CSS Selector Performance has changed! (For the better)

calendar.perfplanet.com/2011/css-selector-performance-has-changed-for-the-better/

Great articles, like Dave Hyatt's Writing Efficient CSS, helped developers adapt to a rudimentary selector matching landscape. We learned from Steve Souders (and others) that selectors match from right to left, and that certain selectors were particularly arduous to match and should best be avoided. For example, we were told that descendant selectors were slow, especially when the right-most selector matched many elements on the page. All this was fantastic information when we had none, but as it turns out, times have changed. Thanks to some amazing work by Antti Koivisto there are many selectors we don't need to worry about anymore.

Antti contributes code to WebKit core and recently spent some time optimizing CSS selector matching. In fact, after finishing his work, he said:

> *"My view is that authors should not need to worry about optimizing selectors (and from what I see, they generally don't), that should be the job of the engine."*

~ Antti Koivisto

Wow! That sounds fantastic to me. I'd love to be able to use selectors in a way that makes sense for my architecture and let the rendering engine handle selector optimization. So, what did he do? Not just one thing, rather he created multiple levels of optimization — we'll take a look at four optimizations in particular:

1. Style Sharing
2. Rule Hashes
3. Ancestor Filters
4. Fast Path

Style Sharing

Style sharing allows the browser to figure out that one element in the style tree has the same styles as something it has already figured out. Why do the same calculation twice!

For example:

```
<div>
 <p>foo</p>
 <p>bar</p>
</div>
```

If the browser engine has already calculated the styles for the first paragraph, it doesn't need to do so again for the second paragraph. A simple but clever change that saves the browser a lot of work.

Rule Hashes

By now, we all know that the browser matches styles from right to left, so the rightmost selector is really important. Rule hashes break a stylesheet into groups based on the rightmost selector. For example the following stylesheet would be broken into three groups:

```
a {}
div p {}
div p.legal {}
#sidebar a {}
#sidebar p {}
```

a	p	p.legal
a {}	div p {}	div p.legal {}
#sidebar a {}	#sidebar p {}	

When the browser uses rule hashes it doesn't have to look through every single selector in the entire stylesheet, but a much

smaller group of selectors that actually have a chance of matching. Another simple but very clever change that eliminates unnecessary work for every single HTML element on the page!

Ancestor Filters

The ancestor filters are a bit more complex. They are *Probability filters* which calculate the likelihood that a selector will match. For that reason, the ancestor filter can quickly eliminate rules when the element in question doesn't have required matching ancestors. In this case, it tests for descendant and child selectors and matches based on class, id, and tag. Descendant selectors in particular were previously considered to be quite slow because the rendering engine needed to loop through each ancestor node to test for a match. The bloom filter to the rescue.

A bloom filter is a data structure which lets you test if a particular selector is a member of a set. Sounds a lot like selector matching, right? The bloom filter tests whether a CSS rule is a member of the set of rules which match the element you are currently testing. The cool thing about the bloom filter is that false positives are possible, but false negatives are not. That means that if the bloom filter says a selector doesn't match the current element, the browser can stop looking and move on the the next selector. A huge time saver! On the other hand, if the bloom filter says the current selector matches, the browser can continue with normal matching methods to be 100% certain it is a match. Larger stylesheets will have more false positives, so keeping your stylesheets reasonably lean is a good idea.

The ancestor filter makes matching descendant and child selectors very fast. It can also be used to scope otherwise slow selectors to a minimal subtree so the browser only rarely needs to handle less efficient selectors.

Fast Path

Fast path re-implements more general matching logic using a non-recursive, fully inlined loop. It is used to match selectors that have any combination of:

1. Descendant, child, and sub-selector combinators, and
2. tag, id, class, and attribute component selectors

Fast Path improved performance across such a large subset of combinators and selectors. In fact, they saw a 25% improvement overall with a 2X improvement for descendant and child selectors. As a plus, this has been implemented for querySelectorAll in addition to style matching.

If so many things have improved, what's still slow?

What is still slow?

According to Antti, direct and indirect adjacent combinators can still be slow, however, ancestor filters and rule hashes can lower the impact as those selectors will only rarely be matched. He also says that there is still a lot of room for webkit to optimize pseudo classes and elements, but regardless they are much faster than trying to do the same thing with JavaScript and DOM manipulations. In fact, though there is still room for improvement, he says:

> *"Used in moderation pretty much everything will perform just fine from the style matching perspective."*

~ Antti

I like the sound of that. The take-away is that if we can *keep stylesheet size sane*, and *be reasonable with our selectors*, we don't need to contort ourselves to match yesterdays browser landscape. Bravo Antti!

Want to learn more? Check out Paul Irish's presentation on CSS performance.