Michael Deitzel

110075

LAB 2

CS445 - Lab 2 Modern Web Browsers

Write a paper about how JS engine optimize JavaScript code.

Before we get into how the JavaScript engine optimizes JavaScript code lets first understand a few things about the JavaScript engine itself. A JavaScript engine is a computer program that executes JavaScript code. It follows ECMAScript standard, which lays out the features required. JavaScript is a high-level language. There are also what are known as low level languages. Low Level is code that the machine understands. High Level is code that only humans understand. In order for JavaScript code to run the computer must translate the High Level into the Low Level. Different engines might have different ways of processing and optimizing code, but the underlying architecture is basically the same. This general version is what we will focus on here. It starts with us and the JavaScript code we write which then gets passed to the Parser. The first block or section of the JavaScript engine is called the Parser. This is fed directly from the JS file itself. The parser will organize the code into an abstract syntax tree or more commonly known as AST. The code then gets passed to the Interpreter. The Interpreter does not work directly on the JS file. It works on the tree. It generates bytecode ran by the machine. The Interpreter reads code passed to it from the AST line by line. While the Interpreter can generate byte code it also communicates with a pipeline which optimizes code. This JavaScript pipeline includes a Profiler and a Compiler. The profiler is like the fast checkout at the grocery store it is always monitoring and watching code so that it can optimize it. The way it does this is by tracking how many times, for example, a function gets called. If a function is called a certain number of times it will be marked as “warm” and if it continues will be marked as “hot”. This is how the JS Engine recognizes what code is being used the most. The Profiler and Compiler work together to produce highly optimized machine code.

Now that we have a basic overview let’s look at it in more detail and maybe see some examples and also have some takeaways that we can implement today. There are three important terms or concepts that this section will focus on – Shapes, Transition Chains, and Inline Cache. The interpreter - once the code reaches here it is now running JS code. Along with the bytecode that gets fed into the JS pipeline there is also what is called “profiling data” that get collected, for example, when a function goes hot. Also the JavaScript engine implements a JS object model. Objects are basically dictionaries with string keys. String keys map to something the spec calls property attributes, as well as, the value itself. Take for example object = {a:7, b: 8}; In this case, the ‘a’ and ‘b’ are string keys in a dictionary. 7 and 8 are values within the property attribute – for.property. Attributes store whether the property is writable, numberable, and configurable. The way to access is by using object.getOwnProperty descripter API. Objects are separated into JS Object and Shape. Shapes occur when multiple objects have similar properties. JSObject a and JSObject b will store the values and Shape will store all of the other information that is shared between JSO a and JSO b. JavaScript optimises object property access based on these shapes.

Next, we have Transition Chains. Shapes inside JavaScript Engine form transition chains. An empty object will initially point to an empty shape that doesn’t have any property on it. Instead of creating new shapes JS Engine remembers the information about the new property. This is done by introducing a back link to the previous shape so we can walk the transition chain backwards until we find the shape that changes the property and then we know where to look into the object. Lastly, we have Inline Cache. JavaScript uses inline cache to memorize information about where to find properties on objects. When we memorize the shape, the next time we call the function we don’t need to get the property information. This gets rid of having to loop altogether making JavaScript significantly faster. Two main takeaways. First, always initialize your object in exactly the same way so they don’t end up having different shapes. Second, don’t mess with property attributes on array elements so that they can be stored and operated on efficiently.