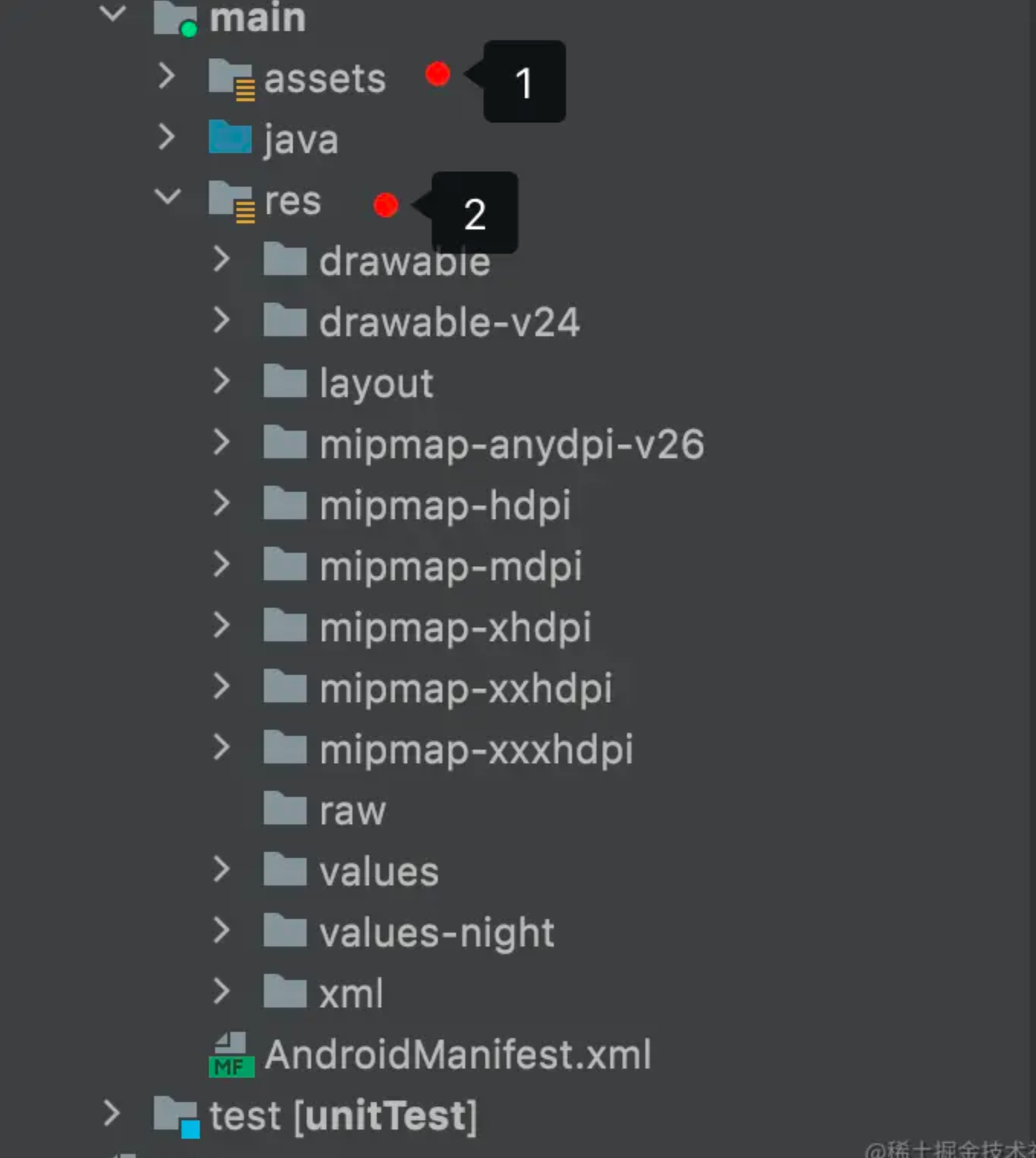
在Android项目中，资源文件一般都是放在下面的两个文件夹中，分别是assets和res，assets一般会存放一些mp4、mp3、html相对较大的文件；而res目录下则是区分很多级子目录，像布局文件（layout）、图片（drawable）等。



那么这两个文件夹下资源加载有什么区别呢？

（1）res文件夹下全部资源，系统会为他们生成一个唯一id，在XML文件中就能够拿到这个id直接访问，相信伙伴们都了解这个，这样访问速度是最块的；  
（2）assets目录下的资源需要通过AssetManager访问，系统不会为他们生成唯一id，因此访问速度会慢些，而且不能直接在XML文件中访问。

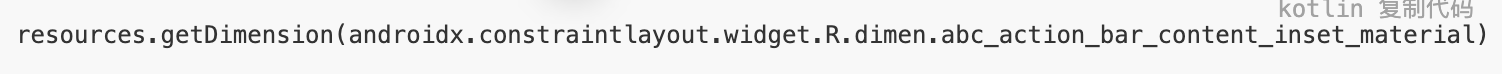
以上就是宿主app中加载资源的方式，但是如果是想要加载插件中的资源，该如何做到呢？在上一节的介绍中，虽然启动了插件中的Activity，但是布局文件并没有加载出来。

因为启动宿主app的时候，插件资源并没有被加载进来，只加载了宿主的资源，虽然启动了插件的Activity，但是因为拿到的是宿主的上下文，获取资源时只能拿到宿主的资源，无法加载插件资源。

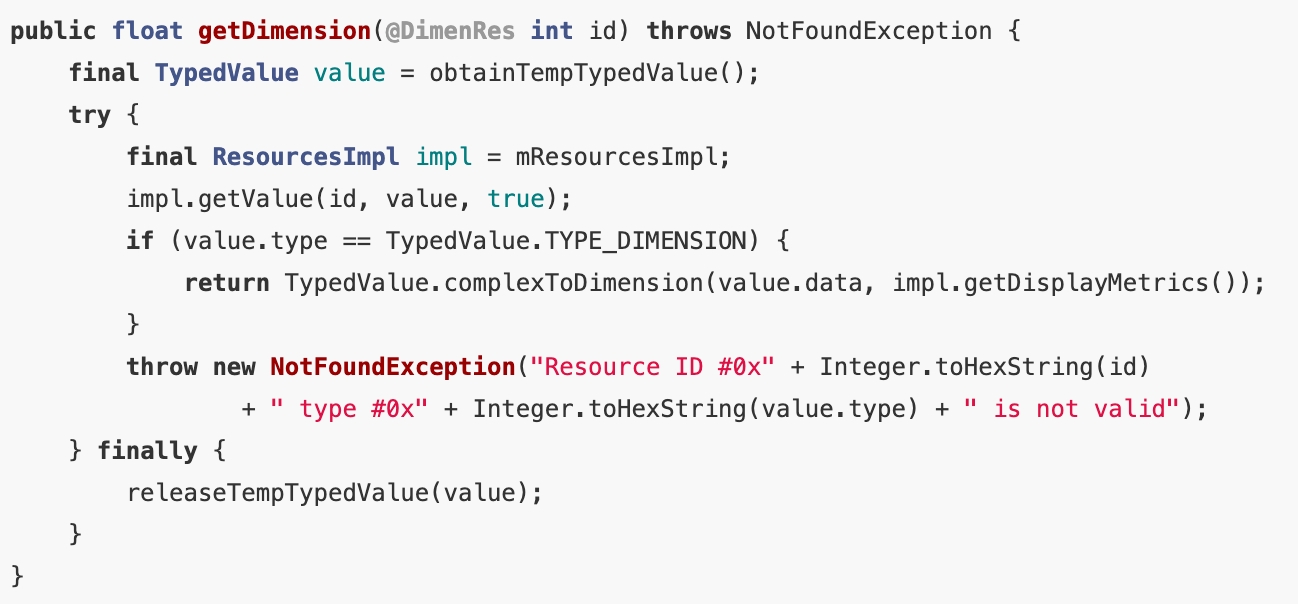
# Android资源加载流程

所以，如果想要加载插件中的资源，那么首先就需要清楚宿主app加载资源的流程，好在某个点通过hook的方式来加载插件中的资源。

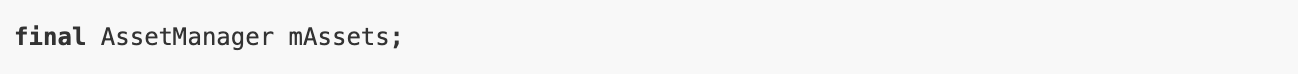
首先我们先看最简单的，获取一个尺寸的值



我们需要关注一个类，ResourcesImpl，全部的资源加载都会集中到这个类中



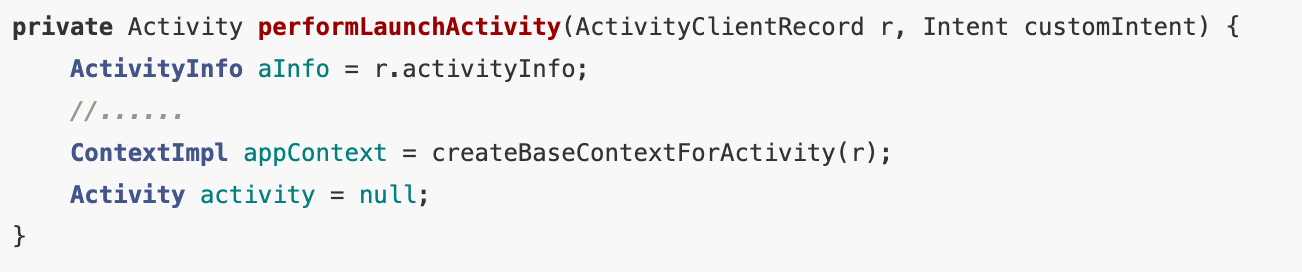
在这个类中，保存一个AssetManager对象，所有的资源加载都是通过AssetManager来实现的



## Android系统资源绑定

当我们想加载资源的时候，都是需要通过上下文context获取resource对象，无论是Application还是Activity，还是其他的组件，既然有获取，那么是在什么时候设置的resource对象呢，那么就需要从context创建开始，这里以Activity为例。

当启动一个Activity的时候，是在Activity的performLaunchActivity方法中，调用了createBaseContextForActivity方法





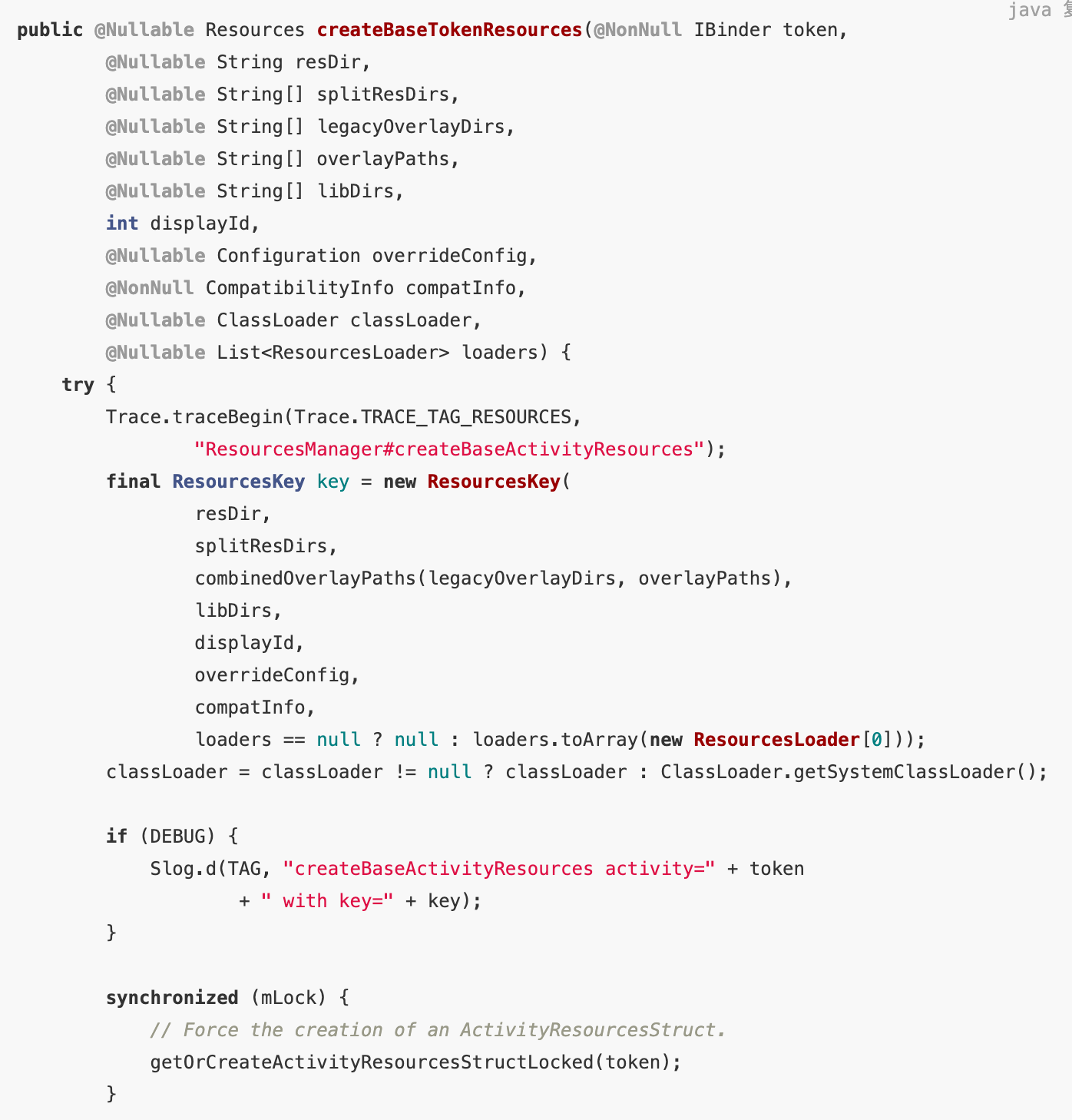
在createBaseContextForActivity方法中，通过ContextImpl实现类调用createActivityContext方法创建了上下文对象，我们继续深入





在这个方法中，创建了一个ContextImpl对象，调用了setResources方法，这里就是我们在调用getResource方法时获取到的Resource对象。

接下来我们看下，Resources对象是如何创建的，是调用了ResourceManager的createBaseTokenResources方法；





resDir，就是文章开头提到的res文件夹，通过这些入参生成了一个ResourcesKey，最终调用了createResourcesForActivity方法



在这个方法中，我们可以看到一个熟悉的类ResourcesImpl，它是通过调用findOrCreateResourcesImplForKeyLocked方法实现的



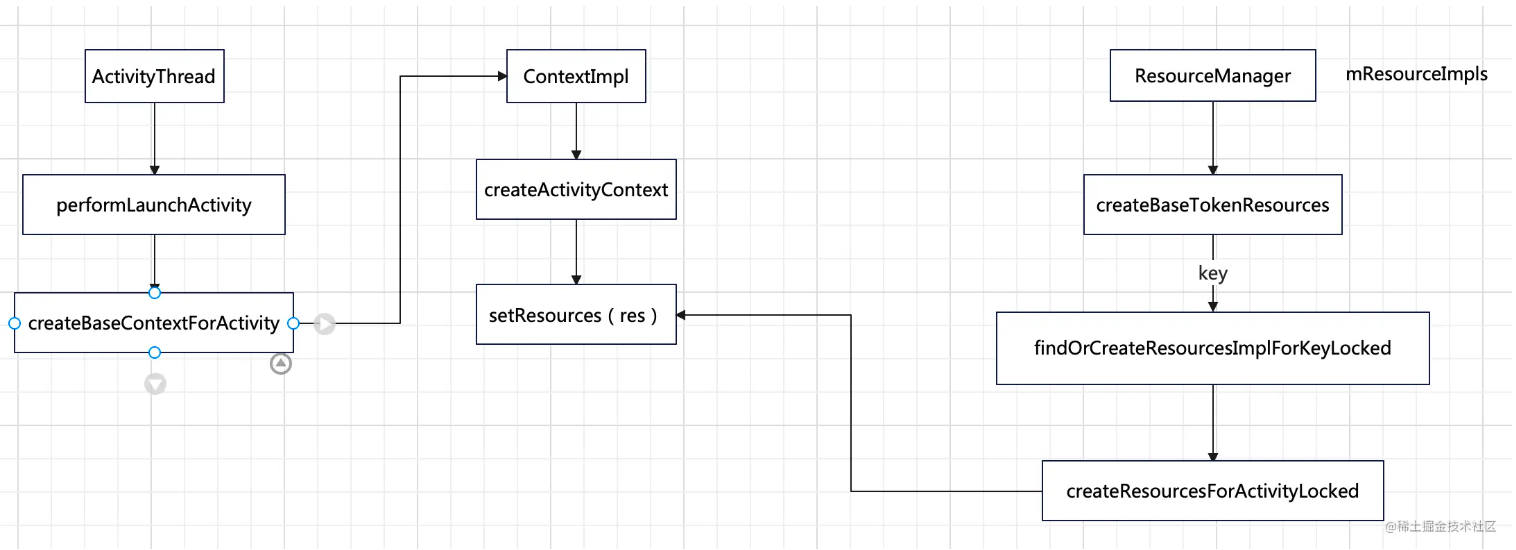
在ResourceManager中，mResourceImpls是一个map，用于存储每个key对应的ResourcesImpl，也就是说每个key只会存在一个ResourcesImpl；如果当前key不存在对应的ResourcesImpl，那么会调用createResourcesImpl创建。



在createResourcesImpl方法中，通过ResourcesKey创建一个对应的AssetManager，并存储在ResourcesImpl中，在本节一开头的时候提到，在加载资源时，主要就是通过这个AssetManager来完成，也就是跟资源key一一对应的AssetManager。



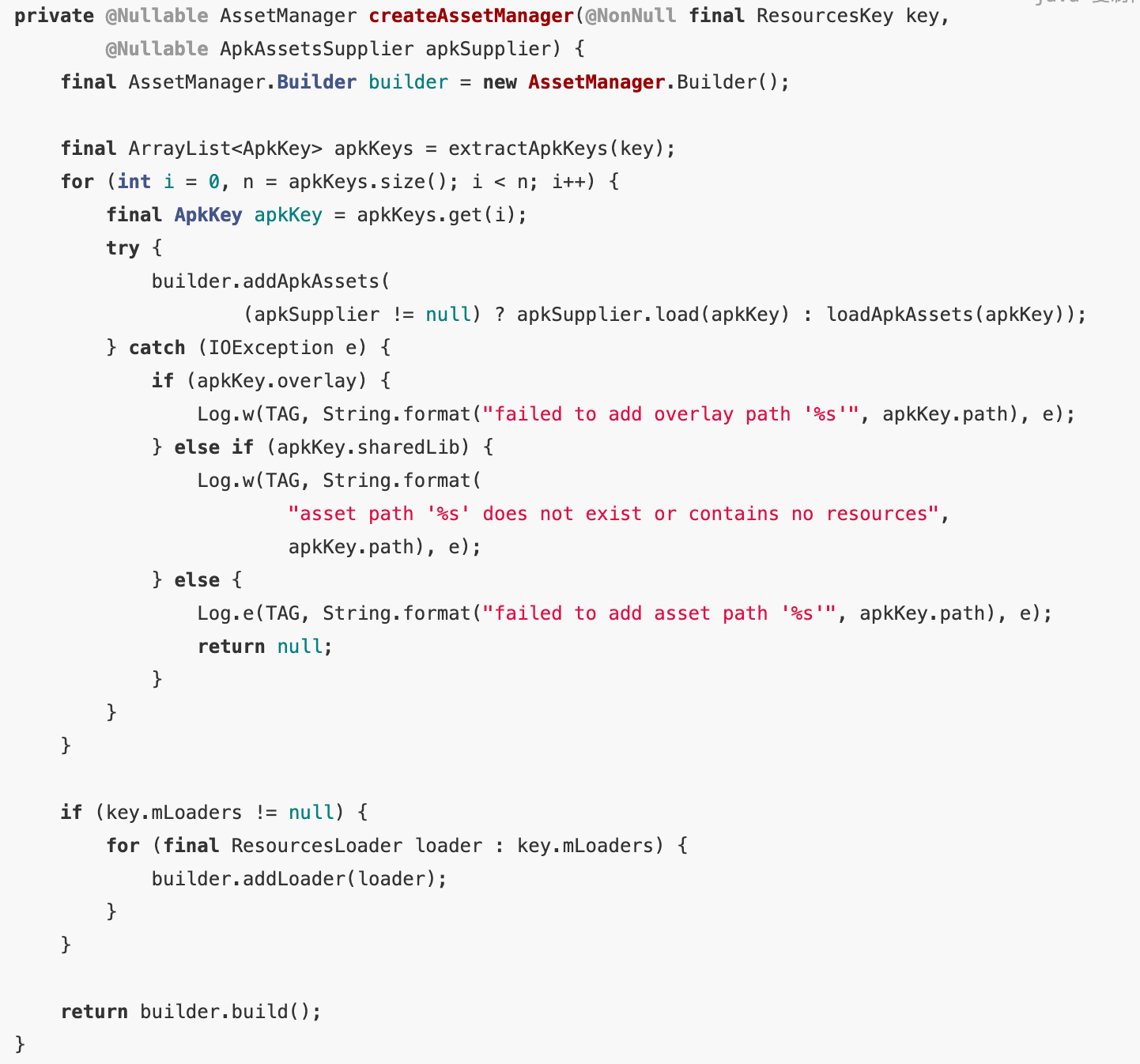
最终调用createResourcesForActivityLocked方法，将ResourcesImpl设置给了Resource。



其实原理还是比较简单，当Activity启动时，会创建context，并设置Resources对象，那么Resources对象通过ResourceManager来创建的，具体怎么创建呢？会根据App中的res资源文件生成一个ResourceKey，通过这个key去查找或者创建ResourceImpl，并存放在Resources对象中。

## 插件资源加载方案

在创建ResourceImpl时，会创建AssetManager与其绑定，然后根据ResourcesKey中资源路径，添加到AssetManager中，这样AssetManager就能够加载当前路径中的资源





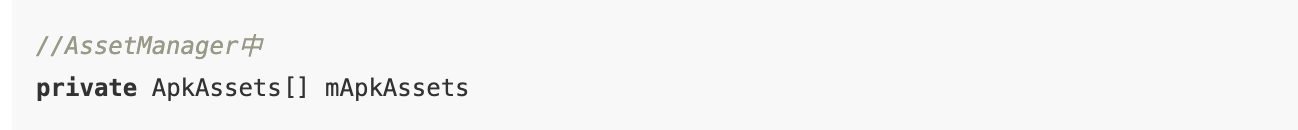
所以跟之前插件合并有些类似，dex文件存储在dexElement中，可以把插件dex跟宿主dex合并，那么资源肯定也能够合并。

（1）将插件资源与宿主资源合并，但是会带来资源冲突的问题；  
（2）创建新的Resource、AssetManager加载插件中的资源，这种方式是推荐方案。

# 插件资源加载实现

## 使用新的资源加载器加载插件资源

其中这种方式很简单，而且不会产生资源冲突。在上一节中，可以看到在创建AssetManager的时候，是调用了addApkAssets将所有资源添加到mApkAssets集合中



与之对应的添加方法就是setApkAssets，现在网上有好多使用addAssetPath这个方法的，这个方法已经过时了，虽然还能用，但是万一哪天给删除了，那就会出问题



网上大部分都是反射这个方法，我自己想直接反射setApkAssets方法，却发现拿不到这个方法，没办法，就只能先反射调用addAssetPath方法了。



通过反射获取addAssetPath方法，注意这里是带参数的方法，然后创建一个Resources对象

public static Resources loadPluginRes2(Context context, String resPath) {

try {

AssetManager assetManager = AssetManager.class.newInstance();

Method addAssetPathMethod = AssetManager.class.getDeclaredMethod("addAssetPath", String.class);

addAssetPathMethod.setAccessible(true);

addAssetPathMethod.invoke(assetManager, resPath);

return new Resources(assetManager, context.getResources().getDisplayMetrics(), context.getResources().getConfiguration());

} catch (Exception e) {

return null;

}

}

既然完成了自定义Resource，那么怎么让插件拿到这个Resource去加载资源呢？

首先在宿主的Application中，创建插件Resource对象，然后重写getResources方法，那么在插件中，通过getApplication().getResources()获取到的Resources对象就是我们自己创建的



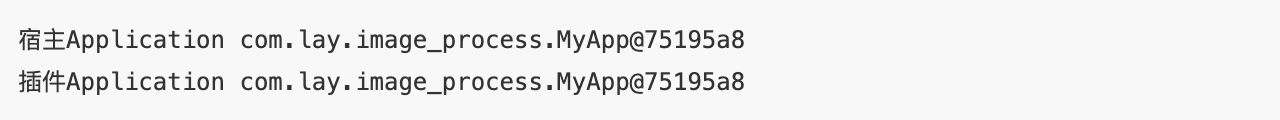
所以在插件中，这里先写了一个简单的BasePluginActivity，同样重写了getResources方法来获取Application的Resources对象。



这样一来，插件中的Activity启动了而且Layout资源被加载展示出来了；那么这么使用没有问题吗？当然有问题

## 问题解决

上述方式使用过程中的问题就是：  
（1）插件和宿主中拿到的Application是一样的，这就意味着插件中的Application不会被执行



（2）宿主中调用application.resources，拿到的是插件中的Resource

这个问题其实很好解决，其实对于资源加载，放在插件中就可以完成，只不过是需要约定好插件apk存放的地址，这样就不需要在宿主Application中做相应的逻辑处理，不会影响到宿主

open abstract class BasePluginActivity : Activity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

initView()

}

open fun initView() {}

override fun getResources(): Resources {

Log.e("TAG", "插件Application $application")

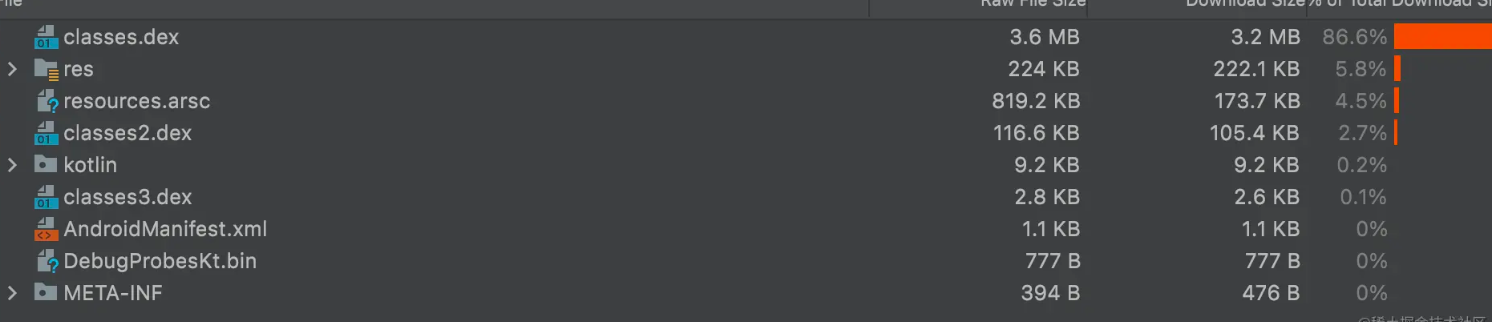
return LoadResUtil.getResources(application, "/sdcard/plugin-debug.apk") ?: super.getResources()

}

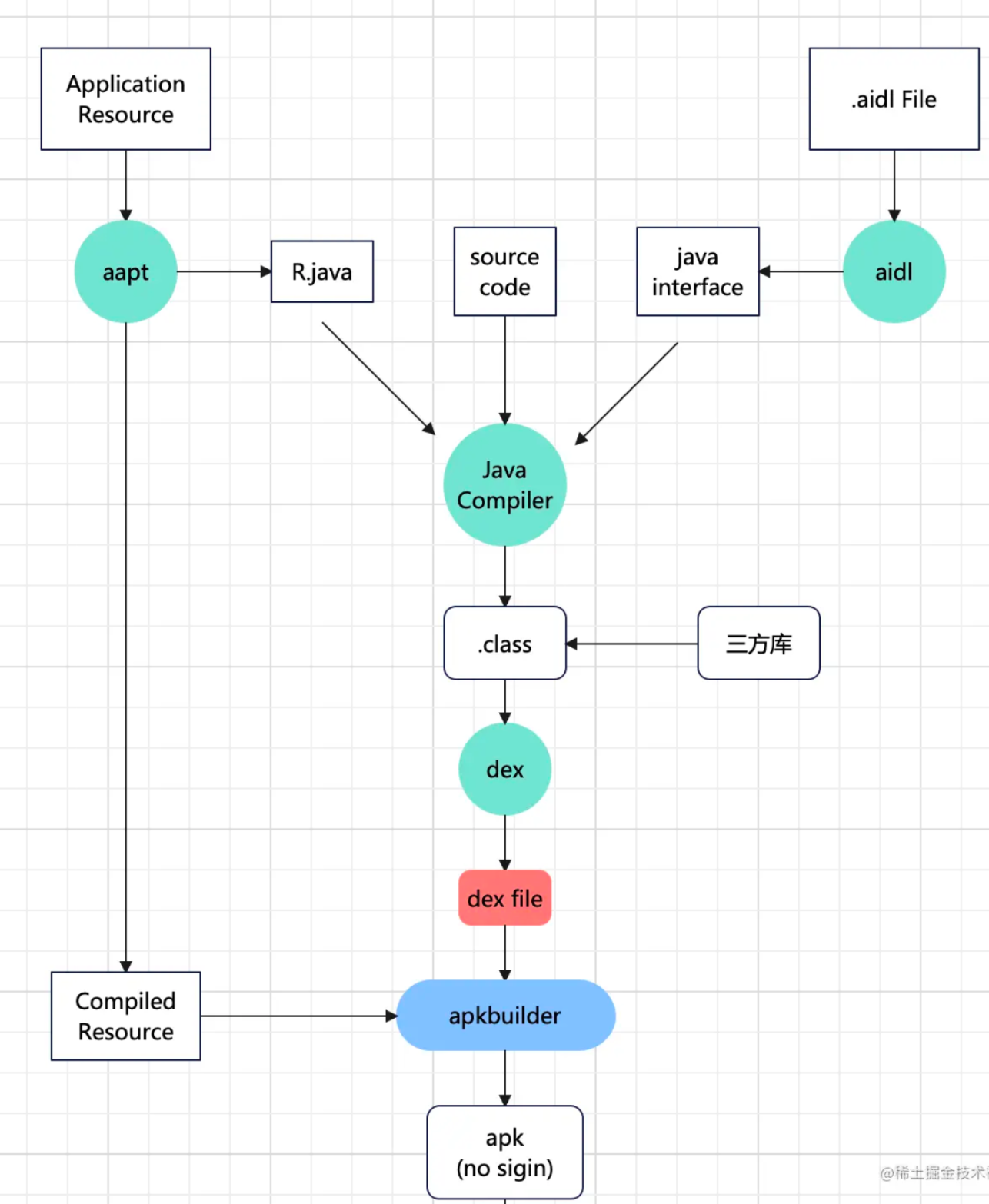
}

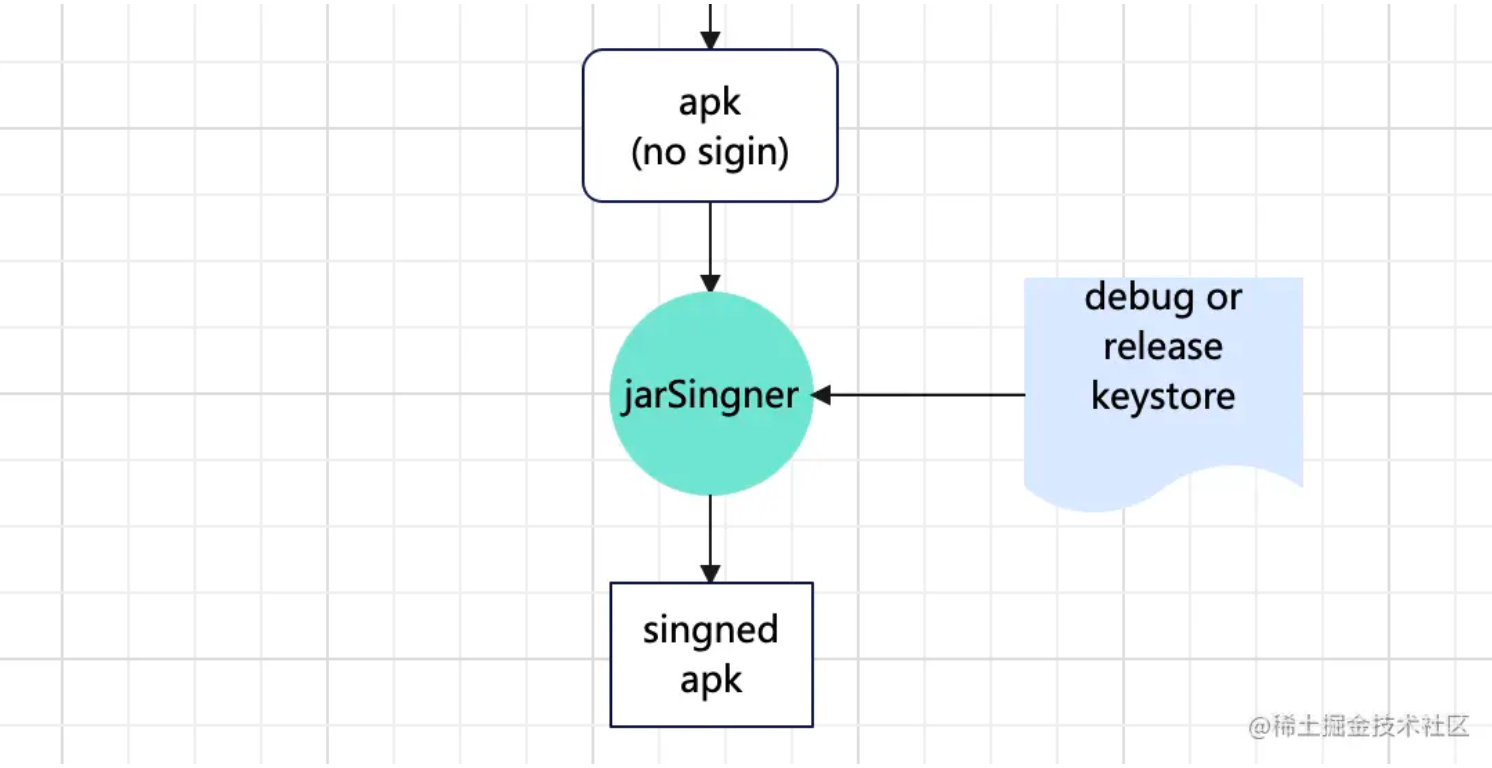
## aapt打包流程

如果想要将插件资源和宿主资源合并打包，那么首先就需要对Android APK的打包流程熟悉，在Android中资源打包工具为Android Asset Packging Tool，简称就是aapt



上面这张图就是打包完成的apk，接下来我们通过流程图来看下，我们一个Android工程是如何打包成一个apk的。





（1）首先Android中的资源Resource（res文件夹下的）通过aapt工具生成R.java文件和.arsc文件；如果项目中使用到了aidl，那么也会把aidl文件转换成Java接口；  
（2）然后，R.java、源码、aidl java接口将会被javac工具编译成.class文件；  
（3）将项目中的.class文件跟三方库的Lib和.class文件一起，通过dex工具转换为dex文件，这个在之前我们使用dx命令行将class文件转换为dex文件，其实原理一致；  
（4）最终通过apkbuilder工具，将没有编译过的资源文件（像asset目录下的）、dex文件一起打包成一个没有签名的apk；  
（5）通过JarSigner工具，配合keystore打包成带签名的文件。

这就是整个apk的打包流程。那么我们需要hook framework层代码去实现吗？显然不可行，只能说可行性很低，因此我们需要关注的就是第一种方式。

## 资源冲突问题解决

爬一下楼，看下BasePluginActivity这个类，我们继承的是Activity，那么如果我们换成AppCompactActivity呢，报错了！

Caused by: java.lang.IllegalArgumentException:

AppCompat does not support the current theme features:

{ windowActionBar: false, windowActionBarOverlay: false, android:windowIsFloating: false, windowActionModeOverlay: false, windowNoTitle: false }

那么在什么地方报的错呢？看下源码



我们可以看到就是，当subDecor为空的时候，会报错；这就意味着前面在拿layout/abc\_screen\_simple\_overlay\_action\_mode或者layout/abc\_screen\_simple这两个资源的时候没有拿到，为什么在插件中没有拿到这个资源呢？

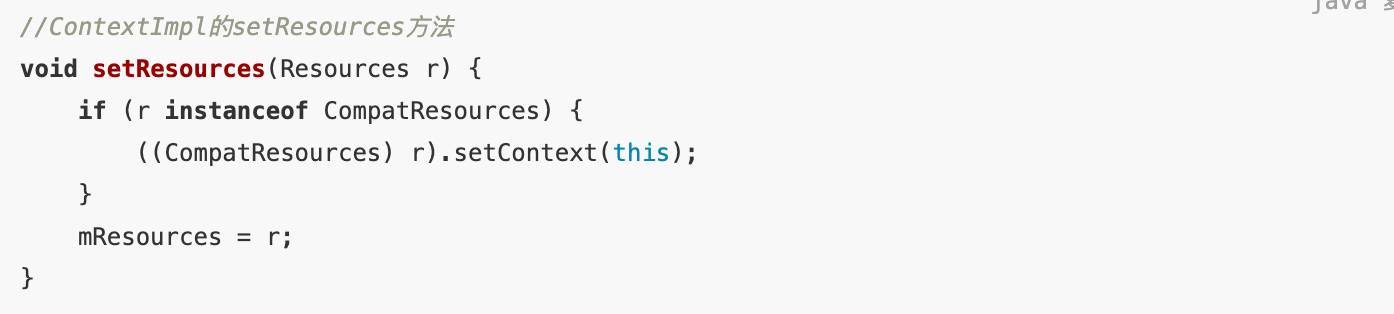
首先我们从前面打包流程说起，一个apk打包时，首先会将res文件夹下资源通过aapt生成R.java文件，以及.arsc文件，并且为每个资源生成唯一的id，当加载资源的时候，其实就是加载这个id，那么为什么会找不到？其实就是双亲委派机制捣的鬼。

AppCompactActivity是Android SDK中的类，当这个类加载的时候，首先会加载宿主的，这是毫无疑问的；但是当插件Activity启动的时候，其实获取到的还是宿主中的class，因为绑定的是宿主的上下文，那么在加载资源的时候，还是会从宿主的Resoure中加载系统资源，而不是插件中的系统资源，最终导致了资源获取不到报错！

所以，我们可以创建插件独有的Context，并与插件资源绑定，这样插件启动后加载的就是插件中的系统资源及apk资源。我们在前面的源码中能够看到，当Activity启动之后，创建了Context，并把资源做了绑定



我们可以看到，Context调用setResources方法，就是将mResources属性赋值，因此可以反射获取这个mResources属性，将其赋值为插件中的资源



这样，我们再把插件的BasePluginActivity做一次修改，创建我们自己的context对象，把插件中的资源赋值给了Context的resource



既然我们的插件Context创建完成了，那么在页面中就不能再使用这种 ****R.layout.activity\_main****的使用方式，因为这个时候还是调用宿主上下文中的R文件，会有资源冲突，需要使用下面这种方式



这个时候，我们再启动插件Activity，发现就不会报错了。