# sass揭秘之@if，@for，@each

作者：结一 日期：2013-08-07 点击：4008

[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。

经过上两篇揭秘，大家心里对sass应该有了很好的认知感了，这篇文章基于前面两篇为基础，请确保你先浏览了前面的两篇文章，不然可能给你带来疑惑感觉。

1. [sass揭秘之变量](http://www.w3cplus.com/preprocessor/sass-basic-variable.html" \t "http://www.w3cplus.com/preprocessor/_blank)
2. [sass揭秘之@mixin，%，@function](http://www.w3cplus.com/preprocessor/sass-mixins-function-placeholder.html" \t "http://www.w3cplus.com/preprocessor/_blank)

这篇文章关键是对sass判断或循环的语法熟悉下，至于逻辑这东西，和其他的语言如js，php什么的是一样一样的，没什么说头。经过前两篇的积累，也就不需要那么啰嗦了，所以只写scss代码。现在你就可以想想css如果用上这些东西会是什么样子的呢？

## **@if**

这个@if就不用解释吧，一看都知道是条件判断。这个东西对于浏览器兼容这块可以出不少力量；然后对于写一些基础的scss，控制样式的输出也是一大利器；当然还有很多很多了，反正是个必备的好东西就是了。先来个简单的例子吧：

$lte7:true !default;*//是否兼容ie6,7*

*//inline-block//ie6-7 \*display: inline;\*zoom:1;*

@mixin inline-block {

display: inline-block;

@if $lte7 {

\*display: inline;\*zoom:1;

}

}

如果变量$lte7为真，就输出\*display: inline;\*zoom:1;，这个例子看着有点熟悉吧，前面@mixin已经说了，哈哈。sass的东西其实也就是这么多，转来转去就转回来了。

既然有@if，那肯定有@else啊

$filter:false !default; *//是否开启ie滤镜//背景色半透明*

@mixin bgcolor-alpha($bgcolor: rgba(0,0,0,.5)){

color:*#fff;*

@if $filter{

filter:progid:DXImageTransform.Microsoft.gradient(enabled='true',startColorstr='#{ie-hex-str($bgcolor)}', endColorstr='#{ie-hex-str($bgcolor)}');

}@else{

background-color: *#333;*

}

background-color:$bgcolor;

}

这是半透明rgba背景的一段代码，高级浏览器用rgba，ie6-8如果开启滤镜用滤镜，不开启滤镜就用纯色，常用于图片下方浮现标题。至于多条件的，可以参考sass揭秘之@mixin,%,@function里面的神来之笔的@mixin prefixer

当然也不可能总是判断一个变量的真假那么简单，没有或与非的情况吧。sass的@if用not,or,and分别表示非，或，与。

$a: false !default;$b: true !default;

@if not($a){

p{

color:red;

}

}

div{

font-size:14px;

@if $a or $b{

width:300px;

}

}

li{

line-height:24px;

@if $a and $b{

float:left;

}

}

想想还是漏了个，sass用==,!=分别表示等于与不等于。

$radius: 5px !default;

.box-border{

border:1px solid *#ccc;*

padding:10px;

@if $radius != 0{

border-radius:$radius;

}

}

$sizeClass: small !default;

.size{

@if $sizeClass == 'small'{

padding:5px;

}@else{

padding:10px;

}

}

是不是感觉很简单，本来就没什么特别的，判断和循环逻辑都是一样的，就是可能sass里面表现不一样吧。

## **@for**

for循环有两种形式，分别为：@for $var from through 和@for $var from to 。$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。先来个简单的：

@for $i from 1 through 3 {

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

}

上面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：

*// class span1-$gridColumns// span的class循环输出，通过变量$gridSpanSwitch来控制是否输出//-----------------------------------------------------* $gridColumns: 12 !default;$gridcolumnWidth: 60px !default;$gridGutter: 20px !default;

%span-base{

float:left;

margin-left:$gridGutter / 2;

margin-right:$gridGutter / 2;

}

@for $i from 1 through $gridColumns {

.span*#{$i} {*

@extend %span-base;

width:($gridcolumnWidth + $gridGutter) \* $i - $gridGutter;

}

}

解析后的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;}

这样一循环，比一个个写爽多了吧，因为float和margin什么的都一样，所以我们使用占位选择器申明，然后@extend调用，那样就是组合申明了。当然如果你要一个更复杂的包括百分比和固定宽度切换的可以看下[sassCore](https://github.com/marvin1023/sassCore" \t "http://www.w3cplus.com/preprocessor/_blank)中的\_grid.scss，这个网格系统就很强大了。

## **@each**

语法：@each $var in

循环一个列表里面所有的值，然后去做点什么呗，那到底做点什么呢，这是个问题。你可以先想个三分钟试试，我们现在能用@each给css赋予什么魅力。

我来扯点其他的。其实@for和@each还是很少用的，但是只要它一上场，对于css来说一般都具有神来之笔，这就是要么不出手，一出手就震惊全场。

好了，插了上面一段后，我们来说说@each的官方实例：

@each $animal in puma, sea-slug, egret, salamander {

.*#{$animal}-icon {*

background-image: url('/images/#{$animal}.png');

}

}

解析后的css：

.puma-icon {

background-image: url('/images/puma.png'); }.sea-slug-icon {

background-image: url('/images/sea-slug.png'); }.egret-icon {

background-image: url('/images/egret.png'); }.salamander-icon {

background-image: url('/images/salamander.png'); }

上面是对icon class的定义，不过icon现在大家一般都用sprite去做，所以这种单独的很少了，不过可以同样用这个来设置background-position，假设这个sprite图片是宽30px，高120px，每个icon大小为30px\*30px：

$sprite: puma sea-slug egret salamander !default;

%sprite-animal{

background: url('/images/animal.png') no-repeat;

}@each $animal in $sprite {

.*#{$animal}-icon {*

@extend %sprite-animal;

background-position:0 -(index($sprite,$animal)\*30px);

}

}

解析后的css：

.puma-icon, .sea-slug-icon, .egret-icon, .salamander-icon {

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

.puma-icon {

background-position: -30px;}

.sea-slug-icon {

background-position: -60px;}

.egret-icon {

background-position: -90px;}

.salamander-icon {

background-position: -120px;}

看起来还是比较帅的哈哈，但是sprite的图片有这么规整的还是很少的，现在有些网站搞sprite图片搞得太狠了，一般一个sprite那是包含各种大小的图片啊，不过这种网站如果维护频繁的话，那简直就是挖了天坑啊。

除了上面的sprite的运用，我们再来说个拍案叫绝的吧。自从bootstrap以后，框架遍地开花，而估计每个里面都会有的就是按钮的样式，而按钮的样式一般都要分颜色，大小吧，甚至按钮组。每个颜色一个class，每个大小还一个class，可是我们实际用的时候呢，估计能用个三分之一就不错了，反正最后就是那些定义好的按钮的样式远远超过了我们的需求，多大的浪费啊。

现在我们借用sass的@each就能帮我们控制这个问题，实用出发，需要什么就生成什么，不铺张不浪费。

//$btnColorClass: (primary *#0078E7 #fff) (green $green #fff);*$btnColorClass: (primary *#0078E7 #fff) (blue #00A3CF #fff) (orange #F47837 #fff) !default;*

@mixin btn-color($bgColor:*#e6e6e6, $textColor:#333){*

color:$textColor;

background-color: $bgColor;

border:1px solid darken($bgColor, 5%);

border-color:lighten($bgColor, 2%) darken($bgColor, 5%) darken($bgColor, 5%) lighten($bgColor, 2%);

&:hover{

background-color: darken($bgColor,5%);

color:$textColor;

}

}

@each $colorClass in $btnColorClass{

$class: nth($colorClass,1);

$bgColor: nth($colorClass,2);

$textColor: nth($colorClass,3);

.btn-*#{$class}{*

@include btn-color($bgColor,$textColor);

}

}

解析后的css：

.btn-primary {

color: white;

background-color: #0078e7;

border: 1px solid #006bce;

border-color: #007df1 #006bce #006bce #007df1;}.btn-primary:hover {

background-color: #006bce;

color: white;}

.btn-blue {

color: white;

background-color: #00a3cf;

border: 1px solid #008fb5;

border-color: #00abd9 #008fb5 #008fb5 #00abd9;}.btn-blue:hover {

background-color: #008fb5;

color: white;}

.btn-orange {

color: white;

background-color: #f47837;

border: 1px solid #f3681f;

border-color: #f57f41 #f3681f #f3681f #f57f41;}.btn-orange:hover {

background-color: #f3681f;

color: white;}

轻松搞定，我们默认有三个颜色，现在如果我们要搞自己的颜色，直接按照第一篇[sass揭秘之变量](http://www.w3cplus.com/preprocessor/sass-basic-variable.html" \t "http://www.w3cplus.com/preprocessor/_blank)来重新定义下变量就ok了，这里你可以直接去掉那段注释//$btnColorClass: (primary #0078E7 #fff) (green $green #fff);就会解析成这两种class。现在你除了发表感叹“真牛B”之外，你也许还会想这是怎么出来的。不明觉厉啊。

下面详细展开下：

其实这里的核心就是$btnColorClass这个变量，也许你以前看到的就是一个变量对应一个值，如果你看了前面两篇文章，你会看到一个变量对应两个或多个值，而这里又升级了，对应了好多个啊，而且还是有组织的。简单来说这就跟js的数组差不多，相当于数组里面还有数组。

这里补充个知识，sass的列表值大致有三种形式分隔，分别为空格，逗号和小括号。空格比较常见，逗号还好了，我们说的第一个@each的官网案例，$list就是以逗号分隔的。

现在转回来，我们这个$btnColorClass，使用了两种分隔，空格和小括号，我们@each循环先得到用小括号分隔的，然后再在@each里面通过nth取得小括号里面用空格分开的列表值。说完了，就这么简单，突然发现把变量说完了，这个答案就解开了。

@if，@for，@each就算讲完了，当然还有一个@while，暂时还没太用它，也没什么特别的，所以就不说了，以后有用到绝妙之处再来说。

经过这三篇sass揭秘文章，大家对sass的了解也应该深入许多了，是不是发现它原来还有如此的迷人魅力。揭秘到这里，其实大家现在可以去分析下[sassCore](https://github.com/marvin1023/sassCore" \t "http://www.w3cplus.com/preprocessor/_blank)的源码了，其实这三篇文章里面好多东西都是从sassCore里面抽出来的，sassCore本身的代码比这些可能要复杂点，但是如果你对着三篇文章理解没问题的，sassCore源码也不会是问题，如有问题可在群中提问

sass揭秘暂时到此为止，休息一段时间，如果反响比较好的话，将继续为大家说下sass实战，包括给我以前写的那篇[浅析sass死亡在团队合作中](http://www.w3cplus.com/blog/769.html" \t "http://www.w3cplus.com/preprocessor/_blank) 文章一个答案。