</talentlabs>

What is Async Processing and Why?

Preface

This reading materials aimed at explaining what async processing is and why we need it. This would help you in understanding the subsequent videos on Promises and Async/Await.

Sequential Processing (a.k.a. Synchronous Processing)

When you are building a program in the JavaScript module, you would find that all of your programs are executing step-by-step, one after another. It's kind of like when you are following a recipe, you will just follow the steps one by one. However, this might be problematic in computer programs.

Imagine the following scenario:
I am having a program that would fetch data from 5 different data sources and display them on a webpage. The third data source is not very stable and would have a very long response time.

If we are running these 5 data fetches one-by-one, what would happen? The program might get stuck at the third data source and look like it has crashed. Users might restart the app, or just close the app and never use it together.

Asynchronous Processing (a.k.a Async Processing)

The solution to the issue mentioned above is Asynchronous Processing. That means the 5 data fetches would not be processed one by one. Instead, they would be fired in parallel, running together. Whichever comes back first, we will just process that response first and display it on the webpage.

If we implement this asynchronous processing successfully, then all these processes would not be blocking each other. The faster process will get done sooner and the slower process will get done later. There won't be any blockage between them.

Two ways of doing Async Processing in JavaScript

There are actually three ways of doing async processing in JavaScript:

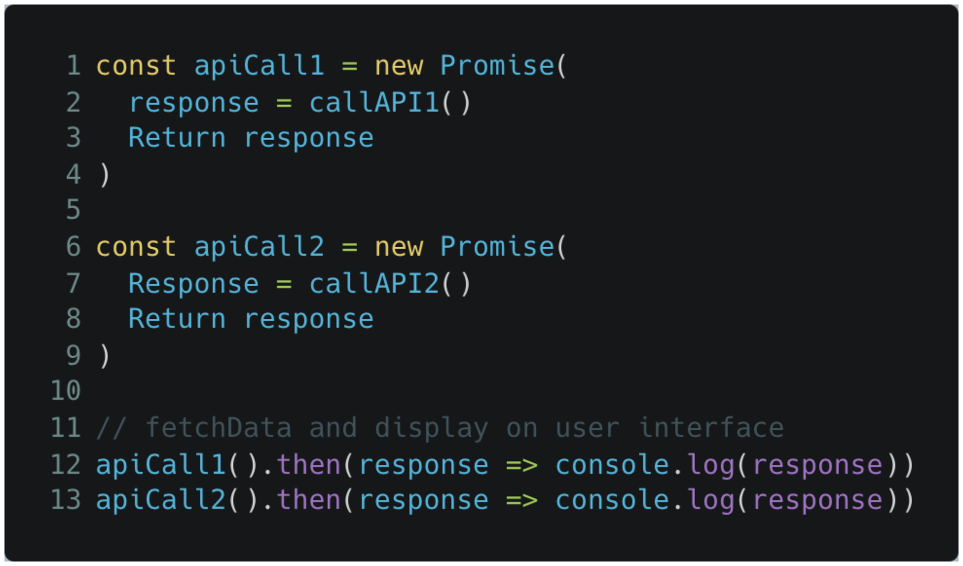
- Callback
- Promise
- Async/Await

However, the first one (Callback) is old-fashioned and very hard to manage in complex applications. It is proven to be a bad practice in JavaScript and most of the developers are not using them anymore.

Instead, most of the developers go for the second and third method now.

Promise

Promise is basically about wrapping a piece of processing or code as a “promise”. This “promise” is gonna return the processing results some time later when it is ready. Let’s take a look at a simple example below. (Do note that these are not real JavaScript code, they are just to show you the concept of Promise)



```
 1 const apiCall1 = new Promise(
 2   response = callAPI1()
 3   Return response
 4 )
 5
 6 const apiCall2 = new Promise(
 7   Response = callAPI2()
 8   Return response
 9 )
10
11 // fetchData and display on user interface
12 apiCall1().then(response => console.log(response))
13 apiCall2().then(response => console.log(response))
```

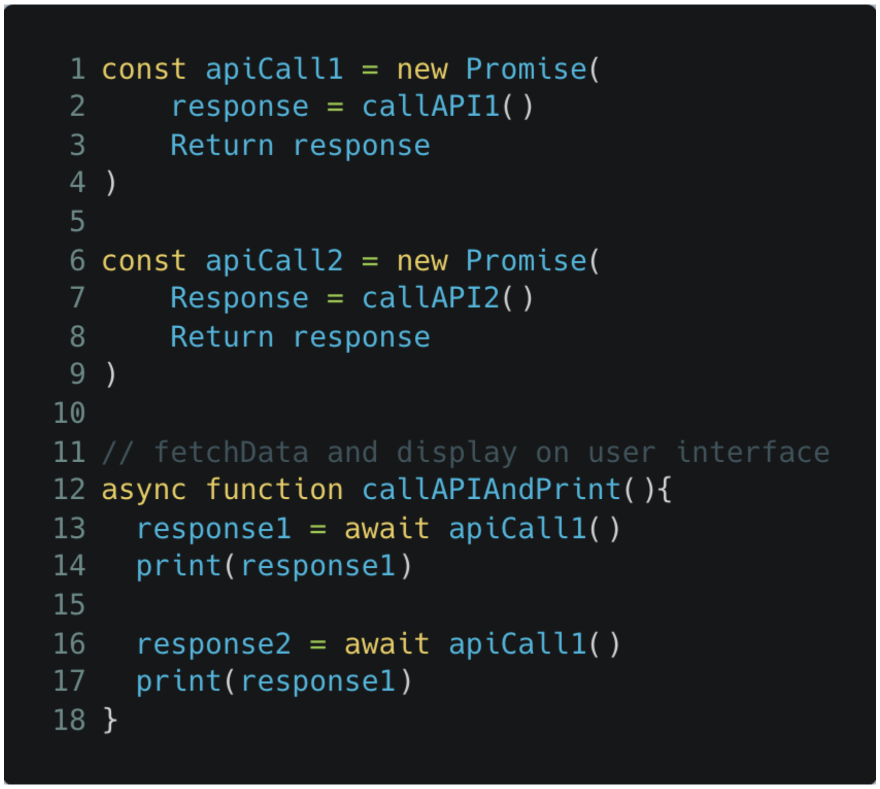
In the above example, the two API calls are being triggered one-by-one, but they are not blocking each other. After calling “`apiCall1()`” (line 12), it would immediately go to “`apiCall2`” without waiting for “`apiCall1()`” to be completed. You can consider the code is being run in the background.

So, the question is, what would happen if the “`apiCall1()`” is done? It is specified by using the “`.then`” statement. Within the “`.then`” statement, we have specified that once there is a response coming back from “`apiCall1()`”, then we would display it in the console using “console.log”.

In this way, both “`apiCall1()`” and “`apiCall2()`” are being run behind the scenes and would not block the subsequent execution. Even if “`apiCall1()`” is super slow, it would not block “`apiCall2()`” from executing.

Async/Await

Async/Await is actually not async processing. It is in fact the opposite of async execution. It is being used for converting async processing into synchronous sequential execution.



```
 1 const apiCall1 = new Promise(
 2     response = callAPI1()
 3     Return response
 4 )
 5
 6 const apiCall2 = new Promise(
 7     Response = callAPI2()
 8     Return response
 9 )
10
11 // fetchData and display on user interface
12 async function callAPIAndPrint(){
13   response1 = await apiCall1()
14   print(response1)
15
16   response2 = await apiCall1()
17   print(response1)
18 }
```

Looking into this example, you can see that we are still having 2 async apiCall functions (`apiCall1` and `apiCall2`). But this time, we want to ensure that these 2 apiCalls are executed in a sequential manner, one after another. In this case, we don’t need to make changes to apiCall1 and apiCall2 functions. We just need to add the “await” keyword when we call `apiCall1` and `apiCall2` (line 13 and 16).

Once we added that, the `apiCall1()` function would not be run in the background. Instead, it would wait at line 13 until `apiCall1()` is done and return with the response before moving to line 14.

In this way, we are essentially converting an async processing function or process that returns a promise, into the traditional synchronous processing.

Usually, we will do this for one of the following reasons

Summary

This is a very quick overview of async processing in JavaScript. In our subsequent videos, we will explain more on how to use promise and async/await.

— End of Preface —