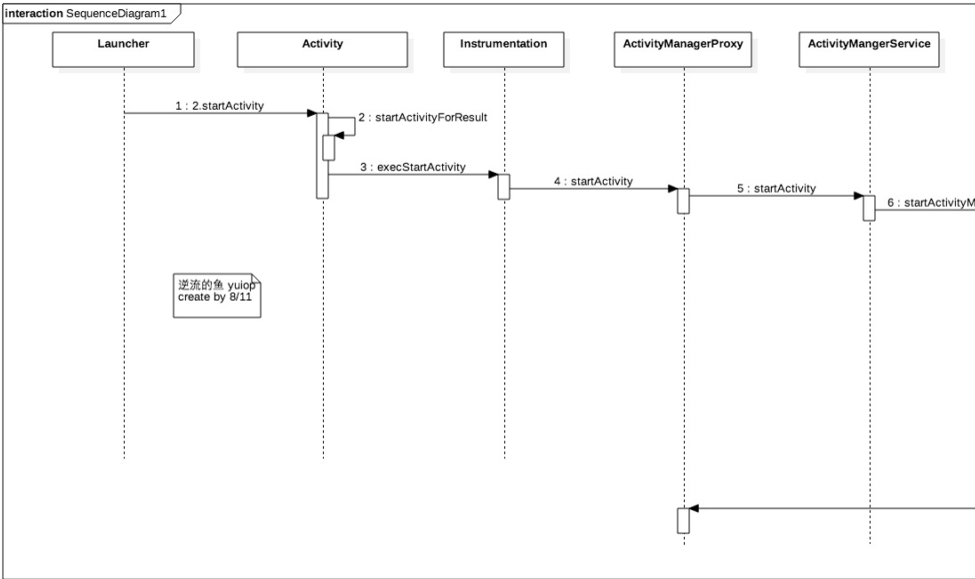
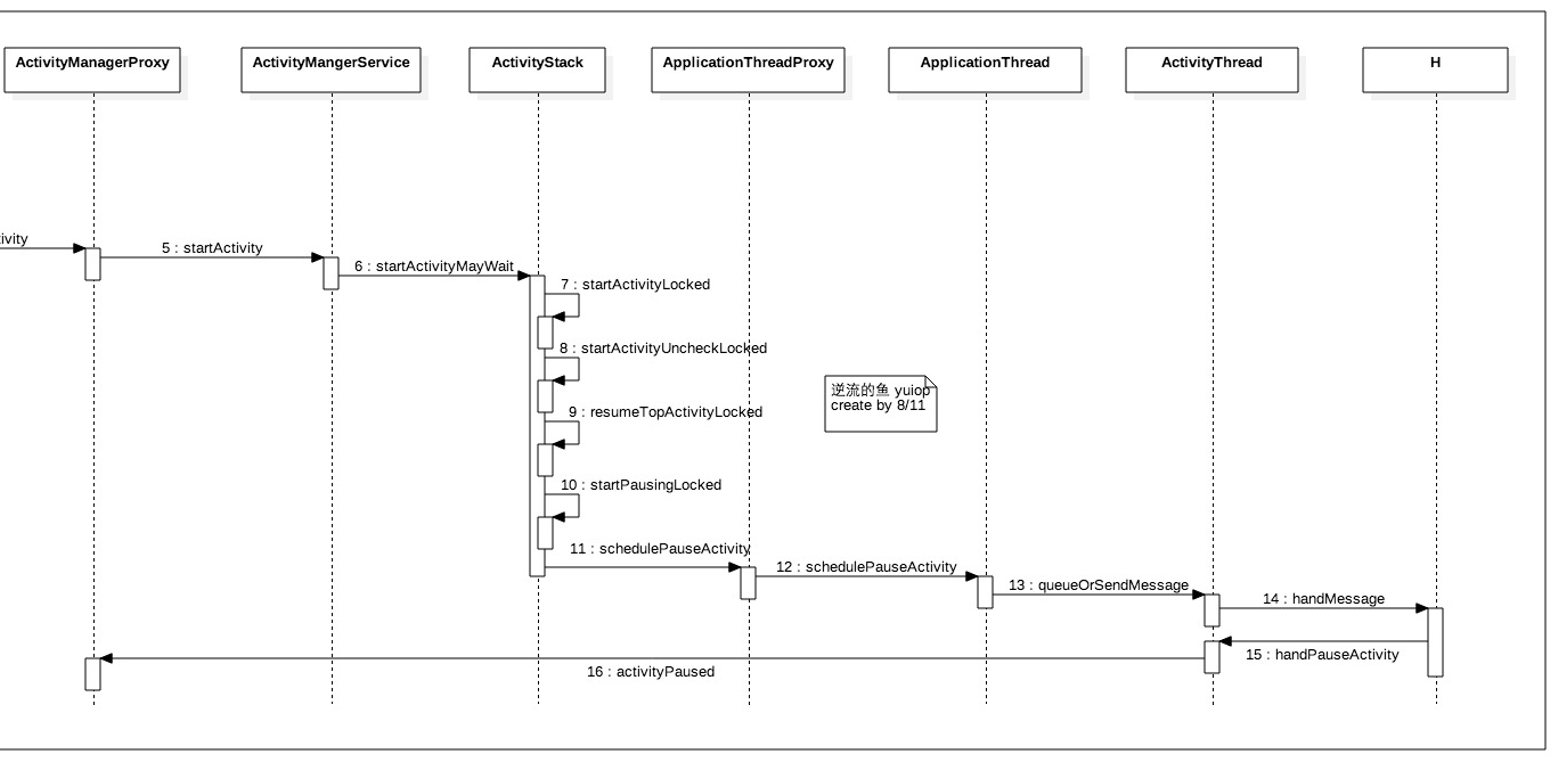
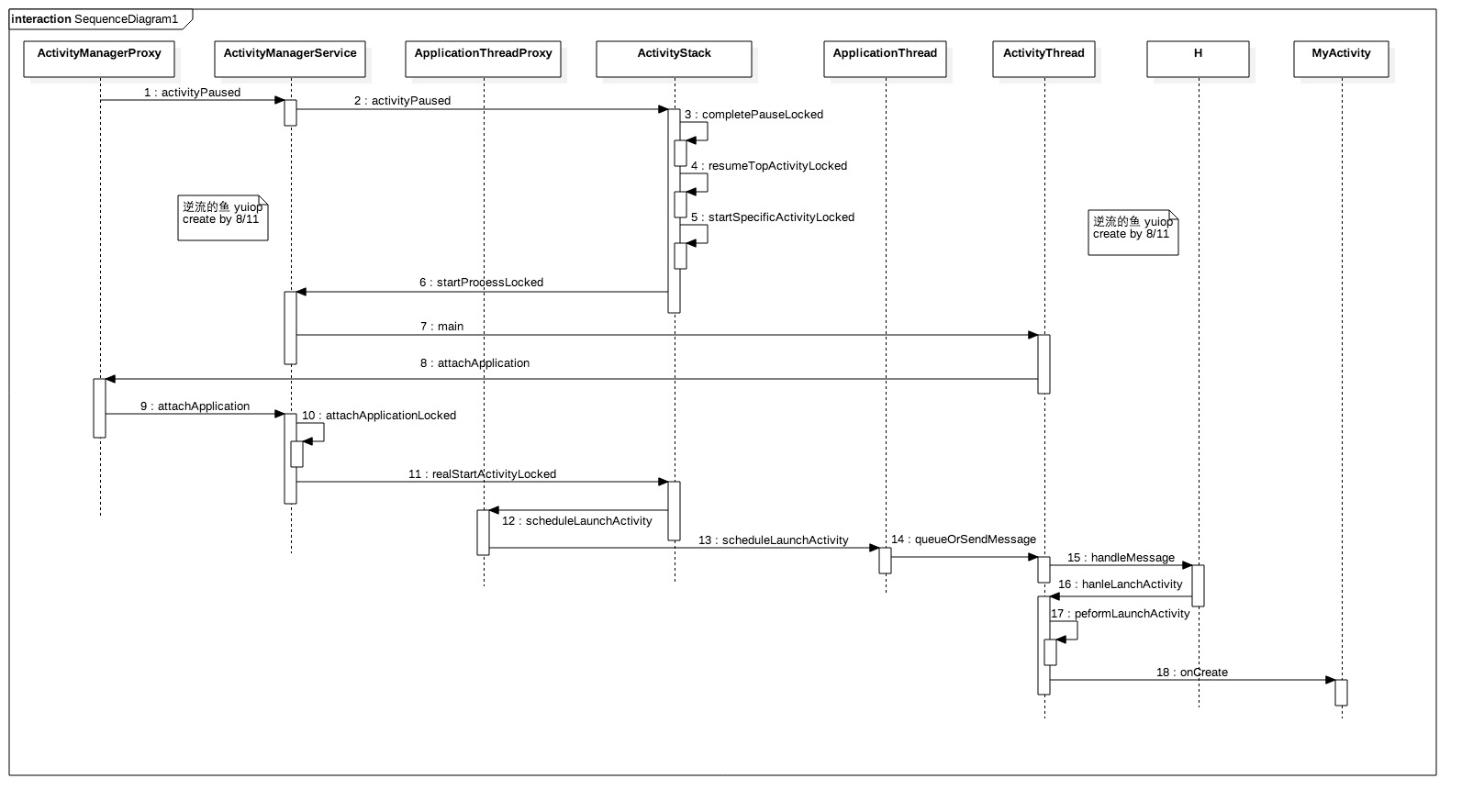
[插件占坑，四大组件动态注册前奏（一） 系统Activity的启动流程](http://blog.csdn.net/hejjunlin/article/details/52190050)

前言：为什么要了解系统Activity，Service,BroadCastReceiver,ContentProvider的启动流程，这是一个对于即将理解插件中的四大组件动态注册，占坑的前提，如果不了解的话，那么很难了解插件hook哪些东西，又是如何骗过AMS来启动Activity,Service,BroadCastReceiver，ContentProvider？

本节主要记录系统Activity的启动流程：   
先看下时序图：







和插件相关主要在第三张图，   
先看H类：这是ActivityThread中的内部类，此类继承自Handler，重写handleMeaage方法, 将msg转换成一个ActivityClientRecord对象，调用ActivityThread类成员函数getPackageInfoNoCheck来获得一个LoadApk对象，并且保存在ActivityClientRecord对象r的成员变量packageInfo中。   
ActivityThread有一个该类型的成员变量mH。scheduleLaunchActivity()里会发送一个LAUNCH\_ACTIVITY类型的消息，该消息被mH处理并调用handleLaunchActivity()。

handleLaunchActivity(r,main)来启动ActivityClientRecord对象t所描述的一个Activity组件。即MyActivity

private class H extends Handler {

public static final int LAUNCH\_ACTIVITY = 100;

public static final int PAUSE\_ACTIVITY = 101;

public static final int PAUSE\_ACTIVITY\_FINISHING= 102;

public static final int STOP\_ACTIVITY\_SHOW = 103;

public static final int STOP\_ACTIVITY\_HIDE = 104;

public static final int SHOW\_WINDOW = 105;

public static final int HIDE\_WINDOW = 106;

public static final int RESUME\_ACTIVITY = 107;

public static final int SEND\_RESULT = 108;

public static final int DESTROY\_ACTIVITY = 109;

public static final int BIND\_APPLICATION = 110;

public static final int EXIT\_APPLICATION = 111;

public static final int NEW\_INTENT = 112;

public static final int RECEIVER = 113;

public static final int CREATE\_SERVICE = 114;

public static final int SERVICE\_ARGS = 115;

public static final int STOP\_SERVICE = 116;

public static final int REQUEST\_THUMBNAIL = 117;

public static final int CONFIGURATION\_CHANGED = 118;

public static final int CLEAN\_UP\_CONTEXT = 119;

public static final int GC\_WHEN\_IDLE = 120;

public static final int BIND\_SERVICE = 121;

public static final int UNBIND\_SERVICE = 122;

public static final int DUMP\_SERVICE = 123;

public static final int LOW\_MEMORY = 124;

public static final int ACTIVITY\_CONFIGURATION\_CHANGED = 125;

public static final int RELAUNCH\_ACTIVITY = 126;

public static final int PROFILER\_CONTROL = 127;

public static final int CREATE\_BACKUP\_AGENT = 128;

public static final int DESTROY\_BACKUP\_AGENT = 129;

public static final int SUICIDE = 130;

public static final int REMOVE\_PROVIDER = 131;

public static final int ENABLE\_JIT = 132;

public static final int DISPATCH\_PACKAGE\_BROADCAST = 133;

public static final int SCHEDULE\_CRASH = 134;

public static final int DUMP\_HEAP = 135;

public static final int DUMP\_ACTIVITY = 136;

public static final int SLEEPING = 137;

public static final int SET\_CORE\_SETTINGS = 138;

public static final int UPDATE\_PACKAGE\_COMPATIBILITY\_INFO = 139;

public static final int TRIM\_MEMORY = 140;

public static final int DUMP\_PROVIDER = 141;

public static final int UNSTABLE\_PROVIDER\_DIED = 142;

public static final int REQUEST\_ASSIST\_CONTEXT\_EXTRAS = 143;

public static final int TRANSLUCENT\_CONVERSION\_COMPLETE = 144;

public static final int INSTALL\_PROVIDER = 145;

String codeToString(int code) {

if (DEBUG\_MESSAGES) {

switch (code) {

case LAUNCH\_ACTIVITY: return "LAUNCH\_ACTIVITY";

case PAUSE\_ACTIVITY: return "PAUSE\_ACTIVITY";

case PAUSE\_ACTIVITY\_FINISHING: return "PAUSE\_ACTIVITY\_FINISHING";

case STOP\_ACTIVITY\_SHOW: return "STOP\_ACTIVITY\_SHOW";

case STOP\_ACTIVITY\_HIDE: return "STOP\_ACTIVITY\_HIDE";

case SHOW\_WINDOW: return "SHOW\_WINDOW";

case HIDE\_WINDOW: return "HIDE\_WINDOW";

case RESUME\_ACTIVITY: return "RESUME\_ACTIVITY";

case SEND\_RESULT: return "SEND\_RESULT";

case DESTROY\_ACTIVITY: return "DESTROY\_ACTIVITY";

case BIND\_APPLICATION: return "BIND\_APPLICATION";

case EXIT\_APPLICATION: return "EXIT\_APPLICATION";

case NEW\_INTENT: return "NEW\_INTENT";

case RECEIVER: return "RECEIVER";

case CREATE\_SERVICE: return "CREATE\_SERVICE";

case SERVICE\_ARGS: return "SERVICE\_ARGS";

case STOP\_SERVICE: return "STOP\_SERVICE";

case REQUEST\_THUMBNAIL: return "REQUEST\_THUMBNAIL";

case CONFIGURATION\_CHANGED: return "CONFIGURATION\_CHANGED";

case CLEAN\_UP\_CONTEXT: return "CLEAN\_UP\_CONTEXT";

case GC\_WHEN\_IDLE: return "GC\_WHEN\_IDLE";

case BIND\_SERVICE: return "BIND\_SERVICE";

case UNBIND\_SERVICE: return "UNBIND\_SERVICE";

case DUMP\_SERVICE: return "DUMP\_SERVICE";

case LOW\_MEMORY: return "LOW\_MEMORY";

case ACTIVITY\_CONFIGURATION\_CHANGED: return "ACTIVITY\_CONFIGURATION\_CHANGED";

case RELAUNCH\_ACTIVITY: return "RELAUNCH\_ACTIVITY";

case PROFILER\_CONTROL: return "PROFILER\_CONTROL";

case CREATE\_BACKUP\_AGENT: return "CREATE\_BACKUP\_AGENT";

case DESTROY\_BACKUP\_AGENT: return "DESTROY\_BACKUP\_AGENT";

case SUICIDE: return "SUICIDE";

case REMOVE\_PROVIDER: return "REMOVE\_PROVIDER";

case ENABLE\_JIT: return "ENABLE\_JIT";

case DISPATCH\_PACKAGE\_BROADCAST: return "DISPATCH\_PACKAGE\_BROADCAST";

case SCHEDULE\_CRASH: return "SCHEDULE\_CRASH";

case DUMP\_HEAP: return "DUMP\_HEAP";

case DUMP\_ACTIVITY: return "DUMP\_ACTIVITY";

case SLEEPING: return "SLEEPING";

case SET\_CORE\_SETTINGS: return "SET\_CORE\_SETTINGS";

case UPDATE\_PACKAGE\_COMPATIBILITY\_INFO: return "UPDATE\_PACKAGE\_COMPATIBILITY\_INFO";

case TRIM\_MEMORY: return "TRIM\_MEMORY";

case DUMP\_PROVIDER: return "DUMP\_PROVIDER";

case UNSTABLE\_PROVIDER\_DIED: return "UNSTABLE\_PROVIDER\_DIED";

case REQUEST\_ASSIST\_CONTEXT\_EXTRAS: return "REQUEST\_ASSIST\_CONTEXT\_EXTRAS";

case TRANSLUCENT\_CONVERSION\_COMPLETE: return "TRANSLUCENT\_CONVERSION\_COMPLETE";

case INSTALL\_PROVIDER: return "INSTALL\_PROVIDER";

}

}

return Integer.toString(code);

}

public void handleMessage(Message msg) {

if (DEBUG\_MESSAGES) Slog.v(TAG, ">>> handling: " + codeToString(msg.what));

switch (msg.what) {

case LAUNCH\_ACTIVITY: {

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityStart");

ActivityClientRecord r = (ActivityClientRecord)msg.obj;

r.packageInfo = getPackageInfoNoCheck(

r.activityInfo.applicationInfo, r.compatInfo);

handleLaunchActivity(r, null);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

} break;

case RELAUNCH\_ACTIVITY: {

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityRestart");

ActivityClientRecord r = (ActivityClientRecord)msg.obj;

handleRelaunchActivity(r);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

} break;

case PAUSE\_ACTIVITY:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityPause");

handlePauseActivity((IBinder)msg.obj, false, msg.arg1 != 0, msg.arg2);

maybeSnapshot();

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case PAUSE\_ACTIVITY\_FINISHING:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityPause");

handlePauseActivity((IBinder)msg.obj, true, msg.arg1 != 0, msg.arg2);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case STOP\_ACTIVITY\_SHOW:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityStop");

handleStopActivity((IBinder)msg.obj, true, msg.arg2);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case STOP\_ACTIVITY\_HIDE:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityStop");

handleStopActivity((IBinder)msg.obj, false, msg.arg2);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case SHOW\_WINDOW:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityShowWindow");

handleWindowVisibility((IBinder)msg.obj, true);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case HIDE\_WINDOW:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityHideWindow");

handleWindowVisibility((IBinder)msg.obj, false);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case RESUME\_ACTIVITY:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityResume");

handleResumeActivity((IBinder)msg.obj, true,

msg.arg1 != 0, true);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case SEND\_RESULT:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityDeliverResult");

handleSendResult((ResultData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case DESTROY\_ACTIVITY:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityDestroy");

handleDestroyActivity((IBinder)msg.obj, msg.arg1 != 0,

msg.arg2, false);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case BIND\_APPLICATION:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "bindApplication");

AppBindData data = (AppBindData)msg.obj;

handleBindApplication(data);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case EXIT\_APPLICATION:

if (mInitialApplication != null) {

mInitialApplication.onTerminate();

}

Looper.myLooper().quit();

break;

case NEW\_INTENT:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityNewIntent");

handleNewIntent((NewIntentData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case RECEIVER:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "broadcastReceiveComp");

handleReceiver((ReceiverData)msg.obj);

maybeSnapshot();

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case CREATE\_SERVICE:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "serviceCreate");

handleCreateService((CreateServiceData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case BIND\_SERVICE:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "serviceBind");

handleBindService((BindServiceData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case UNBIND\_SERVICE:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "serviceUnbind");

handleUnbindService((BindServiceData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case SERVICE\_ARGS:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "serviceStart");

handleServiceArgs((ServiceArgsData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case STOP\_SERVICE:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "serviceStop");

handleStopService((IBinder)msg.obj);

maybeSnapshot();

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case REQUEST\_THUMBNAIL:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "requestThumbnail");

handleRequestThumbnail((IBinder)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case CONFIGURATION\_CHANGED:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "configChanged");

mCurDefaultDisplayDpi = ((Configuration)msg.obj).densityDpi;

handleConfigurationChanged((Configuration)msg.obj, null);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case CLEAN\_UP\_CONTEXT:

ContextCleanupInfo cci = (ContextCleanupInfo)msg.obj;

cci.context.performFinalCleanup(cci.who, cci.what);

break;

case GC\_WHEN\_IDLE:

scheduleGcIdler();

break;

case DUMP\_SERVICE:

handleDumpService((DumpComponentInfo)msg.obj);

break;

case LOW\_MEMORY:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "lowMemory");

handleLowMemory();

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case ACTIVITY\_CONFIGURATION\_CHANGED:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "activityConfigChanged");

handleActivityConfigurationChanged((IBinder)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case PROFILER\_CONTROL:

handleProfilerControl(msg.arg1 != 0, (ProfilerControlData)msg.obj, msg.arg2);

break;

case CREATE\_BACKUP\_AGENT:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "backupCreateAgent");

handleCreateBackupAgent((CreateBackupAgentData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case DESTROY\_BACKUP\_AGENT:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "backupDestroyAgent");

handleDestroyBackupAgent((CreateBackupAgentData)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case SUICIDE:

Process.killProcess(Process.myPid());

break;

case REMOVE\_PROVIDER:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "providerRemove");

completeRemoveProvider((ProviderRefCount)msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case ENABLE\_JIT:

ensureJitEnabled();

break;

case DISPATCH\_PACKAGE\_BROADCAST:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "broadcastPackage");

handleDispatchPackageBroadcast(msg.arg1, (String[])msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case SCHEDULE\_CRASH:

throw new RemoteServiceException((String)msg.obj);

case DUMP\_HEAP:

handleDumpHeap(msg.arg1 != 0, (DumpHeapData)msg.obj);

break;

case DUMP\_ACTIVITY:

handleDumpActivity((DumpComponentInfo)msg.obj);

break;

case DUMP\_PROVIDER:

handleDumpProvider((DumpComponentInfo)msg.obj);

break;

case SLEEPING:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "sleeping");

handleSleeping((IBinder)msg.obj, msg.arg1 != 0);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case SET\_CORE\_SETTINGS:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "setCoreSettings");

handleSetCoreSettings((Bundle) msg.obj);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case UPDATE\_PACKAGE\_COMPATIBILITY\_INFO:

handleUpdatePackageCompatibilityInfo((UpdateCompatibilityData)msg.obj);

break;

case TRIM\_MEMORY:

Trace.traceBegin(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER, "trimMemory");

handleTrimMemory(msg.arg1);

Trace.traceEnd(Trace.TRACE\_TAG\_ACTIVITY\_MANAGER);

break;

case UNSTABLE\_PROVIDER\_DIED:

handleUnstableProviderDied((IBinder)msg.obj, false);

break;

case REQUEST\_ASSIST\_CONTEXT\_EXTRAS:

handleRequestAssistContextExtras((RequestAssistContextExtras)msg.obj);

break;

case TRANSLUCENT\_CONVERSION\_COMPLETE:

handleTranslucentConversionComplete((IBinder)msg.obj, msg.arg1 == 1);

break;

case INSTALL\_PROVIDER:

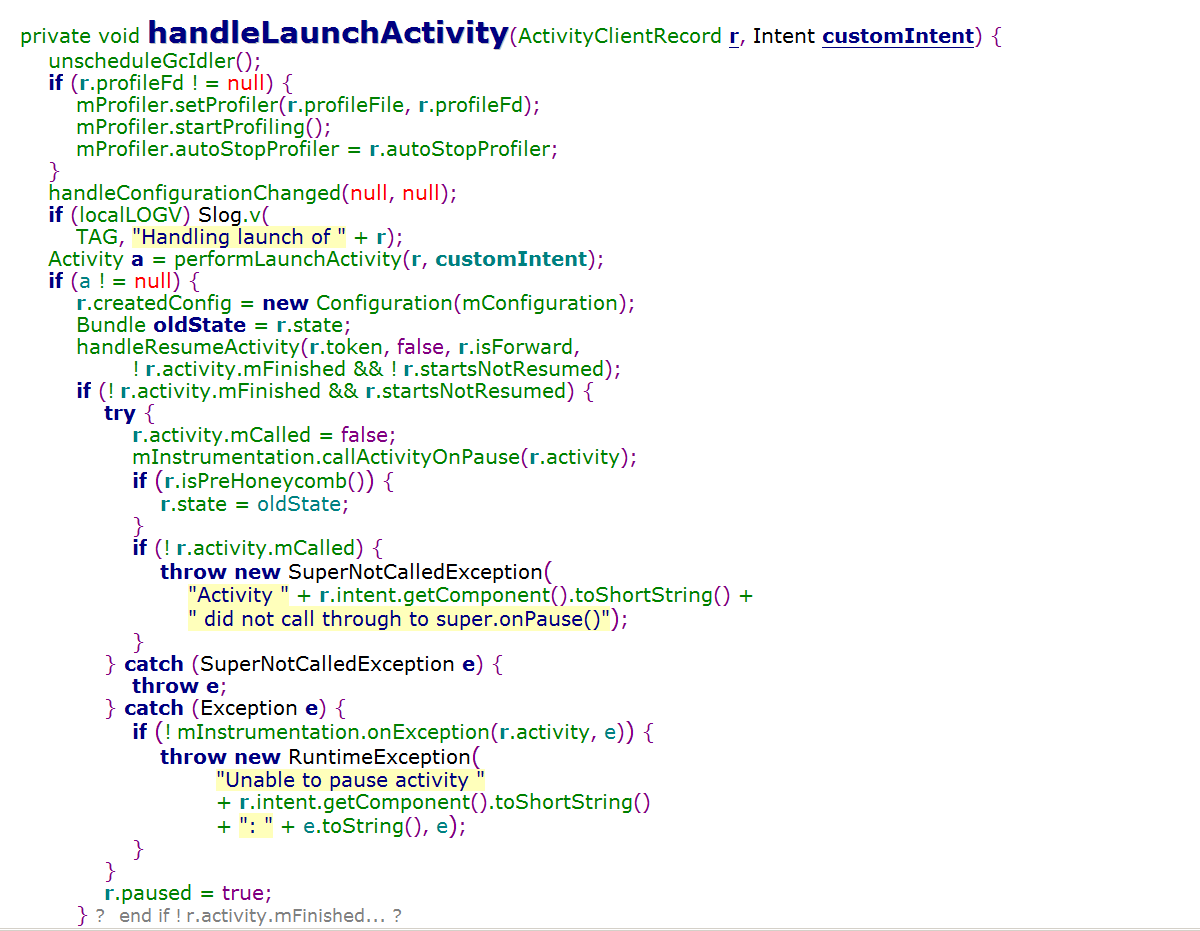
handleInstallProvider((ProviderInfo) msg.obj);

break;

}

if (DEBUG\_MESSAGES) Slog.v(TAG, "<<< done: " + codeToString(msg.what));

}

再看handlePeformLanchActivity方法, 发送一个已中止状态的进程间通信请求给ActivityThread.   


以MyActivity继承为例（假定是我们app的第一个Activity），MyActivity被启动，共要经历5个大的步骤：   
MyActivity组件是由Launcher组件来启动的，而Launcher组件又是Activity管理服务ActivityManagerService来启动的。

MyActivity组件、Launcher组件和ActivityManagerService组件分别运行在不同的进程中，MyActivity启动过程就涉及了   
三个进程，这三个进程通过Binder进程间通信来完成。

Launcher组件启动MyActivity的过程如下：

**Lanuncher组件向ActivityManagerService发送一个启动MyActivity组件的进程间通信请求**

**ActivityManagerService首先将要启动的MyActivity组件的信息保存下来，然后再向Launcher组件发送一个中止状态的进程间通信请求**

**Launcher组件进入到中止状态后，就会向ActivityManagerService发送一个已中止状态的进程间通信请求，以便ActivityManagerService可以继续执行MyActivity组件的操作**

**ActivityManagerService发现用来运行MyActivity组件的应用程序不存在，因此，它就会启动一个新的应用程序进程**

**新的应用程序进程启动完成后，就会向ActivityManagerService发送一个启动完成的进程间通信请求，以便ActivityManagerService可以继续执行启动MyActivity组件的操作。**

**ActivityManagerService将第2步保存下来的MyActivity组件的信息发送给第4步创建的应用程序进程，以便它可以将yActivity组件启动起来。**