内容目录

[1 如何实现Hook系统api 1](#__RefHeading___Toc128_1050277061)

[1.1 hook系统服务接口类图 1](#__RefHeading___Toc130_1050277061)

[1.2 hook 流程 1](#__RefHeading___Toc132_1050277061)

[1.3 客户端获取服务代理对象过程 2](#__RefHeading___Toc134_1050277061)

[1.4 客户端调用具体方法处理流程 2](#__RefHeading___Toc136_1050277061)

[2 如何实现插件的资源加载 3](#__RefHeading___Toc138_1050277061)

[2.1资源解析过程 3](#__RefHeading___Toc140_1050277061)

[2.2 生成ContextImpl对象过程 3](#__RefHeading___Toc142_1050277061)

[2.3 生成ContextImpl.mResources过程 3](#__RefHeading___Toc144_1050277061)

[2.4 hook过程 5](#__RefHeading___Toc146_1050277061)

[3 插件运行的进程如何管理，如何在插件运行在新进程后吧插件也hook掉 5](#__RefHeading___Toc148_1050277061)

[3.1 插件进程的hook过程 5](#__RefHeading___Toc150_1050277061)

[3.2 插件进程的管理 6](#__RefHeading___Toc152_1050277061)

[4 如何建立的插件程序沙箱 7](#__RefHeading___Toc154_1050277061)

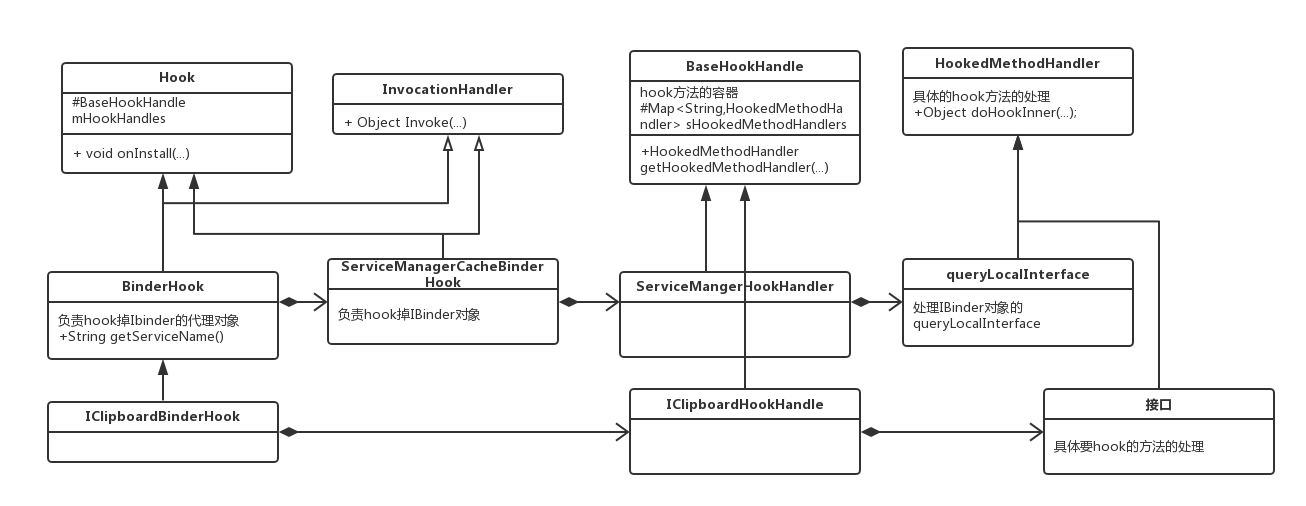
[4.1 沙箱路径的管理 7](#__RefHeading___Toc156_1050277061)

[4.2 如何让用户用户调用mContext.getDataDir()，mContext.getFilesDir();，mContext.getCacheDir();时访问到正确的沙箱路径。 7](#__RefHeading___Toc158_1050277061)

[4.3 如何让用户直接访问”/data/data/包名/文件名” 的时候访问到正确的路径 8](#__RefHeading___Toc160_1050277061)

**1 如何实现Hook系统api**

**1.1 hook系统服务接口类图**

**1**

hook一个系统服务的思路在于,首先所有系统服务都在ServiceManager的静态sCache映射表里面。所以只要想法子代理掉这些映射表里面的对象就可以实现hook，因为所有服务在这个映射表里面都是IBinder的实例,所以可以采用动态代理,生成一个IBinder动态代理对象，就可以实现代理这个IBinder实例了。而客户端获取到的实例都是通过IXXXService.Stub.asInterface(IBinder)这个接口生成的代理服务对象，而这个接口的实现如下：

static public IXXXService asInterface(IBinder obj) {

if (obj == null) {

return null;

}

IXXXService in =

(IXXXService)obj.queryLocalInterface(descriptor);

if (in != null) {

return in;

}

return new IXXXServiceProxy(obj);

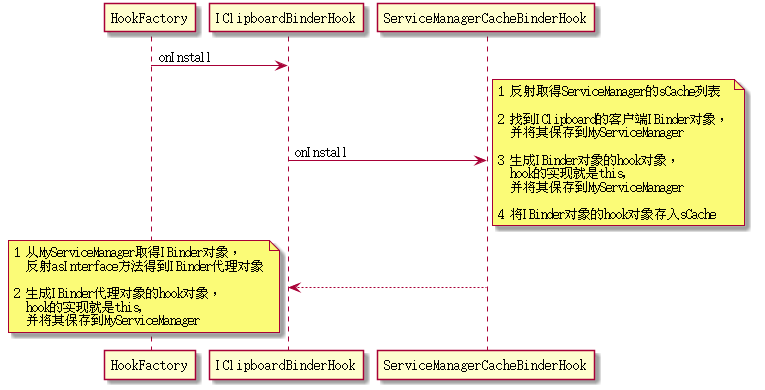
}

会先向IBinder对象调用 queryLocalInterface询问是不是有本地对象，这个函数的作用在于，

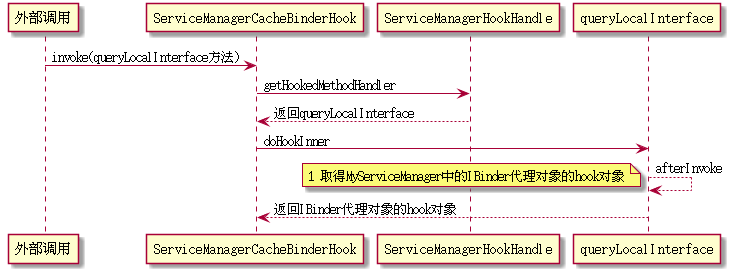
在服务端的时候这个IBinder会返回服务端的实例，而客户端默认直接返回null。

因此只要吧IBinder的 queryLocalInterface给hook掉然后返回一个 IXXXServiceProxy的代理对象就可以成功的吧客户端拿到的服务端代理对象成功hook了。

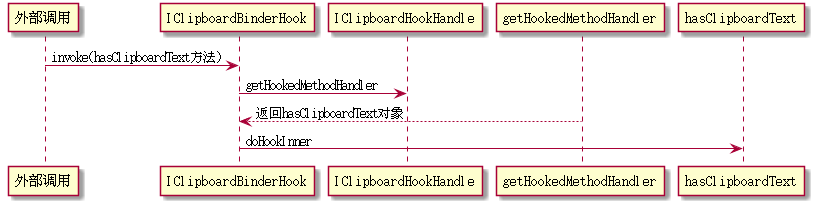
DroidPlugin具体方式是，首先调用ServiceMangerCacheBinder的onInstall的方法，这一步主要是拿到sCache表里面的目标服务的IBinder对象。然后生成这个对象的代理对象,这里使用的是动态代理，并且这个动态代理的InvocationHandler实现对象就是ServiceMangerCacheBinder对象本身，因此所有IBinder代理对象的函数都会转发给其invoke()函数实现，ServiceMangerCacheBinder会首先根据invoke传递进来的方法名字在其成员函数ServiceManagerHookHandler（这个是具体方法实现的容器）里面找到具体的方法实现方法的对象，譬如当改IBinder的queryLocalInterface函数被调用的时候找到的就是一个queryLocalInterface(这个是一个hook方法的实现类哦)对象，这个对象实现了真正的queryLocalInterface函数，他返回了一个 IXXXServiceProxy对象的代理对象，当然也是动态代理哦，这个动态代理的invocationHandler对象是BinderHook对象，因此当代理对象的函数被调用的时候BinderHook的invoke函数会被调用，BinderHook的invoke函数完成具体实现的转发

**1.2 hook 流程**

**1.3 客户端获取服务代理对象过程**



**1.4 客户端调用具体方法处理流程**



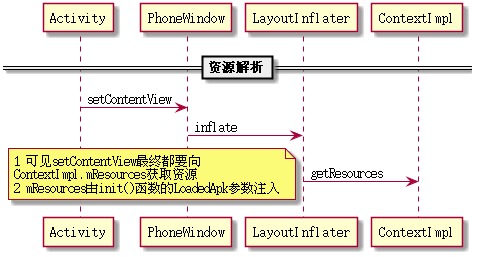
所有的hook类在com.morgoo.droidplugin.hook.proxy包和com.morgoo.droidplugin.hook.binder

包下面

所有hook方法的具体处理在com.morgoo.droidplugin.hook.handle包下面

**2 如何实现插件的资源加载**

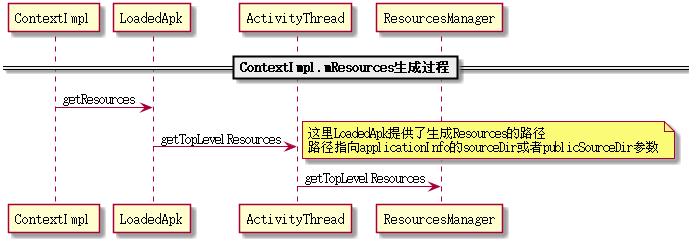
**2.1资源解析过程**



**2.2 生成ContextImpl对象过程**



**2.3 生成ContextImpl.mResources过程**



以下是ReSourcesmanager的生成 Resources的代码

public Resources getTopLevelResources(String resDir, int displayId,

Configuration overrideConfiguration, CompatibilityInfo compatInfo, IBinder token) {

…//省略无关代码

**AssetManager assets = new AssetManager();**

**if (assets.addAssetPath(resDir) == 0) {**

**return null;**

**}**

DisplayMetrics dm = getDisplayMetricsLocked(displayId);

Configuration config;

boolean isDefaultDisplay = (displayId == Display.DEFAULT\_DISPLAY);

final boolean hasOverrideConfig = key.hasOverrideConfiguration();

if (!isDefaultDisplay || hasOverrideConfig) {

config = new Configuration(getConfiguration());

if (!isDefaultDisplay) {

applyNonDefaultDisplayMetricsToConfigurationLocked(dm, config);

}

if (hasOverrideConfig) {

config.updateFrom(key.mOverrideConfiguration);

}

} else {

config = getConfiguration();

}

**r = new Resources(assets, dm, config, compatInfo, token);**

…//省略无关代码

return r;

}

我们可以得出：

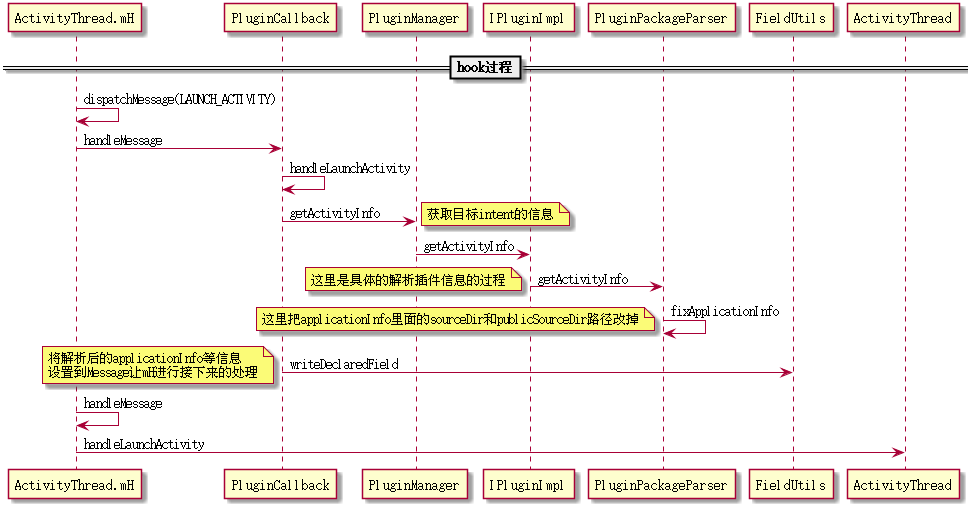
1 要获取资源，首先是从 Resources得到解析器

2 要生成Resources对象，先生成**AssetManager对象然后调用 addAssetPath把对应插件的apk路径传入就好了**

**3 而这个apk路径是从****LoadedApk得到的，LoadedApk又是通过ApplicationInfo得到这个路径的，所以只要hook这个对象并且更改相应路径就好了**

**在DlPlugin插件框架中，插件的资源加载是通过反射生成AssetManager并添加插件apk的路径来的AssetManager对象和Resources对象，然后重载了插件Activity的getAssetManager和getResources等方法，然后把这个反射生成的对象返回实现资源加载**

**2.4 hook过程**

****

由 **Android 插件化原理解析4——Activity生命周期管理 文档，我们知道为了实现借尸还魂这一步，也就是将替身activity的intent改造目标activity的intent需要我们hook掉Activity.mH方法。**

而PluginCallback就是hook掉之后的具体处理过程。

在这里我们进行目标activity信息的具体解析，把生成AssetManager所需要的关键路径更改掉！！

**3 插件运行的进程如何管理，如何在插件运行在新进程后吧插件也hook掉**

**3.1 插件进程的hook过程**

**DroidPlugin为了实现管理插件组件的生命周期，采用的是占坑的法子**，因此可以看到在AndriodMenifest.xml里面注册了很多的stubactivity stubservice stubcontentprovicer

默认的占坑数量为

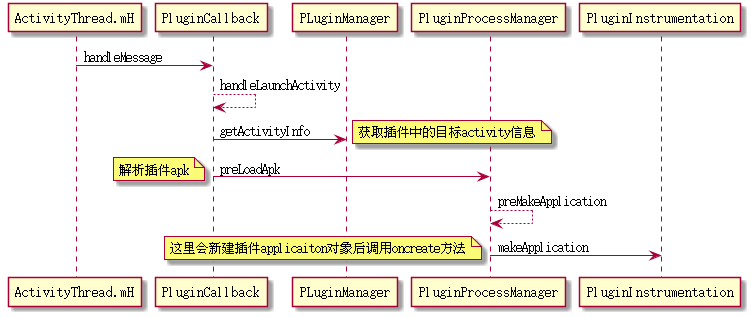
**总共申请了8个进程从PluginP00到PluginP08**

**每个进程有1个standard类型的activity其他的singletask,singleinstance,singletop类型的activity各三个，有1个service 1个provider**

启动activity首先要进行 **偷梁换柱**这一步，也就是吧**intent信息里面的目标activity信息转换为替身activity信息**，这一步是在IActivityManagerHookHandle.startActivity里面进行的

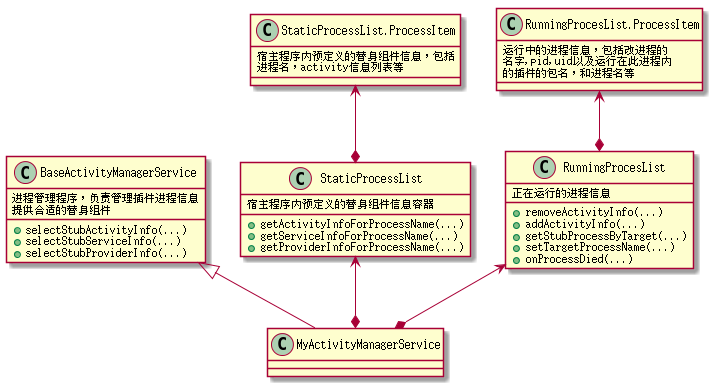
进行偷梁换柱之后，传到AMS的信息就都属于替身Activity的信息了，**假如这个替身是进程PluginP00里的一个activity，那么如果还没有运行这个进程，则AMS会先发消息给Zygote进程让其fork出来一个PluginP00进程，然后调用ActivityThread里的main函数，然后main函数就会新建一个application对象，注意，这个时候的application对象其实还是宿主程序内的application，因为传到AMS的信息还是属于宿主程序内的替身activity的，而宿主程序的application 的onCreate和attachBaseContext里面会进行插件程序的初始化，因此，插件进程也就这样被hook了**

需要注意的是，因为在插件进程运行后并没有自动新建插件的application对象，因此，需要在**借尸还魂**这一步**手动生成这个对象并回调onCreate,**



**3.2 插件进程的管理**

插件进程的管理逻辑都在MyActivityManagerService里面

大概的结构如上，当进行借尸还魂这一步的时候，会从MyActvitiyMangerService里面抽取合适的替身，如果都被占用则会抛出异常

当组件运行成功，并且进行onCreate onDestory的时候会调用MyActvivityManagerService相应函数进程信息记录。这一步是在PluginInstrumentation里面做的

插件进程启动的时候就会start并且bind 插件管理服务PluginManagerService 并注册了一个IApplicationCallback回调给PluginManagerService 当插件进程异常关闭的时候，PluginManagerService可以通过这个回调来检测到链接断开了，然后通知RunningProcessList移除相应进程

**4 如何建立的插件程序沙箱**

**4.1 沙箱路径的管理**

PluginDirHelper专门负责沙箱路径的管理

插件的目录结构如下：

*\* 插件目录结构*

*\* 基本目录： /data/data/com.HOST.PACKAGE/Plugin*

*\* 单个插件的基本目录： /data/data/com.HOST.PACKAGE/Plugin*

*\* source\_dir： /data/data/com.HOST.PACKAGE/Plugin/PLUGIN.PKG/apk/base-1.apk*

*\* 数据目录： /data/data/com.HOST.PACKAGE/Plugin/PLUGIN.PKG/data/PLUGIN.PKG*

*\* dex缓存目录： /data/data/com.HOST.PACKAGE/Plugin/PLUGIN.PKG/dalvik-cache/*

**4.2 如何让用户用户调用mContext.getDataDir()，mContext.getFilesDir();，mContext.getCacheDir();时访问到正确的沙箱路径。**

mContext.getDataDir()最终访问到的是LoadedApk的mDataDir路径，而其他两个路径都是基于这个路径建立的子目录，而LoadedApk.mDataDir访问到的是ApplicationInfo.dataDir路径

在PluginPackageParser解析插件的activity等组件信息的时候会调用fixApplicationInfo将这个路径指向正确的路径

**4.3 如何让用户直接访问”/data/data/包名/文件名” 的时候访问到正确的路径**

DroidPlugin框架hook了libcore.io.Libcore的Os成员变量，这个变量是是所有java层进行io操作最终要调用这个变量的接口，

处理方法都在LibCoreHookHandle这个类里面，所有需要访问文件的接口都会调用tryReplacePath

这个函数将路径/data/data/插件包名/xxx 替换成/data/data/宿主包名/Plugin/插件包名/data/插件包名