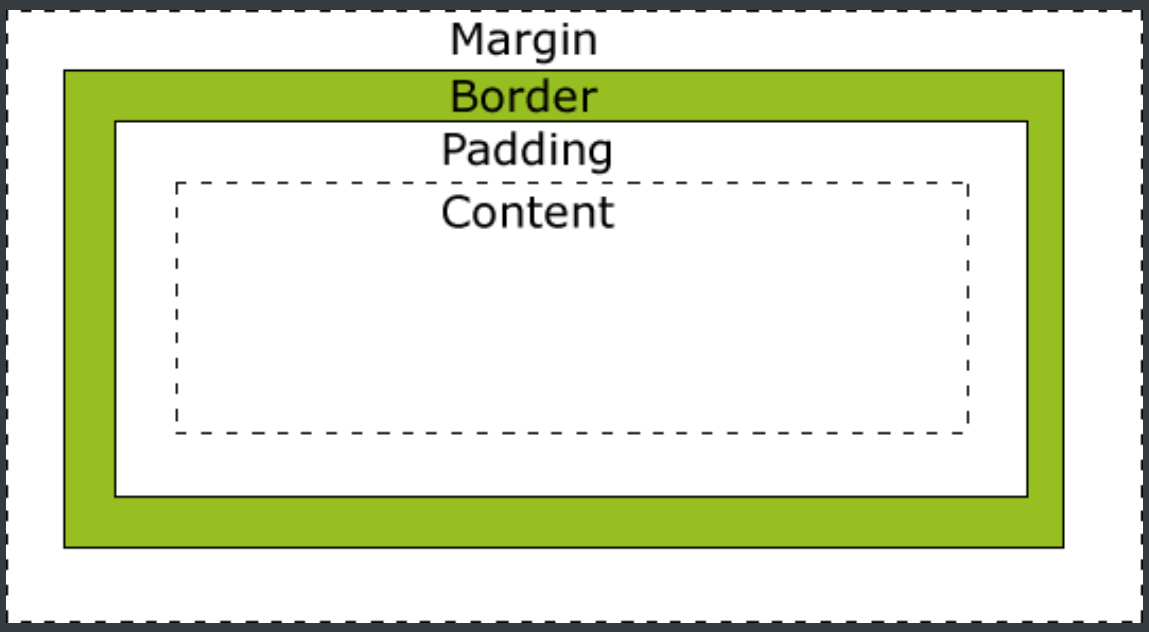
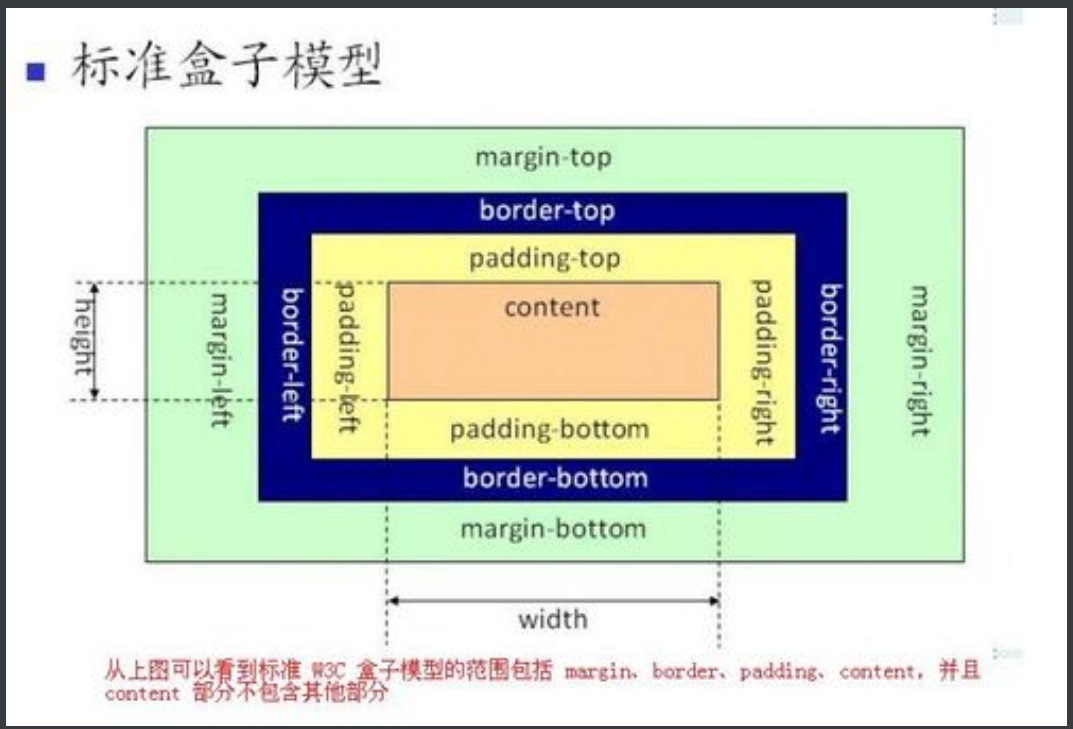
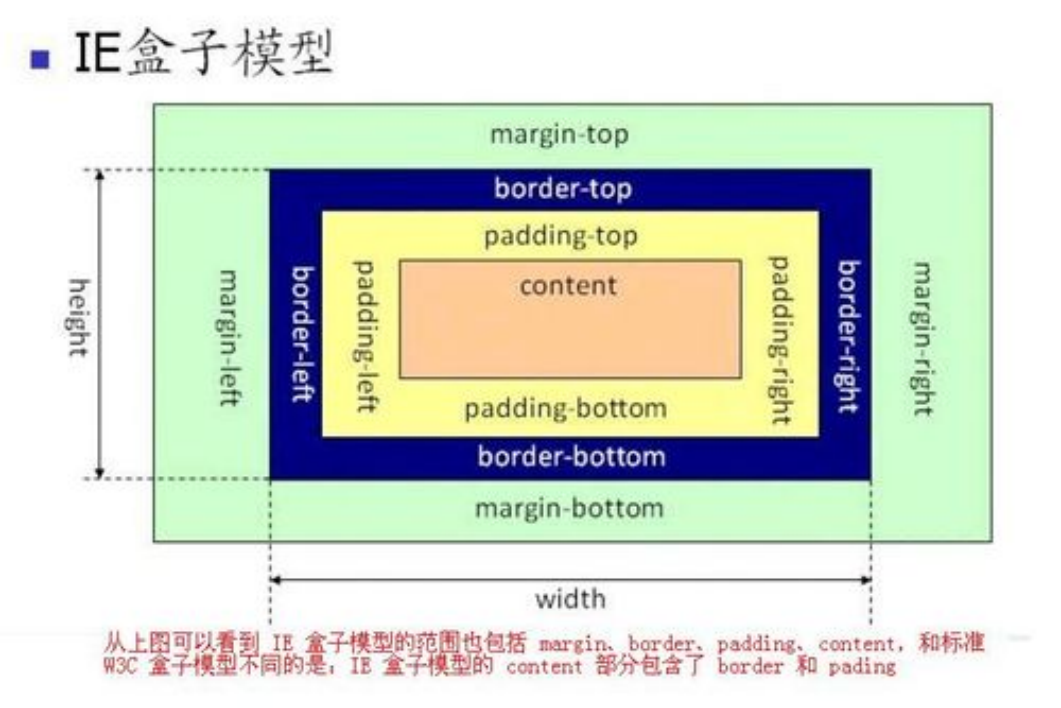
1.盒子模型



所有的html元素都可以看成是一个盒子模型， 一个盒子模型是由：content(内容)， padding（内边距），border（边框），margin（外边距）





标准盒子模型：width就是content，不包含padding 和border： width = content.width。

IE盒子模型： width包含了content， pading 和 border：width = content.width+ + padding + border；

box-sizing: 定义如何计算一个元素的宽度和高度

- border-box： 设置的宽度会包含padding 和 border，实际的内容的宽度： width - border - height； --> 即使加上 pading 和 border 也是一样的（加上 border + padding，内容是减少的）
- content-box：默认值；设置的元素的宽度高度仅仅是内容的宽度和高度。不包含 padding 和 border；(加上padding和border，content不会减少),

box-sizing的例子

2.css3的新属性

选择器:

伪类选择器

1. first-child: 父元素的第一个子元素

```
.parent :first-child {
    color:  red;
}
```

2. last-child：父元素的最后一个子元素
3. nth-child(n)： 父元素的第n个子元素

```
.parent :nth-child(2) {
    color:  red;
}
```

4. nth-last-child(n): 父元素倒数的第n个元素

```
.parent :nth-last-child(2) {
    color:  red;
}
```

属性选择器：选取标签带有某些属性的选择器(^开始位置，$ 结束位置, *任意位置)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="UTF-8">
    <style>
        /* 选取具有class开头是shu的元素 */
        .parent span[class^="shu"] {
            color: red;
```

```
            }
            /* 选取具有class结尾是Shu的元素 */
             .parent span[class$="Shu"] {
                color: yellow;
            }
            /* 选取具有class任意位置是shu的元素 */
            .parent span[class*="Shu"] {
                color: #970;
            }
            /* 选取具有type位置是text的input元素 */
           .parent input[type*="text"] {
                color: #970;
            }
        </style>
    <div
    <div class="parent">
        <span class="shuliqi1">1</span>
        <span class="shuliqi2">2</span>
        <span class="shuliqi3">3</span>
        <span class="indexShu">4</span>
        <span class="indexShu">5</span>
        <span class="indexShuhah">6</span>
        <span class="indexShuhah">7</span>
        <input type="text"/>
    </div>
    <body>
    </body>
    </html>
```

伪元素选择器---> 伪元素不是页面真正的元素，是css的展示样式

- first-letter：文本的第一个单词或者字
- first-line：文本的第一行
- selection：选中的文本
- before：在元素的开始位置创建一个元素， 该元素为行内元素，必须结合content 使用
- after：在元素的结束位置创建一个元素， 该元素为行内元素，必须结合content 使用

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title></title>
    <meta charset="UTF-8">
    <style>
        .parent {
            width: 100px;
        }
        /* 文本的第一个字 */
        .parent::first-letter{
            color: red;
        }
        /* 文本的第一行文字 */
        .parent::first-line{
            color:yellow;
        }
        /* 选中的文字的颜色 */
        .parent::selection{
            color:#085;
        }
        .parent::before {
            content: "开始";
        }
        .parent::after {
            content: "结束";
        }
    </style>
<div
<div class="parent">
   你好呀阿里卡时间段奥斯卡好大就爱上大德哈卡建设大街黄寺大街等哈哈吉收到货噶几是
</div>
<body>
</body>
</html>
```

> 伪类和伪元素的区别： https://www.jianshu.com/p/21eac04082d7。 伪类使用单冒号,而伪元素使用双冒号。

文字效果：

- **word-wrap**: 文字换行---->break-word， normal（浏览器默认处理）

  white-space，**控制空白字符的显示**，同时还能控制是否自动换行。它有五个值： `normal | nowrap | pre | pre-wrap | pre-line`

  word-break，**控制单词如何被拆分换行**。它有三个值： `normal | break-all | keep-all`

  word-wrap（overflow-wrap）**控制长度超过一行的单词是否被拆分换行**，是 `word-break` 的补充，它有两个值： `normal | break-word`

  彻底搞懂word-break、word-wrap、white-space

  超出省略一般这么写:

  ```
  overflow: hidden;
  white-space: nowrap;
  text-overflow: ellipsis;
  ```

- **text-overflow**：clip(修剪)，ellipsis(超出省略)
- **text-decoration**：
  - underline(文本下定义一条线)
  - overline(文本上定义一条线)
  - line-through(穿过文本的一条线)
  - blink(定义闪烁的文本)

- **text-shadow**（h-shadow, v-shadow, blur, color）
  - h-shadow： 水平阴影的位置
  - v-shadow: 垂直阴影的位置
  - blur: 模糊的距离
  - color: 阴影的颜色

```
/* 水平阴影的位置，垂直阴影的位置，模糊的距离，阴影的颜色 */
text-shadow: 10px 10px 5px red;
```

动画

- **transition**：过度效果(从一个样式变为另外一个样式添加效果)

```
transition: property duration timing-function delay;
```

  - property: 规定设置过渡效果的 CSS 属性的名称。
  - duration： 过度时间
  - timing-function： 过渡曲线
  - delay： 延迟多少执行过度效果

```
.box {
  width: 200px;
  transition:width 2s ease-in 2s
}
.box: hover {
  width: 400px;
}
```

- **transform** 2d 或者3d 变换

  translate --> 平面

  rotate(32deg) ---> 旋转

  scale() ---> 缩放

  skew----> 翻转

```
/* 平面位移 */
transform: translate(100px, 100px);

/* 旋转20度 */
transform: rotate(20deg);

/* 沿着x轴翻转30度，y轴翻转20度 */
transform: skew(30deg, 20deg);

transform: scale(2)
```

- **animation**(动画函数，动画时间，动画曲线，延迟时间， 动画次数(n)，执行的方向
  - 动画曲线：linear（匀速）， ease（低速开始，然后加快，然后放慢结束），ease-in（低速开始），ease-out低速结束）， ease-in-out(低速开始和结束)
  - 是否反方向： normal(默认值)； reverse(反方向)， alternate(先正后反，交替)，alternate-reverse(先反后正，交替执行)

```
  @keyframes move {
      0% { width: 100px }
      50% { width: 300px }
      100% { width: 500px }
  }
  .parent {
      height: 10px;
      width: 10px;
      background: red;
      /* 执行的动画函数名字 执行的时间 执行的曲线 延迟的时间 执行的次数(infinite无限)， 是否反方向（alernate反防线） */
      animation: move 5s linear 2s infinite alternate;
  }
```

Animation和transition大部分属性是相同的，他们都是随时间改变元素的属性值，他们的主要区别是transition需要触发一个事件才能改变属性，而animation不需要触发任何事件的情况下才会随时间改变属性值，并且transition为2帧，从from .... to，而animation可以一帧一帧的。

3.选择器及其优先级

!imprtant > 内联样式 > id > class（类）> 元素 > 通配符

各选择器的权重：

内联样式： 1000 id： 0100， class： 0010， 其他的都是0000

选择用哪个css, 看权重的大小，权重大的优先使用， 如果权重相等， 那么按顺序来看， 在后面的回覆盖前面的。

4.BFC

块格式化上下文 , 一个独立的渲染区域

BFC的触发

- 跟元素（html）
- float 不为 none
- position 为 fixed 或者 absolute
- Overflow 不为visible（（超出的不会修剪））
- Display 为 inline-block，flex, table-ceil等

BFC的规则

- 内部的box 垂直排列， 一个接着一个的放
- 垂直方向的距离由margin 决定,同一个bfc的box margi内会重叠，即使存在浮动元素也是如此
- 盒子的margin-left 会与其父级的盒子的border-left相接触，
- BFC区域的元素不会与浮动的元素重叠在一起
- b f c 计算高度的时候， 浮动元素的高度也需要计算进去
- Bfs 是一个独立的渲染空间，不会影响到外部

BFC的应用

- 防止margin重叠

  防止margin重叠

  由于b f cd的第二条规则： 会导致同一个bfc的margin重叠。但是不同的bfc 不会重叠，所以解决的办法就是另一个box 用一个b f c 包起来。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>防止margin重叠</title>
</head>
<style>
    *{
        margin: 0;
        padding: 0;
    }
    p {
        color: #f55;
        background: yellow;
        width: 200px;
        line-height: 100px;
        text-align:center;
        margin: 30px;
    }
</style>
<body>
    <p>●看看我的 margin是多少</p>
  <p>●看看我的 margin是多少</p>
</body>
```

```
    </html>
```

效果是这样的

如果我们不想想margin 重叠， 可以设置一个bFC

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>防止margin重叠</title>
</head>
<style>
    * {
        margin: 0;
        padding: 0;
    }

    p {
        color: #f55;
        background: yellow;
        width: 200px;
        line-height: 100px;
        text-align: center;
        margin: 30px;
    }

    .shu {
```

```
            overflow: hidden;
        }
    </style>

    <body>
        <p>●看看我的 margin是多少</p>
        <div class="shu">
            <p>●看看我的 margin是多少</p>
        </div>
    </body>

    </html>
```

效果如图

- 两栏自适应布局

  由于b f c的第二条规则,

  b f c 里面有一规定： 就是b f c 里面的元素的左边都紧紧挨着b f c的左边， 即使是浮动的元素。

  所以在左左边固定并且float： left， 右边自适应的时候， 右边的也是紧挨着b f c的左边。所以不能实现，但是b f c 里面也还有一条，bfc 不会与浮动元素重叠。 所以我们只需要把右右边的box设置成b f c 就可以了。

  左边固定， 右边自适应的例子

  ```
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
    scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
  ```

```
    <title>Document</title>
</head>
<style>
    *{
        margin: 0;
        padding: 0;
    }
    body {
        width: 100%;
        position: relative;
    }

    .left {
        width: 100px;
        height: 150px;
        float: left;
        background: rgb(139, 214, 78);
        text-align: center;
        line-height: 150px;
        font-size: 20px;
    }

    .right {
        height: 300px;
        background: rgb(170, 54, 236);
        text-align: center;
        line-height: 300px;
        font-size: 40px;
    }
</style>
<body>
    <div class="left">LEFT</div>
  <div class="right">RIGHT</div>
</body>
</html>
```

效果如图

又因为第三条规则。b f c 不会与浮动元素重叠

所以我们让right单独成为一个BFC

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<style>
    *{
        margin: 0;
        padding: 0;
    }
    body {
        width: 100%;
        position: relative;
    }

    .left {
        width: 100px;
        height: 150px;
        float: left;
        background: rgb(139, 214, 78);
        text-align: center;
        line-height: 150px;
        font-size: 20px;
```

```
    }

    .right {
        overflow: hidden;
        height: 300px;
        background: rgb(170, 54, 236);
        text-align: center;
        line-height: 300px;
        font-size: 40px;
    }
</style>
<body>
    <div class="left">LEFT</div>
    <div class="right">RIGHT</div>
</body>
</html>
```

- 清除浮动

  当一个父级不设置高度， 里面的元素有浮动， 那么就有父级高度塌陷的结果. 但是如果这个父级一个b f c。 bfc 里面有一条规则就是 算父级的高度， 需要把浮动元素的高度算上

  所以解决的办法就是： 给父级设置为一个bfc。

  清除浮动的方法

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>清除浮动</title>
</head>
<style>
    .par {
        border: 5px solid rgb(91, 243, 30);
        width: 300px;
    }

    .child {
        border: 5px solid rgb(233, 250, 84);
        width:100px;
        height: 100px;
        float: left;
    }
</style>
<body>
    <div class="par">
        <div class="child"></div>
        <div class="child"></div>
    </div>
</body>
</html>
```

由于 b f c 计算高度的时候， 浮动元素的高度也需要计算进去

所以我们可以设置父级的元素为一个bfc

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>清除浮动</title>
</head>
<style>
    .par {
        border: 5px solid rgb(91, 243, 30);
        width: 300px;
        overflow: hidden;
    }

    .child {
        border: 5px solid rgb(233, 250, 84);
        width:100px;
        height: 100px;
        float: left;
    }
</style>
<body>
    <div class="par">
        <div class="child"></div>
        <div class="child"></div>
  </div>
</body>
</html>
```

5.块级元素

块级元素一般为结构性标记

H1 ~ h6, div, form, ul, table, p等

块级元素的特点

- 块级元素总是从新的一行开始
- 块级元素的宽高，都是可控的
- 块级元素的默认宽高都是100%
- 块级元素可以包含块级元素，行内元素

6.行内元素

行内元素一般是描述性标签

span img， a, br, input, select等

行内元素的特点

- 行内元素与其他元素在同一行
- 行内元素宽高是不可控的
- 行内元素的宽就是内容的宽度，高就是内容的高度
- 行内元素 不可以包含块级元素

7. position的值

一、position 属性的作用

`position` 属性用来指定一个元素在网页上的位置，一共有5种定位方式，即 `position` 属性主要有五个值。

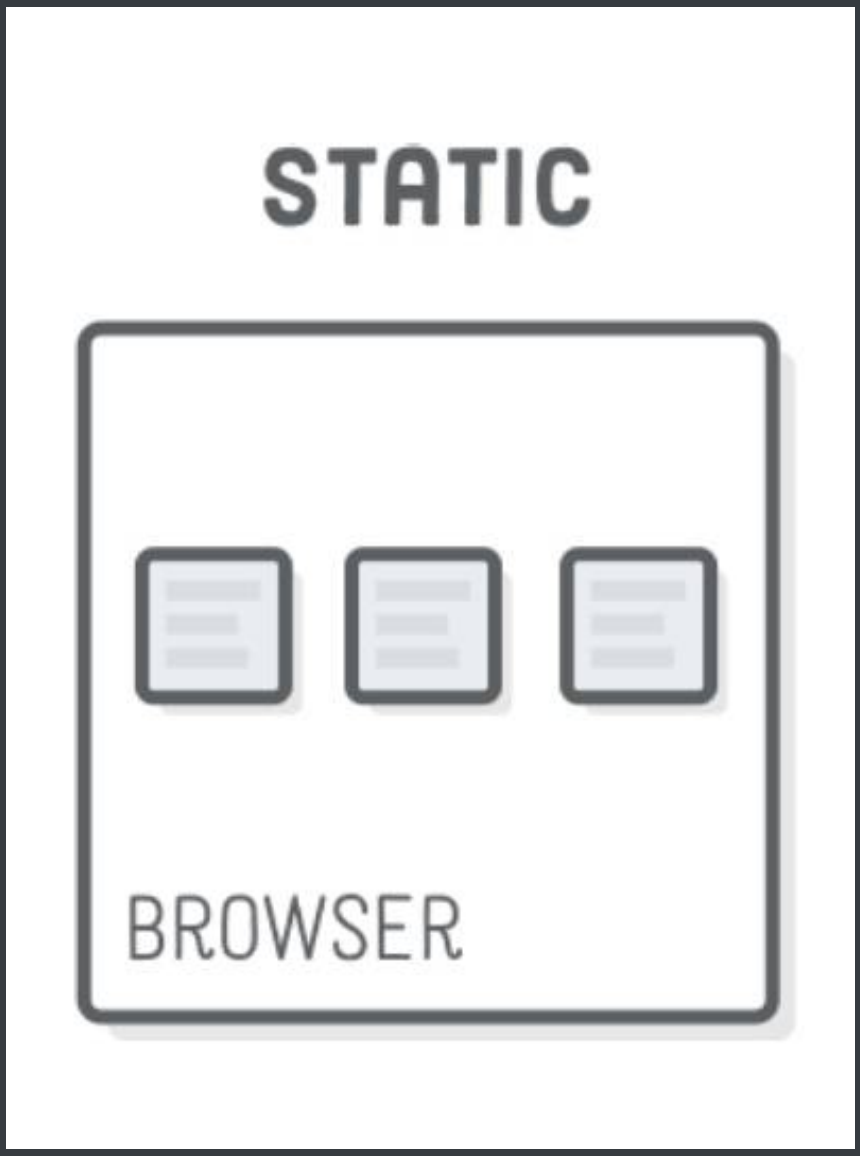
> - `static`
> - `relative`
> - `fixed`
> - `absolute`
> - `sticky`

下面就依次介绍这五个值。最后一个 `sticky` 是2017年浏览器才支持的，本文将重点介绍。

二、static 属性值

`static` 是 `position` 属性的默认值。如果省略 `position` 属性，浏览器就认为该元素是 `static` 定位。

这时，浏览器会按照源码的顺序，决定每个元素的位置，这称为"正常的页面流"（normal flow）。每个块级元素占据自己的区块（block），元素与元素之间不产生重叠，这个位置就是元素的默认位置。



注意，`static` 定位所导致的元素位置，是浏览器自主决定的，所以这时 `top`、`bottom`、`left`、`right` 这四个属性无效。

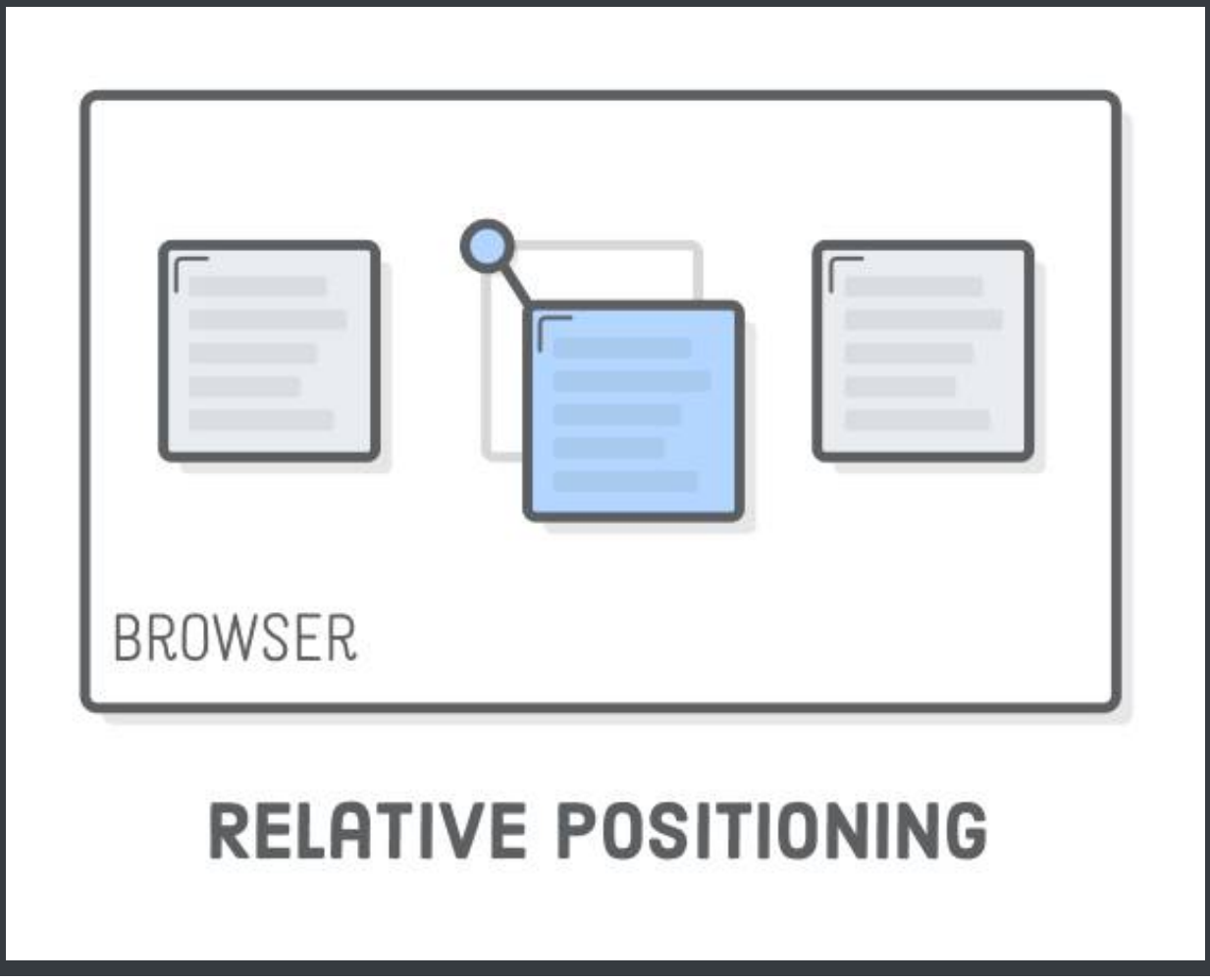
三、relative，absolute，fixed

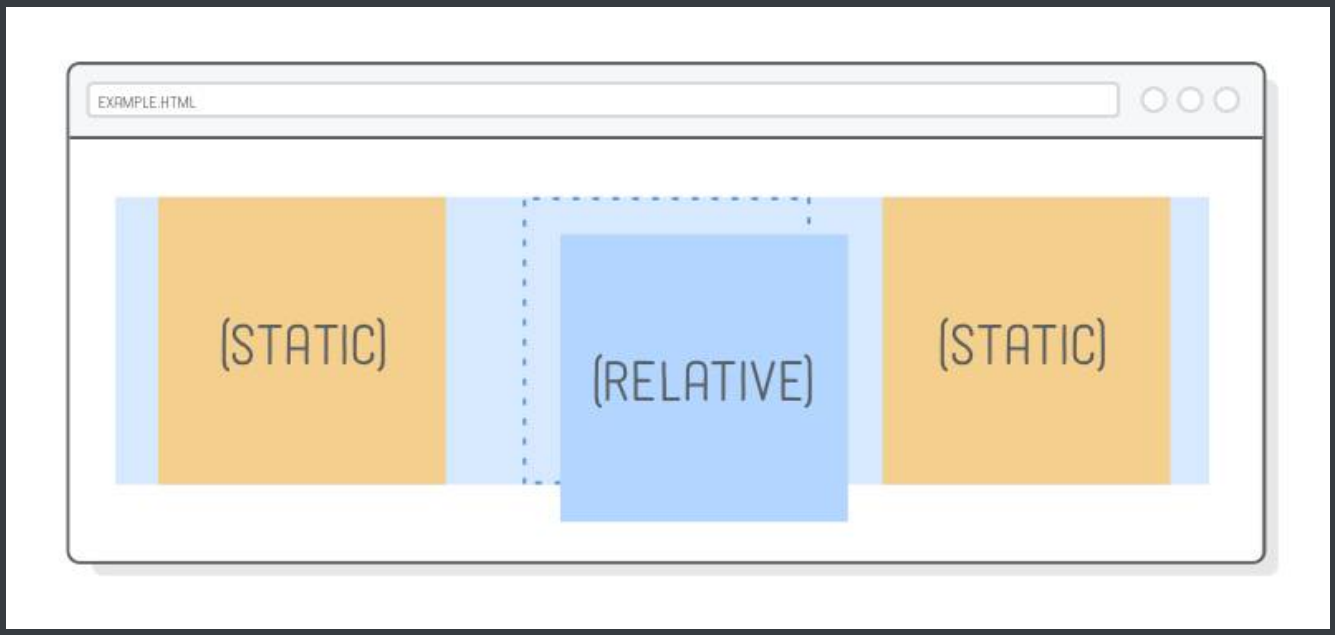
`relative`、`absolute`、`fixed` 这三个属性值有一个共同点，都是相对于某个基点的定位，不同之处仅仅在于基点不同。所以，只要理解了它们的基点是什么，就很容易掌握这三个属性值。

这三种定位都不会对其他元素的位置产生影响，因此元素之间可能产生重叠。

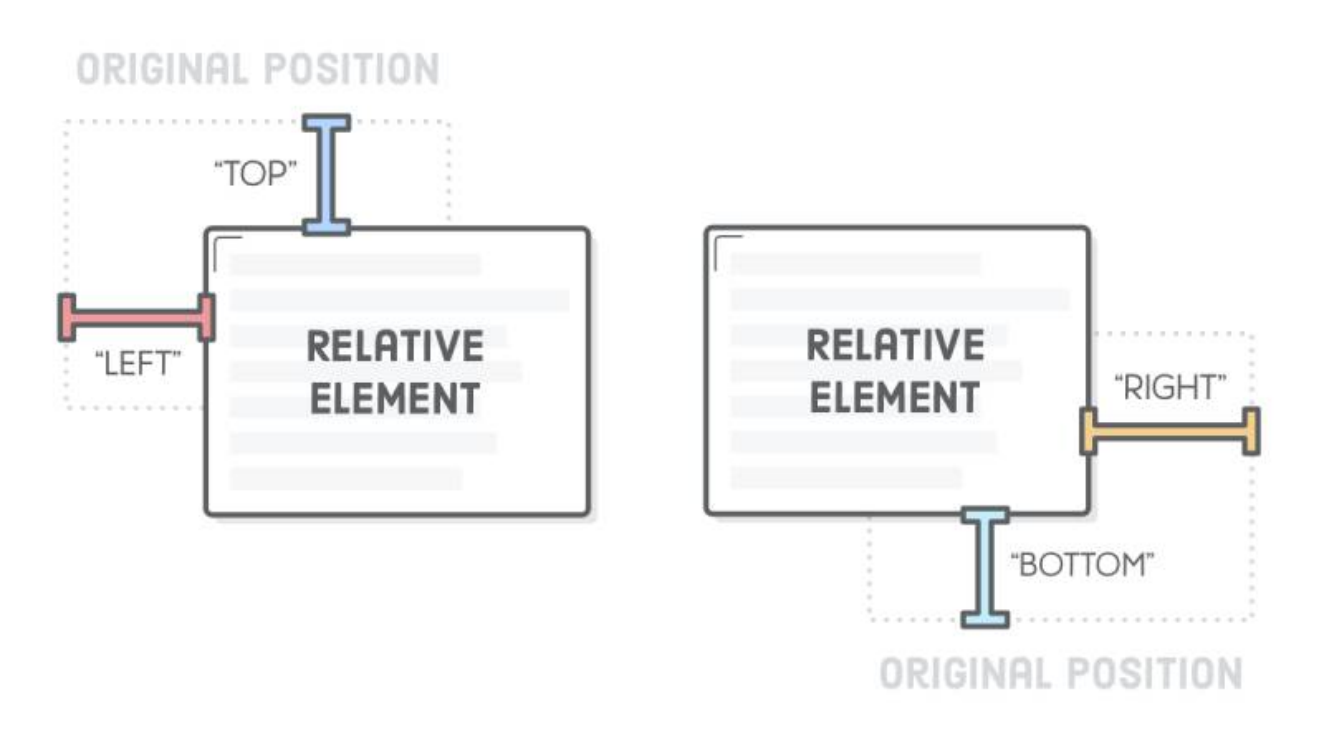
3.1 relative 属性值

`relative` 表示，相对于默认位置（即 `static` 时的位置）进行偏移，即定位基点是元素的默认位置。





它必须搭配 `top` 、 `bottom` 、 `left` 、 `right` 这四个属性一起使用，用来指定偏移的方向和距离。



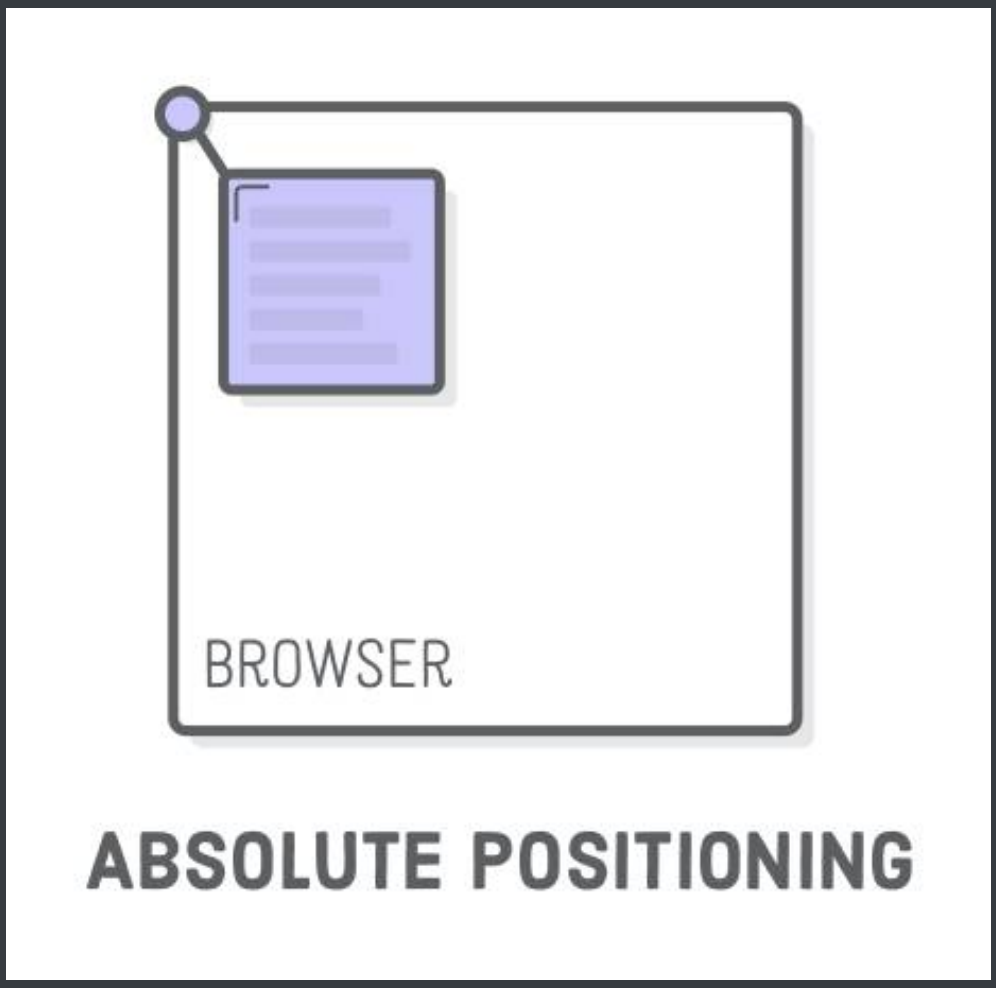

```
div {
position: relative;
top: 20px;
}
```

上面代码中， `div` 元素从默认位置向下偏移 `20px` （即距离顶部 `20px` ）。

3.2 absolute 属性值

`absolute` 表示，相对于上级元素（一般是父元素）进行偏移，即定位基点是父元素。

它有一个重要的限制条件：定位基点（一般是父元素）不能是 `static` 定位，否则定位基点就会变成整个网页的根元素 `html` 。另外， `absolute` 定位也必须搭配 `top` 、 `bottom` 、 `left` 、 `right` 这四个属性一起使用。



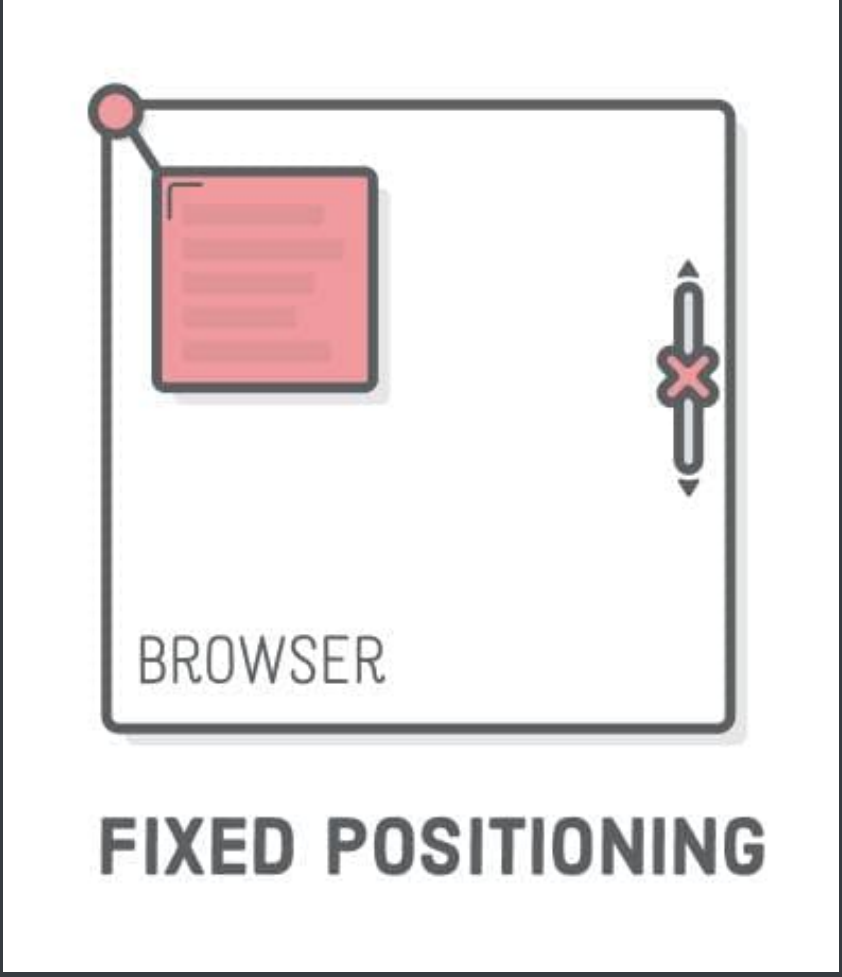

```
/*
HTML 代码如下
<div id="father">
 <div id="son"></div>
</div>
*/

#father {
positon: relative;
}
#son {
position: absolute;
top: 20px;
}
```

上面代码中，父元素是 relative 定位，子元素是 absolute 定位，所以子元素的定位基点是父元素，相对于父元素的顶部向下偏移 20px 。如果父元素是 static 定位，上例的子元素就是距离网页的顶部向下偏移 20px 。

注意， absolute 定位的元素会被"正常页面流"忽略，即在"正常页面流"中，该元素所占空间为零，周边元素不受影响。

3.3 fixed 属性值

fixed 表示，相对于视口（viewport，浏览器窗口）进行偏移，即定位基点是浏览器窗口。这会导致元素的位置不随页面滚动而变化，好像固定在网页上一样。



它如果搭配 top 、 bottom 、 left 、 right 这四个属性一起使用，表示元素的初始位置是基于视口计算的，否则初始位置就是元素的默认位置。

```
div {
position: fixed;
top: 0;
}
```

上面代码中， `div` 元素始终在视口顶部，不随网页滚动而变化。

8. flex 布局

f le x布局

设置为f le x 布局

```
.div {
    display: flex;
}

// 行内元素
span {
display: inline-block;
}
```

也就是说在一个连接里面， 可以发多个请求。

- flex-direction：设置主轴

  row，row- reverse， column， column-reverse
- flex-wrap： 当空间容不下时

  nowrap， wrap，wrap-reverse
- flex-flow：flex-direction flex-flow（row nowrap）
- Justify-content：主轴的对齐返回式

  flex-start, flex-end, center, space-bettween, space-around;
- align-items: 设置交叉轴的对齐方式

  flex-start， flex-end，center， baseline，stretch
- aligin-content：flex-start，lex-end，center， stretch， space-between， space-around

align-items 是单轴， 多轴都是有效的，aligin-content 是多轴有效， 如果当前布局是单轴： align-items ， aligin-content 同时存在， 那么align-items 有效。 如果是多轴，align-items ， aligin-content 同时存在， 那么aligin-content有效

项目的属性

- order： 项目排列的顺序， 数值越小越在前面
- flex-grow： 项目的放大属性，默认值为0， 即 存在剩余空间 也不放大
- flex-shrink： 项目的缩小，默认值是1 ， 如果空间不足， 则虽小
- flex-basis： 浏览器计算剩余空间时， 项目占主轴的空间， 默认值是auto 按照元素自己的大小
- flex： flex-grow flex-shrink：flex-basis（0 1 auto）
- aligin-self： 单独设置某个项目：

  flex-start，flex-end， center， baseline，center， stretch

9.水平垂直居中

垂直居中

- 块级元素 已知道宽高 + position

```
.div {
  position: absolute;
  width: 100px;
  height: 100px;
  background: yellow;
  left: 50%;
  margin-left: -50px;
  top: 50%;
  margin-top: -50px
}
```

- 块级元素 行内元素都有用：flex布局

```
.con {
  width: 100%;
  height: 100vh;
  display: flex;
  justify-content; center;
  align-items: center;
}
.div {
  width: 100%;
  height: 100px;
  background: red;
}
```

- 未知宽高

```
.con {
  width: 100%;
  height: 100vh;
}
.div {
  position: absolute;
  width: 100px;
  height: 100px;
  background: red;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);
}
```

- top/left/bottom/right=0, 绝对定位

```
.div {
  width:100px;
  height: 100px;
  backgrpund: red;
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom:0;
  margin:auto;
}
```

10.水平居中

- 行内元素

```
text-align: center
```

- 块级元素

```
margin: 0. auto
```

- Flex 布局

```
display: flex
justify-content:center
```

- 绝对定位定宽

```
{
  position: absolute;
  width: 100px;
  left: 50%;
  margin-left: -50px;
}
```

- 绝对定位不定宽

```
{
  position: absolute;
  left: 50%;
  transform: translate(-50%, 0)
}
```

- Left/right：0

```
{
  position: absolute;
  width: 200px;
  left: 0;
  right: 0;
  margin: 0 auto;
}
```

11. 垂直居中

- 行内元素

```
.parent {
  height: 200px;
}
.child {
  line-height: 200px;
}
```

- 块级元素

```
.parent {
  display: table;
}
.child {
    display: table-ceil;
    vertical-aligin: middle
}
```

- 绝对定位定宽

```
{
  position: absolute;
  height: 100px;
  top: 50%;
  margin-top: -50px;
}
```

- Flex 布局

```
{
  display: flex;
  aligin-items: center
}
```

- 绝对定位不定高

```
{
  position: absolute;
  top: 50%;
  transform: translate(0, -50%);
}
```

- Button/top： 0

```
{
  positoon: absolute;
  botton: 0;
  top: 0;
  margin: auto 0;
}
```

15 em, rem, px

px 绝对的像素

em: 相对于父级的像素font-size， 浏览器默认fot-size：16px 1em = 16px;

rem: 相对于根元素的font-size 浏览器默认fot-size：16px 1rem = 16px

16.用css实现一个宽高恒为父容器宽度一半的正方形，父容器宽高均不固定

17. position

- absolute： 生成绝对定位，以非static定位的第一个元素进行定位的， left，top
- fixed： 生成绝对定位，按照浏览器窗口来定位的 left，top
- relative： 生成相对定位，根据正常的定位的来定位的 left，top
- static： 没有定位， 以正常的流来布局

19. 左边固定， 右边自适应

float实现
绝对定位

b f c实现

flex