for循环存在父子作用域

for(){} 括号是父作用域 花括号是子作用域 每循环一次就有一个子作用域 let声明的变量 打印出来是每一个 不是最终的.

字符串解构

字符串的解构赋值 是一个变量对应一个字符 若有剩则不管.

判断字符串以什么开头：此方法区分大小写  
\* startsWith()； 返回值是Boolean 类型  
\* endsWith() ； 以什么结尾

接受两个参数

第一个参数，要在此字符串开头搜索的字符，第二个参数是指定从字符串开始的位置，默认从零开始

此方法区分大小写；endsWith()方法和startsWith()方法的语法都是一样的，不过endsWith()方法是从字符串的末尾开始查找。

promise

笔记看:

阮一峰:[http://es6.ruanyifeng.com/#docs/promise#Promise-race](http://es6.ruanyifeng.com/#docs/promise)

<https://segmentfault.com/a/1190000011652907>

用同步的写法 解决异步

Promise.all([$.ajax(),$.ajax()]).then(function(){},function(){})

Promise有返回解决了就会调用then(),参数一function表成功调用 参二function表失败调用

all要求所有都成功 全成功调用then中参一 有一个失败调用参二

参一可接受一数组 数组里为成功时返回的数据 此时可用解构赋值

注意Promise首字符大写

Promise构造函数身上自带几个方法

1. Promise.resolve(传实参);

作用就是new一个promise对象 并且此对象是成功的resolve的状态 参数是 给成功时执行的函数传参

对象.then()执行时 会直接执行成功的函数 并且形参会接到上面的

传实参 传过来的值

2. Promise.reject() 方法同上 这是失败的状态

此两个方法可以快速创建出一个成功或失败的promise对象

3.Promise.all([p1,p2,p3])

Promise.all()方法用于将多个 Promise 实例，包装成一个新的 Promise 实例。

把对个promise对象打包扔到数组里 再将数组作all参数所以all方法后可以继续调用then then接收的参数 是一个数组 数组里装的每个成功传过来的参数 即返回的数据

针对返回的数组 一般用解构

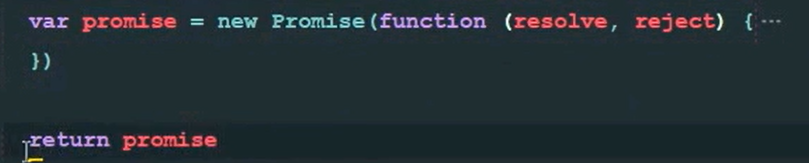
注意,all里面的每一个对象必须都是成功的状态 有一个不成功 就会报错

4.Promise.race()

race是竞速的意思

获取返回最快的那个 不论成功失败

异步

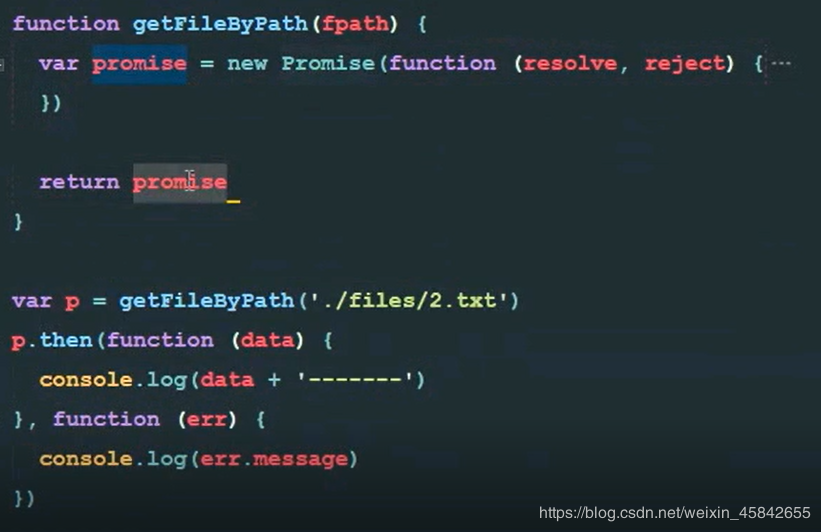


这里表示生成一个实例,并且可以立即拿到实例,但同时这也是一个异步操作,形参function会直接丢进异步队列里面,具体怎样做并不关心.主程序一瞬间拿到实例,此时异步操作都没执行.

如果是一个普通的异步操作则是发起异步操作,里面怎么运行则不关心.

**总结,生成实例且可以立即得到实例,function属于异步并不关心.**

Promise执行顺序



当new Promise的时候立即拿到实例对象,并且把实例return给p,p拿到的时候里面function异步操作肯定没执行,因为主程序是一瞬间执行return的(异步队列属于子程序执行),此时可能异步操作还没开始呢.p拿到结果主程序又一瞬间往下执行,立即为Promise实例通过.then方法指定了成功和失败的回调函数,当.then被主程序执行完后,内部的两个形参resolve和reject才被具体化,就已经变成这么一个函数了,今后当文件读取成功后必然能调用到resolve和reject,因为是.then先执行之后它才读取文件的,因为这是一个异步操作.

generator生成器

普通函数是执行了就一直到底直到结束

generator也是函数 但是执行中间可以暂停

1. generator 有\*号 可以贴着function 也可以贴着函数名 或者直接放function和函数名中间 但不能两个同时贴着 这样generator函数就有了 命名随意 有\*就是generator函数

generator函数不能写成箭头函数

**function**\* *f*() {  
  
}  
**function** \* *f*() {  
yield;  
}  
**function** \**f*() {  
  
}

2.f() 第一次执行函数 函数内部不会执行 而是返回一个gennerator对象 里面有next方法 此时再对象.next()就是执行

3.yield 放弃 表暂时放弃执行的权利 一会再执行

例图 对象.next()第一次执行yield前面的代码会执行

再下一次执行对象.next()就是执行yield后面的 是踹一脚走一脚 走到有yield的地方停 前面执行过的就不会再执行 只执行后面的

走走停停原理其实就是大函数内部分成了多个小函数 执行的都是小函数

yield

yield可以传参 可以返回

传参是next方法传 此时yield就是对应参数(拿变量接收) 注意第一个next传参永远都是废的 因为到yield时就停止 第二个next执行时才接收 如图输出a为5



返回

模块化

语法

1.先scrpit标签的type为module

2.js文件导出模块

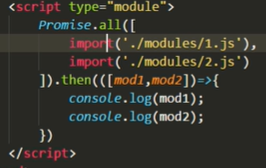
export

3.在script中引入模块

import空格再引号 引号内写模块js文件路径(相当于引入并执行该文件)

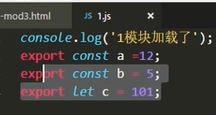
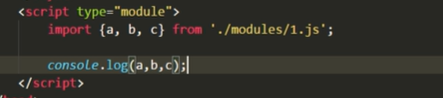
import特点:

1. 模块化 需要放到服务器环境 例localhost
2. import可为绝对路径或相对路径 当前文件夹必须写./ 否则报错 以后都写好习惯
3. import模块无论引入多少次都只会导入一次
4. import有提升,会自动提升到顶部,先使用.相当于函数声明提升
5. 导出的模块,若里面有更改内容,外面也会实时更改
6. 静态import语法不能写在循环或if之类里会报错,动态import可以写
7. 重点import() 可以动态引入,返回值是promide对象 所以可以then all之类方法,通常结合promise使用

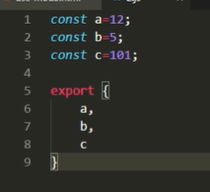
 接收数组 故可形参时就解构

1. 在es6的模块里自动默认采用严格模式 例变量要定义再使用 而为定义变量属全局

以上操作和引入文件没区别 那怎么用模块呢?

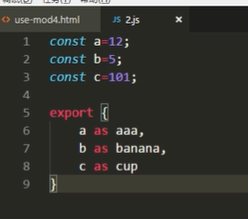
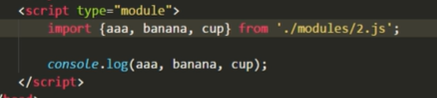


这里{}不能少 若单a就{a} 若多个就{a,b,c}



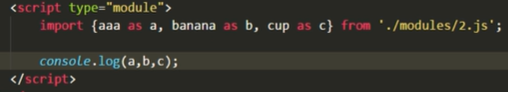
还可以这么导出 类似对象简洁化 但是不能写成a:a这种会报错

导入方法不变 像赋值解构

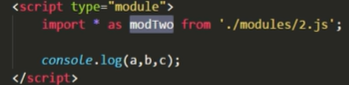


导出可以用别名 若别名则导入里也要用别名

导入也可用别名

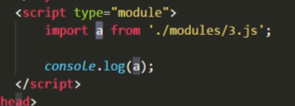


导入可用\* 必须拿别名接 打印出对象对应每个变量名和对应值,default的名default

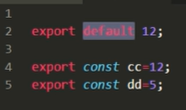
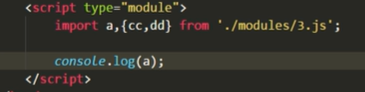


只要是export导出的 导入都需要{}

如果是export default 就不需要{}

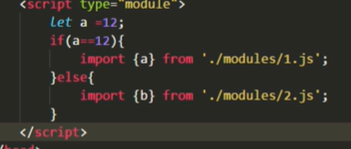


还有这种 逗号隔开 继续{}



js文件里可以直接导入模块 此时不需要type

上述写法是静态的,只能一次性引了就用,若按下面这种语法会报错

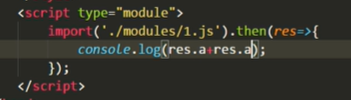


若想根据条件动态引入模块 只能如下写法:

res是对象 里面装所有模块

引入需时间 有些代码是引入成功后再用 就需动态引入模块 成功了就执行then成功

参数res是所有的 含default的



下面这种相当于配动态路径



类(class)与继承

类

注意:class类 不提升不预解析

创建对象实例或类（虽然没有类的概念，但是可以把用于创建新对象的对象看作类），例[构造函数](https://www.baidu.com/s?wd=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)



class Person{}定义类 typeof Person 是函数 知道怎么写就行不较真

类里面有一个真正的构造函数constructor

当new时 Person内的constructor相当于构造函数 会自动执行,传参即传入这里面

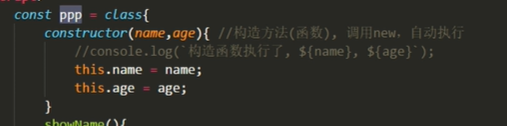
注意这里person后直接是花括号,且这不是json, constructor后不能加逗号 里面的函数写法是按对象简洁语法写 没有function



给对象加属性

给person身上加方法

类似表达式的命名写法 不推荐



es6里属性名可以是表达式 在json里也可以, (重要 记住)

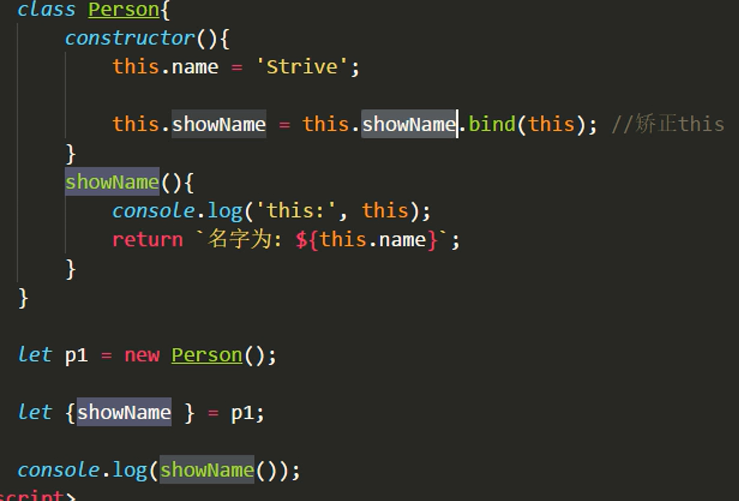
如何使对象属性名为变量?

属性名若直接aaa相当于字符串aaa

若为[aaa]里面的aaa就是变量

[“strive”]相当于就是字符串strive

这里的[]作用相当于表达式 内可写其他加减乘除



自调用时,若不bind则 this是undefined

bind后就是对象

get set 存值函数和取值函数



类的表达式

类可用表达式形式表示,类似函数表达式.class后的关键字如果不省略此关键字只在class类的内部有定义,指代当前类,若用不上可省略

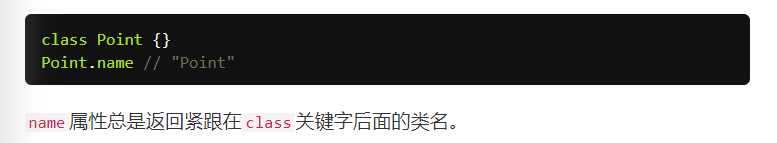


获取所属类的类名

**注意点**

1. 类和模块的内部，默认就是严格模式**,**

如果将方法提取单独使用，this会指向该方法运行时所在的环境（由于 class 内部是严格模式，所以 this 实际指向的是undefined）

1. **不存在提升.**
2. 
3. class内部只能写静态属性/方法 实例属性/方法 其他都不能写 class内部还是用原来都构造函数实现的 现在这样相当于就是个语法糖

**实例属性新写法**

**实例属性除了写在constructor内 还可写在类的最顶层 此时不写this. 其他一样**

****

**静态方法**

**方法前**加上static关键字 则该方法不可被实例调用 而是直接通过类来调用，这就称为“静态方法”。

**静态属性**

静态属性指的是 Class 本身的属性, Class 内部默认没有静态属性,所以constructor是方法 不是属性 不能用点调用

若想静态属性,写法是在实例属性的前面，加上static关键字。(不写实例属性前面也可以 但最好在前面写)

例:static myStaticProp = 42;

Symbol()

1.首字母大写

2.不能new

定义:let syml = Symbol("a");

3.有时需给某属性加特殊标志 表私有属性 唯一的 而 Symbol()返回的值就是唯一的

所以一般用它做一个key 定义一些唯一的私有的

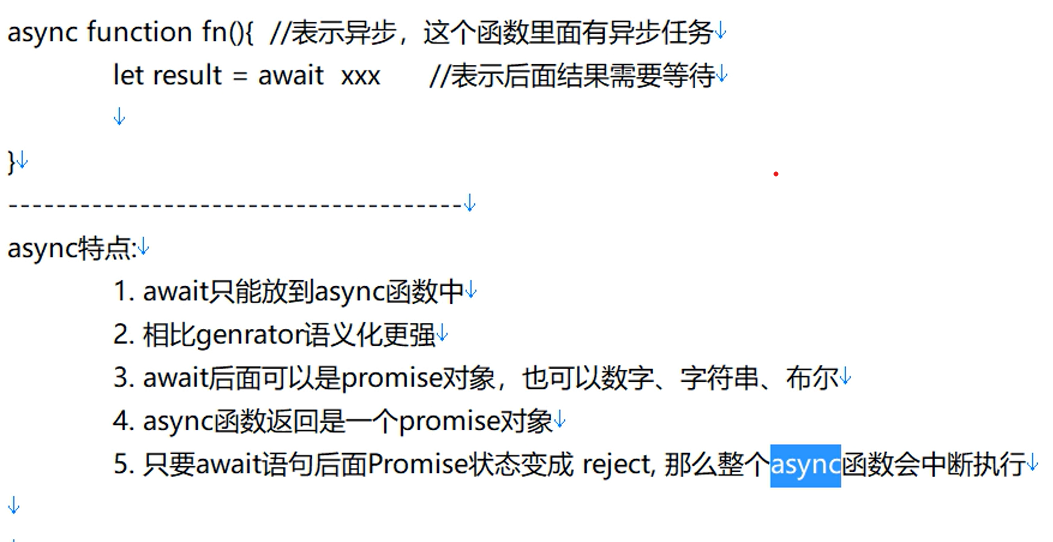
如果用它作key 用for in循环 出不来

1. 是一个单独的数据类型,基本类型.用typeof检测出数据类型为symbol (全小写)
2. *Symbol定义的属性 不可被循环遍历出来  
   \* 只有通过Object.getOwnPropertySymbols(obj)取获取symbol类型的属性*

*// 返回一个symbol类型的key值数组* ***console***.log(***Object***.getOwnPropertySymbols(***obj***));  
*// 将对象的所有key值存入一个数组里面 不含symbol* ***console***.log(***Object***.getOwnPropertyNames(***obj***));

async

.



await 后面可以是任何,都会转成pormise对象

只有成功才会走下一步 失败会中断.

return的将作为下一个then的参数



success

出现问题了

undefined

await 后的返回值 单独拿变量接收 只可在函数内用 在函数外打印 哪怕是全局变量 也是undefined

**若在对象里写async**

{

a:function a(){}

}

可以简写成

{

a(){}

}

加了async可以写为

{

async a(){}

}

**arr.some()**

**定义和用法**

some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。

some() 方法会依次执行数组的每个元素：

* 如果有一个元素满足条件，则表达式返回*true* , 剩余的元素不会再执行检测。
* 如果没有满足条件的元素，则返回false。

**注意：** some() 不会对空数组进行检测。

**注意：** some() 不会改变原始数组。



# 字符串的padStart方法使用

padStart是在前面补字符串内容,一参是补后的总长度(原长度+补的),二参是要补的字符串,若补后长度超过规定的长度,则截断到规定的长度,若满足总长度则不补

str.padStart(补后的总长度,要补的字符串)

案例: 可用于进行补0操作,例时间为9秒时 使其为09秒

let str = '10'  
console.log(str.padStart(2,0)) //10 不进行补  
​  
let str1 = '9'  
console.log(str1.padStart(2,0)) //09 补0了

# 字符串的padEnd方法使用

和padStart类似 唯一的区别是从后面补字符串内容