Callbacks vs. Promises vs. Async/Await: Detailed Comparison

Karnika Gupta · Follow

Published in Women in Technology · 4 min read · Feb 23, 2024

Introduction

Callbacks, promises and async/await, these are the methods to handle asynchronous behaviour in javascript. We need asynchronous programming for fetching data from the server, uploading files and handling user interactions. Initially, we used callbacks but it had consequences like deep nesting and code complexity. To tackle this scenario, concepts like promises

and async/await were introduced. These concepts helped in providing readable and clean code. Now let us dive deep into how we can compare them based on their usage.

Callbacks

Callbacks are those functions which are passed as arguments to another function and are executed when a particular task is completed.

```
function fetchData(callback) {
    setTimeout(() => {
        callback('Data fetched');
    }, 1000);
}

fetchData((result) => {
    console.log(result); // Output: Data fetched
});
```

Here in the above example, we have defined a `fetchData()` function which has a timeout of 1 second. That means the result will be rendered after 1 second.

Callback hell problem: It creates a problem when we have multiple asynchronous operations. There it forms a nested structure which becomes complicated and hard to read code.

Advantages of Callbacks

- They are very simple to use and are widely supported.
- Efficient when working with simple asynchronous operations.

Disadvantages of Callbacks

- Quite complicated when dealing with multiple asynchronous operations which leads to callback hell.
- Error handling is challenging task as the code becomes complicated and hard to understand.

Promises

Promises is a representation of completion or failure of any asynchronous operation. It allows chaining of multiple asynchronous operations. There are basically 3 states in promises i.e., resolve, pending and reject.

```
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Data fetched');
        }, 1000);
    });
}

fetchData()
    .then((result) => {
        console.log(result); // Output: Data fetched
    })
    .catch((error) => {
        console.error(error);
    });
```

In the above code, we have a `fetchData()` function which returns a promise. If the promise is resolved, the result will be displayed and if it is rejected then the catch block will be executed which will display the error.

Chaining Promise:

```
fetchData()
    .then((result) => {
        return processData(result);
    })
    .then((processedData) => {
        console.log(processedData);
    })
    .catch((error) => {
        console.error(error);
    });
```

In the above code, `fetchData()` function is either resolved or rejected. If the request is resolved, `.then` will be executed step-by-step. But is the request is rejected then the chain will be executed till it reaches the `.catch` block to display the error.

Advantages of Promises

- Promises solves the main problem of callback hell by providing chaining. This makes code more readable and clean.
- Error handling is improved with the help of promises as we can use `.catch()` for error handling in promises.

Disadvantages of Promises

- It requires deep understanding of Promises API as it includes multiple properties and methods.

Async/Await

Async/await is a feature that is built on top of promises to make it better and efficient. It is more concise and provides a synchronous-like way to write asynchronous programs. await keyword is always used inside the async function scope.

```
async function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Data fetched');
        }, 1000);
    });
}

async function getData() {
    try {
        const result = await fetchData();
        console.log(result); // Output: Data fetched
    } catch (error) {
        console.error(error);
    }
}

getData();
```

In the above code, `fetchData()` function returns a promise. `getData()` is a async function which contains a `try catch block` . We have result which waits for the `fetchData()` function to get the result. If the promise is resolved then the result will be displayed otherwise catch block will be executed.

Advantages of Async/Await

- It is much more readable as compared to promises and callbacks. It is much alike synchronous code which is easier to understand.

- It is built on top of the promises which provides compatibility between the two.

Disadvantages of Async/Await

- It has limited support in the older versions.

Comparison

- **Performance:** As we compare, promises and async/await is slightly overhead as compared to callbacks because of the abstraction layer. But the difference is as good as null.
- **Readability and Maintainability:** Async/await provides the most readable and maintainable code and followed by promises but callbacks leads to callback hell.
- **Error Handling:** Async/await provides the best error handling syntax which provides much cleaner code. It is followed by promises and callbacks are mostly prone to errors.
- **Sequential vs. Parallel Operations:** All the three approaches provides both the sequential and parallel operations. Async/await and promises provides clean and maintainable syntax for sequential operations.

Best Practices and Use Cases

- **When to Use Callbacks:** Use callbacks either for simple and easy asynchronous operations or when handling those APIs that only support callbacks.
- **When to Use Promises:** Promises are well suitable for handling multiple asynchronous operations in a more structured manner using `.then`.

- **When to Use Async/Await:** Use Async/Await for writing clean and much readable asynchronous code, especially for complex tasks involving multiple asynchronous operations.

Migration and Adoption

- **Migrating from Callbacks to Promises**: Rewrite callback-based code to use Promises for improved readability and error handling.
- **Migrating from Promises to Async/Await:** Convert Promises to Async/Await for even cleaner and more synchronous-like code.

Conclusion

Callbacks, Promises, and Async/Await provides us different approaches towards handling asynchronous operations in JavaScript. While callbacks are simple but prone to callback hell, Promises and Async/Await provide cleaner and more readable code structures. Choosing the right approach depends on the specific requirements and complexity of the asynchronous tasks.

JavaScript | Promises | React | Asynchronous Programming | Best Practices