Sass基础

什么是Sass

在学习这个知识点的时候，我们先来了解一下它的概念

我们可以看到 `sass` 是一个专业级的CSS扩展语言
思考：CSS为什么需要扩展语言？

1. 在书写CSS的时候，我们经常会发现要使用变量，当然现在的CSS里面也可以定义变量
2. 在书写CSS的时候，我们也经要常地的计算四则混合运算
3. 我们在页面上面经常会发现大量相似的代码
4. CSS里面可能会有很多有规律的代码，如 `bootstrap` 经常会出现 `text-primary,text-danger,text-warning` ，还有 `bg-primary,bg-danger,bg-warning`
   这个时候其实我们就发现，CSS还有可以像其它的编程语言一样，进行一个有规律的操作。所以针对这个特点，w3c就开始面向开发者征订标准，目前就产生了一些关于CSS扩展的语言
   目前主流的CSS扩展语言有以下几种
5. `less` 是一个比较早期的CSS预处理理言，也是一个CSS的扩展语言，它的语法简单，很容易上手
6. `sass` 它可以理解为 `less` 的升级版本，它的功能也很强大
7. `stylus` 它的语法与 `sass` 有80%是一样的，功能也差不多
   为了保证同学们的学习速度的接受难度，并保证与目前主流的技术进行统一，我们在这里就选择了 `sass` 来进行学习

Sass编译环境安装

Sass这一门语言并不能直接被浏览器识别（能够被浏览器识别的语言只有3种， `html/css/js` ）,所以我们的电脑上面需要安装一个编译工具，将 `sass` 语言编译成 `css` 语言。目前的编译环境有很多，官方推荐我们使用 `Ruby` 来进行编译
因为我们目前的电脑上面已经安装了 `nodejs` 的环境，而 `nodejs` 的环境也是可以编译 `sass` 的，所以在这里，我们直接使用这个环境编译就可以了

> 在之前的时候我们使用的是编译环境叫 `node-sass` 这个包，但是这个包因为一些原因已经废弃了，不再使用了，所以我们换一个包进行编译，这个包叫 `dart-sass`，这个包在npm里面有的，可以直接使用。这个包在 `npm` 的仓库里面叫 `sass`

安装

```
$ npm install sass --save-dev
```

> 这里要注意，我们没有使用 `--save` 而是使用了 `--save-dev`，这是因为这个包只在开发环境下面才会使用
> **回顾一下：什么是开发依赖，什么是生产依赖**
> 一般情况下的理解就是我们把原材料叫 生产依赖，把工具叫开发依赖，所以这里的 `sass` 它只是一个编译工具，它的目的就是把 `sass` 编译成 `css`

```
{
  "name": "sass-demo1",
  "version": "1.0.0",
  "description": "sass的第一个demo",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "杨标",
  "license": "ISC",
  "devDependencies": {
    "sass": "^1.55.0"
  }
}
```

配置命令

```
"scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "help": "sass --help"
    },
```

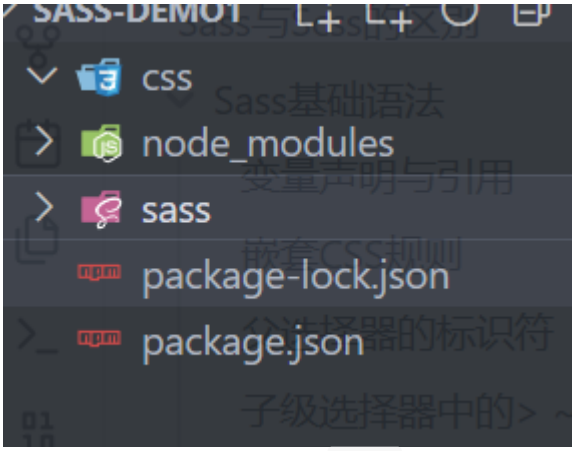
当我们在 `package.json` 配置了上面的 `help` 命令以后，我们去控制台执行 `npm run help` 就会得到下面的结果

```
Usage: sass <input.scss> [output.css]
       sass <input.scss>:<output.css> <input/>:<output/> <dir/>

=== Input and Output ====================
    --[no-]stdin               Read the stylesheet from stdin.
    --[no-]indented            Use the indented syntax for input from stdin.
-I, --load-path=<PATH>         A path to use when resolving imports.
                               May be passed multiple times.
-s, --style=<NAME>             Output style.
                               [expanded (default), compressed]
    --[no-]charset             Emit a @charset or BOM for CSS with non-ASCII
characters.
                               (defaults to on)
    --update                   Only compile out-of-date stylesheets.
```

```
=== Source Maps ========================
    --[no-]source-map          Whether to generate source maps.
                               (defaults to on)
    --source-map-urls          How to link from source maps to source files.
                               [relative (default), absolute]
    --[no-]embed-sources       Embed source file contents in source maps.
    --[no-]embed-source-map    Embed source map contents in CSS.

=== Other ==============================
-w, --watch                    Watch stylesheets and recompile when they change.
    --[no-]poll                Manually check for changes rather than using a
native watcher.
                               Only valid with --watch.
    --[no-]stop-on-error       Don't compile more files once an error is
encountered.
-i, --interactive              Run an interactive SassScript shell.
-c, --[no-]color               Whether to use terminal colors for messages.
    --[no-]unicode             Whether to use Unicode characters for messages.
-q, --[no-]quiet               Don't print warnings.
    --[no-]quiet-deps          Don't print compiler warnings from dependencies.
                               Stylesheets imported through load paths count as
dependencies.
    --[no-]verbose             Print all deprecation warnings even when they're
repetitive.
    --[no-]trace               Print full Dart stack traces for exceptions.
-h, --help                     Print this usage information.
    --version                  Print the version of Dart Sass.
```

在上面的命令里在，我们从第一行就可以的看到，它告诉我们怎么样将sass的文件编译成 `css`

配置编译命令



上面有2个文件夹，我们要将 `sass` 下面的文件编译到 `css` 目录下面去，怎么办呢?

```
  "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "help": "sass --help",
        "start":"sass -w ./sass:./css"
    },
```

> 上面的命令代表的就是监听 `sass` 这个目录，然后将这个目录下面的文件编译到 `css` 的目录

当我们在去执行 `npm run start` 的时候 就启动了sass的编译，它会监听 `sass` 这个文件的变化，同时如果想停止监听应该按下 `Ctrl-C` 两个键

Sass与Scss的区别

> Sass的四个基本点分别如下
>
> 1. 变量
> 2. 嵌套
> 3. 混合
> 4. 继承

首先我们先来看一下 `sass` 的语法

```
/*先看一下Sass的语法*/
.div1
    width: 100px
    height: 100px
```

> 上面的代码就是 `sass` 的语法，它最终会编译成下面的代码

```
@charset "UTF-8";
/*先看一下Sass的语法*/
.div1 {
  width: 100px;
  height: 100px;
}
/*# sourceMappingURL=01.css.map */
```

这么写很不方便，所以一般情况下，我们都不使用 `sass` 的语法【为什么不使用，因为它是一种严格的语法格式，用起来非常麻烦 ，有时候甚至还容易将我们的原本的CSS混淆】

`Sass` 推出来的目的就是为了能够让别人更好 的书这与CSS，所以如果它的语法变得非常难写，别人也就不会用了

为了解决这个问题， `Sass` 推出了新的语法叫 `scss`

> 你们可以理解为 `scss== sass + css`
> 这一种语法采用我们以前所学习的 `css` 的编写方式 ， 然后又将 `sass` 里面的编程思想带入进去了，所以这一种语法一推出，立即就受到了好评

我们在当前的项目下面创建一个 `scss` 的目录 ， 然后再将编译更改一下

```
"start":"sass -w ./scss:./css"
```

```
.div1{
    background-color: red;
}
```

> 这个时候我们可以看到scss更符合我们的书写标准

Sass基础语法

关于SCSS的注释

在SCSS里面的注释有2种情况，如下所示

```
/*这是CSS的注释
  如果你这么写注释，这个注释会生成到CSS文件里面
*/
//在SCSS里面，还有一种注释叫双斜线，这本来是一种JS的注释也早可以的
//这一种注释是不会编译到CSS文件里面去的
```

变量声明与引用

> 变量应该是先定义，后使用

```
$w:100px;           //定义了一个变量
.box{
    width: $w;
}
```

同时要注意，变量是有区域性的

```
.div1{
    $a:red;
}
.box2{
    //这里会报错，因为变量是有区域性的
    color: $a;
}
```

同时要注意，变量是可以重复的定义与赋值的

```
$w:100px;           //定义了一个变量
.box{
    width: $w;
}
$w:200px;
.box2{
    height: $w;
}
```

在重复定义的时候，我们要注意一个点，有时候我们看到别人在定义变量的时候会在后面添加一个`!default`

默认变量

```
$w:100px;           //定义了一个变量
$x:red;
.box{
    width: $w;
    color:$x;
}
$w:200px;
$x:blue !default;
.box2{
    height: $w;
    color:$x;
}

```

我们可以看到最终生成的结果里面，所有的 `$x` 最终都变成了 `red` ，这是因为我们在后面定义 `$x:blue !default;` 加了 `!default` ，它的作用就是定义了一个变量 `$x` ，同时赋值了一个默认值
默认值的特点就是如果你后来没有赋值了，我就使用默认值，如果你在**其它地方**又赋了，那我就使用其它的地方的值

嵌套CSS规则

在以前的时候，如果我们要写CSS的后代，子代等选择器会很麻烦，如下所示

```
//嵌套
.box{
    width: 100px;
    height: 100px;
}
.box>a{
    color: red;
}
```

现在来看一下嵌套规则的使用

```
.box{
    width: 100px;
    height: 100px;
    >a{
        color: red;
    }
    span{
        display: block;
    }
    +div{
        font-size: 32px;
    }
    ~.aaa{
        border-radius: 50%;
        .bbb{
            border: 2px solid black;
        }
    }
}
```

父选择器的标识符 `&`

```
.box{
    &:hover{
        background-color:red;
    }
}
.div1 {
    width: 200px;
    &.aaa{
        width: 300px;
    }
    &+&{
        width: 400px;
    }
}
```

嵌套规则当中的变量

在嵌套规则里面，我们的变量只是外部定义，内部使用，内部定义的东西在外部是用不了的

```
.div1 {
    $x: 100px;
    .abc {
        width: $x;
        $y:200px;
        .def{
            height: $y;
        }
    }
    .box{
        //报错
       width: $y;
    }
}
```

在使用变量的时候，我们经常会遇到下面的问题

```
.box {
    width: 100px;
    .aaa {
        height: 200px;
        $r: red;
    }
}
.bbb {
    color: $r;
}
```

> 我们既希望生成的 `bbb` 是一个全局的选择器，又想要使用 `$r` 这个变量，这样就出现了冲突，因为 `$r` 是一个局部变量，它只能在内部使用，但是如果我们要是把选择器写在了 `aaa` 的下面， 这个时候生成的选器变成了 `.box .aaa .bbb` ，这就不符合我们的要求

为了解决这个问题， `scss` 出了一个新的命令叫 `@at-root` 来完成

```scss
.box {
    width: 100px;
    .aaa {
        height: 200px;
        $r: red;
        @at-root{
            .bbb{
               color: $r;
            }
        }
    }
}
```

混合器

我们经常会发现一个问题，在写CSS的时候，有很多代码都是相同的。这个时候我们就要想着将这些代码提取出来，后期再直接使用

```scss
@mixin circle {
    width: 200px;
    height: 200px;
    border: 2px solid black;
    border-radius: 50%;
    background-image: linear-gradient(to right, red, blue);
}
.box1 {
    color: black;
    font-size: 32px;
    @include circle();
}
.box2{
    display: flex;
    justify-content: center;
    @include circle();
}
```

在上面的混合器里面，我们可以看到，如果定义了一个混合器以后，后面就可以调用这个混合器，混合器的定义使用 `@mixin 名称` ，调用的时候使用 `@include 名称` 完成

混合器中的参数

在使用混合器的过程当中，我们又发现一个问题，上面的混合器它是个固定大小的圆的一个混合器，我们如果想改变这个圆的 `width/height` 怎么办呢？这个时候就要使用到混合器的参数

```scss
//混合器的参数
@mixin circle($r) {
    width: $r;
    height: $r;
    border: 2px solid black;
    border-radius: 50%;
    background-image: linear-gradient(to right, red, blue);
}
.box1{
    display: flex;
    @include circle(100px);
}
```

混合器参数的默认值

sass在定义混合器的参数的时候，也可以给参数设置一个默认值

```
//混合器的参数
@mixin circle($r:200px) {
    width: $r;
    height: $r;
    border: 2px solid black;
    border-radius: 50%;
    background-image: linear-gradient(to right, red, blue);
}
```

@content占位

其实混合器的高频使用场景还是与`@content`占位符结合在一起使用

```
//混合器与占位符的结合
@mixin circle {
    width: 100px;
    height: 100px;
    border: 2px solid black;
    // 占位符
    @content;
    border-radius: 50%;
}

.box1{
    @include circle(){
        background-color: red;
    }
}
```

它最终生成的代码如下

```
.box1 {
  width: 100px;
  height: 100px;
  border: 2px solid black;
  background-color: red;
  border-radius: 50%;
}
```

混合器的典型用法

```
// 混合器的作用就是将原来的CSS混合到现在的CSS里面

/*0~768px*/
@mixin xs {
    @media only screen and (max-width:768px) {
        @content;
    }
}
/*769~991*/
@mixin sm {
    @media only screen and (min-width:769px) and (max-width:991px) {
        @content;
```

```
    }
}
/*992~1200*/
@mixin md {
    @media only screen and (min-width:992px) and (max-width:1200px) {
        @content;
    }
}
/*1200以下*/
@mixin lg {
    @media only screen and (min-width:1201px) {
        @content;
    }
}
```

在上面的代码里面，我们是定义了4个媒体查询的混合器，然后再进行相应的使用，如下做对

```
@include xs(){
    .tab-bar{
        background-color: red;
    }
}
@include lg(){
    .nav-bar{
        font-size: 32px;
    }
}
```

最终帮我们生成的的`css`代码如下

```
@charset "UTF-8";
@media only screen and (max-width: 768px) {
  .tab-bar {
    background-color: red;
  }
}
@media only screen and (min-width: 1201px) {
  .nav-bar {
    font-size: 32px;
  }
}
```

混合器还可以像下面这样使用

```
@mixin kframes($aaa) {
    @keyframes #{$aaa} {
        @content;
    }
    @-webkit-keyframes #{$aaa}{
        @content;
    }

    @-moz-keyframes #{$aaa}{
        @content;
    }
}
```

```
13
14  @include kframes(ani1){
15      from{
16          width: 100px;
17      }
18      to{
19          width: 200px;
20      }
21  }
```

最终生成的

```
1   @keyframes ani1 {
2     from {
3       width: 100px;
4     }
5     to {
6       width: 200px;
7     }
8   }
9
10  @-webkit-keyframes ani1 {
11    from {
12      width: 100px;
13    }
14    to {
15      width: 200px;
16    }
17  }
18  @-moz-keyframes ani1 {
19    from {
20      width: 100px;
21    }
22    to {
23      width: 200px;
24    }
25  }
```

选择器继承

继承的概念与之前JS编程语文里面的继承也是一样的，它可以继承某一个已经存在的选择器

```
1   .btn1{
2       width: 100px;
3       height: 100px;
4       background-color: deeppink;
5   }
6   #div1{
7       border-radius: 50%;
8   }
9   .abc{
10      text-align: center;
11      @extend .btn1;
12      @extend #div1;
13  }
```

最终生成的CSS代码如下

```
.btn1, .abc {
  width: 100px;
  height: 100px;
  background-color: deeppink;
}
#div1, .abc {
  border-radius: 50%;
}
.abc {
  text-align: center;
}
```

基础总结

本质上面`sass`并不难，它有四个核心的基本点，这四个核心的基本点是我们平常写页面经常会使用到的

1. 变量
2. 嵌套
3. 混合与占位符
4. 继承