

Scalable Front-End Architecture using SaSS

Aaron Reisman

Github: <http://github.com/lifeiscontent>
Email: aaron@lifeiscontent.net

- ⦿ 10+ Years of Front-End Experience.
- ⦿ Frequently commits open source code,
- ⦿ Likes: Music, Technology and being Happy.

Hello,

My name is Aaron Reisman.

I've got 10+ years of front-end development experience.

I frequently commit open source code.

I like music, technology, and being happy, as I'm sure you do too.

What is SaSS?

- ⦿ Sass is a CSS Preprocessor
- ⦿ Sass has lots of interesting features
 - ⦿ Variables
 - ⦿ Nesting
 - ⦿ placeholder selectors
 - ⦿ Mixins
 - ⦿ Selector inheritance
 - ⦿ and more
- ⦿ Sass has two syntaxes. The main one is SCSS

Alright, so...

What is SASS?

Sass is a CSS Preprocessor.

It has a lot of interesting features like, Variables, Nesting, Placeholder Selectors, Mixins, Selector Inheritance, and more.

Sass has 2 syntaxes. The main one being SCSS. and it's the one I'll be using today.

Common SaSS Mistakes

- ➊ Over use of nested selectors.
- ➋ Unintentional Duplication of CSS Properties.

before we get into the scalable architecture of things, I'd like to talk about a few common mistakes that people make with SCSS.

Common SaSS Mistakes

- ⌚ Over use of nested selectors.
- ⌚ Unintentional Duplication of CSS Properties.

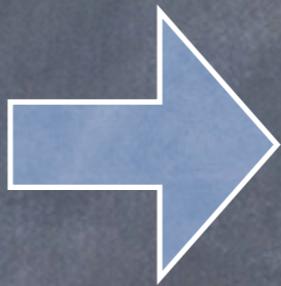
the first one being over use of nested selectors.

Over use of nested selectors.

SCSS

```
.module {  
  .hd {}  
  .bd {  
    ul {  
      li {  
        }  
      }  
    }  
  }  
}
```

BAD



CSS

```
.module {}  
.module .header {}  
.module .bd {}  
.module .bd ul {}  
.module .bd ul li {}
```

```
.module {}  
.module-hd {}  
.module-bd {}  
.module-list {}  
.module-item {}
```

GOOD

```
.module {}  
.module-hd {}  
.module-bd {}  
.module-list {}  
.module-item {}
```

If you look at the code on the left, we've got a module with a few classes and elements associated with it.

lets look at the compiled source of this module on the right.

The reasons why this might be considered bad is

1. The selector engine is working more then it needs to.
2. You're directly associating an element in your style, and if you ever decided to use another element in its place, you'd have to rewrite your code.
3. There isn't a great way to cognitively make a connection to your css in your markup. Hold on to this thought, we will be coming back to it shortly.

all 3 of these problems are solved in the example below.

we've eliminated the over-specified selectors, made each of the classes that are associated with the module apparent, and now allow for the use of any element with our new selectors.

Common SaSS Mistakes

- ⌚ Over use of nested selectors.
- ⌚ Unintentional Duplication of CSS Properties.

The next thing I'd like to briefly talk about is unintentional duplication of CSS Properties.

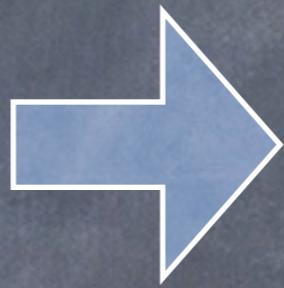
Unintentional Duplication of CSS

SCSS

Solution

CSS

```
@mixin inline-block {  
  display: inline-block;  
  *display: inline;  
  *zoom: 1;  
}  
.module-a {  
  @include inline-block;  
}  
.module-b {  
  @include inline-block;  
}
```



```
.module-a {  
  display: inline-block;  
  *display: inline;  
  *zoom: 1;  
}  
.module-b {  
  display: inline-block;  
  *display: inline;  
  *zoom: 1;  
}
```

Take a look at the code on the left.

When you look at the code on the right, you might notice that the properties we've defined are duplicated between both rules.

This is not ideal.

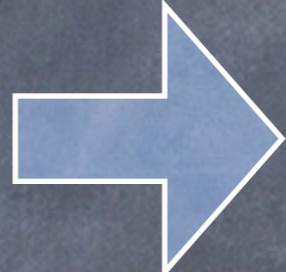
Unintentional Duplication of CSS

SCSS

Solution

CSS

```
.ibf {  
  @include inline-block;  
}  
.my-module-a {  
  @extends .ibf;  
}  
.my-module-b {  
  @extends .ibf;  
}  
.my-module-c {  
  @extends .ibf;  
}
```



```
.ibf,  
.my-module-a,  
.my-module-b,  
.my-module-c {  
  display: inline-block;  
  *display: inline;  
  *zoom: 1;  
}
```

Instead, we should create a base class and extend from it.

Look at the code on the left, it has the same flexibility as the code on the previous slide, but it compiles to a much more efficient result.

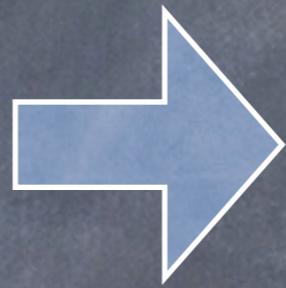
The code on the right has a base selector of .ibf.
if you don't need access to the base selector in your markup, the next slide contains a solution for this case.

Unintentional Duplication of CSS

SCSS

```
%ibf {  
  @include inline-block;  
}  
.my-module-a {  
  @extends %ibf;  
}  
.my-module-b {  
  @extends %ibf;  
}  
.my-module-c {  
  @extends %ibf;  
}
```

Solution



CSS

```
.my-module-a,  
.my-module-b,  
.my-module-c {  
  display: inline-block;  
  *display: inline;  
  *zoom: 1;  
}
```

Using placeholder selectors, the .ibf class is no longer in the compiled set of rules.

A few best practices

now that you're familiar with the pitfalls of SCSS, lets talk about best practices you can start using with this awesome css tool

SCSS Directory Structure

- ⦿ modules/ -- containers for content.
- ⦿ layouts/ -- structure of modules.
- ⦿ _base.scss -- structures layout.
- ⦿ _helpers.scss -- common patterns.
- ⦿ _state.scss -- application state changes.
- ⦿ _theme.scss -- visual style of the layout.
- ⦿ style.scss -- inherits everything above.

This is the directory structure I've found works really well for me,

There are Layouts, Modules, base, helpers, state, theme, and style.scss

Thinking about modules

- ⦿ A module structures the way content is displayed on a page
- ⦿ use placeholder selectors for base modules styles that are commonly used, that don't need access to your markup.
- ⦿ You can scope your modules by wrapping them in mixins.

First, lets talk about Modules.

The role of a Module is to structure the way content is displayed on a page.

Child classes of a module should be prefixed with that modules name.

For example, if your module's name was test, you should prefix your .header class .test-header

Thinking about layouts

- ⦿ A Layout manipulates the way modules are displayed on a page
- ⦿ No color should be applied directly on a layout
- ⦿ layouts should be prefixed with an “L” to specify it’s a layout.
- ⦿ example: .l-constrained

Ok, next up is layouts.

A Layout is to manipulate the way a module, or modules are displayed on a page.

No color should be applied directly on a layout.

Any class that represents a layout should be prefixed with an “L”,
the reason for this goes back to that original point of being able to visually make a connection that this is something different than just a plain class,
it’s a layout class!

Thinking about _base.scss

- `_base.scss` is used to style the structure of your layout.

`base.scss` is used to help structure anything that isn't a layout or a module on your site.

No color should be applied to `base.scss`, only structural properties like, box properties or floats.

Thinking about _helpers.scss

- ⦿ _helpers.scss is used for common patterns to help you with the structure of your layout.
- ⦿ use placeholder selectors if you don't plan to use them in your markup.

helpers.scss is used for common patterns to help aid you with the development of your site.

Things you might consider making into helpers are text-alignment, floats, vertical-alignment and in some cases margin and padding helpers.

Use placeholder selectors for your helpers if you don't need access to them in the markup.

Thinking about `_state.scss`

- ⦿ `_state.scss` is used for anything that modifies the original state of your application.
- ⦿ you can prefix your sites layout (html, body, or main content layer) with a class of something like `.is-active` or `.is-valid`
- ⦿ NOTE: If you have a module state change, put those states in that module.

`state.scss` is used for anything that is modifying the state of your application.

Anything that is a state change should be prefixed with the keyword `is`.

once again, the reason for the prefix of the classes is so you can make a direct connection in your markup to see what kind of class is being applied to the particular element in question. a good example for a state class might be a form that “is valid”, and as an added bonus, now you know immediately which file to open to find this class.

Thinking about _theme.scss

- ⦿ _theme.scss is used for anything that directly correlates to the visual style of your site.
- ⦿ properties that are considered theme: background color, text color, text-shadow, box-shadow, etc.

theme.scss is strictly used for color properties.

One reason why you should do this is, if you have a designer that would like to work on the sites theme, they can, and without the worry of messing anything up, other then the colors, because this file only contains the colors of the site, so the structure of each module will stay perfectly intact.

Thinking about style.scss

- ⦿ style.scss is where all of your modules, layouts, helpers, etc get inherited
- ⦿ Optionally include a css reset file here.

style.scss is strictly used to inherit all of the files in your project.

And if you want to use a css reset file, you can include that here.

Other considerations

If you need to namespace your modules so others can use them on their sites, you can just create a separate CSS file and include the modules under an ID or class to keep your selectors from clashing with theirs.

Other considerations

If you'd like to support IE but also have the option to drop support later with ease, you could create a `_ie.scss` file that would have all the abstracted bug fixes for your styles and then you can remove the include for `_ie.scss` in your `style.scss` to easily drop support.

Tools of the trade

- ⦿ Zen Coding
- ⦿ Sublime Text 2 (it's awesome)
 - ⦿ Package Manager, Python Interpreter, TextMate Bundle Support
- ⦿ Photoshop Extensions
- ⦿ CSS Hat
- ⦿ Guide Guide
- ⦿ Dwarf - On Screen Rulers / Guides

It's time for the code!