# ECMAScript6.0

ES6是JS的一种新语法,他体现在方便简化方面,本质还是JS,就和node和common.js一样,只是一种规范。前面JS部分写过一点,这里写个比较全面的。

## Let和const

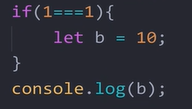
这俩是JS的ES6新规范里替代var的东西。var这玩意存在很大的问题,最严重的就是变量提升,他会导致作用域的问题,甚至会污染全局变量,let和const就是解决这个问题的。

特点:

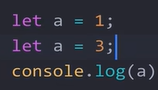
1. 没有变量提升

这种代码,在控制台中是会报错的,他没有变量提升。

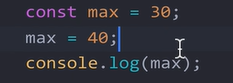
1. 块级作用域

可以理解为作用域只在{}里面,所以前面的代码是不可能在控制台读取到b的,因为我们的log是在作用域外的。解决了var全局污染问题。

1. 不能重复声明

我们如果这样写他会报一个重复声明的错,解决了var变量容易重复声明的问题。

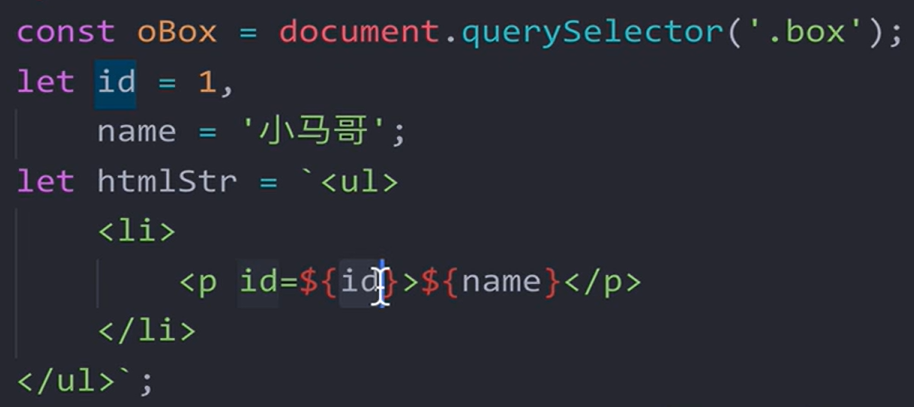
1. const声明的是常量

一旦被声明无法被修改。

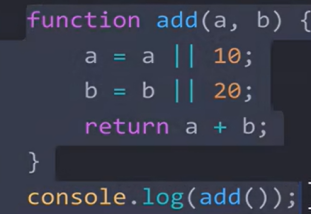
**所以使用上来说,默认情况都用const,当明确变量一定会发生改变时用let。**

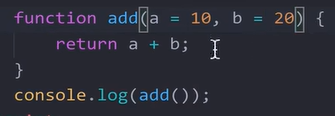
## 模板字符串

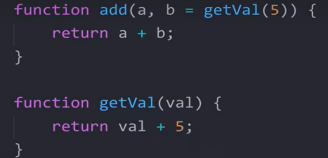
这个东西语法上就是``里面写字符串,如果要用到变量,用${变量名}。他主要解决的就是字符串拼接这一麻烦的问题,他还有一个好处就是换行不用加/n。这在我们之前的shader着色器里有体现，他方便了我们写着色器。



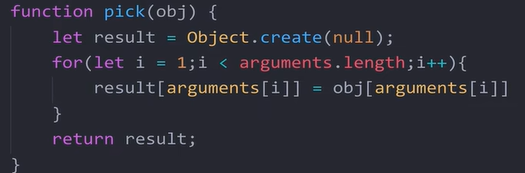
## 函数默认值

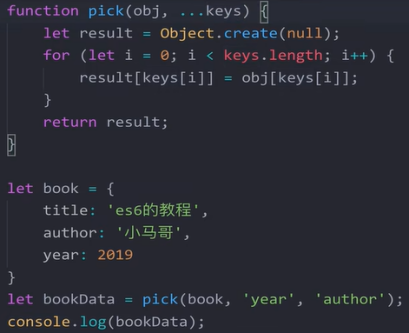
ES5中我们构建函数要想要默认值要在函数里这么写,非常反人类。

ES6就简化了这个问题。

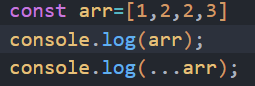
默认表达式也可以是个函数。

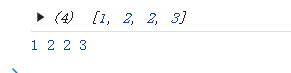
## 函数剩余参数

我们ES5里不确定有几个参数的时候会用arguments来把剩下的参数变成一个伪数组。我们要控制i的位置非常麻烦。

ES6里用...参数名的方法替代了这个arguments方法。

## 函数扩展运算符

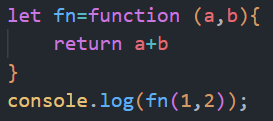


扩展运算符这玩意很简单,就是把一个数组拆成散装的,和上面的剩余参数用法正好是反过来的。



这种方法用来做数组合并是非常合适的。

## 箭头函数

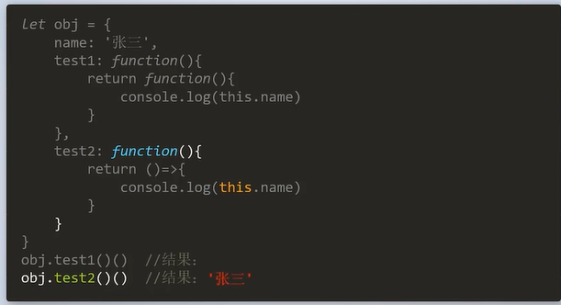
ES5的写法。

ES6箭头函数写法。箭头函数还可以简写：

箭头函数通常用来做回调用。

**箭头函数的this指向**

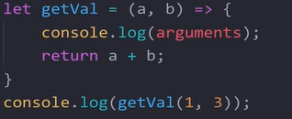
箭头函数没有this,this就指向上一级,可以很好解决this指向问题,比如下面代码。

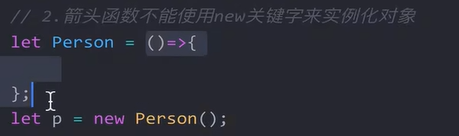


**test1的调用是return的function函数,这个玩意是没有this的指向的。所以是undefined。**

**test2的调用是return的箭头函数，他根本没有this直接用上面function的指向,function是obj调用的,所以用obj.name。**

**注意**

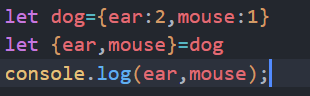
**箭头函数没有arguments。**前面这个代码是会报错的。

函数本身并不像匿名函数一样是个对象,他本身只是一个函数的语法糖,没有办法new,上面代码 依然会报错。**不能当构造函数。**

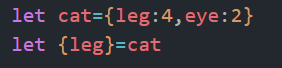
## 解构赋值

就是直接把对象或者数组中的内容拿出来进行一个赋值。实际上是对赋值运算符的一个扩展。

完全解构：

这就是我们完全解构了一个对象。

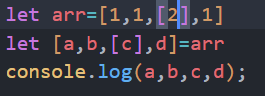
部分解构:

对对象的不完全解构。

也可以用剩余运算符解构。

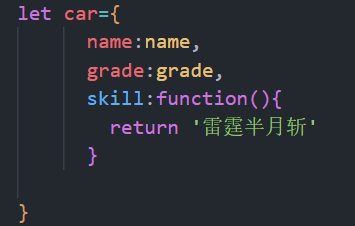
默认值:

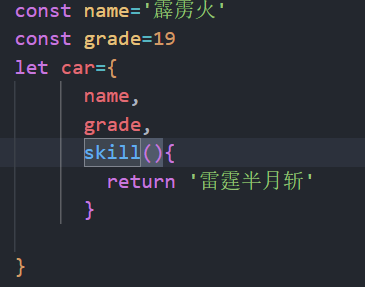
对数组解构：

.**注意:解构时我们左右的格式要是一样的。**

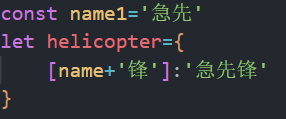
## 扩展的对象的功能

1. 对象中的键和值一样时,可以省略键,代码如下:

这是我们ES5的代码,可能会比较麻烦。

这是我们ES6的代码,精简了很多。**不过要是键和值是一样的才能用。**

1. 对象的键可以采用字符串拼接的方法,代码如下:

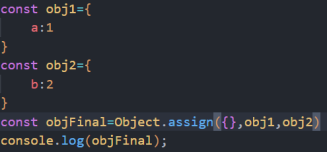
我们可以看到我们对象的键可以用拼接字符串的方法拼起来。

1. is()方法解决===(严格相等的漏洞)

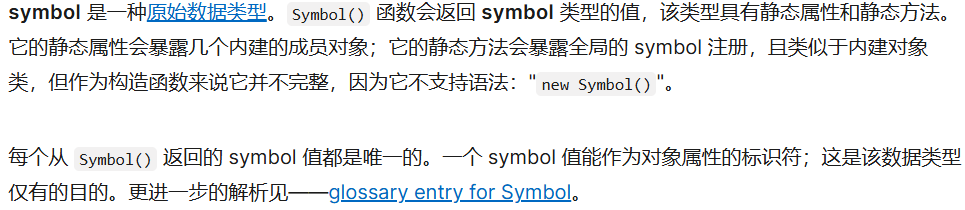
我们的严格相等会出现判断漏洞。ES6的is就解决了这个问题。



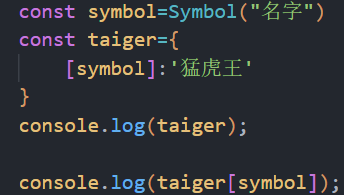
1. assign方法进行对象的合并操作



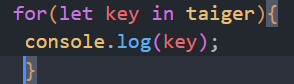
## Symbol类型



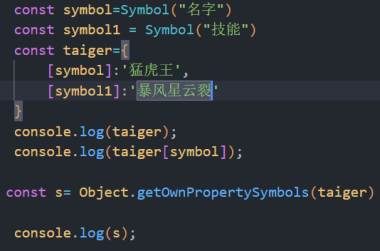
MDN上面是这么说的,他规定了一个独一无二的值。在我们的认知里面独一无二的值好像就是id。所以他的作用也就是作为对象属性的标识符,也就是对象属性的键。

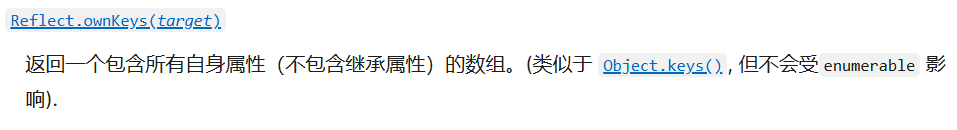
上面是symbol作为对象的键的一个基本代码。

**注意:**

1. **取symbol关联的变量一定要用[变量名],这在上面有过体现。**
2. **像这种传统的for,in解构,是无法得到他的key的。**

**我们用symbol作为对象关联的名字,得到对象的内容就要用Object.getOwnPropertySymbols(obj)这种方法才能拿到键。**

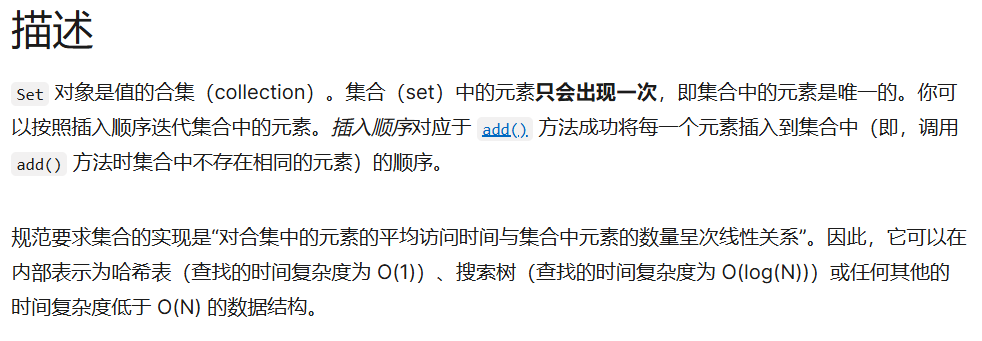
还可以用这个Reflect反射方法拿到键值。



。

Symbol这玩意只在极少数的特定的场景中会用到,了解一下得了。

## Set集合

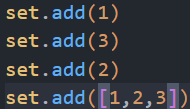


翻译过来就是无重复值的有序列表。

1. 创建集合



1. 往集合里添加值



1. 从集合里删除值



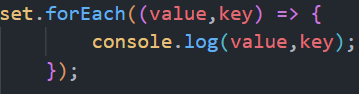
4.检查集合里是否有某一元素



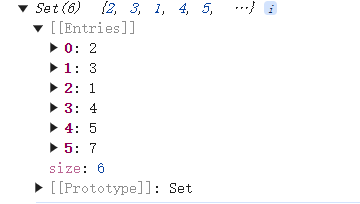
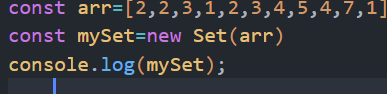
5.集合的大小



6.键和值都是一个东西



1. 可以用来数组去重

重复的都给你去掉。

1. set中的对象无法被释放

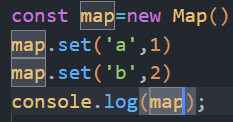
通常用weakSet替代set解决无法释放问题,了解一下即可。

## Map类型

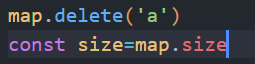


Map和set一样他也是保存数据的列表,区别在于Map存的是键值对。

1.设置map



2.map的大小和删除map



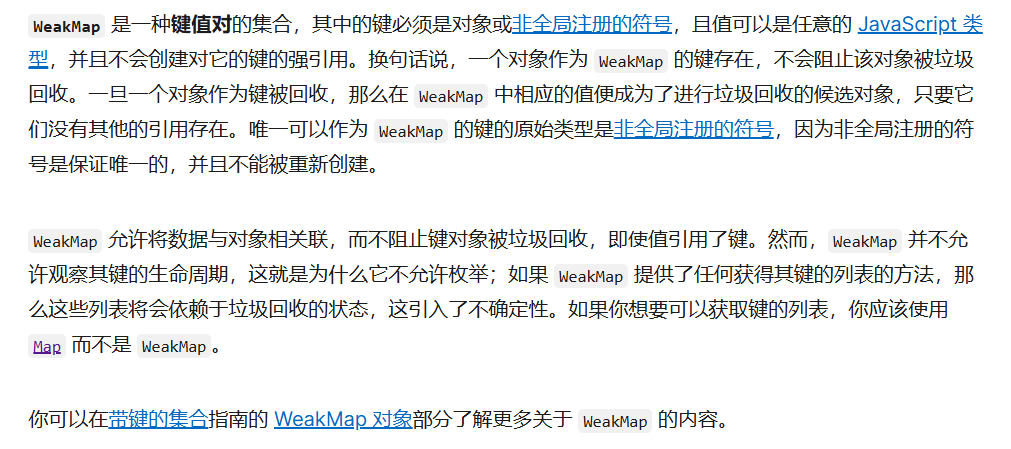
3.清空表



4.键值是任意的



5.weakMap



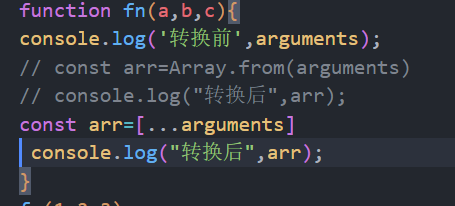
## 数组扩展方法

### from

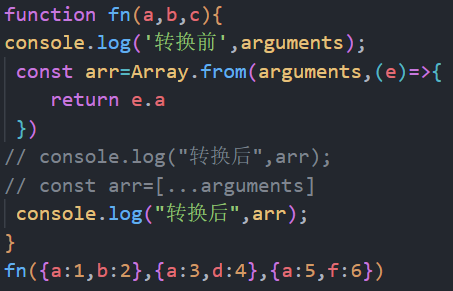
from能将伪数组转换为真正的数组。下面代码就是一个转换过程,把arguments这个伪数组转换成了真正的数组。



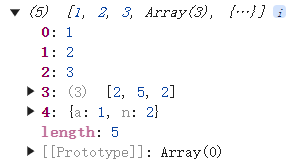
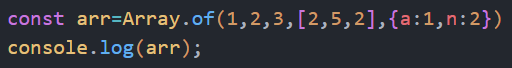
其实扩展运算符也能转换伪数组为真正的数组。下面的执行结果是和上面一样的。



from第二个参数表示的是回调。回调函数可以把他的内容解析出来。实用性不错。

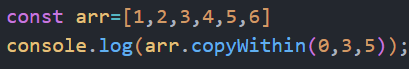


### of



### copywithin



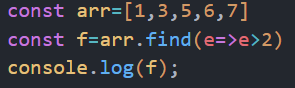


大概意思就是从0个索引开始,被3到5索引的数据覆盖。包左不包右,左闭右开。

这个方法不常用,了解一下就行了。

### find



比如上面的代码,f的值就是第一个满足大于2条件的3。

### findindex

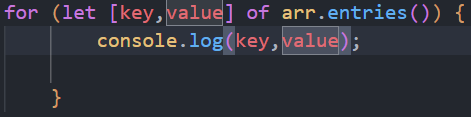


与find非常类似,只不过find返回的是数据,他返回的是索引,上面代码如果用findindex来写的话返回值就是3的索引1了。

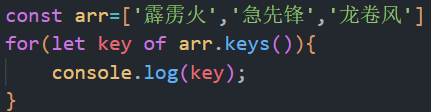


### 6.entries(),keys(),values()

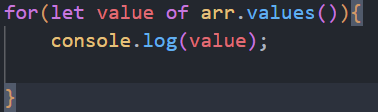










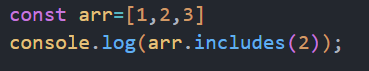


这三个其实返回的是一个迭代器,我们for...of遍历的也是迭代器,下面会写迭代器。

### includes（）

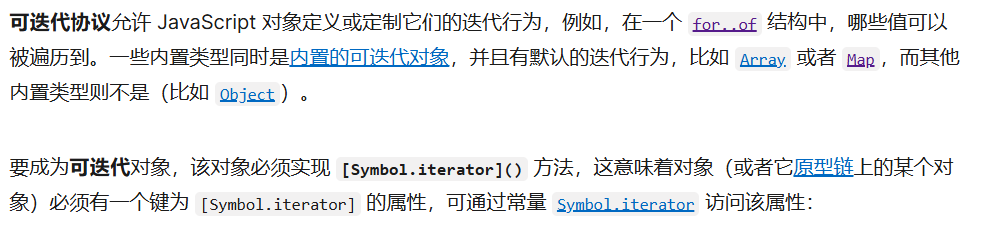


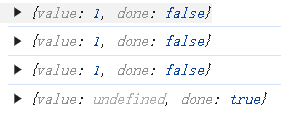
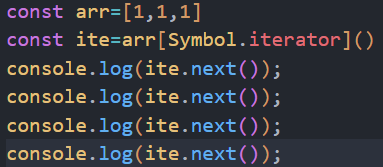
ES5里判断数组中是否有某个元素是用上面的代码判断。



这样就可以直接判断数组里有没有某个数据了。

## 迭代器（Iterater）

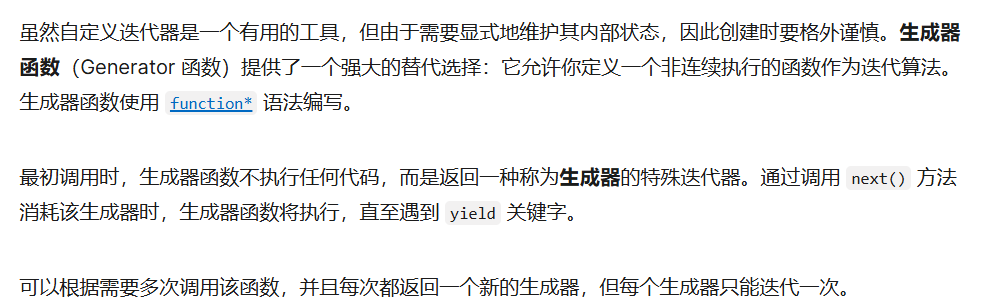




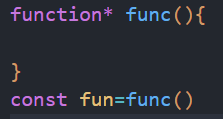
迭代器的使用就是上面那样用[Symbol.iterator]()方法生成迭代器,每次使用next一下。

**迭代器可以理解为一个遍历数据的指针。**

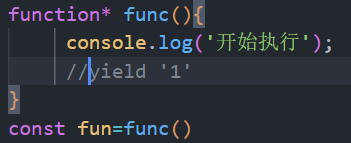
## 生成器（Generator）



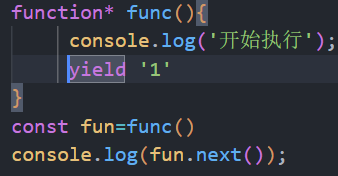
生成器就是一个函数,不过他不是一次就执行完了的,他里面遇到yield会停止,需要next()才能继续执行。专业一点就是他会把函数挂起。

这样我们就实现了一个生成器的创建并且调用了生成器。

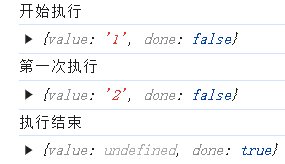
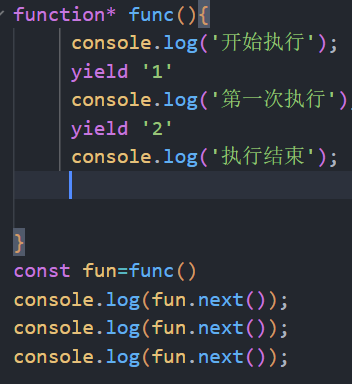
在没有yield的情况下,他和正常函数没啥区别。下面那样写代码它会像正常函数一样执行。



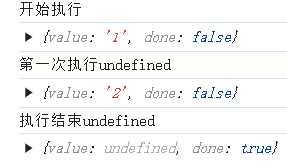
但加了yield之后就不一样了,函数不会执行了,想要执行必须要用next()。



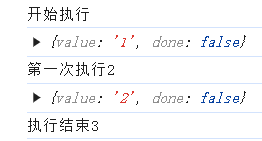
一次next()只能跳过一次yield就会停止,所以我们每有一个yield就要有一个next().



遇到yield就会停止,所以我们如果赋值的话。他是没用的。



我们要在next的时候赋值。**因为遇到yield时next的任务就完成了,赋值不会进行,所以赋值要到下一个next时才有效。**

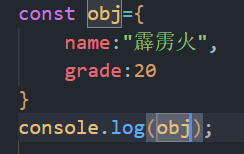


如果没有return，最后一次done为true时是undefined。如果带了return执行的时候value就是return的值。



## 生成器的使用

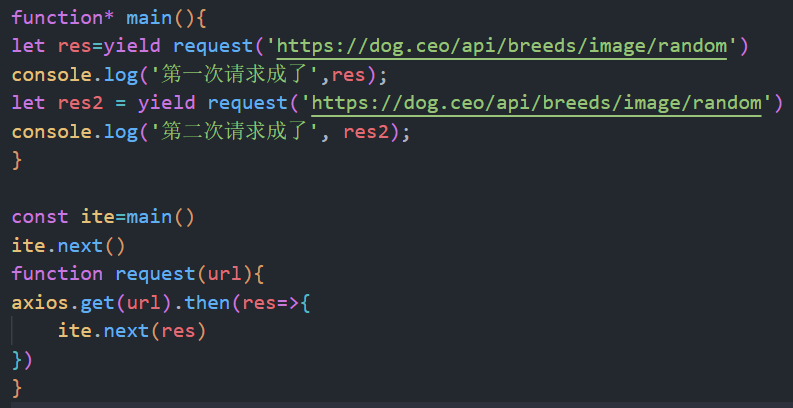
### 给没有迭代器的对象创建一个迭代器。



我们创建一个普通对象很显然是没有迭代器的,生成器在这个时候就有用了。

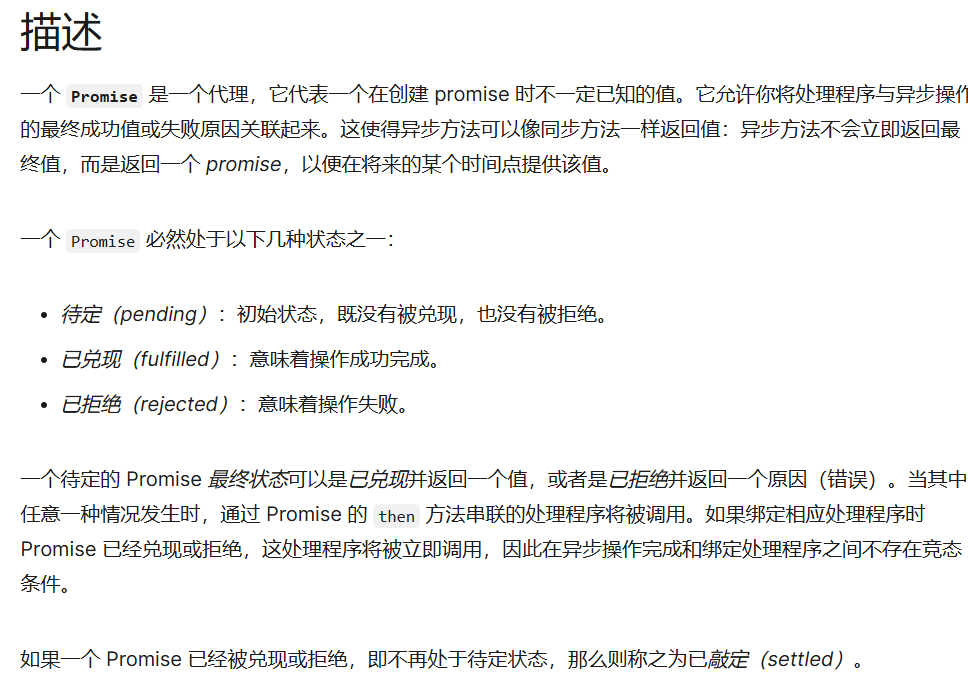


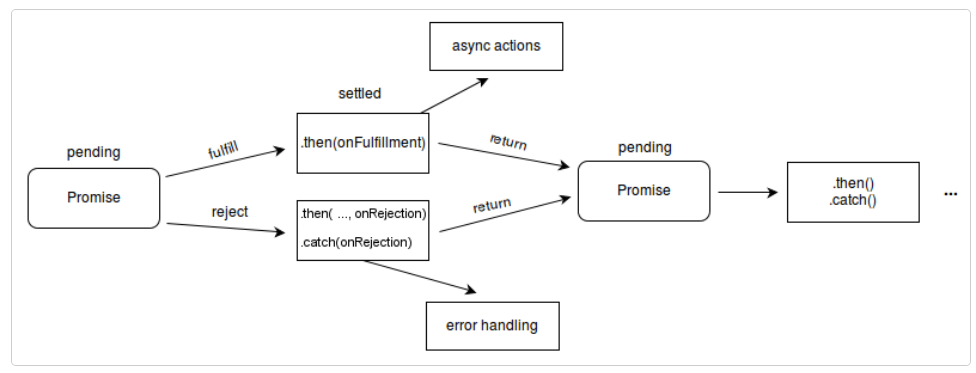
### 异步代码同步化,解决回调函数地狱（类似于async,await）



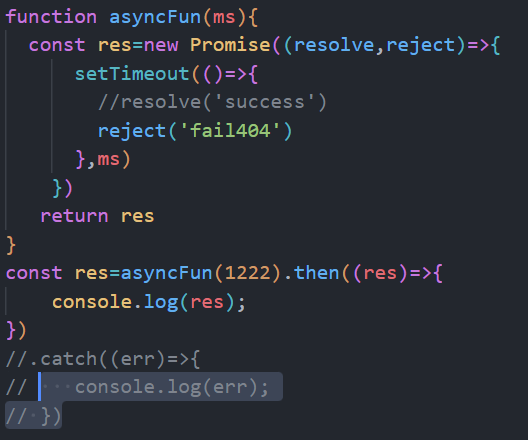
把请求封装成函数,把函数的调用放在生成器函数里,请求完成他才会执行ite.next(res)。把res的值赋给let变量。第一个不完成,第二个别想动,类似于await。

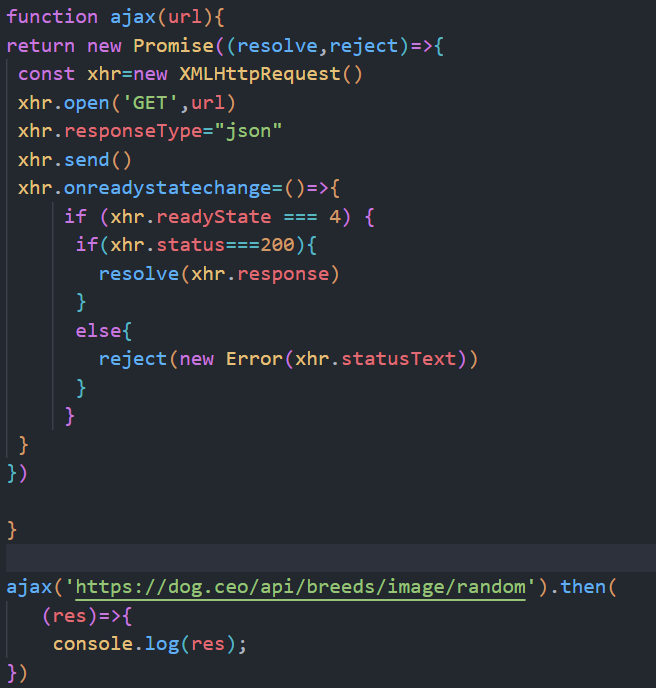
## Promise





上面是MDN给出的一个解释,语言过于专业,显得晦涩了点。简单点说就是promise是帮助实现异步的。解决回调函数地狱问题。

上面代码就是promise的一个具体使用。在异步执行成功后把异步执行的结果放入resolve里,之后就可以执行.then链式调用拿到结果。错的话就放在reject里。.catch链式调用。**他的主要作用依然是异步代码同步化。**

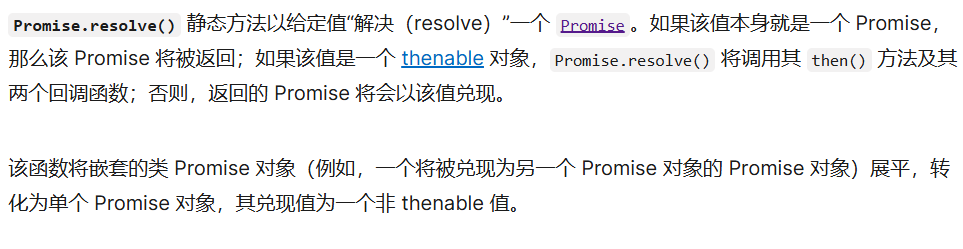


这里用promise配合XML封装一个axios来把promise的好处具象化。就是通过promise的封装实现一个异步的同步写法。方便进行一个链式调用。

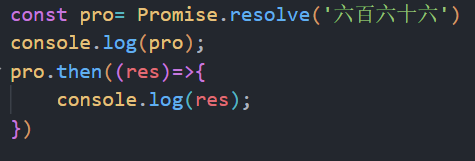
## Promise对象的方法

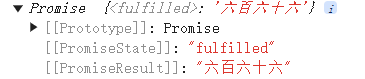
上面提到了Promise的.then和.catch方法,其实promise还有很多方法,像resolve,reject,all,finally,race等。

### Resolve

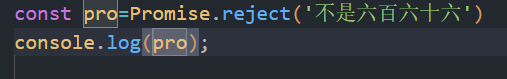


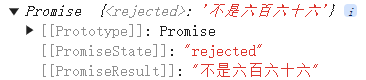
MDN给的专业解释是这样的,依然是晦涩难懂。翻译过来就一个意思,把对象包装成promise对象。

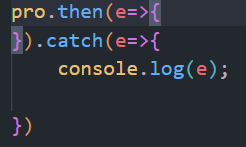




### Reject

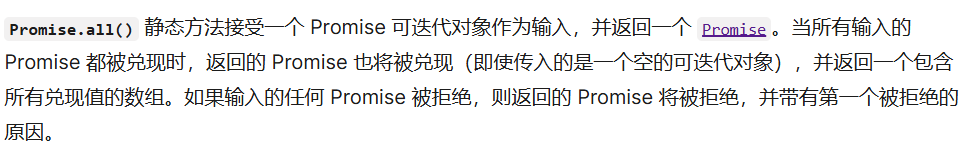




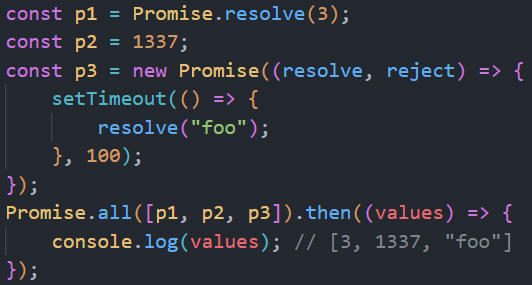


Reject与resolve很像,差别就是状态的不同resolve是fullfilled。Reject是rejected。

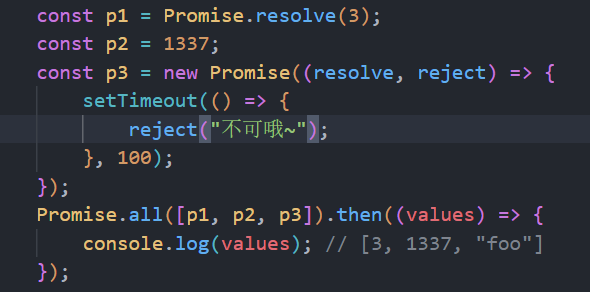
### all



这个all是并行执行的意思,我们在all里输入多个promise对象,所有对象都成功的时候才能.then,不然只能在.catch里面见了。



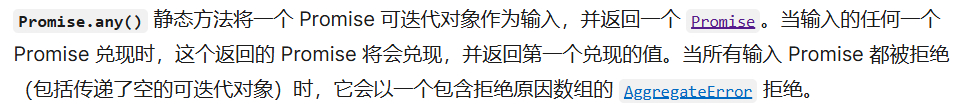
就像上面这样,一次.then就可以拿到所有promise的结果。



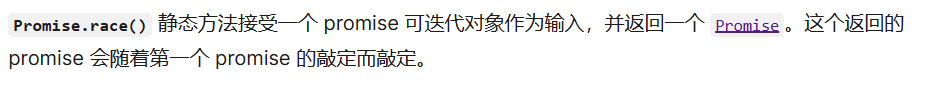
减少了.then的调用次数,但是缺点也很明显,一处错,所有的都错。只要这几个promise里有一个reject拒绝了,就别想拿到一个结果。

### Any

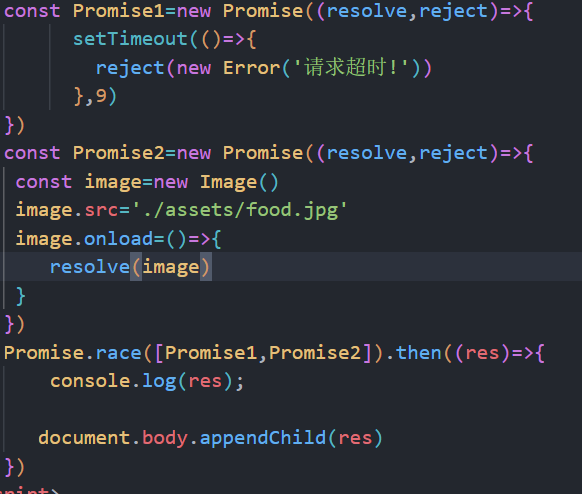
与all是相反的。

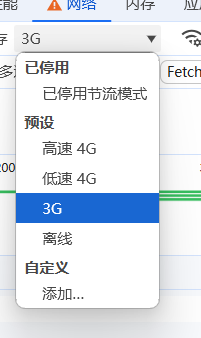


### 5.race



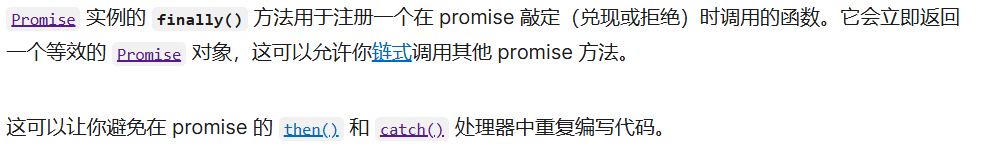
形象一点的理解这个就是:两个Promise赛跑的。他会以第一个执行完的结果为执行完成的结果，**也就是说如果我们加载图片这个请求resolve的时间要比请求超时的长他就会执行请求超时这个reject的promise。**



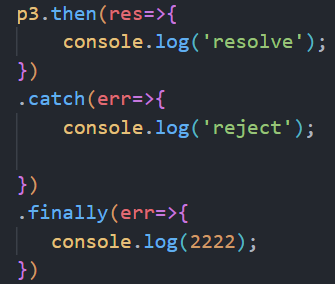
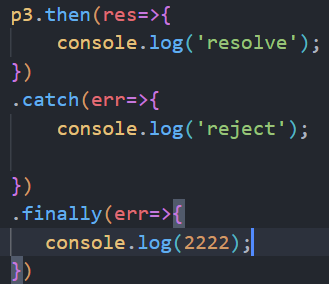


网速慢点的话请求图片就会比我们设置的超时时间长,就执行超时的reject。

### 6.Finally



很明显了,这就是一个独立于.then和.catch意外的方法,不管前面是敲定还是没敲定,resolve还是reject他都会执行。



从上面两段代码（第一个是敲定失败,第二个是成功敲定）可以看出来finally是一定会执行的。