CSS

CSS文字属性

规定文字样式的属性

```
font-style: italic;  倾斜的
font-style: normal;正常的
```

规定文字粗细的属性

```
font-weight: bold;加粗
单词取值: bold加粗 bolder 更粗 lighter 细线（默认）
数字取值:100-900整百数字
```

规定文字大小的属性

```
font-size: 90px;
```

规定文字字体的属性

```
font-family: "微软雅黑";

如果取值是中文，需要用双引号或者单引号，而且设置的字体必须是用户电脑安装的字体
```

文字字体属性补充

1. 如果设置的字体不存在，会默认使用系统的字体宋体
2. 如果设置的字体不存在，而我们又不想使用默认的字体怎么办?

   ```
   font-family: "乱七八糟的字体","微软雅黑";  给字体设置备选方案
   ```

3. 如果想给文本中的英文和中文设置不同的字体怎么办?

   ```
   font-family: "Times New Roman","微软雅黑";
   ```

   如果想给界面的英文设置字体，英文必须放前边，可以实现英文是一种字体，中文是备选字体

   并不是英文名称的就是英文字体，中文也有英文字体，是不是中文字体，要看能不能处理中文字体

文字属性的简写

```
font: italic bold 20px "微软雅黑";
```

> 在这种格式中 style weight 都是可以省略的，size和family不能省略
>
> 并且style 和 weight 位置可交换，size和family不能交换位置且要写在最后

文本属性

文本装饰的属性

```
text-decoration: underline;下划线
text-decoration: line-through;删除线
text-decoration: overline;上划线
text-decoration: none;什么都没有（去掉超链接的下划线）
```

文本水平对齐的属性

```
text-align: center;
text-align: right;
text-align: left;
```

文本缩进的属性

```
text-indent: 2em;  段落首行缩进2个文字
1em代表一个文字，px也可以但不常用
```

颜色属性

```
color: red;
color: rgb(255,0,0 );
color: rgba(255,0,0,1);
color: #ff0000;
```

重点不是color属性，而是其取值的方式

```
格式 color: red;
```

```
rgb: 三原色 red green blue --rgb(0,0,0),括号中写三种颜色的亮度，0-255，0代表不发光，255最亮在前端开发中，常用到灰色，只需要让三个值相等，并且值越接近255，就越接近白色，越接近0就越接近黑色
```

```
rgba: 其中rgb和上边代表一样，a代表透明度，a取值0-1，越小越透明
```

```
十六进制 #000000
    其本质就是RGB,在十六进制中每2位表示一个颜色
    例如 #FFEE00 FF表示R EE表示G 00表示B
    FF转为十进制等于255     00转为十进制等于0
    只要十六进制的每两位都是一样的，就可以简写成一个
    即#FFEE00= #FE0 ,两位数不一样不能简写
```

选择器

标签选择器

```
根据指定的标签名称，在当前界面中找到所有该名称的标签，然后设置属性
```

```
格式：
标签名称{
    属性:属性值
}
p{
    color: #FF0000;
 }
```

注意点：

- 标签选择器，选中的当前界面中所有的标签,这也是它的局限性
- 无论标签藏的多深，都能找到
- 只要是HTML中的标签，都可以作为标签选择器

id选择器

```
 #id1{
    color: red;
 }

<p id="id1">迟到毁一生</p>
```

注意点：

- id在同一页面不能重复，id不能以数字开头，id名不能是标签名
- 在企业开发中，一般情况下id是留给js调用的，我们要随便使用，我们可以换一种选择器，比如类选择器

类选择器

```
根据指定的类名称找到对应的标签，然后设置属性
```

```
 .pp{
     color: red;
 }
 <p class="pp">迟到毁一生</p>
```

注意点：

- 每个HTML标签都有一个属性叫做class, 也就是说每个标签都可以设置类名
- 在同一个界面中class的名称是可以重复的
- 在编写class选择器时一定要在class名称前面加上.
- 类名的命名规范和id名称的命名规范一样
- 类名就是专门用来给CSS设置样式的，id留给js
- 在HTML中每个标签可以同时绑定多个类名

  ```
  <标签名称 class="类名1 类名2 ...">
  错误的写法:<p class="para1" class="para2">
  ```

id选择器和类选择器的区别

1. id相当于人的身份证不可以重复,class相当于人的名称可以重复
2. 一个HTML标签只能绑定一个id名称一个,HTML标签可以绑定多个class名称

3. id选择器是以#开头,class选择器是以.开头
4. id一般情况下是给js使用的, 所以除非特殊情况, 否则不要使用id去设置样式
5. 一般情况下在企业开发中要注重冗余代码的抽取, 可以将一些公共的代码抽取到一个类选择器中, 然后让标签和这个类选择器绑定即可

后代选择器

```
找到指定标签的所有特定的后代标签, 设置属性
```

```
格式:
标签名称1 标签名称2{
    属性:值;
}
先找到所有名称叫做"标签名称1"的标签, 然后再在这个标签下面去查找所有名称叫做"标签名称2"的标签, 然后再设置属性
 div ul li p{
      color: red;
 }
```

注意点:

- 后代选择器必须用空格隔开
- 后代不仅仅是儿子, 也包括孙子/重孙子, 只要最终是放到指定标签中的都是后代
- 后代选择器不仅仅可以使用标签名称, 还可以使用其它选择器, 比如类选择器 .class1 .class2
- 后代选择器可以通过空格一直延续下去

子元素选择器

```
找到指定标签中所有特定的直接子元素, 然后设置属性
```

```
格式:
标签名称1>标签名称2{
    属性:值;
}
先找到所有名称叫做"标签名称1"的标签, 然后在这个标签中查找所有直接子元素名称叫做"标签名称2"的元素
  div>ul>li>p{
     color: purple;
  }
  ...
  <div id="identity">
      <p>我是段落</p>
      <p>我是段落</p>
      <ul>
          <li><p>我是段落</p></li>
      </ul>
  </div>
```

注意点:

- 子元素选择器只会直接查找儿子, 不会查找其他被嵌套的标签
- 子元素选择器之间需要用>符号连接, 并且前后不能有空格
- 子元素选择器不仅仅可以使用标签名称,**还可以使用其它选择器**
- 子元素选择器可以通过>符号一直延续下去

后代选择器和子元素选择器的区别和选择

> 区别:
> 1. 后代选择器使用空格作为连接符号,子元素选择器使用>作为连接符号
> 2. 后代选择器会选中指定标签中的所有的特定后代标签, 也就是会选中儿子/孙子..., 只要是被放到指定标签中的特定标签都会被选中
> 子元素选择器只会选中指定标签中, 所有的特定的直接标签, 也就是只会选中特定的儿子标签
> 共同点:
> 1. 后代选择器和子元素选择器都可以使用标签名称/id名称/class名称来作为选择器
> 2. 后代选择器和子元素选择器都可以通过各自的连接符号一直延续下去
> 如何选择?
> 如果想选中指定标签中的所有特定的标签, 那么就使用后代选择器
> 如果只想选中指定标签中的所有特定儿子标签, 那么就使用子元素选择器

交集选择器

给所有选择器选中的标签中, 相交的那部分标签设置属性

```
格式:
选择器1选择器2{
    属性: 值;
}
例如:
 .para1#identity{
       color: blue;
  }

<p>我是段落</p>
<p class="para1" id="identity">我是段落</p> /*只会选中该条*/
<p class="para1">我是段落</p>
```

注意点:

- 选择器和选择器之间没有任何的连接符号
- 选择器可以使用标签名称/id名称/class名称
- 交集选择器仅仅作为了解, 企业开发中用的并不多

并集选择器

给所有选择器选中的标签设置属性

```
格式:
选择器1,选择器2{
    属性:值;
}

.ht,.para{
    color: red;
}
```

注意点:

- 并集选择器必须使用,来连接
- 选择器可以使用标签名称/id名称/class名称

兄弟选择器

```
相邻兄弟选择器:
    给指定选择器后面紧跟的某个选择器选中的标签设置属性
```

```
格式:
选择器1+选择器2{
    属性:值;
}
```

注意点:

- 相邻兄弟选择器必须通过+连接
- 相邻兄弟选择器只能选中紧跟其后的那个标签, 不能选中被隔开的标签

```
通用兄弟选择器 CSS3
    给指定选择器后面的所有选择器选中的所有标签设置属性
```

```
格式:
选择器1~选择器2{
    属性:值;
}
```

注意点:

- 通用兄弟选择器必须用~连接
- 通用兄弟选择器选中的是指定选择器后面某个选择器选中的所有标签, 无论有没有被隔开都可以选中

序选择器

```
CSS3中新增的选择器最具代表性的就是序选择器
```

```
同级别的第几个 --只管同级别的，不区分是什么类型的标签
:first-child 选中同级别中的第一个标签
:last-child 选中同级别中的最后一个标签
:nth-child(n) 选中同级别中的第n个标签，常用
:nth-last-child(n) 选中同级别中的倒数第n个标签
:only-child 选中父元素中唯一的标签
```

注意点：不区分类型

```
同类型的第几个--既是同级别 又是同类中的第几个
:first-of-type 选中同级别中同类型的第一个标签
:last-of-type  选中同级别中同类型的最后一个标签
:nth-of-type(n) 选中同级别中同类型的第n个标签
:nth-last-of-type(n)  选中同级别中同类型的倒数第n个标签
:only-of-type 选中父元素中唯一类型的某个标签
```

素材:

```
<!--
```

```
<h1>我是标题</h1>
<p>我是段落1</p>
<p>我是段落2</p>
<p>我是段落3</p>
<p>我是段落4</p>
<div>
    <p>我是段落5</p>
    <p>我是段落6</p>
    <p>我是段落7</p>
    <p>我是段落8</p>
</div>
-->
<p class="para">我是段落1</p>
<div>
    <p class="para">我是段落2</p>
    <p class="para">我是段落2</p>
    <h1>我是标题</h1>
</div>
```

需选择器补充

```
:nth-child(odd) 选中同级别中的所有奇数
:nth-child(even) 选中同级别中的所有偶数

:nth-child(xn+y)
x和y是用户自定义的，而n是一个计数器，从0开始递增
```

属性选择器

```
根据指定的属性名称找到对应的标签，然后设置属性
```

```
格式:
[attribute]
作用:根据指定的属性名称找到对应的标签，然后设置属性

[attribute=value]
作用: 找到有指定属性，并且属性的取值等于value的标签，然后设置属性

最常见的应用场景，就是用于区分input属性
input[type=password]{}
<input type="text" name="" id="">
<input type="password" name="" id="">
```

```
p[id]{
    color: red;
}
p[class=cc]{
    color: blue;
}

...

<p id="identity1">我是段落1</p>
<p id="identity2" class="cc">我是段落2</p>
<p class="cc">我是段落3</p>
```

属性选择器补充

```
1.属性的取值是以什么开头的
[attribute|=value] CSS2
[attribute^=value] CSS3
两者之间的区别:
CSS2中的只能找到value开头,并且value是被-和其它内容隔开的
CSS3中的只要是以value开头的都可以找到, 无论有没有被-隔开

2.属性的取值是以什么结尾的
[attribute$=value] CSS3

3.属性的取值是否包含某个特定的值得
[attribute~=value] CSS2
[attribute*=value] CSS3
两者之间的区别:
CSS2中的只能找到独立的单词, 也就是包含value,并且value是被空格隔开的
CSS3中的只要包含value就可以找到

<img src="" alt="abcdef">
<img src="" alt="abc-www">
<img src="" alt="abc ppp">
<img src="" alt="defabc">
<img src="" alt="ppp abc">
<img src="" alt="www-abc">
<img src="" alt="qq">
<img src="" alt="yy">
<img src="" alt="abcwwwmmm">
<img src="" alt="wwwmmmabc">
<img src="" alt="wwwabcmmm">
<img src="" alt="www-abc-mmm">
<img src="" alt="www abc mmm">
<img src="" alt="qq">
```

通配符选择器

给当前界面上所有的标签设置属性

```
格式:
*{
    属性:值;
}
```

由于通配符选择器是设置界面上所有的标签的属性, 所以在设置之前会遍历所有的标签, 如果当前界面上的标签比较多, 那么性能就会比较差, 所以在企业开发中一般不会使用通配符选择器

CSS三大特性

继承性

> 给父元素设置一些属性, 子元素也可以使用, 这个我们就称之为继承性

注意点:

- 并不是所有的属性都可以继承, 只有以color/font-/text-/line-开头的属性才可以继承
- 在CSS的继承中不仅仅是儿子可以继承, 只要是后代都可以继承

- a(超链接)标签的文字颜色和下划线是不能继承的
- h(标题)标签的文字大小是不能继承的

应用场景:

```
一般用于设置网页上的一些共性信息, 例如网页的文字颜色, 字体,文字大小等内容 body{}
```

层叠性

> 层叠性就是CSS处理冲突的一种能力

注意点

- 层叠性只有在多个选择器选中"同一个标签", 然后又设置了"相同的属性", 才会发生层叠性

优先级

> 当多个选择器选中同一个标签, 并且给同一个标签设置相同的属性时, 如何层叠就由优先级来确定

优先级判断的三种方式:

1. 间接选中就是指继承

   > 如果是间接选中, 那么就是谁离目标标签比较近就听谁的]

2. 相同选择器(直接选中)

   > 如果都是直接选中, 并且都是同类型的选择器, 那么就是谁写在后面就听谁的

3. 不同选择器(直接选中)

   > 如果都是直接选中, 并且不是相同类型的选择器, 那么就会按照选择器的优先级来层叠
   >
   > ```
   > id>类选择器>标签选择器>通配符选择器>继承>浏览器默认
   > ```

优先级之！important

```
用于提升某个直接选中标签的选择器中的某个属性的优先级的, 可以将被指定的属性的优先级提升为最高
```

注意点:

- !important只能用于直接选中, 不能用于间接选中
- 通配符选择器选中的标签也是直接选中的
- !important只能提升被指定的属性的优先级, 其它的属性的优先级不会被提升
- !important必须写在属性值得分号前面
- !important前面的感叹号不能省略

优先级之权重问题

```
当多个选择器混合在一起使用时, 我们可以通过计算权重来判断谁的优先级最高
```

权重的计算规则:

- 首先先计算选择器中有多少个id, id多的选择器优先级最高
- 如果id的个数一样, 那么再看类名的个数, 类名个数多的优先级最高
- 如果类名的个数一样, 那么再看标签名称的个数, 标签名称个数多的优先级最高
- 如果id个数一样, 类名个数也一样, 标签名称个数也一样, 那么就不会继续往下计算了, 那么此时谁写在后面听也就是说优先级如果一样, 那么谁写在后面听谁的谁的

注意点:

```
只有选择器是直接选中标签的才需要计算权重，否则一定会听直接选中的选择器的
```

div和span标签

什么是div标签?

```
一般用于配合css完成网页的基本布局
```

什么是span标签?

```
一般用于配合css修改网页中的一些局部信息的样式
```

div和span有那些区别?

- div会单独占一行，而span不回单独占一行
- div是一个容器级的标签，和span是一个文本级的标签

容器级和文本级的区别?

- 容器级的标签中可以嵌套其他所有的标签
- 文本级的标签只能嵌套文字、图片、超链接

容器级的标签有哪些?

```
div h ul ol dl li dt dd ...
```

文本级的标签有哪些?

```
span p buis strong em ins del ...
```

注意点:

哪些标签是文本级的哪些标签是容器级的, 我们不用刻意去记忆, 在企业开发中一般情况下要嵌套都是嵌套在div中, 或者按照组标签来嵌套

CSS的显示模式

在HTML中HTML将所有标签分为两类，分别是容器级和文本级

在CSS中CSS也将所有的标签分为两类, 分别是块级元素和行内元素

块级元素和行内元素

```
块级元素会独占一行
行内元素不会独占一行
```

```
容器级的标签
div h ul ol dl li dt dd ...
文本级的标签
span p buis stong em ins del ...

块级元素
p div h ul ol dl li dt dd
行内元素
span buis strong em ins del
```

块级元素和行内元素的区别?

1. 块级元素

   > 独占一行
   >
   > 如果没有设置宽度，那么默认和父元素一样宽
   >
   > 如果设置了宽高，那么就按照设置来显示

2. 行内元素

   > 不会独占一行
   >
   > 如果没有设置宽高，那么默认和内容一样宽
   >
   > 行内元素是不能设置宽高的

3. 行内块级元素

   > 为了能够让元素既能够不独占一行, 又可以设置宽度和高度, 那么就出现了行内块级元素

CSS显示模式的转换

如何转换css显示模式?

```
设置元素的display属性
```

display的取值:

```
block 块级
inline 行内
inline-block 行内块级
```

快捷键webstorm

```
di display: inline;
db display: block;
dib display: inline-block;
```

背景颜色

```
在CSS中有一个background-color:属性, 就是专门用来设置标签的背景颜色的
```

背景图片

```
在CSS中有一个叫做background-image: url();的属性, 就是专门用于设置背景图片的
```

```
background-image: url(images/girl.jpg);
background-image:
url(http://img4.imgtn.bdimg.com/it/u=2278032206,4196312526&fm=21&gp=0.jpg);
```

注意点:

- 图片的地址必须放在url()中, 图片的地址可以是本地的地址, 也可以是网络的地址
- 如果图片的大小没有标签的大小大, 那么会自动在水平和垂直方向平铺来填充
- 如果网页上出现了图片, 那么浏览器会再次发送请求获取图片

背景图片的平铺属性

如何控制背景图片的平铺方式?

```
在CSS中有一个background-repeat属性, 就是专门用于控制背景图片的平铺方式的
```

取值:

```
repeat 默认, 在水平和垂直都需要平铺
no-repeat 在水平和垂直都不需要平铺
repeat-x 只在水平方向平铺
repeat-y 只在垂直方向平铺
```

应用场景:

```
可以通过背景图片的平铺来降低图片的大小, 提升网页的访问速度
```

背景图片定位属性

如何控制背景图片的位置?

```
在CSS中有一个叫做background-position:属性, 就是专门用于控制**背景图片**的位置
```

格式:

```
background-position: 水平方向 垂直方向;
```

```
background-image: url(images/yxlm.jpg);
background-position: center top;
```

取值:

1、具体的方位词

```
水平方向: left center right
垂直方向: top center bottom
```

2、具体的像素

```
例如: background-position: 100px 200px;
记住一定要写单位, 也就是一定要写px
记住具体的像素是可以接收负数的
```

注意点:

同一个标签可以同时设置背景颜色和背景图片，如果颜色和图片同时存在，那么图片会覆盖颜色

背景缩写

背景属性缩写的格式

```
background: 背景颜色 背景图片 平铺方式 关联方式 定位方式;
background: #fff url() 0 0 no-repeat;
```

注意点:

background属性中， 任何一个属性都可以被省略

背景关联方式

什么是背景关联方式?

默认情况下背景图片会随着滚动条的滚动而滚动， 如果不想让背景图片随着滚动条的滚动而滚动， 那么我们就可以修改背景图片和滚动条的关联方式

如何修改背景关联方式?

在CSS中有一个叫做background-attachment的属性， 这个属性就是专门用于修改关联方式的

格式:

```
background-attachment：scroll;
```

取值:

```
scroll 默认值， 会随着滚动条的滚动而滚动
fixed 不会随着滚动条的滚动而滚动
```

```
快捷键:ba background-attachment:;
```

背景图片和插入图片的区别

插入图片

```
<div class="box2">
    <img src="images/girl.jpg" alt="">
    我是文字
</div>
```

背景图片

```
 .box1{
       background-image: url(images/girl.jpg);
       background-repeat:no-repeat;
      }
...
<div class="box1">我是文字</div>
```

区别:

```
1、背景图片仅仅是一个装饰，不会占用位置
插入图片会占用位置
2、背景图片有定位属性，所以可以很方便的控制图片的位置
插入图片没有定位属性，所有控制图片的位置不太方便
3、插入图片的语义比背景图片的语义要强，所以在企业开发中如果你的图片想被搜索引擎收录，那么推荐使用插入图片
```

CSS精灵图

什么是css精灵图?

```
CSS精灵图是一种图像合成技术
```

css精灵图的作用?

```
可以减少请求的次数，以及可以降低服务器处理压力
```

如何使用精灵图?

```
CSS的精灵图需要配合背景图片和背景定位来使用
```

```
.box{
      width: 86px;
      height: 28px;
      background-image: url(images/weibo.png);
      background-position: -425px -200px;
    }
...
<div class="box"></div>
```

边框属性

什么是边框?

> 边框就是环绕在标签宽度和高度周围的线条

语法:

```
border: 边框的宽度 边框的样式 边框的颜色;
```

```
border-top: 边框的宽度 边框的样式 边框的颜色;
border-right: 边框的宽度 边框的样式 边框的颜色;
border-bottom: 边框的宽度 边框的样式 边框的颜色;
border-left: 边框的宽度 边框的样式 边框的颜色;
```

```
连写(分别设置四条边的边框)
border-width: 上 右 下 左;
border-style: 上 右 下 左;
border-color: 上 右 下 左;
```

```
非连写(方向+要素)
border-left-width: 20px;
border-left-style: double;
border-left-color: pink;
```

```
快捷键:bd+ border: 1px solid #000;
```

边距

内边距

> 边框和内容之间的距离就是内边距

语法

```
padding: 上 右 下 左;
```

```
padding-top: 20px ;
padding-right: 40px ;
padding-bottom: 80px ;
padding-left: 160px ;
```

注意点

```
1.给标签设置内边距之后, 标签占有的宽度和高度会发生变化
2.给标签设置内边距之后, 内边距也会有背景颜色
```

外边距

> 标签和标签之间的距离就是外边距

语法

```
margin: 上 右 下 左;
```

```
margin-top:30px ;
margin-right: 60px;
margin-bottom: 120px;
margin-left: 240px;
```

注意点

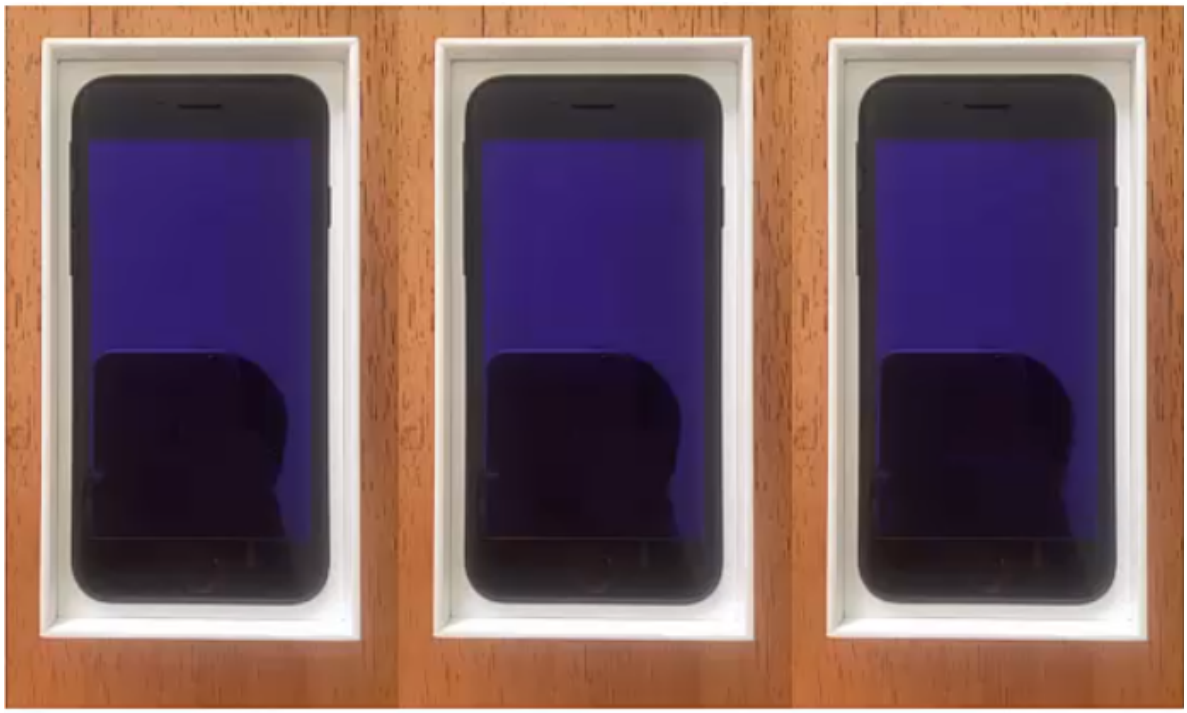
外边距的那一部分是没有背景颜色的

外边距合并现象

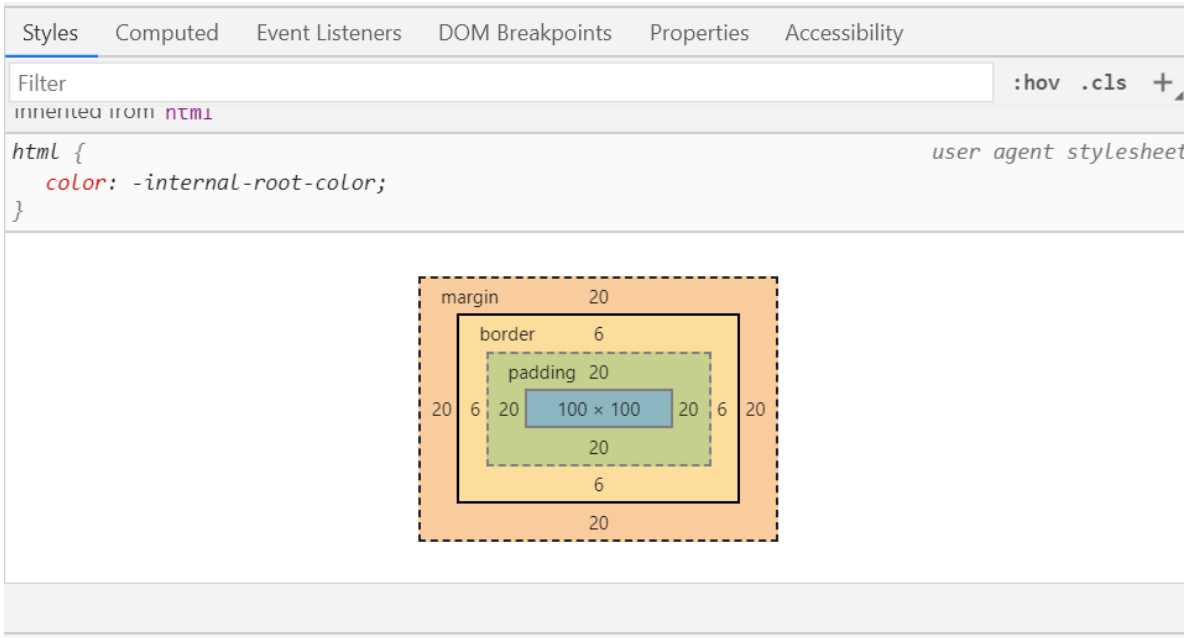
在默认布局的垂直方向上，默认情况下外边距是不会叠加的，会出现合并现象，谁的外边距比较大就听谁的

CSS盒子模型

> CSS盒子模型仅仅只是一个比喻，HTMl中的所有标签都是盒子



在浏览器开发者工具中的style中就是一个盒子模型。



在HTML中所有的标签都可以设置：

宽度/高度 == 指定可以存放内容的区域

内边距 == 填充物

边框 == 手机盒子自己

外边距 == 盒子和盒子之间的间隙

盒子模型的宽度和高度

1. 内容的宽度和高度
   就是通过width/height属性设置的宽度和高度
2. 元素的宽度和高度
   宽度 = [左边框宽度 + 左内边距宽度 + 内容width + 右内边距宽度 + 右边框宽度]
   高度 = [同理可证]
3. 元素空间的宽度和高度
   宽度 = [左外边距 + 左边框 + 左内边距 + width + 右内边距 + 右边框 + 右外边距]
   高度 = [同理可证]

盒子的box-size属性

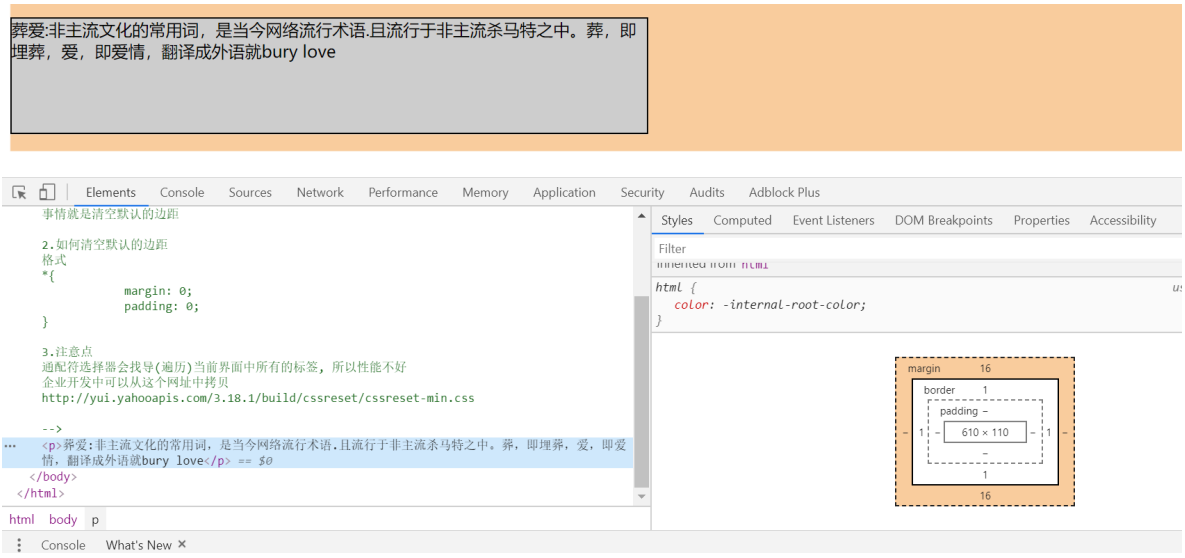
1. CSS3中新增了一个box-sizing属性，这个属性可以保证我们给盒子新增padding和border之后，盒子元素的宽度和高度不变
2. box-sizing属性是如何保证增加padding和border之后，盒子元素的宽度和高度不变?
   和我们前面学习的原理一样，增加padding和border之后要想保证盒子元素的宽高不变，那么就必须减去一部分内容的宽度和高度
3. box-sizing的取值
   3.1 content-box
   > 元素的宽高 = 边框 + 内边距 + 内容宽高
   3.2 border-box
   > 元素的宽高 = 恒等于width/height的宽高

补充

1.text-align:center;和margin:0 auto;区别
text-align: center;作用--设置盒子中存储的文字/图片水平居中margin:0 auto;作用--让盒子自己水平居中
[详见53-盒子居中和内容居中]

浮动

清空默认边距



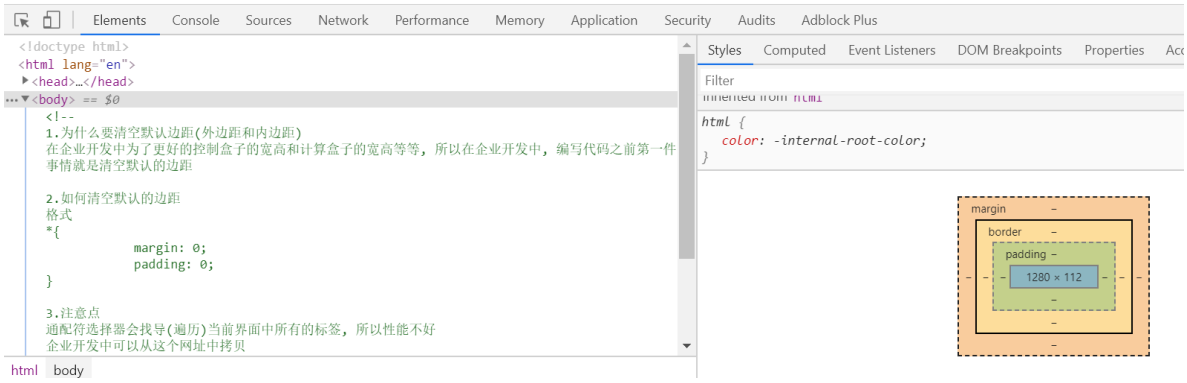
在企业开发中为了更好的控制盒子的宽高和计算盒子的宽高等等，所以在企业开发中，编写代码之前第一件事情就是清空默认的边距

如何清空默认边距
使用通配符选择器
格式

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

清空之后张这样

葬爱:非主流文化的常用词，是当今网络流行术语.且流行于非主流杀马特之中。葬，即埋葬，爱，即爱情，翻译成外语就bury love



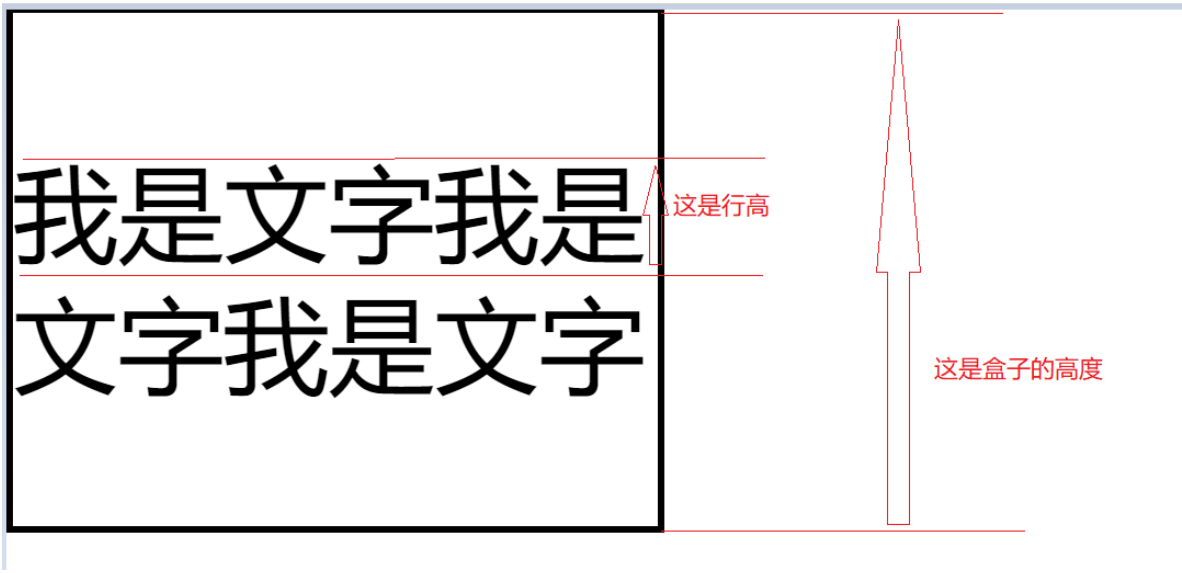
注意点
通配符选择器会找到(遍历)当前界面中所有的标签，所以性能不好企业开发中可以从这个网址中拷贝
http://yui.yahooapis.com/3.18.1/build/cssreset/cssreset-min.css
等价于

```
body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,code,form,fieldset,legend,input
,textarea,p,blockquote,th,td{
       margin:0;padding:0
 }
```

行高和字号

什么是行高?
在CSS中所有的行都有自己的行高
格式：`line-height: 40px;`

注意点:
行高和盒子高不是同一个概念行高指的是每行内容的高度
盒子高指的是元素的高度



规律:
1.文字在行高中默认是垂直居中的
2.在企业开发中我们经常将盒子的高度和行高设置为一样，那么这样就可以保证[　一]行文字在盒子的高度中是垂直居中的
简而言之就是：要想一行文字在盒子中垂直居中，那么只需要设置这行文字的"行高等于盒子的高"即可
3.在企业开发中如果一个盒子中有多行文字，那么我们就不能使用设置行高等于盒子高来实现让文字垂直居中，只能通过设置padding来让文字居中

还原字体和字号

注意点:
1.在企业开发中，如果一个盒子中存储的是文字，那么一般情况下我们会以盒子左边的内边距为基准，不会以右边的内边距为基准，因为这个右边的内边距有误差
2.右边内边距的误差从何而来？因为右边如果放不下一个文字，那么文字就会换行显示，所以文字和内边距之间的距离就有了误差
3.顶部的内边距并不是边框到文字顶部的距离，而是边框到行高顶部的距离

文字界面

建议回顾
61-文字界面　练习一遍

网页的布局方式

什么是网页的布局方式?

网页的布局方式其实就是指浏览器是如何对网页中的元素进行排版的

网页布局分三种

- 标准流（文档流/普通流）
- 浮动流
- 定位流

标准流(文档流/普通流)排版方式

1 其实浏览器默认的排版方式就是标准流的排版方式
2 在CSS中将元素分为三类，分别是块级元素/行内元素/行内块级元素
3 在标准流中有两种排版方式，一种是垂直排版，一种是水平排版垂直排版，如果元素是块级元素，那么就会垂直排版水平排版，如果元素是行内元素/行内块级元素，那么就会水平排版

浮动流排版方式

1 浮动流是一种"半脱离标准流"的排版方式
2 浮动流只有一种排版方式，就是水平排版. 它只能设置某个元素左对齐或者右对齐

注意点:
1.浮动流中没有居中对齐，也就是没有center这个取值
2.在浮动流中是不可以使用margin: 0 auto;

特点:
1.在浮动流中是不区分块级元素/行内元素/行内块级元素的无论是块级元素/行内元素/行内块级元素都可以水平排版
2.在浮动流中无论是块级元素/行内元素/行内块级元素都可以设置宽高
3.综上所述，浮动流中的元素和标准流中的行内块级元素很像

```
    <style>
        .box1{
            width: 100px;
            height: 100px;
            background-color: red;
            /*display: inline-block;*/
            float: left;//和父元素body左对齐
        }
        .box2{
            width: 100px;
            height: 100px;
            background-color: blue;
            /*display: inline-block;*/
            /*margin-left:930px;*/
            float: right;//和父元素body右对齐
        }
     </style>
```

浮动元素的脱标

1.什么是浮动元素的脱标?
脱标:脱离标准流当某一个元素浮动之后，那么这个元素看上去就像被从标准流中删除了一样，这个就是浮动元素的脱标
2.浮动元素脱标之后会有什么影响?
如果前面一个元素浮动了，而后面一个元素没有浮动 ，那么这个时候前面一个元素就会盖住后面一个元素

浮动元素的排序规则

浮动元素排序规则

1 相同方向上的浮动元素，先浮动的元素会显示在前面，后浮动的元素会显示在后面
2 不同方向上的浮动元素，左浮动会找左浮动，右浮动会找右浮动
3 浮动元素浮动之后的位置（第几行），由浮动元素浮动之前在标准流中的位置（第几行）来确定

浮动元素的贴靠现象

1.如果父元素的宽度能够显示所有浮动元素，那么浮动的元素会并排显示2.如果父元素的宽度不能显示所有浮动元素，那么会从最后一个元开始往前贴靠
3.如果贴靠了前面所有浮动元素之后都不能显示，最终会贴靠到父元素的左边或者右边，不管父元素够不够

浮动元素的字围现象

什么是浮动元素的字围现象

浮动元素不会挡住没有浮动元素中的文字，没有浮动的文字会自动给浮动的元素让位置,这个就是浮动元素字围现象

浮动元素的高度问题

1.在标准流中内容的高度可以撑起父元素的高度
2.在浮动流中浮动的元素是不可以撑起父元素的高度的

清楚浮动方式

方式1

1.清除浮动的第一种方式给前面一个父元素设置高度
注意点:
在企业开发中，我们能不写高度就不写高度，所以这种方式用得很少

方式2

给后面的盒子添加clear属性

clear属性取值:
none: 默认取值，按照浮动元素的排序规则来排序(左浮动找左浮动，右浮动找右浮动)
left: 不要找前面的左浮动元素
right: 不要找前面的右浮动元素
both: 不要找前面的左浮动元素和右浮动元素

注意点:
当我们给某个元素添加clear属性之后，那么这个属性的margin属性就会失效

方式3

隔墙法

- 外墙法
- 内墙法

1. 外墙法

```
1 在两个盒子中间添加一个额外的块级元素
2 给这个额外添加的块级元素设置clear: both;属性
```

```
 .wall{
     clear: both;
 }
 ---------------------
<div class="box1">
    <p>我是文字1</p>
    <p>我是文字1</p>
    <p>我是文字1</p>
</div>

<!--<div class="wall"></div>-->

<div class="box2">
    <p>我是文字2</p>
    <p>我是文字2</p>
    <p>我是文字2</p>
</div>
```

```
注意点:
外墙法它可以让第二个盒子使用margin-top属性
外墙法不可以让第一个盒子使用margin-bottom属性
```

2. 内墙法

```
1 在第一个盒子中所有子元素的最后添加一个额外的块级元素
2 给这个额外添加的块级元素设置clear: both;属性
```

```
注意点:
内墙法它可以让第二个盒子使用margin-top属性
内墙法它可以让第一个盒子使用margin-bottom属性
```

外墙法和内墙法有什么区别?

```
外墙法不能撑起第一个盒子的高度，而内墙法可以撑起第一个盒子的高度
```

> 在企业开发中不常用隔墙法来清楚浮动

什么是伪元素选择器

```
伪元素选择器作用就是给指定标签的内容前面添加一个子元素或者给指定标签的内容后面添加一个子元素
```

```
格式:
标签名称::before{
    属性名称:值;
}
给指定标签的内容前面添加一个子元素

标签名称::after{
    属性名称:值;
}
给指定标签的内容后面添加一个子元素
```

```
<style>
       ...
        div::before{
            content: "爱你";
            width: 50px;
            height: 50px;
            background-color: pink;
            display: block; //默认行级元素
        }
        div::after{
            /*指定添加的子元素中存储的内容*/
            content: "么么哒";
            /*指定添加的子元素的宽度和高度*/
            width: 50px;
            /*height: 50px;*/
            /*内容是可以超出标签的范围的，所以高度为0依然可以看见内容*/
            height:0;
            background-color: pink;
            /*指定添加的子元素的显示模式*/
            display: block;
            /*隐藏添加的子元素*/
            visibility: hidden;
        }

</style>
<body>
    <div>
        <!--<p>爱你</p>-->
        我是文字
        <!--<p>么么哒</p>-->
    </div>
</body>
```

清除浮动的方式4

利用伪元素选择器清除浮动本质上就是内墙法，只不过是直接通过CSS代码添加了内墙，其它特性和内墙法都一样

```
<style>
        *{
            margin: 0;
            padding: 0;
        }
        .box1{
```

```
            background-color: red;
            /*margin-bottom: 10px;*/
        }
        .box2{
            background-color: green;
            /*margin-top: 10px;*/
        }
        .box1 p{
            width: 100px;
            background-color: blue;
        }
        .box2 p{
            width: 100px;
            background-color: yellow;
        }
        p{
            float: left;
        }
        .box1::after{
            /*设置添加的子元素的内容为空*/
            content: "";
            /*设置添加的子元素为块级元素*/
            display: block;
            /*设置添加的子元素的高度为0*/
            height: 0;
            /*设置添加的子元素看不见*/
            visibility: hidden;
            /*给添加的子元素设置clear: both;*/
            clear: both;
        }
        .box1{
            /*兼容IE6*/
            *zoom:1;
        }
</style>

<body>
    <div class="box1">
        <p>我是文字1</p>
        <p>我是文字1</p>
        <p>我是文字1</p>

    </div>

    <div class="box2">
        <p>我是文字2</p>
        <p>我是文字2</p>
        <p>我是文字2</p>
    </div>
</body>
```

注意点:
IE6中不支持这种方式, 为了兼容IE6必须给前面的盒子添加*zoom:1;属性

方式5

1.overflow: hidden;作用
1.1可以将超出标签范围的内容裁剪掉
1.2清除浮动
1.3可以通过overflow: hidden;让里面的盒子设置margin-top之后，外面的盒子不被顶下来（之前的解决办法是给外面的盒子设置一个border属性）

```
.box1{
    width: 200px;
    height: 200px;
    background-color: red;
    /*border: 1px solid #000;*/
    overflow: hidden;
}
.box2{
    width: 100px;
    height: 100px;
    background-color: blue;
    margin-top: 20px;
}
```

定位流

定位流的分类

1.1 相对定位

1.2 绝对定位

1.3 固定定位

1.4 静态定位

什么是相对定位

> 相对于自己以前在标准流中的位置来移动

格式

```
position: relative;
top: 20px;
left: 20px;
/*right: 20px;*/
/*margin-bottom: 20px;*/
margin-top: 20px;
```

相对定位的注意点

```
1、相对定位是不脱离标准流的，会继续在标准流中占用一份空间
2、在相对定位中同一个方向上的定位属性只能使用一个，即若果使用了top就不要使用bottom,使用了left就不要使用right
3、由于相对定位是不脱离标准流的，所以在相对定位中是区分块级元素/行内元素/行内块级元素
4、由于相对定位是不脱离标准流的，并且相对定位的元素会占用标准流中的位置，所以当给相对定位的元素设置margin/padding等属性的时会影响到标准流的布局,(本质)效果相当于给[定位之前]元素设置margin/padding
```

什么是绝对定位

> 绝对定位就是相对于body来定位

绝对定位的注意点

```
1、绝对定位的元素是脱离标准流的
2、绝对定位的元素是不区分块级元素/行内元素/行内块级元素
```

格式

```
position: absolute;
```

绝对定位的-参考点

```
1.默认情况下所有的绝对定位的元素，无论有没有祖先元素，都会以body作为参考点
2.如果一个绝对定位的元素有祖先元素，并且祖先元素也是定位流，那么这个绝对定位的元素就会以定位流的那个祖先元素作为参考点
    2.1只要是这个绝对定位元素的祖先元素都可以
    2.2定位流是指绝对定位/相对定位/固定定位
    2.3定位流中只有静态定位不行
3.如果一个绝对定位的元素有祖先元素，并且祖先元素也是定位流，而且祖先元素中有多个元素都是定位流，那么这个绝对定位的元素会以离它最近的那个定位流的祖先元素为参考点(就近原则)
```

绝对定位的-注意点

```
1.如果一个绝对定位的元素是以body作为参考点，那么其实是以网页首屏的宽度和高度作为参考点，而不是以整个网页的宽度和高度作为参考点
2.一个绝对定位的元素会忽略祖先元素的padding
```

定位流的使用

> 子绝父相
>
> 子元素使用绝对定位，父元素使用相对定位

```
相对定位弊端:
    相对定位不会脱离标准流，会继续在标准流中占用一份空间，所以不利于布局界面
绝对定位弊端:
    默认情况下绝对定位的元素会以body作为参考点，所以会随着浏览器的宽度高度的变化而变化
子绝父相:
    子元素用绝对定位，父元素用相对定位
```

绝对定位的水平居中

如何让绝对定位的元素水平居中？
只需要设置绝对定位元素的left:50%;
然后再设置绝对定位元素的 margin-left: -元素宽度的一半px;

什么是固定定位

> 固定定位和前面学习的背景关联方式很像，背景定位可以让背景图片不随着滚动条的滚动而滚动，而固定定位可以让某个盒子不随着滚动条的滚动而滚动

固定定位的注意点

1.固定定位的元素是脱离标准流的，不会占用标准流中的空间
2.固定定位和绝对定位一样不区分行内/块级/行内块级

定位流的z-index属性

> 默认情况下所有的元素都有一个默认的z-index属性, 取值是0.

作用

z-index属性的作用是专门用于控制定位流元素的覆盖关系的

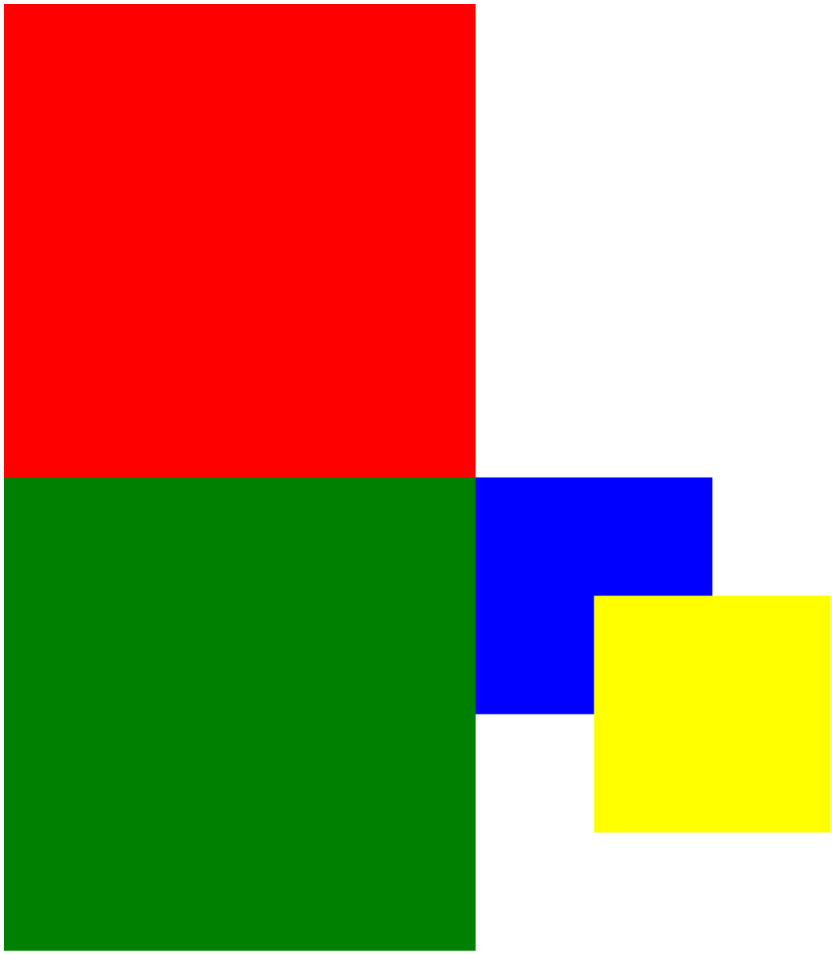
现象

1.默认情况下定位流的元素会盖住标准流的元素
2.默认情况下定位流的元素后面编写的会盖住前面编写的
3.如果定位流的元素设置了z-index属性，那么谁的z-index属性比较大，谁就会显示在上面

注意点

> 从父现象

1、如果两个元素的父元素都没有设置z-index属性，那么谁的z-index属性比较大谁就显示在上面
2、如果两个元素的父元素设置了z-index属性，那么子元素的z-index属性就会失效，也就是说谁的父元素的z-index属性比较大谁就会显示在上面



过渡模块

a标签的伪类选择器

通过我们的观察发现a标签存在一定的状态

```
1、默认状态，从未被访问过
2、被访问过的状态
3、鼠标长按状态
4、鼠标悬停在a标签上状态
```

什么是a标签的伪类选择器

```
a标签的伪类选择器是专门用来修改a标签不同状态的样式的
```

格式：

```
        a:link{
            color: tomato;
        }

       ...
       <a href="http://www.taobao.com">taobao</a>
       <a href="http://www.jd.com">jd</a>
```

```
:link 修改从未被访问过状态下的样式
:visited 修改被访问过的状态下的样式
:hover 修改鼠标悬停在a标签上状态下的样式
:active 修改鼠标长按状态下的样式
```

注意点

```
1、a标签的伪类选择器可以单独出现也可以一起出现
2、a标签的伪类选择器如果一起出现，那么有严格的顺序要求
编写的顺序必须要个的遵守爱恨原则 love hate
3、如果默认状态的样式和被访问过状态的样式一样，那么可以缩写
```

练习：87-a标签导航条

过度模块基本使用

> x： hover这个伪类选择器除了可以用在a标签上，还可以用在其他任何选择器上

例如：将鼠标悬停在一个div上，这个div的宽度由50px变为300px

```
        *{
            margin: 0;
            padding: 0;
         }
         div{
            width: 100px;
            height: 50px;
            background-color: red;
            /*告诉系统哪个属性需要执行过渡效果*/
            transition-property: width, background-color;
            /*告诉系统过渡效果持续的时长*/
            transition-duration: 5s, 5s;

            /*transition-property: background-color;*/
            /*transition-duration: 5s;错误方式*/
         }
         /*:hover这个伪类选择器除了可以用在a标签上，还可以用在其它的任何标签上*/
         div:hover{
            width: 300px;
            background-color: blue;
         }
```

过度三要素

```
1、必须要有属性发生变化
2、必须告诉系统哪个属性需要执行过渡效果
3、必须告诉系统过渡效果持续时长
```

注意点

```
当多个属性需要同时执行过渡效果时用逗号隔开即可
transition-property: width, background-color;
transition-duration: 5s, 5s;
```

过度模块其他属性

transition-delay	**规定过渡效果何时开始。默认是 0。**
transition-timing-function	规定过渡效果的时间曲线。默认是 "ease"。

```
<style>
    *{
        margin: 0;
        padding: 0;
    }
    div {
        width: 100px;
        height: 50px;
        background-color: red;
        transition-property: width;
        transition-duration: 5s;
        /*告诉系统延迟多少秒之后才开始过渡动画*/
        /*transition-delay: 2s;*/
    }
    div:hover{
        width: 300px;
    }
    ul{
        width: 800px;
        height: 500px;
        margin: 0 auto;
        background-color: pink;
        border: 1px solid #000;
    }
    ul li{
        list-style: none;
        width: 100px;
        height: 50px;
        margin-top: 50px;
        background-color: blue;
        transition-property: margin-left;
        transition-duration: 10s;
    }
    ul:hover li{
        margin-left: 700px;
    }
    ul li:nth-child(1){
        /*告诉系统过渡动画的运动的速度*/
        transition-timing-function: linear;
    }
    ul li:nth-child(2){
        transition-timing-function: ease;
    }
    ul li:nth-child(3){
        transition-timing-function: ease-in;
    }
    ul li:nth-child(4){
        transition-timing-function: ease-out;
    }
    ul li:nth-child(5){
        transition-timing-function: ease-in-out;
    }
</style>
```

```
    ...
    <ul>
        <li>linear</li>
        <li>ease</li>
        <li>ease-in</li>
        <li>ease-out</li>
        <li>ease-in-out</li>
    </ul>
```

连写格式:

```
transition: 过渡属性 过渡时长 运动速度 延迟时间;
```

```
过渡连写注意点
1、和分开写一样，如果想给多个属性添加过渡效果也是用逗号隔开即可
    transition: width 5s linear 0s,background-color 5s linear 0s;
2、连写的时可以省略后面的两个参数，因为只要编写了前面的两个参数就已经满足了过渡的三要素
    transition: width 5s,background-color 5s,height 5s;
3、如果多个属性运动的速度/延迟的时间/持续时间都一样，那么可以简写为
transition:all 0s;
```

过渡的编写流程:

```
1.编写过渡套路
    1.1不要管过渡，先编写基本界面
    1.2修改我们认为需要修改的属性
    1.3再回过头去给被修改属性的那个元素添加过渡即可
```

2D转换模块

transform

```
<head>
    <meta charset="UTF-8">
    <title>93-2D转换模块</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            width: 800px;
            height: 500px;
            border: 1px solid #000;
            margin: 0 auto;
        }
        ul li{
            list-style: none;
            width: 100px;
            height: 50px;
            background-color: red;
            margin: 0 auto;
            margin-top: 50px;
            text-align: center;
            line-height: 50px;
```

```
        }
        ul li:nth-child(2){
            /*其中deg是单位，代表多少度*/
            transform: rotate(45deg);
        }
        ul li:nth-child(3){
            /*
            第一个参数:水平方向
            第二个参数:垂直方向
            */
            transform: translate(100px, 0px);
        }
        ul li:nth-child(4){
            /*
            第一个参数:水平方向
            第二个参数:垂直方向
            注意点:
            如果取值是1，代表不变
            如果取值大于1，代表需要放大
            如果取值小于1，代表需要缩小
            如果水平和垂直缩放都一样，那么可以简写为一个参数
            */
            /*transform: scale(0.5, 0.5);*/
            transform: scale(1.5);
        }
        ul li:nth-child(5){
            /*
            注意点:
            1.如果需要进行多个转换，那么用空格隔开
            2.2D的转换模块会修改元素的坐标系，所以旋转之后再平移就不是水平平移的
            */
            transform: rotate(45deg) translate(100px, 0px) scale(1.5, 1.5);
            /*transform: translate(100px, 0px);*/
        }
    </style>
</head>
<body>
<ul>
    <li>正常的</li>
    <li>旋转的</li>
    <li>平移的</li>
    <li>缩放的</li>
    <li>综合的</li>
</ul>
</body>
```

transform-origin 形变中心

```
 <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            width: 200px;
            height: 200px;
            border: 1px solid #000;
```

```
            margin: 100px auto;
            position: relative;
        }
        ul li{
            list-style: none;
            width: 200px;
            height: 200px;
            position: absolute;
            left: 0;
            top: 0;
            /*
            第一个参数:水平方向
            第二个参数:垂直方向
            注意点
            取值有三种形式
            具体像素
            百分比
            特殊关键字
            */
            /*transform-origin: 200px 0px;*/
            /*transform-origin: 50% 50%;*/
            /*transform-origin: 0% 0%;*/
            /*transform-origin: center center;*/
            transform-origin: left top;
        }
        ul li:nth-child(1){
            /*
            默认情况下所有的元素都是以自己的中心点作为参考来旋转的，我们可以通过形变中心点属性来修改它的参考点
            */
            background-color: red;
            transform: rotate(30deg);
        }
        ul li:nth-child(2){
            background-color: green;
            transform: rotate(50deg);
        }
        ul li:nth-child(3){
            background-color: blue;
            transform: rotate(70deg);
        }
    </style>
</head>
<body>
<ul>
    <li></li>
    <li></li>
    <li></li>
</ul>
</body>
```

旋转轴向

```
 <style>
        *{
            margin: 0;
            padding: 0;
```

```
        }
        ul{
            width: 800px;
            height: 500px;
            margin: 0 auto;
        }
        ul li{
            list-style: none;
            width: 200px;
            height: 200px;
            margin: 0 auto;
            margin-top: 50px;
            border: 1px solid #000;
            /*
            1.什么是透视
            近大远小
            2.注意点
            一定要注意，透视属性必须添加到需要呈现近大远小效果的元素的父元素上面
            */
            perspective: 500px;
        }
        ul li img{
            width: 200px;
            height: 200px;
            /*perspective: 500px;*/
        }
        ul li:nth-child(1){
            /*默认情况下所有元素都是围绕Z轴进行旋转*/
            transform: rotateZ(45deg);
        }
        ul li:nth-child(2) img{
            transform: rotateX(45deg);
        }
        ul li:nth-child(3) img{
            /*
            总结:
            想围绕哪个轴旋转，那么只需要在rotate后面加上哪个轴即可
            */
            transform: rotateY(45deg);
        }
    </style>
    ...
    <ul>
        <li><img src="images/rotateZ.jpg" alt=""></li>
        <li><img src="images/rotateX.jpg" alt=""></li>
        <li><img src="images/rotateY.jpg" alt=""></li>
    </ul>
```

给盒子和文字添加阴影

```
1.如何给盒子添加阴影
box-shadow: 水平偏移 垂直偏移 模糊度 阴影扩展 阴影颜色 内外阴影;

2.注意点
2.1盒子的阴影分为内外阴影，默认情况下就是外阴影
2.2快速添加阴影只需要编写三个参数即可
box-shadow: 水平偏移 垂直偏移 模糊度;
默认情况下阴影的颜色和盒子内容的颜色一致

3.如何给文字添加阴影
text-shadow: 水平偏移 垂直偏移 模糊度 阴影颜色 ;
```

动画模块

过度和动画之间的异同

不同点:

```
过渡必须人为的触发才会执行动画
动画不需要人为的触发就可以执行动画
```

相同点:

```
过渡和动画都是用来给元素添加动画的
过渡和动画都是系统新增的一些属性
过渡和动画都需要满足三要素才会有动画效果
```

```
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div{
            width: 100px;
            height: 50px;
            background-color: red;
            /*transition-property: margin-left;*/
            /*transition-duration: 3s;*/

            /*1.告诉系统需要执行哪个动画*/
            animation-name: lnj;
            /*3.告诉系统动画持续的时长*/
            animation-duration: 3s;
        }
        /*2.告诉系统我们需要自己创建一个名称叫做lnj的动画*/
        @keyframes lnj {
            from{
                margin-left: 0;
            }
            to{
                margin-left: 500px;
            }
        }

        /*div:hover{*/
```

```
            /*margin-left: 500px;*/
        /*}*/
    </style>
```

动画属性上

```
<head>
    <meta charset="UTF-8">
    <title>101-动画模块-其它属性上</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div {
            width: 100px;
            height: 50px;
            background-color: red;
            animation-name: sport;
            animation-duration: 2s;
            /*告诉系统多少秒之后开始执行动画*/
            /*animation-delay: 2s;*/
            /*告诉系统动画执行的速度*/
            animation-timing-function: linear;
            /*告诉系统动画需要执行几次*/
            animation-iteration-count: 3;
            /*告诉系统是否需要执行往返动画
            取值:
            normal, 默认的取值, 执行完一次之后回到起点继续执行下一次
            alternate, 往返动画, 执行完一次之后往回执行下一次
            */
            animation-direction: alternate;
        }
        @keyframes sport {
            from{
                margin-left: 0;
            }
            to{
                margin-left: 500px;
            }
        }
        div:hover{
            /*
            告诉系统当前动画是否需要暂停
            取值:
            running: 执行动画
            paused: 暂停动画
            */
            animation-play-state: paused;
        }
    </style>
</head>
<body>
<div class="box1"></div>
</body>
```

动画属性下

动画属性的另一种定义模式

```
<head>
    <meta charset="UTF-8">
    <title>102-动画模块-其它属性下</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        .box1 {
            width: 100px;
            height: 50px;
            background-color: red;
            position: absolute;
            left: 0;
            top: 0;
            animation-name: sport;
            animation-duration: 5s;
        }
        @keyframes sport {
            0%{
                left: 0;
                top: 0;
            }
            25%{
                left: 300px;
                top: 0;
            }
            50%{
                left: 300px;
                top: 300px;
            }
            75%{
                left: 0;
                top: 300px;
            }
            100%{
                left: 0;
                top: 0;
            }
        }

        .box2{
            width: 200px;
            height: 200px;
            background-color: blue;
            margin: 100px auto;
            animation-name: myRotate;
            animation-duration: 5s;
            animation-delay: 2s;
            /*
            通过我们的观察, 动画是有一定的状态的
            1.等待状态
            2.执行状态
            3.结束状态
            */
```

```
            /*
            animation-fill-mode作用:
            指定动画等待状态和结束状态的样式
            取值:
            none: 不做任何改变
            forwards: 让元素结束状态保持动画最后一帧的样式
            backwards: 让元素等待状态的时候显示动画第一帧的样式
            both: 让元素等待状态显示动画第一帧的样式, 让元素结束状态保持动画最后一帧的样式
            */
            /*animation-fill-mode: backwards;*/
            /*animation-fill-mode: forwards;*/
            animation-fill-mode: both;
        }
        @keyframes myRotate {
            0%{
                transform: rotate(10deg);
            }
            50%{
                transform: rotate(50deg);
            }
            100%{
                transform: rotate(70deg);
            }
        }
    </style>
</head>
<body>
<div class="box1"></div>
<div class="box2"></div>
</body>
```

动画的连写

```
<head>
    <meta charset="UTF-8">
    <title>103-动画模块-连写</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        div {
            width: 100px;
            height: 50px;
            background-color: red;
            /*animation: move 3s linear 2s 1 normal;*/
            animation: move 3s;
        }
        @keyframes move {
            from{
                margin-left: 0;
            }
            to{
                margin-left: 500px;
            }
        }
    </style>
```

```
</head>
<body>
<!--
1.动画模块连写格式
animation:动画名称 动画时长 动画运动速度 延迟时间 执行次数 往返动画;

2.动画模块连写格式的简写
animation:动画名称 动画时长;
-->
<div></div>
</body>
```

3D模块

什么是3D和2D

```
  1.什么是2D和3D
   2D就是一个平面，只有宽度和高度，没有厚度
   3D就是一个立体，有宽度和高度，还有厚度
   默认情况下所有的元素都是呈2D展现的
  2.如何让某个元素呈3D展现
   和透视一样，想看到某个元素的3d效果，只需要给他的父元素添加一个transform-style属性，然后设置为preserve-3d即可
```

正方体

```
<head>
    <meta charset="UTF-8">
    <title>107-3D转换模块-正方体</title>
    <style>
    *{
        margin: 0;
        padding: 0;
    }
    ul{
        width: 200px;
        height: 200px;
        border: 1px solid #000;
        box-sizing: border-box;
        margin: 100px auto;
        position: relative;
        transform: rotateY(0deg) rotateX(0deg);
        transform-style: preserve-3d;
    }
    ul li{
        list-style: none;
        width: 200px;
        height: 200px;
        font-size: 60px;
        text-align: center;
        line-height: 200px;
        position: absolute;
        left: 0;
        top: 0;
    }
    ul li:nth-child(1){
        background-color: red;
```

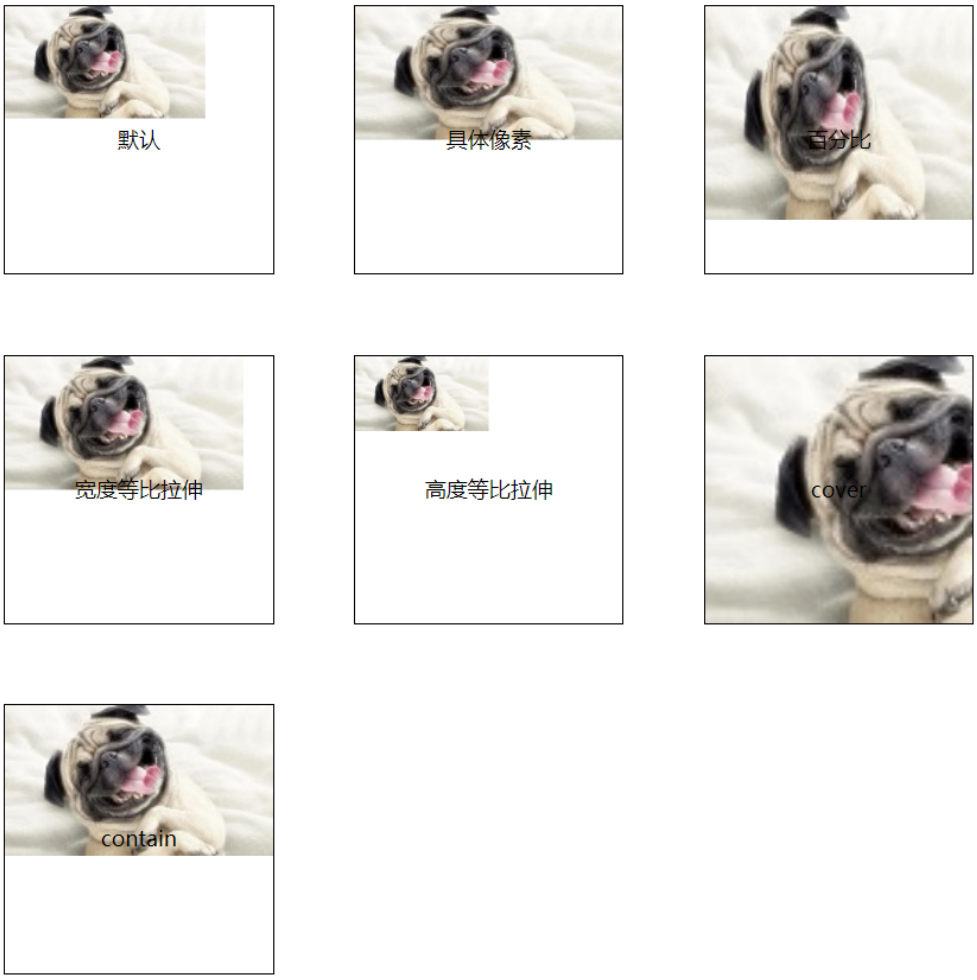
```
        transform: translateX(-100px) rotateY(90deg);
    }
    ul li:nth-child(2){
        background-color: green;
        transform: translateX(100px) rotateY(90deg);
    }
    ul li:nth-child(3){
        background-color: blue;
        transform: translateY(-100px) rotateX(90deg);
    }
    ul li:nth-child(4){
        background-color: yellow;
        transform: translateY(100px) rotateX(90deg);
    }
    ul li:nth-child(5){
        background-color: purple;
        transform: translateZ(-100px);
    }
    ul li:nth-child(6){
        background-color: pink;
        transform: translateZ(100px);
    }

</style>
</head>
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
    <li>6</li>
</ul>
</body>
```

```
<head>
    <meta charset="UTF-8">
    <title>108-3D转换模块-正方体终极</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            width: 200px;
            height: 200px;
            border: 1px solid #000;
            box-sizing: border-box;
            margin: 100px auto;
            position: relative;
            transform: rotateY(0deg) rotateX(0deg);
            transform-style: preserve-3d;
        }
        ul li{
            list-style: none;
            width: 200px;
```

```
            height: 200px;
            font-size: 60px;
            text-align: center;
            line-height: 200px;
            position: absolute;
            left: 0;
            top: 0;
        }
        ul li:nth-child(1){
            background-color: red;
            transform: rotateX(90deg) translateZ(100px);
        }
        ul li:nth-child(2){
            background-color: green;
            transform: rotateX(180deg) translateZ(100px);
        }
        ul li:nth-child(3){
            background-color: blue;
            transform: rotateX(270deg) translateZ(100px);
        }
        ul li:nth-child(4){
            background-color: yellow;
            transform: rotateX(360deg) translateZ(100px);
        }
        ul li:nth-child(5){
            background-color: purple;
            transform: translateX(-100px) rotateY(90deg);
        }
        ul li:nth-child(6){
            background-color: pink;
            transform: translateX(100px) rotateY(90deg);
        }

    </style>
</head>
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
    <li>6</li>
</ul>
</body>
```

背景相关

background-size属性

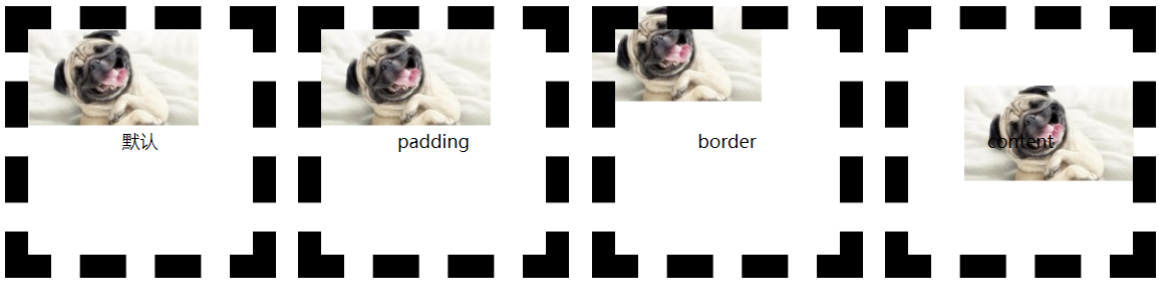



```
<head>
    <meta charset="UTF-8">
    <title>112-背景尺寸属性</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            width: 800px;
            height: 500px;
            margin: 0 auto;
        }
        ul li{
            list-style: none;
            float: left;
            width: 200px;
            height: 200px;
            margin: 30px 30px;
            border: 1px solid #000;
            text-align: center;
            line-height: 200px;
        }
        ul li:nth-child(1){
            background: url("images/dog.jpg") no-repeat;
        }
        ul li:nth-child(2){
            background: url("images/dog.jpg") no-repeat;
            /*
```

```
            第一个参数:宽度
            第二个参数:高度
            */
            background-size:200px 100px;
        }
        ul li:nth-child(3){
            background: url("images/dog.jpg") no-repeat;
            background-size:100% 80%;
        }
        ul li:nth-child(4){
            background: url("images/dog.jpg") no-repeat;
            background-size:auto 100px;
        }
        ul li:nth-child(5){
            background: url("images/dog.jpg") no-repeat;
            background-size:100px auto;
        }
        ul li:nth-child(6){
            background: url("images/dog.jpg") no-repeat;
            /*
            cover含义:
            1.告诉系统图片需要等比拉伸
            2.告诉系统图片需要拉伸到宽度和高度都填满元素
            */
            background-size:cover;
        }
        ul li:nth-child(7){
            background: url("images/dog.jpg") no-repeat;
            /*
            cover含义:
            1.告诉系统图片需要等比拉伸
            2.告诉系统图片需要拉伸到宽度或高度有一个填满元素
            */
            background-size:contain;
        }
    </style>
</head>
<body>
<!--
1.什么是背景尺寸属性
背景尺寸属性是CSS3中新增的一个属性, 专门用于设置背景图片大小
-->
<ul>
    <li>默认</li>
    <li>具体像素</li>
    <li>百分比</li>
    <li>宽度等比拉伸</li>
    <li>高度等比拉伸</li>
    <li>cover</li>
    <li>contain</li>
</ul>
</body>
```

背景图片定位区域属性

background-origin属性




```
<head>
    <meta charset="UTF-8">
    <title>113-背景图片定位区域属性</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            margin-top:100px;
        }
        ul li{
            list-style: none;
            float: left;
            width: 100px;
            height: 100px;
            text-align: center;
            line-height: 100px;
            border: 20px dashed #000;
            padding: 50px;
            margin-left: 20px;
            background: url("images/dog.jpg") no-repeat;
        }
        ul li:nth-child(2){
            /*
            告诉系统背景图片从什么区域开始显示，
            默认情况下就是从padding区域开始显示
            */
            background-origin: padding-box;
        }
        ul li:nth-child(3){
            background-origin:border-box;
        }
        ul li:nth-child(4){
            background-origin:content-box;
        }
    </style>
</head>
<body>
<ul>
    <li>默认</li>
    <li>padding</li>
    <li>border</li>
    <li>content</li>
</ul>
</body>
```

背景绘制区域属性

backgroung-clip属性

```
<head>
    <meta charset="UTF-8">
    <title>114-背景绘制区域属性</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul li{
            list-style: none;
            float: left;
            width: 100px;
            height: 100px;
            text-align: center;
            line-height: 100px;
            border: 20px dashed #000;
            padding: 50px;
            margin-left: 20px;
            background: red url("images/dog.jpg") no-repeat;
        }
        ul li:nth-child(2){
            /*
            背景绘制区域属性是专门用于指定从哪个区域开始绘制背景的，默认情况下会从border区域
开始绘制背景
            */
            background-clip: padding-box;
        }
        ul li:nth-child(3){
            background-clip: border-box;
        }
        ul li:nth-child(4){
            background-clip: content-box;
        }
    </style>
</head>
<body>
<ul>
    <li>默认</li>
    <li>padding</li>
    <li>border</li>
    <li>content</li>
</ul>
</body>
```

多重背景图片

```
<head>
    <meta charset="UTF-8">
    <title>115-多重背景图片</title>
    <style>
        *{
            margin: 0;
            padding: 0;
```

```
        }
        div{
            width: 500px;
            height: 500px;
            border: 1px solid #000;
            margin: 0 auto;
            /*
            多张背景图片之间用逗号隔开即可
            注意点:
            先添加的背景图片会盖住后添加的背景图片
            建议在编写多重背景时拆开编写
            */
            /*background: url("images/animal1.png") no-repeat left
top,url("images/animal2.png") no-repeat right top,url("images/animal3.png") no-
repeat left bottom,url("images/animal4.png") no-repeat right
bottom,url("images/animal5.png") no-repeat center center;*/
            background-image:
url("images/animal1.png"),url("images/animal2.png"),url("images/animal3.png");
            background-repeat: no-repeat, no-repeat, no-repeat;
            background-position: left top, right top, left bottom;
        }
    </style>
</head>
<body>
<div></div>
</body>
```

书写CSS代码的格式

1. 行内样式

   > 可以直接将CSS代码书写到开始标签当中

   `<div style = "color:red">我是DIV</div>`

2. 内嵌样式

```
```
<style>
    div{
        color:blue;
       }
</style>
```
```

3. 外链样式

   通过引入css文件的方式,再head标签

   ```
   <link rel="stylesheet" href="117_书写CSS代码的格式.CSS">
   ```

4. 导入样式

   ```
       <style>
           @import "117_CSS书写格式.CSS";
       </style>
   ```

> 注意：
>
> 外链样式：在加载界面的时候，实现加载CSS样式，再加载HTML结构，所以用户看到界面时已经加载了样式
>
> 导入样式：是先加载HTML结构，再加载CSS样式，所以y用户看到界面时不一定已经加载了样式
>
> 导入样式时CSS2.1推出的，会有兼容问题

编写网页的准备工作

```
1.编写网页第一步：
新建站点文件夹，里面应该包含css文件夹/js文件夹/images文件夹/index.html文件
注意点：
站点文件夹可以是中文，但是里面的子文件夹或者子文件不能出现中文

2.编写网页第二部：
新建css文件，在这个文件夹中做一些初始化操作
清空默认的样式
设置全局的样式
将新建的css文件和当前的index.html文件关联起来
```

边框圆角

什么是边框圆角?

> 将直角边框变为圆角

格式:

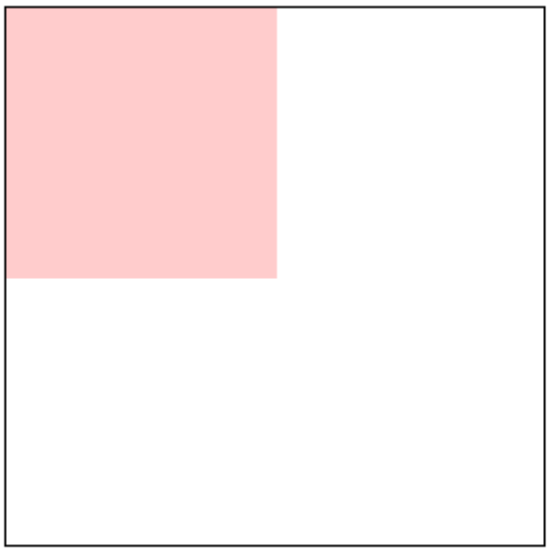
```
border-radius: 左上 右上 右下 左下;
```

将正方形变为圆形的技巧?

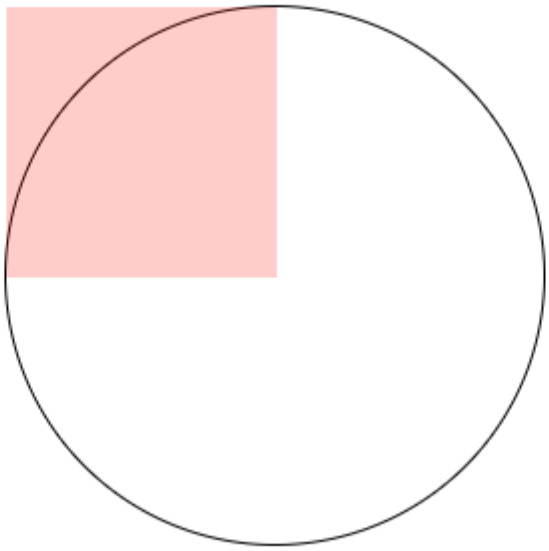
```
border-radius: 50%;
```

系统是如何绘制圆形的?

```
首先根据指定的值找到圆心，按照指定的值作为半径绘制圆弧
```



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



边框圆角的注意点

1. 当省略了某一个角的值之后，系统会自动参考对角的值
   `border-radius: 100px 50px;`

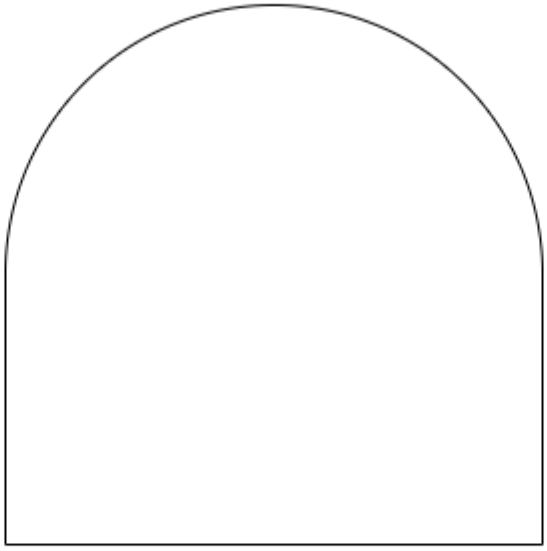
2. 当边框圆角的值 > 边框宽度的时候，外边框和内边框都会变成圆角
   当边框圆角的值 <= 边框宽度的时候，外边框是圆角，内边框是直角

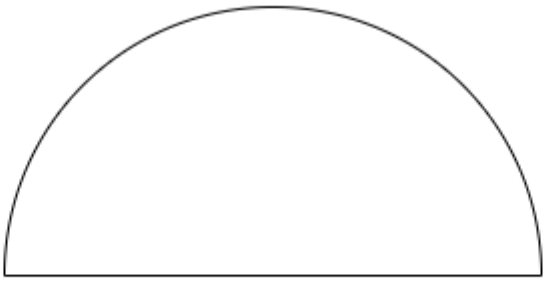
绘制半圆和椭圆

> 可以通过border-xxx-xx-radius的方式单独设置某一个角的值。
> border-xxx-xx-radius接收两个参数，第一个表示水平方向，第二个表示垂直方向
> border-xxx-xx-radius如果只传递了一个参数，那么垂直方向和水平方向的值一样

比如一个正方形，绘制半圆，将左上角和右上角变为圆弧，再将高度变为一半

```
width: 200px;
height: 100px;
border: 1px solid #000;
...
border-top-left-radius: 100px;
border-top-right-radius: 100px;
```





绘制椭圆

绘制椭圆设置水平方向为宽度的一半，设置垂直方向为高度的一半

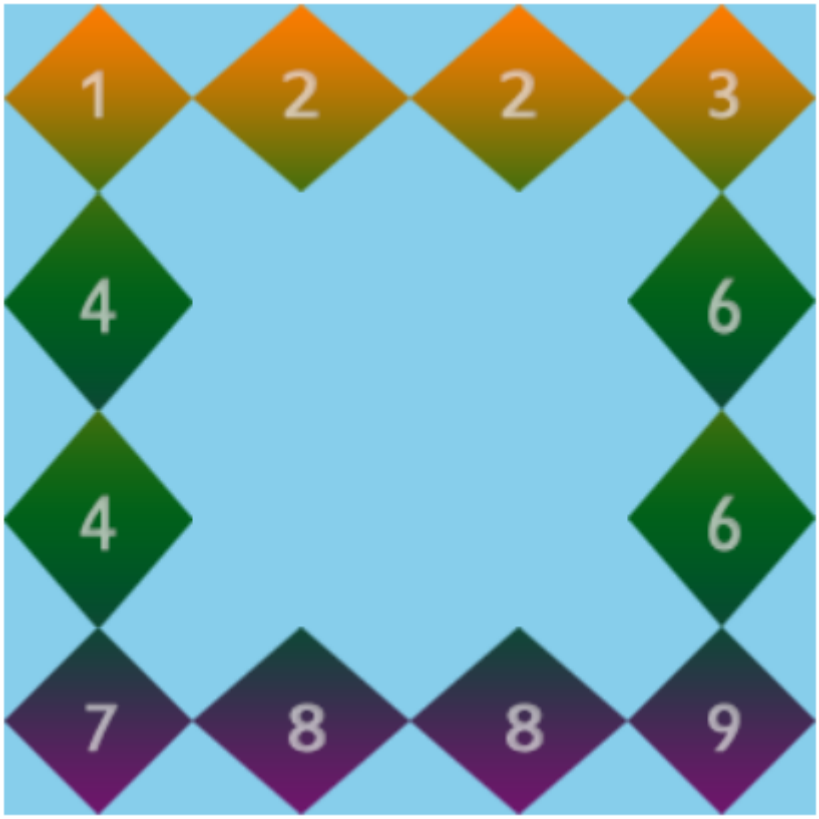
```
width: 400px;
height: 200px;
border: 1px solid #000;
box-sizing: border-box;
margin: 300px auto;
/*绘制椭圆设置水平方向为宽度的一半，设置垂直方向为高度的一半*/
border-top-left-radius: 200px 100px;
border-top-right-radius: 200px 100px;
border-bottom-left-radius: 200px 100px;
border-bottom-right-radius: 200px 100px;
```

边框图片

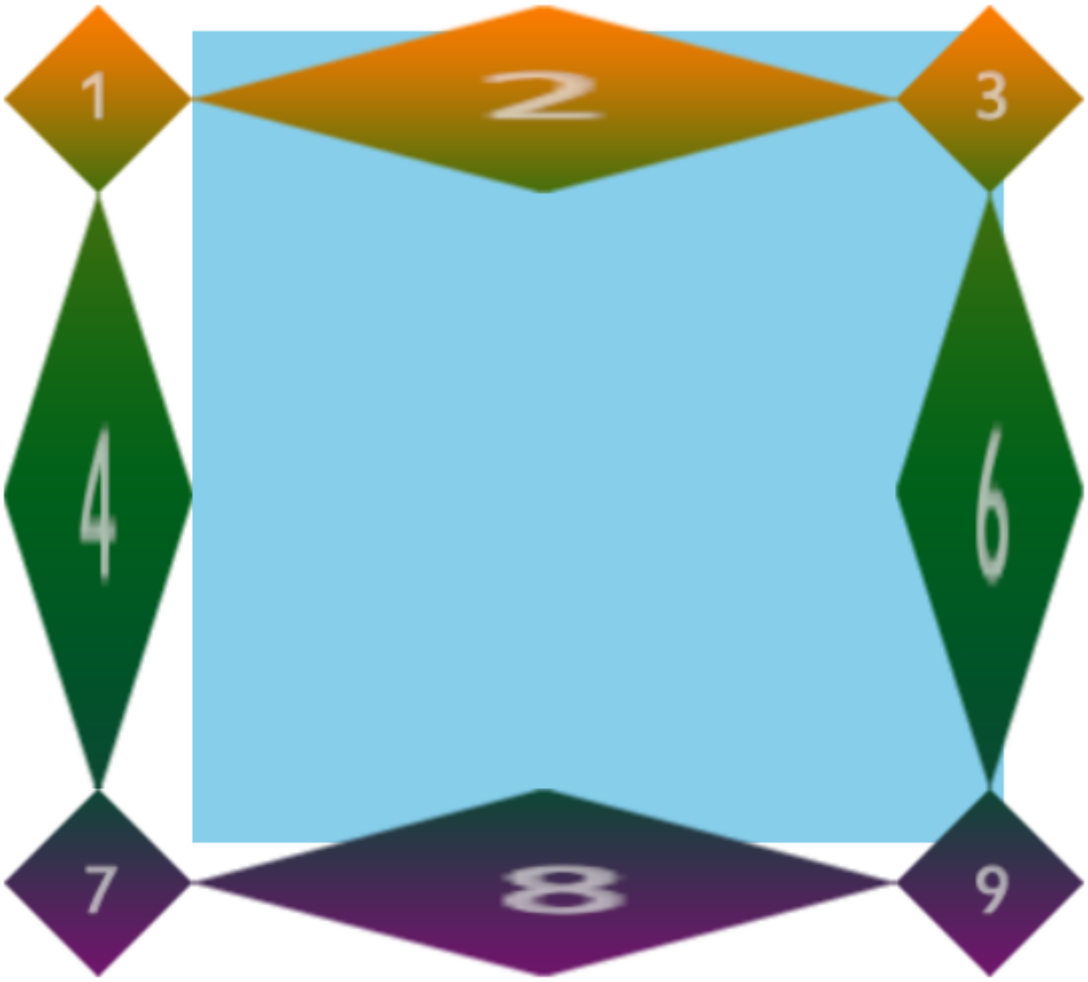
原图


```
*{
     margin: 0;
     padding: 0;
 }
div{
     width: 300px;
     height: 300px;
```

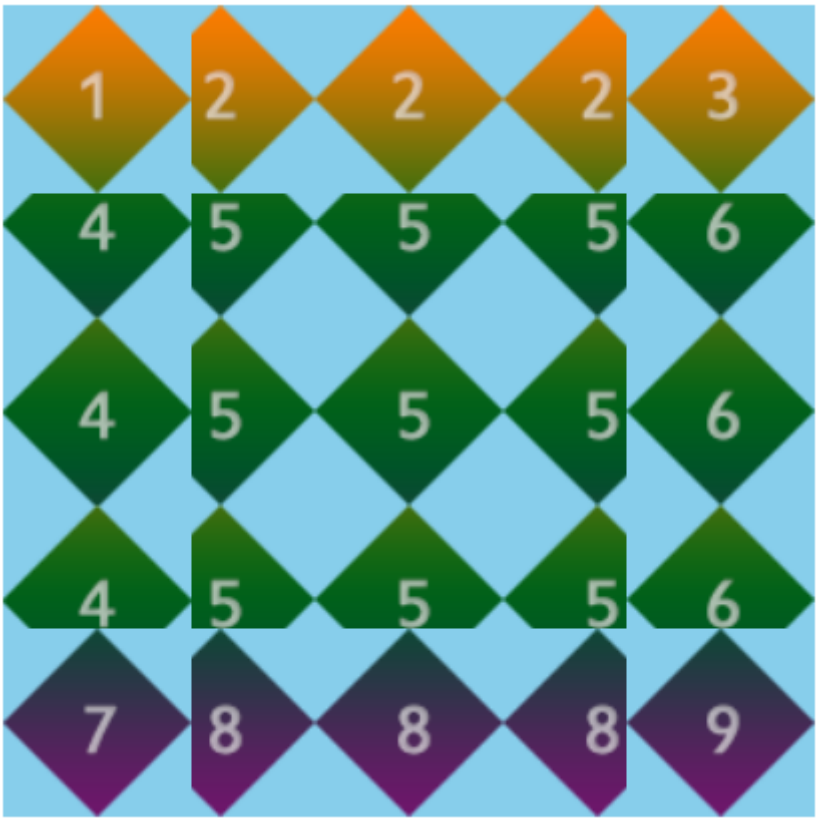
```
    border: 70px solid #000;
    box-sizing: border-box;
    margin: 200px auto;
    background: skyblue;
    /*告诉浏览器让哪一张图片成为边框*/
    /*注意点:
    如果只通过source指定了哪一张图片作为边框的图片，默认情况下会将图片放到边框的四个顶点
    如果设置了边框图片，那么就不会显示边框颜色，边框图片的优先级高于边框颜色*/
    border-image-source: url("images/border.jpg");
    /*告诉浏览器如何对指定的边框图片进行切割
    注意点: 不带单位
    */
    border-image-slice: 70 70 70 70;
    /*告诉浏览器边框图片显示的宽度，并不是指定边框的宽度
    注意点: 如果通过border-image-width指定了边框图片的宽度，那么默认的边框宽度就会失效*/
    /*border-image-width: 10px;*/

    /*告诉浏览器除了边框图片四个角以外的图片如何填充，默认是拉伸*/
    /*border-image-repeat: stretch;*/
    /*border-image-repeat: repeat;*/
    border-image-repeat: round;
```




```
    /*告诉浏览器边框图片需要向外移动多少
                            上    右    下    左*/
    border-image-outset: 10px 30px 50px 70px;
```




```
简写格式:
/*border-image: 资源地址 切割方式 填充模式;*/
border-image: url("images/border.jpg") 70 fill repeat;
```



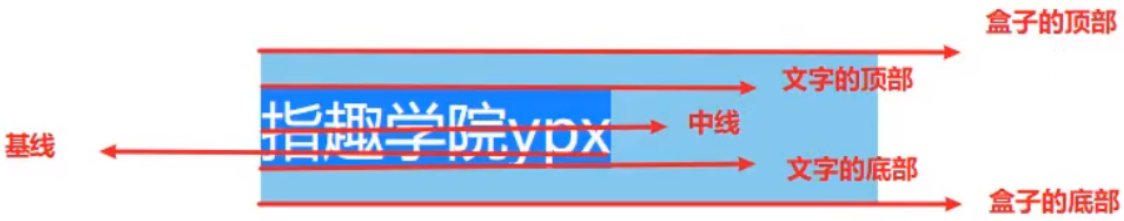
vertical-align

什么是vertical-align?

设置元素的垂直对齐方式

vertical-align的注意点:

```
1. text-align是设置给需要对齐元素的父元素
2. vertical-align是设置给需要对齐的那个元素本身
3. vertical-align只对行内元素有效
```

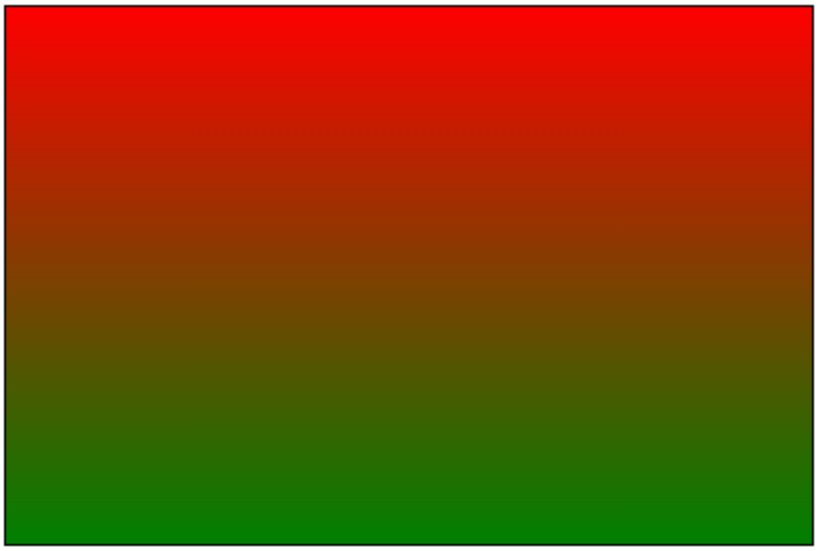



```
默认情况下图片和一行文字的基线对齐
基线就是一行文字中最短那个文字的底部
```

```
vertical-align: baseline;
vertical-align: top;
vertical-align: bottom;
vertical-align: text-top;
vertical-align: text-bottom;
vertical-align: middle;
```

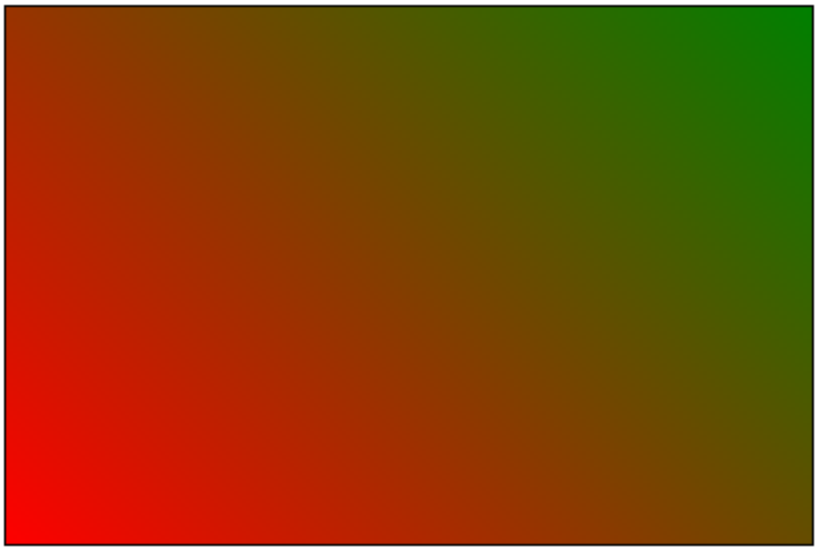
线性渐变

```
background: linear-gradient(red, green);
默认是从上至下渐变
```



```
从下往上
background: linear-gradient(to top ,red, green);
从左往右
background: linear-gradient(to right ,red, green);
从右至左
background: linear-gradient(to left ,red, green);
从左下往右上
background: linear-gradient(to top right ,red, green);
```

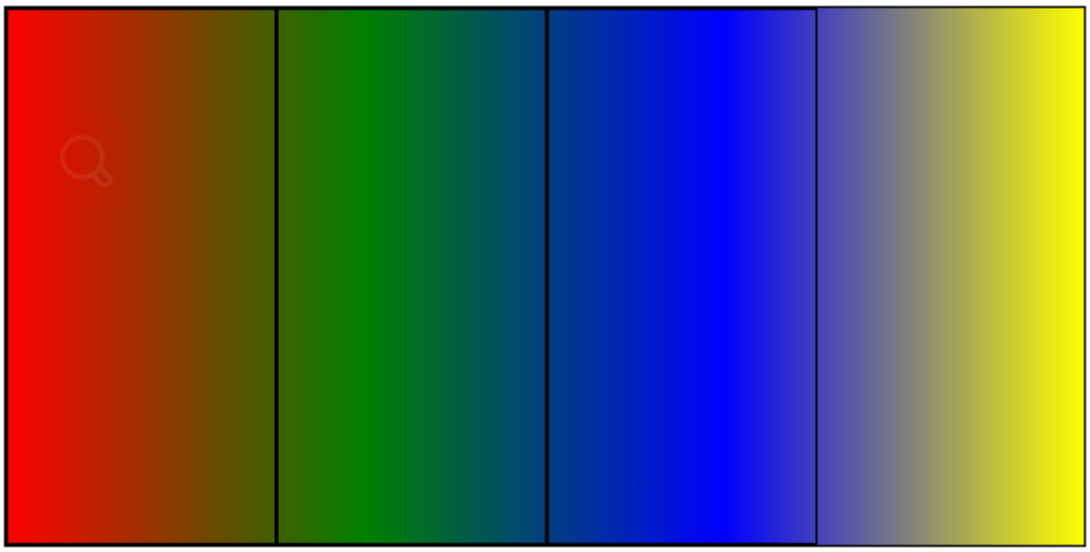
```
45度渐变
background: linear-gradient(45deg ,red, green);
```



线性渐变注意点:

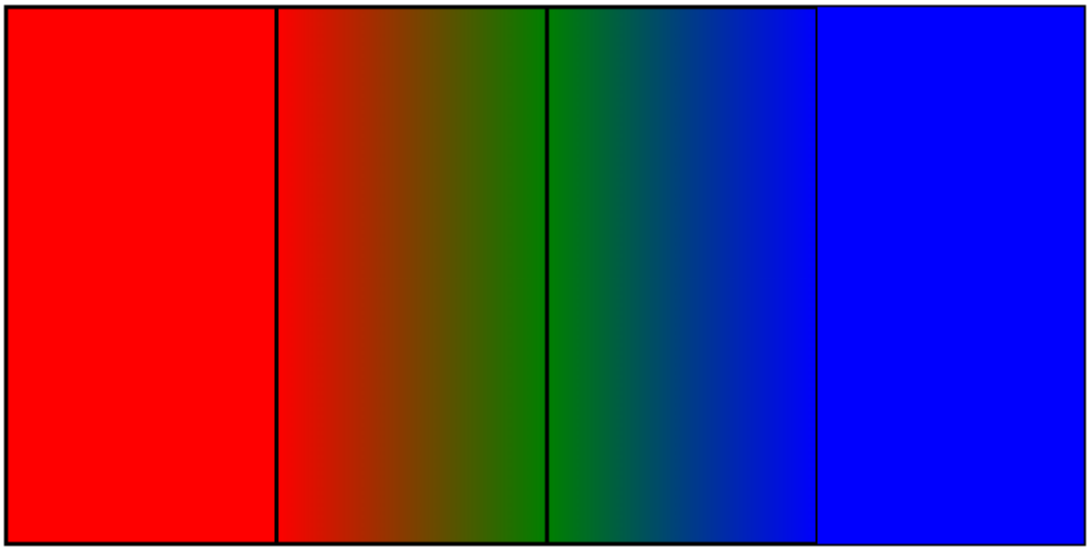
- 至少需要传递2个颜色, 至多没有上限

```
background: linear-gradient(to right, red, green, blue, yellow);
```

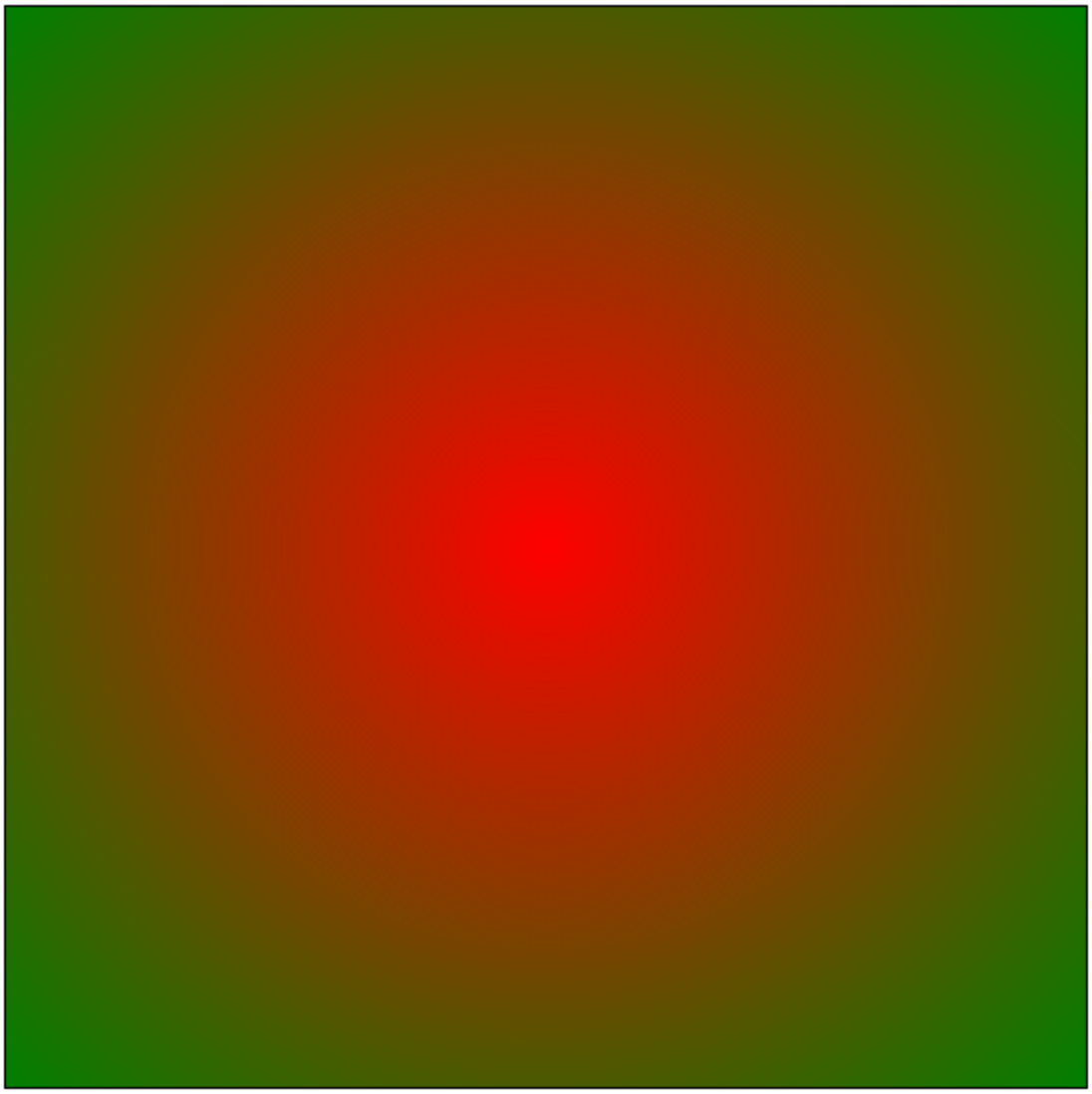


- 默认情况下会自动计算纯色和渐变色的范围, 但是我们也可以手动指定的 格式: 颜色 范围 只有第一个颜色后面的范围是指定纯色的范围, 后面的都是指定渐变的范围

```
background: linear-gradient(to right, red 100px, green 200px, blue 300px);
```



径向渐变



线性渐变：默认从上至下
径向渐变：默认从中心点向四周扩散

```
background: radial-gradient(red, green);
```

线性渐变：可以通过to 关键字的方式修改渐变的方向
径向渐变：可以通过at 关键字的方式修改开始渐变的位置

```
background: radial-gradient(at top left ,red, green);
```

线性渐变：可以通过to deg的方式修改渐变的方向
径向渐变：可以通过at 位置 位置的方式修改开始渐变的位置

```
background: radial-gradient(at 200px 100px ,red, green);
```

线性渐变可以指定纯色和渐变的范围
径向渐变也可以指定扩散的范围

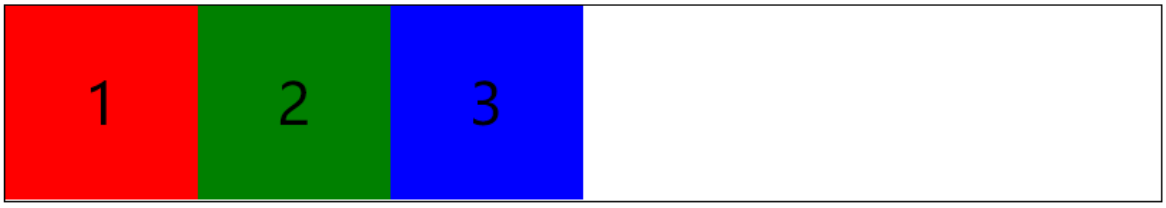
```
background: radial-gradient(100px, red, green);
```

注意点:
如果需要同时指定扩散的位置和扩散的范围，那么范围必须写到at前面

```
background: radial-gradient(100px at 200px 100px ,red, green);
```

伸缩布局

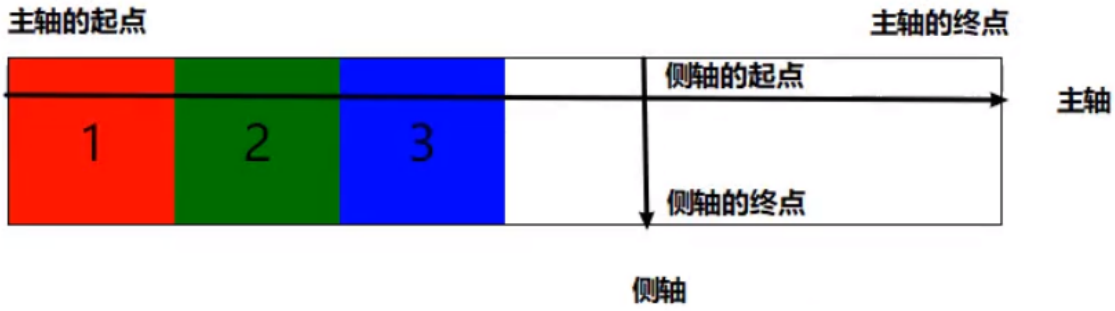
```
display: flex;
```




```
<head>
    <meta charset="UTF-8">
    <title>伸缩布局</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            list-style: none;
            width: 600px;
            border: 1px solid #000;
            margin: 100px auto;
            /*overflow: hidden;*/
            display: flex;
        }
        ul>li{
            width: 100px;
            height: 100px;
            line-height: 100px;
            text-align: center;
            font-size: 30px;
            background: red;
            /*display: inline-block;*/
            /*float: left;*/
        }
        ul>li:nth-child(2){
            background: green;
        }
        ul>li:nth-child(3){
            background: blue;
        }
    </style>
</head>
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
```

```
</ul>
</body>
```

伸缩布局的概念



主轴的方向

在伸缩布局中，默认情况下水平方向是主轴，默认情况下主轴的起点在伸缩容器的最左边，默认情况下所有的伸缩项都是从主轴的起点开始排版的
但是我们也可以通过属性来修改主轴的起点的位置

```
flex-direction: 用于修改主轴起点的位置，写在伸缩容器中
```

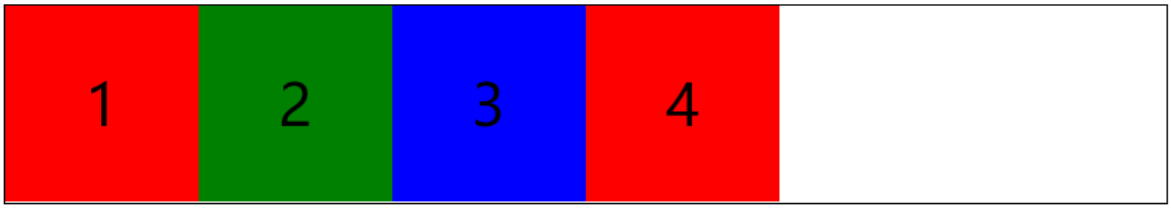
```
flex-direction: row;
flex-direction: row-reverse;
flex-direction: column;
flex-direction: column-reverse;
row: 起点在伸缩容器的最左边，终点在伸缩容器的最右边.伸缩项从左至右的排版，默认的取值
row-reverse: 起点在伸缩容器的最右边，终点在伸缩容器的最左边，从右至左的排版
column: 起点在伸缩容器的最顶部，终点在伸缩容器的最底部，从上至下的排版
column-reverse: 起点在伸缩容器的最底部，终点在伸缩容器的最顶部，从下至上的排版
```

注意点:
在伸缩布局中主轴和侧轴永远都是十字交叉的，只要主轴的方向发生了变化，侧轴也会发生变化

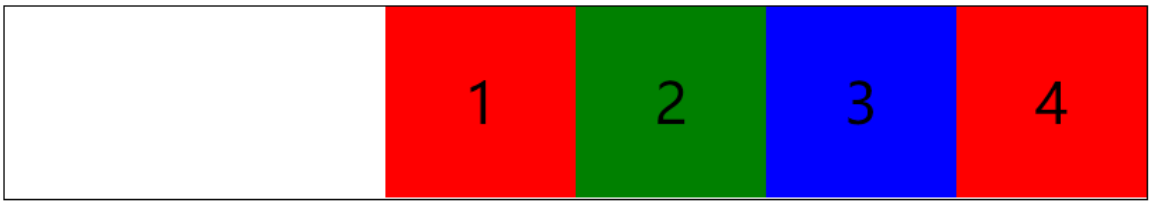
主轴的对齐方式

```
flex-direction: row; 主轴起点的默认值
justify-content: flex-start; 主轴上伸缩项对齐的默认值
注意点: 在设置对齐方式的时候一定要理解两步操作
    1.按照主轴起点对伸缩项进行排版
    2.按照指定的对齐方式对齐排版好的伸缩项,直接平移不会排序
```

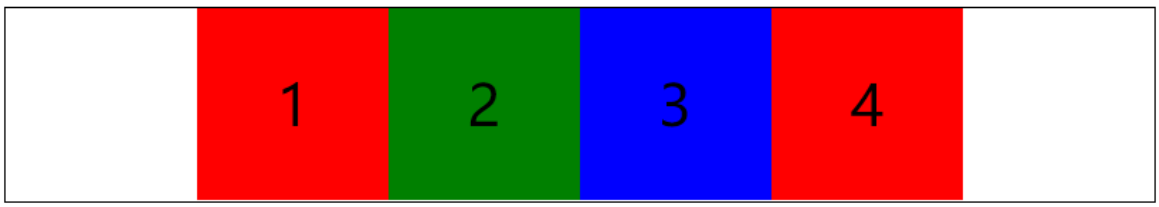
```
justify-content: flex-start
```



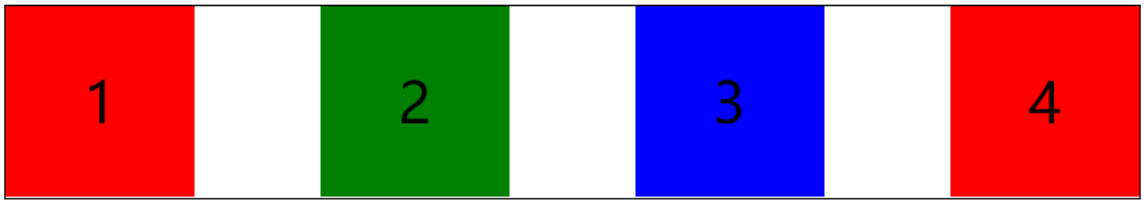
```
justify-content: flex-end;
```



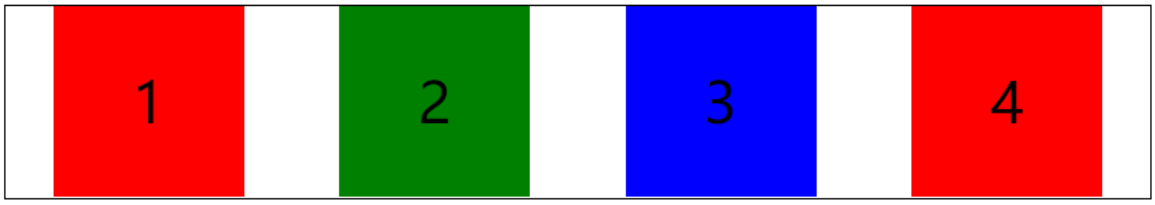

```
justify-content: center;
```




```
justify-content: space-between;
```



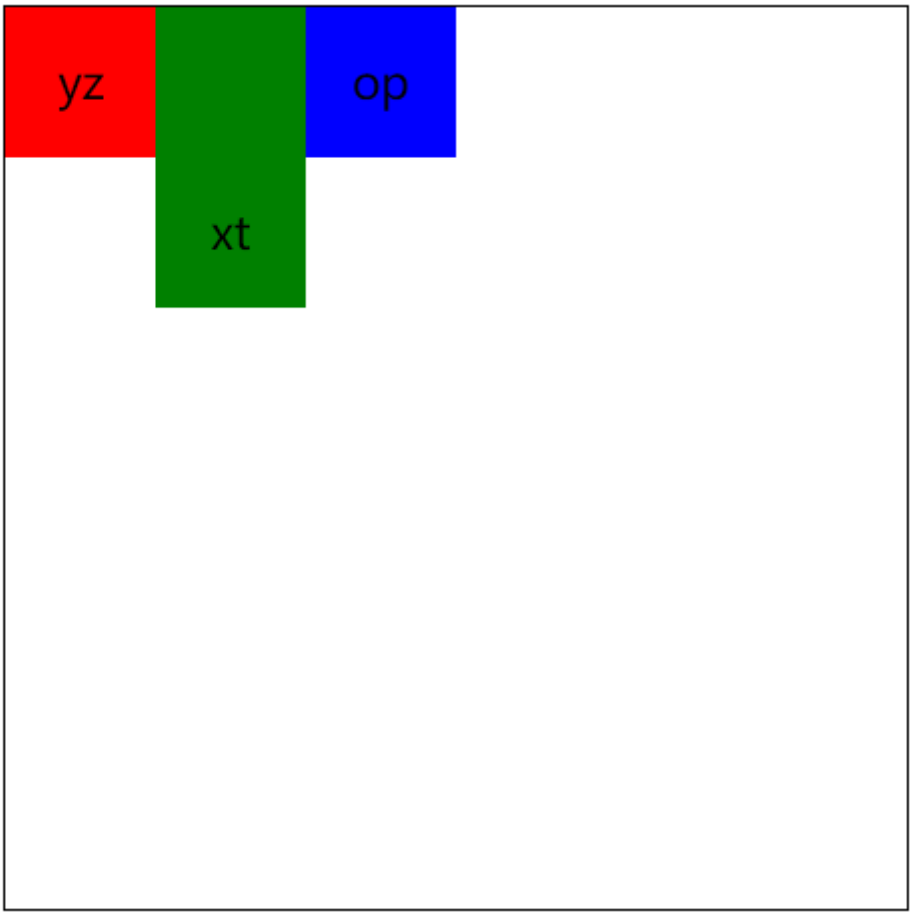

```
justify-content: space-around;
```



侧轴对齐方式一

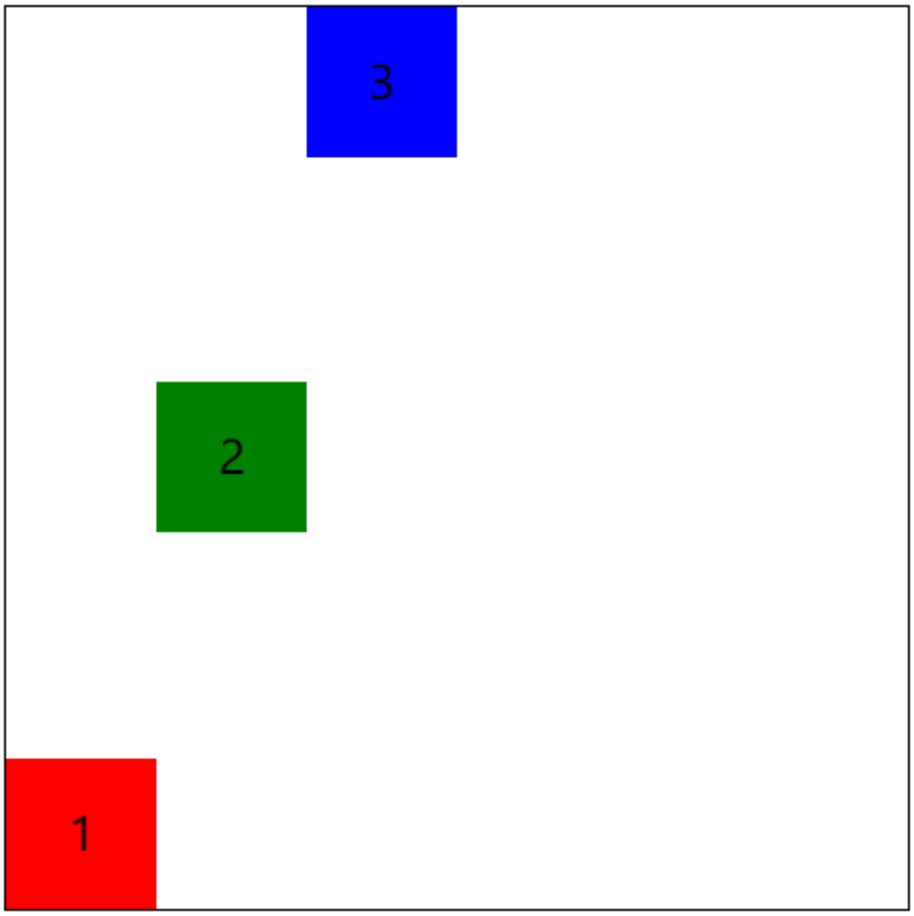
```
<head>
    <meta charset="UTF-8">
    <title>131-伸缩布局</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            list-style: none;
            width: 600px;
            height: 600px;
            border: 1px solid #000;
            margin: 100px auto;
            display: flex;
            /*告诉浏览器主轴的起点在伸缩容器的最左边，告诉浏览器伸缩项从左至右的排版*/
```

```
            flex-direction: row;
            /*告诉浏览器排版好的伸缩项需要和主轴的起点对齐*/
            justify-content: flex-start;
            /*告诉浏览器排版好的伸缩项需要和侧轴的起点对齐,默认取值*/
            /*align-items: flex-start;*/
            /*align-items: flex-end;*/
            /*align-items: center;*/
            /*
            注意点:
            侧轴对比主轴来说没有两端对齐(space-between)和环绕对齐(space-around)
            */
            /*baseline: 让所有伸缩项中的基线在一条直线上对齐*/
            /*align-items: baseline;*/
            /*
            stretch(拉伸对齐/等高对齐):
            让所有的伸缩项的高度变为侧轴的高度
            注意点:
            如果需要设置为拉伸对齐, 那么伸缩项不能设置高度
            如果伸缩项设置了高度, 那么拉伸对齐就会失效
            */
            /*align-items: stretch;*/
        }
        ul>li{
            width: 100px;
            /*不给li设置高度的话，和父元素高度相同，相当于拉伸对齐的效果*/
            height: 100px;
            line-height: 100px;
            text-align: center;
            font-size: 30px;
            background: red;
        }
        ul>li:nth-child(2){
            padding-top: 100px;
            background: green;
        }
        ul>li:nth-child(3){
            background: blue;
        }
    </style>
</head>
<body>
<ul>
    <li>yz</li>
    <li>xt</li>
    <li>op</li>
</ul>
</body>
```



侧轴对齐方式二

如果在伸缩容器中通过 `align-items:` 来控制伸缩项的对齐方式，是一次性（以行为单位）控制所有伸缩项的对齐方式
如果想单独的控制某一个伸缩项在侧轴上的对齐方式，那么需要将控制对齐方式的属性写到伸缩项中`align-self`

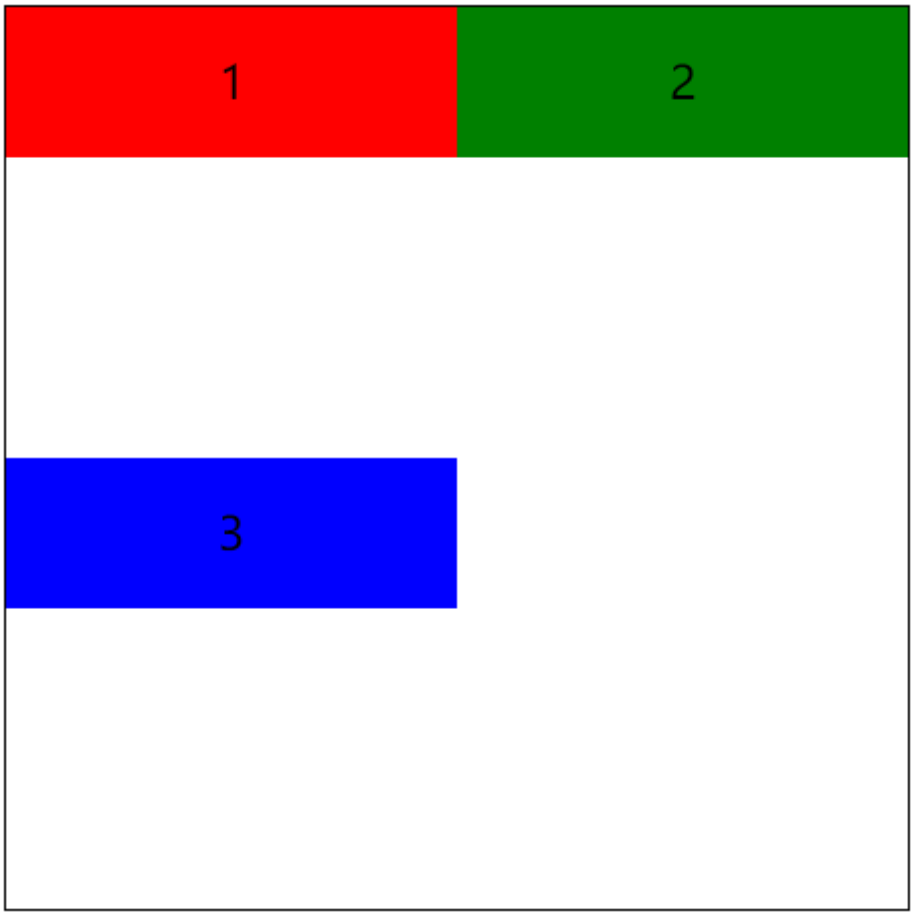



```
        ul>li:nth-child(1){
            align-self: flex-end;
        }
        ul>li:nth-child(2){
            background: green;
            align-self: center;
        }
        ul>li:nth-child(3){
            background: blue;
            align-self: flex-start;
        }
```

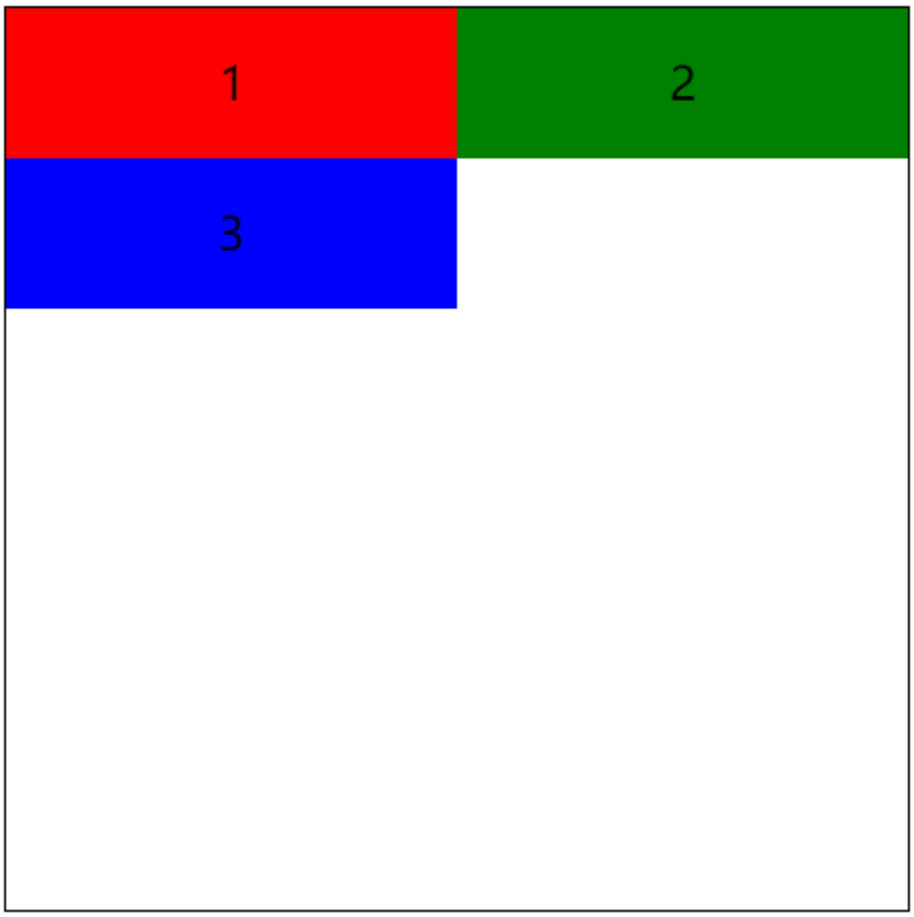
伸缩布局换行和换行对齐的问题

```
<head>
    <meta charset="UTF-8">
    <title>131-伸缩布局</title>
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        ul{
            list-style: none;
            width: 600px;
            height: 600px;
            border: 1px solid #000;
            margin: 100px auto;
            display: flex;
```

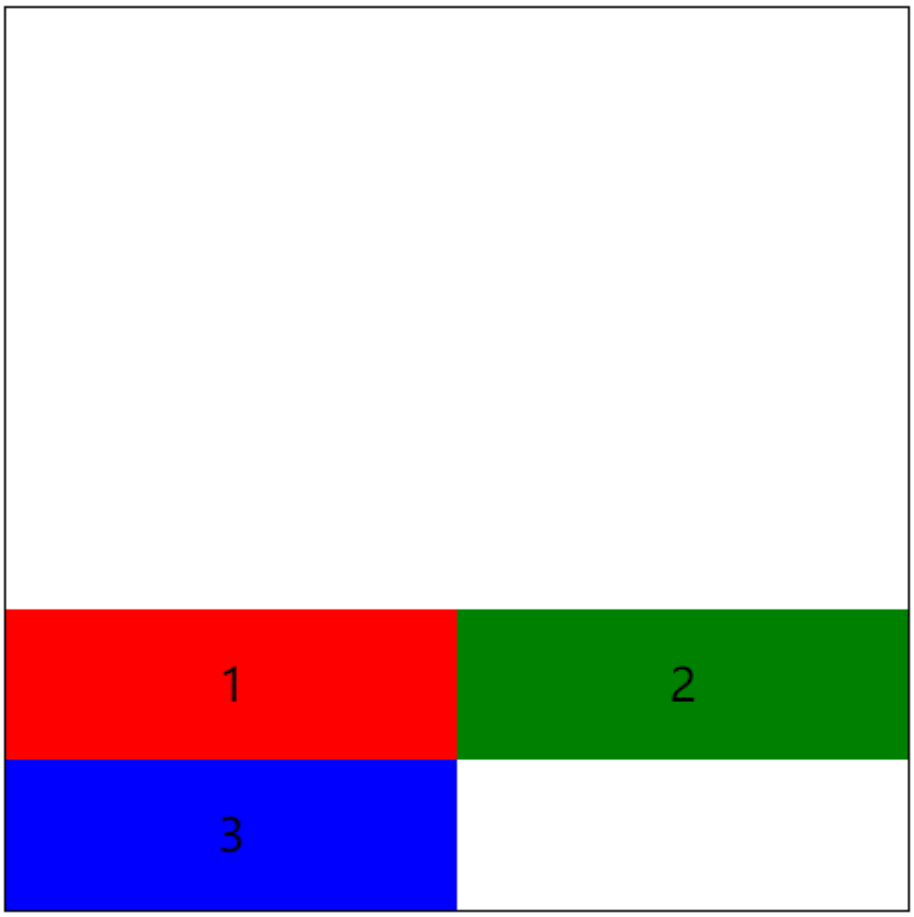
```
            /*
            1.默认情况下如果伸缩容器的一行放不下所有的伸缩项，那么系统会自动等比压缩所有的伸缩项
            2.在伸缩容器中有一个叫做flex-wrap属性，专门用于控制放不下是否需要换行的
            默认的取值：flex-wrap：nowrap 不换行
            wrap：放不下就换行 而不是等比压缩
            wrap-reverse：放不下就换行 ，以行为单位进行反转
            */
            /*flex-wrap: nowrap;*/
            /*flex-wrap: wrap;*/
            /*flex-wrap: wrap-reverse;*/

            /*flex-wrap: wrap;*/
            /*
            在伸缩容器中有一个叫做align-content的属性，是专门用于设置换行之后的对齐方式的
            注意点：只有伸缩项发生了换行这个属性才有效
            flex-start：换行之后和侧轴的起点对齐，一行接一行
            flex-end：换行之后和侧轴的终点对齐，将所有换行之后的内容当做一个整体来操作
            center：换行之后和侧轴的中点对齐
            space-between：换行之后在侧轴上两端对齐
            space-around：换行之后在侧轴上环绕对齐
            stretch：以行为单位进行拉伸，拉伸的部分以空白填充，保证拉伸之后所有的行加起来能够填满侧轴
            */
            /*align-content: flex-start;*/
            /*align-content: flex-end;*/
            /*align-content: center;*/
            /*align-content: space-between;*/
            /*align-content: space-around;*/
            /*align-content: stretch;*/
        }
        ul>li{
            width: 300px;
            height: 100px;
            line-height: 100px;
            text-align: center;
            font-size: 30px;
            background: red;
        }
        ul>li:nth-child(2){
            background: green;
        }
        ul>li:nth-child(3){
            background: blue;
        }
    </style>
</head>
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
</ul>
</body>
```



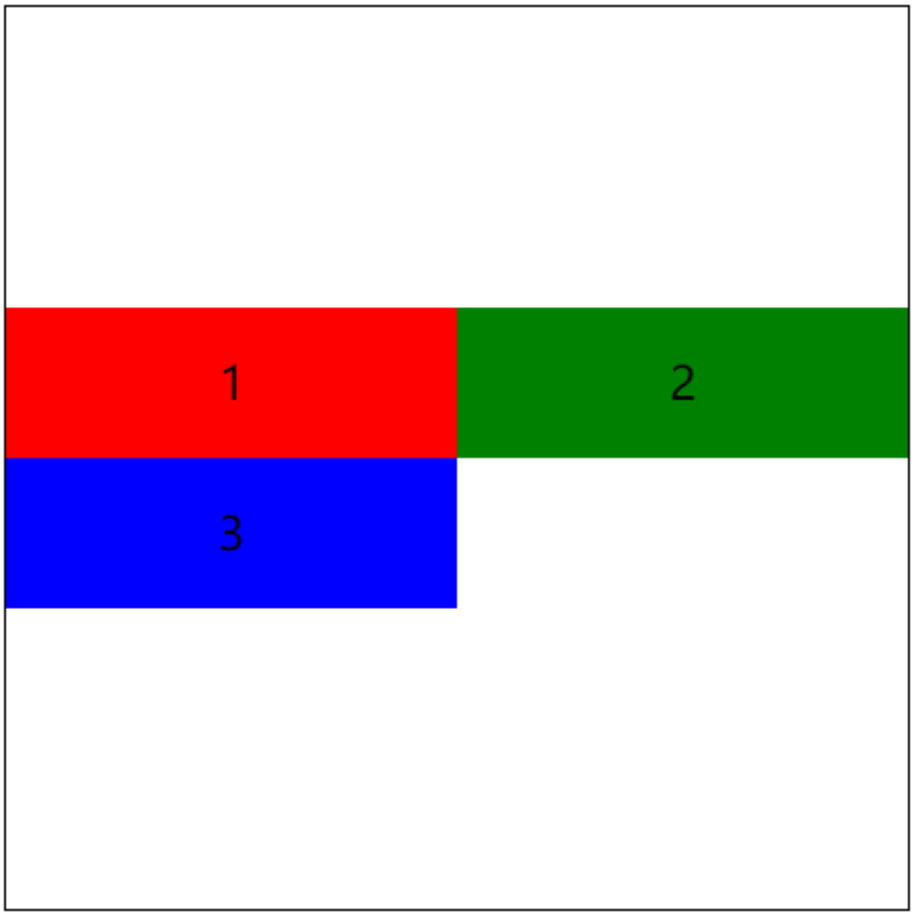

```
align-content: flex-start;
```



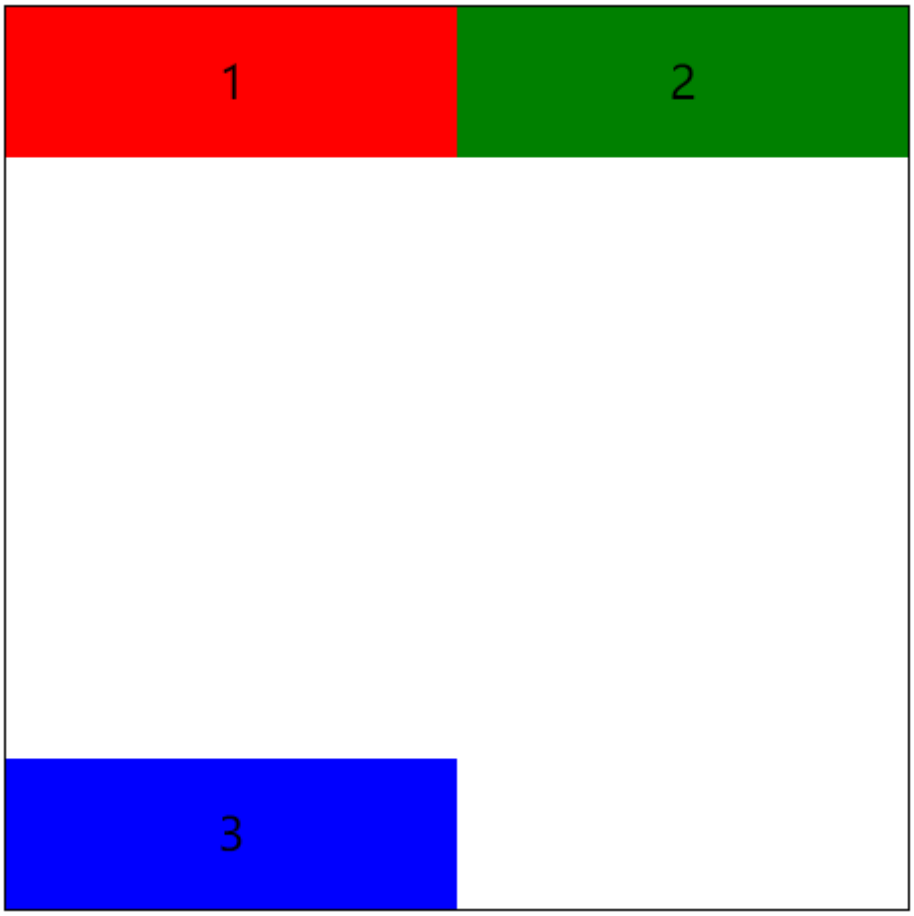
```
align-content: flex-end;
```



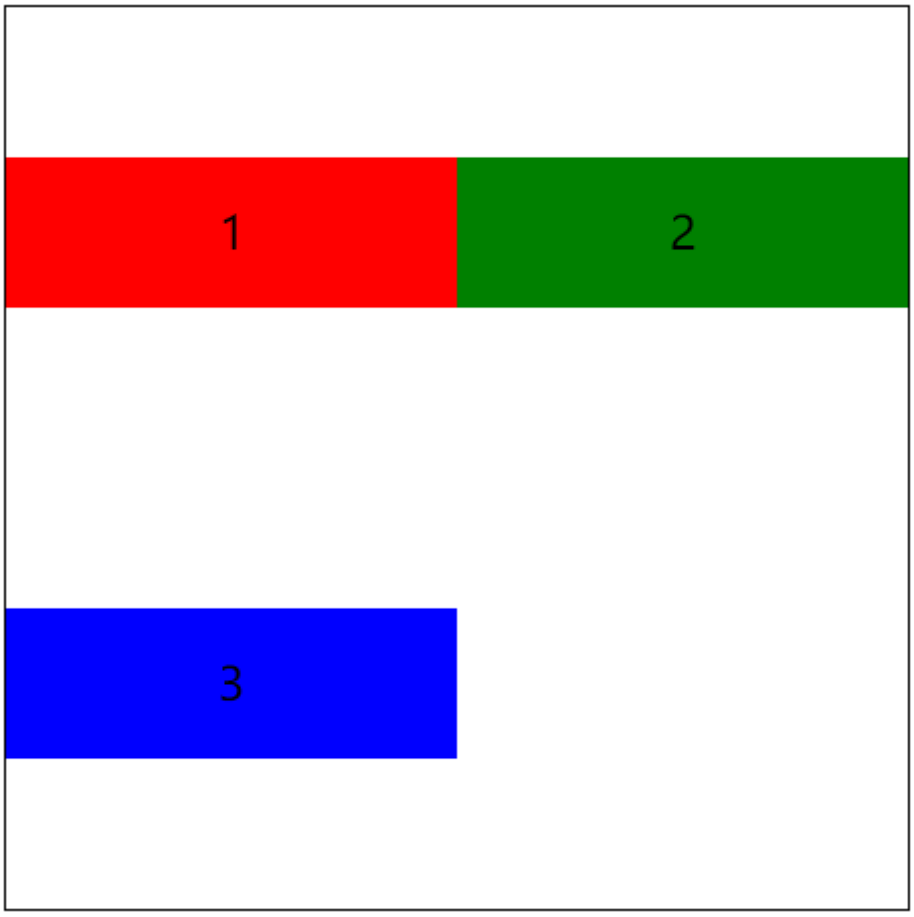

```
align-content: center;
```



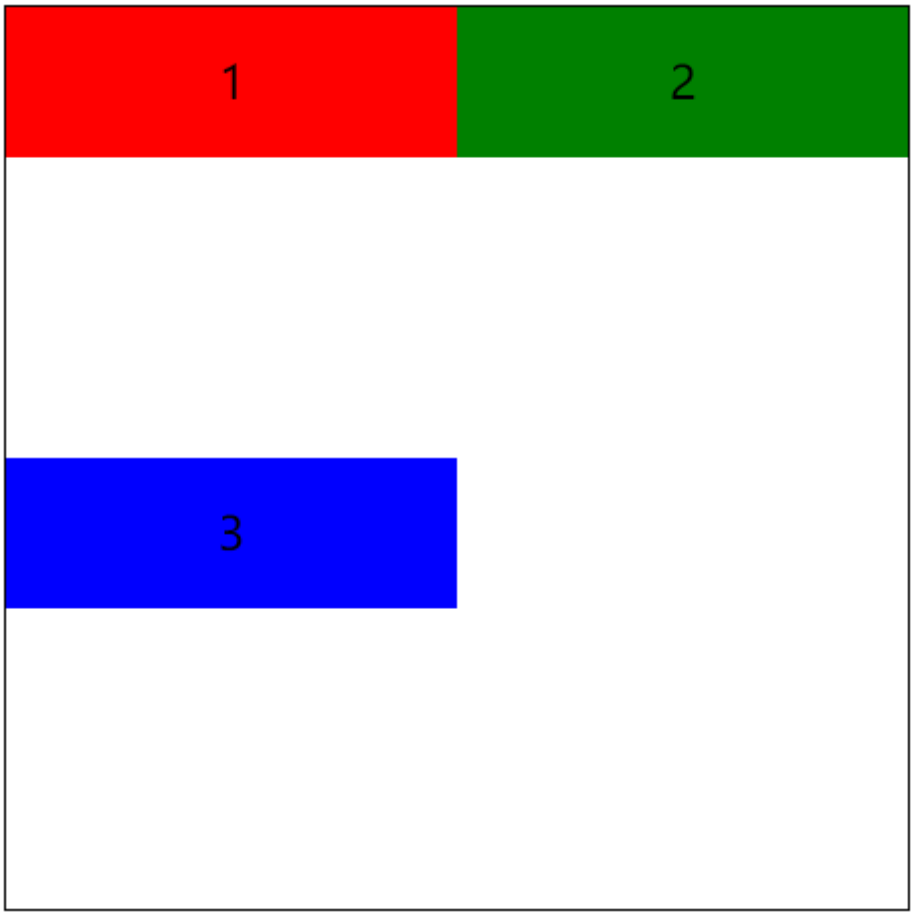
```
align-content: space-between;
```




```
align-content: space-around;
```




```
align-content: stretch;默认
```

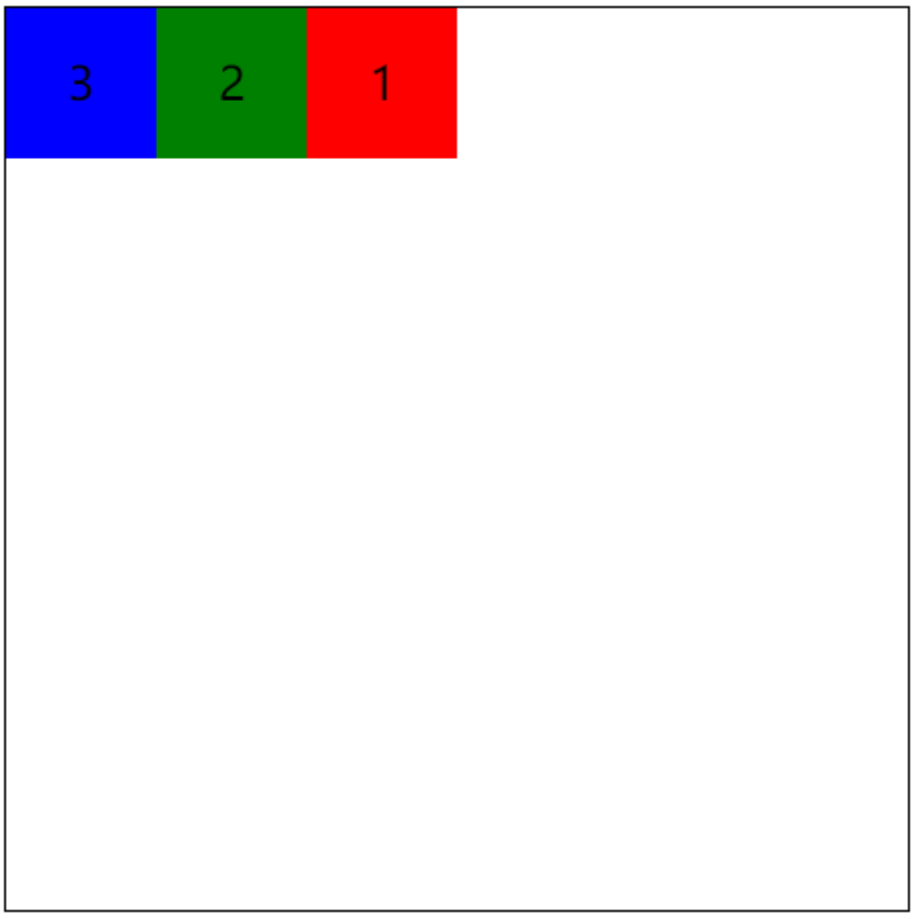


!

伸缩项的排序问题

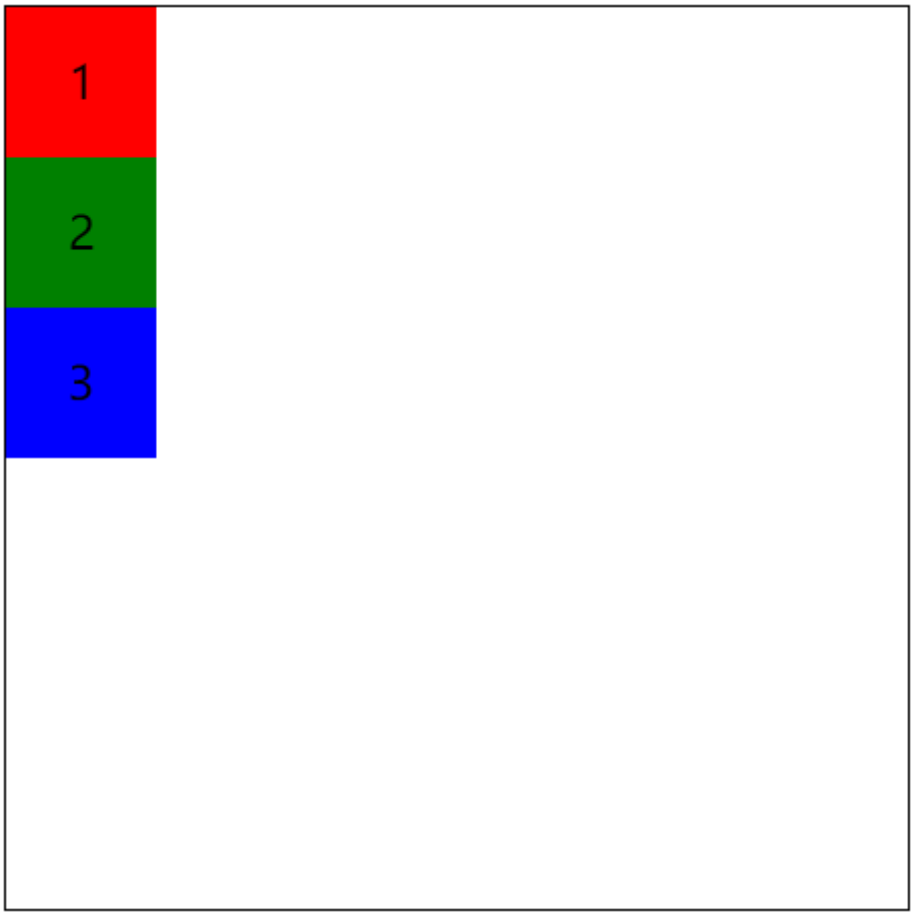
默认情况下每一个伸缩项都有一个order属性，用于决定排序的先后顺序
默认情况下所有伸缩项的order属性的取值都是0
我们可以通过修改order属性的取值来实现伸缩项的排序
order排序的规则：从小到大的排序，越小的显示在越前面，越大的显示在越后面

```
        ul>li:nth-child(1){
            order: 999;
        }
        ul>li:nth-child(2){
            background: green;
            order: 0;
        }
        ul>li:nth-child(3){
            background: blue;
            order: -1;
        }
```

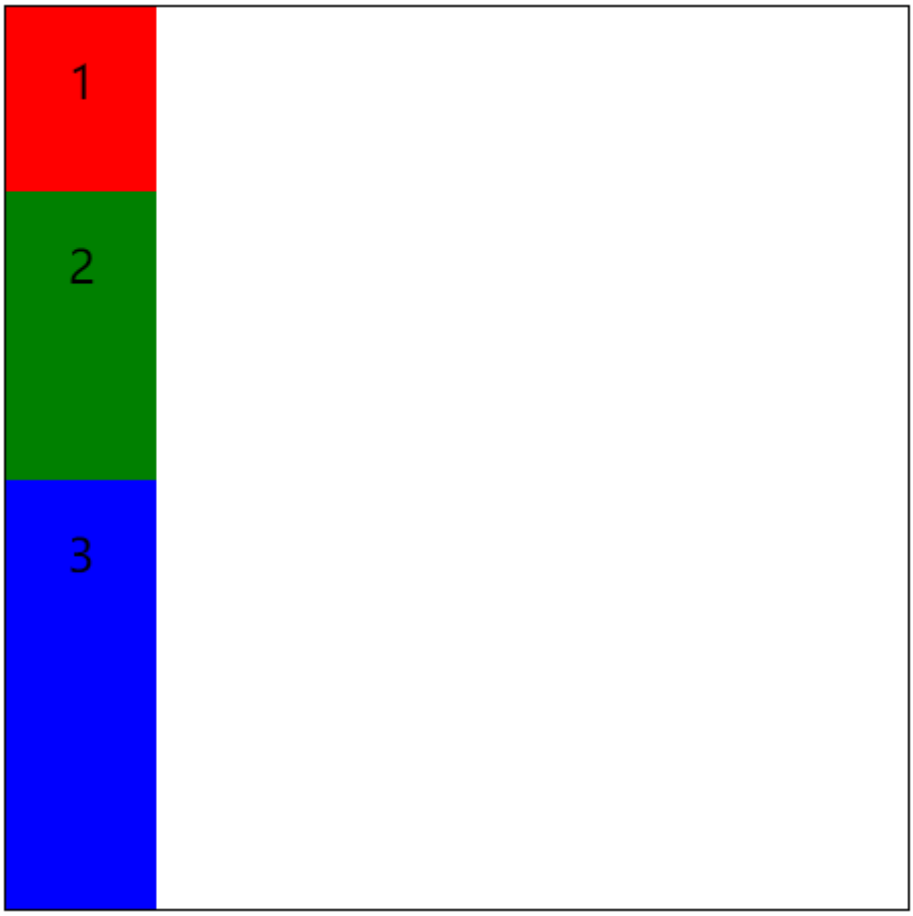


伸缩项的扩充

默认情况下




```
        ul>li:nth-child(1){
            /*
            在伸缩项中有一个flex-grow属性，用于控制当所有伸缩项的宽度总和小于伸缩容器宽度的时候如何扩充自己，以便于所有伸缩项宽度的总和能够填满整个伸缩容器
            默认情况下flex-grow的取值是0，表示我们设置的宽度是多少就按照多少来显示，不进行任何的扩充,如上图
            注意点:
            只有当所有伸缩项的宽度总和小于伸缩容器宽度的时候flex-grow这个属性才有效

            flex-grow缩小的公式
            1.利用伸缩容器宽度 - 所有伸缩项的宽度 = 剩余空间
              600 - 300 = 300
            2.利用剩余空间 / 所有需要扩充份数的总和 = 每一份的大小
              300 / (1 + 4 + 8) = 23.07
            3.利用当前伸缩项的宽度 + 需要的份数的宽度
              第一个伸缩项 = 100 + (1 * 23.07) = 123.07
              第二个伸缩项 = 100 + (4 * 23.07) = 192.28
              第三个伸缩项 = 100 + (8 * 23.07) = 284.56
            */
            flex-grow: 1;
        }
        ul>li:nth-child(2){
            background: green;
            /*flex-grow: 4;*/
        }
        ul>li:nth-child(3){
            background: blue;
            /*flex-grow: 8;*/
        }
```

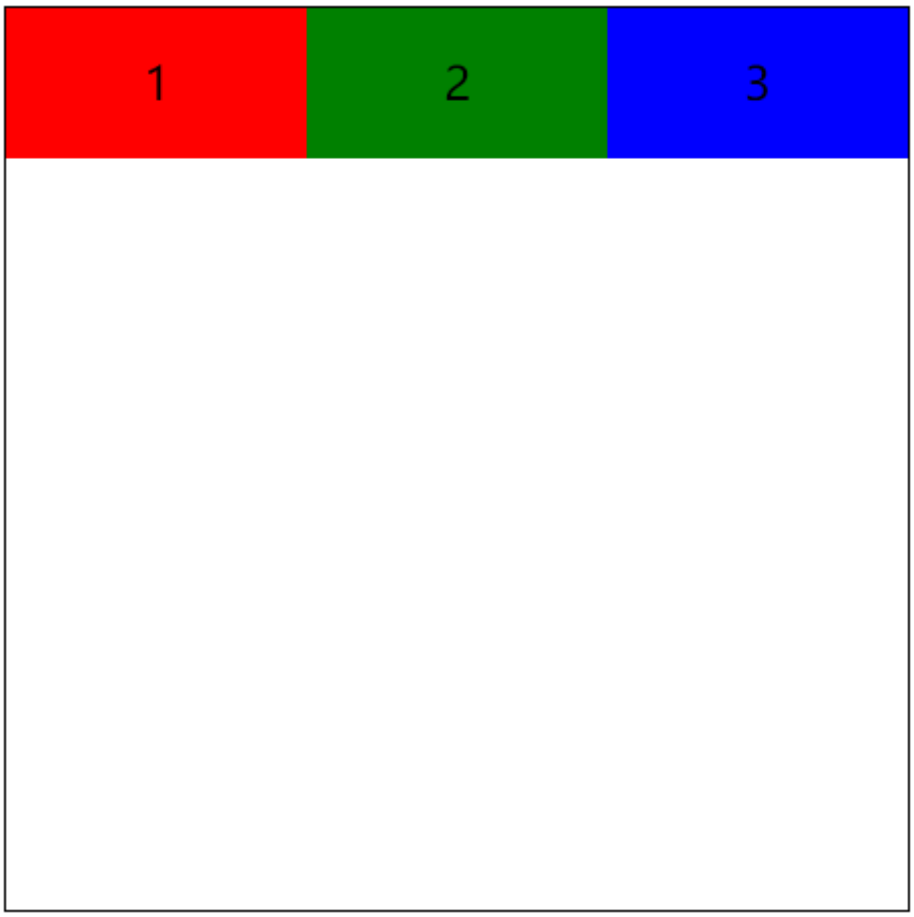


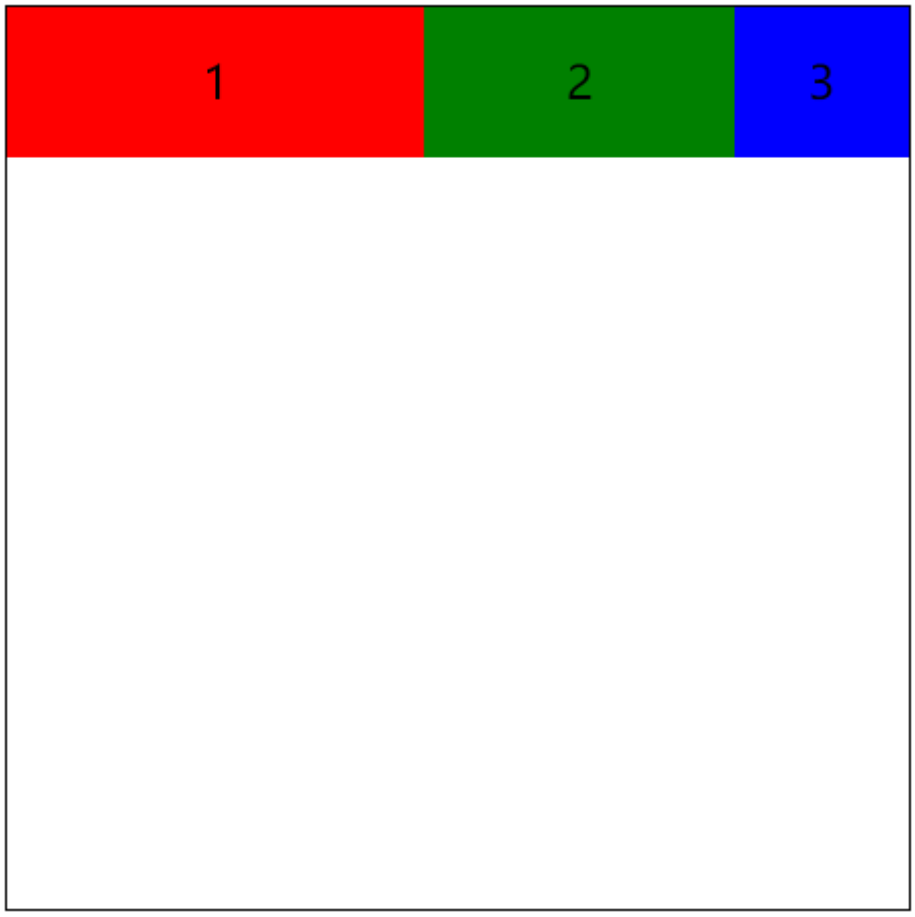
扩充之后如上图

伸缩项的缩小

```
        ul>li:nth-child(1){
            /*
            在伸缩项中有一个flex-shrink属性，用于控制当所有伸缩项的宽度总和大于伸缩容器宽度的时候如何缩小自己，以便于所有伸缩项宽度的总和能够填满整个伸缩容器
            默认情况下flex-shrink的取值是1，表示当所有伸缩项宽度的总和大于伸缩容器宽度的时候等比缩小自己
            注意点:
            只有当所有伸缩项的宽度总和大于伸缩容器宽度的时候flex-shrink这个属性才有效


            flex-shrink扩充的公式
            1.利用所有伸缩项的宽度总和 - 伸缩容器宽度 = 溢出的宽度
            900 - 600 = 300
            2.计算权重值
            利用每一个伸缩项需要的份数 * 当前伸缩项的宽度 然后再相加
            1 * 300 + 4 * 300 + 8 * 300 = 3900
            3.计算每个伸缩项需要缩小的范围
            溢出的宽度 * 当前伸缩项的宽度 * 当前伸缩项需要的份数 / 权重值
            300 * 300 * 1 / 3900 = 23.07
            第一个伸缩项宽度 = 300 - 23.07 = 276.9
            300 * 300 * 4 / 3900 = 92.3
            第二个伸缩项宽度 = 300 - 92.3 = 207.6
            */
            flex-shrink: 1;
        }
        ul>li:nth-child(2){
            background: green;
```

```
        flex-shrink: 4;
    }
    ul>li:nth-child(3){
        background: blue;
        flex-shrink: 8;
    }
```





伸缩项扩充和缩小的注意点

1.如果没有指定flex-grow属性，或者flex-grow:的值是0，那么当前的伸缩项不会被扩充
2.如果flex-shrink的值是0，那么当前的伸缩项不会被缩小
3. 注意点
   前面所写的注释都是说宽度扩充或者宽度缩小，但是这种说法是不严谨的
   也有可能扩充和缩小的是高度，到底是宽度还是高度是由主轴决定的，扩充和缩小的是主轴方向上的值
   也就是说如果主轴是水平方向的，那么扩充和缩小的就是宽度
   也就是说如果主轴是垂直方向的，那么扩充和缩小的就是高度

伸缩项宽度设置

1.在伸缩布局中可以通过flex-basis属性设置伸缩项的宽度
  注意点: flex-basis 只有在伸缩布局中才有效
2.在伸缩布局中如果通过flex-basis设置了宽度，那么再通过width设置宽度就会无效
  也就是说flex-basis的优先级要高于width的优先级
3.在伸缩布局中如果同时通过flex-basis和width设置了宽度，而且一个设置的是auto,一个设置的是具体的值，那么会按照具体的值来显示

伸缩项属性连写

```
/*
flex: 扩充 缩小 宽度;
flex默认值
flex: 0 1 auto;
*/
flex: 0 1 100px;
```