SkinResources（工具类）：利用宿主APK和皮肤包APK的不同Resources来获取到不同APK中的相同名称、相同类别的资源在R文件中的id值，根据是否要替换来决定是使用宿主apk的资源还是皮肤包apk的

资源的支持：资源中在解析xml文件的时候就是使用的Resources来获取apk中的资源（实际上是调用Resources中引用的AssetManager，调用相关API来获得资源）我们可以参照资源的方法来得到皮肤包apk中的资源

例如 getColor方法，首先通过宿主apk中的id值来得到这个资源的名称和类别，然后通过名称和类别通过API来得到皮肤包apk中的id值，然后通过Resources.getColor方法得到这个资源id所对应的颜色值

SkinThemeUtils（工具类）：这个工具类用于修改我们创建项目时选择的预编译系统主题中我们想要更改的一些东西，例如状态栏、操作栏等

（getResId方法的作用是获取宿主apk中主题属性资源的id）

资源的支持：原理还是利用Resources的方式得到皮肤包apk中这些资源id的值手动进行重新设置

SkinAttribute（封装类）：这个类将一个需要解析的xml文件中的每一个view进行了封装，用于记录"那一个view中的哪些属性需要进行换肤

SkinView（SkinAttribute的静态内部类）：成员变量：view（需要换肤的view实例）
skinPairs（记录这个view中需要换肤的属性）

SkinPair（SkinAttribute的静态内部类）：成员变量：attributeName（xml文件中view的属性名称）
resId（在xml文件中给属性设置的资源的id）

所以applySkin方法的作用就是真正执行资源替换这个工作了，调用SkinResources中写好的API，将skinPairs中记录下来的需要换肤的属性进行资源替换

SkinPreference：使用SharePreference来记录上次使用的皮肤包，记录皮肤包apk的地址

SkinViewSupport：外部接口，用于让我们的自定义View也能进行换肤

ShinAttribute.look 这个方法是用于筛选这个view中哪些是可以换肤的属性，当我们指定了需要换肤的属性名称，并不意味着这些属性就一定可以进行换肤，因为如果属性的值为写死的颜色代码，是无法进行换肤的；如果属性的值为系统的资源，我们也是要进行区分的，所以需要进行筛选。筛选之后将可以进行换肤的属性记录下来，最后遍历完这个view的所有属性之后将该view封装为一个shinview增加到mShinViews集合中同时执行换肤操作

ShinLayoutInflaterFactory（观察者） 用来接管系统中view的创建过程。我们可以仿照源码中view的创建流程，在view创建出来之后，通过ShinAttribute.look方法对view的属性进行筛选和记录

Application.ActivityLifecycleCallback 这个接口可以监听整个程序中Activity的生命周期，其中的方法会在各个生命周期方法结束时由系统调用

ShinManager（被观察者） 这个类是公开给使用者进行换肤操作的。其中的loadShin方法用来初始化ShinResource，以及给设置的观察者发送通知，让观察者进行换肤操作

框架运行流程：

1. 在自定义的Application的onCreate中初始化ShinManager，ShinManager的构造方法中会利用Application来完成一系列工具类的初始化，并给Application注册Activity的生命周期监听，并设置被观察者；同时调用ShinManager.loadShin方法加载皮肤
2. 由于此时的SharedPrefence中没有保存过皮肤包apk路径，ShinResource和ShinPreference没有进行初始化，通知所有观察者
3. 由于此时没有Activity启动过，因此ShinManager被观察者没有设置过观察者，此时没有Factory的update方法会回调执行
4. 当MainActivity被启动后，会先执行super.onCreate方法，最终会执行到Activity.onCreate方法，执行getApplication().dispathActivityCreated时，就会回调我们之前注册的生命周期的监听，此时就会给ShinManager被观察者设置观察者，每创建一个新的Activity，就会为ShinManager被观察者设置一个观察者ShinLayoutInflaterFactory
5. 此时每个Activity都会有一个ShinLayoutInflaterFactory被设置了，当setContentView方法执行时，创建View的过程就会被Factory所拦截，执行我们自定义的onCreateView方法，筛选可以进行换肤的属性，记录的同时并执行换肤。由于此时ShinResource没有进行初始化，因此仍使用的是原皮肤
6. 当我们点击按钮进行换肤时，就会调用ShinManager的loadShin方法，对ShinResource进行初始化，通知观察者更新，执行ShinLayoutInflaterFactory的update方法，执行ShinAttribute的applyShin方法遍历之前记录的属性进行换肤

框架运行流程：7. 当APP之前已经进行过换肤后，在Application.onCreate方法中初始化SkinManager的时候，调用loadSkin时把SkinResource也初始化了。但此时通知所有观察者时也是不会有作用的，因为此时没有观察者设置了。当之后有activity启动后，就会回调到我们自定义的SkinLayoutInflaterFactory的onCreateView方法中，此时执行SkinAttribute的load方法时使用的就是皮肤包的皮肤，此时就会执行换肤。

8. 之后所有启动的Activity都会回调到SkinLayoutInflaterFactory的onCreateView方法中，执行load方法，执行换肤