CSS基础知识巩固

参考：后盾人css学习文档，非常万分感谢！

选择器

- 子选择器，选择a的孩子节点b，只能管到孩子，子孙不管 `a>b`

```
/*html*/
<article>
    <h1>儿子1</h1>
    <aside>
        <h2>孙子</h2>
    </aside>
    <h2>儿子2</h2>
</article>

/*css*/
article>h2{
    color: blue;
}
```

> **效果：儿子2变蓝**

- 兄弟选择器
  - 选择a之后的所有兄弟节点 `a~b`

```
/*html*/
<article>
    <h2>兄弟0</h2>
    <h1>兄弟1</h1>
    <h2>兄弟2</h2>
    <h2>兄弟3</h2>
</article>

/*css*/
article h1 ~ h2{
    color: blue;
}
```

> **效果：儿子2、3变蓝**

  - 选择a之后临近的兄弟节点 `a+b`

```
/*html*/
<article>
    <h2>兄弟0</h2>
    <h1>兄弟1</h1>
    <h2>兄弟2</h2>
    <h2>兄弟3</h2>
</article>

/*css*/
article h1 + h2{
  color: blue;
}
```

> **兄弟2变蓝**

属性选择器

- h1[title] 筛选具有title属性的h1标签
  - h1[title="test"] 筛选title属性为"test"的h1标签
  - h1[title^="test"] 筛选title属性值以"test"开头的h1标签
  - h1[title$="test"] 筛选title属性值以"test"结尾的h1标签
  - h1[title*="test"] 筛选title属性值含有"test"的h1标签
  - h1[title~="test"] 筛选title属性值如下的h1标签
    - 以"test"单词(即，test前后有空格)开始
    - 其后有连字符

```
/*html*/
<article>
    <h1 title="test">兄弟0</h2>
    <h1 title="test.com">兄弟1</h1>
    <h1 title="test-com">兄弟2</h2>
    <h1 title=" test">兄弟3</h2>
</article>

/*css*/
article h1[title|="test"]{
    color: blue;
}
```

> **兄弟0、2变蓝**

伪类选择器

- **伪类可以看作以选中元素为基准点，此元素的一些状态或属性**

> **a:link** 选择未被访问的链接
>
> **a:visited** 选择所有访问过的链接
>
> **a:hover** 选择鼠标滑过的链接
>
> **a:active** 选择激活的链接
>
> **input:focus** 选择输入后具有焦点的元素

> **:root** 选择文档的根元素
>
> **:empty** 选择所有没有子元素的p元素
>
> **:not** 排除选择器

- **:target**

:target 用于选择当前活动的锚点元素

```
/*html*/
<p><a href="#news1">Jump to New content 1</a></p>
<p><a href="#news2">Jump to New content 2</a></p>

<p id="news1"><b>New content 1...</b></p>
<p id="news2"><b>New content 2...</b></p>

/*css*/
:target{
    color: blue;
}
```

> **点击后跳转到指定锚点元素，并使其变蓝**

- **:first-child**

选择父元素的第一个子元素

```
/*css*/

/*
article标签内的每一层的第一个子元素
    article->h1 儿子1蓝
    aside->h2   孙子蓝
*/
article :first-child{
    color: blue;
}

/*
匹配作为任意元素的第一个子元素的article元素
*/
article:first-child{
    /*body->article 所有都蓝(因为设置了article内部都蓝)*/
    color: blue;
}
h2:first-child {
    /*aside-h2 孙子蓝*/
    color: blue;
}

/*html*/
<article>
    <h1>儿子1</h1>
    <aside>
        <h2>孙子</h2>
    </aside>
    <h2>儿子2</h2>
```

```
</article>
```

- **:first-of-type**

p:first-of-type 选择的每个p元素是其父元素的第一个p元素

```
/*html*/
<article>
    <h1>儿子1</h1>
    <aside>
        <h2>孙子</h2>
    </aside>
    <h2>儿子2</h2>
</article>

/*css*/
h2:first-of-type {
    /*article->h2 儿子2蓝*/
    /*aside->h2 孙子蓝*/
    color: blue;
}
```

大部分的input类型控件不支持::after和::before,与浏览器兼容性也有关系。

在Chrome环境下，buttom 、 number 、text 、 email 等不支持

[更多的伪类]

权重

权重

规则	粒度
ID	0100
class，类属性值	0010
标签,伪元素	0001
*	0000
行内样式	1000

> ***通配符权重 0,继承 NULL ， 0>NULL**

```
*{
    color: red;
}
h2{
    color: green;
}
<article>
    <h2>父亲<span>儿子</span></h2>
</article>
```

> 显示红色

文本控制

字重

字重指字的粗细定义。取值范围 `normal | bold | bolder | lighter | 100 ~900`。

400对应 `normal`,700对应 `bold` ，一般情况下使用 bold 或 normal 较多。

百分数

百分数是子元素相对于父元素的大小，如父元素是20px，子元素设置为 200%即为父元素的两倍40px。

em

em单位等同于百分数单位，即：1em = 100%

vw|vh

桌面端指的是浏览器的可视区域；移动端指的就是Viewport中的Layout Viewport。

```
vw：1vw /*等于视口宽度的1%*/
vh：1vh /*等于视口高度的1%*/
/*“视区”所指为浏览器内部的可视区域大小，即window.innerWidth/window.innerHeight大小，不包含任务栏标题栏以及底部工具栏的浏览器区域大小。*/
```

行高定义c

```
div {
    font-size:14px;
    line-height: 1.5em;
}
/*行高可以根据字体大小自适应*/
```

组合定义

可以使用 `font` 一次将字符样式定义，但要注意必须存在以下几点：

- 必须有字体规则
- 必须有字符大小规则

```
span {
    font: bold italic 20px/1.5 'Courier New', Courier, monospace;
}
```

大小写转换

- 小号的大写字母

```
font-variant: small-caps;
```

- 字母大小写转换

```
/* 首字母大小 */
text-transform: capitalize;
/* 全部大小 */
text-transform: uppercase;
/* 全部小写 */
text-transform: lowercase;
}
```

文本线条

```
/*下划线*/
text-decoration: underline;
/*删除线*/
text-decoration: line-through;
/*上划线*/
text-decoration: overline;
```

文本阴影

参数列表【颜色，水平偏移，垂直偏移，模糊度】

```
text-shadow: rgba(13, 6, 89, 0.8) 3px 3px 5px;
```

空白处理

```
/*保留文本中的所有空白，同pre标签*/
white-space: pre;
/*合并空白，保留换行符*/
white-space: pre-line;
/*保留空白，保留换行符*/
white-space: pre-wrap
/*禁止文本换行*/
white-space: nowrap
```

文本溢出

- 单行文本

```
/*溢出文本进行换行*/
overflow-wrap: break-word;
/*溢出内容改为...*/
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

- 多行文本

```
width: 200px;
overflow: hidden;
display: -webkit-box;
-webkit-box-orient: vertical;
-webkit-line-clamp: 2;
```

---

word-break: break-all VS overflow-wrap:break-word VS overflow-wrap:anywhere

```
overflow-wrap:normal|break-word|anywhere
/*参数说明*/
1.normal /*默认值,使用浏览器默认的处理方式*/
2.break-word /*设置当字符串过长时，允许在单词内断行-断行发生在单词的空白处*/
3.anywhere /*设置当字符串过长时,允许在单词内断行 -断行发生在单词的任意位置*/
```

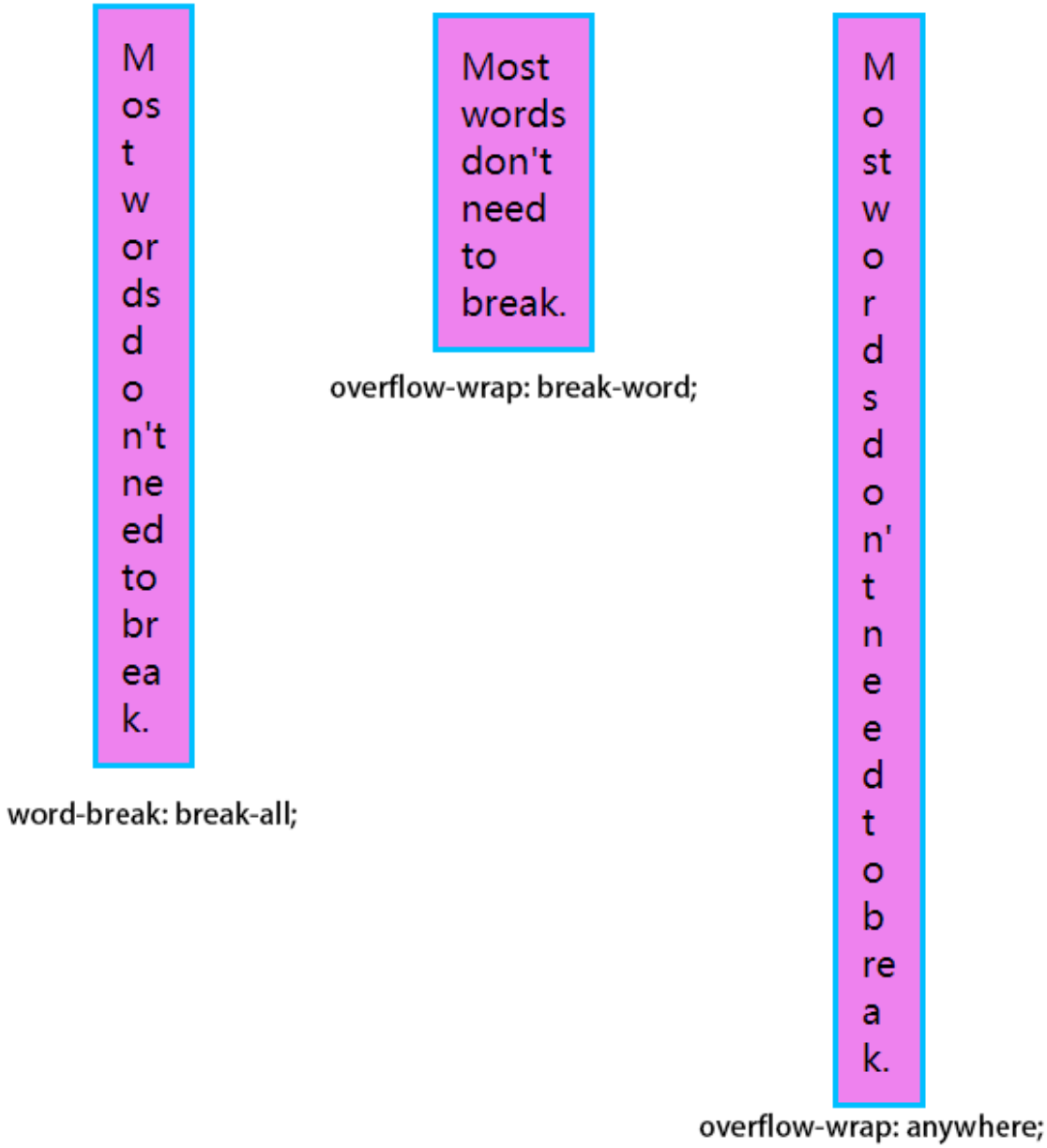
浏览器兼容性

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox Android	Opera Android	iOS Safari	Samsung Internet
overflow-wrap	23	18	49	5.5	12.1	6.1	4.4	25	49	12.1	7	1.5
anywhere	80	80	65	No	67	No	80	80	65	No	No	13.0
break-word	1	12	3.5	5.5	10.5	1	≤37	18	4	11	1	1.0

> 与word-break相比，overflow-wrap仅在无法将整个单词放在自己的行而不会溢出的情况下才会产生换行。

> 连续的英文字符如果可以不用断就不断，如果实在不行，就断开，因此相比break-all可能会留白。

> overflow-wrap:anywhere 就像是 overflow-wrap:break-word 和 word-break:break-all 声明的混合体，主要用在弹性布局中， 即元素尺寸足够的时候单词尽量完成显示，不随便中断，如果尺寸不够，那就能断则断




```
/*css*/
<style>
    p {
        display: inline-block;
        width: min-content;
        padding: 10px;
        margin-left: 100px;
        border: solid deepskyblue;
        vertical-align: top;
        background: violet;
    }
    p + p {
        margin-top: 20px;
    }
    .breakAll {
        word-break: break-all;
    }
    .breakWord {
```

```
        overflow-wrap: break-word;
    }
    .anywhere {
        overflow-wrap: anywhere;
    }
</style>

/*html*/
<p class="breakAll">Most words don't need to break.</p>
<p class="breakWord">Most words don't need to break.</p>
<p class="anywhere">Most words don't need to break.</p>
```

文本间距

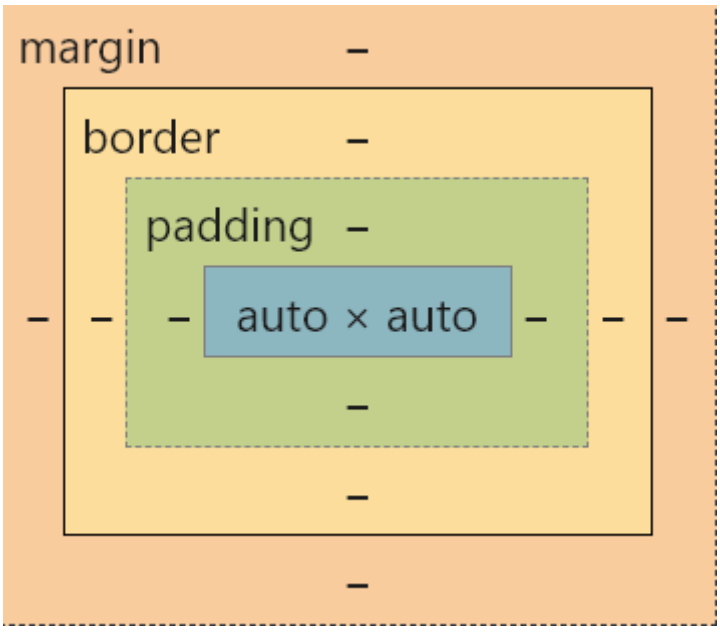
字符间距|单词间距

```
word-spacing: 2em; /*word-spacing不支持汉字*/
letter-spacing: 1em;
```

排版模式

```
/*水平方向自上而下的书写方式*/
writing-mode:horizontal-tb
/*垂直方向自右而左的书写方式*/
writing-mode:vertical-rl
/*垂直方向内内容从上到下，水平方向从左到右*/
writing-mode:vertical-lr
```

盒子模型



外边距合并问题

[嵌套和非嵌套的边距合并问题及解决方案]

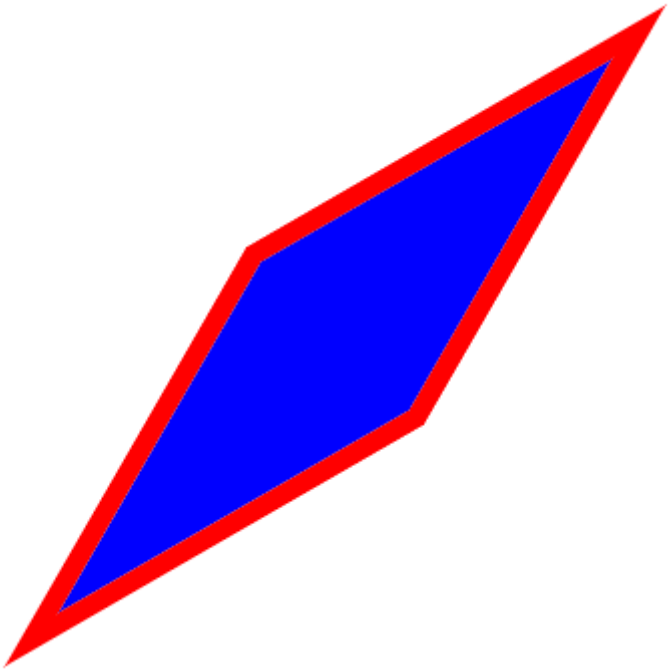
box-sizing

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

- content-box下定义宽和高，会限制content的宽高，整个盒子模型的宽高需要结合外层属性计算。
- border-box定义宽高直接定义盒子模型的宽高，不随内容的变化而变化。

outline

轮廓线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。（不占用空间）



```
/*轮廓线不一定是矩形*/
<style>
.diamond {
    margin: 20px auto;
    height: 20px;
    width: 20px;
    background: blue;
    transform: skew(-30deg,-30deg);
    outline: red solid 2px;
}
</style>

<div class="diamond"></div>
```

元素隐藏

```
/*不占用空间*/
display:none;
/*占用空间*/
visibility:hidden;
/*占用空间*/
opacity:0;
```

溢出overflow

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。

尺寸定义

fill-available

自动填充可用空间

`width: fill-available`对于行内块（inline-block）和块元素（block）起作用,便于进行等高布局

```
<style>
.box {
    height: 20px;
    width:100px;
}
.son {
    width: 30%;
    height: -webkit-fill-available;
    background: #f00;
    display: inline-block;
}
</style>

<div class="box">
    <div class="son"></div>
    <div class="son"></div>
    <div class="son"></div>
</div>
```

根据内容宽度适应

```
width:fit-content; /*根据内容自动适应宽度*/
width:min-content; /*根据最小的内容宽度设置*/
width:fit-content; /*根据最大的内容宽度设置*/
```

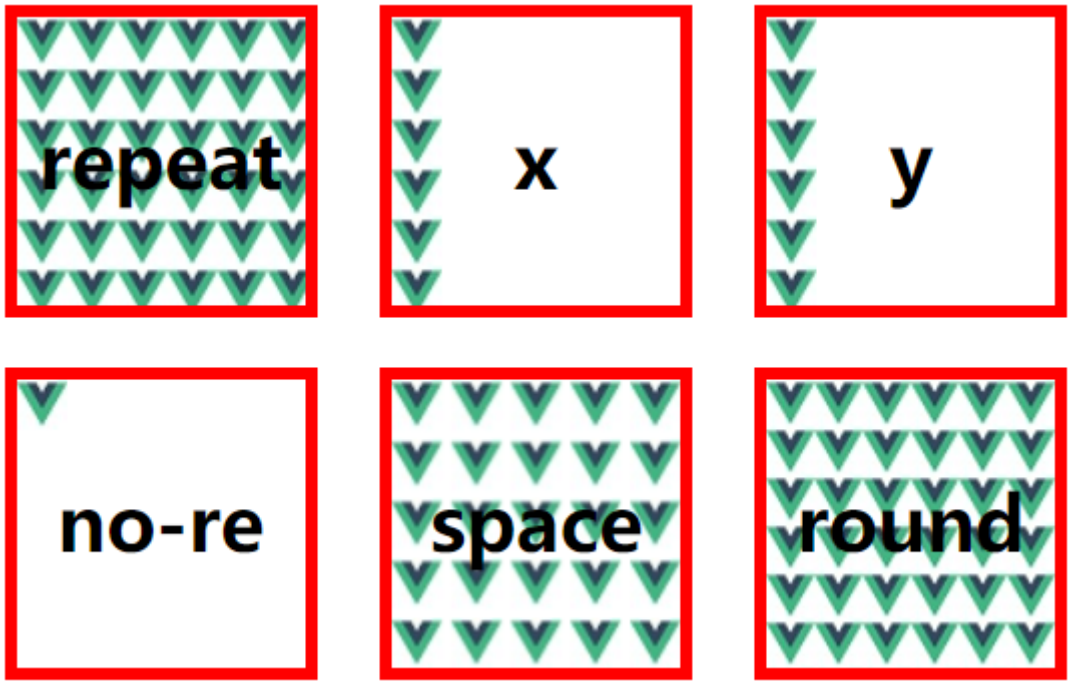
背景

规定背景绘制区域

```
background-clip: border-box(default) | padding-box | content-box;
/*绘制到边框*/
border-box
/*绘制到内边距*/
padding-box
/*绘制到内容*/
content-box
```

背景图重复

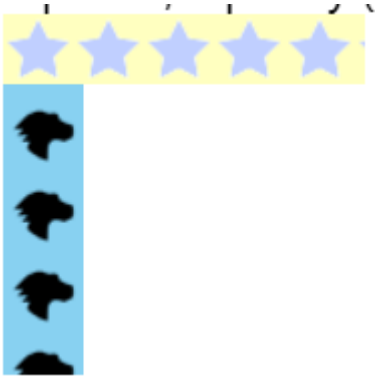
```
/*均匀地分布，图像会尽可能得重复，但是不会裁剪*/
/*图像太大了,没有足够空间完全显示，才会发生裁剪*/
background-repeat: space;
/*缩放，随着允许的空间在尺寸上的增长，被重复的图像将会伸展(没有空隙)，直到有足够的空间来添加一个图像*/
background-repeat: round;
```



> 单值语法是完整的双值语法的简写

```
.five {
    background-image:
 url(https://mdn.mozillademos.org/files/12005/starsolid.gif),

url(https://developer.cdn.mozilla.net/media/redesign/img/favicon32.png);
    background-repeat: repeat-x,
                       repeat-y;
    height: 144px;
}
```

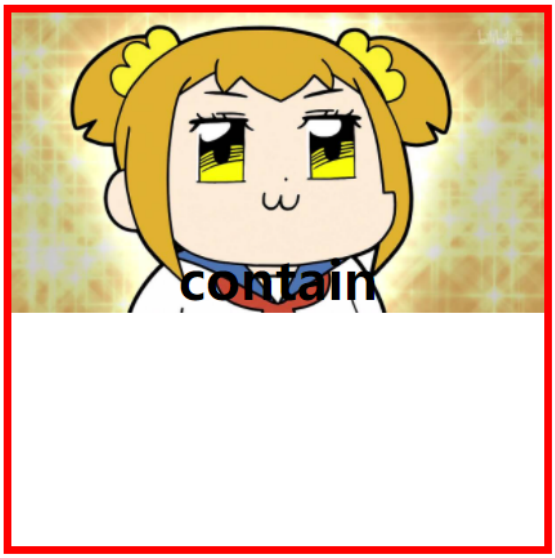


背景固定

```
/*背景固定，背景图像会随着页面其余部分的滚动而移动*/
background-attachment: fixed;
/*背景固定，当页面的其余部分滚动时，背景图像不会移动*/
background-attachment: scroll(default);
```

背景位置\尺寸

```
/* 背景位置，x|y单位 <---> px | % | top\left\right\bottom\center */
background-position: x y;
/* 图片尺寸，length宽高|percentage以父元素百分比|cover|contain*/
/* length|percentage可以使用auto */
/* cover 把背景图像扩展至足够大,背景图某些部分无法显示*/
/* contain 把背景图像扩展到最大尺寸，宽度高度完全适应内容区域,会留白(已经设置no-repeat) */
background-size: length|percentage|cover|contain;
```



组合语句

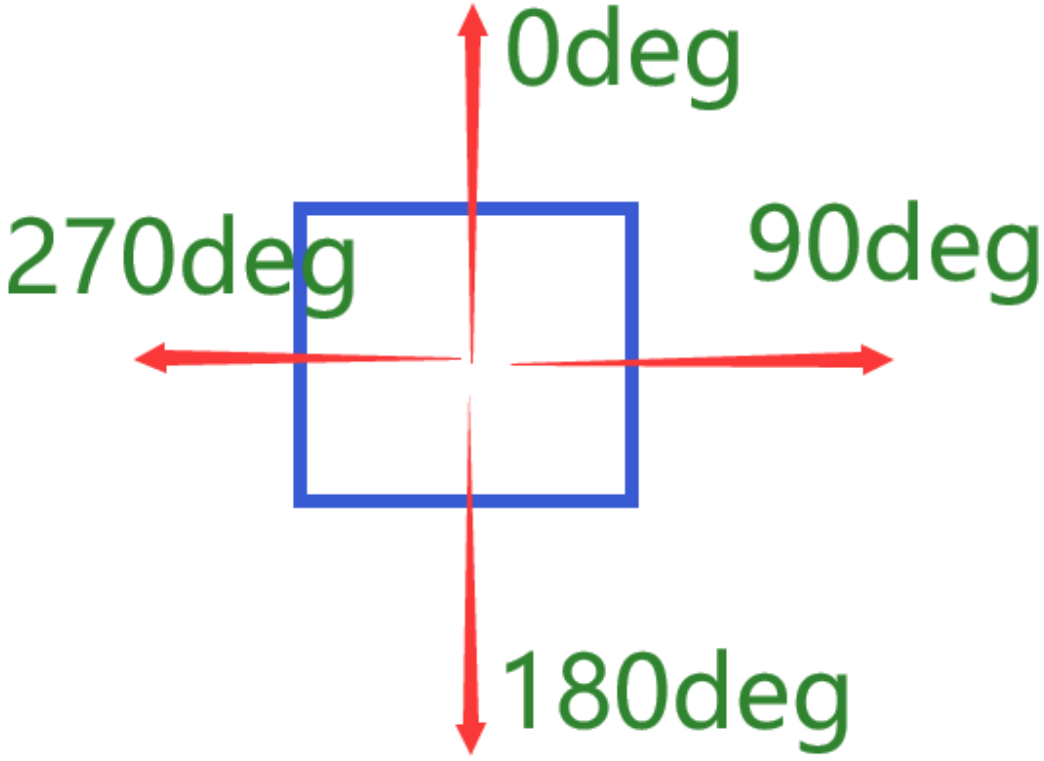
```
/*background缩写，属性顺序有影响*/
background: url(./1.jpg) no-repeat center   /contain  ; /*显示背景图*/
background: url(./1.jpg) center no-repeat   /contain  ; /*不显示背景图*/

/*顺序*/
1. background-color 2. background-image 3. background-repeat 4. background-
attachment 5. background-position
/* attachment:fixed 不能与 position 的方位属性值一起使用，只能与x|y单位 <---> px一起使用
*/
```

颜色渐变

linear-gradient创建颜色线性渐变，返回`<gradient>`是一种特殊的`<image>`数据类型

```
/*默认的线性渐变，默认从上到下*/
background: linear-gradient(red, green);
/*渐变角度，方向*/
/*direction: top|left|right|bottom*/
background: linear-gradient(30deg, red, green);
background: linear-gradient(to [direction], red, green);
```



径向渐变

radial-gradient创建颜色径向渐变，从原点出发的渐变，圆形或椭圆形，返回`<gradient>`是一种特殊的`<image>`数据类型

```
/*默认的径向渐变，默认从内而外*/
background: radial-gradient(#e66465, #9198e5);
/*设置渐变的宽高*/
background: radial-gradient(100px 100px, #e66465, #9198e5);
/*渐变原点位置*/
/*direction: top|left|right|bottom*/
/*direction: percentage*/
background: linear-gradient(at [direction | x%\y%], red, green);
```

标识位

在渐变的颜色列表中插入标志位，控制渐变开始的范围，0%~100%，标识位重合时，无渐变

```
background: linear-gradient(45deg, red 50%, blue 60%);
```

中间阈值

两种渐变颜色的中间点

```
background: linear-gradient(45deg, red ,50% ,blue);
```

渐变重复

```
background: repeating-linear-gradient(90deg, blue, red)
```

数据内容样式

css定制表格

```
table     块级表格来（<table>），表格前后带有换行符。
table-row-group     一个或多个行的分组（<tbody>）。
table-header-group     一个或多个行的分组（<thead>）。
table-footer-group     一个或多个行的分组（<tfoot>）。
table-row     表格行（<tr>）。
table-column-group     一个或多个列的分组（<colgroup>）。
table-column     单元格列（<col>）
table-cell     表格单元格（<td> 和 <th>）
table-caption     表格标题（<caption>）
```

内容对齐

```
text-align: center(default)|left|right;
vertical-align: middle(default)|top|bottom;
```

单元格间距

```
table{
    /*设置单元格间距,会发生合并*/
    border-spacing: 50px 10px;
}
```

边框合并

```
table{
    border-collapse: separate(default)|collapse;
}
```

隐藏空单元格

```
table{
    empty-cells: show(default)|hide;
}
```

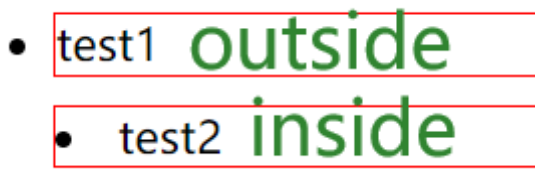
列表

```
<ul>有序列表</ul>
<ol>无序列表</ol>
```

list-style-type值	描述
none	无标记
disc	默认，实心圆
circle	空心圆
square	实心方块
demical	阿拉伯数字.
decimal-leading-zero	0开头的阿拉伯数字.
lower-roman	小写罗马数字.
upper-roman	大写罗马数字.
lower-alpha	小写字母.
upper-alpha	大写字母.
lower-greek	小写希腊字母.
lower-latin	小写拉丁字母（现在没有区别，拉丁字母早期在ie7和以下版本不支持）
upper-latin	大写拉丁字母
hebrew	希伯来编号（特殊编号）
armenian	亚美尼亚编号（特殊编号）
georgian	乔治亚编号（特殊编号）
cjk-ideographic	汉语数字（简单的表意数字）
hiragana	标记是：a, i, u, e, o, ka, ki, 等（日文片假名）
katakana	标记是：A, I, U, E, O, KA, KI, 等（日文片假名）
hiragana-iroha	标记是：i, ro, ha, ni, ho, he, to, 等（日文片假名）
katakana-iroha	标记是：I, RO, HA, NI, HO, HE, TO, 等（日文片假名）

列表标注位置

```
 list-style-position: inside|outside(default);
外部（outside）标志会放在离列表项边框边界一定距离处。
内部（inside）标志处理像是插入在列表项内容最前面的行内元素一样
```



自定义列表样式

```
list-style-image: url(1.png);
list-style-image: radial-gradient(10px 10px, red, black); /*会使list-style-type失效，默认是square*/
```

文本缩进

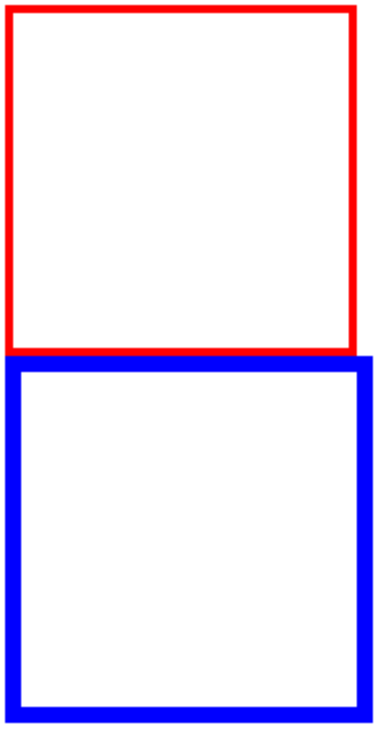
```
 text-indent: 20px;
```

```
/*利用背景图制作假的list的样式*/
ul {
    list-style-type: none;
}
ul li {
    background-image: url(1.jpg);
    background-repeat: no-repeat;
    background-size: 10px 10px;
    background-position: 0 5px;
    text-indent: 20px;
}
```

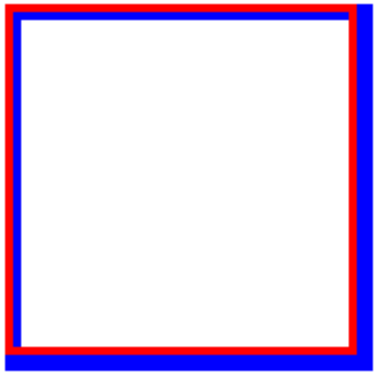
浮动布局

文件流

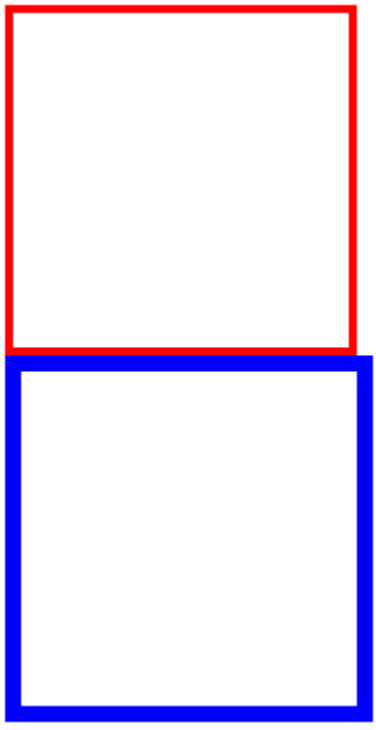
```
div:nth-of-type(1) {
    border: 5px solid red;
}
div:nth-of-type(2) {
    border: 10px solid blue;
}
```



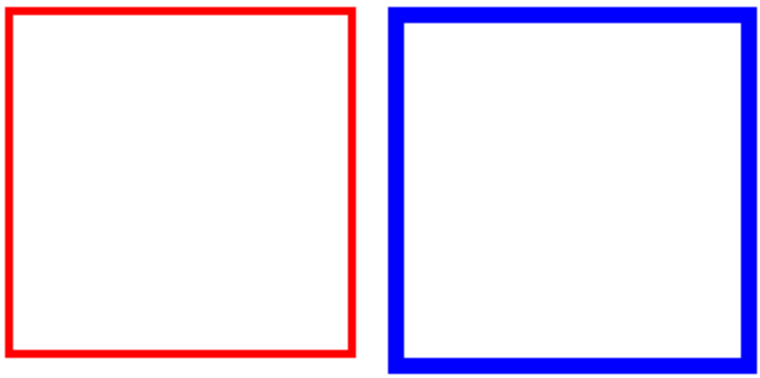
```
div:nth-of-type(1) {
    border: 1px solid red;
    float:left;
}
div:nth-of-type(2) {
    border: 3px solid blue;
}
```



```
div:nth-of-type(1) {
    border: 1px solid red;
}
div:nth-of-type(2) {
    border: 3px solid blue;
    float:left;
}
```



```
div:nth-of-type(1) {
    border: 1px solid red;
    float:left;
}
div:nth-of-type(2) {
    border: 3px solid blue;
    float:left;
}
```



> **总结：**
>
> 从文档流的角度来理解：
>
> 没有设置浮动的div元素独占一行，浮动元素不占用空间，**脱离文档流**，只对后面的元素有影响。
>
> 当两个元素都浮动后，就会进入一个流，会并列排布，且受到margin的影响。
>
> 元素浮动后会变成**行级块。**
>
> 浮动在父级元素的内边框内部。

清除浮动

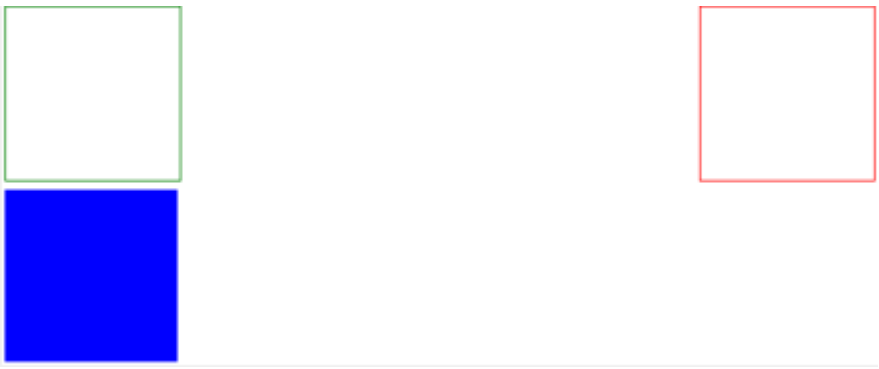
> clear: left|right|both;
> 清除浮动，清除的是其他元素受到浮动元素的影响

```
div.green {
    border: solid 2px green;
    float: left;
}

div.red {
    border: solid 2px red;
    float: right;
}

div.blue {
    background: blue;
    clear: both;
}

<div class="green"></div>
<div class="red"></div>
<div class="blue"></div>
```



```
.clearfix {
    clear: both;
    height: 0;
}

<main>
    <div>float 1</div>
    <div>float 2</div>
    <article class="clearfix"></article>
</main>
/*添加一个子元素clearfix，虽然main无法感知两个float的元素，但是可以感知clearfix，clearfix清除浮动后，会在两个浮动元素下面，达到撑开父级元素main的效果*/
```

> 伪类::after清除浮动

```
content:"";
clear:both;
display:block;
```

> 触发BFC机制

```
overflow:hidden|scroll|auto;
```

环绕距离

```
/*围绕外边距|边界|内边距|内容*/
/*关键字*/
shape-outside: margin-box|border-box|padding-box;
```

[shape-outside高级用法]

区域形状定制

```
/*使用裁剪方式创建元素的可显示区域。区域内的部分显示，区域外的隐藏*/
/*相当于蒙版*/
clip-path
```

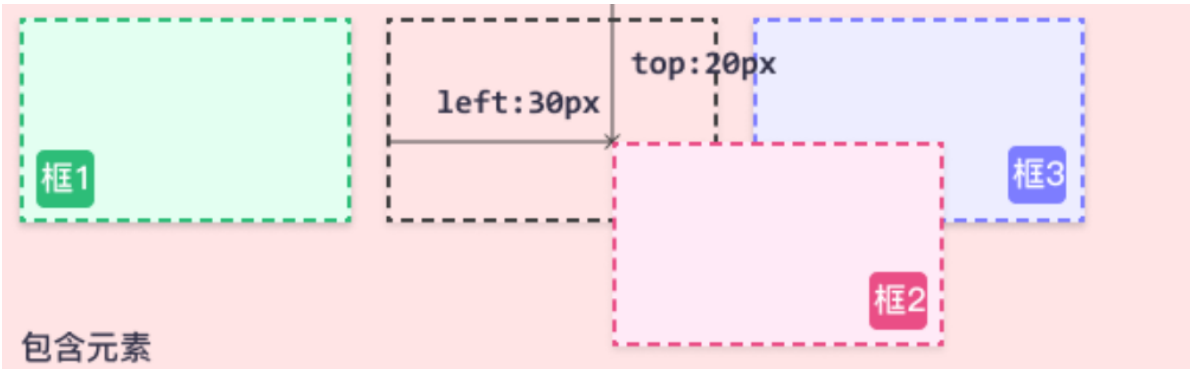
定位布局

> 后出现的定位层级 优先级高

相对定位

`position：relative；`

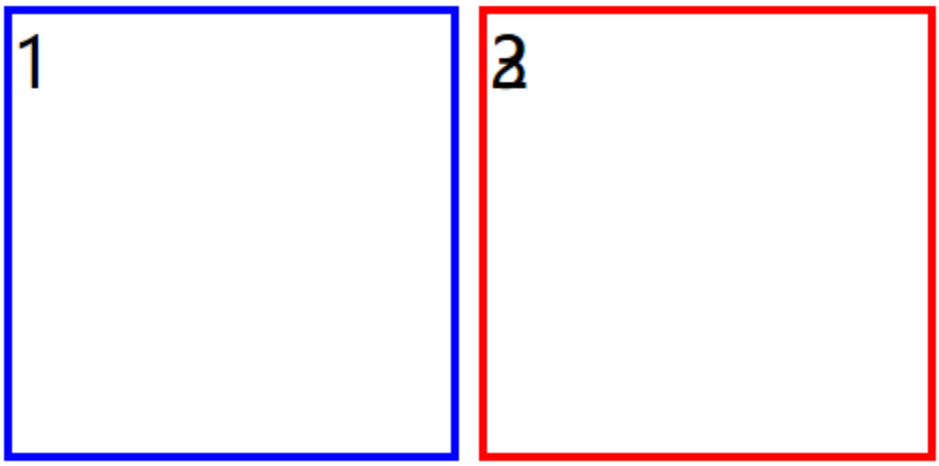
对一个元素使用相对定位，其会出现在其原始位置，然后通过top,bottom,left,right设置偏移距离，会按照原始位置进行相对偏移，但是 仍然会占据原有的空间



绝对定位

`position:absolute;`

如果不设置偏移距离，元素会在原始位置，但是该元素已经脱离了文档流（2、3已经重叠）



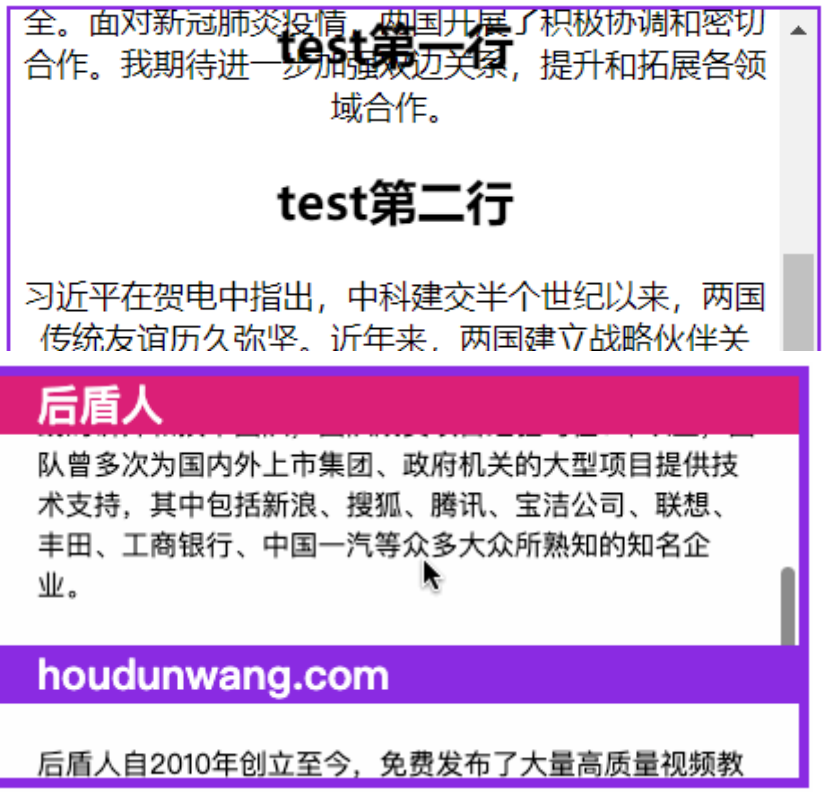
设置偏移后，元素会相当于其包含块（另一个最近的已定位的元素或初始包含元素）进行偏移。元素定位后会变成块级框，无论原来是什么类型。若包含块scroll,则绝对定位元素也会跟着scroll。

当元素不设置尺寸时，top|bottom|left|right会改变元素尺寸。

层次分级

粘性定位

- 设置了position: sticky的元素并不脱离文档流，仍然保留元素原本在文档流中的位置。
- 当元素在容器中被滚动超过指定的偏移值时，元素在容器内固定在指定位置。亦即如果你设置了top: 50px，那么在sticky元素到达距离相对定位的元素顶部50px的位置时固定，不再向上移动（相当于此时fixed定位）。
- 元素固定的相对偏移是相对于离它最近的具有滚动框的祖先元素，如果祖先元素都不可以滚动，那么是相对于viewport来计算元素的偏移量。
- 非同级粘性定位,不属于同一个父元素设置粘性定位时，后面的元素挤掉原来位置的元素，原来的元素会固定在父级元素的底部并跟随其一起向上滚动。
-



- 同级粘性定位会叠加

固定定位

相当于屏幕进行定位

弹性布局

弹性布局类型

```
display:flex 将对象作为弹性伸缩盒显示
display:inline-flex 将对象作为内联块级弹性伸缩盒显示
```

弹性布局方向

指定内部元素是如何在 flex 容器中布局的，定义了主轴的方向(正方向或反方向)。

flex-direction值	描述，从start至end排列
row (default)	水平，从左至右排列
row-reverse	水平，从右至左排列
column	垂直，从上到下排列
column-reverse	垂直，从下到上排列

行堆叠方向

指定flex 元素单行显示还是多行显示 。如果允许换行，这个属性允许你控制行的堆叠方向。

flex-wrap值	描述
nowrap(default)	flex 的元素被摆放到到一行，这可能导致溢出 flex 容器
wrap	flex 元素 被打断到多个行中，根据flex-direction的值的垂直方向的start至end换行
wrap-reverse	flex 元素 被打断到多个行中，根据flex-direction的值的垂直方向的end至start换行

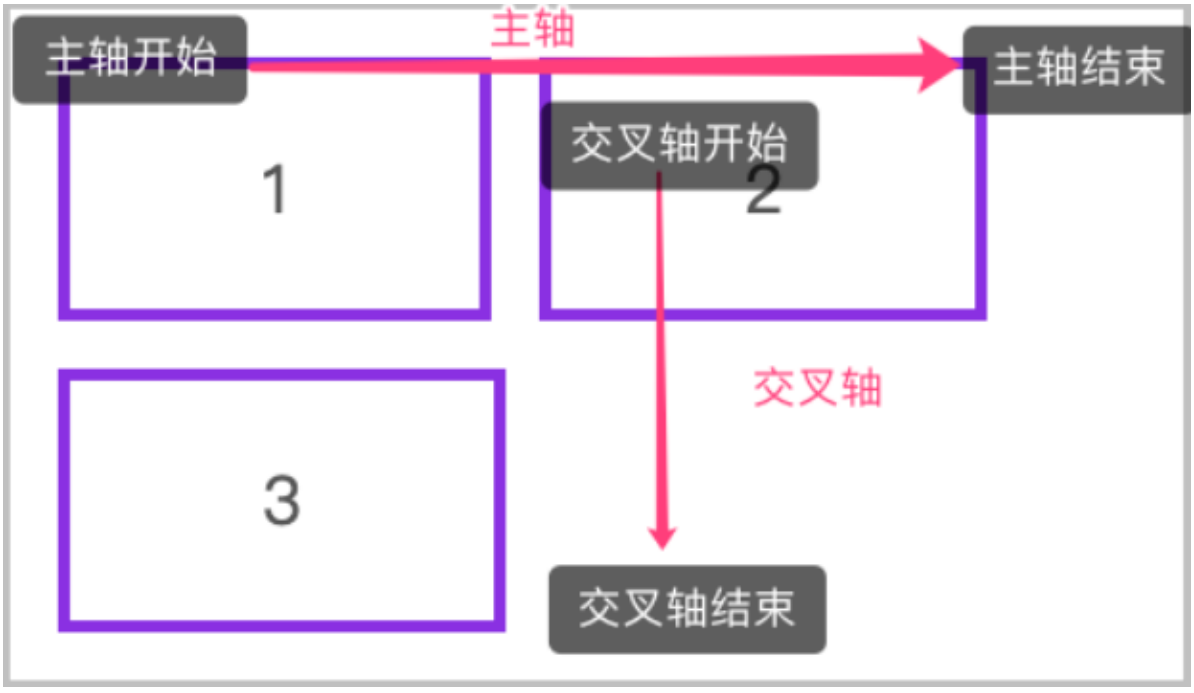
flex-flow

flex-direction 和 flex-wrap 的简写

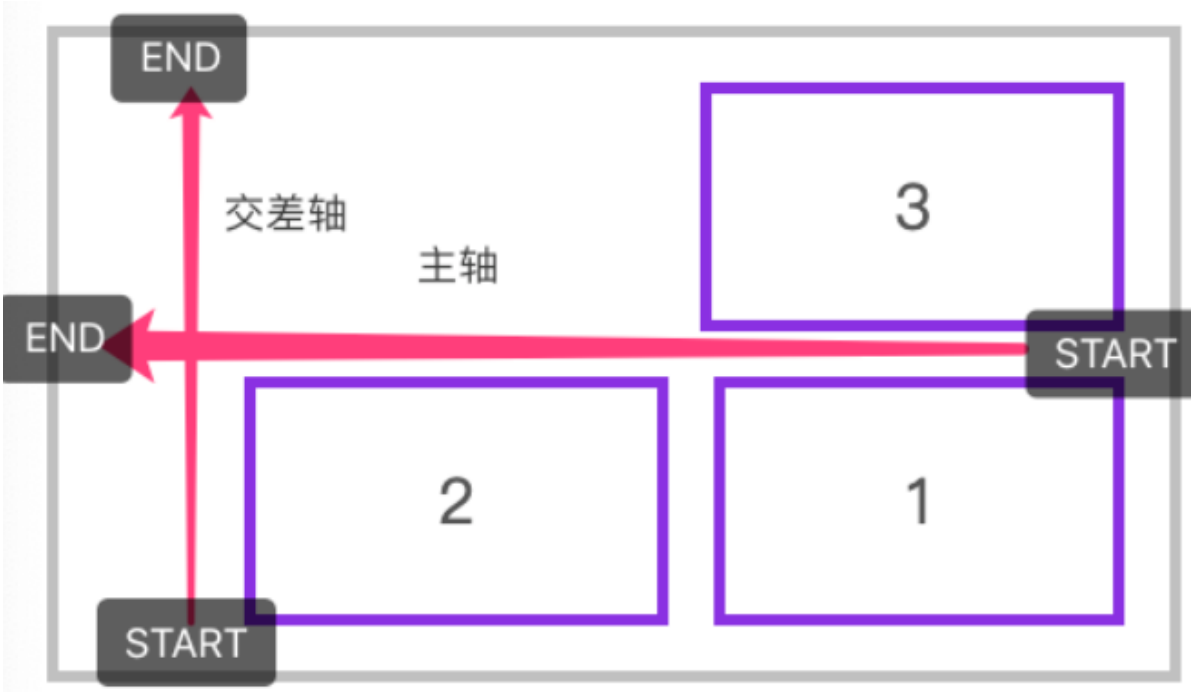
```
flex-flow: <'flex-direction'> | <'flex-wrap'>
```

轴说明

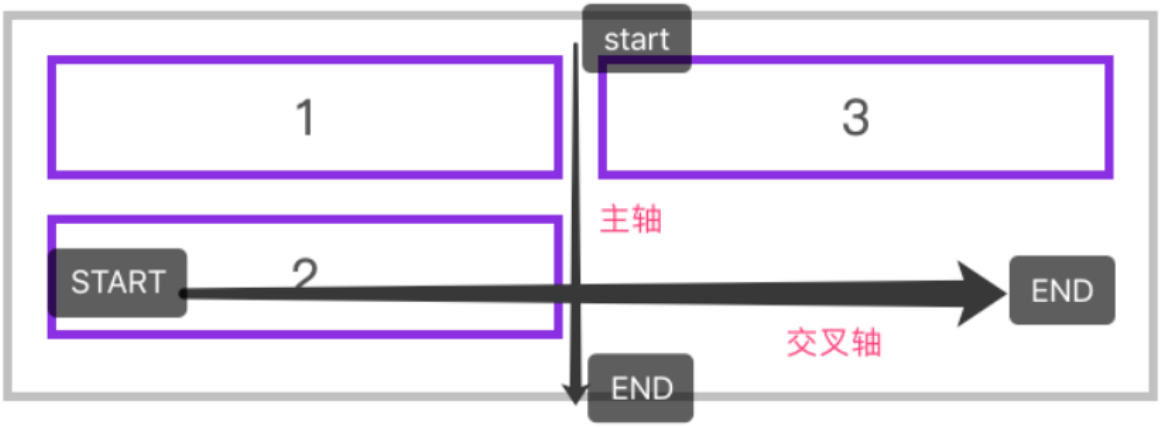
- **flex-flow: row wrap**



- **flex-flow: row-reverse wrap-reverse**



- **flex-flow: column wrap**



对齐方式

> 弹性容器

justify-content 用于设置或检索弹性盒子元素在主轴（方向）方向上的对齐方式

justify-content值	描述
flex-start(default)	位于容器开头
flex-end	位于容器结尾
center	位于容器中心
space-between	均匀排列，首元素在起点，尾元素在终点
space-around	均匀排列，每个元素周围分配相同空间，首元素左侧与尾元素右侧间隙是任意元素间间隔的一半
space-evenly	均匀排列，元素之间间隔相等
stretch	均匀排列，auto-sized元素会被拉伸

使用 align-content（多行效果） 属性对齐交叉轴上的各项（垂直）方向上的对齐方式

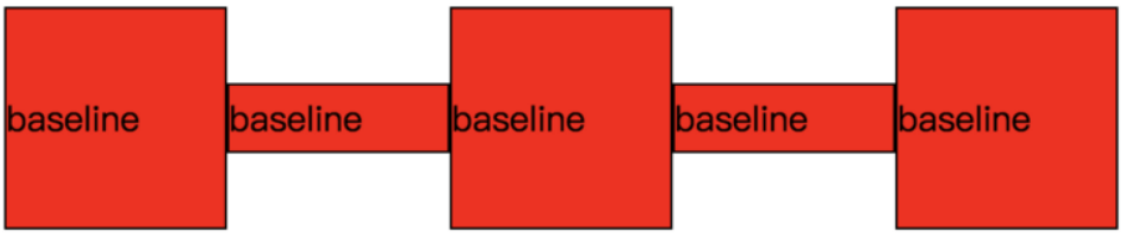
align-items（与align-content效果类似，同时还适用于单行，常用align-items）

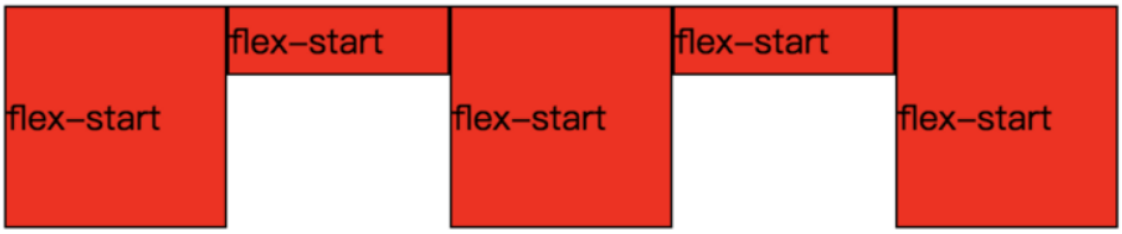
```
align-items:baseline;
```

比如多个容器文字行高不一致时。

baseline：始终按文字基线对齐。

flex-start：始终按 flex 容器起始位置对齐。



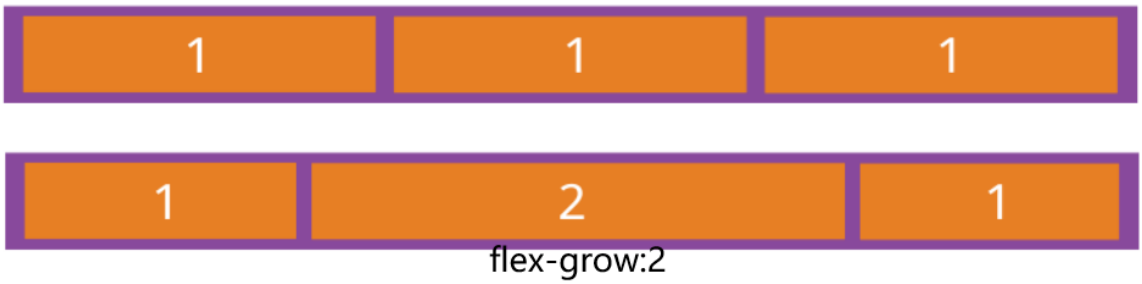


> 弹性元素

align-self可以为单独的弹性项目设置对齐，并会覆盖已有的align-items的值

flex-grow

指定了flex容器中剩余空间的多少应该分配给各个弹性元素，可以为小数，非负数，默认为0



flex-grow:2

flex-shrink

指定 flex 元素的收缩规则，flex 元素仅在默认宽度之和大于容器的时候才会发生收缩，其收缩的大小是依据 flex-shrink 的值，默认为1

举例

父元素 500px。三个子元素分别设置为 150px，200px，300px。

三个子元素的 flex-shrink 的值分别为 1，2，3。

首先，计算子元素溢出多少：150 + 200 + 300 - 500 = -150px。

那这 -150px 将由三个元素的分别收缩一定的量来弥补。

具体的计算方式为：每个元素收缩的权重为其 flex-shrink 乘以其宽度。

所以总权重为 1 * 150 + 2 * 200 + 3 * 300 = 1450

三个元素分别收缩：

- 150 * 1(flex-shrink) * 150(width) / 1450 = -15.5
- 150 * 2(flex-shrink) * 200(width) / 1450 = -41.4
- 150 * 3(flex-shrink) * 300(width) / 1450 = -93.1

三个元素的最终宽度分别为：

- 150 - 15.5 = 134.5
- 200 - 41.4 = 158.6
- 300 - 93.1 = 206.9

```
溢出总量 = 子元素宽度之和 - 父元素宽度

总权重 = X1宽度 * X1-flex-shrink + X2宽度 * X2-flex-shrink + X3宽度 * X3-flex-shrink
......

公式 X1的实际宽度 = X1预设宽度 - X1宽度 * X1-flex-shrink * 溢出总量 /总权重
```

flex-basis

指定了 flex 元素在主轴方向上的初始大小。如果不使用 `box-sizing` 改变盒模型的话，那么这个属性就决定了 flex 元素的内容盒（content-box）的尺寸。

> 优先级：flex-basis > max(min)-height(width) > width

弹性元素组合定义

可以使用一个，两个或三个值来指定 `flex` 属性。

单值语法: 值必须为以下其中之一:

- 一个无单位**数(`<number>`)**: 它会被当作`flex:<number> 1 0;` `<flex-shrink>`的值被假定为1，然后 `<flex-basis>` 的值被假定为`0`。
- 一个有效的**宽度(`width`)**值: 它会被当作 `<flex-basis>`的值。
- 关键字`none`， `auto`或`initial`.

双值语法: 第一个值必须为一个无单位数，并且它会被当作 `<flex-grow>` 的值。第二个值必须为以下之一:

- 一个无单位数：它会被当作 `<flex-shrink>` 的值。
- 一个有效的宽度值: 它会被当作 `<flex-basis>` 的值。

三值语法:

- 第一个值必须为一个无单位数，并且它会被当作 `<flex-grow>` 的值。
- 第二个值必须为一个无单位数，并且它会被当作 `<flex-shrink>` 的值。
- 第三个值必须为一个有效的宽度值， 并且它会被当作 `<flex-basis>` 的值。

弹性元素顺序

属性规定了弹性容器中的可伸缩项目在布局时的顺序。

元素按照 `order` 属性的值的增序进行布局，即由小到大，可为负数。

拥有相同 `order` 属性值的元素按照它们在源代码中出现的顺序进行布局。

栅格系统（区分栅格和栅格内的元素）

定义外部容器

```
display:grid;
```

划分行列

使用 `grid-template-rows` 划分行数， `grid-template-columns` 规则划分列数。

- 指定宽度

```
/*划分两行三列*/
grid-template-rows: 100px 100px;
grid-template-columns: 100px 100px 100px;
```

- 指定百分比

```
/*划分两行四列*/
grid-template-rows: 50% 50%;
grid-template-columns: 25% 25% 25% 25%;
```

- 重复设置

```
/*repeat()可以避免依次指定*/
/*设置两行四列，第1,3列100px宽度，第2,4列50px*/
grid-template-rows: repeat(2, 50%);
grid-template-columns: repeat(2, 100px 50px);
```

- 自动填充

```
/*按比例填充每个栅格100px高，100px宽*/
width: 300px;
height: 200px;
grid-template-rows: repeat(auto-fill, 100px);
grid-template-columns: repeat(auto-fill, 100px);
```

```
/*按比例填充第二个栅格*/
grid-template-rows: repeat(2, 1fr);
grid-template-columns: 20vw auto 30vw;
```

- 按比例划分

```
/*按比例三行三列*/
width: 300px;
height: 200px;
grid-template-rows: repeat(3, 1fr);
grid-template-columns: repeat(3, 1fr);
```

> fr 单位是一个自适应单位，fr单位被用于在一系列长度值中分配剩余空间，如果多个已指定了多个部分，则剩下的空间根据各自的数字按比例分配

- minmax

```
width: 300px;
height: 300px;
display: grid;
grid-template-rows: 100px minmax(100px, 1fr);
grid-template-columns: 100px 1fr;
```

- 组合(简写)定义

```
/*使用 minmax 设置行高取值范围在最小100px ~ 最大1fr*/
grid-template: repeat(3, 100px) / repeat(3, 100px);
```

栅格间距

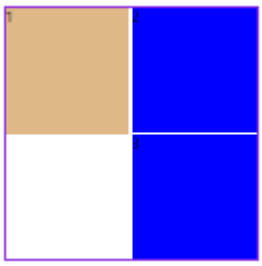
```
row-gap: 20px;
column-gap: 10px;
gap: 20px 10px;
```

```
gap: 0;
gap: 10%;
gap: 1em;
gap: 10px 20px;
gap: calc(20px + 10%);
```

> 多种设置方式

栅格系统内部元素定位

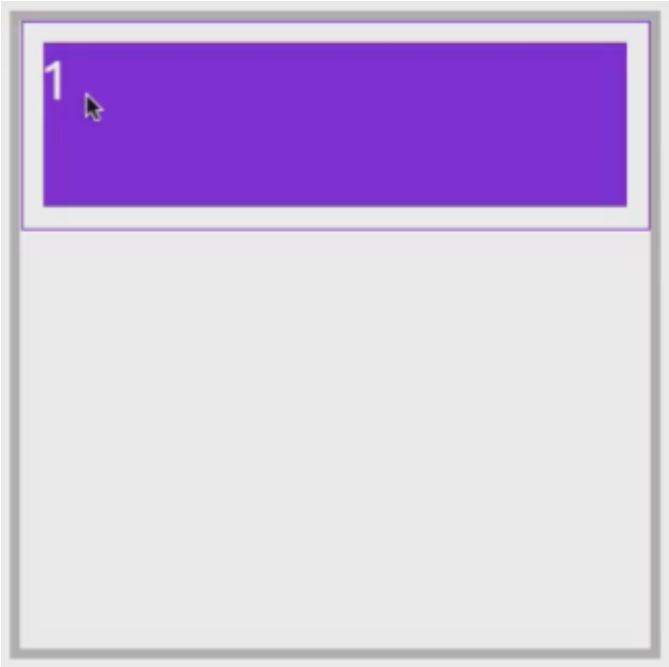
```
main div:nth-child(1){
  background: burlywood;
  grid-row:1/3;
  grid-column:1/2;
  height:150px;
}
```



- 栅格线

> 如果指定的是负数，则指的是从下（右）边界向上（左）边界计算的反向顺序

```
/*根据栅格线设置，指定栅格区域尺寸*/
grid-row-start: 1;
grid-column-start: 1;
grid-row-end: 2;
grid-column-end: 4;
```



- 栅格线命名

```
/*定义栅格线命名*/
grid-template-rows: [r1-start] 100px [r1-end r2-start] 100px [r2-end r3-
start] 100px [r3-end];
grid-template-columns: [c1-start] 100px [c1-end c2-start] 100px [c2-start
c3-start] 100px [c3-end];
/*使用栅格线命名*/
grid-row-start: r1-end;
grid-column-start: c2-start;
grid-row-end: r3-start;
grid-column-end: c3-start;
```

- 栅格线自动命名

```
/*重复设置的栅格线会自动命名*/
grid-template-rows: repeat(3, [r-start] 100px [r-end]);
grid-template-columns: repeat(3, [c-start] 100px [c-end]);
/*使用自动命名：命名+数字*/
grid-row-start: r-start 2;
grid-column-start: c-start 2;
grid-row-end: r-start 2;
grid-column-end: c-end 2;
```

- 偏移量

```
span(-1行不行？)
为网格单元定义一个跨度，使得网格单元的网格区域中的一条边界远离另一条边界线 n 条基线
```

grid-area

`grid-area`更加简洁是同时对 `grid-row` 与 `grid-column` 属性的组合声明。

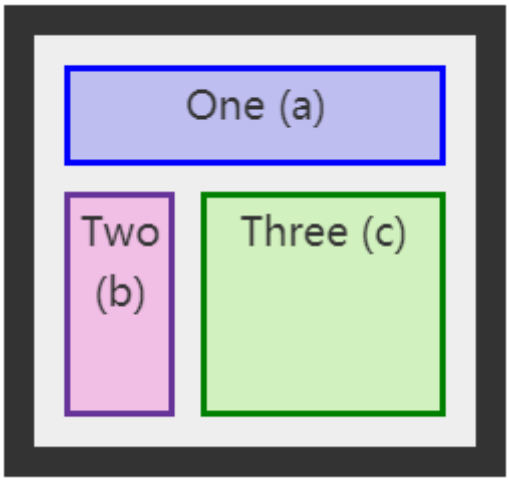
```
grid-area: grid-row-start/grid-column-start/grid-row-end/grid-column-end;
```

栅格区域

> 每一个给定的字符串会生成一行，一个字符串中用空格分隔的每一个单元(cell)会生成一列。**多个同名的，跨越相邻行或列**的单元称为网格区块(grid area)。非矩形的网格区块是无效的。

- **区域声明**

```
grid-template-rows: 80px 1fr 50px;
grid-template-columns: 100px 1fr 50px 60px;
grid-template-areas: "header header header header"
    "nav main main aside"
    "footer footer footer footer";
```




```
grid-template-areas:
            "a a a"
            "b c c"
            "b c c";
```

- **栅格中使用grid-area匹配指定区域**
- **简写**

```
grid-template:
     '栅格名称 栅格名称 栅格名称 栅格名称' 行高
     '栅格名称 栅格名称 栅格名称 栅格名称' 行高
     '栅格名称 栅格名称 栅格名称 栅格名称' 行高/列宽 列宽 列宽 列宽;
```

```
grid-template:
  'header header header header' 80px
  'nav main main aside' auto
  'footer footer footer footer' 50px/100px auto 50px 60px;
```

- **区域占位**

```
grid-template-areas: "top . ."
    "top . ."
    "bottom bottom bottom";
```

自动布局算法

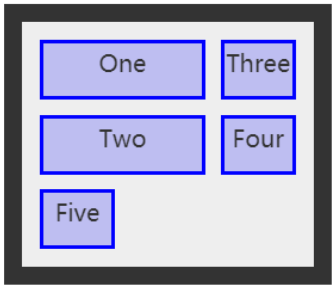
`**grid-auto-flow**` 控制着自动布局算法怎样运作，精确指定在网格中被自动布局的元素怎样排列

`dense` 该关键字指定自动布局算法使用一种“稠密”堆积算法，如果后面出现了稍小的元素，则会试图去填充网格中前面留下的空白。这样做会填上稍大元素留下的空白，但同时也可能导致原来出现的次序被打乱。

```
grid-auto-flow: row;
```

```
grid-auto-flow: column;
```

```
grid-auto-flow: row dense;
```



栅格对齐方式

justify-content与align-content用于控制栅格的对齐方式，比如在栅格的尺寸小于容器的心里时，控制栅格的布局方式。**参考flex布局中的相关属性介绍。**

- **组合简写**

place-content

用于控制栅格的对齐方式，语法如下：

```
place-content: <align-content> <justify-content>
```

place-items

控制所有元素的对齐方式，语法结构如下：

```
place-items: <align-items> <justify-items>
```

place-self

控制单个元素的对齐方式

```
place-self: <align-self> <justify-self>
```