# [Sass (3.4.21) 中文文档](http://www.css88.com/doc/sass/)

Sass 是一个 CSS 的扩展，它在 CSS 语法的基础上，允许您使用[变量 (variables)](http://www.css88.com/doc/sass/#variables_), [嵌套规则 (nested rules)](http://www.css88.com/doc/sass/#nested_rules), [混合 (mixins)](http://www.css88.com/doc/sass/#mixins), [导入 (inline imports)](http://www.css88.com/doc/sass/#import)等功能，令 CSS 更加强大与优雅。使用 Sass 以及 [Compass 样式库](http://compass-style.org/) 有助于更好地组织管理样式文件，以及更高效地开发项目。

**特色 (Features)**

* 完全兼容 CSS3
* 在 CSS 语言的基础上增加变量(variables)、嵌套 (nesting)、混合 (mixins) 等功能
* 通过函数进行颜色值与属性值的运算
* 提供 [控制指令](http://www.css88.com/doc/sass/#control_directives__expressions)等高级功能
* 自定义输出格式

**语法 (Syntax)**

Sass 有两种语法格式。首先是 SCSS (Sassy CSS) ,也是本参考资料示例所使用的格式 , 这种格式仅在 CSS3 语法的基础上进行扩展，这意味着每个CSS样式表是一个同等的SCSS文件。此外，SCSS 也支持大多数 CSS hacks 写法 以及浏览器专属前缀语法 (vendor-specific syntax)，例如，[IE 古老的 filter 语法](http://msdn.microsoft.com/en-us/library/ms530752.aspx)。 这种语法的样式表文件需要以 .scss 作为拓展名。

另一种，也是最早的语法，被称为缩进语法 (Indented Sass)，或者通常说的 "Sass"，它提供了一种更加简介的方式来书写CSS。它使用缩进而不是花括号来表示选择器的嵌套，用换行而不是分号来分隔属性，一些人认为这样做比 SCSS 更容易阅读，书写也更快速。 缩排语法具有 Sass 的所有特色功能， 虽然有些语法上稍有差异； 具体差异在[缩进语法参考](http://sass-lang.com/documentation/file.INDENTED_SYNTAX.html)中都有描述。 使用此种语法的样式表文件需要以 .sass 作为扩展名。

任何一种语法的文件可以直接 [import(导入)](http://www.css88.com/doc/sass/#import) 到另一种语法的文件中使用，只要使用 sass-convert 命令行工具，就可以将一种语法转换为另一种语法：

# Convert Sass to SCSS

$ sass-convert style.sass style.scss

# Convert SCSS to Sass

$ sass-convert style.scss style.ss

请注意，此命令 *不会* 生成CSS文件。要想生成CSS文件，请使用其他地方描述的 sass 命令。

**使用 Sass (Using Sass)**

Sass 有三种使用方式：作为命令行工具，作为独立的 Ruby 模块 (Ruby module)，或者作为 Rack-enabled 框架的插件，包括 Ruby on Rails 与 Merb。无论使用哪种方式都需要首先安装 Sass gem ：

gem install sass

如果你使用的是Windows ，你可能首先需要[安装Ruby](http://rubyinstaller.org/download.html) 。

如果要在命令行中运行 Sass ,只要使用

sass input.scss output.css

你还可以使用Sass命令来监视某个Sass文件的改动，并自动编译来更新 CSS ：

sass --watch input.scss:output.css

如果你的目录里有很多 Sass 文件，你也可以使用Sass命令来监视整个目录：

sass --watch app/sass:public/stylesheets

使用 sass --help 可以列出完整的帮助文档。

在 Ruby 中使用 Sass 也非常容易，Sass gem 安装完毕后，用它运行 require "sass"， 然后按照下面的方法使用 [Sass::Engine](http://sass-lang.com/documentation/Sass/Engine.html)：

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

engine.render #=> "#main { background-color: #0000ff; }\n"

**Rack/Rails/Merb 插件（Plugin）**

在 Rails 3 之前的版本中启用 Sass，需要在 environment.rb 文件中添加一行代码：

config.gem "sass"

对于 Rails 3，则是把这一行加到 Gemfile 中：

gem "sass"

要在 Merb 中启用 Sass，需要在 config/dependencies.rb 文件中添加一行代码：

dependency "merb-haml"

在 Rack 应用中启用 Sass，需要在 config.ru 文件中添加以下代码：

require 'sass/plugin/rack'

use Sass::Plugin::Rack

Sass 样式表跟视图（views）的工作方式不同。 它不包含任何动态内容， 因此只需要在 Sass 文件更新时生成 CSS 即可。 默认情况下，.sass 和 .scss 文件是放在 public/stylesheets/sass 目录下的（这可以通过 [:template\_location](http://www.css88.com/doc/sass/#template_location-option) 选项进行配置）。 然后，在需要的时候，它们会被编译成相应的 CSS 文件并被放到 public/stylesheets 目录下。 例如，public/stylesheets/sass/main.scss 文件将会被编译为 public/stylesheets/main.css 文件。

**缓存 (Caching)**

默认情况下，Sass 会自动缓存编译后的模板（template）与 [partials](http://www.css88.com/doc/sass/#partials)，这样做能够显著提升重新编译的速度，在处理 Sass 模板被切割为多个文件并通过 [@import](http://www.css88.com/doc/sass/#import) 导入，形成一个大文件时效果尤其显著。

如果不使用框架的情况下，Sass 将会把缓存的模板放入 .sass-cache 目录。 在 Rails 和 Merb 中，缓存的模板将被放到tmp/sass-cache 目录。 此目录可以通过[:cache\_location](http://www.css88.com/doc/sass/#cache_location-option) 选项进行自定义。 如果你不希望 Sass 启用缓存功能， 可以将 [:cache](http://www.css88.com/doc/sass/#cache-option) 选项设置为 false。

**配置选项 (Options)**

选项可以通过设置的[Sass::Plugin#options](http://sass-lang.com/documentation/Sass/Plugin/Configuration.html#options-instance_method) hash，具体设置在Rails中的environment.rb或者Rack中的config.ru的文件中：

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

或者，如果你使用Merb，那么可以在init.rb文件中设置Merb::Plugin.config[:sass] hash ：

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

或者通过传递一个选项 (options) hash 给[Sass::Engine#initialize](http://sass-lang.com/documentation/Sass/Engine.html#initialize-instance_method)，   
所有相关的选项也可通过标记在sass 和 scss命令行可执行文件中使用。可用选项有：

| **选项** | **描述** |
| --- | --- |
| :style | 设置输出CSS的代码风格，可以查看[输出的代码风格](http://www.css88.com/doc/sass/#output_style)。 |
| :syntax | 输入文件的语法，:sass 表示缩进语法，:scss表示CSS扩展语法。只有在你自  己构造[Sass::Engine](http://sass-lang.com/documentation/Sass/Engine.html) 实例的情况下有用；当你使用 [Sass::Plugin](http://sass-lang.com/documentation/Sass/Plugin.html)时，它会自  动设置正确的值。默认设置为 :sass 。 |
| :property\_syntax | 强制缩进语法文档使用一个属性语法。如果不使用正确的语法，将抛出一个错  误。:new值表示强制在属性名后面使用一个冒号。例如：color: #0f3或者   width: $main\_width 。 :old值表示强制在属性名前面使用一个冒号。  例如：:color #0f3 或者 :width $main\_width。默认请客下，两种语法都是有  效的。该选项对于SCSS（.scss）文档是无效的。 |
| :cache | 解析 Sass 时是否应该缓存，允许更快的编译速度。默认设置为 true 。 |
| :read\_cache | 如果设置了这个选项，而没有设置:cache选项，那么缓存存在就只读 Sass  缓存，如果没有没有缓存，那就不会编译。 |
| :cache\_store | 如果该选项设置为 [Sass::CacheStores::Base](http://sass-lang.com/documentation/Sass/CacheStores/Base.html) 的子类的实例，该缓存存储将  被用于存储和检索缓存编译结果。默认设置为  [Sass::CacheStores::Filesystem](http://sass-lang.com/documentation/Sass/CacheStores/Filesystem.html)，初始化使用[:cache\_location 选项](http://www.css88.com/doc/sass/#cache_location-option)。 |
| :never\_update | CSS文件永远不应该被更新，即使是模板（template）文件改变。将其设置  为true可能会带来小的性能提升。它总是默认为false。该选项只有在  Rack，Ruby on Rails，或Merb中有意义。 |
| :always\_update | CSS文件总是应该进行更新，即使是在每一个控制器被访问时，而不是只当  模板被修改时更新。默认为false。该选项只有在Rack，Ruby on Rails，或  Merb中有意义。 |
| :always\_check | Sass模板总是应该被检查是否更新，即使是在每一个控制器被访问时，而不  是只在服务启动时。如果一个Sass模板（template）被更新，它会被重新  编译并且覆盖相应的CSS文件。在生产模式中默认为false，否则true。该选  项只有在Rack，Ruby on Rails，或Merb中有意义。 |
| :poll | 如果为true，始终使用[Sass::Plugin::Compiler#watch](http://sass-lang.com/documentation/Sass/Plugin/Compiler.html#watch-instance_method)后端轮询而不是本机  文件系统的后端。 |
| :full\_exception | Sass代码中错误是否应该在生成的CSS文件中提供详细的说明。如果设置为  true，这个错误将显示在CSS文件的注释中 和页面顶部（支持的浏览器）  ，错误内容包括行号和源代码片段。 否则，异常将在Ruby代码中提醒。  在生产模式中默认为false，否则true。 |
| :template\_location | 一个路径，应用根目录中Sass模板（template）的目录。如果散列，  :css\_location 将被忽略，并且这个选项指定输入和输出目录之间的映射。  也可以给定二元列表，代替散列（hash）。默认为css\_location + "/sass"。  该选项只有在Rack，Ruby on Rails，或Merb中有意义。请注意，如果指定  了多个模板位置，它们全部都放置在导入路径中，允许你在它们之间进行导入。 **需要注意的是，由于它可以采用许多可能的格式，这个选项应该只直接设置，**  **不应该被访问或修改。使用**[**Sass::Plugin#template\_location\_array**](http://sass-lang.com/documentation/Sass/Plugin/Configuration.html#template_location_array-instance_method)**，**  [**Sass::Plugin#add\_template\_location**](http://sass-lang.com/documentation/Sass/Plugin/Configuration.html#add_template_location-instance_method)**，**  [**Sass::Plugin#remove\_template\_location**](http://sass-lang.com/documentation/Sass/Plugin/Configuration.html#remove_template_location-instance_method)**方法来代替。** |
| :css\_location | CSS文件输出的路径，当:template\_location选项为一个散列（hash）时，  这个选项将被忽略。默认设置为"./public/stylesheets"。该选项只有在Rack，  Ruby on Rails，或Merb中有意义。 |
| :cache\_location | 其中，高速缓存sassc 文件应写入的路径。在Rails和Merb中默认  为"./tmp/sass-cache"，否则默认为"./.sass-cache" 。如果设置  了[:cache\_store 选项](http://www.css88.com/doc/sass/#cache_location-option)，这个将被忽略。 |
| :unix\_newlines | 如果为true，写入文件时使用Unix风格的换行符。只有在Windows上，  并且只有当Sass被写入文件时有意义（在 Rack, Rails, 或 Merb中，直接使  用[Sass::Plugin](http://sass-lang.com/documentation/Sass/Plugin.html)时，或者使用命令行可执行文件时）。 |
| :filename | 被渲染文件的文件名。这完全是用于报告错误，使用Rack, Rails, or Merb时  自动设置。 |
| :line | Sass模板（template）第一行的行号。用于报告错误的行号。如果Sass模  板（template）嵌入一个Ruby文件中，这个设置是很有用的。 |
| :load\_paths | 一个数组，包含文件系统 或 通过[@import](http://www.css88.com/doc/sass/#import)指令导入的 Sass模板（  template）路径。他们可能是字符串， Pathname （路径名）对象或者是  [Sass::Importers::Base](http://sass-lang.com/documentation/Sass/Importers/Base.html)的子类。该选项默认为工作目录，并且在Rack, Rails,  或 Merb中，该选项无论如何值都是:template\_location。加载路径也可以由  [Sass.load\_paths](http://sass-lang.com/documentation/Sass.html#load_paths-class_method)和 SASS\_PATH 环境变量通知。 |
| :filesystem\_importer | 一个[Sass::Importers::Base](http://sass-lang.com/documentation/Sass/Importers/Base.html)的子类，用来处理普通字符串的加载路径。这应  该从文件系统导入文件。这应该是一个通过构造函数带一个字符串参数（加载  路径），继承自[Sass::Importers::Base](http://sass-lang.com/documentation/Sass/Importers/Base.html)的Class对象。默认为  [Sass::Importers::Filesystem](http://sass-lang.com/documentation/Sass/Importers/Filesystem.html)。 |
| :sourcemap | 控制如何生产sourcemap。这些sourcemaps告诉浏览器如何找到Sass样  式，从而生成每一个CSS样式。该选项有三个有效值：:auto在可能的情况  下使用相对URI，假设在你使用的任何服务器上提供的源样式，那么它们的  相对位置将和本地文件系统是相同的。如果一个相对URI不可用，那么将  被"file:"替换。:file总是使用"file:" URI，这将在本地工作，但不能被部署到  一个远程服务器。 :inline包含sourcemap中完整的源文本，这是最方便的  ，但是可能生产非常大的sourcemap文件。 最后，:none 会导致总是不会  生成sourcemap文件。 |
| :line\_numbers | 当设置为true的时候，定义的选择器的行号和文件名 将被作为注释注入到  编译的CSS中。这对调试来说是有用的，特别是使用[@import](http://www.css88.com/doc/sass/#import)和[@mixin](http://www.css88.com/doc/sass/#mixin)的  时候。这个选项有个别名叫做:line\_comments。当使用 :compressed输出样  式或使用:debug\_info/:trace\_selectors选项时这个选项将自动禁用。 |
| :trace\_selectors | 当设置为true的时候，将在每个选择器之前注入[@import](http://www.css88.com/doc/sass/#import)和[@mixin](http://www.css88.com/doc/sass/#mixin)的完整  轨迹。在浏览器中调试通过[@import](http://www.css88.com/doc/sass/#import)和[@mixin](http://www.css88.com/doc/sass/#mixin)包含进来的样式表时是很有  用的。此选项将取代 :line\_comments 选项，并且被:debug\_info选项取代。  当使用 :compressed输出样式时，这个选项将自动禁用。 |
| :debug\_info | 当设置为true的时候，定义的选择器的行号和文件名 将被注入到编译后的  CSS中,可以被浏览器所识别。用于[FireSass Firebug 扩展](https://addons.mozilla.org/en-US/firefox/addon/103988)，以显示Sass文  件名和行号。当使用 :compressed输出样式时，这个选项将自动禁用。 |
| :custom | 这个选项可用于单个应用程序设置以使数据可用于[定制Sass功能](http://sass-lang.com/documentation/Sass/Script/Functions.html)。 |
| :quiet | 当设置为true的时候，导致禁用警告信息。 |

**语法选择（Syntax Selection）**

Sass命令行工具将使用文件扩展名以确定你使用的是哪种语法，但并不总是一个文件名。sass命令行程序默认为缩进语法，但如果输入应该被解析为SCSS语法，你可以传递--scss选项给她。此外，你可以使用scss命令行程序，它和sass程序完全一样，但是他的默认语法为SCSS。

**编码格式 (Encodings)**

在 Ruby 1.9 及以上环境中运行 Sass 时，Sass 对文件的编码格式比较敏感，首先会根据 [CSS spec](http://www.w3.org/TR/2013/WD-css-syntax-3-20130919/#determine-the-fallback-encoding)判断样式文件的编码格式， 如果失败则检测 Ruby 字符串编码。也就是说，Sass 首先检查 Unicode 字节顺序标记，然后是 @charset 声明，最后是 Ruby 字符串编码，假如都没有检测到，默认使用 UTF-8 编码。

要明确指定样式表的编码，与 CSS 相同，使用@charset声明。在样式文件的起始位置（前面没有任何空白与注释）插入 @charset "encoding-name";， Sass 将会按照给定的编码格式编译文件。注意，无论你使用哪种编码，它必须可以转换为 Unicode 字符集。

默认情况下，Sass 总会以UTF-8编码输出 CSS 文件。当且仅当输出文件包含非ASCII字符时，才会在输出文件中添加 @charset 声明，在压缩模式中，而在压缩模式下 (compressed mode) 使用 UTF-8字节顺序标记代替 @charset 声明语句。

**CSS扩展 (CSS Extensions)**

**嵌套规则 (Nested Rules)**

Sass 允许将一个 CSS 样式嵌套进另一个样式中，内层样式仅适用于外层样式的选择器范围内（愚人码头注：可以理解为层级选择器），例如：

#main p {

color: #00ff00;

width: 97%;

.redbox {

background-color: #ff0000;

color: #000000;

}

}

编译为：

#main p {

color: #00ff00;

width: 97%; }

#main p .redbox {

background-color: #ff0000;

color: #000000; }

这有助于避免父选择器重复，相对于复杂的CSS布局中多层嵌套的选择器 要简单得多。 例如：

#main {

width: 97%;

p, div {

font-size: 2em;

a { font-weight: bold; }

}

pre { font-size: 3em; }

}

编译为：

#main {

width: 97%; }

#main p, #main div {

font-size: 2em; }

#main p a, #main div a {

font-weight: bold; }

#main pre {

font-size: 3em; }

**引用父选择器:& (Referencing Parent Selectors: &)**

有些时候需要直接使用嵌套外层的父选择器，这个就很有用了，例如，你可能喜欢给选择器指定 hover样式，或者当body元素具有某个样式时，在这些情况下，你可以 & 字符来明确地表示插入指定父选择器。 例如：

a {

font-weight: bold;

text-decoration: none;

&:hover { text-decoration: underline; }

body.firefox & { font-weight: normal; }

}

编译为：

a {

font-weight: bold;

text-decoration: none; }

a:hover {

text-decoration: underline; }

body.firefox a {

font-weight: normal; }

&将替换为呈现在CSS文件中的父选择器。这意味着，如果你有一个多层嵌套的规则，父选择器将在被&替换之前完全分解。 例如：

#main {

color: black;

a {

font-weight: bold;

&:hover { color: red; }

}

}

编译为：

#main {

color: black; }

#main a {

font-weight: bold; }

#main a:hover {

color: red; }

& 必须出现在的选择器的开头位置（愚人码头注：也就是作为选择器的第一个字符），但可以跟随后缀，将被添加到父选择的后面。 例如：

#main {

color: black;

&-sidebar { border: 1px solid; }

}

编译为：

#main {

color: black; }

#main-sidebar {

border: 1px solid; }

父选择器 & 被作为一个后缀的时候，Sass 将抛出一个错误。

**嵌套属性 (Nested Properties)**

CSS中有一些属性遵循相同的“命名空间”；比如，font-family, font-size, 和 font-weight都在font命名空间中。在CSS中，如果你想在同一个命名空间中设置一串属性，你必须每次都输出来。Sass为此提供了一个快捷方式：只需要输入一次命名空间，然后在其内部嵌套子属性。例如：

.funky {

font: {

family: fantasy;

size: 30em;

weight: bold;

}

}

编译为：

.funky {

font-family: fantasy;

font-size: 30em;

font-weight: bold; }

命名空间也可以有自己的属性值。例如：

.funky {

font: 20px/24px fantasy {

weight: bold;

}

}

编译为：

.funky {

font: 20px/24px fantasy;

font-weight: bold;

}

**占位符选择器:**%foo**(Placeholder Selectors:**%foo**)**

Sass 支持一种特殊类型的选择器,叫做"占位符选择器" (placeholder selector)。这些看起来像 class 和 id 选择器，除了# 或.用%替换。他们需要在[@extend 指令](http://www.css88.com/doc/sass/#extend)中使用;有关详细信息，请参阅[@extend-Only Selectors](http://www.css88.com/doc/sass/#placeholders)。

当他们单独使用的时候，即没有使用@extend的，使用占位符选择器的规则集将不会被渲染为CSS。

**注释:**/\* \*/**和**//**（Comments:**/\* \*/**and**//**）**

Sass 支持标准的CSS多行注释以/\* \*/以及单行注释 //。在尽可能的情况下，多行注释会被保留在输出的CSS中，而单行注释会被删除。 例如：

/\* This comment is

\* several lines long.

\* since it uses the CSS comment syntax,

\* it will appear in the CSS output. \*/

body { color: black; }

// These comments are only one line long each.

// They won't appear in the CSS output,

// since they use the single-line comment syntax.

a { color: green; }

编译为：

/\* This comment is

\* several lines long.

\* since it uses the CSS comment syntax,

\* it will appear in the CSS output. \*/

body {

color: black; }

a {

color: green; }

如果多行注释的第一个字母是 !，那么注释总是会被保留到输出的CSS中，即使在压缩输出模式下。这可用于在你生成的CSS中添加版权声明。

使用插值语句 (interpolation) ，可以将变量值输出到多行注释中，例如：

$version: "1.2.3";

/\* This CSS is generated by My Snazzy Framework version #{$version}. \*/

编译为：

/\* This CSS is generated by My Snazzy Framework version 1.2.3. \*/

**SassScript**

除了普通的CSS属性的语法，Sass 支持一些扩展，名为SassScript。SassScript允许属性使用变量，算术和额外功能。SassScript可以在任何属性值被使用。

SassScript也可以用来生成选择器和属性名称，当编写[mixins](http://www.css88.com/doc/sass/#mixins)时非常有用。这是通过 [interpolation（插值）](http://www.css88.com/doc/sass/#interpolation_) 完成。

**交互式 shell（Interactive Shell）**

Interactive Shell 可以在命令行中测试 SassScript 的功能。在命令行中输入 sass -i，然后输入想要测试的 SassScript 查看输出结果：

您可以使用交互式shell（Interactive Shell）轻松地尝试 SassScript。   
要箱运行启动 shell ，只要使用-i选项的 sass 命令行（愚人码头注：在命令行中输入 sass -i）。   
在提示符下，输入任何合法的 SassScript表达式，由它他评估并打印出您的结果：

$ sass -i

>> "Hello, Sassy World!"

"Hello, Sassy World!"

>> 1px + 1px + 1px

3px

>> #777 + #777

#eeeeee

>> #777 + #888

white

**变量:**$**（Variables:**$**）**

使用SassScript最直截了当的方法是使用变量。变量以美元符号开始，赋值像设置CSS属性那样：

$width: 5em;

你可以在属性中引用他们:

#main {

width: $width;

}

变量仅在它定义的选择器嵌套层级的范围内可用（愚人码头注：可以理解为块级作用域）。不在任何嵌套选择器内定义的变量则在可任何地方使用（愚人码头注：可以理解为全局变量）。定义变量的时候可以后面带上!global标志，在这种情况下，变量在任何地方可见（愚人码头注：可以理解为全局变量）。例如：

#main {

$width: 5em !global;

width: $width;

}

#sidebar {

width: $width;

}

编译为：

#main {

width: 5em;

}

#sidebar {

width: 5em;

}

由于历史原因，变量名（以及其他所有Sass标识符）可以互换连字符（愚人码头注：-）和下划线（愚人码头注：\_）。例如，如果你定义了一个名为 $main-width，您可以使用 $main\_width访问它，反之亦然。

**数据类型 (Data Types)**

SassScript 支持 7 种主要的数据类型：

* 数字 (例如： 1.2, 13, 10px)
* 文本字符串，带引号字符串和不带引号字符串(例如："foo", 'bar', baz)
* 颜色 (例如：blue, #04a3f9, rgba(255, 0, 0, 0.5))
* 布尔值 (例如： true, false)
* 空值 (例如： null)
* 值列表 (list)，用空格或逗号分隔 (例如： 1.5em 1em 0 2em, Helvetica, Arial, sans-serif)
* maps ，从一个值映射到另一个 (例如： (key1: value1, key2: value2))

SassScript也支持其他所有类型的CSS属性值，比如 Unicode 字符集，或 !important 声明。然而，不会对这些类型的属性值做特殊处理，一律视为不带引号的字符串。

**字符串（Strings）**

CSS指定两种字符串类型：带引号的字符串（愚人码头注：包括双引号和单引号），如"Lucida Grande" 或者 'http://sass-lang.com'，还有不带引号的字符串，如sans-serif 或者 bold。SassScript 识别这两种类型，并且一般来说，在编译输出的CSS文件中不会改变Sass文档中使用的字符串类型。

有一个例外，当使用 [#{} interpolation](http://www.css88.com/doc/sass/#interpolation_)时，带引号的字符串将被编译为不带引号的字符串，这样主要是为了便于使用，比如[mixins](http://www.css88.com/doc/sass/#mixins)中的选择器名称。例如：

@mixin firefox-message($selector) {

body.firefox #{$selector}:before {

content: "Hi, Firefox users!";

}

}

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

编译为：

body.firefox .header:before {

content: "Hi, Firefox users!"; }

**列表（Lists）**

列表(lists) 是指 Sass 如何表示在CSS声明的，类似margin: 10px 15px 0 0 或 font-face: Helvetica, Arial, sans-serif这样的值，列表只是一串其他值，无论是用空格还是用逗号隔开。事实上，独立的值也被视为列表：只包含一个值的列表。

列表本身没有太多的功能，但 Sass list functions 赋予了数组更多新功能：nth 函数可以直接访问数组中的某一项；join 函数可以将多个数组连接在一起；append 函数可以在数组中添加新值；而 @each 指令能够遍历数组中的每一项。

列表本身没有太多的功能，但是 SassScript list functions 使它们非常有用。[nth 函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#nth-instance_method)可以直接访问列表中的某一项；[join 函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#join-instance_method)可以将多个列表拼接在一起；[append 函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#append-instance_method)可以将某项添加到列表中；[@each 指令](http://www.css88.com/doc/sass/#each-directive)可以将添加样式到列表中的每一项。

除了包含简单的值，列表可包含其他列表。例如，1px 2px, 5px 6px包含1px 2px列表和5px 6px列表两个项。如果内外两层列表使用相同的分隔符号，你需要使用括号将内层列表括起来，以明确内层类别的开始和结束位置。   
例如，(1px 2px) (5px 6px) 同样是包含1px 2px列表和5px 6px列表两个项的列表。不同的是，该列表层外用空格分隔，之前列表外层是用逗号分隔。

当列表被编译为 CSS 时，Sass 不会添加任何圆括号，因为CSS不能识别他们。这意味着， (1px 2px) (5px 6px) 和1px 2px 5px 6px 在编译后的 CSS 文件中看起来是完全一样的。然而，它们在 Sass 中却是不同的：第一个是含两个列表的列表，而第二个是含有四个成员的列表。

用 () 表示不包含任何值的空数组（在 Sass 3.3 版之后也视为空的 map）。空数组不可以直接编译成 CSS，比如编译 font-family: () Sass 将会报错。如果数组中包含空数组或空值，编译时将被清除，比如 1px 2px () 3px 或 1px 2px null 3px。

列表也可以没有任何项。这些列表可以用 () 表示（也是一个空的 [map](http://www.css88.com/doc/sass/#maps)）。   
它们不能直接输出到CSS;如果你试图这样做，例如font-family: ()，Sass 将会报错。如果列表中包含空列表或空值，比如 1px 2px () 3px 或者 1px 2px null 3px，在包含列表编译成CSS前，空列表和空值将被删除。

逗号分隔的列表可以保留结尾的逗号。这是特别有用，因为它可以表示一个 单个元素的列表。   
例如，(1,)表示为只包含1的列表，而(1 2 3,)这个表示包含一个列表，这个列表又包含以空格分隔的1,2, 和 3的列表。

**Maps**

Maps代表一个键和值对集合，其中键用于查找值。他们可以很容易地将值收集到命名组中，并且可以动态地访问这些组。在CSS中你找不到和他们类似的值，虽然他们的语法类似于媒体查询表达式：

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

和列表（Lists）不同，Maps必须始终使用括号括起来，并且必须用逗号分隔。Maps中的键和值可以是任意的SassScript对象。一个Maps可能只有一个值与给定的键关联（尽管该值可以是一个列表）。一个给定的值可能与许多键关联。

和列表（Lists）类似，Maps的主要操作使用的是 [SassScript 函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#map-functions)。[map-get函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#map_get-instance_method)用于查找map中的值，[map-merge函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#map_merge-instance_method)用于添加值到map中的值， [@each 指令](http://www.css88.com/doc/sass/#each-multi-assign)可以用来为 map 中的每个键值对添加样式。map中键值对的顺序和map创建时始终相同。

Maps还可以用于任何列表（Lists）能做的事情。当用于一个列表函数时，map被视为键值对列表。例如，(key1: value1, key2: value2)被用于列表函数时，将被视为嵌套列表key1 value1, key2 value2。列表不能被视为maps，不过，空列表除外。 ()表示一个键/值对都没有的map,也可以被视为一个没有元素的列表。

需要注意的是 map 的建（keys）可以是任何 Sass 数据类型（甚至是另一个map），并且声明map的语法允许是任意的SassScript表达式，这个表达式将被评估为一个值以确定建（keys）。

Maps不能转换为纯CSS。作为变量的值或参数传递给CSS函数将会导致错误。使用inspect($value)函数以产生输出字符串，这对于调试 maps 非常有用。

**颜色（Colors）**

任何CSS颜色表达式返回SassScript颜色值。这其中包括了[大量的命名的颜色](https://github.com/nex3/sass/blob/stable/lib/sass/script/value/color.rb#L28-L180)，这些名字字符串不区别带不带引号。

在压缩输出模式，Sass 将输出CSS简短的颜色表示法。例如，在压缩模式下 #FF0000 将输出为red，但是blanchedalmond将输出为 #FFEBCD。

一个用户遇到的常见问题是在其它输出模式中Sass喜欢输出与命名的颜色相同的格式，当压缩的时候，插值到选择器的颜色变得无效语法。为了避免这种情况，如果他们是为了在选择施工中使用，总是给命名的颜色。

**运算 (Operations)**

所有数据类型的支持相等运算(== 和 !=)。此外，每种类型都有其自己特殊的运算方式。

**数字运算 (Number Operations)**

SassScript 支持对数字标准的算术运算（加法+，减法 - ，乘法\*，除法/和取模%）。Sass 数学函数在算术运算期间会保留单位。这意味着，就像在现实生活中，你不能用不相同的单位数字进行算术运算（比如数字后面添加了px和em单位），还有两个单位相同的数字相乘将产生单位平方(10px \* 10px == 100px \* px)。**要知道**，px \* px是无效的CSS单位，Sass会抛出一个错误，因为你试图在CSS中使用无效的单位。

数字支持关系运算符(<, >, <=, >=)，并且所有类型支持相等运算符(==, !=)。

**除法和**/**（Division and**/**）**

CSS允许 / 出现在属性值之间作为分隔数字的方式(愚人码头注：例如font属性，p.ex2{font:italic bold 12px/20px arial,sans-serif;})。由于SassScript是CSS属性语法的扩展，所以它必须支持这一点，同时还允许 / 用于除法。这意味着，在默认情况下，在SassScript中如果两个数字由 / 分隔，在返回的CSS中将以同样的方式出现。

但是，这里有将/解析为除法三种情况。这些涵盖了绝大多数当做除法的案例。 他们是：

1. 如果该值，或值的任何部分，存储在一个变量中或通过函数返回。
2. 如果该值是由括号括起来的，除非这些括号是在一个列表（list）外部，并且值是括号内部。
3. 如果该值被用作另一个算术表达式的一部分。

例如:

p {

font: 10px/8px; // 原生的CSS，不作为除法

$width: 1000px;

width: $width/2; // 使用了变量, 作为除法

width: round(1.5)/2; // 使用了函数, 作为除法

height: (500px/2); // 使用了括号, 作为除法

margin-left: 5px + 8px/2px; // 使用了 +, 作为除法

font: (italic bold 10px/8px); // 在一个列表（list）中，括号可以被忽略。

}

编译为：

p {

font: 10px/8px;

width: 500px;

height: 250px;

margin-left: 9px; }

如果你想纯CSS 的/ 和变量一起使用（愚人码头注：即/不作为除法使用），你可以使用#{}插入他们。例如：

p {

$font-size: 12px;

$line-height: 30px;

font: #{$font-size}/#{$line-height};

}

编译为：

p {

font: 12px/30px; }

**减法，负数，和**-**(Subtraction, Negative Numbers, and**-**)**

在CSS和在Sass中 - 有许多不同的意义。它可以是一个减法运算符（比如在5px - 3px中），也可以表示一个负数（比如在-3px中），还可以是一个一元负运算符（比如在-$var中），或是标识符的一部分（比如在font-weight中）。大多数时候，我们可以很容易的分辨- 到底代表什么，但也有一些棘手的请客。以下作为一般规则，你是最安全的使用-：

* 减法的时候，你总是在 - 两侧保留空格。
* 当表示一个负数或一元负运算时候，在-前面包含一个空格，后面不加空格。
* 如果在一个空格隔开的list（列表）中，你可以将一元负运算使用括号括起来，比如在10px (-$var)中。

- 的不同含义的优先顺序如下：

1. - 作为标识符的一部分。这意味着a-1是一个不带引号的字符串，其值为"a-1"。唯一的例外是单位;Sass 通常允许任何有效的标识符被用作一个标识符，但标识符不可能以数字或连字符开始。这意味着，5px-3px和5px - 3px是相同。
2. - 在不带空格两个数字之间。这表明是减法，所以1-2和1 - 2 是相同的。
3. 字面数字以 - 开头。这表明是一个负数，所以 1 -2是一个含有1和-2的 list（列表）。
4. - 两个数字之间，不论是否带空格。这表明是减法，所以 1 -$var 和 1 - $var 是相同的。
5. - 在值之前。这表明是一元负运算符;该操作需要一个数字，并返回其负值。

**颜色运算 (Color Operations)**

所有算术运算都支持的颜色值，颜色值的运算是分段进行计算的，也就是，依次计算红（red），绿（green），以及蓝（blue）的成分值。例如：

p {

color: #010203 + #040506;

}

计算 01 + 04 = 05, 02 + 05 = 07, 和 03 + 06 = 09，并且编译为：

p {

color: #050709; }

通常[color functions(颜色函数)](http://sass-lang.com/documentation/Sass/Script/Functions.html)比尝试使用颜色运算更加有用，以达到同样的效果。

数字和颜色值之间的算术运算也是分段。例如：

p {

color: #010203 \* 2;

}

计算 01 \* 2 = 02, 02 \* 2 = 04, 和 03 \* 2 = 06，并且编译为：

p {

color: #020406; }

需要注意的是，包含alpha通道（那些由[rgba](http://sass-lang.com/documentation/Sass/Script/Functions.html#rgba-instance_method)或[hsla](http://sass-lang.com/documentation/Sass/Script/Functions.html#hsla-instance_method)函数创建的）的颜色必须具有相同的alpha值，才能进行颜色运算。这样算术不会影响alpha值。例如：

p {

color: rgba(255, 0, 0, 0.75) + rgba(0, 255, 0, 0.75);

}

编译为：

p {

color: rgba(255, 255, 0, 0.75); }

颜色的alpha通道可以使用[opacify](http://sass-lang.com/documentation/Sass/Script/Functions.html#opacify-instance_method)和[transparentize](http://sass-lang.com/documentation/Sass/Script/Functions.html#transparentize-instance_method)函数进行调整。例如：

$translucent-red: rgba(255, 0, 0, 0.5);

p {

color: opacify($translucent-red, 0.3);

background-color: transparentize($translucent-red, 0.25);

}

编译为：

p {

color: rgba(255, 0, 0, 0.8);

background-color: rgba(255, 0, 0, 0.25); }

IE浏览器的滤镜（filters）要求所有的颜色包括 alpha 层，而且格式必须是固定的 #AABBCCDD ，使用 [ie\_hex\_str](http://sass-lang.com/documentation/Sass/Script/Functions.html#ie_hex_str-instance_method) 函数可以轻松的将颜色转化为 IE 滤镜所要求的格式。例如：

$translucent-red: rgba(255, 0, 0, 0.5);

$green: #00ff00;

div {

filter: progid:DXImageTransform.Microsoft.gradient(enabled='false', startColorstr='#{ie-hex-str($green)}', endColorstr='#{ie-hex-str($translucent-red)}');

}

编译为：

div {

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

}

**字符串运算 (String Operations)**

+ 运算可用于连接字符串:

p {

cursor: e + -resize;

}

编译为：

p {

cursor: e-resize; }

请注意，如果带引号的字符串被添加到不带引号的字符串中（也就是说，带引号的字符串在 + 的左侧），   
那么返回的结果是带引号的字符串。同样，如果一个不带引号的字符串添加到带引号的字符串中（不带引号的字符串在 + 的左侧）那么返回的结果是一个不带引号的字符串。 例如：

p:before {

content: "Foo " + Bar;

font-family: sans- + "serif";

}

编译为：

p:before {

content: "Foo Bar";

font-family: sans-serif; }

默认情况下，运算表达式与其他值连用时，用空格做连接符：

p {

margin: 3px + 4px auto;

}

编译为：

p {

margin: 7px auto; }

在文本字符串中，#{}式插值可以用来在字符串中放置动态值：

p:before {

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

}

编译为：

p:before {

content: "I ate 15 pies!"; }

在字符串插值时，Null值被视为空字符串：

$value: null;

p:before {

content: "I ate #{$value} pies!";

}

编译为：

p:before {

content: "I ate pies!"; }

**布尔运算 (Boolean Operations)**

SassScript 支持布尔值的 and, or, 和 not 运算。

**列表运算 (List Operations)**

数组不支持任何特殊运算，只能使用 [list 函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#list-functions) 控制。

**圆括号 (Parentheses)**

圆括号可以用来影响运算的顺序(愚人码头注：优先级)：

p {

width: 1em + (2em \* 3);

}

编译为：

p {

width: 7em; }

**函数 (Functions)**

SassScript定义了一些有用的函数， 这些函数可以像普通 CSS 函数语法一样调用：

p {

color: hsl(0, 100%, 50%);

}

编译为：

p {

color: #ff0000; }

可用函数的完整列表，请参阅[这张页面](http://sass-lang.com/documentation/Sass/Script/Functions.html)。

**关键词参数 (Keyword Arguments)**

Sass 函数允许指定明确的关键词参数 (keyword arguments) 进行调用。 上面的例子也可以改写成：

p {

color: hsl($hue: 0, $saturation: 100%, $lightness: 50%);

}

虽然不够简明，但可以让Sass代码阅读起来更加方便。 关键词参数让函数具有更灵活的接口， 即便参数众多，也不会让使用变得困难。

命名参数（named arguments）可以以任意顺序传入，并且，具有默认值的参数可以省略掉。 由于命名参数也是变量名称，因此，下划线、短横线可以互换使用。

完整的 Sass 函数列表和它们的参数名称，以及在 Ruby 里如何定义你自己的函数的步骤，请见 [Sass::Script::Functions](http://sass-lang.com/documentation/Sass/Script/Functions.html)。

**插值：**#{}（**Interpolation:**#{}**）**

您还可以通过 #{} 插值语法在选择器和属性名中使用 SassScript 变量：

$name: foo;

$attr: border;

p.#{$name} {

#{$attr}-color: blue;

}

编译为：

p.foo {

border-color: blue; }

它也可以使用#{}插值语句把 SassScript 插入到属性值中。在大多数情况下，这种做可能还不如使用直接变量来的方便，但使用 #{}意味着靠近它的运算符都将被视为纯CSS（愚人码头注：可以避免各种运算）。 例如：

p {

$font-size: 12px;

$line-height: 30px;

font: #{$font-size}/#{$line-height};

}

编译为：

p {

font: 12px/30px; }

**SassScript中的**&**（**&**in SassScript）**

就像当它在[选择器](http://www.css88.com/doc/sass/#parent-selector)中使用一样，SassScript中的&指向当前父选择器。下面是一个逗号分隔的列表（list）中包含一个空格的分隔列表（list）。例如：

.foo.bar .baz.bang, .bip.qux {

$selector: &;

}

$selector的值是现在 ((".foo.bar" ".baz.bang"), ".bip.qux")。这个混合选择器在这里加了引号，以表明他们是字符串，但在现实中，他们将不带引号的。即使选择器不包含逗号或空格，&总会有两个嵌套层次，因此它可以保证访问一致性。

如果没有父选择器，&的值将是空。这意味着你可以在一个mixin中使用它来检测父选择是否存在：

@mixin does-parent-exist {

@if & {

&:hover {

color: red;

}

} @else {

a {

color: red;

}

}

}

**变量默认:**!default**（Variable Defaults:**!default**）**

如果分配给变量的值后面添加了!default标志 ，这意味着该变量如果已经赋值，那么它不会被重新赋值，但是，如果它尚未赋值，那么它会被赋予新的给定值。

例如:

$content: "First content";

$content: "Second content?" !default;

$new\_content: "First time reference" !default;

#main {

content: $content;

new-content: $new\_content;

}

编译为：

#main {

content: "First content";

new-content: "First time reference"; }

通过!default赋值的时候，如果变量是 null 值时，将视为未赋值（愚人码头注：所以下面的$content值为 "Non-null content"）：

$content: null;

$content: "Non-null content" !default;

#main {

content: $content;

}

编译为：

#main {

content: "Non-null content"; }

@**规则 和 指令 (**@**-Rules and Directives)**

Sass 支持所有CSS3的 @规则，以及一些已知的其他特定的Sass "指令"。这些在 Sass 都有对应的效果，更多资料请查看 [控制指令 (control directives)](http://www.css88.com/doc/sass/#control_directives) 和 [混入指令 (mixin directives)](http://www.css88.com/doc/sass/#mixins) 。

@import

Sass 扩展了 CSS @import规则，允许其导入 SCSS 或 Sass 文件。被导入的全部SCSS 或 Sass文件将一起合并到同一个 CSS 文件中。此外，被导入文件中所定义的任何变量或[混入（mixins）](http://www.css88.com/doc/sass/#mixins)都可以在主文件（愚人码头注：主文件值的是导入其他文件的文件，即，A文件中导入了B文件，这里的主文件指的就是A文件）中使用。

Sass 会在当前目录和 Rack, Rails, Merb目录下查找其他 Sass 文件。附加搜索目录可以使用[:load\_paths](http://www.css88.com/doc/sass/#load_paths-option)选项或命令行中的--load-path选项指定。

通常，@import 寻找 Sass 文件并将其导入，但在以下情况下，@import 仅作为普通的 CSS 语句，不会导入任何 Sass 文件。

@import 需要一个文件名来导入。默认情况下，它会寻找一个 Sass 文件直接导入，但在以下情况下， 仅作为普通的 CSS @import 规则语句，不会导入任何 Sass 文件。

* 如果文件的扩展名是 .css。
* 如果文件名以 http:// 开始。
* 如果文件名是 url()。
* 如果@import 中包含任何的媒体查询（media queries）。

如果没有上述条件得到满足并且扩展名是.scss 或.sass，那么 Sass 或 SCSS 文件将被导入。如果没有指定扩展名，Sass 将尝试找到以.scss 或.sass为扩展名的该名称文件并导入。

例如,

@import "foo.scss";

或

@import "foo";

这两行代码都能导入文件foo.scss，而

@import "foo.css";

@import "foo" screen;

@import "http://foo.com/bar";

@import url(foo);

将全部编译为

@import "foo.css";

@import "foo" screen;

@import "http://foo.com/bar";

@import url(foo);

Sass 支持在一个 @import 规则中同时导入多个文件。例如：

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

将同时导入rounded-corners和text-shadow 这两个文件。

导入规则中可能含有#{} 插值，但存在一定的限制。不能通过变量动态导入Sass文件；#{}插值仅适用于CSS导入规则。 因此，它仅适用于url() 导入。

例如：

$family: unquote("Droid+Sans");

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

将编译为

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

**Partials**

如果你有一个 SCSS 或 Sass 文件要导入，但不希望将其编译到一个CSS文件，你可以在文件名的开头添加一个下划线。这将告诉Sass不要将其编译到一个正常的CSS文件。然后，在导入语句中却不需要添加下划线。

例如，你可能有一个命名为 \_colors.scss 的文件，但是不会编译成 \_colors.css 文件。你可以这么做

@import "colors";

这样， \_colors.scss 将被导入。

注意，请不要将带下划线与不带下划线的同名文件放置在同一个目录下，比如，\_colors.scss 和 colors.scss 不能同时存在于同一个目录下。否则带下划线的文件将会被忽略。

**嵌套**@import**（Nested**@import**）**

虽然在大部分情况下，一般都是在文档的顶层（愚人码头注：最外层，不在嵌套规则内）使用 @import，但是也可以在CSS 规则和@media 规则中包含@import语句。就像一个基层的 @import ，这里会包含 @import 导入文件的内容。但是，这样导入的规则只能嵌套在原先防止 @import 的地方。

举个例子，如果 example.scss 包含

.example {

color: red;

}

然后（愚人码头注：导入到 #main 样式内）

#main {

@import "example";

}

愚人码头注：这样导入后等同于：

#main {

.example {

color: red;

}

}

将被编译为

#main .example {

color: red;

}

该指令只允许出现在文档顶层（愚人码头注：最外层，不在嵌套规则内），像@mixin 或者 @charset，在文件中，不允许被@import导入到一个嵌套上下文中。

不允许在混人 (mixin) 或控制指令 (control directives) 中嵌套 @import。

@media

Sass 中 @media 指令的行为和纯 CSS 中一样，只是增加了一点额外的功能：它们可以嵌套在CSS规则。如果一个@media 指令出现在CSS规则中，它将被冒泡到样式表的顶层，并且包含规则内所有的选择器。这使得很容易地添加特定media样式，而不需要重复使用选择器，或打乱样式表书写流。例如：

.sidebar {

width: 300px;

@media screen and (orientation: landscape) {

width: 500px;

}

}

编译为：

.sidebar {

width: 300px; }

@media screen and (orientation: landscape) {

.sidebar {

width: 500px; } }

@media的查询（queries）也可以相互嵌套。这些查询（queries）在编译时，将会使用 and 操作符号结合。例如：

@media screen {

.sidebar {

@media (orientation: landscape) {

width: 500px;

}

}

}

编译为：

@media screen and (orientation: landscape) {

.sidebar {

width: 500px; } }

@media 甚至可以使用 SassScript（比如变量，函数，以及运算符）代替条件的名称或者值：

最后，@media 查询（queries）可以包含 SassScript 表达式（包括变量 variables，函数 functions 和操作符operators）代替特征名称和特征值。

$media: screen;

$feature: -webkit-min-device-pixel-ratio;

$value: 1.5;

@media #{$media} and ($feature: $value) {

.sidebar {

width: 500px;

}

}

编译为：

@media screen and (-webkit-min-device-pixel-ratio: 1.5) {

.sidebar {

width: 500px; } }

@extend

设计一个页面时常常遇到这种情况：当一个样式类（class）含有另一个类的所有样式，并且它自己的特定样式。处理这种最常见的方法是在HTML同时使用一个通用样式类和特殊样式类。例如，假设我们设计需要一个普通错误的样式和一个严重错误的样式。我们可以类似这样写：

<div class="error seriousError">

Oh no! You've been hacked!

</div>

我们的样式如下

.error {

border: 1px #f00;

background-color: #fdd;

}

.seriousError {

border-width: 3px;

}

不幸的是，这意味着，我们必须时刻记住使用.seriousError的时候需要搭配使用.error。   
这对于维护来说是一个负担，甚至导致棘手的错误，并且导致无语意的样式。

@extend 指令避免这些问题，告诉 Sass 一个选择器的样式应该继承另一选择器。 例如：

.error {

border: 1px #f00;

background-color: #fdd;

}

.seriousError {

@extend .error;

border-width: 3px;

}

编译为：

.error, .seriousError {

border: 1px #f00;

background-color: #fdd;

}

.seriousError {

border-width: 3px;

}

这意味着.error说定义的所有样式也适用于.seriousError，除了.seriousError的特定样式。相当于，每个带有.seriousError类的元素也带有.error类。

其他使用了.error 规则也会同样继承给.seriousError，例如，如果我们有特殊错误样式的hack：

.error.intrusion {

background-image: url("/image/hacked.png");

}

然后<div class="seriousError intrusion">也同样会使用了 hacked.png 背景。

**它是如何工作的（How it Works）**

@extend通过在样式表中出现被扩展选择器（例如.error）的地方插入扩展选择器（例如.seriousError）。比如上面的例子：

.error {

border: 1px #f00;

background-color: #fdd;

}

.error.intrusion {

background-image: url("/image/hacked.png");

}

.seriousError {

@extend .error;

border-width: 3px;

}

编译为：

.error, .seriousError {

border: 1px #f00;

background-color: #fdd; }

.error.intrusion, .seriousError.intrusion {

background-image: url("/image/hacked.png"); }

.seriousError {

border-width: 3px; }

当合并选择器时，@extend 会很聪明地避免不必要的重复，所以像.seriousError.seriousError 将转换为 .seriousError，此外，她不会生成不能匹配任何元素的选择器（比如 #main#footer ）。

**扩展复杂的选择器（Extending Complex Selectors）**

Class 选择器并不是唯一可以被延伸 (extend) 的，Sass 允许延伸任何定义给单个元素的选择器，比如 .special.cool，a:hover 或者 a.user[href^="http://"] 等，例如：

类（class）选择，并不是唯一可以扩展。她可以扩展任何定义给单个元素的选择器，如.special.cool, a:hover, 或 a.user[href^="http://"]。 例如：

.hoverlink {

@extend a:hover;

}

同带 class 元素一样，这意味着，a:hover定义的样式同样也适用于.hoverlink。例如：

.hoverlink {

@extend a:hover;

}

a:hover {

text-decoration: underline;

}

编译为：

a:hover, .hoverlink {

text-decoration: underline; }

与上面 .error.intrusion 的例子一样， a:hover 中所有的样式将继承给 .hoverlink，甚至包括其他使用到她的样式，例如：

.hoverlink {

@extend a:hover;

}

.comment a.user:hover {

font-weight: bold;

}

编译为：

.comment a.user:hover, .comment .user.hoverlink {

font-weight: bold; }

**多重扩展 (Multiple Extends)**

同一个选择器可以扩展多个选择器。这意味着，它继承了被扩展选择器的所有样式。例如：

.error {

border: 1px #f00;

background-color: #fdd;

}

.attention {

font-size: 3em;

background-color: #ff0;

}

.seriousError {

@extend .error;

@extend .attention;

border-width: 3px;

}

编译为：

.error, .seriousError {

border: 1px #f00;

background-color: #fdd; }

.attention, .seriousError {

font-size: 3em;

background-color: #ff0; }

.seriousError {

border-width: 3px; }

每个带.seriousError类的元素也有.error类和.attention类。   
因此，定义在文档后面的样式优先级高于定义在文档前面的样式：.seriousError的背景颜色是#ff0，而非#fdd，因为 .attention 是在 .error 后面定义。

多重扩展也可以用逗号分隔的选择器列表（list）写入。例如，@extend .error, .attention等同于@extend .error; @extend .attention。

**链式扩展（Chaining Extends）**

一个选择器可以扩展另一个选择器，另一个选择器又扩展的第三选择器选择。 例如：

.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%;

}

现在，带 .seriousError 类的每个元素将包含 .error 类，而带 .criticalError 类的每个元素不仅包含 .criticalError类也会同时包含 .error 类，上面的代码编译为：

.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%; }

**选择器序列 (Selector Sequences)**

选择器序列，比如.foo .bar 或 .foo + .bar，目前还不能作为扩展。但是，选择器序列本身可以使用@extend。例如：

#fake-links .link {

@extend a;

}

a {

color: blue;

&:hover {

text-decoration: underline;

}

}

将被编译为：

a, #fake-links .link {

color: blue; }

a:hover, #fake-links .link:hover {

text-decoration: underline; }

**合并选择器序列 (Merging Selector Sequences)**

有时，选择器序列扩展另一个选择器，这个选择器出现在另一选择器序列中。在这种情况下，这两个选择器序列需要合并。例如：

#admin .tabbar a {

font-weight: bold;

}

#demo .overview .fakelink {

@extend a;

}

技术上讲能够生成所有匹配条件的结果，但是这样生成的样式表太复杂了，上面这个简单的例子就可能有 10 种结果。所以，Sass 只会编译输出有用的选择器。

当两个列 (sequence) 合并时，如果没有包含相同的选择器，将生成两个新选择器：第一列出现在第二列之前，或者第二列出现在第一列之前。例如：

#admin .tabbar a {

font-weight: bold;

}

#demo .overview .fakelink {

@extend a;

}

编译为：

#admin .tabbar a,

#admin .tabbar #demo .overview .fakelink,

#demo .overview #admin .tabbar .fakelink {

font-weight: bold; }

如果两个列 (sequence) 包含了相同的选择器，相同部分将会合并在一起，其他部分交替输出。在下面的例子里，两个列都包含 #admin，输出结果中它们合并在了一起：

#admin .tabbar a {

font-weight: bold;

}

#admin .overview .fakelink {

@extend a;

}

编译为：

#admin .tabbar a,

#admin .tabbar .overview .fakelink,

#admin .overview .tabbar .fakelink {

font-weight: bold; }

@extend**-Only 选择器 (**@extend**-Only Selectors)**

有时候你只会想写一个 @extend 扩展样式类，不想直接在你的HTML中使用。在写一个 Sass 样式库时，这是特别有用，如果他们需要，在这里你可以提供 @extend 扩展样式给用户，如果他们不需要，直接被忽视。

对于这种情况，如果使用普通的样式类，在你你最终生成的样式表中，会有很多额外（愚人码头注：无用）的CSS，并且在HTML被使用时，和其他样式类结合的时候容易造成冲突。这就是 Sass 为什么支持"占位选择器"的原因（例如，%foo）。

占位选择器看起来很像普通的 class 和 id 选择器，只是 # 或 . 被替换成了 %。他可以像 class 或者 id 选择器那样使用，而它本身的规则，不会被编译到 CSS 文件中。例如：

// This ruleset won't be rendered on its own.

#context a%extreme {

color: blue;

font-weight: bold;

font-size: 2em;

}

占位符选择器，就像class和id选择器那样可以用于扩展。扩展选择器，将会编译成CSS，占位符选择器本身不会被编译。例如：

.notice {

@extend %extreme;

}

编译为：

#context a.notice {

color: blue;

font-weight: bold;

font-size: 2em; }

!optional**标记（The**!optional**Flag）**

通常，当你扩展一个选择器的时候，如果说@extend不起作用了，你会收到一个错误提示。   
例如，如果没有 .notice 选择器,你这么写a.important {@extend .notice},将会报错。如果只有h1.notice一个选择器包含了.notice，那么也会报错。因为 h1 会与 a 冲突，并且不会生成新的选择器。

然而，有时候，要想@extend不生成任何新的选择器。只是在选择器后添加 !optional标志就可以了。例如：

a.important {

@extend .notice !optional;

}

**指令中的**@extend**(**@extend**in Directives)**

在指令中使用 @extend 时（比如在@media 中）存在一些限制：Sass 不可以将 @media 层外的 CSS 规则扩展给指令层内的 CSS，这样会生成大量的无用代码。意思是说，如果在 @media （或者其他 CSS 指令）中使用@extend，必须扩展给相同指令层中的选择器。

下面的例子是可行的：

@media print {

.error {

border: 1px #f00;

background-color: #fdd;

}

.seriousError {

@extend .error;

border-width: 3px;

}

}

但下面这个例子会报错：

.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;

}

}

希望有一天，浏览器可以原生支持 @extend 指令，这样就可以在@media和其他指令中使用扩展功能了。

@at-root

@at-root指令导致一个或多个规则被限定输出在文档的根层级上，而不是被嵌套在其父选择器下。   
它可以被用于单一或内联选择器：

.parent {

...

@at-root .child { ... }

}

这将生成：

.parent { ... }

.child { ... }

或者它可以用于包含多个选择器的代码块：

.parent {

...

@at-root {

.child1 { ... }

.child2 { ... }

}

.step-child { ... }

}

这将输出如下：

.parent { ... }

.child1 { ... }

.child2 { ... }

.parent .step-child { ... }

@at-root (without: ...)**和**@at-root (with: ...)**（**@at-root (without: ...)**and `@at-root (with: ...)）**

默认情况下， @at-root只是排除了选择器。然而，它也可以使用@at-root将选择器移动到嵌套指令（比如@media）之外。例如：

@media print {

.page {

width: 8in;

@at-root (without: media) {

color: red;

}

}

}

生成：

@media print {

.page {

width: 8in;

}

}

.page {

color: red;

}

您可以使用@at-root (without: ...) 将规则移动到任何指令之外。你同样可以让多个指令做到这一点,只要多个指令使用空格分隔就可以了：@at-root (without: media supports)会将规则移动到@media 和 @supports查询（queries）之外。

还有有两个特殊值你可以传递给@at-root。"rule"是指正常的CSS规则;@at-root (without: rule)等价于没有查询的@at-root。 @at-root (without: all) 意思是该样式应该移动到*全部*的指令和CSS规则之外。

如果你想指定哪个指令或规则包含，而不是哪些应该排除，那么，你可以使用with代替without。   
例如，@at-root (with: rule)将规则移动到所有指令之外，但在CSS规则内会保留。

@debug

@debug指令打印SassScript表达式的值到标准的错误输出流。这对于调试具有复杂SassScript 表达式的Sass文件非常有用的。 例如：

@debug 10em + 12em;

输出：

Line 1 DEBUG: 22em

@warn

@warn指令打印SassScript表达式的值到标准的错误输出流。这对于警告用户弃用库 或 修复 mixin 轻微的错误是非常有用的。@warn和@debug之间有两个主要区别：

1. 您可以使用--quiet命令行选项或:quiet Sass选项关闭警告。
2. 样式表跟踪将与消息一起被打印出来，这样，用户可以看到他们的样式在哪里引起了警告。

用法示例:

@mixin adjust-location($x, $y) {

@if unitless($x) {

@warn "Assuming #{$x} to be in pixels";

$x: 1px \* $x;

}

@if unitless($y) {

@warn "Assuming #{$y} to be in pixels";

$y: 1px \* $y;

}

position: relative; left: $x; top: $y;

}

@error

@error指令抛出一个SassScript表达式的值作为一个致命的错误，其中包括一个不错的堆栈跟踪。这对于验证混入（mixin）和函数的参数很有用。例如：

@mixin adjust-location($x, $y) {

@if unitless($x) {

@error "$x may not be unitless, was #{$x}.";

}

@if unitless($y) {

@error "$y may not be unitless, was #{$y}.";

}

position: relative; left: $x; top: $y;

}

目前还没有办法捕获错误。

**控制指令和表达式（Control Directives & Expressions）**

SassScript支持一些基本控制指令和表达式，比如仅在在某些条件下包含样式，或者包括相同的样式几次变化。

**注意：** 控制指令是一项高级功能，日常编写过程中并不常用到，主要在 [mixins（混合）](http://www.css88.com/doc/sass/#mixins)指令中使用，尤其是像[Compass](http://compass-style.org/)这样的库。

if()

内置的if()函数可让您在一个条件处理分支并返回两种可能结果。它可以在任何脚本上下文中使用。if函数只判断相对应的一个参数并且返回 -- 这使您可以引用已经定义的或者可以计算的变量，否则将导致错误（例如，除以零）。

if(true, 1px, 2px) => 1px

if(false, 1px, 2px) => 2px

@if

@if 指令需要一个SassScript表达和嵌套在它下面要使用的样式，如果表达式返回值不为 false 或者 null ，那么后面花括号中的内容就会返回：

p {

@if 1 + 1 == 2 { border: 1px solid; }

@if 5 < 3 { border: 2px dotted; }

@if null { border: 3px double; }

}

编译为：

p {

border: 1px solid; }

@if 语句后面可以跟多个@else if语句和一个 @else 语句。   
 如果@if语句失败，Sass 将逐条尝试@else if 语句，直到有一个成功，或如果全部失败，那么会执行@else语句。 例如：

$type: monster;

p {

@if $type == ocean {

color: blue;

} @else if $type == matador {

color: red;

} @else if $type == monster {

color: green;

} @else {

color: black;

}

}

编译为：

p {

color: green; }

@for

@for指令重复输出一组样式。对于每次重复，计数器变量用于调整输出结果。该指令有两种形式：@for $var from <start> through <end> 和 @for $var from <start> to <end>。注意关键字through 和 to的区别。$var可以是任何变量名，比如$i;<start> 和 <end>是应该返回整数的SassScript表达式。当<start>比<end>大的时候，计数器将递减，而不是增量。

@for语句将设置$var为指定的范围内每个连续的数值，并且每一次输出的嵌套样式中使用$var的值。对于from ... through的形式，范围*包括*<start>和<end>的值，但from ... to的形式从<start>开始运行，*但不包括*<end>的值。使用through语法，

@for $i from 1 through 3 {

.item-#{$i} { width: 2em \* $i; }

}

编译为：

.item-1 {

width: 2em; }

.item-2 {

width: 4em; }

.item-3 {

width: 6em; }

@each

@each指令通常格式是@each $var in <list or map>。$var可以是任何变量名，像$length 或者 $name，和<list or map>是一个返回列表（list）或 map 的 SassScript 表达式。

@each 规则将$var设置为列表（list）或 map 中的每个项目，输出样式中包含使用$var的值。 例如：

@each $animal in puma, sea-slug, egret, salamander {

.#{$animal}-icon {

background-image: url('/images/#{$animal}.png');

}

}

编译为：

.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'); }

**多重赋值（Multiple Assignment）**

@each指令也可以使用多个变量，格式为@each $var1,$var2, ... in <list>。如果<list>是列表（list）中的列表，子列表中的每个元素被分配给各自的变量。例如：

@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;

}

}

编译为：

.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; }

因为[maps](http://www.css88.com/doc/sass/#maps)被视为键值对的列表，所以多重赋值也可以很好的工作。例如：

@each $header, $size in (h1: 2em, h2: 1.5em, h3: 1.2em) {

#{$header} {

font-size: $size;

}

}

编译为：

h1 {

font-size: 2em; }

h2 {

font-size: 1.5em; }

h3 {

font-size: 1.2em; }

@while

@while 指令重复输出嵌套样式，直到SassScript表达式返回结果为false。这可用于实现比@for语句更复杂的循环，只是很少会用到例如：

$i: 6;

@while $i > 0 {

.item-#{$i} { width: 2em \* $i; }

$i: $i - 2;

}

编译为：

.item-6 {

width: 12em; }

.item-4 {

width: 8em; }

.item-2 {

width: 4em; }

**混入指令 (Mixin Directives)**

混入(mixin)允许您定义可以在整个样式表中重复使用的样式，而避免了使用无语意的类（class），比如 .float-left。混入(mixin)还可以包含所有的CSS规则，以及任何其他在Sass文档中被允许使用的东西。   
他们甚至可以带[arguments](http://www.css88.com/doc/sass/#mixin-arguments)，引入变量，只需少量的混入(mixin)代码就能输出多样化的样式。

**定义一个混入(mixin):**@mixin**（Defining a Mixin:**@mixin**）**

混入(mixin)通过 @mixin 指令定义。在它后面跟混入的名称和任选的[arguments（参数）](http://www.css88.com/doc/sass/#mixin-arguments)，以及混入的内容块。例如，large-text混入定义如下：

@mixin large-text {

font: {

family: Arial;

size: 20px;

weight: bold;

}

color: #ff0000;

}

混入也可以包含选择器和属性的混合体，选择器中甚至可以包含[parent references（父选择器）](http://www.css88.com/doc/sass/#referencing_parent_selectors_)。 例如：

@mixin clearfix {

display: inline-block;

&:after {

content: ".";

display: block;

height: 0;

clear: both;

visibility: hidden;

}

\* html & { height: 1px }

}

由于历史原因，混入（mixin）的名字（和所有其他 Sass 标识符）可以互换连字符和下划线。例如，如果你定义了一个名为add-column的混入，你可以把它作为add\_column，反之亦然。

**引用混合样式:**@include**（Including a Mixin:**@include**）**

使用 @include 指令可以将混入（mixin）引入到文档中。这需要一个混入的名称和可选的[参数传递给它](http://www.css88.com/doc/sass/#mixin-arguments)，并包括由混入定义的当前规则的样式。 例如：

.page-title {

@include large-text;

padding: 4px;

margin-top: 10px;

}

编译为：

.page-title {

font-family: Arial;

font-size: 20px;

font-weight: bold;

color: #ff0000;

padding: 4px;

margin-top: 10px; }

混入（mixin）也可以包含在任何规则的外（即，在文档的根）,只要它们不直接定义的任何属性或使用任何父选择器引用。例如：

@mixin silly-links {

a {

color: blue;

background-color: red;

}

}

@include silly-links;

编译为：

a {

color: blue;

background-color: red; }

混入（mixin）定义也可以包含其他的混入。例如：

@mixin compound {

@include highlighted-background;

@include header-text;

}

@mixin highlighted-background { background-color: #fc0; }

@mixin header-text { font-size: 20px; }

混入可以包含自己。这行为不同于 Sass 3.3 之前的版本，以前混入递归是被禁止的。

只定义后代选择器的混入可以安全地混入到文件的最顶层。

**参数 (Arguments)**

混入（mixin）可以用 SassScript 值作为参数，给定的参数被包括在混入（mixin）中并且作为为变量提供给混入（mixin）。

当定义一个混入（mixin）的时候，参数被作为变量名，写到混入（mixin）名字后面的括号内，多个参数可以用逗号分隔。然后，当调用混入的时候，值通过对应的参数顺序被传递。 例如：

@mixin sexy-border($color, $width) {

border: {

color: $color;

width: $width;

style: dashed;

}

}

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

编译为：

p {

border-color: blue;

border-width: 1in;

border-style: dashed; }

混入（mixin）也可以使用普通的变量赋值语法为参数指定默认值。然后，当调用混入的时候，如果没有给参数赋值，则自动会使用默认值代替。 例如：

@mixin sexy-border($color, $width: 1in) {

border: {

color: $color;

width: $width;

style: dashed;

}

}

p { @include sexy-border(blue); }

h1 { @include sexy-border(blue, 2in); }

编译为：

p {

border-color: blue;

border-width: 1in;

border-style: dashed; }

h1 {

border-color: blue;

border-width: 2in;

border-style: dashed; }

**关键字参数 (Keyword Arguments)**

混入（mixin）在引入（@include指令）的时候也可以使用明确的关键字参数。例如，上面的例子可以写成：

p { @include sexy-border($color: blue); }

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

虽然这是不够简明，但是它可以使样式表更容易阅读。它给函数呈现了更加灵活的接口，它使多参数的混入更加容易调用。

命名的参数可以按任何顺序进行传递，有默认值的参数可以省略。由于命名参数是变量名，下划线和连字符可以互换使用。

**可变参数 (Variable Arguments)**

有时，不能确定一个混入（mixin）或者一个函数（function）使用多少个参数。例如，用于创建盒子阴影（box-shadow）的一个混入（mixin）可以采取任何数量的box-shadow作为参数。对于这些情况，Sass支持"可变参数",参数在声明混入（mixin）或函数（function）结束的地方，所有剩余的参数打包成一个[列表（list）](http://www.css88.com/doc/sass/#lists)。参数看起来就像普通参数一样，但后面跟随着...。例如：

@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);

}

编译为：

.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;

}

可变参数可以包含任何关键字参数传递给混入（mixin）或者函数（function）。这些可以使用[keywords($args)函数](http://sass-lang.com/documentation/Sass/Script/Functions.html#keywords-instance_method) 来访问，返回一个map，参数名称字符串（无$）和值的键值对。

可变参数，也可以在调用（@include指令）一个混入（mixin）时使用。使用相同的语法，你可以扩展值的列表（list），以便每个值作为单独的参数传入，或扩展值的map，以使每个键值对作为一个关键字参数处理。例如：

@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...);

}

编译为：

.primary {

color: #ff0000;

background-color: #00ff00;

border-color: #0000ff;

}

.secondary {

color: #00ff00;

background-color: #0000ff;

border-color: #ff0000;

}

你可以同时传递一个列表（list）和一个map参数，只要列表（list）在map上之前，比如@include colors($values..., $map...)。

您可以使用可变参数来包装一个混入（mixin）并且添加额外的样式，而不改变混入（mixin）的参数签名。如果你这样做，关键字参数将通过包装的混入（mixin）直接传递。例如：

@mixin wrapped-stylish-mixin($args...) {

font-weight: bold;

@include stylish-mixin($args...);

}

.stylish {

// The $width argument will get passed on to "stylish-mixin" as a keyword

@include wrapped-stylish-mixin(#00ff00, $width: 100px);

}

**传递内容块到混入(Passing Content Blocks to a Mixin)**

样式内容块可以传递到混入（mixin）包含样式的位置。样式内容块将出现在混入内的任何 @content 指令的位置。这使得可以定义抽象 关联到选择器和指令的解析。

例如：

@mixin apply-to-ie6-only {

\* html {

@content;

}

}

@include apply-to-ie6-only {

#logo {

background-image: url(/logo.gif);

}

}

生成:

\* html #logo {

background-image: url(/logo.gif);

}

同样的混入（mixin）可以在.sass 简写语法（@mixin 可以用 = 表示，而 @include 可以用 + 表示）来完成：

=apply-to-ie6-only

\* html

@content

+apply-to-ie6-only

#logo

background-image: url(/logo.gif)

**注意：** 当@content指令指定多次或在一个循环中指定的时候，样式块将在每次调用中被复制并引用。

**变量的作用域和内容块（Variable Scope and Content Blocks）**

传递给混入（mixin）的内容块在其被定义的作用域中进行运算，而不是混入（mixin）的作用域。这意味着混入（mixin）的局部变量**不能**传递给样式块使用，并且变量将解析为全局值：

$color: white;

@mixin colors($color: blue) {

background-color: $color;

@content;

border-color: $color;

}

.colors {

@include colors { color: $color; }

}

编译为:

.colors {

background-color: blue;

color: white;

border-color: blue;

}

另外，这清楚地表明，变量和传递到块中使用的混入，指向块定义的周围其他样式。例如：

#sidebar {

$sidebar-width: 300px;

width: $sidebar-width;

@include smartphone {

width: $sidebar-width / 3;

}

}

**函数指令 (Function Directives)**

Sass 支持自定义函数，并能在任何值或脚本上下文中使用。例如

$grid-width: 40px;

$gutter-width: 10px;

@function grid-width($n) {

@return $n \* $grid-width + ($n - 1) \* $gutter-width;

}

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

就变成了:

#sidebar {

width: 240px; }

正如你看到的，函数可以访问任何全局定义的变量，以及接受参数，就像一个混入（mixin）。函数可以包含语句，并且你必须调用@return来设置函数的返回值。

与混入（mixin）一样，你可以使用关键字参数来调用Sass定义的函数。在上面的例子中，我们可以这样调用函数：

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

建议您在函数前加上前缀，以避免命名冲突，其他人阅读样式表的时候也会知道它们不是 Sass 或者 CSS 的自带功能。例如，如果您在ACME公司工作，你可以给上面的函数取名为-acme-grid-width。

用户自定义的函数也支持[可变参数](http://www.css88.com/doc/sass/#variable_arguments)，方式和混入（mixin）是相同的。

由于历史的原因，函数名（和所有其他Sass标识符）中连字符和下划线可以互换。例如，如果你定义了一个名为grid-width的函数，你可以通过grid\_width调用它，反之亦然。

**输出格式 (Output Style)**

虽然Sass 默认的 CSS 输出格式非常好，并且能反映文档的结构，但是由于每个人的喜好和需求各不相同，因此Sass 支持其他几种格式。

Sass 提供了四种输出格式，可以通过[:style 选项](http://www.css88.com/doc/sass/#style-option) 选项设定，或者在命令行中使用 --style 选项。

Sass 允许您通过设置[:style 选项](http://www.css88.com/doc/sass/#style-option) 或使用 --style 命令行标志，在四种不同的输出格式之间进行选择。

:nested

nested（嵌套）格式是 Sass 默认的输出格式，因为它的格式反映CSS样式与HTML文档结构。每个属性都独占用一行，但缩排不是固定的。每个规则是基于它的何嵌套深度缩进。例如：

#main {

color: #fff;

background-color: #000; }

#main p {

width: 10em; }

.huge {

font-size: 10em;

font-weight: bold;

text-decoration: underline; }

当阅读大型 CSS 文件时，nested（嵌套）格式是非常有用的：不用详细阅读，就可以让你轻松掌握文件的结构。

:expanded

expanded（扩展）格式更像是手写的CSS样式，每个属性和规则都独占用一行。在规则之内的属性缩进的，但规则没有任何特殊的缩进。例如：

#main {

color: #fff;

background-color: #000;

}

#main p {

width: 10em;

}

.huge {

font-size: 10em;

font-weight: bold;

text-decoration: underline;

}

:compact

compact（紧凑）格式比起nested（嵌套）或expanded（扩展）格式占据更小的空间。这种格式重点聚焦在选择器上，不是它们的属性。每个CSS规则独占用一行，该行还包括定义的每个属性。嵌套的规则都是另起一行，不嵌套的选择器会输出空白行作为分隔符。 例如：

#main { color: #fff; background-color: #000; }

#main p { width: 10em; }

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

:compressed

compressed（压缩）格式占用尽可能小的空间，在该文件的末尾会有一个换行，并且除了必要的分隔选择器之外，基本没有多余空格，它还包括其他一些小的压缩，比如选择颜色最小的表示方式。这意味着可读性很差。 例如：

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

**扩展 Sass (Extending Sass)**

对于独特的需求，Sass为用户提供了多项高级定制功能。使用这些功能需要对Ruby有深刻的理解。

**自定义 Sass 函数 (Defining Custom Sass Functions)**

用户通过 Ruby API 可以自定义 Sass 函数，更多信息请查看 [源代码文档](http://sass-lang.com/documentation/Sass/Script/Functions.html#adding_custom_functions)。

**缓存存储（Cache Stores）**

Sass会缓存已经解析的文档，这使得它们可以重复使用，而无需再次解析，除非他们已经被更改。 默认情况下，Sass会将这些缓存文件写到 [:cache\_location](http://www.css88.com/doc/sass/#cache_location-option)指定的文件系统中。如果你不能写入文件系统或者需要ruby进程或计算机共享缓存，那么你可以定义自己的缓存存储，并设置[:cache\_store选项](http://www.css88.com/doc/sass/#cache_store-option)。有关创建自定义缓存存储的详细信息，请查看[源代码文档](http://sass-lang.com/documentation/Sass/CacheStores/Base.html)。

**自定义导入 (Custom Importers)**

Sass导入主要负责获取路径传递给@import并找到这些路径相应的Sass代码。默认情况下，这些代码是从[文件系统](http://sass-lang.com/documentation/Sass/Importers/Filesystem.html)中加载，但是Importers可以从数据库加载，通过HTTP，或者使用不同的文件命名方案，被添加到Sass。

每个importer负责一个单独的加载路径（或任何相应的后端概念）。importer可以和普通的文件系统路径一起放置在[:load\_paths数组](http://sass-lang.com/documentation/file.SASS_REFERENCE.html#load_paths-option)中。

当解析一个@import的时候，Sass将通过加载路径寻找importer来成功地导入路径。一旦被发现，该导入的文件就会被使用。

用户创建的导入必须继承自 [Sass::Importers::Base](http://sass-lang.com/documentation/Sass/Importers/Base.html)。