JavaScript Promise

作者：Rzg
原文链接：https://ld246.com/article/1681112565507
来源网站：链滴

Promise 是一个 ECMAScript 6 提供的类，目的是更加优雅地书写复杂的异步任务。

由于 Promise 是 ES6 新增加的，所以一些旧的浏览器并不支持，苹果的 Safari 10 和 Windows 的 Edge 14 版本以上浏览器才开始支持 ES6 特性。

以下是 Promise 浏览器支持的情况：

dge 14	Firefox 54	Safari 10	Chrome 58 Opera 55

构造 Promise

现在我们新建一个 Promise 对象：

```
new Promise(function (resolve, reject) {
    // 要做的事情...
});
```

通过新建一个 Promise 对象好像并没有看出它怎样 "更加优雅地书写复杂的异步任务"。我们之前遇的异步任务都是一次异步，如果需要多次调用异步函数呢？例如，如果我想分三次输出字符串，第一间隔 1 秒，第二次间隔 4 秒，第三次间隔 3 秒：

实例

```
setTimeout(function() {
    console.log("First");
    setTimeout(function() {
        console.log("Second");
        setTimeout(function() {
            console.log("Third");
        },
        3000);
    },
    4000);
},
1000);
```

这段程序实现了这个功能，但是它是用 "函数瀑布" 来实现的。可想而知，在一个复杂的程序当中，用函数瀑布" 实现的程序无论是维护还是异常处理都是一件特别繁琐的事情，而且会让缩进格式变得非冗赘。

现在我们用 Promise 来实现同样的功能：

实例

```
new Promise(function(resolve, reject) {
    setTimeout(function() {
        console.log("First");
        resolve();
    },
    1000);
}).then(function() {
```

```
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            console.log("Second");
            resolve();
        },
        4000);
    });
}).then(function() {
    setTimeout(function() {
        console.log("Third");
    },
    3000);
});
```

Promise 将嵌套格式的代码变成了顺序格式的代码。

Promise 的构造函数

Promise 构造函数是 JavaScript 中用于创建 Promise 对象的内置构造函数。

Promise 构造函数接受一个函数作为参数，该函数是同步的并且会被立即执行，所以我们称之为起始数。起始函数包含两个参数 resolve 和 reject，分别表示 Promise 成功和失败的状态。

起始函数执行成功时，它应该调用 resolve 函数并传递成功的结果。当起始函数执行失败时，它应该用 reject 函数并传递失败的原因。

Promise 构造函数返回一个 Promise 对象，该对象具有以下几个方法：

- then：用于处理 Promise 成功状态的回调函数。
- catch：用于处理 Promise 失败状态的回调函数。
- finally：无论 Promise 是成功还是失败，都会执行的回调函数。

下面是一个使用 Promise 构造函数创建 Promise 对象的例子：

当 Promise 被构造时，起始函数会被同步执行：

实例

```
const promise = new Promise((resolve, reject) = >{
    setTimeout(() = >{
        if (Math.random() < 0.5) {
            resolve('success');
        } else {
            reject('error');
        }
    },
    1000);
});
promise.then(result = >{
    console.log(result);
}).
catch(error = >{
```

```
    console.log(error);
});
```

在上面的例子中，我们使用 Promise 构造函数创建了一个 Promise 对象，并使用 setTimeout 模拟一个异步操作。如果异步操作成功，则调用 resolve 函数并传递成功的结果；如果异步操作失败，则用 reject 函数并传递失败的原因。然后我们使用 then 方法处理 Promise 成功状态的回调函数，使用 atch 方法处理 Promise 失败状态的回调函数。

这段程序会直接输出 **error** 或 **success**。

resolve 和 reject 都是函数，其中调用 resolve 代表一切正常，reject 是出现异常时所调用的：

实例

```
new Promise(function(resolve, reject) {
    var a = 0;
    var b = 1;
    if (b == 0) reject("Divide zero");
    else resolve(a / b);
}).then(function(value) {
    console.log("a / b = " + value);
}).
catch(function(err) {
    console.log(err);
}).
finally(function() {
    console.log("End");
});
```

这段程序执行结果是:

```
a / b = 0
End
```

Promise 类有 .then() .catch() 和 .finally() 三个方法，这三个方法的参数都是一个函数，.then() 可以参数中的函数添加到当前 Promise 的正常执行序列，.catch() 则是设定 Promise 的异常处理序列，.finally() 是在 Promise 执行的最后一定会执行的序列。 .then() 传入的函数会按顺序依次执行，有任何常都会直接跳到 catch 序列：

实例

```
new Promise(function(resolve, reject) {
    console.log(1111);
    resolve(2222);
}).then(function(value) {
    console.log(value);
    return 3333;
}).then(function(value) {
    console.log(value);
    throw "An error";
}).
catch(function(err) {
    console.log(err);
```

```
});
```

执行结果：

```
1111
2222
3333
An error
```

resolve() 中可以放置一个参数用于向下一个 then 传递一个值，then 中的函数也可以返回一个值传给 then。但是，如果 then 中返回的是一个 Promise 对象，那么下一个 then 将相当于对这个返回的 romise 进行操作，这一点从刚才的计时器的例子中可以看出来。

reject() 参数中一般会传递一个异常给之后的 catch 函数用于处理异常。

但是请注意以下两点：

- resolve 和 reject 的作用域只有起始函数，不包括 then 以及其他序列；
- resolve 和 reject 并不能够使起始函数停止运行，别忘了 return。

Promise 函数

上述的 "计时器" 程序看上去比函数瀑布还要长，所以我们可以将它的核心部分写成一个 Promise 函：

实例

```
function print(delay, message) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            console.log(message);
            resolve();
        },
        delay);
    });
}
```

然后我们就可以放心大胆的实现程序功能了：

实例

```
print(1000, "First").then(function() {
    return print(4000, "Second");
}).then(function() {
    print(3000, "Third");
});
```

这种返回值为一个 Promise 对象的函数称作 Promise 函数，它常常用于开发基于异步操作的库。

回答常见的问题（FAQ）

Q: then、catch 和 finally 序列能否顺序颠倒?

A: 可以，效果完全一样。但不建议这样做，最好按 then-catch-finally 的顺序编写程序。

Q: 除了 then 块以外，其它两种块能否多次使用?

A: 可以，finally 与 then 一样会按顺序执行，但是 catch 块只会执行第一个，除非 catch 块里有异。所以最好只安排一个 catch 和 finally 块。

Q: then 块如何中断?

A: then 块默认会向下顺序执行，return 是不能中断的，可以通过 throw 来跳转至 catch 实现中断。

Q: 什么时候适合用 Promise 而不是传统回调函数?

A: 当需要多次顺序执行异步操作的时候，例如，如果想通过异步方法先后检测用户名和密码，需要先步检测用户名，然后再异步检测密码的情况下就很适合 Promise。

Q: Promise 是一种将异步转换为同步的方法吗?

A: 完全不是。Promise 只不过是一种更良好的编程风格。

Q: 什么时候我们需要再写一个 then 而不是在当前的 then 接着编程?

A: 当你又需要调用一个异步任务的时候。

异步函数

异步函数（async function）是 ECMAScript 2017 (ECMA-262) 标准的规范，几乎被所有浏览器所持，除了 Internet Explorer。

在 Promise 中我们编写过一个 Promise 函数：

实例

```
function print(delay, message) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            console.log(message);
            resolve();
        },
        delay);
    });
}
```

然后用不同的时间间隔输出了三行文本：

实例

```
print(1000, "First").then(function() {
    return print(4000, "Second");
}).then(function() {
    print(3000, "Third");
});
```

我们可以将这段代码变得更好看：

实例

```
async
function asyncFunc() {
    await print(1000, "First");
    await print(4000, "Second");
    await print(3000, "Third");
}
asyncFunc();
```

哈！这岂不是将异步操作变得像同步操作一样容易了吗！

这次的回答是肯定的，异步函数 async function 中可以使用 await 指令，await 指令后必须跟着一个 Promise，异步函数会在这个 Promise 运行中暂停，直到其运行结束再继续运行。

异步函数实际上原理与 Promise 原生 API 的机制是一模一样的，只不过更便于程序员阅读。

处理异常的机制将用 try-catch 块实现：

实例

```
async
function asyncFunc() {
    try {
        await new Promise(function(resolve, reject) {
            throw "Some error";
        });
    } catch(err) {
        console.log(err);
    }
}
asyncFunc();
```

如果 Promise 有一个正常的返回值，await 语句也会返回它：

实例

```
async
function asyncFunc() {
    let value = await new Promise(function(resolve, reject) {
        resolve("Return value");
    });
    console.log(value);
}
asyncFunc();
```

程序会输出:

```
Return value
```

更多内容

JavaScript Promise 对象