2.新版StartActivity启动流程

根Activity启动流程

点击桌面图标，调用Launcher程序的startAactivtySafely方法

```
public boolean startActivitySafely(View v, Intent intent, @Nullable ItemInfo item) {
    if (mIsSafeModeEnabled && !PackageManagerHelper.isSystemApp(this, intent)) {
        Toast.makeText(this, R.string.safemode_shortcut_error, Toast.LENGTH_SHORT).show();
        return false;
    }

    Bundle optsBundle = (v != null) ? getActivityLaunchOptions(v, item).toBundle() : null;
    UserHandle user = item == null ? null : item.user;

    // Prepare intent
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    if (v != null) {
        intent.setSourceBounds(Utilities.getViewBounds(v));
    }
    try {
        boolean isShortcut = (item instanceof WorkspaceItemInfo)
                && (item.itemType == Favorites.ITEM_TYPE_SHORTCUT
                || item.itemType == Favorites.ITEM_TYPE_DEEP_SHORTCUT)
                && !((WorkspaceItemInfo) item).isPromise();
        if (isShortcut) {
            // Shortcuts need some special checks due to legacy reasons.
            startShortcutIntentSafely(intent, optsBundle, item);
        } else if (user == null || user.equals(Process.myUserHandle())) {
            // Could be launching some bookkeeping activity
            startActivity(intent, optsBundle);
        } else {
            getSystemService(LauncherApps.class).startMainActivity(
                    intent.getComponent(), user, intent.getSourceBounds(), optsBundle);
        }
        if (item != null) {
            InstanceId instanceId = new InstanceIdSequence().newInstanceId();
            logAppLaunch(getStatsLogManager(), item, instanceId);
        }
        return true;
    } catch (NullPointerException | ActivityNotFoundException | SecurityException e) {
        Toast.makeText(this, R.string.activity_not_found, Toast.LENGTH_SHORT).show();
        Log.e(TAG, "Unable to launch. tag=" + item + " intent=" + intent, e);
    }
    return false;
}
```

接着调用了startActivityForResult(intent,-1,x) -1表示不用返回结果

```
@Override
public void startActivity(Intent intent, @Nullable Bundle options) {
    if (mIntent != null && mIntent.hasExtra(AutofillManager.EXTRA_RESTORE_SESSION_TOKEN)
            && mIntent.hasExtra(AutofillManager.EXTRA_RESTORE_CROSS_ACTIVITY)) {
        if (TextUtils.equals(getPackageName(),
                intent.resolveActivity(getPackageManager()).getPackageName())) {
            // Apply Autofill restore mechanism on the started activity by startActivity()
            final IBinder token =
                    mIntent.getIBinderExtra(AutofillManager.EXTRA_RESTORE_SESSION_TOKEN);
            // Remove restore ability from current activity
            mIntent.removeExtra(AutofillManager.EXTRA_RESTORE_SESSION_TOKEN);
            mIntent.removeExtra(AutofillManager.EXTRA_RESTORE_CROSS_ACTIVITY);
            // Put restore token
            intent.putExtra(AutofillManager.EXTRA_RESTORE_SESSION_TOKEN, token);
            intent.putExtra(AutofillManager.EXTRA_RESTORE_CROSS_ACTIVITY, true);
        }
    }
    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);
    }
}
```

这里调用了mInstrumentation.execstartActivity

Instrumentation 是Android系统中一系列控制方法的集合(hook),这些方法可以在正常的生命周期之外控制Android控件的运行，也可以控制Andoroid如何加载应用程序

```
 */
public void startActivityForResult(@RequiresPermission Intent intent, int requestCode,
        @Nullable Bundle options) {
    if (mParent == null) {
        options = transferSpringboardActivityOptions(options);
        Instrumentation.ActivityResult ar =
            mInstrumentation.execStartActivity(
                this, mMainThread.getApplicationThread(), mToken, this,
                intent, requestCode, options);
        if (ar != null) {
            mMainThread.sendActivityResult(
                mToken, mEmbeddedID, requestCode, ar.getResultCode(),
                ar.getResultData());
        }
        if (requestCode >= 0) {
```

调用了ActivityTaskManager.getService().startActivity:

```
    }
    try {
        intent.migrateExtraStreamToClipData(who);
        intent.prepareToLeaveProcess(who);
        int result = ActivityTaskManager.getService().startActivity(whoThread,
                who.getOpPackageName(), who.getAttributionTag(), 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;
}
```

调用get()

```
    /** @hide */
    public static IActivityTaskManager getService() {
        return IActivityTaskManagerSingleton.get();
    }
```

再get中调用了create()方法：

```
    public final T get() {
        synchronized (this) {
            if (mInstance == null) {
                mInstance = create();
            }
            return mInstance;
        }
    }
```

这个单例类，拿到了ATMS服务，然后通过asInterface拿到了服务端在本地的代理对象IActivityTaskManager

```
    @UnsupportedAppUsage(trackingBug = 129726065)
    private static final Singleton<IActivityTaskManager> IActivityTaskManagerSingleton =
            new Singleton<IActivityTaskManager>() {
                @Override
                protected IActivityTaskManager create() {
                    final IBinder b = ServiceManager.getService(Context.ACTIVITY_TASK_SERVICE);
                    return IActivityTaskManager.Stub.asInterface(b);
                }
            };
```

就调用ATMS服务的startActivity方法

```
@Override
public final int startActivity(IApplicationThread caller, String callingPackage,
        String callingFeatureId, Intent intent, String resolvedType, IBinder resultTo,
        String resultWho, int requestCode, int startFlags, ProfilerInfo profilerInfo,
        Bundle bOptions) {
    return startActivityAsUser(caller, callingPackage, callingFeatureId, intent, resolvedType,
            resultTo, resultWho, requestCode, startFlags, profilerInfo, bOptions,
            UserHandle.getCallingUserId());
}
```

```
@Override
public int startActivityAsUser(IApplicationThread caller, String callingPackage,
        String callingFeatureId, Intent intent, String resolvedType, IBinder resultTo,
        String resultWho, int requestCode, int startFlags, ProfilerInfo profilerInfo,
        Bundle bOptions, int userId) {
    return startActivityAsUser(caller, callingPackage, callingFeatureId, intent, resolvedType,
            resultTo, resultWho, requestCode, startFlags, profilerInfo, bOptions, userId,
            true /*validateIncomingUser*/);
}

private int startActivityAsUser(IApplicationThread caller, String callingPackage,
        @Nullable String callingFeatureId, Intent intent, String resolvedType,
        IBinder resultTo, String resultWho, int requestCode, int startFlags,
        ProfilerInfo profilerInfo, Bundle bOptions, int userId, boolean validateIncomingUser) {
    assertPackageMatchesCallingUid(callingPackage);
    enforceNotIsolatedCaller("startActivityAsUser");

    userId = getActivityStartController().checkTargetUser(userId, validateIncomingUser,
            Binder.getCallingPid(), Binder.getCallingUid(), "startActivityAsUser");

    // TODO: Switch to user app stacks here.
    return getActivityStartController().obtainStarter(intent, "startActivityAsUser")
            .setCaller(caller)
            .setCallingPackage(callingPackage)
            .setCallingFeatureId(callingFeatureId)
            .setResolvedType(resolvedType)
            .setResultTo(resultTo)
            .setResultWho(resultWho)
            .setRequestCode(requestCode)
            .setStartFlags(startFlags)
            .setProfilerInfo(profilerInfo)
            .setActivityOptions(bOptions)
            .setUserId(userId)
            .execute();
}

ActivityStartController getActivityStartController() {
    return mActivityStartController;
}

private ActivityStartController mActivityStartController;
```

查看obtainStarter： ActivityStarter.java

```
ActivityStarter obtainStarter(Intent intent, String reason) {
    return mFactory.obtain().setIntent(intent).setReason(reason);
}
```

一个工厂接口，返回的是ActivityStarter类

```
 */
@VisibleForTesting
interface Factory {
    /**
     * Sets the {@link ActivityStartController} to be passed to {@link ActivityStarter}.
     */
    void setController(ActivityStartController controller);

    /**
     * Generates an {@link ActivityStarter} that is ready to handle a new start request.
     * @param controller The {@link ActivityStartController} which the starter who will own
     *                   this instance.
     * @return an {@link ActivityStarter}
     */
    ActivityStarter obtain();

    /**
     * Recycles a starter for reuse.
     */
    void recycle(ActivityStarter starter);
}
```

这个类有个默认工厂会初始化：

```
/**
 * Default implementation of {@link StarterFactory}.
 */
static class DefaultFactory implements Factory {
    /**
     * The maximum count of starters that should be active at one time:
     * 1. last ran starter (for logging and post activity processing)
     * 2. current running starter
     * 3. starter from re-entry in (2)
     */
    private final int MAX_STARTER_COUNT = 3;

    private ActivityStartController mController;
    private ActivityTaskManagerService mService;
    private ActivityTaskSupervisor mSupervisor;
    private ActivityStartInterceptor mInterceptor;

    private SynchronizedPool<ActivityStarter> mStarterPool =
            new SynchronizedPool<>(MAX_STARTER_COUNT);

    DefaultFactory(ActivityTaskManagerService service,
            ActivityTaskSupervisor supervisor, ActivityStartInterceptor interceptor) {
        mService = service;
        mSupervisor = supervisor;
        mInterceptor = interceptor;
    }

    @Override
    public void setController(ActivityStartController controller) {
        mController = controller;
    }

    @Override
    public ActivityStarter obtain() {
        ActivityStarter starter = mStarterPool.acquire();

        if (starter == null) {
            if (mService.mRootWindowContainer == null) {
                throw new IllegalStateException("Too early to start activity.");
            }
            starter = new ActivityStarter(mController, mService, mSupervisor, mInterceptor);
        }

        return starter;
    }
```

初始化大小为3

这里有个同步池，存放的这个类对象

所以我们知道最后调用execute()方法就是这个类（ActivityStarter)调用的

```
        return res;
    }
    res = executeRequest(mRequest);
```

在execute方法调用了executeRequest方法，

这里会对本次启动activity的权限进行检测，是否 在清单文件注册，等

```
boolean abort = !mSupervisor.checkStartAnyActivityPermission(intent, aInfo, resultWho,
        requestCode, callingPid, callingUid, callingPackage, callingFeatureId,
        request.ignoreTargetSecurity, inTask != null, callerApp, resultRecord,
        resultRootTask);
abort |= !mService.mIntentFirewall.checkStartActivity(intent, callingUid,
        callingPid, resolvedType, aInfo.applicationInfo);
abort |= !mService.getPermissionPolicyInternal().checkStartActivity(intent, callingUid,
        callingPackage);
```

接着创建了ActivityRecord对象，并传入到了startActivityUnchecked方法。启动了activity后，返回了这个对象

```
}
final ActivityRecord r = new ActivityRecord.Builder(mService)
        .setCaller(callerApp)
        .setLaunchedFromPid(callingPid)
        .setLaunchedFromUid(callingUid)
        .setLaunchedFromPackage(callingPackage)
        .setLaunchedFromFeature(callingFeatureId)
        .setIntent(intent)
        .setResolvedType(resolvedType)
        .setActivityInfo(aInfo)
        .setConfiguration(mService.getGlobalConfiguration())
        .setResultTo(resultRecord)
        .setResultWho(resultWho)
        .setRequestCode(requestCode)
        .setComponentSpecified(request.componentSpecified)
        .setRootVoiceInteraction(voiceSession != null)
        .setActivityOptions(checkedOptions)
        .setSourceRecord(sourceRecord)
        .build();

mLastStartActivityRecord = 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;
}

// Only allow app switching to be resumed if activity is not a restricted background
// activity and target app is not home process, otherwise any background activity
// started in background task can stop home button protection mode.
// As the targeted app is not a home process and we don't need to wait for the 2nd
// activity to be started to resume app switching, we can just enable app switching
// directly.
WindowProcessController homeProcess = mService.mHomeProcess;
boolean isHomeProcess = homeProcess != null
        && aInfo.applicationInfo.uid == homeProcess.mUid;
if (!restrictedBgActivity && !isHomeProcess) {
    mService.resumeAppSwitches();
}

mLastStartActivityResult = startActivityUnchecked(r, sourceRecord, voiceSession,
        request.voiceInteractor, startFlags, true /* doResume */, checkedOptions,
        inTask, inTaskFragment, restrictedBgActivity, intentGrants);
```

构建ActivityRecord 对象

继续查看startActivityUnchecked方法：

```
try {
    mService.deferWindowLayout();
    Trace.traceBegin(Trace.TRACE_TAG_WINDOW_MANAGER, "startActivityInner");
    result = startActivityInner(r, sourceRecord, voiceSession, voiceInteractor,
            startFlags, doResume, options, inTask, inTaskFragment, restrictedBgActivity,
            intentGrants);
    startResultSuccessful = ActivityManager.isStartResultSuccessful(result);
    final boolean taskAlwaysOnTop = options != null && options.getTaskAlwaysOnTop();
    // Apply setAlwaysOnTop when starting an Activity is successful regardless of creating
    // a new Activity or recycling the existing Activity.
    if (taskAlwaysOnTop && startResultSuccessful) {
        final Task targetRootTask =
                mTargetRootTask != null ? mTargetRootTask : mTargetTask.getRootTask();
        targetRootTask.setAlwaysOnTop(true);
    }
```

查看startActivityLocked方法

```
final boolean isTaskSwitch = startedTask != prevTopTask && !startedTask.isEmbedded();
mTargetRootTask.startActivityLocked(mStartActivity,
        topRootTask != null ? topRootTask.getTopNonFinishingActivity() : null, newTask,
        isTaskSwitch, mOptions, sourceRecord);
if (mDoResume) {
```

调用了ensureActivitiesVisible:

```
    if (r.mLaunchTaskBehind) {
        // Don't do a starting window for mLaunchTaskBehind. More importantly make sure we
        // tell WindowManager that r is visible even though it is at the back of the root
        // task.
        r.setVisibility(true);
        ensureActivitiesVisible(null, 0, !PRESERVE_WINDOWS);
        // Go ahead to execute app transition for this activity since the app transition
        // will not be triggered through the resume channel.
        mDisplayContent.executeAppTransition();
    } else if (SHOW_APP_STARTING_PREVIEW && doShow) {
```

调用了updateActivityVisibilities

```
// TODO: Should be re-worked based on the fact that each task as a root task in most cases.
void ensureActivitiesVisible(@Nullable ActivityRecord starting, int configChanges,
        boolean preserveWindows, boolean notifyClients) {
    mTaskSupervisor.beginActivityVisibilityUpdate();
    try {
        forAllLeafTasks(task -> {
            task.updateActivityVisibilities(starting, configChanges, preserveWindows,
                    notifyClients);
        }, true /* traverseTopToBottom */);

        if (mTranslucentActivityWaiting != null &&
                mUndrawnActivitiesBelowTopTranslucent.isEmpty()) {
            // Nothing is getting drawn or everything was already visible, don't wait for
            // timeout.
            notifyActivityDrawnLocked(null);
        }
    } finally {
        mTaskSupervisor.endActivityVisibilityUpdate();
    }
}
```

调用了process:

```
final void updateActivityVisibilities(@Nullable ActivityRecord starting, int configChanges,
        boolean preserveWindows, boolean notifyClients) {
    mTaskSupervisor.beginActivityVisibilityUpdate();
    try {
        mEnsureActivitiesVisibleHelper.process(
                starting, configChanges, preserveWindows, notifyClients);
    } finally {
        mTaskSupervisor.endActivityVisibilityUpdate();
    }
}
```

调用了 setActivityVisibilityState

```
8           } else if (child.asActivityRecord() != null) {
9               setActivityVisibilityState(child.asActivityRecord(), starting, resumeTopActivity);
0           }
1       }
2       if (mTaskFragment.mTransitionController.isShellTransitionsEnabled()) {
3           mTaskFragment.getDisplayContent().mWallpaperController.adjustWallpaperWindows();
```

我们查看： makeVisibleAndRestartIfNeeded 这是还不存在该进程

```
        if (!r.attachedToProcess()) {
            makeVisibleAndRestartIfNeeded(mStarting, mConfigChanges, isTop,
                    resumeTopActivity && isTop, r);
        } else if (r.mVisibleRequested) {
            // If this activity is already visible, then there is nothing to do here.
            if (DEBUG_VISIBILITY) {
                Slog.v(TAG_VISIBILITY, "Skipping: already visible at " + r);
            }

            if (r.mClientVisibilityDeferred && mNotifyClients) {
                r.makeActiveIfNeeded(r.mClientVisibilityDeferred ? null : starting);
                r.mClientVisibilityDeferred = false;
            }

            r.handleAlreadyVisible();
            if (mNotifyClients) {
                r.makeActiveIfNeeded(mStarting);
            }
        } else {
            r.makeVisibleIfNeeded(mStarting, mNotifyClients);
        }
        // Aggregate current change flags.
```

如果activity还没有起来applicatoin==null && app.thread == null

就会调用makeVisibleRestartInNeeded

这里的方法也会启动activity

调用：startSpecificActivity

```
    private void makeVisibleAndRestartIfNeeded(ActivityRecord starting, int configChanges,
            boolean isTop, boolean andResume, ActivityRecord r) {
        // We need to make sure the app is running if it's the top, or it is just made visible from
        // invisible. If the app is already visible, it must have died while it was visible. In this
        // case, we'll show the dead window but will not restart the app. Otherwise we could end up
        // thrashing.
        if (!isTop && r.mVisibleRequested) {
            return;
        }

        // This activity needs to be visible, but isn't even running...
        // get it started and resume if no other root task in this root task is resumed.
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Start and freeze screen for " + r);
        }
        if (r != starting) {
            r.startFreezingScreenLocked(configChanges);
        }
        if (!r.mVisibleRequested || r.mLaunchTaskBehind) {
            if (DEBUG_VISIBILITY) {
                Slog.v(TAG_VISIBILITY, "Starting and making visible: " + r);
            }
            r.setVisibility(true);
        }
        if (r != starting) {
            mTaskFragment.mTaskSupervisor.startSpecificActivity(r, andResume,
                    true /* checkConfig */);
        }
    }
}
```

我们查看makeActiveIfNeeded

这是已经存在的进程。

```
boolean makeActiveIfNeeded(ActivityRecord activeActivity) {
    if (shouldResumeActivity(activeActivity)) {
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Resume visible activity, " + this);
        }
        return getRootTask().resumeTopActivityUncheckedLocked(activeActivity /* prev */,
                null /* options */);
    } else if (shouldPauseActivity(activeActivity)) {
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Pause visible activity, " + this);
        }
        // An activity must be in the {@link PAUSING} state for the system to validate
        // the move to {@link PAUSED}.
        setState(PAUSING, "makeActiveIfNeeded");
        try {
            mAtmService.getLifecycleManager().scheduleTransaction(app.getThread(), appToken,
                    PauseActivityItem.obtain(finishing, false /* userLeaving */,
                            configChangeFlags, false /* dontReport */));
        } catch (Exception e) {
            Slog.w(TAG, "Exception thrown sending pause: " + intent.getComponent(), e);
        }
    } else if (shouldStartActivity()) {
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Start visible activity, " + this);
        }
        setState(STARTED, "makeActiveIfNeeded");

        try {
            mAtmService.getLifecycleManager().scheduleTransaction(app.getThread(), appToken,
                    StartActivityItem.obtain(takeOptions()));
        } catch (Exception e) {
            Slog.w(TAG, "Exception thrown sending start: " + intent.getComponent(), e);
        }
        // The activity may be waiting for stop, but that is no longer appropriate if we are
        // starting the activity again
        mTaskSupervisor.mStoppingActivities.remove(this);
    }
    return false;
}
```

需要resume的话

这里调用resumeTopActivityUncheckedLocked

需要pause的话

这2个都是 调用scheduleTransaction

需要start的话

回到startSpecificActivity

```
void startSpecificActivity(ActivityRecord r, boolean andResume, boolean checkConfig) {
    // Is this activity's application already running?
    final WindowProcessController wpc =
            mService.getProcessController(r.processName, r.info.applicationInfo.uid);

    boolean knownToBeDead = false;
    if (wpc != null && wpc.hasThread()) {
        try {
            realStartActivityLocked(r, wpc, 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.
        knownToBeDead = true;
    }

    r.notifyUnknownVisibilityLaunchedForKeyguardTransition();

    final boolean isTop = andResume && r.isTopRunningActivity();
    mService.startProcessAsync(r, knownToBeDead, isTop, isTop ? "top-activity" : "activity");
}
```

进程存在了

进程还不存在

realStartActivityLocked

```
}

// Create activity launch transaction.
final ClientTransaction clientTransaction = ClientTransaction.obtain(
        proc.getThread(), r.appToken);

final boolean isTransitionForward = r.isTransitionForward();
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.getFilteredReferrer(r.launchedFromPackage), task.voiceInteractor,
        proc.getReportedProcState(), r.getSavedState(), r.getPersistentSavedState(),
        results, newIntents, r.takeOptions(), isTransitionForward,
        proc.createProfilerInfoIfNeeded(), r.assistToken, activityClientController,
        r.createFixedRotationAdjustmentsIfNeeded(), r.shareableActivityToken,
        r.getLaunchedFromBubble()));

// Set desired final state.
final ActivityLifecycleItem lifecycleItem;
if (andResume) {
    lifecycleItem = ResumeActivityItem.obtain(isTransitionForward);
} else {
    lifecycleItem = PauseActivityItem.obtain();
}
clientTransaction.setLifecycleStateRequest(lifecycleItem);

// Schedule transaction.
mService.getLifecycleManager().scheduleTransaction(clientTransaction);
```

获取一个ClientTransaction对象

给这个获取的对象添加callback回调，参数是一个LauncherActivitiyItem对象

设置生命周期状态请求

执行事务

scheduleTransaction

```
    */
    void scheduleTransaction(ClientTransaction transaction) throws RemoteException {
        final IApplicationThread client = transaction.getClient();
        transaction.schedule();
        if (!(client instanceof Binder)) {
            // If client is not an instance of Binder - it's a remote call and at this point it is
            // safe to recycle the object. All objects used for local calls will be recycled after
            // the transaction is executed on client in ActivityThread.
            transaction.recycle();
        }
    }


    */
    public void schedule() throws RemoteException {
        mClient.scheduleTransaction(this);
    }
```

拿到了IApplicatoinTHread 代理对象并调用schedule

Binder通信：

frameworks/base/core/java/android/app/ActivityThread.java

ActivityThread.java

```
1803        public void handleTrustStorageUpdate() {
1804            NetworkSecurityPolicy.getInstance().handleTrustStorageUpdate();
1805        }
1806
1807        @Override
1808        public void scheduleTransaction(ClientTransaction transaction) throws RemoteException {
1809            ActivityThread.this.scheduleTransaction(transaction);
1810        }
1811
```

通过Binder来到AT

scheduleTransaction

frameworks/base/core/java/android/app/ClientTransactionHandler.java

entTransactionHandler.java

```java
public abstract class ClientTransactionHandler {

    // Schedule phase related logic and handlers.

    /** Prepare and schedule transaction for execution. */
    void scheduleTransaction(ClientTransaction transaction) {
        transaction.preExecute(this);
        sendMessage(ActivityThread.H.EXECUTE_TRANSACTION, transaction);
    }
```

线程通信：调用execute

```java
    case EXECUTE_TRANSACTION:
        final ClientTransaction transaction = (ClientTransaction) msg.obj;
        mTransactionExecutor.execute(transaction);
        if (isSystem()) {
            // Client transactions inside system process are recycled on the client side
            // instead of ClientLifecycleManager to avoid being cleared before this
            // message is handled.
            transaction.recycle();
        }
        // TODO(lifecycler): Recycle locally scheduled transactions.
        break;
```

接着调用了 executeCallbacks:

```java
executeCallbacks(transaction);
```
调用了回调

```java
executeLifecycleState(transaction);
```
以及生命周期状态

item.execute: 我们知道item就是之前在创建时通过addCallbacks添加的LaunchActivityItem类

而LifeCycleItem就是ResumeActiivtyItem:

```java
// Set desired final state.
final ActivityLifecycleItem lifecycleItem;
if (andResume) {
    lifecycleItem = ResumeActivityItem.obtain(isTransitionForward);
} else {
    lifecycleItem = PauseActivityItem.obtain();
}
clientTransaction.setLifecycleStateRequest(lifecycleItem);
```
设置生命周期状态请求

```java
// Schedule transaction.
mService.getLifecycleManager().scheduleTransaction(clientTransaction);
```
执行事务

在executeCallbakcs调用了execute:

```java
item.execute(mTransactionHandler, token, mPendingActions);
```

所以查看LaunchActivityItem的execute

```
@Override
public void execute(ClientTransactionHandler client, IBinder token,
        PendingTransactionActions pendingActions) {
    Trace.traceBegin(TRACE_TAG_ACTIVITY_MANAGER, "activityStart");
    ActivityClientRecord r = client.getLaunchingActivity(token);
    client.handleLaunchActivity(r, pendingActions, null /* customIntent */);
    Trace.traceEnd(TRACE_TAG_ACTIVITY_MANAGER);
}
```

我们知道ClientTransacationHandler的实现类时ActivityThread：

```
public final class ActivityThread extends ClientTransactionHandler
        implements ActivityThreadInternal {
    /** @hide */
```

所以查看ActivityThread.handleLaunchActivity:

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

调用了performLaunchActivity:并返回了Activity对象

```
/**  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(isProtectedComponent(r.activityInfo),
                appContext.getAttributionSource());
        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 {
        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());

        // updatePendingActivityConfiguration() reads from mActivities to update
        // ActivityClientRecord which runs in a different thread. Protect modifications to
```

1 创建 component对象

2 创建上下文环境context

3 获取加载类的加载器ClassLoader，然后通过反射创建了Activity对象

4 创建application对象，并调用AttachBaseContext和onCreate方法

```
        // mActivities to avoid race.
        synchronized (mResourcesManager) {
            mActivities.put(r.token, r);
        }

        if (activity != null) {
            CharSequence title = r.activityInfo.loadLabel(appContext.getPackageManager());
            Configuration config =
                    new Configuration(mConfigurationController.getCompatConfiguration());
            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;
            }
                                        5 通过获取统一的资源对象去获取资源
            // Activity resources must be initialized with the same loaders as the
            // application context.
            appContext.getResources().addLoaders(
                    app.getResources().getLoaders().toArray(new ResourcesLoader[0]));

            appContext.setOuterContext(activity);
            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, r.configCallback,
                    r.assistToken, r.shareableActivityToken);

            if (customIntent != null) {
                activity.mIntent = customIntent;
            }                       6 调用attach方法
            r.lastNonConfigurationInstances = null;
            checkAndBlockForNetworkAccess();
            activity.mStartedActivity = false;
            int theme = r.activityInfo.getThemeResource();
            if (theme != 0) {
                activity.setTheme(theme);
            }

            if (r.mActivityOptions != null) {
                activity.mPendingOptions = r.mActivityOptions;
                r.mActivityOptions = null;
            }
            activity.mLaunchedFromBubble = r.mLaunchedFromBubble;
            activity.mCalled = false;
            // Assigning the activity to the record before calling onCreate() allows
            // ActivityThread#getActivity() lookup for the callbacks triggered from
            // ActivityLifecycleCallbacks#onActivityCreated() or
            // ActivityLifecycleCallback#onActivityPostCreated().
            r.activity = activity;
            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()");
            }
            mLastReportedWindowingMode.put(activity.getActivityToken(),
                    config.windowConfiguration.getWindowingMode());
        }
        r.setState(ON_CREATE);

    } 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;    8 返回Activity
}
```

7 最后调用Activiti.onCreate方法

查看newActivity：

```
    */
    public Activity newActivity(ClassLoader cl, String className,
            Intent intent)
            throws InstantiationException, IllegalAccessException,
            ClassNotFoundException {
        String pkg = intent != null && intent.getComponent() != null
                ? intent.getComponent().getPackageName() : null;
        return getFactory(pkg).instantiateActivity(cl, className, intent);
    }
```

查看instantiateActivity: 反射获取对象

```
    */
    public @NonNull Activity instantiateActivity(@NonNull ClassLoader cl, @NonNull String className,
            @Nullable Intent intent)
            throws InstantiationException, IllegalAccessException, ClassNotFoundException {
        return (Activity) cl.loadClass(className).newInstance();
    }
```

查看makeApplication：

```
    @UnsupportedAppUsage
    public Application makeApplication(boolean forceDefaultAppClass,
            Instrumentation instrumentation) {
        if (mApplication != null) {
            return mApplication;
        }

        Trace.traceBegin(Trace.TRACE_TAG_ACTIVITY_MANAGER, "makeApplication");

        Application app = null;

        String appClass = mApplicationInfo.className;
        if (forceDefaultAppClass || (appClass == null)) {
            appClass = "android.app.Application";
        }

        try {
            final java.lang.ClassLoader cl = getClassLoader();
            if (!mPackageName.equals("android")) {
                Trace.traceBegin(Trace.TRACE_TAG_ACTIVITY_MANAGER,
                        "initializeJavaContextClassLoader");
                initializeJavaContextClassLoader();
                Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
            }

            // Rewrite the R 'constants' for all library apks.
            SparseArray<String> packageIdentifiers = getAssets().getAssignedPackageIdentifiers(
                    false, false);
            for (int i = 0, n = packageIdentifiers.size(); i < n; i++) {
                final int id = packageIdentifiers.keyAt(i);
                if (id == 0x01 || id == 0x7f) {
                    continue;
                }

                rewriteRValues(cl, packageIdentifiers.valueAt(i), id);
            }

            ContextImpl appContext = ContextImpl.createAppContext(mActivityThread, this);
            // The network security config needs to be aware of multiple
            // applications in the same process to handle discrepancies
            NetworkSecurityConfigProvider.handleNewApplication(appContext);
            app = mActivityThread.mInstrumentation.newApplication(
                    cl, appClass, appContext);
            appContext.setOuterContext(app);
        } catch (Exception e) {
            if (!mActivityThread.mInstrumentation.onException(app, e)) {
```

1 获取类名

获取ClassLoader

2 创建AppContext环境

3 调用Instrumentation的newApplication对象

```
            Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
            throw new RuntimeException(
                "Unable to instantiate application " + appClass
                + " package " + mPackageName + ": " + e.toString(), e);
        }
    }
    mActivityThread.mAllApplications.add(app);
    mApplication = app;

    if (instrumentation != null) {
        try {
            instrumentation.callApplicationOnCreate(app);
        } catch (Exception e) {
            if (!instrumentation.onException(app, e)) {
                Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
                throw new RuntimeException(
                    "Unable to create application " + app.getClass().getName()
                    + ": " + e.toString(), e);
            }
        }
    }

    Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);

    return app;
}
```

4 调用onCreate方法

查看newApplication

```
 */
public Application newApplication(ClassLoader cl, String className, Context context)
        throws InstantiationException, IllegalAccessException,
        ClassNotFoundException {
    Application app = getFactory(context.getPackageName())
            .instantiateApplication(cl, className);
    app.attach(context);
    return app;
}
```

通过反射获取到APplication

```
    */
    public @NonNull Application instantiateApplication(@NonNull ClassLoader cl,
            @NonNull String className)
            throws InstantiationException, IllegalAccessException, ClassNotFoundException {
        return (Application) cl.loadClass(className).newInstance();
    }
```

app.attach调用了attachBaseContext:

```
 */
@UnsupportedAppUsage
/* package */ final void attach(Context context) {
    attachBaseContext(context);
    mLoadedApk = ContextImpl.getImpl(context).mPackageInfo;
}
```

查看activity.attach方法:

```
final void attach(Context context, ActivityThread aThread,
        Instrumentation instr, IBinder token, int ident,
        Application application, Intent intent, ActivityInfo info,
        CharSequence title, Activity parent, String id,
        NonConfigurationInstances lastNonConfigurationInstances,
        Configuration config, String referrer, IVoiceInteractor voiceInteractor,
        Window window, ActivityConfigCallback activityConfigCallback, IBinder assistToken,
        IBinder shareableActivityToken) {
    attachBaseContext(context);

    mFragments.attachHost(null /*parent*/);

    mWindow = new PhoneWindow(this, window, activityConfigCallback);
    mWindow.setWindowControllerCallback(mWindowControllerCallback);
    mWindow.setCallback(this);
    mWindow.setOnWindowDismissedCallback(this);
    mWindow.getLayoutInflater().setPrivateFactory(this);
    if (info.softInputMode != WindowManager.LayoutParams.SOFT_INPUT_STATE_UNSPECIFIED) {
        mWindow.setSoftInputMode(info.softInputMode);
    }
    if (info.uiOptions != 0) {
        mWindow.setUiOptions(info.uiOptions);
    }
    mUiThread = Thread.currentThread();

    mMainThread = aThread;
    mInstrumentation = instr;
    mToken = token;
    mAssistToken = assistToken;
    mShareableActivityToken = shareableActivityToken;
    mIdent = ident;
    mApplication = application;
    mIntent = intent;
    mReferrer = referrer;
    mComponent = intent.getComponent();
    mActivityInfo = info;
    mTitle = title;
    mParent = parent;
    mEmbeddedID = id;
```

将context赋值给父类，和父类关联

创建PhoneWindow对

设置windowmanager，和WM关联

```
mWindow.setWindowManager(
        (WindowManager)context.getSystemService(Context.WINDOW_SERVICE),
        mToken, mComponent.flattenToString(),
        (info.flags & ActivityInfo.FLAG_HARDWARE_ACCELERATED) != 0);
```

接着看executeLifecycleState(transaction):

```
    /** Transition to the final state if requested by the transaction. */
    private void executeLifecycleState(ClientTransaction transaction) {
        final ActivityLifecycleItem lifecycleItem = transaction.getLifecycleStateRequest();
        if (lifecycleItem == null) {
            // No lifecycle request, return early.
            return;
        }

        final IBinder token = transaction.getActivityToken();
        final ActivityClientRecord r = mTransactionHandler.getActivityClient(token);
        if (DEBUG_RESOLVER) {
            Slog.d(TAG, tId(transaction) + "Resolving lifecycle state: "
                    + lifecycleItem + " for activity: "
                    + getShortActivityName(token, mTransactionHandler));
        }

        if (r == null) {
            // Ignore requests for non-existent client records for now.
            return;
        }

        // Cycle to the state right before the final requested state.
        cycleToPath(r, lifecycleItem.getTargetState(), true /* excludeLastState */, transaction);

        // Execute the final transition with proper parameters.
        lifecycleItem.execute(mTransactionHandler, token, mPendingActions);
        lifecycleItem.postExecute(mTransactionHandler, token, mPendingActions);
    }
```

调用了lifecycleItem,这里是ResumeActivityItem:

```
    @Override
    public void execute(ClientTransactionHandler client, ActivityClientRecord r,
            PendingTransactionActions pendingActions) {
        Trace.traceBegin(TRACE_TAG_ACTIVITY_MANAGER, "activityResume");
        client.handleResumeActivity(r, true /* finalStateRequest */, mIsForward,
                "RESUME_ACTIVITY");
        Trace.traceEnd(TRACE_TAG_ACTIVITY_MANAGER);
    }
```

调用了ClientTransactionHandler.handleResumeActivity,而实现类就是ActivityThread:

```
 */
public final class ActivityThread extends ClientTransactionHandler
        implements ActivityThreadInternal {
    /** @hide */
    public static final String TAG = "ActivityThread";
```

最后就是先调用onPauce，然后看是否是pause状态，是的话就调用onRestart，不是的话，就直接调用onResume

自此，启动流程分析完毕。

接着看看进程没有启动的流程（冷启动）

```
void startSpecificActivity(ActivityRecord r, boolean andResume, boolean checkConfig) {
    // Is this activity's application already running?
    final WindowProcessController wpc =
            mService.getProcessController(r.processName, r.info.applicationInfo.uid);

    boolean knownToBeDead = false;
    if (wpc != null && wpc.hasThread()) {
        try {
            realStartActivityLocked(r, wpc, 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.
        knownToBeDead = true;
    }

    r.notifyUnknownVisibilityLaunchedForKeyguardTransition();

    final boolean isTop = andResume && r.isTopRunningActivity();
    mService.startProcessAsync(r, knownToBeDead, isTop, isTop ? "top-activity" : "activity");
}
```

```
void startProcessAsync(ActivityRecord activity, boolean knownToBeDead, boolean isTop,
        String hostingType) {
    try {
        if (Trace.isTagEnabled(TRACE_TAG_WINDOW_MANAGER)) {
            Trace.traceBegin(TRACE_TAG_WINDOW_MANAGER, "dispatchingStartProcess:"
                    + activity.processName);
        }
        // Post message to start process to avoid possible deadlock of calling into AMS with the
        // ATMS lock held.
        final Message m = PooledLambda.obtainMessage(ActivityManagerInternal::startProcess,
                mAmInternal, activity.processName, activity.info.applicationInfo, knownToBeDead,
                isTop, hostingType, activity.intent.getComponent());
        mH.sendMessage(m);
    } finally {
        Trace.traceEnd(TRACE_TAG_WINDOW_MANAGER);
    }
}
```

这里调用了startProcess
创建进程，然后这里sendMessage，将消息