[插件开发之360 DroidPlugin源码分析（五）Service预注册占坑](http://blog.csdn.net/hejjunlin/article/details/52264977)

在了解系统的activity，service，broadcastReceiver的启动过程后，今天将分析下360 DroidPlugin是如何预注册占坑的？本篇文章主要分析Service预注册占坑，Service占了坑后又是什么时候开始瞒天过海欺骗AMS的？先看下Agenda：

* **AndroidMainfest.xml中概览**
* **Service中关键方法被hook时机**
* **startService被hook**
* **瞒天过海流程图**
* **认识ServiceManager**

## AndroidMainfest.xml中概览



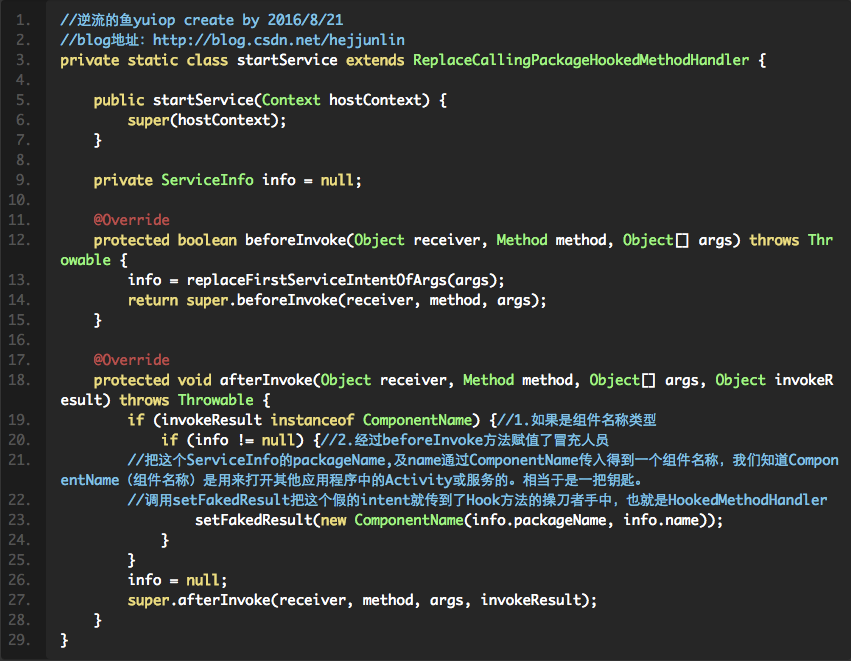
android:process=”:PluginP02” 表示自定义一个名为PluginP02的进程，该属性一旦设置, 那么App启动之后, Service肯定会以远程服务方式启动, —- 可通过adb shell ps 就可以看到有一个独立的进程启动了.

项目 Package为 com.morgoo.droidplugin , 并设置 android:process=”:PluginP02”, 那么App运行时, 通过 adb shell ps | grep com.morgoo.droidplugin , 可以看到有两个进程:

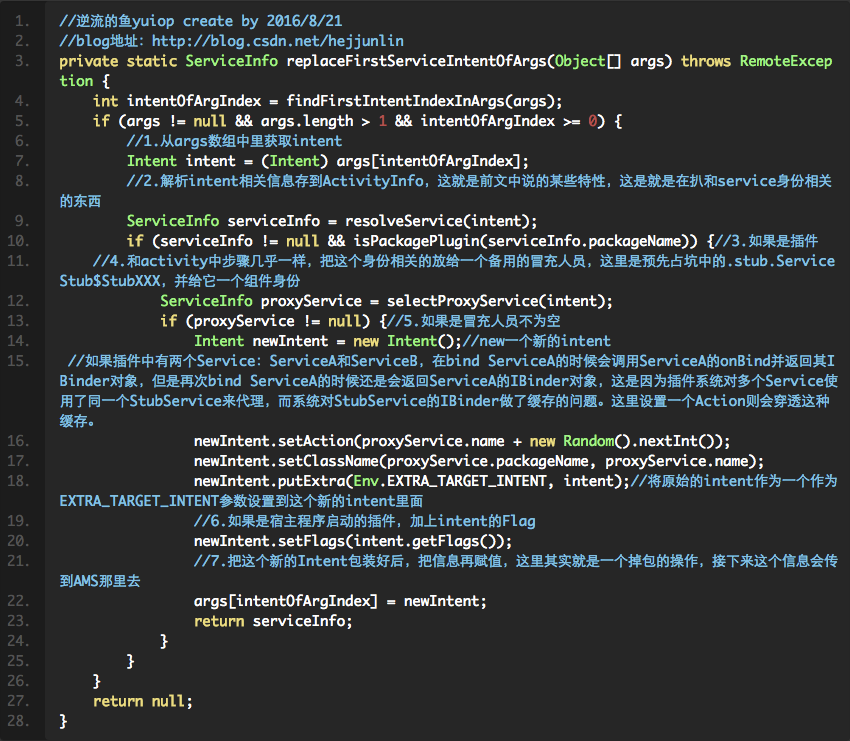
IMG_257

## startService被hook

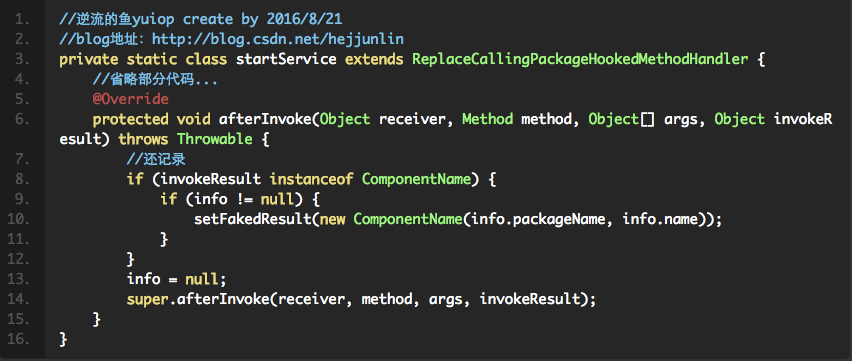
IActivityManagerHookHandle.startService方法



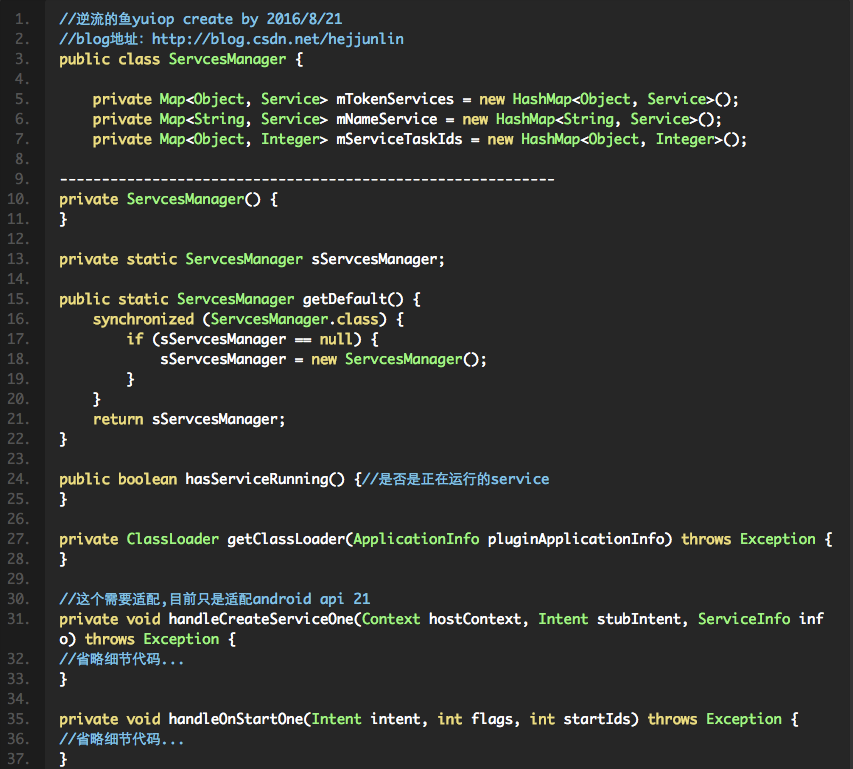
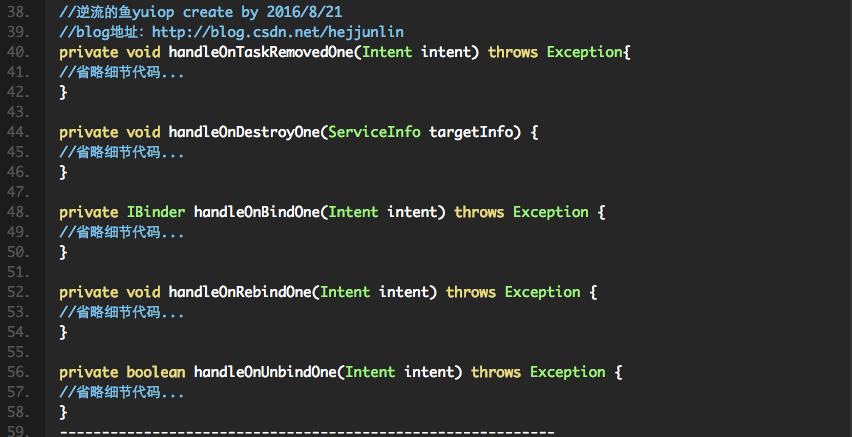
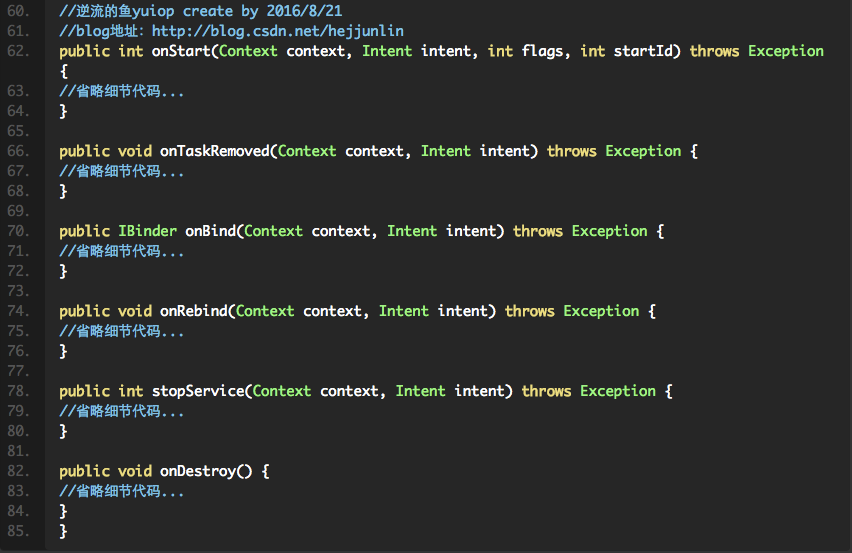
replaceFirstServiceIntentOfArgs()



接下来我们继续回到StartService类中，看下afterInvoke方法



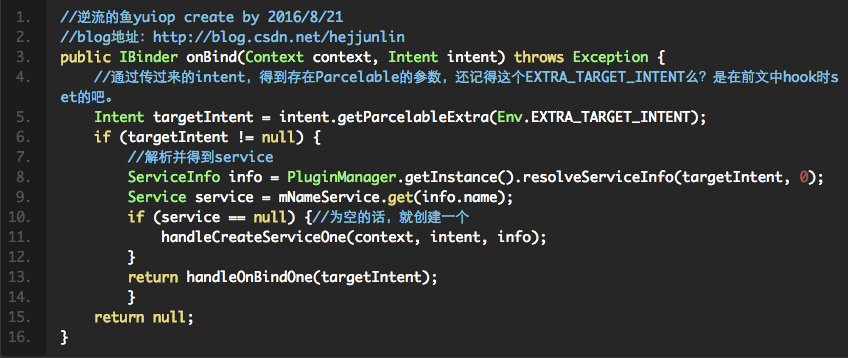
stopService,bindService,unbindService，setServiceForeground都是类似逻辑，可从源码中证实。   
在前一篇[《插件开发之360 DroidPlugin源码分析（四）Activity预注册占坑》](http://blog.csdn.net/hejjunlin/article/details/52258434)，我们曾了解到，8个进程中的service在预注册占坑，在AndroidManifest.xml中只有一个，那如果插件要启动多个service怎么办？这是第一个问题，而我们知道所有和服务相关的都要在系统的systemServer进程中进行注册，难道DroidPlugin要逆天的本事？这是第二个问题   
我们从代码中发现有一个ServceManager，这和android系统中大管家ServiceManager相差一个i，难道要改大管家的职责？又多了一个疑惑   
仔细看了下ServceManager中的代码：

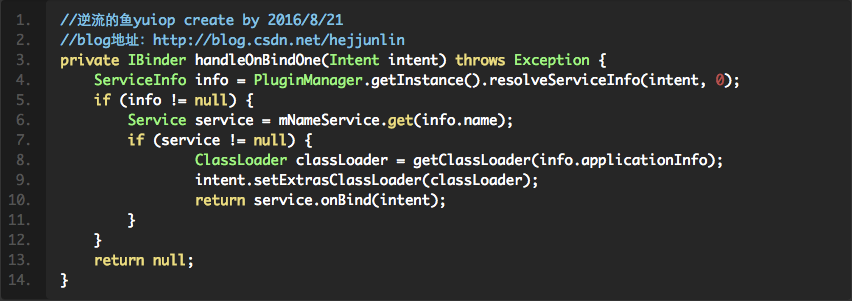
以上代码可总结为：（以虚线分上下两部分）   
1.上部分主要是ServceManager单例及有一个handleXXX相关的方法，下部分主要是Service的一些生命周期方法。   
2.我们基本可以确定之前问题2和问题3的答案了，没有那么逆天的本事来操作ServiceManager，也不能修改大管家ServiceManager的职责   
3.那问题又来了，这个类到底是做啥的？还起一个叫ServceManager的名字，   
以hanldeOnTaskRemoveOne方法为例看看：



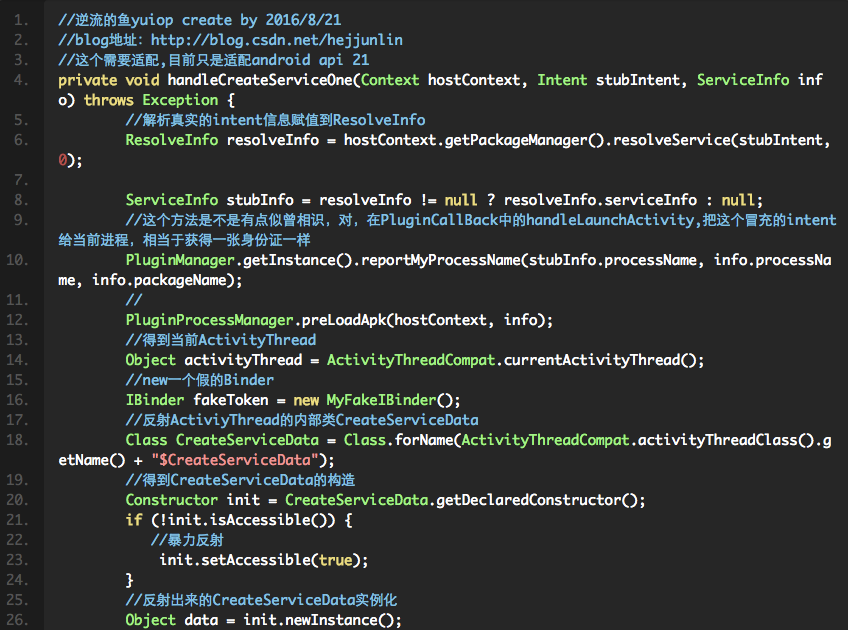
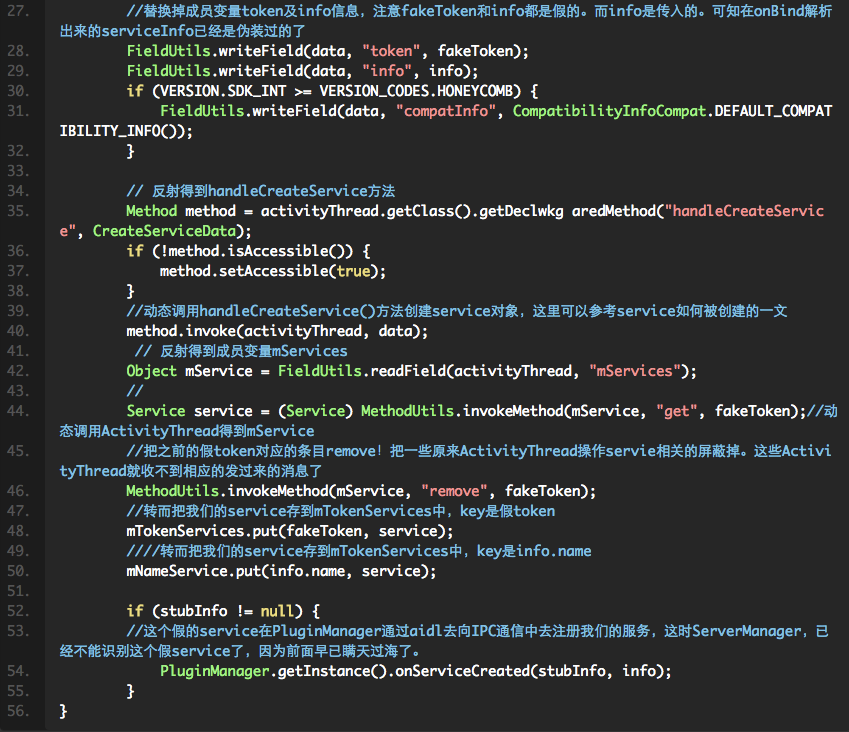
接着再看下ServceManager中的onBind方法：

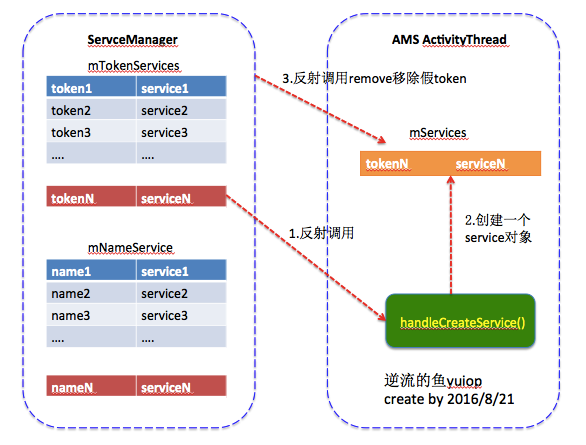


然后调到handleOnBindOne方法中，我们前面分析了一个handleOnTaskRemovedOne方法，下面1-7步骤套路都是一样，主要在return时，直接调用service自身的onBind了，这和我们平时在Activity中new Intent,然后把这个intent传到onBind中.



那么问题来了，还要这么一个类倒腾来干啥呢？还记得上面有一处逻辑service为null时，就调用了handleCreateServiceOne，好戏在这里：

看了上面的分析后，我们再去回想第一个问题，那如果插件要启动多个service怎么办？不知道不没有注意到mTokenServices，mNameService，mServiceTaskIds这些成员变量，它们都是Map，key不一样，value全是Service，如个有多个service在插件中启动了，mTokenServices，mNameService，mServiceTaskIds这些成员变量分别掌握了Service的Token,name，及任务id（通过token拿到的）。那岂不是已经在管理这些多个service了。至于附属于这个类起的名字，叫什么都无所谓。   
以上可用如下流程图表示：   


前面一直有个问题解释的不彻底，就是问题3，ServceManager是否担当了修改系统大管家ServiceManager的职责？接下来，我们就稍微了解下系统大管家ServiceManager是做什么的？（PS: ServiceManager和Binder机制一样，不是一天两个就能研究的清楚的）

## 认识ServiceManager

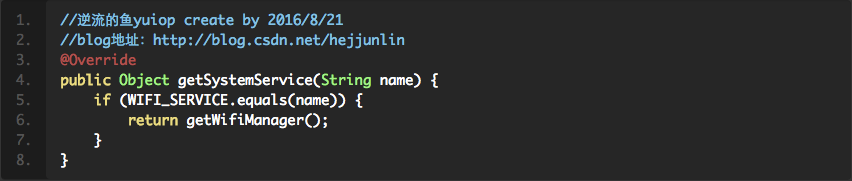
Android系统Binder机制的总管是ServiceManager，所有的Server（System Server）都需要向它注册，应用程序需要向其查询相应的服务。平时，我们在studio上的DDMS中调试程序时，下图这个就是ServiceManager

这里以Java层加入ServiceManager及getService为数据流分析一下。

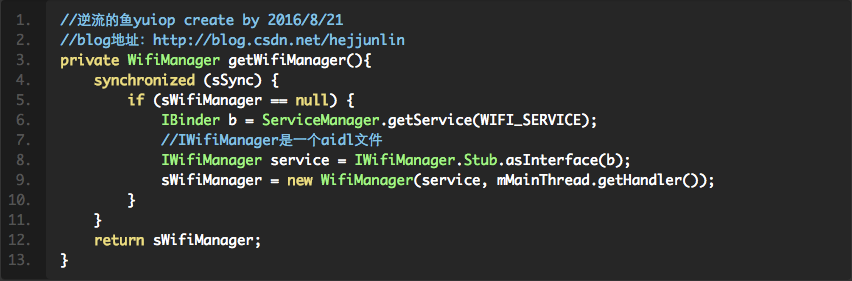
复习一下典型的Binder模式,有利于后面的理解：   
1、客户端通过某种方式得到服务器端的代理对象。从客户端角度看来代理对象和他的本地对象没有什么差别。它可以像其他本地对象一样调用其方法，访问其变量。   
2、客户端通过调用服务器代理对象的方法向服务器端发送请求。   
3、代理对象把用户请求通过Android内核（Linux内核）的Binder驱动发送到服务器进程。   
4、服务器进程处理用户请求，并通过Android内核（Linux内核）的Binder驱动返回处理结果给客户端的服务器代理对象。   
5、客户端收到服务器端的返回结果。

JAVA层代码分析：   
ServiceManager.java (frameworks\base\core\java\android\os)

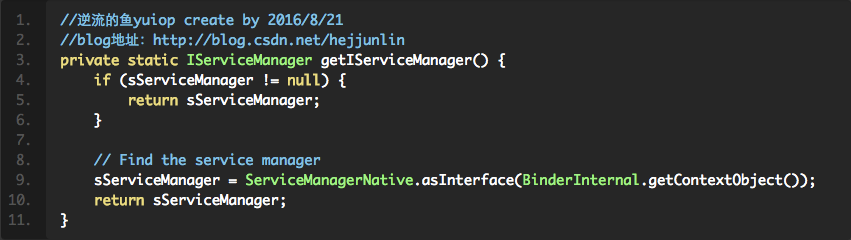
对于xxxManager获取服务端service基本如此用法：   
举例：   
利用ContextImpl.java中的 public Object getSystemService(String name)



然后再调用：ServiceManager.getService(ServiceName);获取相应的服务端



知道了客户端获取一个Service的方法之后，我们回到ServiceManager的服务端：



BinderInternal.getContextObject() @ BinderInternal.java 是一个native 函数：

android\_os\_BinderInternal\_getContextObject @ android\_util\_Binder.cpp   
返回一个 BinderProxy对象保存到类成员mRemote(ServiceManagerProxy类成员)

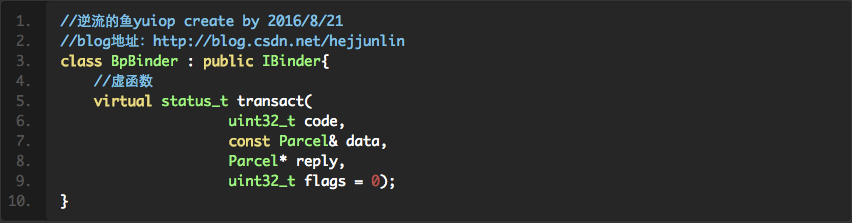
public abstract class ServiceManagerNative extends Binder implements IServiceManager   
ServiceManagerNative 继承自 Binder 并实现了 IServiceManager 接口，利用 asInterface()则提供一个 ServiceManagerProxy 代理对象使用

class ServiceManagerProxy implements IServiceManager   
定义了类ServiceManagerProxy（代理），ServiceManagerProxy继承自IServiceManager，并实现了其声明的操作函数，只会被ServiceManagerNative创建，它实现了IServiceManager的接口，IServiceManager提供了getService和addService两个成员函数来管理系统中的Service。

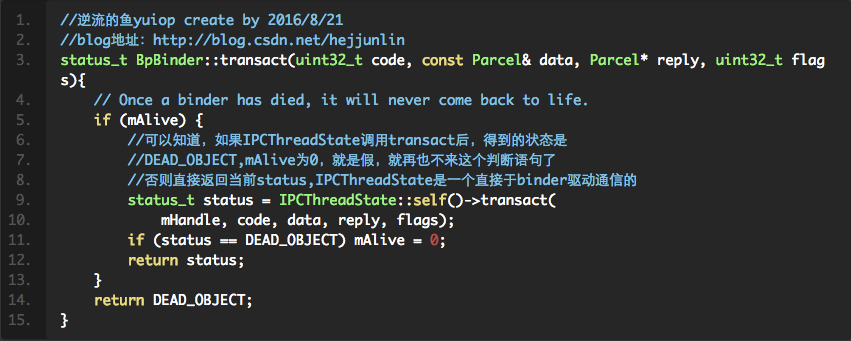


上面代码总结如下：Java层先利用Parcel对象将数据进行序列化，然后利用transact将数据传给binder驱动，就是上面mRemote.transact，mRemote在JNI层是一个叫BpBinder的对象：

JNI层代码分析：   
android\_util\_Binder.cpp



每当我们利用BpBinder的transact()函数发起一次跨进程事务时，其内部其实是调用IPCThreadState对象的transact()。BpBinder的transact()代码如下：



到此，不再向下深究，具体想了解可以看Binder机制及IPC相关内容。   
所以，到这把第2个问题彻底弄明白了，ServceManager和ServiceManager根本不是一回事。到次，Service的瞒天过海得以实现，接着，就是介绍时候说的那样无需修改源码，多进程，多service等功能。另外ContentProvider及BroadCast原理是类似，不再进行分析。   
下篇将开始分析包管理(宿主插件和插件包名有什么规则)，APK解析（为什么可以免安装），进程管理（能确保隐藏起来，不会在process轻易被kill）