

html

权重

可以把样式的应用方式分为几个等级，按照等级来计算权重

1. !important 权重值为10000
 - .. 内联样式 权重值为1000
 - .. ID选择器 权重值为100
 - .. 类，伪类和属性选择器 权重值为10
 - .. 标签选择器和伪元素选择器 权重值为1
2. 其他权重为0

文件英文名

1. 文件夹: index.html 首页 favicon.ico ico图标 image 固定使用的图片素材 uploads 非固定使用图片、
 - . css 文件: base.css 基础公共样式 common 多个页面相同模块 index.css 首页css文件
 - . 页面结构: 容器: container 页头: header 内容: content/container 页面主体: main 尾: footer 导航: nav 侧栏: sidebar 栏目: column
 - 页面外围控制整体布局宽度: wrapper 左右中: left right center 上下: prev next
 - 菜单: menu 子导航: subnav 头部导航: topnav 快捷导航: shortcut
 - 横幅: banner

SEO三大优化

```
1      <!-- meta:desc 网页描述标签 -->
2      <meta name="description" content="京东JD.COM-专业的综合网上购物商城，为您提供正品低价的购物选择、优质便捷的服务体验。商品来自全球数十万品牌商家，囊括家电、手机、电脑、服装、居家、母婴、美妆、个护、食品、生鲜等丰富品类，满足各种购物需求。">
3      <!-- meta:kw 网页关键词标签 -->
4      <meta name="keywords" content="网上购物,网上商城,家电,手机,电脑,服装,居家,母婴,美妆,个护,食品,生鲜,京东">
5      <!-- 网页标题标签 -->
6      <title>京东(JD.COM)-正品低价、品质保障、配送及时、轻松购物! </title>
7      <!-- link:favicon : 浏览器标题栏图标 -->
8      <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
```

HTML骨架

```
1      <!-- 声明html -->
```

```

2 <!DOCTYPE html>
3 <!-- 中文网站 zh-CN 简体中文 / en 英文 搜索引擎归类 + 浏览器翻译 -->
4 <html lang="zh-CN">
5 <head>
6     <!-- charset="UTF-8" 规定网页的字符编码
7         1. UTF-8: 万国码, 国际化的字符编码, 收录了全球语言的文字
8         2. GB2312: 6000+ 汉字
9         3. GBK: 20000+ 汉字 -->
10    <meta charset="UTF-8">
11    <!-- ie(兼容性差) / edge -->
12    <meta http-equiv="X-UA-Compatible" content="IE=edge">
13    <!-- 宽度 = 设备宽度 : 移动端网页的时候要用 -->
14    <meta name="viewport" content="width=device-width, initial-scale=1.0">
15    <!-- 还有SEO三大标签 -->
16    <!-- 网页标题 -->
17    <title>Document</title>
18 </head>
19 <body></body>
20 </html>

```

精灵图

使用精灵图的步骤是什么?

1. 创建一个盒子
2. 设置盒子大小为小图片大小
3. 设置精灵图为盒子的背景图片 background-image:精灵图地址
4. 将小图片左上角坐标 取负值, 设置给盒子的background-position: x y;

建议遵循以下顺序:

1. 布局定位属性: display / position / float / clear / visibility / overflow (建议 display 第一个写, 毕竟关系到模式)
2. 自身属性: width / height / margin / padding / border / background
3. 文本属性: color / font / text-decoration / text-align / vertical-align / white-space / break-word
4. 其他属性 (CSS3) : content / cursor / border-radius / box-shadow / text-shadow / background:linear-gradient

清楚浮动:

1. 直接设置父元素高度
2. 额外标签法 在父元素内容的最后添加一个块级元素

```

1 .clearfix {
2     /* 清除左右两侧浮动的影响 */
3     clear: both;
4 }

```

3. 单伪元素清除法

```

1  /* 单伪元素清除浮动 和 额外标签法原理是一样的 */
2      .clearfix::after {
3          content: '';
4
5          /* 伪元素添加的标签是行内，要求块 */
6          display: block;
7          clear: both;
8
9          /* 为了兼容性 */
10         height: 0;
11         visibility: hidden;
12     }

```

4.双伪元素清楚法

```

1      /* 清除浮动 */
2      .clearfix::before,
3      .clearfix::after {
4          content: '';
5          display: table;
6      }
7      /* 真正清除浮动的标签 */
8      .clearfix::after {
9          /* content: '';
10         display: table; */
11         clear: both;
12     }

```

5.给父元素设置 overflow : hidden

字体图标

icomoon 字库 <http://icomoon.io>

阿里 iconfont 字库 <http://www.iconfont.cn/>

1. 将字体图标的font文件夹放入与html同一个文件 将css 中 @font-face复制到html中的style里
2. 使用标签改字体

```

1  span{
2      font-family: '字体图标名字';
3  }
4  <span>□</span>

```

3.使用 :: before/after 伪元素 添加

```
1  div::after {
2      font-family: '字体图标名字';
3      content: '□';    ①
4      content: '\e91e'; ②
5  }
```

什么网页

网页是图片、链接、文字、声音、视频等元素组成, 其实就是一个html文件(后缀名为html)

网页生成

网页生成制作: 有前端人员书写 HTML 文件, 然后浏览器打开, 就能看到了网页.

HTML是什么

HTML: 超文本标记语言, 用来制作网页的一门语言. 有标签组成的. 比如 图片标签 链接标签 视频标签等...

什么是WEB标准

Web 标准是由 W3C 组织和其他标准化组织制定的一系列标准的集合。 W3C（万维网联盟）是国际最著名的标准化组织

WEB结构

结构写到 HTML 文件中, 表现写到 CSS 文件中, 行为写到 JavaScript 文件中。

基本语法概述

1. HTML 标签是由尖括号包围的关键词, 例如 < html>。
2. HTML 标签通常是成对出现的, 例如 < html> 和 < /html>, 我们称为双标签。标签对中的第一个标签是 开始标签, 第二个标签是结束标签。
3. 有些特殊的标签必须是单个标签 (极少情况), 例如 < br />, 我们称为单标签。

常用标签

标题标签 < h1> - < h6> (重要)

段落 < p> < /p> 换行标签 < br />、水平线 < hr>

文本格式化标签 后者语义强 b strong 加粗 双标记 i em 倾斜 双标 u ins 下划线 双标 s del 删除线 双标

图片标签

- < img src="图片路径" alt="替换文本" title="提示文本" width="宽度" height="高度" border="边框">
align属性 top 上 middle 中 bottom 下 left 左 right 右 **相对路径: 以引用文件所在位置为参考基础, 而建立出的目录路径。** **绝对路径: 是指目录下的绝对位置, 直接到达目标位置, 通常是从盘符开始的路径。** 同级目录: 直接写: 目标文件名字! 下级目录: 直接写: 文件夹名/目标文件名字! 上级目录: 直接下: ../目标文件名字!

超链接: `< a href="跳转目标" target="目标窗口的弹出方式"> 文本或图像 < /a>` `_self`默认 `_blank`为新窗口打开

链接分类:

- 外部链接: 例如 `< a href="http:// www.baidu.com "> 百度 < /a>`。
- 内部链接: 网站内部页面之间的相互链接. 直接链接内部页面名称即可, 例如 `< a href="index.html"> 首页 < /a>`。
- 空链接: 如果当时没有确定链接目标时, `< a href="#"> 首页 < /a>`。
- 下载链接: 如果 href 里面地址是一个文件或者压缩包, 会下载这个文件。
- 网页元素链接: 在网页中的各种网页元素, 如文本、图像、表格、音频、视频等都可以添加超链接。
- 锚点链接: 点我们点击链接, 可以快速定位到页面中的某个位置。
- 超链接标签 (重点)
- 在链接文本的 href 属性中, 设置属性值为 #名字 的形式, 如 `< a href="#two"> 第2集 < /a>`)
- 找到目标位置标签, 里面添加一个 id 属性 = 刚才的名字 , 如: `< h3 id="two">第2集介绍 < /h3>`
- HTML中的注释以“`<!--`”开头, 以“`-->`”结束。

特殊字符: 空格 ; 小于号< ; 大于号> ; 版权© ; 上标⊃ ; 下标⊂ ; 度°

音频标签

- src: 地址 controls: 控制播放控件 autoplay: 自动播放(部分浏览器不支持) loop: 循环播放 source可以让浏览器自己选择播放的格式 mp3和ogg

```
1 < audio controls="controls">
2
3   < source src="音频地址" type="audio/mp3">
4
5   < source src="音频地址" type="audio/ogg">
6
7   -你的浏览器不支持audio
8
9 < /audio>
10
```

视频标签

- src: 地址 controls: 控制播放控件 autoplay: 自动播放(谷歌需要配合muted实现静音播放) loop: 循环播放 poster 显示图片
- source可以让浏览器自己选择播放的格式 mp4和ogg

```
1 < video controls="controls">
2
3 < source src="音频地址.ogg" type="audio/ogg">
4
5 < source src="音频地址.mp4" type="video/mp4">
6
7 你的浏览器不支持video
8
9 < /video>
10
```

列表

无序列表

- 无序列表由几个标签组成？分别表示什么？
- ul标签：表示无序列表的整体
- li标签：表示无序列表的每一项
- 无序列表标签的嵌套规范是什么？
- type属性：circle 圆 | square 方 | disc点（默认）
- ul标签中只允许嵌套li标签
- li标签中可以嵌套任意内容

有序列表

- 有序列表由几个标签组成？分别表示什么？
- type=排列方式 | start 默认从几开始 | reversed 倒序
- ol标签：表示有序列表的整体
- li标签：表示有序列表的每一项
- 有序列表标签的嵌套规范是什么？
- ol标签中只允许嵌套li标签
- li标签中可以嵌套任意内容

自定义列表（dl）

结构：dl>dt（名词）>dd（解释名词）用于内容解释

表格标签

- 完成一个简单的表格，需要由几个标签组成？分别表示什么？
- table标签：表格整体
- tr标签：表格每行
- td标签：对于主题的每一项内容
- 表格基本标签的嵌套规范是什么？
- table > tr > td

表格属性

- border属性：表格边框
- width属性：表格整体的宽度
- height属性：表格整体的高度
- 表格整体大标题：caption标签 • 书写在table标签内部
- 表头单元格：th标签 • 书写在tr标签内部（用于替换td标签）
- 表格的结构标签分别有哪些？表示什么含义？
 - thead：表格头部
 - tbody：表格主体
 - tfoot：表格底部
- 表格结构标签书写在什么位置？
 - 表格结构标签写在table标签内部
 - 表格标签内部用于包裹tr标签

合并单元格

- rowspan：跨行合并→垂直方向合并
- colspan：跨列合并→水平方向合并
- cellpadding 单元格与单元格边框的边距
- cellpadding 单元格内容与单元格边框的边距

表单 form

表单域：action服务器地址 method发送表单数据时使用的网络请求方法：get/post name 用来区分多个表单

input标签属性

- text:文本框;password: 密码; radio: 单选框; checkbox: 多选框;
- file:文件;submit:提交按钮;reset:重置按钮 button: 普通按钮

html5表单新增

- type=email 邮箱 url 地址 date 日期 time 时间 month 月 week 周 number 数字 tel手机号 search搜索框 color颜色 autofocus 自动获取焦点 autocomplete 显示之前搜索的值 默认off /打开on

```

1  <!-- 我们验证的时候必须添加form表单域 -->
2  <form action="">
3      <ul>
4          <li>邮箱: <input type="email" /></li>
5          <li>网址: <input type="url" /></li>
6          <li>日期: <input type="date" /></li>
7          <li>时间: <input type="time" /></li>
8          <li>数量: <input type="number" /></li>
9          <li>手机号码: <input type="tel" /></li>
10         <li>搜索: <input type="search" /></li>
11         <li>颜色: <input type="color" /></li>
12         <!-- 当我们点击提交按钮就可以验证表单了 -->
13         <li> <input type="submit" value="提交"></li>
14     </ul>
15 </form>

```

常用属性

- placeholder 提示用户输入内容
- value属性：用户输入的内容，提交之后会发送给后端服务器
- name属性：当前控件的含义，提交之后可以告诉后端发送过去的数据是什么含义
- 后端接收到数据的格式是：name的属性值 = value的属性值
 - name属性对于单选框有分组功能 • 有相同name属性值的单选框为一组，一组中只能同时有一个被选中
- checkbox：默认选中
- multiple：上传多个文件

注意点：

- 如果需要使用以上按钮功能，需要配合form标签使用
- form使用方法：用form标签把表单标签一起包裹起来即可

• button标签

- type属性值（同input的按钮系列） submit reset button(配合js使用)

select标签

- select标签：下拉菜单的整体
- option标签：下拉菜单的每一项
- 常见属性：
- selected：下拉菜单的默认选中

textarea标签

- 场景：在网页中提供可输入多行文本的表单控件
- 标签名：textarea
- 常见属性：
- cols：规定了文本域内可见宽度
- rows：规定了文本域内可见行数

label标签

- 场景：常用于绑定内容与表单标签的关系
- 标签名：label
- 使用方法①：
 - 使用label标签把内容（如：文本）包裹起来
 - 在表单标签上添加id属性
 - 在label标签的for属性中设置对应的id属性值
- 使用方法②：
 - 直接使用label标签把内容（如：文本）和表单标签一起包裹起来
 - 需要把label标签的for属性删除即可

补充

- readonly="readonly"：设置文本输入框为只读（不能编辑）
- disabled="disabled"：控件属于非激活状态

- required 必须填写
- pattern 定义规则

语义化标签

无语义标签

- 常用于布局的无语义标签有哪两个？各自的特点有哪些？
- div：独占一行
- span：一行中可以显示多

有语义标签

- header：网页头部
- nav：网页导航
- footer：网页底部
- aside：网页侧边栏
- section：网页区块
- article：网页文章

```
1 <div class="header"> </div>
2
3 <div class="nav"> </div>
4
5 <div class="content"> </div>
6
7 <div class="footer"> </div>
8
```

css初识

- css中文名层叠样式表，作用是给页面中的html标签设置样式
- css语法 选择器{属性名：属性值}
- 三种引入：内嵌式 在style标签里,外联式：写在css文件里 用link引入，行内式：在标签里的style属性里

选择器

1. 标签选择器：标签名 {css属性名：属性值}
2. 类名选择器：.类名 {css属性名：属性值} 标签通过class使用
3. id选择器：#id名 {css属性名：属性值} 标签通过id使用 一般配合js使用
4. 通配符名选择器：*{css属性名：属性值} 所有标签选中

字体样式

- 字体大小: font-size: 数字+px
- 字体粗细: font-weight正常: normal 或 400 • 加粗: bold 或 700
- > 字体样式: font-style 正常: normal • 倾斜: italic
- > 字体系列: font-family 具体字体1,具体字体2,具体字体3,具体字体4,...,字体系列
- > 字体连写: font : style weight size family;

文本样式

- 文本缩进: text-indent: 数字+px 数字+em (推荐: 1em = 当前标签的font-size的大小)
- 文本水平对齐方式: text-align: left center right
- 文本修饰: text-decoration: underline下划线, line-through 删除线, overline上划线, none无线 开发中会使用 text-decoration : none ;清除a标签默认的下划线
- text-align : center 能让哪些元素水平居中?
 1. 文本
 2. span标签、a标签
 3. input标签、img标签
 4. 注意点:
如果能让以上元素垂直平居中 需要给以上元素的 父元素 设置行高: line-height
让单行文本垂直居中可以设置 line-height : 文字父元素高度
网页精准布局时, 会设置 line-height : 1 可以取消上下间距
font : style weight size/line-height family ;

补充

- 字母之间的间距letter-spacing
- 单词之间间距word-spacing
- 文本的大小写text-transform
- 文本的装饰text-decoration
- 自动换行word-wrap

颜色

文字: color

背景background-color

方法rgb rgba(a为透明度0~1) 十六进制

复合选择器

后代选择器: 空格

选择器语法: 选择器1 选择器2 { css }

子代选择器: >

选择器语法：选择器1 > 选择器2 { css }

并集选择器：，

选择器语法：选择器1，选择器2 { css }

交集选择器：紧挨着

选择器语法：选择器1选择器2 { css }

伪类选择器：

- 选择器：hover 鼠标悬停 显示内容
- : link 未访问链
- : visited 已经访问
- : active活动链接

焦点伪类选择器：

input: focus{}

结构伪类选择器

- E:first-child/last-child/nth-child(n)/nth-last-child(){}
- n为：0、1、2/偶数2n even/奇数 2n+1/-1 odd/找到前五个 -n+5/后五个n+5
- E:nth-of-type(n){}
- :nth-child→ 直接在所有孩子中数个数（旗下所有孩子男女都有）
- :nth-of-type→ 先通过该 类型 找到符合的一堆子元素，然后在这一堆子元素中数个数（只有旗下的男孩）

```
1  <style>
2      ul li:first-child{
3          background-color: red;
4      }
5  </style>
6
7  <ul>
8      <li>列表项一</li> 选中他
9      <li>列表项二</li>
10     <li>列表项三</li>
11     <li>列表项四</li>
12 </ul>
13
14 <style>
15     ul li:nth-child(2){      选中 ul 的第二个孩子 必须为li
16         /* 字体变成红色 */
17         color: red;
18     }
19
20     ul li:nth-of-type(2){      选中ul 中第二个孩子 li
21         /* 背景变成绿色 */
22         background-color: green;
23     }
```

```

24     </style>
25
26
27     <ul>
28         <li>列表项一</li>
29         <p>乱来的p标签</p> 没被选中
30         <li>列表项二</li>绿色
31         <li>列表项三</li>
32         <li>列表项四</li>
33     </ul>

```

伪元素

- `::before` // `after`
- \1. 必须设置`content`属性才能生效
- \2. 伪元素默认是行内元素

属性选择器

- ~具有这个属性值的标签--^以什么开头--\$结尾--*含指定字符串--|指定属性值为含他的或以他为开头
- `E[arr]`选择`arr`属性的E元素
- `E[arr="var"]` 选择`arr`为`var`的E元素

```

1  /*只选择 type =text 文本框的input 选取出来 */
2  input[type=text] {
3      color: pink;
4  }
5  /* 选择首先是div 然后 具有class属性 并且属性值 必须是 icon开头的这些元素 */
6  div[class^=icon] {
7      color: red;
8  }
9  /* 选择首先是section 然后 具有class属性 并且属性值 必须是 data结尾的这些元素 */
10 section[class$=data] {
11     color: blue;
12 }

```

背景 (以下重点)

1. 背景颜色: `background-color` (`bgc`)
2. 背景图片: `background-image` (`bgi`)
3. 背景固定: `background-attachment: scroll` 随页面滚动 `fixed` 固定
4. 背景大小: `background-size: 宽度 高度; contain` 等比 不超出盒子的最大值 `cover` 覆盖满盒子没有恐怖
5. ==线性渐变: `background-image: linear-gradient (方向: 0deg, 颜色1, 颜色2) =`
6. 重复线性渐变 (`repeating-linear-gradient (方向: 0deg, 颜色1, 颜色2))`
7. 径向`background-image: radial-gradient (渐变形状 圆心位置 at+关键字, 颜色1, 颜色2)`
8. 重复径向渐变 (`repeating-radial-gradient (渐变形状 圆心位置 at+关键字, 颜色1, 颜色2)`
9. 背景平铺: `: background-repeat (bgr) : repeat` 默认水平和垂直都平铺 `no-repeat` 不平铺 `repeat-x/y` 水平/垂直平铺
10. 背景位置: `background-position (bgp) : left center right top center bottom` 数字+px(x轴 y轴)

11. 背景连写: background: color image repeat position

- 如果需要设置单独的样式和连写
1. 要么把单独的样式写在连写的下面
 2. 要么把单独的样式写在连写的里面

元素显示模式

块级元素

- 独占一行（一行只能显示一个）
- 宽度默认是父元素的宽度，高度默认由内容撑开
- 可以设置宽高

行内元素

- 一行可以显示多个
- 宽度和高度默认由内容撑开
- 不可以设置宽高

行内块元素

- 一行可以显示多个
- 可以设置宽高

元素模式转换：

- display: block转换为块级元素 inline-block行内块 inline 行内
- p标签中不要嵌套div、p、h等块级元素
- a标签不能嵌套a标签
- css具有继承性、层叠性、优先级
- 子元素有默认继承父元素样式的特点

优先级排序

- 继承 < 通配符选择器 < 标签选择器 < 类选择器 < id选择器 < 行内样式 < !important
- !important写在属性值的后面，分号的前面！
- !important不能提升继承的优先级，只要是继承优先级最低！
- 实际开发中不建议使用 !important。
- !important如果不是继承，则权重最高，天下第一！

盒子模型

盒子模型一共有几个部分组成？分别是什么？

1. 内容区域: content
2. 边框区域: border
3. 内边距区域: padding

4. 外边距区域: margin

边框

- 边框: border-width粗细/style样式/color颜色 (solid实线, dashed 虚线, 点线 dotted)
属性值: 单个取值的连写, 取值之间以空格隔开 如: `border : 10px solid red;`
- 属性名: border - 方位名词 属性值: 连写的取值
- > 给盒子设置四周 20像素、实线、蓝色的边框, 属性应该如何设置?
`border: 20px solid blue;`
- > 给盒子设置上边框 10像素、虚线、黄色的边框, 属性应该如何设置?
`border-top: 10px dashed yellow;`
- 设置边框图片: ``border-image-source: url(images/1.jpg);``
- 图片边框偏移: ``border-image-slice: 60;``
- 图片边框宽度: ``border-image-width: 30px;``
- 图片边框: ``border-image``
- 属性值: source 图片路径 slice 偏移量 width 边框宽度 outset 边框向外延申距离 repeat 平铺方式

内边距

内边距: padding (外边距 margin 方法与其同)

1. 一个值: 全部 两个值: 上下 左右 三个值: 上 左右 下 四个值: 上 右 下 左
2. 属性名: padding - 方位名词 属性值: 数字 + px

注意: 水平方向的margin和padding布局中有效 垂直方向的margin和padding布局中无效

盒子阴影

- `box-shadow: h-shadow v-shadow blur spread color inset;`
- h/v必需 水平/垂直 b 模糊度 s尺寸 c颜色 i内部阴影 outset 外部

文字阴影

`text-shadow: h-shadow v-shadow blur color`

如果不想盒子被撑大?

- > 解决方法 ①: 手动内减
 - 操作: 自己计算多余大小, 手动在内容中减去
 - 缺点: 项目中计算量太大, 很麻烦
- > 解决方法 ②: 自动内减
 - 操作: 给盒子设置属性 `box-sizing : border-box;`即可
 - 优点: 浏览器会自动计算多余大小, 自动在内容中减去
- 标准流: 又称文档流, 是浏览器在渲染显示网页内容时默认采用的一套排版规则, 规定了应该以何种方式排列元素

- > 常见标准流排版规则： 块级元素：从上往下，垂直布局，独占一行 行内元素 行内块元素：从左往右，水平布局，空间不够自动折行

浮动

`float:left/right`

> 浮动元素的特点有哪些？

- 浮动元素会脱标，在标准流中不占位置
- 浮动元素比标准流高出半个级别，可以覆盖标准流中的元素
- 浮动找浮动，下一个浮动元素会在上一个浮动元素后面左右浮动
- 浮动元素有特殊的显示效果：① 一行可以显示多个 ② 可以设置宽高
- 浮动的元素不能通过`text-align:center`或者`margin:0 auto`

定位

`position: relative/absolute/fixed`

静态定位

- 静态定位就是之前标准流，不能通过方位属性进行移动
- 之后说的定位不包括静态定位，一般特指后几种：相对、绝对、固定

相对定位

- 需要配合方位属性实现移动
- 相对于自己原来位置进行移动
- 在页面中占位置 → 没有脱标

绝对定位

- 需要配合方位属性实现移动
- 默认相对于浏览器可视区域进行移动
- 在页面中不占位置 → 已经脱标

子绝父相

- 子元素：绝对定位
- 父元素：相对定位
- 父元素是相对定位，则对网页布局影响最小

> 使用子绝父相水平垂直居中的操作是什么？

1. 子绝父相 `left: 50%; top: 50%;`
2. `transform: translate (-50%, -50%) ;` //位移自己宽度高度的一半

固定定位

- 需要配合方位属性实现移动

- 相对于浏览器可视区域进行移动
- 在页面中不占位置 → 已经脱标

不同布局方式元素的层级关系：

标准流 < 浮动 < 定位

不同定位之间的层级关系：

- 相对、绝对、固定默认层级相同
- 此时HTML中写在下面的元素层级更高，会覆盖上面的元素
- **z-index：数字；修改定位元素层级**

装饰

垂直对齐：vertical-align

属性值：baseline 默认 基线对齐 top 顶部 middle 中部 bottom 底部

- 可以解决的问题
- 文本框和表单按钮无法对齐问题
- input和img无法对齐问题div中的文本框，文本框无法贴顶问题
- div不设高度由img标签撑开，此时img标签下面会存在额外间隙问题
- 使用line-height让img标签垂直居中问题

=光标类型 cursor==

属性值：default 箭头 pointer小手 text 工字型 move 十字光标

防止拖拽文本域 resize

```
textarea{ resize: none;}
```

边框圆角：

border-radius 属性值：数字+px 百分比

溢出部分显示效果

属性值：visible 溢出可见 hidden 溢出隐藏 scroll 显示滚动条 auto 自动显示滚动条

元素本身隐藏

visibility: hidden；占位置

display: none；不占位置 display: block；（显示）

边框合并

border-collapse: collapse;

过度

```
1  /* 过渡配合hover使用，谁变化谁加过渡属性 */
2      .box {
3          width: 200px;
4          height: 200px;
5          background-color: pink;
6          /* 宽度200，悬停的时候宽度600，花费1秒钟 */
7          /* transition: width 1s, background-color 2s; */
8          /* 如果变化的属性多，直接写all,表示所有 */
9          transition: all 1s;
10     }
11     .box:hover {
12         width: 600px;
13         background-color: red;
14     }*/
15
```

CSS 三角

```
1  .box3 {
2      /* 宽度 高度为0 */
3      width: 0;
4      height: 0;
5      /*居中*/
6      margin: 100px auto;
7      /*20像素 实线的透明色*/
8      border: 20px solid transparent;
9      /*右边框为红色*/
10     border-right-color: red;
11 }
```

CSS 2D转换

转换属性: transform

- **移动: translate** transform:translate(x,y) transform:translateX/Y(n)

不影响其他元素的位置,百分比是自己宽度高度的多少，行内标签无用 **旋转: rotate** transform:rotate(0deg) 角度为正 顺时针，负 逆时针; 中心点为元素中心点 360deg时，可以给本身添加过渡transition **缩放: scale** transform:scale(x,y) 两个参数：宽和高放大;一个参数：第二个参数和第一个参数一样;0.5是缩小 优势：可以改变中心角transform-origin: 方位词 方位词

注意:

- 1.同时使用多个转换，其格式为:transform: translate() rotate() scale() ..等,
- 2.其顺序会影转换的效果。（先旋转会改变坐标轴方向）
- 3.当我们同时有位移和其他属性的时候，记得要将位移放到最前
- 转换transform我们简单理解就是变形有2D和3D之分我们暂且学了三个分别是位移旋转和缩放

- 2D移动translate(x, y)最大的优势是**不影响其他盒子**，里面参数用%，是相对于自身宽度和高度来计算的可以分开写比如translateX(x)和translateY(y)
- 2D旋转rotate(度数)可以实现旋转元素度数的单位是**deg**
- 2D缩放sacle(x,y)里面参数是数字不跟单位可以是小数最大的优势不影响其他盒子**设置转换中心点transform-origin : x y;**参数可以百分比、像素或者是方位名词
- **当我们进行综合写法，同时有位移和其他属性的时候，记得要将位移放到最前**

CSS 动画

制作动画分两步

1. 先定义动画

- **@keyframes 定义动画**
- **@keyframes 动画名称{ 0%{} 100%{}}**
- **0%是动画的开始，100%是动画的完成。这样的规则就是动画序列。**
- 在@keyframes中规定某项CSS样式，就能创建由当前样式逐渐改为新样式的动画效果。
- 动画是使元素从一种样式逐渐变化为另一种样式的效果。您可以改变任意多的样式任意多的次数。请用百分比来规定变化发生的时间，或用关键词"from"和"to"，等同于0%和100%。

2. 在使用(调用)动画

- animation-name: 动画名称
- animation-duration: 持续时间

动画属性

鼠标经过div 让这个div停止动画，鼠标离开就继续动画 `!animation-play-state: paused;`

简写

- animation :动画名称--持续时间--运动曲线--何时开始--播放次数--是否反方向--动画起始或者结束的状态;
- animation: name--duration (数字+s) -- timing-function(**linear匀速** ease慢到快 steps步长) --delay (数字+s) -- iteration-count (iteration重复的 conut次数 (数字) **infinite无限**) -- direction (*normal (向前) 默认值 reverse (向后) **alternate 先向前 后向后** alternate-reverse 先向后后向前) --fill-mode (none 无 **forwards 最后** backwards 最开始 both都保存) ;
- **常写: animation: name 1.4s linear infinite**
- 简写前两个必须写，属性里面不包含animation-play-state
- **暂停动画: animation-play-state: paused;**经常和鼠标经过等其他配合使用
- **想要动画走回来，而不是直接跳回来: animation-direction : alternate**
- 盒子动画结束后，停在结束位置: animation-fill-mode : forwards

CSS 3D转换

三位坐标

- x轴: 水平向右，注意: x右边是正值，左边为负值
- y轴: 垂直向下，注意: y下面是正值，y上面为负值
- z轴: 垂直屏幕，注意: 往外是正值，往里面为负值

主要知识

- 3D位移: `translate3d(x,y,z);`可以分开写, 同2d转换
- 3D旋转: `rotate3d(x (1) ,y (0) ,z (0) ,deg (45deg))`deg为自定义轴旋转, 可以分开写, 同2d转换
- `rotate3d(1,1,0,45deg)`沿着对角线旋转

左手准则

左手准则

- 左手的手拇指指向 x轴的正方向
- 其余手指的弯曲方向就是该元素沿着x轴旋转的方向



左手准则

- 左手的手拇指指向 y轴的正方向
- 其余手指的弯曲方向就是该元素沿着y轴旋转的方向 (正值)



透视: perspective

用来实现3d效果, 透视值越小越大。给父元素添加, 位移的Z轴的正值越大物体越大

3D呈现

`transform-style flat` 不开启, `preserve-3d` 开启

移动web

视口

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

二倍图

- pxcook 开发 设计图 选择 2x 防止模糊失真

百分比布局

- 宽度自适应 高度固定

flex布局/弹性布局

- 浏览器提倡 的布局模型
- 布局网页更简单、灵活 避免脱标
- 父元素添加 display: flex,
- 宽高不设置 : 宽度为内容的多少 高度 拉伸 (容器高度)

组成

- 弹性容器 弹性盒子 主轴 侧轴 / 交叉轴

主轴对齐方式

- justify-content

属性值	作用
flex-start	默认值, 起点开始依次排列
flex-end	终点开始依次排列
center	沿主轴居中排列
space-around	弹性盒子沿主轴均匀排列, 空白间距均分在弹性盒子两侧
space-between	弹性盒子沿主轴均匀排列, 空白间距均分在相邻盒子之间
space-evenly	弹性盒子沿主轴均匀排列, 弹性盒子与容器之间间距相等

侧轴对齐方式

- align-items
 - align-self 控制 在侧轴的对齐方式 设置在子级中
-

属性值	作用
flex-start	默认值, 起点开始依次排列
flex-end	终点开始依次排列
center	沿侧轴居中排列
stretch	默认值, 弹性盒子沿着主轴线被拉伸至铺满容器

伸缩比

- flex: 数字 (整数) 占用父盒子剩余尺寸的 几份 子级中使用

改变主轴方向

- flex-direction: row行 column列
- flex-wrap: wrap 多行排列
- align-content 行对齐方式 与 justify-content 基本相同

移动适配

rem

- rem单位
 - 相对单位
 - rem单位是相对于HTML标签的字号计算结果
 - 1rem = 1HTML字号大小
- 媒体查询
 - @media (尺寸) { html{ font-size: 32px;}}
 - min-width 从小到大写
 - max-width 从大到小
- rem单位尺寸
- 根据视口宽度，设置不同的HTML标签字号
- 书写代码，尺寸是rem单位
 - 2.1 确定设计稿对应的设备的HTML标签字号
 - 查看设计稿宽度 → 确定参考设备宽度(视口宽度) → 确定基准根字号 (1/10视口宽度)
 - 2.2 rem单位的尺寸
 - $N * 37.5 = 68 \rightarrow N = 68 / 37.5$
 - rem单位的尺寸 = px单位数值 / 基准根字号

flexible

- script标签引入flexible.js 框架

less

- css预处理器 文件为 .less
- 浏览器不识别less代码，网页引入对应的css文件
- 单行注释 // css不显示 ctrl+/
- 多行注释/**/ 会输出到css中 shift+alt+a
- 计算: + - : * 除: (100/4) 100./4 转rem : /37.50
- 定义变量: @变量名: 属性值
- 使用: css: @变量名
- 导入 less文件: @import "路径"
- 导出css文件: //out: "路径" json文件 "out": "路径" //out: false 禁止导出

VW/VH

- vw: viewport width 1vw = 1/100视口宽度
- vh: viewport height 1vh = 1/100视口高度
- vw/vh单位尺寸(不能混用)
 - 确定设计稿对应的vw尺寸 (1/100视口宽度)
 - 查看设计稿宽度 → 确定参考设备宽度 (视口宽度) → 确定vw/vh尺寸 (1/100 视口宽度)
 - vw单位的尺寸 = px单位数值 / (1/100 视口宽度)

bootstrap

- 使用 Bootstrap框架快速开发响应式网页
-

使用步骤

1. 引入：Bootstrap提供的CSS代码

```
<link rel="stylesheet" href="./bootstrap-3.3.7/css/bootstrap.css">
```

2. 调用类：使用Bootstrap提供的样式

➤ **container**：响应式布局版心类

-

	超小屏幕	小屏幕	中等屏幕	大屏幕
响应断点	< 768px	>= 768px	>= 992px	>= 1200px
别名	xs	sm	md	lg
容器宽度	100%	750px	970px	1170px
类前缀	col-xs-*	col-sm-*	col-md-*	col-lg-*
列数	12	12	12	12
列间隙	30px	30px	30px	30px

-

- **.container**是 Bootstrap 中专门提供的类名，所有应用该类名的盒子，默认已被**指定宽度且居中**。
- **.container-fluid**也是 Bootstrap 中专门提供的类名，所有应用该类名的盒子，**宽度均为 100%**。
- 分别使用**.row**类名和**.col**类名定义栅格布局的行和列。

- 手册用法：
 - Bootstrap预定义了大量类用来美化页面，掌握手册的查找方法是学习全局样式的重点。
 - 网站首页 → Bootstrap3中文文档 → 全局CSS样式 → 按分类导航查找目标类
 - **<button class="基本样式类 具体样式类">成功</button>**

javascript

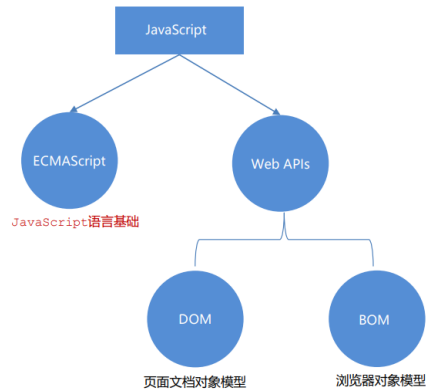
JavaScript是什么？

- JavaScript: 运行在客户端（浏览器）的编程语言，实现人机交互效果
- 网页特效 表单验证 数据交互 服务端编程 node.js
- **组成：ECMAScript webAPIs(DOM BOM)**
 - ECMAScriptL:规定了js基础语法核心知识

- DOM 操作文档
- BOM 操作浏览器，比如页面弹窗

1. 作用和分类

- 作用: 就是使用 JS 去操作 html 和浏览器
- 分类: **DOM** (文档对象模型)、**BOM** (浏览器对象模型)



书写位置： 内联 内部 外部

内部：

- 规范: **script****标签写在 `</body>` 上面**
- 拓展: `alert('你好, js')` 页面弹出警告对话框
- 我们将 `< script>` 放在HTML文件的底部附近的原因是浏览器会按照代码在文件中的顺序加载HTML。
- 如果先加载的 JavaScript 期望修改其下方的 HTML，那么它可能由于 HTML 尚未被加载而失效。
- 因此，将 JavaScript 代码放在 HTML页面的底部附近通常是最好的策略。

外部：

- 代码写在以.js结尾的文件里 引入方式 `<script src=""></script>`
- 1. script标签中间无需写代码，否则会被忽略！
- 2. 外部JavaScript会使代码更加有序，更易于复用，
- 且没有了脚本的混合，HTML 也会更加易读，因此这是个好的习惯。

内联：

代码写在标签内部 `< button onclick="alert('你好js')">< /button>`

```
1 <!-- 内联 -->
2 <button onclick="alert('月薪过万')">点击我</button>
3 <!-- 内部js -->
4 <script>
5     // alert('你好, js')
6 </script>
7 <!-- 外部js -->
8 <script src="./my.js">
9     alert(11)
10 </script>
```

注释

- JavaScript 注释有那两种方式？单行注释 // 多行注释 /* */
JavaScript 结束符注意点 结束符是？ 分号；
结束符可以省略吗？因为 js 中换行符（回车）会被识别成结束符 但为了风格统一，要写每句都写，要么每句都不写

输入输出语句

- 输入：prompt()
- 输出：alert() 输出的内容为标签 想body输出内容
- document.write() 页面弹出警告框
- console.log() 控制台语法 程序员调试使用

字面量

- 字面量 (literal) 是在计算机中描述 事/物
- 如：[] 数组字面量 {} 对象字面量 100 数字字面量

变量

- 变量：装数据的容器

声明变量有两部分构成：声明关键字、变量名（标识） Let 变量名/标识符

- let 不允许多次声明一个变量。

能够说出变量的本质是什么

- 内存：计算机中存储数据的地方，相当于一个空间
变量：是程序在内存中申请的一块用来存放数据的小空间

规则：

1. 不能用关键字
2. 只能用下划线、字母、数字、\$组成，且数字不能开头
3. 字母严格区分大小写，如 Age 和 age 是不同的变量

规范:

1. 起名要有意义
2. 遵守小驼峰命名法
3. 第一个单词首字母小写，后面每个单词首字母大写。例：userName

三种变量

全局变量 js标签内 局部变量 函数内 块级变量 {}

采取就近原则的方式来查找变量最终的值

数组

声明: let 数组名 = []

数组是按顺序保存，所以每个数据都有自己的编号

1. 计算机中的编号从0开始
2. 在数组中，数据的编号也叫索引或下标
3. 数组可以存储任意类型的数据

取值 数组名[下标]

计算机程序可以处理大量的数据，为什么要给数据分类？

- 更加充分和高效的利用内存
- 也更加方便程序员的使用数据

数组操作

- 数组长度 a.length
- 查 数组[下标]
- 增 arr.push(新增的内容) arr.unshift(新增的内容)
- 改 数组[下标] = 新值
- 删 arr.pop() arr.shift() arr.splice(操作的下标,删除的个数)
- 数组.push() 方法将一个或多个元素添加到数组的末尾，并返回该数组的新长度 (重点)
- 数组.pop() 方法从数组中删除最后一个元素，并返回该元素的值
- 数组.shift() 方法从数组中删除第一个元素，并返回该元素的值
- 数组.splice(起始位置，删除几个元素) 方法 删除指定元素

```
1 <script>
2     let arr = ['pink', 'hotpink', 'deeppink']
3     // 访问 / 查询
4     console.log(arr[0])
5     // 2. 改
6     arr[0] = 'lightpink'
7     //增 push  unshift
8     arr.push('blue', 'skyblue')
9     arr.unshift('pink', 'blue')
10    //删  pop  splice
```

```
11     arr.pop()    // 删除最后一个元素
12     arr.splice(1, 1) 从下标为1起 删一个
13
14     </script>
```

基本数据类型

number 数字型 string 字符串型 boolean 布尔型 undefined 未定义型 null 空类型

引用数据类型

object 对象 function 函数 array 数组

注意：

- JavaScript 中的正数、负数、小数等 统一称为 数字类型。
- JS 是弱数据类型，变量到底属于那种类型，只有赋值之后，我们才能确认
- Java是强数据类型 例如 int a = 3 必须是整数

字符串

- 通过单引号（'）、双引号（"）或反引号包裹的数据都叫字符串，单引号和双引号没有本质上的区别，
- 推荐使用单引号。外双内单，或者外单内双
- 拼接 +号 在反引号前前提下使用变量\${变量名} 不需要拼接号

```
1  <script>
2      //+ 号拼接
3      let uname = prompt('请输入您的名字')
4      document.write('我的名字是: ' + uname)
5      //反引号 字符串
6      let age = 81
7      document.write(`我今年${age - 20}岁了`)
8      document.write(`
9          <div>123</div>
10         <p>abc</p>
11     `)
12 </script>
```

布尔数据类型有几个值？ true 和 false

什么时候出现未定义数据类型？以后开发场景是？ 定义变量未给值 如果检测变量是undefined就说明没有值传递

null是什么类型？ 开发场景是？ 空类型 如果一个变量里面确定存放的是对象，如果还没准备好对象，可以放个null

控制台语句经常用于测试结果来使用。数字型和布尔型颜色为蓝色，字符串和undefined颜色为灰色 通过 typeof 检测数据类型

转换数据

Number(数据)

- 如果字符串内容里有非数字，转换失败时结果为NaN（Not a Number）即不是一个数字
- NaN也是number类型的数据，代表非数字

parseInt(数据)

只保留整数

parseFloat(数据)

可以保留小数

转换为字符型:

String(数据) 变量.toString(进制)

运算符

1. 算术运算符: + - * / % 优先级 先乘除后加减, 有括号先算括号里面的
2. 赋值运算符: += -= *= /= %= = 将等号右边的值赋予给左边, 要求左边必须是一个容器变量
3. 一元运算符 -- ++、二元运算符、三元运算符? :
4. 比较运算符: > < >= <= == === != = 只会得到true或false

注意: NaN不等于任何值, 包括它本身

= 和 == 和 === 怎么区别? = 是赋值 == 是判断 只要求值相等, 不要求数据类型一样即可返回true === 是全等 要求值和数据类型都一样返回的才是true 开发中, 请使用 ===

5. 逻辑运算符: 1. 左边为假短路 &&与 一假则假 2. 左边为真则 短路 ||或 一真则真 3. ! 非 真变假 假变真真

```
let a = 3 > 5 && 2 < 7 && 3 == 4
console.log(a);
```

← 答案是false, 此时发生了逻辑与中断

```
let b = 3 <= 4 || 3 > 1 || 3 != 2
console.log(b);
```

← 答案是true, 此时发生了逻辑或中断

```
let c = 2 === "2"
console.log(c);
```

← 答案是false 数据类型不匹配

```
let d = !c || b && a
console.log(d);
```

← 答案是true, 此时发生了逻辑或中断

分支语句

- if分支语句 三元运算符 条件? 满足条件代码: 不满足代码 switch 语句
- switch case语句一般用于等值判断, 不适合于区间判断

- switch case一般需要配合break关键字使用 没有break会造成case穿透
- continue: 结束本次循环, 继续下次循环 break: 跳出所在的循环

```

1  <script>
2      //if语句
3      if () { } else if () {} else()
4      //switch
5      switch (2) {
6          case 1:
7              alert(1)
8              break
9          case 2:
10             alert(2)
11             break
12          case 3:
13             alert(3)
14             break
15          default:
16             alert('没有数据')
17      }
18
19      // 1. 用户输入数字
20      let num = prompt('请您输入一个数字')
21      // 2. 判断条件是 小于 10 则数字前面 + '0' 01 否则 不补
22      // let t = num >= 10 ? num : '0' + num
23      let t = num < 10 ? '0' + num : num
24      document.write(t)
25  </script>

```

循环语句

for循环和while循环有什么区别呢:

1. 当如果明确了循环的次数的時候推荐使用for循环
2. 当不明确循环的次数的時候推荐使用while循环
3. 外层循环一次, 内层循环全部

```

1  <script>
2      // 循环必须有3要素
3      // 变量的起始值
4      let i = 1
5      // 终止条件
6      while (i <= 3) {
7          document.write('月薪过万 <br>')
8          // 变量变化
9          i++
10     }
11     //for循环
12     for (let i = 1; i <= 10; i++) {
13         document.write('月薪过万 <br>')
14     }

```

```

15 // 记忆单词案例
16 // 分析
17 // 1. 外面的循环 记录第n天
18 for (let i = 1; i < 4; i++) {
19     document.write(`第${i}天 <br>`)
20     // 2. 里层的循环记录 几个单词
21     for (let j = 1; j < 6; j++) {
22         document.write(`记住第${j}个单词<br>`)
23     }
24 }
25
26 </script>
27

```

函数

关键词 function

- 格式 function 函数名 () {} 调用 函数名()
- 形参：声明函数时写在函数名右边小括号里的叫形参（形式上的参数）
- 实参：调用函数时写在函数名右边小括号里的叫实参（实际上的参数）
 尽量保持形参和实参个数一致

return关键词

- return后面不接数据或者函数内不写return，函数的返回值是undefined
 return能立即结束当前函数, 所以 return 后面的数据不要换行写

立即执行函数

- 有什么作用？防止变量污染
- 立即执行函数需要调用吗？无需调用，立即执行，其实本质已经调用了
- 有什么注意事项呢？多个立即执行函数之间用分号隔开

```

1 <script>
2     //函数
3     function getMax(x, y) {
4     //返回值
5         return x > y ? x : y
6     }
7     // 实参也可以放变量 传参
8     let max = getMax(num1, num2)
9     // 函数表达式
10    let fn = function () {
11        console.log(111)
12    }
13    fn()
14    //立即执行函数
15    (function () {
16        console.log(111)
17    })();

```

对象

- 声明对象: let 对象名{ 方法 属性}
- 方法名: function(){}
- 属性和值用: 冒号 隔开
- 多个属性用, 逗号隔开
- 访问属性 对象.属性名 调用方法 对象.方法名()
- 改: 对象.属性 = 值 对象.方法 = function() {}
- 增: 对象名.新属性名 = 新值
- 删: delete 对象名.属性名
- 对象如果有这个属性相当于重新赋值 对象如果没有这个属性相当于动态添加一个属性

遍历对象

- for in 循环语句
- 语法
- for (let k in 对象名) {} 重点
- k 变量 k 或者 key value

```

1  <script>
2      // 声明人对象
3      let person = {
4          uname: '刘德华',
5          age: 18,
6          sex: '男',
7          // 方法名: function(){}
8          sayHi: function () {
9              console.log('hi~~~')
10         },
11         mtv: function (s) {
12             console.log(s)
13         }
14     }
15     // console.log(uname)
16     // 1. 访问属性 得到值 对象.属性名
17     console.log(person.uname)
18     console.log(person.age)
19     // 2. 访问属性 得到值 对象['属性名']
20     console.log(person['sex'])
21     // 调用方法 对象.方法名()
22     person.sayHi()
23     person.mtv('无间道')
24     //改
25     person.uname = '马云'
26     person.sayHi =function () {
27         console.log('hhh~~~')
28     }
29     //删
30     delete 对象名.属性名

```

```

31      // document.write()
32
33  //遍历对象
34
35      let obj = {
36          uname: '小明',
37          age: 18,
38          sex: '男'
39      }
40      // for in 循环语句
41      // 语法
42      // for (let k in 对象名) {} 重点
43      // k 变量 k 或者 key value
44      for (let k in obj) {
45          console.log(k) // 属性名
46          // console.log(obj.k) // obj.k 意思是 obj里面的k属性
47          // console.log(obj['k'])
48          console.log(obj[k]) // 属性值
49
50          // 为什么这么写?
51          // k === 'uname' === 'age' === 'sex'
52          // // obj.k
53          // // obj['uname']
54          // obj['sex'] === 18
55      }
56  </script>

```

内置数学对象

- random: 生成0-1之间的随机数 (包含0不包括1)
- ceil: 向上取整 floor: 向下取整 max: 找最大数
- min: 找最小数 pow: 幂运算 abs: 绝对值

```

1  <script>
2      console.log(Math.PI) // 圆周率 π
3      console.log(Math.random()) // 随机数 随机抽奖 随机点名
4      // 返回的是小数 但是能得到 0 得不到 1
5      // 向上取整 返回的整数
6      console.log(Math.ceil(1.1)) // ceil 2
7      console.log(Math.ceil(1.5)) // ceil 2
8      console.log(Math.ceil(1.9)) // ceil 2
9      // 向下取整 返回的整数 floor
10     console.log(Math.floor(1.1)) // floor 1
11     console.log(Math.floor(1.5)) // floor 1
12     console.log(Math.floor(1.9)) // floor 1
13     console.log('-----')
14     // round 就近取整( .5往大取证) 返回的整数
15     console.log(Math.round(1.1)) // round 1
16     console.log(Math.round(1.5)) // round 2
17     console.log(Math.round(1.9)) // round 2
18
19     console.log('-----')

```

```

19     console.log(Math.round(-1.1)) // round -1
20     console.log(Math.round(-1.5)) // round -1
21     console.log(Math.round(-1.9)) // round -2
22     function getRandom(min, max) {
23         //如何生成N-M之间的随机数
24         //0-6    6+1 最大值 (6) 减 最小值 (0) +1 +最小值 (0)
25         //1-6    5+1+1 最大值 (6) 减 最小值 (1) +1 +最小值 (1)
26         return Math.floor(Math.random() * (max - min + 1)) + min
27     }
28
29     </script>

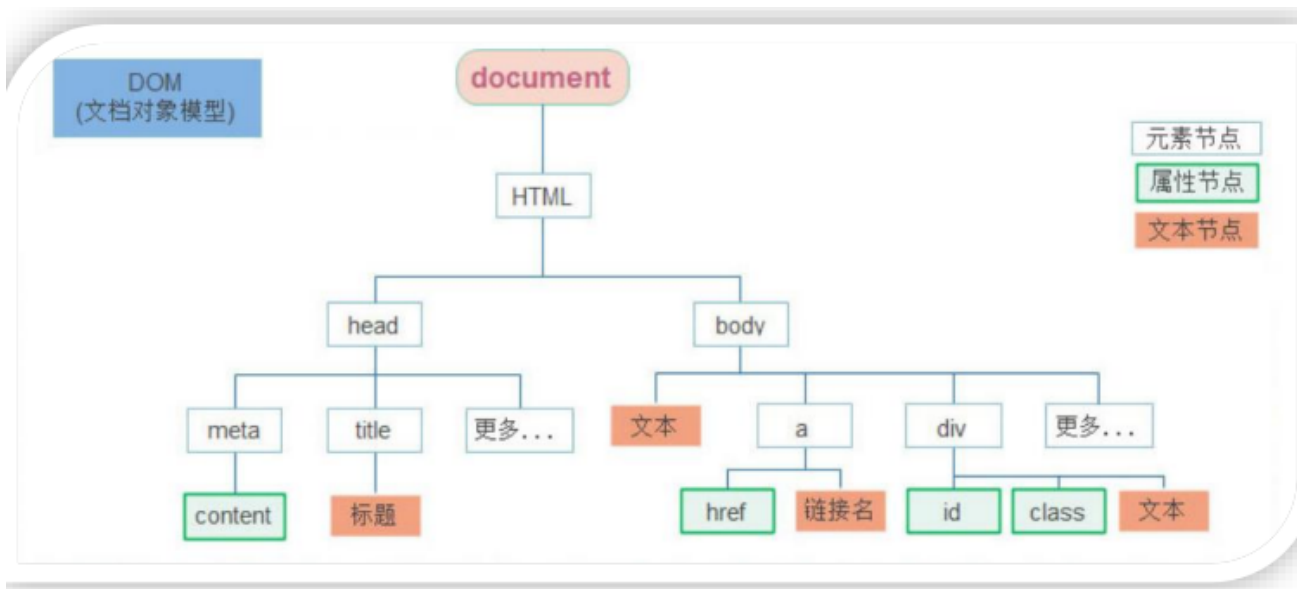
```

Web APIs

DOM: 文本对象模型(操作网页内容), 开发网页内容特效和实现用户交互

DOM树

- 含义: 将 HTML 文档以树状结构直观的表现出来, 我们称之为文档树或 DOM 树
- 作用: 文档树直观的体现了标签与标签之间的关系



DOM对象

- 浏览器根据html标签生成的JS对象 (DOM对象)
- DOM的核心就是把内容当对象来处理

document 对象 document.write() 是用来访问和操作网页内容的

获取对象

- document.querySelector('类名') 获取一个第一个
- document.querySelectorAll('类名') 获取多个 伪数组 通过遍历的方式, 获得里面的每一个dom对象 (元素)

修该元素

1. 修改标签（元素）内容 box是对象 innerText 属性 innerHTML解析标签

```
1  <script>
2      // 1. 获取标签(元素)
3      let box = document.querySelector('div')
4      // 2. 修改标签（元素）内容 box是对象 innerText 属性
5      // 对象.属性 = 值 不识别标签
6      // box.innerText = '有点意思~'
7      // box.innerHTML = '<strong>有点意思~</strong>'
8      // 3. innerHTML解析标签
9      box.innerHTML = '<strong>有点意思~</strong>'
10     box.style.backgroundColor = 'hotpink'
11     box.className = 'one active'
12     // add是个方法 添加 追加
13     box.classList.add('active')
14     // remove() 移除 类
15     box.classList.remove('one')
16     // 切换类
17     box.classList.toggle('one')
18
19     input.value = '小米手机'
20     input.type = 'password'
21 </script>
```

1. 修改元素属性 对象.属性 = 值
2. 修改元素样式属性 对象.style.样式属性 = '值'
 - 注意：
 1. 修改样式通过style属性引出
 2. 如果属性有-连接符，需要转换为小驼峰命名法
 3. 赋值的时候，需要的时候不要忘记加css单位
3. 操作类 className 和 classList
 - className 会覆盖原有的类
 - classList add 添加类 remove 移除类 toggle 切换类（有 删除 无添加）
4. 修改表单元素 按钮关闭 disabled、复选框默认选择 checked、下拉默认选择 selected
 - // disabled 不可用 = false 这样可以让按钮启用
 - 其他两个 true 选中 false 未选

间歇函数（定时器）

```

1  <script>
2      function show() {
3          console.log('月薪过2万')
4      }
5      //创建定时器  setInterval(函数名, 毫秒数)
6      let timer = setInterval(show, 1000)
7      // 清除定时器
8      clearInterval(timer)
9  </script>

```

事件监听

语法 事件源.addEventListener('事件', 事件处理函数) 第三个值 是否捕获了解

1. 事件: click 单击 dblclick双击 mouseenter 鼠标经过 mouseleave鼠标离开 focus/blur 获得失去焦点 scroll 滚动事件 resize窗口尺寸事件 Keydown 键盘按下触发 Keyup 键盘抬起触发

2. 事件监听三要素是什么?

事件源 (谁被触发了)

事件 (用什么方式触发, 点击还是鼠标经过等)

事件处理程序 (要做什么事情)

```

1  <script>
2      //1. 获取按钮元素
3      let btn = document.querySelector('button')
4      //2. 事件监听  绑定事件 注册事件 事件侦听
5      // 事件源.addEventListener('事件', 事件处理函数)
6      btn.addEventListener('click', function () {
7          alert('月薪过万')
8      })
9  </script>

```

高阶函数

函数表达式

let fn = function(){} 函数表达式 为高级函数的一种形式

回调函数

function fn(){} setInterval(fn,1000) 此时fn为回调函数 回头去调用的函数

环境对象

变量this: 他就是个对象【谁调用, this就是谁】是判断this指向的粗略规则

```
btn.addEventListener('click',function(){console.log(this)}) this指向btn
```

思想

排他思想

1. 干掉所有人 使用for循环
2. 复活他自己 通过this或者下标找到自己或者对应的元素

DOM节点

- DOM树里每一个内容都称之为节点
- 元素节点 如div标签
- 属性节点 如class属性
- 文本节点 如 文字

创建节点

创建节点：document.createElement('标签名')

查找结点

- 父节点：子元素.parentNode 返回最近一级的父节点 找不到返回为null
- 子节点：父元素.children 仅获得所有元素节点 返回的还是一个伪数组
- 兄弟节点：nextElementSibling 下一个previousElementSibling 上一个

删除节点

- 删除节点：父元素.removeChild(要删除的元素) 如不存在父子关系则删除不成功
- 删除节点和隐藏节点（display:none）有区别的：隐藏节点还是存在的，但是删除，则从html中删除节点

追加节点：

- 插入到父元素的最后一个子元素 父元素.appendChild(要插入的元素)
- 插入到父元素的某个子元素前面 父元素.insertBefore（要插入的元素，在哪个元素前面）
- 注意：获取元素 类名要加点 追加类 类名不加点

克隆节点：

- 元素.cloneNode(布尔值) cloneNode会克隆出一个跟原标签一样的元素，括号内传入布尔值
- 若为true，则代表克隆时会包含后代节点一起克隆
- 若为false，则代表克隆时不包含后代节点 默认为false

```
1  <body>
2    <ul>
3      <li>我是大毛</li>
4      <li>我是二毛</li>
5    </ul>
6    <script>
7      //获取节点
8      let ul = document.querySelector('ul')
9      // 1. 创建新的标签节点
10     let li = document.createElement('li')
11
12     // 删除节点
```

```

12     ul.removeChild(ul.children[0])
13     li.innerHTML = '我是xiao ming'
14     // 2. 追加节点 父元素.appendChild(子元素) 后面追加
15     ul.appendChild(li)
16     // 3. 追加节点 父元素.insertBefore(子元素, 放到那个元素的前面)
17     ul.insertBefore(li, ul.children[0])
18     //4.将克隆的li节点 追加到ul第一个孩子前面
19     ul.insertBefore(ul.cloneNode(li), ul.children[0])
20 </script>
21 </body>
22

```

时间对象

- 获取当前时间let date = new Date()
- 获取指定时间let date = new Date('1949-10-01')
- 本地时间: new Date().toLocaleString()

方法	作用	说明
getFullYear()	获得年份	获取四位年份
getMonth()	获得月份	取值为 0 ~ 11
getDate()	获取月份中的每一天	不同月份取值也不相同
getDay()	获取星期	取值为 0 ~ 6
getHours()	获取小时	取值为 0 ~ 23
getMinutes()	获取分钟	取值为 0 ~ 59
getSeconds()	获取秒	取值为 0 ~ 59

什么是时间戳

是指1970年01月01日00时00分00秒起至现在的毫秒数，它是一种特殊的计量时间的方式

三种方式获取时间戳

1. 使用 getTime() 方法 let date = new Date()console.log(date.getTime())
2. 简写 +new Date() console.log(+newDate())
3. 使用 Date().now() console.log(Date().now())
4. 但是只能得到当前的时间戳，而前面两种可以返回指定时间的的时间戳

注意：

- 通过时间戳得到是毫秒，需要转换为秒在计算
毫秒数 /1000=秒数
- 转换公式：
d = parseInt(总秒数/ 60/60 /24); // 计算天数
h = parseInt(总秒数/ 60/60 %24) // 计算小时

```
h = h < 10 ? '0' + h : h
m = parseInt(总秒数 / 60 % 60); // 计算分数
m = m < 10 ? '0' + m : m
s = parseInt(总秒数 % 60); // 计算当前秒数
s = s < 10 ? '0' + s : s
```

```
1  <script>
2      let hour = document.querySelector('#hour')
3      let minutes = document.querySelector('#minutes')
4      let scond = document.querySelector('#scond')
5      time()
6      setInterval(time, 1000)
7      function time() {
8          // 1. 得到现在的时间戳
9          let now = +new Date()
10         // 2. 得到指定时间的时间戳
11         let last = +new Date('2021-8-29 18:30:00')
12         // 3. (计算剩余的毫秒数) / 1000 === 剩余的秒数
13         let count = (last - now) / 1000
14         // console.log(count)
15         // 4. 转换为时分秒
16         // h = parseInt(总秒数 / 60 / 60 % 24) // 计算小时
17         let h = parseInt(count / 60 / 60 % 24)
18         h = h < 10 ? '0' + h : h
19         // m = parseInt(总秒数 / 60 % 60); // 计算分数
20         let m = parseInt(count / 60 % 60)
21         m = m < 10 ? '0' + m : m
22         // s = parseInt(总秒数 % 60); // 计算当前秒数
23         let s = parseInt(count % 60);
24         s = s < 10 ? '0' + s : s
25         // console.log(h, m, s)
26         hour.innerHTML = h
27         minutes.innerHTML = m
28         scond.innerHTML = s
29     }
30
31 </script>
```

事件对象

事件对象是对象里有事件触发时的相关信息

在事件绑定的回调函数的第一个参数就是事件对象 一般命名为event、ev、e

e的 部分常用属性

- type 获取当前的事件类型
- clientX/clientY 获取光标相对于浏览器可见窗口左上角的位置
- offsetX/offsetY 获取光标相对于当前DOM元素左上角的位置
- key 用户按下的键盘键的值现在不提倡使用keyCode
- 常用 e.target 真正触发事件的元素 e.target.tagName 事件对象的触发元素的名字

事件流

捕获阶段 从父到子

冒泡阶段 从子到父 事件冒泡是默认存在的

- 当一个元素触发事件后，会依次向上调用所有父级元素的同名事件
- 说明：addEventListener第三个参数传入true代表是捕获阶段触发（很少使用）若传入false代表冒泡阶段触发，默认就是false 若是用 L0 事件监听，则只有冒泡阶段，没有捕获
- 阻止事件流动 事件对象.stopPropagation() 不光在冒泡阶段有效，捕获阶段也有效
- 鼠标经过事件：mouseover 和 mouseout 会有冒泡效果
mouseenter 和 mouseleave 没有冒泡效果(推荐)
- 阻止默认行为，比如链接点击不跳转，表单域的跳转 e.preventDefault()

两种注册事件的区别：

传统on注册（L0）

- 同一个对象,后面注册的事件会覆盖前面注册(同一个事件)
- 直接使用null覆盖偶就可以实现事件的解绑
- 都是冒泡阶段执行的

事件监听注册（L2）

- 语法: addEventListener(事件类型, 事件处理函数, 是否使用捕获)
- 后面注册的事件不会覆盖前面注册的事件(同一个事件)
- 可以通过第三个参数去确定是在冒泡或者捕获阶段执行
- 必须使用removeEventListener(事件类型, 事件处理函数, 获取捕获或者冒泡阶段)
- 匿名函数无法被解绑

```
1  <script>
2      let fa = document.querySelector('.father')
3      let son = document.querySelector('.son')
4      fa.addEventListener('click', function (e) {
5          alert('我是爸爸')
6          e.stopPropagation() //阻止冒泡
7      })
8      son.addEventListener('click', function (e) {
9          alert('我是儿子')
10         // 阻止流动 Propagation 传播
11         e.stopPropagation()
12     })
13     document.addEventListener('click', function () {
14         alert('我是爷爷')
15     })
16     let a = document.querySelector('a')
17     a.addEventListener('click', function (e) {
18         // 阻止默认行为 方法
19         e.preventDefault()
```

```
20     })
21     </script>
```

事件委托

总结：

- 优点：给父级元素加事件（可以提高性能）
- 原理：事件委托其实是利用事件冒泡的特点
- 实现：事件对象.target 可以获得真正触发事件的元素

```
1  <body>
2    <ul>
3      <li>我是第1个小li</li>
4      <li>我是第2个小li</li>
5      <li>我是第3个小li</li>
6      <li>我是第4个小li</li>
7      <li>我是第5个小li</li>
8    </ul>
9    <script>
10      // 不要每个小li注册事件了 而是把事件委托给他的爸爸
11      // 事件委托是给父级添加事件 而不是孩子添加事件
12      let ul = document.querySelector('ul')
13      ul.addEventListener('click', function (e) {
14        // 得到当前的元素
15        e.target.style.color = 'red'
16      })
17    </script>
18  </body>
```

滚动事件

- 事件名：scroll 给 window 或 document 添加 scroll 事件
- 事件名：load 给 window 添加 load 事件 不光可以监听整个页面资源加载完毕，也可以针对某个资源绑定 load 事件
- 事件名：DOMContentLoaded 给 document 添加 DOMContentLoaded 事件

```
1  <script>
2    let div = document.querySelector('div')
3    //页面滚动
4    window.addEventListener('scroll', function() {
5      console.log(111)
6    })
7    // 盒子滚动
8    div.addEventListener('scroll', function() {
9      console.log(111)
10   })
11   //加载事件
12   window.addEventListener('load', function () {
```

```
13     let div = document.querySelector('div')
14     console.log(div)
15   })
16 </script>
```

元素大小和位置

scroll家族

- `scrollLeft`和`scrollTop` 获取取元素内容往左、往上滚出去看不到的距离 可以修改
- `scrollWidth`和`scrollHeight` 获取元素的内容总宽高（不包含滚动条）返回值不带单位
- `document.documentElement` HTML 文档返回对象为HTML元素

offset家族

- `offsetLeft`和`offsetTop` 注意是只读属性 获取元素距离自己定位父级元素的左、上距离
- `offsetWidth`和`offsetHeight` 获取元素的自身宽高、包含元素自身设置的宽高、padding、border

client家族

- `clientLeft`和`clientTop` 注意是只读属性 获取左边框和上边框宽度
- `clientWidth`和`clientHeight` 获取元素的可见部分宽高（不包含边框，滚动条等）
- `resize` 窗口尺寸改变的时候触发事件

```
1     <script>
2         //通过 scrollTop 和 offsetTop 显示隐藏
3         let sk = document.querySelector('.sk')
4         let header = document.querySelector('.header')
5         // 1. 页面滚动事件
6         window.addEventListener('scroll', function () {
7             // 2. 要检测滚动的距离 >= 秒杀模块的offsetTop 则滑入 sk.offsetTop 是 秒杀模块距离上
            边的距离
8             if (document.documentElement.scrollTop >= sk.offsetTop) {
9                 header.style.top = '0'
10            } else {
11                header.style.top = '-80px'
12            }
13        })
14    </script>
```

BOM操作浏览器

window对象

- window对象 是一个全局对象，也可以说是JavaScript中的顶级对象
- 像document、alert()、console.log()这些都是window的属性，基本BOM的属性和方法都是window的。
- 所有通过var/let 定义在全局作用域中的变量、函数都会变成window对象的属性和方法
- window对象下的属性和方法调用的时候可以省略window


```

1  <script>
2      window.document.querySelector('.box')
3      window.setInterval()
4          function fun() {
5              }
6      window.fun()
7      addEventListener('scroll', function () {
8          console.log(111)
9      })
10     window.alert()
11     window.prompt()
12     console.log(window)
13 </script>

```

延时器和定时器

延时器器setTimeout	定时器 setinterval
延迟一段时间之后才执行对应的代码	每隔一段时间就执行一次
let timerId = setTimeout(回调函数, 延迟时间)	let timerId = setInterval(回调函数, 多久执行一次)
清除延时器clearTimeout(timerId)	清除定时器clearInterval(timerId)
延时函数: 执行一次	间歇函数:每隔一段时间就执行一次,除非手动清除

```

1  <button>解除延时器</button>
2  <script>
3      let btn = document.querySelector('button')
4      let timer = setTimeout(function () {
5          console.log(111)
6      }, 3000)
7      // 仅仅执行一次
8      btn.addEventListener('click', function () {
9          clearTimeout(timer)
10     })
11 </script>
12 <script>
13     //定时器
14     function show() {
15         console.log('月薪过2万')
16     }
17     //创建定时器 setInterval(函数名, 毫秒数)
18     let timer = setInterval(show, 1000)
19     // 清除定时器
20     clearInterval(timer)
21 </script>

```

location对象 主要负责网页的地址栏

- location.href 跳转页面
- location.reload() 刷新
- location.search ?后面的内容
- location.hash #后面的内容

```
1 <a href="#one">第一个</a>
2 <a href="#two">第二个</a>
3 <script>
4   location.href = 'http://www.itcast.cn'  跳转链接
5   location.search      //?username=w      表单为post 无法使用
6   location.hash        // #two one
7   location.reload()    //相当于浏览器刷新按钮
8 </script>
```

navigator对象 主要用来获取浏览器的信息

navigator.userAgent 在这个字段里面判断是否有Mobile字段. 如果有表示是手机,反之则表示PC

```
1 <script>
2   // 检测 userAgent (浏览器信息)
3   !(function () {
4     const userAgent = navigator.userAgent
5     // 验证是否为Android或iPhone
6     const android = userAgent.match(/(Android);?[\s\/]+([\d.]+)?/)
7     const iphone = userAgent.match(/(iPhone\sOS)\s([\d_]+)/)
8     // 如果是Android或iPhone, 则跳转至移动站点
9     if (android || iphone) {
10       location.href = 'http://m.itcast.cn'
11     }
12   })()
13
14 </script>
```

history对象 管理历史记录

history.forward() 前进 history.back() 后退 history.go(1/-1) 1前进 -1 后退

```
1 <button class="forward">前进</button>
2 <button class="back">后退</button>
3 <script>
4   let qianjin = document.querySelector('.forward')
5   let houtui = document.querySelector('.back')
6   qianjin.addEventListener('click', function () {
7     // history.forward()
8     history.go(1)
9   })
10  houtui.addEventListener('click', function () {
11    // history.back()
```

```
12     history.go(-1)
13   })
14 </script>
```

js执行机制

- js 把任务分为 同步任务和异步任务

同步任务

let num = 10

异步任务

定时器 事件 click

load 加载

ajax

- 执行执行栈（同步任务）里面的任务，执行完毕再去任务队列（异步任务）里面看看是否有任务，如果有，则得到放入执行栈中执行，再次循环

swiper 插件

插件: 就是别人写好的一些代码,我们只需要复制对应的代码,就可以直接实现对应的效果

<https://www.swiper.com.cn/>

本地存储

- 作用: 可以将数据永久存储在本地(用户的电脑), 除非手动删除 语法 存 localStorage.setItem('键', '值') 取 localStorage.getItem('键') 删 ocalStorage.removeItem('键')
- 存储复杂数据类型: 本地只能存储字符串,无法存储复杂数据类型.需要将复杂数据类型转换成JSON字符串,在存储到本地 转换成JSON字符串的语法
 - JSON.stringify(复杂数据类型) 将复杂数据转换成JSON字符串
 - JSON.parse(JSON字符串) 将JSON字符串转换成对象

```
1 <script>
2   let obj = {
3     uname: '刘德华',
4     age: 17,
5     address: '黑马程序员'
6   }
7   localStorage.setItem('obj', JSON.stringify(obj))
8   JSON.parse(localStorage.getItem('obj'))
```

自定义属性

- 固有属性: 标签天生自带的属性 比如class id title等, 可以直接使用点语法操作
- 自定义属性: 由程序员自己添加的属性,在DOM对象中找不到, 无法使用点语法操作,必须使用专门的API
 - getAttribute('属性名') // 获取自定义属性
 - setAttribute('属性名', '属性值') // 设置自定义属性
 - removeAttribute('属性名') // 删除自定义属性
- 规范一下 为 data-自定义属性
传统的自定义属性没有专门的定义规则,开发者随意定值,不够规范,所以在html5中推出来了专门的data-自定义属性

- 在标签上一律以data-开头 在DOM对象上一律以dataset对象方式获取

```
1 <body>
2   <div class="box" data-index="0" data-name="andy"></div>
3   <script>
4     // 设置自定义属性
5     let box = document.querySelector('.box')
6     // box.setAttribute('myid', 10)
7     // console.log(box.getAttribute('myid'))
8     console.log(box.dataset)
9     console.log(box.dataset.index)
10  </script>
11 </body>
```

jQuery

#

jQuery

常使用的方法

(快速方便查询使用功能 j 就是 JavaScript; Query 查询; 意思就是查询js, 把js中的DOM操作做了封装, 我们可以快速的查询使用里面的功能。)

- 引入方式: 同js script src
- 入口函数: 1. \$(document).ready(function(){代码})
 1. \$(function(){代码}) 推荐使用

```

2. 1 // $('div').hide();
    2 // 1. 等着页面DOM加载完毕再去执行js 代码
    3 // $(document).ready(function() {
    4 //     $('div').hide();
    5 // })
    6 // 2. 等着页面DOM加载完毕再去执行js 代码 入口函数
    7 $(function() {
    8     $('div').hide();
    9 })
10 // 1. $ 是jQuery的别称 (另外的名字) $同时也是jQuery的 顶级对象
11 $(function() {
12     alert(11)
13 });

```

\$符号:jQuery的别称 顶级对象

jQuery对象和DOM对象

- dom对象: 使用原生js获取的对象
- jQuery对象: 使用\$符号 如: \$('div') 伪数组存储
- jQuery对象只能使用jQuery方法 DOM对象则使用原生js属性和方法
- 相互转换: 转后可以使用其属性和方法
 - DOM对象转换为jQuery对象 \$('div')需要元素名加引号 \$(DOM对象(原生获取)) 名字不加引号 \$(myid)
 - jQuery对象转DOM对象 \$('div')[0]/ \$('div').get(0) 0为索引值

```

1 // 1. DOM 对象: 用原生js获取过来的对象就是DOM对象
2 var myDiv = document.querySelector('div'); // myDiv 是DOM对象
3 var mySpan = document.querySelector('span'); // mySpan 是DOM对象
4 console.dir(myDiv);
5
6 // 2. jQuery对象: 用jquery方式获取过来的对象是jQuery对象。 本质: 通过$把DOM元素进行了包装
7 $('div'); // $('div')是一个jQuery 对象
8 $('span'); // $('span')是一个jQuery 对象
9 // 3. jQuery 对象只能使用 jQuery 方法, DOM 对象则使用原生的 JavaScript 属性和方法
10 // myDiv.style.display = 'none';
11 // myDiv.hide(); myDiv是一个dom对象不能使用 jquery里面的hide方法
12 // $('div').style.display = 'none'; 这个$('div')是一个jQuery对象不能使用原生js 的属性和方法
13
14 // 1. DOM对象转换为 jQuery对象
15 // (1) 我们直接获取视频, 得到就是jQuery对象
16 $('video');
17 // (2) 我们已经使用原生js 获取过来 DOM对象
18 var myvideo = document.querySelector('video');
19 (myvideo).play(); jquery里面没有play 这个方法
20 // 2. jQuery对象转换为DOM对象
21 myvideo.play();

```

常用的API

1.1 jQuery基础选择器

- `$("选择器")`直接写css选择器即可加引号

名称	用法	描述
ID选择器	<code>\$("#id")</code>	获取指定ID的元素
全选选择器	<code>\$("*")</code>	匹配所有元素
类选择器	<code>\$(".class")</code>	获取同一类class的元素
标签选择器	<code>\$("div")</code>	获取同一类标签的所有元素
并集选择器	<code>\$("div,p,li")</code>	选取多个元素
交集选择器	<code>\$("li.current")</code>	交集元素

1.2 jQuery层次选择器

名称	用法	描述
子代选择器	<code>\$("ul>li");</code>	使用>号，获取亲儿子层级的元素；注意，并不会获取孙子层级的元素
后代选择器	<code>\$("ul li");</code>	使用空格，代表后代选择器，获取ul下的所有li元素，包括孙子等

1.3 隐式迭代

- 隐式迭代就是把匹配的所有元素内部进行遍历循环，给每一个元素添加方法
`$("div").css("background","pink")` 修改css样式

1.4 筛选选择器

语法	用法	描述
<code>:first</code>	<code>\$("li:first")</code>	获取第一个li元素
<code>:last</code>	<code>\$("li:last")</code>	获取最后一个li元素
<code>:eq(index)</code>	<code>\$("li:eq(2)")</code>	获取到的li元素中，选择索引号为2的元素，索引号index从0开始。
<code>:odd</code>	<code>\$("li:odd")</code>	获取到的li元素中，选择索引号为奇数的元素
<code>:even</code>	<code>\$("li:even")</code>	获取到的li元素中，选择索引号为偶数的元素

- **筛选的方法（重点）**

语法	用法	说明
<code>parent()</code>	<code>\$("#li").parent();</code>	查找父级
<code>children(selector)</code>	<code>\$("#ul").children("li")</code>	相当于 <code>\$("#ul>li")</code> ，最近一级（亲儿子）
<code>find(selector)</code>	<code>\$("#ul").find("li");</code>	相当于 <code>\$("#ul li")</code> ，后代选择器
<code>siblings(selector)</code>	<code>\$("#.first").siblings("li");</code>	查找兄弟节点，不包括自己本身
<code>nextAll([expr])</code>	<code>\$("#.first").nextAll()</code>	查找当前元素之后所有的同辈元素
<code>prevAll([expr])</code>	<code>\$("#.last").prevAll()</code>	查找当前元素之前所有的同辈元素
<code>hasClass(class)</code>	<code>\$('#div').hasClass("protected")</code>	检查当前的元素是否含有某个特定的类，如果有，则返回true
<code>eq(index)</code>	<code>\$("#li").eq(2);</code>	相当于 <code>\$("#li:eq(2)")</code> ，index从0开始

- `parent()` 亲父级 `children()` 亲儿子 `find()` 所有后代 `siblings()` 兄弟节点 `eq()`

1.5链式编程

```
1  $(this).siblings().parent().css('color', 'blue'); //我的兄弟变为蓝色，我本身不变颜色
```

2.1修改css样式

- ```

1 //1. 参数只写属性名 返回参数值
2 $(this).css("color") //red
3 //2. 参数是 属性名, 属性值 属性必须加引号 数字可以不跟单位和引号
4 $(this).css("width", "200")
5 $(this).css("width", "200px")
6 //3. 参数可以是对象形式 属性可以不加引号 用冒号隔开
7 $(this).css({ "color": "white", "font-size": "20px" });
```

## 2.2设置类样式方法

- ```

1  //添加类 addClass()
2  $("div").click(function() {
3      $(this).addClass("current");
4  });
5  // 2. 删除类 removeClass()
6      $("div").click(function() {
7          $(this).removeClass("current");
8      });
9  //3. 切换类 toggleClass()
10     $("div").click(function() {
11         $(this).toggleClass("current");
12     });
```

2.3 类操作与className区别

- 原生 JS 中 className 会覆盖元素原先里面的类名。
- jQuery 里面类操作只是对指定类进行操作，不影响原先的类名。

- ```
1 var one = document.querySelector(".one");
2 one.className = "two"; //会覆盖元素原先里面的类名。
3 $(".one").addClass("two"); //这个addClass相当于追加类名 不影响以前的类名
4 $(".one").removeClass("two");//删除类
```

## 3.1、jquery 效果

- 显示 show () 建议：平时一般不带参数，直接显示隐藏即可。
  - show(毫秒数,fn回调函数)
- 隐藏 hide () 建议：平时一般不带参数，直接显示隐藏即可。
  - hide(毫秒数,fn回调函数)
- 切换toggle () 建议：平时一般不带参数，直接显示隐藏即可。
  - toggle(毫秒数,fn回调函数)

- ```
1   $(function () {
2       $("button")
3       .eq(0)
4       .click(function () {
5           $("div").show(1000, function () {
6               $(this).stop(1000).css({
7                   width: 500,
8                   height: 500,
9               });
10          });
11      });
12      $("button")
13      .eq(1)
14      .click(function () {
15          $("div").hide(1000, function () {});
16      });
17      $("button")
18      .eq(2)
19      .click(function () {
20          $("div").toggle(1000);
21      });
22      // 一般情况下，我们都不加参数直接显示隐藏就可以了
23  });
```

3.2滑动效果

- 下滑动效果slideDown ()
 - slideDown(毫秒数,fn回调函数)

- 上滑动效果slideUp ()
 - slideUp(毫秒数,fn回调函数)
- 切换滑动效果 slideToggle ()
 - slideToggle(毫秒数,fn回调函数)

- ```

1 // 鼠标经过
2 $(".nav>li").mouseover(function() {
3 // $(this) jQuery 当前元素 this不要加引号
4 // // show() 显示元素 hide() 隐藏元素
5 $(this).children("ul").slideDown(200);
6 // });
7 // // 鼠标离开
8 $(".nav>li").mouseout(function() {
9 $(this).children("ul").slideUp(200);
10 // });
11 // 1. 事件切换 hover 就是鼠标经过和离开的复合写法
12 $(".nav>li").hover(function() {
13 $(this).children("ul").slideDown(200);
14 }, function() {
15 $(this).children("ul").slideUp(200);
16 });

```

### 3.3事件切换

- hover([over],out)
  - (1) over:鼠标移到元素上要触发的函数 (相当于mouseenter)
  - (2) out:鼠标移出元素要触发的函数 (相当于mouseleave)
  - (3) 如果只写一个函数, 则鼠标经过和离开都会触发它

- ```

1 // 2. 事件切换 hover 如果只写一个函数, 那么鼠标经过和鼠标离开都会触发这个函数
2     $(".nav>li").hover(function() {
3         // stop 方法必须写到动画的前面
4         $(this).children("ul").stop().slideToggle();
5     });
      
```

3.4动画队列或效果队列

- stop () 停止动画
- fadeIn () 淡入效果 (毫秒数,fn回调函数)
- fadeOut () 淡出效果 (毫秒数,fn回调函数)
- fadeToggle () 淡入淡出切换效果 (毫秒数,fn回调函数)
- fadeTo () 修改透明度 (毫秒数,0-1透明度).

- ```

1 $(function() {
2 $("button").eq(0).click(function() {

```

```

3 // 淡入 fadeIn()
4 $("div").fadeIn(1000);
5 })
6 $("button").eq(1).click(function() {
7 // 淡出 fadeOut()
8 $("div").fadeOut(1000);
9
10 })
11 $("button").eq(2).click(function() {
12 // 淡入淡出切换 fadeToggle()
13 $("div").fadeToggle(1000);
14
15 });
16 $("button").eq(3).click(function() {
17 // 修改透明度 fadeTo() 这个速度和透明度要必须写
18 $("div").fadeTo(1000, 0.5);
19
20 });

```

## 3.5自定义动画

- animate({属性: 属性值}, 毫秒数, 回调函数)

```

1 $(function() {
2 $("button").click(function() { //点击事件
3 $("div").animate({ //为 div 创建动画
4 left: 500, //距离左边500
5 top: 300, //距离上边300
6 opacity: .4, //透明度 0.4
7 width: 500 // 宽度 500
8 }, 500); //500毫秒
9 })
10 })

```

## 4.1 jquery属性操作

- prop () 获取私有属性
  - prop("属性名") 获取属性
  - prop("属性名", 属性值) 设置属性
- attr () 获取 h5 自定义属性
  - attr("属性名") 获取属性
  - attr("属性名", 属性值) 设置属性
- data () 数据缓存 不推荐使用
  - data ("name","value") 附加数据
  - data ("name") 获取数据
- html() 获取 html("n内容")设置元素内容
- text()获取文本内容 text("文本内容") 设置文本内容

- `val()`获取表单值 `val("内容")`设置表单值

```

1 <script>
2 $(function() {
3 //1. element.prop("属性名") 获取元素固有的属性值
4 console.log($("#a").prop("href"));
5 $("#a").prop("title", "我们都挺好");
6 $("#input").change(function() {
7 console.log($("#this").prop("checked"));
8 });
9 // console.log($("#div").prop("index"));
10 // 2. 元素的自定义属性 我们通过 attr()
11 console.log($("#div").attr("index"));
12 $("#div").attr("index", 4);
13 console.log($("#div").attr("data-index"));
14 // 3. 数据缓存 data() 这个里面的数据是存放在元素的内存里面
15 $("#span").data("uname", "andy");
16 console.log($("#span").data("uname"));
17 // 这个方法获取data-index h5自定义属性 第一个 不用写data- 而且返回的是数字型
18 console.log($("#div").data("index"));
19 })
20
21 // 1. 获取设置元素内容 html()
22 console.log($("#div").html());
23 // $("#div").html("123");
24 // 2. 获取设置元素文本内容 text()
25 console.log($("#div").text());
26 $("#div").text("123");
27
28 // 3. 获取设置表单值 val()
29 console.log($("#input").val());
30 $("#input").val("123");
31 </script>

```

## 5.1jQuery遍历元素

- `$("#元素"/dom对象).each(function(i,ele){ })` 主要用DOM处理
- `$.each(对象名, function(i,ele){})` 遍历任何对象。主要用于数据处理，如数组，对象

```

1 $(".p-sum").each(function (i, ele) {
2 money += parseFloat($("#ele").text().substr(1));
3 });
4 $.each(data, function (i, ele)

```

- 创建元素 `$("#<p></p>")`
- 添加元素
  - 内部 父子关系

- append()后面添加
  - prepend() 前面添加
- 外部 兄弟关系
  - after() 元素后面
  - before() 元素前面
- 删除元素
  - remove() 元素本身
  - empty() 元素的子节点 (内容)
  - html("") 清空元素内容
    - empty() 和 html("") 作用等价, 都可以删除元素里面的内容, 只不过 html 还可以设置内容

```

1 $(function() {
2 // 1. 创建元素
3 var li = $("我是后来创建的li");
4 // 2. 添加元素
5 // (1) 内部添加
6 $("ul").append(li); 内部添加并且放到内容的最后面
7 $("ul").prepend(li); // 内部添加并且放到内容的最前面
8
9 // (2) 外部添加
10 var div = $("<div>我是后妈生的</div>");
11 $(".test").after(div);
12 $(".test").before(div);
13 // 3. 删除元素
14 $("ul").remove(); 可以删除匹配的元素 自杀
15 $("ul").empty(); // 可以删除匹配的元素里面的子节点 孩子
16 $("ul").html(""); // 可以删除匹配的元素里面的子节点 孩子
17
18 })

```

## 6.1 jQuery尺寸

- width()/height() 获取元素 宽度/高度
- innerWidth()/innerHeight() 含padding
- outerWidth()/outerHeight()含padding border
- outerWidth(true)/outerHeight(true)含padding border margin

- ```

1  $(function() {
2      // 1. width() / height() 获取设置元素 width和height大小
3      console.log($(".div").width());
4      // $(".div").width(300);
5      // 2. innerWidth() / innerHeight() 获取设置元素 width和height + padding 大小
6      console.log($(".div").innerWidth());
7      // 3. outerWidth() / outerHeight() 获取设置元素 width和height + padding +
border 大小
8      console.log($(".div").outerWidth());
9      // 4. outerWidth(true) / outerHeight(true) 获取设置 width和height + padding +
border + margin
10     console.log($(".div").outerWidth(true));
11 })

```

6.2jQuery位置 (事件使用 scroll)

- offset() 获取被选元素相对于文档的偏移坐标 与父级没有关系
 - offset.top 获取距离文档左侧/顶部的距离 offset().top
 - 可以黄色至偏移 offset({top:10,left:30})
- position() 返回被选元素相对于带有定位的父级偏移坐标 父级没有定位 以文档为准
 - offset.top 获取距离定位父级 左侧/顶部 的距离 position().top
 - 只能读取
- scrollTop()/scrollLeft() 设置或获取元素被卷进去的距离
 - scrollTop() 被卷进去的头部
 - 不跟参数 是 获取 参数为不带单位数字 被卷进去的头部

```

1  <script>
2      $(function () {
3          $(document).scrollTop(100);
4          // 被卷去的头部 scrollTop() / 被卷去的左侧 scrollLeft()
5          // 页面滚动事件
6          var boxTop = $(".container").offset().top; //获取元素到顶部的距离
7          $(window).scroll(function () {
8              //如果滑动距离大于等于 boxTop获取的距离 时 淡入 小于 淡出
9              if ($(document).scrollTop() >= boxTop) {
10                 $(".back").fadeIn();
11             } else {
12                 $(".back").fadeOut();
13             }
14         });
15         // 返回顶部
16         $(".back").click(function () {
17             // $(document).scrollTop(0);
18             $("body, html").stop().animate({
19                 scrollTop: 0,
20             });
21             // $(document).stop().animate({
22             //     scrollTop: 0
23             // }); 不能是文档而是 html和body元素做动画

```

```
24     });
25     });
26 </script>
```

7.3 jQuery事件

mouseover、mouseout、blur、focus、change、keydown、keyup、resize、scroll

- `$("#div").click(function())` 单个事件
- `$("#div").on(事件名, "委托事件 (子元素)", function())`
- 事件切换 `trigger("click")` / `triggerHandle("click")` 不会触发元素默认行为

- ```
1 $(function() {
2 // 1. 单个事件注册
3 $("#div").click(function() {
4 $(this).css("background", "purple");
5 });
6 $("#div").mouseenter(function() {
7 $(this).css("background", "skyblue");
8 });
9
10 // 2. 事件处理on
11 (1) on可以绑定1个或者多个事件处理程序
12 $("#div").on({
13 mouseenter: function() {
14 $(this).css("background", "skyblue");
15 },
16 click: function() {
17 $(this).css("background", "purple");
18 },
19 mouseleave: function() {
20 $(this).css("background", "blue");
21 }
22 });
23 $("#div").on("mouseenter mouseleave", function() {
24 $(this).toggleClass("current");
25 });
26 (2) on可以实现事件委托 (委派)
27 // $("#ul li").click();
28 $("#ul").on("click", "li", function() {
29 alert(11);
30 });
31 // click 是绑定在ul 身上的, 但是 触发的对象是 ul 里面的小li
32 // (3) on可以给未来动态创建的元素绑定事件
33 $("ol li").click(function() {
34 alert(11);
35 })
36 $("ol").on("click", "li", function() {
37 alert(11);
38 })
39 var li = $("我是后来创建的");
```

```

40 $("ol").append(li);
41 })
42 // 1. 事件解绑 off
43
44 $("div").off(); // 这个是解除了div身上的所有事件
45
46 $("div").off("click"); // 这个是解除了div身上的点击事件
47
48 $("ul").off("click", "li");
49
50 // 2. one() 但是它只能触发事件一次
51
52 $("p").one("click", function() {
53
54 alert(11);
55
56 })
57 // 自动触发事件
58 //1. 元素.事件()
59 $("div").click();//会触发元素的默认行为
60 //2. 元素.trigger("事件")
61 $("div").trigger("click");//会触发元素的默认行为
62 // 3. 元素.triggerHandler("事件") 就是不会触发元素的默认行为
63 $("div").triggerHandler("click");
64 </script>

```

## 7.4事件对象

- \$("div").on(事件名, "委托事件 (子元素)", function(event/e){})
  - 阻止默认行为: event.preventDefault() 或者 return false
  - 阻止冒泡: event.stopPropagation()

```

1 <script>
2 $(function() {
3 $(document).on("click", function() {
4 console.log("点击了document");
5
6 })
7 $("div").on("click", function(event) {
8 // console.log(event);
9 console.log("点击了div");
10 event.stopPropagation();
11 })
12 })
13 </script>

```

## 7.5其他方法

- 拷贝
  - \$.extend(true, targetObj, obj); 深拷贝 如果里面有不冲突的属性,会合并到一起

- \$.extend(targetObj, obj);浅拷贝 地址给他 会修改原理的数据

- ```
1 // 1. 浅拷贝把原来对象里面的复杂数据类型地址拷贝给目标对象
2     targetObj.msg.age = 20;
3 // 2. 深拷贝把里面的数据完全复制一份给目标对象 如果里面有不冲突的属性,会合并到一起
4     $.extend(true, targetObj, obj);
```

7.6 多库共存

- // 1. 如果\$ 符号冲突 我们就使用 jQuery
- // 2. 让jquery 释放对\$ 控制权 让用自己决定
 - var suibian = jQuery.noConflict();

```
1     <script>
2         $(function () {
3             function $(ele) {
4                 return document.querySelector(ele);
5             }
6             console.log($("#div"));
7             // 1. 如果$ 符号冲突 我们就使用 jQuery
8             jQuery.each();
9             // 2. 让jquery 释放对$ 控制权 让用自己决定
10            var suibian = jQuery.noConflict();
11            console.log(suibian("span"));
12            suibian.each();
13        });
14    </script>
```

AJAX 请求 处理 响应

1.请求

- \$.get(url, [data], fn) []可写可不写 参数{id: 1}
- \$.post(url, [data], fn)
- \$.ajax()
 - \$.ajax({
type: ", // 请求的方式, 例如GET 或 POST
url: ", // 请求的 URL 地址
data: { }, // 这次请求要携带的数据
success: function(res) { } // 请求成功之后的回调函数
})

2.模板引擎

- form表单
-

属性	值	描述
action	URL地址	规定当提交表单时，向何处发送表单数据
method	get或post	规定以何种方式把表单数据提交到 action URL
enctype	application/x-www-form-urlencoded multipart/form-data text/plain	规定在发送表单数据之前如何对其进行编码
target	_blank _self _parent _top <i>iframe</i>	规定在何处打开 action URL

- method
 - GET：通过URL地址（一般不用）
 - POST：方式适合用来提交大量的、复杂的、或包含文件上传的数据。
- enctype
 - 文件上传的操作时，必须将enctype 的值设置为multipart/form-data

2.1 监听表单提交事件

- submit()方法

- ```

1 $('#form1').submit(function(e) {
2 alert('监听到了表单的提交事件')
3 })
4
5 $('#form1').on('submit', function(e) {
6 alert('监听到了表单的提交事件')
7 })
8
```

## 2.2 阻止表单默认提交行为

- event.preventDefault() 函数，来阻止表单的提交和页面的跳转

●

## 2.3 快速获取表单中的数据

- jQuery提供了 `serialize()`函数
- 注意：在使用`serialize()`函数快速获取表单数据时，必须为每个表单元素添加`name`属性

●

## 2.4.模板引擎art-template

- 语法:
  - {{ }}变量输出{{value}} {{obj.key}} {{obj['key']}} {{a ?b : c}} {{a | b}} {{a +b}}
  - {{@value }} 原文输出 才能保证HTML标签
  - 条件输出
    - {{ifvalue}} 按需输出的内容 {{/if}}
    - {{if v1}}按需输出的内容 {{else if v2}}按需输出的内容 {{/if}}
  - 循环输出
    - {{each arr}} {{/each}}
  - 过滤器
    - {{value | **filterName**}}
    - `template.defaults.imports.filterName = function(value){/return 处理的结果/}`

## 2.5实现原理

- `exec()` 检索字符串中正则表达式的匹配 无返回null
- `replace (旧值, 新)` 函数
- `while` 循环`replace` 实现 多次
- `replace` 替换为真值

```

1 //1.基本语法
2 var str = 'hello'
3 var pattern = /o/
4 // 输出的结果["o", index: 4, input: "hello", groups: undefined]
5 console.log(pattern.exec(str))
6 //2.分组
7 var str = '<div>我是{{name}}</div>'//字符串
8 var pattern = /{{([a-zA-Z]+)}}// 正则 a-zA-Z字母
9 var patternResult = pattern.exec(str) // 提取 str中的匹配字母
10 console.log(patternResult)
11 // 得到 name 相关的分组信息
12 // [{"{{name}}", "name", index: 7, input: "<div>我是{{name}}</div>", groups:
13 0 1 2 3
14 4
15 //undefined]
16 var str = '<div>我是{{name}}</div>'
17 var pattern = /{{([a-zA-Z]+)}}/
18 var patternResult = pattern.exec(str)
19 str = str.replace(patternResult[0], patternResult[1])
20 console.log(str)
21 // 1 替换 0
22 // 输出的内容是: <div>我是name</div>
23 //多次 replace
24 var str = '<div>{{name}}今年{{ age }}岁了</div>'
25 var pattern = /\s*([a-zA-Z+)\s*)/
26 var patternResult = null //定义一个 变量
27 while((patternResult = pattern.exec(str))) { //将str 提取字母
28 str = str.replace(patternResult[0], patternResult[1]) // 替换
29 }
30 console.log(str) // 输出 <div>name今年age岁了</div>
31 //replace 替换为真值
32 var data = { name: '张三', age: 20 }
33 var str = '<div>{{name}}今年{{ age }}岁了</div>'
34 var pattern = /\s*([a-zA-Z+)\s*)/
35 var patternResult = null
36 while ((patternResult = pattern.exec(str))) {
37 str = str.replace(patternResult[0], data[patternResult[1]])
38 }

```

## 2.6模板使用

```

1 //导入模版
2 <script src="./js/template.js"></script>
3 //定义 格式 type="text/html"
4 <script type="text/html" id="tpl-user">
5 <div>姓名: {{name}}</div>
6 <div>年龄: {{ age }}</div>
7 <div>性别: {{ gender}}</div>
8 <div>住址: {{address }}</div>
9 </script>
10 // 定义数据

```

```

11 var data = { name: 'zs', age: 28, gender: '男', address: '北京顺义马坡' }
12
13 // 调用模板引擎
14 var htmlStr = template('tpl-user', data)
15
16 // 渲染HTML结构
17 document.getElementById('user-box').innerHTML = htmlStr

```

## 3.AJAX加强

### XHR

- xhr发起get/post请求
- GET请求步骤：
  - ①创建 xhr 对象
  - ②调用 xhr.open() 函数
  - ③调用 xhr.send() 函数
  - ④监听 xhr.onreadystatechange 事件
- POST步骤：
  - ①创建 xhr 对象
  - ②调用 xhr.open() 函数
  - ③设置 Content-Type 属性（固定写法）
  - ④调用 xhr.send() 函数，同时指定要发送的数据
  - ⑤监听 xhr.onreadystatechange 事件
- responseText 响应数据

- ```

1 // 1. 创建 XHR 对象
2 var xhr = new XMLHttpRequest()
3 // 2. 调用 open 函数, 指定 请求方式 与 URL地址  ?    &拼接
4 xhr.open('GET', 'http://www.liulongbin.top:3006/api/getbooks?id=1&id=2')
5 // 3. 调用 send 函数, 发起 Ajax 请求
6 xhr.send()
7 // 4. 监听 onreadystatechange 事件
8 xhr.onreadystatechange = function() {
9     // 4.1 监听 xhr 对象的请求状态 readyState ; 与服务器响应的状态 status
10    if (xhr.readyState === 4 && xhr.status === 200) {
11        // 4.2 打印服务器响应回来的数据
12        console.log(xhr.responseText)
13    }
14 }
15 // 1. 创建 xhr 对象
16 var xhr = new XMLHttpRequest()
17 // 2. 调用 open()
18 xhr.open('POST', 'http://www.liulongbin.top:3006/api/addbook')
19 // 3. 设置 Content-Type 属性（固定写法）
20 xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
21 // 4. 调用 send(), 同时将数据以查询字符串的形式, 提交给服务器
22 xhr.send('bookname=水浒传&author=施耐庵&publisher=天津图书出版社')

```

```

23 // 5. 监听 onreadystatechange 事件
24 xhr.onreadystatechange = function() {
25     if (xhr.readyState === 4 && xhr.status === 200) {
26         console.log(xhr.responseText)
27     }
28 }
29

```

XML

- XML的英文全称是ExtensibleMarkupLanguage，即可扩展标记语言。因此，XML和HTML类似，也是一种标记语言。
- XML和HTML区别
 - XML和HTML虽然都是标记语言，但是，它们两者之间没有任何的关系。
 - HTML 被设计用来描述网页上的内容，是网页内容的载体
 - XML 被设计用来传输和存储数据，是数据的载体
- 2新特性
 - xhr.timeout= 3000 请求时限
 - FormData对象 快速获取表单数据
 - append()方法 尾部添加数据

```

1 // 获取表单元素
2 var form = document.querySelector('#form1')
3 // 监听表单元素的 submit 事件
4 form.addEventListener('submit', function(e) {
5     e.preventDefault() //阻止 表单提交后跳转页面
6     // 根据 form 表单创建 FormData 对象，会自动将表单数据填充到 FormData 对象中
7     var fd = new FormData(form) //获取表单的数据
8     var xhr = new XMLHttpRequest() //定义实例化对象
9     xhr.open('POST', 'http://www.liulongbin.top:3006/api/formdata')
10    xhr.send(fd) //添加数据
11    xhr.onreadystatechange = function() {}
12 })

```

JSON

- JSON就是 Javascript 对象和数组的字符串表示法，JSON 的本质是字符串。对象和数组
 - 对象结构：对象结构在JSON中表示为 key必须使用英文的双引号包裹的字符串，value的数据类型可以是数字、字符串、布尔值、null、数组、对象6种类型。
 - 数组结构：数组结构在JSON中表示为。数组中数据的类型可以是数字、字符串、布尔值、null、数组、对象6种类型。
 - JSON语法注意事项
 - ①属性名必须使用双引号包裹
 - ②字符串类型的值必须使用双引号包裹

- ③JSON 中不允许使用单引号表示字符串
 - ④JSON 中不能写注释
 - ⑤JSON 的最外层必须是对象或数组格式
 - ⑥不能使用 undefined或函数作为JSON的值
 - JSON 的作用：在计算机与网络之间存储和传输数据。
 - JSON 的本质：用字符串来表示javascript对象数据或数组数据
- JSON.parse() 从 JSON 字符串转换为 JS 对象
 - JSON.stringify() 从 JS 对象转换为JSON 字符串
 - 把数据对象转换为字符串的过程，叫做序列化，例如：调用 JSON.stringify()函数的操作，叫做 JSON 序列化。
 - 把字符串转换为数据对象的过程，叫做反序列化，例如：调用JSON.parse()函数的操，叫做JSON反序列化。

上传文件

- ```

1 //ui结构
2 // <!-- 1. 文件选择框 -->
3 <input type="file" id="file1" />
4 // <!-- 2. 上传按钮 -->
5 <button id="btnUpload">上传文件</button>
6

7
8 <!-- 3. 显示上传到服务器上的图片 -->
9
10
11 // 2验证是否选择了文件
12 //获取上传文件的按钮
13 var btnUpload = document.querySelector('#btnUpload')
14 // 2. 为按钮添加 click 事件监听
15 btnUpload.addEventListener('click', function() {
16 // 3. 获取到选择的文件列表
17 var files = document.querySelector('#file1').files
18 if (files.length <= 0) {
19 return alert('请选择要上传的文件! ')
20 }
21 // 1. 创建 FormData 对象
22 var fd = new FormData()
23 // 2. 向 FormData 中追加文件
24 fd.append('avatar', files[0]) //不为空的话 files数组里就有 东西 把索引为0追加
25 // 1. 创建 xhr 对象
26 var xhr = new XMLHttpRequest()
27 // 2. 调用 open 函数，指定请求类型与URL地址。其中，请求类型必须为 POST
28 xhr.open('POST', 'http://www.liulongbin.top:3006/api/upload/avatar')
29 // 3. 发起请求
30 xhr.send(fd)
31 xhr.onreadystatechange = function() {
32 if (xhr.readyState === 4 && xhr.status === 200) {
33 var data = JSON.parse(xhr.responseText)
34 if (data.status === 200) { // 上传文件成功
35
36 // 将服务器返回的图片地址，设置为 标签的 src 属性

```

```

36 document.querySelector('#img').src = 'http://www.liulongbin.top:3006' + data.url
37 } else { // 上传文件失败
38 console.log(data.message)
39 }
40 }
41 }

```

## jquery上传文件

- ```

1 $(document).ajaxStart(function() {
2     $('#loading').show()
3 })
4 // 自 jQuery 版本 1.8 起, 该方法只能被附加到文档
5 $(document).ajaxStop(function() {
6     $('#loading').hide()
7 })
8
9 $('#btnUpload').on('click', function() {
10     // 1. 将 jQuery 对象转化为 DOM 对象, 并获取选中的文件列表
11     var files = $('#file1')[0].files
12     // 2. 判断是否选择了文件
13     if (files.length <= 0) {
14         return alert('请选择图片后再上传! ')
15     }
16     var fd = new FormData()
17     fd.append('avatar', files[0])
18 })
19 $.ajax({
20     method: 'POST',
21     url: 'http://www.liulongbin.top:3006/api/upload/avatar',
22     data: fd,
23     // 不修改 Content-Type 属性, 使用 FormData 默认的 Content-Type 值
24     contentType: false,
25     // 不对 FormData 中的数据进行 url 编码, 而是将 FormData 数据原样发送到服务器
26     processData: false,
27     success: function(res) {
28         console.log(res)
29     }
30 })
31

```

axios

- axios 发起get/post/ajax请求的语法

- ```

1 var url = 'http://www.liulongbin.top:3006/api/get'
2 // 请求的参数对象
3 var paramsObj = { name: 'zs', age: 20 }
4 // 调用 axios.get() 发起 GET 请求
5
6 axios.get(url, { params: paramsObj }).then(function(res) {

```

```

6 // res.data 是服务器返回的数据
7 var result = res.data
8 console.log(res)
9 })
10
11 // 请求的 URL 地址
12 var url = 'http://www.liulongbin.top:3006/api/post'
13 // 要提交到服务器的数据
14 var dataObj = { location: '北京', address: '顺义' }
15 // 调用 axios.post() 发起 POST 请求
16 axios.post(url, dataObj).then(function(res) {
17 // res.data 是服务器返回的数据
18 var result = res.data
19 console.log(result)
20 })
21 axios({
22 method: '请求类型',
23 url: '请求的URL地址',
24 data: { /* POST数据 */ },
25 params: { /* GET参数 */ }
26 }) .then(callback)

```

## 同源

- 协议、域名、端口都相同的 同源 反之跨域

### 同源策略

- 浏览器提供的一个安全功能 a网站的JavaScript不能与b网站的资源交互

## JSONP

- 解决 跨域数据访问的问题 支持JSONP
- 通过通过<script>标签的src属性，请求跨域的数据接口，并通过函数调用的形式，接收跨域接口响应回来的数据。

```

1 <script>
2 function success(data) {
3 console.log('获取到了data数据: ')
4 console.log(data)
5 }
6 </script>
7 //实现jsonp
8 <script src="http://ajax.frontend.itheima.net:3006/api/jsonp?
9 callback=success&name=zs&age=20"></script>

```

## 自定义参数及回调

- jsonp: 参数名称 默认callback
- jsonpCallack:回调函数名称 jQueryxxx



## 防抖

- 用户在输入框中连续输入一串字符时，可以通过防抖策略，只在输入完后，才执行查询的请求，这样可以有效减少请求次数，节约请求资源；

```
1 var timer = null // 1. 防抖动的 timer
2
3 function debounceSearch(keywords) { // 2. 定义防抖的函数
4 timer = setTimeout(function() {
5 // 发起 JSONP 请求
6 getSuggestList(keywords)
7 }, 500)
8 }
9
10 $('#ipt').on('keyup', function() { // 3. 在触发 keyup 事件时，立即清空 timer
11 clearTimeout(timer)
12 // ...省略其他代码
13 debounceSearch(keywords)
14 })
15
```

## 节流

- 防抖：如果事件被频繁触发，防抖能保证只有最有一次触发生效！前面 N 多次的触发都会被忽略！
- 节流：如果事件被频繁触发，节流能够减少事件触发的频率，因此，节流是有选择性地执行一部分事件！

- ```
1  $(function() {
2      var angel = $('#angel')
3      var timer = null // 1.预定义一个 timer 节流阀
4      $(document).on('mousemove', function(e) {
5          if (timer) { return } // 3.判断节流阀是否为空，如果不为空，则证明距离上次执行间隔不足16毫
          秒
6          timer = setTimeout(function() {
7              $(angel).css('left', e.pageX + 'px').css('top', e.pageY + 'px')
8              timer = null // 2.当设置了鼠标跟随效果后，清空 timer 节流阀，方便下次开启延时器
9          }, 16)
10     })
11 })
12
```

HTTP协议加强

- 主体 内容 方式 三要素
- 通信协议：是指通信的双方完成通信所必须遵守的规则和约定。通俗的理解：通信双方采用约定好的格式来发送和接收消息，这种事先约定好的通信格式，就叫做通信协议。
- 客户端与服务器之间要实现网页内容的传输，则通信的双方必须遵守网页内容的传输协议。
- 网页内容又叫做超文本，因此网页内容的传输协议又叫做超文本传输协议（HyperText Transfer Protocol），简称 HTTP协议。

请求消息

- HTTP请求消息又叫做HTTP请求报文。
- HTTP请求消息由请求行（requestline）、请求头部（header）、空行和 请求体 4个部分组成。
- 请求行由请求方式、URL和 HTTP 协议版本 3个部分组成，他们之间使用空格隔开。
- 请求头部
 - User-Agent 用来说明当前是什么类型的浏览器；
 - Content-Type 用来描述发送到服务器的数据格式；
 - Accept 用来描述客户端能够接收什么类型的返回内容；
 - Accept-Language 用来描述客户端期望接收哪种人类语言的文本内容。
 - 请求头部由多行键/值对 组成，每行的键和值之间用英文的冒号分隔。
- 空行，用来分隔请求头部与请求体。
- 请求体中存放的，是要通过 POST 方式提交到服务器的数据
-

响应消息

- 响应消息就是服务器响应给客户端的消息内容，也叫作响应报文。
-
- 状态行
 - 状态行由 HTTP协议版本、状态码和状态码的描述文本3 个部分组成，他们之间使用空格隔开；
 - 响应头部用来描述服务器的基本信息。响应头部由多行 键/值对 组成，每行的键和值之间用英文的冒号分隔。
 -
 - 响应消息中的空行，用来分隔响应头部与响应体。
 - 响应体中存放的，是服务器响应给客户端的资源内容。

请求方法

- 用来表明要对服务器上的资源执行的操作。最常用的请求方法是 GET 和 POST。

响应状态码

5.2 HTTP响应状态码的组成及分类

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字用来对状态码进行细分。

HTTP 状态码共分为 5 种类型：

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作（实际开发中很少遇到 1** 类型的状态码）
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

1. 2** 成功相关的响应状态码

2** 范围的状态码，表示服务器已成功接收到请求并进行处理。常见的 2** 类型的状态码如下：

状态码	状态码英文名称	中文描述
200	OK	请求成功。一般用于 GET 与 POST 请求
201	Created	已创建。成功请求并创建了新的资源，通常用于 POST 或 PUT 请求

2. 3** 重定向相关的响应状态码

3** 范围的状态码，表示表示服务器要求客户端重定向，需要客户端进一步的操作以完成资源的请求。常见的 3** 类型的状态码如下：

状态码	状态码英文名称	中文描述
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302	Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源（响应消息中不包含响应体）。客户端通常会缓存访问过的资源。

3. 4** 客户端错误相关的响应状态码

4** 范围的状态码，表示客户端的请求有非法内容，从而导致这次请求失败。常见的 4** 类型的状态码如下：

状态码	状态码英文名称	中文描述
400	Bad Request	1、语义有误，当前请求无法被服务器理解。除非进行修改，否则客户端不应该重复提交这个请求。 2、请求参数有误。
401	Unauthorized	当前请求需要用户验证。
403	Forbidden	服务器已经理解请求，但是拒绝执行它。
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。
408	Request Timeout	请求超时。服务器等待客户端发送的请求时间过长，超时。

面向对象编程 和ES6

- 两大编程思想 面向过程 面向对象（多人）
- 面向对象编程 oop 封装 继承 多态

1、ES6中的类和对象

1.1对象

- 对象是有属性和方法组成 一切皆为对象
 - 属性：事物的特征
 - 方法：事物的行为

1.2 类class

- 类抽象了对象的公共部分，它泛指某一大类（class） 对象特指某一个，通过类实例化一个具体的对象
- 创建类
 - `class name{ }`
- 创建实例化对象 类必须使用 `new` 实例化对象
 - `let xx=new name ()`
- 构造方法 需要 `constructor ()` 方法
- 类添加方法 `名字(){}`
- 调用 创建实例化对象 `xx.方法名()`

- ```

1 class name { //创建了一个 name 类
2 let name, age
3 constructor(name,age,){
4 this.name=name
5 this.age=age
6 }
7 say(){
8 console.log("我会说话")
9 }
10 }
11 let ldh=new name("刘德华", 18)
12 ldh.say()
```

## 类的继承

- 关键字 extends Son extends Father
- 关键字 super
  - super关键字 用于访问和调用对象父类上的函数。可以调用父类的构造函数，也可以调用父类的普通函数。

- ```

1  class Father {
2      constructor(name,age,){
3          this.name=name
4          this.age=age
5      }
6      say() {
7          return '我是爸爸';
8      }
9  }
10 class Son extends Father { // 这样子类就继承了父类的属性和方法
11     super(name,age)
12     say() {
13         // super.say() super 调用父类的方法
14         return super.say() + '的儿子';
15     }
16 }
17 var damao = new Son();
18 console.log(damao.say());
```

构造函数

- 构造函数是一种特殊的函数，主要用来初始化对象，即为对象成员变量赋初始值，它总与 new 一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。

- 在JS 中，使用构造函数时要注意以下两点：
 - 1.构造函数用于创建某一类对象，其首字母要大写
 - 2.构造函数要和 new 一起使用才有意义
- new 在执行时会做四件事情：
 - ①在内存中创建一个新的空对象。
 - ②让 this 指向这个新的对象。
 - ③执行构造函数里面的代码，给这个新对象添加属性和方法。
 - ④返回这个新对象（所以构造函数里面不需要 return ）。

构造函数（静态成员和实例成员）

- 静态成员：在构造函数本上添加的成员称为静态成员，只能由构造函数本身来访问
- 实例成员：在构造函数内部创建的对象成员称为实例成员，只由实例化的对象来访问

构造函数原型

- 构造函数通过原型分配的函数是所有对象所共享的
- JavaScript 规定，每一个构造函数都有一个prototype 属性，指向另一个对象。注意这个prototype就是一个对象，这个对象的所有属性和方法，都会被构造函数所拥有。
- 原型对象里面放的是方法，这个方法里面的this 指向的是这个方法的调用者，也就是这个实例对象
- 数组和字符串内置对象不能给原型对象覆盖操作 Array.prototype = {}，只能是 Array.prototype.xxx =function(){} 的方式。

继承

- 我们可以通过构造函数+原型对象模拟实现继承，被称为组合继承。

2.1call()

- 调用这个函数, 并且修改函数运行时的 this 指向 fn.call(this, 其他参数)
- 通过 call() 把父类型的 this 指向子类型的 this，这样就可以实现子类型继承父类型的属性。

- ```

1 // 父类
2 function Person(name, age, sex) {
3 this.name = name;
4 this.age = age;
5 this.sex = sex;
6 }
7 // 子类
8 function Student(name, age, sex, score) {
9 Person.call(this, name, age, sex); // 此时父类的 this 指向子类的 this, 同时调用这个
 函数
10 this.score = score;
11 }
12 var s1 = new Student('zs', 18, '男', 100);
13 console.dir(s1);

```

## 函数

- 调用方式 和 this指向

| 调用方式   | this指向                |
|--------|-----------------------|
| 普通函数调用 | window                |
| 构造函数调用 | 实例对象 原型对象里面的方法也指向实例对象 |
| 对象方法调用 | 该方法所属对象               |
| 事件绑定方法 | 绑定事件对象                |
| 定时器函数  | window                |
| 立即执行函数 | window                |

## 三个注意点:

1. 在 ES6 中类没有变量提升, 所以必须先定义类, 才能通过类实例化对象.
2. 类里面的共有属性和方法一定要加this使用.
3. 类里面的this指向问题.
4. constructor 里面的this指向实例对象, 方法里面的this 指向这个方法的调用者

- 改变this指向 三种方法: call方法 apply方法 . bind 方法
  - 相同点: 都可以改变函数内部的this指向.
  - 区别点:
    - 1.call 和 apply 会调用函数, 并且改变函数内部this指向..
    - 2.call 和 apply传递的参数不一样,call 传递参数 aru1,aru2..形式 apply 必须数组形式[arg]
    - 3.bind 不会调用函数, 可以改变函数内部this指向.
  - 主要应用场景:
    - 1.call 经常做继承.
    - 2.apply经常跟数组有关系. 比如借助于数学对象实现数组最大值最小值
    - 3.bind 不调用函数,但是还想改变this指向. 比如改变定时器内部的this指向.

## S5的严格模式

- 在所有语句之前放一个特定语句"use strict"; (或'use strict';)

## 闭包

- 闭包 (closure) 指有权访问另一个函数作用域中变量的函数。
  - 作用: 延伸变量的作用范围
  - 简单理解就是, 一个作用域可以访问另外一个函数内部的局部变量

## 递归

- 一个函数在内部可以调用其本身 必须要加退出条件 return。

## 浅拷贝和深拷贝

- 1.浅拷贝只是拷贝一层, 更深层次对象级别的只拷贝引用.
- 2.深拷贝拷贝多层, 每一级别的数据都会拷贝.
- 3.Object.assign(target, ...sources) es6 新增方法可以浅拷贝

## 正则表达式

- var 变量名 = /表达式/
- 表达式的变量名.test (str) 返回 true / false
- ^谁开始 \$ 谁结束/[abc]/ 包含a/b/c [a-z] 范围符
- [^a]取反 边界符在外边
- \*0/ 多次 +一次 /多次? 0/一次 {n} n次 {n, }n次 /多次 {n, m} n-m次[

•

| 预定义类 | 说明                                     |
|------|----------------------------------------|
| \d   | 匹配0-9之间的任一数字, 相当于[0-9]                 |
| \D   | 匹配所有0-9以外的字符, 相当于 [^0-9]               |
| \w   | 匹配任意的字母、数字和下划线, 相当于[A-Za-z0-9_]        |
| \W   | 除所有字母、数字和下划线以外的字符, 相当于 [^A-Za-z0-9_]   |
| \s   | 匹配空格 (包括换行符、制表符、空格符等), 相等于[\t\r\n\v\f] |
| \S   | 匹配非空格的字符, 相当于 [^\t\r\n\v\f]            |

- replace替换
  - 1.第一个参数: 被替换的字符串 或者 正则表达式
  - 2.第二个参数: 替换为的字符串
  - 3.返回值是一个替换完毕的新字符串
- switch(也称为修饰符) 按照什么样的模式来匹配. 有三种值:
  - lg: 全局匹配
  - li: 忽略大小写
  - lgi: 全局匹配 + 忽略大小写
  - /表达式/[switch]

## ES5新增方法

- 数组方法
  - forEach()数组遍历
  - map()
  - filter() 数组筛选
  - some() 查找数组
  - every();
  - 1. filter 也是查找满足条件的元素 返回的是一个数组 而且是把所有满足条件的元素返回回来



- 1. some 也是查找满足条件的元素是否存在 返回的是一个布尔值 如果查找到第一个满足条件的元素就终止循环

- ```
1 // forEach 迭代(遍历) 数组
2 var arr = [1, 2, 3];
3 var sum = 0;
4 arr.forEach(function(value, index, array) {
5     console.log('每个数组元素' + value);
6     console.log('每个数组元素的索引号' + index);
7     console.log('数组本身' + array);
8     sum += value;
9 })
10 console.log(sum);
11
12 // filter 筛选数组
13 var arr = [12, 66, 4, 88, 3, 7];
14 // var newArr = arr.filter(function (value, index) {
15 //     // return value >= 20;
16 //     return value % 2 === 0;
17 // });
18 let newArr = arr.filter((value) => value % 2 === 0);
19 console.log(newArr);
20 //some查找
21 var arr1 = ['red', 'pink', 'blue'];
22 var flag1 = arr1.some(function(value) {
23     return value == 'pink';
24 });
25 console.log(flag1);
26
```

- 字符串方法
 - str.trim() 删除两端空白字符
- 对象方法
 - Object.keys(obj) 获取所有属性

ES6

- ECMAScript是由Ecma国际通过ECMA-262标准化的脚本程序设计语言
- 从 ES6 开始，每年发布一个版本，版本号比年份最后一位大 1

ES6新特性

1.1let关键字

- let 关键字用来声明变量，使用 let 声明的变量有几个特点：
 - 1) 不允许重复声明
 - 2) 块级级作用域
 - 3) 不存在变量提升

- 4) 不影响作用域链
- 应用场景：以后声明变量使用 let 就对了

1.2 const 关键字

- 声明常量，const 声明有以下特点
- 1) 声明必须赋初始值
- 2) 标识符一般为大写
- 3) 不允许重复声明
- 4) 值不允许修改
- 5) 块级作用域
- 注意: 对象属性修改和数组元素变化不会出发 const 错误
- 应用场景: 声明对象类型使用 const, 非对象类型声明选择 let

1.3 变量的解构赋值

- ```
1 //数组的解构赋值
2 const arr = ['张学友', '刘德华', '黎明', '郭富城'];
3 let [zhang, liu, li, guo] = arr;
4 //复杂解构 对象
5 let wangfei = {
6 name: '王菲',
7 age: 18,
8 songs: ['红豆', '流年', '暧昧', '传奇'],
9 history: [
10 {name: '窦唯'},
11 {name: '李亚鹏'},
12 {name: '谢霆锋'}
13]
14 };
15 let {songs: [one, two, three], history: [first, second, third]} =
16 wangfei;
17
18 let star = '王宁';
19 let result = `${star}在前几年离开了开心麻花`;
```

## 1.4 模板字符串

- 反引号 可以使用 \${xxx} 形式输出变量

## 1.5 简化对象写法

- ES6 允许在大括号里面，直接写入变量和函数，作为对象的属性和方法。

- ```

1 //属性和方法简写
2 let atguigu = {
3   name,
4   slogan,
5   improve,
6   change() {
7     console.log('可以改变你')
8   }
9 };

```

1.6 箭头函数

- 箭头函数的注意点:
 - 1) 如果形参只有一个, 则小括号可以省略
 - 2) 函数体如果只有一条语句, 则花括号可以省略, 函数的返回值为该条语句的执行结果
 - 3) 箭头函数 this 指向声明时所在作用域下 this 的值0
 - 4) 箭头函数不能作为构造函数实例化
 - 5) 不能使用 arguments (构造函数)
 - 箭头函数不会更改 this 指向, 用来指定回调函数会非常合适

- ```

1 //箭头函数
2 item => item % 2 === 0)
3 //不用函数
4 const result = arr.filter(function(item){
5 if(item % 2 === 0){
6 return true;
7 }else{
8 return false;
9 }
10 });

```

## 1.7 rest 参数

- ...args 用来代替 arguments 用于获取函数的实参
- rest 参数必须是最后一个形参
- rest 参数非常适合不定个数参数函数的场景

- ```

1 /**
2  * 作用与 arguments 类似
3  */
4 function add(...args){
5   console.log(args);
6 }
7 add(1,2,3,4,5);
8 /**
9  * rest 参数必须是最后一个形参
10  */
11 function minus(a,b,...args){

```

```
12 console.log(a,b,args);
13 }
14 minus(100,1,2,3,4,5,19);
```

1.8spread扩展运算符

- 数组合并
- 数组的克隆
- 伪数组转真正的数组

- ```
1 //1. 数组的合并 情圣 误杀 唐探
2 const kuaizi = ['王太利','肖央'];
3 const fenghuang = ['曾毅','玲花'];
4 const zuixuanxiaopingguo = kuaizi.concat(fenghuang);
5 const zuixuanxiaopingguo = [...kuaizi, ...fenghuang];
6 console.log(zuixuanxiaopingguo);
7
8 //2. 数组的克隆
9 const sanzhihua = ['E','G','M'];
10 const sanyecao = [...sanzhihua];// ['E','G','M']
11 console.log(sanyecao);
12
13 //3. 将伪数组转为真正的数组
14 const divs = document.querySelectorAll('div');
15 const divArr = [...divs];
16 console.log(divArr);// arguments
```

## 1.9Symbol

- 独一无二的值
- 

### 2.9.1.Symbol 基本使用

ES6 引入了一种新的原始数据类型 **Symbol**，表示独一无二的值。它是 JavaScript 语言的第七种数据类型，是一种类似于字符串的数据类型。

**Symbol 特点**

- 1) Symbol 的值是唯一的，用来解决命名冲突的问题
- 2) Symbol 值不能与其他数据进行运算
- 3) Symbol 定义的对象属性不能使用 `for...in` 循环遍历，但是可以使用 `Reflect.ownKeys` 来获取对象的所有键名

```

// 创建 Symbol
let s1 = Symbol();
console.log(s1, typeof s1);

// 添加标识的 Symbol
let s2 = Symbol('尚硅谷');

let s2_2 = Symbol('尚硅谷');
console.log(s2 === s2_2);

// 使用 Symbol for 定义
let s3 = Symbol.for('尚硅谷');

let s3_2 = Symbol.for('尚硅谷');
console.log(s3 === s3_2);

```

## 1.10 迭代器

- 需要自定义遍历数据的时候，要想到迭代器。
- 工作原理
  - a) 创建一个指针对象，指向当前数据结构的起始位置
  - b) 第一次调用对象的 next 方法，指针自动指向数据结构的第一个成员
  - c) 接下来不断调用 next 方法，指针一直往后移动，直到指向最后一个成员
  - d) 每调用 next 方法返回一个包含 value 和 done 属性的对象

```

1 <script>
2 // 声明一个数组
3 const xiyou = ['唐僧', '孙悟空', '猪八戒', '沙僧'];
4 // 使用 for...of 遍历数组
5 for(let v of xiyou){
6 console.log(v);
7 }
8 let iterator = xiyou[Symbol.iterator](); // 迭代器 iterator()
9 // 调用对象的 next 方法
10 console.log(iterator.next());
11 console.log(iterator.next());
12 console.log(iterator.next());
13 console.log(iterator.next());
14 console.log(iterator.next());
15 </script>

```

## 1.11 生成器

- 代码说明：
  - 1) \* 的位置没有限制
  - 2) 生成器函数返回的结果是迭代器对象，调用迭代器对象的 next 方法可以得到 yield 语句后的值
  - 3) yield 相当于函数的暂停标记，也可以认为是函数的分隔符，每调用一次 next 方法，执行一段代码
  - 4) next 方法可以传递实参，作为 yield 语句的返回值
- next方法可以传入实参
- 执行获取迭代器对象

```

1 <script>
2 //模拟获取 用户数据 订单数据 商品数据
3 function getUsers(){
4 setTimeout(()=>{
5 let data = '用户数据';
6 //调用 next 方法，并且将数据传入
7 iterator.next(data);
8 }, 1000);
9 }
10 function getOrders(){
11 setTimeout(()=>{
12 let data = '订单数据';
13 iterator.next(data);
14 }, 1000)
15 }
16 function getGoods(){
17 setTimeout(()=>{
18 let data = '商品数据';
19 iterator.next(data);
20 }, 1000)
21 }
22 function * gen(){
23 let users = yield getUsers();
24 let orders = yield getOrders();
25 let goods = yield getGoods();
26 }
27 //调用生成器函数
28 let iterator = gen();
29 iterator.next()
30 </script>

```

## 1.12promise

- Promise 是 ES6 引入的异步编程的新解决方案。语法上 Promise 是一个构造函数，用来封装异步操作并可以获取其成功或失败的结果。
  - 1) Promise 构造函数: Promise (excutor) {}
  - 2) Promise.prototype.then 方法
  - 3) Promise.prototype.catch 方法

- ```

1 //使用 promise 实现
2 const p = new Promise((resolve, reject) => {
3     fs.readFile("./resources/为学.md", (err, data) => {
4         resolve(data);
5     });
6 });
7
8 p.then(value => {
9     return new Promise((resolve, reject) => {
10         fs.readFile("./resources/插秧诗.md", (err, data) => {
11             resolve([value, data]);
12         });
13     });
14 }).then(value => {
15     return new Promise((resolve, reject) => {
16         fs.readFile("./resources/观书有感.md", (err, data) => {
17             //压入
18             value.push(data);
19             resolve(value);
20         });
21     })
22 }).then(value => {
23     console.log(value.join('\r\n'));
24 });

```

1.13 Set

- 1) size 返回集合的元素个数
- 2) add 增加一个新元素, 返回当前集合
- 3) delete 删除元素, 返回 boolean 值
- 4) has 检测集合中是否包含某个元素, 返回 boolean 值
- 5) clear 清空集合, 返回 undefined

- ```

1 <script>
2 //声明一个 set
3 let s = new Set();
4 let s2 = new Set(['大事儿', '小事儿', '好事儿', '坏事儿', '小事儿']);
5 //元素个数
6 console.log(s2.size);
7 //添加新的元素
8 s2.add('喜事儿');
9 //删除元素
10 s2.delete('坏事儿');
11 //检测
12 console.log(s2.has('糟心事'));
13 //清空
14 s2.clear();
15 // console.log(s2);
16 for(let v of s2){

```

```

17 console.log(v);
18 }
19 </script>

```

## 1.14 Map

- ES6 提供了 Map 数据结构。它类似于对象，也是键值对的集合。但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。Map 也实现了 iterator 接口，所以可以使用『扩展运算符』和『for...of...』进行遍历。
- Map 的属性和方法：
  - 1) size 返回 Map 的元素个数
  - 2) set 增加一个新元素，返回当前 Map
  - 3) get 返回键名对象的键值
  - 4) has 检测 Map 中是否包含某个元素，返回 boolean 值
  - 5) clear 清空集合，返回 undefined

```

1 <script>
2 //声明 Map
3 let m = new Map();
4 //添加元素
5 m.set('name', '尚硅谷');
6 m.set('change', function(){
7 console.log("我们可以改变你!!");
8 });
9 let key = {
10 school : 'ATGUGU'
11 };
12 m.set(key, ['北京', '上海', '深圳']);
13 //size
14 console.log(m.size);
15 //删除
16 m.delete('name');
17 //获取
18 console.log(m.get('change'));
19 console.log(m.get(key));
20 //清空
21 m.clear();
22 //遍历
23 for(let v of m){
24 console.log(v);
25 }
26 console.log(m);
27 </script>

```

## 1.15 class

- 1) class 声明类
- 2) constructor 定义构造函数初始化
- 3) extends 继承父类



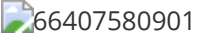
- 4) super 调用父级构造方法
- 5) static 定义静态方法和属性
- 6) 父类方法可以重写

```

1 <script>
2 //手机
3 function Phone(brand, price){
4 this.brand = brand;
5 this.price = price;
6 }
7 //添加方法
8 Phone.prototype.call = function(){
9 console.log("我可以打电话!!");
10 }
11 //实例化对象
12 let Huawei = new Phone('华为', 5999);
13 Huawei.call();
14 console.log(Huawei);
15 //class
16 class Shouji{
17 //构造方法 名字不能修改
18 constructor(brand, price){
19 this.brand = brand;
20 this.price = price;
21 }
22 //方法必须使用该语法, 不能使用 ES5 的对象完整形式
23 call(){
24 console.log("我可以打电话!!");
25 }
26 }
27 let onePlus = new Shouji("1+", 1999);
28 console.log(onePlus);
29 </script>

```

## 2.16数值扩展

- 



## 1.17对象扩展

- 1) Object.is 比较两个值是否严格相等, 与『===』行为基本一致 (+0 与 NaN)
- 2) Object.assign 对象的合并, 将源对象的所有可枚举属性, 复制到目标对象
- 3) \_\_proto\_\_、setPrototypeOf、setPrototypeOf 可以直接设置对象的原型

## 1.18

- ```
1  <script type="module">
```

```

2      //1. 通用的导入方式
3      //引入 m1.js 模块内容
4      import * as m1 from "./src/js/m1.js";
5      // //引入 m2.js 模块内容
6      import * as m2 from "./src/js/m2.js";
7      // //引入 m3.js
8      import * as m3 from "./src/js/m3.js";
9
10     //2. 解构赋值形式
11     import {school, teach} from "./src/js/m1.js";
12     import {school as guigu, findJob} from "./src/js/m2.js";
13     import {default as m3} from "./src/js/m3.js";
14
15     //3. 简便形式 针对默认暴露
16     import m3 from "./src/js/m3.js";
17     console.log(m3);
18 </script>

```

Vue全家桶

常用语法

单向数据绑定: v-bind: 双向绑定 v-model

vue核心

初识Vue:

- 介绍: 动态构建用户界面的渐进式 JavaScript 框架 尤雨溪
- 特点: 遵循mvvm模式

Hello案例

- ```

1 <!-- 准备好一个容器 -->
2 <div id="demo">
3 <h1>Hello, {{name.toUpperCase()}}, {{address}}</h1>
4 </div>
5 <script type="text/javascript">
6 Vue.config.productionTip = false; //阻止 vue 在启动时生成生产提示。
7 //创建Vue实例
8 new Vue({
9 el: "#demo", //el用于指定当前Vue实例为哪个容器服务, 值通常为css选择器字符串。
10 data: {
11 //data中用于存储数据, 数据供el所指定的容器去使用, 值我们暂时先写成一个对象。
12 name: "atguigu",
13 address: "北京",
14 },
15 });
16 </script>

```

- 注意：
- 1.想让Vue工作，就必须创建一个Vue实例，且要传入一个配置对象；
- 2.root容器里的代码依然符合html规范，只不过混入了一些特殊的Vue语法
- 3.root容器里的代码被称为【Vue模板】；
- 4.Vue实例和容器是一一对应的；
- 5.真实开发中只有一个Vue实例，并且会配合着组件一起使用；
- 6.{{xxx}}中的xxx要写js表达式，且xxx可以自动读取到data中的所有属性；
- 7.一旦data中的数据发生改变，那么页面中用到该数据的地方也会自动更新；

## 模板语法

- 1.插值语法：
  - 功能：用于解析标签体内容。
  - 写法：{{xxx}}，xxx是js表达式，且可以直接读取到data中的所有属性。
- 2.指令语法：
  - 功能：用于解析标签（包括：标签属性、标签体内容、绑定事件.....）。
  - 举例：v-bind:href="xxx" 或 简写为 :href="xxx"，xxx同样要写js表达式，且可以直接读取到data中的所有属性。
  - 备注：Vue中有很多的指令，且形式都是：v-????，此处我们只是拿v-bind举个例子。

```

1 <div id="root">
2 <h1>插值语法</h1>
3 <h3>你好, {{name}}</h3>
4 <hr/>
5 <h1>指令语法</h1>
6 <a v-bind:href="school.url.toUpperCase()" x="hello">点我去{{school.name}}学
习1
7 //缩写
8 <a :href="school.url" x="hello">点我去{{school.name}}学习2
9 </div>
10 new Vue({
11 el: '#root',
12 data: {
13 name: 'jack',
14 school: {
15 name: '尚硅谷',
16 url: 'http://www.atguigu.com',
17 }
18 }
19 })

```

## 数据绑定

- Vue中有2种数据绑定的方式：
  - 1.单向绑定(v-bind)：数据只能从data流向页面。
  - 2.双向绑定(v-model)：数据不仅能从data流向页面，还可以从页面流向data。
    - 备注：

- 1.双向绑定一般都应用在表单类元素上（如：input、select等）
- 2.v-model:value 可以简写为 v-model，因为v-model默认收集的就是value值。

```

1 单向数据绑定: <input type="text" :value="name">

2 双向数据绑定: <input type="text" v-model="name">

3 new Vue({
4 el: '#root',
5 data:{
6 name: '尚硅谷'
7 }
8 })

```

## data和el的两种写法

- 1.el有2种写法
  - (1).new Vue时候配置el属性。
  - (2).先创建Vue实例，随后再通过vm.\$mount('#root')指定el的值。
- 2.data有2种写法
  - (1).对象式
  - 2).函数式
  - 如何选择：目前哪种写法都可以，以后学习到组件时，data必须使用函数式，否则会报错。
- 3.一个重要的原则：
  - 由Vue管理的函数，一定不要写箭头函数，一旦写了箭头函数，this就不再是Vue实例了。

```

1 //el的两种写法
2 el: '#root', //第一种写法
3 v.$mount('#root') //第二种写法 */
4 data:{ name: '尚硅谷' }
5 data(){
6 return{
7 name: '尚硅谷'
8 }

```

## Vue中MVVM

- MVVM模型
  1. M：模型(Model)：data中的数据
  2. V：视图(View)：模板代码
  3. VM：视图模型(ViewModel)：Vue实例
- 4. 观察发现：
  - 1.data中所有的属性，最后都出现在了vm身上。
  - 2.vm身上所有的属性 及 Vue原型上所有属性，在Vue模板中都可以直接使用。

## 数据代理

- 回顾Object.defineProperty方法

```

1 <script type="text/javascript" >
2 let number = 18
3 let person = {
4 name: '张三',
5 sex: '男',
6 }
7 Object.defineProperty(person, 'age', {
8 // value: 18,
9 enumerable: true, // 控制属性是否可以枚举, 默认值是 false
10 writable: true, // 控制属性是否可以被修改, 默认值是 false
11 configurable: true // 控制属性是否可以被删除, 默认值是 false
12 // 当有人读取 person 的 age 属性时, get 函数 (getter) 就会被调用, 且返回值就是
13 // age 的值
14 get() {
15 console.log('有人读取 age 属性了')
16 return number
17 },
18 // 当有人修改 person 的 age 属性时, set 函数 (setter) 就会被调用, 且会收到修改
19 // 的具体值
20 set(value) {
21 console.log('有人修改了 age 属性, 且值是', value)
22 number = value
23 }
24 })
25 // console.log(Object.keys(person))
26 console.log(person)
27 </script>

```

## vue中的数据代理

- 1. Vue中的数据代理:
  - 通过vm对象来代理data对象中属性的操作 (读/写)
- 2. Vue中数据代理的好处:
  - 更加方便的操作data中的数据
- 3. 基本原理:
  - 通过Object.defineProperty()把data对象中所有属性添加到vm上。
  - 为每一个添加到vm上的属性, 都指定一个getter/setter。
  - 在getter/setter内部去操作 (读/写) data中对应的属性。

## 事件处理

- 1. 使用v-on:xxx 或 @xxx绑定事件, 其中xxx是事件名;
- 2. 事件的回调需要配置在methods对象中, 最终会在vm上;
- 3. methods中配置的函数, 不要用箭头函数! 否则this就不是vm了;
- 4. methods中配置的函数, 都是被Vue所管理的函数, this的指向是vm 或 组件实例对象;
- 5. @click="demo" 和 @click="demo(\$event)" 效果一致, 但后者可以传参;

```

1 <div id="root">
2 <h2>欢迎来到{{name}}学习</h2>
3 <!-- <button v-on:click="showInfo">点我提示信息</button> -->
4 <button @click="showInfo1">点我提示信息1 (不传参) </button>
5 <button @click="showInfo2($event,66)">点我提示信息2 (传参) </button>
6 </div>
7 const vm = new Vue({
8 el: '#root',
9 data: {
10 name: '尚硅谷',
11 },
12 methods: {
13 showInfo1(event) {
14 // console.log(event.target.innerText)
15 // console.log(this) //此处的this是vm
16 alert('同学你好! ')
17 },
18 showInfo2(event, number) {
19 console.log(event, number)
20 // console.log(event.target.innerText)
21 // console.log(this) //此处的this是vm
22 alert('同学你好! ! ')
23 }
24 }
25 })

```

## • 事件修饰符

### ◦ Vue中的事件修饰符：前三个常用

- 1.prevent: 阻止默认事件（常用）；
- 2.stop: 阻止事件冒泡（常用）；
- 3.once: 事件只触发一次（常用）；
- 4.capture: 使用事件的捕获模式；
- 5.self: 只有event.target是当前操作的元素时才触发事件；
- 6.passive: 事件的默认行为立即执行，无需等待事件回调执行完毕；

## • 键盘事件

### ◦ 1.Vue中常用的按键别名：

- 回车 => enter
- 删除 => delete (捕获“删除”和“退格”键)
- 退出 => esc
- 空格 => space
- 换行 => tab (特殊，必须配合keydown去使用)
- 上 => up
- 下 => down
- 左 => left
- 右 => right

### ◦ 2.Vue未提供别名的按键，可以使用按键原始的key值去绑定，但注意要转为kebab-case（短横线命名）

### ◦ 3.系统修饰键（用法特殊）：ctrl、alt、shift、meta

- (1).配合keyup使用：按下修饰键的同时，再按下其他键，随后释放其他键，事件才被触发。
- (2).配合keydown使用：正常触发事件。
- 4.也可以使用keyCode去指定具体的按键（不推荐）
- 5.Vue.config.keyCodes.自定义键名 = 键码，可以去定制按键别名
- 6.console.log(e.keyCode,e.key); 使用代码查询 按键对应名称 两个单词组合 全小写 + -
  - 如CapsLock 对应名字： caps-lock

## 计算属性

- 1.定义：要用的属性不存在，要通过已有属性计算得来。
- 2.原理：底层借助了Object.defineProperty方法提供的getter和setter。
- 3.get函数什么时候执行？
  - (1).初次读取时会执行一次。
  - 2).当依赖的数据发生改变时会被再次调用。
- 4.优势：与methods实现相比，内部有缓存机制（复用），效率更高，调试方便。
- 5.备注：
  - 1.计算属性最终会出现在vm上，直接读取使用即可。
  - 2.如果计算属性要被修改，那必须写set函数去响应修改，且set中要引起计算时依赖的数据发生改变。
- 1. 简写 当只有get() 时 可以直接写出函数形式

```

1 <script type="text/javascript">
2 Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。
3
4 const vm = new Vue({
5 el: '#root',
6 data: {
7 firstName: '张',
8 lastName: '三',
9 x: '你好'
10 },
11 },
12
13 computed: {
14 fullName: {
15 //get有什么作用？当有人读取fullName时，get就会被调用，且返回值就作为fullName的值
16 //get什么时候调用？1.初次读取fullName时。2.所依赖的数据发生变化时。
17 get() {
18 console.log('get被调用了')
19 return this.firstName + '-' + this.lastName
20 },
21 //set什么时候调用？当fullName被修改时。
22 set(value) {
23 console.log('set', value)
24 const arr = value.split('-')
25 this.firstName = arr[0]
26 this.lastName = arr[1]
27 }
28 }
29 }

```

```

28 }
29 }
30 })
31 //简写
32 fullName(){
33 console.log('get被调用了')
34 return this.firstName + '-' + this.lastName
35 }
36 </script>

```

## 监视属性

- 监视属性watch:
  - 1.当被监视的属性变化时, 回调函数自动调用, 进行相关操作
  - 2.监视的属性必须存在, 才能进行监视!
  - 3.监视的两种写法:
    - (1).new Vue时传入watch配置
    - (2).通过vm.\$watch监视
- 深度监视:
  - (1).Vue中的watch默认不监测对象内部值的改变 (一层)。
  - (2).配置deep:true可以监测对象内部值改变 (多层)。
  - 备注:
    - (1).Vue自身可以监测对象内部值的改变, 但Vue提供的watch默认不可以!
    - (2).使用watch时根据数据的具体结构, 决定是否采用深度监视。
  - (3).// immediate:true, // deep:true, // 两个属性没有时可以简写

```

1 const vm = new Vue({
2 el: '#root',
3 data: {
4 isHot: true,
5 },
6 computed: {
7 info() {
8 return this.isHot ? '炎热' : '凉爽'
9 }
10 },
11 methods: {
12 changeWeather() {
13 this.isHot = !this.isHot
14 }
15 },
16 watch: {
17 isHot: {
18 immediate: true, //初始化时让handler调用一下
19 //handler什么时候调用? 当isHot发生改变时。
20 handler(newValue, oldValue) {
21 console.log('isHot被修改了', newValue, oldValue)

```



```

22 }
23 }
24 }
25 })
26 //通过vm.$watch监视
27 vm.$watch('isHot',{
28 immediate:true, //初始化时让handler调用一下
29 //handler什么时候调用? 当isHot发生改变时。
30 handler(newValue,oldValue){
31 console.log('isHot被修改了',newValue,oldValue)
32 }
33 })

```

- 与computed区别： computed能完成的功能， watch都可以完成。 watch可以执行异步操作
- 所有不被Vue所管理的函数（定时器的回调函数、ajax的回调函数等、Promise的回调函数），最好写成箭头函数，这样this的指向才是vm 或 组件实例对象。

## 绑定样式

- class样式
  - 写法:class="xxx" xxx可以是字符串、对象、数组。
  - 字符串写法适用于：类名不确定，要动态获取。
  - 对象写法适用于：要绑定多个样式，个数不确定，名字也不确定。
  - 数组写法适用于：要绑定多个样式，个数确定，名字也确定，但不确定用不用。
- style样式
  - :style="{fontSize: xxx}"其中xxx是动态值。
  - :style="[a,b]"其中a、b是样式对象。

- ```

1 <div id="root">
2   <!-- 绑定class样式--字符串写法, 适用于: 样式的类名不确定, 需要动态指定 -->
3   <div class="basic" :class="mood" @click="changeMood">{{name}}</div> <br/>
4   <br/>
5   <!-- 绑定class样式--数组写法, 适用于: 要绑定的样式个数不确定、名字也不确定 -->
6   <div class="basic" :class="classArr">{{name}}</div> <br/><br/>
7
8   <!-- 绑定class样式--对象写法, 适用于: 要绑定的样式个数确定、名字也确定, 但要动态决定用不用 -->
9   <div class="basic" :class="classObj">{{name}}</div> <br/><br/>
10
11   <!-- 绑定style样式--对象写法 -->
12   <div class="basic" :style="styleObj">{{name}}</div> <br/><br/>
13   <!-- 绑定style样式--数组写法 -->
14   <div class="basic" :style="styleArr">{{name}}</div>
15 </div>
16 <script type="text/javascript">
17   Vue.config.productionTip = false
18
19   const vm = new Vue({
20     el: '#root',

```

```

21     data:{
22         name:'尚硅谷',
23         mood:'normal',    //字符串
24         classArr:['atguigu1','atguigu2','atguigu3'], //数组
25         classObj:{        //对象
26             atguigu1:false,
27             atguigu2:false,
28         },
29         styleObj:{        //style对象样式
30             fontSize: '40px',
31             color:'red',
32         },
33         styleObj2:{
34             backgroundColor:'orange'
35         },
36         styleArr:[        //style数组样式
37             {
38                 fontSize: '40px',
39                 color:'blue',
40             },
41             {
42                 backgroundColor:'gray'
43             }
44         ]
45     },
46     methods: {
47         changeMood(){
48             const arr = ['happy','sad','normal']
49             const index = Math.floor(Math.random()*3)
50             this.mood = arr[index]
51         }
52     },
53 })
54 </script>

```

条件渲染

- 1.v-if
 - 写法: (1).v-if="表达式"
 - (2).v-else-if="表达式"
 - (3).v-else="表达式"
 - 适用于: 切换频率较低的场景。
 - 特点: 不展示的DOM元素直接被移除。
 - 注意: v-if可以和v-else-if、v-else一起使用, 但要求结构不能被“打断”。
- 2.v-show
 - 写法: v-show="表达式"
 - 适用于: 切换频率较高的场景。
 - 特点: 不展示的DOM元素未被移除, 仅仅是使用样式隐藏掉
- 3.备注: 使用v-if的时, 元素可能无法获取到, 而使用v-show一定可以获取到。

- ```

1 <div id="root">
2 <h2>当前的n值是:{{n}}</h2>
3 <button @click="n++">点我n+1</button>
4 <!-- 使用v-show做条件渲染 -->
5 <h2 v-show="false">欢迎来到{{name}}</h2> -->
6 <h2 v-show="1 === 1">欢迎来到{{name}}</h2> -->
7 <!-- 使用v-if做条件渲染 -->
8 <h2 v-if="false">欢迎来到{{name}}</h2> -->
9 <h2 v-if="1 === 1">欢迎来到{{name}}</h2> -->
10 <!-- v-else和v-else-if -->
11 <div v-if="n === 1">Angular</div>
12 <div v-else-if="n === 2">React</div>
13 <div v-else-if="n === 3">Vue</div>
14 <div v-else>哈哈</div> -->
15 <!-- v-if与template的配合使用 -->
16 <template v-if="n === 1">
17 <h2>你好</h2>
18 <h2>尚硅谷</h2>
19 <h2>北京</h2>
20 </template>
21 </div>

```

## 列表渲染

- v-for指令:
- 1.用于展示列表数据
- 2.语法: v-for="(item, index) in xxx" :key="使用唯一标识" 如: id phone 等
- 3.可遍历: 数组、对象、字符串 (用的很少)、指定次数 (用的很少)

## key的原理

- 面试题: react、vue中的key有什么作用? (key的内部原理)
- 1、虚拟DOM中key的作用:
  - key是虚拟DOM对象的标识, 当数据发生变化时, Vue会根据【新数据】生成【新的虚拟DOM】, 随后Vue进行【新虚拟DOM】与【旧虚拟DOM】的差异比较, 比较规则如下
- 2、对比规则:
  - (1).旧虚拟DOM中找到了与新虚拟DOM相同的key:
    - ①.若虚拟DOM中内容没变, 直接使用之前的真实DOM!
    - ②.若虚拟DOM中内容变了, 则生成新的真实DOM, 随后替换掉页面中之前的真实DOM。
  - (2).旧虚拟DOM中未找到与新虚拟DOM相同的key
    - 创建新的真实DOM, 随后渲染到到页面。
- 3、用index作为key可能会引发的问题:
  - 若对数据进行: 逆序添加、逆序删除等破坏顺序操作:
    - 会产生没有必要的真实DOM更新 ==> 界面效果没问题, 但效率低。
  - 如果结构中还包含输入类的DOM:
    - 会产生错误DOM更新 ==> 界面有问题。

- 4、开发中如何选择key?
  - 1.最好使用每条数据的唯一标识作为key, 比如id、手机号、身份证号、学号等唯一值。
  - 2.如果不存在对数据的逆序添加、逆序删除等破坏顺序操作, 仅用于渲染列表用于展示, 使用index作为key是没有问题的。

## 列表过滤

- 用到 filter方法 和indexOf方法
- indexOf方法: 是否包含字符

```

1 用computed实现
2 new Vue({
3 el: '#root',
4 data:{
5 keyWord: '',
6 persons:[
7 {id: '001', name: '马冬梅', age: 19, sex: '女'},
8 {id: '002', name: '周冬雨', age: 20, sex: '女'},
9 {id: '003', name: '周杰伦', age: 21, sex: '男'},
10 {id: '004', name: '温兆伦', age: 22, sex: '男'}
11]
12 },
13 computed:{
14 filPerons(){
15 return this.persons.filter((p)=>{
16 return p.name.indexOf(this.keyWord) !== -1
17 })
18 }
19 }
20 })

```

## 列表排序

- ```

1  <script type="text/javascript">
2      new Vue({
3      el: '#root',
4      data:{
5          keyWord: '',
6          sortType: 0, //0原顺序 1降序 2升序
7          persons:[
8              {id: '001', name: '马冬梅', age: 30, sex: '女'},
9              {id: '002', name: '周冬雨', age: 31, sex: '女'},
10             {id: '003', name: '周杰伦', age: 18, sex: '男'},
11             {id: '004', name: '温兆伦', age: 19, sex: '男'}
12         ]
13     },
14     computed:{
15         filPerons(){
16             const arr = this.persons.filter((p)=>{
17                 return p.name.indexOf(this.keyWord) !== -1

```

```

18         })
19         //判断一下是否需要排序
20         if(this.sortType){
21             arr.sort((p1,p2)=>{
22                 return this.sortType === 1 ? p2.age-p1.age : p1.age-
p2.age
23             })
24         }
25         return arr
26     }
27 }
28 })
29 </script>

```

数据监测

- Vue监视数据的原理：
 1. vue会监视data中所有层次的数据。
 - 2.如何监测对象中的数据？
 - 通过setter实现监视，且要在new Vue时就传入要监测的数据。
 - (1).对象中后追加的属性，Vue默认不做响应式处理
 - (2).如需给后添加的属性做响应式，请使用如下API：
 - `Vue.set(target, propertyName/index, value)` 或
 - `vm.$set(target, propertyName/index, value)`
 - 1. 如何监测数组中的数据？
 - 通过包裹数组更新元素的方法实现，本质就是做了两件事：
 - (1).调用原生对应的方法对数组进行更新。
 - (2).重新解析模板，进而更新页面。
 - 4.在Vue修改数组中的某个元素一定要用如下方法：
 - 1.使用这些API:push()、pop()、shift()、unshift()、splice()、sort()、reverse()
 - 2.Vue.set() 或 vm.\$set()
- 特别注意：Vue.set() 和 vm.\$set() 不能给vm 或 vm的根数据对象 添加属性！！
 - vue中无法直接索引直接操作数组元素

- ```
1 updateHobby(){
2 this.student.hobby.splice(0,1,'开车')
3 Vue.set(this.student.hobby,0,'开车')
4 this.$set(this.student.hobby,0,'开车')
5 addFriend(){
6 this.student.friends.unshift({name:'jack',age:70})
7 },
8 updateFirstFriendName(){
9 this.student.friends[0].name = '张三'
10 },
```

## 收集表单数据

- 收集表单数据：
- 若：<input type="text"/>，则v-model收集的是value值，用户输入的就是value值。
- 若：<input type="radio"/>，则v-model收集的是value值，且要给标签配置value值。
- 若：<input type="checkbox"/>
- 1.没有配置input的value属性，那么收集的就是checked（勾选 or 未勾选，是布尔值）
- 2.配置input的value属性：
- (1)v-model的初始值是非数组，那么收集的就是checked（勾选 or 未勾选，是布尔值）
- (2)v-model的初始值是数组，那么收集的的就是value组成的数组
- 注：v-model的三个修饰符：
- lazy：失去焦点再收集数据
- number：输入字符串转为有效的数字
- trim：输入首尾空格过滤
-

```

<!-- 准备好一个容器-->
<div id="root">
 <form @submit.prevent="demo">
 账号: <input type="text" v-model.trim="userInfo.account">

 密码: <input type="password" v-model="userInfo.password">

 年龄: <input type="number" v-model.number="userInfo.age">

 性别:
 男<input type="radio" name="sex" v-model="userInfo.sex" value="male">
 女<input type="radio" name="sex" v-model="userInfo.sex" value="female">

 爱好:
 学习<input type="checkbox" v-model="userInfo.hobby" value="study">
 打游戏<input type="checkbox" v-model="userInfo.hobby" value="game">
 吃饭<input type="checkbox" v-model="userInfo.hobby" value="eat">

 所属校区
 <select v-model="userInfo.city">
 <option value="">请选择校区</option>
 <option value="beijing">北京</option>
 <option value="shanghai">上海</option>
 <option value="shenzhen">深圳</option>
 <option value="wuhan">武汉</option>
 </select>

 其他信息:
 <textarea v-model.lazy="userInfo.other"></textarea>

 <input type="checkbox" v-model="userInfo.agree">阅读并接受《用户协议》
 <button>提交</button>
 </form>
</div>
</body>

<script type="text/javascript">
 Vue.config.productionTip = false

 new Vue({
 el: '#root',
 data: {
 userInfo: {
 account: '',
 password: '',
 age: 18,
 sex: 'female',
 hobby: [],
 city: 'beijing',
 other: '',
 agree: ''
 }
 },
 methods: {
 demo() {
 console.log(JSON.stringify(this.userInfo))
 }
 }
 })
</script>

```

## 过滤器 filters:

- 过滤器:
- 定义: 对要显示的数据进行特定格式化后再显示 (适用于一些简单逻辑的处理)
- 语法:
  - 1.注册过滤器: Vue.filter(name,callback) 或 new Vue{filters:{}}
  - 2.使用过滤器: {{ xxx | 过滤器名}} 或 v-bind:属性 = "xxx | 过滤器名"
- 备注:
  - 1.过滤器也可以接收额外参数、多个过滤器也可以串联
  - 2.并没有改变原本的数据, 是产生新的对应的数据

```

//局部过滤器
filters:{
 timeFormater(value,str='YYYY年MM月DD日 HH:mm:ss'){
 // console.log('@',value)
 return dayjs(value).format(str)
 }
}

```

## 内置指令

- v-bind :单向绑定解析表达式, 可简写为 :xxx
- v-model :双向数据绑定
- v-for :遍历数组/对象/字符串
- v-on :绑定事件监听, 可简写为@
- v-if :条件渲染 (动态控制节点是否存在)
- v-else :条件渲染 (动态控制节点是否存在)
- v-show :条件渲染 (动态控制节点是否展示)
- v-text指令:
  - 1.作用: 向其所在的节点中渲染文本内容。
  - 2.与插值语法的区别: v-text会替换掉节点中的内容, {{xx}}则不会。
- v-html指令:
  - 1.作用: 向指定节点中渲染包含html结构的内容。
  - 2.与插值语法的区别:
    - (1).v-html会替换掉节点中所有的内容, {{xx}}则不会。
    - (2).v-html可以识别html结构。
  - 3.严重注意: v-html有安全性问题!!!
    - (1).在网站上动态渲染任意HTML是非常危险的, 容易导致XSS攻击。
    - (2).一定要在可信的内容上使用v-html, 永不要用在用户提交的内容上!
- v-cloak指令 (没有值) :
  - 1.本质是一个特殊属性, Vue实例创建完毕并接管容器后, 会删掉v-cloak属性。
  - 2.使用css配合v-cloak可以解决网速慢时页面展示出{{xxx}}的问题。
- v-once指令:
  - 1.v-once所在节点在初次动态渲染后, 就视为静态内容了。
  - 2.以后数据的改变不会引起v-once所在结构的更新, 可以用于优化性能。
- v-pre指令:
  - 1.跳过其所在节点的编译过程。
  - 2.可利用它跳过: 没有使用指令语法、没有使用插值语法的节点, 会加快编译

## 自定义指令

- 一、定义语法:
  - 1).局部指令: 在directive里书写 对象和函数两种格式
    - 如果vue把东西渲染到页面时 执行 命令 则 函数形式将 无法使用 推荐 使用对象格式
    - 函数形式 只在绑定成功和重新解析时 调用
  - (2).全局指令: Vue.directive 实例对象 之前 配置
- 二、配置对象中常用的3个回调:



- (1).bind: 指令与元素成功绑定时调用。
- (2).inserted: 指令所在元素被插入页面时调用。
- (3).update: 指令所在模板结构被重新解析时调用。
- 三、备注:
- 1.指令定义时不加v-, 但使用时要加v-
- 2.指令名如果是多个单词, 要使用kebab-case命名方式, 不要用camelCase命名。

```

1 //定义全局指令
2 Vue.directive('fbind',{
3 //指令与元素成功绑定时 (一上来)
4 bind(element,binding){
5 element.value = binding.value
6 },
7 //指令所在元素被插入页面时
8 inserted(element,binding){
9 element.focus()
10 },
11 //指令所在的模板被重新解析时
12 update(element,binding){
13 element.value = binding.value
14 }
15 })
16
17 //局部指令
18 directives:{
19 //函数形式 无法处理
20 'big-number'(element,binding){
21 // console.log('big')
22 element.innerText = binding.value * 10
23 },
24 big(element,binding){
25 // console.log('big',this) //注意此处的this是window
26 // console.log('big')
27 element.innerText = binding.value * 10
28 },
29 fbind:{
30 //指令与元素成功绑定时 (一上来)
31 bind(element,binding){
32 element.value = binding.value
33 },
34 //指令所在元素被插入页面时
35 inserted(element,binding){
36 element.focus()
37 },
38 //指令所在的模板被重新解析时
39 update(element,binding){
40 element.value = binding.value
41 }
42 }
43 }

```

# 生命周期

- 生命周期：
  - 1.又名：生命周期回调函数、生命周期函数、生命周期钩子。
  - 2.是什么：Vue在关键时刻帮我们调用的一些特殊名称的函数。
  - 3.生命周期函数的名字不可更改，但函数的具体内容是程序员根据需求编写的。
  - 4.生命周期函数中的this指向是vm 或 组件实例对象
- 常用的生命周期钩子：
  - 1.mounted: 发送ajax请求、启动定时器、绑定自定义事件、订阅消息等【初始化操作】。
  - 2.beforeDestroy: 清除定时器、解绑自定义事件、取消订阅消息等【收尾工作】。`$destroy()`自毁
- 关于销毁Vue实例
  - 1.销毁后借助Vue开发者工具看不到任何信息。
  - 2.销毁后自定义事件会失效，但原生DOM事件依然有效。
  - 3.一般不会对beforeDestroy操作数据，因为即便操作数据，也不会再触发更新流程了。
- 

| 数据代理创建前 beforeCreate() | 数据代理创建完成 Create() |
|------------------------|-------------------|
| 数据挂载前 beforeMount()    | 数据挂载完毕mounted()   |
| 数据更新前beforeUpdate()    | 数据更新完毕updated()   |
| 数据销毁beforeDestory()    | 数据销毁完毕destoryed() |