**Promise**

所谓Promise，简单说就是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）的结果。从语法上说，Promise 是一个对象，从它可以获取异步操作的消息。Promise 提供统一的 API，各种异步操作都可以用同样的方法进行处理。

1.对象的状态不受外界影响。Promise对象代表一个异步操作，有三种状态：Pending（进行中）、Resolved（已完成，又称 Fulfilled）和Rejected（已失败）。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思就是“承诺”，表示其他手段无法改变。

2.一旦状态改变，就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：从Pending变为Resolved和从Pending变为Rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果。如果改变已经发生了，你再对Promise对象添加回调函数，也会立即得到这个结果。这与事件（Event）完全不同，事件的特点是，如果你错过了它，再去监听，是得不到结果的。

ES6 规定，Promise对象是一个构造函数，用来生成Promise实例。

下面代码创造了一个Promise实例。

var promise = new Promise(function(resolve, reject) {

// ... some code

if (/\* 异步操作成功 \*/){

resolve(value);

} else {

reject(error);

}

});

3.Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject。它们是两个函数，由 JavaScript 引擎提供，不用自己部署。

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。

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。

promise.then(function(value) {

// success

}, function(error) {

// failure

});

Promise 新建后就会立即执行。

4. 注意，调用resolve或reject并不会终结 Promise 的参数函数的执行。

new Promise((resolve, reject) => {

resolve(1);

console.log(2);

}).then(r => {

console.log(r);

});

// 2

// 1

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。

new Promise((resolve, reject) => {

return resolve(1);

// 后面的语句不会执行

console.log(2);

})

可以采用链式的then()实现需求，then()有两个参数，一般第二个参数不写由catch()捕获处理。

5.Promise.prototype.catch方法是.then(null, rejection)的别名，用于指定发生错误时的回调函数。

getJSON('/posts.json').then(function(posts) {

// ...

}).catch(function(error) {

// 处理 getJSON 和 前一个回调函数运行时发生的错误

console.log('发生错误！', error);

});

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。

p.then((val) => console.log('fulfilled:', val))

.catch((err) => console.log('rejected', err));

// 等同于

p.then((val) => console.log('fulfilled:', val))

.then(null, (err) => console.log("rejected:", err));

reject方法的作用，等同于抛出错误。因此只要是抛出错误，类似try {

throw new Error('test');

}这种，则reject可以不出现

需要注意的是，catch方法返回的还是一个 Promise 对象，因此后面还可以接着调用then方法。如果没有报错，则会跳过catch方法。

注意，如果作为参数的 Promise 实例，自己定义了catch方法，那么它一旦被rejected，并不会触发Promise.all()的catch方法。

**Promise.resolve()**

有时需要将现有对象转为Promise对象，Promise.resolve方法就起到这个作用。

var jsPromise = Promise.resolve($.ajax('/whatever.json'));

## Promise.reject()

Promise.reject(reason)方法也会返回一个新的 Promise 实例，该实例的状态为rejected。

var p = Promise.reject('出错了');

// 等同于

var p = new Promise((resolve, reject) => reject('出错了'))

p.then(null, function (s) {

console.log(s)

});

// 出错了

### done()

Promise对象的回调链，不管以then方法或catch方法结尾，要是最后一个方法抛出错误，都有可能无法捕捉到（因为Promise内部的错误不会冒泡到全局）。因此，我们可以提供一个done方法，总是处于回调链的尾端，保证抛出任何可能出现的错误。

asyncFunc()

.then(f1)

.catch(r1)

.then(f2)

.done();

### finally()

finally方法用于指定不管Promise对象最后状态如何，都会执行的操作。它与done方法的最大区别，它接受一个普通的回调函数作为参数，该函数不管怎样都必须执行。

下面是一个例子，服务器使用Promise处理请求，然后使用finally方法关掉服务器。

server.listen(0)

.then(function () {

// run test

})

.finally(server.stop);