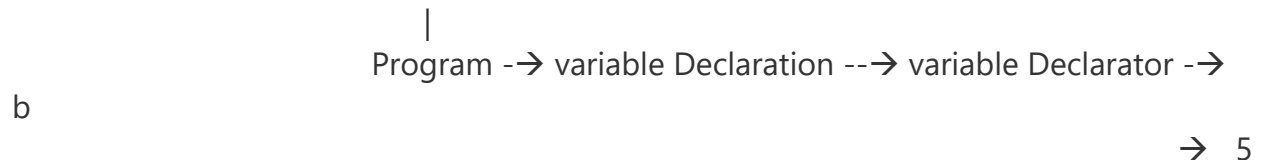


How JS engine optimizes JavaScript code

It starts with the JavaScript code you write, and the JavaScript parser takes the code and tokenizes it into source code and changes it into an abstract syntax tree (AST) and feeds it to the interpreter to generate bytecodes. And the bytecode is fed to the optimizing compiler and generates highly optimized code that runs more efficiently.

Example `let b = 5`. ---→ JavaScript AST visualizer



-Interpreter (V8 in Google Chrome) → generate bytecode

- Compiler (TurboFan in Google Chrome) → optimized machine code, to scan the code ahead of time and to find stubs.

Example2 :> <pre>function concatArray(arr){ var result = ""; for (let i=0; i<arr.length;i++){ result += arr[i] } return result; }</pre>	<p>think every step of the for loop, the interpreter is asking:</p> <ul style="list-style-type: none">Is <code>i</code> an integer?Is <code>result</code> a string?Is <code>arr</code> actually an array?Is <code>arr[i]</code> a string?
--	--

-The baseline compiler will turn the `result += arr[i]` line into a stub, but because this instruction is polymorphic (there is nothing ensuring every time that `arr[i]` is going to be a string for every position on the array) it'll create a stub for every possible combination.

- JIT profiler while the code is executed by the interpreter, a profiler will keep track of how many times the different statements get hit. The moment that number starts growing, it will mark it as warm and if it grows enough, it will mark it as hot. In other words, it will detect which part of your code is being used the most, then it will send them over to be compiled and stored.

Seven tips for improving performance by leveraging optimizations in V8

- Tip1: Declare object properties in constructor
- Tip2: Keep object property ordering constant
- Tip3: Fix function argument types
- Tip4: Declare classes in script scope

Tip5: Use for ...in

Tip6: Irrelevant characters do not affect performance

Tip7: Try/catch/finally is not ruinous

How to write optimized JavaScript

1. **Order of object properties:** always instantiate your object properties in the same order so that hidden classes, and subsequently optimized code, can be shared.
2. **Dynamic properties:** adding properties to an object after instantiation will force a hidden class change and slow down any methods that were optimized for the previous hidden class. Instead, assign all an object's properties in its constructor.
3. **Methods:** code that executes the same method repeatedly will run faster than code that executes many different methods only once (due to inline caching).
4. **Arrays:** avoid sparse arrays where keys are not incremental numbers. Sparse arrays which don't have every element inside them are a **hash table**. Elements in such arrays are more expensive to access. Also, try to avoid pre-allocating large arrays. It's better to grow as you go. Finally, don't delete elements in arrays. It makes the keys sparse.
5. **Tagged values:** V8 represents objects and numbers with 32 bits. It uses a bit to know if it is an object (flag = 1) or an integer (flag = 0) called SMI (SMall Integer) because of its 31 bits. Then, if a numeric value is bigger than 31 bits, V8 will box the number, turning it into a double and creating a new object to put the number inside. Try to use 31-bit signed numbers whenever possible to avoid the expensive boxing operation into a JS object.

