

## Promise

Reads: 9186

在JavaScript的世界中，所有代码都是单线程执行的。

由于这个“缺陷”，导致JavaScript的所有网络操作，浏览器事件，都必须是异步执行。异步执行可以用回调函数实现：

```
function callback() {  
    console.log('Done');  
}  
console.log('before setTimeout()');  
setTimeout(callback, 1000); // 1秒钟后调用callback函数  
console.log('after setTimeout()');
```

观察上述代码执行，在Chrome的控制台输出可以看到：

```
before setTimeout()  
after setTimeout()  
(等待1秒后)  
Done
```

可见，异步操作会在将来的某个时间点触发一个函数调用。

AJAX就是典型的异步操作。以上一节的代码为例：

```
request.onreadystatechange = function () {  
    if (request.readyState === 4) {  
        if (request.status === 200) {  
            return success(request.responseText);  
        } else {  
            return fail(request.status);  
        }  
    }  
}
```

把回调函数 `success(request.responseText)` 和 `fail(request.status)` 写到一个AJAX操作里很正常，但是不好看，而且不利于代码复用。

有没有更好的写法？比如写成这样：

```
var ajax = ajaxGet('http://...');  
ajax.isSuccess(success)  
    .ifFail(fail);
```

这种链式写法的好处在于，先统一执行AJAX逻辑，不关心如何处理结果，然后，根据结果是成功还是失败，在将来的某个时候调用 `success` 函数或 `fail` 函数。

古人云：“君子一诺千金”，这种“承诺将来会执行”的对象在JavaScript中称为Promise对象。

Promise有各种开源实现，在ES6中被统一规范，由浏览器直接支持。先测试一下你的浏览器是否支持Promise：

```
'use strict';  
  
new Promise(function () {});  
  
// 直接运行测试：
```

```
alert('支持Promise!');
```

▶ Run

我们先看一个最简单的Promise例子：生成一个0-2之间的随机数，如果小于1，则等待一段时间后返回成功，否则返回失败：

```
function test(resolve, reject) {  
  var timeOut = Math.random() * 2;  
  log('set timeout to: ' + timeOut + ' seconds.');
```

```
  setTimeout(function () {  
    if (timeOut < 1) {  
      log('call resolve()...');
```

```
      resolve('200 OK');
```

```
    }  
    else {  
      log('call reject()...');
```

```
      reject('timeout in ' + timeOut + ' seconds.');
```

```
    }  
  }, timeOut * 1000);  
}
```

这个 `test()` 函数有两个参数，这两个参数都是函数，如果执行成功，我们将调用 `resolve('200 OK')`，如果执行失败，我们将调用 `reject('timeout in ' + timeOut + ' seconds.')`。可以看出，`test()` 函数只关心自身的逻辑，并不关心具体的 `resolve` 和 `reject` 将如何处理结果。

有了执行函数，我们就可以用一个Promise对象来执行它，并在将来某个时刻获得成功或失败的结果：

```
var p1 = new Promise(test);  
var p2 = p1.then(function (result) {  
  console.log('成功: ' + result);  
});  
var p3 = p2.catch(function (reason) {  
  console.log('失败: ' + reason);  
});
```

变量 `p1` 是一个Promise对象，它负责执行 `test` 函数。由于 `test` 函数在内部是异步执行的，当 `test` 函数执行成功时，我们告诉Promise对象：

```
// 如果成功，执行这个函数：  
p1.then(function (result) {  
  console.log('成功: ' + result);  
});
```

当 `test` 函数执行失败时，我们告诉Promise对象：

```
p2.catch(function (reason) {  
  console.log('失败: ' + reason);  
});
```

Promise对象可以串联起来，所以上述代码可以简化为：

```
new Promise(test).then(function (result) {  
  > console.log('成功: ' + result);  
}).catch(function (reason) {  
  console.log('失败: ' + reason);  
});
```

实际测试一下，看看Promise是如何异步执行的：

```
'use strict';  
  
// 清除log:  
var logging = document.getElementById('test-promise-log');  
while (logging.children.length > 1) {  
  logging.removeChild(logging.children[logging.children.length - 1]);  
}  
  
// 输出log到页面:  
function log(s) {  
  var p = document.createElement('p');  
  p.innerHTML = s;  
  logging.appendChild(p);  
}  
  
new Promise(function (resolve, reject) {  
  log('start new Promise...');  
  var timeOut = Math.random() * 2;  
  log('set timeout to: ' + timeOut + ' seconds.');
```

```
  setTimeout(function () {  
    if (timeOut < 1) {  
      log('call resolve()...');  
      resolve('200 OK');    }  
    else {  
      log('call reject()...');  
      reject('timeout in ' + timeOut + ' seconds.');    }  
  }, timeOut * 1000);  
}).then(function (r) {  
  log('Done: ' + r);  
}).catch(function (reason) {  
  log('Failed: ' + reason);  
});
```

▶ Run

Log:

start new Promise...

set timeout to: 1.4030722670071794 seconds.

call reject()...

Failed: timeout in 1.4030722670071794 seconds.

可见Promise最大的好处是在异步执行的流程中，把执行代码和处理结果的代码清晰地分离了：

```
on_resolve(data) {  
  // TODO
```



Promise还可以做更多的事情，比如，有若干个异步任务，需要先做任务1，如果成功后再做任务2，任何任务失败则不再继续并执行错误处理函数。

要串行执行这样的异步任务，不用Promise需要写一层一层的嵌套代码。有了Promise，我们只需要简单地写：

```
job1.then(job2).then(job3).catch(handleError);
```

其中，`job1`、`job2` 和 `job3` 都是Promise对象。

下面的例子演示了如何串行执行一系列需要异步计算获得结果的任务：

```
'use strict';

var logging = document.getElementById('test-promise2-log');
while (logging.children.length > 1) {
  logging.removeChild(logging.children[logging.children.length - 1]);
}

function log(s) {
  var p = document.createElement('p');
  p.innerHTML = s;
  logging.appendChild(p);
}

// 0.5秒后返回input*input的计算结果：
function multiply(input) {
  return new Promise(function (resolve, reject) {
    log('calculating ' + input + ' x ' + input + '...');
    setTimeout(resolve, 500, input * input);
  });
}

// 0.5秒后返回input+input的计算结果：
function add(input) {
  return new Promise(function (resolve, reject) {
    log('calculating ' + input + ' + ' + input + '...');
    setTimeout(resolve, 500, input + input);
  });
}

var p = new Promise(function (resolve, reject) {
  log('start new Promise...');
  resolve(123);
});

p.then(multiply)
  .then(add)
  .then(multiply)
  .then(add)
  .then(function (result) {
    log('Got value: ' + result);
  });
```

▶ Run

Log:

start new Promise...

calculating 123 x 123...

calculating 15129 + 15129...

calculating 30258 x 30258...

calculating 915546564 + 915546564...

Got value: 1831093128

`setTimeout` 可以看成是一个模拟网络等异步执行的函数。现在，我们把上一节的AJAX异步执行函数转换为Promise对象，看看用Promise如何简化异步处理：

```
'use strict';

// ajax函数将返回Promise对象：
function ajax(method, url, data) {
  var request = new XMLHttpRequest();
  return new Promise(function (resolve, reject) {
    request.onreadystatechange = function () {
      if (request.readyState === 4) {
        if (request.status === 200) {
          resolve(request.responseText);
        } else {
          reject(request.status);
        }
      }
    };
    request.open(method, url);
    request.send(data);
  });
}

var log = document.getElementById('test-promise-ajax-result');
var p = ajax('GET', '/api/categories');
p.then(function (text) { // 如果AJAX成功，获得响应内容
  log.innerText = text;
}).catch(function (status) { // 如果AJAX失败，获得响应代码
  log.innerText = 'ERROR: ' + status;
});
```

▶ Run

```
{
  "categories": [
    {
      "id": "0013738748415562fee26e070fa4664ad926c8e30146c67000",
      "name": "编程",
      "tag": "tech",
      "description": "",
      "display_order": 0,
      "created_at": 1373874841556,
      "updated_at": 1429763779958,
      "version": 5
    },
    {
      "id": "0013738748248885ddf38d8cd1b4803aa74bcda32f853fd000",
      "name": "读书",
      "tag": "other",
      "description": "",
      "display_order": 1,
      "created_at": 1373874824888,
      "updated_at": 1429763779974,
      "version": 5
    }
  ]
}
```

除了串行执行若干异步任务外，Promise还可以并行执行异步任务。

试想一个页面聊天系统，我们需要从两个不同的URL分别获得用户的个人信息和好友列表，这两个任务是可以并行执行的，用 `Promise.all()` 实现如下：

```
var p1 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 500, 'P1');
});
var p2 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 600, 'P2');
});
// 同时执行p1和p2，并在它们都完成后执行then:
Promise.all([p1, p2]).then(function (results) {
  console.log(results); // 获得一个Array: ['P1', 'P2']
});
```

有些时候，多个异步任务是为了容错。比如，同时向两个URL读取用户的个人信息，只需要获得先返回的结果即可。这种情况下，用 `Promise.race()` 实现：

```
var p1 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 500, 'P1');
});
var p2 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 600, 'P2');
});
Promise.race([p1, p2]).then(function (result) {
  console.log(result); // 'P1'
});
```

由于 `p1` 执行较快，Promise的 `then()` 将获得结果 `'P1'`。 `p2` 仍在继续执行，但执行结果将被丢弃。

如果我们组合使用Promise，就可以把很多异步任务以并行和串行的方式组合起来执行。

感觉本站内容不错，读后有收获？

¥ 我要小额赞助，鼓励作者写出更好的教程

还可以分享给朋友

分享 [四川应小云](#) , [Lambert\\_殇](#) 等31人分享过



◀ [AJAX](#)

[Canvas](#) ▶

## Comments



[关于Promise的错误处理](#)

[传说中的海贼王](#) created at 5-18 9:51, Last updated at 5-19 2:55

```
job1.then(job2).then(job3).catch(handleError);
```

最后的handleError函数是在job3失败时才执行，还是在job1,job2,job3任何一个失败就执行了？



[廖雪峰](#)

Created at 5-19 2:55, Last updated at 5-19 2:55

[View Full Discuss](#)[Reply This Topic](#)

### 只能执行两个异步函数

張小強 created at 5-17 13:49, Last updated at 5-17 13:49

只能执行两个函数? .then 一个.catch 一个?

[View Full Discuss](#)[Reply This Topic](#)

### 廖老师，关于resolve，reject函数的问题

sheyu小王 created at 4-21 7:58, Last updated at 5-6 5:32

new Promise(function (resolve, reject)

这个resolve 和 reject 是js内部的函数么?，他们本身只是引起then和catch语句，然后讲参数传递过去?

还有这2个是否可以更换成其他自定义的函数?

如果能，是不是根据前后位置分别指定为 'resolve'和 'reject'函数，并将这个函数的返回值传递给then和catch里面的函数?



kkopite

Created at 5-6 5:32, Last updated at 5-6 5:32

then里面传入resolve的具体实现  
catch出入reject的具体实现函数

[View Full Discuss](#)[Reply This Topic](#)

### setTimeout的第三个参数?

桔小涵 created at 4-18 15:31, Last updated at 4-19 9:33

setTimeout(resolve, 500, input \* input);

setTimeout的第三个参数，是setTimeout第一个参数也就是resolve函数的参数?



廖雪峰

Created at 4-19 9:33, Last updated at 4-19 9:33

setTimeout如果参数多于两个，从第三个参数开始，全部传给第一个函数作为参数:

```
function fn(x, y, z) {  
  console.log(x+' '+y+' '+z);  
}  
  
setTimeout(fn, 1000, 'A', 'B', 'C'); // 1秒后执行fn('A', 'B', 'C')
```

[View Full Discuss](#)[Reply This Topic](#)



## 关于异步callback函数的设置问题

郑翰同学好震撼 created at 4-4 7:01, Last updated at 4-5 3:26



廖大大，  
解释Promise流程机制的图里面的async函数的命名是不是有些误导啊  
看了JS官方的文档，  
new Promise的参数executor会立刻执行，"It runs before the Promise constructor returns the created object."  
async是同步执行的函数，  
真正异步执行的是由resolve和reject触发的.then以及.catch参数中的函数吧？

这段小代码可以测试运行的顺序：

```
var test = new Promise(function(res, rej){
  alert('Now, we are in executor...');
  res('Send a signal to .then()');
});
alert('we have just create the Promise object -- test!');
test.then(function(r){
  alert('We just receive the signal from res(), which says\n'+ ''+r+'');
});
```

Promise设置异步callback采用的方法是  
promise.then(async)

▼ [Read More](#)



廖雪峰

Created at 4-5 3:26, Last updated at 4-5 3:26

异步有很多形式，回调最简单，但是嵌套太难看，promise算另一种，也不好看

写异步的时候，还是用yield吧，真正的“同步”模式写异步代码。ES7据说会把yield升级到async和await，就更简单了

≡ View Full Discuss

↩ Reply This Topic



廖老师，求问下下面这个问题：

beijing\_lmx created at 2015-9-13 2:43, Last updated at 4-4 7:08

在此插入代码

```
var p1 = new Promise(function (resolve, reject) {
  log('start new Promise...');
  resolve('200 ok');
});

log('hello1!');

var p2 = p1.then(function (r) {
  log('Done: ' + r);
});

log('hello2!');

p2.catch(function (reason) {
  log('Failed: ' + reason);
});

log('hello3!');
```



在此插入代码

Log:

start new Promise...

hello1!

hello2!

hello3!

Done: 200 ok

promise内的函数调用不太明白其具体的执行时机。

⤴ Collapse



后背有理想

Created at 2015-11-27 9:22, Last updated at 2015-11-27 9:22

正因为是异步的，所以你确定不了什么时候返回



郑翰同学好震撼

Created at 4-4 7:08, Last updated at 4-4 7:08

```
new Promise(function (resolve, reject) {  
  log('start new Promise...');  
  resolve('200 ok');  
  
});
```

这个部分是顺序执行的  
先执行内部函数  
然后再创建Promise对象

≡ View Full Discuss

↩ Reply This Topic



最大的疑问是.then()与.resolve()谁先执行？

乌鸦与水 created at 2015-12-1 5:23, Last updated at 4-4 7:06

如果.then()执行较慢或是不执行，在new Promise(function (resolve, reject) {})中的resolve()执行不不执行，此时的resolve有没有指向处理函数？有些不明白。



郑翰同学好震撼


Created at 4-4 7:06, Last updated at 4-4 7:06

我是这么理解的：  
执行了resolve只是给then发送了一个信号  
在设置then之后，promise object会检查收到的信号  
如果收到信号了 那么.then就直接执行  
如果没有收到信号 .then就等待

真正异步执行的是.then()的参数函数

你可以试一下这个

```
var test = new Promise(function(res, rej){
    alert('Now, we are in executor...');
    res('Send a signal to .then()');
});
alert('we have just create the Promise object -- test!');
test.then(function(r){
    alert('We just receive the signal from res(), which says\n'+ ''+r+'');
});
```

 View Full Discuss

 Reply This Topic



### 可否增加async和await的内容

[BlinkD](#) created at 4-1 5:27, Last updated at 4-1 9:15


感觉这两个会是异步编程的终极解决方案



[廖雪峰](#)

Created at 4-1 9:15, Last updated at 4-1 9:15

那是ES7的草案，还早着呢

 View Full Discuss

 Reply This Topic



### 用了这么久，才知道setTimeout可以带参数

[赵大欣 Freckles](#) created at 3-31 4:08, Last updated at 3-31 4:08

mark一下

```
var timeoutID = window.setTimeout(func, [delay, param1, param2, ...]);
var timeoutID = window.setTimeout(code, [delay]);
```

 View Full Discuss

 Reply This Topic



### 代码有错误

[Anx的城](#) created at 2015-12-22 3:31, Last updated at 2015-12-22 4:43


`if (timeOut &lt; 1)` 中的 `&lt;` 被转义了



[廖雪峰](#)

Created at 2015-12-22 4:43, Last updated at 2015-12-22 4:43

看得仔细!

 View Full Discuss

 Reply This Topic

Make a Comment

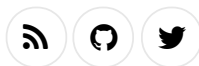
---

廖雪峰的官方网站©2015

Powered by [iTranswarp.js](https://iTranswarp.js)

由[阿里云](#)托管

[广告合作](#)



---

友情链接: [中华诗词](#) - [阿里云](#) - [SICP](#) - [4closure](#)