Asynchronous JavaScript

async await Explained

🚀 Everything about JS async await in just **20 pages**

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Agenda

- async await
- Awaiting Multiple Promises
- Error Handling in async await
- Behind the scenes of async await

Arpitha Rajeev
arpitha.rajeev37@gmail.com

async await

- While **Promises** are introduced in **ES6** to make asynchronous programming simple by removing callback hell
- **async** and **await** are introduced in **ES2017** to simplify the use of **Promises**
- We can only use the **await** keyword within functions that have been declared with the **async** keyword

Arpitha Rajeev
arpitha.rajeev37@gmail.com

async await

- Declaring a function **async** means that the **return** value of the function will be a **Promise** even if no Promise-related code appears in the body of the function
- **await** keyword takes a Promise and turns it back into a return value or a thrown exception

```
async function getHighScore() {
    let response = await fetch("/api/user/profile");
    let profile = await response.json();
    return profile.highScore;
}
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

async await

- **await fetch**() in the above example will wait until the **Promise p** returned by **fetch**() settles
- If **Promise p** if fulfilled, then the value of the **await fetch** is the fulfillment value of p
- If **p** is rejected, then the **await fetch** expression throws the rejection value of p
- **await** is basically used when we need to wait for something to happen, once it happens it then executes the next line

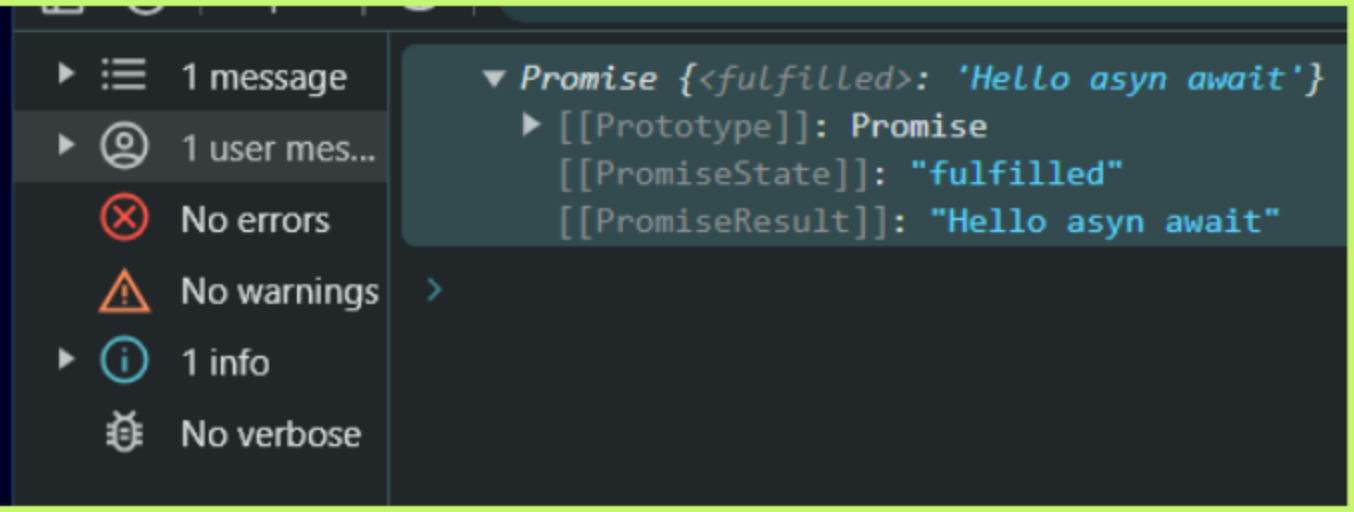
Arpitha Rajeev
arpitha.rajeev37@gmail.com

async await

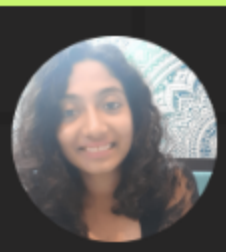
```
async function getData() {
    return 'Hello asyn await';
}

const data = getData();

console.log(data);
```



- In this example, even though **getData**() function is returning a string, it is wrapped up in a **Promise**
- When we console log the data, it prints a Promise that has the state **fulfilled** and a **string** as a **result**
- Hence, using **then**() we can print the result **data.then**(**res** => **console.log**(**res**))



Arpitha Rajeev
arpitha.rajeev37@gmail.com

async await Example

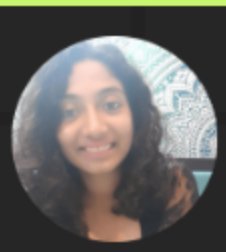
```
const p = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('Resolved');
    }, 2000);
});

async function handle() {
    const val = await p;
    console.log('Done');
    console.log(val);
}

handle();
console.log('Prints without waiting')
```

Output:

```
Prints without waiting
Done
Resolved
```



Arpitha Rajeev
arpitha.rajeev37@gmail.com

async await

- In the above example, line 13 will be printed immediately as it the main program flow that won't get affected by the **async function**
- But line 9 and line 10 will not be printed until **val** is resolved because of the **await** keyword
- Every line of code inside an **async function** that follows an **await expression** will wait until the awaited Promise is resolved before executing
- But lines of code that come before an **await expression** will be executed immediately

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Awaiting Multiple Promises

- Suppose we have these 2 lines inside a **async function,**

```
let value1 = await getJSON(url1);
let value2 = await getJSON(url2);
```

- Fetching **url2** will not start until we fetch **url1,** if value2 does not depend on value1, this code is not efficient and we should try to fetch both **urls** at the same time
- To await a set of concurrently executing async functions, we use **Promise.all**()

```
let [value1, value2] = await Promise.all([getJSON(url1), getJSON(url2)]);
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Error Handling in async await

```
async function fetchData(url) {
    try {
        const response = await fetch(url);
        if (!response.ok) {
            throw new Error('Not ok');
        }
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
fetchData(url);
```

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Error Handling in async await

- **async await** is just a **syntactic sugar** on the use of Promises
- Instead of using **then**() and **catch**() method, **try ... catch block** is used to handle errors in **async await**

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

- When **async function** is called, it gets added to a JavaScript engine's callstack
- When JS engine comes to **await expression**, it removes the **async function** from the callstack until the promise is resolved and goes on to execute the lines in the main program flow
- When the promise is resolved, **async function** is back on to the callstack and lines following the **await keyword** gets executed

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

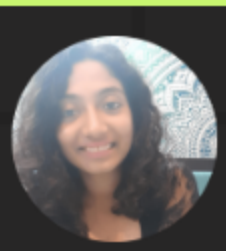
- When **async function** is called, it gets added to a JavaScript engine's callstack
- When JS engine comes to **await expression**, it removes the **async function** from the callstack until the promise is resolved and goes on to execute the lines in the main program flow
- When the promise is resolved, **async function** is back on to the callstack and lines following the **await keyword** gets executed

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

```
const p1 = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('Resolved 1');
    }, 2000);
});

const p2 = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('Resolved 2');
    }, 3000);
});
```



Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

Continuing...

```
async function handle() {
    const val1 = await p1;
    console.log(val1);

    const val2 = await p2;
    console.log(val2);
}

handle();
```


Arpitha Rajeev

arpitha.rajeev37@gmail.com

Behind the Scenes of async await

Step 1:

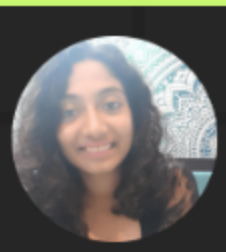
- Initialization (Time = 0ms)
- p1 and p2 are created and their timers start immediately.
- p1 is set to resolve after 2000ms.
- p2 is set to resolve after 3000ms.
- handle function is called.

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

Step 2:

- The handle function starts.
- Encounters await p1, which pauses the function until p1 resolves.
- Execution of the handle function is paused and returns a pending promise.
- The JavaScript engine continues to process other tasks in the event loop.



Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

Step 3 (Time = 2000ms):

- p1 resolves after 2000ms.
- The promise returned by p1 fulfills and places the continuation of the handle function on the callstack
- val1 is assigned 'Resolved 1'.
- console.log(val1) logs 'Resolved 1'.
- After logging 'Resolved 1', the function encounters await p2.
- Execution of handle is paused again until p2 resolves.

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

Step 4 (Time = 3000ms from start, but 1000ms after p1 resolution):

- p2 resolves 3000ms after the start (or 1000ms after p1 resolves).
- The promise returned by p2 fulfills and places the continuation of the handle function on the callstack.
- val2 is assigned 'Resolved 2'.
- console.log(val2) logs 'Resolved 2'.

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Behind the Scenes of async await

- If both the promises are set to resolve at the same time, then they get printed at the same time because in the initialization phase, they are set to resolve as per the time mentioned
- Then the **await p1** will wait for the **p1** to get resolved and then **await p2** but since they are set to resolve at the same initially, they will be printed together
- This proves that JS engine waits for none

Arpitha Rajeev
arpitha.rajeev37@gmail.com

Follow Me

For more such content on Software Development

Arpitha Rajeev
arpitha.rajeev37@gmail.com