JavaScript Engine Optimization JavaScript code

JavaScript Engine

A JavaScript Engine is a computer program that executes a JavaScript code. A JavaScript Engine interprets high level language such as JavaScript into a language a computer understands, bytecode. These engines run embedded in web browsers and servers. Among the commonly available JavaScript Engines are V8 that runs in Google Chrome and SpiderMonkey that runs in Firefox.

JavaScript Engine Pipeline

JavaScript Engine performs compiler optimization, hot code management, caching, garbage collection etc. Whenever a JavaScript is being loaded, the **parser** inside the Engine parses and breaks the code into tokens to creates an Abstract Syntax tree (AST). Next, the **interpreter** generates bytecode one by one from the AST. The **profiler** checks if the code needs optimization, and it passes the code to the compiler. The optimizing **compiler** changes the code into an output that perform faster.

JavaScript's Object Model

According to ECMAScript standard, all objects are defined as dictionaries with key string mapping to property attributes. The property attribute contains:

- **[[Enumerable]]** -> determines whether the property shows up in for-in loops
- **[[Value]]** -> the value itself
- **[[Writable]]** -> determines whether the property can be reassigned to
- **[[Configurable]]** -> determines whether the property can be deleted

Optimization Using Shapes/Hidden Class

JavaScript Engine utilize hidden class or shapes to make accessing property much faster. If multiple objects with the same properties are in a JavaScript code, they will share the same shape in the engine, i.e., all the objects point to the same shape. Thus, instead of having multiple property attribute for each object, the compiler optimizes using one shape that represent all the objects with the same property.

Hidden class or shapes depend on the sequence of the properties in an object, only objects with the same sequence will get mapped to the same shape. In addition, shapes depend on morphism of the object. The complier optimizes monomorphic objects by mapping them to the same shape. In polymorphic objects the compiler creates few shapes while in megamorphic objects creates a lot of shapes with no optimization.

If an object with a certain property is being updated, the shape for that object forms transition chains. The new added property gets a property information assigned to it, so the engine transition the shape to a new one, and upon every addition of property it continues transitioning the shape.

Another way the JS engine optimizes is when two or more empty objects add different properties, the shape branches into multiple shapes. In cases when there are empty objects together with objects that have properties, the empty object will start with empty shape while the object with property will have shape that contains the property.

Knowing how the JS engine optimizes objects, it is recommended (Benedikt and Mathias) that developers should initialize objects in the same way. This will help to minimize the number of shapes formed and enhance efficient operation.

References

1. https://codeburst.io/javascript-compiler-optimization-techniques-only-for-experts-58d6f5f958ca
2. https://mathiasbynens.be/notes/shapes-ics
3. https://javascript.plainenglish.io/js-engine-and-optimization-dac1f7fcb87d
4. https://levelup.gitconnected.com/inside-the-javascript-engine-896c64cb7623