An introduction to Async Await

Skills Network

As you might have already learnt, Java Script is a single-threaded scripting language. That means the process can happen only sequentially and no two processes can happen simultaneously. This is a big deterrent to any language and JS solved this by introducing asynchronous programming through Promises. We have learnt about promises and seen some examples of the same. While Promises solved the issues with synchronous programming, nested `then` can compilcate the structure and readability of the code.

In ES 2017, Async/Await was introduced which addressed this issue and gave way to cleaner, readable code. We will understand the working of async/await in the light of the some examples we used for Promise, for better understanding. By awaiting a promise, we can process the result as and when the promise is fulfilled (or rejected).

In the following code sample, we have created a Promise with a callback where we handle resolve and reject.

```javascript

const axios = require('axios');

const connectToURL = (url)=>{
  const req = axios.get(url);
  console.log(req);
  req.then(resp => {
      let listOfEntries = resp.data.entries;
      listOfEntries.forEach((entry)=>{
        console.log(entry.Category);
      });
    })
  .catch(err => {
      console.log(err.toString())
  });
}
console.log("Before connect URL")
connectToURL('https://api.publicapis.org/entries');
console.log("After connect URL")
```

Copied!

We will see how the same is accomplished with async/await.

```javascript
const axios = require('axios');
const connectToURL = async(url)=>{
    const outcome = axios.get(url);
    let listOfEntries = (await outcome).data.entries;
    listOfEntries.forEach((entry)=>{
      console.log(entry.Category);
    });
}

console.log("Before connect URL")
connectToURL('https://api.publicapis.org/entries');
console.log("After connect URL")
```

Copied!

The best use of async/await can be realized when we have a scenario where some async methods have to happen in sequence. Taking the same example as above, let's first get a list of all entries by and based on categories, send request to get the details of each entry in that category. So, these two actions have to happen one after the other but asynchronously. These can be accomplished with or without async/await. But chaining actions is much cleaner with async await, as you can observe below. In actual situations, the nesting can be multiple level and rendering the code difficult to read and maintain. In such situations, we could use async/await.

The below code is done by nesting the second set of promises into the first.

```
const axios = require('axios');

/*
In the following code we try to get list of all entries from remote url and then based on that make request about each of the
category. Finally print them all out. We are using axios get, which returns a promise.
*/
const axios = require('axios');

/*
In the following code we try to get list of all entries from remote url and then based on that make request about
each of the category. Finally print them all out. We are using axios get, which returns a promise.
*/
const connectToURL = (url)=>{
  const req = axios.get(url);
  req.then(resp => {
      let listOfEntries = resp.data.entries;
      return listOfEntries.map((entry)=>{
          return entry.Category
      })
    }).then((categories)=>{
      let Categories = categories.filter(function(item, pos, self) {
        return self.indexOf(item) == pos;
    })

        Categories.forEach((category)=>{
            const req = axios.get("https://api.publicapis.org/entries?Category="+category);
            req.then(resp=>{
                console.log(category+" - "+resp.data.count);
            }).catch(err => {

            })
        });
    })
  .catch(err => {
      console.log(err.toString())
  });
}
connectToURL('https://api.publicapis.org/entries');
```

Copied!

The same objective is attained using async/await.

```
const axios = require('axios');

/*
In the following code we try to get list of all entries from remote url and then based on that make request about each of the
category starting with 'A'. Finally print the API counts of the category. We are using axios get, which returns a promise.
*/
const axios = require('axios');

async function connectToURL(url){
    const resp = await axios.get(url);
    let listOfEntries = resp.data.entries;
    let Categories = listOfEntries.map((entry)=>{
          return entry.Category
    });
    Categories = [...new Set(Categories)];

    Categories.forEach(async (Category)=>{
      if (Category.startsWith("A")) {
              try {
                const resp = await axios({
                  method: 'get',
                  url: "https://api.publicapis.org/entries?Category="+Category,
                  responseType: 'json'
                })
                console.log(Category+"   "+resp.data.count);
              }
              catch(e) {
                console.log(e);
              }
      }

    });
}
connectToURL('https://api.publicapis.org/entries').catch(err => {
    console.log(err.toString())
});
```

Copied!

You can only await a promise inside an async method. This is because `await` blocks the thread. This will defeat the primary purpose. So the function within which an `await` is used HAS TO BE async.

Author(s)

Lavanya

Changelog

Date	Version	Changed by	Change Description
2023-05-04	1.2	K Sundararajan	Corrected spellings and formatting based on reviews
2022-10-31	1.1	Lavanya	Changed the API URL
2020-11-25	1.0	Lavanya	Created Lab for Asyn Await as a part of async callback programming