

第一章 HTML 基本结构及实体

本章目标：了解 HTML 文档的基本结构
掌握 HTML 结构标签<html><head><title><body>
掌握 HTML 字符实体

本章重点：了解 HTML 文档的基本结构

本章难点：HTML 字符实体的使用

一、HTML 文档的基本结构

HTML 文件是什么？

- ☐ HTML 表示超文本标记语言（Hyper Text Markup Language）。
- ☐ HTML 文件是一个包含标记的文本文件。
- ☐ 这些标记告诉浏览器怎样显示这个页面。
- ☐ HTML 文件必须有 htm 或者 html 扩展名。
- ☐ HTML 文件可以用一个简单的文本编辑器创建。

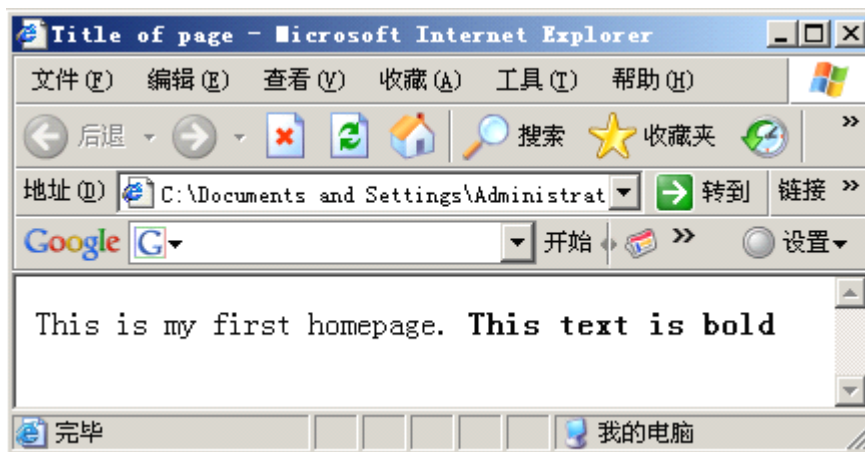
想不想尝试一下？

假如你运行的是 windows 系统，打开记事本，在其中输入以下文本：

```
<html>
<head>
  <title>Title of page</title>
</head>
<body>
  This is my first homepage.
  <b>This text is bold</b>
</body>
</html>
```

将此文件保存为“mypage.htm”。

启动浏览器。在文件菜单中选择“打开”（或者“打开页面”），这时将出现一个对话框。选择“浏览”（或者“选择文件”），定位到你刚才创建的 HTML 文件——“mypage.htm”，选择它，单击“打开”。然后在对话框中，你将看到这个文件的地址，比如说：“C:\MyDocuments\mypage.htm”。单击“确定”，浏览器将显示此页面。



例子解释：

HTML 文档中，第一个标签是<html>。这个标签告诉浏览器这是 HTML 文档的开始。HTML 文档的最后一个标签是</html>，这个标签告诉浏览器这是 HTML 文档的终止。

在<head>和</head>标签之间文本的是头信息。在浏览器窗口中，头信息是不被显示的。

在<title>和</title>标签之间的文本是文档标题，它被显示在浏览器窗口的标题栏。

在<body>和</body>标签之间的文本是正文，会被显示在浏览器中。

在和标签之间的文本会以加粗字体显示。

关于 HTML 编辑器：

用一些所见即所得的编辑器，比如 frontpage，dreamwaver，你可以很容易创建一个页面，而不需要在纯文本中编写代码。

但是假如你想成为一名熟练的网络开发者，我们强烈推荐你用纯文本编辑器编写代码，这有助于学习 HTML 基础。

常见问题：

问：我编写完了 HTML 文件，但是不能在浏览器中看见结果，为什么？

答：请确认你保存了文件，并且使用了正确的文件名和扩展名，例如：“c:\mypage.htm”，并且确认你用浏览器打开时使用同样的文件名。

问：我编辑了 HTML 文件，但是修改结果并没有在浏览器中显示，为什么？

答：浏览器缓存了你的页面，所以它不需要两次读取同样的页面。你修改了这个页面，浏览器并不知道。请使用“刷新/重载”按钮来强迫浏览器读取编辑过的页面。

HTML 元素：

HTML 文档是由 HTML 元素组成的文本文件。

HTML 元素是预定义的正在使用的 HTML 标签。

HTML 标签：

HTML 标签用来组成 HTML 元素。

HTML 标签两端有两个包括字符：“<”和“>”，这两个包括字符被称为角括号。

HTML 标签通常成对出现，比如和。一对标签的前面一个是开始标签，第二个是结束标签,在开始和结束标签之间的文本是元素内容。

HTML 标签是大小写无关的，跟表示的意思是一样的。

HTML 元素：

回忆一下上面的 HTML 例子：

```
<html>
<head>
  <title>Title of page</title>
</head>
<body>
  This is my first homepage.
  <b>This text is bold</b>
</body>
</html>
```

下面是一个 HTML 元素：

```
<b>This text is bold</b>
```

此 HTML 元素以开始标签起始， 内容是：This text is bold，以结束标签中止。标签的目的是定义一个需要被显示成粗体的 HTML 元素。

下面也是一个 HTML 元素：

```
<body>
This is my first homepage.
<b>This text is bold</b>
</body>
```

此 HTML 标签以开始标签<body>起始，终止于结束标签</body>。<body>标签的目的是定义一个 HTML 元素，使其包含 HTML 文档的主体。

为什么使用小写标签？

我们刚说过，HTML 标签是大小写无关的：跟含义相同。当你上网的时候，你会注意到多数教程在示例中使用大写的 HTML 标签，我们总是使用小写标签。为什么？

假如你想投入到下一代 HTML 中，你应该开始使用小写标签。W3C 在他们的 HTML4 建议中提倡使用小写标签，XHTML（下一代 HTML）也需要小写标签。

标签属性：

标签可以拥有属性。属性能够为页面上的 HTML 元素提供附加信息。

标签<body>定义了 HTML 页面的主体元素。使用一个附加的 bgcolor 属性，你可以告诉浏览器：你页面的背景色是红色的，就像这样：

```
<body bgcolor="red">
```

标签<table>定义了一个 HTML 表格。使用一个附加的 border 属性，你可以告诉浏览器：这个表格是没有边框的，代码是：

```
<table border="0">
```

属性通常由属性名和值成对出现，就像这样：name="value"。属性通常是附加给 HTML 元素的开始标签的。

引号样式：

属性值应该被包含在引号中。双引号是最常用的，但是单引号也可以使用。

在很少情况下，比如说属性值本身包含引号，使用单引号就很必要了。

比如：name='John "ShotGun" Nelson'。

注意：中文引号跟英文引号是不一样的。上面所指的引号都是英文状态下的引号。

二、 HTML 实体

有些字符，比如说“<”字符，在 HTML 中有特殊的含义，因此不能在文本中使用。

想要在 HTML 中显示一个小于号“<”，需要用到字符实体。

字符实体：

在 HTML 中，有些字符拥有特殊含义，比如小于号“<”定义为一个 HTML 标签的开始。假如我们想要浏览器显示这些字符的话，必须在 HTML 代码中插入字符实体。

一个字符实体拥有三个部分：一个 and 符号（&），一个实体名或者一个实体号，最后是一个分号（;）

想要在 HTML 文档中显示一个小于号，我们必须这样写：<或者<

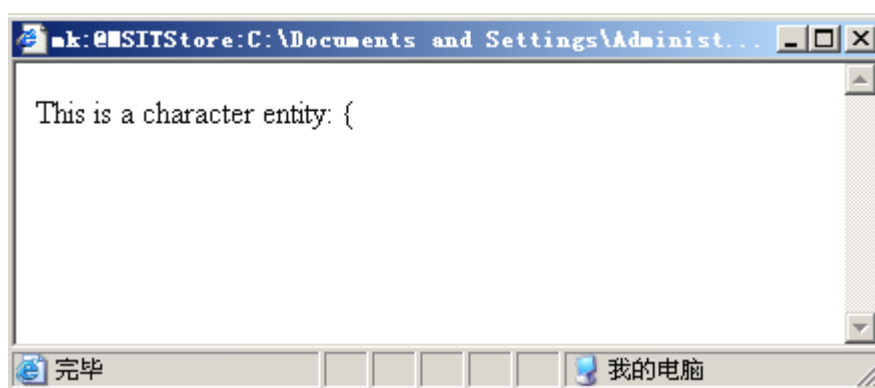
使用名字相对于使用数字的优点是容易记忆，缺点是并非所有的浏览器都支持最新的实体名，但是几乎所有的浏览器都能很好地支持实体号。

注意：实体名是大小写敏感的。

下面这个例子能够让你针对 HTML 实体实践一下。

```
<html>
  <body>
    <p>This is a character entity: &#123;</p>
  </body>
</html>
```

运行代码，结果如下：



不可拆分的空格

在 HTML 中，最常见的字符实体就是不可拆分空格。

通常，HTML 会合并你文档中的空格。假如在你的 HTML 文本中连续写了 10 个空格，其中 9 个会被去掉。想要在 HTML 中插入空格，可以使用实体：

最常用的字符实体：

显示结果	描述	实体名	实体号
	不可拆分的空格	 	
<	小于	<	<
>	大于	>	>
&	and符号	&	&
"	引号	"	"
'	单引号		'

其他一些常用的字符实体：

显示结果	描述	实体名	实体号
¢	分	¢	¢
£	英镑	£	£
¥	人民币元	¥	¥
§	章节	§	§
©	版权	©	©
®	注册	®	®
×	乘号	×	×
÷	除号	÷	÷

第二章 HTML 基本元素的运用

本章目标：掌握下列标签：

段落相关标签<p>
<hr>

格式化相关标签<small><sub><sup><pre>

列表相关标签

图片相关标签

超链相关标签<a>

本章重点：段落相关标签，超链标签

本章难点：超链相关标签<a>

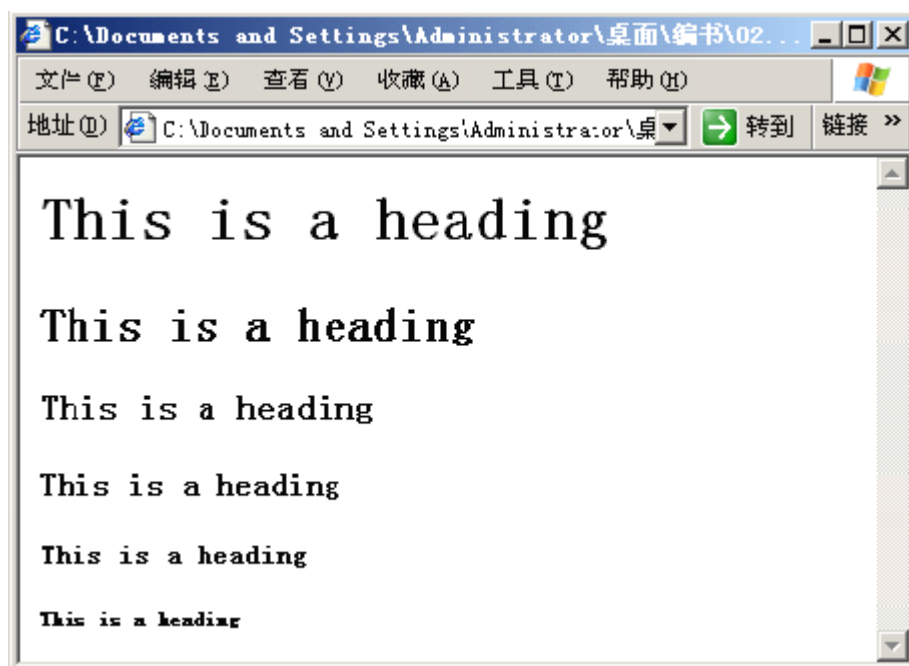
一、段落相关标签

标题元素：

标题元素由标签<h1>到<h6>定义。<h1>定义了最大的标题元素，<h6>定义了最小的。

```
<h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3>
<h4>This is a heading</h4>
<h5>This is a heading</h5>
<h6>This is a heading</h6>
```

运行代码，结果如下：



HTML 自动在一个标题元素前后各添加一个空行。

段落：

段落是用<p>标签定义的。

```
<p>This is another paragraph</p>
```

运行代码，结果如下：



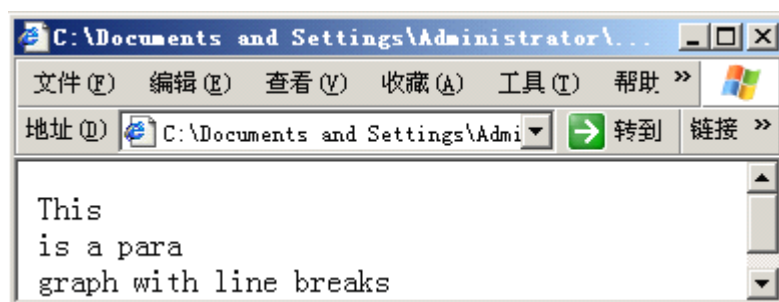
HTML 自动在一个段落前后各添加一个空行。

换行：

当需要结束一行，并且不想开始新段落时，使用
标签。
标签不管放在什么位置，都能够强制换行。

```
<p>This <br> is a para<br>graph with line breaks</p>
```

运行代码，结果如下：



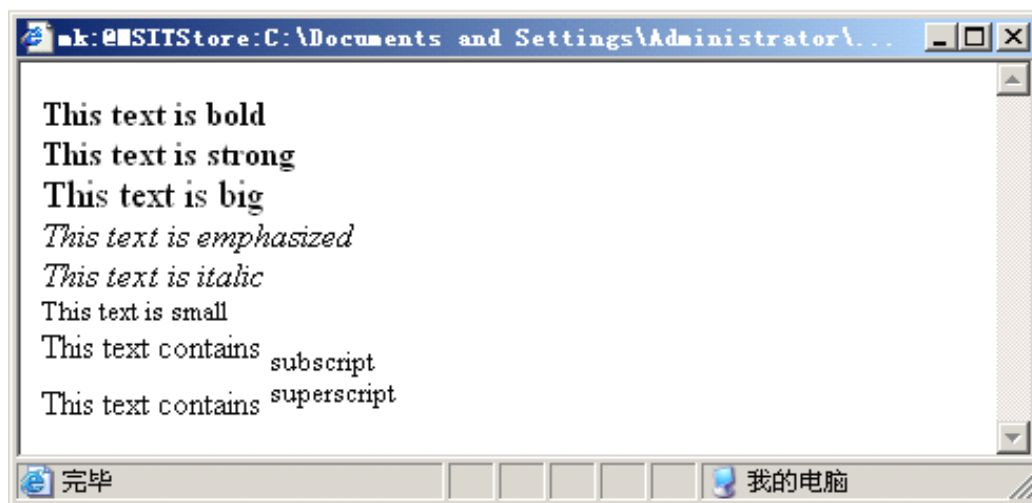
标签是一个空标签，它没有结束标记。

二、 格式化相关标签

格式化文字：

```
<html>
<body>
  <b>This text is bold</b><br>
  <strong>
    This text is strong
  </strong><br>
  <big>
    This text is big
  </big><br>
  <em>
    This text is emphasized
  </em><br>
  <i>
    This text is italic
  </i><br>
  <small>
    This text is small
  </small><br>
  This text contains
  <sub>
    subscript
  </sub><br>
  This text contains
  <sup>
    superscript
  </sup>
</body>
</html>
```

运行代码，结果如下：



这个例子说明了在 HTML 里面可以怎样格式化文本。

三、 列表相关标签

无序列表：

无序列表是一个项目的序列。各项目前加有标记（通常是黑色的实心小圆圈）。无序列表以标签开始。每个列表项目以开始。

```
<ul>
  <li>Coffee</li>
  <li>Milk</li>
</ul>
```

运行代码，结果如下：



无序列表的项目中可以加入段落、换行、图像，链接，其他的列表等等。

有序列表：

有序列表也是一个项目的序列。各项目前加有数字作标记。有序列表以标签开始。每个列表项目以开始。

```
<ol>
```

```
<li>Coffee</li>
<li>Milk</li>
</ol>
```

运行代码，结果如下：

```
1. Coffee
2. Milk
```

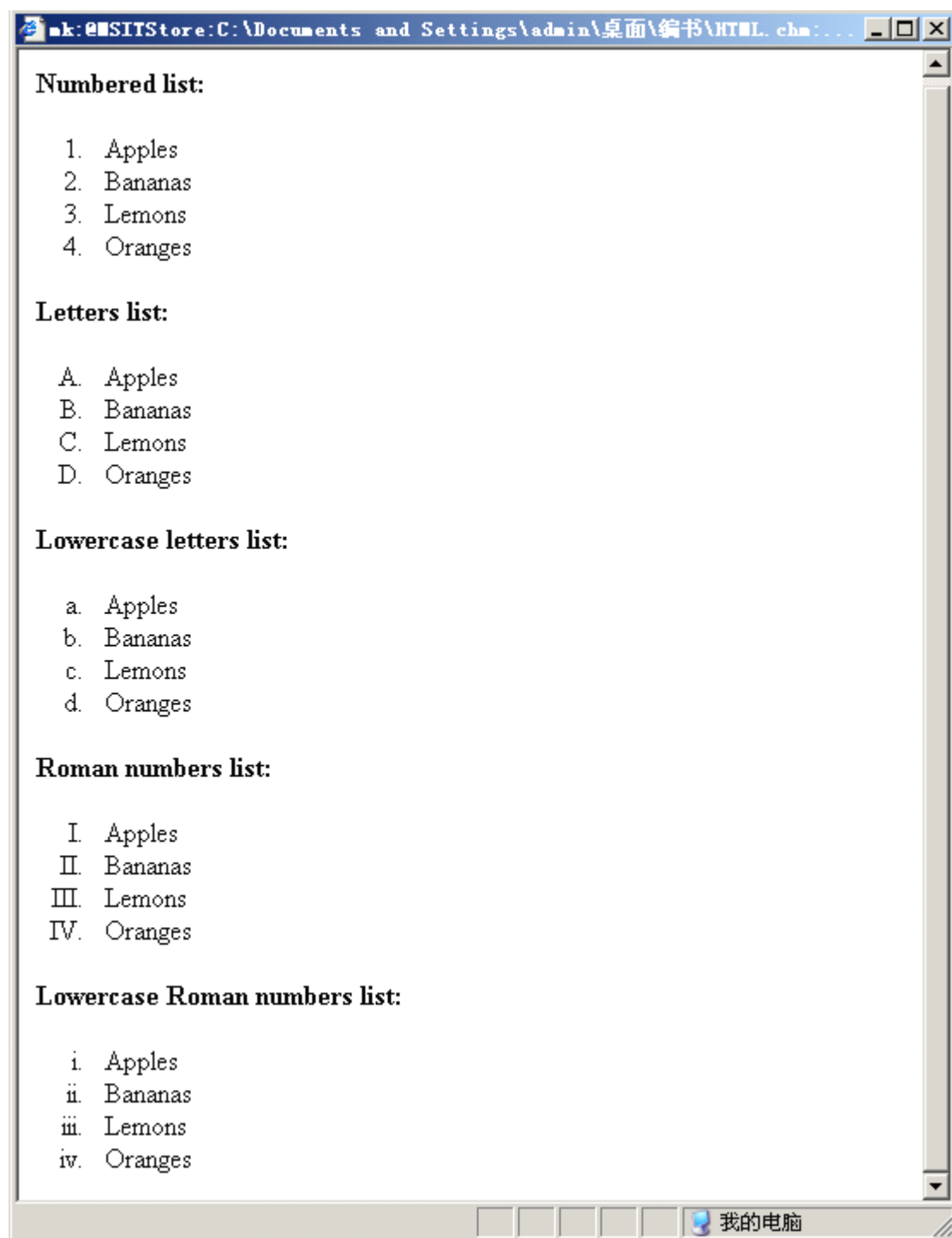
更多示例：

有序列表的不同类型：

```
<html>
<body>
<h4>Numbered list:</h4>
<ol>
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ol>
<h4>Letters list:</h4>
<ol type="A">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ol>
<h4>Lowercase letters list:</h4>
<ol type="a">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ol>
<h4>Roman numbers list:</h4>
<ol type="I">
  <li>Apples</li>
```

```
<li>Bananas</li>
<li>Lemons</li>
<li>Oranges</li>
</ol>
<h4>Lowercase Roman numbers list:</h4>
<ol type="i">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ol>
</body>
</html>
```

运行代码，结果如下：



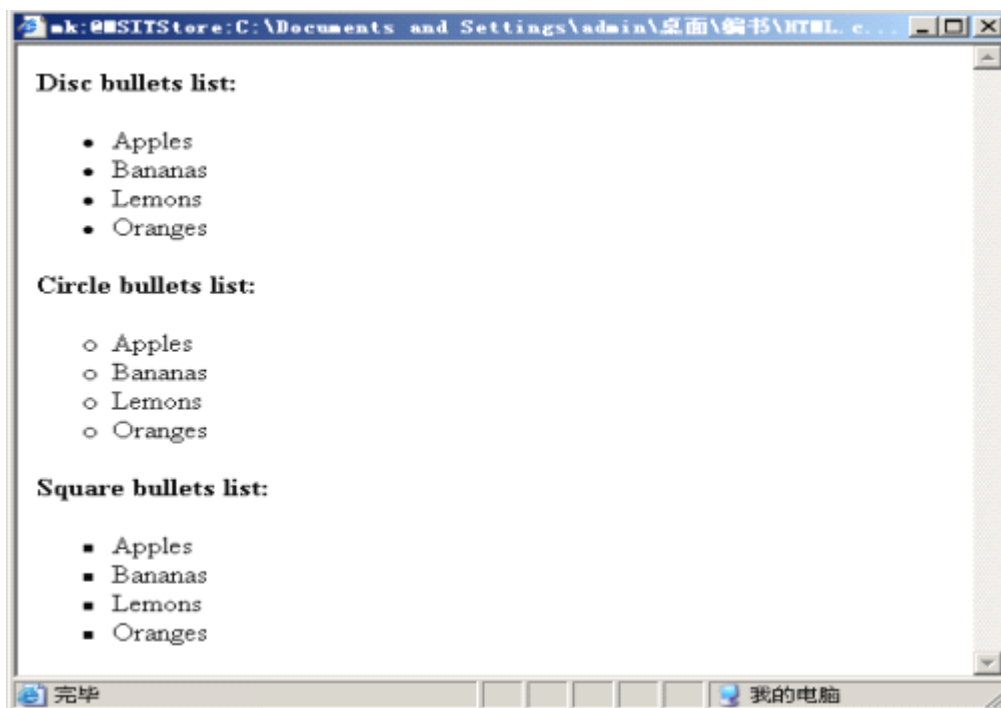
这个例子显示了有序列表的不同类型。

无序列表的不同类型：

```
<html>
```

```
<body>
<h4>Disc bullets list:</h4>
<ul type="disc">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ul>
<h4>Circle bullets list:</h4>
<ul type="circle">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ul>
<h4>Square bullets list:</h4>
<ul type="square">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ul>
</body>
</html>
```

运行代码，结果如下：



这个例子显示了无序列表的不同类型。

四、 图片相关标签

Img 标签和 src 属性：

在 HTML 里面，图像是由标签定义的。

是空标签，意思是说，它只拥有属性，而没有结束标签。

想要在页面上显示一个图像，需要使用 src 属性。“src”表示“源”的意思。“src”属性的值是所要显示图像的 URL。

插入图像的语法：

```

```

URL 指向图像存储的地址。网站“www.w3schools.com”子目录“images”中的图像“boat.gif”的 URL 如下：“http://www.w3schools.com/images/boat.gif”。

当浏览器在文档中遇到 img 标签时，就放置一个图像。如果把 img 标签放在两个段落之间，就会先显示一个段落，然后是这个图像，最后是另外一个段落。

alt 属性：

alt 属性用来给图像显示一个“交互文本”。alt 属性的值是由用户定义的。

```

```

“alt”属性在浏览器装载图像失败的时候告诉用户所丢失的信息，此时，浏览器显示这个“交互文本”来代替图像。给页面上的图像都加上 alt 属性是一个好习惯，它有助于更好地显示信息，而且，对纯文本浏览器很有用。

基本注意点——有用的技巧：

如果一个 HTML 文档包含 10 个图像，那么为了正确显示这个页面，需要加载 11 个文件。加载图像是需要时间的，所以请谨慎使用图像。

更多示例：

调整图像大小：

```
<html>
<body>
  <p>
```

```

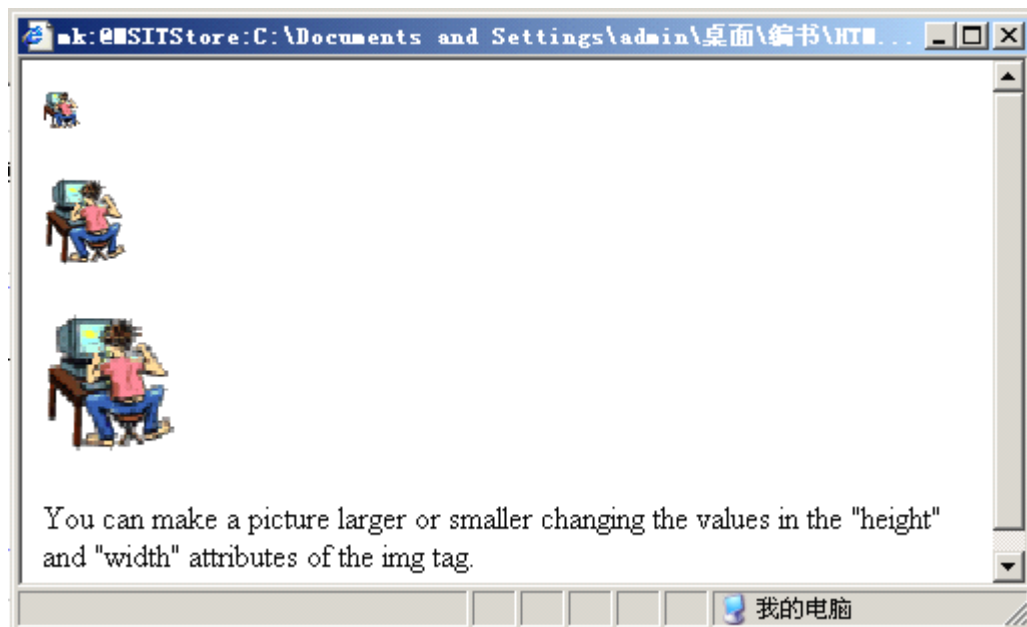

</p>
<p>

</p>
<p>

</p>
<p>
    You can make a picture larger or smaller changing the values in the "height"
    and "width" attributes of the img tag.
</p>
</body>
</html>

```

运行代码，结果如下：



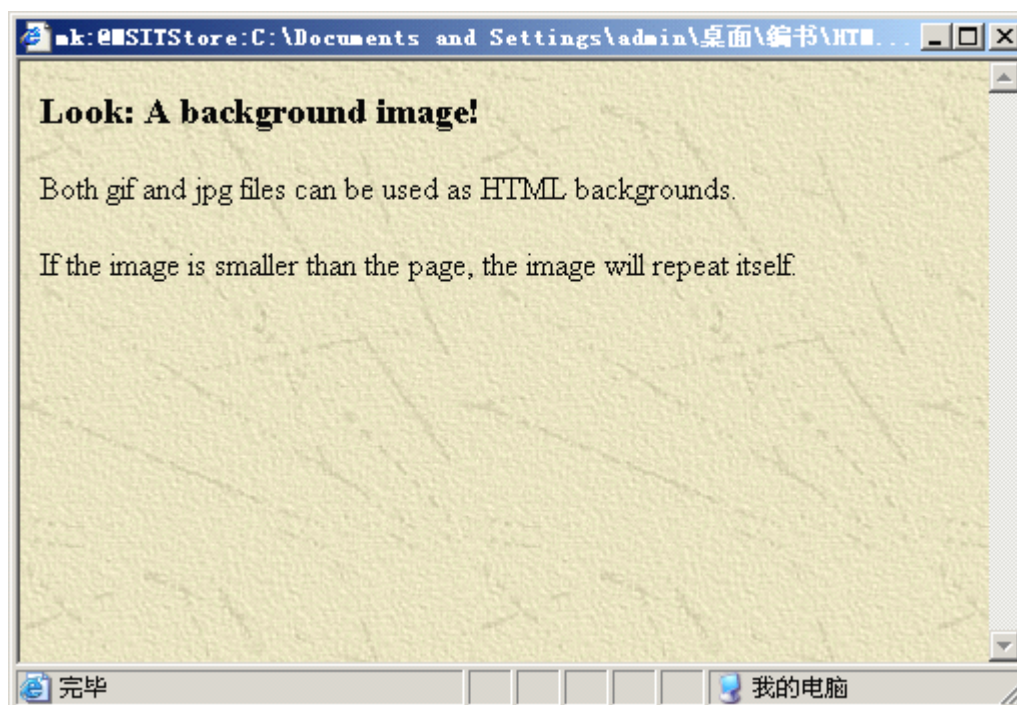
背景图像：

```

<html>
<body background="./images/background.jpg">
    <h3>Look: A background image!</h3>
    <p>Both gif and jpg files can be used as HTML backgrounds.</p>
    <p>If the image is smaller than the page, the image will repeat itself.</p>
</body>
</html>

```

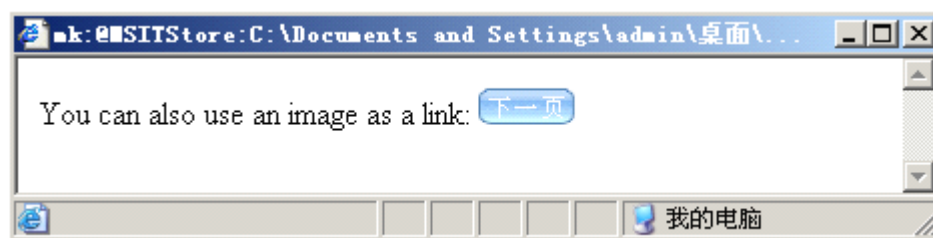
运行代码，结果如下：



图像链接:

```
<html>
<body>
  <p>
    You can also use an image as a link:
    <a href="back.htm">
      
    </a>
  </p>
</body>
</html>
```

运行代码，结果如下:



五、 超链相关标签

锚标签和 href 属性:

HTML 使用锚标签 (<a>) 来创建一个连接到其他文件的链接。锚可以指向网络上的任何资源: HTML 页面, 图像, 声音, 影片等等。

创建一个锚的语法:

```
<a href="url">Text to be displayed</a>
```

锚可以指向网络上的任何资源: HTML 页面, 图像, 声音, 影片等等。

标签<a>被用来创建一个链接指向的锚, href 属性用来指定连接到的地址, 在锚的起始标签<a>和结束标签中间的部分将被显示为超级链接。

这个锚定义了一个到 W3Schools 的链接:

```
<a href="http://www.w3schools.com/">Visit W3Schools!</a>
```

上面这段代码在浏览器中显示的效果如下:

[Visit W3Schools!](http://www.w3schools.com/)

target 属性:

使用 target 属性, 你可以定义从什么地方打开链接地址。

下面这段代码打开一个新的浏览器窗口来打开链接:

```
<a href="http://www.w3schools.com/" target="_blank">Visit W3Schools!</a>
```

锚标签和 name 属性

name 属性用来创建一个命名的锚。使用命名锚以后, 可以让链接直接跳转到一个页面的某一章节, 而不用用户打开那一页, 再从上到下慢慢找。

下面是命名锚的语法:

```
<a name="label">Text to be displayed</a>
```

你可以为锚随意指定名字, 只要你愿意。下面这行代码定义了一个命名锚:

```
<a name="tips">Useful Tips Section</a>
```

你应该注意到了: 命名锚的显示方式并没有什么与众不同的。

想要直接链接到“tips”章节的话, 在 URL 地址的后面加一个“#”和这个锚的名字, 就像这样:

```
<a name="http://www.w3schools.com/html_links.asp#tips">Jump to the Useful Tips Section</a>
```

一个链接到本页面中某章节的命名锚是这样写的：

```
<a name="#tips">Jump to the Useful Tips Section</a>
```

基本注意点——有用的技巧：

尽量在子目录路径后面加一个左斜杠。假如你像下面这样写：
`href="http://www.w3schools.com/html"`，将会产生向服务器产生两个 HTTP 请求，因为服务器会在后面追加一个左斜杠，产生一个新的请求，就像这样：
`href="http://www.w3schools.com/html/"`。

命名锚通常用来在大型文档的开头创建章节表。这个页面的每个章节被加上一个命名锚，到这些锚的链接被放在页面的顶端。

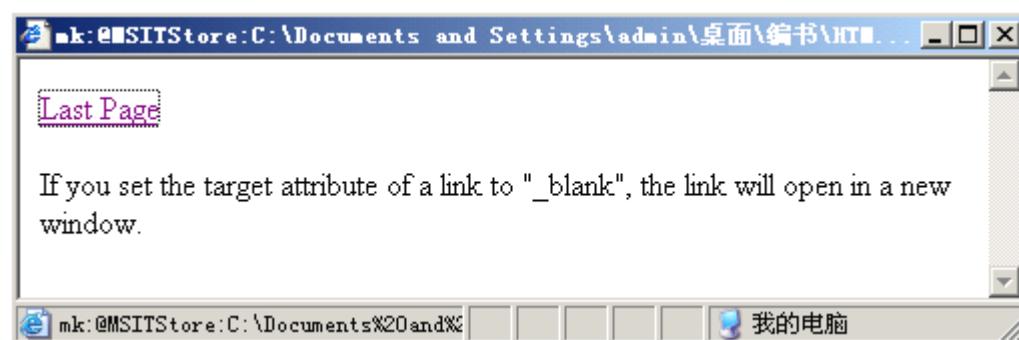
如果浏览器无法找到指定的命名锚，它将转到这个页面的顶部，而不显示任何错误提示。

更多示例：

在新浏览器窗口中打开链接：

```
<html>
<body>
<a href="lastpage.htm" target="_blank">Last Page</a>
<p>
If you set the target attribute of a link to "_blank",
the link will open in a new window.
</p>
</body>
</html>
```

运行代码，结果如下：



单击超连接，打开一个新窗口：



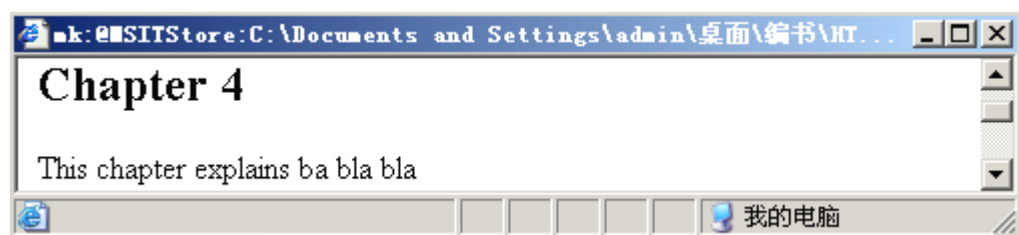
链接到本页面的某个位置：

```
<html>
<body>
  <p>
    <a href="#C4">
      See also Chapter 4.
    </a>
  </p>
  <p>
    <h2>Chapter 1</h2>
    <p>This chapter explains ba bla bla</p>
    <h2>Chapter 2</h2>
    <p>This chapter explains ba bla bla</p>
    <h2>Chapter 3</h2>
    <p>This chapter explains ba bla bla</p>
    <a name="C4"><h2>Chapter 4</h2></a>
    <p>This chapter explains ba bla bla</p>
  </body>
</html>
```

运行代码，结果如下：



单击超连接,



第三章 用 HTML 创建表格

本章目标：了解掌握表格的基本结构<table><tr><th><td>

掌握跨行、跨列属性 colspan rowspan

掌握表格相关修饰属性 border width height bgcolor

background height cellpadding cellspacing

本章重点：掌握表格的基本结构及相关属性

本章难点：掌握跨行、跨列属性 colspan rowspan

一、HTML 表格

表格:

表格是用<table>标签定义的。表格被划分为行（使用<tr>标签），每行又被划分为数据单元格（使用<td>标签）。td 表示“表格数据”（Table Data），即数据单元格的内容。数据单元格可以包含文本，图像，列表，段落，表单，水平线，表格等等。想不想尝试一下？

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

在浏览器中显示如下：

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

表格和 border 属性：

```
<table border="1">
<tr>
<td>Row 1, cell 1</td>
<td>Row 1, cell 2</td>
</tr>
</table>
```

如果不指定 border 属性，表格将不显示边框。有时候这很有用，但是多数时候我们希望

显示边框。

表格头：

表格头使用<th>标签指定。

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

在浏览器中显示如下：

Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

表格中的空单元格

在多数浏览器中，没有内容的单元格显示得不太好。

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td></td>
</tr>
</table>
```

在浏览器中显示如下：

row 1, cell 1	row 1, cell 2
row 2, cell 1	

注意一下空单元格的边框没有显示出来。为了避免这个，可以在空单元格里加入不可分

空格来占位，这样边框能正常显示。

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>&nbsp;</td>
</tr>
</table>
```

在浏览器中显示如下：

row 1, cell 1	row 1, cell 2
row 2, cell 1	

基本注意点——有用的技巧

通常很少使用<thead>，<tbody>，<tfoot>标签，因为浏览器对它们的支持不好。希望这个在 XHTML 的未来版本中得到改变。

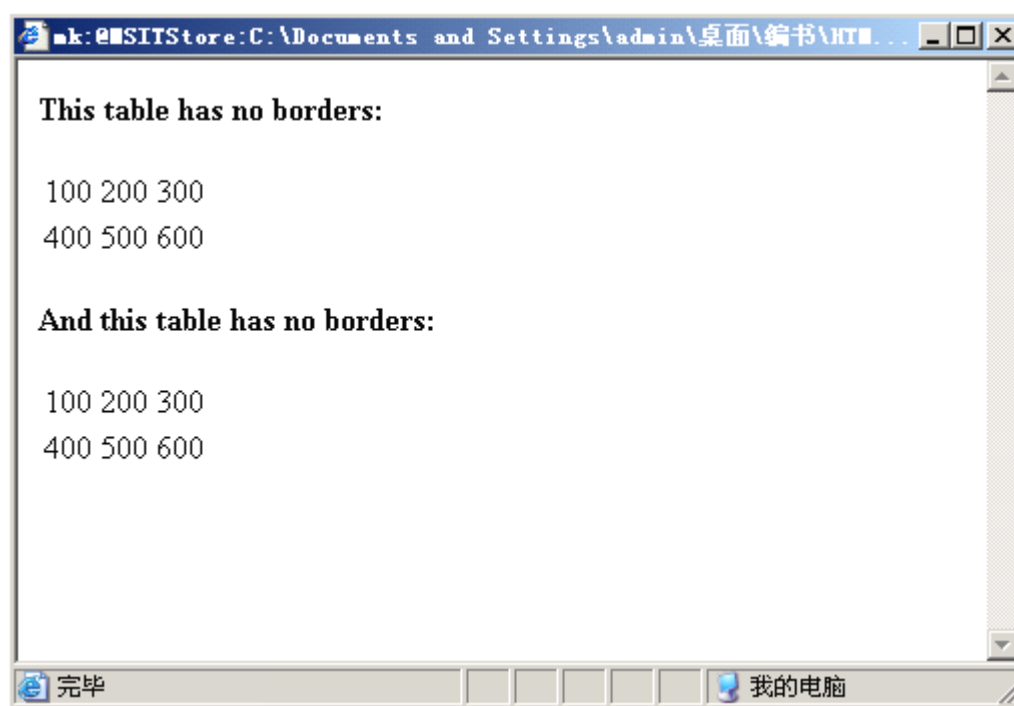
更多示例：

没有边框的表格：

```
<html>
<body>
<h4>This table has no borders:</h4>
<table>
<tr>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>400</td>
<td>500</td>
<td>600</td>
</tr>
</table>
<h4>And this table has no borders:</h4>
<table border="0">
<tr>
```



```
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>400</td>
<td>500</td>
<td>600</td>
</tr>
</table>
</body>
</html>
```



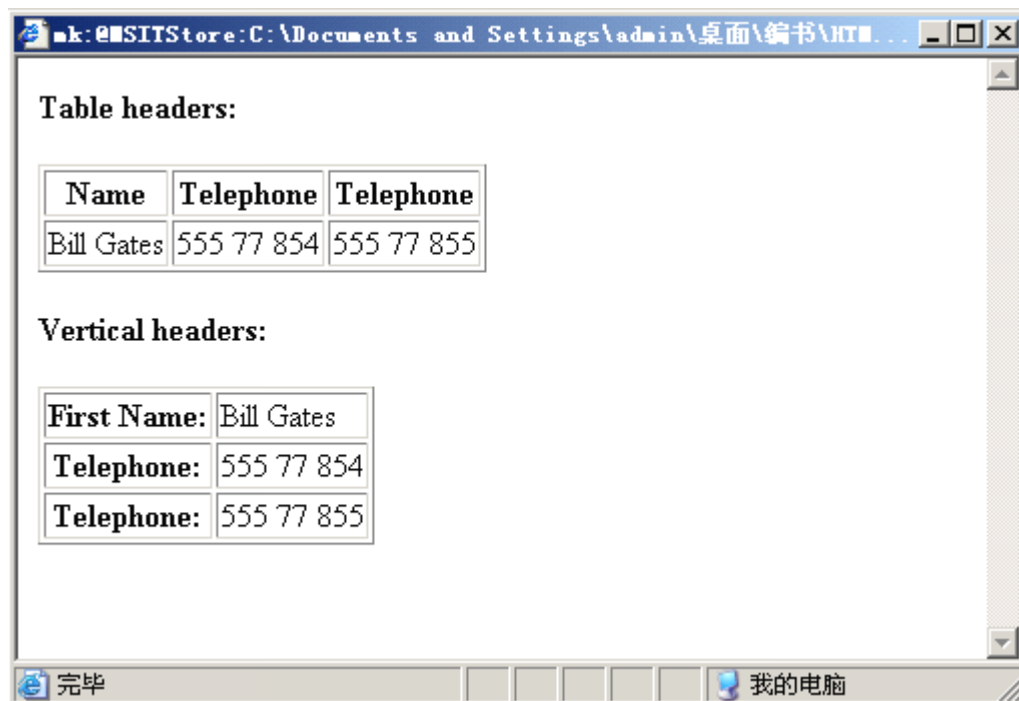
表格头:

```
<html>
<body>
<h4>Table headers:</h4>
<table border="1">
<tr>
<th>Name</th>
<th>Telephone</th>
<th>Telephone</th>
</tr>
<tr>
<td>Bill Gates</td>
```

```

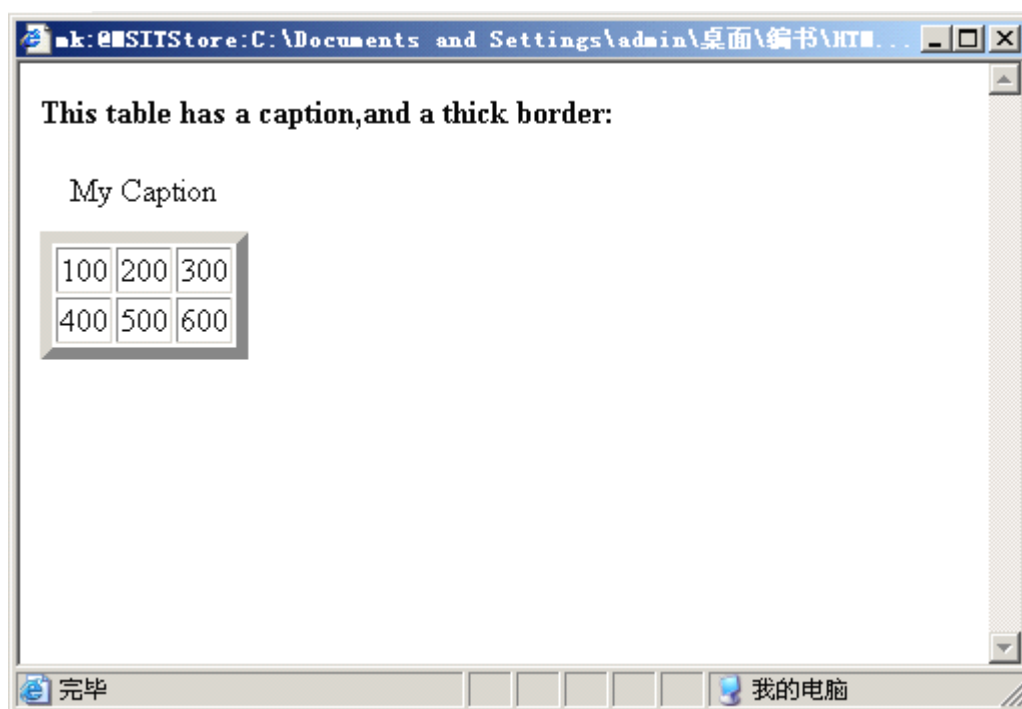
        <td>555 77 854</td>
        <td>555 77 855</td>
    </tr>
</table>
<h4>Vertical headers:</h4>
<table border="1">
<tr>
    <th>First Name:</th>
    <td>Bill Gates</td>
</tr>
<tr>
    <th>Telephone:</th>
    <td>555 77 854</td>
</tr>
<tr>
    <th>Telephone:</th>
    <td>555 77 855</td>
</tr>
</table>
</body>
</html>

```



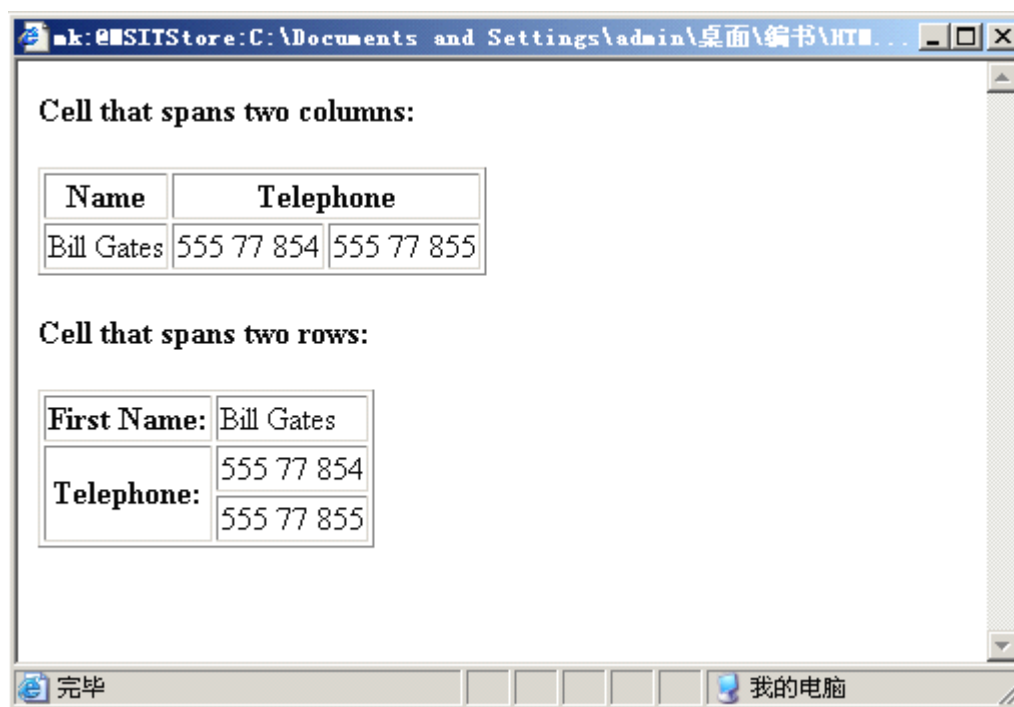
有标题的表格:

```
<html>
<body>
<h4>
This table has a caption, and a thick border:
</h4>
<table border="6">
<caption>My Caption</caption>
<tr>
    <td>100</td>
    <td>200</td>
    <td>300</td>
</tr>
<tr>
    <td>400</td>
    <td>500</td>
    <td>600</td>
</tr>
</table>
</body>
</html>
```



单元格跨行（列）的表格：

```
<html>
<body>
<h4>Cell that spans two columns:</h4>
<table border="1">
<tr>
  <th>Name</th>
  <th colspan="2">Telephone</th>
</tr>
<tr>
  <td>Bill Gates</td>
  <td>555 77 854</td>
  <td>555 77 855</td>
</tr>
</table>
<h4>Cell that spans two rows:</h4>
<table border="1">
<tr>
  <th>First Name:</th>
  <td>Bill Gates</td>
</tr>
<tr>
  <th rowspan="2">Telephone:</th>
  <td>555 77 854</td>
</tr>
<tr>
  <td>555 77 855</td>
</tr>
</table>
</body>
</html>
```



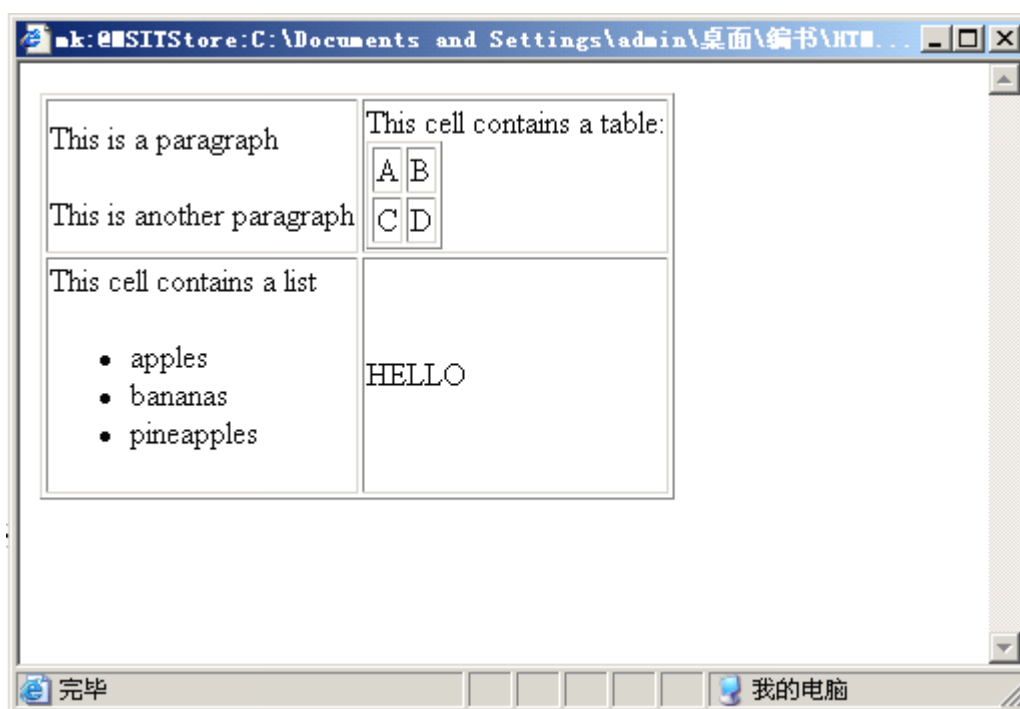
表格内的其他标签:

```
<html>
<body>
<table border="1">
<tr>
  <td>
    <p>This is a paragraph</p>
    <p>This is another paragraph</p>
  </td>
  <td>This cell contains a table:
    <table border="1">
      <tr>
        <td>A</td>
        <td>B</td>
      </tr>
      <tr>
        <td>C</td>
        <td>D</td>
      </tr>
    </table>
  </td>
</tr>
<tr>
  <td>This cell contains a list
    <ul>
```

```

        <li>apples</li>
        <li>bananas</li>
        <li>pineapples</li>
    </ul>
    </td>
    <td>HELLO</td>
</tr>
</table>
</body>
</html>

```



给表格增加背景色或者背景图像:

```

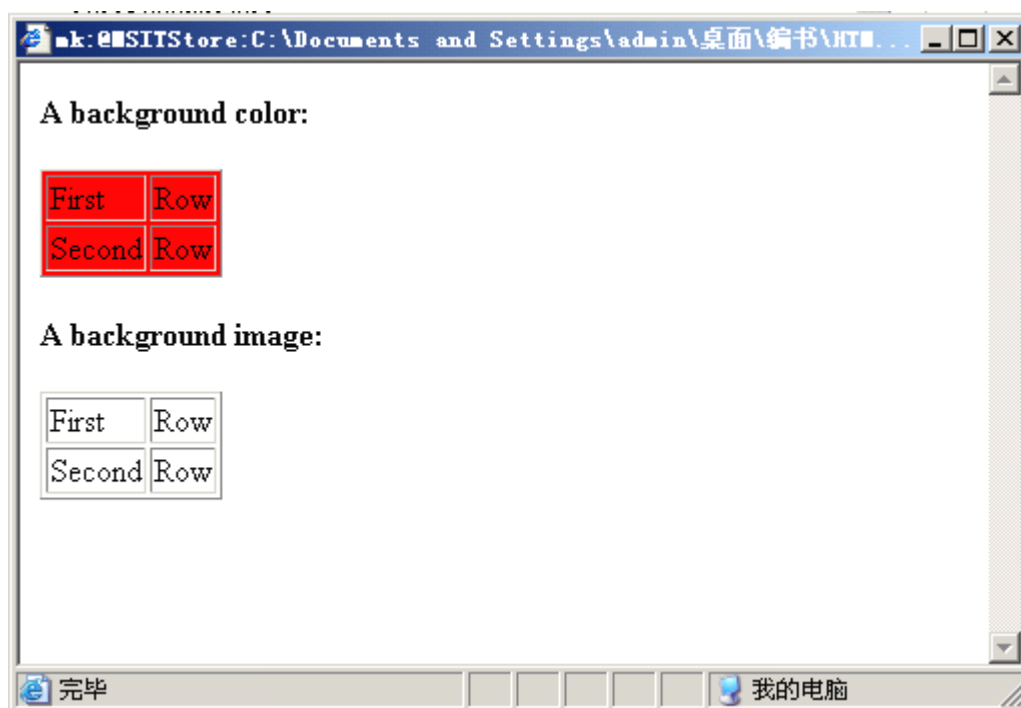
<html>
<body>
<h4>A background color:</h4>
<table border="1" bgcolor="red">
<tr>
    <td>First</td>
    <td>Row</td>
</tr>
<tr>
    <td>Second</td>
    <td>Row</td>
</tr>
</table>

```

```

<h4>A background image:</h4>
<table border="1" background="/images/bgdesert.jpg">
<tr>
    <td>First</td>
    <td>Row</td>
</tr>
<tr>
    <td>Second</td>
    <td>Row</td>
</tr>
</table>
</body>
</html>

```



这个例子说明了如何给表格增加背景。

```

<html>
<body>
<h4>Cell backgrounds:</h4>
<table border="1">
<tr>
    <td bgcolor="red">First</td>
    <td>Row</td>
</tr>
<tr>
    <td background="/images/bgdesert.jpg">Second</td>

```

```

        <td>Row</td>
    </tr>
</table>
</body>
</html>

```



这个例子说明了如何给一个或多个单元格增加背景。

cellpadding 属性:

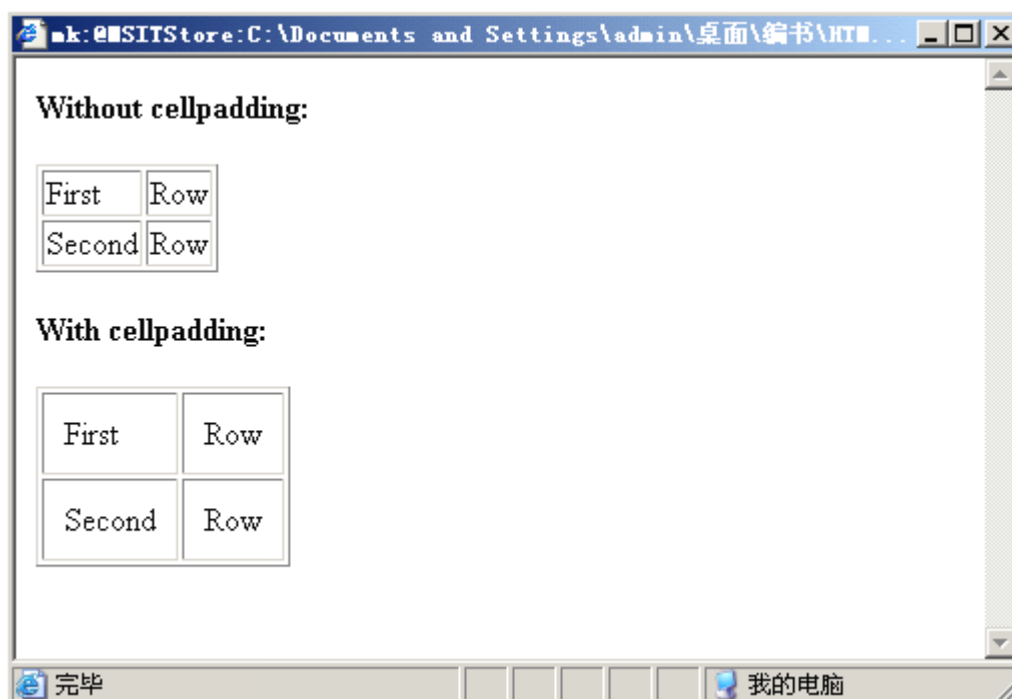
```

<html>
<body>
<h4>Without cellpadding:</h4>
<table border="1">
<tr>
    <td>First</td>
    <td>Row</td>
</tr>
<tr>
    <td>Second</td>
    <td>Row</td>
</tr>
</table>
<h4>With cellpadding:</h4>
<table border="1" cellpadding="10">
<tr>

```



```
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
</body>
</html>
```



这个例子说明了如何使用 cellpadding 属性在表格内容和边框之间留出更多空白。

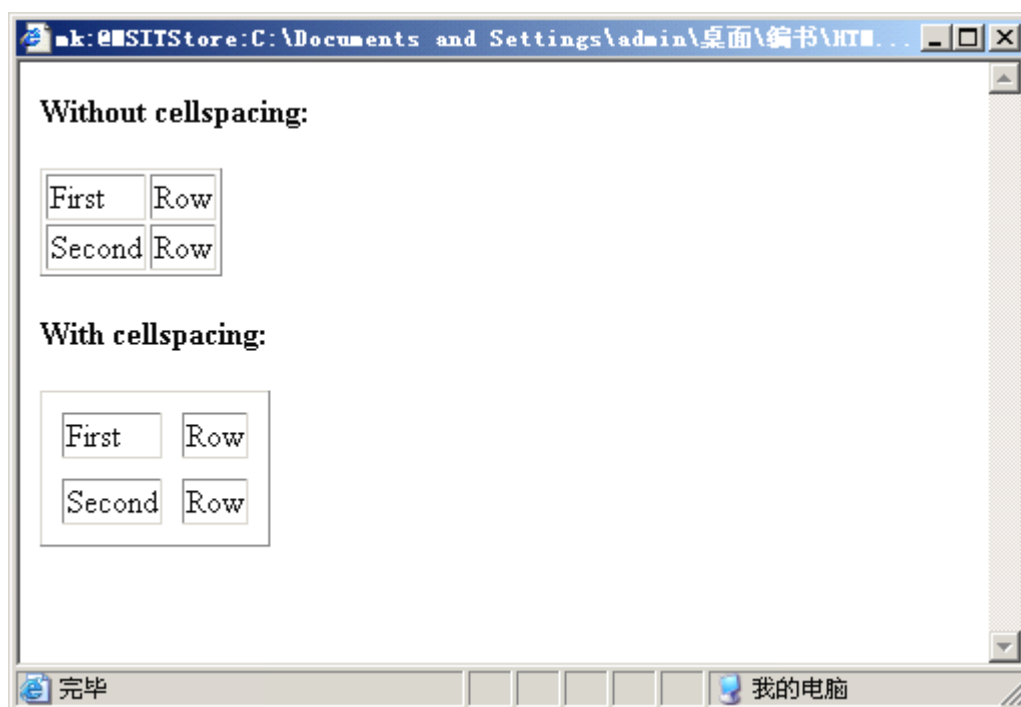
cellspacing 属性:

```
<html>
<body>
<h4>Without cellspacing:</h4>
<table border="1">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
```

```

</tr>
</table>
<h4>With cellspacing:</h4>
<table border="1" cellspacing="10">
<tr>
    <td>First</td>
    <td>Row</td>
</tr>
<tr>
    <td>Second</td>
    <td>Row</td>
</tr>
</table>
</body>
</html>

```



这个例子说明了如何使用 `cellspacing` 属性来增加单元格间距。

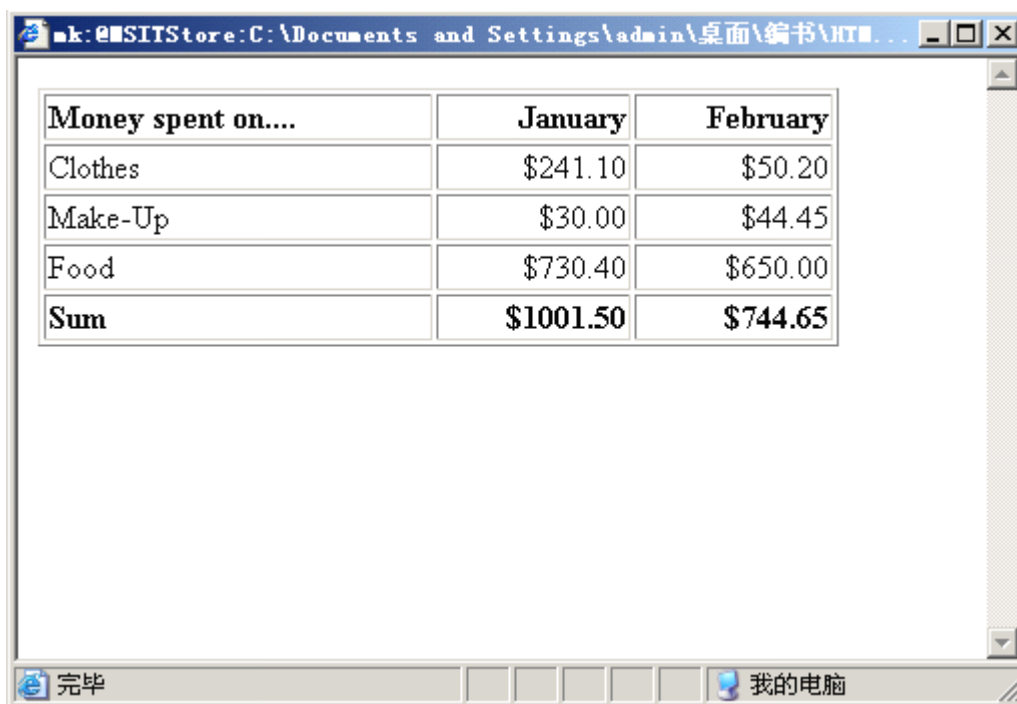
给单元格内容设置对齐方式：

```

<html>
<body>
<table width="400" border="1">
<tr>
    <th align="left">Money spent on....</th>
    <th align="right">January</th>

```

```
<th align="right">February</th>
</tr>
<tr>
  <td align="left">Clothes</td>
  <td align="right">$241.10</td>
  <td align="right">$50.20</td>
</tr>
<tr>
  <td align="left">Make-Up</td>
  <td align="right">$30.00</td>
  <td align="right">$44.45</td>
</tr>
<tr>
  <td align="left">Food</td>
  <td align="right">$730.40</td>
  <td align="right">$650.00</td>
</tr>
<tr>
  <th align="left">Sum</th>
  <th align="right">$1001.50</th>
  <th align="right">$744.65</th>
</tr>
</table>
</body>
</html>
```



The screenshot shows a web browser window with the title bar text: "k: @SITStore: C:\Documents and Settings\admin\桌面\编书\HTM...". The browser displays a table with the following content:

Money spent on....	January	February
Clothes	\$241.10	\$50.20
Make-Up	\$30.00	\$44.45
Food	\$730.40	\$650.00
Sum	\$1001.50	\$744.65

The taskbar at the bottom shows the "完毕" (Completed) button and the "我的电脑" (My Computer) icon.

这个例子说明了如何使用“align”属性来设置单元格的对齐方式，让表格好看一些。

第四章 HTML 表单页面的运用

本章目标：掌握表单基本结构<form>
掌握各种表单元素
能理解 post 和 get 两种提交方式的区别
本章重点：掌握各种表单元素
本章难点：post 和 get 两种提交方式的区别

一、HTML 表单

表单：

表单是一个能够包含表单元素的区域。

表单元素是能够让用户在表单中输入信息的元素（比如文本框，密码框，下拉菜单，单选框，复选框等等）。

表单是用<form>元素定义的：

```
<form>
<input>
<input>
</form>
```

Input:

最常用的表单标签是<input>标签。Input 的类型用 type 属性指定。最常用的 input 类型解释如下：

文本框：在表单中，文本框用来让用户输入字母、数字等等。

```
<form>
First name:
<input type="text" name="firstname">
<br>
Last name:
<input type="text" name="lastname">
</form>
```

在浏览器中显示如下：

First name:

Last name:

单选按钮：当需要用户从有限个选项中选择一个时，使用单选按钮。

```
<form>
<input type="radio" name="sex" value="male">Male
<br>
<input type="radio" name="sex" value="female">Female
</form>
```

在浏览器中显示如下：

☐ Male

☐ Female

注意，各选项中只能选取一个。

复选框：当需要用户从有限个选项中选择一个或多个时，使用复选框。

```
<form>
<input type="checkbox" name="bike">
I have a bike
<br>
<input type="checkbox" name="car">
I have a car
</form>
```

在浏览器中显示如下：

☐ I have a bike

☐ I have a car

表单的 `action` 属性和提交按钮：当用户点击提交按钮的时候，表单的内容会被提交到其他文件。表单的 `action` 属性定义了所要提交到的目的文件，该目的文件收到信息后通常进行相关的处理。

```
<form name="input" action="html_form_action.asp" method="get">
Username:
<input type="text" name="user">
<input type="submit" value="Submit">
</form>
```

在浏览器中显示如下：

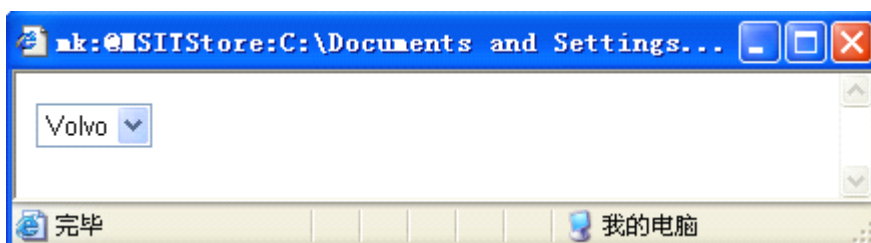
Username:

如果在上面这个文本框中输入一些字符，按下提交按钮以后，输入的字符将被提交到页面“action.asp”。

更多示例：

简单的下拉列表：

```
<html>
<body>
  <form>
    <select name="cars">
      <option value="volvo">Volvo
      <option value="saab">Saab
      <option value="fiat">Fiat
      <option value="audi">Audi
    </select>
  </form>
</body>
</html>
```

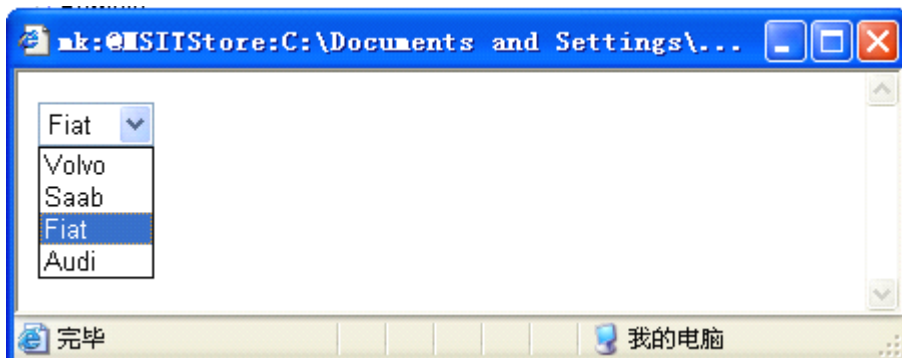


这个例子说明了在 HTML 页面如何创建下拉列表。下拉列表是可以选择的列表。

预选的下拉列表：

```
<html>
<body>
  <form>
    <select name="cars">
      <option value="volvo">Volvo
      <option value="saab">Saab
      <option value="fiat" selected>Fiat
      <option value="audi">Audi
    </select>
  </form>
</body>
```

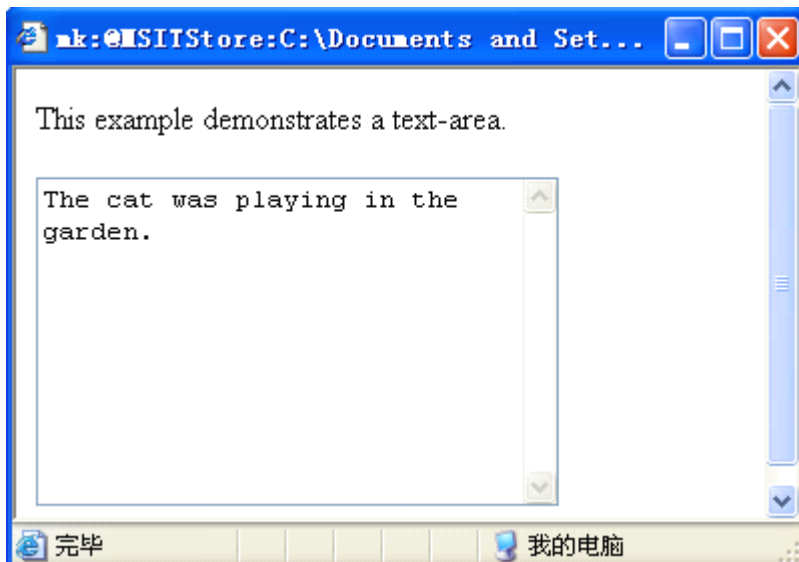
```
</html>
```



这个例子说明了如何创建一个含有预先选定元素的下拉列表。

文本域:

```
<html>
  <body>
    <p>
      This example demonstrates a text-area.
    </p>
    <textarea rows="10" cols="30">
      The cat was playing in the garden.
    </textarea>
  </body>
</html>
```



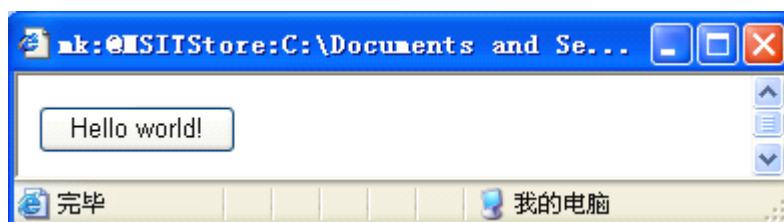
这个例子说明了如何创建文本域（多行文本），用户可以在其中输入文本。在文本域中，字符个数不受限制。

创建按钮:


```

<html>
  <body>
    <form>
      <input type="button" value="Hello world!">
    </form>
  </body>
</html>

```



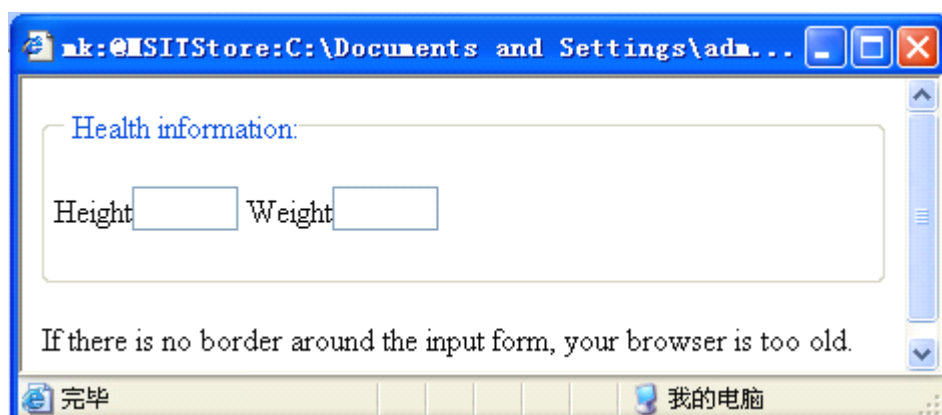
这个例子说明了如何创建按钮。按钮上的文字可以自己定义。

数据周围的标题边框:

```

<html>
  <body>
    <fieldset>
      <legend>
        Health information:
      </legend>
      <form>
        Height<input type="text" size="3">
        Weight<input type="text" size="3">
      </form>
    </fieldset>
    <p>
      If there is no border around the input form, your browser is too old.
    </p>
  </body>
</html>

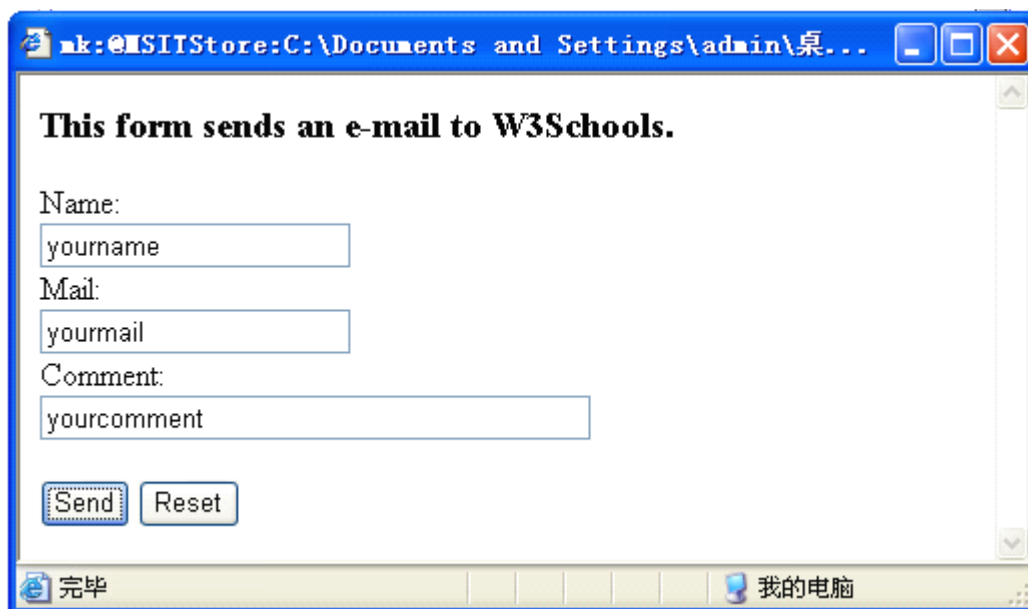
```



这个例子说明了如何在数据周围画带有标题的边框。

从表单发送电子邮件：

```
<html>
<body>
  <form action="MAILTO:someone@w3schools.com" method="post"
  enctype="text/plain">
    <h3>This form sends an e-mail to W3Schools.</h3>
    Name:<br>
    <input type="text" name="name" value="yourname" size="20">
    <br>
    Mail:<br>
    <input type="text" name="mail" value="yourmail" size="20">
    <br>
    Comment:<br>
    <input type="text" name="comment" value="yourcomment" size="40">
    <br><br>
    <input type="submit" value="Send">
    <input type="reset" value="Reset">
  </form>
</body>
</html>
```



这个例子说明了如何从一个表单发送电子邮件

第五章 使用样式表美化页面 1

本章目标：掌握在网页中使用 CSS 的方法

熟悉 CSS 的不同选择器的使用方法

熟悉字体属性：font-family, font-size, font-style,
font-weight

熟悉文本属性：text-align, text-indent, text-decoration,
text-transform, vertical-align, word-spacing,
letter-spacing

本章重点：掌握在网页中使用 CSS 的方法

本章难点：文本属性，字体属性

一、CSS 的工作原理

在这一节，你将学习如何制作自己的第一个样式表。你将了解基本的 CSS 模型，以及在 HTML 文档里使用 CSS 所必需的代码。

级联样式表（CSS）里用到的许多 CSS 属性都与 HTML 属性相似，所以，假如你熟悉采用 HTML 进行布局的话，那么这里的许多代码你都不会感到陌生。我们先来看一个具体的例子。

基本的 CSS 语法：

比方说，我们要用红色作为网页的背景色：

用 **HTML** 的话，我们可以这样：

```
<body bgcolor="#FF0000">
```

用 **CSS** 的话，我们可以这样获得同样的效果：

```
body {background-color: #FF0000;}
```

你会注意到，HTML 和 CSS 的代码颇有几分相似。上例也向你展示了基本的 CSS 模型：

`selector {property: value;}`



选择器：
表明花括号
中的属性设
置将应用于
哪些HTML元素
例如“body”

属性：
例如用于设置
背景色的属性
“background-color”
等等。

值：
比如说
background-color
属性的值可以是
“#FF0000”
代表红色）。

但是把 CSS 代码放在哪里呢？这正是我们下面要讲的

为一个 HTML 文档应用 CSS：

为 HTML 文档应用 CSS，有三种方法可供选择。下面对这三种方法进行了概括。我们建议你第三种方法（即外部样式表）予以关注。

方法 1：行内样式表（style 属性）

为 HTML 应用 CSS 的一种方法是使用 HTML 属性 style。我们在上例的基础之上，通过行内样式表将页面背景设为红色：

```
<html>
  <head>
    <title>例子</title>
  </head>
  <body style="background-color: #FF0000;">
    <p>这个页面是红色的</p>
  </body>
</html>
```

方法 2：内部样式表（style 元素）

为 HTML 应用 CSS 的另一种方法是采用 HTML 元素 style。比如像这样：

```
<html>
  <head>
    <title>例子</title>
    <style type="text/css">
      body {background-color: #FF0000;}
    </style>
  </head>
  <body>
    <p>这个页面是红色的</p>
  </body>
</html>
```

方法 3：外部样式表（引用一个样式表文件）

我们推荐采用这种引用外部样式表的方法。在本教程之后的例子中，我们将全部采用该方法。外部样式表就是一个扩展名为 **css** 的文本文件。跟其他文件一样，你可以把样式表文件放在 Web 服务器上或者本地硬盘上。例如，比方说你的样式表文件名为 **style.css**，它通常被存放于名为 **style** 的目录中。就像下面这样：



现在的问题是：如何在一个 HTML 文档里引用一个外部样式表文件（style.css）呢？答案是：在 HTML 文档里创建一个指向外部样式表文件的链接（link）即可，就像下面这样：

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

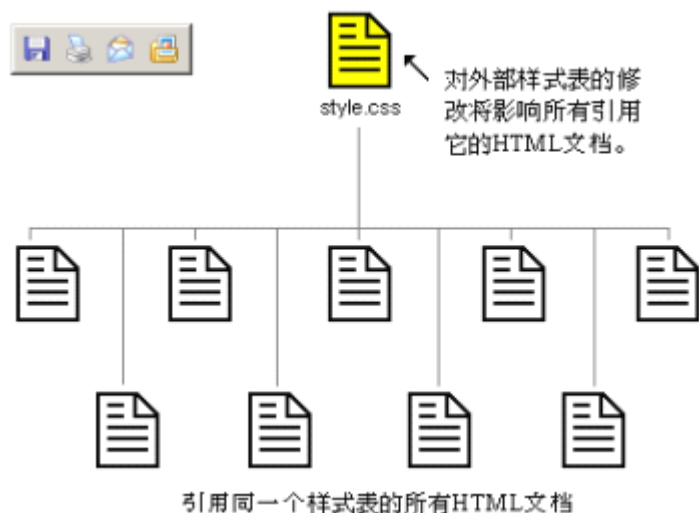
注意要在 href 属性里给出样式表文件的地址

这行代码必须被插入 HTML 代码的头部（header），即放在标签<head>和标签</head>之间。就像这样：

```
<html>
  <head>
    <title>我的文档</title>
    <link rel="stylesheet" type="text/css" href="style/style.css" />
  </head>
  <body>
    ...
```

这个链接告诉浏览器：在显示该 HTML 文件时，应使用给出的 CSS 文件进行布局。

这种方法的优越之处在于：多个 HTML 文档可以同时引用一个样式表。换句话说，可以用一个 CSS 文件来控制多个 HTML 文档的布局。



这一方法可以令你省去许多工作。例如，假设你要修改某网站的所有网页（比方说有 100 个网页）的背景颜色，采用外部样式表可以避免你手工一一修改这 100 个 HTML 文档的工作。采用外部样式表，这样的修改只需几秒钟即可搞定——修改外部样式表文件里的代码即可。

让我们来实践刚刚所学到的知识，自己试试看：

打开记事本（或其他文本编辑器），创建两个文件——一个 HTML 文件，一个 CSS 文件——它们的内容如下：

default.htm:

```
<html>
  <head>
    <title>我的文档</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <h1>我的第一个样式表</h1>
  </body>
</html>
```

style.css:

```
body {
  background-color: #FF0000;
}
```

然后，把这两个文件放在同一目录下。记得在保存文件时使用正确的扩展名（分别为“htm”和“css”）。用浏览器打开 **default.htm**，你所看到的页面应该具有红色背景。恭喜！你已经完成了自己的第一个样式表！

二、 元素的分类与标识（class 和 id）

有时，你希望对特定元素或者特定一组元素应用特殊的样式。在这一节，我们将深入学习如何利用 class 和 id 来为所选元素指定属性。

如何实现为网站上许多标题中的某一个单独应用颜色？如何实现把网站上的链接分为不同的类，并对各类链接分别应用不同的样式？这只是本节将解决的诸多问题中的最具代表性的两个问题。

用 class 对元素进行分类：

比方说，我们有两个由链接组成的列表，它们分别是用于制造白葡萄酒和红葡萄酒的葡萄。其 HTML 代码如下：

```
<p>制造白葡萄酒的葡萄： </p>
<ul>
<li><a href="ri.htm">雷司令（Riesling） </a></li>
<li><a href="ch.htm">夏敦埃（Chardonnay） </a></li>
<li><a href="pb.htm">白比诺（Pinot Blanc） </a></li>
</ul>

<p>制造红葡萄酒的葡萄： </p>
<ul>
<li><a href="cs.htm">卡百内索维农（Cabernet Sauvignon） </a></li>
<li><a href="me.htm">墨尔乐（Merlot） </a></li>
<li><a href="pn.htm">黑比诺（Pinot Noir） </a></li>
</ul>
```



现在，我们希望白葡萄酒的链接全部显示为黄色，红葡萄酒的链接全部显示为红色，其余的链接显示为缺省的兰色。为了实现这一要求，我们将链接分为两类。对链接的分类是通过为链接设置 HTML 属性 class 实现的。

参加如下代码：

```
<p>制造白葡萄酒的葡萄： </p>
<ul>
<li><a href="ri.htm" class="whitevine">雷司令 (Riesling) </a></li>
<li><a href="ch.htm" class="whitevine">夏敦埃 (Chardonnay) </a></li>
<li><a href="pb.htm" class="whitevine">白比诺 (Pinot Blanc) </a></li>
</ul>

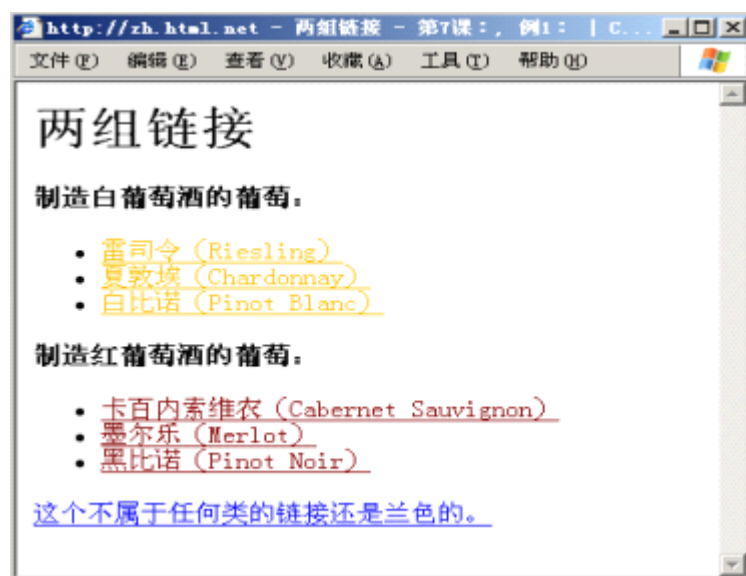
<p>制造红葡萄酒的葡萄： </p>
<ul>
<li><a href="cs.htm" class="redwine">卡百内索维农 (Cabernet Sauvignon) </a></li>
<li><a href="me.htm" class="redwine">墨尔乐 (Merlot) </a></li>
<li><a href="pn.htm" class="redwine">黑比诺 (Pinot Noir) </a></li>
</ul>
```

然后，我们就可以为白葡萄酒和红葡萄酒的链接分别应用不同的风格了。

```
a {
    color: blue;
}

a.whitevine {
    color: #FFBB00;
}

a.redwine {
    color: #800000;
}
```



如上例所示，你可以通过在样式表里利用 **.classname** 来为属于某一类的元素定义 CSS 属性。

利用 id 标识元素：

除了可以对元素进行分类以外，你还可以标识单个元素。这是通过 HTML 属性 id 实现的。HTML 属性 id 的特别之处在于，在同一 HTML 文档中不能有两个具有相同 id 值的元素。文档中的每个 id 值都必须是唯一的。在其他情况下，你应该使用 class 属性。下面，我们来看一个使用 id 属性的例子：

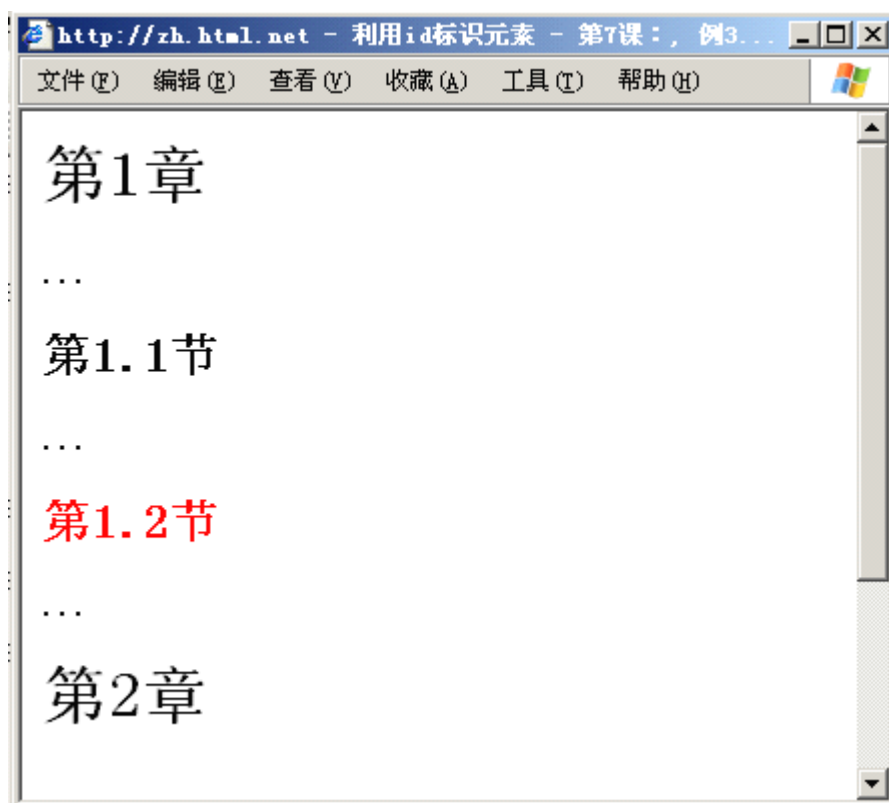
```
<h1>第1章</h1>
...
<h2>第1.1节</h2>
...
<h2>第1.2节</h2>
...
<h1>第2章</h1>
...
<h2>第2.1节</h2>
...
<h2>第2.1.1小节</h2>
...
```

上例是一个文章的各章节的标题。我们自然可以为其中每一章节指定一个 id（如下）：

```
<h1 id="c1">第1章</h1>
...
<h2 id="c1-1">第1.1节</h2>
...
<h2 id="c1-2">第1.2节</h2>
...
<h1 id="c2">第2章</h1>
...
<h2 id="c2-1">第2.1节</h2>
...
<h3 id="c2-1-2">第2.1.1节</h3>
...
```

假如我们要求第 1.2 节显示为红色，那么 CSS 可以这样写：

```
#c1-2 {
    color: red;
}
```



如上例所示，你可以在样式表里通过#id 为特定元素定义 CSS 属性

三、 字体属性

这一节，你将学习字体以及如何用 CSS 来设置字体。我们还会考虑如何解决“网站所选的字体仅当访问者的 PC 上安装有该字体时才会被显示”这一难题。本节将对下列 CSS 属性进行讲解：

- [font-family](#)
- [font-style](#)
- [font-variant](#)
- [font-weight](#)
- [font-size](#)
- [font](#)

字体族[font-family]:

CSS 属性 font-family 的作用是设置一组按优先级排序的字体列表，如果该列表中的第一个字体未在访问者计算机上安装，那么就尝试列表中的下一个字体，依此类推，直到列表中的某个字体是已安装的。

有两种类型的名称可用于分类字体：字体族名称（family-name）和族类名称（generic family）。下面来解释这两个术语。

字体族名称（family-name）:

字体族名称（就是我们通常所说的“字体”）的例子包括“Arial”、“Times New Roman”、“宋体”、“黑体”等等。

族类 (generic family):

一个族类是一组具有统一外观的字体族。sans-serif 就是一例，它代表一组没有“脚”的字体。

下面我们通过三个族类的例子来进行解释：

Times New Roman
Garamond
Georgia

这三个字体族属于 serif 族类。
它们的共同特点是，
笔画两端有“脚”。

Trebuchet
Arial
Verdana

这三个字体族属于 sans-serif 族类。
它们的共同特点是，
笔画两端没有“脚”。

Courier
Courier New
Andale Mono

这三个字体族属于 monospace 族类。
它们的共同特点是，
所有字符的宽度都一样。

你在给出字体列表时，自然应把首选字体放在前面、把候选字体放在后面。建议你在列表的最后给出一个族类 (generic family)，这样，当没有一个指定字体可用时，页面至少可以采用一个相同族类的字体来显示。

下面是一个按优先级排列的字体列表的例子：

```
<body>
  <h1>大标题是 Arial 字体</h1>
  <h2>而次标题是 Times New Roman 字体</h2>
</body>
```



h1 标题将采用 Arial 字体显示。如果访问者的计算机未安装 Arial，那么就使用 Verdana 字体。假如 Verdana 字体也没安装的话，那么将采用一个属于 **sans-serif** 族类的字体来显

示这个 h1 标题。注意我们为“Times New Roman”采用的写法：因为其中包含空格，所以我们用引号将它括起来。

字体样式[font-style]:

CSS 属性 font-style 定义所选字体的显示样式：**normal**(正常)、**italic**(斜体)或 **oblique**(倾斜)。在下例中，所有 h2 标题都将显示为斜体。

```
h1 {font-family: arial, verdana, sans-serif;}
h2 {font-family: "Times New Roman", serif; font-style: italic;}
```



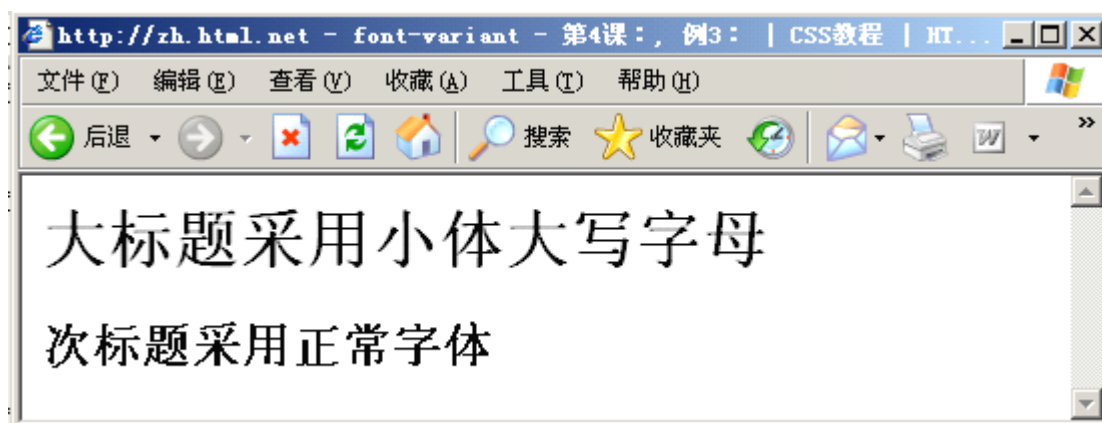
字体变化[font-variant]:

CSS 属性 font-variant 的值可以是：**normal**（正常）或 **small-caps**（小体大写字母）。**small-caps** 字体是一种以小尺寸显示的大写字母来代替小写字母的字体。不太明白？我们来看几个例子：

Sans Book SC	Sans Bold SC	Serif Book SC	Serif Bold SC
ABCABC	ABCABC	ABCABC	ABCABC

如果 font-variant 属性被设置为 **small-caps**，而没有可用的支持小体大写字母的字体，那么浏览器多半会将文字显示为正常尺寸（而不是小尺寸）的大写字母。

```
h1 {font-variant: small-caps;}
h2 {font-variant: normal;}
```



字体浓淡[font-weight]:

CSS 属性 font-weight 指定字体显示的浓淡程度。其值可以是 **normal**（正常）或 **bold**（加粗）。有些浏览器甚至支持采用 100 到 900 之间的数字（以百为单位）来衡量字体的浓淡。

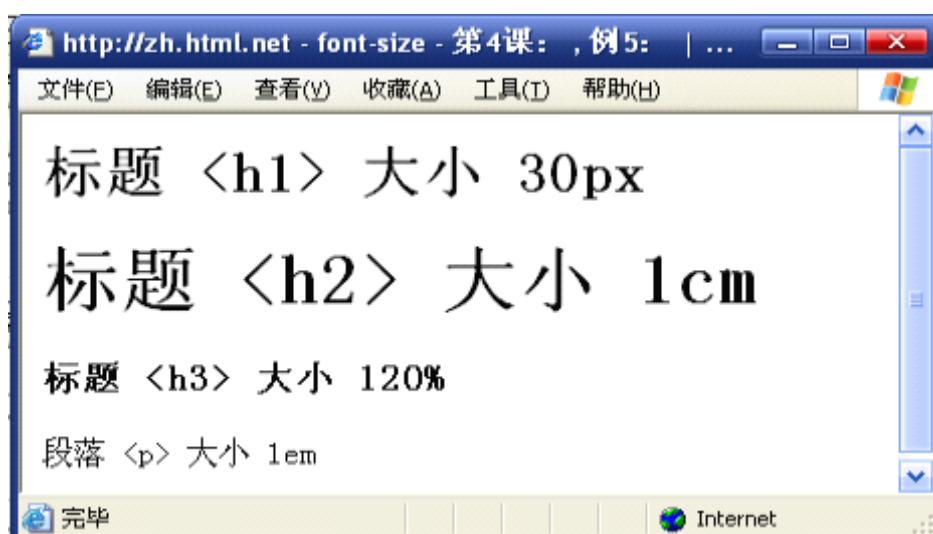
```
p {font-family: arial, verdana, sans-serif;}
td {font-family: arial, verdana, sans-serif; font-weight: bold;}
```

字体大小[font-size]:

字体的大小用 CSS 属性 font-size 来设置。

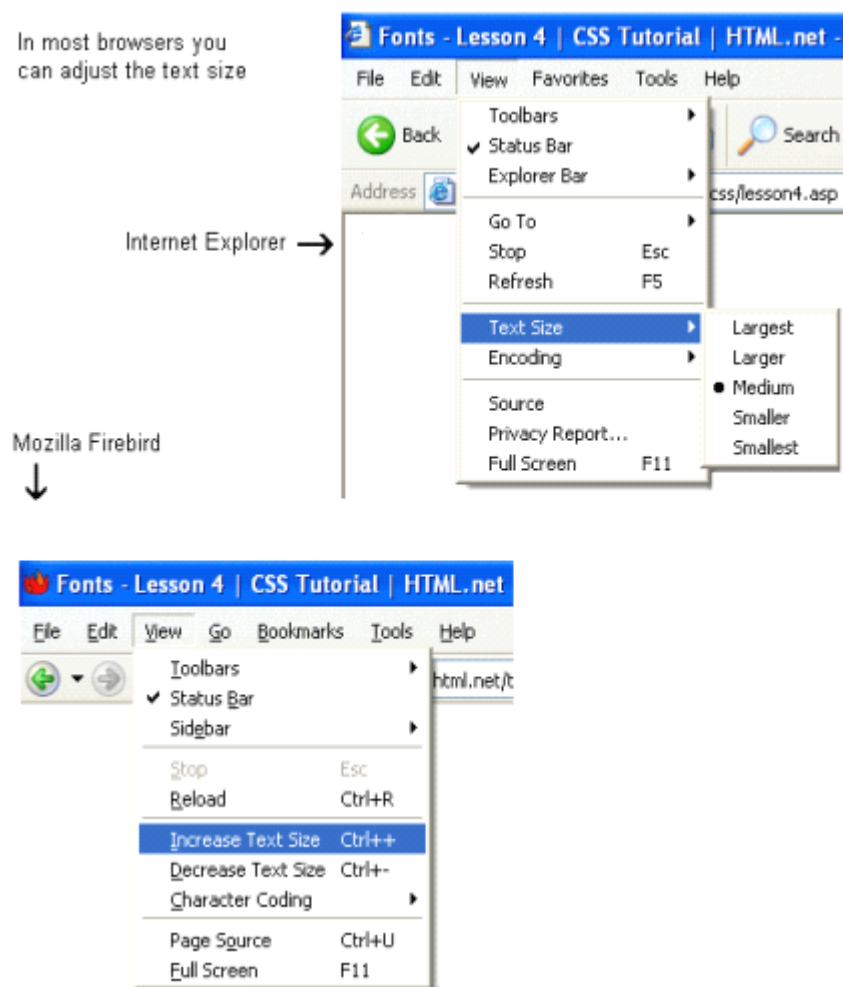
字体大小可通过多种不同单位（比如像素或百分比等）来设置。在本教程中，我们将关注于最常用和最合适的单位。比如：

```
h1 {font-size: 30px;}
h2 {font-size: 12pt;}
h3 {font-size: 120%;}
p {font-size: 1em;}
```



上面四种单位有着本质的区别。‘**px**’和‘**pt**’将字体设置为固定大小，而‘**%**’和‘**em**’允许页面浏览者自行调整字体的显示尺寸。有些页面浏览者可能是残疾者、年长者、视力不佳者，或者他所使用的电脑显示屏显示质量差。为了令你的网站对所有人都具有良好的可用性（accessibility），你应采用像‘**%**’或‘**em**’这种允许用户调节字体显示大小的单位。

下面你能看到我们展示如何在 Mozilla Firefox 和 Internet Explorer 里调整字体大小。自己试试看！这个功能很不错吧？



缩写[font]:

CSS 属性 font 是上述各有关字体的 CSS 属性的缩写用法。

比如说下面四行应用于 p 元素的代码：

```
p {
    font-style: italic;
    font-weight: bold;
    font-size: 30px;
    font-family: arial, sans-serif;
}
```

如果用 font 属性的话，上述四行代码可简化为：

```
p {  
    font: italic bold 30px arial, sans-serif;  
}
```

font 属性的值应按以下次序书写：

font-style | font-variant | font-weight | font-size | font-family

小结：

在这一节，你学习了有关字体设置的用法。记住：CSS 的一个主要优势就是可以在任何时候设置字体，你花几分钟就可以改变整个网站的字体。CSS 节省时间，而且把事情简化。在下一节，我们将学习文本（text）。

四、 文本属性

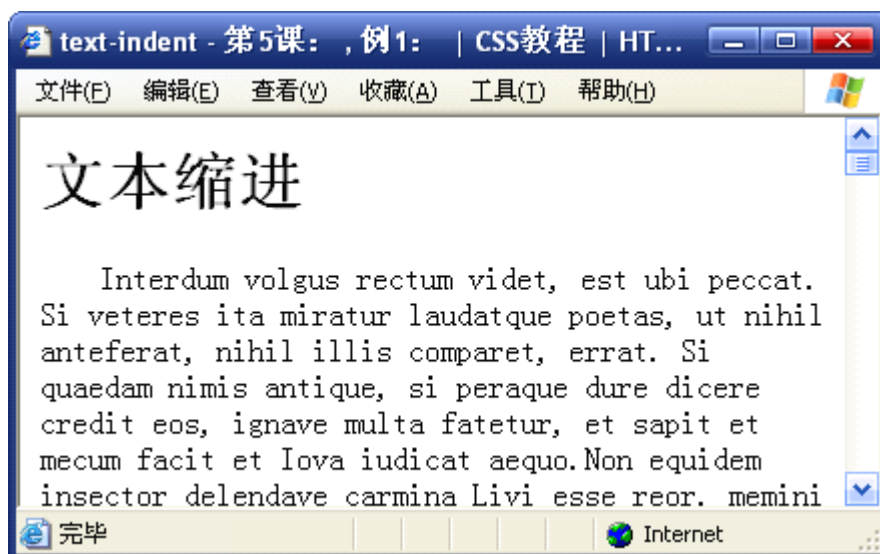
文本的显示格式与式样对于网页设计师来说是一个重要问题。这一节将向你介绍 CSS 在文本布局方面令人激动的特性。本节将对下列 CSS 属性进行讲解：

- [text-indent](#)
- [text-align](#)
- [text-decoration](#)
- [letter-spacing](#)
- [text-transform](#)

文本缩进[text-indent]：

CSS 属性 text-indent 用于为段落设置首行缩进，以令其具有美观的格式。在下例中，我们为采用 p 元素的段落应用了 30 像素的首行缩进。

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

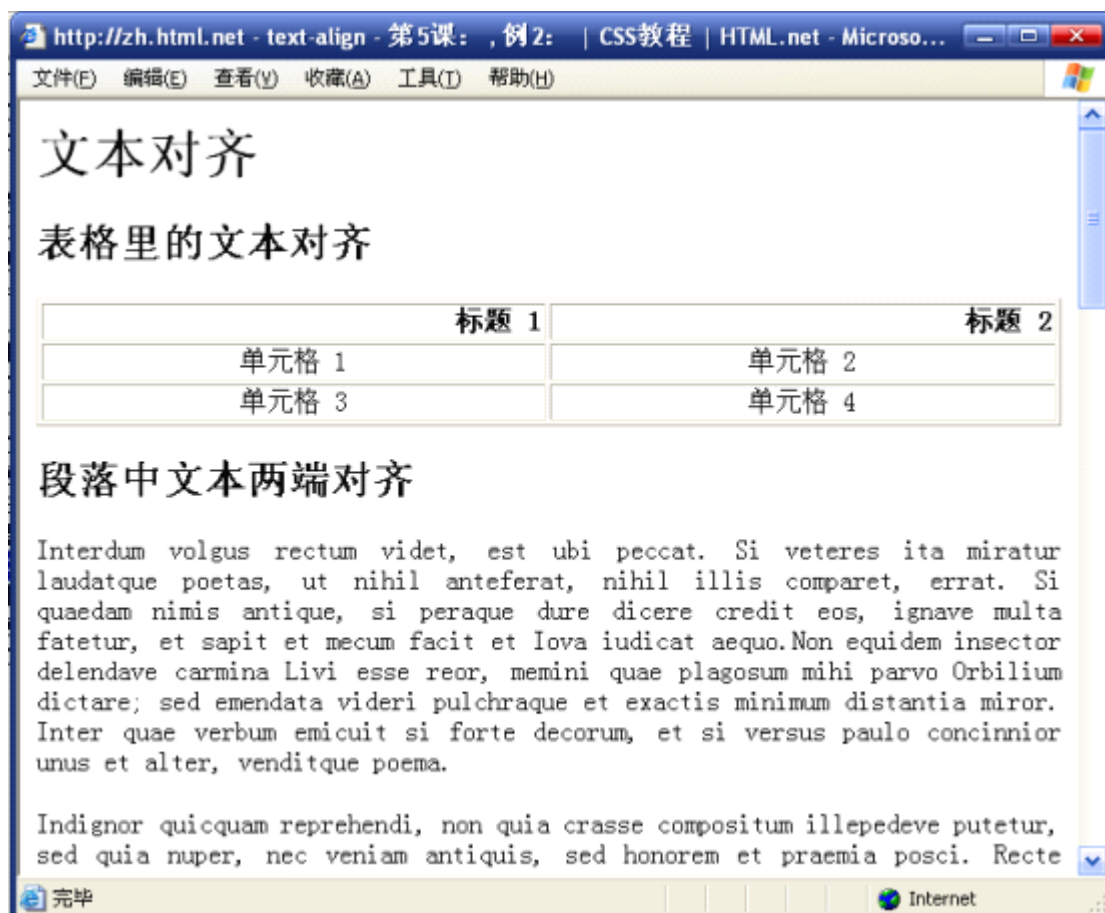


文本对齐[text-align]:

CSS 属性 text-align 与 HTML 属性 align 的功能相同。该属性的值可以是：**left**（左对齐）、**right**（右对齐）或者 **center**（居中）。除了上面三种选择以外，你还可以将该属性的值设为 **justify**（两端对齐），即伸缩行中的文字以左右靠齐。报刊杂志经常采用这种布局。

在下例中，标题（th）中的文字被设置为右对齐，而表中数据（td）被设置为居中。正常的文本段落被设置为两端对齐。

```
th {  
    text-align: right;  
}  
  
td {  
    text-align: center;  
}  
  
p {  
    text-align: justify;  
}
```

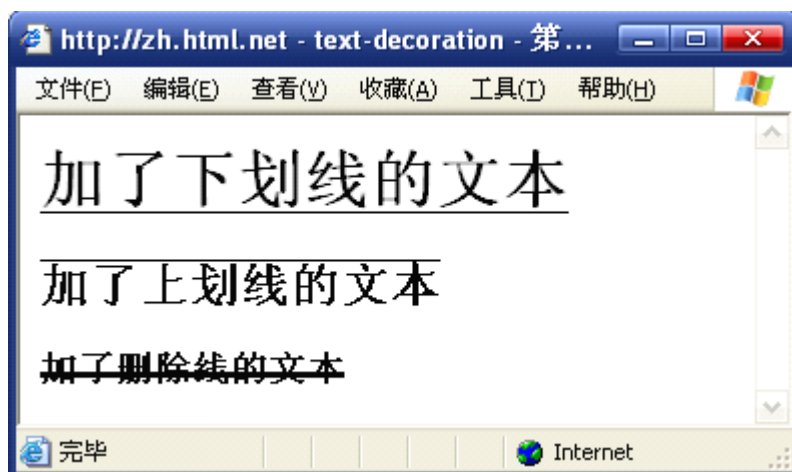
文本装饰[text-decoration]:

CSS 属性 text-decoration 令我们可以为文本增添不同的“装饰”或“效果”。例如，你可以为文本增添下划线、删除线、上划线等等。在接下来的例子中，我们为 h1 标题增添了下划线，为 h2 标题增添了上划线，为 h3 标题增添了删除线。

```
h1 {
    text-decoration: underline;
}

h2 {
    text-decoration: overline;
}

h3 {
    text-decoration: line-through;
}
```

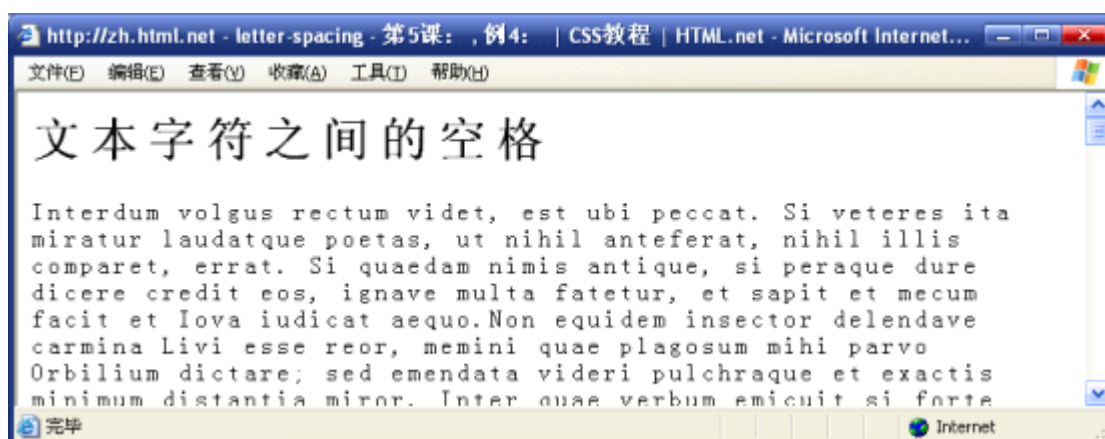


字符间距[letter-spacing]:

CSS 属性 letter-spacing 用于设置文本的水平字间距。我们可以把期望的字间距宽度作为这个属性的值。例如，假如你希望 p 元素里的文本段落的字间距为 3 个像素，而 h1 标题的字间距为 6 个像素，代码可以这样写：

```
h1 {
    letter-spacing: 6px;
}

p {
    letter-spacing: 3px;
}
```



文本转换[text-transform]:

CSS 属性 text-transform 用于控制文本的大小写。无论字母本来的大小写，你可以通过该属性令它首字母大写(capitalize)、全部大写(uppercase)或者全部小写(lowercase)。比如，单词“headline”在展现给网页浏览者时，可以是“HEADLINE”或者“Headline”。text-transform 属性有四个可选值：

Capitalize: 将每个单词的首字母转换为大写。例如：“john doe”将被转换为“John Doe”。

Uppercase: 所有字母都转换为大写。例如: “john doe” 将被转换为 “JOHN DOE”。

Lowercase: 所有字母都转换为小写。例如: “JOHN DOE” 将被转换为 “john doe”。

None: 不作任何转换——文本怎么写的就怎么显示。

来举个例子, 我们将使用一个姓名列表。所有姓名都用(列表项) 标签来标记。我们希望对姓名采用首字母大写的方式, 而对标题采用全部大写的方式。

查看过该例的 HTML 代码后你会发现, 其实在 HTML 代码里我们写的姓名和标题全部都是小写。

```
h1 {  
    text-transform: uppercase;  
}  
  
li {  
    text-transform: capitalize;  
}
```

```
<body>  
  <h1>这个标题采用大写字母</h1>  
  <ul>  
    <li>peter hanson</li>  
    <li>max larsen</li>  
    <li>joe doe</li>  
    <li>paula jones</li>  
    <li>monica lewinsky</li>  
    <li>donald duck</li>  
  </ul>  
  <p>注意, 我们用 CSS 实现了令所有人名的首字母大写。</p>  
</body>
```



第六章 使用样式表美化页面 2

本章目标：熟悉显示属性：**display**
熟悉边框属性：**Border, border-style** 等
熟悉定位属性：**top, Width, Height, Left**
本章重点：如何使用 CSS 样式表进行布局修饰
本章难点：边框属性

一、 显示属性

显示属性允许使用四个值中的一个来定义一个元素：

block：在元素的前和后都会有换行

inline：在元素的前和后都不会有换行

list-item：与 block 相同，但增加了目录项标记

none：没有显示

下面我们来看一个分级属性的例子，代码如下所示：

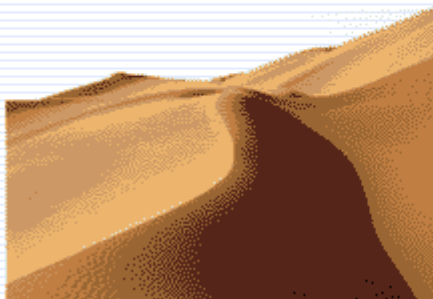
```
p{display: block; white-space: normal} ↓  
em{display: inline} ↓  
li{display: list-item; list-style: square} ↓  
img{display: block} ↗
```

```
<body>  
  <p>  
    <em>sample</em>text<em>sample</em>text<em>sample</em>  
    text<em>sample</em> text<em>sample</em>  
  </p>  
  <ul>  
    <li>list-item 1</li>  
    <li>list-item 2</li>  
    <li>list-item 3</li>  
  </ul>  
  <p><img src= “ss01068. jpg” width= “280” height= “185”  
    alt= “invisible” >  
  </p>  
</body>
```

上段代码的显示效果如下图：

```
sampletextsampletextsampletextsampletextsample
```

- list-item 1
- list-item 2
- list-item 3



我们看到由于定义了<p>的属性为 Block, 所以文本、列表、图片都在不同的位置上打开, Inline 属性使文本不折行, list-style-type 的属性值为 square 使列表项前的符号为方块; 如果我们在上面的代码中做一些改动, 则将以另一种效果显示, 我们在<head>中把“EM”的 display 属性值改为 block, 使其都在新的位置打开; li 的“list-style”属性值改为“Upper-roman” (大写罗马符号), img 的“display”属性值改为“none” (让图片不显示)。修改后的显示效果如下图:

```
sample  
text  
sample  
text  
sample  
text  
sample  
text  
sample
```

- I. list-item 1
- II. list-item 2
- III. list-item 3

二、 边框属性

边框 (border) 可以有多种用途, 比如作为装饰元素或者作为划分两物的分界线。在设置边框方面, CSS 为你提供了无尽选择。

- [border-width](#)
- [border-color](#)
- [border-style](#)

边框宽度[border-width]:

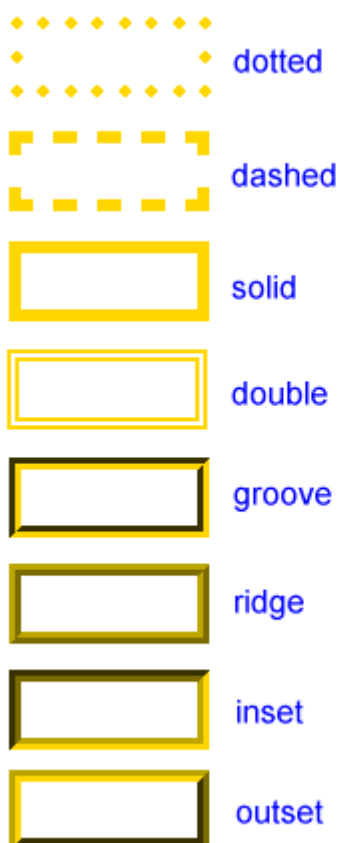
边框宽度由 CSS 属性 `border-width` 定义，其值可以是“thin”（薄）、“medium”（普通）或“thick”（厚）等，也可以是像素值。如下图所示：

**边框颜色[border-color]:**

CSS 属性 `border-color` 用于定义边框的颜色。其值就是正常的颜色值，例如：“#123456”、“rgb(123,123,123)”、“yellow”等。

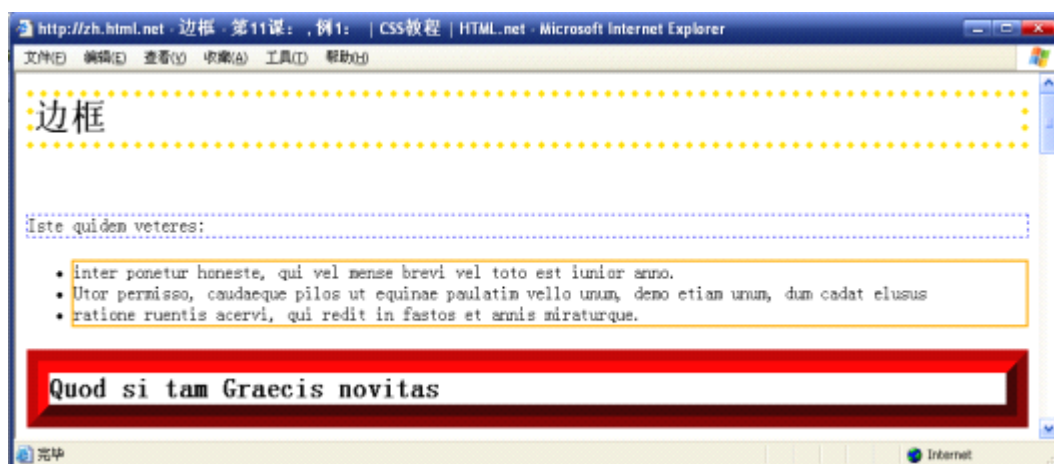
边框样式[border-style]:

边框样式有多种可供选择。下图显示了 8 种不同样式的边框在 Internet Explorer 5.5 里的实际显示效果。在这个例子里，我们为这 8 种边框都选择了“金色（gold）”作为边框颜色、“厚(thick)”作为边框宽度。当然，这只是个例子，你可以为边框设置别的颜色和厚度。如果你不想有任何边框，可以为它取值为“none”或者“hidden”。

**一些示例:**

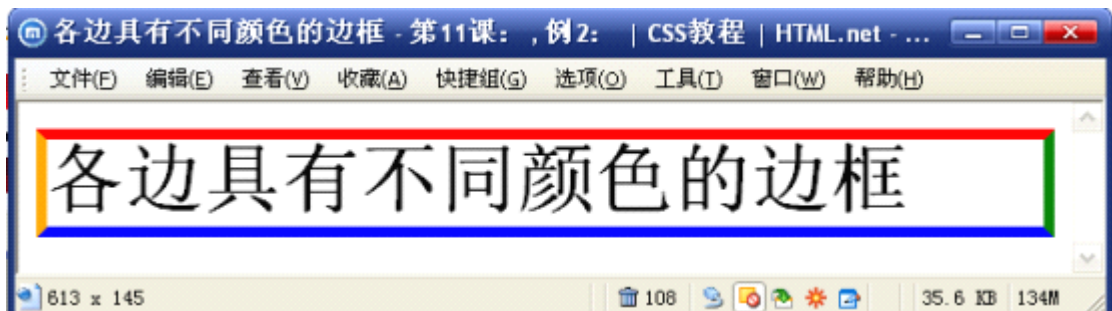
我们可以将上面三个有关边框的 CSS 属性组合起来使用，从而制造出多种多样的变化。来举个例子，我们要为一个文档中的 `h1`、`h2`、`u1` 和 `p` 等元素分别定义不同的边框。尽管其显示效果也许并不那么美观，但是它很好地向你展示了多种变化的可能：

```
h1 {  
    border-width: thick;  
    border-style: dotted;  
    border-color: gold;  
}  
  
h2 {  
    border-width: 20px;  
    border-style: outset;  
    border-color: red;  
}  
  
p {  
    border-width: 1px;  
    border-style: dashed;  
    border-color: blue;  
}  
  
ul {  
    border-width: thin;  
    border-style: solid;  
    border-color: orange;  
}
```



我们也可以为上边框、下边框、右边框、左边框分别指定特定的 CSS 属性。具体做法如下例所示：

```
h1 {  
    border-top-width: thick;  
    border-top-style: solid;  
    border-top-color: red;  
  
    border-bottom-width: thick;  
    border-bottom-style: solid;  
    border-bottom-color: blue;  
  
    border-right-width: thick;  
    border-right-style: solid;  
    border-right-color: green;  
  
    border-left-width: thick;  
    border-left-style: solid;  
    border-left-color: orange;  
}
```



缩写[border]:

跟许多其他属性一样，你也可以将有关边框的 CSS 属性缩写为一个 border 属性。让我们看一个例子：

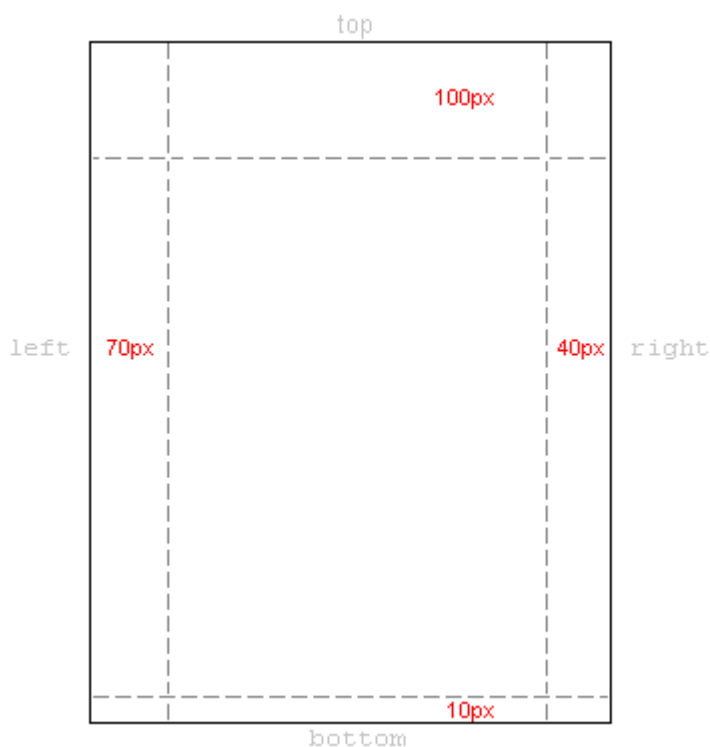
```
p {  
    border-width: 1px;  
    border-style: solid;  
    border-color: blue;  
}
```

可被缩写为：


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

外边距和内边距：

一个元素有上（top）、下（bottom）、左（left）、右（right）四个边。外边距（margin）表示从一个元素的边到相邻元素(或者文档边界)之间的距离。在下面这个例子中，我们将了解如何为文档本身（即 body 元素）定义外边距。下图显示了我们对外边距的要求。

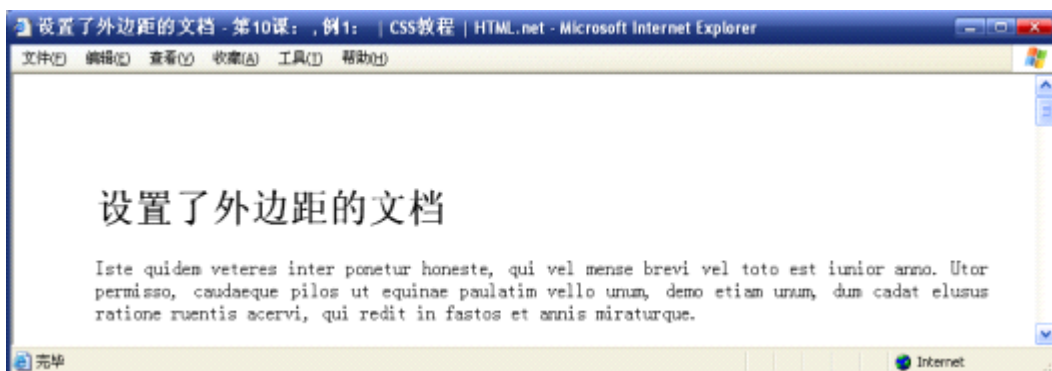


满足上述要求的 CSS 代码如下：

```
body {  
    margin-top:100px;  
    margin-right:40px;  
    margin-bottom:10px;  
    margin-left:70px;  
}
```

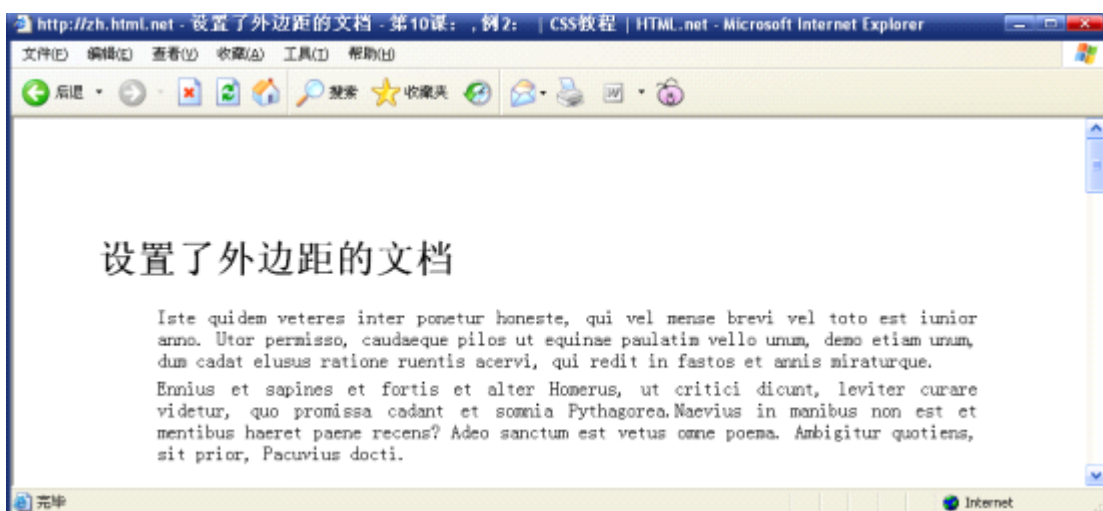
或者你也可以采用一种较优雅的缩写形式：

```
body {  
    margin: 100px 40px 10px 70px;  
}
```



几乎所有元素都可以采用跟上面一样的方法来设置外边距。例如，我们可以为所有用<p>标记的文本段落定义外边距：

```
body {  
    margin: 100px 40px 10px 70px;  
}  
  
p {  
    margin: 5px 50px 5px 50px;  
}
```

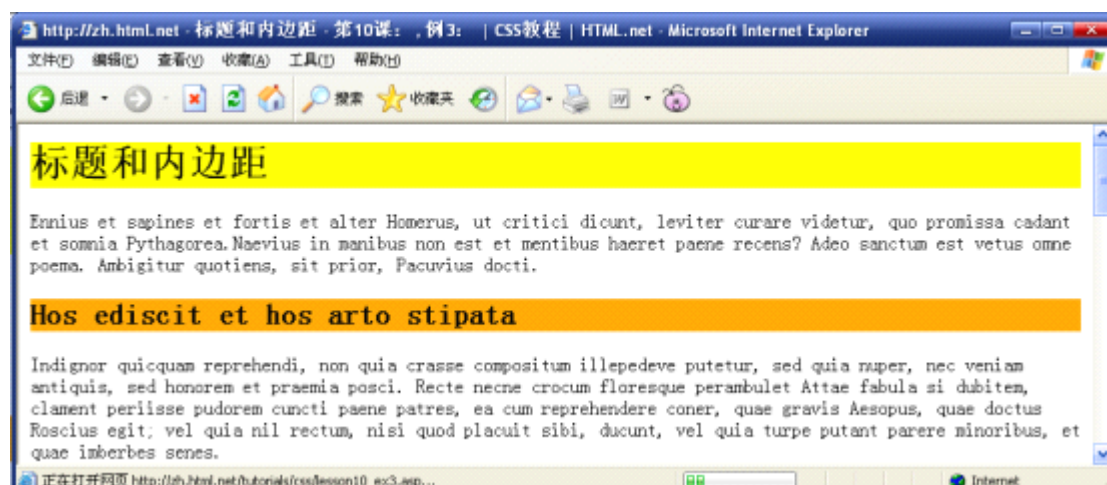


为元素设置内边距：

内边距（padding）也可以被理解成“填充物”。这样理解是合理的，因为内边距并不影响元素间的距离，它只定义元素的内容与元素边框之间的距离。

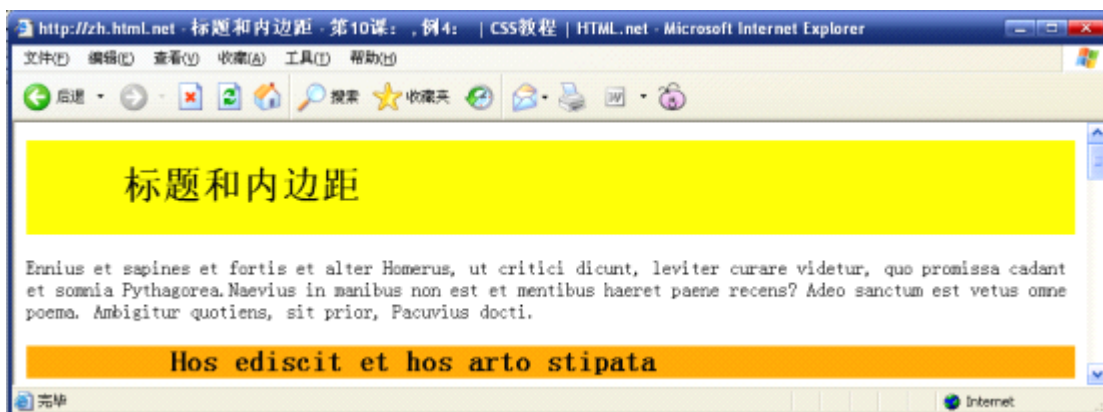
下面我们通过一个简单的例子来说明内边距的用法。在这个例子中，所有标题都具有背景色：

```
h1 {  
    background: yellow;  
}  
  
h2 {  
    background: orange;  
}
```



通过为标题设置内边距，你可以控制在标题文本周围填充多少空白：

```
h1 {  
    background: yellow;  
    padding: 20px 20px 20px 80px;  
}  
  
h2 {  
    background: orange;  
    padding-left: 120px;  
}
```

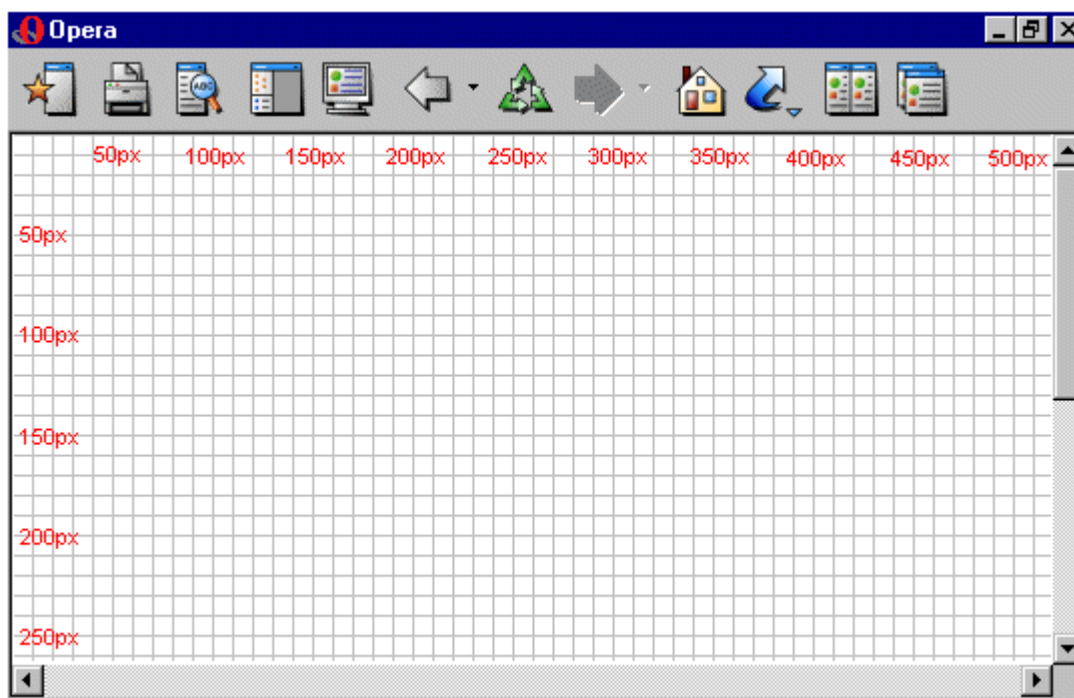


三、 定位属性

CSS 定位令你可以将一个元素精确地放在页面上你所指定的地方。

CSS 定位的原理：

把浏览器窗口想象成一个坐标系统：



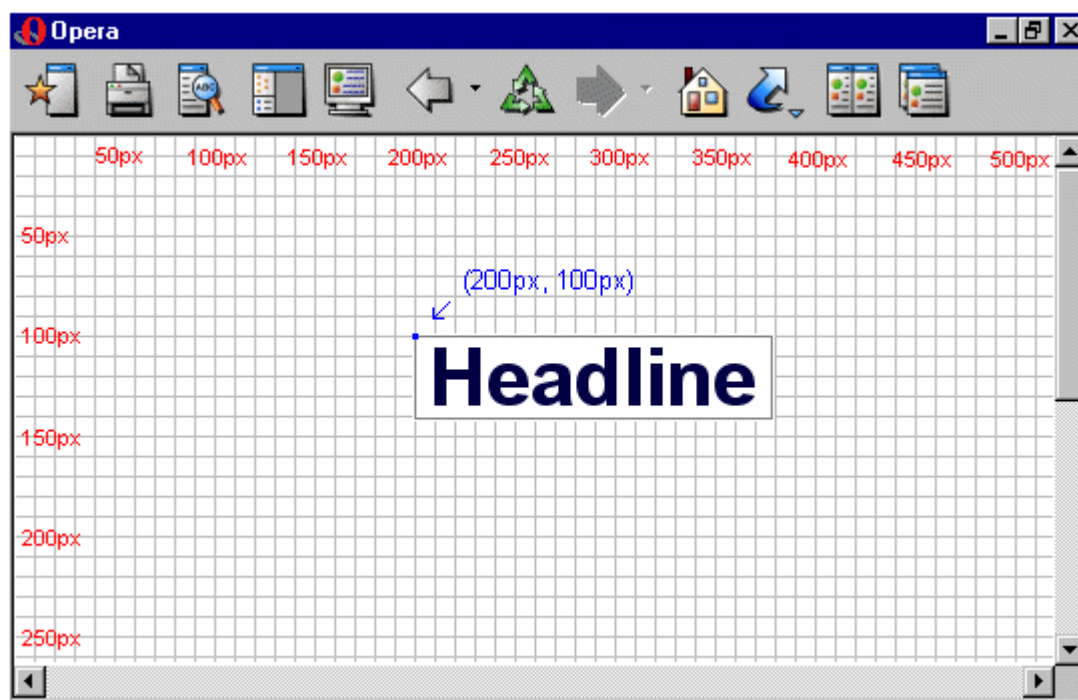
假设我们要放置一个标题：

Headline

如果我们要把这个标题放置在距文档顶部 100 像素、左边 200 像素的地方，我们可以在 CSS 中输入以下代码：

```
h1 {  
    position: absolute;  
    top: 100px;  
    left: 200px;  
}
```

得到的显示效果如下：



正如你所看到的，采用 CSS 定位技术来放置元素是非常精确的。相对于使用表格、透明图像或其他方法而言，CSS 定位要简单得多。

绝对定位：

一个采用绝对定位的元素不获得任何空间。这意味着：该元素在被定位后不会留下空位。要对元素进行绝对定位，应将 `position` 属性的值设为 **absolute**。接着，你可以通过属性 **left**、**right**、**top** 和 **bottom** 来设定将盒子放置在哪里。

举个绝对定位的例子，假如我们要在文档的四个角落各放置一个盒子：

```
#box1 {  
    position: absolute;  
    top: 50px;  
    left: 50px;  
}  
  
#box2 {  
    position: absolute;  
    top: 50px;  
    right: 50px;  
}  
  
#box3 {  
    position: absolute;  
    bottom: 50px;  
    right: 50px;  
}  
  
#box4 {  
    position: absolute;  
    bottom: 50px;  
    left: 50px;  
}
```



相对定位:

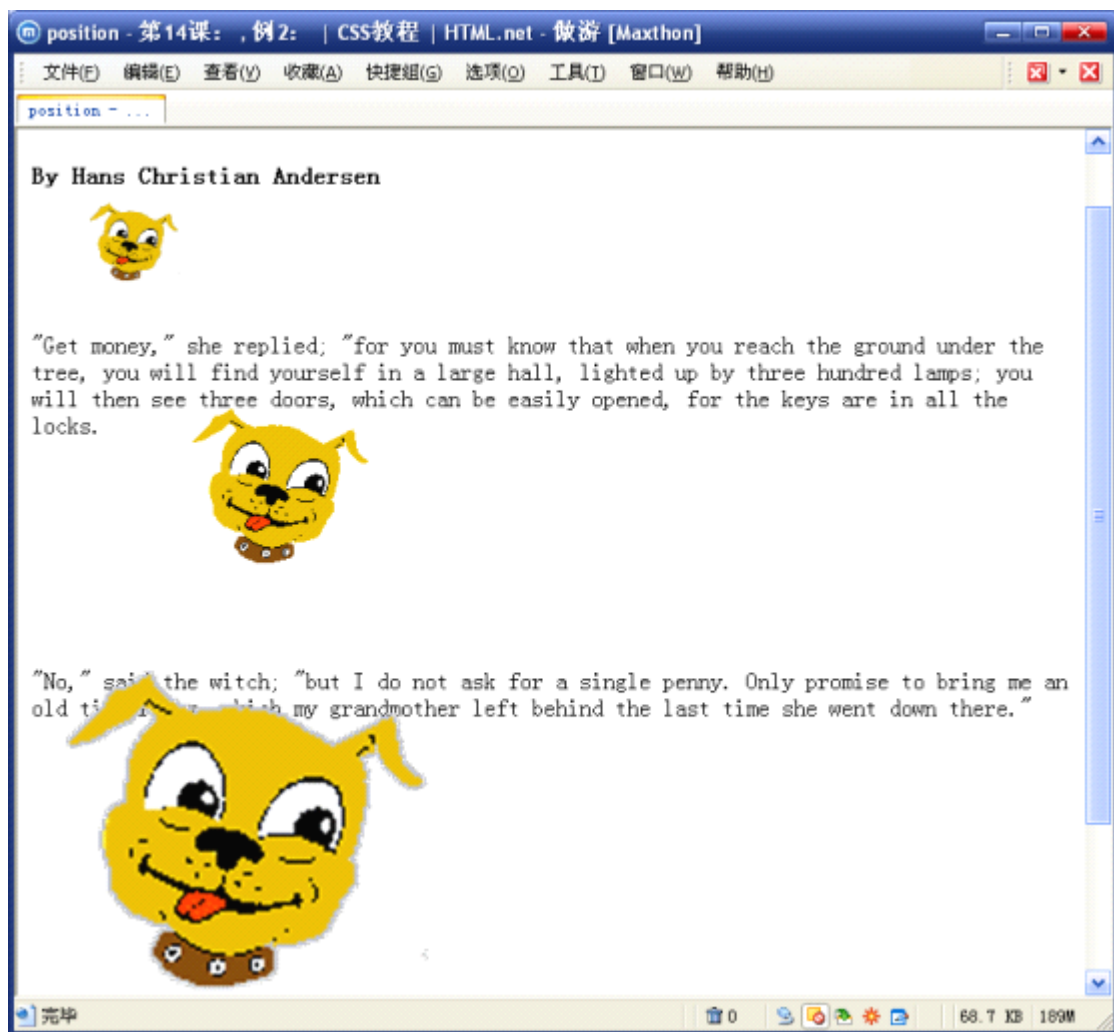
要对元素进行相对定位, 应将 `position` 属性的值设为 `relative`。绝对定位与相对定位的区别在于计算位置的方式。

采用相对定位的元素, 其位置是相对于它在文档中的原始位置计算而来的。这意味着, 相对定位是通过将元素从原来的位置向右、向左、向上或向下移动来定位的。采用相对定位的元素会获得相应的空间。

举个相对定位的例子, 我们可以相对于三张图片在页面上的原始位置来对它们进行相对定位。注意这些图片将在文档中各自的原始位置处留下空位。

```
#dog1 {
    LEFT: 35px; BOTTOM: 15px; POSITION:
relative
}
#dog2 {
    LEFT: 100px; BOTTOM: 50px; POSITION:
relative
}
#dog3 {
    LEFT: 10px; BOTTOM: 70px; POSITION:
relative
}
```

```
<BODY>
  <H1>The Tinder-Box</H1>
  <P><STRONG>By Hans Christian Andersen</STRONG></P>
  <DIV id=dog1>
    <IMG src="dog1.gif"></DIV>
  <P>"Get money," she replied; "for you must know that when you reach the ground
  under the tree, you will find yourself in a large hall, lighted up by three
  hundred lamps; you will then see three doors, which can be easily opened,
  for the keys are in all the locks.
  </P>
  <DIV id=dog2><IMG src="dog2.gif"></DIV>
  <P>"No," said the witch; "but I do not ask for a single penny. Only promise
  to bring me an old tinder-box, which my grandmother left behind the last time
  she went down there." </P>
  <DIV id=dog3><IMG src=" dog3.gif"></DIV>
  <P>Then he went into the third room, and there the dog was really hideous;
  his eyes were, truly, as big as towers, and they turned round and round in
  his head
  like wheels...</P>
</BODY>
```



第七章 HTML 中框架、层的运用

本章目标：掌握框架结构<frameset><frame><iframe>

掌握组织元素：span 和 div

本章重点：框架结构<frameset><frame><iframe>

本章难点：框架的搭建

一、 框架

使用框架，可以在一个浏览器窗口中显示不止一个 HTML 文档。这样的 HTML 文档被称为框架页面，它们是相互独立的。

使用框架的不利因素有：

- 网站开发者需要关心更多 HTML 文档的情况。
- 打印整个页面变得困难。

frameset 标签：

- <frameset>标签定义了如何将窗口拆分成框架。
- 每个 frameset 标签定义了一组行和列。
- 行/列的值指明了每个行/列在屏幕上所占的大小

frame 标签：

- <frame>标签定义了每个框架中放入什么文件。

下面这个例子中，有一个两列的分栏。第一个被设置成窗口宽度的 25%，第二个被设置成窗口宽度的 75%。页面“frame_a.htm”被放在第一个分栏中，“frame_b.htm”被放在第二个分栏中。

```
<frameset cols="25%, 75%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
</frameset>
```

基本注意点——有用的技巧：

假如一个框架有可见边框，用户可以拖动边框来改变它的大小。如果不想让用户改变大小，可以在<frame>标签中加入：noresize="noresize"。

给不支持框架的浏览器写上<noframes>标签。

更多示例：

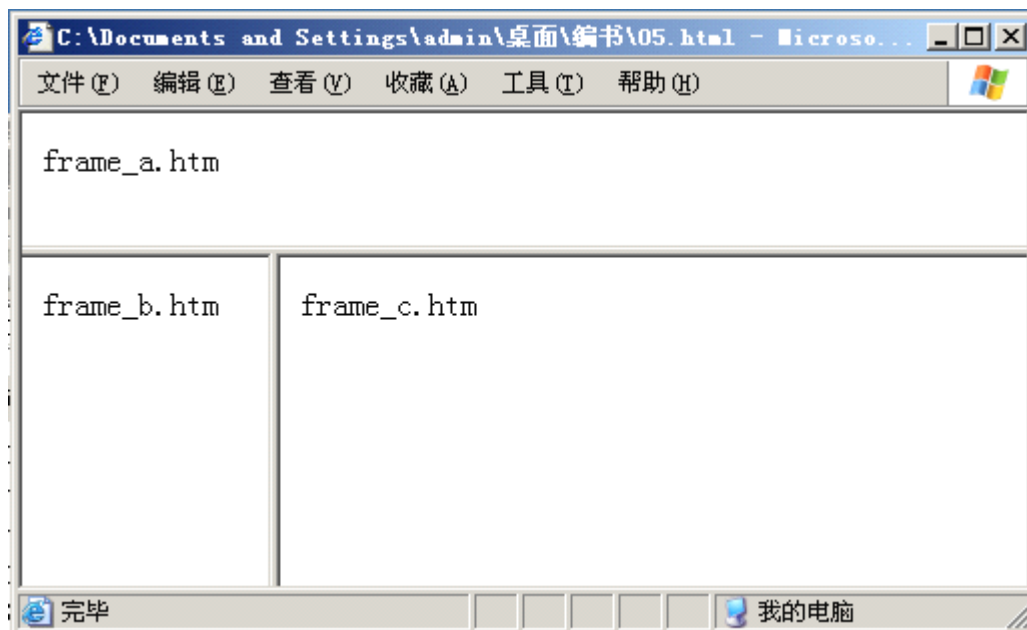
混合框架：

```
<html>
  <frameset rows="50%, 50%">
```

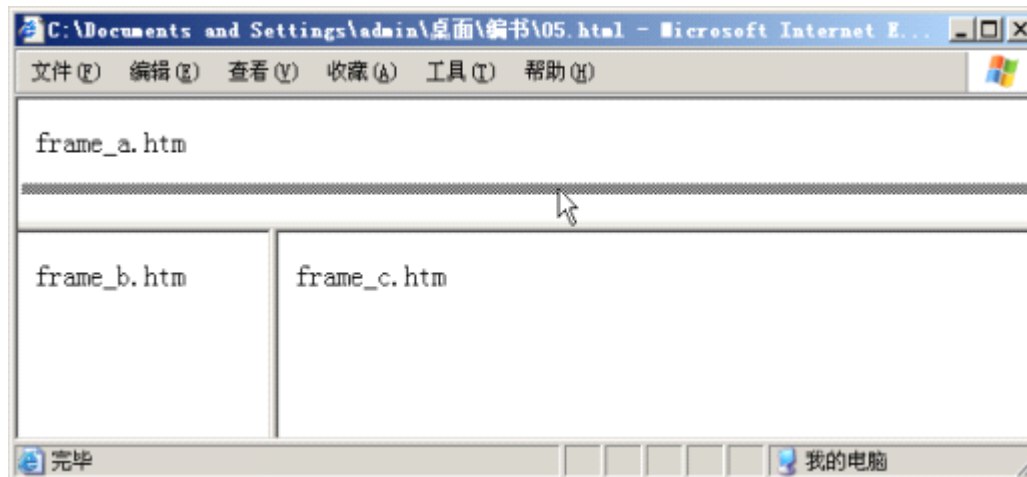
```

<frame src="frame_a.htm">
<frameset cols="25%,75%">
    <frame src="frame_b.htm">
    <frame src="frame_c.htm">
</frameset>
</frameset>
</html>

```



这个例子说明了怎样把三个页面以行列混合的方式放在框架中。



如上图所示：把鼠标移动到框架边界上，你会发现可以调整框架大小。

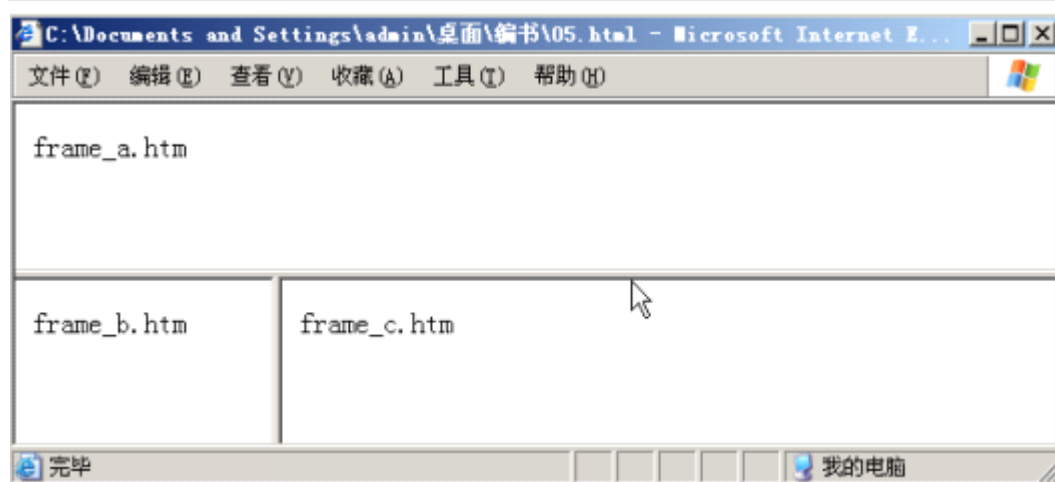
使用了 `noresize="noresize"` 的框架：

```

<html>
  <frameset rows="50%,50%">
    <frame noresize="noresize" src="frame_a.htm">
    <frameset cols="25%,75%">
      <frame noresize="noresize" src="frame_b.htm">

```

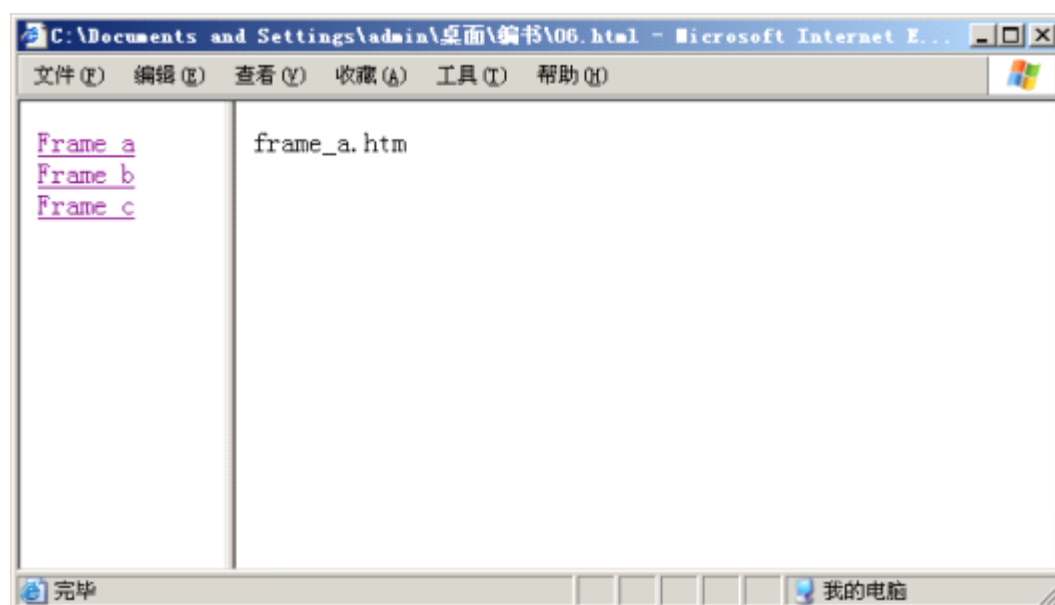
```
<frame noresize="noresize" src="frame_c.htm">
</frameset>
</frameset>
</html>
```



这个例子说明了 noresize 属性。这个框架是不可改变大小的，把鼠标移动到框架边界上，你会发现无法调整大小。

导航框架：

```
<html>
<frameset cols="120,*">
  <frame src="frame_link.htm">
  <frame src="frame_a.htm" name="showframe">
</frameset>
</html>
```



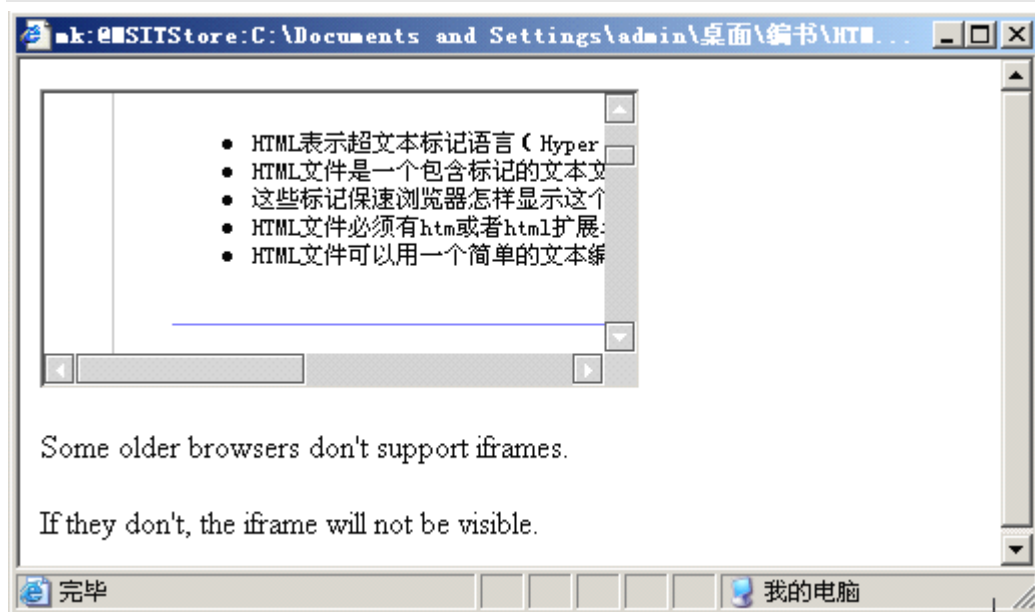
这个例子说明了如何创建一个导航框架。导航框架包含了一系列链接，它们的目标页面在第二个框架中。文件“frame_links.htm”包含了三个链接，链接的代码如下：

```
<a href = "frame_a.htm" target = "showframe">Frame a</a>
<a href = "frame_b.htm" target = "showframe">Frame b</a>
<a href = "frame_c.htm" target = "showframe">Frame c</a>
```

第二个框架将显示链接到的页面。

内联框架：

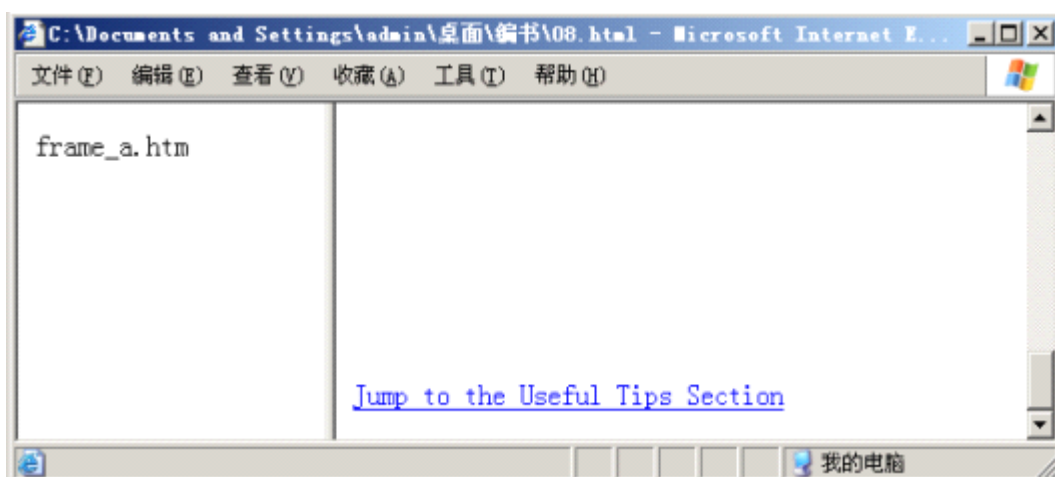
```
<html>
  <body>
    <iframe src="intro.htm"></iframe>
    <p>Some older browsers don't support iframes.</p>
    <p>If they don't, the iframe will not be visible.</p>
  </body>
</html>
```



这个例子说明了如何创建一个内联框架（包含在 HTML 页面里的框架）。

在框架内跳转到指定章节：

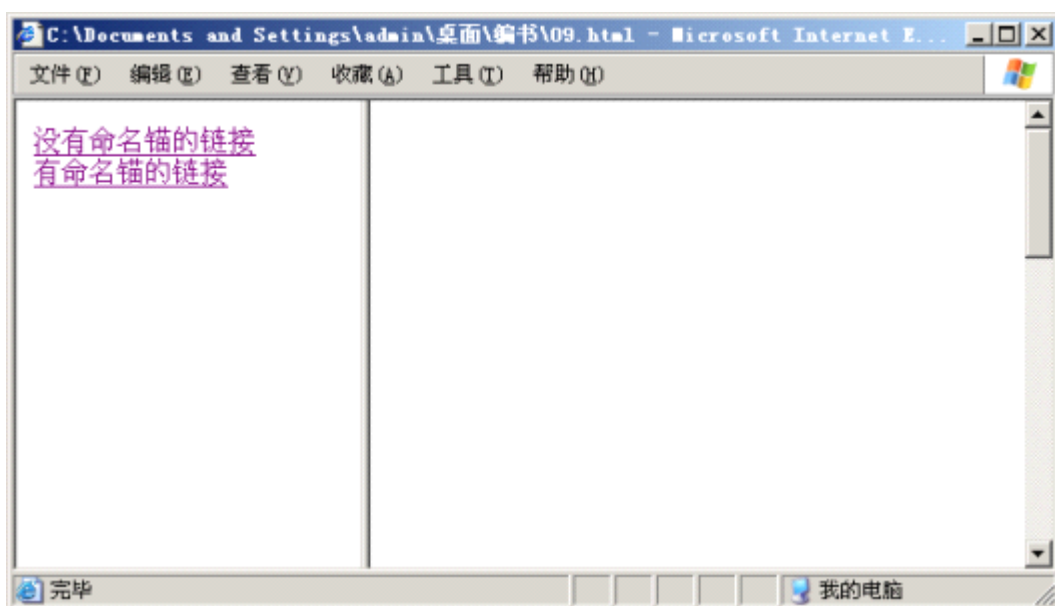
```
<html>
  <frameset cols="30%, 70%">
    <frame src="frame_a.htm">
    <frame src="frame_section.htm#C10">
  </frameset>
</html>
```



这个例子显示了两个框架页。其中一个的源是一个文件的指定章节，该章节在文件“frame_section.htm”中使用代码[](#)指定。

使用导航框架跳转到指定章节：

```
<html>
<frameset cols="200,*">
  <frame src="frame_linksection.htm">
  <frame src="frame_section.htm" name="showframe">
</frameset>
</html>
```



这个例子显示了两个框架页。左边的导航框架包含了一系列以第二个框架为目标的链接（“frame_linksection.htm”），第二个框架显示链接文件（“frame_section.htm”）。导航框架中的一个链接指向目标文件中的指定章节。文件“frame_link”中的 HTML 代码是像这样的：

```
<a href = "frame_section.htm" target = "showframe">没有命名锚的链接</a><br>
<a href = "frame_section.htm#C10" target = "showframe">有命名锚的链接</a>
```

二、 组织元素：span 和 div

span 和 div 元素用于组织和结构化文档，并经常联合 class 和 id 属性一起使用。

用 span 组织元素：

span 元素可以说是一种中性元素，因为它不对文档本身添加任何东西。但是与 CSS 结合使用的话，span 可以对文档中的部分文本增添视觉效果。

让我们用一句本杰明·弗兰克林（Benjamin Franklin）的名言来举个例子：

```
<p>早睡早起  
使人健康、富裕又聪颖。</p>
```

假设我们想用红色来强调弗兰克林先生所认为的“不要在睡眠中度过一天”的好处，我们可以用标签来标记上述每一点好处。然后，我们将这几个 span 设置为相同的 class。这样，我们稍后就可以在样式表里针对这个 class 定义特定的样式。

```
<p>早睡早起  
使人<span class="benefit">健康</span>、  
<span class="benefit">富裕</span>  
和<span class="benefit">聪颖</span>。</p>
```

相应的 CSS 代码如下：

```
span.benefit {  
    color:red;  
}
```

用 div 组织元素：

如前面例子所示，span 的使用局限在一个块元素内，而 div 可以被用来组织一个或多个块元素。

除去这点区别，div 和 span 在组织元素方面相差无几。让我们来看一个例子。我们将历届美国总统按其所属的政营分别组织为两个列表：

```
<div id="democrats">
  <ul>
    <li>富兰克林·D·罗斯福</li>
    <li>哈利·S·杜鲁门</li>
    <li>约翰·F·肯尼迪</li>
    <li>林登·B·约翰逊</li>
    <li>吉米·卡特</li>
    <li>比尔·克林顿</li>
  </ul>
</div>

<div id="republicans">
  <ul>
    <li>德怀特·D·艾森豪威尔</li>
    <li>理查德·尼克松</li>
    <li>杰拉尔德·福特</li>
    <li>罗纳德·里根</li>
    <li>乔治·布什</li>
    <li>乔治·W·布什</li>
  </ul>
</div>
```

在这里，我们可以采用跟上例同样的方法来处理样式表：

```
#democrats {
    background:blue;
}

#republicans {
    background:red;
}
```

第九章 XML 基本知识

本章目标：了解 XML 的应用范围

了解 XML 的文档结构

理解格式正规的 XML 文档的特点

熟悉有效的 XML 文档的编写规则

理解命名空间

本章重点：熟悉有效的 XML 文档的编写规则

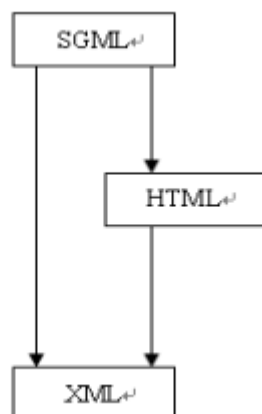
本章难点：理解命名空间

一、XML 的应用范围

人类一直在不断地尝试改进自己的发明，其中也包括人类最伟大的发明——文字的构成。第一个文本处理系统是用纸笔记录文字。现在，计算机文本处理器已经取代了手工处理，它不仅包含原始文档，还负责设置格式、出版和管理。在这些方便的功能整合到字处理之前，是由排字工人遵循书面标记说明来完成所有格式编排的。正是利益于这种实践，人们将“标记”这个词加入到 HTML 和 XML。顾名思义，标记是指加上记号。文本处理环境（如 XML）中使用了相同的标记过程。本意讲述标记语言的历史和创建 XML 文档的方法。

使用脚本语言或 DHTML 能够以各种方式显示信息。这就要求必须为相同的输出编写不同的代码以供不同的浏览器使用，因为这些语言不能跨浏览器兼容。

XML (eXtensible Markup Language, 可扩展标记语言) 克服了这些缺点。顾名思义，XML 是可扩展的，即开发人员可以定义自己的一组标签，并使其他的人或程序能够理解这些标签。HTML 是单标记语言，为特定应用设计，而 XML 则是一系列的标记语言。因此，XML 比 HTML 灵活得多。实际上，由于 XML 标签表示了数据的逻辑结构，不同的应用可以通过不同的方式来解释和使用这些标签。Web 上的数据大多是继承的，XML 继承了 SGML 和 HTML 的优点。也就是说，它不仅继承了 SGML 的特色，还结合了 HTML 的特色。它采用了 SGML 的主要框架，有时，人们也将 XML 称为 SGML 的子集。因此，HTML 是 SGML 的应用，而 XML 是 SGML 的子集。下图显示了标记语言的层次结构。



使用标签对文档进行标记以提供有关内容的信息，不仅能加快搜索速度，而且还能降低网络流量。XML 是由 SGML 修整并改造而来，它是一种元语言，用于描述其他语言。

我们可以使用 XML 为特定目的创建自己的标记语言（如化学标记语言）。XML 是基于文本的格式，允许开发人员描述结构化数据并在各种应用之间发送和交换这些数据，这样客户端就可以显示并自定义数据。

XML 还有助于在服务器之间传输结构化数据。有许多信息是分布在不同的和不匹配的数据库中。如有必要，XML 允许通过使用自定义格式来标识、交换和处理这些数据库可以理解的数据。

XML 和 HTML 有许多相同点和不同点。XML 描述数据，如城市名称、温度和气压；HTML 定义描述数据显示方式的标签，如使用项目符号列表或表格。但 XML 允许开发人员定义任意数量的标签集，使用开发人员有很大的灵活性来决定要使用哪些数据，并确定数据的适用标准或自定义标签。

XML 应用范围：

对于 Internet 和大型企业 Intranet 环境，XML 都是十分有价值的。这是因为它通过灵活、开放及基于标准的格式、访问遗留数据库及将数据发送至 Web 客户端的新方式，来提供了协同工作能力。不仅可以更快地构建应用，而且更易于维护，还可以通过不同的样式表提供多个结构化数据的视图，这将在后面的章节中介绍。

下面使用两个救命来说明使用 XML 会给个人、公司和组织带来什么好处。

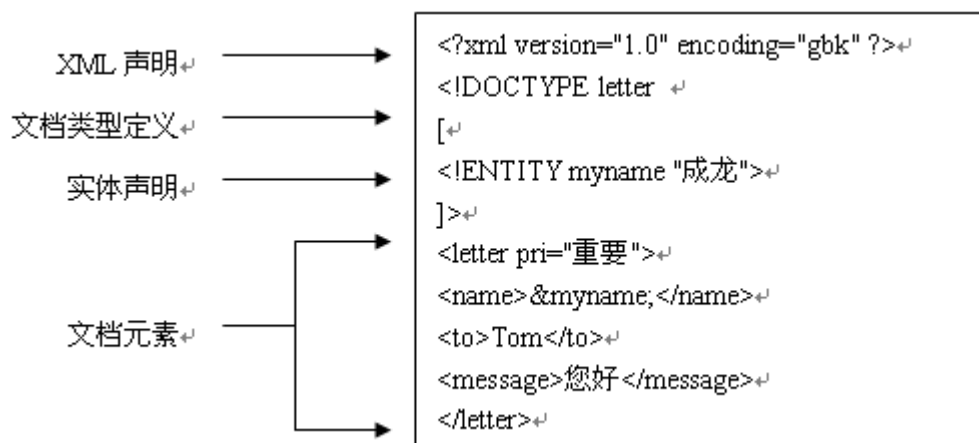
SABRE：SABRE 集团是主要的国际旅游服务商之一，通过旅行社和 Web 提供电子旅游预约。他们使用 Java 应用程序将旅游信息转换为 XML，这样全球的移动电话用户都可以通过移动电话查找、预订和购买机票和门票。XML 自动转化为 Wireless Markup Language（无线标记语言，WML），这是在移动电话上构建应用的标准语言。对于此应用来说，XML 的优势在于其可扩展性、开发速度、能够使用 XML 构建标准资料库以及将它转换到特定的环境中，本例中为移动电话。

化学标记语言：化合物和化学分子是原子的复杂组合。大概有两千万个已知分子，直到最近才有机器可读的表示分子的标准方式。诺丁汉大学的 Peter Murray Rust 教授和伦敦大学帝国理工学院的 Henry Rezza 博士这两们英国科学家开发出了用 XML 描述分子的标准方式。Chemical Markup Language（化学标记语言，CML）预期能够为化学行业节省大量人力物力，还有助于化学家之间以及其他相关学科（如生物和医药）之间的交流。在这种应用中使用 XML 的关键优势之一在于 XML 提供大量的工具，有助于迅速高效地创建 CML 应用。

二、 XML 的文档结构

XML 文档是由一组使用唯一名称标识的实体组成。所有文档都以根或文档实体开始，而且所有实体都是可选的。实体可以被视为更复杂功能的别名。单个实体名称可以代替许多文本。在别名方案中，每当需要引用某个文本时，只需要使用别名，处理器会展开别名的内容。

XML 文档也有一种逻辑结构。逻辑上，文档的组成包括声明、元素、注释、字符引用和处理指令（在文档中使用显式标记表示），如下图所示。



XML 文档始终以一个声明开始，这个声明指定该文档遵循 XML1.0 规范。

XML 声明：

XML 声明的语法如下所示。

```
<?xml version="1.0" ?>
```

XML 声明是可选的，XML1.0 版本是默认值。W3C 规范建议使用 XML 声明，这样可以为文档匹配合适的解析器。发布更新版本时，需要提供相应的 XML 版本号。

XML 声明是处理指令，告知处理代理该文档已经标记为 XML 文档，它还告诉解析器和其他应用程序应如何处理文件中的数据。包括 XML 声明在内的所有处理指令都以 “<?” 开始，以 “>” 结束。“<?” 后面是处理指令的名称，即 “xml”。XML 处理指令要求指定一个 version 属性，并允许指定可选的 standalone 和 encoding 属性。XML 声明至少应有保留名称 xml 以及一个版本号，只有版本号是必需的。encoding 详细信息和 standalone 声明可以跟在版本号后面，如下所示。

```
<?xml version="1.0" standalone="no" encoding="UTF-8" ?>
```

如果包括了可选属性，则必须先指定版本。

standalone 属性可以设置为 yes 或 no。yes 指定不使用外部声明，而 no 则表示将引用外部声明。

所有 XML 解析器都必须支持与 ASCII 相应的 8 位或 16 位 Unicode 编码。

“encoding="UTF-8"”指定作者使用的字符编码。UTF8 与 8 位 ASCII 字符相对应。

“GB2312”或“GBK”与中文字符集相对应。

根元素：

根元素只能有一个，用于描述文档的功能。每个 XML 文档都有一个根元素。<HTML> 是 HTML 的根元素。在 XML 中，可以自定义根元素。例如，使用<BOOK>作者为根元素，如下所示：

```
<?xml version="1.0" standalone="no" encoding="UTF-8" ?>
<BOOK>
</BOOK>
```

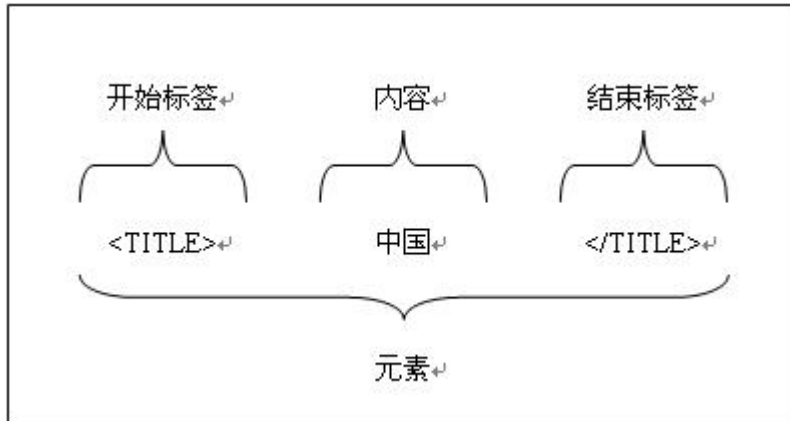
XML 代码：

根据应用需要创建自定义元素和属性。元素涉及标签及其内容。例如：

```
<B>BOLD TEXT</B>
```

标签包括尖括号以及尖括号中的文本。例如，<P>、<I>和</I>是在 HTML 中使用的一些标签。标签告诉用户代理（浏览器）处理结束标签之间的内容。

元素是 XML 内容的基本单元。开始标签，如<element_name>，标志着元素的开始，而结束标签，如</element_name>，标志着元素的结束。图 1.5 举例说明了元素的组成部分。



标签组成了 XML 标记的主要部分。XML 标签与 HTML 标签非常相似。

数据与标记：

XML 文档由数据以及描述该数据的标记组成。数据通常是字符数据，但也可以是二进制数据。标记包括标签、注释、处理指令、DTD 和引用等。

以下是一个字符数据和标记的简单示例。

```
<NAME>成龙</NAME>
```

在本例中，<NAME>和</NAME>是标记，“成龙”是字符数据。

注释：

有时，需要在 XML 文档中包含某些标签，而 XML 解析器（XML 解析器帮助计算机解释 XML 文件）应该忽视这些标签，这种类型的文本称为注释文本。在 HTML 中，使用<!--和-->语法指定注释。在 XML 中也以相同的方式指定注释。注释的语法如下所示。

```
<!-- 在此处写注释 -->
```

使用注释时要遵循以下规则：

注释文本中不应包含“-”或“--”，因为可能会使 XML 解析器产生混淆。

注释绝对不能放在标签中。因此，以下代码是错误的。

```
<NAME <!-- 姓名 -->>汤姆·克鲁斯</NAME>
```

注释不能放在实体声明中，也不能放在 XML 声明之前。XML 声明必须始终是所有 XML 文档的第一行。

注释可用于注释标签集。因此，在以下的代码中，除汤姆·克鲁斯以外的所有名字都会被忽略。

```

.....
<!-- 不显示
<NAME>凯特·温斯莱特</NAME>
<NAME>妮可基曼</NAME>
<NAME>阿诺德</NAME>
-->
<NAME>汤姆·克鲁斯</NAME>
.....

```

注释不能嵌套

处理指令：

处理指令是为使用该 XML 文档的应用提供的一则信息。该指令直接传递到使用该解析器的应用，应用可以将信息传递到另一个应用，或自行对信息进行解释。XML 声明也是一个处理指令。

处理指令的格式相同，如下所示。

```

<?xml :stylesheet type="text/xml" ?>
      ^           ^
      应用的名称  指令信息

```

所有处理指令都必须以<? 开始，以 ?> 结束。

标签间的字符数据的分类：

开始标签和结束标签之间的文本被定义为字符数据。字符数据可以是“<”以外的任何合法（Unicode）字符。“<”字符预留作标签的开始字符。

Unicode 定义一个完全国际化的字符集，可以表示人类语言中的所有字符。它统一了许多字符集，如拉丁字符、希腊字母、阿拉伯文等。

字符数据可以分为以下两类：PCDATA，CDATA。

具体叙述如下：

PCDATA: PCDATA 表示已解析的字符数据。字符数据可被视为 XML 元素的开始标签和结束标签之间的文本。PCDATA 是将来要通过解析器进行解析的文本。文本中包含的标签将被视为标记，实体将会扩展。

CDATA: CDATA 指字符数据。CDATA 是不通过解析器进行解析的文本，文本中包含的标签将不被视为标记，实体不会扩展。在 CDATA 块中，XML 解析器会忽略所有标签和实体引用。为了便于包含大量的特殊字符，提供了 CDATA 块。下面来看以下一段代码。

```

.....
<SAMPLE>
<![CDATA[<DOCUMENT>
<NAME>成龙</NAME>
<EMAIL>jackie@usa.com</EMAIL>
</DOCUMENT>]]>
</SAMPLE>
.....

```

在以上代码中，不允许在 CDATA 块之内使用字符串 “]]>”，因为它表示 CDATA 块的结束。

实体：

实体是 XML 的存储单元。实体可以包含常用的短语、键盘字符、文件、数据库刻录或任何包含数据的项。

在文档中使用实体可以避免在文档中重复键入长段的文本。可以将一个实体名和文本关联，然后每当需要在文档中放入该文本时，就使用此实体名。处理文档时，该实体名将被替换为指定的文本。

在 XML 中，有些字符（如<、>或&）可以包括在文本中，但不能以字面格式存在，否则解析器会生成错误。

XML 规范定义了一个预定义字符实体集，可以用于取代字符的字面格式。一共有 5 个这种表示字符的预定义实体，这些字符可能会与 XML 标记代码混淆，见下表。

<	<
>	>
&	&
"	"
'	'

使用实体引用将实体插入 XML 文档。解析器遇到实体引用时，会将引用替换为实体的内容。例如，可以在标记中使用的实体引用，如：

```
<ORDER VALUE="He said, "Don't jump out of the window!"">
```

应写成：

```
<ORDER VALUE="He said, &quot;Don&apos;t jump out of the window!&quot;">
```

基本上，实体是用于定义常见文本的快捷方式的变量。实体分为两类：一般实体、参数实体。

一般实体：

可以在 XML 文档中的任何位置出现的实体称为一般实体，实体可以声明为内部实体或外部实体。内部实体仅存在于声明它们的文档中，外部实体则指文档外的存储单元。

一般实体命名如下：

```
<!ENTITY address "要以实体表示的文本">
```

替换文本后，该示例会变成：

```
<!ENTITY address "我的地址：美国洛杉矶 第 10 大街 12 号 12 套房">
```

上面指定的实体是一个内部实体。

外部实体使用一个标识符指向文档外的存储单元。外部实体标识符分为两种类型，SYSTEM（系统）和 PUBLIC（公共）。前者用于引用本地计算机（或网络），后者用于引用公共计算机（或网络）。外部实体示例如下：

```
<!ENTITY greeting SYSTEM "test.txt">
```

在本例中，XML 处理器会将实体引用替换为 URI “test.txt” 指定的文档内容。“test.txt” 文件包含引用实体时要放入的文本。SYSTEM 关键字指引解析器在指定 URI 查找文件。使用实体引用将实体插入 XML 文档。实体引用指解锁实体的密钥，已经在实体声明中声明。

语法如下：

```
&ENTITY_NAME
```

如 &address;。

假设将一个地址作为实体保存在共享文件中。每当在 XML 中写入此地址时，将执行与以下代码相似的操作。

```
<LETTER>
  &address;
  <TO>成龙</TO>
  <BODY>嗨!您好!</BODY>
  <FROM>克里斯</FROM>
</LETTER>
```

地址将扩展为：

我的地址：美国洛杉矶第 10 大街 12 号 12 套房

管理实体引用的规则包括：

引用实体前，必须先在 XML 文档中声明该实体。

实体引用不应含有任何空格。例如，“& address;”或“&address;”将导致错误。

实体引用的文本必须是格式良好的 XML 文档。

实体引用可以替代常规的字符数据，还可以在标签属性中使用实体引用。例如：

```
<CLIENT="&APTECH;" PRODUCT="&PRODUCT_ID;" QUANTITY="15">
```

参数实体：

当实体和实体引用都只需在 DTD 中出现时，则使用参数实体。参数实体，无论是内部还是外部，都只在 DTD 中使用。它们不能在文档内容中使用，因为处理器无法识别。

格式良好的参数实体看上去与一般实体相似，区别仅在于前者使用“%”说明符。假设以下示例。

```
<!ENTITY% ADDRESS "实体要表示的文本">
```

参数实体引用与一般实体引用相似。在本例中，使用“%”而不是“&”。

```
%PARAMETER_ENTITY_NAME;
```

稍后会在 DTD 一节中给出实例。

DOCTYPE 声明：

在 XML 文档中，<!DOCTYPE [...]> 声明跟在 XML 声明的后面。实体必须在文档 DOCTYPE 声明中声明。

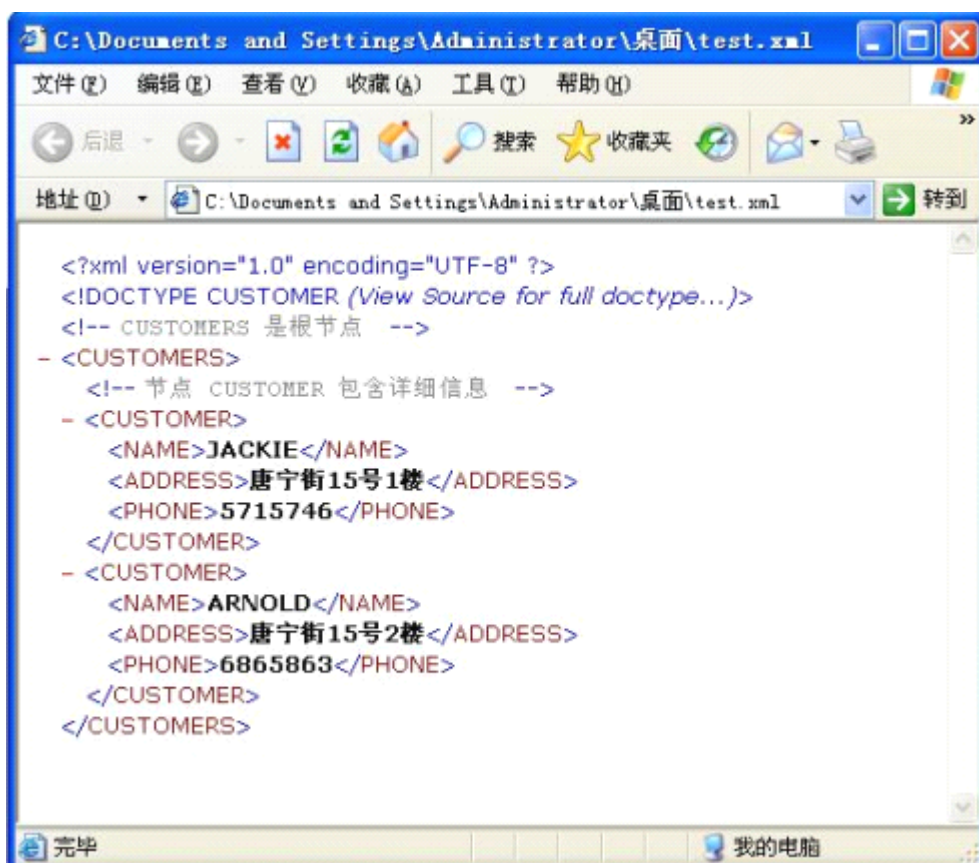
语法如下所示：

```
<?xml version="1.0" ?>
<!DOCTYPE myDoc [
...在此处声明实体...
<myDoc>
...文档正文...
</myDoc>
```

对 XML DOCTYPE 声明进行编码有助于创建文档（如例 2 所示）。使用实体时要考虑更改地址的方便性。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CUSTOMER [
<!ENTITY FIRSTFLOOR "唐宁街15号1楼">
<!ENTITY SECONDFLOOR "唐宁街15号2楼">
]>
<!--CUSTOMERS 是根节点-->
<CUSTOMERS>
  <!--节点 CUSTOMER 包含详细信息-->
  <CUSTOMER>
    <NAME>JACKIE</NAME>
    <ADDRESS>&FIRSTFLOOR;</ADDRESS>
    <PHONE>5715746</PHONE>
  </CUSTOMER>
  <CUSTOMER>
    <NAME>ARNOLD</NAME>
    <ADDRESS>&SECONDFLOOR;</ADDRESS>
    <PHONE>6865863</PHONE>
  </CUSTOMER>
</CUSTOMERS>
```

例 2 的输出结果如图 1.6 所示：



在例 2 中，使用了 FIRSTFLOOR 和 SECONDFLOOR 这两个实体引用。实体引用替换常规字符数据，使用以下语句引用该常规字符数据。

```
<ADDRESS>&FIRSTFLOOR;</ADDRESS>↵
```

遇到此行时，早先在 XML 代码的开头声明的整个字符数据都会被替换为字符。

三、 格式良好和有效的 XML 文档

如果一个 XML 文档满足了最低的要求集（在定义 XML 语法的 XML 1.0 规范中定义），则该文档被视为格式良好。这些要求确保以正确的方式使用正确的词语（在 XML 规范中定义）。如果文档不满足任何一个良好格式的要求，则将发生致命错误。

有效的 XML 文档是格式良好的 XML 文档，符合 Document Type Definition（文档类型定义，DTD）的规则。DTD 定义了文档中的标记必须遵循的规则，还包含指定文档总体结构的定义以及可以接受的数据内容值的类型。有效的 XML 文档还符合 SGML 文档的标准。

至少需要一个元素：

所有格式良好的 XML 文档都必须至少有一个元素。

XML 标签区分大小写：

必须注意确保在标签集使用正确的大小写，也就是说，尽管它们在 HTML 中表示相同的意思，但<HELLO>和<hello>标签是不一亲的。这是因为 XML 区分大小写。

应正确使用结束标签：

除了拼写和大小写与开始标签相同，结束标签应该在前面有一个斜杠“/”。因此，在大多数情况下，以<HELLO>作为开始标签，就应该以</HELLO>作为结束标签。在某些时候，结束标签可以省略。尤其是如果需要使用不带内容的标签，则使用带有尾随斜杠的单个开始标签，如<HR/>。

正确嵌套标签：

请注意，XML 元素可以包含其他元素，但元素的嵌套必须正确。以下代码中的元素嵌套是错误的。

```
.....  
<CONTACT>  
<NAME>成龙</NAME>  
<EMAIL>jackie@china.com>  
</CONTACT>  
</NAME>  
</EMAIL>  
.....
```

它应该是：

```
<CONTACT>  
<NAME>成龙</NAME>  
<EMAIL>jackie@china.com</EMAIL>  
</CONTACT>
```

应使用合法标签：

标签必须以一个字母、下划线（_）或冒号（:）开始，然后是字母、数字、句号（.）、冒号、下划线或连字符（-）的组合，但不能有空格。标签不应以“xml”开头，因为它是保留字。最好不要将冒号作为标签名称的第一个字符（即使这是合法的），因为它会

引起混淆。

标记名称的长度：

尽管 XML 1.0 标准中规定可以使用任何长度的名称，但实际的 XML 处理器可能会限制标记名称的长度。XML 标签名称的长度取决于处理器。

应定义有效的属性：

标签可以指定许多支持属性。一个标签中的属性不能重复。指定一个名称和值对，以等号 (=) 分隔，其中，值使用引号分隔。

```
<CAR MODEL="MARUTI 800" COLOR="WHITE">
```

和 HTML 不同，XML 规定值必须使用引号分隔。在本例中，MODEL 和 COLOR 是 CAR 标签的属性，“MARUTI 800”是 MODEL 属性的值，而“WHITE”是 COLOR 属性的值。属性命名和标签命名遵循相同的规定。不过，必要时，值可以包含空格、标点和实体引用。

所有值都被视为字符串。因此，如果是标签：

```
<WATER_TANK RADIUS="5" DEPTH="20">
```

“5”和“20”会转换为 XML 环境之外的数值。

应验证文档：

文档应遵循 XML 规则，否则浏览器或任何其他 XML 阅读器都无法读取此文档。

四、 XML 文档的编写规则

XML 从 Standard Generalized Markup Language（标准通用化标记语言，SGML）衍生而来，和 SGML 一样，它支持使用 DTD。文档类型声明和文档类型定义不相同。文档类型定义缩写为 DTD。DTD 指定了 XML 文档的语法结构，从而使 XML 解析器能够理解和解释文档的内容。

更明确地说，DTD 定义了元素在文档的树形结构中相关联的方式，并指定了和某些元素一起使用的属性。因此，它也包含可以在文档中包含的元素类型。有效的 XML 文档就是符合其 DTD 的文档。DTD 以简单文本文件的形式出现，可以存储在独立的文件中，也可以嵌入 XML 文件。引用 DTD 的 XML 文档将包含<!DOCTYPE>声明，此声明包含 DTD 声明，或指定外部 DTD 的位置。

为什么使用 DTD：

XML 提供了以独立方式来共享数据的应用。相互独立的人群可以达成协议，使用通用的 DTD 来交换数据。应用可以使用标准 DTD 来验证接收的数据是否有效。DTD 可以用于验证自己的数据。在数据交换领域，出现了无数旨在定义标准 DTD 论坛。DTD 的目的在于定义 XML 文档的合法构建块。它使用一系列合法元素来定义文档结构。

DTD 结构:

DTD 包括许多组件, 如 DOCTYPE 声明、元素声明和属性声明。如前所述, <!DOCTYPE> 声明包含了有关 DTD 位置的信息。一般而言, 元素在字典中定义为“复合实体的基本、必要或不可少的要素”, 但在不同情况下, 它代表不同的对象。在数学中, 它是一个集的成员; 在化学中, 它是由具有相同原子数的原子组成的物质; 而在数据库语言(如 SQL) 中, 它表示表中的一个字段。在 XML 中, 元素是文档的一个逻辑组件。在 XML 文档中声明的每个元素都必须在 DTD 中具有对应的元素声明, 以便在验证时识别身份。同样地, 属性在字典中的定义为“用于表示特性、个性或职责的关联对象”。在 XML 中, 它表示元素的特性。一个元素可以包含表示该元素的特性的属性。必须按照在 XML 文档中声明元素的方法, 在 DTD 中声明属性。总而言之, DTD 的一般结构如下所示。

```
<!DOCTYPE dtd-name [
<!ELEMENT element-name (element-content type)>
<!ATTLIST element-name attribute-name attribute-type
    default-value>
]>
```

可以在 XML 文件中声明 DTD, 也可以将它存储在独立的文件。如果存储在独立的文件, 使用 .dtd 扩展名进行保存。

声明元素:

在 DTD 中, 使用元素声明来声明 XML 元素。元素声明具有以下语法。

```
<!ELEMENT element-name (element-content type)>
```

例如:

```
<!ELEMENT SHOWROOM (TV|LAPTOP)+>
```

空元素:

EMPTY 元素内容类型指定该元素没有子元素或字符数据。将关键字 EMPTY 放在指定位置, 可以声明空元素。语法如下:

```
<!ELEMENT element-name EMPTY>
```

例如:

```
<!ELEMENT img EMPTY>
```

空元素可以具有属性。

带有数据的元素:

带有数据的元素是使用它们的数据类型声明的。此数据类型在括号中指定。语法如下:

```
<!ELEMENT element-name (#CDATA)>
or
<!ELEMENT element-name (#PCDATA)>
or
<!ELEMENT element-name ANY>
```

例如:

```
<!ELEMENT note (#PCDATA)>
```

#CDATA 指元素包含不会通过解析器进行解析的字符数据。#PCDATA 指元素包含要通过解析器进行解析的数据。ANY 指该元素可以包含零个或零个以上任何声明类型的子元素以及字符数据。因此，它是包含所有已声明元素的混合内容的简略表达方式。

带有子元素(序列)的元素：

要定义带有一个或多个子元素的元素，则将子元素的名称放入括号内。语法如下：

```
<!ELEMENT element-name (child-element-name)>
```

或

```
<!ELEMENT element-name (child-element-name,child-element-name,.....)>
```

例如：

```
<!ELEMENT note (to,from,heading,body)>
```

如果在一个序列中声明子元素，并用逗号将它们分开，则这些子元素必须以其在文档中的顺序来显示。子元素可以具有自己的子元素。

声明相同的元素只出现一次：

以下示例声明了子元素 message，它在 note 元素中仅出现一次。语法如下：

```
<!ELEMENT element-name (child-name)>
```

例如：

```
<!ELEMENT note (message)>
```

声明相同的元素至少要出现一次：

以下示例中的“+”符号声明子元素 message 必须至少在 note 元素中出现一次。语法如下：

```
<!ELEMENT element-name (child-name+)>
```

例如：

```
<!ELEMENT note (message+)>
```

声明相同的元素出现零次或多次：

以下示例中的。号声明子元素 message 可以在 note 元素中出现零次或多次。语法如下：

```
<!ELEMENT element-name (child-name*)>
```

例如：

```
<!ELEMENT note (message*)>
```

声明相同的元素出现零次或一次：

以下示例中的“?”号声明子元素 `message` 可以在 `note` 元素中出现零次或一次。语法如下：

```
<!ELEMENT element-name (child-name?)>
```

例如：

```
<!ELEMENT note (message?)>
```

声明混合内容：

以下示例声明 `note` 元素必须至少包含一个 `to` 子元素，有且只有一个 `from` 子元素和一个 `header` 子元素，具有零个或一个 `message` 子元素以及其他已解析的字符数据。

例如：

```
<!ELEMENT note (to+,from,header,message*)>
```

组可以是序列或子元素和/或子组的选择。例如：

序列

```
<!-- 元素 A 由单个元素 B 组成。-->
```

```
<!ELEMENT A (B)>
```

```
<!-- 元素 A 由元素 B 加上元素 C 组成。-->
```

```
<!ELEMENT A (B,C)>
```

```
<!-- 元素 A 由包括选择子组的序列组成。-->
```

```
<!ELEMENT A (B,(C|D),E)>
```

选择

```
<!-- 元素 A 由元素 B 或元素 C 组成。-->
```

```
<!ELEMENT A (B|C)>
```

```
<!-- 元素 A 由包括序列子组的选择组成。-->
```

```
<!ELEMENT A (B|C|(D,E))>
```

例 3 演示如何在 DTD 中使用元素。它使用外部 DTD，本章稍后将详细说明外部 DTD。

例 3：

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE book SYSTEM "Example3.dtd">
<book>
<details>
  <name>xml 使用详解</name>
  <author>成龙来自&country;</name>
```

```

<publication>Mac graw &rights;</publication>
<ptice>&pricenotation;50</price>
</details>
<details>
<name>xml 揭密</name>
<author>Raghu 来自&count;</name>
<publication>Mac graw &rights;</publication>
<ptice>&pricenotation;45</price>
</details>
</book>

```

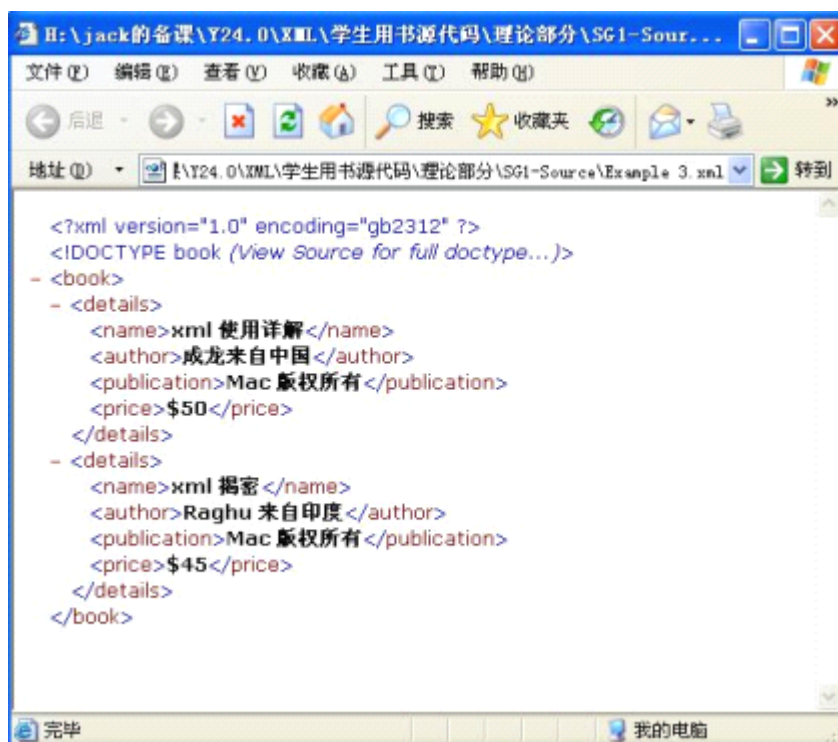
此文件保存为 Example 3.xml。

```

<?xml version = "1.0" encoding="gb2312"?>
<!ELEMENT book (details+)>
<!ELEMENT details ( name, author, publication, price)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publication (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ENTITY country "中国">
<!ENTITY count "印度">
<!ENTITY rights "版权所有">
<!ENTITY pricenotation "$">

```

此文件保存为 Example 3.dtd。例 3 的输出结果下图所示。



在例 3 中使用了外部 DTD，DTD 通过以下语法声明。

```
<!DOCTYPE book SYSTEM "Example3.dtd">
```

在此 DTD 中，声明哪些元素应用于 Example 3.xml。输出结果如上图所示。

```
<!ELEMENT book (details+)>
<!ELEMENT details ( name, author, publication, price)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publication (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

例 3 中的代码演示了已声明的元素。Book 元素有子元素 details，details 元素有子元素 name、author、publication 和 price。元素中的 #PCDATA 表示包含要通过解析器进行解析的数据。

```
<!ENTITY country "中国">
<!ENTITY count "印度">
<!ENTITY rights "版权所有">
<!ENTITY pricenotation "$">
```

例 3 中的代码演示了在 XML 代码中声明的各种实体。每当遇到该实体名称时，将会替换分号中的字符。

属性声明：

在以下示例中，将元素 square 定义为空元素，width 属性类型为 CDATA，width 属性的默认值为 0。而后，把 width 属性值赋为 100。

DTD 示例：

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

XML 示例：

```
<square width="100"/>
```

Default 属性值：

为属性指定一个默认值，确保即使 XML 文档的作者不提供值，该属性也将获取一个值。语法如下：

```
<!ATTLIST element-name attribute-name CDATA "default-value">
```

DTD 示例：

```
<!ATTLIST payment type CDATA "check">
```

XML 示例：

```
<payment type="check">
```

Implied 属性值:

如果开发人员不希望强迫作者提供属性，而且也没有默认值选项，则他们需要使用 implied 属性。语法如下：

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

DTD 示例：

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML 示例：

```
<contact fax="222-899877"/>
```

Required 属性值:

如果没有默认值，但仍然希望在文档中出现此属性，则使用 required 属性。语法如下：

```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

DTD 示例：

```
<!ATTLIST person number CDATA #REQUIRED>
```

XML 示例：

```
<person number="6787"/>
```

Fixed 属性值:

如果希望属性具有固定值，使作者不能更改，则使用 fixed 属性值。如果作者提供其他值，XML 解析器将返回一个错误。语法如下：

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

DTD 示例：

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

XML 示例：

```
<sender company="Microsoft"/>
```

Enumerated 属性类型:

希望属性值成为一组固定合法值的一员时，使用 enumerated 属性值。语法如下：

```
<!ATTLIST element-name attribute-name (eval|eval|..) default-value>
```

DTD 示例：

```
<!ATTLIST payment type (支票|现金) "现金">
```

XML 示例：


```
<payment type="支票">
```

或

```
<payment type="现金">
```

ID 和 IDREF 属性类型:

ID 是标识符类型, 它应该是唯一的。该属性值用于搜索某个元素的特定实例。每个元素都可以具有 ID 类型的一个属性。语法如下:

```
<!-- Topicid 属性提供 Topic 元素的 ID-->
<!ATTLIST Topic Topicid ID #REQUIRED>
.....
<Topic Topicid="Topic4">
  此 Topic 是 XML
</Topic>
```

IDREF 也是标识符类型, 它应只指向一个元素。IDREF 属性可用于引用其他元素中的一个元素, 如以下代码所示。

```
.....
<!-- Prev 和 Next 属性指向另一元素的 ID-->
<!ATTLIST Topic Topicid ID #REQUIRED>
<!ATTLIST Topic Prev IDREF #IMPLIED>
<!ATTLIST Topic Next IDREF #IMPLIED>
.....
<Topic Topicid="Topic4" Prev="Topic3" Next="Topic8">
  <!-- Topics 5-7 丢失-->
  此 Topic 是 XML
</Topic>
.....
```

IDREFS 属性类型:

此属性将多个元素 ID 作为它的值, 各个 IDREF 值之间用空格分开。它用于指向 XML 文档中的相关元素列表, 如以下代码所示。

```
.....
<!ATTLIST Topic Topicid ID #REQUIRED>
<!ATTLIST Topic Prev IDREF #IMPLIED>
<!ATTLIST Topic Next IDREF #IMPLIED>
<!ATTLIST Topic Xrefs IDREFS #IMPLIED>
.....
<Topic Topicid="Topic4" Prev="Topic3" Next="Topic8"
  Xrefs="Topic1 Topic2">
  <!-- Topics 5-7 丢失-->
```

```

    此 Topic 是 XML
</Topic>
.....

```

ENTITY 和 ENTITIES:

这些属性指向以未解析实体(解析器无法处理的实体)形式存在的外部数据。语法如下所示。

```

<!--属性 a 指向单个未解析实体-->
<!ATTLIST A a ENTITY #IMPLIED>

<!--属性 b 指向多个未解析实体-->
<!ATTLIST A b ENTITIES #IMPLIED>

```

NMTOKEN 和 NMTOKENS:

它们用于指定任何有效的一个或多个 XML 名称。将其他组件与元素(如 Java 类或安全算法)关联时,可以使用这些属性。这些属性以单个/多个记号作为值,如以下代码所示。

```

<!ATTLIST Data Authorised_Users NMTOKENS #IMPLIED>
<Data SECURITY="ON" Authorised_Users="Tom">
.....
</Data>
.....

```

DTD 示例:

DTD 定义生成自定义标签的语言的规则。在一个 DTD 中定义每个元素的详细信息、它们的顺序以及每个元素的属性。DTD 分为两种类型。

内部 DTD:

在 XML 文档的 XML 声明后直接编写内部 DTD。应该在 DOCTYPE 定义中编写内部 DTD,如示例所示,这称为包装。语法如下:

```
<!DOCTYPE root-element [element-declarations]>
```

例 4 演示如何使用内部 DTD。

例 4:

```

<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE movies
[
<!ELEMENT movies (movie+)>
<!ELEMENT movie (title,actor+,rating)>
<!ELEMENT title (#PCDATA)>

```

```

<!ELEMENT actor (#PCDATA)>
<!ELEMENT rating (#PCDATA)>
<!ATTLIST movie type CDATA #IMPLIED>
]
>
<movies>
  <movie type="冒险片">
    <title> 空中监狱 </title>
    <actor> 尼古拉斯 凯奇</actor>
    <rating>家长指引</rating>
  </movie>
  <movie type="恐怖片">
    <title> 幽灵 </title>
    <actor> 黛米 摩尔</actor>
    <actor> 帕特里克 斯韦兹</actor>
    <rating>家长指引</rating>
  </movie>
</movies>

```

对于内部 DTD，DTD 代码和 XML 代码包含在一个文档中。该文件的扩展名为.xml（如 Example 4.xml）。在例 4 中，DOCTYPE 语句表示 Document Type Declaration（文档类型声明），而方括号中的语句，则表示 Document Type Definition（文档类型定义）。这可能会产生混淆，但从上下文来看就很清楚指的是哪种意思。格式良好的 XML 文档必须包含至少一个根元素，即单个元素声明。另外，在声明中指定的 DOCTYPE 名称必须与该根元素匹配，在本例中为 movies。

外部 DTD:

外部 DTD 在文档内容之外，并带有扩展名.dtd。在 XML 文件的开头添加的 DTD 引用告诉 XML 处理器在哪里查找外部 DTD、关于其作者的信息、DTD 的目的以及使用的语言。外部 DTD 在 XML 文件的开头通过 SYSTEM 关键字引用。将外部 DTD 声明至 XML 文档的语法如下所示。

```

<?xml version="1.0"?>
<!DOCTYPE movies SYSTEM "Example 5.dtd">

```

在本例中，Example 5.dtd 是在文档内容之外的外部 DTD，并带有必需的扩展名.dtd。

例 5:

```

<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE movies SYSTEM "Example5.dtd">
<movies>
  <movie type="冒险片">
    <title> 空中监狱 </title>
    <actor> 尼古拉斯 凯奇</actor>
    <rating>家长指引</rating>
  </movie>

```

```
<movie type="恐怖片">
  <title> 幽灵 </title>
  <actor> 黛米 摩尔</actor>
  <actor> 帕特里克 斯韦兹</actor>
  <rating>家长指引</rating>
</movie>
</movies>
```

此文件保存为 Example 5.xml。外部 DTD 由文件 Example 5.DTD 提供。

```
<?xml version="1.0" encoding="gb2312" ?>
<!ELEMENT movies (movie+)>
<!ELEMENT movie (title,actor+,rating)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT actor (#PCDATA)>
<!ELEMENT rating (#PCDATA)>
<!ATTLIST movie type CDATA #IMPLIED>
```

DTD 中的内部实体声明：

内部实体的内容在 XML 文档中出现。语法如下：

```
<!ENTITY entity-name "entity-value">
```

DTD 示例：

```
<!ENTITY writer "查尔斯·狄更斯">
<!ENTITY copyright "Copyright XML101.">
```

XML 示例：

```
<author>&writer;&copyright;</author>
```

DTD 中的外部实体声明：

外部实体指内容在 XML 文档之外的实体。SYSTEM 关键字用于指定所有在文档之外的实体。语法如下：

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

在以下示例中，XML 处理器将实体引用替换为指定文档的内容。

```
<!ENTITY writer SYSTEM "http://www.xml101.com/entities/entities.xml">
<!ENTITY copyright SYSTEM "http://www.xml101.com/entities/entities.dtd">
```

XML 示例：

```
<author>&writer;&copyright;</author>
```

DTD 中的参数实体：

只在 DTD 文档中出现。

DTD 示例:

```
.....
<!ENTITY % p "a">
<!ELEMENT roster ((%p;)+)>
<!ELEMENT %p; (name,...)>
.....
```

其中 p 是参数实体, a 是 p 的省略值。根据 p 值的不同, DTD 中 roster 子元素也不同。下面是参数实体的一个具体应用, 两个不同元素的 XML 文件共同关联一个 DTD 文件。第一个 XML 文件是学生花名册, 保存为 Example 6.xml。

例 6:

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE roster SYSTEM "Example 6.dtd" [
<!ENTITY % p "student">
]>
<roster>
  <student ID="s101">
    <name>李华</name>
    <sex>男</sex>
    <birthday>1978.9.12</birthday>
    <score>98</score>
    <skill>Java</skill>
    <skill>Oracle</skill>
    <skill>C Sharp</skill>
    <skill>SQL Server</skill>
  </student>
</roster>
```

上面代码先用内部 DTD 声明, 把参数实体 p 设为 student, 再引用外部 DTD 验证。

第二个 XML 文件是教师花名册, 保存为 Example 7.xml

例 7:

```
<?xml version="1.0" encoding="gb2312" ?>
<!DOCTYPE roster SYSTEM "Example 6.dtd" [
<!ENTITY % p "teacher">
]>
<roster>
  <teacher ID="t101">
    <name>张老师</name>
    <sex>女</sex>
    <birthday>1968.3.1</birthday>
    <skill>Java</skill>
```

```

    <skill>Oracle</skill>
    <skill>C Sharp</skill>
    <skill>SQL Server</skill>
  </teacher>
</roster>

```

上面代码先用内部 DTD 声明，把参数实体 p 设为 teacher，再引用外部 DTD 验证。

以上两个 XML 文件用同一个 DTD 验证，DTD 代码保存在 Example 6.dtd 中，见如下实例。

```

<?xml version="1.0" encoding="gb2312"?>
<!ENTITY % p "a">
<!ELEMENT roster ((%p;)+)>
<!ELEMENT %p; (name,sex,birthday,score?,skill+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT birthday (#PCDATA)>
<!ELEMENT score (#PCDATA)>
<!ELEMENT skill (#PCDATA)>
<!ATTLIST %p; ID #REQUIRED>

```

五、命名空间

命名空间是在 XML 文档中可以用作元素或属性名称的名称集合，它们标识来自特定域（标准组织、公司、行业）的名称。命名空间使浏览器可以执行以下操作。

- 组合来自不同源的文档，并有助于识别元素或属性的源。
- 访问 DTD 或用于验证文档的元素和属性的其他描述。

Uniform Resource Identifier（统一资源标识符，URI）识别 XML 的命名空间。URI 包括 Uniform Resource Name（统一资源名称，URN）和 Uniform Resource Locator（统一资源定位符，URL）。URL 包含对 Web 上的某个文档或 HTML 页面的引用。URN 是标识 Internet 资源的全球唯一编号。

例如：有 3 个名为 batch 的元素。第一个 batch 指 Aptech 培训中心的一批学员，第二个 batch 指一批产品，而第三个 batch 指一批游客。可以使用唯一的 URI 标识 batch 元素。第一个 batch 与 Aptech 计算机教育的 URI 关联，第二个 batch 与茶业 URI 关联，第三个 batch 与旅游的域 URI 关联。要在文档中使用该元素，可以使用以下语法。

```

http://www.Aptech_edu.ac.batch
http://www.tea.org.batch
http://www.digicam.org.batch

```

要通过这种方式逐个引用元素十分麻烦。

命名空间的必要性:

人们开始重用和扩展标准的 DTD 时,因为文档交换,所以对于重名的元素,XML 解析器可能出现冲突。如果使用 DTD 中已经存在的元素或属性名称来扩展该 DTD,解析器将无法获知正在使用的是哪一个。命名空间有助于标准化元素和属性,并为它们加上唯一的标志。

命名空间确保元素名称没有冲突,并阐明它们的来源,但它们不确定如何处理元素。XML 解析器必须知道元素的意义以及如何处理它们。

命名空间的语法:

将一个前缀与可以用作命名空间的 URI 关联,如下所示。

```
xmlns:[prefix]="[命名空间的 URI]"
```

xmlns: 是保留属性。由于 xml 是保留的字符串,它不能用作前缀名称的开头,可以使用 XML 标签中允许的任何其他字符。前缀用作命名空间的别名。例如:

例 8:

```
<?xml version='1.0' encoding="gb2312"?>
<cameras xmlns:digital="http://www.digicam.org"
          xmlns:photo="http://www.photostudio.org">
  <digital:camera prodID="P663" name="傻瓜相机"
    pixels="410000" output_res="640 x 480" int_mem="2 MB"
    price="300.99"/>
  <photo:camera productID="K29B3" name="超级 35 毫米照相机"
    lens="35 毫米" zoom="70 毫米" warranty="1 年" price="99.00"/>
</cameras>
```

在以上 XML 代码中,有关数码相机的信息属于 digital 命名空间,而有关传统相机的信息则属于 photo 命名空间。这样就能够根据两种相机的特定类型验证和处理它们的信息,使数据更灵活和精确。

请注意,尽管前缀(digital 和 photo)只在元素名称中出现,但该元素的属性也属于该元素的命名空间。这意味着 digital:camera 元素上的所有属性也属于 digital 空间。

属性和命名空间:

除非带有前缀,否则属性属于它们的元素的命名空间。

```
.....
<ins.batch ins.type="thirdbatch">夜班</ins.batch>
<ins.batch type="firstbatch">早班</ins.batch>
<ins.batch>下午班</ins.batch>
</ins.batch-list>
.....
```

在以上代码中,两个属性均视为属于相同的命名空间。

```
.....
xmlns="http://www.Aptech_edu.ac"
```

```
xmlns:tea_batch="http://www.tea.org">
<batch-list>
  <batch type="thirdbatch">夜班</batch>
  <batch tea_batch:type="thirdbatch">第三批茶</batch>
  <batch>下午班</batch>
  .....
```

在上面代码中，属于"http://www.Aptech_edu.ac"命名空间的 batch 元素拥有茶业领域（"http://www.tea.org"命名空间）的 tea_batch:type 属性。

可以包括两个名称相同但属于不同命名空间的属性，如下所示。

```
<batch type="firstbatch" tea_batch:type="firstbatch">第一批茶</batch>
```

命名空间应用：

以下示例演示了如何应用两个命名空间，分别是 http://www.Aptech_edu.org 和 http://www.tea.org。源文件是 Example 9.xml。

例 9：

```
<?xml version="1.0" encoding="gb2312" ?>
<sample xmlns:ins="http://www.Aptech_edu.org"
  xmlns:tea="http://www.tea.org">
  <ins:batch-list>
    <ins:batch>夜间培训批次</ins:batch>
    <ins:batch>早间培训批次</ins:batch>
    <ins:batch>午间培训批次</ins:batch>
    <ins:batch>
      第一批茶<tea:batch>批号 333 </tea:batch>
    </ins:batch>
    <ins:batch>
      第二批茶<tea:batch>批号 222 </tea:batch>
    </ins:batch>
  </ins:batch-list>
</sample>
```

上述代码是一个批次列表，但来自两个领域。

一个是培训领域，命名空间是"http://www.Aptech_edu.org"，前缀是 ins。

另一个是茶叶领域，命名空间是"http://www.tea.org"，前缀是 tea。

Batch-list 是批次列表。

在例 9 中，除了那些具有 tea 前缀的元素，所有 3 个 batch 都属于 ins 代表的命名空间。

这样就能够根据 tea 的两个 batch 的特定类型来验证和处理它们的信息，使数据更灵活和精确。

第十章 用样式表格式化显示

本章目标：了解 XML 的应用范围
了解 XML 的文档结构
理解格式正规的 XML 文档的特点
熟悉有效的 XML 文档的编写规则
理解命名空间
本章重点：熟悉有效的 XML 文档的编写规则
本章难点：理解命名空间

六、XML 的应用范围

人类一直在不断地尝试改进自己的发明，其中也包括人类最伟大的发明——文字的构成。第一个文本处理系统是用纸笔记录文字。现在，计算机文本处理器已经取代了手工处理，它不仅包含原始文档，还负责设置格式、出版和管理。在这些方便的功能整合到字处理之前，是由排字工人遵循书面标记说明来完成所有格式编排的。正是利益于这种实践，人们将“标记”这个词加入到 HTML 和 XML。顾名思义，标记是指加上记号。文本处理环境（如 XML）中使用了相同的标记过程。本意讲述标记语言的历史和创建 XML 文档的方法。

使用脚本语言或 DHTML 能够以各种方式显示信息。这就要求必须为相同的输出编写不同的代码以供不同的浏览器使用，因为这些语言不能跨浏览器兼容。

XML (eXtensible Markup Language, 可扩展标记语言) 克服了这些缺点。顾名思义, XML 是可扩展的, 即开发人员可以定义自己的一组标签, 并使其他的人或程序能够理解这些标签。HTML 是单标记语言, 为特定应用设计, 而 XML 则是一系列的标记语言。因此, XML 比 HTML 灵活得多。实际上, 由于 XML 标签表示了数据的逻辑结构, 不同的应用可以通过不同的方式来解释和使用这些标签。Web 上的数据大多是继承的, XML 继承了 SGML 和 HTML 的优点。也就是说, 它不仅继承了 SGML 的特色, 还结合了 HTML 的特色。它采用了 SGML 的主要框架, 有时, 人们也将 XML 称为 SGML 的子集。因此, HTML 是 SGML 的应用, 而 XML 是 SGML 的子集。下图显示了标记语言的层次结构。

```
<html>
  <body>
    <form>
      <select name="cars">
        <option value="volvo">Volvo
        <option value="saab">Saab
        <option value="fiat">Fiat
        <option value="audi">Audi
      </select>
    </form>
  </body>
</html>
```