异步操作- Promise 对象 (ES6)

《JavaScript教程》作者：阮一峰

Promise 原本只是社区提出的一个构想，一些函数库率先实现了这个功能。ECMAScript 6 将其写入语言标准，目前 JavaScript 原生支持 Promise 对象。

1 概述

Promise 对象是 JavaScript 的异步操作解决方案，为异步操作提供统一接口。它起到代理作用（proxy），充当异步操作与回调函数之间的中介。

Promise 的设计思想是，所有异步任务都返回一个 Promise 实例。Promise 实例有一个then方法，用来指定下一步的回调函数。

首先，Promise 是一个对象，也是一个构造函数。

```
function f1(resolve, reject) {
  // 异步代码...
}

var p1 = new Promise(f1);
```

上面代码中，Promise构造函数接受一个回调函数f1作为参数，f1里面是异步操作的代码。然后，返回的p1就是一个 Promise 实例。

```
var p1 = new Promise(f1);
p1.then(f2);
```

Promise 实例有一个then方法，用来指定下一步的回调函数。上面代码中，f1的异步操作执行完成，就会执行f2。

2 Promise 对象的状态

Promise 对象通过自身的状态，来控制异步操作。Promise 实例具有三种状态：

- 异步操作未完成（pending）
- 异步操作成功（fulfilled）
- 异步操作失败（rejected）

上面三种状态里面，fulfilled和rejected合在一起称为 resolved（已定型）。

这三种的状态的变化途径只有两种。

- 从“未完成”到“成功”
- 从“未完成”到“失败”

Promise 的最终结果只有两种。

- 异步操作成功，Promise 实例传回一个值（value），状态变为fulfilled。
- 异步操作失败，Promise 实例抛出一个错误（error），状态变为rejected。

3 Promise 构造函数

JavaScript 提供原生的Promise构造函数，用来生成 Promise 实例。

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

  if (/* 异步操作成功 */){
    resolve(value);
  } else { /* 异步操作失败 */
    reject(new Error());
  }
});
```

上面代码中，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，不用自己实现。

resolve函数的作用是，将Promise实例的状态从“未完成”变为“成功”（即从pending变为fulfilled），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实例的状态从“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。

下面是一个例子。

```
function timeout(ms) {
  return new Promise((resolve, reject) => {
    setTimeout(resolve, ms, 'done');
  });
}

timeout(100)
```

上面代码中，timeout(100)返回一个 Promise 实例。100毫秒以后，该实例的状态会变为fulfilled。

4 Promise.prototype.then()

Promise 实例的then方法，用来添加回调函数。

then方法可以接受两个回调函数，第一个是异步操作成功时（变为fulfilled状态）的回调函数，第二个是异步操作失败（变为rejected）时的回调函数（该参数可以省略）。一旦状态改变，就调用相应的回调函数。

then方法可以链式使用。

```
p1
  .then(step1)
  .then(step2)
  .then(step3)
  .then(
    console.log,
    console.error
  );
```

上面代码中，p1后面有四个then，意味依次有四个回调函数。只要前一步的状态变为fulfilled，就会依次执行紧跟在后面的回调函数。

最后一个then方法，回调函数是console.log和console.error，用法上有一点重要的区别。console.log只显示step3的返回值，而console.error可以显示p1、step1、step2、step3之中任意一个发生的错误。

5 then() 用法辨析

四种写法

```
// 写法一
f1().then(function () {
  return f2();
});

// 写法二
f1().then(function () {
  f2();
});

// 写法三
f1().then(f2());

// 写法四
f1().then(f2);
```

写法一：

```
f1().then(function () {
  return f2();
}).then(f3);
```

f3回调函数的参数，是f2函数的运行结果。

写法二：

```
f1().then(function () {
  f2();
  return;
}).then(f3);
```

f3回调函数的参数是undefined。

写法三：

```
f1().then(f2())
  .then(f3)
```

f3回调函数的参数，是f2函数返回的函数的运行结果。

写法四：

```
f1().then(f2)
  .then(f3);
```

f3回调函数的参数，是f2函数的运行结果。

写法四与写法一只有一个差别，那就是f2会接收到f1()返回的结果。

6 实例：图片加载

```
var preloadImage = function (path) {
  return new Promise(function (resolve, reject) {
    var image = new Image();
    image.onload  = resolve;
    image.onerror = reject;
    image.src = path;
  });
};
```

上面代码中，image是一个图片对象的实例。它有两个事件监听属性，onload属性在图片加载成功后调用，onerror属性在加载失败调用。

上面的preloadImage()函数用法如下。

```
preloadImage('https://example.com/my.jpg')
  .then(function (e) { document.body.append(e.target) })
  .then(function () { console.log('加载成功') })
```

上面代码中，图片加载成功以后，onload属性会返回一个事件对象，因此第一个then()方法的回调函数，会接收到这个事件对象。该对象的target属性就是图片加载后生成的 DOM 节点。

7 Promise 优缺点

Promise 的优点在于，让回调函数变成了规范的链式写法，程序流程可以看得很清楚。它有一整套接口，可以实现许多强大的功能，比如同时执行多个异步操作，等到它们的状态都改变以后，再执行一个回调函数；再比如，为多个回调函数中抛出的错误，统一指定处理方法等等。

而且，Promise 还有一个传统写法没有的好处：它的状态一旦改变，无论何时查询，都能得到这个状态。这意味着，无论何时为 Promise 实例添加回调函数，该函数都能正确执行。所以，你不用担心是否错过了某个事件或信号。如果是传统写法，通过监听事件来执行回调函数，一旦错过了事件，再添加回调函数是不会执行的。

Promise 的缺点是，编写的难度比传统写法高，而且阅读代码也不是一眼可以看懂。你只会看到一堆then，必须自己在then的回调函数里面理清逻辑。

8 微任务

Promise 的回调函数不是正常的异步任务，而是微任务（microtask）。它们的区别在于，正常任务追加到下一轮事件循环，微任务追加到本轮事件循环。

例如

```
setTimeout(function() {
  console.log(1);
}, 0);

new Promise(function (resolve, reject) {
  resolve(2);
}).then(console.log);

console.log(3);
```

上面代码的输出结果是321。因为then是本轮事件循环执行，setTimeout(fn, 0)在下一轮事件循环开始时执行。