网易微专业之《前端开发工程师》

学习笔记

开始时间：2015.12.17

《页面制作》

CSS（四）

盒模型

在“CSS 盒子模型”理论中，所有页面中的元素都可以看成一个盒子，并且占据着一定的页面空间。一个页面由很多这样的盒子组成，这些盒子之间会互相影响，因此掌握盒子模型需要从两个方面来理解：一是理解单独一个盒子的内部结构，二是理解多个盒子之间的相互关系。

每个元素都看成一个盒子，盒子模型是由 content（内容）、padding（内边距）、margin（外边距）和 border（边框）这四个属性组成的。此外，在盒子模型中，还有宽度 width 和高度 height 两大辅助性属性。

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。

记住，是所有的元素都可以看做一个盒子！

下图为一个 CSS 盒子模型的内部结构：

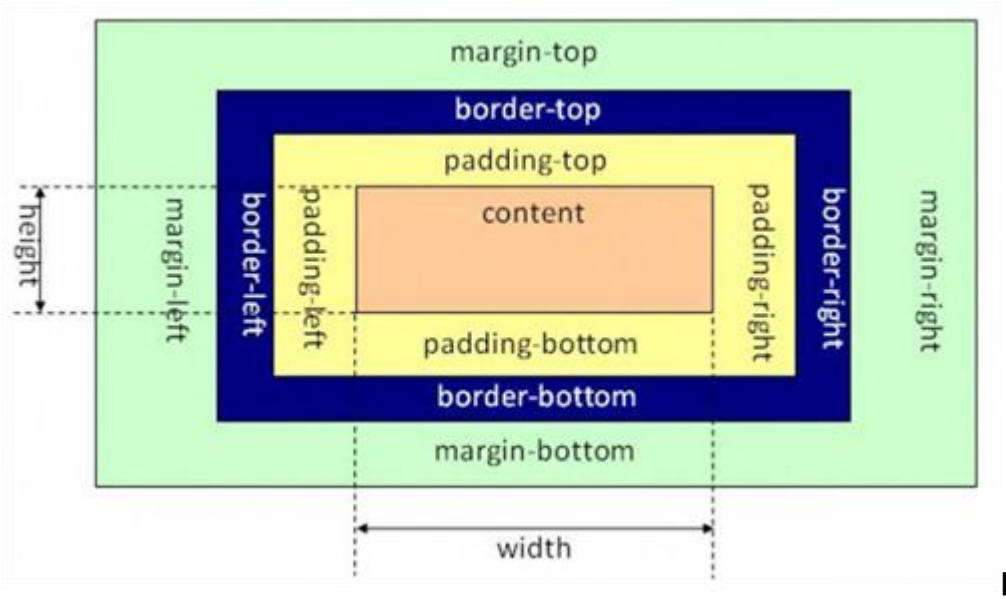


图 1 CSS 盒子模型

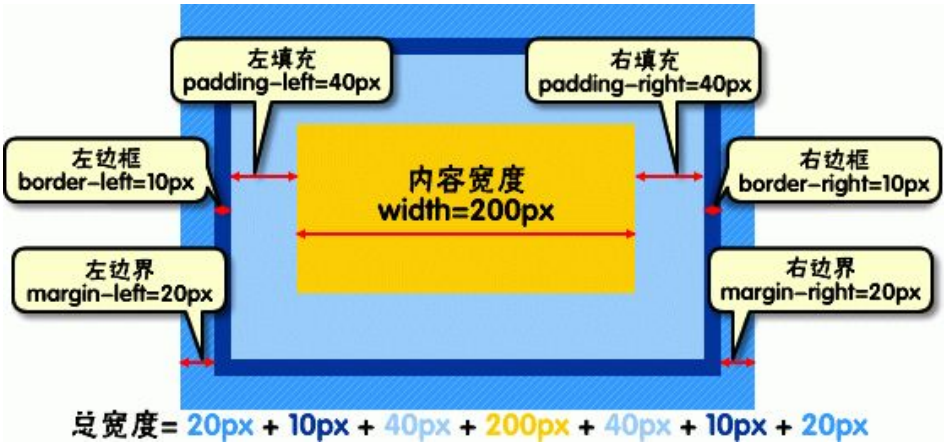
从上图中我们可以得出盒子模型的属性如下：

表 1 盒子模型 4 个属性

属性	说明
border	（边框）元素边框
margin	（外边距）用于定义页面中元素与元素之间的距离
padding	（内边距）用于定义内容与边框之间的距离
content	（内容）可以是文字或图片

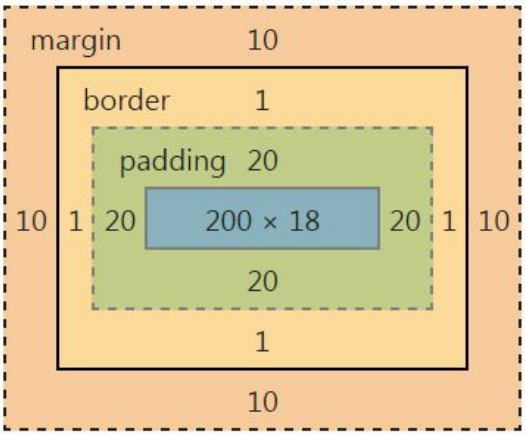
盒模型--宽度(width)和高度(height)

盒模型宽度和高度和我们平常所说的物体的宽度和高度理解是不一样的，css 内定义的宽（width）和高（height），指的是填充以里的内容范围。
因此一个元素实际宽度（盒子的宽度）=左边界+左边框+左填充+内容宽度+右填充+右边框+右边界。



元素的高度也是同理。

元素盒模型参考图：



盒模型--填充(padding)

元素内容与边框之间是可以设置距离的，称之为“填充”。padding 属性接受长度值或百分比值，但不允许使用负值。填充也可分为上、右、下、左(顺时针)。如下代码：

```
div{padding:20px 10px 15px 30px;}
```

顺序一定不要搞混。可以分开写上面代码：

```
div{
    padding-top:20px;
    padding-right:10px;
    padding-bottom:15px;
    padding-left:30px;
}
```

如果上、右、下、左的填充都为 10px;可以这么写

```
div{padding:10px;}
```

如果上下填充一样为 10px，左右一样为 20px，可以这么写：

```
div{padding:10px 20px;}
```

CSS 内边距属性

属性	描述
padding	简写属性。作用是在一个声明中设置元素的所内边距属性。
padding-bottom	设置元素的下内边距。
padding-left	设置元素的左内边距。

padding-right	设置元素的右内边距。
padding-top	设置元素的上内边距。

盒模型--边框(border)

盒子模型的边框就是围绕着内容及补白的线，这条线你可以设置它的粗细、样式和颜色(边框三个属性)。

如下面代码为 div 来设置边框粗细为 2px、样式为实心的、颜色为红色的边框：

```
div{
    border:2px  solid  red;
}
```

- border-style（边框样式）常见样式有:
  dashed（虚线）| dotted（点线）| solid（实线）。
- border-color（边框颜色）中的颜色可设置为十六进制颜色，如:border-color:#888;
- border-width（边框宽度）中的宽度也可以设置为：thin | medium | thick（但不是很常用），最常还是用象素（px）。

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。

盒模型--边界(margin)

元素与其它元素之间的距离可以使用边界（margin）来设置。

这个属性接受任何长度单位、百分数值甚至负值。

属性	描述
margin	简写属性。在一个声明中设置所有外边距属性。
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。

边界也是可分为上、右、下、左。如下代码：

```
div{margin:20px 10px 15px 30px;}
```

padding 和 margin 的区别，padding 在边框里，margin 在边框外。

Margin 在布局中的一个巧用是水平居中。

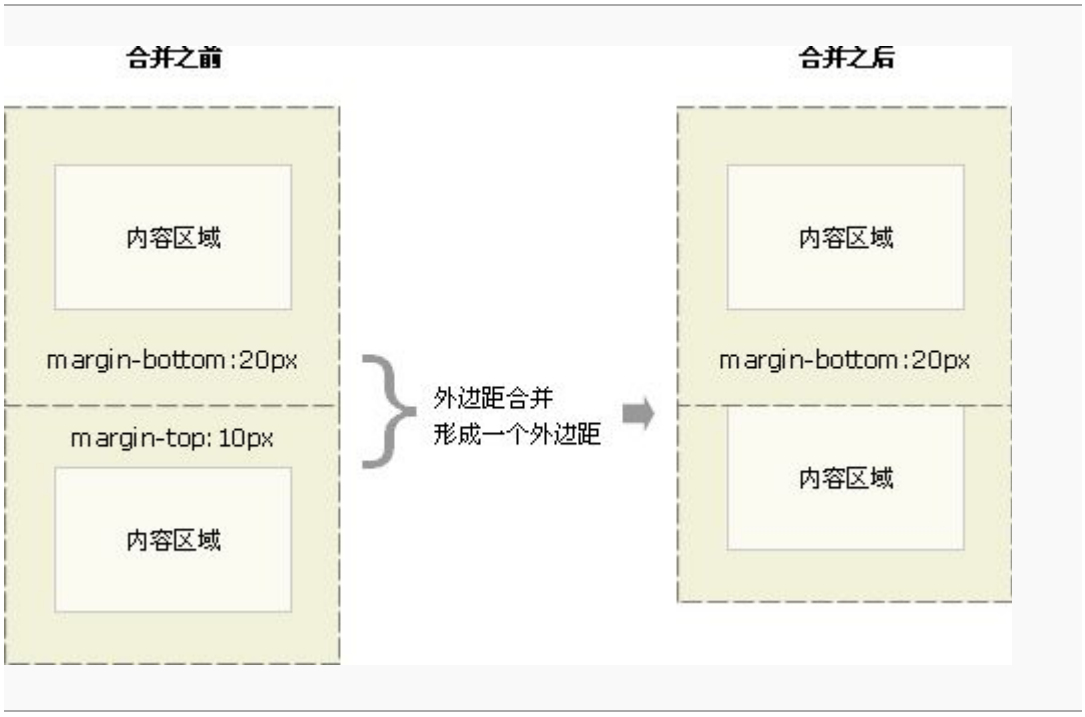
设置 margin:0 auto，可以使元素实现水平居中效果。

CSS 外边距合并

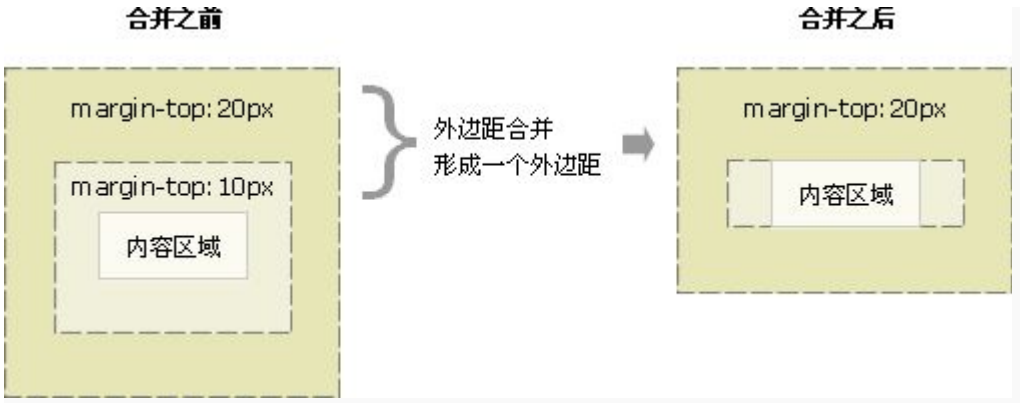
外边距合并（叠加）是一个相当简单的概念。但是，在实践中对网页进行布局时，它会造成许多混淆。

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。请看下图：

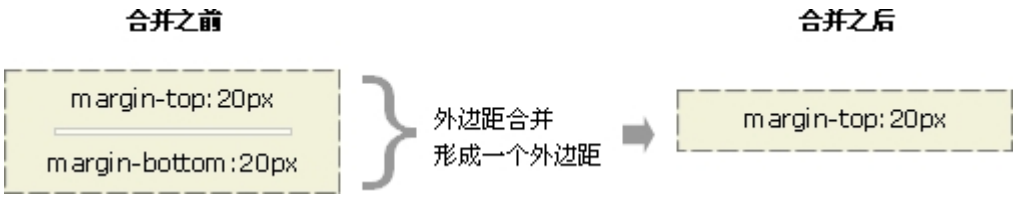


当一个元素包含在另一个元素中时（假设没有内边距或边框把外边距分隔开），它们的上和/或下外边距也会发生合并。请看下图：



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。

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。**

参考资料：

外边距 margin 合并（塌陷），请查看以下资料：

https://developer.mozilla.org/zh-CN/docs/Web/CSS/margin_collapsing

http://www.w3help.org/zh-cn/kb/006/

Border-radius---圆角边框

在 CSS3 中，针对边框，增加了丰富的修饰效果，使得网页更加美观舒服。下面列出了常用的 CSS3 边框属性：

CSS3 边框属性

属性	说明
border-radius	圆角效果
border-colors	多色边框
border-image	边框背景
box-shadow	边框阴影

border-radius 属性

语法：border-radius:长度值;

说明：长度值可以是 px、百分比、em 等。

Border-radius 属性设置长度值顺序：“左上角、右上角、右下角和左下角”。

border-radius 画实心半圆和实心圆

1、实心半圆

实心半圆分为：实心上半圆、实心下半圆、实心左半圆、实心右半圆。我们只要掌握制作一个方向的实心半圆的方法，其他方向的实心半圆就可以轻松实现，因为原理都一样。

假如我们要制作**实现上半圆**，实现方法：把高度(height)设为宽度（width）的一半，并且只设置左上角和右上角的圆角半径与元素的高度一致，而右下角和左下角的圆角半径设置为 0。

```
    width:200px;
        height:100px;
        border:1px solid red;
        border-radius:100px 100px 0 0;
```

2、实心圆

在 CSS3 中，使用 border-radius 属性实现实心圆方法 把宽度(width)与高度(height)值设置为一致（也就是正方形），并且四个圆角值都设置为它们值的一半。

```
    width:100px;
        height:100px;
        border:1px solid red;
        border-radius:50px;
```

border-radius 属性派生子属性

border-radius 属性可以分开，分别为四个角设置相应的圆角值，分别是：
（1）border-top-right-radius：右上角；
（2）border-bottom-right-radius：右下角；
（3）border-bottom-left-radius：左下角；
（4）border-top-left-radius：左上角；

边框阴影 box-shadow

在 CSS3 中，我们可以使用 box-shadow 属性轻松地为元素添加阴影效果。
语法：box-shadow：x-shadow y-shadow blur spread color inset;
说明：
1. x-shadow：设置水平阴影的位置（X 轴），可以使用负值；
2. y-shadow：设置垂直阴影的位置（y 轴），可以使用负值；

3. blur：设置阴影模糊半径；
4. spread：阴影扩展半径，设置阴影的尺寸；这个参数可选，缺省时值为 0。
5. color：设置阴影的颜色；
6. inset：这个参数默认不设置。默认情况下为外阴影，inset 表示内阴影。（这个值可以放在开头位置。）
7.

外阴影 outset 与内阴影 inset

box-shadow 属性最后一个参数用于设置阴影是否是内阴影，还是外阴影。
取值有 2 种：
（1）outset：默认值，外阴影；
（2）inset：内阴影；
技巧：当水平阴影位置 x-shadow 和垂直阴影位置 y-shadow 都为 0 时，阴影都向外发散或者都向内发散。

CSS 轮廓

轮廓（outline）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。
CSS outline 属性规定元素轮廓的样式、颜色和宽度。

属性	描述	CSS
outline	在一个声明中设置所有的轮廓属性。	2
outline-color	设置轮廓的颜色。	2
outline-style	设置轮廓的样式。	2
outline-width	设置轮廓的宽度。	2

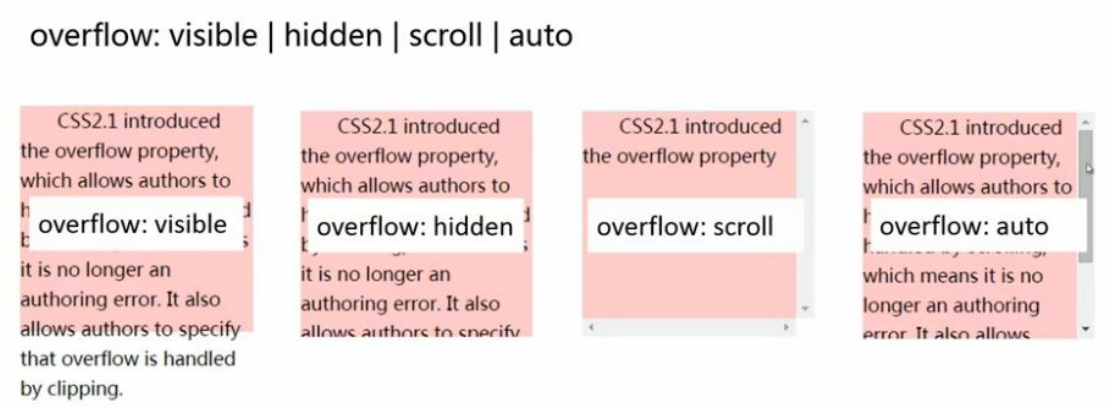
Overflow 属性

overflow 属性规定当内容溢出元素框时发生的事情.

值	描述
visible	默认值。内容不会被修剪，会呈现在元素框之外。
hidden	内容会被修剪，并且其余内容是不可见的。
scroll	内容会被修剪，但是浏览器会显示滚动条以便查看其余的内容。
auto	如果内容被修剪，则浏览器会显示滚动条以便查看其余的内容。

inherit	规定应该从父元素继承 overflow 属性的值。

示例：



Box-sizing 属性

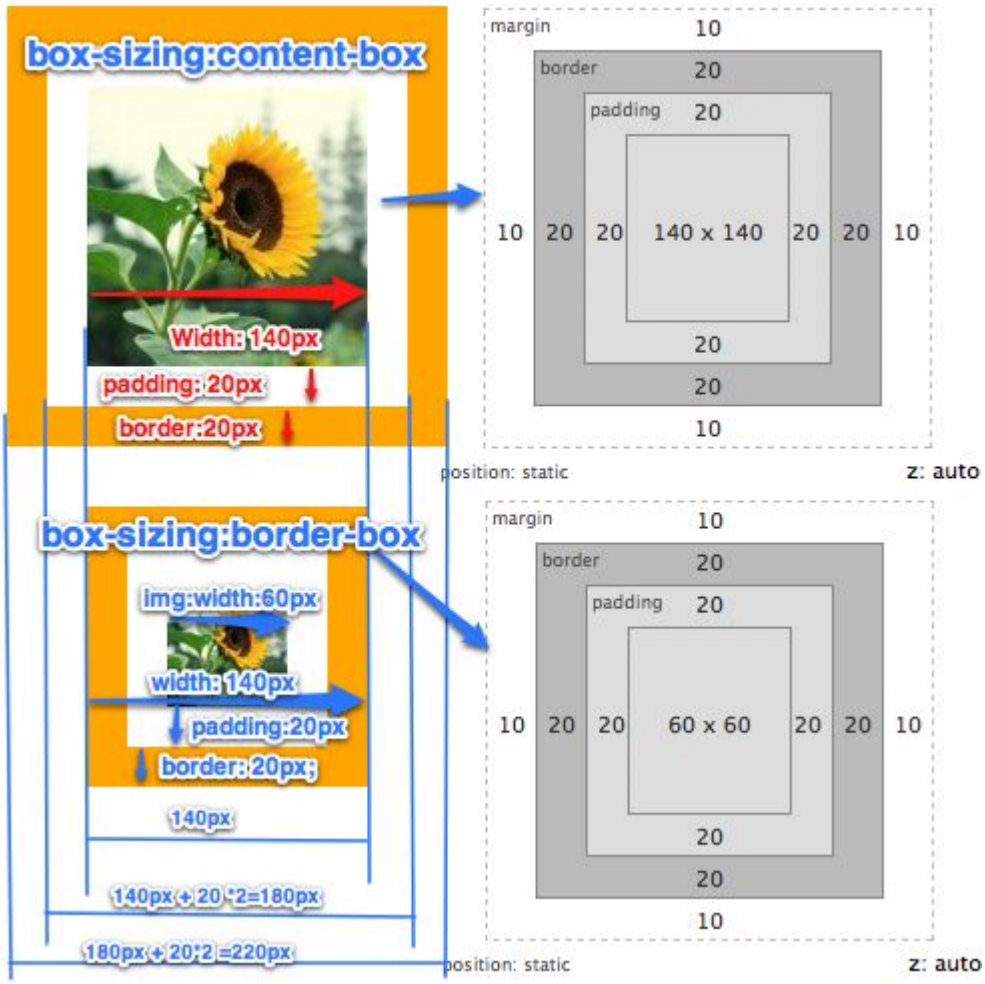
box-sizing 是 CSS3 的 box 属性之一。

语法：

```
box-sizing: content-box|border-box|inherit;
```

属性值	属性值说明
content-box	默认值，其让元素维持 W3C 的标准盒模型，也就是说元素的宽度和高度（width/height）等于元素边框宽度（border）加上元素内距（padding）加上元素内容宽度或高度（content width/ height），也就是 element width/height = border + padding + content width / height
border-box	重新定义 CSS2.1 中盒模型组成的模式，让元素维持 IE 传统的盒模型（IE6 以下版本和 IE6-7 怪异模式），也就是说元素的宽度或高度等于元素内容的宽度或高度。从上面盒模型介绍可知 ,这里的内容宽度或高度包含了元素的border、padding、内容的宽度或高度（此处的内容宽度或高度 = 盒子的宽度或高度—边框—内距）。
inherit	使元素继承父元素的盒模型模式

其中最为关键的是 box-sizing 中 content-box 和 border-box 两者的区别，他们之间的区别可以通过下图来展示，其对盒模型的不同解析：



参考资料：

1.box-sizing 理解： http://www.w3cplus.com/content/css3-box-sizing

2. CSS 属性的浏览器兼容性：http://caniuse.com/

CSS（五）

背景

background 属性

值	描述	CSS
background-color	规定要使用的背景颜色。	1
background-position	规定背景图像的位置。	1
background-size	规定背景图片的尺寸。	3
background-repeat	规定如何重复背景图像。	1
background-origin	规定背景图片的定位区域。	3
background-clip	规定背景的绘制区域。	3
background-attachment	规定背景图像是否固定或者随着页面的其余部分滚动。	1
background-image	规定要使用的背景图像。	1
inherit	规定应该从父元素继承 background 属性的设置。	1

1、背景颜色-background-color

在 CSS 中，使用 background-color 属性来控制元素的背景颜色。

默认值：transparent ，即背景是透明的。

2、背景图像-background-image

CSS 背景图像属性

属性	说明
background-image	定义背景图像的路径，这样图片才能显示

CSS 背景图像属性

属性	说明
background-repeat	定义背景图像显示方式，例如纵向平铺、横向平铺
background-position	定义背景图像在元素哪个位置
background-attachment	定义背景图像是否随内容而滚动

```
background-image:url("../App_images/lesson/run_cj/one piece.jpg");
```

3.background-repeat

语法：

background-repeat：<repeat-style> [, <repeat-style>]*

<repeat-style> = repeat-x | repeat-y | [repeat | no-repeat | space | round]{1,2}

默认值：repeat

属性值	说明
no-repeat	表示不平铺
repeat	默认值，表示在水平方向（x 轴）和垂直方向（y 轴）同时平铺
repeat-x	表示在水平方向（x 轴）平铺
repeat-y	表示在垂直方向（y 轴）平铺
space	背景图像以相同的间距平铺且填充满整个容器或某个方向。
round	背景图像自动缩放直到适应且填充满整个容器。

设置或检索对象的背景图像如何铺排填充。必须先指定 <' background-image '> 属性。

- 允许提供 2 个参数，如果提供全部 2 个参数，第 1 个用于横向，第二个用于纵向。
- 如果只提供 1 个参数，则用于横向和纵向。特殊值 repeat-x 和 repeat-y 除外，因为 repeat-x 相当于 repeat no-repeat，repeat-y 相当于 no-repeat repeat，即其实 repeat-x 和 repeat-y 等价于提供了 2 个参数值 .

4. background-position 属性

语法：

background-position：<position> [, <position>]*

<position> = [left | center | right | top | bottom | <percentage> | <length>] | [left | center | right | <percentage> | <length>] [top | center | bottom | <percentage> | <length>] | [center | [left | right] [<percentage> | <length>]?] && [center | [top | bottom] [<percentage> | <length>]?]

默认值：0% 0%，效果等同于 left top

(1). background-position 取值为“像素值”

设置值	说明
x（数值）	设置网页的横向位置，单位为 px
y（数值）	设置网页的纵向位置，单位为 px

```
background-position:80px 40px;
```

(2). background-position 取值为“关键字”

属性值	说明
top left	左上
top center	靠上居中
top right	右上
left center	靠左居中
center center	正中
right center	靠右居中
bottom left	左下
bottom center	靠下居中
bottom right	右下

5. background-attachment 属性

在 CSS 中，使用背景附件属性 background-attachment 可以设置背景图像是随对象滚动还是固定不动。

语法：

background-attachment：<attachment> [, <attachment>]*

<attachment> = fixed | scroll | local

默认值：scroll

- fixed： 背景图像相对于窗体固定。
- scroll： 背景图像相对于元素固定，也就是说当元素内容滚动时背景图像不会跟着滚动，因为背景图像总是要跟着元素本身。但会随元素的祖先元素或窗体一起滚动。

- local： 背景图像相对于元素内容固定，也就是说当元素随元素滚动时背景图像也会跟着滚动，因为背景图像总是要跟着内容。（CSS3）

6. background-size 属性

在 CSS3 中，我们可以使用 background-size 属性来设置背景图片的大小，这使得我们可以在不同的环境中重复使用背景图片。

语法： background-size: auto | <长度值> | <百分比> | cover | contain

background-size 关键字取值

关键字	说明
cover	即“覆盖”，将背景图片以等比缩放来填充整个容器元素
contain	即“容纳”，将背景图片等比缩放至某一边紧贴容器边缘为止
auto	默认值，不改变背景图片的原始高度和宽度
<长度值>	成对出现如 200px 50px，将背景图片宽高依次设置为前面两个值，当设置一个值时，将其作为图片宽度值来等比缩放
<百分比>	0%~100%之间的任何值，将背景图片宽高依次设置为所在元素宽高乘以前面百分比得出的数值，当设置一个值时同上

对于背景图片，不是可以使用 width 和 height 属性来设置吗？为什么还要增加一个 background-size 属性呢？

记住，背景图片不同于 img 标签引用的图片，对于 img 标签引用的图片，我们可以使用 width 和 height 属性来设置，但是这两个属性不能用于设置背景图片的大小。因此，在 CSS3 中，引入了 background-size 属性来设置背景图片的大小。这里大家要清楚一点，背景图片的大小跟一般图片的大小设置有本质的区别。

7. background-origin 属性

在 CSS3 中，我们可以使用 background-origin 属性来设置元素背景图片平铺的最开始位置。

语法： background-origin ： border-box | padding-box | content-box;

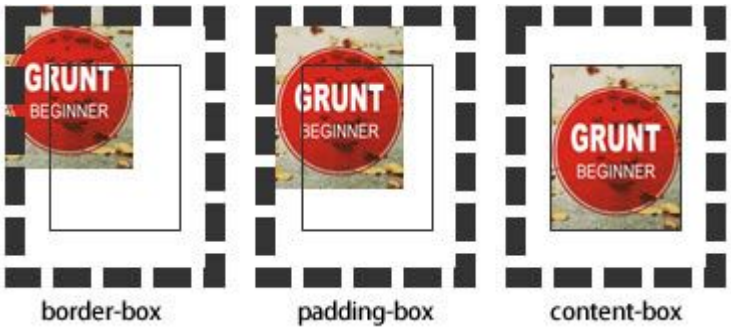
background-origin 属性取值

属性值	说明
border-box	表示背景图片是从边框开始平铺
padding-box	表示背景图片是从内边距开始平铺（默认值）
content-box	表示背景图片是从内容区域开始平铺

总结： background-origin 属性往往都是配合 background-position 属性来使用，其中 **background-origin 属性规定 background-position 属性相对于什么位置来定位。**浏览器默

认采用“background-position:top left;”。因此不管 background-origin 属性值如何变化，背景图片都是从“左上”开始平铺。

效果如下：



需要注意的是，如果背景不是 no-repeat，这个属性无效，它会从边框开始显示。

8. background-clip 属性

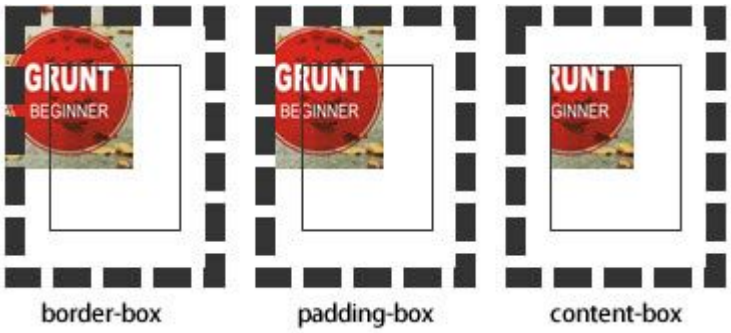
在 CSS3 中，使用 background-clip 属性来将背景图片根据实际需要进行剪切。

语法： background-clip ： border-box | padding-box | content-box | no-clip

属性值	说明
border-box	默认值，表示从边框 border 开始剪切
padding-box	表示从内边距 padding 开始剪切
content-box	表示从内容区域 content 开始剪切
No-clip	不裁切，和**参数 border-box** 显示同样的效果

backgroud-clip 默认值为 border-box。

效果如下图所示：



background-clip 属性指定了背景在哪些区域可以显示，但与背景开始绘制的位置（即 background-origin 属性）无关。背景绘制的位置可以出现在不显示背景的区域。这就相当于背景图片被不显示背景的区域裁剪了一部分一样。

9. Background

多重背景(multiple backgrounds)，也就是 CSS2 里 background 的属性外加 origin、clip 和 size 组成的新 background 的多次叠加，缩写时为用逗号隔开的每组值；

用分解写法时，如果有多个背景图片，而其他属性只有一个（例如 background-repeat 只有一个），表明所有背景图片应用该属性值。

语法缩写如下：

```
background ： [background-color] | [background-image] | [background-position][/ background-size] | [background-repeat] | [background-attachment] | [background-clip] | [background-origin],...
```

可以把上面的缩写拆解成以下形式：

background-image:url1,url2,...,urlN;

```
background-repeat : repeat1,repeat2,...,repeatN;
backround-position : position1,position2,...,positionN;
background-size : size1,size2,...,sizeN;
background-attachment : attachment1,attachment2,...,attachmentN;
background-clip : clip1,clip2,...,clipN;
background-origin : origin1,origin2,...,originN;
background-color : color;
```

注意：

1. 用逗号隔开每组 background 的缩写值；
2. 如果有 size 值，需要紧跟 position 并且用 "/" 隔开；
3. 如果有多个背景图片，而其他属性只有一个（例如 background-repeat 只有一个），表明所有背景图片应用该属性值。
4. background-color 只能设置一个。

CSS3 渐变

CSS3 渐变共有 2 种：（1）线性渐变（linear-gradient）；（2）径向渐变（radial-gradient）。

1、线性渐变-linear-gradient

线性渐变，指的就是指在一条直线上进行渐变，在网页中大多数渐变效果都是线性渐变。

语法：

```
background: linear-gradient(direction, color-stop1, color-stop2, ...);
```

说明：线性渐变的方向取值有 2 种，一种是使用角度（deg），另外一种是使用关键字：

线性渐变的方向取值

属性值	对应角度	说明
to top	0deg	从下到上
to right	90deg	从左到右
to bottom	180deg	从上到下（默认值）
to left	270deg	从右到左
to top left		右上角到左上角（斜对角）
to top right		左下角到右上角（斜对角）

第 2 个参数和第 3 个参数，表示开始颜色和结束颜色，取值可以为关键字、十六进制颜色值、RGBA 颜色等。你可以使用在线调色板来获取颜色值。线性渐变可以有多个颜色值。

也可以定义多种颜色的线性渐变效果，如：

```
background-image:linear-gradient(to left, red, orange,yellow,green,blue,indigo,violet);
```

效果图：

2、径向渐变-radial-gradient

径向渐变，是一种从起点到终点颜色从内到外进行圆形渐变（从中间向外拉，像圆一样）。CSS3 径向渐变是圆形或椭圆形渐变，颜色不再沿着一条直线渐变，而是从一个起点向所有方向渐变。

语法：background:radial-gradient(position ,shape size,start-color,…,stop-color)

说明：

1. position：定义圆心位置,如果提供 2 个参数，第一个表示横坐标，第二个表示纵坐标；如果只提供一个，第二值默认为 50%，即 center.
2. shape size：由 2 个参数组成，shape 定义形状（圆形或椭圆），size 定义大小；
3. start-color：定义开始颜色值；
4. stop-color：定义结束颜色值；
5. **position、shape size 都是可选参数**，如果省略，则表示该项参数采用默认值(center,ellipse)。start-color 和 stop-color 为必选参数，并且径向渐变可以有多个颜色值。

1、定义圆心位置 position

position 用于定义径向渐变的圆心位置，属性值跟 background-position 属性值相似，也有 2 种情况：（1）长度值，如 px、em 或百分比等；（2）关键字。

圆心位置取值（关键字）

属性值	说明
center	中部（默认值）
top	顶部
right	右部
bottom	底部
left	左部
top left	左上
top center	靠上居中
top right	右上
left center	靠左居中
center center	正中
right center	靠右居中
bottom left	左下
bottom center	靠下居中
bottom right	右下

2、定义形状 shape 和定义大小 size

（1）定义形状 shape

属性值	说明
circle	定义径向渐变为“圆形”
ellipse	定义径向渐变为“椭圆形”（默认值）

（2）定义大小 size

size 主要用于定义径向渐变的结束形状大小。

属性值	说明
closest-side	指定径向渐变的半径长度为从圆心到离圆心最近的边
closest-corner	指定径向渐变的半径长度为从圆心到离圆心最近的角
farthest-side	指定径向渐变的半径长度为从圆心到离圆心最远的边
farthest-corner	指定径向渐变的半径长度为从圆心到离圆心最远的角

3、开始颜色 start-color 和结束颜色 stop-color

参数 start-color 用于定义开始颜色，参数 stop-color 用于定义结束颜色。颜色可以为关键词、十六进制颜色值、RGBA 颜色值等。

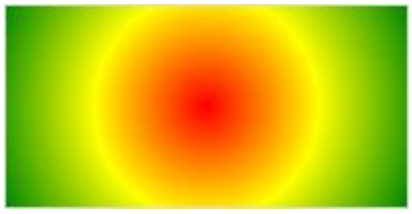
径向渐变也接受一个颜色值列表，用于同时定义多种颜色的径向渐变。
默认情况下，径向渐变颜色节点是均匀分布的，不过我们也可以为每一种颜色添加百分比，来使得各个颜色节点不均匀分布。如下例：

```
#div1{background:-webkit-radial-gradient(red,green,blue);margin-bottom:10px;}
#div2{background:-webkit-radial-gradient(red 5%,green 30%,blue 60%);}
```

案例：

1. 用默认的渐变方向绘制一个最简单的径向渐变

示例代码：



（图一）

- radial-gradient(circle, #f00, #ff0, #080);
- radial-gradient(circle at center, #f00, #ff0, #080);
- radial-gradient(circle at 50%, #f00, #ff0, #080);
- radial-gradient(circle farthest-corner, #f00, #ff0, #080);

以上几句代码都可以实现如（图一）的渐变效果

2. <shape> 和 <size> 使用注意：

- 错误代码：radial-gradient(circle 50px 50px, #f00, #ff0, #080);

因为 circle 是正圆，一个值就能表示其直径长度，所以此时 <size> 只能是一个值。

- 错误代码：radial-gradient(circle 50%, #f00, #ff0, #080);

circle 不接受 <size> 的值是 <percentage>。

3. 不通过 <shape> 来表示圆和椭圆的方法：

以下 2 行代码都可以表示一个圆：

```
radial-gradient(100px, #f00, #ff0, #080); /* 1 */
```

```
radial-gradient(100px 100px, #f00, #ff0, #080); /* 2 */
radial-gradient(50px 100px, #f00, #ff0, #080); /* 3 */
```

- 代码 1：只给出 100px，所以被当成是正圆的半径，于是就能确定一个直径为 100px 的圆；
- 代码 2：给出了 2 个值，按理应该是要画一个椭圆的，但 2 个值相等，所以这个椭圆其实此时是个正圆形态。需要注意的是，代码 2 如果加上 circle，那将是错误语法，因为这是 2 个值只有椭圆才接受；
- 代码 3：表示了一个水平半径为 50px，垂直半径为 100px 的椭圆

3、渐变重复： repeating-*-gradient

3.1repeating-linear-gradient()

用重复的线性渐变创建图像。

repeating-linear-gradient()的语法与 linear-gradient()相同。

示例代码：

- repeating-linear-gradient(#f00, #ff0 10%, #f00 15%);
- repeating-linear-gradient(to bottom, #f00, #ff0 10%, #f00 15%);
- repeating-linear-gradient(180deg, #f00, #ff0 10%, #f00 15%);
- repeating-linear-gradient(to top, #f00, #ff0 10%, #f00 15%);

以上几句代码都可以实现如上图的渐变效果

其实可以使用 linear-gradient() 做出 repeating-linear-gradient() 的效果

- 暴力实现上述（图一）的效果：

```
repeating-linear-linear-gradient(#f00, #ff0 10%, #f00 15%, #ff0 25%, #f00 30%, #ff0 40%, #f00 45%, #ff0 55%, #f00 60%, #ff0 70%, #f00 75%, #ff0 85%, #f00 90%, #ff0);
```

- 利用 background-size 实现上述（图一）的效果：

```
background: linear-gradient(#f00, #ff0 67%, #f00);
background-size: 100% 15%;
```

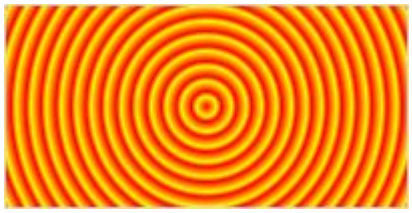
使用 background-size 约束渐变图像的大小，然后通过 background-repeat 来纵向平铺

3.2repeating-radial-gradient()

用重复的径向渐变创建图像。

repeating-radial-gradient()的语法与 radial-gradient()相同。

示例代码：



```
repeating-radial-gradient(circle closest-side, #f00, #ff0 10%, #f00 15%);
```

CSS（六）

布局

内容包括：

布局简介、display(水平居中、居中导航)、position(轮播头图、固定顶栏、遮罩、三行自适应布局)、float(两列布局)、flex(三行两列自适应)

布局：将元素以正确的大小摆放在正确的位置上。

布局属性：

Display:设置元素的显示方式

语法：

display: none | inline | block | list-item | inline-block | table | inline-table | table-caption | table-cell | table-row | table-row-group | table-column | table-column-group | table-footer-group | table-header-group | run-in | box | inline-box | flexbox | inline-flexbox | flex | inline-flex

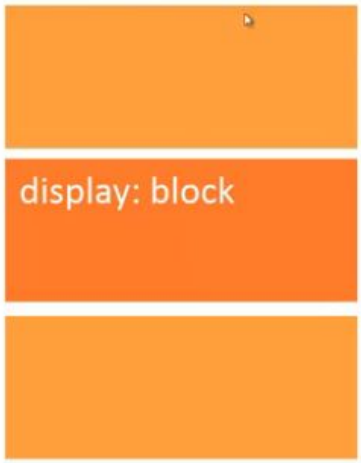
默认值：inline

取值：

- none： 隐藏对象。与 visibility 属性的 hidden 值不同，其不为被隐藏的对象保留其物理空间
- inline： 指定对象为内联元素。
- block： 指定对象为块元素。
- list-item： 指定对象为列表项目。
- inline-block： 指定对象为内联块元素。（CSS2）
- table： 指定对象作为块元素级的表格。类同于 html 标签<table>（CSS2）
- inline-table： 指定对象作为内联元素级的表格。类同于 html 标签<table>（CSS2）
- table-caption： 指定对象作为表格标题。类同于 html 标签<caption>（CSS2）
- table-cell： 指定对象作为表格单元格。类同于 html 标签<td>（CSS2）
- table-row： 指定对象作为表格行。类同于 html 标签<tr>（CSS2）
- table-row-group： 指定对象作为表格行组。类同于 html 标签<tbody>（CSS2）
- table-column： 指定对象作为表格列。类同于 html 标签<col>（CSS2）
- table-column-group： 指定对象作为表格列组显示。类同于 html 标签<colgroup>（CSS2）
- table-header-group： 指定对象作为表格标题组。类同于 html 标签<thead>（CSS2）
- table-footer-group： 指定对象作为表格脚注组。类同于 html 标签<tfoot>（CSS2）
- run-in： 根据上下文决定对象是内联对象还是块级对象。（CSS3）
- box： 将对象作为弹性伸缩盒显示。（伸缩盒最老版本）（CSS3）
- inline-box： 将对象作为内联块级弹性伸缩盒显示。（伸缩盒最老版本）（CSS3）
- flexbox： 将对象作为弹性伸缩盒显示。（伸缩盒过渡版本）（CSS3）
- inline-flexbox： 将对象作为内联块级弹性伸缩盒显示。（伸缩盒过渡版本）（CSS3）
- flex： 将对象作为弹性伸缩盒显示。（伸缩盒最新版本）（CSS3）
- inline-flex： 将对象作为内联块级弹性伸缩盒显示。（伸缩盒最新版本）（CSS3）

Display: Block（块级元素）

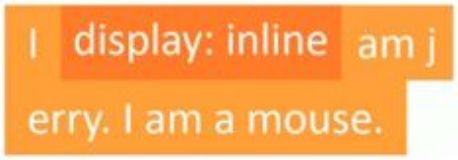
- 默认宽度为父元素宽度
- 可设置宽高
- 换行显示



默认的 display:block 元素有：div,h1-h6,p,ul,form,……

Display: inline（行内元素）

- 默认宽度为内容宽度
- 不可设置宽高
- 同行显示



默认 display:inline 元素有：span,a,label,cite,em,……

Display:block 与 display:inline 比较

display	默认宽度	可设置宽高	起始位置
block	父元素宽度	是	换行
inline	内容宽度	否	同行

Display: inline-block（行内元素）

默认 display:inline-block 元素有：img,input,textarea,select,button,……

Display: none

设置元素不显示。

display:none 或 visibility:hidden

隐藏一个元素可以通过把 display 属性设置为"none"，或把 visibility 属性设置为"hidden"。display 属性设置一个元素应如何显示，visibility 属性指定一个元素应可见还是隐藏。这两种方法会产生不同的结果。

- visibility:hidden 可以隐藏某个元素，但隐藏的元素仍需占用与未隐藏之前一样的空间。也就是说，该元素虽然被隐藏了，但仍然会影响布局.
- display:none 可以隐藏某个元素，且隐藏的元素不会占用任何空间。也就是说，该元素不但被隐藏了，而且该元素原本占用的空间也会从页面布局中消失。

与 Display 相关布局模式

1. 块级元素水平居中

```html
<div>
  <div class="content">content area</div>
</div>
```

HTML

```css
.content{margin: auto;width: 978px;}
```

CSS

2. 居中导航

```html
<ul>
  <li><a href="#">推荐</a></li>
  <li><a href="#">歌单</a></li>
  <li><a href="#">大牌DJ</a></li>
  <li><a href="#">歌手</a></li>
  <li><a href="#">新碟上架</a></li>
</ul>
```

HTML

```css
ul{text-align: center;height: 30px;line-height: 30px;}
li, a{display: inline-block;width: 80px;height: 100%;}
li{margin: 0 10px;}
```

CSS

参考知识：

1. 元素分类

在 CSS 中，html 中的标签元素大体被分为三种不同的类型：块状元素、内联元素(又叫行内元素)和内联块状元素。

常用的块状元素有：

、<p>、<h1>...<h6>、<ol>、<ul>、<dl>、<table>、<address>、<blockquote> 、<form>

常用的内联元素有：

<a>、、
、<i>、、、<label>、<q>、<var>、<cite>、<code>

常用的内联块状元素有：

<img>、<input>

元素分类--块级元素

设置 display:block 就是将元素显示为块级元素。

如下代码就是将内联元素 a 转换为块状元素，从而使 a 元素具有块状元素特点。

```
a{display:block;}
```

块级元素特点：

- 每个块级元素都从新的一行开始，并且其后的元素也另起一行。（真霸道，一个块级元素独占一行）
- 元素的高度、宽度、行高以及顶和底边距都可设置。
- 元素宽度在不设置的情况下，是它本身父容器的 100%（和父元素的宽度一致），除非设定一个宽度。

元素分类--内联元素

在 html 中，、<a>、<label>、 和就是典型的内联元素（行内元素）（inline）元素。当然块状元素也可以通过代码 display:inline 将元素设置为内联元素。

如下代码就是将块状元素 div 转换为内联元素，从而使 div 元素具有内联元素特点。

```
div{display:inline; }
```

内联元素特点：

- 和其他元素都在一行上；

- 元素的高度、宽度及顶部和底部边距不可设置；
- 元素的宽度就是它包含的文字或图片的宽度，不可改变。

解决内联元素（行内元素）间隙 bug 问题

行内元素之间会产生间隙 bug 问题的场景：

当行内元素之间有“回车”、“tab”、“空格”时就会出现间隙。

如下代码：

```
<div>
    <a>1</a>
    <a>2</a>
    <span>33333</span>
    <span>44444</span>
    <em>555555</em>
</div>
```

解决方法：

1、写在一行，之间不要有空格之类的符号。

```
<div>
<a>1</a><a>2</a><span>33333</span><span>44444</span><em>555555</em>
</div>
```

2、使用 font-size:0，设置内联元素的父元素字体大小为 0，然后设置内联元素字体大小。

```
div{font-size:0;}
a,span,em{font-size:16px;}/*div 为 a、span、em 元素的父元素*/
```

3、标签分开写，例如：

```
<div>
    <a>1</a
> <a>2</a
> <span>33333</span
> <span>44444</span
> <em>555555</em>
</div>
```

元素分类--内联块状元素

内联块状元素（inline-block**）**就是同时具备内联元素、块状元素的特点，**代码 display:inline-block 就是将元素设置为内联块状元素。**(css2.1 新增)，<img>、<input>标签就是这种内联块状标签。

inline-block 元素特点：

1、和其他元素都在一行上；

2、元素的高度、宽度、行高以及顶和底边距都可设置。

2. 如何实现浏览器兼容版的 inline-block 显示

IE6,7 支持 inline 元素转换成 inline-block，但不支持 block 元素转换成 inline-block,所以非 inline 元素在 IE6,7 下要转换成 inline-block,需先转换成 inline，然后触发 hasLayout，以此来获得和 inline-block 类似的效果；你可以这样：
全兼容的 inline-block：

```
div {
        display: inline-block;
        *display: inline;
        *zoom: 1;
}
```

对应的脚本特性为 display。

定位（position）：

CSS 定位（Positioning）属性允许你对元素进行定位。

定位的基本思想: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。

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。

CSS 定位属性

- **Position-设置定位方式**
- **Top,right,bottom,left,z-index-设置元素边缘距离参照物边缘的距离**

CSS 定位属性允许你对元素进行定位。

属性	描述

position	把元素放置到一个静态的、相对的、绝对的、或固定的位置中。
top	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。
right	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。
bottom	定义了定位元素下外边距边界与其包含块下边界之间的偏移。
left	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。
overflow	设置当元素的内容溢出其区域时发生的事情。
clip	设置元素的形状。元素被剪入这个形状之中，然后显示出来。
vertical-align	设置元素的垂直对齐方式。
z-index	设置元素的堆叠顺序。

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。

- `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。

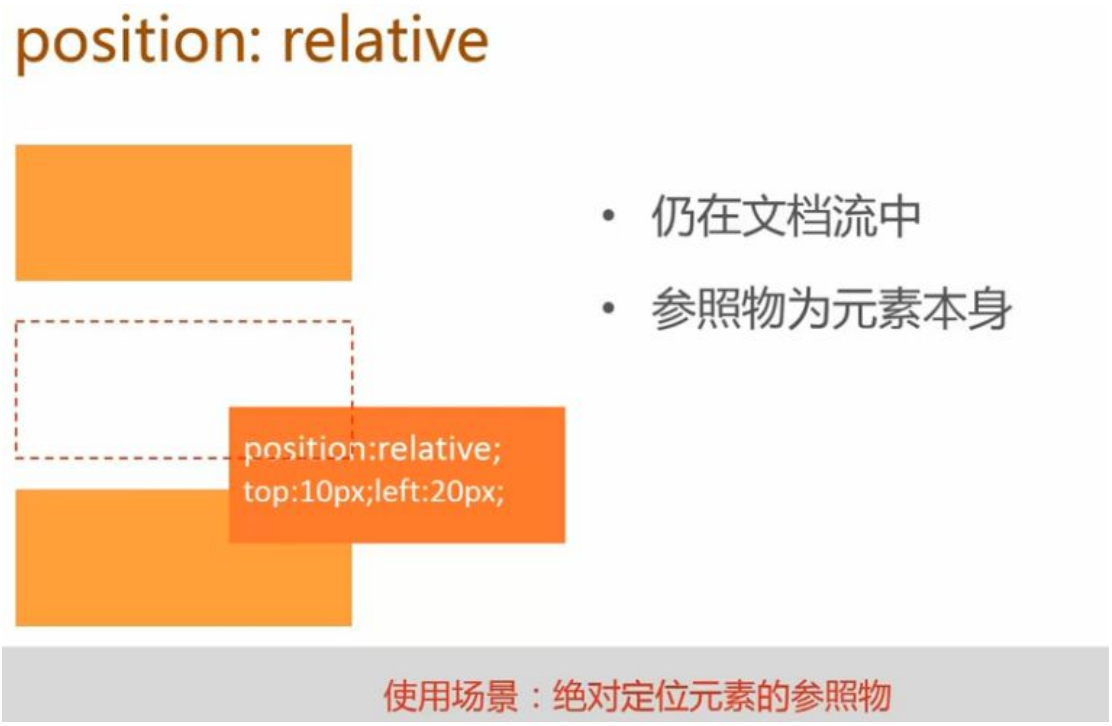
- `absolute`

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。

Position:relative



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。

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。

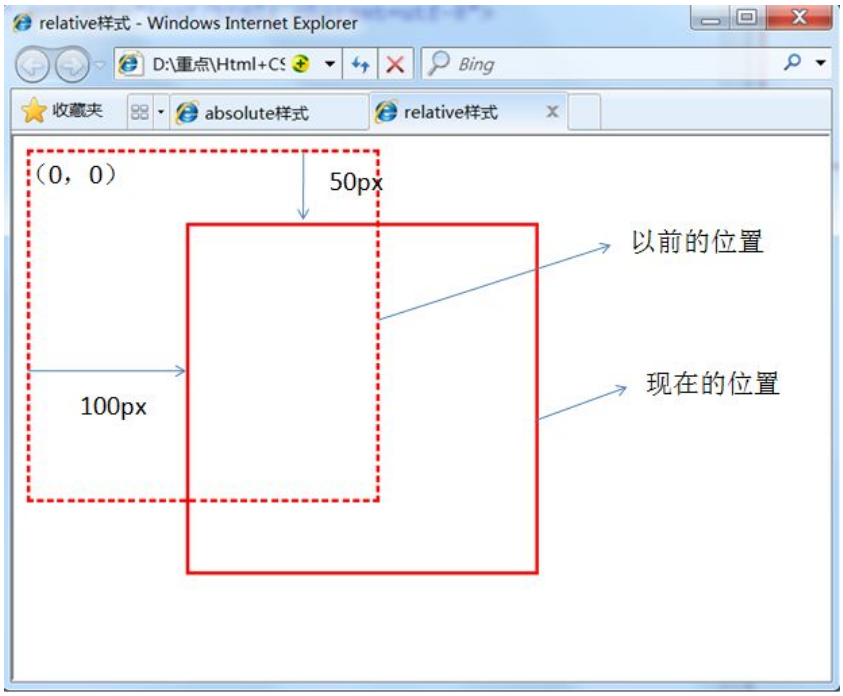
注意，在使用相对定位时，无论是否进行移动，元素仍然占据原来的空间。因此，移动元素会导致它覆盖其它框.

相对定位完成的过程是首先按 static(float)方式生成一个元素(并且元素像层一样浮动了起来)，然后相对于以前的位置移动，移动的方向和幅度由 left、right、top、bottom 属性确定，偏移前的位置保留不动。

如下代码实现相对于以前位置向下移动 50px，向右移动 100px;

```
#div1{
    width:200px;
    height:200px;
    border:2px red solid;
    position:relative;
    left:100px;
    top:50px;
}
<div id="div1"></div>
```

效果图：



Position:absolute

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。

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**。

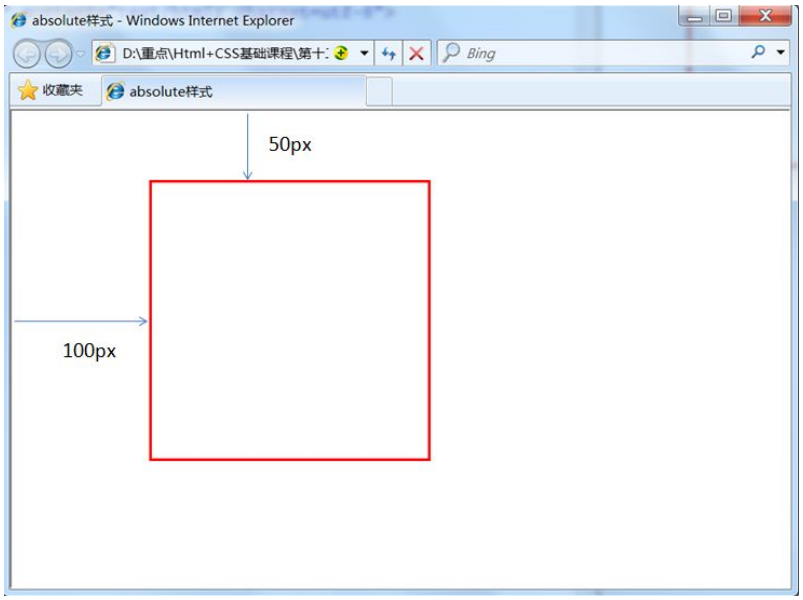
如果想为元素设置层模型中的绝对定位，需要设置 **position:absolute**(表示绝对定位)，这条语句的作用将元素从文档流中拖出来，然后使用 left、right、top、bottom 属性相对于其**最接近的一个具有定位属性的父包含块**进行绝对定位。如果不存在这样的包含块，则相对于 body 元素，即相对于**浏览器窗口**。

对于定位的主要问题是要记住每种定位的意义。**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。**

如下面代码可以实现 div 元素相对于浏览器窗口向右移动 100px，向下移动 50px。

```
div{
    width:200px;
    height:200px;
    border:2px red solid;
    position:absolute;
    left:100px;
    top:50px;
}
<div id="div1"></div>
```

效果如下：



Position:absolute-应用：轮播头图

Position:fixed

fixed：表示固定定位，与 absolute 定位类型类似，但它的相对移动的坐标是视图（**屏幕内的网页窗口**）本身。由于视图本身是固定的，它不会随浏览器窗口的滚动条滚动而变化，除非你在屏幕中移动浏览器窗口的屏幕位置，或改变浏览器窗口的显示大小，因此固定定位的元素会始终位于浏览器窗口内视图的某个位置，不会受文档流动影响，这与 background-attachment:fixed;属性功能相同。

以下代码可以实现相对于**浏览器视图**向右移动 100px，向下移动 50px。并且拖动滚动条时位置固定不变。

```
#div1{
    width:200px;
    height:200px;
    border:2px red solid;
    position:fixed;
    left:100px;
    top:50px;
}
<p>文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本文本。</p>
....
```

Position:fixe-应用：

1. 固定顶栏布局

```html
<body>
  <div class="top">top bar</div>
  <div class="main">main content area</div>
</body>
```

HTML

```css
body{padding-top: 50px;}
.top{position: fixed;top: 0;width: 100%;height: 50px;}
```

CSS

2.遮罩布局

```html
<div class="mask" ></div>
```

HTML

```css
.mask{position: fixed;top: 0;left: 0;z-index: 999;width: 100%;height: 100%;}
```

CSS

3.三行自适应布局

```html
<div class="head">head</div>
<div class="body">body</div>
<div class="foot">foot</div>
```

HTML

```css
.head{position: absolute;top: 0;left: 0;width: 100%;height: 100px;}
.body{position: absolute;top: 100px;left: 0;bottom: 100px;right: 0;}
.foot{position: absolute;bottom: 0;left: 0;width: 100%;height: 100px;}
```

CSS

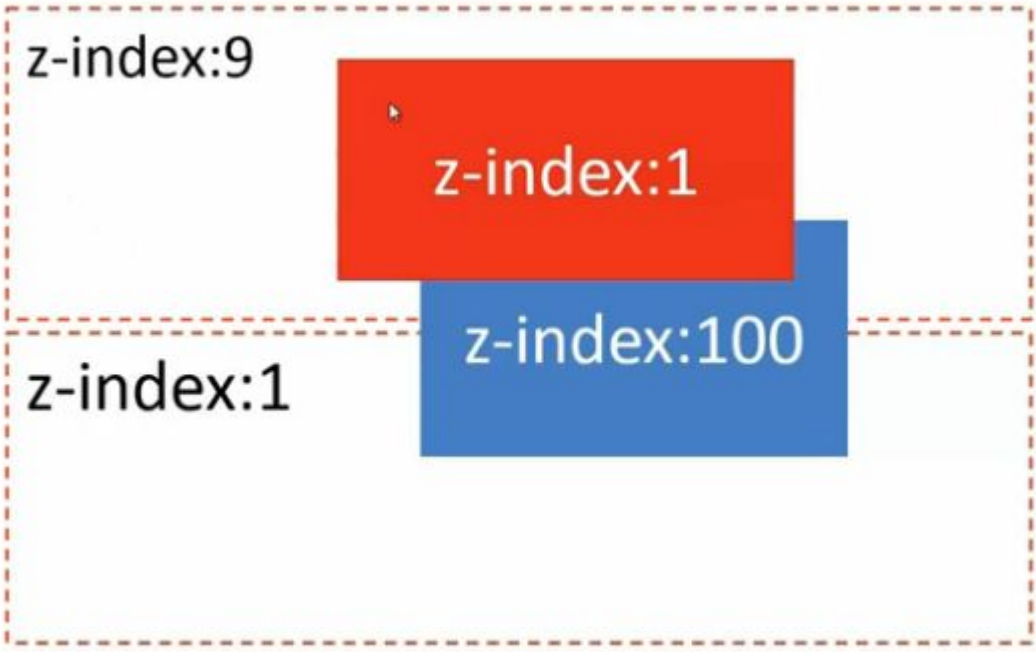
Z-index -设置元素在 z 轴上的排序

z-index 属性设置元素的堆叠顺序。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。

- 元素可拥有负的 z-index 属性值。
- Z-index 仅能在定位元素上奏效（例如 position:absolute;）

值	描述
auto	默认。堆叠顺序与父元素相等。
number	设置元素的堆叠顺序。
inherit	规定应该从父元素继承 z-index 属性的值。

Z-index 栈



Float 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。

float 属性定义元素在哪个方向浮动.

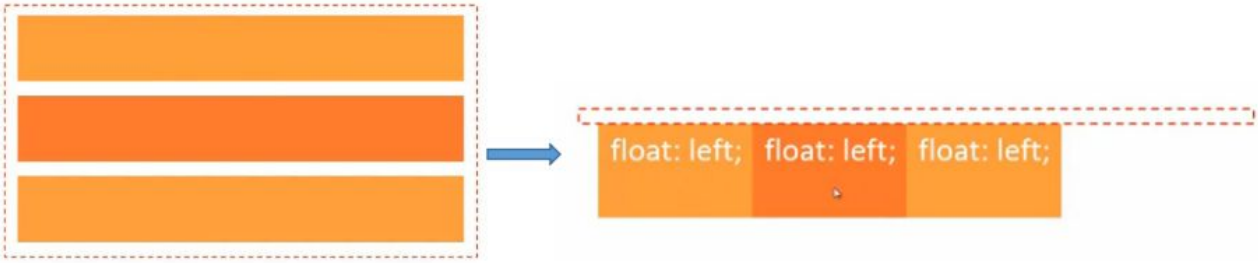
以往这个属性总应用于图像，使文本围绕在图像周围，不过在 CSS 中，任何元素都可以浮动。浮动元素会生成一个块级框，而不论它本身是何种元素。

语法：

```
float: none | left | right
```

Float 浮动基本特征：

1.float 的元素在同一文档流



2.float 元素半脱离文档流

对元素，脱离文档流；对内容，在文档流。

Clear-清除浮动：

元素浮动之后，周围的元素会重新排列，为了避免这种情况，使用 clear 属性。clear 属性规定元素的哪一侧不允许其他浮动元素。

语法：

clear：none | left | right | both

- 应用于(浮动元素)后续元素；
- 应用于块级元素；

使用方式：

1.增加空白元素

2.Clearfix

代码：

```
.clearfix:after{
        Content:”.”;display:block;clear:both;height:0;
        Overflow:hidden;visibility:hidden;}
.clearfix{zoom:1;}
```

Float 应用-两列布局：

```html
<div class="body clearfix">
  <div class="side">side</div>
  <div class="main">main</div>
</div>
```

HTML

```css
.side{float: right;width: 200px;}
.main{float: left;width: 500px;}
.clearfix:after{content: '.';display: block;clear: both;height: 0;
      overflow: hidden;visibility: hidden;}
```

CSS

Flex 伸缩布局:

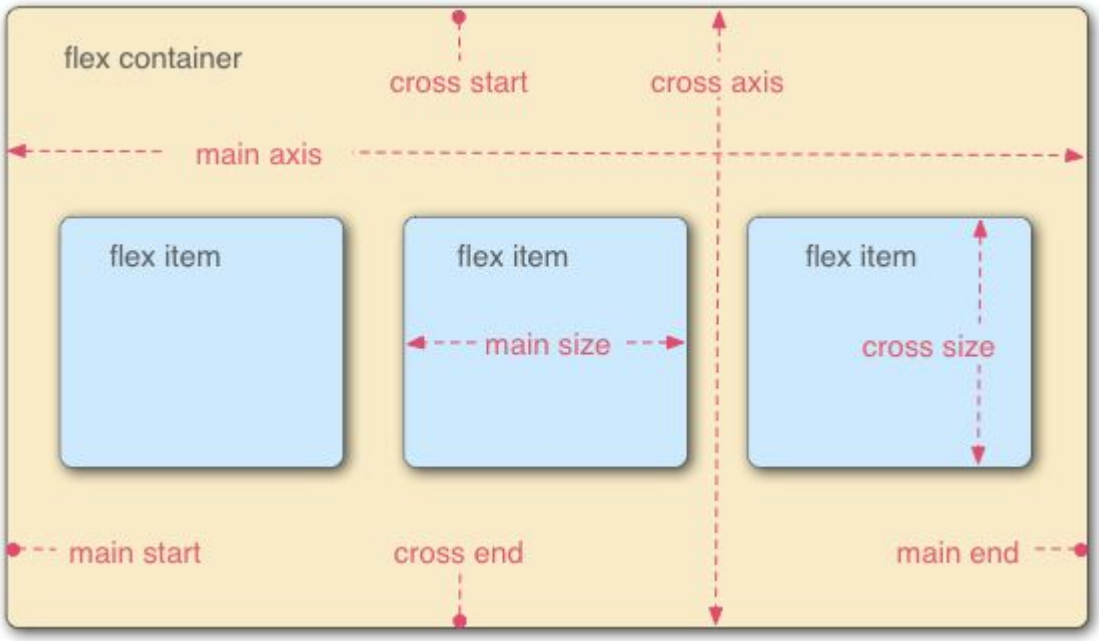
Flex Terms-基本概念：

Flex 容器（flex container）：

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称”容器”。

Flex 项目（flex item）

Flex 容器的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称”项目”。



容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做 main start，结束位置叫做 main end；交叉轴的开始位置叫做 cross start，结束位置叫做 cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做 main size，占据的交叉轴空间叫做 cross size。

创建 flex container 容器：

Display:flex

Flex items（flex 项目）

在文档流中的子元素是弹性子元素

```
<div style="display:flex">
    <div>block</div>                                    √
    <div style="float: left;" >float</div>              √
    <span>inline</span>                                 √
    <div style="position:absloute;"></div>              ×
    <div>
        <div>grandson</div>                             ×
    </div>
</div>
```

弹性布局特性：

与方向有关：

- flex-direction
- Flex-wrap
- Flex-flow
- order

与弹性有关：

- Flex-basis
- Flex-grow
- Flex-shrink
- flex

与对齐有关：

- Justify-content
- Align-items
- Align-self
- Align-content

Flex-direction:

决定主轴的方向（即项目的排列方向）

Flex-direction:row|row-reverse|column|column-reverse

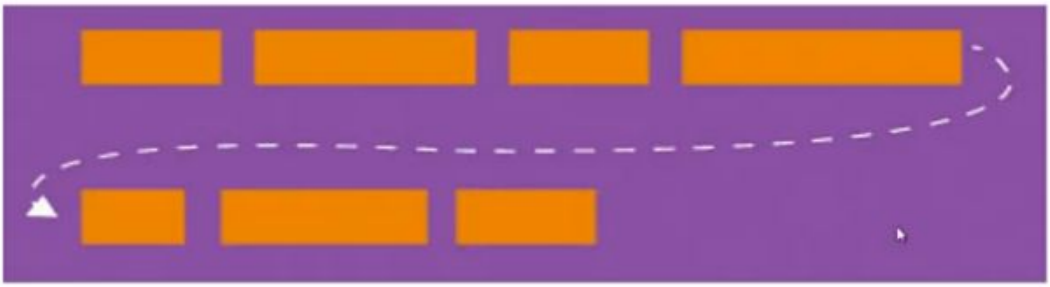
它可能有 4 个值。

- row（默认值）：主轴为水平方向，起点在左端。
- row-reverse：主轴为水平方向，起点在右端。
- column：主轴为垂直方向，起点在上沿。
- column-reverse：主轴为垂直方向，起点在下沿。

Flex-wrap:

默认情况下，项目都排在一条线（又称”轴线”）上。flex-wrap 属性定义，如果一条轴线排不下，如何换行。

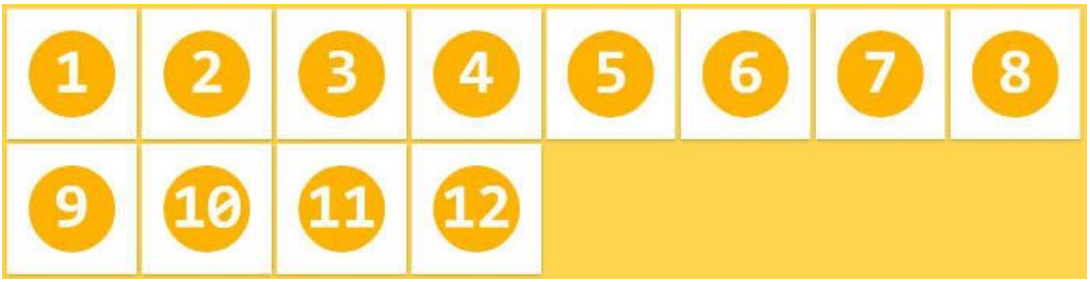
Flex-wrap:nowrap|wrap|wrap-reverse



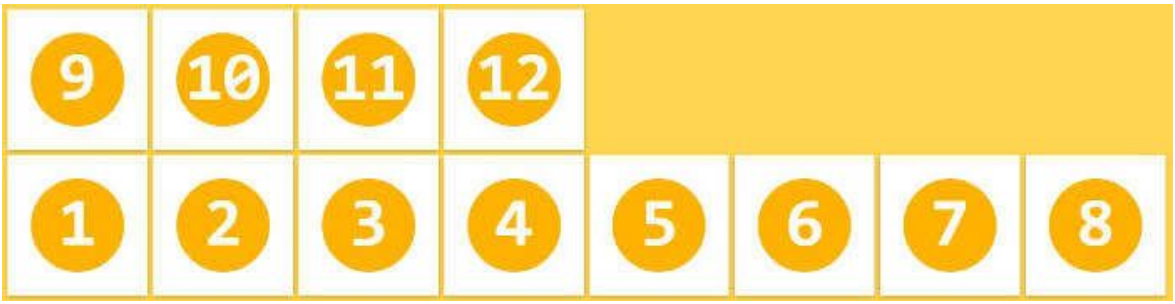
它可能取三个值。

（1）nowrap（默认）：不换行。

（2）wrap：换行，第一行在上方。



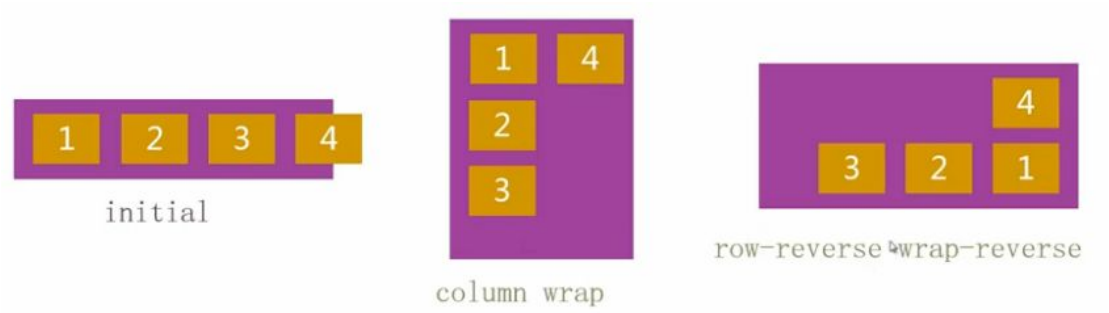
（3）wrap-reverse：换行，第一行在下方。



Flex-flow:

flex-flow属性是flex-direction属性和flex-wrap属性的简写形式，默认值为row nowrap。

Flex-flow:：〈’flex-direction’〉||〈’flex-wrap’〉



Order:

order 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 0。

Order:<interger>
Initial:0（初始值为 0）

Flex-basis

flex-basis 属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为 auto，即项目的本来大小。

- flex-basis: <length> | auto; /* default auto */
- 设置 flex item 的初始宽/高

它可以设为跟 width 或 height 属性一样的值（比如 350px），则项目将占据固定空间。

Flex-grow:

flex-grow 属性定义项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。设置元素所能分配到的空余空间的比例。

- Flex-grow:<number>
- Initial:0

1. 默认情况下，flex-grow:0, flex-item 项目不占据额外空间。
2. 如果只设置一个 flex-item 项目的 flex-grow 为 1，则它将占据剩余空间。

3.如果一个项目的 flex-grow 属性为 2，一个项目为 1，则前者占据的剩余空间比后者多一倍。
4.计算方法：flex-basis + flow-grow/sum(flow-grow) * remain
示例：b,c 将按照 1:3 的比率分配剩余空间
HTML Code：

```
<ul class="flex">
    <li>a</li>
    <li>b</li>
    <li>c</li>
</ul>
```

CSS Code:

```
.flex{display:flex;width:600px;margin:0;padding:0;list-style:none;}
.flex li:nth-child(1){width:200px;}
.flex li:nth-child(2){flex-grow:1;width:50px;}
.flex li:nth-child(3){flex-grow:3;width:50px;}
```

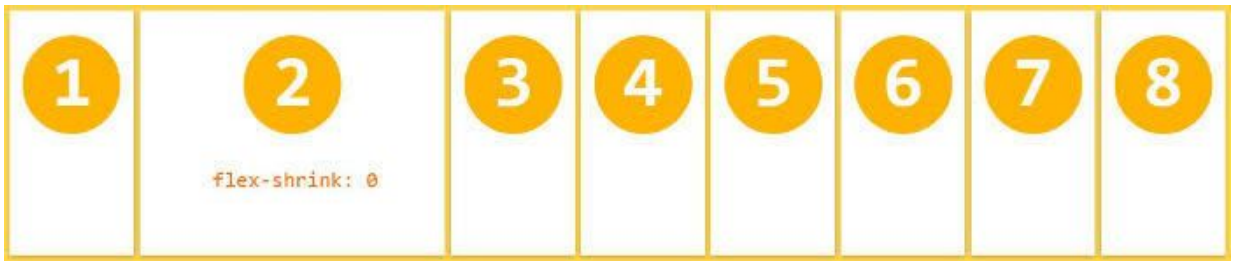
flex-grow 的默认值为 0，如果没有显示定义该属性，是不会拥有分配剩余空间权利的。

本例中 b,c 两项都显式的定义了 flex-grow，flex 容器的剩余空间分成了 4 份，其中 b 占 1 份，c 占 3 分，即 1:3
flex 容器的剩余空间长度为：600-200-50-50=300px，所以最终 a,b,c 的长度分别为：
a: 200+(300/4*0)=200px
b: 50+(300/4*1)=125px
a: 50+(300/4*3)=275px

Flex-shrink:

flex-shrink 属性定义了项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。

- Flex-shrink:<number>
- Initial:1



如果所有项目的 flex-shrink 属性都为 1，当空间不足时，都将等比例缩小。如果一个项目的 flex-shrink 属性为 0，其他项目都为 1，则空间不足时，前者不缩小。
负值对该属性无效。
计算方法：flex-basis + flow-shrink/sum(flow-shrink) * remain
示例：a,b,c 将按照 1:1:3 的比率来收缩空间
HTML Code：

```
<ul class="flex">
    <li>a</li>
    <li>b</li>
```

```
    <li>c</li>
</ul>
```

CSS Code:

```
.flex{display:flex;width:400px;margin:0;padding:0;list-style:none;}
.flex li{width:200px;}
.flex li:nth-child(3){flex-shrink:3;}
```

flex-shrink 的默认值为 1，如果没有显示定义该属性，将会自动按照默认值 1 在所有因子相加之后计算比率来进行空间收缩。

本例中 c 显式的定义了 flex-shrink，a,b 没有显式定义，但将根据默认值 1 来计算，可以看到总共将剩余空间分成了 5 份，其中 a 占 1 份，b 占 1 份，c 占 3 分，即 1:1:3

我们可以看到父容器定义为 400px，子项被定义为 200px，相加之后即为 600px，超出父容器 200px。那么这么超出的 200px 需要被 a,b,c 消化

通过收缩因子，所以加权综合可得 200*1+200*1+200*3=1000px；

于是我们可以计算 a,b,c 将被移除的溢出量是多少：

a 被移除溢出量：(200*1/1000)*200，即约等于 40px

b 被移除溢出量：(200*1/1000)*200，即约等于 40px

c 被移除溢出量：(200*3/1000)*200，即约等于 120px

最后 a,b,c 的实际宽度分别为：200-40=160px，200-40=160px，200-120=80px

Flex

flex 属性是 flex-grow，flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto。后两个属性可选。

- Flex: [<'flex-grow'> <'flex-shrink'>? || <'flex-basis'>]|none
- Initial:0 1 auto

该属性有两个快捷值：auto (1 1 auto) 和 none (0 0 auto)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

取值：

- none： none 关键字的计算值为: 0 0 auto
- <' flex-grow '>： 用来指定扩展比率，即剩余空间是正值时此「flex 子项」相对于「flex 容器」里其他「flex 子项」能分配到空间比例。
  在「flex」属性中该值如果被省略则默认为「1」
- <' flex-shrink '>： 用来指定收缩比率，即剩余空间是负值时此「flex 子项」相对于「flex 容器」里其他「flex 子项」能收缩的空间比例。
  在收缩的时候收缩比率会以伸缩基准值加权
  在「flex」属性中该值如果被省略则默认为「1」
- <' flex-basis '>： 用来指定伸缩基准值，即在根据伸缩比率计算出剩余空间的分布之前，「flex 子项」长度的起始数值。
  在「flex」属性中该值如果被省略则默认为「0%」
  在「flex」属性中该值如果被指定为「auto」，则伸缩基准值的计算值是自身的 <' width '> 设置，如果自身的宽度没有定义，则长度取决于内容。

- 如果缩写「flex: 1」，则其计算值为「1 1 0%」
- 如果缩写「flex: auto」，则其计算值为「1 1 auto」
- 如果「flex: none」，则其计算值为「0 0 auto」

- 如果「flex: 0 auto」或者「flex: initial」，则其计算值为「0 1 auto」，即「flex」初始值

示例：如下情况每个元素的计算宽是多少

HTML Code：

```
<ul class="flex">
    <li>a</li>
    <li>b</li>
    <li>c</li>
</ul>
```

CSS Code:

```
.flex{display:flex;width:800px;margin:0;padding:0;list-style:none;}
.flex :nth-child(1){flex:1 1 300px;}
.flex :nth-child(2){flex:2 2 200px;}
.flex :nth-child(3){flex:3 3 400px;}
```

本例定义了父容器宽（即主轴宽）为 800px，由于子元素设置了伸缩基准值 flex-basis，相加 300+200+400=900，那么子元素将会溢出 900-800=100px；
由于同时设置了收缩因子，所以加权综合可得 300*1+200*2+400*3=1900px；
于是我们可以计算 a,b,c 将被移除的溢出量是多少：
a 被移除溢出量：(300*1/1900)*100，即约等于 16px
b 被移除溢出量：(200*2/1900)*100，即约等于 21px
c 被移除溢出量：(400*3/1900)*100，即约等于 63px
最后 a,b,c 的实际宽度分别为：300-16=284px，200-21=179px，400-63=337px

仍然是上面这个例子，不过将容器的宽度改成了 1500px

HTML Code：

```
<ul class="flex">
    <li>a</li>
    <li>b</li>
    <li>c</li>
</ul>
```

CSS Code:

```
.flex{display:flex;width:1500px;margin:0;padding:0;list-style:none;}
.flex :nth-child(1){flex:1 1 300px;}
.flex :nth-child(2){flex:2 2 200px;}
.flex :nth-child(3){flex:3 3 400px;}
```

本例定义了父容器宽（即主轴宽）为 1500px，由于子元素设置了伸缩基准值 flex-basis，相加 300+200+400=900，那么容器将有 1500-900=600px 的剩余宽度；
于是我们可以计算 a,b,c 将被扩展量是多少：
a 的扩展量：(1/(1+2+3))*600，即约等于 100px
b 的扩展量：(2/(1+2+3))*600，即约等于 200px
c 的扩展量：(3/(1+2+3))*600，即约等于 300px
最后 a,b,c 的实际宽度分别为：300+100=400px，200+200=400px，400+300=700px
从本例能看出：

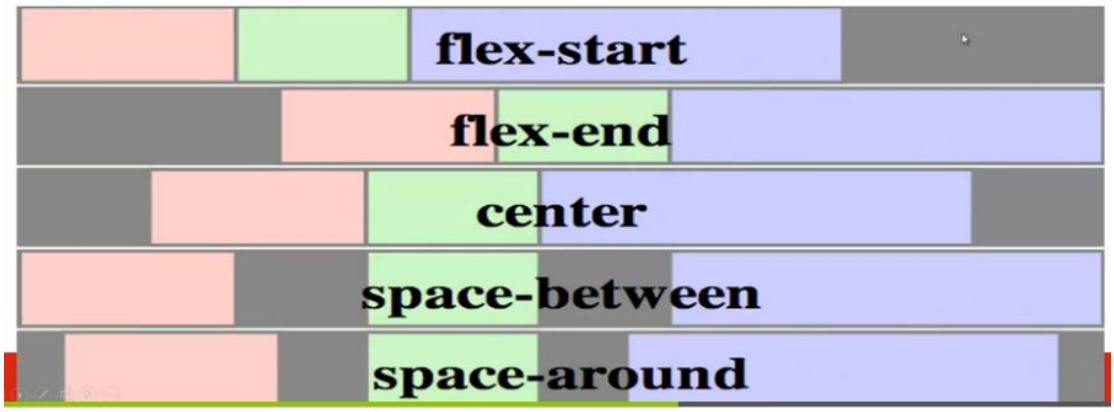
1. 当「flex-basis」在「flex」属性中不为 0 时（包括值为 auto，此时伸缩基准值等

于自身内容宽度），「flex 子项」将分配容器的剩余空间（剩余空间即等于容器宽度减去各项的伸缩基准值）

2. 当「flex-basis」在「flex」属性中等于 0 时，「flex 子项」将分配容器的所有空间（因为各项的伸缩基准值相加等于 0，剩余空间等于容器宽度减去各项的伸缩基准值，即减 0，最后剩余空间值等于容器宽度），所以可以借助此特性，给各子项定义「flex: n」来进行按比例均分容器总宽度

Justify-content

- justify-content: flex-start | flex-end | center | space-between | space-around
- 定义了**项目在主轴(main-axis)上的对齐方式。**

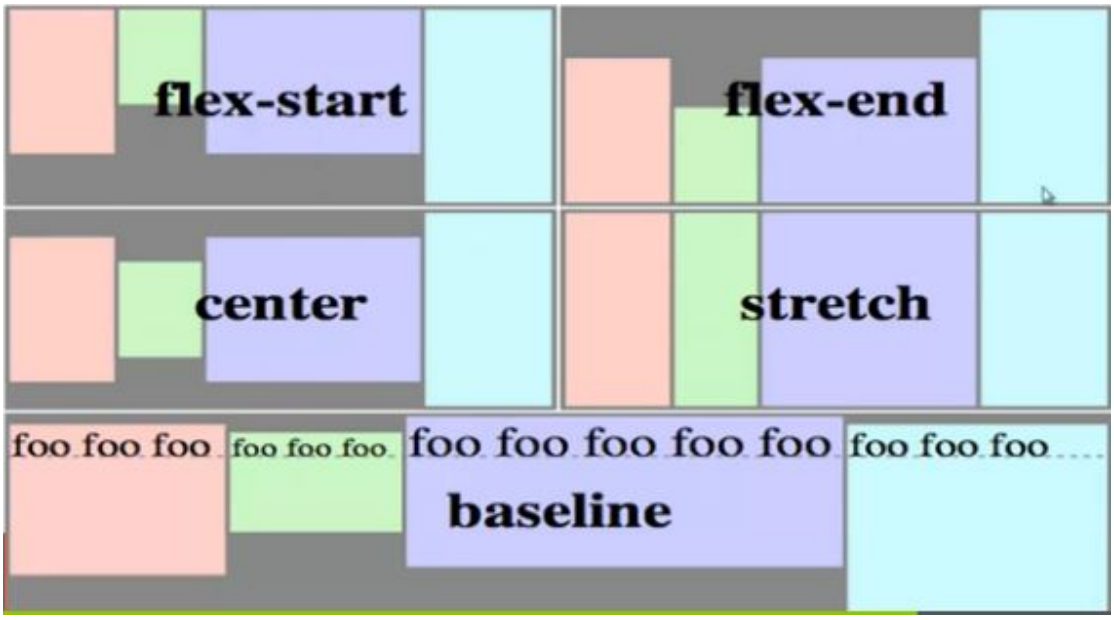


它可能取 5 个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。

- flex-start（默认值）：左对齐
- flex-end：右对齐
- center： 居中
- space-between：两端对齐，项目之间的间隔都相等。
- space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

Align-items

- align-items: flex-start | flex-end | center | baseline | stretch
- 定义**项目在交叉轴(辅轴，cross-axis)上如何对齐。**



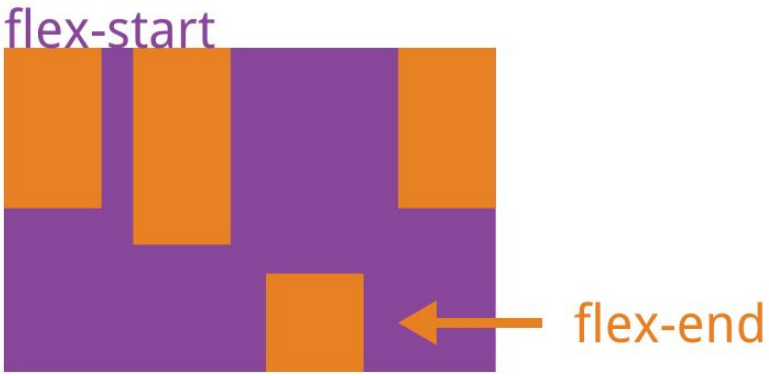
它可能取 5 个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。

- flex-start：交叉轴的起点对齐。
- flex-end：交叉轴的终点对齐。
- center：交叉轴的中点对齐。
- baseline：项目的第一行文字的基线对齐。
- stretch（默认值）：如果项目未设置高度或设为 auto，将占满整个容器的高度。

Align-self

align-self 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。默认值为 auto，表示继承父元素的 align-items 属性，如果没有父元素，则等同于 stretch。

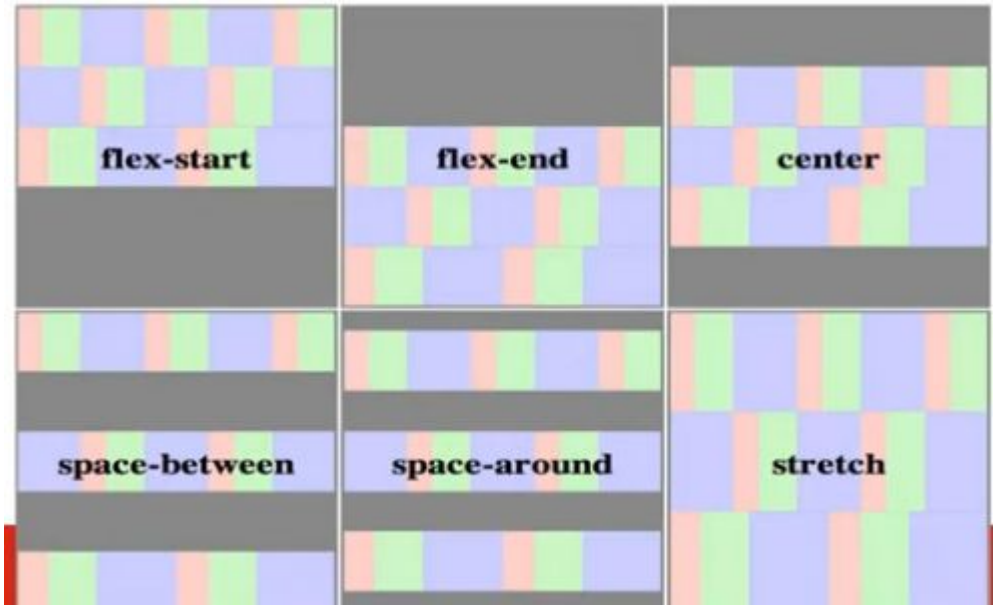
- align-self: auto | flex-start | flex-end | center | baseline | stretch
- 设置单个 flex item 在 cross-axis 方向上对齐方式。



Align-content

align-content 属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

- align-content: flex-start | flex-end | center | space-between | space-around | stretch
- 设置 cross-axis 方向上行对齐方式



该属性可能取 6 个值。

- flex-start：与交叉轴的起点对齐。
- flex-end：与交叉轴的终点对齐。
- center：与交叉轴的中点对齐。
- space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。
- space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍。
- stretch（默认值）：轴线占满整个交叉轴。

Flex 布局应用-三行两列自适应布局

```html
<div class="head">head</div>
<div class="body">
        <div class="side">side</div>
        <div class="main">main</div>
</div>
<div class="foot">foot</div>
```

HTML

```css
body{display: flex;flex-flow: column;}
.head, .foot{height: 100px;}
.body{flex: 1;display: flex;}
.side{width: 200px;}
.main{flex: 1;}
```

CSS

关于 flex 布局，可参看资源：

http://www.xifengxx.com/seo/%E5%89%8D%E7%AB%AF%E5%AD%A6%E4%B9%A0/1408.html