# 安卓动态换肤

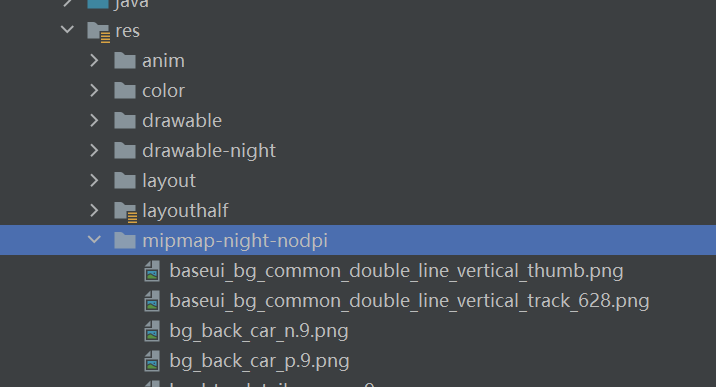
Demo地址：[DFFXT/SkinSwitch: Android 换肤框架 (github.com)](https://github.com/DFFXT/SkinSwitch)

### 1、什么是动态换肤

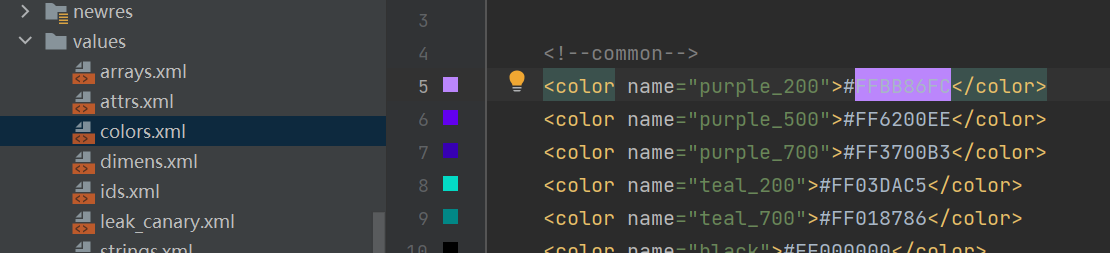
动态在于不用重启app或者重启页面。Android本身支持白天黑夜的切换、支持主题的切换，但都需要重启页面或者重启应用。

Android换肤主要包括图片(drawable、mipmap)切换和颜色(color)切换（尺寸dimens、字符串string、动画anim等一般不包括）。

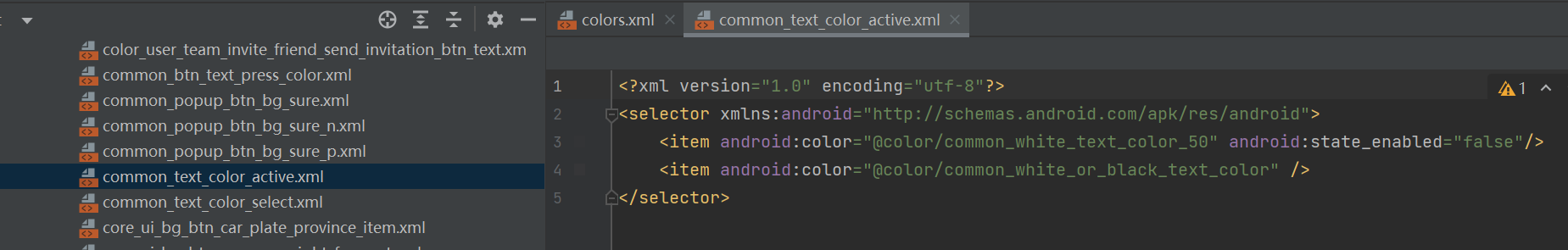
图片资源（一般放置在drawable、mipmap等目录）



颜色资源（一般放置在values目录下的xml中）



带状态的颜色资源（位于color、drawable目录）

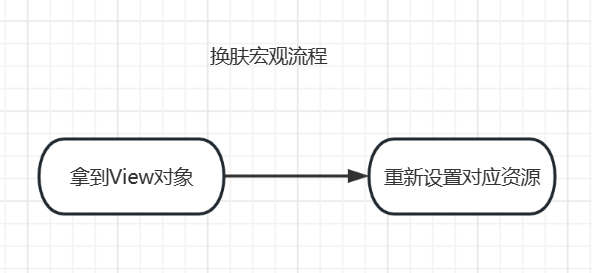


切换皮肤需要达到的效果：图片改变，文字颜色改变

要想达到这个效果，只能通过调用控件的API来更改（如setBackground、setTextColor）

### 2、换肤的宏观流程

换肤的宏观流程就是：在要需要换肤的时候，调用需要换肤的View的API，重新设置图片和颜色。



其中待解决问题有两点

1. 如何在换肤时能拿到正确的View
2. 拿到View了后知道该设置哪个图片哪个颜色

#### 2.1如何获取View

##### 2.1.1 View的生成方式：

1. 直接在代码中new一个View

TextView tv = new AppCompatTextView(context);

1. 解析布局文件生成View

View view = LayoutInflater.from(context).inflate(layoutId, parent, true);

##### 2.1.2 LayoutInflater生成View的流程：

LayoutInflater.java

public View inflate(@LayoutRes int resource, @Nullable ViewGroup root, boolean attachToRoot) {

        final Resources res = getContext().getResources();

        if (DEBUG) {

            Log.d(TAG, "INFLATING from resource: \"" + res.getResourceName(resource) + "\" ("

                  + Integer.toHexString(resource) + ")");

        }

        View view = tryInflatePrecompiled(resource, res, root, attachToRoot); // 预编译视图

        if (view != null) {

            return view;

        }

        XmlResourceParser parser = res.getLayout(resource);

        try {

            return inflate(parser, root, attachToRoot); // 普通方式解析视图

        } finally {

            parser.close();

        }

    }

先判断释是否有预编译生成的View，没有则走普通解析xml的方式。

Inflate方法最终还是会走到createViewFromTag

LayoutInflater.java

View createViewFromTag(View parent, String name, Context context, AttributeSet attrs,

            boolean ignoreThemeAttr) {

// ...................

        try {

            View view = tryCreateView(parent, name, context, attrs); // 尝试创建View

            if (view == null) {

                final Object lastContext = mConstructorArgs[0];

                mConstructorArgs[0] = context;

                try {

                    if (-1 == name.indexOf('.')) {

                        view = onCreateView(context, parent, name, attrs); // 尝试创建View

                    } else {

                        view = createView(context, name, null, attrs); // 尝试创建View

                    }

                } finally {

                    mConstructorArgs[0] = lastContext;

                }

            }

// .............

}

其中关注3个方法：

tryCreateView、onCreateView、createView

tryCreateView

public final View tryCreateView(@Nullable View parent, @NonNull String name,

        @NonNull Context context, @NonNull AttributeSet attrs) {

        if (name.equals(TAG\_1995)) {

*// Let's party like it's 1995!*

            return new BlinkLayout(context, attrs);

        }

        View view;

        if (mFactory2 != null) {

            view = mFactory2.onCreateView(parent, name, context, attrs); // 先使用Factory2创建

        } else if (mFactory != null) {

            view = mFactory.onCreateView(name, context, attrs); // 再使用Factory创建

        } else {

            view = null;

        }

        if (view == null && mPrivateFactory != null) {

            view = mPrivateFactory.onCreateView(parent, name, context, attrs); // 最后使用Factory2创建

        }

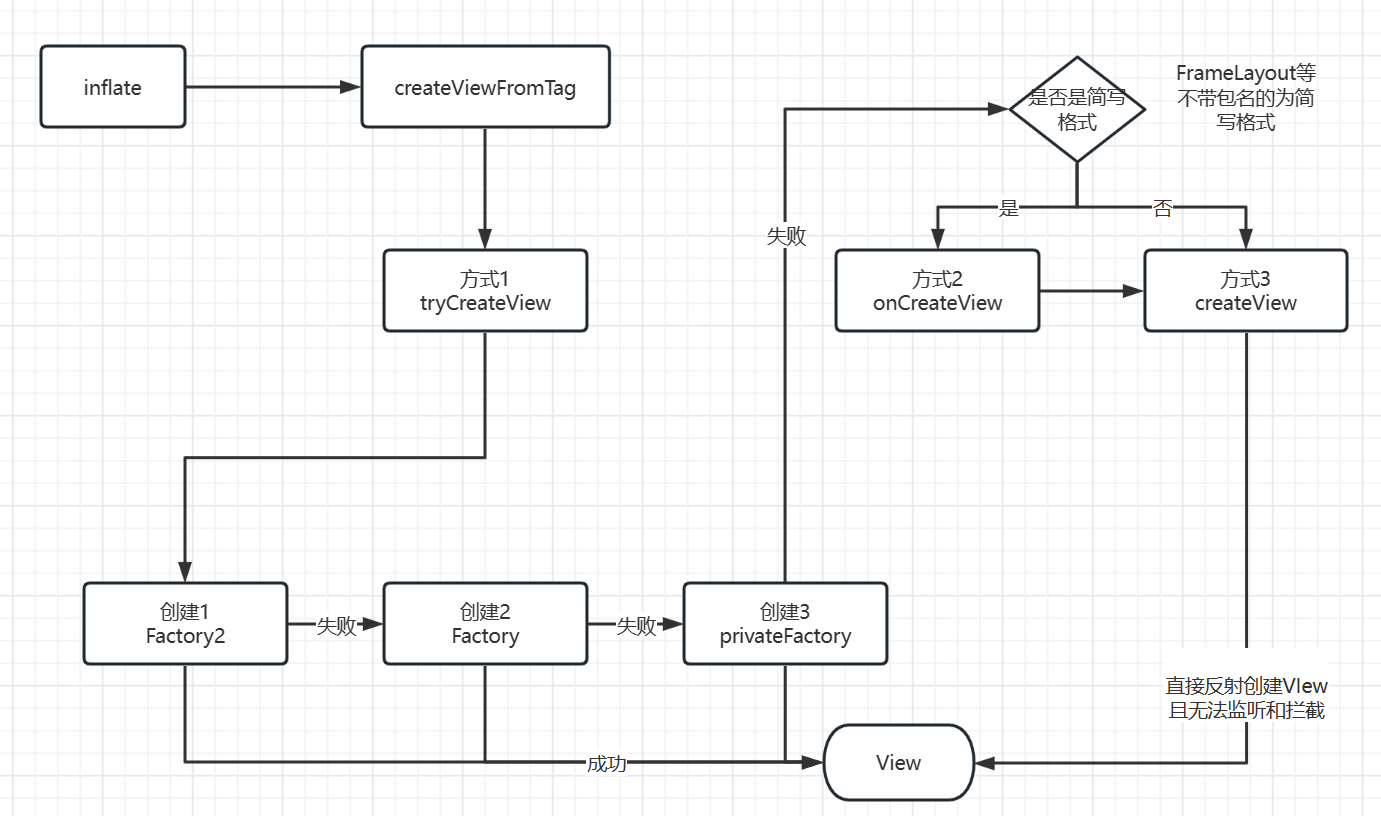
        return view;

}

分析：其中是又通过Factory2、Factory以及privateFactory来创建视图对象。

这些Factory都能在外部进行设置，理论上设置了自己的Factory就能监听所有的View的创建。

Inflate创建View总结：

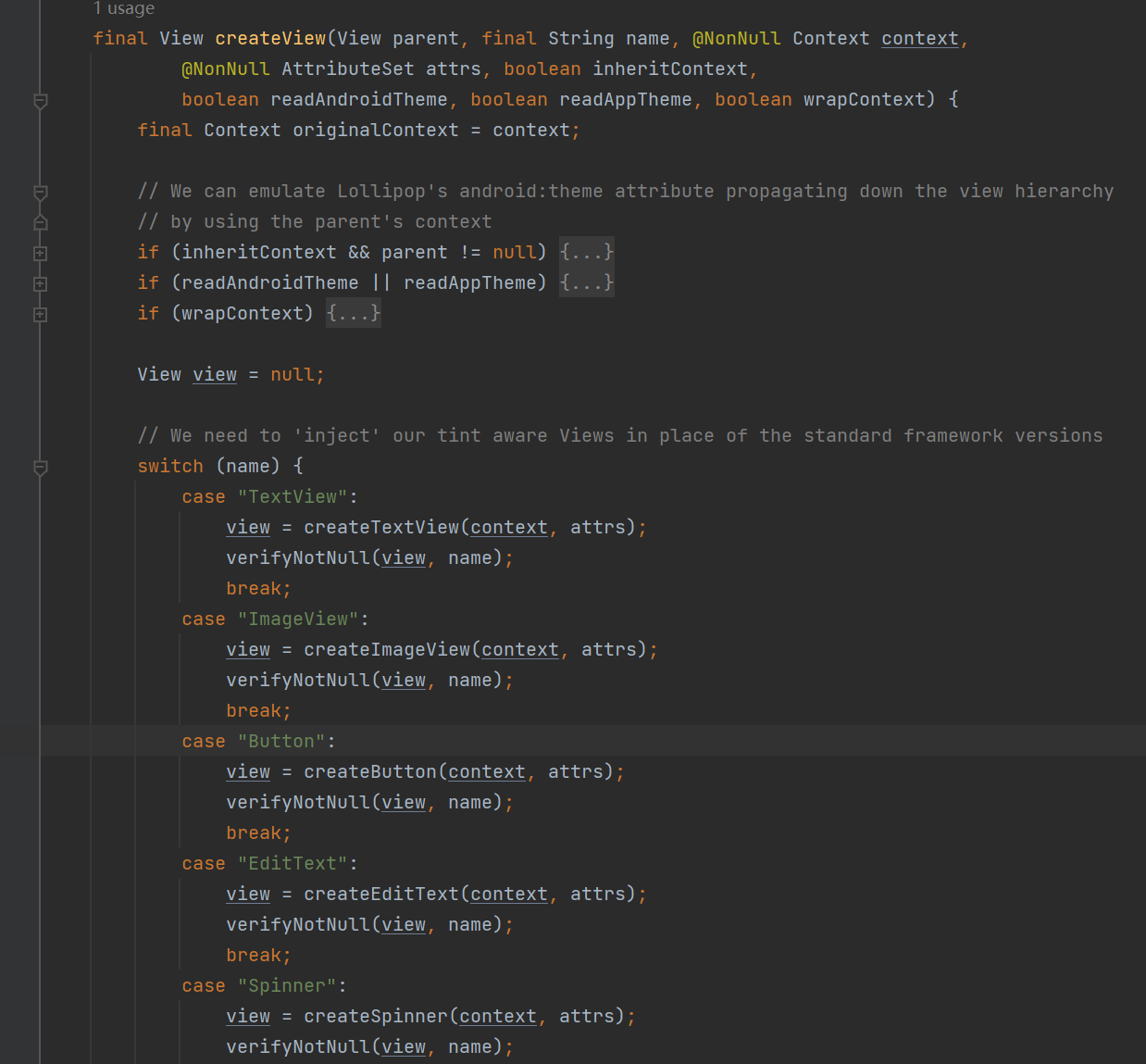


##### 2.1.3开发中两个比较重要的Factory2

AppCompatDelegateImpl，内部实现系统控件转换

系统中老版本控件转换为AppCompatXXX就是通过设置Factory2来实现的（在super.onCreate阶段设置）

androidx.appcompat.app.AppCompatViewInflater.java



实际使用的是MaterialComponentsViewInflater，这个Factory2是通过主题设置的。

public View createView(View parent, final String name, @NonNull Context context,

            @NonNull AttributeSet attrs) {

        if (mAppCompatViewInflater == null) {

            TypedArray a = mContext.obtainStyledAttributes(R.styleable.AppCompatTheme);

            String viewInflaterClassName =

                    a.getString(R.styleable.AppCompatTheme\_viewInflaterClass); // 获取主题中设置的类名

            if (viewInflaterClassName == null) {

                mAppCompatViewInflater = new AppCompatViewInflater();

            } else {

                try {

                    Class<?> viewInflaterClass = Class.forName(viewInflaterClassName);

                    mAppCompatViewInflater =

                            (AppCompatViewInflater) viewInflaterClass.getDeclaredConstructor()

                                    .newInstance(); // 反射创建设置的Factory2

                } catch (Throwable t) {

                    Log.i(TAG, "Failed to instantiate custom view inflater "

                            + viewInflaterClassName + ". Falling back to default.", t);

                    mAppCompatViewInflater = new AppCompatViewInflater();

                }

            }

        }

FragmentLayoutInflaterFactory

只负责解析<fragment>和<androidx.fragment.app.FragmentContainerView>标签，在FragmentActivity.onCreateView中间接被调用。

Activity

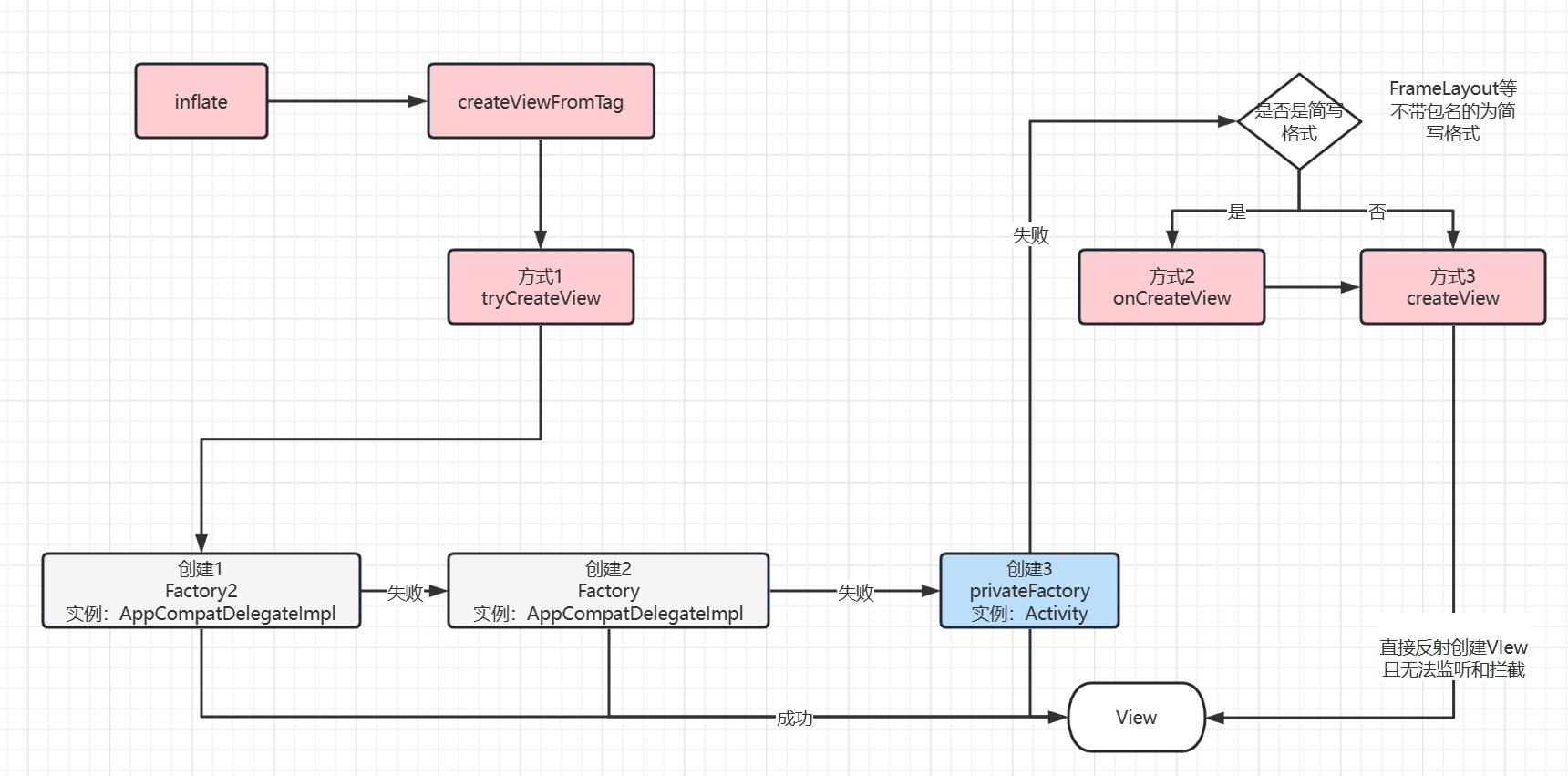
Activity也实现了Factory2接口，Activity实例会赋值给LayoutInflater.mPrivateFactory, 在Activity.attach中设置的。

LayoutInflater中各个Factory实例：

Factory2：AppCompatDelegateImpl 实例，分发到MaterialComponentsViewInflater

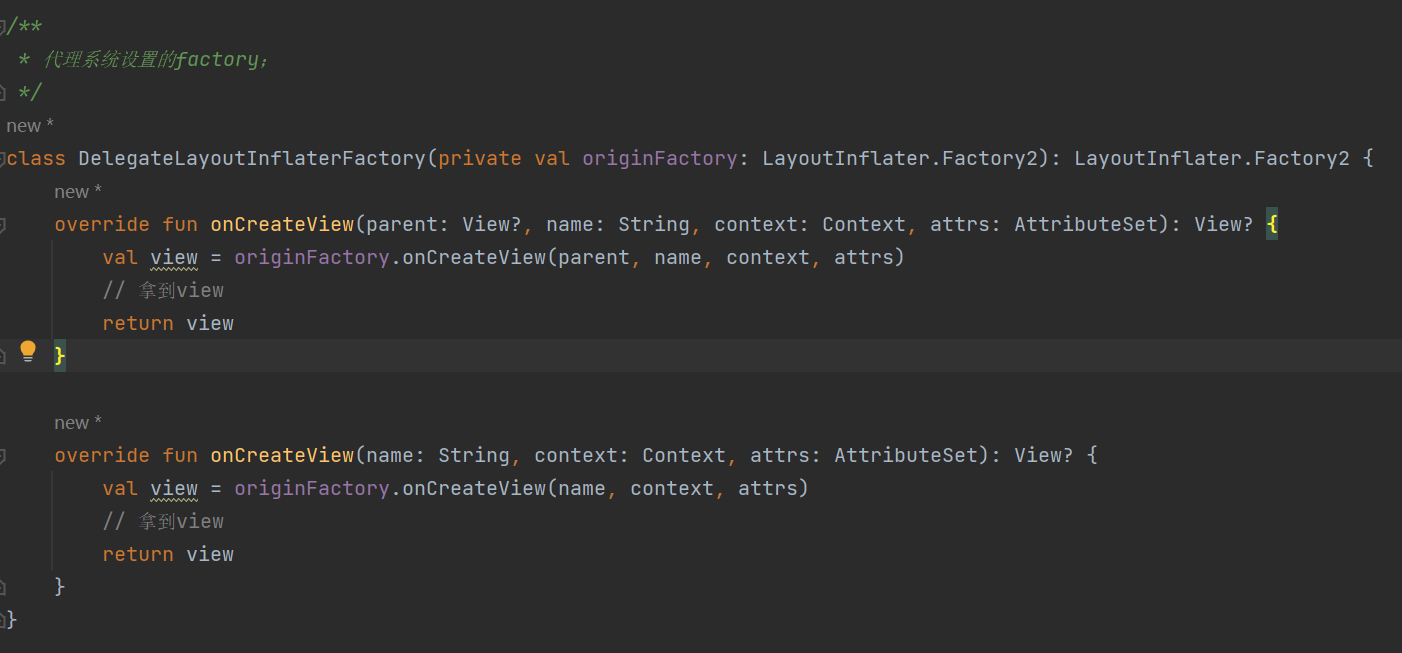
Factory：AppCompatDelegateImpl 实例，分发到MaterialComponentsViewInflater

privateFactory：Activity实例（针对FragmentContainerView会分发给FragmentLayoutInflaterFactory执行）



如何拿到View？

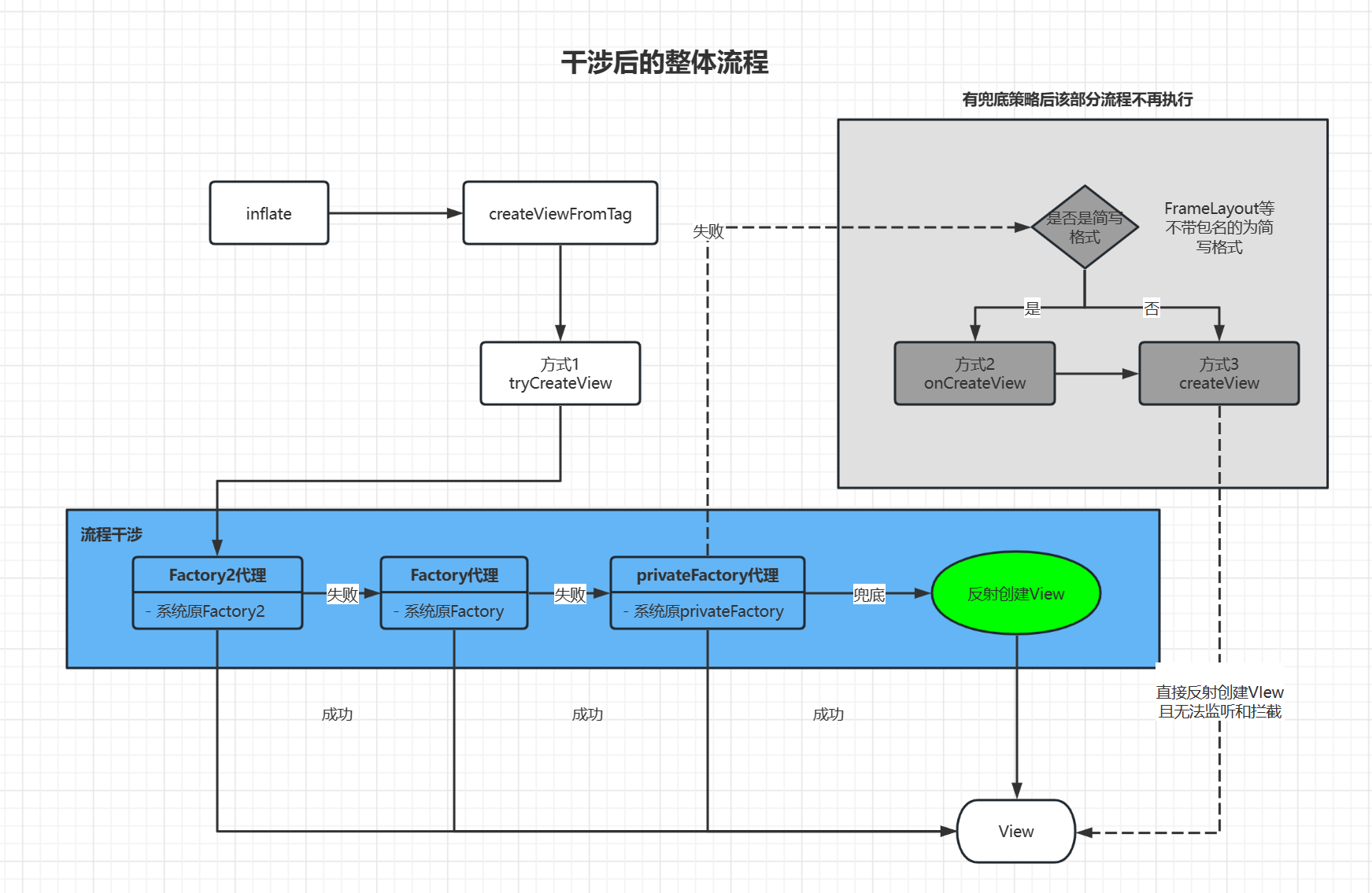
为了不破坏系统自带的Factory和Factory2的功能，可以通过设置代理Factory的方式来拦截



通过反射拿到系统设置的Factory，然后设置为自己的Factory

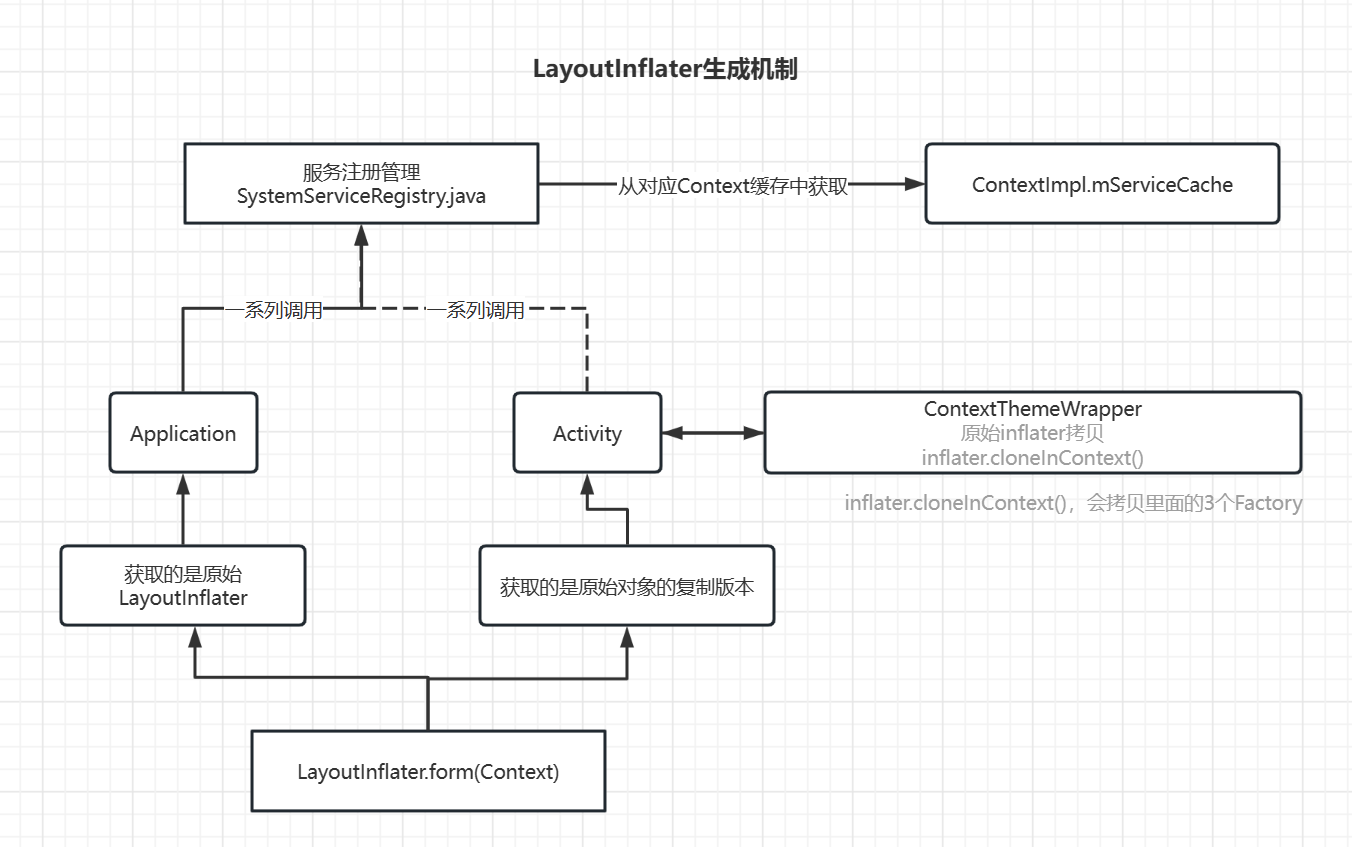
当代理的3种Factory都无法创建对象时，需要通过反射创建View，此时不能走系统反射创建View，因为后续流程无法拦截，无法得到系统反射创建的View。

所以需要进行兜底，我们自己来完成反射创建View的流程



##### 2.1.4、如何代理LayoutInflater#Factory

LayoutInflater的来源



总结：

1. 一个Application对应一个LayoutInflater
2. 一个ContextThemeWrapper对应一个LayoutInflater
3. LayoutInfalter.cloneInContext()的复制会复制内部的Factory
4. 只需要代理application、activity里面Inflater的Factory即可

*/\*\*  
 \* 将original的Factory复制到target，并用FactoryDelegate代理  
 \** ***@param*** *original 原始对象  
 \** ***@param*** *target 新对象，也可以是原始对象  
 \** ***@param*** *onViewCreatedListener 监听创建  
 \*/*fun delegate(  
 original: LayoutInflater,  
 target: LayoutInflater,  
 onViewCreatedListener: IOnViewCreated  
) {  
 // 反射field并设置新的值  
 *factoryFiled*.set(target, DelegateLayoutInflaterFacoty(onViewCreatedListener, original.*factory*))  
 *factory2Filed*.set(target, DelegateLayoutInflaterFacoty(onViewCreatedListener, original.*factory2*))  
 *privateFactoryFiled*.set(  
 target,  
 PrivateFactoryDelegate(target, onViewCreatedListener, original.*privateFactory*())  
 )  
}

class ActivitiesCallback private constructor() :

    Application.ActivityLifecycleCallbacks {

    companion object {

        fun register(application: Application) {

            application.registerActivityLifecycleCallbacks(ActivitiesCallback())

        }

    }

    override fun onActivityCreated(activity: Activity, savedInstanceState: Bundle?) {

        val origin = LayoutInflater.from(activity)

        LayoutInflaterDelegate.delegate(origin, origin , object : IOnViewCreated {

            override fun onViewCreated(view: View, name: String, attributeSet: AttributeSet) {

*// todo 拿到被创建的View*

            }

        })

    }

}

通过view回调，就可以拿到被创建的View，将View通过容器存储起来即可

#### 2.2重新设置皮肤资源

##### 1、生成资源映射

通过干涉布局解析，拿到了所有的View，下一步就是要调用View的API设置背景颜色、背景图片、文本颜色等，但是具体设置什么颜色、什么图片呢？

干涉布局解析时记录View的各种属性。

在Factory2或者Factory中onCreateView有一个AttributeSet参数，里面记录了view的各个属性

public interface Factory2 extends Factory {

        @Nullable

        View onCreateView(@Nullable View parent, @NonNull String name,

                @NonNull Context context, @NonNull AttributeSet attrs);

}

从AttributeSet获取所需属性

public void onViewCreated(View parent, View view, AttributeSet attributeSet) {

    // 要换肤的属性

    int[] skinStyles = new int[]{android.R.attr.textColor, android.R.attr.background,  android.R.attr.backgroundTint};

TypedArray ta = view.getContext().obtainStyledAttributes(attributeSet, skinStyles);

// 获取资源id

    int colorId = ta.getResourceId(0, 0);

    int bgId = ta.getResourceId(1, 0);

    int bghId = ta.getResourceId(2, 0);

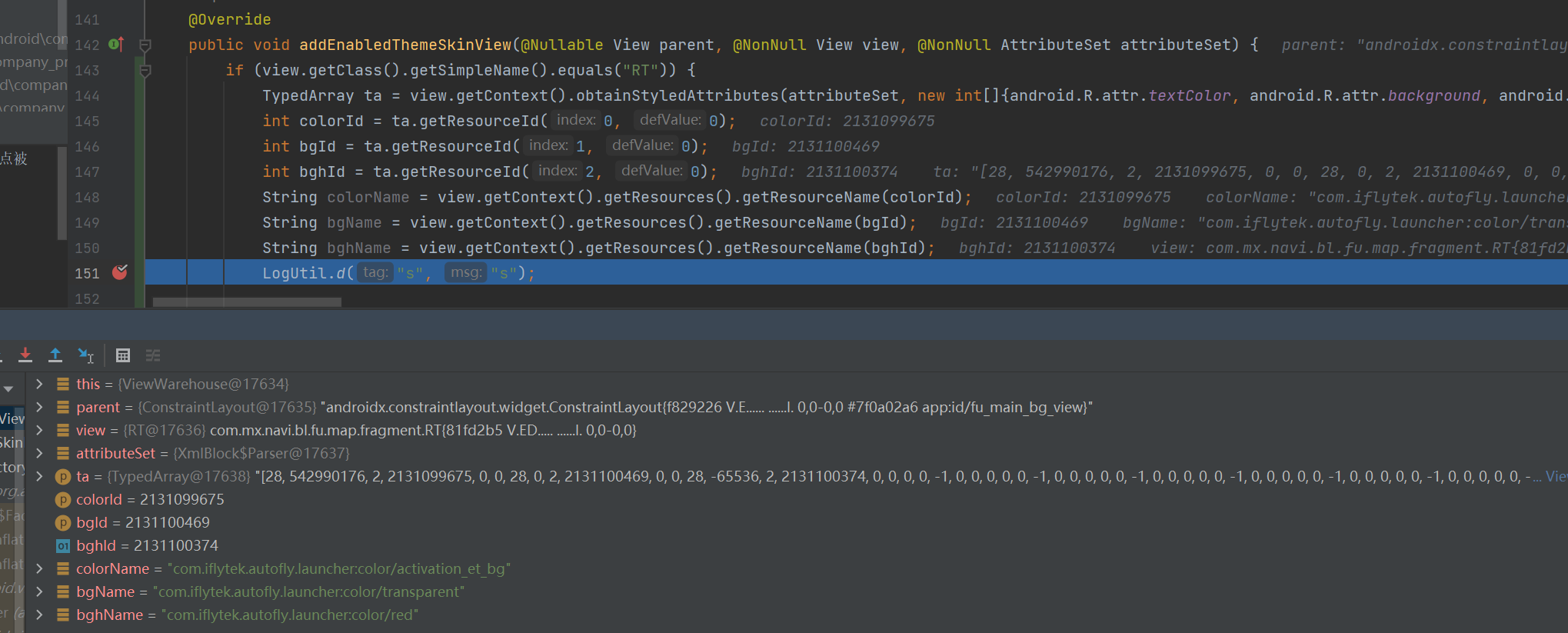
    // 获取资源名称

    String colorName = view.getContext().getResources().getResourceName(colorId);

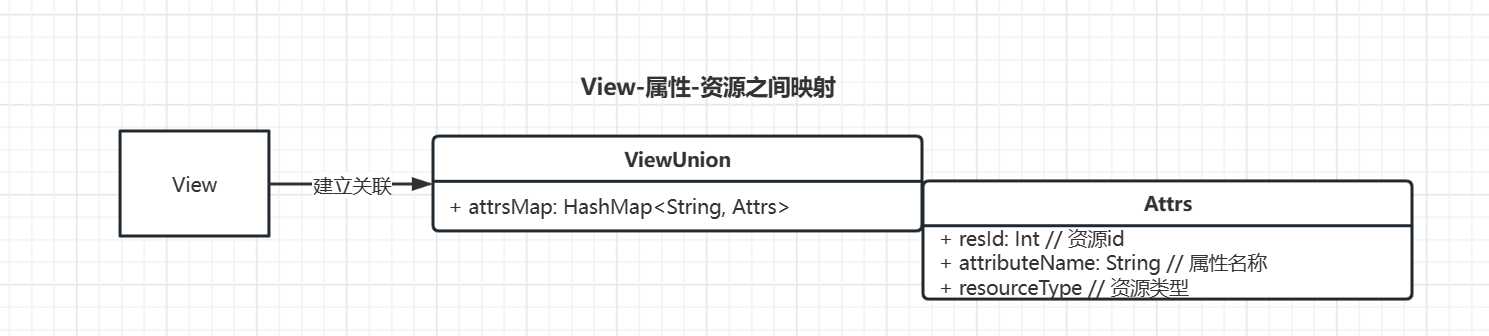
    String bgName = view.getContext().getResources().getResourceName(bgId);

    String bghName = view.getContext().getResources().getResourceName(bghId);

}



最终建立起如下关系的映射，通过View可以知道哪些属性需要切换，以及属性对应的资源id、资源类型等。

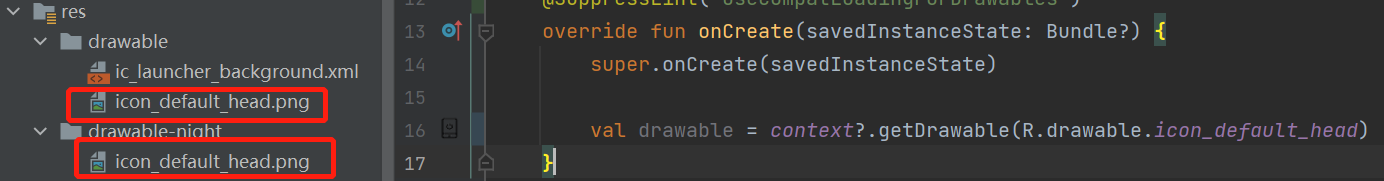


### 3、皮肤包资源生成和读取

首先是皮肤包和白天黑夜的关系。一个皮肤包就是一个apk，一个皮肤包可以包含白天资源和黑夜资源。

安卓本身就支持白天黑夜模式，如果皮肤只有白天黑夜的切换，那就不需要皮肤包了。

#### 1、如何指定获取白天或者黑夜的资源？



在这种情况下，似乎不知道返回的drawable是哪个目录下的？

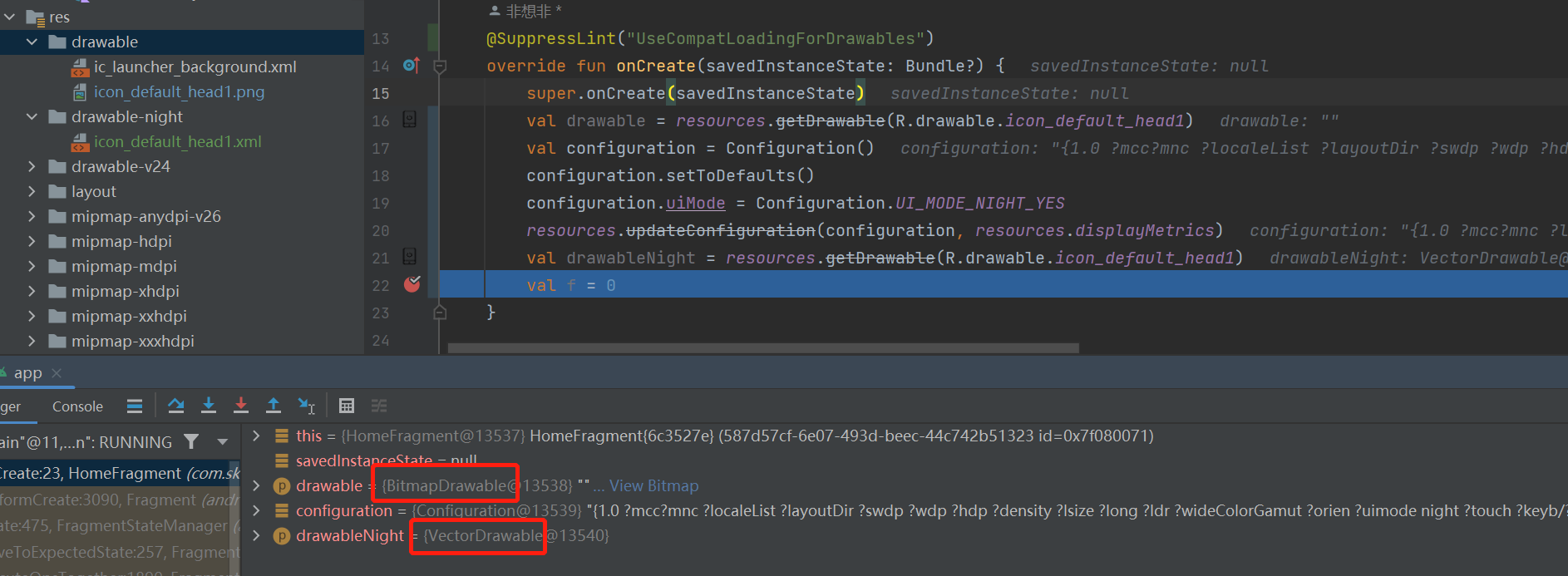
Android资源加载配置：Configuration

Configuration内部包含：字体缩放、屏幕分辨率、白天黑夜模式、布局方向、横竖屏等配置

加载夜间模式下的图片：

1. *// 拷贝配置*
2. val configuration = Configuration(resources.configuration)
3. *// 设置夜间资源*
4. configuration.uiMode = configuration.uiMode or Configuration.UI\_MODE\_NIGHT\_YES
5. *// 设置屏幕dpi，匹配对应的资源*
6. configuration.densityDpi = 320
7. *// 设置横屏资源*
8. configuration.orientation = Configuration.ORIENTATION\_LANDSCAPE

简单验证



相对于589项目来说，只有白天黑夜模式，那么就不用构造皮肤包了。

由于589的白天黑夜是跟随系统的，所以连配置Configuration都省了，（系统切换白天黑夜时会自动更新Resources的Configuration）

跟随系统切换，需要确保切换时不被杀死。

android:configChanges="uiMode"

如果白天黑夜不跟随系统，或者有其他类型的皮肤该如何处理？

#### 2、加载外部皮肤包

皮肤包就是一个APK，要加载皮肤包，那么皮肤包中得有androidManifest.xml、resources.arsc、以及各种资源目录。

由于只加载资源，那么可以删除apk文件中的二进制代码文件(.dex)，不影响加载（如果需要加载代码则不能删除）。

分析正常情况下获取一个drawable的方式：

1. context.getDrawable(R.drawable.icon\_default\_head)

Context.java

@Nullable

public final Drawable getDrawable(@DrawableRes int id) {

    return getResources().getDrawable(id, getTheme());

}

public Drawable getDrawable(@DrawableRes int id, @Nullable Theme theme) throws NotFoundException {

    return getDrawableForDensity(id, 0, theme);

}

Resources.java

public Drawable getDrawableForDensity(@DrawableRes int id, int density, @Nullable Theme theme) {

        final TypedValue value = obtainTempTypedValue();

        try {

            final ResourcesImpl impl = mResourcesImpl;

            impl.getValueForDensity(id, density, value, true);

            return loadDrawable(value, id, density, theme);

        } finally {

            releaseTempTypedValue(value);

        }

    }

ResourcesImpl.java

void getValueForDensity(@AnyRes int id, int density, TypedValue outValue,

            boolean resolveRefs) throws NotFoundException {

        boolean found = mAssets.getResourceValue(id, density, outValue, resolveRefs);

        if (found) {

            return;

        }

        throw new NotFoundException("Resource ID #0x" + Integer.toHexString(id));

    }

所以访问资源是通过Resources或者AssetManager对象来实现的。（当然，拿到了资源名称也可以自己写一套资源获取方式）

构造属于皮肤包的Resource对象

Resources.java

public Resources(AssetManager assets, DisplayMetrics metrics, Configuration config) {

        this(null);

        mResourcesImpl = new ResourcesImpl(assets, metrics, config, new DisplayAdjustments());

}

构造AssetManager、Resources对象

1. val manager1 = AssetManager::class.java.newInstance()
2. // 反射调用addAssetPath方法
3. val method = AssetManager::class.java.getDeclaredMethod("addAssetPath", String::class.java)
4. method.isAccessible = true
5. *// 设置皮肤路径 方法*
6. method.invoke(manager1, “外部皮肤路径”)
7. *// 生成Resources对象*
8. val res = Resources(
9. manager1,
10. context.resources.displayMetrics,
11. context.resources.configuration
12. )

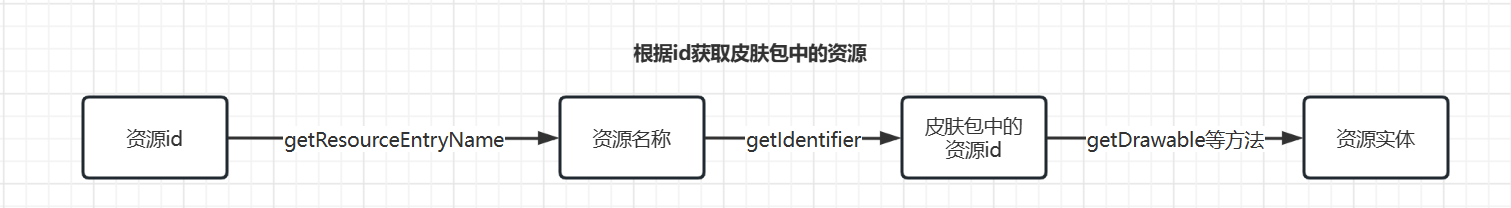
获取皮肤包中某个资源

定位一个资源的方式：

包名:类型/文件名

比如项目中有某个资源id是0xxxxx，但是在皮肤包中，同样名称资源的id却很有可能不一样，所以只能通过名称来对应资源。

1. *// 获取皮肤中的某个资源 R.drawable.icon\_default\_head*
2. *// 通过应用resource获取名称和资源类型*
3. val resourcesName = resources.getResourceEntryName(R.drawable.icon\_default\_head) *// icon\_default\_head*
4. val resourcesType = resources.getResourceTypeName(R.drawable.icon\_default\_head) *// drawable*
5. val skinResources: Resources *// 通过皮肤包构造的Resources*
6. *// 通过皮肤Resource获取对应资源id*
7. val skinDrawableId = skinResources.getIdentifier(resourcesName, resourcesType, "皮肤包包名")
8. val skinDrawable = skinResources.getDrawable(skinDrawableId)



### 4、总结

整个动态换肤中需要做的事情

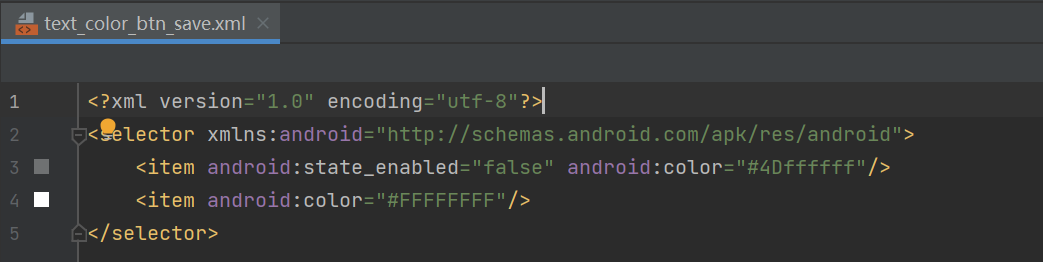
1. 代理Factory，获取所有通过inflater生成的View
2. 解析AttributeSet，拿到view的属性以及设置的id
3. 加载皮肤资源，不管是外置皮肤包方式还是外置apk方式或者内置皮肤方式，都需要构造或者得到Resource对象用于皮肤加载。
4. 等待换肤信号，遍历得到的view，重新设置新的皮肤资源。

[整体设计流程](整体设计.svg)

~~实际使用中drawable和color资源的一些问题~~

~~resources.getResourceTypeName(id) 返回drawable类型，就一定能通过resources.getDrawable(id)来获取么？~~

~~比如：~~

~~~~

~~这种引用方式为：R.drawable.xxx 的资源就不能直接用getDrawable的方式来获取。~~

~~val colorStateList: ColorStateList = context.getColorStateList(R.drawable.text\_color\_btn\_save)~~

注意内存泄露，可以用WeakHashMap存储View对象和View要切换的属性

实际开发中遇到的问题

对应的资源放在对应的文件夹下

比如：有一个ColorStateList资源，但是放在drawable目录下。resources.getResourceTypeName(id) 就会返回drawable类型，但此时并不能通过getDrawable方法来获取。

C589只是一个aar，真正运行时包含launcher壳的代码，由于导航和launcher的换肤方案不一致，导致一个apk中存在两套换肤方案，功能冗余，导航在换肤时不能对壳项目的View进行换肤。

C589：每个Fragment都遍历自己的视图树，得到View来换肤，对于一些不在fragment中的视图就特殊处理。

RecyclerView、ListView由于有缓存机制，缓存的视图不在fragment视图树中，需要在换肤时清空缓存视图。

其他方案：给每个view设置有个app:skin=”true”的属性。解析属性时判断app:skin的值。缺点就是项目所有布局xml都得添加。

一些属性的动态修改需要用到反射，可能不同版本反射字段不一致

android:scrollbarThumbHorizontal="" 系列

android:textCursorDrawable="" 输入框光标

...

一些属性修改需要额外设置drawable bounds参数

android:drawableStart="" TextView componentDrawable系列

android:indeterminateDrawable="" ProgressBar动态图标

...

一些常见属性

1. <attr name="android:background" />
2. <attr name="android:foreground" />
3. <attr name="android:drawableStart" />
4. <attr name="android:drawableEnd" />
5. <attr name="android:drawableTop" />
6. <attr name="android:drawableBottom" />
7. <attr name="android:textColor" />
8. <attr name="android:textCursorDrawable" /> <!--文本框光标-->
9. <attr name="android:textColorHint" />
10. <attr name="android:textColorHighlight" />
11. <attr name="android:textColorLink" />
12. <attr name="android:src" />
13. <attr name="android:divider" /> <!--分割线-->
14. <attr name="android:button" /> <!--按钮-->
15. <attr name="android:thumb" /> <!--进度条图标-->
16. <attr name="android:progressDrawable" /> <!--progressbar进度drawable-->
17. <attr name="android:backgroundTint" />
18. <attr name="android:foregroundTint" />
19. <attr name="android:tint" />
20. <attr name="android:indeterminateDrawable" /> <!--progressbar动画图标-->
21. <attr name="android:scrollbarThumbHorizontal" /> <!--滚动条-->
22. <attr name="android:scrollbarThumbVertical" />

换肤控件需要提供的额外功能

1. 支持动态添加和移除View的换肤功能
2. 方便开发