

第二章 XHTML：浏览器上的新大陆 .....	2
2.1 一切从语义开始 .....	2
2.2 网页的构成 .....	3
2.3 常见标签 .....	4
2.3.1 标题 .....	5
2.3.2 内容 .....	6
2.3.3 列表 .....	9
2.3.4 表格 .....	12
2.3.5 表单 .....	13
2.3.6 注释 .....	17
2.4 比HTML多出一个X .....	17
2.4.1 还多了什么 .....	17
2.4.2 谁影响了网页的样子 .....	19
2.4.3 坚持Strict Markup .....	20
2.4.4 语义化 .....	20
2.5 按语义来分类 .....	21
第四章 用XHTML和CSS来“摆”网页 .....	22
4.1 开始第一个页面 .....	22
4.2 文字的表现 .....	24
4.2.1 字体的外型 .....	24
4.2.2 颜色的定义 .....	31
4.2.3 背景的定义 .....	32
4.3 最简单的布局 .....	34
4.3.1 Margin 与 Padding .....	34
4.3.2 单列固定宽度居中 .....	36
4.3.3 单列自适应宽度 .....	37
4.3.4 奇怪的高度 .....	38
4.3.5 边框的定义 .....	40
4.4 郁闷的盒模型 .....	42
第十三章 亲和力 .....	44
13.1 概念 .....	44
13.2 Web内容可访问性指南 .....	45
13.2.1 并非所有的内容都是显示的 .....	45
13.2.2 保持原有功能的有效性 .....	49
13.2.3 让文字看得见 .....	50
13.2.3 热键操作 .....	52
13.2.4 导航 .....	55
13.2.5 标题与内容 .....	56
13.2.6 语言与编码 .....	57
13.2.7 亲和力声明 .....	58
13.2.8 建议 .....	58

## 第二章 XHTML：浏览器上的新大陆

Web标准带来的是什么？XHTML？

其实XHTML并没有什么变化，带来的是思维的变化。

### 2.1 一切从语义开始

为什么我选的第一个例子是文章呢？因为我想从语义上引导大家来认识XHTML，例子里面有表示标题`<h1>`和表示段落的`<p>`，是不是跟我们小学时写作文的格式很像呢？网页一样可以有语义的，使用带有语义的标签不会影响网页的设计，因为语义属于结构，而我们看到的是通过表现处理过后的网页，你可以把`<h1>`使用CSS定义得比`<p>`还要小，同样也可以把`<p>`定义得比`<h1>`还要大。

为什么要用语义呢？让我们来看图2.1：



图2.1

我们要表达出来的东西通过网页，网页通过计算机连接互联网传达到别人，而负责中转的计算机没法智能到像我们人一样，它可能理解不了我们要表达的内容，它没办法像我们一样可以通过视觉来判断内容的真实性，可能会把我们要表达的信息传达给不相关的人，因此，我们需要语义化来把网页表达成计算机能理解的语言。

其实遵循语义化的目的，不仅仅体现在对搜索引擎的优化上，语义化的代码更具有亲和力，更主要的是会使网页逐渐标准化，任何一种文档格式，只要有统一的标准，那就会大大加强其内容的利用率。最典型的例子，当属RSS（一种基于XML的文档标准，用于对网站内容的聚合）了，如果它不采用统一的标准，我们就不可能通过软件甚至网页去解析它，也不可能这么方便的就获取到其它网站的信息。语义化的终极目标，是构建一个语义网（Semantic Web），这将会让每个人在一个充满知识的网络中受益。

那么，我们为什么不使用计算机已经熟知的带有语义的标签呢？除了`<h1>`和`<p>`还有哪些标签是带有语义的呢？还有很多，例如：表示强调的`<em>`和`<strong>`；表示插入的`<ins>`和删除的`<del>`；表示计算机代码的`<code>`；表示引用别人说的`<cite>`、`<q>`和`<blockquote>`；表示数据的`<table>`等等，在后面的章节中我们都会单独来学习它们的使用方法。

可能你还在疑惑有没有必须这样做？那我们先来看一下小例子，假如我们使用有这样的HTML：

```
<p>传说中的<s>借</s><u>错</u>别字。</p>
```

```
<p>传说中的<del>借</del><ins>错</ins>别字。</p>
```

它在浏览器的表现如图2.2，两行不同的代码看起来显示是一样的。

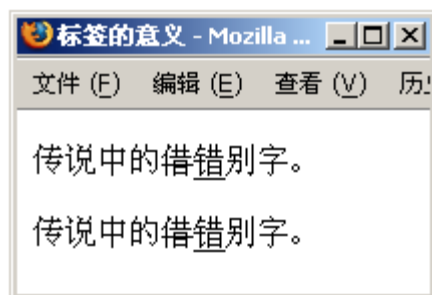


图2.2

假如我把浏览器里的内容拷进Word里会是怎样的情况呢？这时它们的区别就像图2.3所标示的。

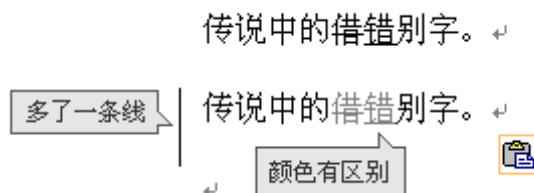


图2.3

把鼠标分别移到“借”和“错”上，可以看到像图2.4这样的提示。

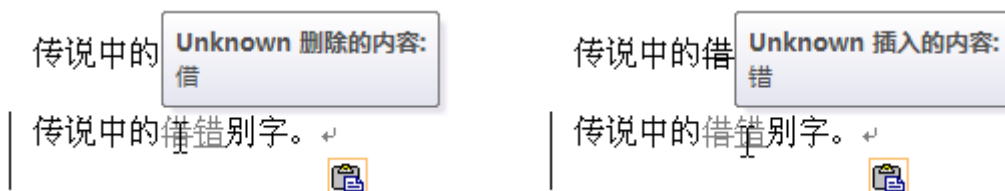


图2.4

Word 2007读懂了我使用的标签了，并对我的内容做出了处理，并且还提供了一些后续的操作来处理好内容。

这只是个小例子，希望能帮助你理解语义的作用，在后面的学习中，我们将学习更多相关的知识，在此之前，先来了解一下网页的构成。

## 2.2 网页的构成

对于浏览器而言，网页不是由内容和图片拼出来的，而是由代码拼出来的，那么，网页需要什么来拼呢？

随意打开几个你喜爱的网站，点击浏览器工具栏中的“查看”→“页面源代码（IE浏览器是源文件）”，会发现几乎所有的网站都会有一些相同的代码，如：<html>、<title>、<body>等，这些就是用来拼网页用的代码。那一个网页应该包括多少必不可少的代码呢？正常来说，网页是由HTML（超文本置标语言，HyperText Markup Language，一种为创建网页和其它可在网页浏览器中看到的信息设计的置标语言）组成的，它的最基本结构是这样：

```
<html>
  <head>这是网页的头部</head>
  <body>这是网页的内容</body>
</html>
```

这个文件的第一个标签是<html>，它是用来告诉浏览器，这是一份HTML，从这里开始，直接对应的最后一个</html>结束，其中，<head>是网页的头部信息，里面包含关于所在网页的信息，它主要是被浏览器所用，不会显示在网页的正文内容里，头部信息通常包括有<title>（标题）、<link>（链接）、<style>（样式）、<script>（脚本）以及<meta>（关于信息）等，而<body>里面才有你在浏览器里看到的内容，也就是

网页的正文。

但是，我们现在看到的网页的代码中最开始常常不是以<html>标签开头的。例如我们会看到像类似这样的：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

DOCTYPE是什么呢？我们把它称之为“文档类型”，虽然DOCTYPE被许多人忽视，但在遵循标准的任何Web文档中，它都是一项必需的元素。其中DOCTYPE就是Document Type的简写，它的作用就是告诉浏览器这个文档是个什么，用的是什么“型号”的HTML。当浏览器知道此文档的类型和所使用的代码型号后，就会根据不同型号的代码解析出相应的网页来，DOCTYPE直接决定了浏览器的显示效果，可以看出它在网页里占据有很大重要性。

有时，你还会在更前面看到类如这样的内容：

```
<?xml version="1.0" encoding="utf-8" ?>
```

它是XML声明，XHTML应当使用的MIME Type为application/xhtml+xml，同样也可以使用MIME Type text/html。声明是XML文件使用的可选声明，它定义使用的XML版本和字符编码类型等设置，但在一些浏览器（IE6）中，声明会影响网页的显示。

这里提到了MIME（Multipart Internet Mail Extension）Type，这是什么呢？首先，我们要了解浏览器是如何处理内容的。在浏览器中显示的内容有 HTML、有 XML、有 GIF、还有 Flash……那么，浏览器是如何区分它们，判断什么内容该用什么形式来显示呢？答案是 MIME Type，也就是该资源的媒体类型。媒体类型通常是通过 HTTP 协议，由 Web 服务器告知浏览器的，更准确地说，是通过 Content-Type 来表示的，例如Content-Type: text/html，表示内容是 text/html 类型，也就是超文本文件。为什么是“text/html”而不是“html/text”或者别的什么？MIME Type 不是个人指定的，是经过 ietf 组织协商，以 RFC 的形式作为建议的标准发布在网上的，大多数的 Web 服务器和用户代理都会支持这个规范。

通常只有一些在互联网上获得广泛应用的格式才会获得一个 MIME Type，如果是某个客户端自己定义的格式，一般只能以 application/x- 开头。XHTML 正是一个获得广泛应用的格式，因此，在 RFC 3236 中，说明了 XHTML 格式文件的 MIME Type 应该是 application/xhtml+xml。

DOCTYPE和XML声明的功能，将会在2.4.2谁影响了网页的样子中为大家作详细的介绍。

HTML是一种基本的Web网页设计语言，它从出现发展到现在，规范不断完善，功能越来越强。但是依然有缺陷和不足，人们仍在不断的改进它，使它更加便于控制和有弹性，以适应网络上日新月异的应用需求。2000年底，国际W3C（World Wide Web Consortium）组织公布发行了XHTML 1.0版本，XHTML 1.0是一种在HTML 4.0基础上优化和改进的新语言，是一种增强了的HTML，目的是基于XML的应用。它的可扩展性和灵活性将适应未来网络应用更多的需求。

在了解XHTML之前我们必须先了解一些标签。

## 2.3 常见标签

首先我们要明确一点：什么是**标签**、什么是**标签的属性**、什么是**属性的值**？我们拿一个网页中最常见的链接来举例，先看下面这行链接代码：

```
<a href="http://www.blueidea.com/" title="网站设计与开发人员之家">蓝色理想</a>
```

其中的“a”，就是HTML的一种标签，用它来标识的内容就表示链接。

标签通常都是由尖括弧包围起来，并分为开始标签（<a>）与结束标签（</a>），内容就被包围在这两个标签之间。

上面的这个例子，用到了两个属性：href和title，分别表示链接的地址和标题，属性的值要用等号与属性相连，并用双引号标注（如："http://www.blueidea.com/"），属性与属性的值通常都写在开始标签内。

那常常听到别人说的元素是什么呢？一个元素由一个标签来定义，包括开始和结束签以及其中的内容。

每个标签都可能有不只一个属性，而每个属性却只能赋予一个属性值。在这里我不想过多的涉及标签的属性涵义及属性的取值，我想强调的是些常用标签的用法与涵义。

如果您想了解哪些标签有哪些属性、都可以取哪些值的话，建议您去买一本《HTML与XHTML权威指南（第6版）》，这是一本很好的工具书，它详细的介绍了(X)HTML的所有标签，放在手边，以后可以随时查阅。

有些标签并不是成对出现的（例如meta、img、br等等），它只有开始标签，没有结束标签，在HTML中它们可以不用结束，但是在XHTML中你必须结束它们，而结束的方式就像XML简写空标签一样，如：  
<p></p>，它内容是空的，就写成这样<p />，同样，<br />就是<br></br>，只是我们把关闭的空标签简写了，也就是很多教程教你添加一个“/”为它结束一样。

## 2.3.1 标题

如果一篇文章没有标题，你就必须粗略看一部分后才能明白它要讲的内容，网页也一样，没有了标题，你就没办法一下子明白它的内容。

- **title**

<title>里的文本并不会出现在网页里面，Windows上大多浏览器都默认出现在浏览器左上角，例如：

```
<head>
```

```
    <title>欢迎大家来学Web标准</title>
```

```
</head>
```

表现如图2.5：



图2.5

- **h系列**

h系列一共有6个标签，从h1至h6，算是6个等级，它们其实就像大纲的级别一样，一级一级下来，<h1>是代表最顶级标题，严格来说它只会在网页上出现一次，就像你的文章一样，只会有一个标题，那文章副标题呢？我总是习惯性就把<h2>当成副标题，然后把sidebar（侧边栏）的栏目名用<h3>来使用。然而，这并不是最理想的做法，我更希望能用h系列把网页划分成像大纲一样的结构。但是，我们做的网页结构往往不会让我们随意划分，因为，网页不仅仅只是一份简单的文档，它包含有更多的内容。

如果你想让网页有更多方面的应用时，你可以按级使用h系列，就像这样：

```
<h1>一级标题</h1>
```

```
.....
```

```
<h2>二级标题</h2>
```

```
    <h3>三级标题<h3>
```

```
    ...
```

```
</h2>二级标题</h2>
```

```
...
```

而不是像这样：

```
<h1>一级标题</h1>
```

```
    <h3>三级标题<h3>
```

```
    ...
```

```
<h2>二级标题</h2>
```

```
...
```

```
    <h4>四级标题</h4>
```

```
...
```

为什么呢？如果你看着一本书，目录是这样的话：

## 1.1 理解Web标准

## 2 XHTML 浏览器上的新大陆

### 3.2.6 属性选择符

.....

你会不会觉得很奇怪呢？虽然说，我们做出来的网页在通过浏览器看时，不管是<h1>还是<h6>，都可以通过CSS把它们定义成一样，但是，你试着从网页拷贝内容进入Word或者其他软件时，电脑不会用看的，它只会读取它认识的东西，如果你的网页结构良好，Word就能帮你自动处理成一份结构良好的文档。

#### ● caption

表格也是有标题的，更准确的说，<caption>是指定表格的简要描述。因为我们以前一直拿表格来布局，就没注意到这个，在学习不是用来布局的表格时，我们就应该关注它了。关于<caption>详细的介绍参考后面的介绍。

#### ● legend

表单元素<fieldset>的标题。关于<legend>详细的介绍参见4.8表单。

## 2.3.2 内容

#### ● p

这是一个有特定语义的标签，表示段落，可以用来区分段落，使用很简单，就像这样：

<p>从前有座山，山里有座庙，庙里有个和尚，和尚正在睡懒觉，挖哈哈！</p>

因为p是表示段落，更多的时候段落只是出现在文章里，所以，并不是所有的内容都要放在<p>里面，最常用的地方自然是出现在类似文章的地方。<p>也可以存在于其他元素中，如：

<blockquote><p>这是你想引用的一段话</p></blockquote>

基本所有的浏览器中对<p>都有一个上下的边距，但是，不管默认是什么样子，我们都可以通过CSS改变它。

#### ● br

有时候，我们的内容并不是像段落一样，而是更像诗歌一样，我们就没必要拿一个个<p>去分割它们，例如我们可以像这样：

<p>程序不是梦，<br />

生于无形无象的禅中，<br />

我们只是那做梦的人。</p>

注意到了没，br这个标签的写法有点特别，它为什么不是<br>或者<br></br>，还记得我们上一节提到的缩写没。

br是break的意思，属于表现层的标签，但是并非所有表现层的东西都不能用，在XHTML 2.0中有line可以代替<br>使用，但是现在应该按实际情况应用它。而且现在我们一样可以用CSS控制它不换行，例如把它的display属性的值改成none。

#### ● wbr

请原谅我告诉你wbr这标签，一个已经不再Web标准中存在的标签，虽然很多人（比如Google）还在用着它，因为它有个作用是CSS无法达到的。如果你在网页上显示很长的英文单词或者网址时，我们并没有很好的办法在所有的浏览器中用CSS控制它换行，但是这个标签可以做到，作用于不会自动换行的区域中时，它会根据实际情况决定是否换行。假如有这样一个例子：

<ul>

<li><a href="#" title="妈妈说就算你注册的域名再长google都能搜索出来">

http://www.mamashuojiusuannizhucede yumingzaichanggoogledounengsousuochulai.cn/</a></li>

<li><a href="#" title="妈妈说就算你注册的域名再长google都能搜索出来">

http://www.<wbr />mama<wbr />shuo<wbr />jiusuan<wbr />ni<wbr />zhucede<wbr />yuming<wbr />zaichang<wbr />google<wbr />douneng<wbr />sousuo<wbr />chulai<wbr />.cn/</a></li>

&lt;/ul&gt;



图2.6

如果宽度足够的话，<wbr>并没有起作用（图2.6），但是，如果宽度没有连着的英文那样长的话就会出现像图2.7这样的显示：

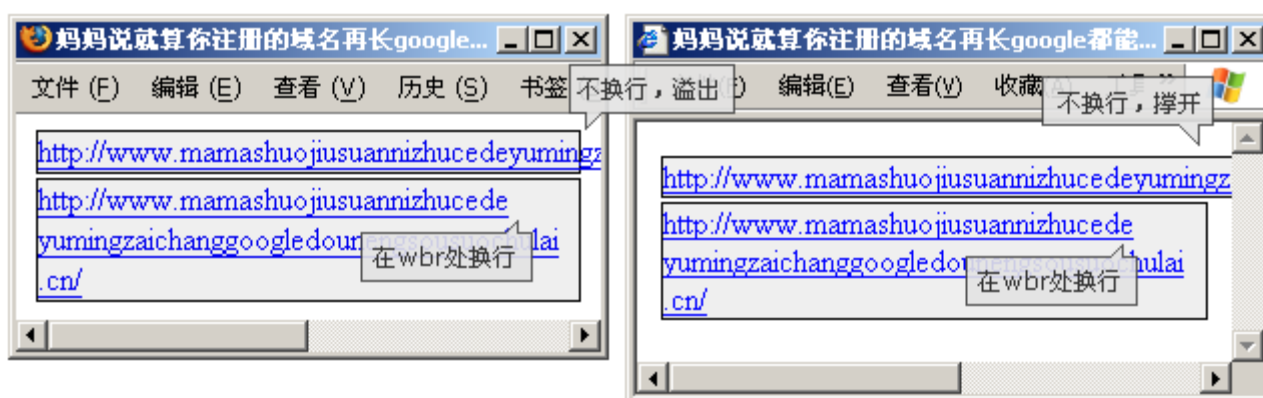


图2.7

除了IE5~6会撑开容器外，其他的浏览器都是溢出容器，<wbr>能在需要换行的地方进行换行，虽然像这样长的网址几乎不存在，但在制作网页时，常常会遇到类似这种情况，除了网址，一些中文的字符也不会自动换行，虽然IE有私有的CSS属性能让它强制换行，但其它浏览器就必须使用其他方式。

#### ● nobr

与wbr作用相反的标签，同样不再Web标准中存在的标签，它是强制某些会换行的内容让其不换行，这个标签并不推荐使用，因为我们可以通过CSS来实现它的功能。

避免使用表现类的标签可以让我们向实现结构与表现的分离走近一小步，同时也要避免教条般的使用，应按需要使用。

#### ● a

<a>可以说是(X)HTML中最伟大的一个元素，需要指定 href 或 name 属性。当有href属性时，<a>就是一个超链接，用于链接到其他网页。

```
<a href="http://www.blueidea.com" title="蓝色理想 - 网站设计与开发人员之家">蓝色理想</a>
```

```
<a href="#csser" title="点击到达分类目录">csser</a>
```

当有name属性时，<a>就是一个锚点，锚点的作用是：如果你的网页很长的话，你可以使用锚点跳到页面的某一部分，例如：

```
<a name="csser">CSSer团队成员</a>
```

当点击<a href="#csser" title="...">CSSer</a>就可直接将网页滚动到标示<a name="csser">CSSer团队成员</a>的地方。在一些浏览器，如IE、Firefox中，可以使用ID代替name，但这并不所有的浏览器都支持的。

两者可以同时有时，它既是一个超链接，也是个锚点。例如：

```
<a name="csser" href="http://www.csser.org">CSSer </a>
```

#### ● strong & em

这两个都是属于强调某一内容的，其中<strong>是重点强调，一般浏览默认都是把<strong>显示为粗体，<em>显示为斜体，例如：

```
<p>今天我买了一个<em>手表</em>，<strong>瑞士Cartier </strong>多功能计时表。</p>
```



我在句子中强调了手表，还重点强调了牌子（瑞士Cartier），是不是跟我们小学写作文差不多呢？在网页中，我们也可以把强调表现出来。虽然，可能想要的效果不是斜体，不过我们依然可以通过CSS再次定义它的表现。

## ● ins & del

我们已经习惯了当网页里的内容有错别字或少字时，就直接修改网页，这并没有什么问题，但是，如果是已经发布的技术文档或者学术论文呢？我们可能就不能像铅笔写字一样，擦掉改掉就行。

<ins>标签表示插入的内容，而<del>标签就是删除了的内容。

<p>有个<del>借</del><ins>错</ins>别字，挖哈哈！</p>

但是，这样还是不足够的，在2.1 一切从语义开始 中的例子Word的提示是“Unknown 删除的内容”、“Unknown插入的内容”，因为我们还无法知道修改的原因，更完整的写法是这样。

<p>美美是个<ins cite="cause.html" datetime="20061225">大</ins>笨蛋，挖哈哈！</p>

在<ins>里，cite属性表示插入的原因，可以链入网址，而datetime属性表示插入的时间，而在<del>里就是表示删除的原因和时间。一般情况下，浏览器会把<ins>标签内的内容添加下划线，给<del>标签内的内容加删除线，属性cite和datetime是不会显示。

可能这样看起来并没有什么用，其实有很多作用的，可以让开发人员在开发文档时相互合作，而且也保持了一些编辑痕迹（比如可以反应别人是什么时候修改、为什么修改的等），同时也可以完成版本控制。当然，这些在现在强大的版本控制软件下已经显得不是那样重要。

## ● abbr & acronym

有时，我们页面会出现一些缩写的词，例如XFN，如果知道的人自然会明白是XHTML Friends Network，可没听过的人呢？<abbr>的含义是“缩略语”，它是对单词或者短语的简写形式的统称。<acronym>的含义是“字头缩略语”，它特指单词形式的缩略语，好像听起来差不多一样，还是先看一下例子吧。

<abbr title="World Wide Web">WWW</abbr>

<acronym title="Radio detecting and ranging">Radar</acronym>

比如说，如果你把DOM念成dom，它就是一个字头缩写；如果你念成三个字母D-O-M，它就不是一个字头缩略词——但它仍是一个缩略语。<abbr>是表示缩略语，而<acronym>是表示字头缩略语。字头缩略语一定是缩略语，但是缩略语不一定是字头缩略语，或者这样说，需要用上<acronym>的也可以使用<abbr>，而使用<abbr>的却不一定能使用<acronym>。在XHTML 2.0中保留<abbr>，可以说<abbr>比<acronym>含义更广，应该废弃<acronym>，以免混淆。

使用<abbr>还有一个问题，就是IE6及以下版本的IE浏览器并不认识这个标签，不过也有变相解决的方式，例如用插入span的方式来代替。

## ● dfn

有的时候，一个词别人不一定知道是什么，但又不是缩略语，例如：

<p><dfn title="Mozilla公司推出的一款网页浏览器">Firefox</dfn>作为开发工具很不错。</p>

它有点像我们写文章时的：“Firefox（Mozilla基金会推出的一款网页浏览器）作为开发工具很不错”，dfn英文原意是definition，也就是术语定义的意思。

## ● kdb

表示由用户从键盘输入的文字，例如：

<p>在浏览器的地址栏打上<kbd>www.loaoao.com</kbd>。</p>

但是，也有人把他理解成用户从键盘输入的按键，例如：

<p>按<kbd>Tab</kbd>测试一下</p>

并能过用CSS把<kdb />定义得有点像键盘按键，如图2.8。

按 Tab 测试一下

图2.8

对于这个这元素，W3C的文档并没有怎样说明，也没有例子参考，谁对谁错也很难说清。我个人更倾向把它用于定义键盘按键，或者可以这样理解，如果<kbd>里面的的是键盘上有的（比如Tab、Esc、Backspace



等)就是直接按键盘上的键,如果不是键盘上的键名的(比如网址、文字等),就由单个字母输入完成。但是这样可能会有些冲突,例如Home,是让输入Home还是按Home键呢?也许这时应该再加上title提示会更好。

- **code & var**

在(X)HTML中,有专门对代码进行标识的标签,那就是<code></code>,例如:

你可以用<code>code</code>标签来表示计算机(程序)代码。

而var是定义一个变量名,例如计算机程序的变量、数学函数的变量等。例如:

```
<code><var>wordcount</var> = 6878;</code>
```

- **pre**

可是你会发现紧紧<code>是不够用的,因为代码都是一行一行的,如果你只用<code>的话,所有用它来标识的代码,就会堆到一行去,这又是为什么呢?因为,<code>是inline-level(行内元素)的,行内元素是不换行的。如果配合<pre>,则<pre>会确保代码的格式与显示出来的一样。

- **samp**

用于定义一个“输出”的内容,例如:

```
<p>你只要点删除,网页就会提示<samp>该会话已移至“已删除邮件”</samp>。</p>
```

- **blockquote**

对于那些一段或者好几段的长篇文字或者其他的引用,就应当使用<blockquote>了,例如:

```
<blockquote cite="http://www.aoao.org.cn">
```

```
    <p>人只要十天不睡觉就会死掉。...我每天只睡一个小时,所以不会死掉。....传说中的文字。</p>
</blockquote>
```

其中的属性cite就是引用的地址。

- **q & cite**

如果只是引用一句话的话,可以选择使用<q>,例如:

```
<p><cite>嗷嗷</cite>曰:<q>珍惜生命,远离Firefox!</q>。</p>
```

注意,这里<cite>的用法是不一样的,在<blockquote>它只是一个属性,而这里是一个标签。

## 2.3.3 列表

在Web标准流行的今天,列表也跟着流行了,试想如果没有Web标准,又有多少人会去使用它呢?一些事物的罗列应使用列表来显示,在XHTML中有三种列表的方法:无序、有序和自定义。

- **ul**

ul是无序列表,就是我们所熟知的圆圈列表,以 <ul>开始,以</ul>结束,每一个列表项都包含在<li>之中,就像这样:

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

效果如图2.9所示:

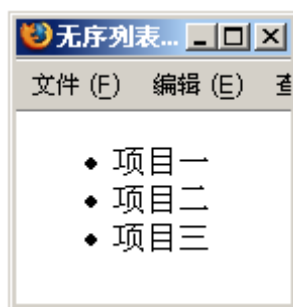


图2.9

直接显示的效果如图2.9，当然我们可以通过CSS去改变它。无序列表常常用于导航条，因为导航条本来就是个列表。虽然<li>是竖排下来的，但是我们依然可以通过CSS让它变成我们想要的样子，在后面有详细介绍。

## ● ol

ol是有序列表，与无序列表最大的不同之处在于，它在前面不再是小黑点，而是数字，虽然CSS能让<ol>与<ul>的外观看起来一样，但是，他们的本质不一样，例如有个歌曲的排行榜：

```
<ol>
```

```
  <li>表白（萧亚轩）</li>
```

```
  <li>有一种爱叫做放手（阿木）</li>
```

```
  <li>慈悲（郑钧）</li>
```

```
</ol>
```

效果如图2.10所示：

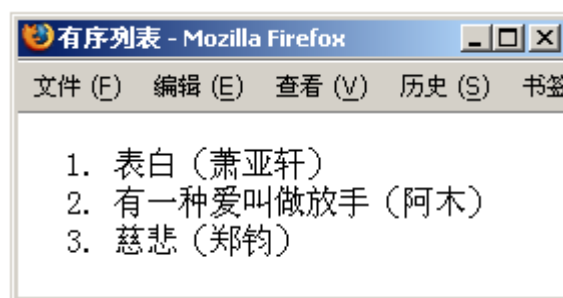


图2.10

直接显示的效果如图2.10。如果是使用无序列表<ul>的话，我们只能直观的从排序来思考，哪一个排第一，虽然常规来说第一个应该是排第一，可有时并不是这样。如果配合CSS的话，我们可以通过“看”来理解，可是，每次我都要说这句：“这些让人类使用是可以，但机器使用起来便有困难了！”

如果你只满足于你的网页只是按你所想的（图片都按设计的样子显示、样式表不丢失）展示在你的访问者的主流浏览器面前，你自然不用在意它。

## ● dl

dl是自定义列表，它有一些不同，可以用来标记一些列表项和描述，以<dl>开始，以</dl>结束。每一个被描述的项目，要包含在<dt>中，而描述的内容要包含在<dd>中，就像这样，我们用来表示某一位作者的作品可以这样呈现：

```
<dl>
```

```
  <dt>《千与千寻》</dt>
```

```
  <dd>剧情说的是10岁的少女千寻与父母一起从都市搬家到了乡下。没想到在搬家的途中，一家人发生了意外。他们进入了汤屋老板魔女控制的奇特世界——在那里不劳动的人将会被变成动物。</dd>
```

```
  <dt>《龙猫》</dt>
```

```
  <dd>和爸爸一起搬到乡下的两姐妹，在家旁的一棵大树下发现了只有好孩子才能看见的TOTORO。其间发生了很多不可思议而有趣的故事。</dd>
```

</dl>

效果如图2.11所示:

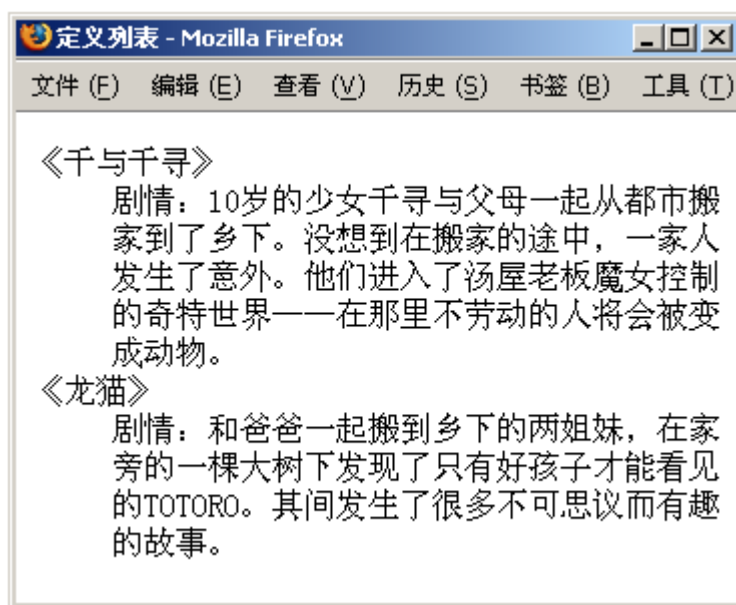


图2.11

在对<dl>的这个标签及子标签<dt>、<dd>的完整性上我跟好友戚佳慧（网名是old9，也是一个Web标准的爱好者，个人博客是：<http://old9.blogsome.com/>）有一个相同的看法，就是最好能多出一个像<ol>、<ul>的子标签<li>来嵌套<dt>、<dd>。

在<dl>里面，不一定要一个<dt>对应一个<dd>，可以是一个<dt>对应多个<dd>，也可以是没有<dd>等等，正因为这样，我们有时也无法确定哪一些是相对应，常常是用这样的方式来判断的：在遇到下一个<dt>之前的<dd>都是对应上一个<dt>。

<dl>可以应用在很多方面，Russ Weakley整理过一些

（<http://www.maxdesign.com.au/presentation/definition/>），我翻译了一部分出来：

<dt>说话者</dt>

<dd>所说的内容</dd>

<dt>图片</dt>

<dd>描述</dd>

<dd>地点</dd>

<dd>摄影师</dd>

<dt>物品名称</dt>

<dd>说明性图片</dd>

<dd>描述</dd>

<dt>网站（链接）</dt>

<dd>描述</dd>

<dt>日期</dt>

<dd>事件</dd>

<dt>事件</dt>

<dd>时间</dd>

<dd>描述</dd>

<dd>地点</dd>

由于<dt>和<dd>并没有单独的其他元素套着，想用CSS控制来对<dt>与<dd>形成个盒状并不是那样自由。

## 2.3.4 表格

曾经，表格是网页设计人最重要的布局排版手段，但是它的存在必不是为了布局，在Web标准中的表格更不应该用来布局。相反，对于多维数据，你应该使用table。为了使数据表格有更强的访问性，了解和使用各种构造表格的组件就很重要了。例如表格标题（<caption>）、摘要（summary 属性）和表格头部单元格（<th>）等。

- **table**

一个表格的开始，表格的元素都应该存在于table标签里面。

- **caption**

<caption>标签可以为表格提供一个简短的说明，也可以当成标题看待。使用的时候，<caption>标签一定要紧接着开始的<table>标签写，默认情况下，大部分可视化浏览器显示表格标题在表格的上方中央，CSS里的caption-side属性用来控制表格标题显示的地方，大部分浏览器只能把表格标题显示在表格的上方或者下方，只有少部分浏览器支持左边或者右边。

- **summary**

<summary>与<caption>的关系有点像标题与描述一样，不过这个属性的值是不会被可视化浏览器显示。如果我们是用看的方式看表格，基本内容都可以看出来，但是使用屏幕阅读器的人就不是那么容易了，他们必须依赖<caption>和<summary>属性才能相对快速地了解表格的内容。当然，通过<summary>可以让表格有更多方面的应用，并非仅仅只是为了满足使用屏幕阅读器的人。

- **tr、th、td**

<tr>用于指定表格中的一行，而<th>是指标题列，大部分浏览器都会把标题列的内容居中并以粗体显示，<td>才是表格中的单元格。

<th>跟<td>这两个都是表格最重要的组成部分，它们必须存在于<tr>的子级，内容都是通过这两者来显示的。

- **thead、tfoot、tbody**

表格也一样有很完整的结构，它有分头部、主体部分、底部。在HTML4中，它们早已经存在了，不过一直没什么人使用它，在XHTML的时代，我们更须要学习它。

尝试创建一份声明是html4 strict类型的网页，使用这样的表格：

```
<table>
  <tr>
    <th>头部</th><th>头部</th>
  </tr>
  <tr>
    <td>内容</td><td>内容</td>
  </tr>
</table>
```

然后通过查看DOM的软件查看结构时（图2.12），会发现，浏览器会自动给表格错误添加上tbody这个标签。

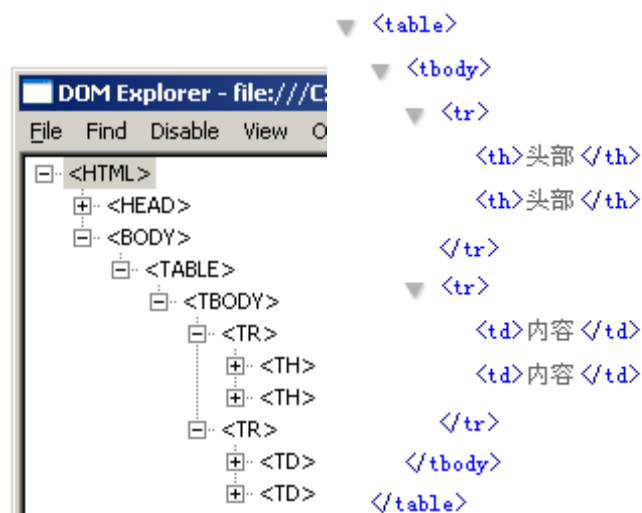


图2.12

不管是HTML还是XHTML，这几个标签都是提倡使用，非一定要用，但是使用有什么好处呢？我个人认为有重要的几点是：

- 减少浏览器为渲染表格时额外的检查及错误的添加，可以加速表格的显示；
- 保持打印时表格的结构；
- 可以为CSS提供更多的控制条件等。

有个特别要注意的地方是：它们顺序是<thead>、<tfoot>、<tbody>，先是表格头部跟底部，然后才是表格的主体内容。

#### ● colgroup、col

有了横向的划分（thead、tfoot、tbody），自然会有纵向的划分。col用于指定基于列的表格默认属性，嵌套的col 属性将覆盖 colgroup属性，而colgroup指定表格中一列或一组列的默认属性。

## 2.3.5 表单

表单(form)是(X)HTML的一个重要部分，主要用于采集和提交用户输入的信息。这里介绍一个很常见的用户登录的例子，在没有CSS的情况下它依然不影响操作，在CSS的渲染下，它可以变得更漂亮。我们先来逐个元素来认识它们。

#### ● from

from是用于申明表单，定义采集数据的范围，也就是<form>和</form>里面包含的数据将被提交到服务器或者电子邮件里。

#### ● fieldset & legend

fieldset用于对表单中的元素进行分组，而在<fieldset>这个组内，<legend>就是它的标题，用于描述<fieldset>所包含的内容。一般可视化浏览器把<fieldset>渲染为带边框的，<legend>一般显示在左上角。要注意的一点是，<legend>元素必须是<fieldset>内的第一个元素，否则<legend>前面的内容将出现在<fieldset>前面，而不是在里面。

fieldset元素不仅仅适用于大块的内容分组，也可以用于选项的分组，就像这样：

```
<fieldset>
  <legend>个人爱好</legend>
  <input type="checkbox" id="likeSleep" name="like"/>
  <label for="likeSleep">睡觉</label>
  <input type="checkbox" id="likeEat" name="like" />
  <label for="likeEat">吃饭</label>
</fieldset>
```

效果如图2.13所示：

图2.13

当然，我们可以通过CSS重新定义它的外观，在后面我们会学习到它。

## ● label

HTML 4.0引入了label对象，你可以使用它将文本与其他任何(X)HTML对象或内部控件关联。无论用户单击<label>或者(X)HTML 对象，被链接的<label>和(X)HTML对象在引发和接收事件时行为一致，而要连接<label>和(X)HTML对象的方式是：将<label>的for属性设置为要关联的(X)HTML对象的id属性。例如：

```
<label for="username">用户：</label><input type="text" id="username" name="username" />
```

当用户单击<label>内的文本（用户：）时，<label>会将焦点设置到文本框。

<label>主要是给表单组件增加可访问性设计的，一般我们都把 label 用在表单里。除以上方法，还可以直接用 label 套嵌整个表单组件和文本标签，例如下面的例子：

```
<label for="likeSleep"><input type="checkbox" id="likeSleep" name="like"/>睡觉</label>
```

```
<label for="likeEat"><input type="checkbox" id="likeEat" name="like" />吃饭</label>
```

图2.14左边为单击“睡觉”在Firefox下选中复选框的，右边是在IE下的截图，注意虚线框的不同。



图2.14

## ● input

这个是在表单里应用得最多的一个标签，一个由属性值来决定标签的意义的标签。

基本结构是：<input type="..." id="..." name="..." value="..." />

### ■ type="text" 单行文本输入框

文本框是一种让访问者自己输入内容的表单对象，通常被用来填写单个字或者简短的回答，如姓名、地址等。一般表现如图2.15。

图2.15

### ■ type="password" 密码输入框

是一种特殊的文本域，用于输入密码。当访问者输入文字时，文字会被“\*”代替，而输入的文字会被隐藏。一般表现如图2.16。

图2.16

### ■ type="checkbox" 复选框

复选框允许在待选项中选中一项以上的选项。一般表现如图2.17。



图2.17

### ■ type="radio" 单选框

当需要访问者在待选项中选择唯一的答案时，就需要用到单选框了。一般表现如图2.18。



图2.18

### ■ type="file" 文件上传框

有时候，需要用户上传自己的文件，文件上传框看上去和其它文本域差不多，只是它还包含了一个浏览按钮（语言是中文的浏览器）。Windows的上传框表现如图2.19。



图2.19

### ■ type="hidden" 隐藏域

元素不会显示在文档里，所以用户也无法操作该元素。该元素通常用来传输一些客户端到服务器的状态信息。虽然此元素不会显示出来，但是用户通过查看 (X)HTML 的源代码还是可以看到该元素属性的值，所以请注意，不要用该元素传递敏感信息，例如密码。

### ■ type="image"

创建一个图像控件，该控件单击后将导致表单立即被提交，并且会提交点击该元素的坐标，例如：

```
<input type="image" name="test" src="test.gif" />
```

将把x坐标以test.x、y坐标以test.y提交，就是在name后面加个.x和.y组成两个新的name，value的任何属性值都将被忽略，src属性指定了img元素。

### ■ type="button"

这是一个普通的按钮，它的值并不会提交，而是显示在按钮上，如果没有通过JavaScript给其添加操作时，它将是普通的可视元素（图2.20）。

这个是普通的按钮

图2.20

### ■ type="submit"

提交按钮，将表单（Form）里的信息提交给表单里action属性所指向的地址。默认的文字是“提交查询”（Firefox中文版），可以通过value属性指定文字（图2.21）。如果用户单击提交按钮提交了表单，并且按钮指定了name标签属性，那么该按钮也将在提交的数据中。

提交查询

图2.21

如果在同一表单中有多个

### ■ type="reset"

重置按钮，默认的文字是“重置”（Firefox中文版），可以通过value属性指定文字。该按钮单击后将重置表单控件为其默认值。

重置

图2.22

这些示例图可能跟你看到的不一样，表单控件会继承系统的主题的风格。

在

### ● textarea



多行文本输入框，也是一种让访问者自己输入内容的表单对象，接受的内容比单行文本输入框更多。  
<textarea id="..." name="..." cols="..." rows="..." wrap="...">多行文本输入框的内容是写在这里面</textarea>

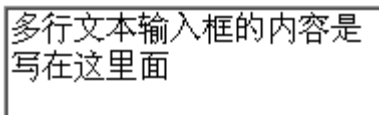


图2.23

其中cols属性定义多行文本框的宽度，单位是单个字符宽度（注：Firefox默认没有滚动条，所以得到额外的宽度）；rows属性定义多行文本框的高度，单位是单个字符高度（IE浏览器是计算单个字符宽度）；wrap属性定义输入内容大于文本域时显示的方式，可选值如下：

- 默认值是文本自动换行；当输入内容超过文本域的右边界时会自动转到下一行，而数据在被提交处理时自动换行的地方不会有换行符出现；
- off，用来避免文本换行，当输入的内容超过文本域右边界时，文本将向左滚动，必须用Return才能将插入点移到下一行；
- virtual，允许文本自动换行。当输入内容超过文本域的右边界时会自动转到下一行，而数据在被提交处理时自动换行的地方不会有换行符出现；
- physical，让文本换行，当数据被提交处理时换行符也将被一起提交处理。

#### ● select & optgroup & option

下拉选择框或列表框，是不是听起来很奇怪呢？没错，<select>也是一个由属性决定表现的，同时，这个属性也是CSS无法代替的，它就是size，看一下例子先（效果如图2.24所示）：

```
<select id="..." name="..." >
  <option value="...">传说中的测试1</option>
  <option value="...">传说中的测试2</option>
  <option value="...">传说中的测试3</option>
</select>
```



图2.24

这种就是默认的下拉选择框，而给添加了size属性的会是怎样呢？试一下（效果如图2.25所示）：

```
<select id="..." name="..." size="4" >
  <option value="...">传说中的测试1</option>
  <option value="...">传说中的测试2</option>
  <option value="...">传说中的测试3</option>
</select>
```

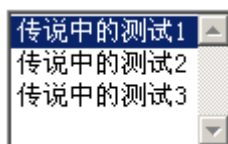


图2.25

这时，<select>变成了列表框，size是决定列表框的行数，当然，我们可以再通过CSS重新定义他的高度。

optgroup标签可以给<select>的options分类，需要使用一个label属性，在可视化浏览器里，它的属性值会在下拉列表里显示为一个不可选的标题（效果如图2.26）。

```
<select id="..." name="...">
```

```

<optgroup label="传说中的分组1" >
<option value="...">传说中的测试1</option>
<option value="...">传说中的测试2</option>
<optgroup label="传说中的分组2" />
<option value="...">传说中的测试3</option>
</optgroup>
</select>

```



图2.26

option就是选择中的内容了，value属性值并不会在浏览器显示，它只是作为提交数据的值。

### 2.3.6 注释

这个其实称不上是一个元素，它也不会显示在网页里，而会被浏览器忽视掉的，但也有些浏览器会当它存在，它本来存在的意义是方便我们编写代码的注释或者备忘用的，就像这样：

```
<!-- 这些代码作用显示标签 -->
```

```
<div class="tags">...</div>
```

在使用注释时要注意的问题是<!-- 跟 -->是配对存在的，如果你使用像这样的：

```
<!--这里是注释的内容 <!--这里是注释的内容-->
```

在IE浏览器里会把整一行都当成注释，但是在一些浏览器（如果Firefox、Opera）在某些情况下（文档声明是严格型的）却不是这样，它们会把前面的“<!--这里是注释的内容”显示出来。如果你想显示“<”之类的特殊符号，你应该使用编码表示，例如使用“&lt;”代替“<”，更多关于特殊符号的编码在后面介绍。

同时也不要再在注释内容中使用“--”，“--”只能发生在(X)HTML注释的开头和结束，如果像这样使用：

```
<!--这里是注释的内容 -- 这里是注释的内容 -->
```

这个同样存在刚才的问题，IE依然会当成注释，而另一些浏览器会把它整行都当成文本显示。

## 2.4 比HTML多出一个X

XHTML是一个基于XML（Extensible Markup Language——可扩展标记语言）的置标语言，看起来与HTML有些相象，只有一些小的但重要的区别，XHTML就是一个扮演着类似HTML角色的XML，所以，本质上说，XHTML是一个过渡，结合了XML的一些功能及HTML大多数的特性。

### 2.4.1 还多了什么

几乎所有的教程都会介绍他们的差别是：

- XHTML必须使用DOCTYPE声明。HTML就没有吗？只是我们一直没注意；
- XHTML的元素必须合理嵌套。HTML就可以乱嵌套吗？浏览器的纵容让我们忘记了很多；
- XHTML标签名称和属性名称必须为小写，属性值使用双引号，元素属性简写是不允许的，所有

XHTML元素必须关闭，这些继承XML的优点：

- 使用命名空间
- 转义字符

前几个比较好理解，先来了解一下什么是命名空间？看一下下面的代码先，为了让大家更容易理解，使用了中文：

```
<程序员>小明</程序员>
```

```
<程序员>小强</程序员>
```

从标签上看，小明跟小强都是程序员，可是我们却无法知道他们来自哪里？那如果换成这样呢？

```
<A公司:程序员>小明</A公司:程序员>
```

```
<B公司:程序员>小强</B公司:程序员>
```

这样是不是很好区分了呢？其实这里的A公司、B公司就是我们要讲的命名空间。在XML中是可以自定义标签和属性的，例如这样：

```
<aoao:wser>嗷嗷</aoao:wser>
```

那机器怎样知道aoao在这个例子中是命名空间呢？所以我们要声明命名空间，就像这样：

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

```
<web xmlns:aoao="http://www.aoao.org.cn">
```

```
  <aoao:wser>嗷嗷</aoao:wser>
```

```
</web>
```

XHTML是一种由HTML向XML过渡的标记语言，它不但继承了HTML的很多标记，也拥有了一些XML的特性，也就是说，XHTML也是可以自定义标记的，它的标签也必须指明所属的命名空间，就像这样：

```
<!DOCTYPE html ...>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:aoao="http://www.aoao.org.cn">
```

```
<head> ..... </head>
```

```
<body>
```

```
  <h1>怎样学习WEB标准</h1>
```

```
  <aoao:wser>嗷嗷</aoao:wser>
```

```
</body>
```

```
</html>
```

其中xmlns:aoao我们已经知道了，那xmlns="..."呢？缺省命名空间用于声明它的元素（如果那个元素没有命名空间前缀）和所有该元素内容中所有没有前缀的元素。假如在缺省命名空间声明里的URI引用为空，那么在声明范围内没有前缀的元素不被认为存在任何命名空间里。

拿上面的代码来说，就是指名了这里所使用的XHTML代码，都是“http://www.w3.org/1999/xhtml”这个命名空间下的。

**转义字符：**在XHTML中，一直有被人提到的就是几个一定要转义的字符，它们分别是：

```
<      &lt;
```

```
>      &gt;
```

```
&      &amp;
```

```
'      &apos;
```

```
"      &quot;
```

其实，不管是应用在HTML还是XHTML，作为严谨的写法，都应该转义，而不是说因为通不过校验才去转义它们。因为这几个字符跟(X)HTML、XML标签相同，而在XML中，直接使用它们还会导致文档结构错误，浏览器的纵容又让我们忘记了很多。

注意：只有“<”字符和“&”字符对于XML来说是严格禁止使用的，其它的都是合法的，为了减少出错，使用实体是一个好习惯。

## 2.4.2 谁影响了网页的样子

这里要讲的并不是通过CSS来改变网页的样子，而是因为浏览器商引起的原因。

### 2.4.2.1 浏览器多模式

当微软开始生产与标准兼容的浏览器时，他们希望确保向后兼容性。为了实现这一点，在IE6以后的版本的IE浏览器里嵌入了两种表现模式：Standards Mode（标准模式或说严格模式——Strict Mode）和Quirks mode（怪异模式或说兼容模式Compatibility Mode）。在标准模式中，浏览器根据W3C所定的规范来显示页面；而在怪异模式中，页面将以IE5，甚至IE4的显示页面的方式来表现，以保持以前的网页能正常显示。

对于这两种模式引起最大的问题就是盒模型的问题，或许现在大家已经忽视了IE5的存在，但是，IE在怪异模式运行的盒模型依然在最新版本的IE7中保存着，一旦应用不得当，IE7将变成跟IE5一样愚蠢。

当然，不只IE浏览器存在两种模式。

Opera 浏览器（Opera 7~8）支持与 IE 相同的两种呈现模式：Quirks Mode和 Standards Mode（有关详细信息，请参阅 <http://www.opera.com/docs/specs/doctype/>），但是Opera9的Quirks Mode又不与之前的Quirks Mode呈现一样，例如不再兼容IE5那种盒模型。

Mozilla Firefox 1+ 支持三种呈现模式：Quirks Mode、Almost Standards Mode（几乎标准的模式）和Standards Mode。Firefox 的 Almost Standards 模式对应于 IE 和 Opera 的 Standards 模式。其中的Almost Standards Mode，除了在处理表格的方式方面有一些细微的差异之外，这种模式与标准模式基本相同。

当然，现在的Opera9在对Web标准的支持方面已经超越了Firefox2的，那么，那些DOCTYPE会引起哪些模式呢？

DOCTYPE 种类有很多种，但是它们有着相同的格式：

```
<!DOCTYPE html 类型 "-//DTD所有者//DTD 文档类型// DTD的语言" "DTD的地址" >
```

这样看可能不大理解，如果换成一个常用的呢？

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

我们可以很清楚地看到这是份HTML的公共声明，由W3C推荐的，类型是XHTML 1.0 Strict，语言是英语，DOCTYPE的地址是<http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>。

明白了没，那这样多的DOCTYPE我们要选择用哪一种才适合呢？又有多少的差别呢？

### 2.4.2.2 开启通往标准的模式

当网页没有使用DOCTYPE声明或者使用HTML4以下（不包括HTML4）的DOCTYPE声明时，基本上所有的浏览器都是使用Quirks Mode呈现。那么，而HTML4或者XHTML呢？

错误的说法：只有XHTML才是Web标准。其实主流浏览器都会把DTD声明是HTML4或者XHTML的网页用Standards Mode呈现。

当页面包含有效的 XHTML 1.0 Transitional DOCTYPE（并且该页被分配为 text/html MIME Type）时，Firefox 会以 Almost Standards Mode呈现该页。当页面包含 XHTML 1.0 Strict 或 XHTML 1.1 DOCTYPE（或者该页被分配为 XML MIME Type）时，该页将以 Standards 模式呈现（有关的详细信息，请参阅 <http://www.mozilla.org/docs/web-developer/quirks/doctypes.html>）。

在附录有一篇“MIME Type引出的两难困境”值得大家阅读，或者应该说，大家应先去阅读一下。

当我们的服务器没有输出application/xhtml+xml的MIME Type而我们又期望有时，我们会选择在文档前加上类如<?xml version="1.0" encoding="utf-8" ?>的XML声明。可是，它却会给我们带来新的问题。引用JJGOD的话：“一切从一个糟糕的浏览器开始，它完全不支持 XHTML。”

IE6，一个占据浏览器市场半壁江山的家伙，它却不能理解这个声明，如果网页的第一行代码不是DOCTYPE，它就要使用Quirks Mode呈现网页。为了兼容老版本而存在的Quirks Mode，会把IE6伪装成IE5，这让我们很多本来可以用的东西变成不可用。

然而，IE的Quirks Mode并不是万恶的，因为IE6对CSS并不是很友好，在某些情况下，让IE6使用Quirks Mode下的CSS反而能达到效果。还好，新版本的IE7已经解决了XML声明会使用Quirks Mode这个问题。但是，试着在开始处加上注释（<!-- ^\_^ -->）IE7的智商一样会降到IE5，甚至IE4的程度，因为IE依然会进入Quirks Mode。

暂时不考虑IE5的话，我们可以选择来让大多浏览器表现一致的DOCTYPE还很多，如HTML 4.0 Transitional、HTML 4.0 Strict、XHTML 1.0 Transitional、XHTML 1.0 Strict、XHTML 1.1等，不管XHTML1.0也好，HTML4.0也好，它们都有过渡型（Transitional）、严格型（Strict）和框架型（Frameset）。似乎“流行”让很多人选择用了XHTML 1.0 Transitional，就像曾经的我一样，没有任何理由，只是因为它的名字是XHTML吗？因为它流行吗？

为什么说暂时不考虑IE5呢？因为它也是一个Web标准支持很差的浏览器，不管你使用什么声明，它只有一个模式，就是Quirks Mode。

那么，我们要选用什么呢？在这里我的推荐是XHTML1.0 Strict，因为它可以让你学到更多东西。

## 2.4.3 坚持Strict Markup

为什么要使用严格型（Strict）的DOCTYPE呢？它可以让浏览器使用它们尽量可能严格、在一定程度上以符合标准的模式来渲染页面。

Tommy Olsson在Web Standards Group的Ten questions for Tommy Olsson一文中很好地阐述了使用Strict的好处：<http://webstandardsgroup.org/features/tommy-olsson.cfm>

我觉得，使用Strict DTD，无论是HTML 4.01 Strict还是XHTML 1.0 Strict，远比讨论是用HTML还是XHTML重要的多。它代表了未来互联网的质量。它将结构和表现分开，使得维护一个站点非常容易。

当然，现在很多人使用XHTML1.0 Transitional DOCTYPE时已经能很好的把结构和表现分开，但是仅仅这样是不够的，对于刚开始接触Web Standards和正确的、语义化的结构的人，认清Transitional和Strict DOCTYPE的区别非常重要。

相信很多人所知道的XHTML1.0 Strict与Transitional的差别就是能不能用target，因为大家可能都没用过其它标签。在HTML4.01 / XHTML1.0 / XHTML1.1严格型的DOCTYPE下，target是不支持的，可它在一些浏览器上依然可用。但是，为了通过W3C的校验，代替者rel出现了，它在JavaScript的配合下完美的绕过了校验。

我知道你一定想问区别是什么，区别就是你的思维，不会因为你用Strict的申明能通得过检验就代表你的网页标准了，两者的区别并非能用文字简单描述出来，在接下来的学习中我们慢慢体会。

## 2.4.4 语义化

虽然这两段代码在没有CSS渲染的情况下，在大多数浏览器中的显示是一样的：

`<p><b>语义化</b>会让Web标准过得更好。</p>`

`<p><strong>语义化</strong>会让Web标准过得更好。</p>`

它们都会以加粗的形式来显示“语义化”这个词，但是它们的意义却不相同。<b>表示以加粗的形式显示内容，而<strong>是表示重点强调的，大多数浏览器会以加粗的形式显示内容。然而，这是没有必要的，如果是为了确定强调内容的显示方式，最好的方法就是使用CSS来定义它们的表现。当你想要的只是视觉上的效果时，就不要使用强调了，或者像这样使用：

`<p><span class="b">语义化</span>会让Web标准过得更好。</p>`

使用没有语义也没有默认表现的<span>，通过CSS来完成视觉效果。这段代码表示的意义就仅仅是

一段文字，没有了特别强调的词。

每次我都是要说：“**这些**让人类使用是可以，但机器使用起来便有困难了。”

我们可以通过看到的内容来进行理解，但有的人还不能做到（例如：盲人），而计算机也不能，它没办法知道你是表示没有任何附加意义或是强调的。虽然我们做的东西是要给别人看的，但是，媒介是谁呢？还是计算机！如果计算机“看”不懂你的内容，那它最多就只能直接传达给别人。然而，我们更希望一份内容不仅仅只是能被复制，为了让文档有更多的利用，所以我们选择语义化。

虽然，语义化在很多情况下还不能达到我们期望的效果，但我们不能因为“别人”不理解就可以不要自己做得更好。

## 2.5 按语义来分类

在之前一直提到语义，但是，怎样来理解这些呢？在外国有人提出了语义分类

([http://westciv.com/style\\_master/house/good\\_oil/semantics/classification.html](http://westciv.com/style_master/house/good_oil/semantics/classification.html))，作者把HTML元素分成三种不同类型的语义单元，即结构语义（structural semantics）、内容语义（content semantics）、修饰形容语义（rhetorical semantics），以下是作者提出来的分类：

- 结构：
  - div
  - span
  - ol, ul, li, dl, dt, dd
  - del, ins
  - h1~h6
  - p
- 内容
  - a
  - abbr
  - acronym
  - address
  - blockquote
  - cite
  - code
  - dfn
  - kbd
  - q
  - samp
  - var
- 修饰形容
  - strong
  - em
- 不能确定的：当元素的内容不同有呈现不同的语义
  - blockquote
  - site
  - q

在这几个分类中，我们必没有看到(X)HTML的所有标签，像applet、basefont、center、dir、font、isindex、menu、s、strike、u这些早在1998年4月HTML4.0标准中就已经被W3C列为“不建议使用(deprecated)”标签自然不在讨论的范围。而我特别在意的br也没有出现，个人倾向把它分在结构里，虽然XHTML2.0会给line代替，而像sup、sub这类属于表现类的标签，在某些情况也有着特别的意义，比如O<sub>2</sub>、X<sup>123</sup>。

## 第四章 用XHTML和CSS来“摆”网页

网页是摆出来的，那怎样才能摆好呢？

通过前面的基础学习之后，在这一章，我们将开始学习具体的应用。在这里，我们完成一个虚拟的图片分享网站的部分页面，让你在练习中学习XHTML与CSS的基础应用。

### 4.1 开始第一个页面

首先，要做的是一个介绍的页面。想象一下我们需要什么呢？一个名字，一句口号，还有一段介绍性的文字。由于是第一个页面，我在这里给出完整一些的代码：

```
<!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" lang="zh-CH" xml:lang="zh-CH">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Welcome to Ao.A!</title>
<meta name="keywords" content="相片,相册,photo,photography" />
<meta name="description" content="Ao.A Album是个用于学习的在线图片管理与分享的小站" />
<style type="text/css">
...
</style>
<!--<script type="text/javascript" src="js/check.js"></script-->
</head>
<body>
<h1>Ao.A Album</h1>
<h2>Make your life beautiful and easy</h2>
<p>全球内最简陋的照片发布、存储、分享平台，通过它你可以学习标准化网页制作的一些知识，
虽然没办法涉及到关于Web标准所有的知识，但我会尽可能地把一些常用的应用拿来出讲。</p>
<p>...</p>
</body>
</html>
```

我们先来分析一下刚才这段代码，我把网页声明为XHTML1.0 strict 严格型的，为什么这么做呢？是因为我希望有一个比较好的开始，从一开始就严格约束你的代码。

`<html xmlns="http://www.w3.org/1999/xhtml" lang="zh-CH" xml:lang="zh-CH">`这行代码中xmlns就是定义命名空间，<http://www.w3.org/1999/xhtml> 是缺省的命名空间声明，而lang="zh-CH" xml:lang="zh-CH"声明了语言，在后续的章节里将会讨论声明语言产生的影响。

HTML只需要使用lang声明，XHTML1.0需要同时使用lang和xml:lang声明，而XHTML1.1和XML只需使用xml:lang，这里我们可以从语言声明猜想出XHTML是夹在HTML跟XML之间的。

从<head>里面可以看到，我并没有把<title>放在最开头，原因是，在一些浏览器（比如IE6），当没有打开编码自动选择时，使用utf-8作为文件编码可能会让网页显示一片空白。一个比较好的习惯是把声明编码的meta放在head的最开始处。在这里我们还是使用text/html的，原因是还有糟糕的浏览器不支持XHTML。为了不一开始涉及过多的问题，我也不使用<?xml version="1.0" encoding="utf-8" ?>之类的xml声明，暂时还是用XHTML化的HTML。

<meta>中name="keywords"和name="description"分别表示关键字与描述，这两个属性是服务于搜索引擎



擎的，meta还可以提供表示其他含义的属性，如name="author"用于表示网页的作者等。

样式表我选择了直接写在页面上，在例子中方便调用，在实际应用中，我经常是使用链入外部样式表。网页可能还需要脚本来帮助做一些事情，但现在还没有实际的功能，可以先注释起来，对于html里一些暂时不需要的内容，都可以使用注释的方式让其无效化。

<body>中才是页面上真正的内容，很简单，让我们在浏览器看一下它们被显示成什么样子（图4.1）：



图4.1

顺便让模拟Windows Mobile的Pocket PC 模拟器也露一下脸（图4.2）：



图4.2

都可以正常显示，那我们开始来学习用CSS控制它成为我们想要的样子。

## 4.2 文字的表现

在前面的一节中，我们已经做了一个最简单的网页。也许你会问：那是网页吗？我怎么看起那只是像一篇文章，没错，可它不但是一篇文章，也是一个网页。

网页的表现形式有很多，用图片、用FLASH等来表现都可以，但是它们都有一个共同要素——文字内容，不同的只是修饰的方式。

CSS是控制内容修饰方式的工具，下面我们就来看一些实际的例子：

### 4.2.1 字体的外型

文字的大小可以使用font-size来控制，如果要把h1的字体变得大一点，可以使用：

```
h1{  
    font-size:3em;  
}
```



图4.3

字体的除了正常的大小定义外，还有一些特殊性，《Bulletproof Web Design》（中文译本：《无懈可击的Web设计》）已经由清华大学出版社出版了，有兴趣对CSS有更深入的了解的朋友可以去看看。）这本CSS进阶的书都花了整整一章来讲它。在这里，我先使用大家习惯的px、em来控制字体大小，在后面第八章会详细讲到字体的特殊性。

font-family能够定义字体，每个浏览器的默认字体不一定都相同，Windows上的大多数浏览器的默认字体是Times New Roman，（中文版的Firefox的某些版本是用默认宋体）但也不排除有的用户自己已修改的。例如：

```
h1{  
    font-family: Verdana, Arial, Helvetica, sans-serif;  
}
```



图4.4

font-family后面跟的是字体名称，使用多种字体时，用半角逗号隔开，浏览器渲染时，先从系统查找第一个字体，见图4.4，例子中的显示的是Verdana，如果找不到就第二个，如果所有的字体都找不到就找最后所指定的同族字体，因此font-family的最后一个值并不是字体名称，而是同族字体族科名称，使用同族字体能保证你所指定的所有字体都不可用时可以有做最终代替，刚才的Times New Roman就是属于Serif。

下面列出常见的字体族科:

- Serif (如Times、Georgia、New Century Schoolbook等): 不等宽, 边角和笔画结尾处有衬线修饰;
- Sans-serif (如Helvetica、Geneva、Verdana、Arial、Univers等): 不等宽, 边角和笔画结尾没有修饰;
- Monospace (如Courier、Courier New、Andale Mono等): 等宽字体, 可以有修饰也可以没有;
- Cursive (如Zapf Chancery、Author、Comic Sans等): 花体字;
- Fantasy (如Western、Woodblock、Klingon等): 其他类

字体名称如果是中文或者中间带有空格时, 应该使用双引号或者单引号括起来, 虽然不括起在主流浏览器上也不会有什么问題, 但那与标准用法相违背。

如果字体名称中带有双引号或者单引号, 应该使用转义符转义。

看一下使用几种不同字体族科在IE6所呈现的效果(图4.5), 这里选择IE6做演示是因为我所使用的IE6并没更改过关于字体族科设置, 同时, 现在大多数用户使用的也是没更改过设置的IE6浏览器。

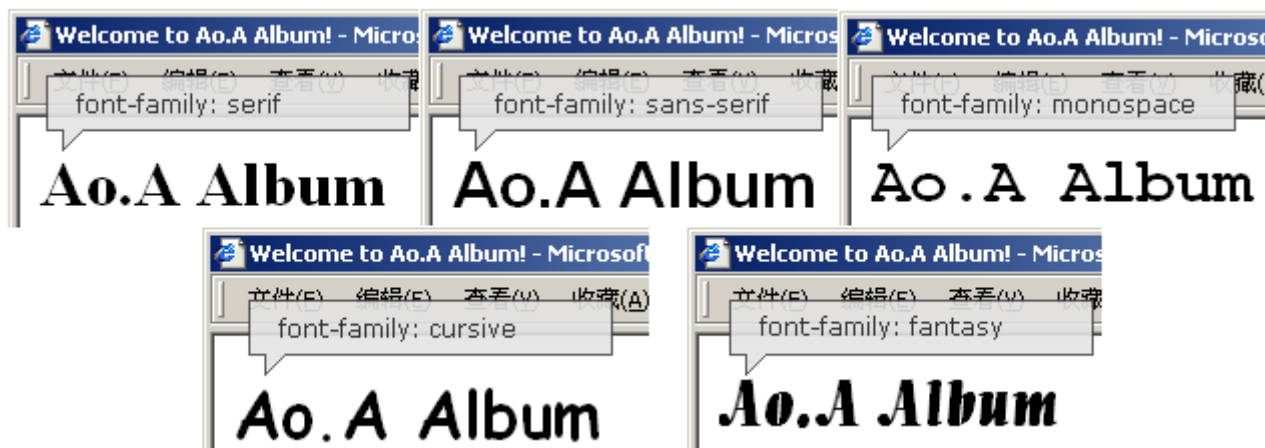


图4.5

这就是当你为网页选择的字体在用户的电脑不存在时的最终的呈现结果, 当然中文就没这样复杂, Windows上默认都是使用宋体。在接下来的例子里的字体使用上, 暂时使用现在比较流行的Verdana、Tahoma、Arial这几个来做演示, 等到在第八章文字的艺术跟字体大小一起讨论。

对于应用于同一类元素的多条规则, 可以合并在一个选择符里面, 就像这样:

```
h1{
    font-size:3em;
    font-family: Verdana, Arial, Helvetica, sans-serif;
}
```

而不是需要像这样写:

```
h1{
    font-size:3em;
}
h1{
    font-family: Verdana, Arial, Helvetica, sans-serif;
}
```

不知道你有没有注意到, <h1>、<h2>的字体看起来比较粗呢? 没错, 绝大部分浏览器默认已经把h系列和b、strong定义font-weight: bolder, 也就是CSS中用来控制字体粗细的, bolder 相对于父元素稍粗, 就是说当网页的字体是正常时, 定义成bolder就是粗体, font-weight还有其他一些属性值: lighter相对于父元素稍细、normal 普通字、bold 粗体字、数字(100,200,300,400,500,600,700,800,900)由小到大代表笔画由细到粗, 对于屏幕显示, 400=normal, 700=bold。

下面尝试一下把<h2>定义成正常的:

```
h2{
```

```
font-weight : normal;
}
```

效果如图4.6:

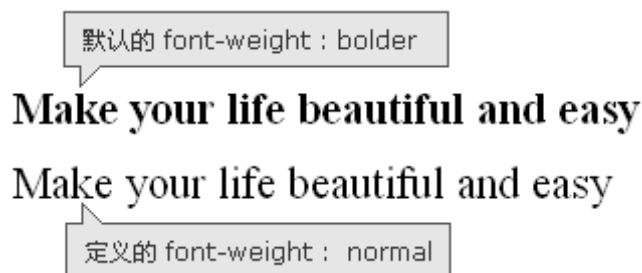


图4.6

在实际应用中，最常使用的也是normal和bold，有的朋友为了节省文件大小使用400和700，的确是省了一点点，而这对于流量大的门户网站来说是很重要的。除了加粗，还有一些属性能够控制字体的表现。

**font-style:** 设定字体样式，取值有normal: 普通字；italic: 斜体字（对于没有斜体变量的特殊字体，将应用oblique）；oblique: 倾斜的字体，例如：

```
h2{
    font-style : italic;
}
```

效果如图4.7:



图4.7

不幸的是，适合屏幕阅读的宋体却不适合斜体，虽然Vista的雅黑在打开ClearType（微软提供的一种技术，通过使屏幕字体的边缘平滑来增强显示，尤其适用于液晶显示设备。）的情况下显示得比较好，但现在大多数人的电脑里还没有这种字体（图4.8）。

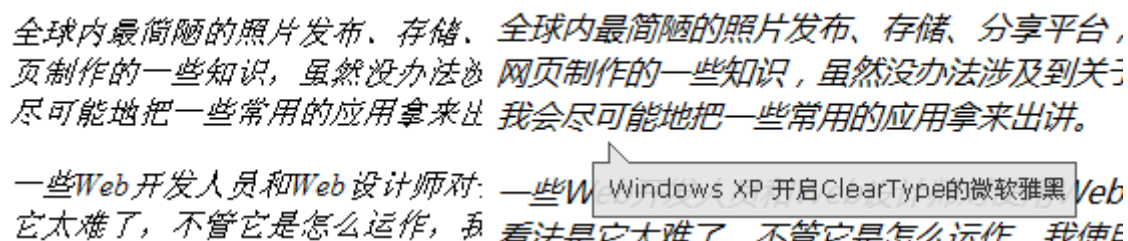


图4.8

**font-variant:** 设置对象中的文本是否为小型的大写字母，取值有normal: 正常的字体；small-caps: 小型的大写字母字体。例如：

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

效果如图4.9:



图4.9

当然，中文是没有大小写的区分了。

**line-height:** 细心的朋友应该看到<p>里面文字的行距跟前后的不一样，这里有几个因素影响到的，其中一个就是line-height，它是表示行与行基线之间的高度，取值有 *length*（本书中所有在CSS属性值中用斜体表示的length并不是直接把length当成属性值，而是代替长度值的一个写法。），百分比数字或由浮点数字和单位标识符组成的长度值，可以使用负值。

line-height有继承的问题，如果没有标明单位，像这样，

```
body{
    font-size:12px;
    line-height:1.5;
}
```

这样行高就是body字体的大小12px乘以1.5等于18px，而在body里面的p元素如果字体是24px，那它的行高就是24px乘以1.5等于48px，是使用自身字体大小乘以继承下来的比计算出来的高度。

一旦标明了单位，例如px、em或者百分比时：

```
body{
    font-size:12px;
    line-height:1.5em;
}
```

这样的话，body里面的元素就直接使用body的行高，也就是18px。而不是再按自身的字体大小重新计算，如图4.10，因为是在body中使用的，受影响是所有的元素，h1、h2的效果更明显。



图4.10

以上这些定义其实可以由一个复合属性一次性定义出来，而不用分成多个来定义，也就是常说的缩写，这个集合属性的写法如下：

```
font : [ [ <'font-style'> || <'font-variant'> || <'font-weight'> ]? <'font-size'> [/ <'line-height'> ]? <'font-family'> ]
| caption | icon | menu | message-box | small-caption | status-bar | inherit
```

比如我们可以这种：

```
body{
    font: normal normal normal 14px/1.2em Verdana, Arial, Helvetica, sans-serif;
}
```

这样就相当于：

```
body{
    font-style: normal;
    font-varian: normal;
    font-weight: normal;
    font-size: 14px;
    line-height: 1.2em;
    font-family: Verdana, Arial, Helvetica, sans-serif;
}
```

font在缩写的情况下，如果部分属性给省略掉时，例如：

```
h1{
    font: 12px/1em Verdana, Arial, Helvetica, sans-serif
}
```

效果如图4.11：



图4.11

这样的话，前面给省略掉的属性，会使用默认属性值：normal，属性值的顺序可以随意，浏览器能自动区分，但并不推荐你这样使用，因为也许还有糟糕的浏览器存在。

字体也可以直接使用系统字体，如下：

- 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。

这种方式更适合使用在软件中的网页，可以跟系统的风格保持一致，像这样h2{font: caption}。同时，你还可以使用强制继承的方式让子级继承父级元素的字体，如：a{font: inherit}，不过，由于字体是自动继承的，所以这个就不常使用，除非已经被定义过。

除了这个最常见的font还有一些常见的关于的字体和文本的属性，如下：

**text-align**：控制文本的对齐方式，取值有left：左对齐；right：右对齐；center：居中；justify：两端对齐，left是默认值。左右对齐和居中都是比较容易理解的，如图4.12所示：



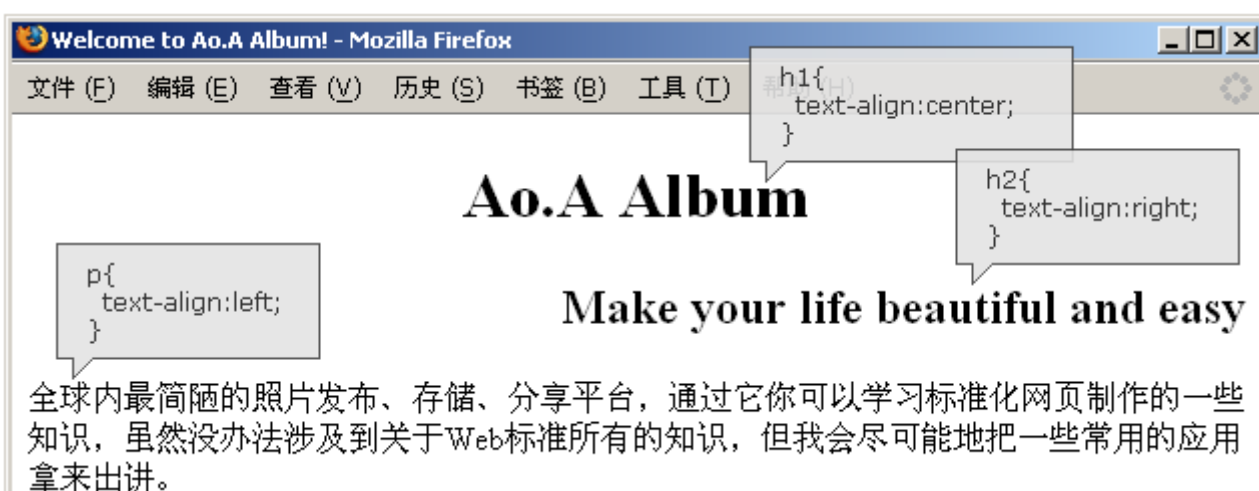


图4.12

h1是居中，h2是右对齐，p是左对齐。那两端对齐呢？再对比一下图4.13先。

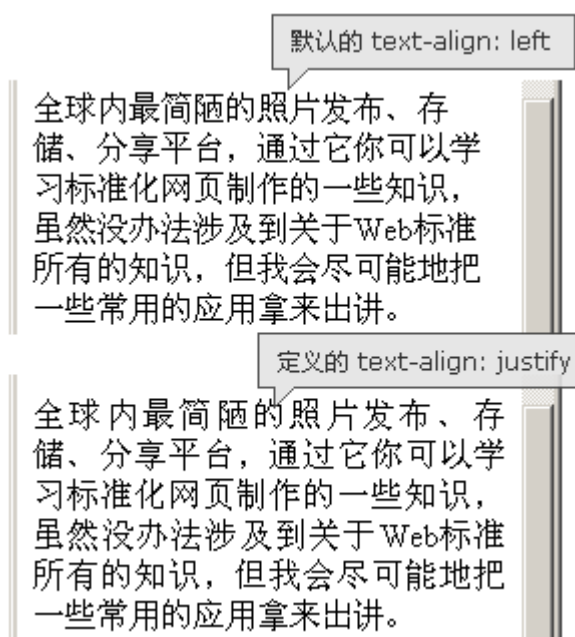


图4.13

图4.13可以看出上边部分，右侧也是对齐，它对文字的处理有点像Word一样，把原本存在于右边的空白分散到文字之间的空白，以达到排版的美观。注意，两端对齐的最后一行文字并不会把尾部的空白分散文字之间。

当然，文本的对齐并不仅仅使用在文本上，在后面我们会学习更多的应用，例如在某些情况下让整个网页居中。

**letter-spacing:** 控制对象中的文字之间的间隔，取值有normal：默认间隔；length：由浮点数字和单位标识符组成的长度值，指定的间隔添加到每个文字之后，但最后一个字将被排除在外，允许负值。

这个属性可以让我们想在两个字之间想增加间隔时不用敲空格的方式来解决，看一下例子：

```
h1,p{
    letter-spacing:10px;
}
```

效果如图4.14：



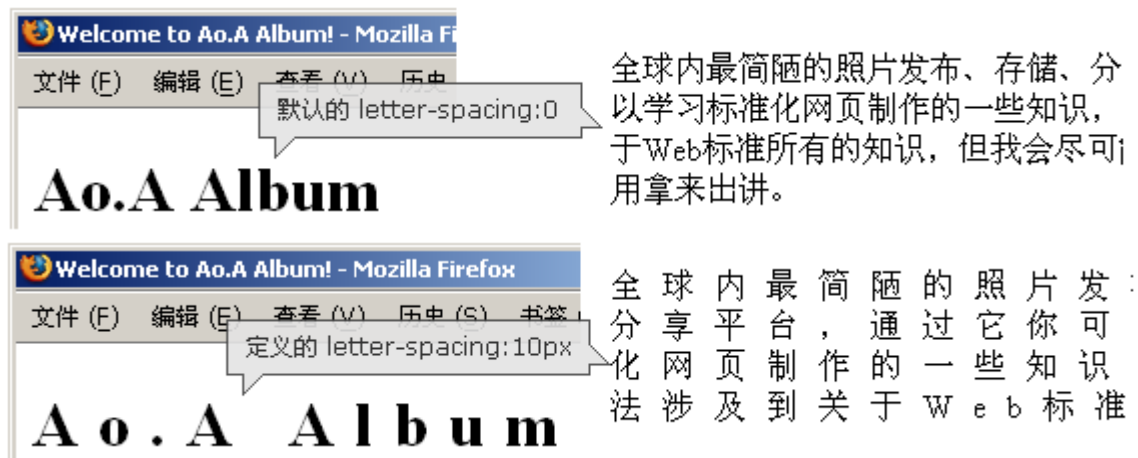


图4.14

**word-spacing:** 控制对象中的单词之间插入的间隔。取值有**normal**: 默认间隔; **length**: 由浮点数字和单位标识符组成的长度值，指定的间隔添加到每个文字之后，但最后一个字将被排除在外，允许负值，例如：

```
h2{
    word-spacing:0.5em;
}
```

效果如图4.15:



图4.15

**text-indent:** 设置对象中的文本的缩进，取值为长度单位，允许负值。这个属性可以让我们不用使用空格的方式来解决段落的缩进的问题，看一下例子：

```
p{
    text-indent:2em;
}
```

效果如图4.16:

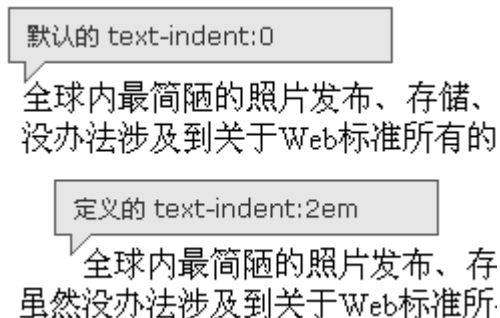


图4.16

对text-indent使用负值也成为以图代字的一个常用手段之一，在后面会介绍到。

**text-decoration:** 设置对象中的文本的装饰。最常见到的就是链接的下划线，例如我们可以这样：

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

效果如图4.17:

# Ao.A Album

图4.17

除了可以装饰下划线还可以使用**none**: 无装饰, 就像正常看到的文字一样; **blink**: 让文字闪烁起来, 不过IE不支持; **line-through**: 贯穿线, 相当于删除线的样子; **overline** : 上划线。

当然还有一些对于文本并不常用的属性, 我们会在后面介绍到。

## 4.2.2 颜色的定义

颜色的定义算是比较简单, 例如:

```
h1{  
    color: #93A0A6;  
}
```

效果如图4.18:



图4.18

在第三章已经介绍了颜色表示的多种写法, 本人更倾向使用#RRGGBB。由于在网页里样式表中颜色是会继承的, 我们并不需要为网页上的每个元素都定义相同的颜色, 例如:

```
body{  
    color : #93A0A6;  
}
```

效果如图4.19:



图4.19

网页上的元素都会自动继承body的颜色的定义，直到它们被重新定义新的颜色或者由浏览器重新定义（比如链接）。

### 4.2.3 背景的定义

背景的定义比文字复杂多了，先从颜色开始，例如：

```
p{
    color : #000;
    background-color : #93A0A6;
}
```

效果如图4.20：

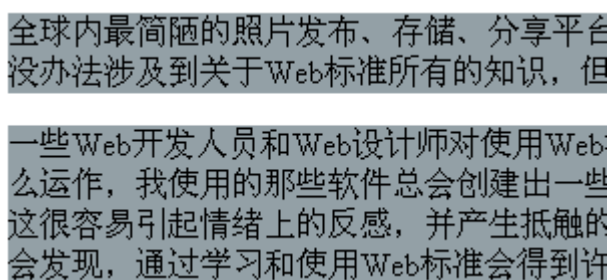


图4.20

背景色的默认值是transparent，也就是透明。当背景色已经不能满足我们的需求时，可以使用background-image再给背景加点图片做装饰，例如：

```
p{
    background-image:url(images/4.1.4.gif);
}
```

效果如图4.21

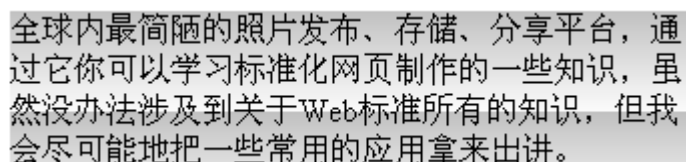


图4.21

其中使用的图片是图4.22。



图4.22

url()里可以是相对地址也可以是绝对地址，如果想对已经定义了的背景图片的元素不使用背景图片时，并不是去掉url括号里的地址，虽然空掉url括号里的地址也可以去掉背景图片，但这并不是正规的写法，应该是使用：“none”，这个也是该属性的默认值，例如：

```
.nobj{
    background-image:none;
}
```

在图4.21中可以看到，图片是以平铺的方式填充了整个元素，如果只想显示一次呢？这时可以使用另

一个相应的属性: **background-repeat**, 取值有:

- **repeat**: 背景图片在纵向和横向上平铺;
- **no-repeat**: 背景图片不平铺;
- **repeat-x**: 背景图片横向平铺;
- **repeat-y**: 背景图片纵向平铺。

例如想让图片横向平铺的话可以这样:

```
p{
    background-image:url(images/4.1.4.gif);
    background-repeat: repeat-x;
}
```

效果如图4.23:

全球内最简陋的照片发布、存储、分享平台，通过它你可以学习标准化网页制作的一些知识，虽然没办法涉及到关于Web标准所有的知识，但我会尽可能地把一些常用的应用拿来出讲。

图4.23

接着，我要**background-position**精准地控制图片的位置，**background-position**的属性值有两个参数，分别用于控制横坐标和纵坐标。默认值是0% 0%，也就是左上角，如果只指定一个值的话，这个值是用在横坐标，而纵坐标将默认为50%；比如：

```
p{
    background-image:url(images/4.1.4.gif);
    background-repeat: no-repeat;
    background-position:50% 50%;
}
```

除了百分比外，还可以使用长度单位，就像这样：**background-position:5px 10px**；。当然，还可以使用位置名称，如：

**background-position: right top**;

这样的话，它相当于：

**background-position: 100% 0%**;

坐标的值可以使用负值，例如：

**background-position: 0 -25px**;

效果如图4.24:

全球内最简陋的照片发布、存储、分享平台，通过它你可以学习标准化网页制作的一些知识，虽然没办法涉及到关于Web标准所有的知识，但我会尽可能地把一些常用的应用拿来出讲。

图4.24

我控制上一个例子的背景向上边上移了25px，这个就是负值的作用，不过没有办法向下边使用负值，如果在下边要显示图片的一部分，只能是把整个的高度减去要显示的高度得到的值做参数。

对于控制背景图片向右边偏移时，方式也是一样的，不过宽度的定义与高度的定义有着一定的区别，有时候，我们使用百分比来定义宽度，这样的话，宽度就可能随着浏览器的宽度变化着，如果要减去以px为单位的宽度就变得困难了，在CSS2中还没提供百分比减去固定宽度的方式。

尽量不要把属性值混合使用，例如**background-position:10px 40%**，在一些浏览器（如Opera8以下的版本）会当成无效处理，以默认值显示。

与背景图片相关的还有一个特别的属性：**background-attachment**，用于设置背景图像是随对象内容滚动还是固定在某一处。取值有：

- **scroll**：默认值，背景图片是随着对象内容滚动；
- **fixed**：背景图片固定。

背景图片固定在书中并不好演示，想象一下网站上常见两侧的浮动广告，你拉到滚动条了，它们依然停留在原地，这个属性是把图片放在背景上。

刚才我们已经知道字体有集合属性，那背景呢？它是这样的：

**background** : background-color || background-image || background-repeat || background-attachment || background-position

**background**跟**font**最大的区别是，**background**其中任何一个属性被单独定义到时，其他属性会自动以默认值定义，分别是：

```
background-color:transparent;
background-image none;
background-repeat:repeat
background-attachment:scroll;
background-position:0% 0%。
```

还有一个要注意的地方是：当同时定义了背景色跟背景图片时，背景图片是压在背景色上面的。

## 4.3 最简单的布局

知道了怎样控制文字后当然要学习怎样控制整体了。

### 4.3.1 Margin 与 Padding

从刚才的一些例子可以看出，网页里的内容并不是紧紧贴着浏览器的边缘显示，而是留有一点的空白，为什么这样呢？因为大部分浏览器默认body的外补丁并不是0，先看一下图4.25：

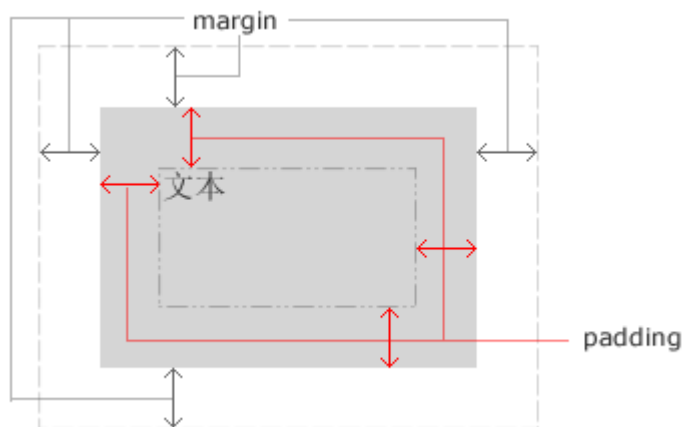


图4.25

**margin**：指的是元素块距离父层或者其他元素的距离，就是图4.25中灰色块距离外边的虚线框的距离。中文有很多叫法，如外补丁、补白、外间距、外边界等。

**padding**：指的是元素块内的元素距离元素的边界的距离，就是图4.25中灰色块的边框距离里面的内容的边缘（虚线框）的距离。中文又叫内补丁。

假如把例子中最外边的虚线边框当成浏览器的边时，**body**就是里面的灰块，**html**里的一些元素在浏览器中都有一些默认样式（详细可参考附录的HTML4默认样式）。

可能你经常会看到别人的样式表中最开头一行就是：

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

这样的做法就是把所有元素的内补丁跟外补丁都定义成0，当某个元素有需要时再单独定义。**margin**跟**padding**都算是复合属性，其属性就是元素上下左右四个方面的距离，它们都是可以单独定义其中一个方向的，例如：

```
p{
    margin-left:10px;
}
```

**margin**的四个方向是上：margin-top；右：margin-right；下：margin-bottom；左：margin-left。

**padding**的四个方向是上：padding-top；右：padding-right；下：padding-bottom；左：padding-left。

注意到没有，我介绍它们时的顺序是用顺时针的方向写的，**margin**和**padding**的取值不一定要四个。但如果取值是四个的话，就是按顺时针的方向即按上—右—下—左的顺序作用于四个方向，例如：

```
p{
    margin:10px 20px 30px 40px;
}
```

上面的例子表示距离顶部是10px，右边是20px、底部是30px、左边是40px；如果取值只一个，将用于全部的四个方向，相当于四个方向的值是一样的；如果取值是两个，第一个用于上和下，第二个用于左和右；如果取值是三个，第一个用于上，第二个用于左和右，第三个用于下。

注意一下**margin**跟**padding**取值不同的地方：**margin**允许负值，而**padding**是不允许的，如果**padding**的值是负值的话会当作无效处理。**margin**的取值还可以是auto，auto是表示自动的意思，那怎样自动法呢？在接下来的一节再揭开答案。

**margin**的表现也有跟**padding**不同的地方：普通的元素上下的**margin**会重叠在一起。

```
*{
    margin:0;
}
h1,h2,p{
    margin:10px;
    background-color:#CCC;
}
```

效果如图4.26：



图4.26

如果自己有写过CSS的朋友可能要问为什么margin有时不会叠加呢？这就是margin的特殊性，对于这个特性以及负值的叠加计算的方式，在后面第六章详细介绍。

### 4.3.2 单列固定宽度居中

为了方便控制文档，我决定在body里添加一个div把所有的元素套起来，因为在页面里可能还要使用很多div，为了让样式更容易控制，我给新添加的这个div加上了个id，叫做wrapper，至于为什么选择叫wrapper，只是业界的一种习惯，大家已经习惯把嵌套在最外面的层命名为wrapper或者wrap，当然你也可以使用自己喜欢的命名，不过命名也是有学问的，关于命名的问题会在后面介绍到。

添加了一个外层的div之后，原来的结构就变成了下面这样：

```
<body>
  <div id="wrapper" >
    <h1>...</h1>
    <h2>...</h2>
    <p>...</p>
    <p>...</p>
  </div>
</body>
```

我把宽度定成773px，这个数字可以说是经典数字，在以前，显示屏多数是以800×600的分辨率显示，在这里分辨率里的浏览器最大化时，除去滚动条占去的21px(这个值是IE滚动条的默认值)外，还剩下6px宽可以分配给两边，773的宽度恰好可以让内容不贴着浏览器边框显示，现在还有些门户的首页依然在使用这个宽度，试一下效果。

```
#wrapper{
  width:773px;
  background:#ccc;
}
```

**width:** 用于定义元素的宽度，可直接用于块元素，取值可以是：

- *length*: 由浮点数字和单位标识符组成的长度值或百分比；
- *auto*: 根据(X)HTML定位规则加载的文档流中分配。

效果如图4.27：

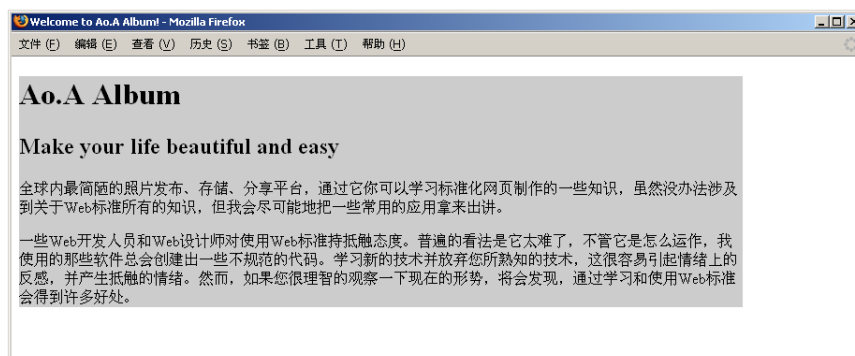


图4.27

细心的朋友可能看到#wrapper上边的间距不大对，这个也是margin的影响，暂时放一边，等后面一起讲解。

如果想让整个div居中的话，可以使用margin:auto来处理。

```
#wrapper{
  margin: 0 auto;
```



```
width:773px;
background:#ccc;
}
```

效果如图4.28:



图4.28

在元素总宽度已知的情况下，当margin-left和margin-right的值都是auto时，剩余的空间的宽度会平分到margin-left和margin-right上面，这样的话，在未知父元素的宽度时，也会自动平分剩余的空间的宽度达到居中的效果。当然，这是浏览器以Standards Mode渲染网页的情况下，那其他方式呢？

在如IE5、IE5.5这些浏览器和在Quirks mode的IE6+，并不支持元素本身使用margin:auto来居中，它们必须依赖于父元素的内容居中来获得元素的居中，就像这样：

```
body{
    text-align:center;
}
#wrapper{
    width:773px;
    background:#ccc;
    text-align:left;
}
```

因为在body使用text-align:center，body里面的内容也会跟着居中对齐，但我并不希望#wrapper里的内容也跟着居中对齐，所以把元素wrapper的内容的对齐方式定义为左对齐。这两种写法并不冲突，我们可以把它们并在一起写，以便兼容多种浏览器。

知道了margin:auto的作用后，我们再接着看一下width:auto，看看它又有什么不一样。

### 4.3.3 单列自适应宽度

现在不少显示屏都是使用1280px的分辨率，使用773px的宽度可能显得有点小了，这时，可以选择使用自适应宽度。

本来直接使用width:100%时，内容就会自动占据所有空间，但我还是期望两边可以留有空白，如果在width:100%的基础上直接再加上margin的宽度的话就会是这样：

```
#wrapper{
    width:100%;
    margin:0 25px;
    background:#ccc;
}
```

效果如图4.29:

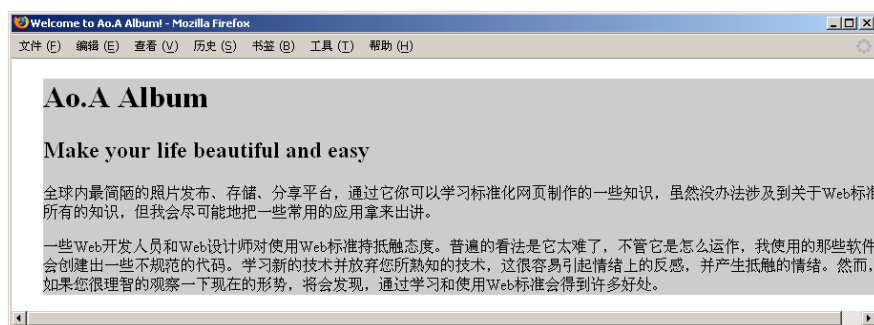


图4.29

看到图4.29没，浏览器的下方出现了滚动条，为什么呢？因为margin的值并不会算进width里，占据的空间大小是margin-left + width + margin-right，就像图4.30一样：

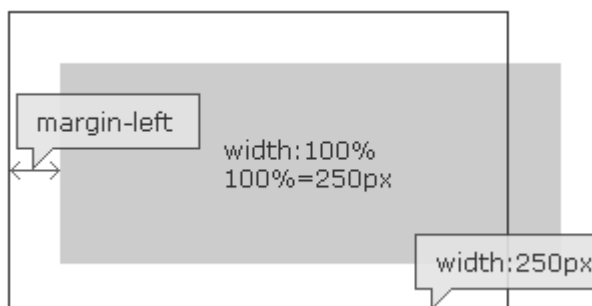


图4.30

那应该怎样，最简单的方式就是不要饲养魔鬼，不要定义宽度，回想一下最开头的没有样式的例子。在没在定义宽度的情况下，width的默认值是auto；

```
#wrapper{
    margin:0 25px;
    background:#ccc;
}
```

效果如图4.31：



图4.31

其实，这几小节的例子不用添加额外嵌套的<div>也可以达到我们现在需要到的效果，在显示的内容外面还套着个<body>，<body>外面还有着个套着<html>，它们一样可以使用样式来控制。那么是不是应该做一下没有最外面的<div>也能显示相同效果的练习呢？当成课后题大家自己练习一下吧。

### 4.3.4 奇怪的高度

高度跟宽度不一样，块元素没定义宽度时是自动填满空间，而没定义高度的元素是按需填充空间，当有多少人就占据多少的高度。如果刚才的例子做这样的测试：

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

效果如图4.32:

全球内最简陋的照片发布、存储、分享平台，通过它你可以学习标准化网页制作的一些知识，虽然没办法涉及到关于Web标准所有的知识，但我会尽可能地把一些常用的应用拿来出讲。

一些Web开发人员和Web设计师对使用Web标准持抵触态度。普遍的看法是它太难了，不管它是怎么运作，我使用的那些软件总会创建出一些不规范的代码。学习新的技术并放弃您所熟知的技术，这很容易引起情绪上的反感，并产生抵触的情绪。然而，如果您很理智的观察一下现在的形势，将会发现，通过学习和使用Web标准会得到许多好处。

图4.32

可以看到背景色只是填满p元素的文字占据的空间，如果定义宽度跟高度时，

```
p{
    width:200px;
    height:100px;
    background-color:#ccc;
}
```

在图4.33，文本没有横向流出元素p，而是在纵向溢出来了。但IE6和以下版本的IE内容并非纵向溢出，而是把元素撑高，还有一些特殊的文本横向溢出，特殊性的问题稍后讨论。

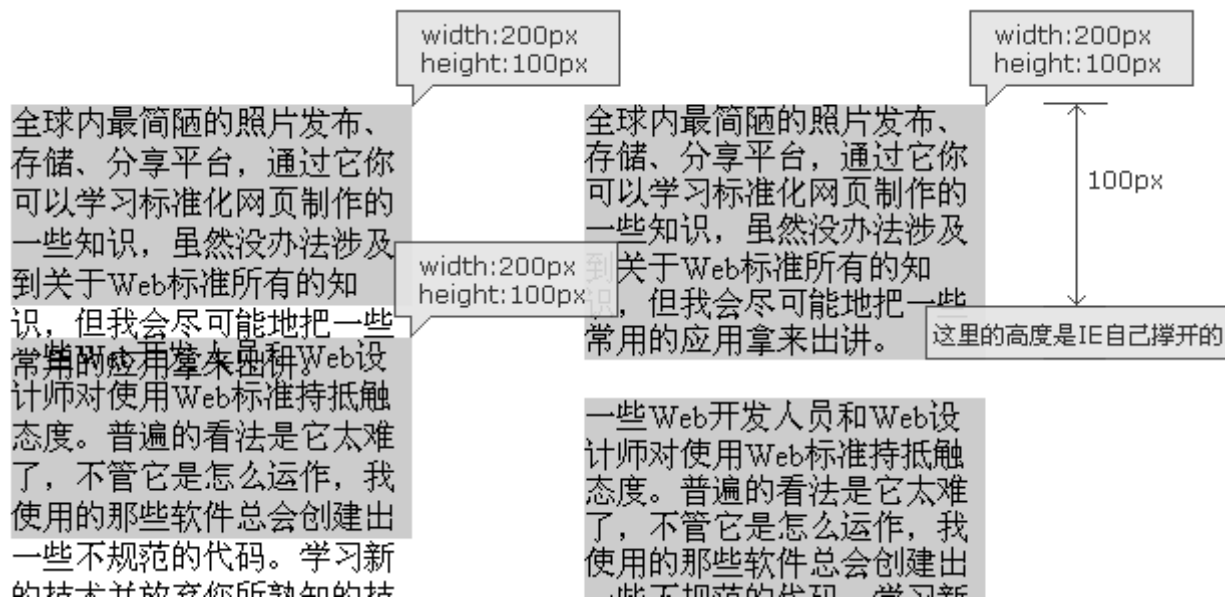


图4.33

我们可以把p理解成一个容器，当容器里的东西满时，东西就会溢出来，就像图4.34:

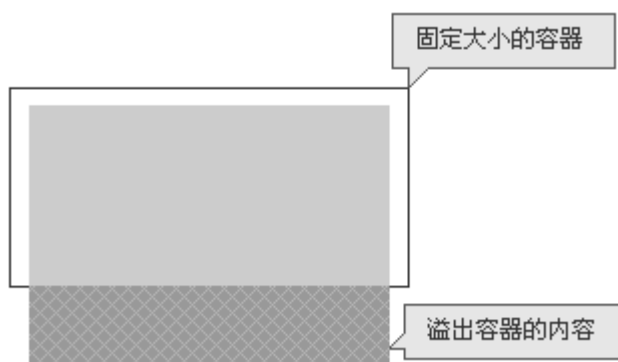


图4.34

### 4.3.5 边框的定义

边框除了正常的应用还可以用在一些特殊的场合，如布局之类的，现在先来学习边框基本应用吧。

**border**：这个也是一个复合属性，它是由**border-width**、**border-color**和**border-style**组成的。

```
h1{
    border: 1px #666 solid
}
```

效果如图4.35：



图4.35

其中1px是border-width的值，#666是border-color的值，solid是border-style。

**border-width**：边框的宽度。取值有medium：默认宽度，默认的宽度是由浏览器的默认定义决定的（比如Firefox和Opera的默认宽度是3px宽，而IE的默认宽度是4px）；thin：小于默认宽度；thick：大于默认宽度；**length**：由浮点数字和单位标识符组成的长度值，不可使用负值。

**border-width**也是使用多个参数值，像margin一样，顺时针方式分配，如果提供全部四个参数值，将按上-右-下-左的顺序作用于四个边框，如果取值只一个，将用于全部的四条边，相当于四条边的值是一样的。如果取值是两个，第一个用于上和下，第二个用于左和右。如果取值是三个，第一个用于上，第二个用于左和右，第三个用于下。

对于单个边框也可以使用**border-top-width**（上边框的宽度）、**border-right-width**（右边框的宽度）、**border-bottom-width**（下边框的宽度）、**border-left-width**（左边框的宽度）来定义，对于这几个指定的边框宽度只能有一个参数值，因为它们只是控制一边。

**border-style**：边框的风格。取值有none：无边框；hidden：隐藏边框；dotted：点线；dashed：虚线；solid：实线边框；double：双线边框；groove：3D凹槽；ridge：菱形边框；inset：3D凹边；outset：3D凸边。**border-style**的属性值也是可以多个参数值，分别作用于多个边，跟border-width的方式一样。

对于单个边框也有单独的属性：**border-top-style**（上边框的风格）、**border-right-style**（右边框的风格）、**border-bottom-style**（下边框的风格）、**border-left-style**（左边框的风格）。同样这几个单独指定一边的边框风格只能有一个参数值。

**border-color**：边框的颜色。取值有color：指定颜色，默认值是元素里面的颜色。**border-color**的属性值也是可以多个参数值，作用于四个边，跟border-width的方式一样。边框的颜色可以使用transparent（透明），

不过并非所有浏览器都支持，像IE6遇到border-color:transparent;时是会忽略掉，以默认值显示效果。

对于单个边框也有单独的属性：**border-top-color**（上边框的颜色）、**border-right-color**（右边框的颜色）、**border-bottom-color**（下边框的颜色）、**border-left-color**（左边框的颜色）。对于这几个指定的边框颜色只能有一个参数值。

border的属性其中一种被定义时，其他属性也会以默认值覆盖，分别是：

- border-width:0
- border-color:元素被定义的颜色
- border-style:none

但是也有特殊元素，如a里面的img，当a有href属性时，img的border的默认值是：

- border-width: medium
- border-color:元素被定义的颜色
- border-style: solid

直接使用border只能对四个边做统一的处理，并不能像border-width:4px 1px 2px 1px这样控制多个边，刚才讲的border:1px #ccc solid其实是这个的缩写。

```
border-width:1px 1px 1px 1px;
border-color:#ccc #ccc #ccc #ccc;
border-style:solid solid solid solid;
```

如果想单独指定其中一边的的话，可以使用**border-top**、**border-right**、**border-bottom**、**border-left**这四个可以用于指定某一边的样式。如：

```
h2{
    border-bottom:2px #333 solid;
}
```

效果如图4.36：

## Make your life beautiful and easy

图4.36

知道为什么下边框比文本长吗？因为<h2>默认是块元素，块元素默认是占据整行的，所以它的边框就会占据整个行。

有时，我们点击一个链接时，会发现，链接的四周有一圈虚线，它并不是边框，是贴在边框外面的，它的名字就叫**outline**，它也是一个复合属性，是由outline-color、outline-style、outline-width组成，其中，outline-style、outline-width与border-style、border-width的取值的参数是一样的，但outline-style、outline-width只能取一个值，因为outline是四个边表现是统一一样的，而不像border那种可以分别指定哪一边是怎样的，outline-color也一样，只能取一个值，同时作用于四个边，outline-color还有一个特别的值，就是invert，这个值是使用背景色的反色，那我们来试一下，

```
h1{
    border:1px #666 solid;
    outline:1px #ccc solid;
}
```

效果如图4.37：



图4.37

可惜的是，outline在IE里并不能通过CSS定义外观，只有在链接或表单控件获得焦点时才会出现。如果加上边框，那元素的宽高是怎样算的呢？接下来我们开始来了解盒模型。

## 4.4 郁闷的盒模型

为什么说郁闷的呢？把4.3.1 Margin 与 Padding那一节的图整合进border的话，就成了下面这张新的图（图4.38），这就是大家常说的盒模型：

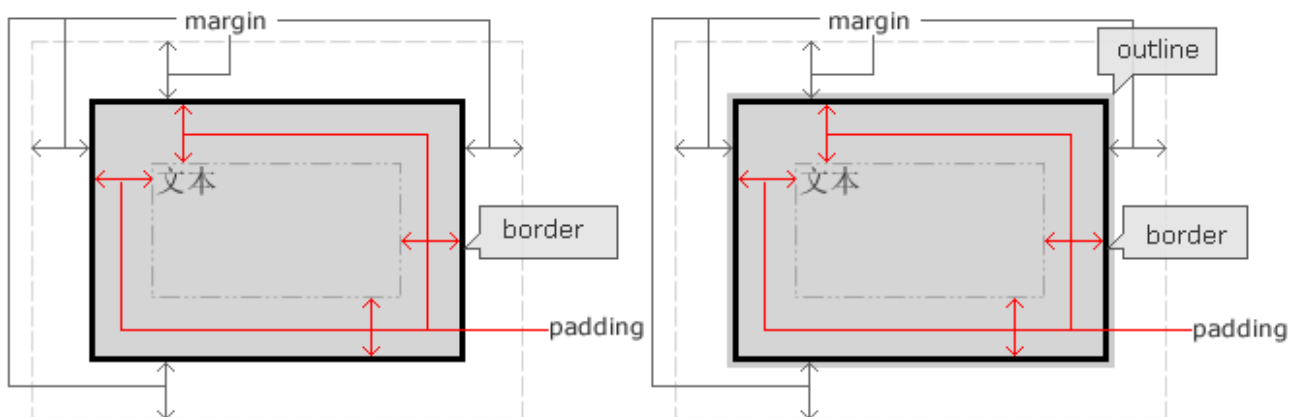


图4.38

灰色的区块里面虚线边框的宽高就是元素的宽高，正常的情况下，border也没有算进width和height里，我并不喜欢这种计算方式，在很多时候这种方式使我的工作复杂化，我必须额外添加一个嵌套的元素来解决问题，而outline是个特别的属性，它是漂在元素四周，不占据空间。还是从浏览器的表现中来理解盒模型吧：

```
<div class="box">
  <div class="inner">传说中的文字</div>
</div>
*{
  margin:0;
  padding:0;
}
.box{
  margin:20px;
  padding:20px;
  border:10px #666 solid;
  width:100px;
}
.inner{
  background-color:#CCC;
}
```

效果如图4.39：

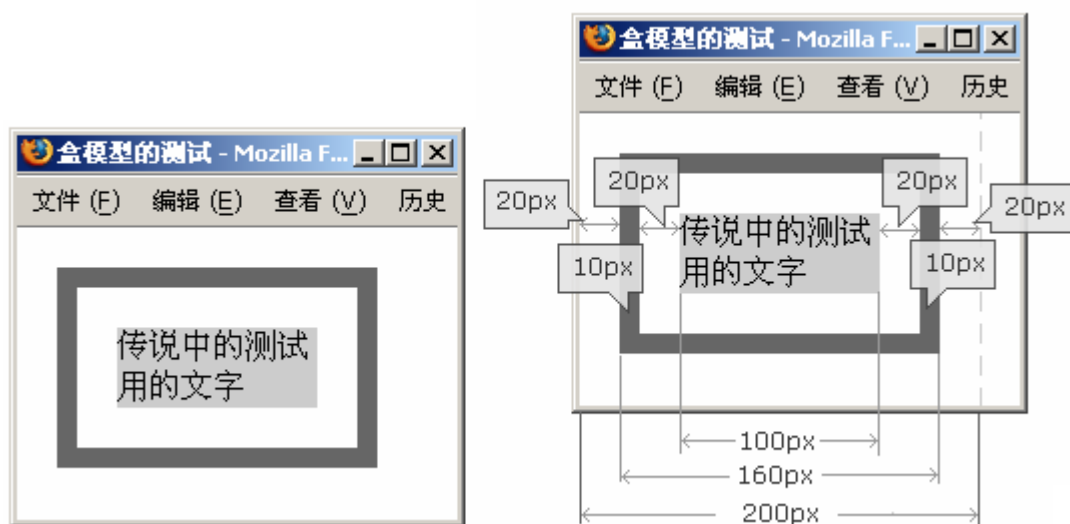


图4.39

来分析一下，为了更容易看清里面的内容，我在里面的层加了背景色，图中最右边的虚线是我添加的，只是为了看清实际的距离。

我定义的外层（.box）的宽度是100px，而它的占用空间的宽度，也就是可视宽度是：

$\text{border-left-width}(10\text{px}) + \text{padding-left}(20\text{px}) + \text{box}(100\text{px}) + \text{padding-right}(20\text{px}) + \text{border-right-width}(10\text{px}) = 160\text{px}$

这里只在图4.39上画出宽度方面的示意图，高度也是这样算。而个人更喜欢IE6+ 在Quirks Mode下和IE5.x/win的这种盒模型（图4.40）：

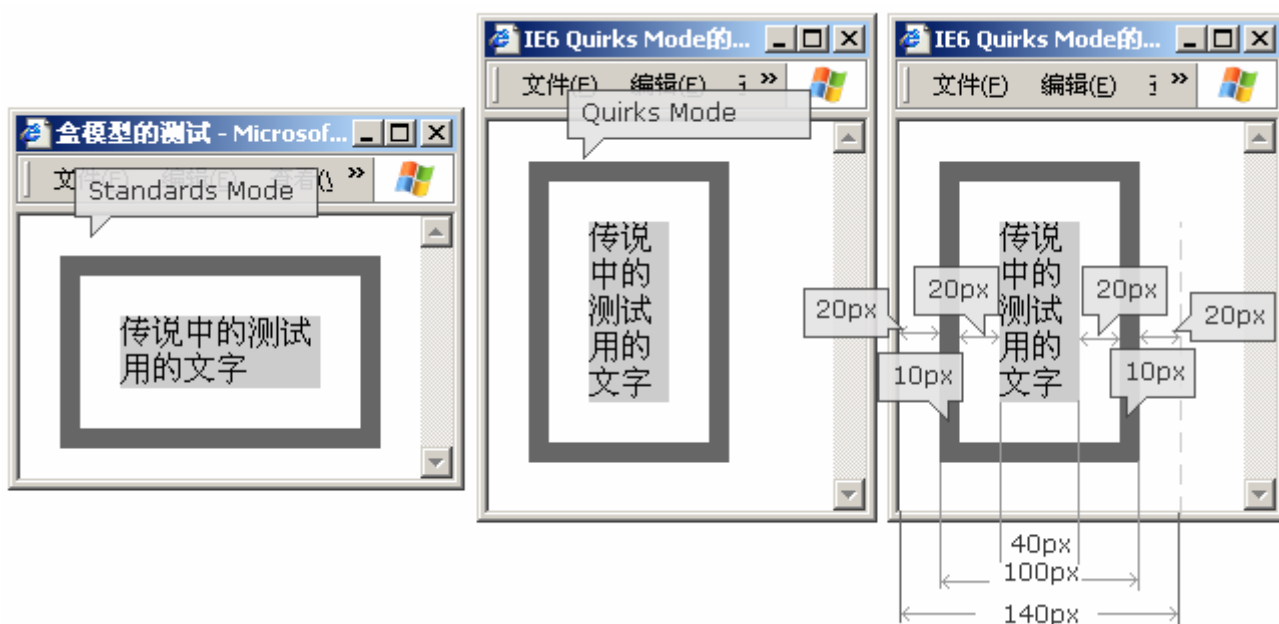


图4.40

IE5.x/win的表现同IE6在Quirks Mode下表现一样，不过IE5.x/win并没有Standards Mode，IE在Quirks Mode下的算法与Standards Mode不一样，.box定义了宽100px，占用空间的宽度就是的100px，padding和border都算在里面。

浏览器兼容的问题让我们选择了W3C推荐的标准，这是兼容的唯一选择，以后出现的浏览器或者现有的浏览器升级都会向W3C靠拢，IE7就是其中的一个例子。

在我从事的工作中，一些项目，是在应用程序中使用Web，因为只是调用IE内核，相当于网页只有IE的显示效果，我可以不考虑兼容的问题，可以选择使用Quirks Mode来处理，或者说IE6在Quirks Mode下的CSS能更好的处理问题，而且IE7在Standards Mode下无法去掉html的边框，无法达到预期要显示的效



果，也只能选择在软件里使用Quirks Mode处理网页。

如果你所制作的网页还需要考虑到IE5.x的浏览器，那也必须了解第二种盒模型。如果你想用CSS在IE6里完成一些特别的布局，你也必须学习第二种盒模型，在特殊的情况下，把某些浏览器引入Quirks Mode可以简单的解决一些问题。但是这不是你现在要学习的，先掌握好第一种盒模型的应用，它已经可以完成大部分效果。

据说CSS3的box model中已经有提议可以让用户选择width是内容宽度还是可视宽度，而在Firefox的私有属性中，有-moz-box-sizing的可以选择border与padding是否要计算在width和height里面，未来总是美好的，可现实总是残酷的。

## 第十三章 亲和力

- 他们可能无法容易地，或者根本不能看见，听到，拖动或处理某些类型的信息。
- 他们可能在阅读或理解原文上有困难。
- 他们可能没有或者无法使用键盘或鼠标。
- 他们可能只有纯文本的屏幕，小屏幕，或低速网络连接。
- 他们可能并不会说或不能通畅理解文档所用的自然语言。
- 他们可能眼睛，耳朵或手正忙碌或受某些事物的干扰（比如在开车上班途中，在一个喧闹的环境下工作，等等）。
- 他们可能使用早期的浏览器，完全不同的浏览器，语音浏览器，或不同的操作系统。

——摘自Web内容可访问性指南 1.0（吴隽辰 译）

### 13.1 概念

“Accessibility”这个单词中文有几个选择的意思：可访问性、无障碍、亲和力，那为什么我也选择用亲和力来描述呢？当一个网页或者网站有一定的可访问性时，它就可能达到无障碍访问。反过来，把网站做到各种各样的人在各种各样的环境下都可以无障碍摄取网站的资源时，就具有很强的可访问性。可访问性、无障碍这两个词用起来都是相对比较量化的，那亲和力呢？亲和力比较早是出现在化学上的，在近些年在也常用在人的身上，简单的意思就是个人的形体上所具备有一种力量能让周围的人感觉你很和蔼可亲，不受到其他因素影响所露出一一种情感力量！

网站面对各种各样的访问者，甚至机器，用相对比较无量化概念的“亲和力”来描述会更适合。当然，我们也不用执着于一个单词的字面意思，因为我们要面向更多层面的东西。

Web Accessibility通常定义为：Web 对于每个人来说都是具有亲和力的，无论残疾与否。

还记得我在第一章提到的电脑残疾吗？我们不应该单单理解成亲和力（或者可访问性、障碍）定义是针对残疾朋友的，而是在各种各样的情况下的我们。大约了解了Accessibility这个词之后就要学会应该做什么？在开始之前先来看一下非技术层面的东西。

当你打开一个网站时，自动给你多弹出几个窗口，当你关闭一个浏览器时又给你弹出个窗口，又或者当你要找一篇文章到，从首页开始到找到那篇文章时已经额外多打开N个窗口或者N个Tab，因为每打开一个页面都会弹开一个新窗口（Tab）来显示内容。换个更形象的例子，例如我去大公司找个朋友，但我不知道他详细位置，便在一层的前台询问时，前台跟我说：“在12层！”，到了12层时我才发现，12层原来很大，划分成很多区块，接着我又开始询问身边的人，“好像是在那边”，到了那边又来一个“好像那里”，到了那里，别人又跟我说：“那边16号，门口还贴着只蜘蛛侠的那间就是”，终于找到了。

假如，在一层的前台的人直接跟我指明在12层16号那间，我是不是可以减少一些询问呢？当然，这是个虚拟的例子，只是想让你感受一下假如发生这种事情后的心情。

回到刚才额外弹出很多窗口的例子，这种情况并不少见，用户选择是在新窗口打开还是在本窗口打开的权力给剥夺了，他为了打开一个链接不得不经历N个PV和N个链接。当然，并非所有的网站都是这样的，如果一个网站变成了这样，你还会不会说这个网站很有亲和力呢？

接着渗点技术性的东西来讲，在刚才找人的例子中，假如我有那个朋友的电话号码，我就会打电话让他下来接我，如果我忘记他的电话号码，那就是我自己的事，但是如果是他不把他的电话号码给我，就可能是他的问题了。如同人与人的关系一样，网站也存在有一样的问题。现在一个网页的站越来越多，所谓的一个网页的站并非整个网站只有一个网页的内容，而是指运用Flash或者AJAX等技术动态加载内容，它们有一个共同的特点就是只有一个网址。当朋友跟我说某某站有份内容不错，因为网站应用了Flash做的，还是动态加载内容的，只有一个网址，让我自己进去后再在某某栏目下找。这种情况是不是跟刚才举的找人的例子有点相同的地方呢？

有些东西是可以通过技术去解决问题，例如你看看2advanced，看看抓虾，人家也是用Flash动态加载，人家也是用AJAX动态加载，能用技术解决的问题就不叫技术问题了，只是在意你有没有去做跟会不会做。

提到的这两个例子只是在亲和力范围很小很小的一部分，我把它们拿出来讲是想让你理解一下，接下来要详细点介绍一些网页与亲和力相关的。

## 13.2 Web内容可访问性指南

Web Content Accessibility Guidelines吴隽辰把里面的Accessibility译成可访问性，隽辰自己说：Web内容的可访问性（Accessibility），主要针对有听觉、视觉、认知、学习障碍的人士来说的，也可以按照建筑学上的理解，把可访问性（Accessibility）理解为“无障碍”。为什么不翻译为“亲和力”？我觉得对于W3的推荐标准，用“亲和力”这个词汇，可能不能体现其权威性。

由于已经有比较完整的译本，我也没必须在这里复制粘贴内容。有兴趣的朋友可以去看一下中文版的WCAG1.0的内容：<http://www.junchenwu.com/WAI/wai-pageauth.html>。随着时代的变化，WCAG2.0（<http://www.w3.org/TR/WCAG20/>）也出台了，但是大方向相同，而且都保持一个特点，写得不是非常明确，因为很多东西并非量化就可以做得好的，就算你完全按着指南对网站进行修改，仍然可能无法达到完全的可访问性与无障碍，可能这一章带给你的又是一些思维性转变的内容，而不是什么技术上的知识。

在接下来的几节里我选择一些比较常见的问题来介绍，但是介绍的内容可能距离Web内容可访问性指南所提到的还很远，一步直接跨进到无障碍，不容易让人接受。包括我本人在开始学习无障碍的时候，阅读了多次《在30天内打造更具亲和力的网站》还未能感受到真正问题的存在。——很多问题只有身处其境时才能感受到。

### 13.2.1 并非所有的内容都是显示的

随着带宽、技术、用户需求的提高，图片、Flash、音频、视频等的东西的应用是越来越多了，但并非所有人的浏览器和设备都支持这些或者都能按设计师的想法显示这些内容。这里并不仅仅是指障碍的人士，也包括障碍的设备、网络。

打个比方，我在公司时，电脑一直是处于无声的状态，并非我的电脑没声卡，而是由于工作的氛围，那我的电脑是不是属于障碍的设备呢？当我访问某些网站时，由于跨域、服务器等原因，10M接入的带宽有时还达不到拨号上网的速度，那我的网络算什么呢？

图片，由于可以从视觉上给人更多更形象的表达，应用得越来越广了，但当遭遇网络问题时，代价也是惨重的。为什么一直提倡表现与结构分离呢？因为可以从某种意义上避免这些问题。当装饰类的图片处理成背景，即使遭遇各种原因导致图片无法显示，也是不会出现让你看到一块块显示不了的区块，而且在就算是使用屏幕阅读器，中低端的手持设备浏览器也可以得到比较好的表现，在“8.1 消失了文字 看到了彩虹”那一节里已经介绍了几种方式，再回想一下。

但是，并非所有的图片都是属性装饰类的图片，不然，像Flickr之类的图片分享类的网站不是整个都没意义

了吗？有些图片是属于内容的一部分，这时就不能简单的把图片当成背景处理了，但是出现图片显示不了的情况应该怎么办呢？

img元素里还有这三个属性：alt、title、longdesc，前两个大家就比较熟悉了，不过因为大部份浏览器都不支持longdesc也就没什么人用了，先来看下alt属性，它是用于为显示不了图片，对象等元素的浏览器指定替换文字，例如：

```
<p> </p>
```

如果这张图片能正常显示的话，有没有alt看上去都是一样的，但是因网络原因或者是浏览器的问题，例如文本浏览器，可能就变成图13.1这样了。

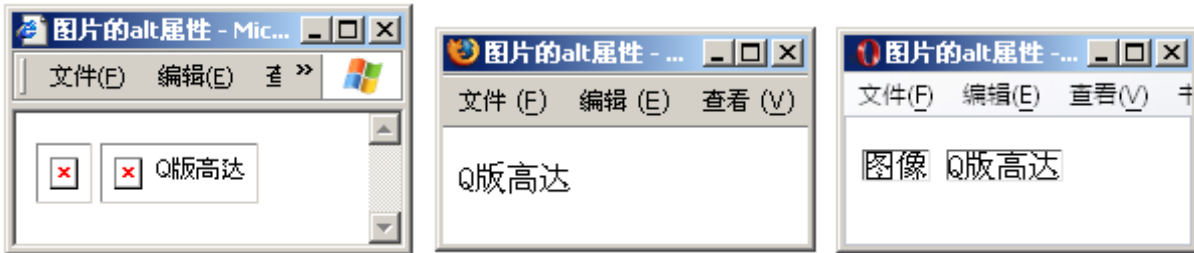


图13.1

这个就是主浏览器对图片的丢失的处理，这时你感觉到什么呢，那我们再深入一下，假如网页是用读的，而不是用看的会怎样，我先把内容改成这样（图13.2）：

```
<p>1  2  3</p>
```



图13.2

使用Opera的语音功能把它读出来。你就会听到：“one two Q \*\*\*\* three”，Opera还不会读中文，当然，盲人也不会使用Opera来读网页，他们自己在使用的可阅读的浏览器。

alt这个属性在正常情况下是没什么特别的作用的，但是在某些情况下却可以发挥很大的作用。又例如：

```
<input type="image" src="image/submit.gif" alt="注册" />
```

```
<input type="image" src="image/sumbit.gif" />
```

有的朋友会使用input的类型是image来做，当网络原因也可能会出现图13.3这样的情况。

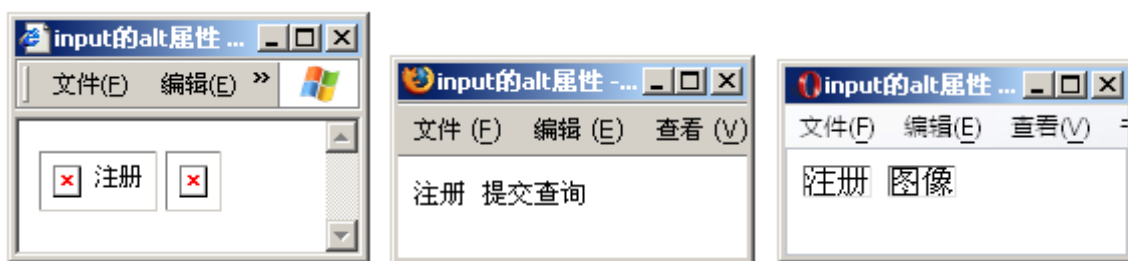


图13.3

当然，我们使用input时更多的时候是用背景图片来处理，这样就算背景图片因为网络原因没加载不能显示也只是影响美观。

在IE浏览器里，把鼠标移上图片有带alt属性的图片上会出现黄色提示（图13.4），但这并不是alt本身的作用，提示应该由title属性去完成，alt属性是替换文本，是用简单的话来描述图片，而title是用于提供额外说明信息，两者并不等同，不能因为看到IE用alt移上去也有提示，用title移上去也有提示就认为两者一样，在其他的浏览器单单使用alt就没有提示的效果，而且在IE里，alt和title同时使用时，是显示title里的内容的。



图13.4


title的应用最多的地方是用是a元素身上，很多人在使用title时总是直接复制文本。例如：

```
<a href="#" title="博客">博客</a>
```

```
<a href="#" title="学完了xhtml+css之后要学什么呢">学完了xhtml+css之后要学什么呢</a>
```

当鼠标移上去时，出现的提示跟链接的文本是一样的，那有什么意义呢？如果是说像这样：

```
<a href="#" title="用xslt+css让RSS显示的跟网页一样漂亮">用xslt+css让RSS显示的跟网...</a>
```

因为设计上的某些需要，原来的链接里的文字过长通过程序截断的还有点意义。可是，title属性还可以做更多有意义的事，例如链接添加描述性文字，加上日期等等。特别是当连接本身并不是十分清楚的表达了链接的目的，例如某网站表示出处的来源是用  这样来表示，如果鼠标移上去没有任何提示，你会随便打开这个链接吗？当然，也有某些网站处于商业利益，把整个站的所有title都去掉，对他们而言，用title就是在消费额外的金钱。

对于alt和title应该写上什么，我并没办法给出非常明确的方案，不过给的建议就是：alt应该写出精要，想象一下如果别人看不到图片时，要用文字什么来代替，把整个图片的情况描述出来吗？那可是longdesc要做的事，同时，并非所有的图片的alt都应该有内容，如果你感觉图片在无法显示的情况下对用户的意义而言可有可无的情况下，你甚至可以用alt=""来处理，当然，装饰类的图片我更倾向于放在背景里。而title里，如果是链接的话，应该更倾向于表达这个链接要做什么，补充一下链接里文本原来没有的信息。如果是用在派生词缩写的abbr身上的话，你可以把缩写要表达的意义描述清楚就行。

顺便说一下，在有些浏览器里title的长度如果过行，就会截断，在后面补上“...”，例如Mozilla核心的浏览器只能显示最先的64个字符（我是使用Firefox2测试）。当然也有朋友通过JavaScript来扩展发送链接的提示，在后面会介绍到。

如果直接需要对图片做详细的描述的话，应该是使用longdesc，longdesc属性可以用来提供链接到一个



包含图片文字描述的单页，但是由于普遍浏览器对longdesc支持不好（我现在知道的只有在Firefox上安装一个叫longdesc的插件才被支持外，还没看到其他浏览器支持），W3C曾经推荐使用描述链接，就是在图片后面再使用一个链接，内容是[D]，就像这样：

`<a href="longdesc.html" title="关于某某某的详细描述">[D]</a>`

不管是使用longdesc还是一个描述链接，都意味着需要一个新的网页来存放。如果描述的内容不是很长，我更建议直接存放在图片附近，[D]几乎没人会注意它，甚至有人怀疑它存在的意义。当然，这个取决于你的站的用户群，如果你认为真的有需要做到类如用ASCII来拼出这样的图案的话，就大胆的用(图13.5)。

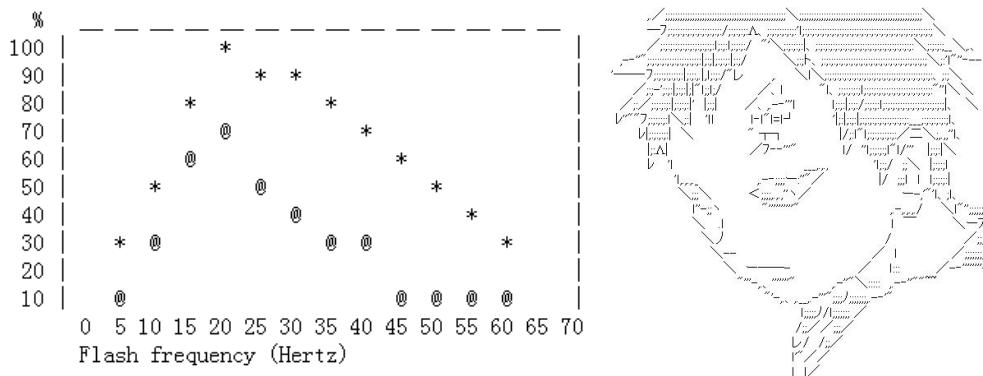


图13.5

不要因为别人说没什么意义，就不要去做，只有身处其境才会明白需求，视力正常的人很难明白失明者的感受，当然，ASCII来拼出的图案绝对不适合盲人。

除了图片，像Flash，音乐之类的东西并非适合所有人，不知道你有没有遇到过，打开了个网页，突然音乐响起，你的感觉是怎样的呢？可能音乐刚好跟这个网页配，让你感觉很舒服，也可能你自己的电脑也开着音乐，两种混在一起让你感觉很郁闷。如果是我一般会选择先关掉网页，为什么呢？假如你在跟一个人说话，他一边跳舞一边跟你说话，你会感觉怎样呢？当然，如果你是专门去看他跳舞的又是另外一回事了。并非说因为要做到所谓的亲和力，就连音乐之类的都不能用，如果可以，请给用户选择的权力，请别自动播放，如果要播放，请给用户有关闭的提示，而不是让用户用关闭网页的方式来关闭音乐。如果要自动播放，请在打开之前让用户做好准备，例如在前一个链接写上“打开直接播放某某某”之类的，而不是“某某某地址”之类的。

刚才讲的这些都是能看到、能听到的内容，假如因为机器原因看不到，听不到呢？不知道你有没有遇到过，打开一个网页，突然卡死，等了半天，接着弹出一个对话框，让你安装某某插件，只因为你的机子上没有要浏览这个网页的所需的插件。当然，现在的先进的浏览器已经能帮你挡下这样，但是网页却出现了一些空白，你猜不出那里本来应该有什么？在Macromedia（已经被Adobe公司收购了）的主页上，如果用户没有安装Flash的浏览器插件，网页是以图片，文本来显示，如果有安装的话，则用Flash更好的来完成这些功能跟效果，因为它通过JavaScript来判断，配合服务端输出内容。不会因为说用户没有安装Flash插件，就不让你看内容，而是通过另一种方式显示内容。

当然，现在Flash的插件跟浏览器的比例可以说是接近1:1，那其他媒体呢？Flash只是个例子。

回到看不到内容的部分，还是先看看Flash的问题，Flash在浏览器的插件还分很多个版本，虽然绝大多数朋友都是使用上最新版本的，但还是有一些朋友使用的低版本的，例如使用Flash内部代码可以实现真正全屏的功能，但是需要Flash插件在9.0.18.60以上的版本。这时候如果你是会怎样做呢？让用户升级，还是换种方式配合JavaScript只是全屏到浏览器的内部的空间来避免直接出现升级的情况呢？据说有的网站的Flash都是导出5.0.x的版本，想一下为什么要这样处理。

接着再深入一点，换到手持设备，除了智能手机能支持弱功能的Flash（Flash Lite）外，大部分可以上网的手机都不支持Flash，可能你的网站并不考虑手持设备，就算考虑，也是可能是输出另一个版本来显示。为什么要提这个呢？因为现在Flash是属于控件，在某次系统自动更新后，IE对控件都有做一个激活的处理，会在Flash的边框上加上一个虚线边，需要点一下激活Flash这个控件后才可以做更多的操作。后来，有人研究出用JavaScript来解决这个问题，可是，并非每个人写的判断都是那样完美的，我常常看到网页的某个区

域空了一片，就写着几个字，“不支持Flash”或者“请升级你的Flash版本”等，其实，我浏览器的Flash版本已经是最新的。

刚才提到手持设备使用另一个版本来代替是一个方案，因为用户使用的东西是不可预计的，当因为网站或者用户设备引起问题时，不应该只是像赌气地回答一样：“不会！不会就是不会”。假如能判断到用户不支持某种媒体时，还能输出“不支持某某某”时，为什么不能输出成一个链接加上一张图片呢？这样不管是对用户还是网站都是一个比较好的作法。

接着再深入一点，不支持JavaScript或者没有打开JavaScript的浏览器。我的Firefox装有一个noScript的插件，默认打开网页是不支持JavaScript的，并不是因为我怕打开JavaScript会有危险，而是我经常访问国外的网站，禁止JavaScript会快很多，国内的一些网站也是，并非是我接入的网络太慢，而是访问跨网络的网站都存在的问题。

禁止JavaScript，很多网站就出现了问题，但也有不少依然可以正常使用，例如淘宝、Gmail，像Gmail这种典型的全站AJAX应用的在禁止JavaScript后发生什么事呢？它是使用另一套使用界面，可以在无有JavaScript的情况下完成各种基本的操作。我曾经拿Gmail与朋友讨论它，这是禁止JavaScript打开Gmail会出现的界面（图13.6）。

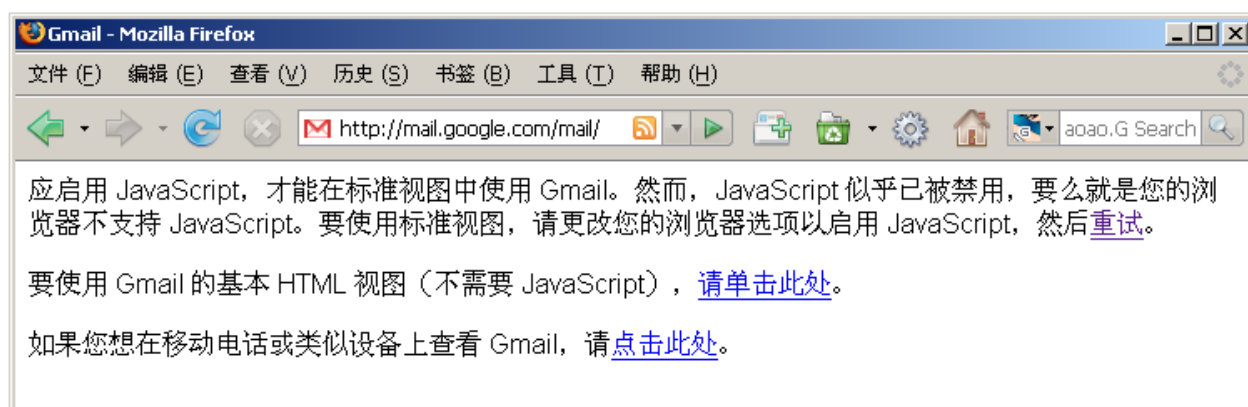


图13.6

主流的浏览器基本没有不支持JavaScript的，先抛开JavaScript被禁止的情况，但是其他的（移动电话或类似设备）呢？除了主流浏览器，大家并不是很关心它们。

如果一个网站的版本不适合在某些设备，可以设计另外的版本来补充，而不是拒绝。个人并不认同一个版本能在各种设备都可以正常显示，有些网页是可以，但有些网页并不适合。还是先回到主流浏览器上，当网页的内容以非文本的情况出现在HTML时，浏览器都可能无法正常显示你原来想要输出的内容，不管是图片、Flash，或者视频、音频，他们都有可能由某些原因显示不了，这时我们可以适合给予补充的信息。

## 13.2.2 保持原有功能的有效性

AJAX的出现让JavaScript的应用变得越来越广，恰到好处的使用会给用户带来很多方便，例如登录，以前的作法就是点击了带到一个新的页面，登录完后又自动跳回刚才的页面。但是现在大家更倾向于在当前页弹出一个层，直接在上面出现登录的表单，可以直接在上面登录，并通过JavaScript在底层去提交表单验证身份，然后对当前的页面再做细节的处理，以达到无刷新页面，减少页面跳转空白的等待，提高用户体验，当然我个人也比较喜欢这种做法。

但是有些朋友在细节上处理得并不是很好，例如使用：

```
<a href="javascript:void(0); " onclick="showLogin()">登录</a>
```

如果一切都按设计的样子显示操作起来当然没什么问题，但是当网络问题或者其他原因导致无法操作时，例如网页还没加载完，脚本还没绑定元素，或者网络太慢部分脚本丢失的话，那原来这个链接就作废了。因为它不是一个完整的链接，它的href里指向的不是一个有效的网页，而是防止页面跳转的脚本。即

使一切都是正常的，但是有的人喜欢使用点击右键，选择新窗口或者新标签打开时，一样会有问题。假如链接是这样的话呢？

```
<a href="login.html" onclick="showLogin();return false;">登录</a>
```

这样的话，即使在不支持或者禁止JavaScript的情况下，都可以保持原来链接的目的，当然，在正常的情况下，它又可以很好的表现。

如果内容是由JavaScript输出的话，就有可能完全看不到内容，如果是不支持或者禁止JavaScript的情况下又应该怎样呢？可以使用 `noscript` 标签来处理禁止JavaScript的情况，`noscript`这个标签从IE3就开始支持，不支持JavaScript的浏览器就不会支持它，支持JavaScript的浏览器检查浏览是否启用了JavaScript，如果开启，浏览器隐藏`noscript`标签和其内容，如果是禁止的情况，浏览器显示`noscript`标签里的内容。例如这样：

```
<noscript><p>传说中的内容。</p></noscript>
```

不过我并不喜欢`noscript`，我更喜欢这样用：

```
<div id="xxx">
```

```
  <a href="xxx.html" title="如果不支持或者禁止JavaScript时是显示这里面的内容"></a>
```

```
</div>
```

在原来的(X)HTML使用上如果不支持或者禁止JavaScript时要显示的内容，让它成为默认的内容，当浏览器支持JavaScript时，把它们替换成更有你想有的内容，例如Flash或者其他的。

有天，我听到mtv.com改版，整个首页都是使用Flash，特别跑去看一下新版，结果我还是看到一个XHTML+CSS的页面，一开始我以还以为自己被忽悠了，当我查看他的源代码时才发现原来是因为浏览器的脚本被我禁止掉了，当我打开JavaScript时，仅仅出现一个满屏的Flash。它是使用JavaScript来输出Flash，并把整个站原来的内容都放在XHTML里，当检测到浏览器是支持JavaScript时，就把原来的内容通过更改CSS的属性，让内容`display:none`隐藏掉，这也是一种比较不错的作法，在对主流浏览器呈现最佳效果的同时，还保持对其他设备的兼容，我这种爱禁止JavaScript的就不要算了，但对文本浏览器，手持设备呢？一份写得很优秀的XHTML躲在Flash背后，为了那些极少人可能会使用的浏览器做准备，也为机器（比如搜索引擎）做准备。

### 13.2.3 让文字看得见

在第八章时，我已经介绍了非常多文字相关的资料了，缩放文本的好处就是给予用户权力，让他们在看不清文字的时候可以放大。当然，像我们正常视力的人，12px、14px感觉已经可以了。但对有些人却不行。我跟我家里聊天时，我总是把QQ里的文字调到最大，因为我老爸看12px 14px大小的字很吃力。想一下，如果视力不好的人浏览网页时，看12px大小的字时就像我们看10px大小的字一样，你看着一大堆10px大小的文字你会感觉怎样呢？

除了字体大小外，还有个更严重的问题就是色彩，看一下像这样的文字你看起来怎样（图13.7）：



图13.7

可能这样并不是你设计的样子，你是这样写着的：

```
body{  
    background-color:#000;  
}
```



```

p{
    padding:5px;
    color:#333;
    background-image:url(xxx.gif);
}

```

那个xxx.gif的图片是白色的，可能因为网络原因没有把图片加载进来，导致出现刚才那样难以阅读的情况。正常情况下它应该是这样的（图13.8）：



图13.8

有去尝试过CSS校验的朋友应该常常看到这样的警告：

- 你没有为你的背景色设置(前景)颜色
- 你没有为你的(前景)颜色设置背景色
- 颜色和背景颜色使用了相同的颜色

有时候，我们因设计的用上了图片来当背景，同时还把背景图片的颜色当成背景色，文字在上面看上去很正常，但并不是所有人看到的都可能这样，例如我以前的手机支持CSS定义的颜色，背景色，也支持图片，支却不支持背景图片，你猜我会看到什么呢？在使用背景图片可以也把同色或者接近的背景色也写上去，因为背景图片是叠在背景色上面的。如果背景图片能显示的话，用不上背景色了。

背景颜色跟文字颜色也可能是因为设计的需求设计成很接近，可能对我们没很大的影响，但是视力差的人呢？如果两个颜色的亮度差异以及色彩差异都大于某个程度的话，这两个颜色就被视为能提供良好的色彩可见性。W3C建议的亮度差异应大于125，而色彩差异则应大于500。那怎样算亮度差异和色彩差异呢？虽然有公式，但太复杂了，也不方便，我推荐你使用一个叫Colour Contrast Analyser的软件（图13.9），它是根据W3C 的算法制作的。

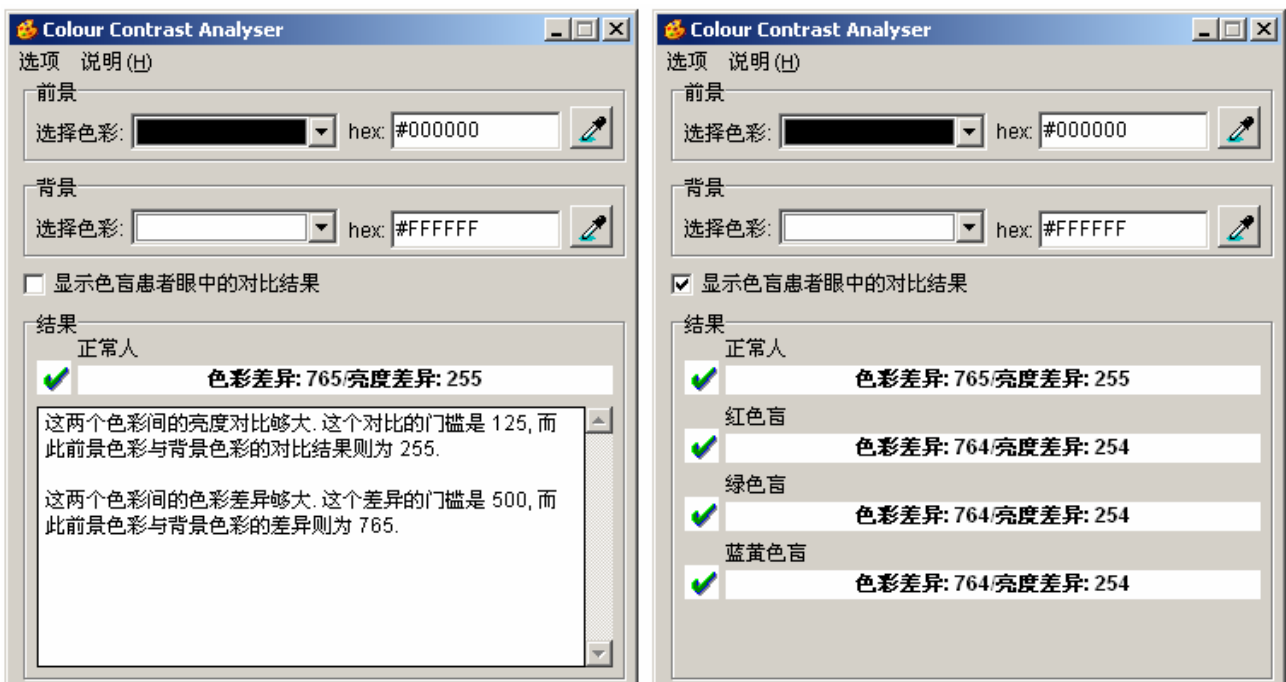


图13.9

使用这个软件可以让你方便地看到能不能达到色彩可见性，还有色盲患着的情况。除了纯文本的文字外，图片里的文字也有可能相同的问题。如果你想让图片里的文字是作内容表达的话，你也应该注意一下色彩可见性。

### 13.2.3 热键操作

鼠标坏了怎么办？买新的。在买新的之前呢？按键盘的Tab键看看网页，找一下哪里卖的鼠标又好用又便宜。

当然这是个玩笑，但是在浏览器前，绝大多数的网页只能按Tab或者shift+Tab来切换链接打开网页。而且说不定有人还把你切换到的链接的虚线框去掉，让你看不到你已经跳到哪个链接上。嗯，还是鼠标好用，不过我们还是要来假设没有鼠标的情况下要改善的地方。

一般链接我们都用对它定义[a:hover](#){}来改变鼠标移上去的效果，就像这样（图13.10）：

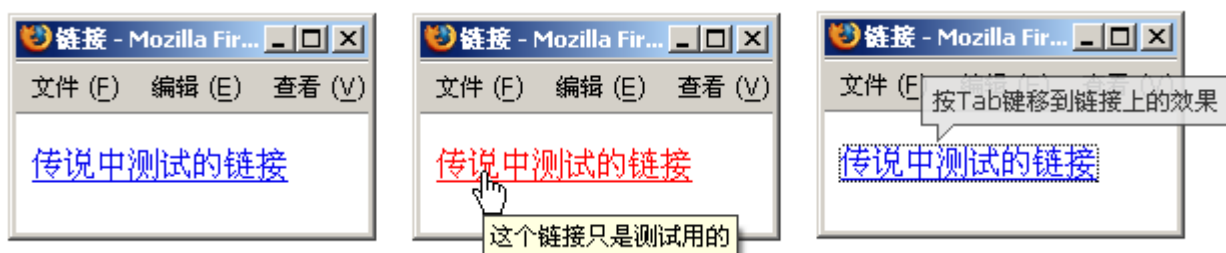


图13.10

鼠标移上去的效果比较明显，按键盘的Tab键移上去默认的只有虚线，假如在刚才的[a:hover](#)后面补充个选择符的话：

```
a:hover,a:focus{
    color:#F00;
}
```

就可以达到图13.11这样的效果。



图13.11

不过IE还不支持:focus这种伪类，更重要的是，IE按Tab键是触发:active这种状态，:active是表示鼠标点击下去的状态，不过我们还是可以这样写：

```
a:hover,a:focus,a:active{
    color:#F00;
}
```

效果如图13.12：

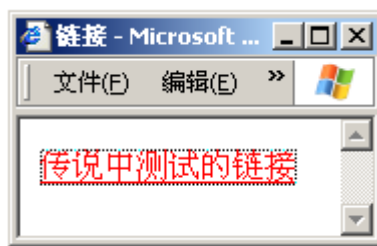


图13.12

如果想按键盘的Tab键跟鼠标点击下去显示是不一样的效果可以分开写，但是IE就没办法，因为它处理成同一种状态。

在网页里，按键盘的Tab键会跳到下一个链接或者表单元素，如果同时按下shift键跟Tab键时，会跳回上一个链接或者表单元素。

其实我们还可以控制按Tab键的顺序，那就是使用tabindex，例如：

```
<p>
  <a href="#" >链接1</a>
  <a href="#" tabindex="2">链接2</a>
  <a href="#" tabindex="1" >链接3</a>
  <a href="#" >链接3</a>
</p>
```

这样在网页里按下一次Tab键是直接跳到“链接3”，再按时是跳到“链接2”，之后才是按顺序跳到他没有使用tabindex的链接中。tabindex的取值0~32767，我建议从1开始，因为有的浏览器忽略掉值是0的，有的浏览器超过32767的数还是可以使用。如果使用相同的值时，是按在文档内容里的顺序跳下去。如果取值为-1会出现什么情况呢？会给浏览器忽视掉，永远跳不到那个元素上。

这样的话，我们就可以把tabindex用在比较重要的地方，例如按一下就跳到搜索框里，可以给用户提供一些方便。

要注意，只在a、area、button、input、object、select和textarea这几个元素有tabindex这个属性，可以把tabindex写在其他元素里，在多数浏览器也是有效的，不过意义就不是很大了。

除了tabindex，其实还有个更方便的属性，它就是accesskey。例如：

```
<a href="#" accesskey="x" >传说中的链接</a>
```

这样的话，只要按特定的键+“x”就可以选中链接，但是各浏览器按的键并不统一。IE系列使用alt+key（alt是键盘的alt键，key表示accesskey的值），如果是链接的话，再按Enter键就可以打开链接，如果是表单元素的可就直接选中元素。Firefox 2.0以下的是也是使用alt+key，如果是链接的话是直接打开链接，但是Firefox 2.0是使用Alt+Shift+Key，官方文档说：“网页提供的快捷键（Access key）现在在 Windows 上请以Alt+Shift+Key 来使用，在 Mac OS X 为 Ctrl+key，Unix 则是 Ctrl+Shift+key。”可是我的SUSE上的Firefox仍然是用 Alt+Shift+Key才是有效。

Opera的方式是按下Shift+Esc 激活accesskey面板，再按Key直接选择，图13.13是我以前做的小站的部分截图：



图13.13

跟Opera使用方式差不多的还有Konqueror, Konqueror是个按下Ctrl 激活 accesskey 面板, 除去已经分配的 accesskey外, 会自动把其他的键按页面链接的顺序按内置的机会分配, 字母分完了就分配数字, 都分完了后面的就没有, 不过Konqueror这个浏览器, 估计没什么人会去用, 现在Linux的发行版大多默认是Firefox。

在IE上使用accesskey取值就要注意了, 最好不要跟浏览器菜单(图13.14)上取值相同, 不然会导致浏览器的菜单无法通过Alt+快捷键来操作。

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

图13.14

因为accesskey并没有显示, 那要通过什么方式知道它们到底是什么值呢? 比较通用的做法是在title里补充这一信息, 例如:

```
<a href="#" accesskey="X" title="这个链接只是测试用的- accesskey键 X">传说中的链接</a>
```

这是意味着要鼠标移上去才可以看得到, 因为按Tab键是不会有title提示的。我也试过用样式处理成这样(图13.15):



图13.15

按Tab键盘会触发:focus, 再通过样式把它显示出来, 不过这样还是没办法直观的看到Access Key。还有另一种方式是:

```
a:after{
    content:("attr(accesskey)");
}
```

效果如图13.16:

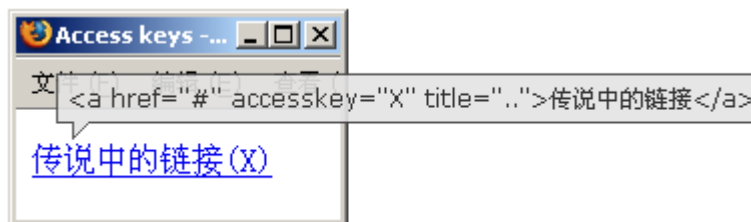


图13.16

通过伪类, 把属性值提出来放在后面显示出来, 如图13.16这样显示。不过不支持:after的浏览器就看不到这样的效果, 比如IE。

也有的朋友直接改变文本的内容, 例如:

```
<a href="#" accesskey="X" title="这个链接只是测试用的">传说中的链接(X)</a>
```

因为中文并不像英文那样可以直接取单词里的字体出来用, 例如像这样:

```
<a href="#" accesskey="C" title="这个链接只是测试用的"><span>C</span>SS</a>
```

然后通过CSS给添加下划线, 就像图13.17这样:

```
a {
    text-decoration:none
}
a span{
    text-decoration:underline
}
```

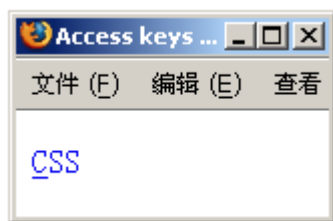


图13.17

当然，如果你不想去掉链接原来的下划线的话，可以使用border-bottom来代替。去掉链接的下划线是用在特定的情况下，例如导航菜单，不然，在一大段文字中你怎样区别没有下划线的链接呢？

```
a span{  
    border-bottom:1px solid;  
    padding:0 0 1px;  
}
```

效果如图13.18：

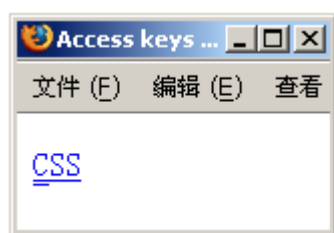


图13.18

虽然我们这一节做的这些可能对大多数人都没什么特别的意义，但是，这些却对一部分人浏览网页起到非常大的作用。

## 13.2.4 导航

如果你是从看别人的代码开始学习Web标准的话，你可以会看到像这样的代码：

```
<a href="#content">跳到内容部分</a>
```

```
<a href="#main">Skip to main content</a>
```

有的还是在网页上的某个小角落显示，有的根本就被隐藏掉。可能你认为在浏览器里跳上一小段距离并不会有多大的意义。其实锚点是有很多作用的，例如最直观的，你有一份文档，很长（如W3C那些文档都是很长的十几页），并在开头有一份目录，你会期望当点击目录的某一项里，就跳到相应的内容，这时锚点的作用就出来了。可能在我们设计的网页里可能没有这样情况，但如果你试着把CSS关掉，可能从网页开始到主要的内容部分就有很长一段距离了，假如把网页放在文本浏览器或者手持设备里呢？一个简单的锚点可能可以让别人减少按多次的方向键。

除了可以使用锚点外，还可以使用<link>来导航。link除了用来链入样式表外还有很多作用。例如在head加上：

```
<link rel="start" title="首页" href="http://www.aoao.org.cn/" />
```

```
<link rel="search" title="搜索" href="http://search.aoao.org.cn/" />
```

使用Opera并打开导航栏时可以看到图13.19这样的，点一下可以直接跳到那个链接的页面上。

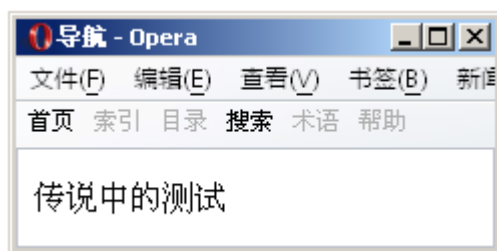


图13.19

我们发文章常常会有上一页、下一页，其实也可以使用这种方式来处理，例如在head里加上：

```
<link rel="prev" title="前一页的标题" href="http://前一页/的/网址" />
```

```
<link rel="next" title="下一页的标题" href="http://下一页/的/网址" />
```

除了这几个还有表示索引的index，表示目录的contents，表示专业术语的glossary，表示帮助的help，表示第一页的first，表示最后一页的last，表示版权的copyright，还有表示作者的author。

如果每个网站都有这种导航的话，我相信大家在浏览网页时会方便不少。不过这个也需要浏览器的支持，例如最多人用的IE就不支持，Firefox需要安装插件才可以使用，更重要的是，这些导航对文本浏览器、屏幕阅读器有着不一样的意义。几个对我们看起来不重要的导航却能方便他们。

除了这些看不到的导航，网页当然还需要看得见的导航，那看得见的导航应该处于(X)HTML里什么位置呢？如果导航比较小，例如只有几个一级菜单，我推荐放在(X)HTML的开头处。但是有的站的导航比较大，例如有很多二级菜单，甚至有三级四级菜单。我推荐放在内容的后面，并在(X)HTML开头的地方提供一个可以跳到导航的链接，不用担心会影响设计，CSS会解决这个问题。

一个网站的导航就像是一本书的目录，当你刚翻开这本书时，你可能比较想看到目录，可当已经翻到某一页内容时，目录对你的作用暂时就变得很小了，相同的道理我们也可以这样处理网页的导航：

首页，导航放在网页的开头处；内容页，导航放在内容的(X)HTML后面。可能你会感觉这样设计很可笑，可能外观一样并会放在同一位置的导航却要放在(X)HTML不同的位置，但当你使用手持设备访问网站时，你就不会感觉到可笑了。

搜索（Google、Baidu等）大家应该经常使用吧，那网站内部的呢？当你在访问一个网站时，突然想搜索网站的某些相关的东西，当然不会跑去自己习惯用的搜索网站来搜索这个网站的内容吧。网站内容的搜索可能更适合你的使用，而且搜索对于一个不了解这个网站的人来说可能是最好的导航。

从视觉上，搜索应该放在网站某一个位置，最好是固定浮动在某个位置，而在代码位置上，还是跟刚才的导航处于同一种情况会比较好。

## 13.2.5 标题与内容

每个网页都应该有个标题，如同每个人都有个名字一样。

同一个网站的每个网页标题应该都不同，如同同一个班级的每个同学的名字不同一样。

标题是个特别的元素，它并不显示在页面里，而是在浏览器上，如果它们的标题都是一样的话，当你打开一大堆窗口或者一大堆标签页时，你就无法去分清他们。

如果我的小站各个页面的标题是图13.20这样的话：

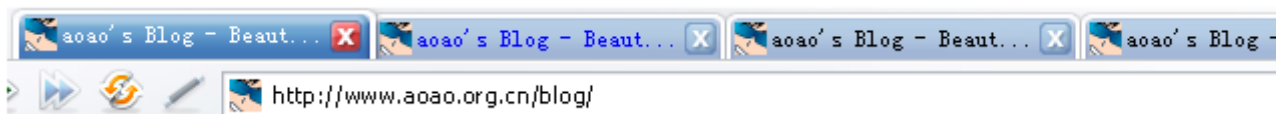


图13.20

你知道哪一个对应哪一页吗？如果是图13.21这样的呢？





图13.21

是不是看起来好一点，title除了给我们可以直观了解网页的大约内容外，从某种意义上还可以给机器看。当然，更重要的是像使用屏幕阅读器之类的，假如你什么都看不到，只能听到一大堆一样的标题时，你怎样选择听哪个网页呢？也有些浏览器会在网页的第一行显示标题，一个更有意义的标题会对使用这种非常规的浏览器的人更友好一些。

除了标题，网页更重要的部分是内容，当然内容还是有分主次的，例如正文，一些相关的信息，不管怎样想，主要的内容都应该是放在前面的，对各种浏览器都好，机器也好。例如用手机看网页，你是想先看内容，还是先看到相关信息呢？即使更用主流的浏览器，出现因网络或者其他原因，网页打开一半给卡住的，如果是内容在前面也许已经可以看完内容，但如果是相关信息放在前面，也许就没机会看到内容。内容在HTML的位置并不会影响到我们的设计，即使内容要显示在网页的右边或者中间，CSS都能帮我们解决这个问题，回头看一下第五章的布局之道。

## 13.2.6 语言与编码

这个是我接触最少的内容，我想不出用什么来引导你，如果我直接说：“假如你不在网页里申明语言的种类，可能会导致屏幕阅读器会以导致无法阅读。”的话，估计大家就直接跳过这节了。

给网页声明语言除了在第八章介绍的会影响文字的默认字体外，还可以给我们提示额外的控制，就像这样：

```
<p> “<span lang="en-UK" xml:lang="en-UK">English</span>” 是英文，必须声明语言。 </p>
```

这样就可以通过CSS控制不同语言的文本：

```
span[lang="en-UK"]{  
    font-family:Verdana, Arial, Helvetica, sans-serif;  
}
```

或者：

```
span[xml\lang="en-UK"]{  
    font-family:Verdana, Arial, Helvetica, sans-serif;  
}
```

效果如图13.22：



图13.22

其实语言选择符(:lang())只是属性选择符的一个缩写。不过，IE6还是不支持这种选择符，如果想控制的话，就得必须添加额外的标记，例如使用class。的确，直接使用class会比声明语言会更简单，但是，声明语言不是为了用于CSS控制。

在WCAG2.0里，语言的要求已经降低了，但是整个网页的声明却是不可少的。


那谁受益呢？阅读器、机器。机器并非百分百的智能，特别对待同一网页使用不同的，例如你的网页是简体中文的，并在里面引用了一段日文的汉字，相间的文字可能就表示着不同的意义，单单简体中文跟



繁体中文在某些词上都不一样，更别说其他的语言。

## 13.2.7 亲和力声明

如果用户不知道我们所做的一切，自己去发现可能需要很长的时间，用户并非像开发者一样了解网站。那为什么不能写一份亲和力声明，在用户操作遇到困难时有解决的办法呢？

那应该在这份亲和力声明写些什么呢？比如把定义的`accesskey`快捷键列出来，并说明一下用法；告诉用户如果感觉文字太小可以通过什么方式来放大、你网站上的链接有做过什么特别的处理。例如站外的链接都是在文字后有个这样（）的小图标。

## 13.2.8 建议

如果是专门研究无障碍的朋友看到这一节的内容可能会笑，因为距离真正的无障碍还很远，也没有把WCAG的思想完全传达出来。虽然我想努力做好，可是很多东西我并不能亲身体验，并不能得到真正的体会。

《Dive Into Accessibility: 在 30 天内打造更具亲和力的网站》这本小书很好，而且是网络发行的，你可以下载到中文的版本。这本书虚构了五个人，他们都有着身体上、心理上、以及技术上的障碍，使他们在浏览网页比较困难，都能代表被障碍困扰的真实人们，包括访问网站所遇到的状况。

当你在为网站进行无障碍改造时，我建议让你的朋友们去评估你的成果，而不仅仅是由检验去决定。可能有身体障碍的朋友给你的意见会是更重要的。

更多关于本书的信息请访问 <http://www.aoao.org.cn>