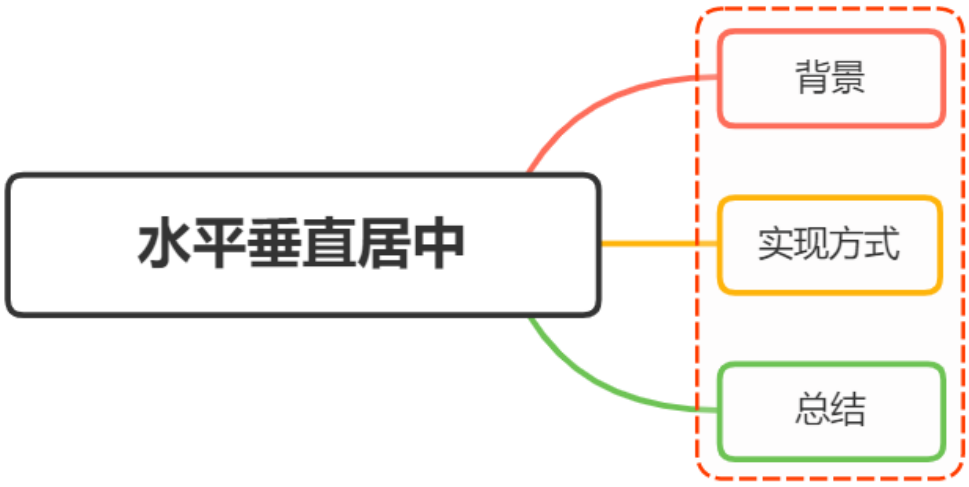
- 其实这是一种折中性质的设计解决方案，多方面因素影响而达不到最佳效果
- 一定程度上改变了网站原有的布局结构，会出现用户混淆的情况

4. 元素水平垂直居中的方法有哪些？如果元素不定宽高呢？



4.1. 背景

在开发中经常遇到这个问题，即让某个元素的内容在水平和垂直方向上都居中，内容不仅限于文字，可能是图片或其他元素

居中是一个非常基础但又是非常重要的应用场景，实现居中的方法存在很多，可以将这些方法分成两个大类：

- 居中元素（子元素）的宽高已知
- 居中元素宽高未知

4.2. 实现方式

实现元素水平垂直居中的方式：

- 利用定位+margin:auto
- 利用定位+margin:负值
- 利用定位+transform
- table布局

- flex布局
- grid布局

4.2.1. 利用定位+margin:auto

先上代码：

HTML | 复制代码

```html
<style>
    .father{
        width:500px;
        height:300px;
        border:1px solid #0a3b98;
        position: relative;
    }
    .son{
        width:100px;
        height:40px;
        background: #f0a238;
        position: absolute;
        top:0;
        left:0;
        right:0;
        bottom:0;
        margin:auto;
    }
</style>
<div class="father">
    <div class="son"></div>
</div>
```

父级设置为相对定位，子级绝对定位 ， 并且四个定位属性的值都设置了0，那么这时候如果子级没有设置宽高，则会被拉开到和父级一样宽高

这里子元素设置了宽高，所以宽高会按照我们的设置来显示，但是实际上子级的虚拟占位已经撑满了整个父级，这时候再给它一个 `margin: auto` 它就可以上下左右都居中了

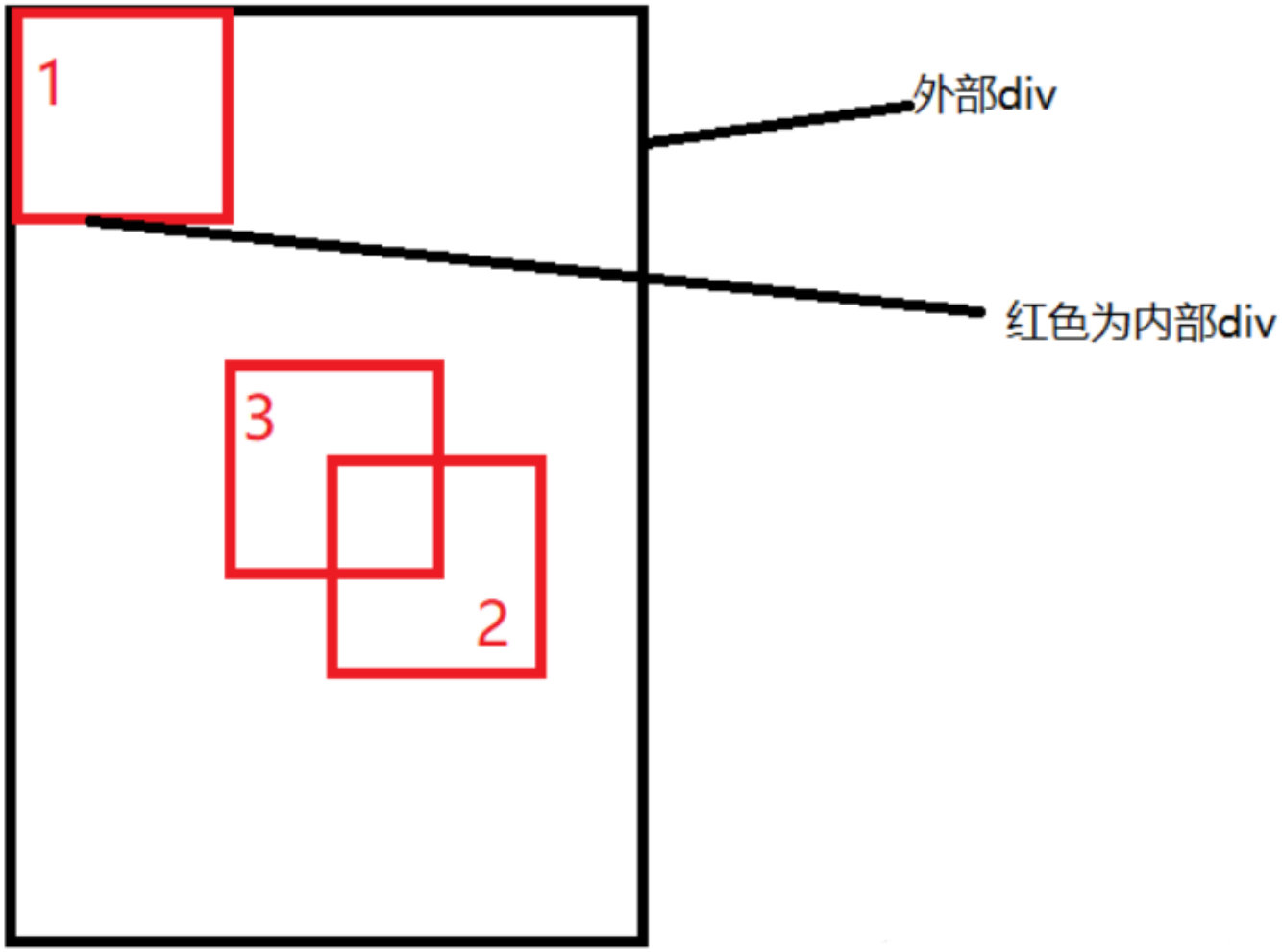
4.2.2. 利用定位+margin:负值

绝大多数情况下，设置父元素为相对定位， 子元素移动自身50%实现水平垂直居中

HTML | 复制代码

```html
<style>
    .father {
        position: relative;
        width: 200px;
        height: 200px;
        background: skyblue;
    }
    .son {
        position: absolute;
        top: 50%;
        left: 50%;
        margin-left:-50px;
        margin-top:-50px;
        width: 100px;
        height: 100px;
        background: red;
    }
</style>
<div class="father">
    <div class="son"></div>
</div>
```

整个实现思路如下图所示：



- 初始位置为方块1的位置
- 当设置left、top为50%的时候，内部子元素为方块2的位置
- 设置margin为负数时，使内部子元素到方块3的位置，即中间位置

这种方案不要求父元素的高度，也就是即使父元素的高度变化了，仍然可以保持在父元素的垂直居中位置，水平方向上是一样的操作

但是该方案需要知道子元素自身的宽高，但是我们可以通过下面 `transform` 属性进行移动

4.2.3. 利用定位+transform

实现代码如下：

CSS | 复制代码

```
<style>
    .father {
        position: relative;
        width: 200px;
        height: 200px;
        background: skyblue;
    }
    .son {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%,-50%);
        width: 100px;
        height: 100px;
        background: red;
    }
</style>
<div class="father">
    <div class="son"></div>
</div>
```

`translate(-50%, -50%)` 将会将元素位移自己宽度和高度的-50%

这种方法其实和最上面被否定掉的margin负值用法一样，可以说是 `margin` 负值的替代方案，并不需要知道自身元素的宽高

4.2.4. table布局

设置父元素为 `display:table-cell` ，子元素设置 `display: inline-block` 。利用 `vertical` 和 `text-align` 可以让所有的行内块级元素水平垂直居中

HTML | 复制代码

```html
<style>
    .father {
        display: table-cell;
        width: 200px;
        height: 200px;
        background: skyblue;
        vertical-align: middle;
        text-align: center;
    }
    .son {
        display: inline-block;
        width: 100px;
        height: 100px;
        background: red;
    }
</style>
<div class="father">
    <div class="son"></div>
</div>
```

4.2.5. flex弹性布局

还是看看实现的整体代码：

HTML | 复制代码

```html
<style>
    .father {
        display: flex;
        justify-content: center;
        align-items: center;
        width: 200px;
        height: 200px;
        background: skyblue;
    }
    .son {
        width: 100px;
        height: 100px;
        background: red;
    }
</style>
<div class="father">
    <div class="son"></div>
</div>
```

`css3` 中了 `flex` 布局，可以非常简单实现垂直水平居中

这里可以简单看看 `flex` 布局的关键属性作用：

- display: flex时，表示该容器内部的元素将按照flex进行布局
- align-items: center表示这些元素将相对于本容器水平居中
- justify-content: center也是同样的道理垂直居中

4.2.6. grid网格布局

```HTML
<style>
    .father {
            display: grid;
            align-items:center;
            justify-content: center;
            width: 200px;
            height: 200px;
            background: skyblue;

        }
        .son {
            width: 10px;
            height: 10px;
            border: 1px solid red
        }
</style>
<div class="father">
    <div class="son"></div>
</div>
```

这里看到，`gird` 网格布局和 `flex` 弹性布局都简单粗暴

4.2.7. 小结

上述方法中，不知道元素宽高大小仍能实现水平垂直居中的方法有：

- 利用定位+margin:auto
- 利用定位+transform
- flex布局
- grid布局

4.3. 总结

根据元素标签的性质，可以分为：

- 内联元素居中布局
- 块级元素居中布局

4.3.1. 内联元素居中布局

水平居中

- 行内元素可设置：text-align: center
- flex布局设置父元素：display: flex; justify-content: center

垂直居中

- 单行文本父元素确认高度：height === line-height
- 多行文本父元素确认高度：display: table-cell; vertical-align: middle

4.3.2. 块级元素居中布局

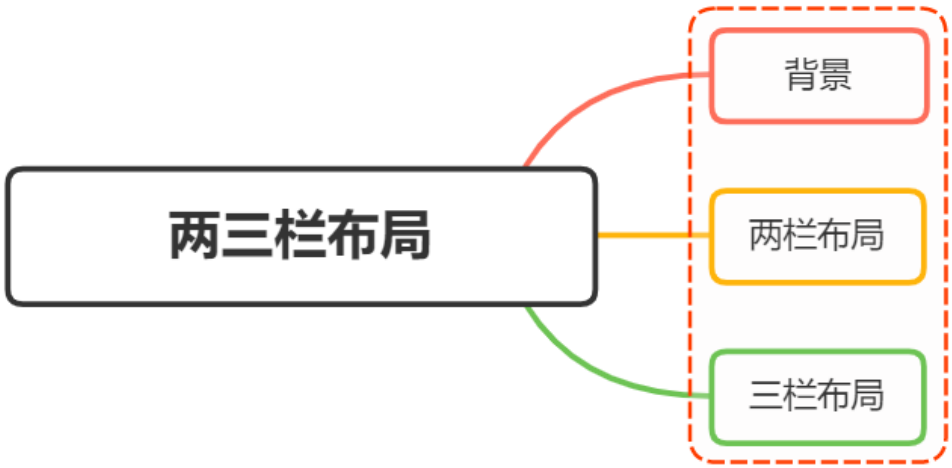
水平居中

- 定宽: margin: 0 auto
- 绝对定位+left:50%+margin:负自身一半

垂直居中

- position: absolute设置left、top、margin-left、margin-top(定高)
- display: table-cell
- transform: translate(x, y)
- flex(不定高，不定宽)
- grid(不定高，不定宽)，兼容性相对比较差

5. 如何实现两栏布局，右侧自适应？三栏布局中间自适应呢？



5.1. 背景

在日常布局中，无论是两栏布局还是三栏布局，使用的频率都非常高

5.1.1. 两栏布局

两栏布局实现效果就是将页面分割成左右宽度不等的两列，宽度较小的列设置为固定宽度，剩余宽度由另一列撑满，

比如 `Ant Design` 文档，蓝色区域为主要内容布局容器，侧边栏为次要内容布局容器

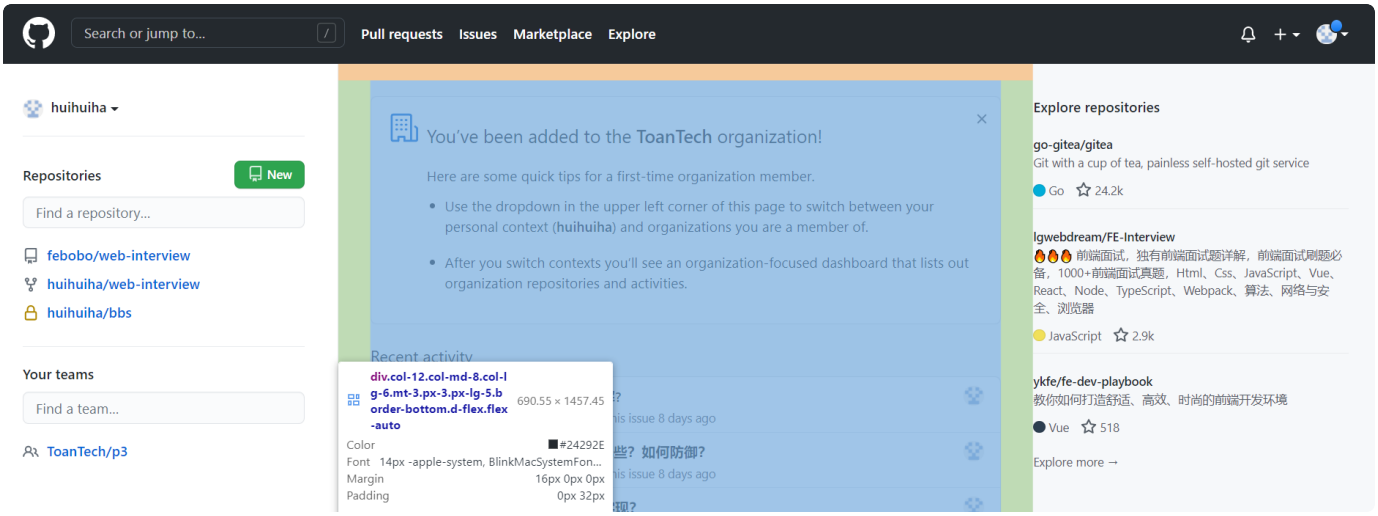
> 这里称宽度较小的列父元素为次要布局容器，宽度较大的列父元素为主要布局容器

这种布局适用于内容上具有明显主次关系的网页

5.1.2. 三栏布局

三栏布局按照左中右的顺序进行排列，通常中间列最宽，左右两列次之

大家最常见的就是 `github`：



5.2. 两栏布局

两栏布局非常常见，往往是以一个定宽栏和一个自适应的栏并排展示存在

实现思路也非常的简单：

- 使用 float 左浮左边栏
- 右边模块使用 margin-left 撑出内容块做内容展示
- 为父级元素添加BFC，防止下方元素飞到上方内容

代码如下：

```HTML
<style>
    .box{
        overflow: hidden; 添加BFC
    }
    .left {
        float: left;
        width: 200px;
        background-color: gray;
        height: 400px;
    }
    .right {
        margin-left: 210px;
        background-color: lightgray;
        height: 200px;
    }
</style>
<div class="box">
    <div class="left">左边</div>
    <div class="right">右边</div>
</div>
```

还有一种更为简单的使用则是采取：flex弹性布局

5.2.1. flex弹性布局

```HTML
<style>
    .box{
        display: flex;
    }
    .left {
        width: 100px;
    }
    .right {
        flex: 1;
    }
</style>
<div class="box">
    <div class="left">左边</div>
    <div class="right">右边</div>
</div>
```

`flex` 可以说是最好的方案了，代码少，使用简单

注意的是， `flex` 容器的一个默认属性值: `align-items: stretch;`

这个属性导致了列等高的效果。 为了让两个盒子高度自动，需要设置: `align-items: flex-start`

5.3. 三栏布局

实现三栏布局中间自适应的布局方式有：

- 两边使用 float，中间使用 margin
- 两边使用 absolute，中间使用 margin
- 两边使用 float 和负 margin
- display: table 实现
- flex实现
- grid网格布局

5.3.1. 两边使用 float，中间使用 margin

需要将中间的内容放在 `html` 结构最后，否则右侧会臣在中间内容的下方

实现代码如下：

HTML | 复制代码

```html
<style>
    .wrap {
        background: #eee;
        overflow: hidden; <!-- 生成BFC，计算高度时考虑浮动的元素 -->
        padding: 20px;
        height: 200px;
    }
    .left {
        width: 200px;
        height: 200px;
        float: left;
        background: coral;
    }
    .right {
        width: 120px;
        height: 200px;
        float: right;
        background: lightblue;
    }
    .middle {
        margin-left: 220px;
        height: 200px;
        background: lightpink;
        margin-right: 140px;
    }
</style>
<div class="wrap">
    <div class="left">左侧</div>
    <div class="right">右侧</div>
    <div class="middle">中间</div>
</div>
```

原理如下：

- 两边固定宽度，中间宽度自适应。
- 利用中间元素的margin值控制两边的间距
- 宽度小于左右部分宽度之和时，右侧部分会被挤下去

这种实现方式存在缺陷：

- 主体内容是最后加载的。
- 右边在主体内容之前，如果是响应式设计，不能简单的换行展示

5.3.2. 两边使用 absolute，中间使用 margin

基于绝对定位的三栏布局：注意绝对定位的元素脱离文档流，相对于最近的已经定位的祖先元素进行定位。无需考虑HTML中结构的顺序

HTML | 复制代码

```html
<style>
  .container {
    position: relative;
  }

  .left,
  .right,
  .main {
    height: 200px;
    line-height: 200px;
    text-align: center;
  }

  .left {
    position: absolute;
    top: 0;
    left: 0;
    width: 100px;
    background: green;
  }

  .right {
    position: absolute;
    top: 0;
    right: 0;
    width: 100px;
    background: green;
  }

  .main {
    margin: 0 110px;
    background: black;
    color: white;
  }
</style>

<div class="container">
  <div class="left">左边固定宽度</div>
  <div class="right">右边固定宽度</div>
  <div class="main">中间自适应</div>
</div>
```

实现流程：

- 左右两边使用绝对定位，固定在两侧。

- 中间占满一行，但通过 margin和左右两边留出10px的间隔

5.3.3. 两边使用 float 和负 margin

HTML | 复制代码

```html
<style>
  .left,
  .right,
  .main {
    height: 200px;
    line-height: 200px;
    text-align: center;
  }

  .main-wrapper {
    float: left;
    width: 100%;
  }

  .main {
    margin: 0 110px;
    background: black;
    color: white;
  }

  .left,
  .right {
    float: left;
    width: 100px;
    margin-left: -100%;
    background: green;
  }

  .right {
    margin-left: -100px; /* 同自身宽度 */
  }
</style>

<div class="main-wrapper">
  <div class="main">中间自适应</div>
</div>
<div class="left">左边固定宽度</div>
<div class="right">右边固定宽度</div>
```

实现过程：

- 中间使用了双层标签，外层是浮动的，以便左中右能在同一行展示
- 左边通过使用负 margin-left:-100%，相当于中间的宽度，所以向上偏移到左侧
- 右边通过使用负 margin-left:-100px，相当于自身宽度，所以向上偏移到最右侧

缺点：

- 增加了 .main-wrapper 一层，结构变复杂
- 使用负 margin，调试也相对麻烦

5.3.4. 使用 display: table 实现

`<table>` 标签用于展示行列数据，不适合用于布局。但是可以使用 `display: table` 来实现布局的效果

HTML | 复制代码

```html
<style>
  .container {
    height: 200px;
    line-height: 200px;
    text-align: center;
    display: table;
    table-layout: fixed;
    width: 100%;
  }

  .left,
  .right,
  .main {
    display: table-cell;
  }

  .left,
  .right {
    width: 100px;
    background: green;
  }

  .main {
    background: black;
    color: white;
    width: 100%;
  }
</style>

<div class="container">
  <div class="left">左边固定宽度</div>
  <div class="main">中间自适应</div>
  <div class="right">右边固定宽度</div>
</div>
```

实现原理：

- 层通过 display: table设置为表格，设置 table-layout: fixed`表示列宽自身宽度决定，而不是自动计算。
- 内层的左中右通过 display: table-cell设置为表格单元。
- 左右设置固定宽度，中间设置 width: 100% 填充剩下的宽度

5.3.5. 使用flex实现

利用 `flex` 弹性布局，可以简单实现中间自适应

代码如下：

HTML | 复制代码

```
<style type="text/css">
    .wrap {
        display: flex;
        justify-content: space-between;
    }

    .left,
    .right,
    .middle {
        height: 100px;
    }

    .left {
        width: 200px;
        background: coral;
    }

    .right {
        width: 120px;
        background: lightblue;
    }

    .middle {
        background: #555;
        width: 100%;
        margin: 0 20px;
    }
</style>
<div class="wrap">
    <div class="left">左侧</div>
    <div class="middle">中间</div>
    <div class="right">右侧</div>
</div>
```

实现过程：

- 仅需将容器设置为 `display:flex;` ，
- 盒内元素两端对其，将中间元素设置为 `100%` 宽度，或者设为 `flex:1` ，即可填充空白
- 盒内元素的高度撑开容器的高度

优点：

- 结构简单直观
- 可以结合 flex的其他功能实现更多效果，例如使用 order属性调整显示顺序，让主体内容优先加载，但展示在中间

5.3.6. grid网格布局

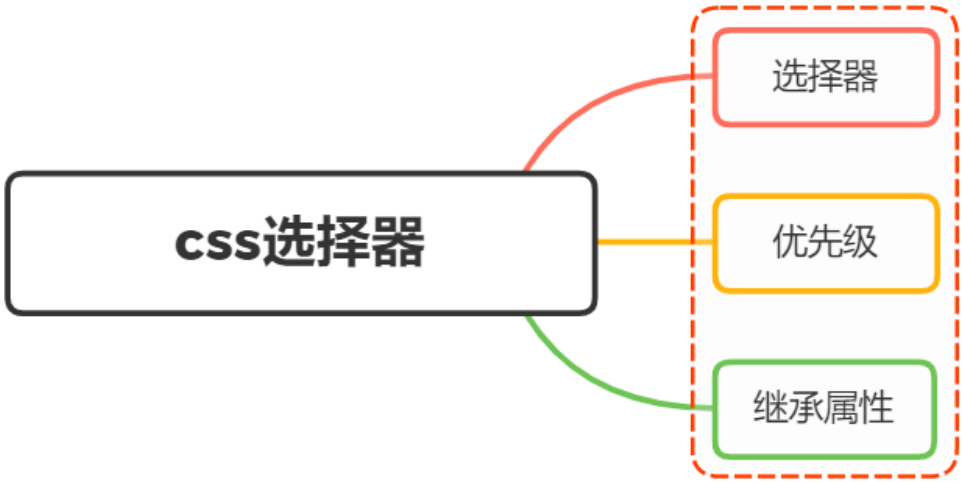
代码如下：

HTML | 复制代码

```HTML
<style>
    .wrap {
        display: grid;
        width: 100%;
        grid-template-columns: 300px auto 300px;
    }

    .left,
    .right,
    .middle {
        height: 100px;
    }

    .left {
        background: coral;
    }

    .right {
        background: lightblue;
    }

    .middle {
        background: #555;
    }
</style>
<div class="wrap">
    <div class="left">左侧</div>
    <div class="middle">中间</div>
    <div class="right">右侧</div>
</div>
```

跟 `flex` 弹性布局一样的简单

6. css选择器有哪些？优先级？哪些属性可以继承？



6.1. 选择器

CSS选择器是CSS规则的第一部分

它是元素和其他部分组合起来告诉浏览器哪个HTML元素应当是被选为应用规则中的CSS属性值的方式

选择器所选择的元素，叫做“选择器的对象”

我们从一个 `Html` 结构开始

HTML | 复制代码

```html
<div id="box">
    <div class="one">
        <p class="one_1">
        </p >
        <p class="one_1">
        </p >
    </div>
    <div class="two"></div>
    <div class="two"></div>
    <div class="two"></div>
</div>
```

关于 `css` 属性选择器常用的有：

- id选择器（#box），选择id为box的元素
- 类选择器（.one），选择类名为one的所有元素
- 标签选择器（div），选择标签为div的所有元素

- 后代选择器（#box div），选择id为box元素内部所有的div元素
- 子选择器（.one>one_1），选择父元素为.one的所有.one_1的元素
- 相邻同胞选择器（.one+.two），选择紧接在.one之后的所有.two元素
- 群组选择器（div,p），选择div、p的所有元素

还有一些使用频率相对没那么多的选择器：

- 伪类选择器

```css
:link ： 选择未被访问的链接
:visited：选取已被访问的链接
:active：选择活动链接
:hover ： 鼠标指针浮动在上面的元素
:focus ： 选择具有焦点的
:first-child：父元素的首个子元素
```

- 伪元素选择器

```css
:first-letter ： 用于选取指定选择器的首字母
:first-line ： 选取指定选择器的首行
:before ： 选择器在被选元素的内容前面插入内容
:after ： 选择器在被选元素的内容后面插入内容
```

- 属性选择器

```css
[attribute] 选择带有attribute属性的元素
[attribute=value] 选择所有使用attribute=value的元素
[attribute~=value] 选择attribute属性包含value的元素
[attribute|=value]：选择attribute属性以value开头的元素
```

在 `CSS3` 中新增的选择器有如下：

- 层次选择器（p~ul），选择前面有p元素的每个ul元素
- 伪类选择器

```css
:first-of-type 表示一组同级元素中其类型的第一个元素
:last-of-type 表示一组同级元素中其类型的最后一个元素
:only-of-type 表示没有同类型兄弟元素的元素
:only-child 表示没有任何兄弟的元素
:nth-child(n) 根据元素在一组同级中的位置匹配元素
:nth-last-of-type(n) 匹配给定类型的元素，基于它们在一组兄弟元素中的位置，从末尾开始计数
:last-child 表示一组兄弟元素中的最后一个元素
:root 设置HTML文档
:empty 指定空的元素
:enabled 选择可用元素
:disabled 选择被禁用元素
:checked 选择选中的元素
:not(selector) 选择与 <selector> 不匹配的所有元素
```

- 属性选择器

```css
[attribute*=value]：选择attribute属性值包含value的所有元素
[attribute^=value]：选择attribute属性开头为value的所有元素
[attribute$=value]：选择attribute属性结尾为value的所有元素
```

6.2. 优先级

相信大家对 `CSS` 选择器的优先级都不陌生：

> 内联 > ID选择器 > 类选择器 > 标签选择器

到具体的计算层面，优先级是由 A 、B、C、D 的值来决定的，其中它们的值计算规则如下：

- 如果存在内联样式，那么 A = 1, 否则 A = 0
- B的值等于 ID选择器出现的次数
- C的值等于 类选择器 和 属性选择器 和 伪类 出现的总次数
- D 的值等于 标签选择器 和 伪元素 出现的总次数

这里举个例子：

```css
#nav-global > ul > li > a.nav-link
```