Regra de Estilo @forward

O que faz a Regra de Estilo @forward?

A regra de estilo @forward é usada quando queremos carregar diversas folhas de estilo dentro de uma única folha de estilo e disponibilizar para o usuário todos os mixins, funções e variáveis de todas as folhas de estilo carregadas dentro da folha de estilo com @forward. Mas o @forward vai além disso, ele não apenas disponibiliza, mas ele é capaz de filtrar cada elemento dentre tantas folhas de estilo e disponibilizar somente o elemento desejado. Isso é possível por que com o @forward nós organizamos todas as folhas de estilo carregadas no arquivo, permitindo que o arquivo scss que vai gerar o arquivo final carregue um único arquivo de entrada por através da regra @use. (Por convenção damos ao arquivo scss que servirá de entrada para as folhas de estilo o nome de “bootstrap”).

As vantagens de usar um arquivo bootstrap são:

- **Economia de tempo:** pois teremos várias formatações desejadas já embutidas dentro do arquivo bootstrap;
- **Um módulo carregado por vez:** apesar de ter “n” módulos sendo carregados dentro dele, o arquivo bootstrap vai carregar somente os módulos necessários para o momento, evitando uma sobrecarga de banda com o carregamento de arquivos ociosos sem necessidade. O que será determinante no uso de um módulo que estiver dentro do bootstrap ou não, será a regra @use que chamará pelo módulo desejado no arquivo “.scss” final;

Como utilizar a regra @forward?

Podemos utilizá-la das seguintes formas:

- ## Modo Simples:

  A forma simples de utilizar essa regra é:

  ```
  @forward “nomeDoArquivo”
  ```

  Dessa forma, a regra vai carregar o nome do arquivo fornecido, ou a URL dele se ele não estiver na mesma pasta ou local.

  Atenção! Quando utilizamos um módulo que contém underline na frente, por exemplo: “_arquivo.scss” , não precisamos colocar o underline, em vez disso vamos colocar: “arquivo.scss”, sempre dentro das aspas.

  Outro detalhe importante sobre a regra @forward é que todos os estilos carregados por através de @forward vão fazer parte do arquivo que estamos trabalhando.

  Veja como utilizá-la logo abaixo:

  ### Primeiro temos que ter um ou mais módulos que serão importados:

  Arquivo: _list.scss

  ```
  @mixin reset {
        margin: 0;
        padding: 0;
        list-style: none;
  }
  ```

  Criamos um mixin dentro de um módulo para resetar as margens, padding e estilo de uma lista;

  ### Carregando o módulo num bootstrap com @forward:

  Arquivo bootstrap.scss:

  ```
  @forward “src/list”;
  ```

  Perceba que no arquivo bootstrap foi carregado só o módulo “_list.scss”, o arquivo bootstrap serve só para armazenar os módulos que serão usados. No exemplo acima temos apenas um módulo, mas geralmente utilizamos “n” módulos dentro de um arquivo bootstrap;

Carregando o arquivo bootstrap no arquivo Sass com @use:

Arquivo style.scss:

```
@use “bootstrap”;

li {
      @include bootstrap.reset;
}
```

Perceba que no arquivo Sass carregamos o arquivo “bootstrap.scss”, e depois utilizamos o mixin “reset” que havia dentro deste arquivo, note que para isso não foi necessário chamar o módulo original do arquivo, chamamos apenas o arquivo bootstrap quando usamos o @include. Isso mostra que o módulo passou a fazer parte do arquivo bootstrap assim que ele foi chamado pela regra @use;

Resultado no arquivo final:

Arquivo Final:

```
.li {
      margin: 0;
      padding: 0;
      list-style: none;
}
```

Perceba que no arquivo final, temos somente as formatações necessárias para o CSS final. Incrível não é?

- ## Usando Prefixos:

Já vimos que quando um módulo se torna parte de um bootstrap, todas suas funções, mixins, variáveis e elementos em geral passam a fazer parte do arquivo bootstrap, mas aí mora um grande perigo. Como temos o costume de usar nomes curtos para definir os nossos elementos, pode acontecer de darmos um nome muito comum aos nossos elementos, talvez idêntico ao utilizado em outros módulos que também estejam no bootstrap. Isso é um problema, pois, quando formos chamar por um elemento, o Sass vai chamar pelo último elemento que tiver o nome que chamamos, e talvez esse não seja o resultado que estamos esperando.

Pensando nisso, o Sass trabalha com um recurso chamado “prefix” ou “prefixo”, que consiste em darmos um prefixo a um módulo importado dentro do bootstrap por através da regra @forward, podemos fazer isso assim que importamos um módulo dentro do nosso bootstrap.

Veja como:

Carregando arquivo bootstrap com prefixo:

Arquivo bootstrap.scss:

```
@forward “src/list” as list-*;  //Aqui usamos o prefix
```

Perceba que usamos o prefixo “list-” e na sequência usamos o * para referenciar a todos os elementos que estiverem dentro daquele módulo, isso vai fazer com que identifiquemos o nome de um elemento pelo módulo correto dentro do arquivo bootstrap.

Carregando o arquivo bootstrap no arquivo Sass com @use:

Arquivo style.scss:

```
@use “bootstrap”;

li {
      @include bootstrap.list-reset;
}
```

Perceba que usamos o prefixo antes do nome do mixin, isso faz com que chamemos o mixin correto do módulo correto.

Resultado no arquivo final:

Arquivo Final:

```
.li {
	margin: 0;
	padding: 0;
	list-style: none;
}
```

- **Controlando Visibilidade dos Elementos:**

Com a regra @forward também é possível controlar a visibilidade de um elemento que está dentro de um módulo. Por exemplo, digamos que não desejamos que o usuário final chame uma determinada variável, função ou mixin, podemos por assim dizer escondê-lo.

Veja como:

Módulo original:

Arquivo _list.scss:

```
@mixin reset {
	margin: 0;
	padding: 0;
	list-style: none;
}

@mixin horizontal {
	li {
		@include reset;
		display: inline-block;
		margin: {
			Left: -2px;
		}
	}
}
```

Perceba que temos 2 mixins nesse módulo, vamos querer esconder o mixin “reset”;

Carregando arquivo bootstrap e escondendo um mixin:

Arquivo bootstrap.scss:

```
@forward “src/list” as list-* hide reset;  //usamos o comando hide para esconder o mixin
```

O hide esconde os elementos de um módulo que não desejamos que o usuário tenha acesso ao puxar os elementos pelo scss gerador do arquivo final. Podemos esconder quantos elementos quisermos, para isso basta separarmos os arquivos pela vírgula.

Carregando o arquivo bootstrap no arquivo Sass com @use:

Arquivo style.scss:

```
@use “bootstrap”;

li {
      @include bootstrap.list-horizontal;
}
```

Perceba que por escondermos o mixin “reset” não é mais possível chamá-lo diretamente.

Resultado no arquivo final:

Arquivo Final:

```
.li {
      margin: 0;
      padding: 0;
      list-style: none;
      margin-left: -2px;
}
```

OBSERVAÇÕES IMPORTANTES!!!

Nunca se esqueça:

- Prefixos são atribuídos somente no arquivo bootstrap, no scss origina podemos dar somente um apelido ao arquivo bootstrap;

- Quando usamos uma variável geralmente ela vai com o $ na frente, se tivermos dado um prefixo a um módulo devemos colocar o prefixo na frente do $, assim:

```
$prefixo-variavel
```

  Se fizermos:

```
prefixo$variavel
                    //Estamos usando um método errado!!!
```

- Só por que colocamos um prefixo sobre um elemento do módulo isso não quer dizer que não temos que declarar o nome do arquivo bootstrap, a forma correta a se fazer é assim:

```
bootstrap.$prefixo-variavel
```

  A forma errada seria:

```
$prefixo-variavel   //ou
$bootstrap.prefixo-variavel
```

- Quanto usamos um arquivo bootstrap é aconselhável usar o apelido * caso já tenhamos dado prefixos aos módulos com a ajuda do @forward, afinal não haverá conflitos já que todos os arquivos já estão identificados com o prefixo. Isso facilita por que não vamos precisar de declarar o nome do arquivo bootstrap antes de cada chamada de elemento;