

SASS 教程

1-1 SASS 简介

CSS 不是一个编程语言，可以用它来开发网页样式，但是没有办法用它进行编程。SASS 的出现，让 CSS 实现了通过代码编程来实现的方式。

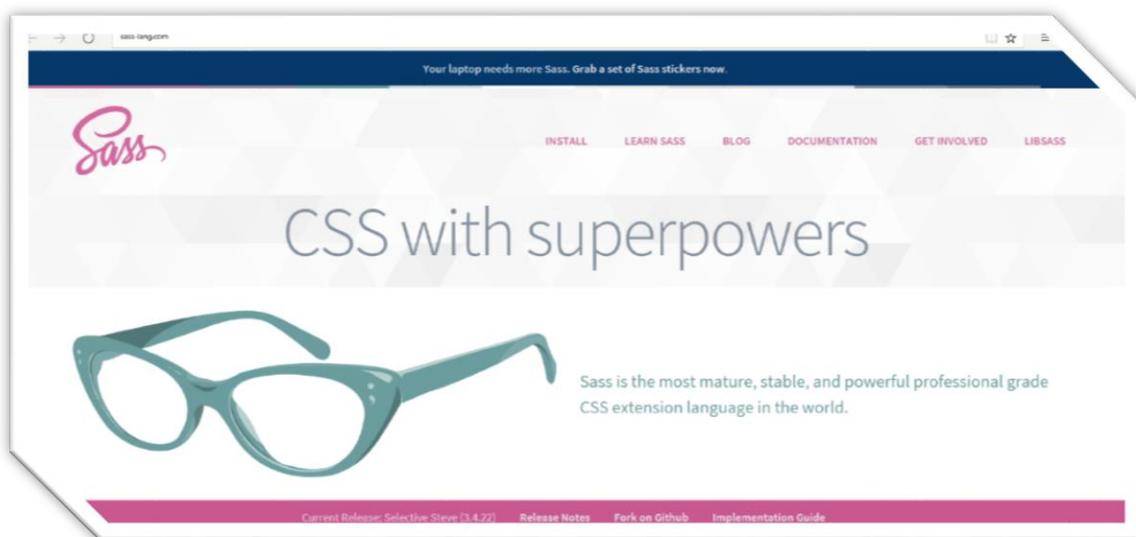
SO...SASS 是一种 CSS 开发工具，提供了许多便利的写法，让 CSS 的处理实现了可编程处理。

SASS 扩展了 CSS3，增加了规则、变量、混入、选择器、继承等等特性，可以生成风格良好的 CSS 样式表文件，易于组织和维护。

1-2 SASS 安装

SASS 是 Ruby 语言开发的一个用于动态编程 CSS 文件的框架，但是学习 SASS 跟 Ruby 没有任何关系，唯一的联系就是 SASS 的运行以来 Ruby 环境。

SASS 官网：<http://sass-lang.com/>



1-2.1 MAC 安装

mac 上一般情况已经有 ruby 环境的支持，所以只需要打开终端，输入一下命令进行安装

```
gem install sass
```

备注：安装完成后，使用如下命令进行测试【查看安装版本命令】

```
sass -v
```

1-2.2 windows 安装

- 安装 ruby
 - sass 的依赖环境，必须安装。
- 通过命令提示符黑窗口执行命令进行安装

```
gem install sass
```

1-3 QUICK START

1-3.1 入门程序

- 文件结构

```
|-- WORKSass/  
    |-- css/  
    |-- demo01.scss
```

- 在指定的文件夹中创建 demo01.scss 文本文件

```
@charset "utf-8"; // 设置支持中文注释  
body{  
    background-color:#fff;  
    font-size:16px;  
}
```

- 命令行中，执行如下命令进行编辑
 - `sass [scss_name]:<targetCss_name>`
 - `sass scss 文件:编译好的目标 css 全路径名称`

```

C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation。保留所有权利。

C:\Users\FLEX>cd D:\RESOURCE\WORKSass
C:\Users\FLEX>d:
D:\RESOURCE\WORKSass>sass demo01.scss:css/demo01.css
D:\RESOURCE\WORKSass>
    
```

- 编译结果：在指定的文件夹 `css` 中生成了指定的 `css` 样式表文件：`demo01.css`

1-3.2 自动编译命令

- `watch` 命令

语法

```
>> sass --watch sass:css
```

```
>> sass --watch 要监听的目录:编译后的 css 文件的位置
```

- 使用 `watch` 命令

开始监听【`--watch .:css` 命令中，`.`表示当前目录】

```

D:\RESOURCE\WORKSass>sass --watch .:css
>>> Sass is watching for changes. Press Ctrl-C to stop.
    
```

修改 `sass` 文件

```

demo01.scss
1  *{
2      margin:0;
3      padding:0;
4  }
5  body{
6      background-color:#fff;
7      font-size:16px;
8      color:#333;
9  }
    
```

命令行监听提示

```
D:\RESOURCE\WORKSass>sass --watch .:css
>>> Sass is watching for changes. Press Ctrl-C to stop.
>>> Change detected to: demo01.scss
    write css/demo01.css
    write css/demo01.css.map
>>> Change detected to: demo01.scss
    write css/demo01.css
    write css/demo01.css.map
```

目标 css 文件样式

```
demo01.scss x demo01.css x
1 * {
2   margin: 0;
3   padding: 0; }
4
5 body {
6   background-color: #fff;
7   font-size: 16px;
8   color: #333; }
9
10 /*# sourceMappingURL=demo01.css.map */
```

1-3.3 编译输出格式

默认情况下，SASS 提供了四种格式的 CSS 输入，默认情况输出是嵌套格式

格式	描述
nested	嵌套
compact	紧凑——
expanded	扩展——完全格式化标准
compressed	压缩——一行

不同格式的展示方式

- 案例 1：嵌套格式——不常用

demo02.scss 【以下是 sass 中嵌套格式的写法】

```
11 ul {
12     font-size: 16px;
13     li {
14         list-style: none;
15     }
16 }
```

demo02.css 【输出的 css 样式中，会保存这样的嵌套的缩进格式】

```
10 ul {
11     font-size: 16px; }
12 ul li {
13     list-style: none; }
14
```

● 案例 2：紧凑格式——常用

- 修改输出样式

■ >> sass --watch .:css --style compact

命令行修改输出格式

```
D:\RESOURCE\WORK\Sass>sass --watch .:css --style compact
>>> Sass is watching for changes. Press Ctrl-C to stop.
```

修改格式 demo02.scss

```
5 body{
6     background-color:#fff;
7     font-size:16px;
8     color:#333;
9 }
10
11 ul{
12     font-size:16px;
13     li{
14         list-style:none;
15     }
16 }
```

输出格式 demo02.css

```
3 body { background-color: #fff; font-size: 16px; color: #333; }
4
5 ul { font-size: 16px; }
6 ul li { list-style: none; }
```

● 案例 3：扩展格式——常用——规范的 CSS 代码格式

修改输出样式格式

>> sass --watch .:css --style expanded

修改 demo02.scss 文件

.. .代码同上. . .

输出 demo02.css 文件格式

```
6 body {
7     background-color: #fff;
8     font-size: 16px;
9     color: #333;
10 }
11
12 ul {
13     font-size: 16px;
14 }
15 ul li {
16     list-style: none;
17 }
```

- 案例 4：压缩格式——常用——用于在线项目的小文件

修改输出样式格式
>> sass --watch .:css --style compressed
修改 demo02.scss 文件
... 代码同上. ...
输出 demo02.css 文件格式
1 body{background-color:#fff;font-size:16px;color:#333}ul{font-

1-3.4 sass 扩展名

- .sass 【sass3.0-版本】
- .scss 【sass3.0+版本，常用】

这两种在书写代码格式上，有一定的区别

根据项目开发的规范，scss 的写法和项目规范更加契合，同时代码的可读性也提高了很多，所以正常情况下，项目组都会采用 .scss 格式的语法进行 sass 程序开发。

.sass	.scss
<pre> 1 /* SASS技术开发 2 by laomu 3 块注释 4 5 6 // 行注释 7 行注释 8 9 @import base 10 11 =alert { 12 color:#ccc; 13 background-color:#333; 14 } 15 16 .alert-warning { 17 +alert; 18 } 19 20 ul 21 font-size:16px; 22 li 23 list-style:none; </pre>	<pre> 1 /* 2 SASS技术开发 3 by laomu 4 块注释 5 */ 6 7 // 行注释 8 // 行注释 9 10 @import "base" 11 12 @mixin alert { 13 color:#ccc; 14 background-color:#333; 15 } 16 17 .alert-warning { 18 @include alert; 19 } 20 21 ul{ 22 font-size:16px; 23 li{ 24 list-style:none; 25 } 26 } </pre>

备注：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。而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。

1-4 SASS 编程基础

有了前面的 QUICK START 部门的简单了解，基本能描述清楚 SASS 的用途了，具体的控制样式 CSS 的定义和其他的程序开发操作，就需要对 SASS 中提供的各种基础性的知识有一个简单的认知和应用过程。

1-4.1.1 变量——Variables

SASS 中的变量，必须是 \$ 符号开头，后面紧跟变量名，变量名称和变量值之间要使用冒号：进行分隔（参考 CSS 属性和值的设定语法），如果值后面加上 [!default] 就表示默认值。

引用变量的值，直接使用变量名称，即可引用定义的变量的值。

- 普通变量，定义之后可以在全局范围内使用

demo.scss
<pre>//sass style //----- \$fontSize: 12px; body{ font-size:\$fontSize; }</pre>
demo.css
<pre>body{ font-size:12px; }</pre>

- 默认变量，sass 的默认变量需要在值的后面加上 !default 进行标识

demo.scss
<pre>//sass style //----- \$baseLineHeight: 1.5 !default; body{ line-height: \$baseLineHeight; }</pre>
demo.css
<pre>body{ line-height:1.5; }</pre>
默认变量的覆盖：在默认变量前后，重新声明变量并赋值即可
默认变量是基于组件化开发的过程中，进行优化处理的

- 特殊变量：一般情况下，我们定义的变量都是属性值，可以直接使用，但是如果变量作为属性或者其他的特殊情况下，必须使用 #{ \$variable } 的形式进行调用。
 - #{ \$variable } 就是取值的一种特殊形式，符合特殊用法。

demo.scss

```

1 $borderDirection:top !default;
2 $baseFontSize:16px;
3 $baseLineheight:1.5;
4 #top_nav{
5     border-#{$borderDirection}:1px solid #ccc;
6 }
7 body{
8     font:#{$baseFontSize}/#{$baseLineheight};
9 }

```

demo.css

```

1 #top_nav {
2     border-top: 1px solid #ccc;
3 }
4
5 body {
6     font: 16px/1.5;
7 }

```

- 全局变量——在变量的后面加上[!global]即可声明全局变量。sass 规划是 3.4 以后的版本中就会增加这个功能。

1-4.2 嵌套——Nesting

SASS 中的嵌套主要说的是选择器嵌套和属性嵌套两种方式，正常项目中通常使用的都是选择器嵌套方案

- 选择器嵌套

demo.scss

```

1 #top_nav {
2     font-size:16px;
3
4     li {
5         width:100px;
6     }
7     a {
8         display:block;
9
10        &:hover{
11            color:#ddd;
12        }
13    }
14 }

```

demo.css

```

1 #top_nav {
2     font-size: 16px;
3 }
4 #top_nav li {
5     width: 100px;
6 }
7 #top_nav a {
8     display: block;
9 }
10 #top_nav a:hover {
11     color: #ddd;
12 }
13
14 /*# sourceMappingURL=demo01.css.map */

```

1-4.3 嵌套——父属性调用

在嵌套的过程中，如果需要用到父元素，在 SASS 中通过 **&符号** 引用父属性

1-4.4 嵌套属性

- 嵌套属性——不常用

所谓属性嵌套，是指某些属性拥有同样的单词开头，如 `:border-left`, `border-color` 都是以 `border` 开头的，所以就出现了属性嵌套语法

demo.scss

```
12 #top_nav_li {
13     border:{
14         style: solid;
15         left:{
16             width:4px;
17             color:#888;
18         }
19         right:{
20             width:2px;
21             color:#ccc;
22         }
23     }
24 }
```

demo.css

```
11 #top_nav_li {
12     border-style: solid;
13     border-left-width: 4px;
14     border-left-color: #888;
15     border-right-width: 2px;
16     border-right-color: #ccc;
17 }
```

1-4.5 混合——Mixin

sass 中可以通过 `@mixin` 声明混合，可以传递参数，参数名称以 `$` 开始，多个参数之间使用逗号分隔，`@mixin` 的混合代码块由 `@include` 来调用

- 无参数混合——不建议使用，如果是这样的代码块，直接使用后面提到的 `@extend` 来处理

demo.scss

```
1 @mixin center-block {
2     margin-left:auto;
3     margin-right:auto;
4 }
5
6 div.container{
7     @include center-block;
8 }
```

demo.css

```
1 div.container {
2   margin-left: auto;
3   margin-right: auto;
4 }
```

- 有参数混合

demo.scss

```
1 @mixin opacity($opacity:50) {
2   opacity:$opacity;
3 }
4
5 #container {
6   @include opacity;
7 }
8
9 #container.banna{
10  @include opacity(90);
11 }
```

demo.css

```
1 #container {
2   opacity: 50;
3 }
4
5 #container.banna {
6   opacity: 90;
7 }
```

- 多参数混合

demo.scss

```
1 @mixin horizontal-line ($border:1px dashed #ccc, $padding:10px){
2   border-bottom:$border;
3   padding-top:$padding;
4   padding-bottom:$padding;
5 }
6 div.top_nav{
7   @include horizontal-line(1px solid #ccc);
8 }
9 .top_nav li {
10  @include horizontal-line($padding:15px);
11 }
```

demo.css

```
1 div.top_nav {
2   border-bottom: 1px solid #ccc;
3   padding-top: 10px;
4   padding-bottom: 10px;
5 }
6
7 .top_nav li {
8   border-bottom: 1px dashed #ccc;
9   padding-top: 15px;
10  padding-bottom: 15px;
11 }
```

1-4.6 继承扩展——inheritance (@extend)

在 SASS 中，通过继承/扩展来减少重复代码，可以让一个选择器去继承另一个选择中所有的样式。

继承某个样式的同时，也会继承样式的扩展。

● 案例

```
06extend.scss
1  .alert{
2    font-size:12px;
3  }
4
5  .alert a{
6    text-decoration: none;
7    &:hover{
8      color:#00BFFF;
9    }
10 }
11
12 .alert_info {
13   @extend .alert;
14   background-color:#DB7093;
```

```
06extend.css
1  .alert, .alert_info {
2    font-size: 12px;
3  }
4
5  .alert a, .alert_info a {
6    text-decoration: none;
7  }
8  .alert a:hover, .alert_info a:hover {
9    color: #00BFFF;
10 }
11
12 .alert_info {
13   background-color: #DB7093;
14 }
```

1-4.7 Partials && @import

CSS 本身包含一个指令@import，但是 CSS 中的@import 每次执行都会发送一次新的请求都会消耗一定的资源

SASS 中扩展了这个指令，会将包含的编译成一个 css 文件，切割成小的部分 (Partials) 包含进来进行处理。

Partials 这样的文件，命名规范是以下划线开头的，这样的 scss 文件不会被编译成 css

文件。

Partials 是用来定义公共样式或者组件的样式的，专门用于被其他的 scss 文件 import 进行使用的

在 SCSS 文件中引入指令@import 在引入 Partials 文件时，不需要添加下划线。详细参考案例代码。

- 案例：

```
_base.scss
1  * {
2    margin: 0;
3    padding: 0;
4  }
```

```
07Partials.scss
1  @import "base";
2
3  div.box {
4    font-size: 12px;
5  }
```

```
07Partials.css
1  * {
2    margin: 0;
3    padding: 0;
4  }
5
6  div.box {
7    font-size: 12px;
8  }
```

1-4.8 注释

SASS 中提供了三种注释

- 多行注释

- 在编译输出的 css 文件中会保留，压缩输出格式中不会保留

```
/*  
 * 多行注释  
*/
```

- 单行注释
 - 在输出 css 文件时不保留

```
// 单行注释
```

- 强制注释
 - 在多行注释的开头，添加感叹号！表示强制保留

```
/*!  
 * 强制注释  
*/
```

- 案例：

```
08comment.scss  
1  /*!  
2   * the comment will be stay here!  
3   */  
4   @import "base";  
5   /*  
6    this is the comment  
7    multipart comment  
8   */  
9   div.box {  
10    font-size: 12px; // this is the comment  
11  }
```

```
08comment.css  
1  /*!  
2   * the comment will be stay here!  
3   */  
4   * {  
5    margin: 0;  
6    padding: 0;  
7  }  
8  
9   /*  
10    this is the comment  
11    multipart comment  
12   */  
13  div.box {  
14    font-size: 12px;  
15  }
```

1-4.9 数据类型

Sass 支持 7 种主要的数据类型

- 数字(例如：1.2, 13, 10px)
- 字符串(例如："foo", 'bar', baz)
- 颜色(例如：blue, #04a3f9, rgba(255, 0, 0, 0.5))
- 布尔值(例如：true, false)
- 空值(例如：null)
- 列表(list), 用空格或逗号分隔 (例如：1.5em 1em 0 2em, Helvetica, Arial, sans-serif)
- 映射(例如：(key1: value1, key2: value2))

1-4.10 数字&数字函数

在 SASS 中可以对数字进行运算

同时 SASS 支持数字函数的运算

```
C:\Users\FLEX>sass -i
>> abs(-10)
10
>> round(3.4)
3
>> round(3.5)
4
>> ceil(3.1)
4
>> floor(4.1)
4
>> percentage(500px/1000px)
50%
>> min(1, 2, 3, 4, 10)
1
>> max(1, 2, 3, 4, 10)
10
```

1-4.11 字符串

SASS 支持字符串的操作

```
ca. sass -i
10
>> "laomu" + hao
"laomuhao"
>> laomu + "hao"
"laomuhao"
>> laomu + "8080"
"laomu8080"
>> laomu - "hao"
"laomu-hao"
>> laomu - hao
"laomu-hao"
>> nihao / laomu
"nihao/laomu"
>> 123 * abc
SyntaxError: Undefined operation: "123 times abc".
```

1-4.12 字符串函数

同时 SASS 对字符串的操作有一些封装的函数的支持，方便快捷的处理字符串操作。
更多操作请参考官方文档。

```
>> $greeting:"hello sass"
"hello sass"
>>
SyntaxError: Invalid CSS after "": expected expression (e.g. 1px, bold), was ""
>> $greeting
"hello sass"
>> to-upper-case($greeting)
"HELLO SASS"
>> to-lower-case($greeting)
"hello sass"
>> str-length($greeting)
10
>> str-index($greeting, "sass")
7
>> str-index($greeting, "hello")
1
>> str-insert($greeting, "css", 7)
"hello csssass"
>> $greeting
"hello sass"
```

1-4.13 颜色

颜色的表示有很多种

- 十六进制 Hex : #ff0000 等等
- RGB:rgb(255, 0, 0)等等
- 字符串 : red, blue, green 等等
- hsl : hsl(0, 100%, 50%)等等
- 等等... ..

SASS 支持所有这些颜色的表示方式

1-4.14 颜色函数——rgb & rgba

通过 rgb() 的形式进行颜色的控制【红、绿、蓝】

```
8 ■ background-color: rgb(255, 193, 50);
```

1-4.15 颜色函数 hsl & hsla

通过 hsl() 的形式进行颜色的控制【色相、饱和度、明度】

```
8 ■ background-color: hsl(38, 100%, 76%)
```

1-4.16 颜色函数 adjust-hue

- 颜色色相修改——H [hsl]

adjust-hue(要调整的颜色, 调整的值)

```
1 ■ $primary-color:#006699;
2
3  body{
4  ■   background-color:adjust-hue($primary-color, 120deg);
5  }
```

```
1  body {
2  ■   background-color: #990066;
3  }
```

1-4.17 颜色函数 lighten & darken

- 颜色明度修改——L [hsl]
 - lighten 让颜色更亮
 - darken 让颜色变暗

```
1 ■ $base-color:#888;
2   $light-color:lighten($base-color, 20%);
3   $darken-color:darken($base-color, 20%);
4
5  body{
6  ■   background-color:$base-color;
7     border:$light-color;
8     color:$darken-color;
9  }
```

```
1  body {
2  ■   background-color: #888;
3  ■   border: #bbbbbb;
4  ■   color: #555555;
5  }
```

1-4.18 颜色函数 saturate & desaturate

- 颜色饱和度修改——S [hsl]

```
1 ■ $base-color:hsl(221, 50%, 50%);
2   $saturate-color:saturate($base-color, 20%);
3   $desaturate-color:desaturate($base-color, 20%);
4
5   body{
6 ■   background-color:$base-color;
7     border:$saturate-color;
8     color:$desaturate-color;
9   }
```

```
1   body {
2 ■   background-color: #4068bf;
3 ■   border: #265fd9;
4 ■   color: #5971a6;
5   }
```

1-4.19 颜色函数 opacity & transparentize

- 颜色透明度修改
 - opacity() 函数, 加深透明度
 - transparentize() 函数, 降低透明度

```
1 ■ $base-color:hsla(221, 50%, 50%, 0.5);
2   $opacity-color:opacity($base-color, 0.2);
3   $transparentize-color:transparentize($base-color, 0.2);
4
5   body{
6 ■   background-color:$base-color;
7     border:$opacity-color;
8     color:$transparentize-color;
9   }
```

```
1   body {
2 ■   background-color: rgba(64, 104, 191, 0.5);
3 ■   border: rgba(64, 104, 191, 0.7);
4 ■   color: rgba(64, 104, 191, 0.3);
5   }
```

1-4.20 列表——list

list 表示列表类型的值

在 CSS 中就是表示属性的一串值

列表中的值可以使用空格或者逗号分隔, 如

- border:#ccc solid 1px; 值就是列表
- font-family:Courier, "Lucida Console", monospace; 值也是列表

列表中可以包含其他的列表，如：

- padding:10px 5px, 5px 5px; 值的列表中有两个列表，用逗号分隔
- padding(10px 5px) (5px 5px); 可以用括号分开，编译成 css 时会去掉这些括号

1-4.21 列表函数

SASS 中的列表相当于其他语言中的数组，SASS 也提供了一些函数方便列表的操作

```
C:\Users\FLEX>sass -i
>> length(5px 10px)
2
>> length(5px 10px 5px 0)
4
>> nth(5px 10px, 1)
5px
>> nth(5px 10px, 2)
10px
>> nth(5px 10px, 3)
SyntaxError: List index is 3 but list is only 2 items long for `nth`
>> index(1px solid red, solid)
2
>> index(1px solid red, red)
3
>> index(1px solid red, 1px)
1
>> index(1px solid red, 1)
1
>> index(1px solid red, 1i)
null
>> append(5px 10px, 5px 5px)
(5px 10px 5px 5px)
>> join(1px 10px, 12px 19px)
(1px 10px 12px 19px)
>> join(1px 10px, 12px 19px, comma)
(1px, 10px, 12px, 19px)
>> _
```

length: 获取列表长度

nth: 获取指定位置的列表项

index: 获取某个元素在列表中的位置，如果没有查询到返回 null

append: 给指定的第一个列表添加一个列表项

join: 合并列表

1-4.22 map 和相关函数

map 就是列表项目中带名称的列表

- \$map : (key1:value1, key2:value2, key3:value3)
 - \$var(key1:value1, key2:value2..): 声明一个 Map
 - length(\$map): 获取 map 中的元素对个数
 - map-get(\$map, key): 获取 \$map 中名称为 key 的值
 - map-keys(\$map): 获取指定 \$map 中所有的 key
 - map-values(\$map): 获取指定 \$map 中所有的 value
 - map-has-key(\$map, key): 判断在 \$map 中是否包含指定的 key
 - map-merge(\$map1, \$map2): 将 \$map1 和 \$map2 合并在一起
 - map-remove(\$map, key): 将指定名称的 key 从 \$map 中移除

```
C:\Users\FLEX>sass -i
>> $colors:(light:#ffffff, dark:#000000)
(light: #ffffff, dark: #000000)
>> length($colors)
2
>> map-get($colors, light)
#ffffff
>> map-get($colors, dark)
#000000
>> map-keys($colors)
("light", "dark")
>> map-values($colors)
(#ffffff, #000000)
>> map-has-key($colors, light)
true
>> map-has-key($colors, gray)
false
>> map-merge($colors, (light-gray:#e5e5e5))
(light: #ffffff, dark: #000000, light-gray: #e5e5e5)
>> $colors:map-merge($colors, (light-gray:#e5e5e5))
(light: #ffffff, dark: #000000, light-gray: #e5e5e5)
>> $colors
(light: #ffffff, dark: #000000, light-gray: #e5e5e5)
>> map-remove($colors, ligh, dark)
(light: #ffffff, light-gray: #e5e5e5)
```

1-4.23 布尔值

SASS 中的布尔值，跟其他语言一样，都是用来表示真/假的逻辑判断的。

取值：true/false, sass 中可以使用比较运算符，返回的就是布尔值

- 比较运算符
 - >、>=、<、<=、!=、==
- 逻辑运算符
 - and、or、not

```
cmd C:\Windows\system32\cmd.exe - sass -i
C:\Users\FLEX>sass -i
>> 5px > 3px
true
>> 4px < 2px
false
>> (5px > 3px) and (5px > 10px)
false
>> (5px > 3px) and (5px < 10px)
true
>> (5px > 3px) or (5px > 10px)
true
>> (5px < 3px) or (5px > 10px)
false
>> not(5px > 3px)
false
>> not(5px < 3px)
true
```

1-4.24 interpolation

Interpolation 可以将一个值插入到另一个值中。

SASS 中可以将表达式放在#{ $\$$ variable}中，用于使用变量的值

```
 $\$$ version:0.0.1;
/*项目版本#{$version}*/

 $\$$ name:"info";
 $\$$ attr:"border";
.alert-#{$name} {
  #{$attr}-color:#ccc;
}
```

1-4.25 控制指令——Control Directives

SASS 中为了更加方便的处理一些带有逻辑性的样式，如满足某些条件的时候使用指定的样式，或者根据指定列表中的项目循环输出样式等，提供了一些控制指令进行处理

- @if: 条件控制指令
- @for: 循环指令
- @each: 循环指令
- @while: 循环指令

1-4.26 @if

@if 指令是 SASS 中的一个控制指令，用于在表达式满足条件（true）的时候输出指定的样式，在不满足条件（false）或者表达式为 null 的情况下输出其他的样式

```
@if 条件 {  
    // 当条件为真时执行的样式  
}
```

同样，也可以通过@else if 和@else 指令结合，进行多条件的判断

```
1  $use-prefixes: true;  
2  $theme: "dark";  
3  body {  
4      @if $theme == dark {  
5          background-color: black;  
6      } @else if $theme == light {  
7          background-color: white;  
8      } @else {  
9          background-color: grey;  
10     }  
11 }  
  
12  
13 .rounded {  
14     @if $use-prefixes {  
15         -webkit-border-radius: 5px;  
16         -moz-border-radius: 5px;  
17         -ms-border-radius: 5px;  
18         -o-border-radius: 5px;  
19     }  
20     border-radius: 5px;  
21 }
```

1-4.27 @for

@for 指令在 SASS 中用于重复处理一组指令

有两种表现形式

- @for \$var from <开始值> through <结束值>
- @for \$var from <start> to <end>

to 和 through 都是表示一个区间，唯一的区别就是停止循环的地方不一样。\$var 可以是任意一个变量名称如\$i，<start>和<end>是 SASS 表达式并且必须是整数

```
$columns: 4;

@for $i from 1 through $columns {
  .col-#{ $i } {
    width: 100% / $columns * $i;
  }
}
```

1-4.28 @each

@each 在 Sass 中主要被用来进行列表或者映射数据的循环

主要表示形式：@each \$var in <list>

\$var 可以是任意变量名称，<list>是 SASS 表达式并且必须是 List

```
$icons: success error warning;

@each $icon in $icons {
  .icon-#{ $icon } {
    background-image: url(../images/icons/#{ $icon }.png);
  }
}
```

1-4.29 @while

@while 指令在 SASS 中用于循环重复处理样式，知道@while 表达式返回 false

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

```
.item-6 {
  width: 12em; }

.item-4 {
  width: 8em; }

.item-2 {
  width: 4em; }
```

1-4.30 用户自定义函数——function

函数的功能主要是数据的运算，SASS 中可以将一些值交给函数进行处理，具体的处理方式由定义的函数具体的设计确定。

```
@function 函数名称(参数列表){  
    // 数据处理  
}
```

```
$colors: (light: #ffffff, dark: #000000);  
  
@function color($key) {  
    @return map-get($colors, $key);  
}  
  
body {  
    background-color: color(light);  
}
```

1-4.31 警告 VS 错误

在自己设计的函数或者 Mixin 中，可以包含一些警告或者错误提示信息，用户在错误使用函数或者 mixin 时，就会看到这样的错误提示。

- @warn: 警告信息——会出现在命令行窗口中，编译提示
- @error: 错误信息——会出现在编译后的 css 文件中，错误提示

@warn message; 警告信息，警告信息一般会在执行 scss 程序生成 css 时触发，所以出现在命令行中。

@error message; 错误信息，错误信息直接显示在编译的 css 文件中。

```
$colors: (light: #ffffff, dark: #000000);  
  
@function color($key) {  
    @if not map-has-key($colors, $key) {  
        @warn "在 $colors 里没找到 #{ $key } 这个 key";  
    }  
  
    @return map-get($colors, $key);  
}  
  
body {  
    background-color: color(gray);  
}
```

```
>>> Change detected to: sass/style.scss  
WARNING: 在 $colors 里没找到 gray 这个 key  
on line 5 of sass/style.scss
```

```
$colors: (light: #ffffff, dark: #000000);  
  
@function color($key) {  
  @if not map-has-key($colors, $key) {  
    @error "在 $colors 里没找到 #{ $key } 这个 key";  
  }  
  
  @return map-get($colors, $key);  
}  
  
body {  
  background-color: color(gray);  
}
```

测试

1-5 SASS 使用注意事项

参考附件中《[编写 SASS 的八个技巧](#)》及文中附带的链接文章。

在时间充裕的情况下，最后页面，使用 SASS 完成样式处理。