# Introducing Sass Modules | CSS-Tricks

17–25 минут

---



Sass только что запустил важную новую функцию, которую вы можете узнать по другим языкам: **модульную систему** . Это большой шаг вперед для `@import`. одна из наиболее часто используемых функций Sass. Хотя текущее `@import`правило позволяет вам использовать сторонние пакеты и разбивать ваш Sass на управляемые «части», у него есть несколько ограничений:

- `@import`также является функцией CSS, и различия могут сбивать с толку
- Если вы используете `@import`один и тот же файл несколько раз, это может замедлить компиляцию, вызвать конфликты переопределения и создать дублированный вывод.
- Все находится в глобальном пространстве имен, включая сторонние пакеты, поэтому моя `color()`функция может переопределить вашу существующую `color()`функцию или наоборот.
- Когда вы используете такую функцию, как `color()`. невозможно точно знать, где он был определен. Откуда `@import`это?

  Авторы пакетов Sass (такие как я) пытались обойти проблемы с пространством имен, вручную добавляя префикс к нашим переменным и функциям, но модули Sass — гораздо более мощное решение. Короче говоря, `@import`заменяется более четкими `@use`и

`@forward`правилами. В течение следующих нескольких лет поддержка Sass `@import`будет объявлена устаревшей, а затем удалена. Вы по-прежнему можете использовать импорт CSS , но он не будет компилироваться Sass. Не волнуйтесь, есть инструмент миграции , который поможет вам выполнить обновление!

### Импортировать файлы с помощью @use

```
@use 'buttons';
```

Новый `@use`похож на `@import`. но имеет некоторые заметные отличия:

- Файл импортируется только один раз, независимо от того, сколько раз вы `@use`его использовали в проекте.
- Переменные, примеси и функции (то, что Sass называет «членами»), начинающиеся со знака подчеркивания ( `_`) или дефиса ( `-`), считаются частными и не импортируются.
- Элементы из используемого файла ( `buttons.scss`в данном случае) доступны только локально, но не передаются для будущего импорта.
- Точно так же `@extends`будет применяться только *вверх по цепочке;* расширение селекторов в импортированных файлах, но не расширение файлов, которые импортируют этот.
- Все импортированные члены *по умолчанию имеют пространство имен* .

  Когда мы `@use`создаем файл, Sass автоматически создает пространство имен на основе имени файла:

```
@use 'buttons'; // creates a `buttons` namespace
@use 'forms'; // creates a `forms` namespace
```

  Теперь у нас есть доступ к элементам из обоих `buttons.scss`и `forms.scss`, но этот доступ не передается между импортами: `forms.scss`по-прежнему нет доступа к переменным, определенным в `buttons.scss`. Поскольку импортированные функции имеют

пространство имен, мы должны использовать новый синтаксис с разделением по точкам для доступа к ним:

```
// variables: <namespace>.$variable
$btn-color: buttons.$color;
$form-border: forms.$input-border;

// functions: <namespace>.function()
$btn-background: buttons.background();
$form-border: forms.border();

// mixins: @include <namespace>.mixin()
@include buttons.submit();
@include forms.input();
```

Мы можем изменить или удалить пространство имен по умолчанию, добавив `as <name>` в импорт:

```
@use 'buttons' as *; // the star removes any namespace
@use 'forms' as f;

$btn-color: $color; // buttons.$color without a namespace
$form-border: f.$input-border; // forms.$input-border with
a custom namespace
```

Использование `as *` добавляет модуль в корневое пространство имен, поэтому префикс не требуется, но эти члены по-прежнему локально привязаны к текущему документу.

### Импорт встроенных модулей Sass

Внутренние функции Sass также переместились в модульную систему, поэтому у нас есть полный контроль над глобальным пространством имен. Есть несколько встроенных модулей — `math`, `color`, `string`, `list`, `map`, `selector` и `meta` — которые должны быть явно импортированы в файл перед их использованием:

```
@use 'sass:math';
$half: math.percentage(1/2);
```

Модули Sass также можно импортировать в глобальное пространство имен:

```
@use 'sass:math' as *;
$half: percentage(1/2);
```

Внутренние функции, имена которых уже имеют префиксы, такие как `map-get`или `str-index`. можно использовать без дублирования этого префикса:

```
@use 'sass:map';
@use 'sass:string';
$map-get: map.get(('key': 'value'), 'key');
$str-index: string.index('string', 'i');
```

Полный список встроенных модулей, функций и изменений названий вы можете найти в [спецификации модуля Sass](#) .

### Новые и измененные основные функции

В качестве дополнительного преимущества это означает, что Sass может безопасно добавлять новые внутренние миксины и функции, не вызывая конфликтов имен. Самый захватывающий пример в этом релизе — `sass:meta`миксин под названием `load-css()`. Это работает аналогично `@use`, но возвращает только сгенерированный вывод CSS и может использоваться динамически в любом месте нашего кода:

```
@use 'sass:meta';
$theme-name: 'dark';

[data-theme='#{$theme-name}'] {
  @include meta.load-css($theme-name);
}
```

Первый аргумент — это URL-адрес модуля (например, `@use`), но его

можно динамически изменять с помощью переменных и даже включать интерполяцию, например `theme-#{$name}`. Второй (необязательный) аргумент принимает карту значений конфигурации:

```
// Configure the $base-color variable in 'theme/dark'
before loading
@include meta.load-css(
  'theme/dark',
  $with: ('base-color': rebeccapurple)
);
```

Аргумент `$with`принимает ключи конфигурации и значения для любой переменной в загруженном модуле, если они оба:

- Глобальная переменная, которая не начинается с `_`или `-`(теперь используется для обозначения конфиденциальности).
- Помечено как `!default`значение, для настройки

```
// theme/_dark.scss
$base-color: black !default; // available for configuration
$_private: true !default; // not available because private
$config: false; // not available because not marked as a
!default
```

Обратите внимание, что `'base-color'`ключ будет устанавливать `$base-color`переменную.

Есть еще две `sass:meta`новые функции: `module-variables()`и `module-functions()`. Каждый возвращает карту имен элементов и значений из уже импортированного модуля. Они принимают один аргумент, соответствующий пространству имен модуля:

```
@use 'forms';

$form-vars: module-variables('forms');
// (
//   button-color: blue,
```

```
//   input-border: thin,
// )

$form-functions: module-functions('forms');
// (
//   background: get-function('background'),
//   border: get-function('border'),
// )
```

Несколько других `sass:meta`функций — `global-variable-exists()`, `function-exists()`, `mixin-exists()`и `get-function()`— получат дополнительные `$module`аргументы, что позволит нам явно проверять каждое пространство имен.

### Настройка и масштабирование цветов

В `sass:color`модуле также есть несколько интересных предостережений, поскольку мы пытаемся отойти от некоторых устаревших проблем. Многие из устаревших ярлыков, таких как `lighten()`. или `adjust-hue()`на данный момент устарели в пользу явных `color.adjust()`и `color.scale()`функций:

```
// previously lighten(red, 20%)
$light-red: color.adjust(red, $lightness: 20%);

// previously adjust-hue(red, 180deg)
$complement: color.adjust(red, $hue: 180deg);
```

Некоторые из этих старых функций (такие как `adjust-hue`) избыточны и не нужны. Другие — нравятся `lighten`. `darken`. `saturate`. и так далее — нужно перестроить с лучшей внутренней логикой. Первоначальные функции были основаны на `adjust()`. который использует линейную математику: добавление `20%`к текущей легкости `red`в нашем примере выше. В большинстве случаев нам действительно нужна `scale()`яркость в процентах относительно

текущего значения:

```
// 20% of the distance to white, rather than current-
lightness + 20
$light-red: color.scale(red, $lightness: 20%);
```

После того, как эти функции быстрого доступа будут полностью объявлены устаревшими и удалены, в конечном итоге они снова появятся `sass:color`с новым поведением, основанным `color.scale()`на `color.adjust()`. Это происходит поэтапно, чтобы избежать внезапных обратных изменений. А пока я рекомендую вручную проверить ваш код, чтобы увидеть, что `color.scale()`может работать лучше для вас.

### Настроить импортированные библиотеки

Сторонние или повторно используемые библиотеки часто поставляются с глобальными переменными конфигурации по умолчанию, которые вы можете переопределить. Раньше мы делали это с переменными перед импортом:

```
// _buttons.scss
$color: blue !default;

// old.scss
$color: red;
@import 'buttons';
```

Поскольку используемые модули больше не имеют доступа к локальным переменным, нам нужен новый способ установки этих значений по умолчанию. Мы можем сделать это, добавив карту конфигурации в `@use`:

```
@use 'buttons' with (
  $color: red,
  $style: 'flat',
);
```

Это похоже на `$with`аргумент в `load-css()`. но вместо того, чтобы использовать имена переменных в качестве ключей, мы используем сами переменные, начиная с `$`.

Мне нравится, насколько явно это делает настройку, но есть одно правило, которое несколько раз ставило меня в тупик: **модуль можно настроить только один раз, при первом использовании** . Порядок импорта всегда был важен для Sass, даже с `@import`. но эти проблемы всегда терпели неудачу молча. Теперь мы получаем явную ошибку, что и хорошо, и иногда удивительно. Обязательно `@use`настройте библиотеки в первую очередь в любом файле «точки входа» (центральный документ, который импортирует все частичные файлы), чтобы эти конфигурации компилировались раньше других `@use`библиотек.

(В настоящее время) невозможно «связывать» конфигурации вместе, оставляя их редактируемыми, но вы можете обернуть сконфигурированный модуль вместе с расширениями и передать его как новый модуль.

### Передавайте файлы с помощью @forward

Нам не всегда нужно использовать файл и обращаться к его членам. Иногда мы просто хотим передать его будущему импорту. Допустим, у нас есть несколько партиалов, связанных с формой, и мы хотим импортировать их все вместе как одно пространство имен. Мы можем сделать это с помощью `@forward`:

```
// forms/_index.scss
@forward 'input';
@forward 'textarea';
@forward 'select';
@forward 'buttons';
```

Члены пересылаемых файлов недоступны в текущем документе, и пространство имен не создается, но эти переменные, функции и

примеси будут доступны, когда другой файл `@use`или `@forward`вся коллекция будут доступны. Если пересылаемые части содержат фактический CSS, он также будет передаваться без создания вывода, пока пакет не будет использован. В этот момент все это будет рассматриваться как один модуль с одним пространством имен:

```
// styles.scss
@use 'forms'; // imports all of the forwarded members in
the `forms` namespace
```

**Примечание:** если вы попросите Sass импортировать каталог, он будет искать файл с именем `index`или `_index`)

По умолчанию все общедоступные члены будут пересылать модуль. Но мы можем быть более избирательными, добавляя предложения `show`or `hide`и называя конкретные члены для включения или исключения:

```
// forward only the 'input' border() mixin, and $border-
color variable
@forward 'input' show border, $border-color;

// forward all 'buttons' members *except* the gradient()
function
@forward 'buttons' hide gradient;
```

**Примечание:** когда функции и примеси имеют общее имя, они отображаются и скрываются вместе.

Чтобы прояснить источник или избежать конфликтов имен между пересылаемыми модулями, мы можем использовать `as`префикс членов партиала при пересылке:

```
// forms/_index.scss
// @forward "<url>" as <prefix>-*;
// assume both modules include a background() mixin
@forward 'input' as input-*;
```

```
@forward 'buttons' as btn-*;

// style.scss
@use 'forms';
@include forms.input-background();
@include forms.btn-background();
```

И, если нам нужно, мы можем всегда `@use`и `@forward`тот же модуль, добавив оба правила:

```
@forward 'forms';
@use 'forms';
```

Это особенно полезно, если вы хотите обернуть библиотеку конфигурацией или любыми дополнительными инструментами, прежде чем передавать ее другим своим файлам. Это может даже помочь упростить пути импорта:

```
// _tools.scss
// only use the library once, with configuration
@use 'accoutrement/sass/tools' with (
  $font-path: '../fonts/',
);
// forward the configured library with this partial
@forward 'accoutrement/sass/tools';

// add any extensions here...



// _anywhere-else.scss
// import the wrapped-and-extended library, already
configured
@use 'tools';
```

Оба `@use`и `@forward`должны быть объявлены в корне документа (не вложенные) и в начале файла. Только `@charset`и простые

определения переменных могут появляться перед командами импорта.

### Переходим к модулям

Чтобы протестировать новый синтаксис, я создал новую библиотеку Sass с открытым исходным кодом ( Cascading Color Systems ) и новый веб-сайт для своей группы — оба все еще находятся в разработке. Я хотел понимать модули и как библиотеку, и как автора веб-сайта. Давайте начнем с опыта «конечного пользователя» при написании стилей сайта с синтаксисом модуля…

### Поддержание и стиль письма

Пользоваться модулями на сайте было одно удовольствие. Новый синтаксис поддерживает архитектуру кода, которую я уже использую. Вся моя глобальная конфигурация и импорт инструментов находятся в одном каталоге (я называю его `config`) с индексным файлом, который пересылает все, что мне нужно:

```
// config/_index.scss
@forward 'tools';
@forward 'fonts';
@forward 'scale';
@forward 'colors';
```

По мере создания других аспектов сайта я могу импортировать эти инструменты и конфигурации туда, где они мне нужны:

```
// layout/_banner.scss
@use '../config';

.page-title {
  @include config.font-family('header');
}
```

Это работает даже с моими существующими библиотеками Sass, такими как Accoutrement и Herman , которые все еще используют старый `@import`синтаксис. Поскольку `@import`правило не будет заменено повсеместно в одночасье, Sass построил переходный период. Модули доступны уже сейчас, но `@import`не будут объявлены устаревшими еще год или два — и будут удалены из языка только через год после этого. Между тем, две системы будут работать вместе в любом направлении:

- Если мы `@import`файл, который содержит новый `@use`/`@forward`синтаксис, импортируются только общедоступные члены без пространства имен.
- Если мы `@use`или `@forward`файл содержит устаревший `@import`синтаксис, мы получаем доступ ко всем вложенным импортам как к единому пространству имен.

Это означает, что вы можете начать использовать новый синтаксис модуля прямо сейчас, не дожидаясь нового выпуска ваших любимых библиотек: и я могу потратить некоторое время на обновление всех моих библиотек!

### Инструмент миграции

Обновление не должно занять много времени, если мы используем инструмент миграции, созданный Дженнифер Такар. Его можно установить с помощью Node, Chocolatey или Homebrew:

```
npm install -g sass-migrator
choco install sass-migrator
brew install sass/sass/migrator
```

Это не одноразовый инструмент для перехода на модули. Теперь, когда Sass снова находится в активной разработке (см. ниже), инструмент миграции также будет получать регулярные обновления, чтобы помочь перенести каждую новую функцию. Рекомендуется

установить его глобально и сохранить для будущего использования.

Мигратор можно запустить из командной строки, и мы надеемся, что он будет добавлен в сторонние приложения, такие как CodeKit и Scout. Укажите на один файл Sass, например `style.scss`. и скажите, какие миграции применять. На данный момент существует только одна миграция под названием `module`:

```
# sass-migrator <migration> <entrypoint.scss...>
sass-migrator module style.scss
```

По умолчанию средство миграции будет обновлять только один файл, но в большинстве случаев мы хотим обновить основной файл *и все его зависимости* : любые частичные файлы, которые импортируются, пересылаются или используются. Мы можем сделать это, упомянув каждый файл отдельно или добавив флаг `--migrate-deps`:

```
sass-migrator --migrate-deps module style.scss
```

Для тестового прогона мы можем добавить `--dry-run --verbose`(или `-nv`для краткости) и посмотреть результаты без изменения каких-либо файлов. Существует ряд других параметров, которые мы можем использовать для настройки миграции — даже один, специально предназначенный для того, чтобы помочь авторам библиотек удалить старые пространства имен, созданные вручную, — но я не буду рассматривать их все здесь. Инструмент миграции полностью задокументирован на сайте Sass .

### Обновление опубликованных библиотек

Я столкнулся с несколькими проблемами на стороне библиотеки, в частности, пытаясь сделать пользовательские конфигурации доступными для нескольких файлов и работая с отсутствующими цепочками конфигураций. Ошибки в заказе может быть сложно отладить, но результаты стоят затраченных усилий, и я думаю, что

скоро мы увидим некоторые дополнительные исправления. Мне еще предстоит поэкспериментировать с инструментом миграции на сложных пакетах и, возможно, написать дополнительный пост для авторов библиотек.

Сейчас важно знать, что Sass защитит нас в течение переходного периода. Мало того, что импорт и модули могут работать вместе, мы также можем создавать файлы « только для импорта », чтобы обеспечить лучший опыт для устаревших пользователей, которые все еще `@import`используют наши библиотеки. В большинстве случаев это будет альтернативная версия основного файла пакета, и вы захотите, чтобы они располагались рядом: `<name>.scss`для пользователей модулей и `<name>.import.scss`для устаревших пользователей. Каждый раз, когда пользователь вызывает `@import <name>`, он загружает `.import`версию файла:

```
// load _forms.scss
@use 'forms';

// load _forms.input.scss
@import 'forms';
```

Это особенно полезно для добавления префиксов для немодульных пользователей:

```
// _forms.import.scss
// Forward the main module, while adding a prefix
@forward "forms" as forms-*;
```

### Обновление Sass

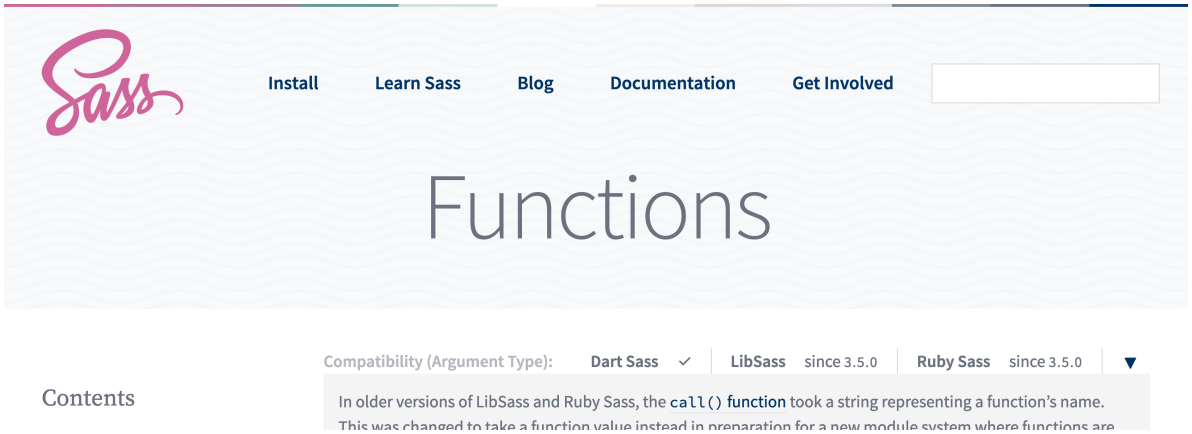
Возможно, вы помните, что несколько лет назад в Sass была заморожена функция, чтобы догнать различные реализации (LibSass, Node Sass, Dart Sass) и, в конечном итоге, отказаться от исходной реализации Ruby . Это замораживание закончилось в прошлом году с несколькими новыми функциями и активными обсуждениями и

разработками на GitHub, но без особой помпы. Если вы пропустили эти выпуски, вы можете найти их в блоге Sass :

- Импорт CSS и совместимость с CSS (Dart Sass v1.11)
- Аргументы содержимого и цветовые функции (Dart Sass v1.15)

Dart Sass теперь является канонической реализацией и, как правило, будет первым, кто реализует новые функции. Если вам нужна последняя версия, я рекомендую переключиться. Вы можете установить Dart Sass с Node, Chocolatey или Homebrew. Он также отлично работает с существующими шагами сборки gulp-sass .

Как и в CSS (начиная с CSS3), больше нет единого унифицированного номера версии для новых выпусков. Все реализации Sass работают с одной и той же спецификацией, но каждая из них имеет уникальный график выпуска и нумерацию, что отражено вместе с информацией о поддержке в прекрасной новой документации, разработанной Джиной .




Модули Sass доступны с **1 октября 2019 года** в **версии Dart Sass 1.23.0** .