

# APK沙箱技术在平台型App中的架构实战

---

黄明登 科大讯飞





关注InfoQ官方信息  
及时获取QCon软件开发者  
大会演讲视频信息

**ArchSummit**  
全球架构师峰会 2016

[北京站] 2016年12月2日-3日  
咨询热线: 010-89880682

**QCon**  
全球软件开发大会

[北京站] 2017年4月16日-18日  
咨询热线: 010-64738142

# 目录CONTENTS

01 业务发展所带来的技术挑战

02 业界已有技术方案的对比分析

03 apk沙箱技术的实现原理

04 讯飞输入法中的实践经验分享

# 讯飞输入法业务发展

- 科大讯飞第一款toC产品，以快捷准确的语音输入为特色，经过6年时间耕耘，输入法市场排名第2，月活用户过亿
- 随着用户量的增大，讯飞输入法开始具备为第三方应用导流的能力，下图为输入法中的导流页面



# 遇到的问题

## 第三方应用接入方式

- apk方式、功能模块方式

## apk方式

- 第三方产品与输入法主产品耦合度低，但是apk需要经过下载、安装的环节，转化率低

## 功能模块方式

- 无需下载安装，转化率高，但与输入法主产品严重耦合，团队之间的沟通协作成本加大，质量控制困难，研发成本增加

# 遇到的问题

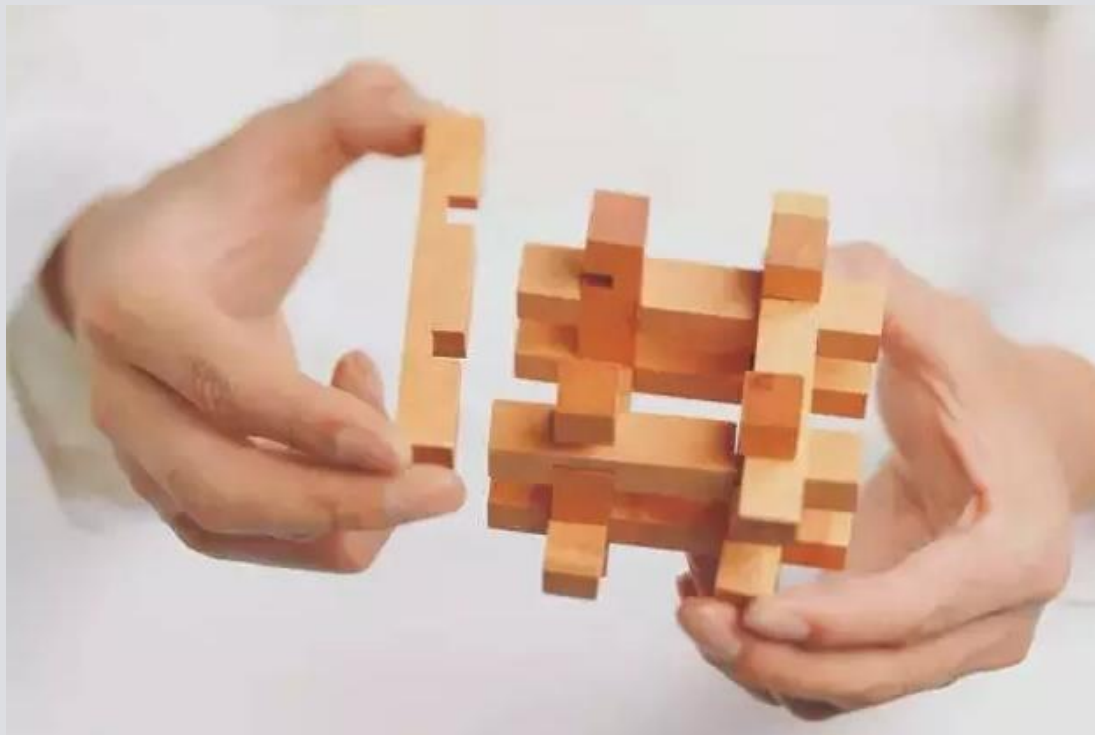
---

- 鱼和熊掌不能兼得？



# 解决思路

apk插件化





# 目录 CONTENTS

讯飞技术沙龙  
爱分享 · 共成长

0 1 业务发展所带来的技术挑战

0 2 业界已有技术方案的对比分析

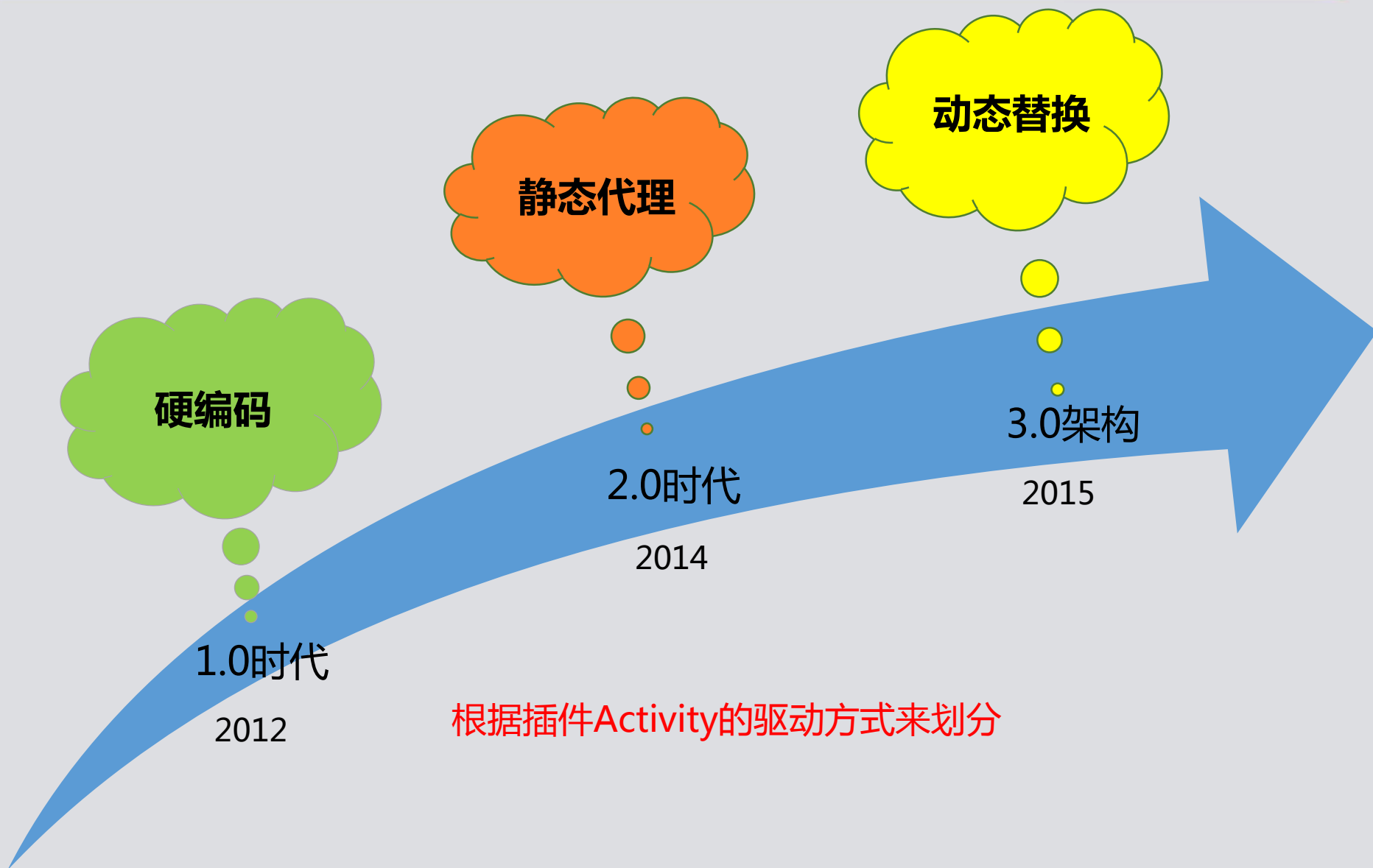
0 3 apk沙箱技术的实现原理

0 4 讯飞输入法中的实践经验分享





# 插件化技术演进路线



# 插件化技术方案对比

分类	硬编码	静态代理	动态替换
技术原理	插件Activity在宿主中 <b>硬编码注册</b> ，插件UI在代码中 <b>硬编码实现</b>	宿主中预注册 <b>ProxyActivity</b> ，在ProxyActivity的各个生命周期里回调插件Activity的对应方法。	宿主中预注册 <b>StubActivity</b> ，当调用插件Activity时框架将intent替换成StubActivity，最后创建Activity时又将intent还原，实际创建插件Activity对象
优点	不用关心系统底层源码，逻辑简单，框架实现成本低	不用关心系统底层源码，框架实现成本较低	对插件约束小，遵循android标准，插件开发维护成本低，可独立运行
缺点	无法动态扩展插件Activity；插件开发成本高，非标准化	插件开发成本高，非标准化	需要熟悉android系统源码，框架开发成本高，技术难度大
代表性开源框架	无	dynamic-load-apk、direct-load-apk	DroidPlugin、VirtualApp、Small、ACDD、DynamicAPK、Android-Plugin-Framework

# 插件类型

## 融合型插件

定义：它与宿主一起形成功能整体，不能脱离宿主而单独存在

拆

插件与宿主关系：分而不离

适用场景：大型app程序拆分，分而治之，便于管理和维护

## 独立型插件

定义：插件自身能够提供完整的功能服务，可以脱离宿主而独立存在

合

插件与宿主关系：合而不并

适用场景：app作为运营平台，为第三方业务提供服务场地（烧烤店里卖水果、飞饼、香烟）

# 沙箱技术

---


apk沙箱技术的核心思想就是在宿主app内部建立起隔离的运行环境，在里面直接运行插件app，插件app与宿主app互不影响，沙箱内插件app的环境感知与标准app没有区别

apk沙箱技术是独立型插件一套完整的解决方案，也正好解决了讯飞输入法当前遇到的问题



# 目录 CONTENTS

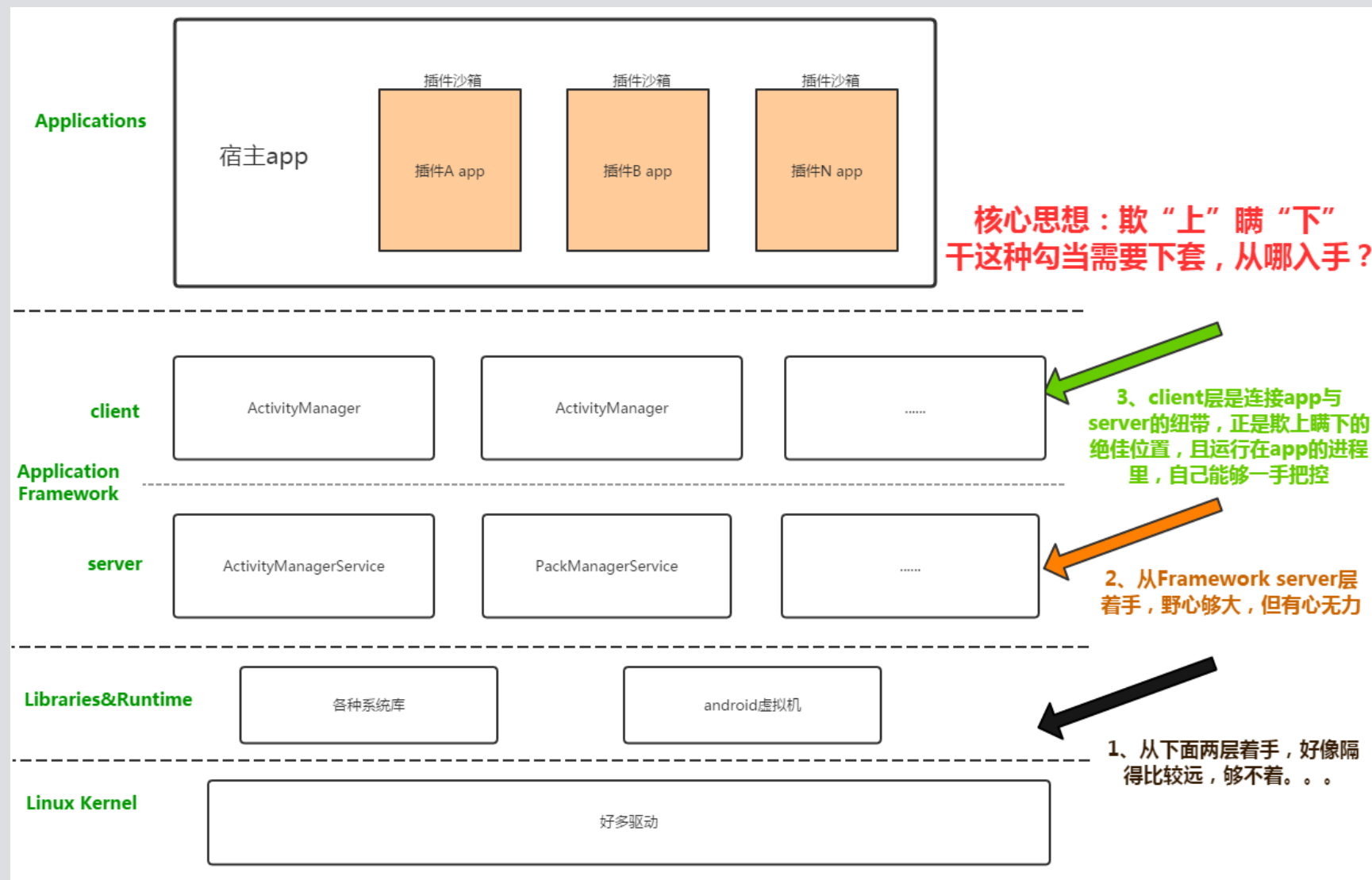
讯飞技术沙龙  
爱分享 · 共成长

- 0 1 业务发展所带来的技术挑战
  - 0 2 业界已有技术方案的对比分析
  - 0 3 apk沙箱技术的实现原理
  - 0 4 讯飞输入法中的实践经验分享
- 

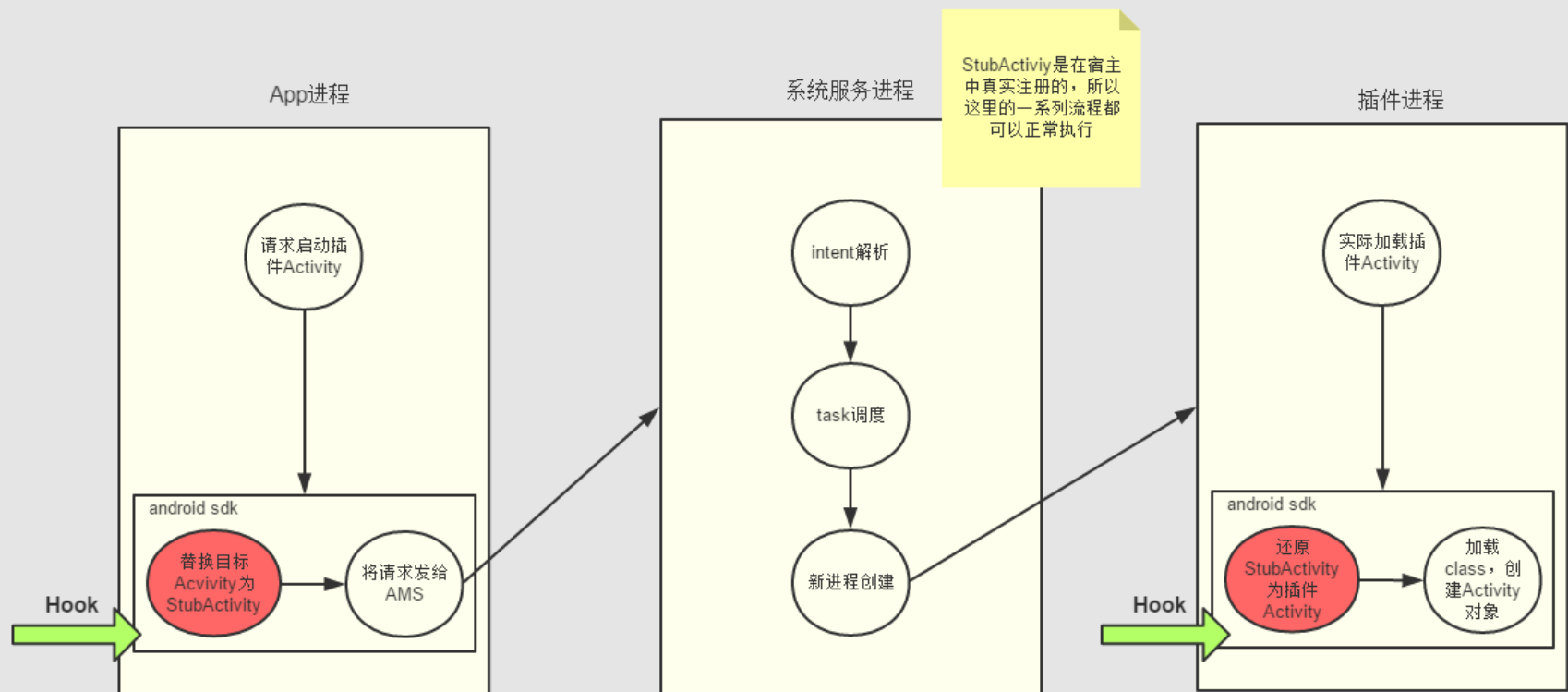
# apk沙箱所面临的技术问题



# 总体思路



# Android组件驱动—Activity





# Android组件驱动—Activity

那么问题来了：

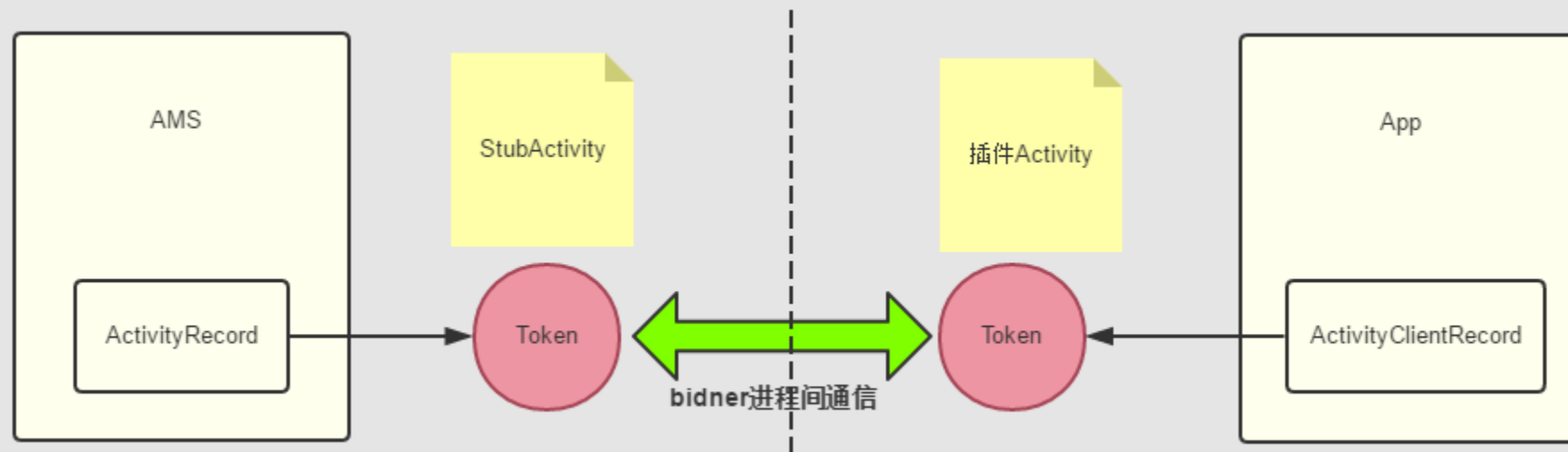
上述过程中，AMS持有的是StubActivity记录，App持有的是插件Activity记录，不是同一个东西，这样整个生命周期流程能够正常地串起来吗？



# Android组件驱动—Activity

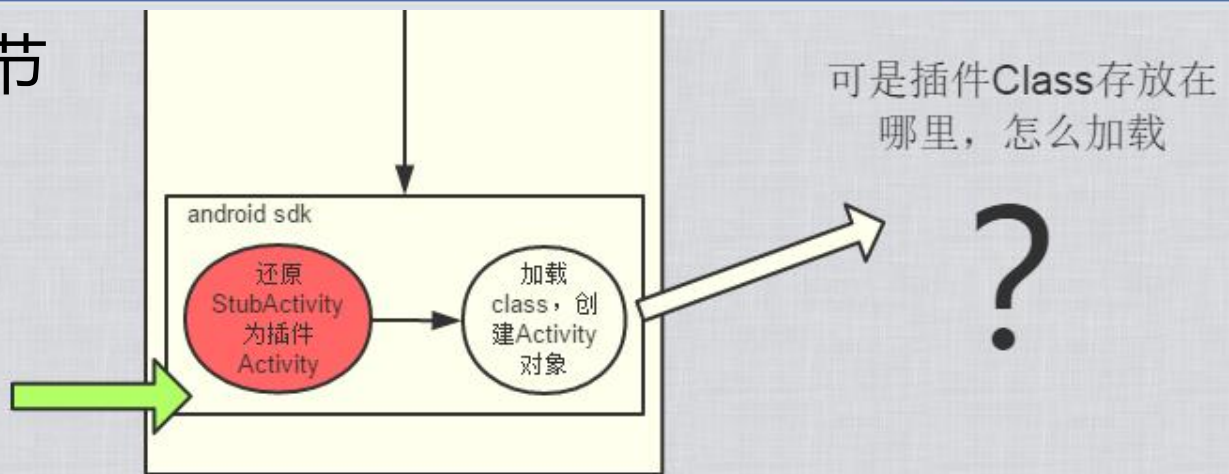
没有问题：

它们之间是通过同一个token作为联系的纽带



# 插件加载

Activity驱动一节  
遗留问题：



分析系统源码 ActivityThread.java：

```
try {  
    java.lang.ClassLoader cl = r.packageInfo.getClassLoader();  
    activity = mInstrumentation.newActivity(  
        cl, component.getClassName(), r.intent);  
    StrictMode.incrementExpectedActivityCount(activity.getClass());  
    r.intent.setExtrasClassLoader(cl);  
    if (r.state != null) {  
        r.state.setClassLoader(cl);  
    }  
}
```

解决思路：构造插件apk的LoadedApk对象，插入当前进程  
ActivityThread对象的mPackage域

# 环境隔离

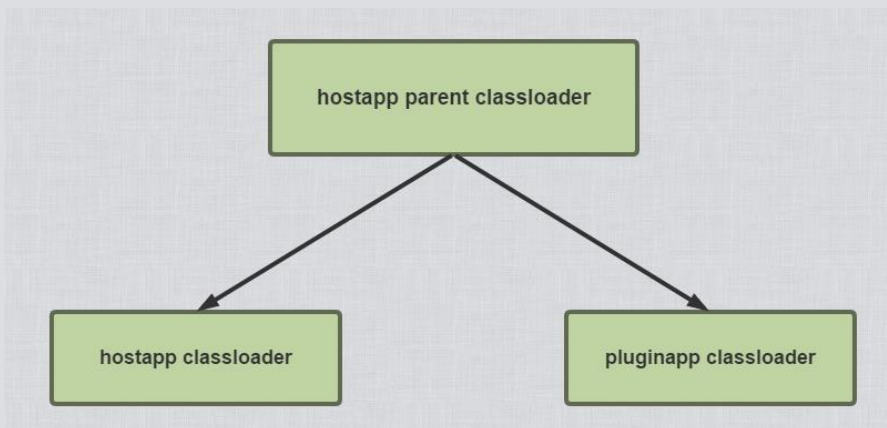
物理隔离：

代码、资源、数据文件在存储空间上的隔离

存储路径：data/data/宿主包名/plugins/插件包名，按照源文件、代码、数据等建立子目录，这些路径要保存到插件LoadedApk对象的成员变量中

运行时隔离：

- 1、进程隔离：为占坑的android组件配置不同的process值
- 2、class隔离：为插件创建独立的classloader，与宿主共享parent classloader，避免代码冲突



## 要解决的问题

- 支持将不同的插件分配到不同的进程
- 支持将同一个插件分配到多个进程
- 插件获取到的进程名称与插件Manifest中定义的一致

## 解决思路

- 宿主中注册多个不同进程的占坑组件
- 维护一张记录表，管理插件进程和占坑进程的绑定关系，根据插件manifest定义的process值来按需分配，使得运行时的进程分布与插件预期的一样
- Hook IActivityManager中的getRunningAppProcesses方法，修改进程名称为插件中定义的值（真实进程名称为占坑组件配置的process值）

# 系统服务调用

## 1、欺骗系统App Ops，以剪贴板服务ClipboardService为例：

```
public void setPrimaryClip(ClipData clip, String callingPackage) {  
    synchronized (this) {  
        if (clip != null && clip.getItemCount() <= 0) {  
            throw new IllegalArgumentException("No items");  
        }  
        if (mAppOps.noteOp(AppOpsManager.OP_WRITE_CLIPBOARD, Binder.getCallingUid(),  
            callingPackage) != AppOpsManager.MODE_ALLOWED) {  
            return;  
        }  
    }  
}
```

解决方案：Hook系统服务的api接口类，将参数中关于packageName的值全部替换成宿主的，用来隐瞒身份

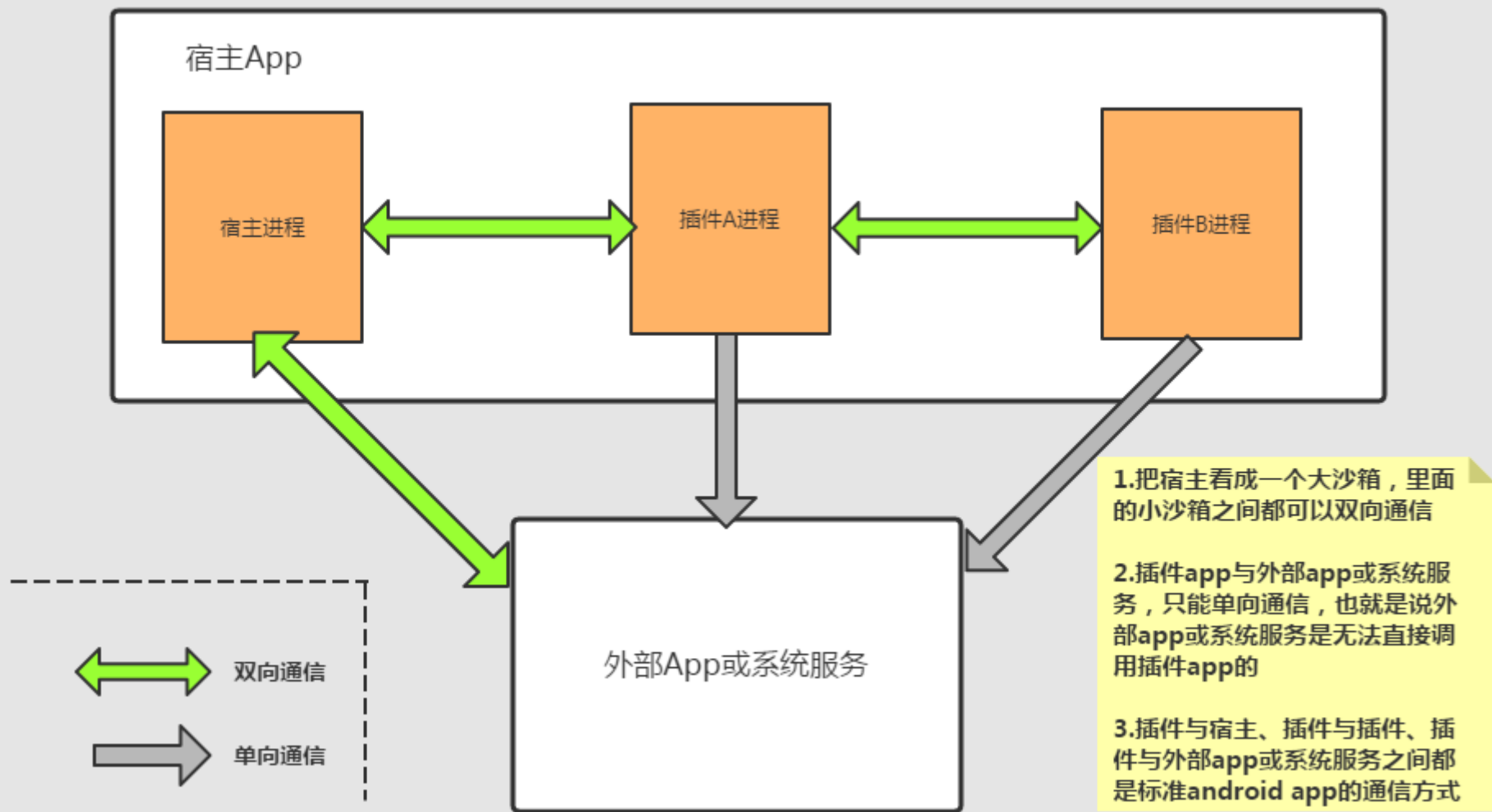
# 系统服务调用

## 2、PendingIntent处理

```
public static PendingIntent getActivity(Context context, int requestCode,
    Intent intent, int flags, Bundle options) {
    String packageName = context.getPackageName();
    String resolvedType = intent != null ? intent.resolveTypeIfNeeded(
        context.getContentResolver()) : null;
    try {
        intent.migrateExtraStreamToClipData();
        intent.prepareToLeaveProcess();
        IIntentSender target =
            ActivityManagerNative.getDefault().getIntentSender(
                ActivityManager.INTENT_SENDER_ACTIVITY, packageName,
                null, null, requestCode, new Intent[] { intent },
                resolvedType != null ? new String[] { resolvedType } : null,
                flags, options, UserHandle.myUserId());
        return target != null ? new PendingIntent(target) : null;
    } catch (RemoteException e) {
        return null;
    }
}
```

解决方案：Hook IActivityManager的getIntentSender方法，将目标intent转为插件框架内部的一个Service中（在宿主中注册），以这个service作为跳板，然后再调用插件的intent

# 沙箱通信





## 安装

- 按照规划的插件物理环境，将插件apk包拷贝/解压到对应的目录即可
- 检查插件权限以及SO库是否与宿主兼容

## 卸载

- 停止正在运行的插件进程
- 删除与插件相关的所有物理文件
- 清除插件运行痕迹（快捷方式、通知栏消息等）

## 升级

- 如果老插件还没有运行，则直接进行升级即可
- 如果老插件已经运行，则先将新插件包保存起来，等插件停止运行后再升级
- 支持强制升级

# Rom适配

---

Rom版本：2.x—6.x

厂商定制Rom、第三方Rom

解决思路：没有捷径，分析源码，主要是看系统api有没有变化

# 目录 CONTENTS

讯飞技术沙龙  
爱分享 · 共成长

- 0 1 业务发展所带来的技术挑战
- 0 2 业界已有技术方案的对比分析
- 0 3 apk沙箱技术的实现原理
- 0 4 讯飞输入法中的实践经验分享

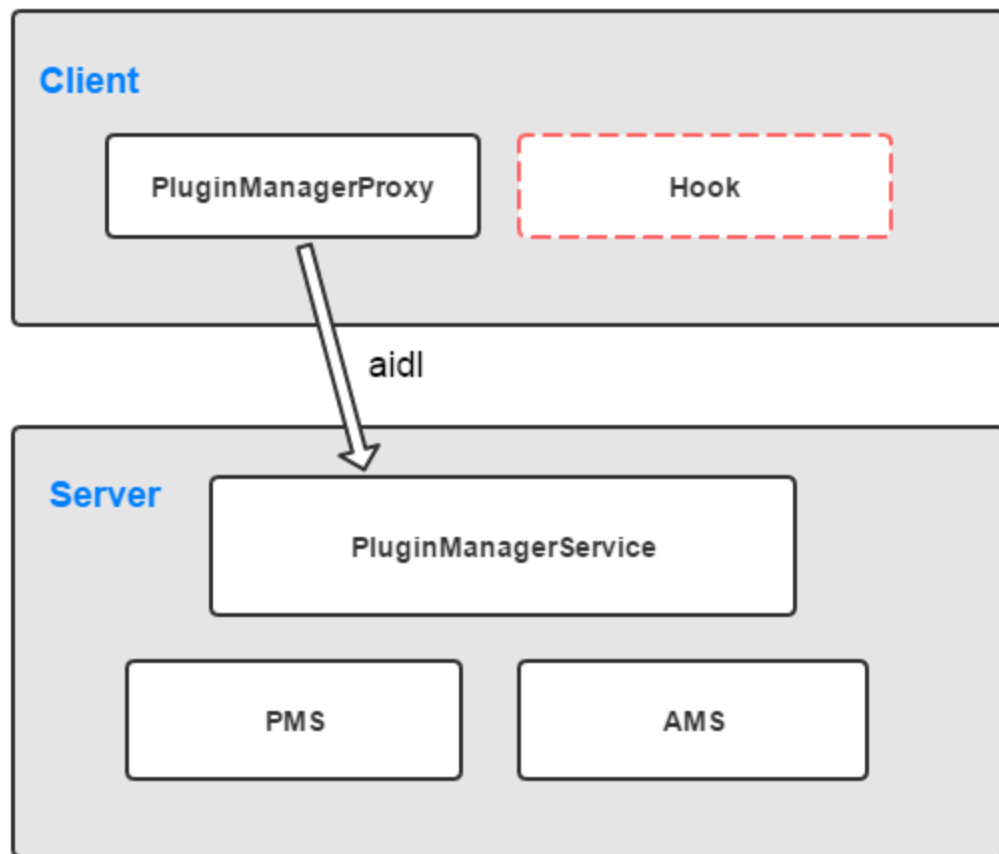
The logo consists of a 3x3 grid of squares in orange, blue, and green, followed by the text "GreenPlug" in a large, black, sans-serif font.

# GreenPlug

---

- 讯飞apk沙箱解决方案：GreenPlug
- 前人种树，后人乘凉，感谢开源框架的贡献：DroidPlug、Android-Plugin-Framework、VirtualApp

# GreenPlug逻辑架构

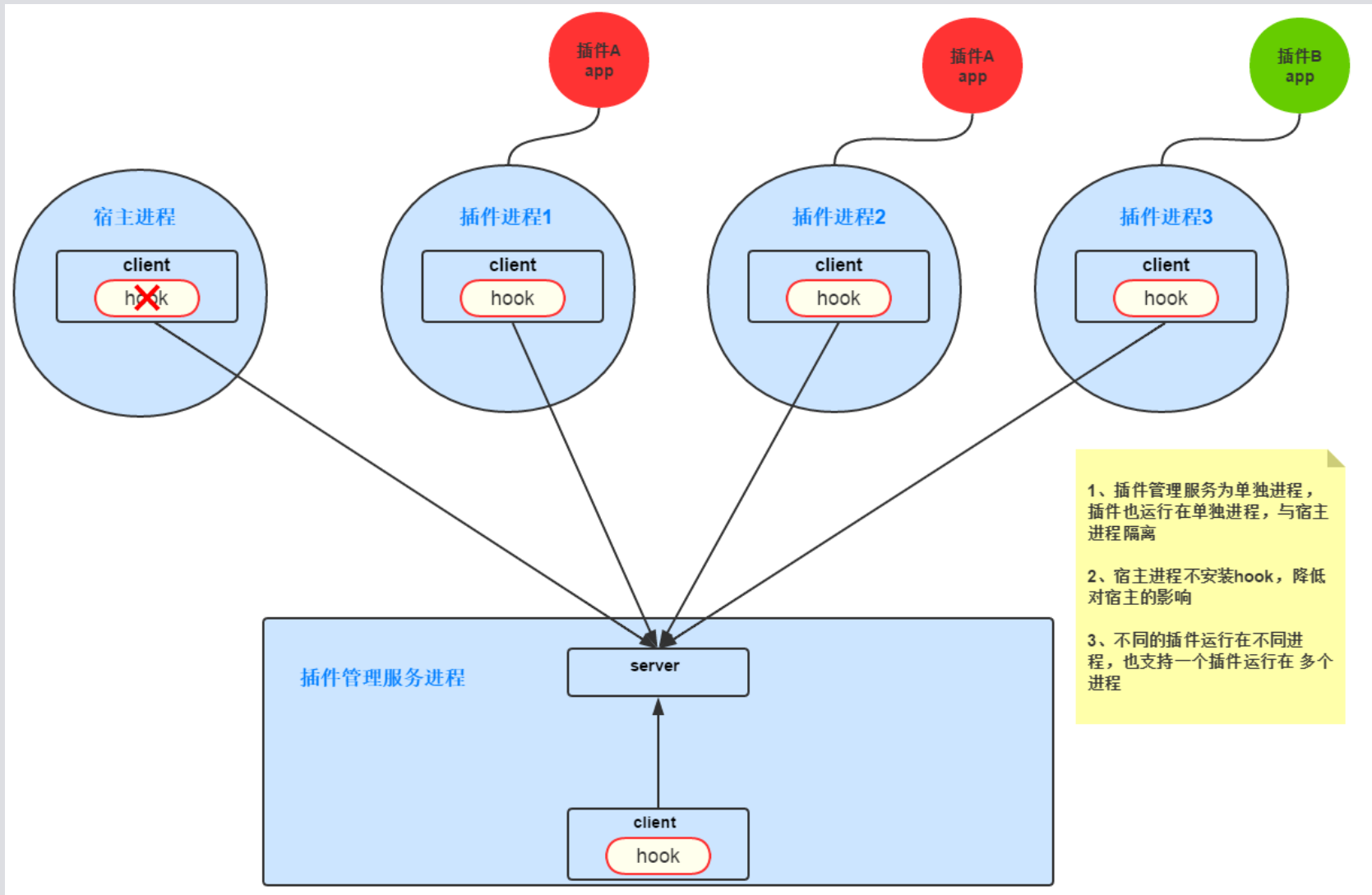


1、整体为c/s架构

2、Server端：提供跨进程的全局统一服务，主要负责插件安装管理（PMS）和插件运行管理（AMS）

3、Client端：提供插件管理服务的客户端接口和hook模块（可选）

# GreenPlug运行时架构



# 已上线插件情况

---

小飞读报app

流量助手app

铃声助手app

架构解耦

各产品团队按照自己的计划研发和发布，互不影响

转化率

插件app下载到激活的转化率从10%提升到90%

# 踩过的坑—Rom适配

---

Rom兼容性问题比预期情况要好，目前遇到过的问题有：

- 小米：显示toast时会判断请求者的进程名称，若进程名称不正确则不显示toast
- 魅族：显示toast的方法为enqueueToastUnrepeated
- 锤子：对webview安装hook会引起崩溃



# 踩过的坑—自定义通知消息

## 问题原因

- Notification消息由系统SystemUI进行绘制和展示，而由于沙箱环境的可见性，系统服务是无法感知插件app的存在，没法获取插件app中的自定义布局文件

## 保守方案

- 将插件的资源文件放到宿主中，并使用public.xml将资源id固化，但此方案会造成插件与宿主的耦合，插件升级更新也不方便

## 激进方案

- 提前绘制通知消息的RemoteView，以Bitmap的保存为ImageView的Src；在ImageView的上方绘制多个透明的TextView，用来接收点击事件并根据点击区域分发给原始处理对象。此方案在手机横竖屏切换时存在问题：感知屏幕翻转，查询通知状态

# 踩过的坑—第三方SDK

---

还是由于沙箱可见性问题，一些涉及与外部app进行通信的SDK存在问题，如：登录、分享、推送等SDK

这些问题没有有效的统一解决方案，只能具体问题具体分析，或者寻找一些替代方案/规避措施

# 踩过的坑—安全软件拦截

---

一些系统级的安全软件也会在app中安装hook，这样会导致与我们自己的hook冲突，最常见的是ActivityThread中的handler callback的hook冲突，这类问题多出现在小米和魅族Rom，导致插件无法正常加载

解决方案：出现这种问题后，实际加载的是用来占位的Activity/Service，而非插件Activity/Service，于是在占位组件的启动逻辑中重新检查和安装hook，然后再次启动插件的组件

# 踩过的坑—初始化失败

---

由于在Manifest中预注册了非常多的占位组件，沙箱框架在初始化时解析这些占位组件时容易出现DeadObjectException，原因是系统PMS服务进行跨进程传输时单次数据量过大

解决方案：分多次或开启多个线程解析Manifest中的占位组件，减少单次传输的数据量

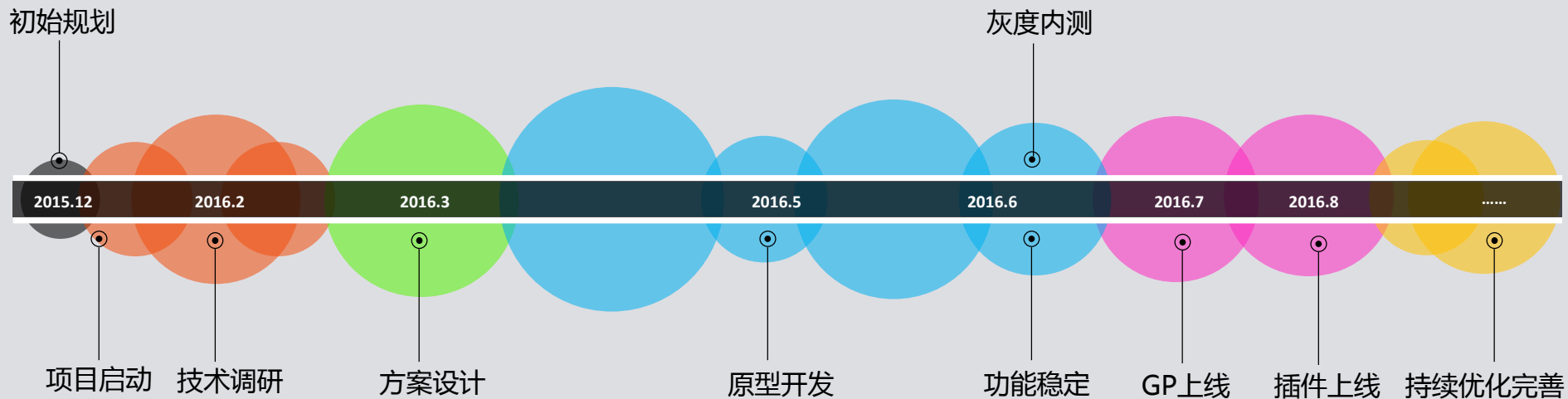
# 踩过的坑—内存回收异常

由于讯飞输入法覆盖人群广，用户基数大，用户中还存在不少的低端机型。而沙箱框架需要开启多个进程，有一定的内存开销，在低端机型中，容易出现内存回收异常，堆栈如下：

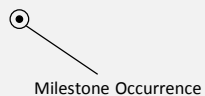
SIGSEGV(SEGV MAPERR)	
#00	pc 00037ae4 /system/lib/libdvm.so [armeabi-v7a]
#01	pc 00029c14 /system/lib/libdvm.so (dvmHeapBitmapScanWalk(HeapBitmap*, void (*)(Object*, void*, void*), void*) +96) [armeabi-v7a]
#02	pc 00037f74 /system/lib/libdvm.so (dvmHeapScanMarkedObjects() +36) [armeabi-v7a]
#03	pc 0002a1d4 /system/lib/libdvm.so (dvmCollectGarbageInternal(GcSpec const*) +824) [armeabi-v7a]
#04	pc 0006e727 /system/lib/libdvm.so [armeabi-v7a]
#05	pc 00053bc7 /system/lib/libdvm.so [armeabi-v7a]
#06	pc 00012e00 /system/lib/libc.so (__thread_entry +48) [armeabi-v7a]
#07	pc 00012558 /system/lib/libc.so (pthread_create +172) [armeabi-v7a]

解决方案：对于Ram小于1G的手机不启用沙箱框架；本地没有安装插件app时也不启用沙箱框架，做到按需开启

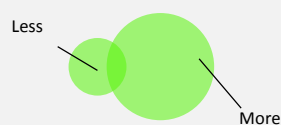
# GreenPlug研发路线



## MILESTONES



## INVOLVEMENT



## PHASE



讯飞移动互联技术团队公众号



THANK

---