# Sass简介

## 什么是 CSS 预处理器？

**定义：**

CSS 预处理器定义了一种新的语言，其基本思想是，用一种专门的编程语言，为 CSS 增加了一些编程的特性，将 CSS 作为目标生成文件，然后开发者就只要使用这种语言进行编码工作。

通俗的说，“CSS 预处理器用一种专门的编程语言，进行 Web 页面样式设计，然后再编译成正常的 CSS 文件，以供项目使用。CSS 预处理器为 CSS 增加一些编程的特性，无需考虑浏览器的兼容性问题”，例如你可以在 CSS 中使用**变量**、**简单的逻辑程序**、**函数（**如右侧代码编辑器中就使用了变量**$color）**等等在编程语言中的一些基本特性，可以让你的 CSS **更加简洁**、**适应性更强**、**可读性更佳**，**更易于代码的维护**等诸多好处。

**其它 CSS 预处理器语言：**

CSS 预处理器技术已经非常的成熟，而且也涌现出了很多种不同的 **CSS 预处理器语言**，比如说：

* Sass（SCSS）
* LESS
* Stylus
* Turbine
* Swithch CSS
* CSS Cacheer
* DT CSS

如此之多的 CSS 预处理器，那么“我应该选择哪种 CSS 预处理器？”也相应成了最近网上的一大热门话题，在 **Linkedin**、**Twitter**、**CSS-Trick**、**知乎**以及各大技术论坛上，很多人为此争论不休。相比过去我们对是否应该使用 CSS 预处理器的话题而言，这已经是很大的进步了。

到目前为止，在众多优秀的 CSS 预处理器语言中就属 **Sass**、**LESS**和 **Stylus**最优秀，讨论的也多，对比的也多。本教程将着重向大家介绍 CSS 预处理器中的 Sass。相信前端开发工程师会喜欢的。

## 什么是 Sass？

[Sass 官网](http://sass-lang.com/)上是这样描述 Sass 的：

Sass 是一门高于 CSS 的元语言，它能用来清晰地、结构化地描述文件样式，有着比普通 CSS 更加强大的功能。

Sass 能够提供更简洁、更优雅的语法，同时提供多种功能来创建可维护和管理的样式表。

**Sass 前世今生：**

Sass 是最早的 CSS 预处理语言，有比 LESS 更为强大的功能，不过其一开始的缩进式语法（Sass 老版本语法，后面课程会详细介绍 ）并不能被大众接受，不过由于其强大的功能和 Ruby on Rails 的大力推动，还是有很多开发者选择了 Sass。

Sass 是采用 **Ruby**语言编写的一款 CSS 预处理语言，它诞生于2007年，是最大的成熟的 CSS 预处理语言。最初它是为了配合 HAML（一种缩进式 HTML 预编译器）而设计的，因此有着和 HTML 一样的缩进式风格。

**为什么早期不如 LESS 普及？**

虽然缩进式风格可以有效缩减代码量，强制规范编码风格，但它一方面并不为大多数程序接受，另一方面无法兼容已有的 CSS 代码。这也是 Sass 虽然出现得最早，但远不如 LESS 普及的原因

## Sass 和 SCSS 有什么区别？

Sass 和 SCSS 其实是同一种东西，我们平时都称之为 Sass，两者之间不同之处有以下两点：

1. 文件扩展名不同，Sass 是以“.sass”后缀为扩展名，而 SCSS 是以“.scss”后缀为扩展名
2. 语法书写方式不同，Sass 是以严格的缩进式语法规则来书写，不带大括号({})和分号(;)，而 SCSS 的语法书写和我们的 CSS 语法书写方式非常类似。

先来看一个示例：

**Sass 语法**

$font-stack: Helvetica, sans-serif //定义变量

$primary-color: #333 //定义变量

body

font: 100% $font-stack

color: $primary-color

**SCSS 语法**

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body {

font: 100% $font-stack;

color: $primary-color;

}

**编译出来的 CSS**

body {

font: 100% Helvetica, sans-serif;

color: #333;

}

**提示：**小伙伴们可以看出来，我们的右侧代码使用的是**SCSS 语法**方式（我们的代码编辑器**不支持** **Sass 语法**方式噢！）。

### 任务

编写第一个".scss"文件，在编辑器的第一行和第二行定义一个变量，并在body中调用对应变量来设置body大小尺寸？

body {

   width: 200px;

  height: 300px;

}

## Sass/SCSS 和纯 CSS 写法差很多吗？

写法差很多吗？这是很多初学者会问的一个问题。那么借此机会简单了解一下。

**Sass 和 CSS 写法有差别：**

Sass 和 CSS 写法的确存在一定的差异，由于 Sass 是基于 Ruby 写出来，所以其延续了 Ruby 的书写规范。在书写 Sass 时不带有大括号和分号，其主要是依靠严格的缩进方式来控制的。如：

Sass写法：

body

color: #fff

background: #f36

而在 CSS 我们是这样书写：

body**{**

color:#fff;

background:#f36;

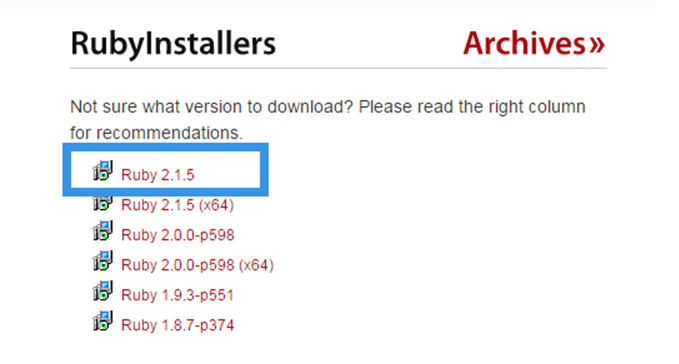
**}**

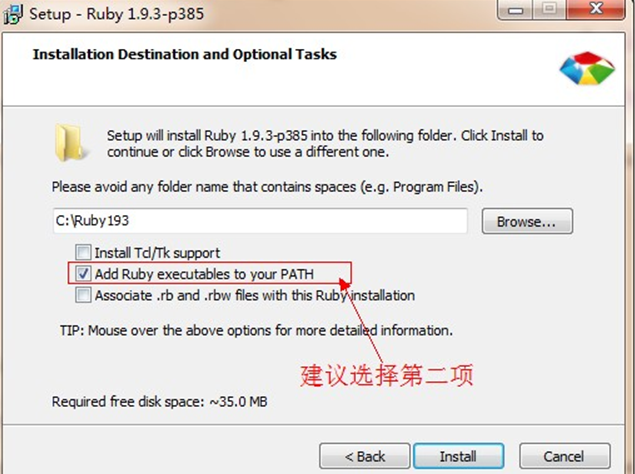
**SCSS 和 CSS 写法无差别：**

SCSS 和 CSS 写法无差别，这也是 Sass 后来越来越受大众喜欢原因之一。简单点说，把你现有的“.css”文件直接修改成“.scss”即可使用。

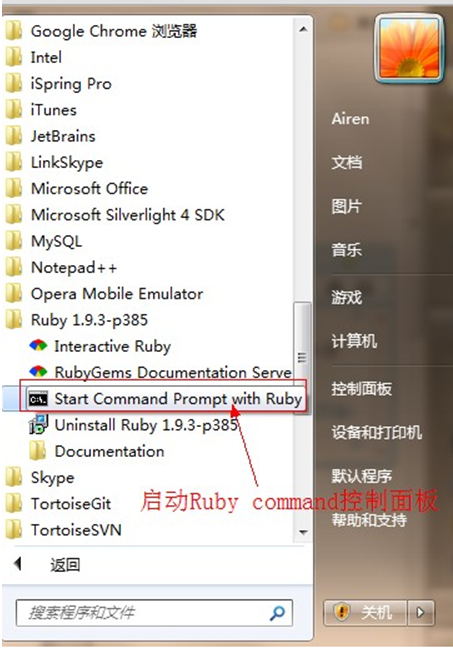
## Sass安装（windows版）

在 Windows 平台下安装 Ruby 需要先有 Ruby 安装包，大家可以到 Ruby 的官网（http://rubyinstaller.org/downloads）下载对应需要的 Ruby 版本。

[](https://img.mukewang.com/54f560af00017cf106790364.jpg)

Ruby 安装文件下载好后，可以按应用软件安装步骤进行安装 Ruby。在安装过程中，个人建议将其安装在 C 盘下，在安装过程中选择第二个选项（不选中，就会出现编译时找不到Ruby环境的情况），如下图所示：  
  
[](https://img.mukewang.com/54f561190001531)

Ruby 安装完成后，在开始菜单中找到新安装的 Ruby，并启动 Ruby 的 Command 控制面板，如下图所示：

[](https://img.mukewang.com/54f5615f00011cb604530648.jpg)

当你的电脑中安装好 Ruby 之后，接下来就可以安装 Sass 了。同样的在windows下安装 Sass 有多种方法。但这几种方法都是非常的简单，只需要在你的命令终端输入一行命令即可。

**1、通过命令安装 Sass**

打开电脑的命令终端，输入下面的命令：

gem install sass

提醒一下，在使用 Mac 的同学，可能需要在上面的命令前加上"sudo"，才能正常安装：

sudo gem install sass

如果上面的方法没有安装成功，可以使用下面的两种方法。

**2、通过 Compass 来安装 Sass**

除了使用 gem 命令来安装 Sass 之外，还可以通过安装 compass 来安装 Sass，因为 Compass 是基于 Sass 开发的一个框架。也就是说，你安装了 Compass，也就同时安装好了 Sass。

同样的在你的命令终端输入下面的命令：

sudo gem install sass

执行完上面的命令之后，就开始安装 Compass 和 Sass。

注：Compass 是一个成熟的、基于 Sass 开发的一个框架，这里面集成了很多写好的 mixins 和 Sass 函数。不过在此暂不做过多阐述。

**3、本地安装 Sass**

由于有时候直接使用上面的命令安装会让你无法正常实现安装（网络受限原因），当碰到这种情况之时，那么安装需要特殊去处理，可以通过下面的方法来实现 Sass 的正常安装：

可以到 [Rubygems](http://rubygems.org/)(http://rubygems.org/) 网站上将 [Sass 的安装包](http://rubygems.org/gems/sass)（http://rubygems.org/gems/sass）下载下来，然后在命令终端输入：

gem install <把下载的安装包拖到这里>

直接回车即可安装成功。

注：在 iOSX 系统平台，可以直接将下载的安装包拖到 "gem install" 后面，如果在是 Windows 系统，需要手功输入安装的文件路径。

**4、淘宝 RubyGems 镜像安装 Sass**

除了下载 Sass 安装包到本地安装之外，碰到网络原因无法安装时还可以使用淘宝 RubyGems 镜像安装 Sass。只是我们需要通过 gem sources 命令来配置源，先移除默认的 https://rubygems.org 源，然后添加淘宝的源 https://ruby.taobao.org：

第一步：移动默认的源

gem sources --remove https://rubygems.org/

第二步：指定淘宝的源

gem sources -a https://ruby.taobao.org/

第三步：查看指定的源是不是淘宝源

gem sources -l

返回结果如下：

\*\*\* CURRENT SOURCES \*\*\*

https://ruby.taobao.org

请确保只有 ruby.taobao.org。如果无误之后，执行下面的命令：

gem install sass

## 查测 Sass 及更新

通过上面的几种方法都可以安装 Sass，但是，我们要如何确认自己是否安装 Sass 成功了呢？其实很简单，只需要通过下面的命令即可：

sass -v

如果在你的命令终端能看到类似这样的信息就表示你的电脑安装 Sass 已成功。也就是说可以正常的使用 Sass 了。

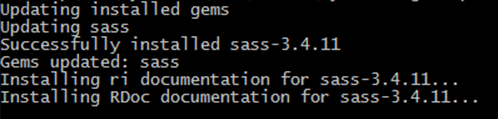
[https://img.mukewang.com/54f5624c0001c2ed02990026.jpg](https://img.mukewang.com/54f5624c0001c2ed02990026.jpg)

**更新 Sass**

维护 Sass 的团队会不断的为 Sass 添加新的功能，那么如何确保自己已安装的 Sass 也具有这些新的功能特性呢？不会是卸载了重新安装吧（虽然安装也就是一个命令的事情）？ 其实不需要这么麻烦，只需要在命令终端执行：

gem update sass

这个时候你看到类似下面这样的信息，表示你的 Sass 已更新到最新版本。

[](https://img.mukewang.com/54f562a5000134d905000119.jpg)

## 卸载（删除）Sass

在常期使用的时候难免会碰到无法解决的问题，有时候可能需要卸载 Sass，然后再重新安装 Sass。那么怎么卸载 Sass 呢？

其实他也就是一句命令的事情：

gem uninstall sass

这样就卸载了 Sass ，但这行命令基本上是使用不上

# Sass的语法格式及编译调试

## Sass 语法格式

这里说的 Sass 语法是 Sass 的最初语法格式，他是通过 tab 键控制缩进的一种语法规则，而且这种缩进要求非常严格。另外其不带有任何的分号和大括号。常常把这种格式称为 Sass 老版本，其文件名以“.sass”为扩展名。  
来看一个 Sass 语法格式的简单示例。假设我们有一段这样的 CSS 代码：

body {

font: 100% Helvetica, sans-serif;

color: #333;

}

现在用 Sass 的语法格式来编写：

$font-stack: Helvetica, sans-serif

$primary-color: #333

body

font: 100% $font-stack

color: $primary-color

在整个 Sass 代码中，我们没看到类似 CSS 中的大括号和分号。  
注：这种语法格式对于前端人员都不太容易接受，而且容易出错。

**二、SCSS语法格式**

SCSS 是 Sass 的新语法格式，从外形上来判断他和 CSS 长得几乎是一模一样，代码都包裹在一对大括号里，并且末尾结束处都有一个分号。其文件名格式常常以“.scss”为扩展名。

同样的这段 CSS 代码：

body {

font: 100% Helvetica, sans-serif;

color: #333;

}

我们使用 SCSS 语法格式将按下面这样来书写：

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body {

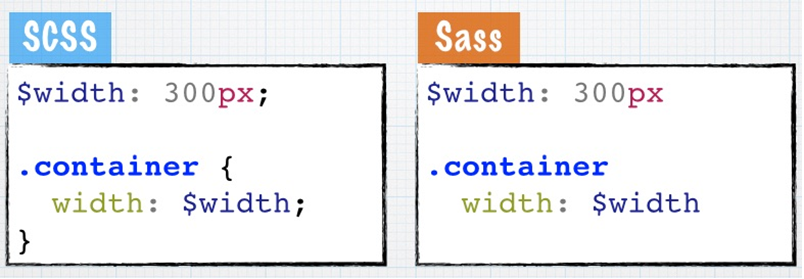
font: 100% $font-stack;

color: $primary-color;

}

这样的语法格式对于从事前端工作的同学来说更易于接受，这也是 SCSS 为什么更被前端人员青眯的原因。

不管是 Sass 的语法格式还是 SCSS 的语法格式，他们的功能都是一样的，不同的是其书写格式和文件扩展名不同，来看一个简单的对比图：

[](https://img.mukewang.com/54f549e4000112aa08040279.jpg)

正因为如此，有不少同学使用 Sass 新的语法规则，而文件扩展名依旧使用的是“.sass”，这也就造成血案了，编译时说编译不出来。在此特别提醒新同学：“.sass”只能使用 Sass 老语法规则（缩进规则），“.scss”使用的是 Sass 的新语法规则，也就是 SCSS 语法规则（类似 CSS 语法格式）。

**特别提醒：本系列教程后面采用的语法格式都将使用的是 SCSS 语法格式。**

## Sass 编译

常常有人会问，使用 Sass 进行开发，那么是不是直接通过“<link>”引用“.scss”或“.sass”文件呢？

那么这里告诉大家，在项目中还是引用“.css”文件，Sass 只不过是做为一个预处理工具，提前帮你做事情，只有你需要时候，他才有攻效。

这样一来，也就有了这章需要介绍的内容—— Sass 的编译。因为 Sass 开发之后，要让 Web 页面能调用 Sass 写好的东西，就得有这么一个过程，这个过程就称之为 Sass 编译过程。Sass 的编译有多种方法：

* 命令编译
* GUI工具编译
* 自动化编译

## [Sass]命令编译

命令编译是指使用你电脑中的命令终端，通过输入 Sass 指令来编译 Sass。这种编译方式是最直接也是最简单的一种方式。因为只需要在你的命令终端输入：

**单文件编译：**

sass <要编译的Sass文件路径>/style.scss:<要输出CSS文件路径>/style.css

这是对一个单文件进行编译，如果想对整个项目所有 Sass 文件编译成 CSS 文件，可以这样操作：

**多文件编译：**

sass sass/:css/

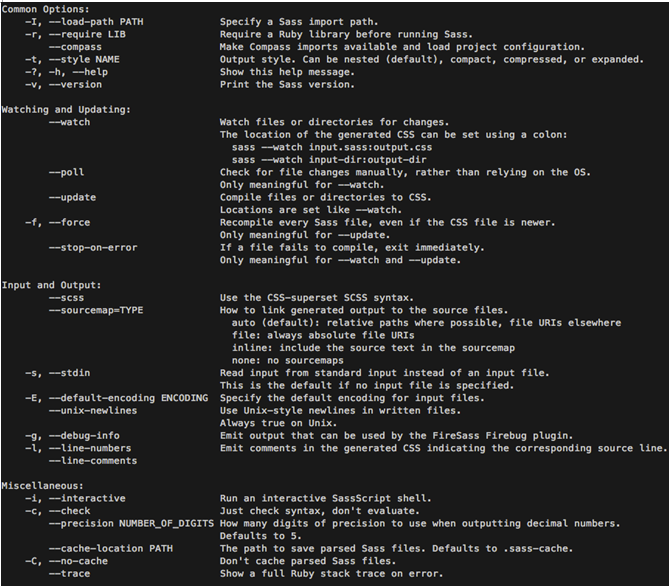
上面的命令表示将项目中“sass”文件夹中所有“.scss”(“.sass”)文件编译成“.css”文件，并且将这些 CSS 文件都放在项目中“css”文件夹中。

**缺点及解决方法：**

在实际编译过程中，你会发现上面的命令，只能一次性编译。每次个性保存“.scss”文件之后，都得重新执行一次这样的命令。如此操作太麻烦，其实还有一种方法，就是在编译 Sass 时，开启“**watch**”功能，这样只要你的代码进行任保修改，都能自动监测到代码的变化，并且给你直接编译出来：

sass --watch <要编译的Sass文件路径>/style.scss:<要输出CSS文件路径>/style.css

当然，使用 sass 命令编译时，可以带很多的参数：

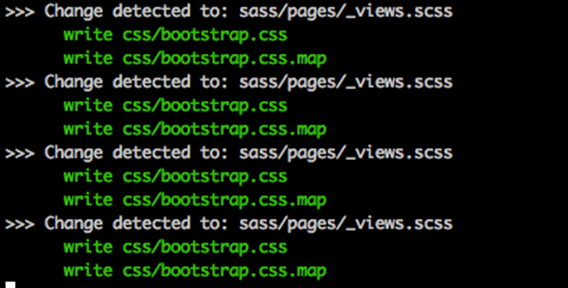
[](https://img.mukewang.com/54f54b7a00014d3006690586.jpg)

**watch 举例：**

来看一个简单的示例，假设我本地有一个项目，我要把项目中“bootstrap.scss”编译出“bootstrap.css”文件，并且将编译出来的文件放在“css”文件夹中，我就可以在我的命令终端中执行：

sass --watch sass/bootstrap.scss:css/bootstrap.css

一旦我的 bootstrap.scss 文件有任何修改，只要我重新保存了修改的文件，命令终端就能监测，并重新编译出文件：

[](https://img.mukewang.com/54f54bcb00016b1005680288.jpg)

## [Sass]GUI 界面工具编译

或许你会说，我一直讨厌使用命令来做事情，我喜欢那种能看得到的界面操作。那么你可以考虑使用 GUI 界面工具来对 Sass 进行编译。当然不同的 GUI 工具操作方法略有不同。如果在此也一一对编译的界面工具做详细的介绍。我们可能需要写一本书来介绍这些编译工具的操作了。所以我们这里做一下简单介绍，对于 GUI 界面编译工具，目前较为流行的主要有：

1. Koala (<http://koala-app.com/>)
2. Compass.app（<http://compass.kkbox.com/>）
3. Scout（<http://mhs.github.io/scout-app/>）
4. CodeKit（<https://incident57.com/codekit/index.html>）
5. Prepros（<https://prepros.io/>）

相比之下，我比较推荐使用以下两个：

* Koala (<http://www.w3cplus.com/preprocessor/sass-gui-tool-koala.html>)
* CodeKit (<http://www.w3cplus.com/preprocessor/sass-gui-tool-codekit.html>)

## [Sass]自动化编译

喜欢自动化研究的同学，应该都知道 **Grunt**和 **Gulp**这两个东东。如果您正在使用其中的任何一种，那么你也可以通过他们来配置 Sass 的编译。这里仅列出两个示例代码（具体情况要根据您的项目环境来做一定的修改，不建议生搬硬套，容易发生命案，呵呵。

**1、Grunt 配置 Sass 编译的示例代码**

module.exports = function(grunt) {

grunt.initConfig({

pkg: grunt.file.readJSON('package.json'),

sass: {

dist: {

files: {

'style/style.css' : 'sass/style.scss'

}

}

},

watch: {

css: {

files: '\*\*/\*.scss',

tasks: ['sass']

}

}

});

grunt.loadNpmTasks('grunt-contrib-sass');

grunt.loadNpmTasks('grunt-contrib-watch');

grunt.registerTask('default',['watch']);

}

想了解 **Grunt**同学请单击这里学习[《Grunt-beginner前端自动化工具》](http://www.imooc.com/learn/30)。

**2、Gulp 配置 Sass 编译的示例代码**

var gulp = require('gulp');

var sass = require('gulp-sass');

gulp.task('sass', function () {

gulp.src('./scss/\*.scss')

.pipe(sass())

.pipe(gulp.dest('./css'));

});

gulp.task('watch', function() {

gulp.watch('scss/\*.scss', ['sass']);

});

gulp.task('default', ['sass','watch']);

## [Sass]常见的编译错误

在编译 Sass 代码时常常会碰到一些错误，让编译失败。这样的错误有系统造成的也有人为造成的，但大部分都是人为过失引起编译失败。

而最为常见的一个错误就是字符编译引起的。在Sass的编译的过程中，是不是支持“GBK”编码的。所以在创建 Sass 文件时，就需要将文件编码设置为“utf-8”。

另外一个错误就是路径中的中文字符引起的。建议在项目中文件命名或者文件目录命名不要使用中文字符。而至于人为失误造成的编译失败，在编译过程中都会有具体的说明，大家可以根据编译器提供的错误信息进行对应的修改。

## [Sass]不同样式风格的输出方法

众所周知，每个人编写的 CSS 样式风格都不一样，有的喜欢将所有样式代码都写在同一行，而有的喜欢将样式分行书写。在 Sass 中编译出来的样式风格也可以按不同的样式风格显示。其主要包括以下几种样式风格：

1. **嵌套输出方式 nested**
2. **展开输出方式 expanded**
3. **紧凑输出方式 compact**
4. **压缩输出方式 compressed**

以上 4 种方法会在后面课程中详细讲解。

### [Sass]嵌套输出方式 nested

1、嵌套输出方式 nested

Sass 提供了一种嵌套显示 CSS 文件的方式。例如

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

display: block;

padding: 6px 12px;

text-decoration: none;

}

}

在编译的时候带上参数“ --style nested”:

sass --watch test.scss:test.css --style nested

编译出来的 CSS 样式风格：

nav ul {

margin: 0;

padding: 0;

list-style: none; }

nav li {

display: inline-block; }

nav a {

display: block;

padding: 6px 12px;

text-decoration: none; }

如下图所示：

[](https://img.mukewang.com/54f54dab00019a0504880296.jpg)

### [Sass]展开输出方式 expanded

2、嵌套输出方式 expanded

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

display: block;

padding: 6px 12px;

text-decoration: none;

}

}

在编译的时候带上参数“ --style expanded”:

sass --watch test.scss:test.css --style expanded

这个输出的 CSS 样式风格和 nested 类似，只是大括号在另起一行，同样上面的代码，编译出来：

nav ul {

margin: 0;

padding: 0;

list-style: none;

**}**

nav li {

display: inline-block;

**}**

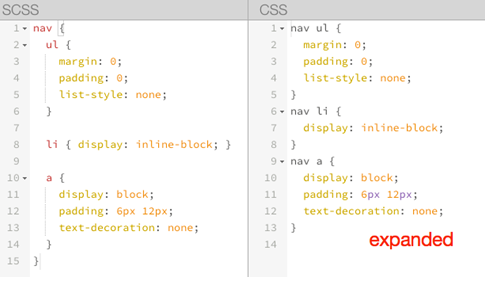
nav a {

display: block;

padding: 6px 12px;

text-decoration: none;

**}**

[](https://img.mukewang.com/54f54e2c0001c2c004850281.jpg)

### [Sass]紧凑输出方式 compact

2、嵌套输出方式 compact

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

display: block;

padding: 6px 12px;

text-decoration: none;

}

}

在编译的时候带上参数“ --style compact”:

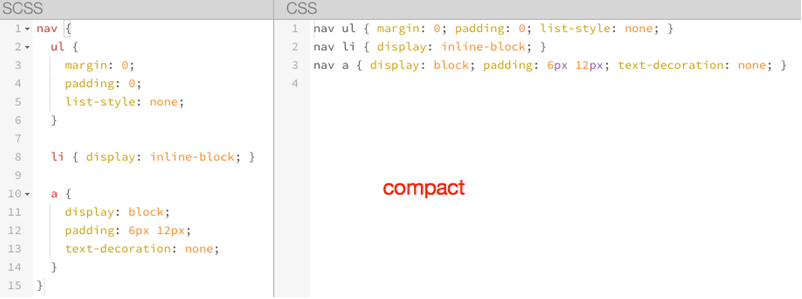
sass --watch test.scss:test.css --style compact

该方式适合那些喜欢单行 CSS 样式格式的朋友，编译后的代码如下：

nav ul { margin: 0; padding: 0; list-style: none; }

nav li { display: inline-block; }

nav a { display: block; padding: 6px 12px; text-decoration: none; }

[](https://img.mukewang.com/54f7b4bb00014f9908020299.jpg)

### [Sass]压缩输出方式 compressed

2、压缩输出方式 compressed

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

a {

display: block;

padding: 6px 12px;

text-decoration: none;

}

}

在编译的时候带上参数“ --style compressed”:

sass --watch test.scss:test.css --style compressed

压缩输出方式会去掉标准的 Sass 和 CSS 注释及空格。也就是压缩好的 CSS 代码样式风格：

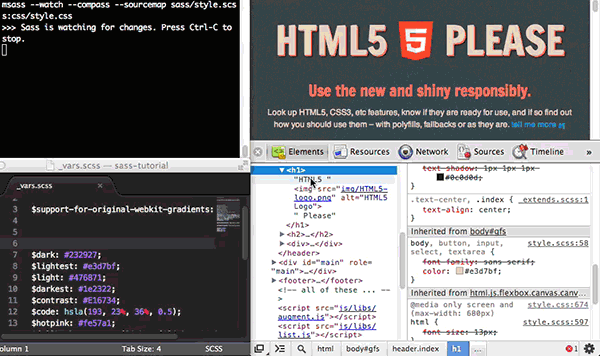
nav ul{margin:0;padding:0;list-style:none}nav li{display:inline-block}nav a{display:block;padding:6px 12px;text-decoration:none}

[](https://img.mukewang.com/54f5511d0001b5c206660401.jpg)

编译出来的CSS样式风格的选择完全是个人喜好问题，可以根据自己喜欢的风格选择参数。

一段时间之后，你实际上就不再需要写 CSS 代码了，只用写 Sass 代码。在这种情况下，你只需要设定输出格式为压缩格式，知道输出的 CSS 代码可以直接使用即可。

## Sass 的调试

[](https://img.mukewang.com/54f7b71d0001bb0b05050268.jpg)

（单击可放大）

Sass 调试一直以来都是一件头痛的事情，使用 Sass 的同学都希望能在浏览器中直接调试 Sass 文件，能找到对应的行数。值得庆幸的是，现在实现并不是一件难事，只要你的浏览器支持“sourcemap”功能即可。早一点的版本，需要在编译的时候添加“--sourcemap”  参数：

sass --watch --scss --sourcemap style.scss:style.css

在 Sass3.3 版本之上（我测试使用的版本是 3.4.7），不需要添加这个参数也可以：

sass --watch style.scss:style.css

在命令终端，你将看到一个信息：

>>> Change detected to: style.scss  
  write style.css  
  write style.css.map

这时你就可以像前面展示的 gif 图一样，调试你的 Sass 代码。

# Sass的基本特性-基础

## [Sass]声明变量

定义变量的语法：

在有些编程语言中（如，JavaScript）声明变量都是使用关键词“var”开头，但是在 Sass 不使用这个关键词，而是使用大家都喜欢的美元符号“$”开头。我想用一张图来解释，我一直坚信，一图胜千言万语：



上图非常清楚告诉了大家，Sass 的变量包括三个部分：

1. 声明变量的符号“$”
2. 变量名称
3. 赋予变量的值

来看一个简单的示例，假设你的按钮颜色可以给其声明几个变量：

$brand-primary : darken(#428bca, 6.5%) !default; // #337ab7

$btn-primary-color : #fff !default;

$btn-primary-bg : $brand-primary !default;

$btn-primary-border : darken($btn-primary-bg, 5%) !default;

如果值后面加上!default则表示默认值。

注：了解 Bootstrap 的 Sass 版本的同学，就一眼能看出，上面的示例代码是 Bootstrap 定义 primarybutton 的颜色。

## [Sass]普通变量与默认变量

**普通变量**

定义之后可以在全局范围内使用。

$fontSize: 12px;

body{

font-size:$fontSize;

}

编译后的css代码：

body{

font-size:12px;

}

**默认变量**

sass 的默认变量仅需要在值后面加上 !default 即可。

$baseLineHeight:1.5 !default;

body{

line-height: $baseLineHeight;

}

编译后的css代码：

body{

line-height:1.5;

}

sass 的默认变量一般是用来设置默认值，然后根据需求来覆盖的，覆盖的方式也很简单，只需要在默认变量之前重新声明下变量即可。

$baseLineHeight: 2;

$baseLineHeight: 1.5 !default;

body{

line-height: $baseLineHeight;

}

编译后的css代码：

body{

line-height:2;

}

可以看出现在编译后的 line-height 为 2，而不是我们默认的 1.5。默认变量的价值在进行组件化开发的时候会非常有用。

## [Sass]变量的调用

在 Sass 中声明了变量之后，就可以在需要的地方调用变量。调用变量的方法也非常的简单。

比如在定义了变量

$brand-primary : darken(#428bca, 6.5%) !default; // #337ab7

$btn-primary-color: #fff !default;

$btn-primary-bg : $brand-primary !default;

$btn-primary-border : darken($btn-primary-bg, 5%) !default;

在按钮 button 中调用，可以按下面的方式调用

.btn-primary {

background-color: $btn-primary-bg;

color: $btn-primary-color;

border: 1px solid $btn-primary-border;

}

编译出来的CSS:

.btn-primary {

background-color: #337ab7;

color: #fff;

border: 1px solid #2e6da4;

}

## [Sass]局部变量和全局变量

Sass 中变量的作用域在过去几年已经发生了一些改变。直到最近，规则集和其他范围内声明变量的作用域才默认为本地。如果已经存在同名的全局变量，从 3.4 版本开始，Sass 已经可以正确处理作用域的概念，并通过创建一个新的局部变量来代替。

**全局变量与局部变量**

先来看一下代码例子：

//SCSS

$color: orange !default;//定义全局变量(在选择器、函数、混合宏...的外面定义的变量为全局变量)

.block {

color: $color;//调用全局变量

}

em {

$color: red;//定义局部变量

a {

color: $color;//调用局部变量

}

}

span {

color: $color;//调用全局变量

}

css 的结果：

//CSS

.block {

color: orange;

}

em a {

color: red;

}

span {

color: orange;

}

上面的示例演示可以得知，在元素内部定义的变量不会影响其他元素。如此可以简单的理解成，**全局变量**就是定义在元素外面的变量，如下代码：

$color:orange !default;

$color 就是一个全局变量，而定义在元素内部的变量，比如 $color:red; 是一个**局部变量**。

除此之外，Sass 现在还提供一个 !global 参数。!global 和 !default 对于定义变量都是很有帮助的。我们之后将会详细介绍这两个参数的使用以及其功能。

**全局变量的影子**

当在局部范围（选择器内、函数内、混合宏内...）声明一个已经存在于全局范围内的变量时，局部变量就成为了**全局变量的影子**。基本上，**局部变量只会在局部范围内覆盖全局变量**。

上面例子中的 em 选择器内的变量 $color 就是一个全局变量的影子。

//SCSS

$color: orange !default;//定义全局变量

.block {

color: $color;//调用全局变量

}

em {

$color: red;//定义局部变量（全局变量 $color 的影子）

a {

color: $color;//调用局部变量

}

}

**什么时候声明变量？**

我的建议，创建变量只适用于感觉确有必要的情况下。不要为了某些骇客行为而声明新变量，这丝毫没有作用。只有满足所有下述标准时方可创建新变量：

1. 该值至少重复出现了两次；
2. 该值至少可能会被更新一次；
3. 该值所有的表现都与变量有关（非巧合）。

基本上，没有理由声明一个永远不需要更新或者只在单一地方使用变量。

温馨小提示：您在学习 sass 时，除了在我们网页上可以做练习，还有一个便利在线编辑器网址如下：

http://sassmeister.com/

## [Sass]嵌套-选择器嵌套

Sass 中还提供了选择器嵌套功能，但这也并不意味着你在 Sass 中的嵌套是无节制的，因为你嵌套的层级越深，编译出来的 CSS 代码的选择器层级将越深，这往往是大家不愿意看到的一点。这个特性现在正被众多开发者滥用。

选择器嵌套为样式表的作者提供了一个通过局部选择器相互嵌套实现全局选择的方法，Sass 的嵌套分为三种：

* 选择器嵌套
* 属性嵌套
* 伪类嵌套

1、选择器嵌套

假设我们有一段这样的结构：

<header>

<nav>

<a href=“##”>Home</a>

<a href=“##”>About</a>

<a href=“##”>Blog</a>

</nav>

<header>

想选中 header 中的 a 标签，在写 CSS 会这样写：

nav a {

color:red;

}

header nav a {

color:green;

}

那么在 Sass 中，就可以使用选择器的嵌套来实现：

nav {

a {

color: red;

header & {

color:green;

}

}

}

### 任务

在编辑器第 2 行和第 5 行输入正确的选择器，使其编译出来的CSS如下：

nav a {

color: red;

}

header nav a {

color: green;

}

## [Sass]嵌套-属性嵌套

Sass 中还提供属性嵌套，CSS 有一些属性前缀相同，只是后缀不一样，比如：border-top/border-right，与这个类似的还有 margin、padding、font 等属性。假设你的样式中用到了：

.box {

border-top: 1px solid red;

border-bottom: 1px solid green;

}

在 Sass 中我们可以这样写：

.box {

border: {

top: 1px solid red;

bottom: 1px solid green;

}

}

### 任务

在编辑器第 2 行、3 行和第 4 行输入正确的代码，使其编译出来的 CSS 如下：

.box {

font-size: 12px;

font-weight: bold;

}

## [Sass]嵌套-伪类嵌套

其实伪类嵌套和属性嵌套非常类似，只不过他需要借助`&`符号一起配合使用。我们就拿经典的“clearfix”为例吧：

.clearfix{

&:before,

&:after {

content:"";

display: table;

}

&:after {

clear:both;

overflow: hidden;

}

}

编译出来的 CSS：

clearfix:before, .clearfix:after {

content: "";

display: table;

}

.clearfix:after {

clear: both;

overflow: hidden;

}

**避免选择器嵌套：**

* 选择器嵌套最大的问题是将使最终的代码难以阅读。开发者需要花费巨大精力计算不同缩进级别下的选择器具体的表现效果。
* 选择器越具体则声明语句越冗长，而且对最近选择器的引用(&)也越频繁。在某些时候，出现混淆选择器路径和探索下一级选择器的错误率很高，这非常不值得。

为了防止此类情况，我们应该尽可能避免选择器嵌套。然而，显然只有少数情况适应这一措施。

### 任务

在编辑器第2行输入正确的代码，使编译出来的CSS代码如下：

.box:before{

content:"伪元素嵌套";

}

## [Sass]混合宏-声明混合宏

如果你的整个网站中有几处小样式类似，比如颜色，字体等，在 Sass 可以使用变量来统一处理，那么这种选择还是不错的。但当你的样式变得越来越复杂，需要重复使用大段的样式时，使用变量就无法达到我们目了。这个时候 Sass 中的混合宏就会变得非常有意义。在这一节中，主要向大家介绍 Sass 的混合宏。

1、声明混合宏

**不带参数混合宏**：

在 Sass 中，使用“@mixin”来声明一个混合宏。如：

@mixin border-radius{

-webkit-border-radius: 5px;

border-radius: 5px;

}

其中 @mixin 是用来声明混合宏的关键词，有点类似 CSS 中的 @media、@font-face 一样。border-radius 是混合宏的名称。大括号里面是复用的样式代码。

**带参数混合宏**：

除了声明一个不带参数的混合宏之外，还可以在定义混合宏时带有参数，如：

@mixin border-radius($radius:5px){

-webkit-border-radius: $radius;

border-radius: $radius;

}

**复杂的混合宏：**

上面是一个简单的定义混合宏的方法，当然， Sass 中的混合宏还提供更为复杂的，你可以在大括号里面写上带有逻辑关系，帮助更好的做你想做的事情,如：

@mixin box-shadow($shadow**...**) {

@if length($shadow) >= 1 {

@include prefixer(box-shadow, $shadow);

} @else{

$shadow:0 0 4px rgba(0,0,0,.3);

@include prefixer(box-shadow, $shadow);

}

}

这个 box-shadow 的混合宏，带有多个参数，这个时候可以使用“ … ”来替代。简单的解释一下，当 $shadow 的参数数量值大于或等于“ 1 ”时，表示有多个阴影值，反之调用默认的参数值“ 0 0 4px rgba(0,0,0,.3) ”。

注：复杂的混合宏中的逻辑关系（@if...@else）后面小节会有讲解。

### 任务

在编辑器第 1 行输入正确的代码，用来声明一个 border-radius 混合宏？

**注意**：这里只是声明，并没有调用，所以右侧浏览器结果窗口是没有任何结果的。下个小节中学习完宏的调用后，就可看到结果了。

## [Sass]混合宏-调用混合宏

在 Sass 中通过 @mixin 关键词声明了一个混合宏，那么在实际调用中，其匹配了一个关键词“@include”来调用声明好的混合宏。例如在你的样式中定义了一个圆角的混合宏“border-radius”:

@mixin border-radius{

-webkit-border-radius: 3px;

border-radius: 3px;

}

在一个按钮中要调用定义好的混合宏“border-radius”，可以这样使用：

button {

**@include** border-radius;

}

这个时候编译出来的 CSS:

button {

-webkit-border-radius: 3px;

border-radius: 3px;

}

## [Sass]混合宏的参数--传一个不带值的参数

Sass 的混合宏有一个强大的功能，可以传参，那么在 Sass 中传参主要有以下几种情形：

A) 传一个不带值的参数

在混合宏中，可以传一个不带任何值的参数，比如：

@mixin border-radius($radius){

-webkit-border-radius: $radius;

border-radius: $radius;

}

在混合宏“border-radius”中定义了一个不带任何值的参数“$radius”。

在调用的时候可以给这个混合宏传一个参数值：

.box {

@include border-radius(3px);

}

这里表示给混合宏传递了一个“border-radius”的值为“3px”。

编译出来的 CSS:

.box {

-webkit-border-radius: 3px;

border-radius: 3px;

}

## [Sass]混合宏的参数--传一个带值的参数

在 Sass 的混合宏中，还可以给混合宏的参数传一个默认值，例如：

@mixin border-radius($radius:3px){

-webkit-border-radius: $radius;

border-radius: $radius;

}

在混合宏“border-radius”传了一个参数“$radius”，而且给这个参数赋予了一个默认值“3px”。

在调用类似这样的混合宏时，会多有一个机会，假设你的页面中的圆角很多地方都是“3px”的圆角，那么这个时候只需要调用默认的混合宏“border-radius”:

.btn {

@include border-radius;

}

编译出来的 CSS:

.btn {

-webkit-border-radius: 3px;

border-radius: 3px;

}

但有的时候，页面中有些元素的圆角值不一样，那么可以随机给混合宏传值，如：

.box {

@include border-radius(50%);

}

编译出来的 CSS:

.box {

-webkit-border-radius: 50%;

border-radius: 50%;

}

## [Sass]混合宏的参数--传多个参数

Sass 混合宏除了能传一个参数之外，还可以传多个参数，如：

@mixin center($width,$height){

width: $width;

height: $height;

position: absolute;

top: 50%;

left: 50%;

margin-top: -($height) / 2;

margin-left: -($width) / 2;

}

在混合宏“center”就传了多个参数。在实际调用和其调用其他混合宏是一样的：

.box-center {

@include center(500px,300px);

}

编译出来 CSS:

.box-center {

width: 500px;

height: 300px;

position: absolute;

top: 50%;

left: 50%;

margin-top: -150px;

margin-left: -250px;

}

  有一个特别的参数“**…**”。当混合宏传的参数过多之时，可以使用参数来替代，如：

@mixin box-shadow($shadows**...**){

@if length($shadows) >= 1 {

-webkit-box-shadow: $shadows;

box-shadow: $shadows;

} @else {

$shadows: 0 0 2px rgba(#000,.25);

-webkit-box-shadow: $shadow;

box-shadow: $shadow;

}

}

在实际调用中：

.box {

@include box-shadow(0 0 1px rgba(#000,.5),0 0 2px rgba(#000,.2));

}

编译出来的CSS:

.box {

-webkit-box-shadow: 0 0 1px rgba(0, 0, 0, 0.5), 0 0 2px rgba(0, 0, 0, 0.2);

box-shadow: 0 0 1px rgba(0, 0, 0, 0.5), 0 0 2px rgba(0, 0, 0, 0.2);

}

## [Sass]混合宏的参数--混合宏的不足

混合宏在实际编码中给我们带来很多方便之处，特别是对于复用重复代码块。但其最大的不足之处是会生成冗余的代码块。比如在不同的地方调用一个相同的混合宏时。如：

@mixin border-radius{

-webkit-border-radius: 3px;

border-radius: 3px;

}

.box {

@include border-radius;

margin-bottom: 5px;

}

.btn {

@include border-radius;

}

示例在“.box”和“.btn”中都调用了定义好的“border-radius”混合宏。先来看编译出来的 CSS：

.box {

-webkit-border-radius: 3px;

border-radius: 3px;

margin-bottom: 5px;

}

.btn {

-webkit-border-radius: 3px;

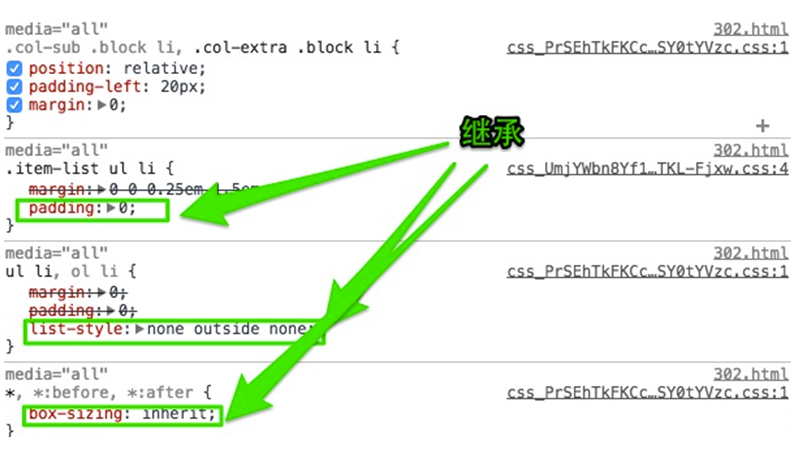
border-radius: 3px;

}

上例明显可以看出，Sass 在调用相同的混合宏时，并不能智能的将相同的样式代码块合并在一起。这也是 Sass 的混合宏最不足之处。

## [Sass]扩展/继承

继承对于了解 CSS 的同学来说一点都不陌生，先来看一张图：

[](https://img.mukewang.com/55261e8200017bfc07910449.jpg)

图中代码显示“.col-sub .block li,.col-extra .block li” 继承了 “.item-list ul li”选择器的 “padding : 0;” 和 “ul li” 选择器中的 “list-style : none outside none;”以及 \* 选择器中的 “box-sizing:inherit;”。

在 Sass 中也具有继承一说，也是继承类中的样式代码块。在 Sass 中是通过关键词 “@extend”来继承已存在的类样式块，从而实现代码的继承。如下所示：

//SCSS

.btn {

border: 1px solid #ccc;

padding: 6px 10px;

font-size: 14px;

}

.btn-primary {

background-color: #f36;

color: #fff;

@extend .btn;

}

.btn-second {

background-color: orange;

color: #fff;

@extend .btn;

}

编译出来之后：

//CSS

.btn, .btn-primary, .btn-second {

border: 1px solid #ccc;

padding: 6px 10px;

font-size: 14px;

}

.btn-primary {

background-color: #f36;

color: #fff;

}

.btn-second {

background-clor: orange;

color: #fff;

}

从示例代码可以看出，在 Sass 中的继承，可以继承类样式块中所有样式代码，而且编译出来的 CSS 会将选择器合并在一起，形成组合选择器：

.btn, .btn-primary, .btn-second {

border: 1px solid #ccc;

padding: 6px 10px;

font-size: 14px;

}

## [Sass]占位符 %placeholder

Sass 中的占位符 %placeholder 功能是一个很强大，很实用的一个功能，这也是我非常喜欢的功能。他可以取代以前 CSS 中的基类造成的代码冗余的情形。因为 %placeholder 声明的代码，如果不被 @extend 调用的话，不会产生任何代码。来看一个演示：

%mt5 {

margin-top: 5px;

}

%pt5{

padding-top: 5px;

}

这段代码没有被 @extend 调用，他并没有产生任何代码块，只是静静的躺在你的某个 SCSS 文件中。只有通过 @extend 调用才会产生代码：

//SCSS

%mt5 {

margin-top: 5px;

}

%pt5{

padding-top: 5px;

}

.btn {

@extend %mt5;

@extend %pt5;

}

.block {

@extend %mt5;

span {

@extend %pt5;

}

}

编译出来的CSS

//CSS

.btn, .block {

margin-top: 5px;

}

.btn, .block span {

padding-top: 5px;

}

从编译出来的 CSS 代码可以看出，通过 @extend 调用的占位符，编译出来的代码会将相同的代码合并在一起。这也是我们希望看到的效果，也让你的代码变得更为干净。

## [Sass]混合宏 VS 继承 VS 占位符

初学者都常常纠结于这个问题“什么时候用混合宏，什么时候用继承，什么时候使用占位符？”其实他们各有各的优点与缺点，先来看看他们使用效果：

**a) Sass 中的混合宏使用**

举例代码见右侧 2-24 行

编译出来的 CSS 见右侧结果窗口。

**总结：**编译出来的 CSS 清晰告诉了大家，他不会自动合并相同的样式代码，如果在样式文件中调用同一个混合宏，会产生多个对应的样式代码，造成代码的冗余，这也是 CSSer 无法忍受的一件事情。不过他并不是一无事处，他可以传参数。

**个人建议**：如果你的代码块中涉及到变量，建议使用混合宏来创建相同的代码块。

**b) Sass 中继承**

同样的，将上面代码中的混合宏，使用类名来表示，然后通过继承来调用：

代码见右侧 26-48 行

**总结：**使用继承后，编译出来的 CSS 会将使用继承的代码块合并到一起，通过组合选择器的方式向大家展现，比如 .mt, .block, .block span, .header, .header span。这样编译出来的代码相对于混合宏来说要干净的多，也是 CSSer 期望看到。但是他不能传变量参数。

**个人建议**：如果你的代码块不需要专任何变量参数，而且有一个基类已在文件中存在，那么建议使用 Sass 的继承。

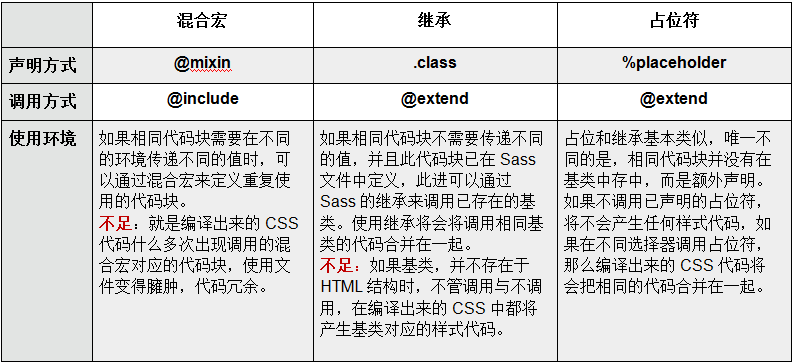
**c) 占位符**

最后来看占位符，将上面代码中的基类 .mt 换成 Sass 的占位符格式：

代码见右侧 50-72 行

**总结：**编译出来的 CSS 代码和使用继承基本上是相同，只是不会在代码中生成占位符 mt 的选择器。那么占位符和继承的主要区别的，“占位符是独立定义，不调用的时候是不会在 CSS 中产生任何代码；继承是首先有一个基类存在，不管调用与不调用，基类的样式都将会出现在编译出来的 CSS 代码中。”

来看一个表格：

[](https://img.mukewang.com/55263aa30001913307940364.jpg)

## [Sass]插值#{}

使用 CSS 预处理器语言的一个主要原因是想使用 Sass 获得一个更好的结构体系。比如说你想写更干净的、高效的和面向对象的 CSS。Sass 中的插值(Interpolation)就是重要的一部分。让我们看一下下面的例子：

$properties: (margin, padding);

@mixin set-value($side, $value) {

@each $prop in $properties {

#{$prop}-#{$side}: $value;

}

}

.login-box {

@include set-value(top, 14px);

}

@each...in...语句会在《Sass进级篇》中 [1-6 @each循环](http://www.imooc.com/code/8811) 中讲解。

它可以让变量和属性工作的很完美，上面的代码编译成 CSS：

.login-box {

margin-top: 14px;

padding-top: 14px;

}

这是 Sass 插值中一个简单的实例。当你想设置属性值的时候你可以使用字符串插入进来。另一个有用的用法是构建一个选择器。可以这样使用：

@mixin generate-sizes($class, $small, $medium, $big) {

.#{$class}-small { font-size: $small; }

.#{$class}-medium { font-size: $medium; }

.#{$class}-big { font-size: $big; }

}

@include generate-sizes("header-text", 12px, 20px, 40px);

编译出来的 CSS:

.header-text-small { font-size: 12px; }

.header-text-medium { font-size: 20px; }

.header-text-big { font-size: 40px; }

一旦你发现这一点，你就会想到超级酷的 mixins，用来生成代码或者生成另一个 mixins。然而，这并不完全是可能的。第一个限制，这可能会很删除用于 Sass 变量的插值。

$margin-big: 40px;

$margin-medium: 20px;

$margin-small: 12px;

@mixin set-value($size) {

margin-top: $margin-#{$size};

}

.login-box {

@include set-value(big);

}

上面的 Sass 代码编译出来，你会得到下面的信息：

error style.scss (Line 5: Undefined variable: “$margin-".)

所以，#{}语法并不是随处可用，你也不能在 mixin 中调用：

@mixin updated-status {

margin-top: 20px;

background: #F00;

}

$flag: "status";

.navigation {

@include updated-#{$flag};

}

上面的代码在编译成 CSS 时同样会报错：

error style.scss (Line 7: Invalid CSS after "...nclude updated-": expected "}", was "#{$flag};")  
幸运的是，可以使用 @extend 中使用插值。例如：

%updated-status {

margin-top: 20px;

background: #F00;

}

.selected-status {

font-weight: bold;

}

$flag: "status";

.navigation {

@extend %updated-#{$flag};

@extend .selected-#{$flag};

}

上面的 Sass 代码是可以运行的，因为他给了我们力量，可以动态的插入 .class 和 %placeholder。当然他们不能接受像 mixin 这样的参数，上面的代码编译出来的 CSS:

.navigation {

margin-top: 20px;

background: #F00;

}

.selected-status, .navigation {

font-weight: bold;

}

在 Sass 的社区正在积极讨论插值的局限性，谁又知道呢，也许我们很快将能够使用这些技术也说不定呢。

## [Sass]注释

注释对于一名程序员来说，是极其重要，良好的注释能帮助自己或者别人阅读源码。在 Sass 中注释有两种方式，我暂且将其命名为：

1、类似 CSS 的注释方式，使用 ”/\* ”开头，结属使用 ”\*/ ”  
2、类似 JavaScript 的注释方式，使用“//”

两者区别，前者会在编译出来的 CSS 显示，后者在编译出来的 CSS 中不会显示，来看一个示例：

//定义一个占位符

%mt5 {

margin-top: 5px;

}

/\*调用一个占位符\*/

.box {

@extend %mt5;

}

编译出来的CSS

.box {

margin-top: 5px;

}

/\*调用一个占位符\*/

## [Sass]数据类型

 Sass 和 JavaScript 语言类似，也具有自己的数据类型，在 Sass 中包含以下几种数据类型：

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

后两个小节详细讲解字符串和值列表数据类型，其它类型与JavaScript中的用法一致。

## [Sass]字符串

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'；
* 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。

## [Sass]值列表

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

像上面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列表函数（Sass list functions）赋予了值列表更多功能（Sass进级会有讲解）：

1. nth函数（nth function） 可以直接访问值列表中的某一项；
2. join函数（join function） 可以将多个值列表连结在一起；
3. append函数（append function） 可以在值列表中添加值；
4. @each规则（@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。

# Sass的基本特性-运算

## [Sass运算]加法

程序中的运算是常见的一件事情，但在 CSS 中能做运算的，到目前为止仅有 calc() 函数可行。但在 Sass 中，运算只是其基本特性之一。在 Sass 中可以做各种数学计算，在接下来的章节中，主要和大家一起探讨有关于 Sass 中的数学运算。

（一）、加法

加法运算是 Sass 中运算中的一种，在变量或属性中都可以做加法运算。如：

.box {

width: 20px + 8in;

}

编译出来的 CSS:

.box {

width: 788px;

}

但对于携带不同类型的单位时，在 Sass 中计算会报错，如下例所示：

.box {

width: 20px + 1em;

}

编译的时候，编译器会报错：“Incompatible units: 'em' and ‘px'.”

## [Sass运算]减法

Sass 的减法运算和加法运算类似，我们通过一个简单的示例来做阐述：

$full-width: 960px;

$sidebar-width: 200px;

.content {

width: $full-width **-** $sidebar-width;

}

编译出来的 CSS 如下：

.content {

width: 760px;

}

同样的，运算时碰到不同类型的单位时，编译也会报错，如：

$full-width: 960px;

.content {

width: $full-width - 1em;

}

编译的时候，编译器报“Incompatible units: 'em' and ‘px’.”错误。

## [Sass运算]乘法

Sass 中的乘法运算和前面介绍的加法与减法运算还略有不同。虽然他也能够支持多种单位（比如 em ,px , %），但当一个单位同时声明两个值时会有问题。比如下面的示例：

.box {

width:10px \* 2px;

}

编译的时候报“20px\*px isn't a valid CSS value.”错误信息。

如果进行乘法运算时，两个值单位相同时，只需要为一个数值提供单位即可。上面的示例可以修改成：

.box {

width: 10px \* 2;

}

编译出来的 CSS:

.box {

width: 20px;

}

Sass 的乘法运算和加法、减法运算一样，在运算中有不同类型的单位时，也将会报错。如下面的示例：

.box {

width: 20px \* 2em;

}

编译时报“40em\*px isn't a valid CSS value.”错误信息。

## [Sass运算]除法

Sass 的乘法运算规则也适用于除法运算。不过除法运算还有一个特殊之处。众所周知“**/**”符号在 CSS 中已做为一种符号使用。因此在 Sass 中做除法运算时，直接使用“/”符号做为除号时，将不会生效，编译时既得不到我们需要的效果，也不会报错。一起先来看一个简单的示例：

.box {

width: 100px / 2;

}

编译出来的 CSS 如下：

.box {

width: 100px / 2;

}

这样的结果对于大家来说没有任何意义。要修正这个问题，只需要给运算的外面添加一个小括号**( )**即可：

.box {

width: **(**100px / 2**)**;

}

编译出来的 CSS 如下：

.box {

width: 50px;

}

除了上面情况带有小括号，“/”符号会当作除法运算符之外，如果“/”符号在已有的数学表达式中时，也会被认作除法符号。如下面示例：

.box {

width: 100px / 2 + 2in;

}

编译出来的CSS：

.box {

width: 242px;

}

另外，在 Sass 除法运算中，当用变量进行除法运算时，“/”符号也会自动被识别成除法，如下例所示：

$width: 1000px;

$nums: 10;

.item {

width: $width / 10;

}

.list {

width: $width / $nums;

}

编译出来的CSS:

.item {

width: 100px;

}

.list {

width: 100px;

}

综合上述，”/  ”符号被当作除法运算符时有以下几种情况：

•    如果数值或它的任意部分是存储在一个变量中或是函数的返回值。  
•    如果数值被圆括号包围。  
•    如果数值是另一个数学表达式的一部分。

如下所示：

//SCSS

p {

font: 10px/8px; // 纯 CSS，不是除法运算

$width: 1000px;

width: $width/2; // 使用了变量，是除法运算

width: round(1.5)/2; // 使用了函数，是除法运算

height: (500px/2); // 使用了圆括号，是除法运算

margin-left: 5px + 8px/2px; // 使用了加（+）号，是除法运算

}

编译出来的CSS

p {

font: 10px/8px;

width: 500px;

height: 250px;

margin-left: 9px;

}

Sass 的除法运算还有一个情况。我们先回忆一下，在乘法运算时，如果两个值带有相同单位时，做乘法运算时，出来的结果并不是我们需要的结果。但在除法运算时，如果两个值带有相同的单位值时，除法运算之后会得到一个不带单位的数值。如下所示：

.box {

width: (1000px / 100px);

}

编译出来的CSS如下：

.box {

width: 10;

}

## [Sass运算]变量计算

在 Sass 中除了可以使用数值进行运算之外，还可以使用变量进行计算，其实在前面章节的示例中也或多或少的向大家展示了。在 Sass 中使用变量进行计算，这使得 Sass 的数学运算功能变得更加实用。一起来看一个简单的示例：

$content-width: 720px;

$sidebar-width: 220px;

$gutter: 20px;

.container {

width: $content-width + $sidebar-width + $gutter;

margin: 0 auto;

}

编译出来的CSS

.container {

width: 960px;

margin: 0 auto;

}

## [Sass运算]数字运算

在 Sass 运算中数字运算是较为常见的，数字运算包括前面介绍的：加法、减法、乘法和除法等运算。而且还可以通过括号来修改他们的运算先后顺序。和我们数学运算是一样的，一起来看个示例。

.box {

width: ((220px + 720px) - 11 \* 20 ) / 12 ;

}

编译出来的 CSS:

.box {

width: 60px;

}

上面这个简单示例是一个典型的计算 Grid 单列列宽的运算。

## [Sass运算]颜色运算

所有算数运算都支持颜色值，并且是分段运算的。也就是说，红、绿和蓝各颜色分段单独进行运算。如：

p {

color: #010203 + #040506;

}

计算公式为 01 + 04 = 05、02 + 05 = 07 和 03 + 06 = 09， 并且被合成为：

如此编译出来的 CSS 为：

p {

color: #050709;

}

算数运算也能将数字和颜色值 一起运算，同样也是分段运算的。如：

p {

color: #010203 \* 2;

}

计算公式为 01 \* 2 = 02、02 \* 2 = 04 和 03 \* 2 = 06， 并且被合成为：

p {

color: #020406;

}

## [Sass运算]字符运算

在 Sass 中可以通过加法符号“+”来对字符串进行连接。例如：

$content: "Hello" + "" + "Sass!";

.box:before {

content: " #{$content} ";

}

编译出来的CSS：

.box:before {

content: " Hello Sass! ";

}

除了在变量中做字符连接运算之外，还可以直接通过 +，把字符连接在一起：

div {

cursor: e + -resize;

}

编译出来的CSS:

div {

cursor: e-resize;

}

注意，如果有引号的字符串被添加了一个没有引号的字符串 （也就是，带引号的字符串在 + 符号左侧）， 结果会是一个有引号的字符串。 同样的，如果一个没有引号的字符串被添加了一个有引号的字符串 （没有引号的字符串在 + 符号左侧）， 结果将是一个没有引号的字符串。 例如：

p:before {

content: "Foo " + Bar;

font-family: sans- + "serif";

}

编译出来的 CSS：

p:before {

content: "Foo Bar";

font-family: sans-serif; }

小伙伴们，到此为止《sass基础入门（基础篇）》的课程已经全部讲解完了，后面还有《sass基础入门（进阶篇）》即将上线，欢迎小伙伴们到时学习。

同时小伙伴们如果想看一下 sass 的案例课程可观看[《Sass和Compass必备技能之Sass篇》](http://www.imooc.com/learn/364)课程。

# Sass进阶篇

## Sass的控制命令

### @if

@if 指令是一个 SassScript，它可以根据条件来处理样式块，如果条件为 true 返回一个样式块，反之 false 返回另一个样式块。在 Sass 中除了 @if 之，还可以配合 @else if 和 @else 一起使用。

假设要控制一个元素隐藏或显示，我们就可以定义一个混合宏，通过 @if...@else... 来判断传进参数的值来控制 display 的值。如下所示：

//SCSS

@mixin blockOrHidden($boolean:true) {

@if $boolean {

@debug "$boolean is #{$boolean}";

display: block;

}

@else {

@debug "$boolean is #{$boolean}";

display: none;

}

}

.block {

@include blockOrHidden;

}

.hidden{

@include blockOrHidden(false);

}

编译出来的CSS:

.block {

display: block;

}

.hidden {

display: none;

}

### @for循环（上）

在制作网格系统的时候，大家应该对 .col1~.col12 这样的印象较深。在 CSS 中你需要一个一个去书写，但在 Sass 中，可以使用 @for 循环来完成。在 Sass 的 @for 循环中有两种方式：

@for $i from <start> through <end>

@for $i from <start> to <end>

* $i 表示变量
* start 表示起始值
* end 表示结束值

这两个的区别是关键字 through 表示包括 end 这个数，而 to 则不包括 end 这个数。

如下代码，先来个使用 through 关键字的例子：

@for $i from 1 through 3 {

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

}

编译出来的 CSS:

.item-1 {

width: 2em;

}

.item-2 {

width: 4em;

}

.item-3 {

width: 6em;

}

再来个 to 关键字的例子：

@for $i from 1 to 3 {

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

}

编译出来的 CSS:

.item-1 {

width: 2em;

}

.item-2 {

width: 4em;

}

### @for循环 （下）

上一小节的那个实例几乎用不着，哈哈，所以其实是没什么营养的东西，只是帮助理解了原来 @for 是这么回事。怎么的也不能就这么忽悠大家啊，大家好不容易抽空看下文章，就这么点扯淡的东西怎么对得住呢。下面再来个营养级别的，@for应用在网格系统生成各个格子 class 的代码：

//SCSS

$grid-prefix: span !default;

$grid-width: 60px !default;

$grid-gutter: 20px !default;

%grid {

float: left;

margin-left: $grid-gutter / 2;

margin-right: $grid-gutter / 2;

}

@for $i from 1 through 12 {

.#{$grid-prefix}#{$i}{

width: $grid-width \* $i + $grid-gutter \* ($i - 1);

@extend %grid;

}

}

编译出来的 CSS:

.span1, .span2, .span3, .span4, .span5, .span6, .span7, .span8, .span9, .span10, .span11, .span12 {

float: left;

margin-left: 10px;

margin-right: 10px;

}

.span1 {

width: 60px;

}

.span2 {

width: 140px;

}

.span3 {

width: 220px;

}

.span4 {

width: 300px;

}

.span5 {

width: 380px;

}

.span6 {

width: 460px;

}

.span7 {

width: 540px;

}

.span8 {

width: 620px;

}

.span9 {

width: 700px;

}

.span10 {

width: 780px;

}

.span11 {

width: 860px;

}

.span12 {

width: 940px;

}

将上面的示例稍做修改，将 @for through 方式换成 @for to:：

//SCSS

@for $i from 1 to 13 {

.#{$grid-prefix}#{$i}{

width: $grid-width \* $i + $grid-gutter \* ($i - 1);

@extend %grid;

}

}

其最终编译出来的 CSS 代码和上例所编译出来的一模一样。

这两段 Sass 代码并无太多差别，只是 @for中的 <end> 取值不同。配合 through 的 <end> 值是 12，其遍历出来的终点值也是 12，和 <end> 值一样。配合 to 的 <end> 值是 13，其遍历出来的终点值是 12，就是 <end> 对就的值减去 1 。

### @while循环

@while 指令也需要 SassScript 表达式（像其他指令一样），并且会生成不同的样式块，直到表达式值为 false 时停止循环。这个和 @for 指令很相似，只要 @while 后面的条件为 true 就会执行。

这里有一个 @while 指令的简单用例：

//SCSS

$types: 4;

$type-width: 20px;

@while $types > 0 {

.while-#{$types} {

width: $type-width + $types;

}

$types: $types - 1;

}

编译出来的 CSS

.while-4 {

width: 24px;

}

.while-3 {

width: 23px;

}

.while-2 {

width: 22px;

}

.while-1 {

width: 21px;

}

### @each循环

@each 循环就是去遍历一个列表，然后从列表中取出对应的值。

@each 循环指令的形式：

@each $var in <list>

如果你没有接触过列表，也不要紧，他也非常简单。

在下面的例子中你可以看到，$var 就是一个变量名，<list> 是一个 SassScript 表达式，他将返回一个列表值。变量 $var 会在列表中做遍历，并且遍历出与 $var 对应的样式块。

这有一个 @each 指令的简单示例：

$list: adam john wynn mason kuroir;//$list 就是一个列表

@mixin author-images {

@each $author in $list {

.photo-#{$author} {

background: url("/images/avatars/#{$author}.png") no-repeat;

}

}

}

.author-bio {

@include author-images;

}

编译出 CSS:

.author-bio .photo-adam {

background: url("/images/avatars/adam.png") no-repeat; }

.author-bio .photo-john {

background: url("/images/avatars/john.png") no-repeat; }

.author-bio .photo-wynn {

background: url("/images/avatars/wynn.png") no-repeat; }

.author-bio .photo-mason {

background: url("/images/avatars/mason.png") no-repeat; }

.author-bio .photo-kuroir {

background: url("/images/avatars/kuroir.png") no-repeat; }

## Sass的函数功能-字符串与数字函数

### Sass的函数简介

在 Sass 中除了可以定义变量，具有 @extend、%placeholder 和 mixins 等特性之外，还自备了一系列的函数功能。其主要包括：

* 字符串函数
* 数字函数
* 列表函数
* 颜色函数
* Introspection 函数
* 三元函数等

当然除了自备的函数功能之外，我们还可以根据自己的需求定义函数功能，常常称之为自定义函数。

下面将给大家详细介绍前 4 种最常用的函数。

### 字符串函数-unquote()函数

字符串函数顾名思意是用来处理字符串的函数。Sass 的字符串函数主要包括两个函数：

* **unquote($string)**：删除字符串中的引号；
* **quote($string)**：给字符串添加引号。

**1、unquote()函数**

unquote() 函数主要是用来删除一个字符串中的引号，如果这个字符串没有带有引号，将返回原始的字符串。简单的使用终端来测试这个函数的运行结果：

//SCSS

.test1 {

content: unquote('Hello Sass!') ;

}

.test2 {

content: unquote("'Hello Sass!");

}

.test3 {

content: unquote("I'm Web Designer");

}

.test4 {

content: unquote("'Hello Sass!'");

}

.test5 {

content: unquote('"Hello Sass!"');

}

.test6 {

content: unquote(Hello Sass);

}

编译后的 css 代码：

//CSS

.test1 {

content: Hello Sass!; }

.test2 {

content: 'Hello Sass!; }

.test3 {

content: I'm Web Designer; }

.test4 {

content: 'Hello Sass!'; }

.test5 {

content: "Hello Sass!"; }

.test6 {

content: Hello Sass; }

注意：从测试的效果中可以看出，unquote( ) 函数只能删除字符串最前和最后的引号（双引号或单引号），而无法删除字符串中间的引号。如果字符没有带引号，返回的将是字符串本身

### 字符串函数-To-upper-case()、To-lower-case()

1、To-upper-case()

To-upper-case() 函数将字符串小写字母转换成大写字母。如：

//SCSS

.test {

text: to-upper-case(aaaaa);

text: to-upper-case(aA-aAAA-aaa);

}

编译出来的 css 代码：

//CSS

.test {

text: AAAAA;

text: AA-AAAA-AAA;

}

2、To-lower-case()

To-lower-case() 函数 与 To-upper-case() 刚好相反，将字符串转换成小写字母：

//SCSS

.test {

text: to-lower-case(AAAAA);

text: to-lower-case(aA-aAAA-aaa);

}

编译出来的 css 代码：

//CSS

.test {

text: aaaaa;

text: aa-aaaa-aaa;

}

### 数字函数简介

Sass 中的数字函数提要针对数字方面提供一系列的函数功能：

* **percentage($value)**：将一个不带单位的数转换成百分比值；
* **round($value)**：将数值四舍五入，转换成一个最接近的整数；
* **ceil($value)**：将大于自己的小数转换成下一位整数；
* **floor($value)**：将一个数去除他的小数部分；
* **abs($value)**：返回一个数的绝对值；
* **min($numbers…)**：找出几个数值之间的最小值；
* **max($numbers…)**：找出几个数值之间的最大值；
* **random()**: 获取随机数

看到上面函数的简介，对于熟悉Javascript 同学而言并不会感觉陌生。因为他们有很多功能都非常类似，接下来对每个函数进行一些简单的测试 。

#### 数字函数-percentage()

1、percentage()

percentage()函数主要是将一个不带单位的数字转换成百分比形式：

>> percentage(.2)

20%

>> percentage(2px / 10px)

20%

>> percentage(2em / 10em)

20%

>>

.footer{

width : percentage(.2)

}

编译后的 css 代码：

.footer{

width : 20%

}

如果您转换的值是一个带有单位的值，那么在编译的时候会报错误信息：

>> percentage(2px / 10em)

SyntaxError: $value: 0.2px/em is not a unitless number for `percentage'

#### 数字函数-round()函数

round() 函数可以将一个数四舍五入为一个最接近的整数：

>> round(12.3)

12

>> round(12.5)

13

>> round(1.49999)

1

>> round(2.0)

2

>> round(20%)

20%

>> round(2.2%)

2%

>> round(3.9em)

4em

>> round(2.3px)

2px

>> round(2px / 3px)

1

>> round(1px / 3px)

0

>> round(3px / 2em)

2px/em

.footer {

width:round(12.3px)

}

编译后的 css 代码：

.footer {

width: 12px;

}

在round() 函数中可以携带单位的任何数值。

#### 数字函数-ceil()函数

ceil() 函数将一个数转换成最接近于自己的整数，会将一个大于自身的任何小数转换成大于本身 1 的整数。也就是只做入，不做舍的计算：

>> ceil(2.0)

2

>> ceil(2.1)

3

>> ceil(2.6)

3

>> ceil(2.3%)

3%

>> ceil(2.3px)

3px

>> ceil(2.5px)

3px

>> ceil(2px / 3px)

1

>> ceil(2% / 3px)

1%/px

>> ceil(1em / 5px)

1em/px

.footer {

width:ceil(12.3px);

}

编译后的 css 代码：

.footer {

width: 13px;

}

#### 数字函数-floor()函数

floor() 函数刚好与 ceil() 函数功能相反，其主要将一个数去除其小数部分，并且不做任何的进位。也就是只做舍，不做入的计算：

>> floor(2.1)

2

>> floor(2.5)

2

>> floor(3.5%)

3%

>> floor(10.2px)

10px

>> floor(10.8em)

10em

>> floor(2px / 10px)

0

>> floor(3px / 1em)

3px/em

.footer {

width:floor(12.3px);

}

编译后的 css 代码：

.footer {

width: 12px;

}

#### 数字函数-abs()函数

abs( ) 函数会返回一个数的绝对值。

>> abs(10)

10

>> abs(-10)

10

>> abs(-10px)

10px

>> abs(-2em)

2em

>> abs(-.5%)

0.5%

>> abs(-1px / 2px)

0.5

.footer {

width:abs(-12.3px);

}

编译后的 css 代码：

.footer {

width: 12.3px;

}

#### 数字函数-random()函数

random() 函数是用来获取一个随机数：

>> random()

0.03886

>> random()

0.66527

>> random()

0.8125

>> random()

0.26839

>> random()

0.85063

## Sass的函数功能-列表函数

### 列表函数

列表函数主要包括一些对列表参数的函数使用，主要包括以下几种：

* **length($list)**：返回一个列表的长度值；
* **nth($list, $n)**：返回一个列表中指定的某个标签值
* **join($list1, $list2, [$separator])**：将两个列给连接在一起，变成一个列表；
* **append($list1, $val, [$separator])**：将某个值放在列表的最后；
* **zip($lists…)**：将几个列表结合成一个多维的列表；
* **index($list, $value)**：返回一个值在列表中的位置值。

列表函数中的每个函数都有其独特的作用与功能，接下来我们通过命令终端向大家展示每个列表函数的功能与使用。

#### length()函数

length() 函数主要用来返回一个列表中有几个值，简单点说就是返回列表清单中有多少个值：

>> length(10px)

1

>> length(10px 20px (border 1px solid) 2em)

4

>> length(border 1px solid)

3

length() 函数中的列表参数之间使用空格隔开，不能使用逗号，否则函数将会出错：

>> length(10px,20px,(border 1px solid),2em)

SyntaxError: wrong number of arguments (4 for 1) for `length'

>> length(1,2px)

SyntaxError: wrong number of arguments (2 for 1) for `length'

#### nth()函数

语法:

nth($list,$n)

nth() 函数用来指定列表中某个位置的值。不过在 Sass 中，nth() 函数和其他语言不同，1 是指列表中的第一个标签值，2 是指列给中的第二个标签值，依此类推。如：

>> nth(10px 20px 30px,1)

10px

>> nth((Helvetica,Arial,sans-serif),2)

"Arial"

>> nth((1px solid red) border-top green,1)

(1px "solid" #ff0000)

注：在 nth($list,$n) 函数中的 $n 必须是大于 0 的整数：

>> nth((1px solid red) border-top green 1 ,0)

SyntaxError: List index 0 must be a non-zero integer for `nth'

#### join()函数

join() 函数是将两个列表连接合并成一个列表。

>> join(10px 20px, 30px 40px)

(10px 20px 30px 40px)

>> join((blue,red),(#abc,#def))

(#0000ff, #ff0000, #aabbcc, #ddeeff)

>> join((blue,red),(#abc #def))

(#0000ff, #ff0000, #aabbcc, #ddeeff)

不过 join() 只能将两个列表连接成一个列表，如果直接连接两个以上的列表将会报错：

>> join((blue red),(#abc, #def),(#dee #eff))

SyntaxError: $separator: (#ddeeee #eeffff) is not a string for `join'

但很多时候不只碰到两个列表连接成一个列表，这个时候就需要将多个 join() 函数合并在一起使用：

>> join((blue red), join((#abc #def),(#dee #eff)))

(#0000ff #ff0000 #aabbcc #ddeeff #ddeeee #eeffff)

在 join() 函数中还有一个很特别的参数 $separator，这个参数主要是用来给列表函数连接列表值是，使用的分隔符号，默认值为 auto。

join() 函数中 $separator 除了默认值 auto 之外，还有 comma 和 space 两个值，其中 comma 值指定列表中的列表项值之间使用逗号（,）分隔，space 值指定列表中的列表项值之间使用空格（ ）分隔。

在 join() 函数中除非明确指定了 $separator值，否则将会有多种情形发生：

如果列表中的第一个列表中每个值之间使用的是逗号（,），那么 join() 函数合并的列表中每个列表项之间使用逗号,分隔：

>> join((blue, red, #eff),(green orange))

(#0000ff, #ff0000, #eeffff, #008000, #ffa500)

但当第一个列表中只有一个列表项，那么 join() 函数合并的列表项目中每个列表项目这间使用的分隔符号会根据第二个列表项中使用的，如果第二列表项中使用是,分隔，则使用逗号分隔；如果第二列项之间使用的空格符，则使用空格分隔：

>> join(blue,(green, orange))

(#0000ff, #008000, #ffa500)

>> join(blue,(green orange))

(#0000ff #008000 #ffa500)

如果列表中的第一个列表中每个值之间使用的是空格，那么 join() 函数合并的列表中每个列表项之间使用空格分隔：

>> join((blue green),(red,orange))

(#0000ff #008000 #ff0000 #ffa500)

>> join((blue green),(red orange))

(#0000ff #008000 #ff0000 #ffa500)

如果当两个列表中的列表项小于1时，将会以空格分隔：

>> join(blue,red)

(#0000ff #ff0000)

如此一来，会有多种情形发生，造成使用混乱的情形，如果你无法记得，什么时候会是用逗号分隔合并的列表项，什么时候是使用空格分隔合并 的列表项，在些建议大家使用 join() 函数合并列表项的时候就明确指定 $separator 参数，用来指定合并的列表中使用什么方式来分隔列表项：

>> join(blue,red,comma)

(#0000ff, #ff0000)

>> join(blue,red,space)

(#0000ff #ff0000)

>> join((blue green),(red,orange),comma)

(#0000ff, #008000, #ff0000, #ffa500)

>> join((blue green),(red,orange),space)

(#0000ff #008000 #ff0000 #ffa500)

>> join((blue, green),(red,orange),comma)

(#0000ff, #008000, #ff0000, #ffa500)

>> join((blue, green),(red,orange),space)

(#0000ff #008000 #ff0000 #ffa500)

>> join(blue,(red,orange),comma)

(#0000ff, #ff0000, #ffa500)

>> join(blue,(red,orange),space)

(#0000ff #ff0000 #ffa500)

>> join(blue,(red orange),comma)

(#0000ff, #ff0000, #ffa500)

>> join(blue,(red orange),space)

(#0000ff #ff0000 #ffa500)

#### append()函数

append() 函数是用来将某个值插入到列表中，并且处于最末位。

>> append(10px 20px ,30px)

(10px 20px 30px)

>> append((10px,20px),30px)

(10px, 20px, 30px)

>> append(green,red)

(#008000 #ff0000)

>> append(red,(green,blue))

(#ff0000 (#008000, #0000ff))

如果没有明确的指定 $separator 参数值，其默认值是 auto。

* 如果列表只有一个列表项时，那么插入进来的值将和原来的值会以空格的方式分隔。
* 如果列表中列表项是以空格分隔列表项，那么插入进来的列表项也将以空格分隔；
* 如果列表中列表项是以逗号分隔列表项，那么插入进来的列表项也将以逗号分隔。

当然，在 append() 函数中，可以显示的设置 $separator 参数，

* 如果取值为 comma 将会以逗号分隔列表项
* 如果取值为 space 将会以空格分隔列表项

>> append((blue green),red,comma)

(#0000ff, #008000, #ff0000)

>> append((blue green),red,space)

(#0000ff #008000 #ff0000)

>> append((blue, green),red,comma)

(#0000ff, #008000, #ff0000)

>> append((blue, green),red,space)

(#0000ff #008000 #ff0000)

>> append(blue,red,comma)

(#0000ff, #ff0000)

>> append(blue,red,space)

(#0000ff #ff0000)

#### zip()函数

zip()函数将多个列表值转成一个多维的列表：

>> zip(1px 2px 3px,solid dashed dotted,green blue red)

((1px "solid" #008000), (2px "dashed" #0000ff), (3px "dotted" #ff0000))

在使用zip()函数时，每个单一的列表个数值必须是相同的：

>> zip(1px 2px 3px, solid , green blue red)

NoMethodError: undefined method `options=' for nil:NilClass

Use --trace for backtrace.

否则将会出错。

zip()函数中每个单一列表的值对应的取其相同位置值：

|--- List ---|--- nth(1) ---|--- nth(2) ---|--- nth(3) ---|  
|------------|--------------|--------------|--------------|  
|    List1   |      1px     |      2px     |      3px     |  
|------------|--------------|--------------|--------------|  
|    List2   |      solid   |      dashed  |     dotted   |  
|------------|--------------|--------------|--------------|  
|    List3   |      green   |      blue    |      red     |  
|------------|--------------|--------------|--------------|

zip()函数组合出来就成了：

1px solid green, 2px dashed blue, 3px dotted red

#### index()函数

index() 函数类似于索引一样，主要让你找到某个值在列表中所处的位置。在 Sass 中，第一个值就是1，第二个值就是 2，依此类推：

>> index(1px solid red, 1px)

1

>> index(1px solid red, solid)

2

>> index(1px solid red, red)

3

在 index() 函数中，如果指定的值不在列表中（没有找到相应的值），那么返回的值将是 false，相反就会返回对应的值在列表中所处的位置。

>> index(1px solid red,dotted) //列表中没有找到 dotted

false

>> index(1px solid red,solid) //列表中找到 solid 值，并且返回他的位置值 2

2

### Introspection函数

Introspection 函数包括了几个判断型函数：

* type-of($value)：返回一个值的类型
* unit($number)：返回一个值的单位
* unitless($number)：判断一个值是否带有单位
* comparable($number-1, $number-2)：判断两个值是否可以做加、减和合并

这几个函数主要用来对值做一个判断的作用，我们来依次看每个函数的功能。

#### Introspection 函数 -type-of()

type-of() 函数主要用来判断一个值是属于什么类型：

返回值：

* number 为数值型。
* string 为字符串型。
* bool 为布尔型。
* color 为颜色型。

>> type-of(100)

"number"

>> type-of(100px)

"number"

>> type-of("asdf")

"string"

>> type-of(asdf)

"string"

>> type-of(true)

"bool"

>> type-of(false)

"bool"

>> type-of(#fff)

"color"

>> type-of(blue)

"color"

>> type-of(1 / 2 = 1)

"string"

#### unit()函数

unit() 函数主要是用来获取一个值所使用的单位，碰到复杂的计算时，其能根据运算得到一个“多单位组合”的值，不过只充许**乘、除**运算：

>> unit(100)

""

>> unit(100px)

"px"

>> unit(20%)

"%"

>> unit(1em)

"em"

>> unit(10px \* 3em)

"em\*px"

>> unit(10px / 3em)

"px/em"

>> unit(10px \* 2em / 3cm / 1rem)

"em/rem"

但加、减碰到不同单位时，unit() 函数将会报错，除 px 与 cm、mm 运算之外：

>> unit(1px + 1cm)

"px"

>> unit(1px - 1cm)

"px"

>> unit(1px + 1mm)

"px"

>> unit(10px \* 2em - 3cm / 1rem)

SyntaxError: Incompatible units: 'cm' and 'px\*em'.

>> unit(10px \* 2em - 1px / 1rem)

SyntaxError: Incompatible units: '' and 'em'.

>> unit(1px - 1em)

SyntaxError: Incompatible units: 'em' and 'px'.

>> unit(1px - 1rem)

SyntaxError: Incompatible units: 'rem' and 'px'.

>> unit(1px - 1%)

SyntaxError: Incompatible units: '%' and 'px'.

>> unit(1cm + 1em)

SyntaxError: Incompatible units: 'em' and 'cm'.

unit() 函数对于单位运算相对来说也没有规律，而且有些单位也无法整合成一个单位，对于我们在 CSS 中运用中并不适合，比如：

>> unit(10px \* 3em)

"em\*px"

>> unit(10px / 3em)

"px/em"

>> unit(10px \* 2em / 3cm / 1rem)

"em/rem"

换句话说，上面运算出来的单位，对于在 CSS 中使用将是没有任何意义的。

#### unitless()函数

unitless() 函数相对来说简单明了些，只是用来判断一个值是否带有单位，如果不带单位返回的值为 true，带单位返回的值为 false：

>> unitless(100)

true

>> unitless(100px)

false

>> unitless(100em)

false

>> unitless(100%)

false

>> unitless(1 /2 )

true

>> unitless(1 /2 + 2 )

true

>> unitless(1px /2 + 2 )

false

具体例子，见右侧代码编辑器。右侧代码实现：用户在调用混合宏时，如果用户没有给参数值加上单位，程序会自动加入单位。

#### comparable()函数

comparable() 函数主要是用来判断两个数是否可以进行“加，减”以及“合并”。如果可以返回的值为 true，如果不可以返回的值是 false：

>>

comparable(2px,1px)

true

>> comparable(2px,1%)

false

>> comparable(2px,1em)

false

>> comparable(2rem,1em)

false

>> comparable(2px,1cm)

true

>> comparable(2px,1mm)

true

>>

comparable(2px,1rem)

false

>> comparable(2cm,1mm)

true

### Miscellaneous函数: 三元条件函数

在这里把 Miscellaneous 函数称为三元条件函数，主要因为他和 JavaScript 中的三元判断非常的相似。他有两个值，当条件成立返回一种值，当条件不成立时返回另一种值：

if($condition,$if-true,$if-false)

上面表达式的意思是当 $condition 条件成立时，返回的值为 $if-true，否则返回的是 $if-false 值。

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

1px

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

2px

### Map

Sass 的 map 常常被称为数据地图，也有人称其为数组，因为他总是以 key:value 成对的出现，但其更像是一个 JSON 数据。

{

"employees": [

{ "firstName":"John" , "lastName":"Doe" },

{ "firstName":"Anna" , "lastName":"Smith" },

{ "firstName":"Peter" , "lastName":"Jones" }

]

}

那么 Sass 的 map 长得与 JSON 极其相似：

$map: (

$key1: value1,

$key2: value2,

$key3: value3

)

首先有一个类似于 Sass 的变量一样，用个 $ 加上命名空间来声明 map。后面紧接是一个小括号 ()，将数据以 key:value 的形式赋予，其中 key 和 value 是成对出现，并且每对之间使用逗号 (,) 分隔，其中最后一组后面没有逗号。

其中键 key 是用来查找相关联的值 value。使用 map 可以很容易收集键的值和动态插入。我们来回忆一下，在 Sass 中常用下面的方式定义变量：

$default-color: #fff !default;

$primary-color: #22ae39 !default;

我们使用 map 可以更好的进行管理：

$color: (

default: #fff,

primary: #22ae39

);

如果哪一天，你需要新增加颜色变量值，在 map 中可以非常随意的添加：

$color: (

default: #fff,

primary: #22ae39,

negative: #d9534f

);

对于 Sass 的 map，还可以让 map 嵌套 map。其实就是 map 的某一个 key 当成 map，里面可以继续放一对或者多对 key:value：

$map: (

key1: value1,

key2: (

key-1: value-1,

key-2: value-2,

),

key3: value3

);

map 的嵌套实用性也非常的强，大家可能有碰到过换皮肤的项目，可能每一套皮肤对应的颜色蛮多的，那么使用此功能来管理颜色的变量就非常的有条理性，便于维护与管理。你可以这样使用：

$theme-color: (

default: (

bgcolor: #fff,

text-color: #444,

link-color: #39f

),

primary:(

bgcolor: #000,

text-color:#fff,

link-color: #93f

),

negative: (

bgcolor: #f36,

text-color: #fefefe,

link-color: #d4e

)

);

在一些介绍 map 的老教程中，你会看到这样的方式声明 map:

$map: (

key1 value1,

key2 value2,

key3 value3

);

或者：

$map:(

key1 value1,

key2 value2,

key3 (

key-1 value-1,

key-2 value-2

),

key4 value4

);

虽然也能编译出 CSS，但建议不这样使用。

#### Maps的函数

前面介绍了使用 map 来管理变量，但要在 Sass 中获取变量，或者对 map 做更多有意义的操作，我们必须借助于 map 的函数功能。在 Sass 中 map 自身带了七个函数：

* **map-get($map,$key)**：根据给定的 key 值，返回 map 中相关的值。
* **map-merge($map1,$map2)**：将两个 map 合并成一个新的 map。
* **map-remove($map,$key)**：从 map 中删除一个 key，返回一个新 map。
* **map-keys($map)**：返回 map 中所有的 key。
* **map-values($map)**：返回 map 中所有的 value。
* **map-has-key($map,$key)**：根据给定的 key 值判断 map 是否有对应的 value 值，如果有返回 true，否则返回 false。
* **keywords($args)**：返回一个函数的参数，这个参数可以动态的设置 key 和 value。

接下来的小节中，我们来简单的了解这些函数的具体使用以及所具有的功能。

#### Maps的函数-map-get($map,$key)

map-get($map,$key) 函数的作用是根据 $key 参数，返回 $key 在 $map 中对应的 value 值。如果 $key 不存在 $map中，将返回 null 值。此函数包括两个参数：

* $map：定义好的 map。
* $key：需要遍历的 key。

来看一个简单的示例，假设定义了一个 $social-colors 的 map:

$social-colors: (

dribble: #ea4c89,

facebook: #3b5998,

github: #171515,

google: #db4437,

twitter: #55acee

);

假设要获取 facebook 键值对应的值 #3b5998，我们就可以使用 map-get() 函数来实现：

.btn-dribble{

color: map-get($social-colors,facebook);

}

编译出来的CSS：

.btn-dribble {

color: #3b5998;

}

我们来看另一种情况，假设 $social-colors 这个 map 没有 $weibo 这个 key:

.btn-weibo{

font-size: 12px;

color: map-get($social-colors,weibo);

}

此时编译出来的是CSS：

.btn-weibo {

font-size: 12px;

}

从 编译出来的 CSS 可以得知，如果 $key 不在 $map 中，不会编译出 CSS，其实在 Sass 中，map-get($social- colors,weibo) 返回了一个 null 值。但在编译出来的 CSS 中，你只知道他没有编译出样式，而且在命令终端编译时，也没有任何错误或者警告信息。说实话，你并不知道他为什么编译不出来样式，或者说他已返回了值为 null。体验不强，也不好排错。其实如果我们自定义一个函数，另外加个判断，那就截然不同。

#### Maps的函数-map-has-key($map,$key)

map-has-key($map,$key) 函数将返回一个布尔值。当 $map 中有这个 $key，则函数返回 true，否则返回 false。

前面的示例，当 $key 不在 $map 中时，使用 map-get($map,$key) 函数将返回一个 null 值。但对于开发人员，并看不到任何提示信息。如果使用 map-has-key($map,$key) 函数就可以改变这一状态。我们来看一个简单的示例。

@if map-has-key($social-colors,facebook){

.btn-facebook {

color: map-get($social-colors,facebook);

}

} @else {

@warn "No color found for faceboo in $social-colors map. Property ommitted."

}

编译出来：

.btn-fackbook{

color: #3b5998;

}

上面看到的示例是 facebook 这个 key 已存在 $social-colors 这个 map 当中。所以能正常编译。如果你手误，将 facebook 输错了：

@if map-has-key($social-colors,faceboo){

.btn-facebook {

color: map-get($social-colors,facebook);

}

} @else {

@warn "No color found for faceboo in $social-colors map. Property ommitted."

}

这个时候，你编译出来的 CSS 代码中，不会有新代码添加，但在命令终端可以看到提示信息：

WARNING: No color found for faceboo in $social-colors map. Property ommitted.

on line 25 of test.scss

是不是非常的友好。但总觉得这样写是傻傻的，总不可能每获取一个 key 都写一个 @if 语句吧。其实不用这么复杂，我们可以自定义一个函数，比如 colors():

@function colors($color){

@if not map-has-key($social-colors,$color){

@warn "No color found for `#{$color}` in $social-colors map. Property omitted.";

}

@return map-get($social-colors,$color);

}

有了这个函数之后，我们就可以这样使用

.btn-dribble {

color: colors(dribble);

}

.btn-facebook {

color: colors(facebook);

}

.btn-github {

color: colors(github);

}

.btn-google {

color: colors(google);

}

.btn-twitter {

color: colors(twitter);

}

.btn-weibo {

color: colors(weibo);

}

编译出来的 CSS:

.btn-dribble {

color: #ea4c89;

}

.btn-facebook {

color: #3b5998;

}

.btn-github {

color: #171515;

}

.btn-google {

color: #db4437;

}

.btn-twitter {

color: #55acee;

}

同时你不难发现，命令终端提示信息：

WARNING: No color found for `weibo` in $social-colors map. Property omitted.

on line 13 of test.scss

那是在 $social-colors 这个 map 中没有 weibo 这个 key。是不是很有意思。

当然，如果你对 Sass 的指令熟悉的话，上面编译出来的 CSS 可以使用 @each：

@each $social-network,$social-color in $social-colors {

.btn-#{$social-network} {

color: colors($social-network);

}

}

#### Maps的函数-map-keys($map)

map-keys($map) 函数将会返回 $map 中的所有 key。这些值赋予给一个变量，那他就是一个列表。如：

map-keys($social-colors);

其返回的值为：

"dribble","facebook","github","google","twitter"

换句话说：

$list: map-keys($social-colors);

相当于：

$list:"dribble","facebook","github","google","twitter";

这个时候，就可以配合 Sass 的 list 做很多事情。

上面的示例，可以做通过 map-keys($map) 来做一个修改：

@function colors($color){

$names: map-keys($social-colors);

@if not index($names,$color){

@warn "Waring: `#{$color} is not a valid color name.`";

}

@return map-get($social-colors,$color);

}

上面代码中最不同之处，我们使 用map-key s将 $social-colors 这个 map 的所有 key 取出，并赋予给一个名 为 $names 的列表。然后通过 index($names,$color) 返回 $color 在 $names 位置，如果这个位置不存在，将返回提示信息，如果存在将返回正确的值。

.btn-weibo{

color: colors(weibo);

}

例如，weibo 不在 $social-color s中，那么不会编译出 CSS，而且在命令终端同样会有提示信息：

WARNING: Waring: `weibo is not a valid color name.`

on line 27 of test.scss

同样，也可以通过 @each 或者 @for 遍历出所有值：

@each:

@each $name in map-keys($social-colors){

.btn-#{$name}{

color: colors($name);

}

}

@for:

@for $i from 1 through length(map-keys($social-colors)){

.btn-#{nth(map-keys($social-colors),$i)} {

color: colors(nth(map-keys($social-colors),$i));

}

}

虽然使用的方法不一样，但最终得到的 CSS 是一样的：

.btn-dribble {

color: #ea4c89;

}

.btn-facebook {

color: #3b5998;

}

.btn-github {

color: #171515;

}

.btn-google {

color: #db4437;

}

.btn-twitter {

color: #55acee;

}

#### Maps的函数-map-values($map)、map-merge($map1,$map2)

**map-values($map)**

map-values($map) 函数类似于 map-keys($map) 功能，不同的是 map-values($map )获取的是 $map 的所有 value 值，可以说也将是一个列表。而且，map-values($map) 中如果有相同的 value 也将会全部获取出来。

如前面的示例，使用：

map-values($social-colors)

将会返回：

#ea4c89,#3b5998,#171515,#db4437,#55acee

值与值之前同样用逗号分隔。

map-merge($map1,$map2)

**map-merge($map1,$map2)**

map-merge($map1,$map2) 函数是将 $map1 和 $map2 合并，然后得到一个新的 $map。如果你要快速将新的值插入到 $map 中的话，这种方法是最佳方法。假设我们有两个 $map:

$color: (

text: #f36,

link: #f63,

border: #ddd,

backround: #fff

);

$typo:(

font-size: 12px,

line-height: 1.6

);

如果希望将这两个 $map 合并成一个 map，我们只要这样做：

$newmap: map-merge($color,$typo);

将会生成一个新的 map:

$newmap:(

text: #f36,

link: #f63,

border: #ddd,

background: #fff,

font-size: 12px,

line-height: 1.6

);

这样你就可以借助 map-get( ) 等函数做其他事情了。

不过有一点需要注意，如果 $map1 和 $map2 中有相同的 $key 名，那么将 $map2 中的 $key 会取代 $map1 中的：

$color: (

text: #f36,

link: #f63,

border: #ddd,

backround: #fff

);

$typo:(

font-size: 12px,

line-height: 1.6,

border: #ccc,

background: #000

);

执行：

$newmap: map-merge($color,$typo);

得到的新 map:

$newmap:(

text: #f36,

link: #f63,

font-size: 12px,

line-height: 1.6,

border: #ccc,

background: #000

);

#### Maps的函数-map-remove($map,$key)、keywords($args)

**map-remove($map,$key)**

map-remove($map,$key) 函数是用来删除当前 $map 中的某一个 $key，从而得到一个新的 map。其返回的值还是一个 map。他并不能直接从一个 map 中删除另一个 map，仅能通过删除 map 中的某个 key 得到新 map。如：

$map:map-remove($social-colors,dribble);

返回的是一个新 map:

$map:(

facebook: #3b5998,

github: #171515,

google: #db4437,

twitter: #55acee

);

如果删除的 key 并不存在于 $map 中，那么 map-remove() 函数返回的新 map 和以前的 map 一样。

$map:map-remove($social-colors,weibo);

返回的值：

$map: (

dribble: #ea4c89,

facebook: #3b5998,

github: #171515,

google: #db4437,

twitter: #55acee

);

keywords($args)

**keywords($args)**

keywords($args) 函数可以说是一个动态创建 map 的函数。可以通过混合宏或函数的参数变创建 map。参数也是成对出现，其中 $args 变成 key(会自动去掉$符号)，而 $args 对应的值就是value。

@mixin map($args...){

@debug keywords($args);

}

@include map(

$dribble: #ea4c89,

$facebook: #3b5998,

$github: #171515,

$google: #db4437,

$twitter: #55acee

);

在命令终端可以看到一个输入的 @debug 信息：

DEBUG: (dribble: #ea4c89, facebook: #3b5998, github: #171515, google: #db4437, twitter: #55acee)