JavaScript report

Sasha Abramowitz, ABRSAS002

History

JavaScript (JS) was initially intended to be scheme for the browser. Rather than scheme Brendan Eich created Mocha. Mocha was intended to be an easy to use programming language for developers that wanted to run simple scripts in a browser. Six months after Mocha was created Netscape (the owners of Mocha) and Sun Microsystems (the owners of Java) made a licensing deal to rename it to JS, so that it could piggyback off of the massive popularity of Java at the time.

With microsoft noticing the popularity of the Netscape browser, they decided to create their own. Their new browser (Internet Explorer) quickly gained massive popularity, giving Microsoft massive control over the web. Thus Microsoft decided to create their own JS, which they named JScript. This fragmented the JS code base and as such Netscape turned to the European computer manufacturers association (ECMA) to unify and regulate JS and JScript. ECMA created ECMA 262 (commonly ECMAscript or ES) in June of 1997 [3], which is a set of guidelines on how to implement JS.

The first draft of ES contained many of the features used in modern day JS, such as first class functions and dynamic typing. However, it was missing important features such as try-catch exception handling and strict equality. Later version of ES added json compatibility, promises, strict mode, let and async functions.

Many of the new features presented a problem for JS. Their inclusion broke backwards compatibility with older versions and subsequently older browsers that used those versions. The main approach to solving this was to transpile JS to an older version. The most popular compiler for this is called babel. While babel is a popular JS compiler for backwards compatibility, TypeScript can be used for the same purpose, since it is transpiled into ES 3. TypeScript is a superset of JS, which is optionally typed, this allows IDEs to better warn for type errors, thus creating an easier environment for developers.

There are other options when compared to TypeScript such as CoffeeScript and Dart, both of which are transpiled into JS and use the JS compiler to run the code. However Dart has its own very fast compiler, which is usually used on the server side. CoffeeScript adds some very pretty (and frankly quite pythonic) syntactic sugar to vanilla JS. A library which also adds some very useful syntax is jQuery and their use of $ operator to make code much more readable.

JavaScript compilers

In 2008 the V8 JS compiler was released by Google for the Chrome browser. This was a direct competitor to SpiderMonkey which was the first JS compiler, made by Brendan Eich at

Netscape and currently maintained by Mozilla. It was created in order to speed up the runtime of JS, so that it could not only do small tasks in a browser, but could also be used for non-browser related projects. This speedup is done by performing just in time (JIT) compilation, instead of compiling to an intermediate language and interpreting that, which is how SpiderMonkey originally worked. V8 initially performs a very fast compilation to relatively unperformant machine code, by a compiler called full-codegen. This is then executed, but at the same time another thread is analysing the code and observing places where optimizations could be applied. These areas where optimization can be applied are called hot code segments and are usually parts of the code that are reached multiple times.

The possible optimizations are passed onto a JIT compiler, called crankshaft. These optimizations come in the form of inlining and inline caching. Inlining replaces function calls with the functions code, so that the functions doesn't have to be looked up. Inline caching remembers the result of a previous function call in order to avoid expensive type checking of function parameters. Once crankshaft has finished its optimizations on stack replacement is performed. This replaces the entire stack, which was running the unoptimized machine code, with the newly optimized code produced by crankshaft. In 2017 V8 was updated and replaced both full-codgen and crankshaft with ignition and turbofan respectively, which are more performant and work better with newer versions of ES.

Frameworks

Shortly after V8 was released NodeJS was created. It made javascript a powerful tool for server side applications and non web development, while using V8 as the compiler. It gave rise to the JS everywhere paradigm. This paradigm allows the use of JS for both front and backend code, allowing a codebase to only use a single language. However, possibly the best thing to come out of NodeJS is the node package manager or npm. This allows NodeJS modules to be easily installed and was the fastest growing open source platform in 2017 [4]. Webpack, which is an npm module, bundles all dependent JS functions into a single JS file, so that only code that is needed is used on the production website. This is used because loading too many scripts can cause a network bottleneck [5].

Three of the most popular npm modules are AngularJS, ReactJS and VueJS. These modules are JS frameworks used for designing web pages. ReactJS took a lot of influence from the declarative UI of AngularJS, but is more lightweight, and introduced a unidirectional data flow and immutability to improve on AngularJS. While VueJS takes a lot of queues from ReactJS, it even more lightweight, is thought of as easier for new developers and very useful for smaller projects. Possibly the biggest advantage of ReactJS is ReactNative, this is a framework for building native apps in JS using ReactJS. It is incredibly useful because it allows for the use of a single framework when building applications and websites for a product. All that being said, these libraries have had a huge influence on how front end developers write modern UI code.

Through all of these framework inclusions JS has evolved from only running simple scripts in the browser to creating real time animations on web pages and even being the language used for the backend development.

References

[1] Cassel, D. (2018). *Brendan Eich on Creating JavaScript in 10 Days, and What He'd Do Differently Today - The New Stack*. [online] The New Stack. Available at: https://thenewstack.io/brendan-eich-on-creating-javascript-in-10-days-and-what-hed-do-differently-today/ [Accessed 23 Sep. 2019].

[2] Ecma-international.org. *Standard ECMA-262-archive*. [online] Available at: https://www.ecma-international.org/publications/standards/Ecma-262-arch.htm [Accessed 23 Sep. 2019].

[3] Bynens, M. (2018). *Celebrating 10 years of V8 · V8*. [online] V8.dev. Available at: https://v8.dev/blog/10-years [Accessed 23 Sep. 2019].

[4] Rogers, M. (2017). *Node.js everywhere*. [online] YouTube. Available at: https://www.youtube.com/watch?v=FqickVjXddM [Accessed 23 Sep. 2019].

[5] *Webpack*. [online] Available at: https://webpack.js.org/concepts/why-webpack/ [Accessed 23 Sep. 2019].

[6] Bak, L. (2008). *V8: an open source JavaScript engine*. [online] YouTube. Available at: https://www.youtube.com/watch?v=hWhMKalEicY [Accessed 23 Sep. 2019].

[7] Coffeescript.org. *CoffeeScript*. [online] Available at: https://coffeescript.org/ [Accessed 23 Sep. 2019].