es6的理解2

**目录：**

迭代器和迭代协议

for...in...

for...of...

应用：...的扩展

Generator生成器函数

next()

yield

Promise构造函数，Promise对象

异步的执行历史：

回调函数

Generator——原理，利用迭代规则衍射的函数

promise

await

回调函数

promise的形式

状态pending、resolved、rejected

动态方法：then()，catch()，finally()

静态方法

Promise.all()

Promise.race()

Promise.resolve()

Promise.reject()

es7中新增的await

应用：promise和await两种方法事项图片异步加载

**主体：**

**迭代器和迭代协议**

for..in..根据对象的key来进行遍历

for..of..根据对象的value来进行遍历，但是不一定！，要看对象原型上面有没有迭代器[Symbol.iterator]。

**迭代器**（函数）：对象被迭代的时候调用的处理函数。一个对象是否能够被迭代，取决于该对象是否拥有迭代器。

**有迭代器的对象**：Array。Map。String。set。。。。。

**迭代协议** - 函数的做事的规则:参考

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Iteration\_protocols

1. 迭代器函数必须返回一个对象，函数是没有参数的

2. 返回的对象中必须包含一个 next 方法

3. next方法必须返回一个对象

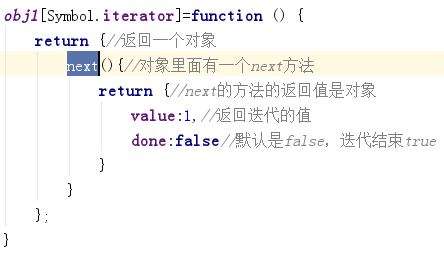
4. next方法返回的对象包含以下两个属性

value ：当前这次循环被迭代出来的值

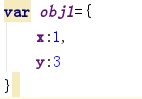
done : 表示当前迭代是否已经完成，如果为true，表示完成了，终止循环，同时value会被忽略，默认值是false

实现迭代协议的**代码**如下：

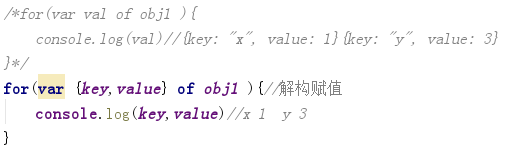
clipboard.png



实例：







应用：...的扩展

**Generator生成器函数**

Generator Function 生成器函数，另外一种新的函数

形式：function后面或者函数名的前面加上一个\*，或函数名前加\*

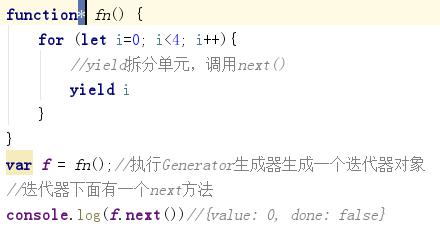
原理：

生成器函数调用以后执行不是函数体本身，而是会执行内部生成器生成一个新的迭代器对象，然后调用迭代器对象下的next方法来实现执行函数调用

配合yield语句，yield放在任意一条语句之前，那么yield会把整个函数拆分成多个执行单元，然后通过迭代器的next方法来调用每一个yield

next()

yield



异步的执行历史：

回调函数

Generator

promise

await

回调函数

promise的形式

状态pending、resolved、promise

静态方法：then()，catch()，finally()

**promise的理解：**

1. promise是构造函数，是一个函数构造器，接收一个函数做为参数，这个函数就是要执行的异步操作

-new Promise(fn(resolve,reject))

因为promise的状态不能访问，所以在参数函数中传参。

在Promise构造函数的函数参数中会接收到两个传入的参数，这两个参数都是一个函数;

第一个参数：是一个函数，调用该函数可以把Promise对象的状态改成resolved状态;

第二个参数：是一个函数，调用该函数可以把Promise对象的状态改成rejected状态;

2. 在Promise对象中有一个属性[[**PromiseStatus**]]：

该属性是一个内部属性，外部不能直接访问，他有三个值，表示了当前Promise对象的完成状态。

-pending：未完成

-resolved：已完成

-rejected：已失败

3. **then**方法是Promise中的异步操作完成以后，后续执行的方法

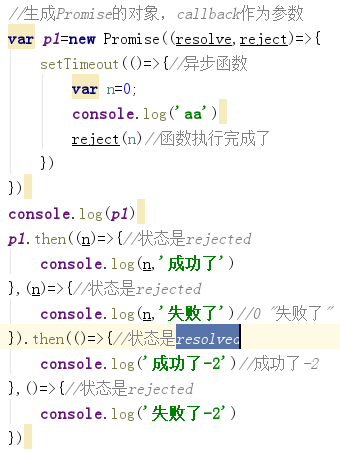
-then（fn，fn）

then方法的执行需要Promise中的异步操作完成以后再去调用，但是默认情下,Promise并不知道异步动作什么时候完成，需要我们在异步操作中去通知Promise事情完成得怎么样了；

当Promise的状态编程**resolved**，那么这个Promise对象的then方法中的**第一个函数**才会被执行；

当Promise的状态编程**rejected**，那么这个Promise对象的then方法中的**第二个函数**才会被执行

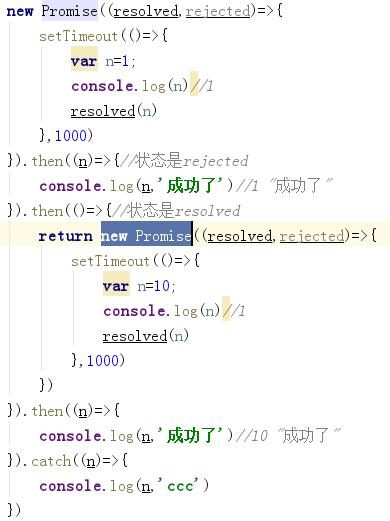
该状态不能直接去修改，我们可以通过Promise为我们提供的几个方法来改变状态



解释：一个then之后，产生的还是p1的Promise对象，他的状态是默认是resolved。

如果想实现then内部也是异步函数，等这个异步执行完再执行后面的，我们需要在then内部再返回出一个promise

实现的代码：



**obj.finally(fn)**

无论是成功还是失败都会执行的

如果一件事情无论是成功还是失败都要需要去做的，那么这个工作就可以放在finally中去完成，比如无论是登录成功还是登录失败都要记录日志

**Obj.catch(fn)**

Promise错误处理

通过**then**处理失败：

精确的捕获每一次失败的时候，同时失败不影响后续的任务

通过**catch**捕获处理失败：

统一捕获错误，当只要有错误就终止后续执行

静态方法：

**Promise.race()**

与all一样，可以处理多个任务，但是race只要有一个任务完成了就会执行后续的操作，而all所有的任务都完成的情况下才执行后续的任务;

**Promise.all([promise1,promise2])**

当有多个异步任务要处理，同时后续的任务需要这多个任务全部完成以后再处理的情况下，我们使用 Promise.all;

all方法中去判断所有任务的状态，如果都完成了，并且都是resolved状态，就会执行all后面的then，只要有一个任务rejected了，那么就会执行then的第二个参数或者all后面的catch;

如果成功了，那么then接收到的参数是所有任务成功完成后的值，通过一个数组来保存;

**Promise.resolve()**

**Promise.reject()**

默认情况下我们得到的一个是pending状态的Promise对象，我们可以通过Promise.resolve()或Promise.reject()这个方法就可以直接得到一个resolved或rejected状态的Promise对象;

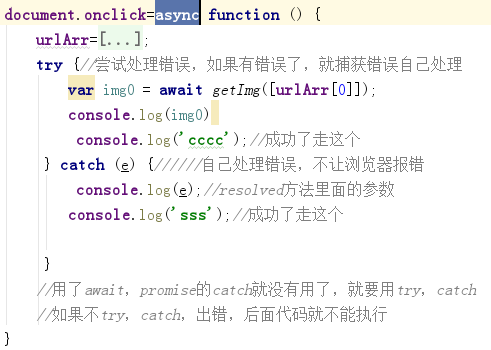


**es7中新增的await——省略了then**

可以在一个返回了Promise对象的函数前面加上 await 关键字，这样就不需要使用then了，但是await必须在一个异步的函数中进行调用。

因为我们用await代替了then，所以promise下面的catch方面就没法用了，这种情况下，可以用try...catch...来做。

try尝试代码，如果报错，走catch代码，并将错误信息通过参数传入catch代码中。



**实现代码原理：**

应用：promise和await两种方法事项图片异步加载

