

28、sass

简介

css编写时权重难以控制，嵌套层级多导致代码编写不方便。sass的出现就是为了解决css的缺点。

sass是世界上最好的css扩展语言。在正常的编写css的基础上，增加了很多变量，函数，循环，判断等类似编程语言的代码模式。极大的提高了编写样式的效率。还有许多的内置函数（如颜色加深，颜色淡化）用于高效的属性设置。使得css也可以用类似脚本的方式进行编写。

sass不能直接被浏览器识别，所以需要进行编译成正常的css文件才能被浏览器使用。

优点

- 1、代码量少
- 2、解决css权重问题
- 3、嵌套更合理
- 4、使用编程语言书写CSS样式更方便维护

sass的编译

html只认识css文件，不认识sass文件的
我们使用node进行编译sass

使用node进行编译sass的时候，首先要安装sass包。为了使用方便，**直接在全局安装**：

```
npm i -g sass
```

使用sass包来编译文件：

```
sass 待编译的sass文件路径 编译后的文件路径
```

我们每一次修改sass文件就需要重新编译一次，所以，node中的sass提供了一种实时编译，每次修改文件，**自动进行编译**：

```
sass --watch 待编译的sass文件路径:编译后的文件路径
```

这样我们就和平常写css一样了，写好后随时可以在浏览器查看，不用每次都进行编译了

如果写的文件比较多的话，每次写一个文件还需要重新实时编译一个文件，也不是特别方便，所以还有一种操作，**直接实时监控文件夹**：

```
$ sass --watch 待编译的sass文件夹:编译后的文件夹
```

sass的学习

sass的文件夹的后缀分两种：`.sass` 和 `.scss` 相对来说 `.scss` 用的比较多
`.scss` 的文件写法：

```
#wrap{  
  width:100px;  
}
```

`.sass` 的文件写法：

```
#wrap  
  width:100px
```

他们的区别在于有没有大括号和分号。我们已经习惯了写大括号和分号了，所以使用 `.scss` 的比较多。

sass的注释

// 单行注释，但是在编译的时候不保留

```
/*
```

多行注释

编译的时候可以保留

压缩的时候不保留

```
*/
```

```
/*!
```

多行注释

编译和压缩的时候都会保留

```
*/
```

变量

在sass中 `$` 来定义变量：

```
$color:red;  
$font_size:12px;  
.header{  
  background-color: $color;  
  font-size:$font_size*2;  
}
```

一般用来定义颜色或者一些常用的像素值

嵌套

嵌套包括：

选择器嵌套

嵌套结构中的自己：&

属性嵌套:必须是能嵌套的属性，比如border background

```
/*后代关系*/
.wrap{
  div{
    width:$font_size*10;
  }
}
/*子类关系*/
ul{
  >li{
    padding:12px;
  }
}
/*大括号中表示自己*/
.nav{
  &:hover{
    background-color: $color;
  }
  li{
    &:hover{
      color:$color;
    }
  }
}
/*群组嵌套按正常写即可*/
/* 属性嵌套 */
.content{
  border: {
    style:solid;
    color:$color;
    width:2px;
  }
}
.left{
  border:1px solid #000{
    left:none;
    bottom:{
      width:3px;
    }
  };
}
```

编译结果：

```
/*后代关系*/
.wrap div {
    width: 120px;
}

/*子类关系*/
ul > li {
    padding: 12px;
}

/*大括号中表示自己*/
.nav:hover {
    background-color: red;
}
.nav li:hover {
    color: red;
}

/*群组嵌套按正常写即可*/
/* 属性嵌套 */
.content {
    border-style: solid;
    border-color: red;
    border-width: 2px;
}

.left {
    border: 1px solid #000;
    border-left: none;
    border-bottom-width: 3px;
}
```

混合器

混合器可以理解为一个函数

定义：

```
@mixin 混合器名称{
```

```
@mixin 混合器名称(形参...){}
```

使用：

```
@include 混合器名称
```

```
@include 混合器名称(实参)——实参大多数不加引号
```

```
/* 定义混合器 */
@mixin bor{
    border:1px solid #000;
}
/* 使用混合器 */
```

```

.box{
  @include bor;
}
/* 带参数的混合器 */
@mixin bac($path,$color,$repeat){
  background:url($path) $color $repeat;
}
/* 使用混合器 */
.box1{
  @include bac("img/1.jpg",red,no-repeat);
}
/* 带有默认值的参数 */
//带有默认值的参数一定要放在所有参数的后面
@mixin bac($path:"img/1.jpg",$color:blue,$repeat:no-repeat){
  background:url($path) $color $repeat;
}
/* 使用混合器 */
.box2{
  @include bac("img/2.jpg",green);
}

```

编译结果：

```

/* 定义混合器 */
/* 使用混合器 */
.box {
  border: 1px solid #000;
}

/* 带参数的混合器 */
/* 使用混合器 */
.box1 {
  background: url("img/1.jpg") red no-repeat;
}

/* 带有默认值的参数 */
/* 使用混合器 */
.box2 {
  background: url("img/2.jpg") green no-repeat;
}

```

继承

使用 `@extend` 将相同的样式提取出来继承使用

当下面的选择器中需要使用到上面选择器的样式，就可以使用继承将上面拿下来使用，而省掉再写：

```

/* 继承 */

```

```
.box1{
  width: 100px;
  height: 100px;
}
.box2{
  @extend .box1;
  border:1px solid #000;
}
```

编译结果：

```
/* 继承 */
.box1, .box2 {
  width: 100px;
  height: 100px;
}

.box2 {
  border: 1px solid #000;
}
```

导入（将文件分类）

在一个文件中定义变量和混合器，在写css的时候文件比较混乱，所以通常会将变量和混合器放在单独的文件中，通过命令导入进来，这样每个文件中的代码都是同一类

`@import` 被引入的文件路径：只要导入进来后，别的文件定义的变量和混合器都能在当前文件使用

变量文件

```
$orange:orange;
$red:red;
```

混合器文件：

```
@mixin bor($style,$width,$color){
  border:$style $width $color;
}
@mixin bac($path,$color,$repeat){
  background:url($path) $color $repeat;
}
```

样式文件：

```
@import "./variable.scss";
@import "./mixin.scss";
```

```
.box{  
  @include bor(solid,1px,$red);  
}
```

编译后的css：

```
.box {  
  border: solid 1px red;  
}
```