SASS

- SASS官网

- **世界上最成熟、最稳定、最强大的专业级CSS扩展语言！**

- `sass` 是一个 css 的预编译工具
- 也就是能够 **更优雅** 的书写 css
- `sass` 写出来的东西 **浏览器不认识**
- 依旧是要转换成 css 在浏览器中运行
- 这个时候就需要一个工具来帮我们做

安装 sass 环境

- 以前的 `sass` 需要依赖一个 `ruby` 的环境
- 现在的 `sass` 需要依赖一个 `python` 的环境
- 但是我们的 node 强大了以后，我们只需要依赖 `node` 环境也可以
- 需要我们使用 npm 安装一个全局的 `sass` 环境就可以了

```
# 安装全局 sass 环境
$ npm install sass -g
```

编译 sass

- 有了全局的 `sass` 环境以后
- 我们就可以对 `sass` 的文件进行编译了
- `sass` 的文件后缀有两种，一种是 `.sass` 一种是 `.scss`
- 他们两个的区别就是有没有 `{}` 和 `;`
- `.scss` 文件

```
h1 {
 width: 100px;
 height: 200px;
}
```

- `.sass` 文件

```
h1
 width: 100px
 height: 200px
```

- 我们比较常用的还是 `.scss` 文件
- 因为 `.sass` 我们写不习惯，当然，如果你能写习惯也比较好用
- 我们先不管里面的的什么内容，至少这个 `.scss` 或者 `.sass` 文件浏览器就不认识
- 我们就要用指令把 这两种 文件变成 css 文件

```
# 把 index.scss 编译，输出成 index.css
$ sass index.scss index.css
```

- 这样我们就能得到一个 css 文件，在页面里面也是引入一个 css 文件就可以了

实时编译

- 我们刚才的编译方式只能编译一次
- 当你修改了文件以后要从新执行一遍指令才可以
- 实时编译就是随着你文件的修改，自动从新编译成 css 文件
- 也是使用指令来完成

```
# 实时监控 index.scss 文件，只要发生修改就自动编译，并放在 index.css 文件里面
$ sass --watch index.scss:index.css
```

- 然后你只要修改 `index.scss` 文件的内容， `index.css` 文件中的内容会自动更新

实时监控目录

- 之前的实时监控只能监控一个文件
- 但是我们有可能要写很多的文件

- 所以我们要准备一个文件夹，里面放的全部都是 sass 文件
- 实时的把里面的每一个文件都编译到 css 文件夹里面
- 依旧是使用指令的形式来完成

```
# 实时监控 sass 这个目录，只要有变化，就会实时响应在 css 文件夹下
$ sass --watch sass:css
```

- 这样，只要你修改 sass 文件夹下的内容，就会实时的相应在 css 文件夹中
- 你新添加一个文件也会实时响应
- 但是你删除一个文件，css 文件夹中不会自动删除，需要我们自己手动删除

sass 语法

- 我们能编译 `sass` 文件了，接下来我们就该学习一下 `sass` 的语法了
- 为什么他这么强大，这么好用，都是靠强大的语法
- `.sass` 和 `.scss` 文件的语法是一样的，只不过区别就是 `{}` 和 `;`

变量

- 定义一个变量，在后面的代码中使用
- 使用 `$` 来定义变量

```
// 定义一个 $c 作为变量，值是 红色
$c: red;

h1 {
 // 在使用 $c 这个变量
 color: $c;
}
```

- 上面定义的变量全局都可以使用
- 我们也可以在规则块内定义私有变量

```
h1 {
 // 这个 $w 变量只能在 h1 这个规则块中使用
 $w: 100px;
 width: $w;
}
```

```
}
```

嵌套

- `sass` 里面我们最长用到的就是嵌套了
- 而且相当的好用

```
h1 {
 width: 100px;

 div {
 width: 200px;
 }
}

// 编译结果
h1 {
 width: 100px;
}

h1 div {
 width: 200px;
}
```

- 这个就是嵌套，理论上可以无限嵌套下去

```
ul {
 width: 100px;

 li {
 width: 90px;

 div {
 width: 80px;

 p {
 width: 70px;

 span: {
 color: red;
 }
```

```
  }
  }
  }
 }
```

嵌套中的 &

- 在嵌套中还有一个标识符是 `&` 我们可以使用
- 先来看一个例子

```
div {
 width: 100px;
 height: 100px;

  :hover {
 width: 200px;
  }
}

// 我想的是 div 被鼠标悬停的时候 width 变成 200
// 但是编译结果却是
div {
 width: 100px;
 height: 100px;
}
div :hover {
 width: 200px;
}
```

- 和预想的结果不一样了
- 这个时候就要用到 `&` 来连接了

```
div {
 width: 100px;
 height: 100px;

  &:hover {
 width: 200px;
  }
}

```

```
// 编译结果
div {
 width: 100px;
 height: 100px;
}
div:hover {
 width: 200px;
}
```

- 这个时候就和我需要的一样了

群组嵌套

- 群组嵌套就是多个标签同时嵌套

```
div {
 width: 100px;

 .box1, .box2, .box3 {
  color: red;
 }
}

// 编译结果
div {
 width: 100px;
}
div .box1, div .box2, div .box3 {
 color: red;
}
```

- 还有一种就是多个标签同时嵌套一个标签

```
h1, h2, h3 {
 width: 100px;

 .box {
  color: red;
 }
}

// 编译结果
```

```
h1, h2, h3 {
  width: 100px;
}
h1 .box, h2 .box, h3 .box {
  color: red;
}
```

嵌套属性

- 在 `scss` 里面还有一种特殊的嵌套
- 叫做 **属性嵌套**
- 和选择器嵌套不一样，是写属性的时候使用的

```
div {
  border: {
  style: solid;
  width: 10px;
  color: pink;
  }
}

// 编译结果
div {
  border-style: solid;
  border-width: 10px;
  border-color: pink;
}
```

- 这个属性嵌套还可以有一些特殊使用

```
div {
  border: 1px solid #333 {
  bottom: none;
  }
}

// 编译结果
div {
  border: 1px solid #333;
  border-bottom: none;
}
```

```
}
```

混入

- 也叫 **混合器**
- 其实就是定义一个 “函数” 在 `scss` 文件中使用

```
// 定义一个混合器使用 @mixin 关键字
@mixin radius {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -ms-border-radius: 10px;
  -o-border-radius: 10px;
  border-radius: 10px;
}
```

- 上面是定义好的一个混合器
- 他是不会被编译的，只有当你使用了他以后，才会被编译

```
// 使用混合器使用 @include 关键字
div {
  width: 100px;
  height: 100px;

  @include radius;
}
```

- 这个就是吧刚才定义的混合器拿过来使用
- 编译结果

```
div {
  width: 100px;
  height: 100px;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -ms-border-radius: 10px;
  -o-border-radius: 10px;
  border-radius: 10px;
}
```

混合器传参

混合器传参

- 我们既然说了，混合器就像一个 “函数” 一样，那么就一定可以像 “函数” 一样传递参数
- 和 “函数” 的使用方式一样，在定时的时候是 “形参”，在调用的时候是 “实参”

```
// 定义混合器
@mixin my_transition($pro, $dur, $delay, $tim) {
  -webkit-transition: $pro $dur $delay $tim;
  -moz-transition: $pro $dur $delay $tim;
  -ms-transition: $pro $dur $delay $tim;
  -o-transition: $pro $dur $delay $tim;
  transition: $pro $dur $delay $tim;
}
```

- 使用这个混合器的时候传递 “实参”

```
div {
  width: 100px;
  height: 100px;

  @include my_transition(all, 1s, 0s, linear);
}
```

- 编译结果

```
div {
  width: 100px;
  height: 100px;
  -webkit-transition: all 1s 0s linear;
  -moz-transition: all 1s 0s linear;
  -ms-transition: all 1s 0s linear;
  -o-transition: all 1s 0s linear;
  transition: all 1s 0s linear;
}
```

- 写了多少个 “形参”，那么调用的时候就要传递多少个 “实参”
- 不然会报错的

参数默认值

- 我们在定义混合器的时候，也可以给一些参数写一些默认值

- 这样一来，就可以不传递 “实参” 了

```
// 设置一些带有默认值的参数
@mixin my_transition($dur: 1s, $pro: all, $delay: 0s, $tim: linear) {
 -webkit-transition: $dur $pro $delay $tim;
 -moz-transition: $dur $pro $delay $tim;
 -ms-transition: $dur $pro $delay $tim;
 -o-transition: $dur $pro $delay $tim;
 transition: $dur $pro $delay $tim;
}
```

- 使用的时候，如果你不传递，那么就是使用默认值

```
div {
 width: 100px;
 height: 100px;

 // 使用的时候，只传递一个，剩下的使用默认值
 @include my_transition(2s);
}
```

- 编译结果

```
div {
 width: 100px;
 height: 100px;
 -webkit-transition: 2s all 0s linear;
 -moz-transition: 2s all 0s linear;
 -ms-transition: 2s all 0s linear;
 -o-transition: 2s all 0s linear;
 transition: 2s all 0s linear;
}
```

继承

- 在 `sass` 里面使用继承可以大大的提高开发效率
- 其实继承很简单，就是把之前写过的选择器里面的内容直接拿过来一份

```
div {
 width: 100px;
 height: 100px;
 background-color: pink;
```

```
}
```

- 这个是之前写过的一个规则样式表
- 接下来我要写另外一个样式了，发现我要写的一些内容和之前这个 div 一样，并且还有一些我自己的内容
- 那么我就可以把这个样式表先继承下来，再写我自己的内容就好了

```
p {
 @extend div;

 font-size: 20px;
 color: red;
}
```

- 编译结果

```
div, p {
 width: 100px;
 height: 100px;
 background-color: pink;
}

p {
 font-size: 20px;
 color: red;
}
```

注释

- 在 `scss` 文件中的注释分为几种

i. 编译的时候不会被编译的注释

```
// 我是一个普通注释，在编译的时候，我就被过滤了
```

ii. 编译的时候会被编译的注释

```
/* 我在编译的时候，会被一起编译过去 */
```

iii. 强力注释

```
/*! 我是一个强力注释，不光编译的时候会被编译过去，将来压缩文件的时候也会存在 */
```

导入文件

- 我们刚才学过了定义变量，定义混合器
- 而这两个内容在定义过以后，如果没有使用，是不会被编译出内容的
- 所以我们可以把变量单独写一个文件，混合器单独写一个文件，然后直接导入后使用

```
// 我是 variable.scss
$w: 100px;
$h: 200px;
$c: pink;

// 我是 mixin.scss
@mixin my_transition($dur: 1s, $pro: all, $delay: 0s, $tim: linear) {
 -webkit-transition: $dur $pro $delay $tim;
 -moz-transition: $dur $pro $delay $tim;
 -ms-transition: $dur $pro $delay $tim;
 -o-transition: $dur $pro $delay $tim;
 transition: $dur $pro $delay $tim;
}

@mixin radius {
 -webkit-border-radius: 10px;
 -moz-border-radius: 10px;
 -ms-border-radius: 10px;
 -o-border-radius: 10px;
 border-radius: 10px;
}
```

- 然后在我们的主要文件中把这个两个文件导入进来就行了

```
// 我是 index.scss
@import './variable.scss';
@import './mixin.scss';

div {
 width: $w;
 height: $h;
 background-color: $c;

```

```
 @include radius;
}

h1 {
 @include my_transition;
}
```

- 编译结果

```
div {
 width: 100px;
 height: 200px;
 background-color: pink;
 -webkit-border-radius: 10px;
 -moz-border-radius: 10px;
 -ms-border-radius: 10px;
 -o-border-radius: 10px;
 border-radius: 10px;
}

h1 {
 -webkit-transition: 1s all 0s linear;
 -moz-transition: 1s all 0s linear;
 -ms-transition: 1s all 0s linear;
 -o-transition: 1s all 0s linear;
 transition: 1s all 0s linear;
}
```