第三章：CSS布局-沈嘉杰

https://live.bytedance.com/9715/3359841

- 布局
  - 确定内容的大小和位置的算法
  - 依据元素，容器，兄弟节点和内容等信息来计算
- 常规流
  - 根元素，浮动，绝对定位的元素会脱离常规流
  - 盒子依据特定的**排版上下文**进行布局
  - 行级排版上下文
    - Inline Formatting Context (IFC)
    - 只包含行级盒子的容器会创建一个IFC
    - IFC 内的排版规则
      - 盒子在一行内水平摆放
      - 一行放不下时，换行显示
      - text-align 决定一行内盒子的水平对齐
      - vertical-align 决定一个盒子在行内的垂直对齐
      - 避开浮动(float)元素*
  - 块级排版上下文

- Block Formatting Context (BFC)
- 某些容器会创建一个BFC
  - 根元素
  - 浮动、绝对定位、inline-block
  - Flex子项和Grid子项
  - overflow 值不是 visible 的块盒
  - 绝对定位内部的元素流的排放会创建一个新的块级排版

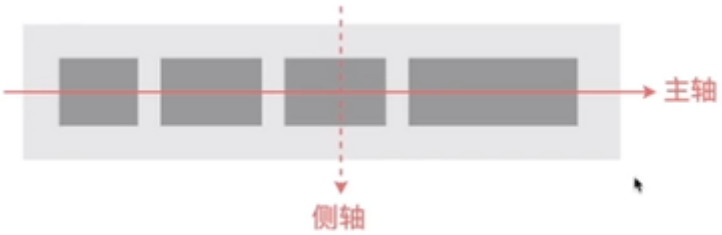
BFC 内的排版规则

- 盒子从上到下摆放
- 垂直 margin 合并
- BFC 内盒子的margin不会与外面的合并
- BFC 不会和浮动元素重叠
  - 当行级盒子中插入一个块级盒子时，实际上会把被块级盒子分隔的两部分都用一个匿名的盒子进行包裹，实际上形成一个BFC

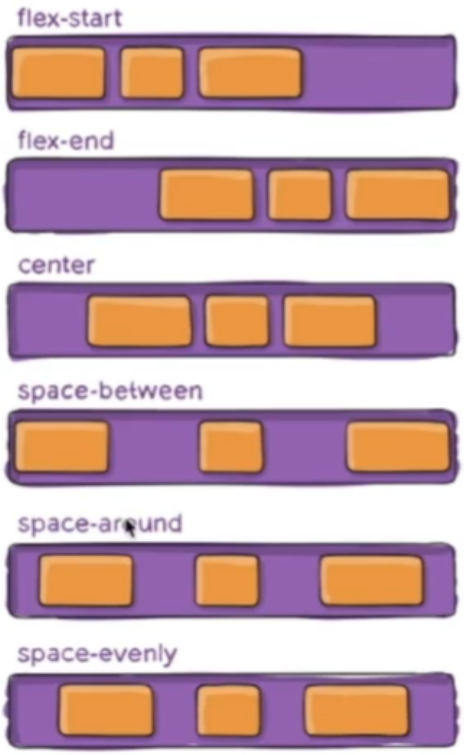
Flexible Box

- 它可以控制子级盒子的:
  - 摆放的流向 (→ ← ↑ ↓)
  - 摆放顺序
  - 盒子宽度和高度
  - 水平和垂直方向的对齐
  - 是否允许折行
  - `display: flex`: 生成一个flex容器(flex上下文)，但是这个容器**整体是块级的**
  - `display: inline-flex`： 生成一个flex容器(flex上下文)，但是这个容器**整体是行级的**
  - `flex-direction`:可以控制容器内的容器排版方向

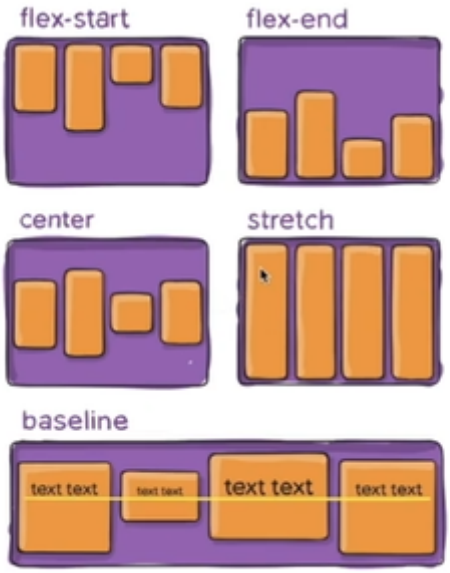
    - `row/column` `row-reverse/column-reverse` 有四种排布方向
- `flex-wrap` 控制元素是否换行
    - `nowrap` 不换行
    - `wrap` 换行
- `flex-grow` 当容器内有**剩余空间**时的**伸展能力**，越大则占的空间越多，默认0
    - `flex-grow: 1` 代表在**剩余空间中**占所有元素 `1/sum(flex-grow)` 份
- `flex-shrink` 当容器内有**空间不足**时的**收缩的容易程度**，默认为1，越大则占的空间越小
    - 为0时代表无法收缩，可能出现溢出的情况
- `flex-basis` 代表基准宽度，会覆盖掉 `width` 属性
    - `flex-basis:content` 基准宽度为内容宽度
- `flex: flex-wrap flex-grow flex-basis` 综合写法
- 主轴和侧轴



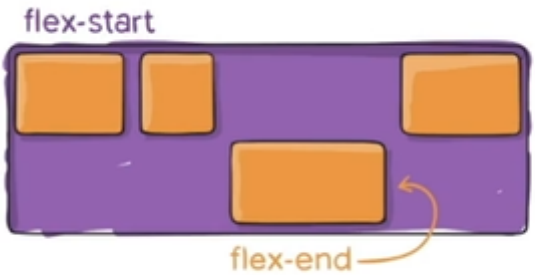
    - `justify-content` 对齐方式



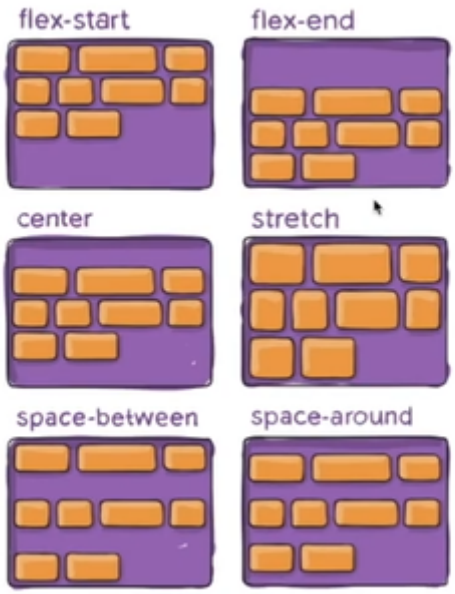
- `align-items` 侧轴的对齐方式



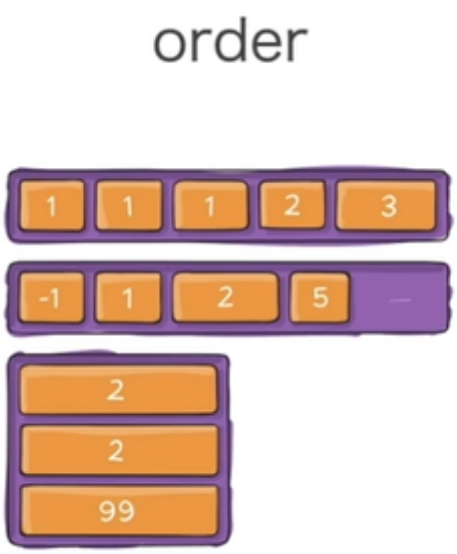
- `align-self` 某个元素的对齐方式



- `align-content` 多行内容的对齐方式



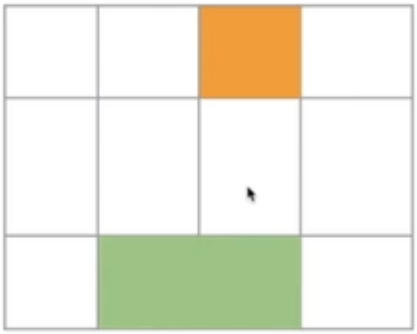
- `order` 排序的优先级，越大越后，默认为0



Grid 布局

- Grid是二维的布局方式
- 使用方式

- display: grid 使元素生成一个块级的 Grid 容器
- 使用 grid-template 相关属性将容器划分为网格
- 设置每一个子项占哪些行/列

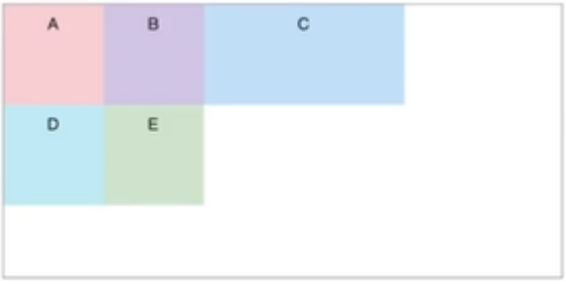


- 设置子项：（案例中有五个元素）
  - 可以看出有**两行三列**，且**行宽列宽**都被指定了

```
grid-template-columns: 100px 100px 200px;
grid-template-rows: 100px 100px

grid-template-columns: 30% 30% auto;
grid-template-rows: 100px auto

grid-template-columns: 100px 1fr 1fr;
grid-template-rows: 100px 1fr
```

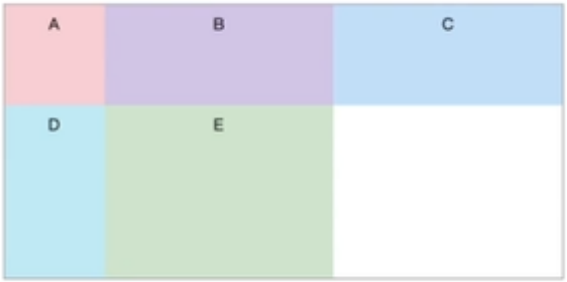


  - `fr` 单位(fraction)代表**剩下部分所占的份数**

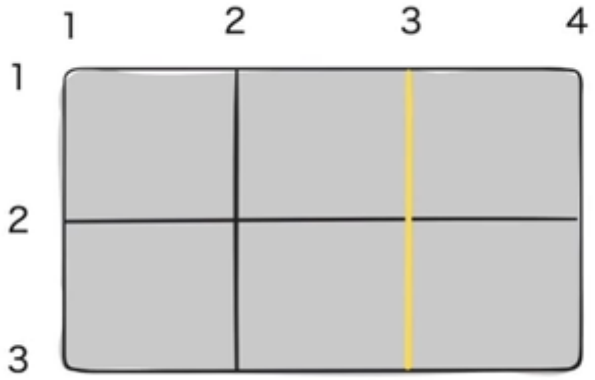
```
grid-template-columns: 100px 100px 200px;
grid-template-rows: 100px 100px

grid-template-columns: 30% 30% auto;
grid-template-rows: 100px auto

grid-template-columns: 100px 1fr 1fr;
grid-template-rows: 100px 1fr
```



- 利用网格线表示区域
  - 网格线

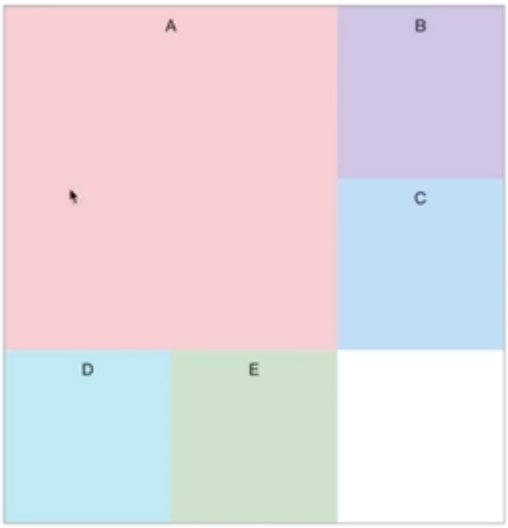


- 通过`grid-row-start`，`grid-row-end`，`grid-column-start`，`grid-column-end`来指定每个格子所占的区域
- 也可以通过`grid-area`来一次性指定上述四个属性，顺序为上界，左界，下界，右界

```
.a {
  grid-row-start: 1;
  grid-column-start: 1;
  grid-row-end: 3;
  grid-column-end: 3;
}
```

```
.a {
  grid-area: 1/1/3/3;
}
```

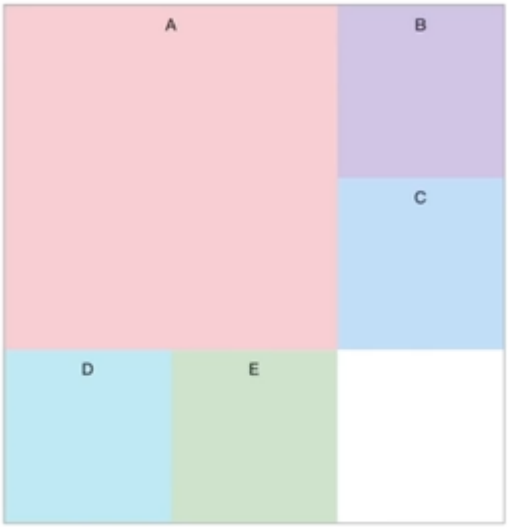
```
.a {
  grid-area: 2/2/4/4;
}
.b {
  grid-area: 1/1/3/3;
}
```




```
.a {
  grid-row-start: 1;
  grid-column-start: 1;
  grid-row-end: 3;
  grid-column-end: 3;
}
```

```
.a {
  grid-area: 1/1/3/3;
}
```

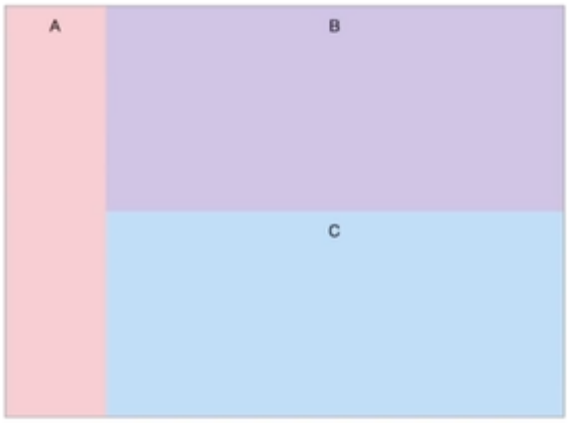
```
.a {
  grid-area: 2/2/4/4;
}
.b {
  grid-area: 1/1/3/3;
}
```



- 网格线也可以通过`grid-template`系列属性来进行命名

```
.preview {
  grid-template-columns:
    [left] 100px [center] 1fr [right];
  grid-template-rows:
    [top]
    1fr
    [middle]
    1fr
    [bottom];
}

.a {
  grid-area: top/left/bottom/center;
}
```

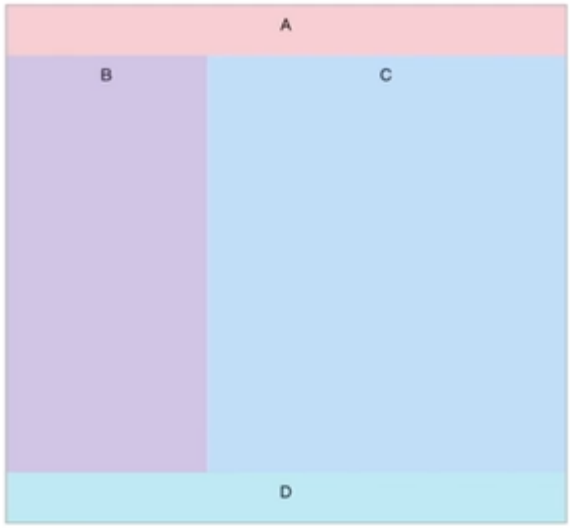


- 也可以通过**命名网格区域**来实现布局
  - 通过`grid-template-area`来命名(需要先通过col和row确定区域数)
  - 通过`grid-area`来分配位置

```
.preview {
  display: grid;
  grid-template-columns: 200px 1fr ;
  grid-template-rows: 50px
                      1fr
                      50px;
  grid-template-areas: "header header"
                       "aside  main"
                       "footer footer";
}

.a {
  grid-area: header;
}

.d {
  grid-area: footer;
}
```

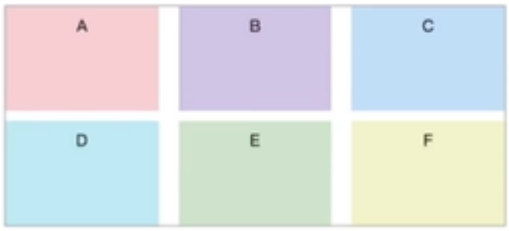


- `grid-gap`网格间隙
  - 包含行间距和列间距，先行后列或者分为两个属性

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

grid-gap: 10px 10px

grid-gap: 20px
```



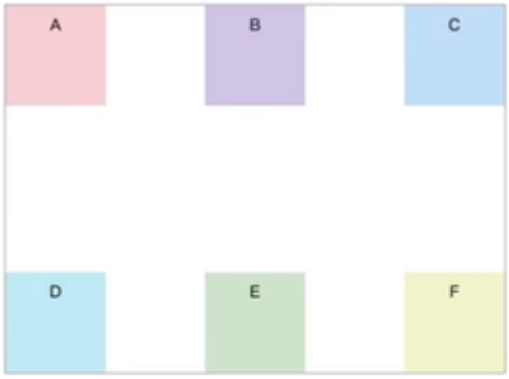
- `justify-items` 网格**行方向**对齐，设置在网络容器上
  - `stretch` 填满网格区域
  - `start, end, center` 分别指左对齐，右对齐和中间对齐
- `align-items` 网格**列方向**对齐， 设置在网络容器上
  - 也有上中下三种对齐方式
- 设置单个元素对齐同样的也有 `align-self` 和 `justify-self`
- 网格不占满容器时，整个网格在容器中的对齐方式使用 `align-content` 和 `justify-content`

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

justify-content: end;
align-content: end

justify-content: space-between;
align-content: space-between

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



表格布局

- 设计细节
- 表格每一列等宽(两种方式)
  - 设置列宽时，只设置 `th` (表头)是不够的，还需要设定 `td` 的宽度
  - 设置 `table-layout: fixed` ,则表格的单元大小只根据第一个格子的大小确定，此时只设置表头的宽度即可

```
table th {
  width: 100px;
}

table th,
table td {
  width: 100px;
}

table {
  width: 200px;
  table-layout: fixed;
}
table th {
  width: 100px;
}
```

        - 若表头设置的宽度填不满整个表的宽度，则会根据他们宽度的比例来分配整个宽度
    - 表格边框
        - 给`table`设置边框**只会**影响整个表格的边框，每个单元格不受影响
        - 需要给`td`，`th`均加上border才可以影响每个单元格
            - 但此时会出现双重边框，此时给`table`设置`border-collapse: collapse`即可合并两个边框
    - 表现得像个表格
        - 容器设置`display: table`
        - 元素设置`display: table-cell`

display: table-*

display属性值	对应的HTML标签
table	table
table-row	tr
table-row-group	tbody
table-header-group	thead
table-footer-group	tfoot
table-cell	td/th
table-caption	caption

        - table的标准出现的较早，早于`flex-box`和`grid`，建议还是先使用`flex-box`和`grid`来进行布局而非`table`布局

浮动

- `float:left` `float: right` 左右浮动
- 浮动的影响

浮动对布局的影响

- 浮动元素脱离常规流，漂浮在容器左边或右边
- 浮动元素贴着容器边缘或另外的浮动元素
- 浮动元素不会影响常规流里面的块级盒子
- 浮动元素后面的行盒会变短以避开浮动元素

- 两个左浮动的影响（足够放下两个图片）

```
<section width="700">
  <img width="200" alt="mojave"
    src="https://p4.ssl.qhimg.com/t017aec0e7
  <img width="200" alt="mojave"
    src="https://p5.ssl.qhimg.com/t01cf14495
  <p>莫哈韦沙漠不仅纬度较高，而且温度要稍微低些，是
  短叶丝兰——约书亚树的特殊栖息地。约书亚树以从茂密的
  间隔的实例等各种形式出现。除了约书亚树森林之外，该
  括加州沙漠里发现的最有趣的地质外观。</p>
</section>

<style>
  section {
    //width: 200px
  }
  img {
    float: left;
  }
  p {
```

莫哈韦沙漠不仅纬度较高，而且温度要稍微低些，是命名该公园的 短叶丝兰——约书亚树的特殊栖息地。约书亚树以从茂密的森林到远远间隔的实例等各种形式出现。除了约书亚树森林之外，该公园的西部包 括加州沙漠里发现的最有趣的地质外观。

- 两个左浮动的影响（放不下两个图片）
  - 可以看出浮动元素没有影响行级盒子整体，只是开始的若干行变短而已

```
<section width="700">
  <img width="200" alt="mojave"
    src="https://p4.ssl.qhimg.com/t017aec0e7
  <img width="200" alt="mojave"
    src="https://p5.ssl.qhimg.com/t01cf14495
  <p>莫哈韦沙漠不仅纬度较高，而且温度要稍微低些，是
  短叶丝兰——约书亚树的特殊栖息地。约书亚树以从茂密的
  间隔的实例等各种形式出现。除了约书亚树森林之外，该
  括加州沙漠里发现的最有趣的地质外观。</p>
</section>

<style>
  section {
    width: 300px
  }
  img {
    float: left;
  }
  p {
```

莫哈韦沙漠不仅纬度较高，而且温度要稍微低些，是命名该公园的短叶丝兰——约书亚树的特殊栖息地。约书亚树以从茂密的森林到远远 间隔的实例等各种形式出现。除了约书亚树森林之外，该公园的西部包括加州沙漠里发现的最有趣的地质外观。

- `clear:left` 设置元素无法与左浮动重叠，即行级盒子会从图片下方开始布局，而非形成环绕图片的效果

莫哈韦沙漠不仅纬度较高，而且温度要稍微低些，是命名该公园的 短叶丝兰——约书亚树的特殊栖息地。

科罗拉多沙漠

科罗拉多沙漠海拔低于3000英尺（910米），环绕着约书亚树国家 公园的东部，其主要特征为墨西哥三齿拉瑞阿低矮丛林、墨西哥刺木、 沙漠滨藜和包括丝兰和灌木仙人掌混合的低矮丛林的生存环境。

- 由于块级盒子不会和浮动元素上下重叠，且BFC的高度包含浮动元素，即可以通过**把浮动的元素控制在块级盒子**内来做到不影响其他元素的效果
  - 于是可以通过创建BFC的方式来达到不重叠的方式
    - 最简单创建BFC的方式即 `overflow: hidden` (不为 `visible`)
  - 或者可以添加一个after伪元素，这样计算高度的时候也能够把浮动元素的高度算在内
    - 此处 `display:block` 的加入使得盒子从行级盒子变为了块级盒子

```css
.clearfix:after {
  content: '';
  display: block;
  clear: both;
}
```

- 定位
  - 通过`postion`属性来设定元素的定位方式

position 属性

static	默认值，非定位元素
relative	相对自身原本位置偏移，不脱离文档流
absolute	绝对定位，相对非 static 祖先元素定位
fixed	相对于视口绝对定位

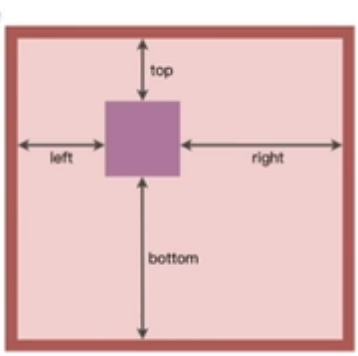
  - relative
    - 仅在其原始基准位置进行上下左右的偏移

position: relative

- 在常规流里面布局
- 相对于自己本应该在的位置进行偏移
- 使用 top、left、bottom、right 设置偏移长度
- 流内其它元素当它没有偏移一样布局

  - absolute
    - 如果无最近的`static`祖先，则相对于**根元素**进行定位

- 脱离常规流
- 相对于最近的非 static 祖先定位
- 不会对流内元素布局造成影响

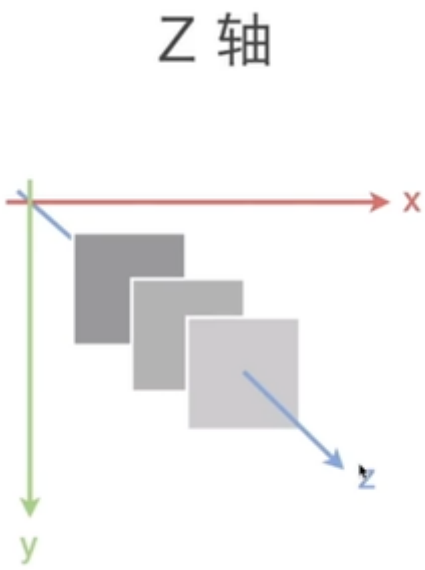


    - 可以用来实现某元素在容器(最近非static祖先)中的视觉居中
    - 当不设置`left`，`right`等定位值时，其默认值为`auto`，此时该元素位于他原本在流内的位置，但由于他不会运行别的流内元素，则会出现和下一个流内元素重叠的效果

  - ### Fixed
    - 相对于**视口**定位
    - 可以做到**菜单行冻结的效果**

堆叠层级

- Z轴
  - Z轴越大越靠近用户，也就表现得更上(**盖住了**其他下面的元素)

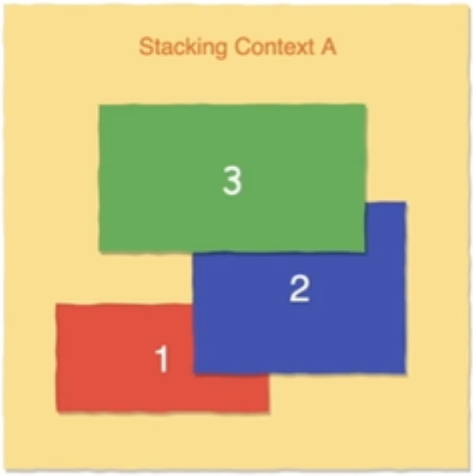


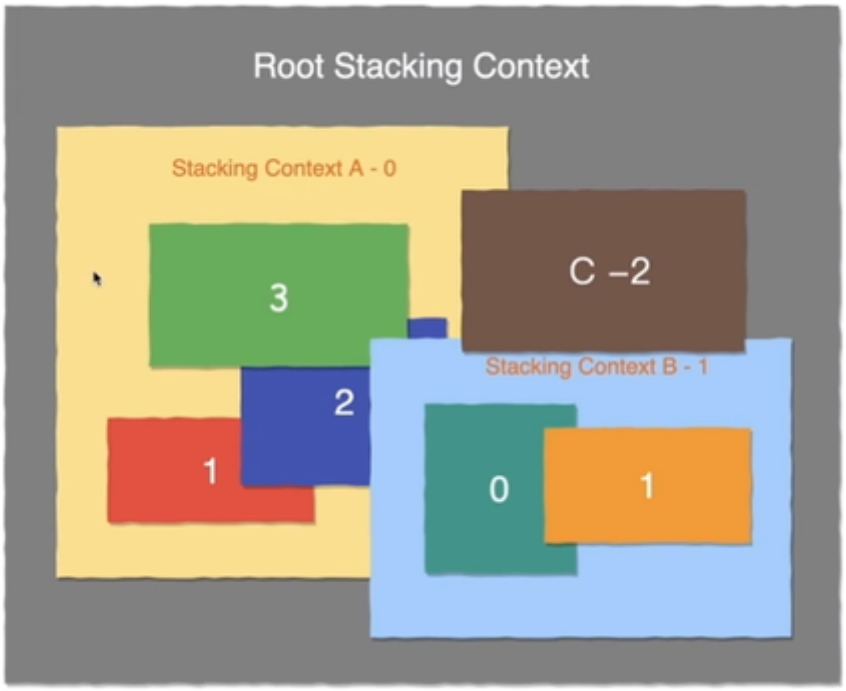
z-index

- 为定位元素指定其在 z 轴的上下层级
- 用一个整数表示，数值越大，越靠近用户
- 初始值为 auto，可以为负数、0、正数

- 堆叠上下文：在同一个堆叠上下文的元素之间可以通过z-index来判断上下关系
  - **不在一个上下文之间**的元素依靠上下文的z-index来处理上下关系

堆叠上下文 Stacking Context





- 如何创建堆叠上下文

堆叠上下文的创建

- Root 元素
- z-index 值不为 auto 的 relative/absolute
- position 是 fixed 的元素
- 设置了某些属性的元素
  - opacity 不为1
  - transform
  - animation

- 绘制顺序
  - 可以看出 `z-index` 是0时即可盖住没有设置 `z-index` 的元素

绘制顺序

- 在每一个堆叠上下文中，从下到上：
  - 形成该上下文的元素的 border 和 background
  - z-index 为负值的子堆叠上下文
  - 常规流内的块级元素
  - 浮动元素
  - 常规流内行级元素
  - z-index 为 0 的子元素或子堆叠上下文
  - z-index 为正数的子堆叠上下文