Promises X Async/Await — what's the difference? When to use what!?

Jaqueline Nacarate · Follow

4 min read · Sep 9

AI generate image

Javascript executes code in a sequential manner, one operation at a time (single-threaded), and it can't be interrupted by other tasks(non-preemptive). Asynchronous operations can block the main thread of execution and make the application unresponsive.

Promises and Async/Await are powerful techniques for managing asynchronous operations in JavaScript. Besides being closely related, they

have different syntax and usage patterns.

Promises:

A promise is an object that provides a structured way to work with values that may not be available immediately but will be resolved at some point in the future, either successfully or with an error. This allows the main thread of execution to continue with other tasks while the asynchronous operation is being performed in the background.

Promises States:

Promises can be in one of three states:

1. **Pending:** Initially, a Promise is in the pending state while the asynchronous operation is ongoing.
2. **Fulfilled (Resolved):** When the operation succeeds, and we call `resolve`, the Promise transitions to the fulfilled state and holds the resolved value.
3. **Rejected:** If an error occurs, and we call `reject`, the Promise transitions to the rejected state and holds the error information.

When the asynchronous operation is completed, the Promise will either be **fulfilled** with a **value** or **rejected** with an **error.**

Consuming Promises:

To work with the result of a Promise, we use the `.then()` method to attach a callback that will be executed when the Promise is resolved. We can also use the `.catch()` method to handle errors if the Promise is rejected.

Chaining Promises:

We can chain multiple `.then()` handlers together to create a sequence of asynchronous operations. Each `.then()` handler receives the value returned by the previous one, allowing us to pass data between asynchronous steps.

```
// Example using promises
const customPromise = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Promise Resolved");
    }, 3000);
  });
};

console.log("Starting the Promise");

customPromise()
  .then((data) => {
    console.log(data);
    console.log("First Promise Completed");

    // Chaining another Promise
    return new Promise((resolve) => {
      setTimeout(() => {
        resolve("Second Promise Resolved");
      }, 2000);
    });
  })
  .then((secondData) => {
    console.log(secondData);
    console.log("Second Promise Completed");
  })
  .catch((err) => {
    console.log(err);
  });

// Output order:
// Starting the Promise
// Promise Resolved
// First Promise Completed
// Second Promise Resolved
// Second Promise Completed
```

Promise basics explained using a birthday example

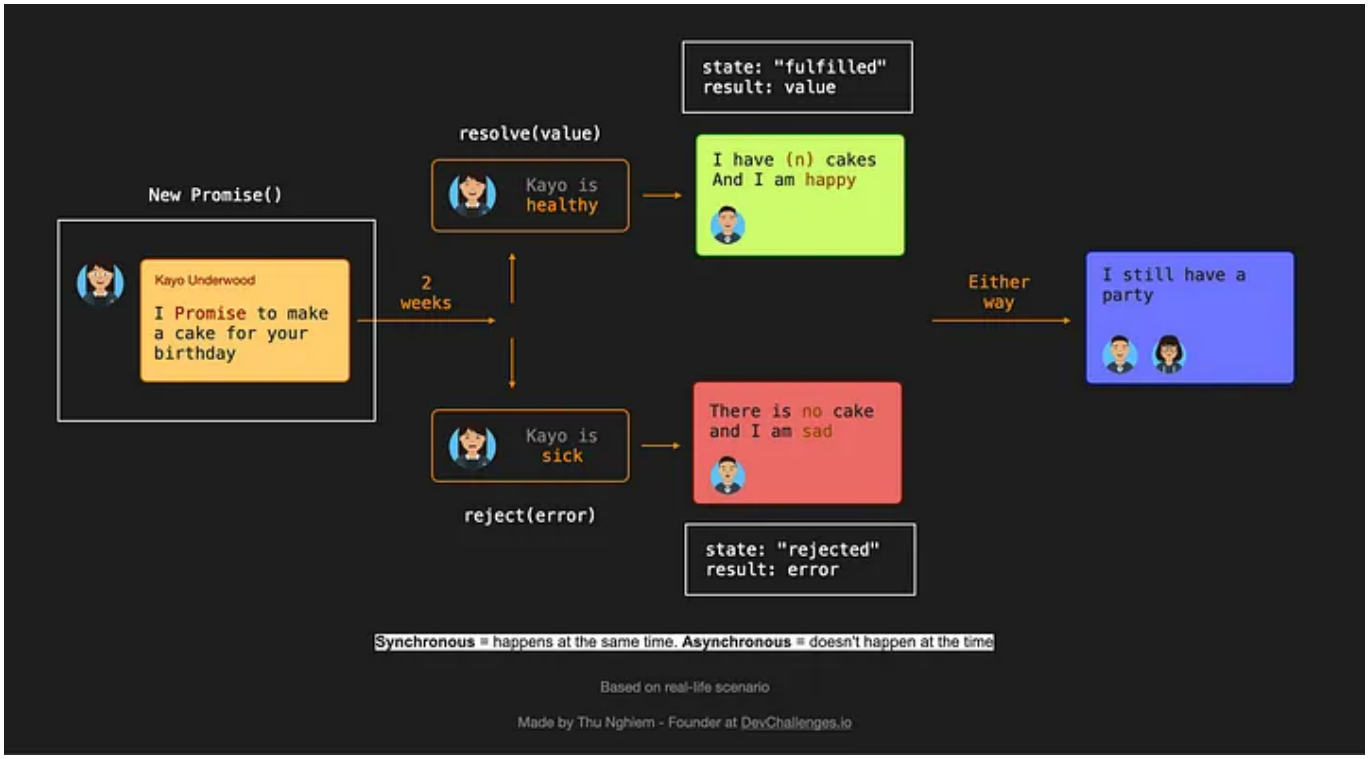


Chart from freeCodeCamp by Thu Nghiem

Async/Await:

Async/Await allows us to write asynchronous code in a more concise and readable way that looks and behaves like synchronous code. It's essentially syntactic sugar built on top of Promises.

In Async/Await, we use the `async` keyword to declare an asynchronous function and within that function, we can use the `await` keyword to pause execution until a Promise is resolved.

Async functions implicitly return promises:

```
Promise.resolve("Hello!");
```

```
// Same as:

async function greet() {
return "Hello!"
};
```

Example using Async/Await

```
// New Promise
const promiseFunction = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Promise resolved successfully");
      reject("Promise rejected with an error");
    }, 2000);
  });
};

// Async/Await
const asyncFunction = async () => {
  try {
    console.log("Start");
    const result = await promiseFunction();
    console.log(result);
    console.log("End");
  } catch (error) {
    console.log(error);
  }
};

asyncFunction();

// Output order:
// Start
// Promise resolved successfully
// End
```

Async/Await vs. Promises — Key differences:

Aspect	Promises	Async/Await
Scope	Only the promise chain itself is asynchronous	The entire wrapper function is asynchronous
Syntax	Uses `.then()` method for handling results	Uses `async` and `await` for a synchronous look
Readability	Chaining can lead to readability challenges	Code looks more synchronous, improving readability
Error Handling	Error handling often done with `.catch()`	Traditional try-catch blocks for intuitive error handling
Use Cases	Lower-level construct, suitable for fine-grained control	Higher-level abstraction for clean and readable code

So When to Use What!?

1. **Promises:** Promises are a great choice when we need better control over asynchronous operations, want to handle errors using `.catch()`, or when working with libraries and APIs that return Promises.
2. **Async/Await:** Async/Await is recommended for most scenarios because of its readability and simplicity. It's particularly useful when we have a sequence of asynchronous operations that depend on each other, as it allows us to write code that looks more like synchronous code.

Summary:

While both Promises and Async/Await are essential for managing asynchronous operations, the primary difference between them is in code readability and maintainability.

In terms of performance, there isn't a significant difference between Promise chains and Async/Await when it comes to the underlying execution

of asynchronous code. Both techniques are built on top of Promises, so they have similar performance characteristics.

Therefore, when deciding between Promise chains and Async/Await, choose Promises when you need more control and flexibility, and opt for Async/Await for cleaner and more readable asynchronous code, especially for sequential operations.

Useful Links:

- MDN: Promise
- Wikipedia: Futures and promises
- JavaScript Promises In 10 Minutes
- Promises Part 1 — Topics of JavaScript/ES6
- JavaScript Promises — Explain Like I'm Five
- History of Promises in JavaScript
- MDN: Synchronous
- MDN: Asynchronous
- MDN: Asynchronous JavaScript
- Async/Await in Node.js

JavaScript | Asynchronous Programming | Promises In Javascript | Asyncawait

Written by Jaqueline Nacarate

2 Followers

Follow

Recommended from Medium

Sandeep Agrawal

Mastering JavaScript Object Methods: A Comprehensive Guid...

JavaScript objects are fundamental constructs that allow developers to store and manipulat...

10 min read · 5 days ago

1

Evelyn Taylor

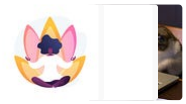
🚫 ⚠️ 5 React useState Mistakes That Can Put Your Job at Risk:...

Hey, React developers! We all love the power and simplicity of the useState hook, but let's...

4 min read · Jul 23

378 6

Lists



Stories to Help You Grow as a Software Developer

19 stories · 384 saves

It's never too late or early to start something

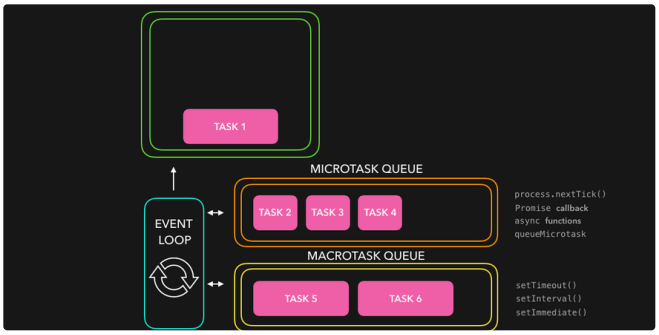
15 stories · 128 saves

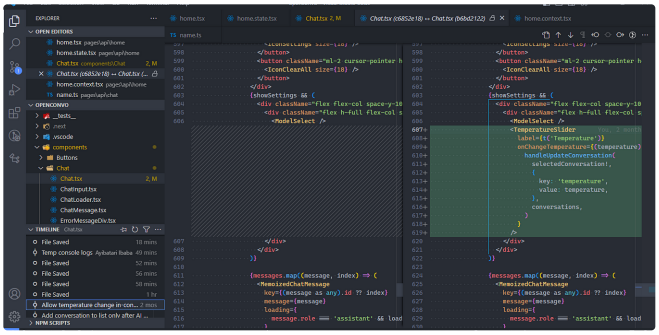
General Coding Knowledge

20 stories · 356 saves

Coding & Development

11 stories · 183 saves





Jeswanth Reddy in Version 1

Difference Between Promise and Async/Await

If you're reading this, you probably understand how the promise and async/awai...

2 min read · May 12

1.5K 13

Coding Beauty in Dev Genius

10 essential VS Code tips & tricks for greater productivity

Boost your productivity with VS Code: discover key features to enhance your codin...

10 min read · Aug 20

2.4K 23

Shubham Gupta

React and JavaScript Interview Questions

Satyam Verma

20 JavaScript Tips and Tricks You Can Use Right Now

The following question were compiled from various interview , if t here are other questio...

13 min read · Sep 11

39 1

Photo by Gabriel Heinzer on Unsplash

4 min read · Aug 10

277 2

See more recommendations