JavaScript Async/Await Explained in 10 Minutes

July 10th 2017 · **JavaScript (https://tutorialzine.com/tag/javascript)**

For the longest of time JavaScript developers had to rely on callbacks for working with asynchronous code. As a result, many of us have experienced callback hell and the horror one goes through when faced with functions looking like this (https://tutorialzine.com/media/2017/07/callback-hell.jpg).

Thankfully, then (or should we say `.then()`) came Promises. They offered a much more organized alternative to callbacks and most of the community quickly moved on to using them instead.

Now, with the most recent addition of Async/Await, writing JavaScript code is about to get even better!

What is Async/Await?

Async/Await is a long anticipated JavaScript feature that makes working with asynchronous functions much more enjoyable and easier to understand. It is build on top of Promises and is compatible with all existing Promise-based APIs.

The name comes from `async` and `await` - the two keywords that will help us clean up our asynchronous code:

Async - declares an asynchronous function (`async function someName(){...}`).

- Automatically transforms a regular function into a Promise.
- When called async functions resolve with whatever is returned in their body.
- Async functions enable the use of `await`.

Await - pauses the execution of async functions. (`var result = await someAsyncCall();`).

- When placed in front of a Promise call, `await` forces the rest of the code to wait until that Promise finishes and returns a result.
- Await works only with Promises, it does not work with callbacks.
- Await can only be used inside `async` functions.

Here is a simple example that will hopefully clear things up:

Let's say we want to get some JSON file from our server. We will write a function that uses the axios (https://github.com/mzabriskie/axios) library and sends a HTTP GET request to (https://tutorialzine.com/misc/files/example.json)https://tutorialzine.com/misc/files/example.json (https://tutorialzine.com/misc/files/example.json). We have to wait for the server to respond, so naturally this HTTP request will be asynchronous.

Below we can see the same function implemented twice. First with Promises, then a second time using Async/Await.

```
// Promise approach

function getJSON(){

    // To make the function blocking we manually create a Promise.
    return new Promise( function(resolve) {
        axios.get('https://tutorialzine.com/misc/files/example.json')
            .then( function(json) {

                // The data from the request is available in a .then block
                // We return the result using resolve.
                resolve(json);
            });
    });

}

// Async/Await approach

// The async keyword will automatically create a new Promise and return it.
async function getJSONAsync(){

    // The await keyword saves us from having to write a .then() block.
    let json = await axios.get('https://tutorialzine.com/misc/files/example.json');

    // The result of the GET request is available in the json variable.
    // We return it just like in a regular synchronous function.
    return json;
}
```

It's pretty clear that the Async/Await version of the code is much shorter and easier to read. Other than the syntax used, both functions are completely identical - they both return Promises and resolve with the JSON response from axios. We can call our async function like this:

```
getJSONAsync().then( function(result) {
    // Do something with result.
});
```

So, does Async/Await make promises obsolete?

No, not at all. When working with Async/Await we are still using Promises under the hood. A good understanding of Promises will actually help you in the long run and is highly recommended.

There are even use cases where Async/Await doesn't cut it and we have to go back to Promises for help. One such scenario is when we need to make multiple independent asynchronous calls and wait for all of them to finish.

If we try and do this with async and await, the following will happen:

```
async function getABC() {
  let A = await getValueA(); // getValueA takes 2 second to finish
  let B = await getValueB(); // getValueB takes 4 second to finish
  let C = await getValueC(); // getValueC takes 3 second to finish

  return A*B*C;
}
```

Each await call will wait for the previous one to return a result. Since we are doing one call at a time the entire function will take 9 seconds from start to finish (2+4+3).

This is not an optimal solution, since the three variables `A`, `B`, and `C` aren't dependent on each other. In other words we don't need to know the value of `A` before we get `B`. We can get them at the same time and shave off a few seconds of waiting.

To send all requests at the same time a `Promise.all()` is required. This will make sure we still have all the results before continuing, but the asynchronous calls will be firing in parallel, not one after another.

```
async function getABC() {
  // Promise.all() allows us to send all requests at the same time.
  let results = await Promise.all([ getValueA, getValueB, getValueC ]);

  return results.reduce((total,value) => total * value);
}
```

This way the function will take much less time. The `getValueA` and `getValueC` calls will have already finished by the time `getValueB` ends. Instead of a sum of the times, we will effectively reduce the execution to the time of the slowest request (getValueB - 4 seconds).

Handling Errors in Async/Await

Another great thing about Async/Await is that it allows us to catch any unexpected errors in a good old try/catch block. We just need to wrap our `await` calls like this:

```
async function doSomethingAsync(){
    try {
        // This async call may fail.
        let result = await someAsyncCall();
    }
    catch(error) {
        // If it does we will catch the error here.
    }
}
```

The catch clause will handle errors provoked by the awaited asynchronous calls or any other failing code we may have written inside the try block.

If the situation requires it, we can also catch errors upon executing the async function. Since all async functions return Promises we can simply include a `.catch()` event handler when calling them.

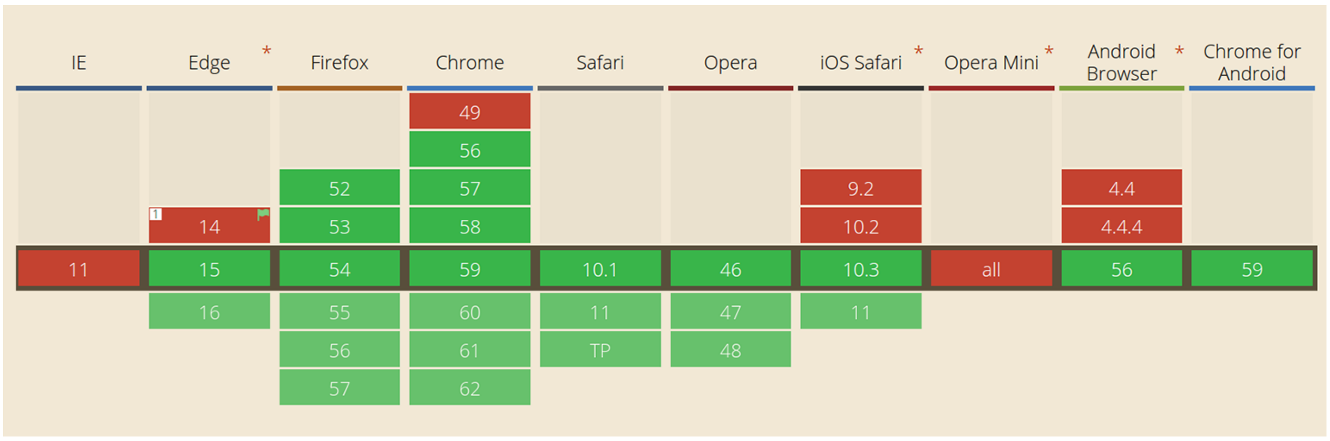
```
// Async function without a try/catch block.
async function doSomethingAsync(){
    // This async call may fail.
    let result = await someAsyncCall();
    return result;
}

// We catch the error upon calling the function.
doSomethingAsync().
    .then(successHandler)
    .catch(errorHandler);
```

It's important to choose which method of error handling you prefer and stick to it. Using both try/catch and .catch() at the same time will most probably lead to problems.

Browser Support

Async/Await is already available in most major browsers. This excludes only IE11 - all other vendors will recognize your async/await code without the need of external libraries.



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			🟥 49						
			🟩 56						
		🟩 52	🟩 57			🟥 9.2		🟥 4.4	
	🟥 1 14	🟩 53	🟩 58			🟥 10.2		🟥 4.4.4	
🟥 11	🟩 15	🟩 54	🟩 59	🟩 10.1	🟩 46	🟩 10.3	🟥 all	🟩 56	🟩 59
	🟩 16	🟩 55	🟩 60	🟩 11	🟩 47	🟩 11			
		🟩 56	🟩 61	🟩 TP	🟩 48				
		🟩 57	🟩 62						

(http://caniuse.com/#search=await)

Async/Await Browser Support

Node developers can also enjoy the improved async flow as long as they are on Node 8 or newer. It should become LTS later this year.

If this compatibility doesn't satisfy you, there are also several JS transpilers like Babel (https://babeljs.io/docs/plugins/transform-async-to-generator/) and TypeScript (https://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-3.html), and the Node.js library asyncawait (https://github.com/yortus/asyncawait) that offer their own cross-platform versions of the feature.

Conclusion

(https://www.facebook.com/sharer/sharer.php?u=https://tutorialzine.com/2017/07/javascript-async-await-explained)

With the addition of Async/Await the JavaScript language takes a huge leap forward in terms of code readability and ease of use. The ability to write asynchronous code that resembles regular synchronous functions will be appreciated by both beginners to JavaScript and veteran coders.

(http://twitter.com/share?text=JavaScript Async/Await Explained in 10 Minutes&url=https://tutorialzine.com/2017/07/javascript-async-await-explained&via=tutorialzine)

- Async on MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function)
- Await on MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await)

(https://plus.google.com/share?url=https://tutorialzine.com/2017/07/javascript-async-await-explained)

- Async/Await: The Hero JavaScript Deserved (https://www.twilio.com/blog/2015/10/asyncawait-the-hero-javascript-deserved.html)
- Where Did Async/Await Come from and Why Use It? (https://appendto.com/2017/06/asyncawait-come-use/)

Related Articles

Learn TypeScript in 30 Minutes

(https://tutorialzine.com/2016/07/learn-typescript-in-30-minutes)

Learn Git in 30 Minutes

(https://tutorialzine.com/2016/06/learn-git-in-30-minutes)
Quick Learn (https://tutorialzine.com/tag/quick-learn)

(https://www.facebook.com/sharer/sharer.php?u=https://tutorialzine.com/2017/07/javascript-async-await-explained)

(http://twitter.com/share?text=JavaScript Async/Await Explained in 10 Minutes&url=https://tutorialzine.com/2017/07/javascript-async-await-explained&via=tutorialzine)

(https://plus.google.com/share?url=https://tutorialzine.com/2017/07/javascript-async-await-explained)

Learn SQL In 20 Minutes

(https://tutorialzine.com/2016/01/learn-sql-in-20-minutes)
MySQL (https://tutorialzine.com/tag/mysql) | Quick Learn (https://tutorialzine.com/tag/quick-learn)

Comments 1

Oldest

Log In To Comment (Https://Tutorialzine.Com/Login)

tz4life (https://tutorialzine.com/@tz4life)
(https://tutorialzine.com/@tz4life)
3 months ago

As you can see in the simple example of async function on MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function). You can achieve what `Promise.all` does with `async` / `await` aswell.

```
function resolveAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function add1(x) {
  var a = resolveAfter2Seconds(20);
  var b = resolveAfter2Seconds(30);
  return x + await a + await b;
}

add1(10).then(v => {
  console.log(v);  // prints 60 after 2 seconds.
});

async function add2(x) {
  var a = await resolveAfter2Seconds(20);
  var b = await resolveAfter2Seconds(30);
  return x + a + b;
}

add2(10).then(v => {
  console.log(v);  // prints 60 after 4 seconds.
});
```

(https://www.facebook.com/sharer/sharer.php?u=https://tutorialzine.com/2017/07/javascript-async-await-explained)

(http://twitter.com/share?text=JavaScript Async/Await Explained in 10 Minutes&url=https://tutorialzine.com/2017/07/javascript-async-await-explained&via=tutorialzine)

(https://plus.google.com/share?url=https://tutorialzine.com/2017/07/javascript-async-await-explained)

Reply · **1**

A community that learns together.

Article License (https://tutorialzine.com/license)
Privacy Policy (https://tutorialzine.com/privacy)
Contact Form (https://tutorialzine.com/contact)
The Book (https://tutorialzine.com/books/jquery-trickshots)

(http://www.facebook.com/Tutorialzine)
(http://twitter.com/Tutorialzine)
(https://plus.google.com/102299652175684141333)

(https://www.facebook.com/sharer/sharer.php?u=https://tutorialzine.com/2017/07/javascript-async-await-explained)

(http://twitter.com/share?text=JavaScript Async/Await Explained in 10 Minutes&url=https://tutorialzine.com/2017/07/javascript-async-await-explained&via=tutorialzine)

(https://plus.google.com/share?url=https://tutorialzine.com/2017/07/javascript-async-await-explained)