# Android Compose换肤

## 前提

Compose框架弃用了传统xml布局解析的方式，采用代码声明来直接写UI。

Compose是一个UI框架，在安卓平台上，仍需遵循Android的资源管理，即Compose除了直接设置一些图片、颜色外，仍然需要通过Resource对象来获取资源。

## 正文

### Compose UI更新方式：



### 设置文本颜色方式：

*Text*(  
 text = "myText",  
 color = *Color*(*resources*.getColor(R.color.*black30*)),  
)

### 动态修改文本颜色（图片同理）：

val myTextColor = *remember* **{** *mutableStateOf*(*Color*(*resources*.getColor(R.color.*black30*)))  
**}***Text*(  
 text = "myText",  
 color = myTextColor.value,  
 modifier = Modifier.*clickable* **{**

**// 点击修改文本颜色** myTextColor.value = *Color*(*resources*.getColor(R.color.*test\_color*))  
 **}**)

### 换肤如何进行？

var isNight = false  
class ResourceSwitcher {  
 fun getColorById(id: Int): Color {  
 if (isNight) {  
 return ...// 通过对应Resource加载对应资源  
 } else {  
 return ...// 通过对应Resource加载对应资源  
 }  
 }  
}  
val switcher = *remember* **{** *mutableStateOf*(ResourceSwitcher())  
**}***Text*(  
 text = "myText",

// 需要换肤的地方用switcher调用即可  
 color = switcher.value.getColorById(R.color.*test\_color*),  
 modifier = Modifier.*clickable* **{**

// 皮肤状态isNight = true  
 // 触发一次更新  
 switcher.value = ResourceSwitcher()  
 **}**)

Compose的更新需要触发重组，通过变更remember的内容就可以重新刷新UI。

## 总结

1. 只需要触发一次重组即可换肤。
2. 没有LayoutInflater反射流程，更稳定、启动更快。
3. 不需要换肤框架。因为全是代码声明，就用不着换肤框架的【拦截-解析-存储-替换】逻辑。
4. 换肤不受系统限制。传统xml换肤中，部分View只能通过反射修改属性，比如EditText的光标、View的滚动条等。
5. 和传统xml换肤相比，同样需要加载对应模式资源的逻辑。如果只是白天黑夜资源，那么很简单，构造对应的白天黑夜Context就行；如果是加载外部皮肤，也只需构造对应的Resource对象即可。compose换肤可直接复用xml换肤加载对应资源的逻辑。
6. 和传统xml换肤互不影响。