LESS CSS

Thursday, February 16, 2017 11:37 AM

Pseudo element selectors

I had something like this:

```
<div class="makePink">
    <div class="childWithPseudo"/>
</div>

LESS
.childWithPseudo {
  &::after {
    content: "";
    border-color: blue;
    &.makePink: {
      border-color: pink;
    }
  }
}
```

What I *wanted* is for childWithPseudo to have a blue border UNLESS the parent has the "makePink" class, then it would have a pink border, but **that doesn't work**. Instead, you have to do it like this:

```
.childWithPseudo {
  &::after {
    content: "";
    border-color: blue;
  }

  &.makePink {
    &::after {
      border-color: pink;
    }
  }
}
```

Switching from mobile to desktop without having to set "initial" for everything

Problem: you have a CSS class where you want one style on mobile and one style on desktop. This is achievable with a media query, but you have to be careful to reset everything between mobile and desktop, e.g.:

```
.exampleMobileAndDesktop {
    background-color: red;
    color: blue;

    .screenMd({
        color: green;
    });
}
```

(note: screenMd is a mixin based on a media query)

In the above example, the background-color is red no matter what, and the color is what changes based on a media query. However, let's say you didn't want the background-color to be red no matter what, then you may want to add "background-color: initial;" to the media query section. If you have a lot of properties, it becomes kind of annoying to figure out what to set to initial.

To avoid this, I kind of cheat my way around it by having two mutually exclusive media queries:

```
.exampleMobileAndDesktop {
    .screenMaxMd({
        background-color: red;
        color: blue;
    });

    .screenMd({
        color: green;
    });
}
```

Converting from SCSS (reference)

The reference link is a CodePen that does some simple find/replaces (i.e. it's not "intelligent"):

- "$" turns into "@"
- "@mixin" turns into "."
- "@include" turns into "."

I don't think it's really worthwhile to use a CodePen for that.

Media queries (reference to my other OneNote about this)

Setting variables based on media queries (reference)

You can't do this because LESS compiles to CSS before you're actually in a browser.

Using LESS CSS media queries in JS

I wanted this for Bot Land since I had a particular use-case: I wanted load images based on a particular media query, but the image URLs had to go through a JavaScript function first to transform them into the right CloudFront link.

Note: ignore all of the following tabbed-in notes and skip to the conclusion beneath them:

> ~~In order to make the media queries accessible by JS, I had to use less-vars-loader, but that only gives me the very basic variable assignments, whereas I had something like this in LESS:~~
>
> ```
> @md: 1024px;
> @mdQuery: ~"(min-width: @{md})";
> .screenMd(@rules) {
>     @media @mdQuery { @rules(); }
> }
> ```
>
> ~~Because @mdQuery uses string interpolation to form the query, less-vars-loader would need to be forked/modified so that it could piece together that information to expose to JavaScript. After that, it would be a simple matter of invoking the loader on the right LESS file (making sure not to have my regular LESS loader *also* parse that data and remove the variables altogether), then calling window.matchMedia on the media query string to see if it matches in JavaScript.~~
>
> ~~Alternatively, this library kind of goes the other way around and tries to put JS variables into LESS, but I don't think that's a good idea for my situation.~~

Turns out HiDeoo found a better solution for all of this and wrote up a nice readme here (I forked his repo). Basically, CSS modules allow for an ":export" pseudo selector that can expose values to the

consumer of CSS (which in my case is JavaScript of course):

9:00 HiDeoo: It's the low level implementation of CSS modules Adam13531
https://github.com/css-modules/icss
9:00 HiDeoo: It's not very documented

Another thing to note is that the specific way that he exported things gives the breakpoint values as NUMBERS and not as "50px". My eventual LESS file ended up like this

```
/* Responsive Sizes (landscape mode) */
@xs: 568; /* iPhone 5   */
@sm: 640; /* 720p Phone */
@md: 1024; /* iPad       */
@lg: 1366; /* iPad Pro   */
@xl: 1920; /* Desktop    */

@screenXsMq: ~"(min-width: @{xs}px)";
@screenSmMq: ~"(min-width: @{sm}px)";
@screenMdMq: ~"(min-width: @{md}px)";
@screenLgMq: ~"(min-width: @{lg}px)";
@screenXlMq: ~"(min-width: @{xl}px)";

:export {
  screenXsMq: @screenXsMq;
  screenSmMq: @screenSmMq;
  screenMdMq: @screenMdMq;
  screenLgMq: @screenLgMq;
  screenXlMq: @screenXlMq;

  xsBreakpoint: @xs;
  smBreakpoint: @sm;
  mdBreakpoint: @md;
  lgBreakpoint: @lg;
  xlBreakpoint: @xl;
}

.screenXs(@rules) {
    @media @screenXsMq { @rules(); }
}
.screenSm(@rules) {
    @media @screenSmMq { @rules(); }
}
.screenMd(@rules) {
    @media @screenMdMq { @rules(); }
}
.screenLg(@rules) {
    @media @screenLgMq { @rules(); }
}
.screenXl(@rules) {
    @media @screenXlMq { @rules(); }
}
```

On the JS side of things:

```
import mediaQueryStyles from '../../../stylesheets/layout/
_mediaqueries.less';
if (window.matchMedia(mediaQueryStyles.screenMdMq).matches) {
     // do something here in JS based on matching the "medium" media
```

```
    query
}
```

Using media query mixins to centralize logic (reference)

Example from that site (slightly modified):

```
/* The LESS CSS that you have to type */
// Mixin
.whenScreenHasMaxWidth(@maxWidth; @rules) {
    @media only screen and (max-width: @maxWidth) {
        @rules();
    }
}

// Usage
.foo {
    width: 100%;

    .whenScreenHasMaxWidth(450px, {
        float: left;
        margin-top: 10px;
    });
}

/* The outputted css */
.foo {
  width: 100%;
}

@media only screen and (max-width: 450px) {
  .foo {
    float: left;
    margin-top: 10px;
  }
}
```

For another example about this, check out this page, which has a lot of mixins based on screen sizes. The sizes themselves may not be so helpful, but seeing how they're laid out could be interesting.

Math / operators

Just do math inline if it's straightforward:

```
@size = 5px;
.foo {
    margin-right: -@size / 2 + 5;
}
```

If you have to specify a variable alongside other values, just don't use spaces:

```
.foo {
    padding: 0px @volumeSliderHandleSize/2;
}
```

Parentheses or no spaces apparently also work, so I'm not totally sure where my note about "no spaces" comes from.

Descendants selector syntax

In CSS, if you have something like this:

```
.class1 .class2 {}
```

You can make it into this in LESS:

```
.class1 {
    & .class2 {}
}
```

A shortcut for that is to omit the ampersand

```
.class1 {
    .class2 {}
}
```

You apparently only need the ampersand when referencing &::hover or ".class.subclass".

Importing as a reference (reference)

You can do "@import (reference) 'foo';" to import an *external* file as a reference but only include any styles in the compiled CSS if they were actually used.

Mixins / inheritance

(A reference that I haven't thoroughly read)

I had a case where I wanted a "default" style for a button and then to be able to override colors as needed. I tried using inheritance ("&:extend(.iconButtonBase);"), but that didn't seem to let me override anything.

I ended up using a mixin, and note that order does matter:

```
.redButton {
    .iconButtonBase;
    background-color: red;
}
```

Using :global (reference)

This is *not* a property of LESS; it's a property of css-modules, but chances are I'm going to be confused in the future, so I'm writing this here.

Doing ".a.b" but 'a' as a global:

```
:global(.a).b {
    // css here
}
```

Doing ".a.b" with both as a global:

```
:global(.a.b) {}
```

The part of this that *does* apply to LESS is doing this inside a LESS class defined in the non-global scope:

```
.my-own-class {
  color: red;
  &:global(.some-other-class) {          // this is like doing ":global(.some-other-class).my-own-class"
```

```
outside of this scope
   background-color: blue;
  }
}
```

Using lesshint to ignore linting particular lines of LESS

6/19/2017

This feature was added from [this pull request] and can be used pretty simply:

```
:global a.scroll-spy-active i { // lesshint qualifyingElement: false
        left: initial!important; // lesshint importantRule:false
}
```

Using stylelint

2/19/2018

It's pretty simple to set this up:

1. Install stylelint
2. Install stylelint-config-standard
3. Make a config file (.stylelintrc)

```
{
  "extends": "stylelint-config-standard",
  "rules": {
    "indentation": 4,
    "no-descending-specificity": null,
    "selector-pseudo-class-no-unknown": [
      true,
      {
        "ignorePseudoClasses": ["global", "export"]
      }
    ],
    "property-no-unknown": [
      true,
      {
        "ignoreProperties": ["user-drag"]
      }
    ]
  },
  "ignoreFiles": [
    "public/stylesheets/layout/_mediaqueries.less"
  ],
  "syntax": "less"
}
```

4. Add this to your package.json file
   a. "lint:css": "stylelint ./public/stylesheets/**/*.less"
5. npm run lint:css

Changing colors when something is disabled

```
.loginButton {
   background-color: #fccc58;
   &[disabled] {
      background-color: lighten(@btn-default-bg, 30%)
   }
```

}

Note: lesshint will complain as of 7/19/2017 about the ampersand (which is required) thanks to this issue.

Colors

JudgeBacon: Adam13531, Quick tip which I found out yesterday, you can darken/lighten colours in Less using built in less functions. For example: darken(colour, % to darken by), useful if you want a colour for when you've got something selected or something.

LepkoQQ: its usually used in variable defining e.g $red: #f00; $red-selected: darken($red, 20), Adam13531

Using variables + units

I wanted to have width take a variable into account and provide that same variable as padding.

```
@contentMargin: 20;

.settingsContent {
    width: ~"calc(100% - (@{contentMargin} * 2)px)";
    margin: unit(@contentMargin, px);
}
```

Another way to do this same thing is to put the unit directly on the variable:

```
@contentMargin: 20px;

.settingsContent {
    width: ~"calc(100% - (@{contentMargin} * 2))";
    margin: @contentMargin;
}
```

I prefer the second way since it lets your definitions have units on them, that way you don't need to keep modifying every user of the variable, you can just modify the variable.

Using calculations in media queries

According to this GitHub comment, you have to write calculations in media queries with extra parentheses:

```
@md: 1024;
@maxMd: (@md - 1);

@screenMdMq: ~"(min-width: @{md}px)";
@screenMaxMdMq: ~"(max-width: @{maxMd}px)";
```

Troubleshooting

Lesshint complains about "decimalZero: 1.0 should be written with leading zero"

"opacity: 1.0;" is apparently wrong and should be "opacity: 1;".

Less seems to ignore order of operations (missing a space between operator and operand)

(that's not a totally accurate problem summary, but look below)

If you have something like this:

```
    right: -1px * (@strip-width / 2) -4px;
```

The "-4" doesn't have a space, so it's treated as another property I think, e.g. "right: prop1 prop2" as opposed to "right: the mathematical difference of prop1 and prop2".