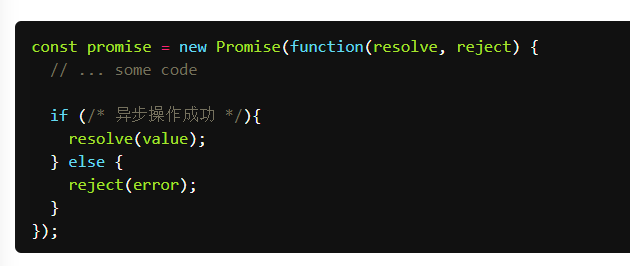
# Promise对象

Promise 是异步编程的一种解决方案



一个最简单的Promise对象结构

1.

resolve函数的作用是，将Promise对象的状态从“未完成”变为“成功”（即从 pending 变为 resolved），在异步操作成功时调用，并将异步操作的结果，作为参数传递出去；

reject函数的作用是，将Promise对象的状态从“未完成”变为“失败”（即从 pending 变为 rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去。

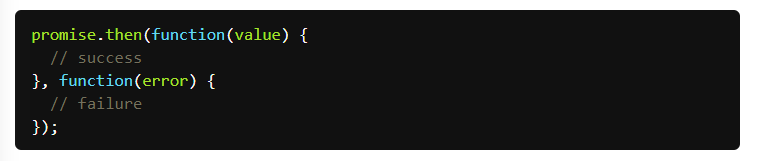
也就是一个是成功的操作，一个是失败的操作

Resolve和Reject都只是返回参数的工具，并没有函数覆盖一说！promise对象自带的，只是返回的路径不一样！！！

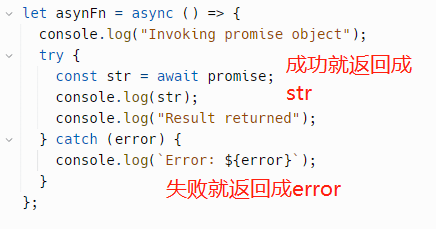
2.

Promise实例生成以后，可以用then方法分别指定resolved状态和rejected状态的回调函数。

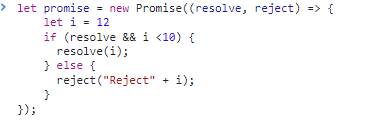
这里resolve就把返回的值当第一个函数value参数，reject返回的值是第二个函数error参数。



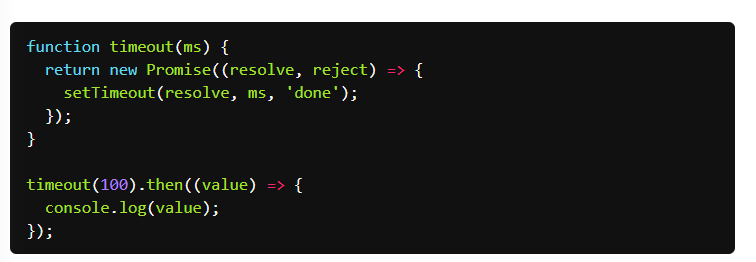
甚至不需要回调函数：



这个promise长这样：



## 一个简单的例子

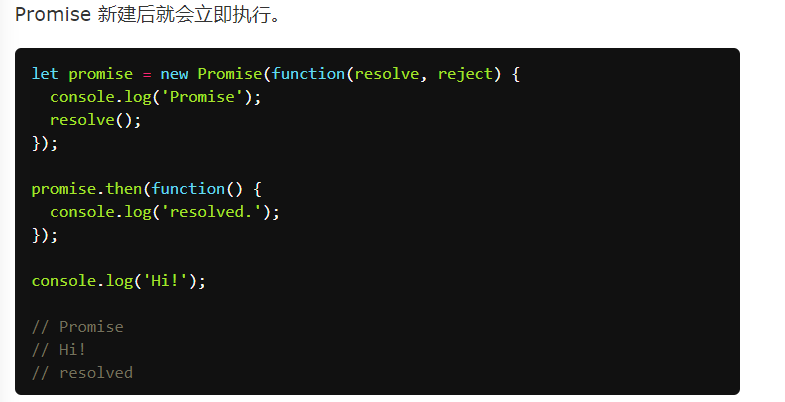


正常的的话一般都会执行setTimeout

上面代码中，timeout方法返回一个Promise实例，表示一段时间以后才会发生的结果。过了指定的时间（ms参数）以后，Promise实例的状态变为resolved，就会触发then方法绑定的回调函数。

讲的是一要变成resolved状态之后才会进到then里面去（他这个then后面的箭头函数是默认的resolved之后的回调，后面还可以有一个rejected的函数放在后面。）

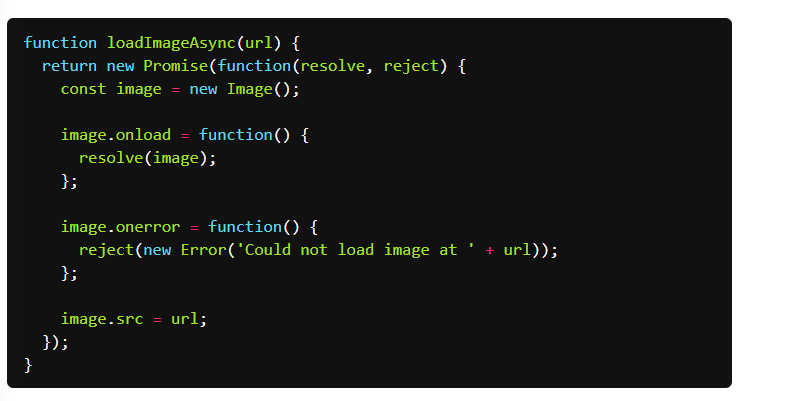
## 另一个例子



上面代码中，Promise 新建后立即执行，所以首先输出的是Promise。然后，then方法指定的回调函数，将在当前脚本所有同步任务执行完才会执行（因为promise是异步的），所以resolved最后输出。

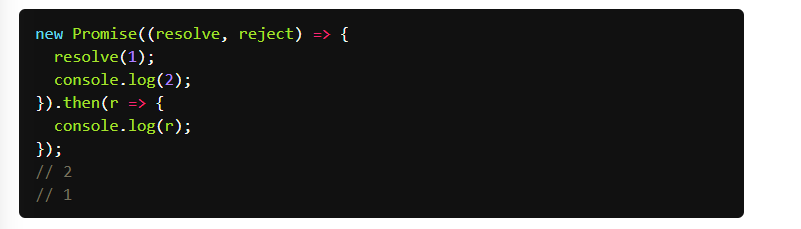
看到那个resolved()没,那就是resolved()

## 加大力度



上面代码中，使用Promise包装了一个图片加载的异步操作。如果加载成功，就调

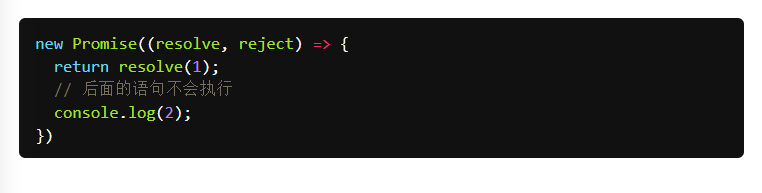
用resolve方法，否则就调用reject方法。



上面代码中，调用resolve(1)以后，后面的console.log(2)还是会执行，并且

会首先打印出来。这是因为立即 resolved 的 Promise 是在本轮事件循环的末尾

执行，总是晚于本轮循环的同步任务。



一般来说，调用resolve或reject以后，Promise 的使命就完成了，后继操作

应该放到then方法里面，而不应该直接写在resolve或reject的后面。所以，

最好在它们前面加上return语句，这样就不会有意外。

就是说在resolve()哪里加上return就够了

## Then的链式调用

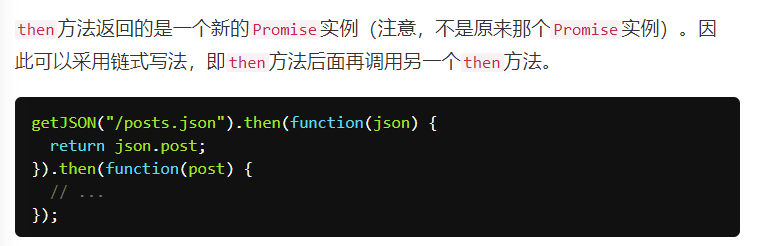
（then里面要返回的是一个新的promise对象才可以进行链式调用，是基于这个新的promise对象的状态来进行调用的是后面then的那个方法的）

Promise 实例具有then方法，也就是说，then方法是定义在原型对象

Promise.prototype上的。它的作用是为 Promise 实例添加状态改变时的回调

函数。前面说过，then方法的第一个参数是resolved状态的回调函数，第二个

参数（可选）是rejected状态的回调函数。



这种链式调用的前提是前一个promise对象必须返回的是一个promise对象才可以调用。（待考证，先这么理解）

### 例子



上面代码中，第一个then方法指定的回调函数，返回的是另一个Promise对象。这时，第二个then方法指定的回调函数，就会等待这个新的Promise对象状态发生变化。如果变为resolved，就调用funcA，如果状态变为rejected，就调用funcB。

可以看见一切都是由上一个promise对象的状态（resolved还是rejected）来决定下一个then进行那个函数，

所以看出，一定要理解什么是promise对象的状态！resolved还是rejected是很重要的两种状态。



箭头函数写法，更容易理解。

## catch()函数的用法定义

Promise.prototype.catch方法是.then(null, rejection)

或.then(undefined, rejection)的别名，用于指定发生错误时的回调函数。

也就是rejected的回调函数。但是功能比单纯的rejected强大一点

### 例子



上面代码中，getJSON方法返回一个 Promise 对象，如果该对象状态变为

resolved，则会调用then方法指定的回调函数；如果异步操作抛出错误，状态就

会变为rejected，就会调用catch方法指定的回调函数，处理这个错误。另外，

then方法指定的回调函数，如果运行中抛出错误，也会被catch方法捕获。

就是说，只要前面的getJSON变为rejected状态可以被捕获错误，其resolved

进行中的回调函数出现错误了也会捕获，相当于.catch()之前的内容全部捕获错误

这个还是比较牛皮。

### 不要自己定义rejected

官方吐槽：可以发现reject方法的作用，等同于抛出错误。



所以官方认定说还是不要用rejected自己定义，直接用catch()的冒泡性质好得多，可以捕获全部的异常。比如上面那个bad里面，rejected是捕获不到then里面resolved()回调函数中的东西的，而catch()可以

### 不同try/catch

跟传统的try/catch代码块不同的是，如果没有使用catch方法指定错误处理的回调函数，Promise 对象抛出的错误不会传递到外层代码，即不会有任何反应。

Promise 内部的错误不会影响到 Promise 外部的代码，通俗的说法就是“Promise 会吃掉错误”。

程序还是可以正常运行，充分体现异步。

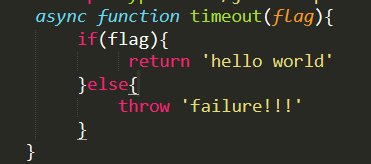
### Catch捕获Catch中的异常



Catch返回的也是一个promise对象，可以用catch来接受并报错。

# Async(异步)

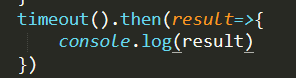
这是一个定义的标签



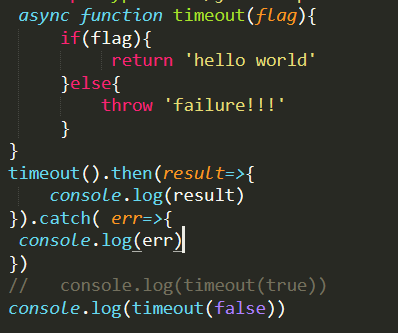
由他定义的函数和普通函数是一样的调用方法，

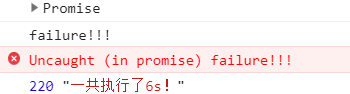
不同的地方就是这个是异步函数，不会影响整个项目同步进行

这种async定义的函数返回是一个promise对象，所以取到里面的东西要用then方法



也可以加上catch()来捕获异常，像这样：

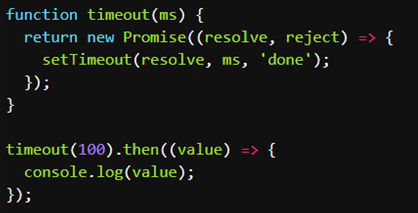




failure就被打印出来了。

## 用处:

可以理解为Promise对象的一个简写构造的方法，不用像原来一样写一堆构造之类的。



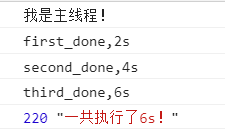
# Await(等待)

await顾名思义就是等待的意思，它后面可以放任何表达式，不过我们更多的是放一个返回promise 对象的表达式。注意await 关键字只能放到async 函数里面（绝对不希望主函数卡在哪儿对吧，所以await只能配合异步条件async用）所以前面要加一个“a”，表示的就是异步的意思。

## 例子



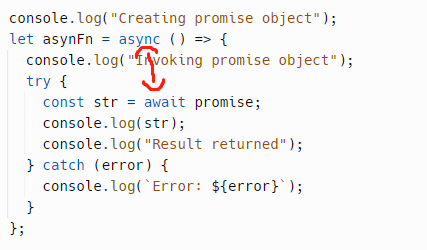
打印结果：



就这样一个例子，await会等待后面的表达函数执行完毕返回值后程序才会接着往下走，

Await之所以要放在async中就是因为他会暂停程序，为了不影响主线程的进展，要将他放

在asycn中。



结合使用

# 关于异步的理解

如果现在开始理解的话，异步的意思就是，在整个主线程上又开了一条分支线程，这条分支

线程就是异步操作，他的进展与主线程无关，主线程继续跑，分线程可以暂停，继续，无所

谓。“步”的意思，应该就是步调吧，指的是与主线程的步调。每个程序都有一个主线程，

而异步说的就是不同于主线程的步伐，应该可以这么理解。上面的asycn中使用await不影

响主线程的console.log(“我是主线程！”)的打印，就是这么体现的

那种类似Ajax，数据库查询的哪一类查询返回值操作，一般都是异步处理的