[<< 正确使用less](http://www.maiziedu.com/wiki/css/use/)[less混合简介>>](http://www.maiziedu.com/wiki/css/mixins/)

**less变量**

**普通的变量**

Css默认不支持变量，当我们使用less之后就可以使用。

其定义方式是 @变量名:值 ，如下，我们定义一个变量green,设置一个颜色green,然后将网页背景设置为绿色：

@green: #801f77;  
header {  
  background: @green;  
  }

注意：由于变量只能定义一次，实际上他们就是“常量”

**作为选择器和属性名**

使用时将变量以@{变量名}的方式使用，使用例子如下：

作为选择器和属性名的变量

@kuandu:width;  
.@{kuandu}{  
  @{kuandu}:150px

}

注意：这里的变量即使选择器又是属性名，不能写错。

**作为URL**

使用时，使用””将变量的值括起来，使用时同样将变量以@{变量名}的方式使用。使用例子如下：

//作为URL的变量  
@imgurl:"https://www.baidu.com/img/";  
header{  
  background: @green url("@{imgurl}bdlogo.png");  
  height: 500px;  
}

**延迟加载**

延迟加载：变量是延迟加载的，在使用前不一定要预先说明。

意思是指：在前面使用了这个变量，但没有定义，但可以在后面给他定义出来，不影响输出。

**定义多个相同名称的变量时**

在定义一个变量两次时，只会使用最后定义的变量，Less会从当前作用域向上搜索。这个行为类似于css的定义中始终使用最后定义的属性值。使用实例如下：

//定义多个相同名称的变量时

@var: 0;  
.class {  
  @var: 1;  
    .brass {  
      @var: 2;  
      three: @var;  //这是的值是3  
      @var: 3;  
    }  
  one: @var;  //这是的值是1  
}

[<< less变量](http://www.maiziedu.com/wiki/css/variable/)[less带多个参数的混合>>](http://www.maiziedu.com/wiki/css/mix/)

**less混合简介**

**普通混合**

**什么是混合**

混合就是一种将一系列属性从一个规则集引入（“混合”）到另一个规则集的方式。

**实例**

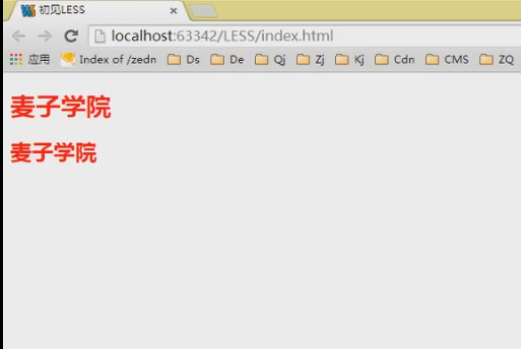
.font\_hn{  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}  
h1{  
  font-size: 28px;  
  .font\_hn;  
}  
h2{  
  font-size: 24px;  
  .font\_hn;  
}

这里对文字都进行了设置

用Koala编译器进行编译后，css文件如下：

.font\_hn {  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}  
h1 {  
  font-size: 28px;  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}  
h2 {  
  font-size: 24px;  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}

**效果图**



**不带输出的混合**

**什么是不带输出的混合**

如果你想创建一个混合集，但是却不想让他输出到你的样式中，方法：在不想输出的混合集的名字后面加上一个括号，就可以实现！

**实例**

//不带输出的混合，类名后面使用()  
.font\_hn(){  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}  
h1{  
  font-size: 28px;  
  .font\_hn;  
}  
h2{  
  font-size: 24px;  
  .font\_hn;  
}

Css文件改动如下：

h1 {  
  font-size: 28px;  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}  
h2 {  
  font-size: 24px;  
  color: red;  
  font-family: microsoft yahei, "黑体", Arial, Simsun, "Arial Unicode MS", Mingliu, Helvetica;  
}

这里就将.font\_hn隐藏掉了

**带选择器的混合**

选择规则集里面的规则，混合类型(mixins)可以包含不仅是属性，而且它们还可以含有选择器。

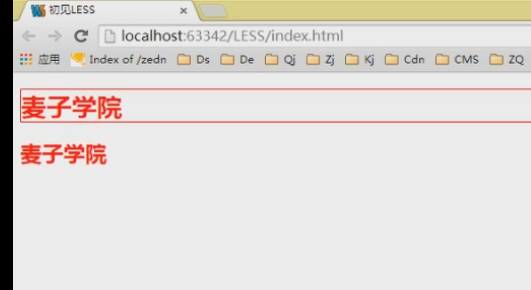
**使用实例**

//带选择器的混合  
.my-hover-mixin {  
  &:hover {  
    border: 1px solid red;  &：代表父级  
  }  
}  
button {  
  .my-hover-mixin();  
}  
h1{  
  .my-hover-mixin();  
}

其css文件为：

button:hover {  
  border: 1px solid red;  
}  
h1:hover {  
  border: 1px solid red;  
}

**效果图**



**带参数的混合**

在 LESS 中，你还可以像函数一样定义一个带参数的属性集合，然后在其他选择器中像调用它

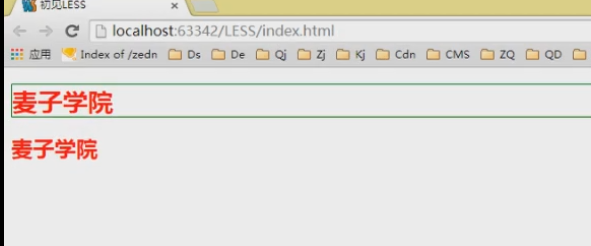
**使用例子**

//带参数的混合  
.border(@color){  
  border: 1px solid @color;  
}  
h1{  
  &:hover{  
    .border(green);  
  }  
}  
h2{  
  &:hover{  
    .border(#000);  
  }  
}

其css文件为：

h1:hover {  
  border: 1px solid #008000;  
  border: 21px #008000 #ff0000;  
}  
h2:hover {  
  border: 1px solid #000000;  
  border: 21px #000000 #ff0000;  
}

**效果图**



**带参数且有默认值的混合**

我们还可以像这样给参数设置默认值。有了默认值，我们可以不用设置属性值也能被调用。

**使用例子**

//带参数并且有默认值的混合  
.border\_you(@color:red){  
  border: 1px solid @color;   
}  
h1{  
  &:hover{  
    .border\_you();  
  }  
}  
h2{  
  &:hover{  
    .border\_you(yellow);  
  }  
}

其编译后的css文件如下：

/\*带参数并且有默认值的混合\*/  
h1:hover {  
  border: 1px solid #ff0000;  
}  
h2:hover {  
  border: 1px solid #ffff00;  
}  
/\*带多个参数的混合\*/  
.divmaizi {  
  color: 1, 2, 3;  
  margin: 10px;  
  padding: 20px;  
}

**less带多个参数的混合**

**带多个参数的混合**

**什么是多参数：**

1.一个组合可以带多个参数，参数之间可以用分号或者逗号分割。

2.但是推荐使用分号分割，因为逗号符号有两个意思。它可以理解为mixins参数分隔符或者css列表分隔符。

**官方阐述：**

1. 两个参数，并且每个参数都是逗号分隔的列表：.name(1,2,3;something.ele)

2. 三个参数，并且每个参数都包含一个数字:.name(1,2,2)

3. 使用伪造的分号创建mixin，调用的时候参数包含一个逗号分割的css列表:.name(1,2,3;)

4. 逗号分隔默认值：.name(@param1:red,blue)

使用同样的名字和同样数量的参数定义多个混合是合法的。在被调用时，LESS会应用到所有可以应用的混合上

**使用例子**

带多个参数的混合

.mixin(@color; @padding:xxx; @margin: 2) {

  color-3: @color;

  padding-3: @padding;

  margin: @margin @margin @margin @margin;

}

.divmaizi{

  .mixin(red;)

}

.divmaizi {

  .mixin(1,2,3;something, ele;132);

}

.divmaizi {

  .mixin(1,2,3);

}

.divmaizi {

  .mixin(1,2,3;);

}

其编译后的css文件如下：

/\*带多个参数的混合\*/  
.divmaizi {  
  color: 1, 2, 3;  
  margin: 10px;  
  padding: 20px;  
}

如果传参的括号里面全部都是以“，”分割，那么会依次传给各个参数值，如果传参的括号里面既有“，”又有“；”那么，会把“；”前面的看作一个整体，传给一个参数值

**定义多个具有相同名称和参数数量的混合：**

定义多个具有相同名称和参数数量的mixins是合法的。Less会使用它可以应用的属性。如果使用mixin的时候只带一个参数，比如.mixin(green)，这个属性会导致所有的mixin都会强制使用这个明确的参数。使用例子如下：

/\*定义多个具有相同名称和参数数量的混合\*/  
.mixin(@color) {  
  color-1: @color;  
}  
.mixin(@color; @padding:2) {  
  color-2: @color;  
  padding-2: @padding;  
}  
.mixin(@color; @padding; @margin: 2) {  
  color-3: @color;  
  padding-3: @padding;  
  margin: @margin @margin @margin @margin;  
}  
  
.some .selector div {  
  .mixin(#008000);  
}

其编译后的css文件如下：

.some.selector dic{

Color-1:#008000;

Color-2:#008000;

Padding-2:2;

}

**命名参数**

引用mixin时可以通过参数名称而不是参数的位置来为mixin提供参数值。任何参数都以用过它的名称来使用，这样就不必按照任意特定的顺序来使用参数

**使用例子**

/\*命名参数\*/  
.mixin(@color: black; @margin: 10px; @padding: 20px) {  
  color: @color;  
  margin: @margin;  
  padding: @padding;  
}  
  
.class1 {  
  .mixin(@margin: 20px; @color: #33acfe);  
}  
.class2 {  
  .mixin(#efca44; @padding: 40px);  
}  
.class3{  
  .mixin(@padding: 80px;)  
}

其编译后的css文件如下:

.class1 {  
  color: #33acfe;  
  margin: 20px;  
  padding: 20px;  
}  
.class2 {  
  color: #efca44;  
  margin: 10px;  
  padding: 40px;  
}  
.class3 {  
  color: #000000;  
  margin: 10px;  
  padding: 80px;  
}

**@arguments变量**

@arguments代表所有的可变参数

**注意事项：**

1. @arguments代表所有可变参数，参数的先后顺序就是你的（）括号内的参数的先后顺序

2. 在使用的赋值，值的位置和个数也是一一对应的，只有一个值，把值赋值给第一个，两个值，赋值给第一个和第二个，三个值赋值给第三个……以此类推，但是需要注意的是假如我想给第一个和第三个赋值，你不能写（值1，，值3），必须把原来的默认值写上去！

**使用例子**

/\*@arguments;\*/  
.border(@x:solid,@c:red){  
  border: 21px @arguments;  
}  
.div1{  
  .border(solid);  
}

其编译后的css文件为：

.div1 {  
    border: 21px solid #ff0000;  
}

**匹配模式**

传值的时候定义一个字符，在使用的时候使用哪个字符，就调用那天规则。

**实例**

.border(all,@w: 5px){  
  border-radius: @w;  
}  
.border(t\_l,@w:5px){  
  border-top-left-radius: @w;  
}  
.border(t\_r,@w:5px){  
  border-top-right-radius: @w;  
}  
.border(b-l,@w:5px){  
  border-bottom-left-radius: @w;  
}  
.border(b-r,@w:5px){  
  border-bottom-right-radius: @w;  
}  
  
footer{  
  .border(t\_l,10px);  
  .border(b-r,10px);  
  background: #33acfe;  
}

其编译后的css文件如下：

footer {  
  border: 21px t\_l 10px;  
  border-top-left-radius: 10px;  
  border: 21px b-r 10px;  
  border-bottom-right-radius: 10px;  
  background: #33acfe;  
}

**混合的返回值**

在使用混合时，有时会用到一些运算，这时就有返回值。使用实例如下：

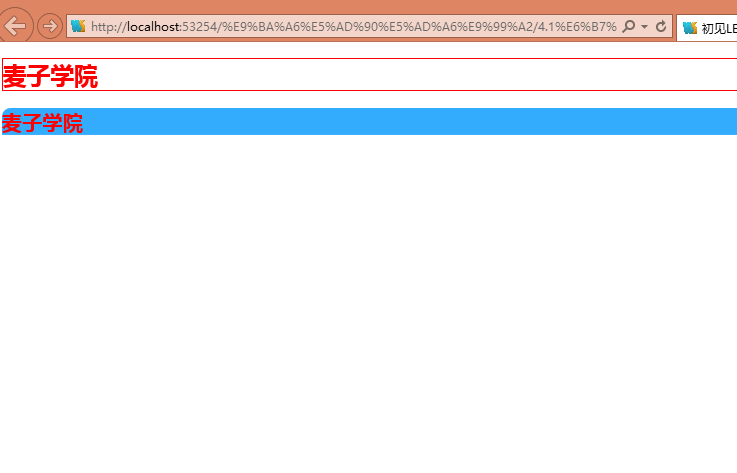
//混合的返回值

.average(@x, @y) {  
  @average: ((@x + @y) / 2);  
  @he:(@x + @y);  
}  
  
div {  
  .average(16px, 50px);  
  padding: @average;  
  margin: @he;  
}

其编译后的css文件如下：

div {  
  padding: 33px;  
  margin: 66px;  
}

**最终效果**



**less嵌套规则**

**嵌套规则**

他模仿了HTML的结构，让我们的css代码更加简洁明了清晰。下面是传统的css写法和less写法的对比：

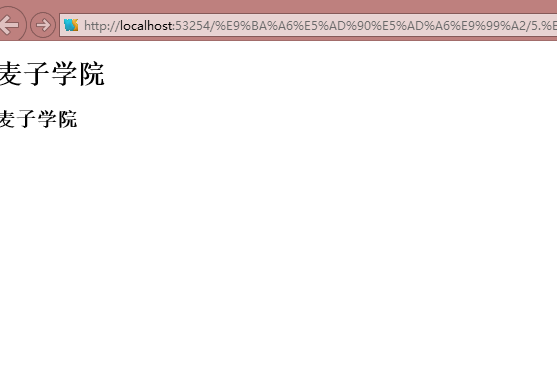
**传统写法**

header{  
  width: 960px;  
}  
header h1 {  
  font-size: 18px;  
  color: green;  
}

**less写法**

header{  
  width: 960px;  
  h1{  
    font-size: 18px;  
    color: green;  
  }  
  .logo{  
    width: 300px;  
    height: 150px;  
    background: darkred;  
     }  
}

**效果图**



从这里可以看出，无论是css还是less在编写的时候也要规范的书写

**父元素选择器**

&表示当前选择器的所有父选择器。下面是两种写法的对比。

**Css写法如下**

header .logo{  
 width: 300px;  
  height: 150px;  
  background: darkred;  
}  
header .logo:hover{  
  background: forestgreen;  
}

**less写法如下**

.logo{  
    width: 300px;  
    height: 150px;  
    background: darkred;

           &:hover{  
           background: forestgreen;

     }  
}

**改变选择器的顺序**

将&放到当前选择器之后，就会将当前选择器插入到所有的父选择器之前。

两种写法的对比：

① .a{  
  .b{  
    .c{  
      color: 123;  
    }  
  }  
}

② .a{  
  .b{  
    .c&{  
      color: 123;  
    }  
  }  
}

它们的编译结果是：

① .a .b .c {  
  color: 123;  
}

②.c.a .b {  
  color: 123;  
}

组合使用生成所有可能的选择器列表，例子如下：

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

其css编译后文件为：

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;  
}

这里的意思是指：将所有可能的组合都列出来。

**less运算**

运算说明：任何数值，颜色和变量都可以进行运算。

**数值运算**

Less会为你自动推断数值的单位，所以你不必每一个值都加上单位。

注意：运算符与值之间必须以空格分开，涉及优先级时以()进行优先级运算

**实例**

.wp{  
  margin: 0 auto;  
  background: forestgreen;  
  width: 450px + 450;  
  height: 400 + 400px;  
}

从代码中可以看出，只需要在其中的一个数值上加上单位，其他单位由less自动加上。

涉及到优先级时，运算思路：结合上方代码 wp的宽度=500px-50px之后\*2之后得到的900px的值。

**代码如下**

.wp{  
  margin: 0 auto;  
  width: (550 - 50)\*2 + 24px;  
  height: 400 + 400px;  
  background:#ff0000 - 55;  //#000021   c8c8c8  
}

**颜色运算**

Less在运算时，先将颜色值转换为rgb模式，然后再转换为16进制的颜色值并且返回

**实例**

.content{   Background:#000000 + 21;  }

**Css文件编译结果**

.content{  background:#212121;  }

注意：既然是转换为rgb模式，我们知道rgb的取值范围是2-255，所以我们计算的时候不能超过这个区间，超过后默认使用最大值255计算

**less函数**

这里我们初步的了解一下函数，在最后进行详解。Less提供了许多用于转换颜色，处理字符和进行算术运算的函数，这些函数使用起来非常方便。

**最常见的rgb函数实例**

.bgcolor{ background:rgb(0,133,0); }

Css结果：

.bgcolor{ background:#008500; }

**命名空间**

将一些需要混合组合在一起，可以通过嵌套多层id或者class来实现这就是命名空间

**使用实例**

#bgcolor(){  
  background: #ffffff;  
  .a{  
    color: #888888;  
    &:hover{  
      color: #ff6600;  
    }  
    .b{  
      background: #ff0000;  
     }  
   }  
 }

.bgcolor{

 Background:#fdfee0;

 #bgcolor>.a;

}

.bgcolor2{

#bgcolor>.a>.b;

}

其编译结果如下：

.bgcolor1 {  
  background: #fdfee0;  
  color: #888888;  
}  
.bgcolor1:hover {  
  color: #ff6600;  
}  
.bgcolor1 .b {  
  background: #ff0000;  
}  
.bgcolor2 {  
  background: #ff0000;  
}

**less作用域**

Less中的作用域与编程语言中的作用域概念非常相似。首先会在局部查找变量和混合，如果没找到，编译器就会在父作用域中查找，依次类推

**使用实例**

@clolor:#ffffff;  
  
.bgcolor{  
  width: 50px;  
  a{  
    color: @clolor;  
  }  
}  
  
@clolor:#ff0000;

**编译后的css文件**

.bgcolor {  
  width: 50px;  
}  
.bgcolor a {  
  color: #ff0000;  
}

**less引入和关键字**

**引入**

你可以引入一个或多个less文件，然后这个文件中的所有变量都可以在当前的less项目中使用！

**一些常用的引入**

@import (reference) "main.less";  //引用LESS文件，但是不输出  
@import (inline) "main.less";  //引用LESS文件，把样式输出到css，但是不能在less中进行操作  
@import (once) "main.less";  //（默认）引用LESS文件，并输出  
@import (less) "index.css";  //无论是什么格式的文件，都把他作为LESS文件操作  
@import (css) "main.less";  //无论是什么格式的文件，都把他作为CSS文件操作

**关键字**

在调用的混合集后面追加！important关键字，可以使混合集里面的所有属性都继承！important。

**使用实例**

.foo (@bg: #f5f5f5, @color: #900) {  
  background: @bg;  
  color: @color;  
  font-size: 16px;  
  font-weight: 900;  
}  
  
.unimportant {  
  .foo();  
}  
.important {  
  .foo() !important;  
}

**下面是css文件**

.unimportant {  
  background: #f5f5f5;  
  color: #990000;  
  font-size: 16px;  
  font-weight: 900;  
}  
.important {  
  background: #f5f5f5 !important;  
  color: #990000 !important;  
  font-size: 16px !important;  
  font-weight: 900 !important;  
}

**less条件表达式**

条件表达式其实就是我们在less中使用比较运算符或者表达式的判断来输入我们的值，根据不同的条件来输出不同的值。

**比较运算符有以下几种**

>,>=,=,=<,<,true。

**使用实例**

.mixin (@a) when (lightness(@a) >= 50%) {   //255/2=127.5  
  background-color: black;  
}  
.mixin (@a) when (lightness(@a) < 50%) {  
  background-color: white;  
}  
.mixin (@a) {  
  color: @a;  
}  
.class1 { .mixin(#ddd) }

.class2 { .mixin(#555) }

**下面是css文件：**

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

**类型检查函数**

类型检查函数也就是所得的值通过函数去判断是那个类型的值

**如下是一些函数**

Iscolor(是否为颜色值)   Isnumber(是否为数字)   isstring(是否为字符)   iskeyword(是否为关键字)   isurl(是否为url)

**单位检查函数**

检查一个值除了数字是否是一个特定的单位

**如下**

ispixel(是否为像素)   ispercentage(是否是百分比)   isem   isunit

**less循环和合并属性**

**循环**

在less中，混合可以调用它自身。这样，当一个混合递归调用自己，再结合Guard表达式和模式匹配这两个特性，就可以写出循环结构。

**以下是我们的使用实例：**

.loop(@counter) when (@counter > 0) {  
  .loop((@counter - 1));    // 递归调用自身 4 3 2 1 0  
  width: (10px \* @counter); // 每次调用时产生的样式代码 50px 40px 30px 20px  10px  
}  
  
div {  
  .loop(5); // 调用循环  
}

**其编译结果如下：**

Div{

Width:10px

Width:20px;

Width:30px;

Width:40px;

Width:50px;;

}

**合并属性**

将我们需要合并的属性合并在一起，合并属性的方式有两种：

1. **在需要合并的属性的：（冒号）的前面加上+就可以完成合并，合并以,分割属性**

实例如下：

+ 合并以后，以逗号分割属性值  
.mixin() {  
  box-shadow+: inset 0 0 10px #555 ;  
}  
.myclass {  
  .mixin();  
  box-shadow+: 0 0 20px black;  
}

其css文件如下：

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

**② “+\_”空格分割所合并的属性值**

实例如下：

+\_ 合并以后，以空格分割属性值  
.a(){  
  background+:#f60;  
  background+\_:url("/sss.jod") ;  
  background+:no-repeat;  
  background+\_:center;  
}  
.myclass {  
  .a()  
}

其编译后的css文件如下：

.myclass {  
  background: #f60 url("/sss.jod"), no-repeat center;  
}

**less其它函数**

**函数库**

函数库包含了许多函数，我们要讲的是以下九种：

① 其它函数；

② 字符串函数；

③ 长度相关函数；

④ 数学函数；

⑤ 类型函数；

⑥ 颜色值定义函数；

⑦ 颜色值通道提取函数；

⑧ 颜色值运算函数；

⑨ 颜色混合函数

**其它函数**

在这里面就包含了五个函数如下：

**Color()函数**

解析颜色，将代表颜色的字符串转换为颜色值。以下是它的一个实例：  body{ background:color(“#f60”) }

Css文件如下：

Body { background:#ff6600; }

**Convert()**

将数字从一种类型转换到另一种类型。实例如下：

body{ width:convert(20cm,px); }

Css文件如下:

body{ width:755.90551181px; }

还可以：

Convert(20s,ms);

Convert(20deg,rad);等

**Data-uri()**

将一个资源内嵌到样式文件，如果开启了ieconmpat选项，而且资源文件的体积过大，或者是在浏览器中使用，则会使用url()进行回退。如果没有指定MIME，则Node.js会使用MIME包来决定正确的MIME。实例如下：

body{ background:data-uri(‘arr.jpg’); } 转换成了base64格式的图片

**Default()**

只能边界条件使用，没有匹配到其他自定义函数（mixin）的时候返回true，否则返回false。实例如下：

.x(1) {  x:11 }  
.x(2) { y:22 }  
.x(@x) when (default()) {z:@x}  
.div1{  
  .x(1)  
}  
.div2{  
  .x(123)  
}

Css编译结果如下：

.Div1{ x:11; }

.div2{ z:123;}

**Unit()**

移除或者改变属性值的单位。实例如下：

div{  
  width: unit(100px);

width:unit(100px,cm); // width:100cm;

width:unit(100,px); // width:100px;  
}

css编译结果如下：

div {  
  width: 100;  
}

**less字符串函数**

和字符串函数相关的有4个，如下：

**escape()函数**

将输入字符串的url特殊字符进行编码处理

不转义的编码：, , / , ? , @ , & , + , ‘ , ~ . ! , $

转义的编码：# , ^ ( , ) { , } , | , ; . , : , > , < , = , [ , ]

**实例如下：**

Div{ d:escape(‘123=abc’); }

Css编译结果如下：

Div{ s;123%3Aabc; }

**e()函数**

css转义，用~“值”符号代替。

**实例如下：**

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

Css就会原文输出，也就是避免编译；

**%()函数**

函数%(string, arguments…)格式化一个字符串

在这里面有6个值，也就是占位符：a A d D s S，它们的含义也不同

D , d, a, A能被任何类型参数替换（颜色值，数字，转义值，表达式……），如果你在字符串中结合使用，整个字符串参数都会替换进去，包括他的引号，然后引号会被替换到字符串参数的原有位置，也许会被转义或者还是不变，取决于占位符是大写字母还是小写字母。

S , s 能被除了颜色值以外的任何类型参数替换，如果你在字符串中结合使用，只会替换程字符串参数的值。而字符串参数一号都被忽略

具体的使用方法需自己慢慢去摸索

**replace()函数**

用另一个字符串替换文本

**实例如下：**

Replace(“HELLO,maizi”,”maizi”,”LESS”);

输出： “HELLO,LESS”;

在这里第二个引号里的maizi就是需要替换的文本，less就是替换的内容。

**less长度函数和数学函数**

**长度相关函数**

**1）length()函数**

返回集合中的条数。

例子：length(1px solid #0080ff);

输出：3

**2）extract()函数**

返回集合中指定索引的值。

例子：@list:”banana”,”tomato”,”peach”;

N:extract(@list,1);

输出：N:banana;

**数学函数**

1).向上取整：ceil()

2).向下取整：floor()

3).将浮点数转换为百分比：percentage()

4).取整和四舍五入：round()

5).计算一个数的平方根，原样保持单位：sqrt()

6).计算数字的绝对值，原样保持单位：abs()

7).正弦函数:sin()

8).反正弦函数:asin()

9).余弦函数：cos()

10).反余弦函数：acos()

11).正切函数：tan()

12).反正切函数：atan()

13).返回π(pi)：pi()

14).乘方运算：pow()

15).取余运算：mod()

16).最小值运算：min()

17).最大值运算：max()

**下面是这些函数的实例：**

div{  
  width: ceil(2.9999999px);   //1.ceil()函数  向上取整  
  width: floor(2.9999999px);   //1.floor()函数  向下取整  
  width:percentage( 0.5px);  将浮点数转换为百分比  
  
//  取整和四舍五入  
  width:4.5px ;  
  width:round(4.5px) ;  
  
  width:4.4px ;  
  width:round(4.4px) ;  
  
  计算一个数的平方根，原样保持单位。  
  width: sqrt(16px);  
  width: sqrt(9px);  
  
  计算数字的绝对值，原样保持单位。  
  top: -999px;  
  top: abs(-999px);  
  
  width: sin(1);  //1弧度角的正弦值  
  width: sin(1deg);//1角度角的正弦值  
  width: sin(1grad); //1百分度角的正弦值  
  
//  反正弦值  
  width: asin(-0.84147098);  
  width: asin(0);  
  width: asin(2);  
  
  
    width: cos(1);  //1弧度角的余弦值  
    width: cos(1deg);//1角度角的余弦值  
    width: cos(1grad); //1百分度角的余弦值  
  
  width: tan(1);  //1弧度角的正切值  
  width: tan(1deg);//1角度角的正切值  
  width: tan(1grad); //1百分度角的正切值  
  
  PI  
  width: pi();  
  
//  乘方运算  
  width: pow(2px,2);  
  width: pow(3px,2);  
  width: pow(4px,2);  
  width: pow(25px,0.5);  
  
//  mod()取余  
  width: mod(3px,2);  
//  min（）最小

//  max（）最大  
  width: min(3px,2px,1px);  
  width: max(3px,2px,1px);  
}

这个需要自己多去练习，熟能生巧，数学函数和一些基本的语法类似，所以说并不是很难理解。

**less类型函数**

**类型相关函数一共有9个**

1）.如果一个值是一个数字，返回真（true），否者返回（false）：isnumber()

2).如果一个只是一个字符串，返回真（true），否者返回（false）：isstring()

3).如果一个值是一个颜色，返回真（true），否者返回（false）：iscolor()

4).如果一个值是一个关键字，返回真（true），否者返回（false）：iskeyword()

5).如果一个值是一个url地址，返回真（true），否者返回（false）：isurl()

6).如果一个值是带像素长度单位的数字，返回真（true），否者返回（false）：ispixel()

7).如果一个值是带em长度单位的数字，返回真（true），否者返回（false）：isem()

8).如果一个值是带百分比单位的数字，返回真（true），否者返回（false）：ispercentage()

9).如果一个值是带指定单位的数字，返回真（true），否者返回（false）：isunit()

这九个函数的用法就和前面条件表达式的时候讲的一样，带进去用就行了，需要多多练习。

**一些例子**

//如果一个值是一个数字，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (isnumber(@x)) {  
  x:@x  
}  
div{  
.m(123);  
.m(ABC);  
}  
  
//如果一个值是一个字符串，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (isstring(@x)) {  
  x:@x  
}  
div{  
.m(123);  
.m("ABC");  
}  
  
//如果一个值是一个颜色，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (iscolor(@x)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m("ABC");  
  .m(red);  
}  
  
//如果一个值是一个关键字，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (iskeyword(@x)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m("ABC");  
  .m(red);  
  .m(ABC);  
}  
  
//如果一个值是一个url地址，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (isurl(@x)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m("ABC");  
  .m(red);  
  .m(ABC);  
  .m(url(arr.jpg));  
}  
  
//如果一个值是带像素长度单位的数字，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (ispixel(@x)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m("ABC");  
  .m(red);  
  .m(ABC);  
  .m(url(arr.jpg));  
  .m(220px);  
  .m(220cm);  
}  
  
//如果一个值是带em长度单位的数字，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (isem(@x)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m("ABC");  
  .m(red);  
  .m(ABC);  
  .m(url(arr.jpg));  
  .m(220px);  
  .m(240em);  
}  
  
//如果一个值是带百分比单位的数字，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (ispercentage(@x)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m("ABC");  
  .m(red);  
  .m(ABC);  
  .m(url(arr.jpg));  
  .m(220px);  
  .m(240em);  
  .m(260%);  
}  
  
//如果一个值是带指定单位的数字，返回'真(true)',否则返回'假(false)'.  
.m(@x) when (isunit(@x,em)) {  
  x:@x  
}  
div{  
  .m(123);  
  .m(220px);  
  .m(240em);  
  .m(280em);  
  .m(290em);  
  .m(260%);  
}

**less颜色值定义函数**

**颜色值定义函数共有7个**

1）.通过十进制红色，绿色，蓝色三种值（RGB）创建不透明的颜色对象：rgb()

2).通过十进制红色，绿色，蓝色，以及alpha四种植（RGBA）创建带alpha透明的颜色对象：rgba()

3).创建格式为#AARRGGBB的十六进制（hex representation）颜色（注意不是#RRGGBBAA ! ）：argb()

4). 通过色相(hue)，饱和度（saturation），亮度（lightness）三种值创建不透明的颜色对象：hls()

5).通过色相(hue)，饱和度（saturation），亮度（lightness），以及alpha四种值（HSLA）创建透明的颜色对象：hsla()

6). 通过色相(hue)，饱和度（saturation），色调（value）三种值（HSV）创建不透明的颜色对象：hsv()

7).通过色相(hue)，饱和度（saturation），色调（value），以及alpha四种值创建透明的颜色对象：hsva()

**一些实例**

//通过十进制红色，绿色，蓝色三种值 (RGB) 创建不透明的颜色对象。  
div{  
  background: rgb(255,0,0);  
  background: rgb(100%,0%,0%);  
}  
  
//通过十进制红色，绿色，蓝色，以及 alpha 四种值 (RGBA) 创建带alpha透明的颜色对象。  
div{  
  background: rgba(255,0,0,0.5);  
  background: rgba(100%,0%,0%,0.5);  
}  
  
//创建格式为 #AARRGGBB 的十六进制 (hex representation) 颜色 (注意不是 #RRGGBBAA ！)。  
div{  
  background: argb(rgba(255,0,0,0.5));  
  background: argb(rgba(100%,0%,0%,0.5));  
}  
  
//通过色相 (hue)，饱和度 (saturation)，亮度 (lightness) 三种值 (HSL) 创建不透明的颜色对象。  
div{  
  background: hsl(90,100%,50%);  
}  
  
//通过色相 (hue)，饱和度 (saturation)，亮度 (lightness)，以及 alpha 四种值 (HSLA) 创建透明的颜色对象。  
div{  
  background: hsla(90,100%,50%,0.5);  
}  
  
//通过色相 (hue)，饱和度 (saturation)，色调 (value) 三种值 (HSV) 创建不透明的颜色对象。  
div{  
  background: hsv(90,100%,50%);  
}  
  
//通过色相 (hue)，饱和度 (saturation)，色调 (value)，以及 alpha 四种值 (HSVA) 创建透明的颜色对象。  
div{  
  background: hsva(90,100%,50%,8%);  
}

这些代码都可以直接用Koala直接运行出结果，希望大家多下去练习。

**less颜色值通道提取函数**

**颜色值通道提取函数一共有12个**

1）.从HSL色彩空间中提取颜色对象的色相值：hue()

2). 从HSL色彩空间中提取颜色对象的饱和度值：saturation()

3). 从HSL色彩空间中提取颜色对象的亮度值:lightness()

4). 从HSV色彩空间中提取颜色对象的色相值：hsvhue()

5). 从HSV色彩空间中提取颜色对象的饱和度值:hsvsaturation()

6). 从色彩空间中提取颜色对象的色调值：hsvvalue()

7). 提取颜色对象的红色值：red()

8). 提取颜色对象的绿色值：gree()

9). 提取颜色对象的蓝色值：blue()

10). 提取颜色对象的透明值：alpha()

11).计算颜色对象的luma的值（亮度的百分比表示方法）：lima()

12).计算没有伽玛校正的亮度值：luminance()

这些函数就是用来提取色彩空间某一对象的值

**实例**

div{  
  
  hue()色相值  
z-index: hue(hsl(90,100%,50%));  90  
saturation()饱和度  
z-index: saturation(hsl(90,80%,50%));  80%  
lightness()亮度值  
z-index: lightness(hsl(90,100%,100%));  100%  
  
  hsv(90,100%,50%)  
  z-index:hsvhue( hsv(90,100%,50%));  函数90  
  z-index:hsvsaturation( hsv(90,100%,50%)); 函数100%  
  z-index:hsvvalue( hsv(90,100%,50%)); 函数50%  
  
rgba(29,199,29,80%)  
  //提取红色  
z-index: red(rgba(29,199,150,80%));  29  
    //提取绿色  
z-index: green(rgba(29,199,150,80%));  199  
    //提取蓝色  
z-index: blue(rgba(29,199,150,80%));  29  
    //提取透明度  
z-index: alpha(rgba(29,199,150,80%));  0.8  
  
  //计算颜色对象luma的值（亮度的百分比表示法）。  
z-index:luma(rgb(100,200,30));  
  //计算没有伽玛校正的亮度值  
  z-index:luminance(rgb(100,200,30));  
  
}

**less颜色值运算函数**

**颜色值运算函数一共有11个**

1).增加一定数值的颜色饱和度：saturate()

2).降低一定数值的颜色饱和度：desaturate()

3).增加一定数值的颜色亮度：lighten()

4).降低一定数值的颜色亮度：darken()

5).降低颜色的透明度（或增加不透明度），令其更不透明：fadein()

6).增加颜色的透明度（或降低不透明的），令其更透明：fadeout()

7).给颜色（包括不透明的颜色）设定一定数值的透明度：fade()

8）.任意方向旋转颜色的色相角度（hue angle）:spin()

9).根据比例混合两种颜色，包括计算不透明度：mix()

10).完全移除颜色的饱和度，与desaturate(@colo,100%)函数效果相同：greyscale()

11).选着两种颜色相比较，得出那种颜色的对比度最大就倾向于对比度最大的颜色：contrast()

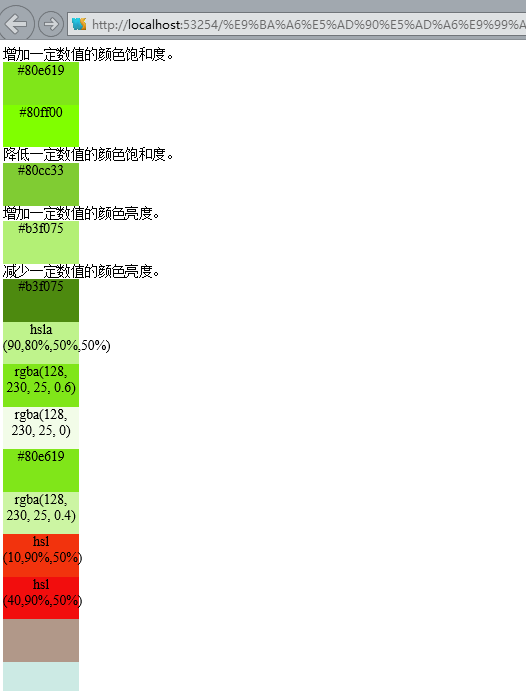
**实例**

body{  
  c:hsl(90,80%,50%);  
  c:saturate(hsl(90,80%,50%),20%);  
}  
div{  
  width: 90px;  
  height: 50px;  
  font-size: 16px;  
  text-align: center;  
}  
.ys1{  
  background: hsl(90,80%,50%);  
}  
.ys2{  
  background: saturate(hsl(90,80%,50%),20%);  
}  
.ys3{  
  background: desaturate(hsl(90,80%,50%),20%);  
}  
.ys4{  
  background: lighten(hsl(90,80%,50%),20%);  
}  
.ys5{  
  background: darken(hsl(90,80%,50%),20%);  
}  
.ys66{  
  background:hsla(90,80%,50%,50%);  
}  
.ys6{  
  background: fadein(hsla(90,80%,50%,50%),50%);  
}  
.ys7{  
  background: fadeout(hsla(90,80%,50%,50%),40%);  
}  
.ys8{  
  background: hsl(90,80%,50%);  
}  
.ys9{  
  background: fade(hsl(90,80%,50%),40%);  
}  
.ys10{  
  background: hsl(10,90%,50%);  
}  
.ys11{  
  background: spin(hsl(0,90%,50%),360);  
}  
  
.ys12{  
  background: rgba(100,50,20,0.5);  
}  
.ys13{  
  background: rgba(0,150,120,0.2);  
}  
.ys14{  
  background: mix(rgba(100,50,20,0.5),rgba(0,150,120,0.2));  
}  
.ys15{  
  background: hsl(90,100%,50%);  
}  
.ys16{  
  background:contrast(hsl(90,100%,50%),#000000,#ffffff,100%)  
}

**下面是css文件的编译结果**：

body {  
  c: #80e619;  
  c: #80ff00;  
}  
div {  
  width: 90px;  
  height: 50px;  
  font-size: 16px;  
  text-align: center;  
}  
.ys1 {  
  background: #80e619;  
}  
.ys2 {  
  background: #80ff00;  
}  
.ys3 {  
  background: #80cc33;  
}  
.ys4 {  
  background: #b3f075;  
}  
.ys5 {  
  background: #4d8a0f;  
}  
.ys66 {  
  background: rgba(128, 230, 25, 0.5);  
}  
.ys6 {  
  background: #80e619;  
}  
.ys7 {  
  background: rgba(128, 230, 25, 0.1);  
}  
.ys8 {  
  background: #80e619;  
}  
.ys9 {  
  background: rgba(128, 230, 25, 0.4);  
}  
.ys10 {  
  background: #f2330d;  
}  
.ys11 {  
  background: #f20d0d;  
}  
.ys12 {  
  background: rgba(100, 50, 20, 0.5);  
}  
.ys13 {  
  background: rgba(0, 150, 120, 0.2);  
}  
.ys14 {  
  background: rgba(65, 85, 55, 0.35);  
}  
.ys15 {  
  background: #80ff00;  
}  
.ys16 {  
  background: #ffffff;  
}

**下面是我们的效果图：**



但就颜色值运算来说，我们并不怎么用，但还是需要了解，希望大家多多练习。

**less颜色混和函数**

**颜色混和函数一共有9个**

1).分别将两种颜色的红绿蓝（RGB）三种值做乘法运算，然后在除以255，输出结果是更深的颜色（译注：对应Photoshop中的“变暗/正片叠底”）：multiply()

2).与multiply()函数效果相反，输出结果是更亮的颜色。（对应Photoshop中的“变亮/滤色”）：screen()

3).结合multiply()与screen()两个函数的效果，令浅的颜色变得更浅，深的颜色变得更深。（对应Photoshop中的“叠加”）注意：输出结果由第一个颜色参数决定：overlay()

4).与overlay()函数效果相似，只是当纯黑色或纯白色作为参数时输出结果不会是纯黑色或纯白色。（对应Photoshop中的“柔光 ”）:softlight()

5).与overlay()函数效果相似，不过由第二个颜色参数决定输出颜色的亮度或黑度，而不是第一个颜色参数决定。（对应Photoshop中的：强光/亮光.线性光/点光）：hardlight()

6).从第一个颜色值中间去第二个（分别计算RGB三种颜色值），输出结果是更深的颜色（对应Photoshop中的“差值/排除”）:differene()

7).效果与difference()函数效果相似，只是输出结果差别更小（lowr contrast）(对应Photoshop中的“差值/排除”)：exclusion()

8).分别对RGB的三种颜色值取平均值，然后输出结果：average()

9).与difference()函数效果相反，输出结果是更亮的颜色。请注意：效果相反不代表做加法运算：negation()

**下面是一些使用例子**

有些长，建议复制到自己的工具里面去看

.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: multiply(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: multiply(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: multiply(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: screen(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: screen(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: screen(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: overlay(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: overlay(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: overlay(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: softlight(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: softlight(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: softlight(#ff6600,#fff);  
}  
  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: hardlight(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: hardlight(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: hardlight(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: difference(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: difference(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: difference(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: exclusion(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: exclusion(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: exclusion(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: average(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: average(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: average(#ff6600,#fff);  
}  
  
.ys1{  
  background:#ff6600 ;  
}  
.ys2{  
  background: #000000;  
}  
.ys3{  
  background: negation(#ff6600,#000000);  
}  
.ys4{  
  background:#ff6600 ;  
}  
.ys5{  
  background: #333;  
}  
.ys6{  
  background: negation(#ff6600,#333);  
}  
.ys7{  
  background:#ff6600 ;  
}  
.ys8{  
  background: #ffffff;  
}  
.ys9{  
  background: negation(#ff6600,#fff);  
}

**下面是我们的效果图：**

