Se lancer dans Sass

La simplicité de CSS est depuis toujours l'une de ses caractéristiques les plus agréables. Les feuilles de style CSS ne sont au fond que de longues listes de règles, chacune consistant en un sélecteur et quelques styles à appliquer. Mais plus nos sites deviennent riches et complexes, ciblant un éventail toujours plus large d'appareils et de tailles d'écran, et plus cette simplicité devient un handicap.

Il y a eu des propositions pour améliorer CSS – l'ajout de constantes ou de variables par exemple – mais aucune n'a été mise en oeuvre par les principaux navigateurs. Et même si l'un d'eux mettait en place un CSS nouveau, meilleur et étendu, il pourrait s'écouler encore des années avant que la nouvelle syntaxe soit supportée par tous les navigateurs et qu'on puisse effectivement l'utiliser.

Heureusement, il y a quelques années les développeurs Hampton Catlin et Nathan Weizenbaum ont proposé une meilleure façon de gérer des feuilles de style complexes. Puisque les navigateurs ne sont pas prêts pour un nouveau CSS, Catlin et Weizenbaum ont conçu une nouvelle syntaxe de feuilles de style permettant d'écrire et de gérer un CSS toujours plus complexe, en utilisant ensuite un préprocesseur (un programme qui fonctionne sur votre ordinateur ou sur un serveur) chargé de traduire cette nouvelle syntaxe intelligente dans le bon vieux CSS que nos navigateurs comprennent.

La nouvelle syntaxe de feuilles de style s'appelle Sass, pour "syntactically awesome style sheets" (feuilles de styles syntaxiquement super). Les premières versions de Sass étaient très différentes de notre CSS; il n'y avait pas d'accolades, et les propriétés devaient être indentées avec un nombre précis d'espaces sous peine de recevoir un message d'erreur du compilateur. Les programmeurs (qui sont habitués à apprendre de nouveaux langages et adorent la difficulté) n'y voyaient pas d'inconvénient, mais pour les designers web normaux la différence avec CSS était suffisante pour les maintenir à distance. Il y avait également un problème pratique : l'incompatibilité de cette syntaxe indentée avec CSS rendait

difficile la conversion de sites complexes, sauf à passer des heures à convertir l'ancien code en Sass.

C'est pourquoi Sass 3.0 a introduit une nouvelle syntaxe, plus proche de CSS, appelée SCSS ("Sassy CSS" qu'on pourrait traduire par "CSS à la Sass" ou bien "CSS classe"). SCSS est un "métalangage" de CSS, ce qui signifie que *tout ce qui est valide en CSS l'est aussi en SCSS*. En d'autres termes, toutes vos feuilles de style fonctionneront sans problème avec le préprocesseur de Sass, ce qui vous permettra de commencer à jouer tranquillement avec Sass sans en connaître encore toutes les facettes.

Contrairement à CSS, le SCSS est un vrai langage de programmation, avec des expressions, des fonctions, des variables, une logique conditionnelle et des boucles. Vous n'êtes pas obligés d'utiliser toutes ces caractéristiques pour tirer profit de Sass, mais elles sont là si vous en avez besoin, et sur de grands projets elles simplifient un CSS complexe ou répétitif.

Pour la simplicité, la plupart des exemples ci-dessous donnent la version SCSS du code plutôt que la version compilée en CSS. La meilleure façon de comprendre comment fonctionne Sass est d'essayer par vous-mêmes, en écrivant quelques styles en SCSS et en voyant le résultat. Cela dit, tous les exemples de code sont disponibles sur GitHub avec leur traduction compilée en CSS, donc vous pouvez voir comment toutes ces nouvelles caractéristiques sont traduites en feuilles de style utilisables.

On commence !

Sass est écrit en Ruby, et distribué via le package manager de Ruby, RubyGems. Si vous connaissez déjà Ruby (et la console de votre ordinateur), vous trouverez sur le site de Sass le mode d'emploi pour l'installation.

NdT : Ruby était notoirement lent. Depuis fin 2014, vous pouvez opter pour Libsass Sass sera compilé instantanément. Libsass est un nouveau compilateur Sass, écrit en C au lieu de Ruby.

Si vous êtes moins à l'aise avec la console, ou si vous souhaitez une façon simple et rapide de travailler avec Sass, il y a Scout, une application disponible pour Mac et Windows, qui contient déjà Ruby et le compilateur de Sass, et les utilisateurs de Mac ont CodeKit (1).

Quelle que soit la solution que vous choisirez, elle compilera automatiquement votre SCSS en CSS à chaque fois que vous enregistrerez votre fichier. Le répertoire où se trouvent vos fichiers Sass s'appelle le dossier input (entrée). Vos fichiers CSS processés sont sauvegardés dans un dossier output (sortie). Ces dossiers peuvent être emboîtés l'un dans l'autre; en fait, un schéma typique serait que le dossier input (que j'appelle "scss", mais vous pouvez l'appeler comme vous voulez) soit à l'intérieur du dossier feuilles de styles :

```
mon_projet/
  index.html
  css/
    style_principal.css
    scss/
      style_principal.scss
      _mixins.scss
      _couleurs.scss
```

Dans mon dossier scss ci-dessus, les fichiers dont le nom commence avec un tiret bas (underscore), par exemple `_mixins.scss`, sont appelés des partiels. Comme leur nom l'indique, ils contiennent des feuilles de style "partielles", des morceaux de code SCSS qui doivent être importés dans votre fichier SCSS principal. Je reviendrai sur les partiels dans un instant, mais maintenant vous savez où ils se trouvent.

Utilisez les partiels pour organiser votre code

La directive `@import` existe déjà dans CSS pour lier entre elles des feuilles de styles externes, et certains développeurs aiment l'utiliser pour organiser une feuille de style complexe en morceaux découpés logiquement. Par exemple, notre fichier `style_principal.css` pourrait ne contenir que des expressions `@import` liant des feuilles de style appliquées spécifiquement à chaque page d'un site :

```
//Syntaxe d'importation dans CSS

@import url('/partage/global.css');
@import url('/pages/accueil.css');
@import url('/pages/blog.css');
```

Cependant, ce n'est pas considéré comme une bonne pratique en termes de performance, car chacun de ces imports est un fichier séparé, qui déclenche une requête et doit être chargé par le navigateur, ce qui peut ralentir le chargement de votre site.

Sass est utile ici car il scinde les feuilles de style dans des partiels. On utilise là aussi la commande `@import` (sous une forme raccourcie), et quand vos fichiers SCSS sont processés, le contenu des partiels est inséré directement dans le CSS final.

```
// Syntaxe d'importation dans Sass

@import '/partage/global';
@import '/pages/accueil';
@import '/pages/blog';
```

Cela donne un fichier CSS unique contenant tous vos styles, et donc une requête unique. Pour optimiser encore le temps de chargement, Sass peut

aussi compacter automatiquement le fichier de sortie CSS en éliminant tous les espaces inutiles et les sauts de ligne.

Il y a un truc bien sûr : les partiels sont une espèce particulière de fichiers SCSS qui n'est pas destinée à être utilisée comme une feuille de style normale. Le code inclu dans un partiel doit être `@import` -é dans une feuille de style pour être utilisé. Les noms de fichiers de partiels doivent commencer par un underscore, par exemple le partiel pages/accueil importé ci-dessus devra être nommé `pages/_accueil.scss` (ces chemins sont toujours relatifs au fichier courant). C'est grâce à l'underscore que le compilateur de Sass reconnaît qu'il s'agit d'un partiel et que par conséquent il ne doit pas être compilé en une feuille de style CSS.

Ne vous répétez pas

Maintenant que nos feuilles de styles sont mieux organisées, essayons de les rendre moins répétitives.

L'une des caractéristiques les plus intéressantes de Sass est la possibilité d'imbriquer des règles. Dans un fichier CSS habituel, les règles (sélecteur/propriété/valeur) sont listées à la suite et chaque sélecteur doit inclure tous ses éléments :

```
//code CSS habituel

body.home .media-unit {
  border: 1px solid #ccc;
  background-color: #fff;
}
body.home .media-unit .right {
  border-left: 1px solid #ccc;
}
body.home .media-unit .right h1 {
  font-size: 24px;
```

```
}
```

Mis à part le fait qu'il est très très répétitif, ce code ne nous aide pas à comprendre les relations unissant les éléments HTML auxquels nous appliquons un style. Avec l'imbrication, nous pouvons écrire un code SCSS qui est à la fois non redondant et plus facile à suivre :

```
//code imbriqué

body.home {
  .media-unit {
    border: 1px solid #ccc;
    background-color: #fff;
    .right {
      border-left: 1px solid #ccc;
      h1 {
        font-size: 24px;
      }
    }
  }
}
```

Le processeur transforme ensuite ce code en CSS normal, identique à la version précédente. Malheureusement, la syntaxe SCSS plus courte ne réduira donc pas la taille de vos fichiers CSS. Mais l'imbrication vous aidera à avoir un code propre, logique et bien organisé, qui devrait être plus facile à entretenir au cours du temps.

Encore un truc sympa concernant l'imbrication : Sass vous permet d'imbriquer les media queries dans d'autres règles, ce qui facilite la visualisation des styles appliqués à un objet donné :

```
//Imbrication de media query
```

```
.container {
  width: 940px; // Si la largeur de l'écran est inférieure
  //  à 940px, passer à une présentation fluide
  @media screen and (max-width:940px) {
    width: auto; }
  }
}
```

Le processeur Sass sait comment convertir ce code en CSS valide, en copiant le sélecteur `.container` à l'intérieur du media query :

```
//Le résultat en CSS

.container {
  width: 940px;
}
@media screen and (max-width:940px) {
 .container {
   width: auto;
 }
}
```

Variables

Les variables Sass permettent de modifier le code plus facilement en réduisant les duplications et de nommer la valeur d'une propriété, la couleur par exemple, ce qui permet de comprendre l'intention derrière un style donné.

Par exemple, un certain nombre d'éléments de notre interface utilisateur ont une valeur de couleur de `#99cc00`, le *vert Maison*. Cette couleur apparaît des centaines de fois dans notre CSS, sur tout ce qui va des boutons aux couleurs de titres, et si nous devions modifier les couleurs de notre marque, ce serait un travail énorme. En utilisant des variables au lieu du code hexadécimal, rien de plus facile : on change la valeur de la variable (qui peut être définie en tête de votre feuille de style, ou même dans un partiel) et la nouvelle couleur apparaît partout, instantanément. Vous pouvez même attribuer la valeur de variables à d'autres variables, ce qui vous permet d'avoir des feuilles de style plus sémantiques :

```
//Valeur d'une variable attribuée à une autre variable

$vert-maison: "#99cc00";
$lien-color-vert-maison: $vert-maison;
a {
  color: $lien-color-vert-maison;
}
```

Mixins

Les Mixins sont des ensembles de propriétés ou de règles que vous pouvez inclure, ou "mixer", dans d'autres règles. Vous les définissez en

utilisant le mot clé `@mixin` et vous les intégrez en utilisant le mot clé `@include`.

Dans cet exemple, Sass va appliquer toutes les propriétés contenues dans le mixin `texte-surligne-gras` à tous les éléments span situés à l'intérieur de `.resultat-surligne` :

```
//Exemple de mixin

$couleur-surlignage: #ffa;
@mixin texte-surligne-gras {
  font-weight: bold;
  background-color: $couleur-surlignage;
}
.resultat-surligne {
  span {
    @include texte-surligne-gras;
  }
}
```

Une fois défini le mixin, vous pouvez le réutiliser n'importe où dans le même fichier. Ici, je veux que les éléments de classe `surligne` aient également la couleur et le font weight spécifiés par le mixin :

```
//Exemple de réutilisation du mixin
.surligne {  @include texte-surligne-gras; }
```

C'est vraiment très pratique quand on applique de nouvelles propriétés CSS à des éléments et cela permet d'assurer une large compatibilité pour tous les navigateurs via les préfixes constructeurs. Avec CSS, les préfixes sont pénibles car très verbeux, nous obligeant à de multiples copier-coller. Les mixins de Sass permettent d'utiliser de nouveaux styles sans écrire une ligne de code.

Ici, j'ai défini un mixin qui applique des coins arrondis de 4px à un élément en utilisant les propriétés préfixées pour WebKit, Firefox et IE, suivies par la propriété standard border-radius de CSS3. J'ai également défini le rayon comme une variable, afin de le changer facilement si besoin :

```
//Un exemple de mixin avec préfixes vendeurs

@mixin rounded-corners {
  $rounded-corner-radius: 4px;
  -webkit-border-radius: $rounded-corner-radius;
  -moz-border-radius: $rounded-corner-radius;
  -ms-border-radius: $rounded-corner-radius;
  border-radius: $rounded-corner-radius;
}
.button {
@include rounded-corners;
}
```

Les mixins peuvent aussi contenir des règles imbriquées, pas seulement des propriétés. Voici une version de la solution clearfix avec Sass :

```
//Clearfix, version Sass

@mixin clearfix {
// Pour les navigateurs modernes
&:before,
&:after {
  content:"";
  display:table;
  }
  &:after {
    clear:both;
  }

// Pour IE 6/7 (trigger hasLayout)
```

```
 &{
 zoom:1;
 }
}
.group {
 @include clearfix;
}
```

Le sélecteur esperluette (&) est une convention Sass qui signifie “cet élément”. Quand ce code est compilé, Sass remplace tous les symboles & par le sélecteur courant, dans ce cas `.group`.

Rendre les feuilles de style plus intelligentes

L’application de mixins à des feuilles de style simples est cool, mais ce qui les rend vraiment super c’est qu’elles peuvent prendre des arguments, tout comme une fonction en JavaScript ou en PHP. Vous pouvez les utiliser en combinaison avec des techniques plus avancées comme des expressions et des fonctions, pour organiser des styles complexes.

Les systèmes de grilles de mise en page sont une bonne application du langage Sass. Il existe sur le marché de nombreux systèmes de grilles 960px prêts à l’usage, mais la plupart demandent d’ajouter à votre code des noms de classes non sémantiques. Sans parler du fait que pour les utiliser, vous devez charger le CSS du système entier, mais vous n’en utiliserez au final qu’une partie.

Pour notre dernier exemple, nous allons mettre en place une simple grille à 12 unités. Plutôt que de définir des noms de classes pour chaque unité de la grille, nous allons créer un mixin qui applique la largeur et les marges correctes.

Tout d’abord, nous devons définir la largeur de nos colonnes et nos gouttières dans une variable :

```
//définition de la largeur des colonnes et des gouttières

$column-width: 60px;   // 12 colonnes = 720px
$gutter-width: 20px;   // 11 gouttières = 220px
```

Ensuite, Sass va faire un peu de maths pour nous. À l’intérieur de notre grille, une unité s’étendra sur un certain nombre de colonnes plus toutes les gouttières entre ces colonnes. Pour en calculer la largeur, nous utiliserons la formule suivante :

width = (column-width-span) + (gutter-width-(span – 1))

Nous pouvons maintenant écrire notre mixin. Contrairement aux exemples précédents, notre mixin prendra un argument (le “span”, c’est à dire l’étendue, mesurée en nombre de colonnes) qui sera passé dans notre mixin en tant que variable. Chaque unité de la grille flottera à gauche, et pour maintenir la gouttière de 20px entre les colonnes, chaque unité aura une marge à droite égale à la largeur de la gouttière :

```
//Simple Grille en Sass
@mixin grid-unit($span) {
  float: left;
  margin-right: $gutter-width;
  width: ($column-width * $span) + ($gutter-width * ($span - 1));
}
```

Simple, n’est-ce pas ? Mais ces quelques lignes renferment une grande puissance. Nous pouvons élaborer une mise en page basique 2/3 avec un “main content” et une “sidebar’, en utilisant des classes nommées de façon plus sémantique et notre mixin Sass :

```
//Mise en page

.container {
  @include clearfix;
  @include grid-unit(12);
  float: none;
  margin: 0 auto;
}
.main-content {
  @include grid-unit(8);
}
.sidebar {
  @include grid-unit(4); margin-right: 0;
}
```

Aussi intéressant que cela soit déjà – et j'utilise une version de ce calculateur de grille sur chacun de mes projets aujourd'hui – nous ne faisons que gratter la surface de ce qu'il est possible de faire avec Sass.

Avec les fonctions mathématiques de Sass il est facile de créer des présentations fluides. Ici, j'utilise la formule proposée par Ethan Marcotte pour créer une version responsive de ma grille de base. Sass ne convertit pas les unités CSS entre elles, j'ai donc inclu mes formules dans la fonction Sass intégrée `percentage()` :

```
//version responsive de la grille précédente

.container {
// result = target / context
// résultat = cible divisée par contexte
  width: percentage(940px / 960px);
  .main-content {
// cet élément est imbriqué dans .container,
// donc son contexte est 940px
    width: percentage(620px / 940px);
```

```
  }
  .sidebar {
    width: percentage(300px / 940px);
  }
}
```

Il existe aussi des fonctions permettant de transformer et d'ajuster les couleurs – vous pouvez les rendre plus claires ou plus foncées, ajuster la saturation ou la transparence, à l'intérieur même de votre feuille de style :

```
//Transformation et ajustement des couleurs

$base-link-color: #00f;
a {
  color: $base-link-color;
}
a:visited {
// réduit la luminosité (en termes HSL)
// de 50%, sans modifier la teinte ni la saturation
  color: darken($base-link-color, 20%);
}
figcaption {
// Génère une valeur rgba() de couleur avec 50% d'opacité
  background-color: transparentize(#fff, 50%);
}
```

Et si ces fonctions ne suffisent pas, vous pouvez définir les vôtres, les partager et les réutiliser dans d'autres projets grâce aux partiels. Jetez un coup d'oeil à la liste complète des fonctions intégrées Sass pour avoir une idée de ce qui est possible.

Et après ?

Le site officiel de Sass regorge d'informations utiles, dont un guide de référence complet du langage SCSS. Vous pouvez commencer avec cette liste de plugins SCSS disponibles pour les éditeurs de texte, qui vous offre quelques outils pour écrire en Sass avec l'éditeur de texte de votre choix.

De plus, le créateur de Sass Hampton Catlin a écrit un guide complet de Sass.

Au-delà du langage Sass, il y a aussi Compass, une bibliothèque de fonctions SCSS développée par Chris Eppstein, qui comprend le reset CSS d'Eric Meyer, le système de grille Blueprint, et des tonnes d'effets typographiques et CSS3. Leurs auteurs l'ont appelé "jQuery pour feuilles de styles", ce qui est une description assez appropriée. Si vous utilisez l'application Scout, le framework Compass est déjà installé. Sinon, vous trouverez les instructions pour son installation et son utilisation sur le site de Compass.

Ressources complémentaires en français

- Les préprocesseurs CSS, c'est senSass, par alsacréations
- Présentation de quelques concepts de Sass par Excilys Labs.
- Pourquoi Sass et Compass sont des choix judicieux, par Human coders.
- Sass, augmentez votre productivité par le blog du webdesign.
- Domptez vos CSS avec Sass, par Seemios Blog
- Sass: tout sur @extend, par Hugo Giraudel
- Sass: mixin ou placeholder ?, par Hugo Giraudel
- Sass et interpolation, par Hugo Giraudel
- Sass Guidelines, par Hugo Giraudel