# sass揭秘之变量

作者：结一 日期：2013-08-02 点击：4037

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

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

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

[sass揭秘](http://www.w3cplus.com/blog/tags/377.html" \t "http://www.w3cplus.com/preprocessor/_blank)

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

因为文章内含有很多sass代码，如需自己动手查看编译结果，推荐使用[sassmeister](http://sassmeister.com/" \t "http://www.w3cplus.com/preprocessor/_blank)这款在线编译工具，方便你阅读学习。

刚开始sass的人，也许其最大的抱怨就是sass的变量了，尤其是那些了解less变量的，更是会说sass的变量做得不如less的变量好，less中有明显的全局变量和局部变量之分，但是到sass这边来，好像有点乱了，分不清了。其实不管sass的变量还是less变量都是有其设计的原理的，有着不一样的科学性，下面我们简单来了解下。

## **全局变量和局部变量**

要了解sass或less的变量，有必要先了解下js的变量定义，先来一段js的代码：

var a = 1;

(function(){

a = 5;

})();

console.log(a);*//5*

上面这段代码，匿名函数里面的a因为没有使用var关键字来定义，所以当我们在函数外打印a的时候，得到的是5，改变了一开始定义的1

var a = 1;

(function(){

var a = 5;

})();

console.log(a);*//1*

而这段代码，匿名函数里面的a使用了var关键字来定义，所以当我们在函数外打印a的时候，得到的是1，一开始定义的值。

如果你明白了上面两个的区别，那下面就好办了，如果你还没有明白，那么建议先去看下js的变量申明。 也许你现在已经看出眉目了，其实sass的变量设计用的是上面两段代码中的第一段思想，即在局部不使用var来申明，而less用的是第二段思想，即局部使用var来申明，这就是很多人所说的为什么sass没有局部变量。我们来两段代码测试下：

sass版

$color:red;

p{

$color:blue;

color:$color;//blue

}

a{

color:$color;//blue

}

less版

@color:red;

p{

@color:blue;

color:@color;//blue

}

a{

color:@color;//red

}

简单总结下：如果全局申明了一个变量a，sass中如果使用到局部中是属于改变a的值，所以后来所有的都是改变之后的值；而less中使用到局部变量则属于在局部重新定义a的值，不影响其他地方。说到底，其实sass和less的变量其实都是js变量的表现形式，两者都有其科学性，只是设计思路不同而已。

当然说到这里，可能有人会说了，那sass就没有局部变量喽。其实不然，正确的说法应该是：在有全局变量的情况下，sass是没有局部变量的。意思是如果要看到sass的局部变量，请不要设置全局变量。代码为证：

p{

$color:blue;

color:$color;//blue

}

a{

color:$color;//$color未定义

}

这是一个没有定义全局变量$color的例子，上面的p元素的color为blue，而a里面就会报错因为$color没定义，它没有获取到p元素里面定义的那个$color变量，这证明了其实sass同样存在局部变量，只是这个前提是得没有全局变量。

相信到这里，大家应该对sass的变量有所了解，不至于那么迷惑吧。下面我们再来两段代码：

p{

$color:blue;

color:$color;//blue

a{

color:$color;//blue

}

}

这一段代码中，a嵌套在p中，然后在p元素中定义了$color这个变量，a里面是可以访问到$color变量的。

p{

$color:blue;

color:$color;//blue

a{

$color:red;

color:$color;//red

}

background-color:$color;//red

span{

color:$color;//red

}

}

div{

color:$color;//$color未定义

}

估计这里有人迷惑了，我们来分析下吧：因为sass的代码是从上到下解析的，所以这段代码执行到p的color值的时候颜色是上面的blue色，然后由于我们在a中又改变了$color的值为红色，所以后面所有p元素范围内用到$color这个变量的都为红色，这反应到我们后来的bgcolor和span的color上面，而p元素之外还是未定义。

这里吐个槽吧，其实个人觉得正是由于这个，使得开发sass的小心定义变量，以避免污染；而less呢却刚好相反，估计到时候就是全局局部满天飞的情况，代码一团糟。你想选择哪种呢？

关于sass的全部变量和局部变量就说到这，既然这篇文章的标题说了是sass变量揭秘，光这全局变量和局部变量肯定是满足不了大家的胃口的，下面我们继续说下sass变量的另一个创新点，变量默认值。

## **变量默认值!default**

可能很多人对这个不是很熟悉，或者觉得这个其实就是个幌子，挂羊头卖狗肉的家伙。其实不然，它的来头可不小，而且是非常的科学，我是佩服的五体投地，因为它从根本上解决了困扰我多年的css组件化开发。简单来两段比较的代码：

无!default

$color:red;$color:blue;

p{

color:$color;//blue

}

有!default

$color:red;$color:blue !default;

p{

color:$color;//red

}

上面说了，sass编译css是从上到下的，后面会覆盖前面的，所以第一段无!default的解析的是blue，而第二段代码由于有了!default，打破了这个规则，使用了前面定义的red。有意思吧，当然这样的使用是体现不了它伟大的价值的。

简单介绍下它的作用吧：假设变量申明带有!default，那么如果在此申明之前没有这个变量的申明，则用这个值，反之如果之前有申明，则用申明的值。当然如果你先!default申明，然后再申明一次，那就没什么意思了，这就是基本的变量覆盖，第一次申明的有无!default都一样。所以你要区分这两种情况：

*//第一种，使用默认值 //变量申明带有!default，但是之前没有这个变量的申明* $color:blue !default;

p{ color:$color;*//blue }*

*//第二种，使用前面定义的值*$color:red; *//变量申明带有!default，但是前面还有这个变量的申明*$color:blue !default;

p{

color:$color;*//red*

}

*//第三种错误的用法，先!default申明*$color:red !default;$color:blue;

上面的第三种错误用法其实和下面的这个覆盖写法是一样一样的：

*//覆盖写法*$color:red;$color:blue;

能一口气看到这里的，应该有点时间，建议简单活动下，摇摇脑袋，伸伸懒腰，因为下面的更加精彩。

其实长久以来，css之所以不能组件化开发，有两大原因：第一个是@import的样式不能合并在一个文件中（这里排除使用压缩工具来合并）；第二个问题就是这里说到的变量问题了。感谢sass帮我们全解决了，大笑三声。

现在假设我们有个scss文件，这里暂且叫做\_imgstyle.scss，代码如下：

*// 变量//---------------------------------*$imgStyleBorder: 1px solid *#ccc !default;*$imgStylePadding: 2px !default;$imgStyleRadius: 8px !default;

*// mixin//---------------------------------*

@mixin img-border($border:$imgStyleBorder,$padding:$imgStylePadding){

border: $border;

padding: $padding;

}

@mixin img-rounded($radius:$imgStyleRadius){

border-radius:$radius;

}

*//样式//---------------------------------*

.img-border{

@include img-border;

}

.img-rounded{

@include img-rounded;

}

接下来我们要在style.scss这个文件里面调用\_imgstyle.scss，代码如下：

*//导入\_imgstyle.scss*@import 'imgstyle';

现在问题来了，如果我们对默认的padding为2px不满意，要改为5px，我们有如下方法：

方法一：重新覆写

*//导入\_imgstyle.scss*@import 'imgstyle';

.img-border{

padding:5px;

}

解析成的css：

.img-border {

border: 1px solid #cccccc;

padding: 2px;}.img-rounded {

border-radius: 8px;}.img-border {

padding: 5px;}

方法二：改变@include的参数

*//导入\_imgstyle.scss*@import 'imgstyle';

.img-border{

@include img-border($imgStyleBorder,5px);

}

解析成的css：

.img-border {

border: 1px solid #cccccc;

padding: 2px;}.img-rounded {

border-radius: 8px;}

.img-border {

border: 1px solid #cccccc;

padding: 5px;}

显而易见，上面的方法都会产生重复的代码，不科学啊。当然这时候可能有人会说了，你脑子浸水了呗，直接在\_imgstyle.scss文件中，修改$imgStyleBorder为5px不就得了。当然如果你要的是每个项目使用这个样式的时候都拷贝一份这个，然后打开把变量修改成你要的值，那么我只好承认我脑子浸水了，不仅脑子浸水，还得吐血了。

现在请上面那些山寨土鳖方法闪一边去，该我们的!default出场了，废话少说，上代码：

*//申明$imgStylePadding为5px*

$imgStylePadding: 5px;

*//导入\_imgstyle.scss*@import 'imgstyle';

解析成的css：

.img-border {

border: 1px solid #cccccc;

padding: 5px;}

.img-rounded {

border-radius: 8px;}

无须去源文件中修改，且解析出来无重复代码，完美！这才是!default的价值，这为组件式开发，更改调用组件里面的变量的值带来了彻底的变化。如果less真没有这个变量的默认值的话，那less的同学们估计得伤心死了。

重量级的都介绍完了，下面把其他的一些小罗罗也简单介绍下吧。

## **变量后面加...**

在css3出现以前，你是绝对没有看到过的，因为它就是为css3而设计的。css3在给css带来翻天覆地的变化之外，也给sass的@mixin传递参数带来纠结了。一般来说我们的@mixin传递的参数是以,来分隔的，但是css3的一些属性可以设置多个值，并且多个值以,分开，如box-shadow:0 0 3px rgba(0,0,0,0.3),inset 0 0 3px rgba(255,255,255,0.3);这让@mixin如何给box-shadow传递参数啊。所以这种变量后面加...的变量就出现了。

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

-webkit-box-shadow:$shadow;

-moz-box-shadow:$shadow;

box-shadow:$shadow;

}

这样就完美解决了这个需求了，漂亮吧哈哈。注意这种变量加...的方式只出现在传递参数中，你可以看到上面的大括号里面的变量都是没有...。除了box-shadow，gradient也可以用，反正能使用多个值的属性里面都可以用。

## **变量用#{}包裹**

一般来说，我们设置的变量都是用于属性值的，而如果用在属性或者选择器上，就得以#{}包裹起来了。

$btnClass: btn !default;$borderDirection: top !default;

.*#{$btnClass}{*

border-*#{$borderDirection}:1px solid #ccc;*

}

解析成的css：

.btn{

border-top:1px solid #ccc;}

## **多个变量一起申明**

其实这个还是很实用的，把多个相关的值写在一个变量里，然后通过nth($var,index)来获取第几个值。

$linkColor: red blue !default;

a{

color:nth($linkColor,1);

&:hover{

color:nth($linkColor,2);

}

}

解析成的css：

a {

color: red; }a:hover {

color: blue; }

关于sass变量揭秘到此为止。如果你对sass比较感兴趣但是还不会，可以试试我们的[sassGuide](http://www.w3cplus.com/sassguide/index.html" \t "http://www.w3cplus.com/preprocessor/_blank)教程，如果已经开始使用sass了，欢迎试用[sassCore](https://github.com/marvin1023/sassCore" \t "http://www.w3cplus.com/preprocessor/_blank)这个库。

顺便说下，本人的面向熟悉sass人员开发的tobe即将上线，欢迎关注，也欢迎到时拍砖。