

拉勾教育

— 互联网人实战大学 —

# 《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

# 第04讲：掌握 CSS 精髓：布局

**CSS** 虽然初衷是用来美化 HTML 文档的  
实际上随着 float、position 等属性的出现  
它已经可以起到**调整文档渲染结构**的作用了  
随着**弹性盒子**以及**网格布局**的推出  
CSS 将承担越来越重要的布局功能

**HTML** 标签决定了页面的逻辑结构

**CSS** 决定了页面的视觉结构



## 布局的两个共同点

- 大多数用于 PC 端

因为 PC 端屏幕像素宽度够大，可布局的空间也大

- 布局是有限空间内的元素排列方式

因为页面设计横向不滚动，纵向无限延伸

所以大多数时候讨论的布局都是对水平方向进行分割



**单列布局**是最常用的一种布局

实现效果就是将一个元素作为布局容器

通常设置一个较小的（最大）宽度来保证不同像素宽度屏幕下显示一致

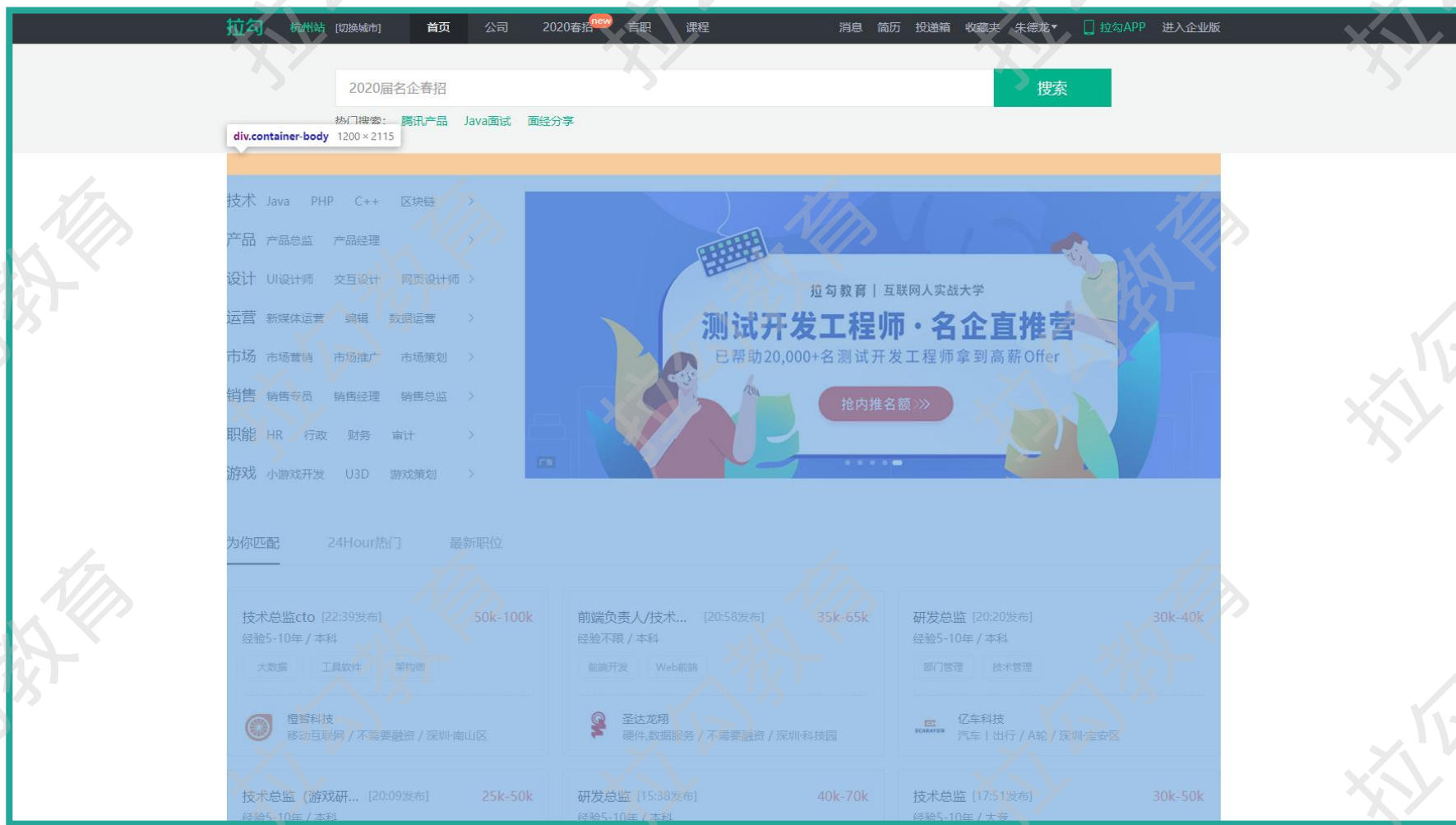


# 单列布局

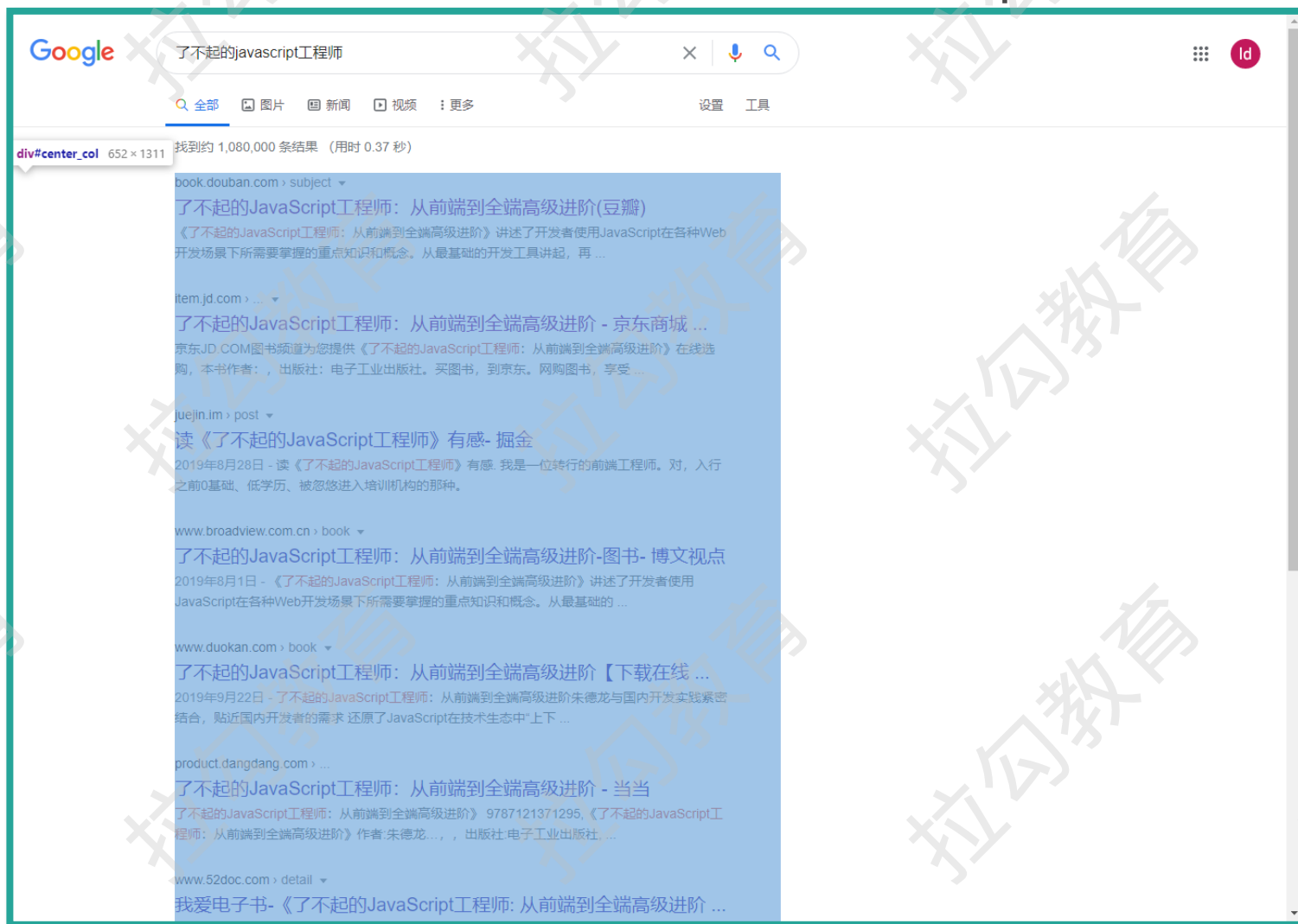
拉勾教育

— 互联网人实战大学 —

蓝色区域为布局容器，水平居中对齐，宽度 1260px



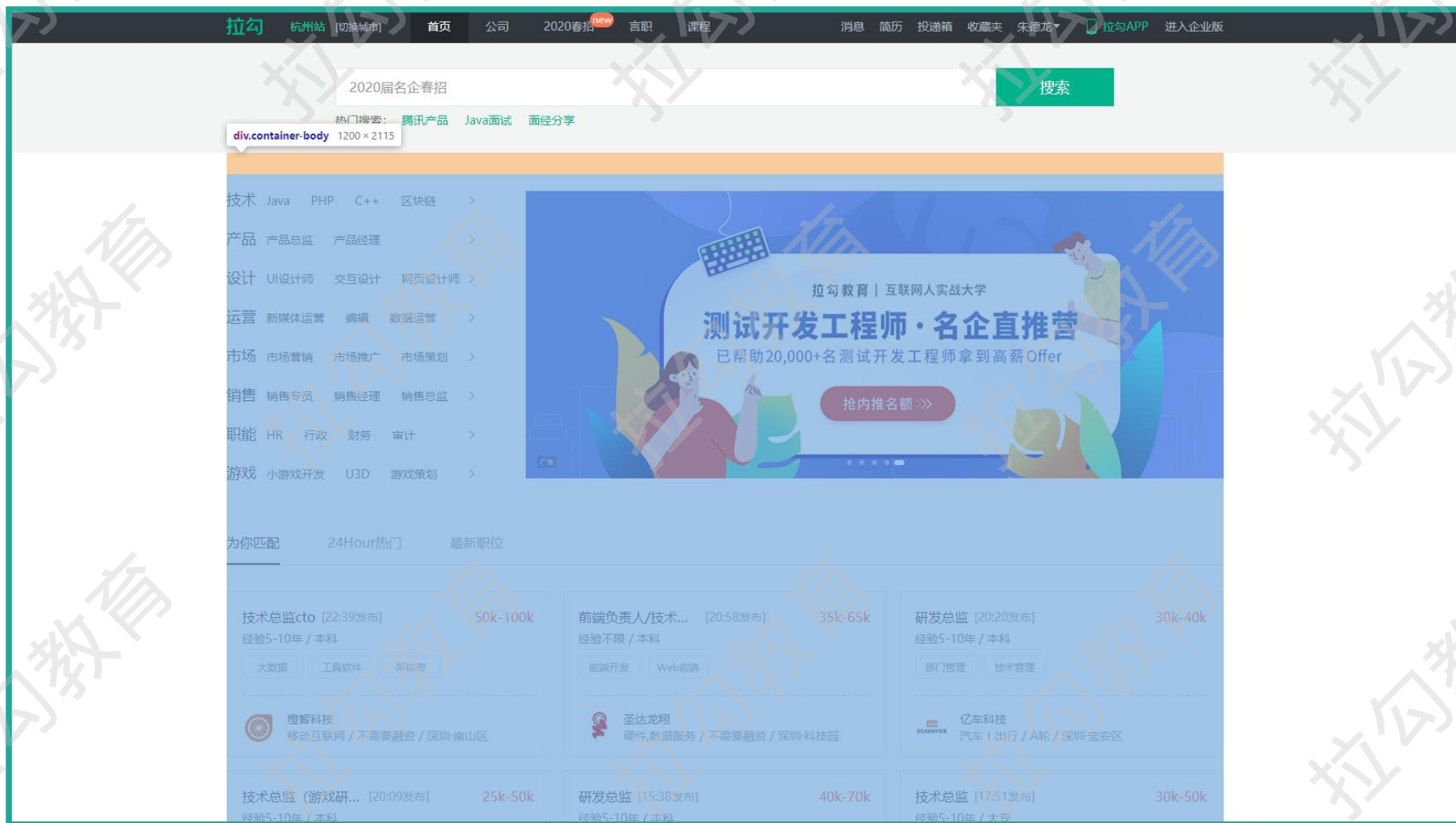
蓝色区域为布局容器，水平左对齐，宽度 652px



# 单列布局

拉勾教育

— 互联网人实战大学 —





**优势**在于基本上可以适配超过布局容器宽度的各种显示屏幕

**缺点**也是源于此，过度的冗余设计必然会带来浪费

在上面的例子中，其实我的屏幕宽度可以显示更多的内容

但是页面两侧却出现了大量空白区域



## 2 列布局

拉勾教育

— 互联网人实战大学 —

**2 列布局**的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

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**

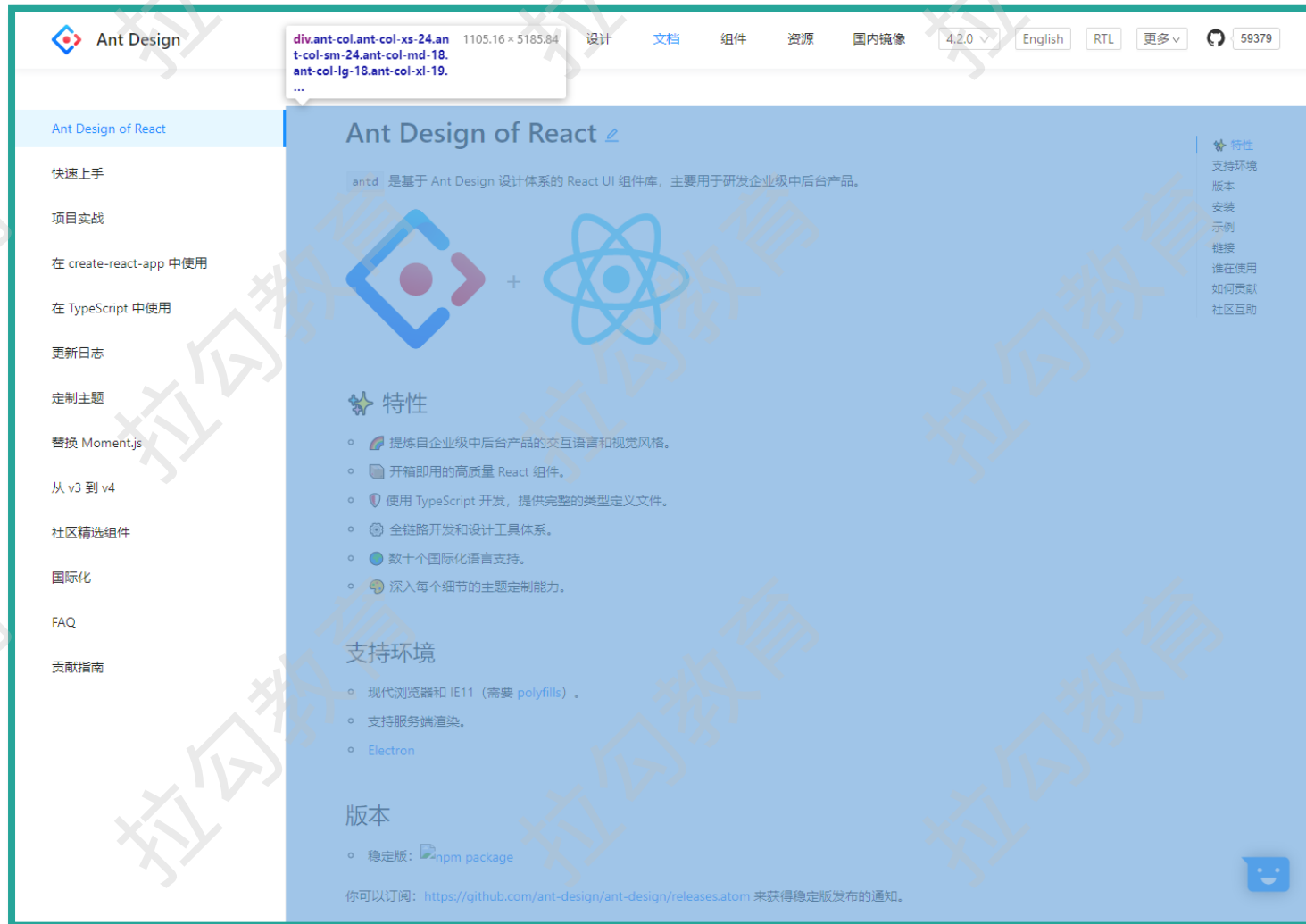


## 2 列布局

拉勾教育

— 互联网人实战大学 —

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



## 3 列布局

拉勾教育

— 互联网人实战大学 —

**3 列布局**按照左中右的顺序进行排列

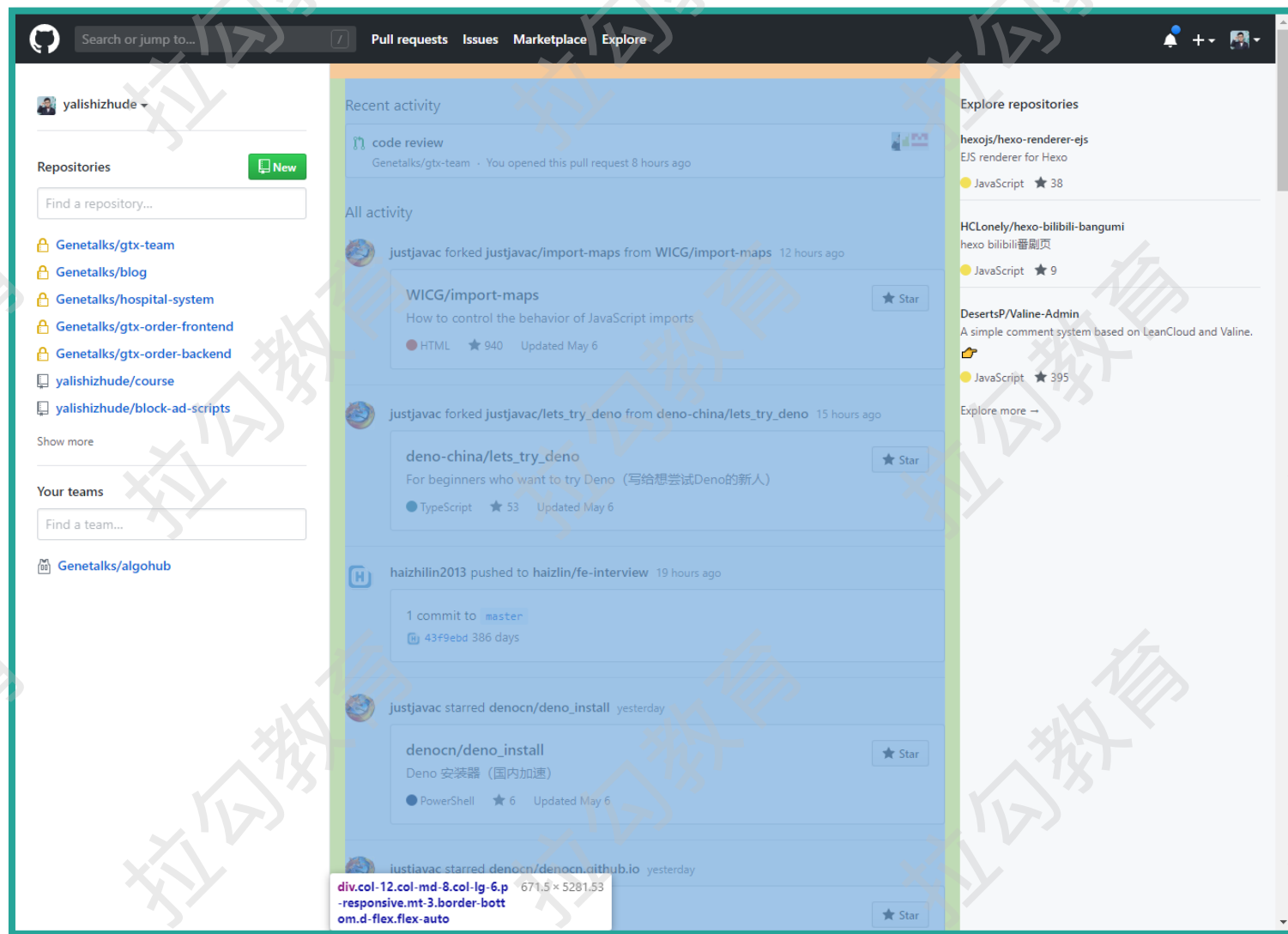
通常中间列最宽，左右两列次之



## 3 列布局

拉勾教育

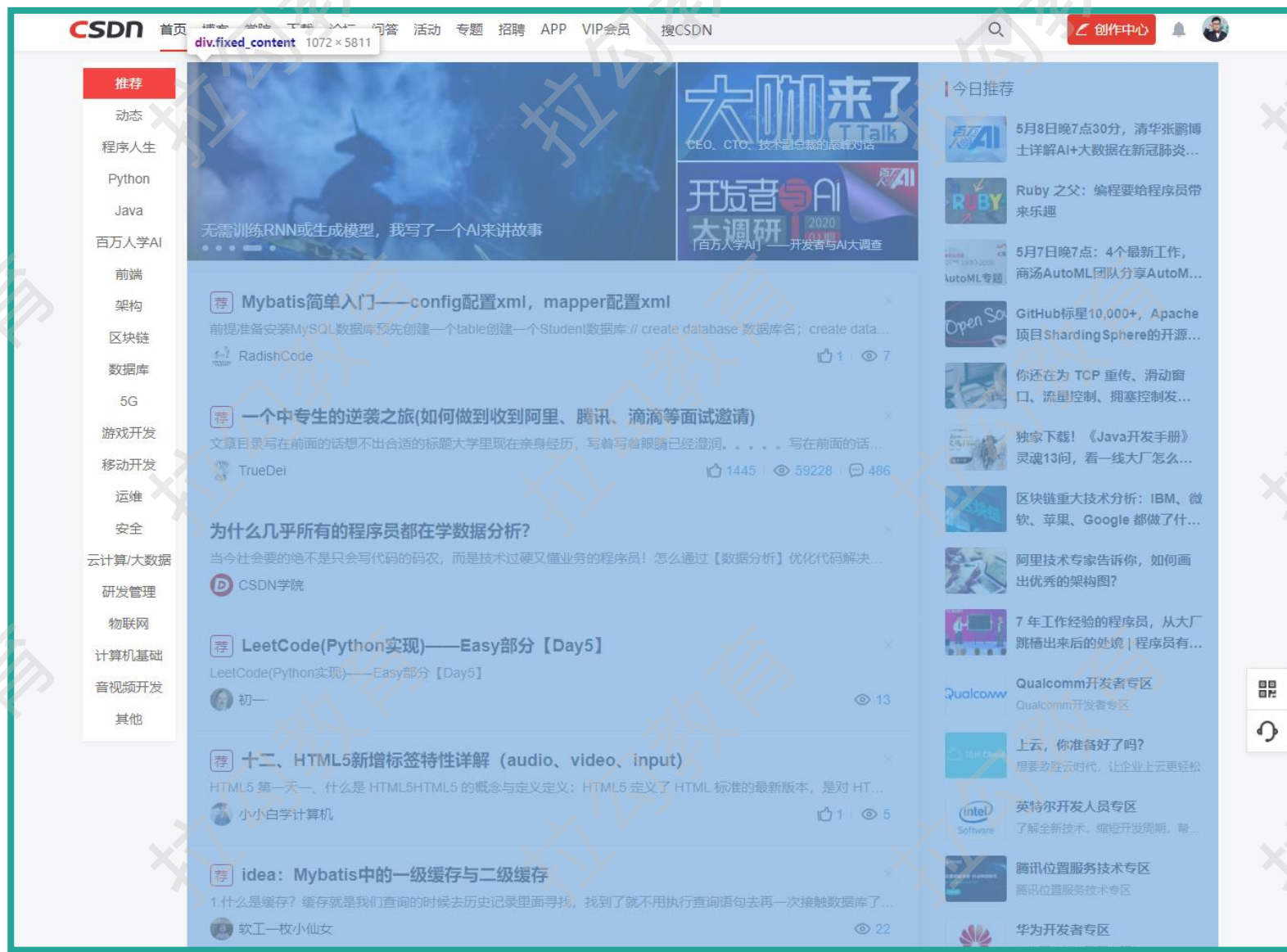
— 互联网人实战大学 —



## 3 列布局

拉勾教育

— 互联网人实战大学 —



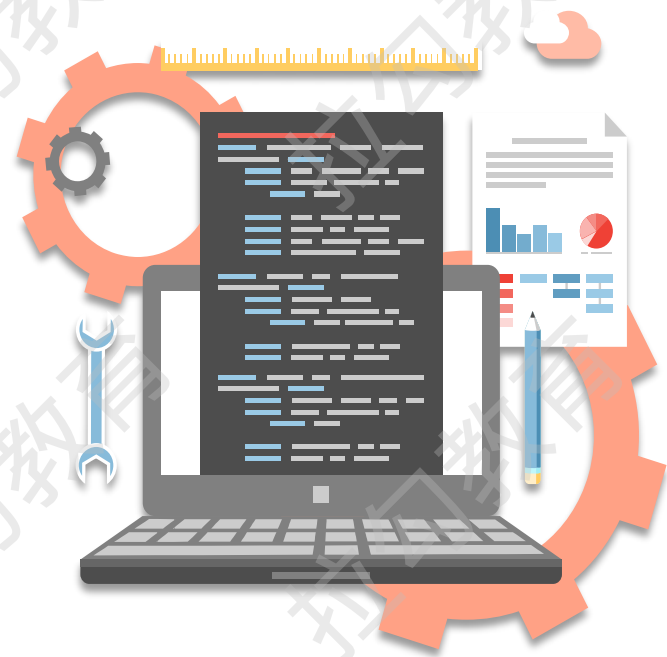
单列布局通过将设置布局容器（最大）宽度以及左右边距为 auto 即可实现

重点讨论 2 列和 3 列布局



通过**归纳法**找到这些方式的共同实现步骤

1. 为了保证主要布局容器**优先级**，应将主要布局容器写在次要布局容器之前
2. 将布局容器进行**水平排列**
3. **设置宽度**，即次要容器宽度固定，主要容器撑满
4. 消除布局方式的**副作用**，如浮动造成的高度塌陷
5. 为了在窄屏下也能正常显示，可以通过媒体查询进行**优化**





```
<style>
/*为了方便查看，给布局容器设置高度和颜色*/
main,aside {
  height: 100px;
}
main {
  background-color: #f09e5a;
}
aside {
  background-color: #c295cf;
}
</style>
<div>
  <main>主要布局容器</main>
  <aside>次要布局容器</aside>
</div>
```

1. 将主要布局容器写在次要布局容器之前

主要布局容器

次要布局容器

```
<style>
.wrap {
  display: flex;
  flex-direction: row-reverse;
}
.main {
  flex: 1;
}
.aside {
  flex: 1;
}
</style>
<div class="wrap">
  <main class="main">主要布局容器</main>
  <aside class="aside">次要布局容器</aside>
</div>
```

2. 将布局容器进行水平排列

次要布局容器

主要布局容器

```
<style>
.wrap {
  display: flex;
  flex-direction: row-reverse;
}
.main {
  flex: 1;
}
.aside {
  width: 200px;
}
</style>
<div class="wrap">
  <main class="main">主要布局容器</main>
  <aside class="aside">次要布局容器</aside>
</div>
```

3. 次要容器宽度固定，主要容器撑满

次要布局容器

主要布局容器

#### 4. 消除副作用(flex不会产生副作用)

次要布局容器

主要布局容器

```
<style>
.wrap {
  display: flex;
  flex-direction: row-reverse;
  flex-wrap: wrap;
}

.main {
  flex: 1;
}

.aside {
  width: 200px;
}
```



```
}  
@media only screen and (max-width: 1000px){  
  .wrap {  
    flex-direction: row;  
  }  
  .main {  
    flex: 100%;  
  }  
}  
  
</style>  
<div class="wrap">  
  <main class="main">主要布局容器</main>
```

```
.wrap {  
  flex-direction: row;  
}  
  
.main {  
  flex: 100%;  
}  
}  
  
</style>  
<div class="wrap">  
  <main class="main">主要布局容器</main>  
  <aside class="aside">次要布局容器</aside>  
</div>
```

### 5. 利用媒体查询进行优化

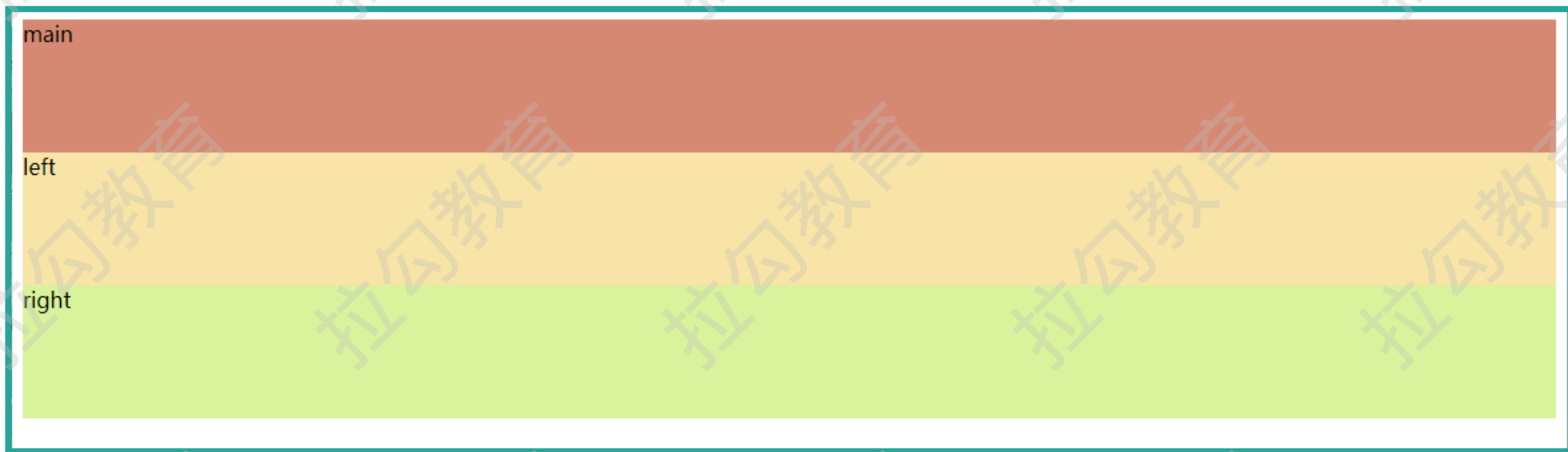
次要布局容器

主要布局容器

```
<style>
/*为了方便查看，给布局容器设置高度和颜色*/
.main, .left, .right {
  height: 100px;
}
.main {
  background-color: red;
}
.left {
  background-color: green;
}
```


```
background-color: green;
}

.right {
background-color: blue;
}
</style>
<div class="wrap">
  <main class="main">main</main>
  <aside class="left">left</aside>
  <aside class="right">right</aside>
</div>
```



```
<style>
  .main, .left, .right {
    float: left;
  }
</style>
<div class="wrap">
  <main class="main">main</main>
  <aside class="left">left</aside>
  <aside class="right">right</aside>
</div>
```

mainleftright





如果直接设置的话

布局容器 left 和 right 都会换行

所以需要通过设置**父元素 wrap 内边距**来压缩主要布局 main

给次要布局容器留出空间

同时通过设置**次要布局容器边距**

以及采用相对定位调整次要布局容器至两侧



```
<style>

.main, .left, .right {
  float: left;
}

.wrap {
  padding: 0 200px 0 300px;
}

.main {
  width: 100%;
}
```

```
.left {  
  width: 300px;  
  position: relative;  
  left: -300px;  
  margin-left: -100%;  
}  
  
.right {  
  position: relative;  
  width: 200px;  
  margin-left: -200px;  
}
```

```
width: 200px;  
margin-left: 200px;  
right: -200px;  
}  
</style>  
<div class="wrap">  
  <main class="main">main</main>  
  <aside class="left">left</aside>  
  <aside class="right">right</aside>  
</div>
```



```
<style>

.main, .left, .right {
  float: left;
}

.wrap {
  padding: 0 200px 0 300px;
}

.wrap::after {
  content: "";
  display: block;
  clear: both;
}
```

```
}  
  
.main {  
  width: 100%;  
}  
  
.left {  
  width: 300px;  
  position: relative;  
  left: -300px;  
  margin-left: -100%;  
}  
  
.right {  
  position: relative;
```

```
.right {  
  position: relative;  
  width: 200px;  
  margin-left: -200px;  
  right: -200px;  
}  
</style>  
<div class="wrap">  
  <main class="main">main</main>  
  <aside class="left">left</aside>  
  <aside class="right">right</aside>  
</div>
```





```
<style>
.main, .left, .right {
  float: left;
}

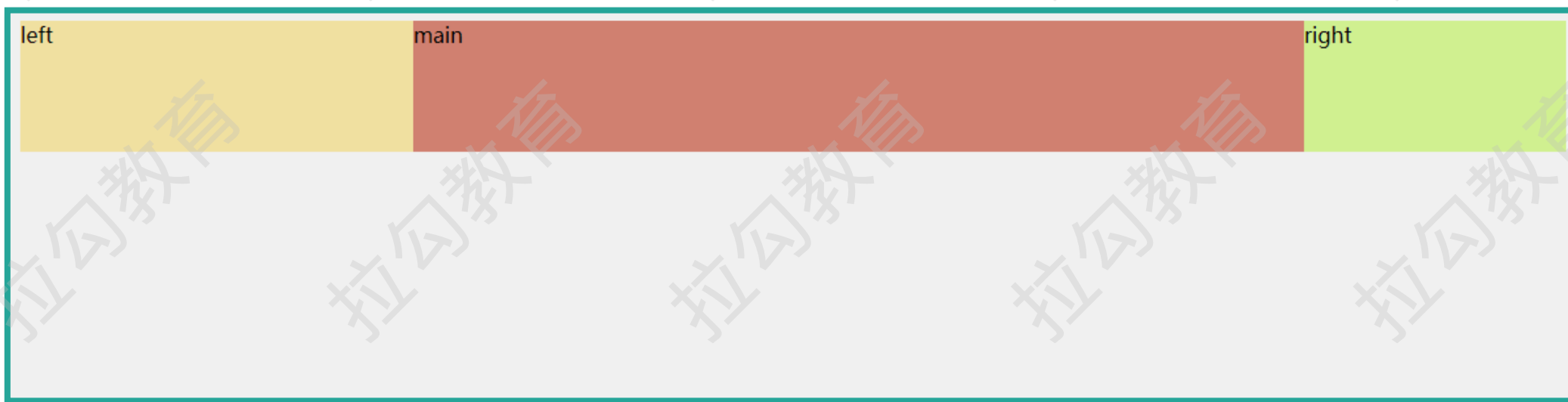
.wrap {
  padding: 0 200px 0 300px;
}

.wrap::after {
  content: "";
  display: block;
  clear: both;
}
```

```
.main {  
  width: 100%;  
}  
.left {  
  width: 300px;  
  position: relative;  
  left: -300px;  
  margin-left: -100%;  
}  
.right {  
  position: relative;
```

```
position: relative;
width: 200px;
margin-left: -200px;
right: 200px;
}
@media only screen and (max-width: 1000px) {
  .wrap {
    padding: 0;
  }
  .left {
    left: 0;
    margin-left: 0;
  }
}
```

```
}  
.right {  
  margin-left: 0;  
  right: 0;  
}  
}  
</style>  
<div class="wrap">  
  <main class="main">main</main>  
  <aside class="left">left</aside>  
  <aside class="right">right</aside>  
</div>
```



## 延伸1：垂直方向的布局

拉勾教育

— 互联网人实战大学 —

垂直方向有一种布局虽然使用频率不如水平方向布局高，但在面试中很容易被问到

这种布局将页面分成上、中、下三个部分

上、下部分都为固定高度，中间部分高度不定

- 当页面高度小于浏览器高度时  
下部分应固定在屏幕底部
- 当页面高度超出浏览器高度时  
下部分应该随中间部分被撑开，显示在页面最底部



## 延伸1：垂直方向的布局

拉勾教育

— 互联网人实战大学 —

```
<style>
.container{
  display: flex;
  height: 100%;
  flex-direction: column;
}
header, footer {
  min-height: 100px;
}
```



## 延伸1：垂直方向的布局

拉勾教育

— 互联网人实战大学 —

```
}  
  
main {  
  flex: 1;  
}  
  
</style>  
  
<div class="container">  
  <header></header>  
  <main>  
    <div>...</div>  
  </main>  
</div>
```

## 延伸1：垂直方向的布局

拉勾教育

— 互联网人实战大学 —

```
}  
</style>  
<div class="container">  
  <header></header>  
  <main>  
    <div>...</div>  
  </main>  
  <footer></footer>  
</div>
```

## 延伸1：垂直方向的布局

拉勾教育

— 互联网人实战大学 —

```
<style>
.container {
  box-sizing: border-box;
  min-height: 100vh;
  padding-bottom: 100px;
}

header, footer {
  height: 100px;
}
```

## 延伸1：垂直方向的布局

拉勾教育

— 互联网人实战大学 —

```
footer {  
  margin-top: -100px;  
}  
  
</style>  
<div class="container">  
  <header></header>  
  <main></main>  
</div>  
<footer></footer>
```

## 延伸2：框架中栅格布局的列数

拉勾教育

— 互联网人实战大学 —

**Bootstrap** 提供了 12 列栅格

**element ui 和 ant design** 提供了 24 列栅格



## 延伸2：框架中栅格布局的列数

拉勾教育

— 互联网人实战大学 —

**Bootstrap** 提供了 12 列栅格

从 **12 列** 说起

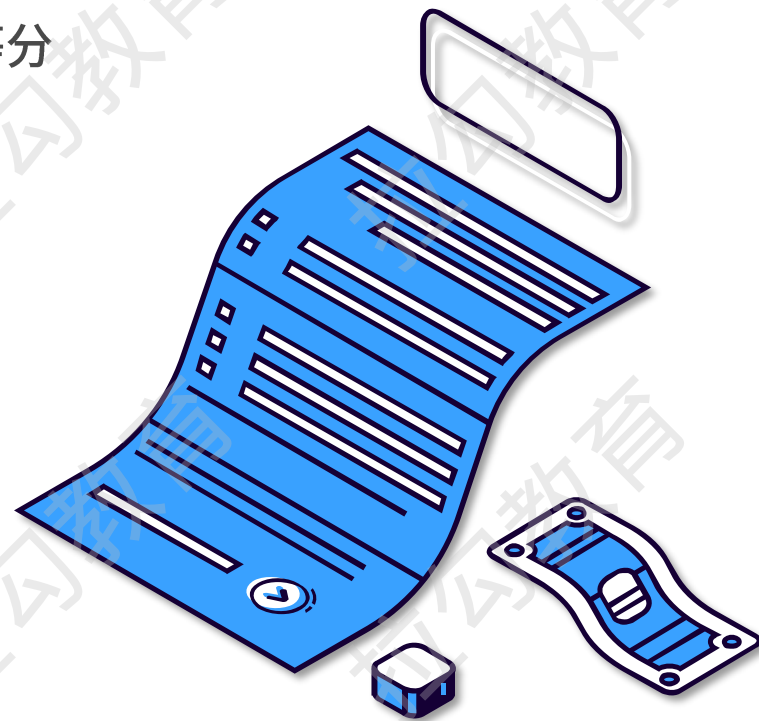
12 这个数字从数学上来说它具有很多约数 1、2、3、4、6、12

也就是说可以轻松实现 1 等分、2 等分、3 等分、4 等分、6 等分、12 等分

比例方面可以实现 1:11、1:5、1:3、1:2、1:1、1:10:1、1:4:1 等

如果换成 **10 或 8**，则可实现的等分比例就会少很多

而更大的 16 似乎是个不错的选择，但对于常用的 3 等分就难以实现



## 延伸2：框架中栅格布局的列数

拉勾教育

— 互联网人实战大学 —

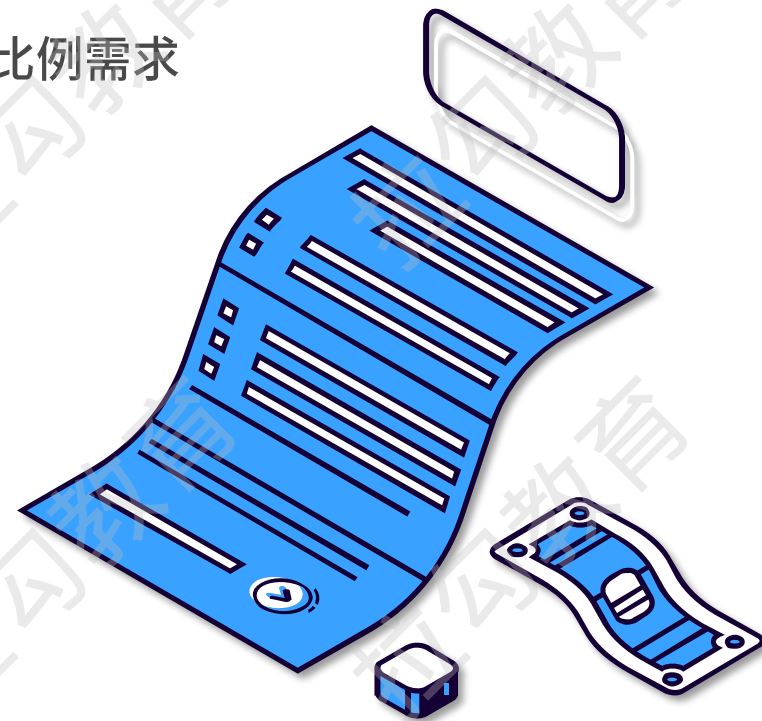
elment ui 和 ant design 提供了 24 列栅格

使用 24 列不使用 12 列

可能是考虑宽屏幕（PC 端屏幕宽度不断增加）下对 12 列难以满足等分比例需求

比如 8 等分

同时又能够保证兼容 12 列情况下的等分比例（方便项目迁移和替换）



学习了几种常见布局

包括单列、2 列、3 列及垂直三栏布局

对 2 列布局和 3 列布局实现方法归纳成了 5 个步骤

[课程代码点击下载](#)





你还想到了使用哪些方法来实现 2 列或 3 列布局 ?



Next: 第05讲 《如何管理你的 CSS 代码》

# 拉勾教育

— 互联网人实战大学 —



下载「拉勾教育App」  
获取更多内容