Efficiently Rendering CSS

css-tricks.com/efficiently-rendering-css/

Published May 24, 2010 by Chris Coyier

I admittedly don't think about this idea very often... **how efficient is the CSS that we write, in terms of how quickly the browser can render it?**

This is definitely something that browser vendors care about (the faster pages load the happier people are using their products). Mozilla has an article about best practices. Google is also always on a crusade to make the web faster. They also have an article about it.

Let's cover some of the big ideas they present, and then discuss the practicalities of it all.

Right to Left

One of the important things to understand about how browsers read your CSS selectors, is that they read them from **right to left**. That means that in the selector `ul > li a[title="home"]` the first thing thing interpreted is `a[title="home"]`. This first part is also referred to as the "key selector" in that ultimately, it is the element being selected.

ID's are the most efficient, Universal are the least

There are four kinds of key selectors: ID, class, tag, and universal. It is that same order in how efficient they are.

```
#main-navigation { } /* ID (Fastest) */
body.home #page-wrap { } /* ID */
.main-navigation { } /* Class */
ul li a.current { } /* Class *
ul { } /* Tag */
ul li a { } /* Tag */
* { } /* Universal (Slowest) */
#content [title='home'] /* Universal */
```

When we combine this right-to-left idea, and the key selector idea, we can see that this selector isn't very efficient:

```
#main-nav > li { } /* Slower than it might seem */
```

Even though that feels weirdly counter-intuitive... Since ID's are so efficient we would think the browser could just find that ID quickly and then find the li children quickly. But in reality, the relatively slow li tag selector is run first.

Don't tag-qualify

Never do this:

```
ul#main-navigation { }
```

ID's are unique, so they don't need a tag name to go along with it. Doing so makes the selector less efficient.

Don't do it with class names either, if you can avoid it. Classes aren't unique, so theoretically you could have a class name do something that could be useful on multiple different elements. And if you wanted to have that styling be different depending on the element, you might need to tag-qualify (e.g. `li.first`), but that's pretty rare, so in general, don't.

Descendant selectors are the worst

David Hyatt:

> *The descendant selector is the most expensive selector in CSS. It is dreadfully expensive — especially if the selector is in the Tag or Universal Category.*

In other words, a selector like this is an efficiency disaster:

```
html body ul li a { }
```

A selector that fails is more efficient than that same selector matching

I'm not sure if there is much we can learn from this, because if you have a bunch of selectors in your CSS that don't match anything, that's, uhm, pretty weird. But it's interesting to note, that in the right-to-left interpretation of a selector, as soon as it fails a match, it stops trying, and thus expends less energy than if it needed to keep interpreting.

Consider why you are writing the selector

Consider this:

```
#main-navigation li a { font-family: Georgia, Serif; }
```

`Font-family` cascades, so you may not need a selector that is that specific to begin with (if all you are doing is changing the font). This may be just as effective, and far more efficient:

```
#main-navigation { font-family: Georgia, Serif; }
```

CSS3 and Efficiency

Kind of sad news from David Hyatt:

> *The sad truth about CSS3 selectors is that they really shouldn't be used at all if you care about page performance.*

The whole comment is worth reading.

CSS3 selectors (e.g. :nth-child) are incredibly awesome in helping us target the elements we want while keeping our markup clean and semantic. But the fact is these fancy selectors are more browser resource intensive to use.

So what's the deal, should we actually not use them? Let's think about practicalities a bit...

Practicalities

That Mozilla article I linked to at the top? Literally 10 years old. Fact: computers were way slower 10 years ago. I have a feeling this stuff was more important back then. Ten years ago I was about to turn 21 and I don't think I even knew what CSS was, so I'm not going to get all old school on you... but I have a feeling we don't talk about this rendering efficiency stuff very much is because it's not that big of a problem anymore.

This is how I'm feeling about it: the best practices we covered above make sense no matter what. You might as well follow them, because they don't limit your abilities with CSS anyway. But you don't have to be all dogmatic about it. If you happen to be in the position where you need to eek out every last drop of performance out of a site and you have never considered this stuff before, it may be worth revisiting your stylesheets to see where you can do better. If you aren't seeing much rendering slowness in your site, then don't worry about it, just be aware for the future.

Super-speed, Zero-practicality

So we know that ID's are the most efficient selectors. If you wanted to make the most efficiently rendering page

possible, you would literally give every single element on the page a unique ID, then apply styling with single ID selectors. That would be super fast, and also super ridiculous. It would probably be extremely non-semantic and extremely difficult to maintain. You don't see this approach even on hardcore performance based sites. **I think the lesson here is not to sacrifice semantics or maintainability for efficient CSS.**

Thanks to Jason Beaudoin for emailing me about the idea. If anyone knows more about this stuff, or if you have additional tips that you use in this same vein, let's hear it!

Just as a quick note, I'd also like to mention that since CSS style selectors are also used in many JavaScript libraries, these same concepts also apply. ID selectors are going to be the fastest while complicated qualified descendant selectors and such will be slower.