# 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;  
}

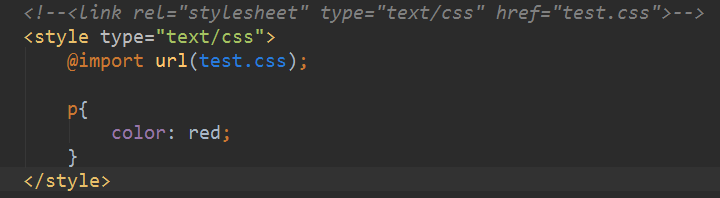
## 1.3 外部样式表

<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

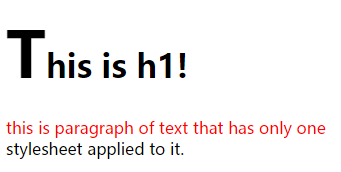
<body>  
 <h1>This is h1!</h1>  
</body>

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>

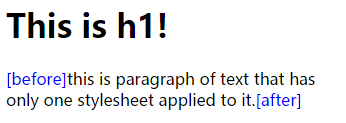
h1:first-letter{  
 font-size: 200%;  
}  
  
p:first-line{  
 color: red;  
}



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

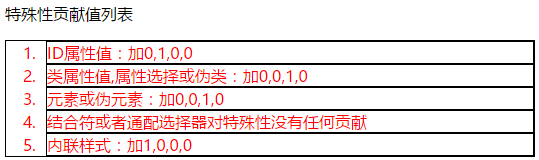
ol的子元素li都继承了ol的样式。当然有些属性是不能继承的，例如border。

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

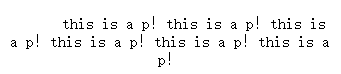
## 6.1 缩进和水平对齐

<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

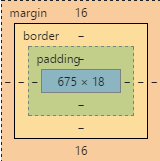
img{  
 vertical-align: middle;  
}



### 6.2.5 长度对齐

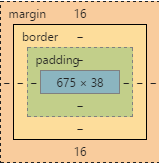
<body>  
 <p>this is p, include   
 <span class="up">  
 up  
 </span>   
 and   
 <span class="down">  
 down  
 </span>  
 </p>  
</body>

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。

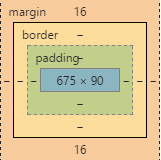
# 第七章 基本视觉格式化

## 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

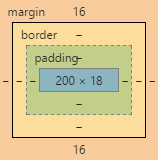
图上：675x90所在矩形框的宽度 = width

### 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;  
}

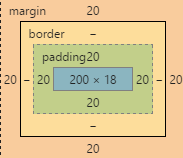




元素框宽度 = 200px

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。

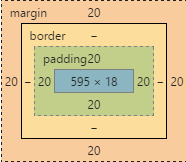
水平属性

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;  
}

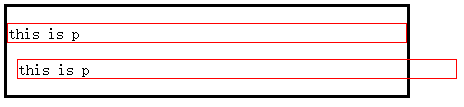


其实p元素相对于黄色边界偏移了40px

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;  
}

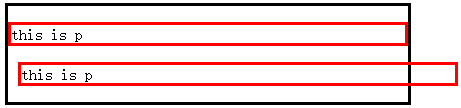


10px + 0 + 0 + 440px + 0 + 0 -50px = 400px（以上的红色边框是为了视图效果需要，实际不存在）

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;  
}



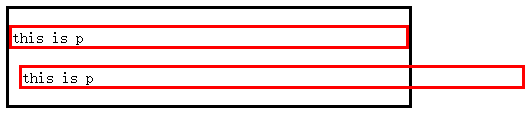
10px + 3px + 0px + width + 0px + 3px – 50px = 400px

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;  
}

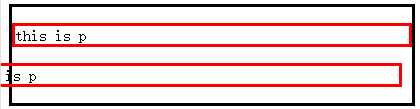


10px +3px + 0px + 500px + 0px + 3px + Margin-right = 400px

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

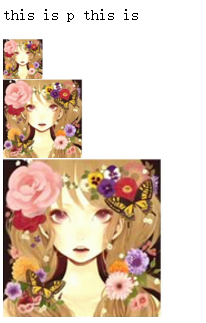
其他规则和非替换元素一样，关键的区别是width为auto，则元素的宽度是内容的固有宽度。

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

垂直7个属性

margin-top border-top padding-top height padding-bottom border-bottom margin-bottom

1.auto高度

三个可以设置auto的值

Margin-top margin-bottom height

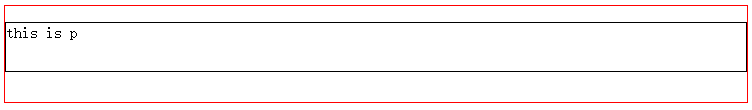
其余都必须有特定的值，如果不设置，默认是0

正常流中的一个块元素的margin-top或margin-bottom设置为auto，它会自动计算为0，如果值为0，那么久不能容易的将正常流元素在其包含块中垂直居中。如果将一个元素的上下边距设置为auto，实际上它们都会重置为0，使元素没有外边距。

Height必须设置为auto或者某种类型的非负值。

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

div{  
 height: 6em;  
 border: 1px solid red;  
}  
  
p{  
 height: 50%;  
 border: 1px solid black;  
}



P元素并没有居中，尽管外边距不设置时为auto，也就是0

div{  
 height: auto;  
 border: 1px solid red;  
}  
  
p{  
 height: 50%;  
 border: 1px solid black;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\793.tmp.png

div{  
 height: auto;  
 background: yellow;  
}  
  
p{  
 border: 1px solid red;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\CD8.tmp.png

如果块级正常流元素设置height：auto，显示时其高度将恰好足以包含其内联内容的行盒。

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

C:\Users\Victor.Zxk\AppData\Local\Temp\EC8.tmp.png

**设置为auto时，并且没有上内边距或下内边距，或者没有上边框或下边框，会在段落上设置一个边框，并认为DIV没有没有内边距，**如果只有块级子元素，其默认高度将是**从最高块级子元素的外边框边界到最低块级子元素的外边框边界（注意不包含外边距，即不包含子元素的margin值）**之间的距离。

div{  
 height: auto;  
 background: yellow;  
}  
  
p{  
 margin-top: 2em;  
 margin-bottom: 2em;  
}



此时DIV高度大致就是段落的高度。

**如果块级元素有上内边距和下内边距，或者有上边框或下边框，其高度则是从最高子元素的上外边距边界到其最低子元素的下外边距边界之间的距离。（意思是包括margin值）**

div{  
 height: auto;  
 background: yellow;  
 border-top: 1px solid red; /\*加上border值以后。DIV高度包含p元素的margin值了\*/  
 border-bottom: 1px solid red;  
}  
  
p{  
 margin-top: 2em;  
 margin-bottom: 2em;  
}



div{  
 height: auto;  
 background: yellow;  
 padding-top: 1px;  
 padding-bottom: 1px;  
}  
  
p{  
 margin-top: 2em;  
 margin-bottom: 2em;  
}



改成内边距还是一样。

div{  
 height: auto;  
 background: yellow;  
 padding-bottom: 1px; /\*如果去掉上内边距\*/  
}  
  
p{  
 margin-top: 2em;  
 margin-bottom: 2em;  
}



那么DIV的height值就不包含margin-top了。

2.合并垂直外边距

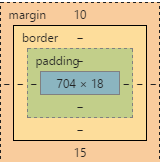
**垂直相邻外边距是可以合并的，这种合并行为只应用于外边距。如果元素有内边距和边框，它们绝对不会合并。**

<body>  
 <ul>  
 <li>1</li>  
 <li>2</li>  
 <li>3</li>  
 <li>4</li>  
 <li>5</li>  
 </ul>  
</body>

ul{  
 background: yellow;  
}  
  
li{  
 margin-top: 10px;  
 margin-bottom: 15px;  
}



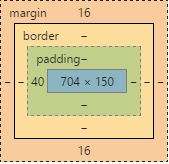
详细分析ul的height



默认li的高度是18

**虽然每个列表的上边距是10px，下边距是15px，但是相邻列表之间的边距会沿着竖轴合并，换句话说，两个外边距中较小的一个会被较大的合并，即虽然上一个li的下边距为10px，但是因为下一个li的上边距为15px，大于上一个li的下边距，所以两个li之间的边距取较大值15px，所以每个li之间的边距是15px**

那么 18\*5 + 15\*4 = 150px, 即ul的height = 150px（如下图所示）

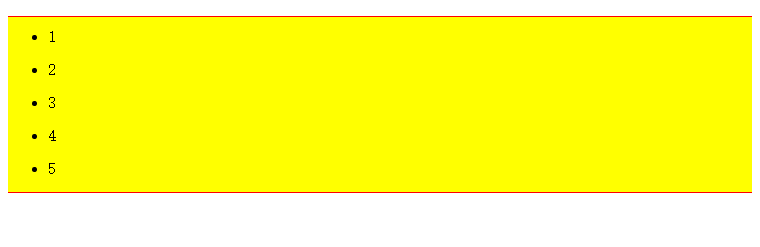


但是为什么是15\*4 而不是**15\*4 （5个li之间的距离，这里并不包括第一个li的上边距和最后一个li的下边距）+ 10px（第一个li的上边距） + 15px（最后一个li的下边距）**呢？

**因为和上一节auto中所说的没有给ul设置外边距或者内边距有关系！**

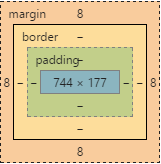
ul{  
 background: yellow;  
 border-top: 1px solid red;  
 border-bottom: 1px solid red;  
}  
  
li{  
 margin-top: 10px;  
 margin-bottom: 15px;  
}

重新给ul设置border值



此时的ul的height值

18px\*5 + 15px\*4 + 10px + 15px + 1px + 1px = 177px（如下图所示）



<body>  
 <ul>  
 <li>1</li>  
 <li>2</li>  
 <li>3</li>  
 <li>4</li>  
 <li class="last">5</li>  
 </ul>  
 <p>this is p</p>  
</body>

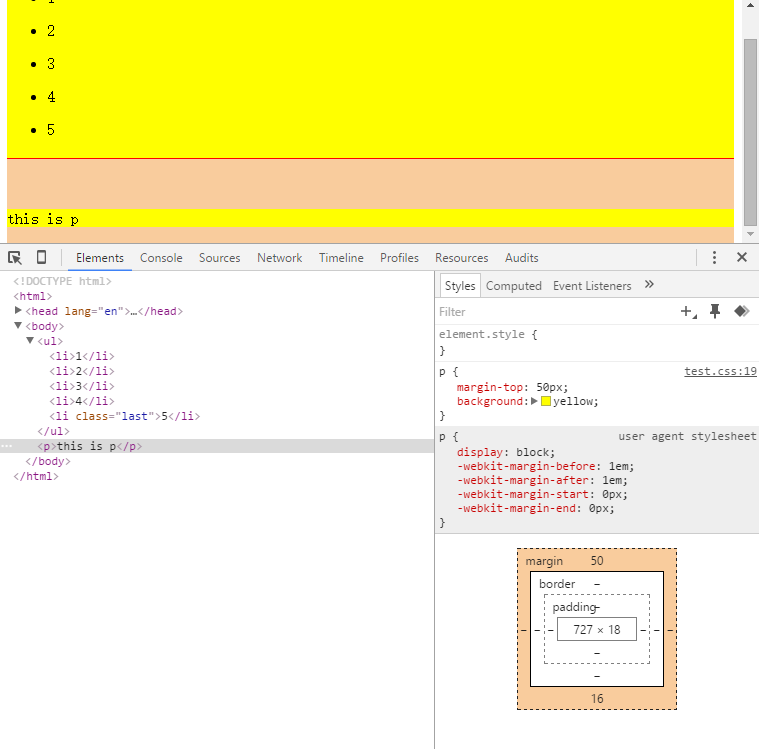
ul{  
 background: yellow;  
 border-top: 1px solid red;  
 border-bottom: 1px solid red;  
 margin-bottom: 30px;  
}  
  
li{  
 margin-top: 10px;  
 margin-bottom: 15px;  
}  
  
li.last{  
 margin-bottom: 20px;  
}  
  
p{  
 margin-top: 50px;  
 background: yellow;  
}

虽然ul的下外边距是30px

但是以为p的上外边距是50px远远大于30px

所以合并后ul和p之间的距离是50px





可以清楚的看到填充的区域是50px

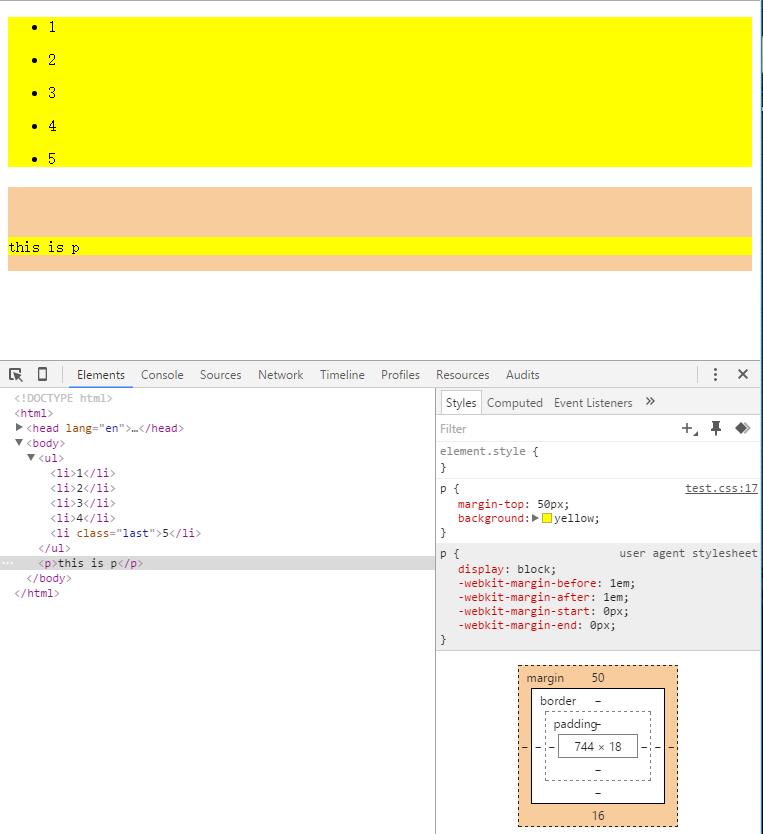
**需要强调的一点是：**

**在包含块上设置边框或内边距时，会使子元素的外边距包含在包含块（也就是父元素）之内。**

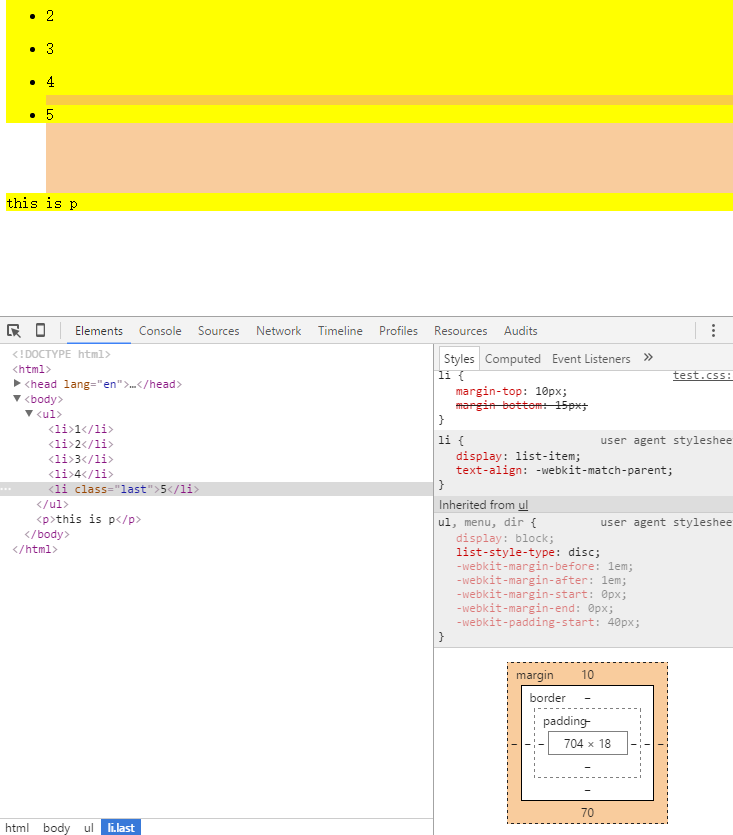
所以以上这个例子最后一个li的下边距虽然是20px，但是这个20px是包含在ul之内的，和ul和p之间的外边距合并是没有任何关系的。

如果去掉ul的外边距

ul{  
 background: yellow;  
 margin-bottom: 30px;/\*这里去掉了ul的外边距\*/  
}  
  
li{  
 margin-top: 10px;  
 margin-bottom: 15px;  
}  
  
li.last{  
 margin-bottom: 70px;  
}  
  
p{  
 margin-top: 50px;  
 background: yellow;  
}



此时ul和p之间的距离就不是50px了。



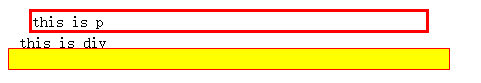
可以明显看出此时的ul和p之间的距离是70px，这个距离是ul li.last和p三者之间外边距合并的结果，取最大值70px！

3.负外边距

负外边距合并：如果两个设置为负值，会取两个外边距绝对值的最大值，如果一个正外边距与一个负外边距合并，会从正外边距减去这个负外边距的绝对值。

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

div{  
 width: 420px;  
 background: yellow;  
 padding: 10px;  
 margin-top: 50px;  
 border: 1px solid red;  
}  
  
p{  
 margin-top: -50px;  
 margin-right: 10px;  
 margin-left: 10px;  
 margin-bottom: 0;  
 border: 3px solid red;  
}



div{  
 width: 420px;  
 margin-top: 50px;  
}  
  
p{  
 margin-top: 0;  
 margin-right: 10px;  
 margin-left: 10px;  
 margin-bottom: -50px;  
 border: 3px solid red;  
}



1.正常情况

li{  
 margin-bottom: 20px;  
}  
  
ul{  
 background: yellow;  
}  
h1{  
 background: yellow;  
}

由于ul没有使用外边距或者内边距，根据合并规则，h1和ul之间的距离是20px



2.如果一边距是正，一边距是负

li{  
 margin-bottom: 20px;  
}  
  
ul{  
 margin-bottom: -30px;  
 background: yellow;  
}  
  
h1{  
 background: pink;  
}

由于合并规则，如果是一个正外边距与一个负外边距合并，会从正外边距减去这个负外边距的绝对值。

20px – 30px = -10px 所以h1和ul的边距是-10px，导致两者重叠，需要注意的是ul位置不变，和h1往上移



并且为什么是粉红色覆盖黄色呢，因为浏览器按从前到后的顺序显示元素，所以文档中后出现的正常流元素可能会覆盖较早出现的元素。

li{  
 margin-bottom: 20px;  
}  
  
ul{  
 margin-bottom: -10px;  
 background: yellow;  
}  
  
h1{  
 background: pink;  
}



当然此种情况下的两者的距离是20px-10px = 10px

<body>  
 <ul>  
 <li>1</li>  
 <li class="last">2</li>  
 </ul>  
 <h1>this is h1</h1>  
</body>

li.last{  
 margin-bottom: 30px;  
}  
  
ul{  
 margin-bottom: -30px;  
 background: yellow;  
}  
  
h1{  
 background: pink;  
}



30px -30px = 0px 所以正好紧挨着

li.last{  
 margin-bottom: 20px;  
}  
  
ul{  
 margin-bottom: -15px;  
 background: yellow;  
}  
  
h1{  
 background: pink;  
}

明显的，两者相差5px



3.如果两者都是负边距

li.last{  
 margin-bottom: 20px;  
}  
  
ul{  
 margin-bottom: -15px;  
 background: yellow;  
}  
  
h1{  
 margin-top: -18px;  
 background: pink;  
}

比如ul和h1都是负边距，那么取两者绝对值的最大值18px，然后和li进行合并

20px – 18px = 2px



其实是有2px的距离，和上面紧挨着的可以比较一下。

4.列表项

C:\Users\Victor.Zxk\AppData\Local\Temp\47EA.tmp.png

列表项前面通常有一个标志，一个圆点或一个数字，这个标志实际上并不是列表项内容区的一部分。

<body>  
 <ul class="first">  
 <li>1</li>  
 <li class="last">2</li>  
 </ul>  
 <ul class="last">  
 <li>1</li>  
 <li class="last">2</li>  
 </ul>  
</body>

ul.first li{  
 border: 1px solid red;  
}  
  
ul.last {  
 list-style-position: inside;  
}  
  
ul.last li{  
 border: 1px solid red;  
}



如果标志放在内部，那么这个列表项相对于其相邻列表项就像一个块级元素一样。

## 7.3 行内元素

### 7.3.1 基本术语和概念

1.匿名文本，所有未包含在行内元素中的字符串

<p>this is <em> em </em> p</p>

序列“this is ”和“ p”都是匿名文本，这里需要注意的是包括空格和其他字符。

2.em框 也成为字符框

3.内容区

在非替换元素中，内容区是元素中各字符的em框串在一起构成的框，在替换元素中，内容区就是元素的固有高度再加上可能有的外边距、边框或内边距。

4.行间距

行间距 = (ling-height) – (font-szie)

实际上还要分为两半，分别应用到内容区的顶部和底部。

行间距只用于非替换元素。

5.行内框

对于非替换元素，行内框的高度 = line-height

对于替换元素，行内框高度 = 替换元素内容区的高度

6.行框

包含该行中出现的行内框的最高点和最低点的最小框。

有用概念

* 1. 内容区类似于一个块级元素的内容框
  2. 行内元素的背景应用于内容区及所有的内边距

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

em{  
 background: yellow;  
}

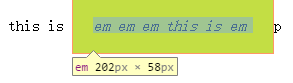
this is em em em this is emp

* 1. 行内元素的边框要包围内容区及所有内边距和边框

em{  
 background: yellow;  
 padding: 20px;  
 border: 1px solid red;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\6452.tmp.png

* 1. 非替换元素的内边距、边框和外边距对行内元素或其生成的框没有垂直效果，也就是说，它们不会影响元素行内框的高度（也不会影响包含该元素的行框的高度）。



em本身行内框的高度，也就是line-height大小不会因为padding等变化。包括它的父元素p元素也不会变化。

em{  
 background: yellow;  
 padding: 20px;  
 border: 1px solid red;  
}  
  
p{  
 margin: 2px;  
}



父元素p并不会因为行内元素变化而变化。

* 1. 替换元素的外边距和边框确实会影响该元素行内框的高度，相应的，也可能影响包含该元素的行框的高度。



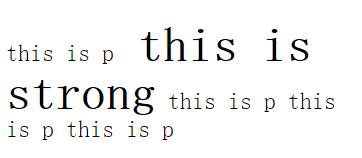
P元素因为替换元素的行内框高度的变化而变化了！

### 7.3.2 行内非替换元素格式化



line-height 行内框 18px

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



这样的效果使得行内框的高度和内容区的高度一致

因为1em是相对于本元素的font-size而言的，即line-height = font-size

p{  
 font-size: 24px;  
 line-height: 1;  
}  
  
strong{  
 font-size: 200%;  
}

这种设置也是和上面一样效果



Line-height作为缩放因子被继承，所以strong和p的line-height的值都为1，因此行内框的高度和内容区的高度一致。

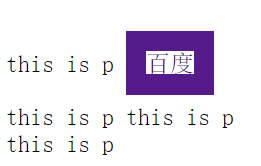
<body>  
 <p>  
 this is p <a href="http://www.baidu.com">百度</a>  
 this is p this is p this is p  
 </p>  
</body>

p{  
 font-size: 24px;  
}  
  
a:link{  
 border: 20px solid blue;  
}



由于没有给超链接设置足够大的line-height来容纳边框，导致覆盖了其他行的内容。可以使用line-height来增加未访问连接的行内框大小。

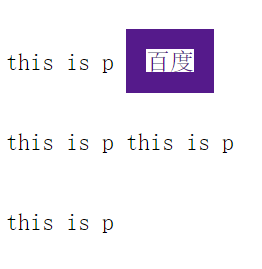
p{  
 font-size: 24px;  
}  
  
a:link{  
 border: 20px solid blue;  
 line-height: 80px;  
}



此时只是增加了超链接所在行的行内框大小。

当然也可以增加段落的line-height。

p{  
 font-size: 24px;  
 line-height: 80px;  
}  
  
a:link{  
 border: 20px solid blue;  
}



Line-height也可以缩放行高

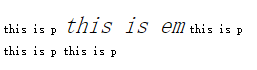
body{  
 line-height: 1.5;  
}  
  
  
p{  
 font-size: 12px;  
}  
  
em{  
 font-size: 24px;  
}

em和p继承了body的缩放因子

body{  
 line-height: 1;  
}  
  
  
p{  
 font-size: 12px;  
}  
  
em{  
 font-size: 24px;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\1EAB.tmp.png

body{  
 line-height: 1.5;  
}  
  
p{  
 font-size: 12px;  
}  
  
em{  
 font-size: 24px;  
}



继承了body的缩放因子。

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

p span{  
 background: red;  
 padding-top: 10px;  
 padding-bottom: 10px;  
}

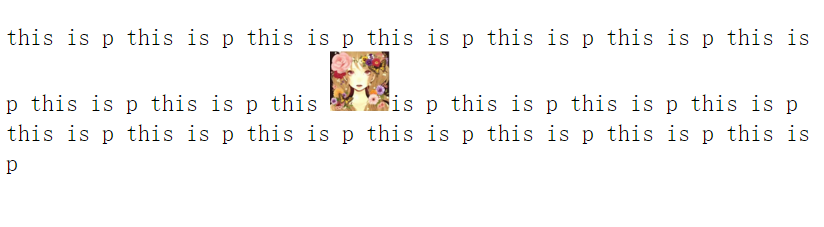


可以看到了重叠，并且覆盖了前一行的字体，后一行的却不会覆盖。根据浏览器从上至下的渲染顺序。

### 7.3.3 行内替换元素格式化

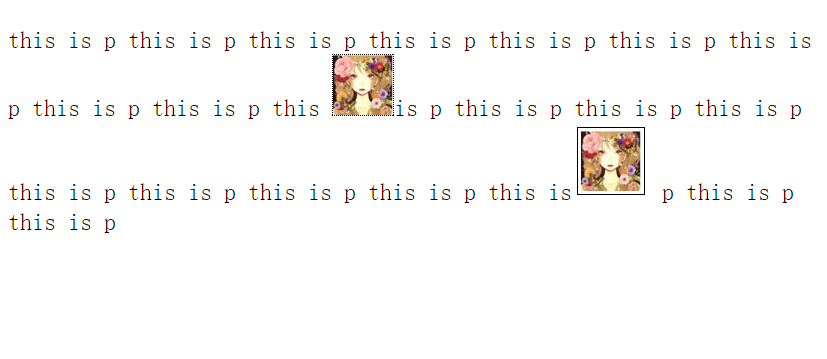
<body>  
 <p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this <img src="6E22.tmp.png">is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 </p>  
</body>

p{  
 font-size: 24px;  
 line-height: 30px;  
}  
  
img{  
 height: 60px;  
 margin: 0;  
 padding: 0;  
 border: none;  
}



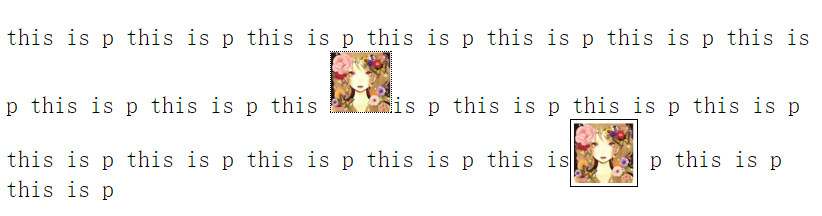
替换元素可以增加行框的高度但是不会影响line-height的值。尽管看起来图像所在的行有空白，但是段落或图像本身的line-height有效值并没有因此改变。

p{  
 font-size: 24px;  
 line-height: 30px;  
}  
  
img{  
 height: 60px;  
}  
  
img.first{  
 border: 1px dotted;  
}  
  
img.last{  
 margin: 5px;  
 padding: 3px;  
 border: 1px solid;  
}



为行内替换元素增加内边距边框和外边距会增大其行内框。

p{  
 font-size: 24px;  
 line-height: 30px;  
}  
  
img{  
 height: 60px;  
}  
  
img.first{  
 border: 1px dotted;  
}  
  
img.last{  
 padding: 3px;  
 border: 1px solid;  
 margin-bottom: -20px;  
}



## 7.4 改变元素显示

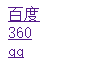
### 7.4.1 改变角色

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

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

默认以行内元素的方式排布

a{  
 display: block;  
}



使其以块级元素的方式显示，好处也很明显，整个元素框都会成为连接的一部分，哪怕你鼠标停留在百度后面的空白区域，也因为它是块级区域的一部分，所以也是可以链接出去的。

li{  
 display: inline;  
 border-right: 1px solid;  
 padding: 0 0.33em;  
}  
  
li:first-child{  
 border-left: 1px solid;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\4F80.tmp.png

使块级元素以行内元素的方式显示，但是它仍然是块级元素。

行内元素可能是块级元素的后代，但是反过来不行！

Display只能影响的是元素如何显示，而不会影响元素的本质。

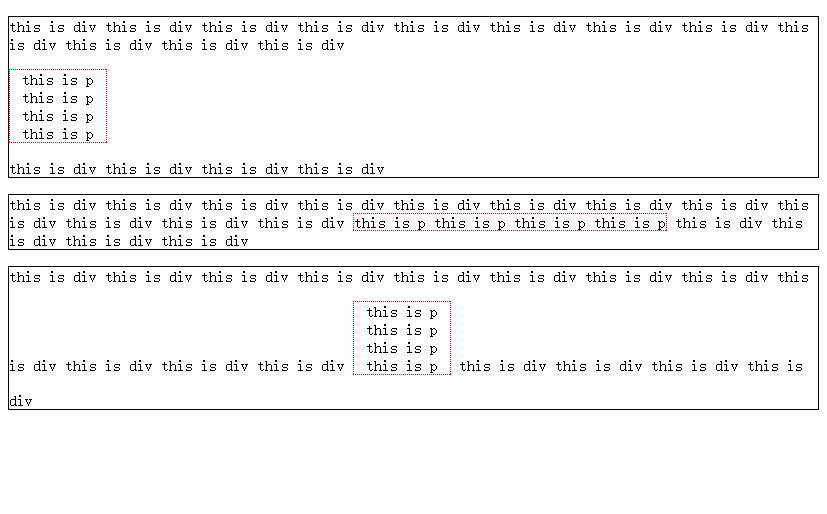
### 7.4.2 行内块元素

行内块元素有点类似于图像放在一个文本行中，行内块元素会作为替换元素放在行中。

行内块元素内部，会像块级元素一样设置内容的样式。

<body>  
 <div class="one">this is div this is div this is div this is div  
 this is div this is div this is div this is div  
 this is div this is div this is div this is div  
 <p>this is p this is p this is p this is p</p>  
 this is div this is div this is div this is div  
 </div>  
  
 <div class="two">this is div this is div this is div this is div  
 this is div this is div this is div this is div  
 this is div this is div this is div this is div  
 <p>this is p this is p this is p this is p</p>  
 this is div this is div this is div this is div  
 </div>  
  
 <div class="three">this is div this is div this is div this is div  
 this is div this is div this is div this is div  
 this is div this is div this is div this is div  
 <p>this is p this is p this is p this is p</p>  
 this is div this is div this is div this is div  
 </div>  
</body>

div{  
 border: 1px solid black;  
 margin: 1em 0;  
}  
  
p{  
 border: 1px dotted red;  
}  
  
div.one p{  
 display: block;  
 width: 6em;  
 text-align: center;  
}  
  
  
div.two p{  
 display: inline;  
 width: 6em;  
 text-align: center;  
}  
  
div.three p{  
 display: inline-block;  
 width: 6em;  
 text-align: center;  
}



第二个div中的行内段落格式化为正常的行内内容，这说明width和text-align被忽略了(它们不能应用于行内元素)，对于第三个div元素，作为行内块元素的段落则有这两个属性，因为它作为一个块级元素被格式化。

### 7.4.3 run-in元素

可以使某些块级元素成为下一个元素的行内部分，对于某些标题效果很好。

# 第八章 内边距、边框和外边距

## 8.1 基本元素框

1.宽度和高度

Width和height不能用于行内非替换元素，只能应用于块级元素和替换元素。

IE6之前的版本对widht和height两个属性的设定有BUG。例如这两个属性可以应用于行内非替换元素等。

2.外边距和内边距

背景会延伸到内边距中，但是不会延伸到外边距中。

margin：top right bottom left

顺序是从上开始顺时针旋转

如果设置百分数值，百分数是相对于父元素的width计算的。所以如果父元素的widht以某种方式发生改变，百分数也会改变。

Margin设置规则

* 1. 如果只设定一个值，那么四个值都是同一个值
  2. 如果设定了两个值，那么下边距复制上边距的值，左边距复制右边距的值（根据顺序可知如果设置两个值，肯定是上边距和右边距）
  3. 如果设定三个值，那么最后一个值左边距复制右边距的值

例如：margin：10px = margin：10px 10px 10px 10px

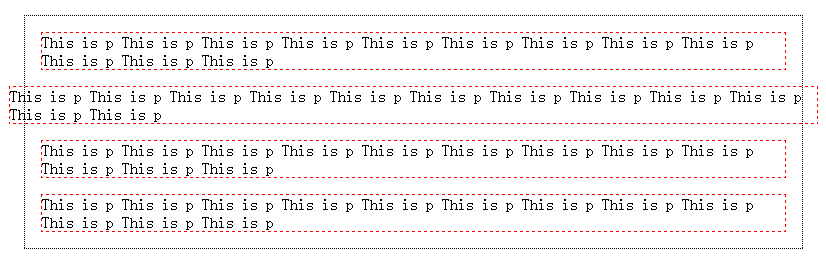
例如：margin：10px 20px = margin：10px 20px 10px 20px

例如： margin：10px 20px 30px = margin：10px 20px 30px 20px

## 8.2 外边距（margin）

<div>  
 <p>  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 </p>  
 <p class="one">  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 </p>  
 <p class="two">  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 </p>  
 <p>  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 This is p This is p This is p This is p  
 </p>  
</div>

div{  
 border: 1px dotted black;  
 margin: 1em;  
}  
  
p{  
 margin: 1em;  
 border: 1px dashed red;  
}  
  
p.one{  
 margin: 0 -1em;  
}

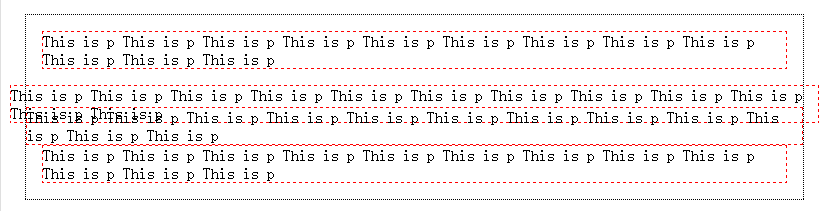


-1em + 0 + 1px + width + 1px + 0 – 1em = div.width

所以 width = div.width + 2em – 2px

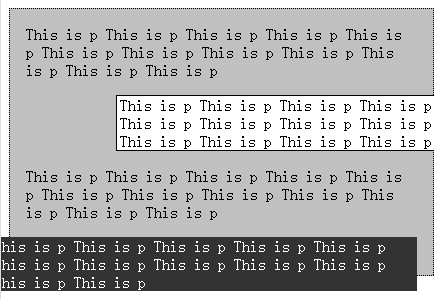
所以第二个p元素比父元素宽

div{  
 border: 1px dotted black;  
 margin: 1em;  
}  
  
p{  
 margin: 1em;  
 border: 1px dashed red;  
}  
  
p.one{  
 margin: 0 -1em;  
}  
  
p.two{  
 margin: -1em 0;  
}



上下负边距导致第三个p元素和第二个以及第四个p元素重叠在一起了。

div{  
 border: 1px dotted black;  
 background: silver;  
}  
  
p{  
 margin: 1em;  
}  
  
p.one{  
 background: white;  
 margin: 1em -1px 1em 25%;  
 border: 1px solid;  
 border-right: none;  
 text-align: center;  
}  
  
p.last{  
 background: #333;  
 color: white;  
 margin: 1em 1em -1em -1em;  
}



外边距和行内元素

strong{  
 margin-top: 25px;  
 margin-bottom: 25px; /\*行内非替换元素无效\*/  
 margin-left: 25px; /\*有效\*/  
 border: 1px dashed;   
}

This is p This is p  **this is strong**This is p This is p

strong{  
 margin: 25px;  
 background: silver;  
}

This is p This is p  **this is strong**  This is p This is p

strong{  
 margin: -50px;  
 background: silver;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\7E4F.tmp.png

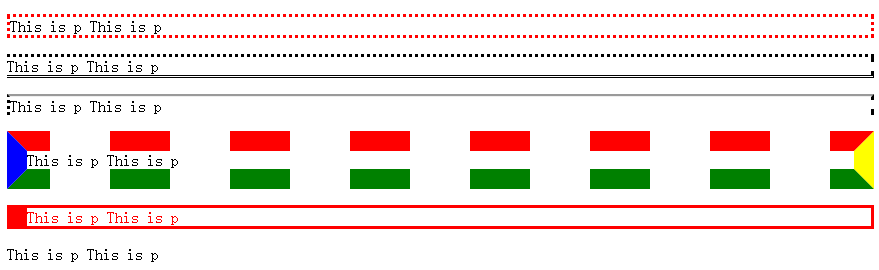
## 8.3 边框（border）

边框的默认颜色是元素本身的前景色。如果元素没有内容，那么color可能是父元素的颜色，因为color是可以继承的。

背景是内容、内边距和边框区的背景。

<body>  
 <p>This is p This is p</p>  
 <p class="two">This is p This is p</p>  
 <p class="three">This is p This is p</p>  
 <p class="four">This is p This is p</p>  
 <p class="five">This is p This is p</p>  
 <p class="six">This is p This is p</p>  
</body>

p:first-child{  
 border-style: dotted;  
 border-color: red;  
}  
  
p.two{ /\*和margin方向一致\*/  
 border-style: dotted dashed double none;  
  
}  
  
p.three{  
 border-left-style: dotted ;  
 border-right-style: dashed;  
 border-top-style: groove;  
}  
  
p.four{  
 border-style: dashed;  
 border-color: red yellow green blue;  
 border-width: 20px;  
}  
  
p.five{  
 color: red;  
 border-style: solid;  
 border-left-width: 20px;/\*可以发现边框和元素的颜色是一样的\*/  
}  
  
p.six{  
 border-width: 20px; /\*没有效果\*/  
 border-style: none;  
}



透明边框

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

a:link,  
a:visited{  
 border-style: solid;  
 border-width: 5px;  
 border-color: transparent; /\*透明色\*/  
}  
  
a:hover{  
 border-color:green ;  
}



在IE7之前并没有提供透明色。

简写边框

a:link,  
a:visited{  
 border-bottom: thick solid green;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\250D.tmp.png

a:link,  
a:visited{  
 border-bottom: thick solid green;  
}  
  
a:link,  
a:visited{  
 border-bottom: thick green; /\*样式重叠，没有任何效果\*/  
}

[百度](http://www.baidu.com/) [360](http://www.baidu.com/) [谷歌](http://www.baidu.com/) [阿里](http://www.baidu.com/)  
边框和行内元素

strong{  
 border-top: 10px solid green;  
 border-bottom: 10px solid green;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\2F99.tmp.png

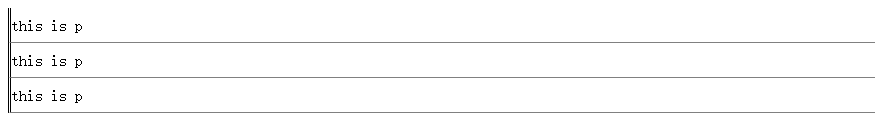
元素的行高都不会改变的。

strong{  
 border-top: 10px solid green;  
 border-bottom: 10px solid green;  
 border-left: 10px solid red;  
}

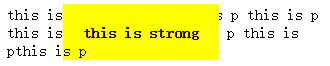
C:\Users\Victor.Zxk\AppData\Local\Temp\3272.tmp.png

## 8.4 内边距（padding）

p{  
 margin: 0;  
 padding: 0.5em 0;  
 border-bottom: 1px solid gray;  
 border-left: 3px double black;  
}



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



行高没有改变，不过由于内边距延伸了背景。

p{  
 margin: 0;  
}  
  
img{  
 padding: 30px;  
 background: yellow;  
}



# 第九章 颜色和背景

## 9.1 前景色

<body>  
 <a href="http://www.baidu.com">百度</a>  
 <a class="warning" href="http://www.baidu.com">百度</a>  
 <a class="success" href="http://www.baidu.com">百度</a>  
</body>

a:link{  
 color: red;  
 }  
  
a:hover{  
 color:green;  
}  
  
a.warning:hover{ /\*拥有warning类的锚覆盖了通用的锚的样式\*/  
 color: pink;  
}

默认的前景色会应用到边框

a{  
 color: red;  
 border-style: solid;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\17FF.tmp.png

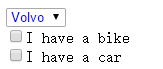
覆盖默认的前景色

a{  
 color: red; /\*前景色\*/  
 border-style: solid;  
 border-color: green;  
}

当然也可以为图像设置边框色

影响表单元素

<select name="cars">  
 <option value="volvo">Volvo</option>  
 <option value="saab">Saab</option>  
 <option value="fiat">Fiat</option>  
 <option value="audi">Audi</option>  
</select>  
<br/>  
  
<input type="checkbox" name="vehicle" value="Bike">I have a bike  
<br/>  
<input type="checkbox" name="vehicle" value="Car">I have a car



继承颜色

## 9.2 背景

元素的背景区包括前景之下直到边框外边界的所有空间，因此内容框和内边距都是元素背景的一部分，且边框画在背景之上。

背景色

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

p{  
 background-color: yellow;  
 padding: 10px;  
}

C:\Users\Victor.Zxk\AppData\Local\Temp\28AF.tmp.png

背景色是不能被继承的，em的背景色是默认的透明色，这样它的祖先元素的背景才能可见。

所以em元素显示的背景色是p元素的背景色。

**需要强调的是背景色是不能被继承的！**

背景图像

允许向任何元素应用背景图像，可以是块级元素也可以是行内元素。

所有的背景属性都是不能继承的！

使用背景图像的时候最好再指定一个背景色。

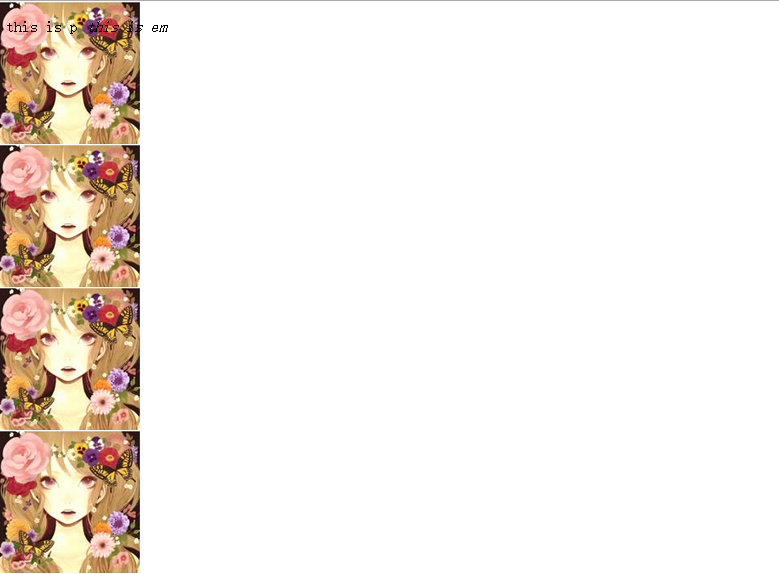
<body>  
 <p>this is p<em> this is em</em></p>  
</body>

body{  
 background-image: url(6E22.tmp.png);  
 background-color: aqua;  
}

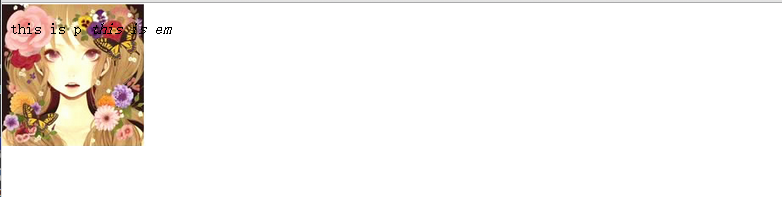


如果出于某种原因无法加载图像，这就会填入一个背景色。

body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: repeat-y;  
}



body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
}



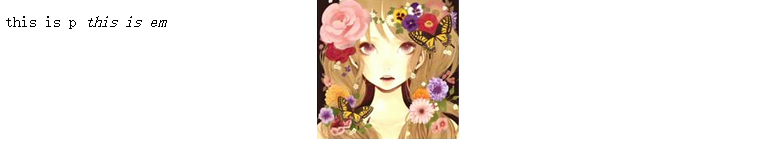
背景定位

body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: center;  
}

body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: top right;  
}



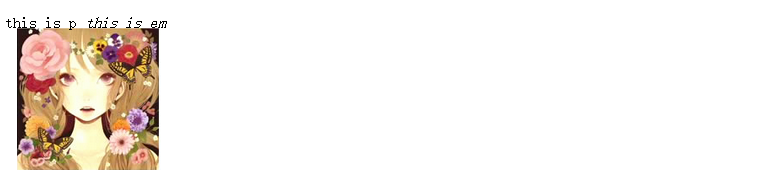
body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: top;/\*如果省略后面的，就表明是center\*/  
}



body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: 100% 0%;  
}



body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: 20px 30px;  
}

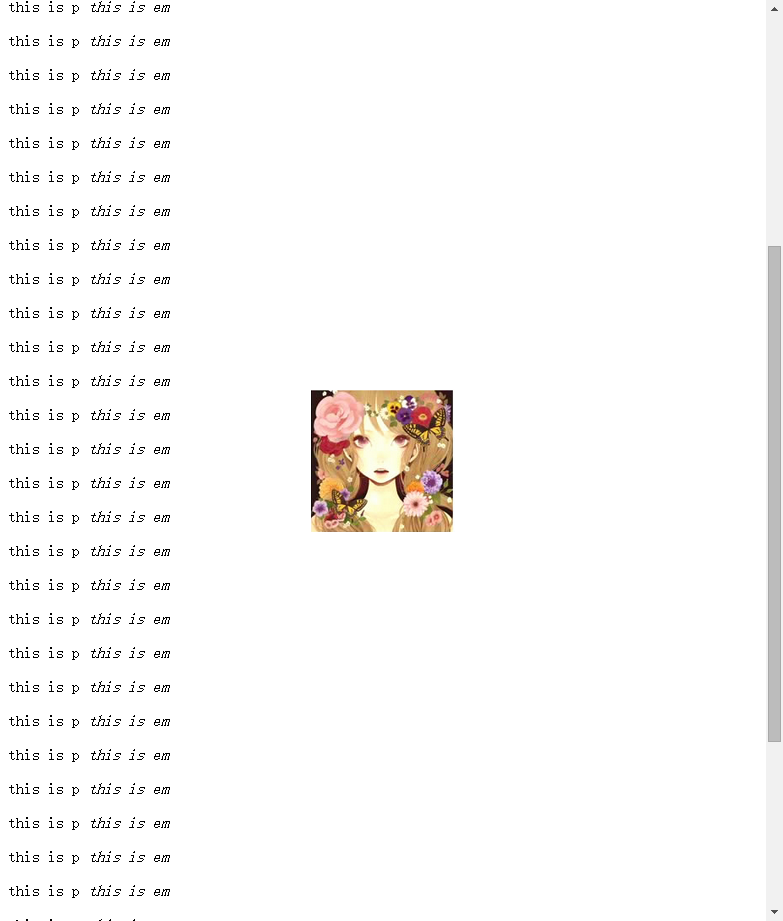


body{  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: -50px -50px;  
}



拉出了元素的背景区。

body {  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: center;  
 background-attachment: fixed;/\*图像位置不受滚动影响\*/  
}





一直保持居中！

body {  
 background-image: url(6E22.tmp.png);  
 background-repeat: no-repeat;  
 background-position: center;  
 background-attachment: scroll;/\*图像位置不受滚动影响\*/  
}





随着滚动条变化！

汇总

body{  
 background: yellow url(6E22.tmp.png) top right repeat-y fixed;  
}



body{  
 background: yellow ;/\*填入至少一个词就可以\*/  
}



body{  
 background: yellow url(6E22.tmp.png) top right repeat-y fixed;  
}  
  
  
body{  
 background: yellow;/\*覆盖了前面的样式\*/  
}



body{  
 background: yellow url(6E22.tmp.png) top right repeat-y fixed;  
}  
  
  
body{  
 background-color: red;/\*覆盖了前面的背景色样式\*/  
}



# 第十章 浮动和定位

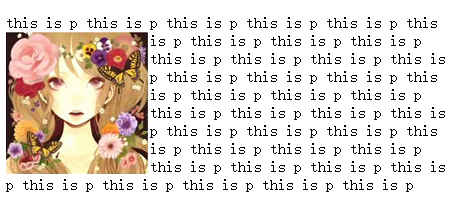
## 10.1 浮动

浮动是非正常流布局。

未设置任何样式之前



img{  
 float: left;  
}



文本则围绕着该图像。

### 10.1.1 浮动元素

1.会以某种方式将浮动元素从文档的正常流中删除，还是会影响布局。

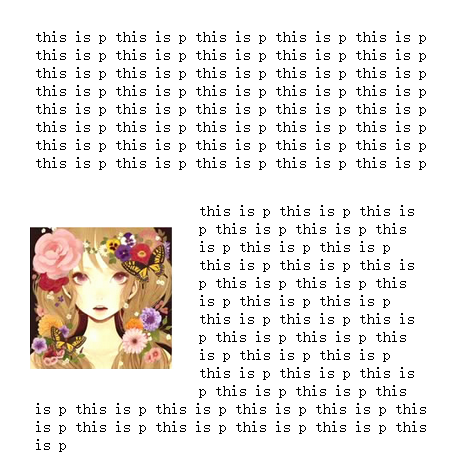
2.浮动元素几乎‘自成一派‘，不过它们还是对文档的其余部分有影响。（一个元素浮动时，其他内容会“环绕”该元素）。

3.浮动元素周围的外边距不会合并。

<body>  
 <p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 </p>  
 <img src="6E22.tmp.png">  
 <p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 </p>

img{  
 float: left;  
 margin: 25px;  
}





可以明显的发现，外边距并没有合并使得img和p之间的外边距是30px，仍然是25px。

### 10.1.2 浮动的详细内幕

深入讨论浮动详细内容之前，首先要建立包含块的概念。

浮动元素的包含块是其最近的块级祖先元素。

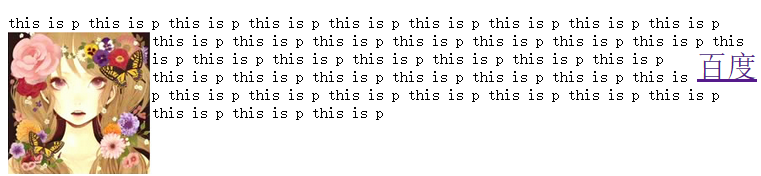
<body>  
 <p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 <img src="6E22.tmp.png">  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 </p>  
</body>

img {  
 float: left;  
}

如上所示，浮动元素img的包含块就是包含该浮动元素的p元素。

此外，浮动元素会生成一个块级框，而不管元素本身是什么。

img {  
 float: left;  
}  
  
a{  
 float: right;  
 font-size: 30px;  
}

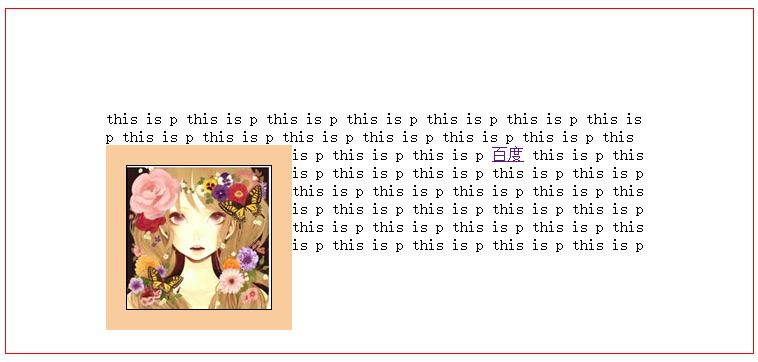


即使该链接元素本身是行内元素，但是因为浮动了，所以会生成一个行内框。它会像块级元素一样摆放和显示。（就好像一个DIV元素），所以没有必要为浮动元素声明display:block。

浮动元素的规则

1. **浮动元素的左（右）外边界不能超出其包含块的左（右）内边界。**

p{  
 border: 1px solid red;  
 padding: 100px;  
}  
img {  
 margin: 20px;  
 border: 1px solid black;  
 float: left;  
}

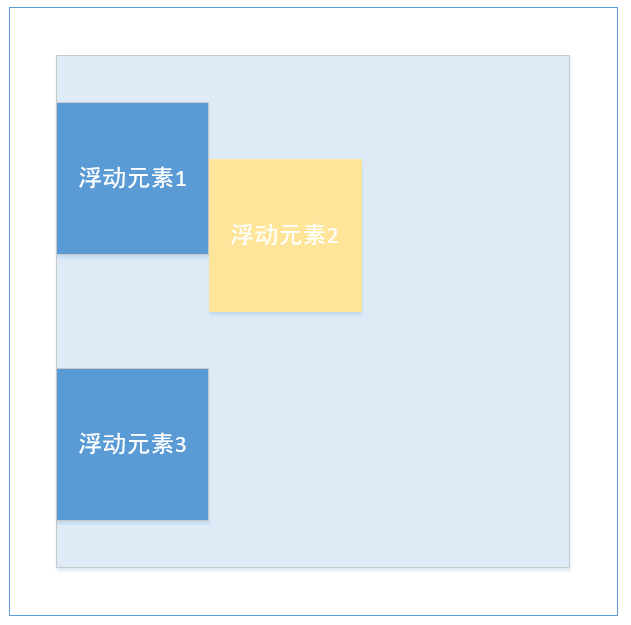






从以上的调试结果可知，图像的外边界不能超出其包含块的左右内边界，但是超出了其上下的内边界。

1. **浮动元素的左（右）外边界必须是源文档中（父元素）之前出现的左浮动（右浮动）元素的右（左）外边界，除非后出现的元素浮动元素的顶端在先出现浮动元素的底端下面。**

****

例如浮动元素2的左外边界是源文档中之前出现的浮动元素1的右外边界。但是浮动元素3因为顶端位于浮动元素1和2的下面，所以仍然是往左左面的父元素的边界浮动。

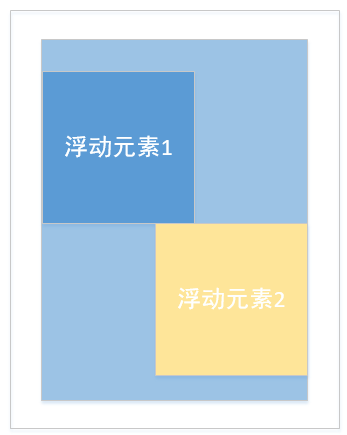
<p>  
 <img class="first" src="6E22.tmp.png">  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 <img class="last" src="6E22.tmp.png">  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p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

p{  
 border: 1px solid red;  
 padding: 200px;  
}  
img {  
 float: left;  
}



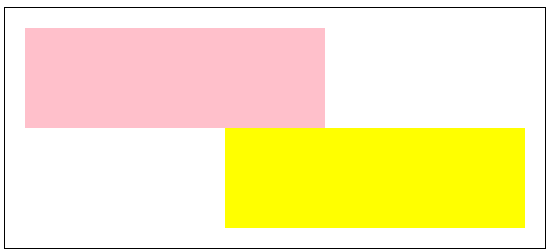
这个规则的好处是防止浮动元素的重叠。使用定位时很容易导致元素的相互覆盖。

1. **左浮动元素的右外边界不会在其右边右浮动元素的左外边界的右边。一个右浮动元素的左外边界不会在其左边任何左浮动元素的右外边界的左边。**



<body>  
 <div class="contain">  
 <div class="left"></div>  
 <div class="right"></div>  
 <div class="clear"></div>  
 </div>  
</body>

div.contain{  
 width: 500px;  
 padding: 20px;  
 border: 1px solid black;  
}  
  
div.left{  
 float: left;  
 background: pink;  
 width: 300px;  
 height: 100px;  
}  
  
  
div.right{  
 float: right;  
 background: yellow;  
 width: 300px;  
 height: 100px;  
}  
  
div.clear{  
 clear: both;  
}

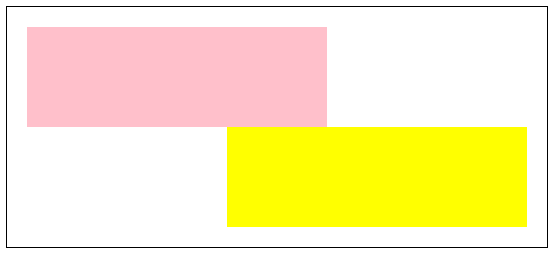


这条规则也很清晰，父元素的500px宽度，有两个300px宽度的DIV，第一个浮动左边，第二个浮动右边，这可以防止第二个图像与第一个图像有重叠，不会变成如下的情况，实际上它会要求第二个DIV向下

浮动。

1. **一个浮动元素的顶端不能比其父元素的内顶端高。**

div.contain{  
 width: 500px;  
 padding: 20px;  
 border: 1px solid black;  
}  
  
div.left{  
 background: pink;  
 width: 300px;  
 height: 100px;  
}  
  
div.right{  
 float: right;  
 background: yellow;  
 width: 300px;  
 height: 100px;  
}  
  
div.clear{  
 clear: both;  
}

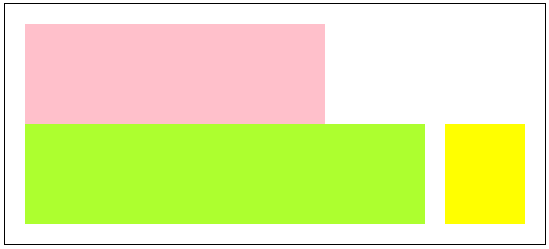


虽然粉红色DIV没有浮动，但是黄色的DIV并不会继续向上浮动到容器DIV的顶端或者更高。

1. **一个浮动元素的顶端不能比之前所有浮动元素或块级元素的顶端更高。**

<body>  
 <div class="contain">  
 <div class="left"></div>  
 <div class="center"></div>  
 <div class="right"></div>  
 <div class="clear"></div>  
 </div>  
</body>

div.contain{  
 width: 500px;  
 padding: 20px;  
 border: 1px solid black;  
}  
  
div.left{  
 float: left;  
 background: pink;  
 width: 300px;  
 height: 100px;  
}  
  
div.center{  
 float: left;  
 background: greenyellow;  
 width: 400px;  
 height: 100px;  
}  
div.right{  
 float: right;  
 background: yellow;  
 width: 80px;  
 height: 100px;  
}  
  
div.clear{  
 clear: both;  
}



黄色浮动元素不如绿色浮动元素出现的早，所以黄色浮动元素不可能比绿色浮动元素的顶端更高。当然也不可能比粉红色浮动元素的顶端更高。

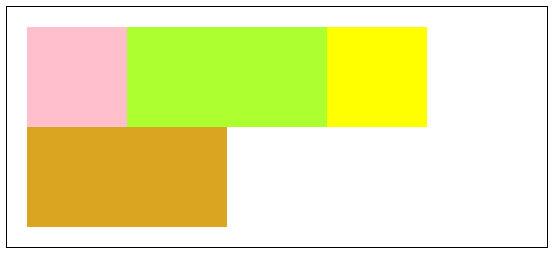
1. **源文档的一个浮动元素之前出现的另一个元素，浮动元素的顶端不能比包含该元素所生成框的任何行框的顶端更高。**
2. **左（右）浮动元素的左边（右边）有另一个浮动元素，前者的右外边界不能在其包含块的右（左）边界的右边（左边）。**

****

意思就是如果父元素没有足够的空间，浮动元素会被挤到一个新的‘行‘上。

<body>  
 <div class="contain">  
 <div class="left"></div>  
 <div class="center"></div>  
 <div class="right"></div>  
 <div class="bottom"></div>  
 <div class="clear"></div>  
 </div>  
</body>

div.contain{  
 width: 500px;  
 padding: 20px;  
 border: 1px solid black;  
}  
  
div.left{  
 float: left;  
 background: pink;  
 width: 100px;  
 height: 100px;  
}  
  
div.center{  
 float: left;  
 background: greenyellow;  
 width: 200px;  
 height: 100px;  
}  
div.right{  
 float: left;  
 background: yellow;  
 width: 100px;  
 height: 100px;  
}  
  
div.bottom{  
 float: left;  
 background: goldenrod;  
 width: 200px;  
 height: 100px;  
}  
  
div.clear{  
 clear: both;  
}



由于第一‘行‘放不下最后一个DIV框，所以它往下面一‘行’挤了。

1. **浮动元素尽可能高的放置。**

满足前面七个条件的基础上，浮动元素的顶端应当与其标记所在的行框的顶端对齐。

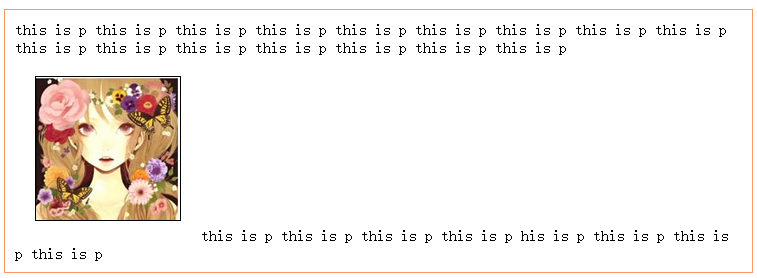
1. **左浮动元素想左浮动尽可能的远，右浮动元素必须向右尽可能远。**

### 10.1.3 实用行为

前面所讲的规则，只处理了浮动元素和父元素的左、右和上边界，而没有涉及下边界。

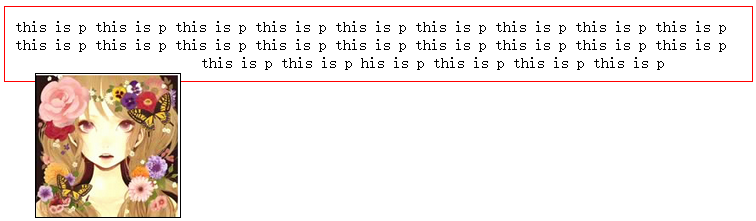
<p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 <img src="6E22.tmp.png">this is p this is p this is p this is p  
 his is p this is p this is p this is p  
</p>

p{  
 border: 1px solid red;  
 padding: 10px;  
}  
img {  
 margin: 20px;  
 border: 1px solid black;  
}



没有浮动的时候。P元素包含了img图像元素。

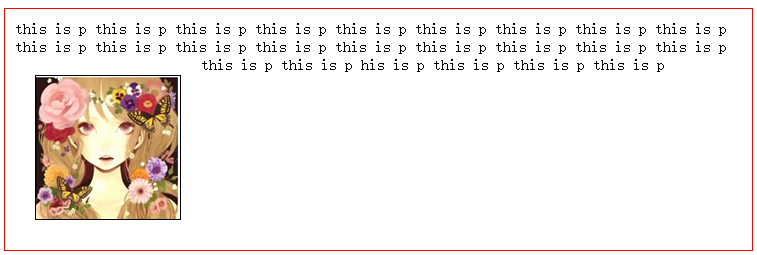
p{  
 border: 1px solid red;  
 padding: 10px;  
}  
img {  
 margin: 20px;  
 border: 1px solid black;  
 float: left;  
}



从上图也可以看出，浮动图像超出了其父元素p的底端。

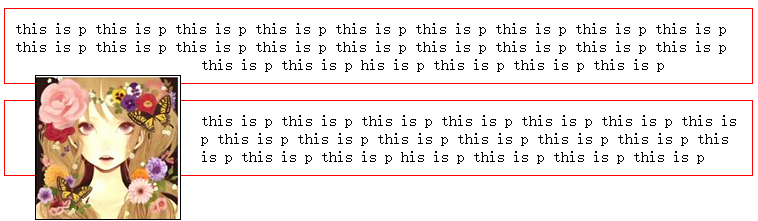
CSS2.1澄清了浮动元素行为的一个方面：浮动元素会延伸，从而包含其所有后代浮动元素。所以通过设置父元素为浮动元素，可以把浮动元素包含在其父元素内。

p{  
 float: left;  
 border: 1px solid red;  
 padding: 100px;  
}  
img {  
 margin: 20px;  
 border: 1px solid black;  
 float: left;  
}



<p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 <img src="6E22.tmp.png">this is p this is p this is p this is p  
 his is p this is p this is p this is p  
</p>  
<p>  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 this is p this is p this is p this is p  
 his is p this is p this is p this is p  
</p>

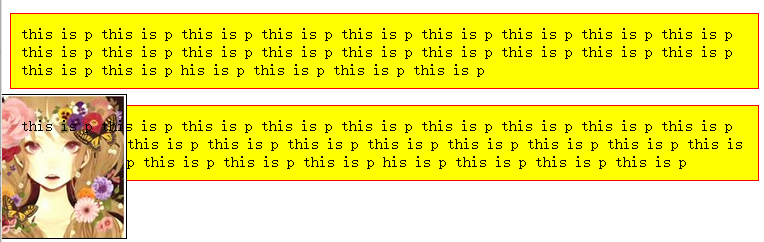
p{  
 border: 1px solid red;  
 padding: 10px;  
}  
img {  
 margin: 20px;  
 border: 1px solid black;  
 float: left;  
}



第二个p元素在img的底下，正是因为img的浮动因素。

负外边距

p{  
 border: 1px solid red;  
 padding: 10px;  
 background: yellow;  
}  
img {  
 margin: -40px 0 0 -40px;  
 border: 1px solid black;  
 float: left;  
}



由于使用负边距，超出了父元素的上边界。

这种向上推的做法往往不是什么好主意。

p{  
 border: 1px solid red;  
 width: 100px;  
 padding: 10px;  
 background: yellow;  
}  
img {  
 border: 1px solid black;  
 float: left;

由于img的宽度比父元素更宽，所以导致img超出了p元素的右边界。

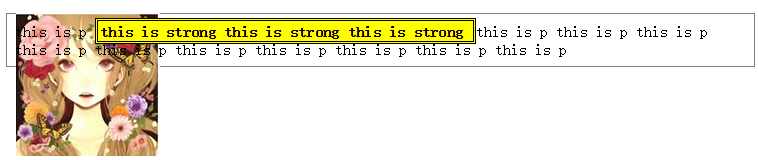


### 10.1.4 浮动元素内容和重叠

1.行内框与一个浮动元素重叠时，其边框、背景和内容都在该浮动元素”之上”显示。

<body>  
 <img src="6E22.tmp.png">  
 <p class="box">  
 this is p  
 <strong>  
 this is strong  
 this is strong  
 this is strong  
 </strong>  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 </p>  
</body>

img{  
 float: left;  
 margin: 0 -200px 10px 10px;  
}  
  
p.box{  
 border: 1px solid gray;  
 padding: 0.5em;  
}  
  
p.box strong{  
 border: 3px double black;  
 background: yellow;  
 padding: 2px;  
}  
  
  
h2{  
 background: yellow;  
}

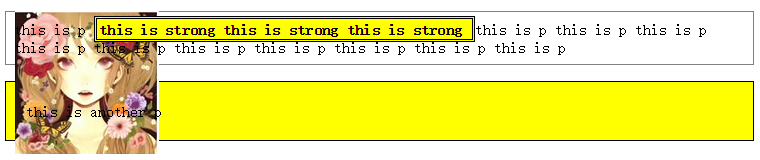


Strong作为行内元素，其边框、背景和内容都在该img浮动元素”之上”显示。

2.块框与一个浮动元素重叠时，其边框和背景在该浮动元素”之下”显示。而内容在浮动元素之上显示。

<body>  
 <img src="6E22.tmp.png">  
 <p class="box">  
 this is p  
 <strong>  
 this is strong  
 this is strong  
 this is strong  
 </strong>  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 </p>  
  
 <p class="last">this is another p</p>  
</body>

img{  
 float: left;  
 margin: 0 -200px 10px 10px;  
}  
  
p.box{  
 border: 1px solid gray;  
 padding: 0.5em;  
}  
  
p.box strong{  
 border: 3px double black;  
 background: yellow;  
 padding: 2px;  
}  
  
h2{  
 background: yellow;  
}  
  
p.last{  
 background: yellow;  
 border: 1px solid black;  
 padding: 20px;  
}



P元素作为块级元素，它的内容显示在浮动元素图像之上，但是边框和背景则被浮动元素遮蔽了。

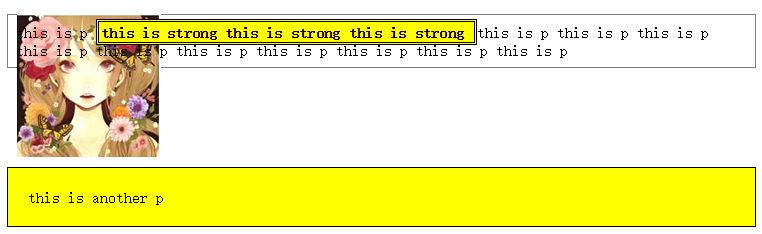
### 10.1.5 清除

可以利用清除设置元素禁止浮动元素出现在它旁边。

为了确保上述的第二个p元素不会放在左浮动元素的右边

改变第二个p元素的样式

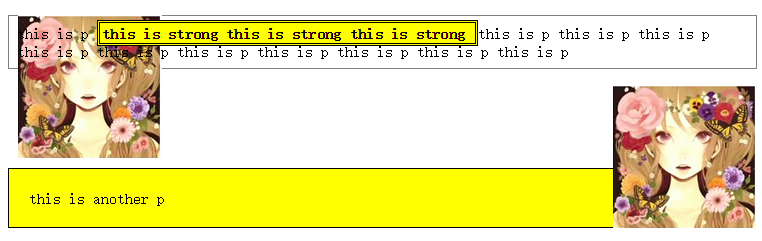
p.last{  
 clear: left;  
 background: yellow;  
 border: 1px solid black;  
 padding: 20px;  
}



这样可以防止p元素左边有浮动元素。但是允许p元素右边有浮动元素。

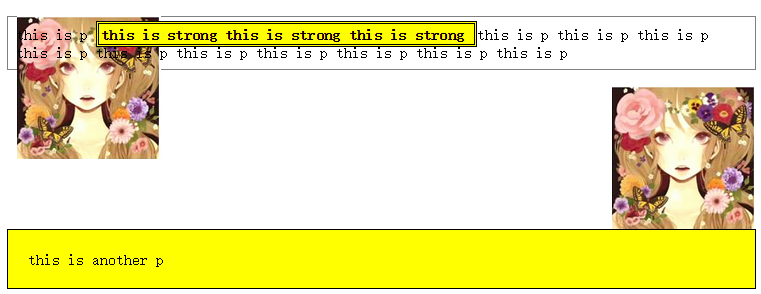
<body>  
 <img class="left" src="6E22.tmp.png">  
 <p class="box">  
 this is p  
 <strong>  
 this is strong  
 this is strong  
 this is strong  
 </strong>  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 this is p  
 </p>  
 <img class="right" src="6E22.tmp.png">  
 <p class="last">this is another p</p>  
</body>

img.left{  
 float: left;  
 margin: 0 -200px 10px 10px;  
}  
  
img.right{  
 float: right;  
}  
  
p.box{  
 border: 1px solid gray;  
 padding: 0.5em;  
}  
  
p.box strong{  
 border: 3px double black;  
 background: yellow;  
 padding: 2px;  
}  
  
h2{  
 background: yellow;  
}  
  
p.last{  
 clear: left;  
 background: yellow;  
 border: 1px solid black;  
 padding: 20px;  
}

只是清除了左边，右边仍然可以有浮动元素。

修改使清除所有的浮动元素。

p.last{  
 clear: both;  
 background: yellow;  
 border: 1px solid black;  
 padding: 20px;  
}



**Clear会增加元素的上外边距，使之最后落在浮动元素的下面。这实际上会忽略为清除元素设置的上外边边距宽度（如果宽度不是很大的情况下）。**

<body>  
 <img class="left" src="6E22.tmp.png">  
 <img class="right" src="6E22.tmp.png">  
 <p class="box">  
 this is p  
 </p>  
</body>

img.left{  
 float: left;  
 width: 50px;  
 height: 50px;  
}  
  
img.right{  
 width: 100px;  
 height: 10px;  
}  
  
  
p.box{  
 border: 1px solid gray;  
}

****

未使用清除前，元素p与浮动的图像元素重叠。

p.box{  
 border: 1px solid gray;  
 clear: left;  
}



使用清除后，自动增加了p元素的上外边距，确保它落在浮动元素img元素的下面。CSS2.1引入了清除区域的概念，清除区域是在元素上外边距之上增加的额外间隔（确保浮动元素不会与该元素重叠），不允许浮动元素进入这个范围，意味着设置clear属性时，设置了清除属性的元素p元素的外边距并不改变，之所以该元素会向下移动是因为清除区域造成的。

给p元素设置上外边距15px

p.box{  
 border: 1px solid gray;  
 clear: left;  
 margin-top: 15px;  
}

****

**那么可以知道，由于img的高度是50px，清除区域的高度是50px-10px（第二个img图像的高度）-15px（p元素的上边距） = 25px**

**这样正好将p元素上边框边界推到了刚好越过浮动元素img元素的下边界。**

**当然如果要为清除元素的顶端与一个浮动元素的底端之间确定一个边距，可以为浮动元素本身设置一个下边距，如果希望img浮动元素和p元素有真正的15px距离**

img.left{  
 float: left;  
 width: 50px;  
 height: 50px;  
 margin-bottom: 15px; /\*设置15px的边距\*/  
}  
  
img.right{  
 width: 100px;  
 height: 10px;  
  
}  
  
  
p.box{  
 border: 1px solid gray;  
 clear: left;  
}



这样p元素和img元素就有15px的间距了。

## 10.2 定位

### 10.2.1 基本概念

Position属性

1. static：元素框正常生成。块级元素生成一个矩形框，作为文档流的一部分，行内元素则会创建一个或多个行框，置于父元素中。
2. relative：元素框偏移某个距离，元素仍保持其未定位前的形状，它所本所占的空间仍保留着。
3. absolute：元素框从文档流完全删除，并相对于其包含块定位。元素原先在正常文档流中所占的空间会关闭，就好像该元素原来不存在一样。元素定位后生成一个块级框，而不论原来在正常流中生成何种类型的框。
4. fixed：元素框的表现类似于将position设置为absolute，不过其包含块是视窗本身。

包含块：对于浮动元素，包含块定义为最近的块级祖先元素。对于定位，情况则没有那么简单。

1. 根元素的包含块（也称为初始包含块）由用户代理建立。在HTML中，根元素就是html元素，在大多数浏览器中，初始包含块是一个视窗大小的矩形。
2. 对于非根元素，如果position是relative或static，包含块则由最近的块级框、表单元格或行内块组选框的内容边界构成。
3. 对于非根元素，如果position值是absolute，包含块设置为最近的position值不是static的祖先元素（可以是任何类型）。

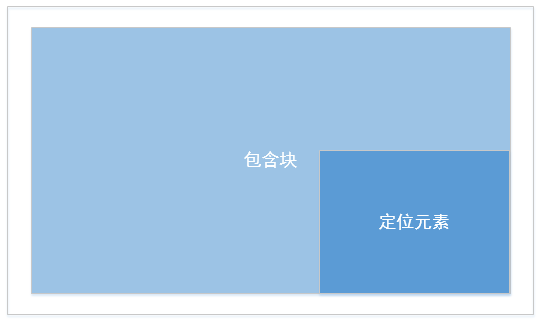
如果祖先元素是块级元素，包含块则色织为该元素的内边距边界。

如果祖先元素是行内元素，包含块则设置为该祖先元素的内容边界。…

元素可以定位到其包含块的外面。

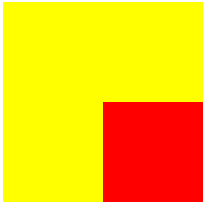
偏移属性：正直会导致向内偏移，使边界朝着包含块的中心移动，而负值会导致向外偏移。

偏移定位元素的外边距边界时，带来的影响是元素的所有一切（包括外边距、边框、内边距、内容）都会在定位的过程中移动。可以为定位元素设置外边距、边框和内边距。



<body>  
 <div class="contain">  
 <div class="inner"></div>  
 </div>  
</body>

div.contain{  
 position: absolute;  
 width: 200px;  
 height: 200px;  
 background: yellow;  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:50%;  
 bottom: 0;  
 left: 50%;  
 right: 0;  
 background: red;  
}

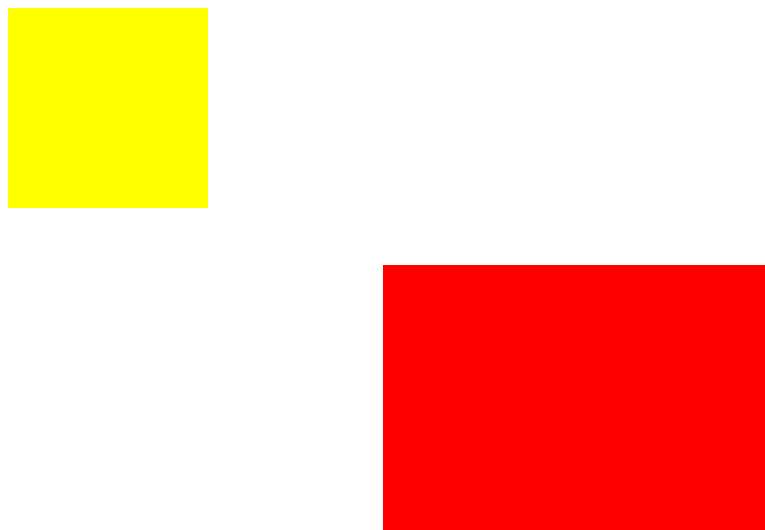


如果div.contain没有设置position，则默认为static

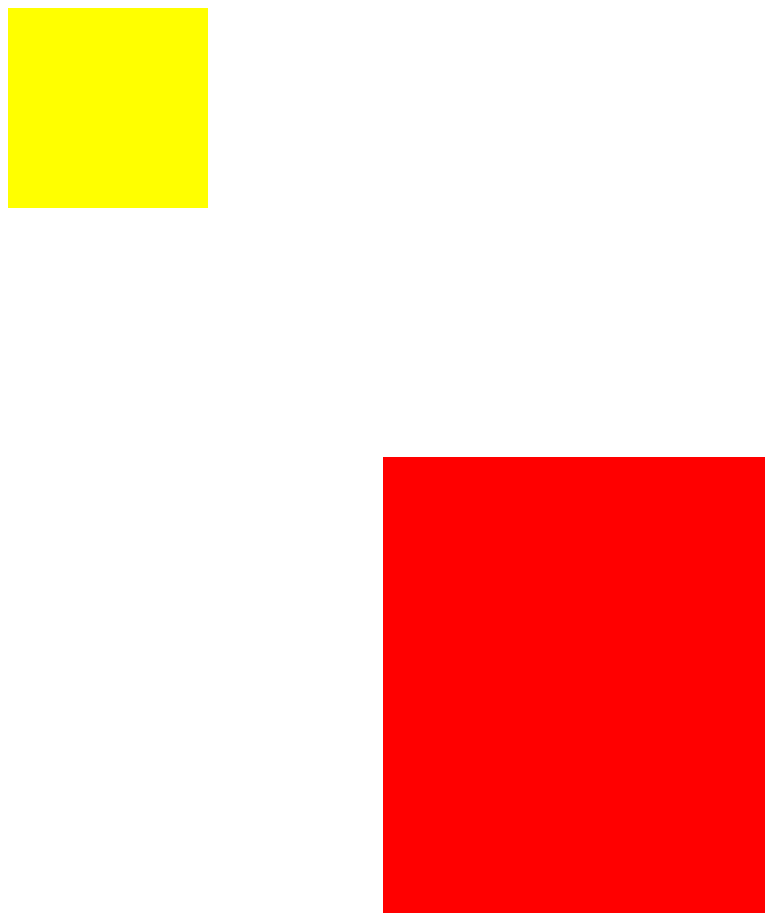
div.contain{  
 width: 200px;  
 height: 200px;  
 background: yellow;

/\*未设置position\*/  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:50%;  
 bottom: 0;  
 left: 50%;  
 right: 0;  
 background: red;  
}

如果没有祖先，元素的包含块定义为初始包含块，即视窗大小。

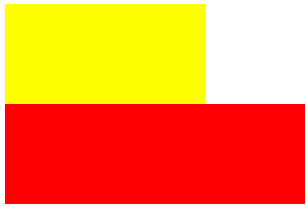


如果视窗大小变化！



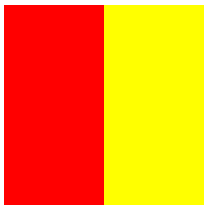
如果设置为负值，超出了包含块的大小

div.contain{  
 position: absolute;  
 width: 200px;  
 height: 200px;  
 background: yellow;  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:50%;  
 bottom: 0;  
 left: 0;  
 right: -50%;  
 background: red;  
}



### 10.2.2 宽度和高度

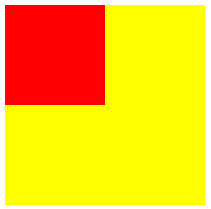
div.contain{  
 position: absolute;  
 width: 200px;  
 height: 200px;  
 background: yellow;  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:0;  
 bottom: 0;  
 left: 0;  
 right: 50%; /\*向内偏移，距离包含块的右边的偏移是50% \*/  
 background: red;  
}



红色的定位元素，黄色的是整个包含块，红色占了1/2的面积大小包含块

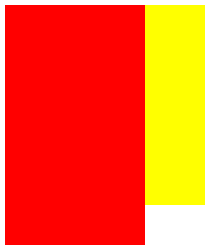
其中的width和heght默认了 50%和100%

div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:0;  
 bottom: 0;  
 left: 0;  
 right: 50%;  
 background: red;  
 width: 50%;  
 height: 50%;  
}

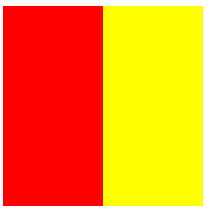


设置内边距，使其部分超出了包含块

div.contain{  
 position: absolute;  
 width: 200px;  
 height: 200px;  
 background: yellow;  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:0;  
 bottom: 0;  
 left: 0;  
 right: 50%;  
 background: red;  
 width: 50%;  
 height: 100%;  
 padding: 20px; /\*设置了内边距\*/  
}



div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:0;  
 bottom: 0;  
 left: 0;  
 right: 50%;  
 background: red;  
 width: auto;  
 height: auto; /\*内容区变小了\*/  
 padding: 20px; /\*设置了内边距\*/  
}



尽管看起来一样，但是内容区变小了，因为增加了内边距20px

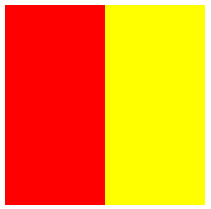
限制宽度和高度、

div.contain{  
 position: absolute;  
 width: 200px;  
 height: 200px;  
 background: yellow;  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:0;  
 bottom: 0;  
 left: 0;  
 right: 50%;  
 background: red;  
 width: 400px; /\*超出了包含块的宽度\*/  
 height: auto; /\*内容器变小了\*/  
}



包含块已经不可见了，如果设置一个最大宽度限制

div.contain{  
 position: absolute;  
 width: 200px;  
 height: 200px;  
 background: yellow;  
}  
  
  
div.inner{  
 position: absolute;/\*其包含块设置为最近的position值不是static\*/  
 top:0;  
 bottom: 0;  
 left: 0;  
 right: 50%;  
 background: red;  
 width: 400px; /\*超出了包含块的宽度\*/  
 max-width: 50%; /\*此时由于width超过了包含块的大小，这里会将其限制为包含块的一半\*/  
 height: auto; /\*内容器变小了\*/  
}

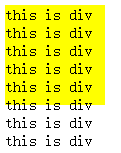


### 10.2.3 内容溢出和剪裁

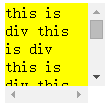
溢出

<div class="contain">  
 this is div  
 this is div  
 this is div  
 this is div  
 this is div  
 this is div  
 this is div  
 this is div  
 <div class="inner"></div>  
</div>

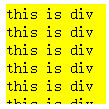
div.contain{  
 position: absolute;  
 width: 100px;  
 height: 100px;  
 background: yellow;  
}



div.contain{  
 position: absolute;  
 width: 100px;  
 height: 100px;  
 background: yellow;  
 overflow: scroll; /\*设置为滚动\*/  
}



div.contain{  
 position: absolute;  
 width: 100px;  
 height: 100px;  
 background: yellow;  
 overflow: hidden; /\*超出部分不显示\*/  
}



div.contain{  
 position: absolute;  
 width: 100px;  
 height: 100px;  
 background: yellow;  
 overflow: auto; /\*默认是滚动条，类似于scroll\*/  
}

内容裁剪

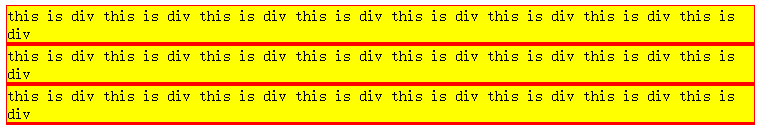
Rect 忽略

### 10.2.4 元素的可见性

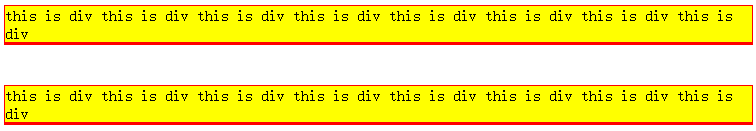
Visibility:hidden 元素不可见，但是元素还在那里，只不过你看不见它

Display:none 元素不仅不显示，还会从文档中删除

div{  
 background: yellow;  
 border: 1px solid red;  
}

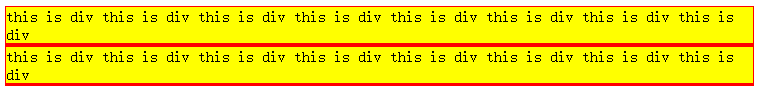


div{  
 background: yellow;  
 border: 1px solid red;  
}  
  
div.second{  
 visibility: hidden;  
}



只是不可见了而已。

div{  
 background: yellow;  
 border: 1px solid red;  
}  
  
div.second{  
 display: none;  
}



把元素从文档中删除了。

Visibility可以继承，如果父元素是hidden，子元素是visible，那么子元素仍然是可以显示的，如果子元素不设置，那么子元素继承hidden，不可显示。

### 10.2.5 绝对定位

需要注意的是默认的position是static。

元素绝对定位时，会从文档流中完全删除。然后相对于其包含块定位，其边界根据偏移属性放置。定位元素不会流入其他元素的内容，反之亦然。绝对定位元素可能覆盖其他元素，或者被其他元素覆盖。

绝对元素的包含块是最近的position值不为static的祖先元素。创作人员通常会选择一个元素作为绝对定位元素的包含块，将其position指定为relative而且没有偏移。

有时想确保body元素为其所有后代建立一个包含块，而不是让用户代理选择初始包含块。

body{  
 position: relative;  
}

元素绝对定位时，会为其后代元素建立一个包含块。

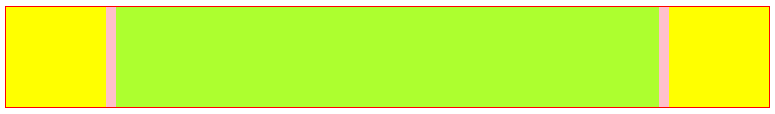
<body>  
 <div class="contain">  
 contain block  
 <div class="inner">  
 inner  
 <div class="innerInner">  
 innerInner  
 </div>  
 </div>  
 </div>  
</body>

div.contain{  
 position: relative;  
 width: 100%;  
 height: 100px;  
 border: 1px solid red;  
 background: pink;  
}  
  
div.inner{  
 position: absolute;  
 top:0;  
 right: 0;  
 width: 50%;  
 height: 100%;  
 margin-left: auto;  
 background: yellow;  
}  
  
div.innerInner{  
 position: absolute;  
 bottom: 0;  
 left: 0;  
 width: 50%;  
 height: 50%;  
 background: greenyellow;  
}

网易笔试题

<body>  
 <div class="contain">  
 <div class="left">  
 </div>  
 <div class="center">  
 </div>  
 <div class="right">  
 </div>  
 </div>  
</body>

div.contain{  
 position: relative;  
 width: 100%;  
 height: 100px;  
 border: 1px solid red;  
 background: pink;  
}  
  
div.left{  
 position: absolute;  
 top:0;  
 left: 0;  
 width: 100px;  
 height: 100%;  
 background: yellow;  
}  
  
div.center{  
 position: absolute;  
 top: 0;  
 left: 110px;  
 right: 110px;  
 height: 100%;  
 background: greenyellow;  
}  
  
div.right{  
 position: absolute;  
 right: 0;  
 top: 0;  
 width: 100px;  
 height: 100%;  
 background: yellow;  
}





中间的长度是自适应的。

粉红色的包含块背景

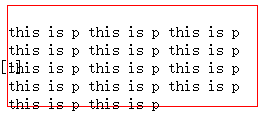
绝对定位元素的放置和大小

自动边偏移

静态位置，元素在正常流中原本的位置。

<div class="contain">  
 <p>  
 this is p this is p  
 this is p this is p  
 this is p this is p  
 this is p this is p  
 <span>[1]</span>  
 this is p this is p  
 this is p this is p  
 this is p this is p  
 </p>  
</div>

div.contain{  
 border: 1px solid red;  
 width: 100%;  
 height: 100px;  
}  
  
span{  
 position: absolute; /\*由于DIV的定位是static（默认）,所以包含快是视窗\*/  
 top:auto; /\*静态时偏移包含块顶端的偏离量就是此时的偏移量\*/  
 left: 0;  
}

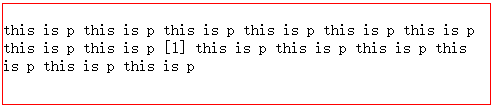


绝对定位一个元素，使用auto，与其“静态”位置偏离量一致。

包含块是视窗，所以左偏移量是0，落在了p元素的外面是正常的。

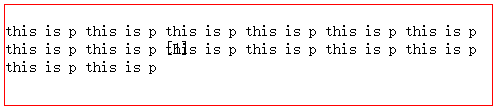
静态位置

div.contain{  
 border: 1px solid red;  
 width: 100%;  
 height: 100px;  
 }  
  
span{  
}



左右偏移量都设置为auto，首先元素从正常流中删除，其次其偏移量相对于静态时的偏移量一致

div.contain{  
 border: 1px solid red;  
 width: 100%;  
 height: 100px;  
 }  
  
span{  
 position: absolute; /\*由于DIV的定位是static（默认）,所以包含快是视窗\*/  
 top:auto; /\*静态时偏移包含块顶端的偏离量就是此时的偏移量\*/  
 left: auto;  
}

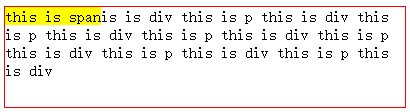


与静态位置一致，但是正常流空间关闭，导致定位元素与正常元素内容重叠。

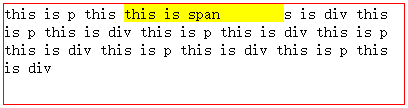
非替换元素的放置和大小

<div class="contain">  
 this is p this is div  
 this is p this is div  
 this is p this is div  
 this is p this is div  
 <span>this is span</span>  
 this is p this is div  
 this is p this is div  
 this is p this is div  
</div>

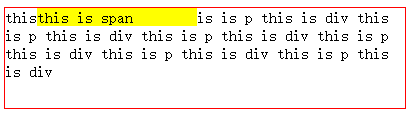
div.contain{  
 position: relative;  
 border: 1px solid red;  
 width: 25em;  
 height: 100px;  
 }  
  
span{  
 position: absolute; /\*包含快是div\*/  
 top:0;   
 left: 0;  
 right: auto;  
 width: auto;  
 background: yellow;  
}



div.contain{  
 position: relative;  
 border: 1px solid red;  
 width: 25em;  
 height: 100px;  
 }  
  
span{  
 position: absolute; /\*包含快是div\*/  
 top:0;  
 left: 1em;  
 right: 1em;  
 width: 10em;  
 background: yellow;  
 margin: 0 auto; /\*左右外边距是auto，促使元素居中\*/  
}



div.contain{  
 position: relative;  
 border: 1px solid red;  
 width: 25em;  
 height: 100px;  
 }  
  
span{  
 position: absolute; /\*包含快是div\*/  
 top:0;  
 left: 1em;  
 right: 1em; /\*过度受限的情况下忽略此值。从左到右的阅读流中\*/  
 width: 10em;  
 background: yellow;  
 margin: 0 1em; /\*左右外边距是1em\*/  
}

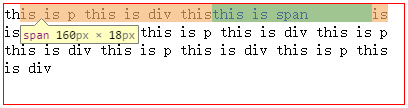


span{  
 position: absolute; /\*包含快是div\*/  
 top:0;  
 left: 1em;  
 right: 1em; /\*过度受限的情况下忽略此值。从左到右的阅读流中\*/  
 width: 10em;  
 background: yellow;  
 margin-left: 1em; /\*左右外边距是1em\*/  
 margin-right: auto;  
}

效果和上面一样

div.contain{  
 position: relative;  
 border: 1px solid red;  
 width: 25em;  
 height: 100px;  
 }  
  
span{  
 position: absolute; /\*包含快是div\*/  
 top:0;  
 left: 1em;  
 right: 1em; /\*过度受限的情况下忽略此值。从左到右的阅读流中\*/  
 width: 10em;  
 background: yellow;  
 margin-left: auto; /\*重置左外边距\*/

margin-right: 1em;  
}



重置了左外边距，过度受限的情况下，同时需要注意偏移和外边距的区别，图中空白的部分的是便宜，黄色的部分是左右外边距，绿色部分是内容区。

如果想要或者垂直居中的效果，跟水平剧中一样，设置margin的上下外边距为auto值。

替换元素的放置和大小