js

可以在 `视图` 菜单中关闭

不再显示关闭

js基础函数

变量命名：建议使用驼峰命名法

1.遵守驼峰命名法。首字母小写，后面单词的首字母需要大写。 myFirstName

2.判断 是否是数字的函数isNaN(value) 如果value 不能转换成数字 返回true 否则返回false

转换成数字型

parseInt 转换成 Int parseFloat 转换成 float

函数定义形式

```
//  定义一个函数
// 使用关键之 function
// 第一种形式
function cal(){
  console.log('定义一个cal 函数')
}

cal()

// 第二种形式
// function 放在后面

sum = function(){
  console.log('定义sum函数')
}

sum()
```

arguments 使用

```
// 当传入参数不知道多少个数时

// 可以使用 arguments 这个伪数组代替

// 特点是:  具有 length 属性
//          按索引方式储存数据
//          不具有数组的 push , pop 等方法


function max(){
```

```
    let num = arguments[0]

    for(let i = 1; i < arguments.length; i++){
      if(num < arguments[i]){
        num = arguments[i]
      }
    }
    return num;
  }

  console.log(max(1,2,3,4))
```

js对象基本使用

```
// 声明一个 对象函数
  function Person(name,age){
    this.name = name;
    this.age = age;
    this.say = function(){
      console.log('saying')
    }
  }

  // 声明一个对象

  var student = {

    chinese:12,

    eng:20,

    math:20

  }

  console.log(student.chinese, student.eng, student.math)

  let person = new Person('zs',10)

  console.log(person.name)
  console.log(person.age)
  person.say()
```

Math对象

```
    // 向上取整 Math.floor();

    console.log(Math.floor(1.7))

    // 向下取整 Math.cile()

    console.log(Math.ceil(1.2))

    // 绝对值

    console.log(Math.abs(-1.2))

    // Math.around() 四舍五入

    console.log(Math.round(1.4))
    console.log(Math.round(1.5))

    // 求最大值 Math.max()
    console.log(Math.max(1,2))

    // 求最小值
    console.log(Math.min(1,2))

    // 求随机值 生成0 - 1 直接的数字
    console.log(Math.random())
    // 保留两位小数
    num = num.toFixed(2);
```

日期对象

```
let now = new Date()
// 获得年份
let year = now.getFullYear()
// 获得月份
let month = now.getMonth()
// 获得日期
let date = now.getDate()

 console.log(date)
console.log(month)
console.log(year)
console.log(now)
```

数组对象

```
1.instanceof 运算符
let arr = [1,2,3]
    // 判断arr 是否为 Array 的类型
  console.log(arr instanceof Array)

  // 在H5中新增的
  console.log(Array.isArray(arr))
```

```
2.数组 中 一般常见的 增加元素和删除元素的 函数
2.1 在元素末尾 怎加元素
let arr = [1,2]

console.log(arr)

//向末尾加入元素
arr.push(3)

console.log(arr)
//向数组的首位增加元素

arr.unshift(3)
console.log(3)

2.2 删除元素
//删除末尾元素
arr.pop()
//删除数组第一个元素
arr.shift()
```

```
数组颠倒
let arr = [1,2,3,4]

arr.reverse()

数组排序

let arr = [1,3,2,4]

arr.sort()
```

indexOf

```
数组索引
indexOf() 查找指定元素 是否存在 如果不存在 返回值 -1
找到的是第一个元素
let str = '12334'

str.indexOf('5')   //返回 -1

如果存在 返回 指定元素的 在 数组当中的索引
```

lastIndexOf()

```
lastIndexOf() 找寻的是 当前字符串之后一次出现这个字符的位置
let str = '122212'

str.lastIndexOf('2')
```

indexOf 和 lastIndexOf()

```
通过indexOf 可以找到 第一个 字符出现的位置
通过lastIndexof 可以找到最后一个字符的位置
通过这两个函数结合 ==> 可以检索出 所有字符的位置

let firstchar = 'a'
let lastchar = 'a'

let str = 'abacda'

let firstplace = str.indexOf(firstchar)

let lastplace = str.lastIndexOf(lastchar)

console.log(firstplace,lastplace)
```

```
数组转成字符串函数
toString()
let num = 2
let str = num.toString()
```

join

```
join 函数 将数组 中的所有元素组合起来形成一个字符串
let str = [1,2,3,4]

const str = str.join(',')   // 表示 用 ， 将 数组元素 分割开来
const str = str.join('')   // 表示 用  将 数组元素连在一起
```

```
let arr = [1,2,3,4,5,6,7,8,9]

console.log('修改前' + arr)

/**
 * @name: arr.join
 * @msg: 使用join 将数组分开
 * @return 返回一个数组
 */
    // 使用 | arr中 元素 分割开来
let newArr = arr.join('|')

console.log('修改后' + newArr)
```

```
concat() 连接 字符串函数

let str1 = 'hello '
let str2 = 'world!'
const str = str1.concat(str2)
const str = str1.concat(str2,srt2)
```

slice splice

```
截取字符串
slice

let str1 = 'hello '
let str2 = 'world!'

str1.slice(0,1) // 截取 0到1 的字符串
str1.slice(1) // 截取1到 末尾的字符串

let str = 'abcde'

/**
     * @name: slice
     * @msg: 截取指定子长的字符串
     * @return String
     */
let newStr1 = str.slice(1,2)

console.log(str)

console.log(newStr1)

let newStr2 = str.slice(1)

console.log(newStr2)
```

```
console.log(str)

/**
     * @name: slice
     * @msg: slice对数组进行操作
     * @return 数组
     */
let arr = [1,2,3,4,5,6,7,8,9]

let newArr = arr.slice(0,5)

console.log(newArr)
```

splice() 删除 增加

删除

```
let str1 = [1,2,3,4]

str1.splice(1,2)  // 起始 位置是 1,  删除元素个数 是 2
str1.splice(1,0)  // 起始 位置是 1,  删除元素个数 是 0

//增加

let str2 = [1,2,3,4]

str2.splice(1,2,5) // 起始 位置是 1,  删除元素个数 是 2 在 索引是 1 的后面加入 5


/**
   * @name: splice
   * @msg: splice 对数组进行 增加 删除操作 不能对字符串进行操作！！！！
   * @return void
   */

// var str = 'abcd'

// str.splice(0, 1)

// console.log(str)

let arr = [1,2,3,4,5]

// 删除指定元素 index = 0 num = 2
arr.splice(0,0)

console.log(arr)

// 增加指定元素
// index = 3 删除个数是 0 将 num = 10 添加到 index 位置上去
```

```
arr.splice(3,0,10)

console.log(arr)
```

字符串对象

charAt(index)

```
charAt(index)
let str = 'abc'
console.log(str.charAt(0))  输出第0个字符
```

toUppercase() 和 toLowerCase()

```
let str1 = 'AbcdE'
// toUppercase 是将所有的字符转变成大写的
str1.toUppercase()
// toLowercase 是将所有的字符转变成小写的
str1.toLowercase()

这两个函数 通常还和 str[index] 或者 str.charAt(index) 一起使用
```

字符串操作

```
连接字符串 concat()
substr(star,length) // 起时索引 和截取长度
slice(star,end) //end 取不到
substring(star,end)
```

字符串替换

```
replace
let str = 'hello js!'
// repalce 作用之后 会返回一个 字符串
const str1 = str.replace('j','vue') // 可以是一个字符
const str1 = str.replace('js','vue') // 可以是一个字符串
```

切分字符串成为一个数组 split()

```
let str = 'a,b,c,d'
const str1 = str.split(',') //以 ',' 为分割符号 将字符串分割成一个数组
const str2 = str.split('') // 以 '' 为分割符号 将字符串分割成一个数组
```

js webAPI

dom操作对象

querySelector

```
querySelector 获取元素 返回的是一个伪数组
 <ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
  </ul>
  <script>
    let li = document.querySelector("ul").querySelectorAll("li")

    for(let i = 0 ; i < li.length;i++){
      // 得到的li 是一个伪数组
      console.log(li[i].innerHTML)
    }
  </script>

innerHTML 和 innerText 区别 前者可也解析 标签 后者不能解析标签
li[i].innerHTML =  '<h1>' + i + '</h1>'
li[i].innerText =  '<h1>' + i + '</h1>'
```

修改样式

```
<div class="container">

  </div>
  <script>

    let box = document.querySelector('.container')

    box.style.backgroundColor = 'red'
 </script>
```

修改类名 覆盖原来的类名

```
 <div class="container">

  </div>
  <script>

    let box = document.querySelector('.container')

    box.className = 'red'

  </script>
```

增加类名 和 删除类名

```
// dom.classList.add(类名)
// dom.classList.remove(类名)

box.classList.add('red')
box.classList.remove('red')
```

设置定时器 和 清除定时器

```
定时器的种类
setTimeout(function(){},time)
setInterval(function(){},time)

//定义一个变量去接收
let time = setTimeout(function(){},time)
let time = setInterval(function(){},time)

// 定义了变量之后 方便 清除 定时器
clearTimeout(time) // 清除对应的计时器
```

注册事件的两种方法

```
1. .onclick = function(){}
2. .addEventListener('click',function(){})
```

事件类型

1 鼠标事件

```
1.鼠标经过 mouseover mouseenter （没有冒泡行为推荐使用）
2.鼠标离开 mouseout  mouseleave （没有冒泡行为推荐使用）
3.鼠标移动
div.addEventListener("mouseover",()=>{
      console.log('over')
})
div.addEventListener("mouseout",()=>{
      console.log('leave')
})
div.addEventListener("mousemove",()=>{
      console.log('move')
})
div.addEventListener("mouseenter",()=>{
      console.log('over')
})
div.addEventListener("mouseleave",()=>{
      console.log('leave')
})
```

2 焦点事件

```
1.获取焦点
2.失去焦点
 input.addEventListener('focus',()=>{
      console.log('得到焦点')
})
input.addEventListener('blur',()=>{
      console.log('失去焦点')
})
```

！！！ 节点操作

1 生成节点

```
let li = document.querySelector('li')
let new_li = document.createElement('li')
```

2 插入节点

```
li.appendChild(new_li)
```

3 删除节点

```
li.removeChild(new_li)
```

4 插入指定位置

```
父元素.insertBefore(插入的元素,指定元素的位置)
 let li = document.querySelector('li')
 let ul = document.querySelector('ul')
 let insert = document.createElement('div')
 ul.insertBefore(insert, li)
```

5 元素节点的区别

子节点

```
firstChild 获取的是 text
firstElementChild 才是我们真正所需要的接待你
firstElementChild = children[0]
```

获取到的子节点和父节点

```
父节点
parentNode
子节点
childNodes 包括text
children 是真正我们所需要的节点
```

事件对象

1.事件常用的属性

```
1.clientX clientY 这个是相对浏览器的 (0, 0)  而言
2.offsetX offsetY 这个是相对 触发事件的 (0, 0)  盒子而言
通过event 默认参数可以获得 固定的属性
let body = document.querySelector('div')
body.addEventListener('mousemove',(event)=>{
 console.log(event.clientX,event.offsetX)
})
```

2.事件流

捕获和冒泡产生

```
事件流分为两个阶段
1.事件捕获阶段：捕获阶段是从上向下执行 一直执行到 选定的dom
2.事件冒泡阶段 从dom 元素 一直执行到 最大的dom元素

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

    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>

        <style>
            .father {
                width: 200px;
                height: 200px;
                background-color: orange;
            }

            .son {
                width: 100px;
                height: 100px;
                background-color: red;
            }
        </style>
    </head>

    <body>
```

```
        <div class="father">
            <div class="son"></div>
        </div>



        <script>
            var divs = document.querySelectorAll('div');

            //true 是捕获阶段 执行顺序 是 document->html->body->father->son

            divs[0].addEventListener('click', function() {
                alert(22);
            }, true)

            divs[1].addEventListener('click', function() {
                alert(11);
            }, true)


            //false 是冒泡阶段 执行顺序 是 son->father->body->html->document
            //冒泡 使 父亲和孩子 的行为 是 小盒子放到大盒子里面  如果并列的关系就不会存在 这
个关系



            // divs[0].addEventListener('click', function() {
            //     alert(22);
            // }, false)

            // divs[1].addEventListener('click', function() {
            //     alert(11);
            // }, false)
        </script>

    </body>

</html>
```

当执行 true 阶段 的时候 是进行捕获阶段 当去触发divs[1]的时候 先会触发divs[0] 再去触发 divs[1] 这是父子关系 会发生捕获事件
如果并列的 就不会发生
当执行 false 阶段 的时候 自动进行冒泡阶段 当去触发divs[1]的时候 先会触发divs[0] 再去触发 divs[1] 这是父子关系 会发生冒泡事件 如果并列的 就不会发生

解决冒泡 事件流的默认行为是 冒泡

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
    <style>
        .father {
            width: 200px;
            height: 200px;
            background-color: orange;
        }

        .son {
            width: 100px;
            height: 100px;
            background-color: red;
        }
    </style>
</head>

<body>
    <div class="father">
        <div class="son"></div>
    </div>

    <script>
        var divs = document.querySelectorAll('div');

        //true 是捕获阶段 执行顺序 是 document->html->body->father->son

        // divs[0].addEventListener('click', function() {
        //     alert(22);
        // }, true)

        // divs[1].addEventListener('click', function() {
        //     alert(11);
        // }, true)


        //false 是冒泡阶段 执行顺序 是 son->father->body->html->document

        //由于 事件流的默认情况是 冒泡 所以 false 写或不写 没有什么区别
        divs[0].addEventListener('click', function() {
            alert(22);
        })

        divs[1].addEventListener('click', function(event) {
            alert(11);
            event.stopPropagation(); //阻止冒泡 使停留在这里
        })
    </script>

</body>

</html>
```

阻止事件流动

```
a.addEventListener('click', function(event) {
   //阻止 跳转效果
   event.preventDefault();
 })
//租着a 链接跳转
```

事件委托

事件委托其实是利用事件冒泡的特点， 给父元素添加事件，子元素可以触发

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <ul>
        <li>123</li>
        <li>123</li>
        <li>123</li>
        <li>123</li>
        <li>123</li>
        <li>123</li>
        <li>123</li>
        <li>123</li>
    </ul>

    <script>
        var ul = document.querySelector('ul');

        //利用捕获 事件 来实现 委托 事件
        ul.addEventListener('click', function(event) {
          // 点击li 产生 冒泡 到 ul 身上 触发当前函数
          // event.target 清楚 获得 点击的 那个li
            event.target.style.backgroundColor = 'red';
            console.log(event.target)
        })
    </script>
</body>

</html>
```

滚动事件和加载事件

滚动事件

```
在页面发声滚动的时候会触发这个函数
window.addEventListener("scroll",()=>{
      console.log(1)
})
```

加载事件

```
在页面加载完成之后就触发这个函数
window.addEventListener('load', function() {
            alert(22);
})
```

document.documentElement

```
document.documentElement 返回的是 html 元素
```

offset系列 是以 最近的父级为标准的

```
offset属性 一共包括
1.offsetLeft //只读属性
2.offsetTop // 只读属性
3.offsetWidth
4.offsetHeight

获取位置通过offsetLeft 但是 修改位置通过 dom.style.left

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

  <style>
    *{
      margin: 0;
      padding: 0;
    }
    div{
      width: 200px;
      height: 200px;
      border: 1px solid yellow;
      padding: 10px;
      background-color:red;
```

```
      /* 盒子box-sizing默认的值是content-box */
      /* box-sizing: border-box; */
    }
  </style>
</head>
<body>
  <div></div>
  <script>
    let div = document.querySelector('div')
    console.log(div.offsetWidth)
     // 盒子box-sizing默认的值是content-box
    // offsetWidth = paddiing + border + width
    // 当盒子属性是 box-sizing 修改为 border-box offsetWidth = width
    console.log(div.offsetLeft)

  </script>
</body>
</html>
```

client 系列

```
client 属性 一共包括
1.clientLeft //只读属性
2.clientTop // 只读属性
3.clientWidth
4.clientHeight
clientWidth 这个不包括 border的值
clientWidth 和 width 的 关系 影响因素 border , box-sizing类型

借助clientWidth 可以用来检测 盒子的宽度大小
```

BOM对象

window对象

定时器-延时函数

```
1.定时器的种类
setTimeout(callback,time) 只会触发一次
setinterval(callback,time) 一直触发
setTimeout(callback,time)
setinterval(callback,time)
clearTimeout()
clearInterval()
```

js执行机制

```
js 是单进程 ：代码从头到尾一条一条按照顺序执行下去
js 分为 同步和异步
同步任务都在主线程上执行，形成一个执行栈。
JS 的异步是通过回调函数实现的。
1、普通事件，如 click、resize 等
2、资源加载，如 load、error 等
3、定时器，包括 setInterval、setTimeout 等

js执行机制
先执行执行栈的同步任务
当发现有回调函数（异步程序） 的时候 将这些 放到任务队列当中去
当执行栈中的同步任务执行完了之后 会将 异步任务中的任务加入到 执行栈中
当执行栈再次执行完成之后 会循环去 访问异步任务
```

swiper 插件

```
熟悉官网,了解这个插件可以完成什么需求 https://www.swiper.com.cn/
看在线演示,找到符合自己需求的demo https://www.swiper.com.cn/demo/index.html
查看基本使用流程 https://www.swiper.com.cn/usage/index.html
查看API文档,去配置自己的插件 https://www.swiper.com.cn/api/index.html
```

getAttribute增加类名

```
let div = document.querySelector('.container')
// 会覆盖原来的 样式
// div.setAttribute("class","red")
// 获取当前元素的类名
// console.log(div.getAttribute('class'))
// 删除当前的类名
// div.removeAttribute('class')
```

正则表达式

创建正则表达式

创建正则表达式

```
let reg = /abc/
let reg = new RegExp('abc')
```

检测规则

```
方法1 匹配 验证字符串是否符合正则规则 => 正则.test(检测的字符串)
方法2 捕获 语法规范 正则.exec(捕获的字符串)
返回值类型 1 字符串里面没有符合规则的片段 return null
           2 基础捕获 返回值是一个数组 只捕获前面的一次存在的
```

基础元字符

元字符就是以 符号去替代 文本内容的 符号

```
1 \s 空格
2 \S 非空格
3 \t 制表符
4 \d 表示数字
5 \D 表示非数字符
6 \w 表示字符中有 数字或者字母或者下划线字符
7 \W 表示除了 数字 字母 下划线 以外的 都行
8 . 表示除了换行以外 任意一个字符都行
```

```
const reg = /\s/  // \s 等价于 空格 ' '

const reg1 = /\S/  // 非空格

const reg2 = /\d/

const reg3 = /\D/

const reg4 = /\w/

const reg5 = /\W/

const reg6 = /./

console.log(reg.test('ab c'))
console.log(reg1.test(''))
console.log(reg2.test('123'))
console.log(reg3.test('asda12321'))
console.log(reg4.test('1231asda$'))
console.log(reg5.test('213*'))
console.log(reg6.test('\n'))
```

边界元字符

```
1.^ 表示字符串 开始
2.$ 表示字符串 结束
3.* 表示出现0 - 多次
4.+ 表示出现1 - 多次
5.? 表示出现0 - 1 次
6.{n} 表示出现n次
7.{n,m} 表示出现n-m次
```

```
  // 表示以一个数字开头
const reg = /^\d/
// 表示以一个数字结尾
const reg1 = /\d$/
console.log(reg.test('12'))
console.log(reg1.test('12'))
```

```
// 出现0次 或者多次 数字
const reg2 = /\d*/
console.log(reg2.test('pspada123'))

// 表示字符串 是由0-多个 数字组成
const reg3 = /^\d*$/

console.log(reg3.test('12'))
// 表示字符串 是由1 - 多个 字符串组成
const reg4 = /^\d+$/

console.log(reg4.test('1'))

// 表示字符串只能有0 - 1 数字 个字符串组成
const reg5 = /^\d?$/
console.log(reg5.test('1'))

// 表示字符串只能有3个数字字符组成
const reg6 = /^\d{3}$/
console.log(reg6.test('112'))

// 表示字符串只能有3个数字字符组成
const reg7 = /^\d{3,4}$/
console.log(reg7.test('1122'))
```

贪婪性

```
  贪婪：尽可能捕获最大值
  非贪婪：尽可能的捕获做小数量
  1.\d+  \d+?
  2.\d*  \d*?
  3.\d?  \d??
  4.{n}  {n}?
  5.{n,} {n,}?
  6.{n,m} {n,m}?
      // 尽可能捕获更多的数字
const reg = /\d+/g;
console.log(reg.exec('ad1231442wqe14'))

// 尽可能捕获更少的数字
const reg1 = /\d+?/
console.log(reg1.exec('2asac1242314'))

// 尽可能捕获更多的数字
const reg2 = /\d*/g;
console.log(reg2.exec('ab1231442wqe14'))
console.log(reg2.exec('1231442wqe14'))

// 尽可能捕获更少的数字
const reg3 = /\d*?/
console.log(reg3.exec('1242314'))

//尽可能捕获更多的数字
const reg4 = /\d?/
console.log(reg4.exec('13'))
```

```
    //尽可能捕获更少的数字
    const reg5 = /\d??/
    console.log(reg5.exec('13'))
```

动画效果

动画：借助定时器 去执行相对应的代码 达到动态的效果

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <!--
    动画就是利用定时器实现动态效果
    设定一个定时器 进行操作
   -->
  <style>
    div{
      position: absolute;
      top: 100px;
      left: 100px;
      width: 100px;
      height: 100px;
      background-color: red;
    }
  </style>
</head>
<body>
  <div></div>
  <script>

    let div = document.querySelector('div')

    // 通过offsetX 获取 通过作用在style 修改距离 style.left
    setInterval(()=>{
      div.style.left = div.offsetLeft + 5 + 'px'
      console.log(div.style.left)
    },100)

  </script>
</body>
</html>
```

jQuery

选择器

```
类选择器  $('.div')
$('.container')
id选择器 $('#div')
$('#container')
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
  <style>
    .container {
      width: 100px;
      height: 100px;
      background-color: red;
    }
  </style>
</head>
<body>
  <div class="container">

  </div>
  <script>
    // 简单设置
    $('.container').css('background-color','yellow')
    // 同时设置多个样式
    $('.container').css({
      'background-color':'red',
      'width':'200px',
      'height':'300px'
    })

  </script>
</body>
</html>
```

设置属性css

```
$('div').css('属性', '值')
```

jQuery中的筛选原理

```
通过父亲选孩子 标注写法
$(father).children().eq(index)
$(father children).eq(index)
通过孩子选父亲
$('.div').parent()
兄弟选择器
$('.div').siblings()
后代选择器
$('ul').find('li')
选择最近的一个孩子
$('.div').children()
```

链式编程

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>

  <ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
<body>
  <script>

    $('ul li').click(function() {
      $(this).css('background-color','red').siblings('li').css('background-
color',"")
    })

  </script>
</body>
</html>
```

设置类样式方法

增加类

```
$('.container').addClass('color')
// 等价于 这两者都是在原有的基础上加上新的类名
let div = document.querySelector('.container')
div.classList.add('color')
$('.container').removeClass('color')
```

移除类

```
$('.container').removeClass('color')
// 等价于
div.classList.remove('color')
```

切换类

```
// 切换类 触发一次 生成 触发第二次就会自动被移除
$('button').click(()=>{
  $('.container').toggleClass('color')
})
```

获取类名

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src = '../jquery.min.js'></script>
  <style>
    .overview{
      width: 100px;
      height: 100px;
    }
  </style>
</head>
<body>
    <div class="overview">
      app
    </div>

</body>
    <script>

       let name = $('.overview').attr('class') //
       console.log(name)
    </script>
</html>
```

jQuery效果

1.演示与隐藏

```
    // 显示与隐藏
    $('button').eq(0).click(()=>{
      $('div').show()
    })
    $('button').eq(1).click(()=>{
      $('div').hide()
    })
    $('button').eq(2).click(()=>{
      $('div').toggle()
    })
```

2.滑动效果

```
// 滑动效果
    $('button').eq(0).click(()=>{
      $('div').slideDown()
    })
    $('button').eq(1).click(()=>{
      $('div').slideUp()
    })
    $('button').eq(2).click(()=>{
      $('div').slideToggle()
    })
```

3.淡入淡出

```
// 淡入淡出
    $('button').eq(0).click(()=>{
      $('div').fadeIn()
    })
    $('button').eq(1).click(()=>{
      // $('div').fadeIn()
      $('div').fadeOut()
    })
    $('button').eq(2).click(()=>{
      $('div').fadeToggle()
    })
```

jQuery 效果演示

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
  <script src="jquery.min.js"></script>
  <style>
    .container {
      width: 100px;
      height: 100px;
      background-color: red;
    }

    .box{
      width: 50px;
      height: 50px;
      background-color:yellow;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box"></div>
  </div>
  <button>show</button>
  <button>hide</button>
  <button>toggle</button>

  <script>

    $('button').eq(0).click(()=>{
      $('.box').show("slow","linear",function(){
        console.log(1)
      })
    })
    $('button').eq(1).click(()=>{
      $('.box').hide("slow","linear",function(){
        console.log(2)
      })
    })
    $('button').eq(2).click(()=>{
      $('.box').toggle("slow","linear",function(){
        console.log(3)
      })
    })

  </script>
</body>
</html>
```

jQuery 事件切换 鼠标事件

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
```

```
  <style>

    .container{
      width: 100px;
      height: 100px;
      background-color: red;
    }

  </style>
</head>
<body>

  <div class="container">

  </div>

  <script>

    $('div').mouseover(function(){
      console.log(1)
    })
    $('div').mouseenter(function(){
      console.log(2)
    })
    $('div').mouseleave(function(){
      console.log(3)
    })


  </script>
</body>
</html>
```

固有属性

prop

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
  <style>
    .container {
      width: 100px;
      height: 100px;
      background-color: red;
    }
  </style>
</head>
<body>
  <a href="#"></a>
```

```
  <script>
    //拿到标签的固有属性
    // 固有属性 并不是 指 style
    // 固有属性 a 标签中的 href type 类型等等
    const a = $('a').prop('href')
    console.log(a)

    // 设置固定属性的值

    const a = $('a').prop('href','123')
    console.log(a)
  </script>
</body>
</html
```

attr

```
```

数据缓存

```
```

文本内容

修改文本内容

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
  <style>
    .container {
      width: 100px;
      height: 100px;
      border: 1px solid red;
    }
  </style>
</head>
<body>

  <div class="container"></div>
  <script>
    // 修改文本内容
    $('.container').html('123')
    // 不能解析 标签
    $('.container').text('<a href = "#"></a>')
    // 能解析 标签
    $('.container').html('<a href = "#">12</a>')
```

```
  </script>
</body>
</html>
```

获取文本内容

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <input type="text" placeholder="123">
  <script>
    // 给input 表单输入内容
    const val = $('input').val('312 ')
  </script>
</body>
</html>
```

元素操作

遍历元素

```

```

增加元素

```
内部增加元素 和 父级 是 父子关系
$('ul').append(li)
增加到最前面
$('ul').prepend(li)
外部增加元素
增加到最外面 和 父级是 兄弟关系
$('ul').before(li)
$('ul').after(li)
```

删除元素

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
  <script src="jquery.min.js"></script>
  <style>
    .container{
      height: 100px;
      width: 100px;
      background-color: red;
    }
    .box{
      height: 50px;
      width: 50px;
      background-color:yellow;
    }
    .box1{
      height: 50px;
      width: 50px;
      background-color:green;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box">box</div>
    <div class="box1">123</div>
  </div>

  <ul>
    <li>1</li>
    <li>2</li>
  </ul>
  <script>
    // 删除元素
    // 删除父节点 和 所有的子节点
    // $('.container').remove()
    // 只删除自己的字节点
    // $('.container').empty()
    // $('.container').html("");

    // 删除某个孩子 和 eq搭配
    // $('.container').children().eq(0) 先选择出来相对应的孩子
    //  $('.container').children().eq(1).remove()
  </script>
</body>
</html>
```

创建元素

```
// 创建一个元素
   $('<li></li>')
```

尺寸修改

只算width height

```
width() height()
当列表不传数 返回值是 width 设置大小
let width = $('.container').Width()
当列表传数 表示修改height 大小
let height = $('.container').height(200)
```

包含pading

```
  let width = $('.container').innerWidth()

  let height = $('.container').innerHeight()

  console.log(width,height)
```

包含pading border

```
   let width = $('.container').outerWidth()
   let height = $('.container').outerHeight()
   console.log(width,height)
```

位置

offset

```
    1.offset 不是相对 父级 是相对 dom 元素而言
    offset({top:10,left:10}) 可以设置 属性
    let left = $('.container').offset().left
    console.log(left)
    let Left = $('.son').offset().left
    console.log(Left)
```

position

```
    2 position 相对于自己的父级而言
    let top1 = $('.container').position().top
    console.log(top1)
    let Top = $('.son').position().top
    console.log(Top)
```

scrollTop()/scrollLeft()

```

```

事件

注册事件

```
click mouseover mouseleave mousemove blur focus keydown keyup
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
  <style>
    .container{
      height: 100px;
      width: 100px;
      background-color: red;
    }
  </style>
</head>
<body>
  <div class="container"></div>
  <input type="text">
  <script>
    // 注册点击事件
    $('.container').mousemove(()=>{
      console.log(1)
    })
    $('input').focus(()=>{
      console.log(1)
    })
    $('input').keydown(()=>{
      console.log(2)
    })
    $('input').keyup(()=>{
      console.log(3)
    })

  </script>
</body>
</html>
```

on注册事件

```
    // 使用 on 注册 点击事件
    // $('.container').on('click',()=>{
    //   console.log('click')
    // })
```

```
    // 优势1 可以同时添加个事件
    // $('.container').on({
    //   click: ()=>{
    //     console.log('click')
    //   },
    //   mouseover:()=>{
    //     console.log('mouseover')
    //   }
    // })
    // 委派效果 利用子选择器
    // 给后代 同时增加一样的属性 减少了循环的次数
    // $('.container').on('click','.son',()=>{
    //   console.log("son")
    // })
    // 优势3 动态创建的元素
    // $('.container').on('click','p',()=>{
    //   console.log('动态创建')
    // })

    // $(".container").append($("<p>我是动态创建的p</p>"));
```

移除on注册的事件

```
$("p").off() // 解绑p元素所有事件处理程序

$("p").off( "click")  // 解绑p元素上面的点击事件 后面的 foo 是侦听函数名

$("ul").off("click", "li");   // 解绑事件委托
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<style>
  .container {
    height: 500px;
    width: 500px;
    background-color: red;
  }
  .son{
    margin: 10px;
    height: 50px;
    width: 50px;
    background-color:black;
  }
</style>
<body>
```

```
  <div class="container">
    <div class="son">son</div>
    <div class="son">son</div>
    <div class="son">son</div>
  </div>

  <script>
    // 解除注册的点击事件
    $('.container').on('click',()=>{
      console.log('click')
    })

    $('.container').off('click')

    // $('.container').on({
    //   click: ()=>{
    //     console.log('click')
    //   },
    //   mouseover:()=>{
    //     console.log('mouseover')
    //   }
    // })
    // 委派效果 利用子选择器
    // 给后代 同时增加一样的属性 减少了循环的次数\
    // 解除委派的 点击事件
    $('.container').on('click','.son',()=>{
      console.log("son")
    })
    $('.container').off('click','.son')
    // 优势3 动态创建的元素
    // $('.container').on('click','p',()=>{
    //   console.log('动态创建')
    // })

    // $(".container").append($("<p>我是动态创建的p</p>"));

  </script>

</body>
</html>
```

事件对象

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js" ></script>
  <style>
    .father{
```

```
      height: 100px;
      width: 100px;
      background-color:red;
    }
    .son{
      width: 50px;
      height: 50px;
      background-color:yellow;
    }
  </style>
</head>
<body>
  <div class="father">
    <div class="son"></div>
  </div>
  <script>
    // 使用事件对象 event
    $('.son').on('click',(event)=>{
      console.log(1)
      // 去除冒泡行为
      event.stopPropagation();
    })
    $('.father').on('click',()=>{
      console.log(2)
    })

    // 合理利用event 进行
    // 默认行为 event.preventDefault()
  </script>
</body>
</html>
```

js ES补充

1.let const var

```
var 存在作用域提升
console.log(age) // undefined
var age = 10
console.log(name) // ReferenceError
let name = 'zs'
// 定义变量的时候 必须赋初始值
 const gender = 'man'
```

2. 解构赋值

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>

    //进行多次赋值
    let [a,b,c] = [1,2,3]
    console.log(a,b,c)

    // 定义对象时
    let person = {
      name:'zs',
      age: 20
    }

    // 进行 解构 内部的 相对应的对象
    let {name} = person
    console.log(name)
    console.log(person)

    // 起别名的时候
    let {name:res} = person
    console.log(res)

  </script>
</body>
</html>
```

3.模板字符串

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    // 模板字符串的使用
    let name = `张三`
    console.log(name)
    // 使用变量形式 ${}
    let Age = 20
    let age = `my age is ${Age}`
    console.log(age)
    let result = {
     name: 'zhangsan',
     age: 20,
     sex: '男'
```

```
 }
 console.log(result)
 let html = ` <div>
     <span>${result.name}</span>
     <span>${result.age}</span>
     <span>${result.sex}</span>
 </div> `;
 console.log(html)

  </script>
</body>
</html>
```

4.展开对象

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    let arr1 = [1,2,3]
    let arr2 = [4,5,6]
    // 进行数组合并
    let arr3 = [...arr1,...arr2]
    console.log(arr3)

    // 利用push函数进行合并
    arr1.push(...arr2)
    console.log(...arr1)
  </script>
</body>
</html>
```

5.数组迭代

1.forEach（数组迭代 输出）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
```

```
    let arr = [1,2,3,4]

    // 使用forEach() 进行遍历
    // 当只给一个参数的时候
    // arr.forEach(function(element){
    //   console.log(element)
    // })
    // 当给两个参数的时候
    // 第一个参数 是数值 第二个参数是 索引号
    arr.forEach(function(element,index){
      console.log(element,index)
    })
    // 当传送的是三个参数的时候
    // 第一个参数 是 element 第二个参数 是 index 第三个参数 是 数组
    arr.forEach(function(element,index,arr){
      console.log(element,index,arr)
    })

    // 使用forEach 求和
    let sum = 0

    arr.forEach( function ( element,index,arr){
      if(arr[index] === element){
        sum+=element
      }
    })

    console.log(sum)
  </script>
</body>
</html>
```

2.map （按照某个新的规则生成一个新的数组）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    let arr = [1,2,3]

    // map 就是将 数组arr 按照一个指定的规则 返回生成一个新的数组
    let Arr = arr.map(function(element) {
      // 新数组的 规则是 元素组的 2 倍
      // 这个return 是 必须要写的
      return element * 2
    })
    console.log(Arr)
  </script>
```

```
</body>
</html>
```

3.filter （过滤器使用）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    // 过滤器的使用 作用是筛选 元素
    // 满足条件的是一个新的数组
    let arr = [1,2,3]

    let Arr = arr.filter(function(element,index,arr){
      // 筛选出 是 偶数的 数
      console.log(element,index,arr)
      return element%2 === 0

    })

    console.log(Arr)

  </script>
</body>
</html>
```

4.some (判断某个条件是否满足)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    // some 数组 中 只要有一个满足就可以
    // 用来判断 数组 中 是否 满足 某个条件
    let arr = [1,2,3]
    let Arr = arr.some(function(element) {
      return element >= 3
    })
    console.log(Arr)
```

```
    // 与之相应的 是 every
    // 只有所有的 元素都满足 条件才能 返回true 否则 返回false

    let Arr1= arr.every(function(element){
      return element >= 1
    })

    console.log(Arr1)

  </script>
</body>
</html>
```

5. reduce （进行去重操作）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    // 使用reduce 方法
    let arr = [1,2,3,4,1]
    // reduce 写法
    // return 的值 返回给 pre
    // 第一次的 pre 是 来自 init
    // arr.reduce(function(pre,cur,index,arr){
    //   return xxx
    // },init)

    // let res = arr.reduce(function(pre,cur,index,arr){
    //   console.log(pre,cur,index,arr)
    //   return cur
    // },[])

    // 去重操作
    let Arr = arr.reduce(function(pre,cur){
      // pre.indexOf(cur) === -1 && pre.push(cur)
      // 如果 pre 中 不存在 cur的话 将 cur 存到 pre数组中去
      if(pre.indexOf(cur) === -1){
        pre.push(cur)
      }
      // console.log(pre,cur)
      return pre
    },[])

    console.log(Arr)

  </script>
</body>
```

```
</html>
```

06 set（使Set方法去除元素）

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>

      let arr = [1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10]

      // 将数组中的重复出现的元素去除
      // 将数组 传入Set当中去
      const ans = new Set(arr)

      // Set 增加元素
      ans.add(20)

      // Set 删除z指定元素
      ans.delete(8)

      // 判断是否存在某个元素

     console.log(ans.has(10))

      // 将Set 转换成一个数组
      // 转换成数组 将 一个数据结构 丢到 Array.from() 中去

      let newArr = Array.from(ans)

      console.log(newArr)

     console.log(ans)

  </script>
</body>
</html>
```

for in 遍历数组

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Document</title>
  <script src="jquery.min.js"></script>
</head>
<body>
  <script>
    // 使用 for in 遍历数组
    let arr = [1,2,3,4]
    for(let index in arr ){
      console.log(index)
    }

    let Obj = {
      name:'zs',
      age:10
    }

    for(let it in Obj){
      // 输出对象的 值得时候 使用得 是 []
      console.log(Obj[it])
    }
  </script>
</body>
</html>
```

CSS

选择器的种类

1.类选择器

```
.类名{

}
```

2.属性选择器

```
#id{

}
```

3.标签选择器

```
p{

}
```

文字属性

1. 字体系类

```
font-family:'微软雅黑' 字体样式
```

2.字体大小

```
font-size:'10px' 字体大小
```

3.字体粗细

```
font-weight: bold
bold 定义粗体
normal 不加粗 （默认值）
数字表示 100-700
```

4.文字样式

1.文本颜色

```
文本文字颜色
color:'red' rgb()
```

2 .对齐文本

```
text-align:'center'
left:左边
right:右边
center:中间
```

3.装饰文本

```
text-decoration
none:没有装饰线
underline:下划线
overline:上划线
line-through:删除线
```

4.文本缩进

```
text-indent
缩进单位
```

5.行间距

```
line-height:一般和盒子高度一样
line-height:height
```

符合选择器

1.后代选择器

```
类名1 类名2{
    将类1中的类2选择出来
}
div ul{
    将div中的ul进行选择出来
}
```

2.子代选择器

```
div>p{

}
div中可能有很多孩子
将他的亲儿子p选择出来
```

3.并集选择器

```
并集选择器
div,p{
    用逗号隔开 意思是和 也就是说将div和p都给选择出来
}
```

4.伪类选择器

```
a:hover{
    经过a的时候将a选择出来
    就是伪类选择器 使用:
}
a:nth-child(1){
    选择第一个孩子
}
```

5. :focus选择器

```
当获得焦点的时候 就会触发css样式
```

css显示模式

1.块元素

```
块元素 直接占一行
display:block
```

2.行内元素

```
不是占一行
display:inline
```

3.行内块元素

```
融合块元素和行内元素的特点
display:inline-block;
```

css背景

1.背景颜色

```
背景颜色
 background-color:颜色值;
```

2.背景图片

```
 background-image:
```

3.背景平铺

```
background-repeat
no-repeat:不平铺
```

4.背景图片的位置

```
 background-position:x,y;
```

5.背景附着

```
 background-attachment : scroll | fixed
 scroll:背景随着图片滚动
 fixed:背景固定
```

圆角边框

```
border-radius:length;
```

盒子阴影

```
    盒子阴影
      box-shadow:
      1.h-shadow 水平阴影 正值的时候表示 向右 负值的时候表示向左
      2.v-shadow 垂直阴影 正值的时候表示 向上 负值的时候表示向下
      3.blur     模糊的距离
      4.spread   阴影的尺寸 阴影面积的尺寸大小
      5.color    阴影的颜色
      6.inset    inset 表示的是内部阴影 outset 表示的是 外部阴影
```

文字阴影

```
    文字阴影
      box-shadow:
      1.h-shadow 水平阴影 正值的时候表示 向右 负值的时候表示向左
      2.v-shadow 垂直阴影 正值的时候表示 向上 负值的时候表示向下
      3.blur     模糊的距离
      4.color    阴影的颜色
```

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

```
    *{
      margin: 0;
      padding: 0;
    }
    .container {
      margin:100px auto;
      width: 100px;
      height: 100px;
      background-color: red;
      box-shadow: 10px -10px 10px 10px orange inset;
    }
    p{
      text-shadow: 10px 20px 5px orange;
    }
  </style>
</head>
<body>
  <!--
    盒子阴影
      box-shadow:
      1.h-shadow 水平阴影 正值的时候表示 向右 负值的时候表示向左
      2.v-shadow 垂直阴影 正值的时候表示 向上 负值的时候表示向下
      3.blur     模糊的距离
      4.spread   阴影的尺寸 阴影面积的尺寸大小
      5.color    阴影的颜色
      6.inset    inset 表示的是内部阴影 outset 表示的是 外部阴影
   -->
     <!--
    文字阴影
      box-shadow:
      1.h-shadow 水平阴影 正值的时候表示 向右 负值的时候表示向左
      2.v-shadow 垂直阴影 正值的时候表示 向上 负值的时候表示向下
      3.blur     模糊的距离
      4.color    阴影的颜色
   -->
  <div class="container">
  </div>

  <p>123142</p>
</body>
</html>
```

浮动

产生浮动

```
float:left
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
  <style>
    *{
      margin: 0;
      padding: 0;
    }
    .box1{
      float: left;
      height: 100px;
      width: 100px;
      background-color: red;
    }
   .box2{
     float: left;
    height: 100px;
      width: 100px;
      background-color:yellow;
    }
    .box3{
      float: left;
      height: 100px;
      width: 100px;
      /* float: right; */
      background-color:orange;
    }
  </style>
</head>
<body>
  <div>
    <div class="box1"></div>
    <div class="box2"></div>
    <div class="box3"></div>
  </div>
</body>
</html>
```

消除浮动

```
  <!-- 清除浮动 clear -->
<div style="width: 100px; height: 100px ;background-color:purple; clear: left;">
</div>
<div style="width: 100px; height:100px;background-color: red;">1212</div>
```

定位

定位模式

```
relative 相对定位
absolute 绝定定位
fixed    固定定位
```

偏移量

```
top left right bottom
```

粘性定位

```
position:sticky
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .header{
      position: sticky;
      top: 0;
      left: 100px;
      height: 100px;
      width: 100px;
      background-color: red;
    }
    .body{
      height: 1000px;
      width: 100px;
      background-color:orange;
    }
  </style>
</head>
<body>
  <div class="header"></div>
  <div class="body"></div>
</body>
</html>
```

固定定位

```
固定定位是不占位置
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    *{
```

```
      margin: 0;
      padding: 0;
    }
    .header{
      position: fixed;
      top: 0;
      left: 100px;
      height: 100px;
      width: 100px;
      background-color: red;
    }
    .body{
      height: 1000px;
      width: 100px;
      background-color:orange;
    }
  </style>
</head>
<body>
  <div class="header"></div>
  <div class="body"></div>
</body>
</html>
```

z-index

```
位置重叠
z-index:1
数值越大 越在那上面
```

元素的隐藏和显示

display

```
      /* 不占位置 */
      /* display: none;  */
      /* 占位置 */
      /* visibility: hidden; */
      /* 溢出隐藏 */
      /* overflow: hidden; */
```

css新增属性

伪类选择器

::before

```
在元素内部的前面插入内容
```

::after

在元素内部后面插入内容

css过渡

谁做过渡给谁加

transition: 要过渡的属性 花费时间 运动曲线 何时开始;
1.属性 ： 想要变化的 css 属性， 宽度高度 背景颜色 内外边距都可以 。如果想要所有的属性都变化过渡， 写一个all 就可以。
2.花费时间： 单位是 秒（必须写单位） 比如 0.5s
3.运动曲线： 默认是 ease （可以省略）
4.何时开始 ：单位是 秒（必须写单位）可以设置延迟触发时间 默认是 0s （可以省略）

属性：style中的属性 width height background-color 等等属性

花费时间 完成指定的属性所需要的时间

运动曲线 linear 平均 ease 逐渐慢下来 ease-in 加速 ease-out 减速

何时开始 就是指定时间 开始

字体图标

字体图标的地址
阿里 iconfont 字库 http://www.iconfont.cn/ 推荐指数 ★★★★★

```
@font-face {
font-family: 'icomoon';
src: url('fonts/icomoon.eot?7kkyc2');
src: url('fonts/icomoon.eot?7kkyc2#iefix') format('embedded-opentype'),
url('fonts/icomoon.ttf?7kkyc2') format('truetype'),
url('fonts/icomoon.woff?7kkyc2') format('woff'),
url('fonts/icomoon.svg?7kkyc2#icomoon') format('svg');
font-weight: normal;
font-style: normal;
}
```

界面样式

```
li {
    cursor: pointer;
}
default 小白
pointer 小手
move 移动
text 文本
not-allowed 禁止
input 去除高亮
input{
    outline:none
}
textarea{ resize: none;}
```

vertical-align 属性应用

```
 vertical-align : baseline | top | middle | bottom
 baseline 默认 基线对齐
 top 元素顶端 和 最高顶端对齐
 middle 把此元素发在父元素中部
 bottom 把元素的顶端和最低元素顶端对齐
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    img{
      width: 500px;
      height: 500px;
      /* vertical-align: baseline|top|bottom|middle; */
    }
  </style>
</head>
<body>
  <div>
    <img src="./01.loader调用的过程.png" alt="">123
  </div>
</body>
</html>
```

溢出文字处理

```
/*1. 先强制一行内显示文本*/  white-space: nowrap;  （ 默认 normal 自动换行）
/*2. 超出的部分隐藏*/  overflow: hidden;
/*3. 文字用省略号替代超出的部分*/  text-overflow: ellipsis;
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    div{
      width: 100px;
      height: 100px;
      background-color: red;
      /* 溢出隐藏 */
      overflow: hidden;
      /* 利用省略号 替代 */
      text-overflow: ellipsis;
      /* overflow text-overflow 要同时使用 */
    }
  </style>
</head>
<body>
  <div>
    21321313123123
  </div>
</body>
</html>
```

属性选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    /* 选出类名是 val_1 */
    div[class="val_1"]{
      background-color:red;
    }
    /* 选出 类名 以 val 开头的 */
    div[class^="val"]{
      width: 100px;
      height: 100px;
    }
    /* 选出类名以5结尾的 */
    div[class$="5"]{
      width: 100px;
      height: 100px;
      background-color:purple;
    }
       /* 选出类名带有5的 */
    div[class*="5"]{
```

```
      width: 100px;
      height: 100px;
      background-color:purple;
    }
  </style>
</head>
<body>
  <div class="val_1">1</div>
  <div class="val_2">2</div>
  <div class="val_3">3</div>
  <div class="val_4">4</div>
  <div class="val_5">5</div>

</body>
</html>
```

2D

2D旋转

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    div{
      width: 100px;
      height: 100px;
      background-color: red;
      /* 谁做过渡给谁加 */
      transition:all 2s ease;
    }
    div:hover{
      transform: rotate(45deg);
    }
  </style>
  <title>Document</title>
</head>
<body>
  <div></div>
</body>
</html>
```

缩放效果

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Document</title>
  <style>
    div{
      width: 100px;
      height: 100px;
      background-color: red;
      transition: all 2s ease;
    }
    div:hover{
      /*缩放效果 不会去影响其他的盒子 */
      transform: scale(2);
    }
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```

盒子移动

盒子移动不会去影响其他盒子的位置

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            *{
                margin: 0;
                padding: 0;
            }
            div{
                height: 100px;
                width: 100px;
                background-color: red;
                /* 只移动X轴 */
                transform:translateX(30px);
                /* 只移动Y轴 */
                transform:translateY(30px);
                /* 只移动X,Y轴 */
                transform: translate(30px,40px);
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

盒子的中心点

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            /* 旋转过程中不会影响其他的盒子的位置 */
            .content{
                height: 100px;
                width: 100px;
                background-color: red;
                /* 转换中心点 */
                transform-origin: left bottom;
                transition: all .5s ease;
            }
            .content:hover{
                transform: rotate(45deg);
            }
            .body{
                height: 100px;
                width: 100px;
                background-color:yellow;
            }
        </style>
    </head>
    <body>
        <div class="content"></div>
        <div class="body"></div>
    </body>
</html>
```

动画

1.定义动画

```
制作动画分为两步：
1.先定义动画 @keyframes 名称
2.再使用（调用）动画
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
```

```
        <style>
            @keyframes move{
                0%{
                    transform: translate(0,0);
                }
                100%{
                    transform: translate(100px,100px);
                }
            }
            div{
                width: 100px;
                height: 100px;
                background-color:red;
                animation-name: move;
                animation-duration:10s;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

2.速度曲线

steps(10) 表示的是 只会走10 步就会到达目的地

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            *{
                margin: 0;
                padding: 0;
            }
            @keyframes move{
                0%{
                    transform:translate(0,0)
                }
                100%{
```

```
                    transform:translate(100px,100px)
                }
            }
            div{
                width: 100px;
                height: 100px;
                background-color: red;
                animation-name: move;
                animation-duration:5s;
                animation-timing-function: ease;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

3 执行次数

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    *{
      margin: 0;
      padding: 0;
    }
    @keyframes move{
      0%{
        transform:translate(0,0)
      }
      100%{
        transform:translate(100px,100px)
      }
    }
    div{
      width: 100px;
      height: 100px;
      background-color: red;
      animation-name: move;
      animation-duration:2s;
      animation-timing-function: steps(3);
      /* 表示无数次 */
      animation-iteration-count: infinite;
    }
  </style>
</head>
<body>
  <div></div>
</body>
```

```
</html>
```

4. animation-fill-mode

```
执行结束之后 停放的位置
forwards:停放在最后的位置
backwards:返回到原来位置
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            *{
                margin: 0;
                padding: 0;
            }
            @keyframes move{
                0%{
                    transform:translate(0,0)
                }
                100%{
                    transform:translate(100px,100px)
                }
            }
            div{
                width: 100px;
                height: 100px;
                background-color: red;
                animation-name: move;
                animation-duration:2s;
                animation-timing-function: steps(3);
                animation-fill-mode: forwards;
            }
        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

5.返回的方向

```
animation-direction:alternate|normal
默认值是normal  表示的是 从头开始
alternate 表示的是 从反过来执行
通常和  animation-iteration-count: infinite; 一起使用
```

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    *{
      margin: 0;
      padding: 0;
    }
    @keyframes move{
      0%{
        transform:translate(0,0)
      }
      100%{
        transform:translate(100px,100px)
      }
    }
    div{
      width: 100px;
      height: 100px;
      background-color: red;
      animation-name: move;
      animation-duration:2s;
      animation-timing-function: steps(3);
      /* 表示无数次 */
      animation-iteration-count: infinite;
      animation-fill-mode: forwards;
      animation-direction: alternate;
    }
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```

3D旋转

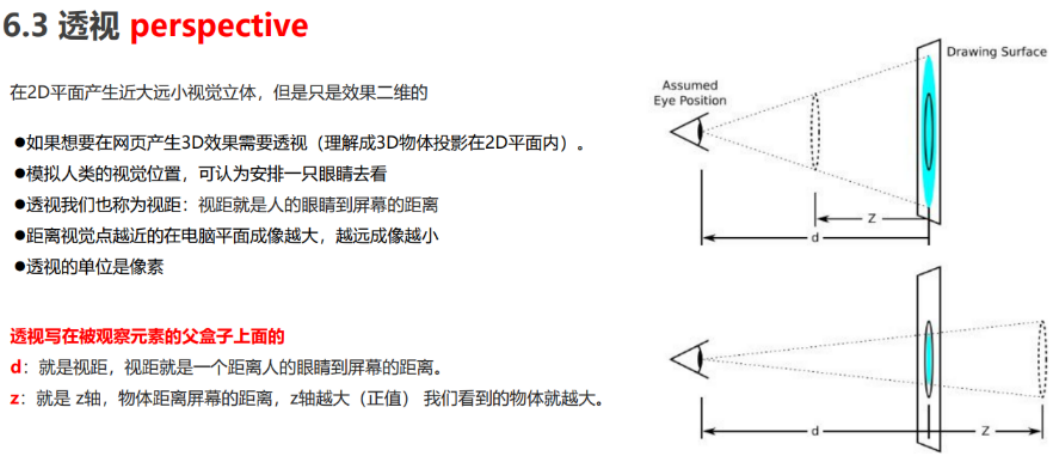
1.偏移量

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            *{
                margin: 0;
                padding: 0;
            }
```

```
            .container {
                width: 100px;
                height: 100px;
                background-color: red;
                /* 3D效果 转换 */
                /* 进行位置偏移 */
                transform: translate3d(100px,100px,100px);
            }
        </style>
    </head>
    <body>
        <div class="container"></div>
    </body>
</html>
```

2.旋转角度

透视介绍



旋转轴的确定 X，Y，Z

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
         body {
            /* 给旋转的父盒子加透视 */
            /* 加了透视 有显示效果出现 */
            perspective: 500px;
        }

        div {
            height: 100px;
```

```
            width: 100px;
            display: block;
            margin: 100px auto;
            transition: all 10s;
        }
        div:hover {
            transform: rotateX(180deg);
        }
  </style>
</head>
<body>
  <div>
    <img src="./pig.jpg" alt="">
  </div>
</body>
</html>
```

旋转Y轴

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    body {
       /* 给旋转的父盒子加透视 */
       /* 加了透视 有显示效果出现 */
       perspective: 500px;
   }

   div {
       height: 100px;
       width: 100px;
       display: block;
       margin: 100px auto;
       /* 过渡效果 */
       transition: all 10s;
   }

   div:hover {
     /* 旋转Y轴 */
       transform: rotateY(180deg);
   }
  </style>
</head>
<body>
  <div>
    <img src="./pig.jpg" alt="">
  </div>
</body>
```

```
</html>
```

旋转Z轴

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

  <style>

    body {

        /* 给旋转的父盒子加透视 */
        /* 加了透视 有显示效果出现 */
        perspective: 500px;

    }

    div {

        height: 100px;
        width: 100px;
        display: block;
        margin: 100px auto;
        /* 过渡效果 */
        transition: all 10s;

    }

    div:hover {
      /* 旋转Y轴 */
        transform: rotateZ(180deg);
    }

  </style>

</head>
<body>
  <div>
    <img src="./pig.jpg" alt="">
  </div>
</body>
</html>
```

旋转综合

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>

    img{
      display: block;
      margin: 100px auto;
      /* background-color: red; */
      transition: all 5s;
    }
    body{
      perspective: 300px;
    }

    img:hover{
      transform: rotate3d(1,1,1,45deg);
    }

  </style>
</head>
<body>
  <div>
    <img src="./pig.jpg" alt="">
  </div>
</body>
</html>
```

3.3D呈现 transfrom-style

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    *{
      margin: 0;
      padding: 0;
    }
    .father{
      position: relative;
      margin: 100px auto;
      width: 300px;
      height: 300px;
      background-color:red;
      /* 控制子盒子是否右3d效果 */
      /* 让子盒子有3d效果 */
      transform-style:  preserve-3d;
      perspective: 500px;
```

```
      transition: all 1s ease;
    }
    .father:hover{
      transform:rotateY(60deg)
    }
    div[class^=box]{
      position: absolute;
      top: 0;
      left: 0;
      width: 100%;
      height: 100%;
      background-color:yellow
    }
    div[class$=box2]{
      transform:rotateX(60deg);
      background-color:purple;
    }
  </style>
</head>
<body>
  <div class="father">
    <div class="box1"></div>
    <div class="box2"></div>
  </div>
</body>
</html>
```

```
  .father{
      position: relative;
      margin: 100px auto;
      width: 300px;
      height: 300px;
      background-color:red;
      /* 控制子盒子是否右3d效果 */
      /* 让子盒子有3d效果 */
      transform-style:  preserve-3d; //让子孩子有3d显示
      perspective: 500px; // 加上透视效果
      transition: all 1s ease; // 加上过渡 作用是 father 盒子选转 谁做过渡给谁加
    }
```

```
.father:hover{
      transform:rotateY(60deg)
 }
//整体旋转
```

```
div[class$=box2]{
      transform:rotateX(60deg);
      background-color:purple;
}
// 给 第二个子盒子 进行旋转
```

flex布局

1.常见的父项属性

- flex-direction：设置主轴的方向
- justify-content：设置主轴上的子元素排列方式
- flex-wrap：设置子元素是否换行
- align-content：设置侧轴上的子元素的排列方式（多行）
- align-items：设置侧轴上的子元素排列方式（单行）
- flex-flow：复合属性，相当于同时设置了 flex-direction 和 flex-wrap

flex-direction

```
属性设置
row 默认值是 从左到右
row-reverse 从右到左
column 从上到下
column-reverse 从下到上
```

justify-content

设置主轴的子元素排列方式

属性值	说明
flex-start	默认值 从头部开始 如果主轴是x轴，则从左到右
flex-end	从尾部开始排列
center	在主轴居中对齐（如果主轴是x轴则 水平居中）
space-around	平分剩余空间
space-between	先两边贴边 再平分剩余空间（重要）

flex-wrap

设置是否换行

```
nowrap默认值，不换行
wrap换行
```

align-items（单行）

该属性是控制子项在侧轴（默认是y轴）上的排列方式 在子项为单项（单行）的时候使用
就是一行当中 有很多元素 其中某一个元素 在 y 轴 上的排列

属性值	说明
flex-start	从上到下
flex-end	从下到上
center	挤在一起居中（垂直居中）
stretch	拉伸 （默认值 ）

align-content（多行）

align-content 设置子项在侧轴上的排列方式 并且只能用于子项出现 换行 的情况（多行），在单行下是没有效果的。

属性值	说明
flex-start	默认值在侧轴的头部开始排列
flex-end	在侧轴的尾部开始排列
center	在侧轴中间显示
space-around	子项在侧轴平分剩余空间
space-between	子项在侧轴先分布在两头，再平分剩余空间
stretch	设置子项元素高度平分父元素高度

flex-flow

flex-flow 属性是 flex-direction 和 flex-wrap 属性的复合属性

flex布局子项常见属性

- flex 子项目占的份数
- align-self 控制子项自己在侧轴的排列方式
- order属性定义子项的排列顺序（前后顺序）

flex属性

```
flex 属性定义子项目分配剩余空间，用flex来表示占多少份数
.item{
   flex : 1;
}
```

align-self

控制子项自己在侧轴上的排列方式

```
span:nth-child(2) {
/* 设置自己在侧轴上的排列方式 */
    align-self: flex-end;
}
```

order

数值越小，排列越靠前，默认为0。

```
.item {
    order: 0;
}
```

rem布局

rem 单位

```
rem 是一个相对单位 是相对 html 而言的
比如，根元素（html）设置font-size=12px; 非根元素设置width:2rem; 则换成px表示就是24px。
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            html{
                font-size: 12px;
```

```
            }
            div{
                font-size: 2rem;
            }
        </style>
    </head>
    <body>
        <div>
            font-size 大小
        </div>
    </body>
</html>
```

媒体查询

```
@media screen and (min-width: 700px){
    div{
        background-color: orange;
    }
}

@media screen and (max-width: 600px) {
    div{
        background-color:blue;
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            html{
                font-size: 2px;
            }


            div{
                width: 100rem;
                height: 100rem;
                background-color: red;
                margin: 100px auto ;
            }

            @media screen and (min-width: 700px){
                div{
                    background-color: orange;
                }
            }
```

```
            @media screen and (max-width: 600px) {
                div{
                    background-color:blue;
                }
            }

        </style>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

less

- Less 变量
- Less 编译
- Less 嵌套
- Less 运算

less变量

```
@变量名:值;
```

```
@color:pink;
```

引入css文件

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <link rel="stylesheet" href="less.css">
    </head>
    <body>
        <div></div>
    </body>
</html>
```

创建less文件

```less
@Color:red;
@Size:12px;

html{
  font-size: @Size;
}

div{
  width: 20rem;
  height: 20rem;
  background-color: @Color;
}
```

生成对应的css文件

```css
html {
  font-size: 12px;
}
div {
  width: 20rem;
  height: 20rem;
  background-color: red;
}
```

less嵌套

引入css文件

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <link rel="stylesheet" href="嵌套.css">
    </head>
    <body>
        <div>
            <div class="son"></div>
        </div>
    </body>
</html>
```

创建一个less文件

```less
@color1:red;
@color2: yellow;
@Szie:12px;
html{
```

```
    font-size: @Szie;
}

div{
    width: 20rem;
    height: 20rem;
    background-color:@color2;
    .son{
        width: 10rem;
        height: 10rem;
        background-color:@color1;
    }
    &:hover{
        background-color:@color1;
    }
}
```

生成对应的css文件

```
html {
    font-size: 12px;
}
div {
    width: 20rem;
    height: 20rem;
    background-color: yellow;
}
div .son {
    width: 10rem;
    height: 10rem;
    background-color: red;
}
div:hover {
    background-color: red;
}
```

当如果遇见 （交集|伪类|伪元素选择器）
内层选择器的前面没有 & 符号，
则它被解析为父选择器的后代；
如果有 & 符号，它就被解析为父元素自身或父元素的伪类。

less运算

引入less文件

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <link rel="stylesheet" href="运算.css" type="text/css">
    </head>
    <body>
        <div></div>
    </body>
</html>
```

生成一个less文件

less运算 就是@Size:10px + 5;

```
@Size:10px + 5;
@Color:red;

html{
    font-size: @Size;
}

div{
    width: 20rem;
    height: 20rem;
    background-color:@Color;
    &:hover{
        background-color:yellow;
    }
}
```

VUE

基本使用vue

创建以一个vue文件

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
```

```
    </head>
    <body>

        <div class="container">

            <!-- 使用插值表达式 -->

            {{username}} ------ {{age}}

        </div>

        <!-- 引入vue 文件夹 -->

        <script src = './vue.js'></script>

        <script>

            const VM = new Vue({
                el : ".container",
                data:{
                    username:'zs',
                    age : 23
                }
            }
                                );
        </script>
    </body>
</html>
```

指令

内容渲染指令

1. v-html

```
<div v-html="html"></div>
v-html 可以解析 标签指令
```

2. v-text

```
<div v-text = 'text'></div>
不能解析标签指令
```

3.插值表达式

```
插值表达式{{}}
最为常用 但是 也是不会去解析标签文件
```

v-model双向绑定

```
<!-- v-modle 双向绑定 -->
<input type="" v-model='usernames'>{{usernames}}
```

v-bind

```
发ajax axios 请求的时候 会 返回一些数据其中就会有 一些自定义属性
```

```
绑定固定属性
<img v-bind:src = "img" alt=""> v-blind 可以省略成:
```

v-on

```
注册事件
```

```
注册函数的时候 可以传递参数
<button  v-on:click="post('post')" >post</button>
```

```
只是单独调用函数    <button  v-on:click="get" >get</button>
```

```
默认参数    <button  v-on:click="event" >event</button>
```

```
v-on 可以简化成 @
```

事件修饰符	说明
.prevent	阻止默认行为（例如：阻止 a 连接的跳转、阻止表单的提交等）
.stop	阻止事件冒泡
.capture	以捕获模式触发当前的事件处理函数
.once	绑定的事件只触发1次
.self	只有在 event.target 是当前元素自身时触发事件处理函数

```
阻止默认行为 prevent
 <a href="#" v-on:click.prevent="post">123</a>
```

修饰符	作用	示例
.number	自动将用户的输入值转为数值类型	<input v-model.number="age" />
.trim	自动过滤用户输入的首尾空白字符	<input v-model.trim="msg" />
.lazy	在“change”时而非“input”时更新	<input v-model.lazy="msg" />

v-if

渲染指令
v-if
12313 sendmessage() 带有返回值 或者 使用 === 等判断符号去判断

v-for

渲染指令
v-for

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        <div class="container">
            <!-- 显示指令 -->
            <div v-if = "sendmessage()">12313</div>
        </div>
        <script src="./vue.js"></script>
        <script>
            const vm = new Vue({

                el:".container",

                data:{
                    usernames: 'zs',
                    age:'12',
                },

                methods: {
                    sendmessage(){
                        return false
                    }
                }
            })
        </script>
    </body>
</html>
```

对一个数组进行 渲染

过滤器

过滤器的使用位置 使用在 使用在插值表达式当中 或者 属性绑定 当中

{{message}} 表示 吧message 渲染到 页面当中去 {{message | function}} | 表示使用 过滤器 这个过滤器 是 一个 函数 调用function 函数 渲染到页面的值 也就是 函数的返回值

```
  <!-- 实现首字母 大写 -->
    <!-- 使用插值 的 方法 实现 过滤器的使用 -->
    <p>{{message | capitalize}}</p>

    <!-- 使用 属性绑定的 方法  -->

    <div :name = 'message|capitalize'></div>

    <!-- 以上两种方法都是将 message 传给 capitalize 函数 按照 过滤器的 规则 进行过滤 -->
```

```
filters:{
        capitalize(str){
          return str.charAt(0).toUpperCase() + str.slice(1)
        }
      }
```

局部过滤器

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        <div class="container">
            <p>{{message|capitalize}}</p>
        </div>
        <script src="./vue.js"></script>
        <script>

            const vm = new Vue({

                el:'.container',

                data:{
                    message:'hello Vue!'
                },

                // 在这里定义的就是 局部 过滤器
                // 在单个 创建的 vue 实例中定义 使用 过滤器 是 局部过滤器

                filters:{
                    capitalize(str){
                        return str.charAt(0).toUpperCase() + str.slice(1)
                    }
                }
```

```
            })

        </script>
    </body>
</html>
```

全局过滤器

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        <div class="container">

            <p>{{message | capitalize}}</p>

        </div>
        <script src = './vue.js'></script>
        <script>
            // 全局过滤器 定义的 使用 放在最上面
            // 给 vue 的 构造函数 绑定 filter 过滤器
            Vue.filter('capitalize',function(str){
                return str.charAt(0).toUpperCase() + str.slice(1)
            })

            const vm = new Vue({
                el:'.container',
                data:{
                    message:'hello vue!'
                }
            })

        </script>

    </body>
</html>
```

当 全局 过滤器 和 局部 过滤器 冲突了 实现 ‘就近原则’ 调用局部 过滤器

侦听器

侦听器的使用

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
        <title>Document</title>
    </head>
    <body>
        <div class="container">
            <input type="text" v-model="usename">
        </div>
        <script src="./vue.js"></script>
        <script>
            const vm = new Vue({

                el:'.container',

                data:{
                    usename:'zs'
                },

                watch:{
                    //  当 这个 属性 发生变化的时候 这个 函数 就会起作用
                    usename(newval,oldval){
                        console.log(newval,oldval);
                    }
                }
            })
        </script>
    </body>
</html>
```

注册 提交事件

引入ajax 请求

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div class="container">
    <input type="text" v-model="usename">
  </div>
  <script src="./vue.js"></script>
  <script src="./jquery.min.js"></script>
  <script>
   const vm = new Vue({

     el:'.container',

     data:{
      usename:'zs'
     },
     watch:{
```

```
          // 利用ajax发送请求
          // 1 如果内容为空 则 发送失败 直接 返回 出去
         usename(newval){
           if(newval === '') return

           $.ajax({

             type : "get",

             url:'../data_ajax.json',

             success:function(e){
              if(newval === e.account){
                console.log('success')
              }
              else {
                console.log('error')
              }
             },

             error:function(message){

               console.log('error' + message)
             }
           })

         }
     }
   })
  </script>
</body>
</html>
```

对象格式的 侦听器

```
handler 处理函数
immediate 是否立即执行
```

deep 作用

```
deep true 是 可以获取 对象的属性
```

计算属性

```
引入 模板字符串
```

```
同过 时时计算 来修改自己的样式
当 牵涉到的 相关内容变化之后 计算属性的那一部分 也会跟着变化
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
```

```
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Document</title>
        <style>
            .son{
                width: 100px;
                height: 100px;
                border: 1px solid red;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <button @click="show">click</button><br/>
            <input type="text" v-model="r"> <br/>
            <input type="text" v-model="g"> <br/>
            <input type="text" v-model="b">
            <div class="son" :style="{backgroundColor:rgb}"></div>

        </div>
        <script src="./vue.js"></script>
        <script>
            const vm = new Vue({
                el:'.container',
                data:{
                    r:100,
                    g:0,
                    b:0
                },
                computed:{
                    rgb(){
                        return `rgb(${this.r},${this.g},${this.b})`
                    }
                },
                methods:{
                    show(){
                        console.log(this.rgb)
                    }
                }

            })
        </script>
    </body>
</html>
```

```
:style="{backgroundColor:rgb}" 在添加属性的时候 要加上{}
rbg 是一个动态的属性 要使用 :style 进行绑定
```

组件

生成文件

生成一个vue 项目 cmd 进入 终端 输入 vue create name

mount

```
new Vue({
  render: h => h(App),
}).$mount('#app')

.$mount('#app') <==> el:'#app'
```

组件的三个组成部分

template ---- 标签文件

script --- js文件书写

style --- 样式书写

注册 组件的 步骤

1. 建立一个组件的文件
2. 导入组件 import Left from '@/components/Left.vue'
3. 在 根组件当中 使用组件

注册全局组件

单页面的全局组件

```
// 全局组件
  Vue.component("Left",{
    template:
    `<div>
          这个是Left组件 -- {{money}}
          <button @click = "add">add</button>
    </div>`,

    data(){
      return {
         money:10
      }
    },
    methods:{
      add(){
        this.money+=10
      }
    }
  }),
  Vue.component("Right",{
    template:
```

```
    `<div>
      这个是Right组件{{number}}
      <button @click = 'add'>add</button>
    </div>`,
    data(){
      return {
        number:10
      }
    },
    methods:{
      add(){
        this.number  = this.number + 1
      }
    }
  })
```

单页面的 局部组件

```
// 局部组件

  const vm = new Vue({

    el:".container",

    data:{
      message:10
    },

    methods: {
      add(){
        this.message += 10
      }
    },

    components:{
      hello:{
        template:`
        <div>
          这个是局部组件
          {{message}}
          <button @click = "add">add</button>
        </div>
        `,

        data(){
          return {
            message:10
          }
        },
        methods:{
          add(){
            this.message += 10
          }
        }
      }
```

```
      }

    })
```

```
在main.js中 通过 Vue.component()
参数1 注册名称
参数2 需要注册组件的 名称
```

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

import count from '@/components/count.vue'

new Vue({
  render: h => h(App),
}).$mount('#app')

Vue.component('Mycount',count);
```

```
在跟组件中进行 注册
```

props 属性

```
引入props属性 是 为了解决 全局组件中的 数据在不同组件 不同使用
避免数据 混乱使用
```

```
使用形式
自己定义一个 属性
props ["init"]
```

```
props 是 只读属性
// props 进行对象化 设置
props:{
    init:{

    }
}
```

```
    props:{
      init:{
        // 属性
        type:Number,

        default:0,

        required:true
      }
    }
```

样式冲突

为了 解决这个问题 消除 样式冲突问题 引入 scoped

```
<style >
  .Left{
    display: flex;
    flex: 1;
    height: 100px;
    background-color:red;
  }
  button{
    background-color:purple;
  }
</style>>
```

引入 scoped

```
<style scoped>
  .Left{
    display: flex;
    flex: 1;
    height: 100px;
    background-color:red;
  }
  button{
    background-color:purple;
  }
</style>>
```

deep

这里只做了解

如果给当前组件的style 节点添加了scoped 属性，则当前组件的样式对其子组件是不生效的。如果想让某些样式对子组件生效，可以使用/deep/ 深度选择器。

生命周期

组件创建阶段

创建阶段 只执行一次

beforecreate

beforeCreate
在 beforeCrrete 创建 之前 props data methods 就是 不能被使用
在 beforeCreate 创建了 props data method 还是不能使用

created

created 创建好的时候
组件中的 props data methods 就被创建好了
但是 模板还没有创建好 不能被 渲染 出来

在 created 组件中 经常发送ajax 请求

beforeMount

在这个生命周期中组件的页面还没有被渲染
也就说在这个地方还拿不到 dom 元素

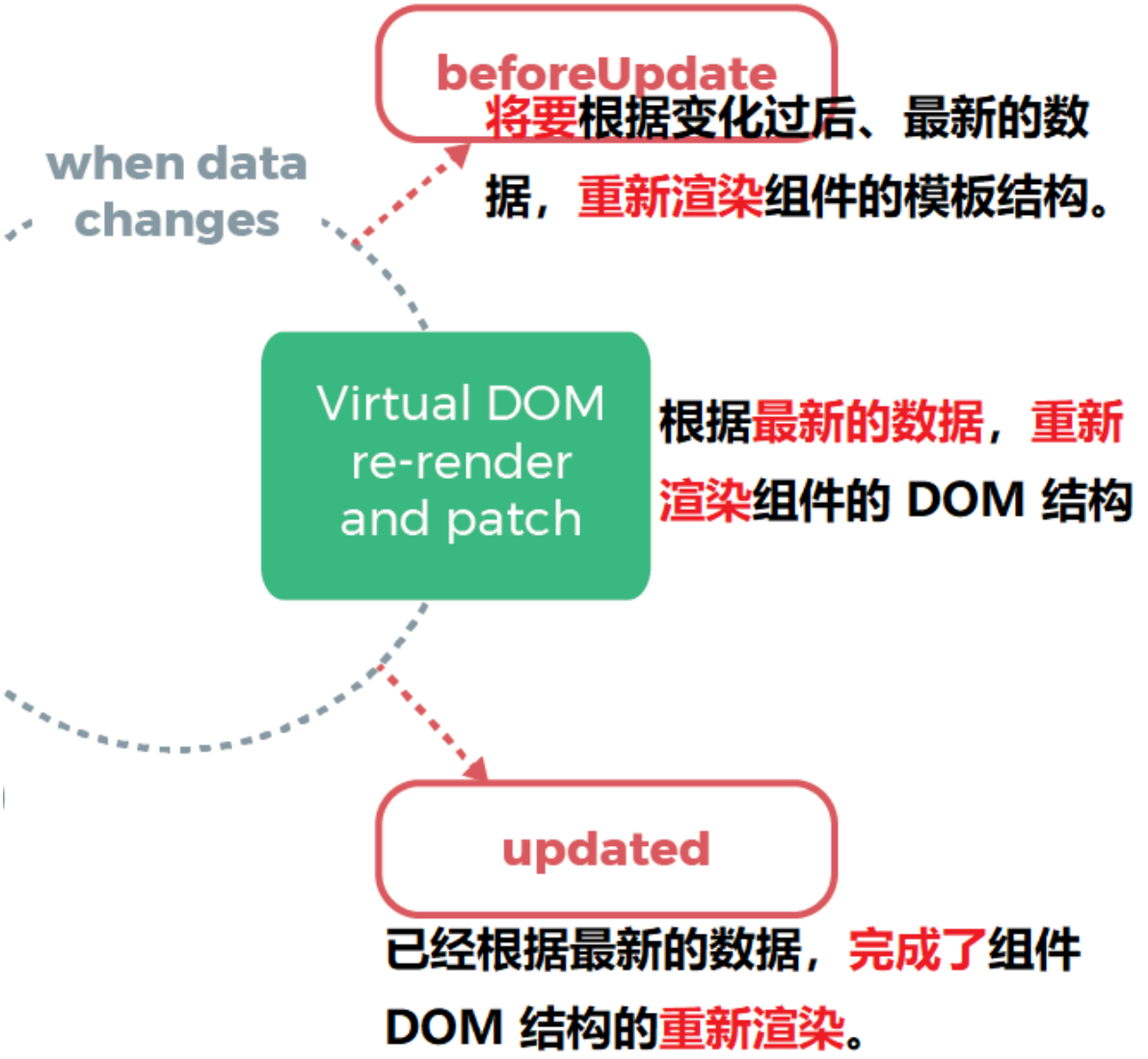
mounted

这个声明周期 是第一个 时期拿到 dom 操作元素

```
mounted(){
  const dom = document.querySelector(".Life")
  console.log(dom)
}
```

组件运行阶段

运行阶段 可以运行 0 - N



beforeupdate

这个阶段 数据 是最新的 因为 经过了 data changes

但是 数据 页面时没有 更新

updated

这个生命周期 的时候 是 已经创建好了 页面数据 也被更新了

组件传输数据

父传子

```vue
<template>
  <div class="component">
    <!-- 父传子 案例 -->
    <!-- 父传子 可以是 传送 简单数据类型 也可以传送 复杂数据类型
      <Son :init = "message" :info = 'info'></Son>
  </div>
</template>

<script>

// 注册组件
import Son from '@/components/Son.vue'

export default {
  // 注册组件
  components: {
      Son
  },
  data(){
    return {
      message:'组件',
      info:{
        name:'zs',
        age:'12'
      }
    }
```

```
导入 组件
注册 组件
引用 组件
```

```
父传子 是 通过 props 属性 来修改的
   <Son :init = "message" :info = 'info'></Son>
   传送数据
   <p> init 的 数据 是 {{init}}</p>
   <p> info 的 数据 是 {{info}}</p>
   进行页面渲染
```

```vue
<template>
  <div class="Son">
    <p> init 的 数据 是 {{init}}</p>
    <p> info 的 数据 是 {{info}}</p>
  </div>
</template>

<script>
export default {
  props:{
    init:{

    },
    info:{

    }
  }
}
</script>

<style >
  .Son{
    height: 100px;
    background-color: red;
  }
</style>
```

```vue
<template>
  <div class="component">
    <!-- 父传子 案例 -->
    <!-- 父传子 可以是 传送 简单数据类型 也可以传送 复杂数据类型 -->
      <Son :init = "message" :info = 'info'></Son>
  </div>
</template>

<script>

// 注册组件
import Son from '@/components/Son.vue'

export default {
```

```
  // 注册组件
  components: {
      Son
  },
  data(){
    return {
      message:'组件',
      info:{
        name:'zs',
        age:'12'
      }
    }
  }

}
</script>

<style lang="less" scoped>

</style>
```

```
<template>
  <div class="Son">
    <p> init 的 数据 是 {{init}}</p>
    <p> info 的 数据 是 {{info}}</p>
  </div>
</template>

<script>
export default {
  props:{
    init:{

    },
    info:{

    }
  }
}
</script>

<style >
  .Son{
    height: 100px;
    background-color:red;
  }

</style>
```

子传父

```
子传父数据 是通过 $emit
```

```
通过this.$emit()
子组件中的
    /**
     * 触发事件
     * 参数 1 触发事件的条件
     * 参数 2 传递的参数
     */
```

```
<template>
<div class="component">
    <!-- 父传子 案例 -->
    <!-- 父传子 可以是 传送 简单数据类型 也可以传送 复杂数据类型 -->
    <Son :init = "message" :info = 'info'></Son>
    <Son1 @getname = "requerData"></Son1> //getname 表示 子组件的 某个属性 是通过
$emit 产生的 类似于 click 也是 // $emit产生的  requerData 表示的 自己的函数
    {{ this.user.name }}
    </div>
</template>

<script>

    // 注册组件
    import Son from '@/components/Son.vue'
    // 注册组件
    import Son1 from '@/components/Son1.vue'

    export default {
        // 注册组件
        components: {
            Son,
            Son1
        },
        data(){
            return {
                message:'组件',
                info:{
                    name:'zs',
                    age:'12'
                },
                user:{
                    name:'zs',
                    age:'32'
                }
            }
        },
        methods: {
            requerData(val){
                this.user.name = val
            }
```

```
        }
    }
</script>

<style lang="less" scoped>

</style>
```

```
<template>
  <div class="Son1">
    <button @click="getdata">点击</button>
  </div>
</template>

<script>
export default {

 data(){
   return {
     name:'lisi',
     age:'23'
   }
 },

 methods:{

   getdata(){
    /**
     * 触发事件
     * 参数 1 触发事件的条件
     * 参数 2 传递的参数
     */
     this.$emit('getname',this.name) // 通过 $emit 产生一个 getname 属性 类似于 产生
click 效果 一致
   }
 }
}
</script>

<style>

</style>
```

兄弟之间的数据共享化

```
关键字 EventBus
```

步骤如下

1. 生成一个EventBus.js 文件

```
import Vue from 'vue'

export default new Vue()
```

2. 生成两个兄弟组件

分别将在两个组件中导入 自己建立的 js 文件

```
<template>
<div class="bor1">
    这个是 bor1组件
    <button @click="send">send data</button>
    </div>
</template>

<script>

    import bus from '../../EventBus'
    export default {
        data(){
            return{
                message:'send data'
            }
        },
        methods:{
            send(){
                bus.$emit('senddata',this.message)
            }
        }
    }
</script>

<style>
    .bor1{
        height: 100px;
        background-color:yellow;
    }
</style>
```

```
<template>
<div class="bor2">
    这个是 bor2组件
    {{acceptdata}}
    </div>
</template>

<script>

    import bus from '../../EventBus'
```

```
    export default {
        data(){
            return {
                message: 'accept data',
                acceptdata:''
            }
        },
        // 创建周期
        created(){
            bus.$on('senddata',(val)=>{
                this.acceptdata = val
            })
        }
    }
</script>

<style>
    .bor2{
        height: 100px;
        background-color:orange;
    }
</style>
```

3. 发送数据一方使用 bus.$emit('senddata',this.message) 进行发送数据
4. 接收数据 一方使用

```
bus.$on('senddata',(val)=>{
    this.acceptdata = val
   })
```

ref

ref 属性使用

$ref 是 为了避免操作dom 而诞生的

ref基本使用

给指定标签加上 ref = '姓名'

调用 this.$refs.姓名

```
<template>
  <div class="ref">
    <h1 ref="MYref">ref组件</h1>
    <button @click="show">show</button>
    <hr>
```

```
  </div>
</template>

<script>
export default {
  methods:{
    // 使用ref 属性 进行设置
    //  ref = 'MYref'
    // 调用 this.$refs.MYref 进行 设置
    show(){
      this.$refs.MYref.style.backgroundColor = 'red'
    }
  }
}
</script>

<style lang="less" scoped>

</style>
```

```
ref 有三种用法：
    1、ref 加在普通的元素上，用this.$refs.（ref值） 获取到的是dom元素

    2、ref 加在子组件上，用this.$refs.（ref值） 获取到的是组件实例，可以使用组件的所有方法。在使用方法的时候直接this.$refs.（ref值）.方法（） 就可以使用了。

    3、如何利用 v-for 和 ref 获取一组数组或者dom 节点
```

1. 加在普通元素上

```
<h1 ref="MYref">ref组件</h1>
this.$refs.MYref.style.backgroundColor = 'red'
```

2. 加在组件上 可以调用 组件中的数据 和 方法

```
 <Left ref="left"></Left>
  this.$refs.left.add()
```

3. 利用v-for 获取ref 获得一组数组

```

```

this.$nextTick(cb)

```
组件的this.$nextTick(cb) 方法 会将这个执行函数 放在 下一个 dom 更新之后去执行
这个 是 涉及到 生命周期 中的 beforeupdate 数据发生变化 之后 数据变成了 最新的 但是页面 没有更新出来 ===》 导致 页面 没有被渲染 最终导致 操作不了dom 元素
```

动态组件

component

component 是 vue 组件中内定的 一个标签是 可以实现 标签转换

第一步 声明一个变量

```
data(){
   return{
     comname:'Left'
   }
  },
```

第二步 给出一个标签

```
<component :is = "comname"></component>
```

第三步 实现标签转换

```
  <button @click="comname  = 'Left'">Left</button>
  <button @click="comname  = 'Right'">Right</button>
```

keep-alive

在动态组件当中 进行切换的时候
当 切换到 另外一个组件当中去的 时候 当前这个组件会被销毁
当 切换回来 这个组件 会重新被 生成
导致 每次 数据 都会被 更新 根据 生命周期 可知

```
  created(){
    console.log('组件被创建了')
  },
  destroyed(){
    console.log('组件被销毁了')
  }
```

验证组件被创建和销毁

为了避免 出现这个 使用 keep-alive

```
  <keep-alive>
      <component :is = "comname"></component>
    </keep-alive>
```

加上了 keep-alive 之后 就会是的或者属性保持活性

与之相对应的 出现了
激活 生命周期 activated
缓存 生命周期 deactivated

include

```
include 属性 是 指定 对 某个组件进行 包含 指定的那个 组件
    <keep-alive include="Left">
      <component :is = "comname"></component>
    </keep-alive>
```

exclude

```
指定那个组件不用进行 缓存
 <keep-alive exclude="Right">
       <component :is = "comname"></component>
    </keep-alive>
```

插槽

```
插槽的作用: 在跟组件 中 调用 子组件的同时 给子组件 传值
```

slot 基本使用

```
在子组件当中 使用一个 slot 标签 在 父组件当中 直接使用
     <Left>
       <p>使用插槽</p>
     </Left>

     <template>
  <div class="Left">
    <h1>Left</h1>
    <hr>
    <button @click="add">add</button>
    message -- {{message}}
    <hr>
    <slot></slot>
  </div>
</template>
```

v-solt

```
利用 v-slot 设定 使用标签
结合 template 一起 使用
```

子组件进行定义

```
<template>
  <div class="Left">
    <h1>Left</h1>
    <hr>

    <slot name="ty"></slot>
    <hr>
    <slot name="pecoh"></slot>
  </div>
</template>
```

父组件进行使用

```
    <Left>
       <template v-slot:ty>
             <p>指定使用 ty 插槽</p>
         </template>

          <template v-slot:pecoh>
             <p>指定使用 pecoh 插槽</p>
         </template>
    </Left>
```

```
v-slot 的简写形式 是 #
```

作用域插槽

```
  父组件中定义 obj 属性 接受数据
  <template #ty = "obj" >

    <p>指定使用 ty 插槽</p>

    {{obj.msg}}

  </template>
```

```
子组件中 写数据
<slot name="ty" msg="hello vue.js"></slot>
```

自定义指令

私有自定义指令

bind

第一个参数 e

```
bind 属性 是只会调用 一次 当页面被更新了 或者是 数据发生了变化 还是 不能 进行再次触发
自定义属性 给自定义属性 进行修改样式
```

```
自定义指令的 关键字 是
directives
```

```
进行绑定
<h1 v-color>App</h1>
```

```
进行注册
directives:{
    color:{
     bind(e){
       e.style.color = 'red'
     }
    }
  }
```

第二个参数 binding

```
binding 是 可以进行 传输变量
1 在 data 数据中 声明 一个变量 在 变量中 进行修改样式
2 直接 利用 v-color = “‘颜色’”
```

```
 在h1 进行注册 Color 属性

 Color 是 data 中的 数据

 <h1 v-color = "Color">App</h1>

 利用binding 进行接收

  bind(e,binding){
       e.style.color = binding.value
  }
```

update

```
     update(e,binding){
       e.style.color = binding.value
     }
    }
```

```
当数据 被触发 了 就会发生更新
```

```
简化形式
当bind 和 updata 中的 数据 一致
可以将自定义指令 定义成函数的 形式
```

```
    color(e,bind){
      e.style.color = bind.value
    }
```

全局自定义指令

```

```

Echarts

初始化 echarts 对象

```
1 创建一个 大盒子 进行包装 盒子
```

```
2 let myChart = echarts.init(document.querySelector(".container"))
创建 一个图表 初始化
调用 echarts.init() 进行初始化
```

```
3 提供数据
 var option = {
            title: {
                text: 'ECharts 入门示例'
            },
            tooltip: {},
            legend: {
                data:['销量']
            },
            xAxis: {
                data: ["衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子"]
            },
            yAxis: {},
            series: [{
                name: '销量',
                type: 'bar',
                data: [5, 20, 36, 10, 10, 20]
            }]
        };
```

```
4 数据显示 出来
 调用 setOption 函数
 myChart.setOption(option)
```

数据可视化 代码实现

```
<!--
 * @Descripttion: 数据可视化实现
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-22 21:43:22
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-22 22:00:55
-->
<template>
  <div class="echarts">
    <div class="box"></div>
  </div>
</template>

<script>
export default {
 data () {
   return {

   }
 },
 mounted(){
   this.init()
 },
 methods:{
   init(){

     let dom = document.querySelector('.box')

     let myecharts = this.$echarts.init(dom)

     let option = {

          xAxis: {
            type: 'category',
            data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
          },
          yAxis: {
            type: 'value'
          },
          series: [
            {
              data: [150, 230, 224, 218, 135, 147, 260],
              type: 'line'
            }
          ]
        };

     myecharts.setOption(option)
   }
 }
}
</script>
```

```
<style>
  .box {
    width: 100px;
    height: 100px;
  }
</style>
```

数据发送请求

1 ajax 发送模式

```
    let Ajax = ()=> {

      $.ajax({
        type:'get',
        url:'../json/data_ajax.json',
        success: function(data){
          console.log("请求成功");
          console.log(data)
        },
        error: function(data){
          console.log("请求失败")
          console.log(data)
        }

      })
    }

    Ajax()
```

2 ajax 进一步封装

```
let Ajax = (type, url)=> {

      $.ajax({
        type, // 执行函数的类型
        url, // 执行的 函数的地址
        success: function(data){  // 成功的时候 调用这个函数
          console.log("请求成功");
          console.log(data)
        },
        error: function(data){    // 失败的时候 调用这个函数
          console.log("请求失败")
          console.log(data)
        }
```

```
    })
}

// 将 ajax 进行封装 传入 调用的形式和即将执行的地址

Ajax("get","../json/data_ajax.json")   // 调用函数
```

3 axios 发送数据

```
get() {
    const data =  axios({
        methods:"get",
        url:"../json/data_axios.json"
    }).then(function(data){
        console.log('succes')
        console.log(data)
    })
    }
```

4 axios 结构赋值

```
async get(){
    const {data : res} = await axios({
        methods:"get",
        url:"../json/data_axios.json"
    })

    console.log(res)
}
},
```

5 封装 axios

```
<script>
    const vm = new Vue({

        el:'.App',

        data:{
            list:[]
        },

        methods:{

            get(methods,url) {

                axios({
                    methods,
                    url
```

```
                }).then((data)=>{
                    console.log(data)
                    this.list = data.data.list
                })
            }
        },
        created(){
            this.get("get","data.json")
        }
    })
</script>
```

6 进一步分装 axios

```
<script>
    const vm = new Vue({
        el:'.App',
        data:{
            list:[]
        },
        methods:{
            async get(methods,url) {
                const {data:res} = await axios({
                    methods,
                    url
                })
                this.list = res.list
            }
        },
        created(){
            this.get("get","data.json")
        }
    })
</script>
```

```
axios 通常 是和 created 进行 搭配 自我调用 自动执行
```

```
created(){
    this.get()
}
```

最后的倔强

节点操作

创建节点

```
// 创建一个节点

let li1 = document.createElement("li")

li1.innerHTML = '这个是第一个新建的节点'
```

插入节点 appendChild

```
// 插入到 ul 最后 中

ul.appendChild(li1)
```

插入到某个位置

```
ul.insertBefore(new,old)
ul.insertBefore(li2,li1)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <ul>
    <li>1</li>
  </ul>

  <script>

    let ul = document.querySelector("ul")

    let lis = document.querySelectorAll("ul li")

    console.log(lis)

    // 创建一个节点

    let li1 = document.createElement("li")

    li1.innerHTML = '这个是第一个新建的节点'

    let li2 = document.createElement("li")

    li2.innerHTML = '这个是第二个新建的节点'

    // 插入到 ul 中
```

```
    ul.appendChild(li1)
    ul.appendChild(li2)

    ul.insertBefore(li2,li1)

  </script>
</body>
</html>
```

阻止冒泡

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .father{
      width: 300px;
      height: 300px;
      background-color:red;
    }

    .son{
      width: 200px;
      height: 200px;
      background-color:orange;
    }
  </style>
</head>
<body>
  <div class="father">
    <div class="son"></div>
  </div>

  <script>

    let son = document.querySelector(".son")

    let father = document.querySelector(".father")

    son.addEventListener("click",function(event){
      console.log(1)
      event.stopPropagation();
    })

    father.addEventListener("click",function(){
      console.log(2)
    })
```

```
  </script>
</body>
</html>
```

委托事件

链式编程

jQuery的增加类

过渡

动画

flex 布局

过滤器的使用

侦听器的调用

计算属性

组件传输数据

父传子

```
1 创建一个组件
2 导入组件到跟组件中
3 在根组件中创建出 需要传送给子组件的数据
4 在子组件中创建出 需要接受的数据
5 进行传输数据
6 传送信息 是通过 数据变量 也就是 通过冒号 :init
```

子传父

```
1 创建一个组件
2 导入组件到跟组件中
3 在子组件中创建出 需要传送给根组件的数据
4 在根组件中创建出 需要接受的数据
5 进行传输数据
6 传送信息 是通过 数据变量 也就是 通过冒号 @函数名
```

兄弟之间传输数据

```
1 创建 一个在中间站 bus
2 在兄弟组件中导入 bus 组件
3 在发送数据中 使用 bus.$emit("send_data",this.message) 语句 进行发送数据
4 在接送数据中 使用  bus.$on("send_data",(val)=>{ this.getData = val;}) 接收数据 在 created 中 进行 接受
```

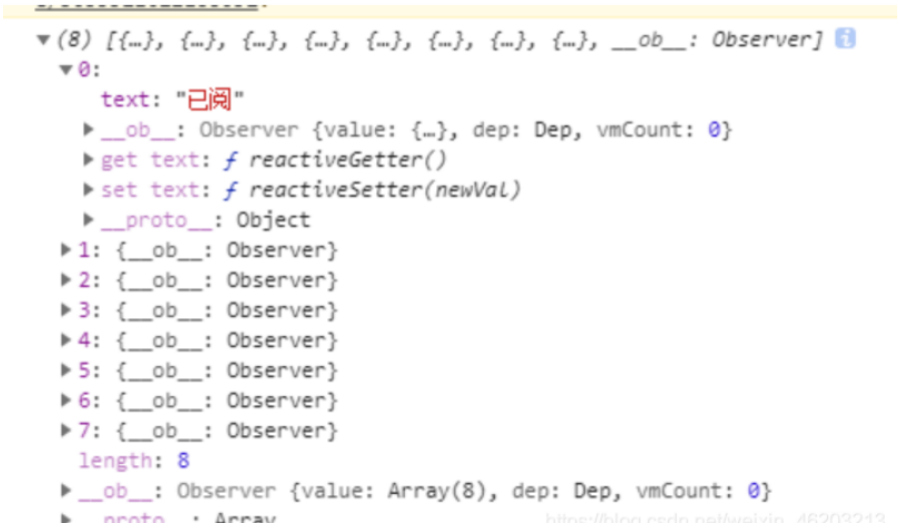
vue项目中常见问题

[vue使用Echarts渲染饼图 请求的后台数据返回{ob: Observer}

二向箔_唯一操作者

于 2020-10-20 21:53:37 发布

940
收藏 5
文章标签： vue json echarts vue.js javascript
版权
因为项目需求要做一个统计的功能，我以前从来没有使用过Echarts统计图属于第一次接触，其实看了官方文档之后觉得并不难，文档说的很清楚，这里不再啰嗦，直接说我在使用过程中遇到的问题：在我将后台请求回来的数据和图表结合的时候（我这里使用的是饼图）发现统计图并没有被渲染出来，非常疑惑感觉要翻车越是被容易轻视的东西越容易翻车，果不其然我做到了。我直接让后台小哥哥返回一个数组里面包含诸多对象，对象的键直接都是name和value是官方文档要求的格式，然鹅并不行，这肯定就是我前端的问题了，我打印出来的数据是{ob: Observer} 这种格式的，最开始还以为是这个原因，瞎弄半天也无果，网上搜说要用 JSON.parse(JSON.stringify(data)) 将返回的数据data先转换为JSON字符串形式，然后再从字符串形式转换成JSON格式。看网上说{ob: Observer} 这种格式的是 Vue 对数据监控添加的属性。
瞎弄了半天其实根本原因就不再这里，解决方法其实很简单：因为异步的原因统计图在渲染的时候后台还没有请求到数据，所以就是空白的鸭，统计图的渲染不要放在生命周期moutend中，而是在后台数据请求完成之后再渲染，怎么样，就一句话不需要做任何改动。
实际开发中每个人遇到的问题都不一样，希望能帮到大家。
希望这篇文章可以启发到你：https://segmentfault.com/q/1010000014779865?utm_source=tag-newest




```
// 获取统计报表数据
tongjiData(){
  orderStatistics(this.$store.getters.getSid).then(res=>{
    console.log(res)
    if(res.status==200){
      this.orderList = res.data;
      this.orderList.forEach((item=>{
            this.Name.push(item.name);
      }))
    // this.orderList=JSON.parse(JSON.stringify(datalist))
    this.drawChart();//数据请求成功之后初始化图表
    }
```


大致错误信息为： methods 希望是一个对象，但是得到了一个函数

错误代码演示：

```
methods(){
      test(){
      console.log(this.student);
    }
  }
```

修正代码：

```
methods:{
    test(){
      console.log(this.student);
    }
  }
```

vue 中 挂载相关事件 合 $ 进行解释

```
// 解释 为什么加上 $
// $ 很重要吗 ==> 这里没有什么魔法。$ 是在Vue 所有实例中都可用的属性 的一个简单约定。这样做会避免与已被定义的数据、方法、计算属性产生冲突。

//  Vue.prototype 进行挂载 当下次使用 echarts 直接调用 this..$echarts 就行 this === Vue.prototype 挂在对象 $echarts 替代 echarts

Vue.prototype.$echarts = echarts
```

页面检索 搜索部分 检索 时时 动态显示

使用 computed 计算属性实现动态 更新

```
<!--
 * @Descripttion: 使用computed 计算属性
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-22 23:29:24
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-24 17:20:21
-->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>输入搜索联想</title>
    <style type="text/css">
      #app {
        width: 400px;
        height: 400px;
        margin: 50px auto;
      }
      table {
        border-collapse: collapse;
        border: 1px solid black;
        margin-top: 20px;
      }

      thead tr {
        background: #4d83d6;
        color: #fff;
      }
      tr td {
        width: 80px;
        line-height: 30px;
        text-align: center;
      }
      tbody tr:nth-child(2n) {
        background: #efefef;
      }
    </style>
    <script type="text/javascript" src="./js/vue.js"></script>
```

```
  </head>

  <body>
    <div id="app">
      <!-- <button @click="search">search</button> -->
      <!-- TODO: 请在下面实现需求 -->
      <span>搜索名字: </span>
      <input placeholder="输入要搜索的名字" v-model="searchQuery"/>
      <table>
        <thead>
          <tr>
            <td v-for="col in columns">{{col}}</td>
          </tr>
        </thead>
        <tbody>
          <tr v-for="entry in search">
            <td v-for="col in columns">{{entry[col]}}</td>
          </tr>
        </tbody>
      </table>
      <!-- {{this.data}} -->
    </div>
  </body>
</html>
<script>
  // TODO: 请在下面实现需求
  new Vue({
    el: "#app",
    // 注意: 请勿修改 data 选项中的数据!!!
    data: {
      searchQuery: "",
      columns: ["name", "gender", "age"],
      newdata:[
      {
          name: "rick",
          gender: "male",
          age: 21,
        },
        {
          name: "demen",
          gender: "male",
          age: 26,
        },
        {
          name: "Jack",
          gender: "male",
          age: 26,
        },
        {
          name: "John",
          gender: "female",
          age: 20,
        },
        {
          name: "Lucy",
```

```
          gender: "female",
          age: 16,
        },
      ],
      data: [
        {
          name: "rick",
          gender: "male",
          age: 21,
        },
        {
          name: "demen",
          gender: "male",
          age: 26,
        },
        {
          name: "Jack",
          gender: "male",
          age: 26,
        },
        {
          name: "John",
          gender: "female",
          age: 20,
        },
        {
          name: "Lucy",
          gender: "female",
          age: 16,
        },
      ],
    },
    methods:{
      get(){
        let dom = document.querySelectorAll('thead tr td')
        dom[0].innerHTML = 'Name'
        dom[1].innerHTML = 'Gender'
        dom[2].innerHTML = 'Age'
      },
    },
    mounted(){
      this.get()
    },
    computed:{

      search(){

        console.log(1)

        this.newdata =  this.data.filter((el,index) => {

          if(el.name.toUpperCase().indexOf(this.searchQuery.toUpperCase()) !==
-1)

          return el
```

```
        })

        return this.newdata
      }
    },
  });
</script>
```

filters 实际应用

需求是 将 名称首字母 大写 但是禁止修改data 数据

做法 1 修改 html 内容 ==> innerHTML
做法 2 就是通过 过滤器的 使用 在内容显示的时候 将数据 进行 指定函数变化

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-22 23:29:24
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-24 20:22:18
-->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>输入搜索联想</title>
    <style type="text/css">
      #app {
        width: 400px;
        height: 400px;
        margin: 50px auto;
      }
      table {
        border-collapse: collapse;
        border: 1px solid black;
        margin-top: 20px;
      }

      thead tr {
        background: #4d83d6;
        color: #fff;
      }
      tr td {
        width: 80px;
        line-height: 30px;
        text-align: center;
      }
```

```
      tbody tr:nth-child(2n) {
        background: #efefef;
      }
    </style>
    <script type="text/javascript" src="./js/vue.js"></script>
  </head>

  <body>
    <div id="app">
      <!-- TODO: 请在下面实现需求 -->
      <span>搜索名字: </span>
      <input placeholder="输入要搜索的名字" v-model="searchQuery"/>
      <table>
        <thead>
          <tr>
            <td v-for="col in columns">{{col | UP}}</td>
          </tr>
        </thead>
        <tbody>
          <tr v-for="entry in search">
            <td v-for="col in columns">{{entry[col]}}</td>
          </tr>
        </tbody>
      </table>
      <!-- {{this.data}} -->
    </div>
  </body>
</html>
<script>
  // TODO: 请在下面实现需求
  new Vue({
    el: "#app",
    // 注意: 请勿修改 data 选项中的数据!!!
    data: {
      searchQuery: "",
      columns: ["name", "gender", "age"],
      newdata:[
      {
          name: "rick",
          gender: "male",
          age: 21,
        },
        {
          name: "demen",
          gender: "male",
          age: 26,
        },
        {
          name: "Jack",
          gender: "male",
          age: 26,
        },
        {
          name: "John",
          gender: "female",
```

```
        age: 20,
      },
      {
        name: "Lucy",
        gender: "female",
        age: 16,
      },
    ],
    data: [
      {
        name: "rick",
        gender: "male",
        age: 21,
      },
      {
        name: "demen",
        gender: "male",
        age: 26,
      },
      {
        name: "Jack",
        gender: "male",
        age: 26,
      },
      {
        name: "John",
        gender: "female",
        age: 20,
      },
      {
        name: "Lucy",
        gender: "female",
        age: 16,
      },
    ],
  },

  filters:{
   /**
   * @name: UP
   * @msg: 使用 过滤器 将首字符 转换成大写
   * @return {*}
   */
    UP(str){
      return str[0].toUpperCase() + str.slice(1)
    }
  },
  methods:{
    /**
     * @name: get
     * @msg: 将显示的字符转你换成首字母大写
     * @return {*}
     */
    // get(){
    //   let dom = document.querySelectorAll('thead tr td')
```

```
      //   dom[0].innerHTML = 'Name'
      //   dom[1].innerHTML = 'Gender'
      //   dom[2].innerHTML = 'Age'
      // },
    },
    mounted(){
      // this.get()
    },
    computed:{

      search(){

        console.log(1)

        this.newdata =  this.data.filter((el,index) => {

          if(el.name.toUpperCase().indexOf(this.searchQuery.toUpperCase()) !==
-1)

          return el

        })

        return this.newdata
      }
    },
  });
</script>
```

复选框问题

选择复选框 问题 checked 属性 判断这个 复选框是否被选择 可以通过 判定复选框时候被选中来判断

```
male.addEventListener('click',function(){

    sex = '男'

    console.log(male.checked,female.checked)

})

female.addEventListener('click',function(){

    sex = '女'

    console.log(male.checked,female.checked)

})
```

2D转换和动画 以及3D转换 强化篇

2D移动属性

```
transform: translate() | translateX() | translateY()
```

```
<!--
 * @Descripttion: 制作动画效果
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-26 20:00:01
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-26 20:27:43
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .container {
      width: 100px;
      height: 100px;
      background-color: red;
    }

    .container:hover {
      /*
      移动 属性
       transform : 属性1 translate(x,y) 属性2 translateX(100px) translateY(100px)
       */
       /* 属性1 */
      /* transform: translate(100px,100px); */
      /* 属性2 */
      /* transform: translateX(100px); */
      /* 属性3 */
      transform: translateY(100px);
    }
  </style>
</head>
<body>
  <div class="container"></div>
  <button class="btn">点击</button>
</body>
</html>
```

旋转效果 和 变换转换中心

```
<!--
 * @Descripttion: my project
 * @version: 1.0
```

```
 * @Author: Typecoh
 * @Date: 2022-05-26 20:30:10
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-26 20:34:31
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>

    .container {

      width: 100px;
      height: 100px;
      background-color: aqua;

    }

    .container:hover{

      /* 2D旋转 */
      /* 指定旋转度数 */

      transform: rotate(45deg);

      /* 是指转换的 旋转中心点 */
      /* 设置旋转中心是 左上边 */
      transform-origin: left top;

    }
  </style>
</head>
<body>
  <div class="container"></div>
</body>
</html>
```

缩放效果

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-26 20:34:55
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-26 20:36:02
-->
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .container {
      width: 100px;
      height: 100px;
      background-color: red;
    }

    .container:hover{
      /* scale 缩放效果 */
      transform: scale(2);
    }
  </style>
</head>
<body>
  <div class="container"></div>
</body>
</html>
```

动画效果

5. 动画

5.2 动画常用属性

属性	描述
@keyframes	规定动画。
animation	所有动画属性的简写属性，除了animation-play-state属性。
animation-name	规定@keyframes动画的名称。（必须的）
animation-duration	规定动画完成一个周期所花费的秒或毫秒，默认是0。（必须的）
animation-timing-function	规定动画的速度曲线，默认是“ease”。
animation-delay	规定动画何时开始，默认是0。
animation-iteration-count	规定动画被播放的次数，默认是1，还有infinite
animation-direction	规定动画是否在下一周期逆向播放，默认是“normal“,alternate逆播放
animation-play-state	规定动画是否正在运行或暂停。默认是"running",还有"paused"。
animation-fill-mode	规定动画结束后状态，保持forwards回到起始backwards

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-26 20:39:09
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-26 23:04:09
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>

  /* 制作动画的 过程  */
  /* 1 定义动画 */
  /* 2 调用动画 */

  .container {
    width: 100px;
    height: 100px;
    background-color: red;

    /* 调用动画 */
    animation-name: move;
    /* 定义动画时间 */
    animation-duration: 3s;

    /* 曲线的类型 */

    animation-timing-function:linear;

    /* 设置 什么时候开始 */

    /* animation-delay: 1s; */

    /* 设置播放次数 */

    animation-iteration-count: infinite;

    /* 设置运行状态 */

    /* animation-play-state: paused; */

    /* 完成后的状态 */

    animation-fill-mode: backwards;

    /* 设置返回方式 */

    animation-direction: alternate;

  }

  /* 定义动画 */
  @keyframes move {

    0%{
      width: 100px;
      height: 100px;
    }

    100%{
      width: 300px;
```

```
        height: 300px;
      }
    }

  </style>
</head>
<body>
  <div class="container"></div>
</body>
</html>
```

3D转化效果

透视 加上透视之后太能显示3D效果
perspective 给需要显示 3D效果的盒子的 父亲添加
近大远小

3D移动效果

移动效果 transform : translate3d | translateX | translateY | translateZ

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-26 23:05:45
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-26 23:24:14
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>

    body{
      /* 近大远小 */
      perspective:500px;
    }

    /* 加上透视的时候注意将 透视 写在 父盒子上面 */
    .container {
      margin:300px 300px;
      width: 100px;
      height: 100px;
      background-color: red;
      perspective: 10;
    }
```

```
    .container:hover{
     transform: translate3d(100px,100px,100px);
    }
  </style>
</head>
<body>
  <div class="container"></div>
</body>
</html>
```

3D旋转的视角

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-26 23:38:06
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-27 14:38:00
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>

    .container {
      width: 100px;
      height: 100px;
      background-color: red;
    }

    .container:hover{
      transition:all 1s;
      /* 以X轴 中心 旋转 X 轴 */
      /* transform: rotateX(90deg); */
      /* 以Y轴 中心 旋转 Y 轴 */
      /* transform: rotateY(90deg); */
      /* 以Z轴 中心 旋转 Z 轴 */
      transform: rotateZ(90deg);
    }
  </style>
</head>
<body>
  <div class="container"></div>
</body>
</html>
```

翻转盒子的小案例

```
解释要下 对背面的盒子进行隐藏
backface-visibility: hidden;
```

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-27 14:45:39
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-27 20:34:02
-->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
      /* 由于盒子的位置关系 存在一个自定义的 上下关系 */
      .box {
          position: relative;
          margin: 100px auto;
          width: 300px;
          height: 300px;
          font-size: 35px;
          color: #fff;
          line-height: 300px;
          text-align: center;
          transition: all .5s linear;
          transform-style: preserve-3d;
      }

      .front,
      .back {

          position: absolute;
          top: 0;
          left: 0;
          width: 100%;
          height: 100%;
          border-radius: 50%;
      }

      .front {
          background-color: pink;
          transition: all .4s;
          backface-visibility: hidden;
      }

      .back {
```

```
            background-color: purple;
            transform: rotateY(180deg);
            /* backface-visibility: hidden; */

        }

        .box:hover {
            transform: rotateY(180deg);
        }
    </style>
</head>

<body>
    <div class="box">
        <div class="back">万事如意！ </div>
        <div class="front">新年快乐！ </div>
    </div>
</body>

</html>
```

导航栏侧翻

注意 X,Y,Z 轴指向 和 正负顺序

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-27 20:58:59
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-27 21:10:23
-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <style>

      body{
        perspective:400px;
      }

      div{
        margin: 100px auto;
        width: 300px;
```

```
      height: 100px;
      position: relative;
      /* transform: persever-3d; */
      transform-style: preserve-3d;
      transition: all .6s;
    }

    div:hover{
      transform: rotateX(90deg);
    }

    li{
      list-style: none;
    }

    .font,
    .back{
      text-align: center;
      line-height: 100px;
      position: absolute;
      width: 100%;
      height: 100%;
    }

    .font{
      z-index: 1;
      background-color:red;
      transform: translateZ(50px);
    }

    .back{
      background-color: purple;
      /* 当同时 出现 移动坐标轴 和 旋转的 时候 移动一定要放在第一位 */
      transform: translateY(50px) rotateX(-90deg);
    }
  </style>

  <div>
    <li class="font">hello</li>
    <li class="back">world</li>
  </div>
</body>
</html>
```

旋转木马案例

```
<!--
 * @Descripttion: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-27 21:13:43
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-27 23:12:24
-->
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
    <style>

      body{
        perspective:400px;
      }
      @keyframes move {
        0%{

        }

        100%{
          transform: rotateY(360deg);
        }
      }
      .box{
        margin: 100px auto;
        width: 100px;
        height: 100px;
        position: relative;
        transform-style: preserve-3d;
        transition: all 10s;
        animation: move 10s linear infinite;
      }

      .font,
      .right,
      .left,
      .back,
      .in{
        position: absolute;
        text-align: center;
        line-height:100px;
        width: 100%;
        height: 100%;
      }

      .in{
        background-color:blue;
      }

      .right{
        background-color: red;
        transform:  translateX(100px) rotateY(90deg);
      }

      .font{
        background-color: yellow;
        transform: translateZ(100px);
```

```
      }

      .left{
        background-color: orange;
        transform:  translateX(-100px) rotateY(-90deg);
      }

      .back{
        background-color: purple;
        transform: translateZ(-100px) rotateY(-180deg);
      }
    </style>
<body>

  <div class="box">
      <div class="font">font</div>
      <div class="right">right</div>
      <div class="left">left</div>
      <div class="back">back</div>
      <div class="in">in</div>
  </div>

</body>
</html>
```

补充知识点

hover

```
hover 触发时 语法格式 : hover
格式 1 div :hover  hover 后面 不跟任何标签 效果是作用在 div
格式 2 div:hover a hover 后面 跟上标签 是作用在 后面那个标签上
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>展开你的扇子</title>
    <link rel="stylesheet" href="./css/style.css" />
  </head>
  <body>
    <div id="box">
      <!--元素 1-->
      <div id="item1">1</div>
      <!--元素 2-->
      <div id="item2">2</div>
      <!--元素 3-->
      <div id="item3">3</div>
      <!--元素 4-->
```

```
      <div id="item4">4</div>
      <!--元素 5-->
      <div id="item5">5</div>
      <!--元素 6-->
      <div id="item6">6</div>
      <!--元素 7-->
      <div id="item7">7</div>
      <!--元素 8-->
      <div id="item8">8</div>
      <!--元素 9-->
      <div id="item9">9</div>
      <!--元素 10-->
      <div id="item10">10</div>
      <!--元素 11-->
      <div id="item11">11</div>
      <!--元素 12-->
      <div id="item12">12</div>
    </div>
  </body>
</html>
```

```
  #box {
  width: 300px;
  height: 440px;
  margin: 100px auto;
  position: relative;
}

#item1,
#item12 {
  background-color: #90e0ef;
}
#item2,
#item8 {
  background-color: #8bdb81;
}
#item3,
#item10 {
  background-color: yellowgreen;
}
#item4,
#item6 {
  background-color: skyblue;
}
#item5,
#item9 {
  background-color: #d9d7f1;
}
#item7,
#item11 {
  background-color: #fed1f1;
}
#box div {
  width: 100%;
```

```
  height: 400px;
  transition: all 1.5s;
  position: absolute;
  left: 0;
  top: 0;
  /*旋转时，以盒子底部的中心为坐标原点*/
  transform-origin: center bottom;
  box-shadow: 0 0 3px 0 #666;
}
/*TODO: 请补充 CSS 代码*/

#box:hover #item1{
  transform:rotate(-60deg);
}
#box:hover #item2{
  transform:rotate(-50deg);
}
#box:hover #item3{
  transform:rotate(-40deg);
}
#box:hover #item4{
  transform:rotate(-30deg);
}
#box:hover #item5{
  transform:rotate(-20deg);
}
#box:hover #item6{
  transform:rotate(-10deg);
}
#box:hover #item7{
  transform:rotate(10deg);
}
#box:hover #item8{
  transform:rotate(20deg);
}
#box:hover #item9{
  transform:rotate(30deg);
}
#box:hover #item10{
  transform:rotate(40deg);
}
#box:hover #item11{
  transform:rotate(50deg);
}
#box:hover #item12{
  transform:rotate(60deg);
}
```

ajax 作为函数 返回值

```
<!--
 * @Descripttion: 返回值
```

```
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2022-05-29 21:51:41
 * @LastEditors: Typecoh
 * @LastEditTime: 2022-05-29 21:56:04
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>

    let ans = 1

    let Ajax = function() {

      $.ajax({
        type:'get',
        url:'',
        // 引入 一个 async 属性 async 默认是 true
        // async 是true 属性 进行的 是 同步机制 当 async 是 false 时 执行的是 单线程机制
        async:false,
        succes:function(res){

          ans = res

          return ans

        }
      })
    }
  </script>
</body>
</html>
```

node 连接数据库 相关知识

node 连接数据库 步骤

```
运行node 文件 指令 是 node + 文件名
```

```
npm i mysql
```

```
// 1 导入 mysql 数据库
const mysql = require('mysql')

// 2 建立MySQL数据库的联系
```

```
const db = mysql.createPool({

  // 数据库的 ip 地址
  host:'127.0.0.1',

  // 登入数据库的账号
  user:'root',

  // 登入数据库的密码
  password:'admin123',

  // 指定要操作的数据库
  database:'class'

})

// 数据库的操作指令
const sqlStr = 'select * from tc'

db.query(sqlStr,(err, results) => {
  if(err){
    return console.log(err.message)
  }
  console.log(results)
})
```

浮动解释

使用有浮动的时候注意父盒子

实训

使用 router 实现隔页传送信息

```
if(this.identity === '管理员'){

    this.$router.push({ path: '/Maner',query: {name: this.name} })
    }

    else if(this.identity === '用户'){

    this.$router.push({ path: '/user',query: {name: this.name} })

}
```

```
去除router-link 下划线问题

.router-link-active {
    text-decoration: none;
}
```

```
import userInfo from '@/components/user_info.vue'
import deptInfo from '@/components/dept_info.vue'
export default {

components:{
userInfo,
deptInfo
},
```

资料

主机

```
主机名 006.3vftp.cn
用户名 typecoh
密码  p2064q8203
域名  http://typecoh.web3v.work
```

git发布步骤

1. ```
   1. git add .
   2. git commit -m "test"
   3. git pull --rebase origin master
   4. git push -u origin master

   协同 -- 分支 创建 和 切换：

   首先

   1.进入文件 使用 git clone + 地址

   2. 是用 git branch （查收 当前的 分支）
   3. git branch + 新建分支
   4. git checkout + 切换到当前分支

   协同 -- 将创建好的 分支 推送到 本地仓库当中：

   1.查看当前分支 git branch

   2. 使用 git checkout + 分支 切换到 想 推送的 分支
   3. 使用 git push -u origin + 分支名
   ```
   ```

```
D:\vs2019\MSBuild\Microsoft\VisualStudio\NodeJs
```

web使用的辅助网站

```
//less相关使用的库
https://less.bootcss.com/
//时间相关的库
https://dayjs.gitee.io/docs/zh-CN/parse/array
//vant 使vue2 组件使用的库
https://youzan.github.io/vant-weapp/#/home
//使用 bootstrap
https://www.bootcss.com/
制作地图的数据
http://datav.aliyun.com/portal/school/atlas/area_selector
```

```
https://www.desmos.com/calculator?lang=zh-CN
```

```
https://www.w3cschool.cn/article/5698195.html php学习路线
```

```
https://www.limfx.pro/ReadArticle/57/yi-zhong-xie-zuo-de-xin-fang-fa markdown 基于vscode 使用
```