

JavaScript ...

JavaScript 教程

JavaScript 简介

JavaScript 用法

JavaScript

Chrome 中运行

JavaScript 输出

JavaScript 语法

JavaScript 语句

JavaScript 注释

JavaScript 变量

JavaScript 数据类型

JavaScript 对象

JavaScript 函数

JavaScript 作用域

JavaScript 事件

JavaScript 字符串

JavaScript 运算符

JavaScript 比较

JavaScript 条件语句

JavaScript switch 语句

JavaScript for 循环

JavaScript while 循环

← JavaScript 异步编程

JavaScript 代码规范 →

JavaScript Promise

在学习本章节内容前，你需要先了解什么是异步编程，可以参考：[JavaScript 异步编程](#)

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：

				
Chrome 58	Edge 14	Firefox 54	Safari 10	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"。我们之前遇到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，第一次间隔 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常冗赘。

现在我们用 Promise 来实现同样的功能：

实例

分类导航

HTML / CSS

JavaScript

服务端

数据库

数据分析

移动端

XML 教程

ASP.NET

Web Service

开发工具

网站建设

Advertisement



反馈/建议

JavaScript break
和 continue 语句

JavaScript typeof

JavaScript 类型
转换

JavaScript 正则
表达式

JavaScript 错误

JavaScript 调试

JavaScript 变量
提升

JavaScript 严格
模式

JavaScript 使用
误区

JavaScript 表单

JavaScript 表单
验证

JavaScript 验证
API

JavaScript 保留
关键字

JavaScript this

JavaScript let 和
const

JavaScript JSON

JavaScript void

JavaScript 异步
编程

JavaScript
Promise

JavaScript 代码
规范

JS 函数

JavaScript 函数
定义

```
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);
});
```

这段代码较长，所以还不需要完全理解它，我想引起注意的是 Promise 将嵌套格式的代码变成了顺序格式的代码。

Promise 的使用

下面我们通过剖析这段 Promise "计时器" 代码来讲述 Promise 的使用：

Promise 构造函数只有一个参数，是一个函数，这个函数在构造之后会直接被异步运行，所以我们称之为起始函数。起始函数包含两个参数 resolve 和 reject。

当 Promise 被构造时，起始函数会被异步执行：

实例

```
new Promise(function (resolve, reject) {
  console.log("Run");
});
```

这段程序会直接输出 **Run**。

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
```

华为云
云嘉年
服务器
元/年

数千企
使用的
低至
消费满
Mate X



反馈/建议

JavaScript 函数
参数

JavaScript 函数
调用

JavaScript 闭包

JS HTML DOM

DOM 简介

DOM HTML

DOM CSS

DOM 事件

DOM

EventListener

DOM 元素

HTMLCollection

对象

NodeList 对象

JS 高级教程

JavaScript 对象

JavaScript
prototype

JavaScript

Number 对象

JavaScript String

JavaScript Date
(日期)

JavaScript Array
(数组)

JavaScript
Boolean (布尔)

JavaScript Math
(算数)

JavaScript
RegExp 对象

JS 浏览器BOM

JavaScript
Window

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()` 可以将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，有任何异常都会直接跳到 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递给 `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 Promise 进行操作，这一点从刚才的计时器的例子中可以看出。

`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 函数：

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

反馈/建议



JavaScript
Window Screen

JavaScript
Window Location

JavaScript
Window History

JavaScript
Navigator

JavaScript 弹窗

JavaScript 计时
事件

JavaScript
Cookie

JS 库

JavaScript 库

JavaScript 测试
jQuery

JavaScript 测试
Prototype

JS 实例

JavaScript 实例

JavaScript 对象
实例

JavaScript 浏览
器对象实例

JavaScript HTML
DOM 实例

JavaScript 总结

JS 参考手册

JavaScript 对象

HTML DOM 对象

JavaScript 异步
编程

```
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 块里有异常。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，需要先异步检测用户名，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支持，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");  
});
```



反馈/建议

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"; // 或者 reject("Some error")
    });
  } catch (err) {
    console.log(err);
    // 会输出 Some error
  }
}
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





学习异步时，对函数里面的 `resolve()` 不太理解，后来上网查了查，知道了他的作用：



118

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）。

通过回调里的 `resolve(data)` 将这个 promise 标记为 `resolved`，然后进行下一步 `then((data)=>{//do something})`，`resolve` 里的参数就是你要传入 `then` 的数据。

龙 6个月前 (05-11)

在线实例

- HTML 实例
- CSS 实例
- JavaScript 实例
- Ajax 实例
- jQuery 实例
- XML 实例
- Java 实例

字符集&工具

- HTML 字符集设置
- HTML ASCII 字符集
- HTML ISO-8859-1
- HTML 实体符号
- HTML 拾色器
- JSON 格式化工具

最新更新

- React component...
- React component...
- React getSnapsh...
- React shouldCom...
- React component...
- React render() ...
- React getDerive...

站点信息

- 意见反馈
- 免责声明
- 关于我们
- 文章归档

关注微信

