# SASS官方文档

作者：大漠 日期：2013-07-29 点击：2560

[preprocessor](http://www.w3cplus.com/blog/tags/309.html" \t "http://www.w3cplus.com/preprocessor/_blank)

[sass](http://www.w3cplus.com/blog/tags/302.html" \t "http://www.w3cplus.com/preprocessor/_blank)

[scss](http://www.w3cplus.com/blog/tags/368.html" \t "http://www.w3cplus.com/preprocessor/_blank)

[](http://www.miaov.com/2013/download/video_download.html)

本文由[Peter](http://peters-playground.com/" \t "http://www.w3cplus.com/preprocessor/_blank)根据[SASS](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html" \t "http://www.w3cplus.com/preprocessor/_blank)官方文档所译，觉得非常的棒，所以 借手转发在[W3cplus](http://www.w3cplus.com/" \t "http://www.w3cplus.com/preprocessor/_blank)上，让更多爱好SASS的同学受益。

如果您需要转载此文，请注明来自于[Peter](http://peters-playground.com/" \t "http://www.w3cplus.com/preprocessor/_blank)的《[Sass官方文档翻译](http://peters-playground.com/2013/01/sass-chinese-reference/" \t "http://www.w3cplus.com/preprocessor/_blank)》。并在显眼之处注明中处出处：[http://peters-playground.com/2013/01/sass-chinese-reference/](http://peters-playground.com/2013/01/sass-chinese-reference/" \t "http://www.w3cplus.com/preprocessor/_blank).英文出处：[http://sass-lang.com/docs/yardoc/file.SASS\_REFERENCE.html](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html" \t "http://www.w3cplus.com/preprocessor/_blank)。

使用 Sass 已经有几个月的时间了，翻来覆去还是在用几个简单的功能，下决心好好学习一下，所以翻译了官方文档。翻译的过程也是学习与积累的过程，深入理解了几个复杂功能，同时也补充了一些基础知识。只是官方文档写得实在绕口，花了很久才翻译完，有几处地方不太理解，翻译不出来，还请大家帮忙，理解错误的地方也请大家指正。关于 Sass 设置的部分暂时未翻译，因为只会用[Codekit](http://incident57.com/codekit/" \t "http://www.w3cplus.com/preprocessor/_blank)，对这些没有概念。如果哪位愿意代为翻译，或者交流，[请联系 Peter](mailto:40132147&" \l "64;&#113;&#113;&#46;&#99;&#111;&#109;?Subject=Hi Peter" \t "http://www.w3cplus.com/preprocessor/_blank)。

Sass 官方文档，[原地址在请看这里](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html" \t "http://www.w3cplus.com/preprocessor/_blank)，相当于 Sass 的功能概览，要学好 Sass 还是要多方面学习，[《Sass and Compass in action》](http://manning.com/netherland/" \t "http://www.w3cplus.com/preprocessor/_blank)这本书下个月底出版，非常期待。

## **1. 特色功能 (Features)**

Sass 的主要特色功能包括：

* 完全兼容 CSS3；
* 在 CSS 语言基础上添加了拓展功能，比如变量、嵌套 (nesting)、混合 (mixin) 等等；
* [函数](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \t "http://www.w3cplus.com/preprocessor/_blank)运算颜色值与属性值 (property value) ；
* 高级命令，如 [控制指令 (control directives)](http://www.w3cplus.com/preprocessor/sass-chinese-reference.html" \l "8" \t "http://www.w3cplus.com/preprocessor/_blank)；
* 可定义多种输出格式；
* [支持 Firebug](https://addons.mozilla.org/en-US/firefox/addon/firesass-for-firebug/" \t "http://www.w3cplus.com/preprocessor/_blank)。

## **2. 语言格式 (Syntax)**

Sass 包含两种语言格式。第一种被称为 SCSS (Sassy CSS)，也是本文示例所使用的格式，作为 CSS3 语言的拓展。也就是说，所有的 CSS3 样式都可以写进 SCSS 文件中，除此之外，SCSS 也能够理解大多数的 CSS hacks 以及浏览器前缀 (vendor-specific)，甚至早期的 IE 滤镜，这些都依赖于 Sass 的特色功能，下文将会详细介绍。这种格式以 .scss作为拓展名。

另一种格式出现的时间更早一些，被成为缩进格式（或者 “Sass”），这是一种让书写 CSS 变得更简洁的方式。它使用“缩进”替代“括号”表示属性嵌套在选择器中，用“换行”替代“分号”表示属性的区分。不少人认为这种格式比 SCSS 更容易阅读，书写也更快速。缩进格式可使用 Sass 的所有功能，只是个别地方采用了不同的表达方式，具体可以通过 [the indented syntax reference](http://sass-lang.com/docs/yardoc/file.INDENTED_SYNTAX.html" \t "http://www.w3cplus.com/preprocessor/_blank)查看区别。这种格式以 .sass作为拓展名。

每种格式都可以被 [导入 (@import)](http://www.w3cplus.com/preprocessor/sass-chinese-reference.html" \l "7_1" \t "http://www.w3cplus.com/preprocessor/_blank) 到另一种格式中使用，也可以通过 sass-convert指令互相转换。

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

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

## **3. 使用 Sass (Using Sass)**

Sass 可以通过以下三种方式使用：在命令行工具中，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）。第一步，安装 Sass gem （Windows 系统下，需要先[安装 Ruby](http://rubyinstaller.org/" \t "http://www.w3cplus.com/preprocessor/_blank)）：

gem install sass

在命令行中运行 Sass：

sass input.scss output.css

监视 Sass 文件，每次更新时，自动编译成 CSS：

sass --watch input.scss:output.css

监视整个文件夹：

sass --watch app/sass:public/stylesheets

更多命令请通过 sass --help查看。

在 Ruby 代码中使用 Sass 也非常容易，安装完毕后，运行 require "sass"然后使用 [Sass::Engine](http://sass-lang.com/docs/yardoc/Sass/Engine.html" \t "http://www.w3cplus.com/preprocessor/_blank)，例如

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

以下关于 Sass 设置的内容暂未翻译，请查看 [原文档](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html" \l "rackrailsmerb_plugin" \t "http://www.w3cplus.com/preprocessor/_blank)。

## **4. CSS 拓展 (CSS Extensions)**

### 4.1. 嵌套规则 (Nested Rules)

Sass 可以将一套 CSS 样式嵌套进另一套样式中使用，嵌套后的样式将继承给它的外层选择器（或父选择器），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; }

嵌套功能可以避免重复使用父选择器，并且让书写复杂的 CSS 结构变得简单、容易，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; }

### 4.2. 引用父选择器 &(Referencing Parent Selectors: &)

在嵌套 CSS 规则时，Sass 提供了一种简便的办法引用父选择器，比如，当给一个元素设定 hover 样式的时候，或者 body元素确定了 class 的时候，这时，可以用 &表示父选择器。

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; }

在编译过的 CSS 文件中，&将被替换成外层的父选择器，如果使用了多层嵌套（最好不要超过三层），最外层的父选择器会被完整地替换给每一层。

*#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; }

### 4.3. 属性嵌套 (Nested Properties)

有些 CSS 属性遵循相同的命名规则 (namespace)，比如 font-family, font-size, font-weight都以 font作为属性名的前半部分（注，这个前半部分就是 namespace，也可单独作 CSS 属性）。使用一套遵循相同命名规则的属性，每次都要重复输入 namespace，略显繁琐。为此，Sass 提供了一种简便的方法，只需书写 namespace 一次，然后在其中嵌套子属性：

.funky {

font: {

family: fantasy;

size: 30em;

weight: bold;

}}

编译为

.funky {

font-family: fantasy;

font-size: 30em;

font-weight: bold; }

Namespace 也可以先设定属性值，再开始嵌套，比如：

.funky {

font: 2px/3px {

family: fantasy;

size: 30em;

weight: bold;

}}

编译为

.funky {

font: 2px/3px;

font-family: fantasy;

font-size: 30em;

font-weight: bold; }

### 4.4. 占位符选择器 %foo(Placeholder Selectors: %foo)

Sass 支持一种特殊类型的选择器，占位符选择器 (placeholder selector)，看起来很像普通的 id或 class选择器，只是 #或.被替换成了 %。通过 [@extend](http://www.w3cplus.com/preprocessor/sass-chinese-reference.html" \l "7_3" \t "http://www.w3cplus.com/preprocessor/_blank) 指令调用，更多信息请查看 [@extend-Only Selectors](http://www.w3cplus.com/preprocessor/sass-chinese-reference.html" \l "7_3_6" \t "http://www.w3cplus.com/preprocessor/_blank)。

当它们单独使用时（没有通过 @extend调用），不会被编译到 CSS 文件中。

## **5. 注释 /\* \*/与 //(Comments: /\* \*/and //)**

Sass 使用 CSS 的多行注释 /\* \*/，同时也支持单行注释 //，前者在编译时将被完整地输出到 CSS 文件中，后者则会被移除。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; }

在压缩输出模式下 (compressed)，如果需要将注释编译到 CSS 文件中，可以将多行注释的第一个字母改为 !，便于添加版权信息。

## **6. SassScript**

除了与 CSS 一样的语法，Sass 也提供了一些被称为 SassScript 的拓展功能。 SassScript 可作用于任何属性，允许属性使用变量、算数运算等额外功能。在使用混合指令 (mixin) 时，SassScript 甚至可以生成选择器或属性名（暂时还不能生成变量）。

### 6.1. Interactive Shell

通过 Interactive Shell 测试 SassScript 的功能。在命令行中输入 sass -i，然后，输入你想测试 SassScript 查看输出结果。

$ sass -i>> "Hello, Sassy World!""Hello, Sassy World!">> 1px + 1px + 1px3px>> #777 + #777#eeeeee>> #777 + #888white

### 6.2. 变量 $(Variables: $)

SassScript 最直接的用法就是变量，变量以美元符号开头，赋值方法与 CSS 属性的用法一样。

$width: 5em;

然后可以这样调用变量

#main {

width: $width;}

嵌套规则内定义的变量只能在嵌套规则内使用，而嵌套规则外（最外层）定义的变量，则可以在任何地方使用。

旧版本的 Sass 使用 !作为变量的前缀字符，用 =赋值，现在这两种方式仍然可以用，但会提示错误，所以请使用 $与 :。

### 6.3. 数据类型 (Data Types)

SassScript 支持六种不同的数据类型：

* 数字，1, 2, 13, 10px；
* 字符串，有引号字符串或无引号字符串，"foo", 'bar', baz；
* 颜色，blue, #04a3f9, rgba(255,0,0,0.5)；
* 布尔型，true, false；
* 空值，null；
* 值列表，用空格或者逗号分开，1.5em 1em 0 2em, Helvetica, Arial, sans-serif。

SassScript 也支持其他 CSS 属性值（property value），比如 Unicode 范围，或 !important声明。然而，Sass 不会特殊对待这些属性值，一律视为无引号字符串 (unquoted strings)。

#### 6.3.1. 字符串 (Strings)

SassScript 支持 CSS 的两种字符串类型：有引号字符串 (quoted strings)，如 "Lucida Grande"'http://sass-lang.com'；与无引号字符串 (unquoted strings)，如 sans-serifbold，在编译 CSS 文件时不会改变其类型。只有一种情况例外，使用 #{ }插值语句 (interpolation) 时，有引号字符串将被编译为无引号字符串，这样方便了在混合指令 (mixin) 中引用选择器名。

@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!"; }

需要注意的是：当 deprecated = property syntax 时 （暂时不理解是怎样的情况），所有的字符串都将被编译为无引号字符串，不论是否使用了引号。

#### 6.3.2. 值列表 (Lists)

所谓值列表 (lists) 是指 Sass 如何处理 CSS 中 margin: 10px 15px 0 0或者 font-face: Helvetica, Arial, sans-serif这样通过空格或者逗号分隔的一系列的值。事实上，独立的值也被视为值列表——只包含一个值的值列表。

[Sass list functions](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \l "list-functions" \t "http://www.w3cplus.com/preprocessor/_blank) 赋予了值列表更多功能：nth function 可以直接访问值列表中的某一项，join function 可以将多个值列表连结在一起，append function 可以在值列表中添加值，而 @each rule 则能够给值列表中的每个项目添加样式。

值列表中可以再包含值列表，比如 1px 2px, 5px 6px是包含 1px 2px与 5px 6px两个值列表的值列表。如果内外两层值列表使用相同的分隔方式，要用圆括号包裹内层，所以也可以写成 (1px 2px) (5px 6px)。当值列表被编译为 CSS 时，Sass 不会添加任何圆括号，因为 CSS 不允许这样做。(1px 2px) (5px 6px)与 1px 2px 5px 6px在编译后的 CSS 文件中是一样的，但是它们在 Sass 文件中却有不同的意义，前者是包含两个值列表的值列表，而后者是包含四个值的值列表。

可以用 ()表示空的列表，这样不可以直接编译成 CSS，比如编译 font-family: ()时，Sass 将会报错。如果值列表中包含空的值列表或空值，编译时将清除空值，比如 1px 2px () 3px或 1px 2px null 3px。

### 6.4. 运算符 (Operations)

所有数据类型均支持相等运算符 ==或 !=，此外，每种数据类型也有其特殊支持的运算符。

#### 6.4.1. 数字运算 (Number Operations)

SassScript 支持数字的加减乘除、取整等运算 (+, -, \*, /, %)，如果需要也可以在不同单位间转换值。

p {

width: 1in + 8pt;}

编译为

p {

width: 1.111in; }

关系运算符 <, >, <=, >=也可用于数字运算，相等运算符 ==, !=可用于所有数据类型。

#### 6.4.1.1. 除法运算 /(Division and /)

/在 CSS 中通常起到分隔数字的用途，SassScript 作为 CSS 语言的拓展当然也继承了这个功能，同时，赋予了 /除法运算的功能。也就是说，默认情况下，如果 /在 SassScript 中把两个数字分隔，编译的 CSS 文件中也是同样的作用。

以下三种情况下 /将被视为除法运算符号（通常只有这三种情况）：

* 如果值，或值的一部分，是变量；
* 如果值被圆括号包裹；
* 如果值是算数表达式的一部分。

p {

font: 10px/8px; // Plain CSS, no division

$width: 1000px;

width: $width/2; // Uses a variable, does division

height: (500px/2); // Uses parentheses, does division

margin-left: 5px + 8px/2px; // Uses +, does division}

编译为

p {

font: 10px/8px;

width: 500px;

height: 250px;

margin-left: 9px; }

如果需要使用变量，同时又要确保 /不做除法运算而是完整地编译到 CSS 文件中起分隔的作用，只需要用 #{ }插值语句将变量包裹。

p {

$font-size: 12px;

$line-height: 30px;

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

编译为

p {

font: 12px/30px; }

#### 6.4.2. 颜色值运算 (Color Operations)

颜色值的运算是分段计算进行的，也就是分别计算红色，绿色，以及蓝色的值：

p {

color: #010203 + #040506;}

计算 01 + 04 = 0502 + 05 = 0703 + 06 = 09，然后编译为

p {

color: #050709; }

通常，使用 [color functions](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \t "http://www.w3cplus.com/preprocessor/_blank) 比计算颜色值更方便一些。

数字与颜色值之间也可以进行算数运算，同样也是分段计算的，比如

p {

color: #010203 \* 2;}

计算 01 \* 2 = 0202 \* 2 = 0403 \* 2 = 06，然后编译为

p {

color: #020406; }

需要注意的是，如果颜色值包含 alpha channel（rgba 或 hsla 两种颜色值），必须拥有相等的 alpha 值才能进行运算，因为算术运算不会作用于 alpha 值。

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); }

颜色值的 alpha channel 可以通过 [opacify](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \l "opacify-instance_method" \t "http://www.w3cplus.com/preprocessor/_blank) 或 [transparentize](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \l "transparentize-instance_method" \t "http://www.w3cplus.com/preprocessor/_blank) 两个函数进行调整。

$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 滤镜要求所有的颜色值包含 alpha 层，而且格式必须固定 #AABBCCDD，使用 ie\_hex\_str函数可以很容易地将颜色转化为 IE 滤镜要求的格式。

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

#### 6.4.3. 字符串运算 (String Operations)

+可用于连结字符串

p {

cursor: e + -resize;}

编译为

p {

cursor: e-resize; }

注意，如果有引号字符串（+左侧）连结无引号字符串，运算结果是有引号的，反过来，无引号字符串（+左侧）连结有引号字符串，运算结果则没有引号。

p:before {

content: "Foo " + Bar;

font-family: sans- + "serif";}

编译为

p:before {

content: "Foo Bar";

font-family: sans-serif; }

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 #{ }插值语句可以被用来添加动态的值：

p:before {

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

编译为

p:before {

content: "I ate 15 pies!"; }

空的值被视作插入了空字符串：

$value: null;p:before {

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

编译为

p:before {

content: "I ate pies!"; }

#### 6.4.4. 布尔运算 (Boolean Operations)

SassScript 支持布尔型的 andor以及 not运算。

#### 6.4.5. 值列表运算 (List Operations)

值列表不支持任何运算方式，只能使用 [list functions](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \l "list-functions" \t "http://www.w3cplus.com/preprocessor/_blank) 控制。

### 6.5. 圆括号 (Parentheses)

圆括号可以用来影响运算的顺序：

p {

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

编译为

p {

width: 7em; }

### 6.6. 函数 (Functions)

SassScript 定义了多种函数，有些甚至可以通过普通的 CSS 语句调用：

p {

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

编译为

p {

color: #ff0000; }

#### 6.6.1. 关键词参数 (Keyword Arguments)

Sass 函数允许使用关键词参数 (keyword arguments)，上面的例子也可以写成：

p {

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

虽然不够简明，但是阅读起来会更方便。关键词参数给函数提供了更灵活的接口，以及容易调用的参数。关键词参数可以打乱顺序使用，如果使用默认值也可以省缺，另外，参数名被视为变量名，下划线、短横线可以互换使用。

通过 [Sass::Script::Functions](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \t "http://www.w3cplus.com/preprocessor/_blank) 查看完整的 Sass 函数列表，参数名，以及如何自定义函数。

### 6.7. 插值语句 #{ }(Interpolation: #{ })

通过 #{ }插值语句可以在选择器或属性名中使用变量：

$name: foo;$attr: border;p.*#{$name} {*

*#{$attr}-color: blue;*}

编译为

p.foo {

border-color: blue; }

#{ }插值语句也可以在属性值中插入 SassScript，大多数情况下，这样可能还不如使用变量方便，但是使用 #{ }可以避免 Sass 运行运算表达式，直接编译 CSS。

p {

$font-size: 12px;

$line-height: 30px;

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

编译为

p {

font: 12px/30px; }

### 6.8. 变量定义 !default(Variable Defaults: !default)

可以在变量的结尾添加 !default给一个未通过 !default声明赋值的变量赋值，此时，如果变量已经被赋值，不会再被重新赋值，但是如果变量还没有被赋值，则会被赋予新的值。

$content: "First content";$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"; }

变量是 null 空值时将视为未被 !default赋值。

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

*#main {*

content: $content;}

编译为

#main {

content: "Non-null content"; }

## **7. @-Rules 与指令 (@-Rules and Directives)**

Sass 支持所有的 CSS3 @-Rules，以及 Sass 特有的 “指令”（directives）。这一节会详细解释，更多资料请查看 [控制指令 (control directives)](http://www.w3cplus.com/preprocessor/sass-chinese-reference.html" \l "8" \t "http://www.w3cplus.com/preprocessor/_blank) 与 [混合指令 (mixin directives)](http://www.w3cplus.com/preprocessor/sass-chinese-reference.html" \l "9" \t "http://www.w3cplus.com/preprocessor/_blank) 两个部分。

### 7.1. @import

Sass 拓展了 @import的功能，允许其导入 SCSS 或 Sass 文件。被导入的文件将合并编译到同一个 CSS 文件中，另外，被导入的文件中所包含的变量或者混合指令 (mixin) 都可以在导入的文件中使用。

Sass 在当前地址，或 Rack, Rails, Merb 的 Sass 文件地址寻找 Sass 文件，如果需要设定其他地址，可以用 :load\_paths选项，或者在命令行中输入 --load-path命令。

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。

* 文件拓展名是 .css；
* 文件名以 http://开头；
* 文件名是 url()；
* @import包含 media queries。

如果不在上述情况内，文件的拓展名是 .scss或 .sass，则导入成功。没有指定拓展名，Sass 将会试着寻找文件名相同，拓展名为 .scss或 .sass的文件并将其导入。

@import "foo.scss";

或

@import "foo";

都会导入文件 foo.scss，但是

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

编译为

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

Sass 允许同时导入多个文件，例如同时导入 rounded-corners 与 text-shadow 两个文件：

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

导入文件也可以使用 #{ }插值语句，但不是通过变量动态导入 Sass 文件，只能作用于 CSS 的 url()导入方式：

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

编译为

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

#### 7.1.1. 分音 (Partials)

如果需要导入 SCSS 或者 Sass 文件，但又不希望将其编译为 CSS，只需要在文件名前添加下划线，这样会告诉 Sass 不要编译这些文件，但导入语句中却不需要添加下划线。

例如，将文件命名为 \_colors.scss，便不会编译 \_colours.css文件。

@import "colors";

上面的例子，导入的其实是 \_colors.scss文件

注意，不可以同时存在添加下划线与未添加下划线的同名文件，添加下划线的文件将会被忽略。

#### 7.1.2. 嵌套 @import

大多数情况下，一般在文件的最外层（不在嵌套规则内）使用 @import，其实，也可以将 @import嵌套进 CSS 样式或者@media中，与平时的用法效果相同，只是这样导入的样式只能出现在嵌套的层中。

假设 example.scss 文件包含以下样式：

.example {

color: red;}

然后导入到 #main样式内

*#main {*

@import "example";}

将会被编译为

#main .example {

color: red;}

Directives that are only allowed at the base level of a document, like @mixin or @charset, are not allowed in files that are @imported in a nested context. 这一句不理解

不可以在混合指令 (mixin) 或控制指令 (control directives) 中嵌套 @import。

### 7.2. @media

Sass 中 @media指令与 CSS 中用法一样，只是增加了一点额外的功能：允许其在 CSS 规则中嵌套。如果 @media嵌套在 CSS 规则内，编译时，@media将被编译到文件的最外层，包含嵌套的父选择器。这个功能让 @media用起来更方便，不需要重复使用选择器，也不会打乱 CSS 的书写流程。

.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 queries 允许互相嵌套使用，编译时，Sass 自动添加 and

@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：

$media: screen;$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; } }

### 7.3. @extend

在设计网页的时候常常遇到这种情况：一个元素使用的样式与另一个元素完全相同，但又添加了额外的样式。通常会在 HTML 中给元素定义两个 class，一个通用样式，一个特殊样式。假设现在要设计一个普通错误样式与一个严重错误样式，一般会这样写：

<div class="error seriousError">

Oh no! You've been hacked!</div>

样式如下

.error {

border: 1px #f00;

background-color: #fdd;}.seriousError {

border-width: 3px;}

麻烦的是，这样做必须时刻记住使用 .seriousError时需要参考 .error的样式，带来了很多不变：智能比如加重维护负担，导致 bug，或者给 HTML 添加无语意的样式。使用 @extend可以避免上述情况，告诉 Sass 将一个选择器下的所有样式继承给另一个选择器。

.error {

border: 1px *#f00;*

background-color: *#fdd;*}.seriousError {

@extend .error;

border-width: 3px;}

上面代码的意思是将 .error下的所有样式继承给 .seriousError，border-width: 3px;是单独给.seriousError设定特殊样式，这样，使用 .seriousError的地方可以不再使用 .error。

其他使用到 .error的样式也会同样继承给 .seriousError，例如，另一个样式 .error.intrusion使用了hacked.png做背景，<div class="seriousError intrusion">也同样会使用 hacked.png背景。

.error.intrusion {

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

#### 7.3.1. How it Works

@extend的作用是将重复使用的样式 (.error) 延伸 (extend) 给需要包含这个样式的特殊样式（.seriousError），刚刚的例子：

.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会很聪明地避免无谓的重复，.seriousError.seriousError将编译为 .seriousError，不能匹配任何元素的选择器（比如 #main#footer）也会删除。

#### 7.3.2. 延伸复杂的选择器 (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：

.hoverlink {

@extend a:hover;}

同 class 元素一样，a:hover的样式将继承给 .hoverlink。

.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，所有 a:hover的样式将继承给 .hoverlink，包括其他使用到 a:hover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; }

#### 7.3.3. 多重延伸 (Multiple Extends)

同一个选择器可以延伸给多个选择器，它所包含的属性将继承给所有被延伸的选择器：

.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 .seriousError将包含 .error与 .attention下的所有样式，这时，后定义的样式享有优先权：.seriousError的背景颜色是 #ff0而不是 #fdd，因为 .attention在 .error之后定义。

多重延伸可以使用逗号分隔选择器名，比如 @extend .error, .attention;与 @extend .error;@extend.attention有相同的效果。

#### 7.3.4. 继续延伸 (Chaining Extends)

当一个选择器延伸给第二个后，可以继续将第二个选择器延伸给第三个，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，每个 .seriousError选择器将包含 .error的样式，而 .criticalError不仅包含 .seriousError的样式也会同时包含 .error的所有样式，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%; }

#### 7.3.5. 选择器列 (Selector Sequences)

暂时不可以将选择器列 (Selector Sequences)，比如 .foo .bar或 .foo + .bar，延伸给其他元素，但是，却可以将其他元素延伸给选择器列：

*#fake-links .link {*

@extend a;}

a {

color: blue;

&:hover {

text-decoration: underline;

}}

编译为

a, #fake-links .link {

color: blue; }

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

text-decoration: underline; }

#### 7.3.5.1 合并选择器列 (Merging Selector Sequences)

有时会遇到复杂的情况，比如选择器列中的某个元素需要延伸给另一个选择器列，这种情况下，两个选择器列需要合并，比如：

*#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：

*#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; }

#### 7.3.6. @extend-Only 选择器 (@extend-Only Selectors)

有时，需要定义一套样式并不是给某个元素用，而是只通过 @extend指令使用，尤其是在制作 Sass 样式库的时候，希望 Sass 能够忽略用不到的样式。

如果使用普通的 CSS 规则，最后会编译出很多用不到的样式，也容易与其他样式名冲突，所以，Sass 引入了“占位符选择器” (placeholder selectors)，看起来很像普通的 id或 class选择器，只是 #或 .被替换成了 %。可以像 class 或者 id 选择器那样使用，当它们单独使用时，不会被编译到 CSS 文件中。

*// This ruleset won't be rendered on its own.*#context a%extreme {

color: blue;

font-weight: bold;

font-size: 2em;}

占位符选择器需要通过延伸指令使用，用法与 class 或者 id 选择器一样，被延伸后，占位符选择器本身不会被编译。

.notice {

@extend %extreme;}

编译为

#context a.notice {

color: blue;

font-weight: bold;

font-size: 2em; }

#### 7.3.7. !optional声明 (The !optionalFlag)

如果 @extend失败会收到错误提示，比如，这样写 a.important {@extend .notice}，当没有 .notice选择器时，将会报错，只有 h1.notice包含 .notice时也会报错，因为 h1与 a冲突，会生成新的选择器。

如果要求 @extend不生成新选择器，可以通过 !optional声明达到这个目的，例如：

a.important {

@extend .notice !optional;}

#### 7.3.8 在指令中延伸 (@extend in Directives)

在指令中使用 @extend时（比如在 @media中）有一些限制：Sass 不可以将 @media层外的 CSS 规则延伸给指令层内的 CSS，这样会生成大量的无用代码。也就是说，如果在 @media（或者其他 CSS 指令）中使用 @extend，必须延伸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;

}}

但不可以这样：

.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，这样就可以在任何指令中使用延伸功能，不再受限制了。

@debug与 @warn两条指令不太熟悉，暂时未翻译，请查看 [原文档](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html" \l "id8" \t "http://www.w3cplus.com/preprocessor/_blank)

## **8. 控制指令 (Control Directives)**

SassScript 提供了一些基础的控制指令，比如在满足条件的时候引用样式，或者在一定范围内重复输出样式。****请注意，控制指令是一种高级功能，不建议频繁使用。****控制指令主要与混合指令 (mixin) 配合使用，尤其是在 [Compass](http://compass-style.org/" \t "http://www.w3cplus.com/preprocessor/_blank) 等样式库中。

### 8.1. @if

@if指令内包含多个条件语句，当返回值不是 false或者 null时，条件成立，输出结果：

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声明后面可跟多个 @else if声明，或者一个 @else声明。如果 @if声明失败，Sass 将逐条执行 @else if声明，如果仍然失败，执行 @else声明，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; }

### 8.2. @for

@for指令在限制的范围内重复输出一系列的样式，每次按要求（变量的值）对输出结果做出变动。这个指令包含两种格式：@for $var from <start> through <end>，或者 @for $var from <start> to <end>，区别在于through与 to的含义：当使用 through时，条件范围包含 <start>与 <end>的值，而使用 to时条件范围只包含<start>的值不包含 <end>的值。另外，$var可以是任何变量，比如 $i；<start>和 <end>必须是整数值。

@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; }

### 8.3. @each

@each指令的格式是 $var in <list>, $var可以是任何变量名，比如 $length或者 $name，而 <list>是一连串的值，也就是值列表。

@each将变量 $var作用于值列表中的每一个项目，然后输出结果，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'); }

### 8.4. @while

@while指令重复输出样式直到返回结果为 false。这样可以做出比 @for指令更复杂的循环，只是很少会用到。例如：

$i: 6;@while $i > 0 {

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

$i: $i - 2;}

.item-6 {

width: 12em; }

.item-4 {

width: 8em; }

.item-2 {

width: 4em; }

## **9. 混合指令 (Mixin Directives)**

混合指令（Mixin）用于定义可重复使用的样式，避免了使用无语意的 class，比如 .float-left。混合指令可以包含所有的 CSS 规则，绝大部分 Sass 规则，甚至通过参数功能引入变量，输出多样化的样式。

### 9.1. 定义混合指令 @mixin(Defining a Mixin: @mixin)

混合指令的用法是在 @mixin后添加名称与样式，比如名为 large-text的混合通过下面的代码定义：

@mixin large-text {

font: {

family: Arial;

size: 20px;

weight: bold;

}

color: #ff0000;}

混合也需要包含选择器和属性，甚至可以用 &引用父选择器：

@mixin clearfix {

display: inline-block;

&:after {

content: ".";

display: block;

height: 0;

clear: both;

visibility: hidden;

}

\* html & { height: 1px }}

### 9.2. 引用混合样式 @include(Including a Mixin: @include)

使用 @include指令引用混合样式，格式是在其后添加混合名称，以及需要的参数（可选）：

.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; }

也可以在最外层引用混合样式，不会直接定义属性，也不可以使用父选择器。

@mixin silly-links {

a {

color: blue;

background-color: red;

}}@include silly-links;

编译为

a {

color: blue;

background-color: red; }

混合样式中也可以包含其他混合样式，比如

@mixin compound {

@include highlighted-background;

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

混合样式中应该只定义后代选择器，这样可以安全的导入到文件的任何位置。

### 9.3. 参数 (Arguments)

参数用于给混合指令中的样式设定变量，并且赋值使用。在定义混合指令的时候，按照变量的格式，通过逗号分隔，将参数写进圆括号里。引用指令时，按照参数的顺序，再将所赋的值对应写进括号：

@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; }

混合指令也可以使用给变量赋值的方法给参数设定默认值，然后，当这个指令被引用的时候，如果没有给参数赋值，则自动使用默认值：

@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; }

#### 9.3.1. 关键词参数 (Keyword Arguments)

混合指令也可以使用关键词参数，上面的例子也可以写成：

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

虽然不够简明，但是阅读起来会更方便。关键词参数给函数提供了更灵活的接口，以及容易调用的参数。关键词参数可以打乱顺序使用，如果使用默认值也可以省缺，另外，参数名被视为变量名，下划线、短横线可以互换使用。

#### 9.3.2. 参数变量 (Variable Arguments)

有时，不能确定混合指令需要使用多少个参数，比如一个关于 box-shadow的混合指令不能确定有多少个 'shadow' 会被用到。这时，可以使用参数变量 …声明（写在参数的最后方）告诉 Sass 将这些参数视为值列表处理：

@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

.shadowed {

-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;}

参数变量也可以用在引用混合指令的时候 (@include)，与平时用法一样，将一串值列表中的值逐条作为参数引用：

@mixin colors($text, $background, $border) {

color: $text;

background-color: $background;

border-color: $border;}$values: *#ff0000, #00ff00, #0000ff;*.primary {

@include colors($values...);}

编译为

.primary {

color: #ff0000;

background-color: #00ff00;

border-color: #0000ff;}

You can use variable arguments to wrap a mixin and add additional styles without changing the argument signature of the mixin. If you do so, even keyword arguments will get passed through to the wrapped mixin. For example:

@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);*}

上面注释内的意思是：$width参数将会传递给 stylish-mixin作为关键词。

### 9.4. 向混合样式中导入内容 (Passing Content Blocks to a Mixin)

在引用混合样式的时候，可以先将一段代码导入到混合指令中，然后再输出混合样式，额外导入的部分将出现在 @content标志的地方：

@mixin apply-to-ie6-only {

\* html {

@content;

}}@include apply-to-ie6-only {

*#logo {*

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

}}

编译为

\* html #logo {

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

****为便于书写，**@mixin**可以用** =**表示，而** @include**可以用** +**表示****，所以上面的例子可以写成：

=apply-to-ie6-only

\* html

@content

+apply-to-ie6-only

*#logo*

background-image: url(/logo.gif)

****注意：**** 当 @content在指令中出现过多次或者出现在循环中时，额外的代码将被导入到每一个地方。

#### 9.4.1. Variable Scope and Content Blocks

The block of content passed to a mixin are evaluated in the scope where the block is defined, not in the scope of the mixin. This means that variables local to the mixin cannot be used within the passed style block and variables will resolve to the global value:

$color: white;@mixin colors($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;}

Additionally, this makes it clear that the variables and mixins that are used within the passed block are related to the other styles around where the block is defined. For example:

*#sidebar {*

$sidebar-width: 300px;

width: $sidebar-width;

@include smartphone {

width: $sidebar-width / 3;

}}

## **10. 函数指令 (Function Directives)**

Sass 允许自定义函数作用于任何值或者字符串，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); }*

结果是

#sidebar {

width: 240px; }

函数可以使用任何变量，或者像混合指令一样使用参数。一个函数通常包含很多连结在一起的指令，通过 @return设定函数的返回值。

自定义的函数也可以使用关键词参数，上面的例子可以这样写：

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

建议在自定义函数前添加前缀，避免与 Sass 的默认函数冲突，也可以让其他人在阅读 Sass 文件时知道这是自定义函数。

## **11. 输出样式 (Output Style)**

Sass 支持多种输出样式，满足不同的需求。通过 :style选项设定输出样式，或者在命令行中使用 --style命令。

### 11.1. :nested

Nested （嵌套）样式是默认的 Sass 输出方式，这种样式清楚地反映了 CSS 与 HTML 的结构。选择器与属性等单独占用一行，缩进量与 Sass 文件中一致，每行的缩进量反映了其在嵌套规则内的层数。当阅读比较大的 CSS 文件时，这种样式可以很容易分析文件的结构。

#main {

color: #fff;

background-color: #000; }

#main p {

width: 10em; }

.huge {

font-size: 10em;

font-weight: bold;

text-decoration: underline; }

### 11.2. :expanded

Expanded 输出方式更像是手写的，选择器、属性等各占用一行，属性根据选择器缩进，而选择器不做任何缩进。

#main {

color: #fff;

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

width: 10em;}

.huge {

font-size: 10em;

font-weight: bold;

text-decoration: underline;}

### 11.3. :compact

Compact 输出方式比起上面两种占用的空间更少，每条 CSS 规则只占一行，包含其下的所有属性。嵌套的选择器在输出时紧邻在一起，与其他组通过空白行分隔。

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

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

### 11.4 :compressed

Compressed 输出方式删除无意义的空格与空白行，以及注释，将文件压缩到最小，同时也会做出其他调整，比如智能换用占空间最小的颜色表达方式。

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

## **12. 拓展 Sass**

Sass 提供了很多高级自定义功能，使用这些功能需要有良好的 Ruby 基础。

### 12.1. 自定义 Sass 函数

用户可以通过 Ruby API 自定义 Sass 函数，详细信息请查看 [source documentation](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \l "adding_custom_functions" \t "http://www.w3cplus.com/preprocessor/_blank)。

以下关于 Sass 缓存与自定义导入设置的内容暂时未翻译，请查看 [原文档](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html" \l "cache_stores" \t "http://www.w3cplus.com/preprocessor/_blank)