Sass高级语法

@at-root跳出嵌套

```
//@at-root
@media only screen and (max-width:768px) {
  .box{
    width: 100px;
    height: 100px;
    .aaa{
      background-color: red;
    }
    @at-root {
      .bbb{
        color: blue;
      }
    }
    @at-root(without: media) {
      .ccc{
        font-size: 32px;
      }
    }
  }
}
```

最终生成的CSS代码如下

```
@media only screen and (max-width: 768px) {
  .box {
    width: 100px;
    height: 100px;
  }
  .box .aaa {
    background-color: red;
  }
  .bbb {
    color: blue;
  }
}
.box .ccc {
  font-size: 32px;
}
```

> 在 `@at-root` 的时候， `without:media` 可以帮我们跳出 `@media` 命令，实现全局

在使用 `@at-root` 的时候， 我们还可以跳多次

```
//混合
@mixin sm {
  @media only screen and (max-width: 768px) {
    @content;
```

```
    }
  }

  @include sm() {
    .box {
      width: 100px;
      height: 100px;
      $primaryColor: lightseagreen;

      @at-root(without: media) {
        @at-root {
          .nav {
            background-color: $primaryColor;
          }
        }
      }
    }
  }
```

最好终生成的好的css如下

```
@media only screen and (max-width: 768px) {
  .box {
    width: 100px;
    height: 100px;
  }
}
.nav {
  background-color: lightseagreen;
}
```

> 这个时候生成的 `.nav` 的样式就既跳出为了 `box` ,也跳出了 `@media`

函数

```
$baseSize: 16px !default;

@function pxToRem($px) {
  @return $px / $baseSize + rem;
}

.box1{
  width: pxToRem(100px);
}
```

> 这里的函数与JS里面的函数大同小异，通过 `@function` 定义函数名，也可以设置参数，同时使用 `@return` 来进行返回

运算

```
//运算符

$x: 100px;
$y: 50px;
$z: 2;

.div1 {
  width: $x + $y;
}

.div2{
  //单位除以单位以后就没有单位了
  line-height: $x / $y;
}

.div3{
  height: $x - $y;
}

.div4{
  width: $x * $z;
}

.div5{
  width: $x % $z;
}
```

条件判断

条件判断

```
$a:50;
.div1{
  width: 100px;
  height: 100px;
  //a<30就是红色，小于70就是蓝色，否则就是黑色
  @if($a<30){
    background-color: red;
  }
  @else if($a<70){
    background-color: blue;
  }
  @else{
    background-color: black;
  }
}
```

三目运算符

> 要注意，三目运算符本质上面就是一个 `if` 的函数

```
//三目运算符

$a: 60;

.box {
  width: 100px;
  height: 100px;
  //$a大于50就是黑色，否则就是蓝色
  background-color: if($a>50, black, blue);
}

//if本质上面是一个函数，我可以模拟去实现

@function bggif($a,$b,$c){
  @if($a){
    @return $b;
  }
  @else {
    @return $c;
  }
}

.box1{
  width: 100px;
  height: 100px;
  background-color: bggif($a>50,black,blue);
}
```

for循环

在这里要注意，for循环的语法有2种

第一种

```
@for $变量 from start  to  end {

}
```

这一种写法里面使用的是 `to end` ，那么主不包含结束

第二种

```
@for $变量 from start through end {

}
```

这一种写法是 `through end` 会包含 `end`

```
@for $i from 1 to 9{
  .div#{$i}{
    width: 100px * $i;
  }
```

数组的定义

```
$arr: (100, 200, 300);
```

数组的取值通过 `nth` 来完成

```
.box {
  //数组的取值
  width: nth($arr,1)+px;
}
```

map的定义

```
$colorMap: (
        primary:lightseagreen,
        success:green,
        danger:red,
        info:gray,
        warn:orange
);
```

@each遍历

遍历数组的集合

```
$arr: (100, 200, 300);
@each $item in $arr{
  .w-#{$item}{
    width: $item + px;
  }
}
```

遍历map对象

```
 $colorMap: (
        primary:lightseagreen,
        success:green,
        danger:red,
        info:gray,
        warn:orange
 );

 @each $key,$value in $colorMap{
   .bg-#{$key}{
     background-color: $value;
   }
   .text-#{$key}{
     color: $value;
   }
   .btn-#{$key}{
     background-color: $value;
   }
 }
```

有了这个对象map的遍历以后，我们就可以得到下了面的对象

```scss
$colorMap: (
        primary:lightseagreen,
        success:green,
        danger:red,
        info:gray,
        warn:orange
);
$lineType: (
        solid:solid,
        dashed:dashed,
        double:double,
        groove:groove,
        dotted:dotted
);
//1-5的线条粗细，并且 颜色是primary~wan

@for $i from 1 through 5 {
  @each $key, $value in $lineType {
    @each $key1, $value1 in $colorMap {
      .border-#{$key}-#{$key1}-#{$i} {
        border: $i + px $value $value1;
      }
    }
  }
}
```