

# CSS 框架之 Sass 全面解析

讲师： 风舞烟

www.ibEIFeng.com

# 目录





目录 .....	2
第二部分 CSS 框架 – SASS 全面解析 .....	4
教学目标.....	4
教学内容.....	4
一、Sass 入门.....	4
1.1、简介 .....	4
1.2、安装.....	5
第一步：安装 Ruby .....	6
第二步：安装 SASS .....	10
1.3、快速上手.....	12
1.4、使用 Sass.....	13
二、Sass 语法.....	15
0. Sass 文件后缀名.....	15
1. 使用变量.....	15
1-1. 变量声明 .....	16
1-2. 变量引用 .....	17
1-3. 变量名 .....	18
2. 嵌套 CSS 规则.....	19
2-1. 父选择器的标识符& .....	21
2-2. 群组选择器的嵌套.....	23
2-3. 子组合选择器和同层组合选择器：>、+和~ .....	25
2-4. 嵌套属性 .....	26
3. 导入 Sass 文件.....	28
3-1. 使用 SASS 部分文件.....	29
3-2. 默认变量值 .....	30
3-3. 嵌套导入 .....	31
3-4. 原生的 CSS 导入 .....	32
4. 静默注释.....	33
5. 混合器.....	35
5-1. 何时使用混合器 .....	37
5-2. 混合器中的 CSS 规则 .....	38
5-3. 给混合器传参 .....	39
5-4. 默认参数值 .....	41
6. 使用选择器继承来精简 CSS.....	42
6-0. 什么是 Sass 继承 .....	42
6-1. 如何使用 Sass 继承 .....	43
6.1.1. 它是如何工作 .....	45
6.1.2. 继承复杂的选择器.....	46
6.1.3. 继承多个选择器.....	47
6.1.4. 链型继承.....	50

---

6.1.5. 继承的局限性 .....	52
6.1.6. 继承交叉合并选择 .....	53
6.1.7. 继承带%的东西 .....	54
6.1.8. 继承在指令中的作用域 .....	55
6-2. 何时使用继承 .....	57
6-3. 使用继承的最佳实践 .....	58
7. 总结 .....	58
三、Sass 函数 .....	59
3.1、Sass 内置函数 .....	59
3.1.1、字符串函数 .....	60
3.1.2、数字函数 .....	64
3.1.3、列表函数 .....	70
3.1.4、三元函数 .....	78
3.1.5、颜色函数 .....	83
3.2、Sass 自定义函数 .....	89
四、Sass 高级用法 .....	90
4.1 条件语句 .....	90
4.2 循环语句 .....	91
五、利用 Koala 编译 Sass .....	92

## 第二部分 CSS 框架 – SASS 全面解析

### 教学目标

-  目标一、SASS 简介、如何下载并使用、快速上手案例
-  目标二、掌握 SASS 语言的特性
-  目标三、掌握 SASS 函数
-  目标四、了解如何在服务器端、客户端和第三方环境中使用 SASS

### 教学内容

#### 一、Sass 入门

##### 1.1、简介

作为前端开发人员，你肯定对 css 很熟悉，但是你知道 css 可以自定义吗？大家都知道，js 中可以自定义变量，css 仅仅是一个标记语言，不是编程语言，因此不可以自定义变量，不可以引用等等。面对这些问题，我们现在来引进一个 SASS，简单的说，他是 css 的升级版，他可以自定义变量，可以有 if 语句，还可以嵌套等等，很神奇吧！那下面我们就来介绍这个神奇的 SASS！



```
.scss .sass

$blue: #3bbfce;
$margin: 16px;

.content-navigation {
  border-color: $blue;
  color:
    darken($blue, 9%);
}

.border {
  padding: $margin / 2;
  margin: $margin / 2;
  border-color: $blue;
}
```

SASS 英文意思 SASS 是 Syntactically Awesome Stylesheets Sass，最早由 Hampton Catlin 开发和设计。一种帮助你简化 CSS 工作流程的方式，帮助你更容易的维护和开发 CSS 内容。

官网：<http://sass-lang.com/>

## CSS with superpowers



Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.

Sass 这个世界上最成熟、稳定和强大的专业级 CSS 扩展语言！

### 1.2、安装

SASS 是 Ruby 语言写的，但是两者的语法没有关系。不懂 Ruby，照样使用。只是必须先安装 Ruby，然后再安装 SASS。

Windows 下安装步骤如下：

## 第一步：安装 Ruby

Step1、直接去 ruby 的官网 ( <https://www.ruby-lang.org/en/> ) 上下载 rubyinstall 的安装包，直接选择现在最新的 2.2，往下拉看到 devkit 的包一起下载，没有说明对应 2.2 下载哪个也拿最近的。








也可以直接找到 rubyinstaller,地址如下：

<http://rubyinstaller.org/downloads/>

### RubyInstallers


### Archives»

Not sure what version to download? Please read the right column for recommendations.


-  Ruby 2.2.3
-  Ruby 2.2.3 (x64)
-  Ruby 2.1.7
-  Ruby 2.1.7 (x64)
-  Ruby 2.0.0-p647
-  Ruby 2.0.0-p647 (x64)
-  Ruby 1.9.3-p551

### DEVELOPMENT KIT


For use with Ruby 1.8.7 and 1.9.3:

 DevKit-tdm-32-4.5.2-20111229-1559-sfx.exe

For use with Ruby 2.0 and above (32bits version only):

 DevKit-mingw64-32-4.7.2-20130224-1151-sfx.exe

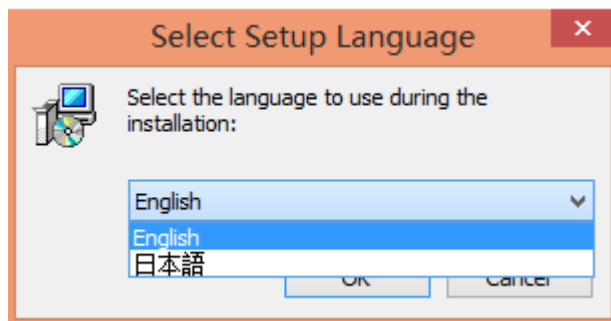
For use with Ruby 2.0 and above (x64 - 64bits only)

 DevKit-mingw64-64-4.7.2-20130224-1432-sfx.exe

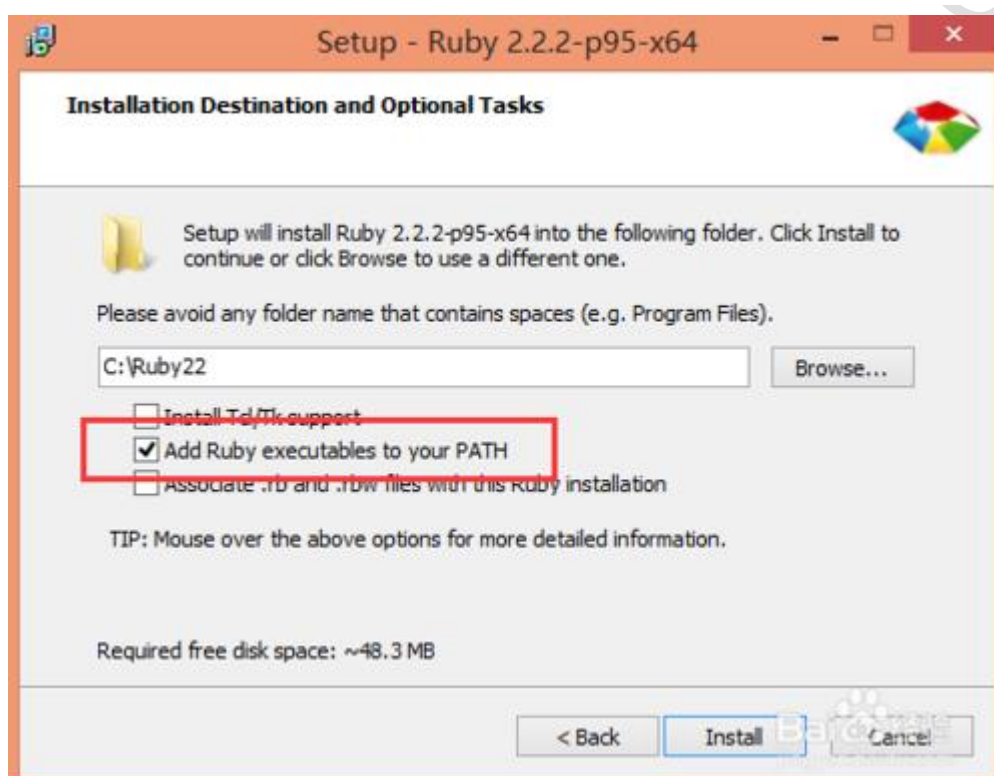
### MD5 & SHA256 Checksums

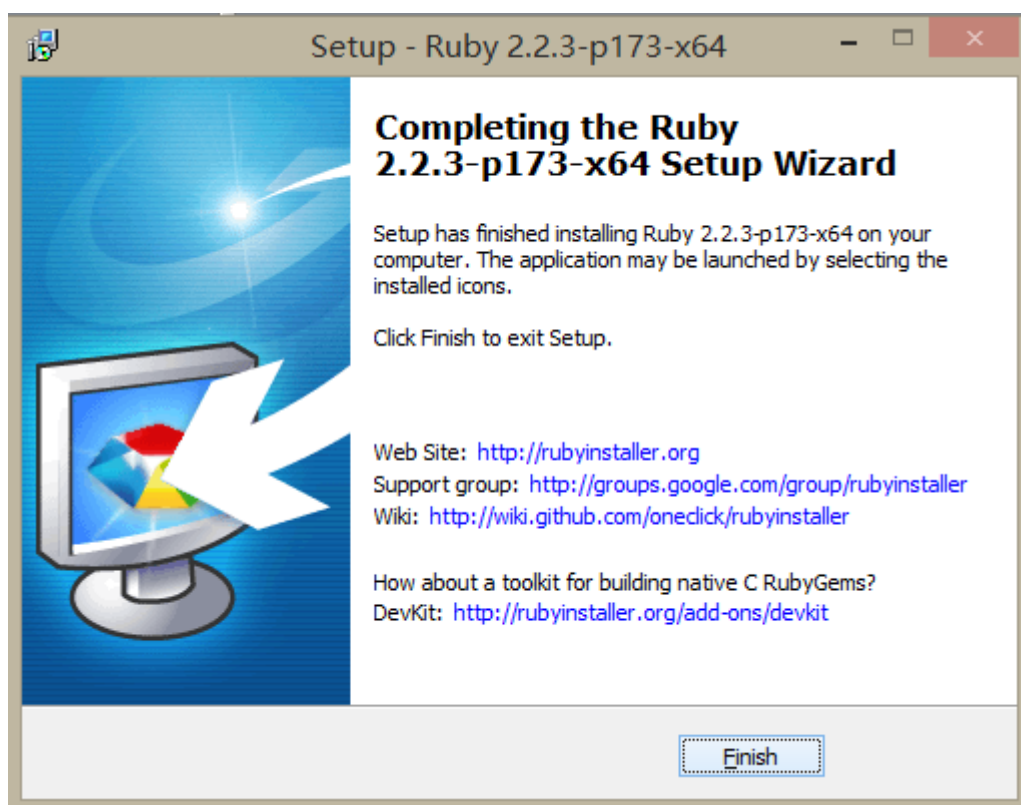
## Step2、安装 ruby

点击安装，选择安装语言



选择安装目录，顺便勾选上添加到环境变量



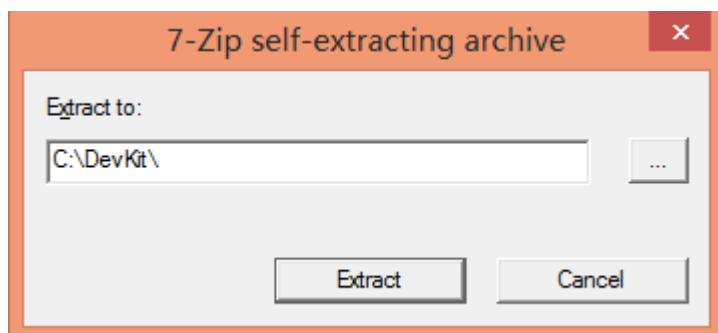


安装完成，打开命令行，`ruby -v`，检验是否安装成功

```
C:\Users\fwy-tech>ruby -v
ruby 2.2.3p173 (2015-08-18 revision 51636) [x64-mingw32]
C:\Users\fwy-tech>
```

Step3、安装 devkit

它是个压缩包，直接解压到指定的文件夹就好了




Step4、配置

创建 **config.yml** 文件，打开命令行，进入 devkit 的解压目录



ruby dk.rb init 初始化创建 config.yml 文件



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.2.9200]
(c) 2012 Microsoft Corporation。保留所有权利。

C:\Users\samir1216>ruby -v
ruby 2.2.2p95 (2015-04-13 revision 50295) [x64-mingw32]

C:\Users\samir1216>cd c:\DevKit

c:\DevKit>ruby dk.rb init
[INFO] found RubyInstaller v2.1.5 at D:/soft/Ruby21-x64
[INFO] found RubyInstaller v2.2.2 at C:/Ruby22

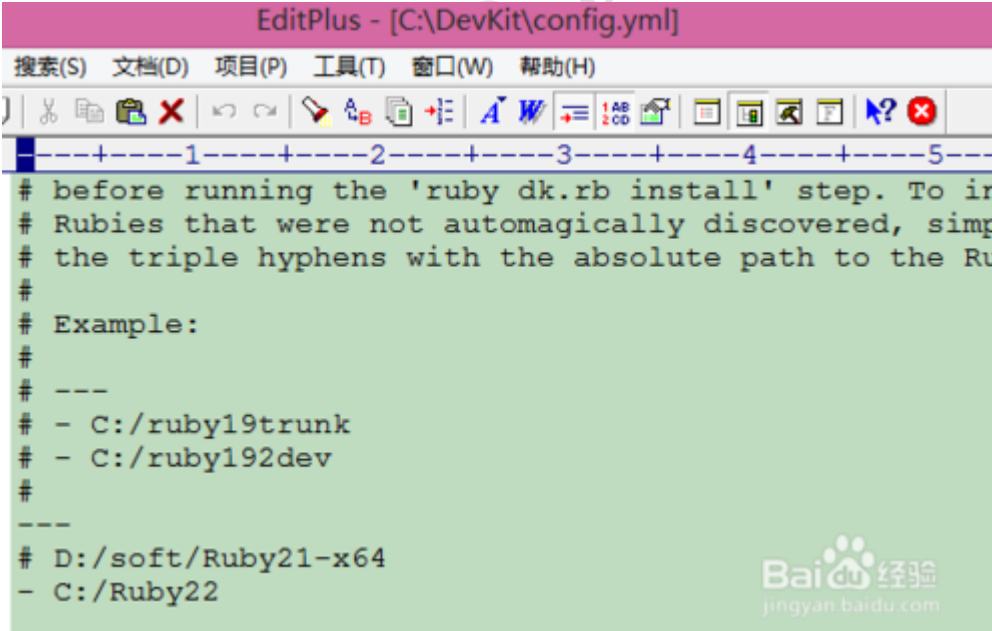
Initialization complete! Please review and modify the auto-generated
'config.yml' file to ensure it contains the root directories to all
of the installed Rubies you want enhanced by the DevKit.

c:\DevKit>
```

## 查看 config.yml

文件最后是否自动加上- C:/Ruby22，如果没有手动加上就可以了

如果以前有装过的，可以先注释掉以前的



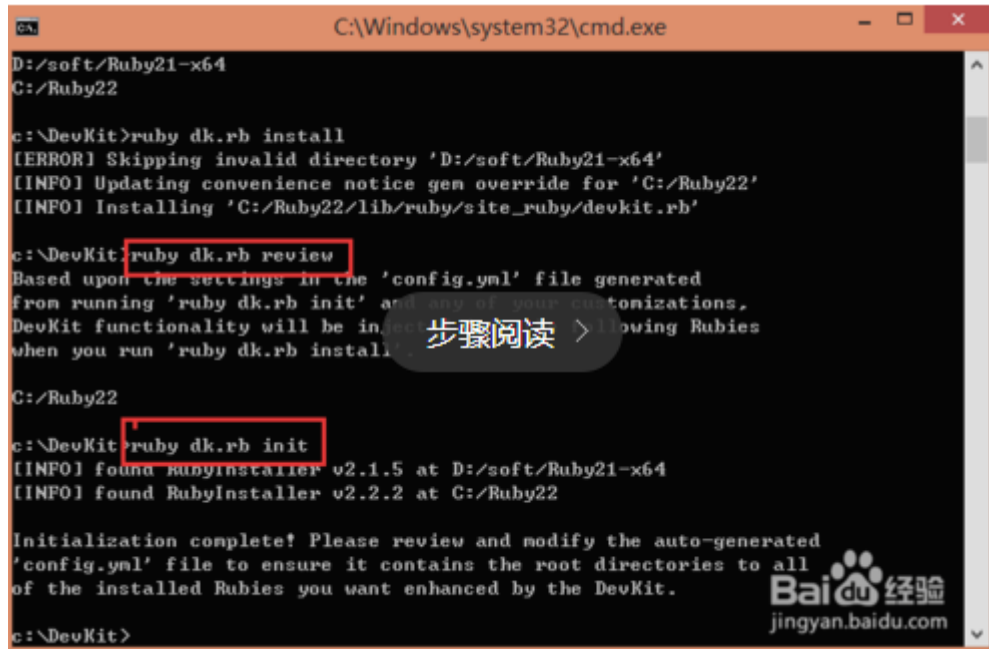
```
EditPlus - [C:\DevKit\config.yml]
搜索(S) 文档(D) 项目(P) 工具(T) 窗口(W) 帮助(H)

---+---1---+---2---+---3---+---4---+---5---
# before running the 'ruby dk.rb install' step. To in
# Rubies that were not automagically discovered, simp
# the triple hyphens with the absolute path to the Ru
#
# Example:
#
# ---
# - C:/ruby19trunk
# - C:/ruby192dev
#
# ---
# D:/soft/Ruby21-x64
- C:/Ruby22
```

## 回到命令行，执行审查和安装

- ruby dk.rb install

- ruby dk.rb review
- ruby dk.rb init



```
C:\Windows\system32\cmd.exe
D:/soft/Ruby21-x64
C:/Ruby22

c:\DevKit>ruby dk.rb install
[ERROR] Skipping invalid directory 'D:/soft/Ruby21-x64'
[INFO] Updating convenience notice gem override for 'C:/Ruby22'
[INFO] Installing 'C:/Ruby22/lib/ruby/site_ruby/devkit.rb'

c:\DevKit>ruby dk.rb review
Based upon the settings in the 'config.yml' file generated
from running 'ruby dk.rb init' and any of your customizations,
DevKit functionality will be injected into the following Rubies
when you run 'ruby dk.rb install'.

C:/Ruby22

c:\DevKit>ruby dk.rb init
[INFO] found RubyInstaller v2.1.5 at D:/soft/Ruby21-x64
[INFO] found RubyInstaller v2.2.2 at C:/Ruby22

Initialization complete! Please review and modify the auto-generated
'config.yml' file to ensure it contains the root directories to all
of the installed Rubies you want enhanced by the DevKit.

c:\DevKit>
```

到此，Ruby 全部安装配置结束！

## 第二步：安装 SASS

1. **打开你的终端或命令提示符。**在 Mac 上的终端。应用程序默认安装。它位于你的“工具”文件夹中。在 Windows 上,运行 cmd.
2. **Sass 安装。**Ruby 使用 Gems 来管理它的各种包的代码像 Sass。在你打开终端窗口类型:

```
gem install sass
```

注意：此时，有可能会出现如下错误：

```
C:\>gem install sass
ERROR: Could not find a valid gem 'sass' (>= 0), here is why:
      Unable to download data from https://rubygems.org/ - Errno::ECONNABORT
ED: An established connection was aborted by the software in your host machine.
- SSL_connect (https://api.rubygems.org/specs.4.8.gz)
```

**原因：**应该是因为 GFW 的原因或者说实在是不稳定！

**解决办法：**使用万能淘宝的镜像站 <http://ruby.taobao.org/>

淘宝 ruby 资源站是完全的镜像复制，而且十五分钟复制更新一次，连接速度很快很稳定

**步骤如下：**

1、把 <http://ruby.taobao.org/> 加入 gem sources;

```
$ gem sources --add https://ruby.taobao.org/ --remove
```

```
https://rubygems.org/
```

```
$ gem sources -l
```

如下图所示：

```
C:\Users\fwy-tech>gem sources --add https://ruby.taobao.org/ --remove https://ru
bygems.org/
https://ruby.taobao.org/ added to sources
source https://rubygems.org/ not present in cache

C:\Users\fwy-tech>gem sources -l
*** CURRENT SOURCES ***

https://ruby.taobao.org/
```

请确保只有 [ruby.taobao.org](http://ruby.taobao.org/)

2、再输入一次 `gem install sass` 就好了

```
C:\Users\fwy-tech>gem install sass
Fetching: sass-3.4.19.gem (100%)
Successfully installed sass-3.4.19
Parsing documentation for sass-3.4.19
Installing ri documentation for sass-3.4.19
Done installing documentation for sass after 9 seconds
1 gem installed
```

3、最终检查。现在,您应该安装了 SASS, 可以通过以下命令检查。在终端应用程序中  
你可以输入:

```
sass -v
```

```
C:\Users\fwy-tech>sass -v
Sass 3.4.19 (Selective Steve)
```

它应该返回 Sass 3.4.15 (Selective Steve)。恭喜你！你已经成功安装。

### 1.3、快速上手

Step1、准备.scss 文件 (input.scss)

```
$blue : blue; //color

.blue {
    color: $blue;
}
```

Step2、命令行执行如下命令

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

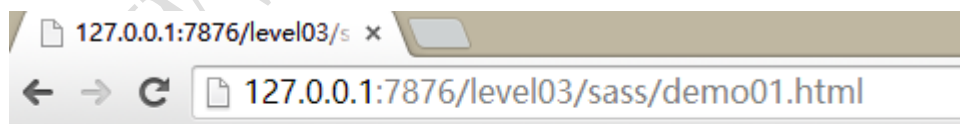
### Step3、观察输出的 output.css 内容

```
.blue { color: blue; }  
  
/*# sourceMappingURL=output.css.map */
```

### Step4、准备 demo01.html

```
<html>  
  
<head>  
  
  <link rel="stylesheet" type="text/css" href="css-source/style01.  
css">  
  
</head>  
  
<body>  
  
  <p class="blue">这是内容</p>  
  
</body>  
  
</html>
```

### Step5、运行看效果



这是内容

## 1.4、使用 Sass

Sass 有三种使用方式：命令行工具、独立的 Ruby 模块，以及包含 Ruby on

Rails 和 Merb 作为支持 Rack 的框架的插件。所有这些方式的第一步都是安装

Sass gem :

```
gem install sass
```

如果你使用的是 Windows , 就需要先安装 Ruby。

如果要在命令行中运行 Sass ,只要输入

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

你还可以命令 Sass 监视文件的改动并更新 CSS :

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

注 : Ctrl+C 可以终止监视

如果你的目录里有很多 Sass 文件 , 你还可以命令 Sass 监视整个目录 :

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

补充 : 如果你想一次性编译一个文件夹里所有 .scss 文件到另一目录中 , 相关命令如下 :

```
sass --update in:out
```

其中的 in 和 out 指的为输入输出文件夹

使用 sass --help 可以列出完整的帮助文档。

## 二、Sass 语法

### 0. Sass 文件后缀名

sass 有两种后缀名文件：一种后缀名为 sass，不使用大括号和分号；另一种就是我们这里使用的 scss 文件，这种和我们平时写的 css 文件格式差不多，使用大括号和分号。本教程中所说的所有 sass 文件都指后缀名为 scss 的文件。在此也建议使用后缀名为 scss 的文件，以避免 sass 后缀名的严格格式要求报错。

### 1. 使用变量

sass 让人们受益的一个重要特性就是它为 css 引入了变量。你可以把反复使用的 css 属性值 定义成变量，然后通过变量名来引用它们，而无需重复书写这一属性值。或者，对于仅使用过一次的属性值，你可以赋予其一个易懂的变量名，让人一眼就知道这个属性值的用途。

sass 使用\$符号来标识变量(老版本的 sass 使用!来标识变量。改成\$是多半因为!highlight-color 看起来太丑了。)，比如\$highlight-color 和\$sidebar-width。为什么选择\$ 符号呢？因为它好认、更具美感，且在 CSS 中并无他用，不会导致与现存或未来的 css 语法冲突。

## 1-1. 变量声明

sass 变量的声明和 css 属性的声明很像：

```
$highlight-color: #F90;
```

这意味着变量\$highlight-color 现在的值是#F90。任何可以用作 css 属性值的赋值都可以用作 sass 的变量值，甚至是以空格分割的多个属性值，如\$basic-border: 1px solid black;，或以逗号分割的多个属性值，如\$plain-font: "Myriad Pro", Myriad, "Helvetica Neue", Helvetica, "Liberation Sans", Arial 和 sans-serif; sans-serif;。这时变量还没有生效，除非你引用这个变量——我们很快就会了解如何引用。

与 CSS 属性不同，变量可以在 css 规则块定义之外存在。当变量定义在 css 规则块内，那么该变量只能在此规则块内使用。如果它们出现在任何形式的{...}块中（如 @media 或者 @font-face 块），情况也是如此：

```
$nav-color: #F90;

nav { $width: 100px; width: $width; color: $nav-color; }

//编译后

nav { width: 100px; color: #F90; }
```

在这段代码中，\$nav-color 这个变量定义在了规则块外边，所以在这个样式表中都可以像 nav 规则块那样引用它。\$width 这个变量定义在了 nav 的{ }规则块内，所以它只能在 nav 规则块 内使用。这意味着是你可以在样式表的其他地方定义和使用\$width 变量，不会对这里造成影响。



只声明变量其实没啥用处，我们最终的目的还是使用它们。上例已介绍了如何使用 `$nav-color` 和 `$width` 这两个变量，接下来我们将进一步探讨变量的使用方法。

## 1-2. 变量引用

凡是 css 属性的标准值( 比如说 `1px` 或者 `bold` )可存在的地方，变量就可以使用。css 生成时，变量会被它们的值所替代。之后，如果你需要一个不同的值，只需要改变这个变量的值，则所有引用此变量的地方生成的值都会随之改变。

```
$highlight-color: #F90;

.selected { border: 1px solid $highlight-color; }

//编译后

.selected { border: 1px solid #F90; }
```

看上边示例中的 `$highlight-color` 变量，它被直接赋值给 `border` 属性，当这段代码被编译输出 css 时，`$highlight-color` 会被 `#F90` 这一颜色值所替代。产生的效果就是给 `selected` 这个类一条 1 像素宽、实心且颜色值为 `#F90` 的边框。

在声明变量时，变量值也可以引用其他变量。当你通过粒度区分，为不同的值取不同名字时，这相当有用。下例在独立的颜色值粒度上定义了一个变量，且在另一个更复杂的边框值粒度上也定义了一个变量：

```
$highlight-color: #F90;

$highlight-border: 1px solid $highlight-color;

.selected {

    border: $highlight-border;
```

```
}  
  
//编译后  
  
.selected {  
    border: 1px solid #F90;  
}
```

这里，`$highlight-border` 变量的声明中使用了 `$highlight-color` 这个变量。产生的效果就跟你直接为 `border` 属性设置了一个 `1px $highlight-color solid` 的值是一样的。最后，我们来了解一下变量命名的实用技巧，以结束关于变量的介绍。

### 1-3. 变量名

sass 的变量名可以与 css 中的属性名和选择器名称相同，包括中划线和下划线。这完全取决于个人的喜好，有些人喜欢使用中划线来分隔变量中的多个词（如 `$highlight-color`），而有些人喜欢使用下划线（如 `$highlight_color`）。使用中划线的方式更为普遍，这也是 compass 和本文都用的方式。

不过，sass 并不想强迫任何人一定使用中划线或下划线，所以这两种用法相互兼容。用中划线声明的变量可以使用下划线的方式引用，反之亦然。这意味着即使 compass 选择用中划线的命名方式，这并不影响你在使用 compass 的样式中用下划线的命名方式进行引用：

```
$link-color: blue;  
  
a {
```

```
color: $link_color;

}

//编译后

a {

    color: blue;

}
```

在上例中，\$link-color 和 \$link\_color 其实指向的是同一个变量。实际上，在 sass 的大多数地方，中划线命名的内容和下划线命名的内容是互通的，除了变量，也包括对混合器和 Sass 函数的命名。但是在 sass 中纯 css 部分不互通，比如类名、ID 或属性名。

尽管变量自身提供了很多有用的地方，但是 sass 基于变量提供的更为强大的工具才是我们关注的焦点。只有当变量与 sass 的其他特性一起使用时，才能发挥其全部的潜能。

## 2. 嵌套 CSS 规则

css 中重复写选择器是非常恼人的。如果要写一大串指向页面中同一块的样式时，往往需要一遍又一遍地写同一个 ID：

```
#content article h1 { color: #333 }

#content article p { margin-bottom: 1.4em }

#content aside { background-color: #EEE }
```

像这种情况，sass 可以让你只写一遍，且使样式可读性更高。在 Sass 中，你可以像俄罗斯套娃那样在规则块中嵌套规则块。sass 在输出 css 时会帮你把这些嵌套规则处理好，避免你的重复书写。

```
#content {  
  article {  
    h1 { color: #333 }  
    p { margin-bottom: 1.4em }  
  }  
  aside { background-color: #EEE }  
}  
  
/* 编译后 */  
  
#content article h1 { color: #333 }  
  
#content article p { margin-bottom: 1.4em }  
  
#content aside { background-color: #EEE }
```

一个给定的规则块，既可以像普通的 CSS 那样包含属性，又可以嵌套其他规则块。当你同时要为一个容器元素及其子元素编写特定样式时，这种能力就非常有用。

```
#content {  
  background-color: #f5f5f5;  
  aside { background-color: #eee }  
}
```

容器元素的样式规则会被单独抽离出来，而嵌套元素的样式规则会像容器元素没

有包含任何属性时那样被抽离出来。

```
#content { background-color: #f5f5f5 }  
  
#content aside { background-color: #eee }
```

多数情况下这种简单的嵌套都没问题，但是有些场景下不行，比如你想要在嵌套的选择器里边立刻应用一个类似于 `:hover` 的伪类。为了解决这种以及其他情况，`sass` 提供了一个特殊结构 `&`。

## 2-1. 父选择器的标识符 `&`

一般情况下，`sass` 在解开一个嵌套规则时就会把父选择器（`#content`）通过一个空格连接到子选择器的前边（`article` 和 `aside`）形成 `#content article` 和 `#content aside`）。这种在 CSS 里边被称为后代选择器，因为它选择 ID 为 `content` 的元素内所有命中选择器 `article` 和 `aside` 的元素。但在有些情况下你却不会希望 `sass` 使用这种后代选择器的方式生成这种连接。

最常见的一种情况是当你为链接之类的元素写 `:hover` 这种伪类时，你并不希望以后代选择器的方式连接。比如说，下面这种情况 `sass` 就无法正常工作：

```
article a {  
  color: blue;  
  :hover { color: red }  
}
```

这意味着 `color: red` 这条规则将会被应用到选择器 `article a :hover`，`article` 元素内链接的所有子元素在被 `hover` 时都会变成红色。这是不正确的！你想把这条规则

应用到超链接自身，而后代选择器的方式无法帮你实现。

解决之道为使用一个特殊的 sass 选择器，即父选择器。在使用嵌套规则时，父选择器能对于嵌套规则如何解开提供更好的控制。它就是一个简单的&符号，且可以放在任何一个选择器可出现的地方，比如 h1 放在哪，它就可以放在哪。

```
article a {  
  color: blue;  
  &:hover { color: red }  
}
```

当包含父选择器标识符的嵌套规则被打开时，它不会像后代选择器那样进行拼接，而是&被父选择器直接替换：

```
article a { color: blue }  
article a:hover { color: red }
```

在为父级选择器添加：hover 等伪类时，这种方式非常有用。同时父选择器标识符还有另外一种用法，你可以在父选择器之前添加选择器。举例来说，当用户在使用 IE 浏览器时，你会通过 JavaScript 在<body>标签上添加一个 ie 的类名，为这种情况编写特殊的样式如下：

```
#content aside {  
  color: red;  
  body.ie & { color: green }  
}  
  
/*编译后*/
```

```
#content aside {color: red};  
  
body.ie #content aside { color: green }
```

sass 在选择器嵌套上是非常智能的，即使是带有父选择器的情况。当 sass 遇到群组选择器（由多个逗号分隔开的选择器形成）也能完美地处理这种嵌套。

## 2-2. 群组选择器的嵌套

在 CSS 里边，选择器 `h1`h2` 和 `h3` 会同时命中 `h1` 元素、`h2` 元素和 `h3` 元素。与此类似，`.button button` 会命中 `button` 元素和类名为 `.button` 的元素。这种选择器称为群组选择器。群组选择器的规则会对命中群组中任何一个选择器的元素生效。

```
.button, button {  
  margin: 0;  
}
```

当看到上边这段代码时，你可能还没意识到会有重复性的工作。但会很快发现：如果你需要在一个特定的容器元素内对这样一个群组选择器进行修饰，情况就不同了。css 的写法会让你在群组选择器中的每一个选择器前都重复一遍容器元素的选择器。

```
.container h1, .container h2, .container h3 { margin-bottom: .8em }
```

非常幸运，sass 的嵌套特性在这种场景下也非常有用。当 sass 解开一个群组选择器规则内嵌的规则时，它会把每一个内嵌选择器的规则都正确地解出来：

```
.container {  
  h1, h2, h3 {margin-bottom: .8em}  
}
```

首先 sass 将 `.container` 和 `h1`、`.container` 和 `h2`、`.container` 和 `h3` 分别组合，然后将三者重新组合成一个群组选择器，生成你前边看到的普通 css 样式。对于内嵌在群组选择器内的嵌套规则，处理方式也一样：

```
nav, aside {  
  a {color: blue}  
}
```

首先 sass 将 `nav` 和 `a`、`aside` 和 `a` 分别组合，然后将二者重新组合成一个群组选择器：

```
nav a, aside a {color: blue}
```

处理这种群组选择器规则嵌套上的强大能力，正是 sass 在减少重复敲写方面的贡献之一。尤其在当嵌套级别达到两层甚至三层以上时，与普通的 css 编写方式相比，只写一遍群组选择器大大减少了工作量。

有利必有弊，你需要特别注意群组选择器的规则嵌套生成的 css。虽然 sass 让你的样式表看上去很小，但实际生成的 css 却可能非常大，这会降低网站的速度。



## 2-3. 子组合选择器和同层组合选择器：>、+和~

上边这三个组合选择器必须和其他选择器配合使用，以指定浏览器仅选择某种特定上下文中的元素。

```
article section { margin: 5px }  
article > section { border: 1px solid #ccc }
```

子组合选择器> 选择一个元素的直接子元素。上例中，第一个选择器会选择 article 下的所有命中 section 选择器的元素。第二个选择器只会选择 article 下紧跟着的子元素中命中 section 选择器的元素。

在下例中，你可以用同层相邻组合选择器+选择 header 元素后紧跟的 p 元素：

```
header + p { font-size: 1.1em }
```

你也可以用同层全体组合选择器~，选择所有跟在 article 后的同层 article 元素，不管它们之间隔了多少其他元素

```
article ~ article { border-top: 1px dashed #ccc }
```

这些组合选择器可以毫不费力地应用到 sass 的规则嵌套中。可以把它们放在外层选择器后边，或里层选择器前边：

```
article {  
  ~ article { border-top: 1px dashed #ccc }  
  > section { background: #eee }  
  dl > {  
    dt { color: #333 }
```

```
dd { color: #555 }  
  
}  
  
nav + & { margin-top: 0 }  
  
}
```

sass 会如你所愿地将这些嵌套规则——解开组合在一起：

```
article ~ article { border-top: 1px dashed #ccc }  
  
article > section { background: #eee }  
  
article dl > dt { color: #333 }  
  
article dl > dd { color: #555 }  
  
nav + article { margin-top: 0 }
```

在 sass 中，不仅仅 css 规则可以嵌套，对属性进行嵌套也可以减少很多重复性的工作。

## 2-4. 嵌套属性

sass 中，除了 CSS 选择器，属性也可以进行嵌套。尽管编写属性涉及的重复不像编写选择器那么糟糕，但是要反复写 border-style`border-width`border-color 以及 border-\*等也是非常烦人的。在 sass 中，你只需敲写一遍 border：

```
nav {  
  
  border: {  
  
    style: solid;
```

```
width: 1px;
```

```
color: #ccc;

}

}
```

属性的规则是这样的：**把属性名从中划线-的地方断开，在根属性后边添加一个冒号:，紧跟一个{}块，把子属性部分写在这个{}块中。**就像 css 选择器嵌套一样，sass 会把你的子属性——解开，把根属性和子属性部分通过中划线-连接起来，最后生成的效果与你手动一遍遍写的 css 样式一样：

```
nav {

  border-style: solid;

  border-width: 1px;

  border-color: #ccc;

}
```

对于属性的缩写形式，你甚至可以像下边这样来嵌套，指明例外规则：

```
nav {

  border: 1px solid #ccc {

    left: 0px;

    right: 0px;

  }

  Background-color:#ccc;

}
```

这比下边这种同等样式的写法要好：

```
nav {  
  
    border: 1px solid #ccc;  
  
    border-left: 0px;  
  
    border-right: 0px;  
  
}
```

和选择器嵌套是非常伟大的特性，因为它们不仅大大减少了你的编写量，而且通过视觉上的缩进使你编写的样式结构更加清晰，更易于阅读和开发。

即便如此，随着你的样式表变得越来越大，这种写法也很难保持结构清晰。有时，处理这种大量样式的唯一方法就是要把它们分拆到多个文件中。sass 通过对 css 原有 @import 规则的改进直接支持了这一特性。

### 3. 导入 Sass 文件

css 有一个特别不常用的特性，即 @import 规则，它允许在一个 css 文件中导入其他 css 文件。然而，后果是只有执行到 @import 时，浏览器才会去下载其他 css 文件，这导致页面加载起来特别慢。

sass 也有一个 @import 规则，但不同的是，sass 的 @import 规则在生成 css 文件时就把相关文件导入进来。这意味着所有相关的样式被归纳到了同一个 css 文件中，而无需发起额外的下载请求。另外，所有在被导入文件中定义的变量和混合器均可在导入文件中使用。

使用 sass 的@import 规则并不需要指明被导入文件的全名。你可以省略.sass 或.scss 文件后缀（见下图）。这样，在不修改样式表的前提下，你完全可以随意修改你或别人写的被导入的 sass 样式文件语法，在 sass 和 scss 语法之间随意切换。举例来说，@import "sidebar";这条命令将把 sidebar.scss 文件中所有样式添加到当前样式表中。

接下来我们将介绍如何使用 sass 的@import 来处理多个 sass 文件。首先，我们将学习编写那些被导入的 sass 文件，因为在一个大型 sass 项目中，这样的文件是你最常编写的那一类。接着，了解集中导入 sass 文件的方法，使你的样式可重用性更高，包括声明可自定义的变量值，以及在某一个选择器范围内导入 sass 文件。最后，介绍如何在 sass 中使用 css 原生的@import 命令。

通常，有些 sass 文件用于导入，你并不希望为每个这样的文件单独地生成一个 css 文件。对此，sass 用一个特殊的约定来解决。

### 3-1. 使用 SASS 部分文件

当通过@import 把 sass 样式分散到多个文件时，你通常只想生成少数几个 css 文件。那些专门为@import 命令而编写的 sass 文件，并不需要生成对应的独立 css 文件，这样的 sass 文件称为局部文件。对此，sass 有一个特殊的约定来命名这些文件。

此约定即，sass 局部文件的文件名以下划线开头。这样，sass 就不会在编译时单独编译这个文件输出 css，而只把这个文件用作导入。当你@import 一个局部文件时，还可以不写文件的全名，即省略文件名开头的下划线。举例来说，你想导入

themes/\_night-sky.scss 这个局部文件里的变量，你只需在样式表中写

@import "themes/night-sky";。

局部文件可以被多个不同的文件引用。当一些样式需要在多个页面甚至多个项目中使用，这非常有用。在这种情况下，有时需要在你的样式表中对导入的样式稍作修改，sass 有一个功能刚好可以解决这个问题，即默认变量值。

### 3-2. 默认变量值

一般情况下，你反复声明一个变量，只有最后一处声明有效且它会覆盖前边的值。

举例说明：

```
$link-color: blue;

$link-color: red;

a {
    color: $link-color;
}
```

在上边的例子中，超链接的 color 会被设置为 red。这可能并不是你想要的结果，假如你写了一个可被他人通过@import 导入的 sass 库文件，你可能希望导入者可以定制修改 sass 库文件中的某些值。使用 sass 的!default 标签可以实现这个目的。它很像 css 属性中!important 标签的对立面，不同的是!default 用于变量，含义是：如果这个变量被声明赋值了，那就用它声明的值，否则就用这个默认值。

```
$fancybox-width: 400px !default;

.fancybox {
```

```
width: $fancybox-width;  
  
}
```

在上例中，如果用户在导入你的 sass 局部文件之前声明了一个 \$fancybox-width 变量，那么你的局部文件中对 \$fancybox-width 赋值 400px 的操作就无效。如果用户没有做这样的声明，则 \$fancybox-width 将默认为 400px。

接下来我们将学习嵌套导入，它允许只在某一个选择器的范围内导入 sass 局部文件。

### 3-3. 嵌套导入

跟原生的 css 不同，sass 允许 @import 命令写在 css 规则内。这种导入方式下，生成对应的 css 文件时，局部文件会被直接插入到 css 规则内导入它的地方。举例说明，有一个名为 \_blue-theme.scss 的局部文件，内容如下：

```
aside {  
  
  background: blue;  
  
  color: white;  
  
}
```

然后把它导入到一个 CSS 规则内，如下所示：

```
.blue-theme {@import "blue-theme"}
```

//生成的结果跟你直接在 .blue-theme 选择器内写 \_blue-theme.scss 文件的内容完全一样。

```
.blue-theme aside {
```

```
background: blue;

color: white;

}
```

被导入的局部文件中定义的所有变量和混合器，也会在这个规则范围内生效。这些变量和混合器不会全局有效，这样我们就可以通过嵌套导入只对站点中某一特定区域运用某种颜色主题或其他通过变量配置的样式。

有时，可用 css 原生的@import 机制，在浏览器中下载必需的 css 文件。sass 也提供了几种方法来达成这种需求。

### 3-4. 原生的 CSS 导入

由于 sass 兼容原生的 css，所以它也支持原生的 CSS@import。尽管通常在 sass 中使用@import 时，sass 会尝试找到对应的 sass 文件并导入进来，但在下列三种情况下会生成原生的 CSS@import，尽管这会造成浏览器解析 css 时的额外下载：

- 被导入文件的名字以.css 结尾；
- 被导入文件的名字是一个 URL 地址( 比如 <http://www.sass.hk/css/css.css> )，  
由此可用谷歌字体 API 提供的相应服务；
- 被导入文件的名字是 CSS 的 url()值。

这就是说，你不能用 sass 的@import 直接导入一个原始的 css 文件，因为 sass 会认为你想用 css 原生的@import。但是，因为 sass 的语法完全兼容 css，所以你可以把原始的 css 文件改名为.scss 后缀，即可直接导入了。



文件导入是保证 sass 的代码可维护性和可读性的重要一环。次之但亦非常重要的就是注释了。注释可以帮助样式作者记录写 sass 的过程中的想法。在原生的 css 中，注释对于其他人是直接可见的，但 sass 提供了一种方式可在生成的 css 文件中按需抹掉相应的注释。

#### 4. 静默注释

css 中注释的作用包括帮助你组织样式、以后你看自己的代码时明白为什么这样写，以及简单的样式说明。但是，你并不希望每个浏览网站源码的人都能看到所有注释。

sass 另外提供了一种不同于 css 标准注释格式 `/* ... */` 的注释语法，即静默注释，其内容不会出现在生成的 css 文件中。静默注释的语法跟 JavaScript、Java 等类 C 的语言中单行注释的语法相同，它们以 `//` 开头，注释内容直到行末。

```
body {  
  color: #333; // 这种注释内容不会出现在生成的 css 文件中  
  padding: 0; /* 这种注释内容会出现在生成的 css 文件中 */  
}
```

实际上，css 的标准注释格式 `/* ... */` 内的注释内容亦可在生成的 css 文件中抹去。当注释出现在原生 css 不允许的地方，如在 css 属性或选择器中，sass 将不知如何将其生成到对应 css 文件中的相应位置，于是这些注释被抹掉。

```
body {
```

```
padding: 1 /* 这块注释内容也不会出现在生成的 css 中 */ 0;  
}
```

注意事项，中文编译报错问题解决方案：

sass 文件编译时候使用 ruby 环境，在 xp 环境中没有任何问题，但是在 windows7 环境下无论是界面化的 koala 工具还是命令行模式的都会出现以下错误：

```
Syntax error: Invalid GBK character "\xE5" "\xE8"
```

```
on line 8 of E:\work\sass\sass\_big_box.scss
```

```
from line 16 of E:\work\sass\sass\main.scss
```

```
Use -trace for backtrace.
```

或者

```
Syntax error: Invalid GBK character "\xE5" "\xE8"
```

```
on line 2 of E:\work\sass\sass\main.scss
```

```
Use -trace for backtrace
```

解决办法：

### 一.koala 可视化编译工具

找到安装目录里面 sass-3.3.7 模块下面的 engine.rb 文件，例如下面路径：

C:\Program Files (x86)\Koala\rubygems\gems\sass-3.3.7\lib\sass

在这个文件里面 `engine.rb`，添加一行代码

```
Encoding.default_external = Encoding.find( 'utf-8')
```

放在所有的 `require XXXX` 之后即可。

## 二. 命令行工具同理

找到 `ruby` 的安装目录，里面也有 `sass` 模块，如这个路径：

C:\Ruby\lib\ruby\gems\1.9.1\gems\sass-3.3.14\lib\sass

在这个文件里面 `engine.rb`，添加一行代码（同方法 1）

```
Encoding.default_external = Encoding.find( 'utf-8')
```

放在所有的 `require XXXX` 之后即可。

你已经掌握了 `sass` 的静默注释，了解了保持 `sass` 条理性和可读性的最基本的方法：嵌套、导入和注释。现在，我们要进一步学习新特性，这样我们不但能保持条理性还能写出更好的样式。首先要介绍的内容是：使用混合器抽象你的相关样式。

## 5. 混合器

如果你的整个网站中有几处小小的样式类似（例如一致的颜色和字体），那么

使用变量来统一处理这种情况是非常不错的选择。但是当你的样式变得越来越复杂，你需要大段大段的重用样式的代码，独立的变量就没办法应付这种情况了。

你可以通过 sass 的混合器实现大段样式的重用。

混合器使用 @mixin 标识符定义。看上去很像其他的 CSS @标识符，比如说 @media 或者 @font-face。这个标识符给一大段样式赋予一个名字，这样你就可以轻易地通过引用这个名字重用这段样式。下边的这段 sass 代码，定义了一个非常简单的混合器，目的是添加跨浏览器的圆角边框。

```
@mixin rounded-corners {  
  -moz-border-radius: 5px;  
  -webkit-border-radius: 5px;  
  border-radius: 5px;  
}
```

然后就可以在你的样式表中通过 @include 来使用这个混合器，放在你希望的任何地方。@include 调用会把混合器中的所有样式提取出来放在 @include 被调用的地方。如果像下边这样写：

```
notice {  
  background-color: green;  
  border: 2px solid #00aa00;  
  @include rounded-corners;  
}  
  
//sass 最终生成:
```

```
.notice {  
  
    background-color: green;  
  
    border: 2px solid #00aa00;  
  
    -moz-border-radius: 5px;  
  
    -webkit-border-radius: 5px;  
  
    border-radius: 5px;  
  
}
```

在 `.notice` 中的属性 `border-radius`、`-moz-border-radius` 和 `-webkit-border-radius` 全部来自 `rounded-corners` 这个混合器。这一节将介绍使用混合器来避免重复。通过使用参数，你可以使用混合器把你样式中的通用样式抽离出来，然后轻松地其他地方重用。实际上，混合器太好用了，一不小心你可能会过度使用。大量的重用可能会导致生成的样式表过大，导致加载缓慢。所以，首先我们将讨论混合器的使用场景，避免滥用。

### 5-1. 何时使用混合器

利用混合器，可以很容易地在样式表的不同地方共享样式。如果你发现自己在不停地重复一段样式，那就应该把这段样式构造成优良的混合器，尤其是这段样式本身就是一个逻辑单元，比如说是一组放在一起有意义的属性。

判断一组属性是否应该组合成一个混合器，一条经验法则就是你能否为这个混合器想出一个好的名字。如果你能找到一个很好的短名字来描述这些属性修饰的样式，比如 `rounded-corners`、`fancy-font` 或者 `no-bullets`，那么往往能够构造一个合适的

混合器。如果你找不到，这时候构造一个混合器可能并不合适。

## 5-2. 混合器中的 CSS 规则

混合器中不仅可以包含属性，也可以包含 css 规则，包含选择器和选择器中的属性，如下代码：

```
@mixin no-bullets {  
  list-style: none;  
  
  li {  
    list-style-image: none;  
    list-style-type: none;  
    margin-left: 0px;  
  }  
}
```

当一个包含 css 规则的混合器通过@include 包含在一个父规则中时，在混合器中的规则最终会生成父规则中的嵌套规则。举个例子，看看下边的 sass 代码，这个例子中使用了 no-bullets 这个混合器：

```
ul.plain {  
  color: #444;  
  
  @include no-bullets;  
}
```

sass 的@include 指令会将引入混合器的那行代码替换成混合器里边的内容。最

终，上边的例子如下代码：

```
ul.plain {  
    color: #444;  
    list-style: none;  
}  
  
ul.plain li {  
    list-style-image: none;  
    list-style-type: none;  
    margin-left: 0px;  
}
```

### 5-3. 给混合器传参

混合器并不一定总得生成相同的样式。可以通过在@include 混合器时给混合器传参，来定制混合器生成的精确样式。当@include 混合器时，参数其实就可以赋值给 css 属性值的变量。如果你写过 JavaScript，这种方式跟 JavaScript 的 function 很像：

```
@mixin link-colors($normal, $hover, $visited) {  
    color: $normal;  
    &:hover { color: $hover; }  
    &:visited { color: $visited; }  
}
```

当混合器被@include 时，你可以把它当作一个 css 函数来传参。如果你像下边这

样写：

```
a {  
    @include link-colors(blue, red, green);  
}  
  
//Sass 最终生成的是:  
  
a { color: blue; }  
  
a:hover { color: red; }  
  
a:visited { color: green; }
```

当你@include 混合器时,有时候可能会很难区分每个参数是什么意思,参数之间是一个什么样的顺序。为了解决这个问题,sass 允许通过语法\$name: value 的形式指定每个参数的值。这种形式的传参,参数顺序就不必再在乎了,只需要保证没有漏掉参数即可：

```
a {  
    @include link-colors(  
        $normal: blue,  
        $visited: green,  
        $hover: red  
    );  
}
```

尽管给混合器加参数来实现定制很好,但是有时有些参数我们没有定制的需要,这时候也需要赋值一个变量就变成很痛苦的事情了。所以 sass 允许混合器声明时给



参数赋默认值。

## 5-4. 默认参数值

为了在@include 混合器时不必传入所有的参数，我们可以给参数指定一个默认值。参数默认值使用\$name: default-value 的声明形式，默认值可以是任何有效的css 属性值，甚至是其他参数的引用，如下代码：

```
@mixin link-colors(  
    $normal,  
    $hover: $normal,  
    $visited: $normal  
)  
{  
    color: $normal;  
    &:hover { color: $hover; }  
    &:visited { color: $visited; }  
}
```

如果像下边这样调用：@include link-colors(red) \$hover 和\$visited 也会被自动赋值为 red。

混合器只是 sass 样式重用特性中的一个。我们已经了解到混合器主要用于样式展示层的重用，如果你想重用语义化的类呢？这就涉及 sass 的另一个重要的重用特性：选择器继承。

## 6. 使用选择器继承来精简 CSS

### 6-0. 什么是 Sass 继承

使用 sass 的时候，最后一个减少重复的主要特性就是选择器继承。基于 Nicole Sullivan 面向对象的 css 的理念，选择器继承是说一个选择器可以继承为另一个选择器定义的所有样式。这个通过@extend 语法实现，如下代码：

```
//通过选择器继承继承样式

.error {

    border: 1px red;

    background-color: #fdd;

}

.seriousError {

    @extend .error;

    border-width: 3px;

}
```

在上边的代码中，.seriousError 将会继承样式表中任何位置处为.error 定义的所有样式。以 class="seriousError" 修饰的 html 元素最终的展示效果就好像是 class="seriousError error"。相关元素不仅会拥有一个 3px 宽的边框，而且这个边框将变成红色的，这个元素同时还会有一个浅红色的背景，因为这些都是在.error 里边定义的样式。

.seriousError 不仅会继承.error 自身的所有样式，任何跟.error 有关的组合选择

器样式也会被 `.seriousError` 以组合选择器的形式继承，如下代码：

```
//.seriousError 从.error 继承样式

.error a{ //应用到.seriousError a

    color: red;

    font-weight: 100;

}

h1.error { //应用到 h1.seriousError

    font-size: 1.2rem;

}
```

如上所示，在 `class="seriousError"` 的 html 元素内的超链接也会变成红色和粗体。

本节将介绍与混合器相比，哪种情况下更适合用继承。接下来在探索继承的工作细节之前，我们先了解一下继承的高级用法。最后，我们将看看使用继承可能会有哪些坑，学习如何避免这些坑。

接下来，我们再通过一些实例一步步由浅入深的了解 Sass 之继承。

## 6-1. 如何使用 Sass 继承

每一个类名都有可能有另一个类名的所有样式和它自己的特定样式的。当一个 div 的身上有两个类名，一个是“one”，另一个是“two”的时候。如下：

*HTML 代码*

```
<div class="one two">
```

Sass 继承

&lt;/div&gt;

### CSS 代码

```
.one {width:100px;height:100px;}  
.two {background:red;border:5px solid #000;}
```

这就意味着，我们要配备一个很好的记忆力才能应急那些棘手的 BUG 问题。很多时候，我们需要一个个的找到类名“one”的样式才逐一改过。这样在无形中加大了。而在 SASS 中，有那么一个功能，继承，可以方便轻松的让这棘手的问题变得很简单。@extend 指令，可以继承你想要的类名。如下：

### SCSS 代码

```
.one {  
    width:100px;height:100px;  
}  
  
.two {  
    @extend .one;  
    background:red;border:5px solid #000;  
}
```

### 编译后的 CSS 代码

```
.one, .two {  
    width: 100px;  
    height: 100px;
```

```
}  
  
.two {  
    background: red;  
    border: 5px solid #000;  
}
```

从上面的例子可以看出，类名“one”，可以被所有的类名继承。而继承类名“.one”的类名还可以有专属于自己的类名，专属的样式和风格，并不影响类名“.one”本身的样式。

#### 6.1.1. 它是如何工作

SASS 中继承(Extend)将想要继承的选择器(如“.two”)和其引用的选择器(如“.one”)组成群组选择器中间用逗号隔开，组成群组选择器。如下：红色背景和边框是类名“.two”单加的样式，而宽度高度都是继承类名(“.one”)的。如下

*SCSS 代码*

```
.one {  
    width:100px;height:100px;  
}  
  
.two {  
    /*继承的样式*/  
    @extend .one;  
    /*独立的样式*/  
    background:red;
```

```
border:5px solid #000;  
  
}
```

### 编译后的 CSS 代码

```
.one, .two {  
    /*继承的样式*/  
    width: 100px;  
    height: 100px;  
}  
  
.two {  
    /*独立的样式*/  
    background: red;  
    border: 5px solid #000;  
}
```

就在合并选择器的时候，SASS 中继承(Extend)是相当聪明的，会自己避免一些不必要的错误，或者说不为人知的错处，也不会产生各种多余的东西如“##two”。

#### 6.1.2. 继承复杂的选择器

要继承的不仅仅是一个类名，可以是一个 id 也可以是一个元素，也可以是某个状态，任何选择器都能继承。如下：

#### SCSS 代码

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

```
}  
  
a:hover {  
    text-decoration: underline;  
}
```

编译后的 CSS 代码

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

就像上面编译出来的一样，在 hover 状态下的样式也能继承。所以不仅是 a 的 hover 状态，几乎任何选择器都能继承。

### 6.1.3. 继承多个选择器

在编写的过程中，往往会遇到一个选择器要继承多个选择器的样式，那么这应该怎么办呢？看看下面的小实例吧。

SCSS 代码

```
.one {  
    width:100px;height:100px;  
}  
  
.two {  
    /*继承的样式*/  
    @extend .one;  
    @extend .three;
```

```
/*独立的样式*/

background:red;

border:5px solid #000;

}

.three {

padding:10px;

}
```

### 编译后的 CSS 代码

```
.one, .two {

width: 100px;

height: 100px;

}

.two {

background: red;

border: 5px solid #000;

}

.three, .two {

padding: 10px;

}
```

上面的例子告诉我，继承多个选择器的样式是得写多个“@extend”，但事实上有没有简便写法呢？看下面的小例子。



## SCSS 代码

```
.one {  
    width:100px;height:100px;  
}  
  
.two {  
    /*继承的样式*/  
    @extend .one, .three;  
    /*独立的样式*/  
    background:red;  
    border:5px solid #000;  
}  
  
.three {  
    padding:10px;  
}
```

## 编译后的 CSS 代码

```
.one, .two {  
    width: 100px;  
    height: 100px;  
}  
  
.two {  
    /*独立的样式*/  
    background: red;
```

```
border: 5px solid #000;

}

.three, .two {

padding: 10px;

}
```

从上面的小例子，不难看出，继承多个选择器的样式是有简便方法的，那就是继承的多个选择器自己用逗号“,”隔开即可。

#### 6.1.4. 链型继承

在咱们编写的过程中，不仅仅的单方面的继承，很多时候都是类名“.three”继承类名“.two”，而类名“.two”又继承了类名“.one”。那么这样的情况在 SASS 中应该怎么写呢?写法如下

*SCSS 代码*

```
.one {

width:100px;height:100px;

}

.two {

/*继承的样式*/

@extend .one;

/*独立的样式*/

background:red;

border:5px solid #000;
```

```
}  
  
.three {  
  
    /*继承的样式*/  
  
    @extend .two;  
  
    /*独立的样式*/  
  
    padding:10px;  
  
}
```

### 编译后的 CSS 代码

```
.one, .two, .three {  
  
    /*继承的样式*/  
  
    width: 100px;  
  
    height: 100px;  
  
}  
  
.two, .three {  
  
    /*独立的样式*/  
  
    background: red;  
  
    border: 5px solid #000;  
  
}  
  
.three {  
  
    /*独立的样式*/  
  
    padding: 10px;  
  
}
```

从编译后的 CSS 中不难看出，类名“.one”变成了群组选择，添加了类名“.two”和类名“.three”。而类名“.two”变成了群组选择，添加了类名“.three”。那么为什么叫做链型继承呢？是因为一层一层的继承，很像一条链子故得名。

### 6.1.5. 继承的局限性

虽然能够继承的选择器数量很多，但是也有很多选择器并不被支持继承的，如包含选择器(.one .two)或者相邻兄弟选择器(.one+.two)目前还是不支持继承。但若继承的元素是“a”，恰巧这个元素“a”又有 hover 状态的样式，那么 hover 状态的样式也会被继承。如下：

*SCSS 代码*

```
.myLink {  
  @extend a;  
}  
  
a {  
  color: blue;  
  &:hover {  
    text-decoration: underline;  
  }  
}
```

*编译后的 CSS 代码*

```
a, .myLink {
```

```
color: blue;

}

a:hover, .myLink:hover {

    text-decoration: underline;

}
```

继承有强大的地方，也有局限的地方。对于 a 的继承可以继承其 hover 状态，但是对于某些选择器就不适用了。

#### 6.1.6. 继承交叉合并选择

继承交叉合并选择，从字面上其实很难理解说的事件什么事儿。既然有点难理解，那么就先看小例子吧，用例子来解释感觉会比较清楚。例子如下

*SCSS 代码*

```
.meng a {

    font-weight:bold;

}

.long .link {

    @extend a;

}
```

*编译后的 CSS 代码*

```
.meng a, .meng .long .link, .long .meng .link {

    font-weight: bold;

}
```

```
}
```

从上面 的例子不难看出，类名“.meng”中的“a”选择器被类名“.long”中的类名“.link”继承了，但是由于没有在同一个父级下，会产生交叉合并的编译结果。这种其实是可以解决的，就是把父级改成相同的即可。若父级不能改的话，那么就乖乖的在写一遍，或者将“.meng a”直接换成类名“.meng”，继承这个类名也可以。

### 6.1.7. 继承带%的东西

有时候你想继承一个类名，但是并不想再在 HTML 中使用，就单单想写一个能够继承的类名。尤其是不想让它出现在 css 样式中。我们可以用占位符来写一些继承的样式(如“%one”)，然后再通过@extend 继承，这样就可以防止被渲染到 CSS 的规则集中。如下：

*SCSS 代码*

```
#meng a%long {  
    color: blue;  
    font-weight: bold;  
    font-size: 2em;  
}  
  
.notice {  
    @extend %long;  
}
```

*编译后的 CSS 代码*

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

### 6.1.8. 继承在指令中的作用域

继承在指令中是有作用域问题的,继承是无法使在指令如(@media)之外的选择器继承的,要是继承就只能写在指令中。实例如下

*SCSS 代码*

```
.one {  
  
    height:300px;  
  
}  
  
@media print {  
  
    .two {  
  
        @extend .one;  
  
        width:300px;  
  
    }  
  
}
```

*编译后的 CSS 代码*

```
.one {  
  
    height: 300px;
```

```
}  
  
@media print {  
  
  .two {  
  
    width: 300px;  
  
  }  
  
}
```

右上可见，类名“.two”并没有继承类名“.one”的样式，那么需要让类名“.two”成功继承类名“.one”的样式，就应该把类名“.one”也放在@media 中，实例如下

### SCSS 代码

```
@media print {  
  
  .one {  
  
    height:300px;  
  
  }  
  
  .two {  
  
    @extend .one;  
  
    width:300px;  
  
  }  
  
}
```

### 编译后的 CSS 代码

```
@media print {  
  
  .one, .two {
```



```
    height: 300px;

}

.two {

    width: 300px;

}

}
```

## 6-2. 何时使用继承

5-1 节介绍了混合器主要用于展示性样式的重用，而类名用于语义化样式的重用。因为继承是基于类的（有时是基于其他类型的选择器），所以继承应该是建立在语义化的关系上。当一个元素拥有的类（比如说 `.seriousError`）表明它属于另一个类（比如说 `.error`），这时使用继承再合适不过了。

这有点抽象，所以我们从几个方面来阐释一下。想象一下你正在编写一个页面，给 html 元素添加类名，你发现你的某个类（比如说 `.seriousError`）另一个类（比如说 `.error`）的细化。你会怎么做？

- 你可以为这两个类分别写相同的样式，但是如果有大量的重复怎么办？使用 sass 时，我们提倡的就是不要做重复的工作。
- 你可以使用一个选择器组（比如说 `.error``.seriousError`）给这两个选择器写相同的样式。如果 `.error` 的所有样式都在同一个地方，这种做法很好，但是如果是分散在样式表的不同地方呢？再这样做就困难多了。
- 你可以使用一个混合器为这两个类提供相同的样式，但当 `.error` 的样式修饰遍布样式表中各处时，这种做法面临着跟使用选择器组一样的问题。这两个类也

不是恰好有相同的 样式。你应该更清晰地表达这种关系。

- 综上所述你应该使用@extend。让.seriousError 从.error 继承样式，使两者之间的关系非常清晰。更重要的是无论你在样式表的哪里使用.error``.seriousError 都会继承其中的样式。

现在你已经更好地掌握了何时使用继承，以及继承有哪些突出的优点，接下来我们看看一些高级用法。

### 6-3. 使用继承的最佳实践

常使用继承会让你的 css 美观、整洁。因为继承只会在生成 css 时复制选择器，而不会复制大段的 css 属性。但是如果你不小心，可能会让生成的 css 中包含大量的选择器复制。

避免这种情况出现的最好方法就是不要在 css 规则中使用后代选择器（比如.foo .bar）去继承 css 规则。如果你这么做，同时被继承的 css 规则有通过后代选择器修饰的样式，生成 css 中的选择器的数量很快就会失控：

```
.foo .bar { @extend .baz; }  
  
.bip .baz { a: b; }
```

值得一提的是，只要你想，你完全可以放心地继承有后代选择器修饰规则的选择器，不管后代选择器多长，但有一个前提就是，不要用后代选择器去继承

## 7. 总结

本部分介绍了 sass 最基本部分,你可以轻松地使用 sass 编写清晰、无冗余、语义

化的 css。对于 sass 提供的工具你已经有了一个比较深入的了解，同时也掌握了何时使用这些工具的指导原则。

变量是 sass 提供的最基本的工具。通过变量可以让独立的 css 值变得可重用，无论是在一条单独的规则范围内还是在整个样式表中。变量、混合器的命名甚至 sass 的文件名，可以互换通用\_和-。同样基础的是 sass 的嵌套机制。嵌套允许 css 规则内嵌套 css 规则，减少重复编写常用的选择器，同时让样式表的结构一眼望去更加清晰。sass 同时提供了特殊的父选择器标识符&，通过它可以构造出更高效的嵌套。

你也已经学到了 sass 的另一个重要特性，样式导入。通过样式导入可以把分散在多个 sass 文件中的内容合并生成到一个 css 文件，避免了项目中有大量的 css 文件通过原生的 css @import 带来的性能问题。通过嵌套导入和默认变量值，导入可以构建更强有力的、可定制的风格。混合器允许用户编写语义化风格的同时避免视觉层面上风格的重复。你不仅学到了如何使用混合器减少重复，同时学习到了如何使用混合器让你的 css 变得更加可维护和语义化。最后，我们学习了与混合器相辅相成的选择器继承。继承允许你声明类之间语义化的关系，通过这些关系可以保持你的 css 的整洁和可维护性。

## 三、Sass 函数

### 3.1、Sass 内置函数

在 Sass 中除了可以定义变量，具有@extend，%placeholders 和 Mixins 等特性之外，还自备了一系列的函数功能。主要包含以下几类：**字符串函数**、**数字函数**、**列表函数**、**Introspection 函数**、**三元函数** 以及 **颜色函数** 等，当然大家还可以根

据需求自定义函数。

### 3.1.1、字符串函数

字符串函数顾名思义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。

#### ➤ `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);  
}  
  
//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; }
```

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。

➤ quote()函数

quote()函数刚好与 unquote()函数功能相反，主要用来给字符串添加引号。如果字符串，自身带有引号会统一换成双引号"。如：

```
//SCSS

.test1 {

    content:  quote('Hello Sass!');

}

.test2 {

    content: quote("Hello Sass!");

}

.test3 {

    content: quote(ImWebDesigner);

}

.test4 {

    content: quote(' ');

}

//CSS

.test1 {

    content: "Hello Sass!";

}

.test2 {
```

```
content: "Hello Sass!";  
  
}  
  
.test3 {  
    content: "ImWebDesigner";  
}  
  
.test4 {  
    content: " ";  
}
```

使用 `quote()` 函数增加只能给字符串增加双引号，而且字符串中间有单引号或者空格时，需要用单引号或双引号括起，否则编译的时候将会报错。

```
.test1 {  
    content: quote>Hello Sass);  
}
```

这样使用，编译器马上会报错：

```
error style.scss (Line 13: $string: ("Hello" "Sass") is not a string  
for `quote`)
```

解决方案就是去掉空格，或者加上引号：

```
.test1 {  
    content: quote>HelloSass);  
}  
  
.test1 {
```

```
content: quote("Hello Sass");  
}
```

同时 quote() 碰到特殊符号，比如说 !、?、> 等，除中折号 - 和下划线 \_ 都需要使用双引号括起，否则编译器在进行编译的时候同样会报错：

```
error style.scss (Line 13: Invalid CSS after "...quote(HelloSass":  
expected ")", was "!);")  
  
error style.scss (Line 16: Invalid CSS after "...t: quote(Hello":  
expected ")", was "?);")
```

### 3.1.2、数字函数

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。

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。

#### ➤ percentage() 函数

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%

>>
```

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`
```

### ➤ round()函数

round()函数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
```

在 `round()` 函数中可以携带单位的任何数值。

#### ➤ `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
```

#### ➤ 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
```

### ➤ 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
```

## ➤ min()函数

min()函数功能主要是在多个数之中找到最小的一个，这个函数可以设置任意多个参数：

```
>> min(1,2,1%,3,300%)  
  
1%  
  
>> min(1px,2,3px)  
  
1px  
  
>> min(1em,2em,6em)  
  
1em
```

不过在 min()函数中同时出现两种不同类型的单位，将会报错误信息：

```
>> min(1px,1em)  
  
SyntaxError: Incompatible units: 'em' and 'px'.
```

## ➤ max()函数

max()函数和 min()函数一样，不同的是，max()函数用来获取一系列数中的最大那个值：

```
>> max(1,5)  
  
5  
  
>> max(1px,5px)  
  
5px
```

同样的，如果在 max()函数中有不同单位，将会报错：

```
>> max(1,3px,5%,6)
```

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

### ➤ List 函数

### 3.1.3、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])`：将两个列表连接在一起，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，否则函数将会报错：

```
>> 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

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 是指列表中的第二个标签值，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()函数中还有一个很有特别的参数\$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；如果第二列表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，在此建议大家使用 `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
```

### 3.1.4、三元函数

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($number)`：判断一个值是否带有带位
- `comparable($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，依次来看每个函数的功能。

#### ➤ `type-of()`函数

`type-of()`函数主要用来判断一个值是属于什么类型：

```
>> 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，不过只允许乘除运算：

```
>> 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
```

➤ comparable()函数

comparable()函数主要是用来判断两个数是否可以相加，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

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
```

### 3.1.5、颜色函数

在 Sass 的官方文档中，列出了 Sass 的颜色函数清单，从大的方面主要分为 RGB、HSL 和 Opacity 三大类函数，当然其还包括一些其他颜色函数，比如说 adjust-color、change-color 等等。

#### ➤ RGB 颜色函数

rgb 颜色只是颜色中的一种表达方式,其中 R 是“red”表示红色,而 G 是“green”绿色,B 是“blue”蓝色。在 Sass 中为 RGB 颜色提供六种函数:

- ◆ `rgb($red,$green,$blue)`: 根据红、绿、蓝三个值创建一个颜色;
- ◆ `rgba($red,$green,$blue,$alpha)`: 根据红、绿、蓝和透明度值创建一个颜色;
- ◆ `red($color)`: 从一个颜色中获取其中红色值;
- ◆ `green($color)`: 从一个颜色中获取其中绿色值;
- ◆ `blue($color)`: 从一个颜色中获取其中蓝色值;
- ◆ `mix($color-1,$color-2,[$weight])`: 把两种颜色混合在一起。

仅从概念上,或许大家没有太多的概念,我们通过下面的命令来做一个简单的测试:

```
$ sass -i //在终端运行这个命令,开启 Sass 的函数计算
```

接下来,分别在终端使用 RGB 函数来进行计算,看其最终结果:

```
$ sass -i

>> rgb(200,40,88) //根据 r:200,g:40,b:88 计算出一个十六进制颜色值
#c82858

>> rgba(#c82858,.65) //根据#c82858 的 65%透明度计算出一个 rgba 颜色值
rgba(200, 40, 88, 0.65)

>> red(#c82858) //从#c82858 颜色值中得到红色值 200
200

>> green(#c82858) //从#c82858 颜色值中得到绿色值 40
40
```

```
>> blue(#c82858) //从#c82858 颜色值中得到蓝色值 88

88

>> mix(#c82858,rgba(200,40,80,.65),.3) //把#c82858 和 rgba(200,40,80,.65)两颜色按比例混合得到一个新颜色

rgba(200, 40, 80, 0.65105)
```

### ➤ RGBA()函数

rgba()函数主要用来将一个颜色根据透明度转换成 rgba 颜色。其语法有两种格式：

```
rgba($red,$green,$blue,$alpha)//将一个 rgba 颜色转译出来,和未转译的值一样
```

```
rgba($color,$alpha) //将一个 Hex 颜色转换成 rgba 颜色
```

其中 rgba(\$color,\$alpha)函数作用更大，主要运用在这样的情形之中。假设在实际中知道的颜色值是#f36 或者 red，但在使用中，需要给他们配上一个透明度，这个时候在原来的 CSS 中，首先需要通过制图工具，得到#f36 或 red 颜色的 R、G、B 值，而不能直接使用：

```
//CSS

color: rgba($f36,.5);//这是无效的写法
```

但在 Sass 中，RGBA()函数就能解决这个问题。我们先来看一个简单的例子，假设在变量中定义了一个基本的变量：

```
$color: #f36;

$bgColor: orange;

$borderColor:green;
```

同时给他们加上.65 的透明度：

```
//SCSS

.rgba {

    color: rgba(#f36,.5);

    background: rgba(orange,.5);

    border-color: rgba(green,.5);

}
```

在这个实例中，我们使用了 Sass 的 rgba 函数，在括号是函数的参数，第一个参数是需要转换的颜色，他可以是任何颜色的表达方式，也可以是一个颜色变量；第二个参数是颜色的透明度，其值是 0~1 之间。上面的代码转译出来：

```
.rgba {

    color: rgba(255, 51, 102, 0.5);

    background: rgba(255, 165, 0, 0.5);

    border-color: rgba(0, 128, 0, 0.5);

}
```

在来看一个调用前面定义的变量：

```
//SCSS

.rgba {

    color: rgba($color,.5);

    background: rgba($bgColor,.5);

    border-color: rgba($borderColor,.5);

}
```

```

}

//CSS

.rgba {

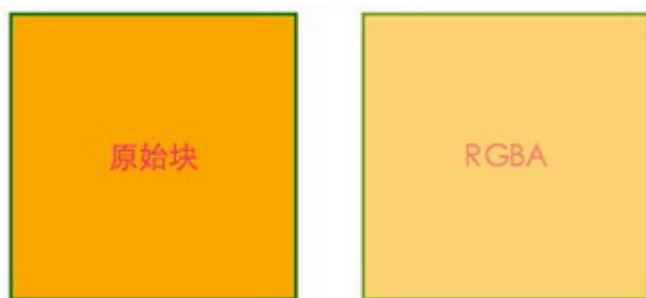
    color: rgba(255, 51, 102, 0.5);

    background: rgba(255, 165, 0, 0.5);

    border-color: rgba(0, 128, 0, 0.5);

}
    
```

我想您应该会看到一个变化，通过 rgba 函数，指定一个透明值，将原色转换成另外一个颜色：



#### ➤ Mix()函数

Mix 函数是将两种颜色根据一定的比例混合在一起，生成另一种颜色。其使用法如下：

```

mix($color-1,$color-2,$weight);
    
```

\$color-1 和 \$color-2 指的是你需要合并的颜色，颜色可以是任何表达式，也可以是颜色变量。

\$weight 为合并的比例（选择权重），默认值为 50%，其取值范围是 0~1 之间。

它是每个 RGB 的百分比来衡量，当然透明度也会有一定的权重。默认的比例是

50%，这意味着两个颜色各占一半，如果指定的比例是 25%，这意味着第一个颜色所占比例为 25%，第二个颜色所占比例为 75%。

```
mix(#f00, #00f) => #7f007f
```

```
mix(#f00, #00f, 25%) => #3f00bf
```

```
mix(rgba(255, 0, 0, 0.5), #00f) => rgba(63, 0, 191, 0.75)
```

在前面的基础上，做一个修改：

```
//SCSS

$color1: #a63;
$color2: #fff;
$bgColor1: #f36;
$bgColor2: #e36;
$borderColor1: #c36;
$borderColor2: #b36;

.mix {
    background: mix($bgColor1, $bgColor2, .75);
    color: mix($color1, $color2, .25);
    border-color: mix($borderColor1, $bgColor2, .05);
}
```

编译的 css 代码：

```
//CSS

.mix {
```



```
background: #ee3366;

color: #fefefe;

border-color: #ed33

}
```

这就是 Mix 函数的工作原理，在函数中指定三个函数，前两个函数是你想混合的颜色（记住，你可以通过颜色变量、十六进制、RGBA、RGB、HSL 或者 HSLA 颜色值）。第三个参数是第一种颜色的比例值。



更多颜色函数参见 Sass 官网：<http://sass-lang.com/>

### 3.2、Sass 自定义函数

自定义函数是用户根据自己一些特殊的需求编写的 Sass 函数。在很多时候，Sass 自带的函数并不能满足功能上的需求，所以很多时候需要自定义一些函数。

例如，我们需要去掉一个值的单位，在这种情形之下，Sass 自带的函数是无法帮助我们完成，这个时候我们就需要定义函数：

```
//去掉一个值的单位，如 12px => 12

// eg. strip-units(12px) => 12

@function stripUnits($number){
```

```
@return $number / ($number * 0 + 1);  
  
}  
  
//eg. double(5px) => 10px  
  
@function double($n) {  
  
    @return $n * 2;  
  
}
```

## 四、Sass 高级用法

### 4.1 条件语句

@if 可以用来判断：

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

配套的还有@else 命令：

```
@if lightness($color) > 30% {  
  
    background-color: #000;  
  
} @else {  
  
    background-color: #fff;  
  
}
```

## 4.2 循环语句

SASS 支持 for 循环：

```
@for $i from 1 to 10 {  
    .border-#{ $i } {  
        border: #{ $i }px solid blue;  
    }  
}
```

也支持 while 循环：

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

each 命令，作用与 for 类似：

```
@each $member in a, b, c, d {  
    .#{ $member } {  
        background-image: url("/image/#{ $member }.jpg");  
    }  
}
```

---

## 五、利用 Koala 编译 Sass

利用 Koala 编译 Sass，类似于我之前讲的利用 Koala 编译 Less，此处不再累述，需要了解的同学请自行参考以下文章：

<http://www.alixixi.com/web/a/2013082790460.shtml>

www.ibEIFeng.com