# 第一章 CSS和文档

## 1.1 替换元素和非替换元素

## 1.2 元素显示角色

替换元素：img、input等

非替换元素：段落、标题、表单元素、列表和XHTML中的几乎所有元素都是非替换元素。

除了替换和非替换元素，元素还分为块级元素和行内元素。

1.块级元素：元素框之前和之后生成了“分隔符”，例如p和div，列表项是块级元素的一个特例，出了表现方式和块元素一样，还会生成一个标记符，无序是圆点，有序是数字。

2.行内元素：在文本行内生成元素框，而不会打断这行文本。例如a，strong，em等。不会在之前和之后产生分隔符，所以也可以出现在另一个元素内容中。

在HTML中块级元素不能继承自行内元素（例如：<p>不能嵌套在<span>中），但是在CSS中对于显示角色如何潜逃不存在任何限制。

<body>  
 <p>this is p <span>this is span</span></p>  
</body>

Span可以继承至p,反过来不行。即行内元素可以继承至块元素。

CSS没有这种限制

p{  
 display: inline;  
}  
span{  
 display: block;  
}

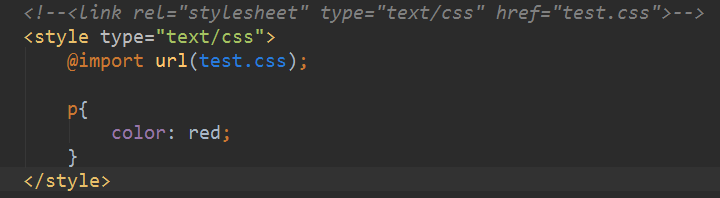
## 1.3 外部样式表

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

## 1.4 文档样式表或嵌套样式表

<style type="text/css">  
 p{  
 color: red;  
 }  
</style>

## 1.5 @import指令



与link类似，用于指示加载一个外部样式表，唯一的区别在于具体的语法和位置，@import出现在style容器中，要放在其他css规则之前。较老的浏览器无法处理不同形式的import指令，可以利用这一点屏蔽样式。link无兼容性，后者CSS2.1以下浏览器不支持,Link 支持使用javascript改变样式，后者不可。

当然，如果有有一个样式表，它需要使用其他外部样式表中的样式，此时@import就非常有用。由于外部样式表不能包含任何文档标记，所以不能使用link元素，但能用@import，需要注意的是，@import只能出现在样式表的开头，CSS要求@import指令出现在样式表中的其他规则之前。

## 1.6 CSS注释

<style type="text/css">  
 *<!--* @import url(test.css);  
 p{  
 color: red;  
 }  
 *-->*</style>

浏览器可能忽略无法识别的标记，对于样式表也一样。如果浏览器无法识别<style></style>，就会将其忽略掉，但是声明不一定会忽略，所以看上去像是正常的文本，这时候这些样式生命就会出现在页面的最上面，但是浏览器应该忽略他们，因为不是<body>的元素嘛，解决的办法是可以在声明包含在一个注释标记中，如果较老的浏览器无法识别<style>，还会忽略注释中的样式声明，因为HTML注释是不会显示出来的，但是如果能识别<style>浏览器仍然能正常读取样式表。

## 1.7 内联样式

<body>  
 <p >this is p <span style="color:red;">this is span</span></p>  
</body>

this is p

this is span

内联样式表不能在style中放@import，通常不建议使用内联样式表。

# 第二章 选择器

## 2.1 元素选择器

html{  
 background: grey;  
}  
  
p{  
 display: inline;  
}  
  
span{  
 display: block;  
}

### 2.1.1 选择器分组

p,span{  
 color: black;  
}

### 2.1.2 通配选择器

\*{  
 color: red;  
}

### 2.1.3 声明分组

p{  
 background: yellow;  
}  
  
p{  
 color: black;  
}

p{  
 background: yellow;  
 color: black;  
}

以上两者效果一样

## 2.2 类选择器

<body>  
 <p class="warning">  
 this is the first p element！  
 </p>  
 <p>  
 this is the second p element!  
 <span class="warning">  
 this is a span element!  
 </span>  
 </p>  
</body>

\*.warning {  
 color: red;  
}

this is the first p element！

this is the second p element! this is a span element!

\*.warning {  
 color: red;  
}  
  
p.warning{  
 background: black;  
}

this is the first p element！

this is the second p element! this is a span element!

### 2.2.1 多类选择器

<body>  
 <p class="warning urgent">  
 this is the first p element！  
 </p>  
 <p class="urgent">  
 this is the second p element!  
 <span class="warning">  
 this is a span element!  
 </span>  
 </p>  
</body>

\*.warning {  
 font-weight: bold;  
}  
  
.urgent{  
 font-style: italic;  
}  
  
.urgent.warning{  
 background: red;  
}

***this is the first p element！***

*this is the second p element!****this is a span element!***

**在IE7之前，不同平台中的IE都不能正确的处理多类选择器,比如p.warning.help只会匹配class为help的p元素，而p.help.warning只会匹配class为warning的p元素，因为warning在选择器最后出现。**

## 2.3 ID选择器

首先，ID应该是唯一的。

不同于类选择器，ID选择器不能结合使用，ID属性不允许以空格分隔词列表。

<body>  
 <p class="urgent warning help">  
 this is the first p element！  
 </p>  
 <p id="important">  
 this is the second p element!  
 <span class="warning">  
 this is a span element!  
 </span>  
 </p>  
</body>

#important{  
 background: yellow;  
}

this is the first p element！

this is the second p element! this is a span element!

## 2.4 属性选择器

### 2.4.1 简单的属性选择

<body>  
 <h1 class="class1">This is h1！</h1>  
 <h1 class="class2">This is h1！</h1>  
 <h2 class="class3">This is h2!</h2>  
 <h3 class="class4">This is h3</h3>  
</body>

h1[class] {  
 color: red;  
}

<body>  
 <a href="http://www.baidu.com" title="百度">百度</a>  
 <a href="http://www.google.com">谷歌</a>  
 <a title="锚">非链接</a>  
</body>

a[href][title]{ /\*注意不能有空格\*/  
 color: red;  
}

[百度](http://www.baidu.com/) [谷歌](http://www.google.com/) 非链接

### 2.4.2 根据具体属性选择

a[href][title]{ /\*注意不能有空格\*/  
 color: red;  
}  
  
a[href="http://www.google.com"]{  
 color: green;  
}

[百度](http://www.baidu.com/) [谷歌](http://www.google.com/) 非链接

a[href][title]{ /\*注意不能有空格\*/  
 color: red;  
}  
  
a[href="http://www.baidu.com"][title="百度"]{  
 background: black;  
}  
  
a[href="http://www.google.com"]{  
 color: green;  
}

[百度](http://www.baidu.com/) [谷歌](http://www.google.com/) 非链接

### 2.4.3 根据部分属性值选择

<body>  
 <p class="urgent warning">this calsses include urgent and warning!</p>  
 <p class="urgent">this class include urgent!</p>  
 <p class="warning">this class include warning!</p>  
</body>

首个p元素是不会匹配到的，因为使用属性选择器需要完全匹配

p[class="urgent"]{  
 color: red;  
}

this calsses include urgent and warning!

this class include urgent!

this class include warning!

p[class~="urgent"]{/\*使用~，部分选择器的关键，不完全匹配特点\*/  
  
 background: yellow;  
}

this calsses include urgent and warning!

this class include urgent!

this class include warning!

<body>  
 <p class="urgent warning">this calsses include urgent and warning!</p>  
 <p class="urgent success">this class include urgent!</p>  
 <p class="warning success">this class include warning!</p>  
</body>

p[class\*=urgent]{ /\*匹配所有含有classs属性为urgent的元素\*/  
 color: red;  
}

this calsses include urgent and warning!

this class include urgent!

this class include warning!

p[class^=war]{ /\*匹配所有以war为开头的的元素\*/  
 color: red;  
}

this calsses include urgent and warning!

this class include urgent!

this class include warning!

p[class$=ss]{ /\*匹配所有以ss结尾的元素\*/  
 color: red;  
}

this calsses include urgent and warning!

this class include urgent!

this class include warning!

当然不单单class属性可用，其他属性也是受用的。但是子串匹配属性选择器受到了浏览器的限制。

### 2.4.4 特定属性选择类型

## 2.5 后代选择器[包含、上下文选择器]

### 2.5.1 选择后代元素

<body>  
 <ul>  
 <li>List Item 1</li>  
 <li>List Item 2</li>  
 <li>List Item 3</li>  
 </ul>  
</body>

ul li{ /\*ul的所有后代元素li都会匹配上\*/  
 color: blue;  
}

* List Item 1
* List Item 2
* List Item 3

<body>  
 <ul>  
 <li>List Item 1  
 <ol>  
 <li>List Item 1-1</li>  
 <li><em>List Item 1-2</em></li>  
 </ol>  
 </li>  
 <li>List Item 2</li>  
 <li>List Item 3</li>  
 </ul>  
</body>

ul li{ /\*ul的所有后代元素li都会匹配上\*/  
 color: blue;  
}  
  
ul em{ /\*不管em嵌套多深\*/  
 color:red;  
}

* List Item 1
  1. List Item 1-1
  2. *List Item 1-2*
* List Item 2
* List Item 3

### 2.5.2 选择子元素

在某些情况下，并不想选择一个任意的后代元素，而是希望缩小范围，只选择另一个元素的子元素。

<body>  
 <p>This is<strong> very </strong>important.</p>  
 <p>This is <em>really <strong> very </strong></em>important.</p>  
</body>

p > strong{ /\*只选择p的子元素，而不是所有的p的后代元素\*/  
 color: red;  
}

This is**very**important.

This is *really****very***important.

p>strong{ /\*只选择p的子元素，而不是所有的p的后代元素\*/  
 color: red;  
}  
  
p>em>strong{ /\*注意属性选择器不允许有空格，所以最好不要有空格\*/  
 color:blue;  
}

This is**very**important.

This is *really****very***important.

### 2.5.2 选择相邻兄弟元素

<div>  
 <ol>  
 <li>ol>li 1</li>  
 <li>ol>li 2</li>  
 <li>ol>li 3</li>  
 </ol>  
 <ul>  
 <li>ul>li 1</li>  
 <li>ul>li 2</li>  
 <li>ul>li 3</li>  
 </ul>  
</div>

li+li{  
 color:red;  
}

1. ol>li 1
2. ol>li 2
3. ol>li 3

* ul>li 1
* ul>li 2
* ul>li 3

需要注意的是第一个元素是不会变色的。用一个结合符只能选择两个相邻兄弟中的第二个元素。

ol+ul{  
 color:red;  
}

1. ol>li 1
2. ol>li 2
3. ol>li 3

* ul>li 1
* ul>li 2
* ul>li 3

**另外需要注意的是IE6之前不支持子选择器和相邻兄弟选择器。**

## 2.6 伪类选择器和伪元素选择器

### 2.6.1 伪类选择器

<body>  
 <div>  
 <ol>  
 <li><a href="http://www.baidu.com">百度</a></li>  
 <li><a href="http://www.qq.com">腾讯</a></li>  
 <li><a href="http://www.360.com">360</a></li>  
 </ol>  
 </div>  
</body>

/\*伪类的顺序很重要\*/  
/\*link-visited-focus-hover-active\*/  
  
a:link { /\*link\visited是静态伪类\*/  
 color: blue;  
}  
  
a:visited{  
 color: red;  
}  
  
  
a:hover{ /\*focus\hover\active是动态伪类\*/  
 color: green;  
}  
  
a:active{  
 color: yellow;  
}

动态伪类可以应用到任何铁元素、

<body>  
 <input type="text">  
</body>

input:focus{  
 background: yellow;  
 font-weight: bold;  
}

****

**IE6只允许动态伪类选择超链接，而不允许其他元素使用，IE7支持所有元素都能应用：hover,但是不支持对表单元素应用：focus样式。**

input:focus{  
 background: yellow;  
 font-weight: bold;  
 font-size: 100px;  
}

**不要依赖这种行为的设计，文字大小改变了，强行改变了网页的布局，会重绘文档。**

<div>  
 <p>This is first P</p>  
 <ul>  
 <li>This is first li</li>  
 <li>This is second li</li>  
 </ul>  
 <p>This is last P</p>  
</div>

/\*注意不是p的子元素变蓝色\*/  
/\*而是第一个作为某元素的子元素的P元素变蓝色\*/  
p:first-child{  
 color: blue;  
}  
  
li:first-child{  
 color: red;  
}

This is first P

* This is first li
* This is second li

This is last P

**IE6之前不支持:first-child，不过IE7支持。**

<body>  
 <div>  
 <ol>  
 <li><a href="http://www.baidu.com">百度</a></li>  
 <li><a href="http://www.qq.com">腾讯</a></li>  
 <li><a href="http://www.360.com">360</a></li>  
 </ol>  
 </div>  
</body>

a:link:hover{ /\*停留在未访问的链接时显示红色\*/  
 color: red;  
}  
  
a:visited:hover{ /\*停留在已访问的链接时显示蓝色\*/  
 color: blue;  
}

**IE6识别不了，它只能识别最后一个属性，比如上面的hover,IE7可以识别。**

### 2.6.2 伪元素选择器

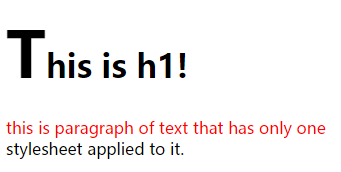
<body>  
 <h1>This is h1!</h1>  
</body>

h1:first-letter{  
 font-size: 200%;  
}



<body>  
 <h1>This is h1!</h1>  
 <p>this is paragraph of text that has only one stylesheet applied to it.</p>  
</body>

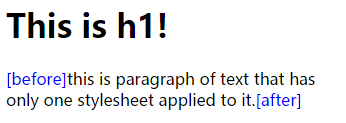
h1:first-letter{  
 font-size: 200%;  
}  
  
p:first-line{  
 color: red;  
}



CSS2中First-letter和first-line只能用于标记或段落之类的块级元素，而不能用于超链接等行内元素。CSS2.1中虽然有扩展，但是还是有限制。

<body>  
 <h1>This is h1!</h1>  
 <p>this is paragraph of text that has only one stylesheet applied to it.</p>  
</body>

p:before{  
 content: '[before]';  
 color: blue;  
}  
  
p:after{  
 content: '[after]';  
 color: blue;  
}



# 第三章 结构和层叠

## 3.1 特殊性

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,0,1
4. 结合符或者通配选择器对特殊性没有任何贡献

<body>  
 <ol>  
 <li>ID属性值：加0,1,0,0</li>  
 <li>类属性值,属性选择或伪类：加0,0,1,0</li>  
 <li>元素或伪元素：加0,0,0,1</li>  
 <li>结合符或者通配选择器对特殊性没有任何贡献</li>  
 </ol>  
</body>

li{ /\*0,0,0,1 只有li一个元素，加0,0,0,1\*/  
 color: red;  
}  
  
ol li{ /\*0,0,0,2 有连个元素，ol和li,加0,0,0,2 所以该规则胜出\*/  
 color: blue;  
}

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,0,1
4. 结合符或者通配选择器对特殊性没有任何贡献

<body>  
 <span class="class1">特殊性贡献值列表</span>  
 <ol>  
 <li>ID属性值：加0,1,0,0</li>  
 <li>类属性值,属性选择或伪类：加0,0,1,0</li>  
 <li>元素或伪元素：加0,0,0,1</li>  
 <li>结合符或者通配选择器对特殊性没有任何贡献</li>  
 </ol>  
</body>

li{ /\*0,0,0,1 只有li一个元素，加0,0,0,1\*/  
 color: red;  
}  
  
ol li{ /\*0,0,0,2 有连个元素，ol和li,加0,0,0,2 所以该规则胜出\*/  
 color: blue;  
}  
  
span.class1{ /\*0,0,1,1 胜出\*/  
 color: pink;  
}  
  
span{ /\*0,0,0,1\*/  
 color: yellow;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,0,1
4. 结合符或者通配选择器对特殊性没有任何贡献

li{ /\*0,0,0,1 只有li一个元素，加0,0,0,1\*/  
 color: red;  
}  
  
ol li{ /\*0,0,0,2 有连个元素，ol和li,加0,0,0,2 所以该规则胜出\*/  
 color: blue;  
}  
  
span.class1{ /\*0,0,1,1 胜出\*/  
 color: pink;  
}  
  
span{ /\*0,0,0,1\*/  
 color: yellow;  
}  
  
  
html>body ol li[id='li']{ /\*0,0,1,4 胜出\*/  
 color: pink;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,0,1
4. 结合符或者通配选择器对特殊性没有任何贡献

li{ /\*0,0,0,1 只有li一个元素，加0,0,0,1\*/  
 color: red;  
}  
  
ol li{ /\*0,0,0,2 有连个元素，ol和li,加0,0,0,2 所以该规则胜出\*/  
 color: blue;  
}  
  
span.class1{ /\*0,0,1,1 胜出\*/  
 color: pink;  
}  
  
span{ /\*0,0,0,1\*/  
 color: yellow;  
}  
  
  
html>body ol li[id='li']{ /\*0,0,1,4\*/  
 color: pink;  
}  
  
#li{ /\*0,1,0,0 该规则胜出\*/  
 color: green;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,0,1
4. 结合符或者通配选择器对特殊性没有任何贡献

### 3.1.1 声明和特殊性

### 3.1.2 通配选择器特殊性

\*{ /\*0,0,0,0 没有贡献\*/  
 font-family: "微软雅黑";  
}  
   
li{ /\*0,0,0,1 胜出\*/  
 font-family: "宋体";  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,1,0
4. 结合符或者通配选择器对特殊性没有任何贡献

注：结合符 ul+li中的+等。

### 3.1.3 ID和属性选择器的特殊性

#li{ /\*0,1,0,0 胜出\*/  
 color: red;  
 }  
  
\*[id='li']{ /\*0,0,1,0\*/  
 color: blue;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,1,0
4. 结合符或者通配选择器对特殊性没有任何贡献

### 3.1.4 内联样式

<body>  
 <span class="class1">特殊性贡献值列表</span>  
 <ol>  
 <li id="li">ID属性值：加0,1,0,0</li>  
 <li>类属性值,属性选择或伪类：加0,0,1,0</li>  
 <li>元素或伪元素：加0,0,1,0</li>  
 <li>结合符或者通配选择器对特殊性没有任何贡献</li>  
 <li id="li1" style="color: red">内联样式：加1,0,0,0</li>  
 </ol>  
</body>

html>body>ol>li#li1{ /\*0,1,0,4\*/  
 color: pink;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,1,0
4. 结合符或者通配选择器对特殊性没有任何贡献
5. 内联样式：加1,0,0,0

### 3.1.5 重要性

span{ /\*0,0,0,1\*/  
 color: red; **!important**;  
}  
  
span{  
 color: blue;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,1,0
4. 结合符或者通配选择器对特殊性没有任何贡献
5. 内联样式：加1,0,0,0

没效果！

## 3.2 继承

<body>  
 <span class="class1">特殊性贡献值列表</span>  
 <ol>  
 <li id="li">ID属性值：加0,1,0,0</li>  
 <li>类属性值,属性选择或伪类：加0,0,1,0</li>  
 <li>元素或伪元素：加0,0,1,0</li>  
 <li>结合符或者通配选择器对特殊性没有任何贡献</li>  
 <li id="li1">内联样式：加1,0,0,0</li>  
 </ol>  
</body>

ol{  
 color: red;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,1,0
4. 结合符或者通配选择器对特殊性没有任何贡献
5. 内联样式：加1,0,0,0

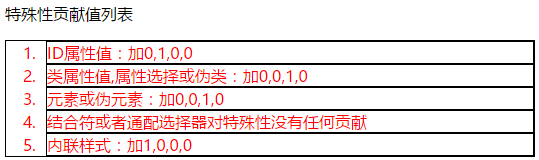
ol的子元素li都继承了ol的样式。当然有些属性是不能继承的，例如border。

ol{  
 color: red;  
 border: 1px solid black;  
}

特殊性贡献值列表

1. ID属性值：加0,1,0,0
2. 类属性值,属性选择或伪类：加0,0,1,0
3. 元素或伪元素：加0,0,1,0
4. 结合符或者通配选择器对特殊性没有任何贡献
5. 内联样式：加1,0,0,0

而不会变成下面这种继承的情况：



大多数框模型属性（包括外边距、内边距、背景和边框）都不能继承。

<body>  
 <p id="page">this is p <em>this is em</em></p>  
 <p>this is p</p>  
</body>

\*{ /\*0,0,0,0 没有贡献\*/  
 font-family: "微软雅黑";  
 color: grey;  
}  
  
#page{  
 color: red;  
}

this is p *this is em*

this is p

从上面可以看出，继承的值连0特殊性都没有，因为\*的属性是带有0特殊性的，em虽然继承了p，但是连0特殊性都没有，所以被\*所带的0特殊性样式覆盖了，所以结果还是灰色的。

这个例子也说明了不加区别地使用通配选择器可能存在问题。

继承值完全没有特殊性。

<body>  
 <ul>  
 <li><a href="http://www.baidu.com">百度</a></li>  
 <li><a href="http://www.360.com">360</a></li>  
 </ul>  
</body>

ul{  
 color: white;  
 background: black;  
}

* [百度](http://www.baidu.com/)
* [360](http://www.360.com/)

继承值完全没有特殊性，所以用户代理的超链接样式就占了上风，在一个WEB浏览器中，意味着他们的颜色可能是蓝色的，因为浏览器的样式包含了以下规则：

a:link{  
 color: blue;  
}

而完全没有想象中的白色字体！

## 3.3 层叠

<body>  
 <p>this is a p</p>  
</body>

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

this is a p

后面一条覆盖了前面一条规则。

a:link{  
 color: blue;  
}  
  
a:visited{  
 color: purple;  
}  
  
a:hover{  
 color: red;  
}  
   
a:active{  
 color: orange;  
}

所有这些选择器的特殊性都是一样的，0,0,0,1

因此与元素匹配的最后一个选择器总是会胜出。

a:hover{  
 color: red;  
}  
   
a:active{  
 color: orange;  
}  
  
a:link{  
 color: blue;  
}  
  
a:visited{  
 color: purple;  
}

任何连接都不会显示：hover和：active，因为：link和：visited规则后出现。又因为连接都必须是要么已访问，要么未访问，所以：link和：visited样式总会覆盖：hover规则。

# 第四章 值和单位

# 第五章 字体

## 5.1 字体系列

### 5.1.1 字体系列

<body>  
 <p class="serif">www.baidu.com</p>  
 <p class="sans-serif">www.baidu.com</p>  
 <p class="monospace">www.baidu.com</p>  
 <p class="cursive">www.baidu.com</p>  
</body>

p.serif{  
 font-family: serif;  
}  
  
p.sans-serif{  
 font-family: sans-serif;  
}  
  
p.monospace{  
 font-family: monospace;  
}  
  
p.cursive{  
 font-family: cursive;  
}

www.baidu.com

www.baidu.com

www.baidu.com

www.baidu.com

### 5.1.2 字体加粗

字体加粗是可以继承的。

<body>  
 <p class="serif">this is p<em> this is em</em></p>  
</body>

p.serif{  
 font-family: serif;  
 font-weight: bold;  
}

**this is pthis is em**

p.serif{  
 font-family: serif;  
 font-weight: normal;  
}  
  
em{  
 font-weight: bolder; /\*更粗\*/  
}

### 5.1.3 字体大小

<body>  
 <p class="serif">this is p<em> this is em</em></p>  
</body>

p.serif{  
 font-family: serif;  
 font-weight: normal;  
 font-size: 20px;  
}  
  
em{  
 font-weight: bolder; /\*更粗\*/  
 font-size: 200%; /\*40px,百分数是根据父元素的大小调整的\*/  
}

this is p**this is em**

<body>  
 <p class="serif">this is p<em> this is em<strong> this is strong</strong></em></p>  
</body>

p.serif{  
 font-family: serif;  
 font-weight: normal;  
 font-size: 20px;  
 }  
  
em{  
 font-weight: bolder; /\*更粗\*/  
 font-size: 200%; /\*40px,百分数是根据父元素的大小调整的\*/  
}  
  
strong{  
 font-size: 200%; /\*80px\*/  
}

this is p**this is em*this is strong***

**如果一个WEB页面上既有文本又有图像，选择使用像素值制定字体大小就非常适应。因为可以更容易让文本设置与页面上的图像元素等高。**

### 5.1.4 风格和变形

<body>  
 <p class="serif">this is a p<strong> this is a strong</strong></p>  
</body>

p.serif{  
 font-family: serif;  
 font-weight: normal;  
 font-size: 20px;  
 font-style: oblique;  
 }  
  
  
  
strong{  
 font-size: 200%; /\*80px\*/  
 font-style: italic; /\*oblique和italic基本没什么区别\*/  
}

*this is a p****this is a strong***

p.serif{  
 font-family: serif;  
 font-weight: normal;  
 font-size: 20px;  
 font-style: oblique;  
 font-variant: small-caps;  
 }

*this is a p****this is a strong***

/\*p.serif{  
 font-family: serif;  
 font-weight: normal;  
 font-size: 20px;  
 font-style: oblique;  
 font-variant: small-caps;  
 }\*/  
  
p.serif{ /\*与上面效果一样，前三者顺序任意，后两者顺序固定，font-size在前，font-family在后\*/  
 font:oblique small-caps normal 20px serif;  
}  
  
strong{  
 font-size: 200%; /\*80px\*/  
 font-style: italic; /\*oblique和italic基本没什么区别\*/  
}

# 第六章 文本属性

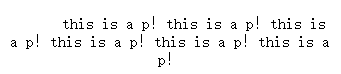
## 6.1 缩进和水平对齐

<body>  
 <p class="serif">this is a p！this is a p！this is a p！this is a p！this is a p！this is a p！</p>  
</body>

p{  
 text-indent: 3em;  
}

**一般只有块级元素才有效果，行内元素无效。**

p{  
 text-indent: 3em;  
 text-align: center;  
}

****

**IE6中有BUG。**

## 6.2 垂直对齐

Line-height属性可以继承，line-height值和字体大小之差就是行间距。

文本行中的每个元素都会生成一个内容区，这由字体大小决定，这个内容区会生成一个行内框，如果不存在其他因素，这个行内框就完全等于该元素的内容区，由line-height产生的行间距就是增加或减小各行内框高度的因素之一。

（Line-height） - （font-size） = 行间距 （该值可能是负数）

行间距/2 分别应用到内容区的顶部和底部，结果就是该元素的行内框了。

例如：font-size=14px line-height=18px（行高）

内容区=font-size=14px 行间距=上行间距+下行间距=（Line-height） - （font-size）=4px

### 6.2.1 基线对齐

Vertical-align只应用与行内元素和替换元素（img input等），该属性不能继承。

<body>  
 <class="serif">this is a p！it is include a img<img src="6E22.tmp.png"> element!</p>  
 <p class="serif">this is a p！</p>  
</body>

img{ /\*把替换元素的底边放在基线上\*/  
 vertical-align: baseline;  
}

this is a p！it is include a img element!

this is a p！

### 6.2.2 上标和下标

<body>  
 <p>this is p, include   
 <span class="sub">  
 sub  
 </span>   
 and   
 <span class="super">  
 super  
 </span>   
 </p>  
</body>

span.sub{  
 vertical-align: sub;  
}  
  
span.super{  
 vertical-align: super;  
}

this is p, include sub and super

如果想让上标或者下标小于其父元素，可以使用font-size属性。

### 6.2.3 底端对齐

img{

vertical-align: bottom;  
}

this is p, include imag 

img{   
 vertical-align: top;  
}



img{  
 vertical-align: text-bottom;  
}

this is p, include imag 

img{  
 vertical-align: text-top;  
}



### 6.2.4 居中对齐

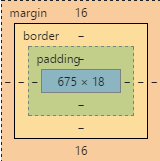
img{  
 vertical-align: middle;  
}



### 6.2.5 长度对齐

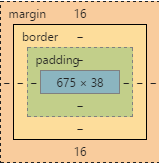
<body>  
 <p>this is p, include   
 <span class="up">  
 up  
 </span>   
 and   
 <span class="down">  
 down  
 </span>  
 </p>  
</body>

this is p, include up and down

未设置样式时行高18px

span.up{  
 vertical-align: 10px;  
}  
  
span.down{  
 vertical-align: -10px;  
}

this is p, include up and down

行高变成了38px = 10px+10px+18px（defaut）

所有的垂直对齐的元素都会影响行高。

## 6.3 字间隔和字母间隔

### 6.3.1. 字间隔

### 6.3.2. 字母间隔

### 6.3.3. 文本转换

<body>  
 <p>this is p</p>  
</body>

p{  
 text-transform: uppercase;  
}

THIS IS P

### 6.3.4. 文本装饰

<body>  
 <p>this is p</p>  
</body>

p{  
 text-decoration: underline;  
}

this is p

p{  
 text-decoration: line-through;  
}

~~this is p~~

<body>  
 <p>this is p </p>  
 <a href="http://www.baidu.com">百度</a>  
</body>

p{  
 text-decoration: line-through;  
}

~~this is p~~

[百度](http://www.baidu.com/)

p{  
 text-decoration: line-through;  
}  
  
a{  
 text-decoration: none;  
}

~~this is p~~

[百度](http://www.baidu.com/)

a:link,  
a:visited{  
 text-decoration: underline overline;  
}

超链接既有下划线又有上划线

<body>  
 <p>this is p <strong>this is a strong</strong></p>  
</body>

p{  
 text-decoration: underline;  
 color: black;  
}  
  
strong{  
 color: red;  
}

this is p **this is a strong**

**下划线的颜色一致，text-decoration不能继承，任何装饰线与父元素的颜色相同。**

<body>  
 <p>this is p <strong>this is a strong</strong> this is p</p>  
</body>

p{  
 text-decoration: underline;  
 color: black;  
}  
  
strong{  
 color: red;  
 text-decoration: none;  
}

this is p **this is a strong** this is p

并不会去掉strong元素的下划线，因为这个下划线是p的下划线，它实际上只是经过了strong元素而已。没有办法去掉父元素生成的下划线。(或者上划线或者贯穿线)

p{  
 text-decoration: underline;  
 color: black;  
}  
  
strong{  
 color: red;  
 text-decoration: underline;  
}

this is p **this is a strong** this is p

虽然没办法去掉下划线，但是可以覆盖父元素下划线的颜色。

### **6.3.5 文本阴影**

strong{  
 text-shadow: green 5px 0.5em;  
}

### **6.3.6 处理空白符**

<body>  
 <p>this is p this is  
 a p this  
 is a p  
 </p>  
</body>

this is p this is a p this is a p

会自动去除一些空格和换行。

p{  
 white-space: normal;  
}

和上面的一样，会自动丢掉多余的空白符，换行符会转换为空格。

p{  
 white-space: pre;  
}

就和文档中表示的一样，浏览器会注意额外的空格，甚至回车

this is p this is

a p this

is a p

p{  
 white-space: nowrap;  
}

this is p this is a p this is a p

会防止元素中的换行，除非使用一个br元素。

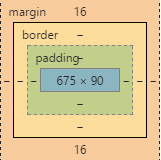
# 第七章 基本视觉格式化

## 7.1 快速复习

元素：

* 1. 正常流：大多数元素都在正常流中，要让一个元素不在正常流中，唯一的办法就是使之成为浮动或定位元素。
  2. 非替换元素：如果元素的内容包含在文档中，称之为飞替换元素。
  3. 替换元素：img和大多数表单元素
  4. 块级元素：段落、标题或div之类的元素。在正常流中时会在其框前和框后生成“换行”，通过声明display：block可以让元素生成块级框。
  5. 行内元素：strong、span、em等。这些元素不会在之前和之后生成“行分隔符”，它们是块级元素的后代。通过声明display：inline可以让元素生成一个行内框。
  6. 根元素：html

## 7.2 块级元素



左外边界—[margin左外边距] —左边框[border]—[padding左内边距] —左内边界

Width = 左内边界到右内边界

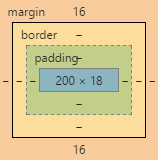
图上：675x90所在矩形框的宽度 = width

### 7.2.1 非替换元素水平格式化

<body>  
 <p>this is p</p>  
</body>

p{  
 width: 200px;  
 background: yellow;  
}

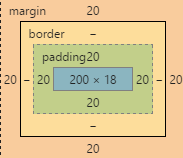




元素框宽度 = 200px

p{  
 width: 200px;  
 padding: 20px;/\*内边距\*/  
 margin: 20px; /\*外边距\*/  
 background: yellow;  
}





元素框的宽度 = 200px + 20px（左内边距）+ 20px（右边距） + 20px（左外边距） + 20px（右外边距） = 280px

需要注意的事元素框的宽度 = 外边距 + 内边距 + 边框 + width

这和CSS中设置的width有本质区别。

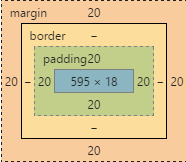
水平属性

margin-left border-left padding-left width padding-right border-right margin-right

其中 width margin-left margin-right 可以设置为auto

其余属性必须设置为特定的值，或者默认宽度为0

p{  
 width: auto;  
 padding: 20px;/\*内边距\*/  
 margin: 20px; /\*外边距\*/  
 background: yellow;  
}





使用auto

1.设置margin-left: auto;

<body>  
 <p>this is p</p>  
</body>

p{  
 width: 100px;  
 margin-left: auto;  
 margin-right: 100px;  
 background: yellow;  
}

例如7个属性和=400px

那么margin-left = 400px – 100px -100px = 200px

2.设置都不为auto

p{  
 width: 100px;  
 margin-left: 100px;  
 margin-right: 100px;  
 background: yellow;  
}

此时会把margin-right设置强制设置为auto

3.设置width: auto;

p{  
 width: auto;  
 margin-left: 100px;  
 margin-right: 100px;  
 background: yellow;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\53CC.tmp.png

可以看到p元素相对屏幕居中了。

4.设置两个外边距为auto

p{  
 width: 100px;  
 margin-left: auto;  
 margin-right: auto;  
 background: yellow;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\559B.tmp.png

相对于屏幕还是居中了。

将两个外边距设置为相等长度是将元素居中的一种正确方法。

5.设置左边距和width为auto

p{  
 width: auto;  
 margin-left: auto;  
 margin-right: 100px;  
 background: yellow;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\5A64.tmp.png

相当于margin-right = 0px

6.设置三个值都为auto

p{  
 width: auto;  
 margin-left: auto;  
 margin-right: auto;  
 background: yellow;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\5C8C.tmp.png

此时两个外边距设置为0px，而width会尽可能的宽。

隐式偏移

<body>  
 <div>  
 <p>this is p</p>  
 </div>  
</body>

div{  
 background: yellow;  
 padding: 20px;  
}  
  
p{  
 margin: 20px;  
 padding: 0;  
 background: red;  
}

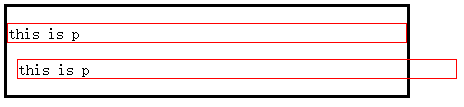


其实p元素相对于黄色边界偏移了40px

负外边距

<body>  
 <div>  
 <p>this is p</p>  
 <p class="next">this is p</p>  
 </div>  
</body>

div{  
 width: 400px;  
 border: 3px solid black;  
   
}  
  
p.next{  
 margin-left: 10px;  
 width: auto;  
 margin-right: -50px;  
}

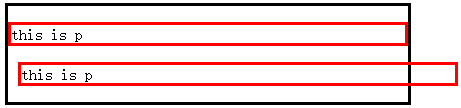


10px + 0 + 0 + 440px + 0 + 0 -50px = 400px（以上的红色边框是为了视图效果需要，实际不存在）

七个元素加起来仍然是400px = 父元素的width

尽管导致子元素超出了其父元素，并且子元素的width大于了父元素的width

div{  
 width: 400px;  
 border: 3px solid black;  
   
}  
  
p{  
 border: 3px solid red;  
}  
  
p.next{  
 margin-left: 10px;  
 width: auto;  
 margin-right: -50px;  
}



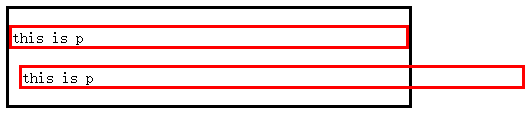
10px + 3px + 0px + width + 0px + 3px – 50px = 400px

Width = 434px

如果还要这是padding，那么width会更小。

设置margin-right为auto

div{  
 width: 400px;  
 border: 3px solid black;  
   
}  
  
p{  
 border: 3px solid red;  
}  
  
p.next{  
 margin-left: 10px;  
 width: 500px;  
 margin-right: auto;  
}

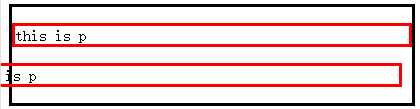


10px +3px + 0px + 500px + 0px + 3px + Margin-right = 400px

Margin-right = -116px

如果设置左外边距为负值

div{  
 width: 400px;  
 border: 3px solid black;  
   
}  
  
p{  
 border: 3px solid red;  
}  
  
p.next{  
 margin-left: -50px;  
 width: auto;  
 margin-right: 10px;  
}



不仅超出了DIV，而且超出了浏览器窗口本身的边界！

**同时需要记住，内边框、边框和内容宽度（及高度）绝对不能为负，至于外边距能小于0.**

### 7.2.2 替换元素水平格式化

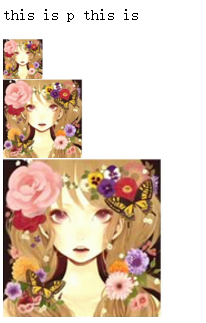
其他规则和非替换元素一样，关键的区别是width为auto，则元素的宽度是内容的固有宽度。

img{  
 display: block;  
 width: auto;  
 margin: 0;  
}

原图像的宽度。

<div>  
 <p>this is p this is</p>  
 <img class="first" src="6E22.tmp.png">  
 <img class="second" src="6E22.tmp.png">  
 <img class="last" src="6E22.tmp.png">  
 </div>

img.first{  
 display: block;  
 width: 40px;  
 margin: 0;  
}  
  
img.second{  
 display: block;  
 width: 80px;  
 margin: 0;  
}  
  
img.last{  
 display: block;  
 width: 160px;  
 margin: 0;  
}



Width变化了，height也会随着比例变化。

### 7.2.3 垂直格式化