Manual de Sass

Miguel Angel Alvarez

desarrolloweb.com

desarrolloweb.com/manuales/manual-sass.html

Introducción: Manual de Sass

Este manual nos introduce en Sass, ofreciendo una guía completa para aprender a trabajar con este preprocesador de CSS.

El Manual de Sass te llevará paso a paso, explicando no solo la sintaxis específica de este preprocesador, sino también el flujo de trabajo que puedes implementar en el desarrollo de tus CSS apoyándote en esta estupenda herramienta.

Sass se ha convertido en una herramienta altamente extendida y apoyada por la comunidad, que nos permite optimizar nuestro tiempo a la hora de desarrollar sitios web. Si deseas saber por qué, este manual te podrá ayudar y te permitirá comenzar a usar Sass en tus proyectos.

Encuentras este manual online en:
http://desarrolloweb.com/manuales/manual-sass.html

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.

Qué es Sass, como usar Sass

Conoce el preprocesador CSS Sass y aprende a usar Sass para optimizar tu trabajo de desarrollo de sitios web.

Comenzamos con este artículo el Manual de Sass, en el que vamos a explicarte las bases para que puedas usar Sass y beneficiarte de todas las ventajas de los preprocesadores CSS. Sass es el más usado de los preprocesadores y una de los requisitos habituales que encontrarás en ofertas de trabajo.

En este manual te explicaremos la sintaxis de Sass y el uso en el marco del desarrollo de sitios web, de modo que puedas automatizar la traducción del código Sass a código CSS estándar. Pero antes de ponernos con el propio Sass conviene darnos un poco de tiempo para aclarar qué son los preprocesadores CSS.

Qué son los preprocesadores CSS

Los preprocesadores CSS son herramientas para los desarrolladores de sitios web, que permiten traducir un código de hojas de estilo no estándar, específico del preprocesador en cuestión, a un código CSS estándar, entendible por los navegadores.

Los preprocesadores básicamente nos ofrecen diversas utilidades que a día de hoy no se encuentran en el lenguaje CSS, o bien no son compatibles con todos los navegadores. Ejemplos pueden ser variables, anidación de selectores, funciones (denominadas mixins), etc.

Al desarrollar con un preprocesador se consigue principalmente un ahorro de tiempo, ya que tenemos que escribir menos código para hacer las cosas. Pero también conseguimos una mayor facilidad de mantenimiento del código, gracias a una mejor organización del código y la posibilidad de editar una vez ciertos valores y que afecten a decenas, o cientos, de lugares del código CSS generado.

No todo son ventajas cuando hablamos de preprocesadores, ya que nos obligan a aprender un nuevo lenguaje y escribir código no estándar. Ese código no estándar debe ser compilado en

CSS, lo que afecta también al flujo de desarrollo. El código del preprocesador no lo entiende el navegador y por lo tanto nos tenemos que dar el trabajo de traducirlo, antes de llevar un sitio web a producción. Afortunadamente, ese flujo está bastante depurado y existen diversas alternativas para la optimización de esa traducción, de modo que no tengamos que intervenir manualmente y nos ahorre tiempo.

Por qué Sass

Cualquier preprocesador es perfectamente válido. Podríamos sin duda elegir otras alternativas como Less o Stylus y estaría estupendo para nosotros y nuestro proyecto, ya que al final todos ofrecen más o menos las mismas utilidades. Pero sin embargo, Sass se ha convertido en el preprocesador más usado y el más demandado.

Al haber recibido un mayor apoyo de la comunidad es más factible que encuentres ofertas de trabajo con Sass o que heredes proyectos que usan Sass, por lo que generalmente te será más útil aprender este preprocesador.

Además, varios frameworks usan Sass, como el caso de Bootstrap. Por ello, si tienes que trabajar con ellos te vendrá mejor aprender Sass.

Cómo es la sintaxis de Sass

Como decimos, la tarea de aprender Sass se divide en dos bloques principales. Primero aprender a trabajar con el código de Sass y conocer su sintaxis especial. La segunda tarea sería aplicar Sass en tu flujo de trabajo, de modo que puedas usar el preprocesador fácilmente y no te quite tiempo el proceso de compilación a CSS. Veremos las dos partes en este primer artículo del Manual de Sass, comenzando por examinar su sintaxis.

Obviamente, no vamos a poder explicar la sintaxis de Sass en unas pocas líneas. Para eso tenemos el manual completo, pero queremos comenzar dando algunos ejemplos para que veas cómo es.

Lo primero que debes conocer de Sass es que existen dos tipos de sintaxis para escribir su código:

- **Sintaxis Sass**: esta sintaxis es un poco diferente de la sintaxis de CSS estándar. No difiere mucho. Por ejemplo, te evita colocar puntos y coma al final de los valores de propiedades. Además, las llaves no se usan y en su lugar se realizan indentados.
- **Sintaxis SCSS**: Es una sintaxis bastante similar a la sintaxis del propio CSS. De hecho, el código CSS es código SCSS válido. Podríamos decir que SCSS es código CSS con algunas cosas extra.

En la práctica, aunque podría ser más rápido escribir con sintaxis Sass, es menos recomendable, porque te aleja más del propio lenguaje CSS. No solo es que te obligue a aprender más cosas, sino que tu código se parecerá menos al de CSS estándar y por lo tanto es normal que como desarrollador te sientas menos "en casa" al usar sintaxis Sass. En cambio, al usar SCSS tienes la ventaja de que tu código CSS de toda la vida será válido para el preprocesador, pudiendo reutilizar más tus conocimientos y posibles códigos de estilo con los que vengas trabajando en los proyectos.

Como entendemos que al usar un preprocesador es preferible mantenerse más cerca del lenguaje CSS, **en este manual usaremos siempre sintaxis SCSS**.

Ahora veamos algunas cosas sobre la sintaxis de Sass. Por ejemplo, el uso de variables.

```
$color-fondos: #F55;

h1 {
  background-color: $color-fondos;
}
```

En este código estás viendo cómo declaramos una variable y a continuación la usamos en un selector. Obviamente, eso no es un CSS válido, ya que no existen este tipo de declaraciones en el lenguaje.

> **Nota:** Hoy CSS ya incorpora variables, aunque se usa otra sintaxis. El problema es que no están disponibles todavía en todos los navegadores, por lo que no es totalmente seguro usarlas, a no ser que implementes PostCSS con CSS Next.

Una vez compilado el código anterior, obtendrías un código estándar como el siguiente:

```
h1 {
  background-color: #F55; }
```

Veremos más sobre variables y muchas otras utilidades de Sass en breve. Pero ahora preferimos dedicar algo de tiempo para explicarte cómo puedes usar Sass en tu proyecto.

Cómo usar Sass

Para usar Sass tienes dos alternativas fundamentales.

1. **Preprocesar con alguna herramienta de interfaz gráfica**, como el caso de Prepros, CodeKit o Scout-App. En principio puede ser más sencillo, ya que no requiere trabajar con la línea de comandos, pero necesitas usar un programa en concreto y no siempre puede estar disponible para ti, o no integrarse en otro flujo de trabajo que puedas tener ya asumido en tu proyecto. Además, las mejores herramientas de interfaz gráfica tienen la desventaja de ser de pago, o necesitar una licencia para desbloquear todo su poder.
2. **Usar la línea de comandos para preprocesar**. Esta es la opción preferida por la mayoría de desarrolladores. No sólo porque no requiere la compra de una licencia por el software de interfaz gráfica, sino principalmente porque la puedes integrar con todo un ecosistema de herramientas para optimizar multitud de aspectos en un sitio web. Además, está al alcance de cualquier desarrollador, ya que todos tenemos un terminal en nuestro sistema operativo y finalmente, permite personalizar completamente el

comportamiento del preprocesador.
3. **Usar herramientas de automatización**. Como tercera opción es muy común también usar herramientas que permiten automatizar el flujo de trabajo frontend, compilando archivos CSS, Javascript, optimizando imágenes, etc. Nos referimos a paquetes como Gulp, Grunt o Webpack (aunque este último es más un empaquetador). Estos sistemas tienen la particularidad que sirven para cubrir todas las necesidades de trabajo con los lenguajes de la web y no solamente se usan para compilar el código Sass. Sería la opción más potente de las tres comentadas, aunque requiere mayor formación para poder usarlas. En este grupo también podríamos unir a PostCSS, aunque esta herramienta solamente nos sirve para convertir el CSS, aplicando diversos plugins, entre los que podría estar la compilación de Sass, y no entra en áreas como el Javascript.

Las anteriores alternativas pueden marear un poco si no estás acostumbrado a oir acerca de tantas tecnologías. No te preocupes demasiado porque vamos a irnos a una opción sencilla que te permita comenzar a usar Sass, sin demasiado esfuerzo ni configuración. En concreto te vamos a explicar ahora a usar Sass desde tu terminal, con las herramientas "oficiales" del propio preprocesador (la alternativa planteada en el punto 2 anterior). Es una posibilidad al alcance de cualquier lector. Nuestro siguiente paso entonces será comenzar por instalar el preprocesador.

Instalar Sass

La instalación de Sass depende del sistema operativo con el que estás trabajando. Aunque todos requieren comenzar instalando el lenguaje Ruby, ya que el compilador de Sass está escrito en Ruby.

Veamos, para cada sistema, cómo hacernos con Ruby.

- **Windows**: Existe un instalador de Ruby para Windows. https://rubyinstaller.org/ Puedes usarlo para faciltar las cosas.
- **Linux**: tendrás que instalar Ruby usando tu gestor de paquetes de la distribución con la que trabajes, apt-get, yum, etc.
- **Mac**: Ruby está instalado en los sistemas Mac, por lo que no necesitarías hacer ningún paso adicional.

Una vez instalado el lenguaje Ruby tendremos el comando "gem", que instala paquetes (software) basados en este lenguaje. Así que usaremos ese comando para instalar Sass. De nuevo, puede haber diferencias entre los distintos sistemas operativos.

Windows

Tendrás que abrir un terminal, ya sea Power Shell, "cmd" o cualquiera que te guste usar. Si no usas ninguno simplemente escribe "cmd" después de pulsar el botón de inicio. Luego tendrás que lanzar el comando.

```
gem install sass
```

Linux

Para instalar Sass, una vez tienes Ruby anteriormente, podrás hacerlo con el siguiente comando en tu terminal.

```
sudo gem install sass --no-user-install
```

Mac OS X

En Mac usarás generalmente "sudo", igual que en Linux. Aunque puedes probar antes sin sudo y usar sudo si realmente te lo pide la consola por no tener los suficientes permisos.

```
sudo gem install sass
```

> **Nota importante para usuarios de Mac:** En Mac OS X es posible que tengas que hacer algunas otras operaciones. En una instalación limpia de Mac quizás tengas que instalar las extensiones de Xcode: `xcode-select --install` O bien instalar el propio Xcode desde el App Store (no te preocupes, porque es gratuito). Luego además tendrás que abrir el Xcode, una vez instalado, para completar la instalación de todos los componentes necesarios.

Para cualquier sistema, una vez instalado Sass, podemos ver si realmente está disponible, haciendo el siguiente comando:

```
sass -v
```

Eso nos debería devolver la versión de Sass instalada en nuestro sistema.

```
) sass -v
Sass 3.5.5 (Bleeding Edge)
```

Compilar archivos de Sass a CSS

Ahora que tienes Sass instalado querrás compilar el código de Sass para convertirlo a CSS estándar. Para ese paso, necesitamos partir de un archivo con código Sass.

Antes en este artículo hemos visto cómo podría ser un pedazo de código en Sass, con sintaxis SCSS. Puedes copiar ese código y guardarlo en un archivo en tu ordenador. Usa cualquier carpeta para hacer esta primera prueba y guarda el archivo con el nombre "ej.scss" (o cualquier otra cosa que te apetezca).

Ahora te debes situar con el terminal en la carpeta donde tengas el archivo que acabas de crear con el código SCSS. Entonces lanzarás el comando para compilar, de esta manera:

```
sass ej.scss ej-compilado.css
```

Como puedes observar, al comando "sass" le indicamos el archivo origen (con código SCSS) y el archivo de destino, donde nos colocará el código CSS estándar.

Podrás observar también, una vez ejecutado el comando, que nos crea el archivo "ej-compilado.sass". Además que ha creado un archivo ej-compilado.css.map, que sirve para que el navegador pueda corresponder (en las herramientas de desarrollo) el código compilado con la referencia donde está el código original, permitiendo ayudarte a la hora de depurar. Hablaremos de mapas más adelante, de modo que no necesitas preocuparte de ese segundo archivo ".map".

Compilar Sass de manera automática con un "watcher"

Habrás observado lo rápido que se ha compilado el archivo Sass para convertirse en código CSS estándar. Pero sin embargo, ejecutar ese comando para compilación cada vez que cambia tu código es un poco tedioso, porque te obliga ir a la consola y lanzar nuevamente el comando con cada pequeña edición que hagas.

Lo que generalmente querrás es tener un vigilante "watcher" que se encargue de compilar automáticamente el fichero cada vez que guardes el archivo original. Así, tu código compilado estará siempre fresco y podrás ahorrar unos segundos preciosos con cada compilación. Al cabo del día significará mucho tiempo y optimizará tu trabajo, haciéndote más feliz.

Paralelamente, el watch nos permite observar un archivo en concreto, o todos los archivos de una carpeta.

Vigilar las modificaciones de un archivo en concreto

Lo consigues con la opción --watch, indicando el archivo origen y destino, igual que hacíamos antes.

```
sass --watch origen.scss destino.css
```

Vigilar todos los archivos de una carpeta

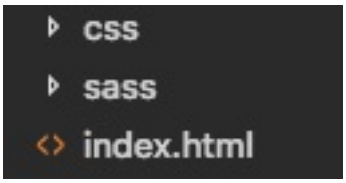
En este caso puedes decirle a Sass la carpeta de origen y la carpeta de destino, donde colocará el código CSS estándar. Separas la carpeta origen de la carpeta de destino con el caracter dos puntos ":".

```
sass --watch carpetaorigen:carpetadestino
```

Ten en cuenta que esas carpetas existirán en tu proyecto y que el comando lo debes invocar desde la raíz del proyecto.

Ya los nombres de las carpetas podrán variar según tus preferencias. Para comenzar, si no tienes ningún requisito en este sentido, te sugerimos crear dos carpetas. Una llamada "sass" donde colocarás tu código fuente, con archivos de extensión ".scss". Otra llamada "css", donde se colocará el código compilado.

El esquema de carpetas sería parecido a esto:



Ahora, lanzarás el watcher con un comando como este (Situado en la raíz del proyecto, donde has visto que está el archivo index.html).

```
sass --watch sass:css
```

Una vez que crees código SCSS en la carpeta "sass", se irán generando los archivos compilados en la carpeta de destino. Por ejemplo, así te quedaría la estructura de archivos y carpetas al crear tu código Sass en la ruta "sass/estilo.scss".

cuando abordemos la organización del código mediante diversos archivos fuente de SCSS, explicaremos cómo puedes evitar que se compilen todos los archivos de Sass y sólo se compile uno de ellos, el raíz. Te adelantamos que puedes evitar que el watcher los procese simplemente nombrando esos archivos con un guión bajo en su inicio. Algo como "_variables.css".

Usar los archivos de código generado

Como hemos dicho, el código SCSS no es compatible con los navegadores, por lo que tenemos que usar el código compilado. Esto quiere decir que, en tu index.html, así como en cualquier otro archivo HTML de tu proyecto, tendrás que enlazar con el código generado y nunca con el código fuente Sass.

Por tanto, en el index.html que has visto en las anteriores imágenes, tendríamos que enlazar con la hoja de estilos compilada, de esta manera:

```
<link rel="stylesheet" href="css/estilo.css">
```

Si has hecho todo bien, deberías ver como te pilla los estilos generados y funciona todo perfectamente.

Con esto ya has aprendido a dar los primeros pasos con Sass y has hecho lo más difícil, crear un entorno de desarrollo amigable, que permita la compilación automática de los archivos Sass cada vez que vamos introduciendo cambios.

Ahora nos queda aprender bien el lenguaje y sacarle todo el partido, manteniendo las buenas prácticas. En los próximos artículos del manual iremos abordando todos los puntos que debes saber, así como otras aproximaciones para compilar Sass que te pueden venir bien en otros casos.

Cómo aprender Sass

Ovbiamente, estamos produciendo este manual para que puedas aprender Sass desde DesarrolloWeb.com. Iremos agregando nuevos artículos para completar los temas básicos que debes saber para usar Sass en tus proyectos. Pero además, ten siempre a mano la propia documentación de Sass, que te servirá como una vía rápida de despejar tus dudas.

Además, si quieres aprender de manera rápida y completa, también te queremos recomendar el Curso de Sass de EscuelaIT. Son 5 horas de clase en la que aprenderás no solo a preprocesar, sino las mejores prácticas para sacarle el mejor partido al preprocesador y organizar el código de tu proyecto.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *08/11/2018*
Disponible online en *http://desarrolloweb.com/articulos/que-es-sass-usar-sass.html*

Variables en Sass

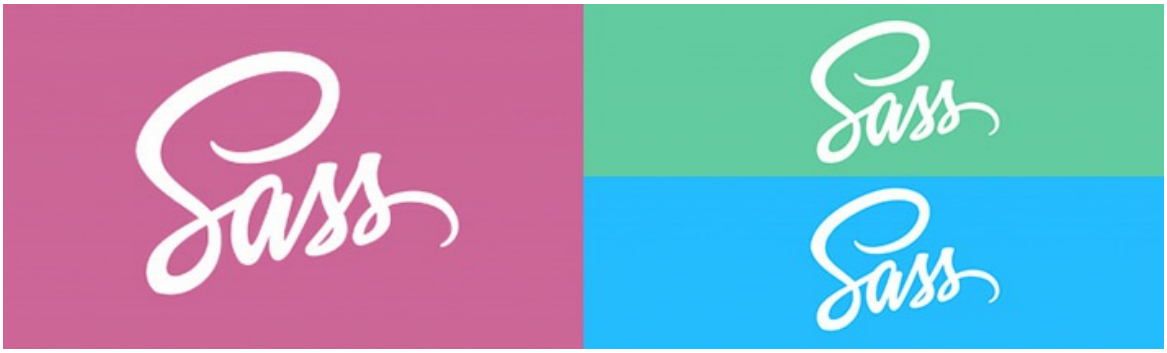
Veamos lo primero que se debe aprender en Sass, a crear y usar variables de CSS, para disponer de un código más fácilmente mantenible.

Después de aprender a usar Sass en el artículo anterior, vamos a dedicarnos a conocer una de sus principales y más sencillas utilidades, las variables.

En CSS siempre se echó de menos la posibilidad de usar variables. Los preprocesadores fueron los primeros en incorporarlas y por supuesto, Sass nos las ofrece para facilitar nuestro día a día con el uso del CSS.

Aunque hoy **ya podemos usar variables de CSS**, pues se han agregado recientemente al estándar, seguimos con un problema fundamental, que **el código con esas variables no será compatible con todos los navegadores**. Motivo por el cual **sigue siendo interesante apoyarse en los preprocesadores**.

Las variables son, como en los lenguajes de programación tradicionales, capaces de almacenar un dato, que luego podremos usar las veces que haga falta a lo largo de todo nuestro código Sass.



Por qué son útiles las variables

Pensemos en una declaración de un color. Por ejemplo el color de texto destacado, o el color de texto que se puede usar para representar un error. Seguramente querramos usar ese mismo color en diversos momentos, asignando a diversos selectores.

La opción habitual sería escribir ese color en todos los lugares donde se necesite, repitiendo una y otra vez el mismo valor RGB aquí y allá. Sin embargo, esto puede producir situaciones poco atrayentes.

- Si es un RGB complicado, algo como #3F6DB8, es posible que no nos acordemos de ese valor cada vez que lo queramos usar, teniendo que ir a otra definición de ese mismo

color, copiando y pegando donde se vuelva a necesitar.
- Aún peor, si más adelante se pretende cambiar ese color por otro. Tendríamos que buscar por todo el proyecto los lugares donde se usó para sustituirlo en todos ellos.

Lo mismo explicado con colores puede ocurrir con tipografías, tamaños de fuente, márgenes, etc.

En cambio, si usamos una variable, podríamos solucionar estas situaciones de una, facilitando la codificación y el mantenimiento del código CSS, creando diseños consistentes, que reflejen correctamente los colores corporativos de cada cliente.

Cómo se definen variables en SCSS (Sass)

Ahora veamos cómo podemos definir variables para el preprocesador Sass.

> **Nota:** Recuerda que en este manual usamos siempre sintaxis SCSS, aunque Sass podría usar también otro tipo de sintaxis.

Las variables se definen igual que otros atributos CSS, solo que no están ligados a un selector en concreto. Además, en Sass las variables comienzan siempre con el caracter "[[--body--]]quot;.

```
$color-primario: #55A;
$color-secundario: #6B6;
$color-texto: #666;

$espaciado-estandar: 10px;
$espaciado-doble: 20px;

$fuente-normal: 1em;
$fuente-pequena: 0.8em;
$fuente-grande: 1.4em;

$tipografia-general: arial, verdana, sans-serif;
$tipografia-alternativa: 'Times New Roman', Times, serif;
```

Como puedes ver en este código, es una buena idea definir en variables aspectos estéticos, que se van a usar a lo largo de todo el código CSS. Eso también ayudará a obtener un código CSS más semántico, fácil de leer y entender por humanos.

Cómo se usan variables en Sass

Ahora veamos cómo podemos usar las variables definidas en el punto anterior. Realmente es bien sencillo, ya que simplemente tendremos que usar el mismo nombre de la variable.

Las variables las usaremos como valores de atributos CSS. Esas propiedades de CSS tendrán

que estar ligados a un selector, igual que siempre en las hojas de estilo en cascada. En el selector podremos mezclar valores de atributos que vengan definidos en variables, con otros valores especificados directamente sin usar variables.

Este código SCSS hace uso de muchas de las variables definidas anteriormente.

```
body {
  color: $color-texto;
  font-family: $tipografia-general;
}

h1 {
  background-color: $color-primario;
  color: $color-blanco;
  font-family: $tipografia-alternativa;
  font-size: $fuente-grande;
  padding: $espaciado-estandar;
  text-transform: uppercase;
  border-radius: 12px;
}

div.destacado {
  border: 1px solid $color-secundario;
  padding: $espaciado-estandar $espaciado-doble;
}
```

Cómo es el código CSS generado

Para acabar, puedes ver el código CSS generado, aunque estoy seguro que no tendrías mucho problema para imaginarlo.

Todo lo que son definiciones de variables se pierde, ya que en el código CSS no se necesitan definir. Solo nos quedan los valores de las variables que se llegaron a aplicar en los atributos de los selectores usados.

```
body {
  color: #666;
  font-family: arial, verdana, sans-serif; }

h1 {
  background-color: #55A;
  color: #fff;
  font-family: "Times New Roman", Times, serif;
  font-size: 1.4em;
  padding: 10px;
  text-transform: uppercase;
  border-radius: 12px; }

div.destacado {
  border: 1px solid #6B6;
  padding: 10px 20px; }
```

Las variables en sí no tienen mucho más que explicar, solo mencionar que es interesante que las uses bastante, ya que con el tiempo tu código CSS crecerá bastante y si has definido un buen conjunto de variables serás capaz de mantener esos valores cómodamente a lo largo de todo tu CSS.

Incluso podrás llevarte las definiciones de variables de un proyecto a otro, cambiando sus valores para adaptarse al nuevo proyecto, con lo que el esfuerzo de definición de esas variables y su utilización podrá rentabilizarse a lo largo del tiempo, en muchos proyectos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *06/04/2018*
Disponible online en *http://desarrolloweb.com/articulos/variables-sass.html*

Anidación de selectores en Sass

El código Sass es más fácil de producir y requiere menos escritura gracias a la posibilidad de anidar selectores CSS.

HTML como lenguaje requiere una organización de las etiquetas en forma de árbol. Gracias a ello, en la lectura del código HTML somos capaces rápidamente de saber qué cosas están dentro de otras.

Sin embargo, a la hora de escribir CSS observarás que todo el código es bastante más plano. En principio no es un problema, pero gracias a la anidación de selectores en el código de Sass somos capaces de escribir menos y conseguir un código que luego será más sencillo de leer por humanos y más fácil de mantener.

Las ventajas de la anidación, que explicaremos a continuación, son tales que el estándar de CSS ya las tiene previsto incluir. Pero como ocurría con las variables, el soporte a día de hoy es prácticamente nulo, así que debemos usar algún tipo de herramienta que convierta el código a algo entendible por los navegadores.

Por qué es útil anidar el código CSS

Como hemos aprendido a lo largo del Manual de Sass, las ventajas de los recursos que el preprocesador nos ofrece, son casi siempre enfocadas a una menor escritura de código y un mejor mantenimiento. Con anidación en el CSS tendrás que escribir menos, pues tendremos selectores que son menos largos y menos repetitivos.

Esto lo podemos ver con un ejemplo más claramente.

Imagina que quieres definir un tipo de caja como "destacada". Ese tipo de caja puede tener encabezados y además dentro podría tener enlaces. Para definir todas esas cosas podrías escribir un HTML como este.

```
<div class="cajadestacada">
  <header>Este es el encabezado <a href="#">Enlace encabezado</a> </header>
  <p>Lorem ipsum...</p>
</div>
```

Ahora, para aplicar estilos a este elemento, podríamos usar un código estándar como este:

```
.cajadestacada {
  background-color: red;
}

.cajadestacada header {
  background-color: black;
  color: #fff;
}

.cajadestacada header a {
  color: #ff6;
}
```

Con la calificación de código "plano", lo que queremos hacer notar es que, cada vez que queremos definir un estilo para algo que va dentro del elemento de clase "cajadestacada", tenemos que escribir el selector, y luego los selectores de los elementos a los que queremos llegar, repitiendo constantemente eso de ".cajadestacada".

Quizás tampoco es un problema muy crítico, pero los preprocesadores nos ayudan a escribir menos código y conseguir ser más rápidos. No solamente la primera vez que se codificó, sino todas las veces que vamos a mantener este CSS.

Cómo anidar selectores en Sass

Como sabes, vamos a explicar la sintaxis SCSS de Sass, que nos permite como alternativa una sintaxis muy próxima a la sintaxis del propio CSS.

A continuación veamos un código donde anidamos los selectores. Esa anidación es similar a la que tenemos en el HTML, donde unas etiquetas están dentro de otras.

```
.cajadestacada {
  background-color: red;

  header {
    background-color: black;
    color: #fff;

    a {
      color: #ff6;
    }
  }
}
```

Ahora nuestro código es menos plano y concuerda más con la propia estructura del HTML que estamos estilizando. Además que hemos podido escribir menos código.

El resultado, una vez compilado, es parecido al que te podrías imaginar, y que hemos colocado como ejemplo al principio de este artículo.

```
.cajadestacada {
  background-color: red; }
  .cajadestacada header {
    background-color: black;
    color: #fff; }
    .cajadestacada header a {
      color: #ff6; }
```

Anidación y buenas prácticas CSS / Sass

Debemos de ser cuidadosos con la anidación y no usarla alocadamente. Esto es algo importante en el desarrollo CSS en general, no solamente con los preprocesadores.

Usar selectores complejos, en los que se apliquen varios niveles de anidación puede llegar a ser contraproducente, ya que estamos siendo muy específicos con ciertos estilos, que llegarán a aplicarse en situaciones muy concretas.

En lugar de anidar a veces es preferible por ejemplo usar clases. Lo bueno de la clase es que la puedes reutilizar todas las veces que desees, a lo largo de cualquier parte de tu HTML, independientemente de la estructura de tu HTML.

La regla, no escrita, es que no debes de definir estilos CSS a selectores que aniden más de tres elementos.

Referencia al selector padre

A la hora de producir código anidado en Sass hay otra técnica que se usa bastante, que es la referencia al selector padre, usando el caracter "&".

Esto nos sirve para que, al definir código anidado, podamos continuar en el uso del selector sobre el que estamos trabajando, a fin de no repetirlo de nuevo. Seguro que con un ejemplo conseguimos entenderlo bien.

Este código de CSS, hace uso del selector padre para definir un estilo para el enlace cuando está en estado "hover".

```
a.dinamico {
  color: red;
  &:hover {
    background-color: #ff6;
  }
```

```
}
```

Simplemente, Sass, cuando vea el caracter "&" entenderá que tiene que sustituirlo por el selector padre de esta anidación. Por tanto, el anterior código compilará a este CSS estándar.

```
a.dinamico {
  color: red; }
  a.dinamico:hover {
    background-color: #ff6; }
```

Otro uso de este operador para referirse al selector padre es cuando estás usando una nomenclatura como "BEM". En estos casos haces nombres de clases como "form--black" o "form__submit", o incluso cosas más complejas como "form__submit--desactivado".

Para poder definir todos los estilos de la clase "form" de una vez y usando anidación, podríamos escribir algo como esto.

```
.form {
  margin: 10px;
  padding: 15px;
  &--black {
    color: #fff;
    background-color: #000;
  }
  &__submit {
    color: red;
    &--desactivado {
      color: #999;
    }
  }
}
```

Que daría como resultado el siguiente CSS estándar compilado.

```
.form {
  margin: 10px;
  padding: 15px; }
  .form--black {
    color: #fff;
    background-color: #000; }
  .form__submit {
    color: red; }
    .form__submit--desactivado {
      color: #999; }
```

Propiedades anidadas

Otra utilidad de la anidación en Sass es cuando tenemos propiedades que comparten una misma raíz, como "font-size", "font-family", etc. Podremos anidarlas de la siguiente forma.

```
h2 {
  color: #666;
  font: {
     family: verdana;
     weight: bold;
     size: 0.9em;
  }
}
```

Esto compilará al siguiente código CSS.

```
h2 {
  color: #666;
  font-family: verdana;
  font-weight: bold;
  font-size: 0.9em; }
```

Hemos aprendido muchas utilidades de anidación de CSS que estoy seguro que usarás mucho en tu día a día con Sass. Espero que hayas disfrutado pensando en todo el código que te vas a ahorrar!

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *25/04/2018*
Disponible online en *http://desarrolloweb.com/articulos/anidacion-selectores-sass.html*

Modularizar el código CSS con Sass

Cómo crear módulos de código Sass e importarlos, con lo que podemos segmentar, ordenar y orientar a componentes el código CSS, en diversos archivos, ayudando al mantenimiento.

Ha llegado el momento de hablar de los imports en el Manual de Sass. Es un punto de mucha importancia, ya que es una de las ventajas fundamentales de trabajar con preprocesadores. Sass te ofrece la posibilidad de dividir el código CSS en varios archivos, lo que es muy útil porque el código CSS suele ser muy extenso, sobre todo en sitios o aplicaciones grandes, lo que lo hace difícil de manejar.

Aunque técnicamente no es un problema disponer de un fichero muy grande, a la hora de editar el código se hace complicado y lento localizar el lugar exacto donde tienes cada estilo que quieras tocar. Por eso es muy interesante tener el código dividido en distintos ficheros, de modo que, cuando quieres modificar algo de tu sitio, sabes perfectamente dónde están esos estilos y puedes localizarlos rápidamente, ya que nuestros archivos tendrán muy poco código.

En resumen, dividir el código CSS de tu aplicación en múltiples ficheros, facilitará enormemente el mantenimiento. Por tanto, serás más productivo y la experiencia de desarrollo de tu CSS será más agradable en el día a día.

Nota: Alguien podría pensar que realmente CSS también nos ofrece esta posibilidad, gracias a los @import, y estará en lo cierto. Aunque esta misma operativa con Sass es mucho más optimizada. Esto es porque, con los @import comunes de CSS, el código reside en varios archivos distintos y el navegador tiene que ir solicitando esos archivos, uno a uno al servidor web, lo que es mucho más lento que si estuviera todo el código en el mismo fichero. En cambio, cuando desarrollas en Sass, durante el proceso de compilación del código, se hace la unión de todos los imports de tu CSS en un único fichero. Por tanto, los imports de Sass son mucho más interesantes que los imports del lado del CSS, porque una vez realizada la tarea del preprocesador, el código CSS resultante estará en un único archivo.

Cómo crear módulos scss a importar

Comenzamos por aclarar que, a continuación, vamos a llamar "módulos" a los archivos que creas para importar desde otros archivos scss.

En realidad es posible importar cualquier código scss desde otros ficheros de Sass, por lo que no necesitas hacer nada especial para crear estos archivos. Sin embargo es útil decirle a Sass qué archivos scss son módulos y qué archivos no lo son.

> **Nota:** cuando estás usando el "watcher" del compilador "oficial" de Sass, el que te enseñamos a utilizar en el artículo sobre Cómo usar Sass, https://desarrolloweb.com/articulos/que-es-sass-usar-sass.html y estás haciendo el seguimiento de una carpeta, Sass no tiene en principio cómo distingir los archivos que son módulos y los que no. Date cuenta que los módulos sólo los has creado para que sean importados desde otras carpetas, por lo que en principio no deberían compilarse ellos solos.

En resumen, para que el compilador de Sass no compile un módulo por separado, tiene que saber qué archivo es un módulo. Para ello simplemente vamos a nombrar el módulo comenzando por un guión bajo "_".

Imagina que quieres crear un módulo con una serie de definiciones de variables. Entonces ese módulo lo podrías llamar "_variables.scss". Entonces lo podrás colocar en tu carpeta de código Sass y, como se ha identificado como un módulo, entonces no se compilará por separado. Y es lógico que lo desees así, pues las definiciones de variables por si solas no te sirven de nada en un archivo CSS.

Así pues, todos los archivos que comiencen por un "_" no generarán nuevos archivos .css, como se puede ver en la siguiente imagen. Solamente estilo.scss generará una salida en estilo.css.

Cómo importar un módulo Sass

Ahora veamos el siguiente paso, dado un módulo, queremos que se cargue en un archivo que sí se compile. Es decir, en el archivo scss principal, iremos realizando todas las importaciones de los módulos, para agrupar todos los archivos que teníamos por separado.

Esto se consigue con la sentencia @import, a continuación del nombre del módulo a importar, sin el guión bajo y sin la extensión. Por ejemplo, para importar el archivo "_variables.scss", realizaremos el siguiente código en el archivo principal, desde el que queremos importar el módulo.

```
@import 'variables';
```

Obviamente, este import debe figurar en el código scss antes que comiences a usar las variables que se están declarando. Cabe aclarar también que no solamente se usan los imports para declaraciones de variables, sino en general para poder organizar cada pieza de tu código CSS, como explicaremos en el siguiente punto.

Ejemplo de imports con Sass

Aunque es muy sencillo veamos un par de listados de código de archivos Sass para ver cómo es el trabajo de importar CSS de ficheros aparte.

Veamos primero el archivo con un código que vamos a importar. Este sería el código de "_fonts.scss", que define algunas fuentes, en variables que se han definido para usar las veces que haga falta a lo largo del código restante.

```
$f-normal: 16pt;
$f-pequena: 12pt;
$f-grande: 20pt;

$tipo-general: 'Trebuchet MS', 'Lucida Sans Unicode', Arial, sans-serif;
$tipo-alternativa: 'Times New Roman', serif;
```

Como puedes comprobar, son reglas de estilos Sass, para la declaración de variables. No hay nada que especifique que este archivo es un módulo. Recuerda que Sass lo podrá deducir por el nombre del archivo con su "_" al inicio.

Por otra parte puedes ver el código donde importamos estos valores. Este sería el listado de "estilos.css":

```
@import "fonts";

body {
   font-size: $f-normal;
   font-family: $tipo-general;
}
```

```
h1 {
   font-size: $f-grande;
   font-family: $tipo-alternativa;
}
```

Cómo organizar tus archivos de CSS

Aunque esto no tiene mucho que ver con Sass, te vamos a dar una serie de apuntes que puedes, o no, tener en cuenta como ideas para organizar los módulos de tu proyecto Sass. Primero cabe aclarar que el desarrollador es el encargado de realizar la organización que juzgue oportuna para su código. En la mayoría de los casos un poco de sentido común ayudará bastante, aunque vamos a ofrecer aquí dos consejos fundamentales que puedes aplicar para conseguir una buena organización de tu código.

> **Nota:** existen diversas aproximaciones a arquitecturas CSS publicadas por desarrolladores de renombre, que ofrecen guías "estandarizadas" para organizar el código CSS. Una bastante conocida es ITCSS, que puede servirte de ayuda para llegar a tu propio modelo de organización del código. Si te interesa profundizar te recomendamos la lectura del artículo ITCSS. Si deseas más consejos de CSS te recomendamos los artículos Arquitecturas de CSS, los problemas y Las soluciones.

Definiciones globales

Es útil tener en archivos separados, módulos, las definiciones que usas desde otros archivos de tu CSS, como por ejemplo los colores, tipos de fuentes, tipos de márgenes, etc. Si son pocas tus definiciones, puede que con un único módulo lo soluciones, pero siempre será aconsejable tener definiciones en archivos por separado de cosas como:

- Colores
- Fuentes
- Espaciados

Reset o normalize

Otro módulo que sería interesante tener al principio, cerca de tus definiciones globales, es el código de tu reset o normalize. No lo hemos dicho, pues resulta obvio, pero los imports se van colocando en el código CSS resultante en el orden en el que han sido importados. Por tanto, su orden aplica también a la regla de la cascada del CSS. Por tanto, el import a tu archivo reset debería ir al principio de todo.

Componentes

Otro gran consejo a la hora de definir tus módulos es aplicar la "componetización". Los estilos de cada pequeño componente de tu interfaz web deben colocarse en un archivo aparte.

Por ejemplo, podemos tener un módulo para defínción de cosas como:

- Botones
- Capas modales
- Paneles
- Tooltips
- Breadcrumbs (migas de pan)
- Cabecera
- Pie
- ...

Así, cada vez que quieras modificar cualquier pequeña parte de la interfaz de tu sitio, sabrás en qué archivo CSS están esos estilos. Encontrarás rápidamente las reglas CSS a modificar y ayudarás a otras personas que más adelante tengan que lidiar con tu código CSS.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *11/06/2018*
Disponible online en *http://desarrolloweb.com/articulos/modularizar-codigo-css-sass.html*

Mixins en Sass

Qué es un mixin y cómo usar mixins en Sass, el preprocesador CSS. Con ellos podemos agilizar la escritura de códigos repetitivos y mejorar el mantenimiento de los estilos del sitio.

En Sass, así como en otros preprocesadores, existen herramientas para reutilizar código, y para hacer cálculos que devuelvan un código procesado. Para esta parcela de trabajo, en Sass tenemos los mixins, que serían algo así como lo que conocemos como funciones en un lenguaje de programación. Hoy en el Manual de Sass aprenderemos acerca de los mixin.

Podemos escribir un mixin e invocarlo tantas veces como sea necesario, produciendo una salida CSS. Un mixin además tiene la habilidad de recibir parámetros, como las funciones, de modo que se produzca la salida a partir del valor de los parámetros recibidos en su invocación.

La utilidad de los mixin es la de permitir al desarrollador escribir menos código, ya que por medio de su procesamiento y el envío de parámetros, los mixins pueden producir salida variada solamente con invocarlos. Al pasarle parámetros distintos podremos obtener varias salidas personalizadas a nuestras necesidades, con un mismo mixin de base. Pero esa no es la principal ventaja. Realmente lo más interesante es producir un código más claro, más expresivo y sobre todo con mayor facilidad de mantenimiento.

Aunque no es de las funcionalidades más elementales de Sass, el uso de mixins no es difícil. Inclusive, si no tienes conocimientos de programación y no sabes lo que son las funciones, los entenderás fácilmente. A medida que veas ejemplos de mixins Sass en este artículo irás entendiendo mejor su utilidad y la potencia que tienen.

Definición de un mixin

Comencemos aprendiendo a definir un mixin. Es algo muy sencillo, pues simplemente tenemos que comenzar a declararlos con la cadena "@mixin", seguido por el nombre del mixin que estamos generando y unas llaves para englobar su código.

Aquí tenemos un primer ejemplo de mixin:

```
@mixin color-invertido {
  background-color: #111;
  color: #eee;
}
```

Este mixin simplemente define dos propiedades de CSS, que serán incluidas allí donde se invoque.

Invocación de un mixin

Ahora veamos cómo llamar ese mixin Sass para producir una salida. Es tan sencillo como escribir "@include" seguido del nombre del mixin que queremos invocar.

Por ejemplo, dado el mixin anterior, "color-invertido", podríamos tener diversos elementos de la página a los que querríamos aplicar esas propiedades. En vez de copiar y pegar todo el tiempo los mismos valores, invocamos el mixin.

```
h1.invertido {
  font-size: 1.3em;
  padding: 15px;
  @include color-invertido;
}

div.invertido {
  padding: 5px;
  @include color-invertido;
}

blockquote {
  margin: 2em 3em;
  padding: 20px;
  text-align: center;
  @include color-invertido;
}
```

> **Nota:** Como podrás imaginar, todos los elementos definidos tendrán incluidos los estilos del mixin. Esto facilita el mantenimiento porque, si más adelante queremos modificar el ese "color invertido", solo habría que cambiarlo en el mixin, una vez, y se traspasará a todos los lugares donde lo estamos usando. Si estás interesado en saber más sobre el mantenimiento de CSS te recomendamos aprender también acerca de algún patrón de arquitectura CSS como ITCSS.

Esto produce como salida:

```
h1.invertido {
  font-size: 1.3em;
  padding: 15px;
  background-color: #111;
  color: #eee; }

div.invertido {
  padding: 5px;
  background-color: #111;
  color: #eee; }

blockquote {
  margin: 2em 3em;
  padding: 20px;
  text-align: center;
  background-color: #111;
  color: #eee; }
```

Paso de parámetros en un mixin Sass

La utilidad de los mixin no sería tal si no tuvieran la capacidad de recibir parámetros. Esto es muy interesante, porque así nuestros mixins pueden producir salida diferente, simplemente mandando valores de parámetros distintos.

Un claro ejemplo de esta utilidad es la inserción automática de los vendor prefixes de CSS, es decir, los prefijos que necesitan algunos navegadores para entender ciertas propiedades CSS nuevas, que todavía no están en el estándar definitivo.

Por ejemplo, una de las propiedades CSS que necesita todavía hoy ciertos vendor prefixes es "transform". Podríamos realizar un mixin que recibe como parámetro la función de transformación. Luego como salida el mixin incluye esa función de transformación, con todos los prefijos que queramos que sean incorporados.

```
@mixin transformar($propiedad) {
  -webkit-transform: $propiedad;
  -ms-transform: $propiedad;
  transform: $propiedad;
}
```

Ahora podemos aplicar ese mixin en cualquier elemento que queramos transformar. Seguiremos usando el método de antes con @include, solo que ahora después del nombre del mixin a invocar, debemos enviar el valor del parámetro con la función de transformación a aplicar.

```
.escalada {
  @include transformar(scale(2, 3))
}
```

```
h1 {
  color: blue;
  @include transformar(rotate(22deg))
}
```

Como puedes imaginar, simplemente cambiando el parámetro, podemos conseguir efectos distintos de transformación, con sus apropiados prefijos. Esto producirá como salida el siguiente CSS procesado.

```
.escalada {
  -webkit-transform: scale(2, 3);
  -ms-transform: scale(2, 3);
  transform: scale(2, 3); }

h1 {
  color: blue;
  -webkit-transform: rotate(22deg);
  -ms-transform: rotate(22deg);
  transform: rotate(22deg); }
```

Los mixins pueden producir todo tipo de salida

Hasta ahora hemos visto mixins que producían una salida con una serie de propiedades, que se incluirán en el selector que nosotros queramos, al aplicar el mixin. Pero realmente un mixin puede producir salidas más complejas, con selectores diversos y aplicando estilos en ellos.

Por ejemplo, este mixin produce una serie de selectores de etiqueta, con sus propiedades:

```
@mixin encabezados {
  h1 {
    font-size: 1.5em;
  }
  h2 {
    font-size: 1.2em;
  }
  h3 {
    font-size: 1em;
  }
}
```

Al invocarse, (@include encabezados) se colocarán esos estilos allí donde se requiera. Quizás este mixin no sea tan útil pero si lo parametrizamos un poco ya podrá comenzar a tener sentido, como veremos en el siguiente ejemplo.

Piensa en el mixin anterior "encabezados" y piensa que quieres aprovechar ese código para preparar estilos con encabezados de diversos tamaños, dependiendo de un parámetro. Para conseguir eso podríamos recibir un parámetro que sea el tamaño del encabezado superior (h1) y luego ir restando para producir los tamaños de encabezados menores.

Conseguir esto es muy sencillo con un parámetro en el mixin, con el que indicamos el tamaño base del H1, sobre el que luego iremos restando.

```
@mixin encabezados($tamano) {
  h1 {
    font-size: $tamano;
  }
  h2 {
    font-size: $tamano - 0.2;
  }
  h3 {
    font-size: $tamano - 0.5;
  }
}
```

Ahora al incluir el mixin tendremos que pasarle el tamaño base del H1, con los paréntesis.

```
@include encabezados(2em);
```

Por ejemplo, podríamos obtener utilidad de este mixin a la hora de definir las mediaqueries.

```
@include encabezados(1.5em);
@media(min-width: 800px) {
  @include encabezados(2em);
}
@media(min-width: 1200px) {
  @include encabezados(2.5em);
}
```

Esto producirá la siguiente salida:

```
h1 {
  font-size: 1.5em; }

h2 {
  font-size: 1.3em; }

h3 {
  font-size: 1em; }

@media (min-width: 800px) {
  h1 {
    font-size: 2em; }

  h2 {
    font-size: 1.8em; }

  h3 {
```

```
    font-size: 1.5em; } }
@media (min-width: 1200px) {
  h1 {
    font-size: 2.5em; }

  h2 {
    font-size: 2.3em; }

  h3 {
    font-size: 2em; } }
```

Conclusión

La verdad es que yo nunca pensé que CSS pudiera necesitar de herramientas similares a las de los lenguajes de programación, con cosas como funciones o mixins, pero lo cierto es que en el día a día pueden ser muy útiles. No solo para poder realizar un mantenimiento más sencillo en un proyecto, sino también para reutilizar código de proyectos a proyectos. Seguro al usarlos irás construyendo utilidades que te vendrán muy bien para los siguientes trabajos.

A medida que explores más aún te darás cuenta que hay otra serie de utilidades en el lenguaje para hacer mixins Sass más complejos, que hagan cosas más espectaculares. De momento lo que hemos visto está bien para comenzar y la mayor parte de los mixins la verdad es que hacen cosas bastante similares a las aprendidas.

De todos modos, en siguientes artículos continuaremos dando ejemplos con mixins más complejos que te vendrá bien conocer y practicar para seguir sacando el mejor partido de Sass.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *18/07/2018*
Disponible online en *http://desarrolloweb.com/articulos/mixin-sass.html*

Herencia en Sass con @extend

Qué es la directiva @extend de Sass, cómo nos permite organizar el código y crear CSS más limpio gracias a la herencia y las clases placeholder.

El objetivo de Sass es doble. Por una parte ser más productivos y por otra parte tener la capacidad de mantener mejor nuestro código CSS. En este artículo vamos a conocer una funcionalidad que nos ayuda en el segundo caso, ya que nos permite de una manera más sencilla ser más concisos en el código, reutilizar partes de nuestras reglas CSS y evitar tener que escribir el mismo código una y otra vez.

Conoceremos la herencia, un mecanismo por el cual un selector puede recibir estilos CSS que nos llegan de declaraciones realizadas con anterioridad. Conseguir este objetivo es sencillo gracias a la directiva @extend y las denominadas "placeholder class", que son una construcción de Sass que no tiene representación en el CSS hasta que no la usemos. Vamos a verlo todo con calma y hacer algunos ejemplos.

Clases placeholder

Este tipo de clases CSS son específicas de Sass, no existen en el lenguaje CSS de momento, así que probablemente será algo nuevo para ti. Básicamente son declaraciones CSS que podemos realizar agrupando diversas reglas de estilo. Hasta aquí son iguales que una class CSS común, la diferencia es que no tienen una representación directa en el código CSS compilado. Sólo se escribirán en el CSS resultante cuando sean usadas.

Comencemos viendo un ejemplo de una de estas clases placeholder. La sintaxis para crearlas consiste en anteponerle un símbolo "%".

```
%heading {
  background-color: blanchedalmond;
  color: brown;
  font-family: 'Times New Roman', Times, serif;
}
```

Esta declaración indica un estilo de base para nuestros encabezados y nos sirve para poder agregarla en donde la necesitemos. Sin embargo, si la dejamos tal cual en nuestro código Sass, sin usarla, no serviría para nada, debido que al compilarse simplemente se eliminaría sin producir salida alguna.

> **Nota:** Si sabes algo de programación, podrías considerarla como una función que nunca se ha invocado, simplemente no entró en marcha en el flujo de ejecución de un programa.

Así que ahora, vamos a aprender a usar estas clases placeholder, de modo que tenga algún sentido apoyarse en ellas.

Directiva @extend de Sass

La directiva @extend nos sirve, como su nombre indica, para extender el código de cierta declaración de CSS con nuevos estilos. Las reglas de estilo con las que podremos extender las declaraciones serán tomadas directamente de las clases placeholder.

Por ejemplo, podemos tener varios encabezados en nuestra página que se extienden mediante el placeholder class creado en el punto anterior. Para ello usamos la sintaxis @extend, seguido del nombre de la clase placeholder que queremos usar.

```
h1 {
  @extend %heading;
  font-size: 2em;
}

h2 {
  @extend %heading;
  font-size: 1.5em;
}
```

Como puedes comprobar, ambos encabezados usarán los mismos estilos base, con la única diferencia que sus tamaños de fuente serán distintos.

Nuestro código Sass ahora es más modular, pero es otra utilidad distinta de la que conocimos con los import de Sass. Alguien podría decir que es similar en funcionalidad a lo que nos ofrecen los mixins, pero realmente los @extend son más sencillos y en la mayoría de las ocasiones es suficiente con ellos.

Lo interesante de este mecanismo proporcionado por @extend es que Sass te produce un código optimizado, ya que las reglas de estilo no se repiten realmente dos veces en cada selector. El código producido mantendrá las buenas prácticas. Puedes verlo a continuación.

```
h1, h2 {
```

```
  background-color: blanchedalmond;
  color: brown;
  font-family: 'Times New Roman', Times, serif; }

h1 {
  font-size: 2em; }

h2 {
  font-size: 1.5em; }
```

Como puedes apreciar, a pesar que hemos usado el @extend en cada uno de los selectores H1 y H2, no se produce una repetición del código, sino que la declaración se realiza de manera que sólo aparecerán esas reglas escritas una única vez.

Una vez declarado un placeholder class lo podemos usar en cualquier parte del código mediante un @extend, por supuesto también en otros archivos que estemos importando con @import.

Caso práctico de optimización con @extend

Como puedes imaginar, usar @extend te facilita el mantenimiento del código, ya que te permite no escribir varias veces las mismas reglas. Si más adelante se quieren cambiar los estilos es suficiente con editar la placeholder class una vez.

Puede haber infinitos casos de uso en los que podamos sacarle partido a esta declaración. Pero por poner un ejemplo concreto que ocurre en la vida real, veamos el siguiente caso.

Imagina que tienes diversas declaraciones de cajas. Podemos tener cajas normales, cajas con borde, cajas con un color de fondo dado, cajas con un espaciado mayor, etc. Y luego podemos tener muchas combinaciones, por ejemplo cajas con borde, y color de fondo. O una caja con un color de fondo y un margen mayor...

Generalmente, cuando escribes los estilos, incluso usando la característica de anidación de selectores de Sass para escribir menos código, haces algo como esto:

```
.caja {
  padding: 10px;
  font-size: 1.2em;

  &-borde {
    border: 1px solid #ddd;
  }

  &-fondo {
    background-color: #f0f0f0;
  }

  &-espaciadoextra {
    padding: 20px;
  }
```

```
}
```

Realmente no hay problema alguno en hacer este código y muchos desarrolladores, incluso frameworks, lo vienen haciendo así. Pero el HTML que debes usar cuando quieres una caja específica es un poco feo.

```
<div class="caja caja-borde">
    Esto es una caja con borde
</div>
```

El problema se magnifica cuando quieres hacer una caja que tiene borde, fondo y espaciado extra, todo a la vez. El código HTML será horrible.

```
<div class="caja caja-borde caja-fondo caja-espaciadoextra">
    Esto es una práctica habitual, pero el class me ha quedado un chorizo, poco mantenible y sucio por ser demasiado largo.
</div>
```

Ahora imagina que usas constantemente ese tipo de caja en tu código (caja + caja con borde + caja con fondo + caja con espaciado extra), con lo cual tienes que repetir ese chorizo de clases en un montón de etiquetas HTML, quedando un marcado redundante, pesado, de complicada lectura y peor mantenimiento.

Gracias a @extend podemos conseguir una situación un poco mejor. Tú podrías seguir teniendo todos esos tipos de cajas, pero además la declaración de diferentes cajas que combinan distintos tipos a la vez, como el caso de antes.

```
%caja {
  padding: 10px;
  font-size: 1.2em;
}

%caja-borde {
  border: 1px solid #ddd;
}

%caja-fondo {
  background-color: #f0f0f0;
}

%caja-espaciadoextra {
  padding: 20px;
}

.caja {
  @extend %caja;

  &-borde {
    @extend %caja;
```

```
    @extend %caja-borde;
  }

  &-fondo {
    @extend %caja;
    @extend %caja-fondo;
  }

  &-espaciadoextra {
    @extend %caja;
    @extend %caja-espaciadoextra;
  }

  &-combinada {
    @extend %caja;
    @extend %caja-borde;
    @extend %caja-fondo;
    @extend %caja-espaciadoextra;
  }
}
```

Bien cierto es que el código Sass nos ha quedao mucho más largo. En este caso no hemos sido más productivos, puesto que nos ha dado más trabajo de escribirlo, pero sin embargo el código CSS resultante será muy optimizado, puesto que las declaraciones de estilos no se van a repetir. Pero lo realmente interesante será en nuestro HTML, ya que estaremos ahorrando expresar todo el chorizo de clases cada vez que queramos usar cajas determinadas.

Por ejemplo para una caja con borde escribimos esto:

```
<div class="caja-borde"> ... </div>
```

Para una caja con varios estilos combinados también usaremos solamente una clase de CSS.

```
<div class="caja-combinada"> ... </div>
```

Por si alguien tiene curiosidad, el código CSS generado será como este:

```
.caja, .caja-borde, .caja-fondo, .caja-espaciadoextra, .caja-combinada {
  padding: 10px;
  font-size: 1.2em; }

.caja-borde, .caja-combinada {
  border: 1px solid #ddd; }

.caja-fondo, .caja-combinada {
  background-color: #f0f0f0; }

.caja-espaciadoextra, .caja-combinada {
  padding: 20px; }
```

Como puedes apreciar, las reglas de estilos CSS no se repiten, por lo que es lo más resumido posible y por tanto muy optimizado. Pero donde realmente obtenemos una ventaja representativa es en nuestro HTML, ya que no tenemos que usar un montón de clases CSS en la misma etiqueta.

Espero que este nuevo truco te haya parecido interesante. No te olvides consultar el Manual de Sass para obtener más detalles útiles para el trabajo con este poderoso preprocesador CSS.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *27/09/2018*
Disponible online en *http://desarrolloweb.com/articulos/herencia-sass-extend.html*