1 Препроцессинг

С развитием технологий и сложности макетов возникла потребность в автоматизации написания CSS. Чтобы избавиться от рутины и превратить написание CSS в процесс, похожий на программирование были придуманы препроцессоры. Они позволяют создавать переиспользуемый CSS код, использовать переменные и расширить возможности написания CSS. Уменьшает повторение CSS и, следовательно, экономит время.

На сегодняшний день ярким представителем препроцессоров является SCSS. Он в любом своём проявлении является расширением CSS, а значит всё, что работает в CSS, работает и в SCSS.

2 Переменные

Наш CSS код может иметь внушительный размер. И, например, некоторые части могут повторяться. Самый часто повторяющимся элементом является цвет. Если нужно поменять цвет, то довольно неприятно его вручную менять в нескольких местах, поэтому для таких случаев удобнее использовать переменные. SCSS позволяет работать с переменными. В CSS они обозначаются двойным тире, а в препроцессорах знаком доллара.

Листинг 1 – Переменные SCSS

```
$blue: #3bbfce;
$dark: #3f3d46;
$text: "cool text";
$text: "default text" !default;


.container {
  border-color: $blue;
  background-color: $dark;
}

.wrapper {
  border-color: $blue;
}
```

```
.content {
      content: $text;
}
```

Значение по умолчанию переменным можно добавить с помощью метки !default в конце значения. В таком случае, если переменной уже было присвоено значение, оно не изменится. А если же переменная пуста, ей будет присвоено новое указанное значение.

И не стоит забывать, что переменные могут быть присвоены любому свойству. Конечно, в CSS есть переменные, но переменные SCSS куда более мощный инструмент. Например, можно использовать переменные в циклах и генерировать значения свойств динамически.

3 Вложенные правила

SCSS позволяет правилам CSS быть вложенными друг в друга. Различие между CSS и SCSS рассмотрено в листингах 2-3.

Листинг 2 – Вложенность стандартного CSS

```
#a {
      color: red;
}

#a #b {
      color: blue;
}

#a #b #c {
      color: green;
}
```

Листинг 3 – Вложенность SCSS

```
#a {
      color: red;
      #b {
            color blue;
            #c {
                  color: green;
            }
```

```
    }
}
```

Код листинга 3 в итоге скомпилируется в обычный CSS. Это просто синтаксис, который выглядит чище и с меньшими повторениями. Помогает работать со сложными макетами. Вложенные свойства точно соответствует структуре макета.

В листинге 4 показано более удобное применение комбинаторов при структурировании своих макетов.

Листинг 4 – Удобное применение комбинаторов

```
.button {
  > span { }
  + span { }
  ~ span { }
}
```

4 Амперсанд

В SCSS существует директива &, которая используется во вложенных селекторах для ссылки на внешний селектор.

Позволяет повторно использовать внешний селектор, например, добавляя псевдокласс или добавляя селектор перед родителем. Чрезвычайно полезная функция. Это также может здорово сэкономить время, если вы знаете, как им пользоваться.

Листинг 5 – Удобное применение псевдоклассов

```
.button {
  &:visited { }
  &:hover { }
  &:active { }
}
```

Одной из самых классных особенностей амперсанда является работа с классами-модификаторами, которая представлена на листинге 6.

Листинг 6 – Работа с классами-модификаторами

```
.btn {
  &-primary {}
  &-secondary {}
```

```
}
```

Это может быть весьма полезно, если используется методология именования (например, БЭМ), в которой используются комбинированные классы тире и подчеркивания, а не комбинированные селекторы.

К сожалению, амперсанд имеет свои минусы. & не позволяет выборочно перемещаться по вложенному дереву селекторов в определенное место и использовать только небольшую часть скомпилированного родительского селектора, который вы хотите использовать.

5 Примеси

Примеси - элементы, реализующие какое-либо чётко выделенное поведение. Позволяют вам определять стили, которые можно повторно использовать. По сути, это повторяющиеся участки кода, которые можно включать в различные селекторы. Также позволяют легко избежать использования несемантических имен классов, таких как .float-left,

Примеси объявляются директивой @mixin. После неё должно стоять имя примеси и, опционально, его параметры, а также блок, содержащий тело примеси. Например, можно определить классический “миксин” на листинге 7, который подключается с помощью @include.

Листинг 7 – Классическая примесь

```
@mixin flexible() {
    display: flex;
    justify-content: center;
    align-items: center;
  }

.elements {
    @include flexible();
    border: 1px solid black;
  }
```

Теперь каждый раз после применения класса .elements к HTML-элементу, последний будет преобразован во Flexbox.

Некоторые вещи в CSS весьма утомительно писать, особенно в CSS, где плюс ко всему зачастую требуется использовать большое количество вендорных префиксов. С помощью примесей приятно определять CSS свойства, индивидуальные для каждого браузера, в одном классе.

Листинг 8 – Кросс-браузерность

```
@mixin border-radius($radius) {         // Префиксы для:
    -webkit-border-radius: $radius;     // Chrome и Safari
    -moz-border-radius: $radius;        // Firefox
    -ms-border-radius: $radius;         // Internet Explorer
    -o-border-radius: $radius;          // Opera
    border-radius: $radius;             // Стандартный CSS
 }

.box {
      @include border-radius(10px);
}
```

6 Фрагментирование и импорт

Директива @import является правилом CSS, которое какое-то время было очень популярным, прежде чем попало в немилость разработчиков из-за проблем с производительностью.

С помощью @import можно подключать CSS внутри других стилей. При этом подключаемые файлы могут либо располагаться на одном сервере или на другом.

Листинг 9 – CSS @import

```
@import "style.css";
@import "css/style.css";
@import url("http://domain.com/css/styles.css");
```

SCSS импортирование нужно потому, что если вы используете для импорта CSS, то каждый файл создает отдельный HTTP запрос. Это означает, что страница будет грузиться медленнее. Импортирование с помощью SCSS создает только один CSS файл, который грузится быстрее.

Листинг 10 – SCSS @import

```
@import "header/header.scss";

@import "main/main.scss";
@import "main/aside.scss";

@import "footer/footer.scss";
```

SCSS компилирует все файлы .scss внутри каталога, который он просматривает. Но если нужно импортировать файл, не обязательно компилировать его. Если добавить в начале имени файла подчеркивание _header.scss, то SCSS не будет его компилировать.

Можно пропустить подчеркивание при ссылке на файл внутри правила @import. Так же можно пропустить расширение. SCSS import понимает, что нужно импортировать файл _header.scss из текущего каталога, если он существует.

Листинг 10 – Пример импортирования

```
@import "header";
```

Возможность указать, какие файлы компилировать, а какие просто включить в другие файлы, позволяет создать структуру файлов и каталогов, которую проще поддерживать.

7 Арифметические операции

Как и в стандартном CSS, вам доступны операции сложения, вычитания, умножения и деления, но в отличии от классического CSS синтаксиса, вам необязательно использовать функцию calc().

Листинг 11 – Сложение и вычитание SCSS

```
$title-size: 10px;
$message-width: 200px;

.title {
    font-size: $title-size + 4px;
}

.message {
```

```
    width: $message-width - 50px;
}
```

Листинг 12 – Результаты сложения и вычитания SCSS

```
.title {
  font-size: 14px;
}

.message {
  width: 150px;
}
```

Есть несколько неочевидных особенностей, которые следует учитывать, чтобы избегать ошибок. Это означает, что, вы не можете работать с числами у которых несовместимы типы данных. Например, складывание px и em или двух чисел одного типа, перемножение которых даст квадратные единицы.

Листинг 13 – Умножение и деление SCSS

```
$title-size: 10px;
$message-width: 200px;
$param: 2;

.title {
    font-size: $title-size * $param
}

.message {
    width: $message-width / ($param * 2)
}
```

Листинг 14 – Результаты умножения и деления SCSS

```
.title {
  font-size: 20px;
}

.message {
  width: 50px;
}
```

Листинг 15 – Неправильное использование арифметических операций

```
$title-size: 10px;
$message-width: 200px;

.title {
```

```
    font-size: $title-size * 2px
}

.message {
    width: $message-width + 10em;
}
```

В листинге 15 при неправильном использовании арифметических операций в SCSS вы получите сообщение об ошибке за попытку использовать недопустимые единицы в CSS.

8 Директивы управления

SCSS - это декларативный язык сценариев, расширение CSS, а не язык программирования. Несмотря на это, он имеет некоторые ограниченные процедурные возможности через свои управляющие директивы.

8.1 Директива @if

Как и следовало ожидать, директива @if и сопутствующие ей @else if и @else позволяют включать код в вывод CSS только при соблюдении определенных условий. Базовый синтаксис прост.

Листинг 16 – Синтаксис @if

```
@if <логическое выражение> {
    <операция>
}
```

Конечно, вы можете использовать столько директив @if в блоке кода, сколько захотите. Но часто хочется “сделать это если верно, иначе сделать другое”. Вот тут и пригодятся директивы @else if и @else.

Листинг 17 – Пример с @if, @else if и @else

```
$value: 12;

a {
    @if $value < 3 {
        color: red;
    }
    @else if $value == 3 {
```

```
        color: blue;
    } @else {
        color: white;
    }
}
```

Листинг 18 – Результаты примера с @if, @else if и @else

```
a {
  color: white;
}
```

При создании сложных условных выражений рекомендуется всегда включать предложение @else. Это гарантирует, что что-то будет выполнено, даже если все другие условия ложны.

8.2 Директива @while

Директива @if выполняет набор операторов за один раз на основе логического выражения. Если вы хотите выполнять операторы несколько раз, но при этом контролировать их выполнение в зависимости от условия, вы можете использовать директиву @ while. Как следует из названия, директива @ while будет продолжать выводить CSS, созданный операторами, пока условие возвращает истину.

Синтаксис директивы @ while очень похож на синтаксис @if. Просто замените ключевое слово.

Обратите внимание на использование строковой интерполяции для ссылки на переменную $p в листинге 19 (#{$p}).

Листинг 19 – Пример с @while

```
$p: 3;

@while $p < 5 {
  .item-#{$p} {
        color: red;
        $p : $p + 1;
    }
}
```

Листинг 20 – Результаты примера с @while

```
.item-3 {
  color: red;
}

.item-4 {
  color: red;
}
```

Единственное, что вам нужно быть осторожными. Убедитесь, что один или несколько операторов, оцениваемых в цикле, изменят результат условного выражения. В противном случае просто продолжится выводиться CSS, созданный операторами, пока вы вручную не отмените его.

8.3 Директива @for

Вы можете использовать директиву @for для выполнения группы операторов определенное количество раз.

Листинг 21 – Синтаксис @if

```
@for <переменная> from <старт> through <конец> {
    <операции>
}
```

Листинг 22 – Пример с @for

```
@for $i from 1 through 3 {
   .list-#{$i} {
      width: 2px * $i;
   }
}
```

Листинг 23 – Результаты примера с @for

```
.list-1 {
  margin-left: 2px;
}

.list-2 {
  margin-left: 4px;
}

.list-3 {
  margin-left: 6px;
}
```

Замена ключевого слова through на to делает цикл эксклюзивным, то есть он не будет выполняться, если переменная равна <конец>.

Значения, указанные в <старт> и <конец>, должны быть численными значениями, но <старт> не должно быть меньше, чем <конец>. В противном случае значение переменной будет уменьшаться, а не увеличиваться.

Листинг 24 – Еще один пример с @for

```
@for $i from 3 through 1 {
   .list-#{$i} {
      width: 2px * $i;
   }
}
```

Листинг 25 – Результаты еще одного примера с @for

```
.list-3 {
  margin-left: 2px;
}

.list-2 {
  margin-left: 4px;
}

.list-1 {
  margin-left: 6px;
}
```

8.4 Директива @each

Наконец, директива @each выполнит набор элементов в списке. Для такой мощной структуры синтаксис так же довольно прост.

Листинг 26 – Синтаксис @each

```
@each <переменные> in <список переменных> {
   <операция>
}
```

Листинг 27 – Пример с @each

```
@each $s in (normal, bold, italic) {
  .#{$s} {font-weight: $s;}
}
```

Листинг 28 – Результат примера с @each

```
.normal {
  font-weight: normal;
}

.bold {
  font-weight: bold;
}

.italic {
  font-weight: italic;
}
```

Вы также можете передать многомерный список в директиву @each, указав несколько переменных, тем самым использовать метод, называемый множественным присваиванием.

В итоге, можно заметить, что директивы управления превращают процесс написания стилей в процесс похожий на программирование и также помогают решать некоторые задачи написанием меньшего количества кода, следовательно более элегантным способом.

9 Функции

Функции позволяют вам определять сложные операции со значениями, которые вы можете повторно использовать в своих стилях. Они позволяют легко абстрагироваться от общих формул и поведения в удобно читаемой форме.

Листинг 29 – Синтаксис @function

```
@function <имя>(<...аргументы>) {
      <операции>
      @return <значение>
}
```

Имя функции может быть любым уникальным идентификатором SCSS. Функция может содержать операторы, а также @return, указывающий на значение, которое будет использоваться в результате вызова функции.

Листинг 30 – Пример использования функции

```
@function pow($base, $exponent) {
	$result: 1;
	@for $_ from 1 through $exponent {
		$result: $result * $base;
	}
	@return $result;
}

.sidebar {
	float: left;
	margin-left: pow(4, 3) * 1px;
}
```

Листинг 31 – Результат использования функции

```
.sidebar {
	float: left;
	margin-left: 64px;
}
```

Функции SCSS, пожалуй, одни из самых полезных аспектов языка. Вы можете писать свои собственные функции и в дополнение к ним использовать функции, которые изначально предоставляет SCSS.