JavaScript 中异步编程的研究与应用

刘为之 李 林
四川大学锦城学院计算机与软件学院 四川 成都 611731

【摘要】由于 JavaScript 在编程中不能同时执行多个任务，因此，异步编程的出现使得这些任务不再需要排队，而是可以异步进行。并且随着 ES6 中 Promise 对象的提出，使得 JavaScript 在异步编程方面有了进一步的提升。本文就此针对 Promise 对象进行了一些讨论。

【关键词】异步编程；Promise 对象；async/await

1 引言

随着 JavaScript 从原来作为浏览器的脚本语言，到现在还为后端服务器服务，JavaScript 的功能一直在不断更新、不断强化。单线程模式的 JavaScript 在异步编程的功能下，增加了任务执行效率，也使得其可以更好地应对多种多样的任务需求以及实现。

2 JavaScript 异步模式简述

在 JavaScript 中，由于执行环境是单线程的原因，让 JavaScript 在执行任务时一次只能够执行一个任务，而之后的任务只能够在队列中排队进行等待。这样会让所要执行的任务不能够立即得到执行，影响任务执行的效率。JavaScript 语言对于浏览器来说，其最主要的目的就是用于对 DOM 进行渲染，只有单线程模式才能避免同时会有多段对 DOM 进行修改的操作。因此，JavaScript 语言选择了单线程模式。虽然该模式可以使得其运行环境变得相对单纯并且更加容易实现；但缺点就是有多个任务同时进行时，JavaScript 只会让这些任务进行排队等候，只有在先完成前面一个任务的情况下才会继续执行后面的任务。所以，为了解决多个任务不能同时进行的问题。JavaScript 下的异步编程模式也就由此产生。

JavaScript 在进行异步编程时，首先所有的任务都会在主线程上等待执行。开始工作时，产生堆和栈。在执行栈中，异步任务就会去外部调用对应的 API，随后会在消息队列中加入各种不一样的事件。当主线程中的同步任务执行完毕后，就会接着去循环消息队列，获取这些事件然后执行。完成这样一次循环则称为事件循环，主线程则会一直进行这样的事件循环。这些被调用的事件是在主线程中被执行的，不是在工作线程。因此，这种工作模式使得 JavaScript 在处理事件的模式上保证了事件不会因此发生阻塞现象。

异步编程主要有四种方法：1. 回调函数。2. 事件监听。3. 发布订阅。4. Promise 对象。本文主要研究第四种：Promise 对象。

3 Promise 对象的研究

本文在介绍 promise 对象之前，首先引入两个概念：宏任务和微任务。在 JavaScript 中，宏任务有 setTimeout 方法和 setInterval 方法；微任务有 promise 和 process.nextTick 方法。JavaScript 的执行顺序就是自上而下逐行执行代码，那么在遇到宏任务和微任务时，会将微任务放到微任务队列中，将宏任务放到宏任务队列中。首先循环微任务队列，将其中所有的微任务执行并输出，当微任务队列为空时，才会开始循环宏任务队列，将宏任务进行输出。一次事件循环的结束标志就是当前所有的宏任务都执行完毕后，且微任务队列中为空，没有还未执行的微任务。

根据 Promise/A 规范，promise 是用来处理异步操作的对象。它本身有 resolve、reject 等方法，原型上有 then、catch 等方法。Promise 对象有三种状态：pending、resolved 和 rejected。Pending 意 味着 promise 的状态为进行中，还没有完成。当状态为 resolved 时说明 promise 为完成状态；状态为 rejected 时，表示 promise 的状态为失败。Promise 的状态就如同它本身的名字含义一样，只有 resolve 函数和 reject 函数才能够改变它的状态。其中 resolve 函数和 reject 函数就是改变 promise 状态的方法。resolve 是将 promise 的状态从 pending 改变成 fulfilled，reject 则是将 pending 改变成 rejected，当 promise 的状态改变后，就不会在进行改变，且 resolved 和 rejected 之间的状态也是不会相互改变的。

Promise 有两个原型方法。第一个原型方法是 then 方法。它有两个参数，一个是 resolve 函数的回调，另一个是 reject 函数的回调，后者可选。第二个原型方法是 catch 方法，该方法则只接受 reject 函数的回调。当 promise 的状态为 rejected 时，catch 方法就会接受该回调并进行输出。通常，在使用 then 方法时，只设置第一个参数，也就是只接受 resolved 的回调，一般使用 catch 方法来接受 rejected 的回调。对于 promise 来说，链式调用是它最突出的部分。由于 then 方法返回的是一个新的 promise 对象，那么就可以如同链式一般，对 then 方法不断进行调用。此外，在链式调用的 then 方法末尾可以追加一个 catch 方法，该 catch 方法就是来接受 rejected 的回调，处理运行中出现的错误。但要注意的是，若前面的 then 方法中返回了一个 rejected 回调，那么后面的 then 方法就不会调用，而是直接跳到末尾的 catch 方法。

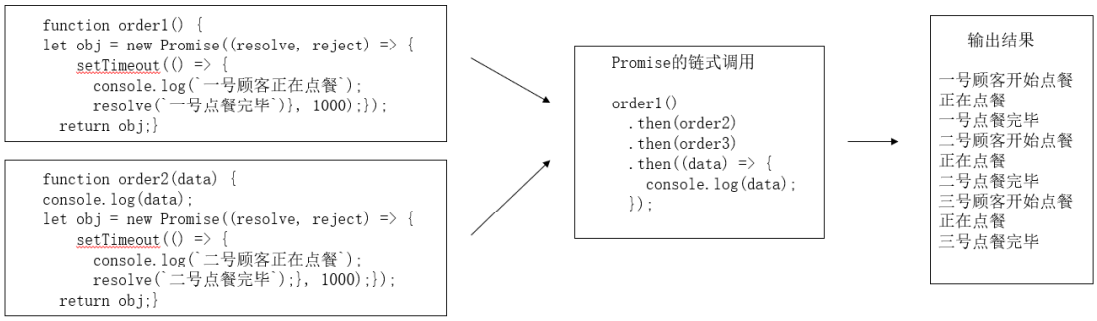

图 1 promise 链式调用示例图

如图 1 代码所示，首先先定义三个函数，order3 的格式与 order2 一致。其中 order2 和 order3 函数都增加了一个 data 形参，这个参数就是接受前面一个函数中 promise 对象返回的 resolve 函数参数。每个函数中创建了一个 promise 对象，并给其设置两个形参，用于接收 resolve 和 reject 函数。在函数内部用了 setTimeout 方法，在该方法里面进行输出并对 resolve 函数传值。在对于它们的链式调用时，前一个 resolve 函数的值会传递给下一个函数的 data 形参，输出结果就如图中最右所示。

随着 ES2017 的提出，引入了一个全新的 async 函数。该函数作为一个语法糖，也是基于 promise 的实现。Async 函数的用法就是将原本的同步函数前面加上 async 关键字，使其成为一个异步函数。该函数一般会搭配 await 进行，且 await 不能再没有 async 声明函数的前提下单独使用。await 返回的是一个 promise 对象。当执行 async 函数时，若函数内部有 await 关键字，那么就会先进行等待并开始运行 await 后面的代码，然后返回一个 promise 对象。只有在执行完后，才会继续后面的任务。若 await 的异步操作成功，则会将 promise 的状态改为 resolved，若失败则是将状态改为 rejected，根据其状态来决定后面的操作。

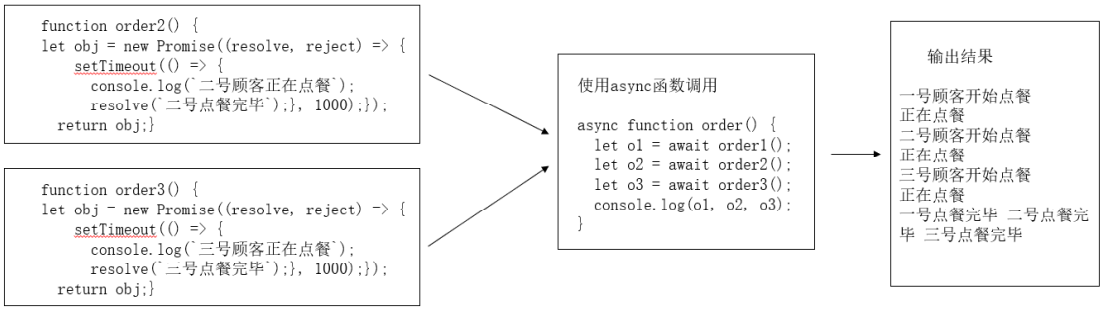

图 2 async 函数调用示例图

如图 2 所示，该代码展示了用 async 函数修改之后的示例。async 函数对于 promise 对象的调用则显得更加的规范整洁，由于 await 返回的是一个 promise 对象，每个 promise 对象返回的 resolved 状态信息就不再需要下一个函数通过形参接收，而是会在最后一起输出，因此图中的函数就不需要形参，并取消对参数的输出。通过 async 函数输出。这样不仅能减少函数代码的复杂程度，还能够更好的呈现出函数调用后的结果。

4 Promise 对象的应用

由于 Promise 提供了异步编程统一的范式和 API，具有广泛的应用场景，以笔者在疫情防控管理系统中采用的 Axios 为例。Axios 是一种 ajax 技术，它也是一种基于 promise 的异步访问技术。在该项目中，首先就是封装一个 axios 方法，接受从前端界面上传来的数据，将该数据与后端数据进行匹配，从而进行验证并进行下一步操作。前端页面则是通过调用封装好的 axios 方法，将用户输入的信息通过该 axios 方法传递给后端并进行处理，实现了前后端的联系。

```
let Axios = (options) => {
  axios({
url: options.url,
method: options.method || “POST”,
data: options.data,
params: options.data,
  }).then((result) => {
   if (options.success) { options.success(result.data); }
}).catch((err) => {
   let msg = err.response ? err.response.data : “请求异常”;
   if (options.error) { options.error(msg);
Message.error({ message: msg, offset: 150 });
   } else {Message.error({ message: msg, offset: 150 });}});};
```

该代码展示的是在 Vue 项目的 main.js 文件中进行的对 axios 方法封装的代码，首先就是通过 axios 方法接收从前端传递的信息，由于 axios 是基于 promise 的

方法，后续则是通过链式调用，对应的使用 then 方法和 catch 方法对不同的状态进行处理操作。在该项目中，先将获取到的信息进行判断，将结果封装成一个 promise 对象，该对象通过一个简单的链式调用进行传递，通过 promise 对象的状态来决定下一步调用的方法。传统的 ajax 请求，在向服务器进行多个顺序请求时，则需要进行代码的层层嵌套，因此产生出一个回调地狱。由于 axios 是基于 promise 的方法，因此在 axios 进行顺序请求时，可以通过链式调用避免回调地狱。这也是使用 promise 一个最主要的原因。当对用户输入的信息进行顺序查询时，axios 就会将结果封装成一个 promise 对象。当用户通过用户名和密码进行登录时，axios 就会获取到用户输入的信息，通过判断用户信息是否匹配，来对结果进行一个封装。若正确，则会封装成一个状态为 resolved 的 promise 对象，此时 then 方法就会接受这个返回值并且进行相应操作。否则会封装成一个状态为 rejected 的对象，此时就会被 catch 方法接收并调用。

Axios 使用 promise 来对方法进行封装的优点就是减少重复代码的编写，并能减少代码的层层嵌套避免回调地狱。Axios 方法会将结果进行一个封装，封装成 promise 对象，then 方法和 catch 方法就会对操作成功和失败的结果进行对应的调用。另一个优点就是便于代码的维护和管理。对于传统的 ajax，随着代码量的增多和嵌套层数的增多，当发生错误时，不利于对代码的检查和更正。Promise 将代码变得简洁之后，就便于开发人员对项目进行精准的错误定位，以此减少对于代码的错误检查步骤，方便维护和管理。

5 结语

Promise 对象的产生，使 JavaScript 对于任务的异步处理变得更加便捷高效，对比于传统的回调函数及其他异步方式，promise 的出现让原本需要层层嵌套的代码得以改善，阻止了回调地狱的产生，从而使得执行流程可视化。通过链式调用能及时发现错误并进行更改和维护。不过 promise 的链式调用也可能会产生回调地狱，且当有大量数据处理时，链式调用会让代码显得复杂。在并发执行的任务和顺序执行的任务的应用场景下，promise 则能很好地处理这类任务。并且由于 es7 中 async 函数的产生，promise 在链式调用上的缺点和问题多数得以优化。因此，在处理异步任务时，promise 对象依然是首选。

【参考文献】

[1] 钱宇虹 . 如何用 Java 回调和线程实现异步调用 [J]. 软件工程师 ,2013(10)

[2] 陈浩 . 基于 JavaScript 的异步编程分析 [J]. 电脑知识与技术 ,2015,11(13) :80-81.

[3] 周安辉 .Node.js 异步编程模式探讨 [J]. 四川职业技术学院学报 ,2018,28(04) :149-154