# **变量（Variables）**

无需多说，看代码一目了然：

@width: 10px;@height: @width + 10px;

#header {

width: @width;

height: @height;

}

编译为：

#header {

width: 10px;

height: 20px;

}

# 

# **混合（Mixins）**

混合（Mixin）是一种将一组属性从一个规则集包含（或混入）到另一个规则集的方法。假设我们定义了一个类（class）如下：

.bordered {

border-top: dotted 1px black;

border-bottom: solid 2px black;

}

如果我们希望在其它规则集中使用这些属性呢？没问题，我们只需像下面这样输入所需属性的类（class）名称即可，如下所示：

#menu a {

color: #111;

.bordered();

}

.post a {

color: red;

.bordered();

}

.bordered 类所包含的属性就将同时出现在 #menu a 和 .post a 中了。

# **嵌套（Nesting）**

Less 提供了使用嵌套（nesting）代替层叠或与层叠结合使用的能力。假设我们有以下 CSS 代码：

#header {

color: black;

}#header .navigation {

font-size: 12px;

}#header .logo {

width: 300px;

}

用 Less 语言我们可以这样书写代码：

#header {

color: black;

.navigation {

font-size: 12px;

}

.logo {

width: 300px;

}

}

用 Less 书写的代码更加简洁，并且模仿了 HTML 的组织结构。

你还可以使用此方法将伪选择器（pseudo-selectors）与混合（mixins）一同使用。下面是一个经典的 clearfix 技巧，重写为一个混合（mixin） (& 表示当前选择器的父级）：

.clearfix {

display: block;

zoom: 1;

&:after {

content: " ";

display: block;

font-size: 0;

height: 0;

clear: both;

visibility: hidden;

}

}

## 

## **@规则嵌套和冒泡**

@ 规则（例如 @media 或 @supports）可以与选择器以相同的方式进行嵌套。@ 规则会被放在前面，同一规则集中的其它元素的相对顺序保持不变。这叫做冒泡（bubbling）。

.component {

width: 300px;

@media (min-width: 768px) {

width: 600px;

@media (min-resolution: 192dpi) {

background-image: url(/img/retina2x.png);

}

}

@media (min-width: 1280px) {

width: 800px;

}

}

编译为：

.component {

width: 300px;

}@media (min-width: 768px) {

.component {

width: 600px;

}

}@media (min-width: 768px) and (min-resolution: 192dpi) {

.component {

background-image: url(/img/retina2x.png);

}

}@media (min-width: 1280px) {

.component {

width: 800px;

}

}

# 

# **运算（Operations）**

算术运算符 +、-、\*、/ 可以对任何数字、颜色或变量进行运算。如果可能的话，算术运算符在加、减或比较之前会进行单位换算。计算的结果以最左侧操作数的单位类型为准。如果单位换算无效或失去意义，则忽略单位。无效的单位换算例如：px 到 cm 或 rad 到 % 的转换。

*// 所有操作数被转换成相同的单位*@conversion-1: 5cm + 10mm; *// 结果是 6cm*@conversion-2: 2 - 3cm - 5mm; *// 结果是 -1.5cm*

*// conversion is impossible*@incompatible-units: 2 + 5px - 3cm; *// 结果是 4px*

*// example with variables*@base: 5%;@filler: @base \* 2; *// 结果是 10%*@other: @base + @filler; *// 结果是 15%*

乘法和除法不作转换。因为这两种运算在大多数情况下都没有意义，一个长度乘以一个长度就得到一个区域，而 CSS 是不支持指定区域的。Less 将按数字的原样进行操作，并将为计算结果指定明确的单位类型。

@base: 2cm \* 3mm; *// 结果是 6cm*

你还可以对颜色进行算术运算：

@color: #224488 / 2; *//结果是 #112244*background-color: #112244 + #111; *// 结果是 #223355*

不过，Less 提供的 [色彩函数](https://less.bootcss.com/functions/" \l "color-operations) 更有使用价值。

## **calc() 特例**

Released [v3.0.0](https://github.com/less/less.js/blob/master/CHANGELOG.md)

为了与 CSS 保持兼容，calc() 并不对数学表达式进行计算，但是在嵌套函数中会计算变量和数学公式的值。

@var: 50vh/2;width: calc(50% + (@var - 20px)); *// 结果是 calc(50% + (25vh - 20px))*

## 

## **转义（Escaping）**

转义（Escaping）允许你使用任意字符串作为属性或变量值。任何 ~"anything" 或 ~'anything' 形式的内容都将按原样输出，除非 [interpolation](https://less.bootcss.com/features/" \l "variables-feature-variable-interpolation)。

@min768: ~"(min-width: 768px)";.element {

@media @min768 {

font-size: 1.2rem;

}

}

编译为：

@media (min-width: 768px) {

.element {

font-size: 1.2rem;

}

}

注意，从 Less 3.5 开始，可以简写为：

@min768: (min-width: 768px);.element {

@media @min768 {

font-size: 1.2rem;

}

}

在 Less 3.5+ 版本中，许多以前需要“引号转义”的情况就不再需要了。

# **函数（Functions）**

Less 内置了多种函数用于转换颜色、处理字符串、算术运算等。这些函数在[Less 函数手册](https://less.bootcss.com/functions/)中有详细介绍。

函数的用法非常简单。下面这个例子将介绍如何利用 percentage 函数将 0.5 转换为 50%，将颜色饱和度增加 5%，以及颜色亮度降低 25% 并且色相值增加 8 等用法：

@base: #f04615;@width: 0.5;

.class {

width: percentage(@width); *// returns `50%`*

color: saturate(@base, 5%);

background-color: spin(lighten(@base, 25%), 8);

}

# **命名空间和访问符**

(不要和 [CSS @namespace](http://www.w3.org/TR/css3-namespace/) 或 [namespace selectors](http://www.w3.org/TR/css3-selectors/" \l "typenmsp) 混淆了)。

有时，出于组织结构或仅仅是为了提供一些封装的目的，你希望对混合（mixins）进行分组。你可以用 Less 更直观地实现这一需求。假设你希望将一些混合（mixins）和变量置于 #bundle 之下，为了以后方便重用或分发：

#bundle() {

.button {

display: block;

border: 1px solid black;

background-color: grey;

&:hover {

background-color: white;

}

}

.tab { ... }

.citation { ... }

}

现在，如果我们希望把 .button 类混合到 #header a 中，我们可以这样做：

#header a {

color: orange;

#bundle.button(); *// 还可以书写为 #bundle > .button 形式*

}

注意：如果不希望它们出现在输出的 CSS 中，例如 #bundle .tab，请将 () 附加到命名空间（例如 #bundle()）后面。

# **映射（Maps）**

从 Less 3.5 版本开始，你还可以将混合（mixins）和规则集（rulesets）作为一组值的映射（map）使用。

#colors() {

primary: blue;

secondary: green;

}

.button {

color: #colors[primary];

border: 1px solid #colors[secondary];

}

输出符合预期：

.button {

color: blue;

border: 1px solid green;

}

# **作用域（Scope）**

Less 中的作用域与 CSS 中的作用域非常类似。首先在本地查找变量和混合（mixins），如果找不到，则从“父”级作用域继承。

@var: red;

#page {

@var: white;

#header {

color: @var; *// white*

}

}

与 CSS 自定义属性一样，混合（mixin）和变量的定义不必在引用之前事先定义。因此，下面的 Less 代码示例和上面的代码示例是相同的：

@var: red;

#page {

#header {

color: @var; *// white*

}

@var: white;

}

# **注释（Comments）**

块注释和行注释都可以使用：

*/\* 一个块注释*

*\* style comment! \*/*@var: red;

*// 这一行被注释掉了！*@var: white;

# 

# **导入（Importing）**

“导入”的工作方式和你预期的一样。你可以导入一个 .less 文件，此文件中的所有变量就可以全部使用了。如果导入的文件是 .less 扩展名，则可以将扩展名省略掉：

@import "library"; *// library.less*@import "typo.css";