Sass

变量

声明变量 `$highlight-color:#F90;`

```
$nav-color:#F90;
nav {
  $width: 100px;
  width: $width;
  color:$nav-color;
}
//编译后
nav {
  width: 100px;
  color:#F90;
}
```

变量作用域 块级作用域{}

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

变量引用变量

嵌套

选择器嵌套

```
#content {
  background-color:#f5f5f5;
  article {
    h1 { color:#333 }
    p { margin-bottom: 1.4em }
  }
  aside { background-color:#EEE }
}
/* 编译后 */
#content { background-color:#f5f5f5 }
#content article h1 { color:#333 }
#content article p { margin-bottom: 1.4em }
#content aside { background-color:#EEE }
```

后代选择器 空格

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

父选择器 & & 被父选择器直接替换

```
article a {
  color:blue;
  &:hover { color:red }
}
/*编译后*/
article a { color:blue }
article a:hover { color:red }
```

群组选择器

```
.container {
  h1, h2, h3 {margin-bottom: .8em}
}
/*编译后*/
.container h1, container h2, container h3 { margin-bottom: .8em }
```

```
nav, aside {
  a {color:blue}
}
/*编译后*/
nav a, aside a {color:blue}
```

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。

子组合选择器 > 和同层组合选择器 +和 ~

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 }
```

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 }
```

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

属性嵌套

```
nav {
  border: {
    style: solid;
    width: 1px;
    color: #ccc;
  }
}
/*编译后*/
nav {
  border-style: solid;
  border-width: 1px;
  border-color: #ccc;
}
```

缩写

常规

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

导入

@import

CSS 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。

SASS 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。

sass局部文件

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。

约定命名 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。

@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";。

默认变量值

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;
}
```

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。

```
/*_blue-theme.scss的局部文件*/
aside {
  background: blue;
  color:white;
}
/*导入到一个CSS规则内*/
.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:#fff;
  }
}
```

嵌套导入

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。

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。

混合器

```
这种方式跟JavaScript的function很像：

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

使用

```
@mixin no-bullets {
  list-style:none;
  li {
    list-style-image: none;
    list-style-type: none;
    margin-left:0px;
    }
  }
ul.plain {
  color:#444;
  @include no-bullets;
}
//sass最终生成
ul.plain {
  color:#444;
  list-style:none;
}
ul.plain li {
  list-style-image: none;
  list-style-type: none;
  margin-left:0px;
}
```

基本使用：
通过sass的混合器实现大段样式的重用
@mixin标识符来定义
通过@include来使用
本质上是（变量）

何时使用

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。

混合器在某些方面跟css类很像。都是让你给一大段样式命名，所以在选择使用哪个的时候可能会产生疑惑。最重要的区别就是类名是在html文件中应用的，而混合器是在样式表中应用的。
这就意味着类名具有语义化含义，而不仅仅是一种展示性的描述：用来描述html元素的含义而不是html元素的外观。而另一方面，混合器是展示性的描述，用来描述一条css规则应用之后会产生怎样的效果。
.notice是一个有语义的类名。如果一个html元素有一个notice的类名，就表明了这个html元素的用途：向用户展示提醒信息。rounded-corners混合器是展示性的，它描述了包含它的css规则最终的视觉样式，尤其是边框角的视觉样式。
混合器和类配合使用写出整洁的html和css，因为使用语义化的类名亦可以帮你避免重复使用混合器。

传参

```
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
  );
}
```

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

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

参数默认值
定制混合器生成的精确样式

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。

静默注释

```
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*/
}
```

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

选择器继承

使用

```
//通过选择器继承继承样式
.error {
  border:1px solid red;
  background-color:#fdd;
}
.seriousError {
  @extend .error;
  border-width:3px;
}
以class="seriousError"修饰的html元素最终的展示效果就好像是class="seriousError error"。
```

@extend语法使用

```
//seriousError从.error继承样式
.error a{  //应用到.seriousError a
  color:red;
  font-weight:100;
}
h1.error { //应用到h1seriousError
  font-size:1.2rem;
}
.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，
```

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。

某个类（比如说.seriousError）另一个类（比如说.error）的细化

何时使用