Javascript Engines: How do they work? How do they achieve performance at scale.

Why do we care?

If you run javascript source code it is the engine thats runs it for you,

Whether in the browser, node, or on an IOT device.

JS engines have evolved over the last twenty two years (as of 2017) to run massive frameworks and enterprise level node servers.

How are JS engines so preformant today?

**Javascript Engines:**

**Browser:** *Chakra (Microsoft Edge), JavaScriptCore (Apple Safari), SpiderMonkey (Mozilla Firefox), V8 (Chrome)*

**Node:** *Chakra, V8, SpiderNode*

**Electron (Chromimum):** *V8*

**IoT (reduced performance + reduced memory size):** *Duktape, JerryScript*

*\*browser and node engines are fast but take quite a bit of memory*

Javascript is defined by ECMAScript, which is defined by the TC39 committee, who review changes and additions and formalize it as a standard, which engine’s take and implement to produce modern JS.

JS is dynamically typed

Wheras C++ is statically typed.

You can add and delete properties on javascript as you wish,

However when you give so little information to the compiler, they have a hard time generating machine code that is fast when they have no information, which is why in C++ you have to provide all this upfront.

C++ is statically type because it allows fast machine code to be generated on compilation.

But despite all this JS is still really fast even though its dynamically typed.

The trick that all modern JS engines use is called Just in Time compilation (JIT compilation)

We are not first compiling ahead of time, finished the compilation, then running the code, we are combining these two steps together.

We compile as we need it, gather information as we run it, then re compile it.

C++ compiles, generates an executable then runs that executable.

In JS compilation and execution happens at the same time.

Generate machine code during run time, not ahead of time (AOT)

What modern engines have is at least two compilers, one an optimizing compiler

Re compiles hot functions with type info from previous execution

Hot function: function that you are using a lot is considered worth speeding up.

It is compiled, run a few times, info is gathered about the types, the compiler says this function is “hot” lets optimize it.

De optimizes if the type has changed

When the optimizer compiler runs, it assumes that it will similar similar types as before

Since javascript is dynamically typed, it may happen that you run the optimizer on a function, and you need to de optimize it and run it with the non optimized compiler.

**The Order of Operations:**

1. Javascript Source Code
2. Parser
3. AST (Abstract Syntax Tree)
4. Baseline Compiler
   1. Uses the AST to produce machine code
   2. Machine code is repeatedly run collecting information, eventually passing it to the optimizing compiler to produce faster machine code.
   3. Once in awhile, the optimized code needs to be de-optimized as types changes in the source code

In V8 the Baseline compiler is an interpreter called Ignition, the optimizing compiler is called turbofan.

Optimizing compiler uses previously used type information, meaning that if types are constantly changing, code cannot be optimized.

Example:

Function load(obj) {

Return obj.x;

};

load({x: 4, y: 7});

load({x: 2, y: 9});

load({x: 3, y: 5});

load({x: 8, y: 1});

*\* as each object has the same order and property types, internally they are considered the same object type, therefore running the function a lot will cause the compiler to flag it as a hot function to be optimized*

*Once optimized, a comparison will be run whenever the object is called, if the comparison passes, the object's current structure remains valid and the optimized code continues to execute, if not, the compiler bails out and optimizes the code as the type can no longer be inferred from previous calls.*

*With up to four different types, the compiler will produce four references to the type and compare each,*

*Past 4, the compiler instead uses passes properties to a more expensive function to compare*

*\*optimization tip: always construct the same type of objects.*

*load({x: 4, a: 7, b: undefined})*

*load({x: undefined, a: undefined, b: 3});*

*Will run optimized wheras*

*load({x:4, a: 7})*

*load({b: 3});*

*Will not, as in the first example the two objects are considered the same type internally.*

“Object.x” is actually complicated for the compiler,

As if you have an object that the compiler doesn’t know anything about, it needs to determine:

Does this object have a property x, is it on the prototype chain, where in memory is the value stored, etc.

How are object represented internally?

Objects types are represented incrementally with transitions for each object property

|  |  |  |
| --- | --- | --- |
| Obj | Obj | Obj |
|  | X: number | X: number |
|  |  | Y: number |

*\*Order matters (example: if two objects each have x & y but in different orders, they will not be considered the same type internally)*

**Try the following:**

*-print-opt-code*: prints code generated by optimizing compiler

*-print-bytecode*: prints bytecode generated by interpreter

*-trace-ic*: different object types a call sites encounters

*-trace-opt* and t*race-depot*: which functions are de-optimized

**Statically Typed** languages are those in which the type of a variable is known at compile time

Examples: java, c, c++;

Example: int a; a = 5;

**Dynamically typed** languages are those in which the type of a variable is interpreted at runtime.

Example: python, ruby, javascript.

Example: a = 5;

**Strong Typing:**

Programming languages in which variables have specific data types.

Variables are bound to a particular data type.

Example: Java, Python.

Example: a = 9; b = “9”; c = concatenate(a,b); d = add(a, b);

Output: type mismatch exception on both the c assignment and the d assignment

String “9” and 9 and not interchangable

To remove error: c = concatenate(str(a), b); d = add(a, int(b));

**Weakly Typed:**

Programming languages in which variables are not of a specific data type

Variables are not bound to a specific data type

Example: C, PHP

Example: Example: a = 9; b = “9”; c = concatenate(a,b); d = add(a, b);

Output: 99; 18

Languages can be of either combination:

|  |  |  |
| --- | --- | --- |
|  | Strong | Weak |
| Static | Java | C |
| Dynamic | Python | PHP |