottom: thick solid gray;}
```

这样值将仅应用于下边框，如图7-45所示，其余部分使用缺省值。由于缺省边框样式为none，元素其他三个边都没有边框出现。

图7-45 缩略属性易于构建样式

可能读者已经猜到，共有四种这样的缩略属性。

border-top, border-right, border-bottom, border-left	
允许值	<边框宽度> || <边框样式> || <颜色>
初始值	参考各个属性
可否继承	否
适用于	所有元素

可以用这些属性生成一些复杂的边框，如图 7-46 所示：

```
H1 {border-left: 3px solid gray;
  border-right: black 0.25em dotled;
  border-top: thick silver inset;
  border-bottom: double rgb(33% 33% 33%) 10px;}
```

图 7-46 十分复杂的边框

如同所看到的那样，实际值的排列顺序并不重要。以下三个规则将生成完全一样的边框，如图 7-47 所示：

```
H1{border-bottom:3px solid gray;}
H2{border-bottom:solid gray 3px;}
H3{border-bottom:3px gray solid;}
```

也可以空缺一些值且使用其缺省值，如下：

```
H3 {color: gray; border-bottom: 3px solid;}
```

由于没有定义边框颜色，则缺省值（前景色）将被使用，如图 7-48 所示。记住，如果空缺了边框样式，那么其缺省值 none 将会使边框无法显示。

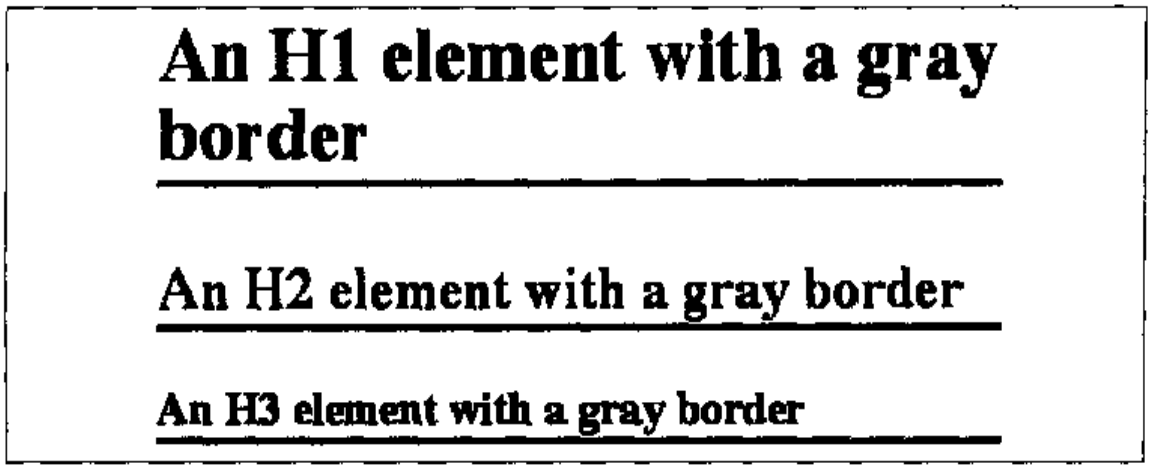


图 7-47 用三种不同方法得到相同的结果

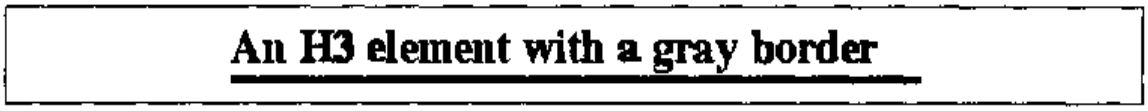


图 7-48 将缺省值填入空缺中

相反，如果只定义样式，仍能得到一个边框。例如，只是简单地想得到一个样式为dashed的边框用在元素的顶端，希望其宽度为缺省值medium，边框颜色从元素自身继承。这时，只须使用如下规则：

```
P.roof {border-top: dashed;}
```

值得注意的另一件事是由于这些“border-side”属性只应于某一特定边，没有值复制的可能，也没有任何意义。每种类型只能有一个值，也就是说，只有一个宽度值，只有一个颜色值，只有一个边框样式。所以不要试图声明多于一个的值类型：

```
H3{border: thin thick solid purple; /* two width values--WRONG */}
```

在这种情况下，整个声明无效且完全被忽略。

最后，还需要注意缩略属性的使用：如果省略了一个值，则自动填充为缺省值。这可能有意料之外的效果。考虑下例：

```
H4 {border-style: dashed solid double;}
H4 {border: medium green;}
```

这将导致H4元素根本没有边框，因为在第二个规则中缺少边框样式，缺省值none被自动使用，这将使整个边框消失。

尽可能快地设置边框

基于以上缩略素材，读者可能会推测还可以更进一步，是的。现在让我们了解一下所有缩略边框属性中最短的一个：border。

border	
允许值	<边框宽度> || <边框样式> || <颜色>
初始值	参考各个属性
可否继承	否
适用于	所有元素

这个属性的优点在于它十分简洁，尽管因为简洁也带来了一些局限性。首先我们将介绍border是如何使用的。如果希望所有的H1元素有粗、银色边框，十分简单，声明的显示见图7-49所示：

```
H1 {border: thick silver solid;}
```

图7-49 很短的边框声明

值将应用于四个边。这当然优于次优的选择，该选择是：

```
H1 {border-top :thick silver solid;
  border-bottom:thick silver solid;
  border-right:thick silrer solid;
  bor   er-left:thick silver solid;}   /* same as previous markup */
```

使用border的缺点是只能定义“全局”样式、宽度及颜色。换句话说，赋给border的值会应用于所有四条边上。如果想使某个边的边框有所不同，那就需要使用其他边框属性，当然，通过层叠仍能使制作者从中受益：

```
H1 {border: thick silver solid;
  border-left-width: 20px;}
```

第二条规则将替换掉由第一条规则指定的左侧边框的宽度值，即用20px代替thick，如图7-50所示。

图7-50 使用层叠以受益

边框与内联元素

这些内容读起来会非常熟悉，因为大部分内容与讨论过的边界与内联元素类似。

首先，无论内联元素的边框设定得有多粗，元素的高度都不会发生变化。我们对加粗文本设置顶端、底端边框：

```
B {border-top: 10px solid gray; border-bottom: 5px solid silver;}
```

这是规范所允许的，但它显然对行高没有影响。然而，由于边框是可见的，它们将显示出来，如图7-51所示。

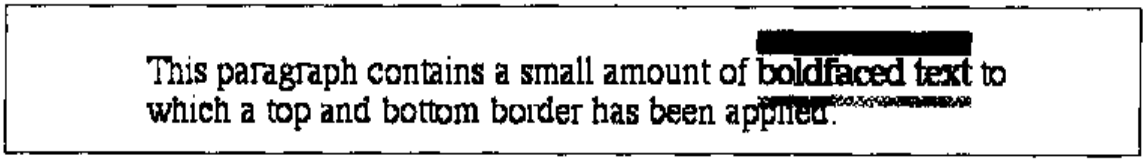


图7-51 内联元素的边框

边框必须位于某个位置，这就是它们所处的位置。

这只是针对内联元素的顶端、底端而言的。左侧与右侧则有所不同。我们首先考虑一种情形：一个小的内联元素位于单行内，如图 7-52 所示。

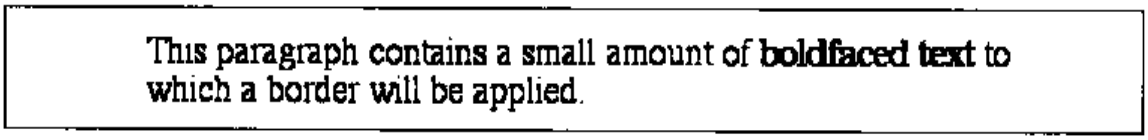


图 7-52 内联元素

如果为内联元素设置左侧与右侧边框，那么，不仅它们可见，它们还会替换周围的文本，如图 7-53 所示：

```
B {border-left: 10px double gray; background: silver;}
```

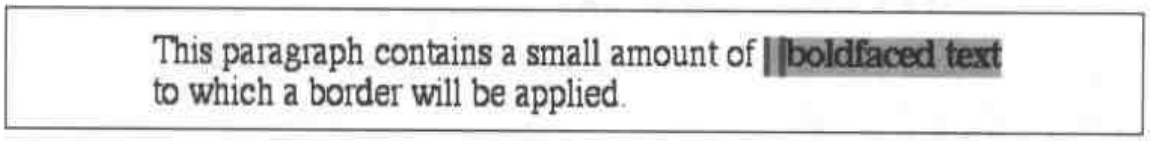


图 7-53 有左侧边框的内联元素

如同所期望的那样，图 7-53 在内联元素的左侧显示了额外的空间，而顶端和底端则没有。

使用边框，正如使用边界一样，浏览器关于行中断的计算不是直接由内联元素所设定的框属性影响的。它们的影响只是边框所占的空间使行的位置移动，从而可能会改变处于行尾的单词。内联元素设置边框并在多行显示的情况见图 7-54：

```
B {border: 3px solid gray; background: silver;}
```

图 7-54 中，左侧边框应用于元素的开始，右侧边框应用于元素的结尾。边框并不一定必须用这种模式来实现，如果环境要求如此，它们也可以应用在内联元素每行的左侧与右侧。例如，槽边框在每行结尾处封闭看起来更好些，如图 7-55 所示。

在图 7-54 中“开放”的行也是可接受的。

警告： 在 Navigator 4.x 或 Explorer 4.x/5.x 中，边框不能应用于内联元素。只有 Opera 3.x 在内联元素周围可以画边框，且只能罩在元素的头部与尾部。这符合 CSS 规范，尽管我们不在这里讨论这个问题（参见第八章可获得更详细的信息）。

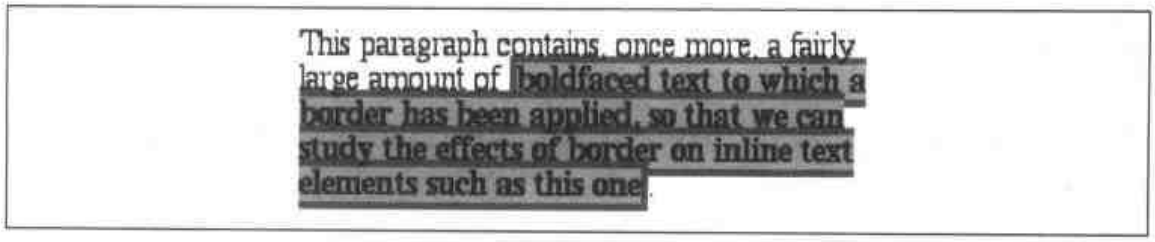


图 7-54 多文本行显示的带边框内联元素

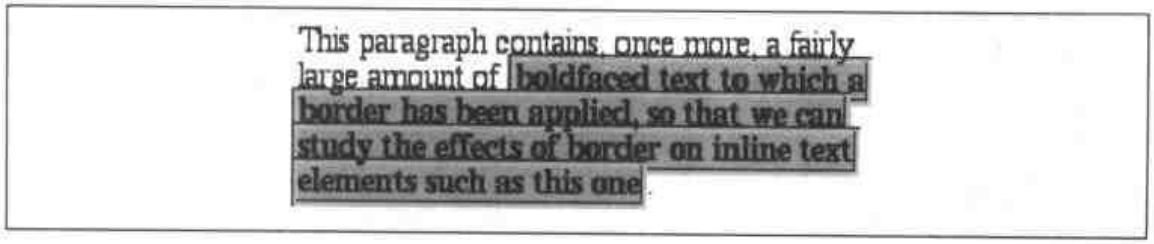


图 7-55 多文本行显示的带边框的内联元素，其边框盒状封闭

边框：已发现的问题

当然，使用边框也会有一些问题。最危险的是Navigator 4.x在块级元素的内容区域周围不画边框，而是在边框与内容之间插入空格。似乎还没有消除这种行为的方法。

抛开所受的限制不谈，border显然是一个非常有用的属性。甚至在Netscape Navigator 4.x可避开一个看起来完全无关的错误。如果指定一个元素的背景色并在Navigator 4.x中浏览它，颜色只在文本区出现，不会覆盖全部内容区及补白。可以用以下声明避开这个问题：

```
border {0.1px solid none;}
```

尽管没有边框出现，也应该与背景色无关，但它会使Navigator 4.x用指定的颜色填到内容区及补白的背景上。

对Navigator而言，设定边框，或其他框属性到内联元素是尤其危险的，其原因同不能使用边界的理由类似。

补白

补白位于元素框的边界与内容区之间。很自然，用于影响这个区域的属性是padding。

padding

允许值	[<长度> | <百分比>]{1,4}
初始值	0
可否继承	否
适用于	所有元素

注意：百分比是指相对于父元素的宽度。

正如所看到的那样，这个属性可以接受任何长度值或一个百分比。如果想让所有的H1元素在各侧都有10像素的补白，十分容易，结果如图7-56所示：

```
H1 {padding: 10px; background-color: silver;}
```

图7-56 应用于H1元素上的补白

另一方面，也许我们希望H1元素有不对称的补白，H2元素有对称的补白，如图7-57所示：

```
H1 {padding: 10px 0.25em 3px 3cm;} /* uneven padding */
H2 {padding: 0.5px 2em;} /* values replicated to the bottom and left sides */
```

想看到补白有些困难，所以加上一个背景色，如图7-58所示：

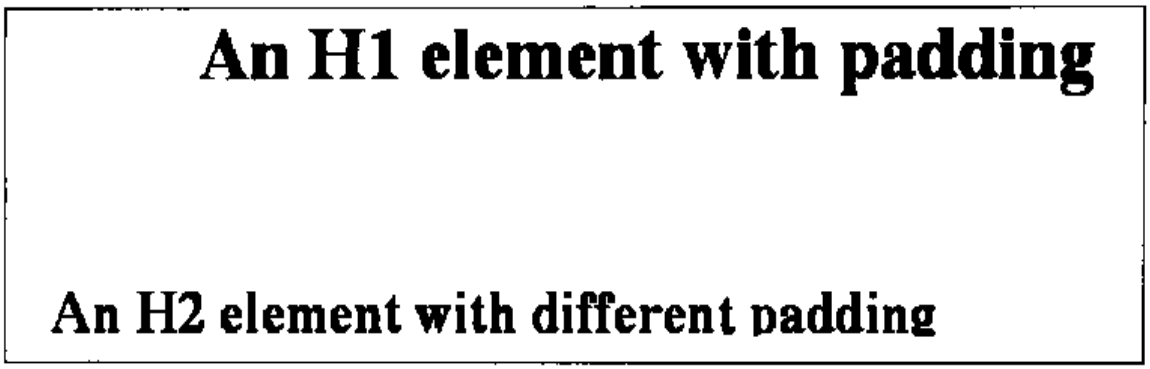


图 7-57 不对称的补白

```
H1 {padding: 10px 0.25em 3ex 3em; background: silver;}
H2 {padding: 0.5em 2em; background: silver;}
```

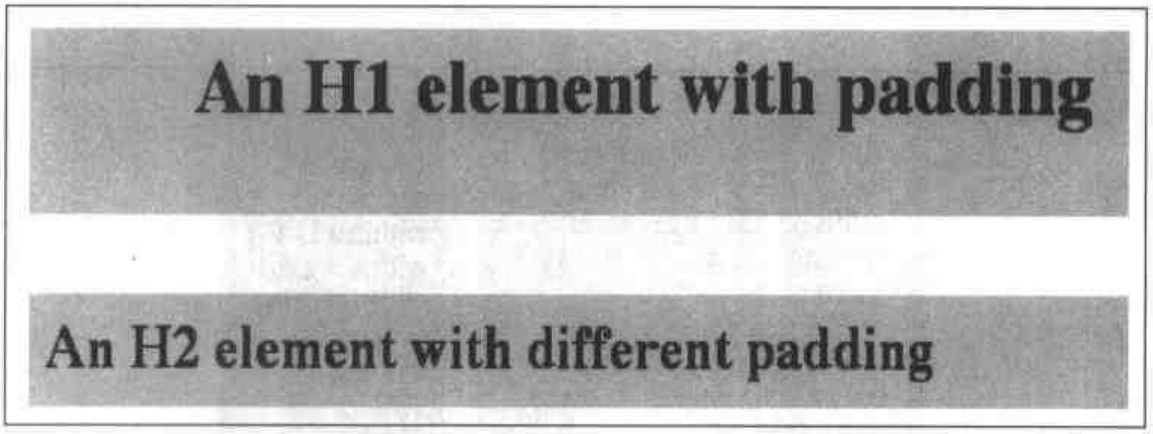


图 7-58 有背景的不对称补白

如图 7-58 描绘的那样，元素的背景延伸到补白中。如前面所讨论的，它还延伸到边框的外部边界，但背景必须首先经过补白才能到达边框。

padding 的缺省值是 0（zero），且补白的值不能为负。

警告：Opera 3.5 允许负补白值，但在 Opera 3.6 中修改了。其他浏览器不允许负的补白值。

百分比与补白

如同前面所声明的，可以为元素的补白设置百分比。百分比是参考父元素宽度计

算得到的，因此，当父元素宽度变化时，它们也将随之变化。例如，假设如下标记，图 7-59 为其结果显示：

```
P {padding: 10%; background-color: silver;}

<DIV STYLE="width: 200px;">
<P>This paragraph is contained within a DIV which has a width of 200 pixels,
so its padding will be 10% of the width of the paragraph's parent element.
Given the declared width of 200 pixels, the padding will be 20 pixels on
all sides.</P>
</DIV>
<DIV STYLE="width: 100px;">
<P>This paragraph is contained within a DIV with a width of 100 pixels,
so its padding will still be 10% of the width of the paragraph's parent.
There will, therefore, be half as much padding on this paragraph as that
on the first paragraph.</P>
```

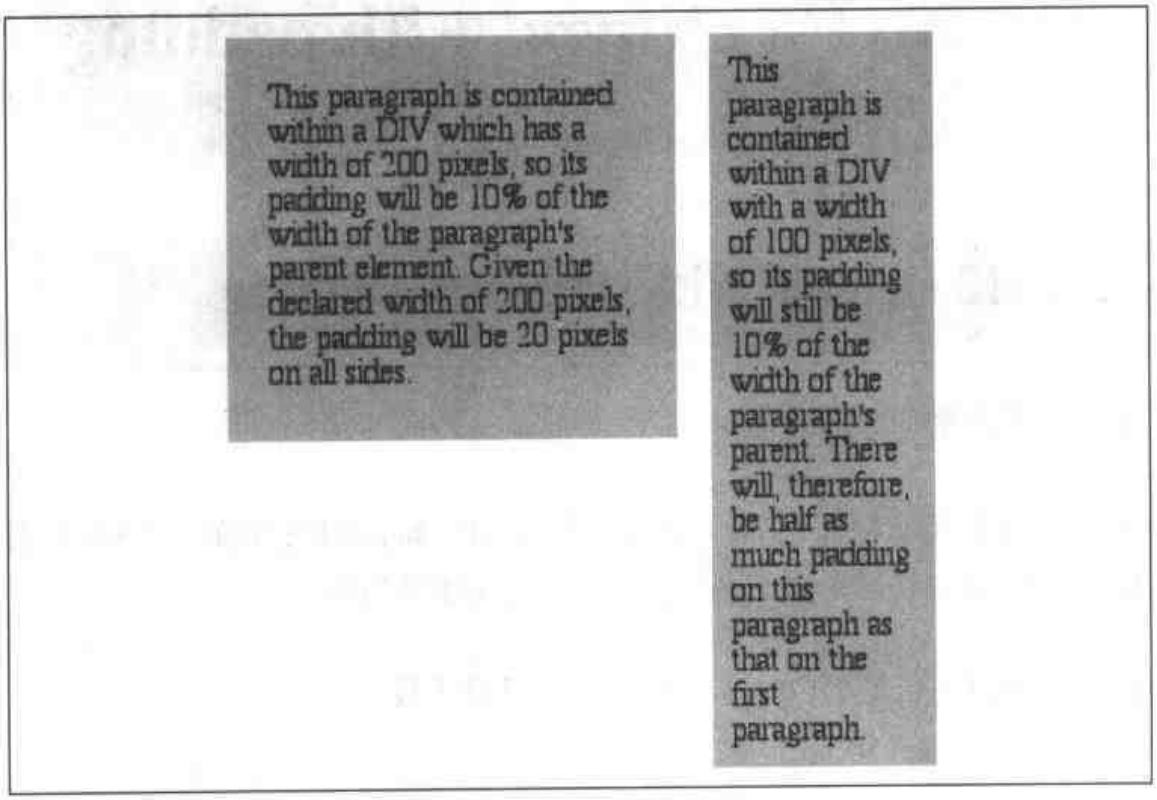


图 7-59 补白、百分比及父元素宽度

以前在“边界”一节，我们曾见到过它。但它值得再复习一次，我们只看看它如何运行即可。

单侧补白

读者可能猜想：有一些属性可以为框的单侧设定补白，而不影响其他侧。

padding-top, padding-right, padding-bottom, padding-left

允许值	<长度> | <百分比>
初始值	0
可否继承	否
适用于	所有元素

注意：百分比是指相对于父元素的宽度。

这些属性正如期望的那样操作。例如，以下两条规则生成等量的补白：

```
H1 {padding: 0 0 0 0.25in;}
H2 {padding-left: 0.25in;}
```

补白与内联元素

在应用于内联元素时，边界和补白没有大的差别。让我们换个方式，先谈谈左侧与右侧补白，如果为左侧补白和右侧补白设值，它们可见，如图7-60所示可以很明显地看到。

```
B {padding-left: 10px; padding-right: 10px; background: silver;}
```

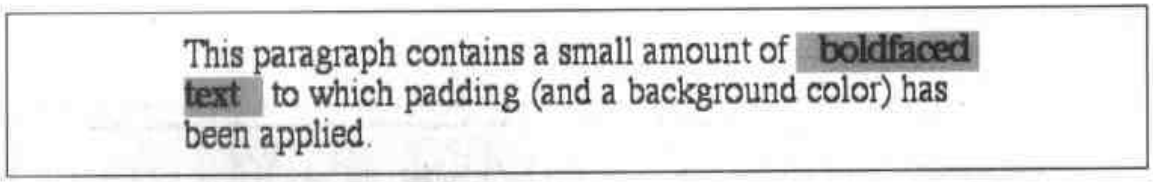


图7-60 用于内联元素的补白

注意加粗文本两端显示出的额外空间，那就是补白。

这些看起来都很熟悉，包括加粗文本多行显示的情形。图7-61显示了补白设置在多行显示的内联元素上的情况。

```
B {padding: 10px; background: silver;}
```

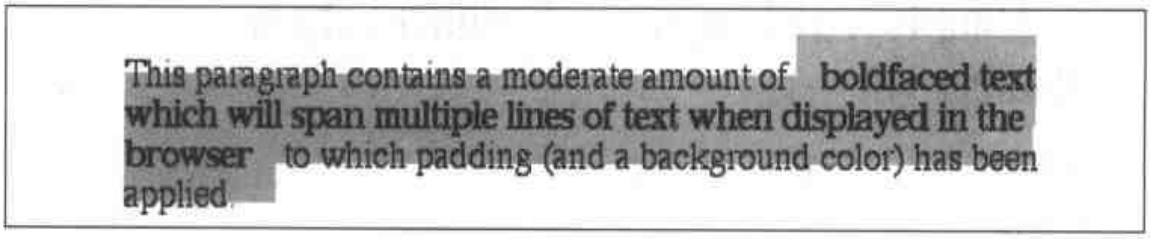


图7-61 补白应用于多行内联元素

同边界一样，左侧补白应用在元素的开头，右侧补白应用在元素的结尾，补白不应用在每一行的开头和结尾。

现在来讨论顶端与底端补白。理论上，一个有背景色及补白的内联元素可以使背景延伸到元素的顶端与底端。从图7-61中可以看到可能的外观。当然，行的高度没有改变，但既然补白中延伸进了背景，那就可以看到，不是吗？

回想那句著名的短句："也许会有具体实现限制。"用户代理不必支持这种效果。

补白：已发现的问题

首先，补白与Navigator 4.x明确不相容。主要的问题是虽可以为带背景色的元素设补白，但背景不会延伸到补白中。需要加一个边框，如在"边界：已发现的问题"中讨论的那样。因此，如果为元素设定了背景色，补白边框，可以看到背景按要求填到了内容区及补白中，但在两者间有一透明空间错误地出现，如图7-62所示。

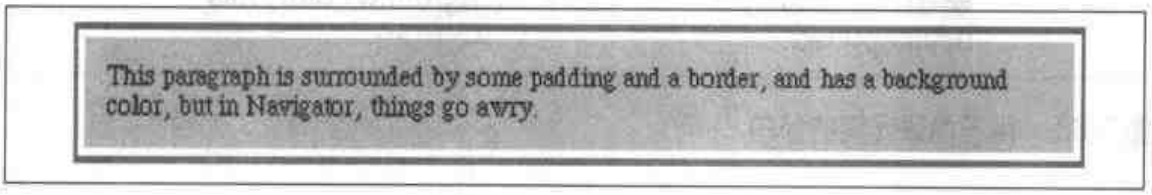


图7-62 Navigator 4中的补白问题

这也许是一种有趣的效果，但不符合CSS规范，且其他浏览器不会这样，因此，最好避开它。

更坏的是，如果在Navigator 4.x中试图将补白应用在内联元素时，将会引起大的混乱。与应用边界在内联元素时类似，在将补白应用其上时也会发生。因此，明智之举是避开设定边界、边框及补白到内联元素上。

Opera 3.5错误地允许补白使用负值，但版本3.6不再受其害了。Internet Explorer 4.x根本不将补白应用到内联元素上。

浮动与清除

读者可能知道浮动元素的概念，从Netscape 1.0开始，就可以通过声明来使图像浮动。如<IMG SRC="B5.gif" align ="right">，会使图像向右方浮动，还允许其他内容（文本或图像）围绕图像“流动”。过去，能够浮动的只有图像，某些浏览器还支持表格。CSS则允许任何元素浮动，从图像到段落到列表。在CSS中，是由属性float来完成这些行为的。

float	
允许值	left | right | none
初始值	none
可否继承	否
适用于	所有元素

例如，向右浮动一个图像，可使用如下标记：

```
<IMG SRC="b5.gif" style="float: right"; alt="section b5">
```

如图7-63所示，图像向浏览器视窗的右侧浮动。这正是我们所期望的。然而，在CSS中浮动元素也引发了一些有趣的结果。

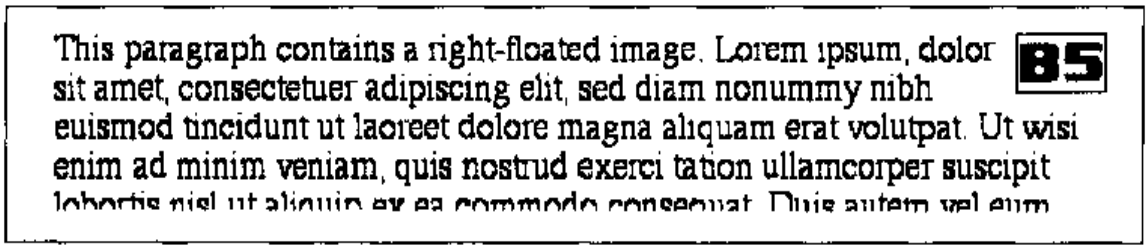


图 7-63 一个浮动的图像

浮动的元素

关于浮动元素，必须记得一个浮动的元素在一些特性上脱离了普通的文档流，尽管它仍影响着布局。在严格与CSS一致的情况下，浮动的元素几乎在它们自己的平面上出现，但仍对文档的其余部分有重大的影响。

当然，当一个元素浮动时，其余内容也随之浮动。这一行为与浮动的图像类似，对浮动的段落而言也是如此。例如，在图 7-64 中，我们可以看到这个效果（为使之更清晰，这里我们加入了边界）：

```
P.aside {float: left; width 5em; margin 1em;}
```

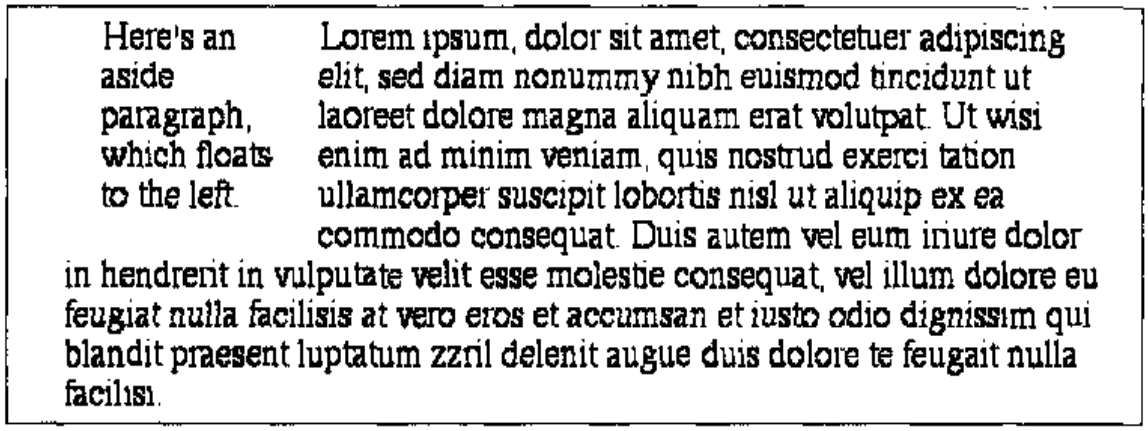


图 7-64 一个浮动着的段落

关于浮动的元素最有趣的现象是浮动的元素周围的边界并不被压缩。如果浮动有 20 像素边界的图像，那么在其周围至少有 20 像素的空白。如果另一个元素与之相邻——水平相邻且垂直相邻，且该元素也有边界，则那些边界不会因浮动元素的边界而压缩。如图 7-65 所示：

```
P IMG {float: right; margin: 20px;}
```

（再次使用纸片－塑料模拟，图像的塑料部分不会重叠围绕其他元素的塑料。）

如果要浮动一个文本元素，一定要记住除非为元素声明宽度，否则CSS规范会使其宽度倾向于0，这样，浮动的段落严格为一个字符宽，假如那是浏览器的宽度最小值。为了避免这个问题，一定要为浮动元素声明宽度。否则，会得到类似于图7-66的结果。

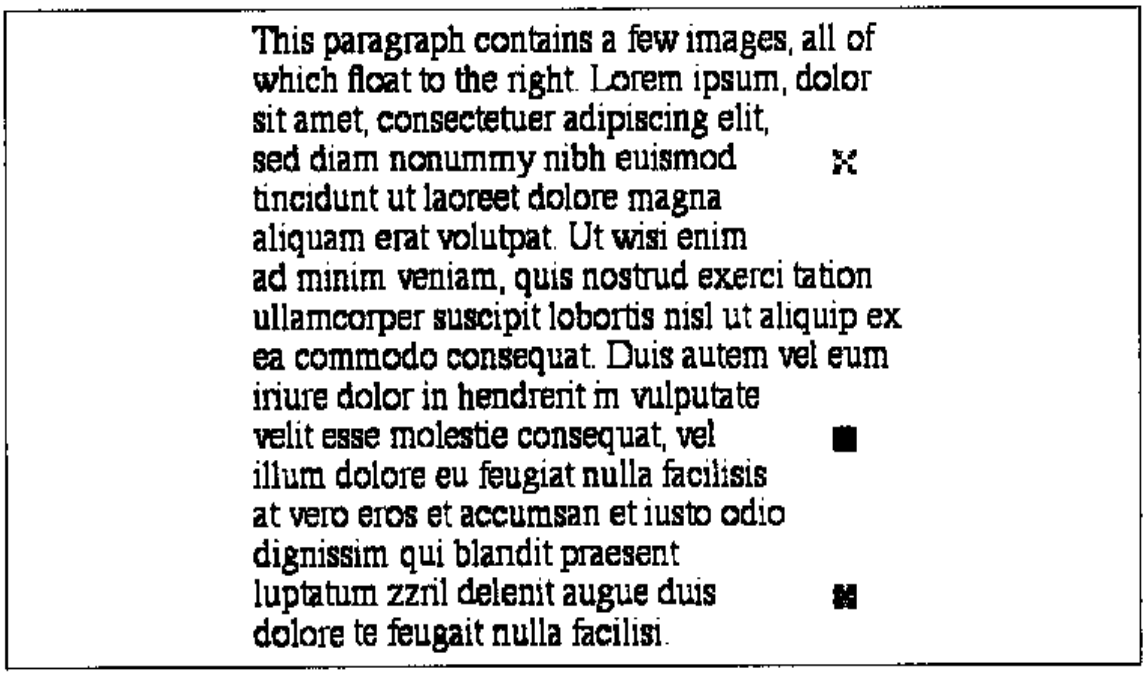


图7-65 带边界的浮动图像

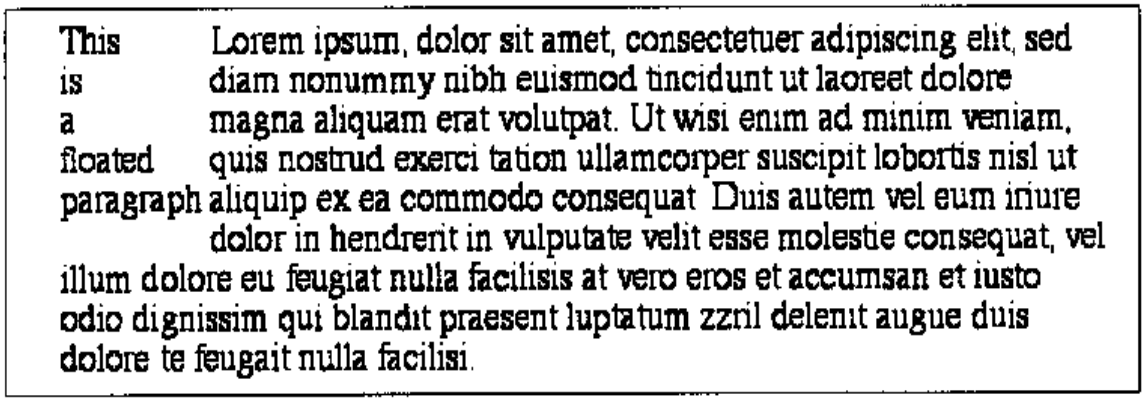


图7-66 带明确宽度的浮动文本

背景与浮动

还有许多有意思的效果与浮动元素有联系。例如一个小文档，由不多的几个段落及 H3 元素组成，第一段包含一个浮动元素。这个浮动图像有 5 像素的右侧边界，期望的文档绘制结果如图 7-67 所示。

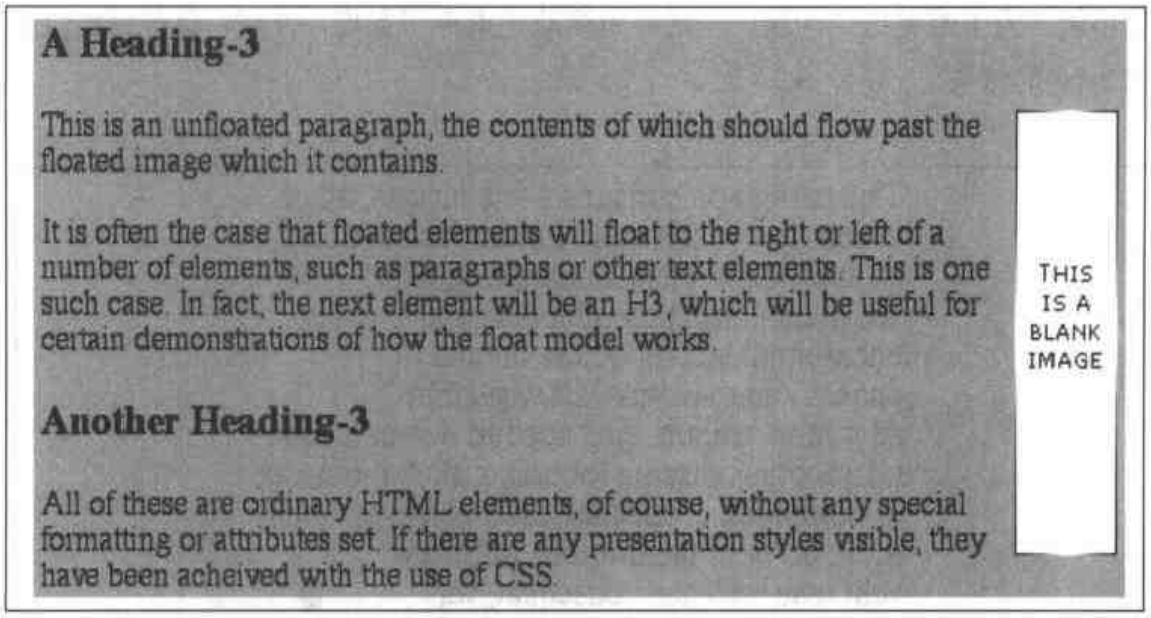


图 7-67 浮动一个图像

没有任何异常发生。但注意看当我们为第一段落设背景时，会发生什么事情，如图 7-68 所示。

第二个例子除了可视背景外，没有任何区别。正如见到的那样，浮动的图像突出于父元素的底侧。当然，在第一个例子中也是如此，只是因为无法看到背景而不明显。这种行为未被禁止。

警告： 在实践中，有些浏览器可能无法正确地做这件事。它们会增加父元素的高度以将浮动元素包含在内，这样做也会导致大量空白空间出现于父元素中。

于是就产生了一个问题：对具有可见背景而且流经浮动元素的元素会发生什么呢。让我们继续前例并修改它，使第二个 H3 元素具有可见背景及边框，如同图 7-69 中那样。

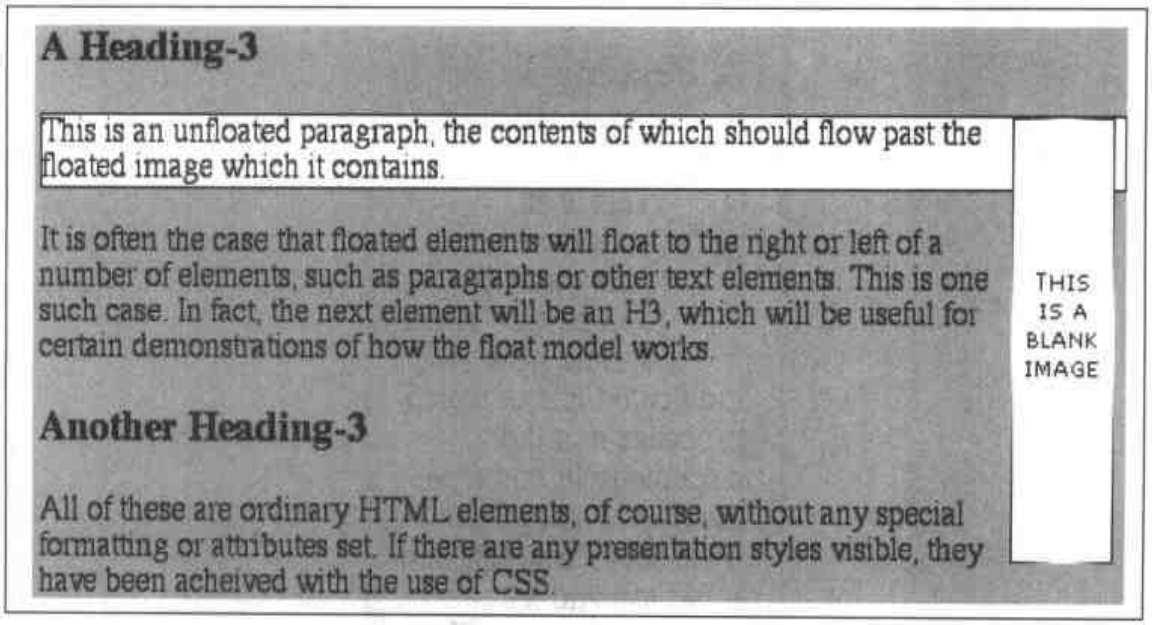


图 7-68 浮动图像与元素背景

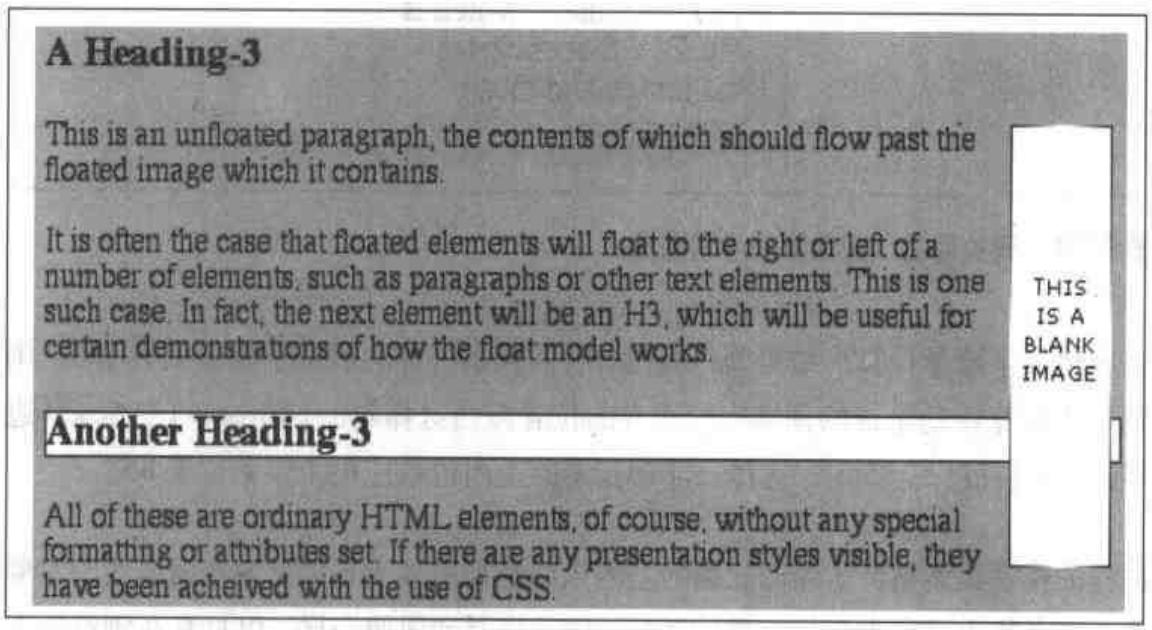


图 7-69 多个浮动图像及元素背景

是的，设置是正确的：H3的内容流经图像，背景“从底下滑过”图像。这与可见背景的段落包含浮动图像的例子没有方式上的差别。

负边界与浮动

负数经常使情形复杂起来。考虑一个有-15px左侧及顶端边界的图像向左侧浮动。图像放置于一个没有补白、边框、边界的DIV中。结果见图7-70。

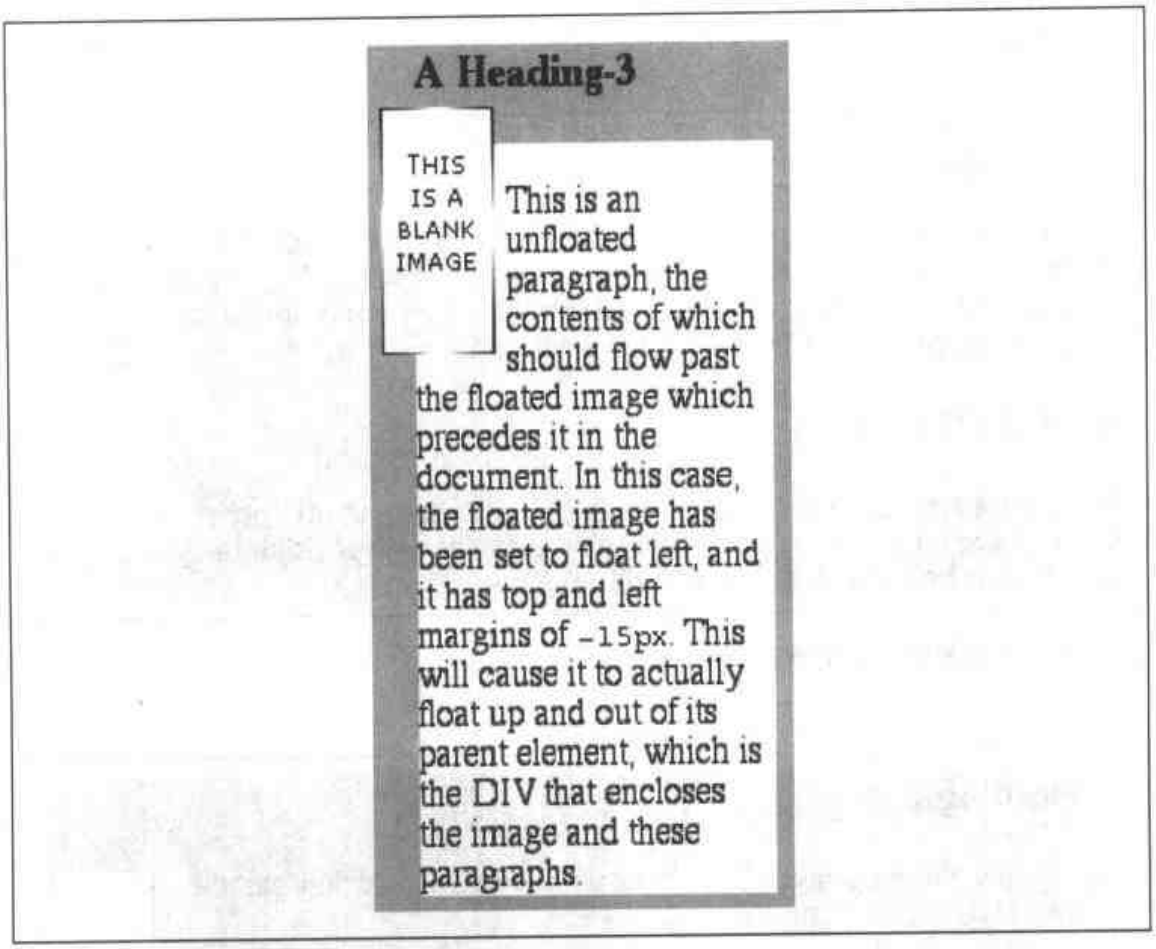


图 7-70　有负值上、左边界的浮动图像

这种情况下数学计算出结果为：假设 DIV 的顶端内部边界位于像素位置 100。浏览器为计算浮动单位的顶端内部边界的位置应当这样做：100px + (-15px)(边界) + 0 (补白) =85px。这样，浮动元素的上内边线应位于像素位置 85。

同样的推理线索解释了浮动元素的左内边线是如何放于其父元素左内边线的左侧的。这个能力可用来产生有趣的效果，同悬挂浮动图像一样，但只有在浏览器支持负边界用于浮动元素的情况下可行，否则，结果会如图 7-71 所示。

有一个重要的问题：当一个元素使用负边界浮动出父元素时，文档显示会怎样呢？例如，一个图像可以向上浮动很远以至于侵入到用户代理早已显示过的段落中。

在这种情况下，应该是用户代理的职责，但 CSS 规范严格声明用户代理不需回流以前的内容以适应后来的文档中发生的变化。换句话说，如果一个图像向上浮动

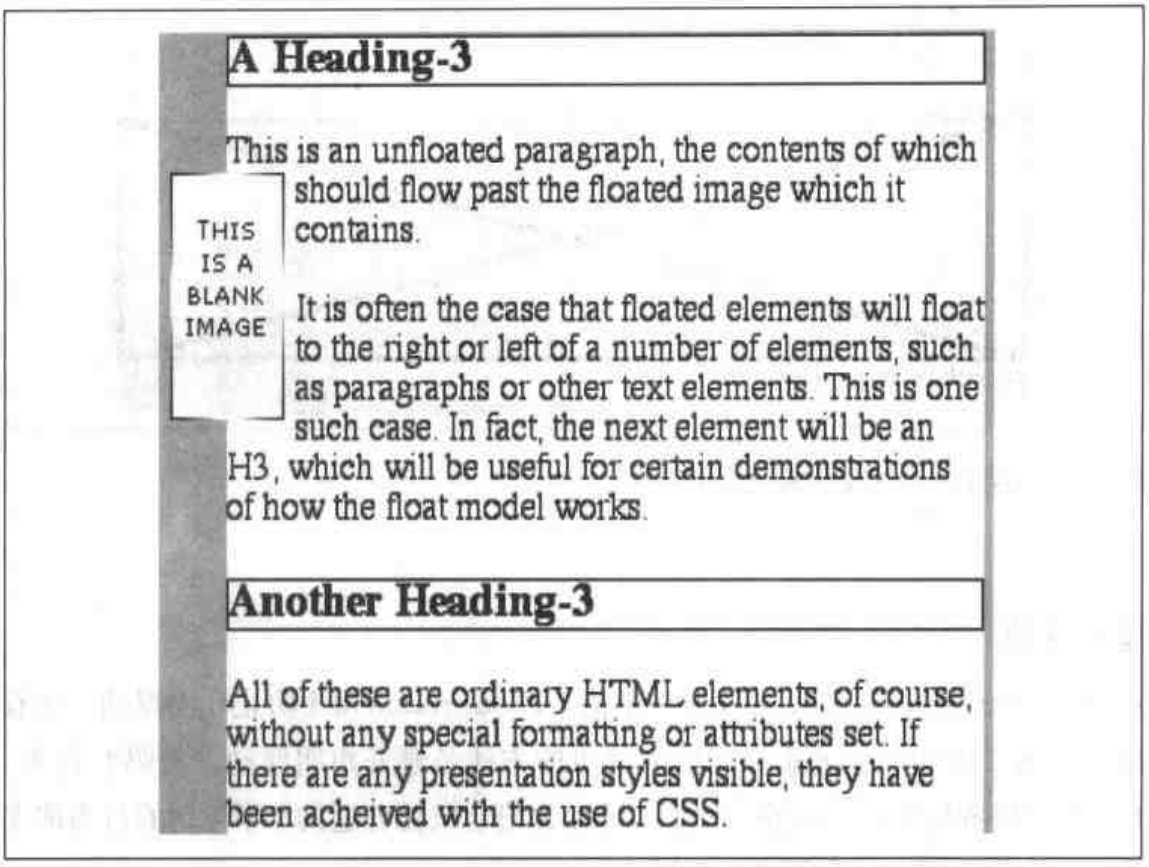


图 7-71　悬挂浮动

到先前的段落中，那它就会重写那里的内容。另一方面，用户代理可以通过流动浮动周围的内容来处理这种情况。依赖于特殊行为不是一个好主意，这会使负边界应用于浮动的功能大大受限。悬挂浮动很安全，但试图在网页内向上推一个元素通常也不合适。

当浮动元素可以浮动出它的父元素时，还可能有一种情况，发生于浮动元素比它的父元素还要宽时。在这种情况下，浮动元素在试图正确显示自身时，必然会超出左内边线或右内边线，这取决于它浮动的方式。在这种情况下，结果可能如图 7-72 所示。

这里，一个向左浮动的图像比它的父元素宽，因此，它的右边线超出了父元素的右边线。假如图像向右浮动，则它会超出它的父元素的左边线。

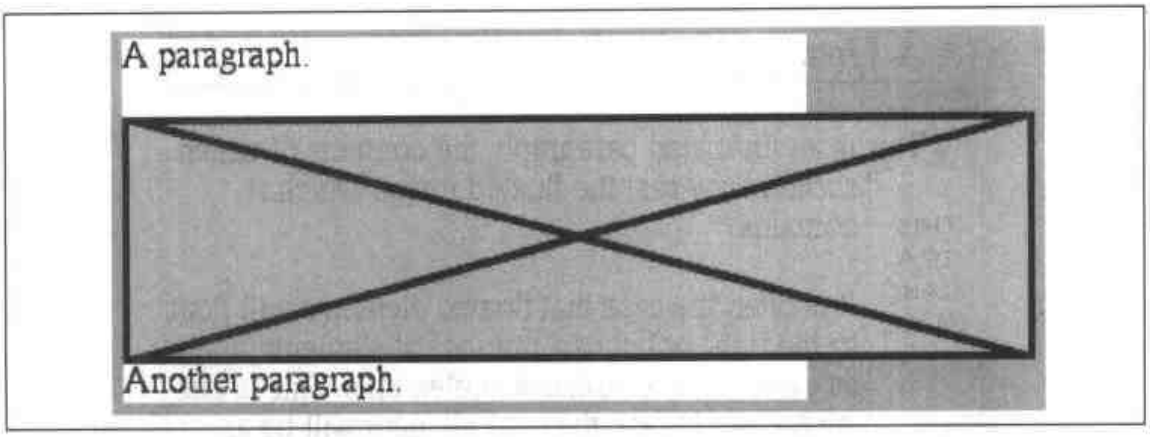


图 7-72 浮动一个比父元素还宽的图像

没有浮动

除 left 和 right 外，float 还有一个值。float:none 用于防止元素浮动。这看起来很笨，因为防止元素浮动的最简单的方法是避开声明浮动，对吗？首先，float 的缺省值是 none。换句话说，值必须出现以使普通的、不浮动的行为被支持，没有它，所有元素都将开始浮动。

第二，有可能从一输出样式列表中重载一个指定的样式。假设读者正使用一个宽服务样式列表，它浮动图像。在一个特殊页面上，不想让图像浮动，则不必写一个完整的新样式列表，只需在文档的嵌入样式列表中加入 IMG {float: none;}。除这种情况外，在 HTML 文档中没有更多的 float:none 的应用。

清除

我们谈论了很多浮动行为，因此只剩一件事待讨论。读者可能并不总希望页面内容流经一个浮动元素 —— 有时，或许非常希望防止它的发生。

如果一篇文档已分好章节，读者也许不希望浮动元素从一节中降到另一节。这种情况下，可以设置每节的第一个元素以禁止浮动元素在它的旁边出现。如果它被放置为与浮动元素相邻，则它下沉至低于浮动元素为止，所有的子序列内容显示在它后面，如图 7-73 所示。

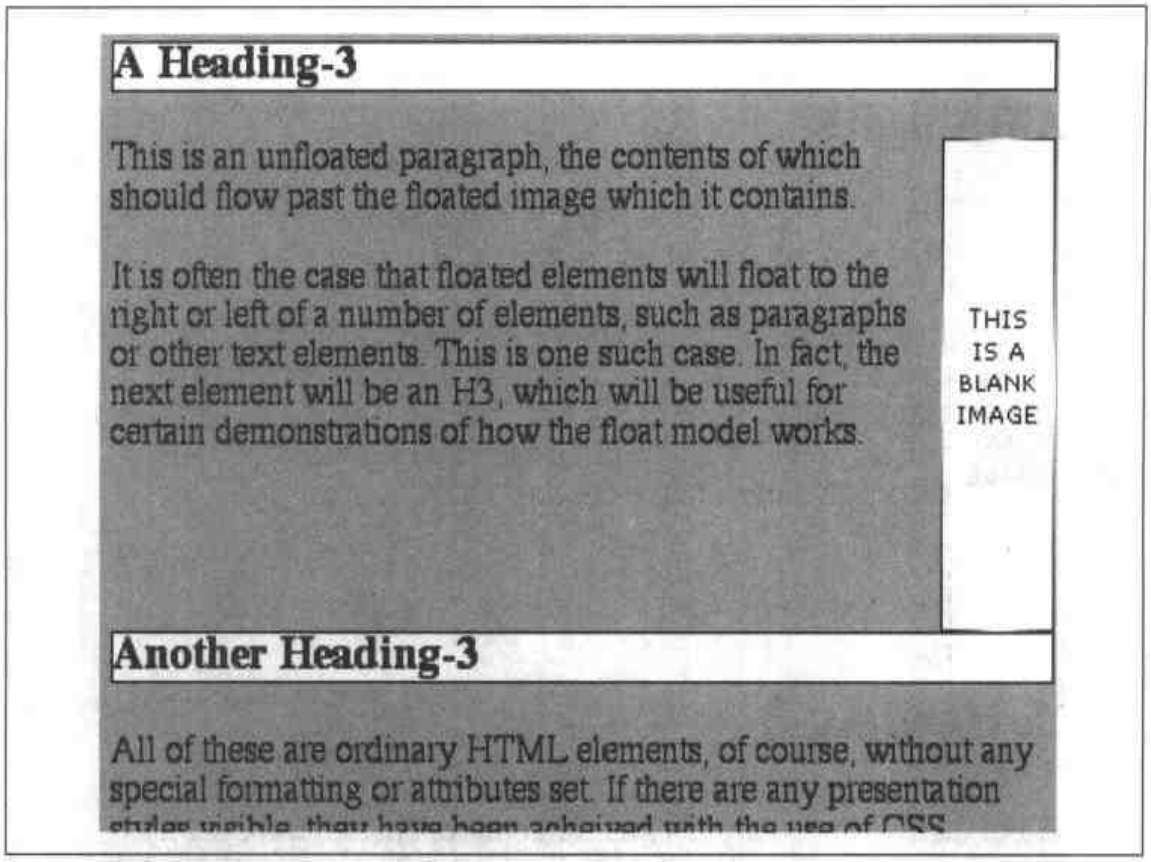


图 7-73 在清除中显示元素

这由 clear（清除）完成。

clear

允许值	left | right | both | none
初始值	none
可否继承	否
适用于	所有元素

例如，为确信所有H2元素没有放置于向左浮动元素的右边，可以声明H2 {clear: left;}。这可以翻译成“确保H2左边没有浮动元素”，而且它是HTML构成<BR clear="left">的替换。图7-74显示了下面的声明，该声明使用clear以防止H2元素流经左部的浮动元素：

```
H2 {clear: left;}
```

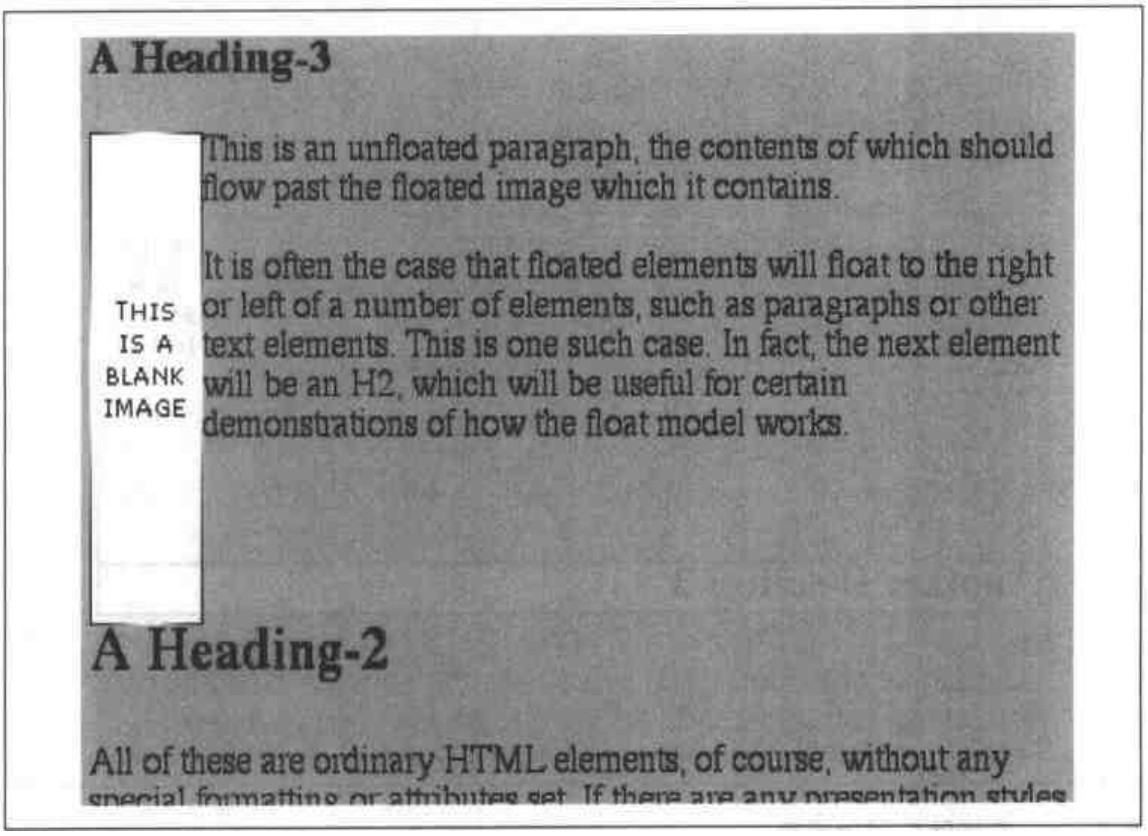


图 7-74 清除左侧

然则，这将允许浮动元素在 H2 元素的右侧出现，如图 7-75 所示。

为避免这类事情发生，应确保H2元素不与任何浮动元素在同一行中出现，使用值 both。这个值在元素两边均防止与浮动元素同时出现，如图 7-76 所示:

```
H2{clear;both;}
```

另一方面，如果我们只担心流经位于其右的浮动元素，那么我们可以使用 H2 {clear: right;}，结果如图 7-77 所示。

最后是clear:none，它允许浮动元素位于其两侧。none也可用于重载其他样式，如图 7-78 所示。尽管文档宽度规则规定 H2 元素两侧均不能有浮动元素，一个特别的 H2 元素仍可被设置成允许其两侧出现浮动元素:

```
H2{clear;both;}

<H2 STYLE="clear: none;">Not Cleared!</H2>
```

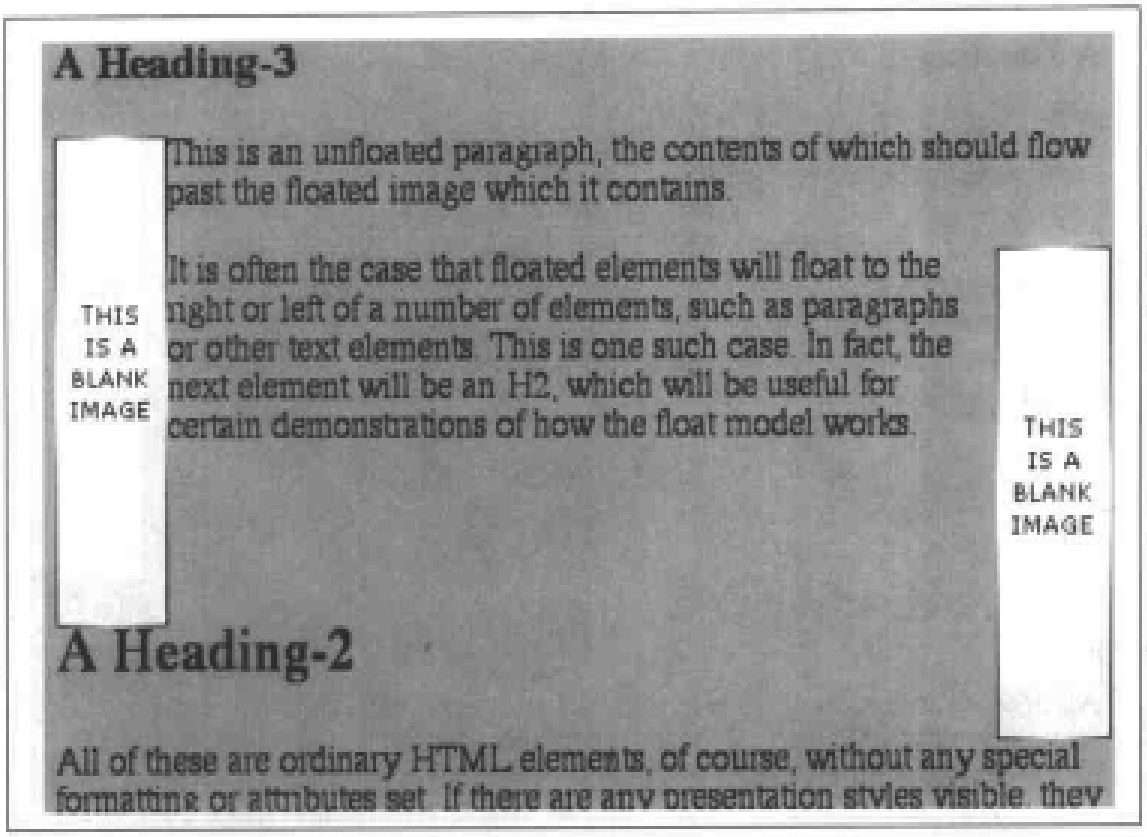


图 7-75 清除左侧，而不是右侧

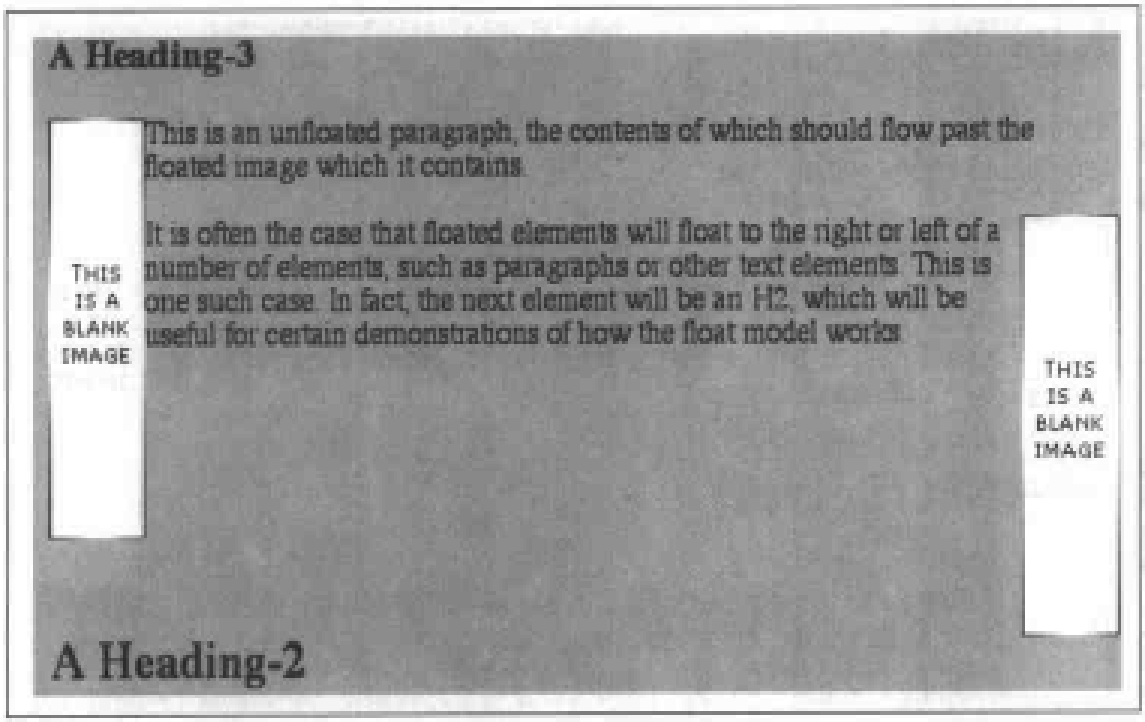


图 7-76 清除两侧

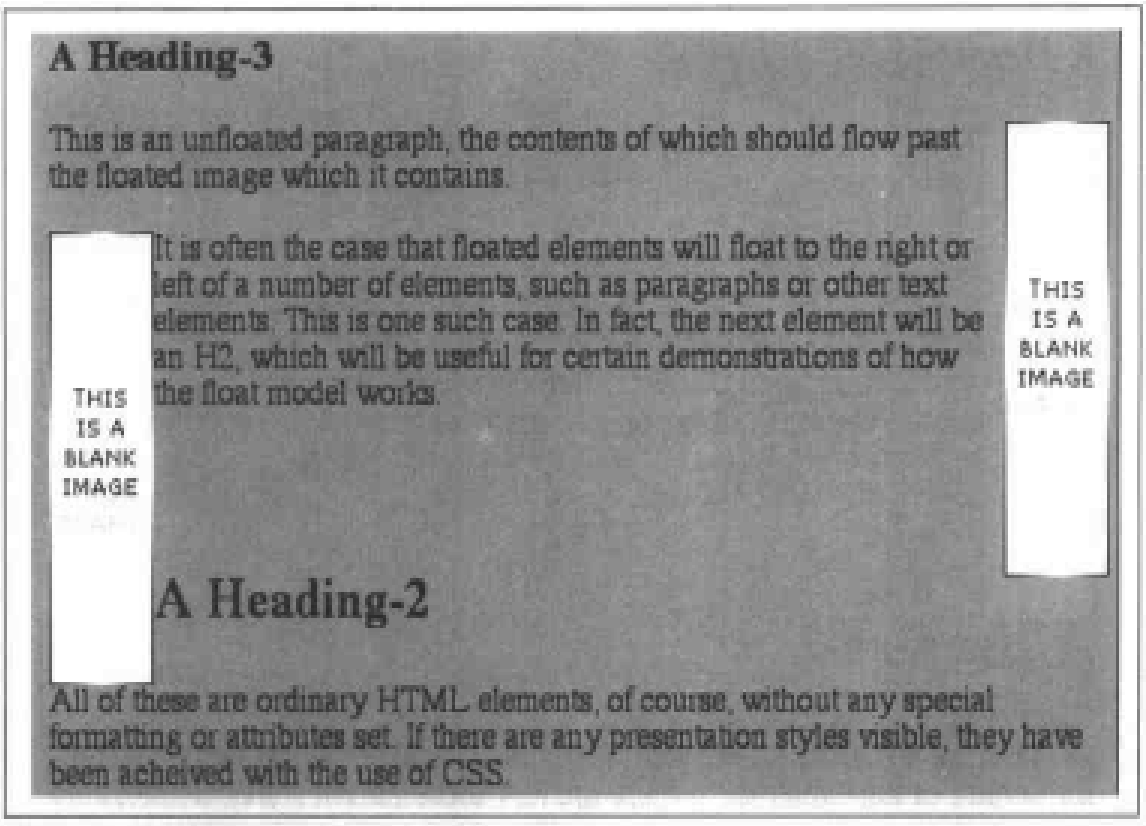


图 7-77 清除右侧

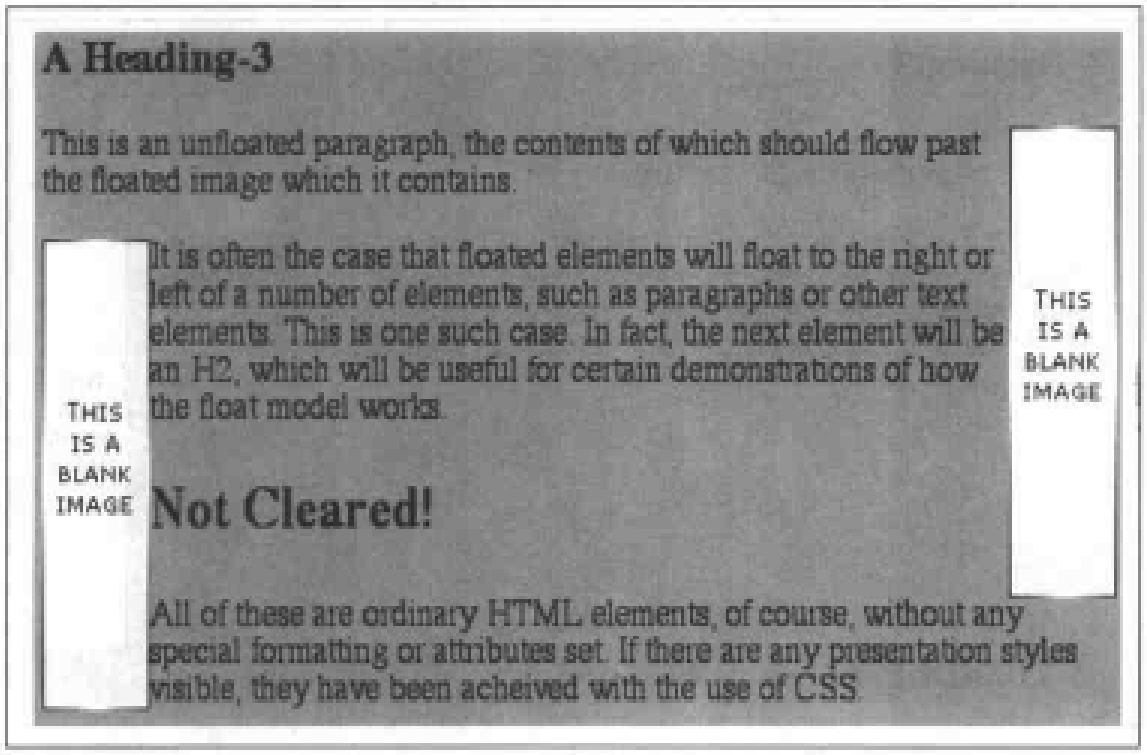


图 7-78 没有清除的情况

clear的工作原理是增加元素的上边界，以使它在低于浮动元素的位置结束，因此清除元素的上边界宽度应当有效地忽略掉。也就是说，一个1.5em的上边界，有可能会增大到10em，或25px，或7.133in，或是为使元素下降至内容区低于浮动元素下端所需的长度。

列表

在CSS1中，共有三种属性可以影响列表项的显示，在CSS2中又有所增加，增加的属性将在第十章“CSS2展望”中介绍。一个缩略属性把它们连在一起。这些属性用来影响列表中使用的项目符号（bullet）类型，将项目符号换成图像及影响项目符号或图像相对于列表项文本的出现位置。

“项目符号”是指列表项旁边的小装饰，如图7-79所示。

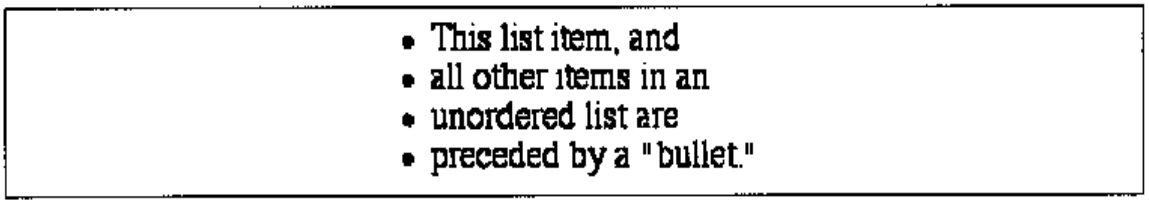


图7-79 项目符号

在无序列表中，它们是小项目符号，而在有序列表中，项目符号可能是字母或数字。

列表项类型

如果读者对HTML中的列表知道得很详细，那也会很熟悉这一部分，为改变列表项的计数或项目符号类型，可以使用List-style-type。

list-style-type

允许值	disc | circle | square | decimal | upper-alpha | lower-alpha | upper-roman | lower-roman | none
初始值	disc
可否继承	是
适用于	列表项元素

值的含义见表7-1。

表7-1 list-style-type属性值及结果

关 键 字	效果
disc	列表项项目符号使用disc（通常为实心圆）
circle	列表项项目符号使用circle（通常空心圆）
square	列表项项目符号使用square（实心或空心方块）
decimal	1，2，3，4，5，...
upper-alpha	A，B，C，D，E，...
lower-alpha	a，b，c，d，e，...
upper-roman	I，II，III，IV，V，...
lower-roman	i，ii，iii，iv，v，……
none	不使用项目符号

这些属性当然只能用于有列表项显示的元素。CSS不区分无序与有序列表项，因此，可以使有序列表使用圆盘而不是使用数字。实际上，list-style-type的缺省值就是disc，如果没有特殊的声明，所有列表（有序与无序）将使用圆盘作为每个项的项目符号。实际上，这个决定是用户代理的职责。即使用户代理没有预定义规则，如OL {List-style-type: decimal;}，它仍可以禁止有序项目符号用在无序列表上，反之亦然。但不能过分依赖它，使用时要小心。

如果想取消项目符号的显示，可以选择值none。none会使用户代理阻止放置任何

东西到项目符号通常应出现的地方，但并不中断有序列表的计数。这样，以下标记会生成如图 7-80 所示的结果：

```
OL LI {list-style-type: decimal;}
LI.off {list-style-type: none;}

<OL>
<LI>Item the first
<LI CLASS="off">Item the second
<LI>Item the third
<LI CLASS="off">Item the fourth
<LI>Item the fifth
</OL>
```

1. Item the first
   Item the second
3. Item the third
   Item the fourth
5. Item the fifth

图 7-80　交错的列表项标记

list-style-type是可继承的，因此如果希望在嵌套列表中使用不同类型的项目符号，就必须单独定义它们。还需要显式声明嵌套列表样式，因为用户代理的样式列表也许已经定义了这样的样式。假定一个用户代理有如下样式定义：

```
UL {list-style-type: disc;}
UL UL {list-style-type: circle;}
UL UL UL {list-style-type: square;}
```

如果是这样的话，就必须定义自己的样式，以覆盖用户代理的样式。在这种情况下，仅有继承是不够的。

列表项图像

有时，预生成的项目符号不能满足要求，需要为每个列表项使用一个图像。过去，为达到这种效果只能仿造它。现在只需使用 list-style-image 声明即可。

list-style-image

允许值	<url> | none
初始值	none
可否继承	是
适用于	列表项元素

这是它的工作过程：

```
UL LI {list-style-image: url(ohio.gif);}
```

这就是所需的一切。一个简单的URL值，就可将图像用作项目符号，如图7-81所示。

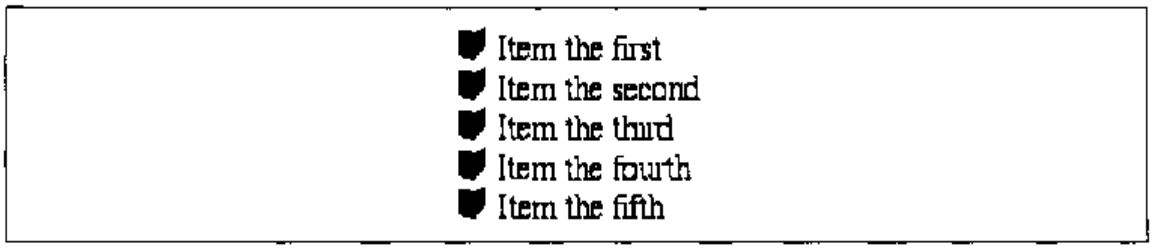


图7-81 图像用作项目符号

当然，还要仔细检查使用的图像，本例中的图像过于清晰了（图7-82所示）：

```
UL LI {list-style-image: url(big-ohio.gif);}
```

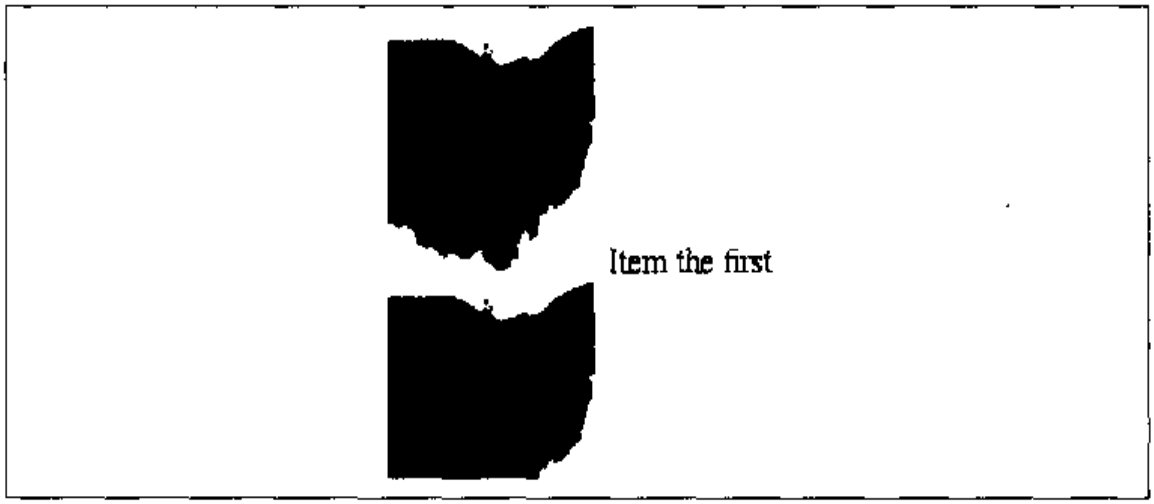


图7-82 将过大的图像用作项目符号

应经常为项目符号类型提供回退。用于未载入图像、发生冲突、用户代理不支持这种格式的显示(如图7-83所示)等情况。因此，应为列表定义一个备份的list-style-type:

```
UL LI {list-style-image: url(ohio.bmp); list-style-type: square;}
```

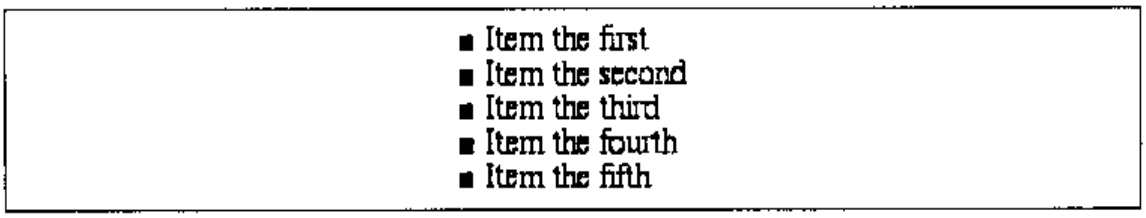


图7-83 为不能使用图像提供回退

还可以将list-style-type设置为缺省值none。这是个好主意，因为list-style-image是可继承的，因此除非采用措施，否则所有嵌套列表均会使用该图像为项目符号:

```
UL {list-style-image: url(ohio.gif); list-style-type: square;}
UL UL {list-style-image: none;}
```

由于嵌套列表继承了表项类型square，且设置为不使用图像作为项目符号，嵌套列表使用square作为项目符号，如图7-84所示。

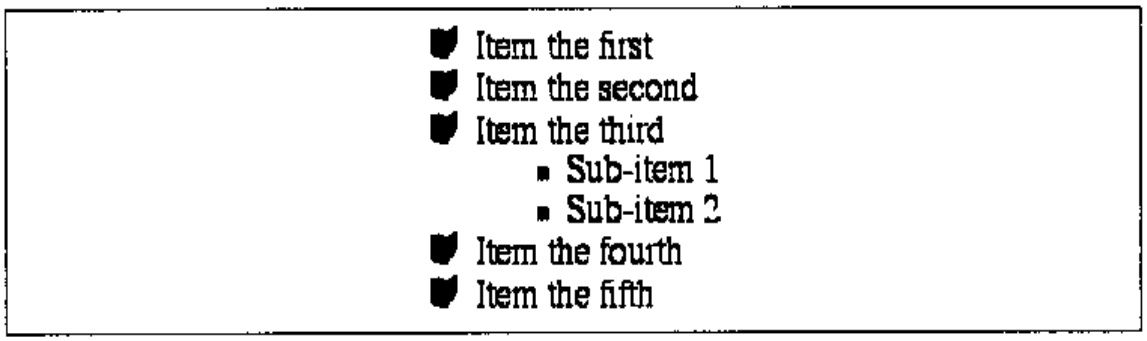


图7-84 在子列表中去除图像项目符号

警告: 在实践中这可能不成立。用户代理也许已经为UL UL定义了list-style-type，因此值square根本不会被继承。随着浏览器的不同，显示结果会有所不同。

对于有序列表，CSS2能提供更强大的排序控制。例如，CSS1中无法自动生成子序数，如“2.1”或“7.1.3”，而用CSS2则可以完成，这些内容在第十章中有简要的介绍。

列表项位置

在CSS1中无法改变列表项的外观，所谓影响外观是指改变项目符号自身相对于列表项内容的位置。这是由list-style-position来完成的。

list-style-position	
允许值	inside | outside
初始值	outside
可否继承	是
适用于	列表项元素

如果项目符号位置设为outside，则会按Web中列表项的正常状况显示，如图7-85所示：

```
LI {list-style-position: outside;}
```

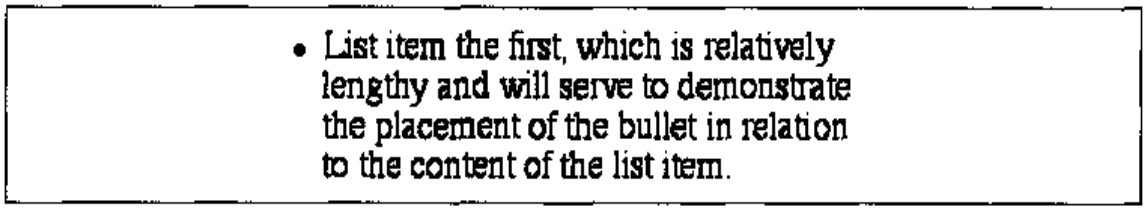


图7-85 项目符号放置于列表项之外

如果希望外观有所变化，可通过将它设置为inside使项目符号进入到内容中：

```
LI.first {list-style-position: inside;}
```

这将使项目符号放置于列表项内容的“内部”。所谓“内部”没有定义精确的方式，图7-86显示了一种可能。

CSS2 提供了更多的项目符号位置控制，第十章将就这些内容进行讨论。

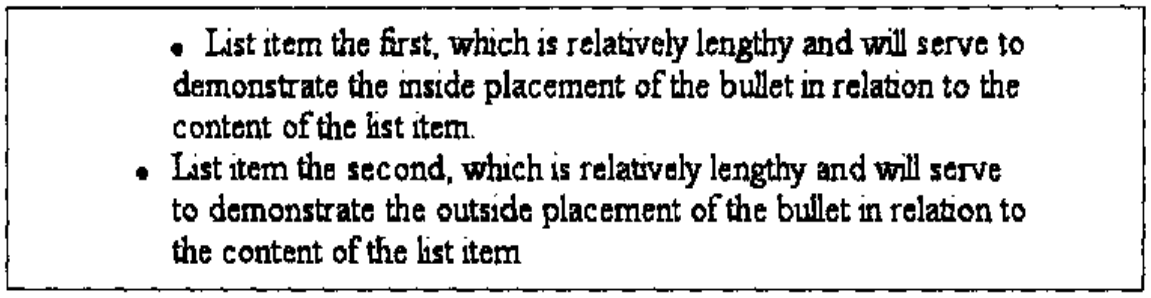


图 7-86 项目符号放于列表项之内与之外

缩略列表样式

为简洁起见，可以组合三个列表样式属性为一个方便使用的属性：list-style。

list-style

允许值	<列表项类型>||<列表项图像>||<列表项位置>
初始值	参考各项
可否继承	是
适用于	列表项元素

例如：

```
LI{ list-style-type :url(sm-ohio.gif)square inside;}
```

如图 7-87 所示，三个属性值均应用在列表项上了。

list-style的值可按任意顺序排列，且它们中的一些可以空缺。只要提供一个值，其他的均可填入缺省值。例如，以下两规则会生成相同的可视效果。

```
LI.norm {list-style: url(img42.gif);}
LI.odd {list-style: url(img42.gif)disc outside;} /* the same thing */
```

list-style按相同的方式覆盖先前的规则。在以下标记中：

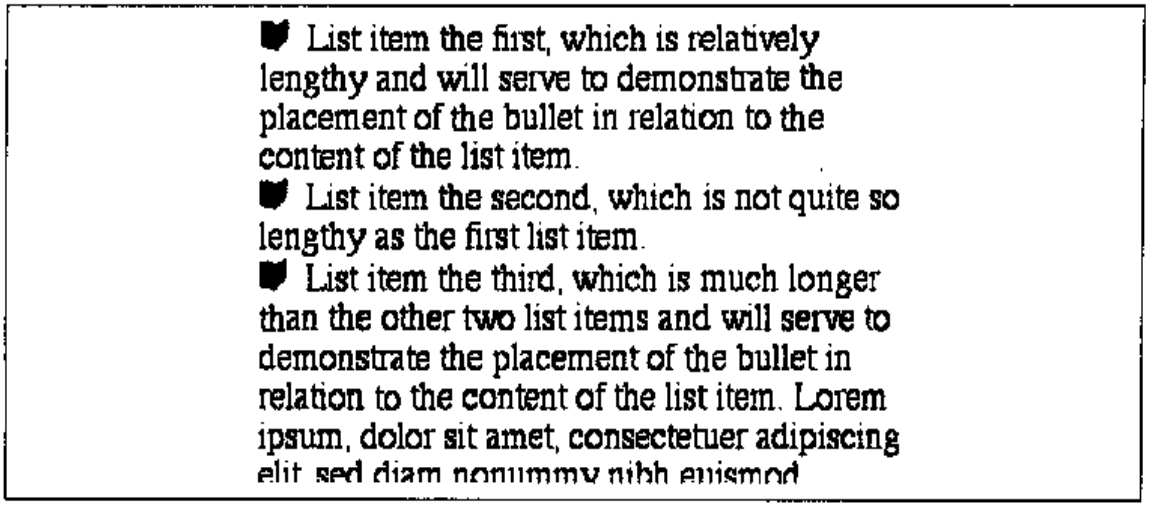


图7-87 把属性合在一起

```
LI {list-style-type: square;}
LI.norm {list-style: url(img42.gif);}
```

使用norm类(class)的列表项都不会使用方块。这是因为规则LI.norm中list-style-type隐式声明的值disc覆盖了先前定义的值square。

小结

能将边界、边框及补白应用于任何元素，是CSS大大优于传统Web标记的原因之一。过去，如要将标题封闭于有颜色的带边框的方块中就意味着要将标题放于表格中。用这种办法来生成如此简单的效果实在是冗余、难看。正是这种能力使得CSS极富竞争力。

在CSS早期的实现中，也有一些支持不好的情况。Explorer 3和Navigator 4对框属性的实现均支持得不够好。现在，情况有了极大的改善，这要归功于Explorer 4、Explorer 5及Opera 3.6(及更新版本)。

从另一方面来说，从CSS的最初实现开始，列表样式就得到了相当好的支持。很多早期的用户代理没有将图像作为项目符号的能力，但显然可支持各种计数的能力。

尽管本章试图提供描述框属性工作的一个清晰视图，但CSS编排格式模型存在着太多细微末节。因此，将这些内容完全包含进来显然会减慢本章的进度。这些将包含于下一章中，该章较倾向于理论，同时还将提供CSS编排格式模型工作的综合解释。

第八章

可视化格式编排

上一章中，我们讲述了很多关于CSS如何处理文档的可视化格式的信息。在那里，我们采取的是实践风格：大量的篇幅用于解释如何工作，只有小部分内容解释为什么。在本章中，我们将主要讲述可视化的理论，很少会涉及实践。

读者也许想问，为什么要用一章的内容来讲述CSS可视绘制的理论基础呢？主要原因是为了包含所有的基础内容。在上一章中，我们提供了尽可能多的不同的例子，但由于CSS中模型的开放性及其功能的强大，任何一本书都不可能讲出所有的属性及效果的可能的组合方式。本书的每个读者都将会为自己的文档效果找到相应的CSS的新方式。

在这样做的过程中，读者也许会遇到用户代理做出看起来很奇怪的行为。当完全掌握CSS中可视绘制模型工作过程后，读者将能够判断某些行为是否是因CSS定义而输出的正确结果（如果是非期望的），或者是否遇到了一个需要报告的错误。（见附录一“CSS资源”，以获得如何报告输出错误的细节）。

基本框

在处理元素时，CSS假定每个元素都生成一个或多个矩形框，叫做元素框（*element box*）。（将来版本的规范也许允许非矩形框，但现在均为矩形框。）每个元素框包

含像核一样的内容区（*content area*）。这个内容区被数量可选的补白、边框及边界包围。它们被认为是可选的，因为均可被设置为零宽度，相当于将它们从元素框中去除。内容区如图 8-1 所示，周围有补白、边界等区域。

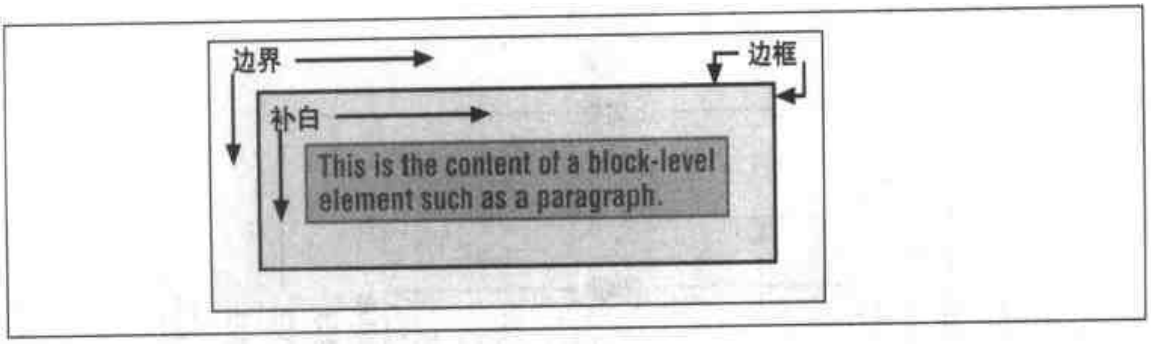


图 8-1　内容区及其周围的补白、边框边界

补白、边界、边框中的每一个都可以设置成不同的属性，如`margin-left`或`border-bottom`。内容的背景（如颜色或图片）也应用于补白，而边界总是透明的，允许任何上级元素的背景为可见。在效果上，边界类似于图片的HSPACE和VSPACE属性，尽管比它们更复杂。补白不能设为负值，而边界却可以。负边界的效果在本章稍后有介绍。

另一方面，边框有其自身的规则。边框使用定义好的样式生成，如`solid`或`inset`，颜色可以使用`border-color`属性来设置。如果没有设置颜色，则边框颜色采用基于元素内容的前景颜色。例如，如果段落文本是白色，则段落周围的边框也是白色，除非网页制作者特殊声明为不同的边框颜色。如果边框样式是间断的类型，则元素的背景在不连续处可见；换句话说，边框有着与内容及补白相同的背景。最后，边框宽度不能为负值。

不同类型的元素在如何格式化上存在着区别。例如，块级元素与内联元素的处理方式不一样，浮动元素则更复杂。让我们顺序来观察各类元素。

块级元素

块级元素，如段落、H1标题、列表及列表元素，其行为方式很有意思，有时可以预测，有时又出人意料。例如沿水平方向与垂直方向处理元素的放置有区别。为

完全理解块级元素是如何处理的，必须清楚地理解大量的边界及区域。如图 8-2 所示。

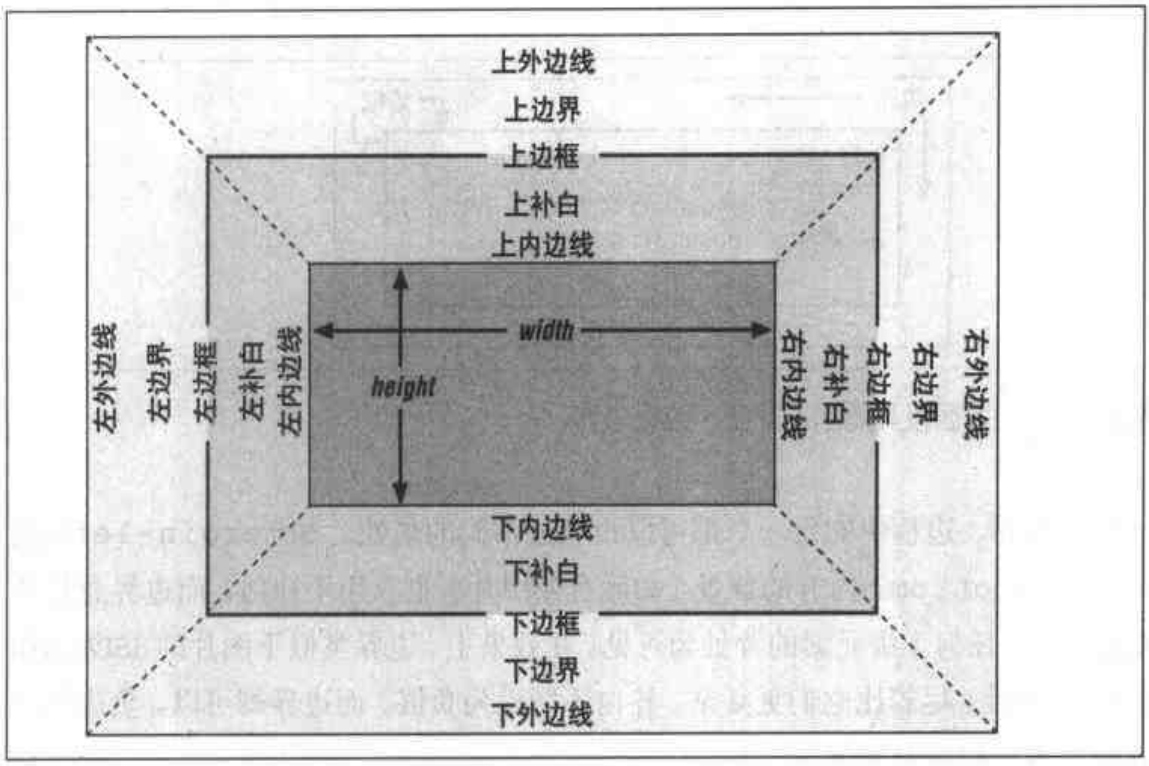


图 8-2 复杂的框模型

通常，元素的width定义为左内边线与右内边线之间的距离，height被定义为上内边线与下内边线之间的距离。这些都是可以应用于元素的属性。

不同的宽度、高度、补白、边界与边框共同决定了文档的布局。多数情况下，宽度与高度自动由浏览器基于可用显示区域及其他因素来决定。当然，在CSS中，可以断言更直接的元素尺寸及显示的控制方式。在水平方向与垂直方向上布局的效果是不同的，因此，我们将分别讨论它们。

垂直方向格式编排

垂直方向的格式编排非常简单，所以我们首先描述它。在前面的章节中已有大量的介绍，因此，在讲述更为复杂的水平方向格式编排之前，先来复习一下重点并探究一些细节。

高度

通常，元素的高度是由其内容决定的。当然，也受其宽度的影响，例如，段落越窄，为容纳其全部的文本（或其他）内容所需的高度越大。

在CSS中，可以为任何块级元素设定一个明确的高度。如果这样做，导致的结果可能不确定。假设指定的高度大于显示内容所需的高度：

```
<P STYLE="height: 10em;">
```

在这种情况下，对额外高度的处理类似于额外的补白，如图8-3所示。

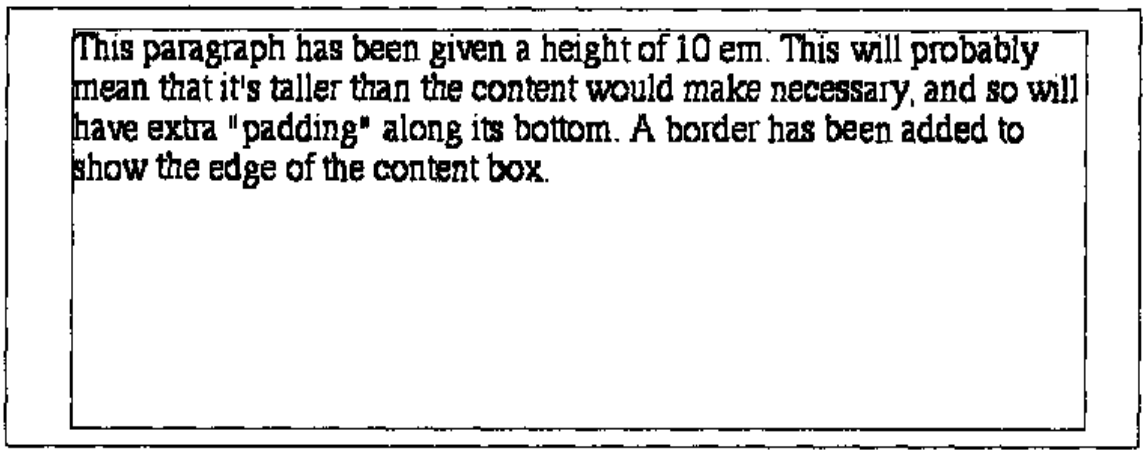


图8-3 为块级元素设定高度

另一方面，假如height比显示内容所需的值小：

```
<P STYLE="height: 3em">
```

浏览器会提供一种在不增大高度的前提下显示全部内容的方式。很可能会为元素增加一个滚动条，如图8-4所示。

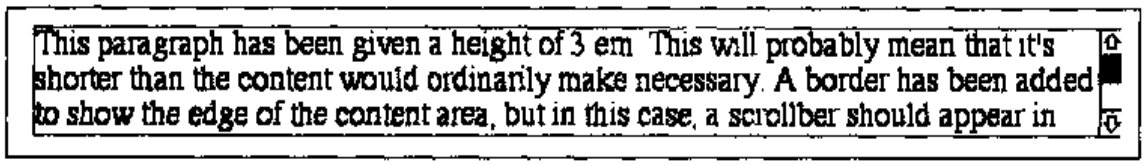


图8-4 处理设定高度低于元素高度的一种方式

在实际应用中，多数浏览器不是这样做的。它们会简单地增大元素的高度，如同

height的值被设为auto一样。这在CSS1中是允许的，CSS1声明如果一个元素不是替换元素，如图像，则可以忽略height的任何值而使用auto。在CSS2中，可以设置将滚动条应用于元素的情况，比如用于段落。

也有可能将块级元素的上边界及下边界设为auto。如果这两个属性之一被设为auto，那么，它将被重设为0（zero），效果即是从元素框中去除了任何上边界或下边界，如图8-5所示。段落边框之间缺少距离是auto被重解释为0的结果：

```
P {margin-top: auto; margin-bottom: auto;}
```

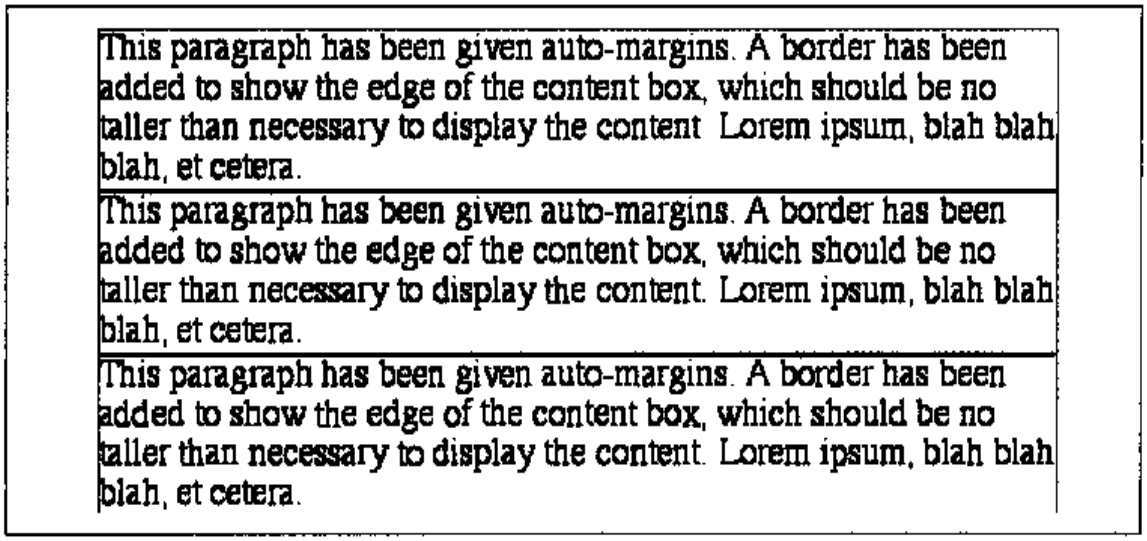


图8-5 自动将边界设成0

压缩垂直边界

垂直格式编排还有一个重要的特征，那就是相邻边界的压缩。这种情况发生于布局中两相邻元素均声明边界时。在前面的章节中曾有过描述，使用的例子是：

```
LI {margin-top: 10px; margin-bottom: 20px;}
```

补白与边框没有压缩现象。如果均未声明，则都缺省为0（zero），此时假定没有设置边框样式。如果设了边框样式，则border-width缺省为medium，而不是0。medium的确切宽度取决于用户代理的编程，但通常为两个或三个像素。

水平方向格式编排

与垂直方向格式编排相反，水平方向格式编排则有些复杂。不过，它开始时很简单，只是把各类元素组合起来使用时才变得复杂。

首先，与垂直边界不同，水平方向上的简单规则是水平边界不压缩。如果将两个块级元素水平相邻放置，且每个元素均有边界，则边界不被压缩。最简单的显示该原则的方式是为两个图像设置边界，然后让它们在同一行中显示，如图8-6所示：

```
<IMG SRC="test1.gif" style="margin:5px;" ALT="first test">
<IMG SRC="test2.gif" style="margin:5px;" ALT="second test">
```

（注意图8-6中的图像实际上是内联元素，但从效果上显示了水平方向相邻的边界不压缩。）

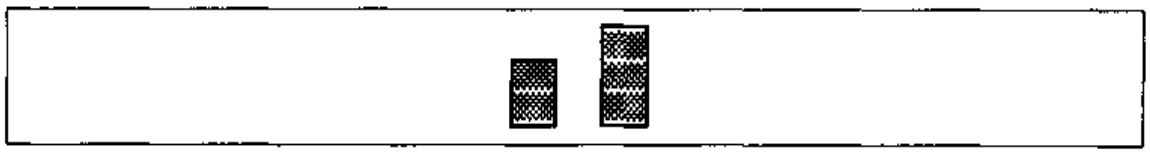


图8-6　水平方向边界不压缩

同样简单的是：非浮动块级元素框各水平组成部分的宽度和等于其父元素的宽度。以DIV中的两个段落为例，其边界被设为1em。内容区宽度（width值）加上左右的补白、边框、边界，总和等于DIV内容区的width，如图8-7所示。

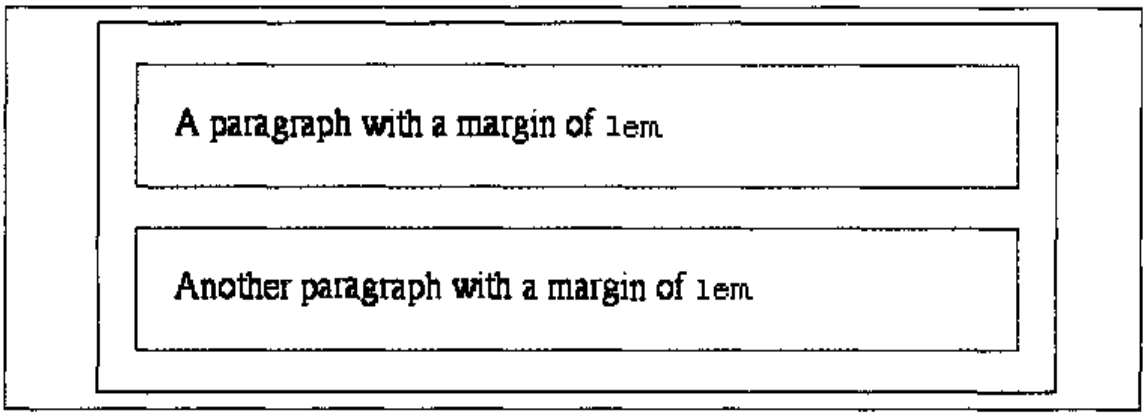


图8-7　元素框与父元素有相同的宽度

因此，如果DIV的width为30em，则每个段落的内容区宽度、补白、边框及边界的总和为30em。在图8-7中，段落周围的“空白”空间实际上是它们的边界。(如果DIV有补白，还会有更多的空白区，本例情况不是那样。)

类似地，列表项元素框的宽度总和等于包含它的列表元素内容区的宽度。如图8-8所示，父元素的边界会影响其子元素的布局。

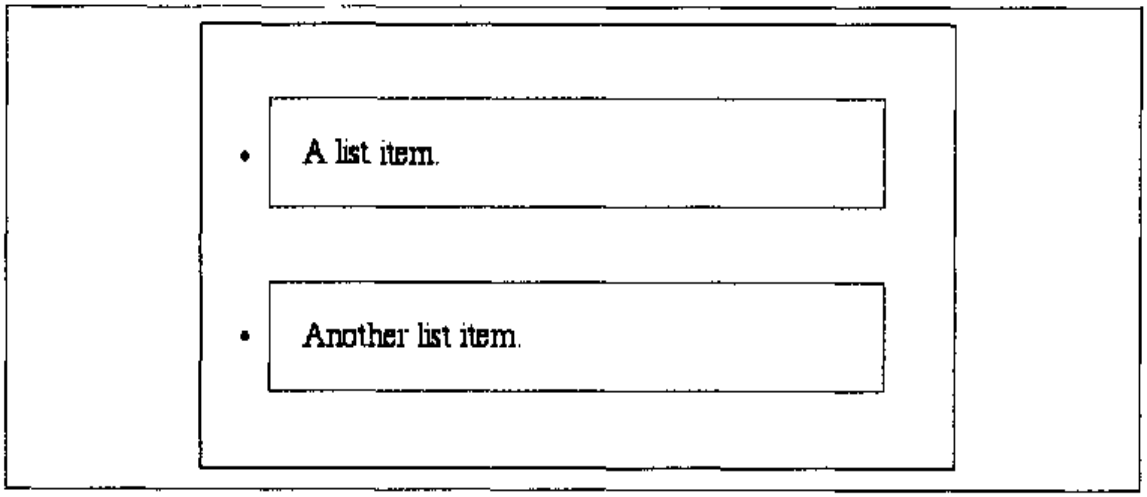


图8-8 列表项的总宽度等于UL元素的宽度

水平方向属性

有很多与框布局有关的属性，即水平方向格式编排的“七属性”(从左开始)：margin-left，border-left，padding-left，width，padding-right，width，padding-right，border-right和margin-right。如图8-9所示。这七个属性必须与父元素的width值相同。

这七个属性中只有三个属性可被设为auto:元素内容区的width，及左边界、右边界。而左补白、右补白、左边框、右边框必须设置为特定的值，否则会缺省为0宽度(假如未声明border-style；如果有一个被设置，则边框宽度设为非确定的medium值)。图8-10提供了框中哪部分可以设为auto值而哪些不能的图示。

width必须设为auto或某种类型的非负值。CSS还允许浏览器为width设一个最小值，块级元素的width不能比它更小。不同浏览器的最小值可能会有所不同，因为规范中未定义它。

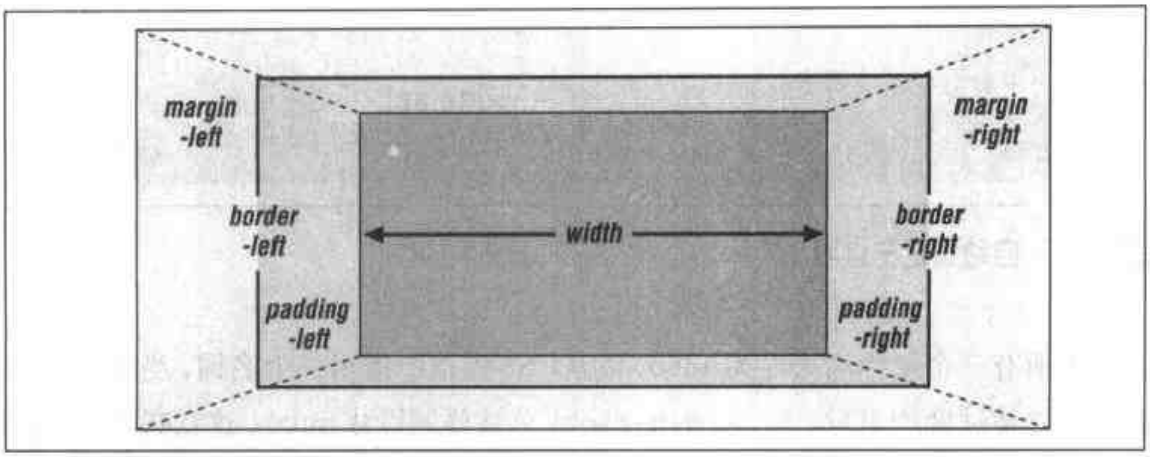


图 8-9 水平格式编排的七属性

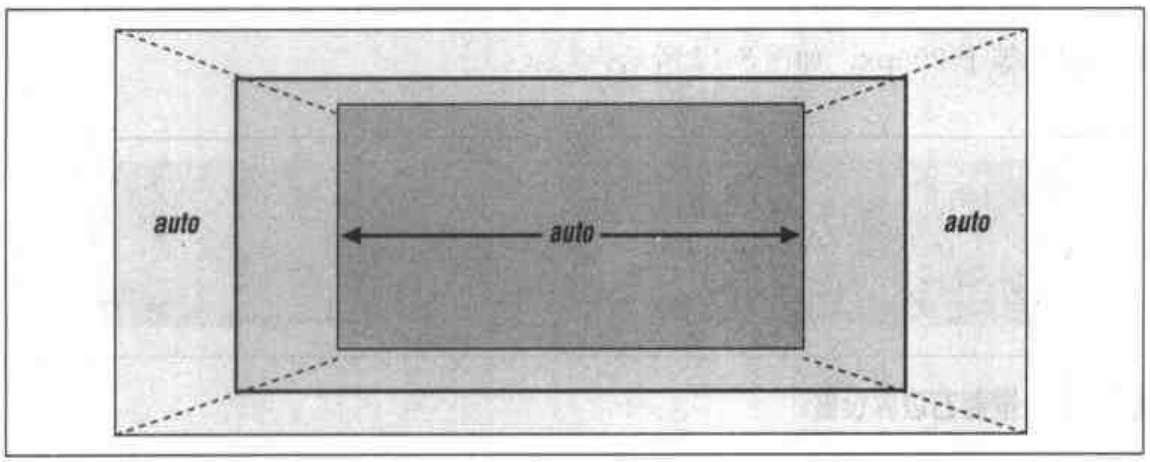


图 8-10 可设为 auto 的水平方向属性

使用 auto

如果 width、margin-left 和margin-right 中只有一个设定为 auto，而其他的值均被指定，则设为auto属性的值等于使元素框宽度等于父元素宽度所需的值。因此，如果七属值的和必须等于 400 像素，且未设置补白和边框，右边界宽度设为 100 像素，而左边界设为 auto，则左边界将为 200 像素宽：

```
P {margin-left: auto; margin-right: 100px; width: 100px;}
```

结果如图 8-11 所示。

从某种意义上，auto可说明为："弥补其他部分与所要求的总和之间的差别。"然则，如果三个属性均设为 100px，没有设置为 auto 的属性，会有什么发生呢？

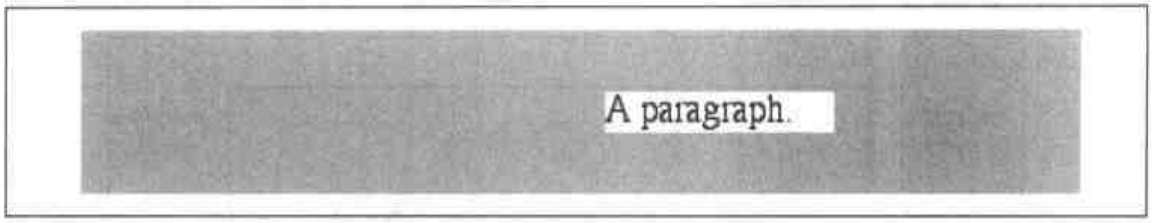


图 8-11 自动设定左边界

在这种所有三个属性均未设为auto，或从CSS规范中借用一个名词，当这些格式编排属性被过紧约束时，则margin-right总被强制设成auto。这意味着如果两个边界及宽度均设为100px，则右边界会被用户代理设为auto:

```
P {margin-left: 100px; margin-right: 100px; width: 100px;}
```

则右边界等于200px，如图8-12所示。

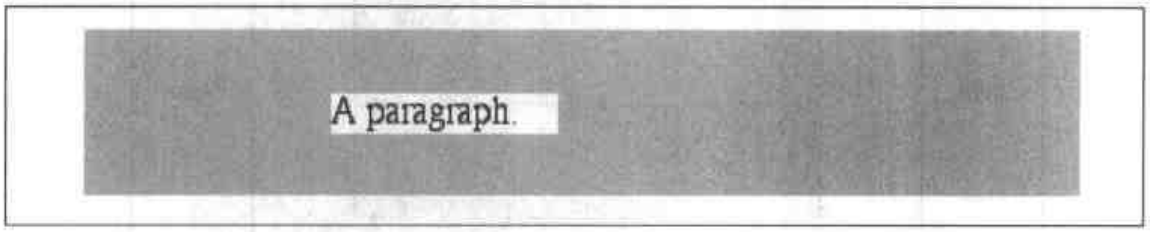


图 8-12 覆盖右边界设置

注意：注意仅对从左到右阅读的语言，如英语来说，margin-right必须强制为auto。在从右到左阅读的语言中，情况则刚好相反，也即，margin-left被强制为auto，而不是margin-right。这一点，在CSS1中不成其为问题，但因为CSS2中引入了相关的书写方向属性，所以必须注意这一点。

如果两个边界都被显式声明，而width为auto，那么，width的值将被设置成一个值，以达到所需要的总宽度（也就是说，父元素内容的宽度）。下面这段标记的显示如图8-13所示:

```
P {margin-left: 100px; margin-right: 100px; width: auto;}
```

这种情况非常常见，事实上，它与设置边界而不声明width的效果相同。下面这段标记的显示一样如图8-13所示:

```
P {margin-left: 100px; margin-right: 100px;} /* same as before */
```

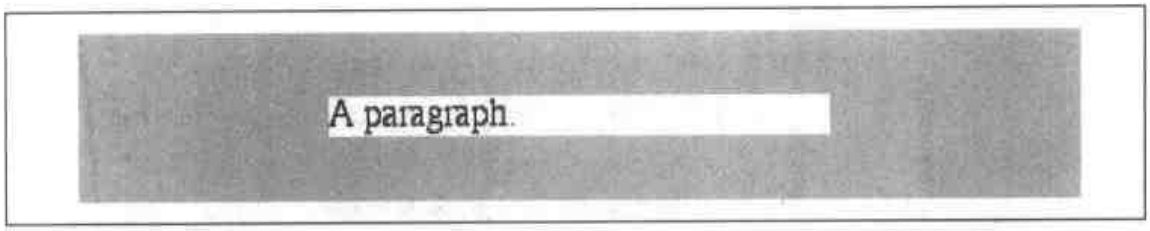


图 8-13 自动设置宽度

警告：在实际应用中，只有 1999 年或更新发布的浏览器能够正确处理 auto，且不是其中所有的浏览器均可以。那些不能正确处理 auto 边界的浏览器会采取不一致的行为，但最为可能的是它们将两个边界均设为 0。能够正确处理的是用于 Macintosh 的 Internet Explorer 4.5，Internet Explorer 5 及 Opera 3.6。

多于一个的 auto

现在来考虑一下三个属性值中有两个设为 auto 的情况。如果两个边界均设为 auto，则它们被设成相等的值，这样使元素在其父元素中居中，如图 8-14 所示：

```
P {width: 100px; margin-left: auto; margin-right: auto;}
```

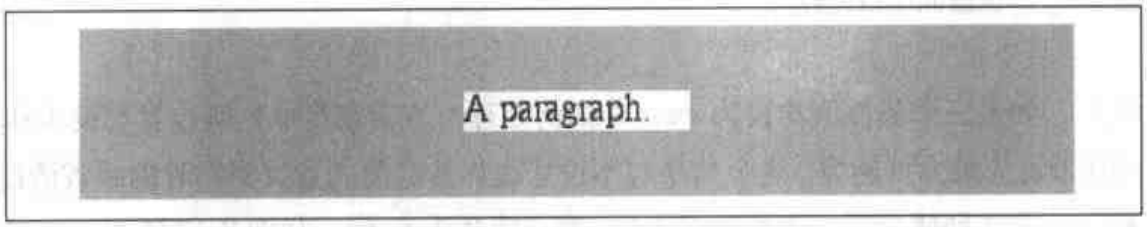


图 8-14 设置显式宽度

实际上，这是将块级元素居中的正确方式。text-align只能应用于块级元素的内联内容，因此设置元素的 text-align值为 center 不会使它居中。应当声明：

```
P {margin-left: auto; margin-right: auto; width: 50%;}
```

这将使所有段落在其父元素中居中，如图 8-15 所示。

警告：只有 Macintosh 的 Internet Explorer 4.5，Internet Explorer 5 和 Opera 3.6 可以用 auto 边界设定使元素居中。

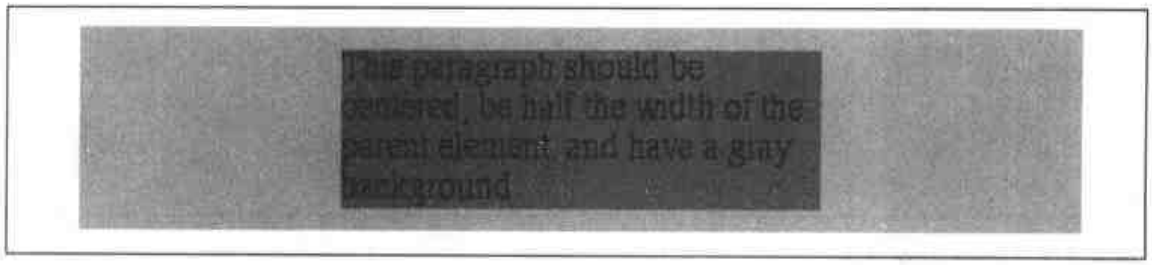


图 8-15 用自动边界使元素居中

另一种可能的情形是两边界之一及宽度被设定为auto。在这种情况下，设为auto的边界降为0:

```
P {width: auto; margin-left: auto; margin-right: 100px;}
```

然后width被设为达到总要求所需的值，如图 8-16 所示。

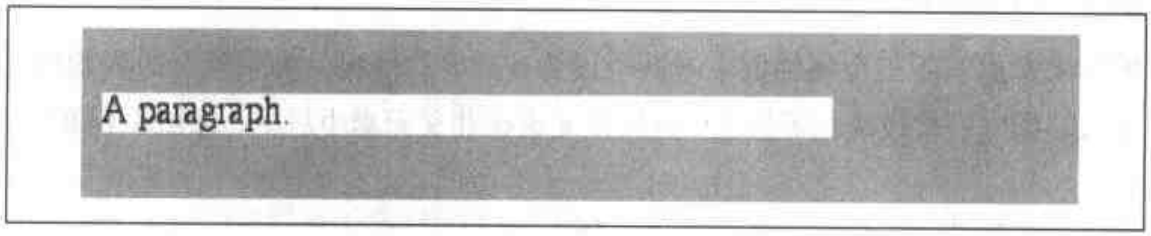


图 8-16 设置显示右边界

最后，如果三个属性值均设为auto，如何处理？答案很简单：两边界均设为0，width设为最大可能值。这个结果与不设置边界及宽度的显式声明时的缺省情况相同。在这种情况下，边界缺省为0，宽度缺省为auto。如图 8-17 所示。

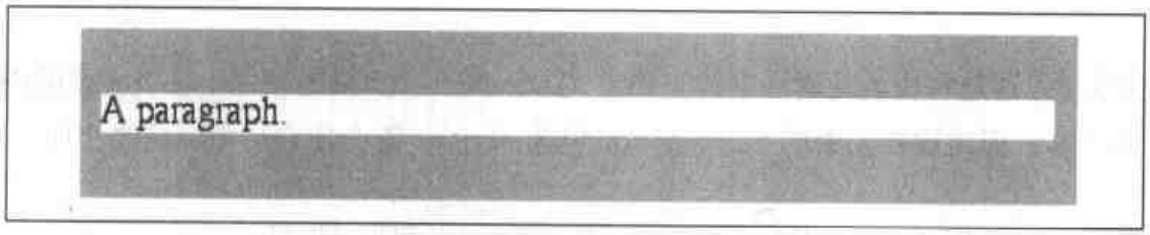


图 8-17 各属性均设为auto

注意既然水平方向上边界不压缩，父元素的补白、边框、边界会影响其子元素。当然，这种影响不是直接的，影响的方式是元素边界（或其他）为其子元素引入了一个位移。垂直边界仍然压缩，如图 8-18 所示:

```
DIV {margin: 20px; padding: 20px;}
P {margin: 10px; padding: 10px}
```

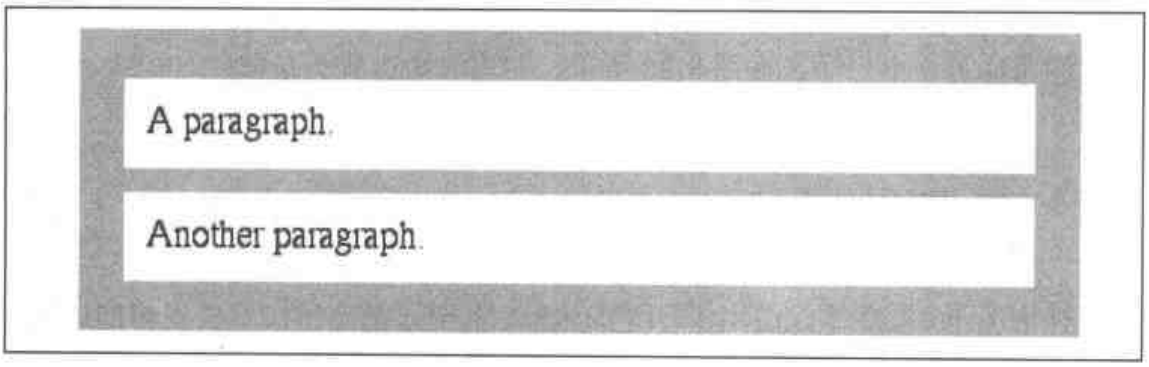


图 8-18 在父元素的边界和补白中隐式声明位移

负边界

现在，这些可能看起来都很简单，你可能会困惑，我怎么会认为它复杂呢？使边界变得复杂的原因是：它可以有负值。

你可能还记得前面介绍过的水平格式的第二条简单的规则是：水平方向7个属性的总和总是等于父元素的width。初看起来，我们可以认为这意味着元素的宽度永远不会超过它的父元素的内容区——而且当所有的元素都是0或者比0大时，确实是这样的。然而，考虑下面这种情况，如图 8-19 所示：

```
DIV {width: 400px; border: 3px solid black}
P.wide {margin-left: 10px; width: auto; margin-right: -50px;
  border: 1px solid gray;}
```

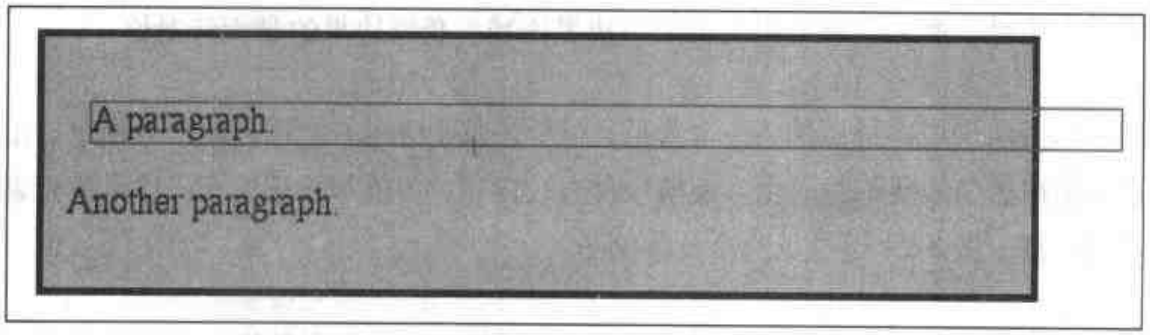


图 8-19 通过负数边界生成更宽的子元素

是的，子元素比其父元素还要宽！这在数学上来说是正确的：10px + 0 + 0 + 450px + 0 + 0 - 50px = 410px。尽管这会导致一个子元素从其父元素中突出出来，

但从技术上来说不违反规范，因为七个属性的值加起来等于总要求。这从语义上来说是一种托辞，但却是一种有效的行为。

让我们考虑另一个例子，如图 8-20 所示，左边界被设置为负值：

```
DIV {width: 400px; border: 1px solid black;}
P.wide {margin-left: -50px; width: auto; margin-right: 10px;
  border: 3px solid gray;}
```

在这种情况下，段落不仅溢出了DIV的边框，甚至还超出了浏览器视窗的边线！

注意：记住补白、边框及内容区宽度均不能设为负数，只有边界可以小于0。

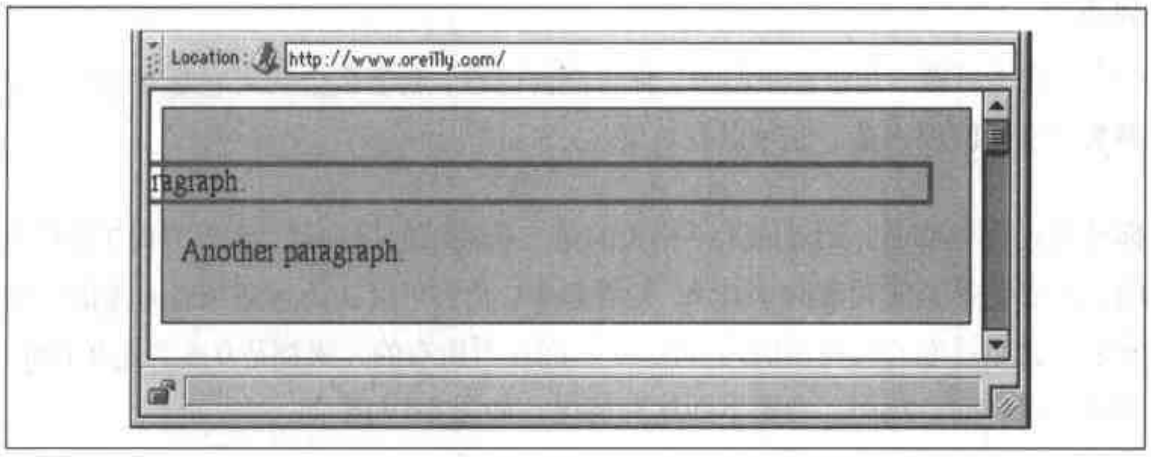


图 8-20　设置负的左边界

负边界对垂直格式编排也存在着冲击，它可影响边界是如何压缩的。如果存在负值垂直边界，则浏览器会用最大的正边界值减去负值边界的最大绝对值。

如果只有两个边界待压缩，一个为正，一个为负，情况的处理方式十分简单。正边界值减去负边界的绝对值，或换一种方式描述，负值加到正值上，结果是元素间的距离。图 8-21 提供了两个具体例子。

可以注意到负上边界与负下边界的“拉动”效果。这与使用负值水平边界使得一个元素伸出父元素的方式相同。考虑以下标记：

```
DIV {width: 400px; border: 1px solid black;}
```

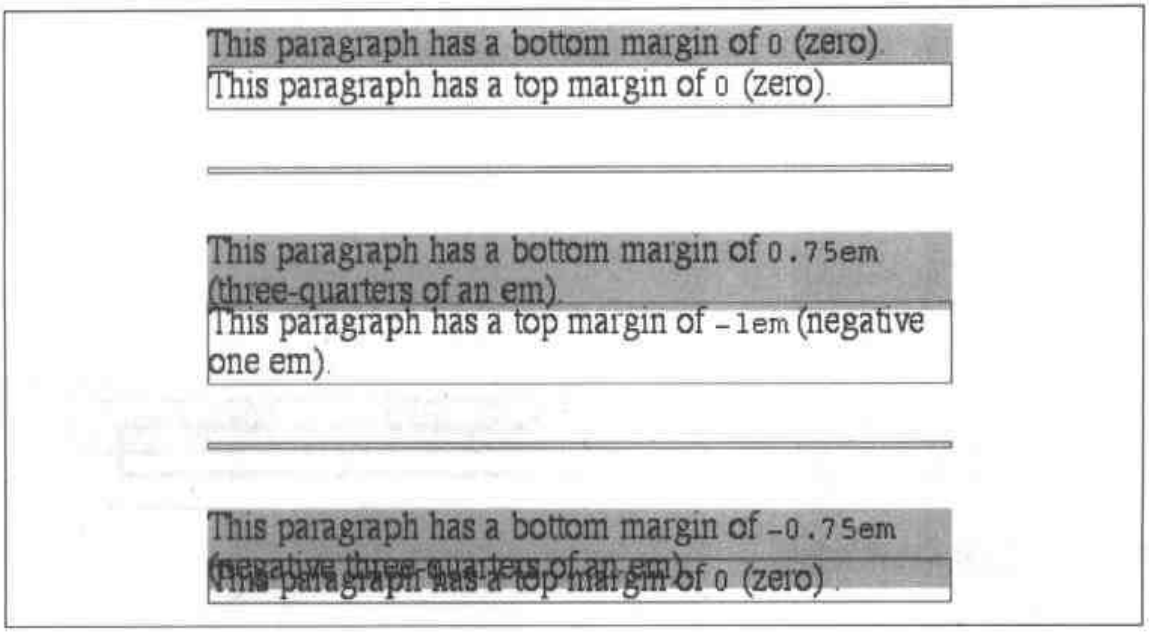


图 8-21 负值垂直边界的例子

```
P.neg {margin-top: -50px; width: auto; margin-right: 10px;
  margin-left: 10px; border: 3px solid gray;}

<DIV STYLE="width: 420px; background-color: silver;
   padding: 10px; margin-top: 75px;">
<P CLASSS="neg">
A paragraph.
</P>
</DIV>
```

如图 8-22 所示，段落因负值上边界被向上拉，因此它处于父元素 DIV 之外!

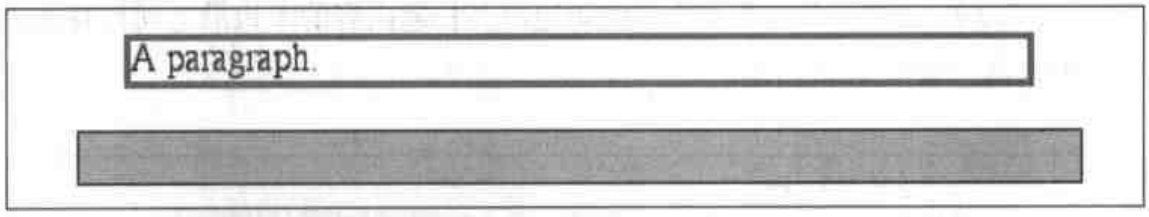


图 8-22 负值上边界的效果

使用负值下边界，则看起来段落之下的内容被向上拉。比较以下标记与图 8-23:

```
DIV {border: 1px solid black;}
P.neg {margin-bottom: -50px; width: auto; margin-right: 10px;
  margin-left: 10px; border: 3px solid gray;}
```

```
<DIV STYLE="width: 420px; background-color: silver;
   padding: 10px; margin-top: 75px;">
<P CLASS="neg">
A paragraph.
</P>
</DIV>
<P>
The next paragraph.
</P>
```

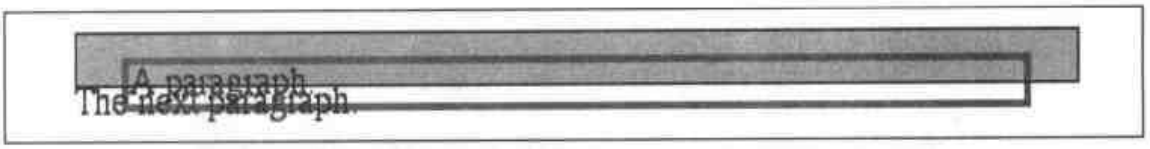


图 8-23 负值下边界的效果

图 8-23 中所发生的是跟在 DIV 后面的元素根据 DIV 的下边沿放置。如同所看到的那样，DIV 结束于其子段落可视下边沿的上面。DIV 后面的元素始于距离 DIV 下边沿的恰当距离处。段落重叠的事实并不重要，至少在技术上是这样的。

接下来是一个列表项边界的例子，一个无序列表及一个标题均被压缩。这种情况下，无序列表及标题都设为负值边界：

```
LI {margin-bottom: 20px;}
UL {margin-bottom: -15px;}
H1 {margin-top: -18px;}
```

两个负边界中较大的那一个(-18px)加到最大正边界(20px)上，生成结果(20px-18px = 2px)。这样，在列表项内容的下边沿与标题内容的上边沿之间只有两个像素的距离。如图 8-24 所示。

图 8-24 压缩边界与负值边界

有一个未解决的行为，即：如果元素因负值边界而重叠，哪个元素“位于顶部”？

注意本节中很少有例子为所有元素使用背景颜色。这是因为如果那样做，内容可能会被其后面元素的背景颜色所重写。

CSS规范中没有规定这种形式的元素重叠发生时会发生些什么，而是交由实现者决定。有争论的是关于所有前景内容总是显示于所有背景内容的“前面”，浮动元素的行为看上去支持这种解释。另一方面，CSS2的z-index属性使这个推理变得复杂。在本书写成之时，应用上仍没有足够进展以明晰地检测这一点，且CSS2中z-index的描述实际上未给这个论题带来解决办法。

最后，如果使用负边界，在所有浏览器中得到的结果可能会不同。由于没有人可清楚地断定哪个是正确的，因此没有哪一个可被当成错误，至少在规范足够清晰之前不能。

列表项

谈及列表项，它们同所讨论过的元素相比，多了一些特殊的规则。典型的列表项跟于一个标记之后，例如一个点或一个数字。这个标记不是列表项内容区的部分，因此，可得到如图8-25所示的效果。

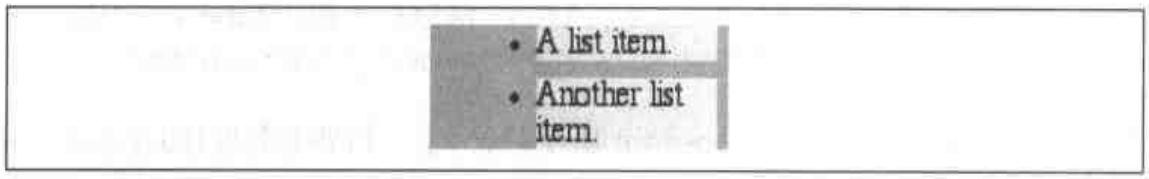


图8-25 列表项内容

在CSS1中，很少谈及文档布局中标记的放置及效果。CSS2引入专门设计的属性来论述这个问题。比如marker-offset。由于该属性及类似属性现今仍未被广泛支持，不在这里展开论述。在第十章“CSS2展望”中，有关于标记属性的简短讨论。

块级替换元素

块级替换元素也是格式编排处理存在差异的原因。最重要的是假定替换元素有实

际的高度与宽度，例如，一个图像有一定的像素宽、高。在这个假定下，如果替换元素的height或width被设为auto，则值总等于元素实际的高度或宽度。这样，如果一个图像宽为150像素，且它的width属性被设为auto，则它的width将等于150px，如图8-26所示：

```
IMG {display: block; width: auto;}
```

图8-26 width为auto的替换元素按其实际大小绘制

替换元素的height和width可以设置、而不使用auto或其实际大小。这通常用于“伸缩”图像，或放大、或缩小。这样，如果图像实际为150px宽，且被设为75px，则图像会以外观宽度的一半来显示。在大多数浏览器中，高度也会随之按比例变化，除非被显式设定为某值。图8-27显示了几种可能。

也可以使用height来缩放图像（或其他替换元素）：

```
<IMG SRC-"test.gif" STYLE="display: block;" ALT="test image">
<IMG SRC="test.gif" STYLE="display: block; height: 50px;" ALT="test image">
<IMG SRC="test.gif" STYLE="display: block; height: 200px;" ALT="test image">
```

这与HTML中对IMG标签使用height属性完全一致。如果图像为100像素高，则缺省情况下其高度为100px。若特别指定为另一个值，则图像会适当地缩放，如图8-28所示。

在几乎其他所有方式中，块级替换元素在格式编排上与块级元素相同：垂直边界压缩而水平边界不压缩，边框与补白在未显式声明时缺省为0。记住，并非所有的替换元素都是图像。例如，大多数表格也是替换元素。

通常，所有的替换元素（块级及其他）可以使用width和height缩放。在其他方式中，内联替换元素的处理有很大区别，如本章稍后所述。

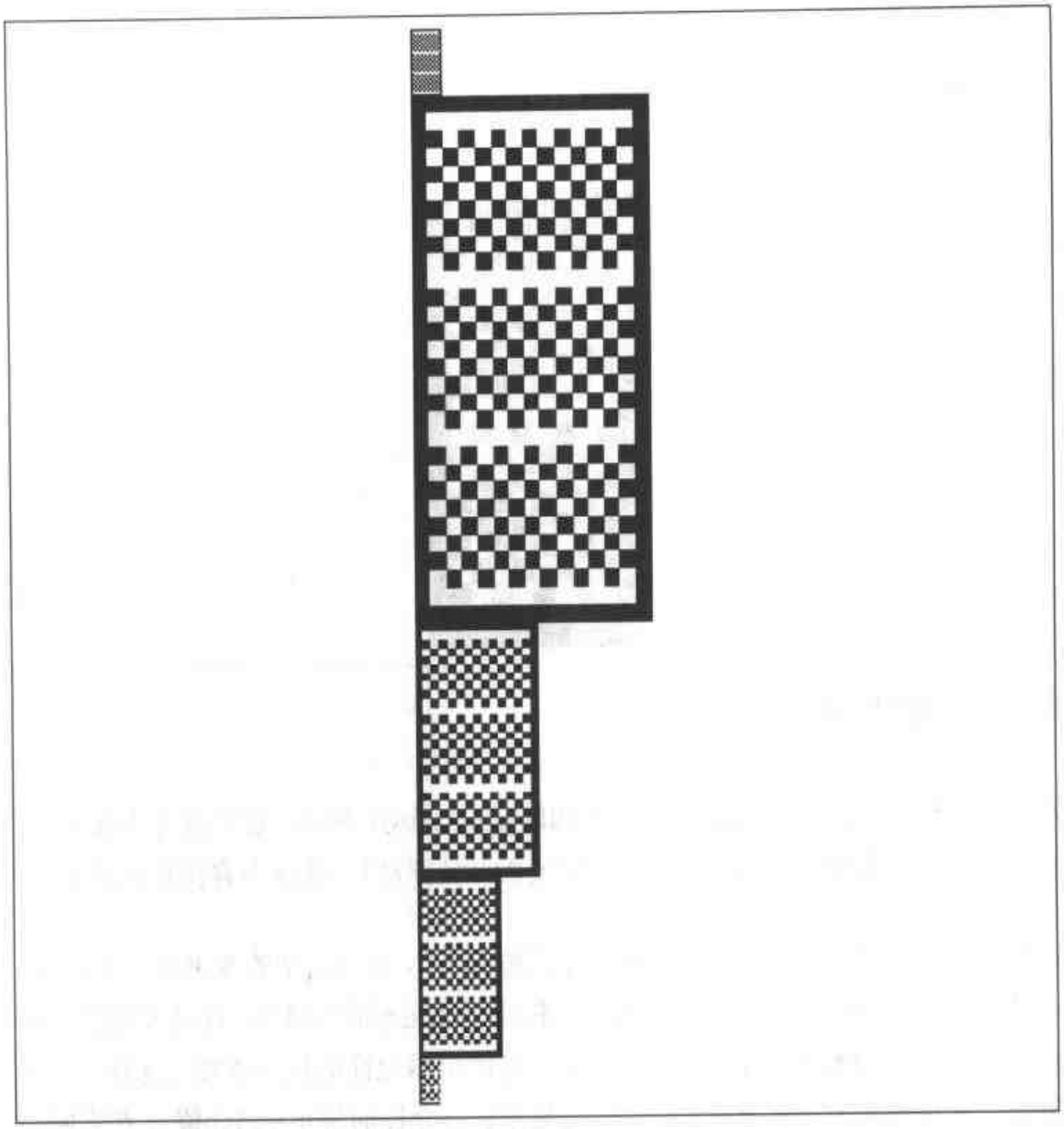


图 8-27　使用 width 属性缩放图像

浮动元素

如上一章所讲述，CSS 允许任何元素浮动，不管是图像、段落，还是列表。但这并非没有代价：浮动元素引入了自身的特殊性。浮动元素在决定文档流方面有不同之处。例如，由其他元素生成的框按浮动元素不存在绘制，但这些元素内容的绘制则必须把浮动元素考虑在内。这就影响了元素框的生成，意味着浮动对这些框存在着间接影响。

一些细节有助于解释这个细节，一个浮动元素成为块级元素，不论其先前是什么

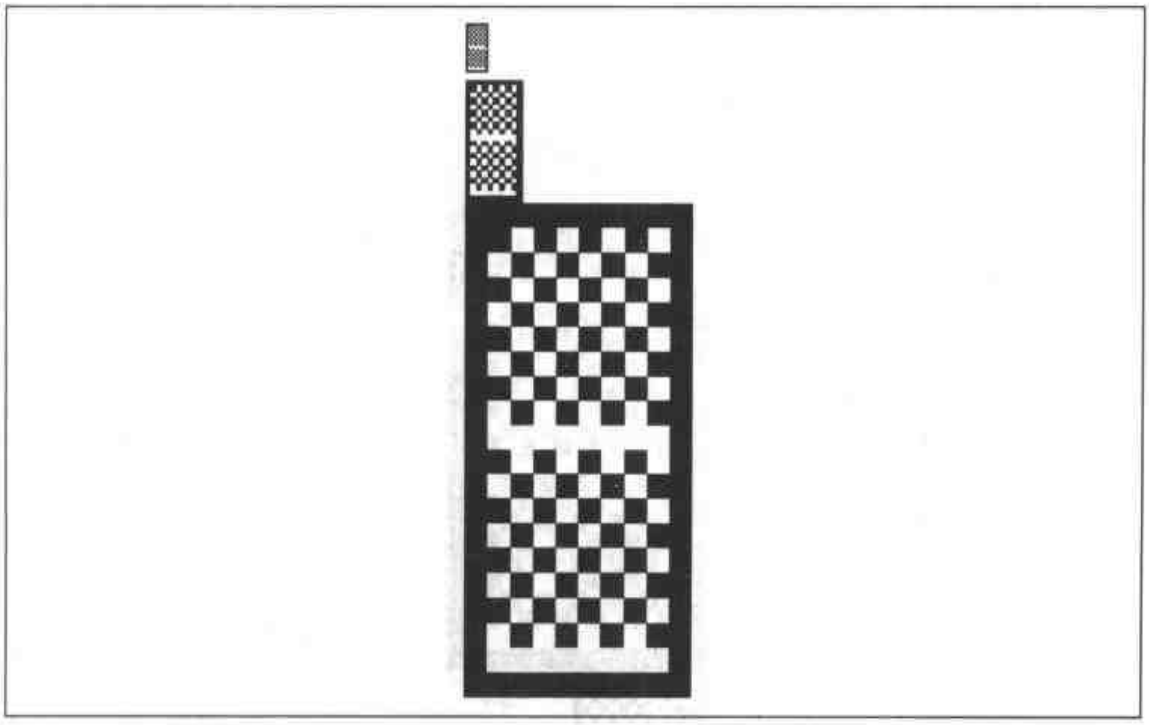


图 8-28 使用 height 属性缩放图像

状态。这样，如果一个图像（通常作为内联元素）浮动，那么，它将成为块级元素。这个块级状态有助于解释为什么当一个元素开始浮动时，其他内容在其周围流动。

记住如果一个文本元素浮动，它的宽度将趋向于0。这与通常的水平规则正相反，水平规则中 width 增大，直至七属性的和等于父元素的 width。浮动元素的宽度缺省为 auto，这样就缺省为0，然后增大到浏览器允许的最小宽度。这样，一个段落文本最小可为一个字符宽（假设这是浏览器允许的最小width值）。在实际应用中，更为可能的是浏览器使浮动元素的宽度为元素中最长的单词宽度，如图 8-29 所示。

This is a floated paragraph

Lorem ipsum, dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis misl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

图 8-29 浮动元素宽度倾向于 0

浮动：细节

一系列特定的规则控制着浮动元素的布置。这些规则与控制补白、宽度计算的规则非常相似，且有着通常意义下的初始外观。它们是：

1. 浮动元素的左（右）外边沿也许不位于其父元素的左（右）内边沿。

这一点非常直观。左浮动元素的外左边沿向左最远可到其父元素的左内边沿；类似地，右浮动元素的外右边沿向右最远可到其父元素的右内边沿，如图8-30所示（在这个及接下来的图中，圆圈数字显示了标记元素在资源中出现的位置，带数字的框显了浮动可视元素的位置及大小。）

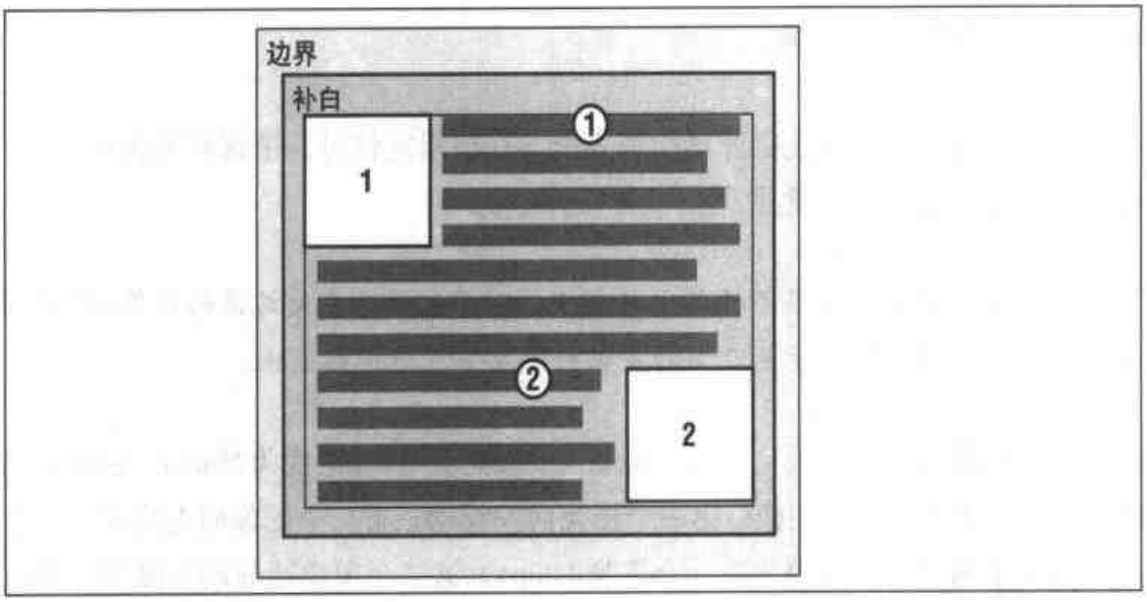


图8-30 向左（1）或向右（2）浮动

2. 浮动元素的左（右）外边沿必须在先出现于文档资源的浮动元素的右（左）外边沿的右（左）侧，除非后一个元素的顶端低于前一个元素的底端。

这条规则可防止浮动元素相互重写。如果一个元素向左浮动，且已经因为早出现于文档资源而存在浮动元素，则后一个元素靠着前一个的右边沿放置。如果一个浮动元素的顶端低于所有先出现的浮动元素的底端，则可以完全浮动到父元素的左内边沿。这样的例子显示于图8-31中。

这条规则的优点在于，既然不需担心一个浮动元素遮盖住另一个，则可以保证所

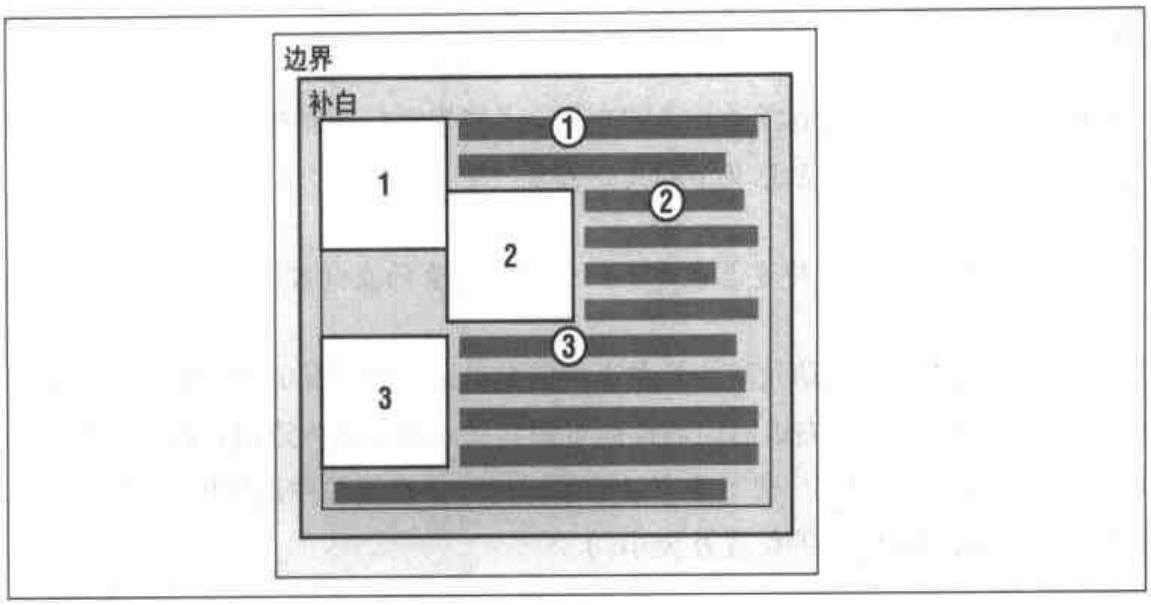


图 8-31 防止浮动元素重叠

有的浮动元素可见。这使得浮动相当安全。在使用定位时，情况将大大地不同，很容易使得元素间相互重叠。

3. 左浮动元素的右外边沿不能位于其右侧的右浮动元素左处边沿的右侧，右浮动元素的左外边沿不能位于其左侧的左漂移元素右外边沿的左侧。

这条规则也是为了防止浮动元素相互重叠。假定有BODY宽度为500px，它的全部内容是两个300px宽的图像。第一个图像向左浮动，第二个图像向右浮动。这条规则防止了第二个图像覆盖第一个图像100px。第二个图像被强制下降只至其顶端低于左浮动元素的底端，如图8-32所示。

4. 浮动元素的顶端不能高于父元素的内顶端。

这是一条简单的规则。这条规则可防止浮动元素沿多种路径浮动到文档顶部。正确的行为见图8-33。

5. 浮动元素的顶端不能高于先于它出现的浮动元素或段落的顶端。

同规则4类似，这条规则可防止浮动元素沿多种路径浮动到其父元素的顶部。这样，如果DIV的第一个子元素是一个段落，然后是又一个段落，浮动元素的顶部不能高于先于它出现的浮动元素的顶端。图8-34显示了这些。

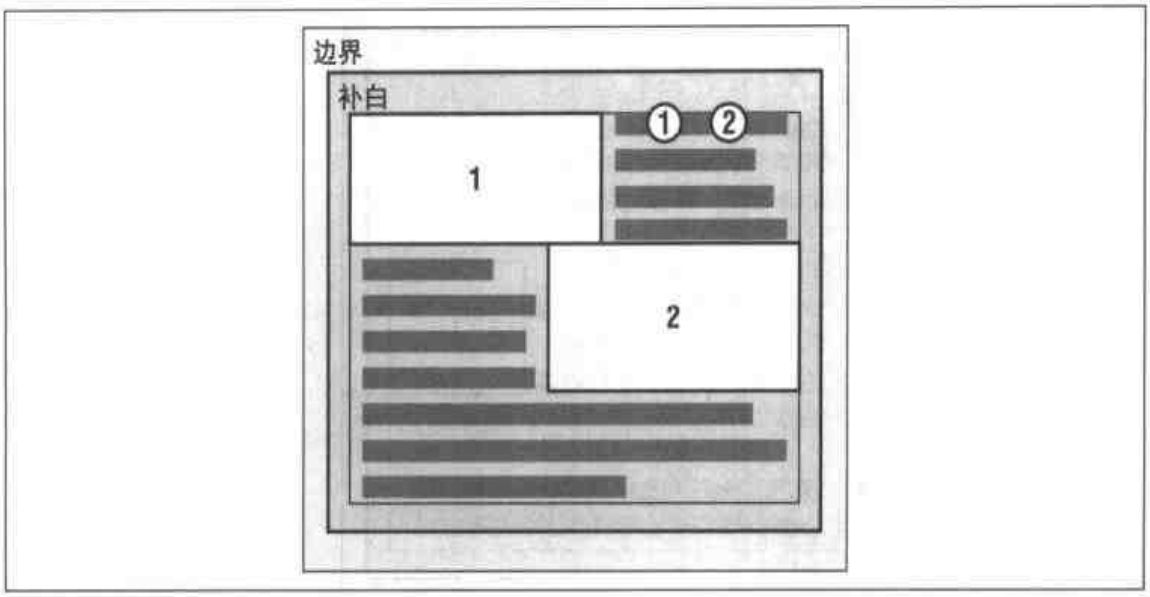


图 8-32 进一步防止重叠

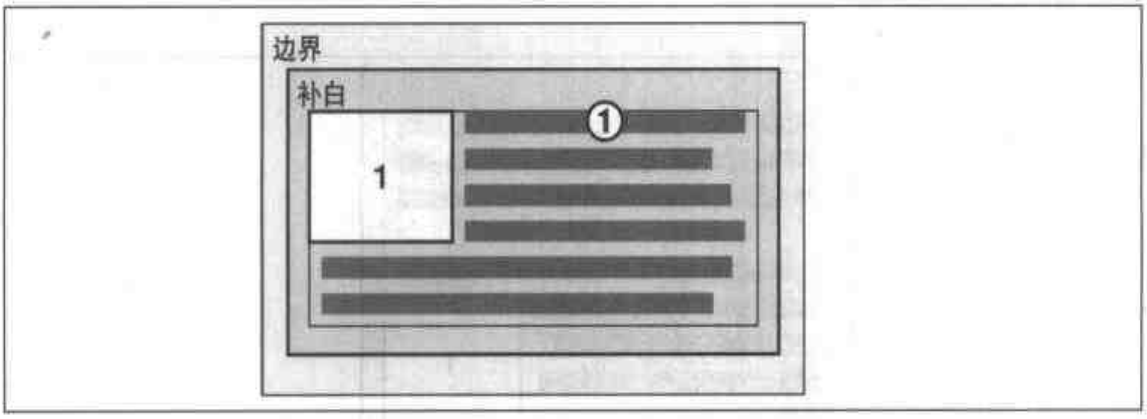


图 8-33 不像汽球，浮动元素不能向上浮动

6. 浮动元素的顶端不能高于行框的顶端，如果行框中有先于浮动元素的内容。

同规则 4 和规则 5 类似，这条规则进一步限制了元素的向上浮动。它不允许元素高于含有先于浮动元素的内容的行。假定在段落的正中有一个浮动图像，浮动元素顶端最高可到生成该图像的和框的顶部。如图 8-35 所示。

7. 一个向左（右）浮动的元素，如果其左（右）侧仍存在浮动元素，则不能够使它的右（左）外边沿处于容纳它的右（左）外边沿的右（左）侧。

换句话说，一个浮动元素不能伸出其容纳元素的边线，除非它的宽度太大而不能被容纳。这可以防止在水平行上相继出现浮动元素，而远远超出容纳块的边沿。

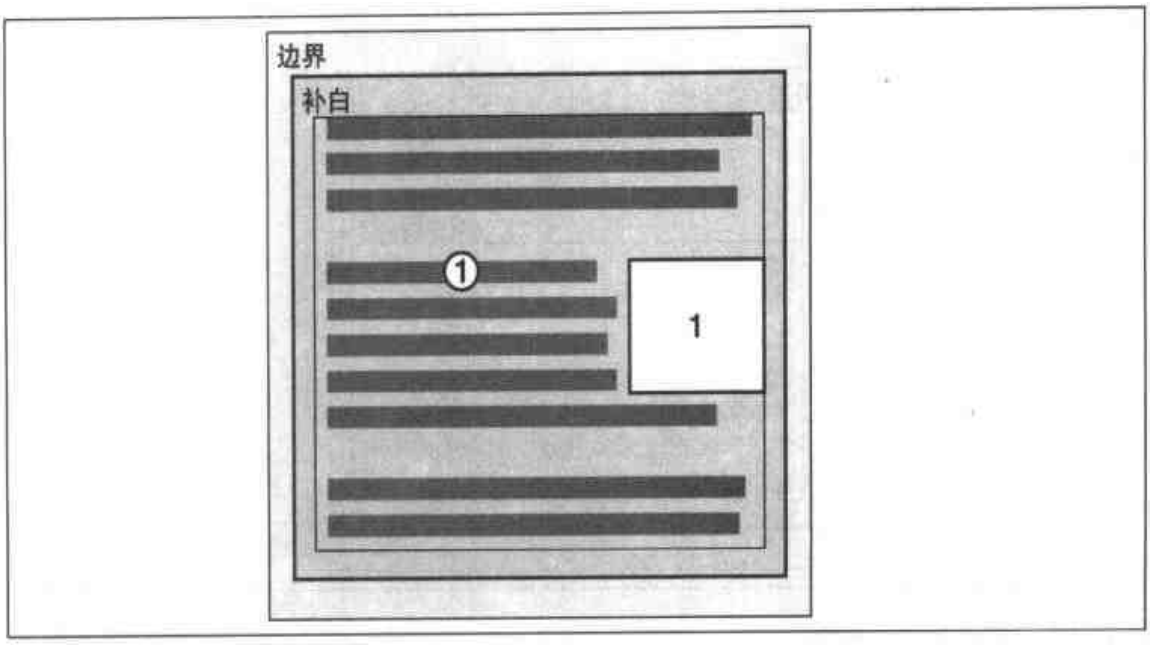


图 8-34 使浮动低于其父元素

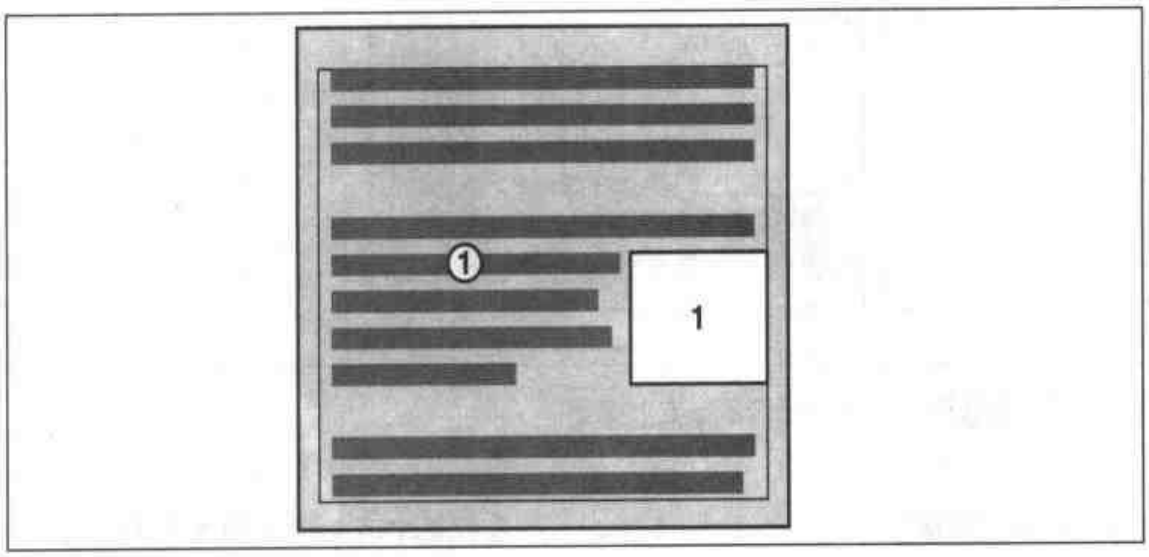


图 8-35 根据关联关系保持浮动级别

如果一个浮动元素在另一个元素之后显示，会超出容纳块，则它下降到低于先前任何浮动元素的位置，如图8-36所示（浮动元素开始于下一行以更清楚地显示所使用的原则）。这个规则最先出现于CSS2，以改正CSS1中的疏漏。

8. 浮动元素的放置应尽可能高。

这条规则从属于以上介绍的七条规则。过去的浏览器使用图像标签出现的行框之

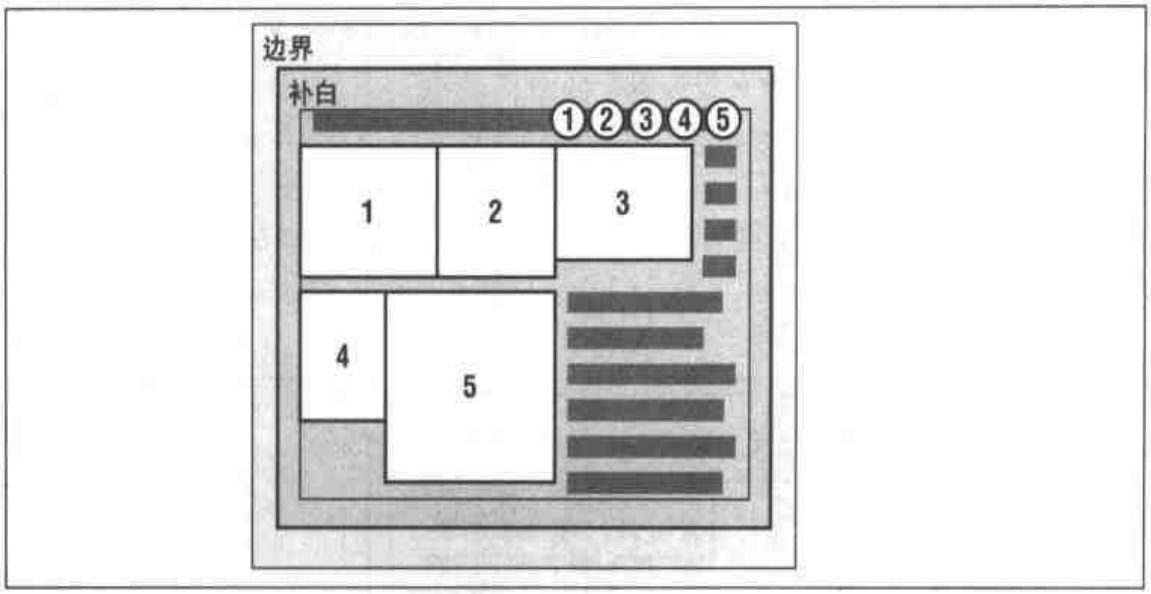


图 8-36 如果没有空间，则另起一行

后的行的顶部来对齐浮动元素的顶部。本规则隐含有使用图像标签出现的行的顶部来对齐的意思，只要空间允许这样做。理论上正确的行为见图 8-37。

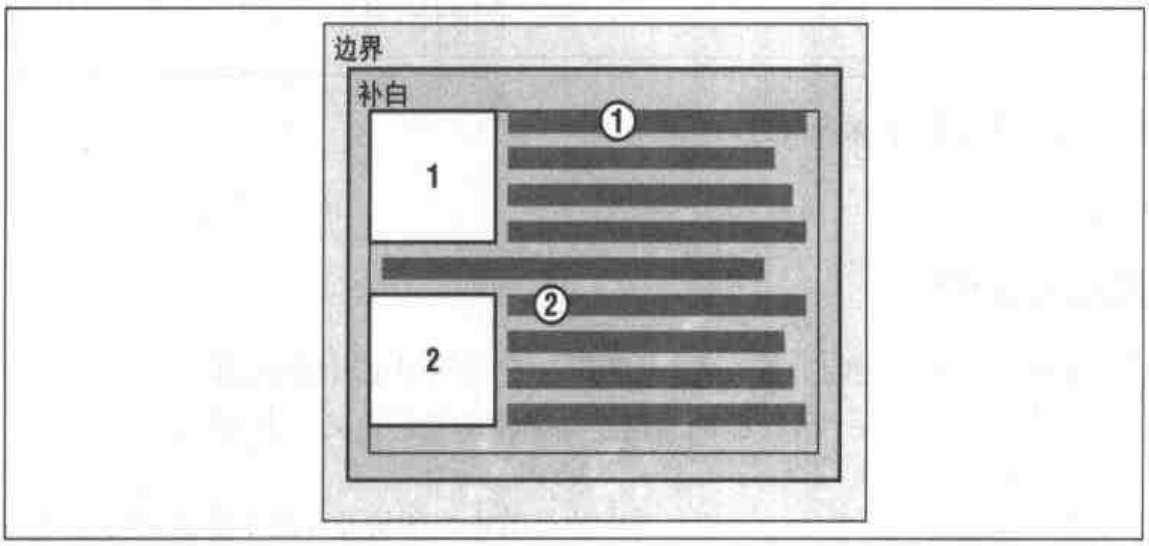


图 8-37 在遵从其他限制下，尽可能高

警告：不幸的是，由于没有预先定义“尽可能高”的含义（实际上成为“尽可能方便地高”），即使在CSS1兼容的浏览器上，也不能信任行为会一致。多数浏览器会遵从以前的实践把图像浮动到下一行，但有一些，如Opera 3.6，会在空间允许的情况下浮动到当前行。

9. 向左浮动的元素必须尽可能向左放置，向右浮动的元素必须尽量向右放置。较高的位置优先给更偏左或偏右的元素。

这条规则隶属于先前引入的规则。规则8中有类似的防止误解的说明，尽管其实并不模糊。如图8-38所示，很容易说出什么情况下，元素已经尽可能地浮动到最左或最右。

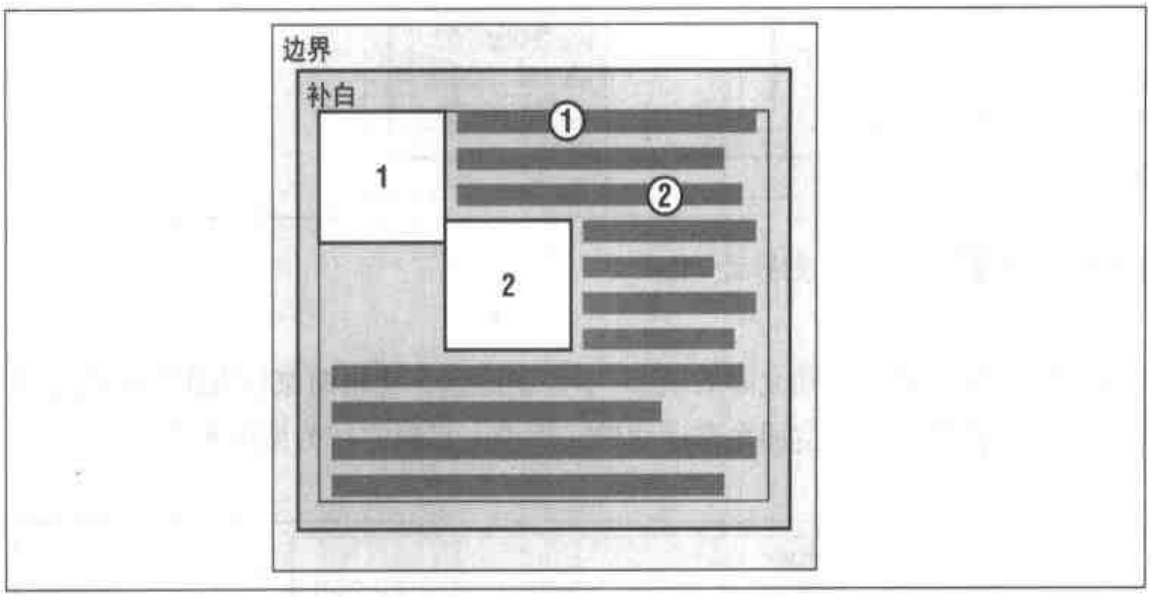


图8-38 尽量浮动到最左（右）

已应用的行为

以上规则有很多有趣的后果，这些后果既是它们的规定引起的，也由它们未规定的东西引起的。第一个要讨论的问题是如果浮动元素比父元素高时会怎样。

事实上，这种情况经常发生，且在前面的章节中有所谈及。以一个简单的文档为例，该文档由一些段落及H3标题组成，第一个段落包含一个浮动元素。进一步来说，浮动图像有5px的右边界。读者会希望文档的绘制如图8-39所示。

当然，没有意外发生。但8-40显示了当第一个段落设置了背景后的情况。

除可见背景外，第二个例子中没什么不同。如图所示，浮动元素突出于其父元素的底部。当然，第一个例子中也是这样的，但并不明显，因为我们无法看到背景。

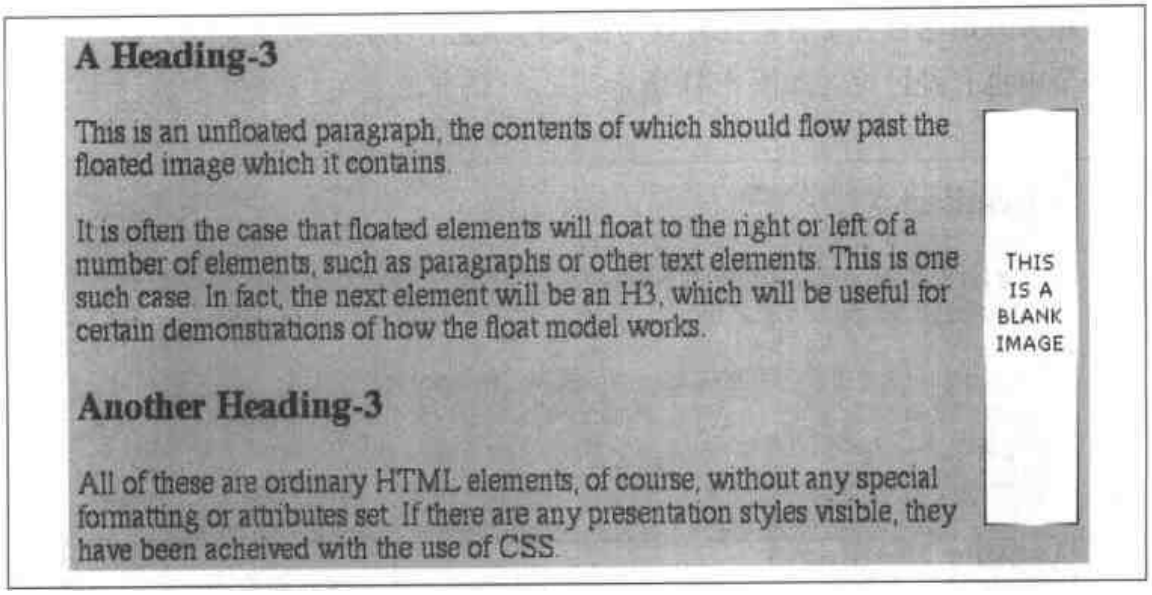


图 8-39 期望的浮动情况

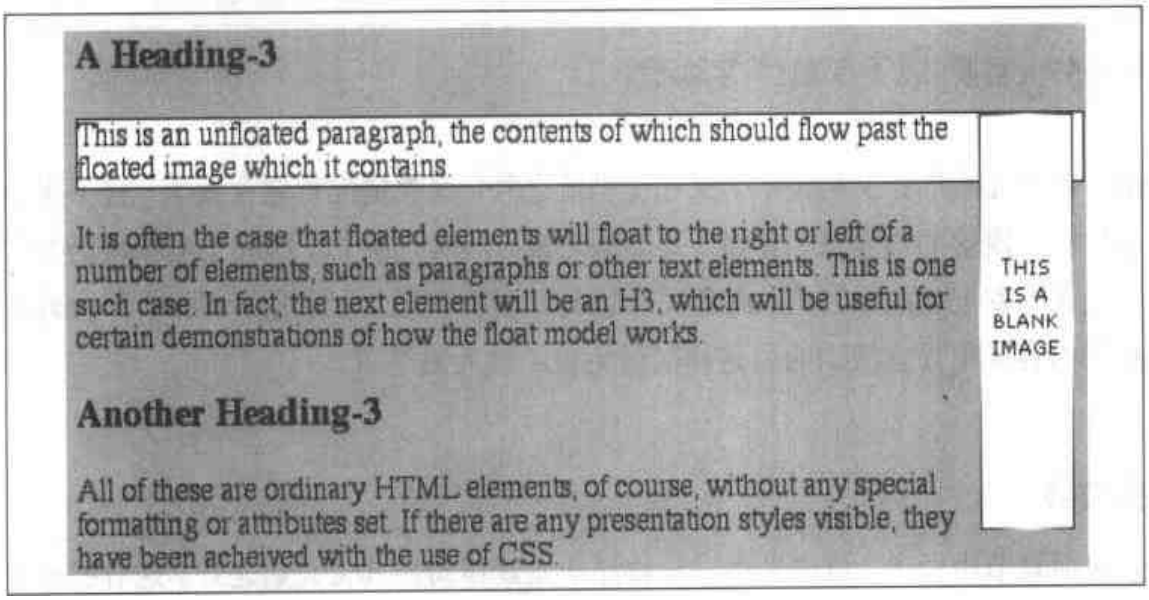


图 8-40 背景及浮动元素

这个行为不受任何限制。我们讨论过的浮动规则只定位浮动元素及其父的左侧、右侧，及顶，有意空缺了底端的定位使得图 8-40 的行为被允许。

警告：在实际应用中，有些浏览器不能够正确处理这件事。它们会增大父元素的高度以容纳浮动元素，即使这样做会在父元素中生成大量额外的空白。

一个相关的话题是关于背景及背景与在文档中先出现的浮动元素的关系，这在以前章节中也有过讨论，如图 8-41 所示。

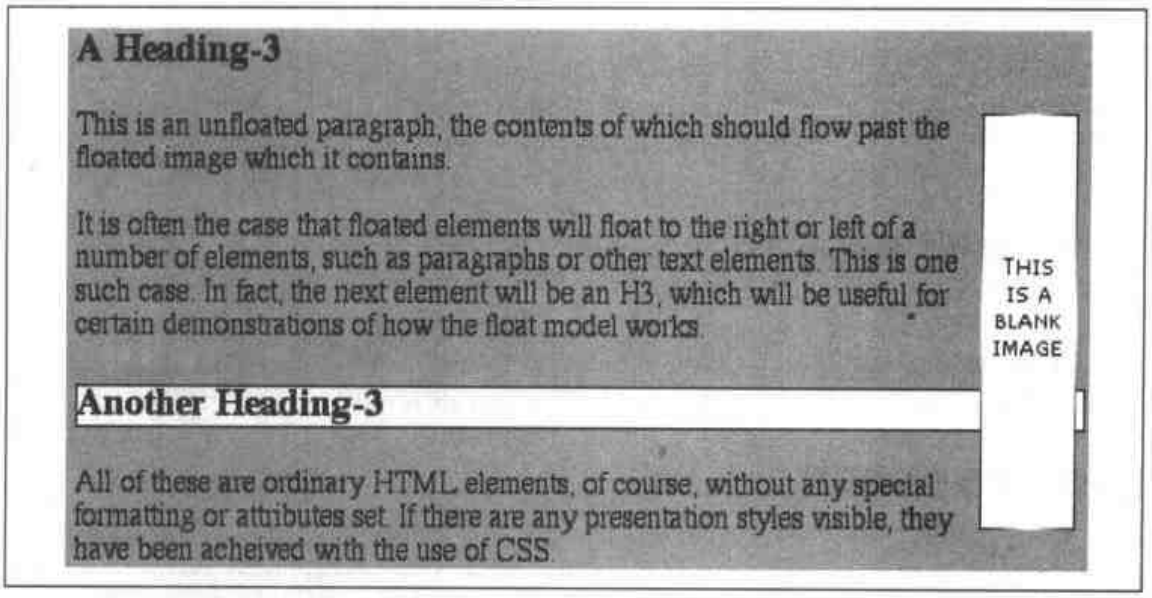


图 8-41 背景“从下方滑过”浮动元素

由于浮动元素既在文档流中，又在文档流之外，这类情况肯定会发生。接下来会怎样呢？段落的内容被“置换”为浮动元素。然而，每个段落的元素宽度仍与其父元素的宽度相同。因此，内容区跨越父元素的宽度、背景也一样。实际内容不是在内容区中任意流动的，这样以避免被浮动元素遮盖。

负边界

如前面章节中所述，负边界可以导致浮动元素移动到其父元素之外。这看起来直接违背前面解释过的规则，其实不是。与元素通过负边界可显得比其父元素宽的道理一样，浮动元素也可以看起来伸到其父元素之外。

考虑一个向左浮动的图像，其左边界、上边界均为 -15px。这个图像置于一个没有补白、边框、边界的 DIV 中。结果如图 8-42 所示。

与外观相反，在技术上不违反浮动元素不能放置于父元素之外的约束。允许该行为的技术细节如下：仔细阅读前面列出的规则，会看到一个浮动元素的外边沿必须在其父元素之内。然而，负边界中可放置内容，从而在效果上覆盖它自身的外边沿，如图 8-43 所示。

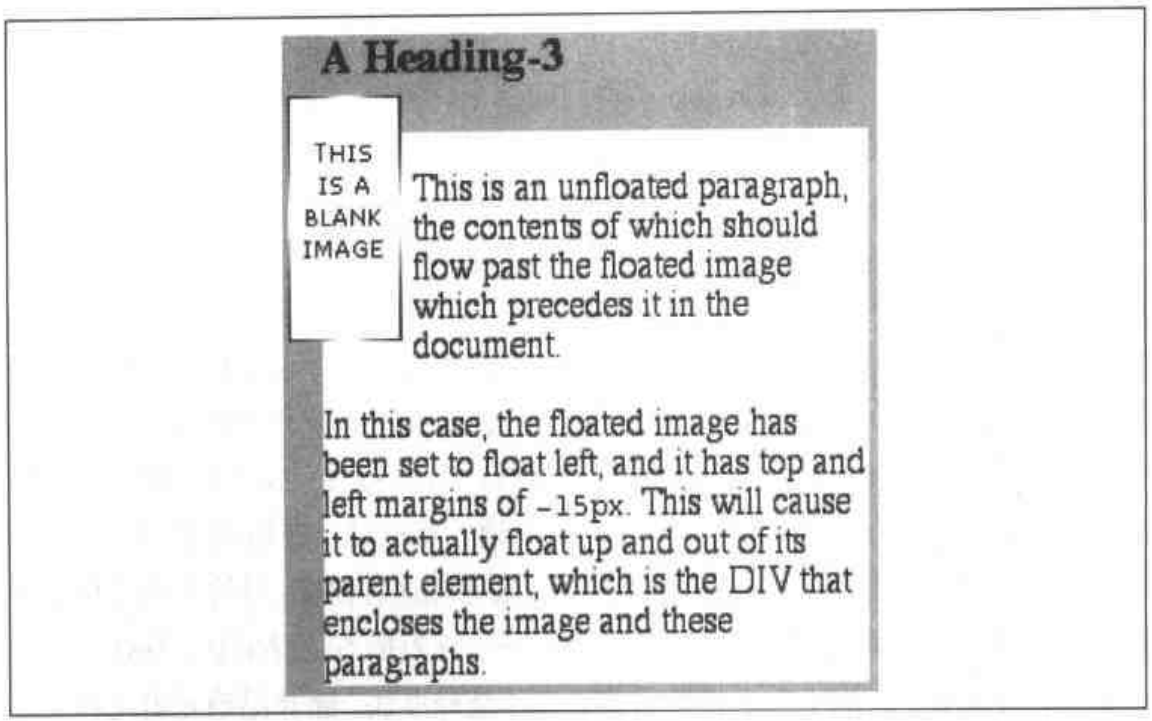


图 8-42 有负边界元素的浮动

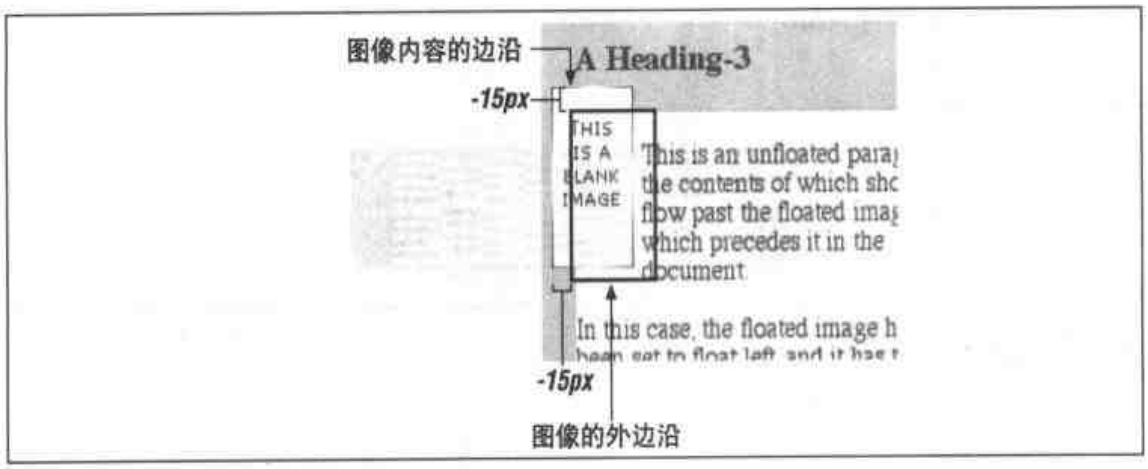


图 8-43 带有负边界元素向上、向左浮动的细节

这种情况下数学计算的过程大致如下：假定DIV的上内边沿在100px的位置。浏览器为了计算出浮动元素上内边沿的位置，进行如下计算：100px + (-15px) 边界 + 0补白 = 85px。这样，浮动元素上内边沿的位置是85px处，尽管这个位置高于浮动元素的父元素的上内边沿位置，这样的数学计算仍不违背规范。类似的过程可用来解释浮动元素的左内边沿是怎样可以放置于其父元素左内边沿的左侧的。

这时，也许很多读者想说“这是错的！”。是的，允许上内边沿高于上外边沿看起来完全错误，但这正是使用负边界所得到的结果，就像负边界应用于普通的、非浮动的元素上可以使这些元素比其父元素宽那样。浮动元素框的四个边均是如此，设置边界为负数，则内容可以覆盖外边沿且在技术上不违背规范。

这里有一个重要的问题，即：当一个元素使用负边界浮动到其父元素之外时，文档显示会发生什么变化呢？例如，一个图像可以浮动很远，进入到用户代理已经显示过的段落中。在这种情况下，文档是否应回流交由用户代理决定。CSS规范中特别声明不要求用户代理回流先前的内容以适应文档后来发生的变化。换句话来说，如果图像浮动到先前的段落中，便简单地重写已经存在的内容。另一方面，用户代理可以通过流动浮动元素周围的内容来处理这种情况，尽管这不是规范所要求的行为。不管采用哪种方式，寄希望于特定行为都不是好主意。悬挂浮动十分安全，但试图在页面上向上推某元素通常不是好办法。除非浮动元素比其父元素宽，否则浮动元素不能够超出其父元素的左内边沿和右内边沿。在这种情况下，浮动元素覆盖左内边沿或右内边沿，由其浮动方向决定，以尽量正确地显示自身。产生的结果类似于图8-44。

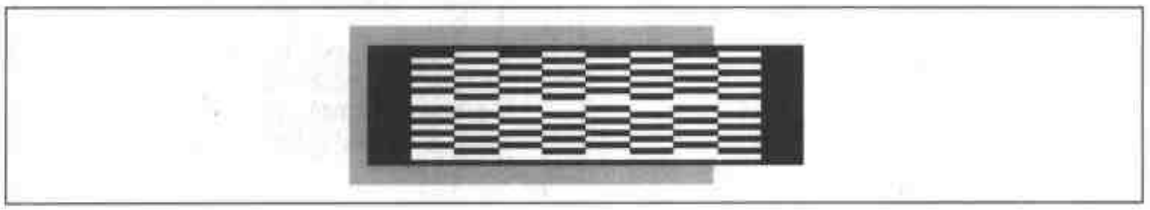


图8-44 浮动一个比父元素宽的元素

内联元素

任何不是块级元素（或直接，或隐含，如浮动元素）的可见元素均为内联元素。为内联元素设置框属性会进入到一个比已经讲述过的内容更为有趣的领域。内联元素的好的例子是EM标签和A标签，这两者都不是替换元素，而图像是替换元素。

警告：注意这些都不适用于表格元素。CSS2引入新的属性和行为来处理表格和表格内容，这些新的特性的行为与块级及内联格式编排有很大的不同，在第十章中将有所概述。

行布局

首先，我们需要理解内联的内容是如何布局的。这不像块级元素那样简单、直接，块级元素只是生成框，而且通常不允许其他内容与之相邻出现。如果不看块级元素，如段落的内部，它当然没什么问题（即使它忽略了浮动）。其内部全是文本行，我们会问“它们是怎样到那里的？怎么控制它们的安排？我怎么来影响它？”

为理解行是如何生成的，首先考虑一个包含非常长的文本的元素的情况，如图8-45所示，注意行已经加上了一个边框，这由将整行包于SPAN元素中并指定一个边框样式来完成：

```
SPAN {border: 1px dashed black;}
```

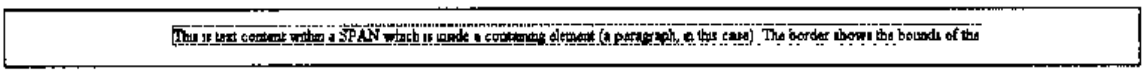


图8-45 一个单行内联元素

这是块级元素包含内联元素的最简单的情况，其方式与一个由两个词组成的段落没有区别。图8-45中唯一的不同是那里有很多词，且大多数段落不像SPAN那样包含显而易见的内联元素。

为从这个简化了的状态转到更熟悉的状态，所需要的只是决定元素应有多宽，然后将行分成小段以适应元素的宽度。这样就得到了如图8-46所示的状态。

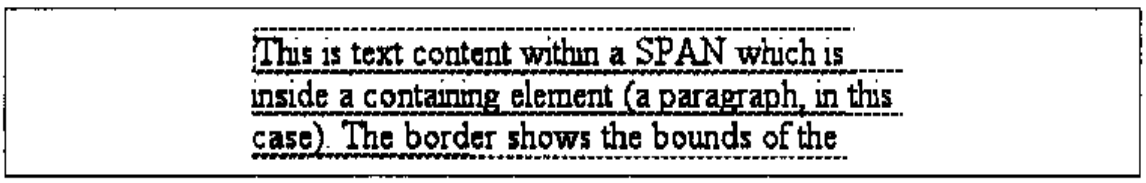


图8-46 多行内联元素

基本上没有什么变化。所做的只是取走那行文本，将其分成小段，然后堆在一起，很简单。

在图8-46中，每行文本的边框恰好是行框顶端与底端相接。这只在未对内联文本设补白或行高度时成立，此时，让我们用可视提示作为参考。注意，实际上边框相互间有轻微的重叠：例如，第一行的下边框刚好低于第二行的上边框。这是因

为边框实际上绘制于每个行框外的下一个像素(假定使用显示器),它们的边框会如图 8-46 所示重叠。

如果改变 SPAN样式，让它有背景颜色，行框的实际放置会非常清楚，如图 8-47 所示。

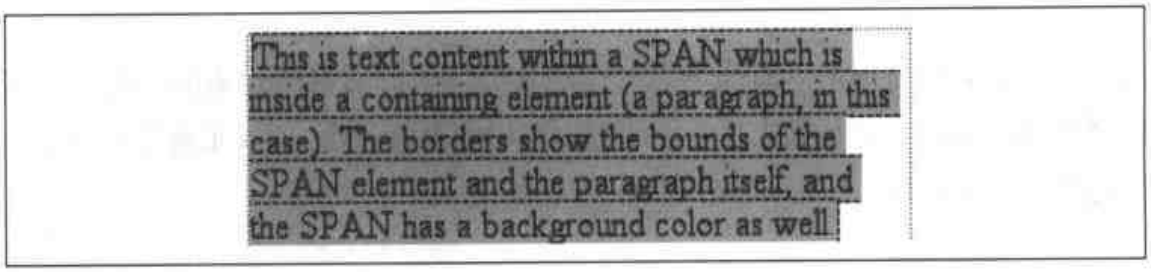


图 8-47　每个行框的全部宽度

可以看到，不是每行都达到了段落内容区的右边沿，图中用灰点线指示出来。每个行框结束于何处是由行框的内容决定的。为提供比较，我们再做一次，但这一次段落采用右对齐，如图 8-48 所示。

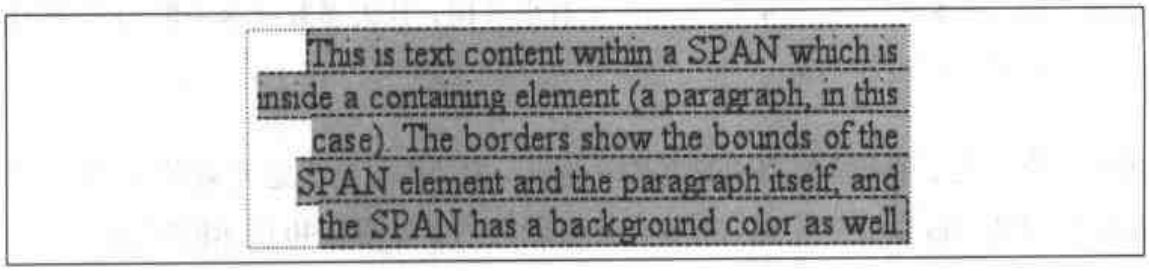


图 8-48　右对齐时的行框布局

这次得到的结果是单行文本的各段堆在一起，右侧在一条线上。如果将段落的 text-align设为 center，则各行的中心在一条线上，如果设为 justify，则每行被强制为段落内容区的宽度。区别在于字母间与单词间的距离，如图8-49所示。

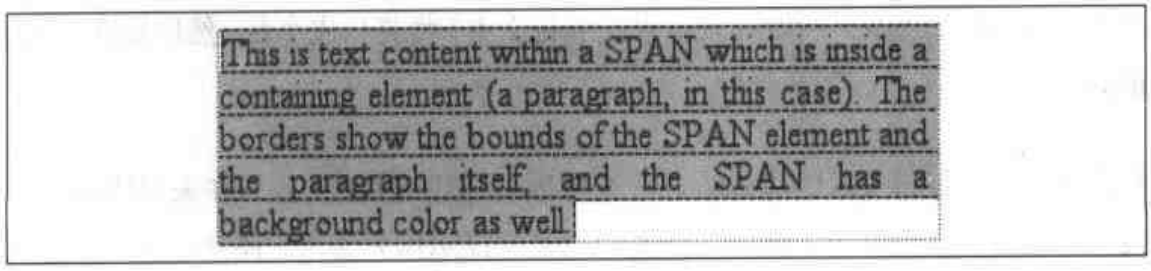


图 8-49　完全对齐时的行框布局

以上基本说明了行框的生成，至少是其最简单情况。接下来会看到，内联格式编排模型却并不简单。

内联格式编排

如同第四章“文本属性”中所讲述的，每个元素都有line-height属性。这与内联元素的显示有很大的关系，需要先进行详细讨论。

首先，我们来确定一下行的高度是如何决定的。行的高度（或行框的高度）是由其构成元素及其他内容（如文本）的高度决定的。理解line-height只应用于内联元素及内联内容而不应用块级元素是重要的。可以为块级元素设置line-height值，但其影响只通过应用于块级元素内的内联内容而产生。例如，考虑下面这个段落：

```
<P STYLE="line-height:0.25em"></P>
```

没有内容，段落没有什么可以显示的，所以没有影响。段落的line-height无论为何值，0.25em还是25in，如果没有内容就没有区别。

从某种意义上来讲，块级元素中每一行的文本都是一个内联元素，尽管它没有任何标签包围。如果愿意，可以写出虚构的标签如下：

```
<P>
<LINE>This is a paragraph with a number of</LINE>
<LINE>lines of text which make up the</LINE>
<LINE>contents.</LINE>
</P>
```

尽管LINE标签不存在，情况就如同它们存在一样。文本的每一行从段落继承样式，因此它们也可以像这样被包含在标签中。因此，为块级元素生成line-height规则的唯一原因是，这样就无需为它的内联元素特别声明line-height，不论它是否是构想中的内联元素。

注意： 虚构的LINE元素实际上解释了设置line-height于块级元素所导致的行为。根据CSS规范，对块级元素声明line-height即设置了块级元素内容的最小线框高度。这样，声明P.spacious {line-height:24pt;}意味着每个线框的最小高度是24pt。从技术角度来讲，内容继承这个行高度只有通过内联元素来继承。大多数文本不包含于内联元素中。这样，如果假定每行包含虚构的LINE元素，则模型会工作得很好。

生成一个行框

以下是用户代理生成行框的步骤。首先，对每个非替换元素（或内联元素之外的文本串）使用font-size来决定其初始内容高度（*content-height*）。这样如果一个内联元素的font-size为15px，则内容高度开始为15px。

第二步，在一指定行的内联元素中根据它们的vertical-align值来对齐。缺省情况下，这会使行中所有的文本沿它们的基线对齐，当然，不同的vertical-align值会产生不同的效果。例如，所有元素都可以采用顶端对齐。本章稍后将继续讨论垂直方向对齐的问题，这里我们假定所有对齐方式均为基线对齐。

此时，line-height开始起作用。假定如下情况：

```
<P STYLE="font-size: 12px; line-height: 12px;">
This is text, <EM>some of which is emphasized</EM>, plus other text<BR>
which is <B STYLE="font-size: 24px;">boldfaced</B> and which is<BR>
larger than the surrounding text.
</P>
```

情况是有一些文本的font-size为12px，另外一些文本的大小是24px。然而，所有文本的line-height均为12px，这是由于line-height是可继承属性。结果是font-size和line-height的差值被一分为二，分别应用在每个元素的内容高度顶端与底端以达到内框。其差值的一半称为半铅条（*half-leading*）。

这样，对font-size和line-height都是12px的文本的每一位，没有高度应用于内容高度上（因为12减12等于0，而0的一半仍是0），因此内联框为12px高。对于加宽文本，font-size和line-height的差值是12px。差值被分成两份以决定半铅条（6px），从内容高度的顶端与底端各减去半铅条的高度以达到内联框的高度，在本例中为12px。这12个像素内联框垂直居中于元素的内容高度中。

看起来仿佛对文本的每一位都做了同样的处理。但事实不是这样。内联框实际上不在一条线上，因为文本是基线对齐的，如图 8-50 所示。

图 8-50 内联框（灰色）是如何影响行框高度的

然而，恰恰是内联框决定了整个行框的高度。行框定义为最高内联框的上沿到最底内联框下沿的距离，如图 8-51 所示。

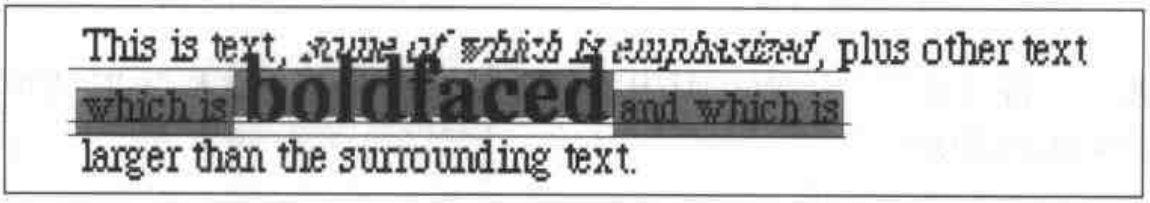


图 8-51 将行框堆在一起

然后行框的顶端靠着上一行的行框底端放置，得到如图 8-52 所示的段落。

图 8-52 堆积行框的最终段落

如图所示，中间一行比其他两行高，但仍不足以容纳其中的文本。这是因为行中内联框的位置迫使它高于 12 像素，但行框仍不足以高到避免与其他行重叠的程度。

如果我们改变内联框的垂直方向对齐，情况就会大不相同。假定将加粗文本的垂直方向对齐方式改为 middle，将会导致如图 8-53 所示的结果。

此时，加粗文本内联框的中央内联框都为 12 像素高，且它们中央共线，这意味着该行框只有 12 像素高，同其他的行一样。然而，这还意味着较大的文本要比刚才更多地进入到其他行中。

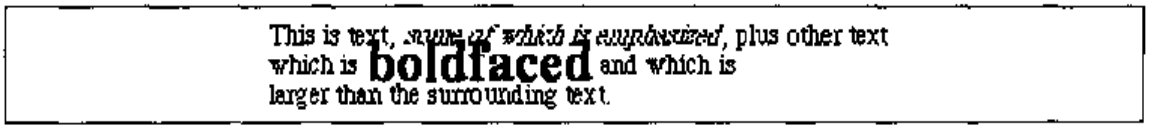


图 8-53 如何改变加粗文本的垂直方向对齐

考虑另一种情况，另一个内联元素与加粗文本处于同一行中，但采用的不是基线对齐方式：

```
<P STYLE="font-size: 12px; line-height: 12px;">
This is text, <EM>some of which is emphasized</EM>, plus other text<BR>
which is <B STYLE="font-size: 24px;">boldfaced</B>
and <SPAN STYLE="vertical-align: top;">tall</SPAN> and which is<BR>
larger than the surrounding text.
</P>
```

此时，回到前面的例子，即中间行比其他两行高的例子。注意文本“tall”在图 8-54 中是如何对齐的。

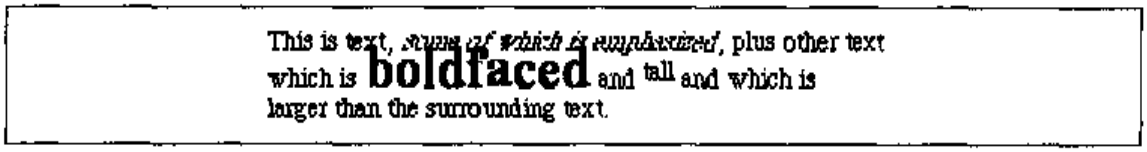


图 8-54 顶端对齐的文本

这里的情况是，“tall”文本的内联框顶端与行框的顶端对齐。同于“tall”文本有着相同的 font-size 和 line-height 值，它的内容高度及行框是一样的。然而，考虑以下标记：

```
<P STYLE="font-size: 12px; line-height: 12px;">
This is text, <EM>some of which is emphasized</EM>, plus other text<BR>
which is <B STYLE="font-size: 24px;">boldfaced</B>
and <SPAN STYLE="vertical-align: top; line-height: 4px;">tall</SPAN>
and which is<BR>
larger than the surrounding text.
</P>
```

由于“tall”文本的 line-height 小于它的 font-size，这个元素的内联框比较小。这会改变文本自身的设置，因为它的内联框的顶端必须与所在行的行框顶端对齐，这样得到的效果如图 8-55 所示。

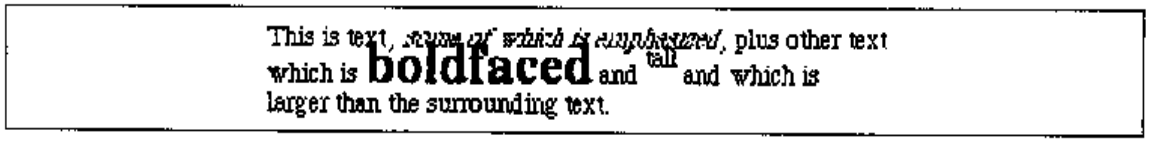


图 8-55 很小的内联框的效果

另一方面，可以给“tall”文本设一个比其字体大小还要大的line-height。例如：

```
<P STYLE="font-size: 12px; line-height: 12px;">
This is text, <EM>some of which is emphasized</EM>, plus other text<BR>
which is <B STYLE="font-size: 24px;">boldfaced</B>
and <SPAN STYLE="vertical-align: top; line-height: 18px;">tall</SPAN>
and which is<BR>
larger than the surrounding text.
</P>
```

由于设置“tall”文本的line-height为18px，font-size与line-height的差值为6像素。这种情况下，3像素的半铅条加到内容区，而不是减（因为line-height比font-size大）。这就会生成一个高为18像素的内联框，且其顶端与行框的上沿对齐。如图 8-56 所示。

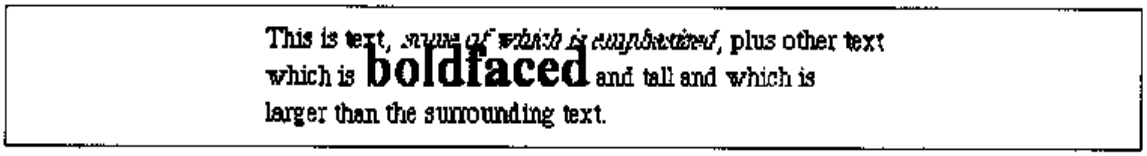


图 8-56 不同行高度下的顶端对齐文本

在进一步深入之前，先看一下为内联元素增加框属性的情形。

增加框属性

在弄清楚先前讨论的内容后，便可以把补白、边界及边框都只用于内联非替换元素，且它们不影响line-height。如果将边框应用于SPAN元素，不使用任何边界和补白，会得到如图 8-57 所示的效果。

边框被如此放置的原因是因为内联元素的边框边沿是由font-size控制，而不是由line-height控制的。换句话说，如果SPAN元素的font-size为12pt，line-height为36pt，则其内容区高度为12pt，且内容区是被边框包围的部分。

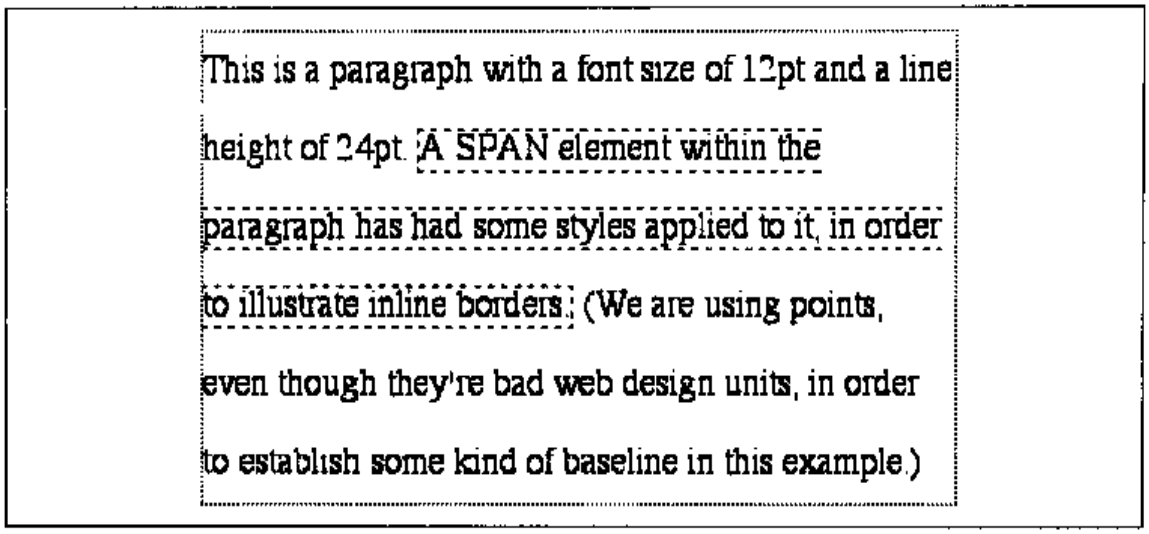


图 8-57 内联边框及行框布局

这个行为可以通过为内联元素指定补白来改变，使得边框远离文本自身（如图 8-58 所示）：

```
SPAN {border: 1px dashed black; padding: 4pt;}
```

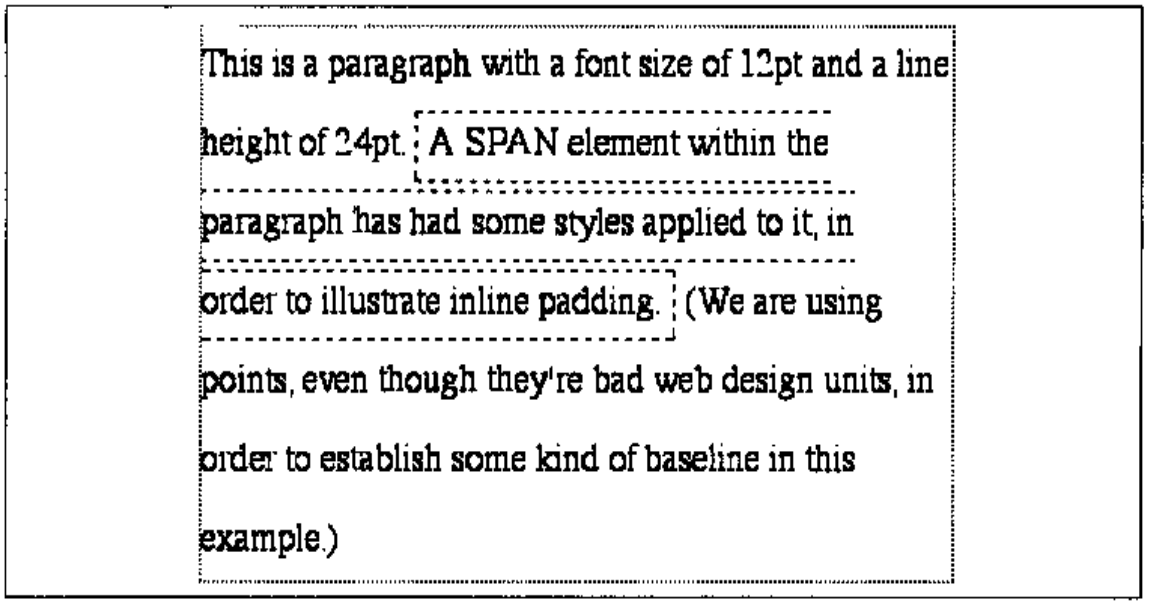


图 8-58 内联补白及行框布局

注意补白不改变内容高度的实际大小，也就不会影响这个元素的内联框的高度。给内联元素增加边框不会影响行框生成的方式，如图 8-59 所示。

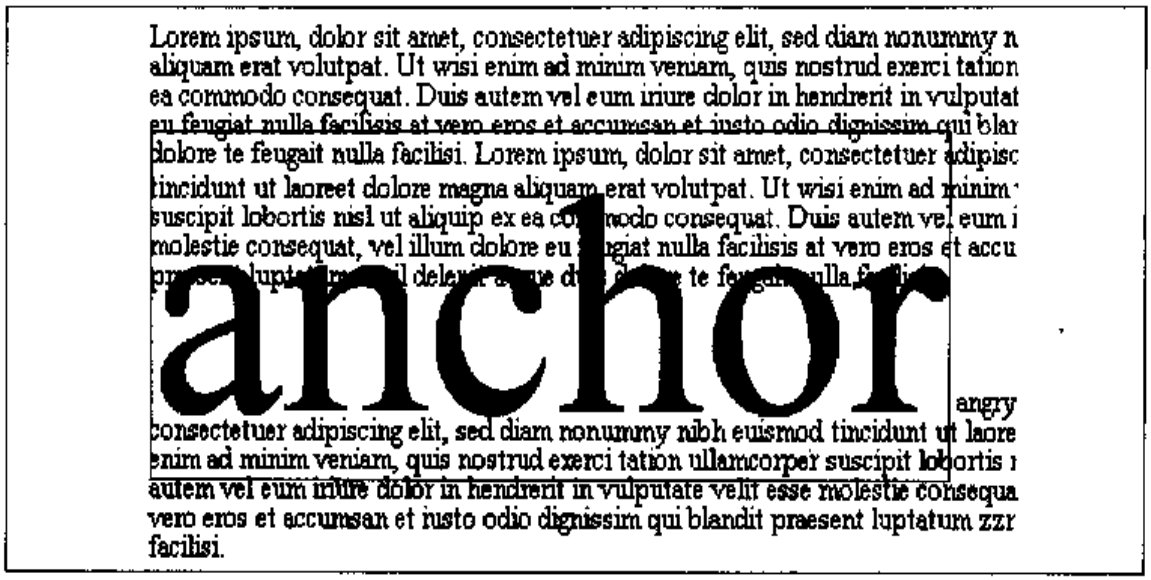


图 8-59 大的内联框会重叠其他许多行

也可以指定边界于非替换内联元素，但效果不会应用到非替换内联元素的顶端和底端，因此不会影响行框的高度。元素的结尾是另一种情况，如同第七章“框与边框”中讲述的那样。这仍是因为多行显示的内联元素与分成小段的单行元素相同。图 8-60 显示了这些样式造成的情况的更多细节：

```
SPAN {border: 1px dashed black; padding: 4pt margin: 8pt;}
```

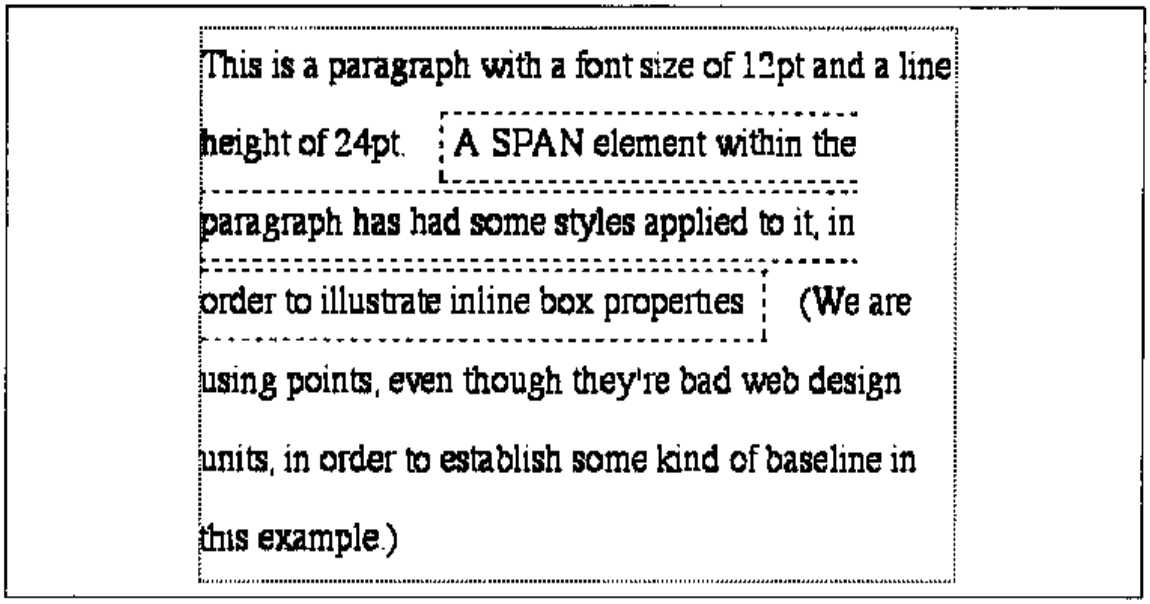


图 8-60 内联边界及行框布局

管理内联元素的行高度

在上一节中，讲述了改变内联元素的line-height导致可能发生一行中的文本覆盖另一行的几种情况。在每种情况里，变化都是针对单独的元素的。怎样才能更普遍地影响元素的 line-height，以防止重叠呢?

一种方式是与 font-size改变的元素联合使用 em单元。例如:

```
P {font-size: 14pt; line-height: 16pt;}
SPAN {background: gray;}
BIG {font-size: 250%; line-height: 1em; background: silver;}

...line in which</SPAN><BIG>some big text</BIG><SPAN>is found...
```

结果如图8-61所示。通过为BIG元素设置 line-height，行框的总高度增大了，这样可以有足够的空间来显示BIG元素，不会重叠其他行，也不会影响段落中所有行的line-height。使用值1em，这样BIG元素的line-height会设置为BIG元素的font-size大小。记住，line-height设置是与元素自身的font-size相关，而不是跟其父元素的font-size相关。

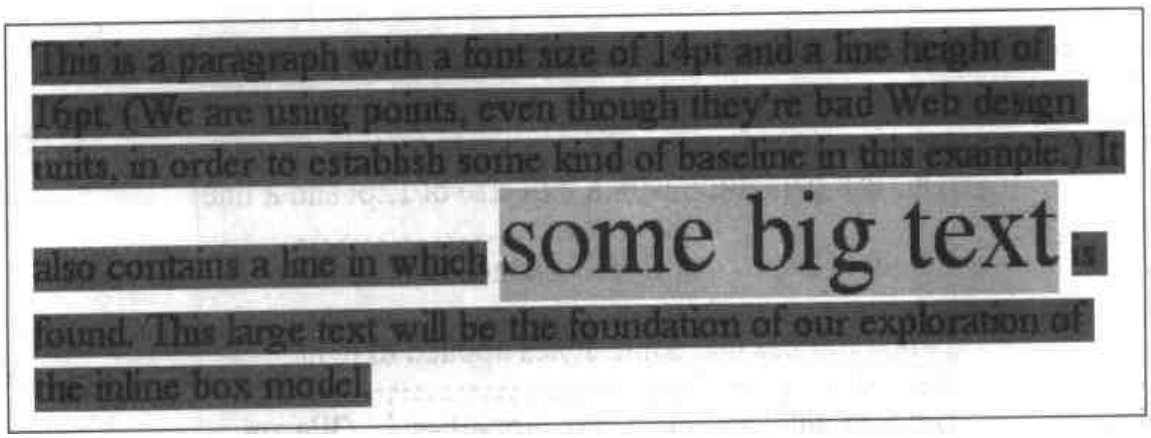


图 8-61 为内联元素增加边框

在做类似于给内联元素增加边框的事情时，记住这些是很重要的，假定想为任何超链接设置5像素的边框:

```
A:link {border: 5px solid blue;}
```

如果未设置足够大的line-height以容纳边框，就会有覆盖掉其他行的危险，如图8-62所示。

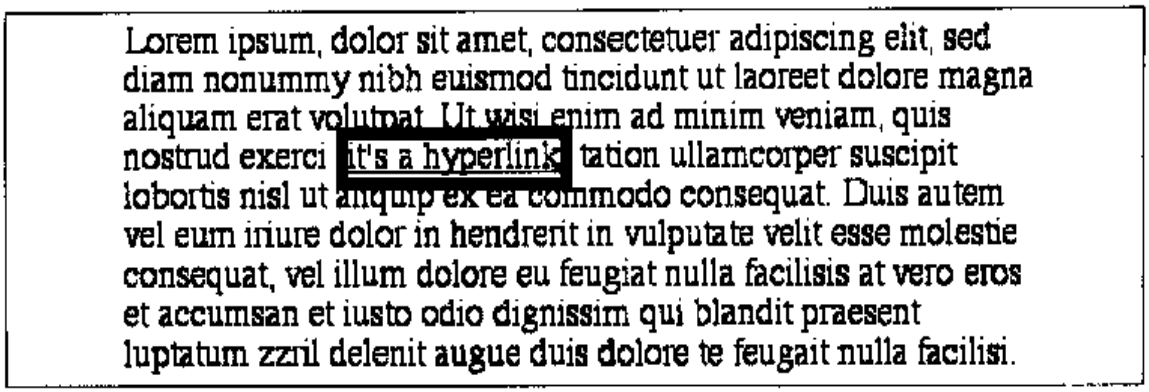


图 8-62 内联边框会重叠

一种解决办法是增大段落的line-height。但这会影响整个元素中的每一行，而不仅仅是加框超链接出现的行：

```
A:link {border: 5px solid blue;}
P {font-sive: 14px; line height; 24px;}
```

因为每行的顶端与底端都增加了额外的空间，超链接周围的边框不会侵犯其他行，如图 8-63 所示。

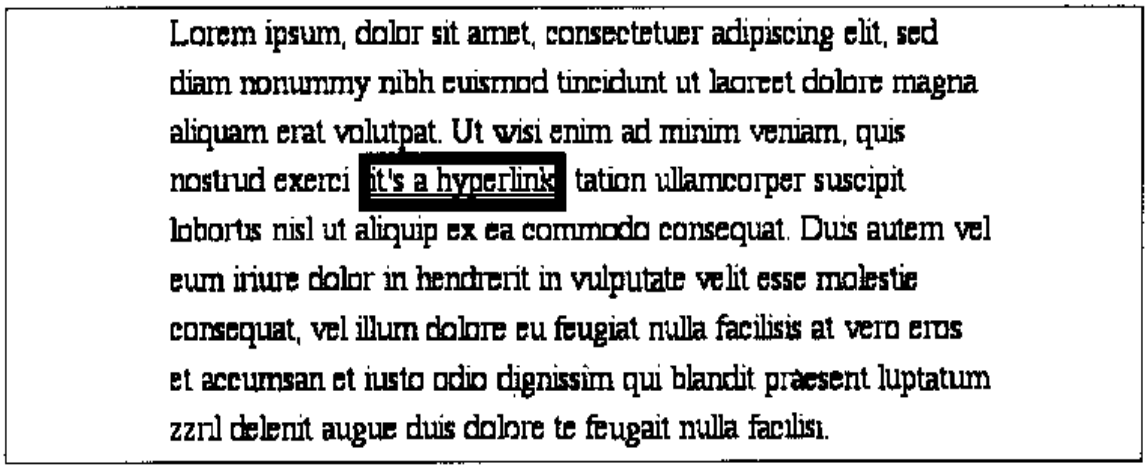


图 8-63 增大 line-height 以给内联边框提供空间

当然，这种处理在上面的特定情况下有效，是因为所有文本的大小都相同。为了覆盖所有的情况，也许简单地增加定位锚元素自身的line-height更有意义，如下：

```
A:link {border: 5px solid blue; line-height: 24px;}
P {font-size: 14px}
```

如果一行中的内容是大小相同的文本，则行框的高度将是行框中包含的最大line-height值（由于它与行框中最高的内联框高度一致），因此它是有效的。而且，它只影响超链接出现的那些行。当然，我们还可采用其他方法。

缩放行高度

还有一种更好的方式来设置line-height，即使用一个纯数字作为line-height的值。这样做的好处在于该数字只用作缩放因子，是继承得来的，而不是计算值。如果希望文档中所有元素的line-height是其font-size的1.5倍，只需声明：

```
BODY {line-height: 1.5;}
```

这个比例因子1.5会在元素间传递，在每个级别上该因子用作每个元素的font-size的乘数。因此，如下的标记会生成如图8-64所示的结果（增加背景只是为了便于说明）。

```
P {font-size: 12px; line-height: 1.5;}
SMALL {font-size: 66%;}
BIG {font-size: 200%;}

<P>This paragraph has a line-height of 1.5 times its font-size. In addition,
any elements within it <SMALL>such as this small element</SMALL> also have
line-heights 1.5 time their font-size... and that includes <BIG>this big
element right here</BIG>. By using a scaling factor, line-heights scale
to match the font-size of any element.</P>
```

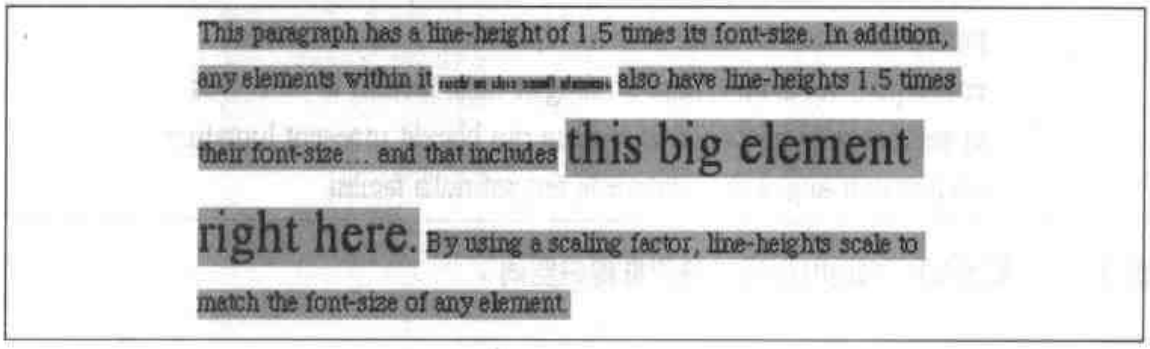


图8-64 对line-height使用缩放因子

在本例中，SMALL元素的line-height为12px，BIG元素的line-height为36px。这也许看起来差距有些太大，但它们符合整体页面设计的要求。当然，如果不希望BIG文本生成太多的额外空间，可使用以下规则替代：

```
P {font-size: 12px; line-height: 1.5;}
SMALL {font-size: 66%;}
BIG {font-size: 200%; line-height: 1em;}
```

警告：任何有用的特性都有弊端。Internet Explorer 3.x会把这个缩放因子当成像素来处理。想像一下line-height为1.5像素的段落，相当有趣。

另一个解决办法是设定样式，使行的高度刚好能容纳其内容。这时可以把line-height定为1.0。这个值会将自身与font-size相乘并得到与每个元素的font-size相同的值。这样，对每个元素来说，其内联框与内容区相同。

内联替换元素

内联替换元素，比如图像，是内联格式编排处理产生重大差别的原因。这些差别源于仍假定替换元素具有实际高度和宽度。例如，一个图像会有一定数量的像素高和宽。

然而，一个有实际高度的替换元素会使行框比通常情况下高。这不会改变内联任何元素的line-height值，包括图像自身。行框只是增高到可以容纳替换元素加上各种框属性。换句话来说，替换元素整体（包括内容、边界、边框、补白）用于定义元素的内联框。下面的标记给出了这样的例子（见图8-65）：

```
P {font-size: 12px; line-height: 18px;}
IMG {height: 30px; margin: 0; padding: 0;}
```

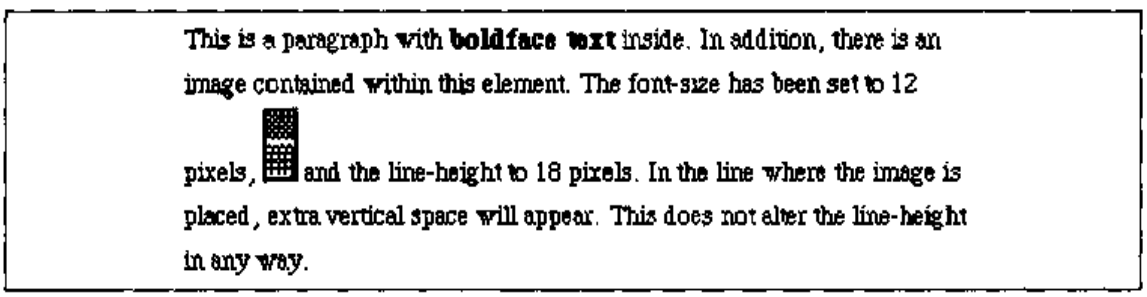


图8-65 替换元素实际上没有增加行高度

尽管有很多空白区，line-height的有效值并没有改变。对图像的内联框没有影响，在这种情况下仍为30px高。

然而，为什么一个内联替换元素也有line-height值？这是为了能够正确定位垂直对齐的元素。vertical-align的百分比值根据元素的行高来计算。这样：

```
P {line-height: 18px;}
IMG {vertical align: 50%;}

<P>The image in this paragraph <IMG SRC="test.gif" ALT="test image">
will be raised 9px.</P>
```

继承的line-height值是使图像增高9像素的原因，而不是其他的值。没有line-height值，则不可能生成垂直对齐的百分比。在垂直对齐时，与图像自身的高度是无关的，line-height的值决定了一切。

增加框属性

在这之后，给内联替换元素应用边界、边框及补白就非常简单了。

补白及边框使用与通常一样的方式应用在替换元素上；补白在实际内容（如图像）周围插入空白，边框围绕补白。这两个属性确实影响行框的高度。如图8-66所示。

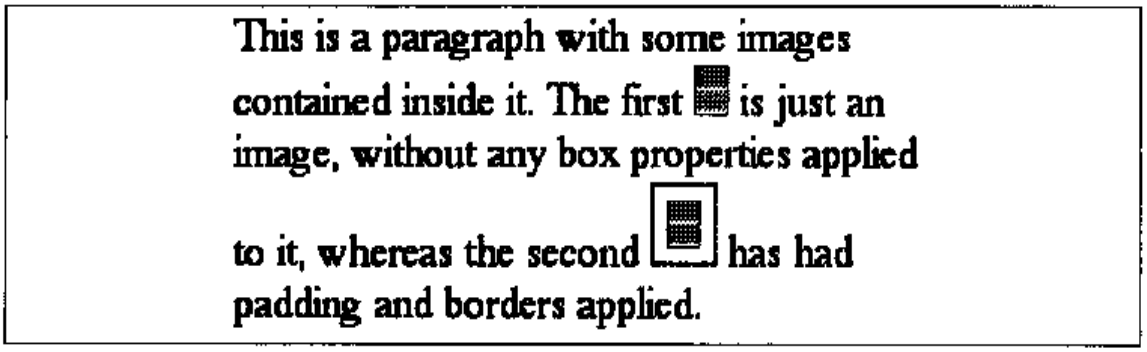


图 8-66 给替换元素增加补白及边框

注意，“第一个”行框高度足以容纳图像，而“第二个”的高度足以容纳图像、补白和边框。这是因为替换元素的全部（内容、补白、边框）构成了该替换元素的内联框。这也是迫使图 8-66 中行框增高的原因。

边界也包含于行框中，但它们有自己的方式。设置正边界没什么可谈的；只是使行框更高些，如图 8-67 所示。

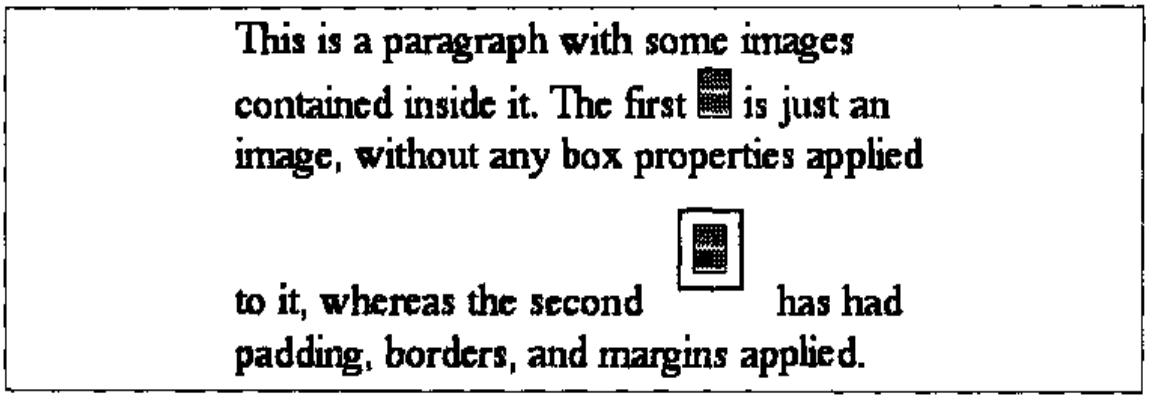


图 8-67　给内联替换元素增加补白、边框及边界

设置负边界，有着正如读者期望的效果：它会使行框变短。如图 8-68 所示，图中图像上方的行被向下拉动。

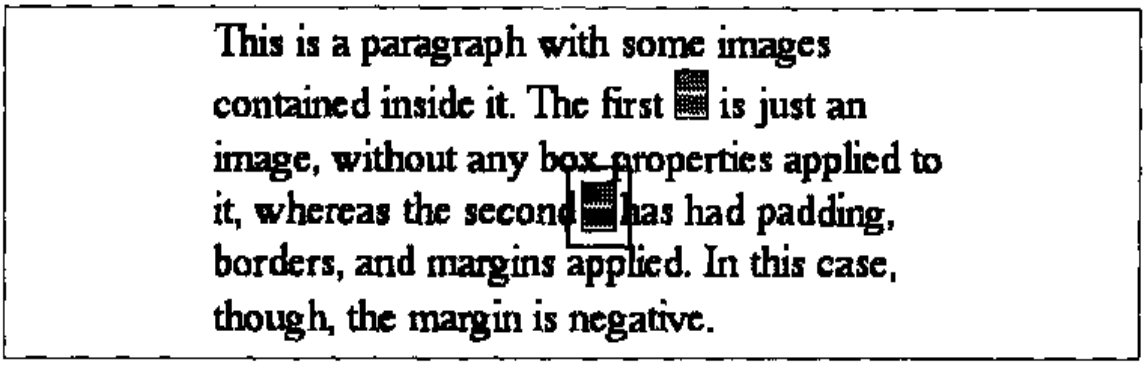


图 8-68　内联替换元素使用负边界的效果

这与负边界应用在块级元素上的操作很相似。这种情况下，负边界使得替换元素的行框比在普通情况下要小。这是使内联替换元素侵入其他行的唯一方式。

小结

尽管CSS格式编排模型的有些特征初看上去不直观，但使用它们却很有意义。在很多情况下，那些看上去没有意义甚至愚笨的规则已证实可以用来避免稀奇古怪的或不是所希望的文档显示。

鉴于此，牢固掌握可视格式编排是理解定位（positioning）如何工作的基础。下一章我们将讲述有关定位的内容，且方式与本章非常类似：讨论较倾向于理论。

第九章

定位

定位（positioning）的思路其实非常简单。它允许用户精确定义元素框出现的相对位置，可以相对于它通常出现的位置，相对于其上级元素，相对于另一个元素，甚至可以是相对于浏览器视窗本身。这个属性的功能很显而易见，也很让人吃惊。但如果知道这是用户代理首先要支持的CSS的部分时，也许就不会那么吃惊了。在本书完成时，定位属性已有了一些很好的实现，本书还试着给读者以预见性的介绍，也许几年后再读本书，它们已经被实现了。

读者可以注意到，与其他章不同，本章中几乎所有的图示都不是用web浏览器生成的。以下是关于定位应用的可靠性与一致性的声明：没有任何一个是可以完全信任的。实际上用手工画理论性的例子比在web浏览器上抓图、然后再用Photoshop来润饰要容易。

这也是为什么本章中几乎（但并非完全）没有浏览器警告及防止误解的声明的原因。我们选择简要地按CSS2规范所提供的内容来描述定位，而不是在旁注中写入说明性文本。也许在本书的第二版中会包含更多实用性建议，但在此时，唯一可以提供的实用性建议是：彻底检查定位代码，并做好定位与实现之间可能出现不一致的心理准备。

基本概念

在讲述定位的特殊机制之前，我们需要介绍大量的概念。这些概念实际上构成了CSS布局的基础，因为每个显示元素都可以用定位的方法来描述。毕竟，任何元素都是放置在屏幕上，或打印在纸上，有位置且必须被定位 —— 一般是由用户代理来完成定位，如果不是其他的话。

包含块（*containing block*）是格式编排发生的关联场景。例如，一个加粗元素的包含块可以是该元素所出现的段落，如图9-1所示。

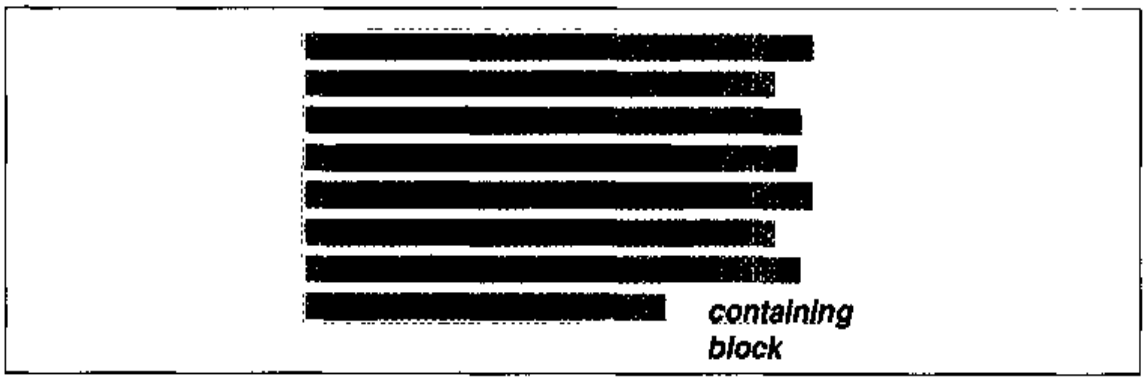


图9-1　包含块的例子

并非CSS中的每一个元素都为其后辈元素生成一个包含块。建立包含块的规则如下：

1. “根元素”的包含块（也叫初始包含块）由用户代理生成。在HTML中，根元素是HTML元素，尽管有的浏览器会不正确地使用BODY。
2. 对于那些未绝对定位的非根元素来说，元素的包含块设置为最近的块级祖先元素的内容区边沿。这甚至在相对定位时也是成立的，尽管一开始看起来可能不是那样。
3. 对那些使用absolute作为position的绝对定位的非根元素，包含块设为最近的position不是static的祖先元素（任何类型）。有以下几种情况：
   a. 如果祖先元素是块级元素，包含块设为祖先元素的补白边沿；换句话说，是被边框约束的区域。

b. 如果祖先元素是内联元素，包含块设为祖先元素的内容边沿。在从左到右方向的语言中，包含块的顶端与左侧是祖先元素中第一个框的顶端与左侧内容边沿，且底端与右侧是最后一个框的底端与右边沿对应的第一个框的右内容边沿，左侧取自最后一个框。顶端与底端也是这样。

如果没有这样的祖先元素，则根元素的内容边沿被用于建立包含块。

关于包含块，最重要的是要记住它为所有后辈元素建立了一个格式编排的上下文。例如，如果边界用百分比声明，百分比计算是同包含块相关的，这就提供给我们规则#2，它指出包含块通常与元素的内容区相同。

另一个关于包含块的重要方面是：元素可定位于它们的包含块之外。这种方式与浮动元素可使用负边界以浮动到上级元素内容区之外很类似。它也使得名词“包含块”应该叫做“定位场景”（positioning context）更合适，但由于规范中使用“包含块”，因此本文沿用它。（我们要尽力避免令大家疑惑）。

定位模式

可以使用position属性来在不同的定位类型中选择。

position

允许值	static | relative | absolute | fixed | inherit
初始值	static
适用于	所有元素
可否继承	否

position的值含义如下：

static

元素框按普通方式生成。块级元素生成一个矩形框，它是文档流的一部分，内联级框是由一个或多个行框的上下文生成的，这些行框流动于其上级元素中。

relative

元素框偏移一定的距离。它的包含块是未定位元素将占有的区域。元素保留未被定位时的形状，且元素通常占有的空间也被保留。相对定位完成的过程是首先按static方式生成一个元素，然后移动这个元素框（或多个框，在内联元素跨越多个行的情况下）。有可能定位元素会重叠其他内容。偏移的方向及幅度由top、right、bottom和left属性联合指定。

absolute

元素框完全从文档流中消除并根据其包含块定位。元素在普通文档流中占有的任何空间都被关闭，就如同元素不存在一样。元素的大小及位置由属性height、width、top、right、bottom、left，加上为元素设置的边界、补白和边框联合定义。绝对定位元素可以有边界，但这些边界不压缩。

fixed

元素的定位方式同absolute一样，但它的包含块是视区本身。在屏幕媒体如web浏览器中，元素在文档滚动时不会在浏览器视窗中移动。例如，它允许框架样式布局。在页式媒体如打印输出中，一个固定元素会出现于每一页中的相同位置。这一点可用于生成流动标题或脚注。

inherit

这个值从其上级元素继承得到。更多的细节见第十章“CSS2展望”。

边偏移

上一节中描述的三个定位模式（relative、absolute和fixed）使用四个不同的属性来描述定位元素的边相对于其包含块的偏移。这四个属性稍后会被命名为边偏移（*side-offset*）属性，它们是使定位工作的重要部分。

top, right, bottom, left

允许值	<长度> | <百分比> | static-position | auto | inherit
初始值	auto
适用于	定位元素(即，position值不是static的元素)
可否继承	否

注意：百分比是指相对于包含块的宽度(right，left)或是包含块的高度(top，bottom)。

这些属性描述了从包含块最近的边开始的偏移（因而名为边偏移）。例如，top描述了定位元素的上沿应放置于距其包含块上沿多远处。对于top而言，正值会使定位元素的上沿向下移动，而负值会使它移到包含块上沿的上方。类似地，left描述定位元素的左外边沿相对于包含块的左边沿偏右（正值）或偏左（负值）多远。另一种认识的方式是正值产生向内的偏移，使边沿移向包含块的中心，负值则产生向外的偏移。

注意： 偏移外边沿的描述可以在勘误表上找到。最初CSS2规范实际上讲的是内容边沿是偏移的，但这被普遍认为是一个严重的错误，而且实际上，规范的其他部分也显示出外边沿是偏移。

偏移定位元素的外边沿隐含的意思是在定位元素的过程中，元素的所有部分都移动，包括边界、边框、补白及内容。换句话来说，有可能为定位元素设置边界、边框及补白。这些会同定位元素保持在一起，且包含于由边偏移属性定义的区域中。

还有两个边偏移属性值在这里要提到。第一个，static-position，使得用户代理将定位元素的某个边放置于未被定位时应出现的位置上。例如，考虑一个未被定位的元素，其上边沿距其包含块上边沿3em。如果元素被定位且其top值为static-position，则定位元素的上边沿位于距离包含块上边沿3em。在本章的稍后部分中，我们会看到它的用处。

另一个值是auto，它能产生一些更为有趣的效果。它与边界的auto设置很类似，但在定位中，它允许生成的元素刚好有显示其内容所需的宽度或高度，而不必明确指明将会有多高或多宽。本章稍后也有详述。

记住这一点很重要，即边偏移属性定义偏移是相对于包含块的相应边（例如，left定义从左侧开始的偏移），而不是包含块的左上角。这是为何填充包含块的右下角时使用下面的值：

```
top: 50%; bottom: 0; left: 50%; right: 0;
```

在这个例子中，定位元素的左外边沿放置于包含块的中线上，包含块宽度的二分之一是定位元素左侧相对于包含块左侧偏移。定位元素的外右边沿相对于包含块的右侧不偏移，它们是重合的。类似的推理也用在顶端与底端：外上边沿位于包含块的二分之一高度处，外下边沿同包含块底端重合，结果如图9-2所示。

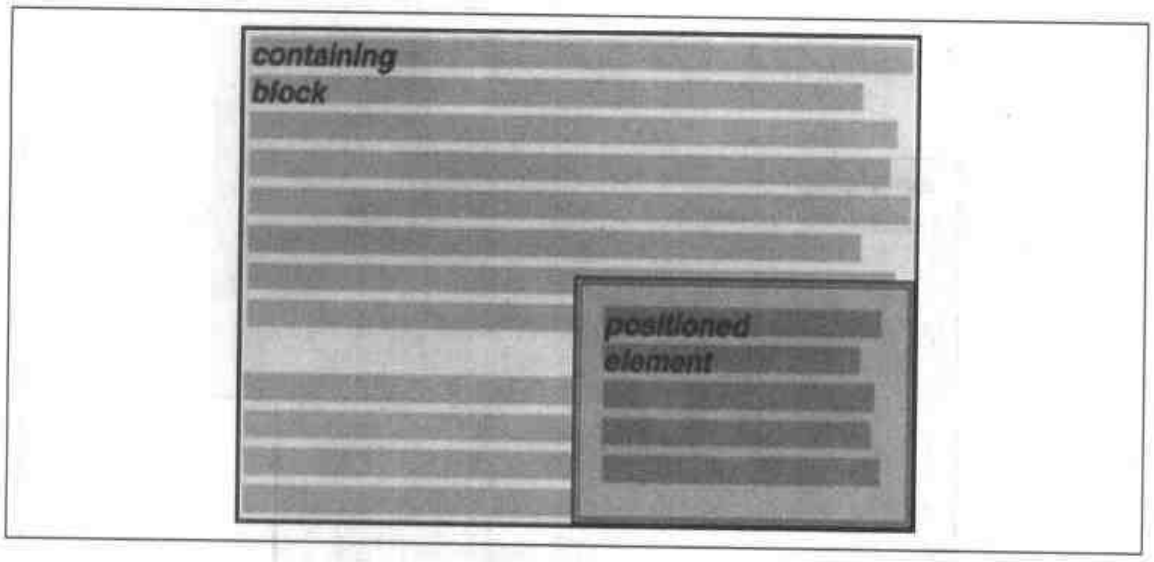


图9-2　在包含块中定位元素

注意： 图9-2所描述的以及本章中大多数的例子只在建立包含块的元素有显式声明的高度时才能够工作。这是因为规范指出，如果包含块的高度未显式指明——比如像一个普通段落那样，高度依赖于内容——则该包含块中任何定位元素的top和bottom均被视为auto来处理。

另外，即使没有显式指出，本节（及以下几节）中的例子都是基于绝对定位的。因为绝对定位是说明top、right、bottom及left如何工作的最简单模式。从这里开始，我们便坚持这一点。

注意定位元素有补白、双边框及一个略有差异的背景颜色。在图9-2中，它没有边界，但如果有的话，会在边框与偏移边沿之间生成空白区。这会使得定位元素看上去没有完全覆盖住包含块的右下角。实际上，是覆盖住了，但人的视觉却没感觉到。换句话来说，以下两种样式设置会有相同的可视外观，假定包含块为100em高，100em宽：

```
top: 50%; bottom: 0; left: 50%; right: 0; margin: 10em;
top: 60%; bottom: 10%; left: 60%; right: 10%; margin: 0;
```

当然，这种相似只限于视觉效果。

使用负边界可以将元素定位到其包含块之外。例如，以下的值会使结果如图9-3所示：

```
top: -5em; bottom: 50%; left: 75%; right: -3em;
```

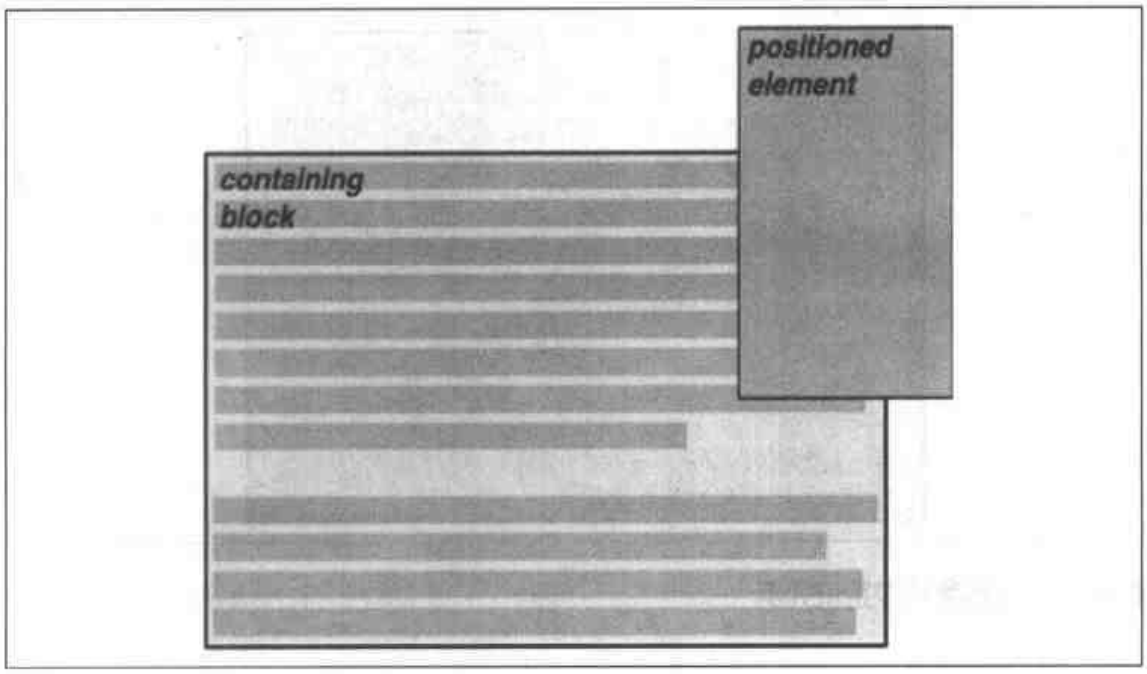


图9-3 使定位元素超出其包含块

现在让我们看一下，为什么省略width和height在定位中不总是件坏事，以及怎样声明它们会有益处。

宽度与高度

当决定定位元素于何处之后，很多情况下会希望前进一步并声明元素的高度与宽度。另外，可能在有的情况下会希望限制定位元素的高度与宽度，更不必说浏览器自动计算宽度、高度的情况。

设置宽度与高度

如果希望给定位元素设置特定的宽度与高度，很明显应该使用属性width和height。

width

允许值	<长度> | <百分比> | auto
初始值	auto
适用于	块级元素和替换元素
可否继承	否

注意：百分比是指相对于包含块的宽度。

height

允许值	<长度> | auto
初始值	auto
适用于	块级元素和替换元素
可否继承	否

尽管有时为元素设置宽度与高度很重要，但这在定位元素时不总是必须的。例如，如果元素四个边的位置使用top、right、bottom和left来描述，则宽度与高度由边的位置来决定。假如希望使用元素从顶端到底端填充其包含块的左半部分，可以使用以下样式，结果如图9-4所示：

```
top: 0; bottom: 0; left: 0; right: 50%;
```

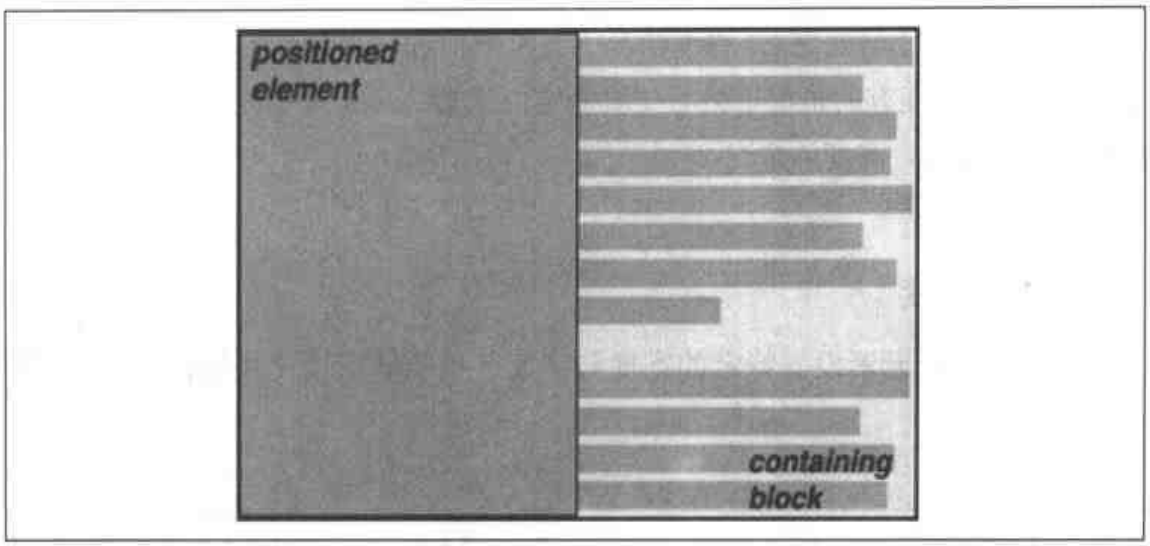


图 9-4 填充包含块的左半部分

由于width和height的缺省值均为auto，图9-4的结果同使用如下样式的结果完全一致：

```
top: 0; bottom: 0; left: 0; right: 50%; width: auto; height: auto;
```

假定希望定位一个元素与其包含块的右上角，宽度为其包含块的三分之一，高度只需满足显示内容的需要，如图9-5所示。

这是auto显示其自身作用的情况。达到该结果所需的样式为：

```
top: 0; bottom: auto; left: auto; right: 0; width: 33%; height: auto;
```

因为top设为0，且bottom和height设为auto，用户代理可以自由设置元素尺寸使之恰好可以显示它的内容，而不是比所需要的高。达要感谢计算绝对定位元素的高度与宽度的修订规则，该规则发布于初始规范的勘误表中。

注意设定一个显式宽度是有帮助的。如果用户代理知道元素有多宽的话，那么根据内容计算其高度就很容易。如果宽度也设为auto，那么用户代理将被迫为高度指定值。这个值可能会因用户代理的不同而不同。因此，指定一个希望的宽度比较好。

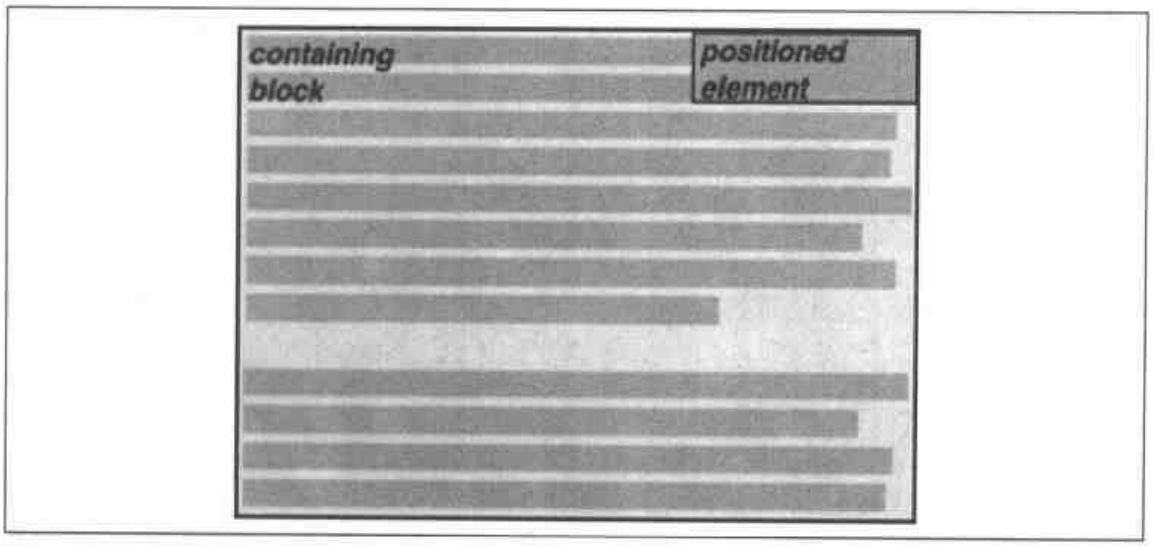


图 9-5 “压缩包装”（shink-wrapping）一个定位元素

当然，情况并非总是如此：也可以设定显式高度并让宽度伸缩以适应内容。这样：

```
top: 0; bottom: auto; left: auto; right: 0; width: auto; height: 10em;
```

无论如何，这里的元素的高度都为10em，但其宽度可以变化以精确适应其内容。这也是被称为“压缩包装”内容，因为它模仿了对盒子或其他产品进行压缩包装的行为。与塑料压缩包装紧紧包住包中内容的方式一样，定位元素也是如此，当然是在正确的样式条件下。

如果bottom设为一个实际的值 —— 百分比或长度 —— 则定位元素的高度将强制受限。作为一个示范，设定bottom为一指定值，结果如图 9-6 所示：

```
top: 0; bottom: 10%; left: auto; right: 0; width: 33%; height: auto;
```

在这种情况下，元素的height必须为包含块高度的90%，因为100% - 0% - 10% = 90%。当然，这里假定没有边界、边框及补白设置于定位元素上，否则，有效的height会减少，尽管整个元素（内容、补白、边界与边框）仍为包含块高度的90%高。

类似地，如果显式声明了高度但留下bottom为auto，那么会出现如图 9-7 所示的结果：

```
top: 0; bottom: auto; left: auto; right: 0; width: 33%; height: 45%;
```

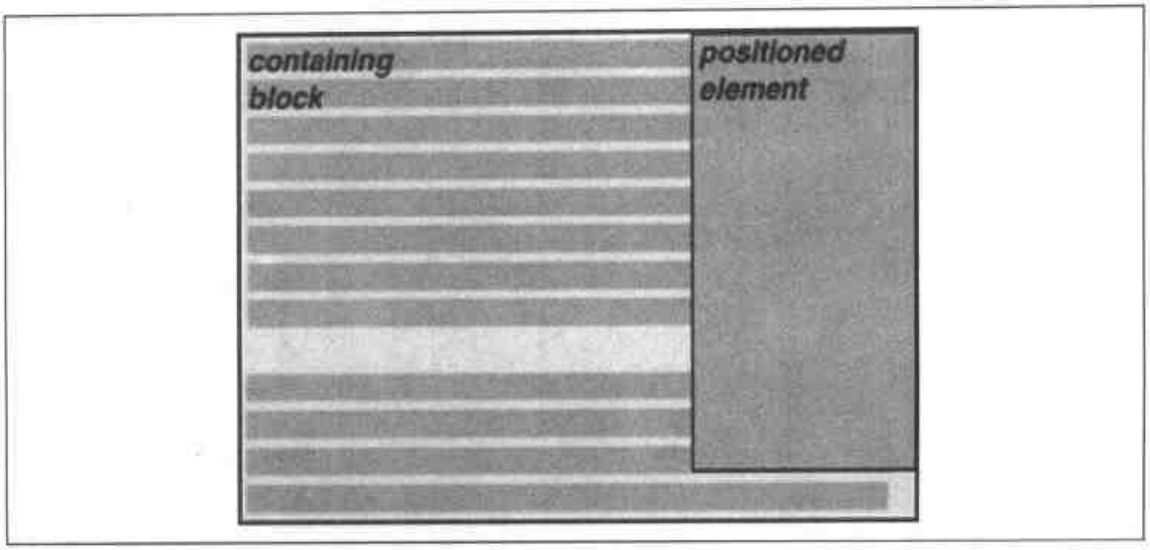


图 9-6 通过显式 bottom 定义高度

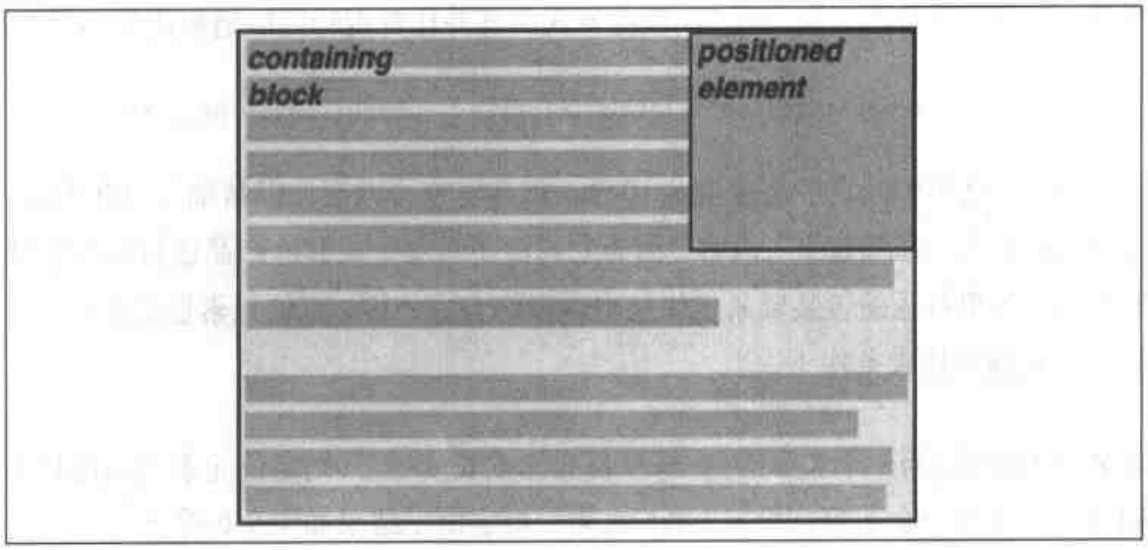


图 9-7 通过显式 height 定义高度

在这种情况下，定位元素底端的放置与声明`bottom:55%`的效果相同，因为100% - 0 - 45%=55%。

当然，很多同样的规则适用于宽度，例如：

```
top: 100px; bottom: 200px; left: 30%; right: 10%; height: auto; width: auto;
```

这里，元素的有效宽度为包含块宽度的60%。

尽管这很精彩，但也引起了一系列的问题。想像有一个定位元素，不希望它小于一个指定的大小，考虑如下样式：

```
top: 10%; bottom: 20%; left: 30%; right: 10%;
```

这样，其高度为包含块高度的70%，宽度为包含块宽度的40%。这个设置会执行得很好，但如果包含块只有50像素高，200像素宽时会如何呢？会生成一个元素，只有35像素高，80像素宽。留给内容显示的空间太少了，但如果设置遮盖了包含块的内容，高度和宽度为auto，元素会充满整个包含块，如本章稍后所示，可以选择强迫内容溢出元素。然而此时，我们只着眼于解决宽度与高度的问题。可以试着显式设置宽度与高度，如下所示：

```
top: 10%; bottom: 20%; left: 50%; right: 10%; width: 30em; height: 15em;
```

然而，这个方案看起来需要输入过多的内容，且在小的浏览环境，如手持设备中会有灾难性后果。况且，它将迫使显式声明高度与宽度，丧失了很多的灵活性。难道为宽度与高度的大小设一些限制不是更好吗？

限制宽度与高度

无论是必须还是愿意这样，可以使用以下的CSS2属性对元素的宽度与高度设一些限制，这些属性被称为*min-max*属性。

min-width

允许值	<长度> | <百分比> | inherit
初始值	与用户代理有关
适用于	除非替换内联元素和表格元素外的所有元素
可否继承	否

注意：百分比是指相对于包含块的宽度。

max-width

允许值	<长度> | <百分比> | none | inherit
初始值	与用户代理有关
适用于	除非替换内联元素和表格元素外的所有元素
可否继承	否

注意：百分比是指相对于包含块的宽度。

min-height

允许值	<长度> | <百分比> | inherit
初始值	0
适用于	除非替换内联元素和表格元素外的所有元素
可否继承	否

注意：百分比是指相对于包含块的高度。

max-height

允许值	<长度> | <百分比> | none | inherit
初始值	none
适用于	除非替换内联元素和表格元素外的所有元素
可否继承	否

注意：百分比是指相对于包含块的高度。

这些属性的名字使它们很好理解。这里是上节中例子的一个解决方法：

```
top: 10%; bottom: 20%; left: 50%; right: 10%;
  min-width: 20em; min-height: 30em;
```

当然，这不是一个很好的解决办法，因为它强迫元素至少为20em宽，30em高。还有一个较好的解决方法：

```
top: 10%; bottom: autoleft: 50%; right: 10%; min-width: auto; min-height: 15em;
```

此时，我们得到了一个宽度为包含块宽度的40%且不会小于15em的元素。同时还改变了bottom和height使它们可以自动得到。这会使元素得到显示内容所必要的高度，而不论它有多窄（当然不会少于15em）。

反之，也可以使用max-width和max-height防止元素过宽或过高。考虑以下一种情况，因奇怪的原因，希望元素有其包含块的四分之三宽，但当达到400像素时要停止变宽。可行的样式是：

```
left: 0%; right: auto; width: 75%; max-width: 400px;
```

min-max属性的最大优点是相对安全地混合使用各种单位。可以在设置基于百分数大小的同时设置基于长度的限制，反之亦然。

注意：min-max属性在与浮动元素联合时很有用。例如，可以使浮动元素的宽度相对于其上级元素来设置（上级元素是其包含块），同时确保浮动元素宽度不会小于10em。反向方法也是可能的：

```
P.aside {float: left; width: 40em; max-width: 40%;}
```

这会将浮动元素设为40em宽，除非40em超过包含块宽度的40%，若超出了，则浮动元素会变窄。

当然，还可以使用这些属性防止元素超出一定的大小，如下：

```
max-height: 30em; max-width: 20em;
```

问题是如果元素的内容不能够完全适合指定元素的大小时，是切去边框呢，还是溢出定位元素？这是下一节将讲述的内容。

内容溢出及剪切

如果元素内容相对于元素大小来说太多，则有溢出元素自身的危险。在这种情况

下有一些选择，CSS允许从中挑选一种，还可以定义一个剪切区以决定元素之外的区域有多少时将溢出，并给出一个剪切掉元素溢出部分的方法。

溢出

假定不论何种原因，元素被指定为一个特定的大小，而内容不适合这个大小。可以使用overflow属性来控制这种情况。

overflow

允许值	visible | hidden | scroll | auto | inherit
初始值	visible
适用于	块级元素和替换元素
可否继承	否

这个属性只应用于以下几种情况中：

- 当元素有负边界时。
- 行框必须宽于其上级元素的内容区，可能因为出现异常的长单词，不能换行的文本如PRE文本，或其他换行不被允许的情况。
- 当块级元素比它的上级元素内容区宽时。
- 当元素框高于其上级元素显式声明的高度时。
- 当元素被绝对定位时。

缺省值visible表示内容在元素框之外的部分为可见。典型地，这会导致内容溢到元素框之外，但不会改变框的形状，以下的样式会生成如图9-8的结果：

```
DIV#sidebar {position: absolute; top: 0; left: 0; width: 25%; height: 7em;
  overflow: visible;}
```

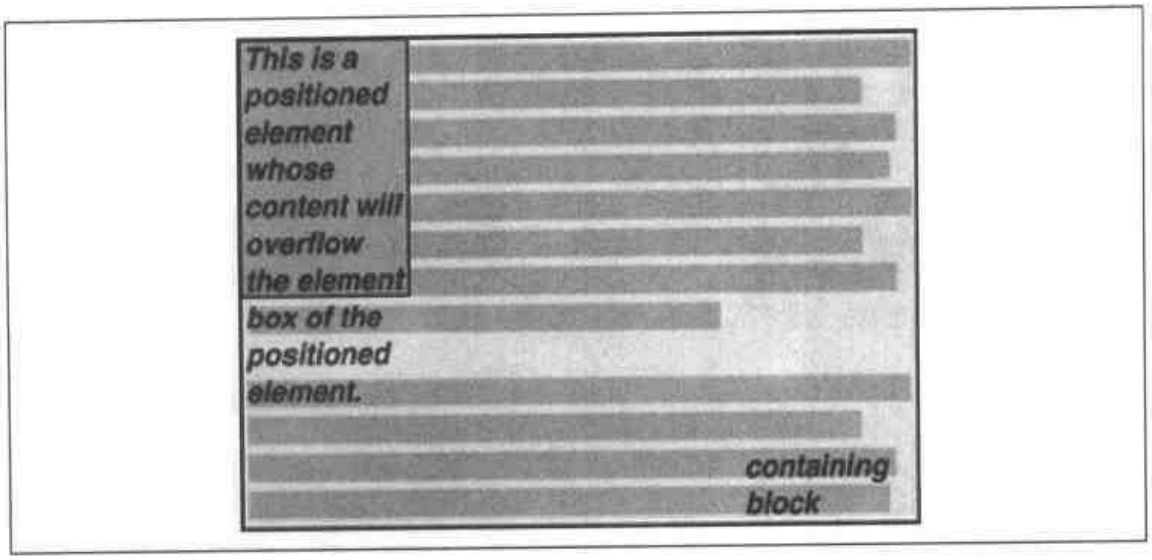


图9-8 溢出元素的内容区

注意：规范没有指明可视溢出的内容是否可以重叠其他元素的内容，但从推理结果来看应该是可以的。由于定位元素可以重叠其他元素，我们可推断它支持定位元素的内容应得到相同的处理。

如果overflow设为scroll，元素的内容被剪切而不可见，但提供了使额外内容可见的方式。在Web浏览器中，这代表一个滚动条（或一套滚动条）或其他策略使得无需改变元素自身的形状来获取内容。一种可能描述于图9-9，由以下样式生成：

```
DIV#sidebar {position: absolute; top: 0; left: 0; width: 15%; height: 7em;
  overflow: scroll;}
```

如果使用了scroll，则扫视机制（例如滚动条）应总是被绘制出。引用规范中的话，“这避免了在一个动态环境中滚动条显示或消失的问题。”这样，即使元素有足够的空间来显示其全部内容，滚动条仍会显示出来。另外，在打印一页或显示文档于页式媒体时，内容的显示应该如同overflow的值被声明为visible一样。

如果overflow设为hidden，元素的内容将被剪切，但没有机制提供给用户来获取内容。考虑如下样式：

```
DIV#sidebar {position: absolute; top: 0; left: 0; width: 15%; height: 7em;
  overflow: hidden;}
```

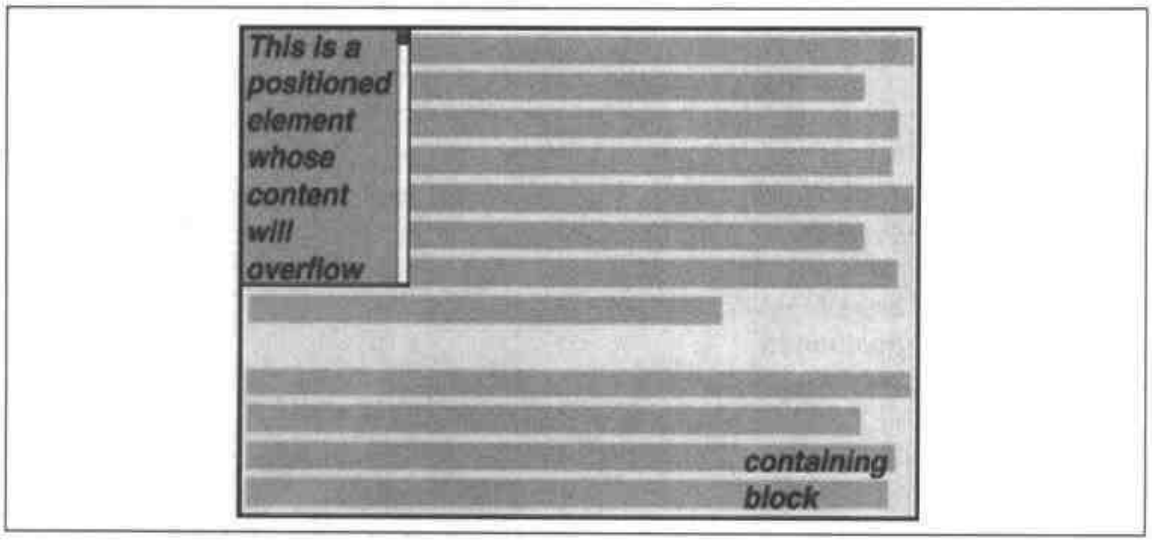


图 9-9 为溢出引入滚动条

在这种情况下，用户无法获取被剪切的内容。这会导致如图 9-10 所示的情况。

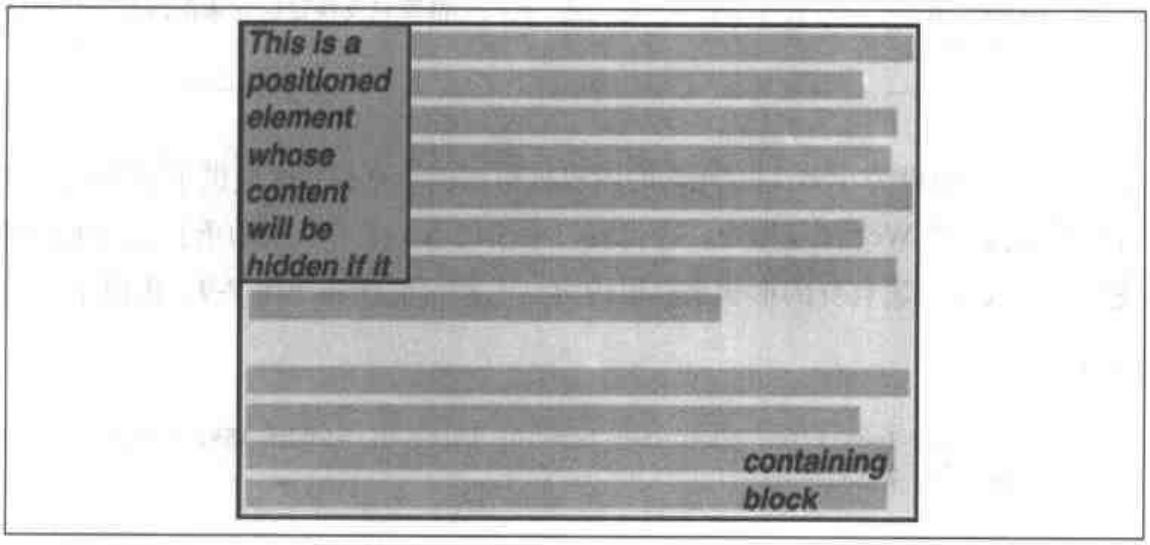


图 9-10 溢出时剪切内容

最后，是 overflow:auto。这种设置允许用户代理来决定采用哪种行为，尽管鼓励在必要时提供滚动机制。这是潜在的使用 overflow 的有用的方法，因为用户代理可以解释它为“只在需要时提供滚动条。”（它们也许不会这样解释，但肯定可以，且非常可能这样）。

最简单的情况是，定位元素的剪切区是元素自身的内容区，如图9-10所示。然而，也许会希望改变剪切区，这是下一节中的内容。

溢出剪切

在元素内容溢出元素框的情况下，且overflow已被设置为内容应被剪切，可以使用overflow-clip属性来改变剪切区的形状。

overflow-clip	
允许值	rect(<上>, <右>, <下>, <左>) | auto | inherit
初始值	auto
适用于	块级元素和带溢出值而不可见的替换元素
可否继承	否

缺省值auto，代表剪切区应该与元素的内容区有相同的大小和位置。另一种可能是根据元素内容区来定义剪切形状。这不会改变内容区的形状，而是改变内容可以绘制的区域。

注意：尽管CSS2可用的剪切形状只有矩形，规范仍提供了在将来的规范中包含其他形状的可能。

这由使用形状值rect(*top*, *right*, *bottom*, *left*)完成。可以指定剪切区不发生变化，如下所示：

```
overflow-clip: rect(0, auto, auto, 0);
```

这与声明overflow-clip: auto没有区别。当然，改变剪切区会更有趣。例如：

```
DIV#sidebar {position: absolute; top: 0; left: 0; width: 5em; height: 7em;
  overflow: hidden; overflow-clip: rect(0.5em, 4em, 0.5em, 1em);}
```

这种设置使剪切区从顶端与底端向内拓展二分之一em，从左侧和右侧向内拓展1em。导致如图9-11所示的结果，增加了一条长划线以描述剪切区的边沿。这条线在用户代理绘制文档时不出现。

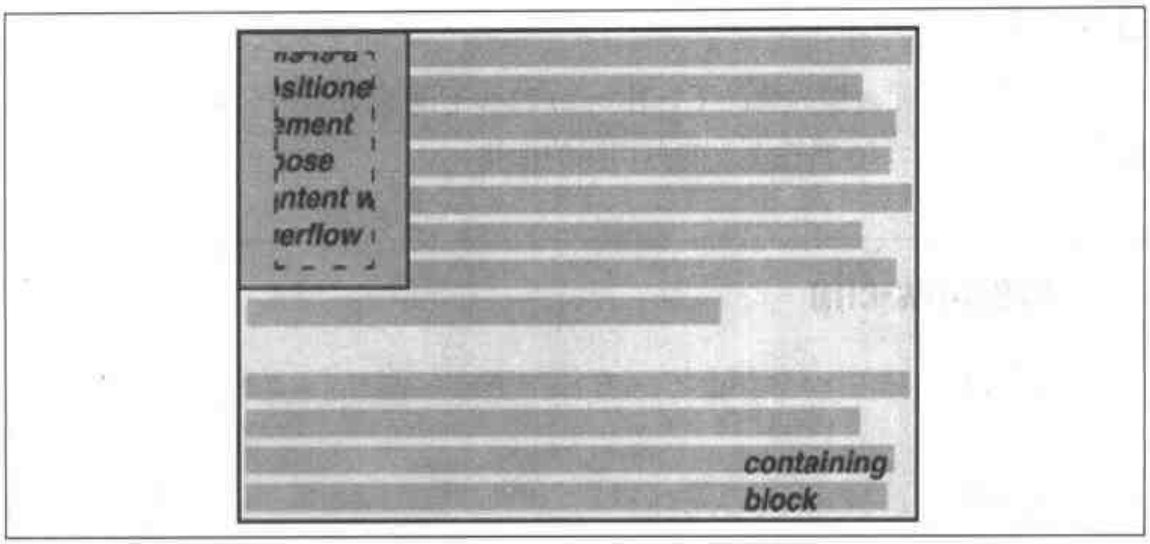


图 9-11 收缩剪切区

rect 的语法十分有趣。从技术上来说，它可以是 rect(*top*, *right*, *bottom*, *left*)，注意逗号，但 CSS2 规范既包括有逗号的例子，也包括没有逗号的例子，且定义 rect 为能接受有逗号及没有逗号的版本。本书中采用有逗号的版本，主要是由于这样易于阅读。

重要的一点是注意 rect(...)的值不是边偏移，而是到元素左上角的距离。这样，一个位于元素左上角的 20 像素 × 20 像素的剪切矩形可以定义为：

```
rect(0, 20px, 20px, 0)
```

rect(...)中允许的值只有长度值和 auto，auto 与“设置剪切边沿为恰当的内容边沿”一样，这样，以下的两个声明实际上是一样的：

```
overflow-clip: rect(auto, auto, 10px, 1cm);
overflow-clip: rect(0, 0, 10px, 1cm);
```

可以设置负长度，这样会延伸剪切区到元素框之外。如果希望将剪切区向上和向左推动四分之一英寸，可以通过使用以下的样式来完成（见图 9-12）：

```
overflow-clip: rect(-0.25in, auto, auto, -0.25in);
```

如图所示，这并没有什么好处。剪切矩形向上及向左扩展，但由于那里没有任何内容，视觉效果与网页制作者声明 overflow-clip: auto 一致。

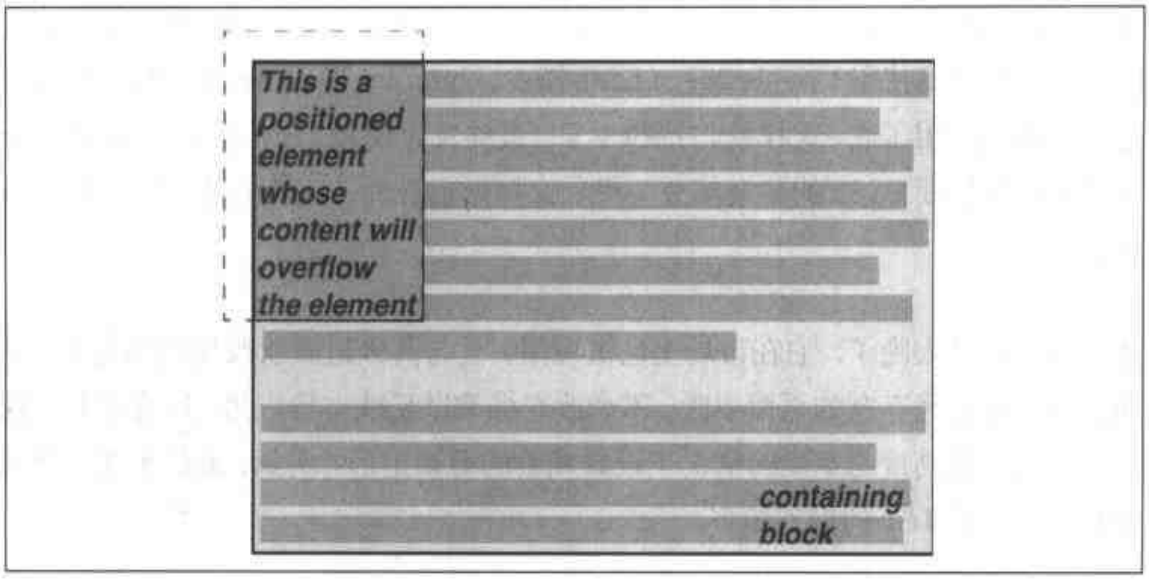


图 9-12 扩展剪切区

另一方面，越过底端和右边沿会很好，而不是顶端和左边沿。图9-13显示了以下样式的结果（长划线只是为了便于说明）：

```
DIV#sidebar {position: absolute; top: 0; left: 0; width: 5em; height: 7em;
overflow: hidden; overflow-clip: rect(0.6em, 6em, 9em, 0);}
```

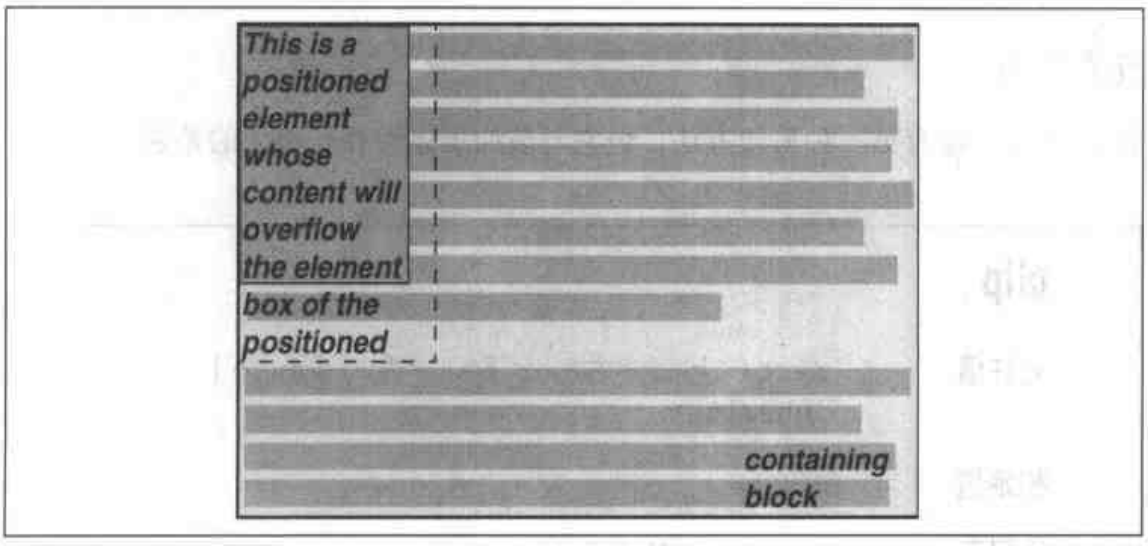


图 9-13 扩展剪切区

这就扩展了内容可视的区域。然而，它不会改变文档流，因此唯一的视觉影响是可以看到元素下方的更多内容。文本不会流到右方，因为行框的宽度仍受限于定位元素的宽度。如果有一个宽于定位元素的图片，或有一很长的行的预先格式文本，则可能会在定位元素之右为可见，直至剪切矩形结束的地方。

也许读者已经意识到了，rect(...)的语法不太完整，它是基于较早的定位部分的草案，该草案使用top-left-offset模式。Internet Explorer在CSS2的完整建议之前就应用了它，这样就与rect(...)最终决定使用side-offset模式发生了冲突，就如同CSS2的其余部分一样。这样做很合理，因为它使得定位与自身相关。

然而那时已经太晚了：它在市场上已被应用，且与其强迫微软改变其浏览器，从而潜在地使已经存在的页面中断，不如改变标准以反映市场应用。这意味着无法在未预先定义高度与宽度的情况下设置相关的剪切矩形。例如，无法生成一个比元素内容区大1em的剪切矩形：

```
position: absolute; top: 0; bottom: 50%; right: 50%; left:0;
```

因为无法知道元素的宽度或高度为多少em，因而无法生成一个中止于元素内容区右侧1em处，或下方1em处的剪切矩形。

另一个复合问题是rect(...)只接受长度单位与auto。增加百分比值为rect(...)的有效值会取得较大的改进，将来的CSS有希望加入这个能力。

元素剪切

在CSS中，还有另一种剪切方法，但它与前面所述的存在很大的差别。

clip

允许值	rect(<上>, <右>, <下>, <左>) | auto | inherit
初始值	auto
适用于	块级元素和替换元素
可否继承	否

这个属性可以使用简单的“与”操作来剪切元素。包含于clip矩形之内的内容显示，之外的部分则不显示。另外，clip矩形是关于元素的外边沿设置的，而不是内容边沿。这样，如果希望（无论什么原因）剪切图片上方的10个像素：

```
<IMG SRC="foo.gif" STYLE = "clip:rect (10px,auto,auto,0);">
```

auto值会使得剪切矩形的底端与图片的底端对齐，右边沿与图片右边沿对齐。left的值0使剪切矩形的左边沿与图片的左边沿重合，而top值10px使剪切矩形的上边沿下移10像素。这会使图片上方的10像素效果上成为不可见。

clip可以应用于任何元素。这样，使用如下标记可以确保显示一个段落的左上角：

```
<P STYLE="clip: rect(0, 10em, 10em, 0);">
```

这会显示一个10em高，10em宽的方块。这个方块画于外左上角，因此任何边界，边框及补白会影响元素有多少为可见及多少被剪切掉。

元素可见性

在所有剪切与溢出之外，还可以控制整个元素的可见性（visibility）。

visibility

允许值	visible | hidden | collapse | inherit
初始值	inherit
适用于	所有元素
可否继承	否

这很容易。如果元素设置为visibility: visible，则它是可见的。

然而，如果元素设为visibility: hidden，它就成为“不可见”（使用规范中的语句）。在不可见状态，元素仍会影响文档的布局，就如同它们可见一样。换句话来说，元素仍在那里：只是无法看到它。注意它与display: none的区别。在后者中，元素未被显示且从文档中完全移走，因而对文档布局没有任何影响。图9-14显示了一个基于以下样式和标记的文档，其中有一个EM元素已被设置为hidden：

```
EM.trans {visibility: hidden; border: 3px solid gray; background: silver;
  padding: 1em;}

<P> This is a paragraph which should be visible. Lorem ipsum, dolor sit amet,
<EM CLASS="trans">consectetuer adipiscing elit, sed diam nonummy nibh </EM>
euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
</P>
```

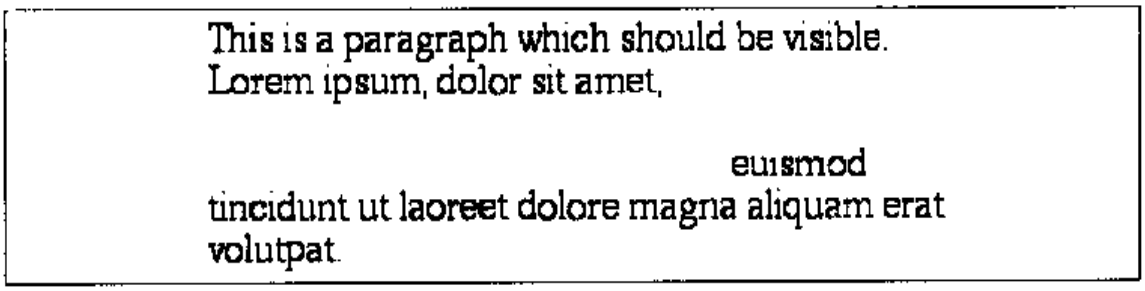


图9-14 隐藏一个元素

元素的任何可见部分，如内容、背景、边框都可成为不可见。注意空间仍存在于那里，因为元素仍是文档布局的一部分。我们只是不能见到它。

注意有可能将hidden元素的后辈元素设为visible。这会使得元素像通常那样显示，除了其祖先元素（可能是兄弟）不可见外。为了这样做，需要显式声明后辈元素为visible，因为visibility是可继承属性。这样：

```
P.clear {visibility: hidden;}
P.clear EM {visibility: visible;}
```

visibility: collapse用于CSS表格绘制，这部分内容未包含在本书中，因为本书写成时，其应用仍不广泛。根据CSS2规范，collapse如果用于非表格元素则意义与hidden相同。从语义角度来看，这有些令人疑惑（由于collapse听上去应触发一些行为，如display: none所示），但并不是那样。

相对定位

定位模式中最易于理解的是相对定位。在这种模式中，定位元素使用边偏移属性来移动。然而，这可能会带来一些有趣的效果。

从表面上来看，这很容易。假定希望将图片向左上方移动。图9-15显示了样式的结果：

```
IMG {position: relative; top: -20px; left: -20px;}
```

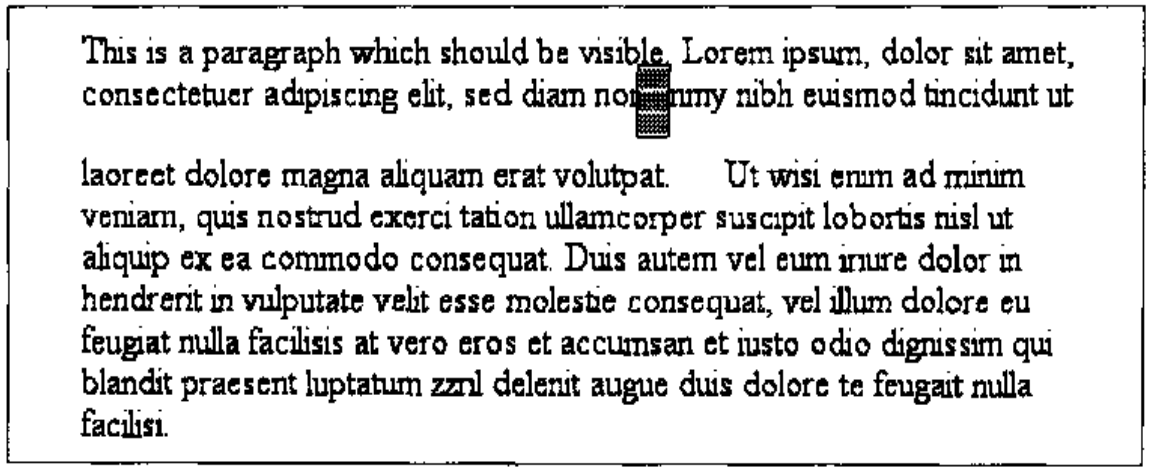


图 9-15 相对定位元素

所做的就是将图片的上沿向上偏移20像素，且将其左边向左偏移20像素。然而，注意图片先前定位位置的空白。那个空白之所以存在，是因为当元素相对定位时，它从它通常的位置移走，但它曾占有的空间并不消失。考虑如下样式的结果，如图 9-16 所示：

```
EM {position: relative; top: 8em; color: gray;}
```

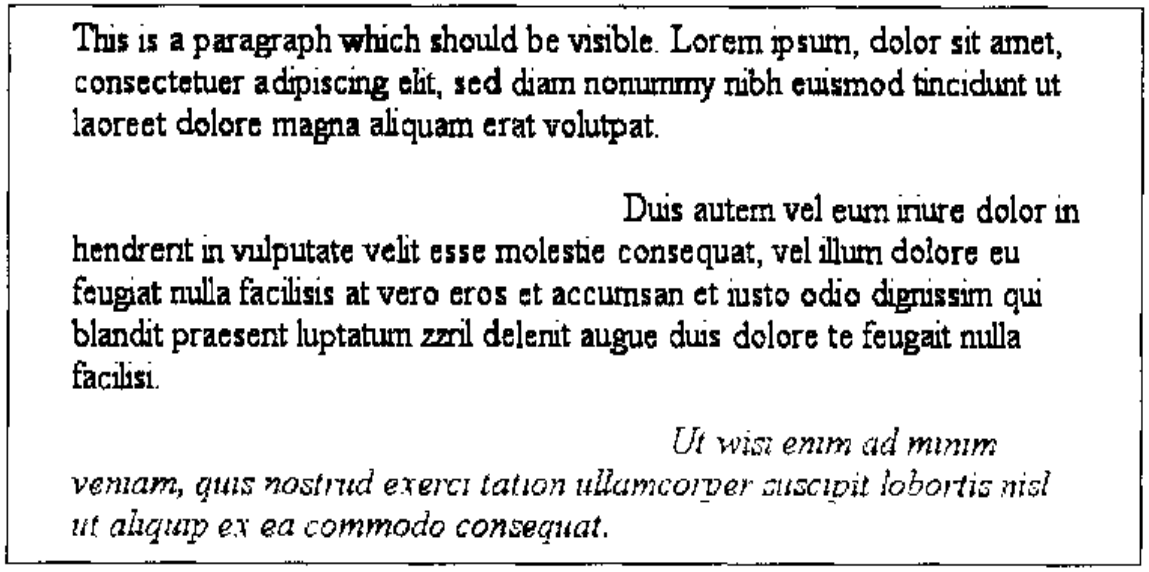


图 9-16 相对定位 EM 元素

可以看到，段落中有一些空白区域。那里是 EM 元素应该出现的位置，且 EM 元素在新位置的布局精确镜像它所空出来的空间。

这是因为一个相对定位元素的包含块在position为static的情况下，是它本应占据的空间。这一点很值得注意，因为有人可能会期望包含块由上级元素定义。实际上，相对定位元素设置其自身的包含块，然后相对于那个上下文偏移自身。

当然，也可以移动一个相对定位元素以重叠其他元素。例如，以下的样式和标记会产生如图9-17所示的结果：

```
EM {position: relative; bottom: -0.5em; color: gray;}
B {position: relative; bottom: 0.5em; color: gray;}

<P>This is a paragraph which should be visible. Lorem ipsum, dolor sit amet,
consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut
laoreet dolore magna aliquam erat volutpat.
<EM>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper
suscipit lobortis nisl ut aliquip ex ea commodo consequat.</EM>
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse
molestie consequat, vel illum dolore eu feugiat nulla <B>facilisis at vero
eros et accumsan et iusto odio dignissim qui blandit praesent luptatum</B>
zzril delenit augue duis dolore te feugait nulla facilisi.</P>
```

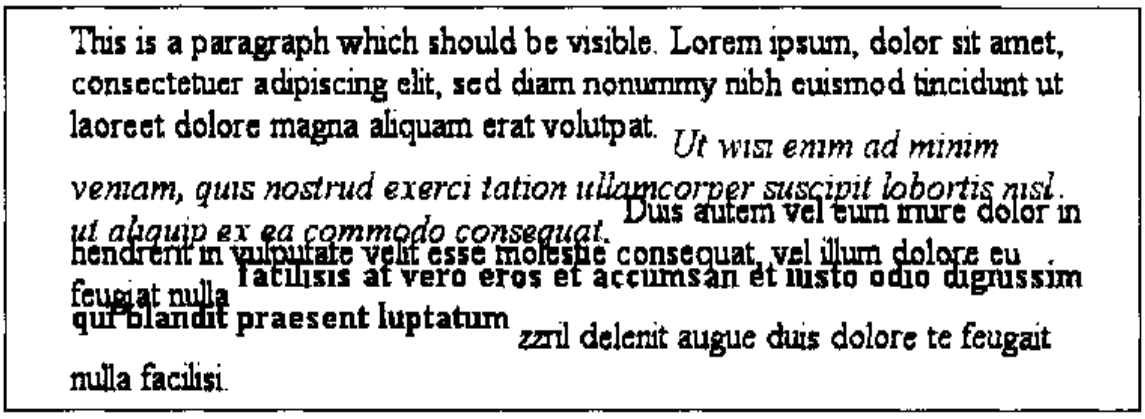


图9-17 两个相对定位元素

当相对定位一个元素时，它立即为每个下级元素生成一个新的包含块。这个包含块与元素定位的位置相关。这样，可以相对一个元素的上级元素来定位它，使自身被相对定位。图9-18显示了如下样式和标记的结果。

```
P {color: gray;}
EM {position: relative; bottom: -0.75em; color: black;}
B {position: relative; bottom: 0.5em; left: 1em; color: black;}

<P>This is a paragraph which should be visible. Lorem ipsum, dolor sit amet,
consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut
```

```
laoreet dolore magna aliquam erat volutpat.
<EM>Ut wisi enim ad minim veniam, <B>quis nostrud exerci tation ullamcorper
</B> suscipit lobortis nisl ut aliquip ex ea commodo consequat.</EM>
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse
molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros
et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril
delenit augue duis dolore te feugait nulla facilisi.</P>
```

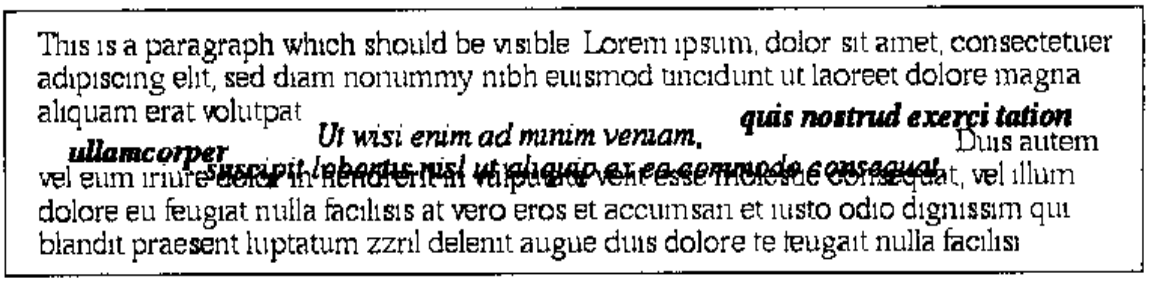


图 9-18 嵌套相对定位

强调文本相对于它正常时出现的位置下移0.75em，如同被期望的那样。加粗文本向左移动了 1em 且向上移动了二分之一 em，但它的移动是相对于突出文本被移动之后的位置来进行的。

相对定位有一个有趣的缺陷：当相对定位元素被过多约束时会怎样呢？例如：

```
EM {position: relative; top: 1em; bottom: 2em;}
```

这里我们的值会引起两种不同的行为。如果只考虑 top: 1em，则元素应向下移动 1em，但 bottom: 2em清楚地要求元素向上移动 2em。

最初的CSS2规范没有指明这种情况应如何处理。在本书写成时，有勘误表声明当相对定位被过多约束时，一个值应被重设为另一个值的相反数。这样，bottom 总是等于负 top，且right将等于负 left。这意味着上例将会按照下面的标记处理：

```
EM {position: relative; top: 1em; bottom: -1em;}
```

这样，元素会向下移动 1em。这个建设的变化也允许用于不同的书写方向上。它声明在相对定位中，如果是从左至右的语言，则 right 总等于 -left，而在从右至左的语言中，则反过来，left 总等于负 right。

绝对定位

由于本章中大多数图和例子（除上前面几节）都是绝对定位的情况，实际上已涉及到它的工作原理了。未被了解的只是当引入绝对定位时处理的细节。

当一个元素绝对定位时，它会完全从文档流中移开。然后根据它的包含块定位，边沿旋转使用边偏移属性。定位元素不围绕其他元素内容流动，其他内容也不围绕定位元素流动。这使得绝对定位元素可以覆盖其他元素，也可以被其他元素覆盖。（本章结尾处有如何影响覆盖顺序方面的内容）。

记住一个绝对定位元素的包含块不必是它的上级元素。实际上，经常不是那样，除非网页制作者采用措施来修正这种情况。幸运的是，这很容易做到。只需选择希望成为绝对定位元素的包含块的元素，然后将其position设为relative，且偏移为0。这样：

```
P.contain{position:relative;}
```

考虑图9-19的例子。其中显示了两个内容相同的段落。然则，第一个段落中包含一内联加粗元素，第二个包含一个绝对定位的加粗元素。在第二个段落中，使用的样式如下：

```
P {position: relative;}   /* establish containing blocks */

<B STYLE="position: absolute; top: auto; right: 0; bottom: 0; left: auto;
 width: 8em; height: 4em;">...</B>
```

两个段落中的大部分文本看上去很普通。然而，在第二段中文本本应出现的空间被封闭，且定位文本覆盖了一些内容。没有办法避免它的发生，如果没有将加粗文本定位到段落之外（使用负的right值）或者未对段落指定一个足够容纳定位元素补白的宽度。同时，由于其背景为透明，上级元素的文本透过定位元素显示出来。避免该情况的唯一方法是给定位元素设置背景。

注意在这种情况下，加粗元素是根据上级元素的内容框来定位的，内容框定义了它的包含块。没有上级元素的相对定位，包含块将是其他的元素。

考虑被定位元素是BODY元素的下级元素的情况，例如，一个段落或标题元素。使

图9-19 绝对定位的效果

用正确的样式，定位元素的包含块将是整个BODY元素。这样，在BODY及文档的第五段上使用如下样式会导致类似图9-20所示的情况：

```
BODY {position: relative;}

<P STYLE="position: absolute; top: 0; right: 25%; left: 25%; bottom: auto;
 width: 50%; height: auto; background: silver;">...</P>
```

段落现在定位于文档的起始处，宽度为文档宽度的一半且覆盖了开始的几个元素。

另外，如果文档滚动，段落也将随之滚动。这是因为元素的包含块是BODY元素的区，而不是视区（viewport）。如果希望定位元素使它相对于视区放置且不随文档的其余部分滚动，我们会在下一节解释如何做。

还有一些有趣的事情，记住绝对定位框可以有背景、边界、边框及补白，也可以使用样式，如同对其他元素一样。这使得它们在生成滚动条，“条状注意”及其他类似效果时很有用。一个例子是当段落被编辑后生成一个“改变标记”的能力。这可以使用如下的标记与样式来完成：

```
SPAN.change {position: absolute; top: 0; left: -5em; width: 4em;
  font-weight: bold;}
P {margin-left: 5em; position: relative;}
```

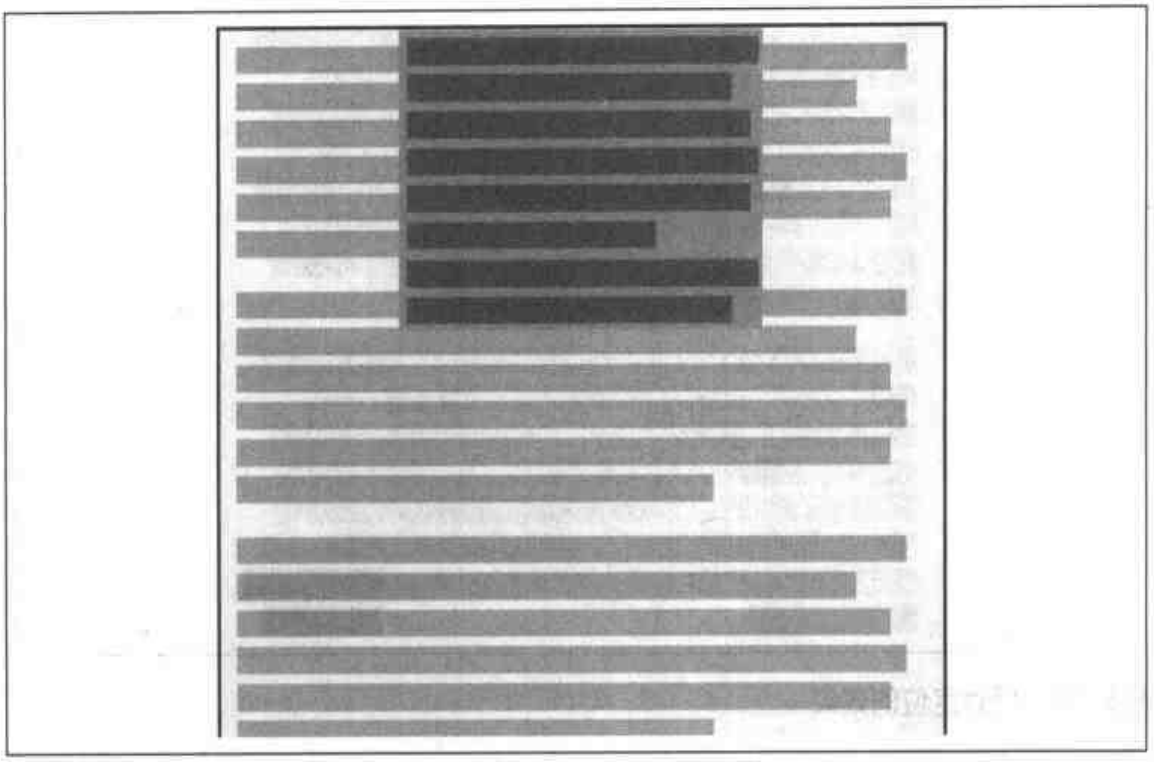


图 9-20 一个绝对定位段落

```
<P> Lorem ipsum, dolor sit amet, consectetuer adipiscing elit,
sed diam nonummy nibh euismod tincidunt ut <SPAN CLASS="change">***</
SPAN> laoreet dolore magna aliquam erat volutpat.</P>
```

尽管这的确依赖于插入额外的元素。但它仍有一些优点：SPAN可以放置于段落的任何位置，并有如图 9-21 所示的结果。

*** Lorem ipsum, dolor sit amet, consectrtuer adipiscing elit, sed diam nomummy nibh euismod tincidnut ut laoreet dolore magna aliquam erat volutpat.

图 9-21 用绝对定位设置“改变条”

然而，也许你希望将修改标记放在被改变的行之后。在这种情况下，只需对样式做小的修改，得到如图 9-22 所示的结果：

```
SPAN.change {position: absolute; top: static-position; left: -5em; width: 4em;
  font-weight: bold;}
```

```
P {margin-left: 5em; position: relative;}

<P> Lorem ipsum, dolor sit amet, consectetuer adipiscing elit,
sed diam nonummy nibh euismod tincidunt ut <SPAN CLASS="change">***</SPAN>
laoreet dolore magna aliquam erat volutpat.</P>
```

Lorem ipsum, dolor sit amet, consectrtuer adipiscing elit, sed diam nomummy nibh *** euismod tincidnut ut laoreet dolore magna aliquam erat volutpat.

图9-22 定义“改变条”的另一方案

还记得本章中何时曾提及static-position吗？下面的例子解释了它是如何工作的，以及它为什么非常有用。

另外，重要的一点是当元素被定位时，它为其后辈元素生成了包含块。可以绝对定位一个元素，然后再绝对定位它的后辈，如图9-23所示。

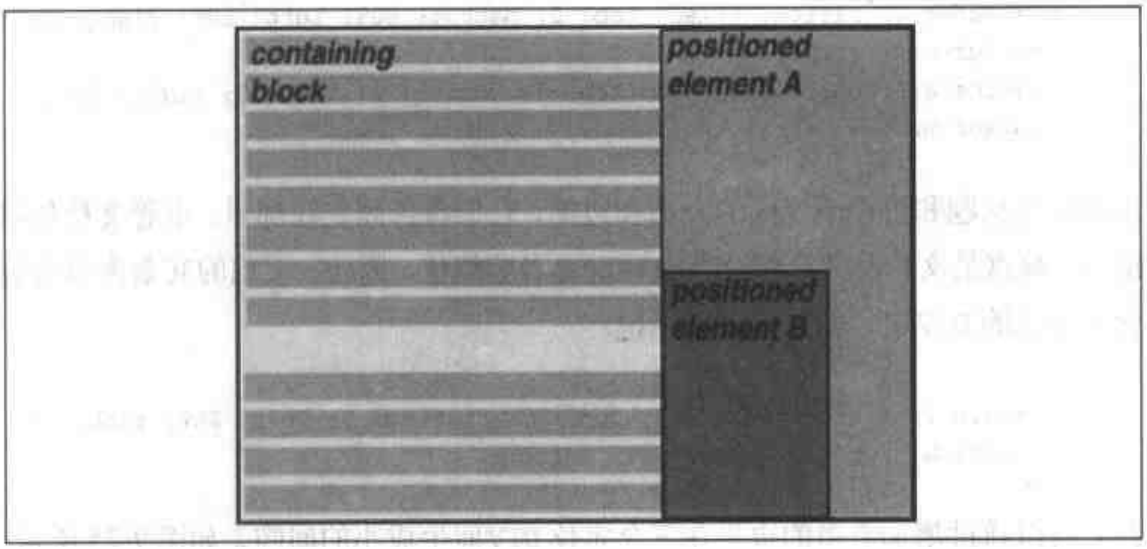


图9-23 嵌套绝对定位元素

位于元素A左下角的小框B是A的后辈，A又是相对定位的DIV元素的后辈。B被绝对定位，同A元素一样。使用的样式如下：

```
DIV {position: relative;}
P.A {position: absolute; top: 0; right: 0; width: 15em; height: auto;
```

```
    margin-left: auto;}
P.B {position: absolute; bottom: 0; left: 0; width: 10em; height: 50%;
    margin-top: auto;}
```

时刻要记住：只有定位元素会为后辈元素建立包含块。前面我们曾经讲到过这一点，但它是如此基础、如此重要，因此我们在这里再重复一次。

固定定位

如同前面节中隐含的，除了固定定位的包含块总是视区外，固定定位与绝对定位十分类似。在这种情况下，元素完全从文档流中移开，而且不跟文档的任何部分存在位置相关的关系。

可以用很多有趣的方法来使用它。首先，可以使用固定定位生成框架样式界面。图 9-24 显示了一个很普通的布局模式。

可使用以下样式来完成：

```
DIV#header {position: fixed; top: 0; bottom: 80%; left: 20%; right: 0;
  background: gray;}
DIV#sidebar {position: fixed; top: 0; bottom: 0; left: 0; right: 80%;
  background: silver;}
```

这会固定标题和边栏于视区的顶端和边侧，它们将会固定在那里，不管文档如何滚动。缺点是文档的其余部分会被固定元素遮挡住。因此，文档的其余内容应包含在自己的 DIV 中，并使用如下标记：

```
DIV#main {position: absolute; top: 20%; bottom: 0; left: 20%; right: 0;
  overflow: scroll; background: white;}
```

甚至可以通过增加恰当的边界在三个定位 DIV 间生成小的间隙，如图 9-25 所示：

```
BODY {background: black; color: silver;}  /* colors for safety's sake */
DIV#header {position: fixed; top: 0; bottom: 80%; left: 20%; right: 0;
  background: gray; margin-bottom: 2px; color: yellow;}
DIV#sidebar {position: fixed; top: 0; bottom: 0; left: 0; right: 80%;
  background: silver; margin-right: 2px; color: maroon;}
DIV#main {position: absolute; top: 20%; bottom: 0; left: 20%; right: 0;
  overflow: scroll; background: white; color: black;}
```

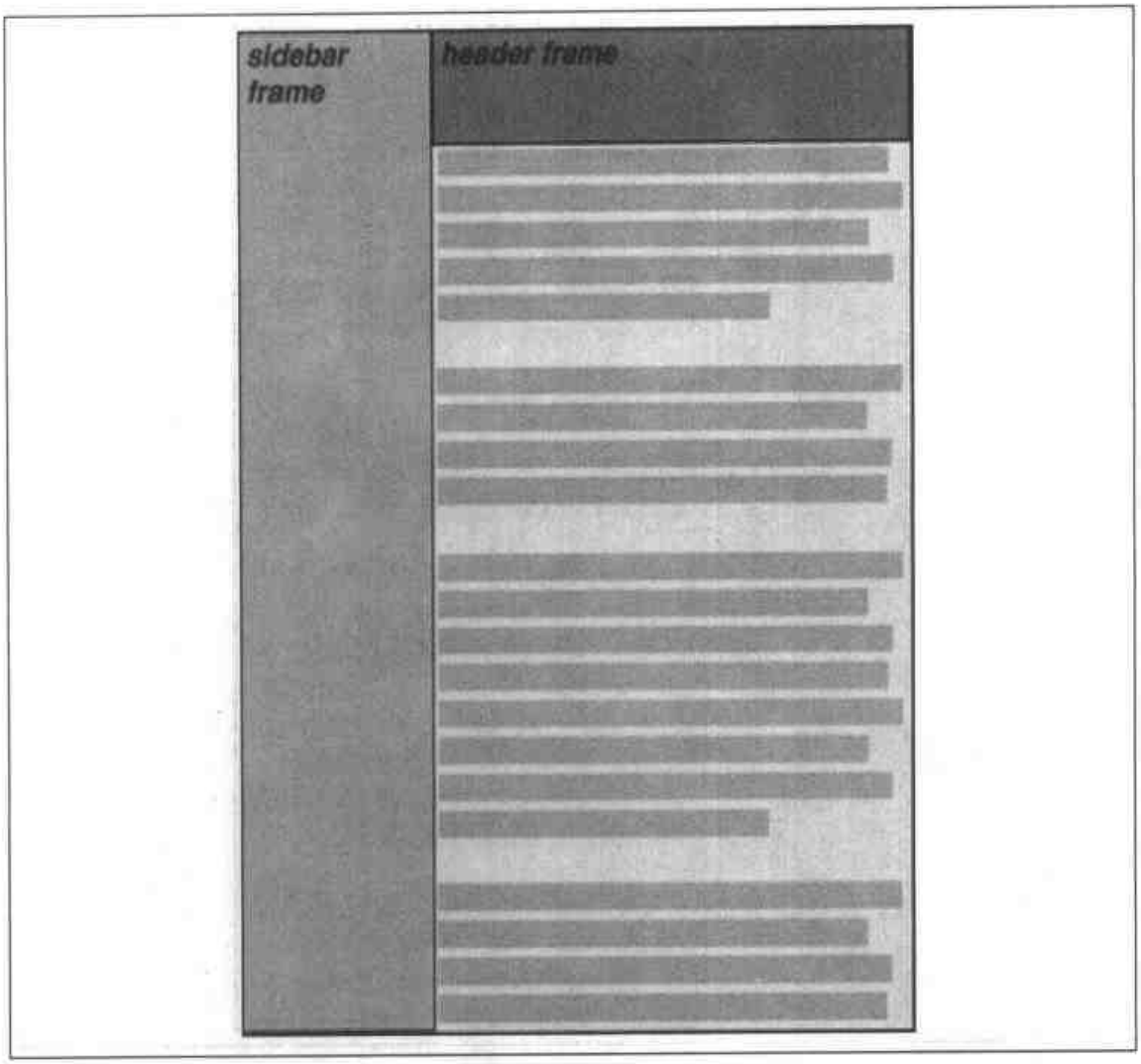


图 9-24 使用固定定位仿真框架

在这种情况下，一个平铺图片可应用于BODY元素的背景。这个图片可以通过由边界生成的间隙显示出来，当然，这个间隙在网页制作者觉得合适的情况下被加宽。如果背景图片无关紧要，则可以对DIV应用简单的边框以取代边界。

层叠定位元素

在使用这些定位时，从可视角度来讲，不可避免地会发生两个元素试图同时出现于同一位置的情况。显示其中一个就会覆盖另外一个，但如何控制哪一个元素出现于“顶部”呢？

这里我们将引入 z-index。

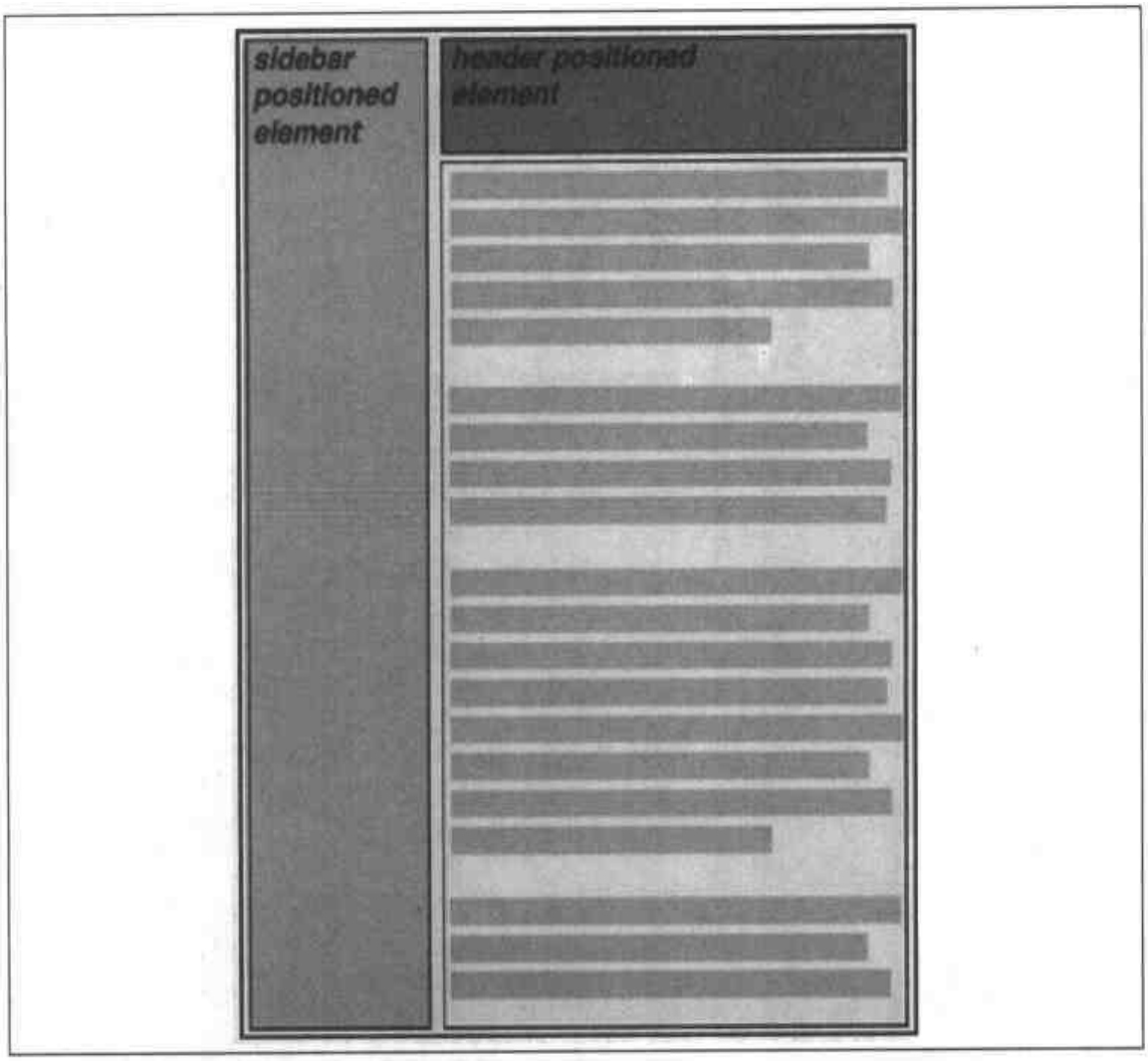


图 9-25 分隔“框架”

z-index

允许值	integer | auto
初始值	auto
适用于	定位元素
可否继承	否

z-index允许网页制作者改变元素相互重叠的顺序。其名字由坐标系得来，在坐标系中，从一侧到另一侧叫做x轴，从上到下叫做y轴。在这种情况下，第三个轴 —— 从前到后，或从近到远 —— 叫做z轴。这样，沿着这条轴的元素使用z-index值来代表。图9-26描述了这个系统。

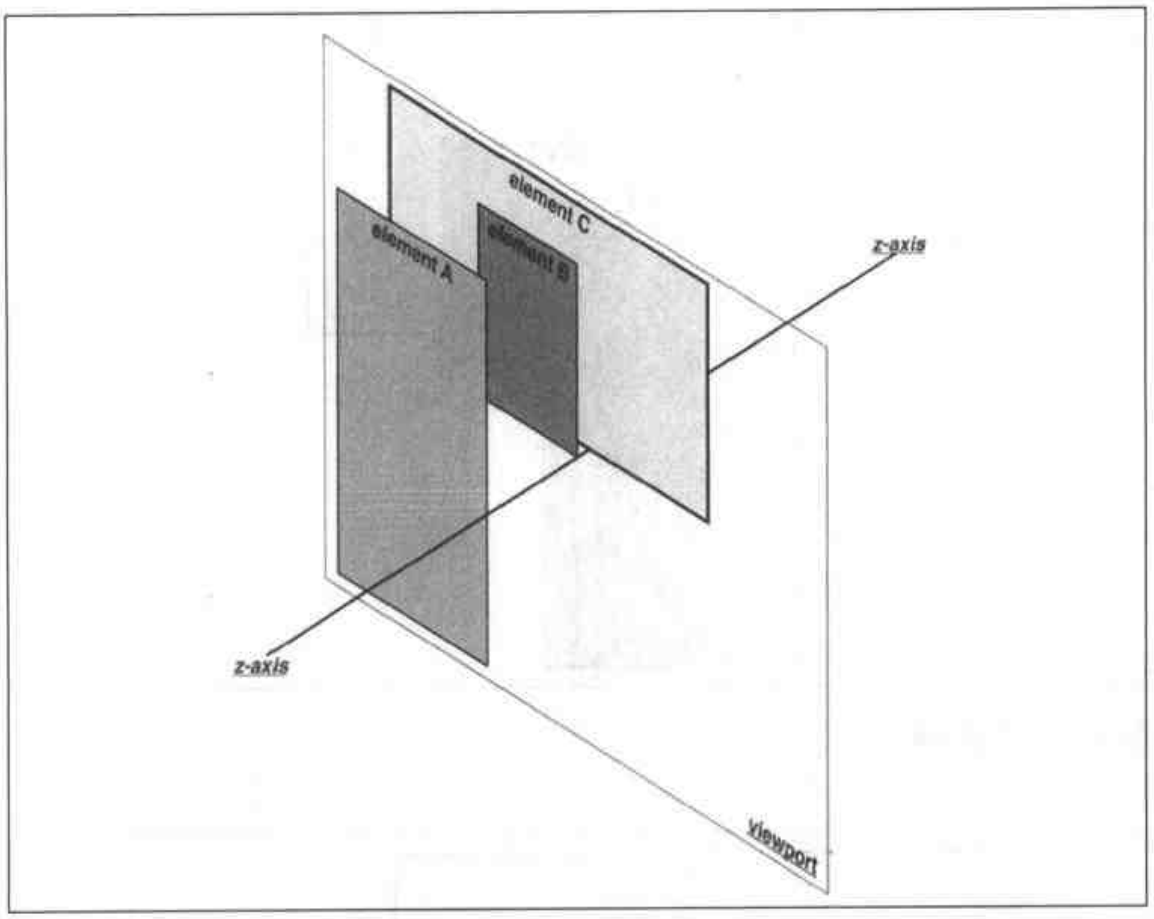


图 9-26 z-index层叠概念图

在这个坐标系中，z-index值高的元素比z-index值低的元素更靠近读者些。这会使值较高的元素覆盖值较低的元素，如图9-27所示。这就被称为层叠(*stacking*)。

任何整数均可用做z-index值，包括负数。给元素指定个一个负值z-index会将它移到离读者更远的地方，即，它被移到层叠堆栈中更低的位置。考虑如下样式，如图 9-28 所示：

```
P.first {position: absolute; top: 0; left: 0;
  width: 20%; height: 10em; z-index: 6;}
P.second {position: absolute; top: 0; left: 10%;
  width: 30%; height: 5em; z-index: 2;}
P.third {position: absolute; top: 15%; left: 5%;
  width: 15%; height: 10em; z-index: -5;}
P.fourth {position: absolute; top: 10%; left: 15%;
  width: 40%; height: 10em; z-index: 0;}
```

每个元素都根据其样式定位，但通常的层叠顺序由z-index值改变。假定段落以

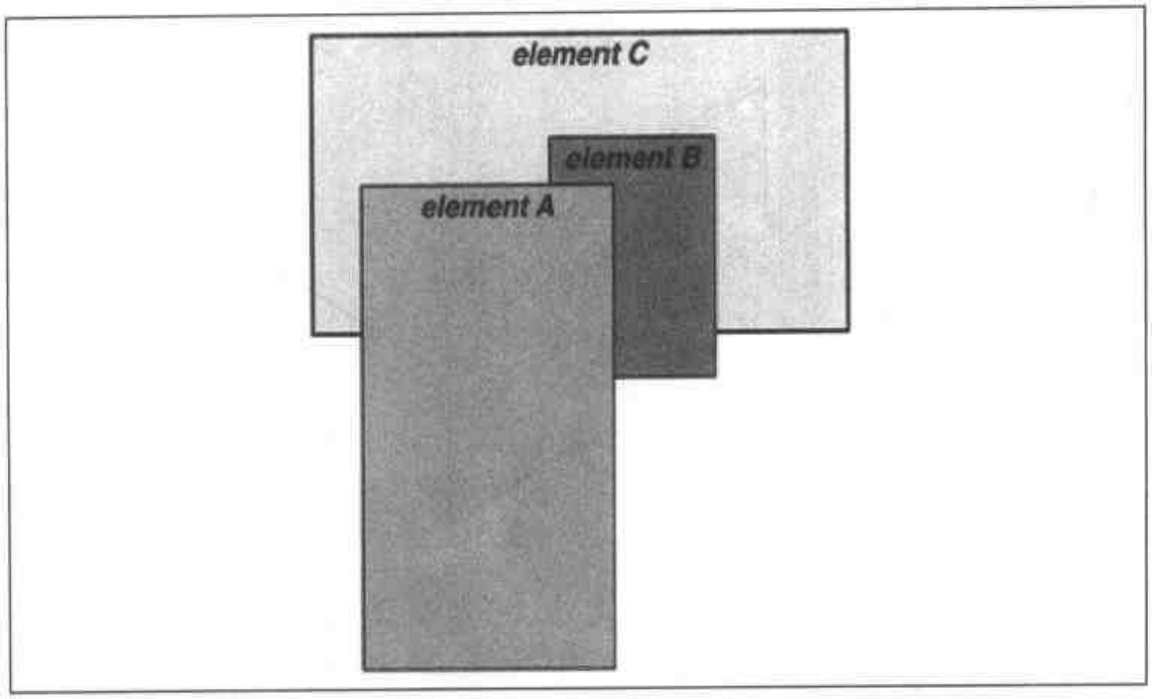


图 9-27 元素如何层叠

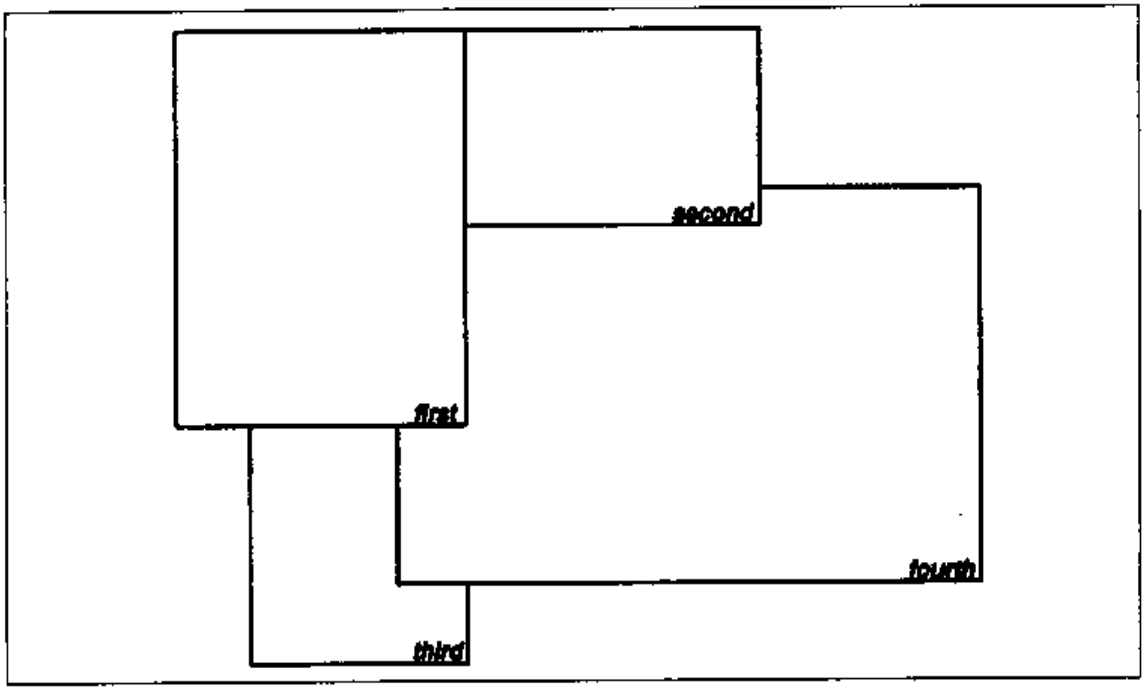


图 9-28 层叠元素可以相互覆盖

数字为序，则合理的层叠顺序会是从低到高，P.first，P.second，P.third，P.fourth。这会将P.first置于其他三个元素之后，P.fourth置于另外三个元素之前。现在，由于有了 z-index，可以控制层叠的顺序。

如同前几个例子所示，没有特别指定z-index值必须是连续的，z-index可以指定任意大小的整数。若想确使一个元素位于其他元素之前，可以使用一行z-indcx：100000的规则。这会在大多数情况下按期望的那样作用——当然，假如曾定义另一元素的z-index为100001（或更高），这个元素会在前面。

一旦为一个元素的z-index指定了值（不是auto），那这个元素就建立自身的局部层叠上下文(*stackig-context*)。这表示元素的所有后辈都会有自己的层叠顺序，与上级元素相关。这与元素建立新包含块的方式类似。给定如下样式，会生成如图9-29所示的效果：

```
P.one {position: absolute; top: 0; left: 0; width: 50%; height: 10em;
  z-index: 10;}
P.two {position: absolute; top: 30%; left: 25%; width: 50%; height: 10em;
  z-index: 7;}
P.three {position: absolute; top: 60%; left: 0; width: 50%; height: 10em;
  z-index: -1;}
P.one B {position: relative; left: 15em; top: 0; z-index: -404;}
P.two B {position: relative; left: 3em; top: -1em; z-index: 36;}
P.two EM {position: relative; top: 4em; left: 7em; z-index: -42;}
P.three B {position: relative; top: 0; left: 3em; z-index: 23;}
```

注意相对定位内联元素的层叠顺序位置。当然，每个这样的元素都会根据其上级元素正确定位。然而，仔细观察后辈P.two，尽管B元素在其上级元素之前，EM在它的后面，但它们两个都位于P.three之前。这是因为z-index值32和-42都是相对于P.two，而不是通常的相对于文档。从某种意义来说，P.two及它的所有下级元素都分享z-index值7，然后将它们自身的小型z-index置于P.two的上下文之中。

如果希望从另一角度来看这个问题，如果B元素的z-index值为7,36，而EM的值是7,-42。这些只是隐含的概念值，并不符合规范。然而，这个系统帮助我们描述了总体的层叠顺序是如何决定的。考虑：

```
P.one          10
P.one B        10,-404
P.two B        7,36
P.two          7
P.two EM       7,-42
P.three B      -1,23
P.three        -1
```

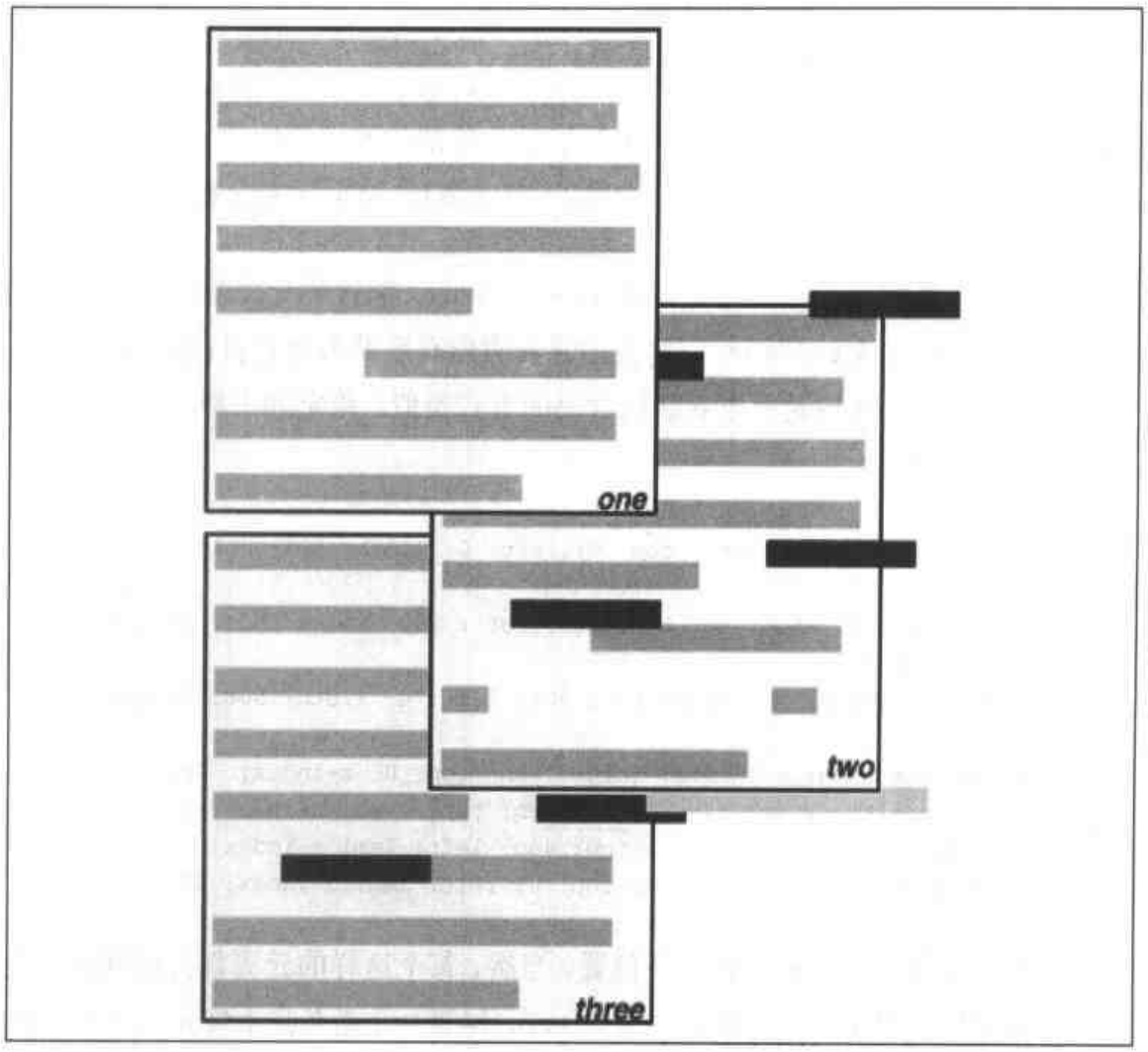


图 9-29 定位和 z-index 的例子

这个概念框架精确描述了元素层叠的顺序。尽管元素的后辈在层叠顺序中可以高于或低于该元素，它们跟这个元素分组在一起。

还有一个待检测的值。规范中对缺省值：auto 如下描述：

> 当前层叠上下文中生成框的层叠级别与其父元素框相同。框不会建立新的局部层叠上下文（CSS2：9.9.1）。

这看起来意味着用户代理可自由使用已用于文档布局的任何层叠算法，然而，这也可能意味着任何设置 z-index：auto 的元素都可能作为 z-index：0 来处理。不幸的是，CSS2规范在这一点上的规定不是很清楚，因此在不同的用户代理间可能存在不一致的问题。

小结

深入研究会发现，定位是强制性很强的技术。如果希望在多浏览器环境中得到一致的行为，很可能会失败。问题的重点不是在某些浏览器中不能工作，而是在很多浏览器中只能部分工作，如 Navigator 4 和 Internet Explorer 4.5。应用定位很有乐趣，或许某一天，我们可以将它应用于表格及框架，从而极大改进可访问性与向后兼容性。在本书中，它仍是创建设计原型的方式，但用于公共网站会显得不合时宜。

事实上，这个观点可应用于 CSS2 的绝大部分内容上，下一章我们将给出一个概览。

第十章

CSS2 展望

在写本书的过程中，我在如何处理CSS2的问题上犹豫再三。它被W3C推荐使用，但它被正确实现的部分却很少，这使得讨论CSS2的细节像是在浪费时间 —— 我们大家的时间。毕竟，不仅笔者必须虚构出所有的构图（更不必说揣摩一些情况下的正确行为），而且读者也无法实践所讨论的内容，因为浏览器根本不接受这些CSS2规则。

但另一方面，CSS2又不能被忽略。因此我最后决定简要而抽象地谈谈CSS2，即本章。本书的下一版定会为需求所驱动而增加关于CSS2的细节信息，只待时机成熟且浏览器开始正确实现CSS2的主要部分时，我们就会开始着手做这件事了。

本章所示的图中，大部分都是虚构的，因为没有其他生成这些例子的方法。告诉读者这些，其目的是减少读者试图解决这些图如何生成时的困扰。以下是CSS2的概述。

与CSS1的区别

只有一小部分CSS1属性添加了新的取值。这些多与寻址问题有关，且这些问题在CSS1写成时不存在或未被考虑。一个有代表性的新值是inherit，它使得所有事物都发生了较大的变化。

显示属性的增加与改变

display属性在CSS2中添加了不少新值。现在，在block、inline、line-item和none之外，还增加了run-in、compact及marker（稍后有讨论），以及一些用于表格的特殊值（稍后也将涉及）。

display的值compact与<DL compact>有类似的效果（假如浏览器支持HTML中的这一点）。基本上，如果一个元素设置为display: compact，如果有足够的空间，它就会显示下一个元素的边界。否则，两个元素都会被当作块级元素来处理。想像一个“压缩”元素为浮动元素，但只在有足够空间时才显示它，而无需改变接下来的元素的格式编排，如图10-1所示。

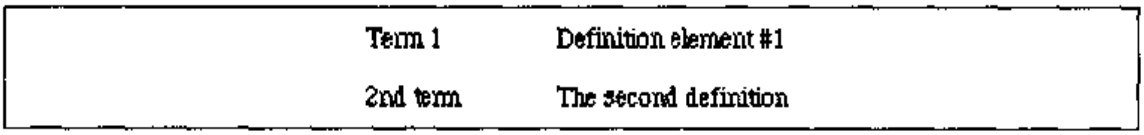


图10-1 定义列表的压缩显示

另一方面，run-in的效果是从紧接着的块级元素的开始位置将块级元素变为一个内联元素。另一方面是，一个块级元素设为run-in将会与下一个块级元素联合在一起，从而两个一起形成一个块级元素。

如下代码：

```
<H3 STYLE = "display:run-in;">A Heading.</H3>
<P>This is a paragraph of text....</P>
```

结果如图10-2所示。

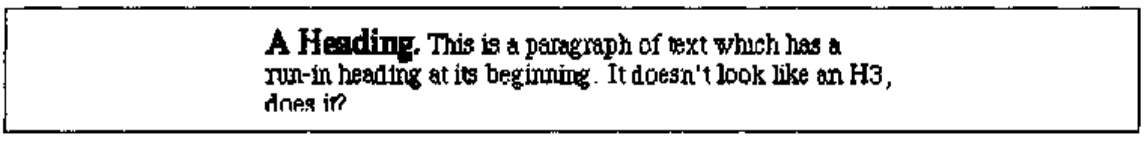


图10-2 一个run-in标题

display类型的run-in可应用于任何块级元素，不仅是标题。然而，这个规则只在当下一个元素是块级元素而不是浮动元素或绝对定位元素时才有效。例如，若将内联定位锚设为run-in，那就不会有任何效果。

另一个关于display的变化是其缺省值是inline，而不是block，CSS1中定义的是block。网页制作者已经把初始缺省值称为错误，因此如果未声明一个值给display，它就被假定为inline。当然，浏览器应有它自己的内置HTML样式，因此不必担心段落会突然合到一起。

更多的继承

最后，本节将讨论CSS2中非常重要的新特性：值inherit。如果问："inherit应用于哪个属性？"答案是："它们中的每一个。"CSS中没有任何属性不接受inherit值。

inherit用于显式声明一特定计算值应从其父元素继承。换句话来说，如果BODY的font-size经计算为14磅，则声明P {font-size: inherit;}将设置段落文本的大小为14磅，只要段落是BODY元素的下级元素。类似地，可以确使超链接总是与包围它们的文本有相同的颜色，只需使用以下的简单声明：

```
A:link, A:visited {color: inherit;}
```

这个改变的能力不能低估。在效果上，它可以替换通常产生效果的专用机制。通常，超链接是蓝色的，除非显式声明它为其他颜色，如果希望同一页面不同区域的链接颜色不同，必须为每种颜色构造不同的规则。

现在，由于有了inherit，如果想使超链接与它们周围文本的颜色一致，只需一条规则便可覆盖全部情况。注意这并不一定是个好的主意，或是inherit可被使用的唯一情况。它只是最经常的可能而已。

CSS2选择符

接下来要讨论CSS2选择符的细节，因为它们可能是规范中最快实现的一部分。因此，尽管在阅读这部分内容时，无法实践这时描述的任何事情，希望大部分可以被以后发行的浏览器包含进去。

基本选择符

首先，在现存的选择符机制，如上下文选择符之外，还有一些新的选择符符号，它们使得构造专用、复杂的选择更加容易，无需分类全部文档中分散的类或ID。

通用选择符

新的选择符中作用最大的是通用选择符。它使用一个星号（*）指定，且匹配文档中的任何元素。这样，使用这个声明以确保所有元素颜色为黑色：

```
* {color: black;}
```

被当作上下文选择符的一部分来使用时，通用选择符可以生成一些有趣的效果。例如，假定希望至少是BODY元素的两级以下的UL元素颜色为灰色。换句话说，BODY元素的任何下级UL元素不为灰色，但其他UL，不论是DIV的下级元素，还是列表项，或是表格，都会成为灰色。这由以下规则完成：

```
BODY * UL {color: gray;}
```

图10-3显示了此声明的效果。

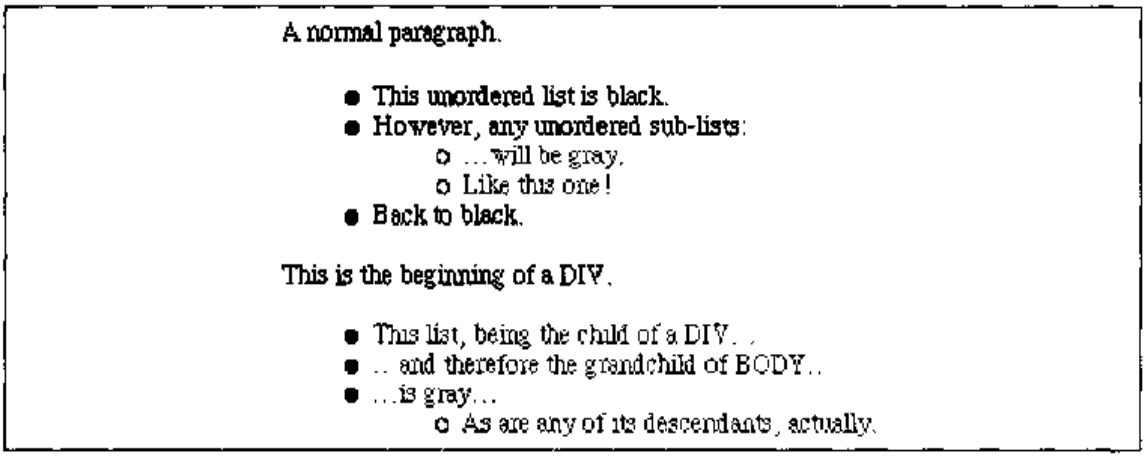


图10-3 使BODY的“孙子”（及它们的后辈）为灰色

另一方面，也许还希望DIV的后辈元素为紫色。可以写成：

```
DIV * {color: purple;}
```

初看上去，这与省略掉*没有区别，省略掉*则依靠继承将颜色带给DIV的所有后辈。然而，差别是真实存在的：显示的规则将匹配每个DIV后辈，且因而替换

了继承机制。这样，甚至定位锚（也是DIV的后辈）也因这条给定的规则而变成紫色，然而简单继承不足以使它们成为紫色。

尽管可以联合使用通用选择符与类及ID选择符，但没有理由这样做。以下两条规则的作用完全相同：

```
*.apple {color: red;}
.apple {color: red;}
```

然而，应该考虑：如果涉及较老的不知道CSS2的用户代理，则*.class(或*#id)是较容易蒙蔽它们的方法。因为两者在CSS1中都是无效选择符的例子。它们应当被CSS1的语法分析程序忽略。如果它们没有被忽略，则可能导致奇怪的结果。因此，在连接类及ID选择符时省略通用选择符也许是一个好主意。

子选择符

另外一个有趣的选择符是子选择符，即大于符号（>），用于匹配那些其他元素的直接后辈：

```
BODY > P {color: green;}

<BODY>
<P>This paragraph is green.</P>
<DIV>
<P>This paragraph is not green.</P>
</DIV>
<P>This paragraph is green.</P>
</BODY>
```

只有第一个和第三个段落与规则匹配，因为它们是BODY的下级。第二个段落是DIV的下级，因此是BODY的孙级，也就与规则不匹配。

子选择符必须有用“>”符号分开的两个或更多个选择符。也有可能使子选择符成为上下文选择符的一部分：

```
DIV OL>LI EM {color: purple;}
```

这条规则匹配列表项元素的后辈EM文本，只要该列表项是OL元素的下级，它是DIV元素的后辈（注意这次“>”符号周围没有空白，这是合法的。这个符号周围的空白是可选的。）这样：

```
<BODY>
<OL>
<LI>The EM text here is <EM>not</EM> purple.</LI>
</OL>
<DIV>
<OL>
<LI>Look, a list:
<UL>
<LI>The emphasized text here <EM>is</EM> purple.</LI>
</UL>
</LI>
</OL>
</DIV>
</BODY>
```

紫色EM文本之所以为紫色，是因为它是LI的曾孙，LI是OL的下级，而OL是BODY元素的孙级元素。第一个EM不匹配是因为它的祖父OL不是DIV的直接下级。

更好的办法是，可以把多种选择符捆绑在一起以精确定位某一指定类型的元素。例如：

```
BODY > OL > LI {color: silver;}

<BODY>
<OL>
<LI>The text here is silver.</LI>
</OL>
<DIV>
<UL>
<LI>Look, a list (and this text is not silver, by the way):
<OL>
<LI>The text here is <EM>not</EM> silver.</LI>
</OL>
</LI>
</UL>
</DIV>
</BODY>
```

在这个规则下，得到类似图10-4的结果。

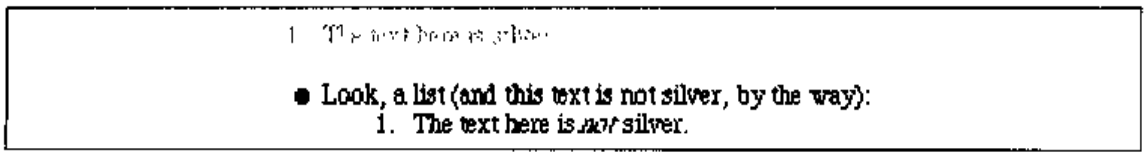


图10-4 只选择孙级元素

资源中的第一项是银色的，因为它是有序列表的下级，而有序列表是BODY的下级。资源中的第二项是一个无序列表的下级，因此它不匹配于此规则。最后，资源中的第三项是一个有序列表的下级，但OL元素是LI元素的下级，因此它也不匹配于此规则。

相邻兄弟选择符

如果觉得有趣，那我们就来继续下一个话题：相邻兄弟（adjacent-sibling）选择符。同子选择符有些类似，但这时，样式应用于在文档树中有共同父元素且相邻的元素。例如：

```
H2 + P {color: silver;}

<H2>Coloring Text</H2>
<P>This paragraph is silver.</P>
<P>This paragraph is not.</P>
<H2>More Coloring Text</H2>
<UL><LI>This is not silver</LI></UL>
<P>Neither is this.</P>

<H2>More Coloring Text</H2>
This text is not silver.
<P>This paragraph is silver.</P>
<P>This paragraph is not.</P>
```

在第一段标记中，段落紧跟在H2标题之后，因此它是银色的。在第二段标记中，与H2邻接的元素是UL，不与规则相匹配，UL之后的段落也不匹配。在最后一段标记中，尽管H2标题之后有文本，但那些文本不属于任何元素，因此紧随文本之后的段落与规则匹配，且颜色为银色。所有这些如图10-5所示。

若希望使任何紧随H2的元素为银色，可以使用通用选择符：

```
H2 + * {color: silver;}
```

在很多情况下可以利用用户代理忽略元素间文本的特点。例如，希望一个文档中跟在EM文本之后的STRONG文本为银色，其余的加粗文本为灰色：

```
STRONG {color: gray;}
EM + STRONG {color: silver;}
<P>While the first strong element is <STRONG>gray</STRONG>, the
```

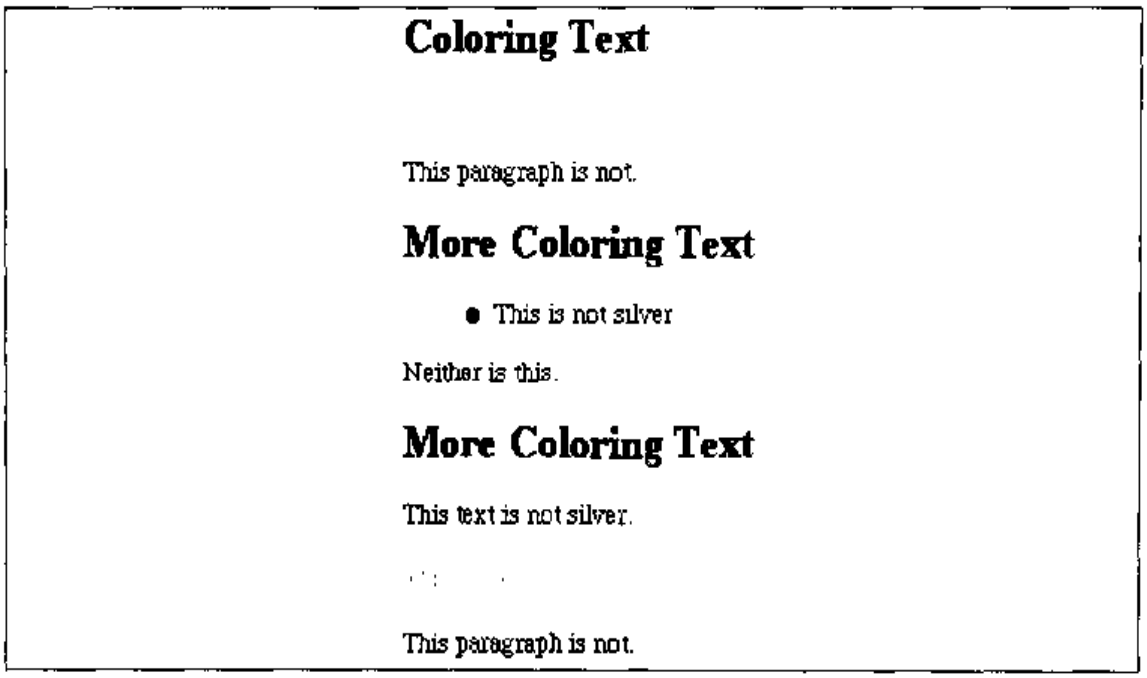


图 10-5　选择相邻元素

```
<EM>second</EM> strong element is <STRONG>silver</STRONG>, because it
follows an "EM" element.
```

结果见图 10-6 所示。

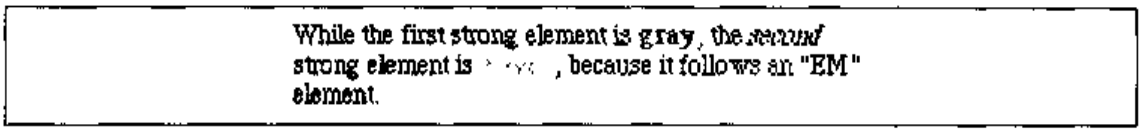


图 10-6　进一步的相邻选择

属性选择符

引入属性选择符后，CSS 变得更加复杂、准确、功能强大。属性选择符有四种匹配方式，每一种都有自身的长处与优点。

属性匹配

首先是生成一个匹配于任何有指定属性的元素的选择符的能力。例如，可以使用 NAME 属性来匹配所有的定位锚，或匹配所有带 BORDER 属性的 IMG 元素，或某个类的所有元素：

```
A[name] {color: purple;}       /* colors any NAME anchor purple */
IMG[border] {border-color: blue;}  /* sets blue border for any bordered IMG */
[class] {color: red;}          /* sets any classed elements red */
```

这些情况中，每个元素的具体属性值并不重要。只要给定属性在元素中出现，元素便匹配该选择符。这样，在下例中，前两个IMG元素匹配先前的规则，而第三个却不匹配，如图10-7所示：

```
IMG[border] {border-color: blue;}

<IMG SRC="one.gif" BORDER="1" ALT="image one (match)">
<IMG SRC="two.gif" BORDER="23" ALT="image two (match)">
<IMG SRC="three.gif" ALT="image three (no match)">
```

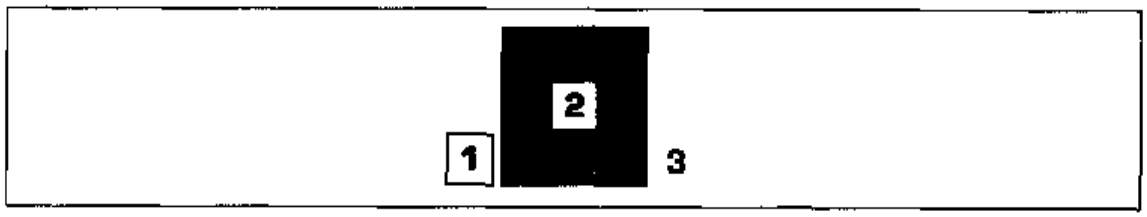


图10-7 属性匹配

值匹配

也可以根据值来匹配属性。假定希望所在超链接样式设为灰色，除了那些指向W3C（http://www.w3.org/）站点首页的链接外。那些链接设为银色。如图10-8所示：

```
A[href] {color: gray;}
A[href="http://www.w3.org1"] {color: silver;}
```

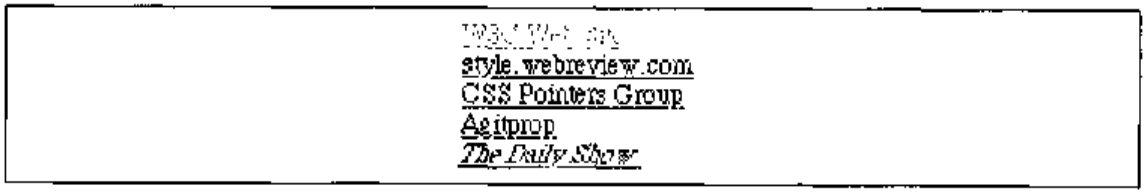


图10-8 属性值匹配

由于有了层叠，当定位锚指向W3C站点时，会应用第二条规则代替第一条规则。

匹配单个属性值

由于HTML 3.2的出现，可以对一指定元素设置多个类名，如:

```
<P CLASS="footnote example reference">
```

在CSS1中，只能涉及一个值，使用如P.example这样的选择符。在CSS2中，可以生成一个选择符，要求类名必须精确匹配；或可以生成一个选择符，只需一个值匹配。P[class="example"]不匹配于前面的三类名段落，因为值不相同(example与foonnote example reference不一样)。

另一方面，P[class~="example"]将匹配footnote example reference，因为这种选择符只要求匹配属性中的一个值。唯一的区别在于波浪符(~)，但区别却很大。

作为一个例子，假定一个文档中的元素可以有一个或多个类值，它们是driving、flying、nautical、directions和title，这样，一个元素可以有driving directions类，或者也许是flying directions title。进一步假定所有的标题都是红色，任何与flying有关的(不包括标题)应该是绿色，任何与driving有关的为紫色。声明如下:

```
*[class~="flying"] {color: green;}
*[class~="driving"] {color: purple;}
*[class~="title"] {color: red;}
```

等号之前的波浪符号(~)使这些选择符匹配class属性中的任何一个值。这样，下面的规则将匹配IMG元素，元素有类figure及包含字Figure的alt属性——如Figure1，Figure2等:

```
IMG[class="figure"][alt~="Figure"] {margin: 5px;}

<IMG SRC="picture13.jpg" CLASS="figure" ALT="Figure 13">
```

模拟类与ID

使用属性选择符，也可以模拟类与ID选择符。下面的成对规则大致相同:

```
P[class="directions"] {color: red;}
P.directions {color: red;}
```

```
DIV[ID="abc123"] {color: blue;}
DIV#abc123 {color: blue;}
```

显然，每对规则中的后者更易于输入和编辑，在大多数情况下我们更乐于使用这种规则。

如果希望精确匹配，可以使用普通的属性选择符。这样，下面的规则：

```
P[class="driving directions"] {color: green;}
```

将会匹配这个标记：

```
<P CLASS="driving directions">This is a side note (and it's green)(/P)
```

如果不那么关心精确匹配，可以将类选择符捆绑在一起，这是CSS2的新特性，使用这个方案，可以用如下的方式用值 driving directions 匹配 class 属性：

```
P.driving.directions {color: blue;}

<P CLASS="driving directions">This is a side note (and it's blue this time).
</P>
```

可以使用相同的选择符匹配值driving directions，因为它既包含driving，又包含 directions，只是顺序不同。

这看起来更容易输入。那为什么还使用较长的属性选择符呢？使用属性选择符是因为 .class和#ID选择符只应用于HTML文档，或其他使用包含类及ID语言的文档。其他语言，如基于XML上的，也许不能得到这些便利，这种情况需要使用属性选择符。

匹配用连字号连接的值

最后一种属性匹配程序通常用于语言匹配，但它的确还有其他用途。任何使用符号|=的属性选择符将会匹配一个值，该值以一特定值开头，并假定该值在连字号分隔值的开头。例如：

```
P[lang|="en"]
```

这个选择符会匹配段落，只要段落的 lang 属性的值为 en，en-us，en-UK，en-Cockney等。

实际上，它可以用于匹配任何格式类似的值。例如，若有ALT文本为fig-1，fig-2，fig-3等等的图像，且希望匹配它们，可使用如下选择符：

```
IMG[alt1="fig"]
```

这是较少被使用的用途，但经常十分有效。任意前面的规则不会匹配值figure或config，因为两者均不是以fig-开头或简单为fig。然而，规则将匹配fig-tree。

更多的伪类与伪元素

尽管可能看起来有些多余，解释的另一个方面是关于伪类与伪元素选择符。

:hover

从:hover开始。基本思路是:hover规则的样式在鼠标“逗留（hover）”于某元素时被应用。例如，当鼠标定位于一个链接上，只要击鼠标便使浏览器跟随该链接，称鼠标“逗留”于链接上。这在某些特征上与著名的JavaScript“rollover”类似，即当鼠标逗留于图像上时，图像发生改变。由于有:hover，可以很容易地指定一种逗留样式（hover style）：

```
A:link {background: white; color: blue;}
A:hover {background: blue; color: white;}
```

这些样式会使定位锚在鼠标逗留在它上面时颜色反转，如图10-9所示。

图10-9 hover样式

事实上，规则A:hover会在鼠标逗留于任何定位锚上时被使用，而不只是用于某个超链接。而其他一些伪类，如:link及:visited，在HTML中限制于A元素，而:hover不是这样。用户代理在理论上允许指定逗留样式于任何元素上，如下所示：

```
P:hover {font-weight: bold;}
```

因此，如果希望确保逗留样式只应用于超链接，应使用如下规则：

```
A:link:hover {background: blue; color: white;}
```

联合伪类的能力是CSS2的新特性。

警告：Internet Explorer 4.x及5.x均只在定位锚上可识别:hover。本书完成时，还没有其他浏览器可识别:hover。

:focus

与:hover类似的是:focus，它用于定义元素样式为“焦点”。例如，一个表格元素，在它准备好接收输入时占有“焦点”。因此，INPUT元素背景可以设为黄色，以突出当前的活动输入：

```
INPUT:focus {backround: yellow;}
```

这个样式只在元素占有焦点时应用，当用户从一个输入移到另一个时，样式从前者移开并应用于后者。这是一个受欢迎的功能，因为它减少了生成这种效果对JavaScript的需要。

注意：使用:hover与:focus的文档回流导致了一系列问题。例如：

```
A:hover {font-size: 200%;}
```

从理论上来说，用户代理应该当鼠标逗留在定位锚上时加倍定位锚中文本的大小，这会导致重显示问题。网页制作者在声明TEXTAREA元素应在占有焦点时改变大小也可能造成类似的问题。未要求用户代理基于指定给伪元素的样式回流文档，只保持它为可见。当然，也有一些可能会回流文档。

:lang

伪类:lang则有着完全不同的特性，它用于对匹配语言的元素应用样式。例如希望所有英文段落为白底黑字，所有法文段落为黑底白字：

```
P:lang(en) {color: black; background: white;}
P:lang(fr) {color: white; background: black;}
```

当然，用户代理是不大可能自己计算出元素语言的。它们必须依赖于文档标记，比如HTML中的lang属性：

```
<P lang="en">This paragraph is in English.</P>
<P lang="fr">Ce paragraphe est en fran&ccedil;ais.</P>
```

结果如图10-10所示。

图10-10 根据语言改变样式

即使这不是很常用，但它仍十分方便。例如，可以定义样式应用于整个文档：

```
HTML:lang(de) {color: black; background: yellow;}
```

这样所有的HTML文档标记为德文，并用黄色背景黑色文本显示。这个标记可使用lang属性在文档开头的META标签中制作出，甚至还可以使用文档的HTTP标题中的值。这与上一节讨论的 |= 属性选择符有些类似，但它更为通用。

:first-child

要介绍的最后一个新伪类选择符是:first-child选择符，用于匹配另一个元素的第一个下级元素。例如，也许希望只要第一个下级元素是段落，则使每个DIV的第一个下级元素用斜体替代普通文本（见图10-11）：

```
P {color: black;}
P:first-child {font-style: italic;}

<BODY>
<P>This paragraph should be italic.</P>
<P>This paragraph should be normal.</P>
<DIV STYLE="border: 1px dashed gray;">
This text should be normal.
<P>This paragraph should be italic.</P>
<P>This paragraph should be normal.</P>
```

```
</DIV>
<DIV STYLE="border: 1px dotted gray;">
<H2>This H2 should be normal.</H2>
<P>This paragraph should be normal.</P>
<P>This paragraph should be normal.</P>
</DIV>
</BODY>
```

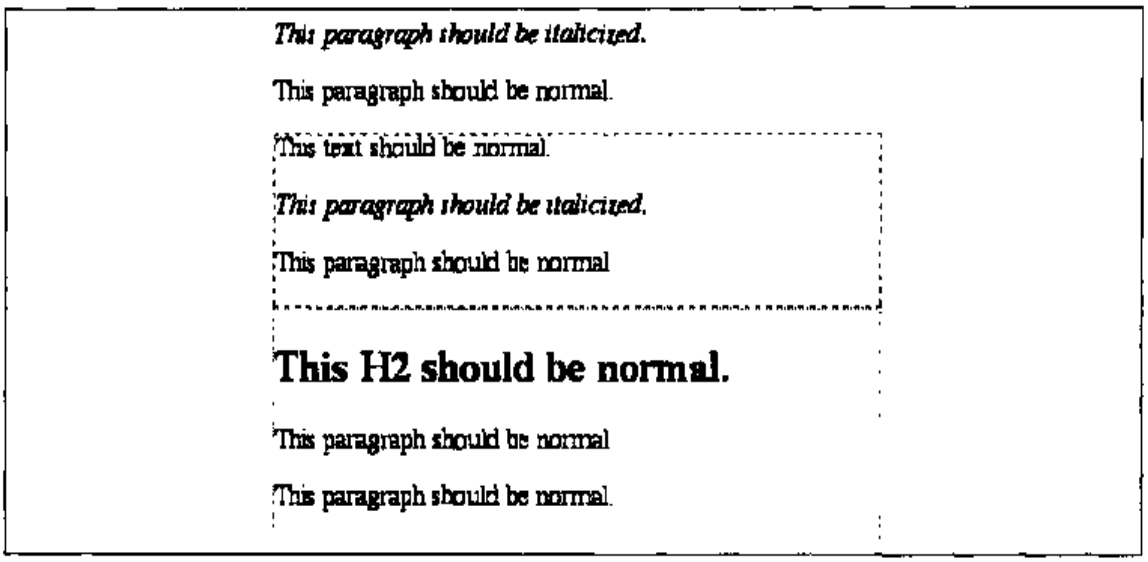


图 10-11 为某些第一个下级元素选择样式

第一个段落成为斜体，因为它是 BODY 元素的第一个下级元素。类似地，第一个 DIV 中的第一个段落成为斜体，因为它是 DIV 的第一个下级元素，尽管有文本先于它出现。只有结构化的元素与本伪类有关，因此，段落前的文本不作为第一个下级元素影响段落的状态。然而，在第二个 DIV 中，H2 是第一个下级元素，因此，不匹配规则 P:first-child。如果目的是使任何元素的第一个下级元素都成为斜体，不论它是什么元素，则只需省略选择符的元素部分，或同通用选择符结合起来使用，生成如图 10-12 所示的结果：

```
*:first-child {font-style: italic;}
```

假如希望应用样式于第一个下级元素的一部分的某些元素；例如，第一个下级段落元素中的强调文本为斜体：

```
P:first-child EM {font-style: italic;}
```

当然，这会匹配所有的第一个下级段落，而不管其父元素如何。假定希望一个规则只应用于 DIV 元素的第一个下级段落。这种情况下，需要使用子选择符：

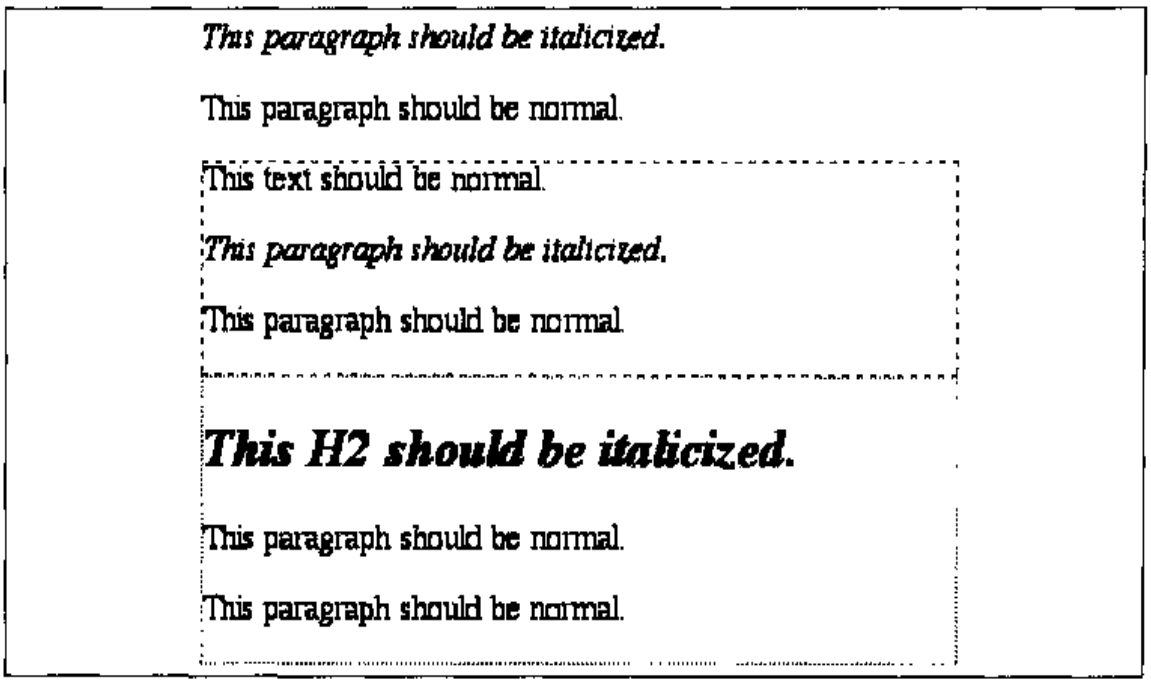


图 10-12 选择任何第一个下级元素

```
DIV > P:first-child {font-style: italic;}
```

这可以解释为:“任何第一个下级段落，且是DIV的下级，应该为斜体。”如果子选择符空缺，如下所示:

```
DIV P:first-child {font-style: italic;}
```

则规则被视为:“任何元素的第一个下级段落，且也是DIV的后辈，应该为斜体。”差别很细微，但的确存在。

其他的伪元素与伪类

还有两个新的伪元素:

```
:before
:after
```

及三个新的伪类:

```
:left
:right
:first
```

这些伪类与伪元素的细节将在本章稍后讨论。

字体与文本

font 属性在 CSS2 中也得到了一些新值:

```
caption
icon
menu
message-box
small-caption
status-bar
```

这些值赋予 font 属性匹配字体系列、大小、粗细及其他的能力，根据用户在其计算机上的特定设置来匹配。例如，Macintosh 按钮的标号典型为使用 9 磅 Geneva，假定用户未做改变，则任何使用值 icon 声明的 font 都会生成 9 磅 Geneva 文本 —— 只要页面是使用 Macintosh 浏览的:

```
SPAN.OScap {font: icon;} /* will look icon labels in OS */
```

当然，在一个 Windows 系统中，字体也会发生变化，且在其他 Windows 管理器（如 X）中，看起来也会有区别。灵活当然可以吸引人，更令人兴奋的是，它还允许网页制作者生成外观对用户来说很熟悉的页面。

新字体属性

CSS2 中的字体得到了两个新属性。font-size-adjust 试图帮助浏览器确保文本有预期的大小，无论浏览器是否可以使用样式表单中指定的字体。经常会出现网页制作者使用用户不可用字体的问题，且当用另一种字体替换时，看上去又过大或过小。这个新属性正是为了解决这个问题而提出的，而且会对那些希望确保文档被任何字体替换时均可读的网页制作者大有帮助。

另一个新字体属性是 font-stretch，允许用户为使用的字体定义可变宽度。这与在桌面印刷系统中设置字符宽度类似。属性使用关键字如 ultra-condensed，wider 及 expanded。变化的处理风格与字体粗细类似，可构造一个压缩与扩展字

体样式表，且关键字分派给不同的字体样式。如果没有样式存在，用户代理可以试着缩放自己的字体，或简单忽略font-streatch。图10-13显示了font-stretch的各种可能值的字体样式。

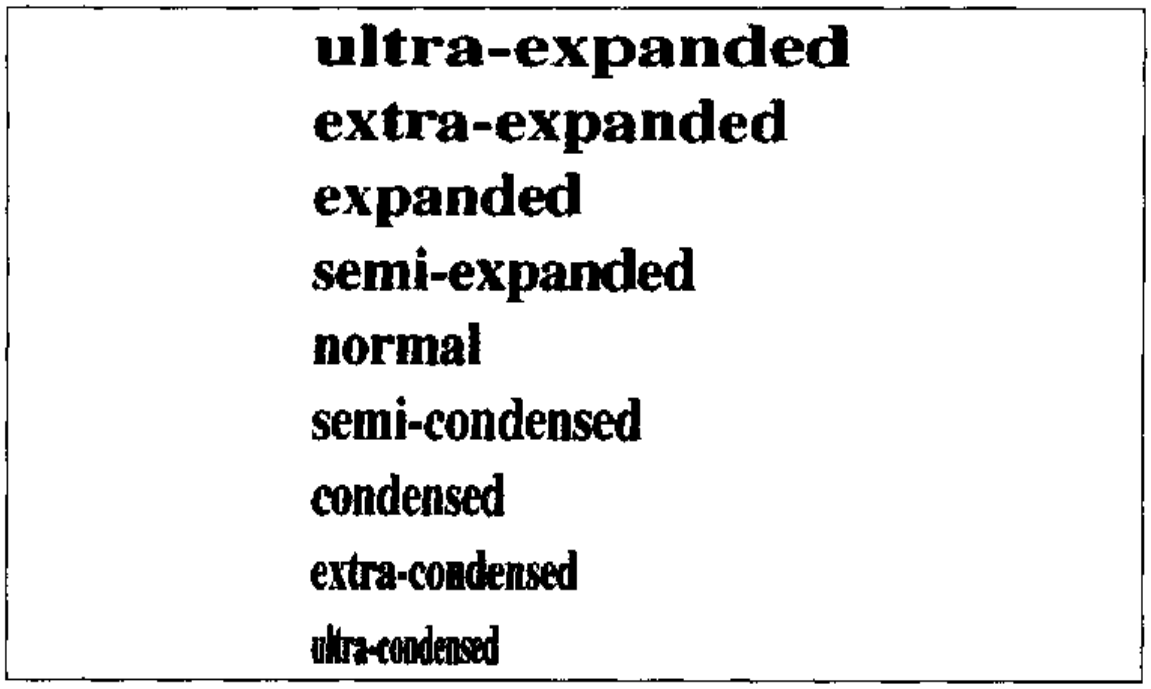


图10-13 扩展的字体

文本阴影

文本项有一个新属性，text-shadow，它的效果可顾名思义：即可以为文本定义指定颜色的投射阴影。甚至可以设置偏移及羽化半径，这样可得到很酷的模糊阴影和光照效果。可以预见，一旦浏览器支持该属性，它就会被广泛使用，图10-14模拟了一些例子。

图10-14 使用文字阴影属性的不同效果

产生内容

产生内容是在不改变内容的前提下，为已存在的内容增加一些其他内容的新方法。使用伪类元素:before和:after以及content属性来实现。下面是一个最基本的例子：

```
P:before, P:after {content: "\""; color: gray;}

<P>This is a quote.</P>
```

浏览器显示如图10-15所示的结果。

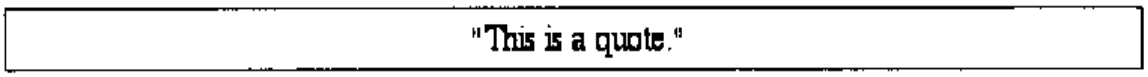


图10-15 增加产生内容

注意双引号标记被转义——前面增加了一个反向下划线。这是必须的，因为内容的文本值必须用双引号引起来。也可以在内容之前（之后）增加图像，使用P:before {content: url(para.gif);}，可以将标记放于每个段落的开始位置，甚至可以将多个值捆在一起，如图10-16所示：

```
P:before {content: url(para.gif) "--";}
```

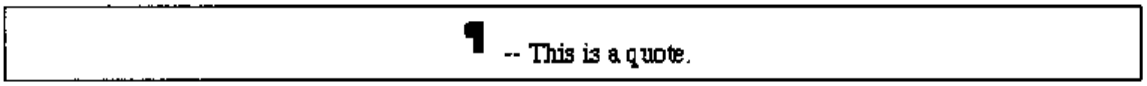


图10-16 在段落前增加一个图像及文本

这会使每个段落由一个段落标记、一个空格、两个划线，然后以又一个空格开头。注意，这些均被视为段落的一部分且在段落的行内。空格出现于双划线之前和之后是因为它们包含于字符串中。如果没有空格，那么在双划线的两侧也就不会有空格出现。

如果希望做真正的引用，使用真正的引号——但你知道，在HTML中指定弯曲的双引号很困难，有时即使试着指定了，它们也不会显示。在CSS2中有处理办法。

可以使用content的其他值：

- open-quote，插入开首引号
- close-quote，插入结尾引号
- no-open-quote，防止插入开首引号
- no-close-quote，防止插入结尾引号

这样，如果希望引用的话前后都加上引号，而不必逐个键入引号，可以让浏览器自动插入它们。

```
BLOCKQUOTE:before {content: open-quote;}
BLOCKQUOTE:after {content: close-quote;}
```

自动编号

CSS2 中包括用于自动编号的属性。首先，为 content 指定一个 counter 值。这有些复杂，且不大可能写出所有的可能性，下面是一个例子。假定希望文档的章和节编号为 1，1.1，1.2，1.3 等，而且，章使用 H1 标题，节使用 H2 标题。可使用如下声明：

```
H1:before {
   content: "Chapter " counter(chapter) ". ";
   counter-increment: chapter;    /* Add 1 to chapter*/
   counter-reset: section;        /* set section to 0 */
 }
 H2:before {
   content: counter(chapter) "." counter(section) " ";
   counter-increment: section;
 }
```

如图 10-17 所示，用户会在 H1 文本前增加“Chapter”和一个数字。这个数字会随 H1 的出现而自动而增长，由于声明了 content-increment: chapter;还将节计数器的值通过 content-reset: section;设为 0，这样，每个节标题（H2），浏览器使用章数，紧跟一个句号（.），然后是自动增长的节数。

不是每次必须增长 1，可以定义任何整数为增长值，包括 0 与负数。如果希望每节都为偶数，如图 1-18 所示，可声明：

```
H2:before {
```

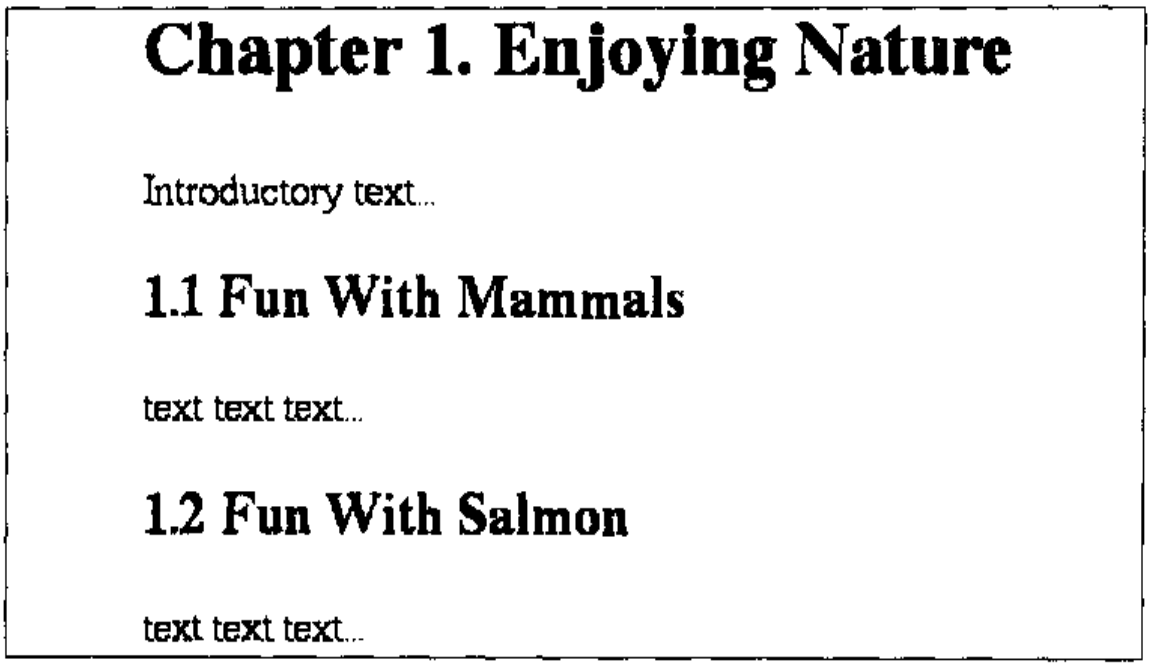


图 10-17 给元素增加计数

```
    content: "Section " counter(section) ". ";
    counter-increment: section 2; /* Add 2 to chapter */
}
```

也可以通过设置display为none来防止元素的计数增长。当然，这样会使整个元素都消失。

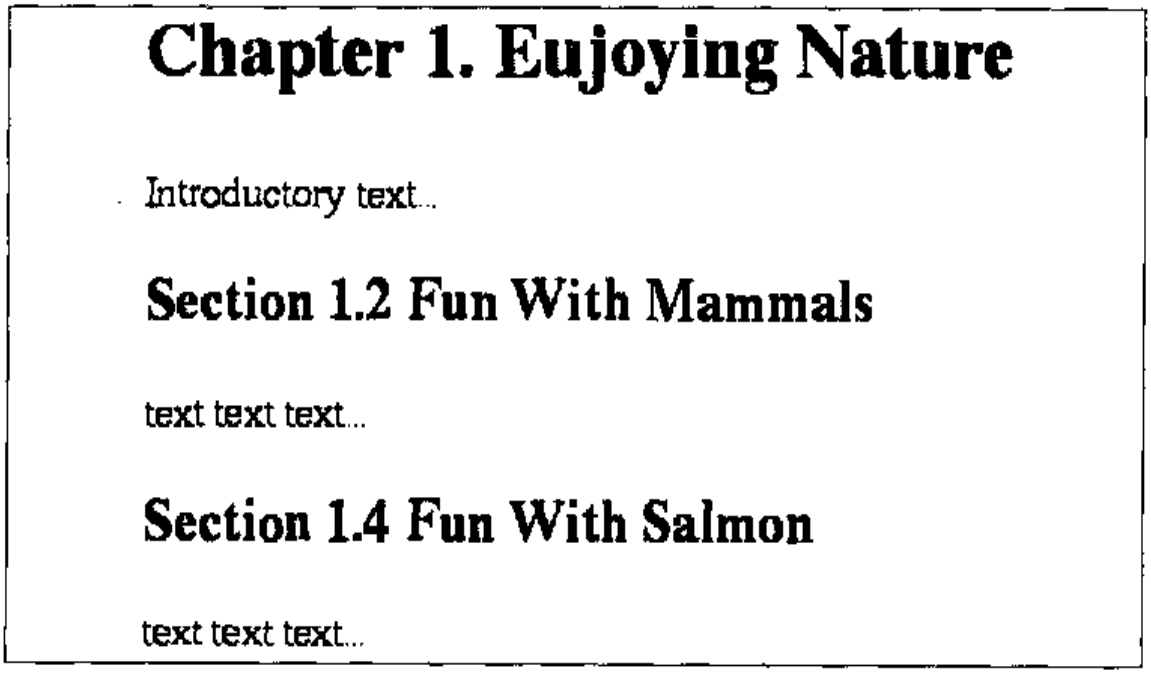


图 10-18 改变计数增长值

标识

通过使用display属性的marker值，可以为所有元素定义自己的标识（marker）样式。位于列表项之前的项目符号与数字都是标识。

假定希望重新生成无序列表的行为方式。为此，将使用图像 *disc.gif* 取代普通标记。若使用标识属性，声明为：

```
LI:before {display: marker;
   content: url(disc.gif);
   marker-offset: 1em;
}
```

这样，在每个列表项之前都插入一个圆盘图像，且距LI内容的左边沿偏移1em，如图 10-19 所示。

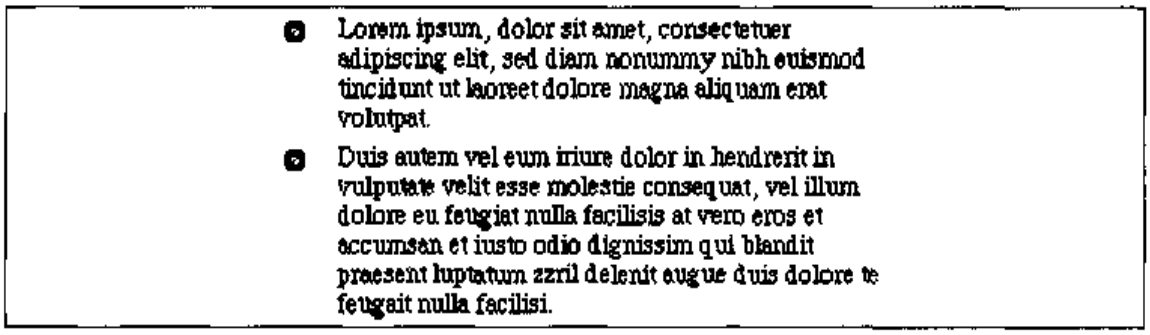


图 10-19 风格化列表标识

标识属性的应用范围不只限于列表项。假定在文档中，有几个段落具有类aside，希望在每个类旁边插入小的提示以使读者注意这些段落，可以用如下的方式：

```
BODY { counter-reset: aside-ctr;}
P {margin-left: 10em;}
P.aside:before {display: marker;
   content: "Aside " counter(aside-ctr) " --";
   counter-increment: aside-ctr;
   text-align: right;
   marker-offset: 1em;
   width: 9.5em;
}
```

效果如图 10-20 所示。

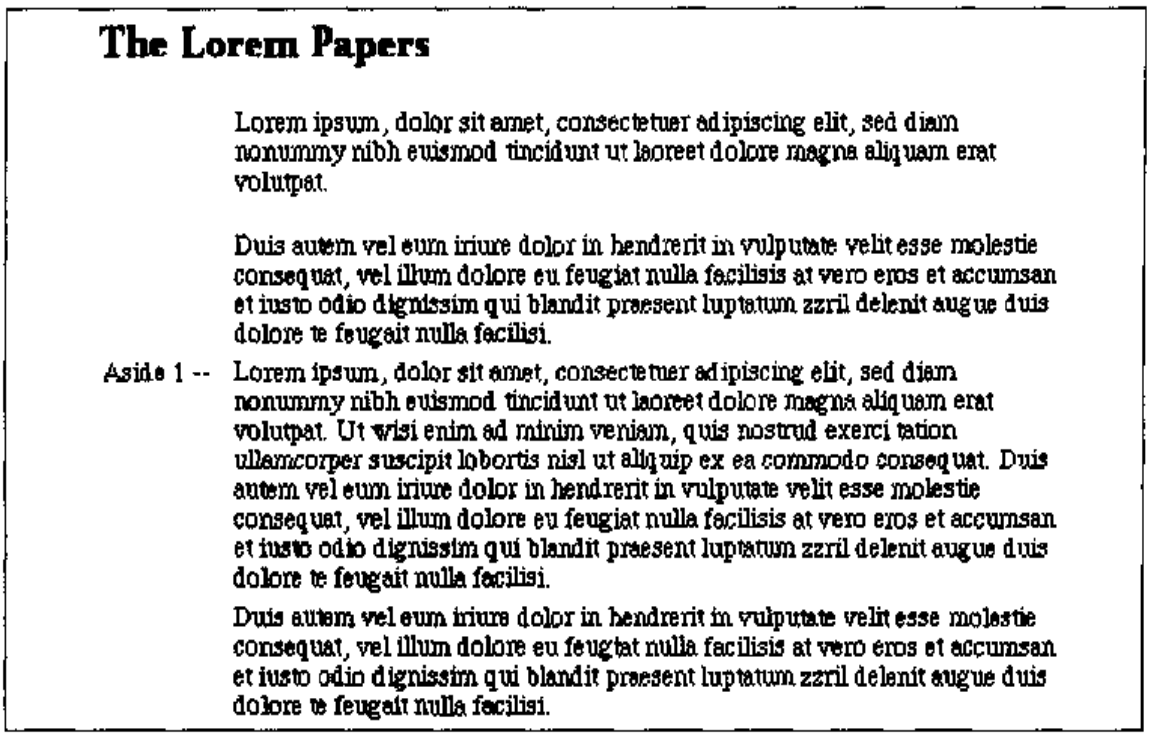


图 10-20 自动编号的 aside

这就是CSS2的另一特征，只要应用适当，允许网页制作者对文档做很多事情。

适应环境

CSS2提供了以下两个能力：改变浏览器的环境，使外表与用户的操作系统一体化。

光标

为实现第一个能力，采用了cursor属性，允许声明浏览器光标在经过一指定元素时的外观。想做一个有趣的关于下载时间的指针吗？将光标改变为等待光标（沙漏或表盘），在它经过超链接时，甚至可以将该属性同“光标文件”（规范未定义）挂钩，这样理论上就可以根据定位锚链接的地址对定位锚分类，并为每种链接载入不同的图标。例如，off-site链接使光标变为球形，而提供帮助的链接触发一个问号光标。

颜色

为使Web页面与用户的桌面环境更相配，CSS2定义了很多新的颜色关键字，如button-highlight，three-d-shadow和gray-text。这些都是为了使用用户的操作系统颜色而设的。总共有27个新颜色关键字，这里不一一列举，可在本章结尾处的表10-1中找到它们。

轮廓

四处移动光标时，也许会希望显示出焦点设置在哪里。例如，如果定义一个按钮，当光标经过它时会在周围出现红色框。有多种轮廓属性，包括outline，outline-color，outline-style和outline-width。为完成红色框的例子，可以声明：

```
IMG.button:hover {outline: solid red 1px;}
```

这样就会有所描述的效果。轮廓样式也可用在客户端图像映像的各区域，用来设置可见轮廓。

边框

在CSS1中，有很多用于给元素设边框的属性，如border-top-width和border-color及border自身。CSS2增加了很多边框属性，多用于使网页制作者能更加有效地控制边框。以前，为边框指定某边设置特别颜色或样式都很困难，只能通过border-left这样的属性，且需要设置多个值。新的CSS2属性致力于解决这个问题，它们的名字即说明了各自的意义：

```
border-top-color
border-right-color
border-bottom-color
border-left-color
border-top-style
border-right-style
border-bottom-style
border-left-style
```

表格

也许是为了描述表格的布局，CSS2包含许多可以直接应用于表格及表格单元的特性。首先，有10个与表格有关的新display的值：

```
table
inline-table
table-column-group
table-column
table-row-group
table-row
table-cell
table-caption
table-header-group
table-footer-group
```

尽管这些值中的大多数都可以从名字推知其效果，但其中至少有两个并不为人所熟知。table-header-group和table-footer-group用于标识表格的页眉与脚注。它们分别显示于表格中所有行的上面和下面，但不超出表格的标题。

另一个有趣的效果是，可以使用text-align属性在一个符号上对齐文本。例如，如果希望根据小数点来对齐一列数字，可以声明：

```
TD { text-align: "."}
```

只要单元集合位于同一列，它们的内容就会被对齐，使所有小数点共线于竖轴。

丝毫不依赖于现有的属性，CSS2提供了一组全新的表格属性，这里是其中的几个：

- border-spacing，影响单元周围边框的放置。
- border-collapse，用于影响可变表格元素的边框如何相互作用。
- table-layout，告诉浏览器是否可以在必要时重定义表格大小。

还有一些属性描述visibility和vertical-align如何应用于表格，还有一个caption-side属性，与<CAPTION>标签上的ALIGN属性相同，speak-header-cell属性，通过语音生成浏览器来控制标题的处理（稍后将有详述。）

媒体类型与 @ 规则

不要太激动，这里并不是要从音频、视频制作的角度来讨论媒体类型。这里要讨论的是用于在各种媒体上呈现的生成规则。媒体的定义类型是:

- screen，例如计算机屏幕。
- print，打印输出或打印预览显示。
- projection，用于方案描述，如滑动显示。
- braille和embossed，用于触觉反馈设备和打印机。
- aural，用于语音生成器。
- tv，用于电视类型显示（想一下WebTV）。
- tty，用于固定宽度字符显示。
- handheld，用于掌上电脑。
- 无处不在的all。

这些都是@media（新的@规则之一）。其他还有:

- @font-face，用于手工定义字体。
- @import，比在CSS1中有更强的功能，允许网页制作者结合媒体类型与@import声明；例如@(print.CSS)print。
- @page允许使用页式媒体样式表单定义一页的样式；例如，@page {size: 8.5in 11in;}。

页式媒体

刚才我们提到了页式媒体(paged media)，应该注意有新属性可应用于这种媒体。其中五个应用于页中断:

```
page-break-before
page-break-after
page-break-inside
```

```
orphans
widows
```

前两个属性用于控制一个页中断是否应出现于一个指定元素之前或之后，后两个是普通桌面发布术语，用于可出现于一页头部或尾部的最少行数。在CSS2中，它们的含义与在桌面发布中相同。

page-break-inside（首先由网页制作者提出）用于定义页中断是否可放置于指定元素中。例如，也许不希望在无序列表中出现页中断，就可以声明UL {page-break-inside: avoid;}。绘制代理（如打印机）将避免中断无序列表。

还有size，用于简单地定义一个页面是否应该用风景或肖像模式打印，以及各轴的长度。如果计划用专业打印系统打印页面，可能会希望使用marks，应用十字或剪切标记于页面上。可声明：

```
@page {size: 8.5in, 11in; margin: 0.5in; marks: cross;}
```

这会将页面设成US标准信件，8.5英寸宽，11英寸高，且在每页的角上都设置有十字标识。

另外，还有新的伪类:left,: right和:first，这些都只应用于@page规则。这样，可在两侧打印时为左侧页和右侧页设置相同的边界：

```
@page:left {margin-left: 0.75in; margin-right: 1in;}
@page:right{margin-left: 1in; margin-right: 0.75in;}
```

:first选择符只应于文档的第一页，因此可以给它较大的上边界或较大的字体大小：

```
@page:first {margin-top: 2in; font-size: 150%;}
```

语音控制

为完整起见，我们还要介绍一些关于听觉样式表单的属性。这些属性帮助定义一个语音浏览器如何读一个页面。也许对许多人这并不重要，但对视觉受伤的人来说，这却是必需的。

首先是voice-family，在结构上与font-family类似：网页制作者可定义特殊声音系列或普通声音系列。有控制读页面速度的属性（speed-rate），还可指定声音的pitch，pitch-range，stress，richness及volume属性，以及允许控制如何读缩略词、标点、日期、数字和时间的属性。有指定声音提示的方法，可用于指定元素（如超链接）之前、之中或之后，有在元素之间或之后插入暂停的方法，甚至可以通过属性azimuth和elevation控制声音发出的空间位置。使用这两个属性，可以定义一个样式表单，文本由用户前方的声音读出，而背景音乐位于用户身后，声音提示来自用户上方。

小结

CSS2显然覆盖了许多内容，如要探索其细节会使本书至少增加四章，还不包括要大大增加已有章节的内容。然而，因为CSS2实际被应用的部分很少，我们这里只提供一个概览而忽视一些细节。毕竟，规范会随应用中不断暴露的缺陷而改变，因此，本书坚持只描述可靠的部分。

为提供快速参考，表10-1给出了关于CSS2新属性的小结。

表10-1 CSS2新属性

CSS2的新属性		
text-shadow	position	border-spacing
font-size-adjust	direction	empty-cells
font-stretch	top	caption-side
unicode-bidi	right	speak-header-cell
	bottom	
cursor	left	volume
outline	z-index	speak
outline-color		pause-before
outline-style	min-width	pause-after
outline-width	max-width	pause
	min-height	cue-before
content	max-height	cue-after
quotes	overflow	cue
counter-reset	clip	play-during
counter-increment	visibility	azimuth
marker-offset	page-break-before	elevation
	page-break-after	speech-rate

表 10-1 CSS2 新属性（续）

CSS2 的新属性		
border-top-color	page-break-inside	voice-family
border-right-color	orphans	pitch
border-bottom-color	widows	pitch-range
border-left-color	size	stress
border-top-style	marks	richness
border-right-style		speak-punctuation
border-bottom-style	border-collapse	speak-rate
border-left-style	border-spacing	speak-numeral
	table-layout	speak-time

CSS2 中的新伪类与伪元素		
:hover	:righ	:before
:left	:first	:after

CSS2 中的新 @ 规则		
@media	@font face	@page

表 10-2 CSS2 中的新值

所有属性		
inherit		

display 属性		
run-in	inline-table	table-footer-group
compact	table-row-group	table-row
marker	table-column-group	table-cell
table	table-header-group	table-caption

font 属性	
caption	message-box
icon	small-caption
menu	status-bar

list-style-type 属性		
decimal-leading-zero	cjk-ideographic	hiragana-iroha
hebrew	hiragana	katakana-iroha
georgian	katakana	lower-greek
armenian		

表10-2 CSS2中的新值（续）

*color*值		
active-border	highlight	three-d-dark-shadow
active-caption	highlight-text	three-d-face
app-workspace	inactive-border	three-d-highlight
background	inactive-caption	three-d-lightshadow
button-face	info-background	three-d-shadow
button-highlight	info-text	window
button-text	menu	window-frame
caption-text	menu-text	window-text
gray-text	scrollbar	

vertical-align属性
length

第十一章

CSS应用

任何指导用书都应该留出一章的内容用于将所有理论知识付诸于实践，这也正是本章所要做的。在下面三个不同的使用CSS的页面设计方案中，会有大量的技巧与提示，这也许会帮助读者解决在使用CSS中遇到的一些问题。

转换方案

既然已经学习了所有的CSS1知识，让我们通过三个转换方案来练习一下新学到的知识。在这三个实例中，有两个是Web页面，另一个是印刷杂志文章。我们将页面分割成各个组成部分，然后使用CSS1及结构化的HTML以最好的方法重新生成同样的效果。

实例1：一致的外观

在这个方案中，将创建一个外部样式表来为整个站点定义基本的、一致的外观。主要目标是生成尽可能简单的样式，使用较少的类或ID。为达到方案的意图，我们假定公司职员有一个标准的书写向导：文档标题用H1，副标题用H2，每页顶部使用标准图形等等。

市场导向使得所有页面应当使用白色背景，背景的左边沿镶嵌一条细细的暗绿色

条纹；主体使用黑色serif字体文本；被访问过的超级链接呈暗灰色，未被访问时呈暗绿色。另外，文档标题必须带下划线，且与每页顶端的标准导航按钮使用类似的颜色，即暗绿背景灰色文本，暗绿色与未被访问的超链接及浏览器视窗左边沿的暗绿色相同。所有的标题，包括文档标题，都使用 sans serif 字体，其他的可自行决定。

这些都非常直观。对文档 BODY，可声明：

```
BODY {font-family: Times, serif; color: black;
  background: white url(pix/grstripe.gif) repeat-y top left;}
```

对于定位锚，需要知道所使用的绿色的颜色值。艺术设计部门认为，这种绿色调不应使用红色和蓝色，只使用40%绿色，因为有些有预见的人使用的是较安全的网络色。我们对访问链接做同样的处理，声明：

```
A:link {color: rgb(0%,40%,0%);}
A:visited {color: rgb(20%,20%,20%);}
```

这样就得到了暗绿与暗灰色的超链接。

接下来是标题。假定所有标题均为sans serif字体，但H1有一些特殊的规则。为了用压缩方式包含所有的内容，声明：

```
H1, H2, H3, H4, H5, H6 {font-family: Verdana,sans-serif;}
H1 {color: rgb(0%,40%,0%); border-bottom: thin solid; width: 100%;}
```

通过第二条声明，不仅使用了标准颜色，还增强了“下划线”的效果，它设置了一个从文本左边沿一直延伸到浏览器视窗右边沿的下边框。这条线也继承了文本的绿色，这样便把标题和导航按钮同页面的其他部分分开了。

做完所有这些之后，需要将样式表链接到站点的页面上。以上声明被收集到一个单独的文件中，该文件存储的URL为http://www.mycomp.com/style/site.css中。然后修改所有的站点页面，使HEAD元素中包含如下规则：

```
<LINK REL="stylesheet" TYPE="text/css"
  HREF="http://www.mycomp.com/style/site.css" >
```

这样便确保了所有文档，甚至那些没有自己的样式声明的文档，都使用站点的总体样式表。图 11-1 显示了页面如何显示的例子。

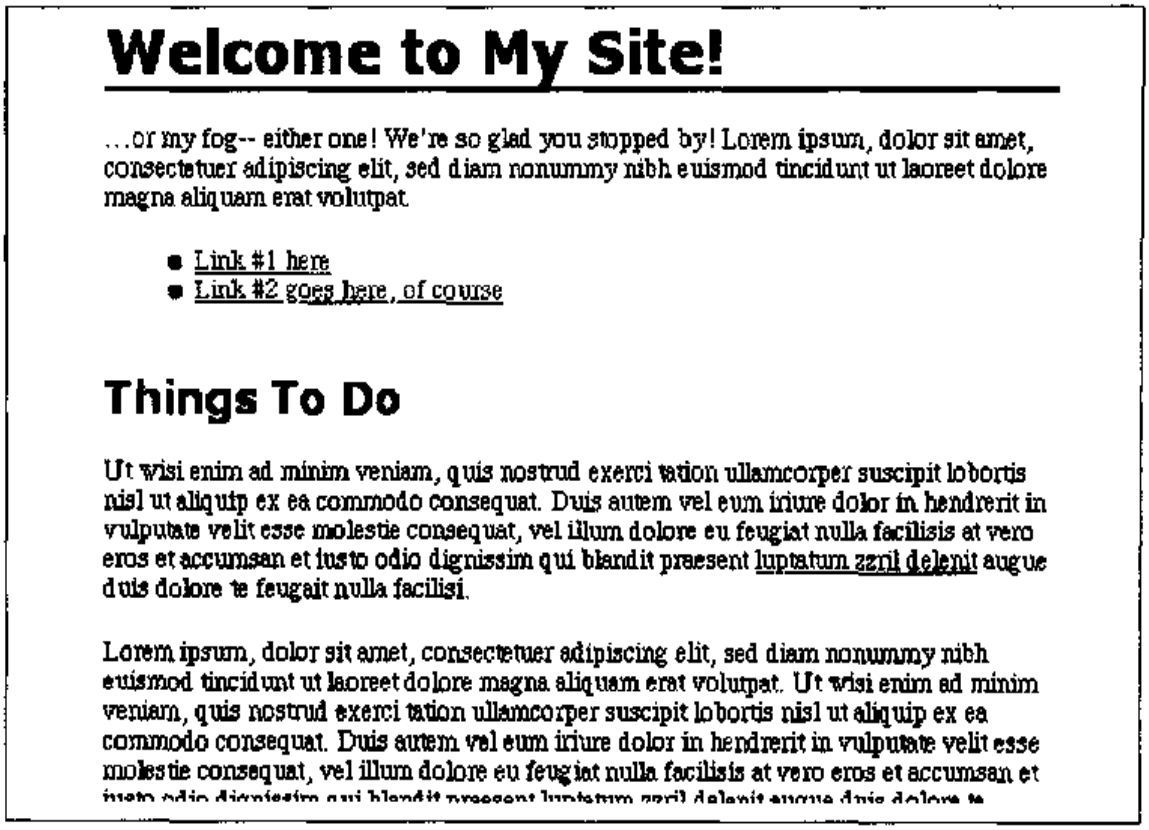


图 11-1 最终结果

实例 2：图书馆目录系统界面

Wattswith 大学的图书馆在过去几年中一直使用基于 Web 的图书馆目录系统，且 Web 开发人员总是尝试着与时代同步。既然样式表正在流行，图书馆人员渴望转换图书系统界面来使用它们。

然而，现有的设计已经很流行，于是管理者下达命令：不要改变外观。任务是拿来已有的页面、拆开它们然后再重组，不让人注意到有任何差别。（对于本设计方案，网络管理员的一部分角色将由读者来扮演。）

系统中最复杂的画面是记录显示画面。由三个区域组成 —— 系统导航条、边栏，及记录显示本身。记录组织在表格中，每个区域位于一个表格单元中，另外，有四个表格单元介于侧导航条与页面的主要部分之间，以生成一些空白。还有一些 font 标签及表格嵌在主表格中，主表格决定页面的布局。页面的轮廓用表格表示，增加了边框及单元补白以使结构更加清晰：

```
<TABLE CELLSPACING=0 BORDER CELLPADDING="10">
<TR>
<TD ROWSPAN=2>sidebar</TD>
<TD COLSPAN=2>navigation bar</TD>
</TR>
<TR>
<TD> </TD>
<TD>main display</TD>
</TR>
</TABLE>
```

外观如图 11-2 所示。显然，还有许多未列出的单元。为了更简明、清晰，真正的内容用标签替换显示。

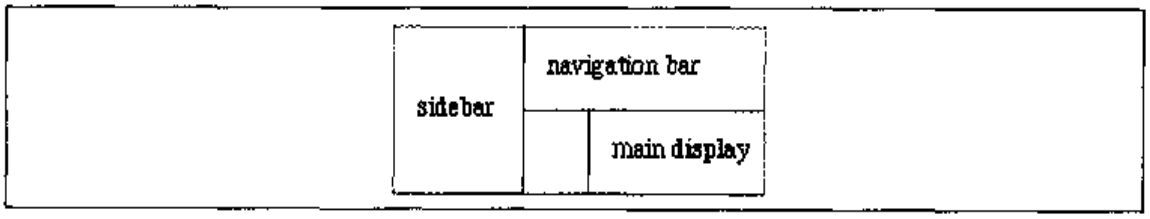


图 11-2　简化的页面结构

为了保证修改简单，并避免使用稍后讲解的技巧，在这个方案中，仍使用表格作为页面的总体布局，但有较小的修改。它只有两个单元：边栏及屏幕的其他部分，而不是四个。导航条成为主要显示的一部分，且空白表格单元完全被清除。剩下的部分为：

```
<TABLE CELLSPACING=0>
<TR>
<TD>sidebar</TD>
<TD>navigation bar and main display</TD>
</TR>
</TABLE>
```

首先，是导航条。每套链接分组成列表，放于主标题之下，这些标题看起来与链接不同。每个部分都使用如下标记：

```
<P>
<FONT FACE="Verdana" COLOR="white" size="+1"><B><U>Heading</U></B></FONT>
<BR>
<FONT FACE="Verdana" SIZE="-1">
<A HREF="link1.html"><FONT COLOR="yellow">Link</FONT></A><BR>
<A HREF="link2.html"><FONT COLOR="yellow">Link</FONT></A><BR>
```

```
<A HREF="link3.html"><FONT COLOR="yellow">Link</FONT></A><BR>
</FONT>
</P>
```

这样就完成了要做的工作。

也许最简单的方法是给边栏的表格单元指定一个类，这样可以为边栏指定特定的外观。这需要在单元首尾加入<TD CLASS="sidebar">和</TD>标记。

现在已经把边栏置于其自身的表格单元类中。因为边栏的背景颜色是绿色，可生成第一个样式：

```
sidebar {background: green;}
```

在导航条中，还希望消除所有的FONT标签，还有其他样式标签（如<B>和<U>）。因为边栏使用字体Verdana，可在样式表中增加：

```
sidebar {background: green; font-family: Verdana; sans-serif;}
```

使用font-family的原因是不希望给边栏指定font-size，因此不能使用font。

现在可以将标题与链接列表放入各段落中，然后对段落加入补白、边界，设置行高，直到与页面外观匹配。然而，简单的办法是不对段落及换行标签做任何处理的P.SPAN标题：

```
<SPAN CLASS=head">Heading</SPAN><BR>
```

警告： 使用SPAN而不用逻辑元素，如标题，其危险在于有些浏览器不能识别SPAN元素。同样，索引机器人不能将SPAN当做文档结构来对待。另一方面，使用SPAN避免了必须处理早期CSS实现中的错误，因此在这个实例中选择了使用SPAN，而不是一些更结构化的元素。

做完这些后，需要声明一个样式来重新生成删除掉的标签的效果。使用：

```
.sidebar .head {font-size: larger; font-weight: bold;
  text-decoration: underline; color: white;}
```

通过使用上下文选择器.sidebar .head，确保只有在.sidebar中的.head才能

接受这些样式，由于整个边栏已经设为使用 Verdana，标题将继承并使用它。至于链接，需要为黄色，所以声明：

```
.sidebar A:link {color: yellow;}
.sidebar A:visited {color: yellow;}
.sidebar A:active {color: yellow;}
```

这些声明会保持链接为黄色。图 11-3 显示了上面新改进的、没有 Font 标签的边栏，由以上的声明及下面的标记产生：

```
<TD CLASS="sidebar"><P>
<SPAN CLASS="head">Heading</SPAN>
<BR>
<A HREF="link1.html">Link</A><BR>
<A HREF="link2.html">Link</A><BR>
<A HREF="link3.html">Link</A><BR>
</P>
</TD>
```

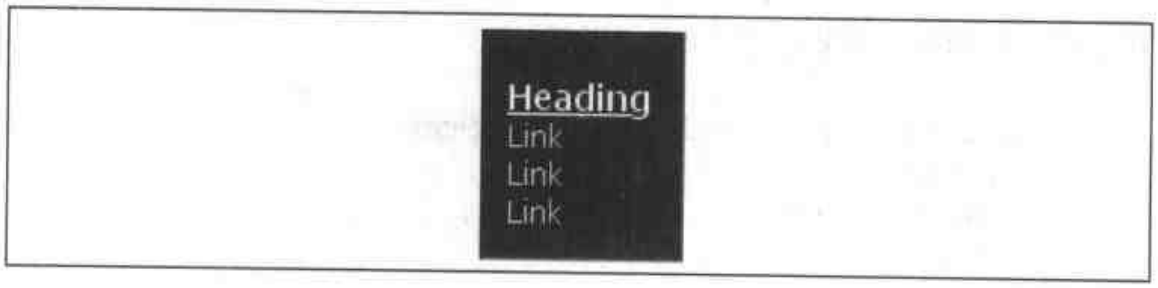


图 11-3　有较好样式的边栏

这很容易吧？现在要处理页面主要部分上方的导航条。这个区域也是绿色背景，其中有一些图片。使用有特定类的 DIV 标签：

```
<DIV CLASS="navbar">
icons
</DIV>
```

现在所需要的只是样式 navbar {background: green;}。

但这还不够。在旧的页面中，导航条与主显示区之间有较小的间隔，但却紧贴边栏，这样就形成了一个反转的绿色“L”形状。我们希望在新构建的网页中仍是如此。通过确保分隔没有设置补白和边框，可保证其宽度与表格单元的宽度相同。

此外，希望导航条下面还有一些空白，因此需要对底端设一个非零边界：只需要几个像素即可。于是我们可以加入：

```
navbar {background: green; padding: 0; margin:0 0 10px 0; width 100%;}
```

导航条的设置完成，如图11-4所示。现在需要使主显示区左侧有一些空白。

图11-4 绿色导航条

毫无疑问，读者已知道它将如何工作。下面生成另一部分，这个类叫做main，包含页面主要部分中除导航条外的所有内容。声明：

```
.main {margin-left: 1.5in}
```

在图11-5中检查其效果，该效果基于以下标记：

```
<TABLE CELLPADDING="0" CELLSPACING="0" BORDER>
<TR VALIGN="top">
<TD CLASS="sidebar">
(sidebar)
</TD>
<TD>
<DIV CLASS="navbar">
(icons)
</DIV>
<DIV CLASS="main">
(content)
</DIV>
</TR>
</TABLE>
```

与最初的布局相比，还是有一些小的区别，但总体上，在文档显示方面没有明显的不同。可以为之感到欣慰，且管理人员不会注意到这点差别。

新设计有两个优点：可以通过编辑较少的样式来改变颜色、字体，而不必修改大量的Font标签；HTML源文件可以得到精简。在这种实例中，页面源文件可以缩减几KB，在大量使用Font标签的实例中，将之转化成使用CSS，文档文件的大小可以减少50%。

图 11-5　最终结果

实例 3：使杂志文章在线

最后，转到《Meerkat Monthly》编辑室所面临的状况。这个专门的杂志负责调查在家庭环境中饲养某些动物的问题，与其他同类杂志不同的是，它还提供动物的一般知识。为刺激销售，编辑人员希望每月将一些文章放到网上。

注意： 在这个实例中使用了很多先进的样式，也许超出了许多用户代理的能力。它很有指导性，且应工作于能够完全、正确支持CSS1的浏览器上。如果在浏览器上尝试这个例子，请做好出错的准备。可以一次执行一步，观察其效果，以便知道哪里出现了错误。

为了试运行，他们决定使用一个一页的文章，谈论该动物在野生环境中的生活规律及对人的一般行为，这篇文章在杂志中如图 11-6 所示。

显然，《Meerkat Monthly》的职员在其桌面印刷程序中使用了一些技巧。很难使网页与书中的外观一致，但我们还是要试着去做。

首先，取来文档布局并决定哪些可以去除。因为 Web 上没有页，页数可以去掉。同样，外边界可按需要修改，因为无需考虑为主题等留出额外空间。然则，编辑希望保留双栏的布局、图片的旋转以及文本的外观。

SOCIETY KATS

The meerkat is an animal of endless fascination. Possessed of what seems to be almost human intellect, living in cooperative societies, and undeniably cute, the popularity of these animals is easy to understand. Dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

"The meerkat is a fun, smart, but often exasperating fellow."

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

图 11-6 原始印刷文档

先来生成双栏。请记住，我们不希望使用表格或像 MULTICOL 这样的标签，因此必须求助于其他的方式。在本实例中，由于每栏都有很多段落，所以我们可以使用DIV标签。所要做的就是将文章的文本一分为二，每一半都放到一个DIV中（文章文本，指的是文章真正的文本，包括标题）。使用印刷页面中栏结束的位置作为第一部分结束的向导：

```
blah blah blah.
</P>
</DIV>
<DIV>
<P>
Blah blah blah
```

做完之后，使用下面的样式来修改第一个 DIV：

```
<DIV STYLE="float: left; width: 40%; margin-left: 10%; margin-right: 5%;">
```

这将使得第一部分的全部文本成为一个位于左边界的浮动块，接下来的文本流经其右侧。换句话来说，是一个两栏布局。第一栏的宽度被声明为浏览器视窗宽度的40%，左边界为视窗宽度的5%。这会使第二栏总宽度自动计算为45%（40 + 10 + 5=55，100 − 55=45）。

这样，两栏将不等宽，如图 11-7 所示，但这正是所要的效果。

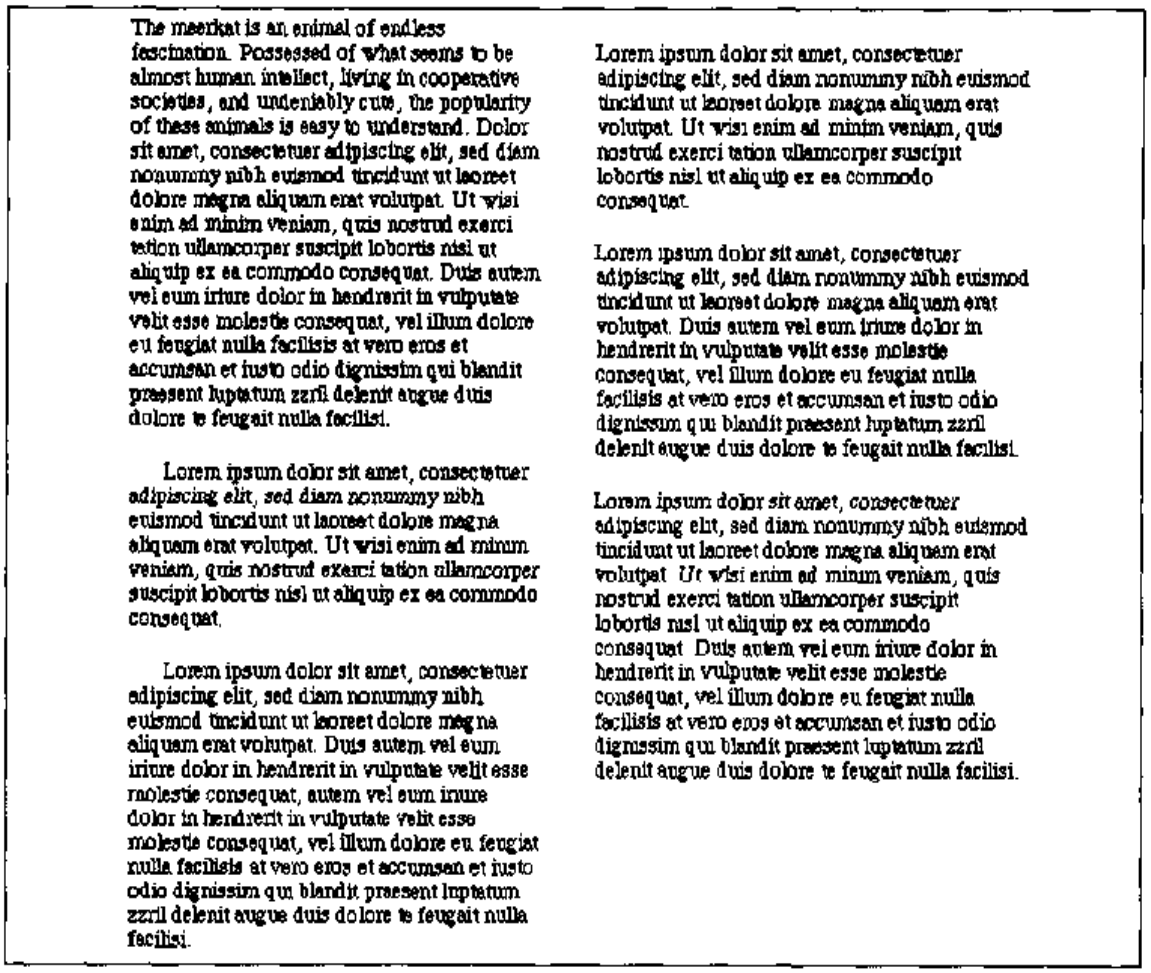


图 11-7　两栏效果

也许最终仍必须调整各区域的位置，但此时，不妨先让它这样。

实际上，还要增加一个标记：

```
<DIV STYLE="float: right; width: 45%;">
```

这是第二栏的DIV。为什么要使它浮动呢？假定第二栏比第一栏长，如果第二栏未设 float，将会生成如图 11-8 的情形。

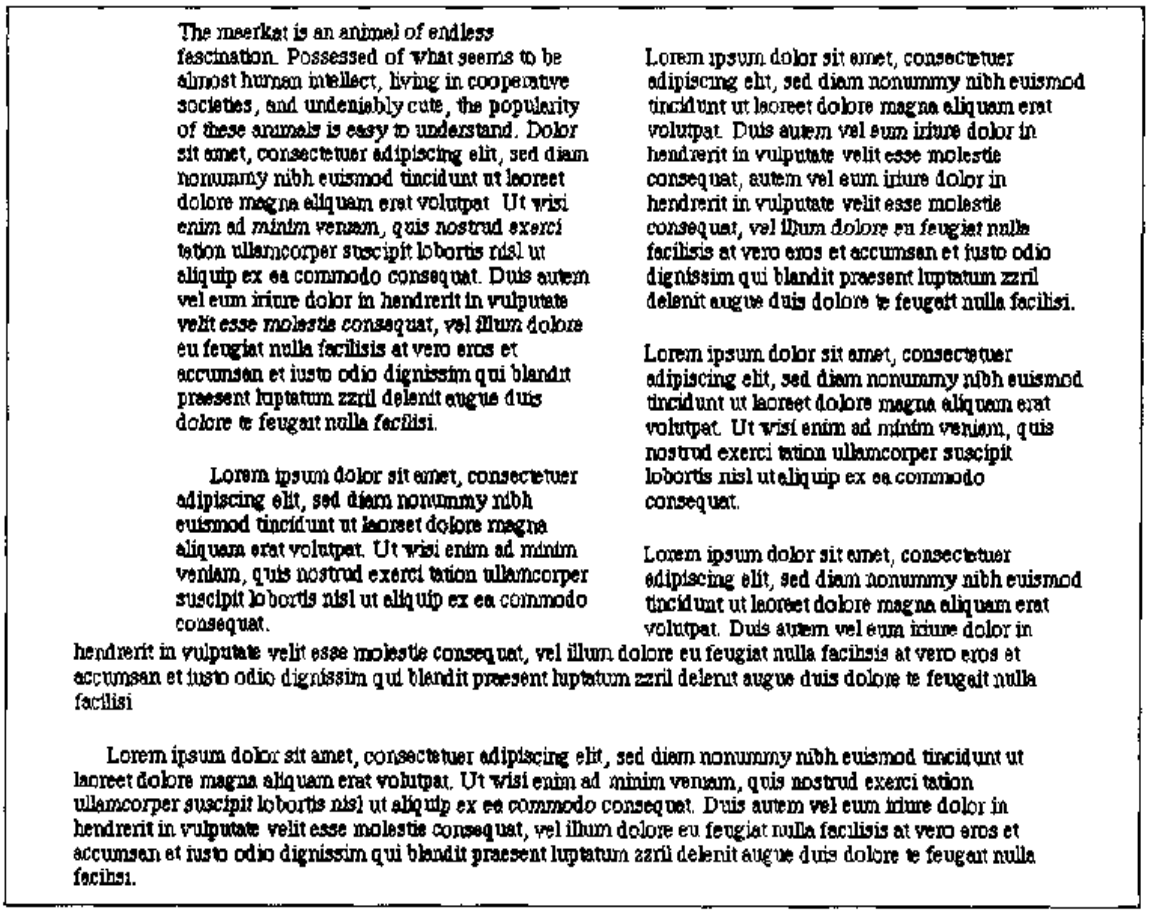


图 11-8 为何两次设置浮动

这与浮动规则一致，但显然不是所希望的。通过设置第二栏浮动，就可以完全避免这种情况，保持第二栏的形状。

现在我们来放置图像。共有两个图像，都在第一栏中，这使事情简单了许多。显然，它们是向左浮动图像。有趣的是重新生成它们，并使它们凸出到栏左边空白部分的效果。

如果只将图像样式设为 float: left，它们将完全包含于第一栏中。然而，由于第一栏有左边界，所以只需给图像设负 margin-left 值：

```
IMG {float: margin-left: -2.5em;}
```

这里有一个潜在的危险。浮动图像左边界为2.5em，而栏自身的左边界为10%。在足够窄的浏览器视窗中，栏的左边界可能会少于2.5em。如果发生这种情况，图像会因过于偏左而超出视窗范围。这样混合使用单位，即使不是直接的，也有风险。更好的选择是：

```
IMG{float: left; margin-left: -10%;}
```

这样，图像的左边界可根据环境伸缩。

由于只有两个图像，且要求同样的效果，这个声明可以完成要求。如图11-9所示。

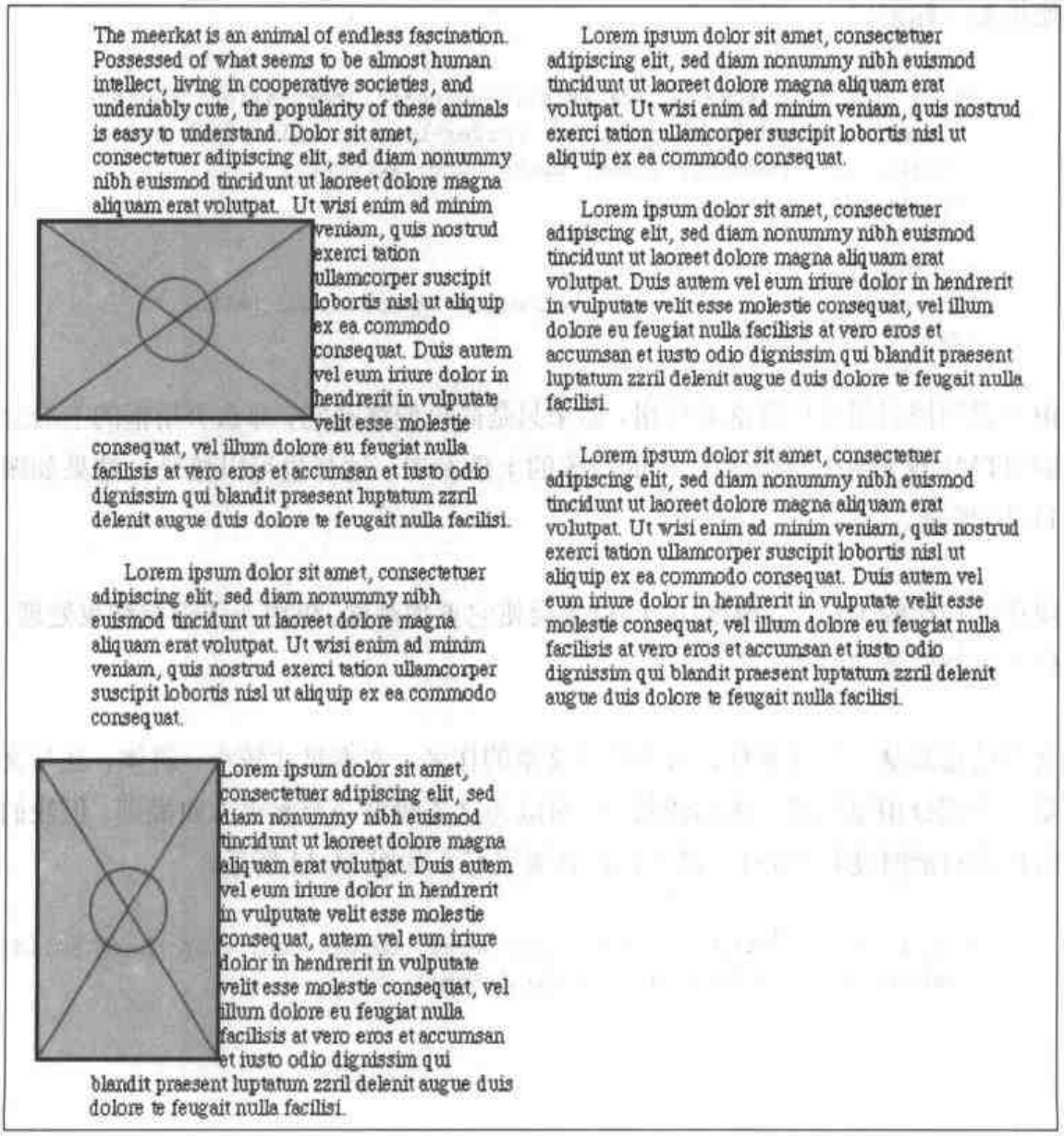


图 11-9　浮动图像

可以看出，第一栏比第二栏长许多。然而，还有很多工作没对第二栏进行，所以先把它放在一边。

用大一些的字体显示引用的偏移文本块经常被称为“pull quote”。本例中就有一个，且位于第二栏的中部，这里我们来决定如何处理它。首先，文本比文章的字体稍大一些，且是sans serif字体。而且，在引用框的顶端与底端有漂亮的线，两者都稍稍超出了文本的左右边框。背景是浅灰，且在框的周围有一些空白把它同主体文本区分开来。引用框的文本居中对齐，宽度约为栏宽度的一半，且显然是向右浮动。

使用如下标记：

```
.pullq {font: 150% Helvetica,Arial,sans-serif; text-align: center;
  border-top: medium black solid; border-bottom: thin black solid;
  margin: 1em; padding: 0.5em; background: #CCCCCC;
  width: 50%; float: right;}

<P CLASS="pullq">
"The meerkat is a fun, smart, but often exasperating fellow."
</P>
```

由于我们把引用当作段落来应用，如果只是简单地浮动它，那么引用框的上沿会同HTML文档中位于引用之后的段落的上沿对齐。这样也可以接受，结果如图11-10所示。

现在文档比较匀称了，调整DIV的位置会使它更加对称，但由于还没有结束处理，仍暂时把它放在一边。

文章结尾处是一个文本框，用来介绍文章的作者。文本尺寸较小，斜体，且与文章其余部分用空白及一条细线隔开。可以为这条线设一个水平方向规则，但我们坚持在可能时使用CSS1。以下标记效果很好，如图11-11所示：

```
.author {font: italic x-small Times,serif; border-top: thin black solid;
  padding-top: 0.25em; margin-top: 0.5em;}
```

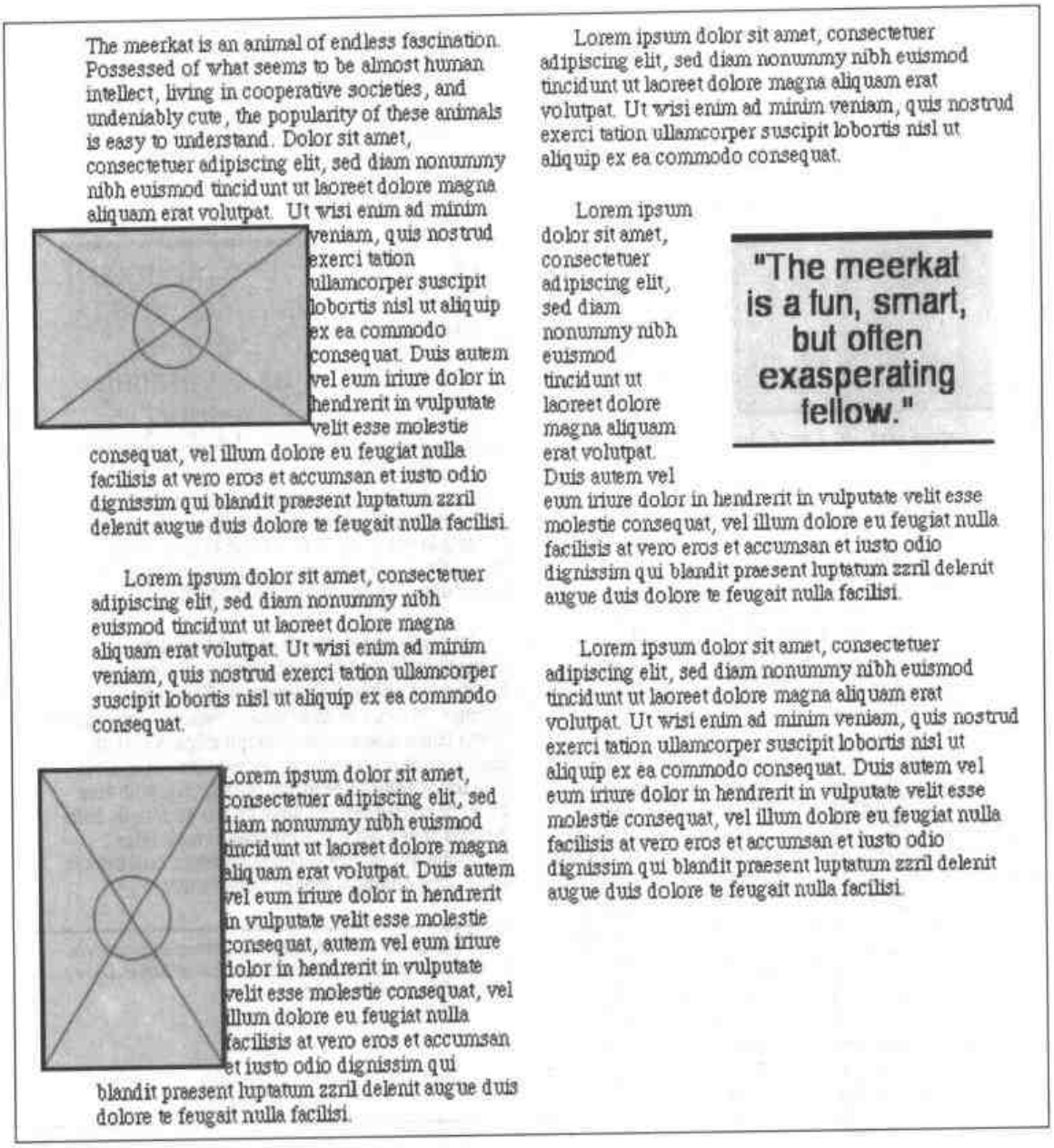


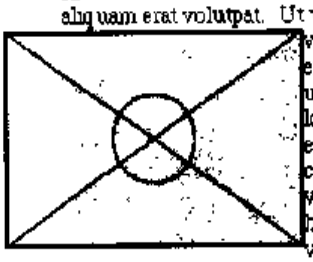
图 11-10 一个引用

在生成标题之前，让我们已设置完文章主体的细节，文章主体使用 serif 字体。我们觉得它应该更易读，所以使用 Times 字体，我们还希望文章为白色背景、黑色文本，这样比较接近打印文本的外观。每个段落首行缩进约半英寸，将之减到四分之一英寸以适应 Web 形式。使用如下标识。

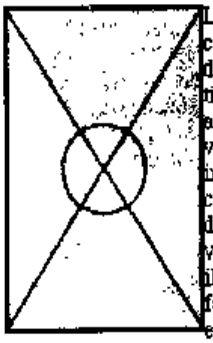
```
BODY {color: black; background: white;}
P {font-family: Times,serif; text-align: justify; text-indent: 0.25in;}
```

图 11-12 显示了段落的外观。

The meerkat is an animal of endless fascination. Possessed of what seems to be almost human intellect, living in cooperative societies, and undeniably cute, the popularity of these animals is easy to understand. Dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

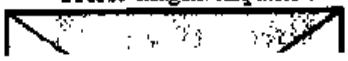
"The meerkat is a fun, smart, but often exasperating fellow."

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat

图 11-11 给作者栏设置样式

The meerkat is an animal of endless fascination. Possessed of what seems to be almost human intellect, living in cooperative societies, and undeniably cute, the popularity of these animals is easy to understand. Dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis



nostrud exerci tation

图 11-12 段落缩进

不管用户监视器分辨率的设置如何，最后一条规则给出了每个段落缩进的数量。然而，文章第一段的第一个字母比较特殊，它比较大且从第一行开始向下延伸，使后续的行紧跟其后。因此，第一个段落的第一行就没有缩进，我们必须自己进行调整，过程如下：

```
.initial {text-indent: 0;}
P.initial:first-letter {font-size: 200%; float: left;}
```

显然，这些规则要求给第一段落的标签上增加属性class="initial"。而且只要这个声明块出现在样式表的后面，声明：text-indent: 0;就可以覆盖先前声明的值。它们比正使用的其他样式特殊，原因是一个类选择符的出现，它也使这些规则可得到预期的效果。:first-letter的值使初始段落的第一个字母为通常大小的两倍，且向左浮动，如图11-13所示。

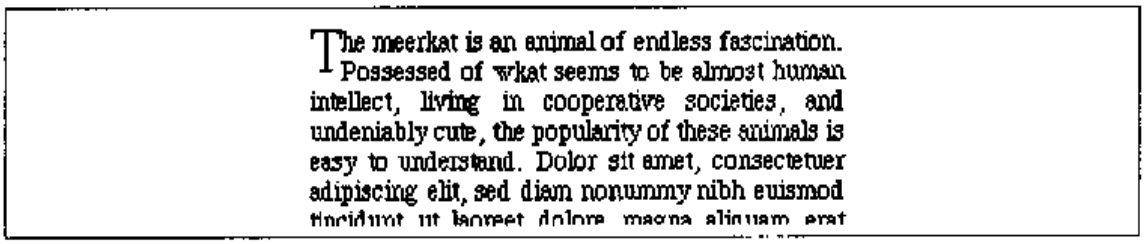


图11-13 首字母样式

注意：在CSS2中，第一个段落不使用类也可得到同样的效果。可通过使用新的CSS2选择符完成，如相邻兄弟选择符：

```
H1 + P {text-indent: 0;}
```

在支持CSS2的用户代理中，会将任何紧随H1元素的段落的text-indent设置为0。然而，由于这里段落是DIV的下级元素，而不是H1元素的直接子元素。因此，需要增加一个子选择符以及一个长子（第一个孩子）伪类：

```
H1 + DIV > P:first-child {text-indent: 0;}
```

这条规则匹配那些紧随H1的DIV的第一个子元素。请参见第十章“CSS2展望”，可得知更多细节。

将文章主体的外观设置好之后，所需做的是调整各部分的位置使两栏的长度相同。现在可以着手做这件事，因为不论如何处理标题，两栏的长度均应相同。应适当

地移动各部分。注意也许无法得到精确的平衡，因为那样势必会中断段落的划分。两栏之间的差别可由设计者自行决定。

做完这些之后，剩下的是重新生成文章的标题。仔细观察图 11-14，我们可看到它的特点很有趣：它是右对齐的，但又没有跟文档的右边界对齐；字母之间的间距很大；第一个字母远大于其他字母；所有字母均设置为 sans serif 字体。

图 11-14 初始文档的标题

我们将它当作 H1 元素并对它设置样式，而不是为标题生成一个新的类。从外观上观察，文本大小是文章主体文本大小的三倍，字母之间的间距约为一个字母大小。我们先从简单的做起，为标题 H1 声明：

```
H1 {font: 300% Helvetica,sans-serif; font-variant: small-caps;
  letter-spacing: 0.75em;}
```

如同前面观察的那样，标题是右对齐的，但没有到右边界的位置，最简单的方法是在 H1 右侧插入补白，引出如下声明：

```
H1 {font: 300% Helvetica,sans-serif; font-variant: small-caps;
  letter-spacing: 0.75em; text-align: right; padding-right: 1em;}
```

图 11-15 显示了此时的进展。

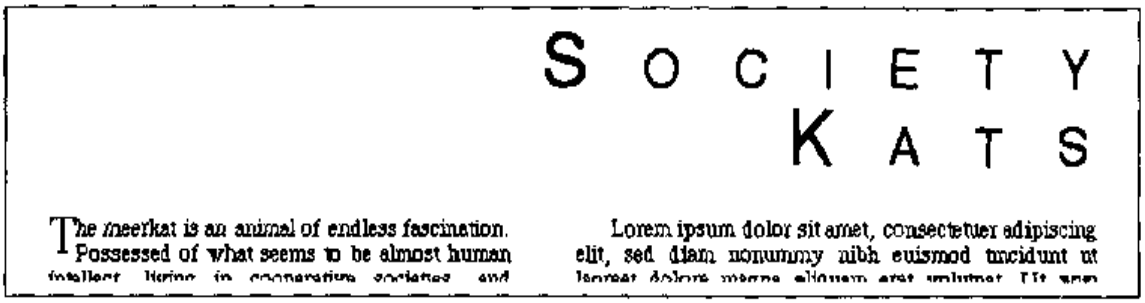


图 11-15 标题样式：中间结果

距最后结果很近了。实际上，只需对标题的第一个字母进行处理。可以通过:first-letter选择符来处理它。“S”大约是“kats”中“K”的两倍大，因此设置：

```
H1 {font: 300% Helvetica,sans-serif; font-variant: small-caps;
  letter-spacing: 0.75em; text-align: right; padding-right: 1em;
  line-height: 1em;}
H1:first-letter {font-size: 200%; line-height: 1px; vertical-align: -100%;}
```

生成的结果如图11-16所示，看起来大致正确了。

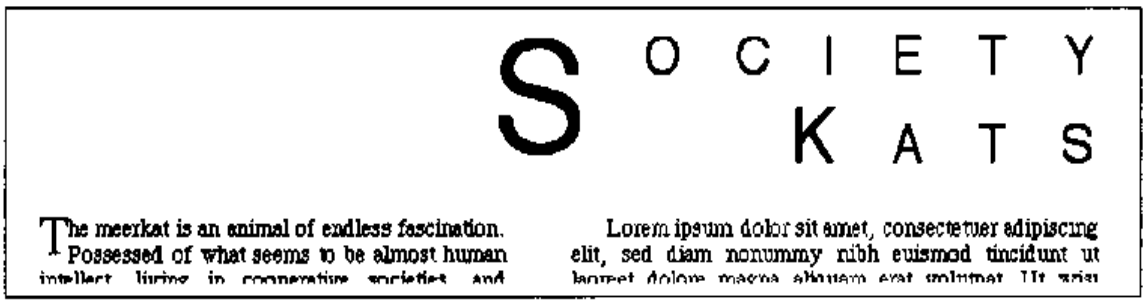


图11-16 最后样式的标题

line-height和vertical-align的值还值得讨论一下。所做的是“S”的内联框减少到只有一个像素高。（可以使用任何值，只要数量很少。）如第八章所述，内联框垂直居中于“S”中。这样“S”的基线被降低到下一行文本的基线（由于-100%垂直对齐使基线降低上级元素的font-size距离）。通常，这会使第一个行框相应变高，但由于声明“S”为line-height: 1px，内联框很小，以至于几乎对行框的高度没有影响。尽管图11-16所示的标题不能精确匹配图11-14中的标题，但看起来很接近。

合起来，以下是全部的样式表：

```
BODY {color: black; background: white;}
P {font-family: Times,serif; text-align: justify; text-indent: 6em;}
IMG{float: left; margin: 0.5em 0.5em 0.5em -10%;}
.pullq {font: 200% Helvetica,Arial,sans-serif; text-align: center;
  border-top: medium black solid; border-bottom: thin black solid;
  margin: 1em; padding: 0.5em; background: #CCCCCC;
  width: 50%; float: right;}
.author {font: italic x-small Times,serif; border-top: thin black solid;
  padding-top: 0.25em; margin-top: 0.5em;}
```

```
.initial {text-indent: 0;}
P.initial:first-letter {font-size: 200%; float: left;}
H1 {font: 300% Helvetica,sans-serif; font-variant: small-caps;
  letter-spacing: 0.75em; text-align: right; padding-right: 1em;
  line-height: 1em;}
H1:first-letter {font-size: 200%; line-height: 1px;vertical-align: -100%;}
```

图 11-17 显示了初始文章与在线文章的比较。

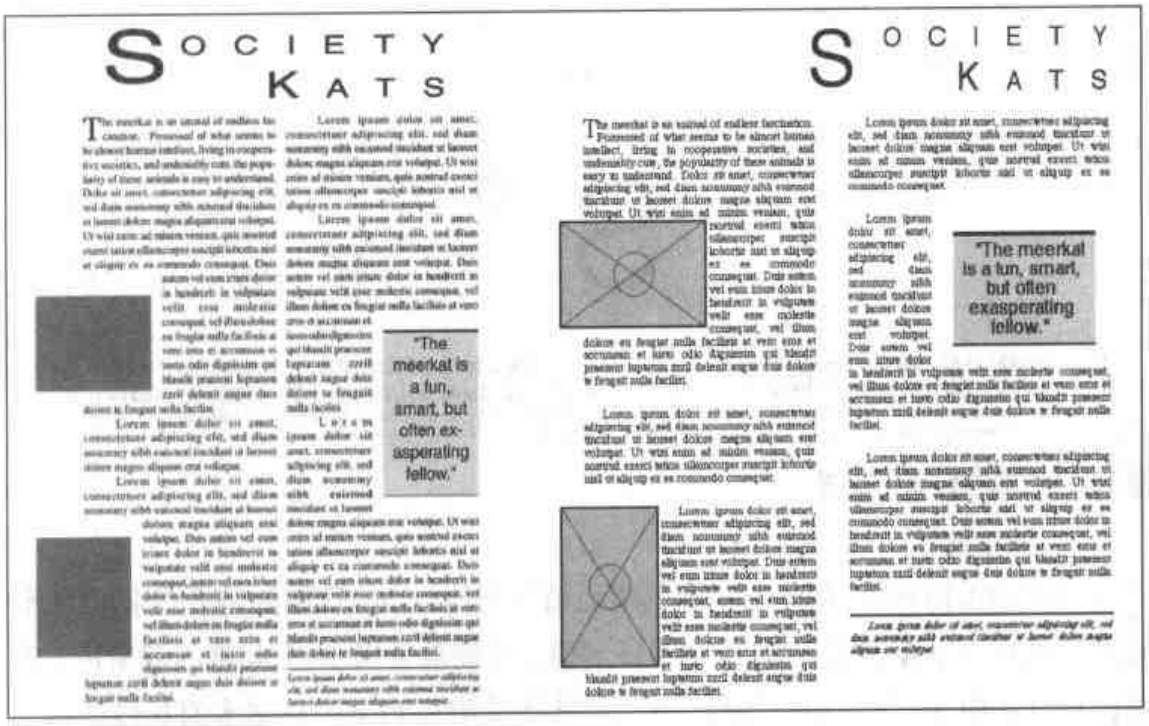


图 11-17 比较

如果使用没有样式表的浏览器浏览该网页，则结果如图11-18所示。尽管不雅观，但仍可读。

最后清理

仔细研究图 11-17，我们发现还存在一些不同之处，当然，后来生成的栏看起来差一些。如何解决这个问题呢？

两个布局中最明显的视觉差异在于标题。印刷版文章标题被拉伸。这个效果可使用CSS2 属性 font-stretch重新生成，但遗憾的是，本书写成之时，该属性仍未被支持。见第十章介绍的 font-stretch 属性。

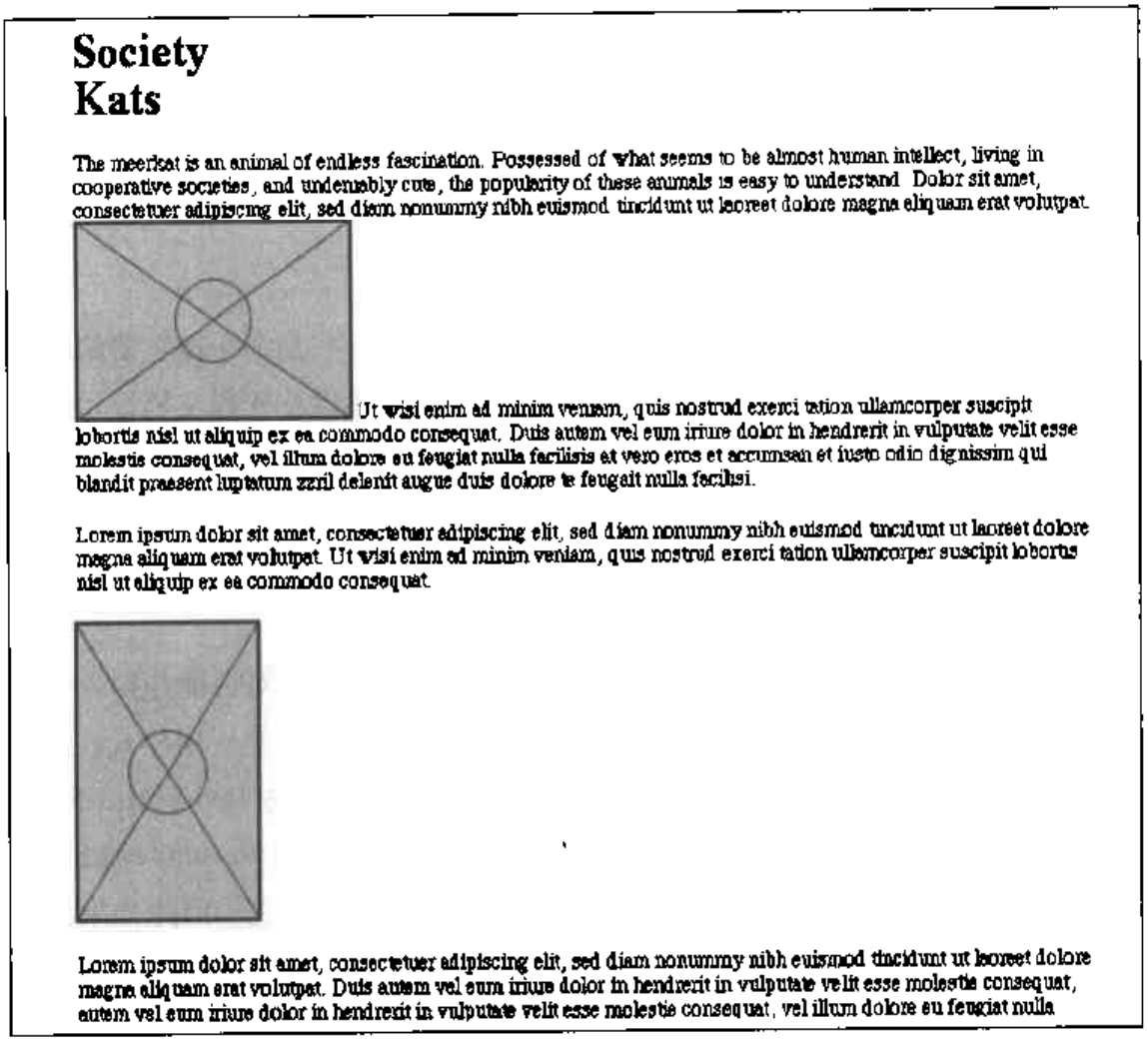


图 11-18 没有任何样式的页面

第一个大写字母“T”也不够匹配。这也可以使用 font-stretch来解决，或设字母的 font-weight 为900。然而，也许最好是不去处理它，因为这只是不重要的效果差别。

然而，怎么处理栏呢？为了使栏适合显示，我们被迫将各栏封装在 DIV 中。尽管这个方案比使用表格好，但仍必须对页面结构进行强制处理，且通过增加元素来生成视觉效果并不是一种好方法。更好的方法是设置 BODY 内容流入两栏中。但 CSS2 不包含栏或栏流的规范。CSS 协会已经讨论增加这些行为，将来也许会实现。现在，被迫采用这种增加 DIV 的途径来代表栏。

当然，这里假定希望生成栏。我们花费了很大的力气来得到这些栏，但真的值得吗？多栏布局在显示器上很难阅读，因为用户也许必须下拉滚动条读完第一栏，

然后回到顶端去读第二栏，继续下拉。增加栏在理论上是一个有趣的实验，但不是Web实现上的好方案。

提示与技巧

这里提供了大量的提示，也许可以节省你的许多时间。一些是关于浏览器的错误行为，其他则描述了书写完全正确的CSS与HTML的方法以及管理文档显示的途径，这里的方法有所简化，未考虑一个行为的后续操作。

使样式工作

这比较容易。如果希望Navigator 4使用CSS，必须打开参考对话框并检查样式表框与JavaScript框。如果JavaScript被禁止，Navigator就不能应用样式。为什么呢？在早期的样式表中，有许多种样式提案。其中之一就是JavaScript Style Sheets（JSSS），早期的CSS与JavaScript的混合。JSSS是由Netscape提出的。尽管JSSS从未被采纳过，但Navigator 4的绘制引擎仍使用它，因此如果没有JavaScript，CSS就不能工作。

通过@import隐藏样式

Navigator可理解LINK，但不能识别@import，可以利用这个特点。因为外部样式表中的样式只能经由LINK或@import引入，可以将所有可能引起Navigator问题的样式分组到一起，并放入待引入的样式表中。由于Navigator拒绝引入这个样式表，就不必处理它不能处理的样式。这个简单的技巧可以省去许多麻烦，但有一个缺点：少数早期版本的Navigator 4.x在试图处理@import声明时会崩溃。但这个bug很快被解决了，现在只有极少的出错版本还在使用中。

通过@import解决边界问题

如果希望使用在Navigator中不能工作的margin规则，使用上一个技巧及层叠可从中受益。假定希望文档的段落在其顶端与H1元素底端之间没有垂直空间，如图11-19所示。

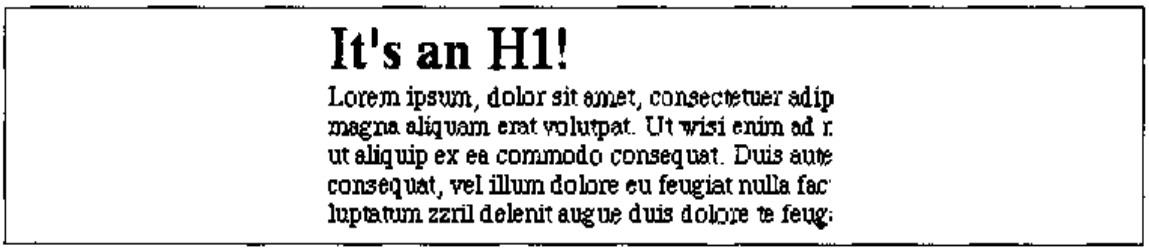


图 11-19 消除通常的间隙

在 Explorer 中，可使用如下规则完成：

```
H1 {margin-bottom: 0;}
P {margin-top: 0;}
```

在 Navigator 中，必须将段落的上边界设为 -1em 才能得到该效果，但这会搞乱 Explorer 的文档显示。如何解决这个冲突？

首先，将所有在 Navigator 不能实现的规则放入外部样式表中，并使用 @import 声明与之挂钩。然后将所有与在 Navigator 中实现较好的边界规则放入另一个外部样式表中，并用 LINK 引入它。（确保 LINK 先于 @import 声明出现。）结果如下：

```
/* file 'link-styles.css' */
H1 {margin-bottom: 0;}
P {margin-top: -1em;}

/* file'import-styles.css' */
H1 {margin-bottom: 0;}
P {margin-top: 0;}

<LINK REL="stylesheet" TYPE="text/css" HREF="link-styles.css"
   TITLE="Linked">
<STYLE TYPE="text/css">
@import url(import-styles.css);
</STYLE>
```

因为Explorer会读入两个样式表，将通过层叠决定使用哪些规则。如果排序正确，引入样式表出现于链接样式表之后，那就会使用引入样式表中的规则，而不是使用链接样式表中的规则。

因此，Explorer 会使用 import-styles.CSS 中的样式。Navigator 则甚至于不读待引入的样式，只有 link-styles.CSS 中的样式可用，于是便使用它。

样式化普通元素

如果文档中有普通标记的固定块出现，比如，一个包含链向站点主要页面的链接表，不必在每个页面中修改HTML标记，我们很容易就可样式化这些元素。

假定有如下的链接表:

```
<TABLE BORDER CELLPADDING="4">
<TR>
<TD><A HREF="home.html">Home Page</A></TD>
<TD><A HREF="read.html">My Writing</A></TD>
<TD><A HREF="fun.html">Fun Stuff!</A></TD>
<TD><A HREF="links.html">Other Links</A></TD>
<TD><A HREF="write.html">Contact Me</A></TD>
</TR>
</TABLE>
```

然而，在每一页上，我们希望包含本页的单元用某种风格来高亮显示。这很容易，只需为每个表格单元增加类即可，如下:

```
<TABLE border cellpadding="4">
<TR>
<TD CLASS="home"><A HREF="home.html">Home Page</A></TD>
<TD CLASS="read"><A HREF="read.html">My Writing</A></TD>
<TD CLASS="fun"><A HREF="fun.html">Fun Stuff!</A></TD>
<TD CLASS="links"><A HREF="links.html">Other Links</A></TD>
<TD CLASS="write"><A HREF="write.html">Contact Me</A></TD>
</TR>
</TABLE>
```

这样，在每页上，我们只需书写一个合适的样式。如果高亮链接应具有黄色背景，则在“Other Links”页中增加下面的规则到样式表，生成图11-20所示的结果:

```
TD.links {background: yellow;}
```

图11-20 高亮显示当前页

类似地，在站点主页，将会在页面上方找到该样式:

```
TD.home {background: yellow;}
```

这是使“工具条”更活跃的快速、简单的途径，无需为特定表格单元配置BGCOLOR属性。

注意：采用本方案，可以取出工具条并将其分割为独立的文件，然后以服务器端引用（*server-side include*）方法在每页面中引用那些文件。Include 在《Web Design in a Nutshell》（Jennifer Niederst 著）和《Apache:The Definitive Guide》（Ben Laurie 与 Peter Laurie 合著）（译注 1）两书中有更详细的描述，这两本书均是由 O'Reilly 出版的。

在 Navigator 中设置全部内容背景

在第六章中，我们谈到过这个问题，但仍值得重复一次。假定希望使用Navigator 4.x的用户可以在文本元素中看到全部背景颜色，而不只是位于文本之后。如果已对文本元素应用背景颜色，可增加声明：`border:0.1px solid none`。这没有可视效果，但在告诉Navigator绘制一个0.1像素、实心、不出现的边框的过程中，背景颜色会充满整个内容区及补白。如果设置可见边框，则在补白与边框间会出现一个间隙，但在其他方面会得到正确的效果。

如果省去了这个声明，所有版本的Navigator 4.x都不会将背景颜色拓展到整个内容区中，只是在元素文本的后面出现。

惊人的缩小文本

有件有趣的事情：使文档文本小到用肉眼无法阅读的地步。可以通过正确使用CSS及无错误浏览器来完成。下面是一个最简单的方法：

```
UL {font-size 75%;}
```

这很简单：无序列表文本大小为普通大小的75%。但如果无序列表嵌套于无序列表时会怎样呢？结果如图 11-21 所示。

译注 1：《Web Design in a Nutshell》的中文版《Web 设计技术手册》，《Apache 权威指南》影印版已由中国电力出版社出版，详情可查询 www.infopower.com.cn。

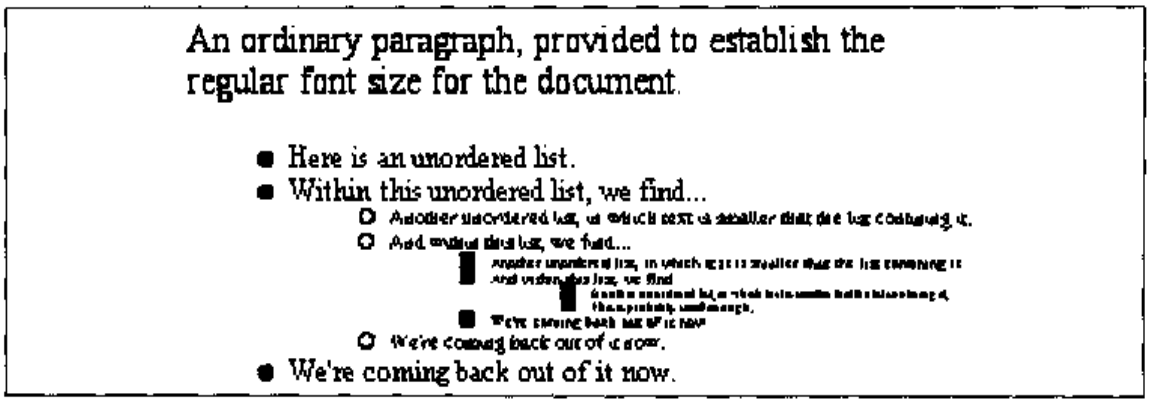


图 11-21 Help me

发生了什么？每层嵌套将字体大小减少四分之一，假定文档基准 font-size为12pt。这样，在第一层，字体大小为12pt的四分之三，即9pt。下一层减至6pt，再下一层为4pt，依此类推。一旦文本小于7pt，则在大部分浏览器上无法阅读（且在多数打印输出设备上乱成一团，无法阅读）。

读者也许会想："哈！为什么要这样缩小列表文本呢？"是的，很容易发现列表中这样的问题。然而，想一下页面大多是如何组织的（使用嵌套表格）并考虑这个规则：

```
BODY {font-size: 12pt;}
TD {font-size: 80%;}
```

如果页面中的表格为三层嵌套，则最小的文本为6pt(12 × 0.8 × 0.8 × 0.8=6.144)。很多复杂的页面至少有三层嵌套，有的甚至还要更多。

保持加粗

这里有一个小技巧可帮助避开大多数版本的Navigator 4 中的错误。当元素设置font-weight: normal时，这个值会被该元素的所有后代继承。当然，这是应该的，但Navigator却处理得不好，给出如下标记：

```
<P STYLE-"font-weight: normal;">This is a paragraph which contains a
<B>boldface element</B>, but Navigator 4 won't make the text bold.</P>
```

这是对的：段落中的所有文本都是正常文本的粗细。因某些原因，Navigator不知

道应该为B元素指定font-weight为bold(或bolder)。类似的问题可能在使用STRONG时也会发生。

解决方法很简单，只要为这些元素显式设置font-weight。在样式表中应加入这条规则：

```
STRONG, B {font-weight: bolder;}
```

这就会解决 Navigator 4 在这方面的问题。

在 Internet Explorer 中的浮动元素

在 Windows 的 Internet Explorer 4.x 中，为了使 float 工作于文本元素，要显式声明width，类似于 width: 10em。老实说，不知道为何这样才能允许浮动。给浮动文本元素声明width确实有意义，但规范中并没有要求声明width以使文本元素成功地浮动。Internet Explorer 4.x 却必须这样做才能使浮动工作。

同时，必须使用最终版本的Explorer 4.x，也就是说，如果仍使用较早的版本，则需要升级它。(感谢 Howard Marvel 发现并共享这一技巧。)

使用与不使用：first-letter 的首字下沉

首字下沉是一个非常通用的文字效果。典型的首字下沉如图 11-22 所示。

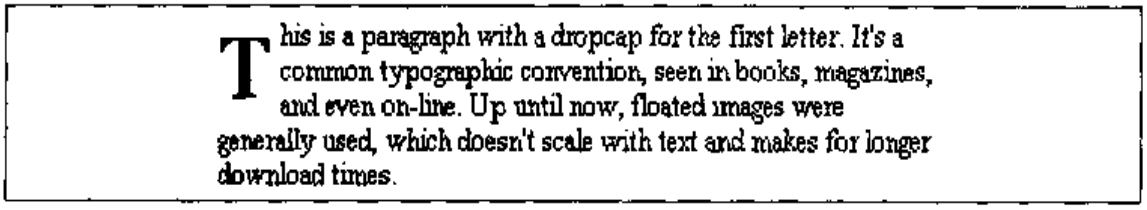


图 11-22 首字下沉

有一个简单的方法，即使用:first-letter伪元素。样式如下：

```
P.intro:first-letter {font-size: 300%; font-weight: bold; float: left;
  width: 1em;}
```

这个样式生成的效果如图 11-22 所示。

然而，较老的浏览器不支持:first-letter伪元素。Internet Explorer 3 .x 及 Navigator 4.x 都是如此。Internet Explorer 4.x 及 5.0 可以使用 SPAN元素来弥补不能支持:first-letter的缺点。如下所示:

```
SPAN.dropcap {font-size: 300%; font-weight: bold; float: left;
  width: 0.75em;}

<P><SPAN CLASS="dropcap">T</SPAN>his is a paragraph with...</P>
```

因为这与虚构的描述:first-letter行为的标记顺序类似，所以它工作得很好。尽管不够优雅，但它确实能工作。使用0.75em的width是因为大多数字母的宽度小于其高度，当然也可以使用其他值，可根据自己的视觉感受来决定。

样式消失

Navigator有一个很明显的错误，如果遇到会让人非常困惑。不论任何情况下，若触发了这个错误(主要原因是框架)，调整浏览器视窗大小会使所有样式消失，只剩下单调的文本。

重载页面会使样式恢复，但这不是令人满意的解决方法。稍好一些的方法是包含进一些JavaScript以自动修正。它的作用是使允许JavaScript的Navigator在视窗调整大小后重新应用样式。如果不支持JavaScript，则CSS 根本不工作，这是试图找出样式不工作的原因时要牢记的一点。

脚本内容如下:

```
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
var agt = navigator.userAgent.toLowerCase();
var is_major = parseInt(navigator.appVersion);
var is_nav = ((agt.indexOf('mozilla') != -1) &&
  (agt.indexOf('spoofer') == -1) &&
  (agt.indexOf('compatible') == -1));
var is_nav4 = (is_nav && (is_major == 4));

if (is_nav4) {onresize = location.reload();}
//-->
```

```
</SCRIPT>
```

这些脚本使得任何版本的Navigator 4在浏览器视窗调整大小时重载文档。

这个技巧中使用的脚本源于文章《Determining Browser Type and Version with JavaScript》，位于http://developer.netscape.com:80/docs/examples/javascript/browser_type.html处。

正确配置CSS服务

最后，有一个与CSS相关的问题。一些网页制作者在使Web主机正确提供外部样式表服务时会遇到困难。在一些Web服务器上，扩展名*.css*文件被映像为MIME类型：x-application/css或“不间断平滑显示（Continuous Slide Show）”，而不是MIME类型：text/css。较老的服务器甚至对*.css*不生成任何映像，会把文件当成text/plain来服务。

注意： 其实扩展名并非重要的部分。重要的是服务器在发送文件时使用的MIME类型。然而，由于大多数Web服务器使用文件扩展名来决定在发送文件时使用的MIME类型，因此是否有一个友好的服务器配置显然变得重要起来。

如果外部样式表使用错误的MIME类型发送，那么样式表会被破坏而不可用。如果有这个问题发生，则应联系ISP并解释该问题。如果他们拒绝修正它，则应向他们解释：IANA（Internet Assigned Numbers Authority，它也认同MIME类型）已经认可*.css*为MIME类型：text/css的扩展名，而平滑显示映像不是IANA所识别的MIME类型。

若他们仍拒绝纠正这个问题，可以通过Web空间中的配置文件自己修复它。如果Web服务器使用基于NCSA的Web服务器，例如Netscape的产品，则可以在位于Web服务器的配置文件.htaccess中增加如下行：

```
ADD Type "text/css; charset=iso-8859-1" .css
```

如果都不成功，且的确需要使用外部样式表，那就必须考虑更换ISP了。

附录一

CSS资源

在Web上，可得到许多很好的关于CSS的资源，下面我们将介绍其中的一部分。

一般性信息

这些资源提供了CSS概览。

CSS Recommendations

http://www.w3.org/TR/REC-CSS1
http://www.w3.org/TR/REC-CSS2

即使其他所有资源都不能得到，我们仍然有这个资源可以使用。规范中包含如何使用户代理一致处理CSS的完整描述，尽管有些精练且形式不总是容易辨识。其中还包含完整的CSS语法及向前兼容的语法规则，这些对写用户代理程序的人来说非常有用，对其他人则没有什么意义。

W3C CSS Activity Page

http://www.w3.org/style/css

这是CSS的在线中心。有链接指向CSS推荐规范，指向考虑中的新思路，指向其他关于CSS的站点，指向历史上的样式表单建议，指向CSS当前应用与实现的信息。还有关于CSS的书籍列表，新的CSS工具的信息及其他有用信息。

W3C CSS Test Suite

http://www.w3.org/style/css/test

提供用于测试CSS实现的相当完整的页面组。组中每个页面测试不同的CSS属性，每个页面测试一个属性。这些测试的大部分是由本书作者，Hakon Lie（Opera Software）和Tim Boland（NIST）提供，CSS协会及浏览器开发商也做出了许多贡献。若想知道浏览器处理CSS1的能力如何，这个站点当为首选。在本书写成之时，测试组只包含CSS1，但不久将会有CSS2的测试组出现。

Error Checkers

通过将自己的CSS在已有的检测器上运行，可以节省很多时间和精力。如果正考虑使用在线帮助，这是我们特别推荐的一种方法。因为如果CSS包含错误，专家告诉你的第一件事便是使用一个生效器（validator）。我们首先来实践一下。

W3C CSS Validator

http://jigsaw.w3.org/css-validator/

如果在使样式表单工作过程中有问题，可能是因为印刷错误，或一些不易忽略的基本错误。要花费很长的时间来检查样式，检测每行的规则，但其实也可以用程序来完成这件事，它会告诉你是否发现了错误，W3C CSS生效器的用途正是如此。可以通过一个样式表单或包含样式的文档的URL来应用它，也可以直接把样式粘贴于它的输入域中，让生效器辨别问题是不是因为错误拼写颜色名（或其他类似问题）所导致的。其主要缺点是对大多数人来说，报告过于学术化。除非十分熟悉HTML与CSS，否则得到的结果可能会让人困惑。

WDG CSScheck

http://www.htmlhelp.com/tools/csscheck/

同W3C生效器的功能类似，但CSScheck提供了更易读的错误信息，这对网页制作初学者非常有用。除了用怪诞的按钮指示出错误之外，CSScheck还提供了每个问题的详细信息，以及造成问题的原因。可以通过在CSScheck执行几个样式并认真阅读它的响应来学习许多如何制作好的文档的信息。

提示、指导及其他实践性建议

阅读了一般信息并检测了文档中的错误后，你可能会开始寻找一些如何使用CSS的建议，以及为何正确的CSS并不完全像想像的那样显示。

Style Sheets Reference Guide（SSRG）

http://style.webreview.com/

这个站点是浏览器性能表（Browser Compatibility Charts）的主站点。该表是CSS1中所有可能属性及值的列表，以及几个主要浏览器中的支持情况，有注解说明其比率。本书写成时，该表包含10个主要浏览器版本，且有望增加CSS2支持追踪调查。

SSRG也是*Web Review*专栏“A sense of style”所有文章的站点。主要提供CSS疑问与解答，CSS通常错误，使用样式的“酷”方法。（SSRG是由本书的作者编辑并维护的。）

CSS Pointers Group（CPG）

http://css.nu/

由Sue Sims和Jan Roland Eriksson维护。CPG收集了许多文章、错误报告及其他资源。许多指针列表简单地指向由Web上许多人提供的材料，但一些最好的文

章是他们自己写的。刚开始导航此站点可能有些困难，但这些页面上大量的信息财富使努力变得有价值。

WSP CSS Samurai Reports

http://www.webstandards.org/css/

一系列关于CSS支持可用Web浏览器上要失败细节的文档，由Web标准工程(WSP)的CSS Action Committee —— 有时被称为“CSS武士” —— 提供。本书写成时，它包含用于Windows及Macintosh的Internet Explorer，及用于Windows的Opera的报告。这些报告提供了浏览器支持中错误的内幕，以及如何测试CSS支持问题。

Agitprop

http://www.metrius.com/agitprop/

收集由Todd Fahrner所写的文章及观察，这个站点是细节信息的金矿。这里每篇文章都包含着CSS网页制作者可用的信息及应知道的内容，从《The Amazing en Unit》到《why Points Suck》，所有希望理解设计工程中交替继承及CSS设计中特殊问题的人访问该站点均可获得帮助。

在线团体

一个人只能读到这些内容，直到他参与到讨论中并提到一些问题。共有两个主要聚会地点讨论CSS，但各关于一些特定类型的讨论，因此，要确信去的是正确的地方。

comp.infosystems.www.authoring.stylesheets

这个Usenet组，经常简写为*ciwas*，是CSS制作者的聚集点。很多这个领域的专家经常会阅读该新闻组，主要的原因是：帮助新的CSS制作者克服学习障碍，并

学习生成的新语言。第二个原因是想知道对CSS某些特征或一个浏览器的实现的争论。在1990年代的后几年，这个新闻组的信噪比非常高，这对于新闻组来说非常罕见，在这种风格下，这种高的信噪比可能会幸运地持续下去。

www-style@W3.org

任何希望参与CSS未来的讨论，清除规范的随意性的人都应加入进来。这里的人都对使CSS更加出色感兴趣。请注意：*www-style* 不是寻求写CSS帮助的地方。寻求CSS制作问题的帮助，应该到 *ciwas* 去。那些以“我怎样才能够…”开始的问题不适合这里。另一方面，以“为何我不能…”或“如果这样是否更酷…”的问题才受欢迎，只要是有关CSS遗漏的功能的问题。

只有注册后，发往 *www-style* 的消息才被接收。为完成注册，应以 *subscribe* 为标题给 www-style-request@w3.org 发邮件，为取消注册，以 *unsubscribe* 为标题给 www-style-request@w3.org 发邮件。

错误报告

已经使样式可用，检测了浏览器支持情况，与专家交谈，结果证明你发现了Web浏览器中的新错误，没有人曾经提起过，你的发现可能不为他人所知。这里是如何使浏览器生产商知道它：

厂商	报告页
Microsoft	http://register.microsoft.com/contactus/contactus.asp (在 Microsoft Products menu 选择“Reporting a product bug”。)
Netscape	http://belp.netscape.com/forms/bug-client.btml
Opera	http://www.operasoftware.com/bugreport.html

附录二

HTML 2.0样式表

本附录提供的式样表摘自CSS1规范。将它们列在此的目的，是想使网页制作者了解浏览器处理HTML的行为是如何使用CSS1规则重新生成的，至少也是大概了解的。彻底地理解这个样式表是理解CSS1如何操作的第一步。重新生成了简单的HTML 2.0样式表以使其更简单，由此产生的疑惑更少。在W3C Web站点上有HTML 3.2的样式表，它是CSS2规范的一部分。

这个HTML 2.0样式表是由Todd Fahrner编写的，与HTML 2.0规范建议的规则一致：

```
BODY {
 margin: 1em;
 font-family: serif;
 line-height: 1.1;
 background: white;
 color: black;
}

H1, H2, H3, H4, H5, H6, P, UL, OL, DIR, MENU, DIV,
DT, DD, ADDRESS, BLOCKQUOTE, PRE, BR, HR { display: block }

B, STRONG, I, EM, CITE, VAR, TT, CODE, KBD, SAMP,
IMG, SPAN { display: inline }

LI { display: list-item }

H1, H2, H3, H4 { margin-top: 1em; margin-bottom: 1em }
```

```
H5, H6 { margin-top: 1em }
H1 { text-align: center }
H1, H2, H4, H6 { font-weight: bold }
H3, H5 { font-style: italic }

H1 { font-size: xx-large }
H2 { font-size: x-large }
H3 { font-size: large }

B, STRONG { font-weight: bolder } /* relative to the parent */
I, CITE, EM, VAR, ADDRESS, BLOCKQUOTE { font-style: italic }
PRE, TT, CODE, KBD, SAMP { font-family: monospace }

PRE { white-space: pre }

ADDRESS { margin-left: 3em }
BLOCKQUOTE { margin-left: 3em; margin-right: 3em }

UL, DIR { list-style: disc }
OL { list-style: decimal }
MENU { margin: 0 }      /* tight formatting */
LI { margin-left: 3em }

DT { margin-bottom: 0 }
DD { margin-top: 0; margin-left: 3em }

HR { border-top: solid }   /* 'border-bottom'could also have been used */

A:link { color: blue }    /* unvisited link */
A:visited { color: red }   /* visited links */
A:active { color: lime }   /* active links */

/* setting the anchor border around IMG elements requires contextual
selectors */

A:link IMG { border: 2px solid blue }
A:visited IMG { border: 2px solid red }
A:active IMG { border: 2px solid lime }
```

附录三

CSS1 属性

本附录列出了所有的CSS1属性，以及CSS1伪类与伪元素。属性名右方的值显示了该属性的浏览器支持信息。看起来如下：

IE4 Y/N IE5 Y/Y NN4 N/N OP3 Y/-

浏览器列表是：

NN4

Netscape Navigator 4

IE4

Internet Explorer 4 (IE 4.5 for Macintosh)

IE5

Internet Explorer 5

Op3

Opera 3.6

每对值中第一个是针对Windows版本的，第二个是针对Macintosh版本的。例如，IE4 Y/N，意味着属性被 Windows 中的 IE4 支持，但 Macintosh 中的 IE4 不支持。可能的支持值是：

Y 支持

N 不支持

P 部分支持（某些值支持，其他值不支持）

Q 奇特的支持（与规范密切相关）

B 有错误的支持（会搞乱显示，甚至使浏览器崩溃）

- 不能应用（浏览器不存在）

为得到关于浏览器支持的细节信息，参见附录四。

:active IE 4Y/Y IE5 Y/Y NN 4 N/N OP3N/-

这个伪类应用于超链接，但不用于命名的定位锚。它用于设置超链接被选择时的样式。

例 `A:active {color: red; background: yellow;}`

值 n/a

缺省值 n/a

可否继承 是

适用于 有一个 HREF 属性的定位锚元素

:first-letter IE4 N/N IE5 N/Y NN4 N/N OP3Y/-

应用样式元素的第一个字母。这个伪类可用于生成首字下沉效果。

例

```
P:first-letter {color: purple;}
    <P>The capital 'T' at the beginning of this paragraph is purple.</P>
```

值 n/a

缺省值 n/a

可否继承 是

适用于 块元素

:first-line IE4 N/N IE5 N/Y NN4 N/N OP3Y/-

应用样式于元素的第一行。样式在视窗重调大小后仍被应用，文本被重设样式仅包含元素的第一行。

例

```
P:first-line {color: red;}
    <P>The first line of this paragraph is red. blah blah blah...</P>
```

值 n/a

缺省值 n/a

可否继承 是

适用于 块元素

!important IE4 Y/N IE5 Y/Y NN4 N/N OP3Y/-

使样式声明更重要，这样可提高在层叠中的权重。重要声明优先于所有其他声明。在CSS1中，重要的制作者样式优先于所有读者样式，甚至是重要的样式。在CSS2中，反了过来，重要的读者样式总是比制作者样式优先，无论其重要性如何。

例

```
H1 {color: maroon !important;}
P.warning {color: rgb(100%,20%,20%) !important; font-weight: bold;}
```

值 n/a

缺省值 n/a

可否继承 是

适用于 样式规则

:link IE4 Y/Y IE5 Y/Y NN4 Y/Y OP3Y/-

这个伪类应用于超链接，但不应用于命名的定位锚。设样式于指向未访问过的URL的超链接上。

例

```
A:link {color: blue;}
```

值 n/a

缺省值 n/a

可否继承 是

适用于 有 HREF 属性的定位锚元素

:visited IE4 Y/Y IE5 Y/Y NN4 N/N OP3Y/-

这个伪类应用于超链接，但不应用于命名的定位锚。设样式于指向已访问过的 URL 的超链接上。

例 `A:visited {color: navy;}`

值 n/a

缺省值 n/a

可否继承 是

适用于 有 HREF 属性的定位锚元素

background IE4 P/Y IE5 Y/Y NN4 P/P OP3P/-

使用一个规则代表不同背景属性的缩略方式。建议使用这个属性而不是其他背景属性，因为这个属性支持得更广泛，且易于输入。

例

```
BODY {background: white url(bg41.gif) fixed center repeat-x;}
P {background: #555 url(http://www.pix.org/stone.png);}
PRE {background: yellow;}
```

值

< 背景颜色 > || < 背景图像 > || < 背景重复 > ||
< 背景附件 > || < 背景位置 >

缺省值 参见各个独立的属性

可否继承 否

适用于 所有元素

< 背景位置 > 允许使用百分比。

background-attachment IE4 Y/Y IE5 Y/Y NN4 N/N OP3N/-

这个属性定义了背景图片是否随元素滚动。通常只应用于BODY元素，且实际上只被该元素大力支持。理论上有可能生成“对齐”背景于使用该属性的多个元素上；在第六章“颜色和背景”中提供了更多有关的细节。

例

```
BODY {background-attachment: scroll;}
DIV.fixbg {background-attachment: fixed;}
```

值 scroll | fixed

缺省值 scroll

可否继承 否

适用于 所有元素

background-color IE4 Y/Y IE5 Y/Y NN4 B/B OP3Y/-

这个属性为元素设置背景颜色。颜色充满内容区及补白，且延伸至元素边框的外边沿。值 transparent 在 Navigator 4.x 中被解释为 black。

例

```
H4 {background-color: white;}
P {background-color: rgb(50%,50%,50%);}
PRE {background-color: #FFFF99;}
```

值 <颜色> | transparent

缺省值 transparent

可否继承 否

适用于 所有元素

background-image IE4 Y/Y IE5 Y/Y NN4 Y/Y OP3Y/-

背景模板上设置图片。取决于background-repeat，背景图片可以为平铺，或只沿一个轴，且平铺的起始位置依赖于background-position的值。

例

```
BODY {background-image: url(bg41.gif);}
H2 {background-image: url(http://www.pix.org/dots.png);}
```

值 <url> | none

缺省值 none

可否继承 否

适用于 所有元素

---

background-position IE4 Y/Y IE5 Y/Y NN4 N/N OP3Y/-

这个属性设置了背景图片（由background-image值定义）的开始位置。background-position用于设定背景平铺的起点，或是其位置（没有平铺）。百分比值不仅定义了元素内的一个位置，而且也定义初始图片同一位置，第六章提供了更多细节。

例

```
BODY {background-position: top center;}
```

值

[<百分比> | <长度>]{1,2} | [top | center | bottom] || [left | center | right]

缺省值 0% 0%

可否继承 否

适用于 块级元素和替换元素

百分比既可指相对于元素的大小，也可指相对于初始图片的大小。

---

background-repeat IE4 P/Y IE5 Y/Y NN4 P/B OP3Y/-

为背景图片设置重复样式。注意与轴相关的重复实际上沿指定轴的两个方向上重复。从初始图片开始重复背景图片，初始图片位置由background-position的值定义。

例

```
BODY {background-repeat: no-repeat;}
```

值 repeat | repeat-x | repeat-y | no-repeat

缺省值 repeat

可否继承 否

适用于 所有元素

---

border IE4 P/P IE5 P/Y NN4 P/P OP3P/-

这是定义一个元素边框的宽度、颜色及样式的缩略属性。注意尽管对值没有任何要求，但空缺掉 <border-style> 会导致没有边框被应用的结果，因为 border-style 的缺省值为 none。

例 `H1 {border: 2px dashed maroon;}`

值 <边框宽度> || <边框样式> || <颜色>

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

---

border-bottom IE4 P/P IE5 P/Y NN4 N/N OP3P/-

这个缩略属性定义了元素下边框的宽度、颜色及样式。关于 border-style 的防止误解的说明同 border。

例 `UL {border-bottom: 0.5in inset green;}`

值 <边框宽度> || <边框样式> || <颜色>

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

---

border-bottom-width IE4P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素下边框的宽度，下边框继承元素的背景，可以有自己的前景(见 border-style)。不允许使负长度值。

例 `UL {border-bottom-width: 0.5in;}`

值 thin | medium | thick | <长度>

缺省值 medium

可否继承 否

适用于 所有元素

border-color IE4 Y/Y IE5 Y/Y NN4 P/P OP3Y/-

设置元素所有边框的前景颜色（见 border-style）。

例
```
H1 {border-color: purple; border-style: solid;}
```

值 <颜色>{1,4}

缺省值 元素自己的颜色值

可否继承 否

适用于 所有元素

border-left IE4 P/P IE5 P/Y NN4 N/N OP3P/-

这个缩略属性定义了一个元素的左边框的宽度、颜色及样式。关于border-style的防止误解的说明与 border 相同。

例
```
P {border-left: 3em solid gray;}
```

值 <边框宽度> || <边框样式> || <颜色>

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

border-left-width IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素左边框的宽度，左边框继承元素的背景可以有自己的背景（见 border-style）。不允许使用负长度值。

例
```
P {border-left-width: 3em;}
```

值 thin | medium | thick | <长度>

缺省值 medium

可否继承 否

适用于 所有元素

border-right IE4 P/P IE5 P/Y NN4 N/N OP3P/-

这个缩略属性定义了元素右边框的宽度、颜色及样式。关于border-style的防止误解的说明与border相同。

例 `IMG {border-right: 30px dotted blue;}`

值 <右边框宽度> || <边框样式> || <颜色>

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

border-right-width IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素右边框的宽度，右边框继承元素的背景，可以有自己的前景(见border-style)。不允许使用负长度值。

例 `IMG {border-right-width: 30px;}`

值 thin | medium | thick | <长度>

缺省值 medium

可否继承 否

适用于 所有元素

border-style IE4 P/Y IE5 P/Y NN4 P/P OP3Y/-

设置元素所有边框的样式，使用由border-color设置的颜色。如果未定义border-color，则使用元素的前景颜色。CSS1只要求识别值none与solid。任何不能识别的值都当作solid处理。

例 `H1 {border-style: solid; border-color: purple;}`

值

none | dotted | dashed | solid | double | groove | ridge | inset | outset

缺省值 none

可否继承 否

适用于 所有元素

border-top IE4 P/P IE5 P/Y NN4 N/N OP3P/-

这个缩略属性定义了元素上边框的宽度、颜色及样式。关于border-style的防止误解的说明与border中的相同。

例 UL {border-top: 0.5in solid black;}

值 <上边框宽度> || <边框样式> || <颜色>

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

border-top-width IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素上边框的宽度，上边框继承元素的背景，也可以有自己的前景（见border-style)。不允许使用负长度值。

例 UL {border-top-width: 0.5in;}

值 thin | medium | thick | <长度>

缺省值 medium

可否继承 否

适用于 所有元素

border-width IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素所有边框的宽度，边框继承元素的背景，也可以有自己的前景（见border-style)。不允许使用负长度值。

例 `H1 {border-width: 2ex;}`

值 [thin | medium | thick | <长度>]{1,4}

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

---

clear IE4 P/Y IE5 P/Y NN4 P/P OP3B/-

定义元素的某一侧为不能旋转浮动图片。效果是向下移动元素直到边框上沿低于浮动图片的底端。

例 `H1 {clear: right;}`

值 none | left | right | both

缺省值 none

可否继承 否

适用于 所有元素

---

color IE4 Y/Y IE5 Y/Y NN4 Y/Y OP3Y/-

设置给定元素的前景色。对文本来说，它设置文本的颜色。这个值元素的所有边框继承，除非边框有由border-color设置的颜色。

例 `STRONG {color: rgb(255,128,128);}`

值 <颜色>

缺省值 与用户代理有关

可否继承 是

适用于 所有元素

---

display IE4 P/P IE5 P/Y NN4 P/P OP3P/-

用于将元素分类。通用的值是none，完全消除元素的显示。将display属性用于文档类型如HTML会造成危险，因为HTML已经定义了display。然而，在XML，display是必不可少的。

在CSS2中，display的值的范围被大大扩展。请参见第十章“CSS2展望”，可获得更多细节。

例 .hide {display: none;}

值 block | inline | list-item | none

缺省值 block

可否继承 否

适用于 所有元素

---

float IE4 P/B IE5 P/Q NN4 P/P OP3B/-

设置元素的浮动方向。通常应用于图片以使文本流动在它们的周围，在CSS中的所有元素均可浮动。注意，对段落元素而言，浮动它们会使其宽度趋向于零，除非显式指定宽度。这样，指定宽度是浮动非替换元素的关键部分。

例 IMG {float: left;}

值 left | right | none

缺省值 none

可否继承 否

适用于 所有元素

---

font IE4 P/Q IE5 P/Y NN4 P/P OP3Y/-

这是其他字体属性的缩略属性。除font-size及font-family外的其他值均可空缺，这两个是有效字体声明所要求的，注意下面错误的例子。

例

```
P {font: bold 12pt/14pt Helvetica,sans-serif;}
P.wrong {font: bold Helvetica,sans-serif;} /* missing a font-size */
P.wrong {font: 12pt Times,serif bold;}   /* font-weight must come before others */
P.wrong {font: 12pt italic Times;}   /* font-style must come before font-size */
P.fancy {font: 14pt Author;}   /* technically correct, although generic font-
families are encouraged for fallback purposes */
```

值

[<字体样式> || <字体变形> || <字体粗细>]? <字体尺寸>
[/ <行高>]? <字体系列>

缺省值 参见各个属性值

可否继承 是

适用于 所有元素

font-family IE4 Y/Y IE5 Y/Y NN4 Y/Y OP3Y/-

用于声明使用的特定字体，或一般字体族系，或两者兼有。注意使用一特定字体系列必须以用户已安装该字体为前提。鼓励使用一般字体系列，因为这会使用户代理去替换相近的字体。

例

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

值 [[<系列名> | <一般系列>],]* [<系列名> | <一般系列>]

缺省值 与用户代理有关

可否继承 是

适用于 所有元素

font-size IE4 P/Q IE5 P/Y NN4 Y/Y OP3Y/-

为字体设置大小。可定义为绝对值、相对值、长度值或百分比。负长度及负百分比禁止使用。指定font-size可能带来许多危险。这些危险在第四章“文本属性”中有描述。

例

```
H2 {font-size: 200%;}
H3 {font-size: 36pt;}
```

值

xx-small | x-small | small | medium | large | x-large | xx-large | larger | smaller | <长度> | <百分比>

缺省值 medium

可否继承 是

适用于 所有元素

百分比是指相对于上级元素的字体大小。

font-style IE4 Y/Y IE5 Y/Y NN4 P/P OP3Y/-

设置字体使用italic或oblique或normal文本。Italic文本通常是字体自带的外观，而oblique文本往往不是自带的，在这种情况下，用户代理可生成一个斜的字体外观。

例 `EM {font-style: oblique;}`

值 normal | italic | oblique

缺省值 normal

可否继承 是

适用于 所有元素

font-variant IE4 Q/Y IE5 Q/Y NN4 N/N OP3Y/-

这个属性现有两个值：small-caps与normal。small-caps可应用于所选的字体或由用户代理计算生成的字体。

例 `H3 {font-variant: small-caps;}`

值 normal | small-caps

缺省值 normal

可否继承 是

适用于 所有元素

font-weight IE4 Y/Y IE5 Y/Y NN4 P/P OP3Y/-

用于设置字体的粗细，使之变粗或变细。值400等于值normal，700等于bold。每个数字值至少与下一个低值一样粗，至少与下一高值一样细。

例 B {font-weight: 700;}

值

normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900

缺省值 normal

可否继承 是

适用于 所有元素

---

height IE4 Y/Y IE5 Y/Y NN4 N/N OP3Y/-

用于设置元素的高度。高度多用于图片上，但可以应用于任何块级元素或替换元素，尽管本书写成之时，这种行为仍不被广泛采用，允许使用负长度值。

例 IMG.icon {height: 50px;}

值 <长度> | auto

缺省值 auto

可否继承 否

适用于 块级元素和替换元素

---

letter-spacing IE4 Y/Y IE5 Y/Y NN4 N/N OP3Y/-

用于在字母间设置空白。字母定义为任何可以显示的字符，包括数字、符号等。长度值用于定义相对于通常空白的改变，而不是空白自身的长度。这样，normal与0同义，允许负值，使字母距离缩小。

例 P {letter-spacing: 0.5em;}

值 normal | <长度>

缺省值 normal

可否继承 是

适用于 所有元素

line-height IE4 P/P IE5 D/Y NN4 P/P OP3Q/-

这将影响行框的布局。line-height 与 font-size 之间的差值叫做铅条(leading)，铅条的一半分别位于元素内容或行文本的上方与下方。不允许使用负值。使用数字定义缩放因子，缩放因子与字体大小相乘，被继承的是缩放因子，而不是计算得出的值。这允许更加智能化的页面，且优于其他设置 line-height 的方法。

使用数字值的缺点是 IE3 将把数字解释成像素数。请参见第八章中的详细讨论。

例

```
P {line-height: 18pt;}
H2 {line-height: 200%;}
```

值 normal | <数字> | <长度> | <百分比>

缺省值 normal

可否继承 是

适用于 所有元素

百分比是指相对于元素自身的文本大小。

list-style IE4 P/P IE5 Y/Y NN4 P/P OP3Y/-

所有其他 list-style 属性的缩略属性。应用于所有显示值为 list-item 的元素。在普通 HTML 中，是所有 <LI> 元素。

例

```
UL {list-style: square url(bullet3.gif) outer;}  /* 值可由LI元素继承 */
```

值

disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none] || [inside | outside] || [<url> | none]

缺省值 未定义缩略属性

可否继承 是

适用于 显示值为list-item的元素

list-style-image IE4 Y/Y IE5 Y/Y NN4 N/N OP3Y/-

用于声明用作有序或无序列表项目符号的图片。这个样式应用于display值为list-item的元素。图片相对于列表项的位置用list-style-position定义。

例
```
UL {list-style-image: url(bullet3.gif);}
```

值 <url> | none

缺省值 none

可否继承 是

适用于 显示值为list-item的元素

list-style-position IE4 Y/Y IE5 Y/Y NN4 N/N Op3 Y/-

这个属性用于声明有序或无序列表中项目符号或数字相对于列表项内容的位置。应用于display值为list-item的元素。如果项目符号设为outside，则放置于列表项元素的边界。CSS中未定义这种情况下的精确行为。

例
```
LI {list-style-position: outer;}
```

值 inside | outside

缺省值 outside

可否继承 是

适用于 显示值为list-item的元素

list-style-type IE4 Y/Y IE5 Y/Y NN4 Y/P OP3Y/-

用于声明有序或无序列表中项目符号计数系统的类型，依赖于特定的值。这个属性应用于display值为list-item的元素。

例
```
UL {list-style-type: square;}
OL {list-style-type: lower-roman;}
```

值

disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none

缺省值 disc

可否继承 是

适用于 显示值为 list-item 的元素

margin IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素所有边界的大小。块级元素的垂直相邻边界压缩为最大边界的大小，而内联元素不存在垂直方向上的边界（允许，但无页面布局效果）。内联元素只有左边界和右边界有效果，且不压缩。负值是允许的，但要谨慎使用。

例 H1 {margin: 2ex;}

值 [<长度> | <百分比> | auto]{1,4}

缺省值 未定义缩略属性

可否继承 否

适用于 所有元素

百分比是指相对于最近上级元素的宽度。

margin-bottom IE4 P/P IE5 P/Y NN4 N/N OP3Y/-

设置元素下边界的大小。负值是允许的，但要谨慎使用。

例 UL {margin-bottom: 0.5in;}

值 <长度> | <百分比> | auto

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

margin-left IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素右边界的大小。允许负值，但要谨慎使用。

例 `P {margin-left: 3em;}`

值 <长度> | <百分比> | auto

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

margin-right IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素右边界的大小。允许负值，但要谨慎使用。

例 `IMG {margin-right: 30px;}`

值 <长度> | <百分比> | auto

缺省值 0

可否继承 否

适用于 所有元素

margin-top IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

设置元素上边界的大小。允许负值，但要谨慎使用。

例 `UL {margin-top: 0.5in;}`

值 <长度> | <百分比> | auto

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

padding IE4 P/P IE5 P/Y NN4 B/B OP3B/-

设置元素所有补白的大小。补白“继承”元素的背景；换句话说，元素背景填充其内容区及补白。给行内元素设补白不影响行高的计算，但会应用于元素的左右两端。若行内元素既有补白又有背景，背景可能会在行内元素出现的地方超出行框的上下边沿，但不要求用户代理支持这种行为。也没有定义上一行的前景内容应该位于这一行的元素背景之上，还是被背景覆盖。不允许负值。

例 `H1 {padding: 2ex;}`

值 [<长度> | <百分比>]{1,4}

缺省值 未定义缩略值

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

padding-bottom IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

这个属性设置元素的下补白大小，下补白“继承”元素的背景。不允许负值。

例 `UL {padding-bottom: 0.5in;}`

值 <长度> | <百分比>

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

padding-left IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

这个属性设置元素左补白的大小，左补白“继承”元素的背景，不允许使用负值。

例 `P {padding-left: 3em;}`

值 <长度>|<百分比>

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

padding-right IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

这个属性设置元素右补白的大小，右补白"继承"元素的背景，不允许使用负值。

例 `IMG {padding-right: 30px;}`

值 <长度>|<百分比>

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

padding-top IE4 P/P IE5 P/Y NN4 B/B OP3Y/-

这个属性设置元素上补白的大小，上补白"继承"元素的背景。不允许使用负值。

例 `UL {padding-top: 0.5in;}`

值 <长度>|<百分比>

缺省值 0

可否继承 否

适用于 所有元素

百分比是指相对于最近块级上级元素的宽度。

text-align IE4 Y/P IE5 Y/Y NN4 Y/P OP3Y/-

设置元素中文本的水平对齐，或更精确地，定义行框与元素的哪一侧对齐，值justify支持程序调整行中内容的字母间距与字间距，且结果可随用户代理变化。

例

```
P {text-align: justify;}
H4 {text-align: center;}
```

值 left | right | center | justify

缺省值 与用户代理有关

可否继承 是

适用于 块元素

text-decoration IE4 P/P IE5 P/P NN4 Q/Q OP3P/-

这个属性给文本设置一定的效果，比如underline或blink。这些装饰会“跨越”未定义文本装饰的下级元素。第四章提供了更多细节。联合使用值是合法的。

不要求用户代理支持blink，实际上只有Netscape Navigator 4.x是这样。

例

```
U {text-decoration: underline;}
.old {text-decoration: line-through;}
U.old {text-decoration: line-through underline;}
```

值 none | [underline || overline || line-through || blink]

缺省值 none

可否继承 否

适用于 所有元素

text-indent IE4 Y/Y IE5 Y/Y NN4 Y/Y OP3Y/-

用于设置元素的首行缩进。常用于生成tab效果且允许负值，负值导致“悬挂缩进”。

例

```
P {text-indent: 5em;}
```

```
H2 {text-indent: -25px;}
```

值 <长度>|<百分比>

缺省值 0

可否继承 是

适用于 块元素

百分比是指相对于上级元素的宽度。

text-transform IE4 Y/Y IE5 Y/Y NN4 Y/Y OP3P/-

这个属性改变元素中字母的状况，而不管初始文本是什么样。值capitalize将字母改为大写并不是精确定义的行为，依赖于对“字”的选择，而这从编程角度来说很难定义。

例

```
H1 {text-transform: uppercase;}
.title {text-transform: capitalize;}
```

值 capitalize | uppercase | lowercase | none

缺省值 none

可否继承 是

适用于 所有元素

vertical-align IE4 P/P IE5 P/Y NN4 N/N OP3P/-

根据行高设置元素基线的垂直对齐，允许负百分比，且会使元素下沉，而不是升高。

例

```
SUP {vertical-align: super;}
.fnote {vertical-align: 50%;}
```

值

baseline | sub | super | top | text-top | middle | bottom | text-bottom | <百分比>

缺省值 baseline

可否继承 否

适用于 内联元素

百分比是指相对于元素自身的行高。

---

white-space IE4 N/N IE5 N/Y NN4 P/P OP3N/-

这个属性定义如何处理元素内的空白。normal 的做法类似于传统 Web 浏览器，将空白序列缩减为一个空白。pre使空白的处理与HTML中的做法相同，空白及回车被保留。nowrap防止元素被行中断。

例

```
TD {white-space: nowrap;}
TT {white-space: pre;}
```

值 normal | pre | nowrap

缺省值 normal

可否继承 是

适用于 块元素

---

width IE4 P/Y IE5 P/Y NN4 P/P OP3Q/-

用于定义元素的宽度。多应用在图片上，但也可应用于任何块级元素及替换元素。禁止使用负值。

例 `TABLE {width: 80%;}`

值 <长度> | <百分比> | auto

缺省值 auto

可否继承 否

适用于 块级元素和替换元素

百分比是指相对于上级元素的宽度。

word-spacing IE4 N/Y IE5 N/Y NN4 N/N OP3Y/-

用于设置字间空白的数量。“字”定义为空白间的字符串。长度值用于定义相对于常用空白的变化，而不是空白本身，这样，normal与0等价。允许使用负值，使字间距离变小。

例 `P {word-spacing: 0.5em;}`

值 normal | <长度>

缺省值 normal

可否继承 是

适用于 所有元素

附录四

CSS支持表

可以看出，浏览器兼容性是接受CSS的最大障碍，本附录提供了浏览器对CSS1支持实现的综合向导。仔细阅读本表，可以观察一个属性及它的值被支持的程度如何。

附录使用以下关键字:

Y Yes（支持）

N No（不支持）

P Partial（部分支持）

B Buggy（有错误的支持）

Q Quirky（奇特的支持）

“Buggy”代表任何错误，从显示混乱到使浏览器崩溃。“Quirky”代表虽然浏览器技术上与规范兼容，但其行为会与网页制作者的期望不同。

这个列表及注释是2000年1月的状况，要知道更新的信息，可访问http://style.webreview.com/（译注1）。

译注1：本书的中文版已经采用了付印前该列表的最新版本。

规范章节	属性或值	Windows 95/98/NT									Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
1.1	Containment in HTML	P	Y	P	Q	Q	Q	Y	Y	Y	P	Y	B	Y	Y
	LINK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y
	<STYLE>...</STYLE>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	@import	N	Y	N	Q	Q	Q	Y	Y	Y	N	Y	N	Y	Y
	<x STYLE="Dec;">	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y	Y	Y
1.2	Grouping	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	x, y, z {dec;}	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
1.3	Inheritance	B	Y	P	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
	(被继承的值)	B	Y	P	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
1.4	Class selector	Y	Y	B	Q	Q	Q	Y	Y	Y	Y	Y	B	Y	Y
	.class	Y	Y	B	Q	Q	Q	Y	Y	Y	Y	Y	B	Y	Y
1.5	ID selector	B	Y	B	B	B	B	Y	Y	Y	B	Y	B	B	Y
	#ID	B	Y	B	B	B	B	B	Y	Y	B	Y	B	B	Y
1.6	Contextual selectors	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	P	Y	Y
	x y z{dec;}	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	P	Y	Y
1.7	Comments	Y	Y	B	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	/* comment */	Y	Y	B	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2.1	anchor	P	Y	N	Y	Y	Y	P	P	Y	P	Y	B	Y	Y
	A:link	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y
	A:active	N	Y	N	Y	Y	Y	N	N	Y	N	Y	N	Y	Y
	A:visited	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	B	Y	Y
2.3	first-line	N	Y	N	N	N	Y	Y	Y	Y	N	Y	B	N	Y
	:first-line	N	Y	N	N	N	Y	Y	Y	Y	N	Y	B	N	Y
2.4	first-letter	N	Y	N	N	N	Y	Y	Y	Y	N	Y	B	N	Y
	:first-letter	N	Y	N	N	N	Y	Y	Y	Y	N	Y	B	N	Y
3.1	important	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y
	!imporant	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y
3.2	Cascading Order	B	Y	P	Y	Y	Y	Y	Y	Y	B	Y	P	Y	Y
	Weight sorting	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y	Y	Y
	Origin sorting	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
	Specificity sorting	B	Y	P	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
	Order sorting	B	Y	N	Y	Y	Y	Y	Y	Y	B	Y	N	Y	Y

规范章节	属性或值	Windows 95/98/NT									Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
5.2.2	font-family	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
	<family-name>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
	<generic-family>	P	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
	... serif	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	... sans-serif	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	... cursive	N	Y	B	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	... fantasy	N	Y	B	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	... monospace	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5.2.3	font-style	P	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	Y	Y
	normal	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	italic	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	oblique	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
5.2.4	font-variant	N	Y	N	Q	Q	Q	Y	Y	Y	N	Y	N	Q	Y
	normal	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	small-caps	N	Y	N	Q	Q	Q	Y	Y	Y	N	Y	N	Q	Y
5.2.5	font-weight	P	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	Y	Y
	normal	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	bold	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	bolder	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	lighter	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	100-900	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.2.6	font-size	Y	Y	P	P	P	P	Y	Y	Y	Y	Y	P	Q	Y
	<absolute-size>	Y	Y	Y	Q	Q	Q	Y	Y	Y	Y	Y	B	Q	Y
	... xx-small - xx-large	Y	Y	Y	Q	Q	Q	Y	Y	Q	Y	Y	B	Q	Y
	<relative-size>	Y	Y	Y	Y	Y	B	Y	Y	Y	Y	Y	N	Y	Y
	... larger	Y	Y	Y	Y	Y	B	Y	Y	Y	Y	Y	N	Y	Y
	... smaller	Y	Y	Y	Y	Y	B	Y	Y	Y	Y	Y	N	Y	Y
	<length>	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y
	<percentage>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
5.2.7	font	P	Y	P	P	P	Y	Y	Y	Y	P	Y	P	Q	Y
	<font-family>	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y
	<font-style>	P	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
	<font-variant>	N	Y	N	Q	Q	Q	Y	Y	Y	N	Y	N	Q	Y
	<font-weight>	P	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	<font-size>	Y	Y	B	Q	Q	P	Y	Y	Y	Y	Y	B	Y	Y
	<line-height>	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
5.3.1	color	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	<color>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5.3.2	background-color	B	Y	P	Y	Y	Y	Y	Y	Y	B	Y	N	Y	Y
	<color>	B	Y	B	Y	Y	Y	Y	Y	Y	B	Y	N	Y	Y
	transparent	B	Y	N	Y	Y	Y	Y	B	Y	B	Y	N	Y	Y
5.3.3	background-image	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	<url>	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	none	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.3.4	background-repeat	P	Y	N	P	Y	Y	Y	Y	Y	B	Y	N	Y	Y
	repeat	Y	Y	N	B	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	repeat-x	P	Y	N	B	Y	Y	Y	Y	Y	P	Y	N	Y	Y
	repeat-y	P	Y	N	B	Y	Y	Y	Y	Y	P	Y	N	Y	Y
	no-repeat	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.3.5	background-attachment	N	Y	N	Y	Y	Y	N	Y	Y	N	Y	N	Y	Y
	scroll	N	Y	N	Y	Y	Y	N	Y	Y	N	Y	N	Y	Y
	fixed	N	Y	N	Y	Y	Y	N	Y	Y	N	Y	N	Y	Y
5.3.6	background-position	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	<percentage>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	<length>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	top	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	center	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	bottom	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	left	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	right	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
5.3.7	Background	P	Y	P	P	Y	Y	P	P	Y	P	Y	P	Y	Y
	<background-color>	B	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	Y	Y
	<background-image>	P	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y
	<background-repeat>	P	Y	B	B	Y	Y	Y	Y	Y	P	Y	B	Y	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
	<background-attachment>	N	Y	N	Y	Y	Y	N	Y	Y	N	Y	Y	Y	Y
	<background-position>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	P	Y	Y
5.4.1	word-spacing	N	Y	N	N	N	N	Y	Y	Y	N	Y	N	Y	Y
	normal	N	Y	N	N	N	N	Y	Y	Y	N	Y	N	Y	Y
	<length>	N	Y	N	N	N	N	Y	Y	Y	N	Y	N	Y	Y
5.4.2	letter-spacing	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	normal	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	<length>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
5.4.3	text-decoration	B	B	B	B	B	B	B	B	B	B	B	B	B	B
	none	Q	Y	N	Y	Y	Y	Y	Y	Y	Q	Y	Y	Q	Y
	underline	Q	Y	B	Y	Y	Y	Y	Y	Y	Q	Y	B	Q	Y
	overline	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	line-through	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	blink	Y	Y	N	N	N	N	N	Y	Y	Y	Y	N	N	N
5.4.4	vertical-align	N	Y	N	P	P	P	P	Y	Y	N	Y	N	P	Y
	baseline	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	sub	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	super	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	top	N	Y	N	N	N	Y	B	Y	Y	N	Y	N	Y	Y
	text-top	N	Y	N	N	N	Y	N	Y	Y	N	Y	N	Y	Y
	middle	N	Y	N	B	N	Y	B	Y	Y	N	Y	N	Y	Y
	bottom	N	Y	N	N	N	Y	B	Y	Y	N	Y	N	B	Y
	text-bottom	N	Y	N	N	N	Y	N	Y	Y	N	Y	N	B	Y
	<percentage>	N	Y	N	N	N	N	Y	Y	Y	N	Y	N	B	Y
5.4.5	text-transform	Y	Y	N	Y	Y	Y	P	Y	Y	Y	Y	N	Y	Y
	capitalize	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	uppercase	Y	Y	N	Y	Y	Y	B	Y	Y	Y	Y	N	Y	Y
	lowercase	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
	none	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
5.4.6	text-align	Y	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	P	Y
	left	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	right	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
	center	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	justify	B	Y	N	Y	Y	Y	Y	Y	Y	B	Y	N	N	Y
5.4.7	text-indent	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	<length>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	<percentage>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5.4.8	line-height	P	Y	P	Y	Y	Y	Q	Y	Y	P	Y	P	Y	Y
	normal	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	<number>	P	Y	N	Y	Y	Y	Y	Y	Y	P	Y	B	Y	Y
	<length>	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
	<percentage>	P	Y	Y	Y	Y	Y	Y	Y	Y	P	Y	B	Y	Y
5.5.01	margin-top	P	Y	B	P	P	Y	Y	Y	Y	P	Y	B	P	Y
	<length>	P	Y	B	P	P	Y	Y	Y	Y	P	Y	B	P	Y
	<percentage>	P	Y	Y	P	P	Y	Y	Y	Y	P	Y	B	P	Y
	auto	P	Y	Y	P	P	Y	Y	Y	Y	P	Y	B	P	Y
5.5.02	margin-right	B	B	P	P	P	Q	Y	B	Y	B	Y	P	P	Y
	<length>	B	Y	Y	P	P	Y	Y	Y	Y	B	Y	Y	P	Y
	<percentage>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	Y	P	Y
	auto	N	Y	N	N	N	Y	Y	Y	Y	N	Y	N	P	Y
5.5.03	margin-bottom	N	Y	Y	P	P	Y	Y	Y	Y	N	Y	N	P	Y
	<length>	N	Y	N	P	P	Y	Y	Y	Y	N	Y	N	P	Y
	<percentage>	N	Y	N	P	P	Y	Y	Y	Y	N	Y	N	P	Y
	auto	N	Y	N	P	P	Y	Y	Y	Y	N	Y	N	P	Y
5.5.04	margin-left	B	Y	P	P	P	Y	Y	B	Y	B	Y	P	P	Y
	<length>	B	Y	Y	P	P	Y	Y	Y	Y	Y	Y	Y	P	Y
	<percentage>	B	Y	Y	P	P	Y	Y	Y	Y	B	Y	Y	P	Y
	auto	N	Y	N	N	N	Y	Y	Y	Y	B	Y	N	P	Y
5.5.05	margin	B	B	B	P	P	Y	Y	B	Y	B	B	B	P	Y
	<length>	B	Y	B	P	P	Y	Y	Y	Y	B	Y	B	P	Y
	<percentage>	B	Y	Y	P	P	Y	Y	Y	Y	B	Y	B	P	Y
	auto	N	Y	Y	P	P	Y	Y	Y	Y	N	Y	B	P	Y
5.5.06	padding-top	B	Y	N	P	P	Q	Y	Y	Y	B	Y	N	P	Y
	<length>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<percentage>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
5.5.07	padding-right	B	Y	N	P	P	Q	Y	Y	Y	B	Y	N	P	Y
	<length>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<percentage>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
5.5.08	padding-bottom	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<length>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<percentage>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
5.5.09	padding-left	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<length>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<percentage>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
5.5.10	padding	B	Y	N	P	P	Q	B	Y	Y	B	Y	N	P	Y
	<length>	B	Y	N	P	P	Y	B	Y	Y	B	Y	N	P	Y
	<percentage>	B	Y	N	P	P	Y	B	Y	Y	B	Y	N	P	Y
5.5.11	border-top-width	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	thin	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	medium	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	thick	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	<length>	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
5.5.12	border-right-width	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	thin	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	medium	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	thick	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	<length>	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
5.5.13	border-bottom-width	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	thin	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	medium	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	thick	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	<length>	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
5.5.14	border-left-width	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	thin	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	medium	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	thick	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	<length>	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
5.5.15	border-width	B	Y	N	P	P	Y	Y	Y	Y	B	Y	N	P	Y
	thin	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	medium	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	thick	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
	<length>	Y	Y	N	P	P	Y	Y	Y	Y	Y	Y	N	P	Y
5.5.16	border-color	P	Y	N	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y
	<color>	P	Y	N	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y
5.5.17	border-style	P	Y	N	P	P	Y	Y	Y	Y	P	Y	N	Y	Y
	none	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	dotted	N	Y	N	N	N	Y	Y	Y	Y	N	Y	N	Y	Y
	dashed	N	Y	N	N	N	Y	Y	Y	Y	N	Y	N	Y	Y
	solid	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	double	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	groove	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	ridge	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	inset	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	outset	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.5.18	border-top	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-top-width>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-style>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<color>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
5.5.19	border-right	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-right-width>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-style>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<color>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
5.5.20	border-bottom	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-bottom-width>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-style>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<color>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
5.5.21	border-left	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-left-width>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<border-style>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y
	<color>	N	Y	N	P	P	Y	P	Y	Y	N	Y	N	P	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
5.5.22	border	P	Y	N	P	P	Y	P	Y	B	P	Y	N	P	Y
	<border-width>	B	Y	N	P	P	Y	P	Y	Y	B	Y	N	P	Y
	<border-style>	P	Y	N	P	P	Y	P	Y	Y	P	Y	N	P	Y
	<color>	Y	Y	N	P	P	Y	P	Y	Y	Y	Y	N	P	Y
5.5.23	width	P	Y	N	P	P	Y	Q	Y	Y	P	Y	N	Y	Y
	<length>	P	Y	N	P	P	Y	Q	Y	Y	P	Y	N	Y	Y
	<percentage>	P	Y	N	P	P	Y	Q	Y	Y	P	Y	N	Y	Y
	auto	P	Y	N	P	P	Y	Q	Y	Y	P	Y	N	Y	Y
5.5.24	height	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	<length>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	auto	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
5.5.25	float	P	Y	N	P	P	Q	B	Q	Y	P	Y	N	B	Q
	left	B	Y	N	B	B	Y	Y	Y	Y	B	Y	N	Y	Y
	right	B	Y	N	B	B	Y	Y	Y	Y	B	Y	N	Y	Y
	none	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.5.26	clear	P	Y	N	P	P	Y	B	Y	Y	P	Y	N	Y	Y
	none	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	left	B	Y	N	B	B	Y	N	Y	Y	B	Y	N	Y	Y
	right	B	Y	N	B	B	Y	Y	Y	Y	B	Y	N	Y	Y
	both	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.6.1	display	P	Y	N	P	P	P	P	Y	Y	P	Y	N	P	Y
	block	B	Y	N	N	Y	Y	Y	Y	Y	B	Y	N	P	Y
	inline	N	Y	N	N	Y	Y	B	Y	Y	N	Y	N	N	Y
	list-item	B	Y	N	N	N	N	N	Y	Y	P	Y	N	P	Y
	none	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
5.6.2	white-space	P	Y	N	N	N	P	N	Y	Y	P	Y	N	N	Y
	normal	Y	Y	N	N	N	Y	N	Y	Y	Y	Y	N	N	Y
	pre	Y	Y	N	N	N	N	N	Y	Y	Y	Y	N	N	Y
	nowrap	N	Y	N	N	N	Y	N	Y	Y	N	Y	N	N	Y
5.6.3	list-style-type	Y	Y	N	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y
	disc	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	circle	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	square	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y

规范章节	属性或值	Windows 95/98/NT								Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
	decimal	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	lower-roman	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	upper-roman	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	lower-alpha	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	upper-alpha	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	none	Y	Y	N	Y	Y	Y	Y	Y	Y	B	Y	N	Y	Y
5.6.4	list-style-image	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	<url>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	none	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
5.6.5	list-style-position	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
	inside	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Q	Y
	outside	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
5.6.6	list-style	P	Y	N	P	Y	Y	Y	Y	Y	P	Y	N	P	Y
	<keyword>	Y	Y	N	Y	Y	Y	Y	Y	Y	P	Y	N	Y	Y
	<position>	N	Y	N	Q	Q	Y	Y	Y	Y	N	Y	N	Q	Y
	<url>	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
6.1	长度单位	P	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y
	em	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	ex	Q	Y	N	Q	Q	Y	Q	Y	Y	Q	Y	Q	Q	Y
	px	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	in	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	cm	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	mm	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	pt	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	pc	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6.2	百分比单位	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	<percentage>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6.3	颜色单位	P	Y	P	Y	Y	Y	Y	Y	Y	P	Y	P	Y	Y
	#000	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y
	#000000	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y
	(RRR,GGG,BBB)	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	(R%,G%,B%)	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
	<keyword>	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	Y	Y	Y

规范章节	属性或值	Windows 95/98/NT									Macintosh				
		NN4	NN6	IE3	IE4	IE5	IE5.5	Opr3	Opr4	Opr5	NN4	NN6	IE3	IE4	IE5
6.4	URL	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y
	<url>	B	Y	Y	Y	Y	Y	Y	Y	Y	B	Y	B	Y	Y

注释

1.1 Containment in HTML: `@import`

WinIE4与WinIE5均可导入文件，甚至在`@import`声明位于文档样式表单结尾时也可行。这从技术上违背了规范，尽管不是主要的失败原因。因此列为“Quirk”。

1.1 Containment in HTML: `<x STYLE= " dec ;">`

Navigator 4 在处理列表项上有问题，这是列为“Buggy”的主要原因。

1.3 Inheritance

Navigator 4 的继承最好的情况是不稳定，最坏的情况是致命的缺陷。列出所有的可能会占用太多空间，但一般较易出问题的是表格与列表。

1.4 Class selector

WinIE4/5 允许类名以数字开头，这在CSS1中是不允许的。在CSS2中允许。

1.5 ID selector

WinIE4/5 允许ID名以数字开头，这在CSS1中是不允许的。所有浏览器都将一指定ID应用于HTML文档中该ID的多个实例，这是不允许的。这是一个错误检测问题，不是CSS规范的失败之处，但足可以将它列入warrant。

1.6 Contextual selectors: `x y z {dec;}`

MacNav4很多关于上下文选择程序的问题都包括表格。例如，`HTML BODY TABLE P`就不能正确地处理。

2.3 `:first-line`

IE3 错误地将样式应用于整个元素。

2.4 `:first-letter`

IE3 错误地将样式应用于整个元素。

3.2 Cascading Order

有太多的问题实例，无法一一列出。

5.2.2 font-family: cursive

尽管优先设置草体字体，Opera不会去应用它，而是用其他字体代替。随系统的不同而不同，其他浏览器未报告这类问题。

5.2.4 font-variant: small-caps

IE4/5通过使所有选中文本大写来近似small-caps样式。尽管从视觉上来说CSS1规范规定的差不多，但它并没有用small-caps的方式来绘制文本。

5.2.6 font-size: xx-small through xx-large

IE4/5中指定small与未定义样式文本的绝对值大小相同，而不是medium。这样，声明绝对字体大小（如font-size: medium）将会在Navigator与Internet中生成不同的字体大小。而这在规范中不是错误，使很多网页制作者感到疑惑。

5.3.2 background-color

Nav4坚持将这个值应用于元素的背景，而不是元素本身。这会导致上级元素背景中有“洞”。

5.3.2 background-color: <color>

Nav4不会将背景颜色应用于整个内容框与补白，而只应用于元素的文本。这可以通过声明一个宽度接近0的边框来解决。

5.3.4 background-repeat: repeat

WinIE4只向下及向右重复。正确的行为是对repeat-y，背景图片沿垂直方向向上、向下铺开，对repeat-x，沿水平方向向左、向右铺开。Nav4在技术上正确应用该属性：由于不支持background-position。无法知道是否可以沿四个方向铺开，或是模仿WinIE4的行为。Opera3.6，MacIE4.5及winIE5均正确。

5.3.4 background-repeat: repeat-x

WinIE4只向右重复，而不是沿两个方向。

5.3.4 background-repeat: repeat-y

WinIE4只向下重复，而不是沿两个方向。

5.3.7 background

Navigator 4.x对于显示输出背景有些力不从心。如果元素周围没有边框，那么只有元素文本后面的背景可见，而不是整个内容区和补白的背景都可见。不幸的是，如果再加上一个边框，那么，在内容区和边框之间将会有一个透明的缝隙。这不是补白，而且没办法消除这个缝隙。

5.4.3 text-decoration: none

根据规范，如果元素有装饰，但它的一个下级没有，则上级元素的效果在下级元素仍可见；从某种意义上来说，是装饰“透射”。这样，如果段落加下划线，但其中一个 STRONG 元素设置为无下划线，段落的下划线会“跨越”STRONG 元素。这也意味下级元素的下划线应与其上级元素的下划线颜色相同，除非下级元素也设置为有下划线。实际中，将内联元素设置为 none 会清除所有的装饰，无论上级单元有怎样的装饰。唯一例外的是 Opera，它能正确地实现规范。

5.4.3 text-decoration: blink

由于这个属性值在 CSS1 中未被要求，只有 Navigator 支持它。

5.4.5 text-transform: uppercase

Opera 3.6 将字中每个内联元素的第一个字母变为大写，这是不应该的。

5.4.6 text-align: justify

在 Nav4 中，这个值可能会在表格中产生故障，其他情况下正常。

5.4.8 line-height: <length>

Nav4 错误地允许这个属性使用负值。

5.4.8 line-height

Opera 3.6在背景设置于文本中的内联元素时，应用背景颜色于行间空白，而不是应用在文本上。

5.5.01 margin-top

所有边界属性用在内联元素时，要么易导致问题，要么完全不被支持。对 margin-top 来说，在 Nav4 及 IE4/5 中用在块级元素上时能支持得很好，在 Opera 3.6 中也较完美。

5.5.02 margin-right

所有的边界属性应用于内联元素时，要么易导致问题，要么完全不被支持。对margin-right而言，IE4及IE5会中在块级元素上有很好的支持，而对内联元素，IE4和IE5完全忽略这个属性。只要不把它应用在浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误的发生。

5.5.03 margin-bottom

所有边界属性应用于内联元素时，要么易导致问题，要么完全不被支持。对margin-bottom而言，IE4及IE5中在块级元素上有很好的支持，而对内联元素，IE4和IE5会完全忽略这个属性。只要不把它用在浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误发生。

5.5.04 margin-left

所有边界属性应用于内联元素时，要么易导致问题，要么完全不被支持。对margin-left而言，IE4及IE5中在块级元素上有很好的支持，而对内联元素，IE4和IE5会完全忽略这个属性。只要不把它应用在浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误的发生。

5.5.05 margin

所有的边界属性应用于内联元素时，要么易导致问题，要么完全不被支持。对margin来说，IE4和IE5中用在块级元素上时有很好的支持，而对内联元素，IE4和IE5会完全忽略这个属性。只要不应用于浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误发生。

5.5.06 padding-top

所有补白属性应用于内联元素时，要么易导致问题，要么完全不被支持。对于padding-top来说，IE4和IE5中用在块级元素上时有很好的支持。只要不应用于浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误。

5.5.07 padding-right

所有补白属性应用于内联元素时，要么易导致问题，要么完全不被支持。对于padding-right来说，IE4和IE5中用在块级元素上时有很好的支持。只要不应用于浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误。

5.5.08 padding-bottom

所有补白属性应用于内联元素时，要么易导致问题，要么完全不被支持。对于padding-bottom来说，IE4和IE5中用在块级元素上时有很好的支持。只要不应用于浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误。

5.5.09 padding-left

所有补白属性应用于内联元素时，要么易导致问题，要么完全不被支持。对于padding-left来说，IE4和IE5中用在块级元素上时有很好的支持。只要不应用于浮动元素或内联元素，Navigator可以很好地工作，否则会触发一些错误。

5.5.10 padding

所有补白属性应用于内联元素时，要么易导致问题，要么完全不被支持。Opera正确地忽略负补白值（尽管Opera 3.5应用负补白），但会根据应用于内联元素的补白的值改变行高，这是错误的。只要不应用于浮动元素或内联元素，Navigator可以很好地工作，否则会发生错误。

5.5.11 border-top-width

Navigator即使在未设置border-style的情况下仍会生成可见边框，且当设置了样式时并不在所有边设置边框。当边框应用于内联元素时情况变得更糟。IE4和IE5可正确处理块级元素的边框，但忽略内联元素的框。

5.5.12 border-right-width

Navigator即使在未设置border-style的情况下仍会生成可见边框，且当设置了样式时并不在所有边设置边框。当边框应用于内联元素时情况变得更糟。IE4和IE5可正确处理块级元素的边框，但忽略内联元素的边框。

5.5.13 border-bottom-width

Navigator即使在未设置border-style的情况下仍会生成可见边框，且当设置了样式时并不在所有边设置边框。当边框应用于内联元素时情况变得更糟。IE4和IE5可正确处理块级元素的边框，但忽略内联元素的边框。

5.5.14 border-left-width

Navigator即使在未设置border-style的情况下仍会生成可见边框，且当设

置了样式时并不在所有边设置边框。当边框应用于内联元素时情况变得更糟。IE4和IE5可正确处理块级元素的边框，但忽略内联元素的边框。

IE4和IE5正确处理块级元素的边框，但忽略内联元素的边框。

5.5.15 border-width

Navigator即使在未设置border-style的情况下仍会生成可见边框，且当设置了样式时并不在所有边设置边框。当边框应用于内联元素时情况变得更糟。IE4和IE5可正确处理块级元素的边框，但忽略内联元素的边框。

5.5.16 border-color

Nav4和Opera不对单独的边设置颜色，像border-color:red blue green purple;。Explorer不能对内联元素应用边框颜色，因为它不对内联元素应用边框。

5.5.17 border-style

如果border-style为none，那么，Navigator 4.x不能重置border-width为0，而是将其设置为不正确的值。

5.5.18 border-top

IE4和IE5不对内联元素应用边框。

5.5.19 order-right

IE4和IE5不对内联元素应用边框。

5.5.20 border-bottom

IE4和IE5不对内联元素应用边框。

5.5.21 order-left

IE4和IE5不对内联元素应用边框。

5.5.22 border

IE4和IE5不对内联元素应用边框。

5.5.23 width

Navigator用不同的方式应用width，但多用于简单的文本元素与图片。WinIE4/5应用于图片与表格，但对大多数文本元素如P及标题则忽略它，Opera 3.6看起来把图片的width设为100%，但这是一个错觉，因为先最小化视窗再最大化视窗会展现正确大小的图片。

5.5.25 浮动

浮动是CSS1规范中最复杂、最难应用的特征之一。基本的浮动通常能被所有浏览器支持，尤其是图片，但当规范被详细测试，或文档结构变得复杂时，浮动经常会发生错误，或根本不发生。文本元素的浮动尤其难以预测，尽管IE5及Opera大幅度清除了它们的行为，现WinIE4和Nav4是该特征的主要违背者。网页制作者要小心使用 `float`，且彻底检测使用 `float` 的页面。

5.5.26 `clear`

类似于 `float`，`clear` 不是可以简单支持的。基本的支持是有的，但当情况复杂起来时，浏览器的行为会失败。彻底检测使用该属性的页面。

5.6.1 `display: inline`

Opera 3.6总能正确处理 `inline`，但经常把回车当做 `BR` 元素，而不是空白。

5.6.3 `list-style-type: none`

MacNav4在使用这个值时给标记显示疑问标识。

5.6.5 `list-style-position-: inside`

MacIE4中设置这个值后，列表项的位置与格式有一些奇怪。

6.1 Length Units: ex

所有支持它的浏览器将ex计算成二分之一em。这是一个可用的近似，但在技术上是错误的。

6.3 color Units:<keyword>

Navigator会为任何外观关键字生成一个颜色。例如，`color: invalidValue` 会生成深蓝，`color: inherit` 会生成一种难看的绿色。

6.4 URLs:<url>

Navigator根据HTML文档决定相对URL，而不是根据样式表单。

词汇表

alternate
　　可选择的（样式表）

border
　　边框

bullet
　　项目符号

clear
　　清除

clip
　　剪切

color
　　颜色

container
　　容器

contextual selector
　　上下文选择符

CSS (Cascading Style Sheets)
　　层叠样式表

declaration
　　声明

directive
　　指示

DTD (document type definition)
　　文档类型定义

float
　　浮动

frame
　　框架

height
　　长度

hover
　　逗留

HTML
　　超文本标记语言

inheritance
　　继承

inline style
　　内联样式

keyword
　　关键字

margin
边界

marker
标识

overflow
溢出

padding
补白

pica
12磅活字

point
磅

property
属性

pseudo-rule
伪规则

selector
选择符

serif
衬线

side-offset
边偏移

specificity
特殊性

structure
结构

style
样式

unit
单位

user agent (UA)
用户代理

variant
变形

weight
权重

width
宽度

XML (Extensible Markup Language)
可扩展标记语言