Understanding The CSS Cascade

When working on my piece about contrast I found myself writing about the CSS cascade. The more I realized I didn't understand the cascade itself. So this is my attempt at understanding the cascade, different types of style sheets and how the interact with each other.

I'm also coming to the realization that this is one of the main issues that attract people to CSS in Javascript. The cascade is complicated and, sometimes, the interaction between user agent and author style sheets can be a real PITA so any option that removes the cascade from the equation seems attractive.

So rather than move everything to Javascript I thought I would try to understand the cascade and how to figure out what value will be shown to the user.

Different kinds of style sheets

According to MDN, browsers can handle 3 different kinds of style sheets.

- The browser has a basic style sheet that gives a default style to any document. These style sheets are named user-agent style sheets
- The author of the Web page defines styles for the document. These are the most common style sheets
- The reader, the user of the browser, may have a custom style sheet to tailor its experience

Most of the time we only have to deal with author style sheets because we don't have access to the user-agent style sheet (and shouldn't change it even if we did) and don't see if the reader has a stylesheet.

It is still important to know what stylesheets are available to avoid confustion in understanding the cascade and specificity rules that follow.

The cascade

How the browser sorts which rule to apply to an element is known as the cascade

(which gives CSS the Cascading part of its name) and helps figure out which rule to use when rules from multiple styelsheets affect the same element.

1. It first filters all the rules from the different sources to keep only the rules that matches the given element and which are part of an appropriate media at-rule
2. Then it sorts the matching rules according to their importance, whether or not they are followed by !important, and by their origin
3. In case of equality, the specificity of a value is considered to choose one or the other

Specificity

When the cascading rules in the prior section refer to specificity they mean the weight of a selector based on the number of different kinds of selectors that apply.

Whenever I have to explain this I've fallen back to Estelle Weyl's CSS SpeciFISHity chart. It has helped me figure out specificity issues many times and it's a fun way to look at the values for different element combinations.

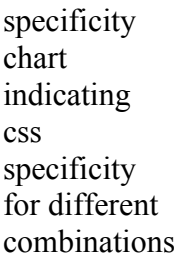

Figure 1: CSS Specifishity chart from Estelle Weyl

I've also extracted the text at the bottom of the image that explains in more detail what the different specificity rules are

- X-0-0: The number of ID selectors, represented by Sharks
- 0-Y-0: The number of class selectors, attributes selectors, and pseudo-classes, represented by Fish
- 0-0-Z: The number of type selectors and pseudo-elements, represented by Plankton a la Spongebob
- *: The universal selector has no value
- +, >, ~: combinators, although they allow for more specific targeting of elements, they do not increase specificity values

- `:not(x)`: The negation selector has no value, but the argument passed increases specificity

Order of precedence

We have one more thing to talk about to, hopefully, get a better idea of how the cascade works. That's the order of precedence for stylesheets from different origins.

The origin of a declaration is based on where it comes from and its importance is whether or not it is declared !important. As our work with CSS becomes more complex we need to figure out if our rules will be pre-empted by a stylesheet with higher precedence.

The precedence of the various origins is, in descending order:

- Transition declarations
- Important user-agent declarations (`!important`)
- Important user declarations (`!important`)
- Important author declarations (`!important`)
- Animation declarations. At any given time a CSS animation takes values from only one `@keyframes` element, and never mixes multiple `@keyframes` together
- Normal author declarations
- Normal user declarations
- Normal user-agent declarations

Declarations from origins earlier in this list win over declarations from later origins.

Conclusion

The cascade is a central part of working with CSS. Yes, it can be a nightmare to figure out. Yes it can cause problems of specificity. But if you use it carefully it can give you awesome results.

Links and Resources

- CSS Cascading and Inheritance Level 3
- Cascade and inheritance

- Which CSS entities participate in the cascade