**SkinCore换肤库技术分析**

1. 背景介绍

早期589使用的是PaintedSkin换肤框架，版本号为3.4.2。经分析，该版本号已经失效，即无法重新拉取依赖，在一台新的机器上运行的话会提示找不到依赖。同时，为了适配模块化换肤，PaintedSkin核心库是以源码方式引入，在换肤时，为了不影响launcher的换肤，有两种方式可选择：

1：给每个需要换肤的View设置一个app:skin="true"属性，这种方式是框架本身支持的方式。

2：遍历导航根节点以下的View，导航换肤根节点有MainFragment的容器、Windowmanger添加的View、导航dialog。但这种方式需要手动清除RecyclerView、ListView里面的缓存。

由于第一种方式工作量巨大，故589选择的第二种方式（也是base代码选择的方式）。

所以，潍柴在初期就更换了SkinCore作为换肤框架，底层实现基本原理和PaintedSkin一致，都是通过LayoutInflater.Factory监听的方式进行换肤。

1. 优缺点分析

优点：

支持皮肤包+白天黑夜切换，即一个皮肤包里面可以制作白天皮肤和黑夜皮肤。

支持各种监听，可以在解析xml时监听，可以在解析时插入自己的逻辑，可以在换肤时监听并拦截某些View的换肤（用于提速）

支持添加自定义属性换肤和更改默认换肤属性

支持且方便模块化换肤，比如指定某个模块能够换肤或者指定某个模块不换肤

如果是制作皮肤包的方式换肤，可以快速配置并部署皮肤包

支持视图调试工具功能，可以在开发中快速定位视图和快速应用xml和kotlin更改

缺点：

项目中如果使用了Dialog类，需要指定一个dialog主题

代码中设置图片和颜色等资源需要通知到换肤框架（该问题是所有换肤框架都无法避免的，除非在编译阶段做字节码替换，但经试验，ASM全量替换会大幅度延长编译时间）

1. 引入方式

|  |
| --- |
| Kotlin // 视情况添加，如果有guava依赖冲突则添加  implementation 'com.google.guava:listenablefuture:9999.0-empty-to-avoid-conflict-with-guava'  // 换肤框架  implementation 'com.github.DFFXT.SkinSwitch:SkinCore:0.19'  // 视情况添加，视图调试框架，支持xml修改实时生效，和kotlin文件修改冷启动生效，用于节省编译时间，具体使用方式需搭配android studio插件使用[插件下载](https://github.com/DFFXT/ViewDebug-Trans)  debugImplementation 'com.github.DFFXT.SkinSwitch:ViewDebug:0.19' |

|  |
| --- |
| Kotlin SkinManager.init(context as Application, 0, DefaultProviderFactory())   // 设置黑夜模式  SkinManager.applyThemeNight(true)  // 切换皮肤主题，不同主题会加载不同的皮肤包，加载失败则使用默认皮肤包  SkinManager.switchTheme(....)  // 监听换肤  SkinManager.addSkinChangeListener(...)  // 新增支持的换肤属性  SkinManger.addAttributeCollection(....)   // 当某个View换肤时，会触发此回调，根据返回状态确定是否需要换肤  AttrApplyManager.onApplyInterceptor = {....} |

1. 实现原理

换肤：SkinCore

反射layoutInflater，设置代理LayoutInfalter.Factory2监听，从而监听View的创建。将解析的属性存储到ViewUnion对象里面，然后将这个对象再通过view.setTag(id, union)的方式存储。

通过构造皮肤包Resource来实现加载外部皮肤包。

通过在解析时解析的app:skin和app:skin\_forDescendants属性来决定一个View是否在换肤时进行换肤从而达到模块化换肤的功能。

1. 引入成本

最大的成本在于代码中动态设置的资源id需要同步到换肤框架，但该问题是所有换肤框架均无法避免的。如果项目已经适配过其它换肤框架，那么转到该换肤框架则可以很方便进行。只需替换换肤监听、对接代码动态设置资源方法、更改切换皮肤方式。

1. 评估结果