目 录

1 Less 基础

1.1 认识 Less

CSS（层叠样式表）是一项出色的技术，它使得网页的表现与内容完全分离，使网站维护工作变得更容易，不会因为内容的改变而影响表现，也不会因为表现的改变而影响内容。

作为一门标记性语言，CSS 的先天性优点是语法相对简单，对使用者的要求较低，但它的致命弱点是需要书写大量看似没有逻辑的代码，不方便维护及扩展，不利于复用。

造成这一现象的很大原因在于 CSS 是一门非程序式语言，没有变量、函数、SCOPE（作用域）等概念。

Less 的出现，为 Web 开发者带来了福音，它是一门 CSS 预处理语言，引入了变量、运算、函数、继承等功能，为 CSS 语言赋予了动态语言的特性。

Less 大大简化了 CSS 的编写，并且降低了 CSS 的维护成本。就像它的名称所说的那样，Less 可以让我们用更少的代码做更多的事情。

1.2 Less 原理

Less 包含一套自定义的语法及一个解析器，用户根据这些语法定义自己的样式规则，这些规则最终会通过解析器，编译生成对应的 CSS 文件。

Less 并没有裁剪 CSS 原有的特性，更不是用来取代 CSS 的，而是在现有 CSS 语法的基础上，为 CSS 加入程序式语言的特性。先看一个简单的例子，Less 代码如下：

```
@color: #4d926f;

header {
  color: @color;
}
h2 {
  color: @color;
}
```

上面的例子定义了一个变量 @color，然后在选择器 header 和 h2 中使用它。编译后的 CSS 代码为：

```
header {
  color: #4d926f;
}
h2 {
```



```
    color: #4d926f;
}
```

从上面的例子可知，Less 并没有改变 CSS 的语法。因此，学习 Less 非常容易，只要你了解 CSS 基础，就可以很容易上手。

1.3 Less 用法

Less 可以直接在浏览器端运行（支持 IE6+、Webkit、Firefox），也可以借助 Node.js 或者 Rhino 在服务端运行。

Less 是一种动态语言，无论是在浏览器端，还是在服务器端运行，最终还是需要编译成 CSS，才会发挥其作用。

1.3.1 浏览器端使用

在浏览器端直接使用 Less，浏览器会直接为页面应用编译后的 CSS 样式，而不是生成单独的 CSS 文件。

在浏览器端直接使用 Less，需要一个脚本的支持，这个脚本就是 Less.js，它 Less 解析器，可以在浏览器端把 .less 文件解析成 CSS 样式。你可以从 http://Lesscss.org 下载最新版本的 Less.js。

浏览器端使用 Less，只需两步：

第一步，引入 .less 文件。

```
<link rel="stylesheet/less" type="text/css" href="styles.less">
```

可以看出，Less 源文件与标准 CSS 文件的引入方式完全相同，只是在引入 .less 文件时，要将 rel 属性设置为“stylesheet/less”。

第二步，引入 Less.js 文件。

```
<script src="Less.js" type="text/javascript"></script>
```

需要特别注意的是：

1）.less 样式文件一定要在 Less.js 之前引入，这样才能保证 .less 文件被正确编译。

2）由于浏览器端使用 Less 时，是使用 ajax 来拉取 .less 文件，如果直接在本机文件系统打开（即地址是 file://开头）或者是有跨域的情况下，会拉取不到 .less 文件，导致样式无法生效。因此，必须在 http(s)协议下使用，即必须在服务器环境下使用。

3）还有一种情况容易导致样式无法生效，就是部分服务器（以 IIS 居多）会对未知后缀的文件返回 404，导致无法正常读取 .less 文件。解决方案是，在服务器中为 .less 文件配置 MIME 值为 text/css。一种更简单的方法，就是直接将 .less 文件改名为 .css 文件即可。

1.3.2 服务器端使用

在开发阶段，在页面中嵌入一个 Less.js 将 Less 在线编译成 CSS 样式，确实很方便。但是，在线编译会产生加载延迟，即便在浏览器中有不足一秒的加载延迟，也会降低性能。如果 Javascrip 执行错误，还会引起美观问题。因此，在生产环境中，并不推荐这种方式，而是推荐在服务器端使用 Less。

在服务器端使用 Less，需要借助于 Less 的编译器，由它将 Less 源文件编译成最终的 CSS 文件。最常用方式，就是利用 node 的包管理器 (npm) 进行安装，安装成功后就可以在 node 环境中对 Less 源文件进行编译。

❑ 安装 Less 编译器

为了方便使用 Lessc 这个全局命令，建议采用全局安装。安装命令如下：

```
$ npm install Less -g
```

如果想安装指定版本，也非常方便，只需在安装包后添加 @VERSION 即可。如，安装 1.6.2 版本的命令如下：

```
$ npm install Less@1.6.2 -g
```

当然，如果你想安装最新版本，可以尝试以下命令：

```
$ npm install Less@latest -g
```

整个安装过程无需人工干预，安装完成后的结果如图 1-1 所示：

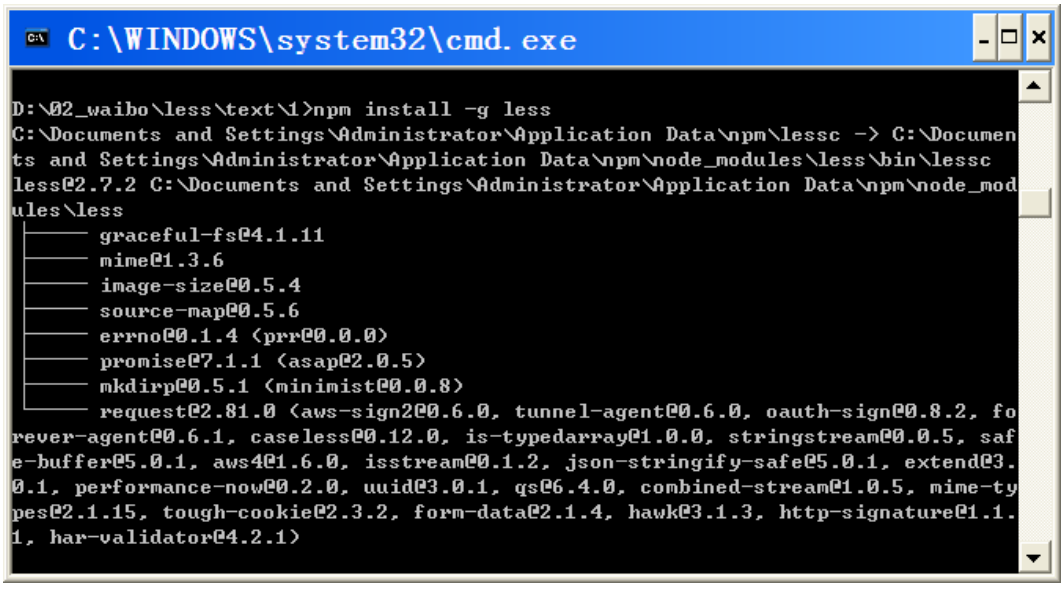

图 1-1 安装 Less 编译器

❑ 代码中用法

只要安装了 Less，就可以在 Node 中像这样调用编译器：

```
var Less = require('Less');

Less.render('.class { width: 1 + 1 }', function (e, css) {
```



```
    console.log(css);
});
```

经过编译生成的 CSS 代码为：

```
.class {
  width: 2;
}
```

你也可以手动调用解析器和编译器：

```
var parser = new(Less.Parser);
parser.parse('.class { width: 1 + 1 }', function (err, tree) {
    if (err) { return console.error(err) }
    console.log(tree.toCSS());
});
```

- 命令行用法

也可以使用命令行，将 Less 文件编译成静态的 CSS 文件，然后在 HTML 文档中直接引入 CSS 文件，而不是 Less 文件。只需将命令行切换到 styles.less 文件所在的目录，并执行 Lessc 命令即可。命令如下：

```
$ Lessc styles.less
```

上面的命令会将编译后的 CSS 输出到 stdout。如果希望将 CSS 代码保存到指定的文件中，就可以使用以下命令：

```
$ Lessc styles.less styles.css
```

上述命令就会将 styles.less 文件编译后的 CSS 代码保存到 styles.css 文件中。如何你希望编译后得到压缩的 CSS，只需提供一个 -x 参数就可以了。命令如下：

```
$ Lessc styles.less styles.css -x
```

1.4 Less 编译工具

虽然你可以选择在浏览器端使用 Less，直接在页面中嵌入一个 Less.js 文件，你也可以选择在服务器端使用 Less，使用命令行将 Less 文件编译成最终的 CSS 文件。

然而，这两种方式都不够灵活，人们更喜欢使用图形界面工具进行编译。常见的工具有 winLess、simpLess、Koala 等，最值得推荐的编译工具，非 Koala 莫属。

Koala 是由国人开放的一款开源的前端预处理器语言图形编译工具，可以跨平台运行，完美兼容 windows、linux、mac，支持 Less、Sass、Compass、CoffeeScript，帮助 Web 开发人员更高效地进行开发。

有了 Koala，就再也不用手动输入命令进行编译，因为 Koala 会监听文件内容的改变，

并自动将 .less 文件编译成 .css 文件，省时省力。你还像以前一样，在页面中直接使用 .css 文件，工作方式没有发生任何改变，也不会改变你的使用习惯。

1.4.1 选择 Koala 的理由

Koala 具有以下优点，这正是推荐它的根本原因：

- 多语言支持：支持 Less、Sass、CoffeeScript 和 Compass Framework。
- 实时编译：监听文件，当文件改变时自动执行编译，这一切都在后台运行，无需人工操作。
- 编译选项：既可统一设置文件的编译选项，也可单独设置某个文件的编译选项。
- 强大的文件右键功能：在文件上右键，即可操作打开文件、打开文件目录、打开输出文件目录、设置输出文件目录、编译、删除六大常用功能。
- 错误提示：如果编译时遇到语法错误，Koala 会自动弹出错误信息，方便开发者定位代码错误位置。
- 跨平台：windows、linux、mac 都能完美运行。
- 免费且负责：Koala 完全免费，而且作者很负责，有什么问题作者都会及时给予答复，意见什么的可以直接提交给作者，一般在下一个版本就能得到解决。

1.4.2 如何使用 Koala

开发者可以在百度搜索“Koala 下载”，根据系统平台下载对应的版本。比如，我们下载是 V2.0.4 版本，它是绿色版本，可以直接使用。

Koala 的界面简单、清晰，也很漂亮。如图 1-2 所示：

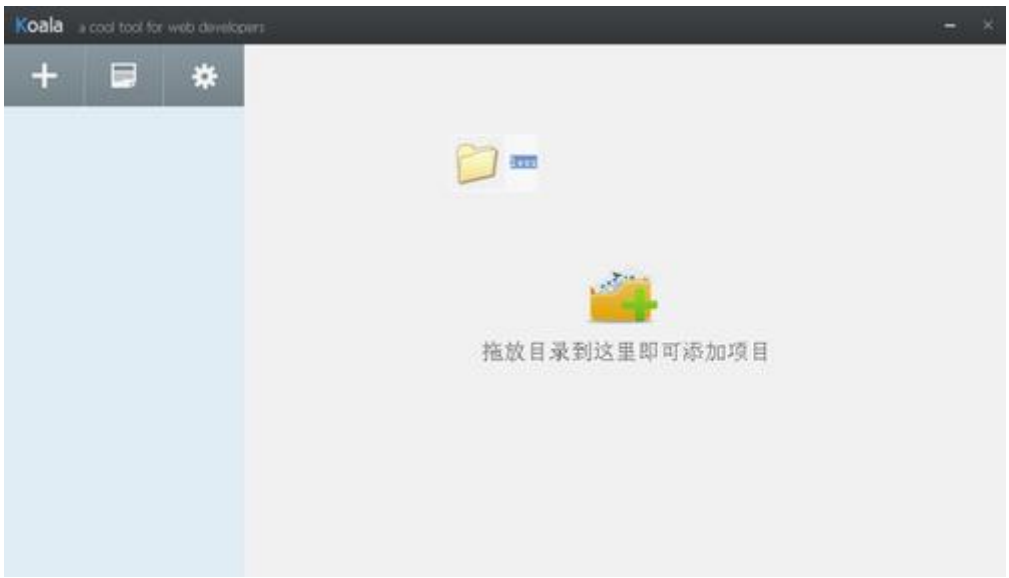

图 1-2 Koala 界面

有了 Koala，一切将变得非常简单：

第一步：把目录拖进窗口，或点击左上角加号图标，选择需要编译的 Less 文件目录。

第二步：在编辑器中编写 Less 代码。

第三步：编写完成后，Ctrl+S 保存文件，Koala 会在后台自动编译出 CSS 文件。

1.4.3 Koala 界面介绍

把文件夹拖入窗口后，Koala 会自动创建相应的工程。Koala 的界面被分为四个区域，分别是按钮区、工程区、工程文件列表区、设置区。如图 1-3 所示：

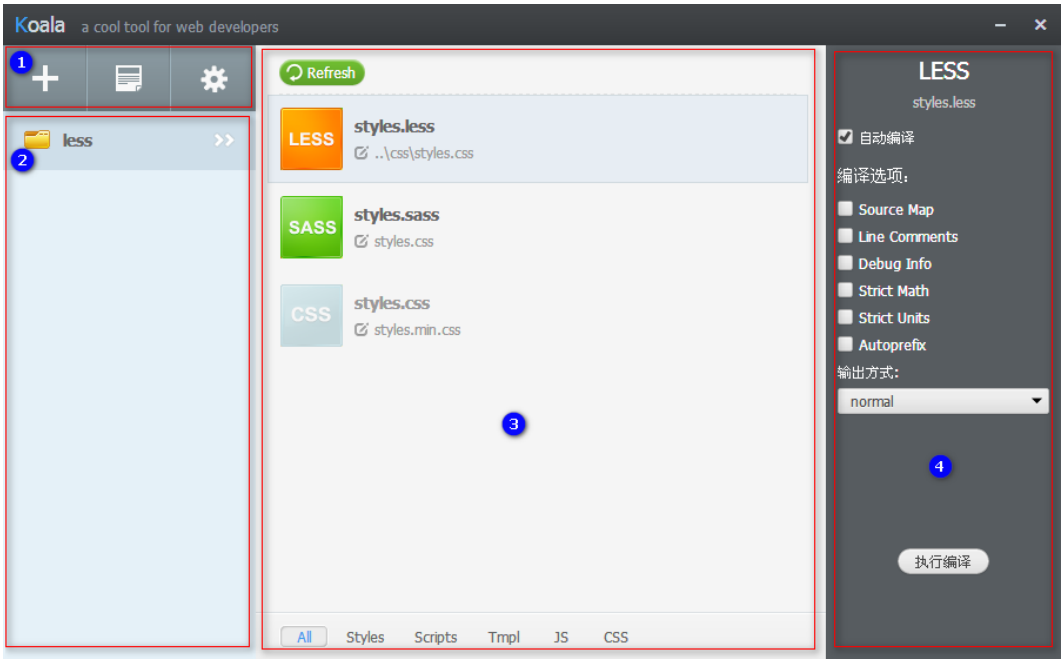

图 1-3 Koala 界面分区

- 按钮区

第一个区域为按钮区，包含三个按钮。第一个按钮用于添加工程，第二个按钮用于打开编译文件的错误提示信息，第三个按钮用于 Koala 的全局设置，可以设置界面语言，Less、Sass 等的默认选项和输出方式及高级设置等。当然，这里也包括 Koala 的当前版本号及作者信息等。

- 工程区

第二个区域为工程区，该区域会显示所有的工程，一行是一个工程。可以直接把文件夹拖入到该区域，每拖入一个文件夹就会添加一个工程。

- 文件列表区

第三个区域为工程的文件列表区，显示被选中工程中的文件列表，每行是一个文件。每个文件的前面，有一个矩形图标用来标识文件的类型，图标后面紧跟文件名及编译输出的 CSS 文件的路径。

如果该文件需要自动编译，图标和文本会以深色正常显示，否则会反灰显示（如，CSS 就被反灰显示）。

可以直接把文件拖入到该区域，拖入的文件会被添加到被选中的工程中。添加文件后，需要点击上面的 refresh 按钮来刷新文件列表。选中文件后，键盘点 delete 键，可以将该文件从工程中移除。也可以在某个文件上右击鼠标，根据弹出的功能菜单执行相应的操作。如图 1-4 所示：

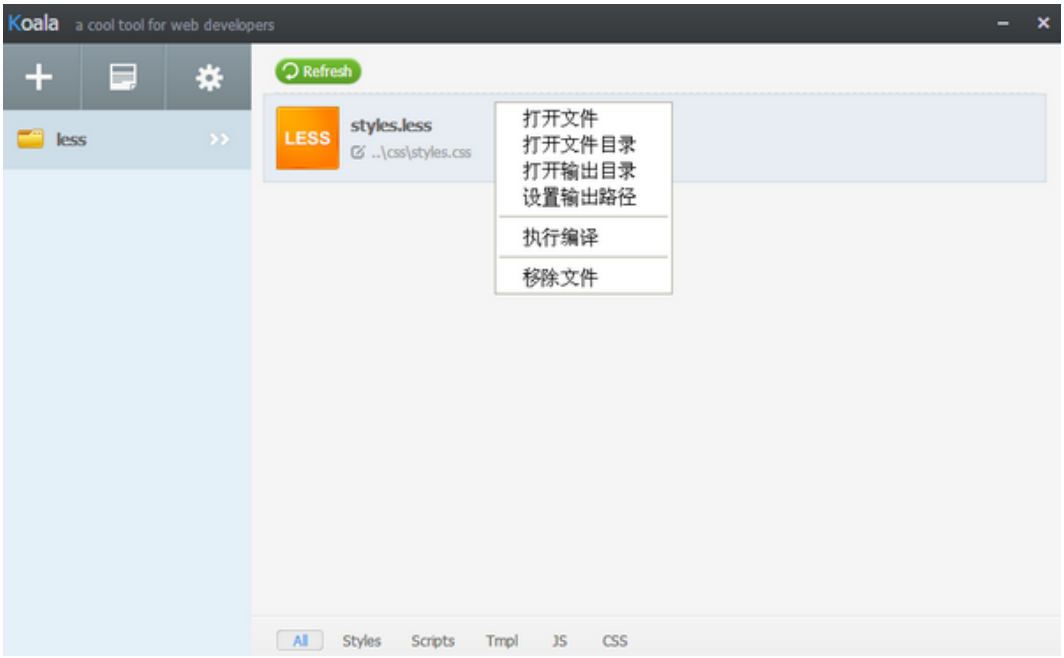

图 1-4 Koala 文件右键菜单

如果工程中的文件类型较多，也可以通过下面的 all/Styles/Scripts/Tmpl/JS/CSS 来过滤需要编译的文件。

❑ 编译选项设置区

第四区域为文件的编译选项设置区，该区域默认不会显示出来，只有当鼠标单击文件列表区的某个文件时，才会显示出来。

这个区域可以针对某个文件设置自动编译、编译选项、输出方式，还可以点击“执行编译”来手动编译。

1、自动编译

当开启自动编译时，在编辑器中按下 Ctrl+S 保存文件时，Koala 会自动编译，并自动更新相应的 CSS 文件。如果没有打开自动编译功能，就需要手动点击“执行编译”，才会编译并更新 CSS 文件。

2、编译选项

1）source map（源文件映射）：开启该功能后，在编译 Less 的同时，会生成相应的 source map 文件。这样，就可以在 Firefox 或谷歌浏览器中，通过 sourcemap 来直接调试 Less 源文件。如，以下 Less 文件（下同）：

```
@color: #4D926F;
```

```
.nav  {
      color: @color;
}
```

编译后 CSS 文件为：

```
.nav {
  color: #4d926f;
}
/*# sourceMappingURL=styles.css.map */
```

可以看出，在生成的 CSS 文件的末尾，会指示生成的 source map 的 URL 信息。本例中，source map 的 URL 为 Less 文件相同目录下的 styles.css.map 文件。styles.css.map 是一个独立的文件，内容如下：

```
{"version":3,"file":"styles.css","sources":["styles.less"],"names":[],"mappings":"AACA;EACI,cAAA;;AAGJ;EACE,cAAA","sourceRoot":"..\\less"}
```

简单说，.map 文件就是一个关系映射文件，里面存储着编译前和编译后代码的文件、行号、列号和变量名的映射关系，以便开发人员根据这些位置信息来调试 Less 源文件。

2）line comments（行注释）：开启该功能后，会在编译出来的 CSS 文件中对应的 CSS 代码上方，提供一个注释，用来说明 CSS 代码来自哪个 Less 文件，以及从哪一行开始。

编译后 CSS 文件为：

```
/* line 3, styles.less */
.nav  {
  color: #4d926f;
}
```

3）debug info（调试信息）：开启该功能后，编译生成的 CSS 文件中就会包含调试信息。这样的话，通过 Firefox 或谷歌的调试工具，就可以定位到 Less 文件，而不是编译后的 CSS 文件。

编译后 CSS 代码为：

```
@media
-sass-debug-info{filename{font-family:file\:\/\/D\:\/less\/styles\.less}line{font-family:\ 32}}
.nav {
  color: #4d926f;
}
```

如果编译生成的 CSS 文件中没有以 @media -sass-debug-info 开头的代码，说明没有

输出调试信息。请使用 Koala 重新编译 Less 文件，并确定已经勾选了 debug info 选项。

4）strict math（严格运算模式）：开启“严格运算模式”后，数学运算必须放在括号中，否则可能会出现错误的结果。如以下 Less 代码：

```
p  {
    line-height: 1+1;
}
```

在关闭“严格运算模式”时，编译后 CSS 代码为：

```
p {
  line-height: 2;
}
```

在开启“严格运算模式”时，编译后 CSS 代码为：

```
p  {
    line-height: 1+1;
}
```

显然，这并不是我们想要的结果。因此，在开启“严格运算模式”后，如果想要得到正确的结果，所有的数学运算都必须放在括号中。如：

```
p  {
    line-height: (1+1);
}
```

5）strict units（严格单位模式）：开启“严格单位模式”后，在编译时将强制验证单位的合法性。如，4px/2px 结果为 2，而不是 2px，而 4em/2px 将报错。如以下 Less 代码：

```
.nav  {
    font-size: 100em/5px;
}
```

由于上述代码没有使用严格单位，因此在“严格单位模式”下，就会出现编译错误。错误信息如下：

SyntaxError: Multiple units in dimension. Correct the units or use the unit function. Bad unit: em/px in D:\less\styles.less:2:4

```
1 .nav  {
2       font-size: 100em/5px;
3 }
```


6）autoprefix（自动前缀模式）：开启“自动前缀模式”后，在编译时会为某些非常新的 CSS 属性添加浏览器私有前缀。如，以下 Less 代码：

```
h2 {
    transform: rotate(45deg);
}
```

在开启“自动前缀模式”时，编译后 CSS 代码为：

```
h2 {
    -webkit-transform: rotate(45deg);
        -ms-transform: rotate(45deg);
            transform: rotate(45deg);
}
```

2 Less 特性

Less 做为 CSS 的一种形式的扩展，它不仅完全兼容 CSS 语法，就连新增的特性也是使用 CSS 语法。所以，学习 Less 是一件轻而易举的事情。

2.1 变量

2.1.1 普通变量

对于程序开发人员来说，变量应该是最熟悉不过的概念了。如果多次重复使用一个信息，将它设置为一个变量，就可以在代码中重复引用。这不仅避免重复定义，还能使代码更容易维护。

Less 中，变量由变量名称和值组成。变量名以 @ 为前缀，由字母、数字、_和-组成，变量名称和值之间用冒号隔开。如：

```
/* 定义变量 */
@color: #4d926f;
/* 应用到元素中 */
header {
    color: @color;
}
h2 {
    color: @color;
}
```


上面的代码定义了一个变量 @color，并给它赋值为 #4d926f。然后，就可以在选择器 header 和 h2 中反复使用它，而不必重复定义。编译后的 CSS 代码为：

```
header {
  color: #4d926f;
}
h2 {
  color: #4d926f;
}
```

从上面的代码可以看出，变量是 VALUE（值）级别的复用，可以将相同的值定义成变量，来统一管理起来。当需要调整样式时，只需要修改相应变量的值就可以了，非常方便。因此，变量适用于定义主题，可以将背景颜色、字体颜色、边框属性等常规样式进行统一定义，不同的主题只需要定义不同的变量文件就可以了。

当然，变量也同样适用于 CSS RESET（重置样式表），在 Web 开发中，往往需要屏蔽浏览器的默认样式，就可以使用 Less 的变量特性。这样，就可以在不同项目间重用样式表，我们仅仅需要在不同的项目样式表中，根据需求重新给变量赋值即可。

2.1.2 变量插值

不仅可以直接在属性值中使用变量，还可以用类似 @{name} 的结构，以“插值”的方式在选择器名、属性名、URL、import、媒体查询中使用变量。在编译时，变量将被替换为它们相应的值。

1）选择器名插值

Less 选择器名称中可以引用任何变量。如：

```
@head: h;
.@{head}2 {
  font-size: 16px;
}
```

编译后的 CSS 代码为：

```
.h2 {
  font-size: 16px;
}
```

2）属性名插值

Less 属性名称中可以引用任何变量。如：


```
@my-property: color;
.myclass {
  background-@{my-property}: #81F7D8;
}
```

编译后的 CSS 代码为：

```
.myclass {
  background-color: #81F7D8;
}
```

3）URL 插值

变量还可以用来保存 URL，并在 url() 中使用它。如：

```
@host: "http://www.waibo.wang/";
h2 {
  color: @color;
  background: url("@{host}img/bg.png");
}
```

编译后的 CSS 代码为：

```
h2 {
  color: #f00;
  background: url("http://www.waibo.wang/img/bg.png");
}
```

4）import 插值

import 语句中，可以使用保存路径的变量。如：

```
@host: "http://www.waibo.wang/";
@import "@{host}/reset.less";
```

5）媒体查询插值

如果你希望在 media query 中使用 Less 变量，你可以直接使用普通的变量方式。因为“~”后面的值是不被编译的，所以可以用作 media query 的参数。如：

```
@singleQuery: ~"(max-width: 480px)";
@media screen, @singleQuery {
    div {
    width:2000px;
```



```
    }
}
```

编译后的 CSS 代码为：

```
@media screen, (max-width: 480px) {
  div {
    width: 2000px;
  }
}
```

2.1.3 变量作用域

Less 中的变量和其他编程语言一样，可以实现值的复用，同样它也有作用域（scope）。简单的讲，变量作用域就是局部变量和全局变量的概念。

Less 中，变量作用域采用的是就近原则，换句话说，就是先查找自己有没有这个变量，如果有，就取自己的变量，如果没有，就查找父元素，依此类推。先看一个简单的例子，Less 文件如下：

```
@width : 20px;
 #homeDiv  {
    @width : 30px;
    #centerDiv  {
        width : @width;  // 此处应该取最近定义的变量 width 的值 30px
    }
 }
 #leftDiv {
      width : @width;  // 此处应该取最上面定义的变量 width 的值 20px
}
```

编译后的 CSS 代码为：

```
#homeDiv #centerDiv {
    width: 30px;
}
#leftDiv {
    width: 20px;
}
```


2.2 Mixins

2.2.1 什么是 Mixin

Less 中，允许你将一个类嵌入到另一个类中，被嵌入的类也可以看作变量。换句话说，你可以用一个类定义样式，然后把它当作变量，在另一个类中，只要引用变量的名字，就能使用它的所有属性，

Less 把这种特性称作 mixin，中文把翻译为“混入”或“混合”，其目的就是从现有的样式中添加属性。请看以下 Less 代码：

```
.bordered {
   border-top: dotted 1px black;
   border-bottom: solid 2px black;
}
```

上述代码中，.bordered 定义了一个属性集。然后，在任何需要使用 .bordered 属性集的选择器中，只需像下面这样调用就可以了：

```
#menu a {
   color: #111;
   .bordered;
}
.post   a {
   color: red;
   .bordered;
}
```

这样一来，.bordered 中定义的属性集，就会在 #menu a 和 .post a 中体现出来。编译后的 CSS 代码为：

```
.bordered {
   border-top: dotted 1px black;
   border-bottom: solid 2px black;
}
#menu a {
   color: #111;
   border-top: dotted 1px black;
   border-bottom: solid 2px black;
```



```
}
.post    a {
    color: red;
    border-top: dotted 1px black;
    border-bottom: solid 2px black;
}
```

从上面的代码可以看出：mixin 其实就是一种嵌套，简单的讲，mixin 就是规则级别的复用。除了类选择器外，你也可以使用 id 选择器来定义 mixin。如：

```
#bordered {
    border-top: dotted 1px black;
    border-bottom: solid 2px black;
}
#menu a {
    color: #111;
    #bordered;
}
```

编译后的 CSS 代码为：

```
#bordered {
    border-top: dotted 1px black;
    border-bottom: solid 2px black;
}
#menu a {
    color: #111;
    border-top: dotted 1px black;
    border-bottom: solid 2px black;
}
```

从上面的代码可以看出，使用 class、id 定义 mixin 时，mixin 的定义会被原封不动的输出到编译生成的 CSS 代码中。

如果希望编译生成的 CSS 代码中不包含 mixin 的定义，在定义 mixin 时，只需在 class、id 的后面添加一对小括号即可。而在调用时，小括号是可选的。如：

```
#x() {
```



```
    border-top: dotted 1px black;
    border-bottom: solid 2px black;
}
#menu a {
    color: #111;
    #x;
}
```

编译后的 CSS 代码为：

```
#menu a {
    color: #111;
    border-top: dotted 1px black;
    border-bottom: solid 2px black;
}
```

2.2.2 带参数的 mixin

在 Less 中，还可以像函数一样定义一个带参数的 mixin，这种形式叫做 Parametric Mixin，即带参数的混入。如：

```
// 定义一个样式选择器
.borderRadius(@radius){
    border-radius: @radius;
}
```

然后，在其他选择器中像这样调用它：

```
// 使用已定义的样式选择器
#header {
    .borderRadius(10px);    // 把 10px 作为参数传递给样式选择器
}
.btn {
    .borderRadius(3px);    // 把 3px 作为参数传递给样式选择器
}
```

编译后的 CSS 代码为：

```
#header {
```



```
    border-radius: 10px;
}
.btn {
    border-radius: 3px;
 }
```

还可以给 Mixin 的参数设置默认值，有了默认值，在调用它的时候，如果没有提供该参数，它就会使用默认值。比如，设置默认值为 5px：

```
.borderRadius(@radius:5px){
    border-radius: @radius;
 }
.btn {
    .borderRadius;
 }
```

由于在调用 .borderRadius 时没有传递参数，它就会使用默认值 5px。编译后的 CSS 代码为：

```
.btn {
    border-radius: 5px;
 }
```

Mixin 还可以带多个参数，参数之间使用逗号或分号隔开。如：

```
.border(@width, @style, @color) {
     border: @width @style @color;
}
h2 {
     .border(2px, dashed, green);
}
```

编译后的 CSS 代码为：

```
h2 {
   border: 2px dashed #008000;
}
```

虽然多个参数可以使用分号或者逗号分隔，推荐使用分号分隔，因为逗号既可以表示混合的参数，也可以表示一个参数中一组值的分隔符。

使用分号作为参数分隔符，就意味着可以将逗号分隔的一组值作为一个变量处理。换句话说，如果编译器在混入的定义或者调用中找到至少一个分号，就会假设参数是使用分号分隔的，则逗号将被看作是一个参数中一组值的分隔符。如：

.name(1, 2, 3; 4, 5) 就包含 2 个参数，1, 2, 3 是一个参数，4, 5 是一个参数。每个参数都是通过逗号分隔的一组值。

.name(1, 2, 3) 就包含 3 个参数，每个参数只含一个数字。也可以使用一个象征性的分号，如 .name(1, 2, 3;)，就可以创建一个只含一个参数，但参数包含一组值的混入。

当包含多个参数时，Mixins 是通过参数的名称，而不是位置来引用参数。也就是说，在使用的时候，如果提供参数名称，则参数的顺序并不重要。命名参数使代码更清晰、更容易阅读。如：

```
.mixin(@color: black, @fontSize: 10px) {
  color: @color;
  font-size: @fontSize;
}
.class1 {
  .mixin(@fontSize: 20px, @color: #F5A9D0);
}
.class2 {
  .mixin(#F79F81, @fontSize: 20px);
}
```

编译后的 CSS 代码为：

```
.class1 {
  color: #f5a9d0;
  font-size: 20px;
}
.class2 {
  color: #f79f81;
  font-size: 20px;
}
```

除此之外，像 JavaScript 中 arguments 一样，Mixin 也有这样一个变量：@arguments。当 Mixin 引用这个参数时，它表示传递进来的所有参数。

很多情况下，@arguments 参数可以省去很多代码。比如，在定义 box-shadow 的属性

值时，如果不想单独处理每一个参数的话，就可以像这样写：

```
.boxShadow(@x:0,@y:0,@blur:1px,@color:#000){
    box-shadow: @arguments;
 }
 #header {
    .boxShadow(2px,2px,3px,#f36);
 }
```

编译后的 CSS 代码为：

```
#header {
    box-shadow: 2px 2px 3px #f36;
 }
```

如果需要在 mixin 中不限制参数的数量，就可以在变量名后添加“...”，表示这里可以使用可变参数。这一点跟 C 语言比较类似。请看以下简单实例：

对于某些属性，它的参数个数是不确定的，而具体的参数个数，只有在使用的时候才知道，这种情况就适合使用可变参数。比如，padding 属性可以接受 1 个、或 2 个、或 3 个、或 4 个参数，就可以使用可变参数。Less 代码如下：

```
.padding(...) {
      padding: @arguments;
}
.class1 {
      .padding(20px);
}
.class2 {
      .padding(20px 30px);
}
.class3 {
      .padding(20px 30px 40px);
}
.class4 {
      .padding(20px 30px 40px 50px);
}
```


编译后的 CSS 代码为：

```
.class1 {
  padding: 20px;
}
.class2 {
  padding: 20px 30px;
}
.class3 {
  padding: 20px 30px 40px;
}
.class4 {
  padding: 20px 30px 40px 50px;
}
```

可变参数的常见形式如下：

```
.mixin(...) { }          // it matches arguments from 0-n
.mixin() { }             // it matches exactly 0 arguments
.mixin(@x: 1) { }        // it matches arguments from 0-1
.mixin(@x: 1, ...) { }   // it matches arguments from 0-n
```

2.2.3 作为函数的 mixin

在一个 mixin 内部定义的变量或 mixin，都调用者可见，因此，它们可以作为它的返回值。如，以下 Less 代码：

```
.count(@x, @y) {
    @sum:(@x + @y);
    @average: ((@x + @y) / 2);
}
```

上述代码在 .count 内部定义了两个变量 @sum 和 @average，则 .count 将拥有两个返回值。调用 .count 时，就可以通过变量 @sum 和 @average 来使用返回值。如：

```
div {
    .count(10px, 30px);      // 调用
    margin: @sum;            // 得到返回值 @sum，即 10px + 30px
    padding: @average;       // 得到返回值 @average，即(10px + 30px) / 2
```



```
}
```

编译后的 CSS 代码为：

```
div {
  margin: 40px;
  padding: 20px;
}
```

除了在 mixin 内部定义的变量外，还可以在一个 mixin 中定义另一个 mixin，内部的 mixin 将成为外部 mixin 的返回值。如，以下 Less 代码：

```
.outerMixin(@value) {
  .nestedMixin() {
    line-height: @value*2;
  }
}
p {
  .outerMixin(2em);
  .nestedMixin();
}
```

编译后的 CSS 代码为：

```
p {
  line-height: 4em;
}
```

2.3 嵌套规则

在使用标准 CSS 时，要为多层嵌套的元素定义样式，要么使用后代选择器从外到内的嵌套定义，要么给这个元素加上类名或 id 来定义。这样的写法虽然很好理解，但维护起来很不方便，因为无法清晰了解到样式之间的关系。

在 Less 中，嵌套规则使这个问题迎刃而解。嵌套规则允许在一个选择器中嵌套另一个选择器，这更容易设计出精简的代码，并且样式之间的关系一目了然。假设以下 HTML 代码片段：

```
<header>
    <h1><a href="http://www.waibo.wang/">歪脖网</a></h1>
    <p>学 Web 开发，就到歪脖网！</p>
```



```
</header>
```

Less 代码可以这样写：

```
header {
    h1 {
        font-size: 26px;
        font-weight: bold;
        a {
            color: #f36;
            text-decoration: none;
            &:hover {
                color: #63f;
                text-decoration: underline;
            }
        }
    }
    p {
        font-size: 12px;
        &.slogan {
            color: #f00;
        }
    }
}
```

这难道不就是 DOM 的写法吗？说实话，当你第一眼看到这种写法，你就会情不自禁地爱上 Less。这种写法减了选择器的层级关系，使代码的结构非常清晰，无论是阅读、还是后期维护都是那么自然，是不是有一种本来就该如此的感觉？

在使用嵌套规则时，需要特别注意 & 符号。内层选择器前面的 & 符号就表示对父选择器的引用。在一个内层选择器的前面，如果没有 & 符号，则它被解析为父选择器的后代；如果有 & 符号，它就被解析为父元素自身或父元素的伪类。

比如，上述代码中，由于选择器 h1 前面没有 & 符号，则 h1 被解析为 header 选择器的后代，即 header h1；而 :hover 和 .slogan 前面有 & 符号，& 符号表示对父选择器的引用，则 &.slogan 表示父元素自身，&:hover 表示父元素的伪类，解析结果为 a:hover 和 p.slogan。编译后的 CSS 代码为：


```
header h1 {
    font-size: 26px;
    font-weight: bold;
}
header h1 a {
    color: #f36;
    text-decoration: none;
}
header h1 a:hover {
    color: #63f;
    text-decoration: underline;
}
header p {
    font-size: 12px;
}
header p.slogan {
    color: #f00;
}
```

事实上，父选择器运算符 & 的作用，就是让当前的选择器和父级选择器，按照特定的规则进行连接。它有多种用途，比如创建重复的类名：

```
.button {
    &-ok {
        background-image: url("ok.png");
    }
    &-cancel {
        background-image: url("cancel.png");
    }
    &-custom {
        background-image: url("custom.png");
    }
}
```


编译后的 CSS 代码为：

```
.button-ok {
  background-image: url("ok.png");
}
.button-cancel {
  background-image: url("cancel.png");
}
.button-custom {
  background-image: url("custom.png");
}
```

在一个选择器中，& 可以重复出现多次，这样，就可以多次引用父选择器而不必重复它的名字。如：

```
.link {
  & + & {
    color: red;
  }
  & & {
    color: green;
  }
  && {
    color: blue;
  }
  &, &ish {
    color: cyan;
  }
}
```

编译后的 CSS 代码为：

```
.link + .link {
  color: red;
}
.link .link {
```



```
    color: green;
}
.link.link {
    color: blue;
}
.link, .linkish {
    color: cyan;
}
```

需要注意的是所有的父选择器，而不是仅仅重复最近的祖先选择器。请看以下例子：

```
.grand {
    .parent {
        & > & {
            color: red;
        }
        & & {
            color: green;
        }
        && {
            color: blue;
        }
        &, &ish {
            color: cyan;
        }
    }
}
```

编译后的 CSS 代码为：

```
.grand .parent > .grand .parent {
    color: red;
}
.grand .parent .grand .parent {
    color: green;
```



```
}
.grand .parent.grand .parent {
  color: blue;
}
.grand .parent,
.grand .parentish {
  color: cyan;
}
```

还可以将 & 放在一个选择器的后面，来改变选择器的顺序，将当前选择器排列到最前面。如：

```
.header {
  .menu {
    border-radius: 5px;
    .no-borderradius & {
      background-image: url('images/button-background.png');
    }
  }
}
```

选择器 .no-borderradius & 会使 .no-borderradius 置于他的父选择器 .header .menu 的前面，形成 .no-borderradius .header .menu 的结构。编译后的 CSS 代码为：

```
.header .menu {
  border-radius: 5px;
}
.no-borderradius .header .menu {
  background-image: url('images/button-background.png');
}
```

将 & 用在一个使用逗号分隔的选择器列表中，可以产生列表中所有选择器的所有可能的排列，这被称作组合爆炸。如：

```
p, a, ul, li {
  border-top: 2px dotted #366;
  & + & {
```



```
        border-top: 0;
    }
}
```

上述列表中有 4 个选择器，列表中所有选择器的所有可能的排列，将有 16 种可能。编译后的 CSS 代码为：

```
p,
a,
ul,
li {
    border-top: 2px dotted #366;
}
p + p,
p + a,
p + ul,
p + li,
a + p,
a + a,
a + ul,
a + li,
ul + p,
ul + a,
ul + ul,
ul + li,
li + p,
li + a,
li + ul,
li + li {
    border-top: 0;
}
```


2.4 运算和函数

2.4.1 运算

在我们的 CSS 中，充斥着大量数值型的 value，比如 color、padding、margin 等。在某些情况下，这些数值之间是有着一定关系的，那么我们怎样利用 Less 来组织这些数值之间的关系呢？请看以下 Less 代码：

```
@init: #111111;
@transition: @init*2;
.switchColor {
    color: @transition;
 }
```

编译后的 CSS 代码为：

```
.switchColor {
    color: #222222;
 }
```

上面的例子中，使用的是 Less 中的 operation 特性。简单的讲，就是 Less 提供了加（+）、减（-）、乘（*）、除（/）算术运算，可以对任何数值型的 value（数字、颜色、变量等）进行运算，来实现它们之间的复杂关系。

毫不夸张地说，Less 的运算已经超出了我们的期望，因为它能自动推断出颜色和数值的单位。比如像下面这样单位运算，将会输出 6px：

```
@var: 1px + 5;
```

同样，也允许使用括号：

```
width: (@var + 5) * 2;
```

并且，还可以在复合属性中进行运算：

```
border: (@width * 2) solid black;
```

2.4.2 函数

Less 中，提供了丰富的颜色和数学函数。比如，针对 color 专门提供了一系列的颜色运算函数，这些函数会先将颜色转化成 HSL 色彩空间，然后在通道级别进行操作。Less 提供了以下颜色运算函数：

```
lighten(@color, 10%);      // return a color which is 10% *lighter* than @color
darken(@color, 10%);       // return a color which is 10% *darker* than @color
```



```
saturate(@color, 10%);       // return a color 10% *more* saturated than @color
desaturate(@color, 10%);    // return a color 10% *Less* saturated than @color

fadein(@color, 10%);          // return a color 10% *Less* transparent than @color
fadeout(@color, 10%);         // return a color 10% *more* transparent than @color
fade(@color, 50%);             // return @color with 50% transparency

spin(@color, 10);              // return a color with a 10 degree larger in hue than @color
spin(@color, -10);             // return a color with a 10 degree smaller hue than @color

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

这些函数使用起来相当简单，就跟使用 JavaScript 中的函数一样。Less 文件如下：

```
@init: #f04615;
#body {
    background-color: fadein(@init, 10%);
 }
```

编译后的 CSS 代码为：

```
#body {
    background-color: #f04615;
}
```

这些都是 Less 的简单用法，当你完全掌握 Less 语法后，你会发现可以用 Less 做更多 JavaScript 的工作。当然，这里只是抛砖引玉，在第 4 章，将对 Less 函数进行详细介绍。

2.5 转义字符

有时候，当需要引入无效的 CSS 语法或 Less 不能识别的字符，就需要使用转义字符。此时，就可以在字符串前面加一个 ~，并将需要转义的字符串放在 "" 中。格式为：

```
~"anything"
```

在编译时，任何包含在 ~"anything" 中的内容，将会原封不动的输出到编译后的 CSS 文件中。如，以下 Less 代码：

```
.weird-element {
   content: ~"^//* some horrible but needed css hack";
```



```
}
```

编译后的 CSS 代码为：

```
.weird-element {
    content: ~"^//* some horrible but needed css hack";
}
```

2.6 注释

适当的注释是保证代码可读性的必要手段，Less 支持两种类型的注释：多行注释和单行注释。

1）形如 /* */ 的多行注释。如：

```
/* Hello, I'm a CSS-style comment */
.class { color: black }
```

2) 双斜线的单行注释。如：

```
// Hi, I'm a silent comment, I won't show up in your CSS
.class { color: white }
```

但需要注意的是：单行注释是不会出现在编译后的 CSS 文件中，如果是针对样式说明的注释，建议使用多行注释。

2.7 命名空间

当我们拥有了大量选择器的时候，特别是团队协同开发时，如何保证选择器之间重名问题？如果你是 java 程序员或 C++ 程序员，我猜你肯定会想到命名空间 Namespaces。

Less 也采用命名空间来对名字进行分组，来避免重名问题。如以下 Less 代码：

```
#mynamespace {
    .home {...}
    .user {...}
}
```

这样我们就定义了一个名为 mynamespace 的命名空间，如果我们要复用 user 这个选择器的时候，在需要混入这个选择器的地方，只需使用 #mynamespace > .user 就可以了。

2.8 @import 指令

Less 中，可以通过 @import 指令来导入外部文件。@import 语句可以放在代码中的任何位置，导入文件时的处理方式取决于文件的扩展名：

- 如果扩展名是 .css，文件内容将被原样输出。

- 如果是任何其他扩展名，将作为 LESS 文件被导入。
- 如果没有扩展名，将给他添加一个 .less 的扩展名，并作为 LESS 文件被导入。

例如：

```
@import "style";          // 导入 style.less
@import "style.less";     // 导入 style.less
@import "style.php";      //  style.php 作为 LESS 文件被导入
@import "style.css";      // 文件内容被原样输出
```

一个网站常常是有多个模块组成，如果只使用一个 .less 文件，编辑起来非常不便，也不利于分工协作。此时，就可以为每个模块单独创建 .less 文件，然后通过 @import 指令将它们合并成一个文件。

假如一个网站包含产品、新闻、BBS 三个模块，就可以为每个模块单独创建一个 .less 文件，分别是 product.less、news.less、bbs.less。然后，在 style.less 中，通过 @import 指令将它们合并成一个文件：

```
@import "product.less";
@import "news.less";
@import "bbs.less";
```

导入外部文件的一个常见应用场景，就是变量共享。通常是在一个 .less 文件中定义一些变量，其他文件只需导入这个 .less 文件，就可以使用这些变量。如，在 base.less 中定义 @color 变量：

```
@color: #fff;
```

然后，在 styles.less 文件中，只需使用 @import 指令导入 base.less 文件，就可以使用它的变量 @color，而不必重复定义。代码如下：

```
@import "base.less";
.myclass {
  background-color: @color;
}
```

styles.less 编译后的 CSS 代码为：

```
.myclass {
  background-color: #fff;
}
```

另外，为了在将 Less 文件编译生成 CSS 文件时，提高对外部文件的操作灵活性，还为 @import 指令提供了一些配置项。语法为：

@import (keyword) "filename";

@import 指令的配置项及其含义见表 2-1。

表 2-1 @import 指令的配置项及含义

选项	含义
reference	使用文件，但不会输出其内容（即，文件作为样式库使用）
inline	对文件的内容不作任何处理，直接输出
less	无论文件的扩展名是什么，都将作为 LESS 文件被输出
css	无论文件的扩展名是什么，都将作为 CSS 文件被输出
once	文件仅被导入一次 （这也是默认行为）
multiple	文件可以被导入多次
optional	当文件不存在时，继续编译（即，该文件是可选的）

一个@import 指令可以使用一个或多个配置项，当使用多个配置项时，各配置项之间用逗号隔开。如：

@import (optional, reference) "foo.less";

2.9 !important 关键字

在调用 mixin 时，如果在后面追加 !important 关键字，就可以将 mixin 里面的所有属性都标记为 !important。如，以下 Less 代码：

```
.mixin() {
  color: #900;
  background: #F7BE81;
}
h2 {
  .mixin() !important;
}
```

编译后的 CSS 代码为：

```
h2 {
  color: #900 !important;
  background: #F7BE81 !important;
}
```


2.10 模式匹配

Less 提供了一种机制，允许根据参数的值来改变 mixin 的行为。比如，以下代码就可以让 .mixin 根据不同的 @switch 值而表现各异：

```
.mixin (dark, @color) {
  color: darken(@color, 10%);
}
.mixin (light, @color) {
  color: lighten(@color, 10%);
}
```

此时，在调用 .mixin 时：如果 @switch 设为 light，就会得到浅色；如果 @switch 设为 dark，就会得到深色。如，以下调用：

```
@switch: light;
.class1 {
    .mixin(@switch, #888);
}
@switch2 : dark;
.class2 {
    .mixin(@switch, #666);
}
```

编译后的 CSS 代码为：

```
.class1 {
  color: #a2a2a2;
}
.class2 {
  color: #808080;
}
```

2.11 条件表达式

当需要根据表达式，而不是参数的值或数量进行匹配时，条件表达式（Guards）就显得非常有用。如果你熟悉函数式编程的话，对条件表达式也不会陌生。

为了尽可能地接近 CSS 的语言结构，Less 使用关键字 when 而不是 if/else 来实现条件

判断，因为 when 已经在 CSS 的 @media query 特性中被定义。

表达式中可以使用比较运算符、逻辑运算符、或检查函数来进行条件判断。

1、比较运算符

Less 包含五个比较运算符：<、>、<=、>=、=，可以使用比较运算符（=）来比较数字，字符串、标识符等，而其余的运算符只能与数字一起使用。如，以下 Less 代码：

```
.mixin (@a) when (@a = 20px){
    color:red;
}
.mixin (@a) when (@a < 20px){
    color:blue;
}
.mixin (@a) {
    font-size: @a;
}

h2 {
    .mixin(20px)
}
```

编译后的 CSS 代码为：

```
h2 {
  color: red;
  font-size: 20px;
}
```

此外，还可以使用关键字 true，它表示布尔真，以下两个 mixin 是相同的：

```
.truth (@a) when (@a) { ... }
.truth (@a) when (@a = true) { ... }
```

在 Less 中，只有 true 表示布尔真，关键字 true 以外的任何值，都被视为布尔假。如：

```
.class {
  .truth(40);  // 不会匹配以上任何定义
}
```

Less 中，Guards 可以是多个表达式，多个表达式之间，使用逗号‘,’分隔。如果其中

任意一个表达式的结果为 true，则匹配成功，这就相当于“或”的关系。如：

```
.mixin (@a) when (@a < -10), (@a > 10) {
    width: 100px;
}
```

上述 Guards 就表示： [-10,10] 之间的值将匹配失败，其余均匹配成功。比如以下调用，.class1 和 .class3 会匹配成功，.class2 将匹配失败：

```
.class1 {
    .mixin(-20);
}
.class2 {
    .mixin(0);
}
.class3 {
    .mixin(20);
}
```

编译后的 CSS 代码为：

```
.class1 {
  width: 100px;
}
.class3 {
  width: 100px;
}
```

2、逻辑运算符

Less 中，Guards 也可以使用 and 和 not 来进行逻辑运算。如，以下 Less 代码：

```
.mixin (@a) when (@a > 50%) and (@a > 5px){
    font-size: 14px;
}
.mixin (@a) when not (@a < 50%) and not (@a < 5px){
    font-size: 20px;
}
.mixin (@a) {
```



```
    color: @a;
}

.class1 {
    .mixin(#FF0000)
 }
.class2 {
    .mixin(#555)
}
```

编译后的 CSS 代码为：

```
.class1 {
  color: #ff0000;
}
.class2 {
  color: #555555;
}
```

3、检查函数

如果想基于值的类型、或特定的单位进行匹配，就可以使用 is* 函数。如：

```
.mixin (@a, @b: 0) when (isnumber(@a)) { ... }
.mixin (@a, @b: black) when (iscolor(@b) and (@a > 0)) { ... }
```

以下是常见的类型检查函数：

Iscolor：是否为颜色值。

Isnumber：是否为数值。

Isstring：是否为字符串。

Iskeyword：是否为关键字。

Isurl：是否为 URL 字符串。

以下是常见的单位检查函数：

Ispixel：是否为像素单位。

ispercentage：是否为百分比。

isem：是否为 em 单位。

isunit：是否为单位。

2.12 循环

在 Less 中，mixin 可以调用它自身。通过这种递归调用，再结合 Guard 表达式和模式匹配，就可以写出各种循环结构。如，使用循环来创建一个网格类：

```
.generate-columns(4);
.generate-columns(@n, @i: 1) when (@i =< @n) {
  .column-@{i} {
    width: (@i * 100% / @n);
  }
  .generate-columns(@n, (@i + 1));
}
```

编译后的 CSS 代码为：

```
.column-1 {
  width: 25%;
}
.column-2 {
  width: 50%;
}
.column-3 {
  width: 75%;
}
.column-4 {
  width: 100%;
}
```

2.13 合并

合并是 LESS 的一个特性，它允许通过指定的语法来为某个属性添加使用逗号或空格分隔的值的列表。对于文本阴影、盒阴影、背景、变换等允许使用值的列表的属性，合并非常有用。

合并的语法，就是在属性名称和冒号之间，添加一个“+”或“+_”标志：

1）当使用“+” 标志时，列表间以逗号分隔。如：

```
.mixin() {
```



```
    box-shadow +: inset 0 0 10px #555;
}
.myclass {
    .mixin();
    box-shadow +: 0 0 20px black;
}
```

编译后的 CSS 代码为：

```
.myclass {
    box-shadow: inset 0 0 10px #555, 0 0 20px black;
}
```

2）当使用“+_” 标志时，列表间以空格分隔。如：

```
.mixin() {
    transform+_: scale(2);
}
.myclass {
    .mixin();
    transform+_: rotate(15deg);
}
```

编译后的 CSS 代码为：

```
.myclass {
    transform: scale(2) rotate(15deg);
}
```

2.14 Extend

Extend 就相当于 Java 的继承，它允许一个选择器继承另一个选择器的样式。Extend 有两种语法格式。

一种是：

```
<selector>:extend(<parentSelector>) { }
```

另一种是：

```
<selector> {
&:extend(<parentSelector>);
```



```
}
```

假设有一个 .inline 的类：

```
.inline {
    color: red;
}
```

现在希望 nav ul 选择器能够让继承 .inline 类的 color 属性，就可以使用以下两种方式的任意一种来实现：

```
nav ul:extend(.inline) {
}
```

或

```
nav ul {
    &:extend(.inline);
}
```

这两种方式得到的结果完全相同，编译后的 CSS 代码为：

```
.inline,
nav ul {
  color: red;
}
```

一个选择器还可以继承多个选择器的属性，只需写多个 :extend 语句就可以了。如，.e 同时继承了 .f 和 .g 的属性：

```
.e:extend(.f) {}
.e:extend(.g) {}
```

为了方便，Less 允许仅使用一个 :extend 语句，只需在括号中提供用逗号隔开的选择器列表即可。什么两个 :extend 语句的等价写法为：

```
.e:extend(.f, .g) {}
```

3 Less 函数

3.1 Color 函数

Less 中，针对 color 专门提供了一系列的颜色运算函数，这些函数会先将颜色转化成 HSL 色彩空间，然后在通道级别进行操作。Less 提供了以下颜色运算函数：

```
lighten(@color, 10%);          // return a color which is 10% *lighter* than @color
```



```
darken(@color, 10%);          // return a color which is 10% *darker* than @color

saturate(@color, 10%);       // return a color 10% *more* saturated than @color
desaturate(@color, 10%);    // return a color 10% *Less* saturated than @color

fadein(@color, 10%);          // return a color 10% *Less* transparent than @color
fadeout(@color, 10%);         // return a color 10% *more* transparent than @color
fade(@color, 50%);             // return @color with 50% transparency

spin(@color, 10);              // return a color with a 10 degree larger in hue than @color
spin(@color, -10);            // return a color with a 10 degree smaller hue than @color

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

这些函数使用起来相当简单，就跟使用 JavaScript 中的函数一样。Less 文件如下：

```
@init: #f04615;
#body {
    background-color: fadein(@init, 10%);
 }
```

编译后的 CSS 代码为：

```
#body {
    background-color: #f04615;
}
```

Less 还提供了获取颜色值的方法：

```
hue(@color);              // returns the `hue` channel of @color
saturation(@color);    // returns the `saturation` channel of @color
lightness(@color);     // returns the 'lightness' channel of @color
```

如果你想在一种颜色的通道上创建另一种颜色，这些函数就显得那么的好用。如：

```
@new: hsl(hue(@old), 45%, 90%);
```

@new 将会保持 @old 的色调，但是具有不同的饱和度和亮度。

3.2 Math 函数

除了颜色的运算函数外，Less 还提供了一组方便的数学函数，你可以使用它们处理一些数字类型的值：

```
round(1.67);   // returns `2`
ceil(2.4);     // returns `3`
floor(2.6);    // returns `2`
```

如果你想将一个值转化为百分比，你可以使用 percentage 函数：

```
percentage(0.5);    // returns `50%`
```

4 参考资料

[1] http://lesscss.org/

[2] http://www.bootcss.com/p/Lesscss/

[3] https://www.ibm.com/developerworks/cn/web/1207_zhaoch_Lesscss/

[4] http://blog.csdn.net/column/details/less.html

[5] http://www.educity.cn/wenda/149286.html

[6] http://www.cnblogs.com/Iona/p/5302476.html

[7] http://www.cnblogs.com/fsjohnhuang/p/4187675.html

[8] http://www.cnblogs.com/hanyangecho/p/3581050.html

