Optimizing JavaScript Execution

Steve Buchanan
DevOps Architect

@buchatech | www.buchatech.com

Overview

Understanding JavaScript Synchronous Code vs. Asynchronous Code for Better Performance

Improving JavaScript Performance, Responsiveness, and Reducing Timeouts through Various Optimizations

Understanding JavaScript Synchronous Code vs. Asynchronous Code for Better Performance

Transition from Synchronous Code to Asynchronous

Understand the Differences

Synchronous Code:

Executes line by line, waiting for each operation to complete before moving to the next

Asynchronous Code:

Allows other operations to continue before the current operation completes,enhancing performance, especially in I/O operations

Identify Synchronous Operations

Look for parts of your code that are blocking or where performance can be improved

Network requests (HTTP calls)

File system operations

Database queries

Long computations

Better Handling of Asynchronous Operations

Convert Synchronous Functions to Promises:

A Promise in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous operation & its resulting value.

Promises are a powerful feature in JavaScript for handling asynchronous operations.

Promises provide a cleaner, more manageable way to deal with operations that take time to complete.

Refactor Your Code

Refactor your synchronous code to use promises and async/await.

Synchronous Code:

```javascript
function processData() {
    const data = fetchData();
    console.log(data);
}
```

->

Asynchronous Code:

```javascript
async function processData() {
    try {
        const data = await new Promise((resolve, reject)
=> {
            // An asynchronous operation without delay
            const success = true; // Change this to false
to simulate a failure
            if (success) {
                resolve('Operation succeeded!');
            } else {
                reject('Operation failed.');
            }
        });
        console.log(data);
    } catch (error) {
        console.error('Error processing data:', error);
    }
}
processData();
```

Improving JavaScript Performance, Responsiveness, and Reducing Timeouts through Various Optimizations

Improving JavaScript Responsiveness

Throttling ensures a function is called at most once in a specified time period

Useful for handling events like window resizing, scrolling, and input field changes

Control the rate at which functions are executed

Debouncing delays the execution of a function until a certain amount of time has passed since the last time it was invoked

Minimizes the time the main thread is blocked, allowing JavaScript web apps to handle other critical tasks like rendering & user interactions more efficiently

Achieved by optimizing loop logic, minimizing DOM manipulations, and using efficient data structures

Reducing Timeouts in JavaScript

Caching Objects

- What It Is:
  - Involves <u>storing frequently accessed data</u> in a temporary storage location so that future requests for that data can be served faster.
- How It Helps:
  - Reduces Load Times: By storing data in memory or a local cache, the need to fetch the same data repeatedly from a remote server is eliminated. This <u>decreases the time spent waiting for network requests</u>.
  - Improves Performance: Accessing cached data is much faster than retrieving it from a database or server, which <u>can prevent your JavaScript code from waiting too long</u> for these operations to complete.

Limiting Library Dependencies

- What It Is:
  - Involves <u>minimizing the number of external libraries</u> your JavaScript code relies on, using only essential ones.
- How It Helps:
  - Decreases Load Time: Fewer dependencies mean <u>fewer files</u> to download and execute, leading to <u>quicker loading times</u>.
  - Reduces Code Complexity: Simplifying your codebase by limiting dependencies can also make it <u>run faster & be easier to maintain</u>, reducing the chances of performance bottlenecks.

Reducing Timeouts in JavaScript

Delaying JavaScript Loading

- What It Is:
  - Can be done by deferring or asynchronously loading scripts. This ensures that the most critical parts of your webpage load first.
- How It Helps:
  - Improves Initial Page Load: By loading essential content before JavaScript, users see the main content sooner, which enhances the perceived performance.
  - Prevents Blocking: Deferred or asynchronous scripts don't block the rendering of the page, allowing other resources to load without waiting for JavaScript to finish.

Using File Compression

- What It Is:
  - Reduces the size of files that are sent over the network. Common methods include gzip and Brotli compression.
- How It Helps:
  - Faster Downloads: Smaller files take less time to download, reducing the overall time required to load your JavaScript files.
  - Reduced Bandwidth Usage: Compressed files consume less bandwidth, which can be particularly beneficial for users with slow internet connections or limited data plans.

Demo

Demo: Caching Objects to Reduce Timeouts in JavaScript

Summary

In this module we covered:

- ❑ Understanding JavaScript Synchronous Code vs. Asynchronous Code for Better Performance
- ❑ Improving JavaScript Performance, Responsiveness, and Reducing Timeouts through Various Optimizations

Why this is important:?

- ❑ Moving to async code is a good tool to have in your tool belt when building JavaScript web apps. Having some techniques to improve responsiveness and reducing timeouts of a web app can make the difference between a web app that's useful and one that's not.