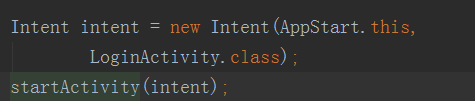
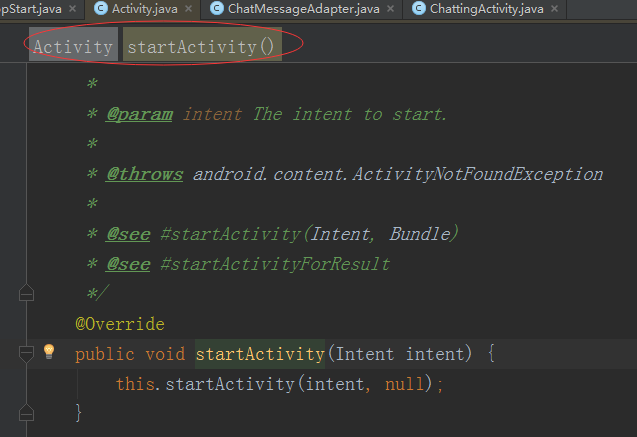
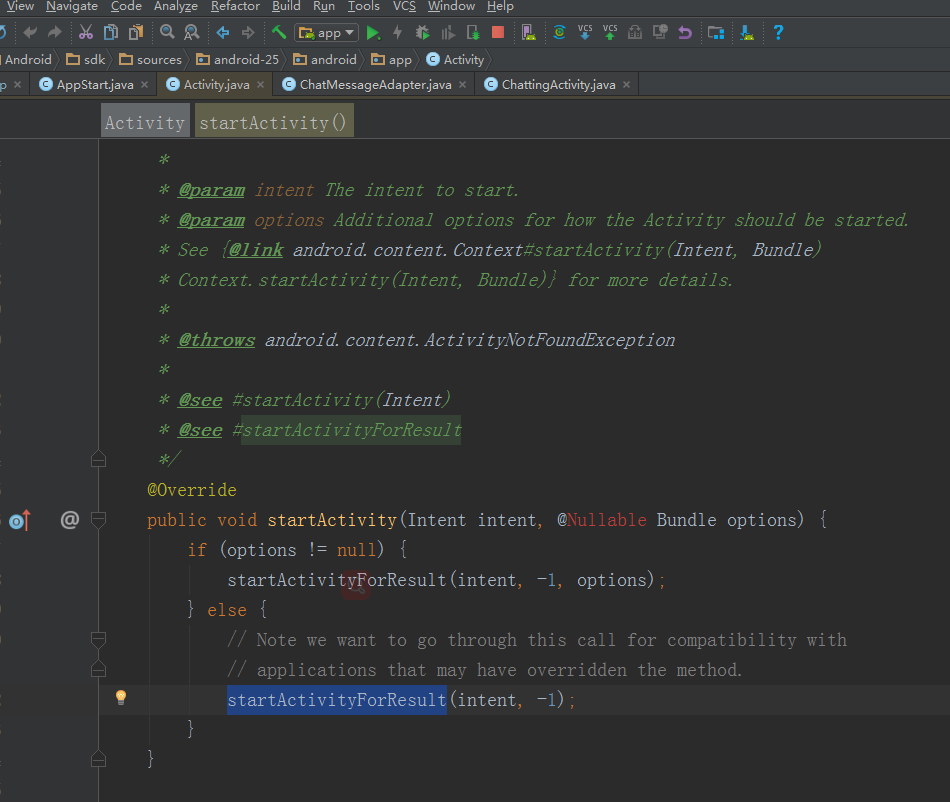
# Activity的启动流程



## 1、调用Activity的startActivity()方法

本质是调用actvityforResult方法





## 2、调用Instrumentation的execStartActivity方法

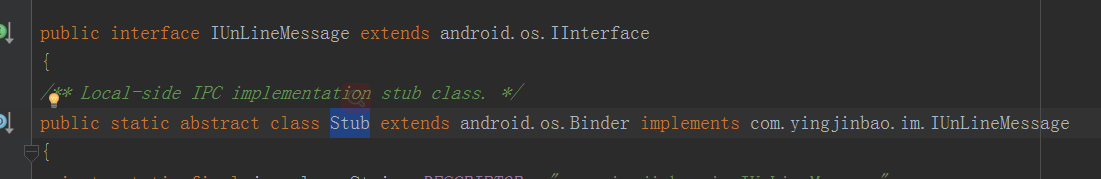
public ActivityResult execStartActivity(  
 Context who, IBinder contextThread, IBinder token, String target,  
 Intent intent, int requestCode, Bundle options) {  
 IApplicationThread whoThread = (IApplicationThread) contextThread;  
 if (mActivityMonitors != null) {  
 synchronized (mSync) {  
 final int N = mActivityMonitors.size();  
 for (int i=0; i<N; i++) {  
 final ActivityMonitor am = mActivityMonitors.get(i);  
 if (am.match(who, null, intent)) {  
 am.mHits++;  
 if (am.isBlocking()) {  
 return requestCode >= 0 ? am.getResult() : null;  
 }  
 break;  
 }  
 }  
 }  
 }  
 try {  
 intent.migrateExtraStreamToClipData();  
 intent.prepareToLeaveProcess(who);  
 int result = ActivityManagerNative.*getDefault*()  
 .startActivity(whoThread, who.getBasePackageName(), intent,  
 intent.resolveTypeIfNeeded(who.getContentResolver()),  
 token, target, requestCode, 0, null, options);  
 *checkStartActivityResult*(result, intent);  
 } catch (RemoteException e) {  
 throw new RuntimeException("Failure from system", e);  
 }  
 return null;  
}

private final Object mSync = new Object();同步锁

本质上通过该方法启动Activity：

int result = ActivityManagerNative.*getDefault*()  
 .startActivity(whoThread, who.getBasePackageName(), intent,  
 intent.resolveTypeIfNeeded(who.getContentResolver()),  
 token, target != null ? target.mEmbeddedID : null,  
 requestCode, 0, null, options);

## 3、IPC通信原理



实现stub，且实现stub的类运行在远程进程中。

## 4、AMS是运行在系统进程中

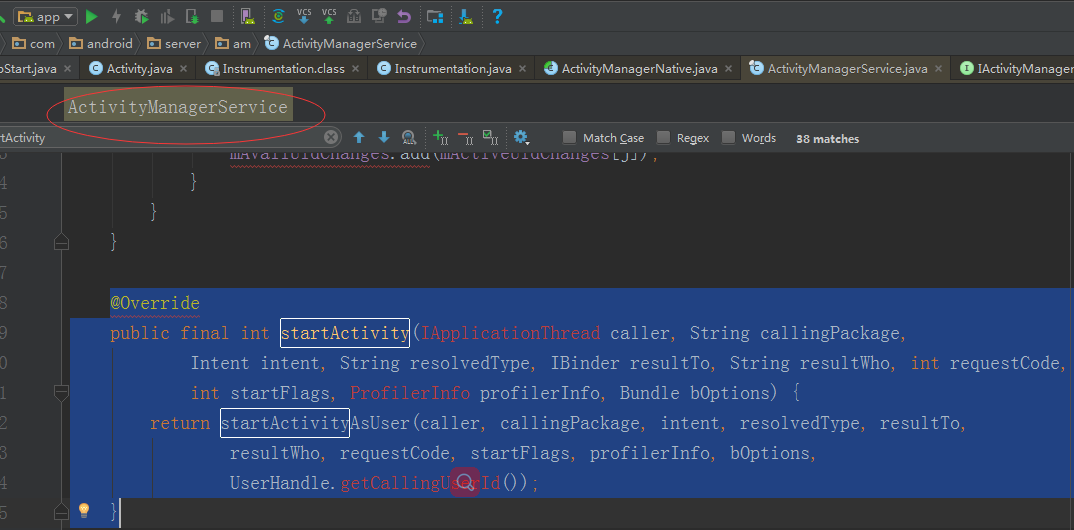
### 4.1 public interface IActivityManager extends IInterface

### 4.2 public abstract class ActivityManagerNative extends Binder implements IActivityManager

其中 ActivityManagerNative 类似一个stub。

### 4.3 public final class ActivityManagerService extends ActivityManagerNative implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback

## 5、启动activity转移到调用AMS的startActivity方法



AMS.java

@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());  
}  
  
final int startActivity(Intent intent, ActivityStackSupervisor.ActivityContainer container) {  
 enforceNotIsolatedCaller("ActivityContainer.startActivity");  
 final int userId = mUserController.handleIncomingUser(Binder.*getCallingPid*(),  
 Binder.*getCallingUid*(), mStackSupervisor.mCurrentUser, false,  
 ActivityManagerService.*ALLOW\_FULL\_ONLY*, "ActivityContainer", null);  
  
 // *TODO: Switch to user app stacks here.* String mimeType = intent.getType();  
 final Uri data = intent.getData();  
 if (mimeType == null && data != null && "content".equals(data.getScheme())) {  
 mimeType = getProviderMimeType(data, userId);  
 }  
 container.checkEmbeddedAllowedInner(userId, intent, mimeType);  
  
 intent.addFlags(FORCE\_NEW\_TASK\_FLAGS);  
 return mActivityStarter.startActivityMayWait(null, -1, null, intent, mimeType, null, null, null,  
 null, 0, 0, null, null, null, null, false, userId, container, null);  
}

### 5.1本质调用ams的startActivityAsUser方法

@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) {  
 enforceNotIsolatedCaller("startActivity");  
 userId = mUserController.handleIncomingUser(Binder.*getCallingPid*(), Binder.*getCallingUid*(),  
 userId, false, *ALLOW\_FULL\_ONLY*, "startActivity", null);  
 // *TODO: Switch to user app stacks here.* return **mActivityStarter.****startActivityMayWait**(caller, -1, callingPackage, intent,  
 resolvedType, null, null, resultTo, resultWho, requestCode, startFlags,  
 profilerInfo, null, null, bOptions, false, userId, null, null);  
}

### 5.2本质上调用的是ActivityStarter类的 startActivityMayWait方法

ActivityStarter.java

final int **startActivityMayWait**(IApplicationThread caller, int callingUid,  
 String callingPackage, Intent intent, String resolvedType,  
 IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,  
 IBinder resultTo, String resultWho, int requestCode, int startFlags,  
 ProfilerInfo profilerInfo, IActivityManager.WaitResult outResult, Configuration config,  
 Bundle bOptions, boolean ignoreTargetSecurity, int userId,  
 IActivityContainer iContainer, TaskRecord inTask) {  
 // Refuse possible leaked file descriptors  
 if (intent != null && intent.hasFileDescriptors()) {  
 throw new IllegalArgumentException("File descriptors passed in Intent");  
 }  
 mSupervisor.mActivityMetricsLogger.notifyActivityLaunching();  
 boolean componentSpecified = intent.getComponent() != null;  
  
 // Save a copy in case ephemeral needs it  
 final Intent ephemeralIntent = new Intent(intent);  
 // Don't modify the client's object!  
 intent = new Intent(intent);  
  
 ResolveInfo rInfo = mSupervisor.resolveIntent(intent, resolvedType, userId);  
 if (rInfo == null) {  
 UserInfo userInfo = mSupervisor.getUserInfo(userId);  
 if (userInfo != null && userInfo.isManagedProfile()) {  
 // Special case for managed profiles, if attempting to launch non-cryto aware  
 // app in a locked managed profile from an unlocked parent allow it to resolve  
 // as user will be sent via confirm credentials to unlock the profile.  
 UserManager userManager = UserManager.get(mService.mContext);  
 boolean profileLockedAndParentUnlockingOrUnlocked = false;  
 long token = Binder.*clearCallingIdentity*();  
 try {  
 UserInfo parent = userManager.getProfileParent(userId);  
 profileLockedAndParentUnlockingOrUnlocked = (parent != null)  
 && userManager.isUserUnlockingOrUnlocked(parent.id)  
 && !userManager.isUserUnlockingOrUnlocked(userId);  
 } finally {  
 Binder.*restoreCallingIdentity*(token);  
 }  
 if (profileLockedAndParentUnlockingOrUnlocked) {  
 rInfo = mSupervisor.resolveIntent(intent, resolvedType, userId,  
 PackageManager.*MATCH\_DIRECT\_BOOT\_AWARE* | PackageManager.*MATCH\_DIRECT\_BOOT\_UNAWARE*);  
 }  
 }  
 }  
 // Collect information about the target of the Intent.  
 ActivityInfo aInfo = mSupervisor.resolveActivity(intent, rInfo, startFlags, profilerInfo);  
  
 ActivityOptions options = ActivityOptions.fromBundle(bOptions);  
 ActivityStackSupervisor.ActivityContainer container =  
 (ActivityStackSupervisor.ActivityContainer)iContainer;  
 synchronized (mService) {  
 if (container != null && container.mParentActivity != null &&  
 container.mParentActivity.state != RESUMED) {  
 // Cannot start a child activity if the parent is not resumed.  
 return ActivityManager.START\_CANCELED;  
 }  
 final int realCallingPid = Binder.*getCallingPid*();  
 final int realCallingUid = Binder.*getCallingUid*();  
 int callingPid;  
 if (callingUid >= 0) {  
 callingPid = -1;  
 } else if (caller == null) {  
 callingPid = realCallingPid;  
 callingUid = realCallingUid;  
 } else {  
 callingPid = callingUid = -1;  
 }  
  
 final ActivityStack stack;  
 if (container == null || container.mStack.isOnHomeDisplay()) {  
 stack = mSupervisor.mFocusedStack;  
 } else {  
 stack = container.mStack;  
 }  
 stack.mConfigWillChange = config != null && mService.mConfiguration.diff(config) != 0;  
 if (DEBUG\_CONFIGURATION) Slog.v(*TAG\_CONFIGURATION*,  
 "Starting activity when config will change = " + stack.mConfigWillChange);  
  
 final long origId = Binder.*clearCallingIdentity*();  
  
 if (aInfo != null &&  
 (aInfo.applicationInfo.privateFlags  
 & ApplicationInfo.PRIVATE\_FLAG\_CANT\_SAVE\_STATE) != 0) {  
 // This may be a heavy-weight process! Check to see if we already  
 // have another, different heavy-weight process running.  
 if (aInfo.processName.equals(aInfo.applicationInfo.packageName)) {  
 final ProcessRecord heavy = mService.mHeavyWeightProcess;  
 if (heavy != null && (heavy.info.uid != aInfo.applicationInfo.uid  
 || !heavy.processName.equals(aInfo.processName))) {  
 int appCallingUid = callingUid;  
 if (caller != null) {  
 ProcessRecord callerApp = mService.getRecordForAppLocked(caller);  
 if (callerApp != null) {  
 appCallingUid = callerApp.info.uid;  
 } else {  
 Slog.w(*TAG*, "Unable to find app for caller " + caller  
 + " (pid=" + callingPid + ") when starting: "  
 + intent.toString());  
 ActivityOptions.abort(options);  
 return ActivityManager.START\_PERMISSION\_DENIED;  
 }  
 }  
  
 IIntentSender target = mService.getIntentSenderLocked(  
 ActivityManager.INTENT\_SENDER\_ACTIVITY, "android",  
 appCallingUid, userId, null, null, 0, new Intent[] { intent },  
 new String[] { resolvedType }, PendingIntent.*FLAG\_CANCEL\_CURRENT* | PendingIntent.*FLAG\_ONE\_SHOT*, null);  
  
 Intent newIntent = new Intent();  
 if (requestCode >= 0) {  
 // Caller is requesting a result.  
 newIntent.putExtra(HeavyWeightSwitcherActivity.KEY\_HAS\_RESULT, true);  
 }  
 newIntent.putExtra(HeavyWeightSwitcherActivity.KEY\_INTENT,  
 new IntentSender(target));  
 if (heavy.activities.size() > 0) {  
 ActivityRecord hist = heavy.activities.get(0);  
 newIntent.putExtra(HeavyWeightSwitcherActivity.KEY\_CUR\_APP,  
 hist.packageName);  
 newIntent.putExtra(HeavyWeightSwitcherActivity.KEY\_CUR\_TASK,  
 hist.task.taskId);  
 }  
 newIntent.putExtra(HeavyWeightSwitcherActivity.KEY\_NEW\_APP,  
 aInfo.packageName);  
 newIntent.setFlags(intent.getFlags());  
 newIntent.setClassName("android",  
 HeavyWeightSwitcherActivity.class.getName());  
 intent = newIntent;  
 resolvedType = null;  
 caller = null;  
 callingUid = Binder.*getCallingUid*();  
 callingPid = Binder.*getCallingPid*();  
 componentSpecified = true;  
 rInfo = mSupervisor.resolveIntent(intent, null /\*resolvedType\*/, userId);  
 aInfo = rInfo != null ? rInfo.activityInfo : null;  
 if (aInfo != null) {  
 aInfo = mService.getActivityInfoForUser(aInfo, userId);  
 }  
 }  
 }  
 }  
  
 final ActivityRecord[] outRecord = new ActivityRecord[1];  
 int res = **startActivityLocked**(caller, intent, ephemeralIntent, resolvedType,  
 aInfo, rInfo, voiceSession, voiceInteractor,  
 resultTo, resultWho, requestCode, callingPid,  
 callingUid, callingPackage, realCallingPid, realCallingUid, startFlags,  
 options, ignoreTargetSecurity, componentSpecified, outRecord, container,  
 inTask);  
  
 Binder.*restoreCallingIdentity*(origId);  
  
 if (stack.mConfigWillChange) {  
 // If the caller also wants to switch to a new configuration,  
 // do so now. This allows a clean switch, as we are waiting  
 // for the current activity to pause (so we will not destroy  
 // it), and have not yet started the next activity.  
 mService.enforceCallingPermission(android.Manifest.permission.*CHANGE\_CONFIGURATION*,  
 "updateConfiguration()");  
 stack.mConfigWillChange = false;  
 if (DEBUG\_CONFIGURATION) Slog.v(*TAG\_CONFIGURATION*,  
 "Updating to new configuration after starting activity.");  
 mService.updateConfigurationLocked(config, null, false);  
 }  
  
 if (outResult != null) {  
 outResult.result = res;  
 if (res == ActivityManager.START\_SUCCESS) {  
 mSupervisor.mWaitingActivityLaunched.add(outResult);  
 do {  
 try {  
 mService.wait();  
 } catch (InterruptedException e) {  
 }  
 } while (outResult.result != START\_TASK\_TO\_FRONT  
 && !outResult.timeout && outResult.who == null);  
 if (outResult.result == START\_TASK\_TO\_FRONT) {  
 res = START\_TASK\_TO\_FRONT;  
 }  
 }  
 if (res == START\_TASK\_TO\_FRONT) {  
 ActivityRecord r = stack.topRunningActivityLocked();  
 if (r.nowVisible && r.state == RESUMED) {  
 outResult.timeout = false;  
 outResult.who = new ComponentName(r.info.packageName, r.info.name);  
 outResult.totalTime = 0;  
 outResult.thisTime = 0;  
 } else {  
 outResult.thisTime = SystemClock.*uptimeMillis*();  
 mSupervisor.mWaitingActivityVisible.add(outResult);  
 do {  
 try {  
 mService.wait();  
 } catch (InterruptedException e) {  
 }  
 } while (!outResult.timeout && outResult.who == null);  
 }  
 }  
 }  
  
 final ActivityRecord launchedActivity = mReusedActivity != null  
 ? mReusedActivity : outRecord[0];  
 mSupervisor.mActivityMetricsLogger.notifyActivityLaunched(res, launchedActivity);  
 return res;  
 }  
}

### 5.3 startActivityMayWait方法内部调用startActivityLocked方法，startActivityLocked调用startActivityUnchecked方法

final int startActivityLocked(IApplicationThread caller, Intent intent, Intent ephemeralIntent,  
 String resolvedType, ActivityInfo aInfo, ResolveInfo rInfo,  
 IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,  
 IBinder resultTo, String resultWho, int requestCode, int callingPid, int callingUid,  
 String callingPackage, int realCallingPid, int realCallingUid, int startFlags,  
 ActivityOptions options, boolean ignoreTargetSecurity, boolean componentSpecified,  
 ActivityRecord[] outActivity, ActivityStackSupervisor.ActivityContainer container,  
 TaskRecord inTask) {  
 int err = ActivityManager.START\_SUCCESS;  
  
 ProcessRecord callerApp = null;  
 if (caller != null) {  
 callerApp = mService.getRecordForAppLocked(caller);  
 if (callerApp != null) {  
 callingPid = callerApp.pid;  
 callingUid = callerApp.info.uid;  
 } else {  
 Slog.w(*TAG*, "Unable to find app for caller " + caller  
 + " (pid=" + callingPid + ") when starting: "  
 + intent.toString());  
 err = ActivityManager.START\_PERMISSION\_DENIED;  
 }  
 }  
  
 final int userId = aInfo != null ? UserHandle.getUserId(aInfo.applicationInfo.uid) : 0;  
  
 if (err == ActivityManager.START\_SUCCESS) {  
 Slog.i(*TAG*, "START u" + userId + " {" + intent.toShortString(true, true, true, false)  
 + "} from uid " + callingUid  
 + " on display " + (container == null ? (mSupervisor.mFocusedStack == null ?  
 Display.*DEFAULT\_DISPLAY* : mSupervisor.mFocusedStack.mDisplayId) :  
 (container.mActivityDisplay == null ? Display.*DEFAULT\_DISPLAY* :  
 container.mActivityDisplay.mDisplayId)));  
 }  
  
 ActivityRecord sourceRecord = null;  
 ActivityRecord resultRecord = null;  
 if (resultTo != null) {  
 sourceRecord = mSupervisor.isInAnyStackLocked(resultTo);  
 if (DEBUG\_RESULTS) Slog.v(*TAG\_RESULTS*,  
 "Will send result to " + resultTo + " " + sourceRecord);  
 if (sourceRecord != null) {  
 if (requestCode >= 0 && !sourceRecord.finishing) {  
 resultRecord = sourceRecord;  
 }  
 }  
 }  
  
 final int launchFlags = intent.getFlags();  
  
 if ((launchFlags & Intent.*FLAG\_ACTIVITY\_FORWARD\_RESULT*) != 0 && sourceRecord != null) {  
 // Transfer the result target from the source activity to the new  
 // one being started, including any failures.  
 if (requestCode >= 0) {  
 ActivityOptions.abort(options);  
 return ActivityManager.START\_FORWARD\_AND\_REQUEST\_CONFLICT;  
 }  
 resultRecord = sourceRecord.resultTo;  
 if (resultRecord != null && !resultRecord.isInStackLocked()) {  
 resultRecord = null;  
 }  
 resultWho = sourceRecord.resultWho;  
 requestCode = sourceRecord.requestCode;  
 sourceRecord.resultTo = null;  
 if (resultRecord != null) {  
 resultRecord.removeResultsLocked(sourceRecord, resultWho, requestCode);  
 }  
 if (sourceRecord.launchedFromUid == callingUid) {  
 // The new activity is being launched from the same uid as the previous  
 // activity in the flow, and asking to forward its result back to the  
 // previous. In this case the activity is serving as a trampoline between  
 // the two, so we also want to update its launchedFromPackage to be the  
 // same as the previous activity. Note that this is safe, since we know  
 // these two packages come from the same uid; the caller could just as  
 // well have supplied that same package name itself. This specifially  
 // deals with the case of an intent picker/chooser being launched in the app  
 // flow to redirect to an activity picked by the user, where we want the final  
 // activity to consider it to have been launched by the previous app activity.  
 callingPackage = sourceRecord.launchedFromPackage;  
 }  
 }  
  
 if (err == ActivityManager.START\_SUCCESS && intent.getComponent() == null) {  
 // We couldn't find a class that can handle the given Intent.  
 // That's the end of that!  
 err = ActivityManager.START\_INTENT\_NOT\_RESOLVED;  
 }  
  
 if (err == ActivityManager.START\_SUCCESS && aInfo == null) {  
 // We couldn't find the specific class specified in the Intent.  
 // Also the end of the line.  
 err = ActivityManager.START\_CLASS\_NOT\_FOUND;  
 }  
  
 if (err == ActivityManager.START\_SUCCESS && sourceRecord != null  
 && sourceRecord.task.voiceSession != null) {  
 // If this activity is being launched as part of a voice session, we need  
 // to ensure that it is safe to do so. If the upcoming activity will also  
 // be part of the voice session, we can only launch it if it has explicitly  
 // said it supports the VOICE category, or it is a part of the calling app.  
 if ((launchFlags & *FLAG\_ACTIVITY\_NEW\_TASK*) == 0  
 && sourceRecord.info.applicationInfo.uid != aInfo.applicationInfo.uid) {  
 try {  
 intent.addCategory(Intent.*CATEGORY\_VOICE*);  
 if (!AppGlobals.getPackageManager().activitySupportsIntent(  
 intent.getComponent(), intent, resolvedType)) {  
 Slog.w(*TAG*,  
 "Activity being started in current voice task does not support voice: "  
 + intent);  
 err = ActivityManager.START\_NOT\_VOICE\_COMPATIBLE;  
 }  
 } catch (RemoteException e) {  
 Slog.w(*TAG*, "Failure checking voice capabilities", e);  
 err = ActivityManager.START\_NOT\_VOICE\_COMPATIBLE;  
 }  
 }  
 }  
  
 if (err == ActivityManager.START\_SUCCESS && voiceSession != null) {  
 // If the caller is starting a new voice session, just make sure the target  
 // is actually allowing it to run this way.  
 try {  
 if (!AppGlobals.getPackageManager().activitySupportsIntent(intent.getComponent(),  
 intent, resolvedType)) {  
 Slog.w(*TAG*,  
 "Activity being started in new voice task does not support: "  
 + intent);  
 err = ActivityManager.START\_NOT\_VOICE\_COMPATIBLE;  
 }  
 } catch (RemoteException e) {  
 Slog.w(*TAG*, "Failure checking voice capabilities", e);  
 err = ActivityManager.START\_NOT\_VOICE\_COMPATIBLE;  
 }  
 }  
  
 final ActivityStack resultStack = resultRecord == null ? null : resultRecord.task.stack;  
  
 if (err != START\_SUCCESS) {  
 if (resultRecord != null) {  
 resultStack.sendActivityResultLocked(  
 -1, resultRecord, resultWho, requestCode, *RESULT\_CANCELED*, null);  
 }  
 ActivityOptions.abort(options);  
 return err;  
 }  
  
 boolean abort = !mSupervisor.checkStartAnyActivityPermission(intent, aInfo, resultWho,  
 requestCode, callingPid, callingUid, callingPackage, ignoreTargetSecurity, callerApp,  
 resultRecord, resultStack, options);  
 abort |= !mService.mIntentFirewall.checkStartActivity(intent, callingUid,  
 callingPid, resolvedType, aInfo.applicationInfo);  
  
 if (mService.mController != null) {  
 try {  
 // The Intent we give to the watcher has the extra data  
 // stripped off, since it can contain private information.  
 Intent watchIntent = intent.cloneFilter();  
 abort |= !mService.mController.activityStarting(watchIntent,  
 aInfo.applicationInfo.packageName);  
 } catch (RemoteException e) {  
 mService.mController = null;  
 }  
 }  
  
 mInterceptor.setStates(userId, realCallingPid, realCallingUid, startFlags, callingPackage);  
 mInterceptor.intercept(intent, rInfo, aInfo, resolvedType, inTask, callingPid, callingUid,  
 options);  
 intent = mInterceptor.mIntent;  
 rInfo = mInterceptor.mRInfo;  
 aInfo = mInterceptor.mAInfo;  
 resolvedType = mInterceptor.mResolvedType;  
 inTask = mInterceptor.mInTask;  
 callingPid = mInterceptor.mCallingPid;  
 callingUid = mInterceptor.mCallingUid;  
 options = mInterceptor.mActivityOptions;  
 if (abort) {  
 if (resultRecord != null) {  
 resultStack.sendActivityResultLocked(-1, resultRecord, resultWho, requestCode,  
 *RESULT\_CANCELED*, null);  
 }  
 // We pretend to the caller that it was really started, but  
 // they will just get a cancel result.  
 ActivityOptions.abort(options);  
 return START\_SUCCESS;  
 }  
  
 // If permissions need a review before any of the app components can run, we  
 // launch the review activity and pass a pending intent to start the activity  
 // we are to launching now after the review is completed.  
 if (Build.PERMISSIONS\_REVIEW\_REQUIRED && aInfo != null) {  
 if (mService.getPackageManagerInternalLocked().isPermissionsReviewRequired(  
 aInfo.packageName, userId)) {  
 IIntentSender target = mService.getIntentSenderLocked(  
 ActivityManager.INTENT\_SENDER\_ACTIVITY, callingPackage,  
 callingUid, userId, null, null, 0, new Intent[]{intent},  
 new String[]{resolvedType}, PendingIntent.*FLAG\_CANCEL\_CURRENT* | PendingIntent.*FLAG\_ONE\_SHOT*, null);  
  
 final int flags = intent.getFlags();  
 Intent newIntent = new Intent(Intent.ACTION\_REVIEW\_PERMISSIONS);  
 newIntent.setFlags(flags  
 | Intent.*FLAG\_ACTIVITY\_EXCLUDE\_FROM\_RECENTS*);  
 newIntent.putExtra(Intent.*EXTRA\_PACKAGE\_NAME*, aInfo.packageName);  
 newIntent.putExtra(Intent.*EXTRA\_INTENT*, new IntentSender(target));  
 if (resultRecord != null) {  
 newIntent.putExtra(Intent.EXTRA\_RESULT\_NEEDED, true);  
 }  
 intent = newIntent;  
  
 resolvedType = null;  
 callingUid = realCallingUid;  
 callingPid = realCallingPid;  
  
 rInfo = mSupervisor.resolveIntent(intent, resolvedType, userId);  
 aInfo = mSupervisor.resolveActivity(intent, rInfo, startFlags,  
 null /\*profilerInfo\*/);  
  
 if (DEBUG\_PERMISSIONS\_REVIEW) {  
 Slog.i(*TAG*, "START u" + userId + " {" + intent.toShortString(true, true,  
 true, false) + "} from uid " + callingUid + " on display "  
 + (container == null ? (mSupervisor.mFocusedStack == null ?  
 Display.*DEFAULT\_DISPLAY* : mSupervisor.mFocusedStack.mDisplayId) :  
 (container.mActivityDisplay == null ? Display.*DEFAULT\_DISPLAY* :  
 container.mActivityDisplay.mDisplayId)));  
 }  
 }  
 }  
  
 // If we have an ephemeral app, abort the process of launching the resolved intent.  
 // Instead, launch the ephemeral installer. Once the installer is finished, it  
 // starts either the intent we resolved here [on install error] or the ephemeral  
 // app [on install success].  
 if (rInfo != null && rInfo.ephemeralResolveInfo != null) {  
 intent = buildEphemeralInstallerIntent(intent, ephemeralIntent,  
 rInfo.ephemeralResolveInfo.getPackageName(), callingPackage, resolvedType,  
 userId);  
 resolvedType = null;  
 callingUid = realCallingUid;  
 callingPid = realCallingPid;  
  
 aInfo = mSupervisor.resolveActivity(intent, rInfo, startFlags, null /\*profilerInfo\*/);  
 }  
  
 ActivityRecord r = new ActivityRecord(mService, callerApp, callingUid, callingPackage,  
 intent, resolvedType, aInfo, mService.mConfiguration, resultRecord, resultWho,  
 requestCode, componentSpecified, voiceSession != null, mSupervisor, container,  
 options, sourceRecord);  
 if (outActivity != null) {  
 outActivity[0] = r;  
 }  
  
 if (r.appTimeTracker == null && sourceRecord != null) {  
 // If the caller didn't specify an explicit time tracker, we want to continue  
 // tracking under any it has.  
 r.appTimeTracker = sourceRecord.appTimeTracker;  
 }  
  
 final ActivityStack stack = mSupervisor.mFocusedStack;  
 if (voiceSession == null && (stack.mResumedActivity == null  
 || stack.mResumedActivity.info.applicationInfo.uid != callingUid)) {  
 if (!mService.checkAppSwitchAllowedLocked(callingPid, callingUid,  
 realCallingPid, realCallingUid, "Activity start")) {  
 PendingActivityLaunch pal = new PendingActivityLaunch(r,  
 sourceRecord, startFlags, stack, callerApp);  
 mPendingActivityLaunches.add(pal);  
 ActivityOptions.abort(options);  
 return ActivityManager.START\_SWITCHES\_CANCELED;  
 }  
 }  
  
 if (mService.mDidAppSwitch) {  
 // This is the second allowed switch since we stopped switches,  
 // so now just generally allow switches. Use case: user presses  
 // home (switches disabled, switch to home, mDidAppSwitch now true);  
 // user taps a home icon (coming from home so allowed, we hit here  
 // and now allow anyone to switch again).  
 mService.mAppSwitchesAllowedTime = 0;  
 } else {  
 mService.mDidAppSwitch = true;  
 }  
  
 doPendingActivityLaunchesLocked(false);  
  
 try {  
 mService.mWindowManager.deferSurfaceLayout();  
 err = **startActivityUnchecked**(r, sourceRecord, voiceSession, voiceInteractor, startFlags,  
 true, options, inTask);  
 } finally {  
 mService.mWindowManager.continueSurfaceLayout();  
 }  
 postStartActivityUncheckedProcessing(r, err, stack.mStackId, mSourceRecord, mTargetStack);  
 return err;  
}

### 5.4 startActivityUnchecked调用ActivityStackSupervisor类的resumeFocusedStackTopActivityLocked方法

private int startActivityUnchecked(final ActivityRecord r, ActivityRecord sourceRecord,  
 IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,  
 int startFlags, boolean doResume, ActivityOptions options, TaskRecord inTask) {  
  
 setInitialState(r, options, inTask, doResume, startFlags, sourceRecord, voiceSession,  
 voiceInteractor);  
  
 computeLaunchingTaskFlags();  
  
 computeSourceStack();  
  
 mIntent.setFlags(mLaunchFlags);  
  
 mReusedActivity = getReusableIntentActivity();  
  
 final int preferredLaunchStackId =  
 (mOptions != null) ? mOptions.getLaunchStackId() : INVALID\_STACK\_ID;  
  
 if (mReusedActivity != null) {  
 // When the flags NEW\_TASK and CLEAR\_TASK are set, then the task gets reused but  
 // still needs to be a lock task mode violation since the task gets cleared out and  
 // the device would otherwise leave the locked task.  
 if (mSupervisor.isLockTaskModeViolation(mReusedActivity.task,  
 (mLaunchFlags & (*FLAG\_ACTIVITY\_NEW\_TASK* | *FLAG\_ACTIVITY\_CLEAR\_TASK*))  
 == (*FLAG\_ACTIVITY\_NEW\_TASK* | *FLAG\_ACTIVITY\_CLEAR\_TASK*))) {  
 mSupervisor.showLockTaskToast();  
 Slog.e(*TAG*, "startActivityUnchecked: Attempt to violate Lock Task Mode");  
 return START\_RETURN\_LOCK\_TASK\_MODE\_VIOLATION;  
 }  
  
 if (mStartActivity.task == null) {  
 mStartActivity.task = mReusedActivity.task;  
 }  
 if (mReusedActivity.task.intent == null) {  
 // This task was started because of movement of the activity based on affinity...  
 // Now that we are actually launching it, we can assign the base intent.  
 mReusedActivity.task.setIntent(mStartActivity);  
 }  
  
 // This code path leads to delivering a new intent, we want to make sure we schedule it  
 // as the first operation, in case the activity will be resumed as a result of later  
 // operations.  
 if ((mLaunchFlags & *FLAG\_ACTIVITY\_CLEAR\_TOP*) != 0  
 || mLaunchSingleInstance || mLaunchSingleTask) {  
 // In this situation we want to remove all activities from the task up to the one  
 // being started. In most cases this means we are resetting the task to its initial  
 // state.  
 final ActivityRecord top = mReusedActivity.task.performClearTaskForReuseLocked(  
 mStartActivity, mLaunchFlags);  
 if (top != null) {  
 if (top.frontOfTask) {  
 // Activity aliases may mean we use different intents for the top activity,  
 // so make sure the task now has the identity of the new intent.  
 top.task.setIntent(mStartActivity);  
 }  
 ActivityStack.*logStartActivity*(AM\_NEW\_INTENT, mStartActivity, top.task);  
 top.deliverNewIntentLocked(mCallingUid, mStartActivity.intent,  
 mStartActivity.launchedFromPackage);  
 }  
 }  
  
 sendPowerHintForLaunchStartIfNeeded(false /\* forceSend \*/);  
  
 mReusedActivity = setTargetStackAndMoveToFrontIfNeeded(mReusedActivity);  
  
 if ((mStartFlags & START\_FLAG\_ONLY\_IF\_NEEDED) != 0) {  
 // We don't need to start a new activity, and the client said not to do anything  
 // if that is the case, so this is it! And for paranoia, make sure we have  
 // correctly resumed the top activity.  
 resumeTargetStackIfNeeded();  
 return START\_RETURN\_INTENT\_TO\_CALLER;  
 }  
 setTaskFromIntentActivity(mReusedActivity);  
  
 if (!mAddingToTask && mReuseTask == null) {  
 // We didn't do anything... but it was needed (a.k.a., client don't use that  
 // intent!) And for paranoia, make sure we have correctly resumed the top activity.  
 resumeTargetStackIfNeeded();  
 return START\_TASK\_TO\_FRONT;  
 }  
 }  
  
 if (mStartActivity.packageName == null) {  
 if (mStartActivity.resultTo != null && mStartActivity.resultTo.task.stack != null) {  
 mStartActivity.resultTo.task.stack.sendActivityResultLocked(  
 -1, mStartActivity.resultTo, mStartActivity.resultWho,  
 mStartActivity.requestCode, *RESULT\_CANCELED*, null);  
 }  
 ActivityOptions.abort(mOptions);  
 return START\_CLASS\_NOT\_FOUND;  
 }  
  
 // If the activity being launched is the same as the one currently at the top, then  
 // we need to check if it should only be launched once.  
 final ActivityStack topStack = mSupervisor.mFocusedStack;  
 final ActivityRecord top = topStack.topRunningNonDelayedActivityLocked(mNotTop);  
 final boolean dontStart = top != null && mStartActivity.resultTo == null  
 && top.realActivity.equals(mStartActivity.realActivity)  
 && top.userId == mStartActivity.userId  
 && top.app != null && top.app.thread != null  
 && ((mLaunchFlags & *FLAG\_ACTIVITY\_SINGLE\_TOP*) != 0  
 || mLaunchSingleTop || mLaunchSingleTask);  
 if (dontStart) {  
 ActivityStack.*logStartActivity*(AM\_NEW\_INTENT, top, top.task);  
 // For paranoia, make sure we have correctly resumed the top activity.  
 topStack.mLastPausedActivity = null;  
 if (mDoResume) {  
 mSupervisor.resumeFocusedStackTopActivityLocked();  
 }  
 ActivityOptions.abort(mOptions);  
 if ((mStartFlags & START\_FLAG\_ONLY\_IF\_NEEDED) != 0) {  
 // We don't need to start a new activity, and the client said not to do  
 // anything if that is the case, so this is it!  
 return START\_RETURN\_INTENT\_TO\_CALLER;  
 }  
 top.deliverNewIntentLocked(  
 mCallingUid, mStartActivity.intent, mStartActivity.launchedFromPackage);  
  
 // Don't use mStartActivity.task to show the toast. We're not starting a new activity  
 // but reusing 'top'. Fields in mStartActivity may not be fully initialized.  
 mSupervisor.handleNonResizableTaskIfNeeded(  
 top.task, preferredLaunchStackId, topStack.mStackId);  
  
 return START\_DELIVERED\_TO\_TOP;  
 }  
  
 boolean newTask = false;  
 final TaskRecord taskToAffiliate = (mLaunchTaskBehind && mSourceRecord != null)  
 ? mSourceRecord.task : null;  
  
 // Should this be considered a new task?  
 if (mStartActivity.resultTo == null && mInTask == null && !mAddingToTask  
 && (mLaunchFlags & *FLAG\_ACTIVITY\_NEW\_TASK*) != 0) {  
 newTask = true;  
 setTaskFromReuseOrCreateNewTask(taskToAffiliate);  
  
 if (mSupervisor.isLockTaskModeViolation(mStartActivity.task)) {  
 Slog.e(*TAG*, "Attempted Lock Task Mode violation mStartActivity=" + mStartActivity);  
 return START\_RETURN\_LOCK\_TASK\_MODE\_VIOLATION;  
 }  
 if (!mMovedOtherTask) {  
 // If stack id is specified in activity options, usually it means that activity is  
 // launched not from currently focused stack (e.g. from SysUI or from shell) - in  
 // that case we check the target stack.  
 updateTaskReturnToType(mStartActivity.task, mLaunchFlags,  
 preferredLaunchStackId != INVALID\_STACK\_ID ? mTargetStack : topStack);  
 }  
 } else if (mSourceRecord != null) {  
 if (mSupervisor.isLockTaskModeViolation(mSourceRecord.task)) {  
 Slog.e(*TAG*, "Attempted Lock Task Mode violation mStartActivity=" + mStartActivity);  
 return START\_RETURN\_LOCK\_TASK\_MODE\_VIOLATION;  
 }  
  
 final int result = setTaskFromSourceRecord();  
 if (result != START\_SUCCESS) {  
 return result;  
 }  
 } else if (mInTask != null) {  
 // The caller is asking that the new activity be started in an explicit  
 // task it has provided to us.  
 if (mSupervisor.isLockTaskModeViolation(mInTask)) {  
 Slog.e(*TAG*, "Attempted Lock Task Mode violation mStartActivity=" + mStartActivity);  
 return START\_RETURN\_LOCK\_TASK\_MODE\_VIOLATION;  
 }  
  
 final int result = setTaskFromInTask();  
 if (result != START\_SUCCESS) {  
 return result;  
 }  
 } else {  
 // This not being started from an existing activity, and not part of a new task...  
 // just put it in the top task, though these days this case should never happen.  
 setTaskToCurrentTopOrCreateNewTask();  
 }  
  
 mService.grantUriPermissionFromIntentLocked(mCallingUid, mStartActivity.packageName,  
 mIntent, mStartActivity.getUriPermissionsLocked(), mStartActivity.userId);  
  
 if (mSourceRecord != null && mSourceRecord.isRecentsActivity()) {  
 mStartActivity.task.setTaskToReturnTo(RECENTS\_ACTIVITY\_TYPE);  
 }  
 if (newTask) {  
 EventLog.*writeEvent*(  
 EventLogTags.AM\_CREATE\_TASK, mStartActivity.userId, mStartActivity.task.taskId);  
 }  
 ActivityStack.*logStartActivity*(  
 EventLogTags.AM\_CREATE\_ACTIVITY, mStartActivity, mStartActivity.task);  
 mTargetStack.mLastPausedActivity = null;  
  
 sendPowerHintForLaunchStartIfNeeded(false /\* forceSend \*/);  
  
 mTargetStack.startActivityLocked(mStartActivity, newTask, mKeepCurTransition, mOptions);  
 if (mDoResume) {  
 if (!mLaunchTaskBehind) {  
 // *TODO(b/26381750): Remove this code after verification that all the decision* // points above moved targetStack to the front which will also set the focus  
 // activity.  
 mService.setFocusedActivityLocked(mStartActivity, "startedActivity");  
 }  
 final ActivityRecord topTaskActivity = mStartActivity.task.topRunningActivityLocked();  
 if (!mTargetStack.isFocusable()  
 || (topTaskActivity != null && topTaskActivity.mTaskOverlay  
 && mStartActivity != topTaskActivity)) {  
 // If the activity is not focusable, we can't resume it, but still would like to  
 // make sure it becomes visible as it starts (this will also trigger entry  
 // animation). An example of this are PIP activities.  
 // Also, we don't want to resume activities in a task that currently has an overlay  
 // as the starting activity just needs to be in the visible paused state until the  
 // over is removed.  
 mTargetStack.ensureActivitiesVisibleLocked(null, 0, !PRESERVE\_WINDOWS);  
 // Go ahead and tell window manager to execute app transition for this activity  
 // since the app transition will not be triggered through the resume channel.  
 mWindowManager.executeAppTransition();  
 } else {  
 mSupervisor.**resumeFocusedStackTopActivityLocked**(mTargetStack, mStartActivity,  
 mOptions);  
 }  
 } else {  
 mTargetStack.addRecentActivityLocked(mStartActivity);  
 }  
 mSupervisor.updateUserStackLocked(mStartActivity.userId, mTargetStack);  
  
 mSupervisor.handleNonResizableTaskIfNeeded(  
 mStartActivity.task, preferredLaunchStackId, mTargetStack.mStackId);  
  
 return START\_SUCCESS;  
}

### 5.5ActivityStackSupervisor 类的resumeFocusedStackTopActivityLocked方法调用 ActivityStack类的resumeTopActivityUncheckedLocked

boolean resumeFocusedStackTopActivityLocked(  
 ActivityStack targetStack, ActivityRecord target, ActivityOptions targetOptions) {  
 if (targetStack != null && isFocusedStack(targetStack)) {  
 return targetStack.**resumeTopActivityUncheckedLocked**(target, targetOptions);  
 }  
 final ActivityRecord r = mFocusedStack.topRunningActivityLocked();  
 if (r == null || r.state != RESUMED) {  
 mFocusedStack.resumeTopActivityUncheckedLocked(null, null);  
 }  
 return false;  
}

### 5.6 ActivityStack类的resumeTopActivityUncheckedLocked方法调用resumeTopActivityInnerLocked方法, resumeTopActivityInnerLocked调用了ActivityStackSupervisor的startSpecificActivityLocked方法。

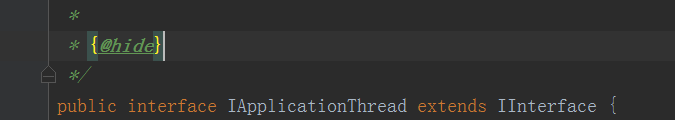
boolean resumeTopActivityUncheckedLocked(ActivityRecord prev, ActivityOptions options) {  
 if (mStackSupervisor.inResumeTopActivity) {  
 // Don't even start recursing.  
 return false;  
 }  
  
 boolean result = false;  
 try {  
 // Protect against recursion.  
 mStackSupervisor.inResumeTopActivity = true;  
 if (mService.mLockScreenShown == ActivityManagerService.*LOCK\_SCREEN\_LEAVING*) {  
 mService.mLockScreenShown = ActivityManagerService.*LOCK\_SCREEN\_HIDDEN*;  
 mService.updateSleepIfNeededLocked();  
 }  
 result = resumeTopActivityInnerLocked(prev, options);  
 } finally {  
 mStackSupervisor.inResumeTopActivity = false;  
 }  
 return result;  
}  
  
private boolean resumeTopActivityInnerLocked(ActivityRecord prev, ActivityOptions options) {  
 if (DEBUG\_LOCKSCREEN) mService.logLockScreen("");  
  
 if (!mService.mBooting && !mService.mBooted) {  
 // Not ready yet!  
 return false;  
 }  
  
 ActivityRecord parent = mActivityContainer.mParentActivity;  
 if ((parent != null && parent.state != ActivityState.*RESUMED*) ||  
 !mActivityContainer.isAttachedLocked()) {  
 // Do not resume this stack if its parent is not resumed.  
 // *TODO: If in a loop, make sure that parent stack resumeTopActivity is called 1st.* return false;  
 }  
  
 mStackSupervisor.cancelInitializingActivities();  
  
 // Find the first activity that is not finishing.  
 final ActivityRecord next = topRunningActivityLocked();  
  
 // Remember how we'll process this pause/resume situation, and ensure  
 // that the state is reset however we wind up proceeding.  
 final boolean userLeaving = mStackSupervisor.mUserLeaving;  
 mStackSupervisor.mUserLeaving = false;  
  
 final TaskRecord prevTask = prev != null ? prev.task : null;  
 if (next == null) {  
 // There are no more activities!  
 final String reason = "noMoreActivities";  
 final int returnTaskType = prevTask == null || !prevTask.isOverHomeStack()  
 ? HOME\_ACTIVITY\_TYPE : prevTask.getTaskToReturnTo();  
 if (!mFullscreen && adjustFocusToNextFocusableStackLocked(returnTaskType, reason)) {  
 // Try to move focus to the next visible stack with a running activity if this  
 // stack is not covering the entire screen.  
 return mStackSupervisor.resumeFocusedStackTopActivityLocked(  
 mStackSupervisor.getFocusedStack(), prev, null);  
 }  
  
 // Let's just start up the Launcher...  
 ActivityOptions.abort(options);  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "resumeTopActivityLocked: No more activities go home");  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 // Only resume home if on home display  
 return isOnHomeDisplay() &&  
 mStackSupervisor.resumeHomeStackTask(returnTaskType, prev, reason);  
 }  
  
 next.delayedResume = false;  
  
 // If the top activity is the resumed one, nothing to do.  
 if (mResumedActivity == next && next.state == ActivityState.*RESUMED* &&  
 mStackSupervisor.allResumedActivitiesComplete()) {  
 // Make sure we have executed any pending transitions, since there  
 // should be nothing left to do at this point.  
 mWindowManager.executeAppTransition();  
 mNoAnimActivities.clear();  
 ActivityOptions.abort(options);  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "resumeTopActivityLocked: Top activity resumed " + next);  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return false;  
 }  
  
 final TaskRecord nextTask = next.task;  
 if (prevTask != null && prevTask.stack == this &&  
 prevTask.isOverHomeStack() && prev.finishing && prev.frontOfTask) {  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 if (prevTask == nextTask) {  
 prevTask.setFrontOfTask();  
 } else if (prevTask != topTask()) {  
 // This task is going away but it was supposed to return to the home stack.  
 // Now the task above it has to return to the home task instead.  
 final int taskNdx = mTaskHistory.indexOf(prevTask) + 1;  
 mTaskHistory.get(taskNdx).setTaskToReturnTo(HOME\_ACTIVITY\_TYPE);  
 } else if (!isOnHomeDisplay()) {  
 return false;  
 } else if (!isHomeStack()){  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "resumeTopActivityLocked: Launching home next");  
 final int returnTaskType = prevTask == null || !prevTask.isOverHomeStack() ?  
 HOME\_ACTIVITY\_TYPE : prevTask.getTaskToReturnTo();  
 return isOnHomeDisplay() &&  
 mStackSupervisor.resumeHomeStackTask(returnTaskType, prev, "prevFinished");  
 }  
 }  
  
 // If we are sleeping, and there is no resumed activity, and the top  
 // activity is paused, well that is the state we want.  
 if (mService.isSleepingOrShuttingDownLocked()  
 && mLastPausedActivity == next  
 && mStackSupervisor.allPausedActivitiesComplete()) {  
 // Make sure we have executed any pending transitions, since there  
 // should be nothing left to do at this point.  
 mWindowManager.executeAppTransition();  
 mNoAnimActivities.clear();  
 ActivityOptions.abort(options);  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "resumeTopActivityLocked: Going to sleep and all paused");  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return false;  
 }  
  
 // Make sure that the user who owns this activity is started. If not,  
 // we will just leave it as is because someone should be bringing  
 // another user's activities to the top of the stack.  
 if (!mService.mUserController.hasStartedUserState(next.userId)) {  
 Slog.w(*TAG*, "Skipping resume of top activity " + next  
 + ": user " + next.userId + " is stopped");  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return false;  
 }  
  
 // The activity may be waiting for stop, but that is no longer  
 // appropriate for it.  
 mStackSupervisor.mStoppingActivities.remove(next);  
 mStackSupervisor.mGoingToSleepActivities.remove(next);  
 next.sleeping = false;  
 mStackSupervisor.mWaitingVisibleActivities.remove(next);  
  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*, "Resuming " + next);  
  
 // If we are currently pausing an activity, then don't do anything until that is done.  
 if (!mStackSupervisor.allPausedActivitiesComplete()) {  
 if (DEBUG\_SWITCH || DEBUG\_PAUSE || DEBUG\_STATES) Slog.v(*TAG\_PAUSE*,  
 "resumeTopActivityLocked: Skip resume: some activity pausing.");  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return false;  
 }  
  
 mStackSupervisor.setLaunchSource(next.info.applicationInfo.uid);  
  
 // We need to start pausing the current activity so the top one can be resumed...  
 final boolean dontWaitForPause = (next.info.flags & *FLAG\_RESUME\_WHILE\_PAUSING*) != 0;  
 boolean pausing = mStackSupervisor.pauseBackStacks(userLeaving, next, dontWaitForPause);  
 if (mResumedActivity != null) {  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "resumeTopActivityLocked: Pausing " + mResumedActivity);  
 pausing |= startPausingLocked(userLeaving, false, next, dontWaitForPause);  
 }  
 if (pausing) {  
 if (DEBUG\_SWITCH || DEBUG\_STATES) Slog.v(*TAG\_STATES*,  
 "resumeTopActivityLocked: Skip resume: need to start pausing");  
 // At this point we want to put the upcoming activity's process  
 // at the top of the LRU list, since we know we will be needing it  
 // very soon and it would be a waste to let it get killed if it  
 // happens to be sitting towards the end.  
 if (next.app != null && next.app.thread != null) {  
 mService.updateLruProcessLocked(next.app, true, null);  
 }  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return true;  
 } else if (mResumedActivity == next && next.state == ActivityState.*RESUMED* &&  
 mStackSupervisor.allResumedActivitiesComplete()) {  
 // It is possible for the activity to be resumed when we paused back stacks above if the  
 // next activity doesn't have to wait for pause to complete.  
 // So, nothing else to-do except:  
 // Make sure we have executed any pending transitions, since there  
 // should be nothing left to do at this point.  
 mWindowManager.executeAppTransition();  
 mNoAnimActivities.clear();  
 ActivityOptions.abort(options);  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "resumeTopActivityLocked: Top activity resumed (dontWaitForPause) " + next);  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return true;  
 }  
  
 // If the most recent activity was noHistory but was only stopped rather  
 // than stopped+finished because the device went to sleep, we need to make  
 // sure to finish it as we're making a new activity topmost.  
 if (mService.isSleepingLocked() && mLastNoHistoryActivity != null &&  
 !mLastNoHistoryActivity.finishing) {  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*,  
 "no-history finish of " + mLastNoHistoryActivity + " on new resume");  
 requestFinishActivityLocked(mLastNoHistoryActivity.appToken, Activity.*RESULT\_CANCELED*,  
 null, "resume-no-history", false);  
 mLastNoHistoryActivity = null;  
 }  
  
 if (prev != null && prev != next) {  
 if (!mStackSupervisor.mWaitingVisibleActivities.contains(prev)  
 && next != null && !next.nowVisible) {  
 mStackSupervisor.mWaitingVisibleActivities.add(prev);  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*,  
 "Resuming top, waiting visible to hide: " + prev);  
 } else {  
 // The next activity is already visible, so hide the previous  
 // activity's windows right now so we can show the new one ASAP.  
 // We only do this if the previous is finishing, which should mean  
 // it is on top of the one being resumed so hiding it quickly  
 // is good. Otherwise, we want to do the normal route of allowing  
 // the resumed activity to be shown so we can decide if the  
 // previous should actually be hidden depending on whether the  
 // new one is found to be full-screen or not.  
 if (prev.finishing) {  
 mWindowManager.setAppVisibility(prev.appToken, false);  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*,  
 "Not waiting for visible to hide: " + prev + ", waitingVisible="  
 + mStackSupervisor.mWaitingVisibleActivities.contains(prev)  
 + ", nowVisible=" + next.nowVisible);  
 } else {  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*,  
 "Previous already visible but still waiting to hide: " + prev  
 + ", waitingVisible="  
 + mStackSupervisor.mWaitingVisibleActivities.contains(prev)  
 + ", nowVisible=" + next.nowVisible);  
 }  
 }  
 }  
  
 // Launching this app's activity, make sure the app is no longer  
 // considered stopped.  
 try {  
 AppGlobals.getPackageManager().setPackageStoppedState(  
 next.packageName, false, next.userId); /\* *TODO: Verify if correct userid* \*/  
 } catch (RemoteException e1) {  
 } catch (IllegalArgumentException e) {  
 Slog.w(*TAG*, "Failed trying to unstop package "  
 + next.packageName + ": " + e);  
 }  
  
 // We are starting up the next activity, so tell the window manager  
 // that the previous one will be hidden soon. This way it can know  
 // to ignore it when computing the desired screen orientation.  
 boolean anim = true;  
 if (prev != null) {  
 if (prev.finishing) {  
 if (DEBUG\_TRANSITION) Slog.v(*TAG\_TRANSITION*,  
 "Prepare close transition: prev=" + prev);  
 if (mNoAnimActivities.contains(prev)) {  
 anim = false;  
 mWindowManager.prepareAppTransition(TRANSIT\_NONE, false);  
 } else {  
 mWindowManager.prepareAppTransition(prev.task == next.task  
 ? TRANSIT\_ACTIVITY\_CLOSE  
 : TRANSIT\_TASK\_CLOSE, false);  
 }  
 mWindowManager.setAppVisibility(prev.appToken, false);  
 } else {  
 if (DEBUG\_TRANSITION) Slog.v(*TAG\_TRANSITION*,  
 "Prepare open transition: prev=" + prev);  
 if (mNoAnimActivities.contains(next)) {  
 anim = false;  
 mWindowManager.prepareAppTransition(TRANSIT\_NONE, false);  
 } else {  
 mWindowManager.prepareAppTransition(prev.task == next.task  
 ? TRANSIT\_ACTIVITY\_OPEN  
 : next.mLaunchTaskBehind  
 ? TRANSIT\_TASK\_OPEN\_BEHIND  
 : TRANSIT\_TASK\_OPEN, false);  
 }  
 }  
 } else {  
 if (DEBUG\_TRANSITION) Slog.v(*TAG\_TRANSITION*, "Prepare open transition: no previous");  
 if (mNoAnimActivities.contains(next)) {  
 anim = false;  
 mWindowManager.prepareAppTransition(TRANSIT\_NONE, false);  
 } else {  
 mWindowManager.prepareAppTransition(TRANSIT\_ACTIVITY\_OPEN, false);  
 }  
 }  
  
 Bundle resumeAnimOptions = null;  
 if (anim) {  
 ActivityOptions opts = next.getOptionsForTargetActivityLocked();  
 if (opts != null) {  
 resumeAnimOptions = opts.toBundle();  
 }  
 next.applyOptionsLocked();  
 } else {  
 next.clearOptionsLocked();  
 }  
  
 ActivityStack lastStack = mStackSupervisor.getLastStack();  
 if (next.app != null && next.app.thread != null) {  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*, "Resume running: " + next  
 + " stopped=" + next.stopped + " visible=" + next.visible);  
  
 // If the previous activity is translucent, force a visibility update of  
 // the next activity, so that it's added to WM's opening app list, and  
 // transition animation can be set up properly.  
 // For example, pressing Home button with a translucent activity in focus.  
 // Launcher is already visible in this case. If we don't add it to opening  
 // apps, maybeUpdateTransitToWallpaper() will fail to identify this as a  
 // TRANSIT\_WALLPAPER\_OPEN animation, and run some funny animation.  
 final boolean lastActivityTranslucent = lastStack != null  
 && (!lastStack.mFullscreen  
 || (lastStack.mLastPausedActivity != null  
 && !lastStack.mLastPausedActivity.fullscreen));  
  
 // This activity is now becoming visible.  
 if (!next.visible || next.stopped || lastActivityTranslucent) {  
 mWindowManager.setAppVisibility(next.appToken, true);  
 }  
  
 // schedule launch ticks to collect information about slow apps.  
 next.startLaunchTickingLocked();  
  
 ActivityRecord lastResumedActivity =  
 lastStack == null ? null :lastStack.mResumedActivity;  
 ActivityState lastState = next.state;  
  
 mService.updateCpuStats();  
  
 if (DEBUG\_STATES) Slog.v(*TAG\_STATES*, "Moving to RESUMED: " + next + " (in existing)");  
 next.state = ActivityState.*RESUMED*;  
 mResumedActivity = next;  
 next.task.touchActiveTime();  
 mRecentTasks.addLocked(next.task);  
 mService.updateLruProcessLocked(next.app, true, null);  
 updateLRUListLocked(next);  
 mService.updateOomAdjLocked();  
  
 // Have the window manager re-evaluate the orientation of  
 // the screen based on the new activity order.  
 boolean notUpdated = true;  
 if (mStackSupervisor.isFocusedStack(this)) {  
 Configuration config = mWindowManager.updateOrientationFromAppTokens(  
 mService.mConfiguration,  
 next.mayFreezeScreenLocked(next.app) ? next.appToken : null);  
 if (config != null) {  
 next.frozenBeforeDestroy = true;  
 }  
 notUpdated = !mService.updateConfigurationLocked(config, next, false);  
 }  
  
 if (notUpdated) {  
 // The configuration update wasn't able to keep the existing  
 // instance of the activity, and instead started a new one.  
 // We should be all done, but let's just make sure our activity  
 // is still at the top and schedule another run if something  
 // weird happened.  
 ActivityRecord nextNext = topRunningActivityLocked();  
 if (DEBUG\_SWITCH || DEBUG\_STATES) Slog.i(*TAG\_STATES*,  
 "Activity config changed during resume: " + next  
 + ", new next: " + nextNext);  
 if (nextNext != next) {  
 // Do over!  
 mStackSupervisor.scheduleResumeTopActivities();  
 }  
 if (mStackSupervisor.reportResumedActivityLocked(next)) {  
 mNoAnimActivities.clear();  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return true;  
 }  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return false;  
 }  
  
 try {  
 // Deliver all pending results.  
 ArrayList<ResultInfo> a = next.results;  
 if (a != null) {  
 final int N = a.size();  
 if (!next.finishing && N > 0) {  
 if (DEBUG\_RESULTS) Slog.v(*TAG\_RESULTS*,  
 "Delivering results to " + next + ": " + a);  
 next.app.thread.scheduleSendResult(next.appToken, a);  
 }  
 }  
  
 boolean allowSavedSurface = true;  
 if (next.newIntents != null) {  
 // Restrict saved surface to launcher start, or there is no intent at all  
 // (eg. task being brought to front). If the intent is something else,  
 // likely the app is going to show some specific page or view, instead of  
 // what's left last time.  
 for (int i = next.newIntents.size() - 1; i >= 0; i--) {  
 final Intent intent = next.newIntents.get(i);  
 if (intent != null && !ActivityRecord.*isMainIntent*(intent)) {  
 allowSavedSurface = false;  
 break;  
 }  
 }  
 next.app.thread.scheduleNewIntent(  
 next.newIntents, next.appToken, false /\* andPause \*/);  
 }  
  
 // Well the app will no longer be stopped.  
 // Clear app token stopped state in window manager if needed.  
 mWindowManager.notifyAppResumed(next.appToken, next.stopped, allowSavedSurface);  
  
 EventLog.*writeEvent*(EventLogTags.AM\_RESUME\_ACTIVITY, next.userId,  
 System.*identityHashCode*(next), next.task.taskId, next.shortComponentName);  
  
 next.sleeping = false;  
 mService.showUnsupportedZoomDialogIfNeededLocked(next);  
 mService.showAskCompatModeDialogLocked(next);  
 next.app.pendingUiClean = true;  
 next.app.forceProcessStateUpTo(mService.mTopProcessState);  
 next.clearOptionsLocked();  
 next.app.thread.scheduleResumeActivity(next.appToken, next.app.repProcState,  
 mService.isNextTransitionForward(), resumeAnimOptions);  
  
 mStackSupervisor.checkReadyForSleepLocked();  
  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*, "resumeTopActivityLocked: Resumed " + next);  
 } catch (Exception e) {  
 // Whoops, need to restart this activity!  
 if (DEBUG\_STATES) Slog.v(*TAG\_STATES*, "Resume failed; resetting state to "  
 + lastState + ": " + next);  
 next.state = lastState;  
 if (lastStack != null) {  
 lastStack.mResumedActivity = lastResumedActivity;  
 }  
 Slog.i(*TAG*, "Restarting because process died: " + next);  
 if (!next.hasBeenLaunched) {  
 next.hasBeenLaunched = true;  
 } else if (*SHOW\_APP\_STARTING\_PREVIEW* && lastStack != null &&  
 mStackSupervisor.isFrontStack(lastStack)) {  
 next.showStartingWindow(null, true);  
 }  
 mStackSupervisor.startSpecificActivityLocked(next, true, false);  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return true;  
 }  
  
 // From this point on, if something goes wrong there is no way  
 // to recover the activity.  
 try {  
 completeResumeLocked(next);  
 } catch (Exception e) {  
 // If any exception gets thrown, toss away this  
 // activity and try the next one.  
 Slog.w(*TAG*, "Exception thrown during resume of " + next, e);  
 requestFinishActivityLocked(next.appToken, Activity.*RESULT\_CANCELED*, null,  
 "resume-exception", true);  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return true;  
 }  
 } else {  
 // Whoops, need to restart this activity!  
 if (!next.hasBeenLaunched) {  
 next.hasBeenLaunched = true;  
 } else {  
 if (*SHOW\_APP\_STARTING\_PREVIEW*) {  
 next.showStartingWindow(null, true);  
 }  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*, "Restarting: " + next);  
 }  
 if (DEBUG\_STATES) Slog.d(*TAG\_STATES*, "resumeTopActivityLocked: Restarting " + next);  
 mStackSupervisor.**startSpecificActivityLocked**(next, true, true);  
 }  
  
 if (DEBUG\_STACK) mStackSupervisor.validateTopActivitiesLocked();  
 return true;  
}

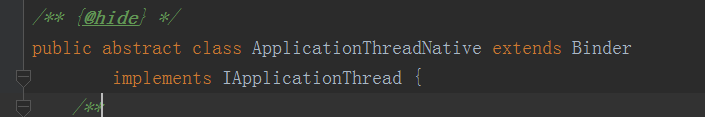
### 5.7 ActivityStackSupervisor类的startSpecificActivityLocked方法调用realStartActivityLocked方法

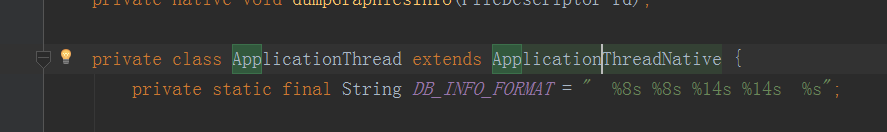
void startSpecificActivityLocked(ActivityRecord r,  
 boolean andResume, boolean checkConfig) {  
 // Is this activity's application already running?  
 ProcessRecord app = mService.getProcessRecordLocked(r.processName,  
 r.info.applicationInfo.uid, true);  
  
 r.task.stack.setLaunchTime(r);  
  
 if (app != null && app.thread != null) {  
 try {  
 if ((r.info.flags&ActivityInfo.*FLAG\_MULTIPROCESS*) == 0  
 || !"android".equals(r.info.packageName)) {  
 // Don't add this if it is a platform component that is marked  
 // to run in multiple processes, because this is actually  
 // part of the framework so doesn't make sense to track as a  
 // separate apk in the process.  
 app.addPackage(r.info.packageName, r.info.applicationInfo.versionCode,  
 mService.mProcessStats);  
 }  
 **realStartActivityLocked**(r, app, andResume, checkConfig);  
 return;  
 } catch (RemoteException e) {  
 Slog.w(*TAG*, "Exception when starting activity "  
 + r.intent.getComponent().flattenToShortString(), e);  
 }  
  
 // If a dead object exception was thrown -- fall through to  
 // restart the application.  
 }  
  
 mService.startProcessLocked(r.processName, r.info.applicationInfo, true, 0,  
 "activity", r.intent.getComponent(), false, false, true);  
}

final boolean realStartActivityLocked(ActivityRecord r, ProcessRecord app,  
 boolean andResume, boolean checkConfig) throws RemoteException {  
  
 if (!allPausedActivitiesComplete()) {  
 // While there are activities pausing we skipping starting any new activities until  
 // pauses are complete. NOTE: that we also do this for activities that are starting in  
 // the paused state because they will first be resumed then paused on the client side.  
 if (DEBUG\_SWITCH || DEBUG\_PAUSE || DEBUG\_STATES) Slog.v(*TAG\_PAUSE*,  
 "realStartActivityLocked: Skipping start of r=" + r  
 + " some activities pausing...");  
 return false;  
 }  
  
 if (andResume) {  
 r.startFreezingScreenLocked(app, 0);  
 mWindowManager.setAppVisibility(r.appToken, true);  
  
 // schedule launch ticks to collect information about slow apps.  
 r.startLaunchTickingLocked();  
 }  
  
 // Have the window manager re-evaluate the orientation of  
 // the screen based on the new activity order. Note that  
 // as a result of this, it can call back into the activity  
 // manager with a new orientation. We don't care about that,  
 // because the activity is not currently running so we are  
 // just restarting it anyway.  
 if (checkConfig) {  
 Configuration config = mWindowManager.updateOrientationFromAppTokens(  
 mService.mConfiguration,  
 r.mayFreezeScreenLocked(app) ? r.appToken : null);  
 // Deferring resume here because we're going to launch new activity shortly.  
 // We don't want to perform a redundant launch of the same record while ensuring  
 // configurations and trying to resume top activity of focused stack.  
 mService.updateConfigurationLocked(config, r, false, true /\* deferResume \*/);  
 }  
  
 r.app = app;  
 app.waitingToKill = null;  
 r.launchCount++;  
 r.lastLaunchTime = SystemClock.*uptimeMillis*();  
  
 if (DEBUG\_ALL) Slog.v(*TAG*, "Launching: " + r);  
  
 int idx = app.activities.indexOf(r);  
 if (idx < 0) {  
 app.activities.add(r);  
 }  
 mService.updateLruProcessLocked(app, true, null);  
 mService.updateOomAdjLocked();  
  
 final TaskRecord task = r.task;  
 if (task.mLockTaskAuth == LOCK\_TASK\_AUTH\_LAUNCHABLE ||  
 task.mLockTaskAuth == LOCK\_TASK\_AUTH\_LAUNCHABLE\_PRIV) {  
 setLockTaskModeLocked(task, *LOCK\_TASK\_MODE\_LOCKED*, "mLockTaskAuth==LAUNCHABLE", false);  
 }  
  
 final ActivityStack stack = task.stack;  
 try {  
 if (app.thread == null) {  
 throw new RemoteException();  
 }  
 List<ResultInfo> results = null;  
 List<ReferrerIntent> newIntents = null;  
 if (andResume) {  
 results = r.results;  
 newIntents = r.newIntents;  
 }  
 if (DEBUG\_SWITCH) Slog.v(*TAG\_SWITCH*,  
 "Launching: " + r + " icicle=" + r.icicle + " with results=" + results  
 + " newIntents=" + newIntents + " andResume=" + andResume);  
 if (andResume) {  
 EventLog.*writeEvent*(EventLogTags.AM\_RESTART\_ACTIVITY,  
 r.userId, System.*identityHashCode*(r),  
 task.taskId, r.shortComponentName);  
 }  
 if (r.isHomeActivity()) {  
 // Home process is the root process of the task.  
 mService.mHomeProcess = task.mActivities.get(0).app;  
 }  
 mService.notifyPackageUse(r.intent.getComponent().getPackageName(),  
 PackageManager.NOTIFY\_PACKAGE\_USE\_ACTIVITY);  
 r.sleeping = false;  
 r.forceNewConfig = false;  
 mService.showUnsupportedZoomDialogIfNeededLocked(r);  
 mService.showAskCompatModeDialogLocked(r);  
 r.compat = mService.compatibilityInfoForPackageLocked(r.info.applicationInfo);  
 ProfilerInfo profilerInfo = null;  
 if (mService.mProfileApp != null && mService.mProfileApp.equals(app.processName)) {  
 if (mService.mProfileProc == null || mService.mProfileProc == app) {  
 mService.mProfileProc = app;  
 final String profileFile = mService.mProfileFile;  
 if (profileFile != null) {  
 ParcelFileDescriptor profileFd = mService.mProfileFd;  
 if (profileFd != null) {  
 try {  
 profileFd = profileFd.dup();  
 } catch (IOException e) {  
 if (profileFd != null) {  
 try {  
 profileFd.close();  
 } catch (IOException o) {  
 }  
 profileFd = null;  
 }  
 }  
 }  
  
 profilerInfo = new ProfilerInfo(profileFile, profileFd,  
 mService.mSamplingInterval, mService.mAutoStopProfiler);  
 }  
 }  
 }  
  
 if (andResume) {  
 app.hasShownUi = true;  
 app.pendingUiClean = true;  
 }  
 app.forceProcessStateUpTo(mService.mTopProcessState);  
  **app.thread.scheduleLaunchActivity**(new Intent(r.intent), r.appToken,  
 System.*identityHashCode*(r), r.info, new Configuration(mService.mConfiguration),  
 new Configuration(task.mOverrideConfig), r.compat, r.launchedFromPackage,  
 task.voiceInteractor, app.repProcState, r.icicle, r.persistentState, results,  
 newIntents, !andResume, mService.isNextTransitionForward(), profilerInfo);  
  
 if ((app.info.privateFlags&ApplicationInfo.PRIVATE\_FLAG\_CANT\_SAVE\_STATE) != 0) {  
 // This may be a heavy-weight process! Note that the package  
 // manager will ensure that only activity can run in the main  
 // process of the .apk, which is the only thing that will be  
 // considered heavy-weight.  
 if (app.processName.equals(app.info.packageName)) {  
 if (mService.mHeavyWeightProcess != null  
 && mService.mHeavyWeightProcess != app) {  
 Slog.w(*TAG*, "Starting new heavy weight process " + app  
 + " when already running "  
 + mService.mHeavyWeightProcess);  
 }  
 mService.mHeavyWeightProcess = app;  
 Message msg = mService.mHandler.obtainMessage(  
 ActivityManagerService.*POST\_HEAVY\_NOTIFICATION\_MSG*);  
 msg.obj = r;  
 mService.mHandler.sendMessage(msg);  
 }  
 }  
  
 } catch (RemoteException e) {  
 if (r.launchFailed) {  
 // This is the second time we failed -- finish activity  
 // and give up.  
 Slog.e(*TAG*, "Second failure launching "  
 + r.intent.getComponent().flattenToShortString()  
 + ", giving up", e);  
 mService.appDiedLocked(app);  
 stack.requestFinishActivityLocked(r.appToken, Activity.*RESULT\_CANCELED*, null,  
 "2nd-crash", false);  
 return false;  
 }  
  
 // This is the first time we failed -- restart process and  
 // retry.  
 app.activities.remove(r);  
 throw e;  
 }  
  
 r.launchFailed = false;  
 if (stack.updateLRUListLocked(r)) {  
 Slog.w(*TAG*, "Activity " + r + " being launched, but already in LRU list");  
 }  
  
 if (andResume) {  
 // As part of the process of launching, ActivityThread also performs  
 // a resume.  
 stack.minimalResumeActivityLocked(r);  
 } else {  
 // This activity is not starting in the resumed state... which should look like we asked  
 // it to pause+stop (but remain visible), and it has done so and reported back the  
 // current icicle and other state.  
 if (DEBUG\_STATES) Slog.v(*TAG\_STATES*,  
 "Moving to PAUSED: " + r + " (starting in paused state)");  
 r.state = PAUSED;  
 }  
  
 // Launch the new version setup screen if needed. We do this -after-  
 // launching the initial activity (that is, home), so that it can have  
 // a chance to initialize itself while in the background, making the  
 // switch back to it faster and look better.  
 if (isFocusedStack(stack)) {  
 mService.startSetupActivityLocked();  
 }  
  
 // Update any services we are bound to that might care about whether  
 // their client may have activities.  
 if (r.app != null) {  
 mService.mServices.updateServiceConnectionActivitiesLocked(r.app);  
 }  
  
 return true;  
}

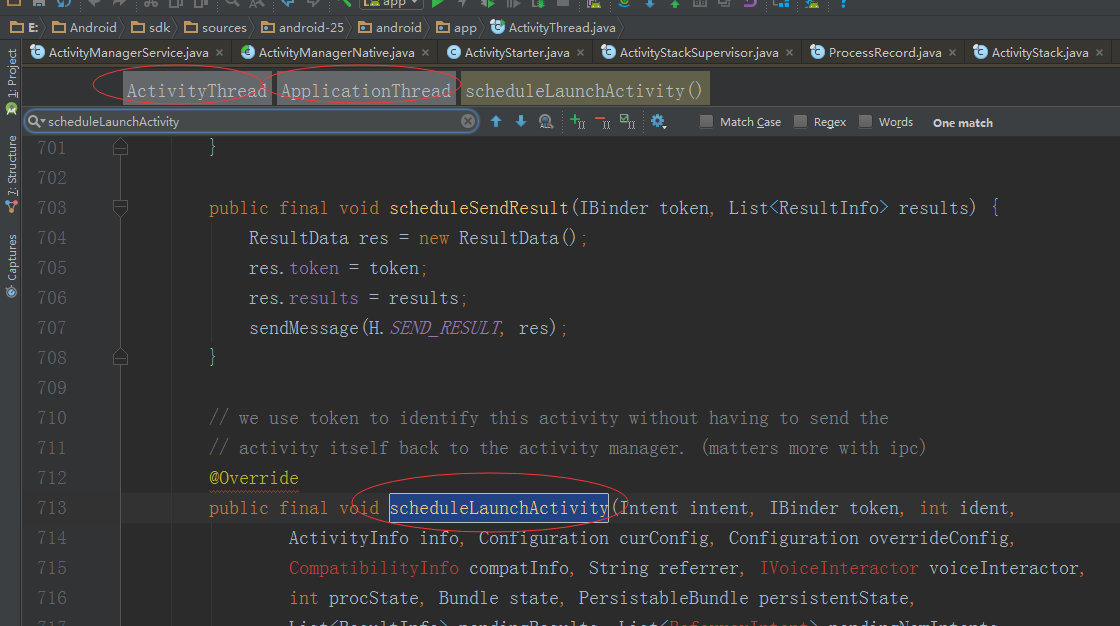
### 5.8构成IPC







运行在系统进程中



### 5.9启动activity

ActivityThread 内部类 ApplicationThread的scheduleLaunchActivity方法启动

@Override  
public final void scheduleLaunchActivity(Intent intent, IBinder token, int ident,  
 ActivityInfo info, Configuration curConfig, Configuration overrideConfig,  
 CompatibilityInfo compatInfo, String referrer, IVoiceInteractor voiceInteractor,  
 int procState, Bundle state, PersistableBundle persistentState,  
 List<ResultInfo> pendingResults, List<ReferrerIntent> pendingNewIntents,  
 boolean notResumed, boolean isForward, ProfilerInfo profilerInfo) {  
  
 updateProcessState(procState, false);  
  
 ActivityClientRecord r = new ActivityClientRecord();  
  
 r.token = token;  
 r.ident = ident;  
 r.intent = intent;  
 r.referrer = referrer;  
 r.voiceInteractor = voiceInteractor;  
 r.activityInfo = info;  
 r.compatInfo = compatInfo;  
 r.state = state;  
 r.persistentState = persistentState;  
  
 r.pendingResults = pendingResults;  
 r.pendingIntents = pendingNewIntents;  
  
 r.startsNotResumed = notResumed;  
 r.isForward = isForward;  
  
 r.profilerInfo = profilerInfo;  
  
 r.overrideConfig = overrideConfig;  
 updatePendingConfiguration(curConfig);  
  
 sendMessage(H.*LAUNCH\_ACTIVITY*, r);  
}

### H Handler处理消息（ActivityThread类中）

private void handleLaunchActivity(ActivityClientRecord r, Intent customIntent, String reason) {  
 // 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  
 WindowManagerGlobal.initialize();  
  
 Activity a = **performLaunchActivity**(r, customIntent);  
  
 if (a != null) {  
 r.createdConfig = new Configuration(mConfiguration);  
 reportSizeConfigurations(r);  
 Bundle oldState = r.state;  
 handleResumeActivity(r.token, false, r.isForward,  
 !r.activity.mFinished && !r.startsNotResumed, r.lastProcessedSeq, reason);  
  
 if (!r.activity.mFinished && r.startsNotResumed) {  
 // The activity manager actually wants this one to start out paused, because it  
 // needs to be visible but isn't in the foreground. We accomplish this by going  
 // through the normal startup (because activities expect to go through onResume()  
 // the first time they run, before their window is displayed), and then pausing it.  
 // However, in this case we do -not- need to do the full pause cycle (of freezing  
 // and such) because the activity manager assumes it can just retain the current  
 // state it has.  
 performPauseActivityIfNeeded(r, reason);  
  
 // We need to keep around the original state, in case we need to be created again.  
 // But we only do this for pre-Honeycomb apps, which always save their state when  
 // pausing, so we can not have them save their state when restarting from a paused  
 // state. For HC and later, we want to (and can) let the state be saved as the  
 // normal part of stopping the activity.  
 if (r.isPreHoneycomb()) {  
 r.state = oldState;  
 }  
 }  
 } else {  
 // If there was an error, for any reason, tell the activity manager to stop us.  
 try {  
 ActivityManagerNative.*getDefault*()  
 .finishActivity(r.token, Activity.*RESULT\_CANCELED*, null,  
 Activity.*DONT\_FINISH\_TASK\_WITH\_ACTIVITY*);  
 } catch (RemoteException ex) {  
 throw ex.rethrowFromSystemServer();  
 }  
 }  
}

private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {  
 // System.out.println("##### [" + System.currentTimeMillis() + "] ActivityThread.performLaunchActivity(" + r + ")");  
  
 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);  
 }  
  
 Activity activity = null;  
 try {

//1、实例化activity  
 java.lang.ClassLoader cl = r.packageInfo.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) {  
 if (!mInstrumentation.onException(activity, e)) {  
 throw new RuntimeException(  
 "Unable to instantiate activity " + component  
 + ": " + e.toString(), e);  
 }  
 }  
  
 try {

//2创建appliaction  
 Application app = r.packageInfo.makeApplication(false, mInstrumentation);  
  
 if (*localLOGV*) Slog.v(*TAG*, "Performing launch of " + r);  
 if (*localLOGV*) Slog.v(  
 *TAG*, r + ": app=" + app  
 + ", appName=" + app.getPackageName()  
 + ", pkg=" + r.packageInfo.getPackageName()  
 + ", comp=" + r.intent.getComponent().toShortString()  
 + ", dir=" + r.packageInfo.getAppDir());  
  
 if (activity != null) {  
 Context appContext = createBaseContextForActivity(r, activity);  
 CharSequence title = r.activityInfo.loadLabel(appContext.getPackageManager());  
 Configuration config = new Configuration(mCompatConfiguration);  
 if (r.overrideConfig != null) {  
 config.updateFrom(r.overrideConfig);  
 }  
 if (*DEBUG\_CONFIGURATION*) Slog.v(*TAG*, "Launching activity "  
 + r.activityInfo.name + " with config " + config);  
 Window window = null;  
 if (r.mPendingRemoveWindow != null && r.mPreserveWindow) {  
 window = r.mPendingRemoveWindow;  
 r.mPendingRemoveWindow = null;  
 r.mPendingRemoveWindowManager = null;  
 }  
 //3、关联activity和appliaction

activity.attach(appContext, this, getInstrumentation(), r.token,  
 r.ident, app, r.intent, r.activityInfo, title, r.parent,  
 r.embeddedID, r.lastNonConfigurationInstances, config,  
 r.referrer, r.voiceInteractor, window);  
  
 if (customIntent != null) {  
 activity.mIntent = customIntent;  
 }  
 r.lastNonConfigurationInstances = null;  
 activity.mStartedActivity = false;  
 int theme = r.activityInfo.getThemeResource();  
 if (theme != 0) {  
 activity.setTheme(theme);  
 }  
  
 activity.mCalled = false;  
 if (r.isPersistable()) {  
 mInstrumentation.callActivityOnCreate(activity, r.state, r.persistentState);  
 } else {  
 mInstrumentation.callActivityOnCreate(activity, r.state);  
 }  
 if (!activity.mCalled) {  
 throw new SuperNotCalledException(  
 "Activity " + r.intent.getComponent().toShortString() +  
 " did not call through to super.onCreate()");  
 }  
 r.activity = activity;  
 r.stopped = true;  
 if (!r.activity.mFinished) {  
 activity.performStart();  
 r.stopped = false;  
 }  
 if (!r.activity.mFinished) {  
 if (r.isPersistable()) {  
 if (r.state != null || r.persistentState != null) {  
 mInstrumentation.callActivityOnRestoreInstanceState(activity, r.state,  
 r.persistentState);  
 }  
 } else if (r.state != null) {  
 mInstrumentation.callActivityOnRestoreInstanceState(activity, r.state);  
 }  
 }  
 if (!r.activity.mFinished) {  
 activity.mCalled = false;  
 if (r.isPersistable()) {  
 mInstrumentation.callActivityOnPostCreate(activity, r.state,  
 r.persistentState);  
 } else {  
 mInstrumentation.callActivityOnPostCreate(activity, r.state);  
 }  
 if (!activity.mCalled) {  
 throw new SuperNotCalledException(  
 "Activity " + r.intent.getComponent().toShortString() +  
 " did not call through to super.onPostCreate()");  
 }  
 }  
 }  
 r.paused = true;  
  
 mActivities.put(r.token, r);  
  
 } catch (SuperNotCalledException e) {  
 throw e;  
  
 } catch (Exception e) {  
 if (!mInstrumentation.onException(activity, e)) {  
 throw new RuntimeException(  
 "Unable to start activity " + component  
 + ": " + e.toString(), e);  
 }  
 }  
  
 return activity;  
}

AMS====>ActivityStarter( startActivityMayWait(...)--->startActivityLocked(...)--->startActivityUnchecked(...));

=====>ActivityStackSupervisor( sumeFocusedStackTopActivityLocked(.....)----> )

=====>ActivityStack( resumeTopActivityUncheckedLocked(...)---->resumeTopActivityinnerLocked(..));

=====> ActivityStackSupervisor(startSpecificActivityLocked(...)----->realStartActivityLocked(...)---------->app.thread启动activity).