预处理语言的诞生

使用最为普遍的三款 CSS 预处理器框架，分别是 Sass、Less CSS、Stylus。

什么是 CSS 预处理器，CSS 预处理器是一种语言用来为 CSS 增加一些编程的的特性，无需考虑浏览器的兼容性问题，可以在 CSS 中使用变量、简单的程序逻辑、函数等等在编程语言中的一些基本技巧，可以让 CSS 更见简洁，适应性更强，代码更直观等诸多好处。

1. Sass 诞生于 2007 年，Ruby 编写，其语法功能都十分全面，可以说 它完全把 CSS 变成了一门编程语言。另外 在国内外都很受欢迎，并且它的项目团队很是强大 ，是一款十分优秀的预处理语言。
2. Stylus 诞生于 2010 年，来自 Node.js 社区，语法功能也和 Sass 不相伯仲，是一门十分独特的创新型语言。
3. Less 诞生于 2009 年，受Sass的影响创建的一个开源项目。 它扩充了 CSS 语言，增加了诸如变量、混合（mixin）、函数等功能，让 CSS 更易维护、方便制作主题、扩充。
4. Sass 与 Stylus 相比于 Less 功能更为丰富，但对于学习成本以及适应时间 ，Less 稍胜一筹。Less 没有去掉任何 CSS 的功能，而是在现有的语法上，增添了许多额外的功能特性，所以学习 Less 是一件非常舒服的事情。

我们将会从语法、变量、嵌套、混入(Mixin)、继承、导入、函数和操作符等方面分别对这三个框架进行比较介绍。

**语法**

在使用 CSS 预处理器之前最重要的是理解语法，幸运的是基本上大多数预处理器的语法跟 CSS 都差不多。

首先 Sass 和 Less 都使用的是标准的 CSS 语法，因此如果你可以很方便的将已有的 CSS 代码转为预处理器代码，默认 Sass 使用 .sass 扩展名，而 Less 使用 .less 扩展名。

下面是这二者的语法：

/\* style.scss or style.less \*/

h1 {

color: #0982C1;

}

你注意到了，这是一个再普通不过的，不过 Sass 同时也支持老的语法，就是不包含花括号和分号的方式：

/\* style.sass \*/

h1

color: #0982c1

而 Stylus 支持的语法要更多样性一点，它默认使用 .styl 的文件扩展名，下面是 Stylus 支持的语法：

/\* style.styl \*/

h1 {

color: #0982C1;

}

/\* omit brackets \*/

h1

color: #0982C1;

/\* omit colons and semi-colons \*/

h1

color #0982C1

你也可以在同一个样式单中使用不同的写法，例如下面的写法也不会报错：

h1 {

color #0982c1

}

h2

font-size: 1.2em

**变量**

你可以在 CSS 预处理器中声明变量，并在整个样式单中使用，支持任何类型的变量，例如颜色、数值（不管是否包括单位）、文本。然后你可以任意引用该变量。

Sass 的变量必须是 $ 开始，然后变量名和值使用冒号隔开，跟 CSS 的属性一致：

$mainColor: #0982c1;

$siteWidth: 1024px;

$borderStyle: dotted;

body {

color: $mainColor;

border: 1px $borderStyle $mainColor;

max-width: $siteWidth;

}

而 Less 的变量名使用 @ 符号开始：

@mainColor: #0982c1;

@siteWidth: 1024px;

@borderStyle: dotted;

body {

color: @mainColor;

border: 1px @borderStyle @mainColor;

max-width: @siteWidth;

}

Stylus 对变量名没有任何限定，你可以是 $ 开始，也可以是任意的字符，而且与变量值之间可以用冒号、空格隔开，需要注意的是 Stylus (0.22.4) 将会编译 @ 开始的变量，但其对应的值并不会赋予该变量，换句话说，在 Stylus 的变量名不要用 @ 开头。

mainColor = #0982c1

siteWidth = 1024px

$borderStyle = dotted

body

color mainColor

border 1px $borderStyle mainColor

max-width siteWidth

上面的三种不同的 CSS 预处理器的写法，最终都将产生相同的结果：

body {

color: #0982c1;

border: 1px dotted #0982c1;

max-width: 1024px;

}

你可以想象，加入你的 CSS 中使用了某个颜色的地方多达数十次，那么要修改颜色时你必须找到这数十次的地方并一一修改，而有了 CSS 预处理器，修改一个地方就够了！

**嵌套**

如果我们需要在CSS中相同的 parent 引用多个元素，这将是非常乏味的，你需要一遍又一遍地写 parent。例如：

section {

margin: 10px;

}

section nav {

height: 25px;

}

section nav a {

color: #0982C1;

}

section nav a:hover {

text-decoration: underline;

}

而如果用 CSS 预处理器，就可以少些很多单词，而且父子节点关系一目了然。我们这里提到的三个 CSS 框架都是允许嵌套语法：

section {

margin: 10px;

nav {

height: 25px;

a {

color: #0982C1;

&amp;:hover {

text-decoration: underline;

}

}

}

}

最终生成的 CSS 结果是：

section {

margin: 10px;

}

section nav {

height: 25px;

}

section nav a {

color: #0982C1;

}

section nav a:hover {

text-decoration: underline;

}

**Mixins (混入)**

Mixins 有点像是函数或者是宏，当你某段 CSS 经常需要在多个元素中使用时，你可以为这些共用的 CSS 定义一个 Mixin，然后你只需要在需要引用这些 CSS 地方调用该 Mixin 即可。

Sass 的混入语法：

/\* Sass mixin error with (optional) argument $borderWidth which defaults to 2px if not specified \*/

@mixin error($borderWidth: 2px) {

border: $borderWidth solid #F00;

color: #F00;

}

.generic-error {

padding: 20px;

margin: 4px;

@ include error(); /\* Applies styles from mixin error \*/

}

.login-error {

left: 12px;

position: absolute;

top: 20px;

@ include error(5px); /\* Applies styles from mixin error with argument $borderWidth equal to 5px\*/

}

Less CSS 的混入语法：

/\* LESS mixin error with (optional) argument @borderWidth which defaults to 2px if not specified \*/

.error(@borderWidth: 2px) {

border: @borderWidth solid #F00;

color: #F00;

}

.generic-error {

padding: 20px;

margin: 4px;

.error(); /\* Applies styles from mixin error \*/

}

.login-error {

left: 12px;

position: absolute;

top: 20px;

.error(5px); /\* Applies styles from mixin error with argument @borderWidth equal to 5px \*/

}

Stylus 的混入语法：

/\* Stylus mixin error with (optional) argument borderWidth which defaults to 2px if not specified \*/

error(borderWidth= 2px) {

border: borderWidth solid #F00;

color: #F00;

}

.generic-error {

padding: 20px;

margin: 4px;

error(); /\* Applies styles from mixin error \*/

}

.login-error {

left: 12px;

position: absolute;

top: 20px;

error(5px); /\* Applies styles from mixin error with argument borderWidth equal to 5px \*/

}

最终它们都将编译成如下的 CSS 样式：

.generic-error {

padding: 20px;

margin: 4px;

border: 2px solid #f00;

color: #f00;

}

.login-error {

left: 12px;

position: absolute;

top: 20px;

border: 5px solid #f00;

color: #f00;

}

**继承**

当我们需要为多个元素定义相同样式的时候，我们可以考虑使用继承的做法。例如我们经常需要：

p,

ul,

ol {

/\* styles here \*/

}

在 Sass 和 Stylus 我们可以这样写：

.block {

margin: 10px 5px;

padding: 2px;

}

p {

@extend .block; /\* Inherit styles from '.block' \*/

border: 1px solid #EEE;

}

ul, ol {

@extend .block; /\* Inherit styles from '.block' \*/

color: #333;

text-transform: uppercase;

}

在这里首先定义 .block 块，然后让 p 、ul 和 ol 元素继承 .block ，最终生成的 CSS 如下：

.block, p, ul, ol {

margin: 10px 5px;

padding: 2px;

}

p {

border: 1px solid #EEE;

}

ul, ol {

color: #333;

text-transform: uppercase;

}

在这方面 Less 表现的稍微弱一些，更像是混入写法：

.block {

margin: 10px 5px;

padding: 2px;

}

p {

.block; /\* Inherit styles from '.block' \*/

border: 1px solid #EEE;

}

ul, ol {

.block; /\* Inherit styles from '.block' \*/

color: #333;

text-transform: uppercase;

}

生成的 CSS 如下：

.block {

margin: 10px 5px;

padding: 2px;

}

p {

margin: 10px 5px;

padding: 2px;

border: 1px solid #EEE;

}

ul,

ol {

margin: 10px 5px;

padding: 2px;

color: #333;

text-transform: uppercase;

}

你所看到的上面的代码中，.block 的样式将会被插入到相应的你想要继承的选择器中，但需要注意的是优先级的问题。

**导入 (Import)**

很多 CSS 开发者对导入的做法都不太感冒，因为它需要多次的 HTTP 请求。但是在 CSS 预处理器中的导入操作则不同，它只是在语义上包含了不同的文件，但最终结果是一个单一的 CSS 文件，如果你是通过 @ import "file.css"; 导入 CSS 文件，那效果跟普通的 CSS 导入一样。注意：导入文件中定义的混入、变量等信息也将会被引入到主样式文件中，因此需要避免它们互相冲突。

reset.css:

/\* file.{type} \*/

body {

background: #EEE;

}

main.xxx:

@ import "reset.css";

@ import "file.{type}";

p {

background: #0982C1;

}

最终生成的 CSS：

@ import "reset.css";

body {

background: #EEE;

}

p {

background: #0982C1;

}

**颜色函数**

CSS 预处理器一般都会内置一些颜色处理函数用来对颜色值进行处理，例如加亮、变暗、颜色梯度等。

Sass：

lighten($color, 10%); /\* returns a color 10% lighter than $color \*/

darken($color, 10%); /\* returns a color 10% darker than $color \*/

saturate($color, 10%); /\* returns a color 10% more saturated than $color \*/

desaturate($color, 10%); /\* returns a color 10% less saturated than $color \*/

grayscale($color); /\* returns grayscale of $color \*/

complement($color); /\* returns complement color of $color \*/

invert($color); /\* returns inverted color of $color \*/

mix($color1, $color2, 50%); /\* mix $color1 with $color2 with a weight of 50% \*/

上面只是简单列了 Sass 的一些基本颜色处理函数，完整的列表请看 [Sass Documentation](http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html).

下面是一个具体的例子：

$color: #0982C1;

h1 {

background: $color;

border: 3px solid darken($color, 50%);

}

Less CSS:

lighten(@color, 10%); /\* returns a color 10% lighter than @color \*/

darken(@color, 10%); /\* returns a color 10% darker than @color \*/

saturate(@color, 10%); /\* returns a color 10% more saturated than @color \*/

desaturate(@color, 10%); /\* returns a color 10% less saturated than @color \*/

spin(@color, 10); /\* returns a color with a 10 degree larger in hue than @color \*/

spin(@color, -10); /\* returns a color with a 10 degree smaller hue than @color \*/

mix(@color1, @color2); /\* return a mix of @color1 and @color2 \*/

LESS 完整的颜色函数列表请看 [LESS Documentation](http://lesscss.org/#-color-functions).

LESS 使用颜色函数的例子：

@color: #0982C1;

h1 {

background: @color;

border: 3px solid darken(@color, 50%);

}

Stylus:

lighten(color, 10%); /\* returns a color 10% lighter than 'color' \*/

darken(color, 10%); /\* returns a color 10% darker than 'color' \*/

saturate(color, 10%); /\* returns a color 10% more saturated than 'color' \*/

desaturate(color, 10%); /\* returns a color 10% less saturated than 'color' \*/

完整的颜色函数列表请阅读 [Stylus Documentation](http://learnboost.github.com/stylus/docs/bifs.html).

实例：

color = #0982C1

h1

background color

border 3px solid darken(color, 50%)

**运算符**

你可以直接在 CSS 预处理器中进行样式的计算，例如：

body {

margin: (14px/2);

top: 50px + 100px;

right: 100px - 50px;

left: 10 \* 10;

}

**一些跟具体浏览器相关的处理**

这是宣传使用预处理的原因之一，并且是一个很好的理由，这样可以节省的大量的时间和汗水。创建一个mixin来处理不同浏览器的CSS写法是很简单的，节省了大量的重复工作和痛苦的代码编辑。

Sass

@mixin border-radius($values) {

-webkit-border-radius: $values;

-moz-border-radius: $values;

border-radius: $values;

}

div {

@ include border-radius(10px);

}

Less CSS

.border-radius(@values) {

-webkit-border-radius: @values;

-moz-border-radius: @values;

border-radius: @values;

}

div {

.border-radius(10px);

}

Stylus

border-radius(values) {

-webkit-border-radius: values;

-moz-border-radius: values;

border-radius: values;

}

div {

border-radius(10px);

}

编译结果：

div {

-webkit-border-radius: 10px;

-moz-border-radius: 10px;

border-radius: 10px;

}

**3D文本**

要生成具有 3D 效果的文本可以使用 text-shadows ，唯一的问题就是当要修改颜色的时候就非常的麻烦，而通过 mixin 和颜色函数可以很轻松的实现：

Sass

@mixin text3d($color) {

color: $color;

text-shadow: 1px 1px 0px darken($color, 5%),

2px 2px 0px darken($color, 10%),

3px 3px 0px darken($color, 15%),

4px 4px 0px darken($color, 20%),

4px 4px 2px #000;

}

h1 {

font-size: 32pt;

@ include text3d(#0982c1);

}

Less CSS

.text3d(@color) {

color: @color;

text-shadow: 1px 1px 0px darken(@color, 5%),

2px 2px 0px darken(@color, 10%),

3px 3px 0px darken(@color, 15%),

4px 4px 0px darken(@color, 20%),

4px 4px 2px #000;

}

span {

font-size: 32pt;

.text3d(#0982c1);

}

Stylus

text3d(color)

color: color

text-shadow: 1px 1px 0px darken(color, 5%), 2px 2px 0px darken(color, 10%), 3px 3px 0px darken(color, 15%), 4px 4px 0px darken(color, 20%), 4px 4px 2px #000

span

font-size: 32pt

text3d(#0982c1)

生成的 CSS

span {

font-size: 32pt;

color: #0982c1;

text-shadow: 1px 1px 0px #097bb7,

2px 2px 0px #0875ae,

3px 3px 0px #086fa4,

4px 4px 0px #07689a,

4px 4px 2px #000;

}

效果图：



**列 (Columns)**

使用数值操作和变量可以很方便的实现适应屏幕大小的布局处理。

Sass

$siteWidth: 1024px;

$gutterWidth: 20px;

$sidebarWidth: 300px;

body {

margin: 0 auto;

width: $siteWidth;

}

.content {

float: left;

width: $siteWidth - ($sidebarWidth+$gutterWidth);

}

.sidebar {

float: left;

margin-left: $gutterWidth;

width: $sidebarWidth;

}

Less CSS

@siteWidth: 1024px;

@gutterWidth: 20px;

@sidebarWidth: 300px;

body {

margin: 0 auto;

width: @siteWidth;

}

.content {

float: left;

width: @siteWidth - (@sidebarWidth+@gutterWidth);

}

.sidebar {

float: left;

margin-left: @gutterWidth;

width: @sidebarWidth;

}

Stylus

siteWidth = 1024px;

gutterWidth = 20px;

sidebarWidth = 300px;

body {

margin: 0 auto;

width: siteWidth;

}

.content {

float: left;

width: siteWidth - (sidebarWidth+gutterWidth);

}

.sidebar {

float: left;

margin-left: gutterWidth;

width: sidebarWidth;

}

实际效果

body {

margin: 0 auto;

width: 1024px;

}

.content {

float: left;

width: 704px;

}

.sidebar {

float: left;

margin-left: 20px;

width: 300px;

}

**注释**

以上三种框架都支持形如 /\* \*/ 的多行注释以及 // 的单行注释。