

An introduction to Async Await



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 complicate 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.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19

1.
2. const axios = require('axios');
3.
4. const connectToURL = (url)=>{
5.   const req = axios.get(url);
6.   console.log(req);
7.   req.then(resp => {
8.     let listOfEntries = resp.data.entries;
9.     listOfEntries.forEach((entry)=>{
10.      console.log(entry.Category);
11.    });
12.  })
13.  .catch(err => {
14.    console.log(err.toString())
15.  });
16. }
17. console.log("Before connect URL")
18. connectToURL('https://api.publicapis.org/entries');
19. console.log("After connect URL")
```

Copied!

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

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12

1. const axios = require('axios');
2. const connectToURL = async(url)=>{
3.   const outcome = axios.get(url);
4.   let listOfEntries = (await outcome).data.entries;
5.   listOfEntries.forEach((entry)=>{
6.     console.log(entry.Category);
7.   });
8. }
9.
10. console.log("Before connect URL")
11. connectToURL('https://api.publicapis.org/entries');
12. 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.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38

1. const axios = require('axios');
2.
3. /*
4. 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
5. category. Finally print them all out. We are using axios get, which returns a promise.
6. */
7. const axios = require('axios');
8.
9. /*
10. In the following code we try to get list of all entries from remote url and then based on that make request about
11. each of the category. Finally print them all out. We are using axios get, which returns a promise.
12. */
13. const connectToURL = (url)=>{
14.   const req = axios.get(url);
15.   req.then(resp => {
16.     let listOfEntries = resp.data.entries;
17.     return listOfEntries.map((entry)=>{
18.       return entry.Category
19.     })
20.   }).then((categories)=>{
21.     let Categories = categories.filter(function(item, pos, self) {
22.       return self.indexOf(item) == pos;
23.     })
24.
25.     Categories.forEach((category)=>{
26.       const req = axios.get("https://api.publicapis.org/entries?Category="+category);
27.       req.then(resp=>{
28.         console.log(category+" - "+resp.data.count);
29.       }).catch(err => {
30.
31.       })
32.     });
33.   })
34.   .catch(err => {
35.     console.log(err.toString())
36.   });
37. }
38. connectToURL('https://api.publicapis.org/entries');
```

Copied!

The same objective is attained using async/await.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36

1. const axios = require('axios');
2.
3. /*
4. 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
5. category starting with 'A'. Finally print the API counts of the category. We are using axios get, which returns a promise.
6. */
7. const axios = require('axios');
8.
9. async function connectToURL(url){
10.   const resp = await axios.get(url);
11.   let listOfEntries = resp.data.entries;
12.   let Categories = listOfEntries.map((entry)=>{
13.     return entry.Category
14.   });
15.   Categories = [...new Set(Categories)];
16.
17.   Categories.forEach(async (Category)=>{
18.     if (Category.startsWith("A")) {
19.       try {
20.         const resp = await axios({
21.           method: 'get',
22.           url: "https://api.publicapis.org/entries?Category="+Category,
23.           responseType: 'json'
24.         })
25.         console.log(Category+"    "+resp.data.count);
26.       }
27.       catch(e) {
28.         console.log(e);
29.       }
30.     }
31.   });
32. }
33. }
34. connectToURL('https://api.publicapis.org/entries').catch(err => {
35.   console.log(err.toString())
36. });

```

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

© IBM Corporation 2023. All rights reserved.