# SASS & SCSS

## 官网

<https://sass-lang.com/>

<https://sass-lang.com/guide>

## CSS预编译语言

sass & scss

less

stylus

## 安装

全局安装：npm install -g sass  
可以在终端命令行使用

## 编译

### 使用sass命令

sass a.scss b.css

说明：把a.scss 编译为 b.css

### --watch

您还可以使用--watch标志查看单个文件或目录 。watch标志指示Sass监视源文件中的更改，并在每次保存Sass时重新编译CSS。如果要监视（而不是手动构建）input.scss文件，只需将监视标志添加到命令中，如下所示：

sass --watch input.scss output.css  
通过使用文件夹路径作为输入和输出，并用冒号分隔它们，可以监视和输出到目录。在此示例中：

sass --watch app/sass:public/stylesheets  
Sass会监视文件app/sass夹中的所有文件进行更改，然后将CSS编译到该public/stylesheets 文件夹中。

## 嵌套

SASS允许选择器嵌套。

**nav** {  
 **ul** {  
 **margin**: 0;  
 **padding**: 0;  
 **list-style**: **none**;  
 }  
 **li** { **display**: **inline-block**; }  
 **a** {  
 **display**: **block**;  
 **padding**: 6**px** 12**px**;  
 **text-decoration**: **none**;  
 }  
}

## 父选择器

父选择器&是Sass发明的特殊选择器，在嵌套选择器中使用它来引用外部选择器。这样就可以以更复杂的方式重用外部选择器，例如添加伪类或在父类之前添加选择器。

在内部选择器中使用父选择器时，会将其替换为相应的外部选择器。

scss:

.**alert** {  
 **&**:**hover** {  
 **font-weight**: **bold**;  
 }  
   
 :**not**(**&**) {  
 **opacity**: 0.8;  
 }  
}

css:

.**alert**:**hover** {  
 **font-weight**: **bold**;  
}  
  
:**not**(.**alert**) {  
 **opacity**: 0.8;  
}

## 变量

变量允许我们单独定义一系列通用的样式，然后在需要的时候去调用。所以在做全局样式调整的时候我们可能只需要修改几行代码就可以了

Sass&SCSS变量很简单：所有变量以$开头。您可以为名称分配一个$，然后可以引用该名称而不是值本身。尽管它们很简单，但是它们却是Sass带来的最有用的工具之一。变量使减少重复，进行复杂的数学运算，配置库等成为可能。

变量声明看起来很像属性声明：<variable>: <expression>。

　　$blue : #1875e7;

　　div {

　　　color : $blue;

　　}

如果变量需要镶嵌在字符串之中，就必须需要写在#{}之中。

　　$side : left;

　　.rounded {

　　　　border-#{$side}-radius: 5px;

}

与只能在样式规则中声明的属性不同，可以在任何位置声明变量。要使用变量，只需将其包含在值中即可。

**$base-color**: **#c6538c**;  
**$border-dark**: **rgba**(**$base-color**, 0.88);  
  
.**alert** {  
 **border**: 1**px solid $border-dark**;  
}

示例：如果修改了$red, 那么引用这个变量的地方都改变了：

**$width**: 5**px**;  
**$style**: **solid**;  
**$red**:**red**;  
**$blue**: **blue**;  
  
.**box1** {  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: **$width $style $red**;  
}  
  
.**box2** {  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **border**: **$width $style $blue**;  
}  
  
.**box3** {  
 **width**: 100**px**;  
 **height**: 100**px**;  
 **background-color**: **$red**;  
}

作用域：

在样式表的顶层声明的变量拥有全局作用域。这意味着在声明它们之后，可以在模块的任何位置对其进行访问。  
在块中声明的代码（SCSS中的花括号或Sass中的缩进代码）是局部作用域，只能在声明它们的块中访问。

**$a**: 10**px**;  
  
.**content** {  
 **$b**: 5**px**;  
 **width**: **$a**;  
 **height**: **$b**;  
}  
  
.**sidebar** {  
 **width**: **$a**;  
  
 *// 报错：* **height**: **$b**;  
}

## 继承

SASS允许一个选择器，继承另一个选择器。比如，现有class1：

　　.class1 {

　　　　border: 1px solid #ddd;

　　}

class2要继承class1，就要使用@extend命令：

　　.class2 {

　　　　@extend .class1;

　　　　font-size:120%;

}

在设计页面时，通常情况下，一个类具有另一类的所有样式以及其自己的特定样式。但这会造成混乱的HTML，很容易因忘记同时包含两个类而产生错误，并且会在标记中带来非语义样式问题。

<div class="error error--serious">

Oh no! You've been hacked!

</div>

.**error** {  
 **border**: 1**px #f00**;  
 **background-color**: **#fdd**;  
}  
  
.**error--serious** {  
 **border-width**: 3**px**;  
}

萨斯的@extend法则解决了这个问题。它写成@extend <selector>，并告诉Sass一个选择器应该继承另一个选择器的样式。

.**error** {  
 **border**: 1**px #f00**;  
 **background-color**: **#fdd**;  
  
 **&--serious** {  
 **@extend** .**error**;  
 **border-width**: 3**px**;  
 }  
}

## 计算功能

SASS允许在代码中使用算式：

　　body {

　　　　margin: (14px/2);

　　　　top: 50px + 100px;

}

## Mixin

Mixin有点像C语言的宏（macro），是可以重用的代码块。

使用@mixin命令，定义一个代码块。

　　@mixin left {

　　　　float: left;

　　　　margin-left: 10px;

　　}

使用@include命令，调用这个mixin。

　　div {

　　　　@include left;

}

## 插入文件

@import命令，用来插入外部文件。

　　@import "path/filename.scss";

如果插入的是.css文件，则等同于css的import命令。

　　@import "foo.css";

## 注释

SCSS共有两种注释风格。

单行注释 // comment，只保留在SASS源文件中，不会带到编译后的css文件。

标准的CSS注释 /\* comment \*/ ，会带到编译后的css文件中。

# Babel

## 网站

网站：https://babeljs.io/

https://www.babeljs.cn/

## 下载

### 初始化

如果您没有package.json，请在安装前创建一个。这将确保与npx命令正确交互 。  
因此首先执行 npm init

### 安装babel cli

虽然您可以在计算机上全局安装Babel CLI，但最好逐个项目在本地安装它。  
这有两个主要原因：  
同一台计算机上的不同项目可能取决于Babel的不同版本，从而允许您一次更新一个版本。  
这意味着您对工作的环境没有隐式依赖。使您的项目更加可移植且易于设置。

我们可以通过运行以下命令在本地安装Babel CLI：  
npm install -D @babel/core @babel/cli  
npm install -D @babel/preset-env

## 使用

在src中新建app.js文件：

**let** sum = (a,b) => {  
 console.log(a+b);  
};  
  
**class** Pet {}  
console.log(sum(100,200));

执行命令babel src/app.js 提示：'babel' 不是内部或外部命令，也不是可运行的程序或批处理文件。  
这是因为把babel cli安装到了项目里面（本地安装）；如果是全局安装-g才可以直接在终端使用这个命令。

当把babel cli安装到本地时，它的命令保存在node\_modules/.bin中，因此我们可以执行命令  
.\node\_modules\.bin\babel src\app.js 编译src文件夹中app.js单个文件  
.\node\_modules\.bin\babel src 编译src文件夹中所有的js文件  
.\node\_modules\.bin\babel src -d lib 编译整个src文件夹中所有的js文件，生成到lib文件夹中

然后每次要写上面的路径过于繁琐，与其直接从命令行运行Babel，不如将命令放入使用本地版本的npm脚本中。  
在package.json中提供了一个执行命令的地方，就是scripts选项。只需在package.json的"scripts"选项中添加一个字段，然后将babel命令放入其中即可：

**"scripts"**: {  
 **"test"**: **"echo \"Error: no test specified\" && exit 1"**,  
 **"build"**: **"babel src -d lib"**},

你现在可以使用npm run build命令进行babel编译 src文件夹，目标文件夹是 lib  
把build 改成 任何名称都可以，重要的 后面的内容

## 插件

执行上面的命令编译之后发现没有任何变化，这是因为babel的运行时基于插件的，没有插件babel什么事也不能干。虽然已经配置了Babel，但它不能执行任何实际操作。

插件目录：<https://www.babeljs.cn/docs/plugins>

### 使用插件

其中ES2015插件中有个arrow-functions插件用来解析ES2015箭头函数  
下载：npm install --save-dev @babel/plugin-transform-arrow-functions  
修改build为：

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"build": "babel --plugins @babel/plugin-transform-arrow-functions src/app.js"

},  
执行npm run build 可以看到箭头函数已经被转换，但是class没有被转换

继续在ES2015插件中下载插件：  
npm install --save-dev @babel/plugin-transform-classes  
修改build为：

**"scripts"**: {  
 **"test"**: **"echo \"Error: no test specified\" && exit 1"**,  
 **"build"**: **"babel --plugins @babel/plugin-transform-arrow-functions,@babel/plugin-transform-classes src/app.js"**},

执行npm run build 可以看到箭头函数和class语法已经都被转换

### babel.config.json

上面写法太繁琐：  
在项目根目录中创建babel.config.json配置：

{  
 **"plugins"**: [  
 **"@babel/plugin-transform-arrow-functions"**,  
 **"@babel/plugin-transform-classes"**]  
}

package.json 中build：

**"scripts"**: {  
 **"test"**: **"echo \"Error: no test specified\" && exit 1"**,  
 **"build"**: **"babel src -d lib"**},

### 预设

ES6有很多插件，如果每个都写，也会很麻烦。可以使用预设，预设就是把一些相关的插件打包在一起。

比如：您可以使用env预设，它启用ES2015 +的转换

npm install @babel/preset-env --save-dev

为了启用预设，您必须在babel.config.json文件中定义它，如下所示：

{

"presets": ["@babel/preset-env"]

}