Изучаем с нуля

frontend edition

sass-scss.ru

Содержание

Документация по Sass FE edition

Версия книги: 1.0.0

Текущая версия Sass: 

Если вы слышали что-то про Sass, или вы хотите изучать Sass, но у вас плохо с английским языком, то эта книга для **вас**.

Документация по Sass на русском языке. Перевод официальной документации для фронтенда. Бэкенд будет затронут в другой книге.

Книга бесплатна в любой момент времени!

Изучайте

Подпишитесь на обновления книги и получайте оповещения о изменениях в книге.

Участие

Если вы хотите помочь с переводом или вычиткой текста - **свяжитесь со мной** через форму на гитбуке.

Основные ресурсы

> Русскоязычный портал: sass-scss.ru

Кстати, на русскоязычном портале, в документации, к каждому разделу подключены комментарии, а это значит, что Вы там можете задавать вопросы.

> Англоязычный портал: sass-lang.com

Changelog

Версия: 1.0.0

- Исправлен перевод
- Убраны спорные переводы
- Исправлены все ссылки, теперь они все ведут туда, куда надо
- Исправлены некоторые ошибки
- Исправлена ошибка в структуре документации

Версия: 0.9.2.5

- Исправлена ссылка на установщик Ruby для Windows
- Добавлены комментарии по тексту
- Выбран правильный перевод для спорных названий функций
- Исправлены некоторые ошибки

- Исправлена ошибка в структуре документации

Преимущества

- **Полная совместимость с CSS**

  > Sass полностью совместим со всеми версиями CSS. Мы уделяем серьезное внимание совместимости, поэтому вы можете легко использовать любые доступные библиотеки CSS.

- **Богатый функционал**

  > Sass может похвастаться большим количеством возможностей, чем любой другой язык расширения CSS. Команда Sass Core бесконечно работает не только для поддержания этих возможностей, но и для того, чтобы быть впереди.

- **Опыт**

  > Sass находится в активной разработке более 8 лет.

- **Поддержка индустрией**

  > Снова и снова сообщество разработчиков выбирает Sass как главное средство написания CSS.

- **Большое сообщество**

  > Sass активно поддерживается и разрабатывается консорциумом высокотехнологичных компаний и нескольких сотен разработчиков.

- **Фреймворки**

  > Есть бесконечное количество фреймворков, построенных на Sass. Compass, Bourbon, и Susy - это только несколько примеров из всего количества.

Синтаксис

Существует два варианта синтаксиса Sass.

SCSS

Первый вариант известен как SCSS (Sassy CSS) и он является расширением синтаксиса CSS. Это означает, что любое допустимое значение в CSS3 стилях будет допустимо и в SCSS. Кроме того, SCSS понимает большинство CSS-хаков и синтаксис вендорных префиксов, например, старый префикс IE filter. Этот синтаксис усиливается с применением возможностей Sass, описанных далее. Файлы этого варианта синтаксиса имеют расширение `.scss`.

SASS

Второй вариант и к тому же самый старый, известен как синтаксис отступов (или просто Sass). Этот синтаксис обеспечивает более краткий вариант написания CSS. Он использует отступы вместо ковычек, указывающие на вложение селекторов, и новые строки, заменяющие запятые, для разделения свойств. Некоторые пользователи считают, что этот синтаксис легче читается и быстрее пишется, чем SCSS.

Синтаксис отступов имеет все те же функции, хотя и некоторые из них имеют немного другой вид написания, об этом можно более подробно ознакомиться в главе документации 'SASS - синтаксис отступов'. Файлы этого варианта синтаксиса имеют расширение `.sass`.

Любой из вариантов синтаксиса может импортировать файлы написанные в другом варианте синтаксиса. Файлы могут автоматически конвертироваться в другой с помощью команды `sass-convert` в командной строке (терминале):

```
# Convert Sass to SCSS
$ sass-convert style.sass style.scss

# Convert SCSS to Sass
$ sass-convert style.scss style.sass
```

Данная команда не генерирует CSS-файл. Для компиляции в CSS-файл используйте команду `sass`.

Использование Sass

Sass может использоваться тремя путями:

1. Как инструмент в командной строке
2. Как отдельный модуль Ruby
3. Как модуль для любого, поддерживающего Rack, фреймворка, в том числе Ruby On Rails и Merb.

Первым шагом для начала работы является установка gem'a Sass:

```
gem install sass
```

Если для работы вы используете Windows, сначала вам необходимо установить Ruby.

Для запуска Sass из командной строки используйте команду[1]:

```
sass input.scss output.css
```

Также вы можете указать Sass следить за файлом и автоматически его компилировать в CSS при любом изменении:

```
sass --watch input.scss:output.css
```

Если у вас в папке имеется несколько файлов Sass, то вы можете указать Sass следить за всей папкой:

```
sass --watch app/sass:public/stylesheets
```

Используйте команду `sass --help` для получения полной документации.

Использовать Sass в Ruby также очень просто. После установки gem'a Sass вы сможете его использовать указав `require "sass"` и с помощью Sass::Engine. Например:

```
engine = Sass::Engine.new("#main {background-color: #0000ff}", :syntax => :scss)
engine.render #=> "#main { background-color: #0000ff; }\n"
```

---

[1] *Перед запуском команды не забудьте создать input.scss*

Использование в Rack/Rails/Merb

Для подключения Sass в Ruby версии ниже 3, укажите в файле `environment.rb` следующую строку:

```
config.gem "sass"
```

Для Ruby 3 вместо вышенаписанного сделайте запись в Gemfile:

```
gem "sass"
```

Для подключения Sass в Merb версии ниже 3, укажите в файле `config/dependencies.rb` следующую строку:

```
dependency "merb-haml"
```

Для подключения Sass в Rack-приложении, укажите в файле `config.ru` :

```
require 'sass/plugin/rack'
use Sass::Plugin::Rack
```

Сам по себе Sass не работает как таблица стилей. Он не оборачивает динамический контент, для того чтобы все работало, нужно сразу же после обновления Sass-файла генерировать CSS-файл.

По-умолчанию, файлы `.sass` и `.scss` находятся в `public/stylesheets/sass` (можно изменить с помощью опции `:template_location`). Затем, когда это необходимо, они компилируются в соответствующие CSS-файлы в `public/stylesheets` . Например, `public/stylesheets/sass/main.scss` будет скомпилирован в `public/stylesheets/main.css` .

Кеширование

По-умолчанию, Sass кеширует компилируемые шаблоны и фрагменты. Это значительно ускоряет повторную компиляцию больших коллекций Sass-файлов, и работает лучше всего, если шаблоны Sass разделены на отдельные файлы, которые импортируются в один файл.

Без использования фреймворков Sass кладет скешированные шаблоны в папку `sass-cache`. В Rails и Merb в папку `tmp/sass-cache`. Папка может быть изменена с помощью опции `:cache_location` .Если Вы не хотите чтобы Sass использовал кеш, то можете отключить данную функцию, если параметру `:cache` зададите значение `false`.

Настройки

Опции могут быть заданы по формату `Sass::Plugin#options` и записаны в `environment.rb` в Rails или в `config.ru` в Rack...

```
Sass::Plugin.options[:style] = :compact
```

... или настройкой `Merb::Plugin.config[:sass]` с записью в `init.rb` в Merb...

```
Merb::Plugin.config[:sass][:style] = :compact
```

... или путем передачи хеш в `Sass::Engine#initialize`. Все соответсвующие опции доступны через метки sass и scss в командной строке.

Доступные опции:

- `:style` - задает стиль выходного файла. Читайте стили выходного файла.
- `:syntax` - синтаксис компилируемого файла. Укажите `:sass` для синтаксиса отступов или `:scss` для синтаксиса подобия CSS. Эта опция используется, если вы работаете с ядром Sass (`Sass::Engine`). Параметр задается автоматически при использовании `Sass::Plugin`. Значение по-умолчанию: `:sass`.
- `:property_syntax` - задает правило, для документов с синтаксисом отступов, использовать один синтаксис свойств. Если правильный синтаксис не используется, то выдается ошибка. Значение `:new` устанавливает правило постановки двоеточия после свойства. Например: `color: #0f3` или `width: $main_width`. Значение `:old` устанавливает правило постановки двоеточия до свойства. Например: `:color #0f3` или `:width $main_width`. По-умолчанию, оба варианта валидны. Данная опция не работает для файлов SCSS.
- `:cache` - позволяет кешировать разобранные файлы Sass для увеличения скорости работы. Значение по-умолчанию `true`.
- `:read_cache` - если это опция указана, а `:cache` нет, то кеш будет только читаться(если он существует), без записи нового.
- `:cache_store` - если опция установлена на подкласс `Sass::CacheStores::Base`, то хранилище кеша будет использоваться хранения и извлечения кешированных результатов компиляции. По-умолчанию `Sass::CacheStores::Filesystem` активируется опцией `:cache_location`.
- `:never_update` - используется когда обновление CSS-файла не требуется, даже если обновился шаблон. Использование данной опции может дать небольшой прирост производительности. По-умолчанию значение данной опции всегда `false`. Имеет смысл использовать в Rack, Ruby On Rails или Merb.
- `:always_update` - используется когда CSS-файл должен обновляться каждый раз после того, как контроллер дает доступ, а не только когда шаблон был изменен. По-умолчанию значение данной опции всегда `false`. Имеет смысл использовать в Rack, Ruby On Rails или Merb.
- `:always_check` - используется когда необходима постоянная проверка Sass-файла на изменения, а не только при запуске сервера. Если файл SCSS был изменен, то произойдет его перекомпиляция в соответствующий CSS-файл. По-умолчанию в режиме разработки значение данной опции - `false`, в других случаях - `true`. Имеет смысл использовать в Rack, Ruby On Rails или Merb.

- `:poll` - когда значение опции `true`, всегда используйте опрос серверной части для `Sass::Plugin::Compiler#watch`, а не стандартной файловой системы.

- `:full_exception` - опция, которая отвечает за полное описание ошибки в Sass-файле в генерируемом CSS. Если значение опции `true`, то ошибка будет отображаться вместе с номером строки и исходным кодом в качестве комментария в файле CSS и на верху странички(только в поддерживаемых браузерах). В противном случае в коде Ruby будет добавлено исключение. По-умолчанию значение `false` в промышленном режиме и `true` в других.

- `:template_location`

Выбор синтаксиса

Если вы работаете с Sass через командную строку, то синтаксис определяется через расширение файла. Однако, синтаксис можно выбрать не только расширением файла. Синтаксис по-умолчанию - синтаксис отступов, но вы можете изменить синтаксис применив параметр `--scss` и тогда входной файл будет читаться как файл с синтаксисом SCSS. Кроме того, вы можете при синтаксисе `scss` использовать команды синтаксиса `sass` и они будут также исполняться.

Кодировка

Когда используется Ruby 1.9 или более поздние версии, Sass сам определяет кодировку файла. Для определения кодировки Sass использует CSS спецификации для определения кодировки таблицы стилей, после определения Sass возвращает кодировку в Ruby для правильной компиляции. Это означает, что сначала идет считывание порядок байтов Unicode, далее директиву `@charset`, потом кодировку строк Ruby. Если ни один из них строго не заданы, то документу присваивается значение по-умолчанию - `UTF-8`.

Чтобы явно указать кодировку таблицы стилей, используйте директиву `@charset` как в обычном CSS. Добавьте конструкцию `@charset "имя кодировки"` в начале таблицы стилей (не пропуская пробелов и перед любыми комментариями) и Sass будет интерпретировать это как заданную кодировку. Обратите внимание на то, что какую бы вы не использовали кодировку, она должна конвертироваться в Unicode.

Sass will always encode its output as UTF-8. It will include a @charset declaration if and only if the output file contains non-ASCII characters. In compressed mode, a UTF-8 byte order mark is used in place of a @charset declaration.

Расширения CSS

Вложенные правила

Sass позволяет вкладывать правила CSS друг в друга. Вложенные правила применяются только для элементов, соответствующих внешним селекторам. Например, такая запись в Sass:

```
#main p {
  color: #00ff00;
  width: 97%;

  .redbox {
    background-color: #ff0000;
    color: #000000;
  }
}
```

компилируется в такой код CSS:

```
#main p {
  color: #00ff00;
  width: 97%; }
  #main p .redbox {
    background-color: #ff0000;
    color: #000000; }
```

Это помогает избежать повторения родительских селекторов, и делает сложные CSS макеты с большим количеством вложенных селекторов гораздо проще. Например:

```
#main {
  width: 97%;

  p, div {
    font-size: 2em;
    a { font-weight: bold; }
  }

  pre { font-size: 3em; }
}
```

компилируется в:

```
#main {
  width: 97%; }
  #main p, #main div {
    font-size: 2em; }
    #main p a, #main div a {
      font-weight: bold; }
  #main pre {
    font-size: 3em; }
```

Ссылка на родителя селектора

Иногда полезно использовать родительский селектор с другим предназначением, чем по-умолчанию. Например, вы можете пожелать специальные стили для случая, когда над выбранными элементами находится курсор или когда тело элемента имеет определенный класс. В этих случаях, с помощью символа `&` вы можете явно указать, где должен быть вставлен родительский селектор, например:

```
a {
  font-weight: bold;
  text-decoration: none;
  &:hover { text-decoration: underline; }
  body.firefox & { font-weight: normal; }
}
```

компилируется в:

```
a {
  font-weight: bold;
  text-decoration: none; }
  a:hover {
    text-decoration: underline; }
  body.firefox a {
    font-weight: normal; }
```

Символ `&` будет заменен на родительский селектор, каким он компилируется в CSS. Это означает, что если у вас есть глубоко вложенное правило, родительский селектор будет полностью вычислен до символа `&` и подставится вместо него. Например:

```
#main {
  color: black;
  a {
    font-weight: bold;
    &:hover { color: red; }
  }
}
```

компилируется в:

```
#main {
  color: black; }
  #main a {
    font-weight: bold; }
    #main a:hover {
      color: red; }
```

Символ `&` должен быть указан в начале составного селектора, также он может содержать последующий суффикс, который будет добавлен к родительскому селектору. Например:

```
#main {
  color: black;
  &-sidebar { border: 1px solid; }
}
```

компилируется в:

```
#main {
  color: black; }
  #main-sidebar {
    border: 1px solid; }
```

Если суффикс не применим к родительскому селектору, Sass выдаст ошибку.

Вложенные свойства

CSS имеет довольно много свойств в пространстве имен, например, `font-family`, `font-size` и `font-weight` находятся в пространстве имен font.

В CSS, если вы хотите установить набор свойств в общем пространстве имен, вы должны вводить его каждый раз. Sass упрощает этот процесс: просто напишите пространство имен единожды, а внутри вы можете вписать любое из вторичных свойств. Например:

```
.funky {
  font: {
    family: fantasy;
    size: 30em;
    weight: bold;
  }
}
```

компилируется в:

```
.funky {
  font-family: fantasy;
  font-size: 30em;
  font-weight: bold; }
```

Само пространство имен также может иметь значение. Например:

```
.funky {
  font: 20px/24px fantasy {
    weight: bold;
  }
}
```

компилируется в:

```
.funky {
  font: 20px/24px fantasy;
    font-weight: bold;
}
```

Шаблонные селекторы

Sass поддерживает специальный тип селекторов под названием шаблонные селекторы. Они похожи на селекторы классов и идентификаторов, только вместо `#` или `.` указывается `%`. Они предназначены для использования с директивой `@extend`.

Для получения дополнительной информации см. селекторы расширения.

Сами по себе, без использования `@extend`, наборы правил, которые используют шаблонные селекторы не окажут влияния на CSS. Например:

```
%inline-type {
    display: inline;
    zoom: 1;
    display: inline-block;
    margin-right: -4px;
    vertical-align: top;
}

%for-grids {
    min-height: 1px;
    position: relative;
    padding-left: 10px;
    padding-right: 10px;
    margin-bottom: 20px;
}

.sgrid-N {
    @extend %for-grids;
}
```

компилируется в

```
.sgrid-N {
    min-height: 1px;
    position: relative;
    padding-left: 10px;
    padding-right: 10px;
    margin-bottom: 20px; }
```

Комментирование

Sass поддерживает стандартный многострочный CSS комментарий, обозначаемый `/* */`, а также однострочный комментарий, обозначаемый `//`. Многострочные комментарии сохраняются на выходе в файле CSS, где это возможно, в то время как однострочные комментарии удаляются. Например:

```
/* Это многострочный комментарий.
 * Так как этот тип комментирования
 * поддерживается в CSS, то он
 * попадет в скомпилированный CSS-файл. */
body { color: black; }

// Это однострочный комментарий.
// CSS не поддерживает данный вариант комментирования.
// Поэтому данного комментария не будет в CSS-файле.
a { color: green; }
```

компилируется в:

```
/* Это многострочный комментарий.
 * Так как этот тип комментирования
 * поддерживается в CSS, то он и
 * попадет в скомпилированный CSS-файл. */
body { color: black; }

a { color: green; }
```

Если первым символом многострочного комментария является знак `!`, то комментарий всегда будет оставаться в результирующем CSS, даже в режиме сжатия. Это полезно, например, для добавления уведомления об авторских правах на ваш CSS.

Поскольку многострочные комментарии становятся частью результирующего CSS, в них разрешена интерполяция. Например:

```
$version: "1.1.3";
/* Этот CSS построен с использованием SomeGrids. Версия: #{$version}. */
```

компилируется в:

```
/* Этот CSS построен с использованием SomeGrids. Версия: 1.1.3. */
```

SassScript

В дополнение к обычному синтаксису CSS, Sass поддерживает небольшой набор расширений под названием SassScript. SassScript добавляет поддержку использования переменных, арифметических и экстра функций. SassScript могут быть использованы в любом значении свойств.

SassScript также может быть использован для создания селекторов и имен свойств, что полезно при написании миксинов. Это делается с помощью интерполяции.

Интерактивная оболочка

Вы можете легко экспериментировать с SassScript с помощью интерактивной оболочки. Для запуска оболочки запустите Sass из командной строки с параметром -i. В командной строке введите любое поддерживаемое SassScript выражение, чтобы оценить его работу, результат выведется для вас на экран:

```
$ sass -i

>> "Hello, Sassy World!"
"Hello, Sassy World!"

>> 1px + 1px + 1px
3px

>> #777 + #777
#eeeeee

>> #777 + #888
white
```

Переменные

Самое простое, что поддерживает SassScript - это использование переменных. Переменные начинаются со знака доллара (`$`) и задаются как свойства CSS:

```
$width: 5em;
```

Вы можете обратиться к ним из свойств:

```
#main {
  width: $width;
}
```

Переменные доступны только в пределах того уровня вложенности селекторов, на котором они определены. Если они определяются вне каких-либо вложенных селекторов, они доступны глобально. Если вы хотите, чтобы объявленная на каком-либо уровне вложенности переменная также была доступна глобально, вы можете определить её со специальной меткой `!global` . Например:

```
#main {
  $width: 5em !global;
  width: $width;
}

#sidebar {
  width: $width;
}
```

компилируется в

```
#main {
  width: 5em;
}

#sidebar {
  width: 5em;
}
```

По историческим причинам, в именах переменных (и прочих Sass идентификаторах) дефисы и подчеркивания взаимозаменяемы. Например, если вы определяете переменную `$main-width` , вы можете получить к ней доступ, написав `$main_width` , и наоборот.

Типы данных

SassScript поддерживает семь основных типов данных:

- числа (1.2, 13, 10px)
- текстовые строки, с кавычками и без них ("foo", 'bar', baz)
- цвета (blue, #04a3f9, rgba(255, 0, 0, 0.5))
- булевы значения (true, false)
- null
- списки значений, с разделительными пробелами или запятыми (1.5em 1em 0 2em; Times New Roman, Arial, sans-serif)
- мапы (например: (key1: value1, key2: value2))

SassScript также поддерживает все другие виды данных CSS, такие как **диапазоны Unicode** и декларации **!important**, однако для них не имеется специальных способов обращения, они используются точно также, как строки без кавычек.

Строки

CSS определяет два вида строк: будь то с кавычками, как "Lucida Grande" или 'http://sass-scss.ru', или же без них, как sans-serif или bold. SassScript признает оба вида строк и, в общем случае, какой тип строк вы используете в Sass документе, тот вы и получите в результирующем CSS. Существует одно исключение: когда используется `#{}` интерполяция, кавычки игнорируются. Это упрощает использование, скажем, имен селекторов в миксинах. Например:

```
@mixin firefox-message($selector) {
  body.firefox #{$selector}:before {
    content: "Hi, Firefox users!";
  }
}

@include firefox-message(".header");
```

компилируется в

```
body.firefox .header:before {
  content: "Hi, Firefox users!"; }
```

Списки

Списки в Sass представлены значениями CSS-деклараций вроде `margin: 10px 15px 0 0` или `font-face: Helvetica, Arial, sans-serif`. Списки - это всего навсего серия из нескольких однотипных значений разделенных пробелами или запятыми. Фактически, индивидуальные значения также являются списками, просто это списки, которые состоят из одного элемента.

Сами по себе списки не делают ничего особенного, но список функций SassScript сделал их полезнее. Функия `nth` может получить доступ к элементам в списке, функция `join` может объединить несколько списков вместе, а функция `append` может добавлять элементы в список. Директива `@each` также может добавлять стили для каждого элемента в списке.

В придачу к обычным значениям, списки могут содержать также другие списки. Например, **1px 2px**, **5px 6px** - это список из двух элементов, который содержит список **1px 2px** и список **5px 6px**. Если внутренние списки содержат тот же разделитель, что и внешний список, необходимо использовать скобки, чтобы было понятно где начинается и где заканчивается внутренний список. Например, **(1px 2px) (5px 6px)** - это список аналогичный предыдущему. Разница в том, что разделители у этого внешнего списка - пробелы, тогда как предыдущий список разделялся запятыми.

Когда списки превращаются в обычный CSS, Sass не добавляет никаких скобок, ибо CSS их не понимает. Это значит, что **(1px 2px) (5px 6px)** и **1px 2px 5px 6px** будут выглядеть идентично, когда она станут CSS. Тем не менее, это не тоже самое для Sass: первый список содержит два списка, тогда как второй содержит 4 числа.

Список может также не содержать значений вообще. Такой список объявляется как `()`, который также является пустым мапом. Они не могут быть непосредственно преобразованы в CSS, если вы попытаетесь написать что-то вроде `font-family: ()`, то Sass выдаст ошибку. Если список содержит пустые списки или `null`-значения, например, как **1px 2px () 3px** или **1px 2px null 3px**, то пустые списки и `null`-значения будут вырезаны при переводе в CSS.

Разделенные запятыми списки могут заканчиваться запятой. Это крайне полезно потому что позволяет представить единичный список. Например, **(1,)** это список, содержащий **1** и **(1 2 3,)**, из одного элемента, разделенного запятыми, и содержащий подсписок из трех элементов, разделенных пробелами.

Мапы (Ассоциативные массивы)

Мапы, или ассоциативные массивы, представляют собой связь между ключами и значениями, где ключи используются для поиска значений. Они позволяют легко собирать значения в именованные группы и получать доступ к этим группам динамически. Они не имеют прямого аналога в CSS, хотя они синтаксически похожи на media-запросы:

```
$map: (key1: value1, key2: value2, key3: value3);
```

В отличие от списков, мапы всегда должны быть заключены в скобки и всегда должны быть разделены запятыми. И ключи и значения в мапах могут быть любыми объектами SassScript. Мап может иметь только одно сопоставление с заданным ключом (хотя это значение может быть списком). Значение ключа, конечно же, может быть сопоставлено многим ключам.

Как и со списками, манипуляции с мапами в основном происходят через функции SassScript. Функция *map-get* смотрит значение в мапе, функция *map-merge* добавляет значения в мапе. Директива `@each` может быть использована для добавления стилей для каждой пары *ключ-значение* в мапе. Порядок пар всегда остается неизменным с момента создания мапа.

Мапы можно использовать везде, где применимы списки. Если для манипуляции картой используются функции для списков, то мапы будут рассматриваться как список пар. Например, **(key1: value1, key2: value2)** будет рассматриваться этими функциями как вложенный список **key1 value1**, **key2 value2**. Однако, обратное утверждение верно только для пустого списка `()`, в остальных случаях списки не могут рассматриваться как мапы. `()` представляет собой одновременно мапы без пар *ключ-значение* и список без элементов.

Заметьте, что ключами мапа может быть любой тип Sass данных (даже другой массив) и синтаксис для объявления карт допускает произвольное применение SassScript-выражений, которые будут рассчитаны для определения ключа мапа.

Мапы не могут быть преобразованы в обычный CSS. Использование мапов в качестве значения переменной или аргумента CSS функции приведет к ошибке. Использование функций `inspect($value)` для получения результирующей строки полезно для отладки мапов.

Цвета

Любое выражение цвета CSS возвращает значение цвета SassScript. Это включает в себя большое количество именованных цветов, которые неотличимы от незаключенных в кавычки строк.

В режиме сжатия, Sass будет выводить наименьшее CSS представление такого цвета. Например, `#FF0000` в сжатом режиме будет выводиться в CSS как **red**, но **blanchedalmond** выведет, как `#FFEBCD`.

Распространенная проблема с которой сталкиваются пользователи именованных цветов возникает из-за предпочтения Sass того же формата, который был напечатан в других режимах, использование названия цвета в имени селектора является недопустимым синтаксисом при сжатии. Чтобы избежать этого, всегда заключайте именованные цвета в кавычки, если они предназначены для использования в конструкции селектора.

Операции

Все типы данных поддерживают операции равенства (`==` и `!=`). Кроме того, каждый тип поддерживает собственные операции, применимые для конкретного типа данных.

Числовые операции

SassScript поддерживает стандартные арифметические операции над числами (сложение `+`, вычитание `-`, умножение `*`, деление `/`, и остаток от деления по модулю `%`). Математические функции Sass сохраняют значение промежуточных результатов в течение выполнения арифметических операций. Это означает, что, как и в реальной жизни, вы не можете работать с числами у которых несовместимы типы данных (например, складывание px и em) и двух чисел одного типа, перемножение которых даст квадратные единицы (`10px * 10px == 100px * px`).

Помните, что **px * px** это невалидный тип данных CSS, и вы получите сообщение об ошибке от Sass за попытку использовать недопустимые единицы в CSS.

Операторы сравнения (<, >, <=, >=, ==, !=), также поддерживаются для чисел.

Деление и знак "/"

CSS использует символ `/` как способ обособления численных данных наряду с пробелами и запятыми, а не для деления чисел. Поскольку SassScript является расширением синтаксиса CSS свойств, он должен поддерживать это, в то же время позволяя применять `/` для деления чисел друг на друга. Это означает, что по-умолчанию, если два числа, разделены символом `/` в SassScript, то они вместе с разделетилем окажутся в результирующем CSS.

Тем не менее, существует три ситуации, в которых символ `/` будет интерпретироваться именно как деление. Они охватывают подавляющее большинство случаев, когда подразумевается именно деление:

- Если значение или любая его часть, хранится в переменной или возвращается функцией
- Если значения заключены в круглые скобки
- Если значение используется как часть другого арифметического выражения

Например:

```
p {
  font: 10px/8px;             // Явный CSS, деление отсуствует
  $width: 1000px;
  width: $width/2;            // Используется переменная, операция деления
  width: round(1.5)/2;        // Используется функция, операция деления
  height: (500px/2);          // Обособление скобками, операция деления
  margin-left: 5px + 8px/2px; // Используется +, операция деления
}
```

компилируется в

```
p {
  font: 10px/8px;
  width: 500px;
  height: 250px;
  margin-left: 9px; }
```

Если вы хотите использовать символ `/` в переменной, в значении обычного CSS `/`, вы можете заключить переменную в `#{}`. Например:

```
p {
  $font-size: 12px;
  $line-height: 30px;
  font: #{$font-size}/#{$line-height};
}
```

компилируется в

```
p {
  font: 12px/30px; }
```

Операции с цветами

Все арифметические операции поддерживаются для значений цветов, в которых они могут быть применены для каждого цвета в отдельности. Это означает, что операция выполняется поочередно отдельно для красной, зеленой и синей составляющей цвета. Например:

```
p {
  color: #010203 + #040506;
}
```

вычислит *01 + 04* = **05**, *02 + 05* = **07**, и *03 + 06* = **09**, в результате будет получен цвет:

```
p {
  color: #050709; }
```

Часто полезнее использовать функции цвета, чем пытаться использовать арифметику, чтобы достичь того же эффекта. Арифметические операции между числами и цветами также работают поочередно для каждой составляющей. Например:

```
p {
  color: #010203 * 2;
}
```

вычислит *01 * 2* = **02**, *02 * 2* = **04**, и *03 * 2* = **06**, и преобразуется в:

```
p {
  color: #020406; }
```

Обратите внимание, что цвета с альфа-каналом (те, которые создаются через rgba или hsla функции) должны иметь одинаковое значение прозрачности для того, чтобы проводить с ними арифметические операции, поскольку арифметика не влияет на значение альфа-канала. Например:

```
p {
  color: rgba(255, 0, 0, 0.75) + rgba(0, 255, 0, 0.75);
}
```

компилируется в

```
p {
  color: rgba(255, 255, 0, 0.75); }
```

Альфа-канал цвета можно регулировать с помощью функций opacify и transparentize. Например:

```
$translucent-red: rgba(255, 0, 0, 0.5);
p {
  color: opacify($translucent-red, 0.3);
  background-color: transparentize($translucent-red, 0.25);
}
```

компилируется в

```
p {
  color: rgba(255, 0, 0, 0.8);
  background-color: rgba(255, 0, 0, 0.25); }
```

IE фильтры требуют, чтобы все цвета включали в себя альфа-слой, и были в строгом формате #AABBCCDD. Вы можете легко конвертировать цвет с помощью функции ie_hex_str. Например:

```
$translucent-red: rgba(255, 0, 0, 0.5);
$green: #00ff00;
div {
  filter: progid:DXImageTransform.Microsoft.gradient(enabled='false', startColorstr='#{ie-hex-str($green)}', endColorst
}
```

компилируется в

```
div {
  filter: progid:DXImageTransform.Microsoft.gradient(enabled='false', startColorstr=#FF00FF00, endColorstr=#80FF0000);
}
```

Операции со строками

Операция конкатенации строк осуществляется при помощи символа `+`. Например:

```
p {
  cursor: e + -resize;
}
```

компилируется в

```
p {
  cursor: e-resize; }
```

Обратите внимание, что, если мы к строке без кавычек добавляем строку в кавычках (строка без кавычек слева от знака `+`), то результатом будет строка без кавычек. И, наоборот, если мы к строке в кавычках прибавляем строку без кавычек, результатом будет строка в кавычках. Например:

```
p:before {
  content: "Foo " + Bar;
  font-family: sans- + "serif";
}
```

компилируется в

```
p:before {
  content: "Foo Bar";
  font-family: sans-serif; }
```

По-умолчанию, если два значения находятся рядом друг с другом, они объединяются через пробел:

```
p {
  margin: 3px + 4px auto;
}
```

компилируется в

```
p {
  margin: 7px auto; }
```

Возможно использовать интерполяцию `#{}` для размещения динамических значений в строке:

```
p:before {
  content: "I ate #{5 + 10} pies!";
}
```

компилируется в

```
p:before {
  content: "I ate 15 pies!"; }
```

Значение `null` обрабатывается как пустая строка в интерполяции:

```
$value: null;
p:before {
  content: "I ate #{$value} pies!";
}
```

компилируется в

```
p:before {
  content: "I ate  pies!"; }
```

Логические операции

SassScript поддерживает операторы **И**, **ИЛИ** и **НЕ** (and, or, not) для логических значений. Например:

```
$menuOn: true;
$menuDropDown: false;

.menu li {
    color: #ff0000;
    text-decoration: underline;
    @if $menuOn == true and $menuDropDown == true {
        position:relative;
    }
}
```

компилируется в:

```
.menu li {
    color: #ff0000;
    text-decoration: underline;
}
```

Операции со списками

Списки не поддерживают каких-либо специальных операций. Для работы с ними существуют специальные функции списков.

Круглые скобки

Скобки могут использоваться для определения порядка действий:

```
p {
  width: 1em + (2em * 3);
}
```

компилируется в

```
p {
  width: 7em; }
```

Функции

SassScript содержит некоторые полезные функции, которые можно вызвать, используя обычный синтаксис CSS:

```
p {
  color: hsl(0, 100%, 50%);
}
```

компилируется в

```
p {
  color: #ff0000; }
```

Вы можете увидеть весь список доступных функций на этой странице.

Именованные аргументы

В функции Sass можно передавать именованные аргументы используя карты. Предыдущий пример может быть записан следующим образом:

```
p {
  color: hsl($hue: 0, $saturation: 100%, $lightness: 50%);
}
```

Несмотря на то, что запись стала длиннее, такое выражение легче прочесть. Это также предоставляет более гибкий интерфейс для написания функций, в котором увеличение количества аргументов не увеличивает трудности.

Именованные аргументы можно передавать в любом порядке, а аргумент со значением по-умолчанию может быть пропущен. Так как именованные аргументы являются переменными, знак подчеркивания и дефис являются взаимозаменяемыми.

Смотрите Sass::Script::Функции, чтобы ознакомиться со всеми функциями Sass и названиями их аргументов, а также инструкциями, описывающими как объявить собственные функции на Ruby.

Интерполяция

Вы также можете использовать переменные SassScript в селекторах и в названиях свойств используя синтаксис `#{}` интерполяции:

```
$name: foo;
$attr: border;
p.#{$name} {
  #{$attr}-color: blue;
}
```

компилируется в

```
p.foo {
  border-color: blue; }
```

Также можно использовать `#{}`, чтобы вставить в SassScript значение свойств. В большинстве случаев это не лучше, чем любой другой способ, но использование `#{}` означает, что любые операции в нем будут рассматриваться как запись CSS. Например:

```
p {
  $font-size: 12px;
  $line-height: 30px;
  font: #{$font-size}/#{$line-height};
}
```

компилируется в

```
p {
  font: 12px/30px; }
```

Символ "&" в SassScript

Также как и в объявлении ссылки на родителя, оператор `&` в SassScript относится к текущему родительскому селектору. Это список, разделенный запятыми, состоящий из списков, разделенных пробелами. Например:

```
.foo.bar .baz.bang, .bip.qux {
  $selector: &;
}
```

В этом примере значением **$selector** является **((".foo.bar" ".baz.bang"), ".bip.qux")**. Составные селекторы приведены, чтобы показать, что они строковые, хоть и указывались без кавычек. Даже если родительский селектор не содержит запятую или пробел, `&` всегда будет иметь два уровня вложенности, поэтому он может быть доступен постоянно.

Если нет родительского селектора, значение `&` будет нулевым. Это означает, что вы можете использовать его в миксинах, чтобы проверить наличие родительского селектора:

```
@mixin does-parent-exist {
  @if & {
    &:hover {
      color: red;
    }
  } @else {
    a {
      color: red;
    }
  }
}
```

Переменные по-умолчанию

Вы можете присваивать значение по-умолчанию переменным, у которых еще нет значения, добавив метку `!default` в конце значения. В таком случае, если переменная уже имела значение, то оно не изменится; если же переменная пуста, ей будет присвоено новое указанное значение.

Например:

```
$content: "Тестовый текст";
$content: "Новый тестовый текст?" !default;
$new_content: "Как пройти в библиотеку?" !default;

#main {
  content: $content;
  new-content: $new_content;
}
```

компилируется в

```
#main {
  content: "Тестовый текст";
  new-content: "Как пройти в библиотеку?"; }
```

Переменные, имеющие значение `null`, рассматриваются меткой `!default` как не имеющие значение:

```
$content: null;
$content: "НЕ-null контент" !default;

#main {
  content: $content;
}
```

компилируется в

```
#main {
  content: "НЕ-null контент"; }
```

Правила и директивы

Sass поддерживает все CSS3 @-правила, а также дополнительные специфические правила Sass, называемые "директивами". Они дают различные эффекты в Sass, которые мы рассмотрим далее. Читайте также директивы контроля и миксины.

Директива @import

Sass расширяет CSS правило[1] `@import`, позволяя импортировать **SCSS** и **SASS** файлы. Все импортированные **SCSS** и **SASS** файлы могут быть объеденены в одном результирующем CSS-файле. Кроме того, любые переменные или миксины, объявленные в импортируемые файлы, могут использоваться в главном файле.

Sass looks for other Sass files in the current directory, and the Sass file directory under Rack, Rails, or Merb. Дополнительные каталоги поиска могут задаваться с помощью опции `:load_paths` или ключ `--load-path` в командной строке.

`@import` использует название файла для импорта. По-умолчанию, `@import` ищет Sass-файлы, но есть несколько правил, по которым `@import` отрабатывает как CSS-правило:

- Если расширение файла `.css`
- Если имя файла начинается с `http://`
- Если имя файла вызывается через `url()`
- Если правило `@import` включает в себя любые медиа-запросы

If none of the above conditions are met and the extension is .scss or .sass, then the named Sass or SCSS file will be imported. If there is no extension, Sass will try to find a file with that name and the .scss or .sass extension and import it.

Например:

```
@import "foo.scss";
```

или

```
@import "foo";
```

будет импортирован `foo.scss`, в то время как:

```
@import "foo.css";
@import "foo" screen;
@import "http://foo.com/bar";
@import url(foo);
```

скомпилируется в

```
@import "foo.css";
@import "foo" screen;
@import "http://foo.com/bar";
@import url(foo);
```

Также возможно импортирование нескольких файлов через одну директиву `@import`. Например:

```
@import "rounded-corners", "text-shadow";
```

будут импортированы файлы `rounded-corners` и `text-shadow`.

Импортирование может содержать в себе интерполяцию `#{}`, но только с некоторыми ограничениями. Невозможно динамически импортировать файлы Sass через переменные; интерполяция нужна только для правила `@import` в CSS. Как таковая, интерполяция работает только с `url()`.

```
$family: unquote("Droid+Sans");
@import url("http://fonts.googleapis.com/css?family=#{$family}");
```

скомпилируется в

```
@import url("http://fonts.googleapis.com/css?family=Droid+Sans");
```

---

[1] *Правилом является только функционал с возможностями CSS, директива - функционал правила CSS + расширение функционалом Sass.*

Фрагменты

Если у Вас есть **SCSS** или **SASS** файл, который вы хотите импортировать, но не хотите его компилировать напрямую в CSS-файл, то добавьте нижнее подчеркивание в начало имени файла. Это сообщит Sass о том, что не нужно компилировать этот файл напрямую в CSS. Когда импортируете такой файл, то нижнее подчеркивание можно не указывать.

Например, у вас есть файл `_color.scss`. Никакого `_colors.css` не будет создано, вам всего лишь надо написать:

```
@import "colors";
```

и файл `_color.scss` будет импортирован.

Обратите внимание на то, что невозможно импортировать фрагмент и полноценный файл Sass с одинаковым именем, а также и находящимися в одной папке, одновременно. Например, файл `colors.scss` не может сосуществовать с фрагментом `_colors.scss`.

Вложенный импорт

Самое популярное место использования `@import` является начало документа, но с Sass директиву можно использовать и в CSS-свойствах и в медиа-условиях. Как и в базовом варианте, `@import` включает в себя контент импортируемого файла. Тем не менее, импортированные правила будут вложены в том же месте, где и вызывалась директива `@import`.

Например, файл `example.scss` содержит в себе:

```
.example {
  color: red;
}
```

далее импортируем

```
#main {
  @import "example";
}
```

компилируется в

```
#main .example {
  color: red;
}
```

Директивы, которые могут использоваться только на базовом уровне документа, такие как `@mixin` и `@charset`, не допускаются в файлах, которые подключаются вложенным импортом.

Невозможно вкладывать `@import` в миксин или директиву управления.

Директива (@media)

Директива `@media` работает также как и стандартное правило CSS, только с дополнительными возможностями: директива может вкладываться в правила CSS. Если директива вложена в CSS-правило, то при компиляции она будет поднята наверх таблицы стилей, а все селекторы в которых была директива переместятся внутрь `@media`. Такой метод позволяет легко добавлять правила в `@media` без повторения селекторов или нарушения потока таблицы стилей. Например:

```
.sidebar {
  width: 300px;
  @media screen and (orientation: landscape) {
    width: 500px;
  }
}
```

компилируется в

```
.sidebar {
  width: 300px; }
  @media screen and (orientation: landscape) {
    .sidebar {
      width: 500px; } }
```

`@media` запросы могут вкладываться друг в друга. После компиляции эти `@media` будут объеденены оператором `and`.

```
@media screen {
  .sidebar {
    @media (orientation: landscape) {
      width: 500px;
    }
  }
}
```

компилируется в

```
@media screen and (orientation: landscape) {
  .sidebar {
    width: 500px; } }
```

Ну и наконец, `@media` запросы могут содержать в себе все возможности SassScript(включая переменные, функции и операторы) в именах компонентов и различных значений. Например:

```
$media: screen;
$feature: -webkit-min-device-pixel-ratio;
$value: 1.5;

@media #{$media} and ($feature: $value) {
  .sidebar {
    width: 500px;
  }
}
```

компилируется в

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5) {
  .sidebar {
    width: 500px; } }
```

Директива @extend

Бывают случаи, когда один селектор должен иметь все стили другого, а также свои собственные. Самый распространенный способ это использовать общий селектор, либо более специфический. Допустим, у нас есть дизайн обычной ошибки и серьезной ошибки. Мы используем такой вариант разметки:

```
<div class="error seriousError">
  Внимание! Ваш аккаунт был взломан.
</div>
```

а стили наши выглядят так:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  border-width: 3px;
}
```

К сожалению, это означает, что мы должны помнить о том, что класс `.error` нужно использовать с классом `.seriousError`. Такой расклад является не удобным, а бывает даже так, что это несет в себе семантические ошибки в разметке.

Директива `@extend` позволяет избежать данных проблем, сообщая Sass, что один селектор должен наследовать стили другого. Например:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
```

компилируется в

```
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd;
}

.seriousError {
  border-width: 3px;
}
```

Это означает, что все объявляемые стили для `.error` будут применяться и к `.seriousError`, в дополнение к характерным для `.seriousError`. В результате получается, что все элементы с классом `.seriousError` технически также будут иметь класс `.error`.

Другие правила, используемые `.error` будут также применяться и для `.seriousError`. Например, у нас есть специальные стили для ошибок, вызванных атакой хакеров:

```
.error.intrusion {
  background-image: url("/image/hacked.png");
}
```

Тогда элемент `<div class="seriousError intrusion">` также будет иметь фон из `hacked.png`.

Как это работает

`@extend` работает по принципу включения расширяющего селектора (в нашем случае это `.seriuosError`) в любой набор стилей, где встречается расширяемый селектор (в нашем случае - `.error`). Таким образом, пример из предыдущей главы:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.error.intrusion {
  background-image: url("/image/hacked.png");
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
```

компилируется в

```
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd; }

.error.intrusion, .seriousError.intrusion {
  background-image: url("/image/hacked.png"); }

.seriousError {
  border-width: 3px; }
```

При слиянии селекторов директива `@extend` не допустит ошибок в виде дублирования, например, `.seriuosError.seriuosError`, самостоятельно исправляя данные ошибки на `.seriuosError`. Кроме того, директива не будет создавать селекторы, которые не смогут ничего значить, например, как `#main#footer`.

Расширение комплексным селектором

Селекторы классов не единственные вещь которые могут быть расширены. Это позволяет расширять любой селектор, включая одиночные элементы, такие как `.special.cool`, `a:hover` или `a.user[href^="http://"]`. Например:

```
.hoverlink {
  @extend a:hover;
}
```

Также, как и с классами, это означает, что все стили объявленные для `a:hover` также применятся и к `.hoverlink`. Например:

```
.hoverlink {
  @extend a:hover;
}
a:hover {
  text-decoration: underline;
}
```

компилируется в

```
a:hover, .hoverlink {
  text-decoration: underline; }
```

Также, как и с `.error.intrusion`, описанным в предыдуших главах, любые правила используемые `a:hover` будут также применятся и к `.hoverlink`, даже если с ними используются другие селекторы. Например:

```
.hoverlink {
  @extend a:hover;
}
.comment a.user:hover {
  font-weight: bold;
}
```

компилируется в

```
.comment a.user:hover, .comment .user.hoverlink {
  font-weight: bold; }
```

Множественные расширения

Одиночный селектор может быть расширен больше, чем одним селектором. Это означает, что он наследует все стили всех расширяющих селекторов. Например:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.attention {
  font-size: 3em;
  background-color: #ff0;
}
.seriousError {
  @extend .error;
  @extend .attention;
  border-width: 3px;
}
```

компилируется в

```
.error, .seriousError {
  border: 1px #f00;
  background-color: #fdd; }

.attention, .seriousError {
  font-size: 3em;
  background-color: #ff0; }

.seriousError {
  border-width: 3px; }
```

В результате, каждый элемент с классом `.seriousError` будет иметь класс `.error`, *а также* класс `.attention`. Таким образом, стили объявленные позже в документе имеют приоритет: фон `.seriousError` имеет цвет `#ff0` и этот цвет предпочтительнее, чем `#fdd`, поскольку `.attention` объявлен позже, чем `.error`.

Множественное расширение может быть записано списком селекторов, разделенных запятыми. Например, `@extend .error, .attention` означает тоже самое, что и `@extend .error; @extend .attention`.

Цепные расширения

Цепное расширение возможно для селектора, который расширяется другим селектором, который в свою очередь расширяется третьим селектором. Например:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
.criticalError {
  @extend .seriousError;
  position: fixed;
  top: 10%;
  bottom: 10%;
  left: 10%;
  right: 10%;
}
```

Теперь все элементы с классом `.seriousError` будут иметь класс `.error` и все элементы с классом `.criticalError` будут иметь классы `.seriousError` *и* `.error`. Вышеприведенный пример компилируется в:

```
.error, .seriousError, .criticalError {
  border: 1px #f00;
  background-color: #fdd; }

.seriousError, .criticalError {
  border-width: 3px; }

.criticalError {
  position: fixed;
  top: 10%;
  bottom: 10%;
  left: 10%;
  right: 10%; }
```

Последовательность селекторов

Последовательность селекторов, такая как `.foo .bar` или `.foo + .bar`, в настоящее время не может быть расширена. Однако, это становится возможным при использовании вложенных селекторов через директиву `@extend`. Например:

```
#fake-links .link {
  @extend a;
}

a {
  color: blue;
  &:hover {
    text-decoration: underline;
  }
}
```

компилируется в

```
a, #fake-links .link {
  color: blue; }
  a:hover, #fake-links .link:hover {
    text-decoration: underline; }
```

Объединение последовательностей селекторов

Иногда последовательность селекторов расширяет другой селектор, который также является часть последовательностью. В этом случае эти две последовательности селекторов должны быть объеденены. Например:

```
#admin .tabbar a {
  font-weight: bold;
}
#demo .overview .fakelink {
  @extend a;
}
```

Несмотря на то, что технически можно было бы сгенерировать все селекторы, которые могли бы соотвтетсвовать любой последовательности, этого не происходит, так как это сделало бы таблицу стилей слишком большой. Вышенаписанному примеру потребовалось бы десять селекторов. Вместо этого Sass генерирует только те селекторы, которые могут быть полезны.

Когда объединяются две последовательности, не имеющих общих селекторов, то генерируются новые селекторы: один с первой последовательностью перед вторым и один со второй последовательностью перед первым. Например:

```
#admin .tabbar a {
  font-weight: bold;
}
#demo .overview .fakelink {
  @extend a;
}
```

компилируется в

```
#admin .tabbar a,
#admin .tabbar #demo .overview .fakelink,
#demo .overview #admin .tabbar .fakelink {
  font-weight: bold; }
```

Если две последовательности имеют какой то общий селектор, то эти селекторы будут объеденены вместе и только различия (если таковые существуют) будут чередоваться. В следующем примере обе последовательности имеют идентификатор `#admin`, значит, в результате эти два идентификатора будут объеденены:

```
#admin .tabbar a {
  font-weight: bold;
}
#admin .overview .fakelink {
  @extend a;
}
```

компилируется в

```
#admin .tabbar a,
#admin .tabbar .overview .fakelink,
#admin .overview .tabbar .fakelink {
  font-weight: bold; }
```

Селекторы расширения

Иногда вы будет писать стили для класса, которого вы хотите использовать только с директивой `@extend` и не хотите использовать непосредственно в HTML. Это особенно важно при написание Sass библиотеки, где вы можете передавать стили пользователям через `@extend`, если им это нужно и игнорировать если нет.

Если вы используете для этого обычные классы, то в конце концов вы можете столкнуться с тем, что у вас таблица стилей намного больше, чем могло бы быть и даже конфликты с другими классами, которые используются в HTML. Вот почему Sass поддерживает шаблонные селекторы(например, `%foo`).

Шаблонные селекторы выглядят также как и селекторы классов или идентификаторов, только `.` или `#` заменены на `%`. Они могут быть использованы в любом месте, где могут находится классы или идентификаторы, и сами по себе они не дают набора правил в генерируемом CSS файле. Например:

```
// Данный набор прпвил не будет сгенерирован
#context a%extreme {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

Тем не менее, шаблонные селекторы могут быть расширены, также как классы или идентификаторы. Расширенные селекторы будут сгенерированы, а базовый шаблонный селектор - нет. Например:

```
.notice {
  @extend %extreme;
}
```

компилируется в

```
#context a.notice {
  color: blue;
  font-weight: bold;
  font-size: 2em; }
```

Метка необязательности (!optional)

Обычно, когда вы расширяете селектор, то получаете ошибку, если директива `@extend` не отработала. Например, если вы напишете `a.important {@extend .notice}`, то получите ошибку, если ни один селектор не будет содержать `.notice`. Также вы получите ошибку, если селектор содержащий в себе `.notice` будет таким: `h1.notice`. Ошибка происходит из-за конфликта между `a` и `h1`, и, исходя из данной ошибки, новый селектор не будет сгенерирован.

Иногда, если нужно, вы можете разрешить директиве `@extend` не создавать каких-либо новых селекторов. Сделать это можно добавив метку неоязательности `!optional` после селектора. Например:

```
a.important {
  @extend .notice !optional;
}
```

Вложение @extend в директивы

Существуют некоторые ограничения применения `@extend` в директивах, таких как `@media`. Sass не в силах внедрить правила CSS, которые находятся снаружи блока `@media` и относятся к селектору внутри него, не создавая огромное количество стилей, благодаря копированию всех стилей в каждом случае. Это означает, что если вы используете `@extend` в `@media` (или в других директивах), то вы можете расширить только те селекторы, которые находятся в том же блоке директивы.

Например, правильное использование:

```
@media print {
  .error {
    border: 1px #f00;
    background-color: #fdd;
  }
  .seriousError {
    @extend .error;
    border-width: 3px;
  }
}
```

и неправильное:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}

@media print {
  .seriousError {
    // INVALID EXTEND: .error is used outside of the "@media print" directive
    @extend .error;
    border-width: 3px;
  }
}
```

> Некорректное расширение: .error используется вне директивы "@media print"

Когда-нибудь мы надеемся на полную поддержку `@extend` в браузерах, что позволит использовать её в **media** и других директивах без ограничений.

Директива @at-root

`@at-root` изымает один или несколько правил из родительского селектора в корневой уровень документа. Такой способ можно использовать с селекторами первого уровня вложения:

```
.parent {
  ...
  @at-root .child { ... }
}
```

В итоге получим:

```
.parent { ... }
.child { ... }
```

Также этот способ может быть использован с несколькими селекторами:

```
.parent {
  ...
  @at-root {
    .child1 { ... }
    .child2 { ... }
  }
  .step-child { ... }
}
```

На выходе получим:

```
.parent { ... }
.child1 { ... }
.child2 { ... }
.parent .step-child { ... }
```

Использование условий

По-умолчанию, `@at-root` изымает только селекторы. Тем не менее, `@at-root` можно использовать и для изъятия селекторов, вложенных в директивы, таких как `@media`. Например:

```
@media print {
  .page {
    width: 8in;
    @at-root (without: media) {
      color: red;
    }
  }
}
```

компилируется в

```
@media print {
  .page {
    width: 8in;
  }
}
.page {
  color: red;
}
```

Вы можете использовать `@at-root (without: ...)` для изъятия из любой директивы. Также вы можете изымать селекторы сразу из нескольких директив, просто разделив их пробелами: `@at-root (without: media supports)` изымет элементы из запросов `@media` и `@supports`.

Существует два пециальных значения у `@at-root`, которые вы также можете использовать. **"rule"** относится к обычным правилам CSS.

- Команда `@at-root (without: rule)` имеет такое же значение, как и `@at-root` без запросов.
- Команда `@at-root (without: all)` обозначает, что стили должны быть изъяты из *всех* директив и правил CSS.

Если вы хотите указать какие директивы или правила включать, а не список тех, которые должны быть изъяты, то вы можете использовать **with** вместо **without**. Например, команда `@at-root (with: rule)` означает, что необходимо изъять элементы из всех директив, но не затрагивать вложенные правила CSS.

Директива @debug

`@debug` выводит значения функций Sass средствами стандартного вывода ошибок. Это полезно для отладки функций Sass, особенно для тех, что содержат сложную структуру данных. Например:

```
@debug 10em + 12em;
```

получим результат:

```
Line 1 DEBUG: 22em
```

Директива @warn

`@warn` выводит значения выражений Sass средствами стандартного вывода ошибок. Данная директива полезна для библиотек, которым нужно предупреждать пользователей об использовании устаревших или восстановленных после незначительных ошибок миксинов. Есть два основных различия между **warn** и **debug**:

1. Вы можете отключить отображение предупреждений опцией `--quiet` в командной строке или опцией Sass `:quiet`.
2. Предупреждение будет выведено вместе с отрывком таблицы стилей, чтобы пользователь мог понять, где в коде требуется его внимание.

Пример:

```
@mixin adjust-location($x, $y) {
  @if unitless($x) {
    @warn "Предположительно, значение #{$x} задано в пикселях";
    $x: 1px * $x;
  }
  @if unitless($y) {
    @warn "Предположительно, значение #{$y} задано в пикселях";
    $y: 1px * $y;
  }
  position: relative; left: $x; top: $y;
}
```

Директива @error

`@error` отображает значение выражений и функций Sass как фатальную ошибку, включая нормальную часть стека трассировки. Эта директива полезна для проверки аргументов миксинов и функций. Например:

```
@mixin adjust-location($x, $y) {
  @if unitless($x) {
    @error "$x не может быть безразмерным, было #{$x}.";
  }
  @if unitless($y) {
    @error "$y не может быть безразмерным, было #{$y}.";
  }
  position: relative; left: $x; top: $y;
}
```

В настоящее время в примере нет возможности поймать ошибку.

Управляющие директивы и выражения

SassScript поддерживает базовые управляющие директивы и выражения для подключения стилей при определённых условиях или подключения одних и тех же стилей несколько раз с изменениями.

Обратите внимание: управляющие директивы - это продвинутые возможности, не для повседневного использования. Они предназначены в основном для использования в миксинах, частично для тех, которые являются частью таких библиотек, как Compass и, соответственно, требующих существенной гибкости.

Функция if()

Встроенная функция **if()** позволяет осуществлять ветвление, опираясь на условие, и возвращает один из двух возможных результатов. Она может быть использована в любом месте скрипта. Функция **if** вычисляет только тот аргумент, который будет возвращён - это позволяет ссылаться на переменные, которые могут быть не определены или иметь вычисления, которые в других случаях привели бы к ошибке (например, деление на ноль).

Директива @if

Директива **@if** принимает выражение SassScript и использует стили, вложенные в неё в случае, если выражение возвращает любое значение, кроме **false** или **null**:

```
p {
  @if 1 + 1 == 2 { border: 1px solid;  }
  @if 5 < 3      { border: 2px dotted; }
  @if null       { border: 3px double; }
}
```

компилируется в

```
p {
  border: 1px solid; }
```

После выражения **@if** может следовать несколько выражений @else if и одно выражение **@else**. Если выражение **@if** вернёт **false**, то будут произведиться попытки вычисления выражений **@else if**, пока одно из них не вернёт истину или до достижения выражения **@else**. Например:

```
$type: monster;
p {
  @if $type == ocean {
    color: blue;
  } @else if $type == matador {
    color: red;
  } @else if $type == monster {
    color: green;
  } @else {
    color: black;
  }
}
```

компилируется в

```
p {
  color: green; }
```

Директива @for

Директива **@for** выводит набор стилей заданное число раз. Для каждого повторения используется переменная-счётчик для изменения вывода.

Директива имеет две формы: **@for $var from** *<начало>* **through** *<конец>* и **@for $var from** *<начало>* **to** *<конец>*. Заметьте различие в словах **through** и **to**.

$var может быть любым именем переменной, таким как **$i**; *<начало>* и *<конец>* - выражения SassScript, которые должны возвращать целые числа. Если *<начало>* больше, чем *<конец>*, счётчик будет убывать, вместо того, чтобы расти.

Выражение **@for** устанавливает переменную **$var** в каждое последующее значение в заданном диапазоне и каждый раз выводит вложенные стили, используя очередное значение переменной **$var**. Форма **from ... through**, диапазон включает значения *<начало>* и *<конец>*, а форма **from ... to** не включает значение *<конец>*. При использовании синтаксиса **through**,

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

компилируется в

```
.item-1 {
  width: 2em; }
.item-2 {
  width: 4em; }
.item-3 {
  width: 6em; }
```

Директива @each

Директива **@each** обычно имеет вид **@each $var in** *<список или карта значений>*. **$var** может быть любой переменной, такой как **$length** или **$name**, и *<список или карта значений>* - это выражение SassScript, возвращающее список или карту значений.

Директива **@each** устанавливает **$var** в каждое из значений списка или карты и выводит содержащиеся в ней стили, используя соответствующее значение **$var**. Например:

```
@each $animal in puma, sea-slug, egret, salamander {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
  }
}
```

компилируется в

```
.puma-icon {
  background-image: url('/images/puma.png'); }
.sea-slug-icon {
  background-image: url('/images/sea-slug.png'); }
.egret-icon {
  background-image: url('/images/egret.png'); }
.salamander-icon {
  background-image: url('/images/salamander.png'); }
```

Множественные значения

Директива **@each** также может использовать несколько переменных по следующему принципу: **@each $var1**, **$var2**, **... in** *<список>*. Если *<список>* - это список списков, то значение каждого элемента внутреннего списка будет назначено соответствующей переменной. Например:

```
@each $animal, $color, $cursor in (puma, black, default),
                                  (sea-slug, blue, pointer),
                                  (egret, white, move) {
  .#{$animal}-icon {
    background-image: url('/images/#{$animal}.png');
    border: 2px solid $color;
    cursor: $cursor;
  }
}
```

компилируется в

```
.puma-icon {
  background-image: url('/images/puma.png');
  border: 2px solid black;
  cursor: default; }
.sea-slug-icon {
  background-image: url('/images/sea-slug.png');
  border: 2px solid blue;
  cursor: pointer; }
.egret-icon {
  background-image: url('/images/egret.png');
  border: 2px solid white;
  cursor: move; }
```

Так как мапы обрабатываются как списки или пары значений, множественное присваивание с ними тоже работает. Например:

```
@each $header, $size in (h1: 2em, h2: 1.5em, h3: 1.2em) {
  #{$header} {
    font-size: $size;
  }
}
```

компилируется в

```
h1 {
  font-size: 2em; }
h2 {
  font-size: 1.5em; }
h3 {
  font-size: 1.2em; }
```

Директива @while

Директива **@while** принимает выражение SassScript и циклично выводит вложенные в неё стили, пока выражение вычисляется как **true**. Она может быть использована для создания более сложных циклов, чем тех, для которых подходит **@for**, хотя она бывает необходима довольно редко. Например:

```
$i: 6;
@while $i > 0 {
  .item-#{$i} { width: 2em * $i; }
  $i: $i - 2;
}
```

компилируется в

```
.item-6 {
  width: 12em; }

.item-4 {
  width: 8em; }

.item-2 {
  width: 4em; }
```

Миксины

Миксины (примеси) позволяют определить стили, которые могут быть использованы повторно в любом месте документа без необходимости прибегать к несемантическим классам вроде `.float-left`. Миксины также могут содержать целые CSS правила или что-либо другое, разрешённое в Sass документе. Они даже могут принимать аргументы, что позволяет создавать большое разнообразие стилей при помощи небольшого количества миксинов.

Объявление миксина

Миксины объявляются директивой **@mixin**. После неё должно стоять имя миксина и, опционально, его параметры, и блок, содержащий тело миксина. Например, можно определить миксин `large-text` следующим образом:

```
@mixin large-text {
  font: {
    family: Arial;
    size: 20px;
    weight: bold;
  }
  color: #ff0000;
}
```

Миксины могут также содержать селекторы, возможно со свойствами. Селекторы даже могут содержать ссылки на родителя. Например:

```
@mixin clearfix {
  display: inline-block;
  &:after {
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
  }
  * html & { height: 1px }
}
```

По историческим причинам, имена миксинов (и всех других идентификаторов в Sass) могут содержать дефисы и символы подчёркивания как взаимозаменяемые символы. Например, если вы определяете миксин `add-column`, вы можете подключать его как `add_column` и наоборот.

Использование миксина

Миксины вызываются в документ директивой **@include**. Она принимает имя миксина и, опционально, передаваемые в него аргументы, и включает стили, определённые этим миксином, в текущее правило. Например:

```
.page-title {
  @include large-text;
  padding: 4px;
  margin-top: 10px;
}
```

компилируется в

```
.page-title {
  font-family: Arial;
  font-size: 20px;
  font-weight: bold;
  color: #ff0000;
  padding: 4px;
  margin-top: 10px; }
```

Миксины могут быть также вызваны вне какого-либо правила (то есть в корне документа), при условии, что они не определяют непосредственно правил и не используют ссылку на родителя. Например:

```
@mixin silly-links {
  a {
    color: blue;
    background-color: red;
  }
}

@include silly-links;
```

компилируется в

```
a {
  color: blue;
  background-color: red; }
```

Миксины также могут включать в себя другие миксины. Например:

```
@mixin compound {
  @include highlighted-background;
  @include header-text;
}

@mixin highlighted-background { background-color: #fc0; }
@mixin header-text { font-size: 20px; }
```

Миксины могут включать самих себя. Этим современный Sass отличается от версий, предшествующих 3.3, где рекурсия в миксинах была запрещена.

Миксины, которые содержат только селекторы для дочерних элементов, могут безопасно быть перенесены на самый верхний уровень документа.

Аргументы

Миксины могут принимать значения SassScript как аргументы, которые передаются при подключении миксина и становятся доступными как переменные внутри него.

При определении миксина, аргументы, пишутся как имена переменных, разделённые запятыми, внутри круглых скобок сразу после имени. Затем, при подключении миксина, значения могут быть переданы аналогичным образом. Например:

```
@mixin sexy-border($color, $width) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}

p { @include sexy-border(blue, 1in); }
```

компилируется в

```
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed; }
```

В миксинах также можно определять значения аргументов по-умолчанию, используя обычный синтаксис установки значений переменных. Затем, при подключении миксина, если ему не будет передан аргумент, то будет использовано значение по-умолчанию. Например:

```
@mixin sexy-border($color, $width: 1in) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}
p { @include sexy-border(blue); }
h1 { @include sexy-border(blue, 2in); }
```

компилируется в

```
p {
  border-color: blue;
  border-width: 1in;
  border-style: dashed; }

h1 {
  border-color: blue;
  border-width: 2in;
  border-style: dashed; }
```

Именованные аргументы

Миксины также можно вызывать используя явно именованные аргументы. Например, вышеуказанный пример мог бы быть записан как:

```
p { @include sexy-border($color: blue); }
h1 { @include sexy-border($color: blue, $width: 2in); }
```

Данный способ записи делает код менее кратким, но более читабельным. Также он позволяет функциям представлять более гибкие интерфейсы: большое количество параметров у функции не усложняет её вызов.

Именованные аргументы можно передавать в любом порядке, аргументы, имеющие значения по-умолчанию, могут быть пропущены. Так как именованные аргументы - это имена переменных, символы подчёркивания и дефисы могут быть использованы как взаимозаменяемые.

Переменные аргументы

Иногда имеет смысл миксину или функции принимать неизвестное количество аргументов. Например, миксин для создания тени блока может принимать любое количество значений тени. Для подобных ситуаций Sass поддерживает возможность передачи "переменных аргументов" - это такие аргументы, которые передаются последними в функцию или миксин, получают все остальные переданные параметры и упаковывают их в список. Эти аргументы выглядят как обычные, но после них следует многоточие. Например:

```
@mixin box-shadow($shadows...) {
  -moz-box-shadow: $shadows;
  -webkit-box-shadow: $shadows;
  box-shadow: $shadows;
}

.shadows {
  @include box-shadow(0px 4px 5px #666, 2px 6px 10px #999);
}
```

компилируется в

```
.shadows {
  -moz-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
  -webkit-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
  box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;
}
```

Переменные аргументы также содержат любые именованные аргументы, переданные в функцию или миксин. К ним можно обратиться через функцию `keywords($args)`, которая возвращает их как соответствие имён (без знака $) значениям (оригинал: which returns them as a map from strings (without $) to values).

Переменные аргументы также можно использовать в миксинах. Используя тот же синтаксис, можно развернуть список значений так, что каждое значение будет передано как отдельный параметр или развернуть карту значений так, что каждая пара будет трактована как именованный аргумент. Например:

```
@mixin colors($text, $background, $border) {
  color: $text;
  background-color: $background;
  border-color: $border;
}

$values: #ff0000, #00ff00, #0000ff;
.primary {
  @include colors($values...);
}

$value-map: (text: #00ff00, background: #0000ff, border: #ff0000);
.secondary {
  @include colors($value-map...);
}
```

компилируется в

```
.primary {
  color: #ff0000;
  background-color: #00ff00;
  border-color: #0000ff;
}
```

```
.secondary {
  color: #00ff00;
  background-color: #0000ff;
  border-color: #ff0000;
}
```

Вы можете передать одновременно список параметров и карту значений, но только если список передаётся перед мапом, как в этом примере: `@include colors($values..., $map...)`.

Вы можете использовать переменные аргументы для того, чтобы обернуть миксин и добавить в него дополнительные стили, не меняя сигнатуру аргументов этого миксина. Если вы это сделаете, то именованные аргументы будут непосредственно переданы в обёрнутый миксин. Например:

```
@mixin wrapped-stylish-mixin($args...) {
  font-weight: bold;
  @include stylish-mixin($args...);
}

.stylish {
  // Аргумент $width будет получен миксином "stylish-mixin" как именованный
  @include wrapped-stylish-mixin(#00ff00, $width: 100px);
}
```

Блоки контента в миксинах

Существует возможность передачи блока стилей в миксин, которые будут расположены среди стилей, подключаемых этим миксином. Эти стили будут размещены вместо директив `@content`, расположенных внутри миксина. Это даёт возможность создания абстракций, зависящих от конструкций селекторов или директив. Например:

```
@mixin apply-to-ie6-only {
  * html {
    @content;
  }
}
@include apply-to-ie6-only {
  #logo {
    background-image: url(/logo.gif);
  }
}
```

компилируется в

```
* html #logo {
  background-image: url(/logo.gif);
}
```

Те же самые миксины могут быть созданы с использованием краткого синтаксиса `.sass`:

```
=apply-to-ie6-only
  * html
    @content

+apply-to-ie6-only
  #logo
    background-image: url(/logo.gif)
```

Обратите внимание: если директива @content указана более одного раза или внутри цикла, блок стилей будет включён несколько раз - в соответствии с количеством её вызовов.

Область видимости переменных и блоков контента

Блоки контента, переданные в миксин, вычисляются в той же области видимости, где определён этот блок, а не миксин. Это значит, что локальные переменные миксина **не могут** быть использованы в передаваемом блоке контента и переменные будут восприняты как глобальные:

```
$color: white;
@mixin colors($color: blue) {
  background-color: $color;
  @content;
  border-color: $color;
}
.colors {
  @include colors { color: $color; }
}
```

компилируется в

```
.colors {
  background-color: blue;
  color: white;
  border-color: blue;
}
```

Кроме того, это означает, что переменные и миксины, которые используются в передаваемом блоке, зависят от других стилей, вблизи которых блок объявлен. Например:

```
#sidebar {
  $sidebar-width: 300px;
  width: $sidebar-width;
  @include smartphone {
    width: $sidebar-width / 3;
  }
}
```

Функции

Существует возможность определить собственные функции в Sass и использовать их в любом значении или контексте скрипта. Например:

```
$grid-width: 40px;
$gutter-width: 10px;

@function grid-width($n) {
  @return $n * $grid-width + ($n - 1) * $gutter-width;
}

#sidebar { width: grid-width(5); }
```

компилируется в

```
#sidebar {
  width: 240px; }
```

Как вы видите, функции имеют доступ к любым глобальным переменным, а также принимают параметры как и миксины (примеси). Функция может содержать несколько операторов, и вы должны вызвать `@return`, чтобы установить возвращаемое значение функции.

Также, как и миксины, определённые в Sass функции могут быть вызваны с именованными аргументами. В предыдущем примере мы могли бы вызвать функцию так:

```
#sidebar { width: grid-width($n: 5); }
```

Рекомендуется использовать префиксы для функций во избежание конфликтов имён, а также, чтобы читатель ваших таблиц стилей понимал, что эти функции не являются частью Sass или CSS. Например, если бы вы работали в компании ACME, то функция выше могла бы быть названа вами как `-acme-grid-width`.

Пользовательские функции также поддерживают переменные параметры, как и миксины.

По историческим причинам, имена функций (а также все остальные идентификаторы в Sass) могут содержать дефисы и символы подчёркивания как взаимозаменяемые символы. Например, если вы определили функцию `grid-width`, вы можете вызывать её как `grid_width` и наоборот.

Стили выходного файла

Хотя стиль вывода CSS, который Sass использует по-умолчанию, достаточно хорош и отражает структуру документа, вкусы и требования отличаются, и поэтому Sass поддерживает несколько других стилей.

Sass позволяет выбрать между четырьмя различными стилями вывода путём установки опции `:style` или использования параметра командной строки `--style`.

Вложенный стиль (:nested)

Вложенный (**nested**) стиль - это стиль Sass по-умолчанию, т.к. он отражает структуру CSS-стилей и HTML-документа, для которого они используются. Каждое свойство находится на отдельной строке, однако отступ не одинаков. Каждое правило имеет величину отступа, зависящую от глубины вложенности этого правила. Например:

```
#main {
  color: #fff;
  background-color: #000; }
  #main p {
    width: 10em; }

.huge {
  font-size: 10em;
  font-weight: bold;
  text-decoration: underline; }
```

Вложенный стиль очень полезен при просмотре больших CSS файлов: он позволяет легко охватить взглядом структуру файла, фактически не читая его.

Развернутый стиль (:expanded)

Развёрнутый (**expanded**) стиль похож на CSS, создаваемый человеком, где каждое свойство и правило занимает отдельную строку. Свойства имеют отступ внутри правил, но сами правила не имеют специальным образом расставленных отступов. Например:

```
#main {
  color: #fff;
  background-color: #000;
}
#main p {
  width: 10em;
}

.huge {
  font-size: 10em;
  font-weight: bold;
  text-decoration: underline;
}
```

Компактный стиль (:compact)

Компактный (**compact**) стиль занимает меньше места, чем вложенный или развёрнутый. Кроме того, он больше уделяет внимания селекторам, чем их свойствам. Каждое CSS правило занимает только одну строку, со всеми свойствами, определёнными в этой же строке. Вложенные правила располагаются следом друг за другом без дополнительного переноса, а отдельные группы правил разделены переносами строк. Например:

```
#main { color: #fff; background-color: #000; }
#main p { width: 10em; }

.huge { font-size: 10em; font-weight: bold; text-decoration: underline; }
```

Сжатый стиль (:compressed)

Сжатый (**compressed**) стиль занимает минимально возможное пространство, используя пробелы только для отделения селекторов друг от друга, а также перевода строки в конце файла. Он также включает в себя некоторые способы незначительного сжатия, такие как выбор кратчайшего представления цвета. Он не предназначен для чтения человеком. Например:

```
#main{color:#fff;background-color:#000}#main p{width:10em}.huge{font-size:10em;font-weight:bold;text-decoration:underli
```

Расширение Sass

Sass предоставляет множество продвинутых возможностей по кастомизации для пользователей с нестандартными требованиями. Использование этих возможностей требует хорошего владения Ruby.

Объявление пользовательских функций в Sass

Пользователи могут объявить собственные функции в Sass, используя Ruby API.

Чтобы узнать больше, смотрите документацию.

Хранилище кеша

Sass кеширует обработанные документы, поэтому они могут быть использованы заново без повторной обработки, если только они не были изменены.

По-умолчанию Sass сохраняет кеш по адресу в файловой системе, указанному в опции :cache_location. Если у вас не возможности записи в файловую систему или хотите предоставить доступ к вашему кешу другим процессам ruby или компьютерам, вы можете указать собственное место расположения кеша, установив значение опции :cache_store.

Более подробную информацию о создании собственного хранилиза кеша смотрите в документации.

Пользовательские импортеры

Импортёры Sass предназначены для загрузки Sass кода с переданных в них адресов. По-умолчанию эти адреса понимаются Sass как пути в файловой системе, однако существует возможность добавить импортёры, способные загружать код из баз данных, по HTTP или использовать отличную от ожидаемой Sass схему именования файлов.

Каждый импортёр ответственен за определённый путь загрузки (или какое-либо другое соответствующее представление бекенда). Импортёры могут быть размещены в массиве :load_paths вместе с обычными путями файловой системы.

При использовании директивы @import Sass просматривает пути для загрузки в поисках импортёра, который способен загрузить этот путь. Если таковой будет найден, то импортируемый файл будет использован.

Пользовательские импортёры должны наследоваться от класса Sass::Importers::Base.