1 创建进程过程

2

反射注入

动态代理

依赖注入

基于注解的反射注入技术

Application 启动过程

如何加载class

如何加载apk

DroidPlugin 主要依赖PackageParser 但是这个类的接口不稳定

通过反射调用 PackageParser

VirtualApp

自定义PackageParser 或者叫自实现

Android系统将每个应用当作一个用户

安装App

HomePresenterImpl.java public void addApp(AppInfoLite info)

AppRepository.java public InstallResult addVirtualApp(AppInfoLite info)

VirtualCore.java public InstallResult installPackage(String apkPath, int flags)

VirtualCore.java

IAppManager mService;

启动App

HomeActivity.java private void initLaunchpad()

HomePresenterImpl.java public void launchApp(AppData data)

LoadingActivity.java public static void launch(Context context,

String packageName,

int userId)

{

Intent intent = VirtualCore.get().getLaunchIntent(packageName,userId);

}

virtual/client/ipc/VAactivityManager.java

io.VirtualApp

用户ID

Process.java

int myUid()

UserHandle myUserHandle()

**user id 与 uid**

**结论**

1. user id = uid / 100000
2. 目前 Android 手机上所有 APP 的 user id 都为 0
3. root 权限与 uid 是否为 0 有关，与 user id 无关

启动 Activity 的方法并非只有在 APP 进程里使用 am start 一种，还有通过 adb 命令 adb shell am start 或在 APP 进程里使用 startActivity 等，它们为什么没有抛出此异常呢？

### 为何 adb shell am start 不抛此异常

**原因：** shell 是拥有 INTERACT\_ACROSS\_USERS\_FULL 权限的，所以 am start 作为其子进程继承了 shell 的 uid 和对应权限

## Activity

启动startActivity

ActivityStackSupervisor.java

void startSpecificAcitivityLocked()

AMS ActivityManagerService

startProcessLocked()

ActivityThread.java

## Service

startService

ActivityServices.java

private final String bringUpServiceLocked()

{

startProcessLocked(…)

}

## ContentProvider

ContentResolver.query()

AMS.java

private final ContentProviderImpl(…)

## Broadcast

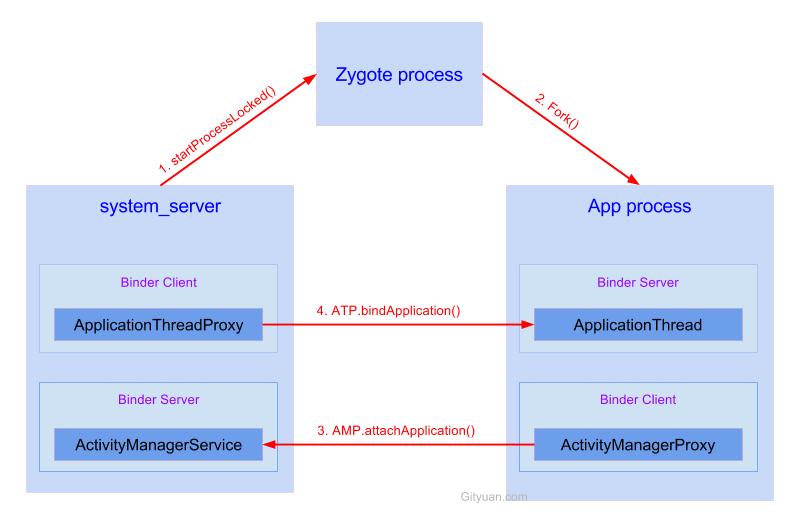
BroadcastQueue.java

final void processNextBroadcast(bool fromMessage)

{

startProcessLocked

}



* ATP: ApplicationThreadProxy
* AT: ApplicationThread (继承于ApplicationThreadNative)
* AMP: ActivityManagerProxy
* AMS: ActivityManagerService (继承于ActivityManagerNative)

1. system\_server进程中调用startProcessLocked方法,该方法最终通过socket方式,将需要创建新进程的消息告知Zygote进程,并阻塞等待Socket返回新创建进程的pid;
2. Zygote进程接收到system\_server发送过来的消息, 则通过fork的方法，将zygote自身进程复制生成新的进程，并将ActivityThread相关的资源加载到新进程app process,这个进程可能是用于承载activity等组件;
3. 创建完新进程后fork返回两次, 在新进程app process向servicemanager查询system\_server进程中binder服务端AMS,获取相对应的Client端,也就是AMP. 有了这一对binder c/s对, 那么app process便可以通过binder向跨进程system\_server发送请求,即attachApplication()
4. system\_server进程接收到相应binder操作后,经过多次调用,利用ATP向app process发送binder请求, 即bindApplication.

system\_server拥有ATP/AMS, 每一个新创建的进程都会有一个相应的AT/AMS,从而可以跨进程 进行相互通信. 这便是进程创建过程的完整生态链.

## Binder IPC 是什么

/dev/binder 虚拟设备 内核

BpServiceManager

Binder Proxy

BnMediaPlayerService

Binder native

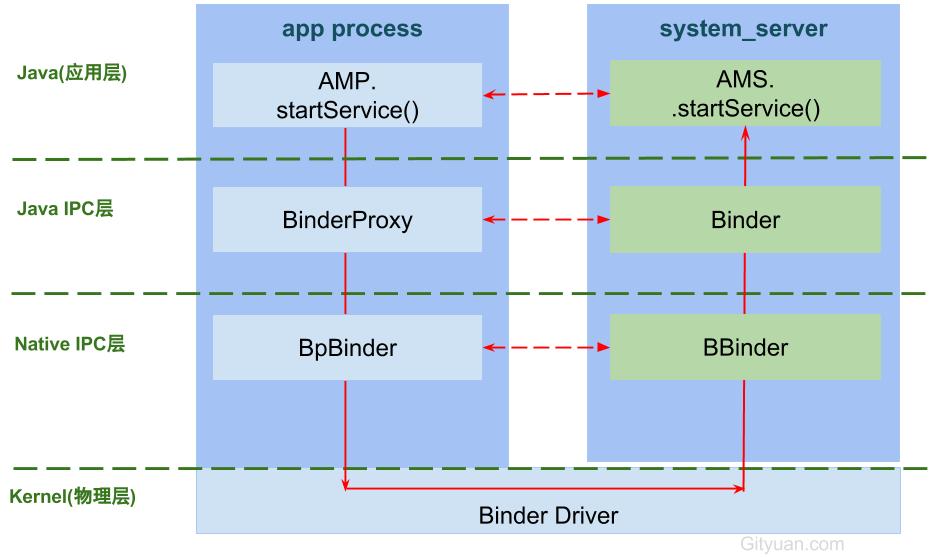
BpServiceManager

BnServiceManager - system\_serve

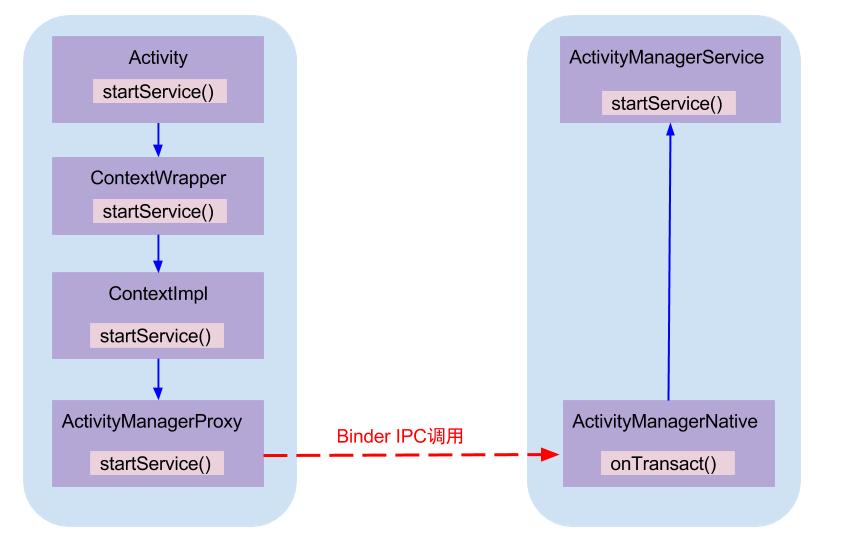
<http://gityuan.com/2016/09/04/binder-start-service/>

<http://gityuan.com/2015/10/31/binder-prepare/>

Binder分层架构



startService原理



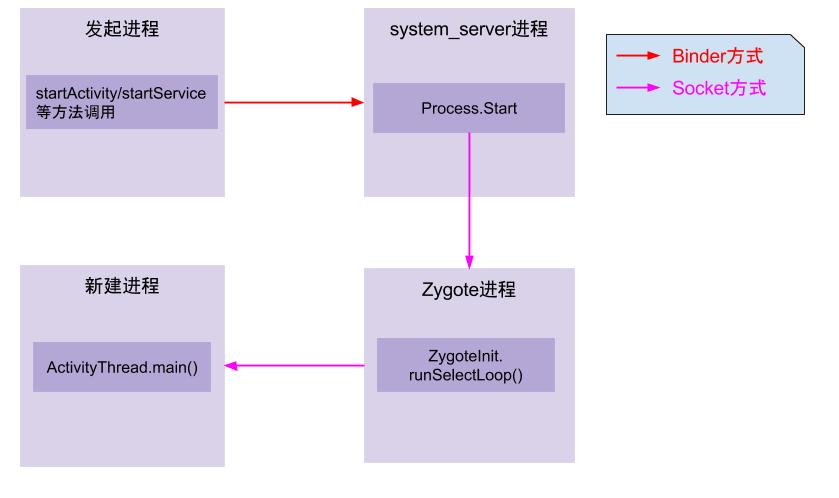
Android进程创建流程

<http://gityuan.com/2016/03/26/app-process-create/>

进程与线程

从Linux角度来说进程与线程都是一个task\_struct结构体，除了是否共享资源外，并没有其他本质的区别

进程创建图



线程与进程的最为本质的区别便是是否共享内存空间，图中VSIZE和Zygote进程相同的才是Zygote的子线程，否则就是Zygote的子进程。

## VirtualApp 启动App过程

Android应用隔离是基于Linux系统的多用户机制实现的，即每个应用在安装时被分配了不同的Linux用户uid/gid。而在VirtualApp中，client应用（通过VirtualApp安装的应用）与host应用（即VirtualApp本身）是具有相同用户uid的。

uid是在安装过程中分配的，**Android安装应用过程分析？**

**MainProcess**

**与一个正常的APP没有任何区别，没有做特殊的事情**

**Server Process:这个进程如何启动的？**

io.virtualapp:x 相当于系统的system\_server进程

waitEngine()

ProviderCall

{

context = 主进程的Context

auth = “virtual.service.BinderProvider”

method = “ensure\_created”

如何启动ContentProvider

context.getContentResolver().call(uri,method,arg,extras);

}

<http://www.jianshu.com/p/1bc01ec36e95>

**VApp Process：如何启动的**

[**http://blog.csdn.net/leif\_/article/details/73223851**](http://blog.csdn.net/leif_/article/details/73223851)

**系统服务的代理和替换**

**应用所持有的是系统服务的BinderProxy,对这些构造代理并替换**

**基础：HOOK框架分析**

**安装流程，四大组件的启动流程，反射，代理**

**StubActivity**

**StubDialog**

**StubContentProvider**

## 技术准备

彻底理解Java动态代理

<http://www.cnblogs.com/flyoung2008/archive/2013/08/11/3251148.html>

Java泛型

<http://baike.baidu.com/item/java%E6%B3%9B%E5%9E%8B?fr=aladdin>

Java反射机制

<http://baike.baidu.com/item/JAVA%E5%8F%8D%E5%B0%84%E6%9C%BA%E5%88%B6?fr=aladdin>

Java注解

<http://baike.baidu.com/item/Java%20%E6%B3%A8%E8%A7%A3?fr=aladdin>

动态加载

<https://github.com/singwhatiwanna/dynamic-load-apk>

ClassLoader

双亲委托机制

**某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，依次递归，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。**

public class PathClassLoader extends BaseDexClassLoader

public class DexClassLoader extends BaseDexClassLoader

class QrTask

{

public static String scanQrImage(){}

}

public void callInPlugin()

{

classLoader = new DexClassLoader(apkPath, dexFile.getAbsolutePath(),

null, getClassLoader());

try

{

Class<?> pluginClass = classLoader.loadClass(“classname”);

Object obj = pluginClass.newInstance();

Method m = pluginClass.getDeclaredMethod(“scanQrImage”, null);

method.invoke(obj,null)

}

}

## 参考资料

<http://bbs.pediy.com/thread-216786.htm>

### VirtualHook: 基于VirtualApp的Java代码hook工具

**梆梆加固后的程序无法用VirtualApp启动**

<https://github.com/rk700/VirtualHook>

<http://rk700.github.io/2017/03/30/YAHFA-introduction/>

ART HOOK框架

时序图 类图 流程图 软件