1、let 声明的变量，只有在自己的作用于有效；用let定义变量，不能重复声明该变量；

const 常量 声明时候必须赋值 声明数值不可修改，声明对象可修改

2、let a = 1 b = 2 [a,b] = [b,a] console.log(a,b) // 2,1 (交互变量)

3、babel-polyfill （处理es6一些方法的兼容库）

4、二进制数 用B 八进制数 用O

Number.isFinite() // 判断一个数是否有尽的

Number.isNaN() // 判断是否是一个数

Number.isInteger() // 判断一个数是不是整数

Math.trunc() // 一个数取整数部分

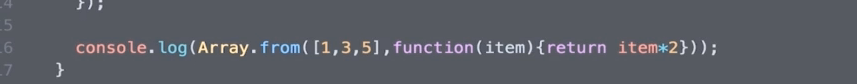
Math.sign() // 判断否是正数 （）

Math.cbrt() // 立方根

数组API:

5、 Array.of() 把一组数据变量 转换成一组数据类型变量

Array.from() 把为数组转换成真正的数组



映射： 第二个参数是一个函数



Array.fill( a ) 一个参数 把数组中的每个数都替换成a ,

第二个参数是数组索引的起始位置，第三个参数是数组的截止位置

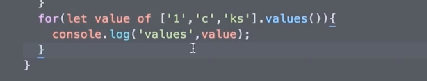
（也就是说讲数组中从起始位置开始替换，替换到截止位置）

1. [].keys()



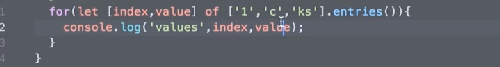
// 0 1 2

1. [].values



// 1 c ks

1. [].entries()



// 0 1 1 c 2 ks

1. [].copyWithin()



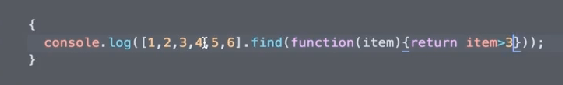
// [4,2,3,4,5]

第一个参数是起始位置替换（索引）

第二个参数是读取位置（索引）

第三个参数是结束位置（这段区间 从开始到结束的区间）

1. [].find()



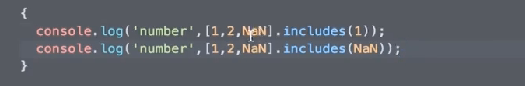
[].find的查找只找出第一个符合条件的成员

1. [].findIndex()



返回的是下标(索引)

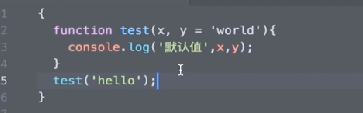
12、[].includes()



看数组中是否有该元素，并且能找到NaN

函数 ：

13、参数默认值

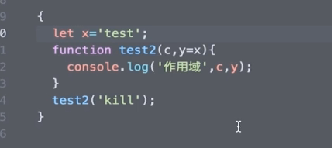


解决参数的默认值的问题... 也可以给y 传参（在y没有传参的时候就默认选择默认值）

1. 作用域问题

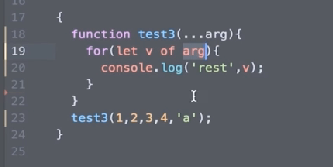


// kill kill



// kill test

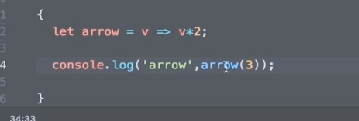
1. 参数问题



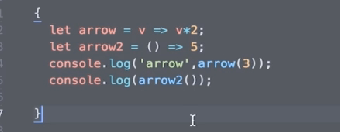
// rest 1 rest 2 rest 3 rest 4 rest a

（res后面不能有其他的参数了，否则会报错）

1. 箭头函数

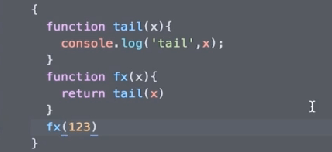


// arrow 6



// 5

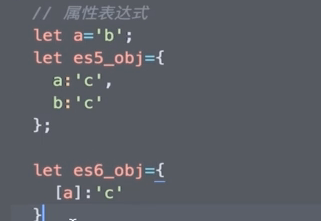
1. 伪调用



// tail 123

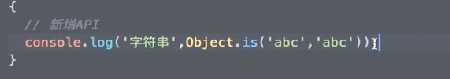
对象：

1. es6的属性表达式



// es6 中[a] 就是 b 因为上面let a = ‘b’ 所以a 的值就是b

19、新增API object.is()



// true true

判断两个值是否相等 （和 ‘===’一样）



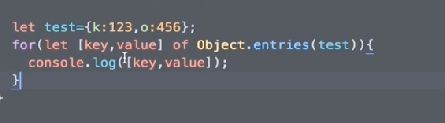
// false false

1. Object.assign()拷贝



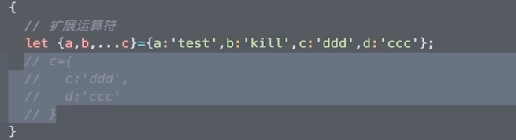
// 浅拷贝 两个对象合并成一个对象了 （浅拷贝）

1. object.entries()



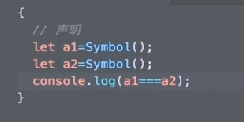
// [k:123] [o:456]

1. 扩展运算符（这种形式不要用）



Symbol: 他就是这种数据类型提供独一无二的值

22、Symbol

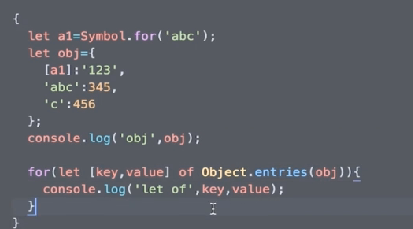


// false



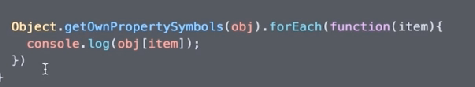
// true

// ‘a3’ 相当于一个key



// 取不到[a1]的值

1. object.getOwnPropertySymbols()



// 123

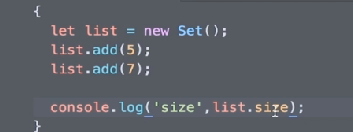
// 能取到obj里面的所有Symbol属性值，得到的是一个数组

1. Reflect.ownkeys()



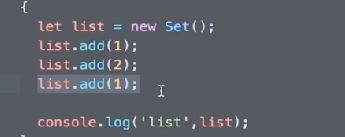
// 能取到obj里面的所有属性值（包括Symbol以及非Symbol），得到的是一个数组

1. Set



// 用add加元素

// size 获取长度



// 不能添加相同的值，不生效（应用：数组去重）



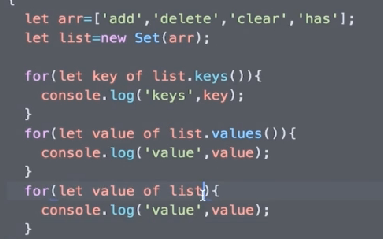
// has判断 arr中是否存在’add’元素 true



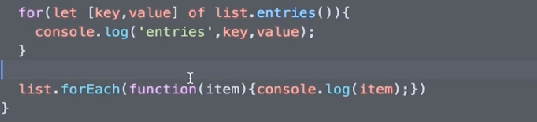
// delete 删除数组中’add’元素



// 清空list 数组



// 遍历 得到的是value



// 得到的依旧是value

1. WeakSet()

WeakSet 和Set 支持的数据不一样

WeakSet 只能支持对象

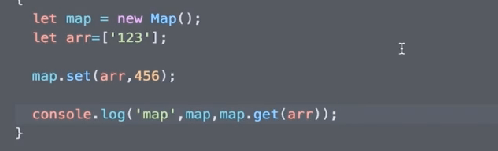
WeakSet 中的对象都是弱引用，他不会检测该对象在其他地方是否用过，

不会与垃圾回收机制挂钩上，不会去检测是否被垃圾回收掉了、

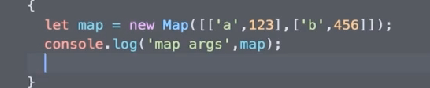
WeakSet 没有size属性，没有clear()，不能遍历

其他和set一样

1. Map



// 添加数据 用set 获取是用get



// 123 456



// 获取长度



// 删除



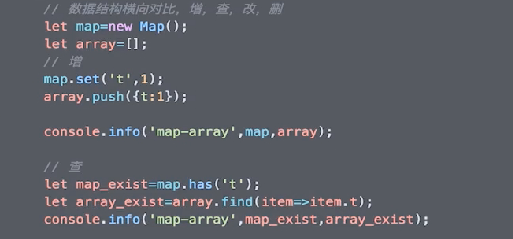
// 清空

其他和set一样

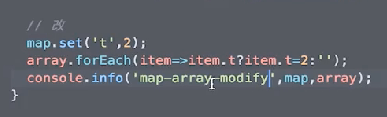
1. WeakMap

数据必须是对象，其他和WeakSet一样

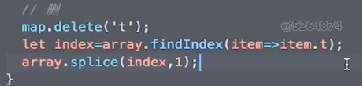
1. Map与Array 的对比（增、删、改、查）



// 查map 返回的是 true array返回的是该对象

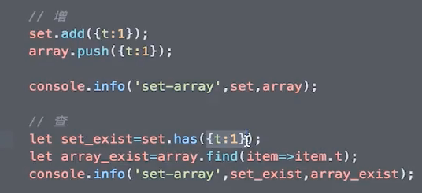


// 修改

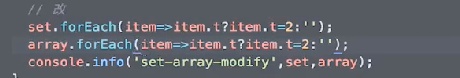


// 删除

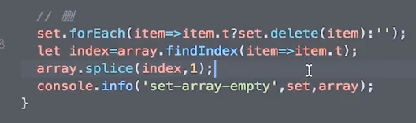
1. Set 与 Array 的对比（增、删、改、查）



// false 查 set.has（） 要查地址

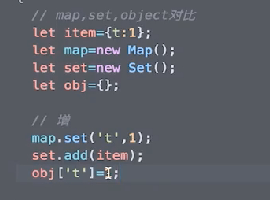


// 修改 不用 add 是因为 原素的值是一个新的对象，不会检测冲突的，他会把原来的 值修改，直接新增了一个 所以也要用foreach

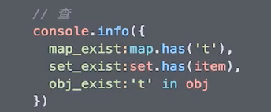


// 删

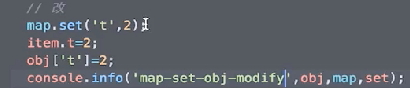
1. Map、Set与Object的对比（增、删、改、查）



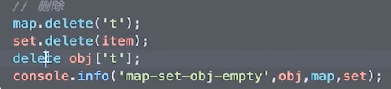
// 增



// 查



// 改

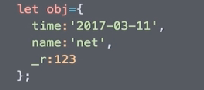


// 删除

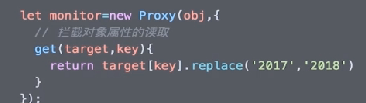
综上总结： 通过对比、在整个数据开发过程中涉及的数据结构，能使用map不使用数 组，尤其是复杂是数据结构，放弃数组，而使用map数据结构，如果对 数据结构要求存储的唯一性，考虑使用Set，优先使用map，如果对数据 的要求比较高，也就是说，保证每个数据的唯一性，要考虑使用set，放 弃使用object，数组也要放弃。

1. proxy和reflect

原始对象：



// 设置原始对象



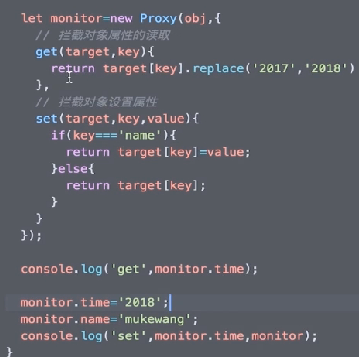
// 通过代理拦截，将 读取的属性中的2017 替换成2018



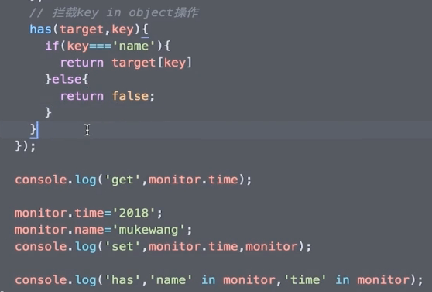
// 通过monitor 去读取

Proxy： 作用就是讲原始对象obj的通过proxy 代理，里面就可以实现代理的东西

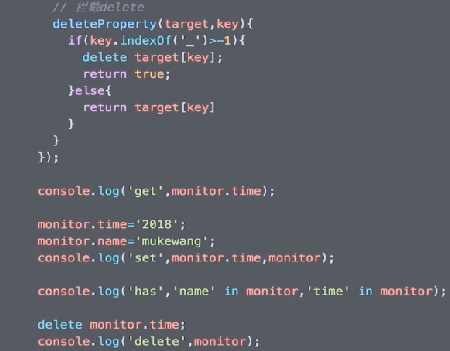
也就是方法，用户需要操作monitor对象，从而去传递给obj对象



// 代理的读（get）和写（set）



// true false(通过代理去欺骗用户)



// time 还在并没有删掉

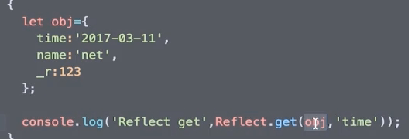


// \_r 被删掉了



// 保护了’time’的属性

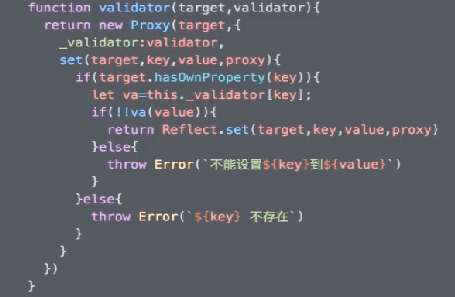
1. reflect

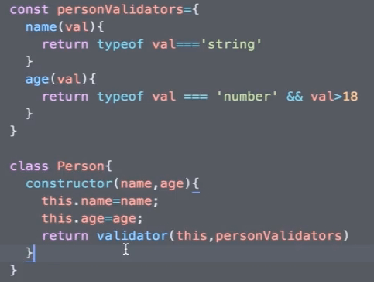


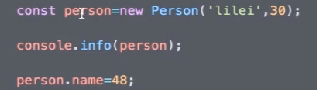
// reflect.get（）读取



// reflect.set（）设置

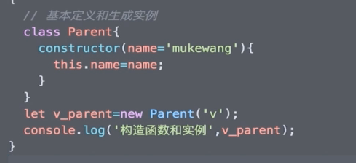


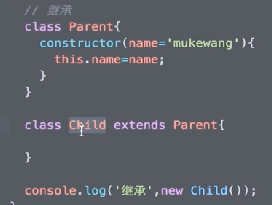




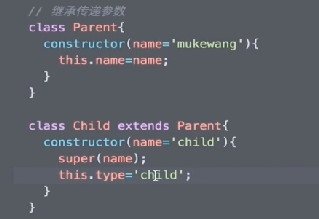
类：

33、类的基本定义和生成实例

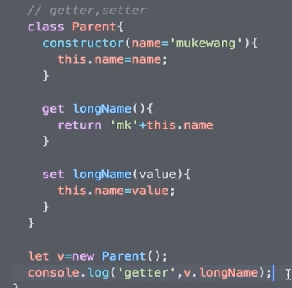




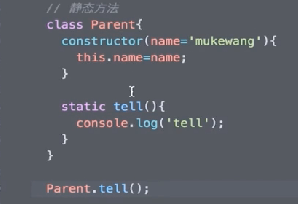
// 继承



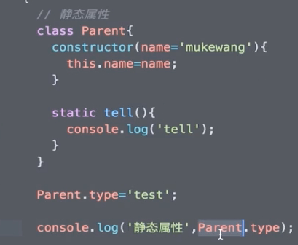
// super 要放在构造函数的第一行（不然会报错）



// getter , setter

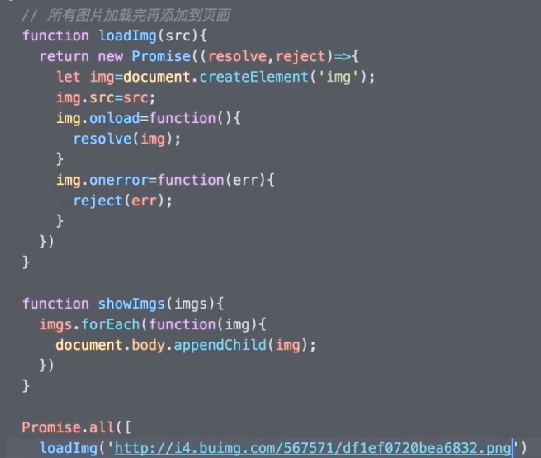


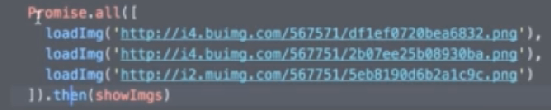
// static 静态方法



// 静态属性

1. Promise.all（）





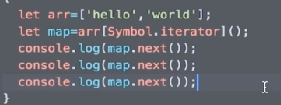
// promise.all() 返回的是promise实例 （3张图片加完完成后才返回promise实例）

1. Promise.race（） （与promise.all()用法一致）

在多个实例中有一个状态率先改变，那么最后race实例的话，也会跟着改变，

其他的都不在响应了

1. Iterator



// {value:’hello’ ,donc:false} {value:’world’ ,donc:false} {value:’undefined’ ,donc:true}

Iterator接口

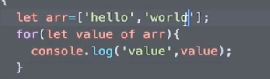


// 简单写接口



// 输出 1 3 2 7 9 8

1. For...of

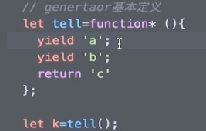


// hello world

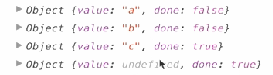
1. generator

（异步编程的解决方式，比promise更高级）

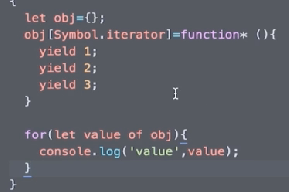
Generator内部包含多个步骤，它遇到每个步骤，也就是说每个步骤的标志就是 yield或者是return,遇到yield或者是return这一步就停止了，不会向下进行了， 进行下一步的时候需要调用next函数进入下一步，直到结束。



// generator 定义的基本方式

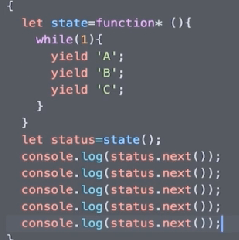


// 返回的结果

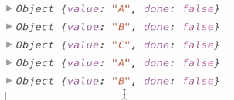


// value 1 value 2 value 3

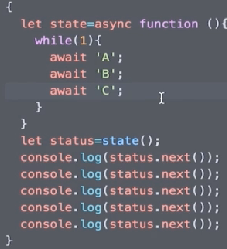
状态机



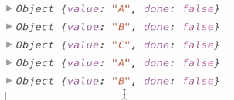
// next() 无限循环 获取state 的当前状态



// 输出结果



// 和generator 一样 是generator的语法糖



// 输出结果

抽奖的应用场景



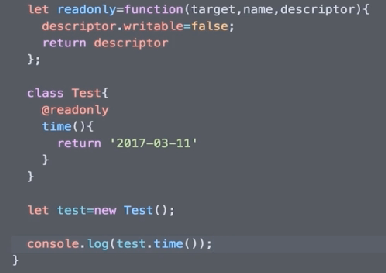
长轮询



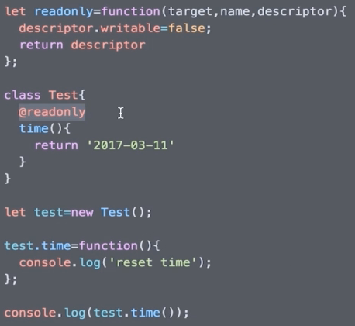
1. Decorator

修饰器是一个函数，用来修改类的行为

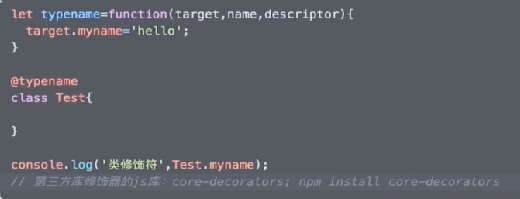
（修饰器是一个函数，修改行为，修改类的行为）



// 2017-03-11

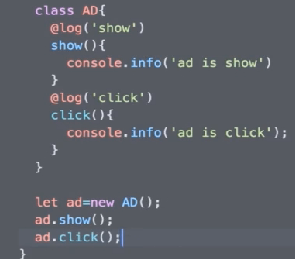


// 加了修饰器以后 把类的本身进行修改 把time进行只读的修改、



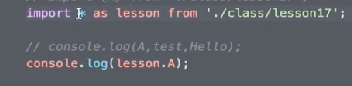
// 类修饰符 hello





1. 模块化





// \*代表所有 as是别名



// 输出