## 正文

下面我将简介 Less 的功能特性。

### 变量

我们常常在 CSS 中 看到同一个值重复多次，这样难易于代码维护。  
Less 中的变量十分强大，可化万物，值得一提的是，其变量是常量 ，所以只能定义一次，不能重复。  
**值变量**

/\* Less \*/

@color: #999;

@bgColor: skyblue;//不要添加引号

@width: 50%;

**#wrap** {

color: @color;

width: @width;

}

/\* 生成后的 CSS \*/

**#wrap** {

color: #999;

width: 50%;

}

以 @ 开头 定义变量，并且使用时 直接 键入 @名称。

在平时工作中，我们就可以把 常用的变量 封装到一个文件中，这样利于代码组织维护。

@lightPrimaryColor: #c5cae9;

@textPrimaryColor: #fff;

@accentColor: rgb(99, 137, 185);

@primaryTextColor: #646464;

@secondaryTextColor: #000;

@dividerColor: #b6b6b6;

@borderColor: #dadada;

**选择器变量**

让 选择器 变成 动态

/\* Less \*/

@mySelector: #wrap;

@Wrap: wrap;

@{mySelector}{ //变量名 必须使用大括号包裹

color: #999;

width: 50%;

}

.@{Wrap}{

color:#ccc;

}

**#@{Wrap}**{

color:#666;

}

/\* 生成的 CSS \*/

**#wrap**{

color: #999;

width: 50%;

}

.wrap{

color:#ccc;

}

**#wrap**{

color:#666;

}

**属性变量**

可减少代码书写量

/\* Less \*/

@borderStyle: border-style;

@Soild:solid;

**#wrap**{

@{borderStyle}: @Soild;//变量名 必须使用大括号包裹

}

/\* 生成的 CSS \*/

**#wrap**{

border-style:solid;

}

**url 变量**

项目结构改变时，修改其变量即可。

/\* Less \*/

@images: "../img";//需要加引号

**body** {

background: url("@{images}/dog.png");//变量名 必须使用大括号包裹

}

/\* 生成的 CSS \*/

**body** {

background: url("../img/dog.png");

}

**声明变量**

有点类似于 下面的 混合方法

- 结构: @name: { 属性: 值 ;};

- 使用：@name();

/\* Less \*/

@background: {background:red;};

**#main**{

@background();

}

@Rules:{

width: 200px;

height: 200px;

border: solid 1px red;

};

**#con**{

@Rules();

}

/\* 生成的 CSS \*/

**#main**{

background:red;

}

**#con**{

width: 200px;

height: 200px;

border: solid 1px red;

}

**变量运算**

不得不提的是，Less 的变量运算完全超出我的期望，十分强大。

- 加减法时 以第一个数据的单位为基准

- 乘除法时 注意单位一定要统一

/\* Less \*/

@width:300px;

@color:#222;

**#wrap**{

width:@width-20;

height:@width-20\*5;

margin:(@width-20)\*5;

color:@color\*2;

background-color:@color + #111;

}

/\* 生成的 CSS \*/

**#wrap**{

width:280px;

height:200px;

margin:1400px;

color:#444;

background-color:#333;

}

**变量作用域**

一句话理解就是：**就近原则**。

/\* Less \*/

@var: @a;

@a: 100%;

**#wrap** {

width: @var;

@a: 9%;

}

/\* 生成的 CSS \*/

**#wrap** {

width: 9%;

}

**用变量去定义变量**

/\* Less \*/

@fnord: "I am fnord.";

@var: "fnord";

**#wrap**::after{

content: @@var; //将@var替换为其值 content:@fnord;

}

/\* 生成的 CSS \*/

**#wrap**::after{

content: "I am fnord.";

}

### 嵌套

& ：代表的上一层选择器的名字，此例便是header。

/\* Less \*/

**#header**{

**&**:after{

content:"Less is more!";

}

.title{

font-weight:bold;

}

**&\_content**{//理解方式：直接把 & 替换成 #header

margin:20px;

}

}

/\* 生成的 CSS \*/

**#header**::after{

content:"Less is more!";

}

**#header** .title{ //嵌套了

font-weight:bold;

}

**#header\_content**{//没有嵌套！

margin:20px;

}

**媒体查询**

在以往的工作中，我们使用 媒体查询，都要把一个元素 分开写

**#wrap**{

width:500px;

}

@**media** screen and (max-width:768px){

**#wrap**{

width:100px;

}

}

Less 提供了一个十分便捷的方式

/\* Less \*/

**#main**{

//something...

**@media** screen{

**@media** (max-width:768px){

width:100px;

}

}

**@media** tv {

width:2000px;

}

}

/\* 生成的 CSS \*/

**@media** screen and (maxwidth:768px){

**#main**{

width:100px;

}

}

**@media** tv{

**#main**{

width:2000px;

}

}

唯一的缺点就是 每一个元素都会编译出自己 @media 声明，并不会合并。

**实战技巧**

可以借助 Less 在元素中，去定义自己的私有样式。

/\* Less \*/

**#main**{

// something..

**&**.show{

display:block;

}

}

.show{

display:none;

}

**const** main = document.getElementById("main");

main.classList.add("show");

结果：

**#main**.show{

display:block;

}

.show{

display:none; //会被覆盖。

}

混合方法

**无参数方法**

方法犹如 声明的集合，使用时 直接键入名称即可。

/\* Less \*/

.card { // 等价于 .card()

background: #f6f6f6;

-webkit-box-shadow: 0 1px 2px rgba(151, 151, 151, .58);

box-shadow: 0 1px 2px rgba(151, 151, 151, .58);

}

**#wrap**{

.card;//等价于.card();

}

/\* 生成的 CSS \*/

**#wrap**{

background: #f6f6f6;

-webkit-box-shadow: 0 1px 2px rgba(151, 151, 151, .58);

box-shadow: 0 1px 2px rgba(151, 151, 151, .58);

}

其中 .card 与 .card() 是等价的。  
个人建议，为了避免 代码混淆，应写成 :

.card(){

//something...

}

**#wrap**{

.card();

}

要点：

`.` 与 `#` 皆可作为 方法前缀。

方法后写不写 `()` 看个人习惯。

**默认参数方法**

Less 可以使用默认参数，如果 没有传参数，那么将使用默认参数。

@arguments 犹如 JS 中的 arguments 指代的是 全部参数。

传的参数中 必须带着单位。

/\* Less \*/

.border(@a:10px,@b:50px,@c:30px,@color:#000){

border:solid 1px @color;

box-shadow: @arguments;//指代的是 全部参数

}

**#main**{

.border(0px,5px,30px,red);//必须带着单位

}

**#wrap**{

.border(0px);

}

**#content**{

.border;//等价于 .border()

}

/\* 生成的 CSS \*/

**#main**{

border:solid 1px red;

box-shadow:0px,5px,30px,red;

}

**#wrap**{

border:solid 1px #000;

box-shadow: 0px 50px 30px #000;

}

**#content**{

border:solid 1px #000;

box-shadow: 10px 50px 30px #000;

}

**方法的匹配模式**

与 面向对象中的多态 很相似

/\* Less \*/

.triangle(top,@width:20px,@color:#000){

border-color:transparent transparent @color transparent ;

}

.triangle(right,@width:20px,@color:#000){

border-color:transparent @color transparent transparent ;

}

.triangle(bottom,@width:20px,@color:#000){

border-color:@color transparent transparent transparent ;

}

.triangle(left,@width:20px,@color:#000){

border-color:transparent transparent transparent @color;

}

.triangle(@\_,@width:20px,@color:#000){

border-style: solid;

border-width: @width;

}

**#main**{

.triangle(left, 50px, #999)

}

/\* 生成的 CSS \*/

**#main**{

border-color:transparent transparent transparent #999;

border-style: solid;

border-width: 50px;

}

要点

- 第一个参数 `left` 要会找到方法中匹配程度最高的，如果匹配程度相同，将全部选择，并存在着样式覆盖替换。

- 如果匹配的参数 是变量，则将会匹配，如 `@\_` 。

**方法的命名空间**

让方法更加规范

/\* Less \*/

**#card**(){

background: #723232;

.d(@w:300px){

width: @w;

**#a**(@h:300px){

height: @h;//可以使用上一层传进来的方法

width: @w;

}

}

}

**#wrap**{

**#card** > .d > **#a**(100px); // 父元素不能加 括号

}

**#main**{

**#card** .d();

}

**#con**{

//不得单独使用命名空间的方法

//.d() 如果前面没有引入命名空间 #card ，将会报错

**#card**; // 等价于 #card();

.d(20px); //必须先引入 #card

}

/\* 生成的 CSS \*/

**#wrap**{

height:100px;

width:300px;

}

**#main**{

width:300px;

}

**#con**{

width:20px;

}

要点

- 在 CSS 中`>` 选择器，选择的是 儿子元素，就是 必须与父元素 有直接血源的元素。

- 在引入命令空间时，如使用 `>` 选择器，父元素不能加 括号。

- 不得单独使用命名空间的方法 必须先引入命名空间，才能使用 其中方法。

- 子方法 可以使用上一层传进来的方法

**方法的条件筛选**

Less 没有 if else，可是它有 when

/\* Less \*/

**#card**{

// and 运算符 ，相当于 与运算 &&，必须条件全部符合才会执行

.border(@width,@color,@style) **when** (@width>100px) **and**(@color=#999){

border:@style @color @width;

}

// not 运算符，相当于 非运算 !，条件为 不符合才会执行

.background(@color) **when** **not** (@color>=#222){

background:@color;

}

// , 逗号分隔符：相当于 或运算 ||，只要有一个符合条件就会执行

.font(@size:20px) **when** (@size>50px) , (@size<100px){

font-size: @size;

}

}

**#main**{

**#card**>.border(200px,#999,solid);

**#card** .background(#111);

**#card** > .font(40px);

}

/\* 生成后的 CSS \*/

**#main**{

border:solid #999 200px;

background:#111;

font-size:40px;

}

要点

- 比较运算有： > >= = =< <。

- = 代表的是等于

- 除去关键字 true 以外的值都被视为 false：

**数量不定的参数**

如果你希望你的方法接受数量不定的参数，你可以使用... ，犹如 ES6 的扩展运算符。

/\* Less \*/

.boxShadow(...){

box-shadow: @arguments;

}

.textShadow(@a,...){

text-shadow: @arguments;

}

**#main**{

.boxShadow(1px,4px,30px,red);

.textShadow(1px,4px,30px,red);

}

/\* 生成后的 CSS \*/

**#main**{

box-shadow: 1px 4px 30px red;

text-shadow: 1px 4px 30px red;

}

**方法使用important！**

使用方法 非常简单，在方法名后 加上关键字即可。

/\* Less \*/

.border{

border: solid 1px red;

margin: 50px;

}

**#main**{

.border() !important;

}

/\* 生成后的 CSS \*/

**#main** {

border: solid 1px red **!important**;

margin: 50px **!important**;

}

**循环方法**

Less 并没有提供 for 循环功能，但这也难不倒 聪明的程序员，使用递归去实现。

下面是官网中的一个 Demo，模拟了生成栅格系统。

/\* Less \*/

.generate-columns(4);

.generate-columns(@n, @i: 1) **when** (@i =< @n) {

.column-@{i} {

width: (@i \* 100% / @n);

}

.generate-columns(@n, (@i + 1));

}

/\* 生成后的 CSS \*/

.column-1 {

width: 25%;

}

.column-2 {

width: 50%;

}

.column-3 {

width: 75%;

}

.column-4 {

width: 100%;

}

1. **属性拼接方法**

+\_ 代表的是 空格；+ 代表的是 逗号。

- 逗号

/\* Less \*/

.boxShadow() {

**box-shadow**+: **inset** **0** **0** **10px** **#555**;

}

.main {

.boxShadow();

**box-shadow**+: **0** **0** **20px** **black**;

}

/\* 生成后的 CSS \*/

.main {

box-shadow: inset 0 0 10px #555, 0 0 20px black;

}

- 空格

/\* Less \*/

.Animation() {

**transform**+**\_**: **scale**(2);

}

.main {

.Animation();

**transform**+**\_**: **rotate**(15deg);

}

/\* 生成的 CSS \*/

.main {

transform: scale(2) rotate(15deg);

}

1. **实战技巧**

下面是官网中的一个非常赞的 Demo

/\* Less \*/

.average(@x, @y) {

@average: ((@x + @y) / 2);

}

**div** {

.average(16px, 50px); // 调用 方法

padding: @average; // 使用返回值

}

/\* 生成的 CSS \*/

**div** {

padding: 33px;

}

可以说 Less 是一门优雅编程语言。

继承

extend 是 Less 的一个伪类。它可继承 所匹配声明中的全部样式。  
**extend 关键字的使用**

/\* Less \*/

.animation{

transition: all .3s ease-out;

.hide{

transform:scale(0);

}

}

**#main**{

**&**:extend(.animation);

}

**#con**{

**&**:extend(.animation .hide);

}

/\* 生成后的 CSS \*/

.animation,**#main**{

transition: all .3s ease-out;

}

.animation .hide , **#con**{

transform:scale(0);

}

**all 全局搜索替换**

使用选择器匹配到的 全部声明。

/\* Less \*/

**#main**{

width: 200px;

}

**#main** {

**&**:after {

content:"Less is good!";

}

}

**#wrap**:extend(**#main** **all**) {}

/\* 生成的 CSS \*/

**#main**,**#wrap**{

width: 200px;

}

**#main**:after, **#wrap**:after {

content: "Less is good!";

}

**减少代码的重复性**

从表面 看来，extend 与 方法 最大的差别，就是 extend 是同个选择器共用同一个声明，而 方法 是使用自己的声明，这无疑 增加了代码的重复性。

方法示例 与上面的 extend 进行对比：

/\* Less \*/

.Method{

width: 200px;

**&**:after {

content:"Less is good!";

}

}

**#main**{

.Method;

}

**#wrap**{

.Method;

}

/\* 生成的 CSS \*/

**#main**{

width: 200px;

**&**:after{

content:"Less is good!";

}

}

**#wrap**{

width: 200px;

**&**:after{

content:"Less is good!";

}

}

**要点**

*翻译官网*

* 选择器和扩展之间 是允许有空格的：pre:hover :extend(div pre).
* 可以有多个扩展: pre:hover:extend(div pre):extend(.bucket tr) - 注意这与 pre:hover:extend(div pre, .bucket tr)一样。
* 这是不可以的，扩展必须在最后 : pre:hover:extend(div pre).nth-child(odd)。
* 如果一个规则集包含多个选择器，所有选择器都可以使用extend关键字。

1. **reference**

Less 中 最强大的特性  
使用 引入的 Less 文件，但不会 编译它。

/\* Less \*/

**@import** (reference) "bootstrap.less";

**#wrap**:extend(.navbar **all**){}

翻译官网：

> 使用@**import** (reference)导入外部文件，但不会添加 把导入的文件 编译到最终输出中，只引用。

1. **once**

@import语句的默认行为。这表明相同的文件只会被导入一次，而随后的导入文件的重复代码都不会解析。

**@import** (once) "foo.less";

**@import** (once) "foo.less"; // this statement will be ignored

1. **multiple**

使用@import (multiple)允许导入多个同名文件。

/\* Less \*/

// file: foo.less

.a {

color: green;

}

// file: main.less

**@import** (multiple) "foo.less";

**@import** (multiple) "foo.less";

/\* 生成后的 CSS \*/

.a {

color: green;

}

.a {

color: green;

}

函数

1. **判断类型**
   * isnumber
   * 判断给定的值 是否 是一个数字。
   * ```**less**
   * **isnumber**(#ff0); // false
   * **isnumber**(blue); // false
   * **isnumber**("string"); // false
   * **isnumber**(1234); // true
   * **isnumber**(56px); // true
   * **isnumber**(7.8%); // true
   * **isnumber**(keyword); // false
   * **isnumber**(url(...)); // false
   * ```
   * iscolor
   * *> 判断给定的值 是否 是一个颜色。*
   * isurl
   * *> 判断给定的值 是否 是一个 url 。*
2. **颜色操作**
   * saturate
   * *> 增加一定数值的颜色饱和度。*
   * lighten
   * *> 增加一定数值的颜色亮度。*
   * darken
   * *> 降低一定数值的颜色亮度。*
   * fade
   * *> 给颜色设定一定数值的透明度。*
   * mix
   * *> 根据比例混合两种颜色。*
3. **数学函数**
   * ceil
   * *> 向上取整。*
   * floor
   * *> 向下取整。*
   * percentage
   * *> 将浮点数转换为百分比字符串。*
   * round
   * *> 四舍五入。*
   * sqrt
   * *> 计算一个数的平方根。*
   * abs
   * *> 计算数字的绝对值，原样保持单位。*
   * pow
   * *> 计算一个数的乘方。*

由于 文章 篇幅有限，所以 只能介绍一些 使用效率高的函数。

如果你想了解更多，可以去官网的[函数链接](http://lesscss.cn/functions/" \t "_blank)

其他

1. **注释**
   * /\* \*/ CSS原生注释，会被编译在 CSS 文件中。
   * /   / Less提供的一种注释，不会被编译在 CSS 文件中。
2. **避免编译**

/\* Less \*/

**#main**{

width:~'calc(300px-30px)';

}

/\* 生成后的 CSS \*/

**#main**{

width:calc(300px-30px);

}

结构： `~' 值 '`

1. **使用 JS**

因为 Less 是由 JS 编写，所以 Less 有一得天独厚的特性：代码中使用 Javascript 。

/\* Less \*/

@content:`"aaa".toUpperCase()`;

**#randomColor**{

@randomColor: ~"rgb(`Math.round(Math.random() \* 256)`,`Math.round(Math.random() \* 256)`,`Math.round(Math.random() \* 256)`)";

}

**#wrap**{

width: ~"`Math.round(Math.random() \* 100)`px";

**&**:after{

content:@content;

}

height: ~"`window.innerHeight`px";

alert:~"`alert(1)`";

**#randomColor**();

background-color: @randomColor;

}

/\* 生成后的 CSS \*/

// 弹出 1

**#wrap**{

width: 随机值（0~100）px;

height: 743px;//由电脑而异

background: 随机颜色;

}

**#wrap**::after{

content:"AAA";

}

https://segmentfault.com/a/1190000012360995#articleHeader8