**Activity的启动流程这一篇够了**

Activity启动.png

**来了小伙子，先自我介绍一下吧**

* 我叫\*\*\*, 我精通Android系统的.....

**什么？你精通Android系统？来，你给我说下Activity的启动流程。**

Activity的启动过程是系统中比较有代表意义的过程，涉及到了各个进程之间的相互交互，以及生命周期的回调控制，这也是为什么在面试过程出现频率这么高的原因之一。

Activity的启动流程在API28以后变成了事务启动的方式，相比之前版本的逻辑有了一些变化，但是万变不离其宗，大致流程还是类似的，只是增加了一些类，用来更好的划分职责，更优的处理逻辑。

在开始之前，我们先简单普及下基础知识。

**Activity 启动主要涉及到3个进程。**

* 1. **系统进程 SystemServer** （负责管理整个framework，是Zygote孵化的第一个进程）
  2. **App进程**（App进程是用户点击桌面icon时，通过Launcher进程请求SystemServer，再调用Zygote孵化的）
  3. **Zygote进程**（所有进程孵化都由Zygote完成，而Zygote是init进程的子进程，也由init进程孵化）
  4. 如果点击桌面icon启动还会涉及到 **Launcher进程**（Zygote孵化的第一个应用进程）

**进程之间靠什么通信？**

我们都知道进程与进程之间是数据隔离的，无法相互访问数据，所以进程之间通信是靠Binder来完成的。

**面试官可能会问你 为什么会用Binder通信，Binder相比Socket有什么优势呢？**

* 我想都没想直接说 **1次拷贝，因为1次拷贝啊**

然而果然被追问了 **为啥Binder能做到1次拷贝，而其他的技术是2次拷贝**

* 我当时比较年轻，虽然我不知道具体细节，但是我知道内存映射。  
  我直接拍脑门就扯 **因为Binder在server端于内核中通过mmap技术建立了内存映射，当我们Client与Server通信的时候，只需要把Client端的通信数据拷贝到内核中与Server映射好的内存区域就相当于拷贝到Server端了........**

**好，可以描述下具体的映射怎么做的么？....**

这里我推荐一篇Binder的文章，我认为是大量介绍Binder的博客中描述的很透彻的，没有废话，适合急性子。有兴趣的同学可以学习以下，**防止面试的时候跟我一样叫大哥。**  
[Android Binder通信一次拷贝你真的理解了吗？](https://links.jianshu.com/go?to=https%3A%2F%2Fmy.oschina.net%2Fu%2F3897543%2Fblog%2F4901658)

**Activity启动流程主要包含几步？**

我们以点击Launcher的一个icon为开始，整体扯一下Activity的启动过程，桌面其实就是LauncherApp的一个Activity

* 1. 当点击Launcher的icon开始，Launcher进程会像AMS发送点击icon的启动信息（这些信息就是在AndroidMainifest.xml中<intent-filter>标签定义的启动信息，数据由PackageManagerService解析出来）
  2. AMS收到信息后会先后经过ActivityTaskManagerService->ActivityStartController->ActivityStarter内部类Request，然后把信息存到Request中，并通知Launcher进程让Activity休眠（补充个小知识点，这个过程会检测Activity在AndroidMainifest.xml的注册，如果没有注册就报错了）
  3. Launcher进程的ApplicationThread对象收到消息后调用handlePauseActivity()进行暂停，并通知AMS已经暂停。  
     实现细节：ActivityThread.sendMessage()通过ActivityThread的H类发送Handler消息，然后触发 mTransactionExecutor.execute(transaction)，  
     执行过程中依赖ActivityClientRecord.mLifecycleState数值并通过ClientTransactionHandler抽象类的实现（ActivityThread）进行分发。  
     注 ：ActivityClientRecord.mLifecycleState（-1 ~ 7分别代表 UNDEFINED, PRE\_ON\_CREATE, ON\_CREATE, ON\_START, ON\_RESUME, ON\_PAUSE, ON\_STOP, ON\_DESTROY, ON\_RESTART）
  4. AMS收到Launcher的已暂停消息后，会检查要启动的Activity所在的进程是否已经启动了，如果已经启动了就打开，如果未启动则通过Process.start(android.app.ActivityThread)来启动一个新的进程。
  5. 进程创建好以后，会调用ActivityThread.main(),初始化MainLooper，并创建Application对象。然后Instrumentation.newApplication()反射创建Application，创建ContextImpl通过Application的attach方法与Application进行绑定，最终会调用Instrumentation.callApplicationOnCreate执行Application的onCreate函数进行一些初始化的工作。完成后会通知AMS进程已经启动好了。  
     通知过程：通过IActivityManager.attachApplication（IApplicationThread thread, long startSeq），将Application对象传入AMS
  6. AMS收到app进程启动成功的消息后，从ActivityTaskManagerService中取出对应的Activity启动信息， 并通过ApplicationThreadProxy对象，调用其scheduleTransaction(ClientTransaction transaction)方法，具体要启动的Activity都在ClientTransaction对象中。
  7. app进程的ApplicationThread收到消息后会调用ActiivtyThread.sendMessage()，通过H发送Handler消息，在handleMessage方法的内部又会调用 mTransactionExecutor.execute(transaction);具体参考第3步  
     最终调用performLaunchActivity方法创建activity和context并将其做关联，然后通过mInstrumentation.callActivityOnCreate()->Activity.performCreate()->Activity.onCreate()回调到了Activity的生命周期。

**Activity启动过程主要涉及哪些类**

* 为了防止后续大量的源码分析过程中影响整体的链路关系，在分析完源码后，我总结了一下相关类，以及调用方法，具体看以下描述。

启动一个Activity一般通过startActivity()

**startActivity(new Intent(OneActivity.this,TwoActivity.class));**

* Activity  
  startActivity()  
  startActivityForResult()
* Instrumentation 用于实现应用程序检测代码的基类。当在打开程序指令的时候运行，这个类将在任何应用程- 序代码之前为您实例化，可以监视系统与应用程序的所有交互。在AndroidManifest.xml文件的<instrumentation>标记。  
  execStartActivity()
* ActivityManagerService  
  startActivity()  
  startActivityAsUser()
* ActivityStarter 用于解释如何启动活动。此类记录所有逻辑，用于确定如何将意图和标志转换为Activity以及关联的任务和堆栈。  
  execute()  
  startActivity()  
  startActivityUnchecked()
* ActivityStackSupervisor  
  resumeFocusedStackTopActivityLocked()
* ActivityStack 单个Activity堆栈的状态和管理  
  resumeTopActivityUncheckedLocked()  
  resumeTopActivityInnerLocked()
* ActivityStackSupervisor Activity堆栈管理  
  startSpecificActivityLocked()  
  realStartActivityLocked()
* ClientTransaction 一种容器，它保存一系列消息（比如声明周期的状态），这些消息可以发送给client。  
  ClientTransaction.obtain(app.thread, r.appToken)//初始化  
  addCallback((LaunchActivityItem.obtain(new Intent(r.intent),...)
* ClientLifecycleManager //该类能够组合多个client生命周期转换请求/回调，并将它们作为单个事务执行  
  scheduleTransaction(clientTransaction)
* ClientTransaction  
  schedule()
* ApplicationThread  
  scheduleTransaction()
* ActivityThread 它管理应用程序进程中主线程中执行的调度和执行活动、广播以及活动管理器请求的其他操作。  
  scheduleTransaction()  
  sendMessage(ActivityThread.H.EXECUTE\_TRANSACTION, transaction);
* ClientTransactionHandler //ActivityThread 继承 ClientTransactionHandler，所以调用了父类scheduleTransaction（）  
  scheduleTransaction()
* TransactionExecutor 以正确的顺序管理事务执行  
  execute()  
  executeCallbacks()  
  transaction.getCallbacks().get(i).execute()
* LaunchActivityItem 请求启动Activity  
  execute()
* ActivityThread  
  handleLaunchActivity()  
  performLaunchActivity()
* Instrumentation  
  callActivityOnCreate()
* Activity  
  onCreate()

**源码层分析整个链路**

我们先看下正常启动Activity的方式，一般我们都会通过以下的方式启动一个新的Activity。

**startActivity(new Intent(OneActivity.this,TwoActivity.class));**

其实这是在Activity中的调用方式，调用的即是父类Activity的startActivity()方法,因参数不同分为两个方法，具体如下

**@Override**

**public void startActivity(Intent intent) {**

**this.startActivity(intent, null);**

**}**

**@Override**

**public void startActivity(Intent intent, @Nullable Bundle options) {**

**if (options != null) {**

**startActivityForResult(intent, -1, options);**

**} else {**

**// Note we want to go through this call for compatibility with**

**// applications that may have overridden the method.**

**startActivityForResult(intent, -1);**

**}**

**}**

最终调用的是startActivityForResult()

**public void startActivityForResult(@RequiresPermission Intent intent, int requestCode) {**

**startActivityForResult(intent, requestCode, null);**

**}**

**public void startActivityForResult(@RequiresPermission Intent intent, int requestCode,**

**@Nullable Bundle options) {**

**if (mParent == null) {**

**options = transferSpringboardActivityOptions(options);**

**Instrumentation.ActivityResult ar =**

**//主要看这里mInstrumentation为Instrumentation对象**

**mInstrumentation.execStartActivity(**

**this, mMainThread.getApplicationThread(), mToken, this,**

**intent, requestCode, options);**

**} else {**

**if (options != null) {**

**mParent.startActivityFromChild(this, intent, requestCode, options);**

**} else {**

**// Note we want to go through this method for compatibility with**

**// existing applications that may have overridden it.**

**mParent.startActivityFromChild(this, intent, requestCode);**

**}**

**}**

**}**

核心逻辑是调用了Instrumentation.execStartActivity()

**public ActivityResult execStartActivity(**

**Context who, IBinder contextThread, IBinder token, Activity target,**

**Intent intent, int requestCode, Bundle options) {**

**...**

**try {**

**intent.migrateExtraStreamToClipData();**

**intent.prepareToLeaveProcess(who);**

**//核心在这一句**

**int result = ActivityManager.getService()**

**.startActivity(whoThread, who.getBasePackageName(), intent,**

**intent.resolveTypeIfNeeded(who.getContentResolver()),**

**token, target != null ? target.mEmbeddedID : null,**

**requestCode, 0, null, options);**

**checkStartActivityResult(result, intent);**

**} catch (RemoteException e) {**

**throw new RuntimeException("Failure from system", e);**

**}**

**return null;**

**}**

ActivityManager.getService()这个获取的是谁？  
逻辑是从IActivityManagerSingleton.get()获取，那IActivityManagerSingleton又是谁？  
IActivityManagerSingleton是这么定义的Singleton<IActivityManager> IActivityManagerSingleton  
get取出来的是IActivityManager,看这个大写I开头就知道是一个接口，实际调用过的是它的实现ActivityManagerService。  
**ActivityManagerService. startActivity()**

**@Override**

**public final int startActivity(IApplicationThread caller, String callingPackage,**

**Intent intent, String resolvedType, IBinder resultTo, String resultWho, int requestCode,**

**int startFlags, ProfilerInfo profilerInfo, Bundle bOptions) {**

**return startActivityAsUser(caller, callingPackage, intent, resolvedType, resultTo,**

**resultWho, requestCode, startFlags, profilerInfo, bOptions,**

**UserHandle.getCallingUserId());**

**}**

**@Override**

**public final int startActivityAsUser(IApplicationThread caller, String callingPackage,**

**Intent intent, String resolvedType, IBinder resultTo, String resultWho, int requestCode,**

**int startFlags, ProfilerInfo profilerInfo, Bundle bOptions, int userId) {**

**return startActivityAsUser(caller, callingPackage, intent, resolvedType, resultTo,**

**resultWho, requestCode, startFlags, profilerInfo, bOptions, userId,**

**true /\*validateIncomingUser\*/);**

**}**

**public final int startActivityAsUser(IApplicationThread caller, String callingPackage,**

**Intent intent, String resolvedType, IBinder resultTo, String resultWho, int requestCode,**

**int startFlags, ProfilerInfo profilerInfo, Bundle bOptions, int userId,**

**boolean validateIncomingUser) {**

**enforceNotIsolatedCaller("startActivity");**

**userId = mActivityStartController.checkTargetUser(userId, validateIncomingUser,**

**Binder.getCallingPid(), Binder.getCallingUid(), "startActivityAsUser");**

**// TODO: Switch to user app stacks here.**

**return mActivityStartController.obtainStarter(intent, "startActivityAsUser")**

**.setCaller(caller)**

**.setCallingPackage(callingPackage)**

**.setResolvedType(resolvedType)**

**.setResultTo(resultTo)**

**.setResultWho(resultWho)**

**.setRequestCode(requestCode)**

**.setStartFlags(startFlags)**

**.setProfilerInfo(profilerInfo)**

**.setActivityOptions(bOptions)**

**.setMayWait(userId)**

**.execute();**

**}**

mActivityStartController.obtainStarter实际调用的是ActivityStarter.execute(),连带调用到ActivityStarter.startActivity()

**private int startActivity(final ActivityRecord r, ActivityRecord sourceRecord,**

**IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,**

**int startFlags, boolean doResume, ActivityOptions options, TaskRecord inTask,**

**ActivityRecord[] outActivity) {**

**int result = START\_CANCELED;**

**try {**

**mService.mWindowManager.deferSurfaceLayout();**

**result = startActivityUnchecked(r, sourceRecord, voiceSession, voiceInteractor,**

**startFlags, doResume, options, inTask, outActivity);**

**} finally {**

**...**

**}**

**postStartActivityProcessing(r, result, mTargetStack);**

**return result;**

**}**

ActivityStarter.startActivityUnchecked()连带调用ActivityStackSupervisor.resumeFocusedStackTopActivityLocked();

**boolean resumeFocusedStackTopActivityLocked(**

**ActivityStack targetStack, ActivityRecord target, ActivityOptions targetOptions) {**

**if (!readyToResume()) {**

**return false;**

**}**

**if (targetStack != null && isFocusedStack(targetStack)) {**

**//主要看这里**

**return targetStack.resumeTopActivityUncheckedLocked(target, targetOptions);**

**}**

**final ActivityRecord r = mFocusedStack.topRunningActivityLocked();**

**if (r == null || !r.isState(RESUMED)) {**

**mFocusedStack.resumeTopActivityUncheckedLocked(null, null);**

**} else if (r.isState(RESUMED)) {**

**// Kick off any lingering app transitions form the MoveTaskToFront operation.**

**mFocusedStack.executeAppTransition(targetOptions);**

**}**

**return false;**

**}**

targetStack为ActivityStack对象，ActivityStack.resumeTopActivityUncheckedLocked()

**boolean resumeTopActivityUncheckedLocked(ActivityRecord prev, ActivityOptions options) {**

**if (mStackSupervisor.inResumeTopActivity) {**

**// 防止递归的**

**return false;**

**}**

**boolean result = false;**

**try {**

**// Protect against recursion.**

**mStackSupervisor.inResumeTopActivity = true;**

**//主要看这**

**result = resumeTopActivityInnerLocked(prev, options);**

**if (next == null || !next.canTurnScreenOn()) {**

**checkReadyForSleep();**

**}**

**} finally {**

**mStackSupervisor.inResumeTopActivity = false;**

**}**

**return result;**

**}**

ActivityStack.resumeTopActivityInnerLocked()调用了mStackSupervisor.startSpecificActivityLocked(next, true, true);其中mStackSupervisor为ActivityStackSupervisor。  
ActivityStackSupervisor.startSpecificActivityLocked()中调用  
ActivityStackSupervisor.realStartActivityLocked()

**final boolean realStartActivityLocked(ActivityRecord r, ProcessRecord app,**

**boolean andResume, boolean checkConfig) throws RemoteException {**

**...**

**// Schedule transaction.**

**mService.getLifecycleManager().scheduleTransaction(clientTransaction);**

**}**

注意这个clientTransaction对象，通过这种方式初始化

//app.thread为IApplicationThread

final ClientTransaction clientTransaction = ClientTransaction.obtain(app.thread, r.appToken);

// 注意下这个LaunchActivityItem.obtain

**clientTransaction.addCallback(LaunchActivityItem.obtain(new Intent(r.intent),**

**System.identityHashCode(r), r.info,**

**// TODO: Have this take the merged configuration instead of separate global**

**// and override configs.**

**mergedConfiguration.getGlobalConfiguration(),**

**mergedConfiguration.getOverrideConfiguration(), r.compat,**

**r.launchedFromPackage, task.voiceInteractor, app.repProcState, r.icicle,**

**r.persistentState, results, newIntents, mService.isNextTransitionForward(),**

**profilerInfo));**

ClientLifecycleManager.scheduleTransaction(clientTransaction);

**void scheduleTransaction(ClientTransaction transaction) throws RemoteException {**

**final IApplicationThread client = transaction.getClient();**

**transaction.schedule();**

**if (!(client instanceof Binder)) {**

**transaction.recycle();**

**}**

**}**

transaction.schedule();就要找到

**public void schedule() throws RemoteException {**

**mClient.scheduleTransaction(this);**

**}**

mClient即是以上描述的ApplicationThread,因此我们跟进ApplicationThread.scheduleTransaction（）

**@Override**

**public void scheduleTransaction**

**(ClientTransaction transaction) throws RemoteException {**

**ActivityThread.this.scheduleTransaction(transaction);**

**}**

我靠，调用了ActivityThread.scheduleTransaction(transaction),但是ActivityThread并没有scheduleTransaction(),所以我们找他继承的类ClientTransactionHandler，发现ClientTransactionHandler果然有scheduleTransaction()

**/\*\* Prepare and schedule transaction for execution. \*/**

**void scheduleTransaction(ClientTransaction transaction) {**

**transaction.preExecute(this);**

**sendMessage(ActivityThread.H.EXECUTE\_TRANSACTION, transaction);**

**}**

以上消息通过ActivityThread H对象进行发送具体解析也在ActivityThread中

**case EXECUTE\_TRANSACTION:**

**final ClientTransaction transaction = (ClientTransaction) msg.obj;**

**mTransactionExecutor.execute(transaction);**

**if (isSystem()) {**

**transaction.recycle();**

**}**

**// TODO(lifecycler): Recycle locally scheduled transactions.**

**break;**

我们可以看下TransactionExecutor.execute(transaction);

//首先，所有回调将按照它们在列表中出现的顺序执行。如果回调需要特定的执行前或执行后状态，

//则客户端将相应地进行转换。然后客户端将循环到最终的生命周期状态（如果提供）。

//否则，它将保持在回调所需的初始状态或最后状态。

**public void execute(ClientTransaction transaction) {**

**final IBinder token = transaction.getActivityToken();**

**log("Start resolving transaction for client: " + mTransactionHandler + ", token: " + token);**

**executeCallbacks(transaction);**

**executeLifecycleState(transaction);**

**mPendingActions.clear();**

**log("End resolving transaction");**

**}**

这里我们主要看下executeCallbacks()方法

**/\*\* Transition to the final state if requested by the transaction. \*/**

**public void executeCallbacks(ClientTransaction transaction) {**

**//transaction.getCallbacks()会获取clientTransaction.addCallbacks()的数据。**

**final List<ClientTransactionItem> callbacks = transaction.getCallbacks();**

**if (callbacks == null) {**

**// No callbacks to execute, return early.**

**return;**

**}**

**log("Resolving callbacks");**

**final IBinder token = transaction.getActivityToken();**

**ActivityClientRecord r = mTransactionHandler.getActivityClient(token);**

**// In case when post-execution state of the last callback matches the final state requested**

**// for the activity in this transaction, we won't do the last transition here and do it when**

**// moving to final state instead (because it may contain additional parameters from server).**

**final ActivityLifecycleItem finalStateRequest = transaction.getLifecycleStateRequest();**

**final int finalState = finalStateRequest != null ? finalStateRequest.getTargetState()**

**: UNDEFINED;**

**// Index of the last callback that requests some post-execution state.**

**final int lastCallbackRequestingState = lastCallbackRequestingState(transaction);**

**final int size = callbacks.size();**

**for (int i = 0; i < size; ++i) {**

**final ClientTransactionItem item = callbacks.get(i);**

**log("Resolving callback: " + item);**

**final int postExecutionState = item.getPostExecutionState();**

**final int closestPreExecutionState = mHelper.getClosestPreExecutionState(r,**

**item.getPostExecutionState());**

**if (closestPreExecutionState != UNDEFINED) {**

**cycleToPath(r, closestPreExecutionState);**

**}**

**item.execute(mTransactionHandler, token, mPendingActions);**

**item.postExecute(mTransactionHandler, token, mPendingActions);**

**if (r == null) {**

**// Launch activity request will create an activity record.**

**r = mTransactionHandler.getActivityClient(token);**

**}**

**if (postExecutionState != UNDEFINED && r != null) {**

**// Skip the very last transition and perform it by explicit state request instead.**

**final boolean shouldExcludeLastTransition =**

**i == lastCallbackRequestingState && finalState == postExecutionState;**

**cycleToPath(r, postExecutionState, shouldExcludeLastTransition);**

**}**

**}**

**}**

transaction.getCallbacks()就是上述过程中ClientTransaction创建过程赋值的LaunchActivityItem对象，因此主要看下LaunchActivityItem.execute()

**@Override**

**public void execute(ClientTransactionHandler client, IBinder token,**

**PendingTransactionActions pendingActions) {**

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

**ActivityClientRecord r = new ActivityClientRecord(token, mIntent, mIdent, mInfo,**

**mOverrideConfig, mCompatInfo, mReferrer, mVoiceInteractor, mState, mPersistentState,**

**mPendingResults, mPendingNewIntents, mIsForward,**

**mProfilerInfo, client);**

**client.handleLaunchActivity(r, pendingActions, null /\* customIntent \*/);**

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

**}**

此时的client为ActivityThread，因此调用ActivityThread.handleLaunchActivity()

**@Override**

**public Activity handleLaunchActivity(ActivityClientRecord r,**

**PendingTransactionActions pendingActions, Intent customIntent) {**

**// If we are getting ready to gc after going to the background, well**

**// we are back active so skip it.**

**unscheduleGcIdler();**

**mSomeActivitiesChanged = true;**

**if (r.profilerInfo != null) {**

**mProfiler.setProfiler(r.profilerInfo);**

**mProfiler.startProfiling();**

**}**

**// Make sure we are running with the most recent config.**

**handleConfigurationChanged(null, null);**

**if (localLOGV) Slog.v(**

**TAG, "Handling launch of " + r);**

**// Initialize before creating the activity**

**if (!ThreadedRenderer.sRendererDisabled) {**

**GraphicsEnvironment.earlyInitEGL();**

**}**

**WindowManagerGlobal.initialize();**

**final Activity a = performLaunchActivity(r, customIntent);**

**...**

**}**

**/\*\* Core implementation of activity launch. \*/**

**private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {**

**ActivityInfo aInfo = r.activityInfo;**

**if (r.packageInfo == null) {**

**r.packageInfo = getPackageInfo(aInfo.applicationInfo, r.compatInfo,**

**Context.CONTEXT\_INCLUDE\_CODE);**

**}**

**ComponentName component = r.intent.getComponent();**

**if (component == null) {**

**component = r.intent.resolveActivity(**

**mInitialApplication.getPackageManager());**

**r.intent.setComponent(component);**

**}**

**if (r.activityInfo.targetActivity != null) {**

**component = new ComponentName(r.activityInfo.packageName,**

**r.activityInfo.targetActivity);**

**}**

**ContextImpl appContext = createBaseContextForActivity(r);**

**Activity activity = null;**

**try {**

**java.lang.ClassLoader cl = appContext.getClassLoader();**

**activity = mInstrumentation.newActivity(**

**cl, component.getClassName(), r.intent);**

**StrictMode.incrementExpectedActivityCount(activity.getClass());**

**r.intent.setExtrasClassLoader(cl);**

**r.intent.prepareToEnterProcess();**

**if (r.state != null) {**

**r.state.setClassLoader(cl);**

**}**

**} catch (Exception e) {**

**...**

**}**

**}**

**try {**

**Application app = r.packageInfo.makeApplication(false, mInstrumentation);**

**...**

**activity.mCalled = false;**

**if (r.isPersistable()) {**

**mInstrumentation.callActivityOnCreate(activity, r.state, r.persistentState);**

**} else {**

**mInstrumentation.callActivityOnCreate(activity, r.state);**

**}**

**if (!activity.mCalled) {**

**}**

**r.activity = activity;**

**}**

**r.setState(ON\_CREATE);**

**mActivities.put(r.token, r);**

**} catch (SuperNotCalledException e) {**

**throw e;**

**} catch (Exception e) {**

**}**

**}**

**return activity;**

**}**

主要跟进mInstrumentation.callActivityOnCreate()

**public void callActivityOnCreate(Activity activity, Bundle icicle,**

**PersistableBundle persistentState) {**

**prePerformCreate(activity);**

**activity.performCreate(icicle, persistentState);**

**postPerformCreate(activity);**

**}**

Activity.performCreate()

**final void performCreate(Bundle icicle, PersistableBundle persistentState) {**

**mCanEnterPictureInPicture = true;**

**restoreHasCurrentPermissionRequest(icicle);**

**if (persistentState != null) {**

**onCreate(icicle, persistentState);**

**} else {**

**onCreate(icicle);**

**}**

**writeEventLog(LOG\_AM\_ON\_CREATE\_CALLED, "performCreate");**

**mActivityTransitionState.readState(icicle);**

**mVisibleFromClient = !mWindow.getWindowStyle().getBoolean(**

**com.android.internal.R.styleable.Window\_windowNoDisplay, false);**

**mFragments.dispatchActivityCreated();**

**mActivityTransitionState.setEnterActivityOptions(this, getActivityOptions());**

**}**

至此已经回调到了Activity的onCreate()方法，Activity也就正式启动了，后续就是对应的声明周期回调。