2017-12-7 11:14:56

sass学习

为CSS引入变量，可以把反复使用的CSS属性值定义为变量，也就是说，类似于bootstrap的用法？只是说bootstrap是将他们定义为class，而CSS则是自定义属性+自定义语句块

sass使用$作为标识变量的符号例如$heightlight-color：#808

任何可以用作CSS使用的赋值都可以用作sass的变量值，甚至可以一次过赋值多个属性，例如$my-border:2px solid #808,赋值的时候，这个属性值并没有生效，除非**引用了这个值**

SASS可以定义在CSS规则块内，也可以在外面。在外面则是其他CSS规则块都可以使用，如果定义在规则块内，则只有这个规则块可以使用

按照教程，我们使用koala作为编写sass的软件吧

**变量引用**

凡是CSS属性可以存在的地方，就可以用被声明的sass变量代替，例如

$my-sass:bolder;

.selector{

font-weight:$my-sass;----------->相当于font-weight:bolder;（当然，需要编译）

}

变量命名可以是中划线或者下划线，一般来说，中划线比较普遍

**嵌套CSS规则**

似乎可以像Java一样设定一个命名空间。

一般来说，如果这样的div构造

<div class=”my-div”>

<p></p>

<span>

<div class=”another-div”>

</div>

</span>

</div>

它选择器的书写会有很多重复的地方，例如

.my-div>p{}

.my-div>span{}等等，前面的父选择器是重复的。在sass中，似乎给了一个命名空间，可以嵌套使用，上面的应该可以这样写

.my-div{

p{}

span{

.another-div{}

}

}

避免了重复书写。需要特别注意，有一些情况下这种嵌套会失效，例如使用:hover等伪类选择器的时候，这时候需要使用sass提供的特殊结构&

**父选择器的标识符&**

碰到伪类选择器等等情况，嵌套的语句可能失效，例如

article a {

color:blue;

:hover {color:red}

}

这和

article a{color:blue}

article a :hover{color:red}

效果不一致，需要写成特别的&标识符,$标识符在编译时会被父类选择器直接替代。

article a{

color:blue;

&:hover {color:red}

}

**群组选择器的嵌套**

例如h1,button,div{color:red}这样的情况，对于h1,button,div都可以生效，当他们有一个共同的父选择器的时候，CSS书写会变成这样.content h1,.content button,.content div{}这个也是重复书写了。

在sass中，可以写成这样.content{h1,button,div{color:red}};

如果是这样nav,aside{a{color:red}};编译之后会变成

nav a{color:red};

aside a{color:red};

也是可以节省书写的时间，尤其是层数较多的时候。

**子组合选择器和同层组合选择器: > + 和~**

对于CSS中子选择器的情况，例如.selector>a{color:red}（表示只有.selector 后面紧跟的a子元素会生效）

sass中+表示选择元素后面紧跟的元素，例如header + p{}则是选择header后面紧跟的p元素。

同层全体组合选择器~，例如p ~ p 表示选择跟在p后面的所有p元素，不管他们之间隔了多少其他元素

使用的时候可以配合嵌套。

**嵌套属性**

在书写border background等等属性的时候，往往需要写很多条语句来逐一规定样式，sass可以嵌套来省略重复书写，并且易于阅读。例如

nav{

border:{

style:solid;

width:1px;

color:#ccc;

}

}

结果相当于

nav {

border-style: solid;

border-width: 1px;

border-color: #ccc; }

**导入SASS文件**

CSS中，可以在文件中通过@import导入其他CSS文件，sass也是可以的。

不同的是，CSS导入的时候是只有执行到@import的时候，浏览器才回去下载相应的CSS文件，而sass则是在生成CSS的时候就将相关的CSS文件导入进来，也就是写在同一个CSS文件中。

**使用sass部分文件**

可以这样想，有时候我们需要写一些不需要编译的sass文件，相当于小零件的sass文件，它们没有必要编译为CSS，只要在需要的时候，在真正需要编译的sass文件中按需导入这些小文件即可，文件复用了，类似于Java的包和类的复用这样的思想。

那么，例如你要导入themes/my\_themes\_sky.scss这个文件，只需要在样式表中写@import ”themes/my\_themes\_sky”即可。

**默认变量值**

类似于JavaScript，反复声明一个变量，最有一处声明的值会覆盖前面的值，尤其是在@import的情况下。

这时候，使用!default可以稍微缓解情况。!default与!important相反，含义是：如果这个变量被赋值了，就用它声明的值，否则使用默认值。

$fancybox-width: 400px !default;

.fancybox {

width: $fancybox-width;

}

在上例中，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。

**嵌套导入**

sass允许@import命令写在CSS规则内。该文件只能在局部生效。

**原生的CSS导入**

导入的时候，后缀名为.css 或者相应的url地址，就可以导入原生的CSS文件

**静默注释**

sass注释语法与Java类似使用//

注释内容不显示在编译生成的CSS文件中

**混合器**

@mixin标识，很像@media，有这个标识符的一大段样式是一个整体，**相当于大型变量对象**。例如这

@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;

}

生成的CSS文件为

.notice {

background-color: green;

border: 2px solid #00aa00;

-moz-border-radius: 5px;

-webkit-border-radius: 5px;

border-radius: 5px;

}

混合器使用过度，会导致样式文件过大，所以需要按需使用。

总得来说，就是遇到很多重复的时候，使用混合器

**混合器中的CSS规则**

混合器不仅可以包含属性，**还可以包含CSS规则，包含选择器和选择器**中的属性

**给混合器传参**

既然sass有变量，有混合器(相当于大变量，相当于函数)那么，混合器里面使用变量也是明显可以的，然后通过传入不同的参数，就可以生成不同的CSS，例如

@mixin my-colors($unread,$reading,$after\_read){

color:$unread;

&:hover{color:$reading;};

$:visited{color:$after\_read;}

}

可以这样引用混合器

a{@include my-colors(blue,red,green);} -->a{color:blue} a:hover{color:red;} a:after\_read{color:green}

引用的时候如果不记得参数顺序，可以使用类似JavaScript对象的写法或者是Json的写法

a{

@include my-colors(

$unread:red,

$hover:blue,

$after\_read:green

);

}

**默认参数值**

为了在@include混合器时不必传入所有参数，可以有默认的参数值，$name:default-value的声明形式

**使用选择器继承来精简CSS**

最后一个减少重复的sass特性是选择器继承，基于Nicole Sulliva面向对象的CSS理念。通过@extend语法实现。例如

.animal{

color:red;

font-size:3vw;

}

.cat{

@extend .animal;

background-color:blue;

}

这种继承是继承.animal的所有样式甚至是组合选择器样式。

**何时使用继承**

和Java等面向对象编程语言一样，适合使用继承的时候就使用。

**继承的高级用法**

除了对类使用继承之外，还有一种常用的高级用法是继承一个html元素样式。

假如一条样式规则继承了一个复杂的选择器，那么它只会继承这个复杂选择器命中的元素所应用的样式。举例来说， 如果.seriousError@extend.important.error ， 那么.important.error 和h1.important.error 的样式都会被.seriousError继承， 但是.important或者.error下的样式则不会被继承。

**继承的工作细节**

关于@extend有两个要点你应该知道。

* 跟混合器相比，继承生成的css代码相对更少。因为**继承仅仅是重复选择器**，而不会重复属性，所以使用继承往往比混合器生成的css体积更小。如果你非常关心你站点的速度，请牢记这一点。

继承遵从css层叠的规则。当两个不同的css规则应用到同一个html元素上时，并且这两个不同的css规则对同一属性的修饰存在不同的值，css层叠规则会决定应用哪个样式。相当直观：通常权重更高的选择器胜出，如果权重相同，定义在后边的规则胜出。

**另外的补充**

1. 如果变量要镶嵌在字符串之中，则必须写在#{}之中，而不是${}，例如$side:left .round{border-#{side}:1px solid #999}
2. 可以在代码中使用算式，例如body{margin:(20px/3);}
3. @mixin最强大之处在于可以传入参数，但是请记住参数顺序或者，使用对象式写法
4. 颜色函数？？？？用于生成系列颜色，例如lighten(#cc3,10%),其中10%可以用于改变，生成系列颜色
5. 可以是用条件语句，例如p{

@if 1+1 == 2 {border:1px solid #222;}

@if 5<3 {border:2px dotted #aaa;}

}

还可以嵌套@else语句

1. 支持for循环，while循环 ，还有each(类似于JavaScript的for each)例如

@for $i from 1 to 10{

.border-#{$i}{

border: #{$i}px solid red;

}

}

$i:6;

@while $i>0{

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

$i: $i-2;

}

@each $member in a,b,c,d,e{

.#{$member}{

background-image:url(“/image/#{$member}.jpg”);

}

}

1. 支持自定义函数

@function double($n){

@return $n\*2;

}

#sidebar{

width:double(5px); ----->结果应该是width:10px;

}