# sass揭秘之@mixin，%，@function

作者：结一 日期：2013-08-05 点击：7553

[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。

在阅读本文章之前，请先确认你已经阅读了上篇文章[sass揭秘之变量](http://www.w3cplus.com/preprocessor/sass-basic-variable.html" \t "http://www.w3cplus.com/preprocessor/_blank)，不然会给你带来疑惑的感觉。

其实很多人之所以对sass或less感兴趣，就是因为他们能使用变量和这个@mixin功能，而后面的%和@function知道的人就比较少了。所以说@mixin这个东西还是很有诱惑力的，没办法，广告做得好啊，大明星。这里之所以把%和@function和@mixin放在一起，当然并非无缘无故，一看@mixin和@function就是兄弟，长得那么像，而%这个后起之秀，更是在一定程度上抢了@mixin的不少风头。

这里先说@mixin和%，谁让它们有竞争关系呢，哈哈。@function这个家伙一看就是函数，先闪一边去。 首先@mixin可以传递参数，而%不行；然后@mixin的调用方式是@include，而%的调用方式是@extend；最后@include产生的样式是以复制拷贝的方式存在的，而@extend产生的样式是以组合申明的方式存在的。概念简单讲解完毕，现在进入代码实例，上战场才是真理。

为了方面测试，我们先约定建立一个\_mixin.scss文件，下面所有的有关@mixin，%和@function的一些定义全部写在这里，再建立一个style.scss来调用我们的\_mixin.scss文件，所以在style的里面先写上一句@import 'mixin';

## **@mixin**

先来一段无参数简单版本的@mixin（@mixin，%，@function全部放在\_mixin.scss文件中）：

// block得有宽度margin左右为auto才能居中

@mixin center-block {

margin-left: auto;

margin-right: auto;

}

这应该是最简单版本的@mixin了，不但没有参数，连样式都只有两条，不过还是很实用的。接下来我们来调用下（调用的全部放在style.scss文件中，先导入\_mixin文件）：

@import 'mixin'; *#header{*

width:1000px;

@include center-block;

}

.gallery{

width:600px;

@include center-block;

}

解析成的css：

#header {

width: 1000px;

margin-left: auto;

margin-right: auto;}.gallery {

width: 600px;

margin-left: auto;

margin-right: auto;}

很显然，上面两个margin左右为auto在各自的选择器中，当然运行是没有问题的，但是如果把这两个一样样式提出来组合申明下多好啊，一看质量就不一样了吗，高端大气上档次了哈哈。这个问题稍后留给我们的%来解决，我们继续@mxixin。

再来个无参数版的，但是包含浏览器兼容方面的：

$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;

}

}

上面的代码，有个$lte7全局变量，我们把这个变量提到\_mixin.scss文件的最上面。注意这里@mixin里面有个@if判断，这是为ie6,7对inline-block部分不兼容的一个处理，默认$lte7为true，意思是需要兼容ie6,7，那么就会输出判断里面的代码\*display: inline;\*zoom:1;，当我们不需要兼容的时候呢，话说高富帅搞的就是搞ie8+的，那设置$lte7为false就没\*display: inline;\*zoom:1;这两个家伙的事了，直接宣布其斩立决了。代码为证：

$lte7:false; @import 'mixin';

.inline-block{

@include inline-block;

}

这里注意：因为我们要重设$lte7为false，所以在@import 'mixin';之前先定义下$lte7:false;，这涉及到变量默认值的使用，如果你不了解请先查阅[sass揭秘之变量](http://www.w3cplus.com/preprocessor/sass-basic-variable.html" \t "http://www.w3cplus.com/preprocessor/_blank)

解析成的css：

.inline-block{

display:inline-block;}

当然如果没有$lte7:false;这个提前申明变量，那么解析成的css应该是这样的：

.inline-block{

display:inline-block;

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

从上面可以看出，如果@mixin里面放点判断，对浏览器的兼容还可以做点有意义的事，不用每次都写一大坨，同时还为以后升级带来一个暗门，直接改变下变量的值重新解析下就ok了，那些为兼容处理的代码统统消失，这比较爽。测试完这个之后，请把$lte7:false;删掉，因为后面还要用到其值true。

现在来个参数简单版的：

@mixin float($float:left) {

float: $float;

@if $lte7 {

display: inline;

}

}

够简单吧，float人人皆知啊。这里$float参数有默认值为left，我们调用下：

.fl{

@include float;

}

.fr{

@include float(right);

}

解析成的css：

.fl{

float:left;

display: inline;}.fr{

float:right;

display: inline;}

因为在传参数的时候$float设置了一个默认值为left，所以调用的时候@include float;和@include float(left);能产生一样的代码。这里先说下我琢磨出来的一个经验，如果某个@mixin无法设置默认的参数，那么这个@mixin要么可以用%来取代，要么就是个鸡肋@mixin，所以请定义@mixin的时候参考这两点判断是否有必要，特殊情况除外。

关于鸡肋@mixin等下再说，我们接着说下多个参数的@mixin：

*// 禁用样式，加!important*

@mixin disabled($bgColor:*#e6e6e6,$textColor:#ababab){*

background-color: $bgColor !important;

color: $textColor !important;

cursor: not-allowed !important;

}

两个参数，一个为背景色，一个为文本色，两个冒号后面的分别为默认值，直接调用@include diasbled;使用的就是默认值，虽然简单，我们还是调用下吧。

.disabled{

@include disabled;

}

解析后的css：

.disabled {

background-color: #e6e6e6 !important;

color: #ababab !important;

cursor: not-allowed !important;}

接着下一个实例，一个属性可以有多个属性值的，写到这里，看过[sass揭秘之变量](http://www.w3cplus.com/preprocessor/sass-basic-variable.html" \t "http://www.w3cplus.com/preprocessor/_blank)的人就想起来了，原来是传参的时候变量得加...:

*//错误定义方法*

@mixin box-shadow($shadow){

-webkit-box-shadow:$shadow;

-moz-box-shadow:$shadow;

box-shadow:$shadow;

}

为了给人说明这...，有必要先搞个错误的东西，那样你就会恍然大悟了。我们来调用下上面的错误定义方法：

.shadow1{

@include box-shadow(0 0 5px rgba(0,0,0,.3));//这个可以运行

}

.shadow2{

@include box-shadow(0 0 5px rgba(0,0,0,.3),inset 0 0 3px rgba(255,255,255,.5));//这个不可运行

}

上面两个代码，我们先运行第一个，会成功解析出css，而第二个就不行了，它就是孙猴子派来捣乱的。

第一个运行解析成的css为：

.shadow1{

-webkit-box-shadow:0 0 5px rgba(0,0,0,.3);

-moz-box-shadow:0 0 5px rgba(0,0,0,.3);

box-shadow:0 0 5px rgba(0,0,0,.3); }

为什么第二个不行呢，因为第二个我们给box-shadow设置了两个值，一个外阴影一个内阴影，并且是以，分开的。实际情况是，我们对box-shadow可以设置很多个值，随我们高兴，没有一定的。这个时候就有了为css3这些妖孽而生的传递的参数后面加...了，上代码：

*//正确定义方法*

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

-webkit-box-shadow:$shadow;

-moz-box-shadow:$shadow;

box-shadow:$shadow;

}

正确的东西得来总是那么不容易，话说一开始研究别人代码的时候，还以为这...是随便加上去好玩的呢。然后我写css3的@mixin的时候把...统统去掉，结果，结果就悲剧了，可以有多个属性值的，只能设置一个属性值，然后就是满天找bug了。注意这里只在传参的时候变量后面添加...，而在大括号内引用的时候是不用加...，接着你可以回过头测试下上面那两个代码了，保准ok！

看完...这个为css3而生的之后，我们再看一个为css3而成的东西@content，之所以提起来，是因为它也是应用在@mixin里面的。按常规来说，我们所有的样式都是在@mixin里面定义好的，然后使用的时候@include就拷贝了这段样式，但是@content改变了这一惯例，它其实没有定义样式，它是定义好了选择器，然后@include的时候，就是选择器定了，你写的样式都放在这个选择器里面。光文字介绍是不能解决问题的，还是实例比较有营养：

@mixin header{

*#header{*

@content;

}

}

我们来简单调用下：

@include header{

width:1000px;

height:200px;

.logo{

width:200px;

}

}

解析后的css：

#header {

width: 1000px;

height: 200px;}#header .logo {

width: 200px;}

看到没，这个选择器以#header为基础，然后@include header里面写的任何样式，都是在这个基础上的。明白@content与上面的其他的区别不，其他的@mixin调用的时候是这样的@include mixin-name($var1,$var2,...,$varn)，而这个@content的调用的时候是这样的@include mixin-name{},大括号里面就是@content表示的内容，里面css样式随便你写啊。

当然上面的@content实例是闲得蛋疼的为简单说明而写的，其实没有什么使用价值的，@content的使用价值其实体现在css3的media-queries，animation的keyframes定义，还有为浏览器兼容的定义。下面实例说明：

*//定义media-queries的最小最大宽度*@mixin screen($res-min, $res-max){

@media screen and ( min-width: $res-min ) and ( max-width: $res-max ){

@content;

}

}

*//定义animation的keyframes*@mixin keyframes($name){

@keyframes #{$name} {

@content;

}

}

*//定义所有不支持圆角的浏览器使用背景图片//得使用[modernizr](http://modernizr.com/)来检测，在html上加class*@mixin no-border-radius{

.no-border-radius{

@content

}

}

又到调用这步了，没办法，不验证下，产生点css，还是有点迷惑：

*#header{*

@include screen(780px,1000px){

color:red;

}

}

@include screen(780px,1000px){

body{

font-size:14px;

}

}

@include keyframes(show){

0% {

opacity:0;

}

100% {

opacity:1;

}

}

*//注意下面这两个的区别*

@include no-border-radius{

.box{

background:url(round-bg.gif) no-repeat;

}

}

.box{

@include no-border-radius{

background:url(round-bg.gif) no-repeat;

}

}

解析后的css：

@media screen and (min-width: 780px) and (max-width: 1000px) {

#header {

color: red;

}

}@media screen and (min-width: 780px) and (max-width: 1000px) {

body {

font-size: 14px;

}

}@keyframes show {

0% {

opacity: 0;

}

100% {

opacity: 1;

}

}.no-border-radius .box {

background: url(round-bg.gif) no-repeat;}.box .no-border-radius {

background: url(round-bg.gif) no-repeat;}

上面那个@include screen我们使用了两种方法去调用，第一种在选择器里面调用，第二种直接调用，两者生成的css是一样的，既然生成的样式没有什么区别，那如何使用呢？其实第一种方式强调的是以选择器为主，当screen是什么时候是什么值，而第二种调用方法强调以media-queries条件为主，可以方便组织所有在这个条件中的都写在一起。如果做响应式布局我们建议使用第二种方法，以断点为主来写样式，把某个断点下的样式全部写在一起。

为了表示media-queries的特殊，我们举了个反例，同样以两种方法调用@include no-border-radius，结果可以看到完全不一样啊，大家千万别以为是@include no-border-radius错了，其实它才是正确的。而media-queries是个为了大家方便使用的特殊案例。

@mixin说到这里，其主要的知识点也说完了，相信大家也收获不少了，为了表示对@mixin的敬意，我再挑几个@mixin来分析。

先来个我们常用的用border生成三角形的@mixin:

*// triangle*

@mixin triangle($direction, $size, $borderColor ) {

content:"";

height: 0;

width: 0;

@if $direction == top {

border-bottom:$size solid $borderColor;

border-left:$size dashed transparent;

border-right:$size dashed transparent;

}

@else if $direction == right {

border-left:$size solid $borderColor;

border-top:$size dashed transparent;

border-bottom:$size dashed transparent;

}

@else if $direction == bottom {

border-top:$size solid $borderColor;

border-left:$size dashed transparent;

border-right:$size dashed transparent;

}

@else if $direction == left {

border-right:$size solid $borderColor;

border-top:$size dashed transparent;

border-bottom:$size dashed transparent;

}

}

这个@mixin主要有三个变量：第一个是方向的，因为三角形根据箭头朝向有四种方向，我们对应常用的css属性top,right,bottom,left；第二个表示三角形的大小；第三个表示颜色。当然你可以挑你常用的那个设置为变量的默认值，那样调用常用的那个就比较简单了，直接@include triangle;就ok了。

下面我们再来个关于css3的神来之笔的@mixin，在说这个之前，先说下前面的那个box-shadow的@mixin，我们里面的样式是一条一条写的，如-webkit-box-shadow:$shadow;-moz-box-shadow:$shadow;box-shadow:$shadow;，这得多山炮啊，一条一条来，不简洁，不科学。下面欢迎我们的prefixer这个@mixin，它对css3的前缀定义有画龙点睛之妙。

*//是否支持某个浏览器的前缀，如果你不想支持，可以设置为false//----------------------------*$prefix-for-webkit: true !default;$prefix-for-mozilla: true !default;$prefix-for-microsoft: true !default;$prefix-for-opera: true !default;$prefix-for-spec: true !default; *// 标准版*

*// prefixer//----------------------------*

@mixin prefixer ($property, $value, $prefixes) {

@each $prefix in $prefixes {

@if $prefix == webkit and $prefix-for-webkit == true {

-webkit-*#{$property}: $value;*

}

@else if $prefix == moz and $prefix-for-mozilla == true {

-moz-*#{$property}: $value;*

}

@else if $prefix == ms and $prefix-for-microsoft == true {

-ms-*#{$property}: $value;*

}

@else if $prefix == o and $prefix-for-opera == true {

-o-*#{$property}: $value;*

}

@else if $prefix == spec and $prefix-for-spec == true {

*#{$property}: $value;*

}

@else {

@warn "Unrecognized prefix: #{$prefix}";

}

}

}

看到没，判断循环输出啊，到这里也许你还是不明白的，我们来调用它来构建css3的一些@mixin，你就明白了。

*//webki和标准*

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

@include prefixer(box-shadow, $shadow, webkit spec);

}

*//webkit moz 和标准*

@mixin box-sizing($type:border-box) {

*// border-box | padding-box | content-box*

@include prefixer(box-sizing, $type, webkit moz spec);

}

*//webkit moz o 和标准*

@mixin transform($property...) {

@include prefixer(transform, $property, webkit moz o spec);

}

现在来分析下上面的那个@mixin prefixer，第一个参数$property是属性，第二个参数$value是值，第三个参数$prefixes就是我们需要添加前缀的组合，因为目前来说前缀包括webkit,moz,o,ms，所以它就是这些值再加上一个标准spec的随意组合。里面对$prefixes进行循环判断，根据不同的值，为属性添加不同的前缀，精彩啊。突然觉得天空一下晴朗多了，爽吧。我们来调用下：

.box{

@include box-shadow(0 0 5px rgba(0,0,0,.3));

@include box-sizing;

@include transform(scale(2));

}

解析后的css：

.box {

-webkit-box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);

box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

box-sizing: border-box;

-webkit-transform: scale(2);

-moz-transform: scale(2);

-o-transform: scale(2);

transform: scale(2);}

说完神来之笔，最后我们再来一段我认为的一个优秀的鸡肋@mixin：

*//设置宽高，默认为auto*

@mixin size($size...) {

$size: if(length($size) > 0, $size, auto);

$width: nth($size, 1);

$height: nth($size, length($size));

@if $width != auto {

$width: if(unitless($width), $width + px, $width);

}

@if $height != auto {

$height: if(unitless($height), $height + px, $height);

}

width: $width;

height: $height;

}

先说下这里面涉及到的一些sass知识点：length($var)表示获取变量的长度，nth($var,index)获取变量第几个位置的值，unitless判断是否无单位，而这段if(unitless($width), $width + px, $width)其实是个三目判断，第一个是条件，第二个是条件为真的时候的值，第三个是条件为假的时候的值，这里面总共用了三个三目判断，哈哈。

其实上面的代码是很优秀的，条理清晰逻辑性很强，而且里面通过判断也有默认值为auto，但是为什么是鸡肋呢，原因很简单：第一，样式就两条太简单，按速度来说，用emmet写丝毫都不逊色于这个，第二，不能变通为%，组合申明样式。如我们上面的center-block虽然也很简单，但可以变通为%申明，提供两种调用方式，可以组合申明样式。其实这个@mixin还少了个单独设置width:auto或height:auto。

@mixin该休息下了，后面的%板凳都坐穿了，再不让它上场，它得拂袖而去告老师了。

## **%**

%在这里不是做单位用的，而是作为占位选择器来用的，什么是占位选择器呢，打个比方说，它就是占着茅坑不拉屎，但是如果你要拉屎它马上就给你让位。这么恶心的我们就不说了。简单来说，就是先定义好一段样式，但是这个样式默认是不会解析出来的，等到你通过@extend调用了才会解析在css文件中，避免了生成冗余浪费的css代码。

接上面的那个@mixin center-block，因为它没有参数，而我们又想要组合申明，那么该%登场了：

%center-block {

@include center-block;

}

下面我们再来给#header,.gallery调用下：

*#header{*

width:1000px;

@extend %center-block;

}

.gallery{

width:600px;

@extend %center-block;

}

解析成的css：

#header, .gallery {

margin-left: auto;

margin-right: auto;}#header {

width: 1000px;}.gallery {

width: 600px;}

实现了我们刚才说的组合申明的需求，完美！说到这里请把目光都从@mixin转到%来，我们要趁热打铁，继续深入%，其实刚才我们是在@mixin center-block的基础上定义了一个%center-block，并在里面调用@extend center-block，那样其实我们暴露了两种方式供选择调用，你要组合申明就使用@extend %center-block，你想单独拷贝进你的选择器就用@include center-block。当然你也可以如下这样直接定义%，而不是使用@include来调用一个已经定义好的@mixin，代码如下：

$lte7:true;

%clearfix {

@if $lte7 {

\*zoom: 1;

}

&:before,

&:after {

content: "";

display: table;

font: 0/0 a;

}

&:after {

clear: both;

}

}

clearfix大家熟悉吧，以前每次使用都要在我们的html结构上加个class.clearfix来调用，现在好了，如果哪个要调用这个，直接@extend：

.wrap{

@extend %clearfix;

}

.row{

@extend %clearfix;

}

解析成的css：

.wrap, .row {

\*zoom: 1;}.wrap:before, .row:before, .wrap:after, .row:after {

content: "";

display: table;

font: 0/0 a;}.wrap:after, .row:after {

clear: both;}

到这里，可能有人就会说你怎么不说下用class定义的样式，也可以用@extend来调用啊。下面说下这两个的区别，上代码：

.fl{

float:left;}

%fr{

float:right;}

这段代码我们去解析下，得到的是如下的css：

.fl {

float: left;}

我们可以看到，class方式申明的.fl本来就是css，肯定输出出来了，而%fr一点样式都没有输出。

现在我们再用@extend来分别调用下：

*#main{*

@extend .fl;

width:700px;

}*#aside\_second{*

@extend %fr;

width:300px;

}

解析成的css：

.fl, #main {

float: left;}#aside\_second {

float: right;}#main {

width: 700px;}#aside\_second {

width: 300px;}

到这里我们发现%定义的比用class定义的确实要胜一筹啊，做到了只有调用才产生样式，不调用就是个小乖乖，完全不添麻烦，不显摆。当然也许还有人会说，那个.fl我可以对应html结构里面的fl这个class。如果你这么说的话，我再给你来段解释：其实写sass的时候，我们需要一些基础的文件，这些基础文件包括的范围可能比较多，不一定每次都能把所有的用上，但是得有这么一个功能在哪里，我需要就直接调用，而不是每次都根据自己的实际需求去写个基础的东西。这其实跟我们使用jquery库的道理是一样的，也许你用的其实就是它的选择器功能，然后添加改变些class，或animate个东西，ajax都没用到，但是ajax这个功能就在哪里，你会有需要的时候。你如果用class来定义些@extend的东西，不可能每个项目都可以用到全部，那样对于这个项目来说用不到的就是多余的无用代码了，所以从现在开始，把@extend的东西都用%来定义吧。

好，现在问题又来了，那哪些我的html结构上用了.fl这个class的怎么办？这个请在你的具体项目中添加这个class，请看代码：

.fl{

@extend %fl;

}

哈哈，居然是直接在具体的项目中调用%就ok了，别气得吐血啊，虽然得到的代码一样，但是这两种是有本质区别的，代码的好坏就在这里了。

当然其实%比class的定义优势不只是在这里，而是在各种复杂环境中，class的@extend直接会解析出令人发指的疯狂代码哈哈，这也是为什么%被创造的原因之一吧。如果你想了解这个发指的情况，可以打开[理解SASS的嵌套，@extend，%Placeholders和Mixins](http://www.w3cplus.com/preprocessor/sass-basic-mixins-nesting-placeholders-extend.html" \t "http://www.w3cplus.com/preprocessor/_blank)，然后在里面搜索“强大的%placeholders”，然后上面一点就是那些糟糕的令人发指的代码了。

这里有几点需要再次提醒下：

1. 首先%定义的占位选择器样式不能传递参数。当然请注意不能传递参数，不代表样式里面不能使用变量。
2. 然后@extend调用%申明的东西时，必须要把%带上，@extend %clearfix正确，而@extend clearfix是错误的调用。
3. 最后%定义的样式，如果不调用是不会产生任何样式的，这也是用%定义样式比用原先的class方法定义样式高明的一方面。

为了表示这“揭秘”的含量，我们对第一点补充下，再搞一段代码：

*//变量依次为：字体大小，边框颜色，focus时边框颜色，圆角大小*$simpleForm: 12px $gray *#52a8ec $baseRadius !default;*

%simple-form-basic{

border: 1px solid nth($simpleForm,2);

padding: 4px;

@if not(unitless(nth($simpleForm,4))){

border-radius:nth($simpleForm,4);

}

&:focus{

outline: 0 none;

}

}

*//ie6,7 zoom组合申明*

%zoom{

@if $lte7 {

\*zoom:1;

}

}

*//通过先定义@mixin，然后在%调用@mixin来实现传递变量*

@mixin float($float:left) {

float: $float;

@if $lte7 {

display: inline;

}

}

%float-right{

@include float(right);

}

第一段代码其实是[sassCore](https://github.com/marvin1023/sassCore" \t "http://www.w3cplus.com/preprocessor/_blank)里面为form元素定义的一些border和radius，%simple-form-basic没有传递参数，但是里面的样式照样可以引用外面的变量，说明下里面有个比较有意思的判断，就是当圆角值有单位的时候才会输出border-radius，因为写0的时候我们是不带单位的，而且写0的时候，除非是覆盖以前的圆角，不然我们是不想有border-radius:0;这段代码的。第二个就简单了，对于ie6,7加个\*zoom:1;，这多好，把\*zoom:1;放在一起组合申明了。第三段代码其实就是变相的实现传递参数，主要就是先定义一个@mixin，在%里面调用@mixin。

好了，%就这么说完了，简单好用。休息下，活动下胳膊和腿，我们再来到下一个主题@function。

## **@function**

@function与@mixin，%这两者的第一点不同在于sass本身就有一些内置的函数，方便我们调用，如强大的color函数；其次就是它返回的是一个值，而不是一段css样式代码什么的。

sass本身内置函数的地址为：[sass functions](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html" \t "http://www.w3cplus.com/preprocessor/_blank)

先说第一个问题，@function返回一个值和上面的@mixin，%有什么不同呢？我们先来用内置的darken函数做个例子：

*//如果在scss里直接这样写错误的*

darken($f00,50%);

因为它返回的是一个值，而值只有放在变量或作为属性值，来段正确的：

*//作为变量值*$redDark:darken($f00,50%) !default;

*//作为属性值*

p{

color:darken($f00,70%);

}

这下大概能明白吧，作为新手其实是很容易犯上面的那个错误的。

下面我们介绍几个常用的内置函数，按照官网上面地址上的顺序来。

### rgba

分为两种：rgba($red, $green, $blue, $alpha)和rgba($color, $alpha)。

第一种跟css3一样，不介绍，第二种对我们有点用，实例：

rgba(*#102030, 0.5) => rgba(16, 32, 48, 0.5)*

rgba(blue, 0.2) => rgba(0, 0, 255, 0.2)

### ie-hex-str

ie-hex-str($color)

这个函数将一个颜色格式化成ie滤镜使用，在做css3使用滤镜兼容的时候用得上，实例：

ie-hex-str(*#abc) => #FFAABBCC*

ie-hex-str(*#3322BB) => #FF3322BB*

ie-hex-str(rgba(0, 255, 0, 0.5)) => *#8000FF00*

### darken

darken($color,$amount)

第一个参数是颜色，第二参数是百分数介于0%-100%，表示将某个颜色变暗多少个百分比。

darken(hsl(25, 100%, 80%), 30%) => hsl(25, 100%, 50%)

darken(*#800, 20%) => #200*

### lighten

lighten($color,$amount)

第一个参数是颜色，第二参数是百分数介于0%-100%，表示将某个颜色变亮多少个百分比。

lighten(hsl(0, 0%, 0%), 30%) => hsl(0, 0, 30)

lighten(*#800, 20%) => #e00*

### percentage

percentage($value)

将一个没有单位的数字转成百分比形式

percentage(0.2) => 20%

percentage(100px / 50px) => 200%

### length

length($list)

返回一个列表的长度

length(10px) => 1length(*#514721 #FFF6BF #FFD324) => 3*

### nth

nth($list, $n);

返回列表里面第n个位置的值

nth(10px 20px 30px, 1) => 10px

nth((Helvetica, Arial, sans-serif), 3) => sans-serif

### unit

unit($number)

得到这个数的单位

unit(100) => ""

unit(100px) => "px"

unit(3em) => "em"

unit(10px \* 5em) => "em\*px"

unit(10px \* 5em / 30cm / 1rem) => "em\*px/cm\*rem"

### unitless

unitless($number)

返回这个数是否没有单位

unitless(100) => true

unitless(100px) => false

### 三目判断

if($condition, $if-true, $if-false)

第一个表示条件，第二个表示条件为真的值，第三个表示为假的值

if(true, 1px, 2px) => 1pxif(false, 1px, 2px) => 2px

可能上面的都提不起你的兴趣，一时记不住也没有关系，大概有个印象，用的时候知道sass可以提供这个函数功能，或者不清楚再去查下它的官方函数。再次提醒刚开始sass的话，注意函数是返回一个值，不能直接放到sass里面直接去运行的，会报错。你可以把这些用在判断或者属性值里面，那么就是一级棒。

下面我们来搞点自己定义的函数吧，先来个简单的：

*// px转em*

@function pxToEm($px, $base: 16) {

@return ($px / $base) \* 1em;

}

调用下：

p{

font-size:pxToEm(20);}

解析后的css：

p {

font-size: 1.25em;}

估计这点小罗罗，是上不了大场面的，@mixin都有一个那么神来之笔，@function怎能没有呢，下面介绍网格布局的一个计算宽度的神来之笔，来自[blankwork](https://github.com/justnorris/Blankwork-Flexible-SASS-Grid-System" \t "http://www.w3cplus.com/preprocessor/_blank)。

*//960网格布局*$\_columns: 12 !default; *// 总列数*$\_column-width: 60px !default; *// 单列宽*$\_gutter: 20px !default; *// 间隔*

@function get\_width($columns:$\_columns, $onlyInnerWidth: true, $\_subtract:0){

*// 默认返回值*

$return: ($\_column-width + $\_gutter) \* $columns - $\_subtract !default;

@if $onlyInnerWidth == true{

*//如果$onlyInnerWidth为true，那么减掉一个间隔$\_gutter*

$return: ($\_column-width + $\_gutter) \* $columns - $\_gutter - $\_subtract;

}

@return $return;

}

首先，你得对960的网格系统比较熟悉，不然你可能有点迷惑。如果你不了解960网格系统，建议你先在w3cplus里面找找。当然也许你还没有感受它的神来之笔，别急，先来调用下：

*#container{*

width:get\_width(12,false);//960px

}

.col-four{

@extend %clearfix;

.col{

@include float;

margin:0 $\_gutter / 2;

width:get\_width(3); //220px

h2{

padding-left:10px

width:get\_width(3,true,10px); //210px

}

}

}

看到没，向那些.span1,.span2,..., .span12说88，想要几个格子就传递数字几，当然还有些特别需求，要不了刚好的网格，比喻比4个格子要少30px，看到上面的get\_width的第三个参数不，专门负责搞定这些额外需求的，get\_width(4,true,30px)得到的就是270px。

所以说这个计算宽度有三种调用形式：第一种默认的如get\_width(3)得到220px；第二种不需要左右margin的如get\_width(3，false)得到240px；第三种可以灵活缩小点宽度如get\_width(3,true,10px)得到210px。

当然计算宽度只是第一步，还有设置column的浮动，然后wrap的闭合子元素的浮动等，都是基于这个函数，一个网格系统就在这个基础上构建成功，它的这个功能应该可以和css3的那个prefixer的@mixin媲美吧。

最后说一下，其实到最后所有的的@mixin，%，@function都是为解析后的css样式服务的，如果你不能带来以下这些优势那么可以考虑不要定义这个东西：

* 化繁为简；
* 样式可以组合申明；
* 浏览器兼容样式判断；
* 灵活实现一些基础常用功能的属性值的改变，如颜色，大小等；

sass揭秘的第二篇文章就到此为止。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/" \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。