##### 其他注意点



必须手动添加分号的两种情况：立即执行函数 和 数组中括号之前[]

;[a, b] = [b , a]

let c = 1

[a, b] = [b , a] // 此时不加分号会报错，js引擎内部默认会把1和[]连在一起

ecmascript是一种由Ecma国际在标准ECMA-262中定义的脚本语言规范

javascript是它的一种实现

前者是后者的规格，后者是前者的一种实现

编码 encodeURI(“字符”)

解码 decodeURI(编码后的英文字符)

checked = “true” 多选框是打上勾的，因为该属性只能接受布尔值,字符串的true默认隐式转换为true，字符串只有为空才是false

<input type=”date” value=”xxxx-xx-xxTxx:xx”> 需要这种格式才能把日期显示出现 

location.reload() 刷新页面

在js中对象是没有length属性的 但是通过该方法可以知道对象可枚举属性的个数Object.keys(obj).length

hasOwnProperty()方法在Object.prototype上

有些时候可以将json数据获取到页面上存放，用来存放的盒子display:none

const a // const定义的常量必须有初始值,向这样,运行就会直接报错

js写代码后面加 ；养成良好的习惯

null作用：初始化组件，函数，销毁函数，占位

html页面的script代码块报错，不会影响另外一个script代码块的执行

()用来表示表达式。不管里面放的是啥,都是表示表达式

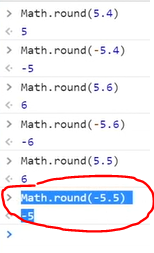
多个变量声明 一个var a = 1,

b = 3; 而不是一直使用var 单独声明，let也一样

for循环遍历，length使用变量缓存起来， obj.length 会先找对象然后再找对象下的属性,会消耗性能,所以需要缓存起来

必须给ul和li加上类名，不可以省略了，其他标签也都经量加上类名

对函数传入参数时，看看是否要对参数进行兼容性处理，比如传了一些不符合类型的参数

 Math.round(-5.5) // -5

定时器记得清除

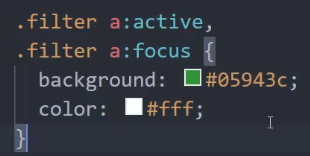
##### css部分

想要完成类似于vue中路由切换的效果，可以使用iframe

在iframe中添加name属性，后续a标签跳转链接时，将target指向该iframe的name



选中按钮样式切换，css处理



使用事件委托时的排他处理

将父元素下的类名为active的元素节点选中，清除它的active类

this.querySelector(".active").classList.remove("active")

给当前选中的元素添加active类

e.target.classList.add("active")

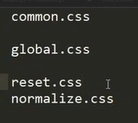
body默认的外边距是8个像素

布局外层一定要有div 包裹容器可以类名定为wrap

颜色一定不能使用英文，各个浏览器对于颜色的认知不同，可能会显示出不同的颜色

两栏布局方式

[【实战】自动阅读插件开发-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175290452673902?_wv=2147487745) 开头



自己写的标签样式初始化的文件，不用reset和normalize命名，

外部有库文件与其重名，为了方便后期维护(有可能后面会引入一些库文件)，不使用它

一般用common 或者global也可以

border: 0 有兼容性问题 border: none 没有兼容性问题

单标签要记得闭合 <input …. />

css级联样式 从右到左开始查找 div h1 p {} 先从p开始

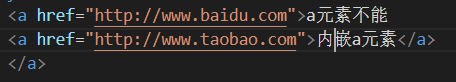
[【实战】模块化开发Todolist-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175294747641198?_wv=2147487745) 34分钟

下面这个类名的意思是 和js对接使用的一个Dom元素 j就是js的意思了 显示input



外层盒子类名写了，ul的类名就不写，直接写li的类名，反之亦然

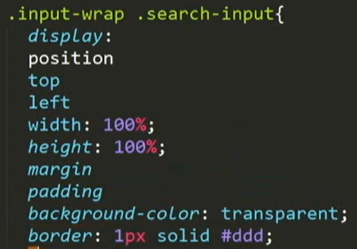
a元素不能内嵌a元素,就算将他们转换为块级元素



最终解析如下：

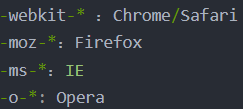


css样式书写顺序



浅灰色 #989898比较好的一个灰色

css样式兼容性



##### 栈内存和堆内存

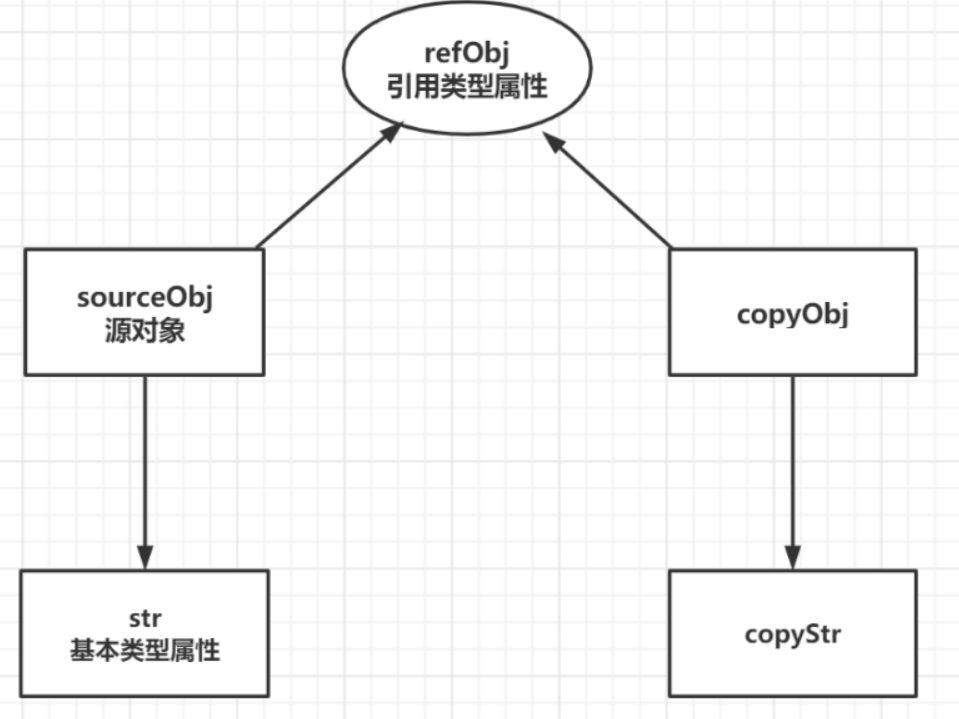
###### 区别

栈（stack）为自动分配的内存空间，它由系统自动释放。  
堆（heap）为[动态分配内存](https://so.csdn.net/so/search?q=%E5%8A%A8%E6%80%81%E5%88%86%E9%85%8D%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "_blank)空间，大小不定，不会自动释放。  
基本数据类型都是保存在栈当中。  
引用类型都是保存在堆当中，引用类型的值都是对象，保存在堆内存，而栈中存储的是对象的标识符以及对象在堆内存中的存储地址。

栈内存中的变量一般都是已知大小或者有范围上限的，算作一种简单存储。而堆内存存储的对象类型数据对于大小这方面，一般都是未知的。

###### 浅拷贝

创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值，如果属性是引用类型，拷贝的就是内存地址 ，所以如果其中一个对象改变了这个地址，就会影响到另一个对象。



soureObj是原对象，它包含基本数据类型属性str和引用数据类型属性refObj, 创建一个新对象copyObj，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值，如果属性是引用类型，拷贝的就是内存地址 ，这个对象改变了这个地址，就会影响到另一个对象。 所以str跟copyStr是不同属性，互不影响，但引用数据类型refObj内存地址相同，改变会对另一个对象产生影响。

简单来说可以理解为浅拷贝只解决了第一层的问题，拷贝第一层的基本类型值，以及第一层的引用类型地址。

实现浅拷贝的方法： （重新在堆中创建内存，拷贝前后对象的基本数据类型互不影响，但拷贝前后的对象引用类型因共享同一块内存，会相互影响。）

对象

1、Object.assign()

2、…(ES6扩展运算符)

数组

1、Array.prototype.slice()

2、Array.prototype.concat()

3、Array.from()

4、…(ES6扩展运算符)

###### 深拷贝

将一个对象从内存中完整的拷贝一份出来,从堆内存中开辟一个新的区域存放新对象,且修改新对象不会影响原对象

深拷贝会拷贝所有的属性，并拷贝属性指向的动态分配的内存。当对象和它所引用的对象一起拷贝时即发生深拷贝。深拷贝相比于浅拷贝速度较慢并且花销较大。拷贝前后两个对象互不影响。

实现深拷贝的方法：

1、JSON.stringify()深拷贝

弊端

如果 obj 的属性有时间对象，则 JSON.stringify 后再 JSON.parse 的结果，时间将转为字符串的形式，而不是对象的形式

如果 obj 的属性有 RegExp (正则表达式的缩写)、Error 对象，则序列化的结果将只得到空对象；

如果 obj 的属性有函数（function），undefined，Symbol， 则序列化的结果会把 function 或 undefined 丢失；

如果obj 的属性有 NaN、Infinity 和 -Infinity，则序列化的结果会变成 null

JSON.stringify() 只能序列化对象的可枚举的自有属性，例如如果 obj 中的对象是有构造函数生成的， 则使用 JSON.parse(JSON.stringify(obj)) 深拷贝后，会丢弃对象的 constructor；

如果对象中存在循环引用的情况也无法正确实现深拷贝，并且会导致死循环，最后抛出异常

-----------------------------------

其他常用场景

判断数组是否包含某对象： 将数组和要比对的对象转换成字符串后，比对字符串包含关系。（对象顺序很关键）

判断对象是否相等（对象顺序很关键）

让 localStorage / sessionStorage 可以存储对象

2、采用递归去拷贝所有层级属性

解决循环引用问题，我们可以额外开辟一个存储空间，来存储当前对象和拷贝对象的对应关系，当需要拷贝当前对象时，先去存储空间中找，有没有拷贝过这个对象，如果有的话直接返回，如果没有的话继续拷贝，这样就巧妙化解的循环引用的问题。

3、lodash函数库实现深拷贝,lodash很热门的函数库，提供了 lodash.cloneDeep()实现深拷贝

[(2条消息) JS 中赋值，浅拷贝，深拷贝的区别\_浪在前端的博客-CSDN博客](https://blog.csdn.net/weixin_47074432/article/details/112919142)

[前端开发之js栈内存和堆内存的区别\_at小白在线中的博客-CSDN博客\_js栈内存和堆内存的区别](https://blog.csdn.net/m0_58371965/article/details/122820541)

##### typeof

typeof("1" - "1") // number

typeof("a" - "b") // NaN

typeof("1" - 1) //number

typeof(true - false) // number

typeof(“ ”) // string

typeof(“”) // string

typeof(a) // a没有被定义 返回undefined

typeof(typeof(a)) // 返回string 因为typeof检查出的结果是一个字符串形式的文字

两数互换 [a, b] = [b, a]

##### 与 或

“” || “输出这个”

“这个不输出” && “输出这个”

||：前面为真直接返回，为假返回后面的

&&：前面为真返回后面的，为假直接返回

##### 自增 自减问题

let a = 5;

let b = --a + a--

console.log(a, b) // a = 3, b = 8

---------------------

let b = --a + --a

console.log(a, b) // a = 3, b = 7

##### 算数运算符

任何数据类型的值+字符串，都是字符串

1 / 0 = Infinity -1 / 0 = -Infinity

0 / 0 = NaN

Infinity != -Infinity

-0 === +0

NaN ===(或==) NaN 结果为false NaN与任何数据类型都不相等,包括自己

引申的考题 如何让a != a 成立 a赋值为NaN

NaN===NaN 、-0===+0涉及到了同值相等的问题。

移步《其他碎片化知识》—>同值相等

\* / - 运算中有字符串和数字进行计算，就会将字符串隐式转为number类型

let str = “123”

console.log(+ str) // 类型为number的123

console.log(- str) // 类型为number的-123

##### 比较预算符

var bool = "1.5" > "11"

console.log(bool) // false

解析：字符串比较是每个位置通过ascii码(字符相对应的十进制代码)进行比较。

1的ASCII码为49 .的ASCII码为46 1和1比较相同，往下比较 .和1比较 .小于1 所以为false

字符串之间比较，比较的是Ascii码

字符串和数字比较 字符串隐式转换成数字

undefined , null 即不大于0 也不小于0 也不等于0

undefined > 0 // false

undefined == 0 // false

undefined < 0 // false

undefined == null // 结果：true

原因：要比较相等性之前，不能将 null 和 undefined 转换成其他任何值，并且规定null 和 undefined 是相等的。null 和 undefined都代表着无效的值。

if(typeof(a) && (-true) + (+undefined) +""){

console.log(“通过了”)

}else{

console.log(“么有通过”)

}

结果：通过了

解析：**typeof(a) 输出字符串的 “undefined”**

(-true) 输出-1

+undefined 输出NaN

-1 + NaN = NaN

**NaN + “” = “NaN”**

##### undefined和null的区别

**相同点**

第一：Undefined和Null两种数据类型都只有一个字面值，分别是undefined和null.

第二：Undefined和Null类型在转换为Boolean类型的值时，都会转换为false.

第三：在需要将两者转换成对象的时候，都会抛出一个TypeError的异常。

第四：Undefined类型派生自Null类型，所以在非严格相等的比较下，两者是相等的。

**不同点**

第一：null是JavaScript的关键字，而undefined是JavaScript的一个全局变量，也就是挂载在window对象上的一个变量，并不是关键字。

第二：在使用typeof运算符进行检测时，Undefined类型的值会返回undefined.而Null类型的值返回为object

第三：在进行数值类型的转换时，undefined会转换为NaN,无法参与计算，而null会转换为0,可以参与计算。

第四：建议：无论在什么情况下都没有必要将一个变量显示的赋值为undefined。如果需要定义某个变量来保存将来要使用的对象，应该将其初始化为null.

##### if判断语句

多个if 和if else 的区别：多个if当找到满足条件的if时还会向下运行其他if；使用ifelse 满足条件就结束

为什么要有else，判断条件之外的异常情况处理

##### 返回false的6个数据

0 "" undefined null NaN false 除此6个之外全部是true

注意： “ ”这个是true

##### 写法

let name = ""

console.log(name || "未找到数据")

##### 鼠标点击事件兼容性处理

a.onclick = function(e){

// 在ie的低版本中鼠标点击事件的参数是保存在window.event中的,其他浏览器是直接在e中.

let event = e || window.event // 兼容性处理

}

##### ！的小妙用

let a = 1

!!a // 两个！！将数据类型进行转换成boolean数据类型

##### 注释写法的注意点

/ \*

\* 写注释 每一行前面都加个\*

/ \*

// 比较运算符

! && || 对于非布尔值进行运算时，会先将其隐式转换为布尔值，然后再运算

##### 终止循环

要想终止一个循环,可以使用break，exit 0 判断条件的变量 i = 0 return

##### for循环面试题

for（单次表达式;条件表达式;末尾循环体）{中间循环体；}

打印0-100 for循环中 ()只能有一句，不能写比较 {}不能出现i++ i--

var i = 101

for(; i-- ;){

console.log(i)

}

##### isNaN

isNaN()在接收到一个值之后，会尝试将这个值转换为数值，如果能成功转换则返回false，否则返回true

isNaN(“1STR”) // true

isNaN(“str”) // true

isNaN(“123”) // false

isNaN(NaN) // true

isNaN(undefined) // true

isNaN(null) // false

isNaN(“”) // false

isNaN() // true

如果想要判断一个值是否严格的等于NaN，推荐使用Number.isNaN()方法，此方法会判断传入的值是否为NaN这个值，如果是，则返回true；否则返回false；

isNaN()函数并不是严格的判断传入的值是否等于NaN，只是将传入的值尝试转换为数值型，如果能，则返回false；如果转换失败，则返回true；

##### 基本数据类型的包装类 String() Number() Boolean()

let a = "123"

let new\_a = new String(a)

console.log(new\_a) // String {'123'}

console.log(new\_a + 1) // 1231

基本数据类型是不能通过 .XX的方式来使用 .XX是引用数据类型的

包装类的问题[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=9&spm_id_from=pageDriver&vd_source=a67e813436aadc4608992466bdaea6bc) 1:01:41

var a = 3

a.len = 4

console.log(a.len) // undefined

==============

var str = “123”

console.log(str.length) // 3 原因:str是基本数据类型，在内部经过包装类包装

等同于 console.log(new String(str).length)

##### /\* 显示类型转换 \*/

###### Number()

Number("1NaN") // NaN 里面只要传入字符串不全都是数字的，都是转换为NaN

Number(“123”) // 123

Number(true) // 1

Number(undefined) // NaN

Number(null) // 0

Number(“”) // 0

Number(“ ”) // 0

###### parseInt(string, radix) 函数可解析一个字符串，并返回一个整数。数字同理。会丢弃小数点后面的数字

**注意： 只有字符串中的第一个数字会被返回。**

**注意： 开头和结尾的空格是允许的。**

**注意：如果字符串的第一个字符不能被转换为数字，那么 parseInt() 会返回 NaN。**

**注意：在字符串以"0"为开始时旧的浏览器默认使用八进制基数。ECMAScript 5，默认的是十进制的基数。**

parseInt("1.3") // 1

parseInt(“13str”) // 13

parseInt() // 里面传入undefined，null，boolean(true && false) 结果都为NaN，不传也是

###### parseFloat(string) string 必需。要被解析的字符串。传入数字也行

parseFloat() 可解析一个字符串，并返回一个浮点数。

parseFloat(“1.2str”) // 1.2

###### toString(radix) 转换成字符串

radix可选。规定表示数字的基数，是 2 ~ 36 之间的整数。若省略该参数，则使用基数 10。

let a = 123

a.toString() // “123”

注意: 不能直接123.toString()

因为 JavaScript 解析器把 . 和 123连在了一起。

因为 JavaScript采用 IEEE 754 的规范 双精度数字，JavaScript 中只有一种数字类型：基于 IEEE 754 标准的双精度。它并没有为整数给出一种特定的类型,所以所有的数字都是小数。

可以使用(123).toString() 或 123..toString() 或 123 .toString() 【中间有空格】

##### /\* 隐式类型转换 \*/

let a = “123”

a++

console.log(a) // 124

##### String

###### 字符串操作的文章

[js常用字符串的方法的整理（查、修改、拼接） - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/357685379)

###### 字母转换大小写

str.toUpperCase()

str.toLowerCase()

str.toLocaleLowerCase()

str.toLocaleUpperCase()

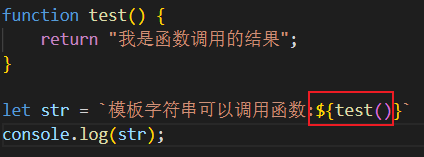
针对一些比较特殊语言，比如土耳其语等语言中有特殊的大小写字母，这些语言的大小写转换需要使用Locale.

英文字母两种都可以

###### 模板字符串(ES6)

`QWE${NAME}`

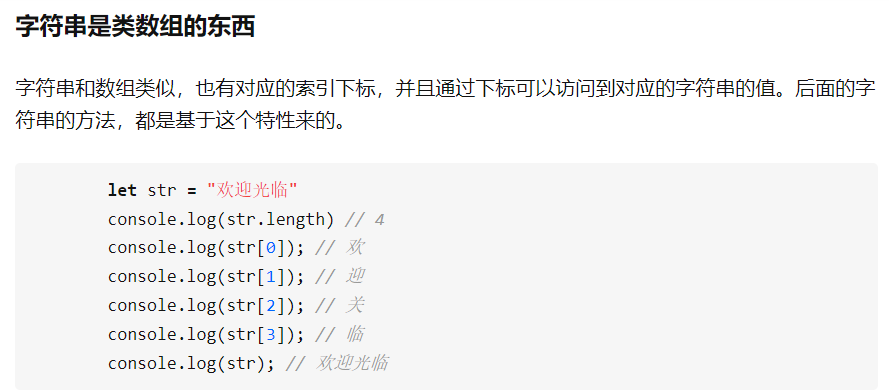
可以调用函数



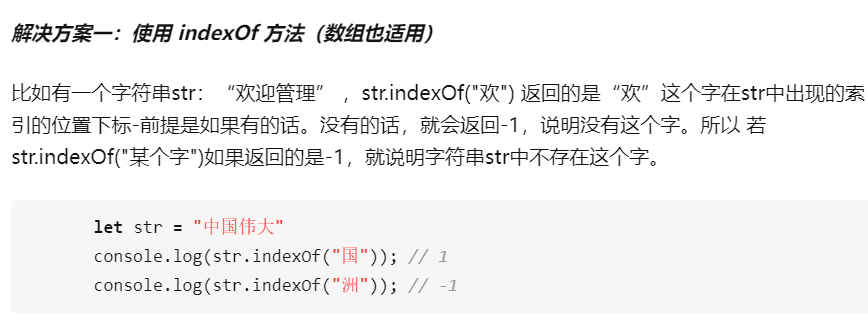


###### 获取字符串字符的指定位数

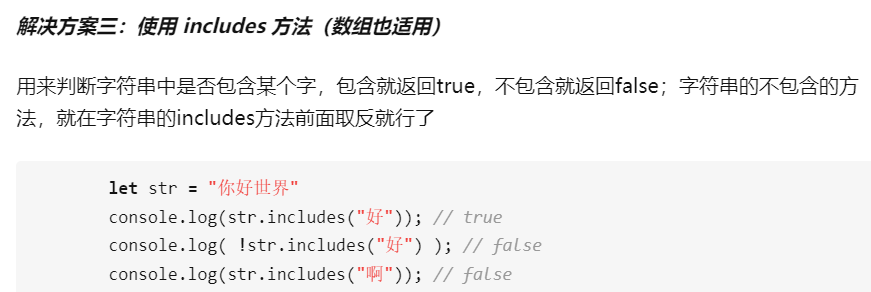
str.charAt(i)



###### 如何查询一个字符串中是否包含某个字（词）





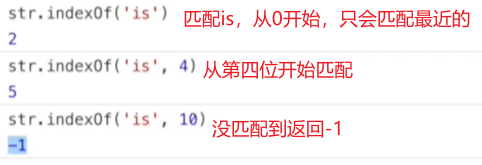


###### 获取字符的unicode编码和将编码转换成字符



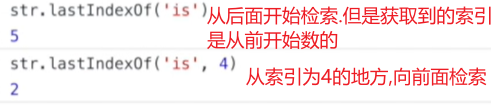
###### indexOf





###### lastIndexOf





###### substr()

从后往前数(从-1开始数),从-4位置(包括-4)开始截取4位





从前面开始数10位，包括索引10的位置,截取4位





###### substring()

从前面开始数到索引为10的,截取包括索引10的字符,截取到索引为13的，

但是后面是开区间，需要写14





倒过来也可以

这是因为js引擎在匹配之前会将10和14倒个位置。

substring()方法不存在倒着数的



两数相等返回空字符串



###### slice(start，end(开区间))

切割字符串

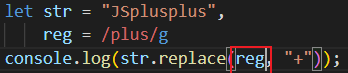
###### ※replace() 非常重要

String.replace() 只能匹配一次,不存在全局匹配



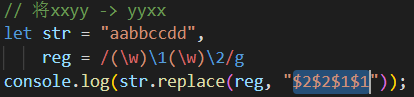
结果

使用正则匹配的表达式模式，实现全局匹配



结果

xxyy进行转换成yyxx



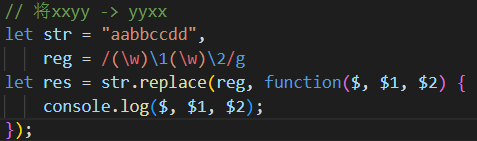
$1是第一个子表达式 (\w)

$2是第二个子表达式 (\w)

replace()第二个参数

下面不一定使用$符号，但是参数位置是一一对应的

还可以传入一个函数，函数由replace()里边帮我们调用



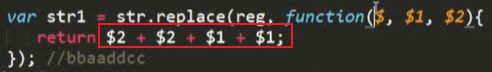
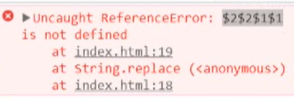
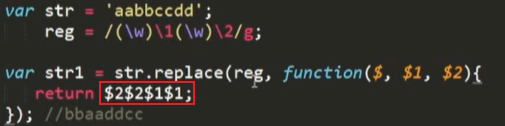
结果

$是字符串里面被匹配出来的字符串

$1是第一个子表达式

$2是第二个子表达式

下面这种写法会报错 $2、$1是一个变量 需要使用 + 进行连接



##### toString()和valueOf()

toString()接收一个参数 基数-radix valueOf()没有参数

toString()返回字符串 valueOf()返回原本的值(原始值)

1. 数组 toString() 返回用逗号分隔字符串

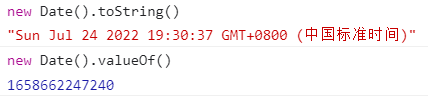
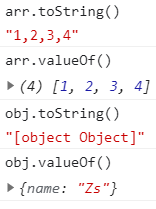
valueOf() 返回数组原本的值

2、对象 toString() 返回对象的类型字符串表示 “[object Object]”

valueOf() 返回对象原本的值

3、Date toString() 返回时间特有的字符串

valueOf() 返回13位毫秒时间戳



##### 字符串转为数组

Array.prototype.slice.call(str) // 不是字符串上的slice，是数组原型上的slice

str.split(“”)

Array.from(str)

[…str]

##### 数组

###### 数组原型是一个数组



###### 数组创建方式

数组字面量：let arr = [ 1, 2, 3]

构造函数： let arr = new Array(1,2,3)

如果传入的是单个值 new Array(2) => [empty x 2] 长度为2

###### 通过length截断数组

字符串不行这么使用 let arr = “123”

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

arr.length = 2

console.log(arr) // [1, 2]

###### 数组理解

数组是一种特殊的对象，只是以这种形式表达

arr.length 在js中有自己属性的都叫做对象

###### join / split

let arr = [“a”, “b”, “c”, “d”]

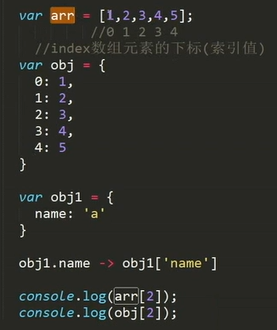
let str = arr.join(“-”)

let arr = str.split(“-”, 3) // 截取前三个

[15\_深拷贝实例\_数组基础\_数组方法\_数组排序-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/11829366430696814?_wv=2147487745)

15分钟

###### 数组底层机制，在js中就是继承了对象而来的



###### 稀松数组

let arr = [,,] 结果是长度为2的数组，最后一个逗号右边的空位被丢弃

let arr = [, 1, 2, 3, ] 长度为4 [empty, 1, 2, 3] // empty并不是值,只是用来标识这一项是空的

但是使用内置构造函数Array创建的数组，传入的参数，不能有空位 let arr = new Array(1,2,3) 不然会报错

数组字面量方式创建的数组已经成型，构造函数的传入的参数肯定不能有空位

###### sort()方法的理解

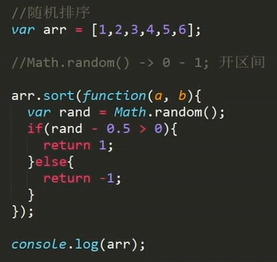
[15\_深拷贝实例\_数组基础\_数组方法\_数组排序-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/11829366430696814?_wv=2147487745)

1小时23分钟

使用sort随机排序

[15\_深拷贝实例\_数组基础\_数组方法\_数组排序-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/11829366430696814?_wv=2147487745)

1小时43分钟



arr.sort((a, b)=>{

return Math.random() – 0.5

})

###### 数组方法

所有数组扩展方法的本质就是遍历循环，然后将每个元素在回调函数里进行操作

ES5添加的扩展：forEach(),filter(),map(),some(),every(),isArray()

ES6添加的扩展：reduce(),from(),find(),findIndex(),includes()

forEach()

arr.forEach((item, index, array)=>{  
 console.log(this) // this指向window

})

arr.forEach((fn(){}, {test: 1})

第二个参数(得是一个对象,this指向一定是指向一个对象)，修改回调函数内的this指向

some,every,filter,map同forEach都使用第二个参数修改this指向

this就是oLi元素节点对象



reduce()方法用来总结归纳数据，很重要

[every/some/reduce/reduceRight使用与重写-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175307632543086?_wv=2147487745) 28分钟

Array.from()





###### 将对象转为数组

Array.from(对象) // 普通对象会转为空数组[]，伪数组会转为数组

Array.prototype.slice.call(对象) // 普通对象会转为空数组[]，伪数组会转为数组

###### 面试注意点：

数组方法继承至数组Array原型上的方法

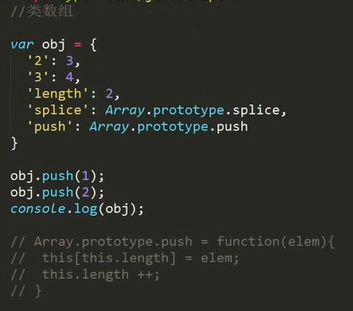
push、unshift方法的返回值，是执行了方法以后的数组长度

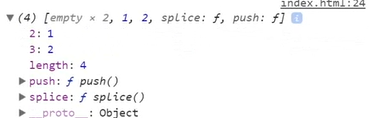
数组方法sort() 不传参数是按照数组元素的ascii码来比较，跟字符串之间的比较一样

##### 类数组

和数组的区别：类数组的\_proto\_指向Object.prototype 数组的\_proto\_指向Array.prototypes

###### 阿里恶心面试题





看着被注释的push方法原理来思考

类数组不一定都是继承至object.prototype

有部分的内置的类数组是继承Array.prototype

类数组转换成数组

Array.prototype.slice.call(arguments)

Array.from(arguments)

##### 函数

###### 构造函数：

特殊的函数，主要是用来初始化对象的，创建多个类似的对象

###### argument和function (…args){}剩余参数

argument获取的是一个伪数组.也可以通过slice截取,返回指定的剩余参数

…args获取的是一个真数组,…args可以获取剩余参数，或全部参数

功能其实都差不多，但是箭头函数是没有arguments的

函数内部变量命名前面加一个 \_ 表示这个变量是私有的。这不是一定要这样,这么写会规范一些

【

var test = function test1(){

let a = 1,

b = 2;

console.log(a, b)

}

test() // 1, 2

test1() // 报错,函数表达式中使用变量名来直接调用,函数名会被忽略。但是函数名可以在函数内部被调用，实现递归

小知识：函数内部最后面如果没有写return,系统会默认给你添加一个return

】

【

function test(a, b){

console.log(a, b)

}

test(1, 2, 3) // 报错吗？不报错，输出1，2

function test(a, b, c){

console.log(a, b, c)

}

test(1, 2) // 报错吗？不报错，输出1，2，undefined

总结：形参和实参数量可以不等

】

【

**argument的应用**

function test(a, b, c){

console.log(arguments) // 获取传入的实参

}

// 实参求和/求积

function sum(){

let a = 0

for(let i = 0; i < arguments.length; i++){

a += argument[i]

}

return a

}

】

【

function test(a, b){

console.log(test.length) // 获取形参数量

}

test(1, 2, 3)

】

【

重要

function test(a, b){

a = 3

b = 4

console.log(a, b)

}

test(1) // 结果：3，4

function test(a, b){

a = 3

b = 4

console.log(arguments[0], arguments[1])

}

test(1) // 结果：3，undefined

描述：

arguments接收一个调用函数时传递进来的实参数组,数据存放在堆内存中,在栈内存中存放对应的指针.

3赋值给了形参a, 值是存在栈内存中。他们不是同一个东西。

但是实参和形参之间存在映射关系,实参变,形参会跟着变;形参修改,打印出来的arguments实参值也会跟着变。但是前提是传入了对应的实参值与形参形成了映射关系,否则你调用arguments中未传入的实参值,都会是undefined

】

【

***function test(a = 1, b){***

***console.log(a, b)***

***console.log(arguments[0])***

***}***

***test(undefined, 2) // 1, 2 undefined***

function test(a = undefined, b){

console.log(a, b)

}

test(1, 2) // 1, 2

】

【

给参数赋默认值是ES6的语法,有些老版本的浏览器不支持,可以使用如下方法

function test(a, b){

a = arguments[0] || 1

b = arguments[1] || 2

console.log(a, b)

}

问题：传的实参为0，会使用默认值

也可以使用如下方法

function test(a, b){

a = typeof(arguments[0]) !== “undefined”? arguments[0] : 1

b = typeof(arguments[1]) !== “undefined”? arguments[1] : 2

console.log(a, b)

】

【

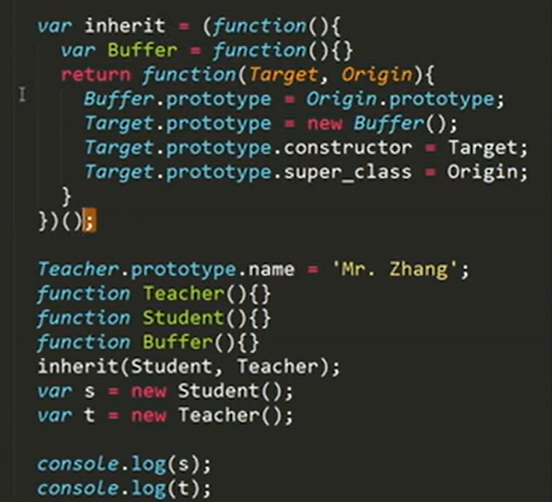
    /\*

     \* Student.prototype === Teacher.prototype，如果Student.prototype 上修改了,Teacher.prototype也会被修改

     \* 通过中间的构造函数，来解决上述问题

     \* 封装对应操作

     \*/



】

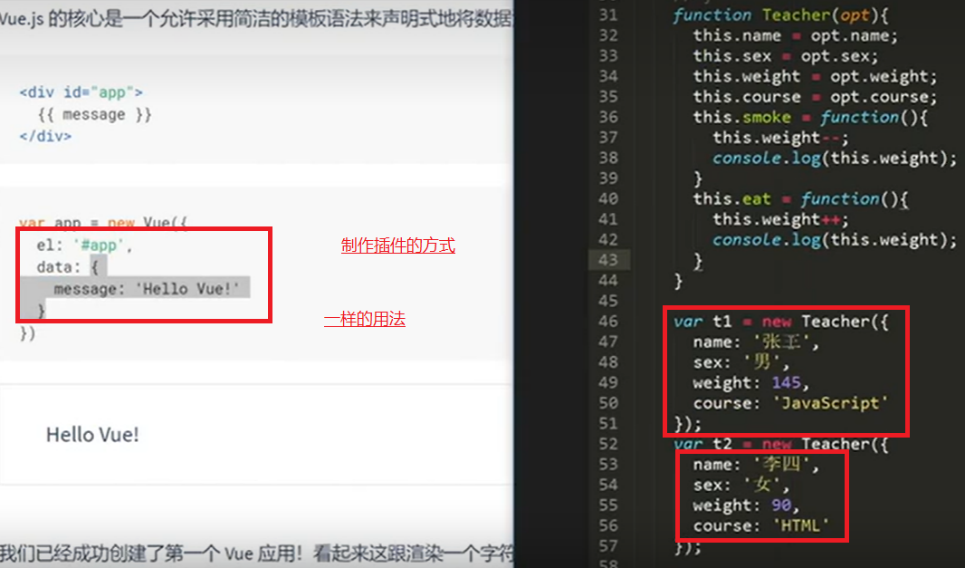
##### 对象

①对象的{}不属于块级作用域，像for(){},if(){},else{}，try{},cath(){}等等的花括号才是块级作用域

②对象的{}的作用域是什么作用域取决于对象所处的作用域，比如对象在全局作域 下定义的，那么对象的{}的作用域就是全局作用域

删除对象中的属性 delete obj.attrs

删除对象中的方法 delete obj.fn fn不能加括号,那是调用



###### 创建对象的三种方式

let obj = new Object()

let obj = { name :”zs”}

let obj = new Foo() // 自定义构造函数创建

let obj = Object.create({name:”zs”}) // 参数是对象的原型

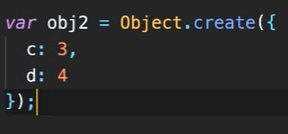
###### 对象都是通过构造函数实例化出来的



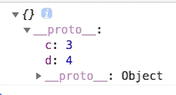
上述的var arr = [] var obj = {} 只是通过字面量的方式去创建对象，底层还是通过new的方式去帮我们实例化对象

###### Object.create()

创建对象时指定原型



obj2的原型就是指向create里面传入的对象参数 {c: 3, d: 4}



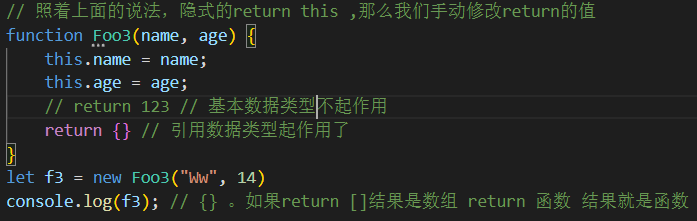
创建无原型的对象



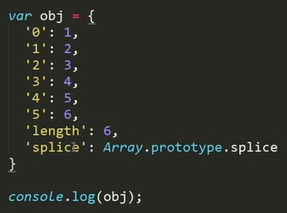


###### 对象中this的问题





###### 对象模拟一个伪数组的样子





splice: Array.prototype.splice 会将结果上的{} 变成[]

##### 递归

找到规律，给出出口, 否则死循环

##### js预编译

一、语言分析

​ JS 会对全部的脚本代码进行解析，通过jS引擎检查你的代码是否存在错误，如果有格式错误就会抛出一个错误，否则语言分析完成。

二、预编译

​ 全局创建

​ 1、创建GO空对象

​ 2、变量声明提升 赋值undefined

​ 3、函数声明提升 并把函数体赋值

当有函数准备执行时：编译函数方法体 局部作用域

​ 1、建立AO空对象

​ 2、变量提升

​ 2.1形参声明提升 赋值undefined

​ 2.2实参声明提升、变量声明提升 赋值undefined

​ 2.3实参形参相统一

​ 3、函数声明提升 并把函数体赋值

三、方法体解析执行

当执行 a = 23,AO对象、GO对象中都没有var a,则会在GO对象中创建一个a:23;且解析执行完毕并不销毁

解析执行完毕AO对象销毁

页面关闭GO对象销毁

注意：在解析执行中 AO才开始创建

1 只有存在作用域的时候才会用预编译

2 es5之前 每个作用域都需要预编译

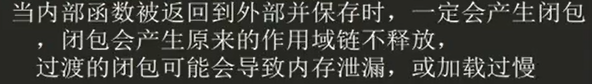
3对于同一作用域一定是先预编译在执行 不同作用域之间不存在编译和执行顺序的

##### 闭包

###### 基础理解

[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=6&spm_id_from=pageDriver&vd_source=a67e813436aadc4608992466bdaea6bc)

1小时54分处



function test1(){

function test2(){}

var a = 1

return test2

}

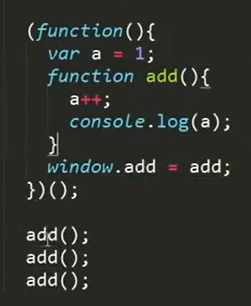
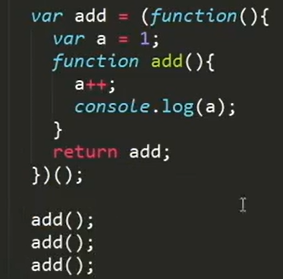
var c = 3

var test3 = test1()

随便看看

// GO里面保存了test1函数 c变量 test3函数

// test1被执行的时候，创建了自己的AO，同时定义了test2，test2产生了自己的作用域链

这两个结果都是2，3，4

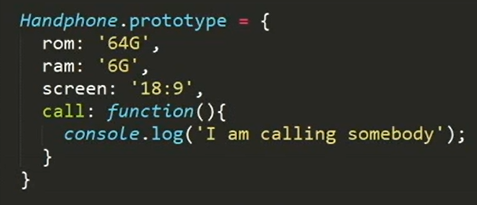
window.add = add 就是将函数内的add函数传给了全局的add。

和return有着异曲同工之妙

##### 原型

实例对象上原型的属性写法（重写原型对象）

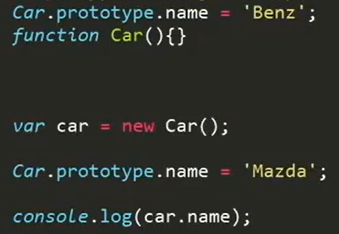
还有一个其他的写法：构造函数名.prototype.rom = “64G” 重新赋值



原型的constructor属性指向构造函数本身,该constructor属性指向的构造函数可以在原型上被修改

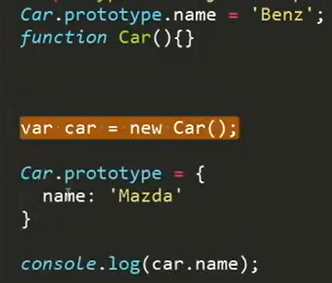
实例对象的\_\_proto\_\_指向的原型对象也可以被修改。虽然没有什么使用的场景,但是可能会问

###### 小案例注意点



结果为Mazda

Car.prototype.name = “Mazda” 这是给name重新赋值



结果为Benz

Car.prototype = {} 是在构造函数实例化对象之后重写了实例对象的原型。不会影响已经之前实例化的的对象,虽然原型已经被改变

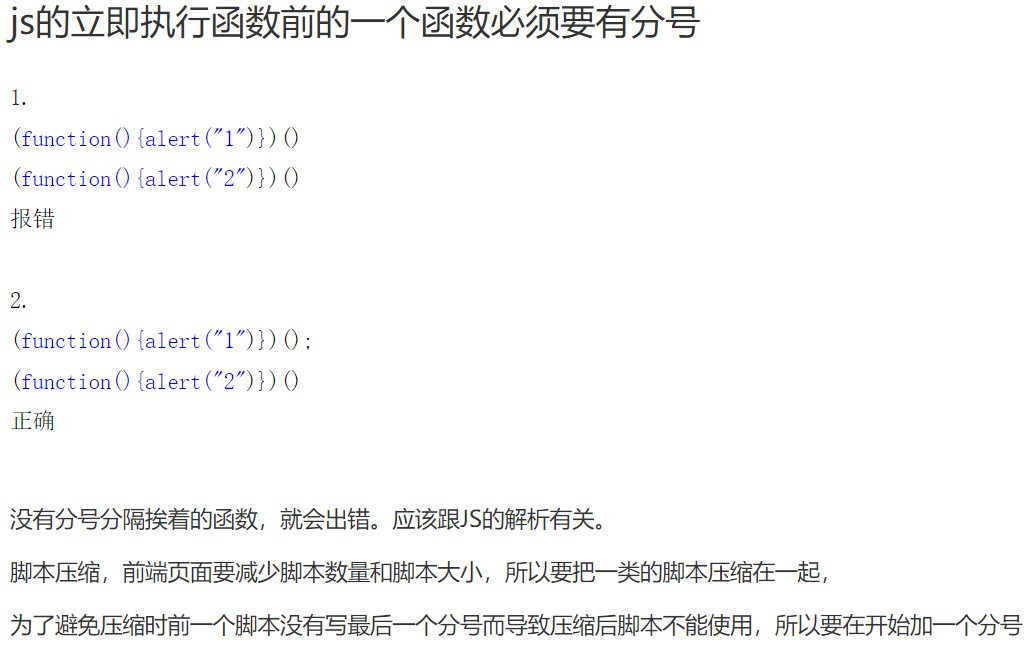
##### 立即执行函数(IIFE immediately-invoked function expression)

###### 书写方式

;(function(){})()

;(function(){}())

###### 加分号



###### 保存立即执行函数的返回值

let num = (function(){ return a + b}(2, 3))

###### 表达式说明

( )用来表示表达式。不管里面放的是啥,都是表示表达式

function test(){}() 能执行吗？ 不行，语法错误

let test = function(){

console.log(1)

}() 能执行

解析：表达式才能被执行符号()执行

function test(a, b){

console.log(1)

}

(2) // 报错吗？不报错，但是也没有值输出

js引擎会将其理解成

function(a, b){

console.log(1)

}

(2)

// 根据上面说的 ()括起来的都是表达式, 表达式里面如果放函数,那么函数名会被忽略

逗号是一种运算符,返回逗号后面的值

console.log((1, 2, 3)) 输出3

将函数声明变成表达式的方法 + - ！ || &&

!function test() {

console.log(1)

}()

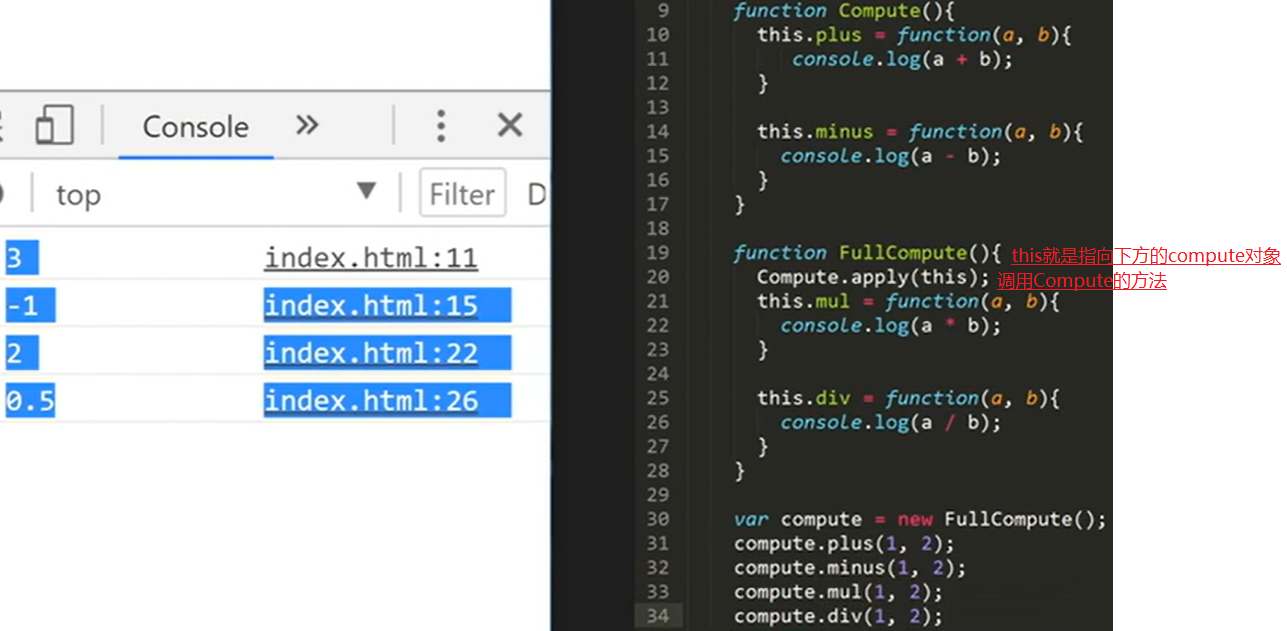
##### ASCII码和UNICODE码

ASCII码有两张表 ，1表是0-127 ，2表是128-255 在该表上一个字符都是一个字节

UNICODE码前面和ASCII完全一样，就是说0-255都是一个字节的，255之后都是2个字节

str.charCodeAt (i) 返回对应字符串字符的字节数

##### apply,call,bind



##### 插件制作

使用立即执行函数

通过window. 的方式抛出值到全局去，类似于return

下面参数也可以写在方法里面传值

// 封装简陋版插件 加减乘除

;

(function() {

    var Compute = function(opt) {

        this.firstNum = opt.firstNum

        this.secondNum = opt.secondNum

    }

    Compute.prototype = {

        add: function() {

            return this.firstNum + this.secondNum

        },

        reduce: function() {

            return this.firstNum - this.secondNum

        },

        mul: function() {

            return this.firstNum \* this.secondNum

        },

        except: function() {

            return this.firstNum / this.secondNum

        }

    }

    window.Compute = Compute

})();

let compute = new Compute({

    firstNum: 12,

    secondNum: 1

})

console.log(compute.add());

有些地方可以直接return就不用再给个值接收然后再return了

比如： function(){

let a = {}

return a // 直接return {}

}

##### CSS圣杯布局优化版

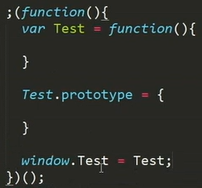
[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=12&vd_source=a67e813436aadc4608992466bdaea6bc)

56分16秒

##### 插件写法

使用立即执行函数，插入到页面中时会自动执行里面的代码.

比如说轮播图插件,页面渲染完成就会立即开始执行轮播

 插件的标准化写法（***面向对象写法***）

如果想要按需执行，可以使用一个变量接收立即执行函数中返回的对象。

###### 插件的所有方法都要写在构造函数的原型上

##### 模块化开发了解

描述：封装在一个作用域里面的功能的一个集合，这些功能可以通过模块的一些特性，导出和导入，让它可以依赖其他模块，也可以被其他模块依赖

使用立即执行函数封闭一个独立的作用域，防止全局的命名污染

模块化开发算是插件的一种异类，带上变量去保存它

[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=12&vd_source=a67e813436aadc4608992466bdaea6bc)

[【实战】模块化开发Todolist-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175294747641198?_wv=2147487745) 59分钟以及1小时49分

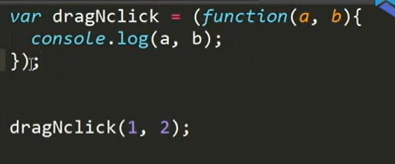
###### 模块化开发写法



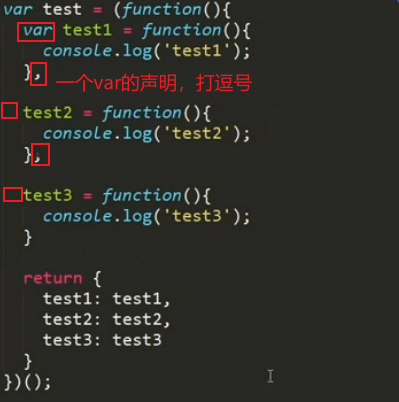
企业中不会使用window.onload 来运行(效率太低)，而是通过时间线。

***多看看vscode 【小野森森—作业(很重要)-- 京东搜索框的提示文字效果】 里面的模块化写法***

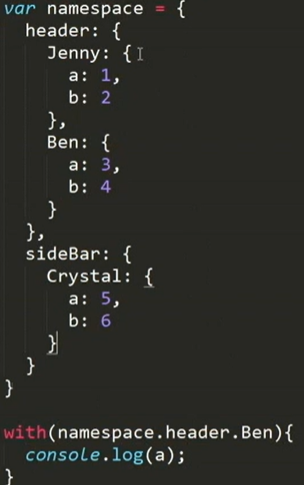
还可以这么写



这么写



###### 早期的模块化开发变量命名冲突问题

 使用with向指定作用域查找a

=============================================================

[14\_三目运算\_对象克隆\_浅拷贝\_深拷贝-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/11829362135729518?_wv=2147487745)

1. 创建独立的作用域,使用一个变量接收里面的返回值
2. 它和立即执行的插件的区别就是,它可以根据需求来调用是否执行,是可控的

var initFb = (function(){

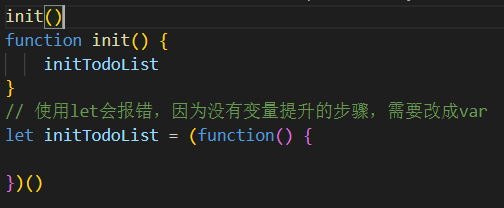
function a(){}

function b(){}

return [a, b]

})()

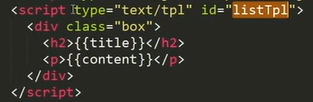
###### 写作过程中碰到的问题

###### 企业级模板写法

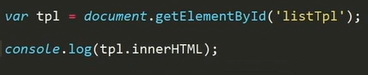
[【实战】模块化开发Todolist-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/14175294747641198?_wv=2147487745) 1小时19分

页面中



type 里面写别的东西(text/tpl)，这样网页就不会解析它

js中



笨一点的模板写法 但是运行效率是最高的



##### 链式调用原理

###### 初始版本



return this 是关键

每次调用完函数就会return this this是指向sched的,所以都可以拿到后续的结果

##### 其他的用法原理

###### 1

[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=13&vd_source=a67e813436aadc4608992466bdaea6bc)

14:57分

this[ ]的用法 中括号里面放字符串

在最早的js引擎对对象的处理都是通过中括号，后面可以使用 . 了

this.name 内部会转换成this[ “ name”]处理

###### 2 好东西

let obj = {  
 name: “Zs”,

age:12

}

for(let key in obj){

console.log(obj[key])

// 使用obj.key会失败 有的人说是因为key是字符串导致的这是错误的说法

// js引擎在内部会将 obj.key => obj[“key”] ，没找到名为”key”的key就是undefined

}

###### 3 in

let obj = {

name: “zs”,

age: 12

}

console.log(“log” in obj)

log必须用引号，其实就是查找 obj[“log”] 所以得是字符串类型

hasOwnProperty排除原型上的自定义属性，但是in不排除

function person(){

console.log()

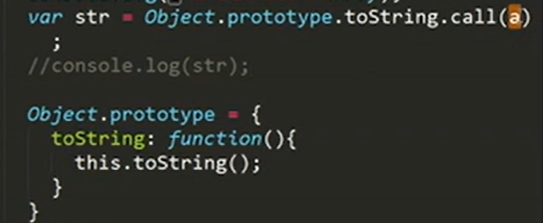
}

obj.

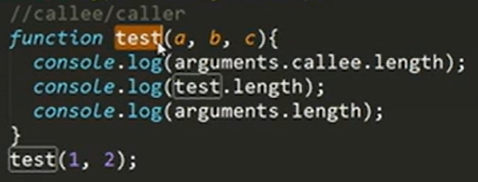
console.log(“log” in obj)

###### Object.prototype.toString.call()

this指向a 输出的就是a的toString()



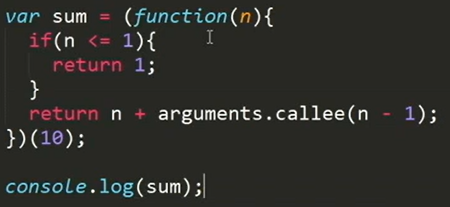
##### callee



arguments.callee指向函数本身，函数test.length = 3 所以arguments.callee.length = 3

严格模式下不能用

###### callee使用场景



arguments.callee 指向函数本身,在匿名函数中，因为函数名称无法使用，所以可以使用这个代替

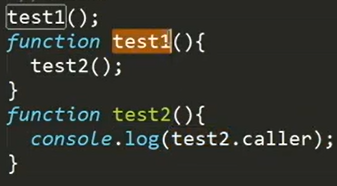
注意：找到是这个函数本身，不是函数名

test.name可以达到同样的效果

let test = function a(){}

test.name // a

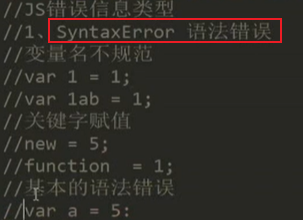
##### caller

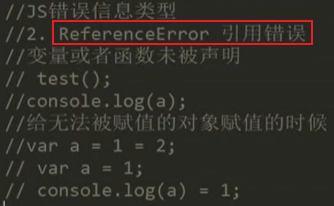


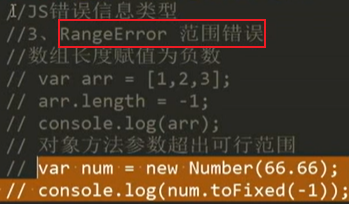
函数test2被test1使用,在test2函数内部使用test2.caller会返回调用test2函数的函数

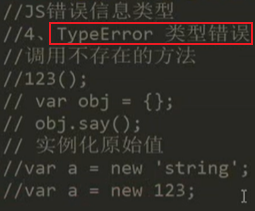
严格模式下不能用

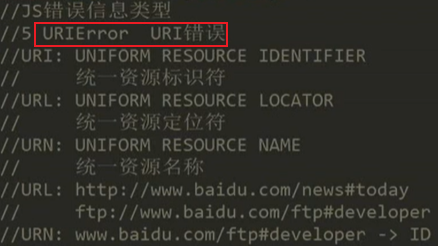
##### js中的错误类型











###### eval()



不推荐使用eval() 有性能，安全性，语法规范(传入字符串或非字符串都能执行),可读性(字符串不好读)等问题。但是在es1 es2里面运用比较广泛所以需要保留下来。ES6是完全摈弃了。

JSON.parse()也能达到字符串转换JSON对象的目的

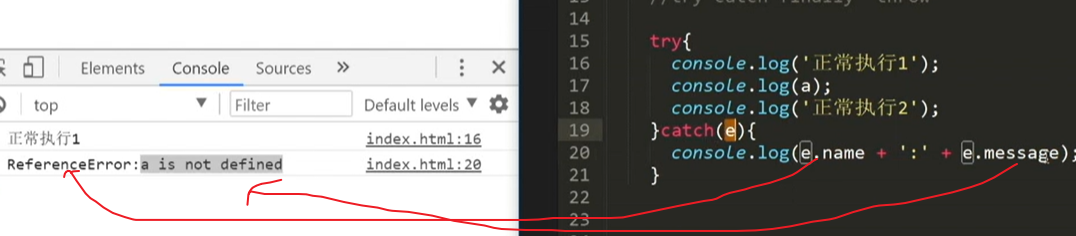
严格模式下eval()有自己的作用域,无法在外部调用eval()里面定义的变量

上面6中错误类型都对应一种构造函数，可以实例化它，人为的抛出错误



还有一个总的Error构造器 new Error()

###### try catch

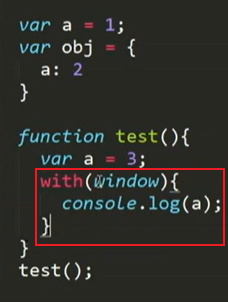


##### JSON字符串

[错误信息、try\_catch、严格模式 (qq.com)](https://ke.qq.com/webcourse/index.html#cid=386920&term_id=104499667&taid=13292197932033896&type=1024&vid=5285890787927249170) 41分钟

JSON对象里面是不可以有方法的

##### with表达式

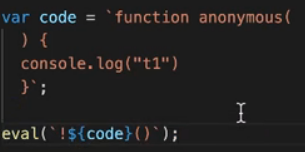
 打印 1

with(){} 括号里面传入对应的window或函数名，会将作用域指向对应的地方

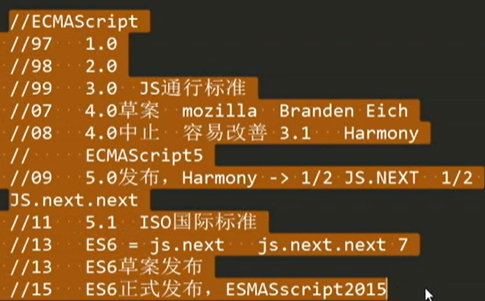
**严格模式下无法使用**

括号里面必须填值，不然会报错

##### eval()的使用



##### ECMA发布历程



[错误信息、try\_catch、严格模式 (qq.com)](https://ke.qq.com/webcourse/386920/104499667#taid=13292197932033896&vid=5285890787927249170) 1小时17分

##### 各种题

###### 逗号运算符面试题

var test = (

function fn1() {

return 1

},

function fn2(){

return “2”

}

)() // 括号写在typeof test()也行

console.log(typeof test) // string

###### 表达式面试题

var a = 10

if(function b(){}){ // (function b(){}) 是一个表达式,表达式会忽略b这个函数名 下方typeof(b)等于undefined

a += typeof(b)

}

console.log(a) // “10undefined”

###### 闭包面试题

[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=7&spm_id_from=pageDriver&vd_source=a67e813436aadc4608992466bdaea6bc)

进度条拉到45分钟。经典闭包面试题

进度条拉到52分钟。原理

function test(){

var arr = []

for(var i = 0; i < 10; i++){

arr[i] = function(){

document.write(i + “ ”)

}

}

return arr

}

var arr = test()

arr[0]() // 10

=================

function test(){

var arr = []

for(var i = 0; i < 10; i++){

(function(j){

arr[j] = function(){

console.log(j)

}

})(i)

}

return arr

}

var arr = test()

arr[0]() // 0

======================

function test(){

for(var i = 0; i < 10;i++){

(function(){

console.log(i)

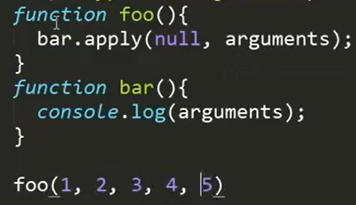
})()

}

}

test() // 0,1,2….

###### 第一道：



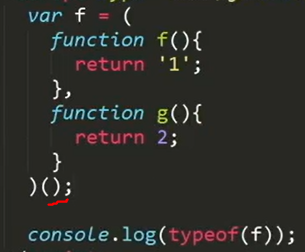
结果是 [1, 2, 3, 4, 5]

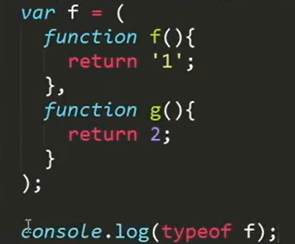
解答;

[【2020全新JS++前端课程】『JavaScript深度指南』之全线技术提升\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1x441167iH?p=13&vd_source=a67e813436aadc4608992466bdaea6bc)

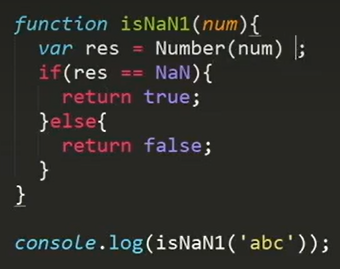
1小时26分

###### 第二道：

 结果 number

 结果 function

###### 第三道：

 输出false

因为传入的参数abc字符经过Number转换是NaN，但是NaN不等于任何数，包括自己

需要改成如下书写，改成字符串的NaN进行比较



###### 第四道：

4.1

{} == {} 吗？

结论：不等于

4.2

为什么不等于？

因为对象是引用类型，存放在堆内存中，引用地址存放在栈内存中，这是两个不同的空对象，存储在两个不同的空间中

4.3

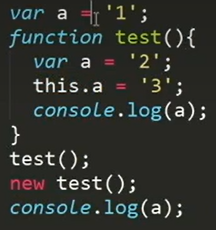
如何让他们相等？

var obj = {}

obj1 = obj

obj1 == obj // true

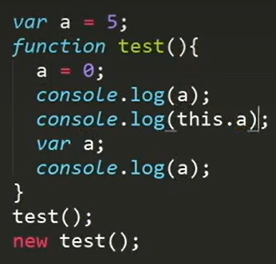
###### 第五题：



this.a指向window 修改的是GO中的a，new test()直接打印a

2，2，3

###### 第六题：

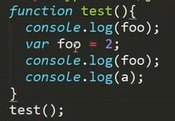


test() 打印出来 0，5，0

new test() 打印出 0，undefined，0

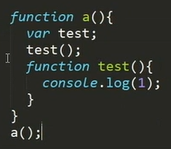
为什么是undefined 因为通过new实例化时并没有给this.a传入值

###### 第七题：



undefined 2 报错：a is not defined

###### 第八题：

 1

答案：1

解析：

1、建立AO空对象

​ 2、变量提升

​ 2.1形参声明提升 赋值undefined

​ 2.2实参声明提升、变量声明提升 赋值undefined

​ 2.3实参形参相统一

​ 3、函数声明提升 并把函数体赋值

###### 第九题：

[14\_三目运算\_对象克隆\_浅拷贝\_深拷贝-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/11829362135729518?_wv=2147487745)

1小时30分



结论：222 111 222 333

###### 第十题：

var getName = function(){

console.log(“222”)

}

function getName(){

console.log(“111”)

}

getName()

解答：

// 222

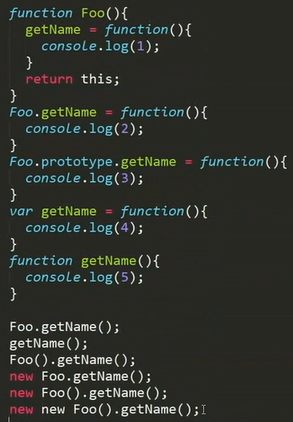
js开始预编译，GO中getName变量为undefined,然后函数声明提升，

getName替换为function getName(){} ,

然后实际运行时执行到函数表达式getName，

重新给getName赋值为function(){}

###### 第十一题：非常重要



2 4 1 2 3 3

解析：

[16\_数组方法\_类数组-前端进阶教程-腾讯课堂 (qq.com)](https://ke.qq.com/course/329070/11829370725664110?_wv=2147487745)

9分钟

###### 第十二题：微信js基础的第8题

var a = {}

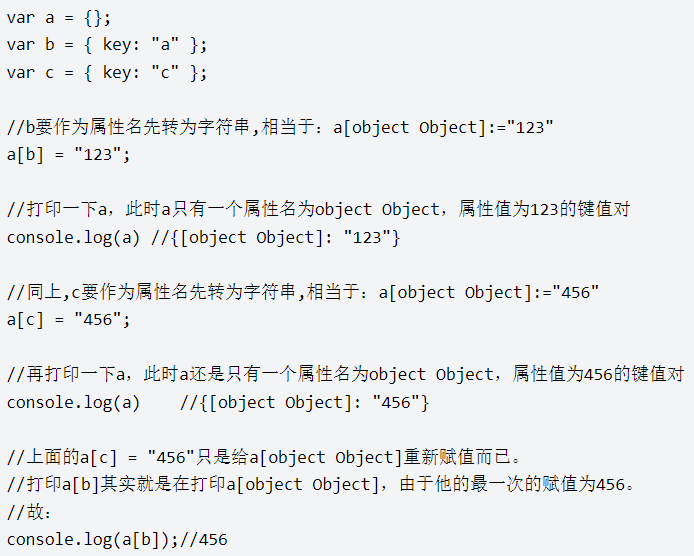
b = {key: “b”}

c = {key: “c”}

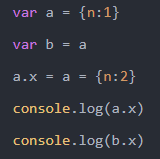
a[b] = 123

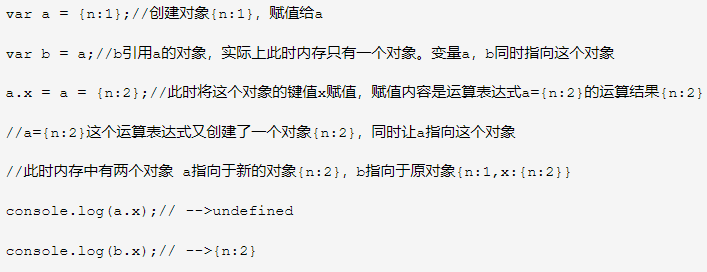
a[c] = 456

console.log(a[b])

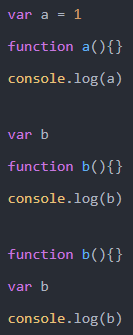


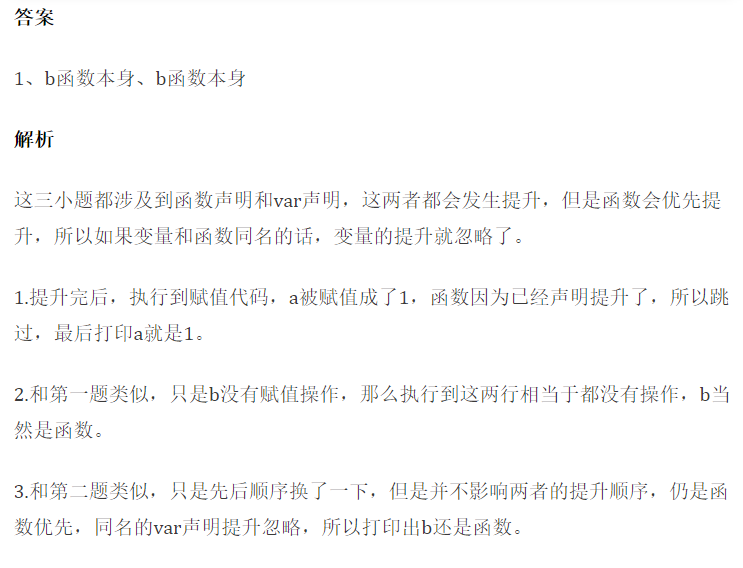
###### 第十三题：微信js基础的第4题





###### 第十四题：微信js基础第22题





第十五题：



严格模式下不允许对象属性名相同，但是不会报错，输出2