Understand the basics of a JavaScript Engine

Introduction

A JavaScript engine is a computer program that executes JavaScript code. It follows the ECMAScript standard and how the language should work and what features it should have. JavaScript engines are just programs that convert JavaScript code into lower level code, so the computer can understand. A JavaScript engine can be implemented as a standard interpreter, or just-in-time compiler that compiles JavaScript to bytecode in some form. The first JavaScript engines were almost only interpreters, but most modern engines employ just-in-time (JIT) compilation for upgraded performance.

Popular JavaScript Engines

All popular browsers have their own implementation of a JavaScript engine. Here are some popular JavaScript engines.

a), Chrome's V8 engine

b), Firefox's SpiderMonkey

c), Safari's JavaScriptCore (a.k.a Nitro, SquirrelFish and SquirrelFish Extreme)

d), Edge's Chakra — but Edge has recently embraced Chromium's V8 engine

The Engine

There are many engines in the market. Each engine has some kind of pipeline with an interpreter and an optimizer compiler pipeline. The interpreter generates bytecode and the optimizer generates highly optimized machine code. In these article I will use the V8 engine as an example.

Inside the Engine ----The lifecycle of browser code

Whenever a programmer sends JavaScript code to the V8 engine, it follows some steps in order to display your "console.log".

A) Parse

The engine makes what we call a Lexical Analysis. We give a JavaScript file to the engine and it first does a lexical analysis. This breaks the code into something called tokens to identify their meaning. After that, we know what the code is trying to do.

B)Abstract Syntax Tree (AST)

Tokens are formed into what we call AST. A syntax tree is a tree representation of the syntactic structure of the JavaScript code and we can use this tool to analyse the AST code conversion.If you are interested in reading more about AST, check this article.

C) Interpreter

The interpreter reads and translates the JavaScript files line by line on the fly (during the running of a computer program without interrupting the run). Based on the generated AST code, the interpreter starts to produce unoptimized bytecode quickly.

Bytecode is not as low level as machine code, but it’s code that is able to be interpreted by the javascript engine in order to run code.

D) Profiler

The profiler is responsible to check our code. It also calls a monitor that monitors and watches our code as it runs and makes notes on how we could optimize this code. Ex: How many times it is being run? What types are used? How we can optimize this code?

So, if the profiler finds some code that can be optimized, it passes this code to the Just-In-Time (JIT) compiler, so it can be compiled and runs faster. We will see these interpreter/compiler pipeline with more detail.

E) Optimizing compiler

The optimizing compiler’s job is to figure out what your input program does, and create an output program that it knows will do the same thing, just faster.

It doesn't translate the files on the fly. It works ahead of the time to create a translation of what code we’ve just written and it compiles down to usually a language that can be understood by our machines.

F) Deoptimize

The optimizing compiler makes certain assumptions based on the profiling data it has, and then produces highly-optimized machine code. But it's possible that at some point, one of the assumptions turns out to be incorrect. The optimizing compiler can deoptimize code.

Optimizing property access

Accessing properties requires several instructions in dynamic languages like JavaScript, so almost every engine has some optimization on that. That optimization in V8 is called hidden classes and we can say that V8 attaches a hidden class to each single object. The purpose of the hidden classes is to optimize property access time.

Hidden classes work similarly to the fixed object layouts (classes) used in languages like Java, except they are created at runtime. The magic here is that objects can share hidden classes, so we need only minimal resources and your code becomes faster.

The benefit becomes clear when we have multiple objects. No matter how many objects there are, as long as they have the same hidden class, we only have to store the information once! This topic about optimization is very extensive and has a different name for it. We can find more information about that here.

References:

https://blog.bitsrc.io/javascript-engines-an-overview-2162bffa1187

https://mathiasbynens.be/notes/shapes-ics

https://levelup.gitconnected.com/inside-the-javascript-engine-896c64cb7623

https://medium.com/nerd-for-tech/crucial-facts-about-javascript-engine-7b264c17f36d

https://softwareengineeringdaily.com/2018/10/03/javascript-and-the-inner-workings-of-your-br
owser/