vanseodesign

People Friendly & Search Friendly Web Design

- Home
- Design
- SEO
- Blog

Understanding Style Precedence in CSS: Specificity, Inheritance, and the Cascade

by Steven Bradley
on Tuesday, June 9th, 2009
in CSS

« Jammed Finger, Hacked Sites, And A New Twitter Client

4 Common But Not-So-Visible SEO Mistakes »

Have you ever run into the situation where you're trying to apply a css style to an element, but it won't take? Your page it seems to be ignoring your css, but you can't figure out why. Maybe you found yourself using !important or adding an inline style as a last resort. There's a good chance the problem you encountered was one of css precedence.

A better understanding of which css styles take precedence can lead to less frustration with css, cleaner code, and more organized css so let's look at three things that control which css rule applies to a given html element:

- Specificity Calculations
- Inheritance
- The Cascade

Learning these rules will take you to the next level in your css development.

Specificity Calculations

Imagine your html contains a paragraph with a class of “bio” applied to it. You also have the following two css rules:

```
p {font-size: 12px}
p.bio {font-size: 14px}
```

Would you expect the text in your paragraph to be 12px or 14px? You can probably guess in this case it will be 14px. The second line of css (p.bio) is more specific than the first when it comes to your class=”bio” paragraph. However, sometimes the specificity isn’t so easy to see.

For example consider the following html and css

```
<div id="sidebar">
<p class="bio">text here</p>
</div>
```

```
div p.bio {font-size: 14px}
#sidebar p {font-size: 12px}
```

The first line of css might seem more specific at first glance, but it’s actually the second line above that would be more specific to the font-size of your paragraph. Why is that?

To answer the question we need to consider the rules of specificity.

Specificity is calculated by counting various components of your css and expressing them in a form (a,b,c,d). This will be clearer with an example, but first the components.

- Element, Pseudo Element: d = 1 – (0,0,0,1)
- Class, Pseudo class, Attribute: c = 1 – (0,0,1,0)
- Id: b = 1 – (0,1,0,0)
- Inline Style: a = 1 – (1,0,0,0)

An id is more specific than a class is more specific than an element.

You calculate specificity by counting each of the above and adding 1 to either a,b,c, or d. It’s also important to note that 0,0,1,0 is more specific than 0,0,0,15. Let’s look at some examples to make the calculation clearer.

- p: 1 element – (0,0,0,1)
- div: 1 element – (0,0,0,1)
- #sidebar: 1 id – (0,1,0,0)
- div#sidebar: 1 element, 1 id – (0,1,0,1)
- div#sidebar p: 2 elements, 1 id – (0,1,0,2)
- div#sidebar p.bio: 2 elements, 1 class, 1 id – (0,1,1,2)

Let’s look again at the example above

```
div p.bio {font-size: 14px} - (0,0,1,2)
#sidebar p {font-size: 12px} - (0,1,0,1)
```

The second has the higher specificity and thus takes precedence.

One last point before we move on. Importance trumps specificity, When you mark a css property with !important you're overriding specificity rules and so

```
div p.bio {font-size: 14px !important}
#sidebar p {font-size: 12px}
```

means the first line of css above takes precedence instead of the second. !important is still mostly a hack around the basic rules and is something you should never need if you understand how the rules work.

Inheritance

The idea behind inheritance is relatively easy to understand. Elements inherit styles from their parent container. If you set the body tag to use color: red then the text for all elements inside the body will also be red unless otherwise specified.

Not all css properties are inherited, though. For example margins and paddings are non-inherited properties. If you set a margin or padding on a div, the paragraphs inside that div do not inherit the margin and padding you set on the div. The paragraph will use the default browser margin and padding until you declare otherwise.

You can explicitly set a property to inherit styles from it's parent container, though. For example you could declare

```
p {margin: inherit; padding: inherit}
```

and your paragraph would then inherit both from it's containing element.

The Cascade

At the highest level the cascade is what controls all css precedence and works as follows.

1. Find all css declarations that apply to the element and property in question.
2. Sort by origin and weight. Origin refers to the source of the declaration (author styles, user styles, browser defaults) and weight refers to the importance of the declaration. (author has more weight than user which has more weight than default. !importance has more weight than normal declarations)
3. Calculate specificity
4. If two rules are equal in all of the above, the one declared last wins. CSS embedded in the html always come after external stylesheets regardless of the order in the html

#3 above is likely the one you'll need to pay attention to most. With #2 just understand that your styles will override how a user sets their browser unless they set their rules to be important.

Also realize that your styles will override the browser defaults, but those defaults do exist and is often what leads to cross browser issues. Using a reset file like Eric Meyer's CSS Reset or Yahoo's YUI Reset CSS helps take the default styles out of the equation.

Summary

Hopefully the above helps sort out some of your css precedence issues. Most of the time if you have a conflict in styles the issue will come down to specificity. At times when you haven't declared some css, but an element is behaving in a way you don't expect it's likely that element has inherited some css from a parent container or a default style of the browser.

A general rule of thumb when declaring your css is to declare properties with the least specificity needed to style your elements. Use #sidebar instead of div#sidebar for example. I admit to breaking this general rule far more than I should, but by using the least specificity needed it will make it easier for you to override a style by declaring a style more specific.

If you use the most specificity you may run into problems later and find yourself having to add unnecessary html in order to be able to add more specificity or you may find yourself falling back on using !important or declaring inline styles. Start with the least specificity and add more only as needed.

Related Posts.

- Do You Use These 7 Attribute Selectors In Your CSS?
- Animation With CSS: It's Easier Than You Think
- CSS Background: There's More To Know Than You Think