Sass : tutoriel

@alioukahere, 21 déc. 2019

> Sass est le langage d'extension CSS de niveau professionnel le plus mature, le plus stable et le plus puissant au monde.

C'est ainsi que Sass se définit sur leur site web.

Sass (Syntactically Awesome StyleSheets), est un préprocesseur CSS. Il ajoute des fonctionnalités qui ne sont pas disponibles à l'aide de la syntaxe CSS de base. Sass permet aux développeurs de simplifier et de gérer plus facilement les feuilles de style de leurs projets.

Mais qu'est-ce que c'est qu'un préprocesseur?

> En informatique, un préprocesseur est un programme qui traite ses données d'entrée pour produire une sortie utilisée comme entrée d'un autre programme. — Wikipedia

C'est tout ce que fait un préprocesseur, nous écrivons notre code "Sass", le préprocesseur traduit ce code en CSS et c'est ce code CSS qui sera utiliser sur les pages web pas le code Sass.

Il existe deux syntaxes différentes de Sass. Le premier, appelé Scss (Sassy CSS) est une extension de la syntaxe du CSS. Les fichiers utilisant cette syntaxe ont l'extension `.scss`.

La deuxième et plus ancienne syntaxe, connue sous le nom de syntaxe indentée (ou parfois simplement "Sass"), utilise l'indentation plutôt que les crochets pour indiquer l'imbrication de sélecteurs, et les nouvelles lignes plutôt que les points-virgules pour séparer les propriétés. Les fichiers utilisant cette syntaxe ont l'extension `.sass`.

Nous allons dans ce tutoriel utiliser la première syntaxe. Nous allons donc créer des fichiers avec l'extension `.scss`.

Installation

Nous allons avant tout commencer par installer Sass en ligne de commande, c'est lui qui va nous permettre de traduire notre code Sass en CSS.

Pour installer Sass, il faut auparavant avoir npm et node installés, si ce n'est pas le cas, rendez-vous sur le site de node pour l'installer. Ensuite pour installer Sass, ouvrez un terminal et entrez la ligne suivante:

```
$ npm i -g sass
```

Une fois l'installation terminer, rentrez la commande suivante pour vérifier que l'installation a bien réussie:

```
$ sass --version
1.23.7 compiled with dart2js 2.6.1
```

Au moment de l'écriture de ce tutoriel, la version de Sass est `1.23.7`.

Utilisation

Pour commencer, rendez-vous dans votre dossier de travail et créez un fichier `style.scss`, ecrivez y ce code:

```
/* style.scss */

body {
  background-color: red;

  .container {
    width: 80%;
    max-width: 1200px;
    background-color: yellow;
  }
}
```

Ne cherchez pas à comprendre ce code pour l'instant.

Nous avons ici du code Sass (C'est juste du CSS, nous avons ici rien de méchant), il faut maintenant compiler ce code en CSS. Pour cela, ouvrez votre terminal et placez-vous dans le dossier dans lequel se trouve votre fichier `style.scss`, puis entre la commande:

```
$ sass style.scss
body {
  background-color: red;
}
body .container {
  width: 80%;
  max-width: 1200px;
  background-color: yellow;
}
```

Cette commande nous affiche le code CSS compilé à partir du fichier `.scss` qu'elle reçoit en paramètre. Pour écrire ce code dans un nouveau fichier, il faut donner le chemin du fichier comme second paramètre à la commande `sass` comme ceci:

```
$ sass style.scss style.css
```

Un fichier `style.css` va donc être créé dans le même dossier que le fichier `style.scss`, si vous regardez son contenu:

```
/* style.css */
```

```css
body {
  background-color: red;
}
body .container {
  width: 80%;
  max-width: 1200px;
  background-color: yellow;
}

/*# sourceMappingURL=style.css.map */
```

C'est du CSS, tout ce qu'il y a de plus ordinaire. Et c'est ce fichier que nous allons appeler dans notre page HTML.

La dernière ligne est plutôt intrigante, au fait quand on a compiler node code `sass`, un autre fichier a été créé, `style.css.map`:

```
{
  "version":3,
  "sourceRoot":"",
  "sources":["style.scss"],
  "names":[],
  "mappings":"AAAA;EACE;;AAEA;EACE;EACA;EACA",
  "file":"style.css"
}
```

Ce fichier contient des informations qui lient chaque ligne de notre fichier CSS à la ligne correspondante dans son fichier source Sass.

Variables

Les variables en Sass c'est exactement le même concept dans les autres langages de programmation comme le Javascript, Python, Java, … Ils servent à stocker des valeurs pour que celles-ci soient utilisés plus tard dans votre programme.

Imaginer juste le scénario, vous développez votre site web, vous avez une couleur primaire qui revient à chaque fois, disons par exemple votre couleur primaire est `#bada55`, vous devez l'utiliser dans la barre de navigation, le pied de page, sur les boutons, … Et maintenant plus tard dans le développement de votre projet, vous vous dites qu'il faut plutôt utiliser la couleur `#55bada` comme couleur primaire au lieu de `#bada55`, vous faites comment? Parcourir tout le fichier et modifier la couleur? Et si vous n'avez pas qu'un seul fichier? Appliquer le même scénario sur la couleur secondaire, la police, la taille de base du texte, … Ca devient tout de suite compliqué à gérer. C'est donc là qu'intervient les variables.

Une variable est composé d'un nom et d'une valeur. Pour définir une variable en Sass, on le précède d'un signe `$` comme ceci:

```
$nom-variable: valeur;
```

Et pour l'utiliser, on l'appelle juste pas son nom `$nom-variable`.

Le nom d'une variable ne doit jamais contenir d'espaces, vous pouvez soit utiliser le kebab-case, camelCase ou le snake_case:

```
$kebab-case: kebaba;
$camelCase: carameliser;
$snake_case: siiiffff;
```

Maintenant nous allons définir la couleur primaire et secondaire de notre application, puis la police à utiliser.

```
/* style.scss */

$primary-color: #bad555;
$secondary-color: #55bada;
$font-family: Roboto, sans-serif;

body {
  font-family: $font-family;
}

h1, h2, h3, h4, h5 {
  color: $primary-color;
}
```

C'est super simple en effet, on défini nos variables, puis on les appelles au besoin.

Nous allons le compiler en CSS toujours avec la commande:

```
$ sass style.scss style.css
```

Et le fichier CSS va donner:

```
/* style.css */

body {
  font-family: Roboto, sans-serif;
}

h1,
h2,
h3,
h4,
h5 {
  color: #bad555;
}

/*# sourceMappingURL=style.css.map */
```

Et si vous modifiez la valeur d'une variable et compiler le code à nouveau, vous verrez les changement dans le fichier CSS.

Une des premières erreur qui arrive souvent, c'est de modifier le fichier `style.scss` et ne pas le compiler pour obtenir le nouveau code CSS valide, puis espérer voir les changements sur la page, c'est une erreur qui arrive souvent, j'ai personnellement fait la même erreur la semaine dernière, en production et j'avais complètement oublié d'automatiser le build, j'ai cru que j'étais devenu fou jusqu'à ce que je m'en rende compte que je suis vraiment fou :-) rassurez-vous donc de toujours compiler le code Sass avant d'aller actualiser la page.

Pour éviter ce problème, vous pouvez dire à Sass d'écouter les changements sur votre fichier et de le compiler à chaque fois que vous le modifier, il suffit juste de rajouter l'argument `--watch` à la commande comme ceci:

```
$ sass --watch style.scss style.css
Sass is watching for changes. Press Ctrl-C to stop.
```

La deuxième ligne dis que Sass écoute les changements sur votre fichier et qu'il faut faire `Ctrl+C` pour l'arrêter.

Et maintenant dès que vous modifier le fichier `style.scss` et que vous l'enregistrer, la console vous affiche un message:

```
Compiled style.scss to style.css.
```

L'autre erreur que vous pourrez faire, c'est de modifier directement le fichier `style.css`, ce qu'il ne faut jamais faire au fait, parce qu'à chaque fois que vous compilez le fichier Sass, tout le contenu du fichier `style.css` (y compris ce que vous avez ajouté manuellement) est effacer et remplacer par le code compilé du Sass. Vous perdrez donc toutes les modifications que vous avez faites.

Imbrication de règles CSS

L'une des plus belles fonctionnalités de Sass, c'est l'imbrication du code CSS, de la même manière que vous imbriquer les balises HTML. Sachez que des règles trop imbriquées entraîneront un code CSS qui pourrait s'avérer difficile à maintenir et est généralement considérée comme une mauvaise pratique.

Si nous avons par exemple un élément `nav` qui définit la navigation de notre site, qui contient un élément `ul` et ainsi de suite. Le code Sass va donner:

```
/* style.scss */

$primary-color: #bad555;
$secondary-color: #55bada;
$font-family: Roboto, sans-serif;
$white: #fff;
```

```
* {
   padding: 0;
   margin: 0;
}

body {
  font-family: $font-family;
}

h1, h2, h3, h4, h5 {
  color: $primary-color;
}

nav {
   background-color: $secondary-color;
   padding: 20px;
   display: flex;
   justify-content: center;

   ul {
      li {
         display: inline-block;
         list-style-type: none;
         font-size: 1.2rem;

         &:not(:last-child) {
            margin-right: 20px;
         }

         a {
            color: $white;
            text-decoration: none;
         }
      }
   }
}
```

Et vous pouvez voir comment tout est bien imbriquer. Le code CSS compiler:

```
/* style.css */

* {
  padding: 0;
  margin: 0;
}

body {
```

```
  font-family: Roboto, sans-serif;
}

h1, h2, h3, h4, h5 {
  color: #bad555;
}

nav {
  background-color: #55bada;
  padding: 20px;
  display: flex;
  justify-content: center;
}
nav ul li {
  display: inline-block;
  list-style-type: none;
  font-size: 1.2rem;
}
nav ul li:not(:last-child) {
  margin-right: 20px;
}
nav ul li a {
  color: #fff;
  text-decoration: none;
}

/*# sourceMappingURL=style.css.map */
```

Mixins

Un mixin en Sass est un ensemble de règles CSS qui peut être utilisé partout sur votre fichier de style. Un mixin peut prendre un ou plusieurs paramètres et retourner un ensemble de règles CSS.

Une des utilisations des mixins peut par exemple être dans le cas où vous utilisez les nouvelles propriétés CSS comme `transform` ou toutes ces propriétés où il faut rajouter un préfixe pour chaque navigateur (`-webkit`, `-moz`, `-ms`). A chaque fois il faut spécifier tous les préfixes puis la règle CSS.

Pour definir un mixin en Sass, nous utilisons `@mixin` suivi du nom du mixin et de son contenu:

```
@mixin box-shadow($x, $y, $blur, $color) {
   -webkit-box-shadow: $x $y $blur $color;
   -moz-box-shadow: $x $y $blur $color;
   -ms-box-shadow: $x $y $blur $color;
   box-shadow: $x $y $blur $color;
```

```
}

@mixin transform($property) {
   -webkit-transform: $property;
   -ms-transform: $property;
   -moz-transform: $property;
   transform: $property;
}
```

Nous définissons deux mixins `box-shadow` qui prend 4 paramètres et `transform` qui prend un seul paramètre. C'est là tout l'intérêt des mixins. Pour utiliser un mixin, on fait:

```
.box {
   width: 200px;
   height: 200px;
   background-color: $primary-color;
   margin: 20px;
   @include box-shadow(10px, 10px, 15px, $secondary-color);

   &:hover {
      cursor: pointer;
      @include transform(skewY(-10deg));
   }
}
```

On l'appelle juste en utilisant `@include`.

Structures conditionnelles @if @else

Les structures conditionnelles en Sass fonctionnent comme dans tout les autres langages. Et pour l'utiliser on utilise le mot clé `@if` comme ceci:

```
@if $bool == true {
   h1 {
      color: red;
   }
}
```

Et nous avons aussi le mot clé `@else` pour le sinon.

Nous allons créer un mixin `text-color` qui prend un paramètre et attribue une couleur au texte en fonction de ce paramètre.

```
@mixin text-color($val) {
   @if $val == 'danger' {
      color: red;
   } @else if $val == 'success' {
      color: green;
```

```
   } @else if $val == 'info' {
      color: turquoise;
   } @else {
      color: black;
   }
}

p {
   @include text-color(success);
   margin: 20px;
}
```

La boucle @for

La directive `@for` permet d'exécuter une action `n` fois, disons 10 fois par exemple.

En Sass, la boucle `for` est utilisée de deux manières:

- `start to end` (1 to 10) qui exclut la borne supérieure. Dans le cas de 1 to 10 donc le 10 sera exclu, la boucle va tourner de de 1 à 9
- `start through end` (1 through 10), dans ce cas, la borne supérieure est incluse. Le parcours 1 through 10 se fera donc de 1 à 10

Disons que nous voulons par exemple créer un layout comme celui de Bootstrap avec 12 colonnes sous la forme col-1, col-2, … et que chaque classe doit avoir un width en fonction de l'espace qu'elle occupe. Nous allons utiliser une boucle `@for` pour cela:

```
@for $i from 1 through 12 {
   .col-#{$i} {
      width: 100% / 12 * $i;
      float: left;
      box-sizing: border-box;
      padding: 5px;

      div {
         background-color: $secondary-color;
         min-height: 200px;
         display: flex;
         justify-content: center;
         align-items: center;
         color: $white;
         font-size: 2rem;
         font-weight: bold;
      }
   }
}
```

C'est aussi simple que cela et super pratique.

Pour faire la concatenation, on utilise `#{$i}`.

Lists en Sass

En programmation, les listes sont des séquences de valeurs. Pour définir une liste en Sass, on sépare les éléments soit par une virgule ou un espace:

```
$colors: red, yellow, green;
$fonts-family: ['Roboto', 'Source Code Pro', 'Fira Code'];
```

Contrairement aux autres langages de programmation, ici l'index de la liste commence à 1 (oui j'ai aussi fait cette tête en le découvrant). Pour accéder a l'élément qui se trouve à l'index `n` on fait:

```
$colors: red, yellow, green;
$fonts-family: ['Roboto', 'Source Code Pro', 'Fira Code'];

body {
   background-color: nth($colors, 2);
   font-family: nth($fonts-family, 3);
}
```

Et pour ajouter un élément à une liste, on utilise `append($list, $elem)`:

```
$border: 2px solid;

h1 {
   border: append($border, red);
}
```

Les Maps en Sass

Les maps en Sass sont une association de clé valeur. Pour definir un map, nous allons faire:

```
$font-weights: ('thin': 200, 'regular': 400, 'medium': 500, 'bold': 700);
```

Et nous récupérons une valeur à travers sa clé avec `map-get($map, $key)`:

```
$font-weights: ('thin': 200, 'regular': 400, 'medium': 500, 'bold': 700);

p {
   font-weight: map-get($font-weights, 'bold');
}
```

@each map

La directive `@each` nous permet de parcourir une liste ou un map et a chaque itération de retourner un élément de la liste ou du map jusqu'à la fin.

```
$colors: red, yellow, green;
$font-weights: ('thin': 200, 'regular': 400, 'medium': 500, 'bold': 700);

@each $color in $colors {
   .text-#{$color} {
      color: $color;
   }
}

@each $key, $value in $font-weights {
   .text-#{$key} {
      font-weight: $value;
   }
}
```