

HTML+CSS

HTML+CSS

前端工程师概述

前端工程师的技能规划

开发工具介绍

项目创建

HTML到底是什么

HTML 标签的语法

网页的基本标签HTML

Web三要素

标题标签

文字标签

段落标签

文字修饰

字体标签

特殊文字

上下角标

列表标签

无序列表

有序列表

自定义列表

图片标签

表格标签 【重点】

表格的标签

表格的合并

链接标签

iframe标签

iframe与a标签的结合

其它标签

进度条标签

计量器

跑马灯

map标签

属性面板集标签

换行标签

线条标签

表单标签

- 文本输入框
- 密码输入框
- 数字输入框
- 颜色输入框
- 日期输入框
- 时间输入框
- 日期时间输入框
- 单选按钮
- 多选按钮
- 文件选择框
- 下拉选择框
- 滑块输入框
- 文本域
- 按钮
 - 提交按钮
 - 重置按钮
 - 普通按钮
 - 双标签按钮

表单通用属性

CSS基础

- 什么是CSS
- 为什么需要CSS
- CSS的三个基本特征

- 统一性
- 便捷性
- 分离性

CSS基础语法

CSS选择器

- 标签选择器
- ID选择器
- class类选择器
- 子代选择器
- 后代选择器
- 分组选择器
- 相邻兄弟选择器

- 普通兄弟选择器
- 属性选择器
- 星号选择器
- 选择器的组合
- 伪类及伪元素选择器
 - 伪类选择器
 - 伪元素选择器
- 伪类与伪元素的区别
 - 理解伪类选择器
 - 理解伪元素选择器
 - 伪类与伪元素的注意事项
- 常见CSS样式（一）
 - 宽度高度
 - 颜色
 - 关于颜色值
 - 背景
 - 字体与文字
 - 边框
 - 普通边框
 - 圆角边框
 - 列表样式
 - 表格
 - 透明度设置
 - 隐藏与显示
 - 溢出处理
 - 鼠标光标
 - 计数器
 - 盒子阴影
 - 滤镜
- 盒子模型及元素类型
 - 关于布局的概念
 - 盒子模型
 - margin外间距
 - 关于auto的情况
 - margin的穿透与折叠
 - padding内间距
 - box-sizing属性

元素类型

- 块级元素
- 行内元素
- 行内块级元素
- 元素类型的转换
- 行内元素中的4条线

CSS高级特性

- 继承性
- 层叠性
- 优先级
 - 选择器相同的时候
 - 选择器不相同的时候

定位

- 相对定位
- 绝对定位
- 固定定位
- 静态定位
- 粘性定位
- 子绝父相
- 定位脱流的影响
- 浮动脱流与定位脱流的区别

CSS的变量及运算符

- CSS变量的使用
- CSS运算符

BFC的概念

- 什么是BFC
- 怎么形成BFC
- BFC的作用

CSS3过渡

- 过渡的属性
- 多个属性的过渡
- 全属性的过渡
- 不对称的过渡效果
- 不能过渡的属性
- 兼容性处理
- 案例

CSS3动画

- CSS动画定义
- CSS动画的使用
- 多个动画的使用
- CSS动画视觉差
- 阶段型动画
- CSS帧动画

- 多列布局

- 裁剪样式

- 矩形裁剪
- 圆形裁剪
- 椭圆裁剪
- 多边形裁剪
- 裁剪的注意事项

- CSS补充点

- img内容的显示方式
- 元素的宽高比
- link与@import的区别
- href与src的区别
- :valid伪类
- 表单元素的required属性

- 弹性盒子与弹性布局

- 弹性盒子
- 弹性盒子的特征
- 弹性切割与流式切割
- 弹性盒子属性的封装

- 弹性盒子伸缩值计算

- 弹性盒子扩张值计算
- 弹性盒子收缩值的计算
- 关于 `flex-basis` 的问题

- 扩展：如何形成全屏的盒子

- 移动端布局规范

- 调整设计稿
- 移动端布局之前的准备工作
- 开发中的注意事项

- Web前端的图标技术

- 第一代图标
- 第二代图标的使用

第三代图标：矢量图标

第四代：将矢量图标变成iconfont

iconfont图标的使用

本地图标上传到iconfont

移动端设计稿的计算

设计稿公式推断

问题一

问题二

问题三

问题四

响应式布局中的媒体查询

什么是响应式布局

媒体查询

屏幕尺寸的划分

媒体查询使用注意事项

扩展

设备的横竖状态改变

使用link来链接响应式CSS

案例

CSS中的grid布局

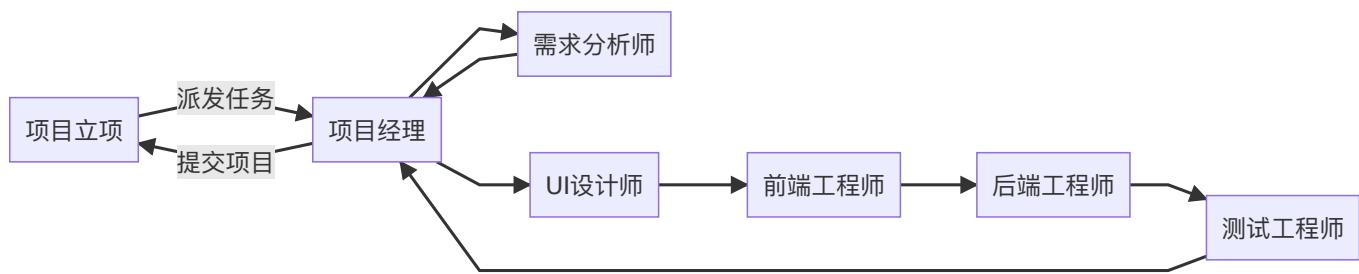
容器属性

项目属性

注意

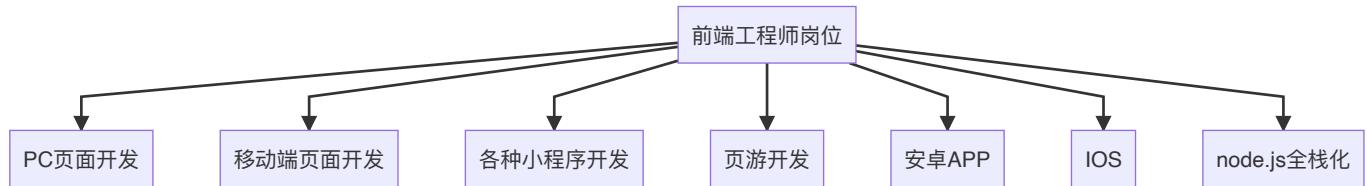
案例

前端工程师概述



1. 前端工程师是一个承上启下的工作岗位，承接UI，对接后台
2. 技术上面来说，现在流行前后端分离式开发，所以前后各司其职

前端工程师的技能规划



1. **HTML** 语言
2. **CSS** 样式与布局
3. **CSS3**，响应式，媒体查询，弹性布局，网格布局
4. **JavaScript**，**DOM**, **BOM**
5. **jQuery**, **bootstrap**, **tailwindcss**, **animation.css**, **autoanimate**
6. **Ajax**, **axios**, **fetch**, **websocket**
7. **TypeScript**, **ECMAScript 6**
8. **node.js**, **express**, **nest.js**, **JWT**
9. **webpack**, **vite**, **sass**, **postcss**, **babel**
10. **vue3**, **react**, **minprogram**
11. **SSR** 服务端渲染, **SEO**优化, 单点登录, 扫码登录
12. **uniApp**

开发工具介绍

1. **HbuilderX**
2. **VSCode**
3. **WebStorm**
4. **SublimeText**
5. **DreamWeaver**

前期我们会使用 **HbuilderX** 来进行开发，这个开发工具会持续到 **CSS** 结束，在进入JS之前会换成 **VSCode**

项目创建

在电脑上面新建一个空的文件夹，然后将这个空文件夹拖入到 **HbuilderX** 的图标上在，即可

在前端项目里面，前端项目是以文件夹为单位存在的，一个文件夹就可以认为是一个项目

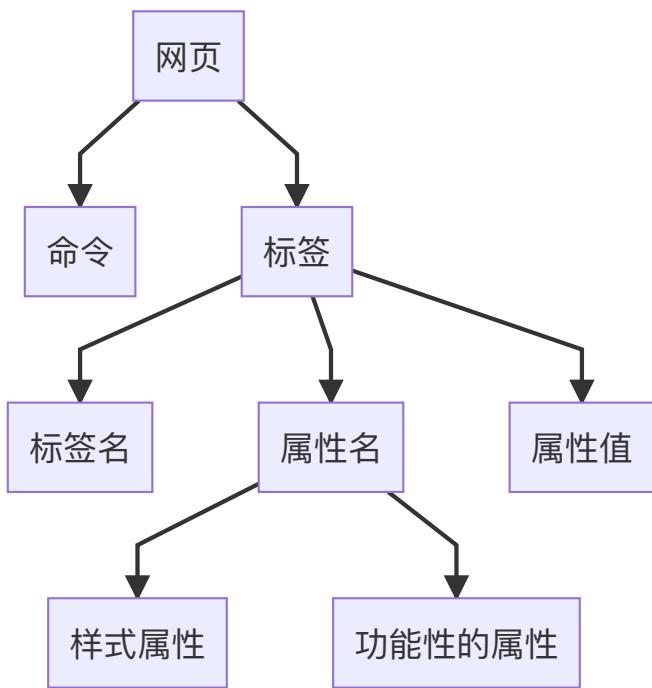


或使用快捷键 **Ctrl+B** 来显示左边的项目管理栏

HTML到底是什么

HTML全称叫超文本标记语言，网页的后缀名是 **.html**，它可以帮我们通过标记代码的形式来开发网页

目前最新的HTML版本叫HTML5，网页是由一系列的**标签**和命令来构成，这些标签我们称之为**HTML标签**



标签是用来说明作用的，属性是用来描述这个标签的特征的

HTML 标签的语法

双标签

```
<标签名 属性名="属性值"></标签名>
```

单标签

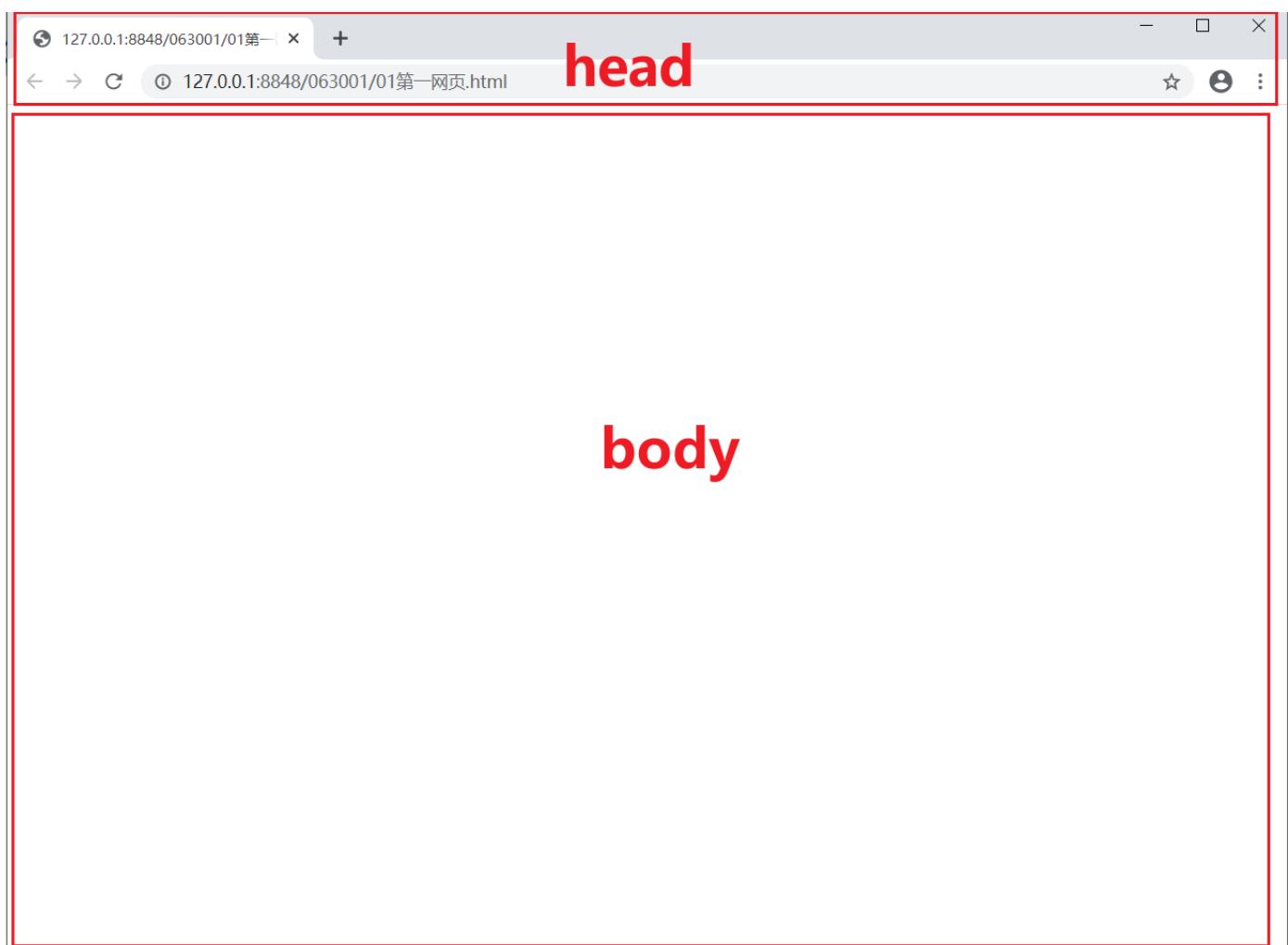
```
<标签名 属性名="属性值" />
```

在开发工具里面书写标签代码的时候，直接写标签名，然后按下键盘 **tab** 键

```
<!DOCTYPE html>
<!-- doc全称document
    type类型
    html网页
-->
<html>
    <head>
        <meta charset="UTF-8" />
        <title>标哥哥的文字</title>
    </head>
    <body>
    </body>
</html>
```

网页与人是一样的，它是由头和身体来组成的

html 相当于整个人， **head** 就是头部， **body** 就是身体

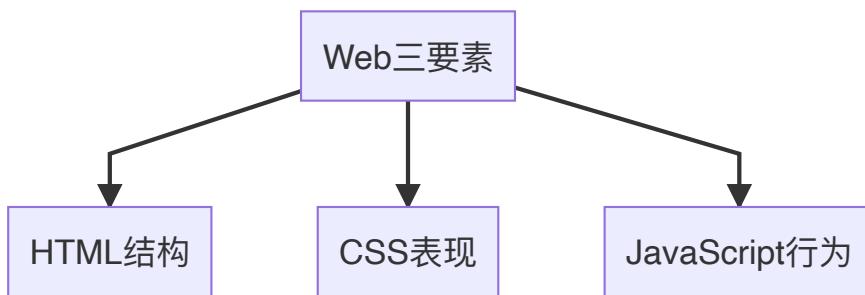


总结标签的特点

1. 大多数标签都是成双成对出现的，这些标签叫双标签，部分只有开始标签的叫单标签
2. 标签内部是可以有属性的，属性是写在开始标签里面的，属性是用于形容当前标签的特征
3. 标签是可以进行嵌套的

网页的基本标签HTML

Web三要素



1. **HTML** 是用于控制页面的结构的，如页面上面显示什么是由它决定的
2. **CSS** 表现指的是页面上面的内容显示成什么样式就是由它决定的

网页是由标签与命令来构成的，现在来了解一下有哪些标签

所有能够在word里面实现的效果，网页当中都可以实现

标题标签

标题是由 **h1 ~ h6** 所构成的标签

```
<h1 align="center">标题党的标题</h1>
<h2 align="right">大型记录片《标哥的暑假班开班了》 </h2>
<h3 align="left">三号标题</h3>
<h4>四号标题</h4>
<h5>5号标题</h5>
<h6>6号标题</h6>
```

标题是由h构成的，分别是1~6号标是，数字越大，字体越小，同时 **align** 用于设置标题的水平排列

文字标签

段落标签

```
<p align="center">这是第一个段落</p>
<p align="right">这是第二个段落</p>
```

1. 段落与段落上下会有间距
2. 段落是可以通过 align 来设置水平排列

文字修饰

```
还是那么的<b>帅气</b>
<hr>
这就是<u>我们</u>的老师，标哥哥呀
<hr>
还看到一个漂亮的小姐姐，它就是我们<i>桃子</i>老师
<hr>
瞧一瞧，看一看，原价<strike>998</strike>的衣服，现价只有9块8
```

在上面的代码里面，我们看到了很多文字修饰标签

- 加粗: ``
- 倾斜: `<i><cite>`
- 下划线: `<u><ins>`
- 删除线: `<strike>`

字体标签

```
<font color="red" size="7" face="华文行楷">
    9块8
</font>
```

- font标签用于设置一些特殊格式的文字
- `color` 用于设置文字的颜色
- `size` 用于设置文字的大小，它的值是1~7
- `face` 用于设置文字的字体样式

特殊文字

1. 版权标签 ©
2. 商标 ®
3. 空格
4. 度数 °
5. 小于 <
6. 大号 >

上下角标

```
我<sub>爱</sub>北京天安门， 天安门上太阳<sup>升</sup>  
x<sup>2</sup>+y<sup>2</sup>=z<sup>2</sup>
```

我_爱北京天安门， 天安门上太阳^升

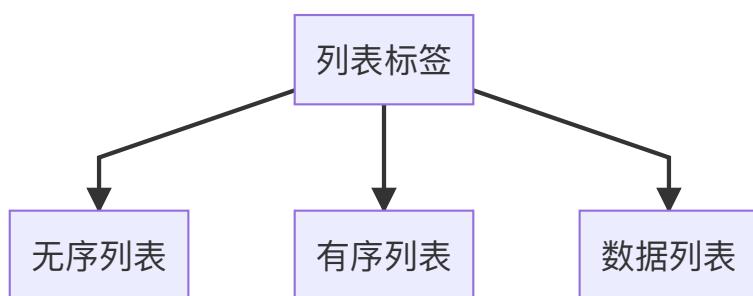
$$x^2 + y^2 = z^2$$

1. 上角标 **sup**
2. 下角标 **sub**

列表标签

引言：列表标签是一个父子标签

父子标签也叫严格型嵌套标签，指在的某个标签的内部只能嵌套指定标签



无序列表

全称叫 `unordered list` , 简称 `ul`

```
<ul>
  <li>张三</li>
  <li>李四</li>
  <li>标哥哥</li>
</ul>
```

-
- 张三
 - 李四
 - 标哥哥

如果想改变前面的符号 , 我们可以通过 `type` 属性来决定

1. `type="disc"` , 是默认值, 它是一个实心圆
2. `type="square"` 它会是一个正方形
3. `type="circle"` 它会是一个圆形

有序列表

全称叫 `ordered list` , 简称 `ol`

```
<ol type="I">
  <li>张三</li>
  <li>李四</li>
  <li>标哥哥</li>
  <li>陈文</li>
  <li>叶俊豪</li>
  <li>刘诗霞</li>
</ol>
```

如果想改变前面的序号的类型 , 我们可以使用 `type` 属性

1. `type="1"` 数字
2. `type="A"` 大写字母
3. `type="a"` 小写字母
4. `type="I"` 大写罗马文
5. `type="i"` 小写罗马文

自定义列表

```
<!-- data list 数据列表
data list data 数据列表里面的数据
data list title 数据列表里面的标题
-->
<dl>
  <dt>男生</dt>
  <dd>陈文</dd>
  <dd>叶俊豪</dd>
  <dt>女生</dt>
  <dd>刘诗霞</dd>
  <dd>陈怡静</dd>
</dl>
```

总结： `ul/ol/dl` 它们都是严格的父子标签， `ol/ul` 下面只能嵌套 `li`， `dl` 只能嵌套 `dd/dt`

案例

1. 第一组
 - 陈文
 - 叶俊豪
2. 第二组
 - 陈文
 - 叶俊豪
3. 第三组
 - 陈文
 - 叶俊豪

```
<ol>
  <li>
    第一组
    <ul type="square">
      <li>陈文</li>
      <li>叶俊豪</li>
    </ul>
  </li>
  <li>
    第二组
    <ul type="square">
      <li>陈文</li>
      <li>叶俊豪</li>
```

```
</ul>
</li>
<li>
    第三组
    <ul type="square">
        <li>陈文</li>
        <li>叶俊豪</li>
    </ul>
</li>
</ol>
```

上面的案例展示就是列表的嵌套关系，ul与ol的下面只能嵌套li，而li的下面才是可以嵌套任何东西

图片标签

在网页里面，我们一般会插入图片，我们可以插入项目中的图片，也可以直接插入互联网上面的图片，如果是插入项目里面的图片，则要把图片放在项目下面

最好是在项目的根目录下面新建一个文件夹叫 来存放图片

```

```

1. **src** 用于设置图片地址,除了使用本地地址以外，还可以使用网络地址
2. **alt** 后备属性，当**src** 不能正确的显示图片的时候就会显示**alt**里面的内容
3. **width** 用于设置宽度
4. **height** 用于设置高度
5. **border** 用于设置边框
6. **vspace** 用于设置垂直方向的间距
7. **hspace** 用于设置水平方向的间距

表格标签 【重点】

表格的标签

```
<table border="1" width="500" cellspacing="0" cellpadding="0">
  <colgroup>
    <col />
    <col />
    <col bgcolor="red" />
    <col />
  </colgroup>
  <tr>
    <th>姓名</th>
    <th>性别</th>
    <th>爱好</th>
    <th>地址</th>
  </tr>
  <tr>
    <td bgcolor="deeppink">张三</td>
    <td>男</td>
    <td>看书</td>
    <td>湖北省</td>
  </tr>
  <tr>
    <td>张三1</td>
    <td>男</td>
    <td>看书</td>
    <td>湖北省</td>
  </tr>
  <tr>
    <td>张三2</td>
    <td>男</td>
    <td>看书</td>
    <td>湖北省</td>
  </tr>
  <tr bgcolor="lightseagreen">
    <td>张三3</td>
    <td>女</td>
    <td>看书</td>
    <td>湖北省</td>
  </tr>
</table>
```

1. <table> 是用于创建表格的标签
2. <tr> 代表表格的行
3. <td> 代表表格的列
4. border 设置表格的边框
5. cellspacing 设置单元格与单元格的间距
6. cellpadding 用于设置单元格与内容的间距
7. width/height 用于设置表格或行或列的宽度及高度
8. align 用于设置水平排列 left/center/right
9. valign 用于设置垂直排列 top/middle/bottom
10. <colgroup> 列的组合
11. <col /> 某一列
12. <thead><tbody><tfoot>
13. <caption> 用于设置表格的标题

表格的合并

姓名	性别	爱好
陈文	男	读书
叶俊豪		睡觉
刘诗霞		逛街, 美甲

表格: table

表格的行: table row 简称 tr

表格里面的数据 table data

表格里面的标题 table header 简称 th

row行 rowspan

col列 colspan

合并span

```

<table border="1" width="500px" cellspacing="0px" cellpadding="0px">
  <tr>
    <th>姓名</th>
    <th>性别</th>
    <th>爱好</th>
  </tr>
  <tr>
    <td>陈文</td>
    <td rowspan="2">男</td>
    <td>看书</td>
  </tr>
  <tr>
    <td>叶俊豪</td>
    <td>睡觉</td>
  </tr>

```

```
</tr>
<tr>
    <td colspan="2">刘诗霞</td>
    <td>玩游戏</td>
</tr>
</table>
```

1. **rowspan** 行的合并
2. **colspan** 列的合并

链接标签

```
<a href="https://www.baidu.com" target="_self">百度一下，你就知道</a>
<a href="02.html">标哥的简历</a>
<a href="http://www.baidu.com" target="_blank">
    
</a>
```

1. **href** 属性，用于设置链接的地址，可以是远程网络地址，也可以是本地地址
2. **target** 属性，用于设置打开网页的方式
 - **_self** 在当前页面打开网页 【默认值】
 - **_blank** 在新的窗体打开网页
 - **_parent** 在父级打开网页
 - **_top** 在顶层打开网页
 - **frameName** 在某一个指定的 **iframe** 当中去打开

链接标签除了跳转到某一个链接地址，还可以跳转到某一个网页的某一个元素上面去，如下所示

```
<h2>第一章节</h2>
<h2 id="xxx">第二个章节</h2>
<h2>第三个章节</h2>
<a href="https://www.baidu.com">我要跳了</a>
<!-- 它不仅可以跳网络地址，还可以跳到指定的某一个元素那里去 --&gt;
&lt;a href="#xxx"&gt;我要去第二个章节&lt;/a&gt;</pre>
```

在网页里面，我们一般使用 # 代表 **id**，所以 **#xxx** 相当于 **id="xxx"**

通过上面的方法，我们还可以在跳转页面的时候指定到某一个元素上面去

```
<a href="04链接标签.html#xxx" target="_blank">我要去第四个页面的第二个章节</a>
```

iframe标签

iframe 全称可以理解为 `insert frame`，嵌入框架网页，这个标签可以在一个网页当中嵌入另一个网页

```
<iframe src="02.html" width="100%" height="500px" frameborder="0"></iframe>
```

1. `src` 用于设置要嵌入的网页地址
2. `width/height` 用于设置宽度与高度
3. `frameborder` 用于设置边框

iframe与a标签的结合

第一种结合方式

```
<a href="02.html" target="bbb">简历展示</a>
<a href="01.html" target="bbb">练习作业</a>
<iframe src="" name="bbb" width="100%" height="500px"></iframe>
```

基本信息			
姓名	张三	求职意向	.NET软件工程师
性别	男	现住城市	荆州
年龄	23	学历	本科
政治面貌	中共党员	专业	计算机科学与技术
健康状况	健康	爱好	篮球
QQ	123456	Email	123456@qq.com
联系电话			
英语水平	CET-4, 能够熟练的阅读专业英文文献, 具有一定的听说能力		

在这一种结合方式里面，我们把 `iframe` 的 `name` 设置在了 `a` 标签的 `target` 属性上面，这样在链接标签点击以后打开网页的时候，它会在指定的 `iframe` 里面打开

第二种结合方式

这一种情况就是 `iframe` 的下面嵌套 `iframe` 的时候

01.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>第一个网页</title>
  </head>
  <body>
    <h1>这是第一个网页</h1>
    <iframe src="02.html" width="100%" height="700"></iframe>
  </body>
</html>
```

02.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>第二个网页</title>
  </head>
  <body>
    <h2>第二个网页</h2>
    <a href="http://192.168.1.1" target="_self">百度一下，你就知道_self</a>
    <hr>
    <a href="http://192.168.1.1" target="_blank">网址2_blank</a>
    <hr>
    <a href="http://192.168.1.1" target="_parent">在父级打开</a>
    <hr>
    <iframe src="03.html" width="100%" height="400px"></iframe>
  </body>
</html>
```

03.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>第三个网页</title>
  </head>
  <body>
    <h3>第三个网页</h3>
    <a href="http://192.168.1.1" target="_parent">在父级打开</a>
    <a href="http://192.168.1.1" target="_top">我想在外上面打开</a>
  </body>
</html>
```

这是第一个网页

第二个网页

[百度一下，你就知道](#) [_self](#)

[网址2](#) [_blank](#)

[在父级打开](#)

第三个网页

[在父级打开](#) [我想在外上面打开](#)

第一个页面嵌套了第二个页面，第二个页面嵌套了第三个页面

1. [_parent](#) 指的是在父级打开
2. [_top](#) 直接跳到最顶层打开

其它标签

进度条标签

```
<progress min="0" max="100" value="90"></progress>
```

计量器

```
<!-- 计量器标签 -->
<!-- 假设我要形容一个学生的分数 -->
<meter min="0" max="100" value="90" low="60" high="80" optimum="100">
</meter>
<!-- 假设我想形容一个电脑硬盘使用量
optimum理想值
-->
<meter min="0" max="100" high="90" low="60" optimum="0" value="95"></meter>
```



跑马灯

```
<marquee direction="left" behavior="slide">
    <h2>套马的汗子你威武雄壮</h2>
</marquee>
```

1. `direction` 用于设置方向,值有 `left/right/up/down`
2. `behavior` 用于设置滚动行为, 值有 `scroll` 循环滚动, `slide` 滚动一次, `alternate` 交替运行
3. `scrollamount` 每次滚动的量是多少, 值越大, 滚动越快
4. `scrolldelay` 设置滚动的间隔时间, 以毫秒为单位, 值越大, 滚动得越快

map标签

这个标签是可以根据图片来结合使用的, 它可以把一张图片拆成不同的区域来点击链接效果

```

<map id="map1" name="map1">
    <!-- rect全称 rectangle 矩形
circle是一个圆
coords坐标
-->
    <area shape="circle" coords="250,250,50" href="http://192.168.1.1"
target="_blank" />
    <area shape="polygon" coords="500,0,0,500,500,500,400,400"
href="https://www.baidu.com" target="_blank" />
</map>
```

属性面板集标签

```
<fieldset>
  <legend>唐诗一首</legend>
  <p>长风破浪会有时</p>
  <p>直挂云帆济沧海</p>
</fieldset>
```

唐诗一首

长风破浪会有时

直挂云帆济沧海

换行标签

```
我应在江湖悠悠
<br />
饮一壶浊酒
```

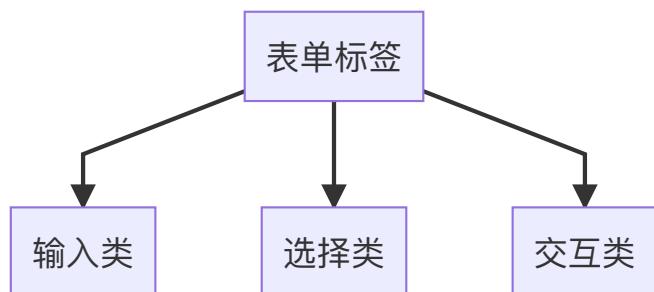
一个 **br** 标签就相当于一个回车

线条标签

```
<hr width="50%" align="center" size="20" color="red" />
```

表单标签

表单标签是在页面上面提供用户“输入”，“选择”，“交互”三大类型的标签



所有的表单标签都应该放在 **<form>** 标签下面

文本输入框

```
<input type="text" placeholder="请输入姓名" />
```

1. **type** 用于指定输入框的类型
2. **placeholder** 用于指定提示文字

密码输入框

```
<input type="password" placeholder="请输入密码">
```

在密码输入框里面，用户看不见自己输入的内容

数字输入框

```
<input type="number" min="16" max="30" step="0.5">
```

1. **min** 代表最小值
2. **max** 代表最大值
3. **step** 代表每次变化的量

颜色输入框

```
<input type="color">
```

日期输入框

```
<input type="date">
```

时间输入框

```
<input type="time">
```

日期时间输入框

```
<input type="datetime-local">
```

单选按钮

```
<input type="radio" name="aaa">男  
<input type="radio" name="aaa" checked>女
```

1. 如果想让单选变成一组，则它们必须具备相同的 `name` 值
2. 如果想让某一个被选中，默认加上 `checked="checked"`，因为属性名与属性值相同，所以可以简写成 `checked`，只保留属性名就可以了，这一种叫“单属性”

扩展

一般情况下，单选按钮 `radio` 都要与 `label` 标签结合在一起使用，效果如下

```
<label>  
    <input type="radio" name="sex">男  
</label>  
<label>  
    <input type="radio" name="sex">女  
</label>
```

在上面的情况里面，我们是直接把 `input` 的单选按钮放在 `label` 标签里面，这样就形成了一个整体，再去点击文字的时候，也相当于点击了单选按钮

还可以通过下面的方式去实现

```
<input type="radio" name="sex" id="nan" >  
<label for="nan">男</label>  
  
<input type="radio" name="sex" id="nv" >  
<label for="nv">女</label>
```

在这一种方式里面，我们没有把 `label` 包裹 `radio`，分开了，只需要在 `radio` 上面指定 `id`，然后在 `label` 上面的 `for` 属性里面指定绑定的 `id` 就可以了

多选按钮

```
<label>
  <input type="checkbox" name="hobby" checked>看书
</label>
<label>
  <input type="checkbox" name="hobby">睡觉
</label>
<label>
  <input type="checkbox" name="hobby" checked>玩游戏
</label>
```

文件选择框

```
<input type="file" multiple accept="image/png">
```

1. **multiple** 用于指定多选
2. **accept** 用于指定选择文件的类型

属性值

值	描述
audio/*	接受所有的声音文件。
video/*	接受所有的视频文件。
image/*	接受所有的图像文件。
MIME_type	一个有效的 MIME 类型，不带参数。请参阅 IANA MIME 类型 ，获得标准 MIME 类型的完整列表。



如果需要其它的类型，可以参考下面的网址

www.iana.org/assignments/media-types/media-types.xhtml

下拉选择框

差
插

稀土掘

```
<select multiple size="7">
  <optgroup label="东北">
    <option>黑龙江省</option>
    <option>吉林省</option>
    <option>辽宁省</option>
  </optgroup>
  <optgroup label="华北">
    <option selected>河北省</option>
    <option>山东省</option>
    <option>山西省</option>
  </optgroup>
</select>
```

1. 下拉选择器就是为了解决 `radio` 和 `checkbox` 选项太多的问题
2. `<optgroup>` 可以将选项进行分组
3. 默认选择某一个可以使用 `selected`
4. 如果需要多选，可以使用 `multiple`
5. 如果想一次性展示多个选项可以通过 `size` 去调整

滑块输入框

```
<input type="range" min="0" max="100" step="50">
```

1. `min` 代表最小值
2. `max` 代表最大值
3. `step` 每次更改的值的大小

文本域

```
<textarea rows="5" cols="40"></textarea>
```

按钮

提交按钮

提交按钮是一个交互式的按钮，它会将当前的表单提交到服务器

```
<input type="submit">
<input type="submit" value="登录">
```

提交按钮默认的文字就是提交，我们还可以通过 `value` 来设置里面的文字内容

```
<form>
  <p>
    账号: <input type="text" placeholder="请输入账号">
  </p>
  <p>
    密码: <input type="password" placeholder="请输入密码">
  </p>
  <p>
    <input type="submit">
  </p>
</form>
```

在上面的案例里面，如果我们点击了提交按钮以后，它会把当前的填写内容提交到服务器

重置按钮

重置就是会将当前表单的值恢复成初始值

```
<input type="reset">
<input type="reset" value="我要重来">
```

重置按钮默认的文本内容是“重置”，还可以通过 `value` 来设置里面文字

普通按钮

这一种按钮是只具备按钮的样式，但是不具备任何功能，这一种按钮就是一个普通按钮

```
<input type="button" value="普通按钮">
```

你们目前使用得最多的就是这个按钮

我们一般在使用的时候不使用上面的方式(`input`)来创建按钮，而使用使用双标签来创建按钮

双标签按钮

```
<button type="submit">  
    <i>提交按钮</i>  
</button>  
<button type="reset">  
    <font color="red">重置按钮</font>  
</button>  
<button type="button">  
    <b>普通按钮</b>  
</button>
```

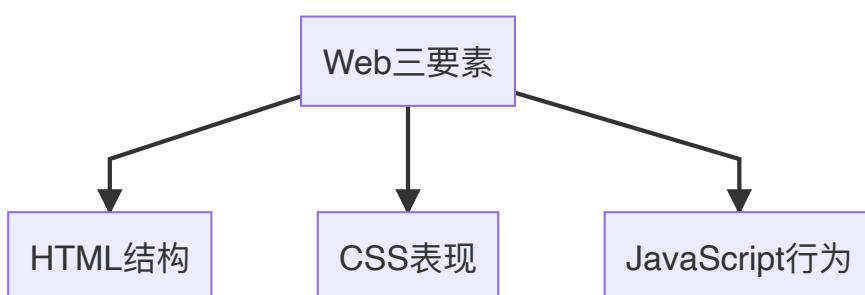
双标签按钮的好处就是可以在里面嵌入自己所需要的东西

表单通用属性

1. `value` 用于设置表单的值
2. `placeholder` 占位符提示
3. `readonly` 只读
4. `disabled` 禁用
5. `maxlength` 输入框的最大长度
6. `required` 用于设置当前字段必填
7. `checked` 用于单选或多选里面默认的选中
8. `selected` 用于下拉选项框里面的默认选中
9. `multiple` 用于 `select` 和 `file` 里面的多选

CSS基础

之前就给同学们提过，Web三要素



1. `HTML` 是网页的结构，网页有多少内容，显示什么内容都是由HTML来决定的
2. `CSS` 是网页的表现形式，网上面的内容或元素表现什么样子是由它决定【我们通常说一个网页很丑，原则是因为CSS没有写好】

3. **JavaScript** 是用户与页面之间的交互行为，行为会产生后果，这个后果就后面的事情【后面在研究】

什么是CSS

CSS的全称叫 **cascading style sheet**，“层叠样式表”，它是一套完整的技术体系，它以一种前所未有的能力来设置网页上面的元素样式，目前已经发展到了第3个版本【所以经常会有一词叫HTML5+CSS3】

CSS是由一个组织 **W3C** 制定并发布的

为什么需要CSS

在以前其它并没有CSS，所以的一切都是由HTML来完成的，但是在我们使用HTML的时候，我们经常会发现有以下几个问题

IT职业技能			
技能分类	技能名称	从事时间(月)	熟练程度
编程语言	C#	18	熟练 (+++)
	C++	10	熟练 (++)
	Matlab	6	熟练 (++)
	VB.Net	临时用到	会用
数据 库	SQLServer 2000	12	熟练 (++)
	SQLServer 2005	18	熟练 (++)
	Oracle 9i	2	熟练
Web技术	Asp.Net	12	熟练 (+++)
	javascript	8	熟练 (+)
	XML	3	熟练 (+)
	Ajax, Jquery, Webservice	4	熟练 (++)
其 它	Wpf	1	熟悉
	Wcf, Servelight		了解

我们之前在做这个作业的时候映到了一个问题

问题：我们发现所有的表格的单元格内容都要居中，这个时候我要对所有的 **td** 添加 **align="center"**，这样**代码的冗余量非常高**

用户信息注册

请填写用户信息

用户名	<input type="text" value="请输入用户名"/>
密码	<input type="password" value="请输入密码"/>
性别	<input type="radio"/> 男 <input type="radio"/> 女
爱好	<input type="checkbox"/> 看书 <input type="checkbox"/> 睡觉 <input type="checkbox"/> 购物 <input type="checkbox"/> 吃饭
籍贯	<input type="text" value="湖北省"/>
头像	<input type="file" value="选择文件"/> 未选择文件
<input type="button" value="注册"/> <input type="button" value="取消"/>	

在上面的家庭作业里面，我们又遇到了几个问题

1. 我们无法设置 `fieldset` 的宽度，我们也无法设置 `input` 表单的宽度
2. 代码的冗余量也过高
3. 我们发现 `table` 可以设置 `width` 而其它的元素又不能设置 `width`，这种情况对我们记忆某一个属性会非常困难【它没有一个统一的东西让我去记忆】

上面的问题由来已久，所以W3C就专门针对这种情况推出了一个新的技术，这个技术叫 `CSS`

CSS的三个基本特征

统一性

统一性是指用统一的属性来设置元素样式

CSS将之前HTML上面所有样式属性全部都废弃掉，转而使用了个 `style` 属性去代替【所有的HTML元素都有 `style`】，并且使用属性的属性名来设置所有元素的统一的样式

```
<fieldset style="width: 300px;">
    这是一个盒子
</fieldset>
<hr style="width: 300px;">
<table style="width: 300px;" border="1">
    <tr>
        <td>这也是一个表格</td>
    </tr>
</table>
```

便捷性

CSS提供了丰富多样的选择器，能够让我们单个或批量的选取我们要设置样式的元素，来进行样式设置，可以极大的简化我们的代码量，减少代码的冗余量

```
<style>
    p{
        color: red;
    }
</style>
```

上面的代码就是用于设置页面上面所有的 **p** 标签都为红色

分离性

第一种分离性的体现

我们在书写样式的时候，我们之前在统一性里面讲到了，所有的元素上面都追加了一个 **style** 的属性，但是这样仍然无法解决一些特殊情况下问题，所以CSS可以将 **style** 属性里面的内容分离到 **<style>** 标签当中去

```
<p style="color: red;">这是第一个段落</p>
```

我们把style属性提出来

```
<style>
    p{
        color: red;
    }
</style>
<p>这是第一个段落</p>
```

第二种分离性的体现

当不同的页面要使用不同的样式的时候，我们就要考虑到能否再次分离

第一个页面

```
<p>这是第一个段落</p>
```

第二个页面

```
<p>这是第二个页面上面的段落</p>
```

如果将2个页面使用同一个样式

```
p{  
    color: red;  
}
```

我们可以在项目下面新建一个 `css` 的文件夹，然后在这个文件夹的下面新建一个 `a.css` 的文件，把公共样式放入到这个 `a.css` 里面

现在结构代码就在HTML文件里面，样式代码 CSS就在 `a.css` 文件里面，它们就分开了，现在我们只需要将它们建立关系

```
<link rel="stylesheet" type="text/css" href="css/a.css" />
```

或

```
<style>  
    @import url("css/a.css");  
</style>
```

CSS基础语法

1. 行内样式

```
<h2 style="color: red; background-color: blue;">孤勇者</h2>  
<标签 style="CSS属性名:CSS属性值;CSS属性名:CSS属性值;"></标签>
```

行内样式情况使用比较少，它是直接把样式写在了 `style` 标签里面

优点：简单

缺点：不利于代码分离，会产生冗余量

2. 内部样式块

```
<style>
  p {
    color: deeppink;
  }
  选择器 {
    CSS属性名:CSS属性值;
    CSS属性名:CSS属性值;
  }
</style>
```

内部样式块就是把CSS的样式代码写在 `<style>` 标签里面，在大多数情况下，我们使用的都是内部的样式块

优点：可以使用丰富多样的选择器来批量或单个选取元素以后设置样式

缺点：不够直观，比行内样式写起来稍微的麻烦一点

3. 外部样式表

当一个页面上面的样式代码过多，或多个页在要使用同样的样式的时候，我们就会把样式代码写在一个单独的 `css` 文件里面，如 `a.css`，然后再通过标签或命令导入【在基础环节我们暂不考虑】

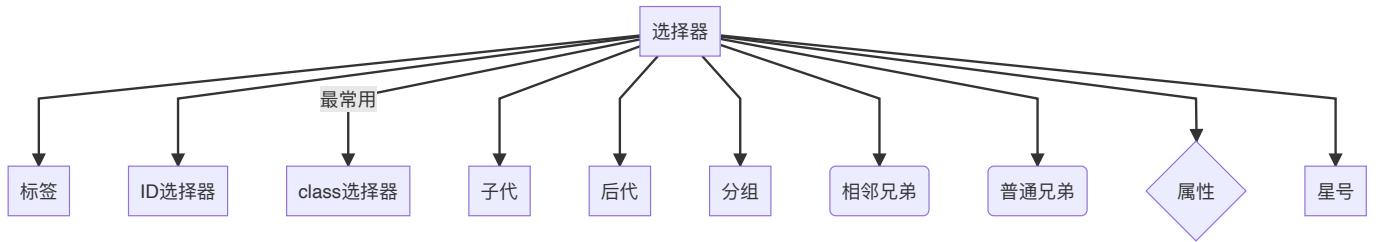
```
<link rel="stylesheet" type="text/css" href="css/a.css" />
<style>
  @import url("css/a.css");
</style>
```

4. CSS注释

在CSS区域进行注释是如下格式

```
<style>
  /* 这才是CSS的注释 */
</style>
<!-- 这是HTML的注释 -->
```

CSS选择器



CSS的选择器是CSS便捷性的一种体系，如果我们要设置页面上面元素的样式就必须先通过选择器选中元素

标签选择器

在HTML里面我们已经学过很多个标签名，它是HTML的标签来进行选择器

```

<style>
  p{
    color: red;
  }
  h2{
    color: pink;
  }
</style>
<p>第一个段落</p>
<p>标哥哥的选择器</p>
<h2>刘诗霞</h2>

```

ID选择器

在网页的标签上面，我们可以对每个元素都指定一个编号，这个编号就是一个ID，**ID不能重复**，在CSS里面，它使用 # 去表示

```

<style>
  #aaa {
    color: red;
  }
</style>
<p>陈文</p>
<p id="aaa">叶俊豪</p>

```

class类选择器

class类选择器可以把它看成是一个微信群，类选择器使用`.`来表示

```
<style>
    /* 将2个女生设置为红色
    类选择器当成 微信群
    */
    /* 相当于创建了一个微信群，群名叫aaa */
.aaa{
    color: red;
}
.bbb{
    font-size: 36px;
}
</style>
<p class="bbb">陈文</p>
<p>叶俊豪</p>
<p class="aaa bbb">刘诗霞</p>
<p class="aaa">陈怡静</p>
```

1. `class` 选择器相当于是一个微信群，一个class样式就是一个微信群
2. 下面的元素就相当于人，人可以加入多个微信群，一个微信群也可以加入多个人

正是因为有这个特点，所以类选择器会非常灵活，它也是我们平常开发中使用得最多的一种选择器

上面的三个选择是CSS的是3个基本选择器

子代选择器

子代选择器使用`>`来表示

```
<style>
    /* 我只想fieldset内部的元素变成红色 */
    /* 我全都要 */
    fieldset>h2{
        color: red;
    }
</style>
<fieldset>
    <h2>张无忌</h2>
    <h2>赵敏</h2>
    <h2>周芷若</h2>
</fieldset>
<h2>张翠山</h2>
```

1. 子代选择器使用 `>` 来表示
2. 子代选择器 `>` 的左右两边可以是任意的基础选择器

```
#aaa>.bbb{}
#aaa>label{}
#aaa>#bbb{}
.ccc>h2{}
```

同时，子代选择器可以不停的向下面选

```
<style>
    /* 想把“张某人”设置为粉色 */
    fieldset>p>label{
        color: red;
    }
</style>
<fieldset>
    <h2 class="aaa">张无忌</h2>
    <h2>赵敏</h2>
    <h2>周芷若</h2>
    <label>小昭</label>
    <p>
        张无忌的儿子叫<label>张某人</label>
    </p>
</fieldset>
<h2 class="aaa">张翠山</h2>
```

后代选择器

后代选择器使用空格来表示

```
<style>
  fieldset a{
    color: red;
  }
</style>
<fieldset>
  <a href="#">百度一下</a>
  <p>
    有问题也可心找
    <a href="#">标哥的博客</a>
    <b>还可以<a href="#">谷歌一下</a>
    </b>
  </p>
</fieldset>
<a href="#">优酷一下</a>
```

后代选择器与子代选择器有一个共同的特点，它们是可以由任何的基础选择器来组合

```
#aaa .b{}
.c a{}
#ddd label{}
```

分组选择器

分组选择器可以将多个基础类型的选择器进行分组，它使用 , 逗号来表示

```
<style>
  /* 将h1和h2都设置为红色 */
  h1, h2{
    color:red;
  }
</style>
<h1>爸爸的爸爸叫爷爷</h1>
<h2>妈妈的爸爸叫外公</h2>

<h1>妈妈的妈妈叫外婆</h1>
<h2>爸爸的妈妈叫奶奶</h2>
```

逗号选择器的左右可以是任意类型选择器，也可以叠加嵌套

```
h1, h2{}  
#a, #b{}  
#a, .b{}  
h1, #a{}  
h1, #a, .b{}
```

相邻兄弟选择器

相邻兄弟选择器是通过哥哥找弟弟，它使用 **+** 来表示

```
<style>  
    /* 请将紧挨着h2标签的后面一个p标签设置为红色 */  
    h2+p{  
        color: red;  
    }  
</style>  
<p>标哥哥</p>  
<h2>我应在江湖悠悠</h2>  
<p>陈怡静</p>  
<p>刘诗霞</p>  
<h3>叶俊豪</h3>  
<h2>饮一壶浊酒</h2>  
<p>陈文</p>
```

1. 相邻兄弟选择器的左右也可以是任何基础类型的选择器
2. 相领选择器也可以叠加使用

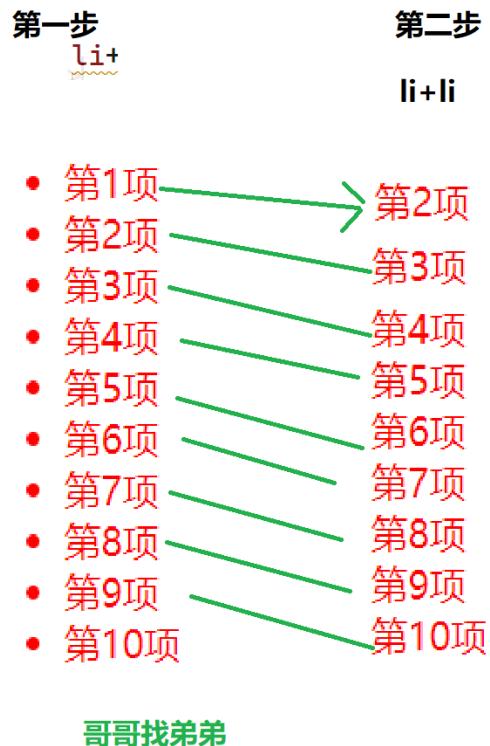
案例

```
<style>  
    li+li{  
        color: red;  
    }  
</style>  
<ul>  
    <li>第1项</li>  
    <li>第2项</li>  
    <li>第3项</li>  
    <li>第4项</li>
```

```
<li>第5项</li>
<li>第6项</li>
<li>第7项</li>
<li>第8项</li>
<li>第9项</li>
<li>第10项</li>
</ul>
```

```
... ...
li+li{
  /* color: red; */
}
li
}
```

```
<li>第1项</li>
<li>第2项</li>
<li>第3项</li>
<li>第4项</li>
<li>第5项</li>
<li>第6项</li>
<li>第7项</li>
<li>第8项</li>
<li>第9项</li>
<li>第10项</li>
...
```



普通兄弟选择器

普通兄弟与相邻兄弟是差不多的，它使用 `~` 来表示，我们通过下面的案例来了解对比

```
<style>
    /* 请将h2标签后面的所有的p标签都设置为红色 */
    h2~p{
        color: red;
    }
</style>
<p>标哥哥</p>
<h2>我应在江湖悠悠</h2>
<p>陈怡静</p>
<p>刘诗霞</p>
<h3>叶俊豪</h3>
<h2>饮一壶浊酒</h2>
<p>陈文</p>
```

属性选择器

属性选择器我们使用 [] 去表示

```
<style>
    /* 将所有colspan合并的格子变成红色 */
    [colspan]{
        color: red;
    }
    [rowspan]{
        background-color: deeppink;
    }
</style>
<table width="500px" border="1">
    <tr>
        <td colspan="2">1</td>

        <td>3</td>
    </tr>
    <tr>
        <td rowspan="2">1</td>
        <td>2</td>
        <td>3</td>
    </tr>
    <tr>
        <td colspan="2">2</td>
    </tr>
```

```
<tr>
    <td colspan="3">1</td>
</tr>
</table>
```

属性选择器的高级用法

1. 属性选择器可以设置具体的值来过滤

```
<style>
/* 我想将www.softeem.com的网址设置为红色 */
[href="http://www.softeem.com"]{
    color: red;
}
</style>
<a href="http://www.softeem.com">软帝</a>
<a href="http://www.softeem.xin">标哥</a>
<a href="http://www.softeem.com">又是软帝</a>
<a href="http://www.softeem.top">柴哥</a>
```

2. 属性值是以某一个具体的值结尾

```
<style>
/* 我希望将所有.jpg的图片高度设置为100px */
[src$=".jpg"]{
    height: 100px;
}
</style>




```

3. 属性值以某一个具体的值开头

```
<style>
    /* 我希望所有image目录下面的图片都设置为width:200px */
    [src^="image"]{
        width:200px;
    }
</style>




```

4. 属性值里面包含了某一个具体的值

```
<style>
    /* 希望所有包含softeem的链接字体设置36px */
    [href*="softeem"]{
        font-size: 36px;
    }
</style>
<a href="http://www.softeem.com">软帝</a>
<a href="http://www.softeem.xin">标哥</a>
<a href="http://www.softeem.com">又是软帝</a>
<a href="http://www.softeem.top">柴哥</a>
<a href="http://www.baidu.com">百度</a>
```

星号选择器

星号选择器代表任意选择器，可以选择所有的内容，它使用 `*` 表示

```
<style>
    /* 希望将fieldset里面所有的元素都设置为红色 */
    fieldset>*{
        color: red;
    }
</style>
<p>张三</p>
<fieldset>
    <h2>李四</h2>
    <h1>王五</h1>
    <a href="#">赵六</a>
</fieldset>
```

总结：所有的选择器都有一个定论，只能是父级找子级（后代），哥哥找弟弟

选择器的组合

官网的说法叫选择器的并联

当某情况下，一个选择器不能完成我们的效果的时候，我们可以将多个选择器并联，以达到目的

```
<style>
    /* 现在，我想将李四设置为红色 */
    /* h2{
        color: red;
    } */
    /* .a{
        color: red;
    } */
    h2.a{
        color: red;
    }
</style>
<h2 class="a">李四</h2>
<p class="a">张三</p>
<h2>王五</h2>
```

伪类及伪元素选择器

伪类与伪元素选择器它是一种特殊的选择器，后天有一个单独的来讲这个区别

伪类与伪元素选择器是用于形容一个元素的**特殊状态或特殊位置**，它们会以：`:` 或 `::` 来表示

伪类选择器

1. `:link` 链接标签被访问之前的状态
2. `:visited` 链接标签被访问之后的状态
3. `:hover` 鼠标放上去以后的特殊
4. `:active` 它指一个元素被鼠标点击并且没有松开的时候的特殊状态

我们的链接标签 `a` 标签它的四个状态是有顺序的，不能随便写，它的顺序采用 `lvha` 来进行，可以理解成“由爱(love)生恨(hate)”

这里还有一个细节给大家说一下，`:link` 和 `:visited` 是链接标签独有的状态，而 `:hover` 和 `:active` 是所有元素都会有状态

5. `:first-child` 第一个子元素
6. `:last-child` 最后一个子元素
7. `:nth-child` 用于选定指定的子元素

`:nth-child(xn+y)` 它内部有一个公式

- 当 `x=0` 时，代表只选定第y个元素
- 当 `x=1` 时，代表从第y项开始，一直选中到最后
- 当 `x=-1` 时，代表从第y项开始，一直到最前面
- 当 `x>1` 时，代表将子元素按x个分成一组，然后再取这个分组里面的第y个
- 这里还可以使用特殊的单词，如 `even` 代表偶数，`odd` 就是奇数

所有以 `child` 结尾的选择器有一个共同的特点，先确定子元素的位置，再确定子元素的类型，这个选择多半适用于子元素相同的时候

8. `:nth-of-type(xn+y)` 用于选中指定的子元素，先确定元素的类型，再确定元素的位置，这个选择器多半适用于子元素不相同的时候
9. `:not` 排除某一种情况
10. `:read-only` 代表表单素被只读的时候
11. `:disabled` 代表的就是表单元素被禁用的时候
12. `:checked` 单选框或复选框 被选中以后的特殊状态
13. `:target` 目标元素的特殊状态
14. `:focus` 当一个元素获得焦点的特殊状态
15. `:focus-within` 某一个元素内部的元素获取焦点，反过来操作外边的元素状态

伪元素选择器

1. `::before/::after` 在当前元素的内部的最前面或最后面追加内容，通过 `content` 属性去追加
 - 通过 `::before/::after` 追加的内容无法通过鼠标选中
 - 通过 `content: url("img/man4.webp");` 可以追加一个图片

- 通过 `content: attr(target);` 来追加某一个元素内部的属性
2. `::first-line` 一个段落的第一行文字
 3. `::first-letter` 段落里面的第一个文字
 4. `::placeholder` 用于设置输入框里面提示符占位信息
 5. `::selection` 选中文字以后

伪类与伪元素的区别

之前已经跟同学样讲过了， 伪类与伪元素其实就是元素的特殊状态或特殊的位置

其实我们可以这么去理解， 它本质上面就类选择器和元素选择器， 而元素就是标签， 所以可以认为是类选择器和标签选择器

进而还可以这么理解

伪类：一个假的类选择器

伪元素：一个假的元素选择器

理解伪类选择器

伪类选择器就是原本可以通过类选择器来实现的， 现在不需要这个类， 转而使用一个看不见的类选择器， 这个就叫伪类选择器

现在我们有下面这个效果需要实现

My name is BiaoGege

```
<p>
  <label>My</label>
  <label>name is BiaoGege</label>
</p>
```

这个时候我们肯定会想到 `::first-child`

```
/* 请将第一个元素My设置为红色，怎么办 */
p>label:first-child{
    color: red;
}
```

但是这个 `:first-child` 到底是伪类还是伪元素呢？【我们现在还不知道】

？不知道的东西我们先不用

```
<style>
    p>label.aaa{
        color: red;
    }
</style>
<p>
    <label class="aaa">My</label>
    <label>name is BiaoGege</label>
</p>
```

现在来进行对比：原来需要添加 `class="aaa"` 的现在我们可以不用不回了，我们可以直接使用 `:first-child` 来实现，也就是说 `:first-child` 所实现的功能其实就是 `class` 类选择器 `.aaa` 的功能，所以我们可以认为

`:first-child==.aaa`

总结：原本需要通过类选择器去实现的效果，现在不需要了，转而使用了 `:` 这种选择器，那么，这种选择器就叫伪类选择器

理解伪元素选择器

伪元素选择器就是伪标签选择器

现在我们通过下面的案例来了解伪元素

My name is BiaoGege

```
<p>
  <label>My</label>
  <label>name is BiaoGege</label>
</p>
```

现在，我们可以通过下面的方式来完成

```
<p>
  <label><b>M</b>y</label>
  <label>name is BiaoGege</label>
</p>
```

我们首先在 M 的字母上面添加了一个 B 标签，将其包裹，然后再添加了CSS代码

```
/* 将里面的第一个字母M变红，怎么办 */
/* 请不考虑使用伪类或伪元素，怎么办 */
p>label>b{
  color: red;
}
```

? 问题就在这里，我们能不能添加标签（元素），来实现这个效果，这个时候我们就要借用 `:first-letter`

```
<style>
  /* 将里面的第一个字母M变红，怎么办 */
  /* 请不考虑使用伪类或伪元素，怎么办 */

  p::first-letter{
    color: red;
  }
</style>
<p>
  <label>My</label>
  <label>name is BiaoGege</label>
</p>
```

综上代码展示我们发现了一个特点，`::first-letter` 相当于在第一个文字的前后添加了标签，但是这个标签你们看不到，所以 `:first-letter` 就是一个伪标签（伪元素）

总结：原来需要添加一个标签才能实现的效果，现在不需要了，转而通过添加一个 `::` 就可以实现的，我们把这一类选择器叫伪元素选择器

伪类与伪元素的注意事项

1. 伪类选择器只要不冲突就可以叠加使用

```
<ul>
    <li>第1项</li>
    <li>第2项</li>
    <li>第3项</li>
</ul>
```

下面的效果是可以的

```
/* 只在第一个li上面hover的时候, 我就变成红色 */
ul>li:first-child:hover{
    color: red;
}
```

下面的效果也可以

```
/* ul在hover的时候将第一个li元素设置为红色 */
ul:hover>li:first-child {
    color: red;
}
```

下面的效果不可以

```
ul>li:hover:first-child{
    color: red;
}
```

2. 在一次选择里面, 伪元素选择器不能叠加
3. 在一次选择里面, 当伪类与伪元素同时存在的时候, 伪元素只能放在最后
4. 一般情况下, 伪类我们使用 : 来表示, 伪元素我们使用 :: 来表示。

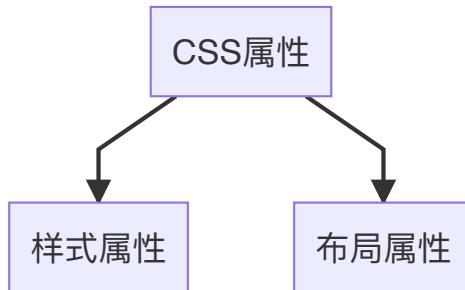
但是目前的浏览器都有兼容性了, 所以有些选择器无论使用单冒号还是双冒号都可以了

常见CSS样式（一）

之前讲CSS的基本特性的时候提到过, 选择器是CSS最基本的, 通过选择器我们可以快速的选择页面上面元素, 但是选中元素以后最终我们还是要设置元素的样式, 这些元素的样式是通过统一的CSS属性名来设置的, 所以CSS的属性名是统一性的一种体现

现在我们就来学习一下常见的CSS属性名（常见CSS样式）

CSS的属性又分为2大类



宽度高度

1. `width/height` 用于设置元素的宽度与高度
2. `max-width/max-height` 用于设置盒的最大宽度与最大高度，盒子的真实高度/宽度不能大于这个值
3. `min-width/min-height` 用于设置盒子的最小宽度与最小高度，盒子的真实高度/宽度不能小于这个值

颜色

1. `color` 用于设置前景色

这个属性比较简单，关键就是后面的性

关于颜色值

目前大多数浏览器能够使用的颜色有三种

1. 枚举色

枚举色就可以叫得出名字的颜色，如 `red/blue/yellow.....`，它可以直接使用英文单词来表示，这种颜色非常直观，缺点也很明显，不可以将所有的颜色都列举出来

2. 十六进制色

上面的枚举色不能够将所有的颜色表示出来，所以我们需要使用数字来精确的表示，十六进制色就是使用十六进制的数值来表示颜色，它的格式是 `#红色绿色蓝色`

十六进制是从 `0~9, A-F`



代表十六进制，前2个值代表红色的权重，中间2个值代表绿色的权重，最后2个值代表蓝色的权重，红绿蓝的三个颜色就可以构成世间所有的颜色，权重值越大，颜色值越深

如红色可以使用 `#FF0000`，蓝色 `#0000FF`，白色 `#FFFFFF`，黑色 `#000000`，十六进制颜色不区分大小写

3. `rgb`三原色

从上面的16进制里面我们了解到了，所有的颜色都通过 红、绿、蓝来组合而成的，我们可以使用红绿蓝的数值去表示，但上面的是16进制的表示，其实还可以使用10进制表示法，它从 `0~255`

`rgb(红色, 绿色, 蓝色)`

如红色 `rgb(255, 0, 0)`，绿色: `rgb(0, 255, 0)`，白色 `rgb(255, 255, 255)`

4. `rgba` 透明度

这个值与上面的 `rgb` 非常像，只是在后面追加了一个 `alpha` 值，`alpha` 值如果为0代表全透明，如果为1代表全不透明，如果介于0~1之间，则是半透明

```
background-color:rgba(255,0,0,0.5); /*这个代表红色的半透明*/
```

如果想把一个颜色设置为全透明，我们还可以使用一个特殊的枚举值 `transparent`

背景

1. `background-color` 背景颜色
2. `background-image` 背景图片
3. `background-repeat` 当背景图片小于盒子的时候，默认会现一个重复平铺行为，这个属性就是用于控制平铺行为了

- `repeat-x` 只在水平方向重复平铺
 - `repeat-y` 只在垂直方向上面重复平铺
 - `repeat` 在水平方向与 垂直方向 上面同时重复平铺
 - `no-repeat` 不重复平铺
4. `background-position` 用于设置背景图的位置，它可以同时设置水平方向和垂直方向的位置

```
background-position:水平 垂直
```

如果只写第一个值，第二值是 `center`。同时在这里要注意，上中下的中在这里是 `center` 而不是 `middle`

5. `background-size` 用于设置背景图片的大小，它后面可以接1~2个值

```
background-size:100px 200px; /*第一个值代表宽度，第二个值代表高度*/
background-size: auto 100px; /*高度设置为100，宽度则自动缩放*/
```

```
background-size:100px; /*如果只有一个值，则代表宽度，高度就会变成auto，根据图片自动缩放*/
background-size:100px auto
```

- `contain` 这个属性值是让背景图片完全显示在盒子里面，这样可能会有一个方向空出来
 - `cover` 这个属性值是让背景图片完全覆盖住盒子，这样可能会有一个方向的图片被裁剪掉显示不出来
6. `background` 它是一个综合属性，后面既可以接颜色，也可以接图片，同时还可以设置上面的几个属性

```
background: red;
background: pink url("img/01.jpg") no-repeat center center / 100px
200px;
```

 特殊应用点：其实背景图片是可以设置多个的

```
/* 图片排在越后面，越在底下 */
background-image: url("img/01.jpg"), url("img/02.png");
background-repeat: no-repeat,no-repeat;
background-size: 100px 200px,500px auto;
background-position: center center,left top;
```

字体与文字

1. `font-family` 设置字体样式

```
font-family: "华文行楷";
```

`font-family` 只能设置电脑上面存在的字体，如果电脑上面没有字体，则会恢复网页的默认字体，针对这种情况，我们一般情况在设置字体的时候可以设置多个字体

```
font-family: "aaa", "bbbb", "microsoft yahei";
```

当第一个字体不生效的时候，它会依次追加后面的字体，直到最后一个

2. `font-size` 设置字体的大小

```
font-size:16px;  
font-size:12pt;
```

上面两个字体的大小是一样的，`px` 代表像素，`pt` 代表字号，同时我们也得出网页的默认字体大小就是 `16px`，同时以谷歌为核心的浏览器的网页的能够看得见的最小字体是 `12px` 【IE最小可以设置到1px】

当面的 `px` 与 `pt` 只是我们在PC端常用的2个单们，还有其它的几个单位

- `px` 像素
- `pt` 字号
- `em` 一个父级元素【也是自己元素】的字体大小，`em`的全称是 `element` 元素
- `rem` 一个根元素字体的大小，它是一个响应式字体单位，它在 `html` 标签的 `font-size` 大小为标准
- `vw/vh` 响应式字体单位，全称叫 `viewport width` 和 `viewport height`

3. `font-weight` 用于设置字体的权重

- `normal` 字体体质正常
- `bold` 字体加粗
- `bolder` 字体还粗一点
- `lighter` 字体变细

正常下完整的字体应该是有9个等级，分别是 `100~900`，如果是完整字体，我们可以设置数字来表示字体的权重，`normal` 对应的是400

```
font-weight:400; /*正常字体*/  
font-weight:100; /*字体变细*/  
font-weight:700; /*字体加粗*/
```

4. `font-style` 字体倾斜的样式

- `italic` 字体倾斜
- `oblique` 文字倾斜

上面的两个方法都可以让页面上面的文字产生倾斜的效果，但是原理是不一样的，`italic` 先到系统的字体库里面去看，看有没有倾斜，如果有倾斜，则使用字体的斜体；如果系统的字体库里面没有这个字体的斜体，它会直接让文字倾斜（也就是退而求其次，使用 `oblique` 来倾斜）

5. `@font-face` 自定义字体

```
/* 自定义字体 */
@font-face {
    /* 定义字体的名称 */
    font-family: "bgg";
    /* 说明字体在什么地方 */
    src: url("fonts/HKSN.ttf");
}
.aaa {
    font-family: "bgg";
}
```

所以你真的没有必要面对一个你根本赢不了敌人

6. `text-align` 文字水平排列，它有 `left/center/right` 的值

在以前的时候我们如果要让文字两端对齐很麻烦，是通过 `text-align:justify` 来完成，但是这个效果实现起来很困难，所以后期推出了一个的属性叫 `text-align-last:justify`

7. `text-decoration` 文字的描述信息，它是一个简写的属性

- `text-decoration-line` 设置线条的位置
 - `underline` 下划线
 - `overline` 上划线
 - `line-through` 中划线（删除线）
 - `none` 不要设置任何线条
- `text-decoration-color` 设置的线条的颜色
- `text-decoration-style` 设置线条的类型
 - `solid` 实现

- `dashed` 虚线
- `dotted` 点线
- `double` 双线
- `wavy` 波浪线

三个属性合起来就变成了上面的 `text-decoration` 一个属性

8. `text-transform` 设置英文字母的大小写转换

- `uppercase` 大写字母
- `lowercase` 小写字母
- `capitalize` 单词的首字母

9. `text-shadow` 设置文字的阴影，它有四个属性值

```
text-shadow:水平偏移 垂直偏移 阴影模糊 阴影颜色;  
text-shadow: 20px 30px 3px blue;
```

在网页的坐标系里面，执行的是“左负右正，上负下正”的操作

10. `text-indent` 首行缩进

```
text-indent:2em;
```

11. `letter-spacing` 文字的间距，如果是中文没有问题，如果是英文等语言，它会把字母隔开

12. `word-spacing` 单词与单词之间的间距

13. `line-height` 文字的行高，行高指的就是一行文字的高度，行高越大，文字每一行的间距就会越大

```
line-height:32px;  
line-height:1.7;
```

👉 如果文字只有一行，我们想让这个文字垂直居中，则可以将这个行高设置成与高度相同

14. 单行文字溢出省略

```
/* 让文字不换行 */
white-space: nowrap;
/* 溢出的部分隐藏 */
overflow: hidden;
/* 如果是文字溢出了，则添加省略号 */
text-overflow: ellipsis;
```

直到多年以后,我才发现,那些我们曾经...

15. 多行文字溢出省略、

```
display: -webkit-box;
/* 盒子里面的内容垂直排列 */
-webkit-box-orient: vertical;
/* 在第2行的时候省略掉 */
-webkit-line-clamp: 2;
/* 溢出的部分隐藏 */
overflow: hidden;
```

直到多年以后,我才发现,那些我们曾经说过再见的人,是否真的就再见了,那些回...

边框

普通边框

1. **border** 用于设置边框
2. **border-width** 用于设置线条的宽度
3. **border-style** 用于设置线条的类型
 - **solid** 实线
 - **dashed** 虚线
 - **dotted** 点线
 - **double** 双线
 - **groove** 线槽
4. **border-color** 线条的颜色，默认值为 **currentcolor** 当前的文本颜色

5. **border-***,这个星号是方法，指的是 `left/right/top/bottom` 四个方法，可以通过它们分别来设置不同的方向
6. 我们还可以将3个值与4个方法结合起来

初始值	<p>as each of the properties of the shorthand:</p> <ul style="list-style-type: none">• <code>border-width</code>: as each of the properties of the shorthand:<ul style="list-style-type: none">◦ <code>border-top-width: medium</code>◦ <code>border-right-width: medium</code>◦ <code>border-bottom-width: medium</code>◦ <code>border-left-width: medium</code>• <code>border-style</code>: as each of the properties of the shorthand:<ul style="list-style-type: none">◦ <code>border-top-style: none</code>◦ <code>border-right-style: none</code>◦ <code>border-bottom-style: none</code>◦ <code>border-left-style: none</code>• <code>border-color</code>: as each of the properties of the shorthand:<ul style="list-style-type: none">◦ <code>border-top-color: currentcolor</code>◦ <code>border-right-color: currentcolor</code>◦ <code>border-bottom-color: currentcolor</code>◦ <code>border-left-color: currentcolor</code>
-----	--

小技巧：如果想在页面上面快速构建一个三角形，使用下面的方法

```
.box{  
    width: 0px;  
    height: 0px;  
    /* background-color: pink; */  
    border: 30px solid transparent;  
    border-right-color: red;  
}
```

圆角边框

1. **border-radius** 圆角半径，它接收1~8个值

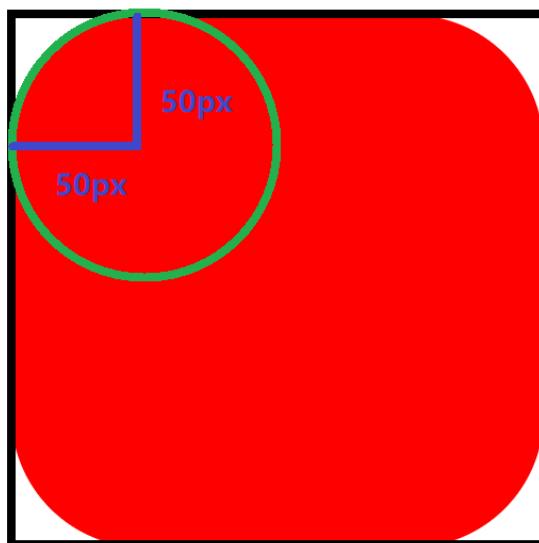
如果是1个值代表四个角相同

如果是2个值第1个值代表左上，第2个代表右上，剩下的参照对角

如果是3个值第1个代表左上，第2个代表右上，第3个代表右下，第4个代表左下

如果是4个值，则顺时针四个角

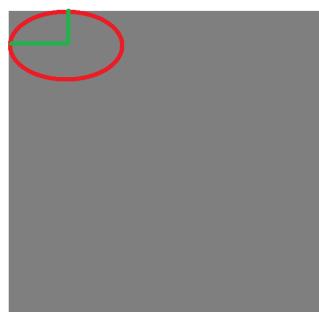
```
border-radius:50px;  
border-radius:50px 30px 100px 10px;
```



这个 **50px** 指的是圆的半径，根据这个半径来设置圆角

2. **border-top-left-radius** 左上角
3. **border-top-right-radius** 右上角
4. **border-bottom-right-radius** 右下角
5. **border-bottom-left-radius** 左下角

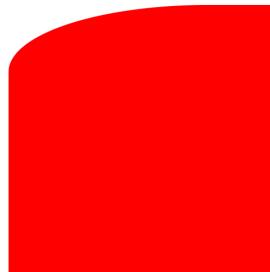
注意：当圆角半径的横轴半径与纵轴半径不相同的时候，它就会是一人椭圆



```
border-radius: 50px 0px 0px 0px ; /*简写*/  
border-radius: 50px 0px 0px 0px / 50px 0px 0px 0px; /*完整写法*/
```

前面的4个值代表横轴半径，后面的3个值代表纵轴半径

```
border-radius: 150px 0px 0px 0px / 50px 0px 0px 0px;
```



列表样式

在之前的时候，我们已讲过了 `ul`/`ol` 标签，它们是有序列表与无序列表，但是今天在讲完了 CSS 的列表样式以后，它们 2 个就没有任何区别了

1. `list-style-type` 列表样式，它的常用的属性值有以下几个

- `none` 取消前面的符号
- `disc` 实心圆
- `circle` 空心圆
- `square` 实心矩形
- `upper-alpha` 大写英文字母
- `lower-alpha` 小写英文字母
- `upper-roman` 大写罗马文
- `lower-roman` 小写的罗马文
- `decimal` 数值
- `decimal-leading-zero` 数值，但是以 0 开始，如 `01`
- `cjk-heavenly-stem` 采用天干记数，如 甲、乙、丙、丁、戊、己、庚、辛、壬、癸
- `cjk-earthly-branch` 采用地支纪年，如 子、丑、寅、卯、辰、巳、午、未、申、酉、戌、亥

2. `list-style-position` 设置列表的符号是在 `li` 元素的内部还是外部

- `outside` 在 `li` 这个元素的外部【默认值】
- `inside` 在 `li` 元素的内部

3. `list-style-image` 可以将列表项前面的符号换成我们的图片

4. `list-style` 它是一个综合属性，它是三个属性的结合

表格

1. 表格添加边框

```
.t1, .t1 tr>td, .t1 tr>th{  
    border: 1px solid black;  
}
```

表格如果要添加边框，要对表格内部的元素格及本身都添加边框

2. 单元格之间的关系

```
border-collapse: separate;      /*使单元格保持分离状态*/  
border-collapse: collapse;     /*全单元格保持合并状态*/
```

3. 单元格与单元格之间的间距

```
border-spacing: 10px;      /*单元格与单元格之间距离10px*/
```

valign 变成了 vertical，原来的 cellspacing 变成了 border-spacing

透明度设置

在这里千万不要弄混了，之前我们学过一个透明色 rgba，今天学的是透明度是要整个元素都设置为透明

The screenshot shows a browser window with two red rectangular boxes containing the text '张三' and '李四'. The top box is labeled '背景半透明' (Background semi-transparent) and the bottom box is labeled '整个元素半透明' (Element semi-transparent). To the left of the browser, the browser's developer tools sidebar displays the following CSS code:

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">  
<title>透明</title>  
<style>  
    .box1 {  
        width: 100px; height: 100px; border: 2px solid black; font-size: 32px; border-collapse: separate; border-spacing: 10px; background-color: #rgba(255, 0, 0, 0.5);  
    }  
    .box2 {  
        width: 100px; height: 100px; border: 2px solid black; font-size: 32px; border-collapse: collapse; border-spacing: 10px; background-color: #rgb(255, 0, 0); opacity: 0.5;  
    }  
</style>
```

透明度的设置我们使用的是 `opacity` 来完成，它的值在0~1之间，1代表全不透明，0代表全透明

```
/*W3C标准写法*/
opacity: 0.5;
/* 滤镜 IE的兼容性写法，在IE8及以下使用 */
filter: alpha(opacity=50);
```

隐藏与显示

1. 通过 `display` 的方式来显示与隐藏

```
display:none; /*隐藏元素*/
display:block; /*显示元素*/
```

2. 通过 `visibility` 的方式来隐藏与显示

```
visibility: visible; /*显示元素*/
visibility: hidden; /*隐藏元素*/
```

区别

1. `display:none` 隐藏元素以后，元素就不再占用位置了，`visibility:hidden` 隐藏元素以后，元素仍然占用之的位置
2. `display:none` 具备株连性，外边的元素隐藏以后，内部的元素就不可能再显示。而 `visibility:hidden` 隐藏以后，内部的元素仍然可以通过 `visibility:visible` 来显示

溢出处理

当元素内部的大小大于外部的大小的时候这个时候就会产生一个溢出的情况



在上面的图片里面我们可以看到，小盒子里面放了一个大图片，但是这个大图片放不小，就溢出了，现在需要处理

溢出处理使用 `overflow` 来完成，它有如下的几个属性值

1. `visible` 溢出以后默认显示出来
2. `hidden` 溢出隐藏
3. `scroll` 滚动条处理
4. `auto` 自动处理，如果溢出就添加 `scroll`，如果不溢出则正常显示

在处理溢出的时候，我们还可以分开处理

```
overflow-x: hidden;  
overflow-y: scroll;
```

鼠标光标

鼠标的光标通过 `cursor` 来进行设置，它有如下的几个属性值

1. `default` 默认值
2. `pointer` 光标变成一个手的形状
3. `wait` 光标变成等待状态（光标变成一个圈在那里旋转）
4. `help` 会变成一个带问号的光标
5. `move` 鼠标会变成一个移动的光标状态
6. `text` 把光标变成输入框的状态
7. `*-resize` 光标变成可调整大小的状态
8. 自定义光标 `cursor: url("img/02.png"), default`

计数器

我们先抛一个问题

```
<fieldset>  
  <p>第一项</p>  
  <p>第二项</p>  
  <p>第三项</p>  
  <p>第四项</p>  
  <p>第五项</p>  

```

在上面的代码 时面，我们希望实现下面的几个点

1. 希望p标签像ul标签里面的列表项一样，在前面加上1,2,3,4,5这样的符号
2. fieldset下面的p有序号，但是序号是从10开始，10,11,12
3. fieldset下在面p序号是只有偶数如2,4,6,8,10

针对上面的特殊需求，我们的 `ul` 标签是无法实现的，这个时候就要使用高级用法计数器
在使用计数器之前一定要先弄清楚3个点

1. 你要对谁计数
2. 你要从什么数开始计数
3. 统计的数增量是多少

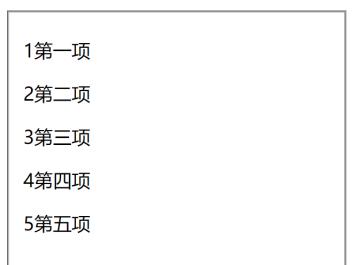
请看下面的使用方法

1. 你要对谁计数，那么就在这个元素的外层添加 `counter-reset` 属性
2. 你要对谁计数，那么就在这个元素上面添加 `counter-increment` 属性
3. 你要把统计的结果放在什么地方，那么就使用伪元素 `::before/:after` 来追加 `counter()` 属性

```
/* 你要对谁计数，那么就在这个元素的外层添加`counter-reset`属性 */
fieldset {
    /* 重置一个计数器，也可以认为是定义一个计数器*/
    counter-reset: aaa 0;
}

/* 你要对谁计数，那么就在这个元素上面添加`counter-increment`属性 */
fieldset>p {
    counter-increment:aaa 1 ;
}

/* 你要把统计的结果放在什么地方，那么就使用伪元素`::before/:after`来追加`counter()`属性 */
fieldset>p::before{
    content: counter(aaa);
}
```



根据上面的案例，我们可以得到3个属性操作

1. `counter-reset:计数器名子 初始值` 初始化一个计数器，并初始化这个计数器值
2. `counter-increment:计数器名子 增量` 在统计数量的时候，使用哪个计数器，并设置增量为多少

3. **counter(计数器)** 从计数器里面拿出当前计数的值

盒子阴影

盒子阴影使用 **box-shadow** 它接收6个值（无数组）

```
box-shadow: 水平偏移 垂直偏移 阴影模糊度 阴影扩散度 阴影颜色 是否内阴影;  
box-shadow: 0px 0px 10px 10px red inset;
```

阴影还可以设置多组

```
box-shadow: 60px 0px 10px red,  
          0px 60px 20px black,  
          -60px 0px 5px blue;
```

滤镜

滤镜是CSS中的一种特殊展示效果，它使用 **filter** 做为属性名

1. **alpha(opacity=50)** 设置透明度
2. **blur(30px)** 高斯模糊
3. **hue-rotate(90deg)** 色相饱和度反转
4. **grayscale(0.5)** 灰度扩展

盒子模型及元素类型

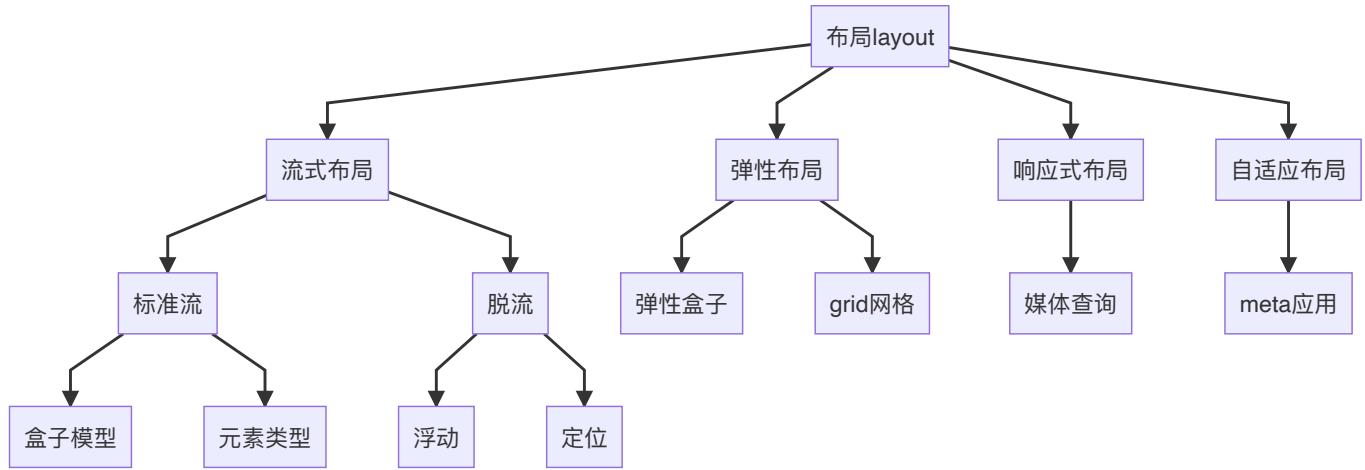
关于布局的概念

之前就已给大家提过了，在CSS的属性里面，我们把CSS的属性分为了2大类的属性，分别是样式属性和布局属性。样式是进行一些细节的微调，而布局是整体的排列，从今天开始我们就要慢慢的接触到布局（layout）

在现行的布局标准里面，我们常把布局分为四大类，想完成布局的学习就必须先完成布局的属性和规范学习

1. 流式布局主要针对的是PC网页，它兼容强，缺点就是比较麻烦，有很多坑
2. 弹性布局主要针对的是移动端，它方便快捷，操作简单，依赖于弹性盒子与网格完成
3. 响应式布局是一种动态布局，页面的布局会根据不同的设备自动做出样式响应式，它界于PC，平板，移动端等多终端
4. 一个网页可以在不同设备，不同尺寸大小下面自动调整，自动适应，这种情况，我们叫

自适应布局。自适应布局是需要借用于JS完成

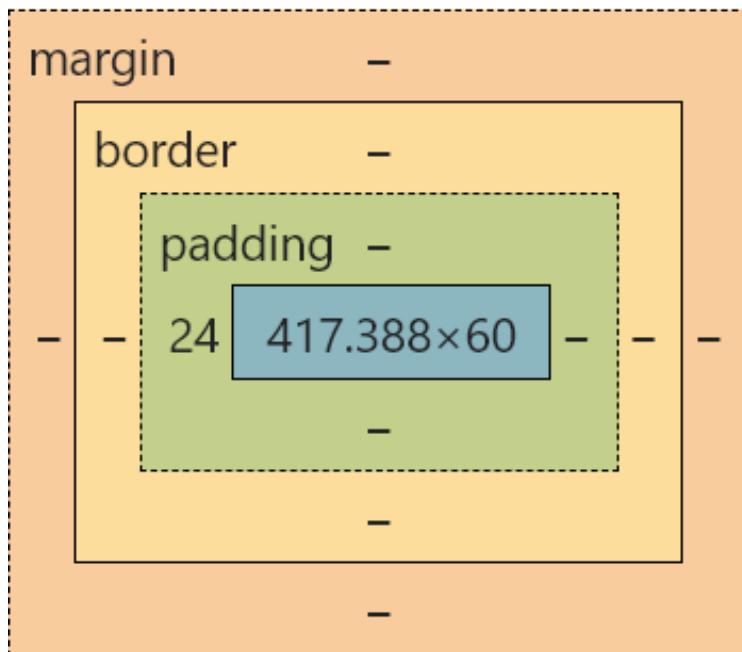


现在我们要从基础的流式布局开始，并且要从标准流开始，就应该学习它的技术

盒子模型

什么是盒子模型？

盒子模型是流式布局当中一套标准技术，它包含四层，如下图所示



盒子模型由外向内分别是4层

1. **margin** 外间距
2. **border** 边框
3. **padding** 内间距
4. **content** 内容

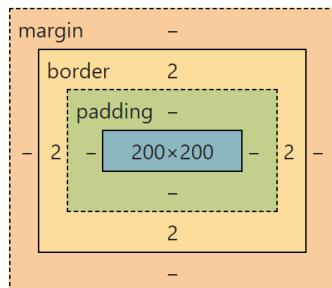
我们在布局的时候，可以将页面上面的每一个元素都认为是一个盒子，然后它们通过一定的规则将这个盒子排列起来就构成了我们的网页

margin外间距

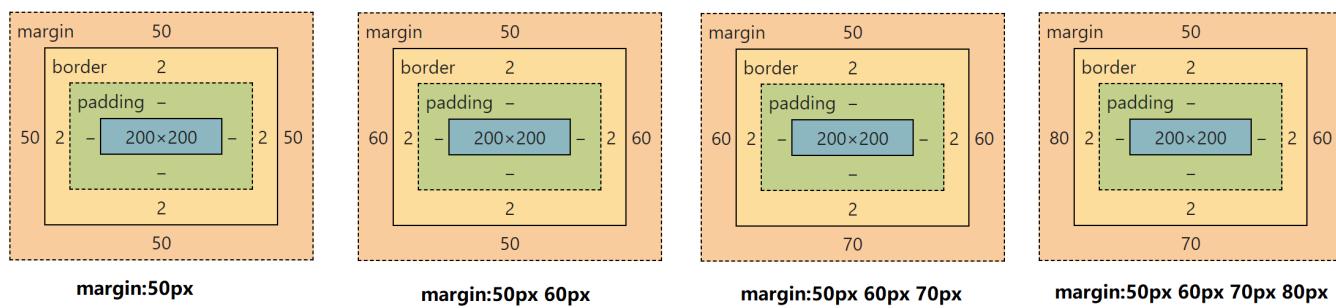
margin指的是2个元素之间的外间距，它接收1~4个值



```
.box {
  width: 200px;
  height: 200px;
  border: 2px solid black;
}
```



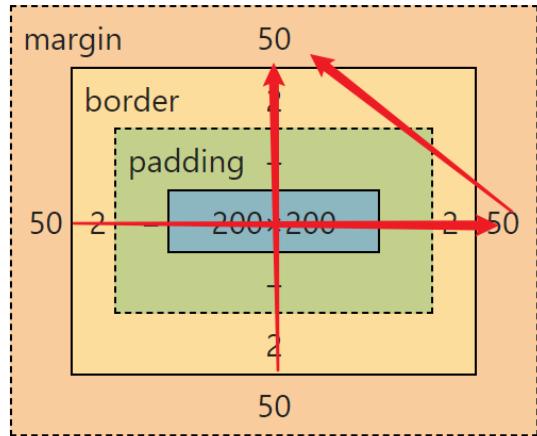
现在我们分别去设置 **margin** 的值，然后看一下它的盒子模型的表现形式



- **margin** 的值设置是从上边开始的

- `margin` 只设置1个值代表四个方向都相同
- `margin` 设置2个值，第1个代表上边，第2个代表右边，剩下的参照对边
- `margin` 设置3个值，第1个代表上边，第2个代表右边，第3个代表下边，剩下的参照对边
- `margin` 设置4个值，从上边开始，依次顺时针一圈

`margin` 的设置总体上来请参考下面这一张图就可以了，`margin`有值就赋值，没值的就参照下面的图来进行



上边的 `margin` 属性是一个简写属性，本质上它是四个方向结合的，所以它可以拆开分开设置

- `margin-left` 设置左边的外间距
- `margin-right` 设置右边的外间距
- `margin-top` 设置上边的外间距
- `margin-bottom` 设置下边的外间距

关于auto的情况

第一步：当我们在一个盒子上面设置 `margin-left:auto` 的时候，这个盒子会去最右边

```
margin-left:auto; /*盒子去了最右边*/
```

第二步：我在在这个盒子上面再去添加 `margin-right:auto`，盒子就到正中间

```
margin-left:auto;
margin-right:auto;
```

第三步：将上面的属性简写

```
margin: 0px auto;
```

第四步：因为网页本身是没有高度，所以上下边距这个0可以设置成 `auto`

```
margin:auto auto;
```

第五步：再次简化代码

```
margin:auto;
```

结论：经过上面在版5个步骤的推断以后，我们得到了一个结果，如果想让一个块级元素左右居中，直接使用 `margin:auto` 就可以了

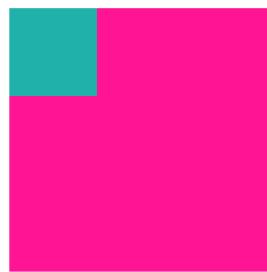
margin的穿透与折叠

margin的穿透现象

请看现象

```
<style>
    .box{
        width: 300px;
        height: 300px;
        background-color: deeppink;
    }
    .small-box{
        width: 100px;
        height: 100px;
        background-color: lightseagreen;
        margin-top: 150px;
    }
</style>
<div class="box">
    <div class="small-box"></div>
</div>
```

代码本意是在一个大盒子里面放一个小盒子，然后在小盒子的上面设置 `margin-top`，这个时候却发生意外



这时候我们可以看到，原本属于小盒子的 `margin` 穿过了外层的大盒子，体现在了大盒子的上面，这种现象就叫 `margin` 的穿透现象

解决方法

- 在外层的大盒子上面添加一个 `border-top` 去解决，原因就是因为外边的盒子没有上边框，`margin` 穿透出去了
- 使用CSS Hack去解决，使用 `BFC` 去解决 【BFC其实是一指一些特殊的属性】，如 `overflow` 可以解决

margin的折叠现象

请看下边的代码

```
<style>
    .box1{
        width: 200px;
        height: 100px;
        background-color: deeppink;
        margin-bottom: 20px;
    }
    .box2{
        width: 200px;
        height: 100px;
        background-color: lightseagreen;
        margin-top: 50px;
    }
</style>
<div class="box1">第一个盒子</div>
<div class="box2">第二个盒子</div>
```



我们在上边的盒子上面添加了下间距 `20px`，同时在下边的盒子上面添加了上间距 `50px`，按照我们的理解，应该是`70px`,结果只有`50px`,这种现象的差异就是margin的折叠现象

margin的折叠指的是上下两个元素的上下间距在同时设置的时候，以大的一个为主
解决方法

1. 在上面两个元素中间给一个空元素，添加 BFC 的属性

```
<div class="box1">第一个盒子</div>
<div style="overflow: auto;"></div>
<div class="box2">第二个盒子</div>
```

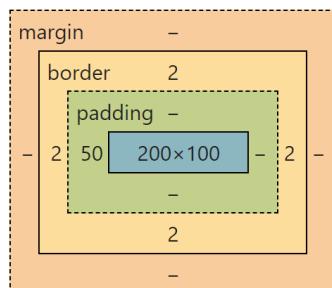
2. 通过BFC去解决【TODO到后面的BFC章节再来讲】

无论是margin的穿透现象还是折叠现象，只存在于上下两个方法，左右不存在

padding内间距

内间距指的是盒子到内容的距离，它使用 `padding` 来表示，与 `margin` 的设置方法相同，也接收1~4值，同时也具备4个方向

1. `padding-left`
2. `padding-right`
3. `padding-top`
4. `padding-bottom`



当我们在设置盒子的内间距的时候，我们发现一个特点，盒子会被撑大

```
.box{  
    width:200px;  
    height:100px;  
    border:2px solid black;  
    padding-left:50px;  
}
```

在上面的代码里面，盒子的宽度被撑大了 **50px**

？问题：如何在设置完padding以后保证盒子模型的大小不变呢？

box-sizing属性

box-sizing 这个属性用于表明盒子的 **width/height** 是设置在了 **内容content区域** 还是 **边框 border 区域**， **box-sizing** 它有两个属性值

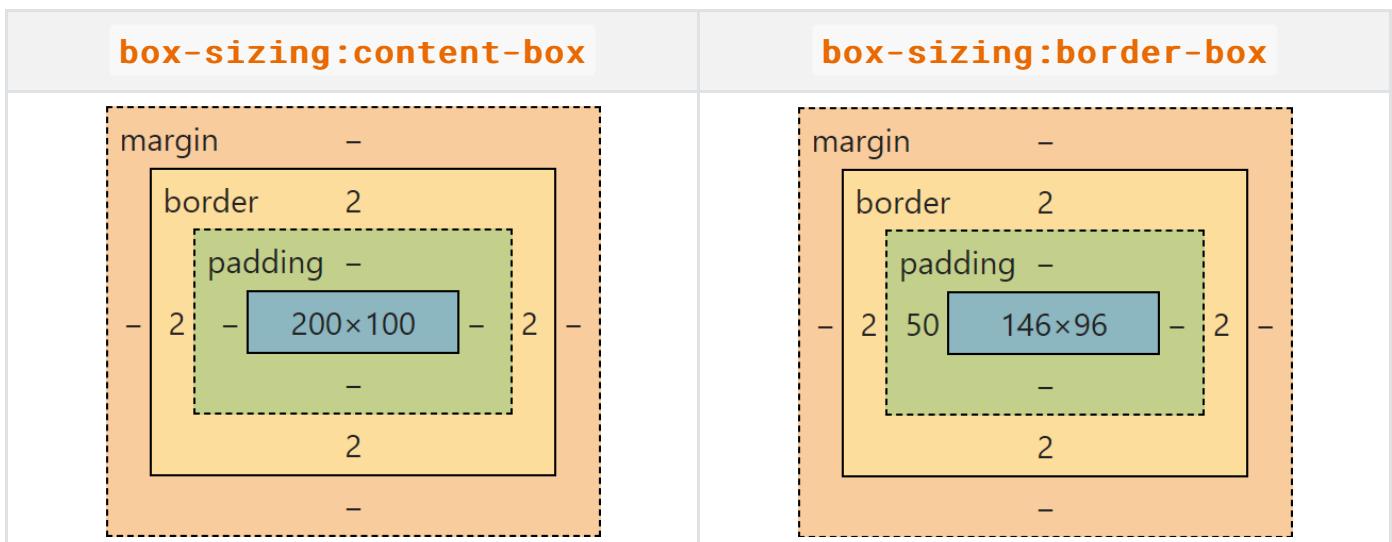
1. **content-box** 代表当前CSS里面的 **width/height** 设置在了 **content 内容区域** 【默认值就是这个】

在这一种情况下，因为我们的 **content** 内容区域的大小固定了，所以当我们再去添加 **padding** 内间距的时候，整个盒子就会被撑大了

2. **border-box** 代表当前的CSS里面 **width/height** 设置在了 **border 边框上面**

在这一种情况下， **border** 的大小就固定了，这个时候的 $width=border+padding+content$

正同的盒子模型图就是box-sizing在不同情况下面的体现



元素类型

首先请看下面的现象

```
<style>
    .div1{
        background-color: deeppink;
        width: 300px;
    }
    .div2{
        background-color: aqua;
    }
    .label1{
        background-color: lightseagreen;
        width: 300px;
    }
</style>


这是一个盒子



这是二个盒子


<label class="label1">这也是一个元素</label>
<label class="label1">这也是一个元素</label>
```

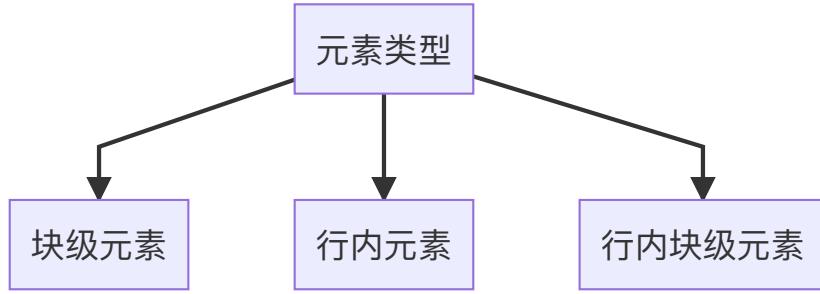
这是一个盒子
这是二个盒子
这也是一个元素 这也是一个元素

在上面的图片里面，它就是效果，通过这个效果，我们可以看到有几点不一样的

1. `div` 元素与 `label` 它们的排列是不一样的，`div` 是独自占用了一行，而 `label` 是可以排列在同一行的
2. `div` 元素可以通过 `width` 来设置宽度，而 `label` 元素对 `width` 的设置好像没有效果
3. `div` 元素默认的宽度好像是 `100%`

要弄清楚上面的三种情况，我们就不得不去了解网页上面的元素类型。

在标准流布局里面，我们把网页上面的元素分为了三种类型



块级元素

块级元素英文名叫 `block`，它默认会具备一个 `display:block` 的属性，页面上面最常见的标签就是这种类型，如 `div, table, ul, li, p, h1~h6` 等这些都是块级元素

块级元素的特征如下

1. 块级元素默认的宽度是父级元素的 `100%`，高度默认是 `0`，可以通过内容来自动的撑开高度
2. 块级元素可以通过 `width/height` 来设置自身的宽度与高度
3. 块级元素独自占用一行
4. 块级元素的水平居中使用 `margin:auto`
5. 块级元素遵守标准的盒子模型

在块级元素里面，有一个非常特殊的标签叫 `div` 标签，它是一个非常纯洁的标签，本身不具备任何样式，专门用于布局

行内元素

行内元素的英文名叫 `inline`，它的表现形式是 `display:inline`，页面上面经常看到的行内元素标签有 `a, label, b, i, u, span` 等，文字也算是行内元素

行内元素的特征如下

1. 行内元素的宽度与高度默认都是 `0`，可以由内容来撑开
2. 行内元素不能通过 `width/height` 来设置自身的宽度与高度
3. 行内元素默认是排在同一行的
4. 行内元素如果在代码里面换行了，页面上面会呈现出一个空格位
5. 行内元素的水平居中是在其外层的块级元素或行内块级上面添加 `text-align:center`
6. 行内元素不遵守标准的盒子模型规范，它只有 `margin-left/margin-right`，没有 `margin-top/margin-bottom`，它的 `padding-left/padding-right` 正常，`padding-top/padding-bottom` 只能把自己撑大，不能撑开外层的元素

在行内元素里面，也有一个非常特殊的标签叫 `span` 标签，它是一个非常纯洁的标签，本身不具备任何样式，专门用于布局

行内块级元素

行内块级元素是既具备块级元素的特点，又具备行内元素的特点，它有表现形式是 `display:inline-block`，所有的表单元素都是行内块级元素

行内块级元素的特征如下

1. 行内块级元素的大小由自身的内容决定
2. 行内块级元素可以通过 `width/height` 来设置自身的宽度与高度
3. 行内块级元素默认排在同一行
4. 行内块级元素如果在代码里面换行了，则会在页面上面产生一个空格位
5. 行内块级元素的水平居中是在其外层的块级元素或行内块级上面添加 `text-align:center`
6. 行内块级元素遵守标准的盒子模型规范
7. 行内块级元素撑开外部盒子的高度的时候，底部默认会多出 `3px`，目前的解决方案是在图片上面添加 `vertical-align:middle`
8. 行内块级元素的垂直居中只用设置行高就可以了，而行内块级元素除了 `line-height`，还要在自身添加一个 `vertical-align:middle`

`img` 元素本身是行内元素，但是表现出来的是行内块级的特征

元素类型的转换

页面上面的三种元素类型相互独立，互不影响，但是它们之间是可以相互转换的

1. 行内元素的表现形式是 `display:inline`
2. 块级元素的表现形式是 `display:block`
3. 行内块级元素的表现形式是 `display:inline-block`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>元素类型转换</title>
    <style>

      .img1{
        /*转换成了块级元素*/
```

```

        display:block;
        margin:auto;
    }
    .bbb{
        border:1px solid black;
        width:100px;
        height:100px;
        /*转换元素类型*/
        display:inline-block;
    }

```

</style>

</head>

<body>

<!-- 本身是行内元素，但是表现的特征是行内块级 -->

<hr />

<div class="bbb">标哥哥</div><div class="bbb">小帅哥</div>

</body>

</html>

在上面的代码里面，我们就将元素的类型进行了转换，以达到我们所需要的效果

行内元素中的4条线



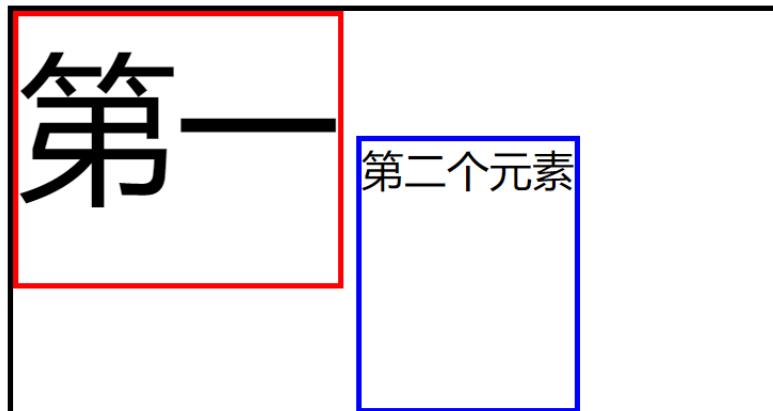
在上面的图里面，我们可以看到，网页不中一行里面有4条线的，看到这4张以后，我们现在再来看下面的代码及效果

```

<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <title>行内元素里面的四条线</title>
    <style>

```

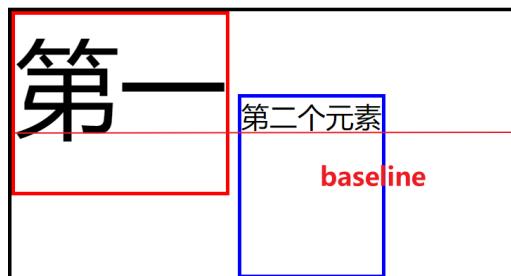
```
.box{  
    border: 2px solid black;  
}  
.s1{  
    height: 100px;  
    display: inline-block;  
    border: 2px solid red;  
    font-size: 60px;  
}  
.s2{  
    height: 100px;  
    display: inline-block;  
    border: 2px solid blue;  
}  
</style>  
</head>  
<body>  
    <div class="box">  
        <span class="s1">第一</span>  
        <span class="s2">第二个元素</span>  
    </div>  
</body>  
</html>
```



为什么2个盒子没有垂直对齐，但是它们的高度又是一样的呢？

The screenshot shows a browser's developer tools with the 'Elements' tab selected. Inside, there's a tree view of HTML elements. Under a div with class 'box', there are two spans: 's1' containing '第一个元素' and 's2' containing '第二个元素'. Below the tree, tabs for '样式' (Style), '已计算' (Computed), '布局' (Layout), '事件侦听器' (Event Listeners), 'DOM 断点' (DOM Breakpoints), '属性' (Properties), and '辅助功能' (Accessibility) are visible. The '已计算' tab is active. In the bottom right of the panel, there's a 'vertical-align' section with 'baseline' highlighted.

在行内元素及行内块级元素里面，垂直对齐的时候，它们默认都是以 **baseline** 来对齐的



它为了两个文字以 **baseline** 基线对齐，所以它不得不将两个盒子上下移动，这样就照成了2个盒子的错位

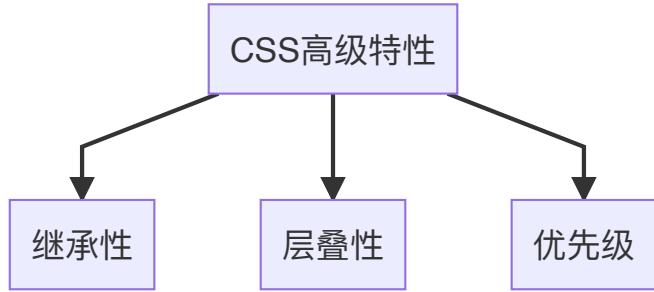
如果要去解决上面的问题，我们就必须更改两个行内块级元素的垂直对齐方式

```
vertical-align:top;
/*或*/
vertical-align:bottom;
```

CSS高级特性

之前的时候我们已经学过了CSS的3个基本点分别是选择器的便捷性，CSS属性的统一性，以及CSS与HTML代码的分离性，这三个特点已经解决了我们平常开发过程当中很多样式问题，但是仍然有一些问题无法实现，这个时候就需要借助于我们CSS的高级特性

CSS的中文名称叫“层叠样式表”



继承性

继承性指的是内部元素默认继承外部元素的样式

怎么理解继承？王思聪和王健林是父子关系，王思聪就可以继承王健林的财产

通俗意义上来说，父级里面有的东西，子级会默认继承下来

```

<style>
  .box{
    border: 2px solid black;
    color: red;
  }
</style>
<div class="box">
  啊啦啦，我是卖报的小行家
  <h2>不等天明去卖报</h2>
</div>

```

在上面的代码里面，我们明明只是把 `box` 设置为了红色，为什么 `h2` 标签也会变成红色呢？

The screenshot shows the browser's developer tools with the 'Elements' tab selected. On the left, the DOM tree displays a `<div class="box">` element containing text and an `<h2>` child element. On the right, the 'Computed' tab of the Styles panel shows the following style definitions for the `.box` class:

```

element.style {
}
.box {
  border: 2px solid black;
  color: red;
}

```

The 'color: red;' rule is highlighted in pink, indicating it is the source of the red color applied to the `h2` element.

上面是 `box` 的样式，我们在浏览器里可以看到，它设置成了红色，现在我们再去看内部的 `h2` 标签

The screenshot shows a browser window with the URL 127.0.0.1:8848/070801/01继承性.html. The page content includes the text "啦啦啦，我是卖报的小行家" and "不等天明去卖报". The browser's developer tools are open, specifically the Elements tab, which displays the DOM tree. A red arrow points from the text "不等天明去卖报" to the corresponding h2 tag in the DOM tree. Another red arrow points from the "color: red;" declaration in the .box style block to the "color: red;" declaration in the h2 style block. The right panel shows the CSS styles for the h2 and .box elements, with a callout box labeled "继承自 div.box" pointing to the .box style block. A red box highlights the "color: red;" declaration in the .box style block, and another red box highlights the "color: red;" declaration in the h2 style block. The text "它继承了父级的样式" is written in red next to the highlighted declaration.

通过上面的图片我们可以得到结果，因为 `h2` 标签继承了父级元素 `box` 的，所以它会有这个红色的文字样式，这个特点就是CSS里面的继承性

在上面图片里面，我们也可以看到一个特点， `color` 这个属性是被继承下来了，而 `border` 没有被继承，这是因为能够默认被继承是样式属性，而布局属性是不能被继承的

其它我们还可以通过一个特殊的CSS属性值来主动的继承

```
h2{  
    /* 我想让我的边框与父级元素的边框一模一样 */  
    border: inherit;  
}
```

层叠性

层叠性的本意是指CSS的样式应该是由多个部分共同组成的

```

<style>
    .box{
        line-height: 50px;
    }
    .a1{
        display: inline-block;
        border: 2px solid black;
    }
</style>
<div class="box">
    <a href="#" style="padding: 0px 10px;" class="a1">百度一下</a>
</div>

```



我们现在想知道为什么 `a` 标签展示出来的效果是这样的？

样式类别	描述
行内样式	<code>element.style { padding: 0px 10px; }</code>
内部样式块	<code>.a1 { display: inline-block; border: 2px solid black; }</code>
自带默认样式	<code>a:-webkit-any-link { color: -webkit-link; cursor: pointer; text-decoration: underline; }</code>
继承样式	<code>.box { line-height: 50px; }</code>

上面图片我们可以看到，层叠性指的是元素终的表现形式是由 **继承样式 + 自带的默认样式 + 内部样式块 + 行内样式** 来共同决定的

层叠性除了决定样式的最终表现形式外，它还可以让我们看到如果样式冲突了，它是怎么解决了，请看下面的效果图

The screenshot shows the Chrome DevTools Elements tab for a page at 127.0.0.1:8848/070801/04层叠性.html. The left pane displays the DOM tree with a selected element `a`. The right pane shows the style inspector with the '样式' (Style) tab selected. It lists several rules:

- `element.style { padding: 0px 10px; }`
- `.a1 { display: inline-block; border: 2px solid black; text-decoration: line-through; }` (覆盖 - Overridden)
- `a:-webkit-any-link { color: -webkit-link; cursor: pointer; text-decoration: underline; }` (覆盖 - Overridden)
- `.box { line-height: 50px; color: red; }` (继承自 div.box - Inherited from div.box)

A red arrow points from the text "覆盖" (Overridden) next to the .a1 rule to the text "覆盖" (Overridden) next to the a:-webkit-any-link rule.

当样式出现冲突的时候，我们就可以看到，谁层叠在了上面，就听谁的

行内样式>内部样式块>默认样式>继承样式

优先级

优先级是为了解决在同一层里面出现样式冲突以后的问题，优先级一般是指选择器的优先级

选择器相同的时候

```
.box{  
    color:red;  
    border:1px solid black;  
    color:yellow;  
}
```

在上面的代码里面，如果选择器相同的时候样式冲突了，我们可以找最后一次出现的样式就可以了，所以 yellow 会覆盖 red

选择器不相同的时候

在我们的开发当中，这种情况是最常见的，我们有很多个选择器，我们可以使用不同的方式来选择同一个元素，这个时候 如果出现了样式冲突，我们应该计算一权重值再来做比较

选择器的权重值

选择器	权重值
星号选择器	0, 0, 0, 0
标签选择器, 伪元素	0, 0, 0, 1
类选择器, 属性选择器, 伪类	0, 0, 1, 0
ID选择器	0, 1, 0, 0
style属性	1, 0, 0, 0
!important	无穷大

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>选择器的权重值计算</title>
    <style>
      /* 1 */
      p{
        color: red !important;
      }
      /* 1+1=2 */
      fieldset>p{
        color: blue;
      }
      /* 10 */
      .p1{
        color: yellow;
      }
      /* 1+10=11 */
      p.p1{
        color: pink;
      }
      /* 1+1+10=12 */
      fieldset>p.p1{
        color: green;
      }
      /* 10+10=20 */
      .f1>.p1{
        color: black;
      }
    </style>
  </head>
  <body>
    <h1>权重值计算</h1>
    <div>
      <h2>普通元素</h2>
      <p>普通文本</p>
      <div>
        <h3>嵌套元素</h3>
        <p>嵌套文本</p>
      </div>
      <div>
        <h3>类选择器</h3>
        <p>类文本</p>
      </div>
      <div>
        <h3>ID选择器</h3>
        <p>ID文本</p>
      </div>
      <div>
        <h3>伪元素</h3>
        <p>伪文本</p>
      </div>
      <div>
        <h3>标签选择器</h3>
        <p>标签文本</p>
      </div>
      <div>
        <h3>属性选择器</h3>
        <p>属性文本</p>
      </div>
      <div>
        <h3>类选择器</h3>
        <p>类文本</p>
      </div>
      <div>
        <h3>组合选择器</h3>
        <p>组合文本</p>
      </div>
      <div>
        <h3>伪类选择器</h3>
        <p>伪类文本</p>
      </div>
      <div>
        <h3>style属性</h3>
        <p>style文本</p>
      </div>
      <div>
        <h3>!important</h3>
        <p>!important文本</p>
      </div>
    </div>
  </body>
</html>

```

```
}

/* 10+1+10=21 */

.f1>p.p1{
    color: red;
}

/* 1+10+1+10=22 */

fieldset.f1>p.p1{
    color: yellow;
}

/* 100 */

#aaa{
    color: blue;
}

/* 1+100=101 */

p#aaa{
    color: red;
}

/* 10+100=110 */

.p1#aaa{
    color: green;
}

</style>

</head>
<body>
    <fieldset class="f1">
        <p class="p1" id="aaa" style="color: blue;">震惊，日本前首相安倍晋三突遭不幸福</p>
    </fieldset>
</body>
</html>
```

上面的代码就是用来展示选择器的权重值的计算过程，但有一个根本原则不能违背

`style>ID>class>标签`这是一个总体原则，官方的说法就是选择器权重不进位

```
}

</style>
<div>
  <div>
    <div>
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div class="div1">我是第11个</div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

在上面的代码里面，我们可以看到最终显示的效果是 blue，因为标签选择器无论怎么叠加，都不能大于 class 选择器，也就是值不能进位

定位

定位是脱流布局的另一种体现形式，在之前我们学习过了浮动，它与浮动脱流还是有一点区别的，目前使用的定位方式有5个，它使用 position 做为属性值

一个元素一旦使用了定位以后，它就会多出如下的5个属性值

1. left 调整左边的位置
2. right 调整的右边的位置
3. top 调整上边的位置
4. bottom 调整下边的位置
5. z-index 调整Z轴的层级，默认情况下，定位以后的元素会高于标准流一个Z轴层级

相对定位

相对定位我们使用属性值 `relative`

```
position: relative;
```

特点

1. 相对定位的元素如果发生位置变化以后，不再用新的位置空间



在上面的图片里面，我们可以看到，盒子设置 `top:50px` 下向偏移以后，下面的h2标签并没有被挤下来，也没有占用文字内容的空间

2. 相对定位的元素如果位置发生变化以后，原来的位置仍然要占用保留



在上面的图片里在，我们可以看到盒子设置 `top:-60px` 以后上向移动，但是仍然占用了原来的位置，所以下在h2就不能顶上去

3. 相对定位的 `left/right/top/bottom` 来设置位置偏移以后，它是相对于原来的位置再进行偏移

在上面的效果里面，我们可以看到，一个元素经过相对定位调整位置以后，它仍然占用原来的位置，所以它没有脱流

绝对定位

绝对定位使用 `absolute`

```
position: absolute;
```

特点

- 当一个元素绝对定位以后，它在这个位置上面立即脱流，不再占用原来的位置



- 绝对定位它如果使用 `left/right/top/bottom` 来进行位置设置的时候，是需要找参照物的。绝对定位的参照物是找外层的定位元素，如果找不到，最终会找到 `body`

```
left: 50px;  
top: 50px;
```

所以我们会看到上图的情况，`div1` 会向父级找定位元素，但是父级没有，就继续向外找，一直找到 `body` 标签，如果还没有找到，则直接以 `body` 为参照物来定位

如果我们现在在外层的元素上面添加 `position: relative` 以后，再来看

发现了一个定位
则以这个元素为参照物来进行位置设置

绝对定位，向外层找参照物

发现了一个定位
则以这个元素为参照物来进行位置设置

所以，这里的left:50px;top:50px;
则参照的是外层的box这个盒子，而不再是body标签了

05绝对定位.

固定定位

固定定位使用 **fixed** 为属性值

```
position:fixed;
```

固定定位顾名思义就是固定在浏览器的某一个地方，它不会随着页面的滚动而滚动

特点：

1. 固定定位一流以 **浏览器** 为标准进行定位
2. 固定定位是脱流的

绝对定位与固定定位的相同点：都脱流了

绝对定位与固定定位的不同点：

1. 绝对定位是要向外层找参照物的，如果找不到了就以 **body** 为参照物，而固定定位是一流以浏览器为参照物
2. 正是因为绝对定位如果以 **body** 参照物，所以网页的内容在滚动的时候，绝对定位的元素也会滚动，而固定定位是以浏览器为标准，固定在了浏览器的某一个地方，它不会随着页面的滚动页滚动

静态定位

静态定位就是取消定位，清除定位，它使用 `static`

```
position:static
```

当一个元素使用了 `static` 定位以后，则所有的定位特性都会消失，`left/right/top/bottom/z-index` 5个属性全部失效

粘性定位

1. 当它的位置让它可以正常呈现的时候，它的定位等同于 `position: static`，随着正常的文档流滚动
2. 当它的位置不足以让它正常显示，但它的父元素有足够的空间让它显示，它的定位等同于 `position: fixed`
3. 当它的父元素的空间不够让它显示，它的定位等同于 `position:static`，它又可以随着文档流滚动了

原话：当它的父元素的空间不足以让它显示，它的定位等同于 `position:absolute`

粘性定位没有脱流

子绝父相

子绝父相的本意是指当一个元素使用了绝对定位以后，外层应该使用相对定位来约束这个绝对定位【相当于给一个参物】，如果没有这个参照物约束，那么这个元素就会一直找到 `body`

子级元素使用绝对定位，外层元素使用相对定位，这一种特殊的关系我们叫子绝父相

应用场景：当一个元素要以另一个元素为标准进行位置设置的时候，最好的办法就是子绝父相

知识扩展：子绝父相它只是一个最基本的概念，外层元素不一定非要是相对定位 `relative`，可以是除了 `static` 静态定位以外的任何定位都可以

注意：在CSS3里面 `transform` 也可以用来约束绝对定位

定位脱流的影响

1. 定位脱流是在原来的位置直接脱流，并高于默认的文档流一个层级，会盖住标准流，但是可以通过 `z-index` 调整层级，可以通过 `left/right/top/bottom` 来调整位置
2. 定位脱流不占用原来的位置
3. 定位脱流无视元素类型，所有元素都会变成 `display:block`
4. 定位脱流以后的元素会丢失宽度，默认由内容来撑开，但是可以通过 `width/height` 来调整宽高，**它还可以通过`left/right`来拉开宽度，也可以通过`top/bottom`拉开高度**
5. 定位脱流以后的元素会形成BFC（没有 `margin` 的穿透现象）
6. 定位脱流可以使用特殊的技术来居中【重点：引伸出了流的概念】

```
.box{  
    position: absolute;  
    left: 0;  
    right: 0;  
    top: 0;  
    bottom: 0;  
    /*上面的4个属性就是拉开了流*/  
    margin: auto;  
}
```

浮动脱流与定位脱流的区别

相同点

1. 两种脱流都不占用位置
2. 两种脱流都无视元素类型
3. 两种脱流都会丢失宽度
4. 两种脱流都会有BFC现象

不同点

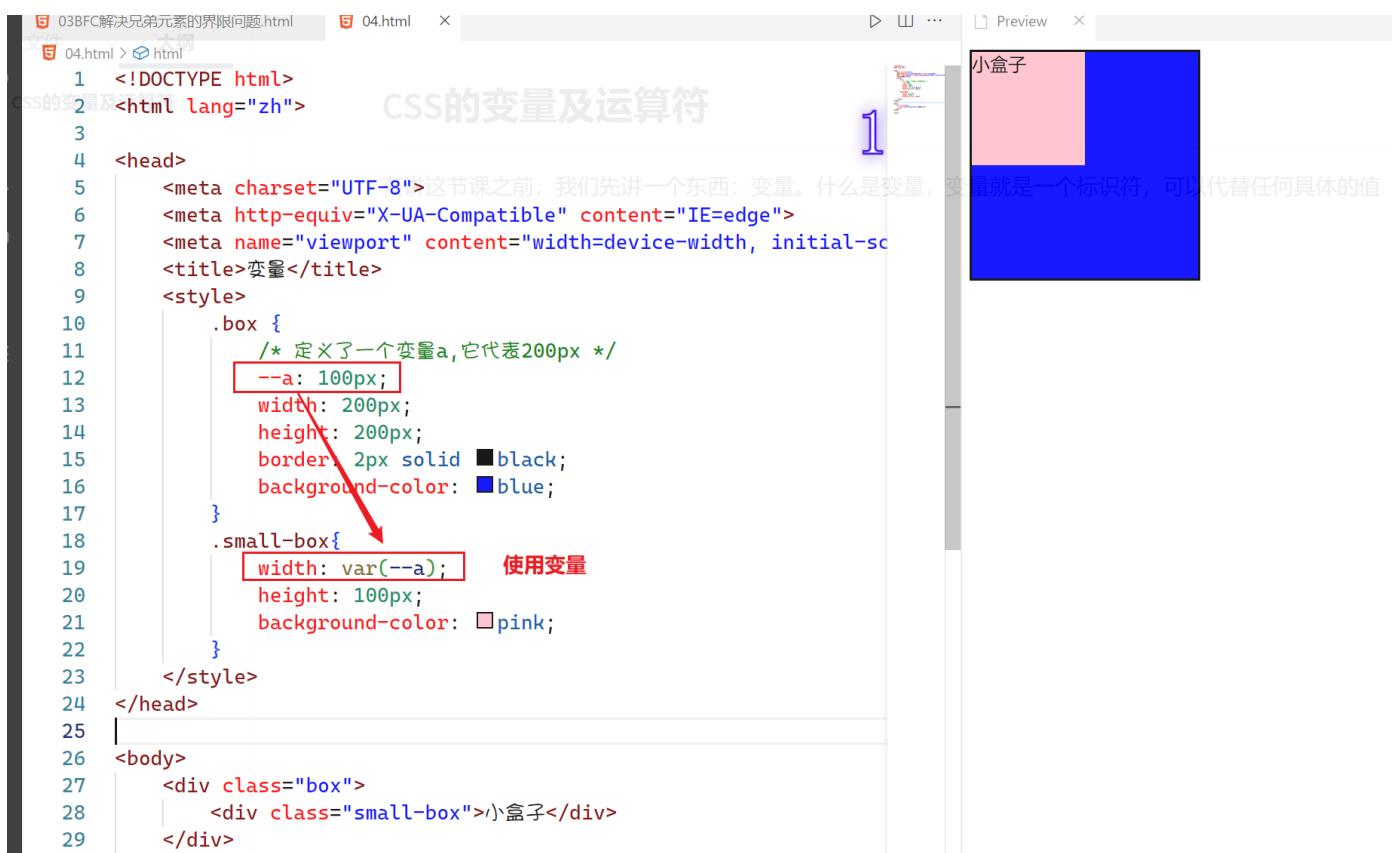
1. 浮动脱流不能改变Z轴的层级，而定位脱流可以通过 `z-index` 来调整Z轴的层级
2. 浮动脱流的元素必须是 `width/height` 来设置宽高，而定位脱流以后的元素还可以通过 `left/right/top/bottom` 来拉开宽高（本质上拉的是流）
3. 浮动脱流以后的元素是无论如何也不能居中的，只能找父级标准流；而定位脱流以后的元素是可以通过拉开流以后再设置 `margin:auto` 居中
4. 浮动脱流不占用布局位置但是占用内容位置，定位脱流既不占用布局位置又不占用内容

位置

CSS的变量及运算符

CSS变量的使用

在讲这节课之前，我们先讲一个东西：变量。什么是变量，变量就是一个标识符，可以代替任何具体的值



```
03BFC解决兄弟元素的界限问题.html 04.html > Preview >
04.html > html
1 <!DOCTYPE html>
2 <html lang="zh">
3
4 <head>
5   <meta charset="UTF-8"> 这节课之前，我们先讲一个东西：变量。什么是变量，变量就是一个标识符，可以代替任何具体的值
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-s
8   <title>变量</title>
9   <style>
10    .box {
11      /* 定义了一个变量a, 它代表200px */
12      --a: 100px;
13      width: 200px;
14      height: 200px;
15      border: 2px solid black;
16      background-color: blue;
17    }
18    .small-box{
19      width: var(--a);    使用变量
20      height: 100px;
21      background-color: pink;
22    }
23  </style>
24 </head>
25
26 <body>
27   <div class="box">
28     <div class="small-box">小盒子</div>
29   </div>
-->
```

在上面的代码里面，它就一个定义变量和使用变量的过程

1. 定义变量

```
--变量名:变量值;
--a:100px;
--b:red;
```

2. 使用变量

```
width:var(--a); /*使用变量值--a*/
background-color:var(--b); /*使用变量值--b*/
```

```
<style>
  .box {
    /* 定义了一个变量a, 它代表200px */
    --a: 100px;
    --b:red;
    --c:200px;
    width: 200px;
    height: var(--c);
    border: 2px solid black;
    background-color: blue;
  }
  .small-box{
    width: var(--a);
    height: 100px;
    background-color: var(--b);
  }
</style>
<body>
  <div class="box">
    <div class="small-box">小盒子</div>
  </div>
</body>
```

在上面的代码里面，我们可以看到，自己定义的变量，自己是可以使用的，同是我的后代元素也是可以使用的，超过个范围别人就不能用了

The screenshot shows a code editor with a file named style.css. The code defines a variable `--a` with a value of 100px. It then uses this variable to define the width of a class `.box` (200px), a class `.small-box` (100px), and a class `.box2` (200px). The `.box` and `.small-box` elements have red and blue backgrounds respectively, while the `.box2` element has a black border. A red arrow points from the text "box2不是box1的后代，所以不能使用box1定义的变量--b" to the line where `background-color: var(--b);` is used in the `.box2` definition.

```
<style>
  .box {
    /* 定义了一个变量a,它代表200px */
    --a: 100px;
    --b: red;
    --c: 200px;
    --d: 100px;
    width: 200px;
    height: var(--c);
    border: 2px solid black;
    background-color: blue;
  }
  .small-box {
    width: var(--a);
    height: 100px;
    background-color: var(--b);
  }
  .box2 {
    width: 200px;
    height: 200px;
    border: 2px solid black;
    background-color: var(--b);
  }
</style>
</head>
<body>
  <div class="box">
    <div class="small-box">盒子</div>
  </div>
  <div class="box2"></div>
</body>
```

思考：如果定义一个全局的变量？

```
html{
  --bgg:yellow;
  --xxx:green;
}
:root{
  --bgg:yellow;
  --xxx:green;
}
```

因为 `html` 是网页的根标签，所以我们可以在里面定义全局变量，同时 `:root` 的伪类就是 `html` 标签，我们也可以通过它来定义

CSS运算符

在CSS里面，我们可以进行简单的运算

```
width:calc(100px + 200px);
width:calc(100px - 50px);
width:calc(100px / 2);
width:calc(100px * 2 - 30px)
```

1. 符号的左右必须有一个空格
2. 它遵守的就是四则混合运算

```
:root{  
    --safeWidth:1400px;  
}  
  
.container {  
    /*使用变量*/  
    width: var(--safeWidth);  
    outline: 5px solid black;  
    margin: auto;  
}  
.right{  
    outline: 2px solid red;  
    float: right;  
    /* width: 686px; */  
    /*运算符，变量结合起来使用*/  
    width: calc(var(--safeWidth) - 714px);  
}
```

BFC的概念

什么是BFC

BFC(block formatting context)：简单来说，BFC就是一种属性，这种属性会影响着元素的位置以及与其兄弟元素之间的相互作用。

中文常译为块级格式化上下文。是 W3C 中 CSS2.1 规范中的一个概念，它决定了元素如何对其内容进行位置设置，以及与其他元素的关系和相互作用。

在进行 **盒子模型** 布局的时候，BFC 提供了一个环境，在这个环境中按照一定规则进行布局不会影响到其它环境中的布局。比如浮动元素会形成 BFC，浮动元素内部子元素的主要受该浮动元素影响，两个浮动元素之间是互不影响的。也就是说，如果一个元素符合了成为 BFC 的条件，该元素内部元素的布局和定位就和外部元素互不影响(除非内部的盒子建立了新的 BFC)，是一个隔离了的独立容器。(在 CSS3 中，BFC 叫做 Flow Root)

1. BFC 是一个封闭的环境，在这个环境中布局不会影响到外边的元素，也不会被外边的元素影响

2. BFC是一个封闭的环境，它不会影响兄弟环境

怎么形成BFC

形成BFC是有很多个CSS属性都可以完成了

1. `overflow` 的属性值除了 `visible` 以外的，如 `auto/scroll/hidden` 都可以了
2. `float:left` 或 `float:right`
3. `position:fixed` 或 `position:absolute`
4. `display` 的值为 `inline-block` , `table-cell` , `table-caption`
5. 上面的四个部分的属性本意上面都不是为了形成BFC的，所以用起来的时候会有点怪怪的感觉

在后面的CSS3里面，就专门推出一个形成BFC的属性值，叫 `display:flow-root`，这个属性什么都不干，专门形成BFC，这个属性没有任何副作用

BFC的作用

1. 解决父级元素包裹不了子级元素的问题

思考：什么情况下，父级元素包裹不了子级元素

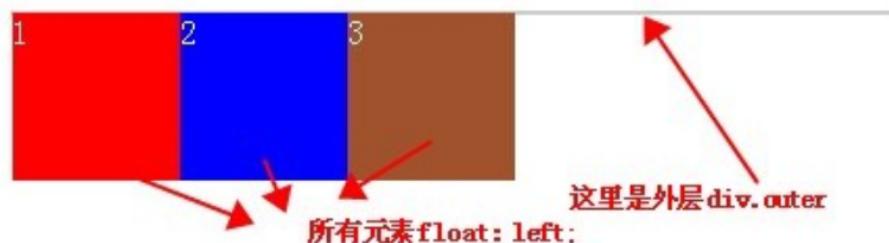
- `margin` 穿透现象，内部的 `margin` 穿透到外部去了，这个时候父级元素就没有包裹住子级元素
- 浮动脱流以后，父级元素包裹不了子级元素，造成高度坍塌

对于上面的 `margin` 的穿透现象及浮动脱流照成的高度坍塌现象，我们都可以使用BFC来解决

2. 兄弟元素划清界限

» 包含浮动元素

问题案例：高度塌陷问题：在通常情况下父元素的高度会被子元素撑开，而在这里因为其子元素为浮动元素所以父元素发生了高度坍塌，上下边界重合。这时就可以用 **bfc** 来清除浮动了。【父元素包裹住子元素】



上面的图的情况就是浮动以后造成的父级元素高度坍塌，这个时候我们可以在父级元素上面添加 **BFC** 属性，来恢复父级元素的元素

» 不被浮动元素覆盖

问题案例：div浮动兄弟遮盖问题：由于左侧块级元素发生了浮动，所以和右侧未发生浮动的块级元素不在同一层内，所以会发生div遮挡问题。可以给灰色块加 **overflow: hidden**，触发 **bfc** 来解决遮挡问题。【兄弟元素之间划清界面】



在上图当中，我们可以看到，当元素浮动以后，兄弟元素之间就没有划清界面，我们可以在浮动元素的后面添加 **BFC** 来解决这个问题

没有添加BFC的时候



添加完BFC以后



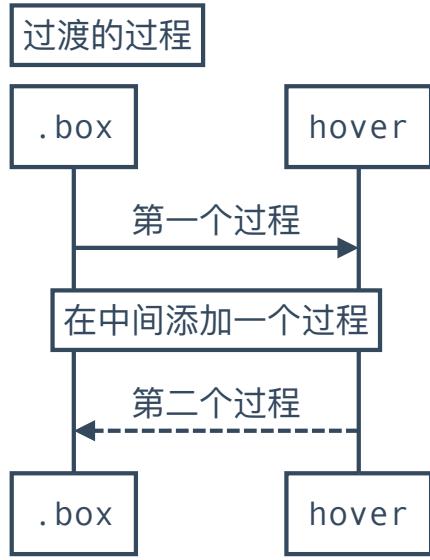
当添加完BFC的属性以后，我们可以看到，两个元素就彻底的划清界面了

CSS3过渡

CSS过渡指的是元素从一个状态到另一个状态的转变过程，过渡使用的是CSS当中的 `transition` 这个属性，这个属性本身很简单，但是它需结合起来使用

```
<!DOCTYPE html>
<html lang="zh">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>过渡</title>
<style>
    .box{
        width: 100px;
        height: 100px;
        background-color: red;
    }
    .box:hover{
        background-color: blue;
    }
</style>
</head>
<body>
    <div class="box"></div>
</body>
</html>
```

在上面的代码里面，我们可以看到，当鼠标放在元素上面以后，元素会由红色变成蓝色，当鼠标松开以后，元素会由蓝色转变成红色，这个转变是一瞬间就完成的，那么，我们如果在这个转里面添加一个过程，这过程就叫过渡



过渡的属性

1. **transition-property** 要执行过渡的属性
2. **transition-duration** 要执行过渡的时间，它可以以 **s** 秒为单位，也可以以 **ms** 毫秒为单位】
3. **transition-timing-function** 过渡的效果，也叫过渡的时间函数【默认值是 **ease** 】
 - **linear** 匀速
 - **ease-in** 先慢后快， **ease** 单词是轻松的意思， **in** 是开始的意思
 - **ease-out** 先快后慢， **out** 代表结束的意思
 - **ease-in-out** 前后都很慢， 中间很快
 - 我们还可以在浏览器里面手动的编辑我们需要的效果，如 **cubic-bezier(0.74, -1.06, 0.35, 2.34)**
4. **transition-delay** 过渡等待【默认值0s】

上面的四个属性是可以合起来一起写的

```

transition-property: width;
transition-duration: 1s;
transition-timing-function: linear;
transition-delay: 3s;

```

下面的4个属性值可以简写成下面的方式

```
transition: width 1s linear 3s;
```

```
transition 后面的四个属性值是可以更改位置的，如 transition: 1s 3s linear  
width ;
```

但是一定要注意，在属性值里面有2个时间，第一个时间代表过渡持续的时间，第二个时间代表等待时间

多个属性的过渡

在上面学习属性的时候，我们已可以设置一个属性的过渡，如果有多个属性值发生变化以后也需要过渡，那怎么办呢？

```
<style>  
    .box {  
        width: 100px;  
        height: 100px;  
        background-color: deeppink;  
        /*执行过渡*/  
        transition-property: width,height;  
        /*transition-duration: 2s,0.5s;*/  
        transition-duration: 2s;  
        transition-timing-function: linear,ease-in;  
        transition-delay: 4s,0s;  
    }  
  
.box:hover {  
    width: 400px;  
    height: 200px;  
}  
</style>  
<div class="box">  
    盒子  
</div>
```

在上面的属性里面，我们可以看到，如果多个属性需要过渡，我们也是可以的

同样多个属性的过渡也是可以简写成一个 `transition`

```
transition: width 2s linear 4s,height 0.5s;
```

全属性的过渡

当所有变化的属性都需要过渡的时候，我们可以使用 `all` 来代替

```
transition-property: all;      /*这里就使用了all来代替所有变化的属性*/
transition-duration: 4s;
transition-timing-function: linear;
transition-delay: 2s;
```

综合成一个属性

```
transition:all 4s linear 2s;
```

不对称的过渡效果

请看下面代码

```
<style>
  .box{
    width: 100px;
    height: 100px;
    background-color: deeppink;
    transition-property: width;
    transition-duration: 2s;
    transition-delay: 0s;
    transition-timing-function: linear;
  }
  .box:hover{
    width: 300px;
  }
</style>
<body>
  <div class="box">
  </div>
</body>
```

我们可以看到，当鼠标进入触发 `hover` 效果或鼠标离开失去 `hover` 效果，它的效果是一样的，但是我们现在希望改变一下

如：鼠标进去 `hover` 的时候过渡时长为 `2s`，鼠标离开失去 `hover` 效果的时候，过渡时长为 `5s` 怎么处理呢

```

<html>
  <head>
    <meta charset="utf-8">
    <title>标准的过渡效果</title>
    <style>
      .box{
        width: 100px;
        height: 100px;
        background-color: deeppink;
        transition-property: width;
        transition-delay: 0s;
        transition-timing-function: linear;
        transition-duration: 5s;    失去hover的时候，它是5s
      }
      /* 进来的时候是2s，离开的时候是5s */
      .box:hover{
        width: 300px;
        transition-duration: 2s;    进入hover的时候它是2s
      }
    </style>
  </head>
  <body>
    <div class="box">
    </div>
  </body>
</html>

```

像上面这种情况，就称之为不对称的过渡，它的进入效果和离开效果是不一样的

不能过渡的属性

在CSS3的过渡里面，并不是所有的属性都可以过过渡

1. `float`
2. `display`
3. `visibility`
4. `overflow`
5. `position`
6. `z-index`
7. `font-family`
8. `text-align/text-align-last`
9. `cursor`

同时 `auto` 的属性值也是不可以实现过渡效果的

兼容性处理

过渡是一个CSS3的属性，它在旧版本的浏览器里面是需要注意它兼容性

浏览器支持

表格中的数字表示支持该属性的第一个浏览器版本号。

紧跟在 -webkit-, -ms- 或 -moz- 前的数字为支持该前缀属性的第一个浏览器版本号。

属性	Chrome	Edge	Firefox	Safari	Opera
transition	26.0 4.0 -webkit-	10.0	16.0 4.0 -moz-	6.1 3.1 -webkit-	12.1 10.5 -o-
transition-delay	26.0 4.0 -webkit-	10.0	16.0 4.0 -moz-	6.1 3.1 -webkit-	12.1 10.5 -o-
transition-duration	26.0 4.0 -webkit-	10.0	16.0 4.0 -moz-	6.1 3.1 -webkit-	12.1 10.5 -o-
transition-property	26.0 4.0 -webkit-	10.0	16.0 4.0 -moz-	6.1 3.1 -webkit-	12.1 10.5 -o-
transition-timing-function	26.0 4.0 -webkit-	10.0	16.0 4.0 -moz-	6.1 3.1 -webkit-	12.1 10.5 -o-

对于低版本的浏览器，我们需要添加特定的前缀

1. 以谷歌为核心的浏览器，我们要添加 `-webkit-`
2. 以IE为核心的浏览器则要添加 `-ms-`
3. 火狐浏览器则是使用 `-moz-`
4. 欧朋浏览器使用 `-o-`

案例

1. 根据效果图完成如下效果



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>过渡的案例</title>
    <style>
      .switch-button {
        --h: 40px;
```

```
        width: 80px;
        height: var(--h);
        display: block;
        border-radius: calc(var(--h) / 2);
    }

.switch-button>input{
    display: none;
}
.switch-inner-box {
    width: 100%;
    height: 100%;
    background-color: #f5f5f5;
    border-radius: inherit;
    /* 过渡 */
    transition: all 0.3s linear;
}
.switch-button>input:checked+.switch-inner-box {
    background-color: #7dc9fd;
}
/* 小圆 */
.switch-button .circle {
    width: var(--h);
    height: var(--h);
    background-color: #ffffff;
    border-radius: 50%;
    transition: inherit;
    /* 把边框设置在小球上面 */
    border: 5px solid #f5f5f5;
    box-sizing: border-box;
}
.switch-button>input:checked+.switch-inner-box>.circle{
    margin-left: 40px;
    background-color: #bee3fd;
    /* 在切换过来以后，把边框的颜色变掉 */
    border: 5px solid #7ec9fd;
}

```

</style>

</head>

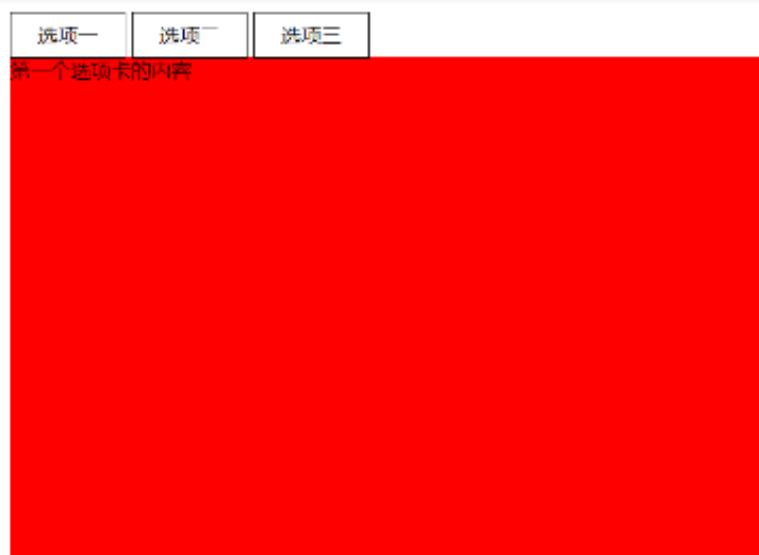
<body>

<!-- checkbox -->

<label class="switch-button">

```
<input type="checkbox">
<div class="switch-inner-box">
    <div class="circle"></div>
</div>
</label>
</body>
</html>
<!--
总结:
1.checked的伪类实现状态的切换
2.使用了CSS的变量及CSS的计算
3.使用了inherit来主动继承外部的样式
4.使用了display来切换元素类型
5.使用了过渡，让效果变得更加的简单
-->
```

2. 要据效果图完成案例



```
<!DOCTYPE html>
<html lang="zh">
    <head>
        <meta charset="UTF-8">
        <title>选项卡案例</title>
        <style>
            .clearfix::after {
                content: "";
                display: block;
                clear: both;
            }
        </style>
    </head>
    <body>
```

```
.box{
    width: 600px;
    /* 溢出隐藏 */
    overflow: hidden;
}
.label-box {
}
.label-box>label {
    display: block;
    width: 130px;
    height: 50px;
    border: 2px solid black;
    text-align: center;
    line-height: 50px;
    float: left;
    margin-right: 20px;
}
.tab-list{
    height: 400px;
    width: calc(600px * 3);
    /* 过渡 */
    transition: all 0.5s linear;
}
.tab-item{
    width: 600px;
    height: 100%;
    float: left;
}
.tab-item:nth-child(1){
    background-color: red;
}
.tab-item:nth-child(2){
    background-color: blue;
}
.tab-item:nth-child(3){
    background-color: lightblue;
}
#a1:checked~.tab-list{
    margin-left: 0;
}
#a2:checked~.tab-list{
    margin-left: -600px;
```

```

    }
    #a3:checked~.tab-list{
        margin-left: -1200px;
    }
    .box>input{
        display: none;
    }

```

</style>

</head>

<body>

```

<div class="box">
    <!-- 三个label标签 -->
    <div class="label-box clearfix">
        <label for="a1">选项一</label>
        <label for="a2">选项二</label>
        <label for="a3">选项三</label>
    </div>
    <input type="radio" name="aaa" id="a1">
    <input type="radio" name="aaa" id="a2">
    <input type="radio" name="aaa" id="a3">
    <div class="tab-list clearfix">
        <div class="tab-item">第一个选项卡</div>
        <div class="tab-item">第二个选项卡</div>
        <div class="tab-item">第三个选项卡</div>
    </div>
</div>

```

</body>

</html>

<!--

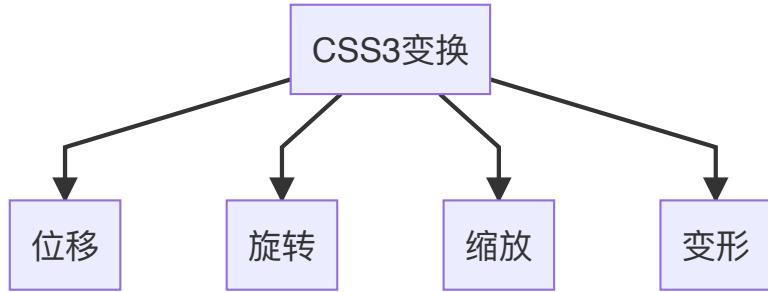
总结:

1. 目前的状态切换可以有checked, target, hover, active,
这里我们使用是checked, 并且是单选
2. 合理的布局, 选择器只是能哥哥找弟弟 ,
所以input标签是要在切换元素的前面
3. 添加过渡效果
4. 要适合的使用新知识, 如calc

-->

CSS3变换

CSS3里面的变换指的是页面上面的元素发生了位置，角度，大小，形状的变化



它使用 `transform` 来做为属性名，主要就是上面四种情况下面的变化，同时也分为2D变换与3D变换

位移

位移指的是元素的X轴，Y轴或Z轴的位置发生了变化，它执行左负右正，上负下正的原来的位置在移动

1. `translateX(大小)` 在水平方向发生位置移动
2. `translateY(大小)` 在垂直方向发生位置移动
3. `translateZ(大小)` 在Z轴方向上面移动 【默认是不会有效果的，需要开启3D空间】
4. `translate(X轴大小, Y轴大小)` 同时在X轴与Y轴方向移动，默认值是0
5. `translate3d(X轴, Y轴, Z轴)` ,同时在三个轴上面移动，默认值是0 【默认是没有效果的，需要开启3D空间】

`translate` 的位置有移动有2个特点

1. 它与相对定位非常像，没有脱流，不占用新的位置，也不放弃旧的位置
2. 它位移的时候如果是以百分比位移的，则是以自身的百分比为参照来进行的【重要】

缩放

缩放指元素的大小像气球一样进行缩放，它也是有X轴，Y轴和Z轴的缩放

1. `scaleX(1)` 在X轴发生缩放，默认值是1
2. `scaleY(1)` 在Y轴发生缩放，默认值是1
3. `scaleZ(1)` 在Z轴发生缩放，默认值是1 【默认Z轴是没有效果的，需要开启3D空间】
4. `scale(X轴, Y轴)` 同时在X轴与Y轴上面发生缩放，如果只有一个值，则第二个值与第一个值保持一致
5. `scale3d(X轴, Y轴, Z轴)` 同时在3根轴上面进行缩放 【默认Z轴是没有效果的，需

要开启3D空间】

旋转

旋转指的是一个元素在X轴，Y轴或Z轴发生了角度的变化

1. `rotateX(0deg)` 元素沿着X轴发生旋转
2. `rotateY(0deg)` 元素沿着Y轴发生旋转
3. `rotateZ(0deg)` 元素沿着Z轴发生旋转，顺时针为正数
4. `rotate(0deg)` 元素沿着Z轴发生旋转
5. `rotate3d(X轴, Y轴, Z轴)` 同时在3根轴上面旋转【没有开启3D的情况下是无效果】

 小技巧：根据右手原则，右手握住轴的方向是正值

旋转的单位有2个，第一个是 `deg` 代表度数，第二个是 `turn` 代表圈， $1\text{turn}=360\text{deg}$

变形

1. `skewX(0deg)` 在X轴发生倾斜
2. `skewY(-0deg)` 在Y轴发生倾斜
3. `skew(X轴, Y轴)` 同时在X轴与Y轴发生倾斜，默认值是0

多个变换的结合

上面我们已经学过了4种变换情况，那么如果一个元素想进行多种变换，怎么办呢？

```
transform: 变换1 变换2 ...;  
/* 元素，放大，再X轴移动100px,再Z轴旋转45deg */  
transform: scale(1.5) translateX(100px) rotateZ(45deg);
```

注意事项

在进行旋转的时候，三根轴有可能会发生变化

```
/* 这个元素在哪里 */  
transform: rotateZ(90deg) translateX(200px);
```

在上面的代码里面，因为先进行了Z轴的旋转90deg,所以X轴的方向发生了变化，这个时候它是向下移动的

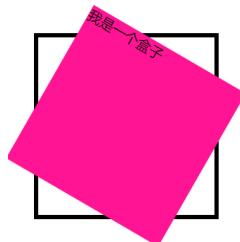
变换的其它属性

变换使用的属性 `transform`，它除了这个属性以外还有其它的属性

设置变换的起点位置

我们在进行变换的时候，默认都会有一个变换的起点位置，如下代码所示

```
width: 100%;  
height: 100%;  
background-color: deeppink;  
transform: rotateZ(30deg);
```



这个时候我们可以看到，Z轴旋转的时候，变换的默认的起点位置在中间，那么，我们能不能更改这个位置呢？

设置变换的起点我们需要通过 `transform-origin` 来完成，它后面跟1-2个属性值，默认值是 `center`

```
transform-origin: center center;
```

第一个属性值代表X轴的变换起点位置，第二个属性值代表Y轴的变换起点位置

当上面的代码里面，我们把旋转的起点位置设置在右上角以后

```
transform-origin:right top;
```



根据上面的代码，我们可以看到，它的属性值可以

是 `left/right/top/bottom/center` 来表示，还可以是具体的像素值来表示，如下所示

```
transform-origin:0px 100px;  
trasnform-origin:0px center;  
transform-origin 100px top;
```

backface-visibility属性

这个属性是用于设置当元素如果发生变换以后，背对用户的时候是否可见，默认是可见的，它有2个属性值

1. `visible` 背对用户时可见【默认值】
2. `hidden` 背对用户时不可见

【重点】形成的3D变换的条件

针对3D的场景我们一定要弄清楚，有真3D和假3D的区别，如下图所示



在上面的图片里面，它看起来像个3D，其实是一个假3D，也叫平面3D



而在上面的场景里面，我们看到的就是真3D效果

形成平面3D【视距，景深】

平面3D就是看起来有3D的效果，但是本身还是没有形成Z轴，在专业的说话里面叫视距，也景深

视距：人物的视角看某一个东西的距离叫视距，视距有一个特点，它是近的东西看起来大一些，远处的东西看起来小一些



这个时候所展现出来的效果，并不好，我希望实一个“远小近大”的效果

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The DOM tree on the left shows a `<div class="box">` containing an ``. The right panel shows the 'Styles' tab with the following CSS:

```
element.style {  
}  
.small-box {  
    width: 100%;  
    height: 100%;  
    transform: rotateX(40deg);  
}
```

A red box highlights the `transform: rotateX(40deg);` line, and a red arrow points from the text "在这个上面，我们让元素沿着X轴旋转40度" to this line.



这个时候图片在进行旋转的时候，就会有“远小近大”的效果

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The DOM tree on the left shows a `<div class="box">` containing an ``. The right panel shows the 'Styles' tab with the following CSS:

```
element.style {  
}  
.box {  
    width: 200px;  
    height: 200px;  
    border: 5px solid black;  
    margin: 100px;  
    perspective: 200px; 视距  
}
```

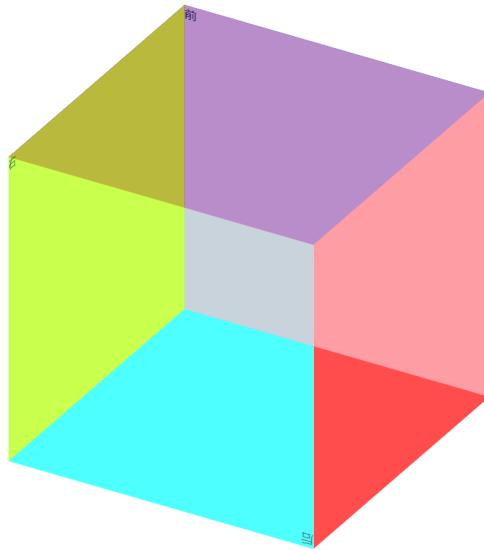
A red box highlights the `perspective: 200px;` line, and a red arrow points from the text "视距" to this line.

形成平面3D效果的基本条件就是在变换元素的外层添加 `perspective`

真3D的实现【3D空间的开启】

真正的3D它应该是有Z轴的，现在我们就来开启Z轴，开启3D，现在我们通过下面的案例来学习

```
transform-style: preserve-3d; /*开启3D空间*/  
transform-style: flat; /*平面的2D空间，默认是个值*/
```



```
<!DOCTYPE html>  
<html lang="zh">  
  
<head>  
    <meta charset="UTF-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>真3D</title>  
    <style>  
        .box {  
            width: 200px;  
            height: 200px;  
            border: 2px solid black;  
            margin: 200px;  
            transform: rotateX(10deg) rotateY(10deg);  
            /* 将当前这个盒子转换成3D空间 /开启3D空间 */  
            transform-style: preserve-3d;  
            /* 默认的2D平面空间 */  
        }  
    </style>  
</head>  
<body>  
    <div class="box">
```

```
        /* transform-style: flat; */
        position: relative;
    }

    .box>div {
        width: 100%;
        height: 100%;
        position: absolute;
        left: 0;
        top: 0;
    }

    .front {
        background-color: rgba(255, 0, 0, 0.3);
        transform: translateZ(100px);
    }

    .back {
        background-color: rgba(0, 255, 0, 0.3);
        transform: rotateY(180deg) translateZ(100px);
    }

    .left {
        background-color: rgba(0, 0, 255, 0.3);
        transform: rotateY(90deg) translateZ(100px);
    }

    .right {
        background-color: rgba(110, 110, 110, 0.3);
        transform: rotateY(-90deg) translateZ(100px);
    }

    .top{
        background-color: rgba(0, 0, 0, 0.3);
        transform: rotateX(90deg) translateZ(100px);
    }

    .bottom{
        background-color: rgba(255, 255, 0, 0.3);
        transform: rotateX(-90deg) translateZ(100px);
    }
}

</style>
</head>
```

```

<body>
  <div class="box">
    <div class="front">前</div>
    <div class="back">后</div>
    <div class="left">左</div>
    <div class="right">右</div>
    <div class="top">上</div>
    <div class="bottom">下</div>
  </div>
</body>

</html>

```

变换的注意事项

- 所有的行内元素都不支持 `transform` 变换的，除非转换成块级或行内块级
- 不要将 `background-color` 与 `preserve-3d` 一起使用



不要将`background-color`与`preserve-3d`一起使用

```

09变换的注意事项.html > html > head > style > .box
1 <!DOCTYPE html>
2 <html lang="zh">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width,
7     <!--<title>变换的注意事项</title>-->
8     <style> text/javascript </script>
9   </body> .box{
10   width: 400px;  不要将background-color与
11   height: 400px;  preserve-3d一起使用
12   border: 2px solid black;
13   background-color: pink;
14   perspective: 400px;
15   /* transform-style: preserve-3d; */
16 } 算 布局 事件侦听器 DOM 断点 属性 辅助功能

```

在上面的代码里面，我们一旦同时使用，会有问题



现在将这2个属性同时使用了

```

09变换的注意事项.html > html > head > style > .box
1 <!DOCTYPE html>
2 <html lang="zh">
3   <head>
4     <div><meta charset="UTF-8">
5     <div><meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <div><meta name="viewport" content="width=device-width,
7     <!--<title>变换的注意事项</title>-->
8     <style> text/javascript </script>
9   </body> .box{
10   width: 400px;  现在将这2个属性同时使用了
11   height: 400px;
12   border: 2px solid black;
13   background-color: pink;
14   perspective: 400px;
15   transform-style: preserve-3d;
16 } 算 布局 事件侦听器 DOM 断点 属性 辅助功能

```

兼容性

属性	Chrome	Edge	Firefox	Safari	Opera
transform	36.0 12.0 -webkit-	10.0	16.0 10.0 -moz-	4.0 -webkit-	23.0 15.0 -webkit-
transform-origin (three-value syntax)	36.0 12.0 -webkit-	10.0	16.0 10.0 -moz-	4.0 -webkit-	23.0 15.0 -webkit-
transform-style	36.0 12.0 -webkit-	11.0	16.0 10.0 -moz-	4.0 -webkit-	23.0 15.0 -webkit-
perspective	36.0 12.0 -webkit-	10.0	16.0 10.0 -moz-	4.0 -webkit-	23.0 15.0 -webkit-
perspective-origin	36.0 12.0 -webkit-	10.0	16.0 10.0 -moz-	4.0 -webkit-	23.0 15.0 -webkit-
backface-visibility	36.0 12.0 -webkit-	10.0	16.0 10.0 -moz-	4.0 -webkit-	23.0 15.0 -webkit-

CSS3渐变

渐变指的是元素的背景进行颜色渐变，它使用 `background-image`。渐变的形式可以分为三种渐变

1. 线性渐变
2. 径向渐变
3. 圆锥渐变

线性渐变

线性渐变指元素沿着一个角度或方向发生颜色变化，线性渐变使用 `linear-gradient` 来完成，它的语法格式如下

```
background-image: linear-gradient( to 方向, 颜色1 [开始位置] [结束位置], 颜色2 [开始位置] [结束位置] ... );
```

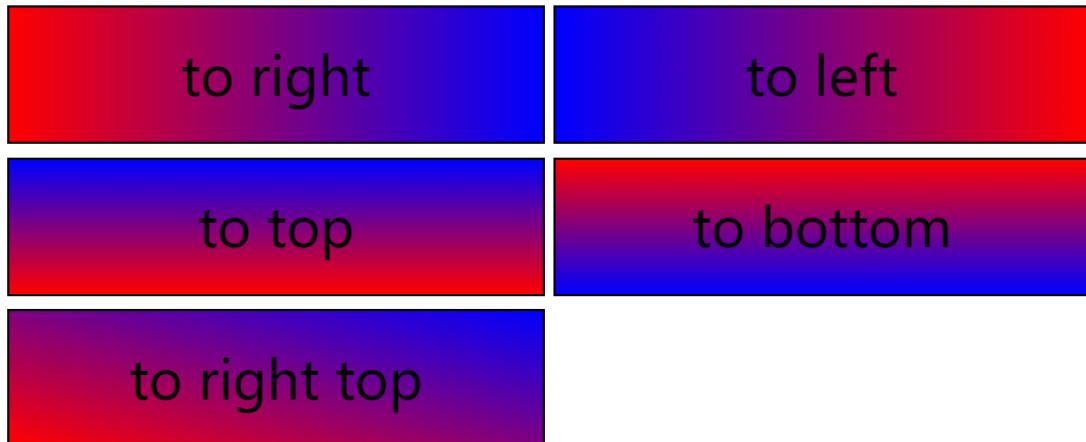
线性渐变的方向

```
.box1{  
    background-image: linear-gradient(to right, red, blue);  
}  
.box2{  
    background-image: linear-gradient(to left, red, blue);  
}  
.box3{  
    background-image: linear-gradient(to top, red, blue);  
}
```

```
}

.box4{
    background-image: linear-gradient(to bottom, red, blue);
}

.box5{
    background-image: linear-gradient(to right top, red, blue);
}
```



总结：在上面的代码里面，我们可以看到，它的方向是可以通过
left/right/top/bottom 来结合而成的

线性渐变的角度

渐变除了通过方向来渐变以外，还可以通过角度来实现渐变

```
.box1 {
    background-image: linear-gradient(90deg, red, blue);
}

.box2{
    background-image: linear-gradient(180deg, red, blue);
}

.box3{
    background-image: linear-gradient(270deg, red, blue);
}

.box4{
    background-image: linear-gradient(360deg, red, blue);
}

.box5{
    background-image: linear-gradient(45deg, red, blue);
}
```



总结：正上方就是0度，然后顺时针绕一圈是360度。度角的渐变比方向的渐变更加灵活

注意事项：在线性渐变里面，有一种兼容性的写法是加 `-webkit-` 的写法

```
background-image: -webkit-linear-gradient(right, red, blue);
```

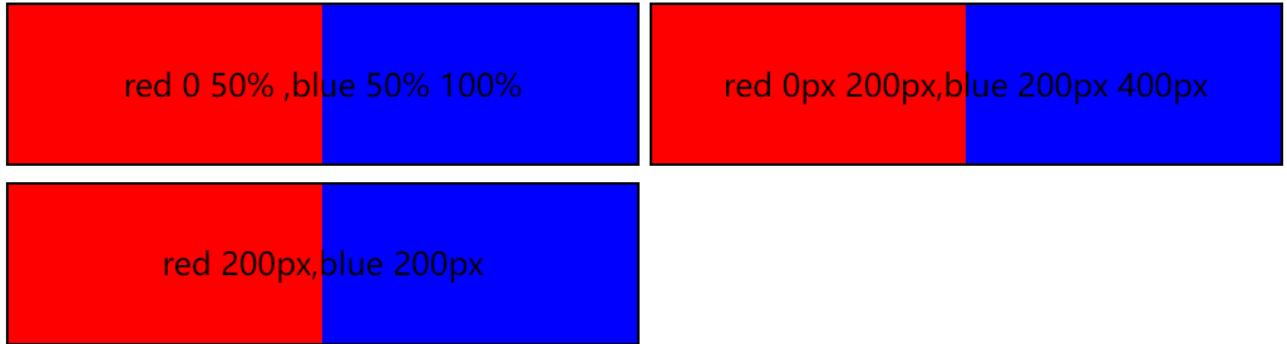
在兼容性写法里面，它的位置是开始位置，不是结束位置，并且没有 `to`

线性渐变的位置

```
.box1{
    /*使用百分比做为开始与结束位置*/
    background: linear-gradient(to right, red 0 50%, blue 50% 100%);
}

.box2{
    /*使用具体的像素值做为开始与结束位置*/
    background: linear-gradient(to right, red 0px 200px, blue 200px 400px);
}

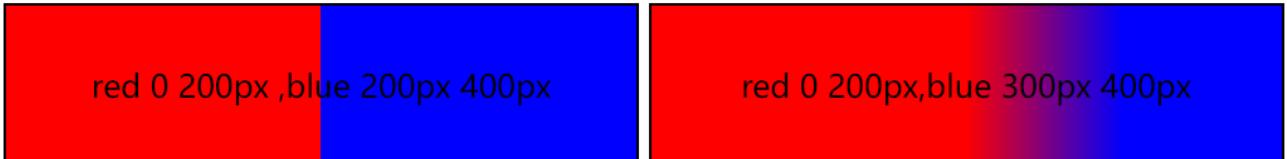
.box3{
    /*第一个颜色开始值可以省略，省略以后是0*/
    background-image: linear-gradient(to right, red 200px, blue 200px);
    /*最后一个颜色结束值可以省略，省略以后就是100%*/
}
```



在上面的代码里面，我们分别对颜色值进行了开始与结束位置的设置，但有一个特点，后一个颜色的开始位置就是前面一个颜色结束，这样会造成2个颜色的变化非常明显

在进行颜色的位置设置的时候，如果后面一个颜色的开始位置与前面一个颜色的结束位置之间还有空间，则这个空间就使用两种颜色进行渐变

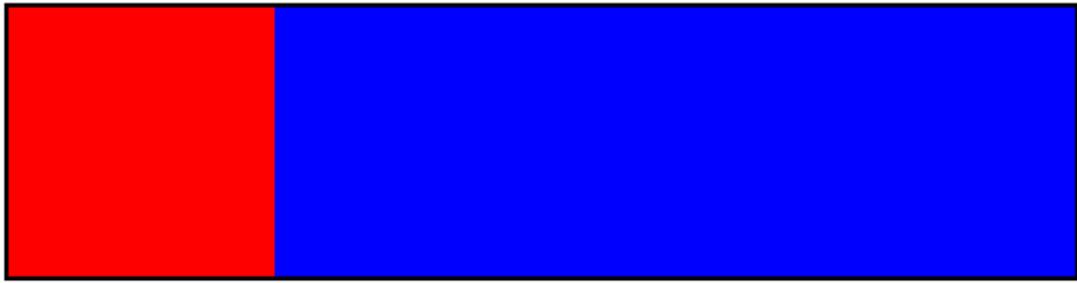
```
.box1{
    background: linear-gradient(to right, red 0 200px ,blue 200px
400px);
}
.box2{
    background-image: linear-gradient(to right, red 0 200px,blue 300px
400px);
}
```



重复线性渐变

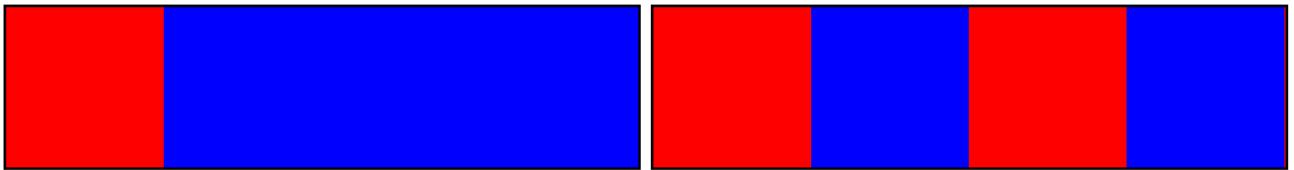
当在进行渐变色的设置的时候，如果一个渐变色不能铺满怎么元素，则剩下的部分将会用最后一个颜色进行填充

```
.box1{
    /*盒子的总宽度是400px,但是我们只设置到了200px的位置*/
    background: linear-gradient(to right, red 0px 100px,blue 100px
200px);
}
```



现在请看下面的效果

```
.box1{  
    background: linear-gradient(to right, red 0px 100px,blue 100px  
200px);  
}  
.box2{  
    background: repeating-linear-gradient(to right,red 0px 100px,blue  
100px 200px);;  
}
```



在第二个图里面，我们可以看到，当我们设置了 `repeating-linear-gradient` 以后，当渐变填充不满的时候，剩下的部分它会重复平铺进行

强调：

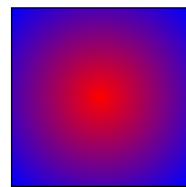
因为渐变使用的是 `background-image`，所以渐变应该算是一个背景图片，而背景图片应该是可以使用多次（多背景设置）

径向渐变

径向渐变指元素按照一定的半径发生变化，它使用 `radial-gradient` 来完成，它的语法格式如下

```
background-image:radial-gradient([形状 at 横坐标 纵坐标],颜色1 [开始位置]  
[结束位置]...);
```

```
.box1{  
    background: radial-gradient(red,blue);  
}
```

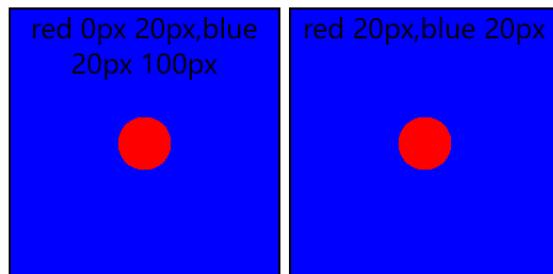


我们可以看到，颜色由内向外进行了渐变

径向渐变的位置

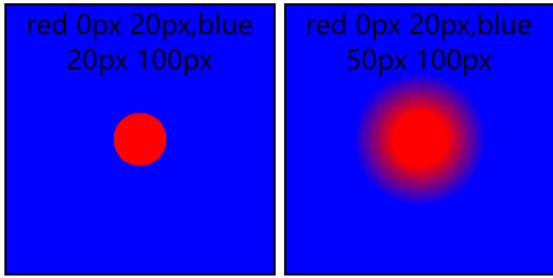
径向渐变也是可以像线性渐变一样设置起始与结束位置的

```
.box2{  
    background:radial-gradient(red 0px 20px,blue 20px 100px);  
}  
.box3{  
    background: radial-gradient(red 20px,blue 20px);  
}
```



同时，我们也知道，如果后一个颜色的开始位置与第一个颜色的结束位置相同，则颜色会变得非常清晰，如果后一个颜色开始位置与前一个颜色的结束位置存在空间，则会进行渐变。如下所示

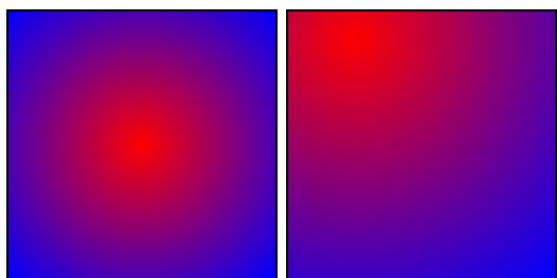
```
.box2{  
    background:radial-gradient(red 0px 20px,blue 20px 100px);  
}  
.box3{  
    background: radial-gradient(red 0px 20px,blue 50px 100px);  
}
```



径向渐变圆心位置

默认情况下，径向渐变都是从正中心开始的，它的圆心在正中心，我们其实是可以设置这个圆心的位置的，如下所示

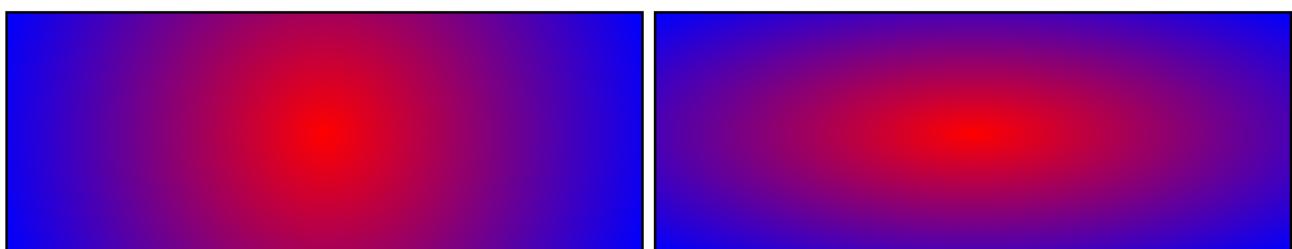
```
.box1 {  
    background: radial-gradient(circle at 100px 100px, red, blue);  
}  
.box2{  
    background: radial-gradient(circle at 50px 20px, red, blue);  
}
```



如果不设置圆心，则默认在正中间

径向渐变的形状

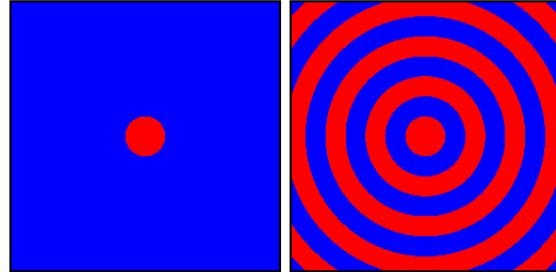
```
.box1 {  
    background: radial-gradient(circle, red, blue);  
}  
.box2{  
    background: radial-gradient(ellipse, red, blue);  
}
```



重复径向渐变

它与重复渐线渐变是一样的，当渐变颜色填充不满元素的时候，我们就需要使用这个东西

```
.box1{  
    background: radial-gradient(red 0px 15px, blue 15px 30px);  
}  
.box2{  
    background-image: repeating-radial-gradient(red 0px 15px, blue 15px 30px);  
}
```



圆锥渐变

圆锥渐变指的是颜色沿着一定的角度旋转去发生渐变，它使用 `conic-gradient`，它的语法格式如下

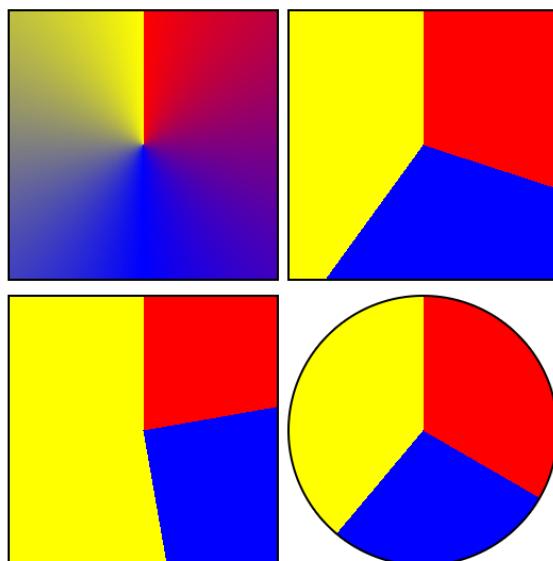
```
background:conic-gradient(颜色1 [开始位置] [结束位置],...);
```

这里的开始位置与结束位置的设置跟之前所讲过了线性渐变及径向渐变都是一样的，所以在这里就不再赘述

```

.box1{
    background: conic-gradient(red,blue,yellow);
}
.box2{
    background: conic-gradient(red 0% 30%,blue 30% 60%,yellow 60% 100%);
}
.box3{
    background: conic-gradient(red 0deg 80deg,blue 80deg 170deg,yellow 170deg 360deg);
}
.box4{
    background: conic-gradient(red 0deg 120deg,blue 120deg 220deg,yellow 220deg 360deg);
    border-radius: 50%;
}

```



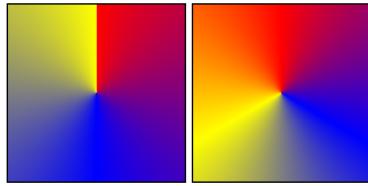
圆锥渐变它的起始位置与结束位置可以设置百分比，也可以设置角度

小技巧：如果在圆锥渐变里面，它的开始的颜色和最后一个结束的颜色相同，则会实现无缝渐变

```

.box1{
    background: conic-gradient(red,blue,yellow);
}
.box2{
    background: conic-gradient(red,blue,yellow,red);
}

```



重复圆锥渐变

重复的圆锥渐变与之前的2个是一样的，属性是 `repeating-conic-gradient`

CSS3动画

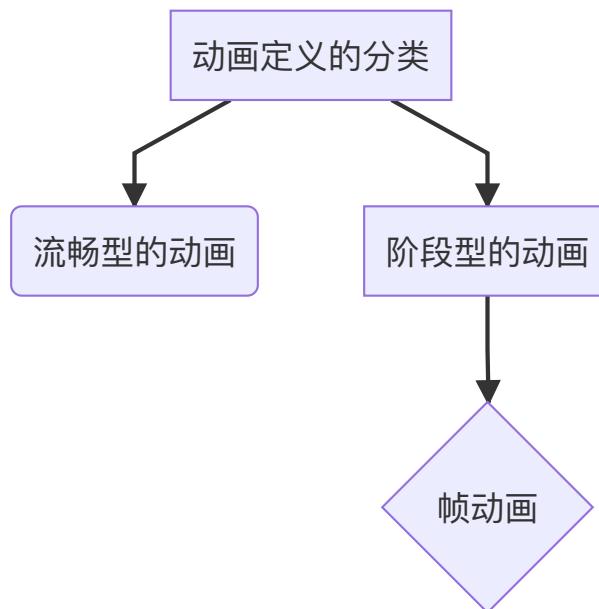
CSS3里面的动画与过渡非常相似，过渡有4个属性，而CSS3里面的动画有8个属性

CSS3里面的动画的学习是分2个阶段的



CSS动画定义

动画从定义的结果或方式上面去划分可以分为下面几类



定义CSS动画需要使用到CSS里面的命令 `@keyframes`，它的语法格式如下

```
@keyframes 动画名称{  
    from{}  
    to{}  
}
```

上面的 `from` 代表动画的开始， `to` 代表动画的结束

除了上面的 `from/to` 来定义开始与结束以外，我们还可以通过百分比的方式来进行

```
@keyframes 动画名称{  
    0%{}  
    25%{}  
    30%{}  
    100%{}  
}
```

在上面的定义方式里面，我们的 `0%` 相当于 `from`，我们的 `100%` 相当于 `to`，使用百分比定义的好处是可以在任意时刻定义状态，如 `25%`, `30%` 等

注意：并不是所有的动画都需要开始与结束的

```
/*省略开始*/  
@keyframes 动画名称{  
    100%{}  
}  
/*省略结束*/  
@keyframes 动画名称 {  
    0%{}  
    40%{}  
}  
/*即省略开始，也省略结束*/  
@keyframes 动画名称{  
    50%{}  
    80%{}  
}
```

CSS动画的使用

当一个动画定义好了以后，其它的元素就可以调用这个动画。动画的调用主要是通过下面的8个属性来完成

1. `animation-name` 动画的名称 【必填】
2. `animation-duration` 动画执行一次的时间 【必填】
3. `animation-iteration-count` 动画重复的次数 【默认值是1】，如果希望动画一直重复执行，则可以通过设置 `infinite` 无穷大来实现
4. `animation-timing-function` 动画执行的时间函数 【默认值是 `ease`】，如果希望动画匀速执行可以设置 `linear`，这里面的属性值和 `transition-timing-function` 保持一致
5. `animation-delay` 动画的等待时间 【默认值为0】
6. `animation-direction` 动画执行的方向，【默认值 `normal`】
 - `normal` 正常的
 - `reverse` 逆向的
 - `alternate` 正向与逆向交替运行
 - `alternate-reverse` 逆向与正向交替运行
7. `animation-play-state` 动画的播放状态
 - `running` 运行状态 【默认值】
 - `paused` 暂停状态
8. `animation-fill-mode` 动画在结束以后停留在什么状态
 - `backwards` 回到开始状态
 - `forwards` 停留在结束状态

上面的8个属性也可以结合成1个属性 `animation`

```
animation: 动画名称 动画时间 [次数] [时间函数] [等待时间] [方向] [播放状态] [结束状态];
```

上面的属性值是可以任意更改位置的，但是要注意，上面有珍个播放时间与等待时间，第一个时间是动画播放时间，第二个时间才是等待时间

多个动画的使用

之前在讲过渡的时候，我们一个元素可以执行多个属性的过渡，现在在动画里面，一个元素也可以同时使用多个动画，如下所示

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>多动画</title>
    <style>
        /* 第一个动画定义了小球的缩放 */
        @keyframes box-1 {
            0% {
                transform: scale(0.8);
            }

            50% {
                transform: scale(1.2);
            }

            100% {
                transform: scale(0.8);
            }
        }

        /* 第二个动画, 定义小球的移动 */
        @keyframes box-2 {
            0% {
                margin-left: 0;
            }

            100% {
                margin-left: 500px;
            }
        }

        .box {
            width: 50px;
        }
    </style>
</head>
<body>
    <div class="box" style="background-color: red; height: 50px; border-radius: 50%;">
```

```

        height: 50px;
        background-color: deeppink;
        border-radius: 50%;
        /* 使用动画 */
        animation: box-1 1s infinite linear,
                    box-2 10s linear forwards;
    }

```

</style>

</head>

<body>

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

</body>

</html>

在上面的代码里面，我们可以看到定久了2个动画，同时这2个动画全部都使用在了 **box** 元素上面，这就说明一个元素是可以同时使用多组动画的

上面的属性值是连起来写的，也可以分开来写

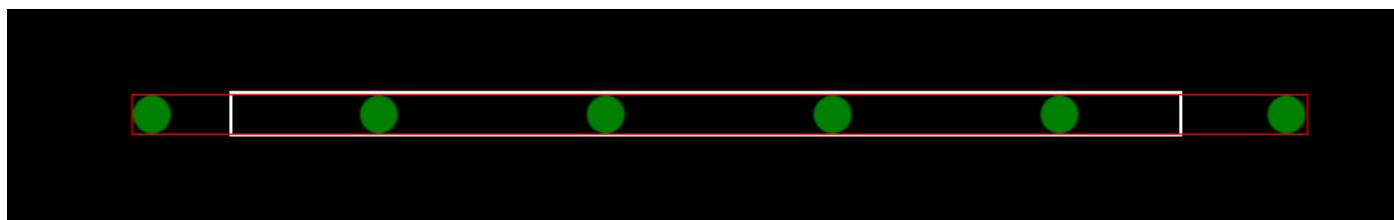
```

animation-name: box-1,box-2;
animation-duration: 1s,10s;
animation-iteration-count: infinite,1;
animation-timing-function: linear;
animation-fill-mode: unset,forwards;

```

CSS动画视觉差

视觉差这个词叫视觉欺骗，它是通过一个种特殊的技术来让人产生一种视觉错误解



在上面的图片里面，它就是一个视觉欺骗，给我们的感觉就是后面的豆子是无穷无尽的

```

<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>动画的视觉差</title>
<style>
  * {
    margin: 0;
    padding: 0;
    list-style-type: none;
  }

  body {
    background-color: #000000;
  }

  .outer-box {
    border: 5px solid #ff0000;
    width: calc(30px * 4 + 100px * 3);
    margin: 150px;
    overflow: hidden;
  }

  /* 动画的定位 */
  @keyframes bean-ani {
    0% {
      margin-left: 0;
    }
    100% {
      margin-left: -130px;
    }
  }

  .bean-list {
    border: 2px solid #ffffff;
    /* BFC */
    display: flow-root;
    width: calc(30px * 5 + 100px * 4);
    animation: bean-ani 3s linear infinite;
  }

  .bean-list>li {
    width: 30px;
    height: 30px;
    background-color: #ffffff;
    border-radius: 50%;
    float: left;
  }

```

```

        }
    .bean-list>li+li {
        margin-left: 100px;
    }

```

</style>

</head>

<body>

<div class="outer-box">

<ul class="bean-list">

</div>

</body>

</html>

阶段型动画

阶段型动画并不是一种新的动画，它只是一种特殊的动画的定义方式

```

<!DOCTYPE html>
<html lang="zh">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>阶段型动画</title>
    <style>
        .div1 {
            width: 100px;
            height: 100px;
            background-color: deeppink;
            animation: ani1 10s linear forwards;
        }

        @keyframes ani1 {
            0% {
                margin-left: 0;
            }

```

```

        }
        49.9%{
            margin-left: 0;
        }
        50%{
            margin-left: 400px;
        }
        99.9%{
            margin-left: 400px;
        }
        100%{
            margin-left: 800px;
        }
    }

```

</style>

</head>

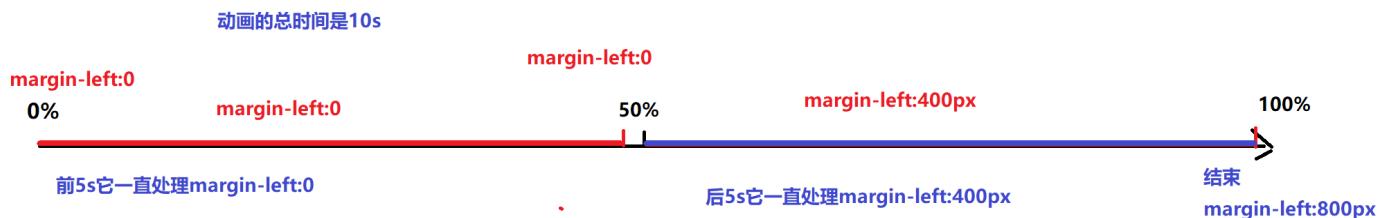
<body>

<div class="div1"></div>

</body>

</html>

在上面的动画定义方式里面，我们就可以把它认为是一种阶段型的动画



上面的动画定义方式可以直接简写成如下的定义方式

```

@keyframes ani1 {
    0%, 49.9% {
        margin-left: 0;
    }
    50%, 99.9%{
        margin-left: 400px;
    }
    100%{
        margin-left: 800px;
    }
}

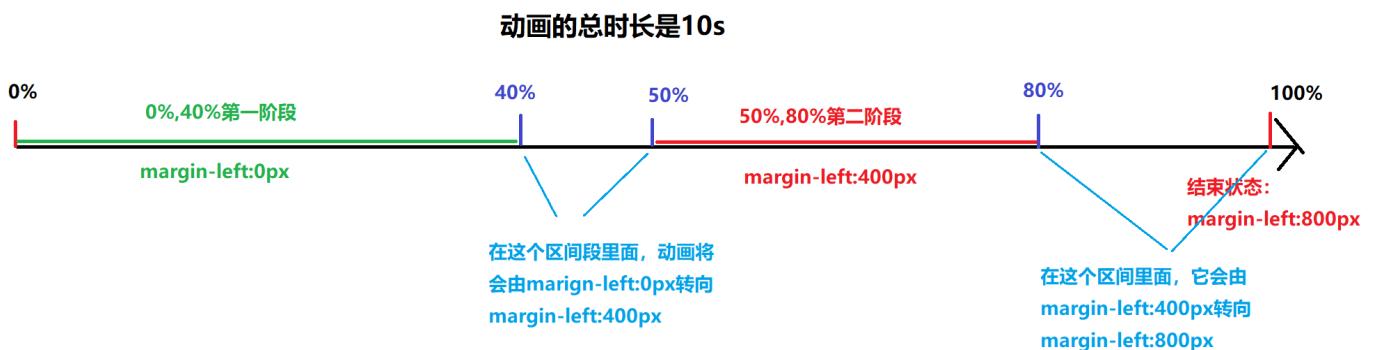
```

阶段型动画的定义在不同的阶段里面，我们也可以执行相应的过渡

```

@keyframes ani1 {
    0%, 40% {
        margin-left: 0;
    }
    50%, 80%{
        margin-left: 400px;
    }
    100%{
        margin-left: 800px;
    }
}

```





上面的代码就是复用阶段型动画及视觉差来实现的一个循环的滚动的轮播图

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>阶段型动画</title>
    <style>
        .outer-box {
            border: 5px solid red;
            width: 639px;
            overflow: hidden;
        }

        .img-box {
            height: 426px;
            width: calc(639px * 5);
            /* BFC */
            display: flow-root;
            animation: ani1 10s linear infinite;
        }

        .img-box>img {
            float: left;
        }

        @keyframes ani1{
            0%, 20%{
```

```

        transform: translateX(0);
    }
25.1%, 45%{
    transform: translateX(calc(639px * -1));
}
50.1%, 70%{
    transform: translateX(calc(639px * -2));
}
75.1%, 95%{
    transform: translateX(calc(639px * -3));
}
100%{
    transform: translateX(calc(639px * -4));
}
}
</style>
</head>
<body>
<div class="outer-box">
<div class="img-box">





</div>
</div>
</body>
</html>

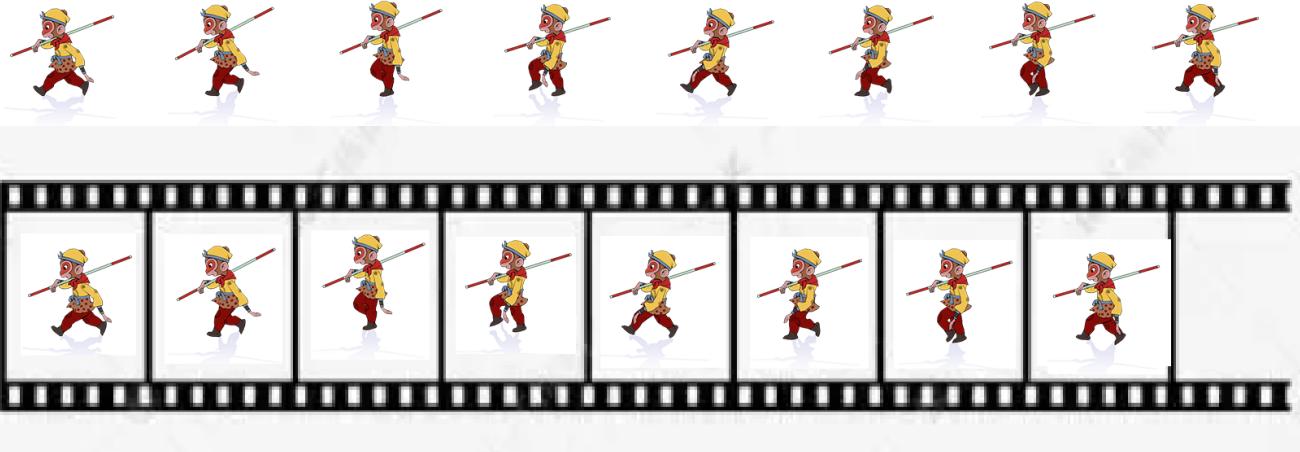
```

CSS帧动画

我们把特殊的阶段型动画叫帧动画



上面的图叫精灵图，我们可以把图片上面的每一个小图片放在一个类似于胶片的格子里面，如下所示



里面的每一个小图片，我们就可以认为是每一帧，而一帧也可以看是每一个阶段

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>帧动画</title>
    <style>
        .monkey-sun{
            width: 200px;
            height: 180px;
            border: 2px solid black;
            background-image: url("img/1.png");
            animation: monkey-sun-ani 1s linear infinite;
        }

        @keyframes monkey-sun-ani{
            0%, 12.5%{
                background-position-x: calc(200px * 0);
            }
            12.6%, 25%{
                background-position-x: calc(200px * -1);
            }
            25.1%, 37.5%{
                background-position-x: calc(200px * -2);
            }
            37.6%, 50%{
                background-position-x: calc(200px * -3);
            }
            50.1%, 62.5%{
```

```

        background-position-x: calc(200px * -4);
    }
62.6%, 75%{
    background-position-x: calc(200px * -5);
}
75.1%, 87.5%{
    background-position-x: calc(200px * -6);
}
87.6%, 100%{
    background-position-x: calc(200px * -7);
}
}
</style>
</head>
<body>
<div class="monkey-sun">

</div>
</body>
</html>

```

在上面的代码里面，我们使用的特殊的阶段型动画来完成上面的操作，让人物行走起来了
从上面的代码当中我们可以得到一个点，只要阶段型的动画分配合理，我们可以实现这一种一帧一帧播放的效果，但是问题就在于如果精灵图过多或者阶段过多，动画的定义就会变得非常复杂，所以CSS3里面推出专门用于帧动画的属性值

```

<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>真正的帧动画</title>
    <style>
        .box{
            width: 200px;
            height: 180px;
            border: 1px solid black;
            background-image: url("img/1.png");
            animation: box-ani 1s steps(8) infinite;
        }
    </style>

```

```

@keyframes box-ani{
    0%{
        background-position-x: 0;
    }
    100%{
        background-position-x: -1600px;
    }
}
</style>
</head>
<body>
    <div class="box"></div>
</body>
</html>

```

在上面的代码里面，我们使用了 `steps(8)` 来将动画自动的切换成了8帧运行

多列布局

多列布局是一种特殊的布局形式，它是CSS3里面新增的几个属性，使用起来非常简单

1. `column-count` 用于设置列的数量
2. `column-gap` 列与列之间的间距
3. `column-rule-style` 列与列之间线条的类型【它的类型与边框线条的类型相同】
4. `column-rule-color` 列与列之间线条的颜色
5. `column-rule-width` 列与列之间线条的宽度
6. `column-rule` 是上面3个属性的结合，相当于边框的3个属性
7. `column-span` 让某一个元素横跨多少列，它只有 `1/all` 两个属性值
8. `column-width` 每一列的最佳列宽度，它会自动帮我们生成列数

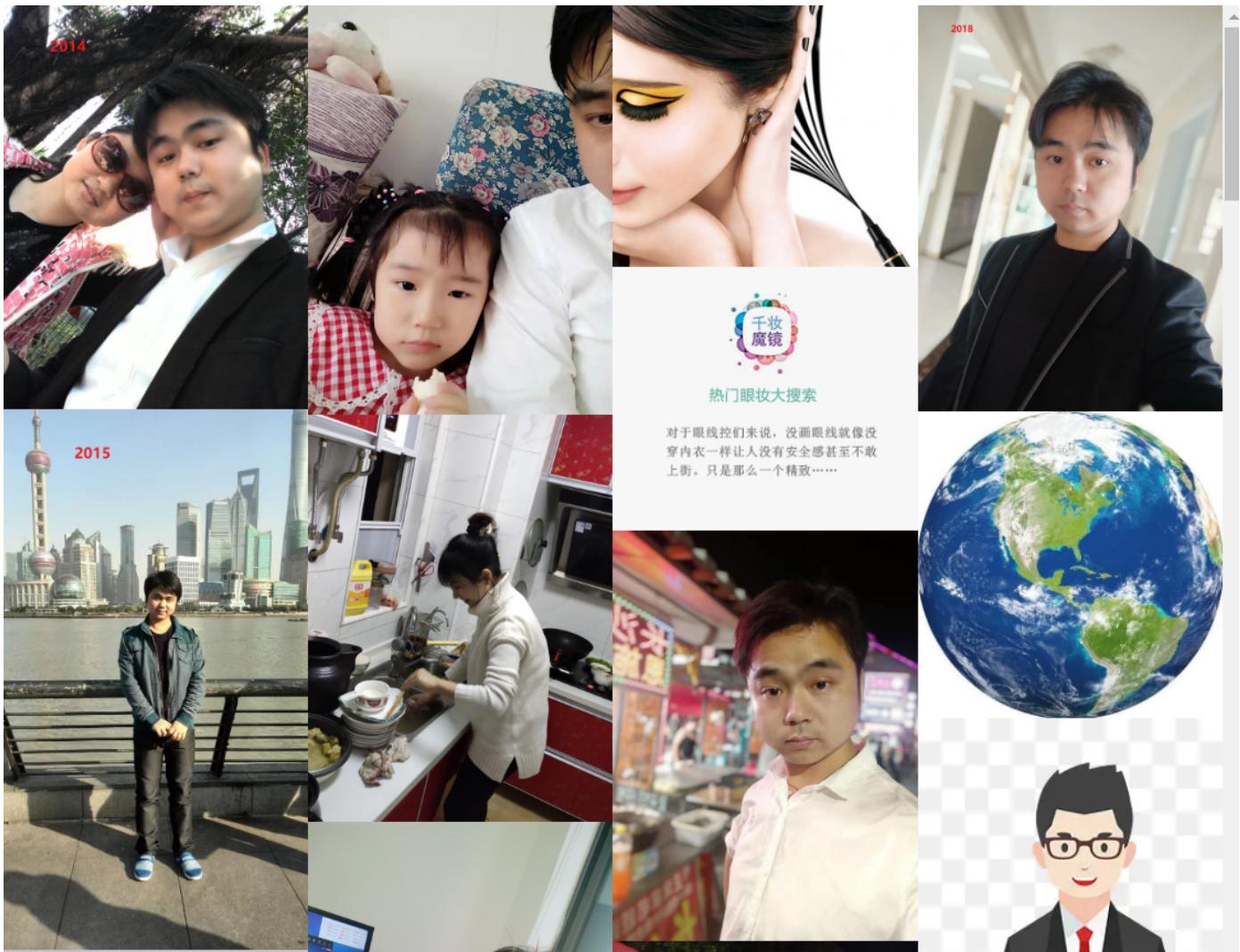
三国演义

第一回 宴桃园豪杰三结义 斩黄巾英雄首立功

滚滚长江东逝水，浪花淘尽英雄。是非成败转头空。青山依旧在，几度夕阳红。白发渔樵江渚上，惯看秋月春风。一壶浊酒喜相逢。古今多少事，都付笑谈中。——调寄《临江仙》 话说天下大势，分久必合，合久必分。周末七国分争，并入于秦。及秦灭之后，楚、汉分争，又并入于汉。汉朝自高祖斩白蛇而起义，一统天下，后来光武中兴，传至献帝，遂分为三国。推其致乱之由，殆始于桓、灵二帝。桓帝禁锢善类，崇信宦官。及桓帝崩，灵帝即位，大将军窦武、太傅陈蕃共相辅佐。

佐。时有宦官曹节等弄权，窦武、陈蕃谋诛之，机事不密，反为所害，中涓自此愈横。建宁二年四月望日，帝御温德殿。方升座，殿角狂风骤起。只见一条大青蛇，从梁上飞将下来，蟠于椅上。帝惊倒，左右急救入宫，百官俱奔避。须臾，蛇不见了。忽然大雷大雨，加以冰雹，落到半夜方止，坏却房屋无数。建宁四年二月，洛阳地震；又海水泛溢，沿海居民，尽被大浪卷入海中。光和元年，雌鸡化雄。六月朔，黑气十余丈，飞入温德殿中。秋七月，有虹现于玉堂；五原山岸，尽皆崩裂。种种不祥，非止一端。帝下诏问群臣以灾异之由，议郎蔡邕上疏，以为蜺堕鸡化，乃妇寺干政所致，言颇切直。帝览奏叹息，因起更衣。曹节在后窃

视，悉宣告左右；遂以他事陷邕于罪，放归田里。后张让、赵忠、封谞、段珪、曹节、侯览、蹇硕、程旷、夏恽、郭胜十人朋比为奸，号为“十常侍”。帝尊信张让，呼为“阿父”。朝政日非，以致天下人心思乱，盗贼蜂起。时巨鹿郡有兄弟三人，一名张角，一名张宝，一名张梁。那张角本是个不第秀才，因入山采药，遇一老人，碧眼童颜，手执藜杖，唤角至一洞中，以天书三卷授之，曰：“此名《太平要术》，汝得之，当代天宣化，普救世人；若萌异心，必获恶报。”角拜问姓名。老人曰：“吾乃南华老仙也。”言讫，化阵清风而去。角得此书，晓夜攻习，能呼风唤雨，号为“太平道人”。



多列形成的照片瀑布流

裁剪样式

裁剪指的是像剪刀一样去把一个元素的四周的四角进行裁剪，它的CSS属性是 `clip-path`

矩形裁剪

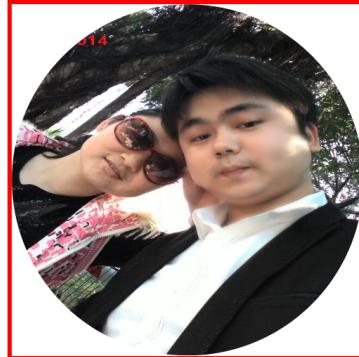
矩形裁剪使用的是 `inset` ,它的语法格式如下

```
clip-path:inset(上边 右边 下边 左边);  
/*还可以在裁剪的时候设置圆角*/  
clip-path:inset(上边 右边 下边 左边 round 圆角);
```

圆形裁剪

圆形裁剪使用的是 `circle`, 它的语法格式如下

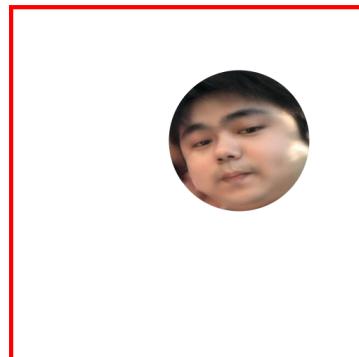
```
clip-path: circle(半径 at 圆心横坐标 圆心纵坐标);
```



对于上面的效果，我们现在已经可以通过三种方式来实现了

```
/*第一种：圆形裁剪*/  
clip-path: circle(200px at 200px 200px);  
/*第二种：圆角边框*/  
border-radius: 50%;  
/*第三种：矩形裁剪使用round*/  
clip-path: inset(0 0 0 0 round 200px);
```

虽然说我们已经有很多种方式去显示一个圆，但是如果想实现下面的效果，则只能使用圆形裁剪



```
clip-path: circle(80px at 257px 149px);
```

椭圆裁剪

椭圆裁剪使用的是 `ellipse`，它的语法格式如下

```
clip-path:ellipse(横轴半径 纵轴半径 at 横坐标 纵坐标);  
clip-path: ellipse(70px 100px at 110px 124px);  
/*当横轴半径 与 纵轴半径相同时，它又会变成一个圆*/  
clip-path: ellipse(100px 100px at 100px 100px);
```



多边形裁剪

多边形裁剪使用的是 `polygon`，它的语法格式如下

```
clip-path:polygon(x1 y1,x2 y2...);  
clip-path: polygon(100px 0,0 300px,300px 300px,200px 0);
```

这里要注意，它的点与点之前是用逗号隔开的

裁剪的注意事项

1. 裁剪并不会改变元素自身的大小，只是显示效果变了
2. 裁剪是从盒子模型的 `border` 开始的
3. 使用裁剪属性以后不会有 `outline` 的效果，也不会有外阴影的效果

CSS补充点

img内容的显示方式

在img图片手动的去调整大小以后，图片的宽高比可能就会失效，这个时候图片就会变形，我们可以通过 `object-fit` 来调整图片的内容的显示方式

1. `object-fit:cover` 覆盖

2. `object-fit:contain` 包含
3. `object-fit:fill` 填充【默认值，它会拉伸图片填充设定的宽度与高度】



上面的图片就分别是 `cover`, `contain`, `fill` 的情况

还有一个属性是 `object-position` 用于调整图片内容的位置

元素的宽高比

在设置元素的时候，我们经常要对元素的宽度与高度做比例处理，这个时候可以使
用 `aspect-ratio` 来完成‘

```
.div2{  
    background-color: blue;  
    /* 无论宽度怎么变，高度必须是宽度的2倍，怎么办 */  
    /* 外观比例，也叫宽高比 */  
    aspect-ratio: 1/2;  
    height: 200px;  
}
```

link与@import的区别

```
<link rel="stylesheet" href="css/a.css" type="text/css">  
<style>  
    @import url("css/b.css");  
</style>
```

我们如果要引入CSS的样式，我们既可以使用 `link` 标签也可以使用 `@import` 的命令

区别1： `link` 是标签，它是在HTML里面用的，它不仅可以连接一个CSS文件，还可以链接其它的东西，如图标；而 `@import` 它是一个CSS的命令，它只能在CSS里面使用，并且只能导入CSS的文件

```
<link rel="shortcut icon" href="./aaa.png">
```

区别2：`@import` 是CSS的命令，所以可以从一个CSS文件里面导入另一个CSS文件

```
/*a.css文件*/
@import url("b.css");
@import url("c.css");

/*后面再继续写CSS代码*/
```

在没有学习 `webpack` 或其它的类似的构建工具之前，请先不要使用

区别3：它们的加载顺序是有区别的

```
<link rel="stylesheet" href="css/a.css">
<style>
    @import url("css/b.css");
</style>
```

通过 `link` 连接的CSS是在页面加载的时候会同时加载样式，而 `@import` 命令导入的CSS是在页面完整加载完成以后才开始进行的，如果网速不好会造成页面空白或闪烁

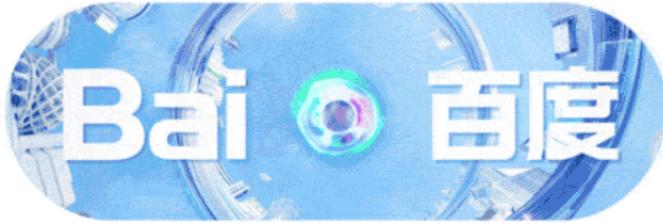
href与src的区别

`href` 与 `src` 都是HTML标签里面的属性，也都是用于链接地址的，但是它的区别非常大

1. `href` 的意思是去什么什么地方
2. `src` 的意思是把什么什么东西拿过来【加载过来，后期在JS里面就会有一个加载事件 `onload`】

```
<a
href="https://www.baidu.com/img/pc_27928619fedf2633952764032537980f.gif">图
片</a>
<hr>

```



通过上面的现象，我们可以看到，`a` 标签的 `href` 是去了某一个地方，而 `img` 标签的 `src` 则是把这个图片拿过来【加载过来】

```
<!-- 这里会有一个链接，跳去标哥博客 -->
<a href="http://www.softeem.xin:8090">标哥博客</a>
<hr>
<!-- 把标哥博客拿过来 嵌入在我的网页里面 -->
<iframe src="http://www.softeem.xin:8090" style="width: 100%;height: 500px;"></iframe>
```

因为 `link` 标签使用的是 `href` 属性，所以在通过 `link` 都会导入 CSS 文件的时候，要注意 CSS 文件里面的路由，因为 `href` 是以被连接的那个文件的路径为主

:valid 伪类

`:valid` 伪类表示表单标签验证通过以后

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>valid</title>
  <style>
    /*需要使用:valid的伪类
     指的是表单元素验证通过以后的状态
     第一步：先添加验证的要求，所以我们在输入框的上面添加了required必填
     那个验证要求就是必填
     第二步：设置验证通过以后的特殊状态伪类:valid;
    */
    .txt:valid{
```

```
}

span{
    display: none;
}
.txt:valid+span{
    display: inline;
}

```

</style>

</head>

<body>

 <form>

 <input type="text" class="txt" required>

 恭喜你通过了

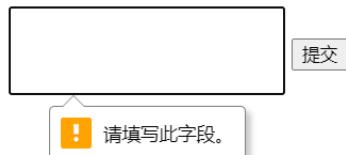
 </form>

</body>

</html>

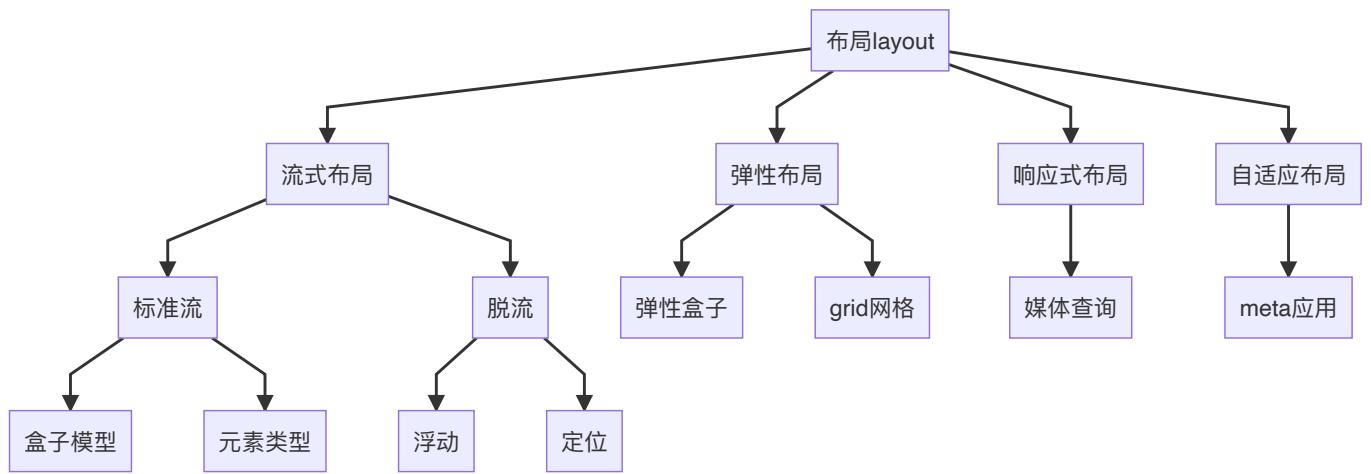
表单元素的required属性

required 意为必须的意思，当表单标签添加上这个属性的时候在点击提交按钮的时候如果该表单元素没有值则会提示如下信息



注意使用 required 属性时表单标签必须放在form标签内部

弹性盒子与弹性布局

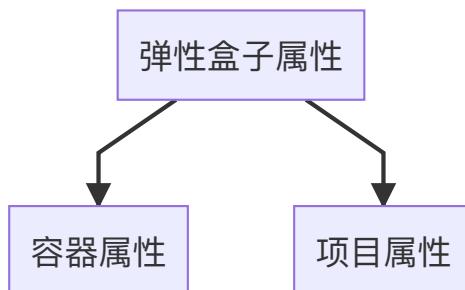


之前我们在学习布局的时候已经学习了流式布局标准，流式布局主要是针对于PC端传统网页来进行布局的，它的兼容性非常之高，所以也是一种很普遍的布局方式，但是传统的流式布局对于初学者来说会非常麻烦，并且也不适用于目前比较流行的移动端布局，所以后期W3C推出一种新的布局标准叫弹性布局

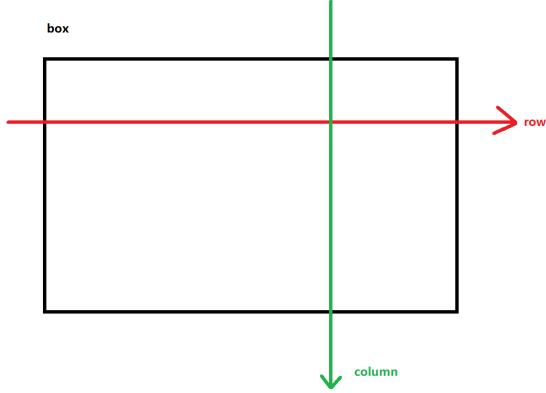
弹性布局是一系列的技术集合，最主要的就是指弹性盒子及grid网格

弹性盒子

弹性盒子是实现弹性布局主要技术，来是由一整套CSS的属性来完成的，它的属性分为2大类



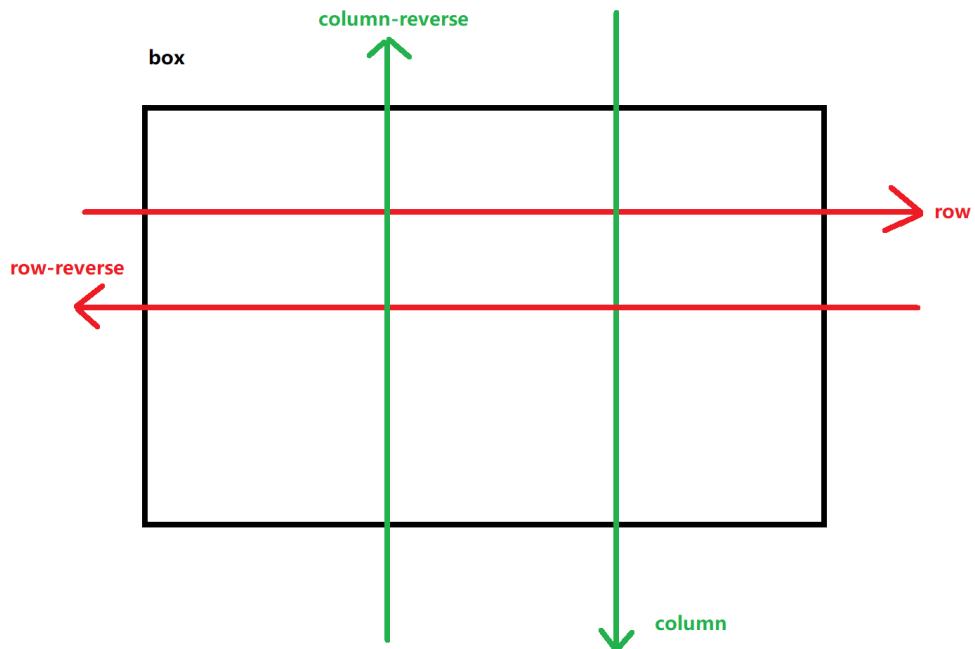
1. **display:flex** 将一个元素变成弹性盒子，当一个元素变成弹性盒子以后，它的内部就会多出2根轴，分别是水平方向的轴叫 **row** 和垂直方向的轴叫 **column**，并且在默认情况下会有一根轴叫主轴，默认的主轴是 **row**



从上面的图形里面我们可以看到 `row` 这个方向是主轴，弹性盒子里面的元素默认会沿着主轴排列，所以默认情况下，弹性盒子里面的元素会从左向右水平排列

2. `flex-direction` 调整弹性的方向，也就是调整主轴的方向

- `row` 主轴水平方向【默认值】
- `row-reverse` 主轴水平反转
- `column` 主轴为竖着方向
- `column-reverse` 主轴竖着反转



当一旦确定了主轴以后，另一个一根轴就叫副轴

3. `justify-content` 设置元素在主轴上面的排列

- `flex-start` 在弹性开始的位置【默认值】
- `flex-end` 在弹性结束的位置
- `center` 居中
- `space-between` 使用空间隔开
- `space-around` 空间环绕
- `space-evenly` 空间均匀分配



4. align-items 设置元素在副轴上面的排列

- **flex-start** 在弹性开始的位置
- **flex-end** 在弹性结束的位置
- **center** 居中
- **baseline** 基线对齐
- **stretch** 如果在副轴上面没有设置大小，则会拉伸【默认值】

5. flex-wrap 用于设置是否可以在主轴方向换行

默认情况下，弹性盒子的主轴是不允许换行的，这个时候就当内部元素的总大小大于盒子的大小的时候就会出现压缩，我们可以通过设置这个属性让盒子换行

- **nowrap** 不换行【默认值】
- **wrap** 换行
- **wrap-reverse** 换行以后再将副轴翻转

6. align-content 在副轴上面调整主轴的位置【多主轴的设置】，它的属性值与 justify-content 保持一致

- **flex-start** 在弹性开始的位置【默认值】
- **flex-end** 在弹性结束的位置
- **center** 居中
- **space-between** 使用空间隔开
- **space-around** 空间环绕
- **space-evenly** 空间均匀分配

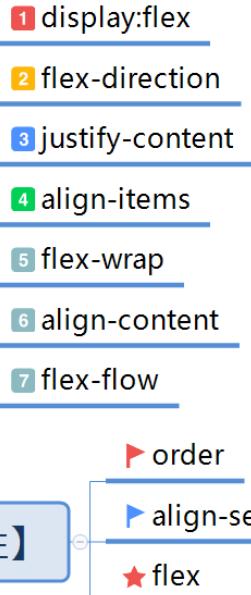
7. flex-flow 这个属性就是 flex-direction 及 flex-wrap 这两个属性的结合

```
flex-direction: row;
flex-wrap: wrap;
/*上面2行代码效果等同于下面的代码*/
flex-flow: row wrap;
```

8. order 用于设置弹性盒子内部元素在主轴的排列顺序，order的值越大，越靠近弹性结束的地方；order的值越小，越靠近弹性开始的地方

9. align-self 单独设置某一个项目在副轴的排列，它与 align-items 的属性值保持一致

10. flex 当前元素在弹性盒子容器里面占剩下空间的百分比



弹性盒子

设置在弹性盒子上面【容器属性】

设置在弹性盒子内部元素上面【项目属性】

弹性盒子的特征

当一个元素变成弹性盒子以后，它及内部的元素会有哪些变化呢？

对当前容器的影响

1. 弹性盒子会形成BFC
2. 弹性盒子默认情况下主轴设置 `overflow` 是无效的，因为它会压缩内部元素的大小【如果不希望内部元素被压缩，则需要将内部元素的压缩值 `flex-shrink:0`】

对容器内部元素的影响

1. 主轴收缩，副轴拉伸（通俗的理解就是宽度丢失），但是仍然可以通过 `width/height` 去重新设置宽度与高度
2. 弹性盒子内部的元素全部都是 `block` 类型（无视元素类型）
3. 弹性盒子内部的元素是不允许使用 `float` 的，用了也没有效果
4. 弹性盒子内部的元素是可以使用 `position` 的，也可以通过 `left/right/top/bottom` 来设置位置

当弹性布局与定位布局结合的时候，默认是听弹性布局的【因为定位是在当前位置直接脱流，不主动调整位置】，如果定位的元素使用了 `left/right/top/bottom` 则以定位为标准

5. 弹性盒子内部的元素可以通过 `flex` 属性来调整大小

弹性切割与流式切割

在之前讲解流式布局的时候，我们可以通过定位来对页面上面的元素空间进行切割，如三栏式布局等，现在我们来复习一下



现在上面的要求，盒子的总宽度是不固定的，左边的盒子1的宽度是 100px，如何将剩下的空间给盒子2

第一种思路：流式切割

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>流式切割</title>
    <style>
        .box{
            height: 100px;
            border: 5px solid black;
            position: relative;
        }
        .div1{
            width: 100px;
            height: 100px;
            background-color: deeppink;
        }
        .div2{
            /* 如果把剩下的给第这个盒子 */
            background-color: lightseagreen;
            position: absolute;
            top: 0;
            left: 100px;
        }
    </style>
</head>
<body>
    <div class="box">
        <div class="div1"></div>
        <div class="div2"></div>
    </div>
</body>
</html>
```

```
        height: 100%;  
        right: 0;  
    }  
  
```

```
</style>  
</head>  
<body>  
  <div class="box">  
    <div class="div1">1</div>  
    <div class="div2">2</div>  
  </div>  
</body>  
</html>
```

在定位切割里面，我们主要使用的是 `left/right/top/bottom` 来拉开定位的流，让元素的大小得到满足

第二种思路：弹性切割

```
<!DOCTYPE html>  
<html lang="zh">  
<head>  
  <meta charset="UTF-8">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>流式切割</title>  
  <style>  
    .box{  
      height: 100px;  
      border: 5px solid black;  
      display: flex;  
      flex-direction: row;  
    }  
    .div1{  
      width: 100px;  
      height: 100px;  
      background-color: deeppink;  
    }  
    .div2{  
      /* 如果把剩下的给第这个盒子 */  
      background-color: lightseagreen;  
      flex: 1;  
    }  
  
```

```

        }
    </style>
</head>
<body>
    <div class="box">
        <div class="div1">1</div>
        <div class="div2">2</div>
    </div>
</body>
</html>

```

```

6         <meta name="viewport" content="width=device-width, initial-sca
7             <title>流式切割</title>
8             <style>
9                 /* 如果把剩下的给第这个盒子 */
10                .box{
11                    background-color: lightseagreen;
12                    height: 100px;
13                    position: absolute;
14                    border: 5px solid black;
15                    top: 0;
16                    left: 100px;
17                    display: flex;
18                    flex-direction: row;
19                    height: 100%;
20                    right: 0;
21                }
22                .div1{
23                    width: 100px;
24                    height: 100px;
25                    background-color: deeppink;
26                }
27            </style>
28        </head>
29        <body>
30            <div class="box">
31                <div class="div1">1</div>
32                <div class="div2">2</div>
33            </div>
34        </body>
35    </html>

```

我们主要使用的是 left/right/top/bottom 来拉开定位的流，让元素的大小得到满足。

flex: 1;

添加这个属性



弹性盒子属性的封装

在使用弹性布局的时候，我们会大量使用到弹性盒子，同时弹性盒子的属性有几个会经常使用，所以我们会将这些经常使用的属性进行封装，以方便快速布局

flex-box.css文件

```
@charset "utf-8";
/* 弹性盒子的封装 */
.flex-row{
    display: flex;
    flex-direction: row;
}
.flex-col{
    display: flex;
    flex-direction: column;
}
.j-c{
    justify-content: center;
}
.j-s-b{
    justify-content: space-between;
}
.j-s-a{
    justify-content: space-around;
}
.j-s-e{
    justify-content: space-evenly;
}
.a-c{
    align-items: center;
}
.flex-1{
    flex: 1;
    overflow: auto;
}
```

当对上面的弹性盒子CSS属性封装好了以后，我们就可以在布局的时候直接使用了

```
<div id="app" class="flex-col">
  <div class="content-box flex-1">上</div>
  <ul class="tab-bar flex-row j-s-a">
    <li class="flex-col a-c j-s-a">
      
      微信
    </li>
    <li class="flex-col a-c j-s-a">
      
      通讯录
    </li>
    <li class="flex-col a-c j-s-a">
      
      发现
    </li>
    <li class="flex-col a-c j-s-a">
      
      我
    </li>
  </ul>
</div>
```

弹性盒子伸缩值计算

```
<!DOCTYPE html>
<html lang="zh">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>弹性盒子</title>
  <style>
    .box{
      width: 500px;
      height: 100px;
      border: 1px solid black;
      display: flex;
      flex-direction: row;
```

```

    }
    .div1{
        width: 100px;
        height: 100px;
        background-color: deeppink;
    }
    .div2{
        background-color: lightgray;
        flex: 1;
    }
    .div3{
        background-color: lightsalmon;
        flex: 1;
    }

```

</style>

</head>

<body>

```

<div class="box">
    <div class="div1">1</div>
    <div class="div2">2</div>
    <div class="div3">3</div>
</div>

```

</body>

</html>



在上面的代码里面，我们使用了 `flex:1` 来扩展后面的2个盒子，那么这2个盒子的宽度各是多少呢？

首先我们要知道一点， `flex` 属性是三个属性值的缩写

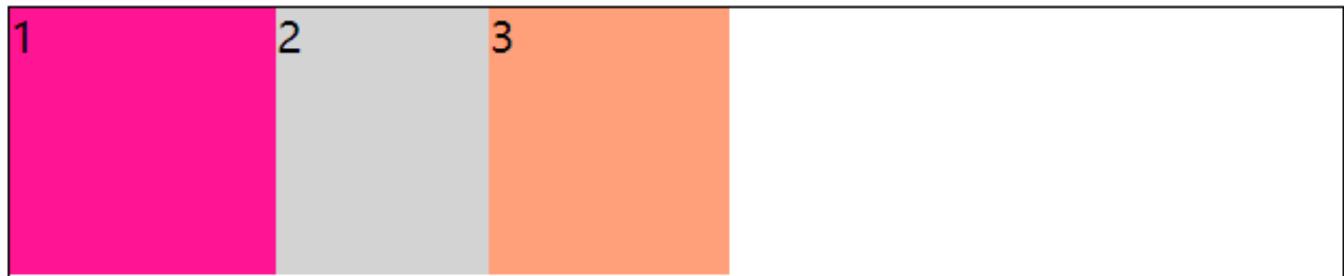
CSS 属性 `flex` 规定了弹性元素如何伸长或缩短以适应flex容器中的可用空间。这是一个 **简写属性**，用来设置 `flex-grow`，`flex-shrink` 与 `flex-basis`。

1. `flex-grow` 代表弹性的扩展比例

2. `flex-shrink` 代表弹性的收缩比例
3. `flex-basis` 代表弹性盒子的基准值

弹性盒子扩张值计算

当一个容器大于弹性盒子内部元素的大小总和的时候，我们可以使用 `flex-grow` 来让内部的元素自动扩张，以填满剩下的空间



```
.box{  
    width: 500px;  
    height: 100px;  
    border: 1px solid black;  
    display: flex;  
    flex-direction: row;  
}  
.div1{  
    width: 100px;  
    height: 100px;  
    background-color: deeppink;  
}  
.div2{  
    background-color: lightgray;  
    width: 80px;  
}  
.div3{  
    background-color: lightsalmon;  
    width: 90px;  
}
```

在上面的代码里面，盒子总的宽度是 `500px`，内部三个盒子总宽度加起来 $100+80+70=250$ ，所以总的宽度应该是小于 `500px`

接下来，我想让 `div2` 与 `div3` 自动填充剩下的空间，则 `div2` 与 `div3` 怎么设置，设置以后宽度各是多少呢

```

.div1{
    width: 100px;
    height: 100px;
    background-color: deeppink;
    /*扩张*/
    flex-grow: 1;
}

.div2{
    background-color: lightgray;
    width: 80px;
    /*扩张*/
    flex-grow: 2;
}

.div3{
    background-color: lightsalmon;
    width: 90px;
    /*扩张*/
    flex-grow: 3;
}

```

现在我们在后面的2个元素上面分别添加了 `flex-grow` 来扩展，则后面的2个元素的宽度具体值是多少呢？

计算公式，在这里我们先计算 `div1`

1. 第一步：计算剩下的空间是多少

$$500 - 100 - 80 - 90 = 230$$

2. 计算总的份数

$$1 + 2 + 3 = 6$$

3. 计算自己扩张的值

$$1 / 6 * 230 = 38.333$$

4. 总的宽度

$$100 + 38.333 = 138.333$$

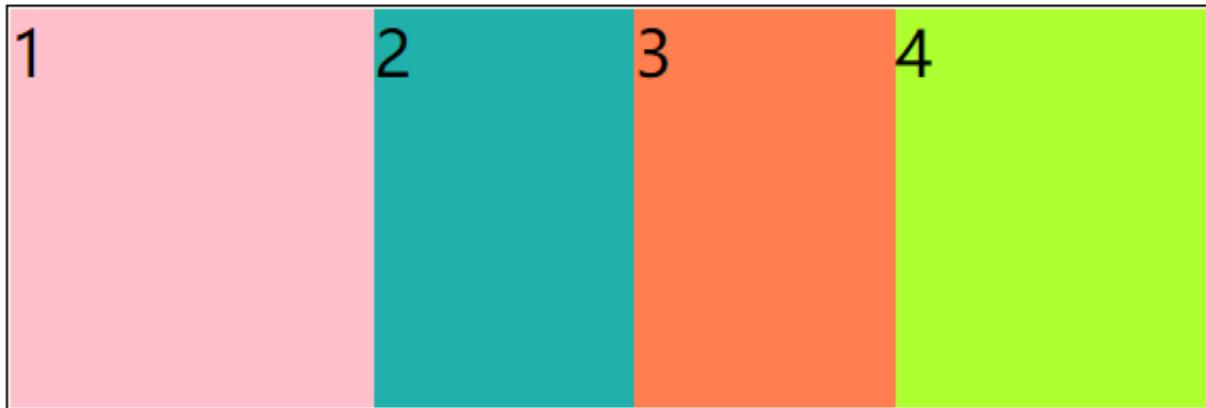
最终经过计算以后，我们得出结果就是 **138.333**



弹性盒子收缩值的计算

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>弹性盒子收缩值的计算</title>
  <style>
    .box{
      width: 300px;
      height: 100px;
      border: 1px solid black;
      display: flex;
    }
    .div1{
      width: 140px;
      background-color: pink;
    }
    .div2{
      width: 100px;
      background-color: lightseagreen;
    }
    .div3{
      width: 100px;
      background-color: coral;
    }
    .div4{
      width: 120px;
      background-color: greenyellow;
    }
  </style>
</head>
<body>
```

```
<div class="box">
  <div class="div1">1</div>
  <div class="div2">2</div>
  <div class="div3">3</div>
  <div class="div4">4</div>
</div>
</body>
</html>
```



现在在一个盒子里面放了4个盒子，每个盒子的宽度都不一样，最终照的结果是盒子总的宽度大于外边的盒子的宽度，这个时候就会出现一个压缩情况，那么每个盒子真实的大小是多少呢？

计算公式，在这里我们先计算 **div1**

1. 第一步：计算自己差多少

$$300 - 140 - 100 - 100 - 120 = -160$$

2. 计算总的份数

$$1 \times 140 + 1 \times 100 + 1 \times 100 + 2 \times 120 = 580$$

3. 计算自己的收缩比例

$$140 \times 1 / 580 = 0.2413$$

4. 计算自己收缩的值

$$160 \times 0.2413 = 38.620$$

5. 计算自己剩下多少

$$140 - 38.620 = 101.379$$

关于 `flex-basis` 的问题

`flex-basis` 是元素的主轴大小基准值,在一个弹性盒子的内部,如果设置了 `flex-basis` 以后,则在计算压缩比例的时候要用不用这个 `flex-basis` 而不是使用 `width/height` 所设置的值

```
width: 140px;  
background-color: pink;  
flex-shrink: 2;  
flex-basis: 150px;
```

在上面的计算过程里面,应该使用 `flex-basis` 来计算

```
/*如果有了弹性的基准值,则使用基准值来计算*/  
/*  
第一步:计算差多少  
400 - 150 - 100 - 100 - 120 = -70  
第二步:计算总的份数  
150*2 + 100*1 + 100*1 + 120*2 = 740  
第三步:计算自己的收缩比例  
150*2 / 740 = 0.4054  
第四步:计算自己收缩的值  
0.4054 * 70 = 28.3783  
第五步:计算自己剩下多少  
150 - 28.3783 = 121.6216  
*/
```

Note: 当一个元素同时被设置了 `flex-basis` (除值为 `auto` 外) 和 `width` (或者在 `flex-direction: column` 情况下设置了 `height`) , `flex-basis` 具有更高的优先级.

总结

```
flex:1;  
flex: 1 1 auto;  
flex: 扩张 收缩 基准;
```

基准值如果是 `auto` 则代表与宽度,高度相同

扩展：如何形成全屏的盒子

形成全屏的盒子有多种方法，同时也要注意这个技术非常重要

第一种方法：通过固定定位实现

这一种方法是流式布局当中很常见的一种方法

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>全屏盒子设置</title>
    <style>
        #app{
            width: 100%;
            height: 100%;
            background-color: pink;
            /* 以浏览器为标准定位 */
            position: fixed;
            left: 0;
            top: 0;
        }
    </style>
</head>
<body>
    <div id="app"></div>
</body>
</html>
```

第二种方法：通过设置html,body来实现

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>全屏盒子设置</title>
    <style>
```

```

        *{
            margin: 0;
            padding: 0;
            list-style-type: none;
        }

        html, body{
            width: 100%;
            height: 100%;
        }

        #app{
            width: 100%;
            height: 100%;
            background-color: pink;
        }
    
```

</style>

</head>

<body>

<div id="app"></div>

</body>

</html>

第三种：使用非标准模式【知道就可以了，不要用】

在非标准模式下面，网页的高度默认会变成100%，触发非标准模式只用删除网页的声明头，如HTML5的网页里面就删除下面的代码

```
<!DOCTYPE html>
```

第四种：使用vw/vh来进行设置

vw 全称是 **viewport width**

viewport 的全称是视口，就是用来显示的窗口或看得见的窗口，在目前做移动端的网页或响应式的网布，我们都需要设置一个 **viewport**

```
<meta name="viewport" content="width=device-width,initial-scale=1,user-scalable=no,minimum-scale=1,maximum-scale=1" />
```

- **width=device-width** 代表当前视口的宽度与设备的宽度相同
- **initial-scale** 代表初始缩放
- **user-scalable=no** 代表不允许用户缩放
- **minimum-scale** 最小缩放比例
- **maximum-scale=1** 最大缩放比例

通过上面的设置以后，我们的 `viewport` 的宽度就与设备的宽度相同的

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>全屏盒子</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        *{
            margin: 0;
            padding: 0;
        }
        #app{
            background-color: red;
            /* 100vw代表的就是视口viewport 的 width的100% */
            width: 100vw;
            /* 100vh代表的就是视口viewport 的 height的100% */
            height: 100vh;
        }
    </style>
</head>
<body>
    <div id="app"></div>
</body>
</html>
```

移动端布局规范

调整设计稿



对于像上面这种标准的移动端APP页面，它有几个特点

1. 它的设计稿的宽度与高度应该是
 - 375px * 667px
 - 750px * 1334px
2. 浏览器的 iphone 6/7/8 的大小中 375px * 667px，如果设计稿不上面的大小，后期我们就要手动的去计算设计稿的比例
3. 如果设计稿是375px的宽度，则可以在pxcook直接使用
4. 如果设计稿的大小是750px，则需要在 pxcook 里面手动的调整为 2x 大小 【现在只需要计算设计稿的大小就可以了，把设计还是要还原了1x的大小】



移动端布局之前的准备工作

1. 准备图标与切图文件，图标如果是矢量图标一定要转换成 **iconfont**
2. 添加视口 **viewport**

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

3. APP开发如无特殊规定，应该设置全屏盒子
4. 计算设计稿的大小

```
:root{  
    font-size:calc(100vw / 设计稿的宽度 * 基准值);  
}
```

5. 恢复 **body** 标签的 **font-size**，不然网页的默认字体会变成非常大

```
body{  
    font-size:16px;  
}
```



在这个设计稿里面，只有设计为全屏以后，我才能进行上下部分的切割

```
<!DOCTYPE html>  
<html lang="zh">  
<head>  
    <meta charset="UTF-8">
```

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>消息提醒</title>
<link rel="stylesheet" href="css/flex-box.css">
<style>
  * {
    margin: 0;
    padding: 0;
    list-style-type: none;
  }

  :root {
    --primaryColor: #ffc81f;
    /* font-size: calc(375px / 750 * 100); */
    font-size: calc(100vw / 750 * 100);
  }

  body {
    font-size: 16px;
  }

  #app {
    width: 100vw;
    height: 100vh;
  }
</style>
</head>
<body>
  <div id="app"></div>
</body>
</html>
```

开发中的注意事项

1. 标准的APP界面上面应该分为标题部分与内容部分，标题部分应该是固定不变的，内容部分是可以根据内容的多少进行滚动

所以第一件事情就是切割页面

```
<div id="app" class="flex-col">
  <div class="title-bar">标题</div>
  <div class="content-box flex-1">内容</div>
</div>
```

上面是结构

```
#app{
  width: 100vw;
  height: 100vh;
}
.title-bar{
  background-color: red;
  height: 46px;
}
.content-box{
  background-color: pink;
}
```

2. 设置整个项目的主题色

```
:root{
  --primaryColor:#ffc81f;
}
```

3. 移动端的容器不应该是设置固定的 `px` 宽度， 默认应该就是整个设备的宽度， 但也不要设置 `width:100%`
4. 弹性盒子在主轴上面设置 `overflow` 默认是有问题的
5. 在移动端页面布局完成以后， 最好添加一个响应式， 防止移动端的页面在PC端打开的时候不适应的情况

```
@media only screen and (min-width:768px) {  
    /* 说明当前的网页已经不在手机设备上面了 */  
    /* 这个时候我们就要让我们的网页呈现出一个固定的大小 375px */  
    :root {  
        font-size: calc(750px / 750 * 100);  
    }  
    #app{  
        width: 750px;  
        margin: auto;  
    }  
}
```

Web前端的图标技术

在开发网页的时候我们可能需要使用很多个小图标，图标的技术经过发展以后有了很大的变化。以前的PC端我们可以使用 精灵图 来完成网页图标的使用

第一代图标

第一代图标使用图片来完成



第一代图标的使用非常简单，直接使用img就可以了

```

```

但是使用图片做为图标就会有问题

```
.img1{  
    width: 200px;  
    height: 200px;  
}
```

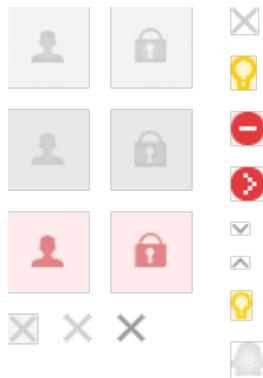
如果我们去改变图片的大小，这样就会失真



这一种图标的优点就是使用简单，缺点也很明显，不能够放大，一旦放大就失真，它还有一个缺点就是如果图标过多，就会到服务器加载多次

第二代图标的使用

第二代图标使用的是精灵图或纯粹的图片来完成，如 `.jpg/.png/gif` 等图标



```
<div class="box"></div>
<style>
    .box{
        width: 38px;
        height: 38px;
        border: 1px solid black;
        background-image: url("img/pwd-icons-new.png");
    }
</style>
```

上面的图标在使用的时候很麻烦，它将多个图标放在了一起，需要通过背景图的形式来定位图片。但也有优点，它只需要加载一次就可以了

第二代图标相较于第一代图标它就是把所有的小图片都集中在了一起，然后再进行相应的加载，这个会减少加载次数，但是并不会解决图标放大失真的问题

第三代图标：矢量图标

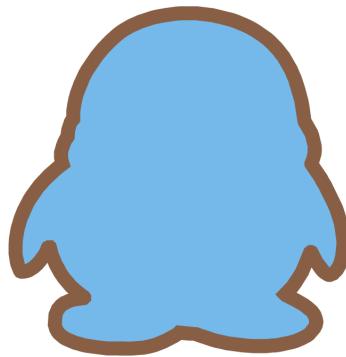
矢量图有一个特点，就是无论你怎么放大，也不会失真，矢量图标一般以 **svg** 结尾

现在我们用矢量图标来完成一次操作

```
<style>
    .img1{
        width: 400px;
        height: 400px;
    }
</style>

```

我们将图片放大到了 **400px** 的大小，这个时候 图标因为是矢量图格式，所以没有变模糊，也没有失真



问题：

1. 如果有多个图标，仍然要加载多个img，非常麻烦
2. 矢量图标不好更改颜色

第四代：将矢量图标变成iconfont

所有的图标都可以像字体一样去使用和设置，这就是第4代图标 **iconfont**

<https://www.iconfont.cn>-阿里巴巴矢量图标库

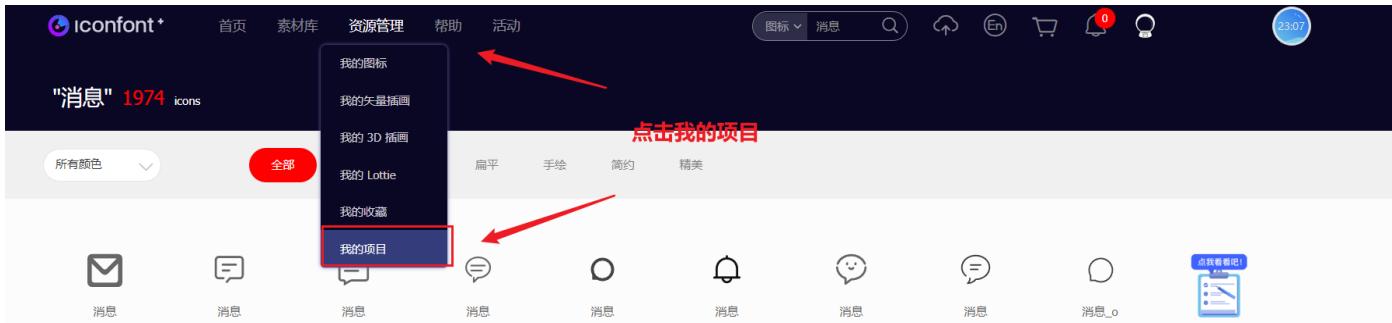
第一步：注册登录

这一个网站是专门为各们同学提供矢量图标的网站，我们需要先注册一下

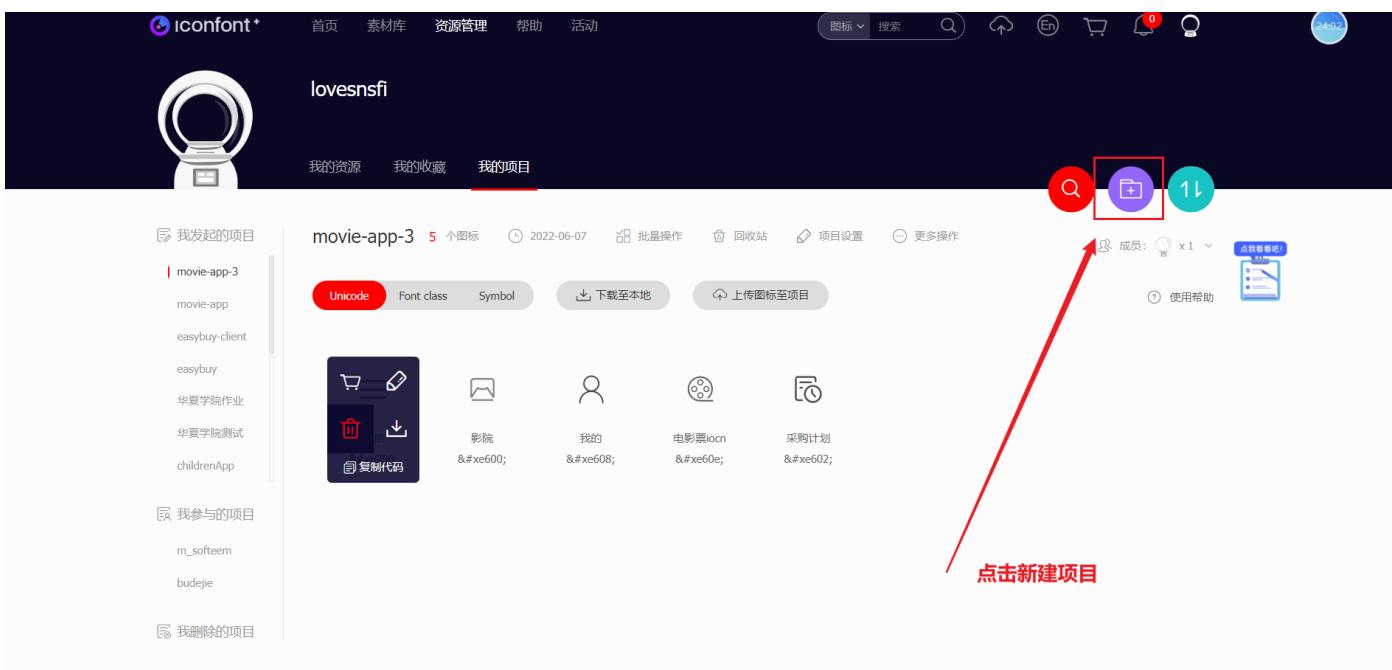


注册完毕以后就要登录进来了

第二步：点击我的项目



第三步：新建项目



新建项目

项目名称

H2204

项目描述

H2204班讲课的时候用到的图标

FontClass/
Symbol 前缀

icon-

Font Family

iconfont

字体格式

- 彩色 ②
- WOFF2
- WOFF
- TTF
- EOT
- SVG
- Base64 ②

所有者 (超级
管理员)

lovesnsfi(如需修改, 请在更多操作里转让项目)

协作者

请在「项目成员」中统一管理

新建

取消

第四步：挑选图标

The screenshot shows the iconfont.cn website interface. At the top, there is a navigation bar with links for '首页', '素材库', '资源管理', '帮助', and '活动'. A search bar contains the text '图标 龙虾' with a magnifying glass icon. Below the search bar, a red arrow points to the search input field with the text '输入需要搜索的图标' (Input the icon you want to search). The main content area displays a grid of 25 icons related to lobsters, each with a small preview image and its name below it. The icons include various styles of lobsters and shrimp.

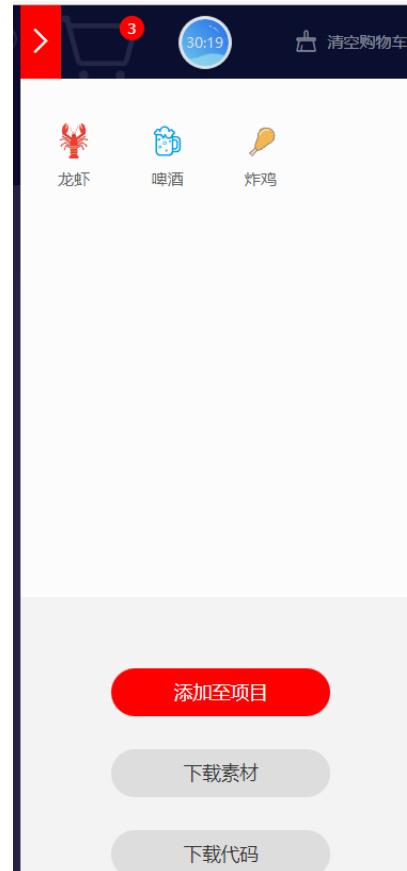
输入需要搜索的图标以后，点击搜索，这个时候就会出现很多与之相关的图标，然后选中你自己需要的图标



选中自己所需要图标，“添加入库”

The screenshot shows the iconfont.cn website interface again, this time searching for '炸鸡'. The search bar at the top contains the text '图标 炸鸡'. A red arrow points to the search input field with the text '你会在这里看到，所以添加的图标都会在这个地方我们点击这里' (You will see it here, so click here to add the icons). The main content area displays a grid of 7 icons related to fried chicken, each with a small preview image and its name below it. One specific icon of a fried chicken leg is highlighted with a dashed white border.

把所有需要的图标添加入库完毕以后，点击上图中的位置



第五步：添加选中的图标到项目当中



The screenshot shows the iconfont project management interface. On the left, there's a sidebar with sections for '我发起的项目' (H2204), '我参与的项目' (m_softteam, budejie), and '我删除的项目'. The main area displays a project named 'H2204' with 3 icons. The icons are: 1. 啤酒 (Beer) with Unicode  2. 龙虾 (Lobster) with Unicode  3. 炸鸡 (Fried Chicken) with Unicode . A red box highlights these three icons. At the top right, there are user stats (成员: 1), a help link ('使用帮助'), and a '点我看吧!' button.

当前的项目里面已经有三个图标了

截止上面的操作，我们所有的图标的准备工作已经完成，现在就开始使用了

iconfont图标的使用

iconfont图标的使用有3种方式

1. **unicode** 使用方法
2. **fontclass** 使用方法
3. **symbol** 使用方法 【基于JS使用的，后期再讲】

无论是使用哪一种方式，都要先把字体下载下来

The screenshot shows the same iconfont project management interface as before, but with a red arrow pointing to the '下载至本地' (Download to Local) button. This button is located in the toolbar above the icon list. The icons and their corresponding Unicode values are listed below: 啤酒 (), 龙虾 (), and 炸鸡 ().

> iconfont

名称	修改日期	类型	大小
demo.css	2022/7/25 14:37	层叠样式表文档	9 KB
demo_index.html	2022/7/25 14:37	Microsoft Edge ...	10 KB
iconfont.css	2022/7/25 14:37	层叠样式表文档	1 KB
iconfont.js	2022/7/25 14:37	JavaScript 源文件	21 KB
iconfont.json	2022/7/25 14:37	JSON 源文件	1 KB
iconfont.ttf	2022/7/25 14:37	TrueType 字体文件	5 KB
iconfont.woff	2022/7/25 14:37	WOFF 文件	4 KB
iconfont.woff2	2022/7/25 14:37	WOFF2 文件	3 KB

下载以后解压，会有如上图所示的文件，将整个文件夹复制到项目的根目录

第一步：将iconfont.css文件连接到网页当中

```
<link rel="stylesheet" href="iconfont/iconfont.css">
```

第二步：unicode方式使用

```
<style>
  .span1{
    font-size: 560px;
    color: red;
  }
</style>
<span class="iconfont span1">#xe662;</span>
```

因为iconfont本质上是把图标变成了字体，所以我们可以像字体一样去设置它的大小和颜色



优点：unicode的模型使用图标，它的兼容性高，也是矢量图标，使用方便

缺点：它只能设置纯色的图标

第三步：fontclass方式的使用

```
<style>
  .span2{
    font-size: 260px;
    color: deeppink;
  }
</style>
</span>
```



这种方式使用的图标也是矢量图标，它通过 `class` 的方式来实现

优点：

1. 相较于 `unicode` 的方式来说，它更直观一些，可以直接通过 `class` 的名称来快速定位图标
2. 它也是矢量图，放大不会失真
3. 它也是通过字体的方式来设置图标，所以可以通过 `font-size` 及 `color` 来设置大小及颜色

缺点：

1. 它只能设置纯色的图标
2. 内部是通过伪元素 `::before` 来实现的，所以对于低版本的浏览器不支持 `::before` 就不能使用

本地图标上传到iconfont

在进行界面开发的时候，如果UI设计师没有为我们准备图标，我们可能需要从网上查找，但是如果UI设计师已经帮我准备好了图标，我们就可以直接使用。只是在使用之前需要将这些图标转换成 `iconfont`



我发起的项目 H2204 3 个图标 2022-07-25 批量操作 回收站 项目设置 更多操作

H2204

movie-app-3

movie-app

easybuy-client

easybuy

华夏学院作业

华夏学院测试

Unicode

Font class

Symbol

下载至本地

上传图标至项目



啤酒



龙虾



炸鸡

点击按钮，将本地图标上传到iconfont的服务器

我参与的项目

m_softteam

budejie

我删除的项目

图标

矢量插画

3D 插画

Lottie

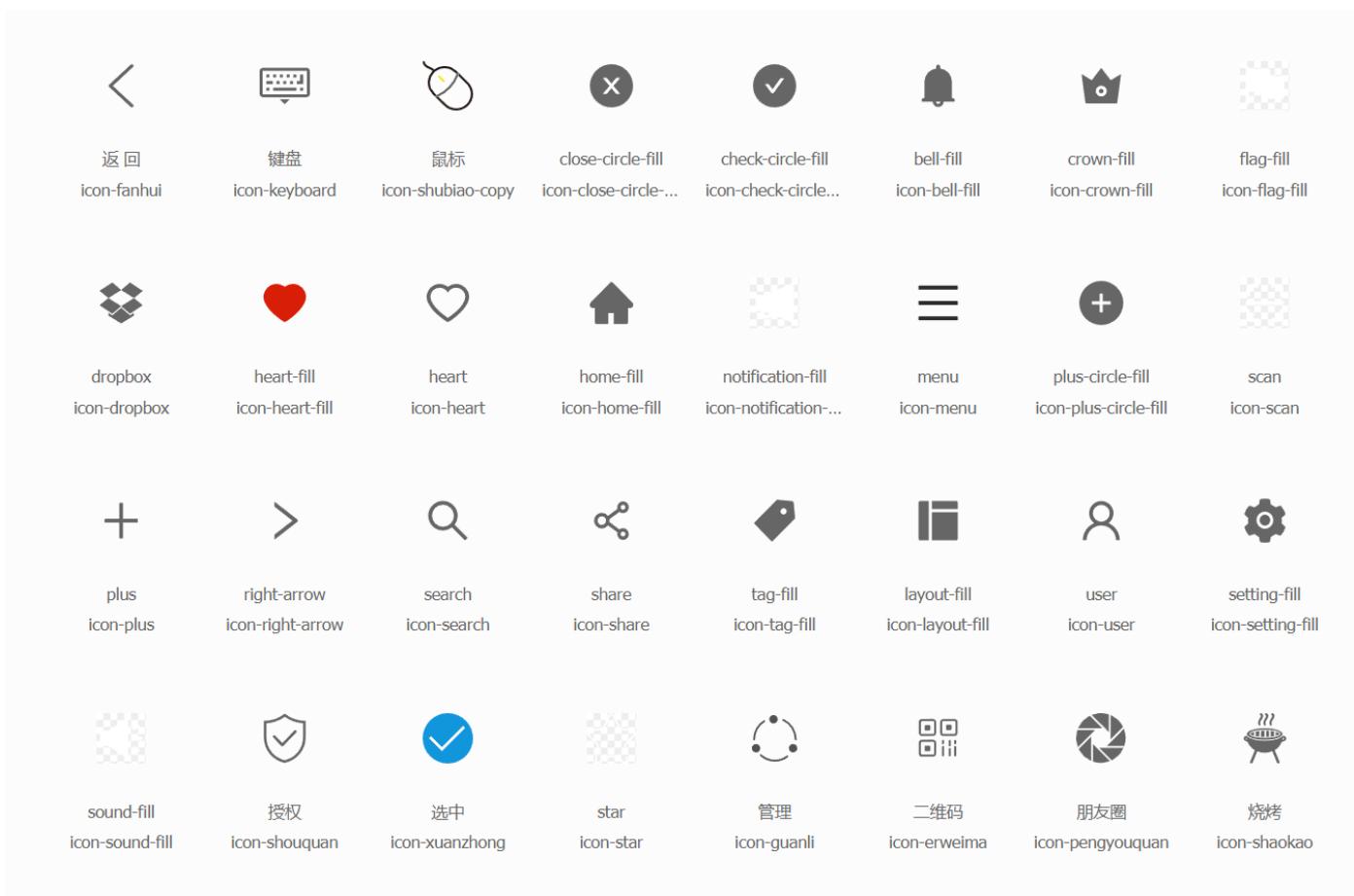
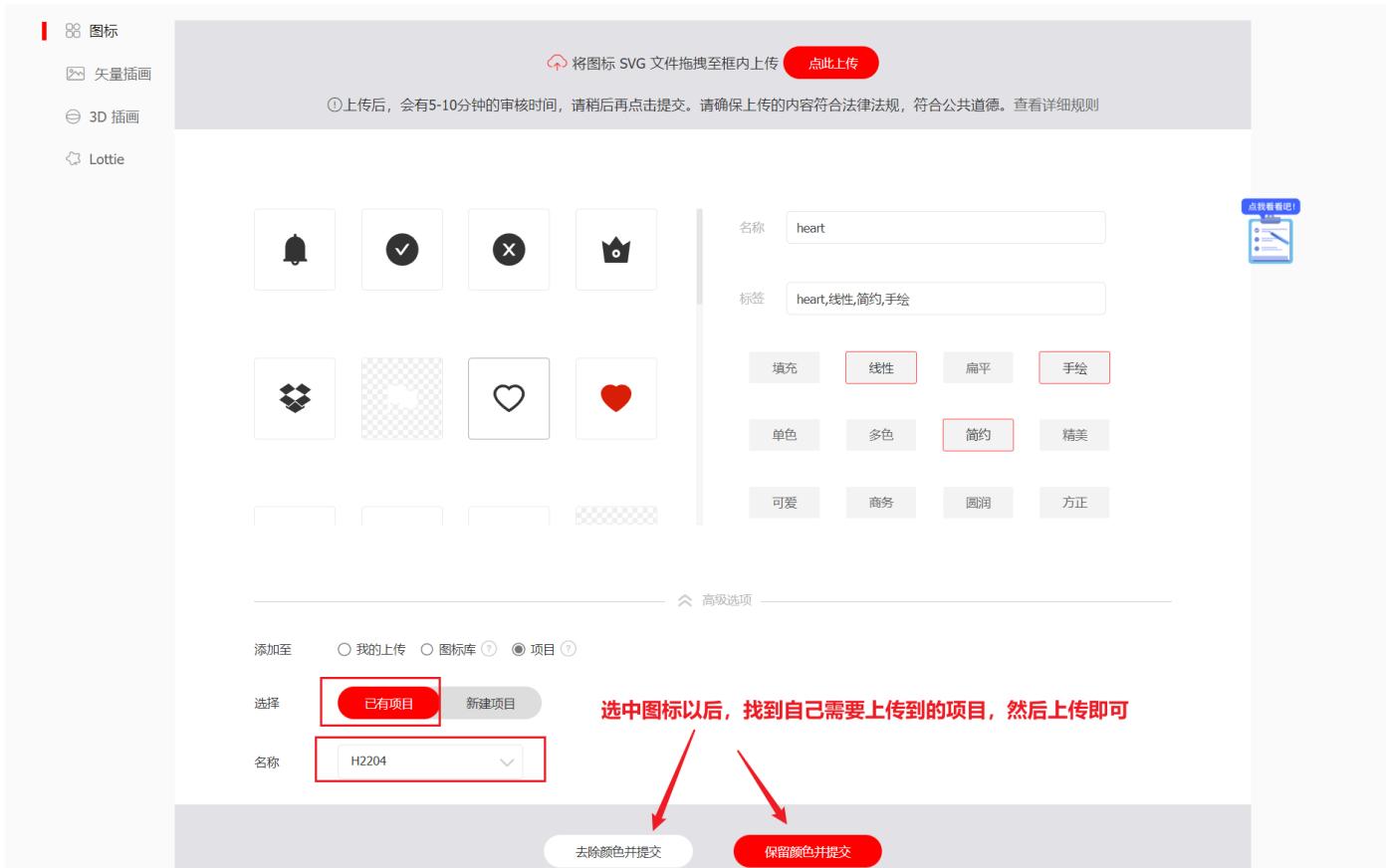


将图标 SVG 文件拖拽至框内上传

请您确保上传的内容符合法律法规，符合公共道德。查看详细规则

上传图标

点击以后，选中需要上传的图标文件



上传审核通过以后，所以的图标就都会显示出来，这个时候，只需要按照第4个操作过程，下载下来以后，使用就可以了。

移动端设计稿的计算

在之前我们在进行开发的时候，我们总是使用了 750 或 375 的设计稿，这一种标准的设计，在这种设计稿的模式下面，我们可以使用 pxcook 来直接计算



如果设计稿是 750px 的设计稿，我们可以直接使用这个 pxcook 这个工具来除以2



但是对于这种不是 750 或 375 的设计稿，我们应该怎么办呢？

现在我们来推一下公式



以上是1080的设计稿，我们现在想看一下 853 在 375px 的浏览器设备上面应该显示为多少？

设计稿

$$750 \quad = \quad 92$$

浏览器设备

$$375 \quad \times \quad 46$$

$$x = 92 * 375 / 750$$

结果 = 当前值 * 浏览器的宽度 / 设计稿的宽度

设计稿公式推断

根据上面推断过程，我们得到了一个点，如果想得到真实在浏览器当中的大小，应该是使用
设计稿上面量出来的大小 * 浏览器的宽度 / 设计稿的宽度

$$f(a) = a * \text{浏览器的宽度} / \text{设计稿的宽度}$$

a代表的就是在设计稿上面量出来的真实大小

现在我们深度使用我们的公式去书写，就会得到下面的代码

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>移动端设计稿的计算</title>
    <link rel="stylesheet" href="css/flex-box.css">
    <style>
        *{
            margin: 0;
            padding: 0;
            list-style-type: none;
        }
        #app{
            width: 100vw;
```

```

        height: 100vh;
    }
    :root{
        --primaryColor:#ffc81f;
    }
    .tab-bar{
        background-color: var(--primaryColor);
        height: calc(92px * 375 / 750);
    }
    .head-pic{
        width: calc(100px * 375 / 750);
        height: calc(100px * 375 / 750);
    }

```

</style>

</head>

<body>

```

<div id="app" class="flex-col">
    <div class="tab-bar">下</div>
    <div class="content-box flex-1">
        <div class="user-box">
            
        </div>
    </div>
</div>

```

</body>

</html>

我们发现在设计元素大小的时候非常不方便，最重要的就是下面的部分的代码

```
height: calc(92px * 375 / 750);
```

在这样的代码里面，我们又发现只需要将设计稿上面得到的大小乘以一个比例

思考：有没有什么快速的方案去相乘呢？

能不能有这么一个方法，我只写一个 92 代表的就是 $92px * (375 / 750)$

思路：在之前的时候，我们学过一个单位叫 `rem`，`rem` 的全称指 `html` 元素也就是 `root element`，它代表 `html` 标签的 `font-size` 的大小，默认的 `rem` 大小是 `16px`

```
1rem = 1 * 16px
```

突破：如果我要是把 `html` 标签的 `font-size` 改成了 `0.5px`

```
html{
    font-size:0.5px;
}
1rem = 1 * 0.5px;
92rem = 92 * 0.5px;
```

结论

```
*{
    margin: 0;
    padding: 0;
    list-style-type: none;
}

#app{
    width: 100vw;
    height: 100vh;
}

/* html rem */
:root{
    --primaryColor:#ffc81f;
    font-size: calc(375px / 750);
}

.tab-bar{
    background-color: var(--primaryColor);
    height: 92rem;
}

.head-pic{
    width: 100rem;
    height: 100rem;
}
```

我们使用了 `rem` 这种单位，这种单位快速的帮我们计算了一个结果值

问题一

经过上面的使用，我们已经掌握了设计计算的基本规律，但是仍然有些细节需要注意

```
.login-box{  
    width: 853rem;  
    height: 557rem;  
    background-color: pink;  
    margin: 612rem auto 0;  
}
```

我们在使用 **rem**，我们发现这个值非常大，这是因为我们在计算rem的时候我们给了一个公式

当前设备宽度 / 设计稿宽度

在计算的时候有可能会出现除不尽，为了提高精度，我们一般会乘以一个经较大的基准值

$f(a) = a * \text{当前设备宽度} / \text{设计稿宽度} * \text{基准值}$

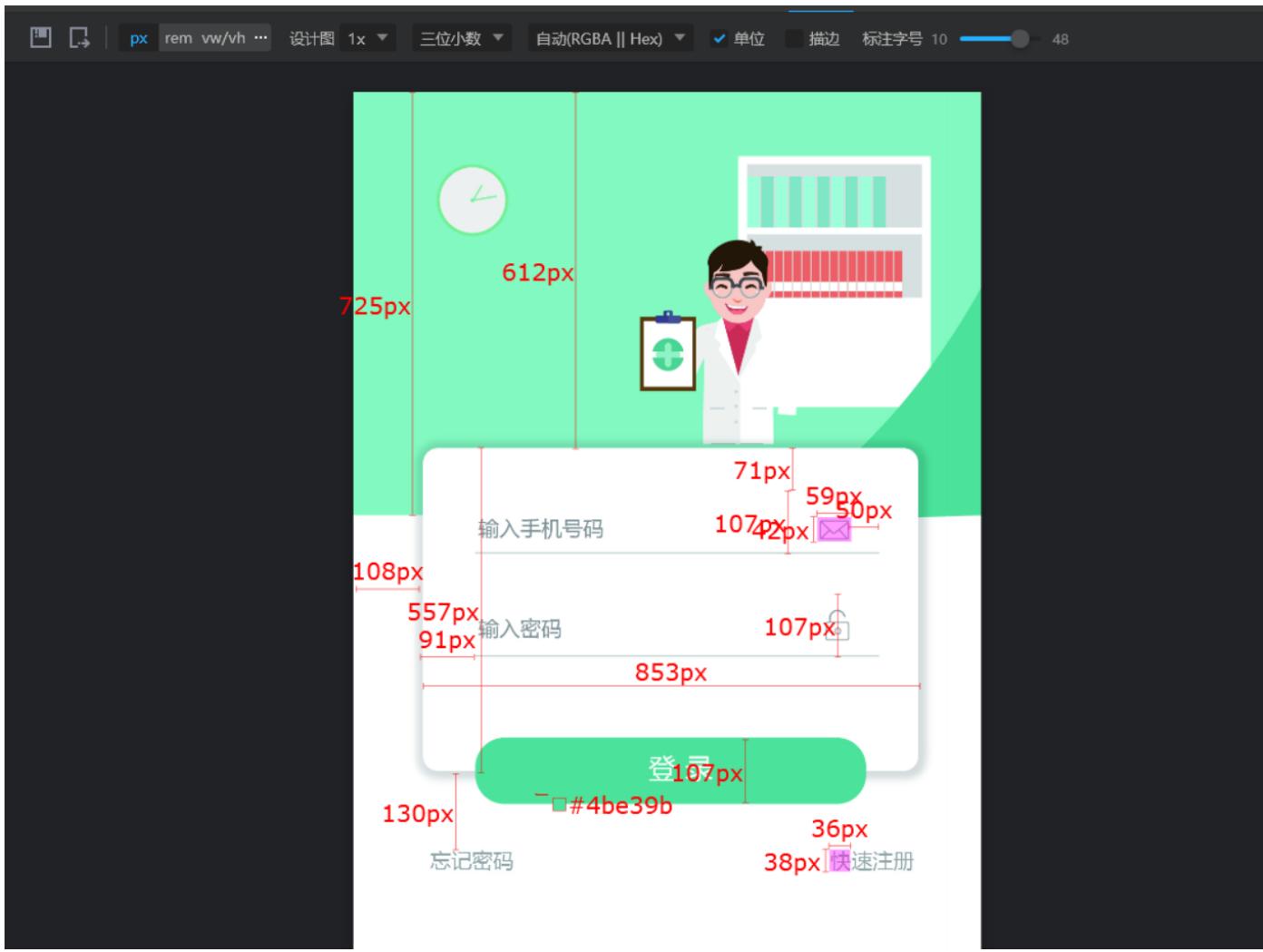
这个基准值可以随便给

如当前设计稿为1080，设备宽度不375px，则rem的计算应该是这样的

```
:root{  
    font-size:calc(375px / 1080 * 100)  
}
```

这个时候我们给的基准值就是100，但是这个100必不是固定的，每个公司或每个项目团队给的值都不一样的

==问题==：我们在设计稿上面不能够快速的知道最后的结果是多少



解决方法一：直接使用pxcook提供的功能



解决方法二：使用HbuilderX开发工具里面的转换



```
.login-box{  
    width: 852px;  
}  
/style>  
d>
```

1 852px
2 852px->8.52rem

这个时候在使用的时候会自动帮我们把输入的 **px** 转换成所需要用到的 **rem**

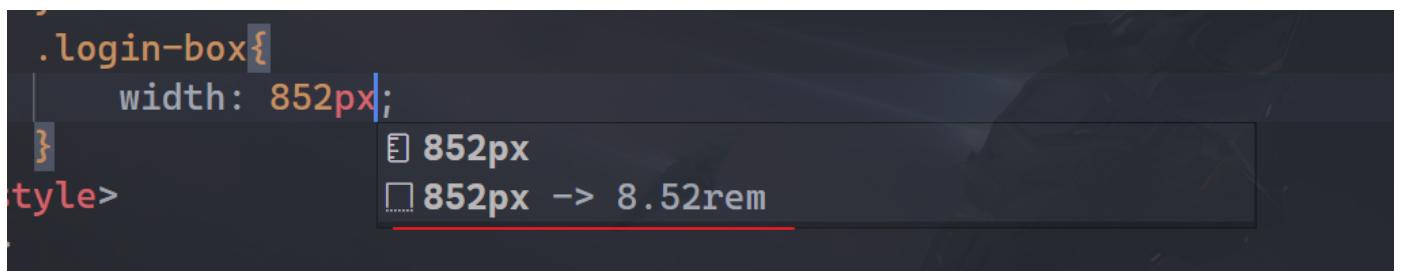
解决方法三：使用VSCode开发工具里面的插件转换



安装完成插件以后，我们要设置基准值



设置完了以后，一定要重新启动 **VSCode**



在使用的时候它也会多一个选项，帮我们转换成 **rem**

问题二

我们现在改变了 **rem** 默认的大小

```
:root{  
  font-size: calc(375px / 1080 * 100); /*34.72px*/  
}
```

这个时候网页的默认字体就变了，不再是原来的16px。所以我们现在要恢复网页的默认字体大小

```
body {  
    font-size: 16px;  
}
```

问题三

我们之前在推断公式的时候，我们把设备的宽度固定成了 **375px**，但是在实际的使用场景下面，我们设备的大小是不固定的

```
:root {  
    font-size: calc(375px / 750 * 100);  
}
```

在上面的代码里面，我们是假设所有的手机都是 **375px**

思考：为什么写 **375px**？

因为 **iphone 6/7/8** 这三个设备的宽度就是 **375px**

思路：有什么东西还可以代表设备的宽度？

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

我们之前在写代码的时候，总是会添加视口，其中 **width=device-width** 代表视口的宽度与设备的宽度相等了

突破：视口是 **viewport**，它会有一个单位叫 **vw**，而 **100vw** 代表的就是视口的宽度，也就是设备的宽度

结论：通过上面的推断与分析，我们的公式可以列举如下

```
:root {  
    font-size: calc(100vw / 750 * 100);  
}
```

问题四



我们用 `rem` 写了一个网页，但是这个网页如果是在PC浏览器上面打开了，会出现很大的问题，如何控制呢

解决方案：使用 `@media` 媒体查询

响应式布局中的媒体查询

什么是响应式布局

响应式布局指的是页面的样式或布局会随着设备的大小，类型及状态的改变而主动的做出改变

场景一：一个页面在手机端的竖屏状态下是一个样式，在横屏状态下又是一个状态

场景二：一个页面在手机端，平板，PC端所展现出来的样式或不一样，这也算是响应式

现在我们先通过一个简短的案例来完成一下

需求：我们现在需要一个盒子，在 `0px~767px` 这个范围之前表现出背景为红色，然后大于 `767px` 则背景变成蓝色

```
<!DOCTYPE html>
<html lang="zh">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>响应式入门</title>
    <style>
        .box{
            width: 200px;
            height: 200px;
            border: 2px solid black;
            background-color: blue;
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
</html>
```

上面的代码只能保证当前的盒子变成蓝色，如果让盒子在小于767以后改变，这个时候就要使用响应式

媒体查询

响应式的核心是一个叫媒体查询的东西，媒体查询是CSS的一个命令 `@media` ,通过这个媒体查询，我们可以根据设备的多媒体状态

```
@media [only/not] 设备类型 [and 附加条件] {
    /*CSS代码*/
}
```

常用媒体查询

```

@media print {
    /*打印设备*/
}

@media screen {
    /*屏幕显示*/
}

@media all {
    /*所有多媒体设备*/
}

```

现在我们来试一下第一种：查询到网页在打印设备上面的时候

姓名	性别	爱好
张三1	男	看书睡觉
张三2	男	看书睡觉
张三3	男	看书睡觉
张三4	男	看书睡觉
张三5	男	看书睡觉
张三1	男	看书睡觉
张三2	男	看书睡觉

现在有上面的一个网页，网页上面有一个 `table1`，我们希望在打印这个网页时候，表格里面所有的文字都居中对齐，怎么办呢

```

.table1 {
    text-align:center;
}

```

上面的代码是有效果的，而这个效果的体现形式应该是在打印设备上面，所以代码如下

```

/* 媒体查询打印设备，仅仅只是在打印设备上面生效 */
@media only print {
    .table1 {
        text-align: center;
    }
}

```

第二种：查询屏幕多媒体设备

针对上一个章节抛出来的那个疑问 我们现在需要一个盒子，在 0px~767px 这个范围之前表现出背景为红色，然后大于 767px 则背景变成蓝色

```
/* 0~767px 变成红色 */
@media only screen and (min-width:0px) and (max-width:767px) {
    .box{
        background-color: red;
    }
}
```

1. `min-width` 代表最小宽度
2. `max-width` 代表最大宽度

一个设备的最小宽度就是0，所以上面的代码可以简写成如下

```
@media only screen and (max-width:768px) {
    .box {
        background-color: red;
    }
}
```

我们现在在做一次扩展

? 假设，我们现在希望在 `300px~767px` 之间变成红色，其它的时候变成蓝色，怎么办呢

```
@media only screen and (min-width:300px) and (max-width:767px) {
    .box {
        background-color: red;
    }
}
```

同时，我们还可以将屏幕进行连续的划分

```
/*
0~767px 红色
768px ~ 991px 蓝色
991px ~ 1200px deeppink分色
1200px以上为gray灰色
*/
@media only screen and (max-width:768px) {
    .box {
        background-color: red;
    }
}
@media only screen and (min-width:768px) and (max-width:991px){
```

```
.box{
    background-color: blue;
}

}

@media only screen and (min-width:991px) and (max-width:1201px){
.box{
    background-color: deeppink;
}
}

@media only screen and (min-width:1201px) {
.box{
    background-color: gray;
}
}
```

现在我们来完成另一个案例，通过4张图片来完成图片的响应式切换

```
<!DOCTYPE html>
<html lang="zh">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>图片响应式</title>
<style>
.box{
    border: 2px solid black;
    display: flex;
    /* 弹性换行 */
    flex-wrap: wrap;
}
@media only screen and (max-width:768px){
/* 一排显示1张 */
.box>img{
    width: 100%;
}
}
@media only screen and (min-width:768px) and (max-width:991px){
/* 一排显示2张 */
.box>img{
    width: 50%;
}
}
```

```

@media only screen and (min-width:991px){
    /* 一排显示4张 */
    .box>img{
        width: 25%;
    }
}

/*
    0~767px 一排显示1张图片
    768px ~ 991px 一排显示2张图片
    991px 以上 一排显示4张图片
*/

```

</style>

</head>

<body>

<div class="box">

</div>

</body>

</html>

在上面的代码里面，我们可以根据不同的代码需求，来实现图片每次显示的数量

屏幕尺寸的划分

在上面做屏幕的响应式的时候我们发现了一个点，我们总是对屏幕进行了几个尺寸的划分，屏幕尺寸的划分也有一个规范，现在大多数的UI框架所使用的都是 **bootstrap** 的屏幕划分规范

bootstrap 框架把屏幕划分成了4个等级

尺寸范围	说明
0~767px	手机
768px~991px	平板
992px~1200px	PC
1201px及以上	大型PC

媒体查询使用注意事项

在使用媒体查询的语法的时候，有一些注意事项需要注意

1. 严格遵守媒体查询的语法，单词与单词之间一定要加上空格
2. 媒体查询的代码应该在CSS代码的最后面加载

```
@media only screen and (max-width:768px) {  
    .box{  
        background-color: red;  
    }  
}  
.box{  
    width: 100px;  
    height: 100px;  
    border: 2px solid black;  
    background-color: blue;  
}
```

上面的写法就是错的

3. 媒体查询代码里面选择器的权限只能大于或等于外边的普通样式代码，否则就覆盖不了

```
#div1{  
    width: 100px;  
    height: 100px;  
    border: 2px solid black;  
    background-color: blue;  
}  
  
@media only screen and (max-width:768px) {  
    .box{  
        background-color: red;  
    }  
}
```

扩展

设备的横竖状态改变

在手机端里面，因为手机端是具备重力感应的，我们经常会把手机横着或竖着，对于手机横着或竖着，也是可以通过媒体查询来获取信息的

```
@media only screen and (orientation:portrait){
    /* 竖向的 */
    .box{
        background-color: red;
    }
}

@media only screen and (orientation:landscape){
    /* 横向的 */
    .box{
        background-color: blue;
    }
}
```

使用link来链接响应式CSS

```
<link rel="stylesheet" href="css/01.css" media="screen and (max-width:768px)">
<link rel="stylesheet" href="css/02.css" media="screen and (min-width:768px)">
```

案例



预处理脚本

虽然可以直接使用 Bootstrap 提供的 CSS 样式表，不要忘记 Bootstrap 的源码是基于最流行的 CSS 预处理脚本 - [Less](#) 和 [Sass](#) 开发的。你可以采用预编译的 CSS 文件快速开发，也可以从源码定制自己需要的样式。



一个框架、多种设备

你的网站和应用能在 Bootstrap 的帮助下通过同一份代码快速、有效适配手机、平板、PC 设备，这一切都是 CSS 媒体查询 (Media Query) 的功劳。



特性齐全

Bootstrap 提供了全面、美观的文档。你能在这里找到关于 HTML 元素、HTML 和 CSS 组件、jQuery 插件方面的所有详细文档。

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

<head>
    <meta charset="UTF-8">
```

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>响应式练习</title>
<style>
    * {
        margin: 0;
        padding: 0;
        list-style-type: none;
    }

    /*
        0~767px: 手机端
        768px~991px: 平板端
        992px~1200px: PC端
        1200px~: 大屏幕
    */
    .container {
        width: 1200px;
        border: 1px solid red;
        min-height: 50px;
        margin: auto;
    }
    .box{
        display: flex;
    }
    .box>.item{
        width: 33%;
        box-sizing: border-box;
        padding: 0 10px;
    }
    .box>.item>img{
        width: 100%;
    }
    @media only screen and (max-width: 768px) {
        .container{
            width: 100%;
        }
        /*让盒子里面的元素换行*/
        .box{
            flex-wrap: wrap;
        }
        /*调整盒子里的宽度，每排显示一个*/
    }

```

```
.box>.item{
    width: 100%;
}

}

@media only screen and (min-width:768px) and (max-width:992px) {
    .container{
        width: 768px;
    }
}

@media only screen and (min-width:992px) and (max-width:1200px) {
    .container{
        width: 992px;
    }
}

@media only screen and (min-width:1200px) {
    .container{
        width: 1200px;
    }
}

}

</style>
</head>

<body>
    <div class="container box">
        <div class="item">
            
            <h3>预处理工具</h3>
            <p>虽然可以直接使用 Bootstrap 提供的 CSS 样式表，但是不要忘记，Bootstrap 的源码是采用最流行的 CSS 预处理工具 <a href="/css/#less">Less</a> 和 <a href="/css/#sass">Sass</a> 开发的。你可以直接采用预编译的 CSS 文件快速开发，也可以从 Bootstrap 源码自定义自己需要的样式。</p>
        </div>
        <div class="item">
            
            <h3>一个框架、多种设备。</h3>
            <p>你的网站和应用能在 Bootstrap 的帮助下通过同一份源码快速、有效地适配手机、平板和 PC 设备，这一切都是 CSS 媒体查询 (Media Query) 的功劳。</p>
        </div>
    </div>

```

```
<div class="item">
    
    <h3>功能完备</h3>
    <p>Bootstrap 提供了全面、美观的文档，你能在这里找到关于普通 HTML 元素、HTML 和 CSS 组件以及 jQuery 插件方面的所有详细文档。</p>
</div>
</div>
</body>

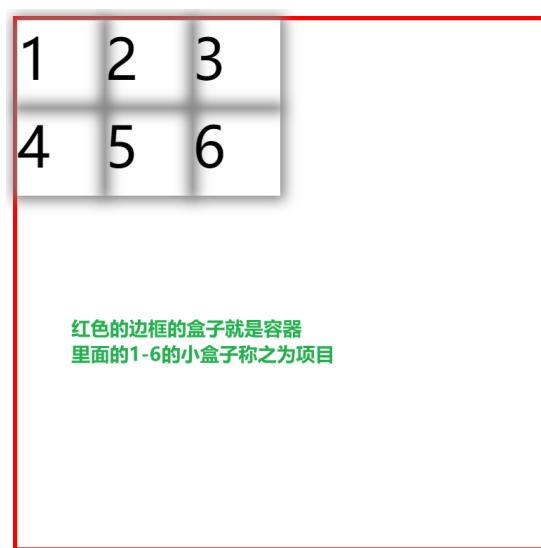
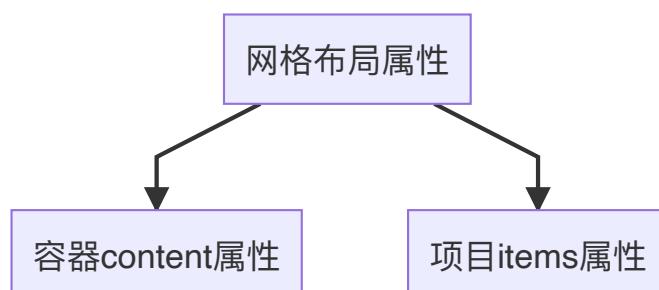
</html>
```

这个题目的思路其实就是让外边的容器实现逐次递减的过程

CSS中的grid布局

grid布局是一种新兴的布局方式，它是在弹性盒子的布局上面去实现二维布局方式（存在行与列的关系）

网格布局里面有行 **row** 与列 **column** 的概念的，并且网格布局的属性也分为2大类



容器属性

容器就是用于放东西的盒子，容器属性指的就是这个盒子上面的属性

1. `display:grid` 将容器变成网格
2. `grid-template-columns` 定义列的模板

```
grid-template-columns: 100px 200px 300px ; /*定义固定的列宽*/  
grid-template-columns: 100px auto 100px; /*auto可以自动设置大小*/  
grid-template-columns: 33.3% 33.3% 33.4%; /*也可以设置百分比*/  
grid-template-columns: 1fr 1fr 1fr; /*fr是片段、份 的意思*/  
grid-template-columns: 100px 100px minmax(100px, 200px); /*minmax(设置  
最小值和最大值区间)*/  
grid-template-columns: repeat(3,100px);  
grid-template-columns: repeat(auto-fill,200px); /*根据某一个大小自动去填充  
*/;
```

3. `grid-template-rows` 定义行的模板，它的定义方式与列的定义方式是一样的
4. `grid-row-gap` 网格中行与行的间距
5. `grid-column-gap` 网格中列与列的间距
6. `grid-gap` 单元格的间距，是上面2个属性的简写，第一个代表行间距，第二个默认不写与第一个值相同，写了就代表列间距
7. `justify-content` 让项目在容器里面左右居中，属性值与弹性盒子一致
8. `align-content` 让项目在容器里面上下居中，属性值与弹性盒子一致
9. `place-content` 是 `align-content` 与 `justify-content` 的简写

```
place-content: center flex-end;  
/*相当于*/  
align-content:center;  
justify-content:flex-end;
```

10. `grid-template-areas` 对网格中每个区域定义一个名子

```
grid-template-areas: 'a b c'  
                      'd e f';
```

定义了一个名子以后，后期可以快速的让一个项目元素去指定的网格中

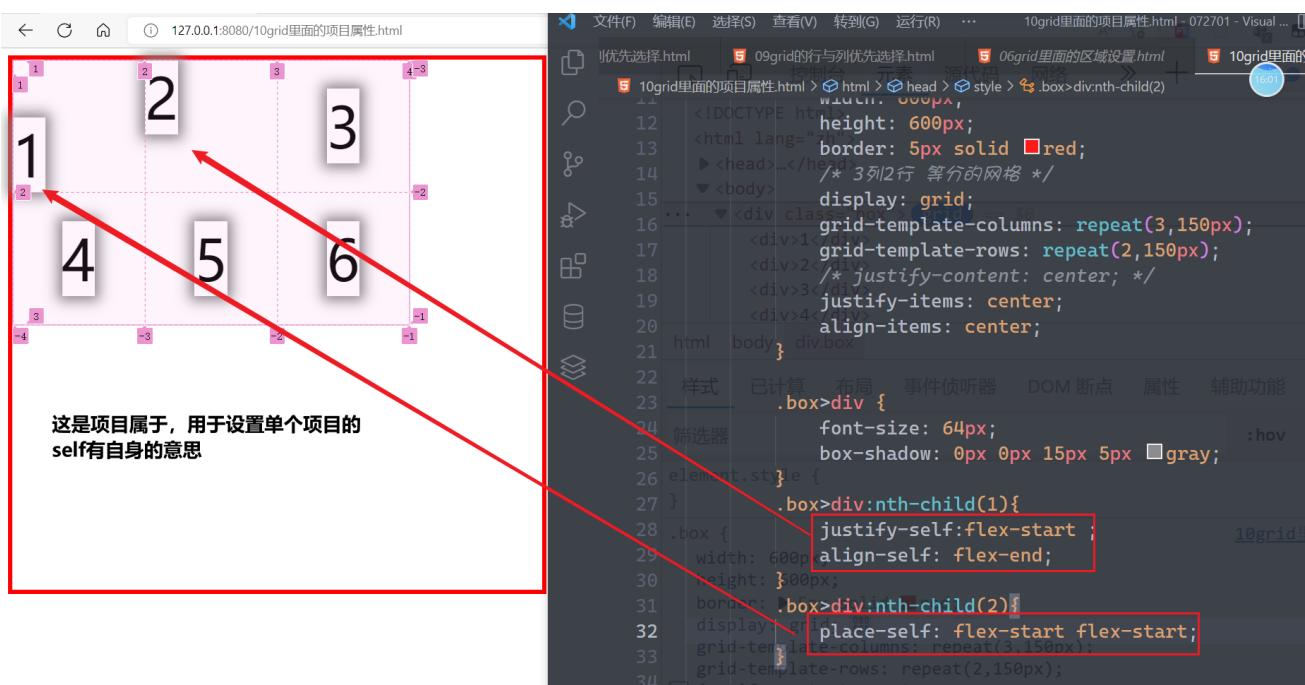
11. `grid-auto-rows` 设置自动添加的行的高度

12. `grid-auto-columns` 设置自动添加的列的宽度
13. `grid-auto-flow` 设置网格中的元素是先从行开始还是先从列开始
 - `row` 先从行开始
 - `column` 先从列开始
 - `row dense` 从行开始，稠密的排列【只有有位置，我就插入进去】
 - `column dense` 从列开始，稠密的排列【只有有位置，我就插入进去】
14. `justify-items` 设置在容器上面，让容器里面的每一个项目都在小格子里面左右居中
15. `align-items` 设置在容器上面，让容器里面的每一个项目都在小格子里面上下居中

项目属性

网格当中的每一个元素都称之为项目，当容器把自己拆分成多个网格以后

1. `justify-self` 用于设置项目自身的左右位置
2. `align-self` 用于设置项目自身的上下位置
3. `place-self` 用于同时设置项目的左右，上下位置



4. `grid-column-start` 某一个项目元素列开始的线条位置
5. `grid-column-end` 某一个项目元素列结束的线条位置
6. `grid-column` 同时设置结束与开始的线条位置

```
grid-column: 1 / 3;  
/*span有合并和跨度的意思*/  
grid-column: 1 / span 2;
```

7. **grid-row-start** 某一个项目元素行开始的线条位置
8. **grid-row-end** 某一个项目元素行结束的线条位置
9. **grid-row** 同时设置结束与开始的线条位置

```
grid-row: 1 / 3;  
/*这与上面的列是一样的，跨2个*/  
grid-row: 1 / span 2;
```

10. **grid-area** 让一个项目元素去指定的区域



注意

弹性盒子里面具备的特殊在网格元素里面同样具备

案例

1. 实现下面布局



2. 实现下面布局

1	2	3	C
4	5	6	
7	8	9	-
0			.

3. 实现下面布局

点击开始以后，转盘开始抽奖

一等奖	二等奖	三等奖	作业
奶茶	开始	大笑一个 祝你好运	
晚自习			
热干面	笑一个	晚自习	祝你好运

4. 实现下图的布局

