Overview of WebKit’s CSS JIT Compiler

Mar 26, 2014
by Benjamin Poulain
@awfulben

When it comes to performance, JavaScript generally gets the headlines. But if you look carefully, web pages are not all JavaScript, the performance of all the other parts also has a dramatic impact on the user experience. Observing only JavaScript gives a single point of view over the complex mosaic that is web performance.

Making CSS faster and more scalable is an area of research in the WebKit project. The DOM and CSS do not always scale very well and it is sadly still common to see missed frames during complex animations.

A technique we use to speed things up in JavaScript is eliminating slow paths with Just In Time Compilers (JIT). CSS is a great candidate for JIT compilation, and since the end of last year, WebKit compiles certain CSS Selectors on the fly.

This blog post introduces the CSS Selector JIT, its pros and cons, and how you can help us make CSS better.

How and Why Compile CSS?

CSS is the language telling the engine how to present the DOM tree on screen. Every web engine uses a complex machinery that goes over everything in the DOM and applies the rules defined by CSS.

There is no direct link between the DOM tree and the style rules. For each element, the engine needs to collect a list of rules that apply to it. The way CSS and the DOM tree are connected is through the CSS selectors.

Each selector describes in a compact format what properties are required for an element to get a particular style. The engine has to figure out which elements have the required properties, and this is done by testing the selectors on the elements. As you can imagine, this is a complicated task for anything but the most trivial DOM tree (fortunately WebKit has many optimizations in this area).

So how did we make this faster? A simple solution we picked is to make testing a selector on an element very fast.

The way selector matching used to work is through a software machine, the *SelectorChecker* instance, taking two inputs: a selector and an input element. Given the inputs, a *SelectorChecker* goes over each part of the selector, and tries to find the required properties in the tree ending with the input element.

The following illustrates a simplified version of how selector testing used to work:

Click to start

The problem with *SelectorChecker* is that it needs to be completely generic. We had a complicated selector interpreter, capable of handling any combination of difficult cases for any selector. Unfortunately, big generic machines are not exactly fast.

When using the CSS JIT, the task of matching a selector is split in two: first compiling, then testing. A JIT compiler takes the selector, does all the complicated computations when compiling, and generates a tiny binary blob corresponding to the input selector: a compiled selector. When it is time to find if an element that matches the selector, WebKit can just invoke the compiled selector.

The following animation illustrates the same process as above, but using the CSS Selector JIT:

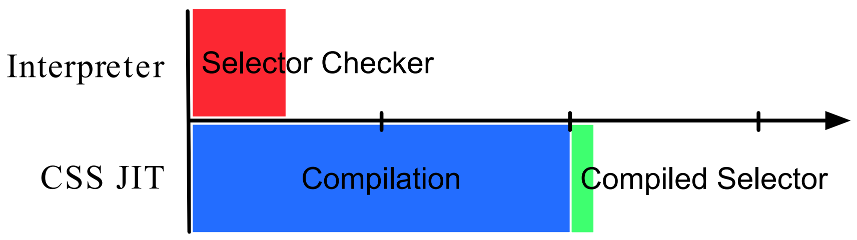
Click to start

Obviously, all the complexity related to CSS Selectors is still there. The Selector JIT Compiler is a big generic machine, just like *SelectorChecker* was. What changed is that most of that complexity has been moved to compilation time, and it only happens once. The binary generated at runtime is only as complex as the input selector.

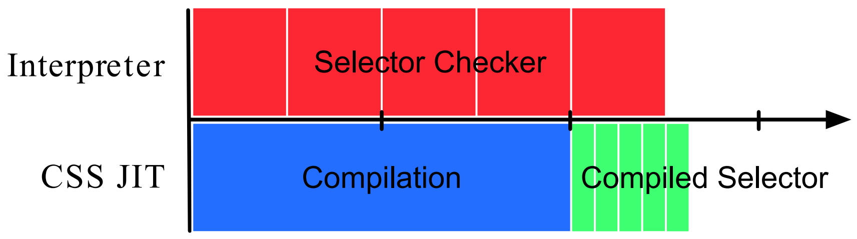
There is Beauty in Simplicity

Although one might think that employing a JIT always makes execution faster, it is a fallacy. The truth is adding a compiler starts by making everything slower, and the compiler makes up for it by creating very fast machine code. The overall process is only a gain when the combined execution time of the compiler and compiled code is smaller than the execution time of the compiler.

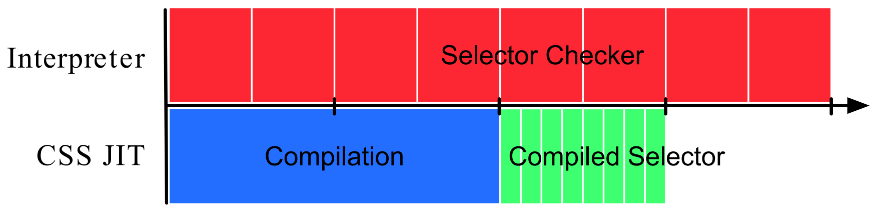
When the workload is small, the time taken by the compiler is bigger than the gain. For example, let's say we have a JIT compiler that is 4 times slower than *SelectorChecker*, but the compiled code is 4 times as fast as *SelectorChecker*. Here is the time diagram of one execution:



With this kind of timing, we can run 5 full queries on the old C++ selector checker and still be faster than the JIT.



When the JIT compiler is fast enough and the workload is large enough, the compiled version wins:



This constraint is also a reason why benchmarks running for a long time can be misleading, they can hide slow compilers. JIT compilers can help to have a great throughput for long running programs, but no real web page behaves that way. The latency introduced by compilation also has the potential to become a disaster for animations.

Does this mean we shot ourselves in the foot by making something that is only fast in benchmarks? Not really, we fixed that problem too.

There are several ways to mitigate the latency introduced by a JIT compiler. JavaScriptCore uses multiple advanced subsystems to reach that goal. So far, the Selector JIT can get away with a simple solution: make the compiler extremely fast.

There are two key parts to the speed of this compiler.

1. First, the compiler is very simple. Making optimizations can take a lot of time, so we decided to optimize very little. The generated binary is not perfect but it is fast to produce.
2. The second trick is to use very fast binary generation. To do that, the compiler is built on top of JavaScriptCore's infrastructure. JavaScriptCore has tools to generate binaries extremely quickly, and we use that directly in WebCore.

In the most recent versions of the JIT, the compilation phase is within one order of magnitude of a single execution of *SelectorChecker*. Given that even small pages have dozen of selectors and hundreds of elements, it becomes easy to reclaim the time taken by the compiler.

How Fast Is It?

To give an idea of order of magnitude, I have prepared a little microbenchmark for this blog. It tests various use cases, including things that used to be slow on WebKit.

On my Retina Macbook Pro, the benchmark runs in about 1100 milliseconds on a WebKit from December, and in less than 500 milliseconds on today's WebKit nightly. A gain of 2x is generally what we expect on common selectors.

Obviously, the speed gains depends greatly on the page. Gains are sometimes much larger if the old WebKit was hitting one of the slow paths, or could be smaller for selectors that are either trivial or not compiled. I expect a lot to change in the future and I hope we will get even more feedback to help shaping the future of CSS performance.

What About querySelector?

The functions `querySelector()` and `querySelectorAll()` currently share a large part of infrastructure with style resolution. In many cases, both functions will also enjoy the CSS JIT Compiler.

Typically, the querySelector API is used quite differently from style resolution. As a result, we optimize it separately so that each subsystem can be the fastest for its specific use cases. A side effect of this is that querySelector does not always give a good picture of selector performance for style resolution, and vice versa.

How Can You Help?

There is ongoing work to support everything *SelectorChecker* can handle. Currently, some pseudo types are not supported by the JIT compiler and WebKit fall backs to the old code. The missing pieces are being added little by little.

There are many opportunities to help making CSS faster. The existing benchmarks for CSS are extremely limited, there is nothing like JSBench for CSS. As a result, the input we get from performance problems on real websites is immensely valuable.

If you are a web developer, or a WebKit enthusiast, try your favorite website with WebKit Nigthly. If you run into performance problems with CSS, please file a bug on WebKit's bug tracker. So far, every single bug that has been filed about the CSS JIT has be hugely useful.

Finally, if you are interested in the implementation details, everything is open source and available on webkit.org. You are welcome to help make the web better.

You can send me questions to @awfulben on twitter. For more in-depth discussions, you can write an email on webkit-help (or file a bug report). ■

Next

Introducing the WebKit FTL JIT

Learn more

Previously

Advanced Layout Made Easy with CSS Regions

Learn more