SASS专题讲解

1. sass的含义

Sass (全称: Syntactically Awesome StyleSheets), Sass 是一款强化 CSS 的辅助工具，它在 CSS 语法的基础上增加了变量 (variables)、嵌套 (nested rules)、混合 (mixins)、导入 (inline imports) 等高级功能，这些拓展令 CSS 更加强大与优雅。

使用 Sass 以及 Sass 的样式库有助于更好地组织管理样式文件，以及更高效地开发项目

2. 当你直接搜索sass的时候, 会出现两个不同的名称, 一个是sass一个是scss, 这两个又是啥? 有啥关系, 有啥区别呢?

2.0 sass和scss是CSS预处理器Sass提供的两种不同的语法，两者相似并且都做同样的事情，但是以不同的风格书写。SCSS是最新的，被认为比Sass更好

2.1 sass含义

Sass 是一门高于 CSS 的元语言。Sass 能够提供更简洁、更优雅的语法，同时提供多种功能来创建可维护和管理的样式表。
Sass 是采用 Ruby 语言编写的一款 CSS 预处理语言，它诞生于2007年，是最大的成熟的 CSS 预处理语言。最初它是为了配合HAML（一种缩进式 HTML 预编译器）而设计的，因此有着和 HTML 一样的缩进式风格。SASS是CSS3的一个扩展，增加了规则嵌套、变量、混合、选择器继承等等。通过使用命令行的工具或WEB框架插件把它转换成标准的、格式良好的CSS代码。

2.2 scss含义

Scss 是 Sass 3 引入新的语法，是Sassy CSS的简写，是CSS3语法的超集，也就是说所有有效的CSS3样式也同样适合于Sass。说白了Scss就是Sass的升级版，其语法完全兼容 CSS3，并且继承了 Sass 的强大功能。也就是说，任何标准的 CSS3 样式表都是具有相同语义的有效的 SCSS 文件。另外，SCSS 还能识别大部分 CSS hacks（一些 CSS 小技巧）和特定于浏览器的语法，例如：古老的 IE filter 语法。
由于 Scss 是 CSS 的扩展，因此，所有在 CSS 中正常工作的代码也能在 Scss 中正常工作。也就是说，对于一个 Sass 用户，只需要理解 Sass 扩展部分如何工作的，就能完全理解 Scss。大部分扩展，例如变量、parent references 和 指令都是一致的；唯一不同的是，SCSS 需要使用分号和花括号而不是换行和缩进。

2.3 二者区别及使用

区别:
Sass 和 SCSS 其实是同一种东西，我们平时都称之为 Sass，两者之间不同之处有以下两点：
文件扩展名不同，Sass 是以“.sass”后缀为扩展名，而 SCSS 是以“.scss”后缀为扩展名
语法书写方式不同，Sass 是以严格的缩进式语法规则来书写，不带大括号({})和分号(;)，而 SCSS 的语法书写和我们的 CSS 语法书写方式非常类似。

编译规则:
sass编译需要通过ruby安装, 配置对应的安装环境 比较复杂, 详见: 如何安装
sass(https://www.sass.hk/install/)

但是为了提高开发效率我们再开发的时候目前使用的SCSS 通过VScode中的插件(Easy Sass)进行使用编译处理CSS

3. sass (scss) 的特点

3.1 完全的兼容CSS3

3.2 再CSS基础上增加变量 , 嵌套 , 混合等功能

3.3 通过函数进行颜色值与属性值的运算

3.4 提供控制指令等高级功能

3.5 自定义输出格式

4. 如何使用SCSS

第一步: 再站点CSS文件中创建一个以 .scss为后缀名的文件 (01.scss)

第二步: 随意编写一些基础的CSS你学过的代码,

```
.test {
  width: 100px;
  height: 100px;
  background-color: red;
}
```

第三部: 此时保存后, 我们能看到css文件中会自行创建两个独立的后缀名为CSS的文件 (01.css 和 01.min.css)

```
01.css和01.min.css 区别
两者仅仅是文件大小、有无注释、格式美化（可读性）上的区别，使用上（作用效果）并没有什么区别。
压缩和未压缩的区别，.min是压缩版的（去掉多余的注释、空格等）文件较小，易于加载，另外的就是未压缩的了文件稍大，保留完整的注释、可读性强的标准格式化文件。
一般生产环境用.min.css（加载速度快，提升体验） ,开发试验阶段用.css（便于理解）
```

5.CSS拓展功能

5.1 使用特性

CSS书写代码规模较大的Web应用时，容易造成选择器、层叠的复杂度过高，因此推荐通过SASS预处理器进行CSS的开发，SASS提供的**变量**、**嵌套**、**混合**、**继承**等特性，让CSS的书写更加有趣与程式化。

5.2 Scss中的注释

Scss中的注释, 支持标准的 CSS 多行注释 `/* */`，以及单行注释 `//`，前者会 被完整输出到编译后的 CSS 文件中

```
.test {
    //这是基础代码, 单行住址
    /*这是多行注释*/
    width: 100px;
    height: 100px;
    background-color: red;
}
```

5.3 Scss中的嵌套语法

Sass 允许将一套 CSS 样式嵌套进另一套样式中，内层的样式将它外层的选择器作为父选择器，其中我们可以使用各式各样的选择器: 例如：

```
Scss语法(注意:如果没有大于号的话则使用的是后代选择器)
#main {
    width: 200px;
    height: 200px;
    background-color: orange;
    >p {
        width: 50px;
        height: 50px;
        background-color: purple;
    }
}
```

```
CSS编译语法
#main {
  width: 200px;
  height: 200px;
  background-color: orange;
}

#main > p {
  width: 50px;
  height: 50px;
  background-color: purple;
}
```

```
HTML标签内容使用的时候
<div id="main">
    <p></p>
</div>
```

嵌套功能避免了重复输入父选择器，而且令复杂的 CSS 结构更易于管理：

```
Scss代码部分，解决共用一个父元素的烦恼
#bobo {
    width: 400px;
    height: 400px;
    background-color: yellow;

    div,
    .box {
        width: 200px;
        height: 100px;
        background-color: green;
    }
}
```

```
编译后的CSS代码
#bobo {
  width: 400px;
  height: 400px;
  background-color: yellow;
}

#bobo div,
#bobo .box {
  width: 200px;
  height: 100px;
  background-color: green;
}
```

5.4 Scss中父选择器 `&`

在嵌套 CSS 规则时，有时也需要直接使用嵌套外层的父选择器，例如，当给某个元素设定 `hover` 样式时，可以用 `&` 代表嵌套规则外层的父选择器。

```
SCSS语法/* 父选择器 */
.contain {
    width: 200px;
    height: 200px;
    background-color: blue;
    p {
        width: 100px;
        height: 100px;
        background-color: yellow;
    }
    &:hover>p {
        width: 100px;
        height: 100px;
        background-color: aqua;
    }
}
```

```
CSS编译后的语法/* 父选择器 */
.contain {
  width: 200px;
  height: 200px;
  background-color: blue;
}

.contain p {
  width: 100px;
  height: 100px;
  background-color: yellow;
}

.contain:hover > p {
  width: 100px;
  height: 100px;
  background-color: aqua;
```

```
}
```

编译后的 CSS 文件中 `&` 将被替换成嵌套外层的父选择器，如果含有多层嵌套，最外层的父选择器会一层一层向下传递：

```
/* 传递 */ /* 父选择器 */
.contain {
    width: 200px;
    height: 200px;
    background-color: blue;

    p {
        width: 100px;
        height: 100px;
        background-color: yellow;

        &>a {
            font-weight: bold;
        }
    }
}
```

```
CSS编译结果
/* 传递 */
/* 父选择器 */
.contain {
  width: 200px;
  height: 200px;
  background-color: blue;
}

.contain p {
  width: 100px;
  height: 100px;
  background-color: yellow;
}

.contain p > a {
  font-weight: bold;
}
```

`&` 必须作为选择器的第一个字符，其后可以跟随后缀生成复合的选择器，例如

```
#main {
  color: black;
  &-sidebar { border: 1px solid; }
}
```

```
#main {
    color: black;
}
#main-sidebar {
    border: 1px solid;
}
```

5.5 属性嵌套

有些 CSS 属性遵循相同的命名空间 (namespace)，比如 `font-family, font-size, font-weight` 都以 `font` 作为属性的命名空间。为了便于管理这样的属性，同时也为了避免了重复输入，Sass 允许将属性嵌套在命名空间中，例如：(font / list / background等等)

```
SCSS代码部分区域
.contain {
    text: {
        align: center;
        decoration: underline;
        transform: lowercase;
    }
}
```

6. SassScript

6.1 变量 :

变量以美元符号开头，赋值方法与 CSS 属性的写法一样：全局命名的变量为全局变量, 再局部 (也就是{}里面定义的变量被称之为局部变量 , 不在嵌套规则内定义的变量则可在任何地方使用（全局变量）。将局部变量转换为全局变量可以添加 `!global`)

```
SCSS编写 /* 变量声明  再全局定义的变量 */
$width_1: 250px;
$height_1: 200px;

#main {
    width: $width_1;
    height: $height_1;
    background-color: red;
}
.main {
    $width_2: 150px !global;
    $height_2: 150px;
    width: $width_2;
    height: $height_2;
    background-color: yellow;

    >p {
        width: $height_2;
        height: 100px;
        background-color: orange;
    }
}
```

```
/*前面不带 !global 此时为代码报错, 因为$width_2不是全局变量 */
.contain {
    width: $width_2;
}
```

```
@charset "UTF-8";
/* 变量声明 */
#main {
  width: 250px;
  height: 200px;
  background-color: red;
}

.main {
  width: 150px;
  height: 150px;
  background-color: yellow;
}

.main > p {
  width: 150px;
  height: 100px;
  background-color: orange;
}

/* 此时为代码报错, 因为$width_2不是全局变量*/
.contain {
  width: 150px;
}
```

6.2 继承

class 使用 @extend , 在设计网页的时候常常遇到这种情况：一个元素使用的样式与另一个元素完全相同，但又添加了额外的样式。通常会在 HTML 中给元素定义两个 class，一个通用样式，一个特殊样式。假设现在要设计一个普通错误样式与一个严重错误样式，一般会这样写：(案例1, 使用相同的代码, 案例2, 公共的版心, 多个类名使用)

```
原始代码:HTML部分
<div id="main"></div>
<p class="box"></p>
原始代码:CSS部分
#main{
    width: 300px;
    height: 300px;
    background-color: blue;
}
.box{
    width: 300px;
    height: 300px;
    background-color: yellow;
}
```

```
SCSS代码
```

```
#main {
    width: 300px;
    height: 300px;
    background-color: blue;
}
.box {
    @extend #main;
    background-color: yellow;
}
```

编译后的

```
/* 拥有相同的样式,继承前面的元素 */
#main, .box {
  width: 300px;
  height: 300px;
  background-color: blue;
}
.box {
  background-color: yellow;
}
```

公共版心案例

```
/* scCSS公共版心*/
.bancen {
    width: 1600px;
    margin: 0 auto;
}
.logo{
    @extend .bancen;
    height: 100px;
    background-color: orange
}
.nav{
    @extend .bancen;
    height: 80px;
    background-color: aqua;
}
```

编译后的代码

```
/*公共版心*/
.bancen, .logo, .nav {
  width: 1600px;
  margin: 0 auto;
}
.logo {
  height: 100px;
  background-color: orange;
}
.nav {
  height: 80px;
  background-color: aqua;
}
```

6.3 混合

混合指令（Mixin）用于定义可重复使用的样式，避免了使用无语意的 class，比如 `.float-left`。混合指令可以包含所有的 CSS 规则，绝大部分 Sass 规则，甚至通过参数功能引入变量，输出多样化的样式。(定义混合: @mixin 调用混合: @include)

一次混合多次调用; 一个选择器中可以调用多次混合

```
/* 定义混合 large-text 就是定义混合的名字; */
@mixin large-text {
  font: {
    family: Arial;
    size: 20px;
    weight: bold;
  }
  color: #ff0000;
}
/* 调用混合器 */
.page-title {
  @include large-text;
  padding: 4px;
  margin-top: 10px;
}
编译为:
.page-title {
  font-family: Arial;
  font-size: 20px;
  font-weight: bold;
  color: #ff0000;
  padding: 4px;
  margin-top: 10px;
}
```

6.4 注意事项 : 继承和混合的区别:

继承

在sass中为了进一步减少代码的重复书写，还提出了继承的概念，继承是基于选择器的，也就是说一个选择器可以继承为另一个选择器定义的所有样式。默认浏览器样式是不会被继承的，因为它们不属于样式表中的样式。

继承的原理并不是通过样式属性的复制来实现的，而是通过选择器的复制实现的，所以跟混合器相比，继承生成的css代码相对更少。

但是继承要慎用，因为继承的子选择器不仅会继承父选择器自身的所有样式，而且任何跟父选择器有关的组合选择器也都会被子选择器继承，也就是说所有父选择器出现的地方，子选择器会复制一份，然后替换成子选择器。最好不要在后代选择器中继承，这会导致选择器的个数依然可能会变得相当大。

混合:

在sass我们可以通过变量来复用小规模的样式（颜色、字体等），但如果我们想要复用一整段的样式代码，那变量就没办法了。还好sass定义了一个新特性 — 混合器，来专门实现多行代码的复用。如果你发现自己在不停地重复一段样式，那就应该把这段样式构造成混合器，尤其是这段样式本身就是一个逻辑单元，比如说是一组放在一起实现某个特定的页面样式。

6.5 编程条件语句

```
/* SCSS条件控制 */
p {
    @if (1+1==2) {
        border: 1px solid;
    }

    @if (5 < 3) {
        border: 2px dotted;
    }

    @if (null) {
        border: 3px double;
    }
}
/* 编译后的 CSS*/
p {
    border: 1px solid;
}
```

6.6 模块化引入

任何一门语言都是以模块化开发, 包括我们的CSS, CSS中有自己独有的import导入形式, 其实在SCSS中也存在对应的模块化导入形式,

两种模块化引入方式:

@import

@use

7. 案例及案例整合

1. 案例1
2. 案例2
3. 案例3