别具光芒 CSS

属性、浏览器兼容与网页布局

李烨 编著

人民邮电出版社
POSTS & TELECOM PRESS

别具光芒 CSS

属性、浏览器兼容与网页布局

本书适用于

- 网页设计与制作人员
- Web标准网站开发人员
- CSS布局设计与美化人员
- 网页重构人员
- 网页设计与制作培训学员
- (X)HTML+CSS初学者

本书精彩内容

装帧设计：董志桢

分类建议：计算机/网页制作/CSS
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-18123-7

9 787115 181237 >

ISBN 978-7-115-18123-7/TP

定价：59.00 元

别具光芒CSS

属性、浏览器兼容与网页布局

李烨 编著

人 民 邮 电 出 版 社

北 京

图书在版编目（CIP）数据

别具光芒：CSS 属性、浏览器兼容与网页布局/李烨编著. —北京：人民邮电出版社，2008.10
ISBN 978-7-115-18123-7

Ⅰ. 别… Ⅱ. 李… Ⅲ. 主页制作—软件工具，CSS Ⅳ. TP393.092

中国版本图书馆 CIP 数据核字（2008）第 069034 号

内 容 提 要

本书结合大量范例与实际应用的实例，详细介绍了W3C发布的层叠样式表CSS 2.1规范，浏览器对于CSS 2.1规范解释的异同，以及使用XHTML和层叠样式表对网页进行结构化与美化的实际制作方法。本书内容由浅入深，不仅介绍了Web标准和层叠样式表的各个属性，还结合实例对属性的实际应用进行讲解，同时配合在不同浏览器内的效果展示，针对读者实际制作中可能遇到的问题，提供了解决问题的思路和方法。

本书适用于希望系统学习CSS的初学者，也适用于从事网页设计制作和网站建设的从业人员，也可以作为各大中专院校相关专业的教学辅导和参考用书，或作为相关培训机构的培训教材。

别具光芒——CSS 属性、浏览器兼容与网页布局

◆ 编　　著　李　烨
责任编辑　杜　洁

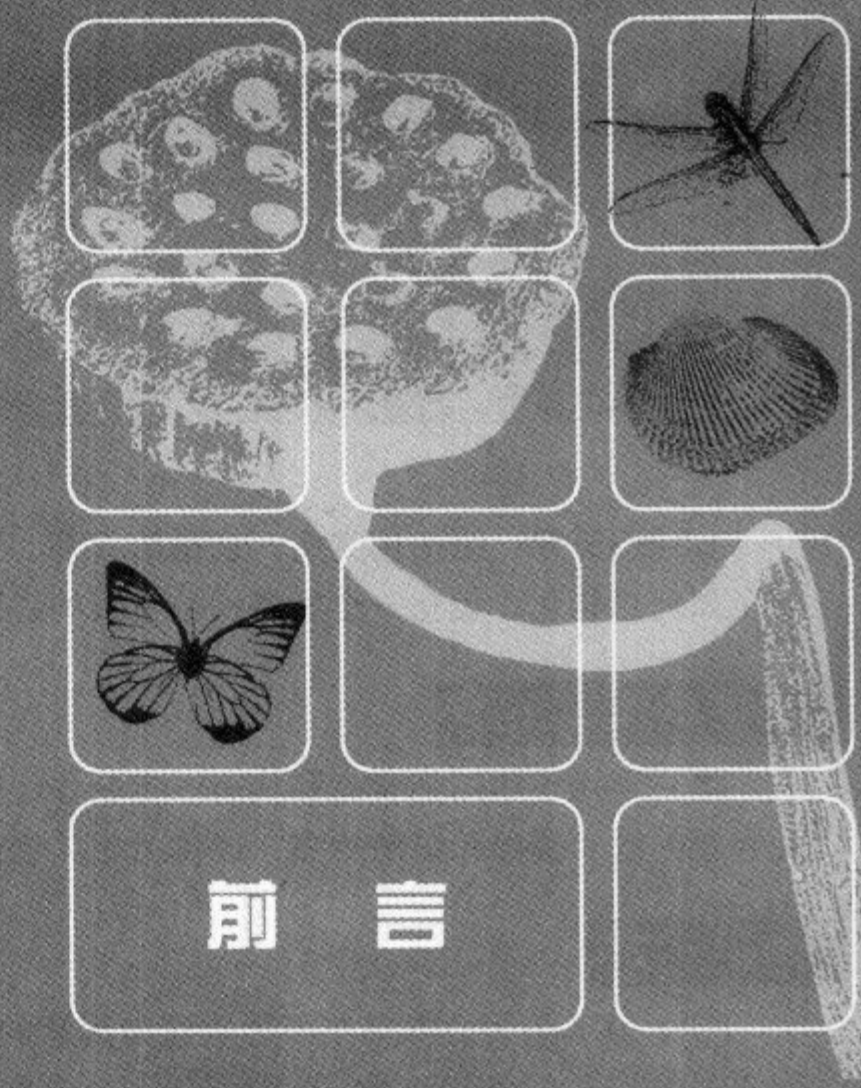
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：27
字数：811 千字　　2008 年 10 月第 1 版
印数：1－4 000 册　　2008 年 10 月北京第 1 次印刷

ISBN 978-7-115-18123-7/TP

定价：59.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154



前 言

关于本书

随着Web标准在国内的逐步普及，如何结构化和美化网页是摆在每个页面制作人员面前的问题。而将结构和表现分离，使得层叠样式表（CSS）的重要性凸显出来。

本书详细介绍了层叠样式表（CSS）的各属性以及原理，同时结合大量实用的范例解释属性的实际用途，还分析CSS属性在浏览器内可能遇到的各种情况，提供了详细的操作步骤，具有很强的实用性及参考性。在本书的最后还提供了一个网站布局的详细实例，详细演示了如何结构化内容以及CSS美化的实际应用。

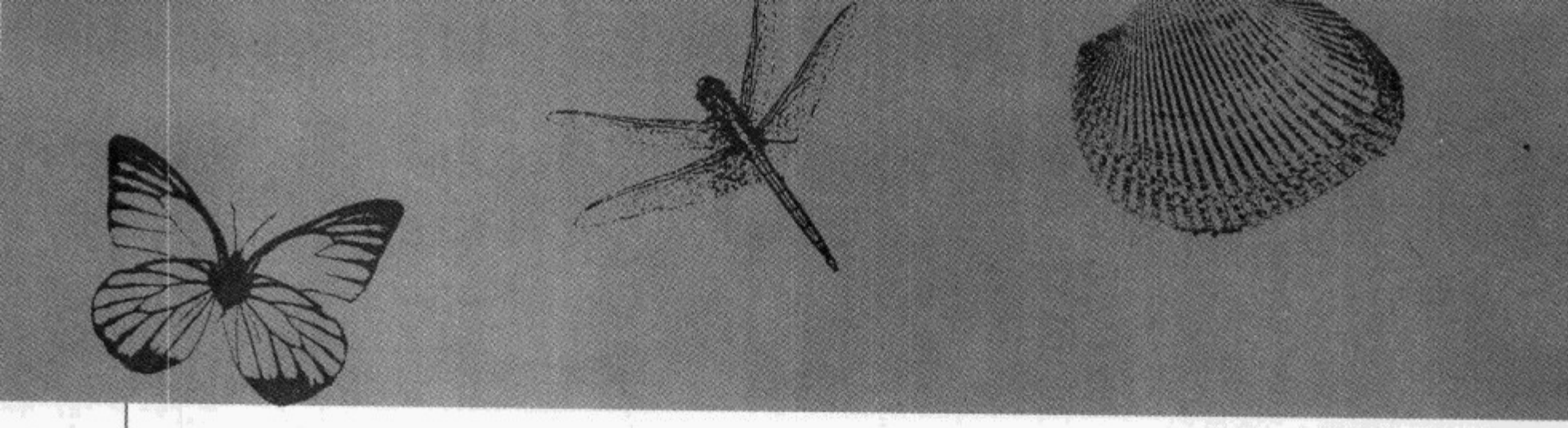
本书作者自1999年起开始从事网页设计及制作工作，先后在多家网络公司任职，积累了大量网页设计制作方面的经验，熟悉CSS网页布局和美化的多种技巧。本书就如何对网页进行分析及合理规划布局结构进行了一定的探讨，同时又结合了实例来讲解，不管是一个初学者还是想了解CSS的专业人士，都能在这本书中找到自己感兴趣的部分。

此外，本书通过大量注释详尽的图解为读者化解了阅读和学习上的障碍，使得读者可以更加快速和高效地提升自己的网页制作技能。

本书结构

本书分为3个部分，共17章，具体结构划分如下。

- 第1部分：Web标准，包括第1章和第2章。这部分内容主要介绍了Web标准的原理及（X）HTML的初步知识。
- 第2部分：层叠样式表CSS，包括第3~16章。这部分内容详细地讲解了W3C制定CSS 2.1规范的内容，包括选择器和属性等，并结合小型实例讲解了属性的实际用途。



第3部分：结构化实例，包括第17章。这部分内容主要通过从设计图的分析开始，展现了如何结构化、美化一个旅游网站首页的过程。

本书的读者对象

本书适用于CSS初学者和具有一定制作经验的网页制作者，也可以作为各大中专院校相关专业的教学辅导和参考用书，或作为相关培训机构的培训教材。

本书约定

（1）为了简便，本书在不影响读者阅读的前提下采用了简称，例如，“层叠样式表”简称“CSS”，（X）HTML指HTML和XHTML这两种语言。

（2）如果不做特殊说明，本书中所涉及的（X）HTML示例都假定在一个合法的（X）HTML文档的<body>中，而CSS则在文档的<style>标签中。

（3）本书所介绍的CSS规范为W3C网站所公布的CSS 2.1规范发布于2008年2月28日之前的内容。CSS 2.1勘误表请参见http://www.w3.org/Style/css2-updates/CR-CSS21-20070719-errata.html（英文）。

（4）为了使示例在浏览器内能更易于观察，在示例文件中一般会添加一些颜色、字体等设定，而这些设定不会影响示例本身的效果。读者可到作者的网站（http://www.ddcat.net）下载这些示例代码。

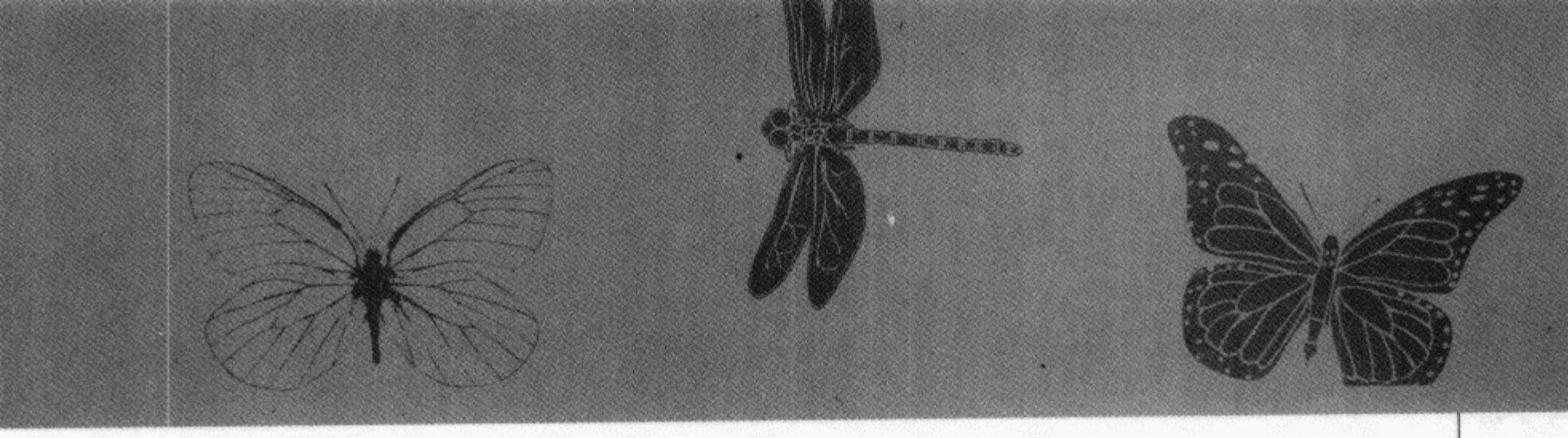
（5）为了精简示例代码，突出重点，本书中的部分代码中使用“……”代替代码中一些与本例关系不大的部分，如：

```
div {
……
font-size : 1em;
}
<div>……</div>
```

则表示“font-size : 1em”是与本例相关的设定，而其他部分可参见示例文件，一般是为了使示例更加清晰便于阅读而增加的颜色等设定。

（6）由于操作系统及浏览器的差异，CSS在浏览器内的表现也存在差异，本书将只针对Windows操作系统中的主流浏览器IE 6.0、IE 7.0、Firefox 2.0、Opera 9.2及Safari 3.0做测试演示，同时，大部分情况下，Firefox、Opera及Safari的显示效果是一致的，因此如果不做特殊说明，则只取其中之一进行效果演示。对于其他浏览器，读者可自行测试效果。

（7）在本书中，经常需要在不同的浏览器内观察效果，读者可免费下载安装浏览器Mozilla Firefox（http://www.mozilla.net.cn/）、Opera（http://www.opera.com/）和Safari（http://www.apple.com/safari/download/）以备测试使用。



（8）为了给读者提供更多的学习资源，同时弥补本书篇幅有限的遗憾，书中提供了大量的参考链接，许多本书无法详细介绍的问题都可以通过这些链接找到答案。因为这些链接地址会因时间而有所变动或调整，所以在此说明，这些链接信息仅供参考，本书无法保证所有的这些信息是长期有效的。

（9）广大读者如果有好的建议或在学习本书中遇到疑难问题，欢迎登录网站（http://bbs.ddcat.net）进行探讨，也可发电子邮件联系我们（dujie@ptpress.com.cn）。

属性说明

在本书中，对于每个CSS属性都有一个定义列表，其样式如下：

语法	font-family : [[<字体名> | <字体系列名>] [, <字体名>| <字体系列名>]*] | inherit
说明	设定元素内文本的字体名称
值	字体名：字体的名称。 字体系列名：某个字体系列
初始值	跟浏览器的设置有关
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

其中：

```
font-family : [[ <字体名> | <字体系列名> ] [, <字体名>| <字体系列名>]* ] | inherit
```

“font-family”是属性名称，而“:”后面是属性可以接受的值。每个属性所允许的取值都用下面的语法加以列举：

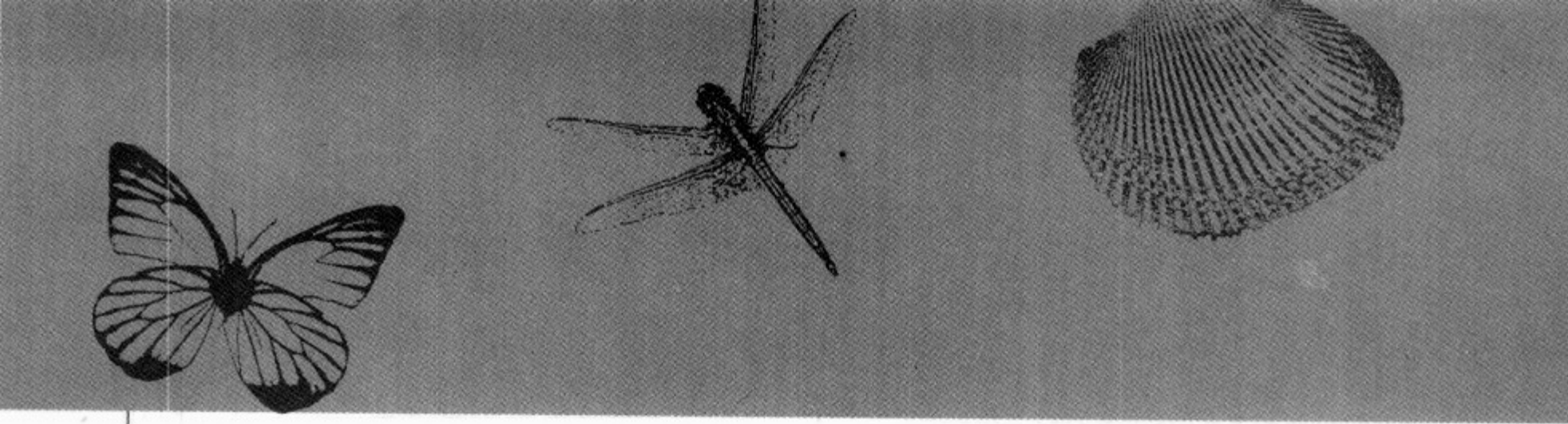
```
属性名 : [ <长度> | thick | thin ] (1, 4)
属性名 : [ <系列名称>, ] * <系列名称>
属性名 : <url>?<颜色> [ /<颜色> ] ?
属性名 : <url>||<颜色>
```

位于“<”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<'font-family'>来表示的。

没有被“<”和“>”包围的值为关键字值，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。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。

单竖线“|”分割的值表示这些值中的一个必须出现，如 [<字体名> | <字体系列名>]



表示<字体名>或者 <字体系列名>必须出现1个。

双竖线“||”，例如［ X || Y ］意味着X或Y，或者两者都出现，它们可以按任意顺序出现。

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，如“V W | X || Y Z”等同于“［V W］|［X || Y Z］］”。

每个文字或者括号必须跟以下修饰符中的1个。

● 星号“*”表示前面的值或者分组被重复0次或者多次。例如：“<示例值>*”就表示“示例值”可以出现任意次，包括0，而上限没有规定。

● 加号“+”表示前面的值或者分组被重复1次或多次。例如：“<示例值>+”就表示“示例值”至少必须出现1次，也可以多次。

● 问号“?”表示前面的值或者分组是可选的。例如“［Flash HTML］?”表示“FLASH HTML”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“<长度>{1, 4}”表示长度值最少出现1次，最多可出现4次。

以下给出几个例子，以帮助读者理解。

● HTML || CSS || JavaScript表示至少必须使用三者之一，而它们可以按任意顺序出现。例如“HTML”，“HTML CSS”，“JavaScript CSS HTML”都是合法的。

● ［Flash | JavaScript］? HTML || CSS表示“Flash”或者“JavaScript”可以使用，适用哪个是可选的。而且HTML或CSS（或者两者都有）跟在后面。这样可能出现的组合有“Flash HTML”、“Flash CSS HTML”、“JavaScript CSS”、“CSS HTML”等。

● HTML+ CSS表示1个或者多个HTML后面必须跟1个CSS，如“HTML HTML CSS”、“HTML CSS”、“HTML HTML HTML HTML CSS”等。

● HTML CSS{1,4}［Flash | JavaScript ］［PHP | ASP］表示“HTML CSS Flash PHP”、“HTML CSS CSS Flash ASP”等，CSS可以出现最多4次。

● ［ <字体名> | <字体系列名>］］, <字体名>| <字体系列名>］*］| inherit表示其可能的组合是“<字体名>”、“<字体名>, <字体名>”、“<字体名>,<字体系列名>,<字体系列名>”等，或者是“inherit”。

编者

2008.8

目 录 CONTENTS

第1部分 Web标准

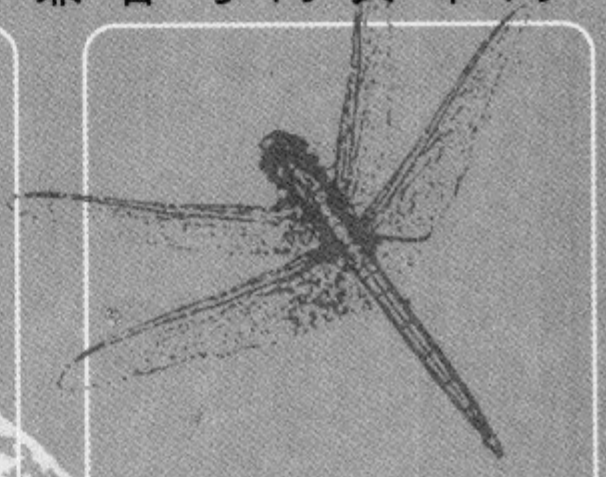
第2部分 层叠样式表CSS

第3部分　结构化实例

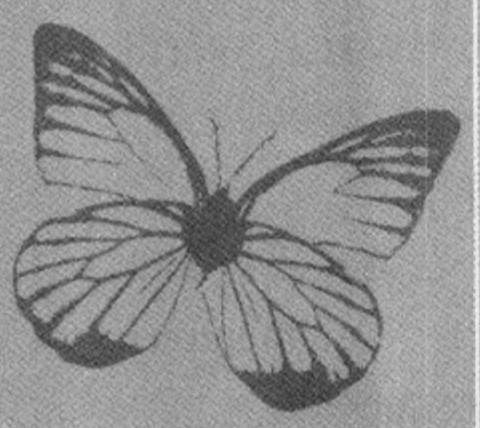
Chapter 17

第17章　旅游网站 ……356

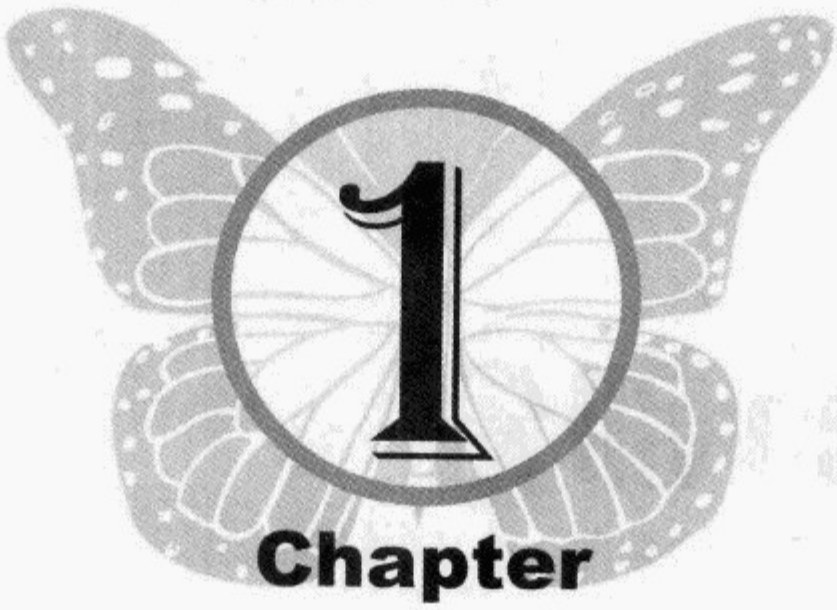
CSS 属性、浏览器兼容与网页布局



第1部分 Web 标准



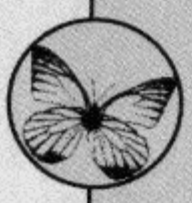
别具光芒 CSS



第1章 Web标准概述

Web标准并不是一个强制性的东西，它只是W3C（World Wide Web Consortium，万维网联盟）提出的一个建议性的文档。Web标准不是某一个标准，而是一系列标准的集合。

这些标准大部分由W3C起草和发布，也有一些是其他标准组织制定的标准，比如ECMA（European Computer Manufacturers Association，欧洲计算机制造联合会）的ECMAScript标准。



提示： W3C是一个非盈利组织，官方网站网址为http://www.w3c.org。根据W3C官方网站的介绍，W3C会员包括生产技术产品及服务的厂商、内容供应商、团体用户、研究实验室、标准制定机构和政府部门，一起协同工作，致力在万维网发展方向上达成共识。而ECMA是1961年成立的，旨在建立统一的计算机操作格式标准（包括程序语言和输入输出）的组织。

提示： 关于Web标准的更多文章，读者可以参阅中文网站“网页设计师”（http://www.w3cn.org/）。

1.1 Web标准概述

网页主要由3部分组成：结构（Structure）、表现（Presentation）和行为（Behavior）。对应的标准也分3个方面：

- 结构化标准语言，主要包括XHTML（Extensible HyperText Markup Language，可扩展超文本标示语言）和XML（Extensible Markup Language，可扩展标示语言）；
- 表现标准语言，主要包括CSS（Cascading Style Sheets，层叠样式表）；
- 行为标准，主要包括对象模型（如W3C DOM、ECMAScript等）。

结构是网页的骨架，例如：

```
<h1> WEB标准</h1>
 <p>WEB标准并不是一个强制性的东西，它只是W3C（World Wide Web Consortium，万维网联盟）提出的一个建议性的文档。WEB标准不是某一个标准，而是一系列标准的集合。</p>
```

其中，XHTML标签将内容区分定义为1级标题<h1>和段落<p>，将内容结构化可以使内容被检索、进行交互等。

表现是网页的外貌，也就是最终呈现在浏览器内的样子，通过层叠样式表（CSS）来实现页面的不同表现。行为是对内容的交互及操作，例如判断用户填写的资料是否正确、根据不同的用户显示不同的内容等。

符合Web标准的页面，结构合理而且代码清晰，良好清晰的结构使得搜索引擎能够方便地判断与评估信息，从而建立更精确的索引。同时，在更老版本的浏览器中——即使CSS/XSL样式无法解析，它也能显示出完整的信息和结构。

符合Web标准的页面也很容易被转换成其他格式文档，例如数据库或者WORD格式，也容易被移植到新的系统，比如机顶盒、PDA等——这是XML具有的优势。

符合Web标准的页面也具有天生的“易用性（accessibility，也译为：无障碍性）”，不仅仅是普通浏览器可以阅读，那些有残疾的人们也可以通过盲人浏览器、声音阅读器正常使用。

注意：Web标准并非只有上述这些简单内容，本书只讨论其中和静态页面制作有关的部分。

1.2 表现与结构的分离

对于网页制作者来讲，学习Web标准首先要明确的就是“表现与结构的分离”，如图1－1所示。

只有实现了结构与表现分离，才能很好地实现数据的检索、交换、易用性等。

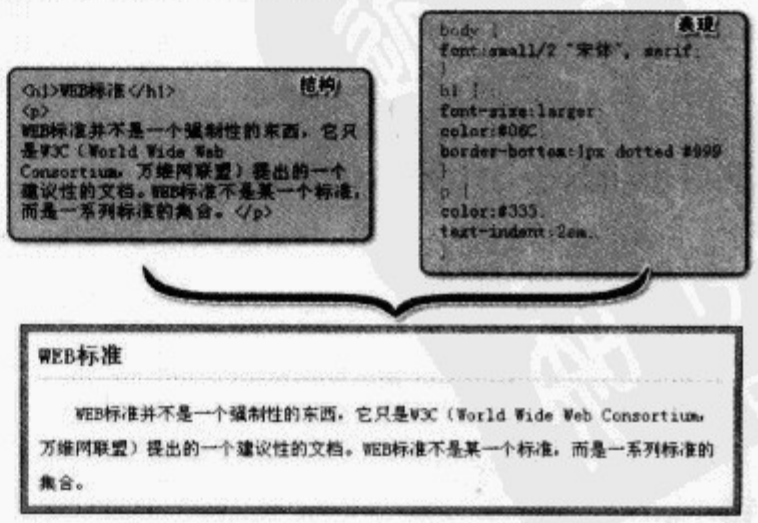
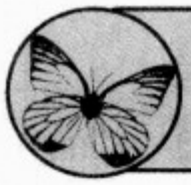


图1-1 表现与结构的分离

1.3 易用性

网站的“易用性（accessibility，也译为：无障碍性）”并非只针对残疾人，尽管这可能是一个主要原因。一个具有亲和力的站点对任何人来说，显示效果都同样出色，这里的“出色”并不是指外观的华丽，而是对内容的良好显示。网站应该可以被使用不同浏览器或浏览设备的人所访问，比如可上网的手机。

使用语义化、结构化的XHTML将是制作者创造易用性站点的最好的方法。



提示：要对文档的易用性有一个基础的概念，读者可以尝试使用一个文字化的浏览器（如Lynx）看看内容是否依然起作用。

在制作具有易用性站点的时候，要特别注意以下几点。

1. 避免使用框架

框架可以把浏览窗口分割成几个独立的部分，每部分都是由独立的HTML文档组成，虽然这样可以使用同样的头文件或者菜单，但是，使用框架将存在如下多项缺点。

- 不利于搜索。从搜索引擎搜到网站的访问者很可能访问的是缺少了某些重要信息（导航链接）的文档。
- 书签不可用。大多数浏览器不能在一个框架网站的页面上使用书签。当访问者打开书签后，将打开的是框架设置的默认地址，往往去的都是这个网站的首页。
- 打印变得愈加的困难。许多访问者在打印文档的时候，同样会遇到问题。大多数的浏览器都会要求你解开这个框架，否则不能打印。
- 浏览设备的局限性。不使用图形化浏览器（它支持框架）的人将无法访问站点，例如使用手机的访问者，就无法获得他们感兴趣的内容。

因此，易用性准则不建议大家使用框架。

2. 表单

表单对于提供用户交互的意义很重要，但是不合适的表单设置往往使网页的亲和力大打折扣。例如，某个很难选中的单选按钮，或者一个很长的下拉菜单。

一个普遍的问题是用什么给表单布局。一些人说，可以把一个表单看作一个列表数据，可以用表格来构造，然而还有一些人提议用CSS来布局。两种方法都是可用的，但是如果你用的是表格，请确保这个表格是有意义的，而且要保证当一个包含表单的表格被线性后是可用的。一些相关的标签（如<label>、<fieldset>和<legend>）可以使表单更具有亲和力，更容易使用。

3. JavaScript和cookies

不要依靠JavaScript。许多访问者可能会认为如果禁用JavaScript，就可以更加安全地浏览网页或者可以避免弹出窗口；也可能某些访问者正在使用的浏览器根本就不支持JavaScript。

当然有一些例子显示JavaScript能够给访问者提供更好的体验。一个例子就是校验表单的输入。这并不是说不应该使用JavaScript，而是说不应该创建一个完全依靠JavaScript运行的网站。同样的事也适用于cookies。如果访问者不接受cookie网站就拒绝运行，那么cookie就使用错了。

1.4 难点所在

虽然Web标准看上去很诱人，但是初学者也很容易犯错误，以至于使自己的学习进度停滞不前，一般会遇到的困难基本为以下几点。

1.4.1 DIV+CSS不等于Web标准

刚刚开始尝试制作符合Web标准页面的制作者们往往认为："Web标准就是DIV+CSS，就是将原来用表格<table>的地方用层 替代，原来是表格嵌套，现在是层嵌套"——这是一个最大的错误。

正如1.2节所述，要实现"表现与结构的分离"，因此，制作者首先要做的事情，就是抛弃将设计图视为内容的思维方式。

页面里有的仅仅是内容，没有修饰的情况下，它看上去就是一张白纸，上有一些文字和图片（这个图片是指内容中的图片，例如新闻中的照片等）。这些文字图片仅仅是依次罗列下来，没有任何样式。然后加入表现，将所有修饰的图片作为背景，用CSS来定义每一块内容的位置、字体、颜色等。

这样制作的页面才是内容与表现分离的，就是说，当你抽掉CSS文件，剩下的就是干净的内容。这样才能在文本浏览器、手机、PDA等设备中阅读，才能随时修改CSS实现改版。

1.4.2 正确使用XHTML标签

XHTML的标签不是用来做"表现"的，而是用来定义结构的，因此，对不同的内容使用正确的XHTML标签对于建立良好的文档结构是很重要的。何处使用<h1>、何处使用<p>都要合理，这样不仅便于理解文档内容，同时对于CSS的编写也很重要。

许多人已经习惯用元素来控制表现，而不是结构。例如，一段列表内容可能会使用下面这样的标识：

```
项目一<br />
项目二<br />
项目三<br />
```

如果采用一个无序列表代替也许会更好：

```
<ul>
 <li>项目一</li>
 <li>项目二</li>
 <li>项目三</li>
</ul>
```

读者或许会觉得"<li>显示的是一个圆点，我不想用圆点"。事实上，CSS没有设定元素看起来是什么样子，完全可以用CSS去掉圆点。一个具有良好结构的HTML页面是网页的基础，也是一个好的开始，因此学习HTML的基础知识是非常有必要的。

提示：HTML也有不同的版本，目前使用最广泛的是HTML 4.0版。本书将只讨论一些HTML的基本用法及注意事项。读者可以访问中文网站"W3school"（http://www.w3school.com.cn/）或者英文网站"W3Schools"（http://www.w3schools.com/），更详细地学习HTML 4.0的知识。

同时要注意的是，不要滥用HTML标签，这点往往体现在过渡嵌套使用标签上。虽然有时候为了达到更好控制内容及布局的目的，需要嵌套一些 或者，但是不要任意地去嵌套，当嵌套了过多的 才能完成表现的时候，往往需要重新考虑HTML的结构和现用的方法是否合理了。

1.4.3 表格本身并没有被抛弃

表格<table>并没有被抛弃，而是不要再用它来布局页面，表格仍有它从定义起就拥有的用途——存放表格性数据，例如，一个班级的学生成绩单或者一个仓库的库存商品。

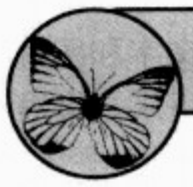
放弃表格不用，而使用其他的标签来模拟表格，无疑是舍本逐末的举动。

1.4.4 善于利用CSS

也许读者在写HTML时已经养成习惯，当希望字体大一点就用<h1>，希望在前面加个点符号就用<li>，似乎<h1>的意思是大的，<li>的意思是圆点。而实际上，通过CSS，<h1>能变成小的字体，<li>能够变成一张图片等。不要强迫用结构元素实现表现效果，而应该使用CSS来确定元素的外观。

例如，可以使原来默认的6级标题看起来大小一样：

```
h1, h2, h3, h4, h5, h6 {
font-family: 宋体, serif;
font-size: 12px;
}
```



提示：关于CSS的详细讲解，请见本书的第2部分。

对于多次引用的样式可以用class来定义，不需要每个元素都定义id；也不是一定要用 ，有的内容完全可以用<p>来代替，同样都是块级元素，一样有盒模型的7个参数。

1.4.5 不要滥用class

对于初学者来讲，如何为HTML元素添加CSS是一个非常困惑的事情。虽然看上去表现与结构分离了，但是几乎每个元素都被添加了class，而且样式定义详细到几乎每个属性都要单独定义成class。例如，某几个元素都有相同的灰色背景色，那么它们就会有一个相同的定义，如右所示：

```
<div id="nav" class="bg1">……</div>
<ul id="menu" class="bg1">……</div>
```

这么做的弊端在哪里？当需要将nav层的背景变为蓝色的时候，就需要修改HTML代码，给它另一个class。而如果要修改无序列表menu的背景色，同样也要修改HTML代码才可以完成，这无疑又陷入了另一个怪圈。

其实可以如右定义CSS：

```
#nav,
#menu {
background-color:#ccc;
}
```

灵活地运用CSS不同的选择器（selector）来进行CSS定义，将大大提高工作效率。

小窍门：将通用的样式写在外部CSS文件中，然后在页面内调用，同时还可以将不同的CSS定义分在几个文件中。

1.4.6 应对浏览器

由于Web标准不是强制性的，因此，浏览器的制造商完全可以不遵循它。而为了竞争市场，各浏览器之间也保留了一些自己私有的东西（特别是目前使用范围最广的Windows IE）。因此制作者为了达到各种浏览器之间效果的统一而耗费很多时间，使用各种的“技巧”（CSS Hack）来使效果趋于完美。但是，有时候制作者应该停下来思考一下。

- 在不同的浏览器之间相差几个或者更少的像素真的有那么不能忍受吗？
- 只是为了视觉上更眩目而要增加大量的代码来实现浏览器兼容真的有那么重要吗？

“能够在任何浏览器中显示”并不意味着“在任何一个浏览器中显示的效果相同”。使一个文

档在不同的浏览器和平台上都有相同的显示效果是不可能的，即便只用图片也不可能，因为发布在网上的文档将被不同的操作系统上的不同的浏览器软件所显示，并显示在不同大小与质量的显示器上（或者显示在非显示器上），访问者也可能改变了浏览器的默认字体或者其他喜好。CSS Hack并非是解决浏览器兼容问题的万能药。

1.4.7 “通过验证”并不是最终目的

W3C校验仅仅是帮助制作者检查XHTML代码的书写是否规范，CSS的属性是否都在CSS 2的规范内。代码的标准化仅仅是第一步，并不是说通过了校验，网页就标准化了。

使用很多方法都可以欺骗校验程序而得到通过验证的那个小图标。让网页具有良好的结构、更快的浏览速度、更友好的界面以及对更多设备的支持才是最终目的。

提示： XHTML校验网址为http://validator.w3.org/，校验方式有网址校验、文件上传校验。CSS校验网址为http://jigsaw.w3.org/css-validator/，校验方式有网址校验、文件上传校验、直接贴入代码校验。

1.5 SEO简介

SEO是Search Engine Optimization的缩写，中文为“搜索引擎优化”，一般可简称为搜索优化。与之相关的搜索知识还有Search Engine Positioning（搜索引擎定位）、Search Engine Ranking（搜索引擎排名）。

SEO的主要工作是通过了解各类搜索引擎如何抓取互联网页面、如何进行索引以及如何确定其对某一特定关键词的搜索结果排名等技术，来对网页进行相关的优化，使其提高搜索引擎排名，从而提高网站访问量，最终提升网站的销售能力或宣传能力的技术。

搜索是除了电子邮件以外被用得最多的网络行为方式。通过搜索引擎查找信息是当今网民们寻找网上信息和资源的主要手段。搜索引擎营销已经成为网络营销最重要的组成部分。如何使自己的网站被主要的搜索引擎收录，然后获得较高的排名，已成为网站建设者们绞尽脑汁的话题。

在国外，SEO开展较早，那些专门从事SEO的技术人员被Google（谷歌）称之为“Search Engine Optimizers”，简称SEOs。由于Google是目前世界最大搜索引擎提供商，所以Google也成为了全世界SEOs的主要研究对象，为此Google官方网站专门有一页介绍SEO，并表明Google对SEO的态度。

而在页面制作的过程中，应该注意以下几个事项。

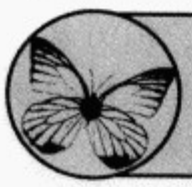
（1）静态页面。将信息页面和频道、网站首页改为静态页面，有利于搜索引擎更快更好地收录。

（2）页面标题（Page Title）的关键词优化。必须列出信息的标题、网站的名称以及相关关键字。

（3）<meta>标签的优化。通过<meta>标签设置页面的关键字和简介，是过去搜索引擎优化的重要手法，现在已经不是关键因素，但仍不可忽略，主要包括说明（description）和关键字（keywords）。

关键字密度要适度，通常为2%~8%，也就是说你的关键字必须在页面中出现若干次，或者在搜索引擎允许的范围内，要避免堆砌关键字。

（4）针对Google（谷歌）制作Sitemaps。Google的Sitemaps是对原来robots.txt的扩展，它使用XML格式来记录整个网站的信息并供Google读取，使搜索引擎能更快更全面地收录网站的内容。可以使用Google提供的Sitemap生成器制作（需要技术人员制作）。



提示： Google的Sitemap生成器URL为https://www.google.com/webmasters/tools/docs/zh_CN/sitemap-generator.html。

也可以由技术部人员制作更全面的Sitemaps。

（5）图片的关键词优化。图片的替代（alt属性）关键词也不要忽略，其另外一方面的作用是，当图片不能显示的时候，可以给访问者一个替代解释语句。

（6）避免表格的嵌套。目前本站的表格嵌套太多，搜索引擎通常只读取3个<table>的嵌套，如果太多，会造成部分有用信息没有被检测到。

（7）采用Web标准进行网站重构。尽量使网站的代码符合W3C的HTML 4.0或XHTML 1.0规范。通过XML＋CSS技术进行网站重构，减少了表格及冗余代码，提高网站页面的扩展性和兼容性，可以使更多浏览器支持。

（8）网站结构的扁平化规划。目录和内容结构最好不要超过3层，如果有超过3层的，最好通过子域名来调整和简化结构层数。另外目录命名的规范做法是使用英文而不是拼音字母。

（9）页面容量的合理化。合理的页面容量会提升网页的显示速度，增加对搜索引擎蜘蛛程序的友好度。同时建议JavaScript和CSS尽量用链接文件。

（10）外部文件策略。把JavaScript文件和CSS文件分别放在外部文件中。这样做的好处是把重要的页面内容放到页面顶部，同时能缩小文件大小，这样有利于搜索引擎快速准确地抓取页面重要内容。其他的字体（<font>）和格式化标签（
等）也尽量少用，建议采用CSS定义。

（11）外部链接。尽可能多地让其他跟你主题相关的网站链接本站，同时尽量同PR值更高的网站进行相互链接。如果网站提供与主题相关的导出链接，被搜索引擎认为有丰富的与主题相关的内容，也有利于排名，例如各类招商网站、投融资网站的概念。另外避免了链接不顾质量的大面积撒网，对搜索引擎而言宁少要精。

提示： PR值全称为PageRank，PageRank（网页级别）是Google用于评测一个网页“重要性”的一种方法。在综合了诸如Title标识和Keywords标识等所有其他因素之后，Google通过PageRank来调整结果，使那些更具“重要性”的网页在搜索结果中另网站排名获得提升，从而提高搜索结果的相关性和质量。

（12）网站地图。网站自身的网站地图是搜索引擎更全面索引收录你的网站的重要因素。建议制作基于文本的网站地图，内含网站所有栏目、子栏目。网站地图的3大因素：文本、链接、关键词，都极其有利于搜索引擎抓取主要页面内容。特别是动态生成目录网站尤其需要创建网站地图。

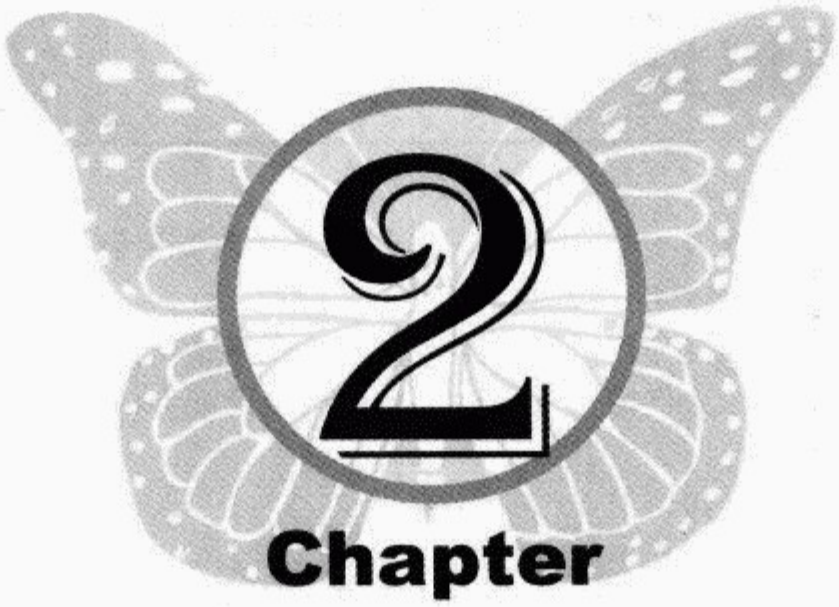
（13）图像热点。除AltaVista、Google明确支持图像热点链接外，其他引擎暂不支持。当“蜘蛛”程序遇到这种结构时，就会无法辨别。因此尽量不要设置图像热点（Image Map）链接。

（14）Flash应用。Flash由于不含文字信息，应尽量用于功能展示和广告，少用于网站栏目和页面。

（15）JS脚本。在不支持JS脚本的浏览器里<noscript> 标签会起到重要提示作用，对搜索引擎的Spider搜索也会有帮助。

（16）Frame框架。Frame标签会被搜索忽略，尽量少用，如果一定要用，则应正确使用<noframe>标签， 在<noframe></noframe>区域中包含指向框架页的链接或带有关键词的描述文本，同时在框架以外的区域也要出现关键词文本。

（17）资讯的内部链接。这些内部链接有助于提高网站排名和PR值，例如相关资讯、推荐资讯等。



第2章
结构与XHTML

对于网页制作者来说，理解Web标准首先要理解结构与表现分离的意义，对于刚刚接触网页制作的人或者已经使用表格布局很长时间的人来说，这也是难点所在。

2.1 理解结构与表现

目前网页的制作往往都是从一张设计图开始的。美工人员用设计软件制作一张页面效果图，而制作人员负责把这个效果图转换成HTML文件。

对于制作人员来说，将图片分割然后再组合，使用表格是再简单快捷不过的了，特别是现在可视化开发软件越来越强大，人们习惯于动动鼠标点点设设就完成了，而不去考虑实际的代码是什么乱七八糟的样子。

例如，下面的HTML代码片段：

```
<table border="0" cellpadding="0" cellspacing="0" width="100%">
 <tbody>
 <tr>
  <td width="20"><img src="/images/index_07.gif" height="97" width="14"></td>
  <td>
   <table id="new" border="0" cellpadding="0" cellspacing="0" width="100%">
    <tbody>
    <tr>
     <td valign="top"><font color="#103667">最近更新</font></td>
    </tr>
    <tr>
      <td align="left" valign="middle">  姥姥家的小厨房：<a href="/wenzi/chufang/021.htm">正月十五雪打灯
</a></td>
    </tr>
    <tr>
     <td align="left" valign="middle">  猫走四方：<a href="/wenzi/sifang/sifang_005.htm">神游</a></td>
    </tr>
    <tr>
      <td align="left" valign="middle">  猫说故事：<a href="/wenzi/gushi/x_001.htm">十分钟年华老去
</a></td>
    </tr>
    <tr>
     <td align="right" valign="bottom">----&amp;gt;&amp;gt;&amp;gt;&amp;gt;<a href="/news.htm">以前更新</a></td>
    </tr>
    </tbody>
   </table>
  </td>
     <td align="right" width="20"><img src="/images/index_08.gif" height="97" width="14"></td>
 </tr>
 </tbody>
</table>
```

这么多的代码只是为了实现一个“最近更新”的列表效果，如图2-1所示。

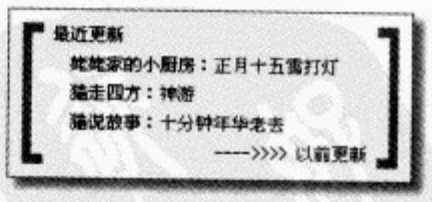


图2-1 代码效果图

可以看出，代码嵌套了若干层的<table>、<tr>和<td>标签，这不仅使HTML源代码阅读起来非常困难，更重要的是对于数据内容的搜索和重复利用都非常不利，同时大量的冗余代码使页面文件变得庞大，增加服务器的负载。而当设计人员改变某些设计（例如去掉左右的边框）的时候，往往需要重写整段代码。

解决这些问题的一个有效的方法就是，把结构、行为与表现分离。

2.1.1 内容

在分离结构与表现之前，首先要明确什么是页面的内容。

内容就是访问者真正想了解的信息，它可以包含数据、文档或者图片、多媒体等。注意这里强

调的“真正”，是指纯粹的数据和信息本身，而不包含辅助的信息（如装饰性的图片），如图2-2所示。

因此图2-2中实际的内容为：

［壁纸］似水流年（Coffee+Time）买了个咖啡磨，终于可以磨我的咖啡豆了。于是，屋子里飘荡着淡淡的清香。这几天一直在收拾屋子，随便做了几张桌面，也算换个新。

不过，这样的内容不利于阅读，也无法美化，更不利于数据搜索及利用，因此需要把它们结构化。

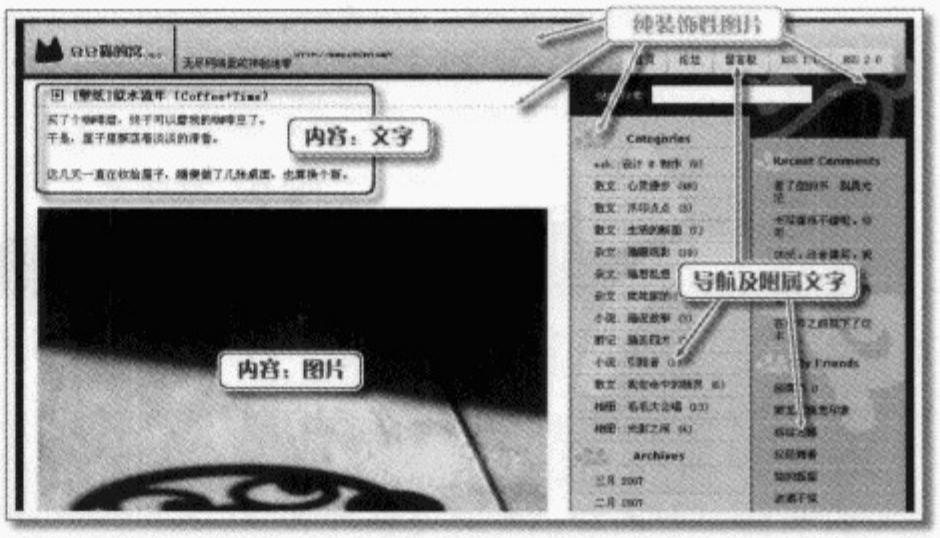
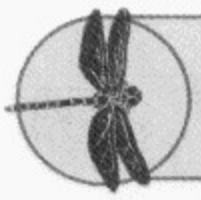


图2-2　网页的组成



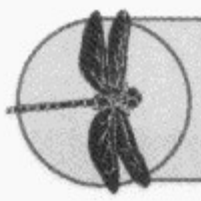
注意：页面中的导航及栏目列表等，从严格意义上讲，是不属于“内容”的，但是作为一个美观且易用性比较强的网页，这些又是不可缺少的。

2.1.2 结构（Structure）

网页的结构化就是将没有格式的内容用HTML或者XHTML标签分割成不同的部分，使内容更加具有逻辑性、易用性，同时使内容可以由CSS来控制外观表现。

对于图2-2中的内容，（X）HTML的结构是很简单明了的，如下所示：

```
1级标题 → <h1>［壁纸］似水流年（Coffee+Time）</h1>
  段落 → <p>买了个咖啡磨，终于可以磨我的咖啡豆了。于是，屋子里飘荡着淡淡的清香。这几天一直在收拾屋子，随便做了几张桌面，也算换个新。</p>
  段落 → <p><img src="/sheying/photo/coffee_01.jpg"/></p>
```



注意：网页的内容远远比这两个例子要复杂，因为页面内容一般还要包含网站的名称标识、导航、菜单等附属内容，同时为了能够实现样式控制可能还要添加一些其他的代码。

而图2-1所示的内容，HTML结构应该如下所示：

```
标题 → <h2>最近更新</h2>
列表 → <ul>
         <li>姥姥家的小厨房：<a href="/wenzi/chufang/021.htm">正月十五雪打灯</a></li>
         <li>猫走四方：<a href="/wenzi/sifang/sifang_005.htm">神游</a></li>
         <li>猫说故事：<a href="/wenzi/gushi/x_001.htm">十分钟年华老去</a></li>
       </ul>
标题 <h3><a href="/news.htm">以前更新</a></h3>
```

从上面的结构代码读者可以发现，里面没有包含字体、颜色、背景图片等信息，这就是结构与表现的分离。

2.1.3 表现（Presentation）

虽然定义了结构，但是用浏览器浏览页面会显得很呆板，文字的颜色没有变化、没有背景图片、没有花边或者线条的修饰。而这些文字颜色、背景图片、文字的位置等用来改变内容外观的东西，称之为“表现”，如图2-3所示。

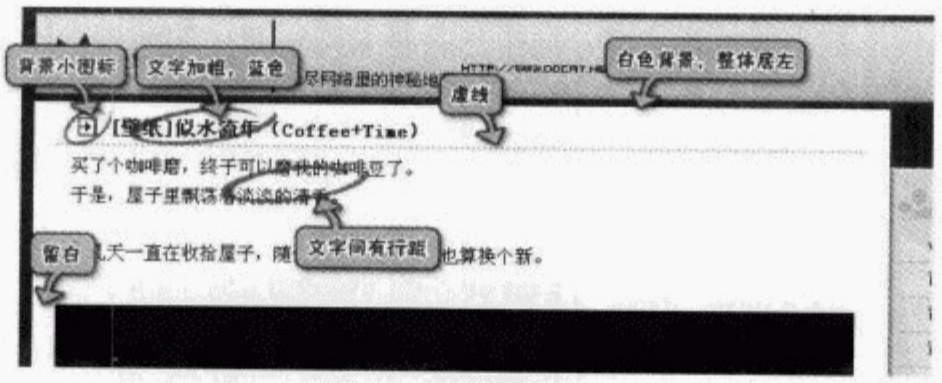


图2-3 网页的表现

2.1.4 行为（Behavior）

行为就是对内容的交互及操作效果，例如，使用JavaScript判断一些表单提交，或者实现菜单的显示和隐藏等。

（X）HTML页面是由“结构、表现和行为”这3方面组成的。内容是基础，然后是附加上结构和表现，最后再对它们做点“行为”，如图2-4所示。

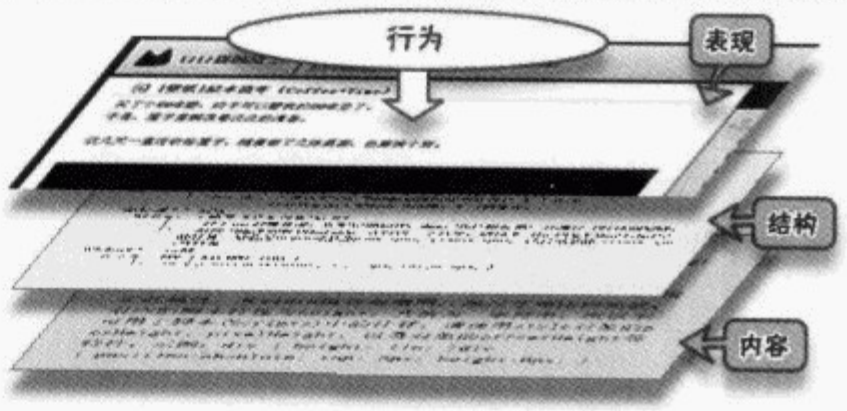


图2-4 结构、表现和行为的关系

2.2 从HTML到XHTML

网页是由超文本标记语言（HTML，HyperText Markup Language）构成的。“超文本”就是指页面内可以包含图片、链接、音乐、程序等非文字的元素；“标记”就是说它不是程序语言，只是由文字及标记组合而成。浏览器或者其他可以浏览网页的设备将这些HTML语言“翻译”过来，并照定义的格式显示出来，转化成我们看到的网页。

2.2.1 HTML简史

HTML作为定义万维网的基本规则之一，最初由蒂姆·本尼斯李（Tim Berners-Lee）于1989年研制出来。

HTML的设计者是这样考虑的：HTML格式将允许科学家们透明地共享网络上的信息，即使这些科学家使用的计算机差别很大。因此，这种格式必须具备如下几个特点。

- 独立于平台，即独立于计算机硬件和操作系统。这个特性对各种受从是至关重要的，因为在这个特性中，文档可以在具有不同性能（即字体、图形和颜色差异）的计算机上以相似的形式显示文档内容。
- 超文本。允许文档中的任何文字或词组参照另一文档，这个特性将允许用户在不同计算机中的文档之间及文档内部漫游。
- 精确的结构化文档。该特性将允许某些高级应用，如HTML文档和其他格式文档间互相转换以及搜索文本数据库。

本尼斯李选择使用SGML（标准通用标记语言，Standard Generalized Markup Language）作为HTML的开发模板。作为一种当时正在出现的国际标准，SGML具有结构化和独立于平台的优点。SGML的标准化水平也确保了它长久的生命力，这意味着采用SGML格式的文档在相当长的时间里不需要重新构建。

SGML是独立于平台的，因为它对文档的语义结构或含义进行编码描述，而不是对文档的实际外观进行编码描述。因此，某书某章节的标题将标为“Chater Title”，而不是“Helvetica 18pt Centered”。如果在不具备Helvetica字型或不支持不同大小字母的计算机上显示文档，则后一种样式会失败，而前一种样式可以在任何系统上（智能化地）显示。每个读者都以一种对其计算机有利的方式定义章节标题的外表，相应地，并以这种风格来规范所有的文本。

这种结构的另一个特征是：按语义编码的文本可以由计算机更智能地自动处理。例如：如果每个章节标题都用Chapter Title标志，再把章节号码作为一种属性，读者就可以要求只看第18章，SGML软件相应地会查找第18章标题和第19章标题，并抽取它们之间的所有内容。如果不用标准格式的字体和代码来标志文本的话，这个工作对计算机来说是无法完成的。

SGML的一大优点是它的灵活性。SGML本身并不是一种格式，而是定义其他格式的一种规范，用户可以创建新格式来编码某类文件（如技术手册、电话号码秒簿和法律文书）的所有结构，只需先阅读定义，任何能使用SGML的软件都能读懂它。人们已经为普通文档和十分专业化的文档建立了许多的文档类型定义（DTD，Document Type Definitions）。HTML只是一种DTD，或SGML的一种应用。

自1989年以来，HTML及万维网的使用和发展有了巨大的变化。当美国国家超级计算机应用中心（NCSA，National Center for Supercomputing Applications）在1993年初首次构建Mosaic浏览器时，NCSA的科学家们把自己需要的特性添加到HTML中，包括直接插入图形。当允许人们把位图、照片和图表放入到文档中以后，万维网的规模和使用出现了爆炸性的增长，第二年，HTML的发展很快。HTML的新标记不时地被一个又一个的浏览器引入，有一些新标记流行起来，而有一些又消失了。有些增加部分设计得很糟，很多甚至不遵从SGML规范。

到了1994年，HTML几乎以失控的状态发展。在互联网工程工作小组（IETF，Internet Engineering Task Force）主持下，1995年11月在瑞士日内瓦举行的第一次WWW会议上成立了一个HTML工作小组。它的主要任务是把HTML形式化成为一种 SGML DTD，称之为HTML Level 2（HTML 2.0，由本尼斯李最初设计的HTML被定义为Level 1）。标准化之后，HTML就可以被安全地扩展到将来的各个级别的版本，从而利用了SGML的实质性能和它的格式化结构。

尽管有关的各方从来没有取得完全一致的共识，但万维网联盟HTML工作组（World Wide Web Consortium's HTML Working Group）还是集中了1996年的万维网发展的成果，产生了HTML3.2版本。

HTML 4.0及其以后版本继承了以往版本的所有特点，并在以下几个方面有所发展：

- 更加明确了文档的结构和表现形式上的区别，以鼓励使用格式表（style sheet）来取代使用元素和属性进行表现的方式。
- 更加优良的表单（form）性能，加入了访问关键词（access key）、构建对称的表单控件能力、构建对称的下拉菜单控件的能力和动态标签（active label）。
- 在文本描述的标记中包含对象。
- 一种新用户端的文本包括在图像映射元素（map element），使得网页设计者可以集成文本和图像链接。
- 可以将替代图像的文本包括在图像元素（IMG element）中，也可以将图像映射（image map）包含到区域元素中（area element）。
- 在所有元素中支持title和language两个属性。
- 更多的表格属性，包括Caption、Column groups和方便的非可视信息（non-visual reading）的表现机制。

由HTML诞生的初衷以及其定义，也可以看出，HTML是用作“结构化”内容的，而不是来“美化”内容的。

2.2.2 HTML的缺点

HTML发展到今天存在3个主要缺点不能适应现在越来越多的网络设备和应用的需要，比如：

- 手机、PDA、信息家电都不能直接显示HTML；
- 由于HTML代码不规范、臃肿，浏览器需要足够智能和庞大才能够正确显示HTML；
- 数据与表现混杂，这样你的页面要改变显示，就必须重新制作HTML。

因此HTML需要发展才能解决这个问题，于是W3C又制定了XHTML（Extensible HyperText Markup Language，可扩展超文本链接标示语言），XHTML是HTML向XML过度的一个桥梁，它是一种增强了的HTML。

XML是Web发展的趋势，XHTML是当前替代HTML 4.0标记语言的标准，使用XHTML 1.0，只要遵守一些简单规则，就可以设计出既适合XML系统，又适合当前大部分HTML浏览器的页面，这个指导方针可以使Web平滑地过渡到XML。

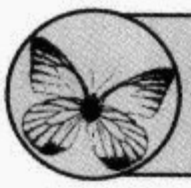
使用XHTML的另一个优势是它非常严密，同时XHTML能与其他基于XML的标记语言、应用程序及协议进行良好的交互工作。XHTML是Web标准家族的一部分，能很好地工作在无线设备等其他用户代理上。在网站设计方面，XHTML可助制作者去掉表现层代码的恶习，帮助制作者养成标记校验来测试页面工作的习惯。

提示：在本书完稿之日止，HTML 5.0正在研发当中，它增加了一些用于结构化的新标签，以求能更加适应Web标准，不过其推行仍需要用户端的广泛支持。因此目前使用XHTML仍然是比较好的选择。

2.2.3 从HTML到XHTML

使用HTML 4.01是可以制作出现代的、结构化的、兼容标准的站点的。然而为了做到向整洁的语义化的代码转变，并且为XML和未来的其他标记语言做好准备，作者建议使用XHTML1.0来制作网页。

XHTML是基于HTML的，它是更严密、代码更整洁的HTML版本，所以只要注意其中的要点，就能很容易地向XHTML迈进。



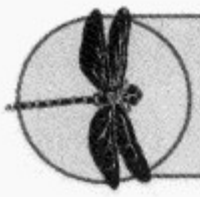
提示：在W3C的网站（www.w3c.org）上有一个开放源代码（open-source）的软件叫HTML Tidy，可以帮助使用者直接将HTML转换为XHTML。

XHTML和HTML之间最大的区别在于以下几个方面。

1. 选择DTD定义文档的类型

DOCTYPE是document type（文档类型）的简写，用来说明本文件用的XHTML或者HTML是什么版本。DTD是一个XML文档，解释了哪些标签、属性或值对于HTML的一个特定类型是有效的。在XHTML中必须声明文档的类型，以便于浏览器知道正浏览的文档是什么类型的，并且声明要位于HTML文件的第一行，如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>标题</title>
</head>
<body>
 body的内容
</body>
</html>
```



注意：DOCTYPE声明不是XHTML文档的一部分，也不是文档的一个元素，所以没必要加上结束标签。

其中的DTD（如xhtml1-transitional.dtd）叫文档类型定义，里面包含了文档的规则，浏览器就根据定义的DTD来解释页面的标签，并展现出来。不指定文档类型或者代码错误，都将导致浏览器进入一种“怪异模式（Quirks Mode）”，浏览器会按照自己的理解去解释HTML文档和CSS。关于浏览器的工作模式，请参见本书［16.1.3 浏览器的的工作模式］一节。

要建立符合标准的网页，DOCTYPE声明是必不可少的关键组成部分。XHTML 1.0 提供了3种DTD声明可供选择。

● 过渡的（Transitional）。要求非常宽松的DTD，它允许继续使用HTML 4.01的标识（但是要符合xhtml的写法）。完整代码如右：

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

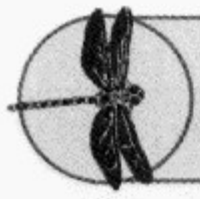
● 严格的（Strict）。要求严格的DTD，不能使用任何表现层的标识和属性，完整代码如右：

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

● 框架的（Frameset）。专门针对框架页面设计使用的DTD，如果页面中包含有框架，需要采用这种DTD。完整代码如右：

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

对于初次尝试Web标准的制作者来说，只要选用过渡型的声明（xhtml1-transitional.dtd）就可以了。它依然可以兼容表格布局、表现标识等。但是对于要提高的制作者来说，推荐使用严格的声明（xhtml1-strict.dtd），然后通过校验来检查是否做到了表现和结构的分离。



注意：如果使用严格的DTD，则有些被XHTML废弃的标签和属性，将不起作用，例如<font>标签和target属性。本书绝大部分的示例文件都采用了严格的（xhtml1-strict.dtd）声明。

2. 设定一个名字空间（Namespace）

通俗地讲，名字空间就是给文档做一个标记，告诉别人这个文档是属于谁的。名字空间声明允许通过一个URL来识别名字空间，只要直接在DOCTYPE声明后面添加如下代码：

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

其中“XMLns”是“XHTML namespace”的缩写，“http://www.w3.org/1999/xhtml”并不是指一个具体的文件，仅仅是给它一个名字而已。

由于XML允许定义自己的标签，制作者定义的标签和其他人定义的标签有可能相同，但表示不同的意义。当文件交换或者共享的时候就容易产生错误。为了避免这种错误发生，XML采用名字空间声明，允许通过一个网址指向来识别制作者的标签。

XHTML是HTML向XML的过渡，因此它需要符合XML文档规则，因此也需要定义名字空间。又因为XHTML 1.0不能自定义标签，所以它的名字空间都相同，就是“http://www.w3.org/1999/xhtml”。

然而，w3.org的校验器不会由于这个属性没有出现在要校验的XHTML文档中而报告错误。这是因为“XMLns=http://www.w3.org/1999/xhtml”是一个固定的值，即使文档里没有包含它，它也会自动加上。

3. 定义语言编码

为了被浏览器正确解释和通过标识校验，所有的XHTML文档都必须声明它们所使用的编码语言。代码如下：

```
<meta http-equiv="Content-Type" content="text/html; charset=GB2312" />
```

这里声明的编码语言是简体中文GB2312。

4. XHTML元素一定要被正确地嵌套使用

在HTML里一些元素可以不正确嵌套也能正常显示，如右：

```
<span><strong>这里是内容</span></strong>
```

而在XHTML必须要正确嵌套之后才能正常使用，如右：

```
<span><strong>这里是内容</strong></span>
```

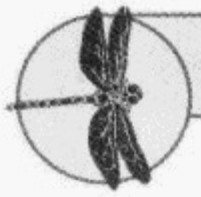
这个错误通常发生在当嵌套多层之后的标签里面，如：

错误	正确
  - 设计软件 - 制作软件 - Dreamweaver - GoLive - 编程软件 	 - 设计软件 - 制作软件 - Dreamweaver - GoLive - 编程软件

5. XHTML 文件一定要有正确的组织结构

所有的XHTML应该正确地嵌套在以<html>开始以</html>结束的元素里面，其他的元素可以有子元素，并且子元素也要被正确地嵌套在它们的父元素内。如右：

```
<html>
 <head> ... </head>
 <body> ... </body>
</html>
```



注意： <html>、<head>和<body>元素必须出现，并且<title>元素必须在<head>元素里。

6. 标签名字一定要用小写字母

因为XHTML文档是XML的一种，而XML对大小写是敏感的，像 和<P>是两个不同的标记，如：

错误	正确
<BODY> <P>XHTML大小写敏感。</P> </BODY5>	<body> XHTML大小写敏感。 </body>

7. 所有的XHTML元素一定要关闭

在代码中不能有没有关闭的空元素，特别是
、<img>、<input>这样的空标签。

空标签表示这个标签没有结束标签，而不像其他标签那样成对出现，例如，<img>是一个空标签，因此<img></img>是错误的，空标签在最后加入“/”来表示结束。例如：

错误	正确
关闭标签了吗？ 真的关闭标签了吗？ 关闭标签了吗？ 	关闭标签了吗？ 真的关闭标签了吗？ 关闭标签了吗？

续表

错误	正确
<img src="img/cat.jpg">	

8. 属性名字必须小写

属性和标签的要求一样，要小写。如：

错误	正确
<table **WIDTH**="100%">	<table width ="100%">

9. 属性值必须带上英文双引号

属性的值需要用英文双引号“”括起来，如：

错误	正确
<table width=**100%**>	<table width ="**100%**">

10. 属性的简写被禁止

在HTML中某些属性可以简写，但是在XHTML中不能简写。在HTML中简写的属性和其在XHTML中书写的规范，如：

HTML	XHTML
checked	checked="checked"
disabled	disabled="disabled"
selected	selected="selected"
noresize	noresize="noresize"

11. 用id属性代替name属性

HTML 4.01中为<a>、<applet>、<frame>、<iframe>、<img>和<map>定义了一个name属性，在XHTML里除了表单<form>和链接<a>外，name属性不能使用，应该用id来替换。如：

错误	正确
	

12. 内容文字注意转义

由于HTML的标签是以“<>”来表示的，因此“<>”不能直接出现在内容文字中，而需要转义，即使用其他符号来代替会引起问题的符号，例如右边代码：

```
<p>段落<p>元素</p>
```

出现在内容中的“<”和“>”符号应该如右进行转义：

```
<p>段落&lt;p&gt;元素</p>
```

浏览器会将“<”和“>”还原为“<”和“>”显示出来。HTML中比较常见的需要转义的符号如下表所示。

英文字符	转义序列	注解
&	&amp;	和
<	<	小于号
>	>	大于号
"	"	双引号
		空格

续表

英文字符	转义序列	注解
©	©	版权符
®	&reg	注册符

另外，HTML使用的字符集是ISO &859 Larin-1字符集，该字符集中有许多标准键盘上无法输入的字符。对这些特殊字符只能使用转义序列，转义序列的书写需要注意以下几点：

- 转义序列各字符间不能有空格；
- 转义序列必须以英文“;”结束；
- 单独的“&”不被认为是转义开始；
- 区分大小写。

13. xml:lang属性和lang属性

lang属性可以应用于几乎所有的XHTML元素，它指定了元素中内容的语言属性。如果网页定义为XHTML 1.1或者XML格式，那么要使用xml:lang属性，因为xml:lang属性是在XML中确定语言信息的标准用法。

如果网页使用HTML格式，那么应该同时使用xml:lang和lang属性。如：

```
<div lang="no" xml:lang="no">Heia Norge!</div>
```

此属性常用于<html>标签中，代表整个文档使用了某种语言，例如：

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

xml:lang属性可以使搜索引擎了解页面使用了何种语言，搜索引擎可以按语言把页面归类，或者启动某些自动翻译系统。xml:lang属性也可以使排版工具了解你的页面使用了何种语言，这样相应的排版工具就可以进行切换标点符号、转换格式等操作。

语言属性的重要性在浏览器里容易被忽略，可是对语音辨识阅读器就很重要。若没有语言提示，阅读器就会遇到障碍，无法决定下段文字该用什么语言念出。

2.3 理解（X）HTML标签的语义

结构是网页的基础，因此构建一个合理的结构是页面制作的第一步。要做到合理地结构化，首先要对（X）HTML有比较深入的了解，在充分理解XHTML标签“语义”的基础上才能顺利地将内容结构化。

2.3.1 （X）HTML与浏览器默认样式

（X）HTML元素（Element）构成了（X）HTML文件，这些元素是由（X）HTML标签（tag）所定义的，如图2-5所示。

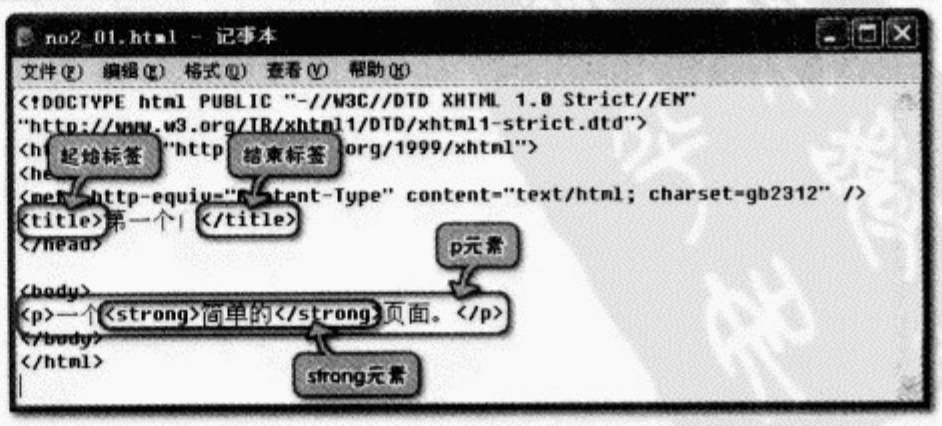
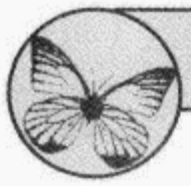
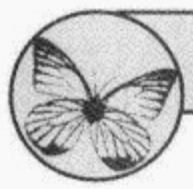


图2-5 （X）HTML标签与元素



提示：读者可以参见下载文件包内［/第1部分/第2章：结构与表现/html_01.html］文件。

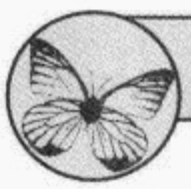
当用户使用浏览器浏览网页的时候，实际存在着3个样式：制作者（作者）设定的样式、浏览器默认样式和用户（访问者）设定的样式。默认状态下，制作者的样式表规则较用户样式表规则优先级高，而这两者都高于浏览器默认样式。



提示：关于样式的优先级，可参见本书［4.6 层叠］一节。

当制作者和用户都不设定页面样式的时候，浏览器使用自己的样式表，遵循Web标准的浏览器，会参照W3C规范来制定自己的样式表，如<p>标签表示“段落（Paragraphs）”，其间的内容会显示在一个新行内，而标签表示“强调”，则在标签之间的内容被加粗显示，这是浏览器预定的显示方法——浏览器的默认样式。

不同的浏览器的默认样式也会略有不同，如图2-6所示。



提示：读者可以参见下载文件包内［/第1部分/第2章：结构与表现/html_02.html］文件。

而正是由于浏览器默认样式的存在，往往会造成制作者的迷惑，认为某些标签就只能是浏览器显示的那种样子，例如，只能是粗体显示，<h1>则总是显示很大的字体，其实，制作者完全可以通过设定CSS样式来改变它。

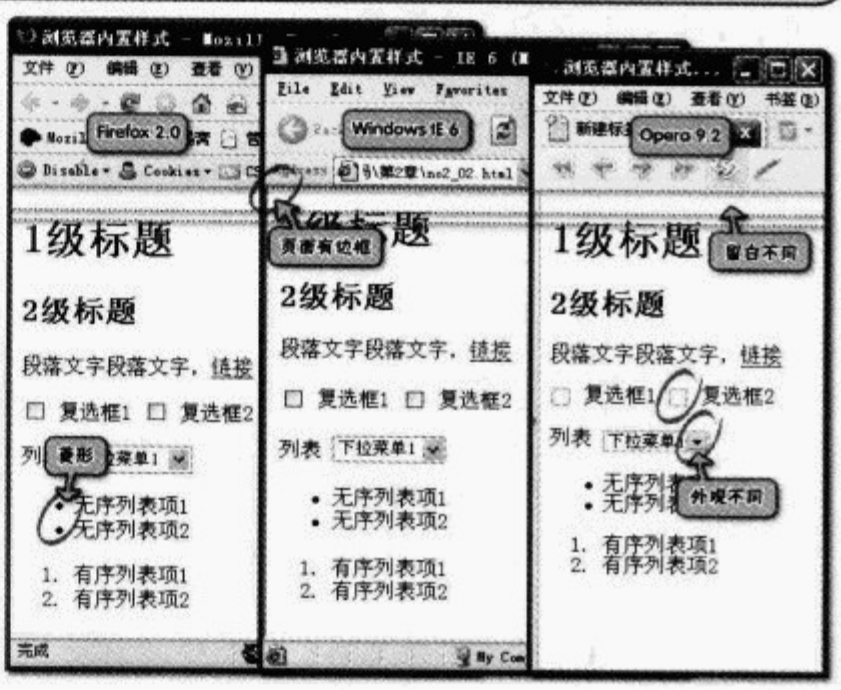
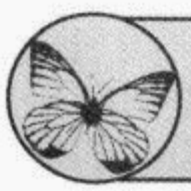


图2-6　不同浏览器之间默认样式的差异

2.3.2 常用的XHTML标签和属性

XHTML的标签都是有语义（Semantics）的，一个XHTML元素的存在就意味被标记内容的那部分有相应的结构化的意义，例如，<p>标签表示是“段落”，<ul>标签表示“无序列表”，而<table>标签表示“表格”。因此，要制作一个良好结构的页面，就要按照内容来选择适合的XHTML标签，而不是根据外观来区分，比如不应该用 来代替<h1>标记标题。

语义化对网页的好处，最主要的就是对搜索引擎友好，有了良好的结构和语义，网页内容自然容易被搜索引擎抓取，网站的推广便可以省下不少的功夫。具体的XHTML标签的语义和属性在此不再赘述，下面将一些容易遗忘或者混淆的标签和属性予以说明。



提示：读者可以访问中文网站“W3school”（http://www.w3school.com.cn/）或者英文网站“W3Schools”（http://www.w3schools.com/），更详细地学习XHTML的知识。

1. 6级标题

<h1>、<h2>、<h3>、<h4>、<h5>、<h6>作为标题使用，并且依据重要性递减。<h1>是最高的等级。例如：

```
<h1>文档标题</h1>
<h2>次级标题</h2>
```

搜索引擎会注意到标题标签内的内容，因此不要使用其他标签来替代标题，例如：

```
<div class="title">文档标题</div>
<span class="title">文档标题</span>
```

很明显，对于上述代码，搜索引擎不会认为它是标题。

2. <p>

段落（Paragraphs）标签<p>是处理文字时经常用到的标签，大段的文字一般应该使用<p>标签，例如：

```
<p>语义化对网页的好处，最主要的就是对搜索引擎友好，有了良好的结构和语义，网页内容自然容易被搜索引擎抓取，网站的推广便可以省下不少的功夫。</p>
<p>具体的HTML标签的语义和属性在此不再赘述，下面将一些容易遗忘或者混淆的标签和属性予以说明。</p>
```

同时，**不建议**使用
换行，更不需要用多个
来区分段落与段落。<p>标签中的文字会自动换行，而且换行的效果优于
。段落与段落之间的空隙也可以利用 CSS来控制，很容易而且清晰地区分出段落与段落。同时还可以通过设定行高（line-height）很容易地改变文字的行间距，还可以定义首行文字缩进（text-indent）、文字对齐方式（text-align）等属性。

3. 、<ol>和

（Unordered Lists，无序列表）可以用于导航条，当浏览器不支持CSS的时候，导航仍然很清晰明了，虽然美观方面差了一点。例如：

```
<ul>
 <li><a href="http://www.ddcat.net/blog/">首页</a></li>
 <li><a href="http://bbs.ddcat.net">论坛</a></li>
 <li><a href="http://www.ddcat.net/blog/index.php?do=gb">留言板</a></li>
 <li><a href="http://www.ddcat.net/blog/rss.php">RSS 1.0</a></li>
 <li><a href="http://www.ddcat.net/blog/rss2.php">RSS 2.0</a></li>
</ul>
```

其效果如图2-7所示。

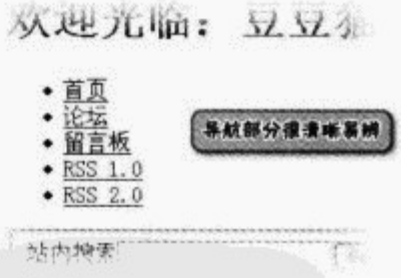


图2-7 浏览器内显示未设定CSS样式的导航列表

有序列表（Ordered Lists）顾名思义就是有先后顺序，因此<ol>可以使用在排行榜或者章节列表等地方，例如：

```
<h4>文章点击排行</h4>
<ol>
 <li>CSS图片自动等比例缩小且垂直居中</li>
 <li>上中下三行布局，上下定高，中间栏自适应</li>
 <li>文本水平对齐（text-align）</li>
</ol>
```

无论是<ul>还是<ol>，其内的列表项都是<li>，而<li>是不能脱离<ul>或者<ol>单独存在的，此点需要特别注意。在<ul>（<ol>）和<li>之间不能插入其他的内容（例如文字或者其他标签）。例如右边的代码是错误的：

```
<ol>
 <h4>文章点击排行</h4>
 <li>CSS图片自动等比例缩小且垂直居中</li>
 <li>上中下三行布局，上下定高，中间栏自适应</li>
 <li>文本水平对齐（text-align）</li>
</ol>
```

可以通过设定CSS样式来控制是否显示列表符号，以及列表符号以何种样式显示。

4. <dl>、<dt>和<dd>

释义列表（Definition Lists）是一列事物以及与其相关的解释。释义列表的开始标签是<dl>，每个被解释的事物的开始标签为<dt>，每个解释的内容的开始标签是<dd>。在<dd>标签中的内容可以是段落、回车符、图像、连接或者是其他的列表等。例如，英汉词典里面的词的解释就可以用释义列表，如右所示：

```
<dl>
 <dt> presentation</dt>
 <dd> n. 介绍, 陈述, 赠送, 表达</dd>
 <dt>structure</dt>
 <dd> n. 结构, 构造, 建筑物</dd>
 <dd> vt. 建筑, 构成, 组织</dd>
</dl>
```

释义列表的应用很广泛，但是也不要拿它当作<table>的替代品而用来排版布局，语义永远是第一位的。

5. 和

标签是强调，标签是重点强调。大部分浏览器用斜体显示强调的内容，用粗体来显示重点强调的内容，然而使用斜体显示汉字将使阅读变得困难，因此可以通过定义CSS来改变和的表现。但是，如果只是想要视觉上的粗体效果，而并不是要强调内容的重要性，就不要使用这两个标签。例如：

```
<p><em>强调</em> 的文本通常用斜体显示，而<strong>特别强调</strong> 的文本通常以粗体显示。</p>
```

<b>标签也是以粗体显示文字，但是，这明显是一个包含“表现”的标签，因此不建议使用。

6. <table>、<td>、<th>、<caption>和summary属性

XHTML中的表格不应该用来布局，但是如果是为了显示表格类的数据，就应该使用表格。

- <table>标签为表格总体定义用，其内包含下面的标签。
- <caption>标签为表格标题，<caption>标签必须紧随<table>标签之后，而且每个表格只能定义一个标题，在浏览器内通常这个标题会被居中显示于表格之上。
- <tr>标签为表格行，<tr>内包含<th>和<td>标签。
- <th>标签为表头单元格，浏览器默认以粗体显示。
- <td>标签为单元格。
- <thead>标签为表格头部，其内必须有<tr>标签。
- <tbody>标签为表格主体内容，其内必须有<tr>标签。
- <tfoot>标签为表格注脚，其内必须有<tr>标签。
- summar属性为摘要。

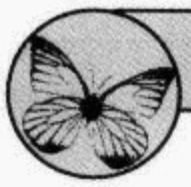
还可以使用scope属性可用于取代headers属性，标记含有表头信息的单元格，其中各数值的内容如下。

- row指示当前单元格，为包含当前单元格的行提供相关的表头信息。
- col指示当前单元格，为根据当前单元格指定的列提供相应的表头信息。
- rowgroup指示当前单元格，为包含当前单元格的其余行组提供相关的表头信息。
- colgroup指示当前单元格，为根据当前单元格指定的其余列组提供相应的表头信息。
- abbr用于定义表头单元格中的缩写名，如果没有定义该属性，则将默认单元格内容为节略形式。

例如：

```
<table id="report" summary="一班学生成绩单">
 <caption>
  成绩单
 </caption>
 <thead>
  <tr>
   <th scope="col">学号</th>
   <th scope="col">姓名</th>
   <th scope="col">语文</th>
   <th scope="col">数学</th>
```

```
</tr>
</thead>
<tbody>
<tr>
 <th scope="row">1</th>
 <td>张三</td>
 <td>80</td>
 <td>85</td>
</tr>
<tr>
 <th scope="row">2</th>
 <td>李四</td>
 <td>85</td>
 <td>89</td>
</tr>
<tr>
 <th scope="row">3</th>
 <td>王五</td>
 <td>90</td>
 <td>94</td>
</tr>
</tbody>
<tfoot>
<tr>
   <td colspan="4">共3名学生</td>
</tr>
</tfoot>
</table>
```



提示： 读者可以参见下载文件包内［/第1部分/第2章：结构与表现/ table.html］文件。

使用<th>标签来标示表头，就不需要使用类似右边的代码：

```
<td><strong>学号</strong></td>
```

<thead>、<tfoot>以及<tbody>标签使得制作者可以对表格中的行进行分组。当创建表格时，这种划分使浏览器有能力支持独立于表格标题和页脚的表格正文滚动。当比较长的表格被打印时，表格的表头和页脚可被打印在包含表格数据的每张页面上。不过，需注意的是，这3个标签必须同时出现在<table>内，这样浏览器可以在收到所有数据前呈现页脚。

而<tbody>标示出了表格的主体部分，当表格很长时，可以使用多个<tbody>标签来分隔表格内容，浏览器会先显示已下载完的<tbody>内的内容。

7. <q>、<blockquote>、<cite>和cite属性

论坛和博客经常会用到引用别人的话，用<q>标签来标记简短的单行引用。除IE以外的其他Web浏览器会自动识别在<q>之间的内容，为其添加引号来标示此处为引用，例如有HTML代码如下，则其在浏览器内显示如图2-8所示。

```
<p>《盒模型（BOX Model）》一文中指出：<q cite="http://www.ddcat.net/blog/ archives/2007/02/ 138.html">实际宽度 = 左边界 + 左边框 + 左填充 + 内容宽度（width）+ 右填充 + 右边框 + 右边界</q></p>
```

不过<q>标签可能会引起一些易用性的问题，正因为如此，有人建议不使用<q>，而是手动地插入引用标记，例如，在一个包含适当的类的中加入单行的引用内容，然后用CSS来控制样式，但是这个没有语义上的意义。

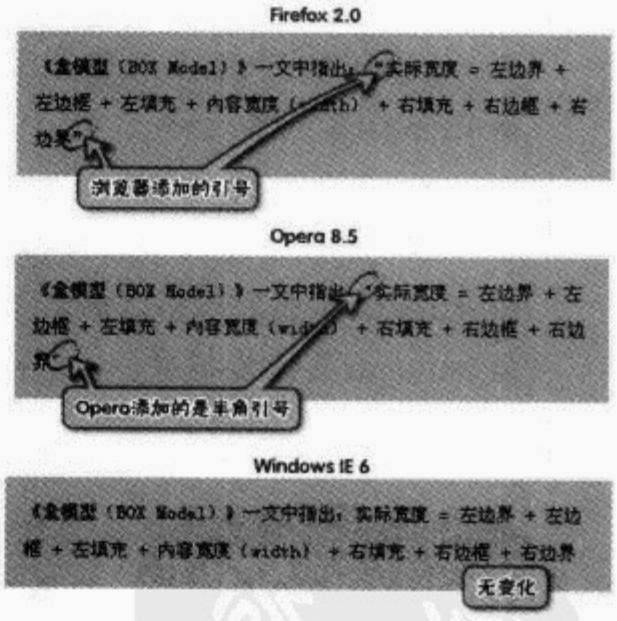
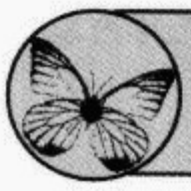


图2-8 <q>标签在不同浏览器中的表现差异



提示： 读者可以参见Mark Pilgrim写的《The Q tag》（英文，http:// diveintomark.org/archives/2002/05/04/the_q_tag）关于处理<q>相关问题的文章。

对于那些一段或者好几段的长篇引用，就应当使用<blockquote>标签。一段文章不可以直接放在 <blockquote>中，引用的内容还必须包含在一个块级元素中，通常是<p>。

例如下列代码在浏览器内显示如图2-9所示。

```
<blockquote cite="http://www.ddcat.net/blog/archives/2007/02/138.html">
    <p>而对于Windows IE 5.x及更前的版本，把这个盒模型的定义搞错了，它认为：宽度（width）= 元素内容 + 填充 + 边框</p>
    <p>这个确实很容易搞错，很多对于盒模型定义没有深入了解的人也同样容易犯这个错误。</p>
</blockquote>
```

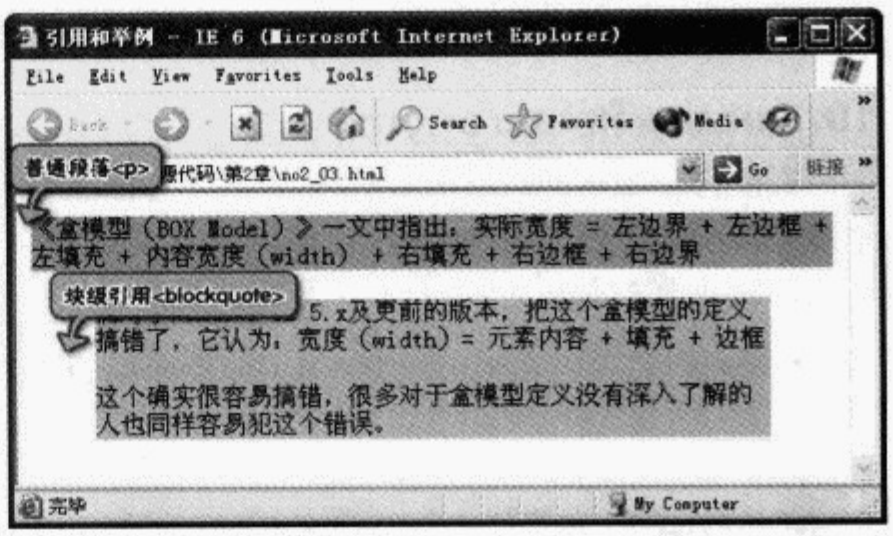
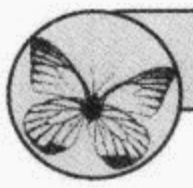


图2-9 <blckquote>在IE 6.0内的表现



提示：读者可以参见下载文件包内［/第1部分/第2章：结构与表现/ blockquote.html］文件。

属性cite既可以与<q>标签一起用，也可以与<blockquote>一起用，用来提供引用内容的来源地址。但是如果使用来代替<q>标记引用内容,那么就不能使用cite属性。

<cite>作为标签，是举例的意思，例如右边代码，<cite>元素在浏览器内默认用斜体显示。

```
<p><cite>《别具光芒》</cite>是本书作者编写的一个网页制作类书籍系列。</p>
```

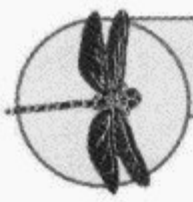
8. <ins>和<del>

<ins>标签表示插入的内容，而<del>标签则表示已被删除的内容，这两个标签搭配使用，来描述文档中的更新和修正。这两个标签还可以添加cite属性和datetime属性来表明删除的原因以及删除的时间。ins是表示插入，也有这样的属性。

例如有HTML代码如下，其在浏览器内显示如图2-10所示。

图2-10 ins元素和del元素在浏览器内的显示

```
<p>absolute：将元素从文档流中拖出，进行绝对定位。<del datetime="20070803">此时元素不具有外边距</del><ins datetime="20070803" cite="http://www.ddcat.net/blog/archives/2006/08/121.html">元素可以有外边距，但这些外边距不压缩</ins>，还可以有补白和边框。</p>
```



注意：HTML 4.01中的<s>标签也表示为删除线，但是就语义来讲，还是应该使用<del>标签。

9. <dfn>、<code>、<samp>、<kbd>、<var>和<address>

- <dfn>定义一个定义项目。
- <code>定义计算机代码文本。
- <samp>定义抽样的计算机代码。
- <kbd>定义键盘文本。
- <var>定义变量。
- <address>定义一个地址，可以使用它来定义地址、签名或者文档的作者身份。

这些标签使用的都比较少，大部分在浏览器内都会默认显示为斜体，这些标签虽然不是必须

的，但是灵活运用不仅让内容结构更加清晰，还可以通过设定CSS来丰富页面的表现力。

10. <abbr>和<acronym>

<abbr>标签是表示Web页面上的简称，<acronym>标签为取首字母缩写。这两个标签看上去似乎没有特别的用处，但是对网站的亲和力起到很重要的作用。通过给缩写文字加上相应标签，并且添加title属性和设定CSS样式，可以让用户更加容易理解缩写的含义，例如：

```
<abbr title=" Extensible HyperText Markup Language">XHTML</abbr>
<acronym title="Cascading Style Sheets">CSS</acronym>
```

注意：这里把简称和缩写分开而论，简称范围比缩写大，取首字母的缩写用<acronym>标签。

Windows的IE 6.0以下的浏览器暂不支持<abbr>标签。在IE里，可以为<acronym>标签设定CSS，但是不能对<abbr>标签设定CSS，IE会为<acronym>标签的title属性显示提示，但是会忽略<abbr>标签。因此，可以通过一些变通的办法来解决这个问题，例如：

```
<abbr title=" Extensible HyperText Markup Language "><span class="abbr">XHTML</span></abbr>
```

通过增加一个来设定CSS。

11. alt属性和title属性

title属性用来为元素提供额外说明信息，当鼠标指向具有title属性的元素时，会在鼠标下出现title内设定的文字，如图2-11所示。

```
<p>title属性可以加在<a href="#" title="链接的
title">链接</a>和<strong title="特别强调的title">
其他一些标签</strong>内</p>
```

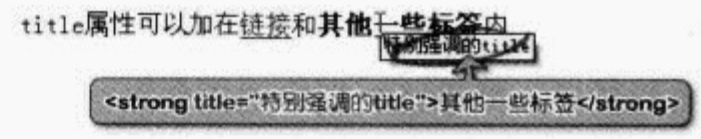


图2-11 title属性在浏览器内的显示

注意：title属性可以用在除了<base>、<basefont>、<head>、<html>、<meta>、<param>、<script>和<title>之外的所有标签，但并不是必须的。读者可以参见下载文件包内［/第1部分/第2章：结构与表现/ title.html］文件。

alt属性为不能显示图像、窗体或applets的用户端指定替换文字，替换文字的语言由lang属性指定，如图2-12所示。由图2-12可以发现，在IE内，当图片没有设定title属性时，会在鼠标下显示alt属性内的文字，而Firefox等浏览器则不显示。而当同时设定了alt和title属性时，则IE会显示title属性内的文字，如图2-13所示。

图2-12 alt属性在浏览器内的显示

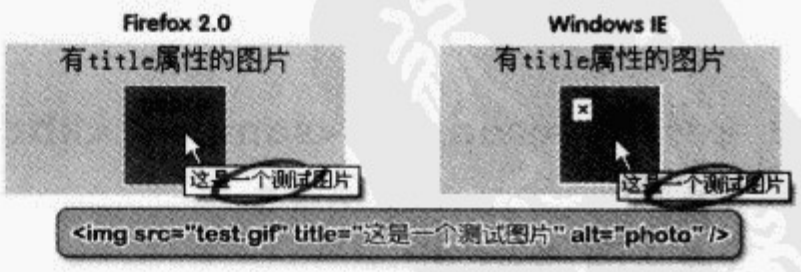
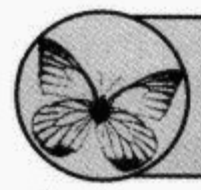


图2-13 title属性在浏览器内的显示



提示：为链接、缩写等一些元素添加title属性，将会增加网页的易用性，方便访问者浏览网站内容。

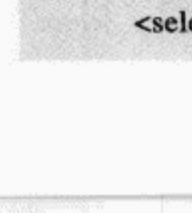
12. 表单

表单是实现同用户交互的很重要的手段，登录网站、搜索、提交调查等都需要使用各种表单来完成。表单包括以下几个标签。

- <form>：定义供用户输入的表单
- <input>：定义输入域
- <textarea>：定义文本域（一个多行的输入控制）
- <select>：定义一个选择列表
- <optgroup>：定义选项组
- <option>：定义下拉列表中的选项
- <button>：定义一个按钮
- <label>：定义一个控制的标签
- <fieldset>：定义域
- <legend>：定义域的标题

图2-14　<fieldset>和<legend>标签的作用

其中，<fieldset>和<legend>标签可能读者很少会用到，其实这两个标签在提高表单的易用性方面有很大的用处。<fieldset>和<legend>可以将表单内容分类，并且通过标题来告诉用户表单的内容，例如下列的代码在浏览器内显示如图2-14所示。



提示：读者可以参见下载文件包内［/第1部分/ 第4章/from.html］文件。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="zh-CN" xml:lang="zh-CN">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>表单 :: 常用的XHTML标签和属性 :: 结构与表现</title>
</head>

<body>
<form id="test_form" method="post" action="#">
 <fieldset>
  <legend>用户基本信息</legend>
  <label for="nikename">昵称：</label>
 <input name="nikename" id="nikename" type="text" size="16" maxlength="30" />
  <label for="age">年龄：</label>
  <input name="age" id="age" type="text" size="6" maxlength="8" />
    <label>密码：</label>
    <input name="password" id="password" type="password" value="" size="16" />
  <input type="radio" name="sex" id="sex_male" value="male" />
  <label for="sex_male">男</label>
  <input type="radio" name="sex" id="sex_female" value="female" />
  <label for="sex_female">女</label>
 </fieldset>
 <fieldset>
  <legend>用户联系信息</legend>
  <label for="address">地址：</label>
  <input name="address" id="address" type="text" size="32" maxlength="100" />
    <label for="postalcode">邮政编码：</label>
 <input name="postalcode" id="postalcode" type="text" size="6" maxlength="6" />
  <label for="phone">电话：</label>
  <input name="phone" id="phone" type="text" size="11" maxlength="11" />
  <label for="locus">所在地：</label>
  <select name="locus">
```

```
    <optgroup label="直辖市">
     <option value="1">北京</option>
     <option value="2">天津</option>
     <option value="3">上海</option>
     <option value="4">重庆</option>
     </optgroup>
     <optgroup label="省">
     <option value="5">河北省</option>
     <option value="6">广东省</option>
       </optgroup>
    </select>
   </fieldset>
   <input type="submit" name="btn_submit" value="提交" />
   <input type="reset" name="btn_reset" value="重置" />
</form>
</body>
</html>
```

<label>标签也一直被忽视了，其实使用<label>标签不仅可以增加表单的易用性，还可以用来控制简单的表单布局。<label>标签有以下两种使用方法。

● 使用<label>标签环绕表单元素，例如：

```
<label><input name="nikename" id="nikename" type="text" size="16" maxlength="30"/></label>
```

● 将说明性的文字用<label>标签环绕，然后使用for属性来指向其关联的表单元素，for属性的值等于表单元素的id属性的值例如：

```
<label for="nikename">昵称：</label>
<input name="nikename" id="nikename" type="text" size="16" maxlength="30" />
```

第2种方法的好处在于，如果用户在<label>元素内点击文本，浏览器就会自动将焦点转到和标签相关的表单控件上。例如单击上例中的文字“昵称”，光标就会自动跳到其后面的文本框内。对于单选按钮和多选按钮这样比较小的表单元素，使用<label>标签可以使用户更容易点选。

<optgroup>标签用来将下拉列表框<select>内的选项<option>分组，当选项比较多的时候，分组可以使用户更容易找到选项。使用<optgroup>标签必须设定label属性，例如：

```
<optgroup label="直辖市">
```

图2-15 <optgroup>标签的作用

上例中的下拉列表框在浏览器内显示如图2-15所示。

由于表单是同用户交互的主要途径，因此保证表单的易用性尤为重要。在很多情况下，需要辅助以脚本程序来实现更好的客户体验。

13. 和

和是不具备特别语义的标签，多数情况下当作“容器”或者“分割”来使用。

默认是块级（block）元素，在默认情况下，对于块级（block）元素浏览器会自动换行，每个块级元素都在新的一行内显示。在制作的时候，可以使用 将互相有联系的内容结构包含起来，例如：

```
<div>
 <h2>结构化</h2>
  <p>网页的结构化就是将没有格式的内容用HTML或者xHTML标签分割成不同的部分，使内容更加具有逻辑性，易用性，同时使内容可以由CSS来控制表现形式。</p>
</div>
```

可以通过设定 的id属性或者class属性，使用CSS对其进行定位、美化等处理。大部分的页面的总体布局，也是使用 的嵌套来完成的。同时，当要处理比较复杂的设计的时候，可

能也需要嵌套1个或多个 来完成，如圆角矩形。

默认是行内（inline）元素，即元素后不换行，几个inline元素可以在同一行内显示，如<a>、、等。关于元素的类型，请参见本书［3.4 元素类型］一节。

和的一个特点在于，浏览器对其没有预设的样式。

2.3.3 （X）HTML各个元素对搜索引擎的权重比例

搜索机器人对于页面内的元素的重视程度各不相同，最普遍的评分标准如下。

- 内部链接文字：10分
- <head>内的<title>元素：10分
- 域名：7分
- <h1>、<h2>标题：5分
- 每段首句：5分
- 路径或文件名：4分
- 相似度（关键词堆积）：4分
- 每句开头：1.5分
- 加粗或斜体：1分
- 文本用法（内容）：1分
- title属性：1分（注意不是<title>标签，是title属性）
- 图片的alt属性：0.5分
- Meta描述（description属性）：0.5分
- Meta关键词（keywords属性）：0.05分

在制作页面的时候，合理地安排使用（而不是滥用）这些标签和内容，可以提高页面被搜索引擎收录的概率。

2.4 网站整体制作基本流程

一个良好的网站制作流程，可以使网站制作者的工作效率大大提高。

2.4.1 总体流程与分工

一般的网站制作可以按以下流程来进行。

1. 前期策划

无论是大的门户网站还是只有几个页面的个人主页，都需要做好前期的策划工作，特别是当客户本身对于网站的需求不明确的时候，承接网站制作的一方更要做好充分的前期策划工作。

网站虽然看起来是一些电子文件，但是这些文件同网站的功能以至于服务器的操作系统都有着密切的关系。同客户明确网站主题、栏目设置、整体风格、所需要的功能及实现的方法，甚至于域名的申请、虚拟主机或服务器的购买、开发制作的周期以及后期的维护等细节及报价，是制作一个网站的良好开端。

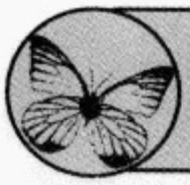
2. 项目细化及实施

在前期策划案得到认可后，就需要将项目细化分为两个部分进行，一个是前台页面设计制作，一个是后台程序及功能实现（当然，如果有这种需求的话）。

客户一般都是“以貌取人”的，往往要求制作方迅速提供一到两套设计图，而对于网站的实际功能究竟如何，却不甚了了。所以，就算网站有后台程序部分，而最早进入到实际制作流程的，往往是美工设计。

事实上，如果有一个好的网站策划与分工，后台程序可以和美工设计共同开始甚至于先于美工设计开始。而到实施部分，实际的工作分工又可分为以下3个部分。

（1）页面美工设计。美工设计人员应该在网站策划阶段就同客户充分接触，以了解客户对网站设计的需求及其个人品位，以便在设计过程中有一个基调，而提高设计稿的被认同率。



提示：设计者要勇于面对的一个现实：无论多么有“思想”有“创意”的设计稿如果得不到客户的认可，也是废纸一张。

美工首先要对网站风格有一个整体定位，包括标准字、Logo、标准色彩、广告语等。然后再根据此定位分别做出首页、二级栏目页及内容页的设计稿。首页设计包括版面、色彩、图像、动态效果、图标等风格设计，也包括Banner、菜单、标题、版权等模块设计。

一般会设计1～3套不同风格的设计稿交由客户讨论及提出修改意见，直到确定了最终方案，则按需求说明设计出所有需要的页面的设计图。

（2）静态页面制作。美工设计好各个页面的效果图后，就需要制作成HTML页面，以供后台程序人员将程序整合，制作页面的方法。

（3）程序开发。程序开发人员可以先行开发功能模块，然后再整合到HTML页面内，也可以用制作好的页面进行程序开发，但是为了程序能有很好的移植性和亲和力，还是推荐先开发功能模块然后再整合到页面的方法。

2.4.2 静态页面制作

静态页面的制作看似简单，似乎只是把设计图纸转变成可在浏览器里浏览的页面。但是如何让页面和设计图保持一致而又符合网络浏览的习惯，如何让页面既像图纸中那样美丽又有较快的速度和易用性，对于网站能留住更多的浏览者是个很关键的问题。

页面制作应该从网站策划时就要参与进来，网页内容元素的组成、它们之间的关系、如何进行交互等都应该掌握，而不是只看一个或者几个设计图。但是目前的实际状况却往往都是从设计图开始，到把设计图做成HTML文件分毫不差地在浏览器内显示为止，而真正的HTML代码结构是否合理却被忽视。但是，作为Web标准推行的目的，却是将结构的合理性、网页的易用性，放在首要位置的。

1. 提取内容

拿到一张设计图，不要立刻就用软件来划分切片和输出图片，先观察一下图纸，将内容从设计图中提取出来，虽然大部分情况下，设计图的内容只是美工设计人员随意添加的文字和图片，但是制作人员需要将它们区分出来，哪部分是新闻、哪部分是广告、哪部分是用户上传的图片等，然后用适合的（X）HTML标签来将这些内容结构化。

在结构化的同时，制作人员对页面的布局、配色、细节的处理、交互效果的实现等也就有了一个整体的认识。

2. 拆分图纸

当制作好内容结构以后，就可以将图纸拆分成需要的“原料”，以便在组装页面的时候使用，一般需要从图纸中拆分提取的有以下几个步骤。

（1）分离颜色。其中一般包括3部分配色：页面主辅颜色搭配的基本配色，普通超级链接的配色和导航栏超级链接的配色。

（2）提取尺寸。按照设计图的尺寸来搭建网页才会符合图纸上的设计，不过也不是说必须严格按照设计图来做，一点儿也不能差，有些时候可以灵活掌握。

（3）分离背景图。背景图可能是大面积重复的图案，也可能就是一张图片，一般和内容没有关系的装饰性图片都可以考虑制作成背景图。

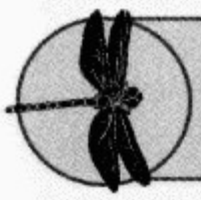
（4）分离图标及特殊边框。小图标及花边边可以给网页增添细节和亮点，框理论上讲其使用方法和背景图片类似，不过根据情况往往需要单独输出。

当把需要的图片和尺寸都准备好了，就可以着手编写CSS文件了。

3. 制作

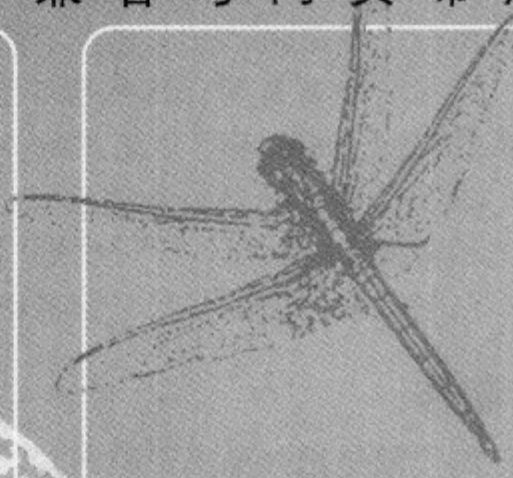
制作就是把分离出来的图片、尺寸，通过CSS（可能还要修改HTML文件）将（X）HTML文件的外观（即在浏览器内浏览）美化成与设计图中差不多的样子。

同时，如果有动态效果，例如弹出菜单、表单验证等，还需要插入JavaScript来完成。



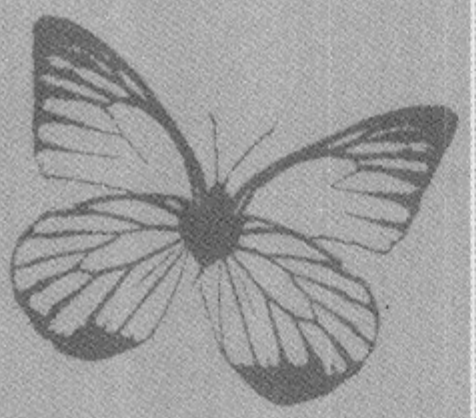
注意：这里说的是“差不多”的样子，因为很多时候设计图和实际情况会有差别，而不太可能完全一样。

因此静态页面制作的步骤一般为：构建内容的结构→提取尺寸及图片→样式表美化→细节处理→优化样式表。在本书第17章中将结合实例按照这样的步骤来制作网页。

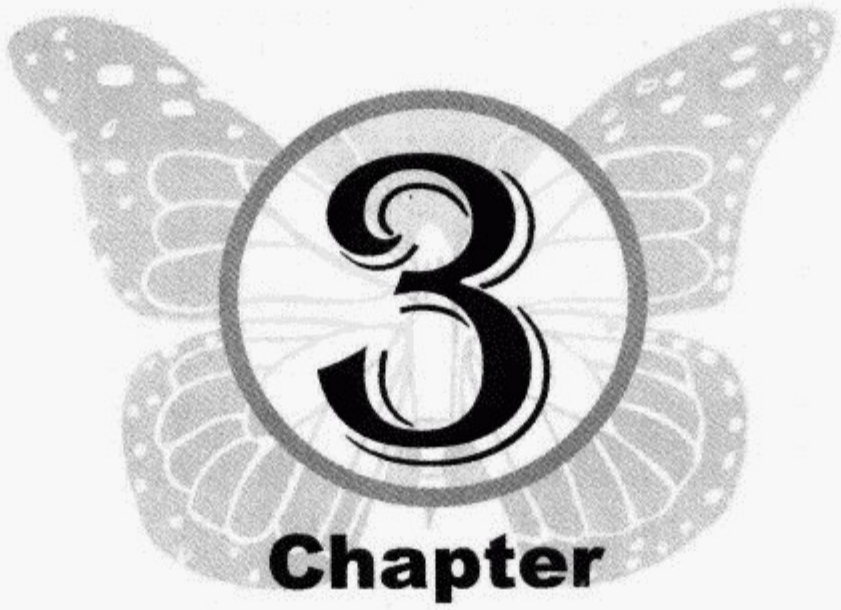


第2部分

层叠样式表CSS



别具光芒 CSS



第3章 CSS入门

层叠样式表（CSS，Cascading Style Sheet），简称样式表，是一种标记性语言。它使得网页的灵活性增加，设计者可以通过修改样式表的定义而使网页呈现完全不同的外观，而当网站拥有几十甚至上百个页面的时候，修改页面链接的样式表文件即可修改页面的外观，可以大大地减少工作量。

3.1 CSS简介

利用CSS将表现同结构分离，不仅可以精简（X）HTML文档的代码，还可以使文档清晰易读，同时，将装饰性的图片从文档内抽出，作为背景来实现，可以提高网页的显示速度。而通过编写不同的CSS规则，可以让同一个（X）HTML文档呈现出截然不同的外观。

3.1.1 起源

从1990年代初HTML被发明开始，样式表就以各种形式出现了。

不同的浏览器制定了各自的样式语言，用户可以使用这些样式语言来调节网页的显示方式。一开始样式表是给用户用的，最初的HTML版本只含有很少的显示属性，用户来决定网页应该怎样被显示。但随着HTML的成长，为了满足设计师的要求，HTML获得了很多显示功能。随着这些功能的增加外来定义样式的语言越来越没有意义了。

1994年哈坤·利提出了CSS的最初建议。伯特·波斯（Bert Bos）当时正在设计一个叫做Argo的浏览器，他们决定一起合作设计CSS。

当时已经有过一些样式表语言的建议了，但CSS是第一个含有“层叠”的概念。在CSS中，一个文件的样式可以从其他的样式表中继承下来。用户在有些地方可以使用他自己喜欢的样式，在其他地方则继承，或“层叠”制作者的样式。这种层叠的方式使制作者和用户都可以灵活地加入自己的设计，混合各人的爱好。

哈坤于1994年在芝加哥的一次会议上第一次展示了CSS的建议，1995年他与波斯一起再次展示这个建议。当时W3C刚刚建立，W3C对CSS的发展很感兴趣，并为此组织了一次讨论会。1996年底，CSS已经完成。1996年12月CSS发布第一个版本，即CSS 1（Cascading Style Sheets Level 1）。

1997年初，W3C内组织了专门管理CSS的工作组，其负责人是克里斯·里雷。这个工作组开始讨论第一版中没有涉及的问题，1998年5月，CSS第二版发布，即CSS 2（Cascading Style Sheets Level 2 Revision 1）。到本书完稿之日止，CSS 3（Cascading Style Sheets Level 3）还在开发中。CSS 3不但加入更强的功能，而且将使 CSS 更易于处理和使用。CSS 3 包含CSS 2，并加入新的选项、丰富的超文本、更多的饰边和背景、垂直文件、用户交互、语音、多媒体设备显示等。

3.1.2 神奇的CSS

要领略CSS的神奇性，读者可以访问一个名为“CSS禅意花园”（CSS Zen Garden，http://www.csszengarden.com/）的网站。设计师Dave Shea建立这个网站的目的就是让广大的网页设计师认识到CSS的重要性。网站提供一套标准的XHTML页面及CSS文件，访问者可以下载这些文件，然后自行修改CSS定义，以体现不同的设计风格，如图3-1所示。

这些风格各异的页面，如果查看源代码读者将会发现，其XHTML文件是相同的，而如此纷繁的视觉效果，只因为引用了不同的CSS文件，由此让访问者领略CSS的能力及重要性。

大部分网站，往往外观设计是统一的，只是在内容或者部分栏目上有区别，通过链接同一个样式表文件，就可以实现这种统一设计，同时，当修改样式表文件的时候，链接了这个样式表的（X）HTML页面的外观都会改变，大大减少了工作量。

同时，浏览器读入样式表文件后，将放在缓存内，其他也链接了这个样式表文件的页面将不用再读取服务器，因此减少了网络流量和服务器负担。

而对于需要在页面内加入动态效果的制作人员来说，可以通过脚本或者程序语言修改对象的CSS属性，从而更改页面显示效果，例如，可以使用JavaScript语言来控制一个层（ ）是否可见，这样就可制作可以隐藏和显示的导航菜单等。

另外，CSS扩展了原先的标记功能，能够实现更多的效果，CSS甚至超越了Web页面的本身显示功能，而把样式扩展到多种媒体上，如打印机、手机等，显示了难以抗拒的魅力。

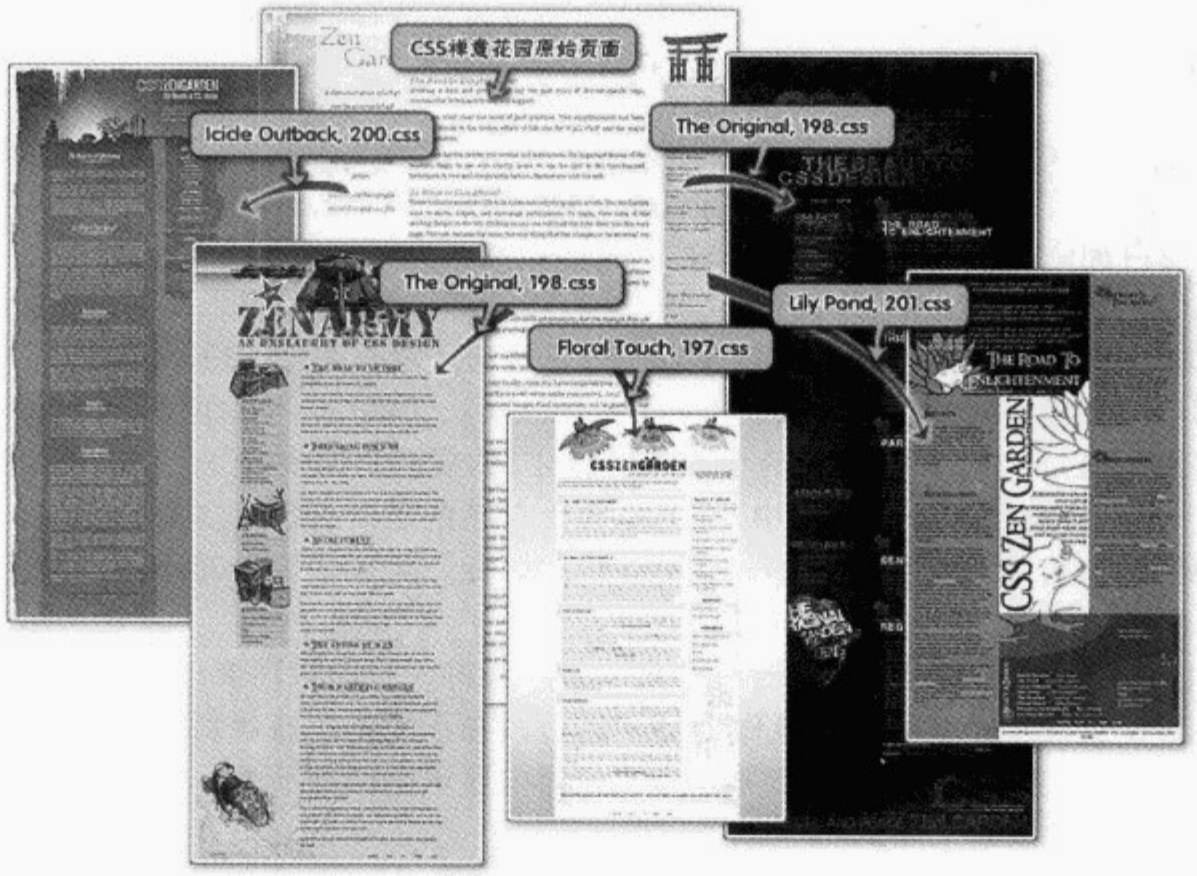


图3-1　CSS禅意花园（CSS Zen Garden）

3.1.3 CSS与HTML

CSS是对HTML 3.2以前的HTML版本语法的一次重大革新，在以前的HTML版本中，各种功能的实现是通过标记元素实现的，这也造成了各个浏览器厂商为了标新立意创建各种只有自家支持的标记，各种标记互相嵌套，可以达到不同的效果。

比如要在一段文字中把一部分文字变成红色以达到警示的目的，HTML 3.2中是这样的：

```
<p><font color="red">这里是警告！</font></p>
```

如果一个页面内有很多这样的警告文字，那么页面内就包含了很多<font>标签，而这个标签除了告诉浏览器这里的文字是红色的以外，别无他用。最让人烦恼的是，当某天设计者决定把红色的字换成蓝色的，那么制作页面的人就需要查找替换掉所有页面中所有的这些<font>标记。

而在CSS中，可以预先定义一个CSS类，然后在HTML中所有的“警告”段落只需要引用同一个样式即可，如图3-2所示。读者可以参见下载文件包内［/第2部分/第3章：CSS入门/base01.html］文件。

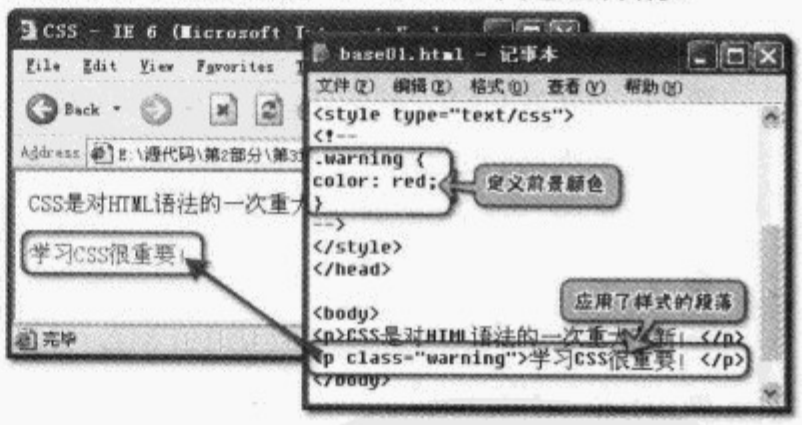


图3-2　引入CSS后HTML的改变

当修改这个类的属性时，所有引用该类的段落也会自动改变，如图3-3所示。读者可以参见下载文件包内［/第2部分/第3章：CSS入门/base02.html］文件。

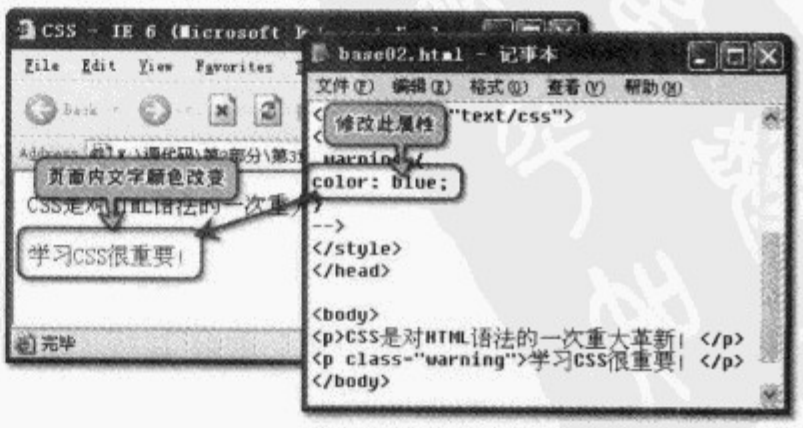


图3-3　修改CSS定义后HTML的改变

CSS简化了（X）HTML中各种繁琐的标记，使得各个标记的属性更具有一般性和通用性，特别当采用引用外部CSS文件的时候，页面的

(X)HTML可以变得非常干净和整洁。

在本书[第2章 结构与XHTML]中，读者已经了解到(X)HTML文档最理想的状态应该是页面内只有对访问者有用的内容以及一个良好的结构，而不包含任何“装饰”的成分，所有同“外表”有关系的东西都放到CSS文件中定义，即“结构和表现分离”。

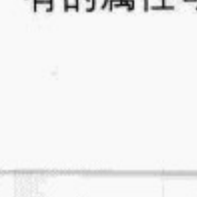
3.1.4 CSS与浏览器

CSS的功能再神奇，也是建立在浏览器或其他设备对其支持的基础上。

自从CSS 1的版本之后，W3C又发布了CSS 2和CSS 2.1，而CSS 3也正在制定中。CSS得到了更多的充实，其能控制的属性及效果更加丰富，不过，目前还没有完全支持CSS 3的浏览器。

对于CSS 2.1支持比较好的浏览器有Mozilla Firefox、Opera、Safari等，而拥有最多用户的Windows IE(包括IE 7.0)，却并没有完全支持CSS 2.1的规范，这使得CSS在的灵活应用方面大打折扣，同时，由于浏览器本身在开发中对CSS理解上存在的差异和程序本身的错误，更使得相同的CSS在不同的浏览器内可能会出现不一样的表现，因此，可能需要针对不同的浏览器来设定不同的CSS，这也增加了CSS的难度和复杂度。

为了使相同的页面在不同的浏览器内都可以显示良好，有时候需要一些技巧和方法(如CSS Hack)，但是即使这样，也有可能不能做到相同的页面在不同的浏览器看上去完全一样。不过不用烦恼，内容的易用性才是重要的，而不是在浏览器内显示得分毫不差。



提示：更多的关于浏览器与Hack的内容，请参阅本书[第16章：浏览器与Hack]。

3.1.5 CSS 2.1与CSS 2

CSS 2.1是基于CSS 2的修订版，其目的是为了替代CSS 2。CSS 2.1纠正了CSS 2中的许多错误，还增加了一些新的特性(feature)，CSS 2.1保留了大部分的CSS2的内容，修改了部分内容，并且去除了一些内容。被去除的部分，有可能在CSS 3中重新使用。

CSS 2.1的修改包括以下内容：

- 保持兼容CSS 2被广泛接受并执行的部分；
- 包含所有CSS 2勘误表；
- 按照公认的惯例修改了CSS 2中执行结果不一致的地方；
- 去除了没有被支持的属性；
- CSS 2.1的目标是使其适应HTML和XML，而不是只适应其中之一；
- 去除了被CSS 3废弃的属性；
- 根据实际需要增加了少量的新属性值。

在本书中将着重介绍CSS 2.1的属性，对于被删除的CSS 2的规则将只作简要介绍。如果浏览器支持CSS 2则可能还会支持这些被删除的属性。而支持CSS 2.1的浏览器可能不再支持这些属性。

3.2 CSS的使用方法

CSS既简单又复杂。“简单”是指其规则定义很简单，“复杂”则是指其涉及的属性范围很广泛，从文字大小、颜色到边框的样式，再到元素的定位等，而对于不同的(X)HTML元素，其中有的属性可以使用，有的则完全无效；同时，浏览器对于样式表支持的不同，使得CSS定义更加

复杂。

从样式表插入（X）HTML文件的方式来看基本可以分为**行内式样式、嵌入式样式表、外部样式表和导入式样式表**4种。

3.2.1 行内式样式（Inline Style）

行内式样式（Inline Style），即使用style属性，将CSS直接写在HTML标签中，如：

```
<p>这里是普通文字。</p>
<p style="color:red">这里文字是红色。</p>
```

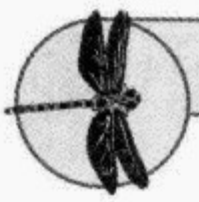
“color: red”是CSS规则，而style属性可以应用在<body>内的所有（X）HTML标签上，但是不能应用在<body>以外的标签上，如<title>和<head>等。

注意：style属性是写在HTML标签内的，因此，它使得表现与结构混在在一起，同时，修改起来也不方便，所以不推荐使用。一般可以在测试页面某部分的时候使用，例如本书以后某些例子中就会用到。

3.2.2 嵌入式样式表（Embedded Style Sheets）

嵌入式样式表（或称内部样式表）使用“<style></style>”标签嵌入到（X）HTML文件的头部（<head>）中，如：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/ xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>CSS</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css">
<!--
.warning {
color: blue;
}
-->
</style>
</head>
```



注意：<style>标签必须使用type属性，因此<style type="text/css">是固定写法，不可省略。

对于一些不能识别<style>标签的浏览器，使用（X）HTML的注释标签（<!--注释文字-->）把样式表包含起来，这样不支持<style>标签的浏览器会忽略样式内容，而支持<style>的浏览器会解读样式表。

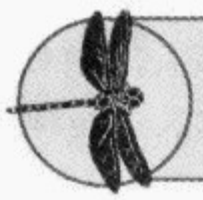
3.2.3 外部样式表（Link Style Sheets）

样式表外部文件以【.CSS】为扩展名，其实质是一个纯文本文件，可以用任何的文本编辑器来编写，文件不能包含任何的（X）HTML标签。通过在<head>内（不是在<style>标签内）使用<link>标签将样式表文件链接到（X）HTML文件内，如：

```
<head>
<title>豆豆猫的窝</title>
<link rel="stylesheet" href="basic.css" type="text/css" media="all" />
</head>
```

页面内可以同时链接多个样式表文件，例如：

```
<head>
<title>豆豆猫的窝</title>
<link rel="stylesheet" href="basic.css" type="text/css" media="all" />
<link rel="stylesheet" href="font.css" type="text/css" media="all" />
<link rel="stylesheet" href="nav.css" type="text/css" media="all" />
</head>
```



注意： CSS文件内，不需要<style>标签，也不要添加（X）HTML的注释标签，直接书写CSS声明即可。读者可以参见下载文件包内［/第2部分/samplecss/default.css］文件。

1. href属性

href属性指明链接的样式表文件的链接地址，可以是绝对地址，也可以是相对地址。例如：

```
<link rel="stylesheet" href="http://www.ddcat.net/basic.css" type="text/css" media="all" />
```

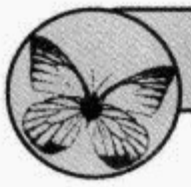
2. type属性

type属性说明包含内容的类型，一般使用type="text/css"表示为样式表。

3. rel属性

rel属性定义了文档与链接的关系，通过rel属性，可以定义多个样式表以供访问者选择，如：

```
<link rel="stylesheet" type="text/css" href="spring.css" title="春天" media="screen" />
<link rel="alternate stylesheet" type="text/css" href="autumn.css" title="秋天" media="screen" />
```



提示： 读者可以参见下载文件包内［/第2部分/第3章：CSS入门/ alternate.html］文件。

此时，如果使用Firefox浏览此页面，则可以在菜单栏［查看|页面风格］中看到title属性定义的风格列表，如图3-4所示。不过，目前只有部分浏览器支持可选择的样式表功能。

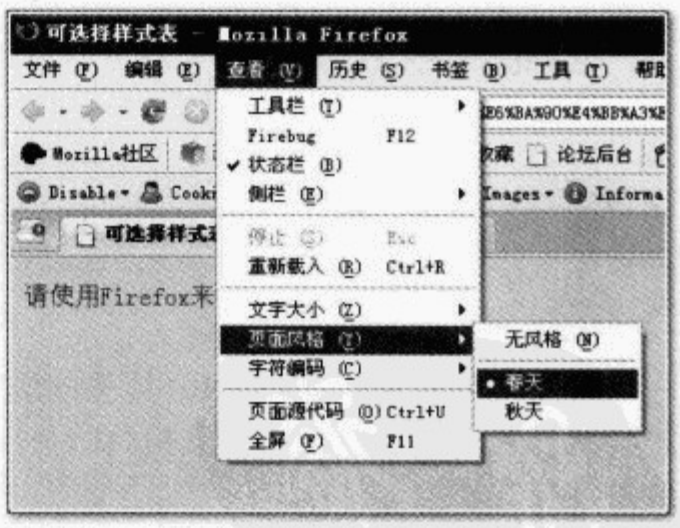


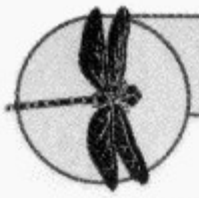
图3-4 在浏览器内选择页面样式

4. media属性

media属性为可选属性，用于指定样式表被接受的介质或媒体。允许的值有：

- screen（默认值），用于文档在屏幕媒体的呈现，如电脑显示器，所有Web浏览器都运行于这样的系统屏幕媒体用户代理；
- print，应用于不透明的页面材料，以及文档在打印预览的状态；
- projection，应用于投影演示，例如投影仪或打印到透明胶片；
- speech，应用于屏幕阅读器等发声设备；
- braille，应用于盲文触摸式反馈设备；
- embossed，用于打印的盲人印刷设备；
- handheld，用于手持设备，如个人数码助理或可上网的手机；

- tty，应用于使用固定字符宽栅格的媒介，如电传、终端或显示能力有限的手提设备；
- tv，应用于电视类型的设备（低分辨率，彩色，屏幕滚动能力有限，有声音）；
- all，所有输出设备。



注意： 媒体类型名是大小写敏感的。

目前的 Web 浏览器支持最广的是all、screen和 print，而其他的类型则很少被支持。

可以针对不同的媒体类型设定不同的CSS文件，例如针对打印机设定打印用的CSS：

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

如果要指定多个媒体，则关键字之间用英文逗号“,”隔开，例如：

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

某些CSS属性只能在特定的媒体类型内使用，例如“'page-break-before'”只能在打印机这种可以分页的媒体设备上使用。而属性的某些值在不同的媒体设备上表现可能不同，例如“font-size”设定字体的大小，在计算机显示器和打印机上有可能大小并不一样。

3.2.4 导入式样式表

导入式样式表，使用“@import”导入外部的样式表文件，它需要写在<style>标签内，如：

```
<style type="text/css" media="all">
<!--
@import url(basic.css);
@import "mystyle.css";  /* 不带url，两种写法都可以 */
-->
</style>
```

这经常用来取代连接CSS到（X）HTML中的<link>标签，这样做的好处是，一些老浏览器（如Netscape 4.x）不接受@规则，进而不连接样式表。此时如果页面拥有良好的结构，则只剩下朴素功能的（X）HTML，虽然没有华丽的外表，但是访问者仍能很好地查看内容。

@import规则也可以指定媒体类型，例如：

```
@import url(basic.css) screen, print;
```

同时，在外部CSS文件内也可以使用“@import”导入另外的样式表文件，例如在“base.css”文件内可以加入如右代码：

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

不过，用户端会忽略在规则内出现或在其他任何一个规则之后出现的任何@import规则。例如右边代码：

```
@import "base.css";
p { color: blue; }
@import "form.css";
```

根据CSS的规定，第二个@import是非法的。CSS解析器忽略整个@规则，从而有效的样式表将缩减为如右边代码：

```
@import "base.css";
p { color: blue; }
```

因此@import规则要写在样式表文件的最开始。右边这个例子中，第二个@import规则是无效的，因为它出现在一个@media规则中。

```
@import "base.css";
@media print {
 @import "print-main.css";
 body { color: red; }
}
p {color: blue; }
```

对于Windows IE，使用导入式样式表，有可能会出现文档样式暂时失效的现象，即首先显示出没有应用任何样式的文档，然后页面闪烁一次重新显示应用了样式的文档。而避免这种情况的方法是在导入样式的<style>标签前插入一个空的<script>标签，XHTML代码如右：

```
<script type="text/javascript"></script>
<style type="text/css" media="all">
<!--
@import url(basic.css);
-->
</style>
```

或者也可以使用<link>标签来引入外部样式表。

3.2.5 应用

虽然以上几种方式可以根据需要随意运用，但是在实际应用中，本着表现与内容分离的原则，作者推荐使用外部式样式表，其优点是：

- 独立于（X）HTML文件，便于修改；
- 多个文件可以引用同一个样式表文件，从而保持页面外观的统一；
- 同一个样式表文件，浏览器只需要下载一次，就可以在其他同样链接了该文件的页面内使用；
- 浏览器会先显示（X）HTML内容，然后再根据样式表文件进行渲染，从而使访问者可以更快地看到内容。

由于可以同时使用多个外部样式表，因此可以把样式分类放在不同的文件中，在不同的页面中调用不同的外部样式文件，例如某个网站的首页可以使用如下代码：

```
<link rel="stylesheet" href="basic.css" type="text/css" media="screen " /> /*页面公用样式*/
<link rel="stylesheet" href="home.css" type="text/css" media="screen" /> /*首页专用样式*/
```

而其“博客”频道，则使用：

```
<link rel="stylesheet" href="basic.css" type="text/css" media="screen " /> /*页面公用样式*/
<link rel="stylesheet" href="blog.css" type="text/css" media="screen" /> /*博客频道专用样式*/
```

而其“论坛”频道，则使用：

```
<link rel="stylesheet" href="basic.css" type="text/css" media="screen " /> /*页面公用样式*/
<link rel="stylesheet" href="bbs.css" type="text/css" media="screen" /> /*论坛专用样式*/
```

不过相对于特殊情况，也要灵活使用嵌入式样式表和行内式样式表。例如，由于浏览器先载入（X）HTML文件，后载入外部样式表文件，然后再进行渲染，因此大型网站为了防止首页加载后可能出现的无样式的情况，会对首页采用嵌入式样式表的方法。

3.2.6 维护和组织样式表

对于小型的网站，也许1个样式表文件就可以容纳所需要的样式规则，但是对于中大型网站，样式表文件可能会变得庞大无比，每个栏目也许都有自己独特的样式，而不同的开发人员，对于样式的命名以及调用都有自己的偏好，那么如何来组织这些样式表文件也成为一个很重要的问题。下面将介绍比较好的管理样式表的方法。

1. 分目录放置

将针对不同媒体的样式表放置在不同的文件夹可以利于查找和管理。例如：

（1）在网站的根目录建立【css】文件夹；

（2）在【css】文件夹内建立针对不同的媒体建立不同的文件夹，如【screen】、【print】等；

（3）将相应的文件放置在这些文件夹内。

2. 多个样式表

在一个网站中，往往不同的栏目和页面有一部分使用相同的样式，而另一部分使用不同的样式。例如，每个页面的头部导航可能是一样的，但是注册页面需要表单的样式，而普通的内容页则不需要。

按照不同的功能模块把样式放在不同的文件内，只在需要的（X）HTML文件内调用相应的文件，例如：

- 将最基本的样式规则（字体控制、颜色等）保存在【default.css】文件内；
- 将对表单元素的样式规则保存在【form.css】文件内；
- 将控制版面布局的样式规则保存在【layout.css】文件内等。

同时，可以使用@import将几个CSS文件组成1个CSS文件以方便在（X）HTML文件内调用，例如在【home.html】文件内使用外部样式表：

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

而在【base.css】文件内使用@import如右所示：

```
@import url(default.css);
@import url(layout.css);
```

3.3 基本样式规则

样式表由一些样式规则组成，浏览器将这些规则应用到相应的元素中。

3.3.1 基本语法

样式规则由“选择器（或称选择符，selector）”和“声明（declaration）”组成，而声明又由“属性（property）”和“值（value）”组成，如图3-5所示。

注意： CSS的属性（property）和（X）HTML的属性（attribute）是两个不同的概念。但是由于某些翻译原因，在很多书籍文献中将CSS的property也翻译为属性，本书为了不使读者产生混淆，沿用了此译法。读者可以参见下载文件包内［/第2部分/第3章：CSS入门/base03.html］文件。

选择器指明文档中要应用此样式规则的元素，图3-5中的样式是定义所有的<p>元素的文字将变成蓝色，而其他的元素（如<h1>）不会受影响，如图3-6所示。

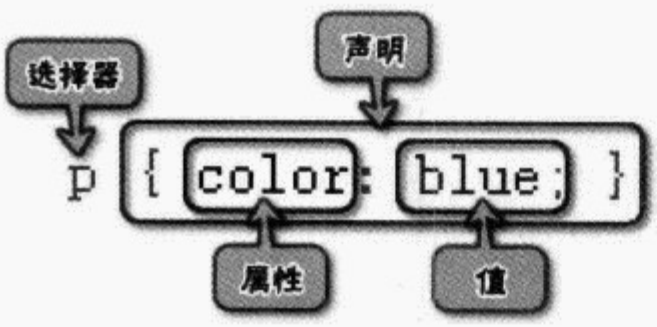


图3-5 样式规则

图3-6 选择器对元素的控制

在英文大括号“{}”中的是声明。属性和值之间用英文冒号“:”分隔。

值可以是带单位的数字、百分比或者单个关键字（keyword，如“blue”），也可以是由英文空格分开的一组关键字，例如对边框的定义如右：

```
div { border: 1px solid blue; }
```

并不是每个属性都接受多个关键字，多个关键字一般用于属性的缩写，而有的缩写则有顺序要求。

提示： 关于属性的缩写，将在允许缩写的属性内详细介绍。有些关键字不使用空格分隔，也将在具体的章节内详细介绍。

3.3.2 继承与层叠

继承和层叠是样式表的主要特点。

1. 继承

（X）HTML元素可以从其父元素那里继承部分样式规则，如图3-7所示。读者可以参见下载文件包内［/第2部分/第3章：CSS入门/base04.html］文件。

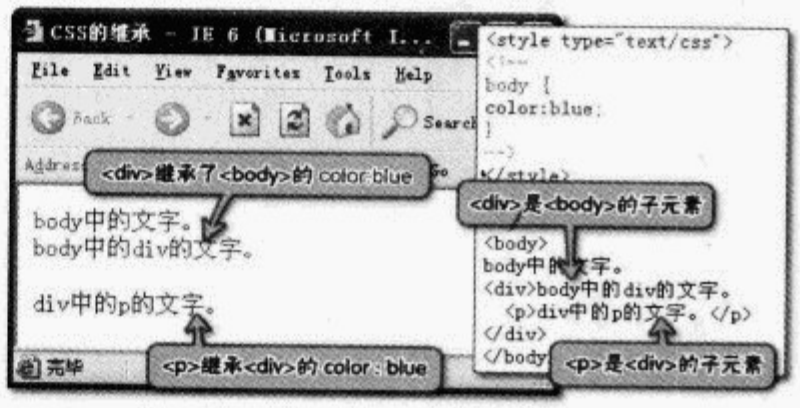


图3-7 CSS的继承

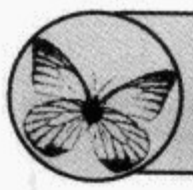
继承是很重要的特性，通过CSS继承，子元素将继承最高级元素（在本例中是<body>）所拥有的属性（如<p>、<ul>、等），不需要另外的规则，子元素的子元素也一样。通过继承，使得不用对每个元素都要定义规则，从而减少了工作量。

不过，有些属性是不被继承的，如背景色和边框等。关于CSS的继承，将在［4.5 继承］一节中详细讲解。

2. 层叠

CSS中对相同的选择器可以重复定义规则，以最后定义的为准，这就是“层叠”的含义。例如有代码如下，其在浏览器内的显示如图3-8所示。

```
p {
color: red;
background: gray;
color: yellow;
}
<p> CSS中对相同的选择器可以重复定义规则，以最后定义的为准，这就是“层叠”的含义。</p>
```



提示：读者可以参见下载文件包内［/第2部分/第3章：CSS入门/ base05.html］文件。关于CSS的层叠，将在［4.6层叠］一节中详细讲解。

图3-8 CSS的层叠

3.3.3 分组

利用分组书写规则，可以大大增加CSS的灵活性。例如，要使多个元素拥有相同的样式，或者使不同的样式应用于一个或者一组元素，还有针对于浏览器的Bug使用的Hack等。

1. 分组选择器

如果要对不同的选择器设定相同的CSS，例如，需要设定<h2>、<h3>、<p>和<ul>的前景色为蓝色，如右所示：

```
h2 { color : blue; }
h3 { color : blue; }
p { color : blue; }
ul { color : blue; }
```

这样写无疑很繁琐，因此可以将其合并，如图3-9所示。为了代码的清晰，也可以分行书写，如图3-10所示。

不同的选择器前后顺序没有区别，例如右边两行CSS规则的效果是相同的：

```
h2, h3, p, ul {color : blue;}
h3, p, ul, h2 {color : blue;}
```

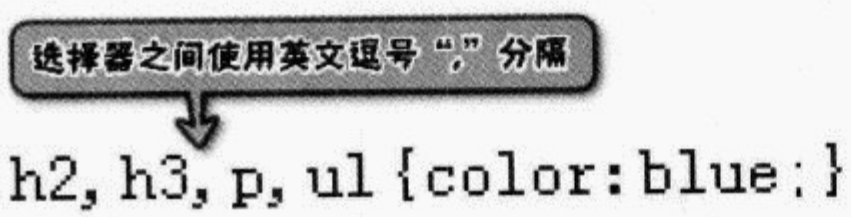


图3-9 分组选择器的书写方式

```
h2,
h3,
p,
ul {
color:blue;
}
```

图3-10 分组选择器的分行书写方式

2. 分组声明

当对于一个选择符有多条声明时，可以分开书写，例如：

```
p { font-size : 1em }
p {color : blue}
```

也可以像选择器那样合并成组书写，而每条声明间用英文分号“;”分隔，如图3-11所示。

图3-11 分组声明的书写方式

为了使样式更加易于阅读，一般将每条声明写在一个新行内，如图3-12所示。最后一条声明可以没有分号，但是为了以后修改方便，一般也加上分号。

```
p {
font-size: 1em;
color: blue;
}
```

图3-12 分组声明的分行书写

3. 综合应用

分组书写增加了CSS的灵活性，同时可以减少代码的冗余，例如右边代码：

```
p { color : blue; line-height : 200%; text-align : center; }
ul { color : blue; line-height : 200%; }
```

这两组规则的大部分声明是相同的，只有少部分不同，因此可以将相同的声明分组书写，而再对不同的部分单独书写，如右所示：

```
p, ul { color : blue; line-height : 200%; }
p { text-align : center; }
```

3.3.4 注释

CSS内的注释和(X)HTML的注释标签不同，其格式如右：

```
/* 注释文字 */
```

注释文字可以放在单独的行内（可回行），也可以放在声明之后，如：

```
/* 主样式表 */
body { font-size:12px; } /* 设置页面文字尺寸 */
p { line-height: 130%; } /* 设置段落的行高 */
/* 注释文字也可以多行。
在单独的CSS文件内，或在（X）HTML文件的<style>标签中，
都可以使用。
*/
```

小窍门： 添加注释有助于以后修改和使用。不过对于大型网站的比较复杂的样式文件，建议撰写单独的说明文档。

3.3.5 缩写

浏览速度对于网站来说至关重要，而影响速度的因素有很多种，包括Web服务器的速度、访问者网络连接情况以及浏览器必须下载的文件大小等。控制构成网站Web页面的文件大小对于减低网站数据流量，提高浏览速度也非常重要。

通过使用CSS缩写以及其他的一些简单技巧，可以在很大程度上减少样式表的大小。

1. 使用CSS的缩写属性

CSS中某些属性是可以缩写的，用来代替多个相关属性的集合。例如：

```
p {
padding-top: 5px;
padding-right: 3px;
padding-bottom: 4px;
padding-left: 2px;
}
```

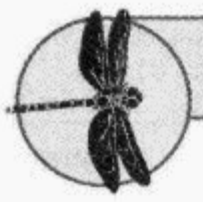
可缩写为：

```
p {
padding: 5px 3px 4px 2px;
}
```

类似的情况有：边框（border）、边界（margin）、填充（padding）等，这些属性，都包含4个边，可以合并成一行，按照“上、右、下、左”（顺时针）的顺序来定义，值中间以空格来分隔，如“margin: 5px 10px 15px 20px;”。

同时，有如下情况，值还可以再缩写。

● 如果“上≠下”但是“左=右”，可以简写成3个值（上、左右、下），如“margin: 5px 10px 20px;”，等同于“margin: 5px 10px 20px 10px;”。



注意： 如果“上=下”但是“左≠右”，则不可以缩写。

● 如果“上=下”且“左=右”，可以简写成两个值（上下、左右），如“margin: 5px 10px;”，等同于“margin: 5px 10px 5px 10px;”。

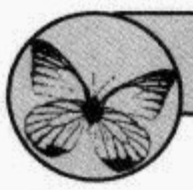
● 如果上、下、左、右都相等，就可以合并成一个值，如“margin: 5px;”等同于“margin: 5px 5px 5px 5px;”。

而另一种情况，如背景（background）、字体（font）、列表（list-style）等，虽然没有4个边框，但是也可以缩写，例如：

```
div {
background-color: #ffffff;
background-image: url(home.jpg);
background-repeat: no-repeat;
background-position: right center;
}
```

可缩写为：

```
div {
background: #ffffff url(home.jpg)
no-repeat right center;
}
```



提示： 各种属性的具体缩写方法，将在以后的相关章节详细介绍。

2. 颜色值的缩写

CSS中的颜色，可以使用十六进制的数字来表示，而类似于“#006633”这种两位重复的颜色值可以缩写为“#063”，上例中的“#ffffff”则可缩写为“#fff”。

3. 利用继承

子元素自动继承父元素的属性值，像颜色、字体等，所以对于可以继承的CSS规则，不需要重复定义。

4. 0px与0

无论什么单位，0就是0，因此0px = 0pt = 0in = 0。

3.3.6 注意事项

1. 空格

在CSS中空格是有效的，特别是在一些缩写的规则中，各值之间使用空格分开（例如对边框的定义），如图3-13所示。

多个空格会被缩略为1个，如图3-13所示，因此也可以用空格来缩写代码，以达到整齐清晰的目的，如下所示：

```
p {
  color:blue;
  line-height:200%;
  text-align:center;
}
```

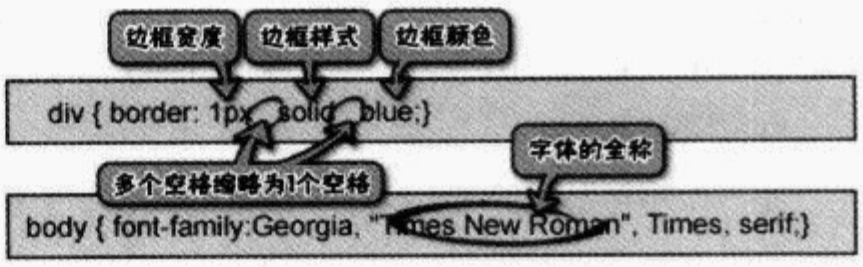


图3-13 空格与值的关系

2. 大小写敏感

CSS本身对大小写不敏感，但是，当在XHTML中使用时，由于XHTML是区分大小写的，因此要保持CSS的选择器名称定义和XHTML里的标签是一致的。例如有右上XHTML代码：

```
<div id="Box">box</div>
```

则对应的CSS如右下代码：

```
#Box { …… }
```

3. 容错

对于未声明的属性和方法，CSS分析器会忽略它的存在，如：

```
p { font-size: 12p;}    /* "12p"不是一个有效的值，被略过 */
h3, h4 & h5 {color: red;} /* "&"是CSS中没有的符号，此行整个被略过 */
p {font-size: 1em; text-color: red;} /* text-color不是一个合法属性，被略过，"font-size: 1em"有效。 */
```

3.4 元素类型

在［2.3.1 （X）HTML与浏览器内置样式］一节中介绍过（X）HTML的元素，元素是文档结构的基础。在CSS细则中解释为每个元素生成了一个包含了元素内容的框（box，也译为“盒子”）。但是不同的元素显示的方式会有所不同，例如 和就不同，而和<p>也不一样。在文档类型定义（DTD）中对不同的元素规定了不同的类型，这也是DTD对文档之所以重要的原

因之一。

3.4.1 替换和不可替换元素

从元素本身的特点来讲，可以分为替换和不可替换元素。

1. 替换元素

替换元素就是浏览器根据元素的标签和属性，来决定元素的具体显示内容。例如，浏览器会根据<img>标签的src属性的值来读取图片信息并显示出来，而如果查看（X）HTML代码，则看不到图片的实际内容。又例如，根据<input>标签的type属性来决定是显示一个输入框还是单选按钮等。

（X）HTML中的<img>、<input>、<textarea>、<select>、<object>都是替换元素。这些元素往往没有实际的内容，即是一个空元素，例如：

```
<img src="cat.jpg" />
<input type="submit" name="Submit" value="提交" />
```

浏览器会根据元素的标签类型和属性来显示这些元素。可替换元素也在其显示中生成了框。

2. 不可替换元素

（X）HTML的大多数元素是不可替换元素，即其内容直接表现给用户端（如浏览器）。例如：

```
<p>段落的内容</p>
```

是一个不可替换元素，文字“段落的内容”全被显示。

3.4.2 显示元素

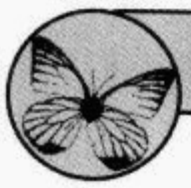
除了可替换元素和不可替换元素的分类方式外，CSS 2.1中元素还有另外的分类方式：块级元素（block-level）和行内元素（inline-level，也译作“内联”元素）。

1. 块级元素

在视觉上被格式化为块的元素，最明显的特征就是它默认在横向充满其父元素的内容区域，而且在其左右两边没有其他元素，即块级元素默认是独占一行的。

典型的块级元素有 、<p>、<h1>到<h6>等。可替换元素有可能是块级元素，而块级元素一般都不是可替换元素。通过CSS设定了浮动（float属性，可向左浮动或向右浮动）以及设定显示（display）属性为“block”或“list-item”的元素都是块级元素。

但是浮动元素比较特殊，由于浮动，其旁边可能会有其他元素的存在。而“list-item”（列表项<li>），会在其前面生成圆点符号或者数字序号。



提示：关于display属性，可以参见本书［9.2 显示类型：display属性］一节。

2. 行内元素

行内元素不形成新内容块，即在其左右可以有其他元素，如<a>、、等，都是典型的行内元素。display属性等于“inline”的元素都是行内元素。几乎所有的可替换元素都是行内元素，如<img>、<input>等。

不过元素的类型也不是固定的，通过设定CSS的display属性，可以使行内元素变为块级元素，也可以让块级元素变为行内元素。关于块级元素与行内元素，将在本书［第8章 框模型］中详细介绍。

3.5 媒体类型

在本章［3.2.3的4. media属性］一节内介绍了CSS中支持的媒体类型，制作者可以决定在不同的媒体上文档应该如何呈现于屏幕、纸面、语音合成器或者盲文设备等。某些CSS属性是只为特定的媒体而设计，如cue-before属性是为语音用户端设计的。

有时候不同媒体类型的样式表可以共享一个属性，不过对于那个属性要赋予不同的值。例如，字体尺寸font-size属性用于屏幕和打印媒体。不过这两个媒体是如此地不同，因此对于共同的属性需要不同的值，即在计算机屏幕上的文档一般要用比纸面上的文档大一些的字体。又例如，屏幕上无衬线字体要容易阅读一些，而在纸面上有衬线的字体要容易阅读一些。基于这些原因，有必要声明一个样式表（或样式表中的一段）适用于特定的媒体类型。

3.5.1 指定媒体相关的样式表

目前有以下两个方法来指定样式表的媒体相关性。

（1）在样式表中通过“@media”规则或“@import”规则来指定目标媒体。关于媒体的类型说明请参见［3.2.3的4. media属性］一节。

例如：

```
@import url("loudvoice.css") aural; /* 参见［3.2.4 导入式样式表］*/
@media print {
 一些样式规则
}
```

一个“@media”规则以一系列规则（以花括号分割）来指定目标媒体类型（以逗号分割）。@media结构允许不同媒体的样式规则存在于同一样式表中如右：

```
@media print {
 body { font-size: 10pt }
}
@media screen {
 body { font-size: 12pt }
}
@media screen, print {
 body { line-height: 1.2 }
}
```

（2）在文档语言中指定目标媒体。

如同本章［3.2.3 外部样式表］一节中介绍的那样，使用<link>的“media”属性指定一个外部样式表的目标媒体。

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

3.5.2 媒体组

虽然有的CSS属性只能在某个类型的媒体中使用，但是多数的CSS属性通常可以适用于若干个媒体，因此在后面对CSS属性详细介绍的定义列表中，有“媒体”一项，其中列出的是媒体组而不是单一的媒体类型。每一个属性适用于这个媒体组中包含的所有媒体类型。

CSS 2.1有如下几种分组方式：

- 连续媒体（continuous，例如显示器）或页面媒体（paged，例如打印机）；
- 视觉（visual）、音频（audio）、语音（speech）或触觉（tactile）；
- 栅格（grid，字符栅格设备）或者位图（bitmap）；
- 交互（interactive，对于允许用户交互的设备）或静态（static，不支持交互的设备）；
- 包含所有的媒体类型（all）。

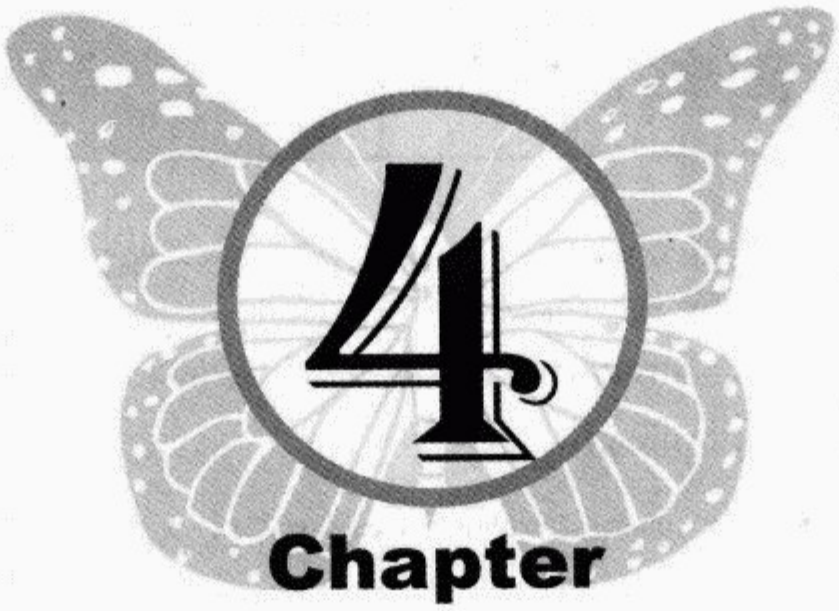
以上几种不同分类方式的媒体组，CSS 2.1媒体类型与媒体组的对照如下所示。

媒体类型	媒体组			
	连续/页面	视觉/音频/语音/触觉	栅格/位图	交互/静态
braille	连续	触觉	栅格	两者全是
embossed	页面	触觉	栅格	静态
handheld	两者全是	视觉、音频、语音	两者全是	两者全是
print	页面	视觉	位图	静态
projection	页面	视觉	位图	交互
screen	连续	视觉、音频	位图	两者全是
speech	连续	语音	不适用	两者全是
tty	连续	视觉	栅格	两者全是
tv	两者全是	视觉、音频	位图	两者全是

例如，对字体样式（font-style）属性的定义列表如下：

语法	font-style : normal | italic | oblique | inherit
说明	设定元素内文字的样式
值	normal：正常的字体。 italic：斜体，对于没有斜体变量的特殊字体，将应用oblique oblique：倾斜的字体
初始值	normal
继承性	继承
适用于	所有元素
媒体	**视觉**
计算值	同指定值

其中“媒体：视觉”一项，表示其可以实用于属于“视觉”媒体组的那些媒体类型，如“screen”和“print”等。



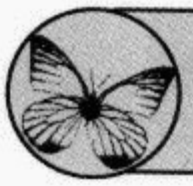
第4章
文档结构与选择器

CSS通过与（X）HTML的文档结构相对应的选择器（Selector）来达到控制页面表现的目的，而文档结构不仅仅在CSS的应用上非常重要，对于行为层（例如使用JavaScript控制元素的行为）同样也非常重要。

4.1 文档结构

（X）HTML文档可以看作一个家族树，这个树有1个祖先——根元素，然后各元素依次向下排列，例如有XHTML代码如下，其文档树如图4-1所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/ TR/xhtml1/
DTD/xhtml1-strict.dtd" >
<html xmlns= " http://www.w3.org/1999/xhtml" >
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>文档结构与选择器</title>
</head>
<body>
 <h1>第3章<em>CSS入门</em></h1>
 <p><acronym title= "Cascading Style Sheets，层叠样式表" >CSS</acronym>是一种标记性语言。</p>
 <ol>
  <li>CSS的<em>优缺点</em></li>
  <li>CSS的使用方法
        <ul>
         <li>内联式样式</li>
         <li>嵌入式样式表</li>
         <li>外部样式表</li>
        </ul>
       </li>
  <li><strong>基本</strong>样式规则</li>
 </ol>
  <p>CSS通过与（X）HTML的文档结构相对应的<a href="selector01.html" title="关于选择器的内容">选择
器（<em>selector</em>）</a>来达到控制页面表现的目的。</p>
</body>
</html>
```



提示：读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/structure.html］文件。

CSS大部分能力是基于元素的“父子”关系，如果元素A包含了元素B，那么元素A就是“父元素”，被包含的元素B是“子元素”。每个元素都是另一个元素的“父”或者“子”或者两者都是。例如，<body>既是<html>的子元素，又是<h1>的父元素。在家族树中，父子元素是相连的，而且父元素在子元素的上面一层。

“父”与“子”有时候又被一般化为“祖先（或称前辈）”和“后代（或称子孙）”，从一个元素到另一个元素中间跨越了一层或更多层，就是“祖先/后代”关系。例如图4-1中，<html>就是<h1>的祖先，<h1>则是<html>的后代。<body>是所有浏览器能显示的元素的祖先，而<html>是所有元素的祖先，也称为“根元素（root）”。“祖先/后代”关系包含“父子”关系。

有着相同父元素的元素之间互为“兄弟”关系。例如在图4-1中，<h1>和<ol>为兄弟关系，<body>是它们共同的父元素，<ul>里的3个<li>也互相为兄弟关系。

在下一节中，多种选择器都是针对文档结构匹配的，因此掌握文档结构的意义非常重要。

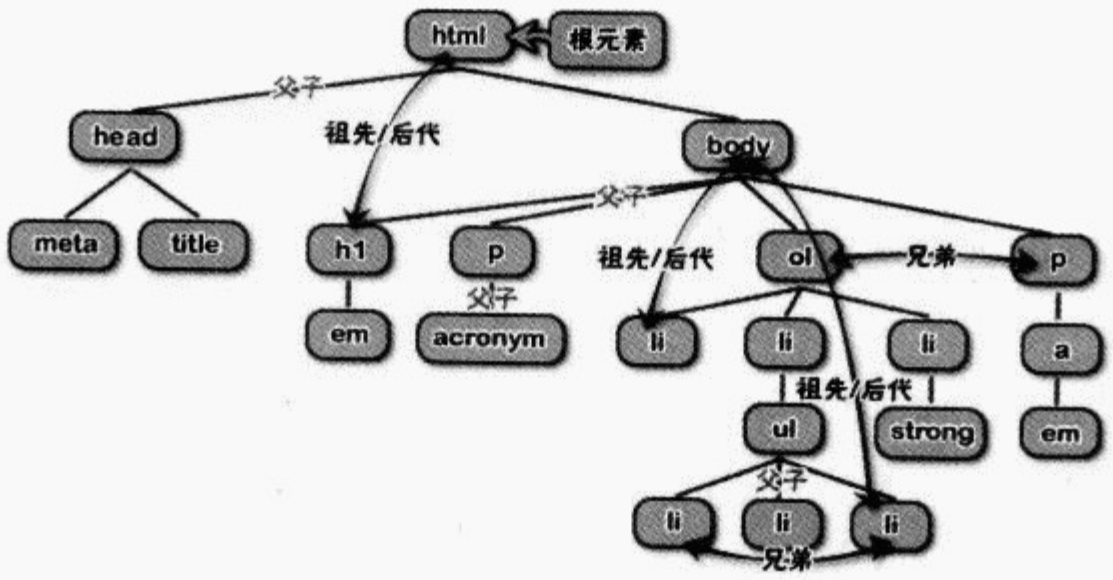


图4-1　文档结构树

4.2 CSS选择器

在本书［3.3 基本样式规则］一节中已经介绍了CSS的基本语法，如右：

样式规则由“选择器”和“声明”组成，而声明又由“属性”和“值”组成。

CSS的语法很简单，而选择器却复杂而强大，要实现对各个元素的精确控制，需要对不同的选择器有深入的理解。

提示： CSS 2.1的选择器类型非常丰富，但是由于一些浏览器只支持其中部分选择器，因此在应用上大打折扣。

4.2.1 通配选择器（Universal Selector）

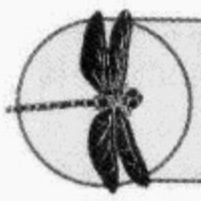
在很多计算机语言里，通常用英文的“*”号代表所有的元素，因此通配选择器定义的规则会应用在所有元素上。因此一般在CSS的开始时如右定义：

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

这样定义是把所有的元素的边距（margin）和补白（padding）定义为0，以清除浏览器的默认样式。通配选择器也可以用来定义某元素下的所有元素，例如右边代码：

```
div * { background-color: blue; }
```

则 的所有后代元素的背景色都为蓝色。



注意： 由于通配选择器会影响所有的元素，因此除了清除浏览器的内置样式以外，应谨慎使用。

4.2.2 类型选择器（Type Selectors）

类型选择器，以文档语言元素类型作为选择器，即以（X）HTML标签作为选择器，如标题“h1”、段落“p”、无序列表“ul”、列表项“li”等。

在前面的例子里使用的，大部分都是类型选择器，例如右边代码：

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

类型选择器的特点就在于简单，而且明确，

缺点是针对性较差，特别是对于表单元素，因为表单元素大部分是使用<input>标签，但是type属性不同，而类型选择器则无法精确匹配type属性不同的元素。

类型选择器通常用于定义通用的CSS规则，然后再对有特殊需求的元素采取其他的方法来定义规则，例如使用ID选择器或者类选择器。

例如，本书示范页面中，对于显示示例的 进行如右定义：

```
div {
background: #6C3; /* 定义背景色 */
border:1px solid #060; /* 定义div的边框 */
margin:0 auto 10px; /* 定义div的边距 */
…… /* 其他声明 */
}
```

如此定义，使得所有的 元素都具有了相同的外观。

4.2.3 ID选择器（ID Selectors）

可以为（X）HTML元素添加ID属性，有XHTML代码如右：

```
<div id="logo">网站的Logo图片</div>
```

相对地，在CSS中使用ID选择器来为具有此ID的元素设定CSS规则，如图4-2所示。

图4-2 ID选择器

ID属性可以定义在几乎所有（X）HTML元素上，但是同一个（X）HTML文件内，ID不允许重复，因此ID属性一般用来定义比较特殊的元素，例如页面内用来控制版面布局的模块的定义：

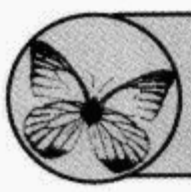
```
<body>
 <div id="header">这里是头部信息，例如网站名称。</div>
 <div id="main">这里是页面的主要内容。</div>
 <div id="footer">这里是页脚内容，例如联系地址等。</div>
</body>
```

而对应的CSS为：

```
#header { …… }
#main { …… }
#footer { …… }
```

ID也可应用在某个需要特别操作的元素，例如一个提交表单的按钮：

```
<input name="submit_form"id="submit_btn" type="submit" value="提交" />
```



提示：虽然重复的ID在用浏览器浏览页面的时候不会报错，但是如果使用JavaScript操作元素就会出问题。

还有一种限定应用的元素类型的定义方法，如图4-3所示。此CSS规则只匹配 的元素，而<p id="logo">或者的元素则不匹配。

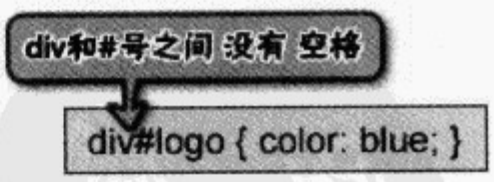


图4-3 限定元素类型的ID选择器

4.2.4 类选择器（Class Selectors）

类选择器的定义如图4-4所示。

类选择器和ID选择器很像，是通过为（X）HTML元素添加class属性而生效的，可应用在<body>内的任何元素上，不过类选择器是可重复的。例如：

```
<div class="warning">注意！ </div>
<p class="warning">注意！ </p>
```

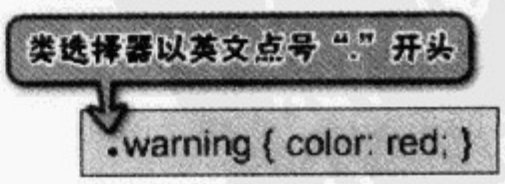
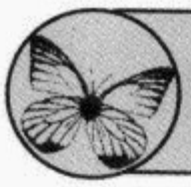


图4-4 类选择器

```
<p>这里要<span class=" warning">注意！ </span></p>
```



提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector01.html］文件。

也可以针对不同类型的元素对同一个名称的类选择器设定不同的样式规则，例如对上面的XHTML代码，可以设定CSS规则如下，其在浏览器内的显示如图4-5所示。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector02.html］文件。

```
div.warning { color: red; }
p.warning { color: blue; }
span.warning { color: yellow; }
```

图4-5　针对不同元素定义类选择器的显示效果

由图4-5可以发现，类选择器的定义非常灵活，即使是相同名称的类选择器，也可以有不同的表现。同一个元素可以设定多个类，之间用空格分开，如右所示：

```
<div class="menu main sample">一个元素可以设定多个class属性</div>
```

这几个类的样式会同时作用于该元素，如果其中有重复定义的规则，则按照“特殊性”的规定来决定如何显示。关于选择器的特殊性，请参见［4.6 层叠］一节。

4.2.5 包含选择器（Descendant Selectors）

包含选择器，也称“后代选择器”，其定义规则如图4-6所示。

包含选择器的运作原理是建立在文档结构树的基础上，上述规则的含义是“E1元素内包含的所有E2元素”，或者“E1元素后代中的所有E2元素”。例如有XHTML代码如下，其在浏览器内的效果如图4-7所示。

```
div em { color: blue; }
<div>div中的<em>em</em>
  <p>div中的p中的<em>em</em></p>
</div>
<p>与div同级的p中的<em>em</em>。</p>
```

利用包含选择器来定义：

```
div em {……}
```

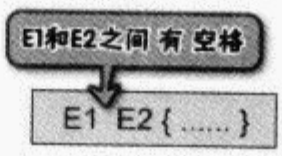
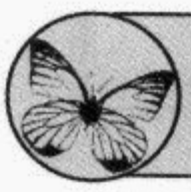


图4-6　包含选择器

图4-7　使用包含选择器定义CSS的显示效果



提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector03.html］文件。

只对 中所有后代元素生效，而没有包含在 中的<p>中的不受影响。

包含选择器可以有多层，例如：

```
div p a { …… }                  /* <div>中的<p>中的<a> */
#logo li em a { …… }            /* ID为logo的元素中的<li>中的<em>中的<a> */
#main p.warning strong { …… }   /* ID为main的元素中的应用了warning 类的<p>中的<strong> */
```

包含选择器和文档结构关系非常密切，例如有如下XHTML结构：

```
<div id="test">
 <ul>
  <li>ul中的li</li>
```

```
  <li>ul中的li</li>
 </ul>
 <ol>
  <li>ol中的li</li>
  <li>ol中的li</li>
 </ol>
</div>
```

此时，如果定义CSS如右：

```
#test li { color: red; }
```

则会匹配“text”层内所有的<li>，而如果如右定义CSS：

```
#test ul li { color: blue; }
```

只匹配<ul>中的<li>，对于<ol>中的<li>则没有影响。同样地，如果如右定义CSS：

```
#test ol li { color: yellow; }
```

则只匹配<ol>中的<li>，对于<ul>中的<li>则没有影响。包含选择器使得选择器的应用变得复杂而灵活，熟练掌握包含选择器，可以减少代码的冗余度，提高CSS的应用灵活性。

提示： 目前主流的浏览器都支持以上几种选择器。而下面所列的选择器，则会有部分浏览器不支持。在以下的章节中，除非特别声明，则IE 7.0、Firefox 2.0及Opera 8.5以及更高版本的浏览器中都支持，而IE 6.0及更早的版本则不支持。

4.2.6 子元素选择器（Child Selectors）

子元素选择器的功能和包含选择器很像，其匹配也是基于文档树的关系，其定义如图4-8所示。

与包含选择器不同的是，子元素选择器只对元素的子元素生效，而不会影响到其他后代元素。例如：

```
div > em { color:red;}
<div>
 <em>这个em是div的子元素</em>
 <p>p中的<em>是p的子元素</p>
</div>
```

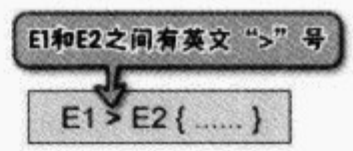


图4-8 子元素选择器

则其在浏览器内的显示效果如图4-9所示。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector04.html］文件。

由于IE 6.0及更早的版本不支持相邻兄弟选择器，因此这种选择器经常用来对IE 6.0隐藏某些CSS规则使用。

图4-9 使用子元素选择器定义CSS的显示效果

4.2.7 相邻兄弟选择器（Adjacent Sibling Selectors）

相邻兄弟选择器基于文档结构的“兄弟”关系，其定义如图4-10所示。

其含义是，与E1元素相邻的下一个兄弟元素，例如，右侧代码在浏览器内的效果如图4-11所示。

```
div + p { color: red; }
<p>与div相邻的上一个p</p>
<div>
 <p>div中的p</p>
</div>
<p>与div相邻的下一个p</p>
<p>不与div相邻的p</p>
```

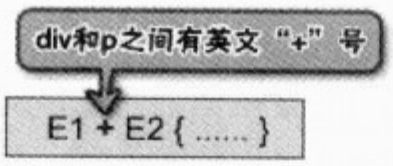
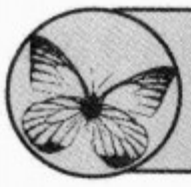


图4-10　相邻选择器

图4-11　使用相邻选择器定义CSS的显示效果



提示：读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector05.html］文件。

同子元素选择器类似，相邻兄弟选择器也可以用来对IE 6.0隐藏某些CSS规则使用。

4.2.8　属性选择器（Attribute Selectors）

顾名思义，属性选择器可以根据某个属性是否存在或属性的值来匹配元素。因此能够实现一些比较复杂的匹配。属性选择器有4种匹配方式，其各具特色。

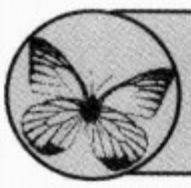
1. 简易属性匹配

简易的属性匹配定义如图4-12所示。只匹配具有“att”属性名称的E元素，而不考虑“att”的值为何。

例如，有代码如下，则其在浏览器内的效果如图4-13所示。

```
div[class] { color: red; }
<div class="box1">具有class="box1"属性的div</div>
<p class="box1">具有class="box1"属性的p</p>
<div class="box2">具有class="box2"属性的div</div>
<div id="main">具有id="main"属性的div</div>
```

图4-12　属性匹配选择器



提示：读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector06.html］文件。

属性匹配可以匹配该元素任何合法的（X）HTML属性，也可以同时匹配多种属性，例如有代码如下，其在浏览器内的效果如图4-14所示。

图4-13　使用属性匹配选择器定义CSS的显示效果

图4-14　匹配多个属性的属性选择器的显示效果

```
a[title][class] { color: green; }
<ul>
  <li><a href="#" title="带有title属性的链接">只有title属性的链接</a></li>
  <li><a href="#" class="new">只有class属性的链接</a></li>
  <li><a href="#" title="带有title属性的链接，有class属性的链" class="new">既有title属性，也有class属性的链接</a></li>
</ul>
```

2. 精确值匹配

精确值匹配的属性选择器选择具有指定的属性且属性值等于指定的值的元素，例如：

```
input[type="text"] { border: 1px solid blue; }
```

则将匹配具有type="text"属性的<input>元素，例如有XHTML代码如下，则其在浏览器内的显示如图4-15所示。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector07.html］文件。

```
<form id="form1" method="post" action="#">
 <fieldset>
  <legend>信息登记</legend>
  <label>昵称：
  <input name="nikename" type="text" size="16" maxlength="30" />
  </label>
  <label>年龄：
  <input name="age" type="text" size="6" maxlength="8" />
  </label>
  <label>密码：
  <input name="password" type="password" id="password" value="" size="16" />
  </label>
  <label>
  <input type="radio" name="sex" value=" male" />
  男</label>
  <label>
  <input type="radio" name="sex" value=" female" />
  女</label>
 </fieldset>
</form>
```

值匹配的属性选择器也可以同时匹配多种属性，例如：

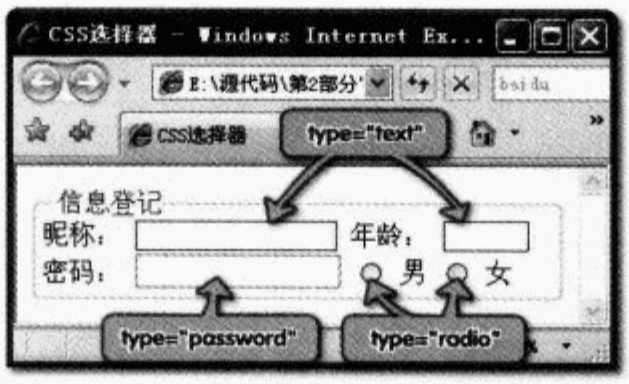
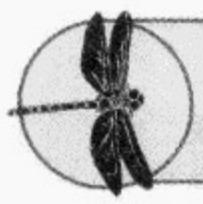


图4-15 值匹配的属性选择器的显示效果

```
input[type="radio"][name="sex"] { …… }
```



注意： 属性值必须是标识符或字符串。选择器中属性名和值的大小写敏感性取决于文档语言的敏感性。

3. 部分值匹配

（X）HTML元素（或者非（X）HTML的其他类型文件的元素）的某些属性值可以是多个，中间以空格分开，例如在［4.1.4 类选择器］中介绍了一个元素设定多个类的情况。而当要匹配多个属性值中的某一个的时候，就需要使用部分值匹配的属性选择器，其定义如图4-16所示。

在图4-16中，所有E元素中具有“att”属性且该属性包含“val”值的那些元素。例如有代码如下，则其在浏览器内的显示如图4-17所示。

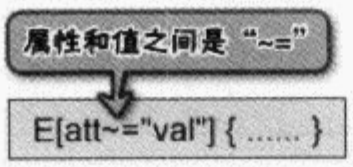


图4-16 部分值匹配的属性匹配选择器

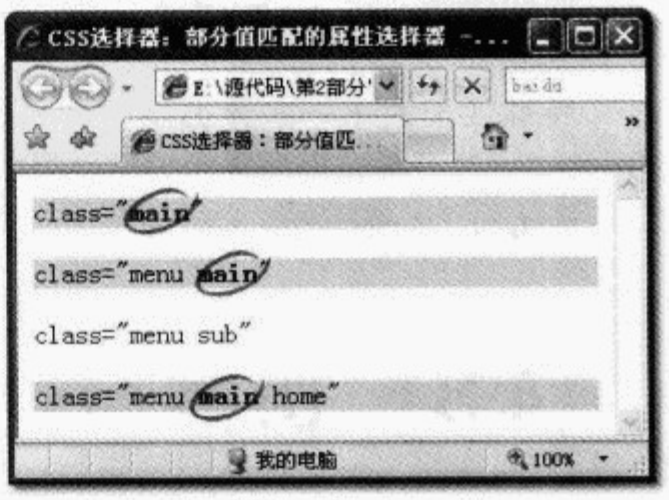
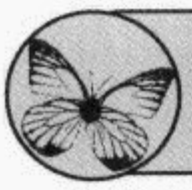


图4-17 部分值匹配的属性选择器的显示效果

```
p[class~="main"] { background:yellow; }
<p class="main">class=&quot;<strong>main</strong>&quot;</p>
<p class="menu main">class=&quot;menu <strong>main</strong>&quot;</p>
<p class="menu sub">class=&quot;menu sub&quot;</p>
<p class="menu main home">class=&quot;menu <strong>main</strong> home&quot;</p>
```



提示：读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector08.html］文件。

<p>元素的class属性的值不分先后顺序，只要包含值“main”则匹配元素。此时选择器内的值中不能包含空格，例如下面的代码是错误的：

```
p[class~="menu main"] { background:yellow; }
p[class~=" main "] { background:yellow; }
p[class~="  main"] { background:yellow; }
```

4. 特殊匹配

特殊匹配的属性选择器定义如图4-18所示。

当元素的“att”属性值是一个以英文“-”号分割的字词列表，并且以“val”开头，则匹配成功。匹配总是从属性值的头上开始。例如，有XHTML代码如下，则其在浏览器内显示如图4-19所示。

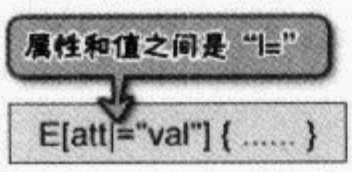
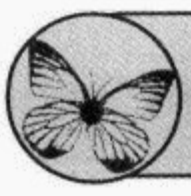


图4-18 特殊匹配的属性匹配选择器

图4-19 特殊匹配的属性选择器的显示效果



提示：读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/selector09.html］文件。

```
p[class|="menu"] { background:yellow; }
<p class="menu">class=&quot;menu&quot;</p>
<p class="menu-main">class=&quot;menu-main&quot;</p>
<p class="menu-sub">class=&quot;menu-sub&quot;</p>
<p class="menu_main">class=&quot;menu_main&quot;</p>
<p class="other-menu">class=&quot;other-menu&quot;</p>
```

特殊匹配对于一些遵循同一命名规范来定义的属性值非常有效，例如，某些图片全部以“picture-”开头，而后跟随编号，那么，要对这些图片设定CSS就可以使用特殊匹配：

```
img[src|="picture"] { …… }
```

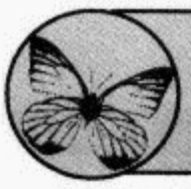
4.3 伪类与伪元素

伪类和伪元素是两种有意思的选择器，之所以称“伪”，是因为它们实际上并不存在于源文档或者文档树中，但是它们又确实可以显示出效果。

4.3.1 伪类（Pseudo-Classes）

应用最广泛的伪类，就是链接的普通状态、鼠标悬停状态、已单击过的链接以及当前激活的链接，虽然没有在XHTML内对<a>元素设定class属性，也可以生效。例如：

```
a:link { color: blue; }
a:visited { color: green; }
a:hover { color: red; }
a:active { color:yellow; }
<ul>
 <li><a href="http://www.ddcat.net/">链接1</a></li>
 <li><a href="http://www.4css.cn/">链接2</a></li>
 <li><a href="http://color.ddcat.net/">链接3</a></li>
 <li><a href="http://bbs.ddcat.net/">链接4</a></li>
</ul>
```



提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/pseudo_classes.html］文件。

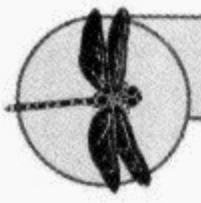
伪类以英文冒号“:”开头，例如：

```
div:hover { …… }
input:focus { …… }
```

1. 链接伪类 :link和:visited

为了便于访问者区分哪些链接是已经点击过的，浏览器在显示未访问的连接和已访问的链接时，总有一些不同。CSS提供了下面两种伪类来定义，这两个伪类是互相排斥的。

- :link伪类适用于那些还未被访问的链接；
- :visited伪类适用于用户已经访问过的链接。



注意： 当访问者清除了浏览器的缓存以后，已访问过的链接会恢复为未访问的链接。

2. 动态伪类:hover、:active和:focus

交互的用户端有时根据用户的动作改变渲染效果。CSS为通用的情况提供3个伪类。

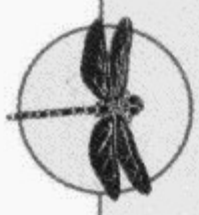
● :hover伪类适用于用户指向一个元素（以某个指点设备），但还没有激活它的时候。例如，鼠标指针指向一个段落（<p>）时应用该伪类。

● :active伪类适用于一个元素被用户激活的时候。例如，在用户按下鼠标到放开鼠标的这一段时间内。

● :focus伪类适用于一个元素获得焦点（接受键盘事件或其他形式的文本输入）的时候。

这些伪类互不排斥。一个元素同时可以匹配其中的若干个。例如：

```
input:focus { background: yellow }
input:focus:hover { background: white } /* 匹配在伪类:focus和伪类:hover中的<input>元素。 */
```



注意： CSS并没有规定哪些元素可以有上述的状态，或者如何进入和离开这些状态。不同的设备和用户端对于指向和激活元素的表现方法可能有所不同。在CSS 1中，“:active”伪类和“:link”、“:visited”是互相排斥的。而在CSS 2中一个元素可以既是“:visited”又是“:active”。具体适用哪一个属性由一般的层叠规则决定。IE 6.0及更早的版本，只支持<a>元素的:hover伪类，而IE 7.0、Firefox 2.0及Opera 8.5等浏览器支持其他元素的:hover伪类，如div:hover { …… }。

:focus伪类和:hover伪类很像，:hover应用在访问者的鼠标经过某个元素时，而:focus则是在访问者点击了什么或者使用Tab键跳入某个元素时（即元素获得了“焦点”时）生效。因此通常在表单上设定:focus，以提示用户目前光标所处的位置，例如：

```
input:hover { border:1px solid red; }
input:focus { background: #9CF; }
```

则在浏览器内显示如图4-20所示。IE 7.0及更早的版本都不支持:focus伪类。

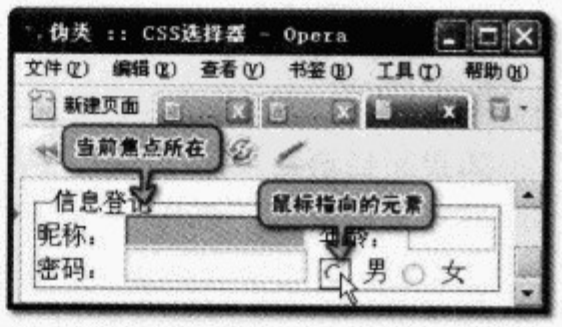


图4-20　:focus的应用

3. 多种链接样式

通过类选择器和伪类，可以为相同页面的不同元素内的链接定义不同的CSS，例如下列代码，其在浏览器内的显示如图4-21所示。

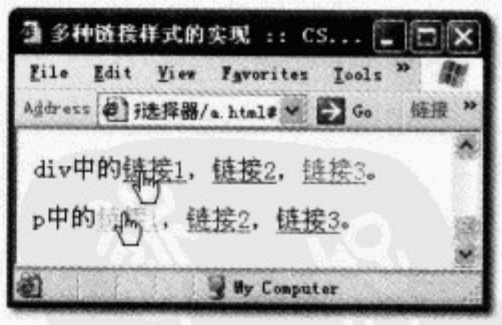


图4-21　页面中不同颜色的链接

```
div a:link { color: blue; }
div a:visited { color: gray; }
div a:hover { color: red; }
p a:link { color: green; }
p a:visited { color: brown; }
p a:hover { color: gold; }
```

```
<div>div中的<a href="http://www.ddcat.net/" title="测试链接">链接1</a>，<a href="http://bbs.ddcat.net/" title="测试链接">链接2</a>，<a href="#" title="测试链接">链接3</a>。</div>
<p>p中的<a href="http://www.ddcat.net/" title="测试链接">链接1</a>，<a href="http://bbs.ddcat.net/" title="测试链接">链接2</a>，<a href="#" title="测试链接">链接3</a>。</p>
```


提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/ a.html］文件。

4. 子元素伪类:first-child

在一些情况下，需要对某个元素的第一个子元素进行操作（例如<ul>的第一个<li>）此时就需要用到“:first-child”伪类，其语法如下：

```
E:first-child { …… }
```

上述规则将匹配某些元素中的第一个子元素，且此子元素的类型为E的，如图4-22所示。

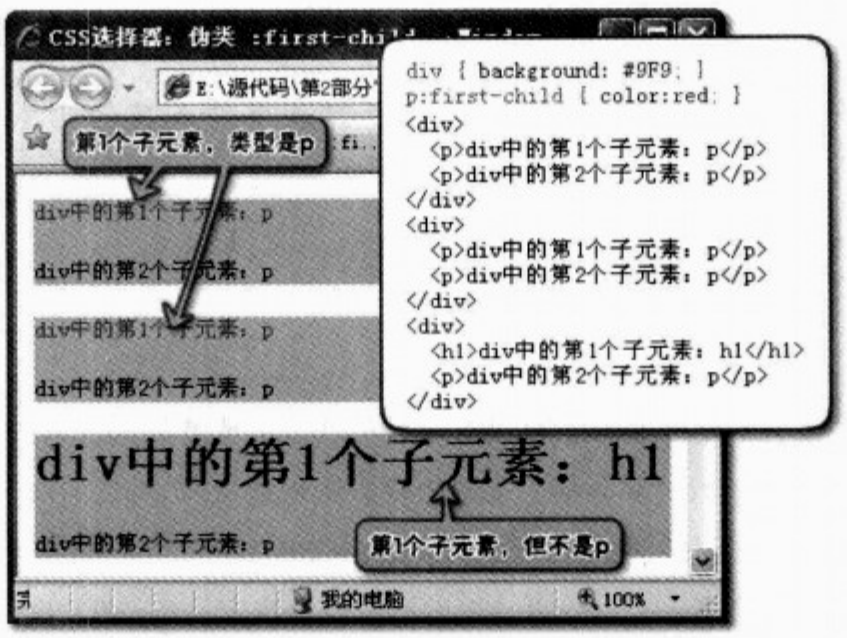


图4-22 :first-child的应用

提示： IE 6.0及更早的版本则不支持“:first-child”。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/ first-child.html］文件。

5. 语言伪类 :lang

如果文档语言规定了一个元素的自然语言是如何定义的，就可能基于元素的语言，写出 CSS 的选择器来匹配它。在HTML中，语言是由lang属性、<meta>元素、可能还有从协议得到的信息（如HTTP头）的组合决定的。而XML则使用一个xml:lang属性。

伪类:lang（C）匹配以语言C表示的元素。这里“C”是一个语言代码，在《HTML 4.0》以及《RFC 1766》中定义，它的匹配方式和“|=”操作符一样。例如有如下代码，其在浏览器内的显示如图4-23所示。

```
:lang(en) { color:blue;}
<p lang="en" xml:lang="en">lang=&amp;quot;en&amp;quot;</p>
<p lang="en-US" xml:lang="en-US">lang=&amp;quot;en-US&amp;quot;</p>
<p lang="uk" xml:lang="uk">lang=&amp;quot;uk&amp;quot;</p>
```

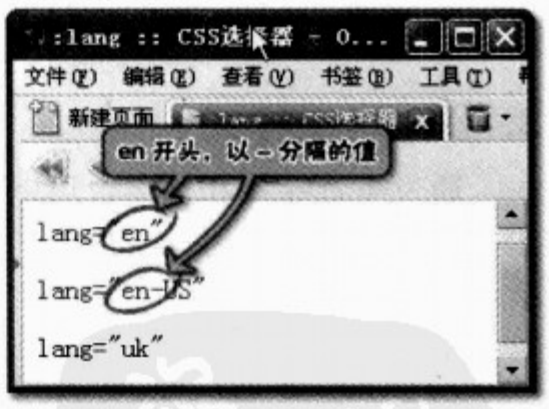
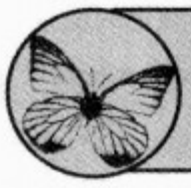


图4-23 :lang的应用



提示： IE 7.0及更早的版本则不支持“:lang”。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/ lang.html］文件。

4.3.2 伪元素（Pseudo-Elements）

伪元素同样也是以英文“:”开头，它的作用是向文档内插入虚构的元素来实现一些效果。

1. :first-line伪元素

伪元素:first-line对一个段落的第一个格式化的行应用特殊的样式。例如有代码如下，其在浏览器内的显示如图4-24所示。选择器“p:first-line”并不匹配任何实际的（X）HTML元素。它匹配由浏览器在每一段开始插入的一个伪元素。当改变段落的宽度时，第一行的内容也随之改变。

```
p:first-line { color: blue; }
p { background: #9F9; width: 20em; }
.p2 { width: 15em; }
 <p>选择器"p:first-line"并不匹配任何实际的（X）HTML元素。它匹配由浏览器在每一段开始插入的一个伪元素。</p>
 <p class="p2">选择器"p:first-line"并不匹配任何实际的（X）HTML元素。它匹配由浏览器在每一段开始插入的一个伪元素。</p>
```

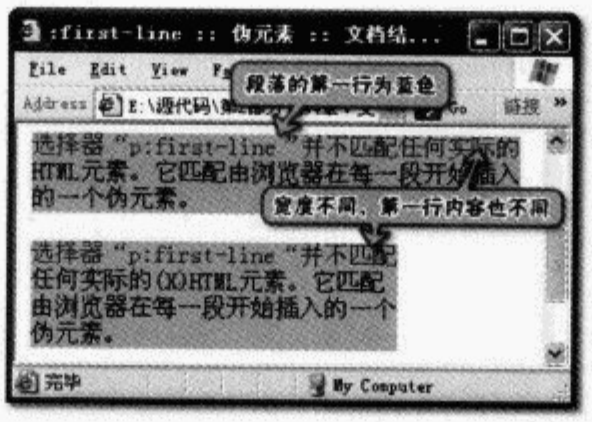


图4-24　:first-line的应用

注意：第一行的长度取决于很多因素，如页面的宽度、字体尺寸等。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/ first-line.html］文件。

浏览器可能加以“重写”包含“:first-line”的虚构标签。类似下述代码：

```
<p><p:first-line>选择器“p:first-line”并不匹配任何实际的</p:first-line>（X）HTML元素。它匹配由浏览器在每一段开始插入的一个伪元素。</p>
```

如果一个伪元素截断了一个实际的元素，则需要的效果通常可以通过一个虚构的标签来达到。因此，如果在上面的例子中加入一个元素：

```
<p><span class="test">选择器“p:first-line”并不匹配任何实际的（X）HTML元素。</span>它匹配由浏览器在每一段开始插入的一个伪元素。</p>
```

浏览器可能在为“p:first-line”插入虚构标签时，为加入开始和结束标签：

```
<p><p:first-line><span class="test">选择器“p:first-line”并不匹配任何实际的</span></p:first-line><span class="test">（X）HTML元素。</span>它匹配由浏览器在每一段开始插入的一个伪元素。</p>
```

:first-line伪元素只可以和块级元素连用，其特性和行内元素类似，但是有一些特定的限制。只有下列CSS属性可以应用在:first-line伪元素：字体属性、颜色属性、背景属性、word-spacing、letter-spacing、text-decoration、vertical-align、text-transform、line-height、text-shadow以及clear。

2. :first-letter伪元素

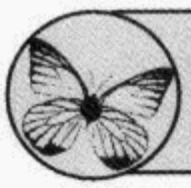
伪元素:first-letter匹配一段文字的第一个中文字或者字母，可以用于“词首大写”以及“大写字母下沉”，这些都是常用的印刷效果。如果不设定:first-letter的float属性，则它的特性和一个行内元素类似。

可以应用在:first-letter伪元素上的CSS属性包括字体属性、颜色属性、背景属性、 text-decoration、vertical-align（仅当不浮动“float:none”时）、text-transform、line-height、

margin、padding、border、float、letter-spacing、word-spacing、text-shadow以及clear。例如下列的代码，其在浏览器内的显示如图4-25所示。

图4-25 :first-letter的应用

```
p:first-letter {
color: red;
font-size: 200%;              /* 字体大小为200% */
}
.p2:first-letter {
float: left;                  /* 向左浮动 */
}
.p2 span {
text-transform: uppercase;  /* 字母大写 */
}
<p>伪元素:first-letter匹配一段文字的第一个中文字或者字母。</p>
<p><span>The first</span> few words of an article in The Economist.</p>
```



提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/ first-letter.html］文件。

这一例子的虚构标签为：

```
<p><p:first-letter>伪</p:first-letter>元素:first-letter匹配一段文字的第一个中文字或者字母。</p>
<p><span><p:first-letter>T</p:first-letter>he first</span> few words of an article in The Economist.</p>
```


注意： :first-letter伪元素的标签与内容（既第一个字符）毗邻，而:first-line伪元素的开始标签插入在伪元素所连的元素的开始标签之后。

标点符号（Unicode字符属性里面定义为开始标点符、结束标点符和其他标点符号）如果出现在首字母前，也要包含在内，如图4-26所示。

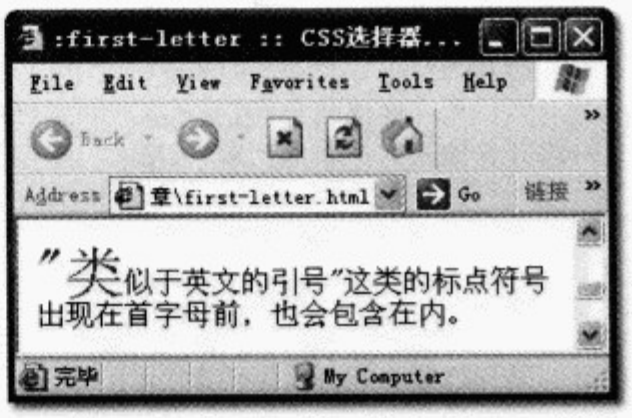
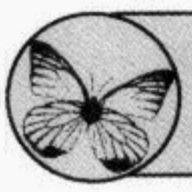


图4-26 包含英文标点符号的首字



提示： Ps，Open punctuation（开始标点符）。Pe，Close punctuation（结束标点符）。Po，Other punctuation（其他标点符号）。具体参看http://www.unicode.org/（英文）的内容。

如果文字以中文的引号开始，不同的浏览器处理方法则不太一样，如图4-27所示。

同:first-line伪元素一样，:first-letter伪元素只匹配块级元素的一部分:first-letter元素在:first-line元素之内。:first-line的属性将为:first-letter继承，但是如果:first-letter指定了相同的属性，则继承的属性被覆盖。

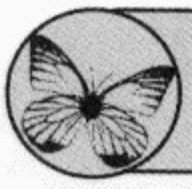
3. :before和:after伪元素

伪元素:before和:after用来在一个元素的内容之前（:before）或之后（:after）插入生成的内容。例如下述代码在浏览器显示如图4-28所示。

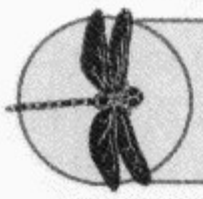
图4-27　包含中文标点符号的首字

```
p:before { content: "[["; color:green; }
p:after { content: "]]"; color:green; }
<p>伪元素:before和:after</p>
```

图4-28　:before和:after的应用



提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/before_after.html］文件。



注意： :before和:after必须和CSS的content属性配合使用，请参见本书［12.2 生成的内容］一节。IE 7.0及更早的版本不支持:before和:after伪元素。

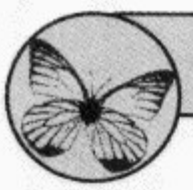
4.3.3 注意

在伪类和伪元素的定义方面，有下列几点需要特别注意。

1. 链接的伪类要注意定义顺序

对于<a>的4种状态，此处需要注意定义的顺序，即“LVHA（LoVeHAte，喜欢和讨厌）”，如下所示：

```
:Link  >  :Visited  >  :Hover  >  :Active
```



提示： 按这种顺序定义，也是由于层叠的原因，请参见本章的［4.6层叠］一节。

2. 伪类与伪元素在选择器中的位置

伪类可以在选择器的任何位置出现，而伪元素只可以出现在选择器的主体之后，例如左边代码，而右边代码是无效的：

```
a:hover span { …… }
div p:first-child { …… }
```

```
p:first-letter:hover { …… }
div:first-line a { color: red; }
```

4.4 指定值、计算值和实际值

制作者定义CSS时，不需要对每个属性都指定一个值，而对于解释网页的用户端（如浏览器），当它解析了一个文档并且生成了文档树，它必须为文档树中的每一个元素，根据目标媒介类型所适用的每一个属性，指定一个值。也就是说，无论制作者是否在CSS中定义了某个属性，每个属性都有一个值，可能是制作者定义得值，也可能是属性的初始值，或者浏览器内部样式的值。

制作者所写的CSS值，可以是一个具体的数字，也可能是一个百分比，或者一个缩放因子，而对于浏览器来说，它必须将用户定义的值转变为可以在屏幕上显示的具体的数值。一个属性最终的值的确定需要经过4步计算：首先是通过规则确定一个“指定值”，然后转换成用以继承的“计算值”，如果需要再转换为一个绝对的“使用值”，最后依照局部的环境限定转换为“实际值”。

1. 指定值

用户端根据下列规则给每个属性分配指定值（按照优先级排序）：

- 如果值中包含层叠，使用层叠；
- 否则，如果属性是继承的且元素不是文档的根元素，使用它父元素的计算值；
- 否则使用初始值（初始值在每个属性的定义列表内有说明）。

2. 计算值

在层叠中，指定值决定计算值，而指定值可以是一个绝对值（例如颜色值“red”），也可以是一个相对值（例如“em”和“ex”需要根据像素或者绝对长度来计算）。

对于绝对值，不需要经过计算来得到计算值。而相对值必须转换为计算值：百分比要乘以一个参考值（每一个属性都会定义参考值是什么），包含相对单位的值（em、ex、px）必须乘以相应的字体或点的尺寸以得到绝对值，“auto”值必须由各属性给出的公式加以计算，某些关键字（smaller、bolder、inherit）根据它们的定义而加以替换。

大部分情形下，元素继承的是计算值。不过有一些属性的指定值也可以被继承（例如line-height属性的数字值）。子元素不继承计算值的情况在属性定义中有描述。

当指定值不是“inherit”时，属性的计算值根据属性定义列表中“计算值”一项来确定。关于“inherit”值，请参见［4.5.3 “inherit”值］一节。

3. 使用值

有时候，计算值可能要到文档被用户端解释的时候才能确定，例如，某个 设定了宽度为其包含块宽度的80%，那么，其计算值需要在用户端确定了其包含块的宽度以后，才能计算出来。使用值就是指从绝对值中去除了其他因素以后计算出来的值。

4. 实际值

实际值就是使用值实际应用中的近似值。原则上，使用值是用户端用来绘制元素的值，但是用户端在某些特定的环境下可能不能使用这个值。例如边框的宽度只能以整数的像素值来显示，因此可能会计算出一个近似的宽度值，再例如，某些显示设备只能用黑白及灰度值来显示彩色的内容。

4.5 继承

在［3.3.2继承与层叠］一节中曾简要介绍过继承：（X）HTML元素可以从其父元素那里继承部

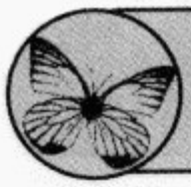
分CSS属性，即使当前元素并没有定义该属性。

4.5.1 值的继承

继承也是基于文档树的，文档树中元素的某些属性可以被其子元素继承，每一个CSS属性都定义了它能否被继承。例如，有下列代码，其在浏览器内显示如图4-29所示。

```
p { color: green; }
<p>文档树中一元素的<strong>某些值</strong>可以被其子元素继承。</p>
```

图4-29　值的继承



提示： 读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/inheritance.html］文件。

要设定文档的某些默认样式属性，可以在文档树的根上设定该属性，如果这个属性可以继承，则其后代元素将继承这个属性，如color、font-size等属性。在（X）HTML中，<html>或<body>元素可以实现这一功能。例如右上代码：

```
body { color: black; }
```

由于color属性是可继承的，所有<body>元素的后代都继承颜色值为“black”。指定的百分比值不被继承，但是计算值可以被继承。例如右下代码：

```
body { font-size: 12px; }
h1 { font-size: 120% }
<body>
 <h1>标题1 的<em>文字</em>会大一些</h1>
 <p>段落的文字继承body的定义</p>
</body>
```

<h1>元素的font-size属性的计算值为“14.4px”（其父元素值12px的120%）。由于“font-size”的计算值被继承，元素也继承“14.4px”的计算值。但是由于1个像素（1px）是电脑最小的显示单位，不可能显示“0.4px”，所以<h1>和元素“font-size”的实际值是“14px”。

4.5.2 “inherit”值

每一个属性可以指定值为“inherit”，即对于给定的元素，该属性和它父元素相对属性的计算值取一样的值。继承值通常只用作后备值，它可以通过显式地指定“inherit”而得到加强，例如：

```
p { font-size: inherit; }例子：
```

4.5.3 继承的局限性

继承虽然减少了重复定义的麻烦，但是，有些属性是不能继承的，例如border（边框）、margin（边距）、padding（补白）和背景等。

这样设定是有道理的，例如，为一个 设定了边框，如果此属性也继承的话，那么在这个 内所有的元素都会有边框，这无疑会产生一个让人眼花缭乱的结果。同样地，影响元素位置的属性，如margin（边距）和padding（补白），也不会被继承。

```
body { font-size: 12px; }
<h2>2级标题的文字不是12px。</h2>
```

同时，浏览器的默认样式也在影响着继承的结果。例如右边代码：

这是因为浏览器的默认样式设定了<h2>的CSS规则。同时，有些老版本的浏览器可能对继承支持得不太好，例如，某些浏览器当遇到<table>时，就会丢失所有的继承的属性，例如如下代码，在IE 5.5中的显示如图4-30所示。

```
body { font-size: 12px; }
<p>段落的文字继承body的定义</p>
<table>
 <tr>
     <td>单元格</td>
     <td>单元格</td>
 </tr>
</table>
```

图4-30 IE 5.5中表格没有继承<body>的属性

因此，如果要照顾到这些老版本的浏览器，则可能需要如右定义：

```
body, table, th, td { …… }
```

4.6 层叠

在［2.3.1（X）HTML与浏览器内置样式］一节中曾介绍过，样式表可能有3个不同的来源：制作者、用户和浏览器。这3个来源的样式表可能在范围上有重叠，它们根据层叠规则互相作用。

4.6.1 层叠的顺序

CSS的层叠对每一个样式规则指定一个权重。如果要应用若干个规则，那么权重最大的那个规则具有优先权。层叠规则依据下面几个步骤进行。

1. 查找有冲突的元素

浏览器会找到那些存在疑问的元素和属性的声明，如果相关联的选择器匹配存在疑问的元素，则声明适用。

2. 样式表的来源

按照规则的重要性（普通或者重要）和来源（用户、制作者或者浏览器）来从低到高排序：

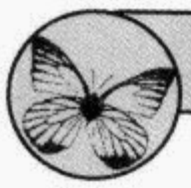
（1）浏览器的默认样式；

（2）用户定义的普通样式；

（3）制作者定义的普通样式；

（4）制作者定义的重要性（“!important”声明）样式；

（5）用户定义的重要性（“!important”声明）样式。



提示：重要性（“!important”声明），请参见本书［4.6.4 重要性］一节。

对于制作者定义的外部CSS文件引入的规则，它们的权重取决于它们引入的顺序。例如：

```
<link rel="stylesheet" href="basic.css" type="text/css" media="all" />
<link rel="stylesheet" href="font.css" type="text/css" media="all" />
```

则font.css中的定义高于basic.css中的定义。对于在样式表中使用@import规则引入的其他样式表，优先级规则同样适用。嵌入式样式表的规则高于从文件引入的样式规则。行内样式表则又高于嵌入式样式表。

3. 选择器的特殊性

声明的第2排序基于选择器的特殊性：特殊的选择器超越一般的选择器。伪元素和伪类分别被视为一般元素和一般类。

4. 规则出现的先后次序

最后，根据规则出现的先后次序来排列。如果两条规则具有相同的权重，相同的来源和相同的特殊性，则后出现的规则超越先出现的规则。

引入的样式表中的规则被认为出现在样式表本身的所有规则之前。除了个别声明的"!important"指定，上述策略给予制作者的样式表比用户样式表更大的权重。

4.6.2 特殊性的计算

既然有层叠的规则，那么，如果有如下代码，其在浏览器内会如何显示呢？

```
.warning { color: red; }
p { color: green; }
<p class=" warning" >层叠和继承的规则如何实现？ </p>
```

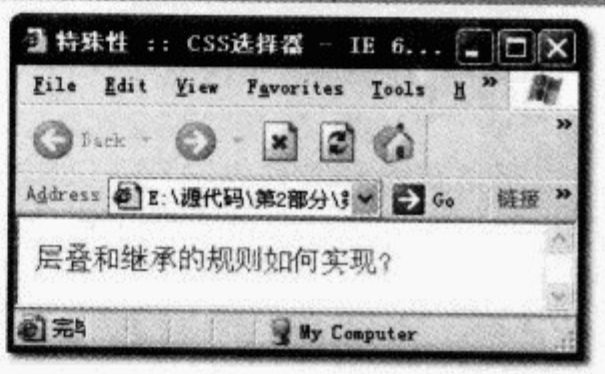


图4-31 选择器的特殊性

此代码在浏览器内的显示如图4-31所示。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/specificity.html］文件。

这是由于类选择器"warning"和类型选择器"p"的"特殊性"不同。特殊性（specificity）描述了不同选择器的相对权重（weight）。一个选择器的特殊性是这样计算的：

- 如果CSS属性是通过（X）HTML元素的style属性定义的，则记为a=1，否则记为0；由于style属性是写在（X）HTML标签内的，因此不存在选择器，所以：a=1, b=0, c=0, 且d=0；
- 计算选择器中ID选择器的数量，计为b；
- 计算选择器中类选择器、属性选择器和伪类的数量，计为c；
- 计算选择器中类型选择器的数量，计为d；
- 忽略伪元素。

将这4个数字（a, b, c, d）相连（数字进制要以大的为准），得到特殊性。例如：

```
* { …… }
li { …… }
ul li { …… }
ul ol+li { …… }
h1 + *[rel="up"] { …… }
ul ol li.warning { …… }
li.menu.level { …… }
#x34y { …… }
<p styel="……">
```

```
/* 特殊性 = 0, 0, 0, 0 */
/* 特殊性 = 0, 0, 0, 1 */
/* 特殊性 = 0, 0, 0, 2 */
/* 特殊性 = 0, 0, 0, 3 */
/* 特殊性 = 0, 0, 1, 1 */
/* 特殊性 = 0, 0, 1, 3 */
/* 特殊性 = 0, 0, 2, 1 */
/* 特殊性 = 0, 1, 0, 0 */
/* 特殊性 = 1, 0, 0, 0 */
```

特殊性高的规则会取代特殊性低的规则，无论其书写的先后顺序如何，例如右上代码，或者如右下代码：

```
h1 {color: red;}            /* 0,0,0,1 */
body h1 {color: green;}     /* 0,0,0,2 （胜出）*/
```

```
h2.grape {color: purple;}   /* 0,0,1,1 （胜出）*/
h2 {color: silver;}         /* 0,0,0,1 */
```

4.6.3 继承和特殊性

在特殊性的框架下，继承的特殊性为"0"。也就是说，任何显式的规则声明都会覆盖掉继承的样式，例如有如下代码：

```
em { color: blue; }
p.list { color: gray; }
<p class="list">继承的特殊性为<em>"0"</em>。</p>
```

虽然“p.list”的特殊性为“0，0，1，1”，但是，对“em”的color声明还是会覆盖掉从“p.list”继承的color样式，因此在浏览器内的显示如图4-32所示。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/cascade.html］文件。

图4-32 继承与特殊型

因此，如果需要让<p>内的能呈现和<p>一样的颜色，则需要如下定义：

```
p.list, p.list em { color: gray; }
```

4.6.4 重要性

虽然层叠和特殊性决定了CSS规则的最后应用效果，但是，也可以通过声明某个规则的“重要性”来提高此规则的权重，如图4-33所示。读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/impor-tant.html］文件。

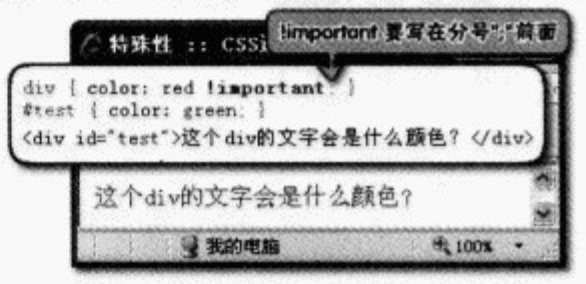


图4-33 重要性的表现

虽然ID选择器的特殊性高于类型选择器，但是由于类型选择器（div）属性值后面添加了“!important”重要性声明，因此“color：red”这条声明的特殊性最高。

“!important”重要性声明的权重甚至高于内联式样式，例如有如下代码，其在浏览器内显示如图4-34所示。

```
div { color: red !important; }
<div style="color: blue;">设定了style的div</div>
```

在CSS 1中，制作者的“!important”规则超越用户的“!important”规则。但是在CSS 2中，用户的“!important”规则具有最高的优先级，这样可以让网页更具易用性，例如有些视力不好的用户，可能会设定比较大的字体，这样就可以防止制作者定义了过小的字体而使用户阅读困难。

图4-34 重要性的权重高于内嵌式样式

4.6.5 非CSS的表现类内容

现在仍然有一些制作者在（X）HTML文档内插入一些表现类的内容，例如<font>标签和align属性，这些表现类的内容被认为具有0特殊性，并且被当作是插入在作者样式表的开头部分，因此可能会被后面定义的样式规则覆盖。

例如，下列代码在浏览器内显示如图4-35所示。

图4-35 重要性的权重高于内嵌式样式

```
p {
background: yellow;
text-align: left;
}
<p align="right">&lt;p align=&quot;right&quot;&gt;, CSS: text-align: left;</p>
```

由图4-35可以看到，虽然<p>的align属性定义为“right”，即右对齐，但是由于CSS中定义

了“text-align: left;”，因此文字还是左对齐显示。但是，如果是下面的代码：

```
.test { color: green; }
<p class="test">p内的示例<font color= "blue">文字</font>。</p>
```

则<font>内的文字依然是蓝色（blue），因为对于<font>，<p>的color属于继承，因此标签内的属性高于继承值，但是如果增加CSS规则如右：

```
font { color: red; }
```

则<font>内的文字会变为红色（red），即“color=" blue"”被CSS的“color: red;”覆盖。本小节示例代码读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/cascade2.html］文件。

4.7 CSS 3新增选择器前瞻

CSS 3在CSS 2支持的选择器的基础上，又增加了多种功能强大而且非常实用的选择器，在本节内将对这些选则器作简单介绍。读者可以访问W3C的官方网站http://www.w3.org/（英文）了解更多关于CSS 3的信息。

4.7.1 更多的属性选择器

虽然到本书完稿之日，CSS 3还未正式公布，但是其部分内容已经被一些浏览器采纳并支持，例如子字符串匹配的属性选择器等。

CSS 3对属性选择器的又增加了3种子字符串的匹配方式。

- E[att^="val"]：匹配所有E元素中att属性的值以“val”开始的所有元素。
- E[att$="val"]：匹配所有E元素中att属性的值以“val”结束的所有元素。
- E[att*="val"]：匹配所有E元素中att属性的值中包含字符串“val”的所有元素。

例如有XHTML结构如下：

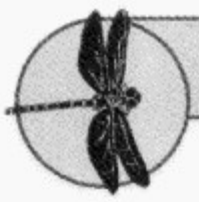
```
<div class="nav-primary">class=&quot; nav-primary&quot;</div>
<div class="content-primary">class=&quot;content-primary&quot;</div>
<div class="content-secondary">class=&quot;content-secondary&quot;</div>
<div class="tertiary-content">class=&quot;tertiary-content&quot;</div>
<div class="nav-secondary">class=&quot;nav-secondary&quot;</div>
```

提示：读者可以参见下载文件包内［/第2部分/第4章：文档结构与选择器/css3selector.html］文件。

如果定义如下CSS规则，其显示如图4-36所示。

```
div[class^="nav"] { background:#ff0; }
```

图4-36 E［att^="val"］选择器的应用



注意：此处的匹配与“-”无关，只是简单的字符串匹配。

如果定义如下CSS规则，其显示如图4-37所示。

```
div[class$="primary"] { background: #CF9; }
```

如果定义如下CSS规则，其显示如图4-38所示。

```
div[class*="content"] { background: #0CF; }
```

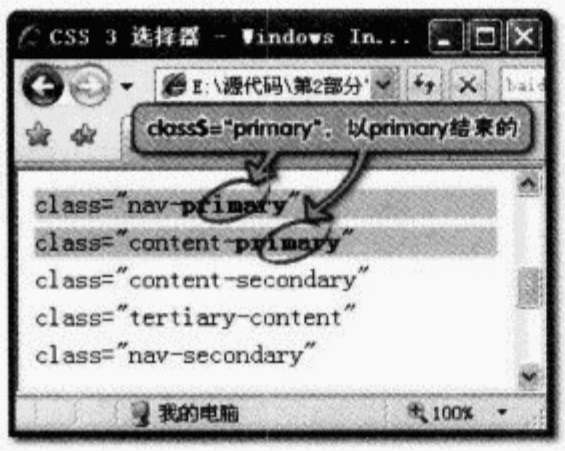


图4-37 E[att$="val"]选择器的应用

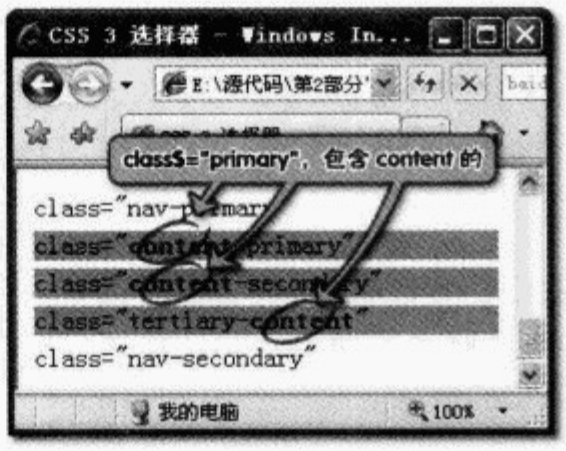
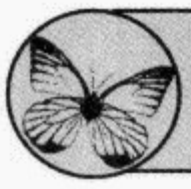


图4-38 E[att*="val"]选择器的应用



提示：目前支持这3种匹配方式的属性选择器的浏览器有IE 7.0、Firefox 2.0、Opera 9.2、Safari等。

4.7.2 普通兄弟选择器

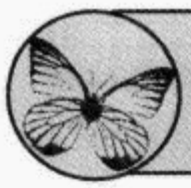
CSS 2内有相邻选择器，只匹配相邻的兄弟元素，而普通兄弟选择器则可以匹配不相邻的兄弟元素，其语法如下：

```
E ~ F { …… }
```

匹配所有F元素，如果它是E的兄弟元素，且在E之后出现。例如有如下代码，其在浏览器内显示如图4-39所示。

```
.test ~ p { background: #FC0; }
<div>
 <div>div 1</div>
 <p>p 1</p>
 <div class="test">div 2, class=&quot;test&quot;</div>
 <p>p 2</p>
 <div>div 3
  <p>div 3 内的p</p>
 </div>
 <p>p 3</p>
 </div>
```

图4-39 普通兄弟选择器的应用



提示：读者可以参见下载文件包内[/第2部分/第4章：文档结构与选择器/css3selector2.html]文件。

4.7.3 结构伪类（Structural Pseudo-Classes）

CSS 3增加了大量的结构伪类，利用文档结构树来实现表现，从而可以减少页面内class属性和ID属性的定义，使得文档更加简洁。但是由于这些伪类在本书完稿之前基本上没有浏览器支持，因此在此只作简单的介绍。

1. E:root

匹配文档的根元素。在（X）HTML中，根元素就是<html>元素。例如右上代码，在（X）HTML文档中，其效果等同于如下代码：

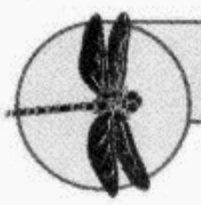
```
:root { border: 1px solid blue; }
```

```
html { border: 1px solid blue; }
```

2. E:nth-child(n)

匹配所有在其父元素中排第n个的E元素。n可以是数字/关键字/公式，例如：

```
tr:nth-child(3) { …… }        /* 匹配所有表格里面排第3的行<tr> */
tr:nth-child(2n+1) { …… }     /* 2n+1，公式，匹配所有奇数行 */
tr:nth-child(odd) { …… }      /* odd：关键字，匹配所有奇数行 */
tr:nth-child(2n) { …… }       /* 2n：匹配所有偶数行*/
tr:nth-child(even) { …… }     /* even：关键字，匹配所有偶数行li */
```



注意：元素的第一个子元素索引为“1”。

利用这个伪类，可以很容易地实现双背景色甚至多背景色表格等效果。

3. E:nth-last-child(n)

匹配所有在其父元素中排倒数第n个的E元素。这个伪类与:nth-child（n）类似，只是以最后一个子元素为起点计算。

4. E:nth-of-type(n)

匹配所有E类型的子元素中的第n个，所有E类型的子元素被分离出来排序。例如：

```
img:nth-of-type(2n+1) { float: right; }
img:nth-of-type(2n) { float: left; }
```

5. E:nth-last-of-type(n)

匹配所有E类型的子元素中的倒数第n个，所有E类型的子元素被分离出来排序。

6. E:last-child

匹配所有E元素，当此元素是其父元素的最后一个子元素时。例如：

```
li:last-child { …… }
```

7. E:first-of-type

根据E的类型匹配所有兄弟元素中第一个此类型元素，类似于“:nth-of-type(1)”。

8. E:last-of-type

根据E的类型匹配所有兄弟元素中最后一个此类型元素。

9. E:only-child

匹配那些是其父元素唯一孩子的E元素。

10. E:only-of-type

根据E的类型，匹配那些在其兄弟里是唯一此类元素的E元素。

11. E:empty

匹配没有子元素的E元素。在DOM树里，元素的内容也属于其子元素节点之一。例如右边代码，p:empty将只匹配第一个“ ”。

```
p:empty {
width:100px;
height:20px;
background:yellow;
}
<p></p>
<p>Text</p>
<p><em></em></p>
```

4.7.4 UI元素伪类和伪元素

UI（User Interface）指用户界面（如表单<form>控制），对其增加的状态伪类，可以使其更具表现力。

1. E:enabled和E:disabled

这两个伪类匹配E的激活（enabled）和无效（disabled）两种状态，例如：

```
input[type="text"]:enabled { border:1px solid blue; }
input[type="text"]:disabled { border:1px solid gray; }
<input name="nikename" type="text" size="16" maxlength="30" />
<input name="age"  type="text" size="10" maxlength="12" disabled="disabled" />
```

则第1个<input>会是蓝色的边框，而第2个<input>则是灰色的边框。

2. E:checked

这个伪类匹配E元素的选中（checked）状态，但是此“选中状态”指的是单选按钮（radio）或者打钩选择框（checkbox），而不是下拉列表框的选中（select）。

```
input:checked { border:1px solid red; }
```

3. E::selection 伪元素

::selection伪元素允许制作者定义选中或者高亮文字的样式。

提示： Firefox 2.0、Opera 9.2和Safari 3.0支持部分UI元素伪类，Safari支持::selection伪元素，读者可以在这些浏览器内测试下载文件包内［/第2部分/第4章：文档结构与选择器/css3selector4.html］文件。

4.7.5 其他伪类

1. 否定伪类E:not（s）

匹配所有类型不是s的E元素，而s是一个简单选择器。简单选择器包括类选择器、 通配选择器（*）、属性选择器、类选择器、ID选择器、内容选择器（未完成）及伪类。例如：

```
p:not(:first-child) { …… }  /* 匹配所有不是第1个子元素的<p>元素 */
```

2. 目标伪类 E:target

<a>元素有一个name属性，用以制作锚点，实现页面内的跳转以及跳转到某个页面的特定的位置，例如：

而:target伪类匹配当点击了跳转的链接**跳转到**这个锚点时的状态，例如右下代码：

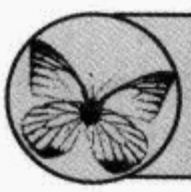
则当页面跳转到“页首”的时候，“页首”的背景色会变为红色。CSS 3对于CSS 2来讲，是一个很大的飞跃，因此其制定的进度也很缓慢，在本书完稿之前，还在制定中。

```
<body>
<a name="top">页首</a>
……
<a href="#top">跳到页首</a>
</body>
```

```
:target { background: red; }
```

4.8 命名规范

ID选择器和类选择器的名称是由制作者制定的，良好的命名规范可以使团队合作更好，无论在项目开发，还是产品维护上都起到了至关重要的作用。应该说命名规范是一种约定，也是程序员之间良好沟通的桥梁。



提示：命名规范并非一成不变，除了一部分是程序语言的要求以外，大部分内容只是一个大家约定俗成的规范。

命名的基本原则是：在CSS 2.1中，标识符（包括元素名、类和ID）只能包含字符[A-Za-z0-9]以及ISO10646字符编号U+00A1及以上，加上连字号“-”和下划线“_”；它们不能以数字开头或者连字号后面跟一个数字。它们还可以包含转义字符加任何ISO 10646字符作为一个数字编码，例如，标识符"B&W?"可以写成"B\&W\?"或"B\26 W\3F"。

同时还有以下几点需要特别注意。

1. 注意名称的语义

同（X）HTML的语义类似，选择器的名称也应该注意其语义，尽量做到命名与“表现”分离，例如有某段文字需要用红色来提醒用户特别注意，那么可以如左定义类的名称，也可以如右定义：

```
<p class="warning">特别注意</p>
```

```
<p class="red">特别注意</p>
```

但是，很明显“warning”要比“red”更加有语义，“red”表示的是其表现为“红色”，而如果某天需要将这些提醒的颜色改为黄色或者其他颜色，那么“red”的名称就会让人迷惑了。

类似的情况比较普遍的，还有在布局方面，例如有两个左右并列的布局板块，左边是菜单，右边是内容，那么如左边定义就比较合适，而右边的命名方式就不太合适：

“left”和“right”也是表示表现的名称，而“menu”和“content”则是表示此板块内容的名称，让人很容易明白。

```
<div id="menu" >这里是菜单</div>
<div id="content" >这里是内容</div>
```

```
<div id="left" >这里是菜单</div>
<div id="right" >这里是内容</div>
```

2. 使用有意义的单词

尽量使用易于辨认的英文单词来做名称，以求清晰易懂，最好不要使用缩写。例如：

页头：header	页脚：footer	标志：logo	滚动：scroll
登录条：loginbar	注册：regsiter	状态：status	
导航：nav	子导航：subnav	菜单：menu	子菜单：submenu
标签页：tab	侧栏：sidebar	页面主体：main	内容：content

续表

文章列表：list	栏目标题：title	提示信息：msg	小技巧：tips
热点：hot	新闻：news	搜索：search	下载：download
服务：service	投票：vote	指南：guild	
广告：banner	版权：copyright	友情链接：friendlink	合作伙伴：partner
加入：joinus			

当1个单词无法完全表现内容的意义时，可以使用多个单词来定义，如news_title、news_list、content_title等。

3. 分段书写方法

当需要用多个单词来表示，单词之间如何连接？一般的书写方法有以下几种。

● 骆驼命名法（camelCasing）：第一个字母小写，随后的每个单词的第一个字母大写。混合使用大小写字母来构成名称，如newsTitle、subMenu。

● 帕斯卡命名法（PascalCasing）：与骆驼命名法类似。只不过骆驼命名法是首字母小写，而帕斯卡命名法是首字母大写，如StudentName。

● 下划线命名法：顾名思义就是在命名中加入了英文的下划线“_”的命名规则，如news_title、news_list。

● 连字符命名法：使用英文连字符“-”来连接单词，如sidebar-menu、content-main。对于属性选择器和CSS 3中的某些选择器，连字符“-”无疑更加具有优势。

上述几种书写方法都得到了广泛的应用，但是，由于有时候页面要结合不同的服务器端程序来实现交互和动态显示等功能，因此，还要遵守相应的程序语言的命名规则。

4.9 选择器综合运用

选择器是CSS规则的基础，通过使用不同类型的选择器和选择器的组合，可以实现对不同元素的匹配，而不需要对每个元素都定义class或者ID。例如有XHTML如下：

```
<body>
<div id="test">
 <p>段落1，<strong>段落1内的strong1 </strong></p>
 <ul>
  <li class="first">列表项1，<strong>列表项1内的strong2</strong>。</li>
  <li>列表项2。</li>
  <li>
      <ol>
       <li>列表项3，<strong>列表项3内的strong3</strong>。</li>
       <li>列表项4。</li>
      </ol>
     </li>
 </ul>
 <div class="sample1">
  <p>段落2，<strong>段落2内的strong4</strong>。</p>
 </div>
</div>
<div id="test2">
 <div class="sample1">
  <p>段落3，<strong>段落3内的strong5</strong>。</p>
 </div>
</div></body>
```

除了可以直接使用类型选择器（div、p、li、strong等）、ID选择器（#test、#test2）、类选择器（.first、.sample1）之外，还可以通过包含选择器进行更精确的匹配，如图4-40所示。

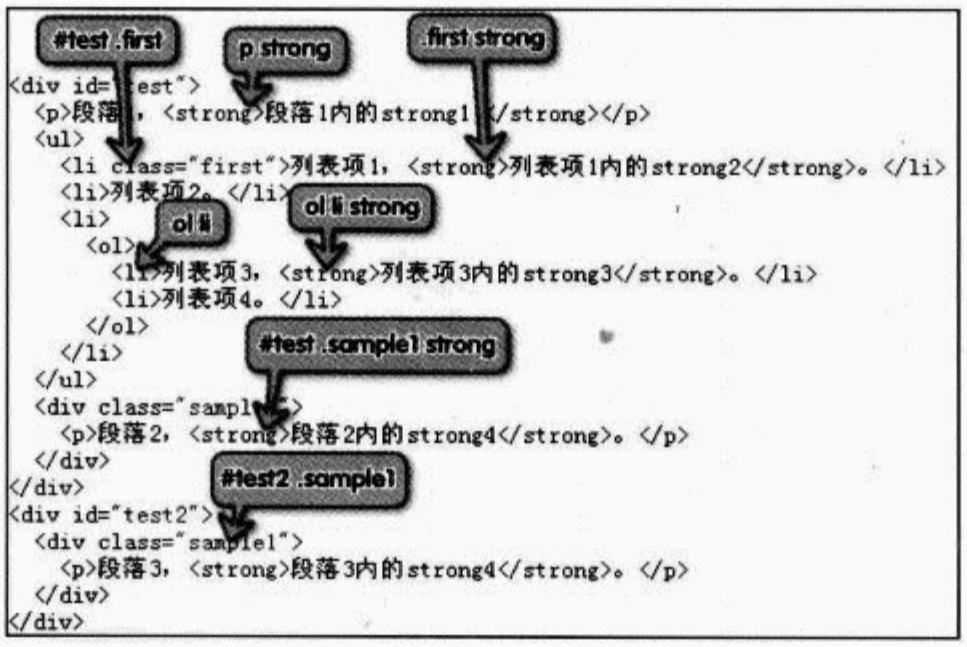


图4-40　选择器的综合运用

选择器的组合是很丰富的，在实际应用中，要注意不要过于复杂，只要能生效即可，例如，在图4-40中，要针对<ol>内的<li>元素，若使用左边代码，可达到目的，则不需要写右边代码：

```
ol li { …… }
```

```
#test ul li ol li { …… }
```

如果能通过包含选择器来匹配元素，一般情况下就没有必要为元素添加类或者ID，这样也可以使（X）HTML文档简洁易读。

第5章
单位和值

CSS声明由属性和值组成（如图3-5所示），而不同的属性又有不同的值，因此本章将详细介绍这些值以及单位的知识。

5.1 颜色<color>

颜色给人的感觉最直接，当一个页面打开的时候，访问者首先感觉到的往往就是页面内的颜色。颜色值可以是一个关键字或一个RGB数字。

注意：尽管颜色给文档带来可观的信息，并使文档更加容易阅读，但是也要考虑到某些颜色的组合会给色盲的读者带来困难。如果使用背景图片或背景颜色，则应该相应地调整前景颜色。

5.1.1 颜色关键字

而由于显示设备、操作系统、显示卡以及浏览器的不同，即使是一模一样的页面颜色也会有不尽相同的显示效果。当前使用最为广泛的操作系统主要是Windows、MAC OS、Linux、UNIX这几种，而这些操作系统内置的调色板之间存在着或多或少的差异，所以即使使用同一台显示器显示同一个颜色，显示出的效果也会略有不同。

计算机所使用的显示卡的优劣也会直接影响颜色的显示效果，例如，分别使用支持8位真彩色（256种颜色）和24位真彩色（1600万种颜色）显示卡的两台计算机显示同一个图像的效果会有很明显的差距。而用户要浏览网页内容时，必须使用相应的网页浏览工具，而不同的浏览器内置了不尽相同的调色板，所以浏览器的不同也会影响颜色的显示效果。

颜色名称的关键字列表如图5-1所示。读者可以参见下载文件包内［/第2部分/第5章：单位和值/ colorkeywords. html］文件。

其中“orange”是CSS 2.1新增加的关键字，在CSS 2中只有16个颜色关键字。在之前的章节内，很多例子都使用到了颜色关键字，例如：

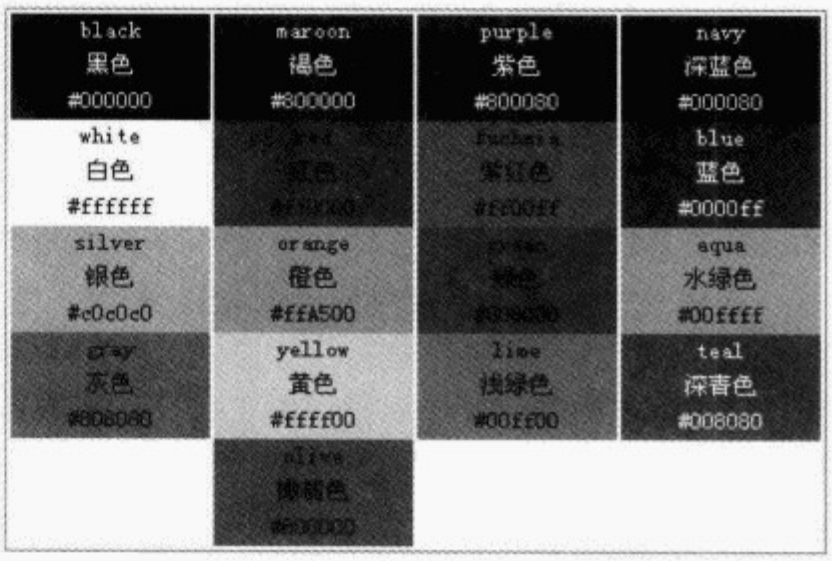


图5-1　颜色值关键字

```
h1 { color: maroon }
p { color: olive }
```

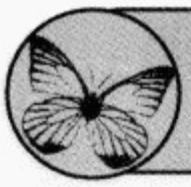
提示：实际上，还有更多的颜色关键字可以被不同的浏览器识别，但是由于浏览器内置调色板的不同，可能显示出来的颜色不尽相同。打印机对于颜色的表现更加复杂，涉及打印机的分辨率、墨水的类型、型号等因素。

这些颜色远远不能满足设计的需要，因此，更多时候CSS中的颜色是使用RGB颜色来定义。

5.1.2 RGB颜色

日常见的白光，实际由红（Red）、绿（Green）、蓝（Blue）3种波长的光组成，这3种光是自然界中所有颜色的基础，光谱中的所有颜色都是由这3种光的不同强度构成。

显示器显示颜色的原理也是这样，因此颜色可以使用RGB颜色模型来定义。RGB颜色有以下4种表示方式。



提示： 本小节示例读者可以参见下载文件包内［/第2部分/第5章：单位和值/ color.html］文件。

1. 百分比

图5-2 百分比RGB颜色

以从0~100%的百分比值来表示颜色，如图5-2所示。百分比值越大表示光越强，因此：

```
rgb(100%, 100%, 100%) = 白色
rgb( 0%, 0%, 0%) = 黑色
rgb( 100%, 0%, 0%) = 红色
rgb( 0%,100%, 0%) = 绿色
rgb( 0%, 0%, 100%) = 蓝色
```

百分比值如果超过100%，会被修正为100%，如果设定负值，则会被浏览器修正为0，例如：

```
rgb(-40%, 50%, 100%) => rgb(0, 50%, 100%)
rgb(1240%, 2250%, 10000%) => rgb(100%, 100%, 100%)
```

也可以设定小数百分比值，但是如果浏览器不支持小数百分比值，则小数点有可能被浏览器忽略，例如左边代码。浏览器可能会把小数修正为与其最近的整数，因此左边的设定会被修正为右边代码：

```
rgb(25.5%,40%,98.6%)
```

```
rgb(26%,40%,99%)
```

2. 数字颜色

数字颜色的语法和百分比类似，只是用0~255之间的数字替换百分比值，例如左边代码。超出范围的值也同样会被修正，不过某些设备（如打印机），和sRGB有不同的设备范围；某些超出0~255的范围的颜色是可以呈现的（在设备范围之内），而其他一些在0~255范围内的颜色如果在设备范围之外，则会被加以修正，如右边代码。

```
rgb(255, 0, 0) = rgb(100%, 0%, 0%)
```

```
p { color: rgb( -200, 50, 3000); } => p { color: rgb(0, 50, 255); }
```

数字颜色不允许有小数，这是与百分比不同的地方。

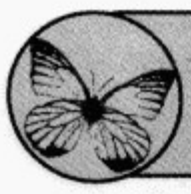
3. 十六进制颜色

目前应用最为广泛的，还是十六进制颜色，如图5-3所示。十六进制以2位表示1个颜色，因此，十六进制的“FF”等于十进制的“255”，而十六进制的“00”等于十进制的“0”。

在［3.3.4 缩写］一节内曾简要介绍了颜色的缩写，由于每2位表示一种颜色，因此类似于“#006600”这种两位重复的颜色值可以缩写为“#060”。而“#808080”则不能缩写。浏览器会自动复制数值，例如“#123”会被复制为“#112233”，而“#880”则是“#888800”，而不是“#808000”或其他的值。

```
p { background-color: #ccccrr; }
```

浏览器不会对十六进制值进行修正，因此错误的值可能会使本条属性被忽略，例如如上代码在浏览器内的显示如图5-4所示。



提示： 十六进制中的英文字母“A”到“F”不区分大小写，“#FF00CC”和“#ff00cc”是相同的。

图5-3　十六进制颜色

图5-4　错误的十六进制颜色在浏览器内的处理方法

5.1.3 关键字transparent

CSS 1中可以指定背景颜色（background-color）为“transparent（透明）”，而CSS 2中，还可以为边框颜色（border-color）设定“transparent”值，例如右上代码：

对于背景颜色来说，默认的值就是“transparent”，而边框的默认值是由color属性指定的颜色值，但是由于大部分元素边框样式（border-style）默认为“无（none）”。但是，有些浏览器不支持边框颜色的“transparent”值，例如右下代码在IE 6.0/7.0内的显示如图5-5所示。

```
div { background-color: transparent; }
p { border-color: transparent ; }
```

```
.test {
background: yellow;
}
.test div {
color: blue;
border:5px solid transparent;
}
<div class="test3">
  <div>border:5px solid transparent;</div>
</div>
```

图5-5　“transparent”在浏览器内的处理方法

5.1.4 网页安全色（Web-safe Colors）

网页中的颜色会受到各种不同环境的影响。即使网页使用了非常合理、漂亮的配色方案，但是如果每个人浏览时看到的效果都各不相同，那么这个配色方案的意愿就不能够非常好地传达给浏览者。

那么要通过什么方法才能解决这一问题呢？答案就是216网页安全色。216网页安全色是指在不同硬件环境、不同操作系统、不同浏览器中都能够正常显示的颜色集合（调色板），也就是说这些颜色通过任何终端浏览用户显示设备上的现实效果都是相同的。所以使用216网页安全色进行网页配色可以避免原有的颜色失真问题。

网络安全色是当红色（Red）、绿色（Green）、蓝色（Blue）颜色数字信号值（DAC Count）为0、51、102、153、204、255时构成的颜色组合，它一共有6×6×6 = 216种颜色（其中彩色为210种，非彩色为6种），如图5-6所示。

提示：读者可以参见下载文件包内［/第2部分/第5章：单位和值/transparent.html］文件。

十进制的“0、51、102、153、204、255”转换为十六进制即“00、33、66、99、CC、FF”，因此也可以缩写为“0、3、6、9、C、F”，如#036（#003366）、#FFF（#FFFFFF）、#CCC（#CCCCCC）等。

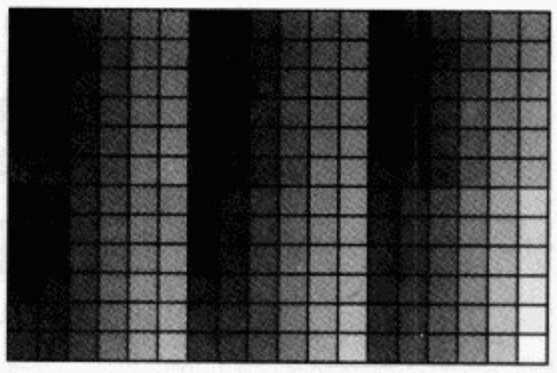
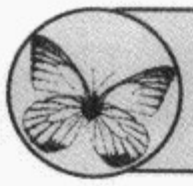


图5-6 216网络安全色

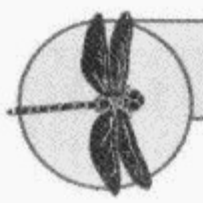


提示： 关于网页安全色更多的内容，可参见http://www.ddcat.net/sheji/peise/3/menu03_01.htm（中文）。

5.2 整数值<integer>和实数值<number>

某些值的类型可以包括整数值<integer>或实数值<number>。实数和整数都只以十进制符号表示。整数值包括一个或多个0~9的数字。实数值可以是一个整数值，或者在英文点号“.”后接零个或多个数字。整数和实数可以前缀一个“+”或“-”来表示符号。例如：

```
div { z-index: 100; }    /* 整数值 */
p { line-height: 1.5;}   /* 实数值 */
```



注意： 很多接受整数和实数作为其值的属性实际上会有取值范围的限制，通常是非负值。

5.3 长度<length>

长度值是指水平或垂直方向的度量。为了元素能在页面内正确地显示，往往需要设定元素的大小、边距等，这些都属于长度。

5.3.1 格式

长度值<length>的格式为：一个可选的符号字符（“+”或“-”，“+”是默认的符号），紧接在后的是一个实数值，再紧接在后的是一个单位标识符（如px、in等），如果值为“0”，单位标识符可以省略，如右边代码：

```
div {
pdding: 2.6em;
width: 400px
margin-left: -30mm;
}
```

某些属性允许负的长度值，例如边距（margin），不过这将使格式化模型变得复杂，并可能伴随着与实现显示其效果相关的限制。如果某些属性不支持负的长度值，它会被转换到最接近的可以被支持的值。

5.3.2 长度单位

对于计算机的屏幕设备而言，像素（Pixel，px）是一个最基本的单位，就是一个点。其他

所有的单位，都和像素成一个固定的比例换算关系。所有的长度单位基于屏幕进行显示的时候，都统一先换算成为像素的多少，然后进行显示。所以，就计算机的屏幕而言相对长度和绝对长度没有本质差别。任何单位其实都像素，差别只是比例不同。

如果把讨论扩展到其他输出设备，比如打印机，基本的长度单位可能不是像素，而是其他的和生活中的度量单位一致的单位了。有两种类型的长度单位：绝对单位和相对单位。

1. 绝对单位

绝对单位有以下几种。

- in：英寸，1英寸=2.54厘米。
- cm：厘米，1厘米=0.394英寸。
- mm：毫米，1毫米=0.1厘米。
- pt：磅，标准的印刷上的量度，广泛应用于打印机、排字机以及字处理软件等；72磅=1英寸。
- pc：pica（12点活字），另一种印刷术语，1pica=12磅，6pica=1英寸。

例如：

```
h1 { margin: 0.5in }          /* 英寸 */
h2 { line-height: 3cm }       /* 厘米 */
h3 { word-spacing: 4mm }      /* 毫米 */
h4 { font-size: 12pt }        /* 磅 */
h5 { font-size: 1pc }         /* picas */
```

绝对长度单位是对于输出设备（Output Device）而言的，例如pt，这是文字排版工具（例如Microsoft Word）中常用的字体单位，同一篇文档打印在纸面上的结果是一样的。

如果指定的长度不被支持，用户端会将其近似为实际值。

2. 相对单位

相对长度单位规定一个长度相对于另外一个长度属性。相对长度在不同的设备上、甚至相同的设备而不同的设定上（例如不同显示分辨率的显示器），实际显示出来的长度可能会不相同。相对单位包括以下几种。

- em：相对于字体的大小（font-size）。
- ex：相对于字体的小写字母“x”的高度，即使字体中没有x字母，ex还是会有定义。
- px：相对于浏览设备的像素点，在大多数情况下是指计算机的显示器。

例如：

```
h1 { margin: 0.5em }
div { margin: 1ex }
p { font-size: 12px }
```

浏览器会根据指定的数字值和单位来计算显示中的实际值，例如：

```
h1 { line-height: 1.2em } /* <h1>的行高是<h1>的字体尺寸的1.2倍，如果字体尺寸是10px，则行高为10px×1.2=12px */
h1 { font-size: 1.2em } /* <h1>的字体尺寸是其继承来的字体尺寸的1.2倍 */
```

em和ex是和字体尺寸（font-size）相关的，因此如果有如右设定：

```
p { line-height: 2em; }
```

如果设定不同的字体尺寸，行高也会随着变化，如图5-7所示。读者可以参见下载文件包内[/第2部分/第5章：单位和值/values.html]文件。

ex是以字体的小写x字母的高度为标准的，而相同字号大小的不同字体的小写x高度不一定相同，如图5-8所示。

理论上虽然如此，但是实际上，很多字体都没有给定具体的小写字母x的值，因此很多用户

端会认为1em=2ex。

图5-7 em与字体大小

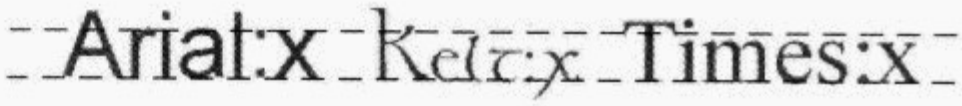


图5-8 不同字体的小写x字母高度不同

相对于以上几个单位，读者最为熟悉的还是px。那么px为什么会是相对单位？Pixel是由Picture和Element这两个字母所组成的，是用来计算数位影像的一种单位。如果把一张图片在屏幕上放大数倍，会发现图片其实是由许多有色彩的小方点所组成，这些小方点就是构成影像的最小单位“像素”（Pixel）。那么多少个像素等于1英寸呢？这与显示器的PPI有关。

提示：DPI原来是印刷上的记量单位，意指每平方英寸上，所印刷的网点数（Dot Per Inch），但在电脑与印刷结合，数位输入、输出设备快速发展的同时，大多数的人已将数位影像的解析度用DPI表示，但印刷时计算的网点大小（Dot）和电脑的显示像素（Pixel）并不相同，所以较专业的人士，会用PPI表示数位解析度，藉以区分二者。

Windows系统默认设定为96ppi，即1英寸长度内有96个像素，Mac OS系统则设定为72 ppi，而CSS 2规范推荐的却是90ppi。实际应用中，浏览器往往会使用显示器的设置，但是在其他设备上，结果就不尽相同了。图片的尺寸一般都使用像素来表示。

5.3.3 应用

使用绝对单位定义的长度值一般是用在打印设备的定义上，而对于电脑显示器来讲，其显示的效果可能会与设计的原意相差甚远，因此对于显示器一般建议使用相对单位定义的长度值或者百分比值。

而目前很多设计者更喜欢使用px作为单位来制作页面，觉得这样比较容易控制效果，但是在一些情况下，使用em或者百分比值可以让页面更具“弹性”。

例如，对于视力不好的访问者，设定了比较大的字体，而如果采用px作为行高，那么大的文字由于小的行高可能会叠加在一起，或者溢出到元素框之外。

5.4 百分比<percentage>

百分比值<percentage>为一个可选的符号字符（“+”或“-”，“+”是默认的符号），紧接在后的是一个实数值，紧接在后的是“%”。百分比值总是相对于另外一个值，如长度。允许百分比值的每一个属性也定义了百分比所参考的值。这个值可以是以下几种情况：

- 同一元素的另外一个属性的值；

- 其祖先元素的属性值；
- 或者格式化上下文的值（如包含块的宽度）。

如果根元素的属性指定了百分比值，而百分比值又被定义为参考某个属性的继承值，那么结果值就是百分比乘以那个属性的初始值。

子元素通常继承其父元素的计算值，例如，下列代码其在浏览器内的显示如图5-9所示。读者可以参见下载文件包内［/第2部分/第5章：单位和值/values2.html］文件。

```
div {
background : #FF9;
font-size : 10pt;
line-height : 120%;   /* 相对于font-size的120% = 12pt */
}
p {
font-size : 20pt;
background : #FF6;   /* 背景色黄色部分为<p>的行高 */
}
<div><p>p的行高继承div的12pt</p></div>
```

图5-9　<p>继承的是 行高的计算值

5.5 关键字

关键字（keywords）以标识符（identifier）的形式出现。关键字不可以放置在英文双引号或英文单引号之间。例如右边的写法都是错误的，具体的关键字将在各属性内介绍。

```
width: "auto";
background: "red";
border: 'none'
```

5.6 字符串<string>

字符串<string>可以包含在英文双引号“”或单引号“‘”中。在双引号对中不可以再出现双引号，除非将其转义（如“\””或“\22”），单引号与此类似（如“\’”或“\27”）。例如下面右边代码：

```
"this is a 'string'"      /* 双引号内可以包含单引号 */
"this is a \"string\""    /* 双引号内的双引号要转义 */
'this is a "string"'      /* 单引号内可以包含双引号 */
'this is a \'string\''    /* 单引号内的单引号要转义 */
```

一个字符串不可以直接包含新行。要在字符串中包含新行，可以使用转义字符“\A”或者“\00000a”（十六进制的A在Unicode中表示换行字符，在CSS中表示通用的“newline”）。例如下面左边代码相当于右边代码：

```
"this is a\A long string"
```

```
"this is a
long string"
```

为了美观或其他原因，可以将字符串分成几行。不过在这种情况下，换行本身要加以转义。例如，右边两个选择器是等效的：

```
A[TITLE="a not s\
o very long title"] { ….. }
A[TITLE="a not so very long title"] { …… }
```

5.7 URL + URN = URI

读者可能都比较熟悉URL，但对URI和URN这两个词就比较陌生了。URI、URL和URN是识别、定

位和命名互联网上的资源的标准途径。Web上地址的基本形式是URI，它代表统一资源标识符（Uniform Resource Identifilter）。URI分为URL和URN这两大类。

URL（Uniform Resouce Locator，统一资源定位符），目前URI的最普遍形式就是无处不在的URL或统一资源定位器。URN（Uniform Resource Name，统一资源名称），是URL的一种更新形式，不依赖于位置，并且有可能减少失效连接的个数。但是其流行还需假以时日，因为它需要更精密软件的支持。

URL与URN的不同之处在于前者不仅标识资源，而且还指出了访问资源的方式，比如采用何种协议（如HTTP、FTP等），而URN则没有。

用来在CSS属性值中指定URI的函数符号是“url()”，例如，以下定义都是正确的：

```
body { background: url("http://www.ddcat.net/images/ddcat_96.jpg"); } /* 使用双引号 */
body { background: url('http://www.ddcat.net/images/ddcat_96.jpg'); }  /* 使用单引号 */
body { background: url(http://www.ddcat.net/images/ddcat_96.jpg); }    /* 不使用引号 */
```

括号“()”、逗号“,”、空白字符、单引号和双引号如果出现在URI中，则必须用反斜杠转义，如“\(”、“\)”、“\,”等。根据URI的类型，也可以将上述字符写成URI转义，如“(” = %28、“)” = %29等。

为了创建不依赖于资源的绝对位置的模板样式表，用户可以使用相对URIs。相对URI根据基准URI解析为完全URI。对于CSS样式表，基准URI是样式表的位置，而不是引用该样式表文件的源文档的位置。

例如，假定如左的规则，定位在由如右URI指定的样式表中：

```
body { background: url("banner.jpg"); }
```

http://www.ddcat.net/style/basic.css

则源文档<body>的背景将由如右URI指定的图片生成，注意用户端在处理指定不可获得或不适用的资源的URI时，方法可能不同。

http://www. ddcat.net/style/banner.jpg

5.8 其他值

CSS 2规范内增加了一些新的属性以及与其相关的值，其中大部分是应用在显示器以外的设备上的，例如屏幕阅读机等，在此作简要的介绍。

5.8.1 记数器<counter>

记数器<counter>表示为标识符。CSS 2中，记数器值<counter>只可以由content属性引用。

例如，下面的样式表为每一章<h1>中的<p>进行编号。编号的方式是以罗马数字编号，后接一个点和一个空格：

```
p {counter-increment: par-num}
h1 {counter-reset: par-num}
p:before {content: counter(par-num, upper-roman) ". "}
```

提示： content属性只能在:before和:after伪元素内使用。关于计数器请参见本书［12.2 生成的内容］一节。读者可以参见下载文件包内的［/第2部分/第5章：单位和值/values3.html］文件。

5.8.2 角度<angle>

角度值<angle>使用在语音样式表中。其格式为一个可选的符号字符（“+”或“-”，“+”是

默认值），紧接在后的是一个实数值，紧接在后的是一个角度单位标识符。角度单位标识符有3种：deg（度）、grad（梯度）和rad（弧度）。

角度值可以是负数。用户端会将它们规范到0~360度，如-10度和350度是一样的。例如，一个直角是“90度”或“100梯度”或“1.5707963267948 97弧度”。

5.8.3 时间<time>

时间值<time>使用在语音样式表中。它们的格式是一个实数值，紧跟在后的是一个时间单位标识符。时间单位标识符有ms（毫秒）和s（秒）。时间值不可以为负数。

5.8.4 频率<frequency>

频率值<frequency>使用在语音样式表中。它们的格式是一个实数值，紧跟在后的是一个频率单位标识符。频率单位标识符有Hz（赫兹）和kHz（千赫兹）。频率值不可以是负数。例如，200Hz（或200hz）是低音，而6kHz（或6khz）是高音。

5.9 不支持的值的处理

如果遇到了不支持的值，浏览器会忽略掉该样式声明。例如：

```
h3 {
display: inline;
display: run-in;
}
```

如果浏览器支持“run-in”这个值，则display属性的值为“run-in”，如果浏览器不支持，则display属性的值为“inline”。

第6章 字体

为了达到对文字内容的美化或者突出等效果，设计者往往通过改变文字的字体或者粗细等来实现，本章将讲解和字体、字号大小有关的CSS属性。

样式是从印刷而来，因此CSS中也沿用了很多印刷术语，如下所示。

- 字体（Font）：是一系列字号、样式和磅值相同的字符（如10磅黑体Palatino）。现多被视为字样的同义词。
- 字面（Face）：是所有字号的磅值和格式的综合。
- 字体集（Font family）：是一组相关字体。
- 磅值（Weight）：用于描述字体粗度。典型的磅值，从最粗到最细，有极细、细、book、中等、半粗、粗、较粗、极粗。
- 样式（Style）：字形有3种形式，Roman type是直体，Oblique type是斜体，Utakuc type是斜体兼曲线（比Roman type更像书法体）。

6.1 字体集：font-family属性

英文字体一般都是由若干系列字体组成，称为字体集（Font Family，或译“字体家族”）。例如“Times New Roman”、“Arial”或者“Courier”。每一字体集又包括黑体、粗体、斜体等风格的字体成员，例如对于Courier字体集，其成员由“Courier New”、“Courier New bold”、“Courier New Italic”组成。在字体名称中的“Bold”、“Medium”、“Italic”等表示其变形状态。

6.1.1 语法

font-family属性定义元素内文字以何种字体来显示，具体定义列表如下：

语法	font-family : [[<字体名>| <字体系列名>] [, <字体名>| <字体系列名>] *] | inherit
说明	设定元素内文本的字体名称
值	字体名：字体的名称。 字体系列名：某个字体系列
初始值	跟浏览器的设置有关
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

定义多个字体时，字体名以英文逗号“,”分隔，如右：

```
p { font-family : Georgia, Times, serif; }
```

含有空格的字体名要用英文引号括起，如右：

```
p { font-family : "Times New Roman"; }
```

对于中文字体，也要用英文引号括起，如右：

```
p { font-family : "宋体"; }
```

双引号（"）和单引号（'）都可以使用，但是在使用style属性为元素设定内联式样式表的时候要注意引号的嵌套问题，例如：

```
<p style="font-family: '宋体'; ">本段落为宋体字
</p>
```

此处style属性值使用双引号，则字体名要使用单引号。

在CSS中设定字体的局限性很大，因为通过CSS设定的字体是否能正常显示，要看访问者的计算机内是否安装了该种字体的字库文件，而不是设计者的计算机内的字库文件或者服务器的字库文件。而对于大多数访问者来说，其计算机内往往只有操作系统自带的几种字库文件，因此设定字体的时候，最好是使用大多数计算机上可能有的常见的TrueType字体，例如，Arial、Times New Roman、Courier、Verdana和Century Gothic都是常见的字体，中文操作系统里面Windows自带宋体、黑体及楷体。

说明：TrueType字体是由AppleComputer公司和Microsoft公司联合提出的一种数学字形描述技术，它采用几何学中二次B样条曲线及直线来描述字体的外形轮廓，其特点是：既可以作打印字体，又可以用作屏幕显示，无论放大或缩小，字符总是光滑的，不会有锯齿出现。

字体名并不是随便输入的，字体在各个操作平台之上叫的名称可能会有变化。例如，Courier字体在MAC机上叫作Courier New，在一台计算机上可能叫Italic的字体在另一台计算机上可能就叫作Oblique。

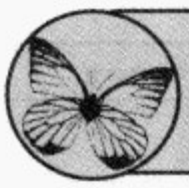
如何确定计算机对某种字体的确切名称？这取决于计算机所用的操作系统平台。

● Windows的用户：使用应用软件，例如Microsoft Word中的字体菜单中所列出的确切的字体名称。

● Mac机用户：不要相信应用软件列出的字体名称，而应该打开系统文件夹［system folder］，按照其中对字体的拼写在样式表中使用字体名称。

6.1.2 常用字体系列

有一些字体的外观是很相似的，这点在英文字体上尤为明显，而这些外观类似的字体往往是一个字体系列，例如，Times字体系列、网页设计中很著名的04字体系列等。除了这些设计者定义的字体系列以外，在CSS中也定义了几个一般的字体系列（generic font family）。



提示：本小节的示例页面请参见下载文件包内［/第2部分/第6章：字体/ font-family.html］文件。

1. Serif字体

Serif字体是一种成比例且有衬线（Serif）的字体。成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；衬线是指加在字母上的装饰性的细线，这种字体强调了字母笔画的开始及结束，因此较易前后连续性的辨识，比较容易阅读辨认。

英文中的“Times New Roman”字体和中文的宋体、细明体（繁体中常用）就属于Serif字体系列，如图6-1所示。

图6-1 Serif字体的特点

2. Sans Serif字体

Sans Serif字体是成比例但没有衬线的字体，例如英文的“Arial”字体和中文的“黑体”，就属于Sans Serif字体，如图6-2所示。

这种字体比较适合作标题字，而不适合用在大面积文字上，因为容易造成字母辨认的困扰，常会有来回重读及上下行错乱的情形。在小字体的场合，通常Sans Serif字体比Serif字体更清晰。

图6-2 Sans Serif字体的特点

3. Monospace字体

Monospace是一种无比例的字体，它通常用于模拟打字机打出的文本，或者用来显示程序代码等，这种字体的字母是等宽的，比较容易阅读辨认，但是美观性比较差，英文的“Courier New”字体就是这种字体，如图6-3所示。

图6-3 Monospace字体的特点

4. Cursive字体

Cursive是一种模拟人手写的草体文字的字体，一般都是由曲线和比衬线更复杂的线条构成，例如英文“Comic Sans”字体就属于Cursive字体，如图6-4所示。

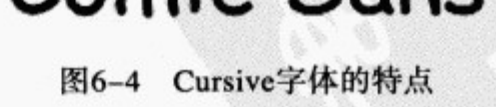


图6-4 Cursive字体的特点

Cursive字体通常很难在屏幕上阅读,一般情况下不鼓励使用，一些浏览器甚至不支持这种字体。

5. Fantasy字体

Fantasy字体包含那些既不能通过某种单一的特征来定义，又不能归于以上4类字体的字体，例如英文的Ventilate字体。由于这是一个包罗广泛的范畴，所以更多是通过说明它不属于某个范畴（其他4种字体集）来定义，而不是定义它属于某个范畴。

例如，Windows系统的“WingDings”字体和“Symbol”字体，本身就是一些符号字体，因此就属于Fantasy字体系列，如图6-5所示。

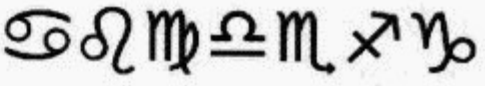
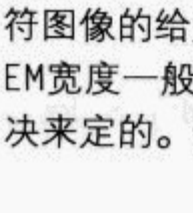


图6-5 Windows系统的WingDings字体

6.2 字体尺寸：font-size属性

字体尺寸是一个很重要的属性，很多其他属性的设定，都可以以字体大小为基础，以达到弹性设计的目的，例如：

```
p { width : 30em; } /* 段落宽度为20个字宽 */
```



提示：本小节的示例页面请参见下载文件包内［/第2部分/第6章：字体/ font-size.html］文件。

6.2.1 语法

font-size属性定义元素内文字的尺寸大小，具体定义列表如下：

语法	font-size : <绝对尺寸> | <相对尺寸> | <长度> | <百分比> | inherit
说明	设定元素内文字的尺寸
值	绝对尺寸：包括xx-small（最小）、x-small（较小）、small（小）、medium（正常）、large（大）、x-large（较大）和xx-large（最大）7个关键字。 相对尺寸：包括larger（相对增大）和smaller（相对减小）2个关键字。 长度：长度值，不可为负值。 百分比：基于父元素中字体的尺寸
初始值	medium
继承性	继承
适用于	所有元素
媒体	视觉
计算值	绝对长度值

例如：

```
body { font-size : small; }
h1 { font-size : 1.5em; }
strong { font-size: larger; }
```

设定的字体大小与浏览器显示出来的实际大小，取决于字的“全身方框（em square）”。“全身方框”原是一印刷专用名词，专指铅字刻字面的单位尺寸。在TrueType字体中，指用于显示字符图像的给定面积。文字EM高度是统一不变的。对西文来讲，EM的宽度可以调整。对于中文来讲，EM宽度一般是固定的且与高度相等。因此文字实际在浏览器内看上去的效果，是由字体的设计者决来定的。例如，同样设定为24px大小的不同字体的英文，其在浏览器内实际显示如图6-6所示。

全身方框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，如图6-6所示。对于某些字体来说，一些字符比默认的基线间的距离更高是很有可能的，如图6-6中第1种字体。而font-size属性的作用就是提供某个尺寸的全身方框。

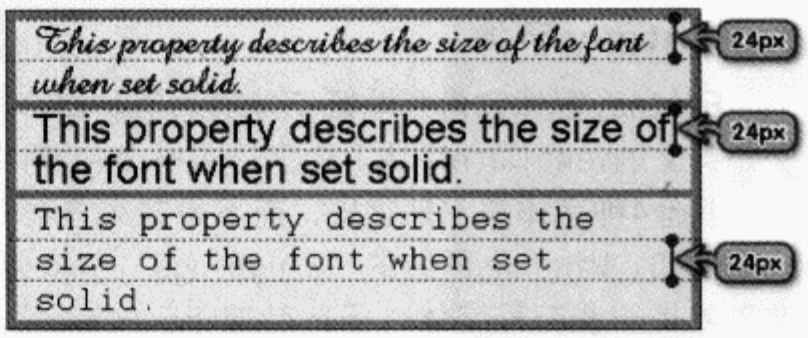


图6-6 不同的字体在浏览器内实际显示出的大小不同

6.2.2 绝对尺寸

绝对尺寸包括xx-small、x-small、small、medium、large、x-large和xx-large这7个关键字。在这7个关键字之间存在一个固定的缩放因子，这个缩放因子在CSS 2中规定为1.2，而在CSS 1中则为1.5。例如，如果“medium”为12px，则“large”就为14.4px（12px×1.2）。

CSS中规定默认的字体尺寸为“medium”，不过浏览器有可能不按照这个设定，例如IE 5.5就认为默认字体大小为“small”，如图6-7所示。

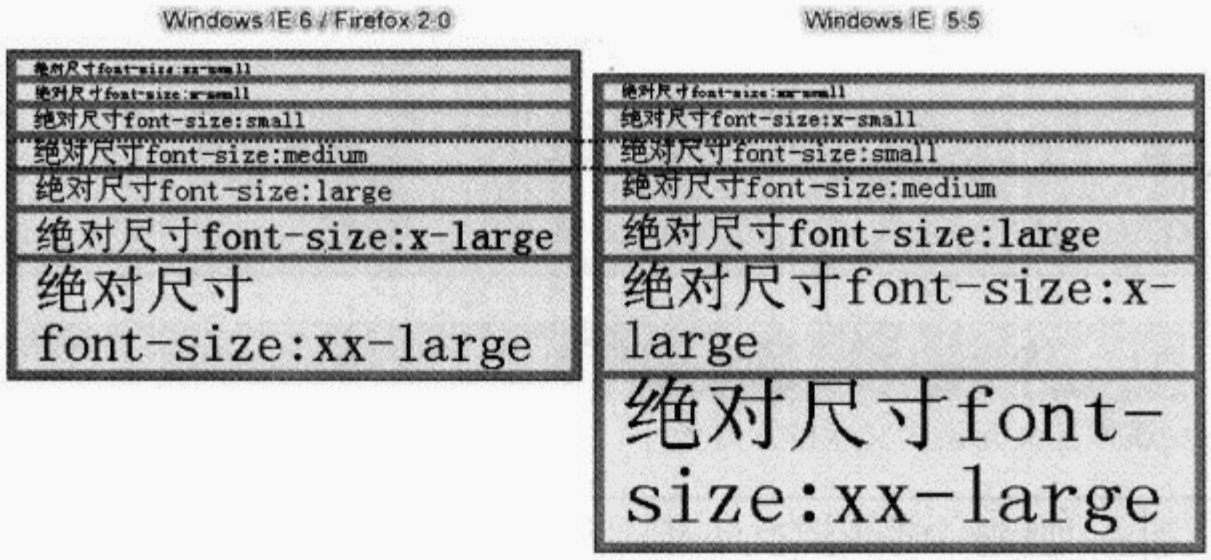


图6-7 浏览器对于默认字体大小的不同处理

因此实际的字体尺寸还要看浏览器的具体设置，例如，改变Firefox 2.0浏览器的设置，将影响字体的大小，如图6-8所示。

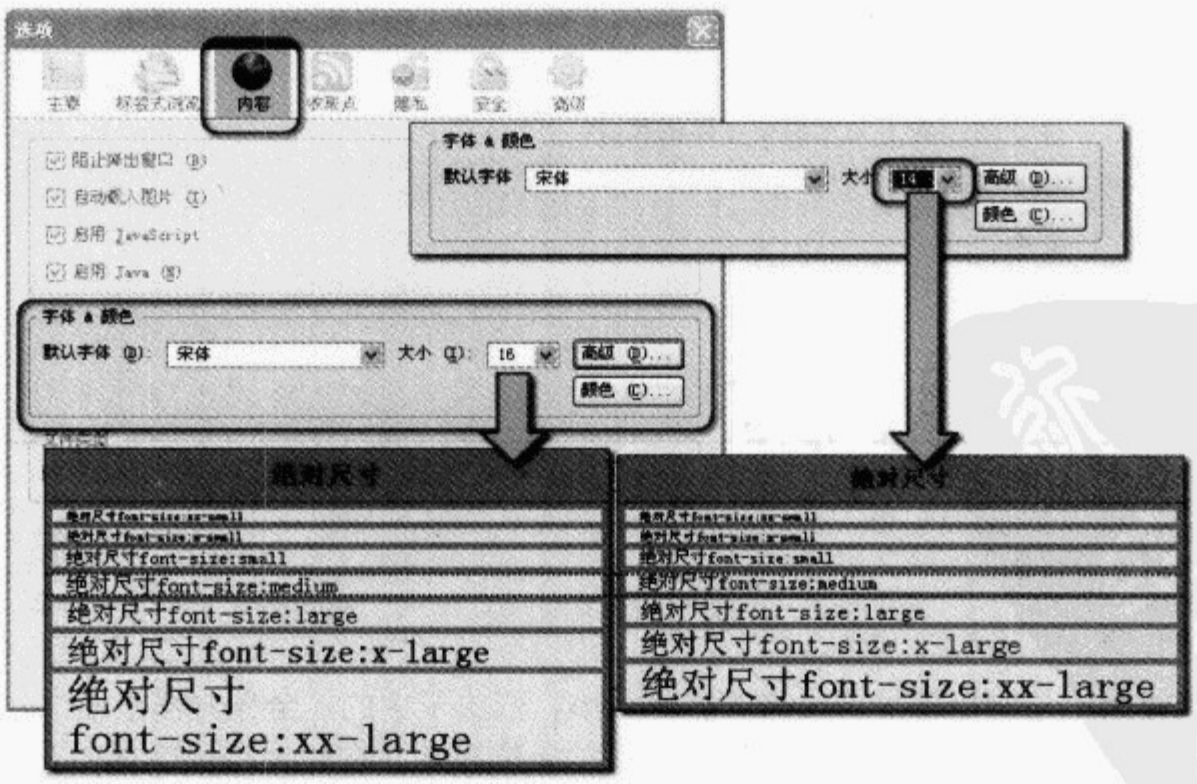


图6-8 改变浏览器的设置对绝对尺寸的影响

目前大部分浏览器的默认字体大小为16px，因此CSS 1和CSS 2关键字尺寸换算对照如下表所示。

关键字	CSS 1缩放因子：1.5	CSS 2缩放因子：1.2
xx-small	5px	9px
x-small	7px	11px
small	11px	13px
medium	16px	16px
large	24px	19px
x-large	36px	23px
xx-large	54px	28px

因此可以为<body>设定：

```
body { font-size: small; }
```

在大部分的浏览器内，都将显示13px的文字。

6.2.3 相对尺寸

相对尺寸的2个关键字larger和smaller比较好理解，它们使元素的字体尺寸相对于父元素的尺寸变大（larger）或者变小（smaller），而且使用和绝对尺寸一样的缩放因子，如图6-9所示。

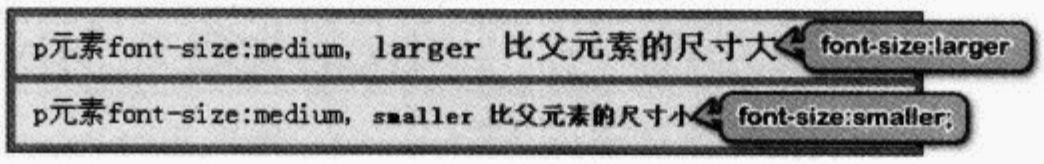


图6-9　相对尺寸

当元素嵌套的时候，其字体大小都是以其父元素字体大小的计算值为基准变大或缩小，例如有代码如下，则其显示如图6-10所示。

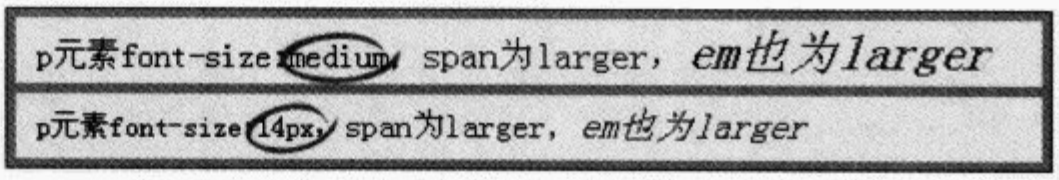


图6-10　相对尺寸的嵌套

```
#fontSize3 p {
font-size : medium;
}
#fontSize3 span {
color : #00f;
font-size : larger;
}
#fontSize3 span span {
color : #f00;
}
#fontSize3 p.font4 {
font-size : 14px;
}
<div id="fontSize3">
  <p>p元素font-size:medium，<span>span为larger，<em>em也为larger</em></span></p>
  <p class="font4">p元素font-size:14px，<span>span为larger，<em>em也为larger</em></span></p>
</div>
```

6.2.4 百分比和em

百分比和em尺寸的计算，也是以父元素的字体尺寸为基础的，而且，“1em”和“100%”的大小是一样的。例如有代码如下，则其显示如图6-11所示。

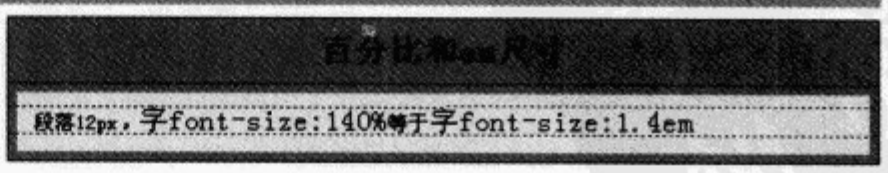


图6-11　百分比尺寸和em尺寸

```
p { font-size: 12px; }
.font5 { font-size: 140%; }
.font6 { font-size: 1.4em; }
```

```
<p>段落12px，<span class="font5">font-size: 140%</span>等于<span class="font6">font-size:
1.4em</span></p>
```

6.2.5 尺寸的继承与浏览器的显示

字体尺寸是可继承的，但是继承的是计算值，而不是百分比，例如有代码如下，其显示如图6-12所示。

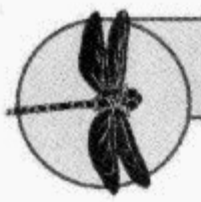
图6-12 字体尺寸的继承

```
#fontSize5 { font-size: 12px; }
#fontSize5 p { font-size: 140%; }
#fontSize5 span { color: #f00; }
<div id="fontSize5">
 <p>p 12px*140% = 16.8px，<span>span 继承16.8px，</span>而不是140%</p>
</div>
```

由于计算中会出现小数，而像素值（px）不可能是小数，因此在实际显示的时候，有的浏览其会四舍五入，有的浏览器会舍掉小数部分。例如有代码如下，其在不同的浏览其内显示如图6-13所示。

```
.size1 {
color:#f00;
font-size: 24px;
}
.size2 {
color:#F90;
font-size: 24.4px;
}
.size3 {
color:#33C;
font-size: 24.5px;
}
.size4 {
color:#909;
font-size: 25px;
}
<p><span class="size1">字24px</span>，<span class="size2">字24.4px</span>，<span class="size3">字
24.5px</span>，<span class="size4">字25px</span></p>
```

图6-13 不同字体尺寸在浏览器内的显示



注意：奇数尺寸的字在横向可能会有变形。

而对于通过继承而得出的尺寸值，浏览器究竟会如何显示呢？例如如下代码，其显示如图6-14所示。由于浏览器对于小数像素值的取舍不同，其实际效果可能有所差别。

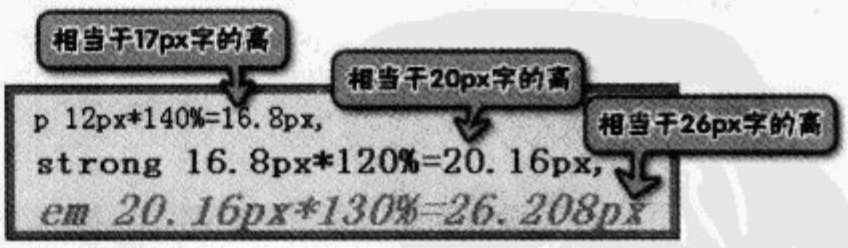


图6-14 字体尺寸继承后在浏览器内的显示

```
#fontSize5 { font-size : 12px; }
#fontSize5 p { font-size : 140%; }
#fontSize5 strong { font-size : 120%; }
#fontSize5 em { font-size : 130%; }
<div id="fontSize5">
 <p>p 12px*140%=16.8px,<strong>strong 16.8px*120%=20.16px,<em>em 20.16px*130%=26.208px</em>
 </strong></p>
</div>
```

6.2.6 分辨率与弹性设计

在［第5章　单位和值］中，曾经介绍过分辨率（dpi）对显示的影响，例如当分辨率为72dpi时，36px和0.5in的文字大小应该是相同的，但是到了96dpi的系统中，0.5in等于48px的文字，因此在一个页面内要避免长度单位的混用。

而为了美观与易于控制，设计者往往习惯使用像素（px）来定义字体大小，例如定义页面整体字体为12px，小标题字为14px等。虽然这样一般情况下看上去很好，但是，对于视力不好的人来说，大片的12px的文字阅读起来是比较困难的。

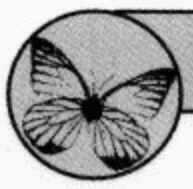
虽然部分浏览器提供了放大缩小文字的功能，但是对于IE 6.0及更早的版本来讲，一旦设定了字体大小为12px，访问者就不能使用菜单中的功能来调整文字大小。

同时，如果分别对文字和标题定义字体尺寸，当需要将文字的尺寸变大的时候，还需要相应修改标题的尺寸，以及其他可能相关的元素的尺寸（比如<h1>、<h2>、等），因此在设计的时候尽量使用尺寸关键字以及百分比等来定义文字的尺寸。

例如，对于左边的XHTML代码，可以如右边所示地定义文字和标题的尺寸：

```
<body>
 <h1>1级标题</h1>
 <p>具体的文字说明。</p>
</body>
```

```
body { font-size: small; }
h1 { font-size: 200%; }
```



提示：读者可以参见下载文件包内［/第2部分/第6章：字体/ font-size2.html］文件。

大部分浏览器默认的文字大小为16px，因此small的大小为13px，而在［6.2.2 绝对尺寸］中曾介绍过，IE 5.5会将small显示为medium大小，因此如果需要兼容IE 5.5，则应针对IE 5.5使用CSS hack技巧，如右所示：

```
body {
font-size: small;
}
* html body {
font-size: x-small;
f\ont-size: small;
}
```

只有IE识别选择器“* html body”，而“f\ont-size”中，反斜杠“\”使IE 5.5忽略掉此条属性，从而接受“x-small”为字体尺寸值，而IE 6.0 和IE 7.0则会识别此条属性，因此以“small”为最终值。在本书［第16章：浏览器与Hack］中将更加系统地介绍针对不同的浏览器设定CSS规则的方法。

而其他元素的文字尺寸，则可以以<body>的字体尺寸为基数，设定<h1>的尺寸为“200%”、<h2>的尺寸为“180%”等，这样设定，当用户改变浏览器的字体尺寸的时候，其他元素的尺寸都将成比例地改变，同时，当设计者要改变字体大小的设定时，只要修改<body>的值，其他元素也会相应改变。

这样，访问者可以根据自己的情况来调整文字的大小，使阅读更加舒适。

不过在使用百分比的时候，要注意可能因为元素嵌套使得计算后的文字尺寸看起来过于小或者大。而且，百分比计算后产生的小数部分容易产生误差，因此在不同的浏览器内，字的大小可能会有差别。因此可能需要通过测试来选择合适的百分比或者关键字的组合。

6.3 字体磅值：font-weight属性

加粗的字体看上去比一般的字体醒目，而细小的字体则显得不是很重要，因此可以用来强调重要的内容，例如（X）HTML中的默认样式，就是加粗的文字，字体的粗细可以通过font-weight属性来定义。本小节的示范页面请参见下载文件包内［/第2部分/第6章：字体/

font-weight.html] 文件。

6.3.1 语法

font-weight属性定义元素内文字的粗细，具体定义列表如下：

语法	font-weight : normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit
说明	设定元素内文字的粗细
值	normal：正常的字体，相当于400。 bold：粗体，相当于700。 bolder：比normal粗。 lighter：比normal细。 100：字体至少像200那样细。 200：字体至少像100那样粗，像300那样细。 300：字体至少像200那样粗，像400那样细。 400：相当于normal。 500：字体至少像400那样粗，像600那样细。 600：字体至少像500那样粗，像700那样细。 700：相当于bold。 800：字体至少像700那样粗，像900那样细。 900：字体至少像800那样粗
初始值	normal
继承性	继承
适用于	所有元素
媒体	视觉
计算值	见下文

例如，可以如下定义字体粗细：

```
p { font-weight: normal; }  /* 400 */
h1 { font-weight: 700; }    /* bold */
body { font-weight: 400; }
strong { font-weight: bolder; } /* 500（如果是可用的） */
```

有些字体本身就比一般的字体粗，比如中文字的“黑体”，但是，由于访问者的用户端可能没有这些字体，因此一般情况下，还是推荐使用font-weigth属性来实现简单的粗体显示效果。

6.3.2 继承

font-weight属性是可继承的，不过继承的是计算值，例如如下代码，虽然对em定义了“font-weight: bolder”，而其子元素<a>只是继承了计算出的粗细值，因此其在浏览器内显示和em中的其他文字一样粗细，而不是“更粗（bolder）”如图6-15所示。

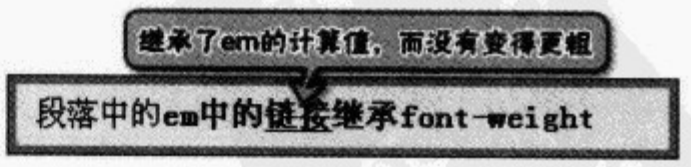


图6-15 font-weight的继承

```
#fontWeight em {
font-style : normal;
font-weight : bolder;
}
<div id="fontWeight">
  <p>段落中的<em>em中的<a href="#">链接</a>继承font-weight</em></p>
</div>
```

6.3.3 浏览器显示原理

虽然看上去font-weight有很多值可以选择，但实际上，受使用的字体的限制，往往很多情况下，不同的值显示出来的效果却是相同的，如图6-16所示。

font-weight，Arial	
font-weight : bold	font-weight : bolder
font-weight : lighter	font-weight : 100
font-weight : 200	font-weight : 300
font-weight : 400	font-weight : 500
font-weight : 600	font-weight : 700
font-weight : 800	font-weight : 900

字体是微，宋体	
font-weight : bold	font-weight : bolder
font-weight : lighter	font-weight : 100
font-weight : 200	font-weight : 300
font-weight : 400	font-weight : 500
font-weight : 600	font-weight : 700
font-weight : 800	font-weight : 900

图6-16　font-weight不同值的现实效果

1. 9个级别的关键字

100到900这9个关键字表示字体粗细的9个级别来同字体相关联（例如，OpenType字体就使用了9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。

实际上，在这些数字中并不存在具体的字体粗细的约定。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可能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。

一般情况下，400等价于“normal”，700等于“bold”。其他数字不对应任何font-weight的关键字，但是它们可以对应普通的字体变形名。

- 如果某种字体的变形标记为“Normal”、“Regular”、“Roman”或“Book”，那么它便被分配给400。
- 如果同时存在标记为“Medium”以及标记为“Book”、“Regular”、“Roman”或“Normal的字体”，那么通常分配为500。
- 如果标记为“Medium”的字体变形是唯一可用的字体，就不能分配为500。

浏览器会根据设定的级别对应字体，如果设定的字体级别不够9个，那么默认的填充“空白”的算法是这样的：

- 如果500没有分配，那么它和400分配的字体一样；
- 如果600、700、800或900中任何一个数值没有分配，那么首先是向更粗的已分配关键字靠拢分配，然后是向更细的方向靠拢分配；
- 如果300、200或100中任何一个数值没有分配，那么首先是向更细的已分配关键字靠拢分配，然后是向更粗的方向靠拢分配。

假定“Rattlesnake”家族中有4个磅值，从细到粗依次是“Regular/Medium/Bold/Heavy”，则其9级font-weight映射关系如下表所示。

可用字体	分配值	填空
"Rattlesnake Regular"	400	100、200、300
"Rattlesnake Medium"	500	
"Rattlesnake Bold"	700	600
"Rattlesnake Heavy"	800	900

2. bolder与lighter

当设定元素的font-weigth属性为“bolder”（或“lighter”）时，浏览器需要先确定其从父元素继承的font-weight值，然后再选择与其最接近的，更粗（或更细）的字体。如果没有可用

的字体，浏览器会将元素的字体粗细设定为下一个（或上一个）数字值。如果值已到900（或100），则其值不变。

6.4 字体样式：font-style属性

字体样式包含“normal（普通）”、“italic（斜体）”和“oblique（倾斜）”3种样式。font-style属性定义元素内文字的样式，具体定义列表如下：

语法	font-style : normal | italic | oblique | inherit
说明	设定元素内文字的样式
值	normal：正常的字体。 italic：斜体，对于没有斜体变量的特殊字体，将应用oblique。 oblique：倾斜的字体
初始值	normal
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

提示：本小节的示例页面请参见下载文件包内［/第2部分/第6章：字体/ font-style.html］文件。

初始值“normal”表示正常的字体文本，但是如果字体本身就是斜体字，例如“Van Dijk”，则“normal”也显示斜体，如图6-17所示。

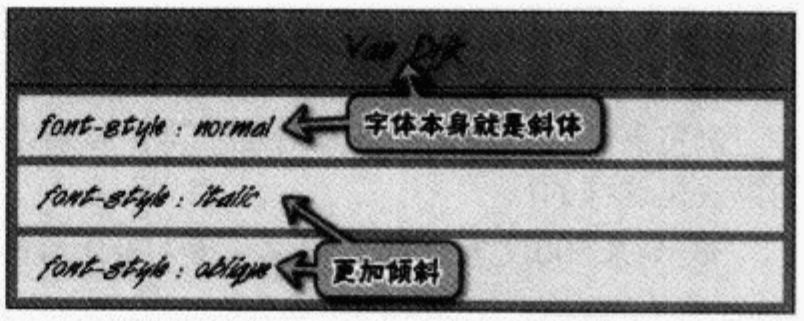


图6-17 斜体字体的显示

斜体（italic）文本基本上保持了原有的字体，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，它的衬线也可以变为倾斜的。字体名中带有“Italic”、“Cursive”或“Kursiv”的字体通常标记为“italic”。

字体名中带有“Oblique”、“Slanted”或“Incline”的字体通常标记为“oblique”。

如果“italic”字体不存在，则改用“oblique”，但是如果“oblique”不存在，则不能用“italic”来替代，因此浏览器会简单地将正常字体倾斜一个角度来显示。

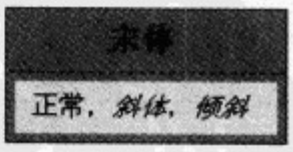


图6-18 宋体字斜体与倾斜的实际显示效果

在很多情况下，斜体和倾斜字体几乎看不出区别，特别是对那些没有特别提供这两种样式的字体来说，例如下列代码显示如图6-18所示。

```
#fontStyle3 { font-family: "宋体"; }
.style1 { font-style: normal; }
.style2 { font-style: italic; }
.style3 { font-style: oblique; }
<div id="fontStyle3">
  <h1>宋体</h1>
 <p><span class="style1">正常</span>，<span class="style2">斜体</span>，<span class="style3">倾斜</span></p>
</div>
```

斜体一般可以用来表示“引用”的内容。

6.5 字体变形：font-variant属性

字体变形主要应用于西文文字。本小节的示范页面请参见下载文件包内［/第2部分/第6章：字体/ font-variant.html］文件。

font-variant属性定义元素内文字是否为小型的大写字母，具体定义列表如下：

语法	font-variant : normal | small-caps | inherit
说明	设定元素中的文本是否为小型的大写字母
值	normal：正常的字体。 small-caps：小型的大写字母字体
初始值	normal
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

小型大写字体的小写字母的字型和大写字母的字型相似，但是尺寸较小且比例略有不同，如图6-19所示。

图6-19 小型大写字体

有些字体提供特别的小型大写字母的字体变形，但是大部分情况下没有这种原始的小型大写字体，浏览器会模拟一个，一种方法是通过使用一个常规字体，并将其小写字母替换为缩小过的大写字母；或者使用常规字体中未缩小的大写字母替换小型大写字体中的字型，因此文本全部以大写字母出现。

6.6 缩写的字体属性：font属性.

使用font属性，可以将font-style、font-variant、font-weight、font-size、line-height以及font-family缩写在一行内。本小节的示范页面请参见下载文件包内［/第2部分/第6章：字体/ font.html］文件。

6.6.1 语法

font属性具体定义列表如下：

语法	font : [[<'font-style'> || <'font-variant'> || <'font-weight'>] ? <'font-size'> [/ <'line-height'>] ? <'font-family'>] | caption | icon | menu | message-box | small-caption | status-bar | inherit
说明	设定元素中文本的字体属性
值	font-style：请参阅 font-style 属性。 font-variant：请参阅 font-variant 属性。

续表

值	font-weight：请参阅 font-weight 属性。 font-size：请参阅 font-size 属性。 line-height：请参阅 line-height 属性。 font-family：请参阅 font-family 属性。 caption：使用有标题的系统控件的文本字体（如按钮、菜单等）。 icon：使用图标标签的字体。 menu：使用菜单的字体。 message-box：使用信息对话框的文本字体。 small-caption：使用小控件的字体。 status-bar：使用窗口状态栏的字体
初始值	取决于用户端
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同各具体属性

font属性缩写有一定的书写顺序，如左边的CSS定义可以缩写为右边的代码（如图6-20所示），各个属性值之间以英文空格分开。

```
em {
font-family: "宋体", serif;
font-size: 14px;
line-height: 1.5em;
font-style: normal;
font-weight: bold;
font-variant: small-caps;
}
```

```
em { font: normal bold small-caps 14px/1.5em "宋体", serif; }
```

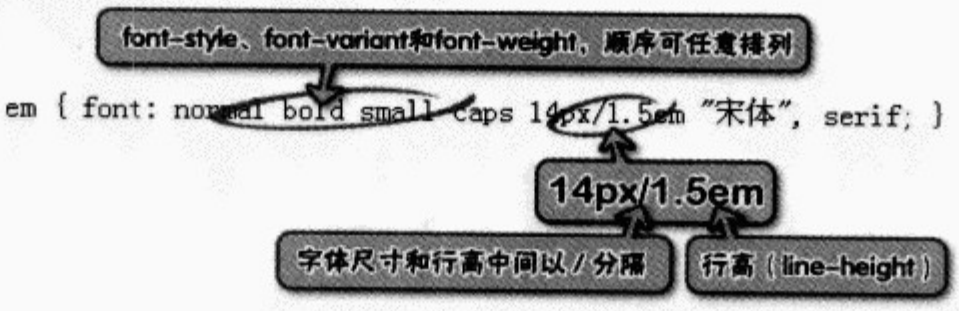


图6-20 font缩写属性的书写顺序

6.6.2 注意

使用font属性还需要注意以下几点。

（1）font内必须同时包含字体尺寸和字体集的值。前3项font-style、font-variant和font-weight可以定义也可以不定义，但是，font-size和font-family的值必须同时定义，同时它们的顺序是固定的，即font最简单的定义如右：

```
font : 字体尺寸 字体集;
```

例如以下左边的定义是正确的，而右边的定义都是错误的：

```
body { font : 12px "宋体"; }
```

```
body { font : 12px; }
body { font : "宋体"; }
body { font : "宋体" 12px; }
```

（2）font-family的值是所有缩写值中最后1个。多个字体之间依然用英文逗号“,”分隔，同样需要注意字体名和引号的使用。

（3）行高（line-height）可选。行高为可选项，如果有行高，字体尺寸和行高之间以英文"/"分隔。关于行高（line-height）的详细介绍，请参见本书［7.3 行高（line-height）］一节。

（4）font定义依照层叠的原则，会覆盖前面单独定义的属性值。例如有CSS定义如下：

```
p {
font-size : 14px;
font-weight : bold;
font-style : italic;
font : 12px "宋体", serif;
}
<p>示例文字示例文字示例文字示例文字示例文字示例文字</p>
```

最后定义的font属性会覆盖前面定义的font-style、font-size等属性，并且将任何未显式指定值的属性重置为初始值，因此上面的CSS实际等同于以下左边的定义，但是，如果按以下右边顺序定义，则根据层叠的原则，后定义的属性会覆盖font中的定义。

```
p {
font-style : normal;
font-variant : normal;
font-weight : normal;
font-size : 12px;
font-family : "宋体", serif;
line-height : normal;
}
```

```
p {
font : 12px "宋体", serif;
font-weight : bold;
}
```

6.6.3 系统字体

CSS 2新增加了几个使用系统字体的值，使用这些值，可以让元素字体的尺寸、家族、粗细、样式和变形同操作系统的设定保持一致。其详细值可以有以下几种情况。

- caption：使用有标题的系统控件的文本字体（如按钮、菜单等）。
- icon：使用图标标签的字体。
- menu：使用菜单的字体。
- message-box：使用信息对话框的文本字体。
- small-caption：使用小控件的字体。
- status-bar：使用窗口状态栏的字体。

例如，在某个表单内有若干个表单按钮，代码如下：

```
<form id="test_form" method="post" action="#">
 <fieldset>
  <legend>系统字体演示</legend>
  <input type="button" name="test_button1" value="按钮1" />
  <input type="button" name="test_button2" value="按钮2" />
  <input type="button" name="test_button1" value="按钮1" class="test1" />
 </fieldset>
</form>
```

对这些按钮设定了CSS如下左，但是，出于某些需求，其中"按钮3"需要让其同系统信息对话框的文字保持一致，则可以对设定CSS代码如下右。

```
input { font: bold medium "黑体", sans-serif; }
```

```
.test1 { font: message-box; }
```

在浏览器内显示如图6-21所示。系统字体只能全体设定。也就是说，字体集、尺寸、粗细、样式等都同时设定。如果需要的话，这些值还可以个别地修改。例如右边代码：

```
button {
font: 600 9pt Charcoal;
}
button {
font: caption;
font-size: 1em;
}
```

图6-21 对按钮设定系统字体

6.7 调整与拉伸

字体调整（font-size-adjust）和字体拉伸（font-stretch）这两个属性包含在CSS 2中，但是到了CSS 2.1又被删除了，因为从CSS 2规范发布到CSS 2.1规范发布几乎没有浏览器真正支持它们。因此，在本小节只作简要的介绍。

6.7.1 字体调整：font-size-adjust属性

在［6.2 字体尺寸（font-size）］一节内曾经介绍过，字体显示的实际尺寸和字体本身设计相关，同时，对于西文字体来说，由于文字包含大小写字母，字体主观上的明显尺寸和可读性更依赖于字母x顶部到底部的距离（x-height），即字体的小写字型的高度，如图6-22所示。

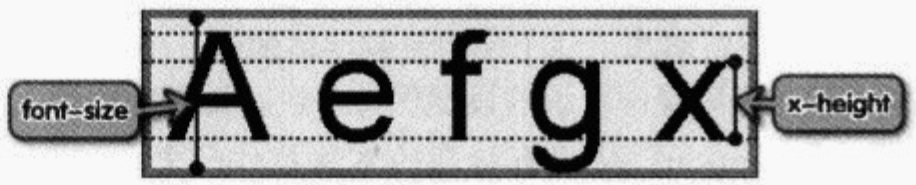


图6-22 字体的x-height

字体尺寸除以x-height，称为视觉值（aspect value），该值越大，说明该字体较小的尺寸也可读。反之，如果视觉值低，在一个阈值之下，字体将迅速地成为不可读，其速度要比视觉值高的字体大的多。

例如，字体Verdana的视觉值为0.58，即当Verdana字体的尺寸为100个单位时，它的x-height为58个单位。而Times New Roman字体的视觉值为0.46。因此，Verdana在较小尺寸时的可见性要比Times New Roman好。图6-23列出了同是12px大小的几种字体的视觉效果。

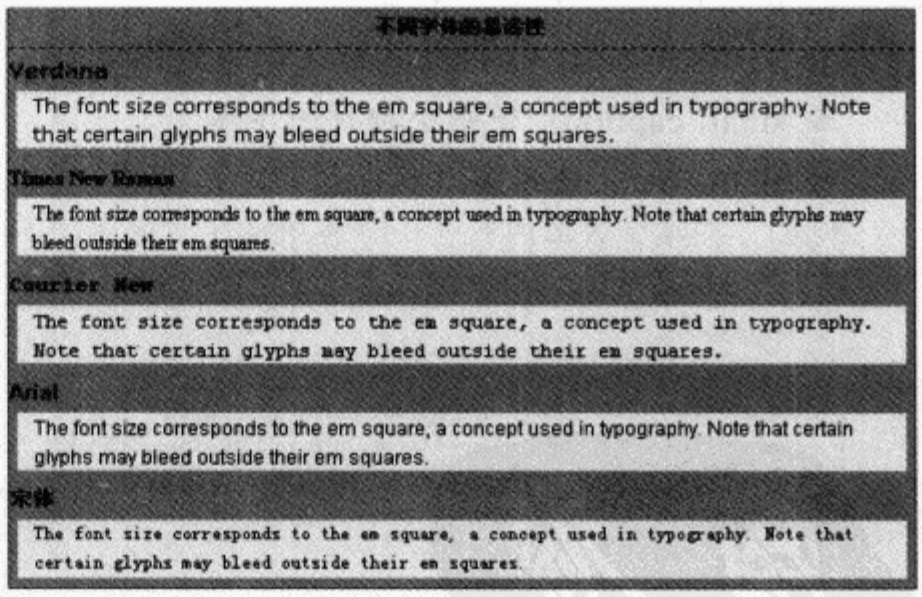


图6-23 相同大小的不同字体的视觉效果

字体调整（font-size-adjust）属性允许制作者对于一个元素指定一个视觉值，具体定义列表如下：

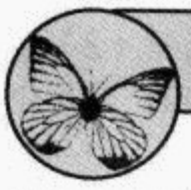
语法	font- size-adjust : <数值> | none | inherit
说明	设定字体名称序列是否强制使用同一尺寸
值	数值：为字体序列中所有字体强迫指定同一尺寸。 none：允许字体序列中每一字体遵守它的自己的尺寸
初始值	none

续表

继承性	继承
适用于	所有元素
媒体	视觉

例如对于某个字体，可以设定其视觉值为：

```
p {
font : 20px "Times New Roman", serif;
font-size-adjust : 0.8;
}
```



提示： Firefox 2.0或许会识别font-size-adjust的值，但是却是将字体尺寸简单地放大。

6.7.2 字体伸展：font-stretch属性

字体样式（font-stretch）定义元素中文本的文字是否横向的拉伸变形，具体定义列表如下：

语法	font-stretch : normal | wider | narrower | <绝对值> | inherit
说明	设定元素内文字是否横向拉伸变形
值	normal：不应用拉伸变形。 narrower：使用比当前设定的值字体宽度更小的值。 wider：使用比当前设定的值字体宽度更大的值。 绝对值：包括从最窄到最宽的8个关键字，其顺序为（从最窄到最宽）ultra-condensed、extra-condensed、condensed、semi-condensed、normal、semi-expanded、expanded、extra-expanded、ultra-expanded
初始值	normal
继承性	继承
适用于	所有元素
媒体	视觉

6.8 字体匹配原理

在本章前几小节内介绍了字体的各个CSS属性，而用户端（例如浏览器）如何来匹配这些属性是一个很复杂的过程，在本节内将介绍其匹配的原理，读者可以只作简单了解即可。

6.8.1 字体的匹配步骤

字体匹配的操作步骤如下。

（1）用户端创建或者获取一个字体属性数据库，这个数据库包含用户端支持的CSS字体属性，例如，本机安装的所有字体列表。如果用户端遇到两个相同的字体，则会简单地忽略掉其中一个。

（2）用户端取出一个元素来并为其创建一个字体属性列表，来决定应用哪些字体属性。基于这个列表，用户端通过font-family属性来选择一个初始的字体集来显示元素，如果匹配完成，则用户端使用此字体，否则进行下一步。

（3）如果上一步匹配不成功，用户端将查找font-family属性中的下一个值，如果有值，并

且重复第（2）步的动作。

（4）如果匹配字体成功，但是如果该字体不包含当前的字符的字形（glyph），此时若font-family属性还有可选择的值，则对下一个值重复第（2）步的动作。

小知识：字符对应的字体数据称为glyph，字体文件中通常带有一个字符映射表（charmap），用来把字符映射到对应glyph的索引值。因为字符集的编码方式有多种，所以可以存在多个子映射表，以支持从不同编码的字符到glyph索引的映射。如果某个字符没有对应的glyph，返回索引0，glyph 0通常显示一个方块或者空格。

（5）如果最终没有字体能匹配，则使用用户端默认的font-family设定，并且进行第（2）步的匹配步骤，以获得最接近的默认字体来显示元素。如果含有不能显示的特殊字符，用户端可能会使用其他适当的字体来显示这些字符。

而对于非常重要的第（2）步，其具体还可以分为以下几步。

❶首先尝试匹配字体样式font-style属性。被标有“italic”或者“oblique”的字体会匹配CSS的“italic”关键字，如果两者都没有，则匹配失败。

❷其次尝试匹配字体变形font-variant属性。用户端字体库内没有“small-caps”标示的字体会被假定为“normal”。而被标示了包含“small-caps”的字体会匹配“small-caps”关键字，或者用大写字母替换小写的字母来合成。

❸接下来尝试匹配字体磅值font-weight属性，这个值总是会匹配，不会失败。

❹最后尝试匹配字体尺寸font-size属性。字体尺寸的显示可能存在着一些误差（例如通过计算而来的尺寸），而实际显示的尺寸和用户端相关，有的用户端可能会允许最多20%的误差，而有些用户端可能只允许10%的误差。

6.8.2 设定字体集的注意事项

浏览器使用的是访问者系统内的字体文件，因此当通过font-family属性设定某个字体的时候，如果访问者的系统内没有这种字体，那么对于字体的匹配将失败，因此建议设定多个相似的字体，来预防没有匹配字体的情况，如右：

```
h3 { font-family: Arial, Helvetica, sans-serif; }
```

浏览器可以顺序查找，如果没有“Arial”字体，则查找是否有“Helvetica”字体，如果有则应用显示，如果没有则使用“sans-serif”系列的字体代替。同时，在定义字体的时候，最好指定一种常用字体系列，以防止所有设定的字体都不存在，例如以下左边代码。可以只定义一个字体系列，而不指定具体的字体名，例如以下右边代码。

```
body { font-family: "宋体", serif; }
h1 { font-family: "黑体", sans-serif; }
```

```
p { font-family : serif; }
```

当只设定某种字体系列时，浏览器内具体显示的字体将由浏览器的默认设定来决定。

6.8.3 字体的选择

在一个文档里指定字体选择行为有4种：名字匹配、智能匹配、综合和下载。

1. 字体名字匹配

在这种情况下，用户端使用一个可获得的与要求的字体具有相同的家族名的字体。匹配信息局限于CSS字体属性，包括家族名。这是CSS 1使用的唯一方法。

2. 智能字体匹配

在这种情况下，用户端使用一个可获得的与要求的字体外观最接近的字体(尺寸不一定匹配)。匹配信息包括字体类型的信息 (文本或符号)、衬线的特性、重量、大写高度、X高度、提升、下沉、倾斜等。

3. 字体综合

在这种情况下，用户端创建一个字体，不仅在呈现上和要求的字体接近，而且匹配它的尺寸。综合信息包括匹配信息，通常比某些匹配情况会要求更精确的值值。特别地，综合要求精确的宽度尺寸、字符到字型的替换以及在需要保留指定字体的布局特征时的定位信息。

4. 字体下载

最后，用户端从网络上获得一个字体。该过程和在网络上获得图形、声音或小应用程序相似。类似地，在页面呈现之前会引起一定的延迟。

注意：渐进渲染是下载和另外一个方法的组合；它先提供一个临时的替换字体（使用名字匹配，智能匹配或综合），因此在要求的字体下载时，可以先阅读内容。一旦真正的字体下载完毕，它替换了临时字体，当然希望不要再重新排列内容。

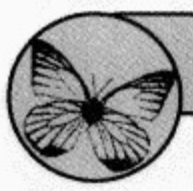
6.8.4 @font-face规则

在CSS 2中，制作者可以使用字体描述“@font-face”规则，指定用户端从网络上获得一个字体。不过由于浏览器对其支持的不完整，因此在CSS 2.1中又删除了这个属性。

语法	@font-face { font-family : name ; src : url(url) ; sRules }
说明	设定文档内嵌入的字体
值	name：字体名称。 url：使用绝对或相对地址指定字体。 sRules：样式表定义

例如：

```
@font-face {
font-family: "Robson Celtic";
src: url("http://site/fonts/rob-celt")
}
h1 { font-family: "Robson Celtic", serif }
```



提示：由于一般的中文字体文件的字节数都在2M以上，因此不推荐使用此方法。

第7章 文本

绝大部分网页的内容，包括导航、版权信息等，往往都是由大段的文字和图片组成，因此使用CSS来设定段落的样式是非常重要的。大段密集的文字往往会显得很枯燥乏味，通过设置文本和文字的属性，不仅可以改变文字之间的行距、对齐方式，还可以实现上标或者下标、悬挂缩进等效果。大部分的文本属性都是作用在块级元素上的，也有少数对行内元素起作用。

7.1 文本水平对齐：text-align属性

text-align属性用以设定元素内文本的水平方向的对齐方式。本小节的示范页面请参见下载文件包内［/第2部分/第7章：文本/text-align.html］文件。

7.1.1 语法

text-align属性具体定义列表如下：

语法	f text-align : left | right | center | justify | inherit
说明	设置元素内文本的水平对齐方式
值	left（左对齐）、right（右对齐）、center（居中）、justify（两端对齐）
初始值	跟浏览器的设置及书写方向有关
继承性	继承
适用于	块级元素
媒体	视觉
计算值	初始值或同指定值

例如，有以下代码，其在浏览器内的显示如图7-1所示。

```
<p style="text-align:left;"><strong><img src="img/ddcat_anim.gif" alt="图片" width="88" height="31" /></strong>文本左对齐：<strong>text-align: left</strong>。</p>
<p style="text-align:right;"><img src="img/ddcat_anim.gif" alt="图片" width="88"height="31" />文本右对齐：<strong>text-align: right</strong>。</p>
<p style="text-align:center;"><strong><img src="img/ddcat_anim.gif" alt="图片" width="88" height="31" /></strong>文本居中对齐：<strong>text-align: center</strong>。</p>
<p style="text-align:justify;">文本两端对齐：<strong>text-align:justify</strong>。文本两端对齐，test text,文本两端对齐。test text,文本两端对齐，test text,文本两端对齐。test text,文本两端对齐。</p>
```

前3种对齐方式都很好理解，而最后一种两端对齐（text-align:justify）可以让大段的文本看起来比较整齐，不过两端对齐的表现可能会因为浏览器的不同而有所不同，如图7-2所示。

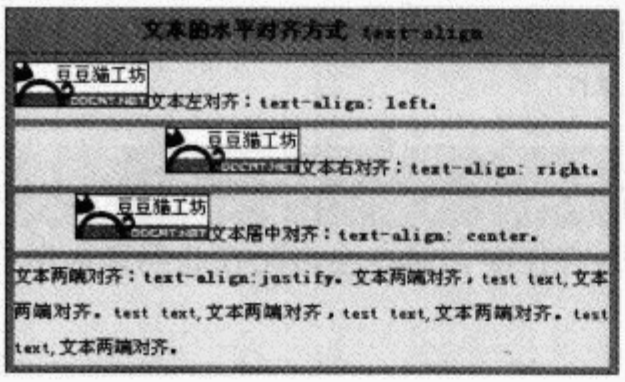


图7-1 水平对齐方式

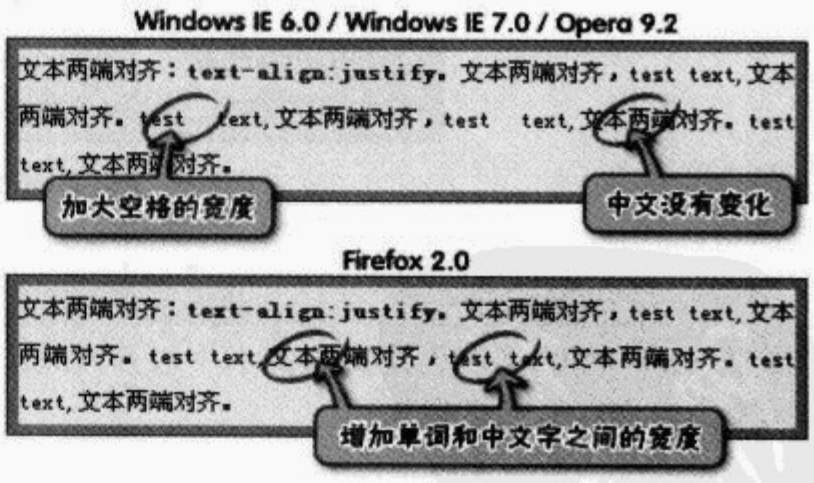


图7-2 不同浏览器对于两端对齐的不同表现

7.1.2 适用于：块级元素

text-align属性只有对块级元素设定才会生效。例如有如下设定：

```
<p style="text-align:right;"><img src="img/ddcat_anim.gif" alt="图片" width="88" height="31" style="text-align:center;" /></p>
```

虽然对图片设定了居中对齐，但是在浏览器内的效果如图7-3所示。

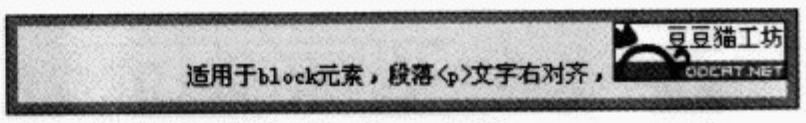


图7-3 图片与对齐方式

由图7-3可以看到，因为图片<img>默认为行内元素，所以即使定义了"text-align:center"，也仍然同文字一起右对齐。因此不能直接通过对图片的设定来达到单独图片的对齐。如果想让单个图片达到居中的效果，应该在其外嵌套一个块级元素，然后设置这个元素的对齐方式为居中对齐，如下所示：

```
p { text-align:center; }
<p><img src="img/ddcat_anim.gif" alt="图片" width="88" height="31" /></p>
```

其效果如图7-4所示。

图7-4 图片居中

7.1.3 继承

text-align属性是可继承的。当设定了某个元素的水平对齐方式以后，其后代元素将继承该设定，除非对后代元素进行单独设定，例如有一个 其设定如下：

```
# textAlign3 { text-align: right; }
```

则这个 内的元素如果不进行设定，则会全部右对齐，如图7-5所示。

注意： 不同浏览器之间的继承略有差别。例如在Opera中，表头<th>将不继承右对齐，而依然居中对齐，除非对其专门定义。

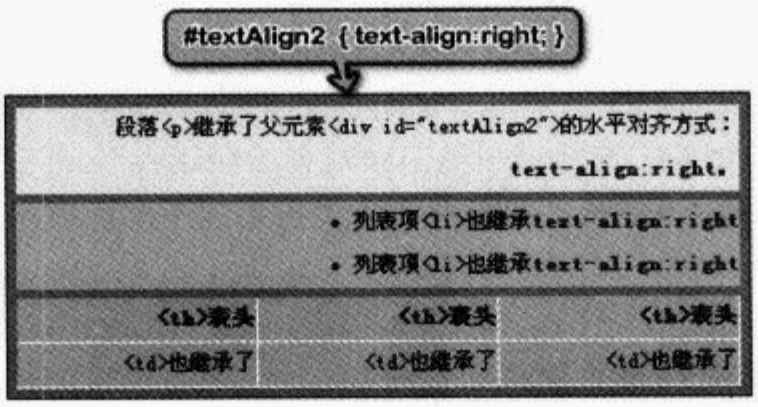


图7-5 text-align的继承性

7.1.4 应用：整体居中

虽然text-align用以设定**文本**的水平对齐方式。但是在IE中，对齐会应用在此元素内**所有**的后代元素上，例如有如下代码：

```
#textAlign3 {
text-align:center;
}
#textAlign3 p {
width:70%;
}
<div id="textAlign3">
  <p>本段落会在IE内居中显示，而在Firefox和Opera内居左显示。</p>
</div>
```

其在IE 5.5及以后的版本、Firefox 2.0和Opera 8.5中显示的效果如图7-6所示。

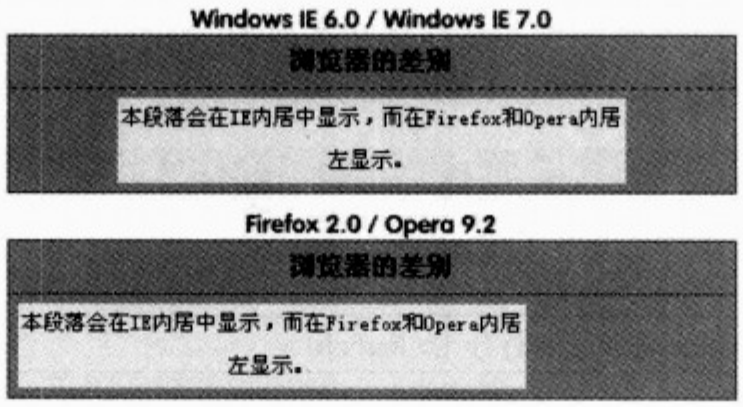
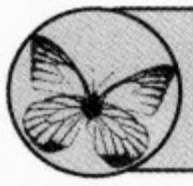


图7-6　不同浏览器的差别



提示：本小节的示范页面请参见下载文件包内［/第2部分/第7章：文本/text-align-center.html］文件。

由图7-6可以发现，<p>元素本身也居中显示了，因此可以利用此特性来设置页面内容在IE内的整体居中效果，例如有XHTML代码如右：

```
<body>
 <div id="wrap">
  <h1>页面居中</h1>
  <p>设置CSS使页面整体居中。</p>
 </div>
</body>
```

要使 在浏览器内居中，则需要设置CSS如下：

```
body {
text-align: center; /* 在IE内居中 */
}
#wrap {
width: 90%; /* 设定宽度才能显示出居中的效果。*/
margin: 0 auto; /* 在Firefox及Opera等浏览器内居中。*/
}
```

此时在浏览器内浏览，页面内的元素都将居中显示，如图7-7所示。

图7-7　浏览器内页面整体居中

提示：此效果主要是为了使页面在IE 5.5内居中，而IE 6.0及7.0 支持“margin: 0 auto;”的设置，如果不考虑IE 5.5，则无需设定<body>的text-align属性。此时页面内所有的文本都继承<body>的设定而居中显示，因此实际应用中，可以再设定wrap层的对齐方式为左对齐。

7.2 文本缩进：text-indent属性

写文章的时候每个段落的开始要空两个空格，这样可以容易分辨文字段落。在网页内的大段

文字内容，也适用这个规则，很多制作者喜欢在文字前加入两个全角空格来达成这种效果，其实，可以通过设定CSS的文本缩进text-indent属性来完成。

7.2.1 语法

text-indent属性具体定义列表如下：

语法	text-indent : <长度> | <百分比> | inherit
说明	设置元素中第一行文本的缩进
值	长度值：带有单位的数值，允许为负值。 百分比值：百分比是指相对于父元素的宽度，允许为负值
初始值	0
继承性	继承
适用于	块级元素
媒体	视觉
计算值	依照指定的百分比或绝对长度

缩进只对应元素内第一行文本起作用，而浏览器内元素的默认状态是无缩进的，如图7-8所示。

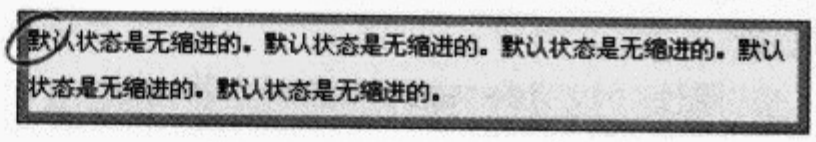


图7-8 元素默认无缩进状态

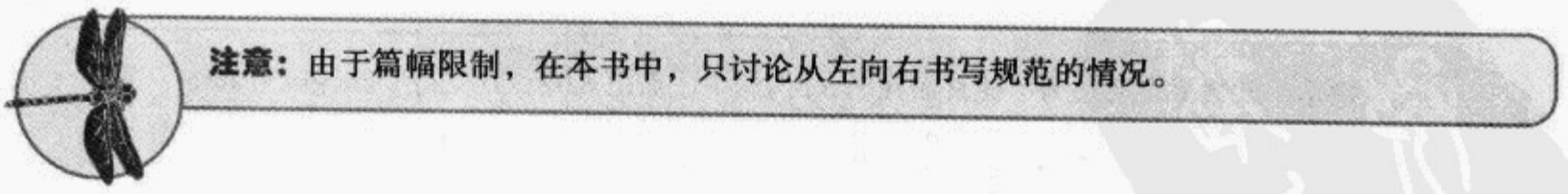
缩进与水平对齐一样，适用于块级元素，而且可被继承。本小节的示例页面请参见下载文件包内［/第2部分/第7章：文本/text-indent.html］文件。

7.2.2 正值缩进

当缩进为正值时，缩进方向和文字的书写方向一致。例如有如下XHTML代码：

```
#textIndent1 p { text-indent:2em; }
<div id="textIndent1">
  <p>文字缩进2个字符宽：<strong>text-indent:2em;</strong>，可以看到只有第一行文字缩进。填充文字填充文字填充文字填充文字填充文字填充文字。</p>
  <p><img src="img/ddcat_anim.gif" alt="图片" width="88" height="31" />带图片文字缩进2个字符宽：<strong>text-indent:2em;</strong></p>
</div>
```

此代码显示如图7-9所示。从图7-9可以看到，段落的首行向右缩进，这是因为按照简体中文的书写习惯，是从左到右的顺序，因此左边是开始点，正值向右缩进。而如果将页面按照阿拉伯等地区的从右向左的书写习惯，则正值缩进为从首行的右端开始，如图7-10所示。



注意：由于篇幅限制，在本书中，只讨论从左向右书写规范的情况。

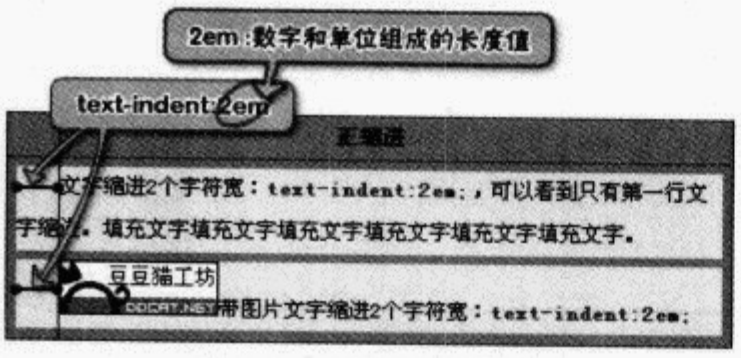


图7-9 文字的正值缩进

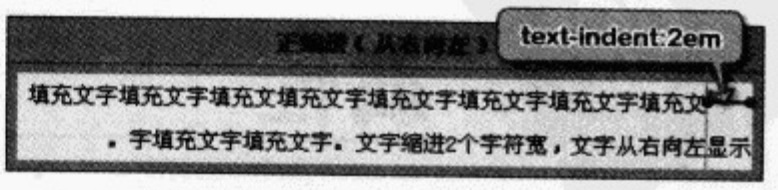


图7-10 从右往左显示文字的缩进

其在IE 5.5及以后的版本、Firefox 2.0和Opera 8.5中显示的效果如图7-6所示。

应用百分比缩进的时候要注意，百分比的基数是由**父元素的宽度**决定，而不是元素本身的宽度。例如有如下代码，则其实际效果如图7-11所示。

```
# textIndent1 {
width : 400px;
}
#textIndent1 p {
width : 300px;
text-indent : 50%;
margin : 5px;
}
<div id="textIndent1">
<p>段落宽度300px。</p>
</div>
```

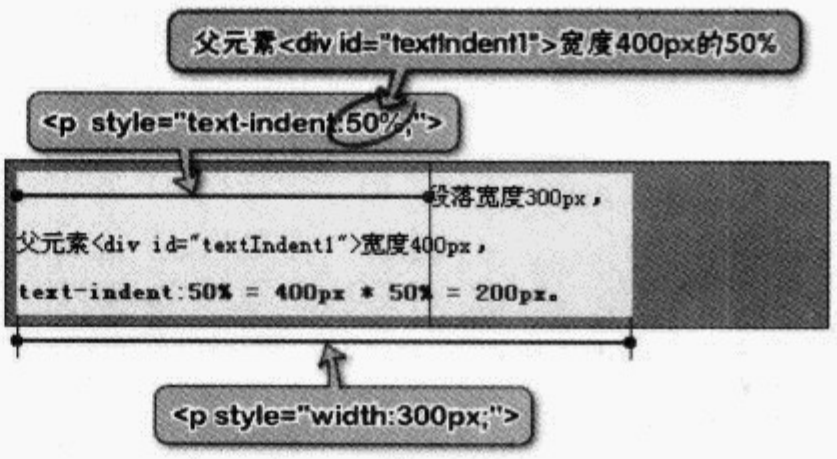


图7-11 百分比缩进

7.2.3 负值缩进

负值的缩进方向与正值缩进方向相反，例如有代码如下，其显示如图7-12所示。

```
p { text-indent : -2em; }
<p>情况1：使首行文字向左突出，如果未设定元素的padding-left，则文字会突出元素的左边框。<strong>text-indent:-2em;</strong></p>
```

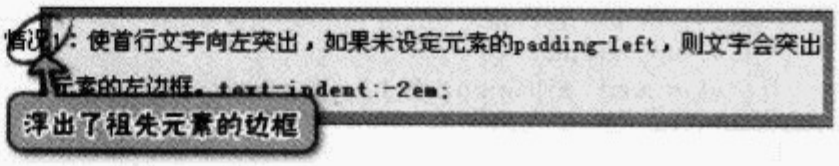


图7-12 负值缩进在不同浏览器内的表现

提示：溢出的部分的显示情况根据元素及其祖先元素的overflow属性来定。关于overflow属性，请参见本书［9.6.1 溢出：overflow属性］一节。

一般有两种情况会用到负值缩进：悬挂缩进和隐藏文字。例如有如下代码：

```
p {
padding-left : 3.5em;
text-indent : -3.5em;
}
<p>情况2：设定元素的padding-left，实现"悬挂缩进效果"<strong>padding-left:3.5em;text-indent:-3.5em;</strong>。填充文字，填充文字，填充文字，填充文字，填充文字，填充文字，填充文字。</p>
```

此时向右突出的首行文字会完整显示在元素内容区域内，如图7-13所示。

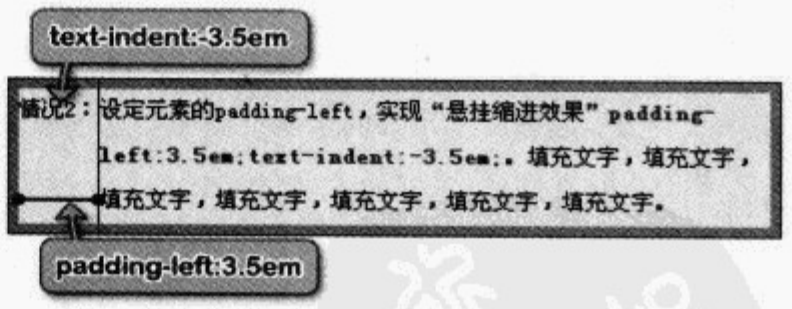


图7-13 悬挂缩进

7.2.4 应用：隐藏单行文字

在本书［6.1 字体集（font-family）］一节内介绍过，在网页内应用字体受到很大的限制，因此对于设计图中一些设定了特殊字体的文字一般都使用图片来达到显示效果。

例如，有设计图及其对应的XHTML代码如图7-14所示。此处<h3>使用了特殊字体，因此需要使用图片来显示，但是，如果直接在<h3>标签内插入图片，

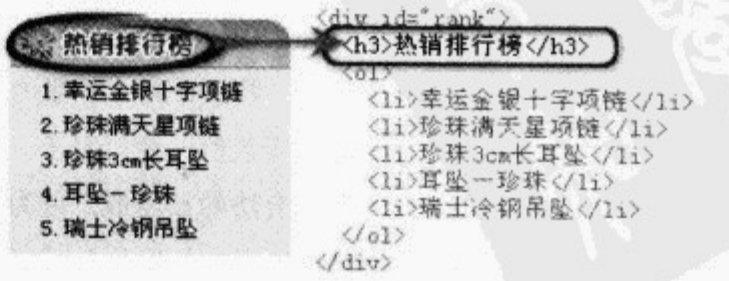


图7-14 设计图及其对应的XHTML代码

如下所示：

```
<h3><img src="img/rank_h3.gif" alt="热销排行榜" /></h3>
```

虽然这样也可以达到设计要求，但是违背了“表现与结构分离”的原则，因为这个图片不是“内容”而是“表现”，<h3>的内容应该是文字“热销排行榜”，因此这个图片应该通过设定<h3>的背景图片来实现，其css代码如下，此时浏览器中显示如图7-15所示。

```
#rank h3 {
font-size: 1em;
background : url(img/rank_h3.gif) no-repeat;
height : 27px;
}
```

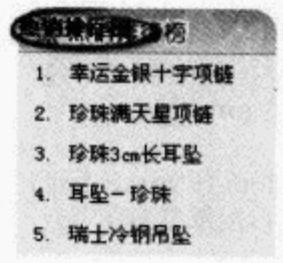


图7-15 rank层标题效果

此时<h3>内的文字是不需要显示的，因此对<h3>增加缩进定义如下，将文字向左缩进-9999em处，完全超出了浏览器可视范围之外，从而达到了隐藏文字的目的，此时浏览效果如图7-16所示。

```
#rank h3 {
font-size : 1em;
background : url(img/rank_h3.gif) no-repeat;
height : 27px;
text-indent : -9999em;
}
```

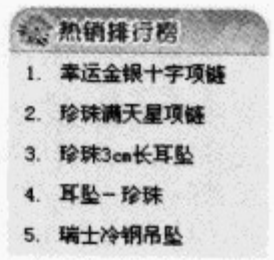
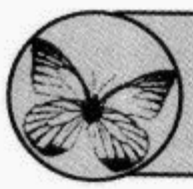


图7-16 rank层标题最终效果

这种隐藏文字的方法不仅局限于应用在标题上，包括导航菜单、按钮等使用特殊字体的设计，都可以应用此方法来实现。但此方法只能应用在单行文本上，因为缩进只能对第一行文字起作用。

7.3 行高：line-height属性

大片密密麻麻的文字往往会让人觉得乏味，因此适当地调整行高（line-height）可以减低阅读的困难与枯燥，并且使页面显得美观。行高指的是文本行的基线间的距离，但是文本之间的空白距离不仅仅是行高决定的，同时也受到字号的影响。



提示： 本小节的示例页面请参见下载文件包内［/第2部分/第7章：文本/line-height.html］文件。

7.3.1 语法

line-height属性的具体定义列表如下：

语法	line-height : normal | <实数> | <长度> | <百分比> | inherit
说明	设置元素中行的高度
值	normal：默认行高，一般为1到1.2。 实数：实数值，缩放因子。 长度：合法的长度值，可为负数。 百分比：百分比取值基于元素的字体尺寸
初始值	normal

续表

继承性	继承
适用于	所有元素
媒体	视觉
计算值	长度和百分比值为绝对值；其他同指定值

行高指的是文本行的基线间的距离。而基线（Base line），指的是一行字横排时下沿的基础线，基线并不是汉字的下端沿，而是英文字母x的下端沿，同时还有文字的顶线（Top line）、中线（Middle line）和底线（Bottom line），用以确定文字行的位置，如图7-17所示。行高与字体尺寸的差称为行距（leading），如图7-18所示。

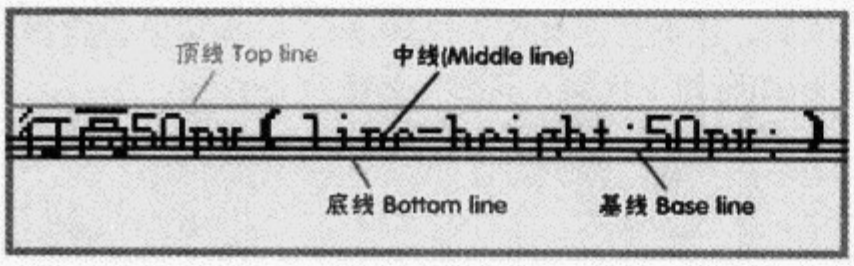


图7-17　文字的基线

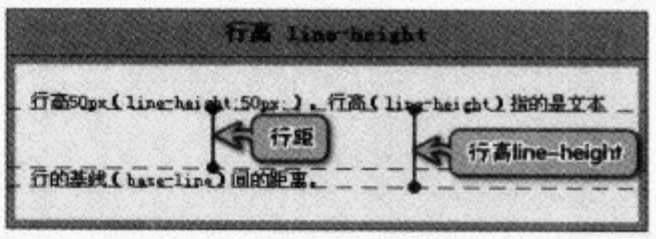


图7-18　行高

7.3.2 内容区域、行内框和行框

理论上讲，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，如图7-19所示。

行内元素会生成一个行内框（inline box），行内框只是一个概念，它无法显示出来，但是它又确实存在。在没有其他因素影响的时候，行内框等于内容区域，而设定行高则可以增加或者减少行内框的高度，即：将行距的值（行高-字体尺寸）除以2，分别增加到内容区域的上下两边，如图7-20所示。

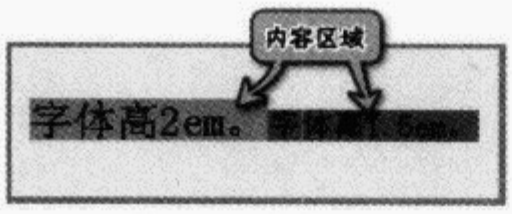


图7-19　内容区域

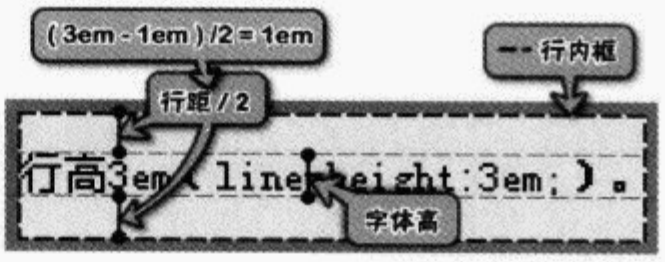


图7-20　行内框与行高

由于行高可以应用在任何元素上，因此同一行内的若干元素可能有不同的行高和行内框高，例如有如下代码，其显示如图7-21所示。

```
<p style="line-height:20px;">&lt;p&gt;行高20px。<strong style="line-height:50px;">&lt;strong&gt;行高50px。
</strong><span style="line-height:30px;">&lt;span&gt;行高30px。</span></p>
```

这里又有一个新的概念—行框（line box）。同行内框类似，行框是指本行的一个虚拟的矩形框，其高度等于本行内所有元素中行高最大的值。因此，当有多行内容时，每行都会有自己的行框，如图7-22所示。

图7-21　行内框与行框

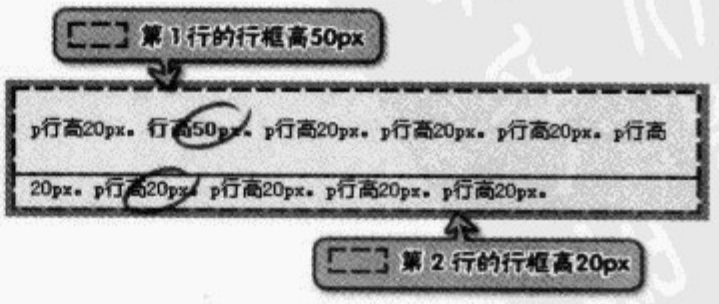


图7-22　多行内容的行框

注意：行框的高度只同本行内元素的行高有关，而和父元素的高度（height）无关。理解行框和行内框的概念对于学习本章［7.4垂直对齐：vertical-align属性］一节的内容非常重要。

7.3.3 行高的计算与继承

以em、ex和百分比为单位的行高，其基数是元素本身的字体尺寸。例如有代码如下：

```
<p style="font-size:20px;line-height:2em;">字高20px，行高2em。</p>
<p style="font-size:30px;line-height:2em;">字高30px，行高2em。</p>
```

2个段落的行高都为2em，但是字体大小不同，因此显示如图7-23所示。行高可以设定得比字体高度小，此时多行的文字将叠加到一起，例如有如下代码，其显示如图7-24所示。

```
p {
font-size : 20px;
line-height :10px;
}
<p>字高20px，行高10px。此时多行的文字将叠加到一起。</p>
```

图7-23 行高的计算

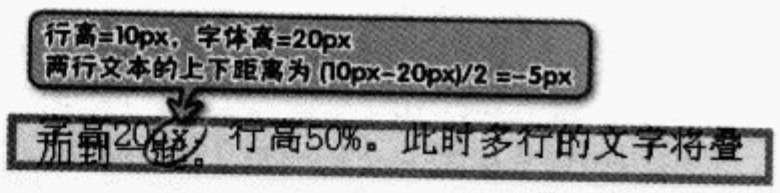


图7-24 比字体高度小的行高

行高是可继承的，但是继承的是计算值，例如有如下代码：

```
p {
font-size :20px;
line-height : 2em;
}
p span {
font-size : 30px;
}
<p>字高20px。<span>字高30px。</span></p>
```

<p>元素的行高2em，字体尺寸为20px，因此计算值为40px，虽然元素本身的字体尺寸为30px，不过其继承的行高仍为40px。但是在不同的浏览器内显示的效果却不尽相同，如图7-25所示。

由于继承的是计算值，因此当元素内的文字字体尺寸不一样的时候，如果设定固定的行高很可能造成字体的重叠，例如有如下代码，其显示如图7-26所示。

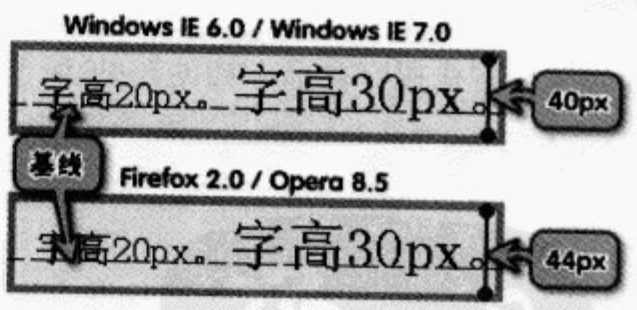


图7-25 行高的不同表现

```
p {
font-size : 20px;
line-height : 1em;
}
p span {
font-size : 30px;
}
<p>字高20px，行高1em，当文本为多行时可能会发生文字重叠的想象。<span>字高30px。</span></p>
```

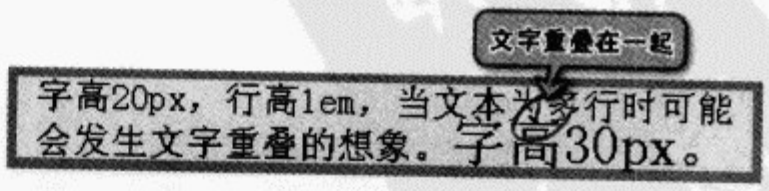


图7-26 行高继承造成文字叠加

为了避免这种情况，可以为每个元素单独定义行高，但是这样很繁琐，因此可以定义一个没有单位的实数值作为缩放因子来统一控制行高，缩放因子是直接继承的，而不是继承计算值。例如修改上例中的行高为以下左边代码，则上例中的XHTML代码显示如图7-27所示。

```
p { line-height : 1; }
```

图7-27 缩放因子对行高的影响

当内容中含有图片的时候，如果图片的高度大于行高，则含有图片行的行框将被撑开到图片的高度，如图7-28所示。

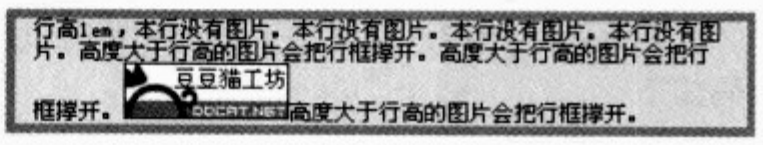


图7-28 含有图片的行

注意： 图片虽然撑开了行框，但是不会影响行高，因此也不会影响到基于行高来计算的其他属性。当行内含有图片的时候，图片和文字的垂直对齐方式默认是基线对齐，关于垂直对齐将在本章［7.4 垂直对齐：vertical-align属性］一节中讨论。

7.3.4 浏览器的差别与错误

浏览器在显示的时候往往会有自己的表现形式，例如在Opera内，行高将按照CSS定义的将行距除以2增加到内容区域的上下两边，而IE和Firefox则不是完全平分，如图7-29所示。

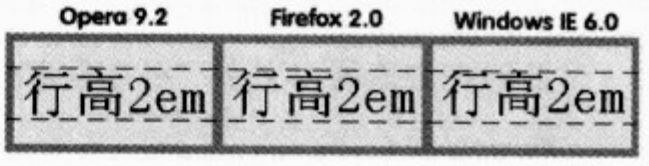


图7-29 不同浏览器对行高的显示

不过，相差的1或2个像素在实际显示中一般不会有太大的影响，因此可以忽略不计。

比较严重的错误是IE 6.0对于含有图片或者表单元等可替换行内元素的行高失效的问题，不过，在IE 7.0中已经修正了这个错误，但是其表现同其他浏览器也不相同。例如有如下代码，其显示如图7-30所示。

```
#lineHeight4 p { line-height : 60px; }
#lineHeight4 fieldset{ border : 0; }
<div id="lineHeight4">
  <p>内容含有图片在［IE 6］内浏览line-height将失效。<img src="../../img/ddcat_anim.gif"alt="图片" width="88"
height="31" /></p>
  <form id="testForm" action="#">
   <fieldset>
     <p><label for="test1">表单元素</label><input type="text" maxlength="16" value="IE6内行高失效" /></p>
    </fieldset>
  </form>
</div>
```

由图7-30可以发现，IE 7.0中将半行距分别加在了图片的上下，而由于图片默认是基线对齐，因此文字的基线下移了，这显然不符合CSS中的规定。对于IE 6.0中行高失效的问题，需要使用CSS Hack手段来针对IE 6.0设定替换元素的上下补白来修正。

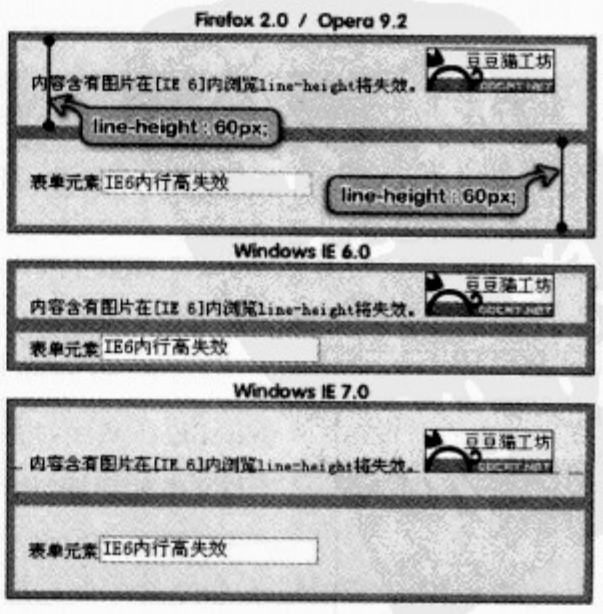
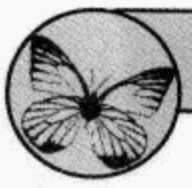


图7-30 包含替换元素的行高在IE内失效



提示： 关于针对IE 6.0的CSS Hack，请参见本书［第16章：浏览器与Hack］。

7.3.5 应用：单行文字在垂直方向居中

在网页设计中，往往为了突出标题而添加背景图案，如图7-31所示。假设此标题的XHTML代码如下：

```
<div id=" #sample" >
 <h2>热门帖大盘点</h2>
 ……
</div>
```

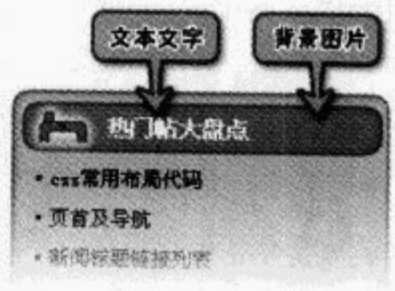


图7-31 包含背景图片的标题

此时如果只设定<h2>的背景图片和高，则文字会偏上，如图7-32所示。针对这个现象，一般只需要设定与高度相等的行高即可，相关代码如下：

此时在浏览器内文字已经在垂直位置上居中显示，如图7-33所示。

```
#sample h2 {
height : 31px;
line-height : 31px;
……
}
```


图7-32 未设定行高的标题文字

图7-33 设定行高后的标题文字

此方法同样可以运用在其他需要文字垂直居中显示的地方，如列表项、导航条等。

7.4 垂直对齐：vertical-align属性

上一小节讲解了行高与单行纯文字的垂直居中，而如果行内含有图片和文字，在浏览器内浏览时，读者可以发现文字和图片在垂直方向并不是沿中线居中，而是沿基线对齐，如图7-34所示。这是因为元素默认的垂直对齐方式为基线对齐（vertical-align: baseline）。

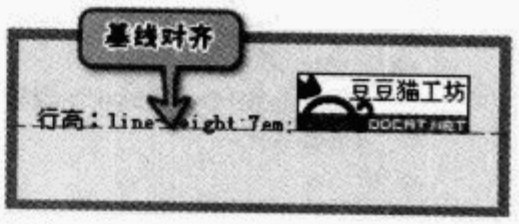


图7-34 文字和图片内容默认垂直对齐方式为基线对齐

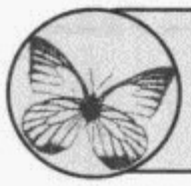
7.4.1 语法

vertical-align属性的具体定义列表如下：

语法	vertical-align : baseline | sub | super | top | text-top | middle | bottom | text-bottom | <百分比> | <长度> | inherit
说明	设置元素内容的垂直对齐方式
值	baseline（基线对齐）、sub（下标）、super（上标）、top（顶端对齐）、text-top（与文本的顶端对齐）、middle（中部对齐）、bottom（底端对齐）、text-bottom（文本的底端对齐）。 百分比和长度：CSS 2，可为负数

续表

初始值	baseline
继承性	不继承
适用于	行内元素和单元格（table-cell）元素
媒体	视觉
计算值	百分比和长度值为绝对长度，其他同指定值



提示：单元格元素的垂直对齐方式比较独特，将在［7.4.3单元格的垂直对齐］中介绍。本小节的示范页面请参见下载文件包内［/第2部分/第7章：文本/vertical-align.html］文件。

此处需要特别注意的是：垂直对齐属性只对**行内元素**有效。例如有如下代码：

```
<p style="vertical-align:super;">垂直对齐<span>上标</span></p>
```

<p>元素默认为块级元素，因此在浏览器内浏览时将不会有任何变化。而如下代码：

```
<p>垂直对齐<span style="vertical-align:super;">上标</span></p>
```

元素默认为行内元素，因此显示如图7-35所示。行内元素还包括图片、表单输入元素等，同时，垂直对齐不能被继承。

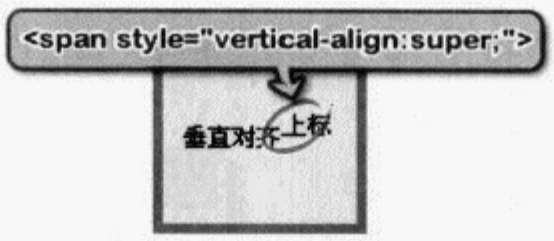


图7-35　垂直对齐属性只对行内元素有效

7.4.2 属性值详解

在［7.3 行高 line-height］一节中，曾经介绍了文本的基线、顶线、中线和底线，还有内容区域、行内框和行框，而本节的垂直对齐和这几个概念密切相关。

垂直对齐主要属性值的表现形式如图7-36所示。

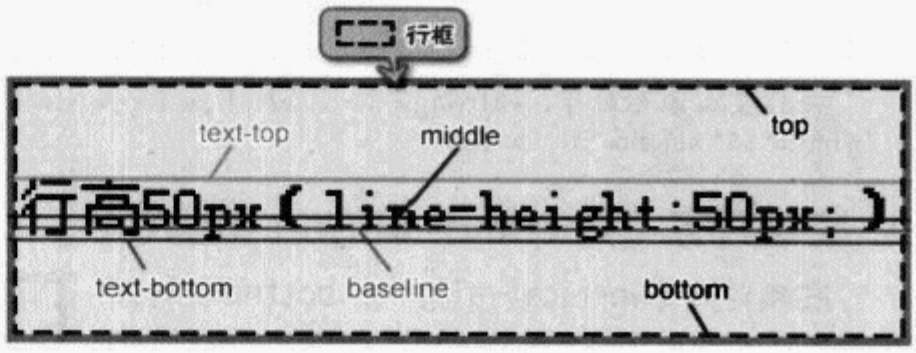


图7-36　垂直对齐的主要属性值示意

1. 基线对齐（vertical-align : baseline）

基线对齐（vertical-align : baseline）使元素的基线同父元素的基线对齐，例如有如下代码，则其显示如图7-37所示。

```
p strong {
line-height : 7em;
font-size : 2em;
vertical-align : baseline;
}
```

```
<p>基线对齐<strong>vertical-align:baseline;</strong></p>
```

图7-37　基线对齐

而像图片或者输入框这样的元素，本身没有基线，则将其底端同父元素的基线对齐，如图7-34所示。

2. 顶端对齐（vertical-align : top）

顶端对齐（vertical-align ： top）是将元素的行内框的顶端与行框的顶端对齐，例如有如下代码，则其显示如图7-38所示。

```
p {
line-height : 7em;
}
p strong {
vertical-align:top;
line-height:2em;
}
p img {
vertical-align : top;
}
 <p>顶端对齐：<strong>vertical-align:top;</strong><img src="img/ddcat_anim.gif" alt="图片" width="88"
height="31" /></p>
```

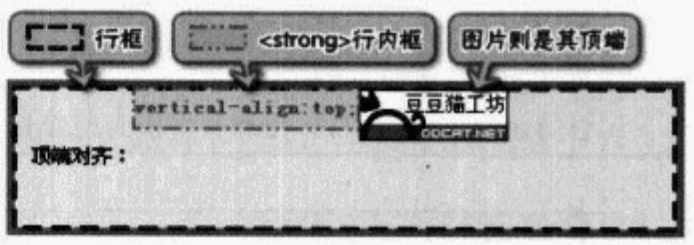


图7-38 顶端对齐

对于元素，不仅设定了vertical-align，还设定了line-height，这是因为在本章［7.3.2 内容区域、行内框和行框］一节中关于行内框的说明中介绍过，行高可以改变行内框的高度，如果不重新设定行高，则元素继承了父元素<p>的行高，因此行内框高和行框的高度是一样的，则顶端对齐将看不出效果。

3. 文本顶端对齐（vertical-align : text-top）

文本顶端对齐（vertical-align ： text-top）是将元素行内框的顶端同文本行的顶线对齐，例如有如下代码，其显示如图7-39所示。

```
p {
line-height : 7em;
}
p strong {
vertical-align : text-top;
line-height : 2em;
}
p img {
vertical-align : text-top;
}
 <p>文本顶端对齐：<strong> vertical-align:top;</strong><img src="img/ddcat_anim.gif" alt="图片
"width="88" height="31" /></p>
```

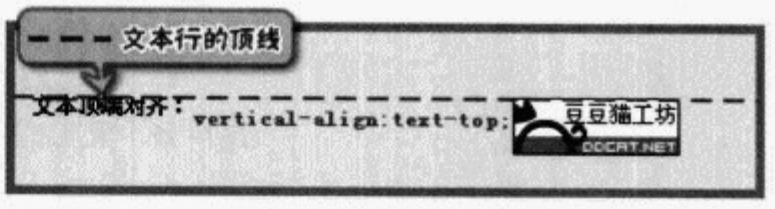


图7-39 文本顶端对齐

4. 底端对齐（vertical-align : bottom）

底端对齐（vertical-align ： bottom）与顶端对齐（vertical-align ： top）相反，如图7-40所示。

图7-40 底端对齐

5. 文本底端对齐（vertical-align : text-bottom）

文本底端对齐（vertical-align ： text-bottom）与文本顶端对齐（vertical-align ： text-top）相反，如图7-41所示。

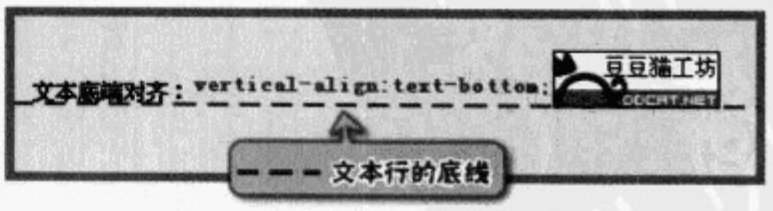


图7-41 文本底端对齐

6. 中间对齐（vertical-align : middle）

中间对齐（vertical-align ： middle）通常使用在图片上，将图片的垂直方向的中线与文本行的中线对齐，例如有XHTML代码如下，其显示如图7-42所示。

```
p img { vertical-align : middle; }
    <p>中间对齐为基线上方0.5ex处 <img src="img/ddcat_ad.gif"alt="图片"width="180" height="60" /></p>
```

图7-42　中间对齐

中线的定义为：中线位于基线的上方，与基线的距离为小写字母x高度（即ex）的一半，如图7-36所示。而ex同字体尺寸相关，大部分浏览器认为1ex = 0.5em，因此会将基线以上四分之一em处作为中线来对齐。同在行高显示上的差别一样，在中间对齐上，各浏览器之间也稍有些差异。

7. 上标和下标

上标（vertical-align:super）使元素的基线（替换元素的底端）相对于父元素的基线升高，下标（vertical-align:sub）使元素的基线降低，移动的幅度CSS规范中没有规定，由浏览器来决定。例如有如下代码，其显示如图7-43所示。

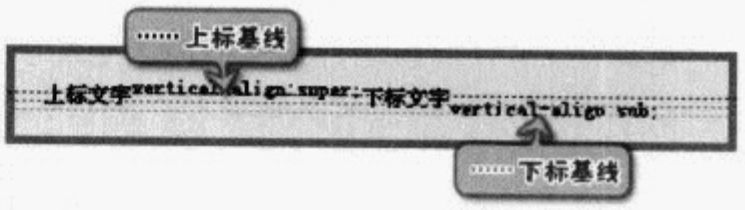


图7-43　上标和下标

```
    <p>上标文字<span style="vertical-align:super;">vertical-align:super;</span>下标文字<span style="vertical-align:sub;">vertical-align:sub;</span></p>
```

上下标不会改变元素文字的尺寸大小。

8. 长度值和百分比

和上下标类似，长度值和百分比值可使元素的基线（替换元素的底端）相对于父元素的基线升高（正值）或者降低（负值）。上下标的移动尺寸是由浏览器确定的，而设定长度值或者百分比，可以精确控制文字上下移动的幅度。

百分比与行高有关，例如有如下代码，其显示如图7-44所示。

```
p { line-height : 2em; }
    <p>行高2em，纵向百分比对齐：<span style="vertical-align:100%;">100%正数向上</span>，而<span style="vertical-align:-100%;">-100%负数向下</span>。</p>
```

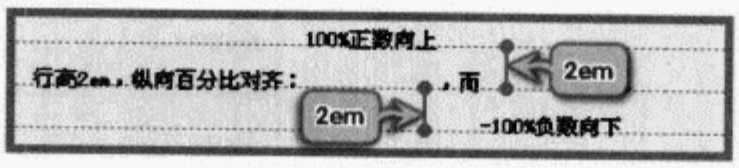


图7-44　百分比对齐

设置垂直对齐会影响到行框高，例如有如下代码，其显示如图7-45所示。

```
p { line-height : 2em; }
    <p>垂直对齐<span style="vertical-align:2em;">正数向上</span>，而<span style="vertical-align:-2em;">负数向下</span>。&lt;p&gt;行高2em，而设置垂直对齐的文字撑开了行框。</p>
```

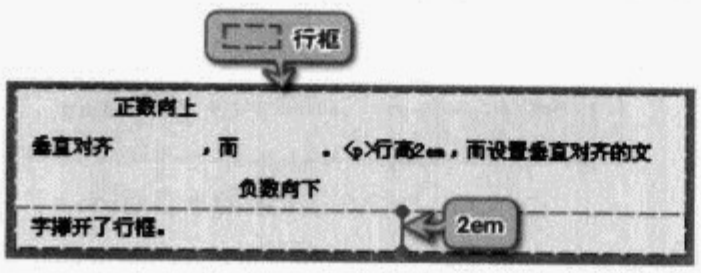


图7-45　垂直对齐对行框的影响

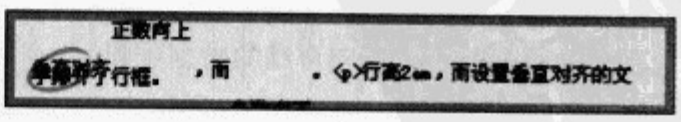
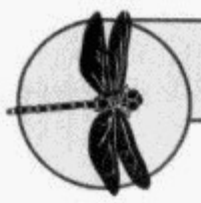


图7-46　IE中的叠加混乱



注意：在IE中设定百分比或者数值对齐会造成内容的叠加混乱，如图7-46所示。

9. 综合对齐

由于垂直对齐和行框高度有关，而行框高度由行内元素的行内框高度决定，因此在综合应用的时候，往往会出现有趣的现象。例如有如下代码，其显示如图7-47所示。

```
p { line-height:40px; }
 <p>行高40px：<span style="vertical-align: baseline;">baseline</span><span style="vertical-align: top;">top</span><span style="vertical-align: bottom;">bottom</span><span style="vertical-align: text-top;">text-top</span><span style="vertical-align: text-bottom;">text-bottom</span><span style="vertical-align: middle;">middle</span></p>
```

由图7-47可以发现，顶端对齐（top）元素的文字和文本底端（text-bottom）对齐元素的文字在一条水平线上，而底端对齐（bottom）元素的文字和文本顶端（text-top）元素的文字在一条水平线上，出现这样情况的原因是由于没有针对元素设定行高，因此元素继承了<p>元素的行高。而文本底端对齐的元素的行内框将行框撑开，如图7-48所示。

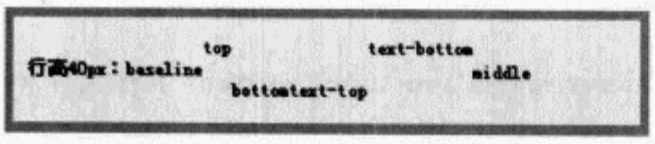


图7-47 垂直对齐的混合应用

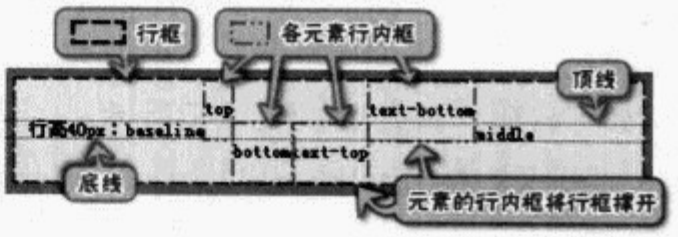


图7-48 垂直对齐的元素撑开行框

7.4.3 奇怪的IE

在本章［7.3.4 浏览器的差别与错误］一节中曾经介绍过，当行内含有图片时，在IE 6.0中行高将失效，而在对垂直对齐的处理上，IE也显示出奇怪的现象。当内容不含有图片时，在IE 6.0中，top、text-top、bottom及text-bottom都将失效，但是当行内含有图片时，这些设定又将起作用，但是行高依然是失效的，因此就出现了奇怪的垂直对齐显示，如图7-49所示。

而IE 7.0虽然修复了含有图片的内容行高失效的错误，但是，其显示也很奇怪，如图7-50所示。

图7-49 IE 6.0中奇怪的垂直对齐

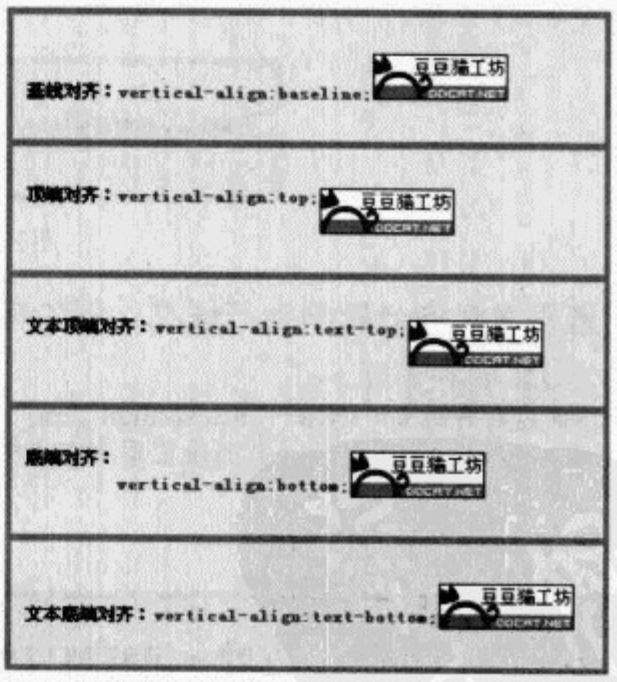


图7-50 IE 7.0中奇怪的垂直对齐

对比图7-49和图7-50，可以发现，IE 7.0对于垂直对齐的理解还是错误的。

7.4.4 文档类型与纯图片内容的垂直对齐

在前面的小节内讨论的都是文字和图文混合的情况，而当元素内容只有图片的时候，又会出现奇怪的现象，例如有如下代码，读者也可参见下载文件包内［/第2部分/第7章：文本/vertical-align2.html］文件，其显示如图7-51所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www. w3.org/TR/xhtml1
/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="zh-CN" xml:lang="zh-CN">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>vertical-align［垂直对齐］:: css文本</title>
<style type="text/css">
<!--
p {
line-height : 60px;
background : #6CF;
}
-->
</style>
</head>

<body>
<div id="verticalAlign">
 <h2>图片的垂直对齐，过度型DTD </h2>
   <p>图文混合<img src="../../img/ddcat_anim.gif" alt="图片" width="88" height="31" style="vertical-
align:middle;" /></p>
   <p><img src="../../img/ddcat_anim.gif" alt="图片" width="88" height="31" style="vertical-align:middle;"
/></p>
</div>
</body>
</html>
```

图7-51　纯图片内容的行高失效

此处需注意的是，此文档的文档类型（DOCTYPE）为过度型（Transitional）。

在本书的［2.2.3从HTML 到XHTML］一节中，曾介绍过定义文档类型（DOCTYPE）的重要性，浏览器会根据文档类型来解释文档内的元素以及CSS，如果将上例中的文档类型修改为严格的（Strict），如下所示，其显示如图7-52所示。

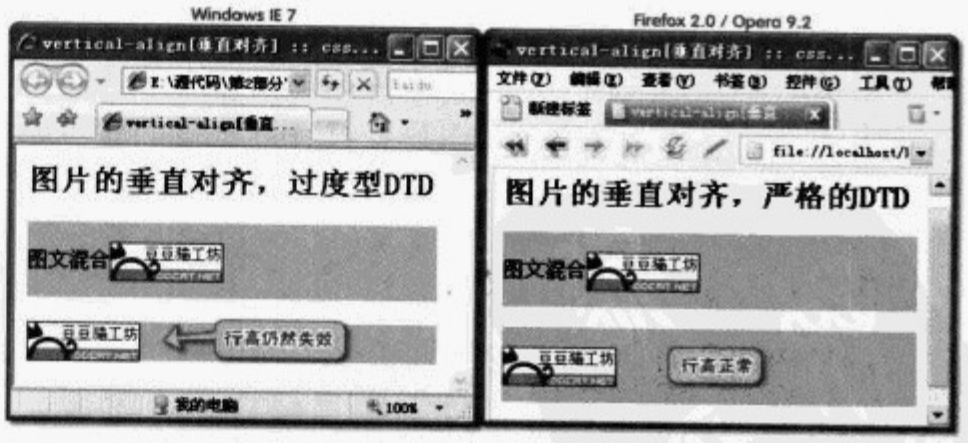
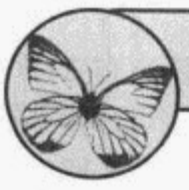


图7-52　修改文档类型对纯图片内容行高的影响

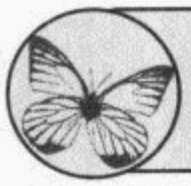
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN""http://www.w3.org/TR/xhtml1
/DTD/xhtml1-strict.dtd">
```



提示：读者可参见下载文件包内［/第2部分/第7章：文本/vertical-align3.html］文件。

7.4.5 单元格的垂直对齐

表格是比较特殊的元素，浏览器往往会对它进行特别处理。例如有如下代码，其显示如图7-53所示。



提示：本小节的示例页面请参见下载文件包内［/第2部分/第7章：文本/vertical-align_td.html］文件。

```
p, td { height:3em; }
<p>未设定行高的段落</p>
<table>
 <tr>
    <td>未设定行高的单元格</td>
 </tr>
</table>
```

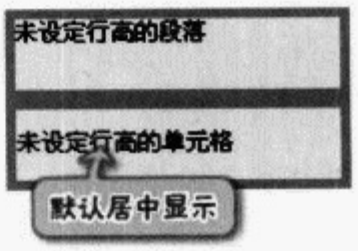


图7-53 单元格的默认显示

在本章［7.3 行高］一节中曾经介绍过：行框同元素的行高（line-height）有关，而和高度（height）无关。<p>元素虽然设定了高度，但是没有设定行高，因此文字并未垂直居中，而单元格内的文字却会在垂直方向居中显示。

如果对单元格设置行高但是不设置垂直对齐，例如有如下代码，其在浏览器的显示如图7-54所示。

```
td {
height:70px;
line-height:40px;
}
<table>
 <tr>
    <td>单元格行高40px：<img src="img/btn1.gif" alt="img" width="21" height="20" style="vertical-align: baseline;" /><img src="img/btn1.gif" alt="img" width="21" height="20" style="vertical-align: top;" /><img src="img/btn1.gif" alt="img" width="21" height="20" style="vertical-align: bottom;" /><img src="img/btn1.gif" alt="img" width="21" height="20" style="vertical-align: text-top;" /><img src="img/btn1.gif" alt="img" width="21" height="20" style="vertical-align: text-bottom;" /><img src="img/btn1.gif" alt="img" width="21" height="20" style="vertical-align: middle;" /></td>
 </tr>
</table>
```

图7-54 单元格中元素的垂直对齐

由图7-54可以发现，当单元格高度大于行框高度时，行框在单元格内会垂直居中显示。当为单元格<td>设定垂直对齐方式以后，其在浏览器的显示如图7-55所示。

```
td { height:70px; }
<table>
 <tr>
  <td>无设定</td>
  <td style="vertical-align : baseline;">基线baseline</td>
  <td style="vertical-align : top;">top</td>
  <td style="vertical-align : bottom;">bottom</td>
  <td style="vertical-align : text-top;">text-top</td>
  <td style="vertical-align : text-bottom;">text-bottom</td>
  <td style="vertical-align : middle;">middle</td>
 </tr>
</table>
```

而如果为<td>设定行高为40px，则上述代码显示如图7-56所示。

从图7-55和图7-56中可以看到不同浏览器在处理方式上的差异。关于单元格垂直方向的视觉格式化，请参见本书［11.3.7 单元格内容的对齐］一节。

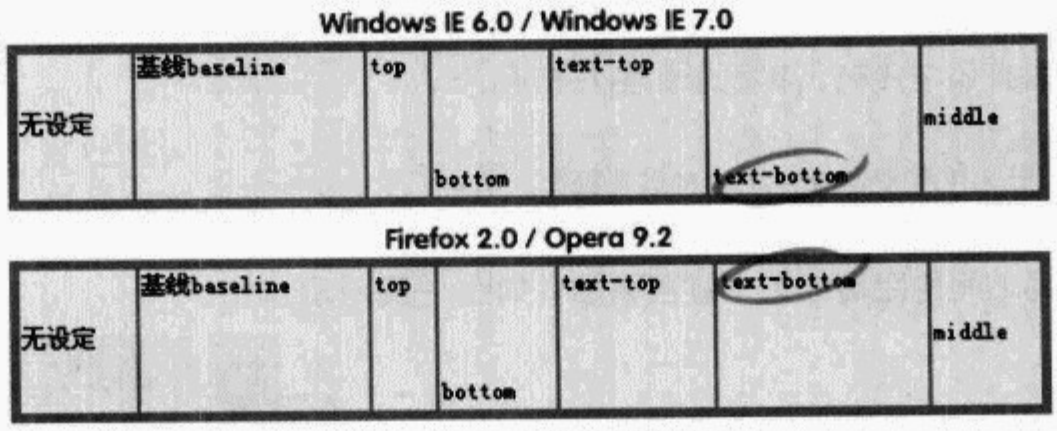


图7-55　单元格的垂直对齐

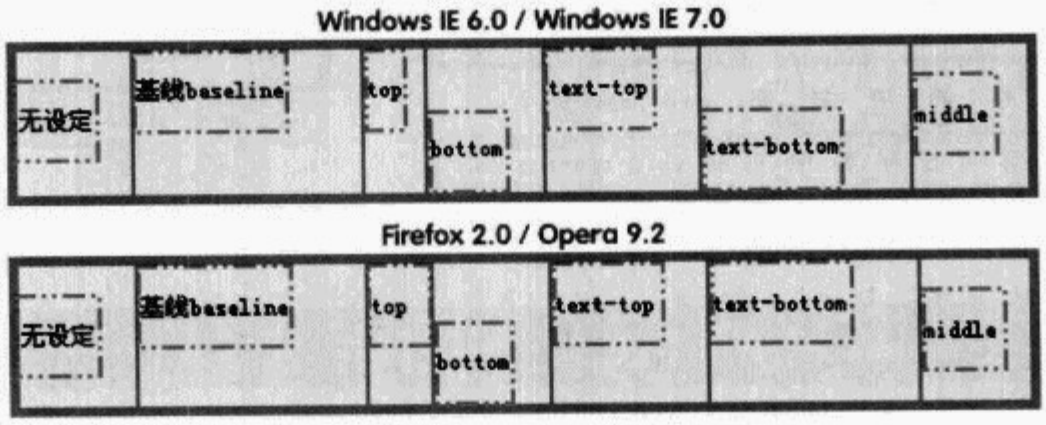
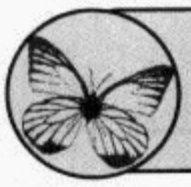


图7-56　设定行高的单元格的垂直对齐

7.5 单词间隔（word-spacing）和字母间隔（letter-spacing）

字母间隔（letter-spacing）和单词间隔（word-spacing）都是从英文排版而来，这两个属性的差别在于：单词间隔是以空格来判断的。



提示：本小节的示例页面请参见下载文件包内［/第2部分/第7章：文本/letter-spacing.html］文件。

7.5.1 单词间隔：word-spacing属性

word-spacing属性的具体定义列表如下：

语法	word-spacing : normal | <长度> | inherit
说明	设置元素内单词之间的间隔，会受文字对齐方式影响
值	normal：默认间隔，由当前字体和（或）用户端定义。 长度：长度值，允许为负值
初始值	normal
继承性	继承
适用于	所有元素
媒体	视觉
计算值	如果是“normal”则为0，否则为绝对长度

浏览器不能判定文字的内容，因此单词的判定是以空格为准的，没有空格分隔的字母和汉字，

被认为是一个单词，因此也就不会对应单词间隔，而如果在汉字中间插入空格，则可以看到单词间隔的效果，例如有如下代码，其显示如图7-57所示。

```
p { word-spacing: 1em; }
<p>没有空格的中文和英文word-spacing都没有效果。</p>
<p>含 有 空 格 的 中 文 和 word spacing</p>
```

负值会使单词之间的距离缩近，甚至叠加，如图7-58所示。

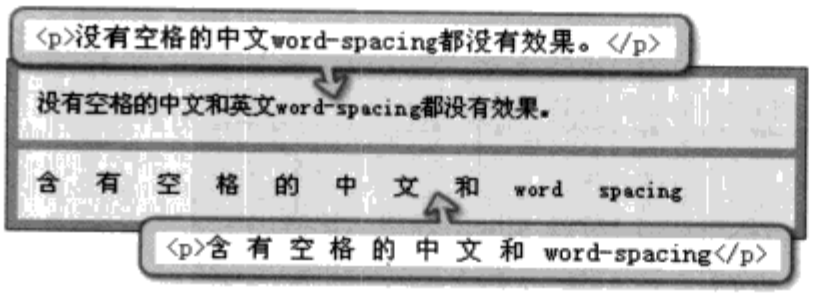


图7-57 空格与word-spacing

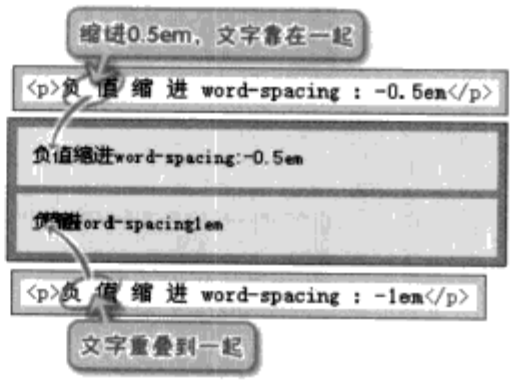


图7-58 负值单词间隔word-spacing

7.5.2 字母间隔：letter-spacing属性

letter-spacing属性的具体定义列表如下：

语法	letter-spacing : normal | <长度> | inherit
说明	设置元素内字母之间的间隔。该属性将指定的间隔添加到每个字母之后，但最后一个字将被排除在外。会受文字对齐方式影响
值	normal：默认间隔，由当前字体和（或）用户端定义，一般情况下为0。 长度：允许为负值
初始值	normal
继承性	继承
适用于	所有元素
媒体	视觉
计算值	“normal”或绝对长度

字母间隔设定每个字母和汉字之间的距离，例如有如下代码，其显示如图7-59所示。

```
p { letter-spacing: 1em; }
<p>字母间隔letter spacing</p>
```

图7-59 字母间隔letter-spacing

7.5.3 水平对齐的影响和继承

文字的水平对齐方式（text-align）将影响这两种间隔，特别是两端对齐（text-align : justify;），其显示如图7-60所示。字母间隔和单词间隔都是可继承的属性，因此定义的时候要特别注意，如果要消除影响，就要对元素的后代元素重新定义相关属性。

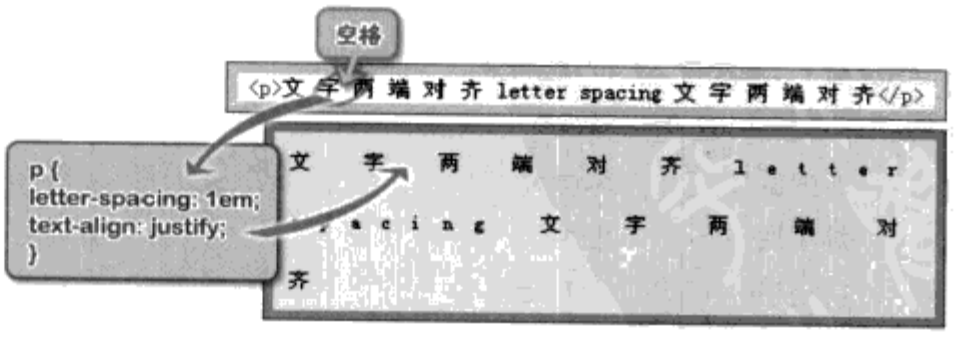


图7-60 水平对齐方式对间距的影响

7.6 文本转换：text-transform属性

text-transform属性设置元素内文本的大小写方式，其语法如下，在浏览器内的效果如图7-61所示。

语法	text-transform : capitalize | uppercase | lowercase | none | inherit
说明	设置元素内文本的大小写
值	capitalize：将每个单词的第一个字母转换成大写，其余无转换发生。 uppercase：转换成大写。 lowercase：转换成小写。 none：默认值，无转换发生
初始值	none
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

提示：本小节的示例页面可以参见下载文件包内［/第2部分/第7章：文本/text-transform.html］文件。

对于首字母大写，不同浏览器的处理方式不尽相同，如图7-62所示。

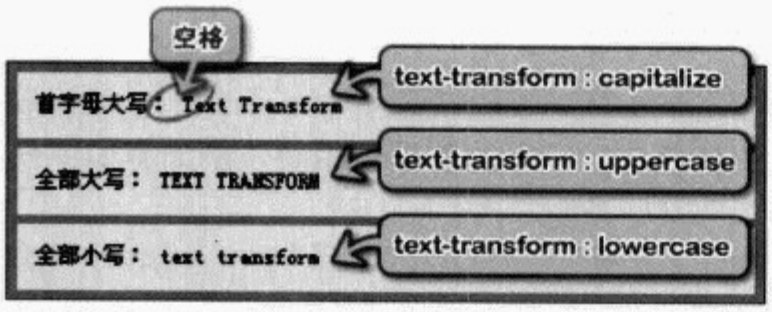


图7-61　文本转换（text-transform）

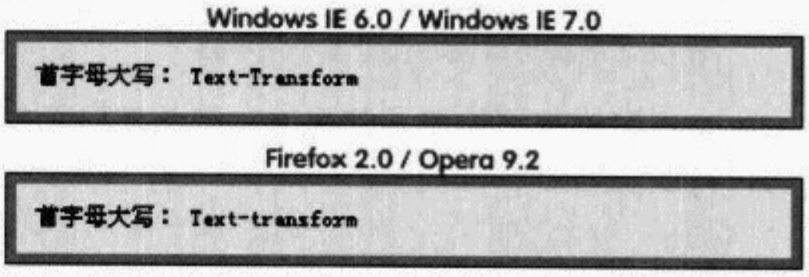


图7-62　浏览器对于首字母大写的区别

7.7 文本装饰：text-decoration属性

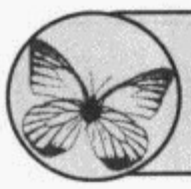
text-decoration属性是个属性值比较复杂的属性。其最常见的应用就是对于带有链接的文字的设定（显示或者不显示下划线），其实该属性可以应用在任何元素上。

text-decoration属性语法如下，各属性值的显示如图7-63所示。

语法	text- decoration : none | [underline || overline || line-through || blink] | inherit
说明	设置元素内文本的装饰
值	none（默认值，无装饰）、underline（下划线）、overline（上划线）、line-through（贯穿线）、blink（闪烁）

续表

初始值	none（个别元素不是）
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	同指定值



提示： 本小节的示范页面可以参见下载文件包内［/第2部分/第7章：文本/text-decoration.html］文件。

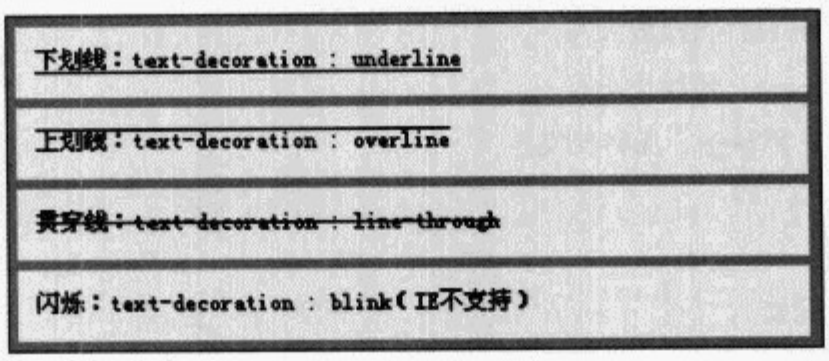


图7-63 文本装饰（text-decoration）各值的表现形式

一般的（X）HTML元素文本装饰（text-decoration）的默认值是“none”，不过也有一些元素默认值可能不同，例如，链接的默认值为“underline”，如图7-64所示。

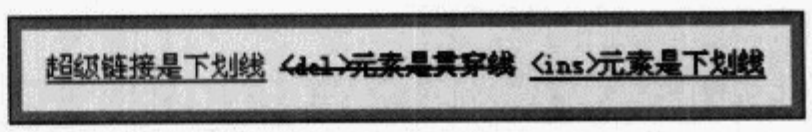


图7-64 不同html元素文本装饰（text-decoration）的默认值不同

同时文本装饰可以同时设置多个属性值，多个值之间用空格分开，如果最后一个值为“none”，则会清除前面的设定，不显示装饰。例如有如下代码，其显示如图7-65所示。

```
p { text-decoration : underline overline ; }  /* 关键字之间以英文空格分隔 */
<p>文本装饰可以同时设置多个值</p>
```

文本装饰是不被继承的属性，因此其显示的线的颜色与父元素的前景色相同，例如有如下代码，其显示如图7-66所示。

```
p {
color : #000;
text-decoration : underline;
}
strong {
color : #C30;
}
<p>文本装饰不被继承，<strong>子元素的线的颜色同父元素一样</strong></p>
```

图7-65 文本装饰可同时设定多个值

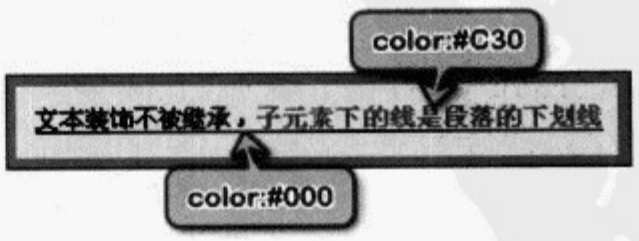


图7-66 文本装饰装饰线的颜色

其中元素并没有继承父元素<p>的文本装饰，显示的下划线是<p>元素的，而不是元素的，如果增加对元素的定义如下，则其显示如图7-67所示。

```
strong { text-decoration : overline; }
```

由图7-67可知，如果对某个元素定义了文本装饰，则其装饰线将贯穿整个元素，包括其子元素。

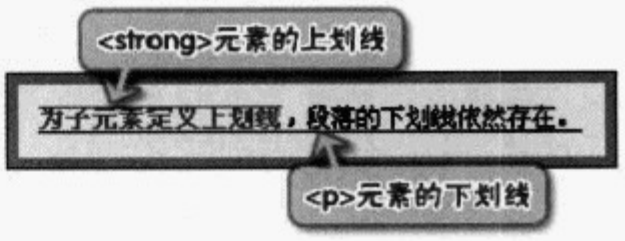


图7-67　段落的装饰线会贯穿子元素

7.8 空白：white-space属性

white-space属性影响到浏览器对于单词和文本行间空白的处理方式。本小节的示范页面请参见下载文件包内［/第2部分/第7章：文本/white-space.html］文件。

7.8.1 语法

white-space属性的具体定义列表如下：

语法	white-space : normal | pre | nowrap | pre-wrap | pre-line | inherit
说明	设置元素内空格的处理方式
值	normal：默认处理方式，文本自动处理换行，假如抵达容器边界内容会转到下一行。 pre：换行和其他空白字符都将受到保护。 nowrap：强制在同一行内显示所有文本，直到文本结束或者遭遇 元素。 pre-wrap：CSS 2.1，换行和其他空白字符都将受到保护，但是也会自动回行。 pre-line：CSS 2.1，空格及制表符会被压缩，换行将受到保护，但是也会自动回行
初始值	normal
继承性	不继承
适用于	所有元素（CSS 2.1）、块级元素（CSS 1和CSS 2）
媒体	视觉
计算值	同指定值

7.8.2 属性值详解

“pre-wrap”和“pre-line”是CSS 2.1增加的属性值，目前只有少数的浏览器支持这2个值。

1. normal

对于文本内的空格，（X）HTML中规定，多个空格会被压缩成一个空格，元素的内容会自动回行，如图7-68所示。

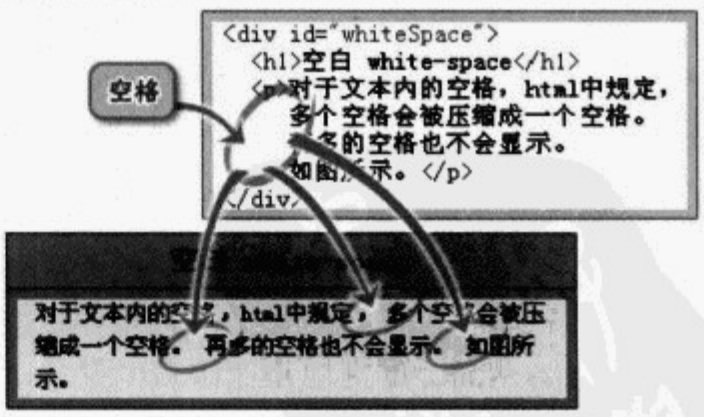
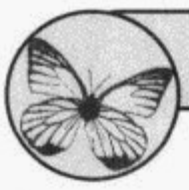


图7-68　HTML对于空格的处理



提示：在（X）HTML中，换行（回车）、空格和TAB（制表符），都被认为是空格。

这个效果等同于设置元素的white-space属性为normal，如：

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

2. pre

如果设定元素“white-space : pre”，则所有的空格将不会被压缩，而是按照原样显示，此时的元素类似（X）HTML中的<pre>元素，如图7-69所示。

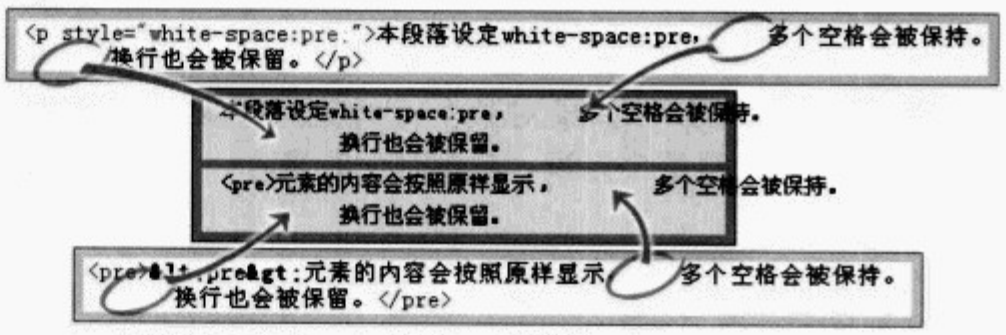


图7-69 “white-space:pre”与<pre>元素

3. nowrap

设定元素“white-space : nowrap”将强制在同一行内显示所有文本，直到文本结束或者遭遇
元素，如图7-70所示。

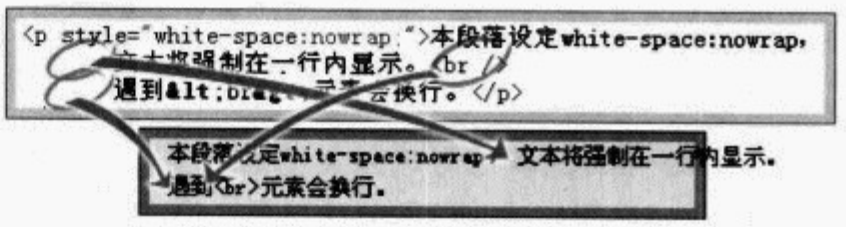
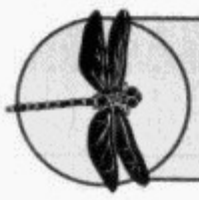


图7-70 white-space : nowrap



注意：对于溢出内容的处理，视元素的overflow属性而定，可以参见本书［9.6.1 溢出：overflow属性］一节。

4. pre-wrap

设定元素“white-space : pre-wrap”，则所有的空格、制表符和换行将不会被压缩，而是按照原样显示，但是同时会保持内容自动回行，如图7-71所示。

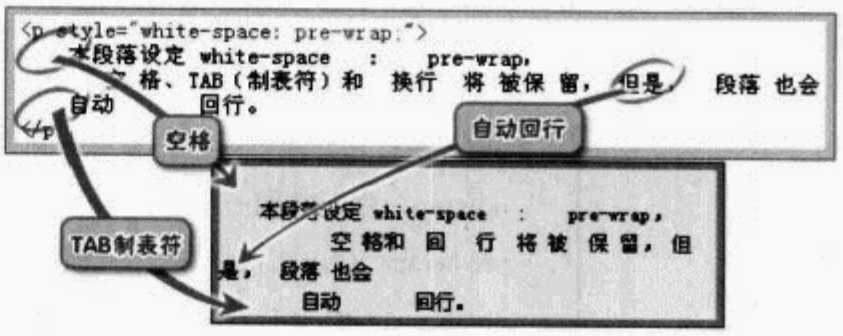


图7-71 white-space : pre-wrap

5. pre-line

设定元素“white-space : pre-line”，则所有的空格和制表符将被压缩，但是换行将被保留，同时内容会自动回行，如图7-72所示。

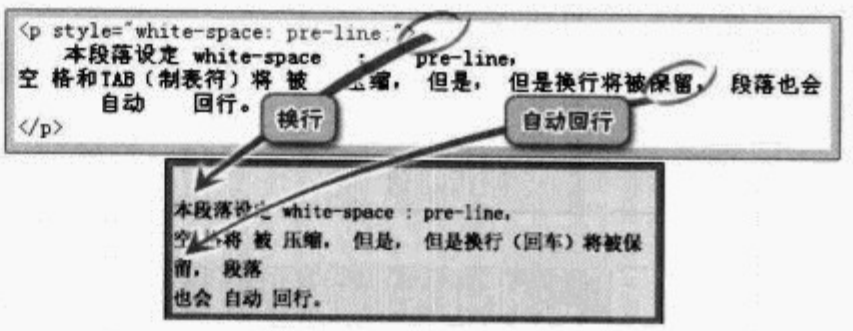
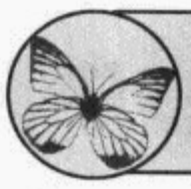


图7-72 white-space : pre-line



提示：目前IE 6.0 / 7.0以及Firefox 2.0不支持这2个值。Safari 3.0浏览器支持这2个值，而Opera 9.2只支持“pre-wrap”。

7.8.3 应用：显示不回行文本

white-space属性应用比较多的是“white-space : nowrap”，例如有设计图如图7-73所示。

根据设计，每行显示一个链接的标题文字，而且最多显示1 1个汉字，但是当采用程序来动态生成内容的时候，由数据库取出的标题内容可能会超过1 1个汉字，因此页面显示有可能会被破坏，如图7-74所示。

因此，需要设定列表项<li>的white-space属性为“nowrap”，使得文字在一行内显示，同时还需要设定“overflow : hidden;”来隐藏多余的文字，CSS代码如下：

```
li {
width : 11em; /* 设定宽度，overflow才会生效 */
```

```
white-space : nowrap;
overflow : hidden;
}
```

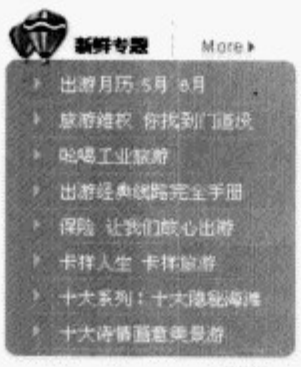


图7-73　设计图

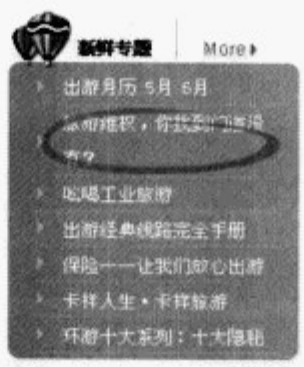


图7-74　动态生成的内容破坏了设计效果

7.9 文本阴影：text-shadow属性

text-shadow属性是CSS 2增加的属性，但是由于大部分的浏览器不支持此属性，因此在CSS 2.1中又被删除了，其具体语法如下：

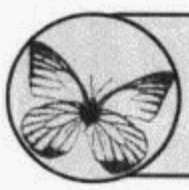
语法	text-shadow : none | [<颜色> || <长度1> <长度2> <长度3>? ,] * [<颜色> || <长度> <长度> <长度>?] | inherit
说明	设置元素中文本的文字是否有阴影及模糊效果
值	颜色：CSS中合法的颜色值，指定阴影的颜色，如果不设定则采用color属性的值。 长度1：长度值，可为负值，指定阴影的水平延伸距离。 长度2：长度值，可为负值，指定阴影的垂直延伸距离。 长度3：长度值，不可为负值，指定模糊效果的作用距离。 如果仅仅需要发光效果，将长度1和长度2全部设定为0
初始值	none
继承性	不继承
适用于	所有元素
媒体	视觉

text-shadow属性接受一个英文逗号“,”分割的阴影效果列表，并应用到该元素的文本上。阴影效果按照给定的顺序应用，因此有可能出现互相覆盖，但是它们永不会覆盖文本本身。

每个阴影效果必须指定阴影偏移，而模糊半径、阴影颜色是可选值。阴影偏移由两个长度值指定到文本的距离，长度1指定距离文本右边的水平距离，负值将会把阴影放置在文本的左边。长度2指定距离文本下边的垂直距离，负值将会把阴影放置在文本上方。

在阴影偏移之后，可以指定一个模糊半径。模糊半径是个长度值，指出模糊效果的范围。

例如有如下代码，其在Safari 3.0内的浏览效果如图7-75所示。



提示：本小节的示例页面可以参见下载文件包内［/第2部分/第7章：文本/text-shadow.html］文件。

```
.shadow1 { text-shadow: #C00 0.2em 0.3em 0.3em; }
.shadow2 { text-shadow: #C00 -0.2em -0.3em 0.3em; }
```

```
.shadow3 { text-shadow: 3px 3px red, yellow -3px 3px 2px, 3px -3px; }
.shadow4 { text-shadow:#f00 0 0 0.3em; }
<p class="shadow1">文本阴影1  text-shadow: #C00 0.2em 0.3em 0.3em</p>
<p class="shadow2">文本阴影2  text-shadow: #C00 -0.2em -0.3em 0.3em</p>
<p class="shadow3">文本阴影3  text-shadow: #C00 0.2em 0.3em 0.3em</p>
<p class="shadow4">文本发光  text-shadow: #f00 0 0 0.3em</p>
```

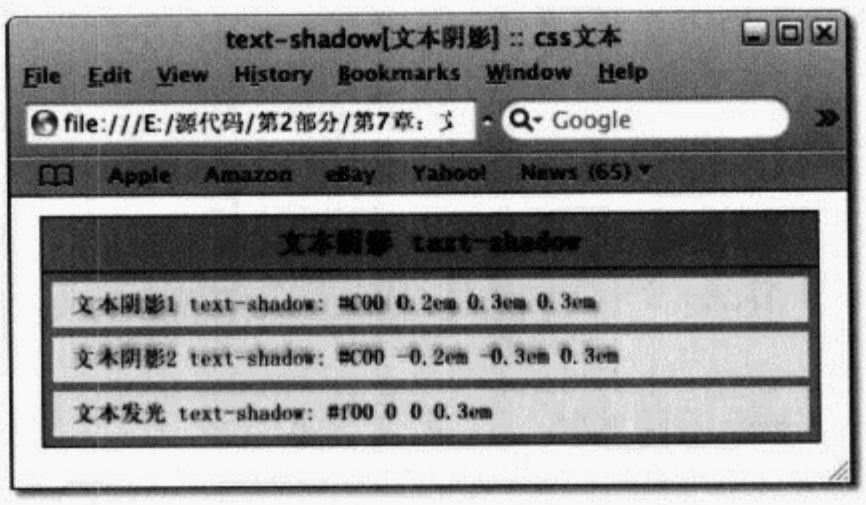
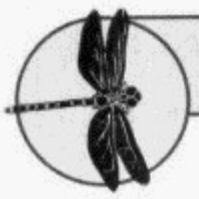


图7-75 文本阴影的浏览器效果



注意： Safari 3.0不支持多个阴影效果的列表。

7.10 文字方向direction和编码方式unicode-bidi

文字方向（direction）属性与编码方式（unicode-bidi）属性是CSS 2增加的属性，这两个属性一般需要同时设定。direction属性语法如下：

语法	direction : ltr | rtl | inherit
说明	设置元素文本流入的方向
值	ltr：默认值，文本从左到右流入。 rtl：文本从右到左流入。 inherit：文本流入方向由继承获得
初始值	ltr
继承性	继承
适用于	所有元素（CSS 2.1）、块级元素（CSS 1和CSS 2）
媒体	视觉
计算值	同指定值

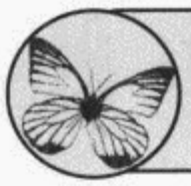
unicode-bidi属性语法如下：

语法	unicode-bidi : normal | bidi-override | embed
说明	用于同一个页面里存在从不同方向读进的文本显示
值	normal：默认值。 bidi-override：严格按照direction属性的值重排序。 embed：direction属性的值指定嵌入层，在元素内部进行隐式重排序
初始值	normal

续表

继承性	继承
适用于	与direction属性一起使用
媒体	视觉
计算值	同指定值

在本章［7.2 文本缩进：text-indent属性］一节中曾经介绍过缩进的同文字方向的关系，对于大部分的文字都是从左向右书写，少部分的文字则是从右向左书写，如阿拉伯文。



提示：本小节的示例页面请参见下载文件包内［/第2部分/第7章：文本/direction.html］文件。

direction属性指定了块的基本书写方向，以及Unicode双向算法中嵌入和超越的方向，另外，它还规定了表格列布局的方向、水平溢出的方向，以及块设置了“text-align: justify”时，最后一个不完全的行的位置。只设定direction属性不会影响拉丁文的字母、数字和汉字的显示方向，它们总是以ltr值被显示，如图7-76所示。

当对inlene元素设定direction属性则必须设定unicode-bidi属性为“embed”或 “bidi-override”，其差别如图7-77所示。

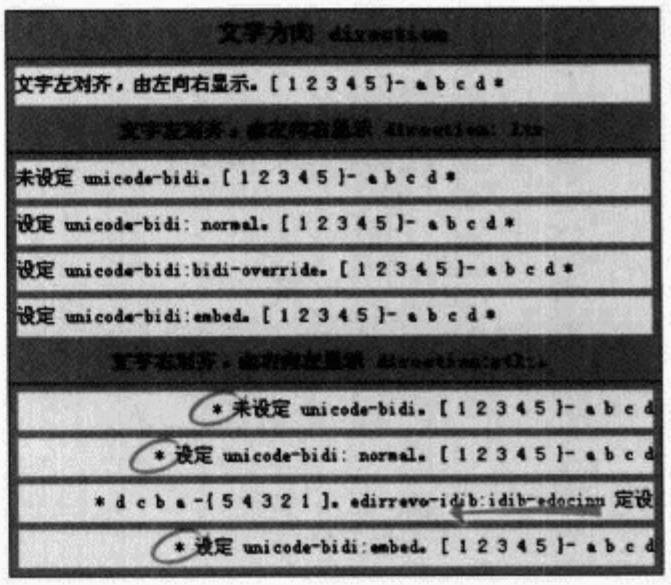


图7-76　文字方向各值示意

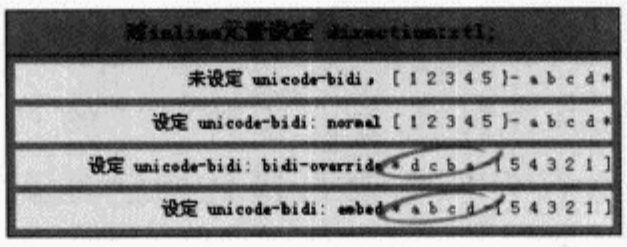
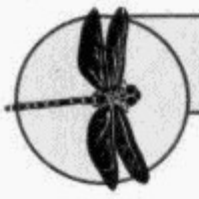


图7-77　对行内元素设定文字方向



注意：unicode-bidi属性必须同direction属性一起使用。

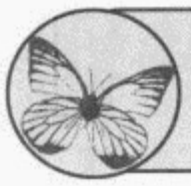
第8章 框模型

框模型（Box Model，也译为“盒模型”）是CSS非常重要的概念，也是比较抽象的概念。文档树中的元素都产生矩形的框（Box），这些框影响了元素内容之间的距离、元素内容的位置、背景图片的位置等。而浏览器根据视觉格式化模型（Visual Formatting Model）来将这些框布局成访问者看到的样子。

因此，要掌握使用CSS布局的技巧，就需要深入了解框模型和视觉格式化模型的原理。本章着重介绍框模型及简单的视觉格式化内容，浮动和定位等视觉格式化模型将在下一章内详细介绍。

8.1 框模型（Box Model）

浏览器内显示的元素都可以看作是一个装了东西的矩形的盒子，这些矩形的盒子嵌套、叠加或者并列在一起，形成了页面。



提示："box model"往往被译为“盒模型”。但是盒子是具有厚度的，也就是说盒子是三维的，而框则没有厚度，是二维的，因此本书采用了“框模型”这个译法。

每一个元素的“框（Box）”由以下几部分组成。

● 内容（content）。如文字、图片或者其他元素等，内容也可以看作是一个长方形的框，width（宽度）和height（高度）2个CSS属性设定的就是内容框的宽度和高度。

● 边框（border）。边框（也译为边界）是可以具体显示出来的，可以设定宽度、外观样式和颜色。

● 补白（padding）。补白（也译为填充、内边距、内补丁等）是内容框与边框之间的距离，补白部分显示的是背景。

● 边距（margin）。边距（也译为边白、外边距、外补丁等）是边框外的透明区域，用来设定本元素与其他元素之间的距离。

一个元素框又有上、右、下、左4个方向的边，如图8-1所示。

由图8-1可以发现，1个元素所占的区域其实是由几个矩形框组成：元素的内容框、补白形成的框、元素的边框以及边距形成的框。这些框的边缘又有如下定义。

● 元素内容框的边缘，称为“内容边（content edge）或内边（inner edge）”，4条内容边形成内容框（content box）。

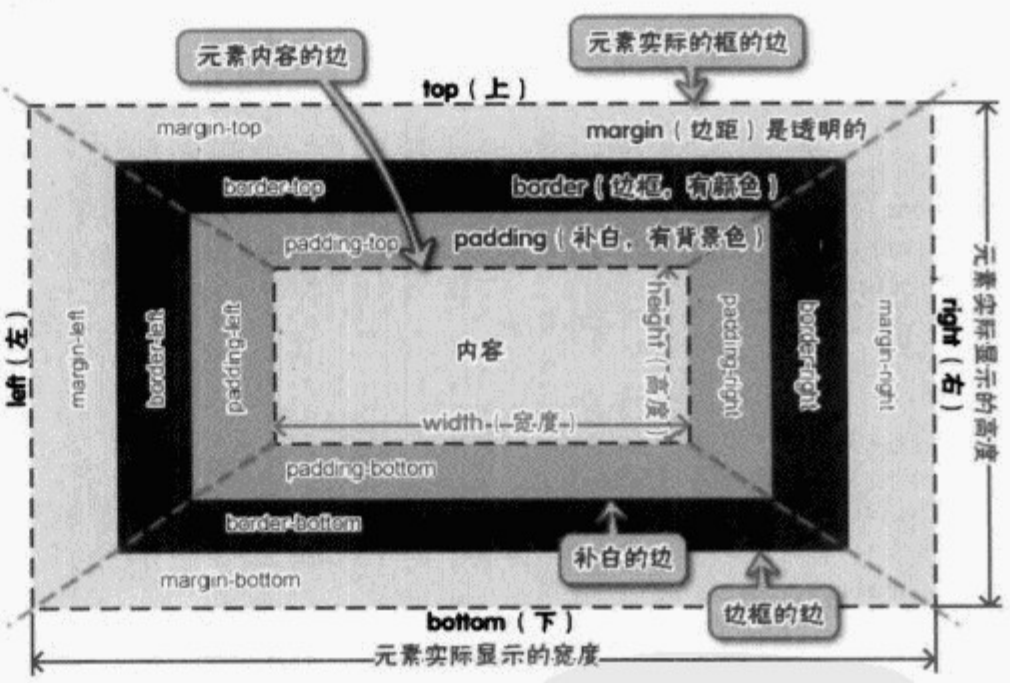


图8-1　框模型示意

● 补白形成的框的外边缘，称为“补白边（padding edge）”，补白边围绕框的补白。如果补白宽度为0，则补白边和内容边重合。框的补白边定义了包含块的边。4条补白边形成补白框（padding box）。

● 边框形成的框的外边缘，称为“边框边（border edge）”。如果边框宽度为0，则边框边和补白边重合。4条边框边形成边框框（border box）。

● 边距形成的框的外边缘，称为“边距边（margin edge）”或“外边（outer edge）”，边距边围绕框的边距。如果边距宽度为0，则边距边和边框边重合。4条边距边形成边距框（margin box）。

内容框的尺寸（宽度和高度）取决于若干个因素：产生框的元素是否设定了width属性或height属性，框是否包含文本或其他框，框是否是一个表格等。例如下列代码其内容框与框之间

的关系，如图8-2和图8-3所示。

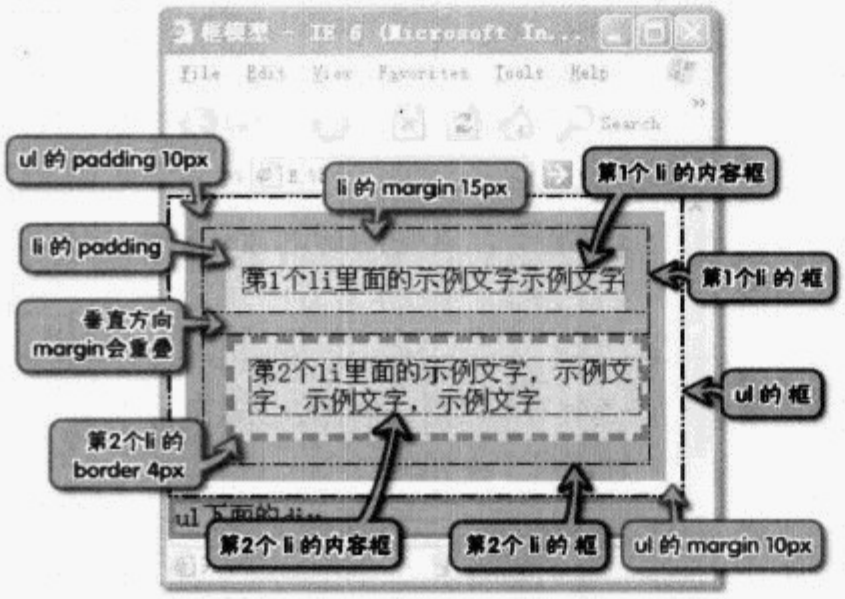


图8-2 内容框与框的关系

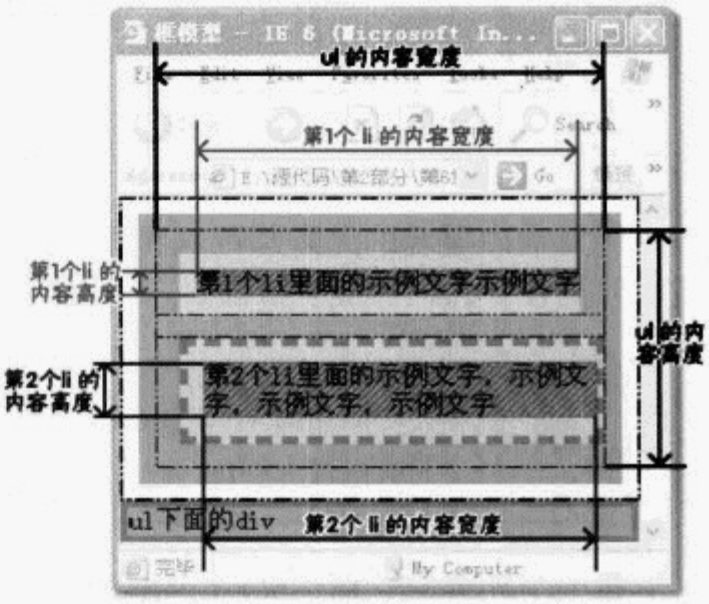
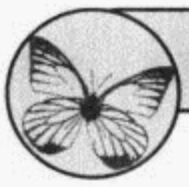


图8-3 内容的尺寸与框的关系



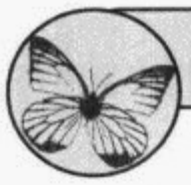
提示：读者可以参见下载文件包内［/第2部分/第8章：框模型/box_model.html］文件。

```
* {   /* 清除浏览器默认的样式 */
margin : 0;
padding : 0;
color : #666;
}
div {
background : #FC6;
border : 4px solid #F90;
}
ul {
background : #FC6;
padding : 10px;   /* 缩写：上右下左全部都是10px */
margin : 10px;
list-style : none;  /* 不显示圆点 */
}
li {
background : #FF9;
margin : 15px;
padding : 10px 0 10px 10px;    /* 缩写的顺序是：上 右 下 左 */
}
.sample {
border-width : 5px;
border-style : dashed;
border-color : #F90;
margin-right : 0;
}
<ul>
 <li>第1个li里面的示例文字示例文字</li>
 <li class="sample">第2个li里面的示例文字</li>
</ul>
<div>ul下面的div</div>
```

由图8-2和图8-3，可以发现：

元素的框宽度 = 左边距（margin-left）+ 左边框宽（border-left-width）+ 左补白（padding-left）+ 内容宽度（width）+ 右补白（padding-right）+ 右边框宽（border-right-width）+ 右边距（margin-right）

元素的框高度 = 上边距（margin-top）+ 上边框宽（border-top-width）+ 上补白（padding-top）+ 内容高度（height）+ 下补白（padding-bottom）+ 下边框宽（border-bottom-width）+ 下边距（margin-bottom）



提示：关于margin在垂直方向重叠的情况，可以参见本章［8.9.2.2 边距的重叠］一节。

在IE 5.5及更早的版本，以及在怪异模式中的IE 6.0/7.0中，会错误地将框模型理解为：

```
width = border-left + padding-left + 内容宽度 + padding-right + border-right
height = border-top + padding-top + 内容高度 + padding-bottom + border-bottom
```

例如下列代码：

```
div {
margin : 10px;
padding : 15px;
width : 300px;
border : 5px solid #ccc;
}
```

则 的框宽度应为360px（10px + 5px + 15px + 300px + 15px + 5px +10px），而在IE的错误框模型中，框的宽度为320px（10px + 300px +10px），实际的内容宽度为260px（300px - 15px×2 - 5px×2）。因此会造成元素尺寸显示的不正确。关于浏览器的怪异模式，可以参见本书［16.1.3 浏览器的的工作模式］一节。

8.2 包含块（Containing Block）

包含块是视觉格式化模型的一个重要概念，它与框模型类似，也可以理解为一个矩形，而这个矩形的作用是为它里面包含的元素提供一个参考，元素的尺寸和位置的计算往往是由该元素所在的包含块决定的。在本章只简单介绍一些视觉格式化的必要知识，在下一章将详细介绍视觉格式化的内容。

8.2.1 视口（viewport）

浏览器的窗口一般由3个部分组成，如图8-4所示。

连续媒介的用户端（例如电脑的浏览器）通常提供给用户一个视口（屏幕上的一个窗口或浏览区域），用户通过它来浏览文档。当视口尺寸改变时，例如用户调整了浏览器的窗口大小，用户端可能会改变文档的布局。

如果视口比文档设定的大小要小，用户端往往会提供滚动机制（例如浏览器的滚动条）。对于一个渲染区域而言，最多只能有一个视口，不过用户端可以对多个渲染区域加以渲染（即对同一文档提供不同的视口）。

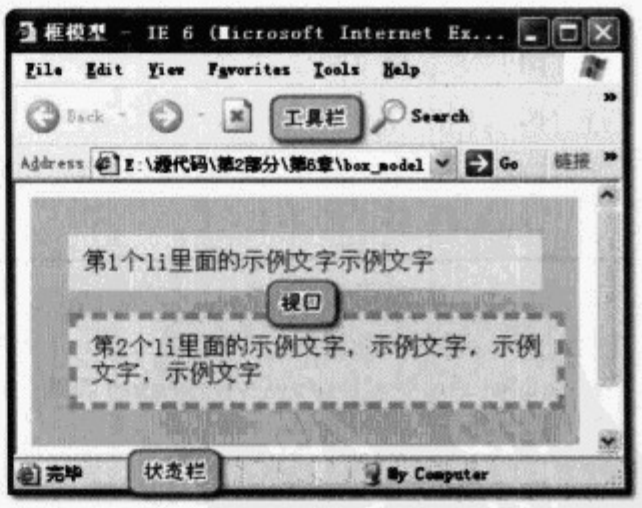


图8-4　浏览器的视口

8.2.2 包含块

一个元素的包含块的定义如下。

（1）根元素存在的包含块称为初始包含块，在（X）HTML中，根元素是<html>元素（尽管有的浏览器会不正确地使用<body>元素），而初始包含块的direction属性与根元素相同。初始包含块的宽度可以由根元素的width属性指定。如果该属性取值为“auto”，用户端提供初始宽度（如

视口的当前宽度）。初始包含块的高度可以由根元素的height属性指定。如果该属性取值为“auto”，包含块的高度将调整以适应文档内容。初始包含块不可以被定位或浮动（即用户端忽略根元素的position和float属性）。

（2）对于其他元素，如果该元素的定位（position）为“relative（相对）”或者“static（静态）”，它的包含块由它最近的块级、单元格（table cell）或者行内块（inline-block）祖先元素的内容框创建。

（3）如果元素设定了“固定定位（position: fixed）”，包含块由视口创建。

（4）如果元素设定了“绝对定位（position: absolute）”，包含块由最近的position属性为“absolute”、“relative”或者“fixed”的祖先元素创建，具体方法如下。

如果祖先元素是行内元素，包含块取决于祖先元素的direction属性。

- 如果direction为“ltr”，包含块的顶、左边是该祖先元素创建的第一个框的顶、左补白边，它的底、右边是该祖先元素创建的最后一个框的底、右补白边。
- 如果direction为“rtl”，包含块的顶、右边是该祖先元素创建的第一个框的顶、右补白边，它的底、左边是该祖先元素创建的最后一个框的底、左补白边。

否则，祖先的补白边形成包含块。如果不存在这样的祖先元素，则元素的包含块为初始包含块。

在浏览器生成显示的页面的时候，每一个框都有一个定位，这个定位受其包含块的影响，不过它不被包含块所限制，而且可能会溢出到包含块之外。例如，下面的XHTML文档，其文档结构及在浏览器内的显示如图8-5所示。当元素都没有定位的时候，无定位元素的包含块的创建如下表所示。

图8-5 文档结构图

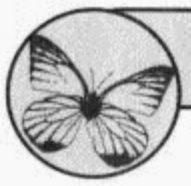
产生框的元素	创建包含块的元素
body	初始包含块（与用户端相关）
div1	body
p1	div1
p2	div1
em1	p2
strong1	p2

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/ TR/xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="zh-CN" lang="zh-CN">
<head>
  <title>包含块</ title>
 </ head>
 <body id="body">
  <div id="div1">
   <p id="p1">这里是第一个段落p的文字。</p>
   <p id="p2">这里是<em id="em1">第<strong id="strong1">2</strong>个</em>段落的文字。</p>
  </div>
 </ body>
</ html>
```

此时，如果对层“div1”设定定位如以下左边代码，则“div1”层的包含块不再是“body”，而是初始包含块（因为没有其他定位祖先框）。此时如果再增加对“em1”的定位如以下右边代码：

```
#div1 {
background: #0C6;
position: absolute;
left: 20px;
top: 30px;
}
```

```
#em1 {
background: #FC3;
position: absolute;
left: 50px;
top: 40px;
}
```



提示： 读者可以参见下载文件包内［/第2部分/第8章：框模型/containing_block.html］文件。

则元素设定了定位的包含块的创建如下表所示。

产生框的元素	创建包含块的元素
body	初始包含块
div1	初始包含块
p1	div1
p2	div1
em1	div1
strong1	em1

“em1”定位后，它的包含块变为由它最靠近的祖先定位框“div1”创建的那个框，如图8-6所示。

因此对元素的左（left）和上（top）的位置的计算，如图8-7所示。

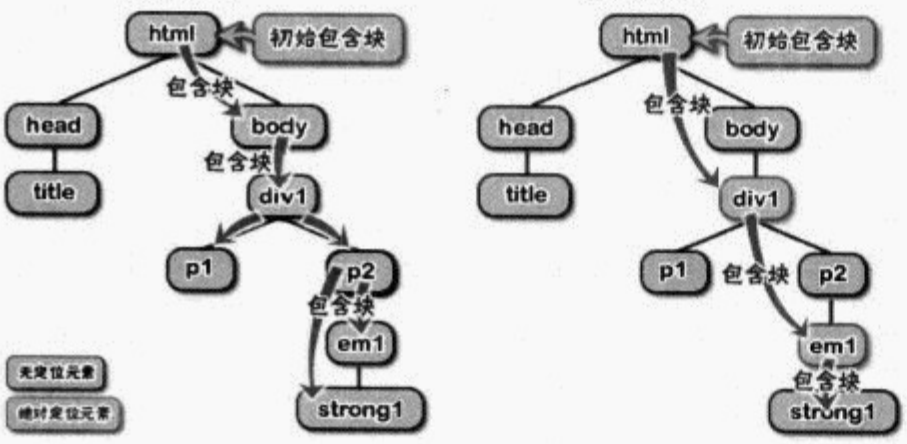


图8-6　改变元素定位属性对包含块的影响

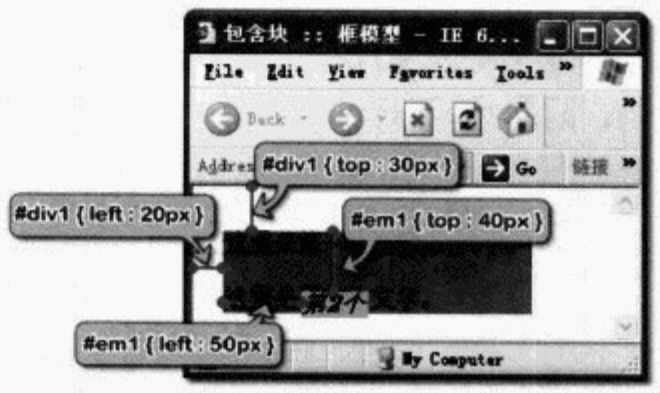
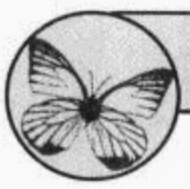


图8-7　包含块的改变对元素定位的影响



提示： 包含块的概念对于理解CSS“定位”的概念非常重要。

8.3 宽度：width属性

width属性用以设定块级元素和替换元素产生的内容框的宽度。本节示例代码可以参见下载文件包内［/第2部分/第8章：框模型/width.html］文件。

8.3.1 语法

width属性具体定义列表如下：

语法	width : <长度> | <百分比> | auto | inherit
说明	设定块级元素和替换元素产生的内容框的宽度
值	长度：合法的长度值，不可为负值。 百分比：百分比基数为元素包含块的宽度，不可为负值。 auto：宽度取决于其他属性的值，参见［8.10.1 块级元素的水平格式化］小节
初始值	auto
继承性	不继承
适用于	所有元素，除了非替换的行内元素、表格行和行组（row groups）
媒体	视觉
计算值	依照指定的百分比或者“auto”或者为绝对长度，如果不设定则为“auto”

块级元素内容框的宽度，如图8-1所示。

8.3.2 行内元素的宽度

块级元素生成块框（block box），行内元素生成行内框（inline box）。非替换的行内元素（如<a>和等）的宽度是其内容经过浏览器解释后实际的宽度，而不能通过设定width属性为非替换的行内元素指定宽度，例如下列代码，其显示如图8-8所示。

```
.sample1 strong { width: 200px; }
<p class="sample1">对strong元素设定<strong>width:100px</strong>无效</p>
```

对strong元素设定width:200px无效

图8-8 width属性对非替换的行内元素无效

而替换的行内元素具有“内在尺寸（intrinsic dimensions）”，即尺寸是由元素自身决定的，而不会受周围环境影响（例如图片文件有其自己的宽度和高度）。CSS并不规定元素的内在尺寸范围，在CSS 2.1中假定，只有替换元素才具有内在尺寸。对于替换元素（如<img>和<input>等）width属性是有效的，例如下列代码，其显示如图8-9所示。

```
.sample1 img { width: 200px; }
<p class="sample1">对img元素设定<img src="../../img/ddcat_anim.gif" alt="示例图片"title="示例图片"
/>width:200px有效</p>
```

此效果类似于为<img>标签设定width属性：

```
<img src="../../img/ddcat_anim.gif" width="200" alt="示例图片" title="示例图片" />
```

如果同时定义<img>标签的width属性，和CSS的width属性，根据层叠的原理，CSS的属性将胜出，例如下列代码，其显示如图8-10所示。

```
.sample1 img { width: 200px; }
<p class="sample1">img有width属性width=&quot;88&quot; <img src="../../img/ddcat_anim.gif "width="88" alt="示例图
片" title="示例图片" /></p>
```

由图8-9和图8-10还可以发现，定义图片的宽度时，如果不定义图片的高度，则图片会等比例放大（或者缩小）。

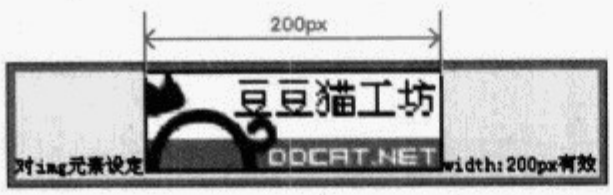


图8-9 width属性对替换的行内元素有效

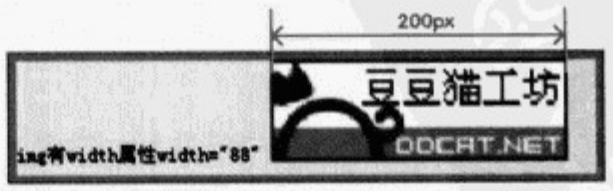


图8-10 CSS的width属性特殊性高于<img>标签的width属性

8.3.3 长度和百分比

长度值可以是具体的像素值，也可以为em值或者其他合法的值，例如右边代码：

```
p { width: 200px; }
div { width: 25em; }
```

在CSS 2中，宽度的百分比值的基数为元素包含块的宽度，而CSS 1中则是其父元素的内容框宽度。例如下列代码：

```
#width2 {
background : #6C3;
border : 1px solid #060;
width : 400px;
padding : 0 20px;
position : relative;
}
#wrap1 {
width : 300px;
border : 0;
margin : 0;
padding : 0 20px;
background : #FC3;
}
<div id="width2">
  <div id="wrap1">
    <p class="sample3">这个p不是绝对定位</p>
    <p class="sample4">这个p是绝对定位</p>
  </div>
</div>
```

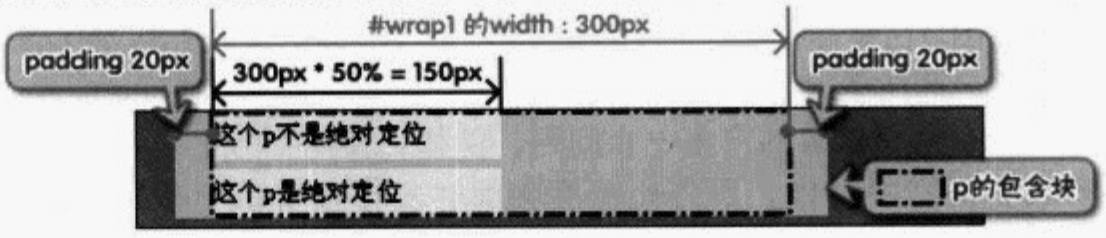


图8-11　未定位的元素的百分比宽度

此时若定义CSS如下，其显示如图8-11所示。

```
#wrap1 p { width:50%; }
```

根据［8.2.2 包含块］一节中元素包含块的生成原理可知，“wrap1”内的<p>元素未绝对定位，因此<p>元素的包含块由其最近的块级祖先元素的内容框创建，即“wrap1”的内容框。“wrap1”的宽度为300px，因此<p>的宽度为150px。此时，如果增加“sample4”的定位如右：

```
.sample4 {
position : absolute;
background : #3CF;
left : 20px;
top : 30px;
}
```

则对于“sample4”来说，其包含块就不是“wrap1”，而是“width2”，因为“width2”设定了“position : relative”，因此其显示如图8-12所示。

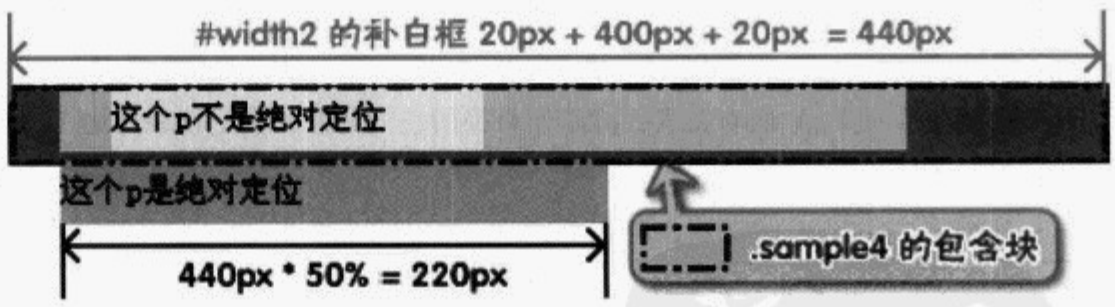
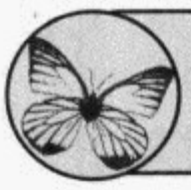


图8-12　绝对定位的元素的百分比宽度



提示：对于IE 6.0还是以CSS 1的规范来理解百分比宽度，即百分比基数为父元素的内容框宽度。而IE 7.0已经支持CSS 2规范。

如果元素内容的宽度大于元素本身的宽度，将发生溢出，而对于溢出内容的处理（如隐藏或者产生滚动条），可以通过overflow属性来设定。关于元素的定位，可以参见本书［9.3 定位］一节。

8.4 最大宽度（max-width）和最小宽度（min-width）

最大宽度（max-width）和最小宽度（min-width）是CSS 2增加的属性，可以为元素设定一个宽度范围。在弹性布局中，经常会使用百分比和em作为宽度，以保证元素间可以按一定的比例缩放，例如用户选择使用大的字号，而元素的补白、宽度、行高等也会相应改变以保持版面的可读性。

但是这样做容易产生一个问题，例如，使用百分比来设定元素的宽度的时候，当窗口变得很窄或者很宽的时候，百分比宽度会同时变得很窄或者很宽，因此，可以使用最大宽度（max-width）和最小宽度（min-width）来为其限定一个范围。

max-width属性具体定义列表如下：

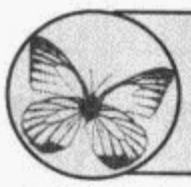
语法	max-width: <长度> | <百分比> | none | inherit
说明	设定元素的最大宽度
值	长度：合法的长度值，不可为负值。
	百分比：百分比基数为元素包含块的宽度，不可为负值。
	none：没有限制。
初始值	none
继承性	不继承
适用于	所有元素，除了非替换的行内元素、表格行和行组（row groups）
媒体	视觉
计算值	依照指定的百分比或绝对长度

最小宽度（min-width）具体定义列表如下：

语法	min-width: <长度> | <百分比> | inherit
说明	设定元素的最小宽度
值	长度：合法的长度值，不可为负值。
	百分比：百分比基数为元素包含块的宽度，不可为负值
初始值	0
继承性	不继承
适用于	所有元素，除了非替换的行内元素、表格行和行组（row groups）
媒体	视觉
计算值	依照指定的百分比或绝对长度或者为“none”

浏览器首先会计算元素的宽度，如果宽度超过最大宽度的值，则按最大宽度显示；如果宽度小于最小宽度的值，则按最小宽度值显示；最大宽度和最小宽度不一定同时定义。

如果max-width属性的值小于min-width属性的值，将会被自动转设为min-width属性的值。例如下列代码，其显示如图8-13所示。



提示：本节示例代码请参见下载文件包内［/第2部分/第8章：框模型/max_min_width.html］文件。

```
#width1 {
font-size : small;
width : 400px;
```

```
}
p {
width : 85%;
margin : 5px 0;
padding:0;
}
.sample1 {
max-width : 300px;
}
.sample2 {
width : 200px;
min-width : 250px;
}
.sample3 {
max-width : 200px;
min-width : 300px;
}
<div id="width1">
  <p>width: 80% = 400px * 85% = 340px</p>
  <p class="sample1">max-width: 300px;</p>
  <p class="sample2">width:200px; min-width:250px</p>
  <p class="sample3">max-width:200px; min-width:300px</p>
</div>
```

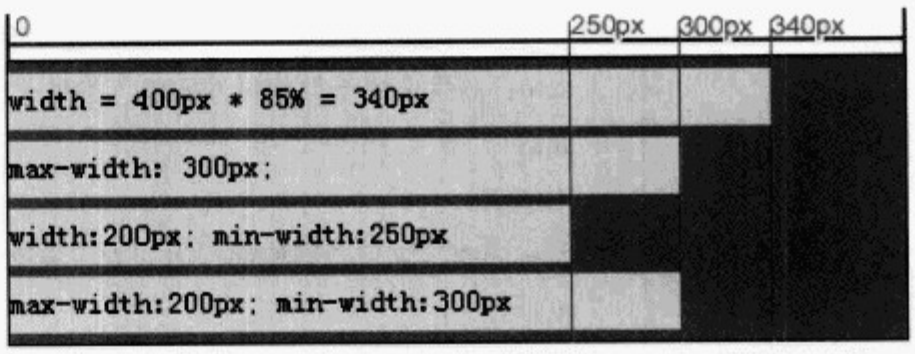


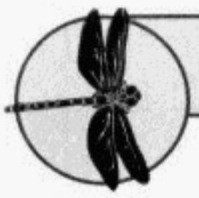
图8-13　最大宽度和最小宽度

由图8-13可以发现：

● 虽然<p>的宽度为85%，但是“sample1”的最大宽度为300px，因此“sample1”显示宽度为300px；

● “sample2”的宽度为200px，但是同时设定了最小宽度250px，因此“sample2”显示宽度为250px；

● “sample3”的最大宽度值小于最小宽度值，最大宽度值自动转设为最小宽度值，因此“sample3”的最大宽度为300px。



注意： IE 6.0不支持最大宽度和最小宽度，IE 7.0支持这2个属性。

8.5 高度：height属性

高度（height）用以设定块类元素和替换元素生成的内容框的高度。本小节示例，读者可以参见下载文件包内［/第2部分/第8章：框模型/height.html］文件。

8.5.1 语法

height属性具体定义列表如下：

语法	height : <长度> | <百分比> | auto | inherit
说明	设定块级元素和替换元素生成的内容框的高度
值	长度：合法的长度值，不可为负值。 百分比：百分比基数为元素包含块的高度，不可为负值。如果包含块的高度没有显式给出（即取决于内容的高度），该值等同于“auto”。

续表

	auto：高度取决于其他属性的值，参见［8.10.3 块级元素的垂直格式化］一节
初始值	auto
继承性	不继承
适用于	所有元素，除了非替换的行内元素、表格行和行组（row groups）
媒体	视觉
计算值	百分比或者同“auto”或者为绝对长度，如果未设定则为“auto”

块级元素的高度很好理解，即其内容框的高度。

8.5.2 行内元素的高度

height属性对于非替换的行内元素无效，非替换行内元素的框高度是由行高决定的，例如下列代码，继承了<p>的行高30px，因此其显示如图8-14所示。

```
.sample1 { line-height : 30px; }
.sample1 strong { height : 50px; }
<p class="sample1">段落行高30px，strong设定<strong> height:50px 无效</strong></p>
```

而对于替换元素（如<img>和<input>等），同width属性类似，height属性是有效的，例如下列代码，其显示如图8-15所示。

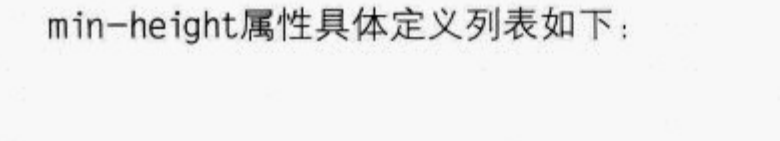


图8-14 非替换行内元素height属性无效

图8-15 替换的行内元素height属性有效

```
.sample1 img { height : 50px; }
<p class="sample1">对img元素设定<img src="../../img/ddcat_anim.gif" alt="示例图片"title="示例图片"
/>height:50px有效</p>
```

同样CSS的height属性特殊性高于<img>标签的height属性，定义图片的高度时，如果不定义图片的宽度，则图片会等比例放大（或者缩小）。

8.6 最大高度（max-height）和最小高度（min-height）

同样也可以为元素设定最大高度（max-height）和最小高度（min-height）。max-height属性具体定义列表如下：

语法	max-height: <长度> | <百分比> | none | inherit
说明	设定元素的最大高度
值	长度：合法的长度值，不可为负值。 百分比：百分比基数为元素包含块的高度，不可为负值。 none：没有限制
初始值	none
继承性	不继承
适用于	所有元素，除了非替换的行内元素、表格行和行组（row groups）
媒体	视觉
计算值	百分比同指定值或者为绝对长度值

min-height属性具体定义列表如下：

语法	min- height: <长度> | <百分比> | inherit
说明	设定元素的最小高度
值	长度：合法的长度值，不可为负值。 百分比：百分比基数为元素包含块的高度，不可为负值。
初始值	0
继承性	不继承
适用于	所有元素，除了非替换的行内元素、表格行和行组（row groups）
媒体	视觉
计算值	依照指定的百分比或绝对长度或者为“none”

浏览器首先会计算元素的高度，如果宽度超过最大高度的值，则按最大高度显示；如果高度小于最小高度的值，则按最小高度值显示；最大高度和最小高度不一定同时定义。

如果max-height属性的值小于min-height属性的值，将会被自动转设为min-height属性的值。例如下列代码，其显示如图8-16所示。

```
div {
height : 200px;
……
}
p {
height : 80%;
……
}
.sample1 {
max-height : 140px;
}
.sample2 {
height: 80px;
min-height: 100px;
}
.sample3 {
max-height: 80px;
min-height: 140px;
}
<div id="height1">
 <p>height
 200px*80%=160px</p>
 <p class="sample1">max-height:140px;</p>
 <p class="sample2">height:80px;
min-height:100px;</p>
 <p class="sample3">max-height:80px;
min-height:140px;</p>
</div>
```

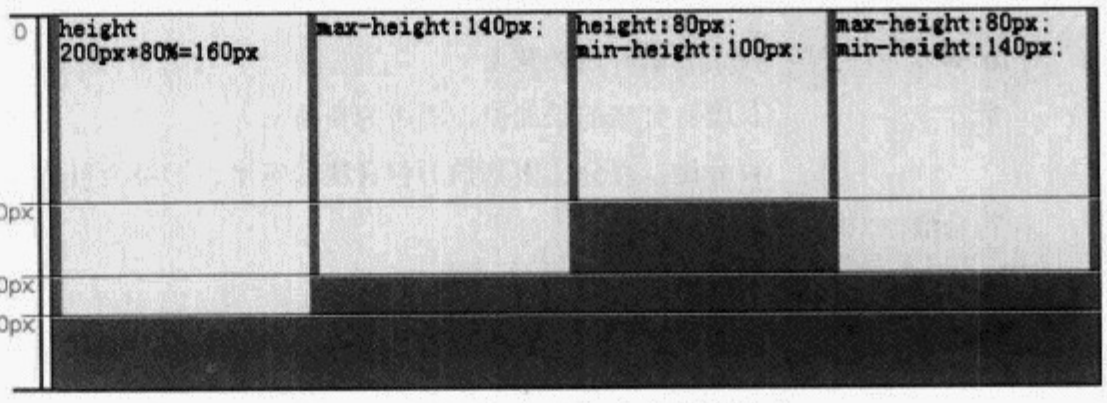


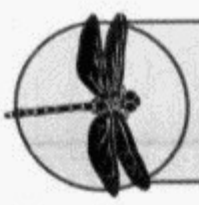
图8-16　最大高度和最小高度的设定

由图8-16可以发现：

● 虽然<p>的高度为80%（200px*80% =160px），但是“sample1”的最大高度为140px，因此“sample1”显示高度为140px；

● “sample2”的高度为80px，但是同时设定了最小高度100px，因此“sample2”显示宽度为100px；

● “sample3”的最大高度值小于最小高度值，最大高度值自动转设为最小高度值，因此“sample3”的最大高度为140px。



注意： IE 6.0不支持最大高度和最小高度，IE 7.0支持这2个属性。本节示例代码可以参见下载文件包内［/第2部分/第8章：框模型/max_min_height.html］文件。

8.7 补白：padding属性

padding属性也译为填充、内边距、内补丁等，是内容框与边框的内边缘之间的距离，补白部分显示的是背景。

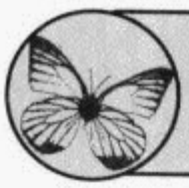
补白又分为4个方向的属性——上、右、下、左，而padding属性则是这4个方向属性的缩写。4个方向padding-top（上补白）、padding-right（右补白）、padding-bottom（下补白）和padding-left（左补白）的具体定义列表如下：

语法	padding-top: <长度> | <百分比> | inherit
	padding- right: <长度> | <百分比> | inherit
	padding- bottom: <长度> | <百分比> | inherit
	padding- left: <长度> | <百分比> | inherit
说明	设定元素补白的宽度
值	长度：合法的长度值，不可为负值。
	百分比：百分比值基数为包含块的宽度，不可为负值
初始值	0
继承性	不继承
适用于	所有元素，除了表格行组（table-row-group）、表格头部组（table-header-group）、表格注脚组（table-footer-group）、表格行（table-row）、表格列组（table-column-group）和表列（table-column）
媒体	视觉
计算值	依照指定的百分比或绝对长度

长度单位可以是px，也可以是em、in等，左边的定义是正确的，补白不允许有负值，因此右边的写法是错误的：

```
p { padding-left: 50px; }
div { padding-top: 3em; }
em { padding-bottom: 1in; }
```

```
p { padding-left: -50px; }
div { padding-top: -4%; }
em { padding-bottom: -2em; }
```



提示：本小节样例代码可以参见下载文件包内［/第2部分/第8章：框模型/padding.html］文件。

8.7.1 缩写属性：padding

在本书［3.3.5 缩写］一节中介绍过CSS的缩写属性，对于像边距和补白这样有4个边的属性，其4个边的值可以按“上、右、下、左”的顺序缩写在一行内，即使用缩写属性padding，其具体定义列表如下：

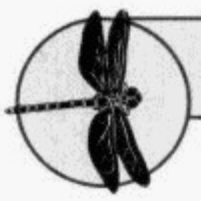
语法	padding: <补白宽度>{1,4} | inherit
说明	设定元素补白的宽度
值	<补白宽度>{1,4}：4个方向的宽度值，可为1~4个值
初始值	视各属性而定
继承性	不继承

续表

适用于	所有元素，除了表格行组（table-row-group）、表格头部组（table-header-group）、表格注脚组（table-footer-group）、表格行（table-row）、表格列组（table-column-group）和表列（table-column）
媒体	视觉
计算值	视各具体属性

同时，对于4个方向的值，还可进一步缩写，例如：

```
body { padding: 2em }              /* 4个方向的补白全部为 2em */
div { padding: 1em 2em }           /* 上 = 下 = 1em，右 = 左 = 2em */
p { padding: 1em 2em 3em }         /* 上 = 1em，下 = 3em，右 = 左 = 2em */
li { padding: 1em 2em 3em 4em }    /* 上 = 1em，右 = 2em，下 = 3em，左 = 4em */
```



注意：如果“上=下”但是“左≠右”，则不可以缩写。

由于padding属性的初始值为0，因此右边的2条规则是等价的：

```
p { padding-left : 30px; }
p { padding : 0 0 0 30px; }
```

但是，同时使用4个方向的补白和padding缩写属性，浏览器将按照层叠的原则来取舍。例如左边的规则按照层叠的原则，最后等价于右边的规则：

```
p { padding : 10px; }
p { padding-right : 20px; }
div { padding-right : 20px; padding : 10px; }
```

```
p { padding : 10px 20px 10px 10px; }
div { padding : 10px; }
```

8.7.2 补白、宽度和高度

如图8-1所示，补白是边框内边到内容边的距离，补白不包含在width属性或height属性之内。同时，行内元素的补白有效。例如下列代码，其显示如图8-17所示。

```
#padding1 .sample1 {
padding : 20px;
width : 330px;
height : 30px;
border:10px solid #999;
}
#padding1 p span {
padding : 10px;
background : #FC3;
}
<div id="padding1">
 <p class="sample1">p { width:330px; height:30px; padding:20px;}</p>
 <p>普通p内的<span>行内元素span的补白</span></p>
</div>
```

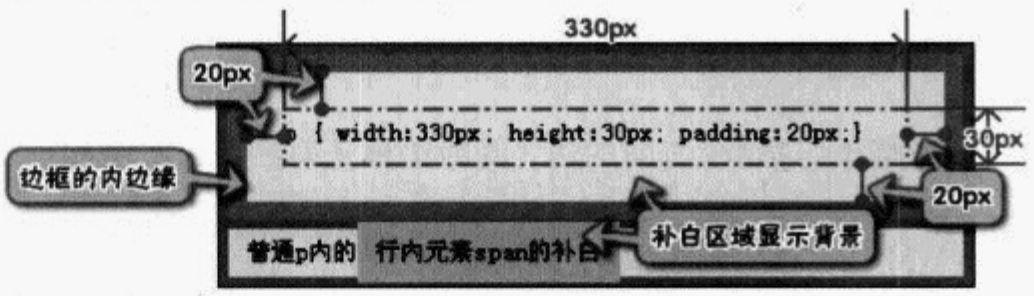


图8-17 补白与宽度和高度的关系

由图8-17可以发现，补白框内显示出背景颜色。而对于行内元素，补白不会影响到行高，因此元素的垂直方向的背景色会在行框外显示。

8.7.3 百分比值补白

补白的百分比值基数为**包含块的宽度**，不可为负值。无论是上下补白还是左右补白，都以包含块的宽度为基数，而与包含块的高度无关。例如下列代码，其显示如图8-18所示。

```
#padding2 {
width : 400px;
height : 150px;
……
}
#padding2 p {
padding : 5%;
……
}
#padding2 em {
display: block; /* 为了显示出p的内容区域 */
background:#CF9;
}
#padding2 .sample2 {
position : relative; /* 生成span的包含块 */
width : 300px;
height : 30px;
padding : 20px;
}
#padding2 .sample2 span {
display : block;
background : #FC3;
position : absolute;
padding : 5%;
top : 0;
left : 0;
}
<div id="padding2">
 <p><em>普通p，父元素宽度400px，补白为 400px*5% = 20px</em></p>
   <p class=" sample2" ><em>p相对定位，宽300px，高30px，补白20px<span>绝对定位的span，padding:
5%;</span></em></p>
</div>
```

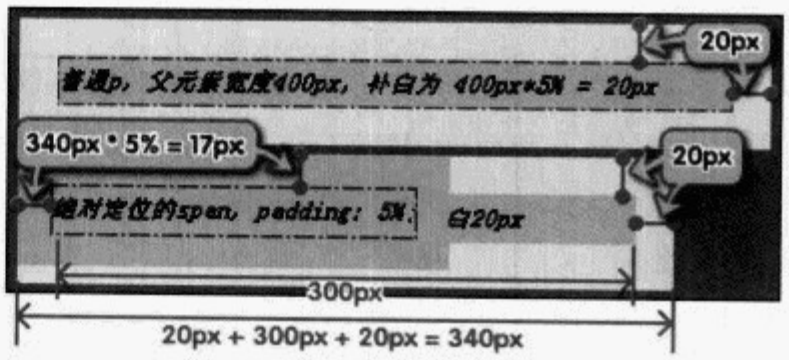


图8-18 补白的百分比宽度

由图8-18可以发现，绝对定位的元素的包含块“sample2”虽然设定了高度为30px，而其的上下补白仍按照其包含块的宽度为基数来计算。

8.8 边框：border属性

边框也包括4个方向的边，同时边框还有颜色、宽度和样式3个属性，因此定义稍微复杂一些。

8.8.1 边框颜色

边框颜色包括border-top-color（上边框颜色）、border-right-color（右边框颜色）、border-bottom-color（下边框颜色）、border-left-color（左边框颜色）和border-color（缩写属性）。

4个方向的边框颜色具体定义列表如下：

语法	border-top-color: <颜色> | transparent | inherit
	border-right-color: <颜色> | transparent | inherit
	border-bottom-color: <颜色> | transparent | inherit
	border-left-color: <颜色> | transparent | inherit
说明	设定元素边框的颜色
值	颜色：符合规定的颜色值。
	transparent：边框是透明的，但是有宽度

续表

初始值	color属性的值
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	如果取color属性的值，则为color属性的计算值，否则同指定值

如果不定义边框的颜色，边框将显示为color属性（前景色）定义的颜色。关于color属性，请参见本书［10.2 前景色：color属性］一节。

transparent值表示边框是透明的，但是有宽度，会显示出父元素的背景，例如下列代码，其显示如图8-19所示。

```
.color1{
border-color : transparent;
border-width : 5px;
border-style : solid;
}
.color1 em {
background : #FC6;
display : block;
}
<div id="borderColor">
 <h3>border-color［边框颜色］</h3>
 <p>没有边框的p</p>
 <p class="color1"><em>边框颜色为透明，5px宽。</em></p>
</div>
```

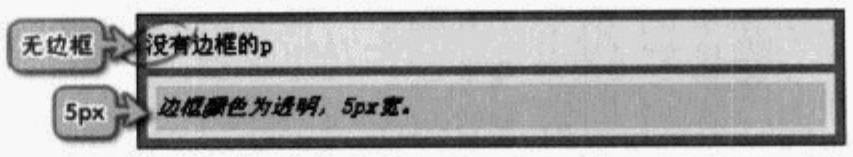


图8-19　透明边框的显示

IE 6.0及更早的版本不支持透明边框，会使用前景色显示边框颜色，如图8-20所示。

图8-20　IE 6.0及更早的版本中透明边框的显示

border-color属性为4个方向边框颜色的缩写属性，其具体定义列表如下：

语法	border-color: <颜色>{1,4} | inherit
说明	设定元素边框的颜色
值	<边距宽度>{1,4}：4个边的颜色，可为1~4个值
初始值	视各属性而定
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	视各具体属性

border-color属性的值也可以缩写，例如：

```
body { border-color : red; }        /* 4个方向的边框全部为红色 */
div { border-color : red blue; }    /* 上 = 下 = red，右 = 左 = blue */
p { border-color : red blue green; }  /* 上 = red，下 = green，右 = 左 = blue */
li { border-color : red blue green yellow; } /* 上 = red，右 = blue，下 = green，左 = yellow */
```

8.8.2 边框宽度

边框宽度包括：border-top-width、border-right-width、border-bottom-width、border-left-width及缩写属性border-width，其具体定义列表如下：

语法	border-top-width: thin | medium | thick | <长度> | inherit
	border-right-width: thin | medium | thick | <长度> | inherit

续表

	border-bottom-width: thin | medium | thick | <长度> | inherit
	border-left-width: thin | medium | thick | <长度> | inherit
说明	设定元素边框的宽度
值	thin：一个窄的边框。
	medium： 一个中等的边框。
	thick：一个厚边框。
	长度：长度值，不可为负值
初始值	0
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	绝对长度值；如果为“none”和“hidden”，则值为0

关键字值的解释取决于用户端，不过遵循如下关系：thin <= medium <= thick。

而border-width属性为4个方向边框宽度的缩写属性，其具体定义列表如下：

语法	border-width: <边距宽度>{1,4} | inherit
说明	设定元素边框的宽度
值	<边距宽度>{1,4}：4个边的宽度值，可为1~4个值
初始值	视各属性而定
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	视各具体属性

border-width属性的值也可以缩写，例如：

```
body { border-width : 2em; }          /* 4个方向的边框宽度全部为 2em */
div { border-width : 1em 2em; }       /* 上 = 下 = 1em，右 = 左 = 2em */
p { border-width : 1em 2em 3em; }     /* 上 = 1em，下 = 3em，右 = 左 = 2em */
li { border-width : 1em 2em 3em 4em; } /* 上 = 1em，右 = 2em，下 = 3em，左 = 4em */
```

8.8.3 边框样式

只定义边框的宽度并不会使元素显示边框，因为边框样式的默认设定为“none（无）”，边框样式规定了边框的线型（实线、双线、点线等），而且也包括4个方向的属性border-top-style、border-right-style、border-bottom-style、border-left-style和缩写属性border-style。

其具体定义列表如下：

语法	border-top- style: none | hidden | <关键字> | inherit
	border-right- style: none | hidden | <关键字> | inherit
	border-bottom- style: none | hidden | <关键字> | inherit
	border-left- style: none | hidden | <关键字> | inherit
说明	设定元素边框的线型
值	none：没有边框。该值迫使border-width的计算值为0。
	hidden：和none相似，除非在表格元素中解决边框冲突时。
	线型的关键字包括：

续表

值	dotted：边框是一系列的点。 dashed：边框是一系列的短线条的段。 solid：边框是一条单一的线。 double：边框有两条实线。两条线宽和其中的空白的宽度之和等于border-width的值。 groove：边框看上去好像是雕刻在画布之内。 ridge：和grove相反：边框看上去好像是从画布中凸出来。 inset：该边框使整个框看上去好像是嵌在画布中。 outset：和inset相反，该边框使整个框看上去好像是从画布中凸出来
初始值	none
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	同指定值

属性border-style具体定义列表如下：

语法	border-style: <边框线型>{1,4} | inherit
说明	设定元素边框的线型
值	<边框线型>{1,4}：4个边的线型，可为1~4个值
初始值	视各属性而定
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	视各具体属性

border-style属性的值也可以缩写，例如：

```
body { border-style : dotted; }              /* 4个方向的边框全部为dotted */
div { border-style : solid dotted ;}         /* 上 = 下 = solid，右 = 左 = dotted */
p { border-style : solid dotted dashed ;}    /* 上 = solid，下 = dashed，右 = 左 = dotted */
li { border-style : solid dotted dashed outset; } /* 上 = solid，右 = dotted，下 = dashed，左 = outset */
```

浏览器不同，边框样式的显示也有所不同，而且边框在背景之上，如图8-21所示。

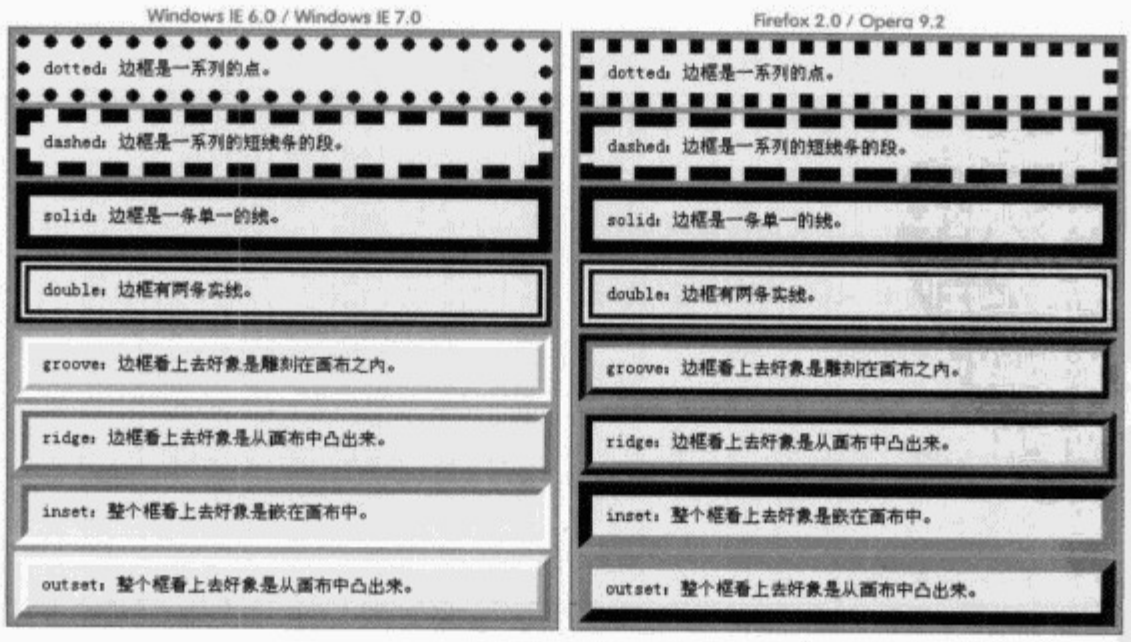


图8-21　不同浏览器边框样式的不同表现形式

8.8.4 不同方向的边框属性缩写

在前面3个小节内介绍了边框的3个属性：颜色、宽度和样式，而边框分为“上、右、下、左”4个方向，这4个方向上还各有一种缩写属性，包括border-top、border-right、border-bottom和border-left。同前面的缩写属性不同，这4个属性的参数值包含的是颜色、宽度和样式这3个属性的值，具体定义列表如下：

语法	border-top: [<宽度> || <样式> || <颜色>] | inherit
	border-right: [<宽度> || <样式> || <颜色>] | inherit
	border-bottom: [<宽度> || <样式> || <颜色>] | inherit
	border-left: [<宽度> || <样式> || <颜色>] | inherit
说明	设定元素的边框颜色、宽度和样式
值	宽度：边框的宽度。
	样式：border-style属性允许的样式。
	颜色：border-color属性允许的值。
初始值	视各属性而定
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	视各具体属性

宽度、样式和颜色的书写顺序没有要求，例如下列规则是等价的：

```
p { border-top : 4px solid #ccc; }
p { border-top : #ccc 4px solid; }
p { border-top : 4px #ccc solid; }
```

如果省略颜色值，则使用color属性的值为边框颜色，例如右边代码：

```
p { border-top : 4px solid ; }
```

而如果省略边框样式，则不显示边框，因为边框样式的默认值是“none”。例如右边代码：

```
p { border-top : 4px #ccc; }
```

8.8.5 缩写属性border

边框除了上面所介绍的各个属性以外，还有一个总的缩写属性：border，其具体定义列表如下：

语法	border: [<宽度> || <样式> || <颜色>] | inherit
说明	设定元素的边框颜色、宽度和样式
值	宽度：边框的宽度。
	样式：border-style属性允许的样式。
	颜色：border-color属性允许的颜色
初始值	视各属性而定
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	视各具体属性

border属性定义的是全部4个方向的边框属性，宽度、样式和颜色的书写顺序没有要求，如果省略颜色值，则使用color属性的值为边框颜色，而如果省略边框样式，则不显示边框。

也可以使用border属性取消边框的定义，例如浏览器默认会给带链接的图片添加一个边框，而如果不希望出现这个边框，可以如左边代码定义，或者如右边代码定义：

```
img { border : none; }
```

```
img { border : 0; }
```

8.8.6 行内元素的边框

行内的元素可以有4个方向的边框，例如下列代码，其显示如图8-22所示。

```
#border2 p strong {
border : 2px solid #F90;
line-height : 4em;
}
#border2 p span {
border : 2px solid #F90;
padding : 5px;
margin : 5px;
}
#border2 p img {
border : 2px dashed #39F;
}
<div id="border2">
  <p><strong>strong元素</strong>，<span>span元素有补白</span>，<img src="../../img/ddcat_96.jpg" align="图片" title="示例图片" width="96" height="96" /></p>
</div>
```

图8-22　行内元素的边框

由图8-22可以发现，元素虽然设定了行高4em，但是其边框仍然紧邻内容区域，这表示边框到内容区域的距离同元素的行高无关，而是围绕在元素补白的外面，因此要改变行内元素边框与内容的距离需要增加补白。

8.8.7 应用：文字链接的装饰

为了提示访问者，往往会为页面内的带链接的文字设定特别的样式，来同普通文字区分出来，最常用的方法是设定特殊的颜色与下划线，而利用边框，可以为链接文字添加比较有趣的效果。例如下列代码，其显示如图8-23所示。

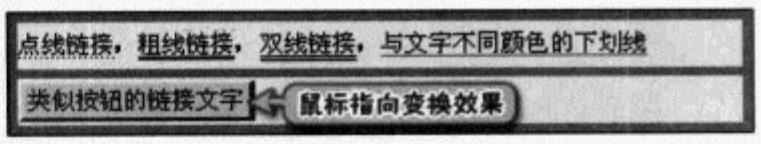


图8-23　利用边框生成不同的文字链接效果

```
#border3 a { text-decoration : none; } /* 去除文字链接默认的下划线 */
.link1 { border-bottom : 1px dotted; }
.link2 { border-bottom : 2px solid; }
.link3 { border-bottom : 3px double; }
.link4 { border-bottom : 1px solid #f00; }
.link5 {
padding : 3px;
background : #ccc;
border : 2px outset #ccc;
}
.link5:hover { border-style : inset; } /* 鼠标指向时的效果 */
<div id="border3">
  <h3>文字链接的边框</h3>
   <p><a class="link1" href="#" title="链接边框样式示例">点线链接</a>，<a class="link2" href="#" title="链接边框样式示例">粗线链接</a>，<a class="link3" href="#" title="链接边框样式示例">双线链接</a>，<a class="link4" href="#" title="链接边框样式示例">与文字不同颜色的下划线</a></p>
   <p><a class="link5" href="#" title="链接边框样式示例">类似按钮的链接文字</a></p>
</div>
```

8.9 边距：margin属性

边距（margin），可译为“边白”、“外边距”、“外补丁”等，是边框外的透明区域（如果父元素有背景，则会显示出来父元素的背景），如图8-1所示。边距一般可以用来设定本元素与其他元素之间的距离。在网页排版布局中，边距是经常用到的属性。

边距也有“上、右、下、左”4个方向，而margin属性则是这4个方向属性的缩写。本小节样例代码可以参见下载文件包内［/第2部分/第8章：框模型/margin.html］文件。

margin属性具体定义列表如下：

语法	margin：<边距宽度>{1,4} | inherit
说明	设定元素边距的宽度
值	<边距宽度>{1,4}：4个边的宽度值，可为1~4个值
初始值	视各属性而定
继承性	不继承
适用于	所有元素，表格类元素中只包含表格标题（table-caption）、表格（table）和行内表格（inline-table）
媒体	视觉
计算值	视各具体属性

同padding属性一样，margin属性的值也可以缩写，例如：

```
body { margin: 2em }          /* 4个方向的边距全部为 2em */
div { margin: 1em 2em }       /* 上和下 = 1em，右和左 = 2em */
p { margin: 1em 2em 3em }     /* 上=1em，右=2em，下=3em，左=2em */
li { margin: 1em 2em 3em 4em } /* 上=1em，右=2em，下=3em，左=4em */
```

对于表格类元素，只有表格标题（table-caption）、表格（table）和行内表格（inline-table），而单元格（table-cell）、表格行（table-row）等margin属性无效。例如下列代码，其显示如图8-24所示。

```
#table1 {
background: #06C;
margin : 20px;
width : 360px;
border-collapse : collapse;
}
#table1 th,
#table1 td {
background : #FF9;
margin : 20px;
border-bottom:1px solid #F90;
}
#table1 caption {
background : #FC6;
margin : 10px;
}
#table1 thead {
margin : 10px;
}
<table cellspacing="0" id="table1">
 <caption>
  表格标题margin: 10px;
 </caption>
 <thead>
```

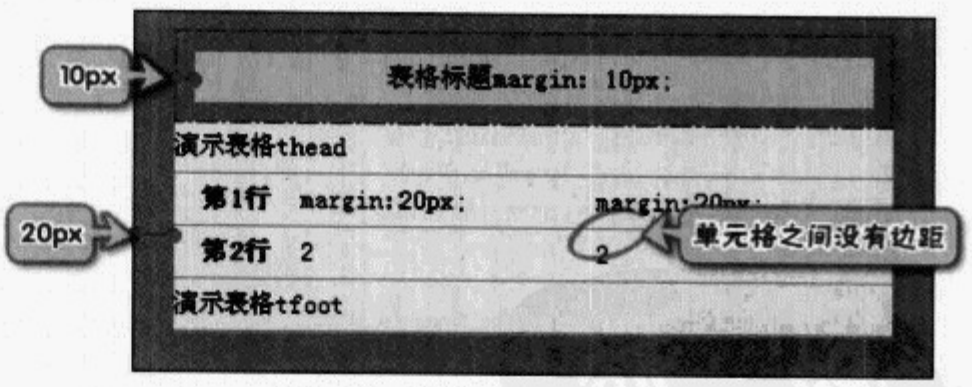


图8-24 表格中各元素的边距

```
<tr>
 <td colspan="3">演示表格thead</td>
</tr>
</thead>
<tr>
 <th scope="row">第1行</th>
 <td>margin:20px;</td>
 <td>margin:20px;</td>
</tr>
<tr>
 <th scope="row">第2行</th>
 <td>2</td>
 <td>2</td>
</tr>
<tfoot>
<tr>
 <td colspan="3">演示表格tfoot</td>
</tr>
</tfoot>
</table>
```

由图8-24可以发现，虽然在CSS里对单元格<td>设定了margin属性，但是实际并没有显示边距。单元格之间的距离是由border-collapse和border-spacing属性来控制的。

IE不支持表格标题<caption>的边距。关于表格的格式化内容，请参见本书［第11章：表格］。对于元素在水平和垂直方向上的边距，视觉格式化模型稍有不同，在下面将分开讲解。

8.9.1 水平方向的边距：margin-left属性和margin-right属性

水平方向margin-left属性和margin-right属性的具体定义列表如下：

语法	margin-left : <长度> | <百分比> | auto | inherit margin-right : <长度> | <百分比> | auto | inherit
说明	设定元素边距的宽度
值	长度：长度值，可为负值。 百分比：百分比值基数为包含块的宽度，可为负值。 auto：根据规定的几种情况表现不同，请参见［8.10 常规流向中的视觉格化］一节
初始值	0
继承性	不继承
适用于	所有元素，表格类元素中只包含表格标题（table-caption）、表格（table）和行内表格（inline-table）
媒体	视觉
计算值	依照指定的百分比或绝对长度

水平方向的边距不会发生重叠，例如下列代码，其显示如图8-25所示。

```
.sample1 img {
margin-left: 30px;
margin-right: 55px;
}
<div id="margin1">
    <p class="sample1"><img src="../../img/ddcat_anim.gif" alt="示例图片" title="示例图片"width="88" height="31" /><img src="../../img/ddcat_anim.gif" alt="示例图片" title="示例图片"width="88" height="31" />文字</p>
</div>
```

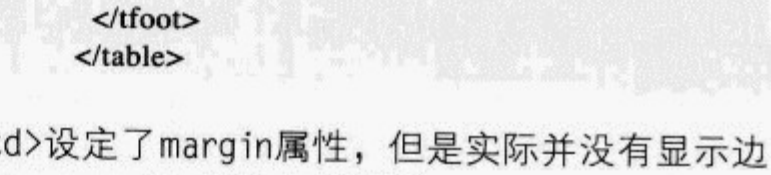


图8-25 水平方向的边距

元素的边距为本元素的边框边到父元素的内容边的距离，例如如下代码，其显示如图8-26所示。

```
.sample2 {
padding:0 20px; /* 左右20px补白 */
margin:10px;
}
.sample2 img{
margin-left: 10px;
margin-right: 20px;
```

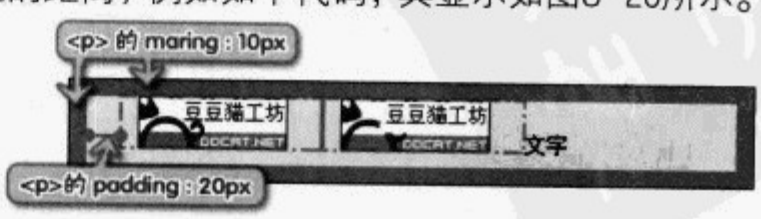
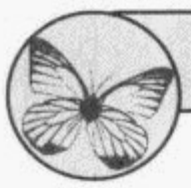


图8-26 元素的边距与父元素的内容边的关系

```
}
<div id="margin1">
    <p class="sample2"><img src="../../img/ddcat_anim.gif" alt="示例图片" title="示例图片" width="88" height="31" /><img src="../../img/ddcat_anim.gif" alt="示例图片" title="示例图片" width="88" height="31" />文字</p>
</div>
```



提示：边距的“auto”值，将在［8.10 常规流向中的视觉格式化］一节内介绍。

8.9.2 垂直方向的边距：margin-top属性和margin-bottom属性

垂直方向的边距margin-top属性和margin-bottom属性的具体定义列表如下：

语法	margin-top : <长度> | <百分比> | auto | inherit
	margin-bottom : <长度> | <百分比> | auto | inherit
说明	设定元素边距的宽度
值	长度：长度值，可为负值。
	百分比：百分比值基数为包含块的宽度，可为负值。
	auto：根据规定的几种情况表现不同，请参见［8.10 常规流向中的视觉格化］一节
初始值	0
继承性	不继承
适用于	所有元素，表格类元素中只包含表格标题（table-caption）、表格（table）和行内表格（inline-table）
媒体	视觉
计算值	依照指定的百分比或绝对长度

垂直方向与水平方向的边距不同的是行内元素和边距的重叠。

1. 行内元素

垂直方向的边距对于非替换的行内元素无效（例如<a>、）。这些行内元素垂直方向的距离由行高决定，而不是元素自身的边距。例如下列代码，其显示如图8-27所示。

```
.sample3 { line-height: 30px; }
.sample3 strong { margin-top: 50px; }
<p class="sample3"><strong>strong, margin-top:50px;</strong>p行高为30px;</p>
```

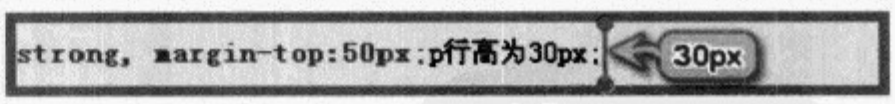


图8-27 非替换的行内元素垂直方向边距无效

替换元素的上下边距有效，例如下列代码，其显示如图8-28所示。

```
.sample3 { line-height: 30px; }
.sample3 img { margin: 10px 0;}
<p class="sample3"><img src="../../img/ddcat_anim.gif" alt="示例图片" title="示例图片"width="88" height="31" />文字x</p>
```

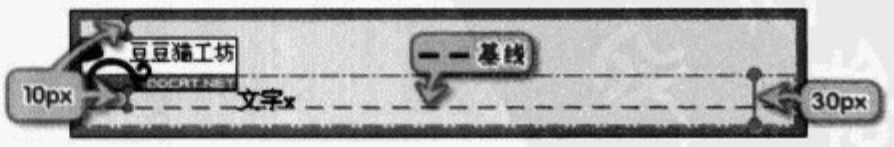


图8-28 替换元素的上下边距有效

由于默认的垂直对齐方式为基线对齐，因此图片的下边距为图片的下边沿距本行基线的距离。

2. 边距的重叠

元素间垂直方向的边距有可能会发生重叠，分为以下几种情况。

（1）常规流向的块级元素。

一个文档中的元素，可以看作很多漂浮在河（文档流）上的盒子，这些盒子顺序排列并各自占据一定的空间（占位），如果拿走其中的某个盒子（元素脱离文档流），那么后面的盒子就会顺序向前占据空出来的空间。

“浮动”或者“绝对定位”的元素会脱离文档流，而CSS中的“常规流向（normal flow）”指的是没有浮动或者绝对定位的元素的情况。常规流向中，两个或多个相邻的块框垂直边距会发生重叠，其结果是相邻边距宽度中较大的值。

例如下列代码，其显示如图8-29所示。

```
#margin2 p {
margin-top: 10px;
margin-bottom: 20px;
}
<div id="margin2">
  <p>margin-top : 10px; margin-bottom: 20px;</p>
  <p>margin-top : 10px; margin-bottom: 20px;</p>
</div>
```

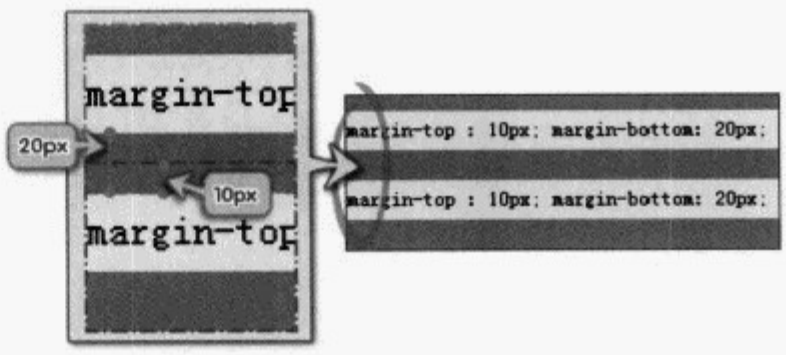
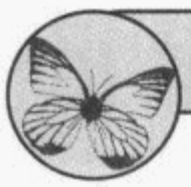


图8-29　垂直方向的边距会重叠

相邻元素不限于兄弟元素，也包括父子。例如下列代码，其显示如图8-30所示。



提示：读者可以参见下载文件包内［/第2部分/第8章：框模型/margin2.html］文件。

```
* { /* 清除浏览器样式 */
margin : 0;
padding : 0;
}
body {
background : #6c6;
font : 12px/20px "宋体", serif;
}
div {
background : #360;
margin : 10px 0;
}
p {
background : #FF9;
margin : 20px;
}
span {
display : block; /* 将span定义为块级元素 */
background : #FC3;
margin : 30px 0;
}
<div>
  <p><span>相邻元素不限于兄弟，</span>也包括后代。</p>
  <p>多个相邻的元素的垂直边距<span>都会发生重叠</span></p>
</div>
<div>下一个div</div>
```

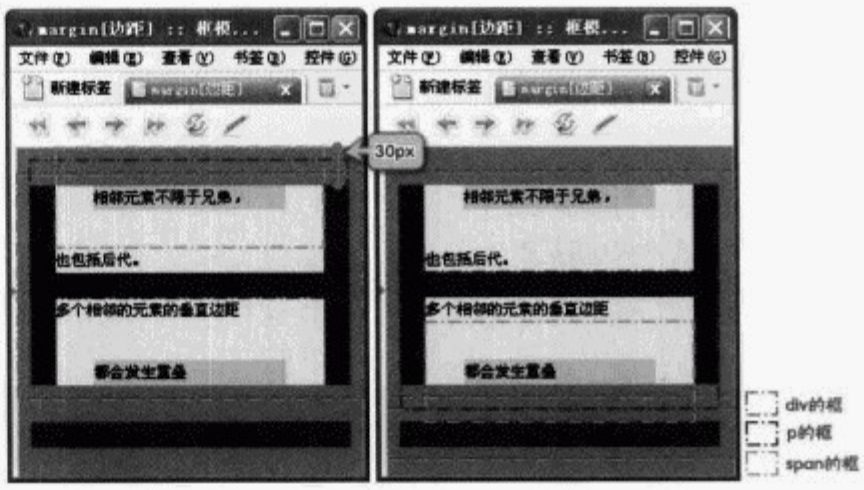


图8-30　多个相邻元素之间的垂直边距重叠

由图8-30可以发现，相邻的 、<p>和之间的垂直边距发生重叠，最终边距值为他们中边距最大的值30px。

而元素与其后代元素垂直边距的重叠只限于元素本身没有文字内容、边框及补白的情况，例如上例中，如果为 设定1px的边框（或者补白）则 的垂直边距与其相邻的后代的边距不重叠，如图8-31所示。

（2）浮动（float）元素。

浮动元素与其他元素框的垂直边距不重叠，包括浮动元素和其后代之间。例如下列代码其显

示如图8-32所示。

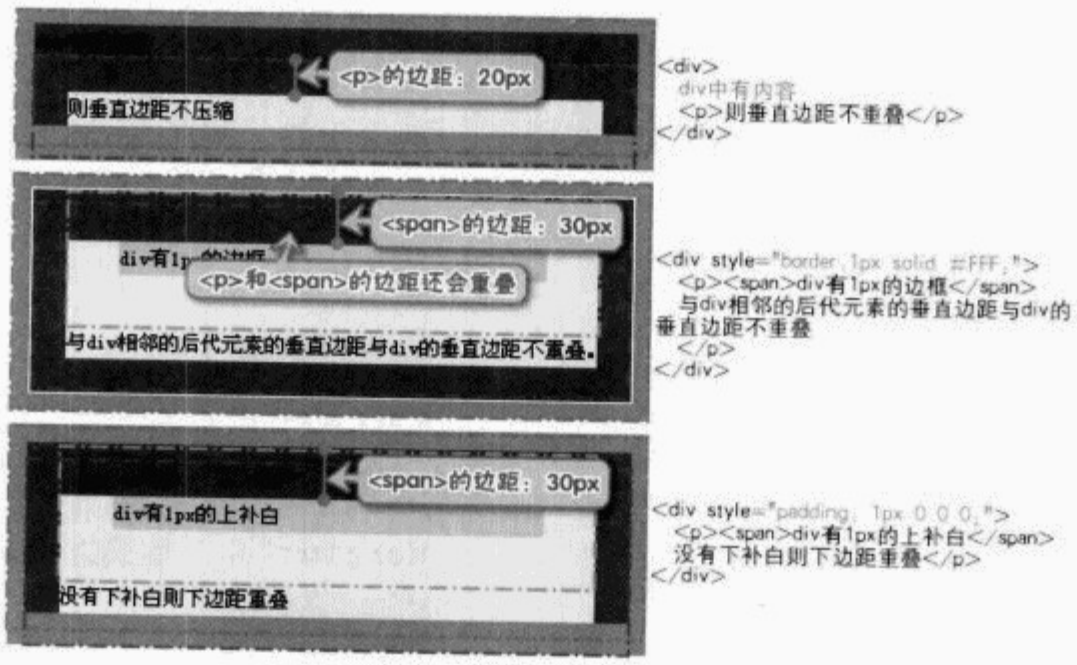


图8-31 有边框或者补白的元素与其后代元素之间的垂直边距不重叠

```
div {
background : #360;
margin : 10px;
}
p {
background:#FF9;
margin : 20px;
}
#margin1 {
float : left;
}
<div>普通的div</div>
<div id="margin1">
 <p>浮动的div内的p，垂直边距不重叠。</p>
</div>
```

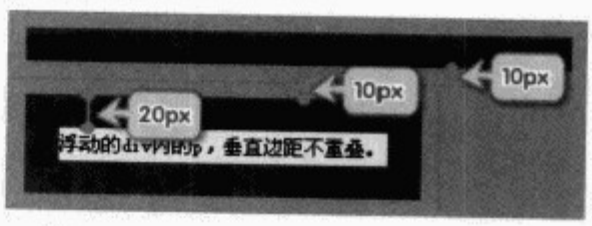
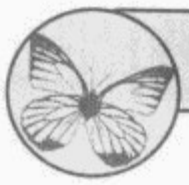


图8-32 浮动元素的垂直边距不重叠



提示：关于float属性，请参见本书［9.4.1 设定浮动：float属性］一节。

（3）overflow属性。

如果元素的overflow属性值不是“visible”，其后代元素垂直边距不重叠。例如下列代码，其显示如图8-33所示。

```
#margin2 {
height : 70px;
overflow : auto;
margin: 10px;
……
}
p {
background:#FF9;
margin:20px;
}
<div id="margin2">
 <p>div的overflow属性为auto</p>
 ……
</div>
```

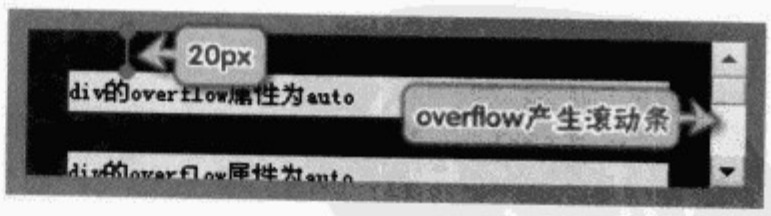
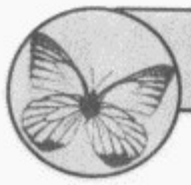


图8-33 overflow属性与边距的重叠



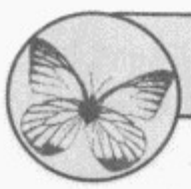
提示：关于overflow属性，请参见本书［9.6.1 溢出：overflow属性］一节。

（4）绝对定位。

绝对定位的框的边距不重叠，包括该元素和其后代之间。例如下列代码，其显示如图8-34所示。

```
#margin3 {
position : relative; /* 生成包含块 */
height:70px;
}
p {
background:#FF9;
margin : 15px;
}
#margin3 p {
position : absolute;
}
#margin3 p span {
margin : 10px;
……
}
<div id="margin3">
 <p>绝对定位的p，垂直边距不重叠</p>
</div>
```

图8-34　绝对定位的元素垂直边距不重叠



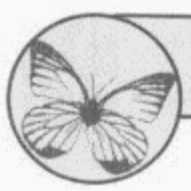
提示：关于绝对定位，请参见本书［9.3 定位］一节。

（5）行内块元素：inline-block属性。

display属性设定为“inline-block”的元素边距不重叠，包括该元素和其后代之间。例如下列代码，其显示如图8-35所示。

```
#inlineBlock {
margin: 10px;
……
}
#inlineBlock p {
display: inline-block; /* inline-block元素具有块级元素特性，同时可在一行内显示*/
margin:20px;
……
}
<div id="inlineBlock">
 <p>inline-block元素的边距不重叠</p>
 <p>inline-block元素的边距不重叠</p>
</div>
```

图8-35　inline-block元素的垂直边距不重叠



提示：关于display属性，请参见本书［9.2 显示类型：display属性］一节。

（6）根元素和其后代之间的边距不重叠。

8.9.3 百分比值边距

同补白的百分比值类似，边距的百分比值的基数也是基于包含块的宽度，例如下列代码，其显示如图8-36所示。

```
#margin3 {
width : 400px;
……
}
#margin3 p {
margin : 5%;
……
}
#margin3 .sample4 {
position : relative;
width : 300px;
height : 30px;
padding : 20px;
}
#margin3 .sample4 span {
position : absolute;
margin : 5%;
display : block;
background : #FC3;
top : 0;
left : 0;
}
<div id="margin3">
  <p>普通p，父元素宽度400px，边距为400px*5% = 20px</p>
    <p class=" sample4" >p相对定位，宽300px，高30px，补白各20px<span>绝对定位的span，margin:
5%;</span></p>
  </div>
```

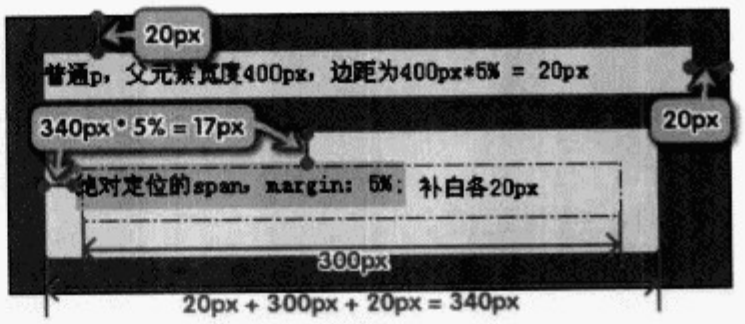


图8-36 百分比边距值

由图8-36可以发现，百分比值的边距与元素的包含块的高度无关，只与宽度有关，“sample4”虽然设定了高度为30px，而其子元素的上下边距仍按照其包含块的宽度为基数来计算。

8.9.4 负值边距

与补白不同，边距允许负值的存在，例如下列代码其显示如图8-37所示。

```
#margin4 {
line-height : 30px;
width : 400px;
……
}
#margin4 .sample5 {
margin : 0 -20px;
background : #FC3;
}
#margin4 .sample6 {
width : 80%;
margin :-20px 0 0 0;
background : #FC3;
}
#margin4 .sample7 {
width : 80%;
margin-bottom : -20px;
background : #FC3;
}
<div id="margin4">
 <p>普通p</p>
 <p class="sample5">p，左右边距 -20px</p>
 <p>普通p</p>
 <p class="sample6">p，上边距 -20px</p>
 <p>普通p</p>
 <p class="sample7">p，下边距 -20px</p>
</div>
```

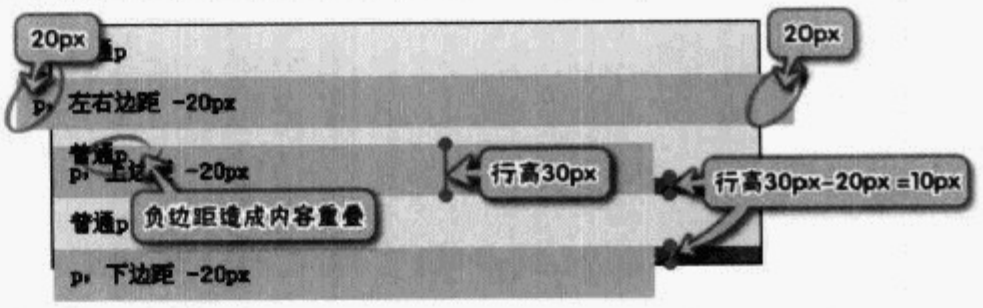


图8-37 负值边距的显示

由图8-37可以发现，负值边距使<p>元素的框超出了其父元素的框，有时还会造成元素间内

容的重叠。有些浏览器对于负值边距的处理可能会出现问题，如图8-38所示。

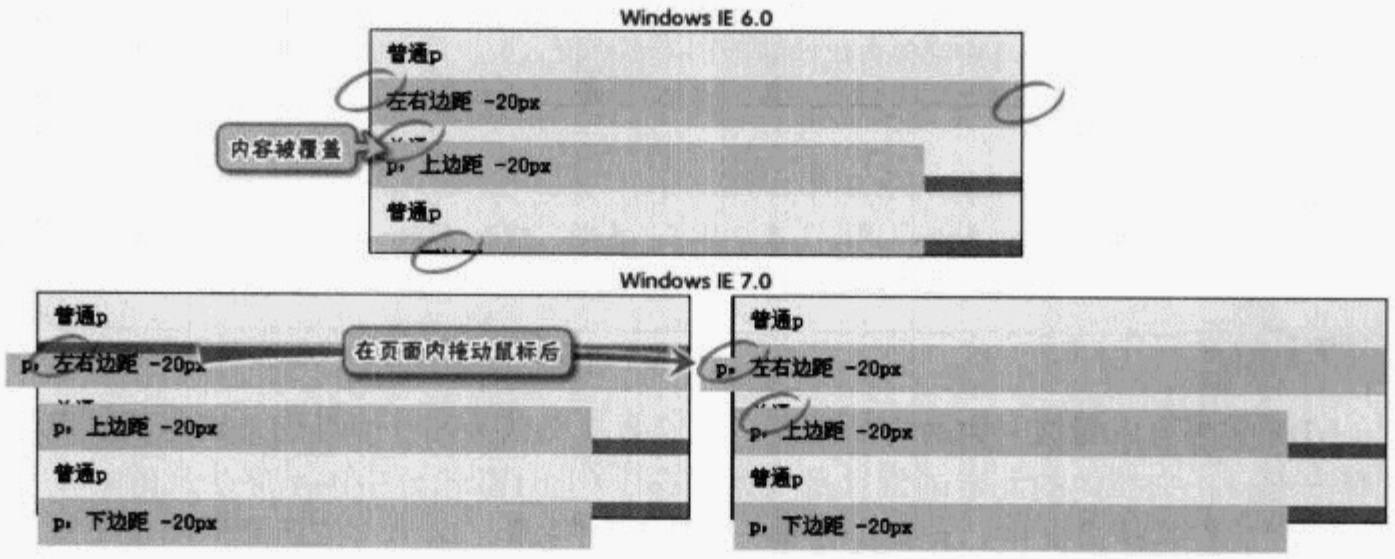
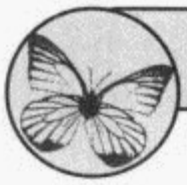


图8-38　浏览器对于负值边距显示的问题



提示：关于IE的显示问题，可以参见本书［16.2.1 hasLayout属性］一节。

垂直方向的负边距也会重叠，在最大的正边距中减去绝对值最大的负边距。如果没有正边距，则从零中减去绝对值最大的负边距。例如下列代码其显示如图8-39所示。

```
.sample9 { margin-bottom : 20px; }
.sample10 { margin-top : -10px; }
<p class="sample9">下边距20px;</p>
<p class="sample10">上边距 -10px</p>
```

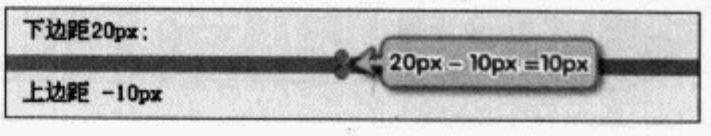


图8-39　垂直方向负值边距的重叠

由图8-39可以发现，2个<p>元素之间的距离不是20px，而是10px，同理也可以解释图8-37中负边距的段落“sample6”与其上面的<p>元素的内容互相重叠的现象。

8.9.5　应用：元素水平居中

除了在本书［7.1.4　应用：整体居中］介绍的使页面整体水平居中的方法以外，还可以利用负边距实现页面水平居中的布局，代码如下，其显示如图8-40所示。读者也可以参见下载文件包内［/第2部分/第8章：框模型/margin_ sample.html］文件。

```
body {
padding : 0 0 0 50%;
margin : 0;
}
#wrap {
width : 400px;
height : 200px;
background : #FC6;
 margin-left : -200px;  /* 元素宽度的一半 */
}
<body>
<div id="wrap">
 内容内容
</div>
</body>
```

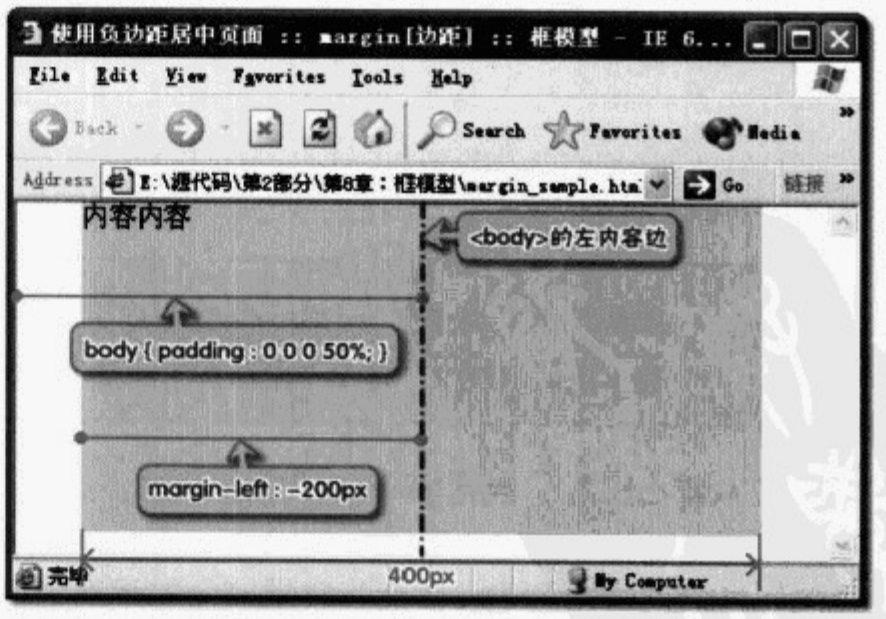
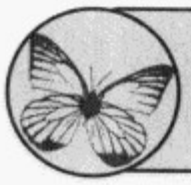


图8-40　负边距实现页面居中

由图8-40可以发现，<body>的左内容边在视口宽度的50%处，而“wrap”层宽度400px，同时左边距为-200px，因此显示的时候，“wrap”层向左200px，实现了“wrap”层在视口内的居中。



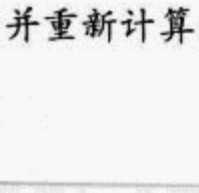
提示： 负值边距可以使水平方向居中，而不能实现垂直方向居中，这是为什么？因为垂直方向的百分比边距基数是包含块的宽度而不是高度。

8.10 常规流向中的视觉格式化

（X）HTML文档可以看成一条数据的河流，浏览器从文档头部开始直到文档尾部结束，读取一部分解释一部分，文档内的元素的位置是根据其在（X）HTML代码中的顺序生成的，元素具有高度和宽度，因此在常规流向中，元素会占据一定的空间位置，这个占位还会影响位于其后面的元素的位置。而浮动和绝对定位会改变元素的显示顺序及位置，产生层叠等效果。

视觉格式化模型则指导浏览器如何生成元素框及如何显示这些框。

在本书的［3.4.2 显示元素］一节中曾经简要介绍过块级（block-level）元素和行内（inline-level）元素。一个元素框的类型，部分地影响它在视觉格式化模型中的表现，display属性决定了框的类型。本节介绍常规流向中未定位的的元素的视觉格式化。



提示： 常规流向的视觉格式化内容还包括“相对定位”的内容，此部分内容将在下一章详细介绍。

8.10.1 块级元素的水平格式化

在常规流向中，块框一个接一个地垂直放置，无论这个框的宽度是否占满了一行，它后面的元素总是要在新行内显示，两个兄弟框之间的垂直距离取决于margin属性，如图8-41所示。在块格式化内容中相邻的框的垂直边距会发生重叠。

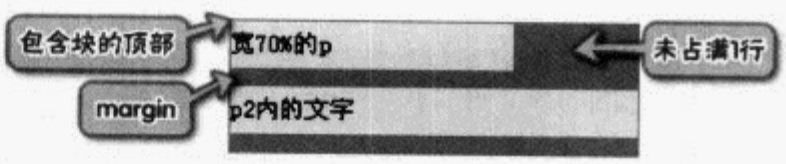
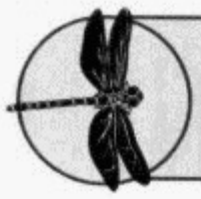


图8-41 块级元素的格式化

在块格式化内容中，起点是一个包含块的顶部，每一个框左边距边与包含块的左内容边相接触（对于从右到左的格式化，右边距边接触右内容边），即使存在浮动也是如此。

一个框包括2个方向：水平方向和垂直方向，而这2个方向的格式化是不同的。对于水平方向的块级元素，其各相关属性的使用值遵循以下的原则：

margin-left + border-left-width + padding-left + width + padding-right + border-right-width + margin-right + 滚动条的宽度（如果存在）= 包含块的宽度



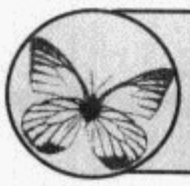
注意： 浏览器根据overflow属性生成滚动条，滚动条的宽度和浏览器及操作系统等有关。在下面的示例中，将不考虑存在滚动条的情况。

在本章［8.2.2 包含块］一节内介绍过：如果元素的定位（position）为“relative”或者“static”，它的包含块由它最近的块级元素、单元格（table cell）或者行内块（inline-block）祖先元素的内容框创建。因此，常规流向的元素的包含块的宽度就是其父元素的内容框宽度。

水平格式化有时候很简单，有时候又比较复杂。

● 如果上述所有的属性指定了非“auto”的值，这些值被称为是“过度约束”，其中之一的计算值将不得不和它的指定值不同：如果direction属性为“ltr”，margin-right的指定值被忽略，并重新计算以满足上面的等式；如果direction属性为“rtl”，对margin-left采取上述的方法。

- 如果只有一个值指定为“auto”，它的计算值从等式中得出。
- 如果width设定为“auto”，则其他的“auto”值成为0，而width从等式的剩余部分得到。
- 如果margin-left和margin-right为“auto”，它们的计算值相同。



提示： 本小节示例，读者可参见下载文件包内［/第2部分/第8章：框模型/block_box.html］文件。

1. 定义了width属性的元素

通过width属性可以定义元素的内容的宽度，例如有如下代码，其显示如图8-42所示。

```
#blockFormatting2 {
width:400px;
}
#blockFormatting2 p {
width : 300px;  /* 设定p的宽度 */
margin: 0 0 5px;  /* 分隔两个<p>元素，使显示清晰 */
}
#blockFormatting2 .p2 {
padding : 0 20px;  /* 设定p的左右有20px的补白 */
margin : 0 10px;  /* 设定p的左右有10px的边距 */
}
#blockFormatting2 .p2 em {
background : #FC3;
display : block;  /* 使<em>变为块级元素以显示p元素的内容宽度（em会自动充满p的内容宽度）*/
}
<div id="blockFormatting2">
 <p>宽300px</p>
 <p class="p2"><em>左右padding:20px 左右margin:10px</em></p>
</div>
```

由图8-42可以发现，第2个<p>元素“p2”由于发生了过度约束，因此其右边距的设定值10px将被抛弃，而其实际的宽度根据等式计算得出：

右边距=400px(宽度)-10px(<p>左边距)-20px(<p>左补白)-300px(<p>宽度)-20px(<p>右补白) =50px

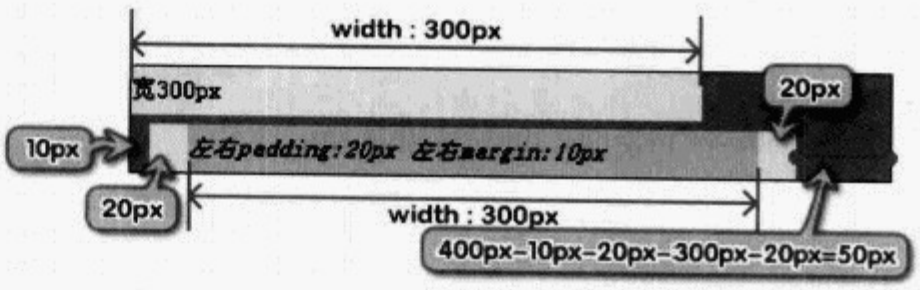
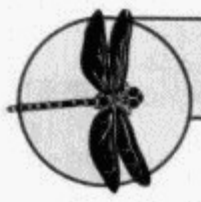


图8-42　定义了width属性的元素的显示宽度



注意： 如果设定了border属性，则宽度还要再加上border的值。

当设定了width值，则元素的框的宽度有可能会大于其父元素的内容宽度，此时将发生“溢出（overflow）”，即元素在父元素的内容框之外被描绘。例如有如下代码，其显示如图8-43所示。

```
#blockFormatting3 {
……
width: 300px;
}
#blockFormatting3 p {
background:#fc0;
width: 300px;
margin: 10px;
}
```

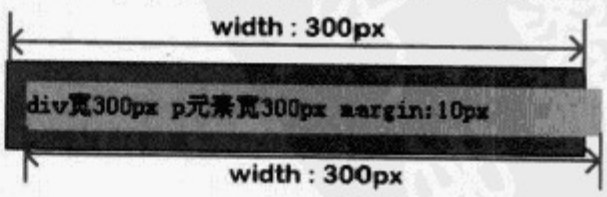


图8-43　内容的溢出

```
<div id="blockFormatting3">
 <p>div宽300px  p元素宽300px margin:10px</p>
</div>
```

由于发生了“过度约束”，因此margin-right的值被重新计算得出：

margin-right的值 = 300px(宽度)-10px (<p>左边距)-300px(<p>宽度) = -10px

但是对于IE 6.0及更早的版本，却不是这样处理，浏览器会自动扩展父元素以包含溢出的元素，如图8-44所示。

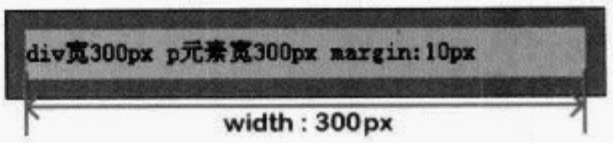
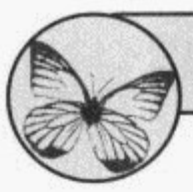


图8-44 IE 6.0 对内容的溢出的处理



提示：关于IE 6.0的问题，请参见本书［16.2.1.5 具有Layout的元素的表现］一节。

元素如果设定了最小宽度（min-width）和最大宽度（max-width），则会同时遵循这2个属性的设定，计算出宽度值后再计算等式中的其他值。

2. width属性的auto值

在本章［8.3 宽度］一节内介绍过，width属性的初始值是“auto”，其表现为：非浮动的块级元素按照常规方式顺序显示，它的框宽度总是会充满其父元素的内容框，因此width设定为“auto”时，其他的“auto”值为0。

因此当CSS中没有明确定义width属性时，元素的宽度总是尽量充满父元素的内容区域。例如有如下代码，其在浏览器内显示如图8-45所示。

```
.p1 {
margin: 10px;
border:5px solid #FC3;
}
<div style="width: 330px;">
  <p>div宽度为330px，未设定边距等的p1</p>
  <p class="p1">p2有10px的边距和5px的边框</p>
</div>
<div style="width: 400px;">
 <p>div宽度为400px，未设定边距等的p1</p>
 <p class="p1">p2有10px的边距和5px的边框</p>
</div>
```

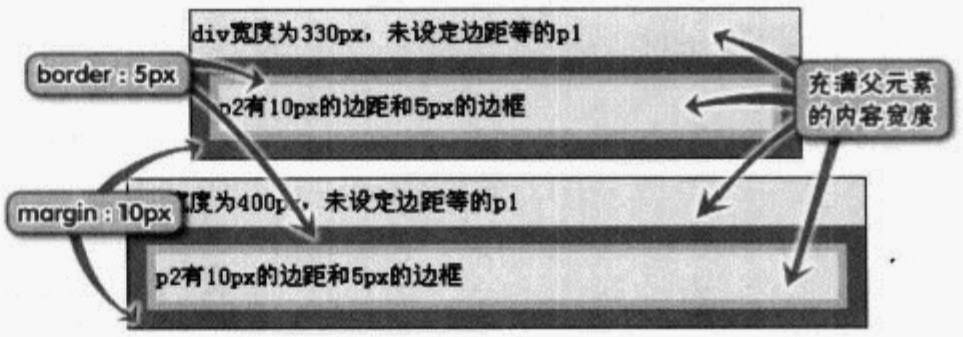


图8-45 未设定width属性的非浮动元素的显示

由图8-45可以发现，2个 内的<p>元素都充满了 的宽度，其中第2个<p>元素的内容宽度为父元素的内容宽度减去边距和边框的宽度。

如果对 增加padding属性，如下所示，则其显示如图8-46所示。

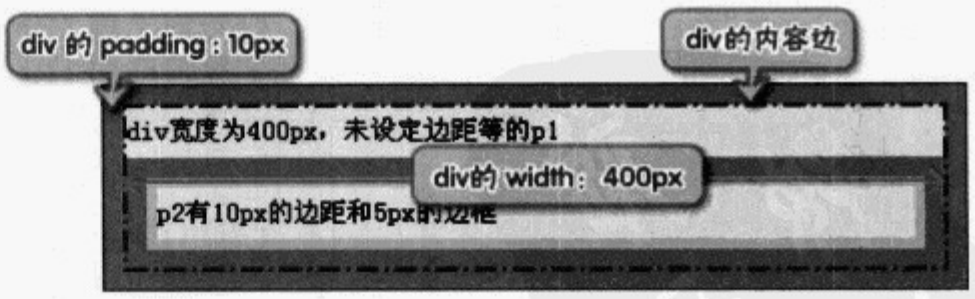


图8-46 元素的父元素的内容宽度区域

```
<div style="width:400px; padding:10px;">
 <p>div宽度为400px，未设定边距等的p1</p>
 <p class="p1">p2有10px的边距和5px的边框</p>
</div>
```

3. 左右边距的auto值

横向的7个属性中，除了width属性外，margin-left和margin-right这2个属性也可以定义值

为“auto”，余下的几个属性如果没有设定明确的值，则默认为“0”。

例如有如下代码，其显示如图8-47所示。

```
.p3,
.p4 {
margin: 0 20px 0 0;  /* margin的4个值的顺序为上右下左 */
width: 200px;
}
.p4 {
margin-left: auto;  /* 覆盖前面定义的0 */
}
<div>
 <p class="p3">左边距0,右边距20px</p>
 <p class="p4">左边距auto,右边距20px</p>
</div>
```

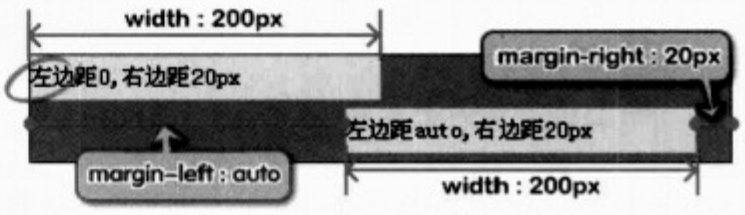
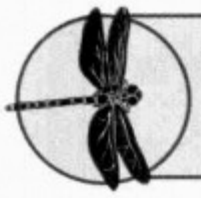


图8-47　设定auto值的margin属性的表现

由图8-47可以发现，设定了“margin-left: auto”的第2个段落整体偏向右侧，因为其左边距自动设定为父元素 的宽度与<p>的宽度和右边距之和的差，即：

左边距 = 父元素宽度-（段落宽度200px + 右边距20px）

<p>元素整体的框宽度还是等于父元素的内容宽度。而对于第一个段落，虽然左右边距都设定了具体的数值，但是由于其总宽度不等于父元素的内容宽度，因此浏览器强制设定其右边距为“auto”。



注意：此例中默认的文字方向（direction属性）为“从左至右”，而如果设定了文字方向为“从右至左”，则左边距会被强制设定为“auto”。

如果设定具体的边距值，而宽度设为“auto”，代码如下，其显示如图8-48所示。

```
.p5 { margin : 5px 20px; }
<div>
 <p class="p5">左右边距20px,宽度auto</p>
</div>
```

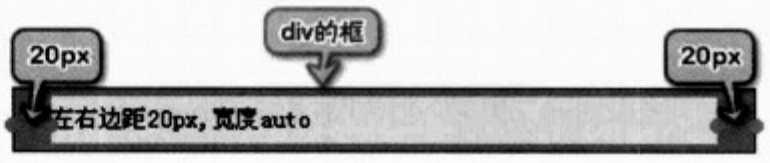


图8-48　宽度为“auto”的表现

由图8-48可以发现，宽度会自动填充父元素宽度减去左右边距的剩余部分，因为width的初始值是“auto”。

在本书［7.1.4　应用：整体居中］一节中介绍过使页面整体居中的方法，即设定“margin : auto”，但同时要设定元素的宽度，例如如下代码，其显示如图8-49所示。

```
.p7 {
width: 200px;
margin : 5px auto;
}
<div>
 <p class="p7">宽度200px，边距auto</p>
</div>
```

图8-49　左右边距为“auto”的表现

左右边距平分了 宽度与<p>宽度的差值，因此<p>元素居中显示。还有一种情况是，只设定1个边距，而另一个边距和宽度都为“auto”，例如下列代码，其显示如图8-50所示。

```
.p8 {
width : auto;
margin-left : auto;
margin-right : 20px;
}
<div>
 <p class="p8">宽度和左边距为auto，右边距20px</p>
</div>
```

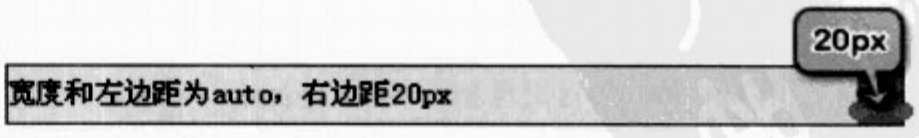


图8-50　只设定右边距，宽度和左边距为“auto”的表现

由图8-50可以发现，<p>元素的宽度会占满 剩余的宽度，而左边距为“0”。如果将这3个值都设定为“auto”，则边距会默认为“0”，而宽度为“100%”。代码如下，其显示如图8-51所示。

```
.p6 {
width: auto;
margin: auto;
}
<div>
 <p class="p6">宽度和边距都为 auto</p>
</div>
```

图8-51 3个属性全为“auto”的表现

由上面3个例子可以总结出：当width属性值为“auto”时，总是会尽量占据最大的宽度。

4. 边距的负值

在本章［8.9.4 负值边距］一节中介绍过，边距可以设定为负值，而左右有负值边距也遵循水平7个属性和的公式：

框宽度 = margin-left + border-left + padding-left + width + padding-right + border-right + margin-right

例如下列代码，其显示如图8-52所示。

```
# blockFormatting5 {
width : 400px;
border:1px solid #060;
}
.p9 {
margin-right : -20px;
.....
}
.p10 {
margin : 0 20px 0 -20px;
background : #FC0;
border : 5px solid #06c;
}
<div id="blockFormatting5">
 <p class="p9">左边距0,右边距-20px</p>
 <p class="p10">左边距-20px,右边距20px，边框5px</p>
</div>
```

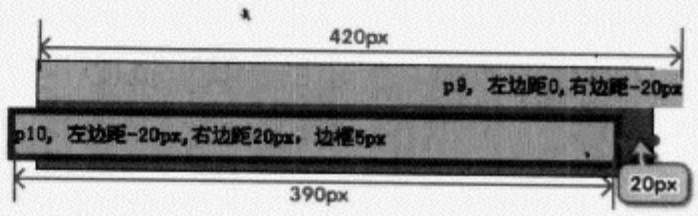


图8-52 负值边距与宽度的计算

“blockFormatting5”的宽度为400px，“p9”未定义width和margin-left属性，因此为“auto”，因此可得等式：

400px = auto(0) + 0 + 0 + auto + 0 + 0 + (-20px)

因此，“p9”的宽度为420px。而“p10”定义了左右边距和边框，未定义width属性，其等式为：

400px = -20px + 5px + 0 + auto + 0 + 5px + 20px

因此，“p10”的宽度为390px。

8.10.2 应用：宽度自适应的布局

利用width属性的auto值，可以实现元素宽度自适应其父元素的布局。例如有网页布局如图8-53所示。

XHTML结构如下：

```
<body>
 <div id="header">这里是头部</div>
 <div id="menu">这里是定宽的菜单</div>
   <div id="content">这里是自适应宽度的内容
</div>
 </body>
```

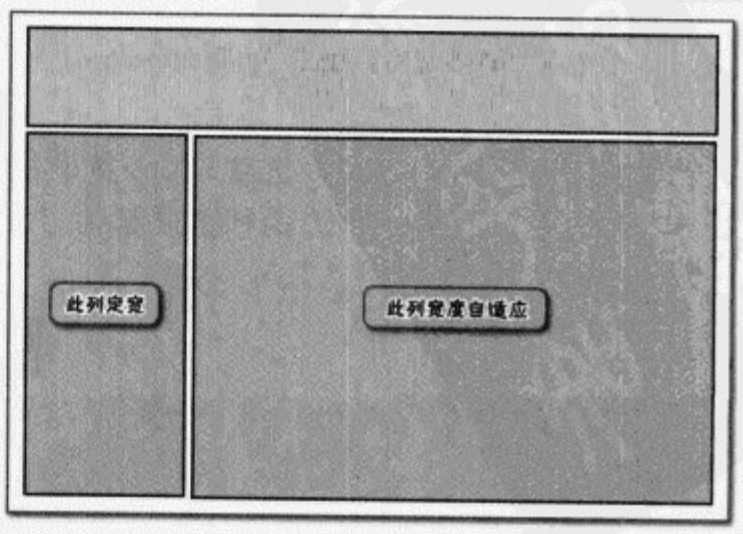
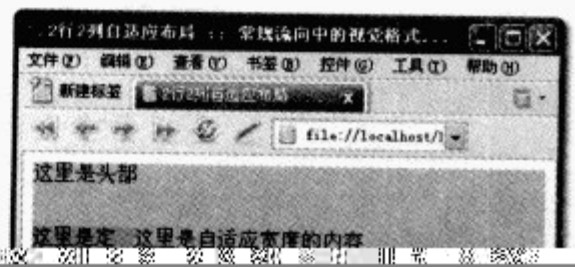


图8-53 常用的2列布局

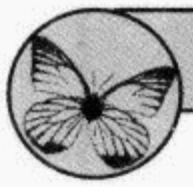
假设需要“menu”层宽度为20%，而“content”层自动占满屏幕剩下的宽度，则其CSS设定如下：

```
#header {
background : #fc9;
height : 50px;
}
#menu {
width : 20%;
background : #9cf;
```



出内容的处理（例如隐藏或者产生滚动条），可以通过overflow属性来控制。

图8-56 元素的高度小于内容高度的表现



提示：关于overflow属性，可以参见本书［9.6.1 溢出：overflow属性］一节。

当overflow属性值为“visible（初始值）”的时候，IE 6.0会自动扩展元素高度以包含内容，如图8-57所示。元素如果设定了最小高度（min-height）和最大高度（max-heigth），则会依照这2个属性的定义决定高度值。

图8-57 IE 6.0对于溢出内容的错误处理

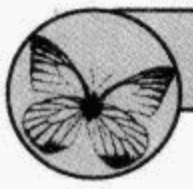
2. height属性与auto值

height属性的初始值为“auto”，与width属性不同，height属性为“auto”时，高度取决于该元素是否有任何块类子元素。如果它只包含行内子元素，高度等于最顶端的行框的顶到最底端的行框的底之间的距离。例如下列代码，其显示如图8-58所示。

```
#blockFormatting8 {
line-height : 30px;
……
}
#blockFormatting8 span {
line-height : 20px;
vertical-align : top;
}
#blockFormatting8 em {
line-height : 40px;
vertical-align : bottom;
}
<div id="blockFormatting8">div行高30px，<span>span元素的行高20px，顶端对齐</span>div行高30px，div行高30px，<em>em元素行高40px，底端对齐</em>div行高30px</div>
```

图8-58 只包含行内子元素的元素的高度

如果元素包含块类子元素，高度等于最顶端的块类子框的顶边界到最底端的块类子框的底边界，子框可以是一个匿名框。例如下列代码，其显示如图8-59所示。



提示：关于匿名块框，请参见本书［9.1.1 块框的生成（block box）］一节。

```
div { …… }
p { …… }
<div>
 <p>段落内的文字</p>
 div内的文字，将生成匿名块框
</div>
```

图8-59 包含块类子元素的元素的高度

元素的子元素中浮动框和绝对定位框被忽略，例如下列代码，其显示如图8-60所示。

```
.p11 {
float : left;
height : 40px;
background : #FC3;
}
<div id="blockFormatting10">
 <p class="p11">p11左浮动</p>
 <p>本段文字不浮动</p>
</div>
```

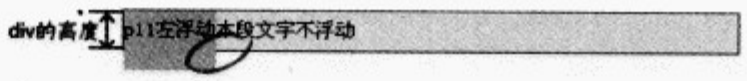


图8-60 含有浮动子元素的元素的高度

由图8-60可以发现， 的高度只包含了非浮动元素的高度，而浮动的“p11”的高度并不包含在内，因此如果元素内部只有浮动或者绝对定位的子元素，则元素高度为0。例如下列代码，其显示如图8-61所示。

```
.p11 {
float : left;
……
}
#blockFormatting11 {
……
}
.p12 {
position: absolute;
right: 20px;
……
}
<div id="blockFormatting11">
 <p class="p11">段落1左浮动</p>
 <p class="p12">段落2绝对定位</p>
</div>
```

图8-61 只包含浮动子元素的元素高度为0

因此，在使用浮动或着绝对定位的时候，要特别注意此点。对于相对定位的子元素，则只考虑其未偏移的位置，例如下列代码，其显示如图8-62所示。

```
.p13 {
position: relative;
height:40px;
top: 10px;
left: 20px;
}
<div id="blockFormatting12">
 <p class="p13">段落相对定位</p>
</div>
```

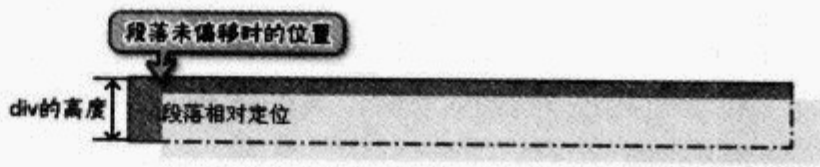
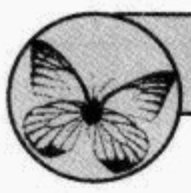


图8-62 包含相对定位的子元素的元素的高度



提示：关于定位和浮动，将在下一章详细介绍。

3. 上下边距的auto值

上下边距如果设定为“auto”，则其计算值为0。

4. 垂直方向的边距会发生重叠

详细内容可以参考本章［8.9.2.2 边距的重叠］一节。

8.10.4 应用：高度自适应浏览器窗口

百分比值的高度以包含块的高度为基数来计算，但是由于视口不为初始包含块提供高度而是让其适应文档内容，因此，当元素包含块的高度没有明确定义的时候，百分比值将等同于“auto”。

这是高度和宽度最重要的区别。

因此希望通过设定元素100%的高度来得到一个自适应浏览器窗口的布局对（例如右边代码）是达不到预期效果的：

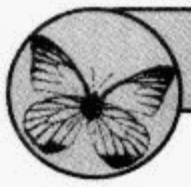
```
div { height : 100%; }
<body>
 <div>希望div高度自适应浏览器的大小</div>
<body>
```

初始包含块的高度可以由根元素的height属性指定，因此给根元素设定一个高度则达到为初始包含块设定高度的目的，如左边代码。但是 如果不设定绝对定位，其最近的块级祖先是<body>，因此同时要为<body>也设定高度，如右边代码。

```
html { height : 100%; } /* 根元素高度为视口的高度 */
```

```
html, body { height : 100%; }
```

此时应注意， 的框的实际高度还受边距、边框和补白的影响；<body>也遵守框模型的规定。因此如果设定 的高度为100%，同时设定 的上下边距，或者<body>有上下边距、补白、边框等属性，页面会产生滚动条。



提示：读者可以参见下载文件包内［/第2部分/第8章：框模型/height2.html］文件。

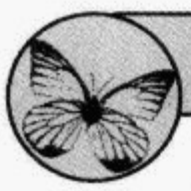
8.10.5 行内元素的格式化

在行内格式化内容中，行内框一个接一个地水平排列，起点是包含块的顶部。水平方向的行内框之间有边距、边框和补白，行内框的高度取决于元素的行高，垂直方向的边框和补白不会影响行内框的高度。

在同一行的行内元素处于同一个行框内，行框的宽度取决于包含块的宽度，高度取决于本行内所有行内元素中行内框最高的。行内框在行框内垂直方向的对齐可以有不同的方式：顶部对齐、底部对齐、基线对齐等。关于行框、行内框和行高，可以参见本书［7.3 行高（line-height）］一节。

1. 行内非替换元素

对于行内的非替换元素，其行内框可能会超过行框的宽度，长的行内框将分割成几个框，而这些框将会分布在几个行框中。如果一个行内框发生分割，边距、边框和补白属性不会应用在断开的地方。例如有如下代码，其显示如图8-63所示。



提示：读者可以参见下载文件包内［/第2部分/第8章：框模型/inline_box.html］文件。

```
p {
padding:0;
}
strong {
margin: 10px;
padding: 10px;
border:2px solid #39F;
}
#inlineFormatting {
line-height: 3em;
}
.sample1 {
line-height: 2em;
}
<div id="inlineFormatting">
 <p>段落内的文字，<strong>行内元素strong发生回行，水平方向的补白、边框和边距的表现</strong></p>
```

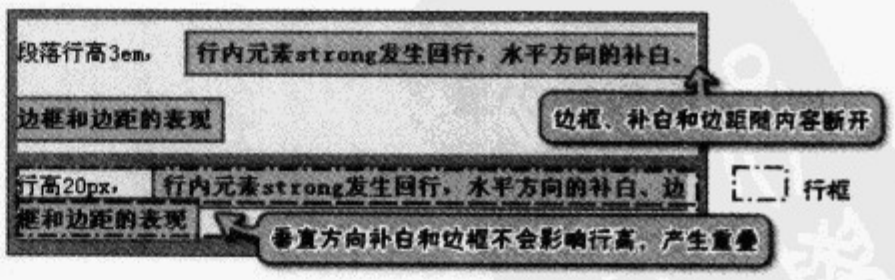


图8-63 行内非替换元素的格式化

```
    <p class="sample1">段落内的文字，<strong>行内元素strong发生回行，水平方向的补白、边框和边距的表现</strong></p>
  </div>
```

由图8-63可以发现，水平方向上边框、边距和补白会随着内容的断裂而断开。行内非替换元素垂直方向的补白和边框从内容边的顶和底部开始计算，而与行高无关，同时行内元素的框高度也不会影响行高，垂直方向的补白和边框可能会在行框高度以外被描绘，因此可能产生重叠的现象。

行内非替换元素的height属性无效，而其内容边由字体决定，但是CSS规范没有详细规定，因此浏览器可能会使用em框或者字体上行字母和下行字母最大高度来生成内容框。

2. 行内替换元素

行内替换元素的4个方向的边距的计算值如果是“auto”，则使用值为0。如果元素的width和heigth属性的计算值都为“auto”，则使用值为其内在尺寸的高度和宽度。

如果height和width的计算值都为“auto”，而元素没有内在的宽度（高度），但是有内在高度（宽度）和内在的比例；或者width和height只有一个为“auto”，另一个有其他的计算值，同时元素有内在比例；此时按照右边公式计算：

```
width的使用值 = 使用的高度 * 内在比例
heigth的使用值 = 使用的宽度 * 内在比例
```

例如下列代码，其显示如图8-64所示。

```
.sample2 { height : 30px; }
.sample3 { width : 120px; }
<div>
 <p>图片的widht和height全为auto：<img src="../../img/ddcat_ad.gif" alt="示例图片" /></p>
   <p>图片的width为auto，而height为30px：<img src="../../img/ddcat_ad.gif" alt="示例图片"class="sample2" /></p>
   <p>图片的height为auto，而width为120px：<img src="../../img/ddcat_ad.gif" alt="示例图片"class="sample3" /></p>
</div>
```

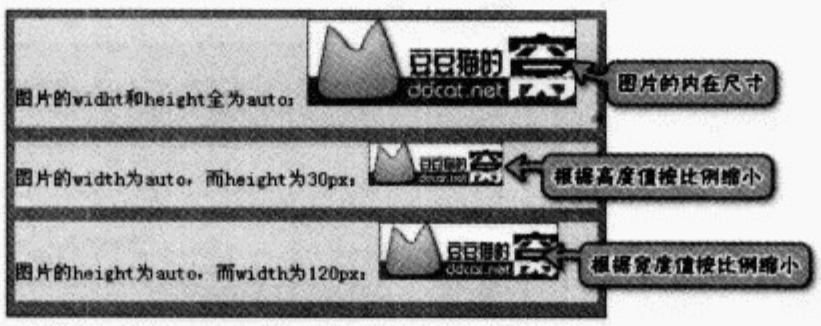
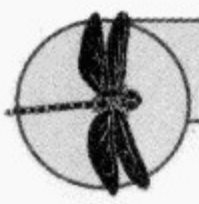


图8-64　行内替换元素的尺寸计算



注意：本例在XHTML代码中没有设定<img>元素的widht和height属性。

如果height和widht的计算值都是“auto”，并且元素有内在比例，但是没有内在的高度或者宽度，同时包含块的宽度不依赖于替换元素的宽度，则widht的使用值根据块级非替换元素的等式计算得出。

如果width和height的计算值为“auto”，而上述条件都不满足，则width的使用值设为300px。如果对于设备300px太宽，则用户端应该按2:1比例，而且适合设备的最大长方形的宽度。

（1）百分比内在尺寸。

百分比内在宽度首先要根据包含块的宽度计算，但是包含块宽度不能依赖于替换元素的宽度，否则元素的百分比内在宽度不能被计算且假定元素没有内在宽度。

如果包含块有明确指定的高度或者替换元素是绝对定位的，则百分比内在高度根据包含块的高度计算。如果不满足以上条件，那么替换元素的百分比值无法计算则假定元素没有内在高度。

（2）表单的替换元素。

对于表单内的替换元素，一般不具有内在比例，当宽度和高度为“auto”的时候，显示的

值由浏览器决定，例如下列代码，其显示如图8-65所示。

```
form { …… }
form p { …… }
#test2 { height : 30px; }
#test4 { width : 140px; }
<form id="testForm" action="#">
 <fieldset>
    <legend>示例表单</legend>
    <p><label for="test1">width/height : auto</label>
    <input name="test1" id="test1" /></p>
    <p><label for="test2">height: 30px;</label>
    <input name="test2" id="test2" /></p>
    <p><label for="test3">width: auto</label>
     <select name="test3" id="test3">
     <option>选择1</option>
      <option>选择选择选择选择选择选择选择选择选择选择2</option>
     <option>选择3选择3</option>
    </select>
    </p>
    <p><label for="test4">width: 140px</label>
     <select name="test4" id="test4">
     <option>选择1</option>
     <option>选择选择选择选择选择选择选择选择选择选择2</option>
     <option>选择3选择3</option>
    </select>
    </p>
    <p><label for="test5">多行文本</label>
    <textarea name="test5" cols="30" rows="3" id="test5"></textarea></p>
 </fieldset>
</form>
```

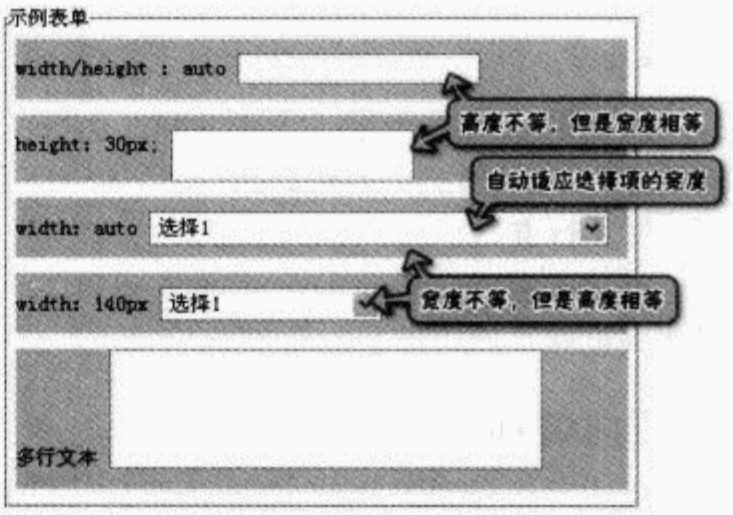
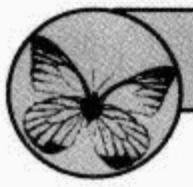


图8-65 表单元素的格式化

不同的浏览器对于表单的显示也不尽相同，如图8-66所示。



提示： 在IE中，不能完全通过CSS属性来控制表单元素，例如下拉列表框的高度等。

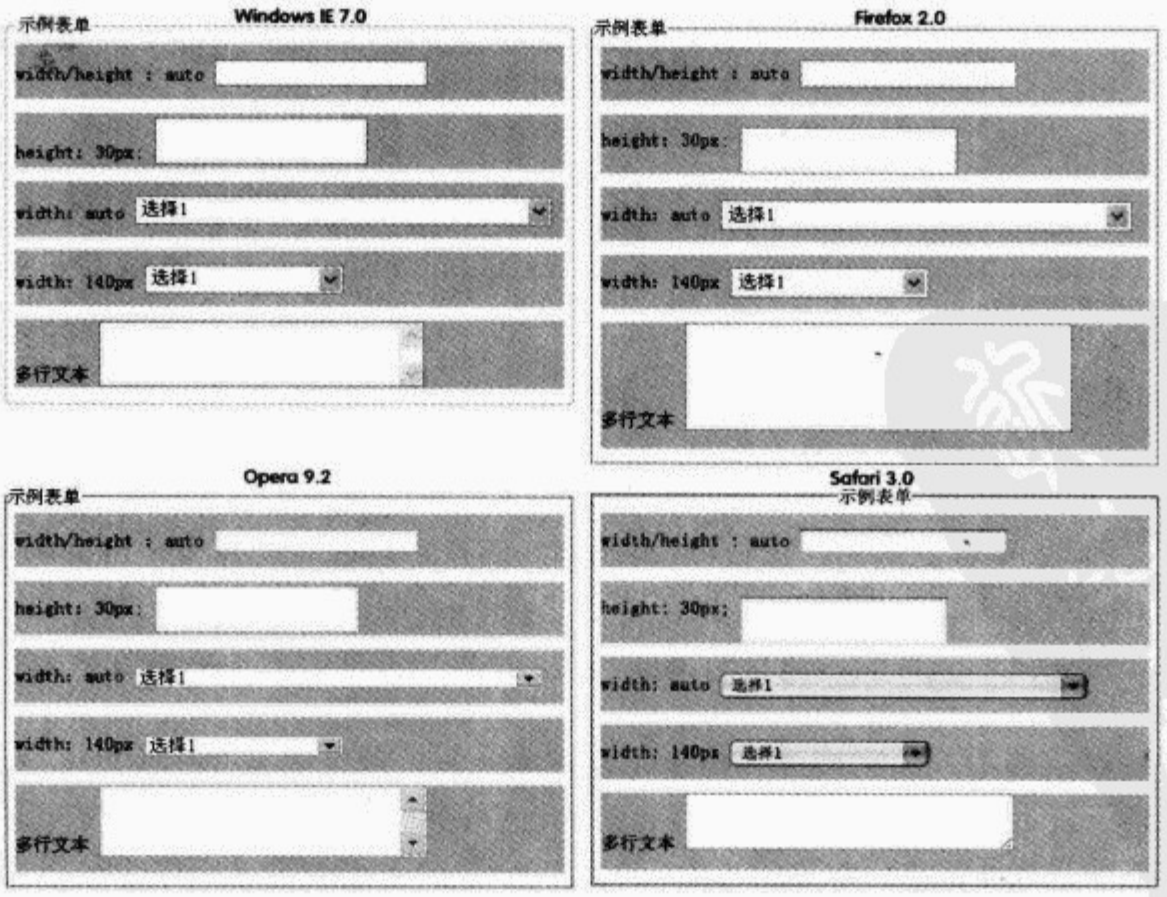


图8-66 不同浏览器对表单元素的显示不同

第9章
浮动、定位与视觉格式化模型

视觉类型的媒体根据CSS的视觉格式化模型（Visual Formatting Model）的规则来处理文档树中的元素，例如如何生成元素框，处理各元素之间的关系，以及根据框的尺寸、定位等CSS属性来确定元素的位置，从而将（X）HTML转化成制作者设计的样子。因此，要掌握使用CSS控制页面内元素的技巧，就需要深入了解框模型及视觉格式化模型的原理。

视觉格式化模型中也有“布局（layout）”的概念，但是这个与页面设计经常用到的类似“3行2列”或者“3行3列”中的排版布局的概念不同，视觉格式化模型中的布局是指每个元素该如何来显示。

在CSS 2.1中，一个控制框的布局可以根据3种定位方案。

- 常规流向。CSS 2.1中，常规流向包含块框的块格式化，行内框的行内格式化，块框或行内框的相对定位，以及插入框的定位。
- 浮动。在浮动模型中，一个框首先根据常规流向布局，再将它从流中取出并尽可能地向左或向右偏移。内容可以排列在一个浮动的边上。
- 绝对定位。在绝对定位模型中，一个框整个地从常规流向中脱离（它对后续的兄弟元素没有影响），并根据一个包含块来分配其定位。

在上一章已经介绍了框模型和一些视觉格式化的概念，本章将详细介绍比较复杂的浮动和定位。

9.1 视觉格式化模型控制框的生成

浏览器根据视觉格式化模型的规定来显示页面内的元素，例如如何显示一个行内元素，因此，理解视觉格式化模型有助于理解各CSS属性会在浏览器内产生何种现象。

注意： 视觉格式化模型并不指定格式化的所有方面（例如，它并不指定字符间距算法），因此浏览器在处理规范未包含的格式化情形时，表现可能有所不同。

在本书［8.2 包含块（containing block）］一节中介绍的视口和包含块的概念，就属于视觉格化模型的基础内容。视觉格式化模型中，文档树中的每个元素都根据包含框模型产生零个或多个控制框。这些框的布局由下列内容控制：框的尺寸和类型、定位方案（常规流向，浮动和绝对定位）、文档树中元素间的关系、外部信息（例如图形的内在尺寸等）。

提示： 在上一章内介绍的框模型属于一种抽象的概念，而理解浏览器在处理页面内元素的方法，可以想象为浏览器将所有的元素及其内容都转化成一个个方框，这些方框有的可以相邻横着码放，有的却必须独占一行，大的框内可能有很多小的框，有些方框设定可能会叠加在别的方框上面。因此，有很概念同“框（box）”相关，例如行内框、行框、块框等。

9.1.1 块框的生成（block box）

在［8.10 常规流向中的视觉格式化］一节中简要介绍了块级元素的格式化。某些display属性的取值产生块级元素：block、list-item和run-in（某些时候），以及table。

1. 主块框

块级元素（除了display属性为“table”的元素）生成一个“主块框（principal block box）”，主块框或者只包含块框（block box），或者只包含行内框（inline box）。主块框决定其后代框的包含块，并且生成内容，而且主块框是在任何定位方案中都要牵涉到的框。主块框参与块级上下文格式化。

某些块级元素在主块框之外生成额外的框，例如列表项（list-item），会产生一个额外的框来放置项目标记（例如项目前面的圆点或者序号），这些额外的框根据主块框来定位。

2. 匿名块框

文档树内的元素在浏览器内的显示，都可以看作为一些矩形框的排布和叠加，虽然有时候似乎制作者并没有定义相关的属性。例如有下列代码：

```
<div>
 div内的文字
 <p>p内的文字</p>
</div>
```

看来包含行内和块内容。为了使格式化简单一些，浏览器会假定有一个匿名块框围绕在文字“div内的文字”周围，如图9-1所示。

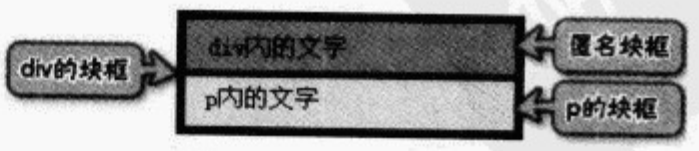


图9-1 匿名块框的生成

换句话说，如果一个块框（本例中 生成的

框）在其中包含另外一个块框（本例中<p>生成的框）或者插入框（run-in box），那么浏览器将强迫在它内部（本例 中）只包含块框或者插入框，而将任何的行内框（本例中 内的文字）都包含在一个匿名的块框或者插入框之内。

匿名框的属性从包含它的非匿名框那里继承而来（本例中是 的框）。不能继承的属性取它们的初始值，例如，匿名框的字体从 继承，但是边距为0。再例如右边代码：

```
body { display : inline; }
p { display : block; }
<body>
 body内的文字1
 <p>p内的文字</p>
 body内的文字2
</body>
```

<body>被设定为行内元素，而其中又包含块级元素<p>，因此其控制框如图9-2所示。

对引起生成匿名包含块的元素设定的属性仍适用于该元素的框和其内容。例如上例中如果为<body>添加border属性，代码如下：

边框会围绕文字1（行结尾处是开放的）和文字2（行开始处是开放的），如图9-3所示。

```
body {
display: inline;
border:2px solid;
line-height:2em;
}
p {
display: block;
}
<body>
body内的文字1
<p>p内的文字</p>
body内的文字2
</body>
```

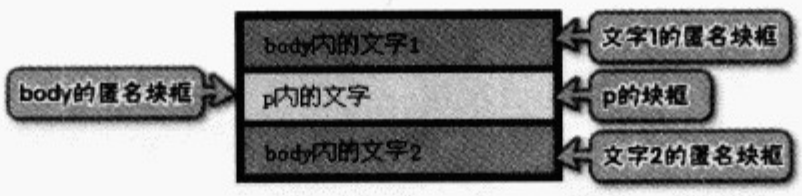


图9-2　匿名块框的生成

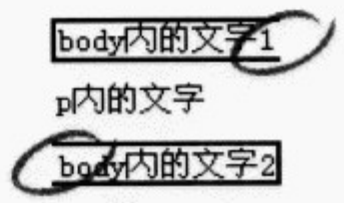
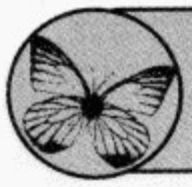


图9-3　引起匿名控制框的元素自身的属性仍有效



提示：读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/anonymous_block_box.html］文件。

虽然CSS 2.1规范中是如此规定的，但是由于在CSS 1和CSS 2的规范中没有相应的详细说明，因此对于不同的用户端，其执行结果可能会有所不同。例如有的浏览器可能会将这种嵌套的框包含在一个“匿名行框”内，而用绘制一个完整的边框，或者一些其他的方式来处理，如图9-4所示。

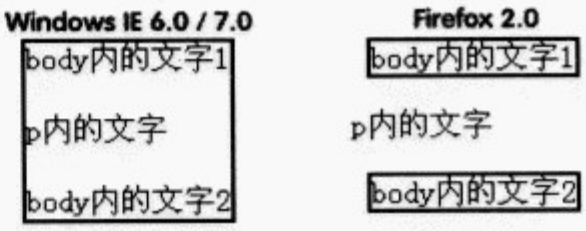


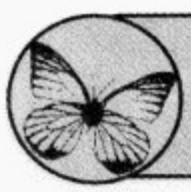
图9-4　不同浏览器的不同处理方法

9.1.2 行内框（inline box）

在［8.10 常规流向中的视觉格式化］一节中简要介绍了行内元素的格式化。某些display属性的取值形成行内元素：inline、inline-table和run-in（某些时候）。行内元素生成行内框。

同块框类似，在一些情况下，浏览器也会生成匿名行内框，例如下列代码：

```
<p>段落的内容1，<strong>strong的内容</strong>，段落的内容2</p>
```



提示：读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/anonymous_inline_box.html］文件。

<p>元素生成一个块控制框，其内还有几个行内框。“strong的内容”的框是一个行内元素产生的行内框，而其他的框（“段落的内容1”和“段落的内容2”）是块类元素<p>产生的。后者就称为匿名行内控制框，因为它们没有与之相关的行内元素，如图9-5所示。

匿名行内框从其父块框那里继承可以继承的属性，非继承属性取其初始值。本例中，匿名框的颜色继承自<p>，而背景是透明的。

空白内容按照white-space属性的设定进行压缩后将不产生匿名行内框。

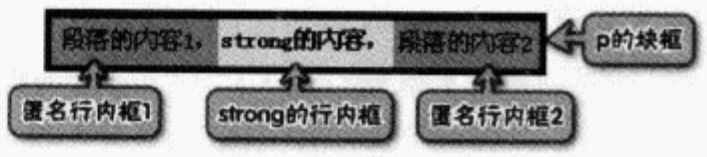
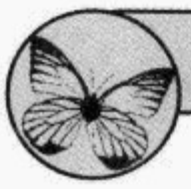


图9-5 匿名行内框的生成

9.1.3 插入框（run-in box）

插入（run-in）类型的元素，有时候可能是块级元素，有时候可能是行内元素，视以下几种情况而定。



提示：“run-in”在印刷术语中的意思是接排部分，插补到正文上的排版。

- 如果插入框包含块框，则插入框为块框。
- 如果一个块框（不浮动，也不是绝对定位）跟随在插入框之后，则插入框将成为块框的第一个行内框，但是插入框不能插入到一个已经以插入框开始或者其自身就是一个插入框的块内。
- 除了以上情况，插入框将成为一个块框。

例如，以下列代码，其显示如图9-6所示。

由图9-6可以发现，由于设定了<h3>的显示类型（display属性）为“run-in”，而跟随<h3>后面的兄弟元素<p>是块级元素（产生块框），因此，<h3>将成为<p>的第一个行内框。

```
h3 { display : run-in; }
<h3>块级元素</h3>
<p>可以生成块框的元素。</p>
<h3>行内元素</h3>
<p>不形成新的内容框，生成行内框的元素。</p>
<h3>插入元素</h3>
<p>根据上下文关系产生块框或者行内框。</p>
```

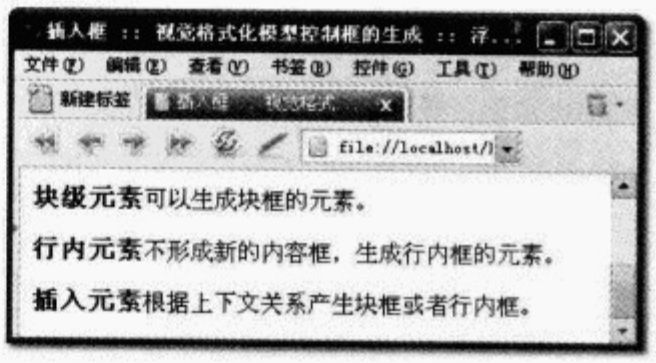
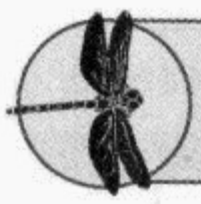


图9-6 插入框与块框



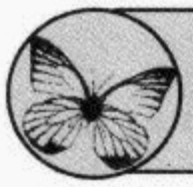
注意：大部分浏览器并不支持“run-in”类的元素，例如IE 6.0 / 7.0和Firefox 2.0。读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/run-in_box.html］文件。

插入元素的属性继承自文档树中它的父元素（本例中为<body>），而不是来自视觉上它成为其一部分的那个块框（本例中的<p>）。

9.2 显示类型：display属性

一个元素框的类型，部分地影响它在视觉格式化模型中的表现，display属性决定了框的显示类型，在前面的章节介绍的块级元素和行内元素，就是最常见的2种类型。

（X）HTML元素都有自己的默认类型，例如 、<p>、<ul>和<li>等是块级（block）元素，而、、和<img>等是行内（inline）元素，而通过设定元素的display属性，可以让变成块级元素，也可以让<p>变成行内元素。

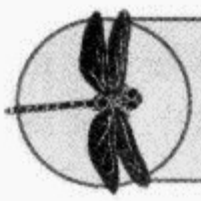


提示：读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/display.html］文件。

9.2.1 语法

display属性具体定义列表如下。

语法	display : inline | block | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit
说明	设定元素的显示类型
值	inline：CSS 1，行内元素将生成1个或多个行内框。 block：CSS 1，块级元素将生成块框。 list-item：CSS 1，将块元素指定为列表项目，并可以添加可选项目标志。 none：CSS 1，元素将不产生任何框，在浏览器内将不显示。 inline-block：CSS 2.1，元素产生一个块框，但是在一个行内框中呈现。 run-in：CSS 2，元素为块级元素或基于内容之上的行内元素。 table：CSS 2，将元素作为块级的表格显示。 inline-table：CSS 2，将表格显示为无前后换行的行内元素。 table-caption：CSS 2，将元素作为表格标题显示。 table-cell：CSS 2，将元素作为表格单元格显示。 table-column：CSS 2，将元素作为表格列显示。 table-column-group：CSS 2，将元素作为表格列组显示。 table-header-group：CSS 2，将元素作为表格头部组显示。 table-footer-group：CSS 2，将元素作为表格注脚组显示。 table-row：CSS 2，将元素作为表格行显示。 table-row-group：CSS 2，将元素作为表格行组显示
初始值	inline
继承性	不继承
适用于	所有元素
媒体	所有媒体
计算值	见下文



注意：在CSS 2中还有“compact”和“marker”2个值，但是由于缺乏广泛的支持，所以CSS 2.1去除了这2个值。

提示：“table、inline-table、table-caption、table-cell、table-column、table-column-group、table-footer-group、table-row和table-row-group”这些值使一个元素表现为类似一个表格元素，将在本书［第11章：表格］内详细介绍。属性值“list-item”请参见本书［12.1列表］一节。

虽然display属性的初始值是“inline”，但是用户端的默认样式表可以超越它。例如，默认状态下，段落<p>就会显示为块级元素，而不是行内元素。“block”和“inline”2个值在此不再

赘述，下面介绍其他几个属性值。

1. none

display属性设定为“none”的元素将不产生任何的框，也就是说，元素对布局没有影响，浏览器将不显示该元素，包括其后代元素。例如下列代码，其显示如图9-7所示。

```
.sample1 { display : none; }
.sample1 strong { display : block; }
<div>
 <p>普通的段落文字</p>
 <p class="sample1">这个段落不显示，<strong>包括子元素</strong>。</p>
</div>
```

普通的段落文字

图9-7 display属性值为“none”的元素不显示

由图9-7可以发现，“sample1”的display属性为“none”，因此浏览器不显示该元素，而且该元素也**不占据任何位置**。而且，虽然“sample1”的子元素设定display属性为“block”，浏览器也不会显示。

2. inline-block

“inline-block（行内块）”是CSS 2.1增加的一个值，“inline-block”类型的元素产生块框，但是却又具有行内元素的特性，例如，可以像行内元素一样在一行内显示，其表现比较类似一个行内替换元素。以下列代码，其显示如图9-8所示。

```
#display1 {
text-align : center;
}
#display1 p {
display : inline-block;
width : 100px;
height : 50px;
}
<div id="display1">
 <h4>inline-block</h4>
 <p>段落是块级元素</p>
 <p>段落是块级元素</p>
 <span>行内元素</span>
</div>
```

图9-8 “inline-block”值的显示

由图9-8可以发现，由于<p>设定了display属性为“inline-block”，因此元素的内容和<p>在一行内显示，但是，同时<p>元素依然具有块级元素的特性，如可以设定宽度高度。同时，由于 设定了文字水平居中“text-align : center;”，因此2个<p>和在 内整体居中了。有些浏览器并不支持“inline-block”类型，如IE 7.0及更早的版本和Firefox 2.0。

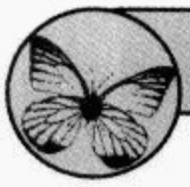
（1）IE中的“inline-block”。

对于IE，对行内元素设定“display: inline-block;”，例如右边代码：

```
span { display: inline-block; }
```

这样会触发IE内部的“hasLayout”属性，从而使行内元素拥有了“inline-block”类型元素的某些表现。而IE中直接设定块级元素“display: inline-block;”是达不到预期效果的，而需要如下设定：

```
div {
display:inline-block; /* 先使用display:inline-block属性触发hasLayout属性 */
……
}
div {
display:inline; /* 再定义display:inline，让块元素呈递为行内元素 */
}
```



提示：关于IE内部的“hasLayout”属性，请参见本书［16.2.1 hasLayout属性］一节。

对于Firefox，虽然可以使用其私有属性“display: -moz-inline-box;”来模拟此效果，但是由于这个私有属性会产生其他问题，因此不推荐使用。

（2）常规流向中的行内块（inline-block）元素的格式化。

常规流向中的行内块元素，如果width属性值为“auto”，则其宽度会被压缩到可容纳其内容的最小宽度，类似于浮动元素。关于浮动元素的宽度压缩，请参见本章［9.4.2 浮动元素的视觉格式化内容］一节。

margin-left和margin-right的计算值如果为“auto”，则其使用值为0；margin-top和margin-bottom如果为“auto”，则使用值为0。“inline-block”类型的替换元素等同于行内替换元素。

3. list-item

“list-item”意为列表项，就类似于（X）HTML内的<li>标签，该类元素会产生一个主块框和一个列表项行内框。

4. run-in

“run-in”属性值根据上下文关系生成块框或者行内框，可以参见本章［9.1.3 插入框（run-in box）］一节。

9.2.2 应用：显示或隐藏元素

利用display属性和:hover伪类，可以实现当鼠标指向某个元素的时候显示某些元素。例如下列代码，其显示如图9-9所示。

```
a:hover {
border: 0;
}
a span {
display : none;
}
a:hover span {
display: inline;
background:#FC3;
}
<p><a href="#" title="鼠标指向显示文字">指过来看看……<span>隐藏的文字出现啦！ </span></a></p>
```

图9-9　display属性的应用

利用<a>元素的:hover伪类，使原本不显示的的display属性为“inline”，因此内的文字又被显示出来。

但是，IE 6.0在处理:hover伪类的时候会出现问题，即当“a”和“a:hover”的CSS定义是一样的，也就是说如果“a:hover”中没有样式的改变，:hover就不会被触发。但如果在“a:hover”增加一些特定的属性，例如右边代码之一：

```
a:hover{ border : none; }
a:hover{ background: none; }
```

此时:hover就可以触发了。这样的属性还包括padding、direction、text-align、text-indent、float、overflow、position等。在实际的应用中，可以结合定位，实现二级下拉菜单、显示详细信息等复杂的效果。

9.3 定位

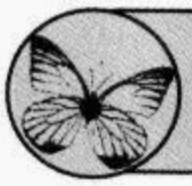
利用定位，可以让制作者按照设计的需要任意安排某个元素的位置，同时，采取绝对定位的元素，与常规流向（常规流向包括相对定位）的元素有不同的格式化方法。

9.3.1 选择定位方式：position属性

position属性可以设定元素的定位方式，其具体定义列表如下：

语法	position : static | relative | absolute | fixed | inherit
说明	设定元素的定位方式
值	static：静态的，无特殊定位。 absolute：绝对定位，将元素从文档流中拖出，使用left、right、top、bottom等属性进行绝对定位，而其堆叠顺序可通过z-index属性定义。 relative：相对定位，依据left、right、top、bottom等属性在正常文档流中偏移位置。 fixed：固定定位，元素在视口内的位置固定不变
初始值	static
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	同指定值

“static”表示元素是静态的，即没有特殊的定位，同时对该元素设定top、right、bottom和left属性是无效的。其余各值的详细介绍请参见后面的章节。



提示：在本章将使用“静态定位”来表示一个元素在常规流向中的位置，即元素的position属性为“static”且float属性为“none”（不浮动）情况下的位置。

9.3.2 设定框偏移：top、right、bottom、left属性

如果一个元素的position属性值的值不是“static”，那么该元素被称为“定位”的。定位的元素生成定位框，其定位基于4个属性：top（上）、right（右）、bottom（下）、left（左）。

top属性的语法如下所示。

语法	top : <长度> | <百分比> | auto | inherit
说明	设定元素的框的边距的顶边相对该框的包含块的顶边向下偏移的量
值	详解见下
初始值	auto
继承性	不继承
适用于	定位元素
媒体	视觉
计算值	对于“position:relative”，参见［9.3.3 相对定位］一节。对于“position:static”，为“auto”。否则，如果指定为长度值，则相应为绝对长度；如果指定为百分比，则为指定值；否则为“auto”

right属性的语法如下。

语法	right : <长度> | <百分比> | auto | inherit
说明	设定元素的框的边距的右边相对该框的包含块的右边向左偏移的量
值	详解见下
初始值	auto
继承性	不继承
适用于	定位元素
媒体	视觉
计算值	对于“position:relative”，参见［9.3.3 相对定位］一节。对于“position:static”，为“auto”。否则，如果指定为长度值，则相应为绝对长度；如果指定为百分比，则为指定值；否则为“auto”

bottom属性的语法如下。

语法	bottom : <长度> | <百分比> | auto | inherit
说明	设定元素的框的边距的下边相对该框的包含块的下边向上偏移的量
值	详解见下
初始值	auto
继承性	不继承
适用于	定位元素
媒体	视觉
计算值	对于“position:relative”，参见［9.3.3 相对定位］一节。对于“position:static”，为“auto”。否则，如果指定为长度值，则相应为绝对长度；如果指定为百分比，则为指定值；否则为“auto”

left属性的语法如下：

语法	left : <长度> | <百分比> | auto | inherit
说明	设定元素的框的边距的左边相对该框的包含块的左边向右偏移的量
值	详解见下。
初始值	auto
继承性	不继承
适用于	定位元素
媒体	视觉
计算值	对于“position:relative”，参见［9.3.3 相对定位］一节。对于“position:static”，为“auto”。否则，如果指定为长度值，则相应为绝对长度；如果指定为百分比，则为指定值；否则为“auto”

这4个偏移属性的含义如图9-10所示。CSS 2.1中规定偏移量的计算以定位元素的边距边为准。

- 长度：长度值，可以为负值。
- 百分比：基于包含块的宽度（left和right）或者高度（top和bottom），可以为负值。

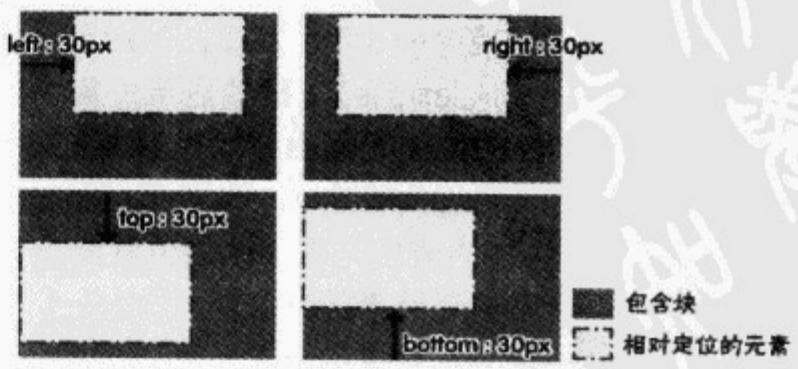


图9-10　4个偏移量的示意

● auto：对于非替换元素，该值的效果取决于与之相关的属性中的哪一个也设定了“auto”。对于替换元素，则该值的效果只与替换内容的内在尺寸有关。

对于负值，则向相反方向偏移，如图9-11所示。

在下面的小节将详细谈论“auto”值的不同情况。

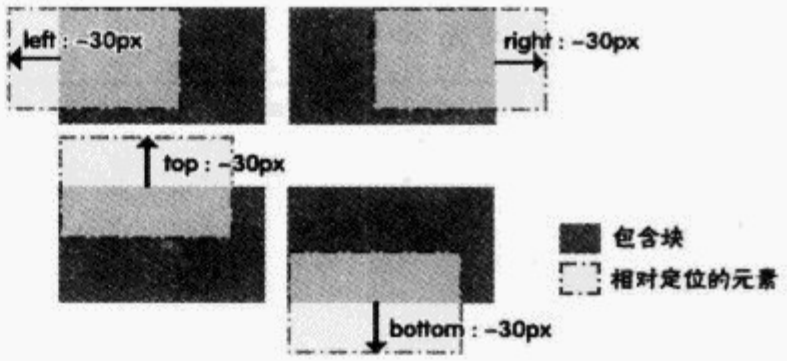
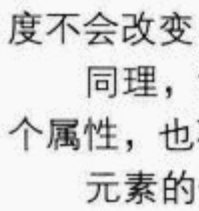


图9-11 负值的偏移方向

9.3.3 相对定位

position属性设定为“relative”的元素是相对定位元素，简单地说，相对定位就是相对于元素的静态定位产生偏移。



提示：本小节示例代码，可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/position_relative.html］文件。

1. 相对定位与偏移量

相对定位的元素属于常规流向布局，即元素还处于文档流内，它对于其后续的兄弟元素仍有影响，同时它生成的框也将在上下文格式化中占据位置，只是元素会根据top、right、bottom和left属性设定的值在自己的静态定位上有所偏移，而不是相对其包含块的边。

例如下列代码，当没有对<p>元素设定定位的时候，其显示如图9-12所示。

```
#position1 {
padding:10px;
……
}
#position1 .sample1 {
……
}
<div id="position1">
  <p class="sample1">段落1中的文字</p>
  <p>段落2中的文字</p>
</div>
```

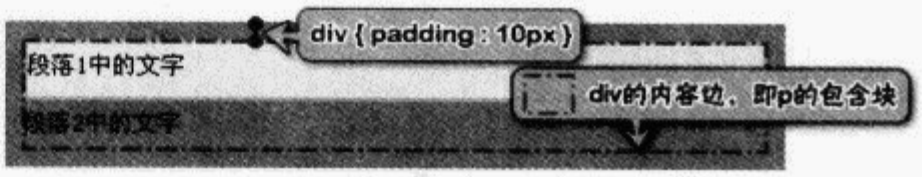


图9-12 未定位的元素的显示

如果增加对“sample1”的定位如下：则其显示如图9-13所示。

```
#position1 .sample1 {
position : relative;
right : 15px;
bottom : 15px;
}
```

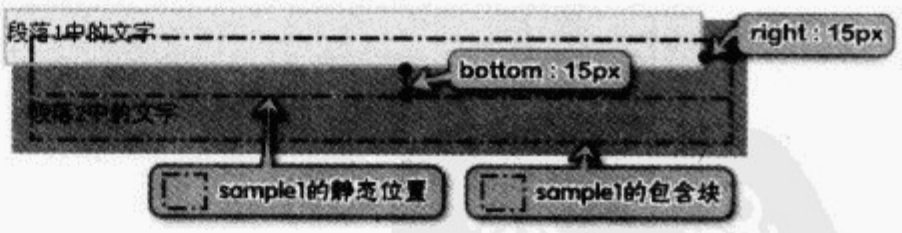


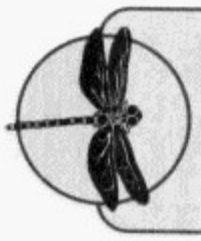
图9-13 相对定位元素的偏移

对比图9-12和图9-13可以发现，“sample1”在 内的占位并没有改变，“段落2”的位置也没有改变，只是“sample1”框的显示位置相对于它自己的静态定位产生了偏移。

设定left属性的值使元素由原始位置向右偏移，而设定right属性使元素向左偏移，元素的宽度不会改变，也不会因为设定了left或者right属性而发生分割或者拉伸。

同理，top属性的值使元素由原始位置向下偏移，而bottom属性则使元素向上偏移，设定这2个属性，也不会影响到元素的高度。

元素的偏移量对其后面的兄弟元素的位置不会有影响，因此可能会发生框的重叠现象。



注意： 相对定位对于下列类型的元素的影响，CSS 2.1规范并未定义：table-row-group、table-header-group、table-footer-group、table-row、table-column-group、table-column、table-cell和table-caption。

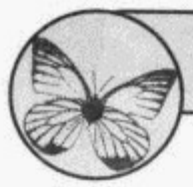
2. “auto”值与宽度、高度和边距的计算

相对定位的元素也属于常规流向的布局，因此其格式化也遵循［8.10 常规流向中的视觉格式化］一节中介绍的规则。

（1）水平方向。

在水平方向，相对定位的元素也遵循7属性的原则，即：

margin-left + border-left-width + padding-left + width + padding-right + border-right-width + margin-right = 包含块的宽度



提示： 关于常规流向的格式化内容，请参见本书［8.10 常规流向中的视觉格式化］一节。

同时，对于相对定位元素的偏移量，还有如下规定。

● 如果left和right属性都为“auto”（初始值），则它们的计算值为0，也就是框水平方向不发生偏移，仍保留在原位。

● 如果left属性值为“auto”，它的计算值是负的right属性的值，也就是框向左偏移right属性的值。

● 如果right属性值为“auto”，它的计算值是负的left属性的值，也就是框向右偏移left属性的值。

● 如果left和right属性的值都不是“auto”，那么定位被过度约束，其中之一的计算值将被忽略，其规则如下：如果包含块的direction属性为“ltr”，则right属性将被强制设定为负的left的值；如果包含块的direction属性为“rtl”，则left属性将被强制设定为负的right的值。

例如下列代码，其显示如图9-14所示。

```
.sample2 {
background : #fc3;
position : relative;
}
<div id="position2">
  <p class="sample2">left为auto，则left = -right</p>
</div>
```

图9-14　left和right属性为“auto”的相对定位元素不发生偏移

当未设定偏移量时，“sample2”停留在原位，其由于未设定其widht属性，因此其宽度占满包含块的宽度（也就是父元素的内容宽度）。如果增加“sample2”的偏移如下，则其显示如图9-15所示。

```
#position2 .sample2 { right : 20px; }
```

图9-15　left属性为“auto”的相对定位

如果同时定义left和right属性，代码如下，其显示如图9-16所示。

```
#position3 p {
background : #fc3;
position : relative;
right : 20px;
left : 30px;
```

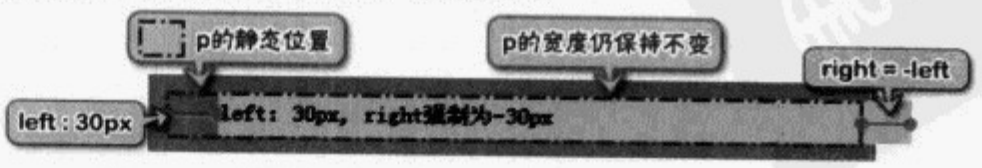


图9-16　同时定义left和right属性的相对定位元素的表现

```
}
<div id="position3">
 <p>left: 30px, right强制为-30px</p>
</div>
```

由图9-16可以发现，由于示例代码的默认direction属性是“ltr”，因此，right属性的值被强制设定为负的left属性的值，即“right: -30px”。

（2）垂直方向。

与水平方向类似，设定top和bottom属性不会拉伸或者分割相对定位的元素的框，如果top和bottom其中1个属性为“auto”，则其值等于另一个属性的负值。如果两个属性都为“auto”，其计算值为0。如果两个属性都不是“auto”，则bottom属性强制为负的top的值。

例如，以下列代码，其显示如图9-17所示。

```
#position4 p {
background : #fc3;
position : relative;
top : 20px;
bottom : 40px;
}
<div id="position4">
 <p>top: 20px, bottom强制为-20px</p>
</div>
```

图9-17 top和bottom属性全都不是“auto”时bottom属性强制为负的top属性的值

3. 产生包含块

上文已经提到过，如果元素设定了“绝对定位（position: absolute）”，包含块由最近的position属性为“absolute”、“relative”或者“fixed”的祖先元素创建。

相对定位的控制框保持它们通常的大小，包括分行以及原来为它们保留的位置。一个相对定位的框创建一个新的包含块来包含常规流向的子元素和定位的后代元素。因此，相对定位最常用的情况，就是为后代中绝对定位的元素产生包含块。

9.3.4 绝对定位

在CSS规范的绝对定位模型中，绝对定位的框，偏移量也由top、right、bottom和left属性来决定，但是与相对定位不同，绝对定位的框，基于其包含块偏移。绝对定位的框不属于常规流向，也就是说，绝对定位的元素脱离了正常的文档流，它既不占位也不会对后续元素的定位有影响。

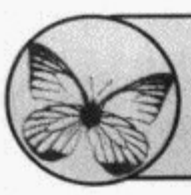
绝对定位的框有margin属性，而且其边距不会重叠。一个绝对定位框为它的常规流向子元素和定位后代元素生成一个新的包含块。不过，绝对定位元素的内容不会在其他框的周围排列。它们可能会也可能不会挡住另外一个框的内容，这取决于互相重合的框的堆叠层级。绝对定位包括绝对（absolute）和固定（fixed）2种情况。

1. absolute

position属性设定为“absolute”的元素，相对于其包含块产生偏移。读者要掌握好绝对定位，就要深入理解包含块的概念。更多信息可以参见［8.2.2 包含块］一节中关于“绝对定位（position: absolute）”的内容。

（1）块级元素。

对于块级元素，其包含块由position属性不是“static（初始值）”的最近的祖先块级元素生成，如果不存在这样的祖先元素，则初始包含块为元素的包含块。例如下列代码：



提示：本小节示例代码，读者可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/position_absolute.html］文件。

```
body {
margin : 15px;
padding : 20px;
border : 3px dashed #909;
font : small/2em "宋体", serif;
}
div {
……
margin : 10px;
padding : 10px;
border : 3px solid #060;
}
.sample1 {
……
position: absolute;
top : 10px;
left : 10px;
}
<body>
  <div id="position1">
    <p class="sample1">段落1中的文字</p>
    <p>段落2中的文字</p>
  </div>
</body>
```

由图9-19可以发现，“sample1”的定位以<body>的上补白边和左补白边，也就是以<body>生成的包含块为基准发生偏移。此时，如果对 添加相对定位如下，则其显示如图9-20所示。

```
div { position : relative; }
```

由图9-20可以发现，“sample1”以 生成的包含块为基准产生了偏移，同时，由于其在 内没有占位，因此和 内的其他内容发生了重叠。

与相对定位类似，负的偏移量将使元素向属性定义的方向的相反方向移动，例如下列代码，其显示如图9-21所示。

```
div {
position : relative;
……

}
.sample2 {
position : absolute;
top : -10px;
left : -10px;
}
<div>
  <p class="sample2">绝对定位的sample2</p>
  <p>普通段落文字</p>
</div>
```

（2）行内元素。

和<body>都未设定position属性，即是静态的，因此，对于绝对定位的段落“sample1”来说，其包含块为初始包含块，因此其显示如图9-18所示。

由图9-18可以发现，“sample1”绝对定位后，其在 内将不再占位。此时，如果对<body>添加相对定位如下，则<body>的补白边包含的区域将生成“sample1”的包含块，其显示如图9-19所示。

```
body { position:relative; }
```

图9-18　包含块为初始包含块的绝对定位（position: absolute）的元素

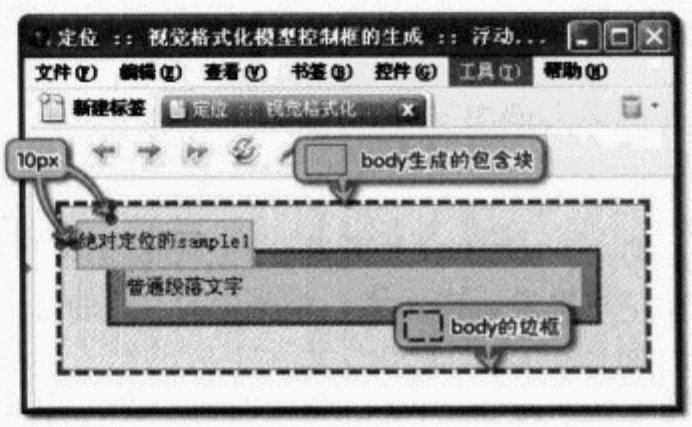


图9-19　<body>生成包含块后“sample1”的定位

图9-20　生成包含块后“sample1”的定位

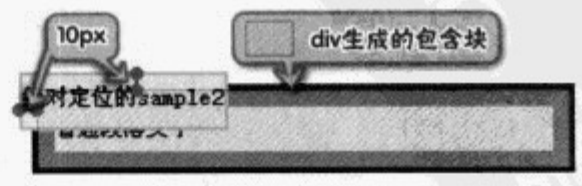


图9-21　负值偏移量的表现

对于行内元素的定位，由于其祖先元素可能是块级元素也可能是行内元素，因此情况变得比较复杂。如果其最近的非“static”的祖先是块级元素，则行内元素的定位同绝对定位的块级元素类似，即以祖先元素的补白边为基准。例如下列代码：

```
#position3 {
line-height : 40px;
……
}
p {……}
strong { …… }
em { …… }
<div id="position3">
  <p>段落内包含<strong>strong元素，strong元素内包含<em>em元素</em>，不同的包含块对于绝对定位的元素影响不同</strong>。在不同浏览器内显示也不同。</p>
</div>
```

当所有的元素都不设定定位属性时（全部是静态元素），此时，元素和元素的包含块由块级祖先元素<p>元素创建，其显示如图9-22所示。

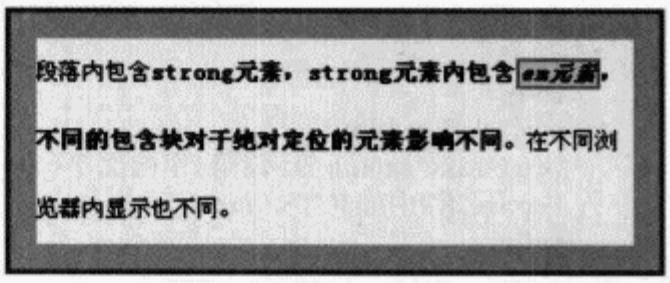


图9-22 静态元素的显示

如果增加如下CSS规则，则其显示如图9-23所示。

```
p {
position : relative;
}
em {
position : absolute;
top : 20px;
left : 35px;
}
```

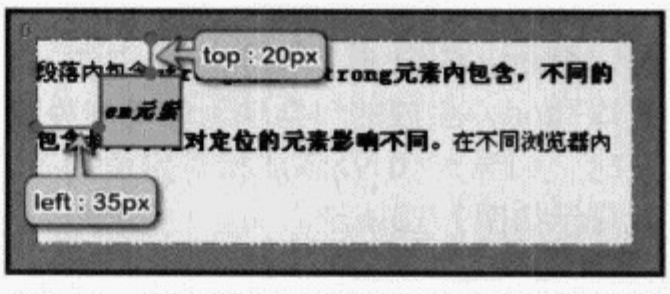


图9-23 块级祖先生成行内元素的包含块的定位

而如果再增加下列CSS规则：

```
strong { position : relative; }
```

此时，元素将生成元素的包含块，元素是行内元素，根据CSS 2.1的规定，元素的偏移是以元素生成的第一个行内框的左边和上边为准（direction属性为“ltr”），因此其效果应该如图9-24所示。

图9-24 CSS 2.1中规定的行内元素绝对定位的计算

但是绝大部分的浏览器并没有这样来计算，而是将元素的第一行的内容区域的上边和左边作为定位的基准，如图9-25所示。

而在Opera 9.2中其显示如图9-26所示。

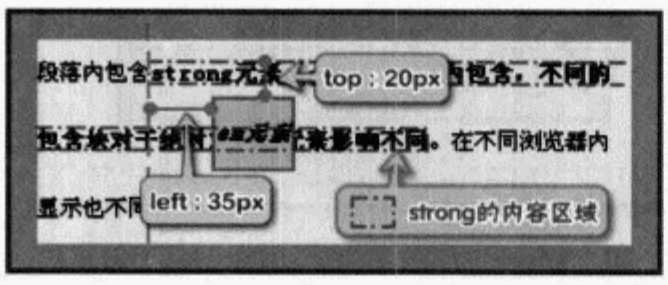


图9-25 大部分浏览器内行内元素的绝对定位

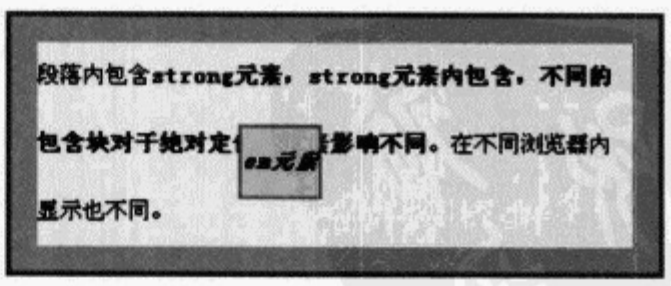


图9-26 绝对定位的行内元素在Opera 9.2中的显示

由图9-26还可以发现，的边框变高了，即边框包含了元素的补白部分，这是因为行内元素决对定位后，被自动转换为块级元素，即绝对定位的元素其display属性的值，会被修改为“block”。

2. fixed

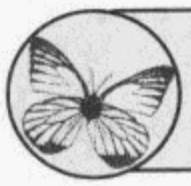
position属性设定为“absolute”或“fixed”的元素都是绝对定位元素，也就是说，固定定

位（position: fixed）是绝对定位的一个子类。

与“absolute”模式类似，“fixed”的元素的边距也不会同其他元素的边距重叠。与“absolute”不同的是，position属性为“fixed”的元素的包含块总是由视点创建。

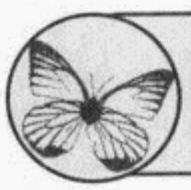
对于媒体类型为“handheld”、“projection”、“screen”、“tty”和“tv”的设备，固定框不会随着文档的滚动而移动。关于媒体类型，请参见本书［3.2.3.4 media属性］一节。

对于媒体类型为“print”的设备，固定定位框在每页里重复，且位置相对于页面框固定，即使页面是通过一个视口来显示（例如在打印预览里面）。对于其他的媒体类型，固定定位属于未定义的属性，即不支持此定义。



提示：制作者可以通过指定媒体类型来实现在某些设备上的固定定位（例如“screen”），而对于另一些设备则不使用固定定位（例如“print”）。

对于电脑屏幕来讲，固定定位的元素将“悬浮”在浏览器的固定位置上，例如下列代码，其显示如图9-27所示。



提示：本小节示例代码，读者也可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/position_fixed.html］文件。

```
.sample1 {
……
position: fixed;
top:10px;
left:10px;
}
<div id="position1">
    <p class="sample1">绝对定位的
sample1</p>
  <p>静态的内容</p>
  ……
</div>
```

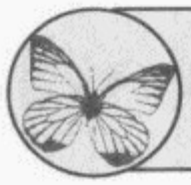
图9-27　固定定位的元素的显示

IE 6.0及其更早的版本不支持固定定位，但是在某些情况下，可以使用“absolute”来模拟一个固定定位的层。在包含块的定义中，当position属性为“absolute”的元素没有position属性为“absolute”、“relative”或者“fixed”的祖先元素时，初始包含块（<html>元素创建的包含块）就是元素的包含块，而此时，绝对（absolute）定位的元素其视觉效果等同于固定（fixed）定位，因此当需要设定固定定位的元素的祖先元素都是“静态（static）”的时候可以如右设定CSS：

由于<html>元素设定了“overflow : hidden”，即不显示滚动条，而针对<body>元素设定了“overflow : auto”，因此实际滚动的是<body>元素的滚动条，由于IE 7.0和其他浏览器支持固定定位，因此可以使用IE专有的条件注释将这段CSS添加在嵌入式样式表中，如右：

```
html{
overflow : hidden; /* 隐藏<html>元素的滚动条*/
}
body {
height : 100%;
overflow : auto;  /* 显示body元素的滚动条 */
}
.sample1 {
position : absolute;
top : 20px;
left : 20px;
}
<body>
<div id="position1">
  <p class="sample1">绝对定位的sample1</p>
  ……
</div>
</body>

<!--[if lte IE 6]>
<style type="text/css">
……
</style>
<![endif]-->
```



提示：如果“sample1”的祖先元素有相对定位或者绝对定位，则不能采用此方法。关于IE的条件注释，请参见本书［16.2.2 条件注释］一节。

3. 绝对定位的水平格式化

本节内一些术语定义如下。

- **静态定位：**指元素在常规流向中的位置。
- **静态定位的包含块：**指假设元素的position属性为“static”且float属性为“none”时元素框的包含块。
- **静态定位的left：**指的是包含块的左边到静态定位的框的左边距边的距离，如果框在包含块的左边，则left值为负。
- **静态定位的right：**指的是包含块右边到静态定位的框的右边距边的距离，如果框在包含块的左边，则right值为正。

例如下列代码，其中“sample3”的静态定位如图9-28所示。

```
<div>
 <p class="sample3">绝对定位的sample3，未设定偏移量</p>
 <p>绝对定位元素后面的段落内的文字，绝对定位元素后面的段落内的文字。</p>
</div>
```

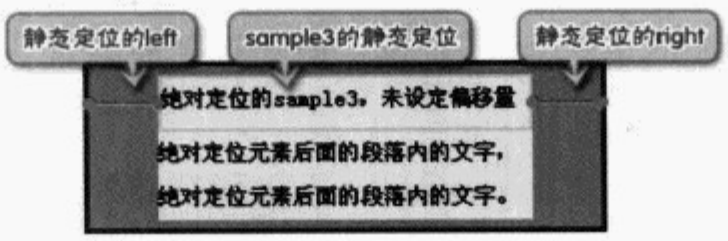
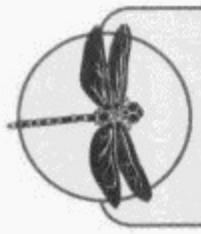


图9-28 静态定位的left和right



注意：为了不去实际计算静态定位的框，浏览器可能会推测一个大概的位置。本小节示例代码，读者可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/position_absolute_horizontal.html］文件。

（1）绝对定位的非替换元素。

绝对定位的非替换元素，其各相关属性的使用值必须满足下面的等式：

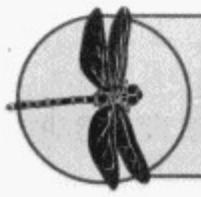
left + margin-left + border-left-width + padding-left + width + padding-right + border-right-width + margin-right + right + 滚动条的宽度（如果存在）= 包含块的宽度

提示：如果边框样式为“none（初始值）”，则边框宽度为0，padding的初始值是0，在本节内为了简化计算过程，将只讨论有边距（margin）、边框（border）、宽度（width）和偏移量的情况。

如果left、width和right属性全为“auto”：

- 设定margin-left和margin-right中的“auto”值为0；
- 如果静态定位的包含块的direction属性为“ltr（rtl）”，设定元素的left（right）为静态定位的left（right）值，width属性会被压缩到其内容的实际宽度，并且计算出right（left）的值。

例如下列代码，其显示如图9-29所示。



注意：若不特别声明，则本节内的示例的direction属性为“ltr”。Opera 9.2及更早的版本中绝对定位的显示存在一定的问题，请用Firefox 2.0或者Safari 3.0浏览示例文件。

```
#horizontal1 {
……
}
.sample3 {
……
position : absolute;
margin : auto;
}
<div id="horizontal1">
 <p class="sample3">绝对定位的sample3，未设定偏移量</p>
 <p>绝对定位元素后面的段落内的文字，绝对定位元素后面的段落内的文字。</p>
</div>
```

由图9-29可以发现，绝对定位的元素的width属性如果为“auto（初始值）”，则会被压缩至其内容的实际宽度，而不是像常规流向中充满父元素的宽度。本例代码中，“sample3”最大允许宽度为：

图9-29　left、width和right属性全部为“auto”的情况

"sample3"最大允许宽度= 包含块宽度-left -"sample3"左边框宽度 - "sample3"右边框宽度

如果元素内容的实际宽度大于等式的计算值，则元素会回行，如图9-30所示。

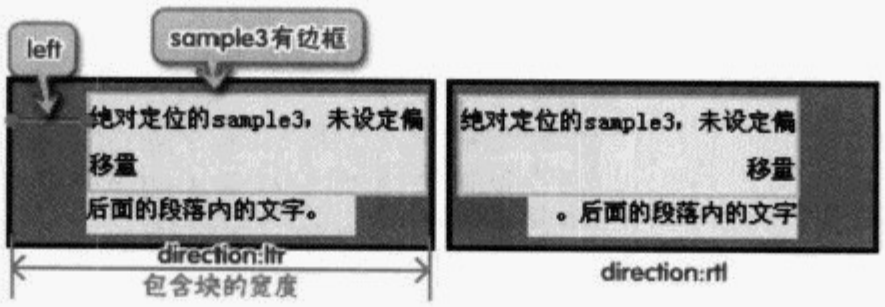
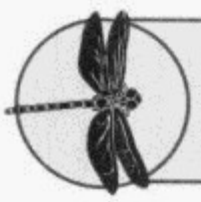


图9-30　width属性为“auto”的元素内容的实际宽度大于计算出的宽度会回行



注意：Windws IE 6.0会将父元素的内容区域宽度作为包含块的宽度来计算，而Windws IE 7.0则会将包含块的宽度直接作为元素可允许的最大宽度。

如果left、width和right属性都不是“auto”。

- 如果margin-left和margin-right都是“auto”，则根据2个边距值相等的原则来计算等式，除非这样计算会产生负值，在这种情况下，如果静态定位的包含块的direction属性是“ltr（rtl）”，设定margin-left（margin-right）为0，计算出margin-right（margin-left）。
- 如果margin-left和margin-right中只有一个是“auto”，则计算出另一个值。
- 如果发生“过度约束”的情况，若包含块的direction属性是“ltr（rtl）”，则忽略right（left）的值，而重新计算其值。

例如下列代码，其显示如图9-31所示。本例中direction属性是“ltr”。

```
div {
……
```

```
padding :10px;
width : 450px;
}
.sample4 {
border:2px solid #06C;
background: #CFF;
width:300px;
margin:0 auto;
position: absolute;
left:30px;
right:50px;
}
<div>
 <p class="sample4">sample4，left、width和right都不是"auto"。</p>
 <p>绝对定位元素后面的段落内的文字，绝对定位元素后面的段落内的文字。</p>
</div>
```

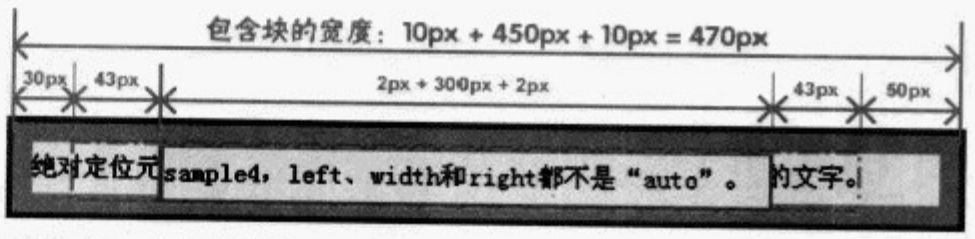


图9-31 左右边距为auto而left、width和right属性有具体的值的元素的绝对定位

对于“sample4”，其包含块的宽度为 = 10px + 450px + 10px = 470px。

由于“sample4”的left、width和right属性都设定了具体的值，而边距为“auto”，则其左边距值 = (470px — 30px - 2px - 300px - 2px - 50px)/2 = 43px。IE不会根据以上公式来计算左右边距的值，而是按照下面这种情况来显示元素。

如果缩小包含块的宽度，使等式产生负值，例如将 的宽度设为：

```
div { width : 280px; }
```

此时包含块的宽度 = 10px + 280px + 10px = 300px，因此，margin-left将被设定为0，则margin-right的值 = 300px - (30px + 2px + 300px +2px + 50px) = -84px，如图9-32所示。

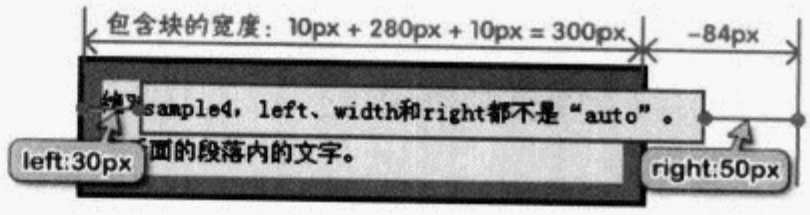


图9-32 计算产生负值时的计算

如果左右边距中只有1个值为“auto”，如下列代码，其显示如图9-33所示。

```
#horizontal5 {
……
padding:10px;
width: 450px;
}
#horizontal5 .sample4 {
……
border:2px solid #06C;
width:300px;
margin:0 10px 0 auto;
position: absolute;
left:30px;
right:50px;
}
<div id="horizontal5">
 <p class="sample4">left、width、right和margin-right都不是"auto"</p>
 <p>绝对定位元素后面的段落内的文字，绝对定位元素后面的段落内的文字。</p>
</div>
```

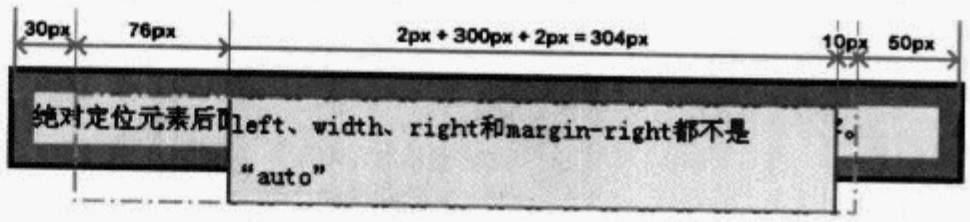
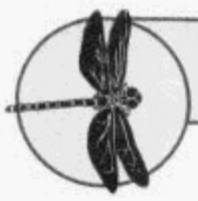


图9-33 左右边距中只有1个值为“auto”的元素的定位

margin-left的值= (10px + 450px + 10px) - (30px + 2px + 300px +2px + 10px + 50px) =76px



注意： IE 可能不能正确计算这个值，而认为margin-left值为0，计算margin-right的值。

如果上面所讲的情况都不符合，则设定margin-left和margin-right中的“auto”值为0，并

且根据下面的规则之一来显示元素。

● left和width属性是“auto”，而right属性不是“auto”，则压缩宽度，计算出left的值。例如下列代码，其显示如图9-34所示。

```
.sample4 {
……
margin : 0 auto;
position : absolute;
right : 50px;
}
<div>
 <p class="sample4">绝对定位元素</p>
 <p>绝对定位元素后面的段落内的文字。</p>
</div>
```

图9-34　left和width属性是“auto”而right属性不是“auto”的情况

● left和right属性是“auto”，而width属性不是“auto”，则如果包含块的direction属性是“ltr（rtl）”，则设定left（right）为静态定位的left（right）值，计算出right（left）值。

对上例中“sample4”的CSS规则修改如下，则其显示如图9-35所示。

```
.sample4 {
……
margin : 0 auto;
position : absolute;
left : auto;
right : auto;
width : 200px;
}
```

图9-35　left和right是“auto”而width属性不是“auto”的情况

● width和right属性是“auto”，而left属性不是“auto”，则宽度被压缩，计算出right值。

对上例中“sample4”的CSS规则修改如下，则其显示如图9-36所示。

```
.sample4 {
……
left : 30px;
right : auto;
width : auto;
}
```

图9-36　width和right属性是“auto”而left不是“auto”的情况

● left属性是“auto”，而widht和right属性不是“auto”，则计算出left。

对上例中“sample4”的CSS规则修改如下，则其显示如图9-37所示。

```
.sample4 {
……
left : auto;
right : 50px;
width : 100px;
}
```

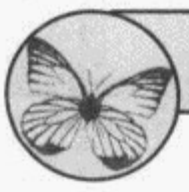
图9-37　left属性是“auto”而widht和right属性不是“auto”的情况

● width属性是“auto”，而left和right属性不是“auto”，则计算出width属性。

对上例中“sample4”的CSS规则修改如下，则其显示如图9-38所示。

```
.sample4 {
……
left : 30px;
right : 50px;
width : auto;
}
```

图9-38　left属性是“auto”而widht和right属性不是“auto”的情况



提示：IE 6.0及更早的浏览器会按照第3种情况来处理此种情况。

● right属性是“auto”，而left和width属性不是“auto”，则计算出right属性。

对上例中“sample4”的CSS规则修改如下，则其显示如图9-39所示。

```
.sample4 {
……
left : 30px;
right : auto;
width : 100px;
}
```

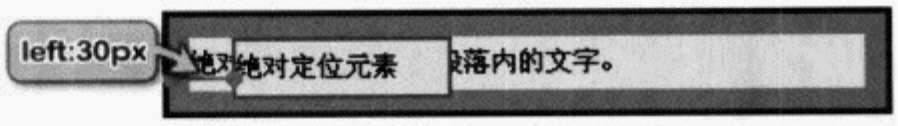
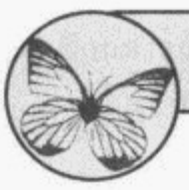


图9-39 left属性是“auto”而widht和right属性不是“auto”的情况

（2）绝对定位的替换元素。

这一情况和前一情形类似，但是替换元素有内在宽度，因此计算的步骤如下。

● width属性的值由行内替换元素决定。



提示：关于行内替换元素的宽度的格式化，请参见本书［8.10.5 行内元素的格式化］一节。

● 如果left和right属性的值都是“auto”，若静态定位包含块的direction属性为“ltr”，设定left为静态定位的left值；若direction属性为“rtl”，设定right为静态定位的right值。

● 如果left或者right属性的值有1个是“auto”，将margin-left和margin-right中的“auto”值设为0。

● 如果到这一步margin-left和margin-right属性的值还是“auto”，则根据这2个值必须相等的原则来计算等式，除非会产生负值，这种情况下，如果静态定位的包含块的direction属性是“ltr（rtl）”，设定margin-left（margin-right）为0，计算出margin-right（margin-left）。

● 如果在这一步只有一个“auto”存在，根据等式解出那个值。

● 如果在这一步发生“过度约束”的情况，若包含块的direction属性是“ltr（rtl）”，则忽略right（left）的值，而重新计算其值。

例如下列代码：

```
div { position : relative; }
img { width: 180px; }
.sample5 { margin: 0 10px 0 20px; }
<div>
  <p>图片绝对定位，<img src="../../img/ddcat_ad.gif" alt="示例图片" class="sample5" />left和right都是auto，
left和right都是auto，left和right都是auto。</p>
</div>
```

图片“sample5”在静态定位时，其显示如图9-40所示。

如果为“sample5”添加绝对定位的规则如下，则其显示如图9-41所示。

```
.sample5 {
position : absolute;
margin: 0 10px 0 20px;
}
```

由图9-41可以发现，图片的left为静态定位的left值。如果修改“sample5”的CSS，为元素设定右边距和右偏移量，则其显示如图9-42所示。

由于left是“auto”而right属性不是，因此margin-left的“auto”值设为0。此时，

图9-40 静态的行内替换元素的显示

图9-41 left和right都是“auto”的替换元素的绝对定位

如果增加left属性的规则如下，则将发生“过度约束”，因此right值被忽略而重新计算，其显示如图9-43所示。

```
.sample5 {
position : absolute;
margin : auto 10px 0 auto;
right : 30px;
}
```

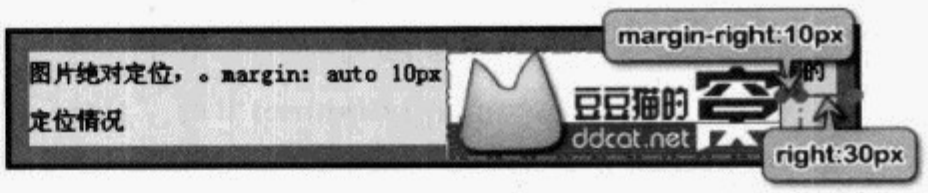


图9-42　right和margin-right属性不为“auto”时替换元素的定位

```
.sample5 {
……
left : 40px;
}
```

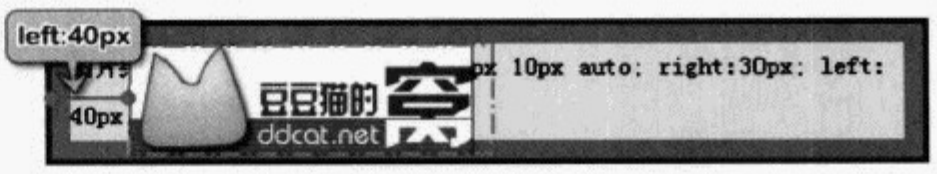
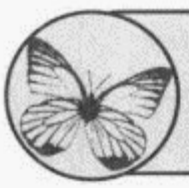


图9-43　“过度约束”情况下right属性值被重新计算



提示：当为<img>元素设定了宽度时，在Firefox 2.0、Opera 9.2和Safari 3.0中不会如此来计算。请使用IE浏览此示例。

4. 绝对定位的垂直格式化

在本书［8.10.3　块级元素的垂直格式化］中介绍过，视口不为初始包含块生成高度，其高度由内容决定。

与水平方向类似，本节内静态定位的top指的是包含块的顶边到静态定位的框的上边距边的距离，如果框高于包含块，则值为负。例如下列代码，其中“sample7”的静态定位如图9-44所示。

```
div {
……
padding : 10px;
height: 80px;
}
.sample7 {
border : 2px solid #06C;
background : #CFF;
width : 200px;
margin : auto;
}
<div>
 <p class="sample7">绝对定位元素内的文字。绝对定位元素内的文字。</p>
 <p>绝对定位元素后面的段落内的文字。绝对定位元素后面的段落内的文字。</p>
</div>
```

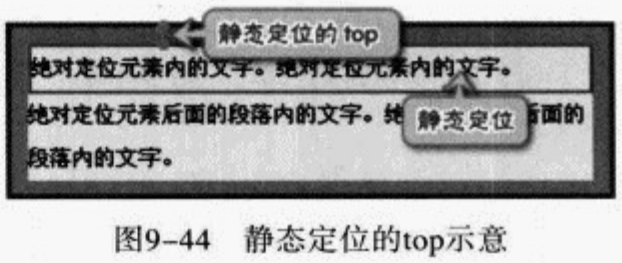
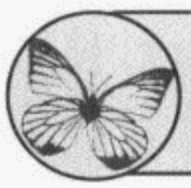


图9-44　静态定位的top示意



提示：本小节示例代码，读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/ position_absolute_vertical.html］文件。

（1）绝对定位的非替换元素。

对于绝对定位的元素，其各相关属性的使用值必须满足如下规则：

top + margin-top + border-top-width + padding-top + height + padding-bottom + border-bottom-width + margin-bottom + bottom + 滚动条的高度（如果存在）= 包含块的高度

提示：如果边框样式为“none（初始值）”，则边框宽度为0，padding的初始值是0，而滚动条的高度与用户端有关，在本节内为了简化计算过程，将只讨论有边距（margin）、边框（border）、宽度（width）和偏移量的情况。

如果top、height和bottom属性值全部为“auto”：

- 设定top的值为静态定位的top值；
- height属性的值由内容决定；
- 设定margin-top和margin-bottom中的“auto”值为0；
- 计算出bottom的值。

例如，对上例中的“sample7”增加CSS规则如下，其显示如图9-45所示。

```
.sample7 {
height : auto;
top : auto;
bottom : auto;
position : absolute;
margin-top: 10px;
}
```

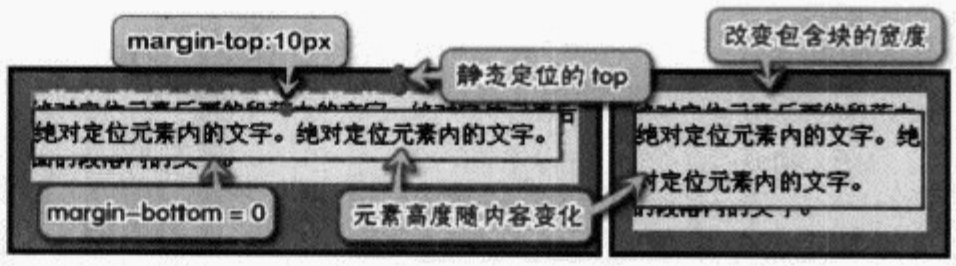
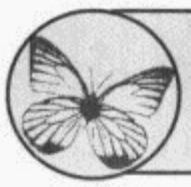


图9-45 绝对定位元素垂直方向的高度由内容决定



提示： height、top和bottom属性的初始值都为“auto”，因此没有特殊情况不用声明，在本例中是作为示例而特别强调。

如果top、height和bottom属性值全都不是“auto”：

- 如果margin-top和margin-bottom全部是“auto”，则根据2个边距值相等的原则来计算等式；
- 如果margin-top和margin-bottom中只有一个是“auto”，计算出其值；
- 如果发生“过度约束”，则忽略bottom的值，并且重新计算其值。

例如，对上例中的“sample7”增加CSS规则如右：

```
.sample7 {
margin: auto;
height : 30px;
top : 20px;
bottom : 15px;
}
```

的高度为80px，补白10px，因此“sample7”的包含块的高度为100px（10px + 80px + 10px）。“sample7”的上下边距的值应该为：

上（下）边距值 = (100px −20px −2px −30px −2px −15px)/2 = 15.5px

由于1px是最小像素单位，因此浏览器可能会将margin-top定为15px，而margin-bottom为16px，如图9-46所示。

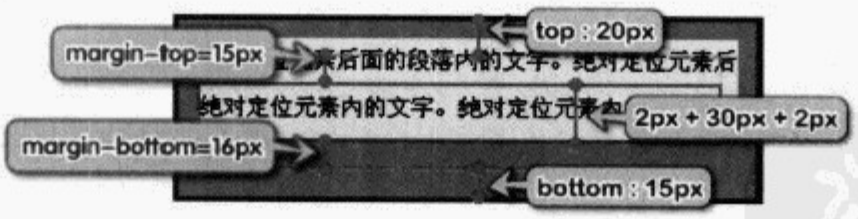
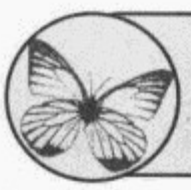


图9-46 margin-top和margin-bottom为“auto”时的计算



提示： IE并不按照此规则来计算边距，而是将margin-top设定为0，而计算margin-bottom的值，如图9-47所示。

如果不是以上两种情况，则按下列6条规则之一来计算。

- 如果top和height属性的值是“auto”而bottom属性的值不是“auto”，则高度基于元素的内容，设定margin-top和margin-bottom中的“auto”值为0，计算出top的值。

例如，修改上例中“sample7”相应的CSS规则如下，则其显示如图9-47所示。

```
.sample7 {
height : auto;
top : auto;
bottom : 15px;
}
```

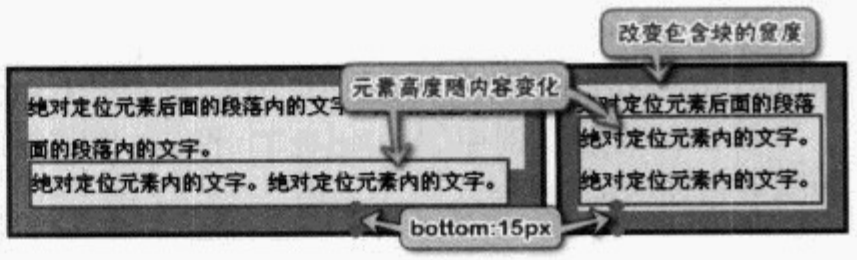


图9-47　top和height属性的值是“auto”而bottom属性的值不是“auto”的情况

● 如果top和bottom属性的值是“auto”而height的值不是“auto”，则设定top的值为静态定位的top值，而设定margin-top和margin-bottom中的“auto”值为0，计算出bottom值。

例如，修改上例中“sample7”相应的CSS规则如下，则其显示如图9-48所示。

```
.sample7 {
height : 30px;
top : auto;
bottom : auto;
}
```

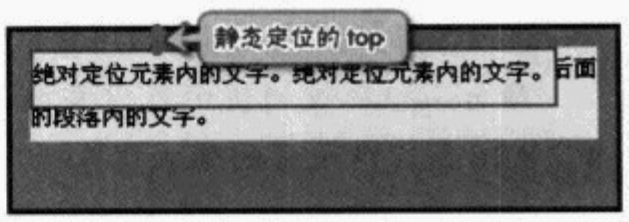


图9-48　top和bottom属性的值是“auto”而height的值不是“auto”的情况

● 如果height和bottom属性的值是“auto”而top的值不是“auto”，则内容决定高度，而设定margin-top和margin-bottom中的“auto”值为0，计算出bottom值。

例如，修改上例中“sample7”相应的CSS规则如下，则其显示如图9-49所示。

```
.sample7 {
height : auto;
top : 20px;
bottom : auto;
}
```

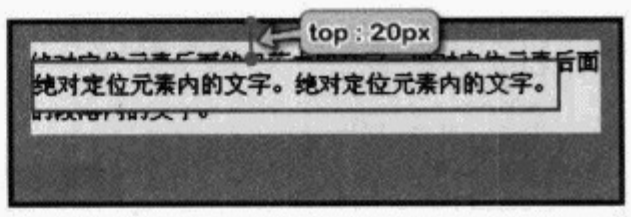


图9-49　height和bottom属性的值是“auto”而top的值不是“auto”的情况

● 如果top属性的值是“auto”而height和bottom属性的值不是“auto”，设定margin-top和margin-bottom中的“auto”值为0，计算出top值。

例如，修改上例中“sample7”相应的CSS规则如下，则其显示如图9-50所示。

```
.sample7 {
height : 30px;
top : auto;
bottom : 15px;
}
```

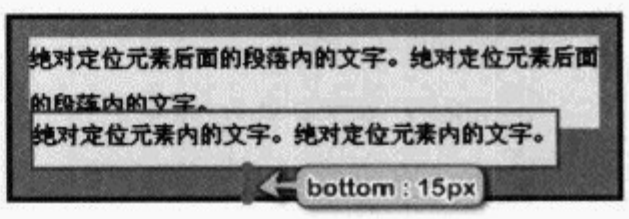


图9-50　top属性的值是“auto”而height和bottom属性的值不是“auto”的情况

● 如果height属性的值是“auto”而top和bottom属性的值不是“auto”，设定margin-top和margin-bottom中的“auto”值为0，计算出height值。

例如，修改上例中“sample7”相应的CSS规则如下，则其显示如图9-51所示。

```
.sample7 {
height : auto;
top : 20px;
bottom : 15px;
}
```

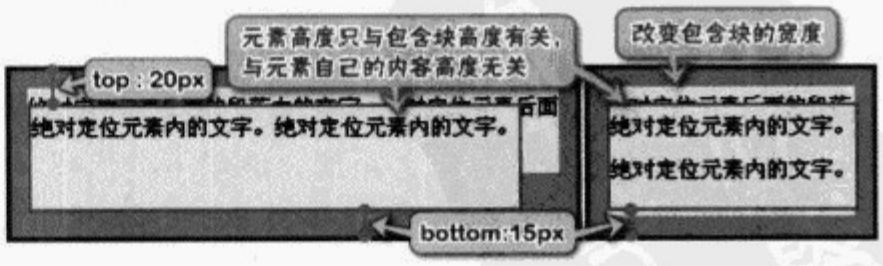
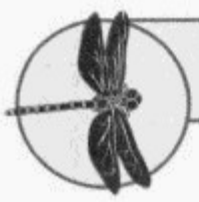


图9-51　height属性的值是“auto”而top和bottom属性的值不是“auto”的情况



注意： IE 6.0不能正确处理此种情况，而是按照前面介绍的第3种情况来显示元素。

● 如果bottom属性的值是“auto”而top和height属性的值不是“auto”，设定margin-top和margin-bottom中的“auto”值为0，计算出bottom值。

例如，修改上例中“sample7”相应的CSS规则如下，则其显示如图9-52所示。

```
.sample7 {
height : 30px;
top : 20px;
bottom : auto;
}
```

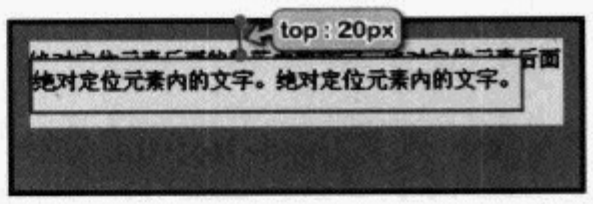


图9-52 bottom属性的值是“auto”而top和height属性的值不是“auto”的情况

（2）绝对定位的替换元素。

这一情形和前述类似，只是元素有一个内在高度，因此计算的步骤如下。

❶ height属性的值由行内替换元素决定。关于行内替换元素的宽度的格式化，请参见本书［8.10.5 行内元素的格式化］一节。

❷ 如果top和bottom属性的值都是“auto”，设定top属性的值为静态定位的top值。

替换元素由于具有内在高度，并且可以设定上下边距（行内非替换元素的上下边距无效），因此，替换元素可能会将行框撑开，例如下列代码，其显示如图9-53所示。

```
div { line-height : 20px; }
.sample8 { margin : 10px 0; }
<div>
  <p><img src="../../img/ddcat_ad.gif" alt="示例图片" class="sample8" width="180" height="60" />文字文字文字</p>
</div>
```

图9-53 行内替换元素高度将行框撑开

如果将图片的垂直对齐方式改为顶对齐，代码如下，则其显示如图9-54所示。

```
.sample8 { vertical-align : top; }
```

图9-54 替换元素的顶对齐

因此，当设定图片为绝对定位后，top和bottom的值为初始值“auto”，其显示如图9-55所示。

```
.sample8 { position : absolute; }
```

图9-55 替换元素的静态定位的top值示意

❸ 如果bottom属性的值是“auto”，将margin-left和margin-right中的“auto”值设为0。

❹ 如果到这一步margin-top和margin-bottom属性的值还是“auto”，则根据这2个值必须相等的原则来计算等式。

但是，IE不会如此计算边距值，例如下列代码：

```
div {
……
padding :10px;
height : 100px;
}
.sample8 {
position : absolute;
margin : auto 0;
top : 15px;
bottom : 10px;
```

```
}
<div>
  <p><img src="../../img/ddcat_ad.gif" alt="示例图片" class="sample8" width="180" height="60" />文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字</p>
</div>
```

“sample8”的top和bottom属性值不是“auto”，因此至步骤❹，则margin-top和margin-bottom的值应该按照相等来计算，但是在部分浏览器内，却不会如此计算，而是按照步骤❸来显示，如图9-56所示。

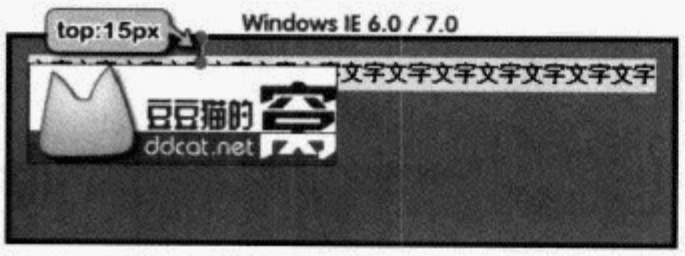


图9-56　当margin-top和margin-bottom属性为“auto”而top和bottom属性不是时浏览器的不同表现

图9-57　符合CSS规范的显示

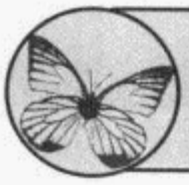
从图9-56可以发现，IE 6.0/7.0将margin-top和margin-bottom设定为0，而重新计算bottom的值。除了IE 6.0以外，其他浏览器可以正常处理此情况，如图9-57所示。

❺ 如果在这一步只有一个“auto”存在，根据等式解出那个值。

❻ 如果在这一步发生“过度约束”的情况，则忽略bottom的值，而重新计算其值。

9.3.5　堆叠顺序：z-index属性

由于绝对定位的元素脱离了文本流，不再占位，因此可能会与静态的内容发生重叠，同样的，如果文档内含有2个以上的定位元素，也可能发生重叠。例如下列代码，其显示如图9-58所示。



提示：本小节示例代码，读者可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/z-index.html］文件。

```
.sample1 {
……
position: absolute;
top:10px;
left:10px;
}
.sample2 {
……
position: absolute;
top:5px;
left:30px;
}
<div>
 <p class="sample1">绝对定位的sample1</p>
 <p class="sample2">绝对定位的sample2</p>
</div>
```

图9-58　绝对定位的元素互相重叠

由图9-58可以发现，“sample2”遮挡住了“sample1”的内容，因为在文档结构内，“sample2”位于“sample1”之后，默认状态下，后面的定位元素会在前面的定位元素之上。

不过通过堆叠顺序z-index属性可以改变这个顺序。z-index属性的具体定义列表如下：

语法	z-index: auto | <整数> | inherit
说明	设定定位元素的当前堆叠内容中框的堆叠次序，同时指明该框是否生成局部堆叠容器
值	auto：生成的框在当前堆叠内容中的堆叠层级和它的父框相同，该框不生成新的局部堆叠容器。 <整数>：该整数是生成的框在当前堆叠内容中的堆叠层级，该框也生成了一个局部堆

续表

	叠容器，在其中它的堆叠层级是0
初始值	auto
继承性	不继承
适用于	定位元素
媒体	视觉
计算值	同指定值

在CSS 2.1中，每个框都有3个纬度，除了前面反复介绍的水平方向和垂直方向以外，z-index使元素框有第3维度：Z轴，z-index数值越大的元素，其位置离访问者越近，如图9-59所示。

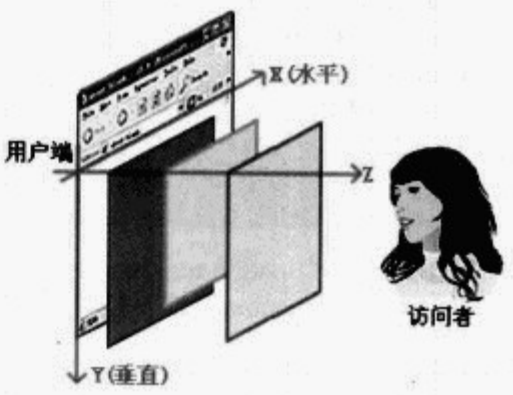


图9-59 元素框的3个维度

在CSS中用“堆叠容器（stacking context）”来描述这种堆叠现象，一个堆叠容器内还可以包含更多的堆叠容器。根元素产生根堆叠容器，每个生成框（包括匿名框）都属于一个堆叠容器，而位于某个堆叠容器内的框也都有一个整数的堆叠层级，这个级别基于它在Z轴上相对于它所在的堆叠容器中其他框的定位，数值大的堆叠层级位于数值小的堆叠层级之上，相同级别的框则根据它们在文档树中的位置来决定上下，后出现的元素在先出现的元素之上。

只有position属性不是“static”的元素才具有z-index属性（初始值为“auto”）。根元素生成根堆叠容器，其他的堆叠容器由z-index属性不是“auto”的定位元素生成（包括绝对定位和相对定位）。通过z-index属性，可以改变定位层的堆叠顺序，例如在上例中，为“sample1”添加CSS规则如下，则其显示如图9-60所示。

```
.sample1 { z-index: 1; }
```

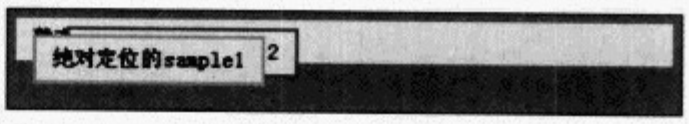
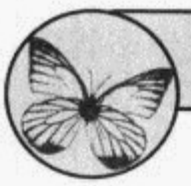


图9-60 通过设定z-index属性改变定位元素的堆叠顺序

每个堆叠容器由下列堆叠层级组成（从后到前的顺序）：

（1）元素的背景和边框构成的堆叠容器；
（2）堆叠层级是负数的后代堆叠容器；
（3）包含流内非行内级后代的堆叠容器；
（4）浮动元素和其内容的堆叠层级；
（5）流内的行内级后代的堆叠层级；
（6）z-index属性值为“auto”的定位后代的堆叠层级和z-index属性值为0的后代堆叠容器；
（7）正堆叠层级的后代堆叠容器。



提示：堆叠容器和包含块没有必然的联系。

由上面的顺序（6），读者就可以理解为什么元素设定了定位（相对或绝对），就可能会位于其他静态内容之上。例如下列代码：

```
.sample1 {
position : absolute;
……
}
.sample2 {
position : absolute;
……
}
strong {
position : relative;
```

```
background: #FC3;
}
.sample1 strong {
color:#f00;
border:2px solid;
}
<div>
 <p class="sample1">绝对定位的sample1<strong>sample1内的strong</strong></p>
 <p class="sample2">绝对定位的sample2<strong>sample2内的strong</strong></p>
 <p>静态的内容</p>
</div>
```

由于4个定位元素都没有设定z-index属性的值，即z-index属性为初始值“auto”，因此，他们所处的堆叠容器是相同的，根据其在文档树内的顺序，“sample2”在“sample1”之后，其显示如图9-61所示。

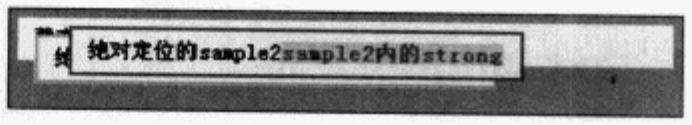


图9-61　z-index属性的“auto”值的堆叠顺序与元素在文档内的物理顺序有关

此时，如果对“sample1”内的元素增加z-index规则如右：

```
.sample1 strong { z-index : 10; }
```

则“sample1”内的元素的堆叠层级就比其他3个元素高，因此将显示在最上面，如图9-62所示。

图9-62　z-index的整数值与“auto”值的关系

但是，如果同时增加“sample2”的z-index属性如右：

```
.sample2 { z-index : 1; }
```

此时“sample2”将生成新的局部堆叠容器，而其后代元素将基于这个堆叠容器来排序。由于“sample2”设定了z-index属性，而“sample1”仍是“auto”，因此“sample2”（次序7）的堆叠层级比“sample1”（次序6）高，因此即使“sample1”内的元素的z-index属性的值大于“sample2”的z-index属性的值，但是由于其父元素的堆叠层级低于“sample2”，所以仍要位于“sample2”之下，其显示如图9-63所示。

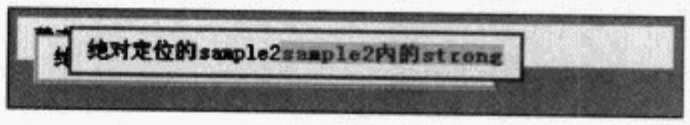
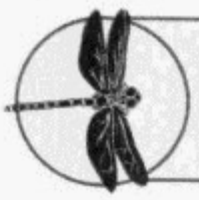


图9-63　元素的z-index与后代元素的关系

由此可见，元素的堆叠顺序还同其父元素的堆叠顺序有关。



注意：IE会为定位元素的z-index属性设定z-index的值为0，因此，不会出现如图9-60所示的情况，子元素的层叠顺序永远由其父元素的顺序决定。

z-index也允许负值，但是其表现在不同的浏览器内可能不太相同，例如下列代码，其显示如图9-64所示。

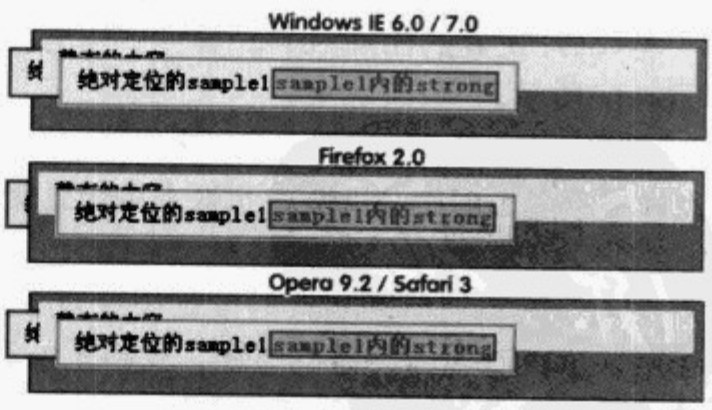


图9-64　不同浏览器对于z-index的负值的不同格式化

```
#z_index6 {
……
z-index : 0;
}
#z_index6 .sample2 {
……
z-index: -1;
left: -20px;
}
<div id="z_index6">
 <p class="sample1">绝对定位的sample1<strong>sample1内的strong</strong></p>
 <p class="sample2">绝对定位的sample2<strong>sample2内的strong</strong></p>
 <p>静态的内容</p>
</div>
```

如果不定义元素的背景，则下面的内容可透过透明区域显示出来，如图9-65所示。

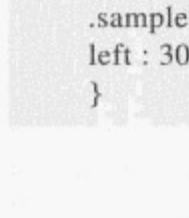
图9-65 透明区域的堆叠显示

9.3.6 IE中的position

IE占有了浏览器市场的绝大部分份额，因此它也是被报告问题最多的浏览器，而定位中的问题是比较突出的。

1. text-align与定位

在IE中，由于text-align属性会影响所有元素的水平位置，因此对于相对定位的元素也会有影响，例如下列代码：



提示：本小节示例代码，读者也可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/ position_ie.html］文件。

```
div {
position : relative;
……
}
.sample1 {
position : relative;
width : 200px;
……
}
.sample2 {
position : absolute;
left : 30px;
……
}
<div>
 <p>普通段落</p>
 <p class="sample1">相对定位</p>
 <p class="sample2">绝对定位</p>
 <p>普通段落</p>
</div>
```

当 元素的text-align属性的值分别为“left”、“center”和“right”时，在IE内显示如图9-66所示。

由图9-66可以发现，由于没有定义偏移量，即left和right都是“auto”，因此定位元素的left为静态定位时的left值。元素水平位置受到text-align属性的影响，因此会如此显示。

如果为定位元素添加偏移量“left : 30px”，CSS如下，则其显示如图9-67所示。由图9-67可以发现，绝对定位的元素的位置按照偏移量正确显示，而相对定位元素仍然受text-align属性的影响。因此在定位元素的时候，要特别注意。

```
.sample1,
.sample2 {
left : 30px;
}
```

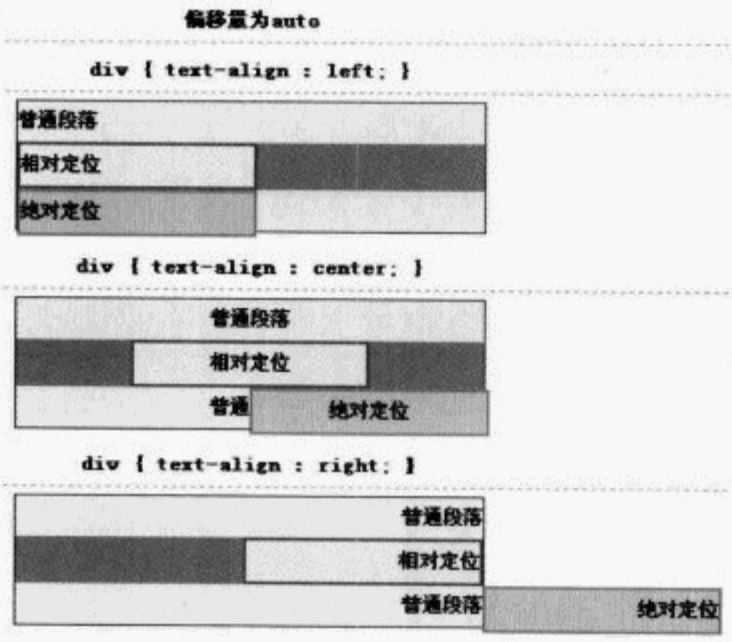


图9-66　IE中text-align属性对偏移量为“auto”的元素的影响

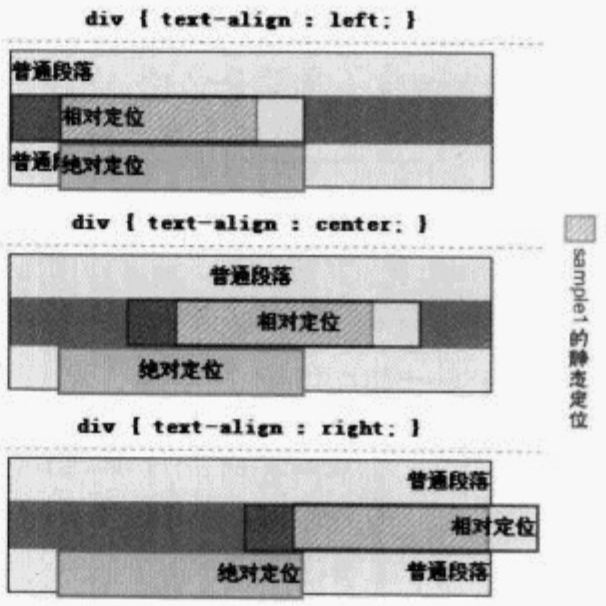


图9-67　IE中text-align属性对相对定位元素的影响

2. IE 6.0的问题

在定位的表现上，IE 6.0存在着一个非常严重的错误，例如下列代码，在IE 6.0内显示如图9-68所示。

```
#ie6-1 {
background : #6C3;
border : 10px solid #060;
position : relative;
margin : 20px;
padding : 10px;
}
.sample3 {
.....
height : 30px;
position : absolute;
top : 20px;
left : 30px;
}
<div id="ie6-1">
 <p class="sample3">绝对定位的sample1</p>
 <p>普通段落文字</p>
</div>
```

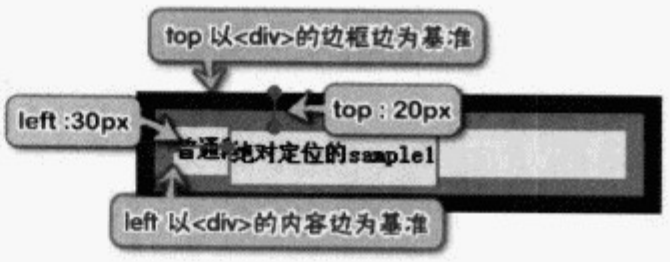


图9-68　IE 6.0对于定位的错误显示

由图9-68可以发现， 没有设定宽度和高度，即使用初始值“auto”，IE 6.0并没有正确显示偏移量，它以错误的基准来计算偏移量。但是，当给 添加一个高度的时候：

```
div { height : 50px; }
```

则在IE 6.0内将正常显示定位，如图9-69所示，这是由于height属性触发了IE内部的“hasLayout”属性，从而使得其能正确解释定位。

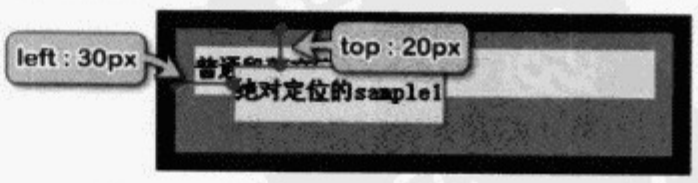
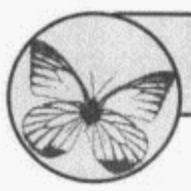


图9-69　增加高度以后IE 6.0显示正常

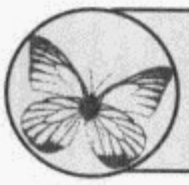


提示：关于IE内部特有的“hasLayout”属性，参见本书［16.2.1 hasLayout属性］一节。

9.3.7　应用：显示提示内容

在本书［2.3.2 常用的XHTML标签和属性］中介绍过，(X)HTML标签的title属性，当鼠标指

向元素的时候，浏览器会将title属性的值显示出来，但是这个提示的样式是制作者不可控制的，而且其容量也有限制。本章［9.2.2 应用：显示或隐藏元素］一节内介绍了利用链接，在指向的时候显示隐藏的文字，而结合定位，也可以实现丰富多彩的提示效果。



提示：因为IE 6.0及更早的版本不支持非<a>元素的:hover伪类，因此对于IE 6.0只能利用<a>元素来实现效果，或者使用JavaScript来模拟:hover伪类的效果。

例如，有设计稿如图9-70所示，相应的XHTML代码如下：

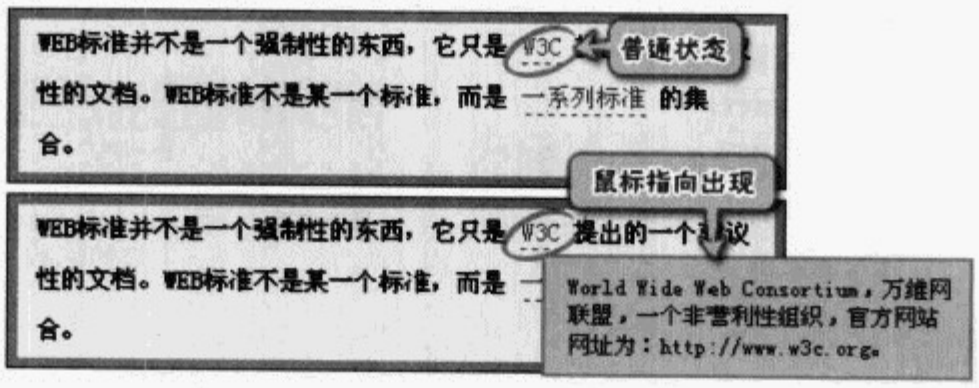
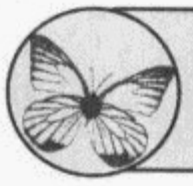


图9-70 提示文字设计稿

```
<div>
  <p>WEB标准并不是一个强制性的东西，它只是<a href="#" class="tips">W3C<span> World Wide Web Consortium，万维网联盟，一个非营利性组织，官方网站网址为：http://www.w3c.org。</span></a>提出的一个建议性的文档。WEB标准不是某一个标准，而是<a href="#" class="tips">一系列标准<span>结构化标准语言主要包括XHTML和XML；表现标准语言主要包括CSS；行为标准主要包括对象模型（如W3C DOM、ECMAScript等）。</span></a>的集合。</p>
</div>
```



提示：在此省略各元素的背景、边框等的设定，读者可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/position_sample.html］文件。

在本例中，链接文字内的元素是对链接文字的说明，因此当鼠标没有指向链接文字的时候不显示，指向的时候才显示出来，以模拟title属性的效果。因此需要设定“tips”内的元素相对于<a>元素绝对定位，将<a>元素相对定位，以生成包含块，CSS规则如下，其显示如图9-71所示。

```
.tips {
position : relative;
}
.tips span {
……
position : absolute;
top : 1.5em;
left : 1.5em;
}
```

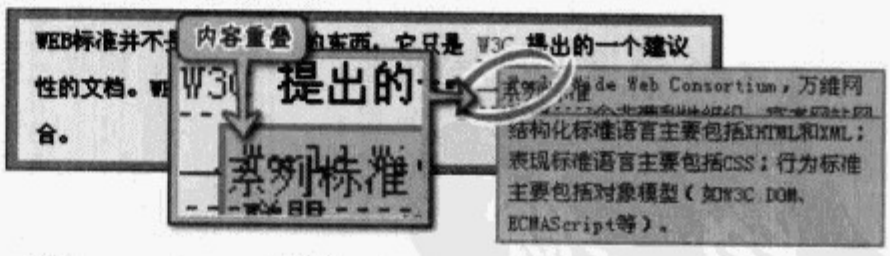
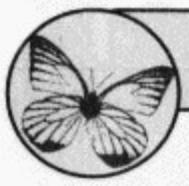


图9-71 元素相对于<a>元素绝对定位后的显示效果

此时，由于堆叠层级的问题，使得第一个<a>元素的内容在第2个<a>元素文字之下，这是由于2个<a>元素都是定位元素，如果修改“.tips”，则会同时作用于2个<a>元素，导致第2个<a>元素永远高于第1个<a>元素，因此，需要将<a>元素的定位属性删除，而增加对:hover伪类的定位，CSS规则如下：

```
.tips:hover { position:relative; }
```



提示：此定义还可以同时解决IE 6.0对于:hover伪类的触发问题。

此时，显示如图9-72所示。

而当鼠标指向<a>元素的时候，触发:hover伪类，<a>元素生成的包含块，因此元素相对于<a>元素绝对定位，如图9-73所示。

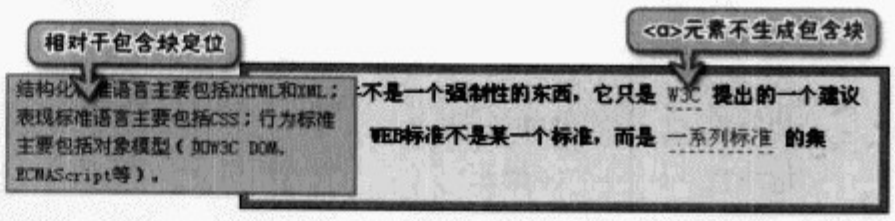


图9-72　改变<a>元素定位属性后的显示效果

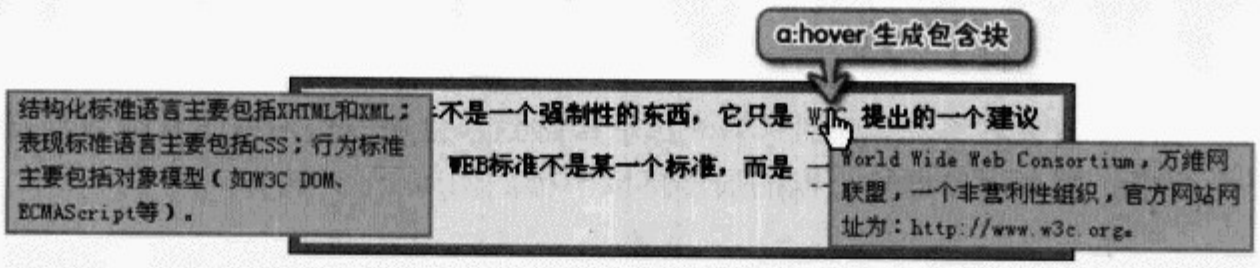


图9-73　当鼠标指向时<a>元素将生成元素的包含块

由图9-73可以发现，由于普通状态的<a>元素不再是定位元素，因此其堆叠顺序小于定位元素，将不会发生遮盖现象。

对于正常状态下不需要显示的元素，可以使用2种方法将其隐藏，设定display属性为none，或者设定其left值为一个非常大的负数值，例如“left:-999em”，此时，元素的显示位置为屏幕的左边框以外，也达到了隐藏的效果，如图9-74所示。

图9-74　将的left设定成负值

而再设定当鼠标指向<a>元素时，将的left属性设为正常值（例如1.5em），则可实现的显示，CSS规则如下，此时鼠标指向链接的效果如图9-75所示。

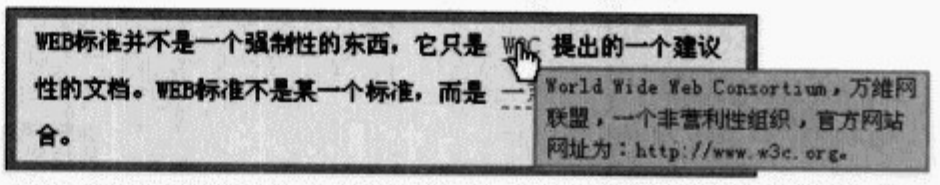
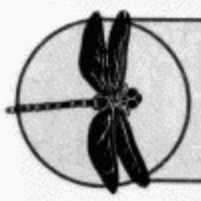


图9-75　鼠标指向出现提示内容的完成效果

```
.tips:hover span { left:1.5em; }
```



注意：Opera对于绝对定位元素的绘制存在程序问题，可能会出现元素显示不完全的现象。

9.4 浮动与清除

常规流向中，块级元素独占一行，但是浮动的元素，却可以在其他内容的左边或者右边显示，因此，浮动是实现页面排版布局的一个重要属性。

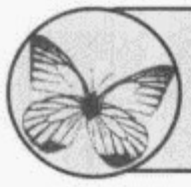
9.4.1 设定浮动：float属性

float属性可以设定元素是否向左或向右浮动，其具体定义列表如下：

语法	float: left | right | none | inherit
说明	指定一个框是否应该向左、向右浮动或不浮动
值	left：该元素产生一个块框，并向左浮动。 right：该元素产生一个块框，并且向右浮动。 none：不浮动
初始值	none
继承性	不继承
适用于	除了绝对定位元素（绝对和固定）和display属性值为“none”的元素
媒体	视觉
计算值	同指定值

一个浮动框向左或者右移动，直到它的外边沿接触到包含块的边沿或者其他浮动框的外边沿。如果存在一个行框，那么浮动框和行框的顶部排成一行。例如下列代码：

```
<div>
  <p><img src="../../img/ddcat_ad2.gif" title="浮动的图片" alt="示例图片" />图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。图片旁边的文字。</p>
</div>
```



提示：本小节示例代码，可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/float.html］文件。

当不设定元素浮动的时候，其显示如图9-76所示。

如果为<img>元素设定CSS规则如下，则其显示如图9-77所示。

```
img {
float : left;
margin-right : 5px;
}
```

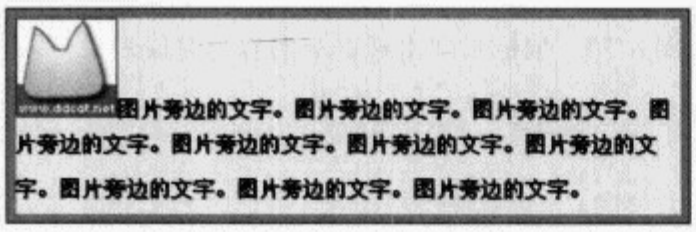


图9-76 常规流向中内容的显示

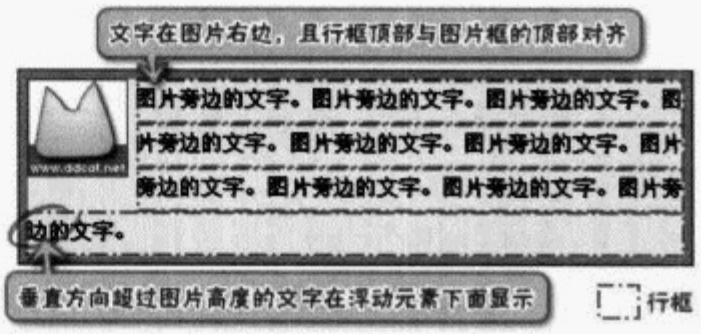


图9-77 设定图片左浮动后的效果

由图9-77可以发现，不浮动的文字的行框缩短了，以容纳下浮动的内容，而在浮动元素的下方，行框又恢复了原来的长度，同时文字第一个行框的顶端与图片的顶端对齐，因此浮动图片与文字形成了一种“图片环绕”的排版效果。相同地，如果设定图片右浮动，其效果如图9-78所示。

```
img {
float : right;
}
```

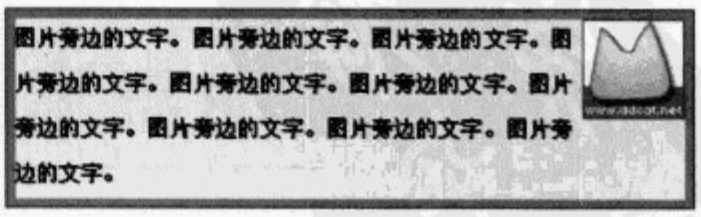


图9-78 设定图片右浮动后的效果

9.4.2 浮动元素的视觉格式化内容

浮动非常有用，但是有时候也容易使人迷惑，因为它不仅涉及包含块，还涉及行框、行内框与块框等内容，特别是不同的浏览器对于浮动的不同解释，更加容易使人产生迷惑。在本小节内将介绍符合CSS 2.1规范的视觉格式化内容，对于某些浏览器有不同的表现，将只做简要介绍。

1. 浮动框的margin、width和height属性

如果浮动框的边距值（4个方向）是“auto”，则其值为0。浮动的非替换元素的width属性值如果为“auto”，则元素宽度将被压缩。如果height属性为“auto”，则高度适应其内容的高度；替换元素则由其内在宽度和高度决定。例如下列代码，其显示如图9-79所示。

```
p {
float : left;
……
}
#float3 p img {
display : block; /* 图片将独占1行 */
}
<div id="float3">
 <p>元素1</p>
 <p>元素2的内容元素2的内容元素2的内容</p>
 <p><img src=" ../../img/btn1.gif" alt=" 示例图片" />元素3</p>
</div>
```

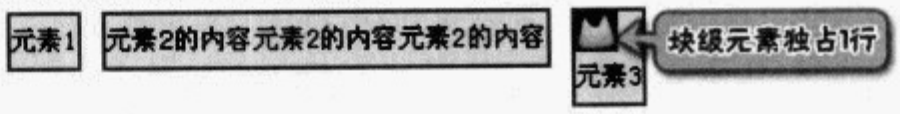


图9-79　width属性为“auto”的浮动元素的宽度被压缩

由图9-79可以发现，浮动元素的宽度为“auto”时，其宽度被压缩到适合其内容的实际宽度。如果浮动元素的宽度大于父元素的宽度，则按父元素的overflow属性决定如何渲染。关于overflow属性，请参见本章［9.6.1 溢出：overflow属性］一节。

浮动元素的height属性如果为“auto”，则其高度会适应其内容的高度，包括浮动的后代元素。例如下列代码，其显示如图9-80所示。

```
#float4 .sample1 {
float : left;
……
}
#float4 strong {
float : left;
height : 50px;
background : #FC6;
}
<div id="float4">
 <p class="sample1"><strong>浮动元素内的浮动子元素</strong>浮动段落的内容。</p>
 <p>段落2</p>
</div>
```

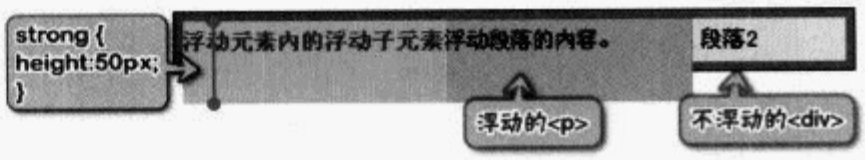


图9-80　浮动元素的heigth属性为“auto”会包含其内的浮动后代元素

由图9-80可以发现， 没有浮动，因此它的高度只包含不浮动的段落2，而“sample1”为浮动元素，因此其高度包含了浮动的子元素。这是浮动元素的一大特性，在页面排版布局的时候非常重要。

浮动元素的垂直边距不重叠，包括元素与其父元素和后代元素之间，例如下列代码，其显示如图9-81所示。

```
#float5 p {
……
margin:20px;
}
#float5 strong {
 display : block; /* 块级元素才具有上下边距和width属性 */
width : 100px;
background : #FC3;
margin : 10px;
}
#float5 .sample1 {
float : left;
background : #6cf;
}
<div id="float5">
 <p class="sample1"><strong>strong1</strong></p>
```

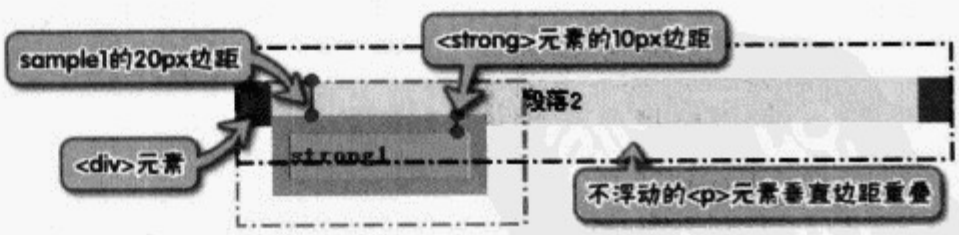


图9-81　浮动框的上下边距不重叠

```
 <p>段落2</p>
</div>
```

由图9-81可以发现，浮动的“sample1”的上下边距以及其子元素的上下边距都不会重叠。不过IE 6.0及以前版本并不是如此显示，它会扩展父元素的高度以包含浮动的子元素，同时IE 6.0存在“浮动元素双倍边距”的问题，即浮动元素如果有与其浮动方向同方向的边距（如左浮动元素有左边距），则会产生双倍的边距，如图9-82所示。

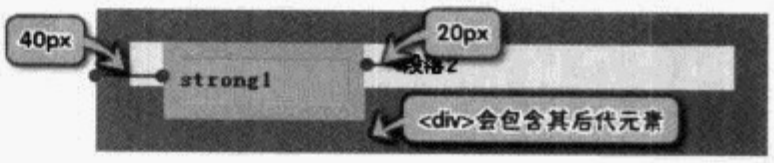
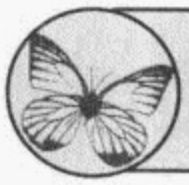


图9-82 IE 6.0的“浮动元素双倍边距”问题



提示：关于“浮动元素双倍边距”的问题，请参见本书［16.4.3 Windows IE浏览器常见Bug］一节。

2. 浮动框与行框、行内框

浮动的框不在流向中，在该浮动框之前或之后创建的非定位块框仍旧垂直排列，就好像该浮动框并不存在一样，不过紧接着该浮动框创建的行框将缩短而给浮动框以空间。

例如下列代码，其显示如图9-83所示。

```
p {
margin : 5px;
……
}
.sample2 {
float:left;
margin:0 20px;
……
}
<div>
 <p class="sample2">段落1段落1段落1段落1</p>
 <p>段落2</p>
 ……
</div>
```

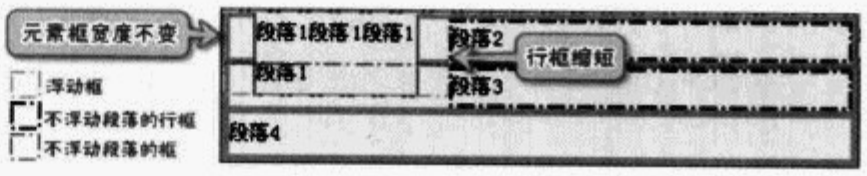


图9-83 浮动框与行框的关系

由图9-83可以发现，不浮动的<p>元素的框并没有缩短，它还是独占1行，只是其内的行框缩短以容纳浮动元素，同时也说明浮动元素脱离了文档流，因此浮动框可能会覆盖多个常规框。在浮动框之前当前行的任何内容将重新排列在浮动框另一边第一个可用的行上。

例如，如果行内框在一个左浮动框之前，那么它将被放置在行框的剩余空间内，左浮动框放置在同一行，并且顶部和行框顶部对齐，而行内框将移动到左浮动框的右边。例如下列代码，2个段落的内容相同，只是第2个段落内的元素左浮动，其显示如图9-84所示。

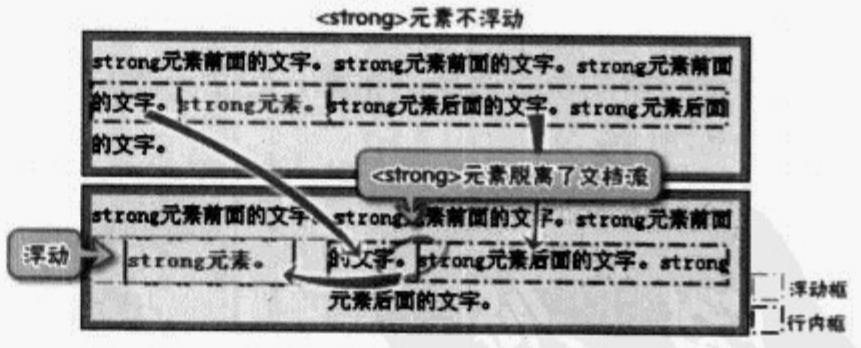
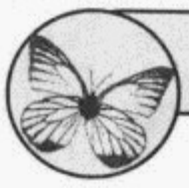


图9-84 行内框与浮动框

```
.sample2 {
float:left;
……
}
<div>
    <p>strong元素前面的文字。strong元素前面的文字。strong元素前面的文字。<strong>strong元素。
</strong>strong元素后面的文字。strong元素后面的文字。</p>
        <p>strong元素前面的文字。strong元素前面的文字。strong元素前面的文字。<strong
```

```
class="sample2">strong元素。</strong>strong元素后面的文字。strong元素后面的文字。</p>
  </div>
```

由图9-84可以发现，浮动的元素是与不浮动时所在行的行框的顶端对齐，而不是整个包含块（<p>元素生成）的顶对齐，因此图9-84中，浮动元素对于第1行的文字无影响。而原本在元素前面（左边）的文字，由于元素的左浮动而移动到其右边。



提示：请读者自己修改代码，来观察当元素右浮动时的表现。

但是对于IE 7.0及更早的版本和Firefox 2.0，却不是如此处理，而是将浮动框移动到本行行框之下，如图9-85所示。此点在使用浮动布局的时候要特别注意。

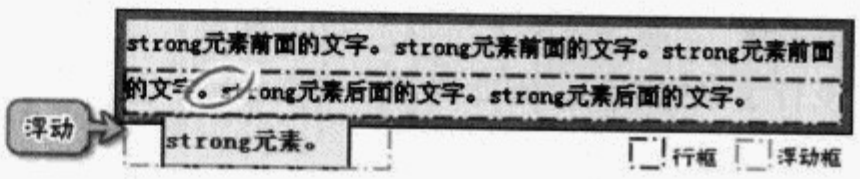


图9-85　Firefox 2.0内对于浮动框和行内框的显示

如果缩短的行框太小而不能容纳下任何其他的内容，则它将向下移动直到有合适的空间或者旁边没有浮动的内容为止。例如下列代码，其显示如图9-86所示。

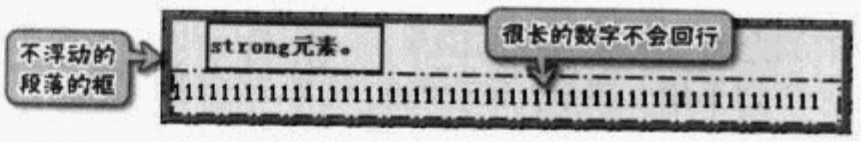


图9-86　很长的数字不会自动回行因此移动到下面显示

```
.sample2 {
float:left;
……
}
<div>
    <p><strong class="sample2">strong元素。</strong> 1111111111111111111111111111111111111111111111111111</p>
</div>
```

由图9-86可以发现，不浮动的段落的位置没有改变，只是它的行框下移了。

同样地，如果没有足够的水平空间来容纳浮动框，则浮动框将向下移动到可容纳它的宽度，或者直到它的旁边没有其他的浮动内容。例如下列代码，其显示如图9-87所示。

图9-87　浮动元素移动到下面显示

```
.sample2 {
float:left;
……
}
<div>
    <p>1111111111111111111111111111111111111111111111111111<strong class="sample2">strong元素。
</strong></p>
</div>
```

3. 多个浮动框

多个浮动框也按照浮动的原则，向左或者右移动，直到它的外边沿接触到包含块的边沿或者其他浮动框的外边沿。例如下列代码，其显示如图9-88所示。

图9-88　多个浮动框的关系

```
.sample3 {
float:left;
margin:5px;
width:100px;
……
}
.w1 {
width:200px;
}
```

```
.h1 {
height:40px;
}
<div id="float9">
 <p class="sample3 h1">段落1段落1</p>
 <p class="sample3">段落2段落2</p>
 <p class="sample3 w1">段落3段落3</p>
 <p>段落4段落4段落4段落4段落4段落4段落4段落4段落4</p>
</div>
```

由图9-88可以发现，第1个浮动框向左移动到包含块的左边沿，第2个浮动框向左移动，直到第1个浮动框的右边沿，第3个浮动框由于没有足够的空间，因此向下、向左移动，直到接触到第1个浮动框的右边沿，此时也有足够的空间容纳它，因此它在此显示。不浮动的段落4的内容在浮动元素的右边显示。但是不同的浏览器可能会有不同的显示，如图9-89所示。

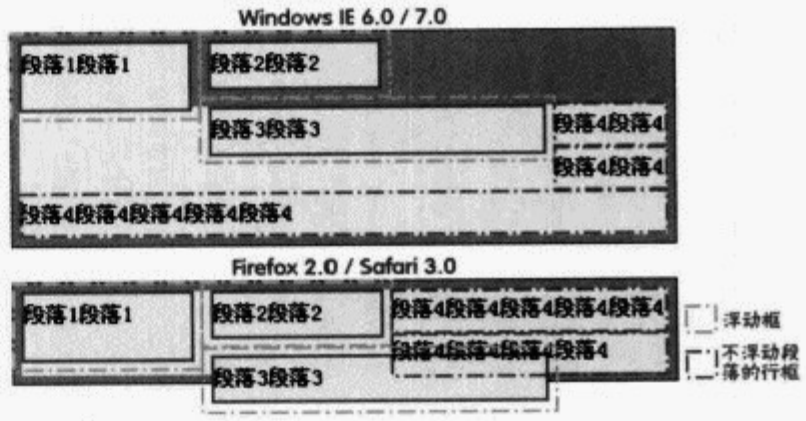


图9-89 浏览器的不同格式化方式

如果增加第3个浮动框的宽度，CSS规则如下，则其显示如图9-90所示。

```
.w1 { width : 300px; }
```

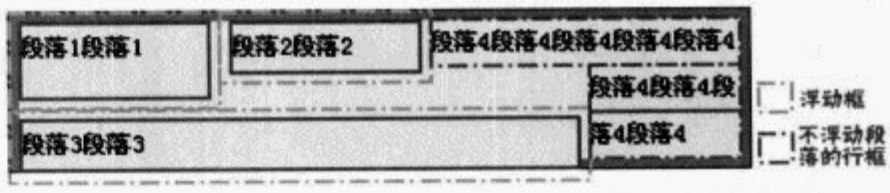


图9-90 浮动框向下移动到可以容纳它的位置

由图9-90可以发现，第3个浮动框继续向下直到一个可以容纳自己的位置。

同样的，不同的浏览器可能会有不同的显示，如图9-91所示。

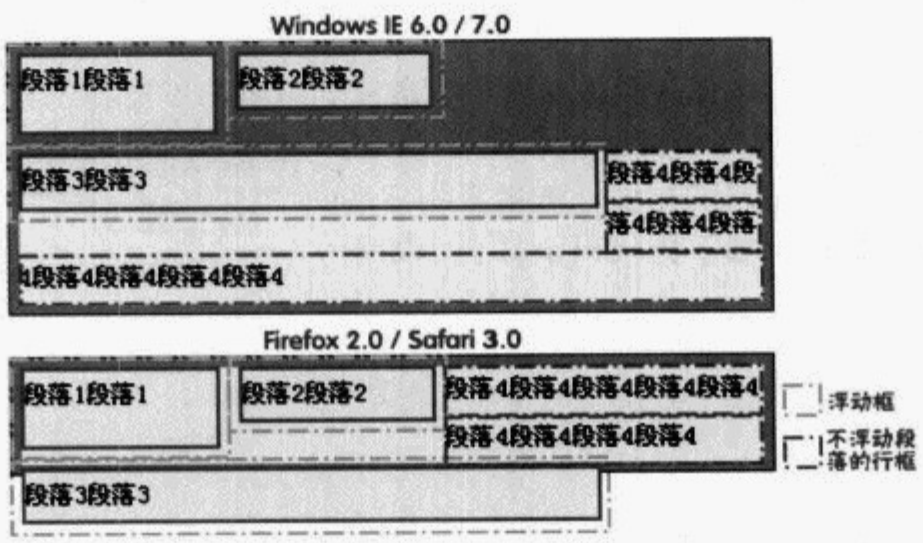


图9-91 浮动框下移时浏览器的不同格式化方式

对于右浮动也是一样，例如下列代码，其显示如图9-92所示。

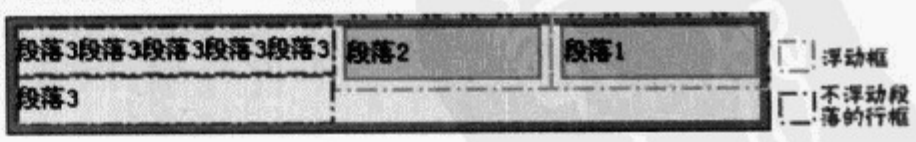


图9-92 多个右浮动框的排列

```
.sample4 {
float:right;
......
}
<div id="float11">
 <p class="sample4">段落1</p>
 <p class="sample4">段落2</p>
 <p>段落3段落3段落3段落3段落3段落3</p>
</div>
```

由图9-92可以发现，虽然2个<p>元素都右浮动，但仍然是第1个浮动框向右移动到包含块的右边沿，而第2个浮动框向右到第1个浮动框的左框边，即第1个右浮动元素在第2个右浮动元素的右边。

当浮动元素和静态元素混在一起的时候，情况就会变得稍微复杂一些，例如下列代码，各元素的浮动过程如图9-93所示。

```
.sample3 {
float : left;
……
}
.sample4 {
float : right;
……
}
<div id="float12">
 <p class="sample4">段落1，右浮动</p>
 <p>段落2段落2段落2段落2段落2段落2段落2段落2</p>
 <p class="sample3">段落3，左浮动</p>
 <p class="sample4">段落4，右浮动</p>
 <p class="sample3">段落3，左浮动</p>
 <p>段落6段落6</p>
</div>
```

由图9-93可以发现，不浮动的“段落2”后面的浮动元素并没有浮动到“段落2”的左边或者右边，而是在它们所在的行内左右浮动。

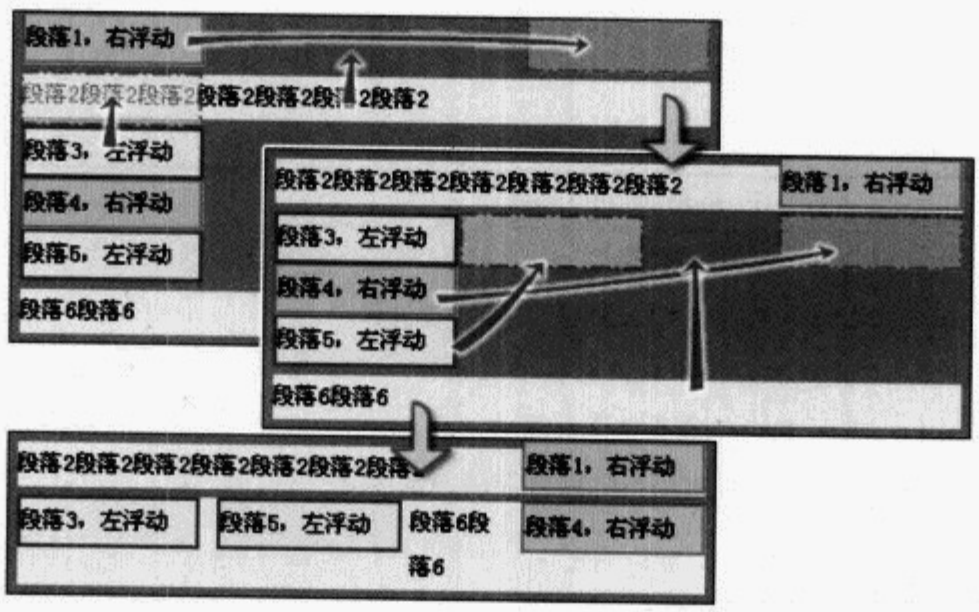
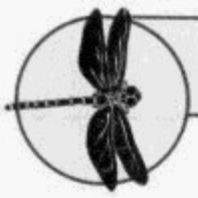


图9-93　多个浮动框的视觉格式化



注意：浮动是实现网页排版布局的重要属性，因此读者要深入理解多个元素浮动的情况。

4. 包含块与浮动元素

浮动元素与其兄弟元素的浮动子元素之间，也可能会互相影响，例如下列代码，其显示如图9-94所示。

```
p {
margin :5px;
……
}
.sample3 {
float : left;
……
}
.sample5 strong {
float : left;
……
}
<div>
```

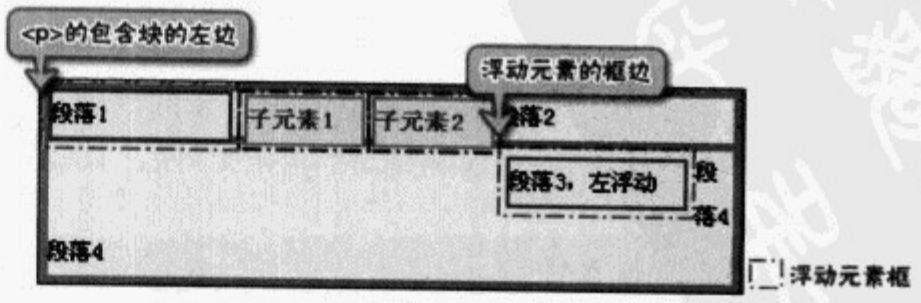


图9-94　浮动元素与不浮动元素的浮动子元素

```
<p class="sample3">段落1</p>
<p class="sample5"><strong>子元素1</strong><strong>子元素2</strong>段落2</p>
<p class="sample3">段落3，左浮动</p>
<p>段落4段落4</p>
</div>
```

由图9-94可以发现，“段落1”向左浮动到包含块的左边缘（ 的内容边），其兄弟元素“段落2”内的“子元素1”左浮动，至段落1的右边沿，而“子元素2”在“子元素1”的右边，“段落3”的左边接触到“子元素2”的右边。

如果增加段落2（“sample5”）的左边距，则其显示如图9-95所示。

```
.sample5 { margin-left: 130px; }
```

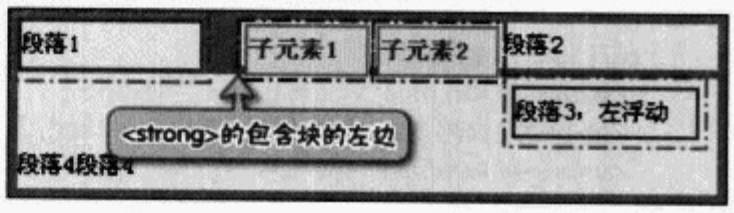


图9-95 包含块对浮动的影响

右浮动的情况也类似。浮动元素如果没有遇到其他浮动元素，则其左右浮动的范围也不会超出其包含块的左右边缘。

9.4.3 清除浮动：clear属性

浮动元素的特点之一，就是可以与其他元素在一行内显示，但是有时候制作者并不希望这样，那么就要使用clear属性来清除浮动。

clear属性的具体定义列表如下：

语法	clear: none | left | right | both | inherit
说明	设定不允许有浮动元素的边
值	none：本元素的左右两边都可以有浮动元素。 left：本元素的左边不允许有浮动元素。 right：本元素的右边不允许有浮动元素。 both：本元素的两边都不允许有浮动元素。
初始值	none
继承性	不继承
适用于	块级元素
媒体	视觉
计算值	同指定值

提示：在CSS 1中，clear属性可以应用于所有元素，而在CSS 2及CSS 2.1中，clear属性只能应用于块级元素。本小节示例代码，读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/clear.html］文件。

1. 属性值详解

清除浮动clear属性用来标识元素的哪一侧不允许出现相邻的浮动框，不过这个浮动框必须是在元素之前出现的，即其静态定位位于元素的前面，同时，清除浮动框不影响本元素内部的子孙元素。例如下列代码，当没有清除浮动的时候，其显示如图9-96所示。

```
.sample1 {
float : left;
……
}
.sample2 {
```

```
float : right;
……
}
<div>
 <p class="sample1">段落1，左浮动</p>
 <p class="clearFloat1">段落2</p>
 <p class="sample2">段落3，右浮动</p>
 <p class="clearFloat1">段落4</p>
</div>
```

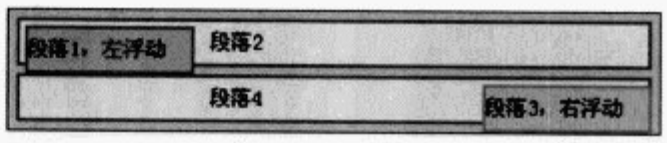


图9-96　未清除浮动时示例的显示

如果增加CSS规则如下，则其显示如图9-97所示。

```
.clearFloat1 { clear : left; }
```

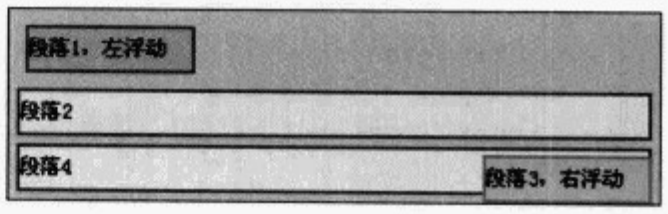


图9-97　清除左浮动后示例的显示

由图9-97可以发现，由于浮动元素后面的<p>元素设定了清除左浮动（clear ： left），而对于“段落4”，因为它前面的“段落3”是右浮动，因此并没有影响。同理，如果设定清除右浮动（clear ： right），CSS规则如下，则其显示如图9-98所示。

```
.clearFloat1 { clear : right; }
```

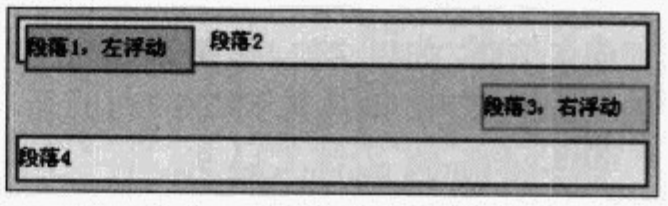


图9-98　清除右浮动后示例的显示

如果设定清除全部（clear ： both），CSS规则如下，则其显示如图9-99所示。

```
.clearFloat1 { clear : both; }
```

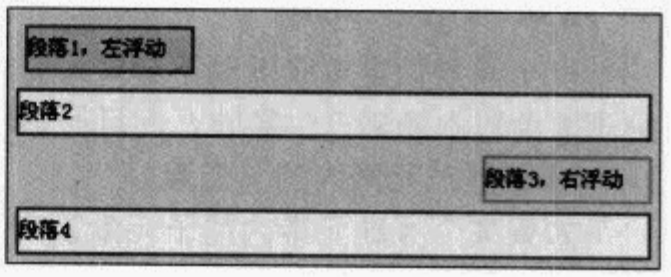


图9-99　清除全部浮动后示例的显示

2. 清除本元素前面的浮动元素

同时，clear属性只对静态定位位于本元素前面的（而不是视觉上的）浮动元素有效，例如下列代码，其显示如图9-100所示。

```
.sample1 {
float : left;
……
}
.clearFloat1 {
clear:left;
}
<div>
 <p class="sample1">段落1，左浮动</p>
 <p class="sample1 clearFloat1">段落2，左浮动</p>
 <p class="sample1">段落3，左浮动</p>
</div>
```

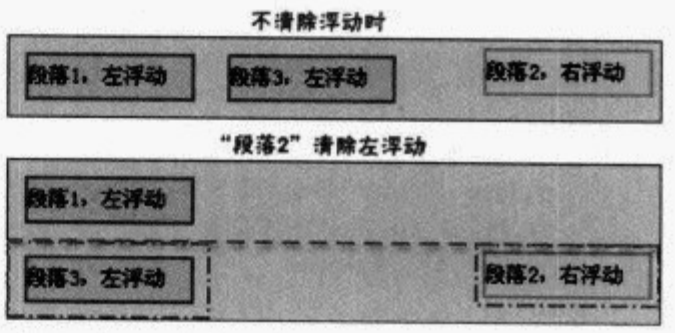


图9-100　清除浮动只对本元素前面的浮动元素有效

由图9-100可以发现，虽然“段落2”设定了“clear ： left”，而在其后左浮动的段落3却未受影响。

3. 浮动与不浮动元素的边距

对于不浮动的非定位元素，使用clear属性要注意：设定了clear属性的非浮动元素，会下降到浮动元素的下面，元素的上边框边与浮动元素的下边距边接触。

例如下列代码，其显示如图9-101所示。

```
p{
margin : 10px 5px;
……
}
```

```
.sample1 {
margin : 20px;
float : left;
……
}
.clearFloat1 {
clear:left;
}
<div>
 <p class="sample1">段落1，左浮动</p>
 <p>段落2，不清除浮动</p>
 <p class="clearFloat1">段落3，清除浮动</p>
</div>
```

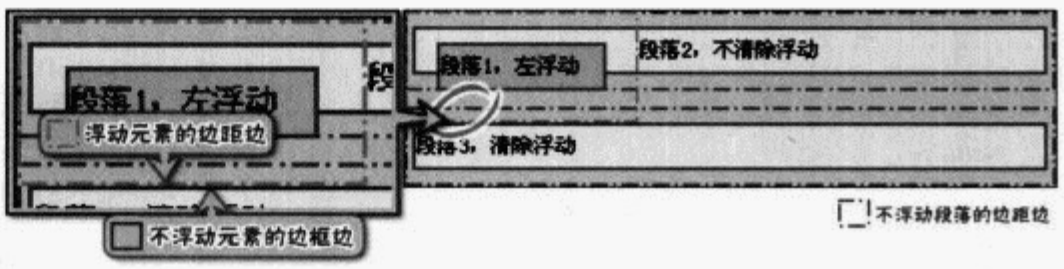


图9-101 清除浮动后不浮动元素的上边框边与浮动元素的下边距边接触

由图9-101可以发现，浮动的“段落1”的下边距似乎和“段落3”的上边距发生了重叠，其实际情况是：浮动的“段落1”将“段落3”顶到了下面，如果增加“段落3(clearFloat1)”的上边距如下，则其显示如图9-102所示。

```
.clearFloat1 { margin-top : 40px; }
```

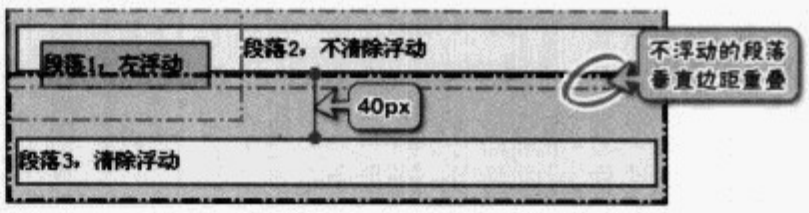


图9-102 不浮动元素垂直方向边距依然会重叠

4. 元素内的子元素

一个元素设定清除浮动不会对本元素的后代产生影响，但是本元素的后代元素如果清除浮动，则可能会对本元素产生影响，而且此影响同本元素是否浮动有关系。

(1) 浮动元素内的子元素。

如果某个浮动元素内的子元素设定了clear属性(除“none”外的值)，则只会在元素内产生影响。例如有如下代码，其显示如图9-103所示。

```
.sample4 {
float : left;
……
}
.sample5 strong {
float : left;
……
}
<div>
 <p class="sample4">段落1</p>
 <p class="sample4 sample5"><strong>子元素1</strong><strong>子元素2</strong>段落2</p>
 <p class="sample4">段落3，左浮动</p>
 <p>段落4段落4</p>
</div>
```

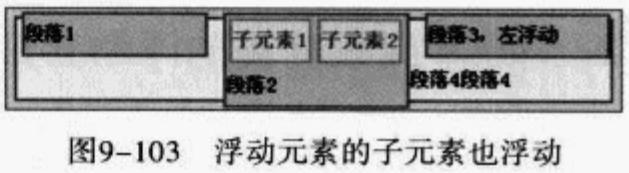


图9-103 浮动元素的子元素也浮动

如果增加“段落2”的子元素的CSS规则如下，则其显示如图9-104所示。

```
.sample5 strong {
float : left;
clear : left;
……
}
```

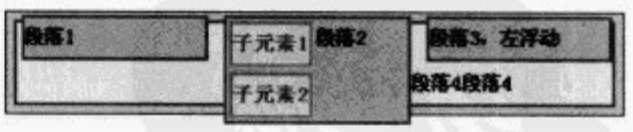


图9-104 浮动元素内的子元素设定clear属性

由图9-104可以发现，“子元素2”由于清除左浮动而降到“子元素1”下面，而子元素1的位置没有变。

(2) 不浮动元素内的子元素。

对于上例，如果修改“段落2”的CSS，使其不浮动，而其子元素仍然浮动，代码如下：

```
.sample5 { float : none; }
<div>
 <p class="sample4">段落1</p>
 <p class="sample5"><strong>子元素1</strong><strong>子元素2</strong>段落2</p>
```

```
<p class="sample4">段落3，左浮动</p>
<p>段落4段落4</p>
</div>
```

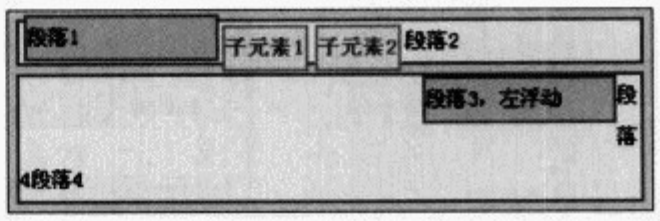


图9-105　非浮动元素内包含浮动元素

元素不清除浮动时，其显示如图9-105所示。

如果增加元素的清除属性如下，则其显示如图9-106所示。

```
.sample5 strong {
float : left;
clear : left;
……
}
```

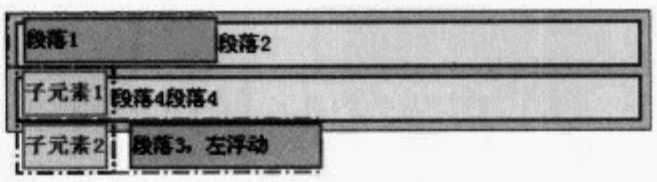


图9-106　非浮动元素内的元素清除浮动

由图9-106可以发现，非浮动元素内的子元素清除浮动后，下降到左浮动的“段落1”之下。如果增加“段落2”的左边距如下，其显示如图9-107所示。

由图9-107可以发现，虽然由于包含块的限制“子元素1”没有与“段落1”接触，但是仍下降到“段落1”下面1行内。此点差异在使用浮动和清除属性实现排版布局的时候需要特别注意。

```
.sample5 {
……
margin-left:160px;
}
```

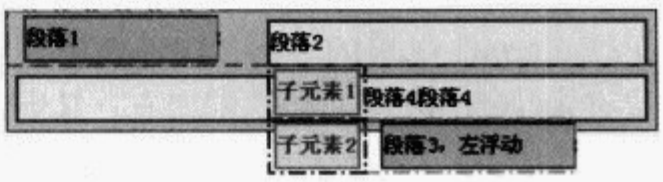
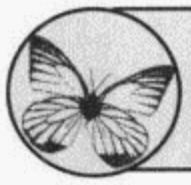


图9-107　非浮动元素内的元素清除浮动与包含块

9.4.4　应用：3行3列布局设计

在［8.10.2　应用：宽度自适应的布局］一节内已经介绍过，利用浮动和边距实现2行2列自适应的设计布局，而本节内介绍的，是3行3列定宽的布局。



提示：本小节示例代码，读者可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/ float_sample.html］文件。

设计图如图9-108所示，其XHTML代码如下：

```
……
<body>
<div id="wrap">
 <div id="header">这里是#header</div>
 <div id="menu">……</div>
 <div id="content">……</div>
 <div id="other">……</div>
 <div id="footer">这里是#footer</div>
</div>
</body>
```

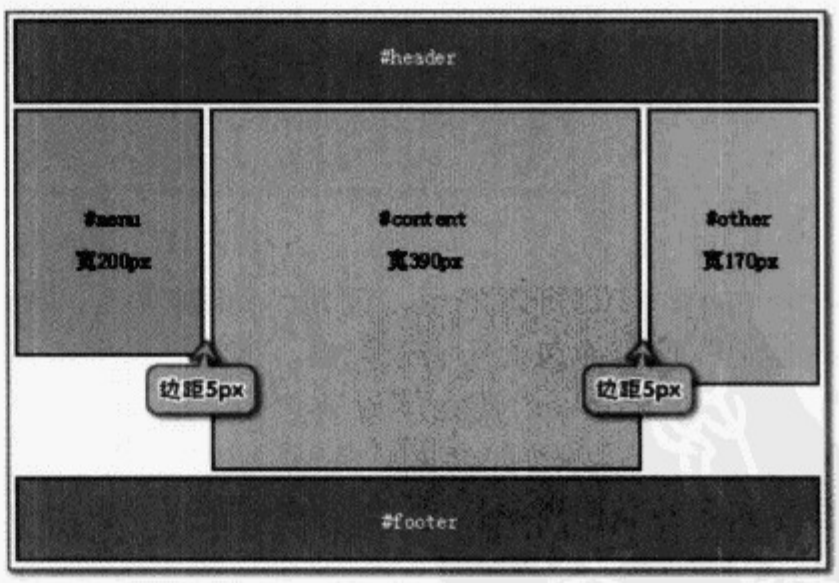


图9-108　3行3列定宽布局设计示意图

由于中间3列的宽度固定，因此可以设定3列左浮动，CSS规则如下，其显示如图9-109所示。

```
#header {
……
}
#menu {
width : 200px;
```

```
float : left;
margin-right : 5px;
background : #9f9;
}
#content {
width : 390px;
float : left;
margin-righ : 5px;
background : #9cf;
}
#other {
width : 170px;
float : left;
background : #fc6;
}
```

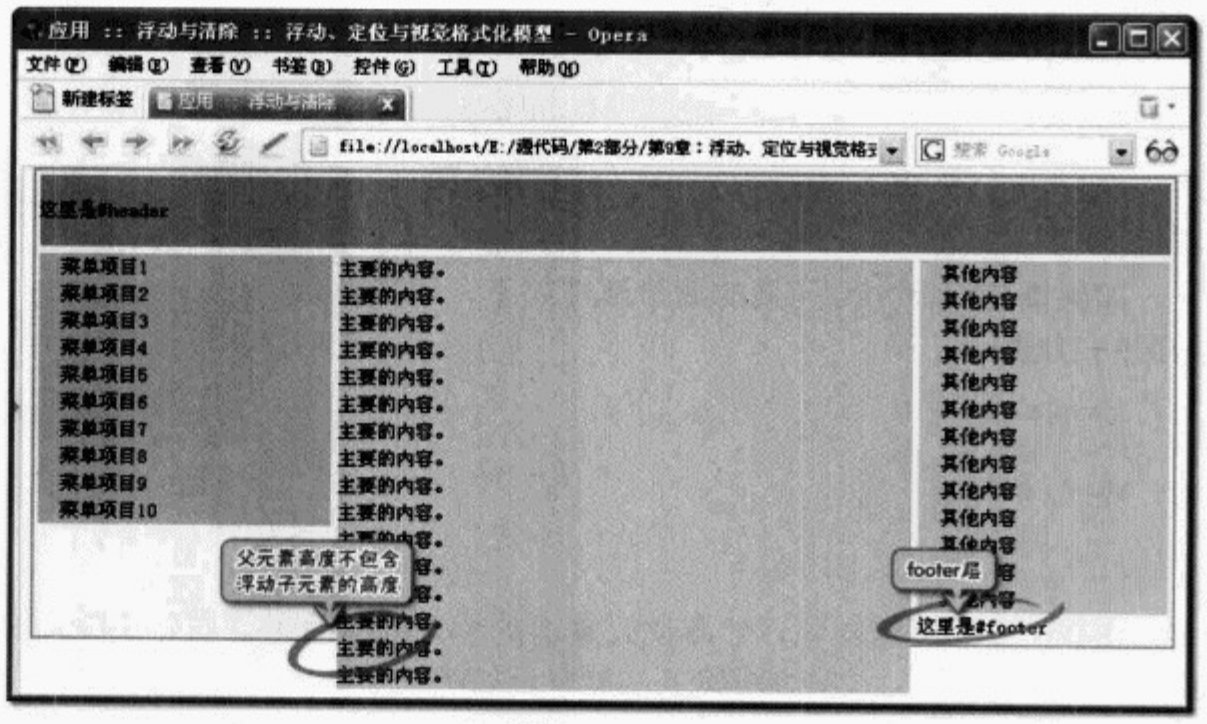


图9-109 浮动层布局

由图9-109可以发现，不浮动的footer层出现在content层的右边，因此需要对footer层设定clear属性，CSS规则如右，其显示如图9-110所示。

```
#footer {
……
clear : both; /* 也可只清除左浮动 */
}
```

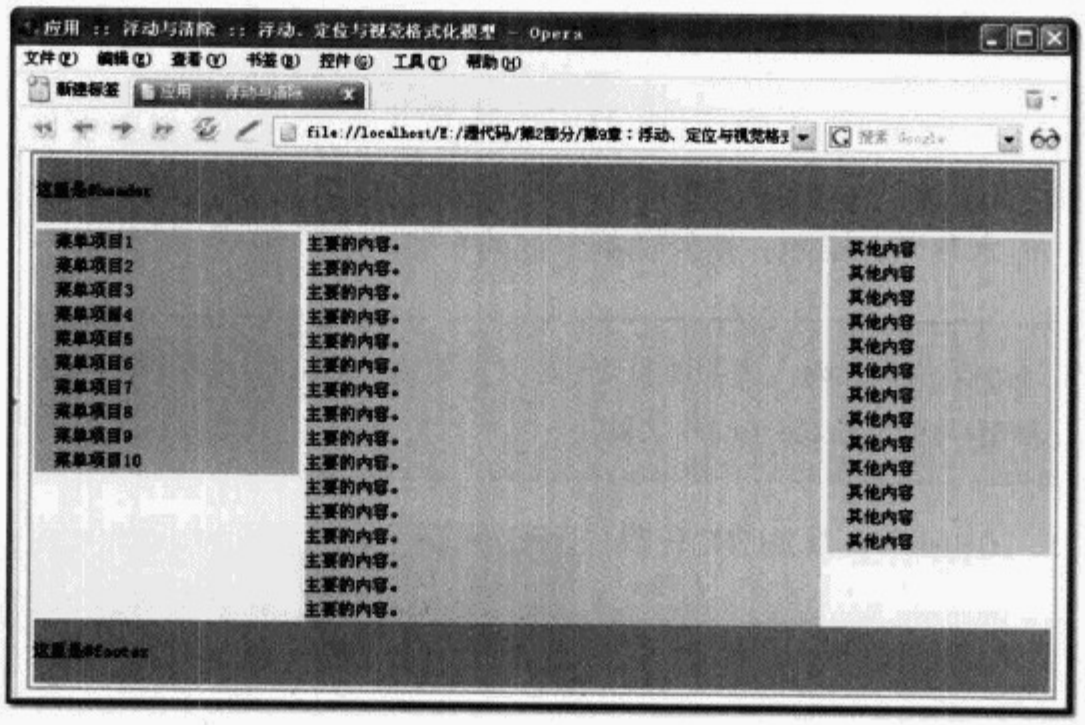
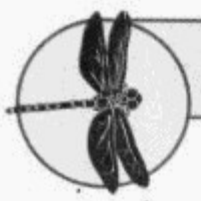


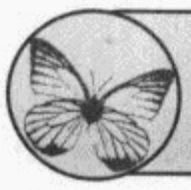
图9-110 footer层设定clear属性

由图9-103可以发现，设定clear属性后，footer层不仅位于所有浮动层之下，同时，父层wrap的高度也能包含所有子元素。但是，此时footer层和content层之间没有边距，因此需要为content层添加下边距，CSS规则如右，其显示如图9-111所示。

```
#content {
……
margin-bottom:5px;
}
```



思考： 为什么不为footer层增加上边距？参见本章［9.4.3.3 浮动与不浮动元素的边距］。



提示： 本例只是一个理想状态下的基础步骤，在实际应用中，可能还需要根据具体情况调整。

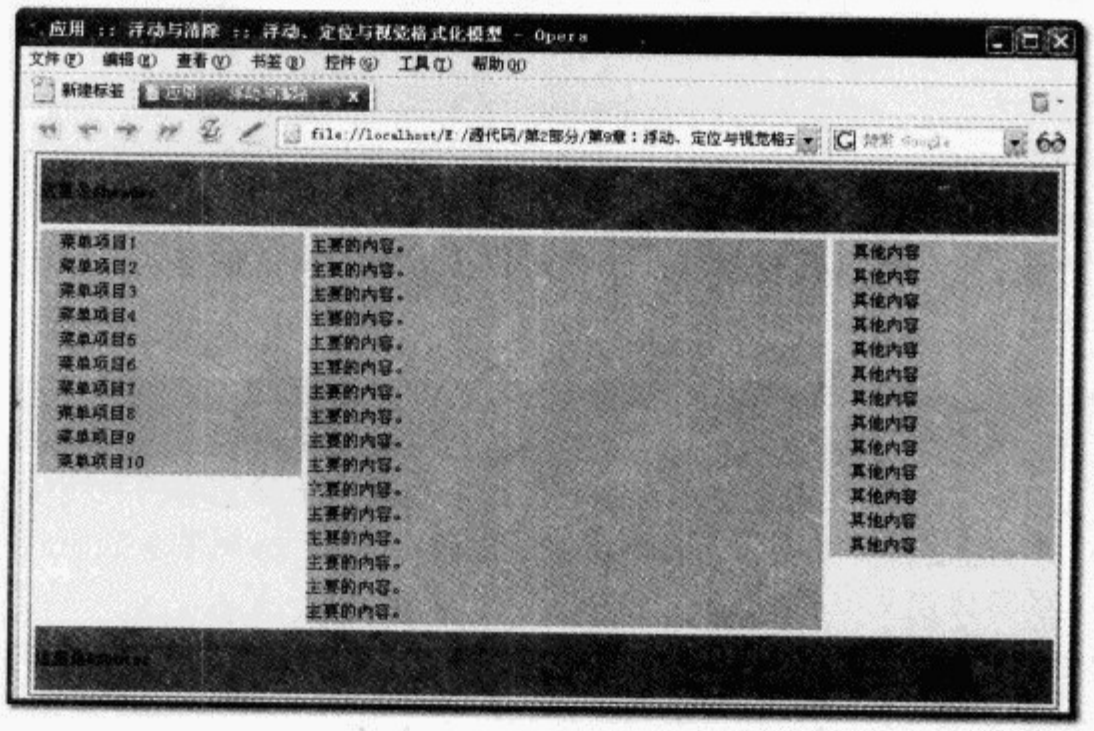


图9-111　Opera 9.2中的浏览效果

9.5 display、float和position

display、float和position这3个属性都会影响框的生成和布局，浏览器按照下列规则格式化。

（1）如果display属性的值为“none”，则忽略position和float属性，元素不生成框。

（2）否则，如果position属性的值为“absolute”或者“fixed”，则元素框绝对定位，其float属性强制为“none”，而display属性值按照下表设定。框的定位由top、right、bottom和left属性的值相对于其包含块确定。

（3）否则，如果float属性的值不是“none”，则框产生浮动，并且display属性按照下表设定。

（4）否则，如果元素是根元素，display属性按照下表设定。

（5）否则，根据display属性的设定显示元素。

指定值	计算值
inline-table	table
inline、run-in、table-row-group、table-column、table-column-group、table-header-group、table-footer-group、table-row、table-cell、table-caption、inline-block	block
其他	同指定值

也就是说，如果同时为元素指定绝对定位和浮动，则忽略浮动而对元素绝对定位，同时，元素的display属性根据上表设定。此外，如果设定了元素浮动，则元素的display属性也会自动转换，而不需要再设定display属性。

9.6 溢出和剪切

通常情况下，块的内容由框的边包围。某些情况下，一个框可能溢出，即它的内容部分或全部在框的外面显示，例如下面几种情况：

- 由于不能回行造成行框比块框宽。
- 块框比包含块宽很多，比如设定了过宽的width属性值，导致块框超出了包含块的边。

- 元素的内容高度比给其height属性设定的值要高。
- 绝对定位的框。
- 具有负边距的框。
- text-indent属性造成的悬挂缩进。

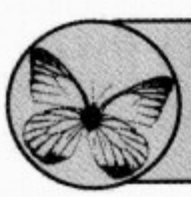
一旦有溢出产生，overflow属性指定一个框是否（以及如何）被剪切。clip属性指定了剪切区域的尺寸和形状。

9.6.1 溢出：overflow属性

在前面的章节里，曾经多次遇到过内容的宽度或者高度大于父元素的宽度或者高度，从而发生了“溢出”，而对于溢出部分如何显示，则可以通过设定overflow属性来控制。

overflow属性的具体定义列表如下：

语法	overflow : visible | hidden | scroll | auto | inherit
说明	设定当一个块类元素的内容溢出了元素的框（它作为内容的包含块）时，是否剪切。
值	visible：该值规定内容不被剪切，即它可以在块框之外得到渲染。 hidden：该值规定内容被剪切，对于剪切区域之外的内容不提供滚动浏览。 scroll：该值规定内容被剪切，并且无论内容是否溢出，都会显示滚动条。 auto：取决于用户端，但是对于溢出的框应该提供一个滚动机制。
初始值	visible
继承性	不继承
适用于	块级和替换元素
媒体	视觉
计算值	同指定值



提示：本小节示例代码，读者也可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/overflow.html］文件。

其中，各属性值详细解释如下。

1. visible

“visible”是overflow属性的初始值，当内容溢出的时候，将在框外得以显示，因此溢出的内容可能会与后面内容重叠在一起。例如下列代码，其显示如图9-112所示。

```
div {
padding : 10px;
overflow : visible;
height : 80px;
……
}
p {
width : 450px;
……
}
<div>
  <p>……</p>
</div>
<div>后面的内容。</div>
```

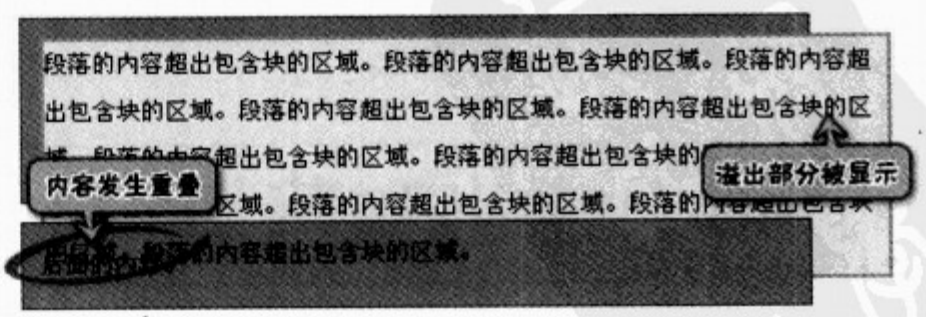


图9-112 overflow属性值为“visible”时的显示

提示： IE 6.0及更早版本会扩展元素框以包含内容。

2. hidden

“hidden”即隐藏，因此溢出的内容被剪切，而且浏览器不提供滚动条，因此用户无法访问被剪切的内容。例如下列代码，其显示如图9-113所示。

```
#overflow2 {
overflow : hidden;
height : 80px;
}
<div id="overflow2">
 <p>……</p>
</div>
```

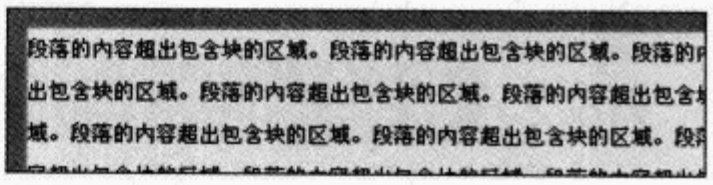


图9-113　overflow属性值为“visible”时溢出部分隐藏不显示

3. scroll

如果设定overflow属性值为“scroll”，则元素边框内会出现滚动条，无论内容是否会溢出，这避免了在动态环境中滚动条一会出现一会消失的问题。如果设定了该值而目标媒介是“print”或“projection”，溢出部分的内容应该被打印。例如下列代码，其显示如图9-114所示。

```
#overflow3 {
overflow : scroll;
height : 80px;
}
<div id="overflow3">
 <p>……</p>
</div>
```

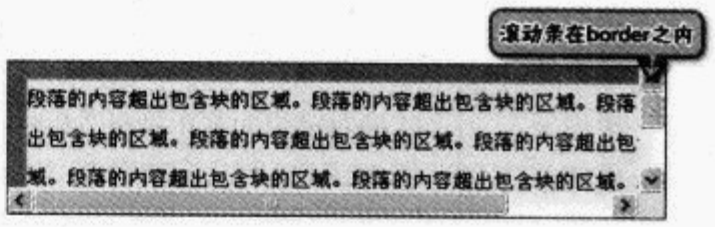
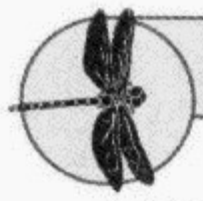


图9-114　overflow属性值为“scroll”时元素有滚动条



注意： 滚动条是由用户端生成，到CSS 2.1为止，还不能通过CSS控制其样式。

如果设定overflow属性为“scroll”，则即使内容没有溢出，滚动条也一直存在，如图9-115所示。

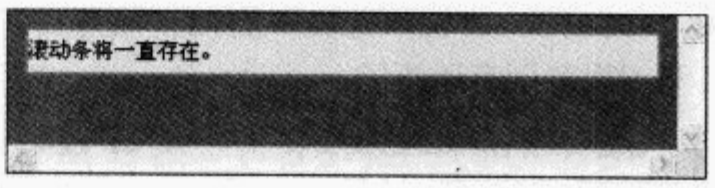


图9-115　overflow属性值为“scroll”时滚动条始终存在

4. auto

overflow属性为“auto”时，将根据实际发生溢出的情况以决定是否有滚动。例如下列代码，其显示如图9-116所示。

```
#overflow5 {
overflow : auto;
height : 80px;
}
<div id="overflow5">
 <p>……</p>
</div>
```

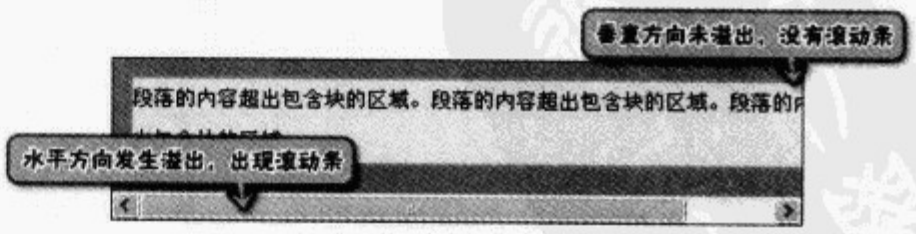


图9-116　overflow属性值为“auto”时滚动条视情况而定

此时，如果不定义 的高度，例如下列代码：

```
p {
width : 450px;
……
}
#overflow6 {
```

```
overflow : auto;
}
<div id="overflow6">
  <p>……</p>
</div>
```

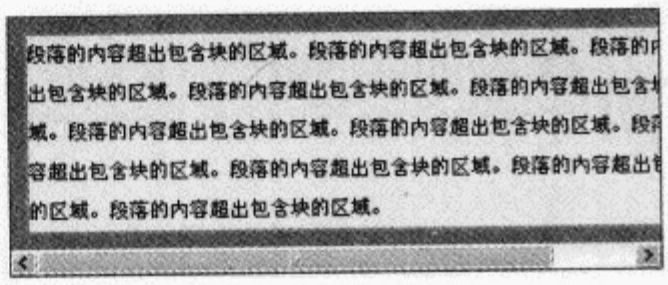


图9-117 overflow属性值为“auto”时不设定高度的情况

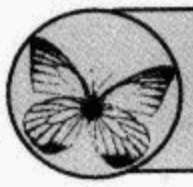
由于<p>元素的宽度大于 的内容宽度，因此发生溢出，但是 并未定义高度，即其高度由内容高度决定，因此只有横向的滚动条，而高度会适应内容的高度，其显示如图9-117所示。

9.6.2 剪切：clip属性

一个剪切区域定义了元素经渲染的内容的哪一部分是可见的。默认情况下，剪切区域和元素框的尺寸和形状是一样的。不过，剪切区域可以由clip属性改变。

clip属性的具体定义列表如下：

语法	clip: rect(<上>,<右>,<下>,<左>) | auto | inherit
说明	设定元素经渲染的内容的哪一部分是可见的
值	auto：元素不剪切。 rect(<上>,<右>,<下>,<左>)：4个方向的偏移量。
初始值	auto
继承性	不继承
适用于	绝对定位元素（参见［9.3.4 绝对定位］）
媒体	视觉
计算值	如果指定长方形的值，则为4个计算的长度；否则同指定值。



提示：本小节示例代码，读者也可参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/clip.html］文件。

“auto”值表示元素不剪切，按照其正常范围显示。而对于设定剪切范围，CSS 2.1规范和CSS 2规范则有所不同。

1. CSS 2规范与CSS 2.1规范的不同点

clip属性在CSS 2.1中进行了修改，其与CSS 2规范的不同有以下几点：

（1）适用元素的不同。CSS 2中，clip属性适用于块类和替换元素，且overflow属性不为“visible”的元素。而CSS 2.1中，clip属性适用于绝对定位的元素。

（2）偏移量的计算方法不同。

```
rect:(<上>,<右>,<下>,<左>)
```

在CSS 2中，<上>、<右>、<下>、<左>的偏移量定义同绝对定位的偏移量定义相同，即定位元素的边到相应方向的边的距离。而CSS 2.1中，<上>和<下>指定了从框的上边框边计算的偏移；<左>和<右>及<left>指定了从框的左边框边（文本方向为左到右）计算的偏移量。偏移量的定义如图9-118所示。

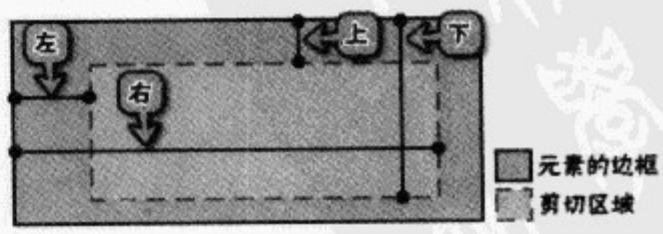


图9-118 CSS 2.1中偏移量的计算示意

例如下列代码，其显示如图9-119所示。

```
#clip1 {
```

```
height:80px;
position:relative;
……
}
#clip1 p {
position:absolute; /* 绝对定位的元素才能设定剪切 */
clip:rect(10px 300px 50px 30px);
……
}
<div id="clip1">
 <p>……</p>
</div>
```

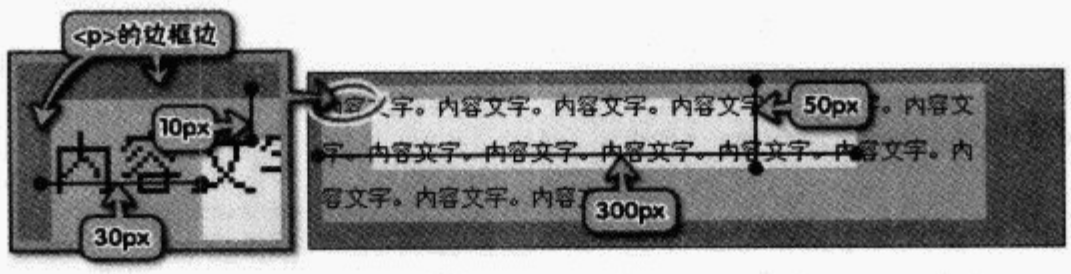


图9-119 偏移量的计算

因此，元素实际的显示区域：

```
宽度=右偏移量-左偏移量
高度=下偏移量-上偏移量
```

如果其中1个得出的值为0或小于0，则元素没有可显示区域。

（3）偏移量的分隔符号不同。在CSS 2.1中，4个偏移量中间使用英文逗号“,”分隔，而在CSS 2中，则使用英文空格分隔，而某些浏览器（例如IE）则只支持使用英文空格分隔的偏移量，如右所示：

```
p { clip:rect(5px, 300px, 40px, 5px); } /* CSS 2.1 */
p { clip:rect(5px  300px  40px  5px); } /* CSS 2 */
```

2. 偏移量的“auto”值

偏移量的值可以是“auto”，表示该剪切方向的边等同于元素生成的边框边，对于“上”和“左”的偏移量（文字方向为左向右的情况），“auto”值等于0，例如下列代码，其显示如图9-120所示。

```
#clip2 {
height : 80px;
position : relative;
……
}
#clip2 p {
position : absolute; /* 绝对定位的元素才能设定剪切 */
clip:rect(auto 300px 50px auto);
……
}
<div id="clip2">
 <p>……</p>
</div>
```

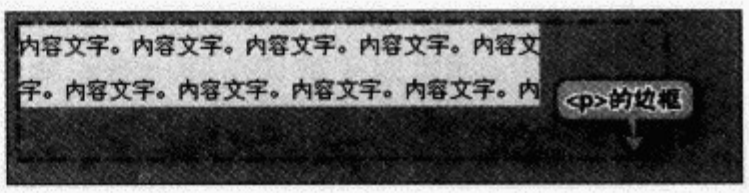


图9-120 “上”和“左”偏移量为“auto”值

而对于“下”偏移量，“auto”值相当于元素的height属性值、垂直方向补白及边框宽度的和（如果元素overflow属性为“scroll”，则还有滚动条的高度）。而对于“右”偏移量（文本方向为左向右），“auto”值相当于width属性值、水平方向补白及边框宽度的和（如果元素overflow属性为“scroll”，则还有滚动条的宽度）。例如，修改上例中<p>的偏移量如下，则其显示如图9-121所示。

```
p { clip:rect(10px auto auto 20px); }
```

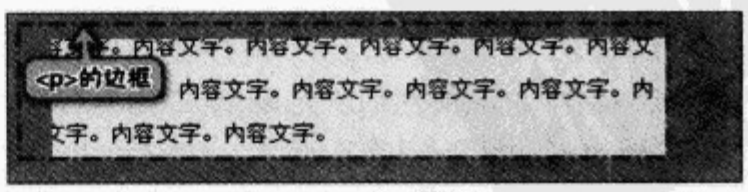


图9-121 “下”和“右”偏移量为“auto”值

3. 偏移量的负值

偏移量允许为负值，但是，如果右偏移量和下偏移量为负值，则元素将没有显示区域。而左偏移量和上偏移量为负值，例如下列代码：

```
#clip4 p {
……
clip:rect(0 300px 50px 0);
}
#clip4 span {
position : absolute;
top:-10px;
left:-15px;
……
}
<div id="clip4">
 <p><span>p的子元素</span>……</p>
</div>
```

<p>内绝对定位的元素的偏移量为负值，因此其显示如图9-122所示。

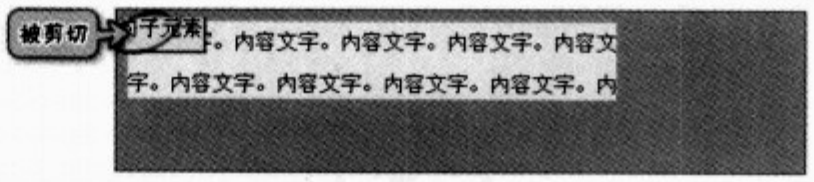


图9-122 绝对定位在剪切范围外的子元素被剪切

而如果修改<p>元素的剪切范围：

```
p { clip:rect(-10px 300px 50px -20px); }
```

则其显示如图9-123所示。

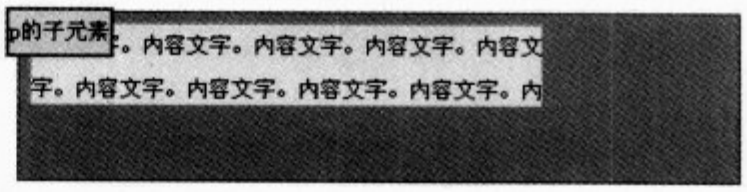


图9-123 clip属性偏移量为负值

如果剪切区域超出了用户端文档窗口的边界，操作系统可能将内容剪切到该窗口。

9.6.3 clip与overflow属性的关系

如果元素设定了clip属性，则在显示范围外的内容都将被剪切，包括内容、后代元素、背景、边框以及滚动条等。例如下列代码，其显示如图9-124所示。

```
#clip6 p {
overflow : scroll;
clip:rect(10px auto auto 20px);
……
}
<div id="clip6">
 <p>……</p>
</div>
```

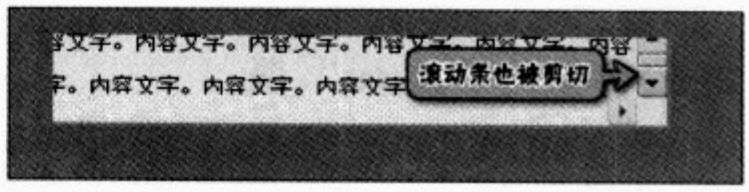


图9-124 设定clip属性的元素内所有的内容都可能在剪切范围之内

而overflow属性对于剪切范围也有影响，例如下列代码：

```
p {
overflow : scroll;
clip:rect(-10px 300px 50px -20px);
……
}
span {
position:absolute;
top:-10px;
left:-15px;
……
}
<div>
 <p><span>p的子元素</span>……</p>
</div>
```

这是前一节［9.5.2.3 偏移量的负值］中的例子，此时，如果对<p>元素增加CSS规则如下，则其显示如图9-125所示。实际上，只要overflow属性的值不是"visible"，都会发生剪切。

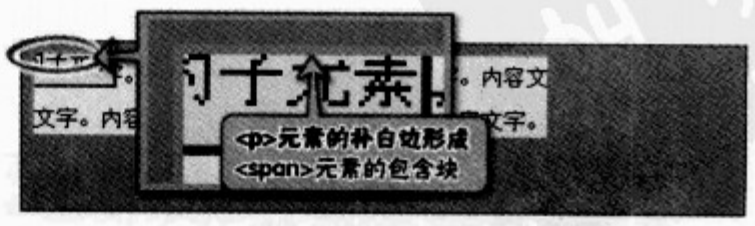


图9-125 overflow属性对于剪切的影响

```
p { overflow : scroll; }
```

9.7 可视性：visibility属性

“可视性”visibility属性指定一个元素生成的框是否被渲染，具体定义列表如下。

语法	visibility: visible | hidden | collapse | inherit
说明	设定元素是否可视
值	visible：元素可视。 hidden：元素隐藏。 collapse：主要用来隐藏表格的行或列
初始值	visible
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

9.7.1 属性值详解

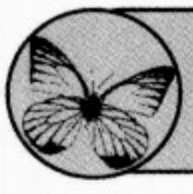
在［9.2 显示类型：display属性］一节内介绍过，display属性为“none”时，元素将不生成框，因此也不会显示，更不会占位。而如果设定“visibility : hidden”则会生成元素框，只是元素“不可视”，而其他非视觉的属性都将生效，如widht、padding等。

例如下列XHTML代码，当不设定visibility属性，即visibility属性为初始值时，其显示如图9-126所示。

普通的段落文字
段落的内容，strong子元素。
普通的段落文字

图9-126　不设定visibility属性的时候示例的显示

```
div { …… }
p { …… }
strong { …… }
.sample1 { height: 40px; }
<div>
  <p>普通的段落文字</p>
  <p class="sample1">段落的内容，<strong>strong子元素</strong>。</p>
  <p>普通的段落文字</p>
</div>
```



提示：本小节示例代码，读者可以参见下载文件包内［/第2部分/第9章：浮动、定位与视觉格式化模型/ visibility.html］文件。

如果设定如下的CSS规则，则其显示如图9-127所示。

```
.sample1 { display : none; }
```

普通的段落文字
普通的段落文字

图9-127 “sample1”的display属性为“none”

由图9-127可以发现，“sample1”没有占位。而如果不设定display属性，而设定visibility属性如下，则其显示如图9-128所示。

```
.sample1 { visibility : hidden; }
```

由图9-128可以发现，“sample1”虽然没有显示出来，但是仍占有布局位置。visibility属性的“collapse”值应用在表格的行、行组、列以及列组上，主要用来隐藏表格的行或列，隐藏的行或列能够被其他内容使用。而对于表格的其他元素，其作用和“hidden”类似。

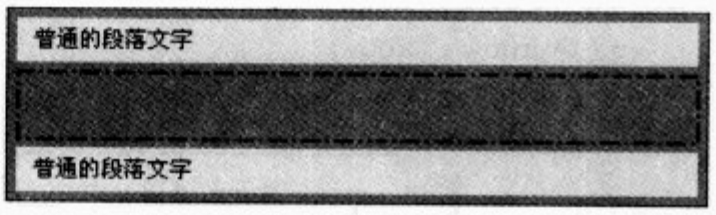


图9-128 “sample1”的visibility属性为“hidden”

例如下列代码，其显示如图9-129所示。

```
table { …… }
td { …… }
.sample2 { visibility : collapse; }
<table>
 <tr>
  <td>1-1</th>
  <td>1-2</td>
  <td>1-3</td>
 </tr>
 <tr class="sample2">
  <td>2-1</th>
  <td>2-2</td>
  <td>2-3</td>
 </tr>
 <tr>
  <td>3-1</th>
  <td>3-2</td>
  <td>3-3</td>
 </tr>
</table>
```

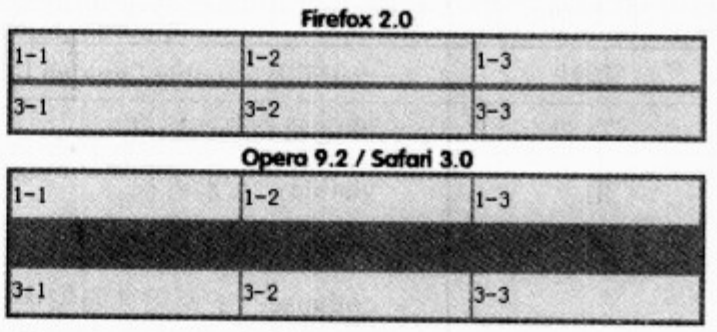


图9-129 不同浏览器对visibility属性的“collapse”值得不同显示方式

由图9-129可以发现，不同浏览器对于visibility属性的“collapse”值有不同的表现，这两种方式CSS 2.1规范并没有严格规定。IE 7.0及更早版本不支持此值。

9.7.2 应用：显示及隐藏元素

在［9.2.2 应用：显示或隐藏元素］一节内，介绍了利用display属性来实现元素的显示和隐藏，但是，这样做的问题在于，“display : none”的元素不会占位，因此当其显示的时候，它后面的元素会发生移动，以便给它空间，而当它隐藏的时候，后面的元素又会来占据它的位置。而在某些设计中，并不希望发生这样的移动，因此可以使用“visibility : hidden”来实现显示或隐藏元素。例如下列代码，其显示如图9-130所示。

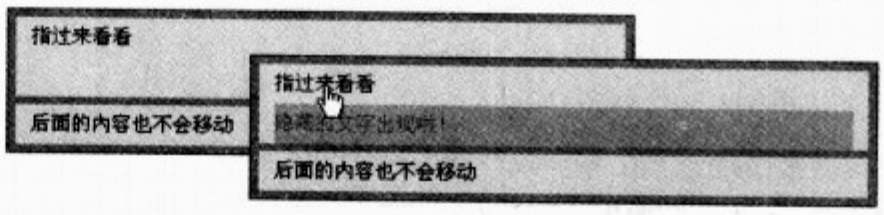
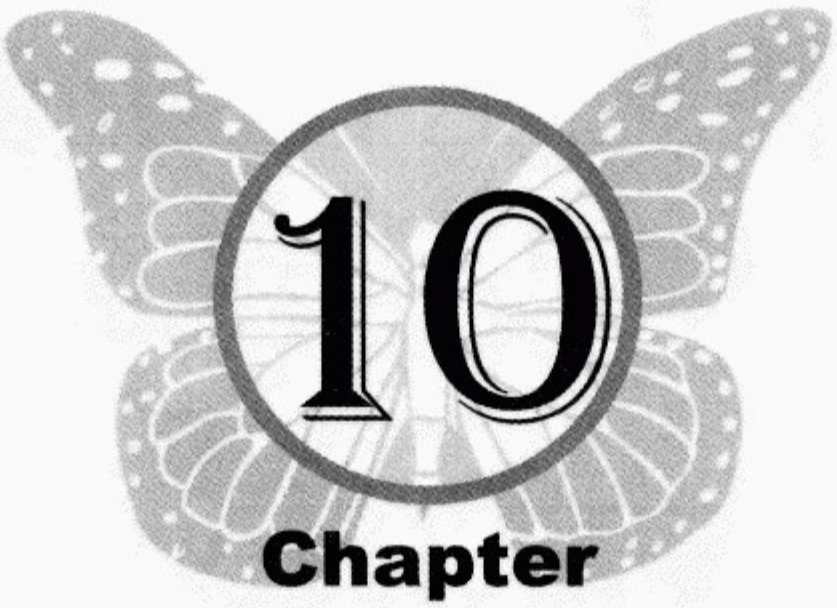


图9-130 利用visibility属性显示及隐藏元素

```
#visibility5 a {
……
}
#visibility5 a span{
visibility:hidden;
……
}
#visibility5 a:hover {
border:0; /* 使Windows IE 6.0 的:hover能正常触发 */
}
#visibility5 a:hover span{
visibility: visible;
}
<div id="visibility5">
 <p><a href="#" title="鼠标指向显示文字">指过来看看<span>隐藏的文字出现啦！ </span></a></p>
 <p>后面的内容也不会移动</p>
</div>
```



第10章 颜色与背景

当一个页面呈现在访问者眼前，颜色和图片往往是最先给访问者深刻印象的东西。通过CSS，可以设置元素的前景色、背景色以及背景图片，还可以设置背景图片的重复方式。灵活利用背景图片是实现网页表现的重要手段。

10.1 颜色基础

如果将浏览器的窗口看作一块画板，那么背景色（Background Color）就是画布的颜色，而前景色（Foreground Color）就是画笔的颜色。而浏览器对于一个元素的渲染，可以分成几层，如图10-1所示。

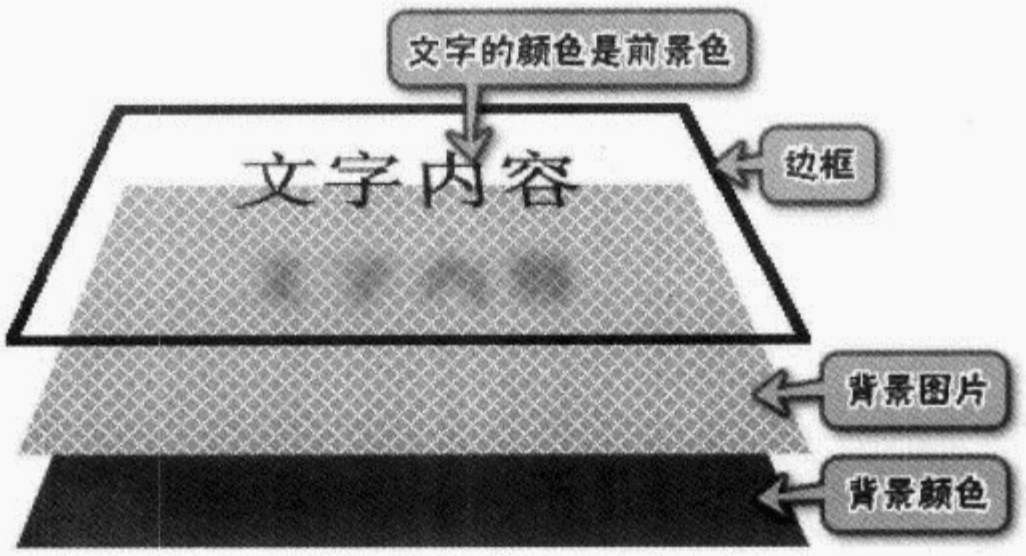


图10-1 元素的分层渲染

由图10-1可以发现，背景色在最下层，而文字在最上层。

色彩可分为无彩色和有彩色两大类。无彩色有明有暗，表现为白、黑、灰。有彩色表现很复杂，如红、黄、蓝等，但可以用3组特征值来确定：其一是彩调（Hue），也就是色相；其二是明暗，也就是明度（Value）、亮度；其三是纯度、彩度（Chroma）。“色相”、“明度”和“纯度”称为色彩的三属性。明度和色相合并为二线的色状态，称为色调，如图10-2所示。

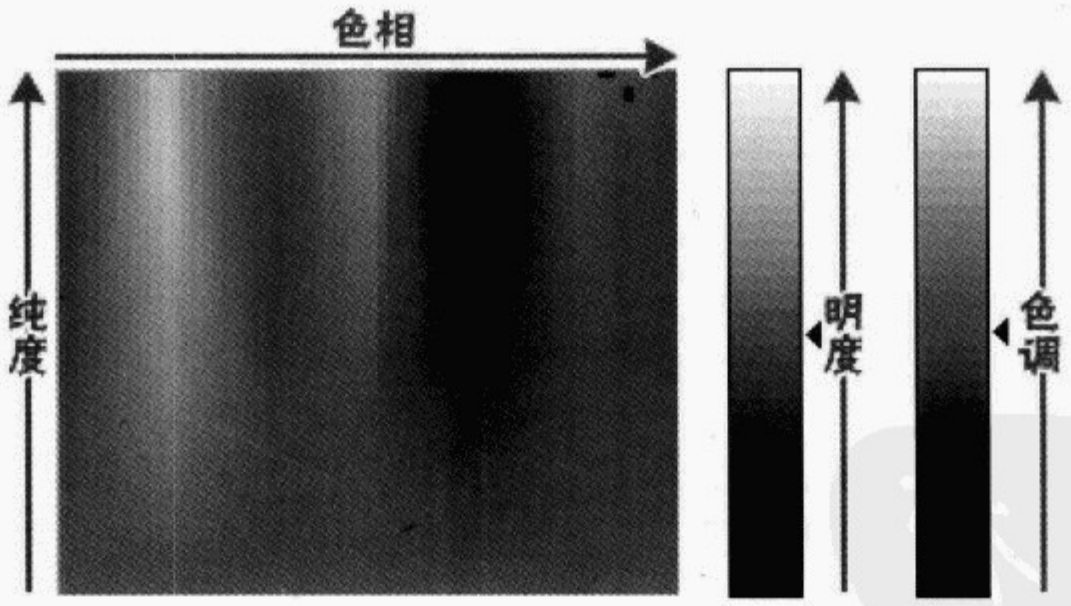


图10-2 色彩的三属性

表示颜色的方法有很多种，而将色相按照一定的变化排成环形，叫做色相环，这种表示色相的方法比较直观，而根据不同的划分方法，又分为10色相环、12色相环、24色相环等。同时，也有按照作者名字命名的色相环，如图10-3所示，就是伊登12色相环。

伊登12色相环其中间为3原色（primary colors）——红（red）、黄（yellow）和蓝（blue）。3原色两两叠加，形成“次色（secondary hues）”，即：橙（红+黄）、绿（黄+蓝）和紫（蓝+红）。而次色再次和原色叠加，则形成“间色（intermediate hues）”，即：红橙、橙黄、黄绿、蓝绿、蓝紫、红紫。

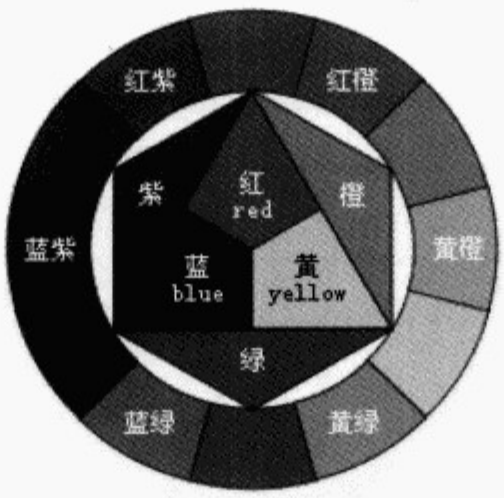
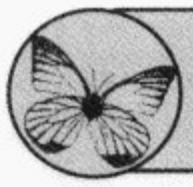


图10-3　伊登12色相环



提示：色的原色和色光不同，色光的原色为“红绿蓝”。更多关于配色设计的知识，可以参考http://design.yesky.com/homepage/43/ 2538543.shtml。

在色相环上，相邻的颜色为类似色，而相对的颜色为补色（或称对比色），例如红色和红橙色、红紫色为类似色，而绿色为补色。

配色时，使用相似色会使整体感比较协调，而使用补色则可以起到突出重点的作用。同时还要注意前景色和背景色的对比关系，如果前景色和背景色非常相近，那么无疑会使访问者阅读起来非常困难；但是如果对比过于强烈，又容易造成视觉疲劳，如图10-4所示。

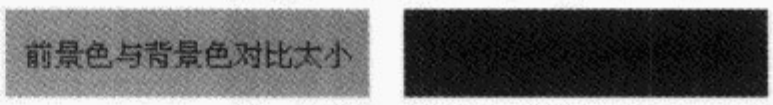


图10-4　前景色和背景色的对比关系

高纯度、高亮度的颜色（如黄色、红色、蓝色等）一般可以作为突出色使用，起到画龙点睛的作用，而不适合大面积地作为背景色来使用。

10.2 前景色：color属性

在大多数情况下，前景色就是文字的颜色。color属性具体定义列表如下：

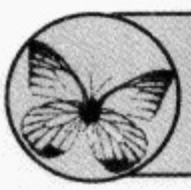
语法	color: <颜色> | inherit
说明	设置元素的前景色
值	颜色：合法的颜色值，可参见本书［5.1 颜色］一节
初始值	取决于用户端
继承性	继承
适用于	所有元素
媒体	视觉
计算值	同指定值

对于非替换元素，color属性设置的就是元素内文字的颜色，且颜色值是继承的，不过浏览器的默认设置可能会覆盖继承的值，例如下列代码其显示如图10-5所示。

```
#color1 { color : #C30; }
<div id="color1">
    <p>段落文字，<em>段落内的em</em>，<a href="http://www.ddcat.net/" title="访问猫窝">访问猫窝
</a></p>
</div>
```

段落文字，段落内的em，访问猫窝

图10-5 颜色的继承

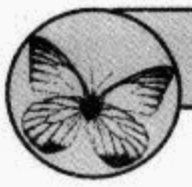


提示：本小节示例代码，读者可以参见下载文件包内［/第2部分/第10章：颜色与背景/color.html］文件。

由图10-5可以发现，<p>元素继承了父元素“#color1”的color属性值，而元素继承了<p>元素的color属性值，但是链接文字的颜色为浏览器默认的值。

10.2.1 链接

为了突出显示带有链接的文字，一般会为链接设置不同于普通文字的颜色，同时利用链接伪类（:link和:visited）和动态伪类（:hover、:active和:focus）来使链接更美观，也更利于访问者辨认。同时，还可以为不同功能的链接（比如导航链接和普通链接）设置不同的颜色值来加以区分，例如下列代码，其显示如图10-6所示。



提示：关于伪类，请参见本书［4.3.1 伪类（Pseudo-Classes）］一节。

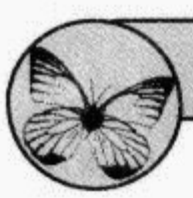
```
/* 设置div"color2"内的链接样式 */
#color2 { color:#666; }
#color2 a:link{ color:#06f; }
#color2 a:visited { color:#960;}
#color2 a:hover { color:#F00; }
/* 设置 "color2"内的li的链接样式 */
#color2 li a:link { color:#063; }
#color2 li a:visited { color:#699; }
#color2 li a:hover { color:#C00; }
/* 设置"color2"内的li内的"here"类链接样式 */
#color2 li a.here { color:#F00; }
<div id="color2">
 <p>段落内的<a href="http://www.ddcat.net/" title="访问猫窝">链接文字</a></p>
 <ul>
  <li>列表内的<a href="http://www.ddcat.net/blog/" title="访问猫窝博客">猫窝博客</a></li>
    <li>列表内的<a href="http://www.ddcat.net/bbs2007/" title="访问猫窝论坛"class="here">猫窝论坛
</a></li>
  <li>列表内的<a href="http://www.ddcat.net/" title="访问猫窝">访问猫窝</a></li>
 </ul>
</div>
```

段落内的链接文字

- 列表内的猫窝博客
- 列表内的猫窝论坛
- 列表内的访问猫窝

图10-6 多个链接样式的定义

定义多个链接样式的时候，特别要注意选择器的层叠与特殊性，例如在本例中，首先定义的是“color2”内的链接样式，然后针对“color2”内的<li>元素内的链接设置样式，最后定义的“#color2 li a.here”，则是针对<li>内的class属性为“here”的<a>元素，由于“#color2 li a.here”具有最高的特殊性，因此它将覆盖掉前面的定义，包括伪类。



提示：关于选择器的层叠与特殊性，请参见本书［4.6层叠］一节。

10.2.2 边框

在本书［8.8.1 边框颜色］一节内介绍过，如果不设置border-color属性，则将以元素的前景色即color属性的值为边框的颜色值。例如右边的两个规则是等价的：

```
p {
color: red;
border: 2px solid;
}
p {
border: 2px solid red;
}
```

对<img>元素设置color属性并不能改变图片本身的颜色，但是由于color属性可以影响边框，因此color属性可以用来设定图片的边框颜色，例如下列代码，其显示如图10-7所示。

```
img {
color:#F60;
border:2px solid;
}
 <p> <img src="../../img/ddcat_96.jpg" alt="示例图片" width="96" height="96" /></p>
```

图10-7　图片的边框

10.2.3 表单元素

对于表单内的替换元素，并不会继承表单<form>元素的color属性，例如下列代码，其显示如图10-8所示。

```
form {
color :  #C30;
……
}
<form id="testForm" action="#">
 <fieldset>
    <legend>示例表单</legend>
    <p><label for="test1">姓名：</label>
    <input name="test1" id="test1" /></p>
    <p><input id="sex1" name="sex" type="radio" value="m" /><label for="sex1">男</label>
      <input id="sex2" name="sex" type="radio" value="f" /><label for="sex2">女</label></p>
    <p><label for="test3">城市：</label>
     <select name="test3" id="test3">
     <option>选择1</option>
     <option>选择2</option>
     <option>选择3</option>
     </select>
      </p>
      <p><label for="test4">留言：</label>
      <textarea name="test4" cols="30" rows="3" id="test5"></textarea></p>
 </fieldset>
</form>
```

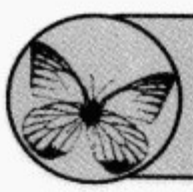
因此，要改变其内文字的颜色，需要单独定义，例如：

```
input { color : #09F; }
```


图10-8　表单内的替换元素不继承祖先元素的color属性

10.3 背景

在本书［2.1 理解结构与表现］一节内介绍过，要将结构与表现分离开来，对于设计图中属于装饰类而不是内容的图片，都不应该出现在(X)HTML的内容中，而应该通过设置元素背景图片的方式来实现。因此，设置与背景有关的若干属性是利用率非常高的属性。



提示：本小节示例代码，可以参见下载文件包内［/第2部分/第10章：颜色与背景/background.html］文件。

10.3.1 背景颜色：background-color属性

背景颜色由background-color属性设置，其具体定义列表如下：

语法	background-color : <颜色> | transparent | inherit
说明	设置元素的背景颜色
值	颜色：合法的颜色值，请参见本书［5.1 颜色］一节。 transparent：透明
初始值	transparent
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	同指定值

通过CSS可以设定元素的背景颜色和背景图片，CSS 2.1中，元素的背景是指包括内容、补白和边框在内的区域，而CSS 2则只是指内容和补白所在的区域。而边距是透明的，因此可以显示出祖先元素的背景。例如下列代码，其显示如图10-9所示。

```
#background1 {
background-color : #CF9;
border : 5px dashed #090;
……
}
<div id="background1">
  <p>段落文字，<em>段落内的em</em></p>
</div>
```

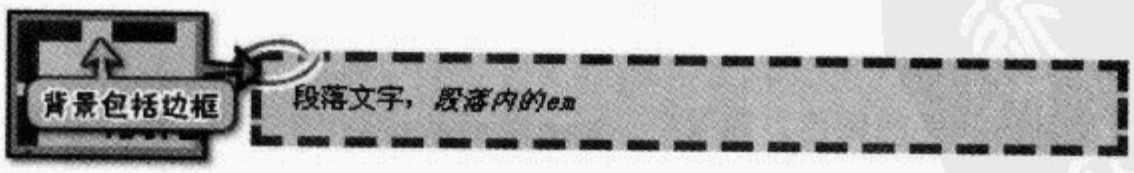


图10-9 背景所指的范围包括边框、补白和内容区域

而IE 7.0及更早的版本则仍是以补白框作为背景的显示区域，如图10-10所示。

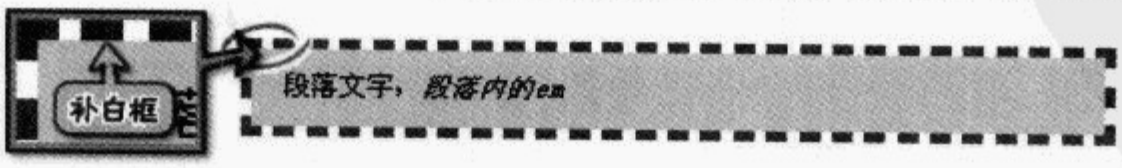


图10-10 IE 7.0及更早的版本以补白框作为背景的显示区域

背景颜色的初始值为“transparent”即“透明”，因此可以显示出祖先元素的背景颜色。例如，在图10-9中，<p>元素和元素都没有设置背景颜色，因此可以看到其祖先元素 的背景颜色。

行内元素的背景区域为元素的内容区域，不过可以通过设置padding属性来扩大其背景区域，例如下列代码，其显示如图10-11所示。

```
#background2 em {
background-color : #CFF;
border : 2px dashed #06C;
padding : 5px;
}
#background2 strong {
background-color : #FC3;
}
<div id="background2">
 <p>段落文字，<strong>行内strong元素</strong>，<em>行内em元素</em></p>
</div>
```

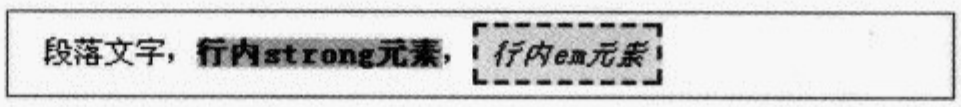


图10-11　行内元素的背景区域

由图10-11可以发现，元素的背景区域被padding和border属性扩大了。

10.3.2 背景图片：background-image属性

丰富多彩的图片让网页具有赏心悦目的外观，但是，作为装饰性的图片，则应该从内容中分离出来，而通过设置元素的背景图片来使其在页面内显示出来。

背景图片可以通过background-image属性设置，其具体定义列表如下：

语法	background-image : <uri> | none | inherit
说明	设置元素的背景图片
值	uri：图片的链接地址，请参见本书［5.7 URL + URN = URI］一节。 none：无图片
初始值	none
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	绝对的URI或者为“none”

background-image属性指定了元素的背景图片。设置背景图片时，制作者也应该同时设置一个背景色以考虑背景图片不可用的情况。如果背景图片可用，它在背景色之上得到渲染，如图10-1所示，因此，在图形的透明区域，背景色是可见的。例如下列代码：

```
#background3 p{
background-color : #FC6;
background-image : url(../../img/ddcat2.gif);
}
#background3 .sample1 {
background-color : #6CF;
}
<div id="background3">
   <p>背景图片在背景颜色之上</p>
   <p class="sample1">背景图片在背景颜色之上</p>
</div>
```

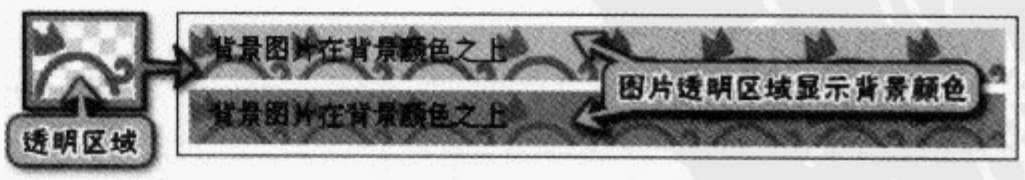


图10-12　透明背景图片与背景颜色

两个<p>元素统一设定了背景图片，但是有不同背景颜色，由于背景图片是一个透明的GIF图片，因此其显示如图10-12所示。

提示： GIF和PNG格式的图片都可以实现透明效果，其中GIF格式的图片只能实现单通道透明，PNG格式的文件则可以设置不同透明度（比如从透明到不透明的渐变），但是，IE 6.0及更早的版本，不支持透明度的PNG图片。

对于行内元素，背景图片默认也是在其内容区域内，如图10-13所示。

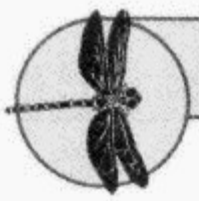
图10-13 行内元素的背景图片显示区域

10.3.3 背景图片重复：background-repeat属性

由图10-12可以发现，背景图片的内在尺寸比元素框小，因此会在元素的背景区域内重复显示，可以通过background-repeat属性来控制是否重复显示背景图片及重复的方式。

background-repeat属性详细定义列表如下：

语法	background-repeat : repeat | repeat-x | repeat-y | no-repeat | inherit
说明	设置元素的背景图片重复的方式
值	repeat：背景图片在水平和垂直方向都重复显示。 repeat-x：背景图片只在水平方向重复显示。 repeat-y：背景图片只在垂直方向重复显示。 no-repeat：背景图片不重复，只显示1次
初始值	repeat
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	同指定值



注意： background-repeat属性只针对背景图片，而对背景颜色无效。

只有在设定了背景图片的时候，background-repeat属性才会生效，例如下列代码，其显示如图10-14所示。

```
#background5 {
background-color : #FC3;
}
#background5 p{
background-color : #CFF;
height : 80px;
background-image : url(../../img/ddcat2.gif);
}
#background5 .sample2 {
background-repeat : repeat-x;
}
#background5 .sample3 {
background-repeat : repeat-y;
}
#background5 .sample4 {
background-repeat : no-repeat;
}
```

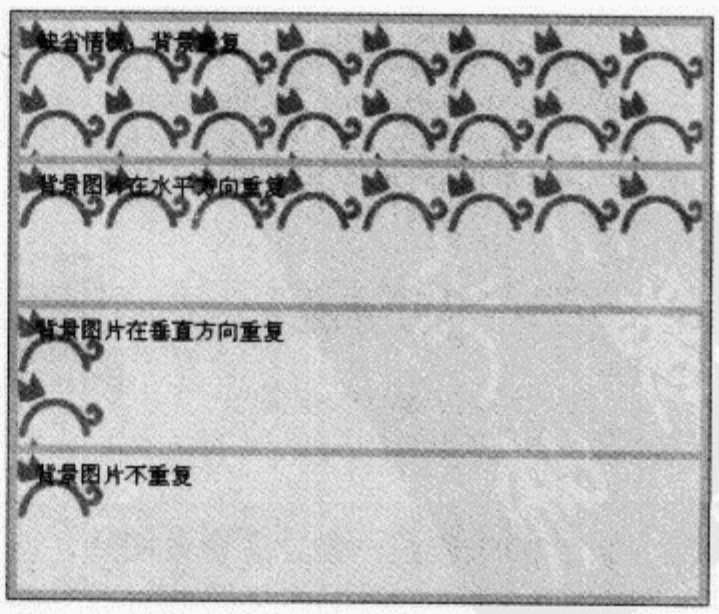


图10-14 background-repeat属性各值的表现

```
<div id="background4">
 <p>缺省情况，背景重复</p>
 <p class="sample2">背景图片在水平方向重复</p>
 <p class="sample3">背景图片在垂直方向重复</p>
 <p class="sample4">背景图片不重复</p>
</div>
```

10.3.4 背景图片附属：background-attachment属性

background-attachment属性来控制背景图片是否会随元素的滚动条而滚动，其详细定义列表如下：

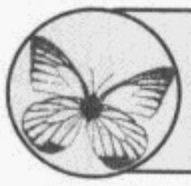
语法	background-attachment : scroll | fixed | inherit
说明	设置元素的背景图片的显示方式
值	scroll：背景图片随滚动条滚动。 fixed：背景图片相对于视口固定
初始值	scroll
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	同指定值

本属性同background-repeat属性一样只当指定了背景图片时才会生效。

如果设定元素的background-attachment属性为“fixed”，背景图片会相对于视口定位，且不滚动，无论是滚动整个页面，还是元素内的滚动条，其背景图片都会固定不动。例如下列代码：

```
#attachment {
height : 100px;
overflow : auto;
background-color : #CF9;
background-image : url(../../img/ddcat1.gif);
background-attachment : fixed;
}
```

```
<body>
 ……
 <div id="attachment">
 ……
 </div>
……
</body>
```



提示：本小节示例代码，读者可以参见下载文件包内［/第2部分/第10章：颜色与背景/background-attachment.html］文件。

此时，虽然是对attachment层设定的背景图片，但是，其背景的定位实际上以视口的范围为准，如图10-15所示。

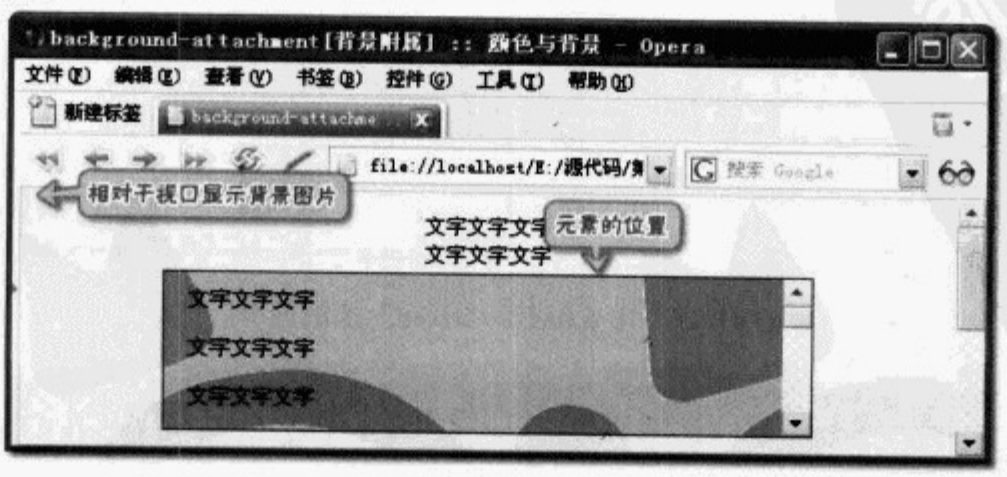


图10-15　background-attachment属性值为“fixed”时背景的位置以视口为准

因此如果滚动元素内部的滚动条，背景图片不会移动，如图10-16所示。

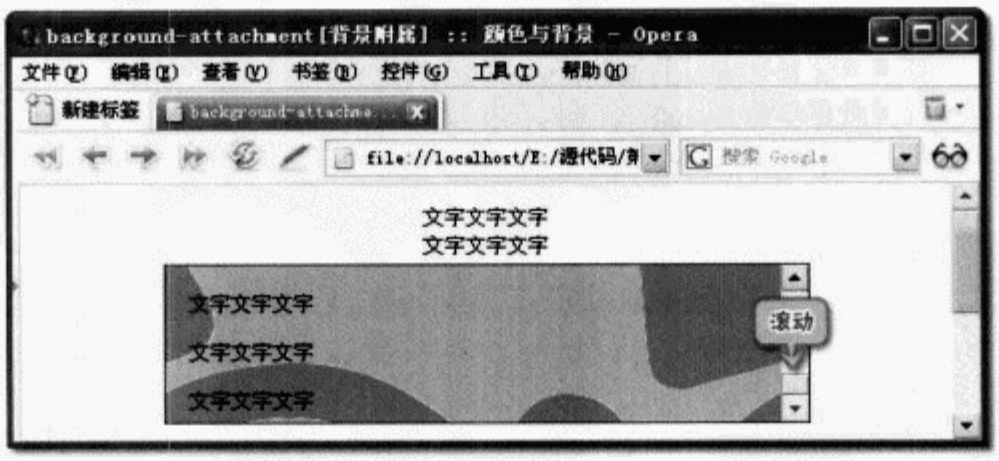


图10-16 background-attachment属性值为“fixed”时背景图片的位置以视口为准

由图10-16可以发现，attachment层显示出来的背景的范围，仍然是本元素的背景区域（边框、补白和内容区域），因此如果滚动窗口的滚动条，其显示如图10-17所示。

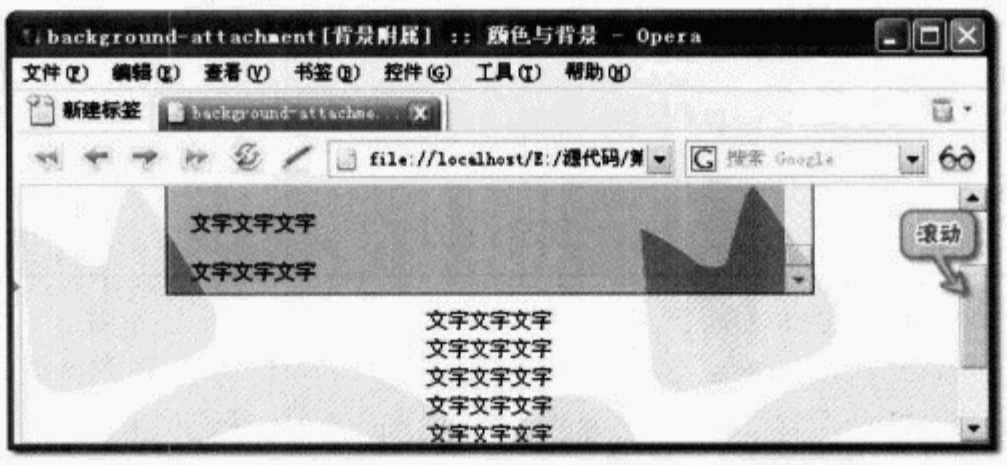


图10-17 background-attachment属性值为“fixed”时背景图片不随滚动条滚动

此时由于背景图片的定位相对于视口，因此，有些情况下（例如元素的位置在背景图片尺寸范围外）需要设定背景图片重复才可能看到背景图片。

如果浏览器不支持background-attachment属性的“fixed”值，其显示效果同“scroll”一样。而IE 6.0对于“fixed”值的理解与规范不同，背景图片并不是相对于视口固定，而是相对于元素固定，如图10-18所示。

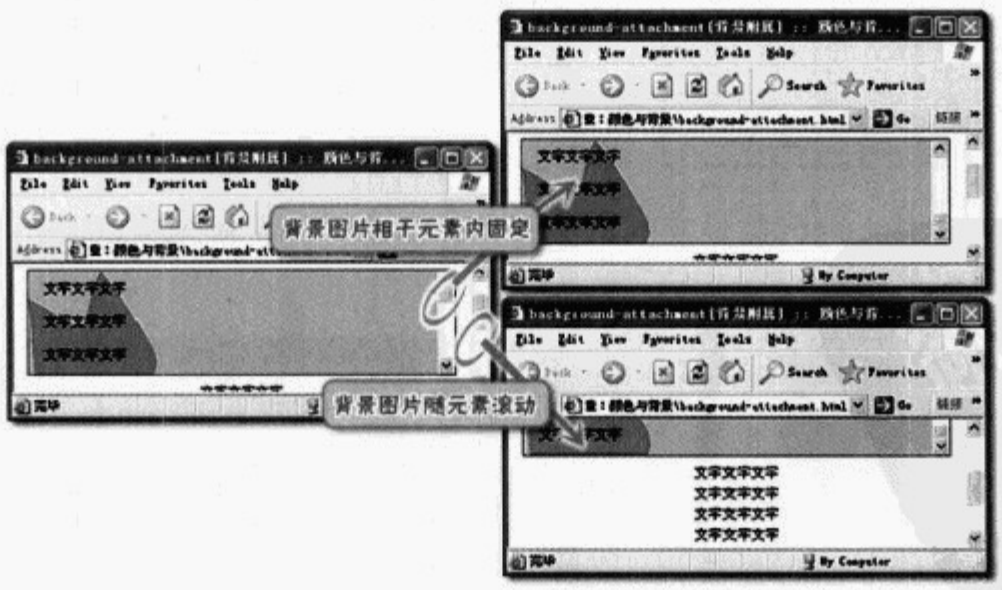


图10-18 IE 6.0对于“fixed”值的理解

10.3.5 背景图片定位：background-position属性

background-position属性来控制背景图片在元素内的起始位置，其详细定义列表如下：

语法	background-position : [[<百分比> | <长度> | left | center | right] [<百分比> | <长度> | top | center | bottom]?] | [[left | center | right] || [top | center | bottom]] | inherit
说明	设置元素的背景图片的起始位置
值	top：相当于垂直方向为0。 right：相当于水平方向为100%。 bottom：相当于垂直方向为100%。 left：相当于水平方向为0。 center：相当于水平方向为50%，或者垂直方向为50%。 <百分比>及<长度>见下文
初始值	0
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	如果是<长度>则同指定值，否则为百分比值

本属性同background-repeat属性一样只当指定了背景图片时才会生效。

虽然语法表中的值比较多，实际上background-position属性的定义很简单，例如右边代码都是合法的定义。

```
background-position : center top;
background-position : 10px 30px;
background-position : 20% 40%;
background-position : center right;
background-position : bottom;
```

1. 图片定位的起点

在上一小节内介绍过，CSS 2.1中规定的背景的范围为边框边包围的范围，但是，对于背景图片，其起始位置，却不是边框边，而是补白边，如图10-19所示。

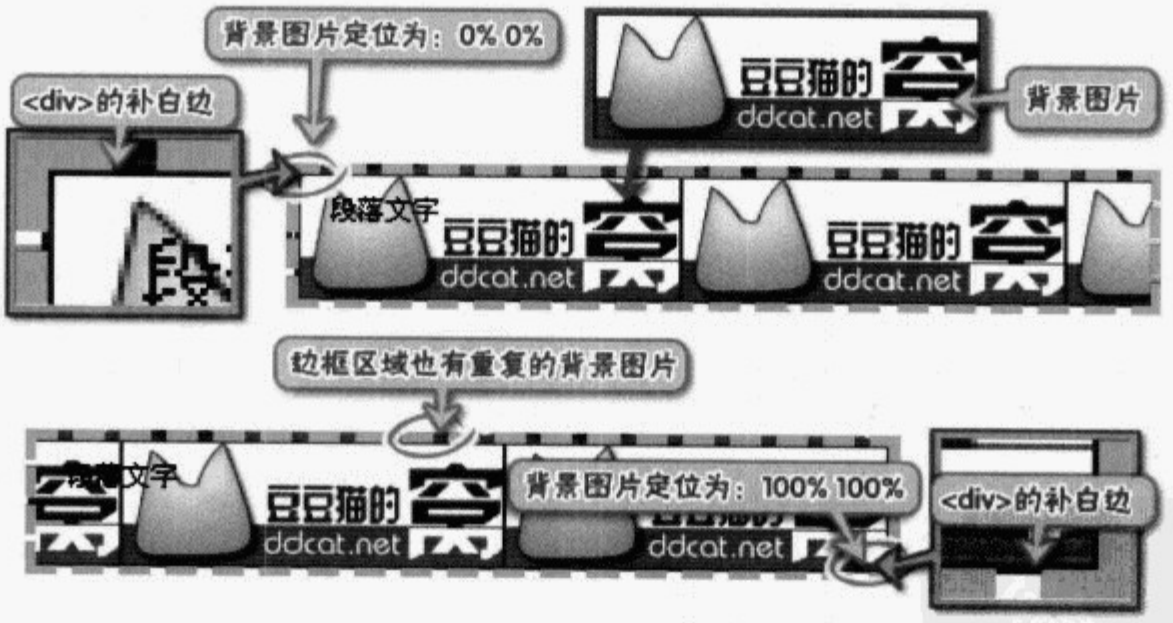
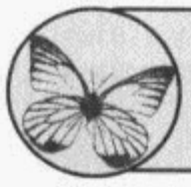


图10-19　背景图片的起始位置为元素的补白边

背景图片从补白边开始渲染，这样就避免了边框会覆盖住背景图片的问题。不过，由图10-19还可以发现，边框边所包含的位置，依然有背景图片，因为图片的重复区域是边框边包围的区域。



提示：在IE 7.0及更早的版本中，背景图片的定位的问题，请参见本书［16.2.1 hasLayout属性］一节。

2. 值的顺序

background-position属性的值可以是1个也可以是两个，如果只有1个值，则第2个值会被假定为“center”。例如左边代码会被认为是右边代码：

```
background-position : bottom;
background-position : 50px;
```

```
background-position : bottom center;
background-position : 50px center;
```

如果两个值中有1个不是关键字（关键字表明了定位的方向）或者都不是关键字，则认为第一个值为水平方向的定位，而第2个值为垂直方向的定位。例如：

```
background-position : 20px center; /* 水平方向左20px，垂直方向 center */
background-position : 20px 50px; /* 水平方向左20px，垂直方向 50px */
 background-position : bottom 30px; /* 水平方向bottom，垂直方向 30px ，由于bottom不是水平方向可接受的关键字，因此条规则将被抛弃 */
background-position : bottom right; /* 全部是关键字，则按其关键字方向 */
```

因此，如果两个值中有非关键字的值时，一定要注意书写顺序。

百分比值和长度值允许负值，负值将使图片在元素左补白边之左和上补白边之上显示，例如下列代码，其显示如图10-20所示。

```
#background7 p {
background-color : #CFF;
background-image : url(../../img/ddcat3.gif);
background-position : -20px -20px;
border : 5px dashed #FC0;
}
<div id="background7">
 <p>段落文字</p>
</div>
```

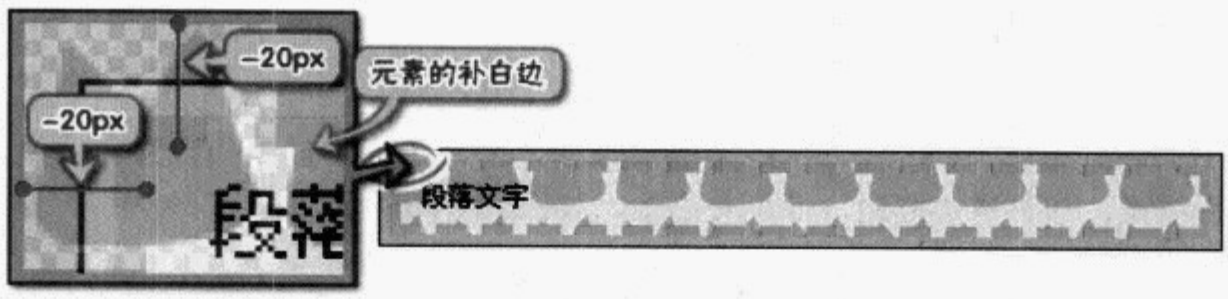


图10-20 负值背景图片定位的显示

3. 长度值

长度值比较好理解，即按照前面介绍的起点位置，值为正数时向右及向下偏移，值为负数时，向左和向上偏移。例如下列代码，其显示如图10-21所示。

```
# background8 p {
line-height : 40px;
background-color : #CFF;
background-image : url(../../img/ddcat3.gif);
background-position : 40px 10px;
background-repeat : no-repeat;
}
```

```
# background8 .position1 {
background-repeat : repeat;
}
<div id="background8">
 <p>文字</p>
 <p class="position1">文字</p>
</div>
```

图10-21 背景图片长度值定位

由图10-21可以发现，背景图片的重复，也是以定位后的位置为起点。

4. 百分比值

百分比值的定位与长度值不同，它不是根据某个值计算出便宜的数值，而是将背景图片的百分比位置与元素的百分比位置对应，例如下列代码：

```
background-position : 30% 20%; .
```

其含义是，将背景图片上水平位置30%和垂直位置20%的点，与元素补白框的水平位置30%和垂直位置20%的点对齐，例如下列代码：

```
#background9 p {
padding:10px;
width:280px;
height:80px;
background-color: #CFF;
background-image:url(../../img/ddcat_ad.gif);
background-position:30% 20%;
background-repeat:no-repeat;
}
<div id="background9">
  <p>文字</p>
</div>
```

图10-22　背景图片百分比值定位的显示

背景图片的尺寸宽为180px，高为60px，则其（30%，20%）的点为（54px，12px）；元素<p>的补白框宽度为300px（10px+280px+10px），高度为100px（10px+80px+10px），则其（30%，20%）的点为（90px，20px），其显示如图10-22所示。如果背景重复，也是在定位的基础之上，如图10-23所示。

图10-23　背景图片重复时也是以定位为基础

5. 关键字值

5个关键字中，除了“center”，其他都表明了其定位方向，各关键字定义如下。

- top：相当于垂直方向为0%。
- right：相当于水平方向为100%。
- bottom：相当于垂直方向为100%。
- left：相当于水平方向为0%。
- center：相当于水平方向为50%，或者垂直方向为50%。

当两个属性值都为关键字时，其对应的位置如图10-24所示。

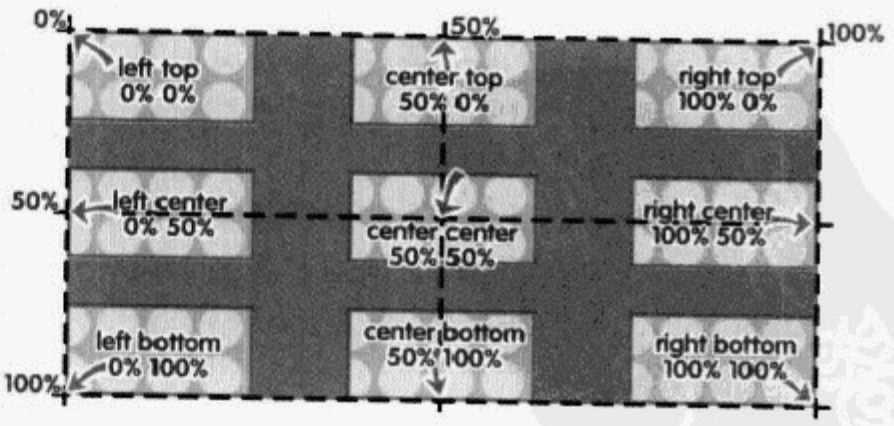


图10-24　关键字对应的背景图片位置

在上文介绍过，关键字和长度值、百分比值混用时，无论关键字值所表示的是哪个方向，都会将第一个值作为水平方向值，第2个值作为垂直方向值，因此要特别注意书写顺序。

不过有些浏览器（例如IE 7.0及其更早的版本）则会根据关键字表示的方向来显示背景图片，例如下列代码，其在各个浏览器内显示如图10-25所示。

```
#background10 p {
line-height:60px;
```

```
background-color: #CFF;
background-image:url(../../img/ddcat3.gif);
background-position: bottom 30px;
background-repeat:no-repeat;
}
<div id="background10">
 <p>文字</p>
</div>
```

由图10-25可以发现，在Opera 9.2中，由于第一个值“bottom”不是合法的水平方向的关键字，因此此条规则被忽略。而在IE 7.0内，识别“bottom”值为垂直方向的“100%”，同时将第2个值应用在水平方向定位。

图10-25 浏览器对于关键字和长度值混用的不同理解

10.3.6 缩写属性：background

背景的颜色、图片、定位等属性，可以使用缩写background属性来简化，定义列表如下：

语法	background : [<'background-color'> || <'background-image'> || <'background-repeat'> || <'background-attachment'> || <'background-position'>] | inherit
说明	设置元素的背景
值	参见各详细属性定义
初始值	视各详细属性
继承性	不继承
适用于	所有元素
媒体	视觉
计算值	视各详细属性

background属性中包含的这些属性不必全写，因为background属性先设置所有的独立背景属性为它们的初始值，然后赋予在声明中显式给出的值。例如下列定义都是正确的：

```
p { background : #FFF; } /* 设定背景颜色为白色 */
p { background : #FFF url(ddcat.gif); } /* 设定背景颜色为白色，背景图片为"ddcat.gif" */
 p { background : url(ddcat.gif) no-repeat right bottom; } /* 背景图片为"ddcat.gif"，不重复，定位为右下 */
```

同时也因为background属性会将没有显式定义的属性设置为初始值，因此要特别注意规则的先后顺序，如右上代码：

```
p {
background-image : url(ddcat.gif);
background : #FFF no-repeat;
}
```

由于第2条background属性会覆盖第一条的定义，因此<p>元素将有白色的背景色但是没有背景图片，但是如果将2条规则调换位置如右下代码：

```
p {
background : #FFF no-repeat;
background-image : url(ddcat.gif);
}
```

则<p>元素会有白色的背景色和不重复显示的背景图片。可以利用此点特性，先定义某类元素的通用属性，然后再针对个别元素设定专用的属性，如下代码：

```
a {
background : #FCC no-repeat left bottom;
……
```

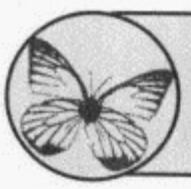
```
}
a.home {
background-image : url(home.gif);
}
a.blog {
background-image : url(blog.gif);
}
<ul>
 <li><a href="#" titile="返回首页" class="home">首页</a></li>
 <li><a href="#" titile="进入博客" class="blog">博客</a></li>
<ul>
```

此时，2个<a>元素会有相同的背景色以及其他的属性，但是有不同的背景图片。

10.3.7 <html>元素的背景

对于根元素，其背景覆盖整个渲染区域，对于(X)HTML文档，如果根元素<html>元素的背景颜色background-color属性设置为“transparent”并且背景图片background-image属性设置为“none（无）”，则使用<body>的相关属性来替代<html>元素的属性，并且不再对<body>的背景进行绘制。例如下列代码，其显示如图10-26所示。

图10-26　<html>元素有背景颜色时的显示



提示：本示例代码，读者可参见下载文件包内［/第2部分/第10章：颜色与背景/background2.html］文件。

```
html {
background-color : #6CF;
}
body {
margin : 20px;
border : 4px solid #FFCC33;
background : #FFFF99 url(../../img/ddcat_ad.gif) no-repeat;
color : #000;
}
<body id="c_background">
 <p>段落文字，<em>段落内的em</em></p>
</body>
```

如果将<html>元素的背景颜色设置为“transparent”，CSS如下，其显示如图10-27所示。

```
html { background-color : transparent; }
```

由图10-27可以发现，<html>元素的背景颜色和图案为<body>元素的背景，而<body>元素背景变为透明的。在CSS 2.1规范中，推荐作者指定<body>元素的背景而不是<html>的背景。如果<body>也未设置背景，则使用浏览器内置的样式。在制作页面的时候，即使<body>元素的背景是白色，也要设置background-color属性值，因为有可能访问者的浏览器的背景色不是白色而是灰色或者其他颜色。

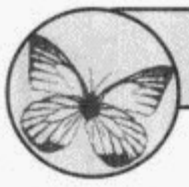
图10-27　<html>元素的背景颜色为“transparent”时的显示

10.4 应用

前景色和背景结合，可以制作出很多好的视觉效果，特别是在链接、菜单、栏目框等表现的处理上。下面将介绍几种常用的效果。本示例代码，可以参见下载文件包内［/第2部分/第10章：颜色与背景/background_sample.html］文件。

10.4.1 灵活使用背景

在页面设计中，为了分隔内容或者装饰效果，经常会使用线段、颜色、Logo等方法来区分不同类型的内容或者栏目，在一般情况下，设定元素的边框即可实现分隔线的效果，但是对于一些border-style属性无法实现的框线效果，就需要使用背景图片来实现。



提示：在CSS 3中将有更多的border属性可以设置，包括圆角边框、边框图片等。

在使用背景的时候，要注意以下几点。

● 应用背景范围的可伸缩性。例如，利用背景图片来实现边框或者分隔线的时候，要注意宽度或者高度可能会变化，因此可以尽量利用background-repeat属性，来实现在宽度或者高度上的扩展。

● 背景图片的定位。背景图片的定位要注意水平方向和垂直方向的值的顺序。

● 利用已有的结构中的所有元素。由于背景的属性可以应用在所有元素上，因此善于利用背景是完成页面“表现”的重要手段。有时候，可能需要添加额外的元素（例如 或者元素）来实现效果，不过在实现的时候要把握一个原则——尽量少地添加与内容结构无关的元素来实现。

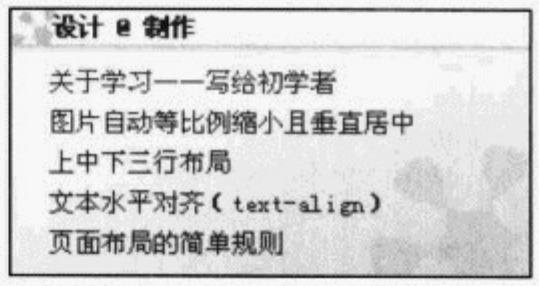


图10-28 样例设计效果图

如图10-28所示的设计效果图，其对应的XHTML代码如下：

```
<div id="design">
 <h4>设计 @ 制作</h4>
 <ul>
   <li><a href="#" title="关于学习——写给初学者">关于学习——写给初学者</a></li>
  <li><a href="#" title="图片自动等比例缩小且垂直居中">图片自动等比例缩小且垂直居中</a></li>
  <li><a href="#" title="上中下三行布局">上中下三行布局</a></li>
  <li><a href="#" title="文本水平对齐（text-align）">文本水平对齐（text-align）</a></li>
  <li><a href="#" title="页面布局的简单规则">页面布局的简单规则</a></li>
 </ul>
</div>
```

对设计图的分析如图10-29所示。

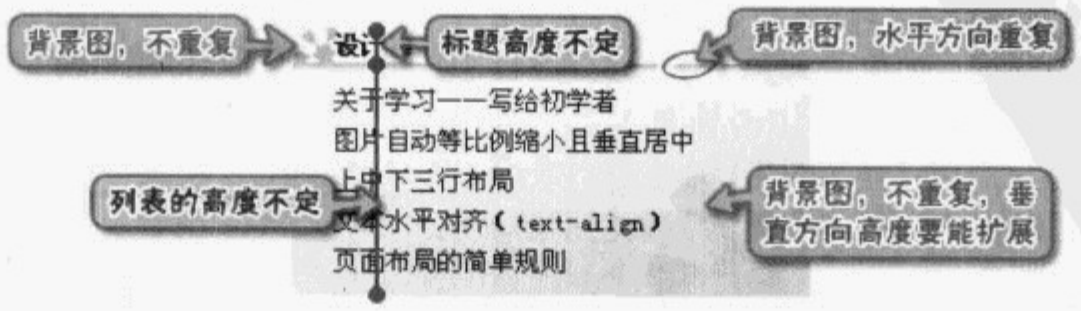


图10-29 对设计效果图的分析