Tutorial de SASS

Los archivos CSS pueden llegar a crecer mucho al avanzar nuestros proyectos, por lo que se vuelven complejos, difíciles de entender y mantener. Por esa razón se crearon preprocesadores que nos pueden ayudar. Sass introduce ciertos apartados que no se encuentran en CSS, tales como el manejo de variables, y otras funcionalidades que se verán en el presente tutorial.

Requisitos para el tutorial

- Conocimientos básicos de desarrollo web con HTML y CSS

Instalación

Para instalar Sass en sistemas basados en Unix, ejecute:

```
$ sudo apt install ruby-sass
```

SASS

Variables

El uso de variables es simple, es como en cualquier otro lenguaje. Colocas nombre a la variable, y le asignas cierto valor. Simple. En Sass, el nombre de las variables inician con `$` . Luego de asignarles cierto valor, este puede ser usado en cualquier parte del documento.

```
/* scss: archivo var.scss */
$font-stack: Helvetica,sans-serif;
$primary-color: #333;

$base-color: #c6538c;
$border-dark: rgba($base-color, 0.88);


body  {
    font: 100% $font-stack;
    color: $primary-color;
}

.alert {
    border: 1px solid $border-dark;
}
```

Para procesar el documento `scss` (o `sass`, cuyas diferencias es el uso de `{}` y `;`), debe ejecutar:

```
$ sass --watch var.scss
```

El flag `--watch` permitirá que cada vez que se modifique el archivo **scss** se genere un nuevo css automaticamente.

```
/* css: archivo var.css*/
body {
  font: 100% Helvetica, sans-serif;
  color: #333; }

.alert {
  border: 1px solid rgba(198, 83, 140, 0.88);
}
```

Scope

Las variables declaradas en la parte superior del archivo son *globales*. Esto significa que pueden ser accedidas en cualquier parte del módulo, después de ser declarado. Existen también variables *locales*, que solo pueden ser usadas en los bloques en las que son declaradas.

```
/* scss: scope.scss */
$global-variable: global value;

.content {
  $local-variable: local value;
  global: $global-variable;
  local: $local-variable;
}

.sidebar {
  global: $global-variable;

  // This would fail, because $local-variable isn't in scope:
  // local: $local-variable;
}
```

```
/* css: scope.css */
.content {
  global: global value;
  local: local value; }

.sidebar {
  global: global value;
}
```

Las variables locales pueden ser declaradas con el mismo nombre que las globales. Si esto sucede, tendremos dos variables diferentes con el mismo nombre, una global y otra local. Esto es muy útil cuando por error se crean variables locales con el mismo nombre y cambiamos accidentalmente el valor de la variable global.

```
/* scss: variables.scss */
$new_variable: global value;

.alert {
    $new_variable: local value;
    value: $new_variable
}

.other {
    value: $new_variable
}
```

```
/* css: variables.css */
.alert {
  value: local value;
}

.other {
  value: global value;
}
```

Anidamiento o *Nesting*

Cuando trabajamos en nuestros archivos HTML, podemos ver claramente el anidamiento de las etiquetas, lo que no es claro muchas veces en CSS. Sass nos permite tener un mejor anidamiento para que podamos identificar nuestros estilos fácilmente de forma jerárquica como en nuestros HTML.

```scss
/* scss: nesting.scss */
nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
    }

    li {
        display: inline-block;
    }

    a {
        display: block;
        padding: 6px 12px;
        text-decoration: none;
    }
}
```

```css
/* css: nesting.css */
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Extender o Herencia

Esta es una de las más útiles funcionalidades de Sass. *@extend* nos permite compartir un conjunto de propiedades con uno u otro selector. En el ejemplo, se crearan dos bloques que podrian ser heredados por cualquier selector, solo *extenderemos* uno.

```scss
/* scss: extend.scss */
%message-shared {
    border: 1px solid #ccc;
    padding: 10px;
    color: #333;
}

%equal-heights {
    display: flex;
    flex-wrap: wrap;
}

.message {
    @extend %message-shared;
}

.success {
    @extend %message-shared;
    border-color: green;
}

.error {
    @extend %message-shared;
    border-color: red;
}

.warning {
    @extend %message-shared;
    border-color: yellow;
}
```

```css
/* css: extend.css */
.message, .success, .error, .warning {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
  border-color: yellow;
}
```

Vemos como `.message`, `.success`, `.error` y `.warning` tiene el mismo comportamiento que `%message-shared`, mientras que los estilos aplicados en `%equal-heights` no son colocados en el *css* final porque este no fue *extendido*.

Operadores

Sass ofrece operadores matemáticos (+, -, *, / y %) para realizar operaciones.

```
/* scss: operators.scss */
.container {
    width: 100%;
}

article[role="main"] {
    float: left;
    width: 600px / 960px * 100%;
}

aside[role="complementary"] {
    float: right;
    width: 300px / 960px * 100%;
}
```

```
/* css: operators.css */
.container {
  width: 100%;
}

article[role="main"] {
  float: left;
  width: 62.5%;
}

aside[role="complementary"] {
  float: right;
  width: 31.25%;
}
```

Conclusión

El uso de Sass como alternativa para precompilar los archivos css de nuestro proyecto nos ayuda a mantener un mejor orden y estructura cuando estos comienzan a crecer y hacerse más complejos para mantener.

Existen muchos frameworks que emplean por defecto Sass, como Angular, para el manejo de los estilos de sus componentes.

Referencias

- Sass: CSS with superpowers
- Sass Tutorial - W3schools

> Puede revisar los archivos del presente documento en el repositorio