Gmail Bikash Adhikari <virtual.bikash@gmail.com>

# Async-Await and Promise all in JavaScript

1 message

**Bikash** <bikash.caim@gmail.com> Sat, Mar 8, 2025 at 8:27 AM
To: virtual.bikash@gmail.com

Let's break down `async/await` and `Promise.all` in JavaScript, explaining what they are, how they work together, and when to use them.

## Async/Await

- **Purpose:** `async/await` is syntactic sugar built on top of promises. It makes asynchronous code look and behave more like synchronous code, making it easier to read and write.
- `async` **Function:**
  - The `async` keyword is placed before a function declaration.
  - It automatically turns the function into an asynchronous function, which implicitly returns a promise.
  - It allows you to use the `await` keyword inside the function.
- `await` **Keyword:**
  - The `await` keyword is used inside an `async` function.
  - It pauses the execution of the `async` function until the promise it precedes is resolved.
  - It returns the resolved value of the promise.
  - If the promise is rejected, it throws an error.
- **Error Handling:**
  - You handle errors in `async/await` using `try...catch` blocks, just like in synchronous code.

## Example:

JavaScript

```javascript
async function fetchData() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
    return data;
  } catch (error) {
    console.error('Error fetching data:', error);
    throw error; //rethrow the error so that calling functions can also handle it.
  }
}

fetchData();
```

## Promise.all()

- **Purpose:** `Promise.all()` takes an array of promises as input and returns a single promise.
- **Resolution:**
  - The returned promise resolves when all of the input promises have resolved.
  - The resolved value is an array containing the resolved values of the input promises, in the same order.
- **Rejection:**
  - The returned promise rejects as soon as any of the input promises reject.
  - The rejection reason is the reason of the first rejected promise.
- **Use Case:** `Promise.all()` is useful when you need to perform multiple asynchronous operations concurrently and wait for all of them to complete before proceeding.[1]

**Example:**

JavaScript

```
const promise1 = Promise.resolve(1);
const promise2 = Promise.resolve(2);
const promise3 = Promise.resolve(3);

Promise.all([promise1, promise2, promise3])
  .then((values) => {
    console.log(values); // Output: [1, 2, 3]
  })
  .catch((error) => {
    console.error('An error occurred:', error);
  });

//Example with fetch.
async function fetchMultipleUrls(urls){
  try{
    const promises = urls.map(url => fetch(url).then(response => response.json()));
    const results = await Promise.all(promises);
    console.log(results);
    return results;
  } catch(error){
    console.error("Error fetching multiple urls", error);
  }
}

const urlArray = ['https://api.example.com/data1', 'https://api.example.com/data2'];
fetchMultipleUrls(urlArray);
```

### Combining Async/Await and Promise.all()

You can effectively combine `async/await` and `Promise.all()` to perform multiple asynchronous operations concurrently within an `async` function.

### Example:

JavaScript

```javascript
async function processData() {
  try {
    const results = await Promise.all([
      fetch('https://api.example.com/data1').then(res => res.json()),
      fetch('https://api.example.com/data2').then(res => res.json()),
    ]);
    console.log('All data fetched:', results);
    // Process the results...
  } catch (error) {
    console.error('Error processing data:', error);
  }
}

processData();
```

### Key Points:

- `async/await` simplifies asynchronous code by making it look synchronous.
- `Promise.all()` allows you to execute multiple promises concurrently.
- Use `try...catch` for error handling in `async/await` functions.
- `Promise.all()` rejects as soon as any of the input promises reject, so it's important to handle potential errors.

- When you have multiple independent async operations that must all complete before moving forward, `Promise.all()` is the correct tool.

**Sources**
1. https://github.com/Vets-Who-Code/web-curriculum
2. https://github.com/Aditi571/TV_show
3. https://github.com/nishantmendiratta/javascript360
4. https://juejin.cn/post/7330143281525571638
5. https://github.com/niranjan-sharma-s/react-hooks
6. https://github.com/adarshdhital007/TypeScript