6 async/await Use Cases In JavaScript

@new_javascript

1. Fetching Data from an API

```
async function fetchData() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error("Failed to load data", error);
  }
}
```

Retrieve data from a server and await its arrival before processing.

2. Reading Files in Node.js

```
const fs = require('fs').promises;

async function readFile() {
  try {
    const data = await fs.readFile('example.txt', 'utf8');
    console.log(data);
  } catch (error) {
    console.error("Failed to read file", error);
  }
}
```

Read a file asynchronously without blocking, handling the result as soon as it's available.

3. Chaining Asynchronous Operations

```
async function processDataFlow() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    const processedData = await processData(data);
    console.log(processedData);
  } catch (error) {
    console.error("Failed in processing", error);
  }
}

```

Perform a series of dependent asynchronous operations, handling each in turn.

4. Image Preloading

```
async function loadImage(url) {
  return new Promise((resolve, reject) => {
    const image = new Image();
    image.onload = () => resolve(image);
    image.onerror = () => reject(new Error('Failed to load image'));
    image.src = url;
  });
}

async function useImage(url) {
  try {
    const image = await loadImage(url);
    document.body.appendChild(image);
  } catch (error) {
    console.error(error.message);
  }
}
```

Load an image and use it in the document once fully loaded.

5. Database Operations in Node.js

```
const { Pool } = require('pg');
const pool = new Pool();

async function queryDatabase() {
  try {
    const res = await pool.query('SELECT * FROM users WHERE id = $1', [1]);
    console.log(res.rows[0]);
  } catch (error) {
    console.error('Query error', error.stack);
  }
}
```

Perform and await the result of a database query.

6. Using async with setTimeout

```
function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function runWithDelay() {
  console.log("Start");
  await delay(1000);
  console.log("Delayed for 1
```

Use Case: Perform and await the result of a database query.

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like Comment Repost