01 CSS介绍

CSS简单介绍

CSS概述

CSS(Cascading Style Sheets)指层叠样式表，就是对html进行样式修饰语言。

样式通常存储在CSS文件中，多个样式可以层叠为一个，共同作用一个HTML元素。

层叠：就是层层覆盖叠加，如果不同的CSS样式对同一html标签进行修饰，样式有冲突的部分应用优先级高的，不冲突的部分共同作用

样式表：就是CSS属性样式的集合

CSS作用

修饰html使其html样式更加好看

提高样式代码的复用性

html的内容与样式相分离 便于后期维护

CSS引入方式

从CSS 样式代码插入的形式来看基本可以分为以下3种：

内嵌式、内部式和外部式

内嵌式CSS样式，就是把CSS代码直接写在现有的HTML标签中。

```
<p style="color:sienna;margin-left:20px">这是一个段落。</p>
```

内部式CSS样式，就是可以把CSS样式代码写在<style type="text/css"></style>标签之间。

```
<style type="text/css">
hr{color:sienna;}
p{margin-left:20px;}
body{background-image:url("images/back40.gif");}
</style>
```

外部式CSS样式，就是把CSS代码写一个单独的外部文件中。文件以".css"为扩展名。

一种在<head>内使用<link>标签将CSS样式文件链接到HTML文件内。

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

一种是直接通过\@import方式引入

```
<style type="text/css">@import url("base.css");</style>
```

注意：

1、CSS样式文件名称以有意义的英文字母命名，如 main.css。

2、rel="stylesheet" type="text/css" 是固定写法不可修改。

3、<link>标签位置一般写在<head>标签之内。

CSS编码规范

- CSS书写规范
    - 使用CSS缩写属性。
    - 以"."开头。
    - 简写命名，但前提是要让人看懂。
    - 16进制颜色代码缩写。
    - 连字符选择器使用中横线"-"来为选择器命名，不建议使用" _"。
    - 不要随意使用id。
- 注意事项
    - 命名应反映该元素的功能或使用通用名称。
    - 一律小写，不允许使用大写字母或 _。
    - 尽量用英文，采用简明有语义的英文单词进行组合 。
    - 尽量不缩写，除非一看就明白的单词。
    - 避免class与id重名 。
    - 尽可能提高代码模块的复用，样式尽量用组合的方式。
    - 每个声明结束都应该带一个分号，不管是不是最后一个声明。
    - 确保代码能够自我描述、注释良好并且易于他人理解。

02 CSS选择器

指定CSS要作用的标签，那个标签的名称就是选择器。意为：选择哪个容器。

CSS选择器的类型：

- #id：ID选择器，其实使用的是标签的中的id属性。
- .class：CLASS选择器，匹配class包含(不是等于)特定类的元素
- Element：元素选择器，使用的就是html的标签名
- [属性]：属性选择器，为带有此属性的所有元素设置样式
- *：通用元素选择器，匹配页面任何元素（不建议使用）

ID选择器

ID 选择器可以为标有特定 id 的 HTML 元素指定特定的样式。

HTML元素以id属性来设置ID选择器,CSS 中ID选择器以 "#" 来定义。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#idDiv{color:red;}
</style>
<head>
<body>
<div id="idDiv">通过id颜色变红</div>
</body>
</html>
```

在一个 HTML 文档中，ID 选择器会使用一次，而且仅一次。

ID 选择器不能结合使用，因为 id 属性不允许有以空格分隔的词列表。

CLASS选择器

CLASS选择器用于描述一组元素的样式，CLASS选择器有别于ID选择器，CLASS可以在多个元素中使用。

CLASS选择器在HTML中以class属性表示, 在 CSS 中，类选择器以一个点"."号显示。

类选择器允许以一种独立于文档元素的方式来指定样式。

该选择器可以单独使用，也可以与其他元素结合使用。

提示：只有适当地标记文档后，才能使用这些选择器，所以使用这两种选择器通常需要先做一些构想和计划。

要应用样式而不考虑具体设计的元素，最常用的方法就是使用类选择器。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
.classDiv{color:green}
</style>
<head>
<body>
<div class="classDiv">通过class颜色变绿</div>
</body>
</html>
```

类选择器可以结合元素选择器来使用。

例如，您可能希望只有段落显示为红色文本：

```
p.important {color:red;}
```

选择器现在会匹配 class 属性包含 important 的所有 p 元素，但是其他任何类型的元素都不匹配，不论是否有此 class 属性。选择器 p.important 解释为："其 class 属性值为 important 的所有段落"。

CSS 多类选择器

在 HTML 中，一个 class 值中可能包含一个词列表，各个词之间用空格分隔。例如，如果希望将一个特定的元素同时标记为重要（important）和警告（warning），就可以写作：

```
<p class="important warning">
This paragraph is a very important warning.
</p>
```

这两个词的顺序无关紧要，写成 warning important 也可以。

我们假设 class 为 important 的所有元素都是粗体，而 class 为 warning 的所有元素为斜体，class 中同时包含 important 和 warning 的所有元素还有一个红色色的背景 。就可以写作：

```
.important{font-weight:bold;}
.warning{font-style:italic;}
.important.warning{background:red;}
```

通过把两个类选择器链接在一起，仅可以选择同时包含这些类名的元素（类名的顺序不限）。

如果一个多类选择器包含类名列表中没有的一个类名，匹配就会失败。

请看下面的规则：

```
.important.urgent {background:silver;}
```

不出所料，这个选择器将只匹配 class 属性中包含词 important 和 urgent 的 p 元素。因此，如果一个 p 元素的 class 属性中只有词 important 和 warning，将不能匹配。不过，它能匹配以下元素：

```
<p class="important urgent warning">
This paragraph is a very important and urgent warning.
</p>
```

属性选择器

属性选择器可以根据元素的属性及属性值来选择元素。

简单属性选择

如果希望选择有某个属性的元素，而不论属性值是什么，可以使用简单属性选择器。

如果您希望把包含标题（title）的所有元素变为红色，可以写作：

```
[title] {color:red;}
```

可以只对有 href 属性的锚（a 元素）应用样式：

```
a[href] {color:red;}
```

还可以根据多个属性进行选择，只需将属性选择器链接在一起即可。

例如，为了将同时有 href 和 title 属性的 HTML 超链接的文本设置为红色，可以这样写：

```
a[href][title] {color:red;}
```

根据具体属性值选择

除了选择拥有某些属性的元素，还可以进一步缩小选择范围，只选择有特定属性值的元素。

例如，假设希望将指向 Web 服务器上某个指定文档的超链接变成红色，可以这样写：

```
a[href="http://www.w3school.com.cn/about_us.asp"] {color: red;}
```

与简单属性选择器类似，可以把多个属性-值选择器链接在一起来选择一个文档。

```
a[href="http://www.w3school.com.cn/"][title="W3School"] {color: red;}
```

请注意，这种格式要求必须与属性值完全匹配。

如果属性值包含用空格分隔的值列表，匹配就可能出问题。

请考虑一下的标记片段：

```
<p class="important warning">
This paragraph is a very important warning.
</p>
```

如果写成 p[class="important"]，那么这个规则不能匹配示例标记。

要根据具体属性值来选择该元素，必须这样写：

```
p[class="important warning"] {color: red;}
```

根据部分属性值选择

如果需要根据属性值中的词列表的某个词进行选择，则需要使用波浪号（~）。

假设您想选择 class 属性中包含 important 的元素，可以用下面这个选择器做到这一点：

```
p[class~="important"] {color: red;}
```

如果忽略了波浪号，则说明需要完成完全值匹配。

该选择器等价于我们在类选择器中讨论过的点号类名记法。

也就是说，p.important 和 p[class="important"] 应用到 HTML 文档时是等价的。

那么，为什么还要有 "~=" 属性选择器呢？因为它能用于任何属性，而不只是 class。

例如，可以有一个包含大量图像的文档，其中只有一部分是图片。对此，可以使用一个基于 title 文档的部分属性选择器，只选择这些图片：

```
img[title~="Figure"] {border: 1px solid gray;}
```

这个规则会选择 title 文本包含 "Figure" 的所有图像。没有 title 属性或者 title 属性中不包含 "Figure" 的图像都不会匹配。

子串匹配属性选择器

下面为您介绍一个更高级的选择器模块，它是 CSS2 完成之后发布的，其中包含了更多的部分值属性选择器。按照规范的说法，应该称之为"子串匹配属性选择器"。

很多现代浏览器都支持这些选择器，包括 IE7。

下表是对这些选择器的简单总结：

类型	描述
[abc^="def"]	选择 abc 属性值以 "def" 开头的所有元素
[abc$="def"]	选择 abc 属性值以 "def" 结尾的所有元素
[abc*="def"]	选择 abc 属性值中包含子串 "def" 的所有元素

可以想到，这些选择有很多用途。

举例来说，如果希望对指向 W3School 的所有链接应用样式，不必为所有这些链接指定 class，再根据这个类编写样式，而只需编写以下规则：

```
a[href*="w3school.com.cn"] {color: red;}
```

提示：任何属性都可以使用这些选择器。

特定属性选择类型

这种属性选择器最常见的用途还是匹配语言值。

请看下面的例子：

```
*[lang|="en"] {color: red;}
```

上面这个规则会选择 lang 属性等于 en 或以 en- 开头的所有元素。因此，以下示例标记中的前三个元素将被选中，而不会选择后两个元素：

```
<p lang="en">Hello!</p>
<p lang="en-us">Greetings!</p>
<p lang="en-au">G'day!</p>
<p lang="fr">Bonjour!</p>
<p lang="cy-en">Jrooana!</p>
```

CSS 选择器参考手册

选择器	描述
[attribute]	用于选取带有指定属性的元素。
[attribute=value]	用于选取带有指定属性和值的元素。
[attribute~=value]	用于选取属性值中包含指定词汇的元素。
[attribute|=value]	用于选取带有以指定值开头的属性值的元素，该值必须是整个单词。
[attribute^=value]	匹配属性值以指定值开头的每个元素。
[attribute$=value]	匹配属性值以指定值结尾的每个元素。
[attribute*=value]	匹配属性值中包含指定值的每个元素。

综合案例

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
div[name='nameDiv']{color:yellow}
</style>
<head>
<body>
<div name="nameDiv">通过属性颜色变黄</div>
</body>
</html>
```

元素选择器

element(元素)选择器将样式添加到具有指定元素名称的所有元素

元素选择器表示方式：元素名{}

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
span{color:blue}
</style>
<head>
<body>
<div><span>通过元素颜色变蓝<span></div>
</body>
</html>
```

派生选择器

通过依据元素在其位置的上下文关系来定义样式，你可以使标记更加简洁。

在 CSS1 中，通过这种方式来应用规则的选择器被称为上下文选择器 (contextual selectors)，这是由于它们依赖于上下文关系来应用或者避免某项规则。在 CSS2 中，它们称为派生选择器，但是无论你如何称呼它们，它们的作用都是相同的。

派生选择器允许你根据文档的上下文关系来确定某个标签的样式。通过合理地使用派生选择器，我们可以使 HTML 代码变得更加整洁。

后代选择器

后代选择器（descendant selector）又称为包含选择器。

后代选择器可以选择作为某元素后代的元素。

> **根据上下文选择元素**

我们可以定义后代选择器来创建一些规则，使这些规则在某些文档结构中起作用，而在另外一些结构中不起作用。

举例来说，如果您希望只对 h1 元素中的 em 元素应用样式，可以这样写：

```
h1 em {color:red;}
```

上面这个规则会把作为 h1 元素后代的 em 元素的文本变为 红色。其他 em 文本（如段落或块引用中的 em）则不会被这个规则选中：

> **语法解释**

在后代选择器中，规则左边的选择器一端包括两个或多个用空格分隔的选择器。选择器之间的空格是一种结合符（combinator）。每个空格结合符可以解释为"... 在 ... 找到"、"... 作为 ... 的一部分"、"... 作为 ... 的后代"，但是要求必须从右向左读选择器。

因此，h1 em 选择器可以解释为 "作为 h1 元素后代的任何 em 元素"。如果要从左向右读选择器，可以换成以下说法："包含 em 的所有 h1 会把以下样式应用到该 em"。

> **具体应用**

后代选择器的功能极其强大。有了它，可以使 HTML 中不可能实现的任务成为可能。

假设有一个文档，其中有一个边栏，还有一个主区。边栏的背景为蓝色，主区的背景为白色，这两个区都包含链接列表。不能把所有链接都设置为蓝色，因为这样一来边栏中的蓝色链接都无法看到。

解决方法是使用后代选择器。在这种情况下，可以为包含边栏的 div 指定值为 sidebar 的 class 属性，并把主区的 class 属性值设置为 maincontent。然后编写以下样式：

```
div.sidebar {background:blue;}
div.maincontent {background:white;}
div.sidebar a:link {color:white;}
div.maincontent a:link {color:blue;}
```

有关后代选择器有一个易被忽视的方面，即两个元素之间的层次间隔可以是无限的。

例如，如果写作 ul em，这个语法就会选择从 ul 元素继承的所有 em 元素，而不论 em 的嵌套层次多深。

子元素选择器

与后代选择器相比，子元素选择器（Child selectors）只能选择作为某元素子元素的元素。

如果您不希望选择任意的后代元素，而是希望缩小范围，只选择某个元素的子元素，请使用子元素选择器（Child selector）。

例如，如果您希望选择只作为 h1 元素子元素的 strong 元素，可以这样写：

```
h1 > strong {color:red;}
```

这个规则会把第一个 h1 下面的两个 strong 元素变为红色，但是第二个 h1 中的 strong 不受影响：

```
<h1>This is <strong>very</strong><strong>very</strong> important.</h1>
<h1>This is <em>really <strong>very</strong></em> important.</h1>
```

子选择器使用了大于号（子结合符）。

子结合符两边可以有空白符，这是可选的。

如果从右向左读，选择器 h1 > strong 可以解释为"选择作为 h1 元素子元素的所有 strong 元素"。

> **结合后代选择器和子选择器**

```
table.company td > p
```

上面的选择器会选择作为 td 元素子元素的所有 p 元素，这个 td 元素本身从 table 元素继承，该 table 元素有一个包含 company 的 class 属性。

相邻兄弟选择器

相邻兄弟选择器（Adjacent sibling selector）可选择紧接在另一元素后的元素，且二者有相同父元素。

如果需要选择紧接在另一个元素后的元素，而且二者有相同的父元素，可以使用相邻兄弟选择器（Adjacent sibling selector）。

例如，如果要增加紧接在 h1 元素后出现的段落的上边距，可以这样写：

```
h1 + p {margin-top:50px;}
```

这个选择器读作："选择紧接在 h1 元素后出现的段落，h1 和 p 元素拥有共同的父元素"。

> **语法解释**

相邻兄弟选择器使用了加号（+），即相邻兄弟结合符（Adjacent sibling combinator）。

注释：与子结合符一样，相邻兄弟结合符旁边可以有空白符。

```
<div>
<ul>
<li>List item 1</li>
<li>List item 2</li>
<li>List item 3</li>
</ul>
<ol>
<li>List item 1</li>
<li>List item 2</li>
<li>List item 3</li>
</ol>
</div>
```

在上面的片段中，div 元素中包含两个列表：一个无序列表，一个有序列表，每个列表都包含三个列表项。这两个列表是相邻兄弟，列表项本身也是相邻兄弟。不过，第一个列表中的列表项与第二个列表中的列表项不是相邻兄弟，因为这两组列表项不属于同一父元素（最多只能算堂兄弟）。

请记住，用一个结合符只能选择两个相邻兄弟中的第二个元素。请看下面的选择器：

```
li + li {font-weight:bold;}
```

上面这个选择器只会把列表中的第二个和第三个列表项变为粗体。第一个列表项不受影响。

> **结合其他选择器**

相邻兄弟结合符还可以结合其他结合符：

```
html > body table + ul {margin-top:20px;}
```

这个选择器解释为：选择紧接在 table 元素后出现的所有兄弟 ul 元素，该 table 元素包含在一个 body 元素中，body 元素本身是 html 元素的子元素。

伪类选择器

伪类用于向某些选择器添加特殊的效果。

:link向未访问的链接添加特殊的样式。

:visited向访问过的链接添加特殊的样式。

:hover在鼠标移到链接上时添加的特殊样式。

:hover 必须位于 :link 和 :visited 之后！！

:active向活动的链接添加特殊的样式，当你点击一个链接时它变成活动链接。

:active必须位于:hover之后！！

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#aa:link{color:gray} /* 未访问的链接 */
#aa:visited{color:green} /* 已访问的链接 */
#aa:hover{color:red} /* 鼠标移动到链接上 */
#aa:active{color:white} /* 鼠标移动到链接上 */
</style>
<head>
<body>
<p>:link、:visited、:hover、:active用法</P>
<p>链接颜色为灰色，鼠标放上显示红色，点击鼠标显示白色，完成访问后显示绿色</p>
<a id="aa"href="http://www.baidu.com"target="_blank">这是一个超链接</a>
<p>请注意字体颜色变化</p>
</body>
</html>
```

:before选择器向选定的元素前插入内容。

使用[content]{.underline} 属性来指定要插入的内容。

:after选择器向选定的元素之后插入内容。

使用content 属性来指定要插入的内容。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#bb:before{content:'开始：'}
#bb:after{content:'结束！'}
</style>
```

```
<head>
<body>
<p>:before和:after用法：</p>
<p>在文本前面加上开始，后面加上结束</p>
<div id="bb">这是一句台词。</div>
</body>
</html>
```

:first-child选择器匹配第一个子元素。

:last-child选择器用来匹配父元素中最后一个子元素

:focus选择器用于选择具有焦点的元素。

:focus应用于元素具有焦点的时间内。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#cc li:first-child{color:red}
#cc li:last-child{color:blue}
#username:focus{background:red}
</style>
<head>
<body>
<p>:first-child和:last-child用法：</p>
<p>获取列表第一个元素，并设置成红色；列表最后一个元素设置成蓝色</p>
<ul id="cc">
    <li>第1条数据</li>
    <li>第2条数据</li>
    <li>第3条数据</li>
    <li>第4条数据</li>
</ul>
<hr>
<p>:focus用法：</p>
<p>文本框获取焦点后背景变为红色 </p>
<input id="username"type="text"value="focus">
</body>
</html>
```

:disabled选择器匹配每个禁用的的元素。

:enabled 选择器匹配每个启用的的元素。

:empty选择器选择每个没有任何子级的元素，包括文本节点。

:not选择器匹配每个元素是不是指定的元素/选择器。

否定选择器允许您反转任何选择。此选择器在CSS3中添加。

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#userId:enabled{background:red}
#dept:disabled{background:gray}
/* :empty */
div p{width:150px; height:20px; background:yellow}
div p:empty{width:150px; height:20px; background:blue; }
/* :not */
divinput{background:green}
div:not(input){background:yellow}
</style>
<head>
<body>
<p>disabled为禁用的表单背景置灰，enabled为启动的表单元素背景变为红色</p>
<input id="userId"type="text"value="enabled"><br />
<input id="dept"type="text"disabled="disabled"value="disabled"><br />
<hr/>
<p>元素内容为空背景设为蓝色，否则为黄色(empty)</p>
<div>
    <p>第一条</p>
    <p></p>
    <p>第三条</p>
</div>
<hr/>
<p>标签中所有input文本框背景均设为绿色，不是input的元素则设为黄色<p>
<div>
<input type="text"/><br/>
<textarea rows="3"col="20"></textarea>
</div>
</body>
</html>
```

03 CSS属性

背景(Background)

background：背景缩写属性可以在一个声明中设置所有的背景属性。

> 语法：

```
background: background-color background-image background-position/background-size
            background-repeat background-origin background-clip
            background-attachment  initial|inherit;
```

允许上述值有部分缺失

background-attachment：设置背景图像是否固定或者随着页面的其余部分滚动

background-color：设置一个元素的背景颜色

background-image：设置一个元素的背景图像

background-position：设置背景图像的起始位置

background-repeat：设置如何平铺对象的 background-image 属性

background-size：指定背景图片大小

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
body{
/*  background: yellow url(images/beijing.jpg) no-repeat fixed center; */

    background-color:yellow;
    background-image:url('images/beijing.jpg');
    background-repeat:no-repeat;
    background-attachment:fixed;
    background-position:center;
    background-size:300px200px;
}
</style>
<head>
<body>
<p>背景颜色为黄色，背景图像居中显示，且不平铺</p>
</body>
</html>
```

颜色(Color)

opacity：设置元素的透明度级别。

```
<!DOCTYPE html>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
    div{
    background-color:red;
    opacity:0.5;
    filter:Alpha(opacity=50); /* IE8 and earlier */
    }
</style>
</head>
<body>
    <div>这里div背景色透明度为50%</div>
</body>
</html>
```

字体(Font)

font：在一个声明中设置所有字体属性。

> 语法：

```
font:font-style font-variant font-weight font-size/line-height font-family
```

font-size和font-family的值是必需的。如果缺少了其他值，默认值将被插入。

font-family：指定一个元素的字体

font-size：设置字体大小

font-style：指定文本的字体样式

font-variant：设置小型大写字母的字体显示文本，这意味着所有的小写字母均会被转换为大写。

font-weight：设置文本的粗细

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#content{
/* font:italic bold 16px '微软雅黑'; */
color:red;
font-family:'微软雅黑';
font-size:16px;
font-style:italic;
font-variant:normal;
font-weight:bold;
}
</style>
<head>
<body>
<p>字体设置：字体颜色为红色，斜体，加粗，大小为12px，微软雅黑</p>
<div id="content">
this is a main content.
</div>
</body>
</html>
```

文本(Text)

direction：指定文本方向/书写方向

line-height：设置行高

text-align：指定元素文本的水平对齐方式

text-decoration：规定添加到文本的修饰

text-indent：定文本块中首行文本的缩进

text-overflow：指定当文本溢出包含它的元素，应该发生什么

text-shadow：向文本设置阴影

text-transform：控制文本的大小写

vertical-align：设置一个元素的垂直对齐

white-space：指定元素内的空白怎样处理

word-break：规定自动换行的处理方法

word-spacing：增加或减少字与字之间的空白

word-wrap：允许长的内容可以自动换行

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#textCss{
    color:green;
    direction:ltr;
    line-height:40px;
    text-align:left;
    text-indent:20px;
    white-space:normal;
    text-shadow:2px2px2px red;
}
</style>
<head>
<body>
<p>文本设置：字体为绿色，行高为40px，左对齐，首行缩进20px，有阴影</p>
<div id="textCss">
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容，
    这是一个文本内容.
</div>
</body>
</html>
```

尺寸(Dimension)

height：设置元素的高度。height属性不包括填充，边框，或页边距！

max-height：设置元素的最大高度。max-height属性不包括填充，边框，或页边距！

max-width：设置元素的最大宽度。max-width属性不包括填充，边框，或页边距！

min-height：设置元素的最低高度。min-height属性不包括填充，边框，或页边距！

min-width：设置元素的最小宽度。min-width属性不包括填充，边框，或页边距！

width：设置元素的宽度。width属性不包括填充，边框和页边距！

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
.div1{
    width:200px;
    height:90px;
    background-color:red;
    float:left;
}
.div2{
        width:200px;
        min-height:90px;
        background-color:green;
        float:left;
    }
    .div3{
        width:200px;
        max-height:90px;
        background-color:blue;
        float:left;
    }
</style>
<head>
<body>
    <p>css尺寸：定义三块内容：宽度均为200px，高度一块设置为90px，一块设置为最小90，一块设置为最大90px</p>
    <div class="div1">
<p>这是固定大小的位置</p>
<p>这是固定大小的位置</p>
<p>这是固定大小的位置</p>
</div>
    <div class="div2">
<p>这是最小行高的位置</p>
<p>这是最小行高的位置</p>
<p>这是最小行高的位置</p>
</div>
    <div class="div3">
<p>这是最高行高的位置</p>
</div>
</body>
</html>
```

04 CSS定位(Positioning)

CSS 定位 (Positioning) 属性允许你对元素进行定位。

> **CSS 定位和浮动**

CSS 为定位和浮动提供了一些属性，利用这些属性，可以建立列式布局，将布局的一部分与另一部分重叠，还可以完成多年来通常需要使用多个表格才能完成的任务。

定位的基本思想很简单，它允许你定义元素框相对于其正常位置应该出现的位置，或者相对于父元素、另一个元素甚至浏览器窗口本身的位置。显然，这个功能非常强大，也很让人吃惊。要知道，用户代理对 CSS2 中定位的支持远胜于对其它方面的支持，对此不应感到奇怪。

另一方面，CSS1 中首次提出了浮动，它以 Netscape 在 Web 发展初期增加的一个功能为基础。浮动不完全是定位，不过，它当然也不是正常流布局。我们会在后面的章节中明确浮动的含义。

> **一切皆为框**

div、h1 或 p 元素常常被称为块级元素。这意味着这些元素显示为一块内容，即"块框"。与之相反，span 和 strong 等元素称为"行内元素"，这是因为它们的内容显示在行中，即"行内框"。

您可以使用 display 属性改变生成的框的类型。这意味着，通过将 display 属性设置为 block，可以让行内元素（比如 <a> 元素）表现得像块级元素一样。还可以通过把 display 设置为 none，让生成的元素根本没有框。这样的话，该框及其所有内容就不再显示，不占用文档中的空间。

但是在一种情况下，即使没有进行显式定义，也会创建块级元素。这种情况发生在把一些文本添加到一个块级元素（比如 div）的开头。即使没有把这些文本定义为段落，它也会被当作段落对待：

```
<div>
div context
<p>p context.</p>
</div>
```

在这种情况下，这个框称为无名块框，因为它不与专门定义的元素相关联。

块级元素的文本行也会发生类似的情况。假设有一个包含三行文本的段落。每行文本形成一个无名框。无法直接对无名块或行框应用样式，因为没有可以应用样式的地方（注意，行框和行内框是两个概念）。但是，这有助于理解在屏幕上看到的所有东西都形成某种框。

CSS 定位机制

CSS 有三种基本的定位机制：普通流、浮动和绝对定位。

除非专门指定，否则所有框都在普通流中定位。也就是说，普通流中的元素的位置由元素在 (X)HTML 中的位置决定。

块级框从上到下一个接一个地排列，框之间的垂直距离是由框的垂直外边距计算出来。

行内框在一行中水平布置。可以使用水平内边距、边框和外边距调整它们的间距。但是，垂直内边距、边框和外边距不影响行内框的高度。由一行形成的水平框称为行框（Line Box），行框的高度总是足以容纳它包含的所有行内框。不过，设置行高可以增加这个框的高度。

在下面的章节，我们会为您详细讲解相对定位、绝对定位和浮动。

> **CSS position 属性**

通过使用 position 属性，我们可以选择 4 种不同类型的定位，这会影响元素框生成的方式。

position 属性值的含义：

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，置于其父元素中。

relative：元素框偏移某个距离。元素仍保持其未定位前的形状，它原本所占的空间仍保留。

absolute：元素框从文档流完全删除，并相对于其包含块定位。包含块可能是文档中的另一个元素或者是初始包含块。元素原先在正常文档流中所占的空间会关闭，就好像元素原来不存在一样。元素定位后生成一个块级框，而不论原来它在正常流中生成何种类型的框。

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。

提示：相对定位实际上被看作普通流定位模型的一部分，因为元素的位置相对于它在普通流中的位置。

> **CSS 定位属性**

- bottom：对于绝对定位元素，bottom属性设置单位高于/低于包含它的元素的底边。

  对于相对定位元素，bottom属性设置单位高于/低于其正常位置的元素的底边。
- right：right 属性规定元素的右边缘。该属性定义了定位元素右外边距边界与其包含块右边界之间的偏移。
- top：top 属性规定元素的顶部边缘。该属性定义了一个定位元素的上外边距边界与其包含块上边界之间的偏移。
- left：left 属性规定元素的左边缘。该属性定义了定位元素左外边距边界与其包含块左边界之间的偏移。
- clear：指定段落的左侧或右侧不允许浮动的元素。
- display：规定元素应该生成的框的类型。
- float：指定一个盒子（元素）是否应该浮动。
- overflow：指定如果内容溢出一个元素的框，会发生什么。
- position：指定一个元素（静态的，相对的，绝对或固定）的定位方法的类型。
- visibility：指定一个元素是否是可见的。
- z-index：指定一个元素的堆叠顺序。

  拥有更高堆叠顺序的元素总是会处于堆叠顺序较低的元素的前面。

CSS 相对定位

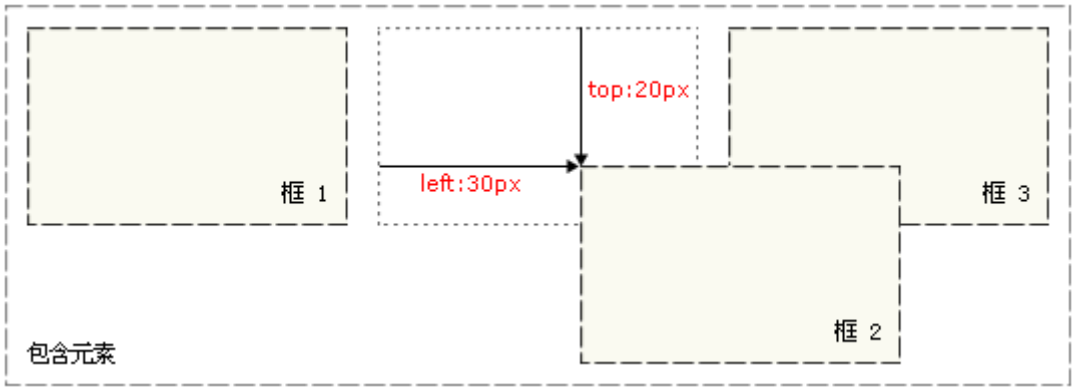
设置为相对定位的元素框会偏移某个距离。元素仍然保持其未定位前的形状，它原本所占的空间仍保留。

相对定位是一个非常容易掌握的概念。如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素"相对于"它的起点进行移动。

如果将 top 设置为 20px，那么框将在原位置顶部下面 20 像素的地方。如果 left 设置为 30 像素，那么会在元素左边创建 30 像素的空间，也就是将元素向右移动。

```
#box_relative {
 position: relative;
 left: 30px;
 top: 20px;
}
```

如下图所示：



注意，在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框。

```
<html>
<head>
    <meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
    <style type="text/css">
    #d1{
    width:100px;
    height:100px;
    background-color:red;
    float:left;
    }
    #d2{
    width:100px;
    height:100px;
    background-color:green;
    position:relative;
    left:30px;
    top:20px;
    float:left;
    }
    #d3{
    width:100px;
    height:100px;
    background-color:blue;
    float:left;
    }
    </style>
</head>
<body>
    <div id="d1"></div>
    <div id="d2"></div>
    <div id="d3"></div>
</body>
</html>
```

CSS 绝对定位

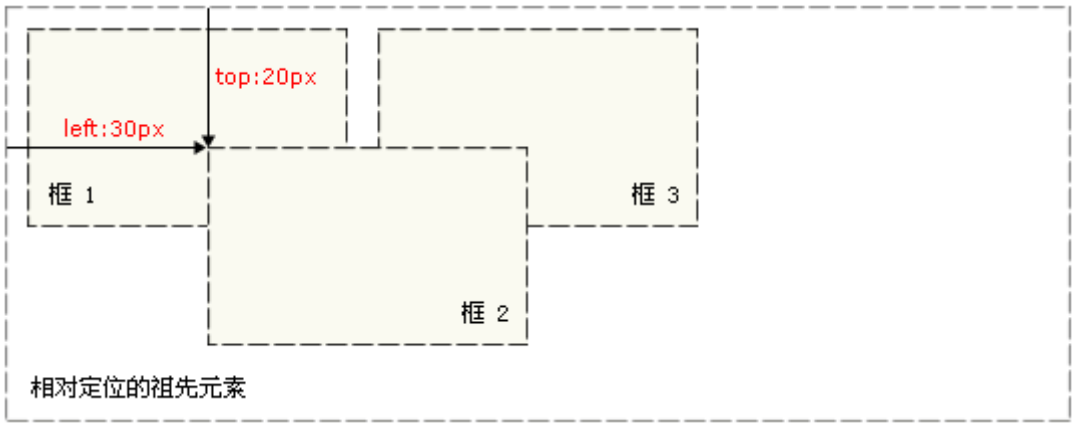
设置为绝对定位的元素框从文档流完全删除，并相对于其包含块定位，包含块可能是文档中的另一个元素或者是初始包含块。元素原先在正常文档流中所占的空间会关闭，就好像该元素原来不存在一样。元素定位后生成一个块级框，而不论原来它在正常流中生成何种类型的框。

绝对定位使元素的位置与文档流无关，因此不占据空间。这一点与相对定位不同，相对定位实际上被看作普通流定位模型的一部分，因为元素的位置相对于它在普通流中的位置。

普通流中其它元素的布局就像绝对定位的元素不存在一样：

```
#box_relative {
 position: absolute;
 left: 30px;
 top: 20px;
}
```

如下图所示：



绝对定位的元素的位置相对于最近的已定位祖先元素，如果元素没有已定位的祖先元素，那么它的位置相对于最初的包含块。

对于定位的主要问题是要记住每种定位的意义。所以，现在让我们复习一下学过的知识吧：相对定位是"相对于"元素在文档中的初始位置，而绝对定位是"相对于"最近的已定位祖先元素，如果不存在已定位的祖先元素，那么"相对于"最初的包含块。

注释：根据用户代理的不同，最初的包含块可能是画布或 HTML 元素。

提示：因为绝对定位的框与文档流无关，所以它们可以覆盖页面上的其它元素。可以通过设置 z-index 属性来控制这些框的堆放次序。

```
<html>
<head>
    <meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
    <style type="text/css">
    #d1{
    width:100px;
    height:100px;
    background-color:red;
    float:left;
    }
    #d2{
    width:100px;
    height:100px;
    background-color:green;
    position:absolute;
```

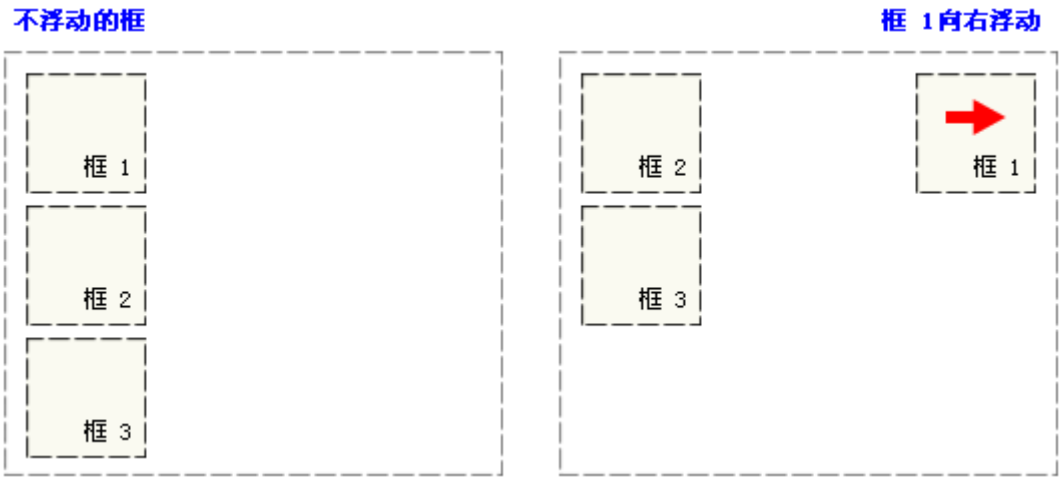
```
    left:30px;
    top:20px;
    float:left;
    }
    #d3{
    width:100px;
    height:100px;
    background-color:blue;
    float:left;
    }
    </style>
</head>
<body>
    <div id="d1"></div>
    <div id="d2"></div>
    <div id="d3"></div>
</body>
</html>
```

CSS 浮动

浮动的框可以向左或向右移动，直到它的外边缘碰到包含框或另一个浮动框的边框为止。

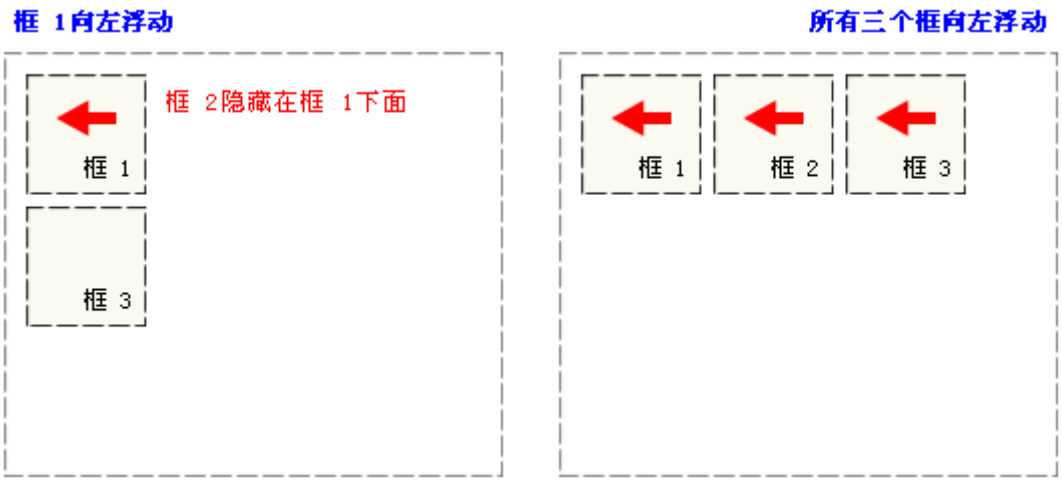
由于浮动框不在文档的普通流中，所以文档的普通流中的块框表现得就像浮动框不存在一样。

请看下图，当把框 1 向右浮动时，它脱离文档流并且向右移动，直到它的右边缘碰到包含框的右边缘：

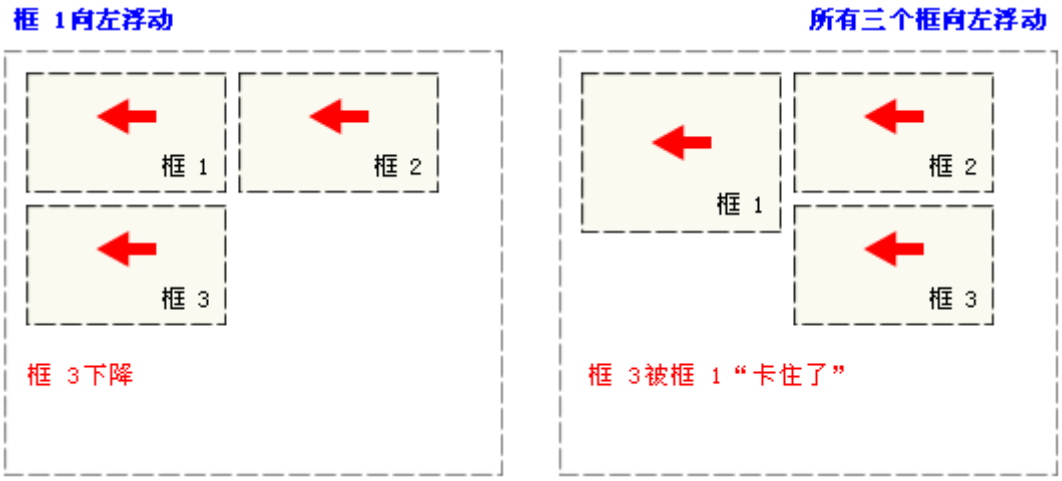


再请看下图，当框 1 向左浮动时，它脱离文档流并且向左移动，直到它的左边缘碰到包含框的左边缘。因为它不再处于文档流中，所以它不占据空间，实际上覆盖住了框 2，使框 2 从视图中消失。

如果把所有三个框都向左移动，那么框 1 向左浮动直到碰到包含框，另外两个框向左浮动直到碰到前一个浮动框。



如下图所示，如果包含框太窄，无法容纳水平排列的三个浮动元素，那么其它浮动块向下移动，直到有足够的空间。如果浮动元素的高度不同，那么当它们向下移动时可能被其它浮动元素"卡住"：



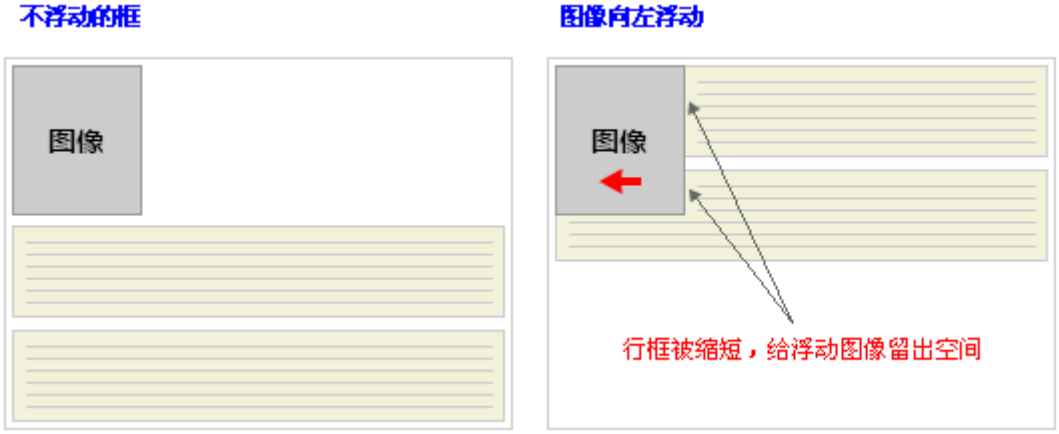
CSS float 属性

在 CSS 中，我们通过 float 属性实现元素的浮动。

行框和清理

浮动框旁边的行框被缩短，从而给浮动框留出空间，行框围绕浮动框。

因此，创建浮动框可以使文本围绕图像：



要想阻止行框围绕浮动框，需要对该框应用 clear 属性。clear 属性的值可以是 left、right、both 或 none，它表示框的哪些边不应该挨着浮动框。

为了实现这种效果，在被清理的元素的上外边距上添加足够的空间，使元素的顶边缘垂直下降到浮动框下面：

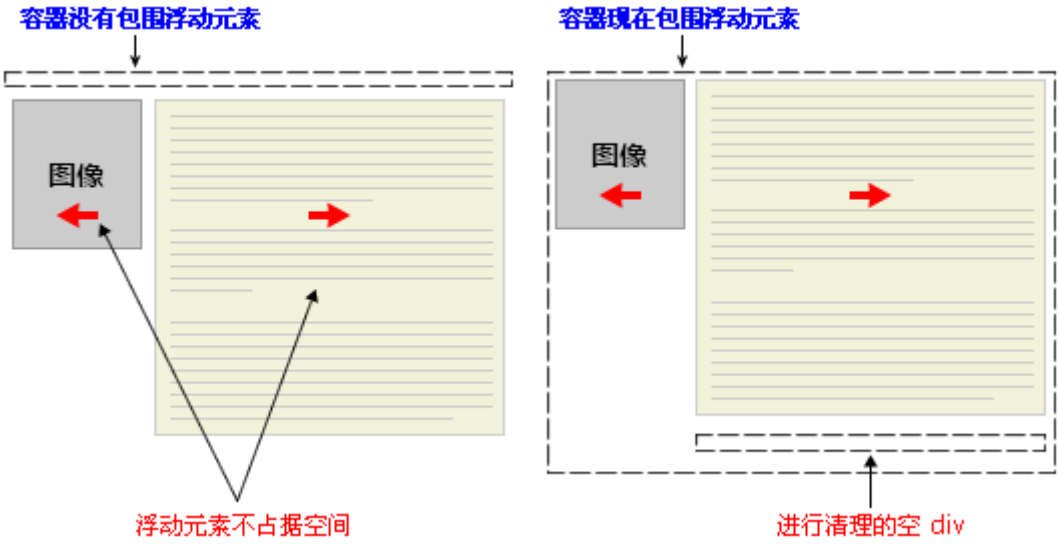
这是一个有用的工具，它让周围的元素为浮动元素留出空间。

让我们更详细地看看浮动和清理。假设希望让一个图片浮动到文本块的左边，并且希望这幅图片和文本包含在另一个具有背景颜色和边框的元素中。您可能编写下面的代码：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <style>
            .news{
                background-color: gray;
                border:solid 1px black;
            }
            .newsimg{
                float:left;
            }
            .newsp{
                float:right;
            }
        </style>
    </head>
    <body>
        <div class="news">
            <img width="100px"height="100px"src="images/renwu.jpg"/>
            <p>some text</p>
        </div>
    </body>
</html>
```

这种情况下，出现了一个问题。因为浮动元素脱离了文档流，所以包围图片和文本的 div 不占据空间。

如何让包围元素在视觉上包围浮动元素呢？需要在这个元素中的某个地方应用 clear：



不幸的是出现了一个新的问题，由于没有现有的元素可以应用清理，所以我们只能添加一个空元素并且清理它。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <style>
            .news{
                background-color: gray;
                border:solid1px black;
            }
            .news img{
                float:left;
            }
            .clear{
                clear:both;
            }
        </style>
    </head>
    <body>
    <div class="news">
    <img width="100px"height="100px"src="images/renwu.jpg"/>
    <p>some text</p><div class="clear"></div>
    </div>
    </body>
</html>
```

这样可以实现我们希望的效果，但是需要添加多余的代码。常常有元素可以应用 clear，但是有时候不得不为了进行布局而添加无意义的标记。

不过我们还有另一种办法，那就是对容器 div 进行浮动：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
```

```
        <style>
            .news{
            background-color: gray;
            border:solid1px black;
            float:left;
            }
            .news img{
            float:left;
            }
            .news p{
            float:right;
            }
        </style>
    </head>
    <body>
        <div class="news">
            <img width="100px"height="100px"src="images/renwu.jpg"/>
            <p>some text</p>
        </div>
    </body>
</html>
```

这样会得到我们希望的效果。不幸的是，下一个元素会受到这个浮动元素的影响。为了解决这个问题，有些人选择对布局中的所有东西进行浮动，然后使用适当的有意义的元素（常常是站点的页脚）对这些浮动进行清理。这有助于减少或消除不必要的标记。

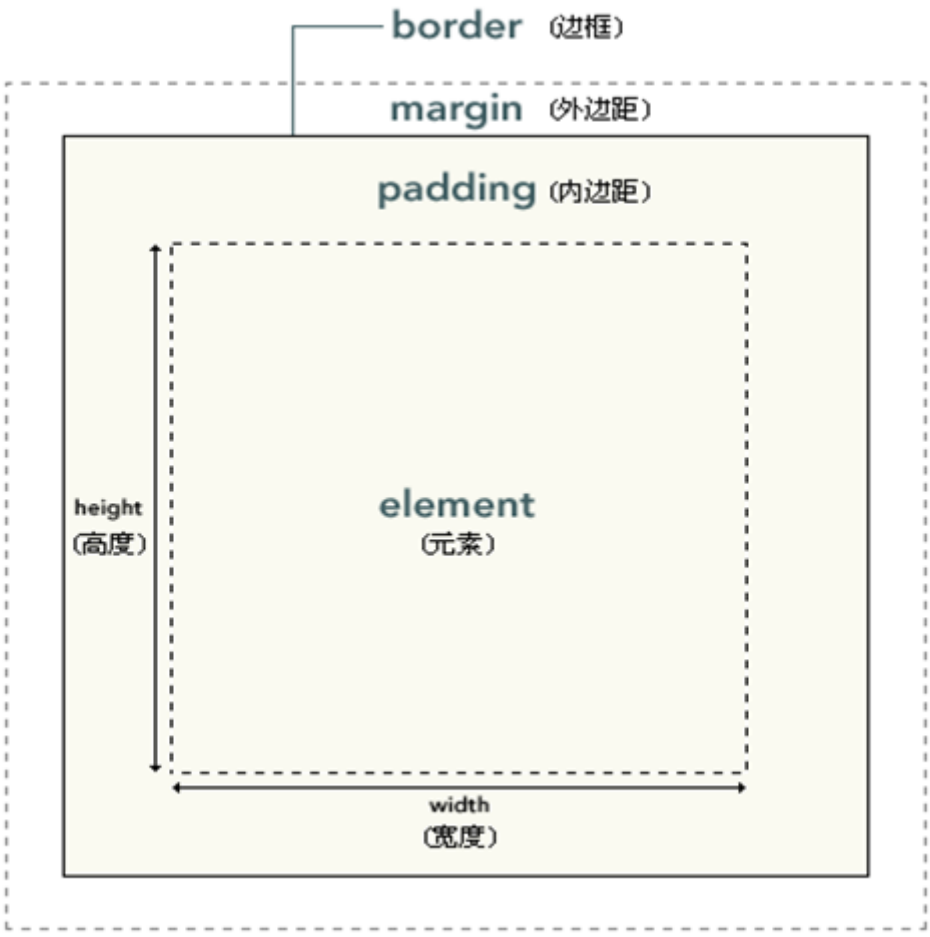
综合案例

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
#left{
    width:30%;
    height:200px;
    background-color:red;
    float:left;
}
#center{
    width:30%;
    height:200px;
    background-color:green;
    float:left;
}
#right{
    width:30%;
    height:200px;
    background-color:blue;
    float:left;
}
img{
    position:absolute;
    right:100px;
    z-index:-1
```

```
}
#center span{
    visibility:hidden
}
</style>
<head>
<body>
    <p>定位设置：文档内容分左中右三部分，且右侧图片位于文档内容之下。</p>
    <div id="left">这是左侧文档内容</div>
    <div id="center">这是<span>中间的</span>文档内容</div>
    <div id="right">这是右侧的文档内容</div>
    <img src="images/beijing.jpg"width="100px"height="100px"/>
</body>
</html>
```

05 CSS框模型

CSS 框模型 (Box Model) 规定了元素框处理元素内容、内边距、边框 和 外边距 的方式。



元素框的最内部分是实际的内容，直接包围内容的是内边距。内边距呈现了元素的背景。内边距的边缘是边框。边框以外是外边距，外边距默认是透明的，因此不会遮挡其后的任何元素。

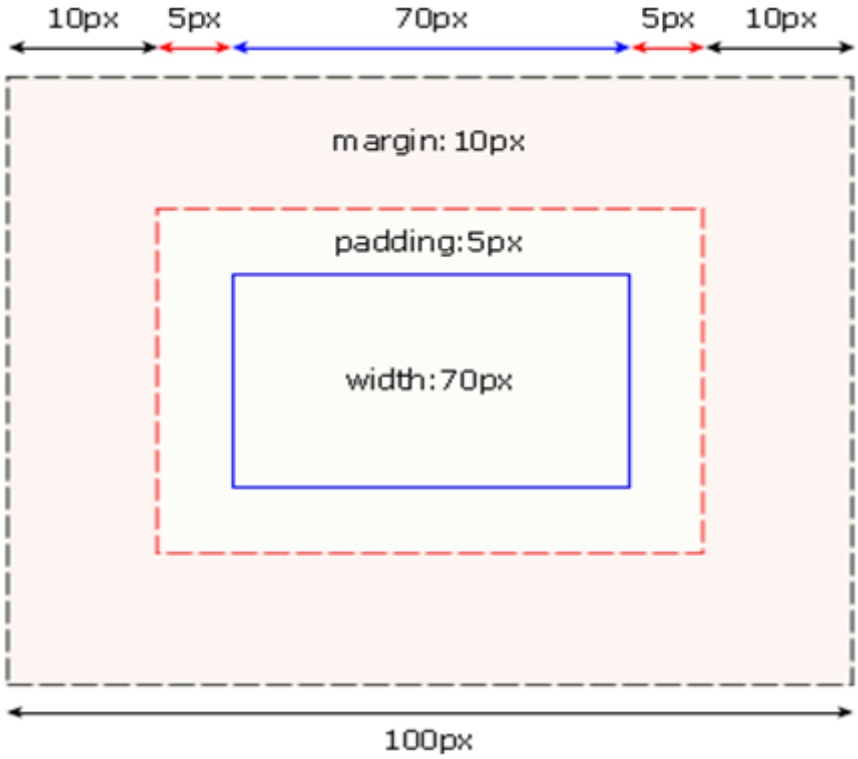
提示：背景应用于由内容和内边距、边框组成的区域。

内边距、边框和外边距都是可选的，默认值是零。但是，许多元素将由用户代理样式表设置外边距和内边距。可以通过将元素的 margin 和 padding 设置为零来覆盖这些浏览器样式。这可以分别进行，也可以使用通用选择器对所有元素进行设置：

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

在 CSS 中，width 和 height 指的是内容区域的宽度和高度。增加内边距、边框和外边距不会影响内容区域的尺寸，但是会增加元素框的总尺寸。

假设框的每个边上有 10 个像素的外边距和 5 个像素的内边距。如果希望这个元素框达到 100 个像素，就需要将内容的宽度设置为 70 像素，请看下图：




```
#box{
width:70px;
margin:10px;
padding:5px;
}
```

提示：内边距、边框和外边距可以应用于一个元素的所有边，也可以应用于单独的边。

提示：外边距可以是负值，而且在很多情况下都要使用负值的外边距。

> **浏览器兼容性**

一旦为页面设置了恰当的 DTD，大多数浏览器都会按照上面的图示来呈现内容。然而 IE 5 和 6 的呈现却是不正确的。根据 W3C 的规范，元素内容占据的空间是由 width 属性设置的，而内容周围的 padding 和 border 值是另外计算的。不幸的是，IE5.X 和 6 在怪异模式中使用自己的非标准模型。这些浏览器的 width 属性不是内容的宽度，而是内容、内边距和边框的宽度的总和。

虽然有方法解决这个问题。但是目前最好的解决方案是回避这个问题。也就是，不要给元素添加具有指定宽度的内边距，而是尝试将内边距或外边距添加到元素的父元素和子元素。

内边距

元素的内边距在边框和内容区之间。控制该区域最简单的属性是 padding 属性。

CSS padding 属性定义元素边框与元素内容之间的空白区域。

padding:padding 简写属性在一个声明中设置所有填充属性（内边距）。padding 属性接受长度值或百分比值。该属性可以有1到4个值。均不可以为负值。

> **用法**

padding: 上内边距 右内边距 下内边距 左内边距

```
padding:10px 5px 15px 20px;
    /*上内边距是 10px
    右内边距是 5px
    下内边距是 15px
    左内边距是 20px*/
```

padding: 上内边距 右内边距和左内边距 下内边距

```
padding:10px 5px 15px;
    /* 上内边距是 10px
    右内边距和左内边距是 5px
    下内边距是 15px */
```

padding: 上内边距和下内边距 右内边距和左内边距

```
 padding:10px 5px;
    /*上内边距和下内边距是 10px
    右内边距和左内边距是 5px*/
```

padding: 上右下左四个内边距

```
 padding:10px;
/*所有四个内边距都是 10px*/
```

padding-bottom：设置元素的底部内边距（底部空白）。

padding-left：设置一个元素的左内边距（空格）。

padding-right：设置一个元素的右内边距（空白）。

padding-top：设置一个元素的顶部内边距（空白）。

```
img{
/*  padding:20px 30px 10px 40px;  上 右 下 左*/
    padding-left:40px;
    padding-top:20px;
    padding-right:30px;
    padding-bottom:10px;
}
```

边框

元素的边框 (border) 是围绕元素内容和内边距的一条或多条线。

CSS border 属性允许你规定元素边框的样式、宽度和颜色。

在 HTML 中，我们使用表格来创建文本周围的边框，但是通过使用 CSS 边框属性，我们可以创建出效果出色的边框，并且可以应用于任何元素。

元素外边距内就是元素的的边框 (border)。元素的边框就是围绕元素内容和内边据的一条或多条线。

每个边框有 3 个方面：宽度、样式，以及颜色。

可以设置的属性（按顺序）：border-width, border-style,和border-color.

如果上述值缺少一个没有关系。

也可对边框的四个属性分别设置border-left, border-top, border-right, border-bottom.

border-width 指定边框的宽度

border-style 指定边框的样式

border-color 指定边框的颜色

```
p
{
border:5px solid red;
}
等同于
p
{
border-width:5px;
border-style:solid;
border-color: red;
}
```

由于 border-style 的默认值是 none，如果没有声明样式，就相当于 border-style: none。因此，如果您希望边框出现，就必须声明一个边框样式。

也可单独对边框的上下左右四个位置进行分别设置，四个位置为border-left，border-top，border-right，border-bottom，其中每个位置的宽度，样式，颜色都可以单独设置。

border-left:把左边框的所有属性设置到一个声明中。

可以按顺序设置如下属性： border-left-width, border-left-style, border-left-color.

```
p
{
border-left:5pxdouble#ff0000;
}
等同于
p
{
border-style:solid;
border-left-width:5px;
border-left-style:double;
border-left-color:#ff0000;
}
```

border-top把上边框的所有属性设置到一个声明中

> 可以按顺序设置如下属性：
>
> border-top-width, border-top-style, and border-top-color.

具体用法参照border-left

border-right把左边框的所有属性设置到一个声明中。

> 可以按顺序设置如下属性：
>
> border-right-width, border-right-style, and border-right-color.

具体用法参照border-left

border-bottom：设置一个声明中所有底部边框属性。

> 可以按顺序设置如下属性：
>
> border-bottom-width, border-bottom-style,和border-bottom-color.

具体用法参照border-left

border-radius设置四个 border-*-radius 属性。该属性允许您为元素添加圆角边框！

```
div
{
border:2pxsolid;
border-radius:25px;
}
```

边框与背景

CSS 规范指出，边框绘制在"元素的背景之上"。这很重要，因为有些边框是"间断的"（例如，点线边框或虚线框），元素的背景应当出现在边框的可见部分之间。

CSS2 指出背景只延伸到内边距，而不是边框。后来 CSS2.1 进行了更正：元素的背景是内容、内边距和边框区的背景。大多数浏览器都遵循 CSS2.1 定义，不过一些较老的浏览器可能会有不同的表现。

CSS 边框属性

属性	描述
border	简写属性，用于把针对四个边的属性设置在一个声明。
border-style	用于设置元素所有边框的样式，或者单独地为各边设置边框样式。
border-width	简写属性，用于为元素的所有边框设置宽度，或者单独地为各边边框设置宽度。
border-color	简写属性，设置元素的所有边框中可见部分的颜色，或为 4 个边分别设置颜色。
border-bottom	简写属性，用于把下边框的所有属性设置到一个声明中。
border-bottom-color	设置元素的下边框的颜色。
border-bottom-style	设置元素的下边框的样式。
border-bottom-width	设置元素的下边框的宽度。
border-left	简写属性，用于把左边框的所有属性设置到一个声明中。
border-left-color	设置元素的左边框的颜色。
border-left-style	设置元素的左边框的样式。
border-left-width	设置元素的左边框的宽度。
border-right	简写属性，用于把右边框的所有属性设置到一个声明中。
border-right-color	设置元素的右边框的颜色。
border-right-style	设置元素的右边框的样式。
border-right-width	设置元素的右边框的宽度。
border-top	简写属性，用于把上边框的所有属性设置到一个声明中。
border-top-color	设置元素的上边框的颜色。
border-top-style	设置元素的上边框的样式。
border-top-width	设置元素的上边框的宽度。

轮廓(Outline)

outline（轮廓）是绘制于元素周围的一条线，位于边框边缘的外围，可起到突出元素的作用。

outline简写属性在一个声明中设置所有的轮廓属性。

> 可以设置的属性分别是（按顺序）：

outline-color, outline-style, outline-width

如果不设置其中的某个值，也不会出问题，比如 outline:solid #ff0000; 也是允许的。

outline-color：指定轮廓颜色

outline-style：指定轮廓样式

outline-width：指定轮廓的宽度

```
p{
outline:yellow dotted 10px;
}
等同于
P{
    outline-color:yellow;
    outline-style:dotted;
    outline-width:10px;
}
```

外边距

围绕在元素边框的空白区域是外边距。设置外边距会在元素外创建额外的"空白"。

设置外边距的最简单的方法就是使用 margin 属性，这个属性接受任何长度单位、百分数值甚至负值。

设置外边距的最简单的方法就是使用 margin 属性。该属性可以有1到4个值。可以为负值。

> **用法**

margin: 上外边距 右外边距 下外边距 左外边距

```
margin:10px 5px 15px 20px;
  /*上外边距是 10px
  右外边距是 5px
  下外边距是 15px
  左外边距是 20px */
```

margin:上外边距 右外边距和左外边距 下外边距

```
 margin:10px 5px 15px;
    /* 上外边距是 10px
    右外边距和左外边距是 5px
    下外边距是 15px */
```

margin:上外边距和下外边距 右外边距和左外边距

```
margin:10px 5px;
    /* 上外边距和下外边距是 10px
        右外边距和左外边距是 5px
    */
```

margin:上右下左四个外边距

```
margin:10px;
   /*
   所有四个外边距都是 10px
   */
```

margin-bottom：设置元素的下外边距。

margin-left：设置元素的左外边距。

margin-right：设置元素的右外边距。

margin-top设置元素的上外边距。

```
img{
/*  margin:30px 40px 30px 40px    上 右 下 左; */
    margin-left:40px;
    margin-top:30px;
    margin-right:40px;
    margin-bottom:30px;
}
```

外边距合并

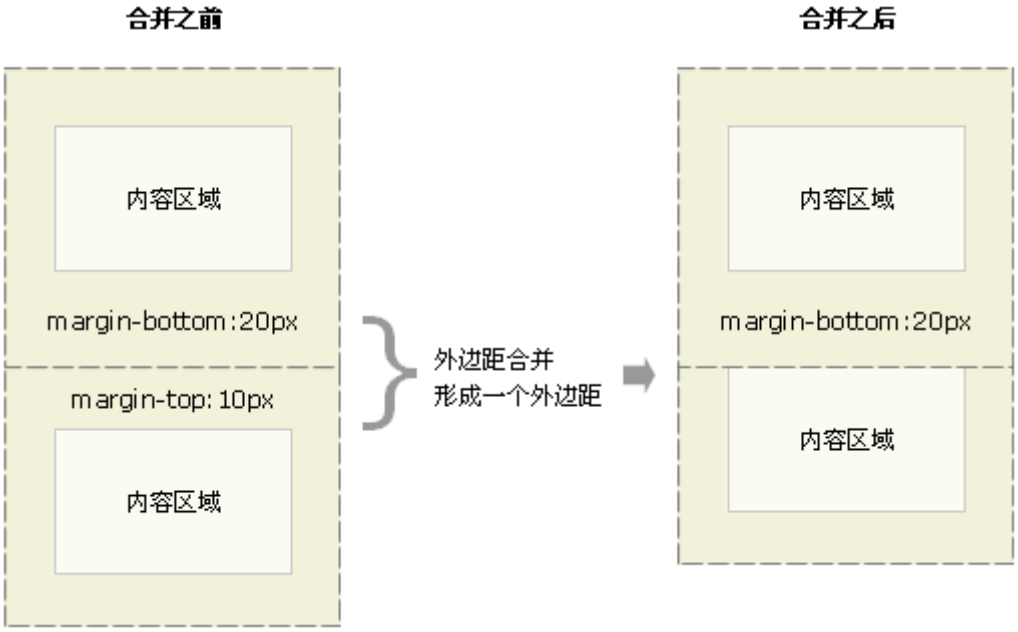
外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。

合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。

外边距合并（叠加）是一个相当简单的概念。但是，在实践中对网页进行布局时，它会造成许多混淆。

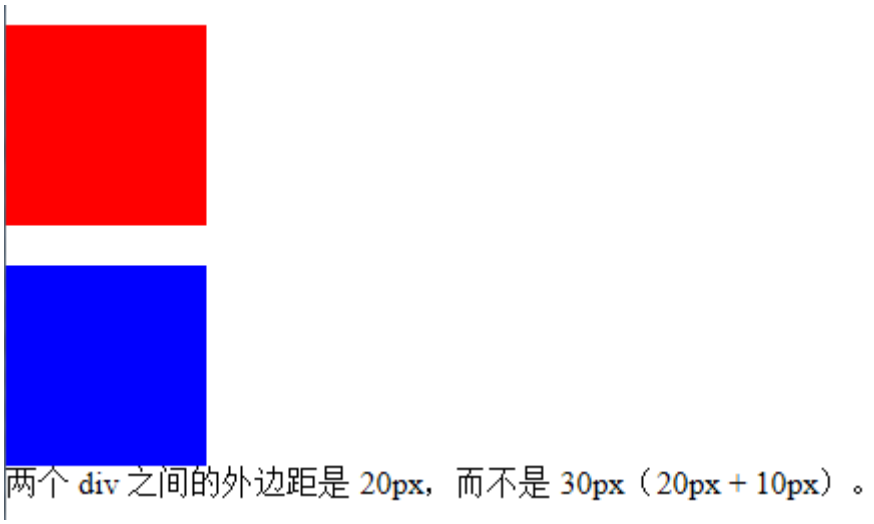
简单地说，外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。

当一个元素出现在另一个元素上面时，第一个元素的下外边距与第二个元素的上外边距会发生合并。如下图：

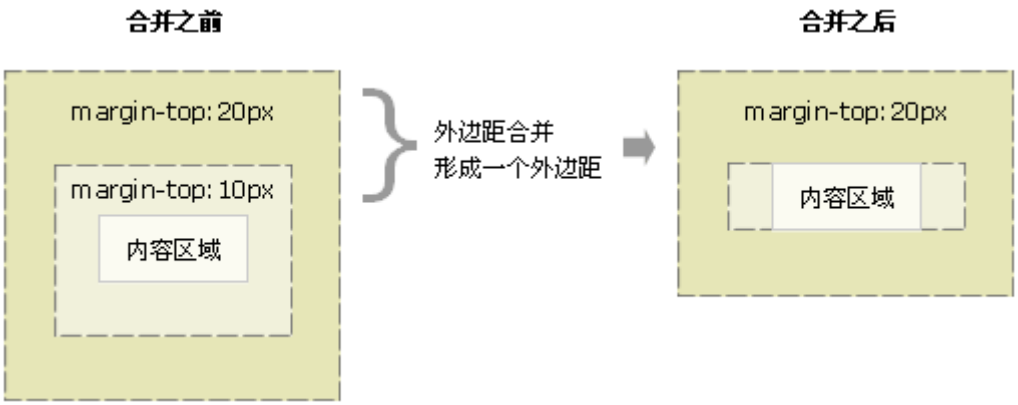


```
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
    <style type="text/css">
    *{
    margin:0;
    padding:0;
    border:0;
    }
    #d1{
    width:100px;
    height:100px;
    margin-top:20px;
    margin-bottom:20px;
    background-color:red;
    }
    #d2{
    width:100px;
    height:100px;
    margin-top:10px;
    background-color:blue;
    }
    </style>
</head>
<body>
    <div id="d1"></div>
    <div id="d2"></div>
    <p>两个 div 之间的外边距是 20px，而不是 30px（20px + 10px）。</p>
</body>
</html>
```

显示如下：



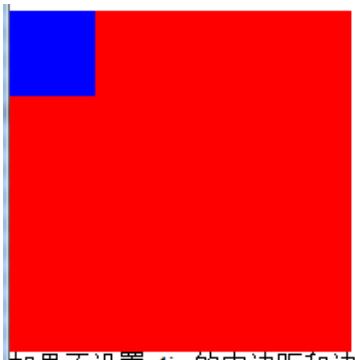
当一个元素包含在另一个元素中时（假设没有内边距或边框把外边距分隔开），它们的上和/或下外边距也会发生合并。请看下图：




```
<html>
<head>
    <meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
    <style type="text/css">
    *{
    margin:0;
    padding:0;
    border:0;
    }
    #outer{
    width:200px;
    height:200px;
    background-color:red;
    margin-top:20px;
    }

    #inner{
    width:50px;
    height:50px;
    background-color:blue;
    margin-top:10px;
    }
    </style>
</head>
<body>
    <div id="outer">
    <div id="inner">
    </div>
    </div>
    <p>如果不设置 div 的内边距和边框，那么内部 div 的上外边距将与外部 div 的上外边距合并（叠加）。</p>
</body>
</html>
```

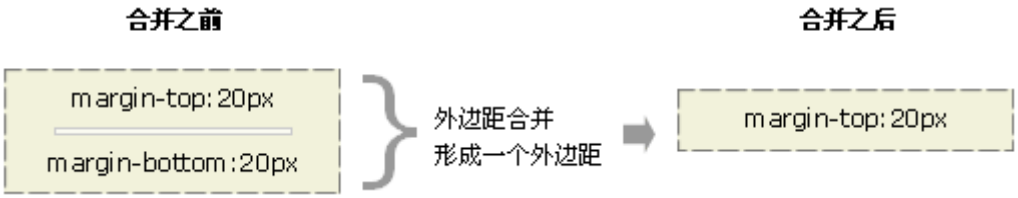
显示如下：



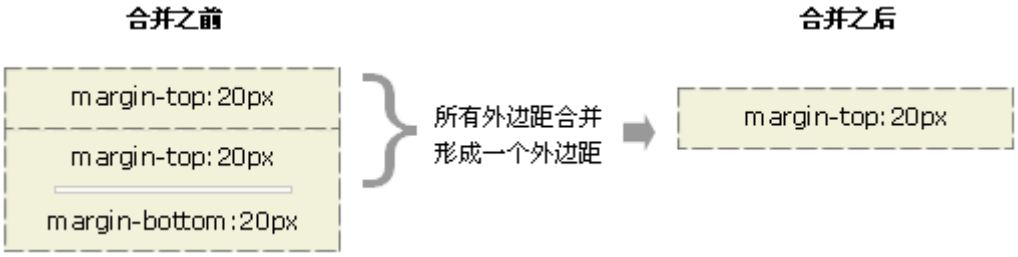
如果不设置 div 的内边距和边框，那么内部 div 的上外边距将与外部 div 的上外边距合并（叠加）。

尽管看上去有些奇怪，但是外边距甚至可以与自身发生合并。

假设有一个空元素，它有外边距，但是没有边框或填充。在这种情况下，上外边距与下外边距就碰到了一起，它们会发生合并：

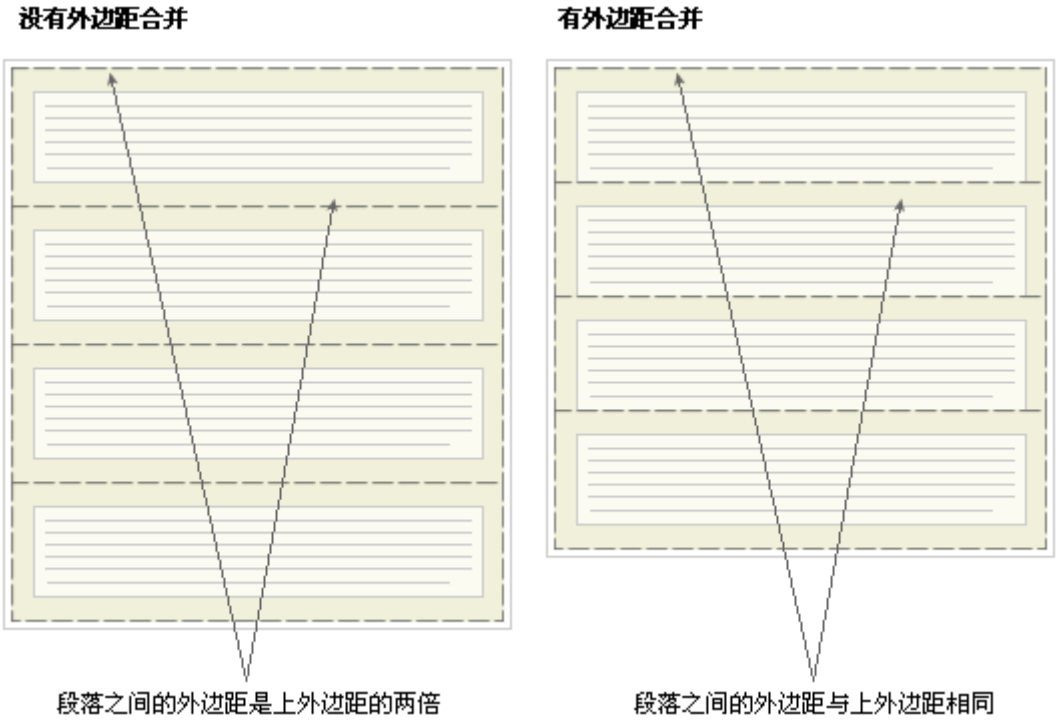


如果这个外边距遇到另一个元素的外边距，它还会发生合并：



这就是一系列的段落元素占用空间非常小的原因，因为它们的所有外边距都合并到一起，形成了一个小的外边距。

外边距合并初看上去可能有点奇怪，但是实际上，它是有意义的。以由几个段落组成的典型文本页面为例。第一个段落上面的空间等于段落的上外边距。如果没有外边距合并，后续所有段落之间的外边距都将是相邻上外边距和下外边距的和。这意味着段落之间的空间是页面顶部的两倍。如果发生外边距合并，段落之间的上外边距和下外边距就合并在一起，这样各处的距离就一致了。



注释：只有普通文档流中块框的垂直外边距才会发生外边距合并。行内框、浮动框或绝对定位之间的外边距不会合并。

综合案例

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"content="text/html; charset=UTF-8"/>
<style type="text/css">
.boxDiv{
    border:5px solid green;
}
img{
/*  border:5px solid red; */
    border-width:5px;
    border-style:solid;
    border-color:red;

/*  outline:yellow dotted 10px; */
    outline-color:yellow;
    outline-style:dotted;
    outline-width:10px;

/*  padding:20px 30px 10px 40px;  上 右 下 左*/
    padding-left:40px;
    padding-top:20px;
    padding-right:30px;
    padding-bottom:10px;

/*  margin:30px 40px 30px 40px   上 右 下 左; */
    margin-left:40px;
    margin-top:30px;
    margin-right:40px;
    margin-bottom:30px;

/*  border-radius:100px;  */
```

```
    border-radius:100px100px100px100px;    /*左上 右上 右下 左下 */
    float:center;
}
</style>
<head>
<body>
<p>设置图片边框为圆角边框弧度为20px，距外边框为30px，内边框为20px,设置外边框轮廓为点状</p>
<div class="boxDiv">
    <img src="images/renwu.jpg">
</div>
</body>
</html>
```

显示如下：

设置图片边框为圆角边框弧度为20px，距外边框为30px，内边框为20px,设置外边框轮廓为点状