网站首页

一种 动态 样式 语言.

LESS 将 CSS 赋予了动态语言的特性，如 变量， 继承，运算， 函数. LESS 既可以在 客户端 上运行 (支持IE 6+, Webkit, Firefox)，也可以借助Node.js或者Rhino在服务端运行。

Download less.js version 1.3.1

Fork me on GitHub

LESS可以这样来写CSS:

```
@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {
  box-shadow:         @style @c;
  -webkit-box-shadow: @style @c;
  -moz-box-shadow:    @style @c;
}
.box-shadow(@style, @alpha: 50%) when (isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}
.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30%) }
}
```

在引入**less.js**前先要把你的样式文件引入：

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

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。

```
// LESS

@color: #4D926F;

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

```
/* 生成的 CSS */

#header {
  color: #4D926F;
}
h2 {
  color: #4D926F;
}
```

混合

混合可以将一个定义好的class A轻松的引入到另一个class B中，从而简单实现class B继承class A中的所有属性。我们还可以带参数地调用，就像使用函数一样。

```
// LESS

.rounded-corners (@radius: 5px) {
  border-radius: @radius;
  -webkit-border-radius: @radius;
  -moz-border-radius: @radius;
}

#header {
  .rounded-corners;
}
#footer {
  .rounded-corners(10px);
}
```

```
/* 生成的 CSS */

#header {
  border-radius: 5px;
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
}
#footer {
  border-radius: 10px;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
}
```

嵌套规则

我们可以在一个选择器中嵌套另一个选择器来实现继承，这样很大程度减少了代码量，并且代码看起来更加的清晰。

```
// LESS

#header {
  h1 {
    font-size: 26px;
    font-weight: bold;
  }
  p { font-size: 12px;
    a { text-decoration: none;
      &:hover { border-width: 1px }
    }
  }
}
```

```
/* 生成的 CSS */

#header h1 {
  font-size: 26px;
  font-weight: bold;
}
#header p {
  font-size: 12px;
}
#header p a {
  text-decoration: none;
}
#header p a:hover {
  border-width: 1px;
}
```

函数 & 运算

运算提供了加，减，乘，除操作；我们可以做属性值和颜色的运算，这样就可以实现属性值之间的复杂关系。LESS中的函数一一映射了JavaScript代码，如果你愿意的话可以操作属性值。

```
// LESS

@the-border: 1px;
@base-color: #111;
@red:        #842210;

#header {
  color: @base-color * 3;
  border-left: @the-border;
  border-right: @the-border * 2;
}
#footer {
  color: @base-color + #003300;
  border-color: desaturate(@red, 10%);
}
```

```
/* 生成的 CSS */

#header {
  color: #333;
  border-left: 1px;
  border-right: 2px;
}
#footer {
  color: #114411;
  border-color: #7d2717;
}
```

在客户端使用

引入你的 .less 样式文件的时候要设置 rel 属性值为“stylesheet/less”:

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

然后点击页面顶部download按钮下载 less.js, 在<head> 中引入:

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

注意你的less样式文件一定要在引入less.js前先引入。

备注：请在服务器环境下使用！本地直接打开可能会报错！

监视模式

监视模式是客户端的一个功能，这个功能允许你当你改变样式的时候，客户端将自动刷新。

要使用它，只要在URL后面加上'`#!watch`'，然后刷新页面就可以了。另外，你也可以通过在终端运行`less.watch()`来启动监视模式。

在服务器端使用

安装

在服务器端安装 LESS 的最简单方式就是通过 npm(node 的包管理器), 像这样:

```
$ npm install less
```

如果你想下载最新稳定版本的 LESS，可以尝试像下面这样操作:

```
$ npm install less@latest
```

使用

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);
});
```

上面会输出

```
.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());
});
```

配置

你可以向解析器传递参数:

```
var parser = new(less.Parser)({
    paths: ['.', './lib'], // Specify search paths for @import directives
    filename: 'style.less' // Specify a filename, for better error messages
});

parser.parse('.class { width: 1 + 1 }', function (e, tree) {
    tree.toCSS({ compress: true }); // Minify CSS output
});
```

在命令行下使用

你可以在终端调用 LESS 解析器:

```
$ lessc styles.less
```

上面的命令会将编译后的 CSS 传递给 `stdout`, 你可以将它保存到一个文件中:

```
$ lessc styles.less > styles.css
```

如何你想将编译后的 CSS 压缩掉，那么加一个 `-x` 参数就可以了.

LESS 语法

LESS 做为 CSS 的一种形式的扩展，它并没有阉割 CSS 的功能，而是在现有的 CSS 语法上，添加了很多额外的功能，所以学习 LESS 是一件轻而易举的事情，果断学习之！

变量

很容易理解:


```
@nice-blue: #5B83AD;
@light-blue: @nice-blue + #111;

#header { color: @light-blue; }
```

输出:

```
#header { color: #6c94be; }
```

甚至可以用变量名定义为变量:

```
@fnord: "I am fnord.";
@var: 'fnord';
content: @@var;
```

解析后:

```
content: "I am fnord.";
```

请注意 LESS 中的变量为完全的‘常量’，所以只能定义一次.

混合

在 LESS 中我们可以定义一些通用的属性集为一个class，然后在另一个class中去调用这些属性. 下面有这样一个class:

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

那如果我们现在需要在其他class中引入那些通用的属性集，那么我们只需要在任何class中像下面这样调用就可以了:

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

`.bordered` class里面的属性样式都会在 `#menu a` 和 `.post a`中体现出来:

```
#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;
}
```

任何 CSS *class*, *id* 或者 元素 属性集都可以以同样的方式引入.

带参数混合

在 LESS 中，你还可以像函数一样定义一个带参数的属性集合:

```
.border-radius (@radius) {
  border-radius: @radius;
  -moz-border-radius: @radius;
  -webkit-border-radius: @radius;
}
```

然后在其他class中像这样调用它:

```
#header {
  .border-radius(4px);
}
.button {
  .border-radius(6px);
}
```

我们还可以像这样给参数设置默认值:

```
.border-radius (@radius: 5px) {
  border-radius: @radius;
  -moz-border-radius: @radius;
  -webkit-border-radius: @radius;
}
```

所以现在如果我们像这样调用它的话:

```
#header {
  .border-radius;
}
```

radius的值就会是5px.

你也可以定义不带参数属性集合,如果你想隐藏这个属性集合，不让它暴露到CSS中去，但是你还想在其他的属性集合中引用，你会发现这个方法非常的好用:

```
.wrap () {
  text-wrap: wrap;
  white-space: pre-wrap;
  white-space: -moz-pre-wrap;
  word-wrap: break-word;
}

pre { .wrap }
```

输出:

```
pre {
```

```
  text-wrap: wrap;
  white-space: pre-wrap;
  white-space: -moz-pre-wrap;
  word-wrap: break-word;
}
```

`@arguments` 变量

`@arguments`包含了所有传递进来的参数. 如果你不想单独处理每一个参数的话就可以像这样写:

```
.box-shadow (@x: 0, @y: 0, @blur: 1px, @color: #000) {
  box-shadow: @arguments;
  -moz-box-shadow: @arguments;
  -webkit-box-shadow: @arguments;
}
.box-shadow(2px, 5px);
```

将会输出:

```
  box-shadow: 2px 5px 1px #000;
  -moz-box-shadow: 2px 5px 1px #000;
  -webkit-box-shadow: 2px 5px 1px #000;
```

模式匹配和导引表达式

有些情况下，我们想根据传入的参数来改变混合的默认呈现，比如下面这个例子：

```
.mixin (@s, @color) { ... }

.class {
  .mixin(@switch, #888);
}
```

如果想让`.mixin`根据不同的`@switch`值而表现各异，如下这般设置：

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

现在，如果运行:

```
@switch: light;
```

```
.class {
  .mixin(@switch, #888);
}
```

就会得到下列CSS:

```
.class {
  color: #a2a2a2;
  display: block;
}
```

如上，.mixin就会得到传入颜色的浅色。如果@switch设为dark，就会得到深色。

具体实现如下：

第一个混合定义并未被匹配，因为它只接受dark做为首参
第二个混合定义被成功匹配，因为它只接受light
第三个混合定义被成功匹配，因为它接受任意值

只有被匹配的混合才会被使用。变量可以匹配任意的传入值，而变量以外的固定值就仅仅匹配与其相等的传入值。

我们也可以匹配多个参数：

```
.mixin (@a) {
  color: @a;
}
.mixin (@a, @b) {
  color: fade(@a, @b);
}
```

Now if we call .mixin with a single argument, we will get the output of the first definition, but if we call it with *two* arguments, we will get the second definition, namely @a faded to @b.

引导

当我们想根据表达式进行匹配，而非根据值和参数匹配时，导引就显得非常有用。如果你对函数式编程非常熟悉，那么你很可能已经使用过导引。

为了尽可能地保留CSS的可声明性，LESS通过导引混合而非if/else语句来实现条件判断，因为前者已在@media query特性中被定义。

以此例做为开始：

```
.mixin (@a) when (lightness(@a) >= 50%) {
  background-color: black;
}
.mixin (@a) when (lightness(@a) < 50%) {
  background-color: white;
}
.mixin (@a) {
  color: @a;
}
```

when关键字用以定义一个导引序列(此例只有一个导引)。接下来我们运行下列代码：

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

就会得到：

```
.class1 {
  background-color: black;
  color: #ddd;
}
.class2 {
  background-color: white;
  color: #555;
}
```

导引中可用的全部比较运算有： **> >= = =< <**。此外，关键字`true`只表示布尔真值，下面两个混合是相同的：

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

除去关键字true以外的值都被视示布尔假：

```
.class {
  .truth(40); // Will not match any of the above definitions.
}
```

导引序列使用逗号‘,’—分割，当且仅当所有条件都符合时，才会被视为匹配成功。

```
.mixin (@a) when (@a > 10), (@a < -10) { ... }
```

导引可以无参数，也可以对参数进行比较运算：

```
@media: mobile;

.mixin (@a) when (@media = mobile) { ... }
.mixin (@a) when (@media = desktop) { ... }

.max (@a, @b) when (@a > @b) { width: @a }
.max (@a, @b) when (@a < @b) { width: @b }
```

最后，如果想基于值的类型进行匹配，我们就可以使用is*函式：

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

下面就是常见的检测函式：

    iscolor
    isnumber

```
isstring
iskeyword
isurl
```

如果你想判断一个值是纯数字，还是某个单位量，可以使用下列函式：

```
ispixel
ispercentage
isem
```

最后再补充一点，在导引序列中可以使用**and**关键字实现与条件：

```
.mixin (@a) when (isnumber(@a)) and (@a > 0) { ... }
```

使用**not**关键字实现或条件

```
.mixin (@b) when not (@b > 0) { ... }
```

嵌套规则

LESS 可以让我们以嵌套的方式编写层叠样式. 让我们先看下下面这段 CSS:

```
#header { color: black; }
#header .navigation {
  font-size: 12px;
}
#header .logo {
  width: 300px;
}
#header .logo:hover {
  text-decoration: none;
}
```

在 LESS 中, 我们就可以这样写:

```
#header {
  color: black;

  .navigation {
    font-size: 12px;
  }
  .logo {
    width: 300px;
    &:hover { text-decoration: none }
  }
}
```

或者这样写:

```
#header        { color: black;
```

```
  .navigation  { font-size: 12px }
  .logo        { width: 300px;
    &:hover    { text-decoration: none }
  }
}
```

代码更简洁了，而且感觉跟DOM结构格式有点像.

注意 & 符号的使用—如果你想写串联选择器，而不是写后代选择器，就可以用到&了. 这点对伪类尤其有用如 :hover 和 :focus.

例如:

```
.bordered {
  &.float {
    float: left;
  }
  .top {
    margin: 5px;
  }
}
```

会输出

```
.bordered.float {
  float: left;
}
.bordered .top {
  margin: 5px;
}
```

运算

任何数字、颜色或者变量都可以参与运算. 来看一组例子:

```
@base: 5%;
@filler: @base * 2;
@other: @base + @filler;

color: #888 / 4;
background-color: @base-color + #111;
height: 100% / 2 + @filler;
```

LESS 的运算已经超出了我们的期望，它能够分辨出颜色和单位。如果像下面这样单位运算的话:

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

LESS 会输出 6px.

括号也同样允许使用:

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

并且可以在复合属性中进行运算:

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

Color 函数

LESS 提供了一系列的颜色运算函数. 颜色会先被转化成 *HSL* 色彩空间, 然后在通道级别操作:

```
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
```

使用起来相当简单:

```
@base: #f04615;

.class {
  color: saturate(@base, 5%);
  background-color: lighten(spin(@base, 8), 25%);
}
```

你还可以提取颜色信息:

```
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，例如:

```
@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.

Math 函数

LESS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%`
```

命名空间

有时候，你可能为了更好组织CSS或者单纯是为了更好的封装，将一些变量或者混合模块打包起来, 你可以像下面这样在#bundle中定义一些属性集之后可以重复使用:

```
#bundle {
  .button () {
    display: block;
    border: 1px solid black;
    background-color: grey;
    &:hover { background-color: white }
  }
  .tab { ... }
  .citation { ... }
}
```

你只需要在 #header a中像这样引入 .button:

```
#header a {
  color: orange;
  #bundle > .button;
}
```

作用域

LESS 中的作用域跟其他编程语言非常类似，首先会从本地查找变量或者混合模块，如果没找到的话会去父级作用域中查找，直到找到为止.

```
@var: red;

#page {
  @var: white;
  #header {
    color: @var; // white
  }
```

```
}

#footer {
  color: @var; // red
}
```

注释

CSS 形式的注释在 LESS 中是依然保留的:

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

LESS 同样也支持双斜线的注释, 但是编译成 CSS 的时候自动过滤掉:

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

Importing

你可以在main文件中通过下面的形势引入 `.less` 文件, `.less` 后缀可带可不带:

```
@import "lib.less";
@import "lib";
```

如果你想导入一个CSS文件而且不想LESS对它进行处理，只需要使用`.css`后缀就可以:

```
@import "lib.css";
```

这样LESS就会跳过它不去处理它.

字符串插值

变量可以用类似ruby和php的方式嵌入到字符串中，像`@{name}`这样的结构:

```
@base-url: "http://assets.fnord.com";
background-image: url("@{base-url}/images/bg.png");
```

避免编译

有时候我们需要输出一些不正确的CSS语法或者使用一些 LESS不认识的专有语法.

要输出这样的值我们可以在字符串前加上一个 ~, 例如:

```
.class {
  filter: ~"ms:alwaysHasItsOwnSyntax.For.Stuff()";
}
```

我们可以将要避免编译的值用 “”包含起来，输出结果为:

```
.class {
  filter: ms:alwaysHasItsOwnSyntax.For.Stuff();
}
```

JavaScript 表达式

JavaScript 表达式也可以在.less 文件中使用. 可以通过反引号的方式使用:

```
@var: `"hello".toUpperCase() + '!'`;
```

输出:

```
@var: "HELLO!";
```

注意你也可以同时使用字符串插值和避免编译:

```
@str: "hello";
@var: ~`"@{str}".toUpperCase() + '!'`;
```

输出:

```
@var: HELLO!;
```

它也可以访问JavaScript环境:

```
@height: `document.body.clientHeight`;
```

如果你想将一个JavaScript字符串解析成16进制的颜色值, 你可以使用 `color` 函数:

```
@color: color(`window.colors.baseColor`);
@darkcolor: darken(@color, 10%);
```

其他国家的LESS站点

English: http://lesscss.org（需要翻墙）
Russian: http://lesscss.ru
German:http://lesscss.de

Japanese: http://less-ja.studiomohawk.com/
Belarus: http://www.designcontest.com/show/lesscss-be

About

Alexis Sellier -LESS作者

LESS中国官网

说明：这个页面是为了给热爱Bootstrap的攻城师一个快速的查阅LessCss文档的途径。由于已经有LESS中国官网的杰出翻译了，因此直接剽窃过来啦，哈哈。由衷的感谢他们的贡献！