MAY 30, 2023 / #JAVASCRIPT

How Does JavaScript Work Behind the Scenes? JS Engine and Runtime Explained

Esther Christopher

So you may know that your code somehow compiles and executes in your browser to

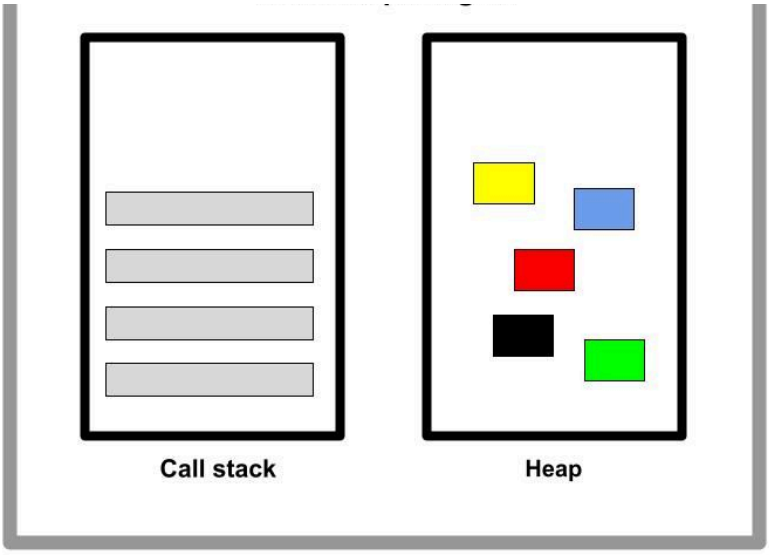
components that come into play to enable the output?

Let's dive a little into JavaScript behind the scenes. The abstract part that you can't exactly see.

Why should a seemingly abstract subject matter to you? An understanding of the inner workings of JavaScript allows you to explore the language beyond the surface level and from a deeper perspective.

It provides contextual information on the language and how the JavaScript engine optimizes code. This will give you some important foundational knowledge which shapes the way you write code. It also helps you write more efficient, scalable, and maintainable code.

The JavaScript Engine



JavaScript Engine showing the Call stack and the Heap

The JavaScript engine is simply a computer program that interprets JavaScript code. The engine is responsible for executing the code.

Every major browser has a JavaScript engine that executes JavaScript code. The most popular one is the Google Chrome V8 engine. Google's V8 powers Google Chrome and Node.js, a back-end JavaScript runtime environment used to build server-side applications.

Other major browser engines include:

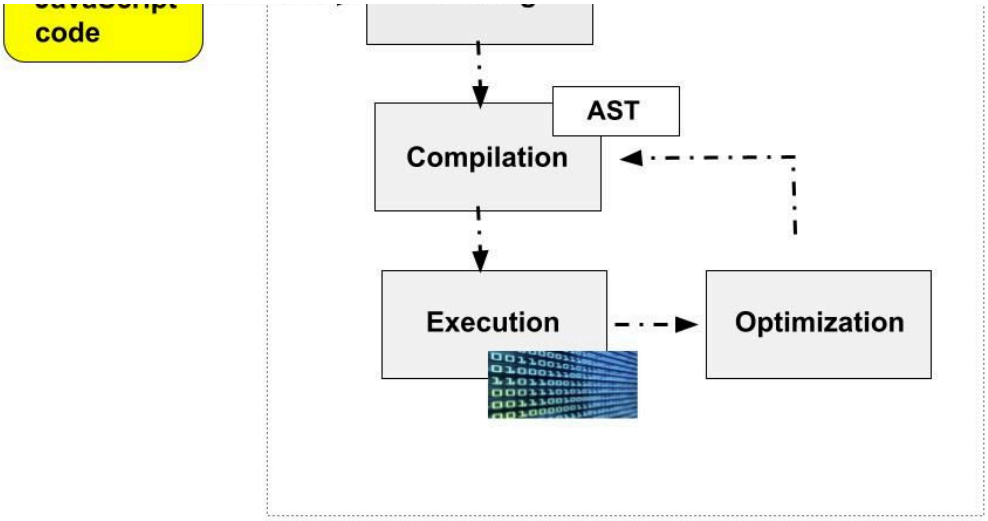
- JavaScriptCore which powers the Safari browser
- Chakra which powers Internet Explorer

Any JavaScript engine typically contains a call stack and a heap. The call stack is where the code is executed. The heap is an unstructured memory pool that stores all the objects needed for the application.

Since the computer's processor only understands binary, 0's and 1's, the code has to be translated to 0's and 1's.

When a code snippet passes into the engine, the code is initially parsed, that is read. The code is subsequently parsed to a data structure called the abstract syntax tree (AST). The resulting tree is then used to create machine codes.

Execution happens in the JavaScript engine call stack using the execution context. This is the environment where JavaScript code is executed.

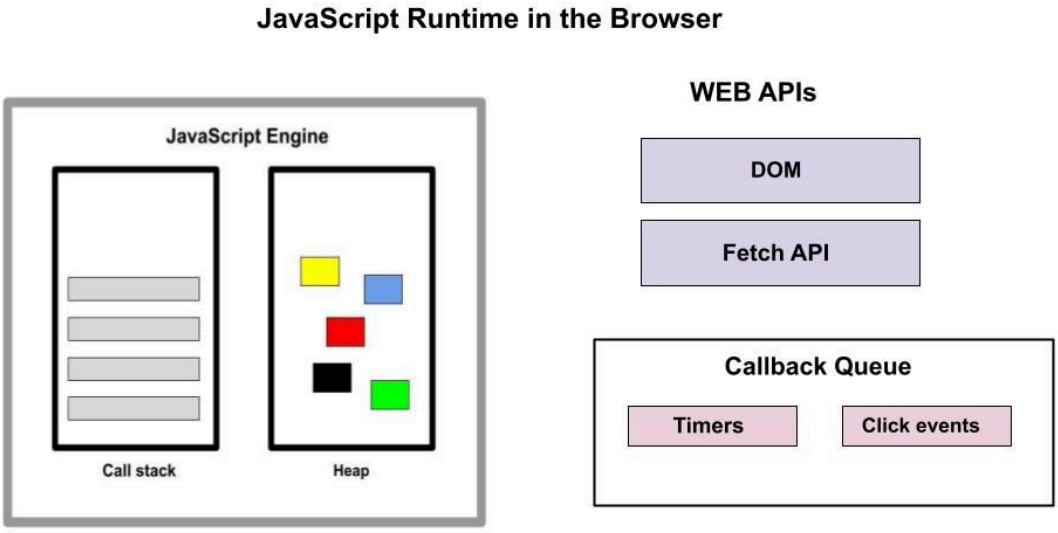


A diagram illustration showing the JavaScript execution process

The JavaScript Runtime

Think of the JavaScript runtime as the house that encompasses all the components needed to run JavaScript. This house comprises the JavaScript engine, Web APIs, and the callback queue.

Web APIs are functionalities that are provided to the engine but are not part of the JavaScript language. They are accessible to the engine through the browser and help access data or enhance browser functionality. Examples are the Document Object Model (DOM) and Fetch APIs.



A diagram of JavaScript Runtime in the Browser containing the JavaScript Engine, WEB APIs, and the Callback Queue

The callback queue includes callback functions that are ready to be executed. The callback queue ensures that callbacks are executed in the First-In-First-Out (FIFO) method and they get passed into the stack when it's empty.

The browser runtime and Node.js are examples of runtime environments.

When JavaScript executes within a web browser it is operating within the browser's runtime environment.

elements, handling events, and manipulating the page structure.

Node.js provides a server-side runtime environment for executing JavaScript outside the browser. Because it executes JavaScript outside the browser, it does not have access to the web APIs. Instead, the Node.js runtime environment replaces it with something called C++ bindings and the thread pool.

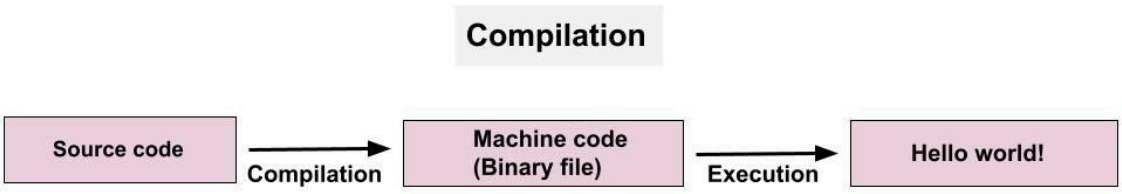
JavaScript Optimization Strategies

Modern JavaScript engines have some optimization strategies put in place to enhance the performance of code execution. These optimizations occur dynamically during the execution process. Let's look at some of these strategies.

Just-in-Time compilation

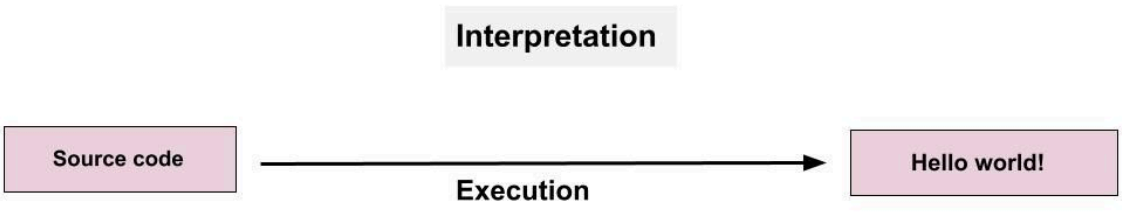
The process that involves the translation of JavaScript code into machine code occurs using compilation and interpretation.

file to be executed by the computer.



A diagram showing the code compilation process

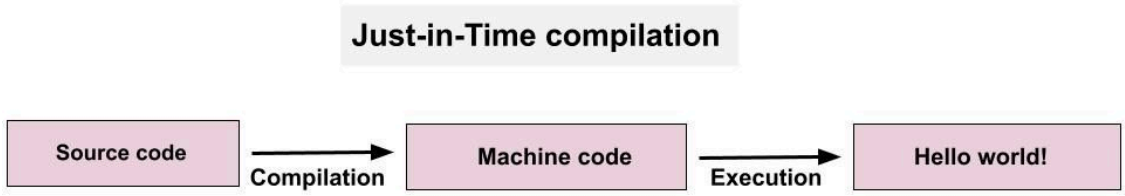
In contrast, during interpretation, the interpreter goes through the source code and interprets it line by line, executing each line as it encounters it.



A diagram showing the code interpretation process

compiled languages.

In order to optimize the performance of web applications, JavaScript combines both compilation and interpretation. This is called Just-in-Time compilation. This method compiles the entire code into machine code all at once and executes it.



A diagram showing Just-in-Time compilation of code

Just-in-Time compilation involves the same two processes as regular compilation, but here the machine code isn't written into a binary file. The code is also executed right away after compilation.

This has had a significant impact on the speed of code execution in JavaScript. So hopefully this helps dispel

To fully optimize JavaScript code, the engine first creates an unoptimized version of the machine code so it can start executing immediately. While that is ongoing, the code is being re-optimized and recompiled in the background of the currently running program execution. This is done multiple times to produce the final, most optimized version.

The process of parsing, compilation, and execution happens in some special thread in the engine that can't be accessed from the code.

What is Inlining?

Inlining is another optimization technique JavaScript employs to improve performance and speed.

```
function add(a, b) {
  return a + b;
}

let result = 0;
result = result + 5;
result = result + 3;

console.log(result); //
```

directly called. Instead, the code inside the function `return a + b;` is inserted at the call site.

This optimization is done especially for functions that are called repeatedly. The JavaScript engine will run the function as it normally would. But as the function gets called often, the engine replaces the function call with the actual code of the function at the call site. This helps to prevent several function calls and improve performance.

Performance Considerations

Several factors affect the performance of your web application. As the JavaScript engine employs some strategies to ensure optimization, there are also some best practices to be taken into consideration by developers for efficient execution.

Techniques such as minimizing DOM manipulation and reducing function calls enhance code performance.

Frequent access and interaction to the DOM slows down the rendering of web pages and contributes to

interaction by batching DOM updates to reduce overhead.

Additionally, reducing function calls takes up the performance a notch. By reducing function calls, you streamline your code and make it more efficient making your JavaScript applications faster and more responsive.

```
// Inefficient code with unnecessary function calls
function calculateTotal(a, b, c) {
  return addNumbers(a, b) + multiplyNumbers(c, b);
}
function addNumbers(x, y) {
  return x + y;
}
function multiplyNumbers(x, y) {
  return x * y;
}
// Improved code with reduced function calls
function calculateTotal(a, b, c) {
  const sum = a + b;
  return sum + c * b;
}
console.log(calculateTotal(2, 3, 4)); // Output: 23
```

In the inefficient code, the `calculateTotal()` function makes separate function calls to `addNumbers()` and

In the improved code, the function calls are reduced by directly performing the addition and multiplication operations within the `calculateTotal()` function. By reducing function calls, the code becomes more efficient and improves execution speed.

Future JavaScript Developments and Trends

There will continue to be improvements and advancements in JavaScript engines and runtime environments. These changes are geared towards improving the performance of web applications.

One such advancement is the rise of WebAssembly. WebAssembly brings near-native performance to web applications and supports multiple languages. It opens up new possibilities for performance optimization and execution speed.

It is important for JavaScript developers to stay updated with these trends and adapt new coding best practices accordingly.

So many processes are involved in how your JavaScript code is being parsed until it renders a functional web application.

This article provides a high-level overview of the main concepts. It explains how the JavaScript engine executes code, the runtime, and its components. It also goes on to explain optimization strategies and highlight performance considerations.

Understanding how JavaScript operates behind the scenes shapes the way developers approach problems and write more efficient codes. It also helps them stay ahead of the learning curve and adapt easily to future changes in JavaScript features.

For more in-depth learning, you can visit these resources:

- Execution context
- Event loop
- Microtask queueing

If you enjoyed reading this article, then connect with me on Twitter and LinkedIn where I share my

Esther Christopher

I am a Front-end Developer and a Technical Writer

If you read this far, thank the author to show them you care. Say Thanks

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers. Get started

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Trending Guides

- Date Formatting in JS
- Cancel a Merge in Git
- Install Java in Ubuntu
- Full Stack Career Guide
- Smart Quotes Copy/Paste
- Sets in Python
- SQL Temp Table
- Comments in YAML
- Python End Program
- Python Dict Has Key
- Exit Function in Python
- Python Import from File
- Python Merge Dictionaries
- Reactive Programming Guide
- What's a Greedy Algorithm?
- Java Iterator Hashmap
- What is a Linked List?
- Python Ternary Operator
- Python Sort Dict by Key
- JavaScript Array Length
- Kotlin vs Java
- HTML Form Basics
- Pandas Count Rows
- Python XOR Operator
- Python List to String
- String to Array in Java
- Parse a String in Python
- Copy a Directory in Linux
- Center Text Vertically CSS
- Edit Commit Messages in Git

Mobile App

Our Charity