**AppTransition动效源码分析**

**AppTransition代表activity组件的切换过程，启动或是退出activity都会执行AppTransition，Android系统定义了多达十几种应用的transition类型，这些类型定义具体可参考WindowManager类。这里主要以TRANSIT\_TASK\_OPEN类型为例，场景以桌面点击图库冷启动为例。**

**打开AppTransition的log：**

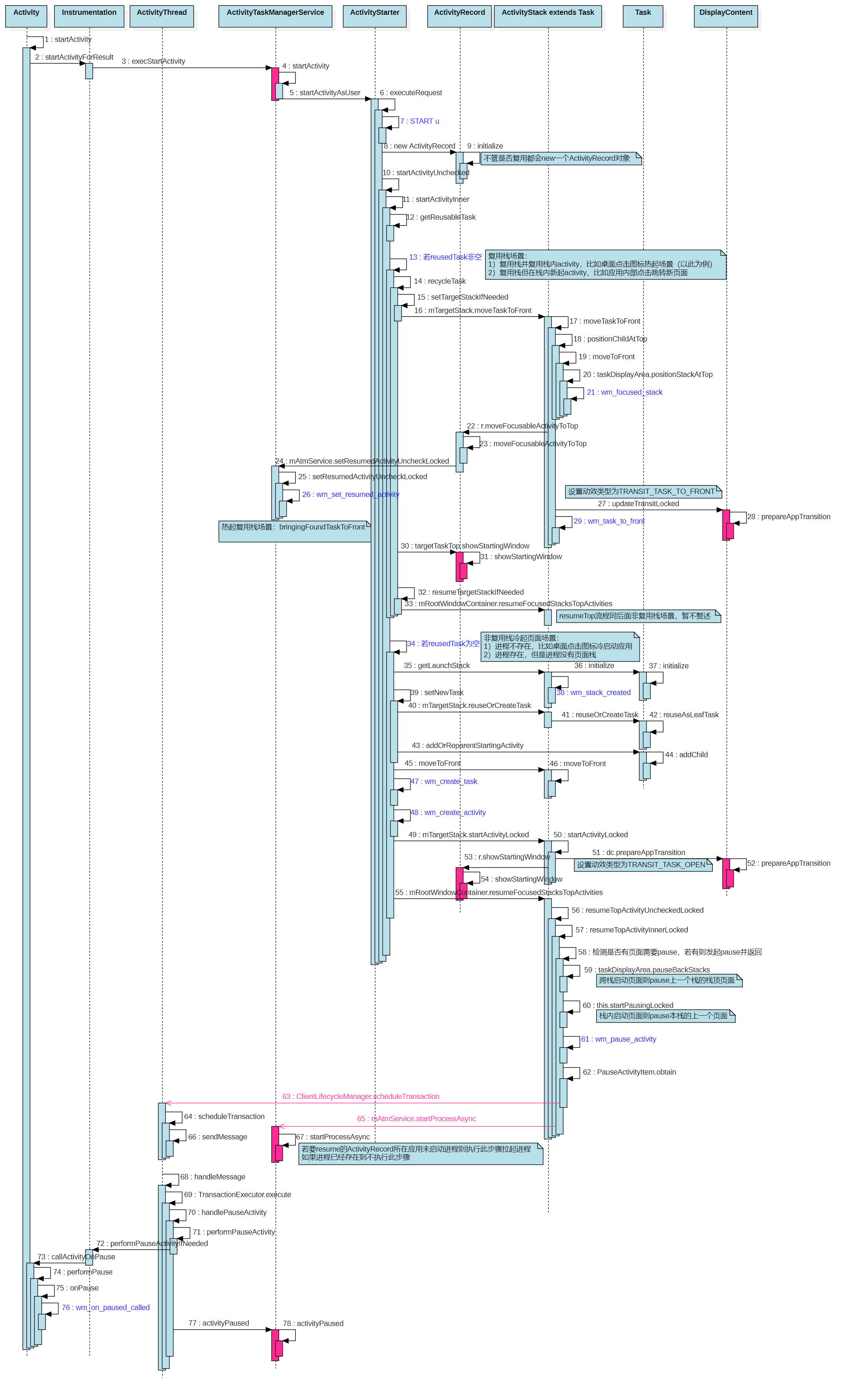
**adb shell wm logging enable-text WM\_DEBUG\_APP\_TRANSITIONS WM\_DEBUG\_REMOTE\_ANIMATIONS**

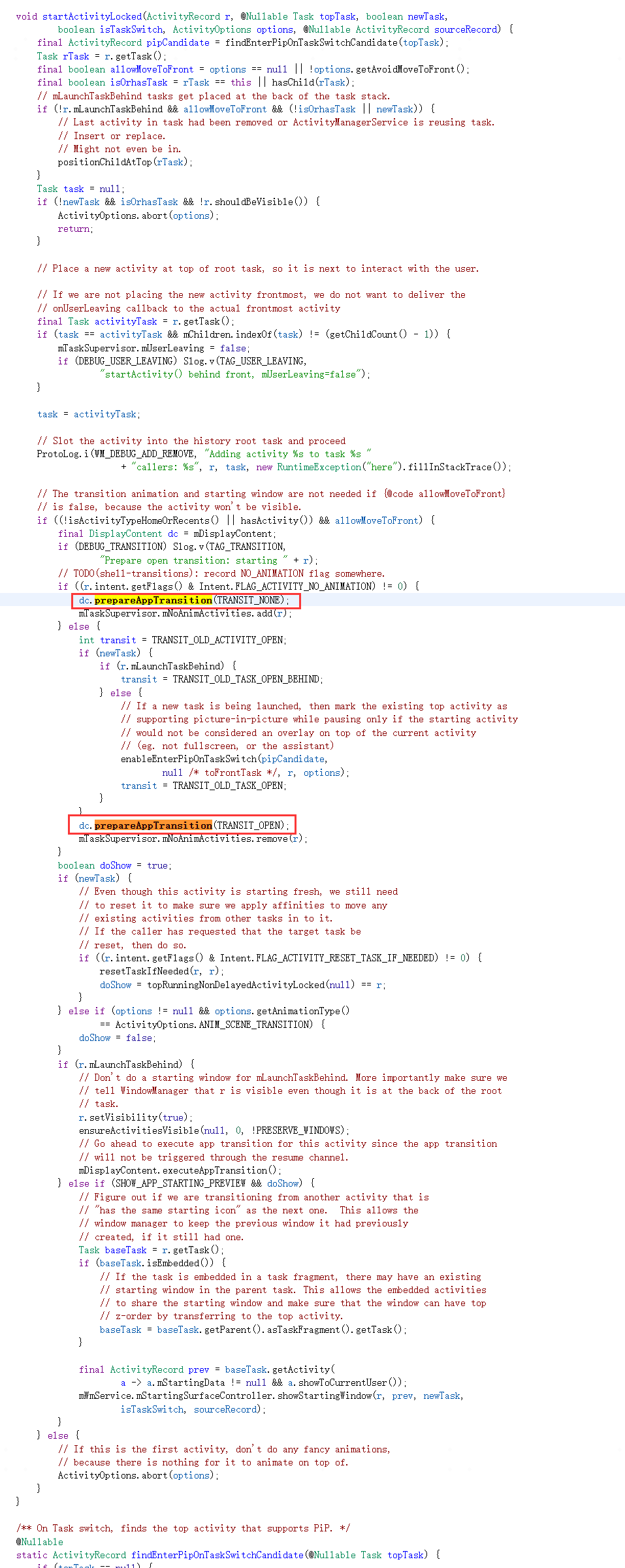
**其中，WM\_DEBUG\_APP\_TRANSITIONS代表开启的是AppTransition相关日志；WM\_DEBUG\_REMOTE\_ANIMATIONS表示开启的是RemoteAnimation相关的日志，桌面打开动效使用的就是RemoteAnimation。**

**一、冷启动跳转新应用**

**1、prepareAppTransition准备阶段**

**在startActivity阶段会调用DisplayContent.prepareAppTransition去设置AppTransition类型为TRANSIT\_TASK\_OPEN，执行的是图中reusedTask为空分支。**





**最后调用了showStartingWindow后面分析**

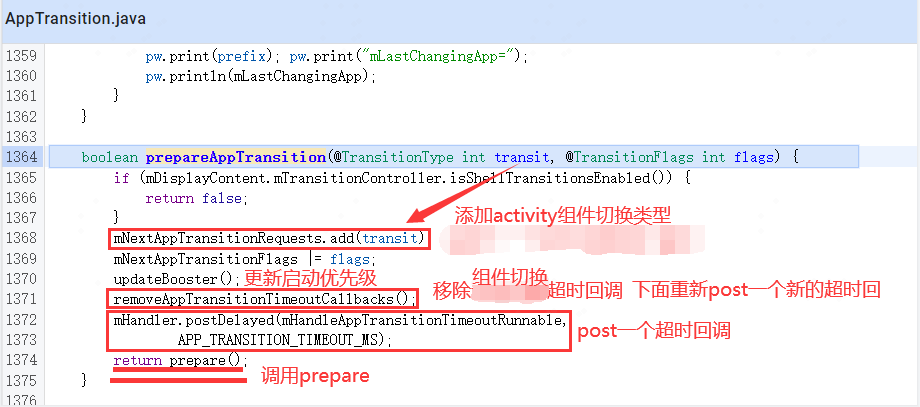
**prepareAppTransition**



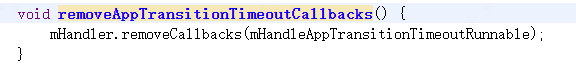
**mAppTransition:**

clipboard.png

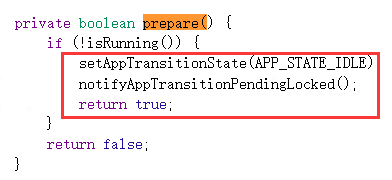
**mAppTransition.prepareAppTransition**



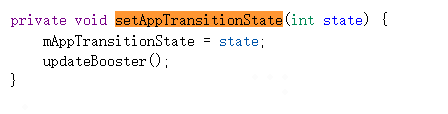
**removeAppTransitionTimeoutCallbacks: 默认是5000毫秒**



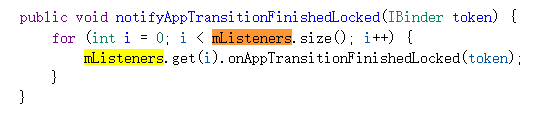
**prepare:如果没有在执行中，就设置状态为APP\_STATE\_IDLE空间状态，唤醒过渡动画等待执行**



**setAppTransitionState：**



**notifyAppTransitionFinishedLocked：**



**mListener是数组存放ApptTransitionListener:**

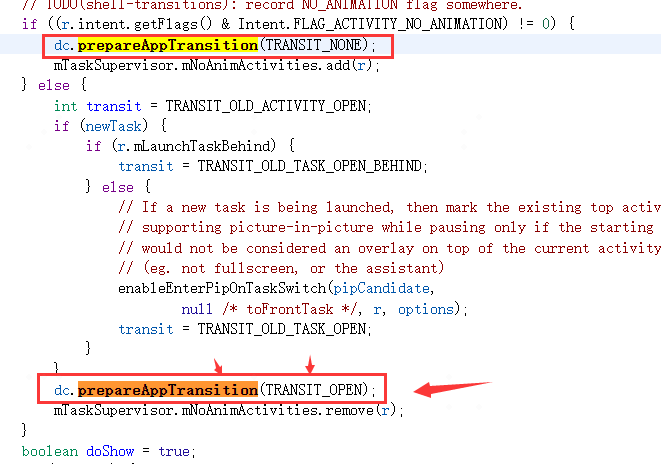
clipboard.png

**最后调用了数组中的监听器的onAppTransitionFinishedLocked**

**09-08 09:42:05.169 1479 5339 I ActivityTaskManager: START u0 {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10200000 cmp=com.wtf.gallery3d/.app.MainActivity bnds=[48,1681][294,1992] (has extras)} from uid 10100**

**09-08 09:42:05.179 1479 5339 V WindowManager: Prepare app transition: transit=TRANSIT\_TASK\_OPEN mNextAppTransition=TRANSIT\_UNSET alwaysKeepCurrent=false displayId=0 Callers=com.android.server.wm.DisplayContent.prepareAppTransition:5168 com.android.server.wm.DisplayContent.prepareAppTransition:5162 com.android.server.wm.ActivityStack.startActivityLocked:2414 com.android.server.wm.ActivityStarter.startActivityInner:2246 com.android.server.wm.ActivityStarter.startActivityUnchecked:1987**

**Prepare app transition日志中的transit是打算要设置的新transit，mNextAppTransition是当前已经存在的transit。**



**该函数功能如下：**

**1）设置mNextAppTransition表明要执行的activity组件切换类型，设置值（非TRANSIT\_UNSET值）后使得AppTransition.isTransitionSet()返回true表面已经设置了AppTransition。**

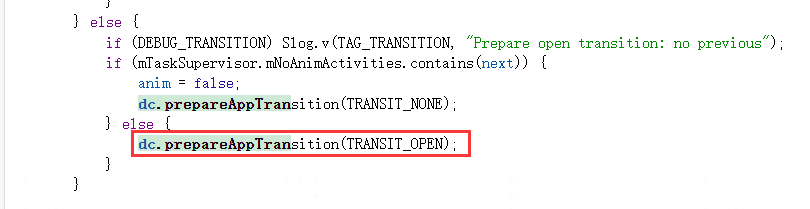
**2）该函数若重复执行，按其内部逻辑存在覆盖和不覆盖两种场景：覆盖场景常见的比如TRANSIT\_TASK\_OPEN替换掉TRANSIT\_TASK\_CLOSE、TRANSIT\_ACTIVITY\_OPEN 替换掉TRANSIT\_ACTIVITY\_CLOSE，也就是常说的open activity和open task的优先级要高于close activity和close task；不覆盖场景比如热启应用复用栈时会出现activityPaused阶段执行到resumeTop去设置TRANSIT\_TASK\_OPEN尝试替换掉startActivity阶段设置的TRANSIT\_TASK\_TO\_FRONT，但不会被替换成功。**

**3）执行AppTransition.prepare()函数设置mAppTransitionState为APP\_STATE\_IDLE，即设置AppTransition的状态为IDLE空闲态，因为只有空闲状态才能执行下一个即将要执行的AppTransition。**

**完成这一步表明App Transition准备完成，但离动画执行还很远**

**其实在activityPaused阶段去resumeTop时仍然会触发一次设置TRANSIT\_TASK\_OPEN，代码和日志如下：**

**resumeTopActivity：**

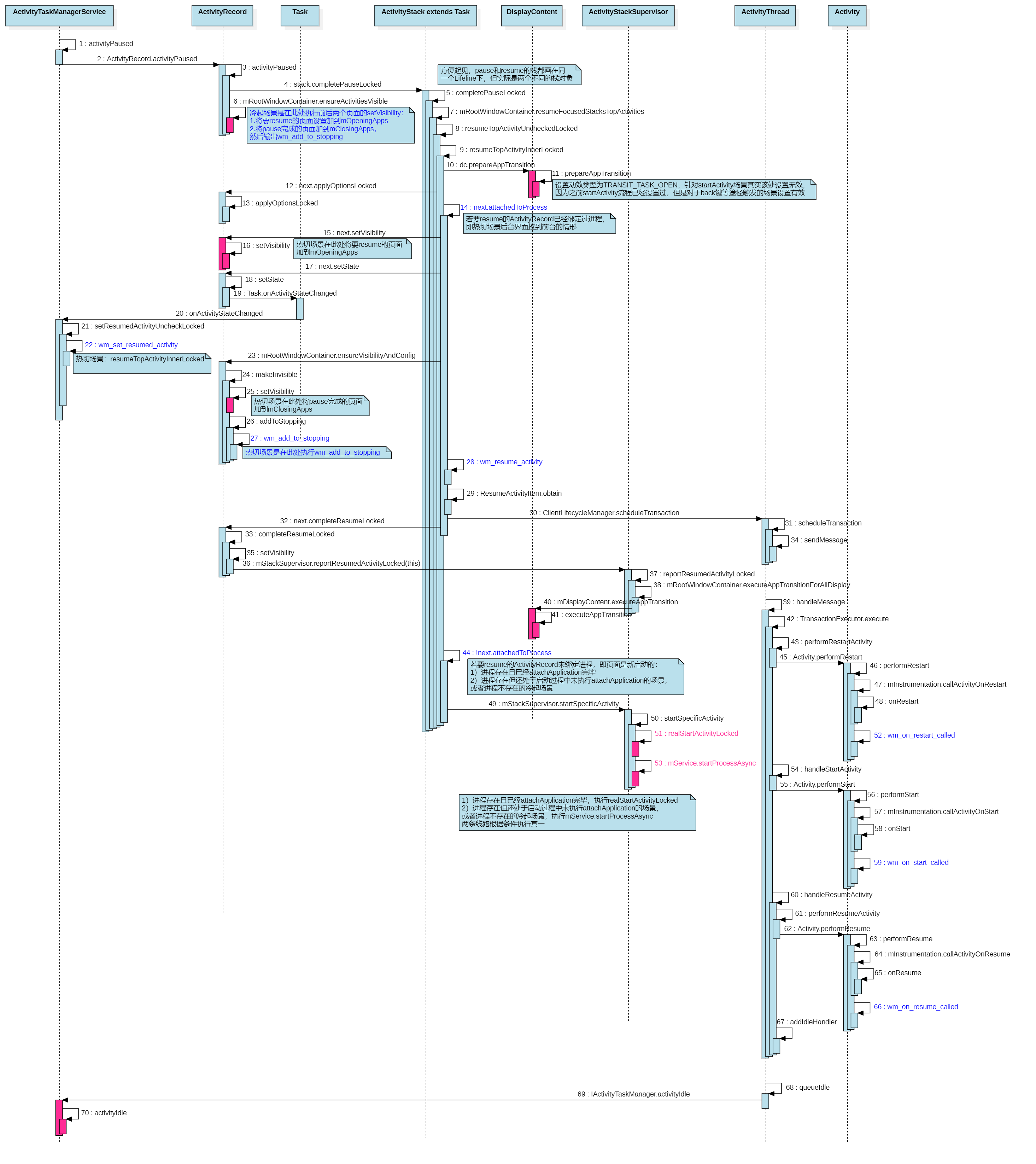


**09-08 09:42:05.191 1479 1495 V WindowManager: Prepare app transition: transit=TRANSIT\_TASK\_OPEN mNextAppTransition=TRANSIT\_TASK\_OPEN alwaysKeepCurrent=false displayId=0 Callers=com.android.server.wm.DisplayContent.prepareAppTransition:5168 com.android.server.wm.DisplayContent.prepareAppTransition:5162 com.android.server.wm.ActivityStack.resumeTopActivityInnerLocked:2059 com.android.server.wm.ActivityStack.resumeTopActivityUncheckedLocked:1710 com.android.server.wm.RootWindowContainer.resumeFocusedStacksTopActivities:2478**

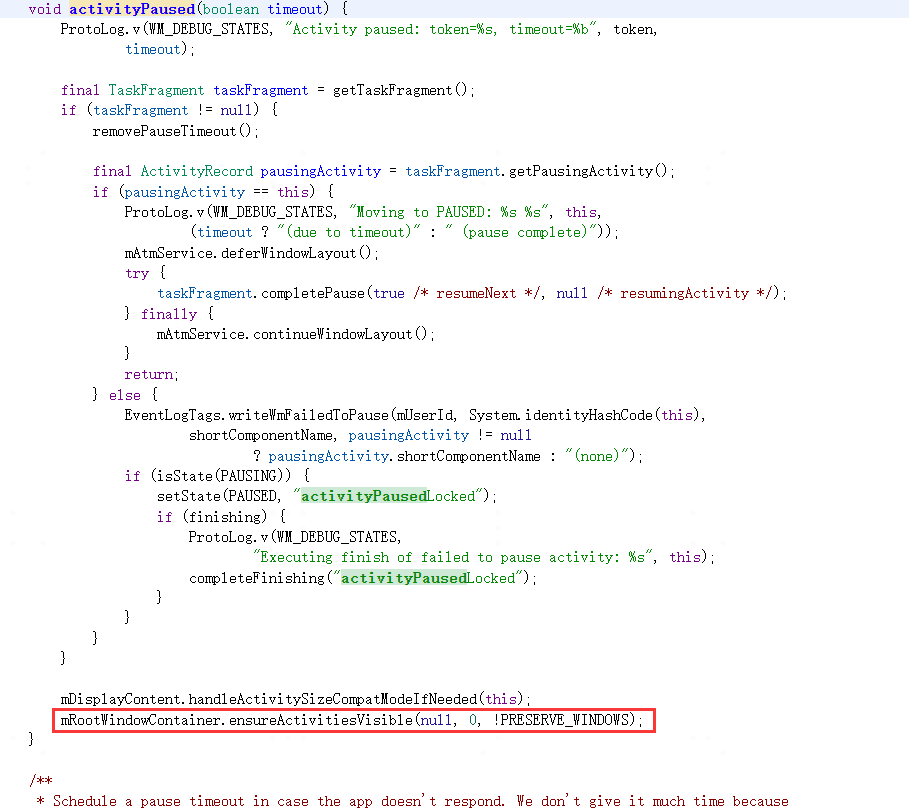
**activityPaused阶段的resumeTop设置动效类型为TRANSIT\_TASK\_OPEN，针对startActivity场景其实该处设置无效，因为之前startActivity流程已经设置过，但是对于back键等途径触发的resumeTop场景的设置仍然有效。**

**2、setVisibility阶段**

**ActivityRecord.setVisibility函数的主要功能是把设置true的ActivityRecord加到DisplayContent.mOpeningApps列表，把设置false的ActivityRecord加到DisplayContent.mClosingApps列表，该函数在每次启动页面（不论热启冷起）的整个过程中可能涉及多次调用，是可重入的，且在执行DisplayContent.executeAppTransition之前都不会真正的commitVisibility（GOOD TO GO阶段才会真正执行），仅仅是填充mOpeningApps和mClosingApps，目的是在GOOD TO GO真正做动效时有目标的退场和入场应用去执行动效。**



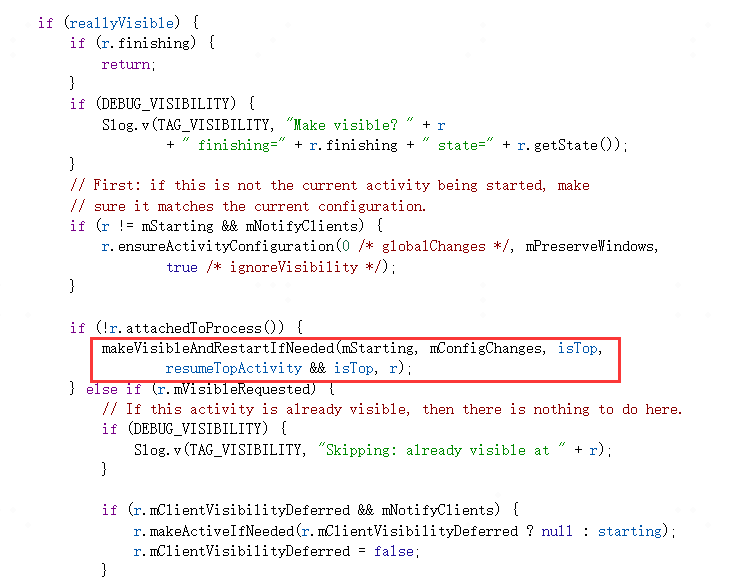
**冷起时首次触发桌面加到mClosingApps、图库加到mOpeningApps是由activityPaused阶段的以下函数触发：**



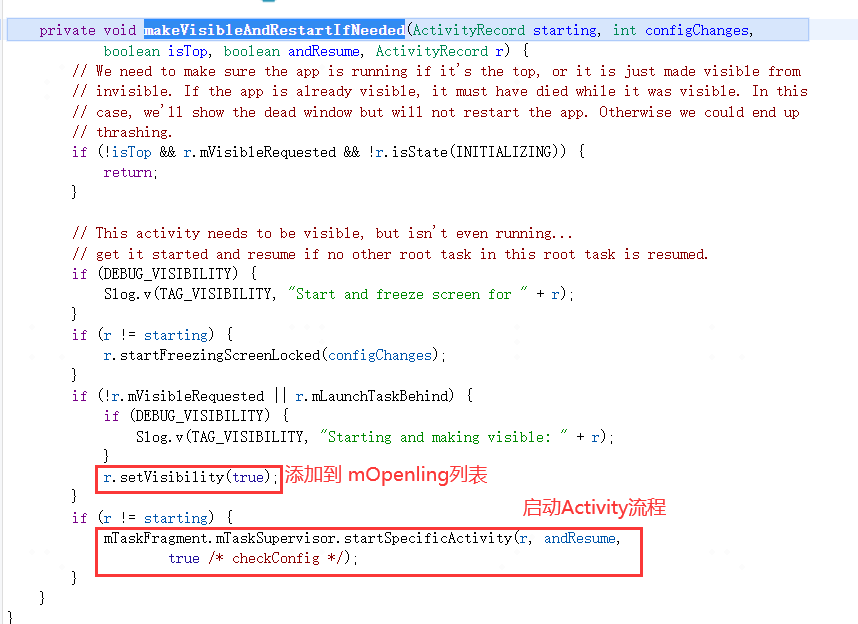
**之前分析应用启动时曾说过，冷起新栈，由completePaused结束的那次ensureActivitiesVisible来触发将桌面makeInvisible进而触发ActivityRecord.setVisibility(false)加到mClosingApps、将图库makeVisibleAndRestartIfNeeded进而触发ActivityRecord.setVisibility(true)加到mOpeningApps。**



**图库ensure执行的分支：**



**makeVisibleAndRestartIfNeeded：**



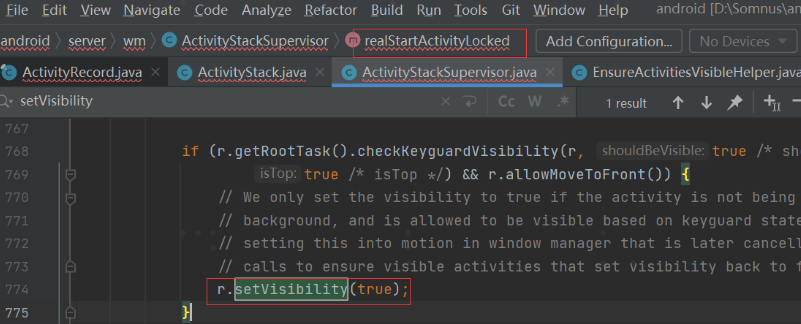
**09-08 09:42:05.192 1479 1495 V WindowManager: setAppVisibility(Token{5817814 ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, visible=true): mNextAppTransition=TRANSIT\_TASK\_OPEN visible=false mVisibleRequested=false Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.EnsureActivitiesVisibleHelper.makeVisibleAndRestartIfNeeded:223 com.android.server.wm.EnsureActivitiesVisibleHelper.setActivityVisibilityState:155 com.android.server.wm.EnsureActivitiesVisibleHelper.lambda$Bbb3nMFa3F8er\_OBuKA7-SpeSKo:0 com.android.server.wm.-$$Lambda$EnsureActivitiesVisibleHelper$Bbb3nMFa3F8er\_OBuKA7-SpeSKo.accept:12 com.android.internal.util.function.pooled.PooledLambdaImpl.doInvoke:307**

**桌面ensure执行的分支：**

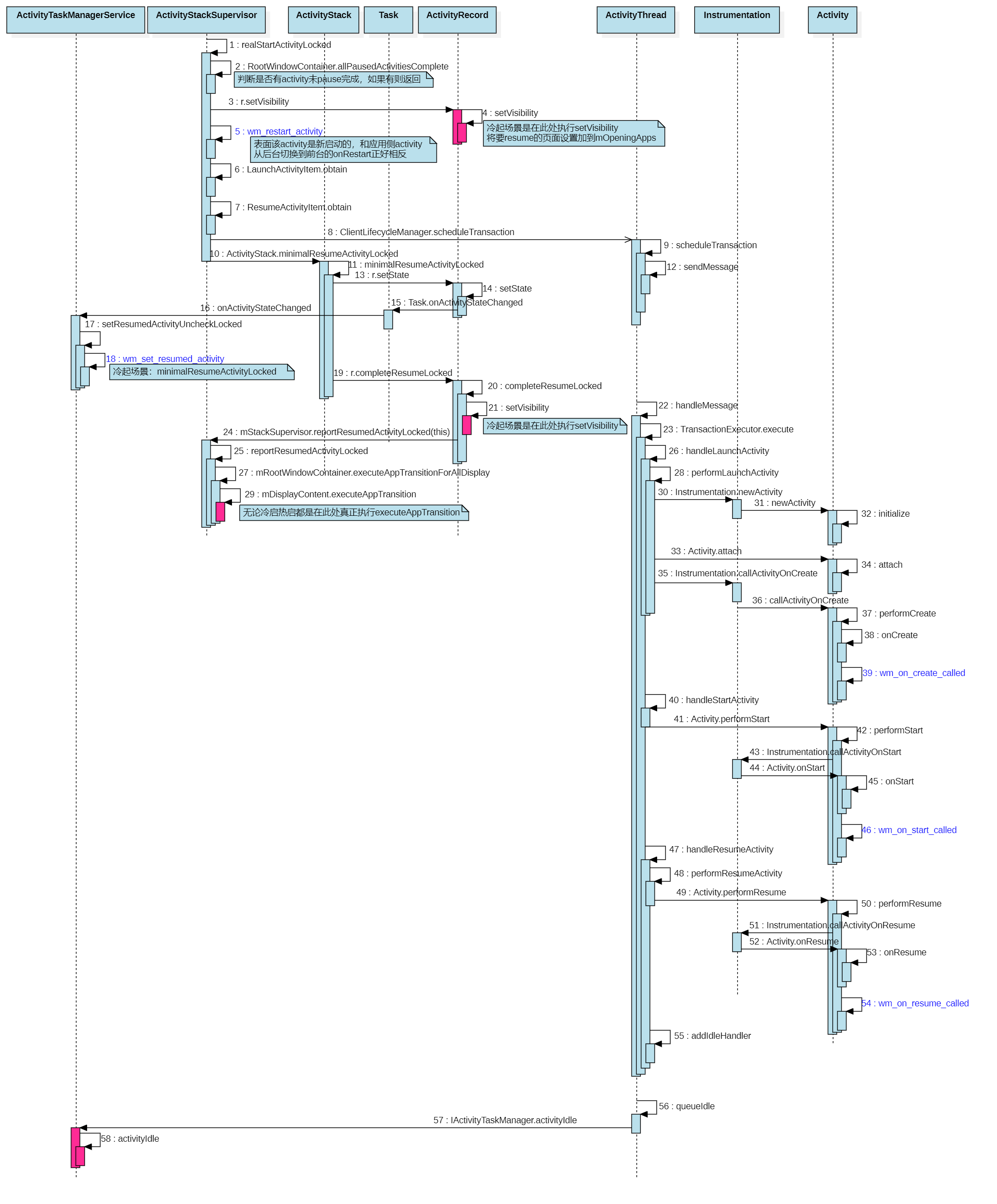


**09-08 09:42:05.192 1479 1495 V WindowManager: setAppVisibility(Token{6d196ad ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}}, visible=false): mNextAppTransition=TRANSIT\_TASK\_OPEN visible=true mVisibleRequested=true Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityRecord.makeInvisible:5168 com.android.server.wm.EnsureActivitiesVisibleHelper.setActivityVisibilityState:182 com.android.server.wm.EnsureActivitiesVisibleHelper.lambda$Bbb3nMFa3F8er\_OBuKA7-SpeSKo:0 com.android.server.wm.-$$Lambda$EnsureActivitiesVisibleHelper$Bbb3nMFa3F8er\_OBuKA7-SpeSKo.accept:12 com.android.internal.util.function.pooled.PooledLambdaImpl.doInvoke:307**

**虽然activityPaused中的completePaused可以最终触发到桌面和图库的ActivityRecord.setVisibility，但由于是冷起栈，realStartActivityLocked阶段仍然可以触发要启动的图库执行一次ActivityRecord.setVisibility(true)，且该次才是最关键的一次，日志如下：**



**09-08 09:42:05.235 1479 4303 V WindowManager: setAppVisibility(Token{5817814 ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, visible=true): mNextAppTransition=TRANSIT\_TASK\_OPEN visible=false mVisibleRequested=true Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityStackSupervisor.realStartActivityLocked:881 com.android.server.wm.RootWindowContainer.startActivityForAttachedApplicationIfNeeded:2139 com.android.server.wm.RootWindowContainer.lambda$5fbF65VSmaJkPHxEhceOGTat7JE:0 com.android.server.wm.-$$Lambda$RootWindowContainer$5fbF65VSmaJkPHxEhceOGTat7JE.apply:8 com.android.internal.util.function.pooled.PooledLambdaImpl.doInvoke:315**



**而无论是冷启还是热启，都会执行ActivityRecord.completeResumeLocked，该函数同样会把要resume的页面执行ActivityRecord.setVisibility(true)，虽然对于冷起场景此次调用属于多余的操作，但是对于热启复用栈场景该处很关键，因为热启场景不会调用realStartActivityLocked但是仍然会调用completeResumeLocked，冷启场景日志如下：**

**09-08 09:42:05.239 1479 4303 V WindowManager: setAppVisibility(Token{5817814 ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, visible=true): mNextAppTransition=TRANSIT\_TASK\_OPEN visible=false mVisibleRequested=true Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityRecord.completeResumeLocked:5373 com.android.server.wm.ActivityStack.minimalResumeActivityLocked:1028 com.android.server.wm.ActivityStackSupervisor.realStartActivityLocked:1020 com.android.server.wm.RootWindowContainer.startActivityForAttachedApplicationIfNeeded:2139 com.android.server.wm.RootWindowContainer.lambda$5fbF65VSmaJkPHxEhceOGTat7JE:0**

**3、executeAppTransition阶段**

**无论是冷启还是热启，都是由ActivityRecord.completeResumeLocked阶段触发的DisplayContent.executeAppTransition来最终执行AppTransition，当然此处仅是执行AppTransition，并不代表能一定执行动效，因为动效的执行需要依赖mOpeningApps中的应用绘制完成才能够执行。DisplayContent.executeAppTransition功能如下：**

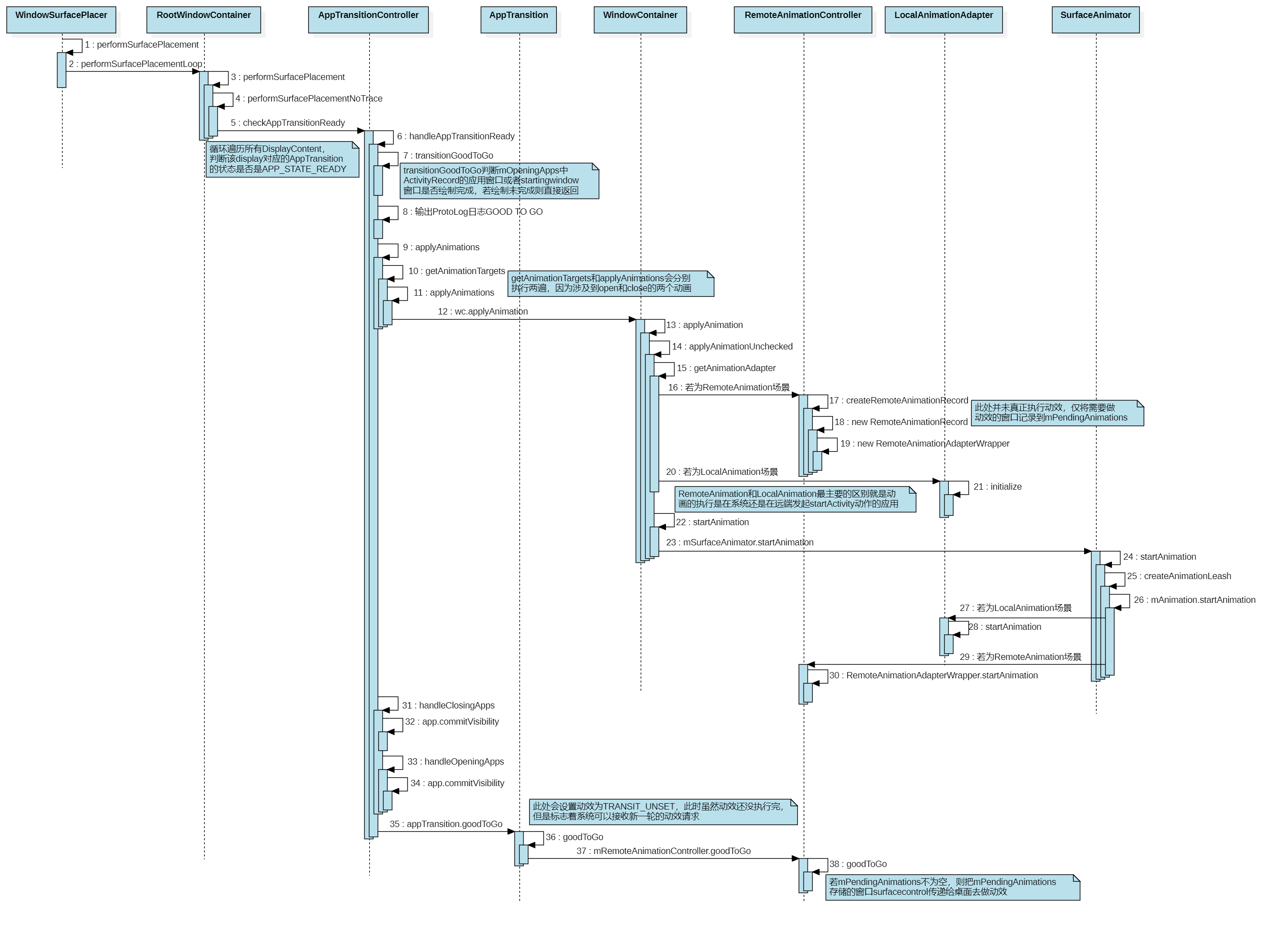
**1）首先调用AppTransition.isTransitionSet()判断是否有设置transit，若未设置则该函数不产生任何作用。**

**2）调用AppTransition.setReady()函数将mAppTransitionState从APP\_STATE\_IDLE状态切换到APP\_STATE\_READY，只有ready之后才有可能触发GOOD TO GO流程，ready之前就算窗口绘制完毕（假设有此异常流程）也无法执行到GOOD TO GO。**

**09-08 09:42:05.240 1479 4303 W WindowManager: Execute app transition: mNextAppTransition=TRANSIT\_TASK\_OPEN, displayId: 0 Callers=com.android.server.wm.RootWindowContainer.executeAppTransitionForAllDisplay:2399 com.android.server.wm.ActivityStackSupervisor.reportResumedActivityLocked:2092 com.android.server.wm.ActivityRecord.completeResumeLocked:5402 com.android.server.wm.ActivityStack.minimalResumeActivityLocked:1028 com.android.server.wm.ActivityStackSupervisor.realStartActivityLocked:1020**

**4、GOOD TO GO阶段**

**执行完executeAppTransition并不一定立马就能触发动效的执行，需要等要打开的ActivityRecord窗口绘制完成（有starting window的只需要starting window绘制完成，不需要等到主窗口绘制完成），才能触发GOOD TO GO真正执行动画。GOOD TO GO执行动画流程如下：**

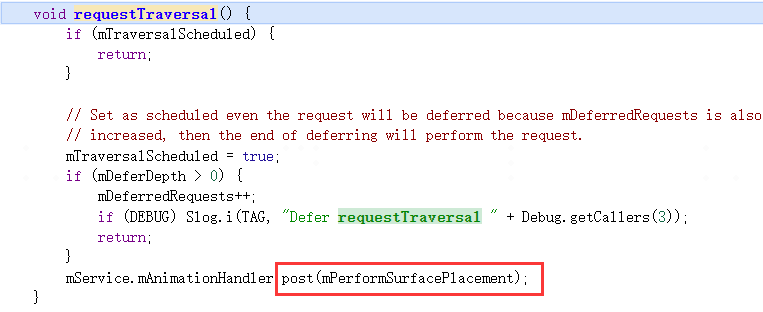


**在wms每次触发performSurfacePlacement刷新surface时，都会尝试检查下DisplayContent是否有设置了动画且满足Ready状态、要做动画的应用是否绘制完成，如果这些条件都满足则进入GOOD TO GO阶段去apply动效真正执行动效。**

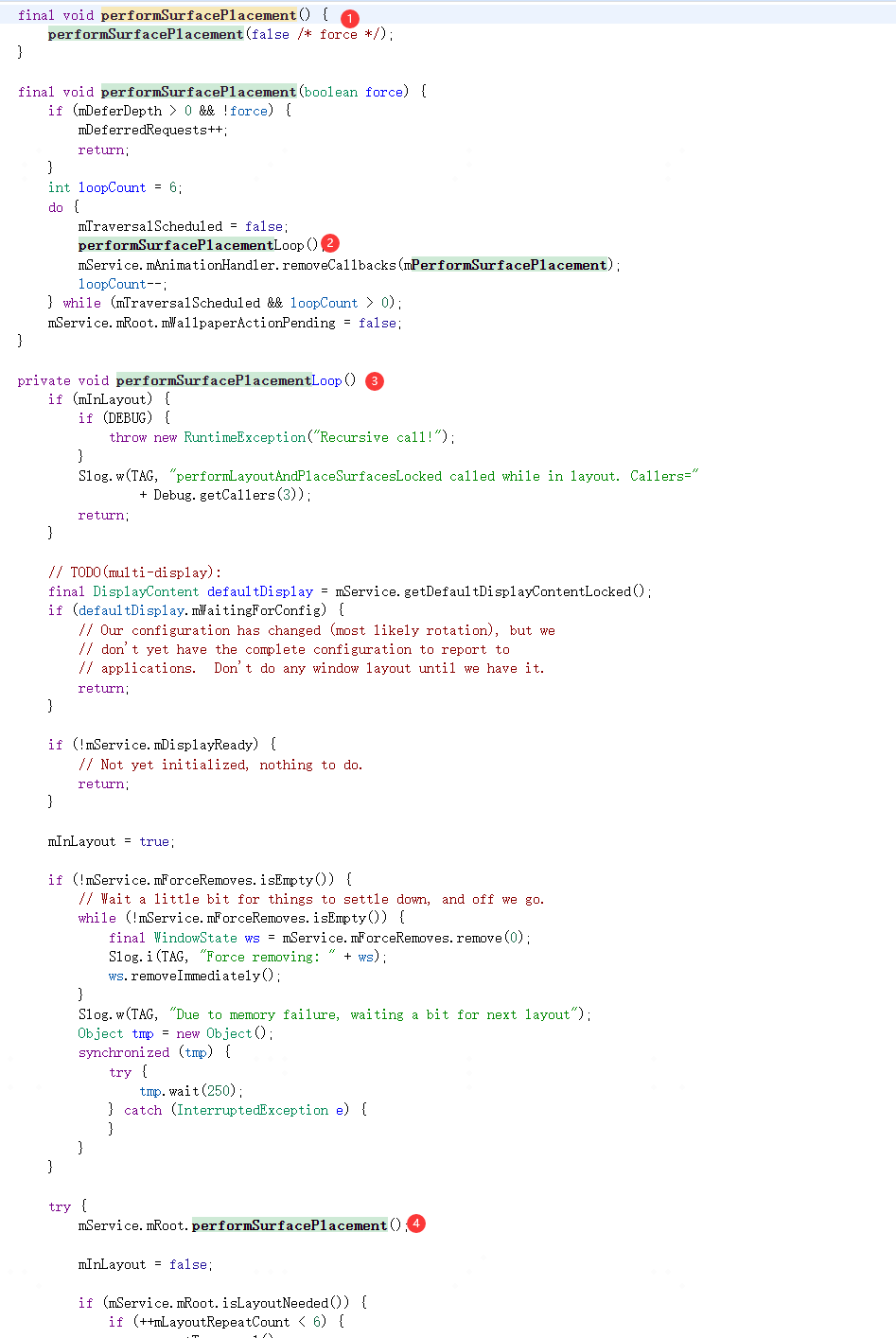
**RootWindowContainer.checkAppTransitionReady()函数在每次尝试刷新时都会调用到，函数内部会判断DisplayContent上的AppTransition.isReady()即mAppTransitionState状态是APP\_STATE\_READY这一条件是否满足，若满足则说明已经设定了切换动效，且已经执行了executeAppTransition进入了ready阶段：**



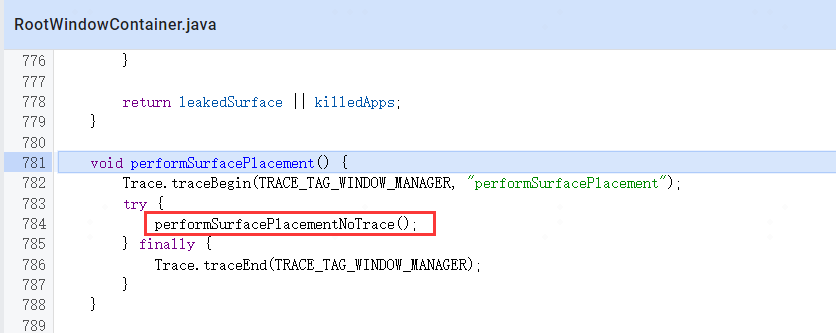
**requestTraversal：**



**performSurfacePlacement：**



**RootWindowContainer.performSurfacePlacement：**



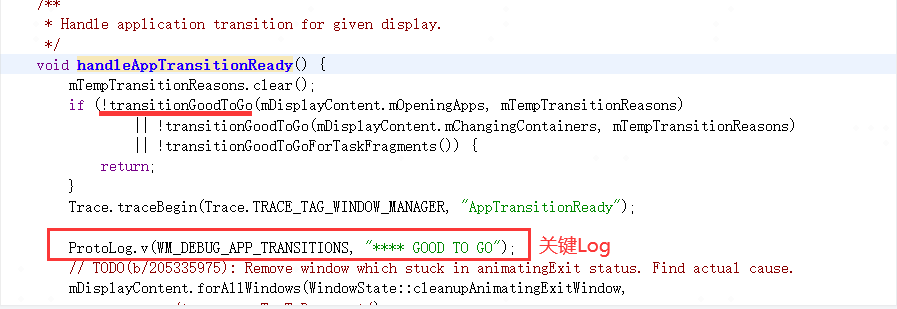
**performSurfacePlacementNoTrace：**



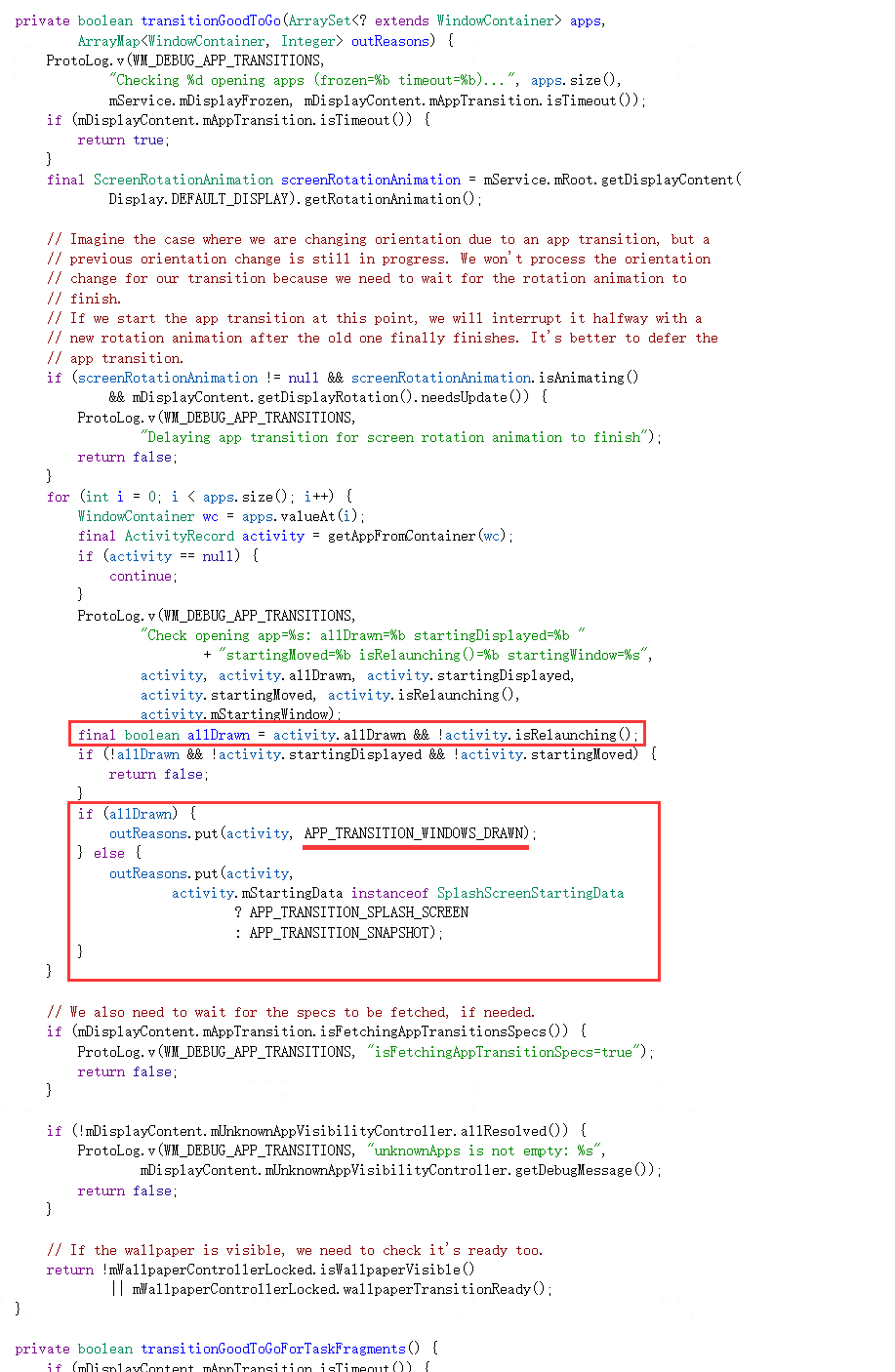
**checkAppTransitionReady：**



**在满足ready后，继续执行AppTransitionController.handleAppTransitionReady()，该函数一上来就调用transitionGoodToGo判断mOpeningApps中的页面窗口是否都已绘制完成（有starting window的只需要starting window绘制完成，不需要等到主窗口绘制完成），若没有绘制完成则直接返回表示无法执行动画，因为窗口还没准备好；另外，transitionGoodToGo还判断了mChangingContainers中窗口是否绘制完成，这个场景是前台的悬浮窗跟全屏之间进行切换时才会遇到，在后续章节再继续分析。**



**transitionGoodToGo：**



**09-08 09:42:05.246 1479 1539 V WindowManager: Check opening app=ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}: allDrawn=false startingDisplayed=true startingMoved=false isRelaunching()=false startingWindow=Window{e599a7d u0 Splash Screen com.wtf.gallery3d}**

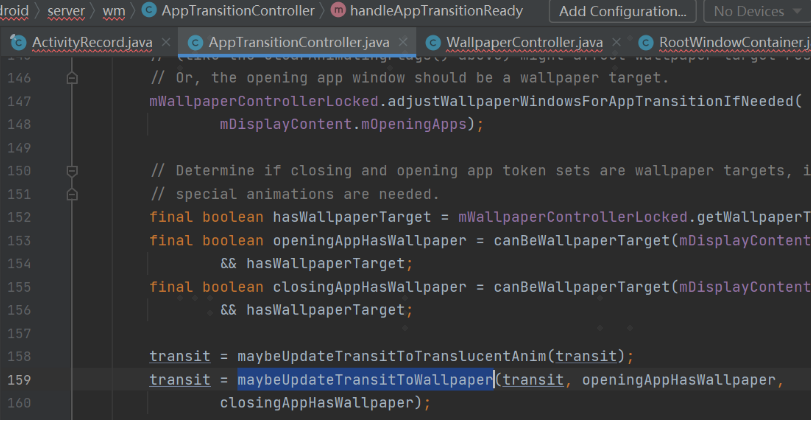
**桌面打开应用就属于有startingwindow绘制完成但主窗口并未绘制完毕的情形。**

**transitionGoodToGo满足条件后，紧接着会输出关键的一行标志性日志：**

**09-08 09:42:05.246 1479 1539 V WindowManager: \*\*\*\* GOOD TO GO（handleAppTransitionReady打印）**

**输出该日志代表马上要执行最终的动效流程了。**

**在执行动效之前transit还有可能被再度修改一次，比如桌面打开应用场景由于mWallpaperTarget要等到ActivityRecord.onAnimationFinished才能触发，所以在此阶段mWallpaperTarget仍然是桌面，这导致maybeUpdateTransitToWallpaper函数会把TRANSIT\_TASK\_OPEN或者TRANSIT\_TASK\_TO\_FRONT更改为TRANSIT\_WALLPAPER\_CLOSE，但是这个并不影响桌面打开应用的动效执行，因为桌面打开动效用的是RemoteAnimation，这个往下继续分析会讲到，如果是LocalAnimation，这个transit的变化会影响到最后动效adapter的选择进而最终影响到动画的类型。**

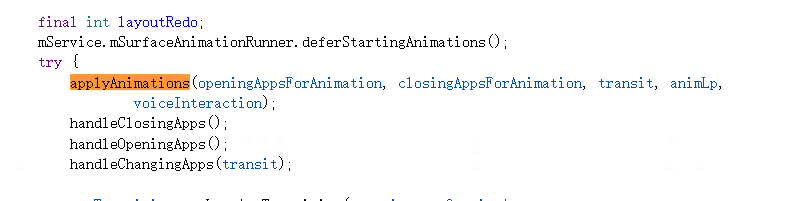


**09-08 09:42:05.246 1479 1539 V WindowManager: New wallpaper target=Window{8a35853 u0 com.wtf.launcher/com.wtf.launcher.Launcher}, oldWallpaper=Window{8a35853 u0 com.wtf.launcher/com.wtf.launcher.Launcher}, openingApps={ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, closingApps={ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}}**

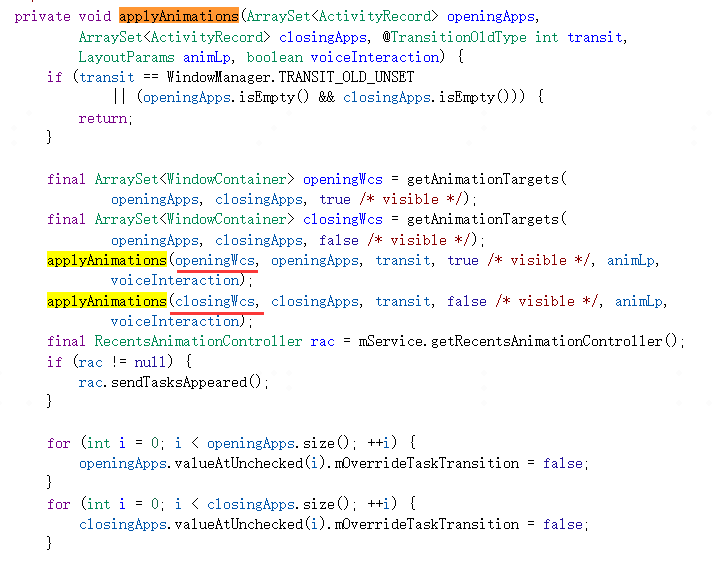
**09-08 09:42:05.246 1479 1539 V WindowManager: New transit away from wallpaper: TRANSIT\_WALLPAPER\_CLOSE**

**在做完transit可能存在的变更之后，进入正题applyAnimations，该函数中有个关键函数getAnimationTargets，该函数主要功能就是用来提升动效执行的目标对象层级，比如从桌面打开app这种mOpeningApps和mClosingApps中的ActivityRecord分属不同的Task，则本来执行动效的对象是ActivityRecord，我们可以提升到以各自的Task去做动画，即动效的执行节点从ActivityRecord变成Task。通过执行两次getAnimationTargets分别拿到真正要执行open和close的动效目标对象也就是Task，接下来去以Task作为openingWcs和closingWcs去做动效，**

**接着看handleAppTransitionReady方法里面 会调用到applyAnimations：**



**applyAnimations：**



**09-08 09:42:05.246 1479 1539 V WindowManager: Changing app ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931} visible=false performLayout=false**

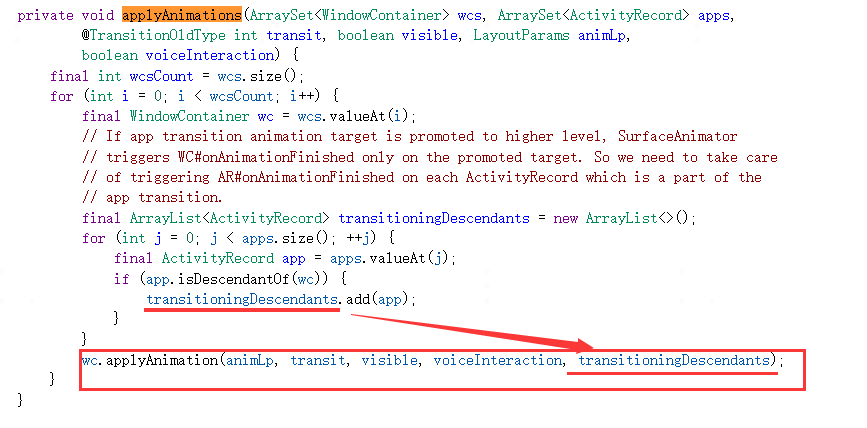
**09-08 09:42:05.246 1479 1539 V WindowManager: getAnimationTarget in={ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, out={Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}}**

**09-08 09:42:05.246 1479 1539 V WindowManager: Changing app ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815} visible=true performLayout=false**

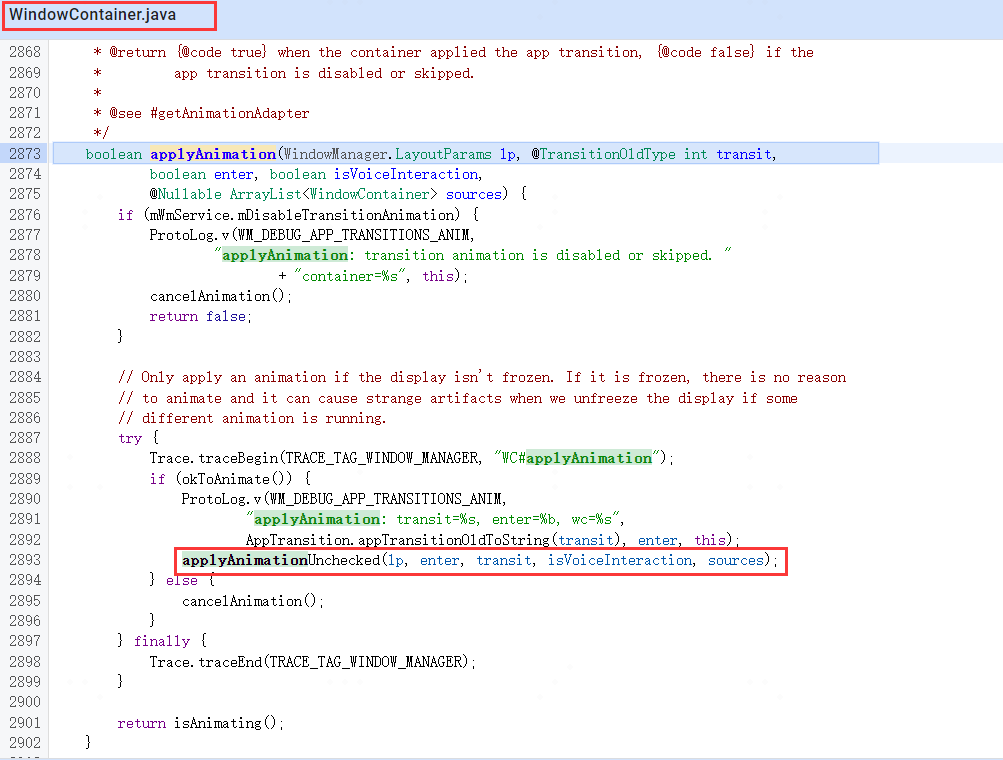
**09-08 09:42:05.246 1479 1539 V WindowManager: getAnimationTarget in={ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}}, out={Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}}**

**然后以openingWcs和closingWcs中的Task对象去applyAnimation**

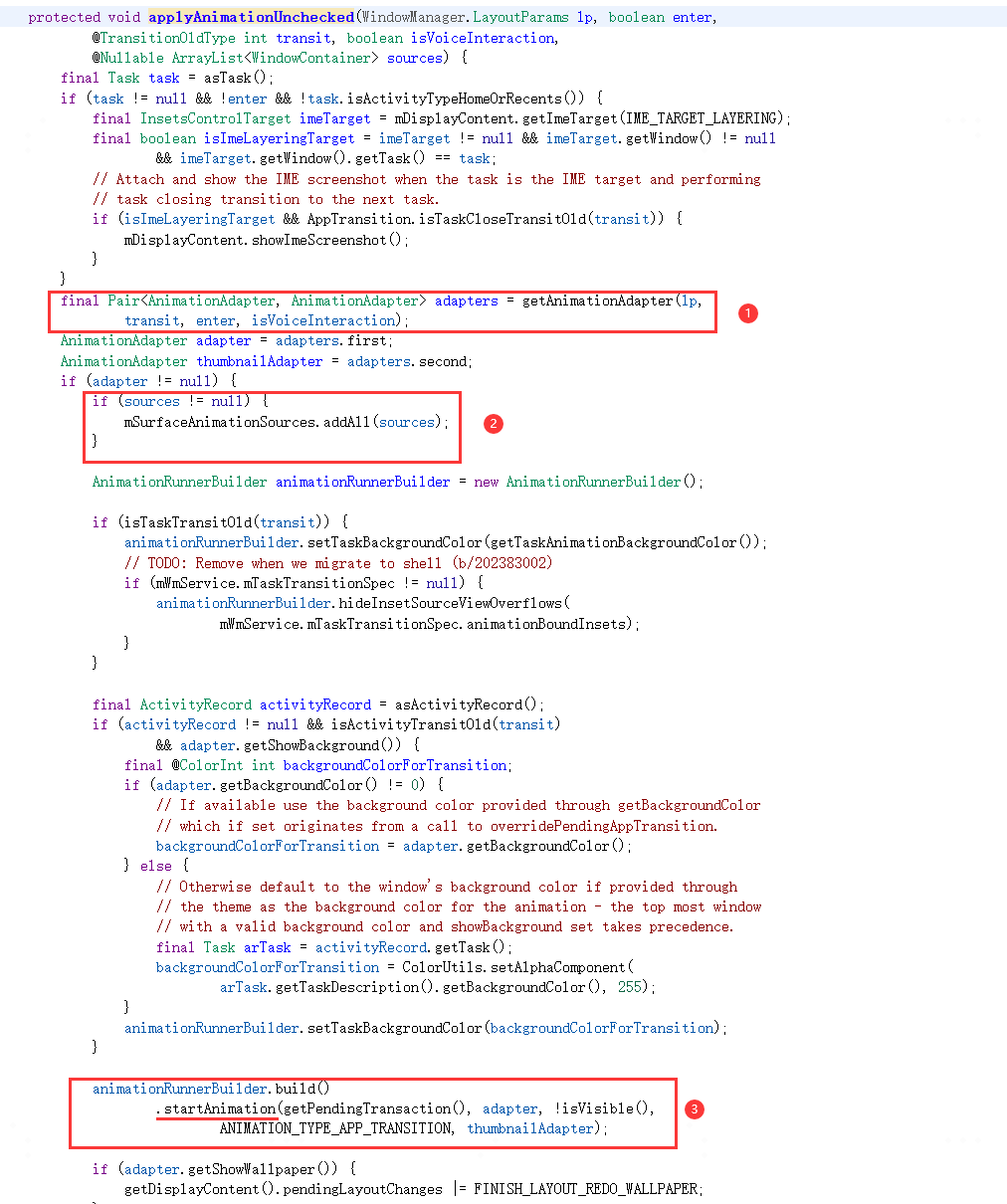
**applyAnimations：**



**applyAnimations：**



**applyAnimationUnchecked：**

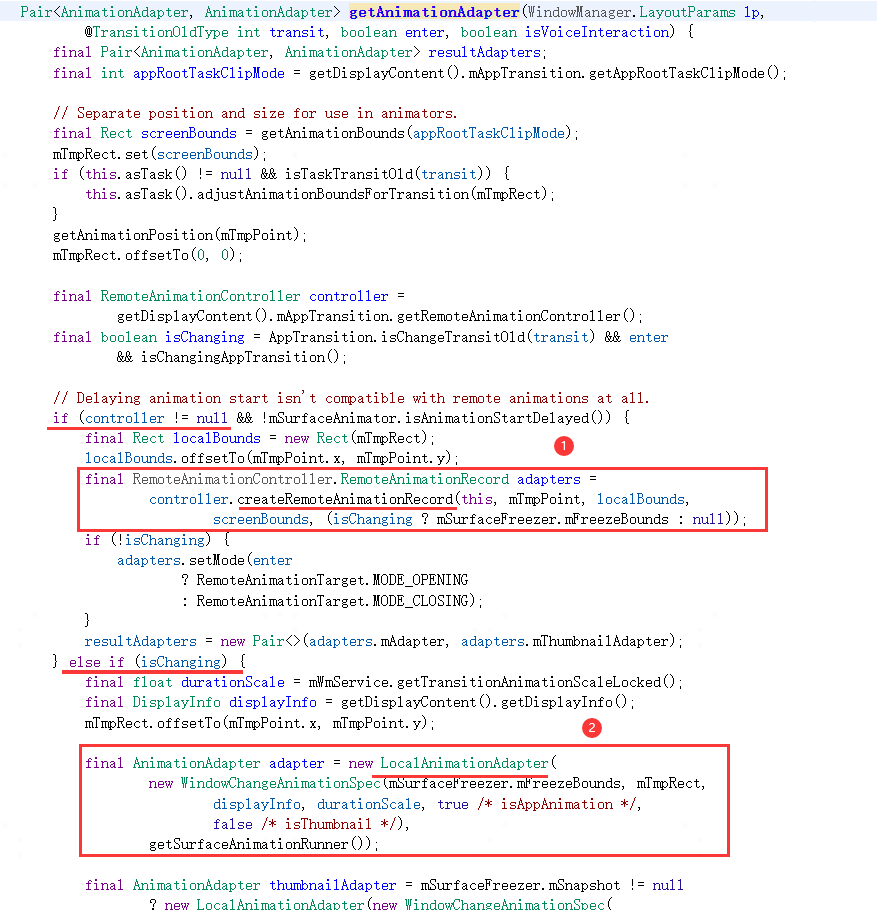


**getAnimationAdapter先根据transit获取一个AnimationAdapter，以便动画执行需要用到。该函数会先判断是否是RemoteAnimation场景，桌面打开应用的动效就是RemoteAnimation，此时系统会构造出来RemoteAnimationController，如果这个对象非空代表是RemoteAnimation，此时会构造RemoteAnimationAdapterWrapper，这种adapter其实最终不会在系统端执行，而是一步步调度到应用端也就是桌面程序，在创建RemoteAnimationAdapterWrapper后会同时会把要做动效的Task加到RemoteAnimationController.mPendingAnimations。如果不是RemoteAnimation则是LocalAnimation，这种使用的LocalAnimationAdapter运行在系统侧。**

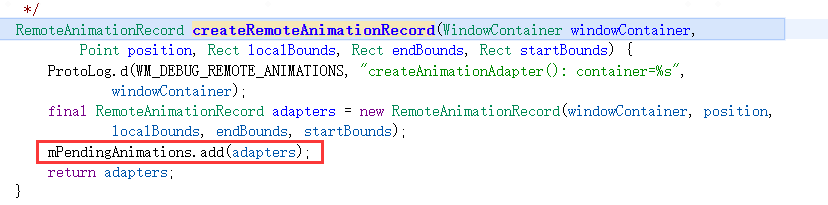
**一般过渡动画有在system\_server进程中执行的本地动画和在例如systemui等进程执行的远程动画两种情况。**

**常见的本地动画：App内切换Activity等。**

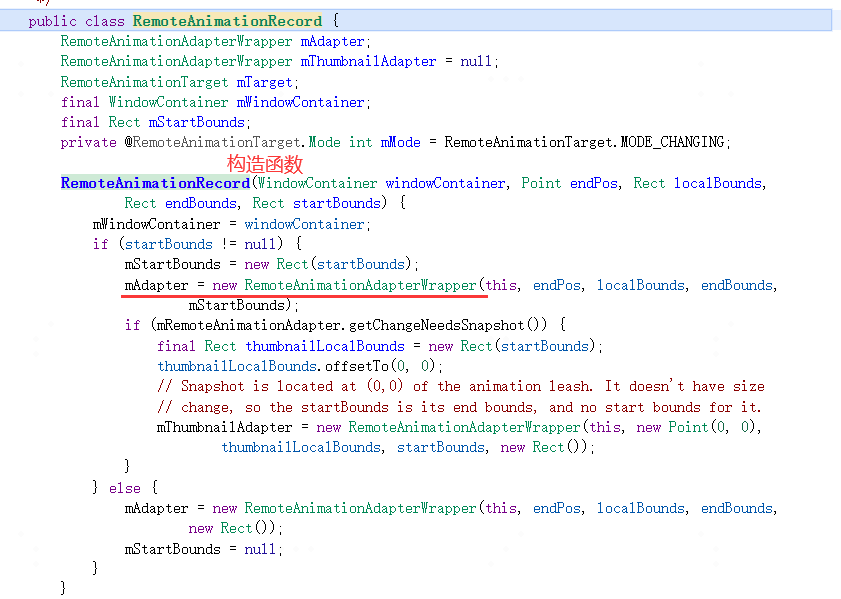
**常见的远程动画：从桌面进入App，从App回到桌面，从Notification悬浮窗进入Activity等。**



**createRemoteAnimationRecord：**



**RemoteAnimationRecord ：**

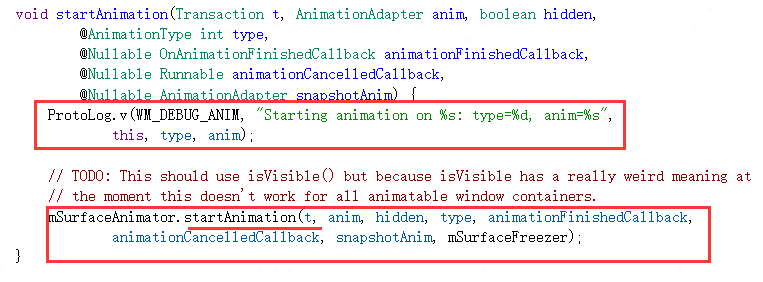


**获取到adapter后紧接着会执行填充mSurfaceAnimationSources，填充的来源是mOpeningApps和mClosingApps，前面我们说过经历提升之前提及的getAnimationTargets动作之后，用来执行动效的ActivityRecord以及被提升为Task，那么执行动效的对象是Task，此时我们执行结束动效执行的onAnimationFinished也是执行在Task（Task、ActivityRecord等窗口视图数据结构都继承自WindowContainer）上，那么我们还是需要再执行下ActivityRecord的onAnimationFinished，所以mSurfaceAnimationSources就是用来存储ActivityRecord对象以便在Task动效做完时能执行到要打开和关闭的ActivityRecord的onAnimationFinished。**

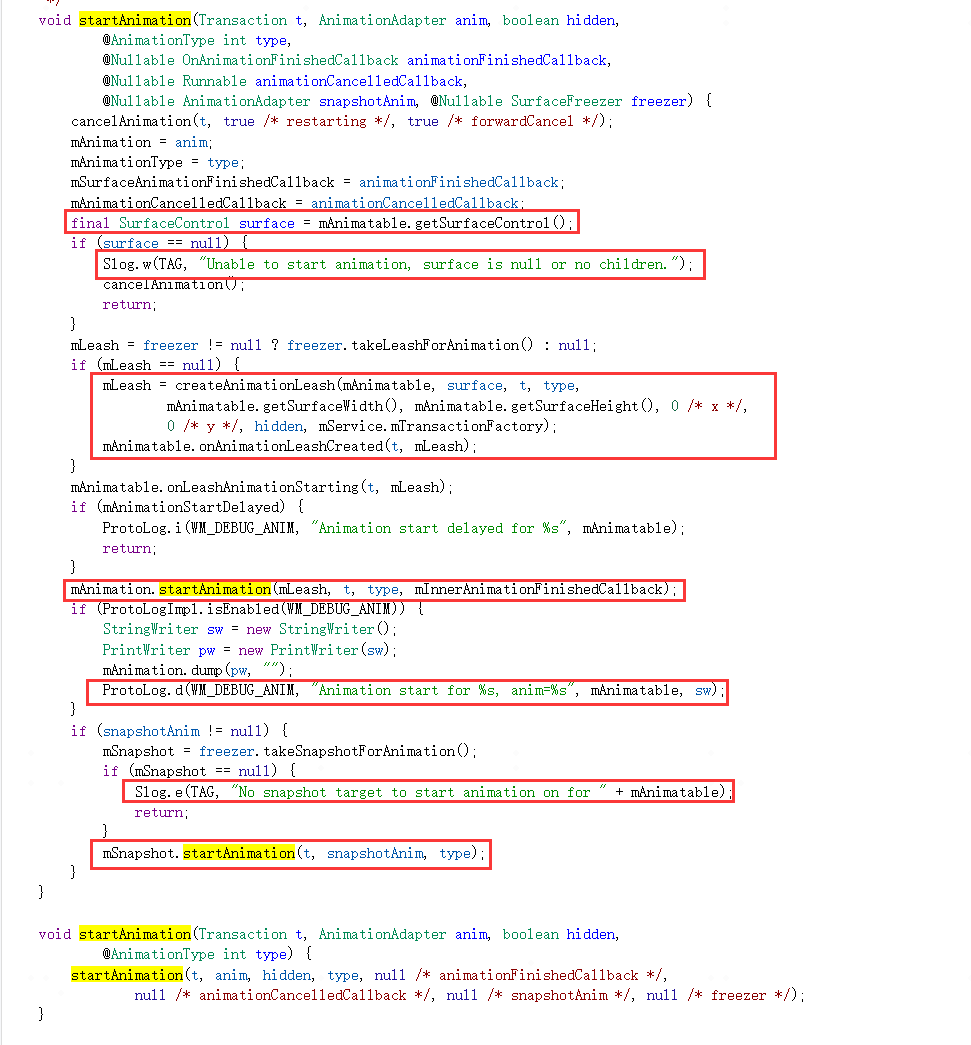
**填充完mSurfaceAnimationSources后，就会真正执行下WindowContainer.startAnimation去触发动效的执行SurfaceAnimator.startAnimation，SurfaceAnimator.startAnimation中会创建动效所需的leash，然后把Task挂到该leash图层下，然后去使用前面获取到的adapter去执行动效。**

**LocalAnimationAdaper直接开始执行动画：**

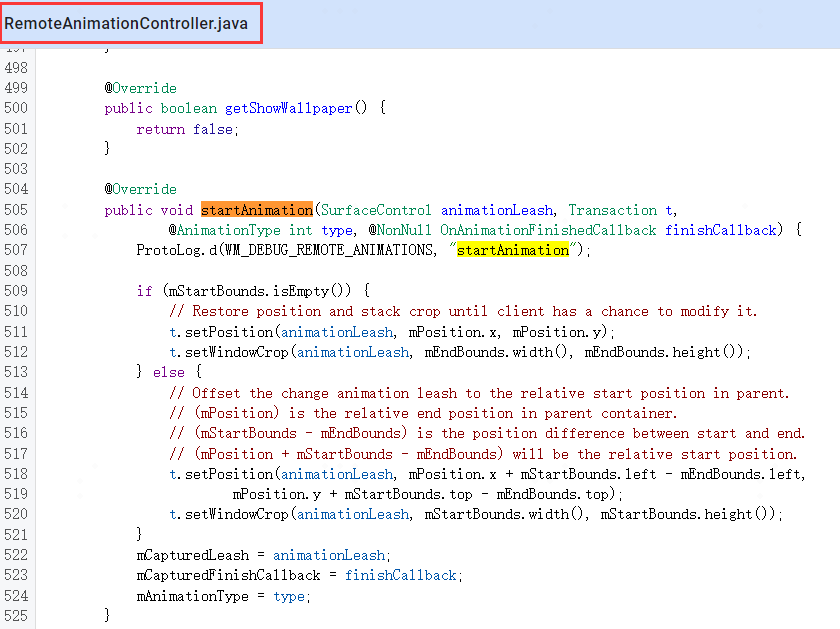
**startAnimation**



**SurfaceAnimator.startAnimation：**



**RemoteAnimationAdaper执行startAnimation则只是做些最终通知远程端做动画之前的最后一项初始化动作：**



**09-08 09:42:05.246 1479 1539 D WindowManager: createAnimationAdapter(): container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}**

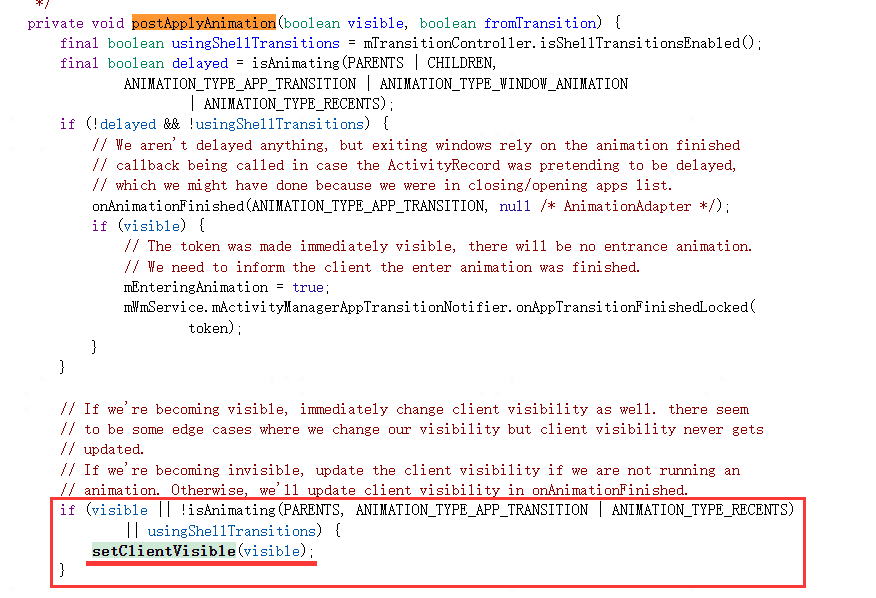
**09-08 09:42:05.248 1479 1539 D WindowManager: startAnimation**

**09-08 09:42:05.248 1479 1539 D WindowManager: createAnimationAdapter(): container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}**

**09-08 09:42:05.250 1479 1539 D WindowManager: startAnimation**

**AppTransitionController.handleAppTransitionReady中执行完applyAnimations后，紧接着会执行handleClosingApps()和handleOpeningApps()，这两个函数主要是把open和close的ActivityRecord执行到ActivityRecord.commitVisibility去设置ActivityRecord.mVisible，以及调用ActivityRecord.postApplyAnimation函数去尝试调用setClientVisible去sendAppVisibilityToClients发送窗口可见性变化到应用端，但是此处需要注意的是：open的ActivityRecord才会立马执行setClientVisible为true，close则会延缓执行setClientVisible为false，因为close的ActivityRecord还需要执行关闭动画，所以要等到ActivityRecord.onAnimationFinished。其实就算是open的ActivityRecord也不是等到这一次才去setClientVisible为true，而是在更早的时刻：1）冷启则构造ActivityRecord就是mClientVisible为true；2）不管冷启还是热启，执行ActivityRecord.setVisibility是都会把要open的ActivityRecord设置setClientVisible为true，当然要close不会执行任何动作仍然保持mClientVisible为上一次在前台时的true直到ActivityRecord.onAnimationFinished。**

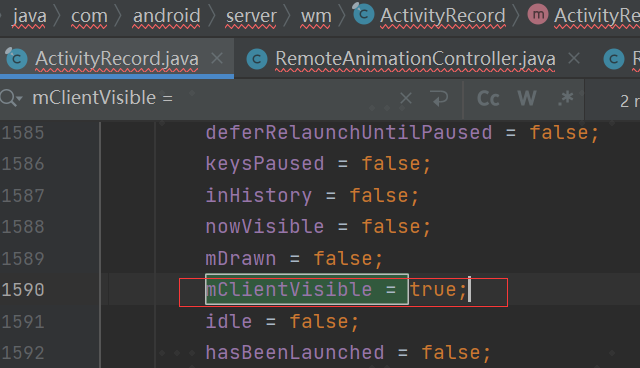
**ActivityRecord.postApplyAnimation函数去尝试调用setClientVisible：**



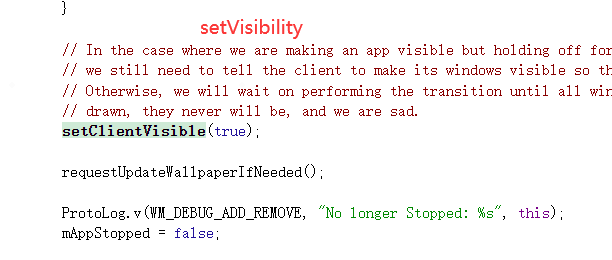
**ActivityRecord.onAnimationFinished函数去尝试调用setClientVisible：**



**冷启构造ActivityRecord设置mClientVisible为true：**

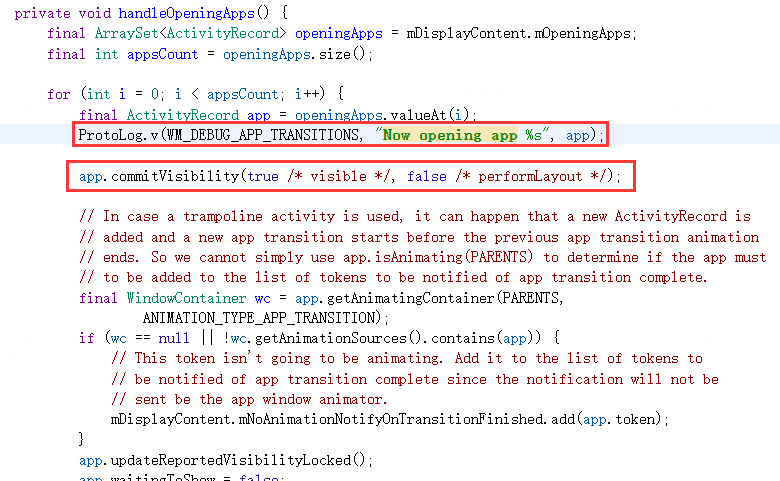


**冷启或者热启执行ActivityRecord.setVisibility把要open的ActivityRecord设置setClientVisible为true：**

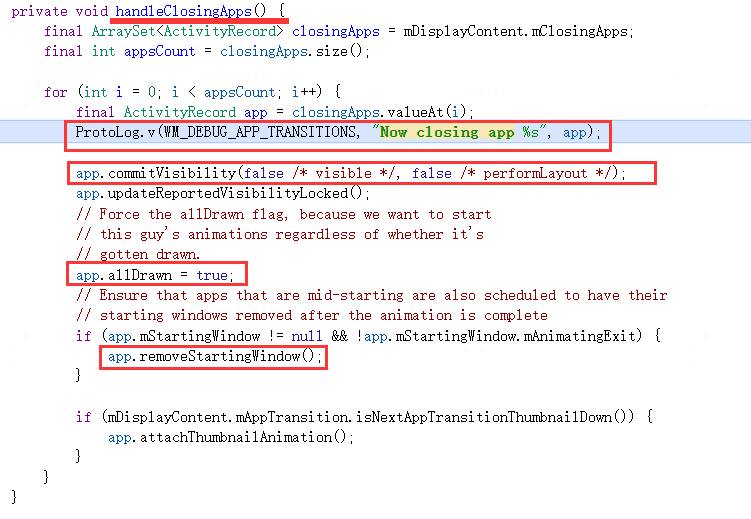


**这个可见性setClientVisible的真正变更的逻辑总结下来就是：open的ActivityRecord在startActivity阶段去构造时或者setVisibility阶段就可以设置true，close的ActivityRecord要等到onAnimationFinished设置为false。**

**handleOpeningApps():**



**handleClosingApps():**



**Log:**

**09-08 09:42:05.251 1479 1539 V WindowManager: Now closing app ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}**

**09-08 09:42:05.251 1479 1539 V WindowManager: commitVisibility: ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}: visible=false mVisibleRequested=false**

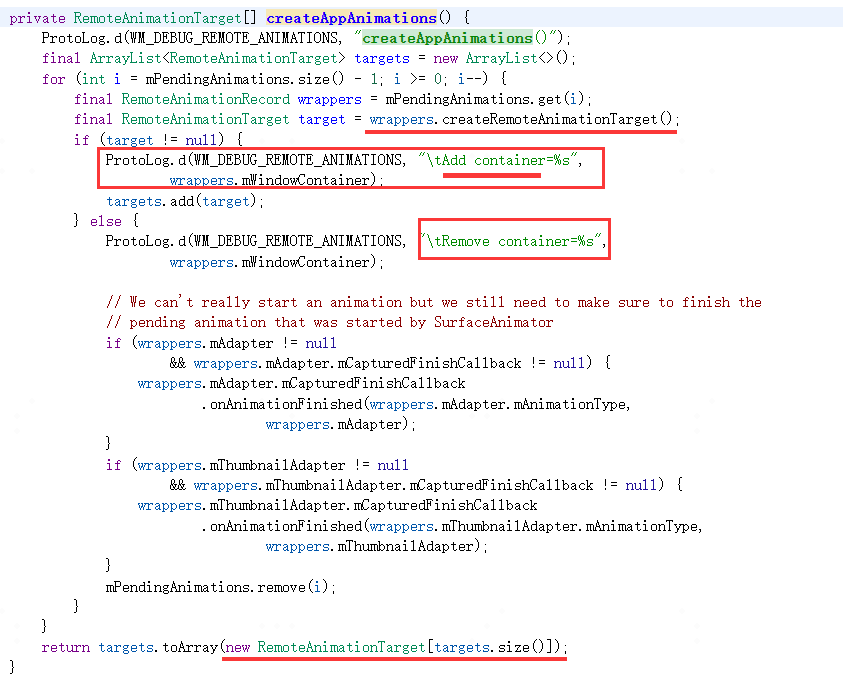
**09-08 09:42:05.251 1479 1539 V WindowManager: Now opening app ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}**

**09-08 09:42:05.251 1479 1539 V WindowManager: commitVisibility: ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}: visible=true mVisibleRequested=true**

**在GOOD TO GO阶段的最后，需要执行AppTransition.goodToGo()，去设置动效为TRANSIT\_UNSET，此时虽然动效还没执行完毕，但是已经标志着系统可以接收新一轮的动效请求。如果是RemoteAnimation，则前面startAnimation仅是初始化了动效的一些参数，真正的触发执行是由AppTransition.goodToGo调度RemoteAnimationController.goodToGo()去把mPendingAnimations存储的窗口surfacecontrol传递给桌面去真正做动效。onAnimationStart就是通过aidl调度的远端的桌面程序进行托管RemoteAnimation动画的执行。**



**createAppAnimations:**



09-08 09:42:05.252 1479 1539 D WindowManager: goodToGo()

09-08 09:42:05.252 1479 1539 D WindowManager: createAppAnimations()

09-08 09:42:05.252 1479 1539 D WindowManager: Add container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}

09-08 09:42:05.252 1479 1539 D WindowManager: Add container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}

09-08 09:42:05.262 1479 1539 D WindowManager: startAnimation(): Notify animation start:

09-08 09:42:05.262 1479 1539 I WindowManager: Starting remote animation

09-08 09:42:05.263 1479 1539 I WindowManager: container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}

09-08 09:42:05.263 1479 1539 I WindowManager: Target:

09-08 09:42:05.263 1479 1539 I WindowManager: mode=1 taskId=5815 isTranslucent=false clipRect=[0,0][1080,2412] contentInsets=[0,96][0,0] prefixOrderIndex=43 position=[0,0] sourceContainerBounds=[0,0][1080,2412] screenSpaceBounds=[0,0][1080,2412] localBounds=[0,0][1080,2412]

09-08 09:42:05.263 1479 1539 I WindowManager: windowConfiguration={ mBounds=Rect(0, 0 - 1080, 2412) mAppBounds=Rect(0, 96 - 1080, 2412) mWindowingMode=fullscreen mDisplayWindowingMode=fullscreen mActivityType=home mAlwaysOnTop=undefined mRotation=ROTATION\_0}

09-08 09:42:05.263 1479 1539 I WindowManager: leash=Surface(name=Surface(name=Task=5815)/@0x72710c8 - animation-leash)/@0xdab6e82

09-08 09:42:05.263 1479 1539 I WindowManager: container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}

09-08 09:42:05.263 1479 1539 I WindowManager: Target:

09-08 09:42:05.263 1479 1539 I WindowManager: mode=0 taskId=5931 isTranslucent=false clipRect=[0,0][1080,2412] contentInsets=[0,0][0,0] prefixOrderIndex=47 position=[0,0] sourceContainerBounds=[0,0][1080,2412] screenSpaceBounds=[0,0][1080,2412] localBounds=[0,0][1080,2412]

09-08 09:42:05.263 1479 1539 I WindowManager: windowConfiguration={ mBounds=Rect(0, 0 - 1080, 2412) mAppBounds=Rect(0, 96 - 1080, 2412) mWindowingMode=fullscreen mDisplayWindowingMode=fullscreen mActivityType=standard mAlwaysOnTop=undefined mRotation=ROTATION\_0}

09-08 09:42:05.263 1479 1539 I WindowManager: leash=Surface(name=Surface(name=Task=5931)/@0xed53246 - animation-leash)/@0x835d993

动效结束桌面回调系统：

09-08 09:42:05.517 1479 3574 D WindowManager: app-onAnimationFinished(): mOuter=com.android.server.wm.RemoteAnimationController@1c31bce

09-08 09:42:05.517 1479 3574 D WindowManager: onAnimationFinished(): mPendingAnimations=2

09-08 09:42:05.517 1479 3574 D WindowManager: app-release(): mOuter=com.android.server.wm.RemoteAnimationController@1c31bce

09-08 09:42:05.517 1479 3574 D WindowManager: onAnimationFinished(): Notify animation finished:

09-08 09:42:05.517 1479 3574 V WindowManager: setClientVisible: ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815} clientVisible=false Callers=com.android.server.wm.ActivityRecord.onAnimationFinished:6772 com.android.server.wm.WindowContainer.doAnimationFinished:2586 com.android.server.wm.WindowContainer.onAnimationFinished:2595 com.android.server.wm.Task.onAnimationFinished:3688 com.android.server.wm.-$$Lambda$dwJG8BAnLlvKNGuDY9U3-haNY4M.onAnimationFinished:2

09-08 09:42:05.518 1479 3574 D WindowManager: container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}

09-08 09:42:05.519 1479 3574 D WindowManager: container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}

09-08 09:42:05.519 1479 3574 I WindowManager: Finishing remote animation

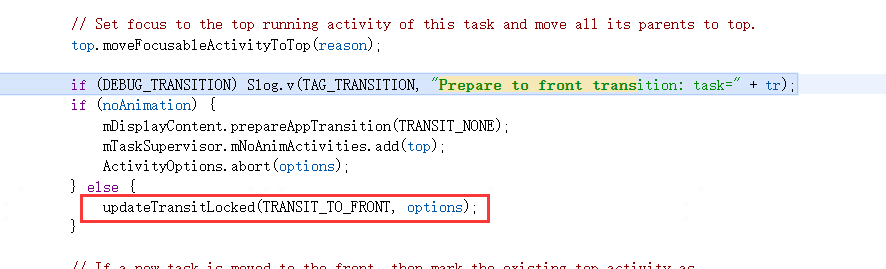
**二、热启动跳转新应用**

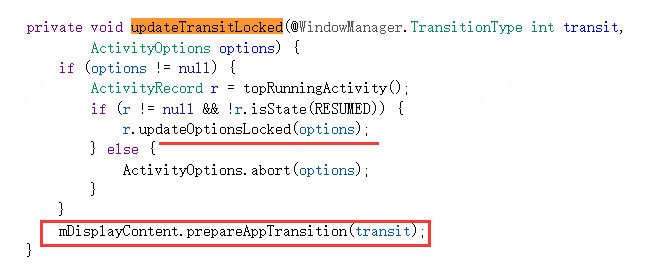
**1、prepareAppTransition准备阶段**

**在startActivity阶段会调用DisplayContent.prepareAppTransition去设置AppTransition类型为TRANSIT\_TASK\_TO\_FRONT，执行的是前文中startActivity流程的reusedTask非空分支。**

09-08 09:42:07.711 1479 4064 I ActivityTaskManager: START u0 {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10200000 cmp=com.wtf.gallery3d/.app.MainActivity bnds=[48,1681][294,1992] mCallingUid=10100} from uid 10100 and from pid 3367

09-08 09:42:07.715 1479 4064 V WindowManager: Prepare app transition: transit=TRANSIT\_TASK\_TO\_FRONT mNextAppTransition=TRANSIT\_UNSET alwaysKeepCurrent=false displayId=0 Callers=com.android.server.wm.DisplayContent.prepareAppTransition:5168 com.android.server.wm.ActivityStack.updateTransitLocked:2805 com.android.server.wm.ActivityStack.moveTaskToFront:2900 com.android.server.wm.ActivityStarter.setTargetStackIfNeeded:3156 com.android.server.wm.ActivityStarter.recycleTask:2450





**在activityPaused阶段去resumeTop时仍然如冷起场景一样都会执行重新设置transit的动作：**

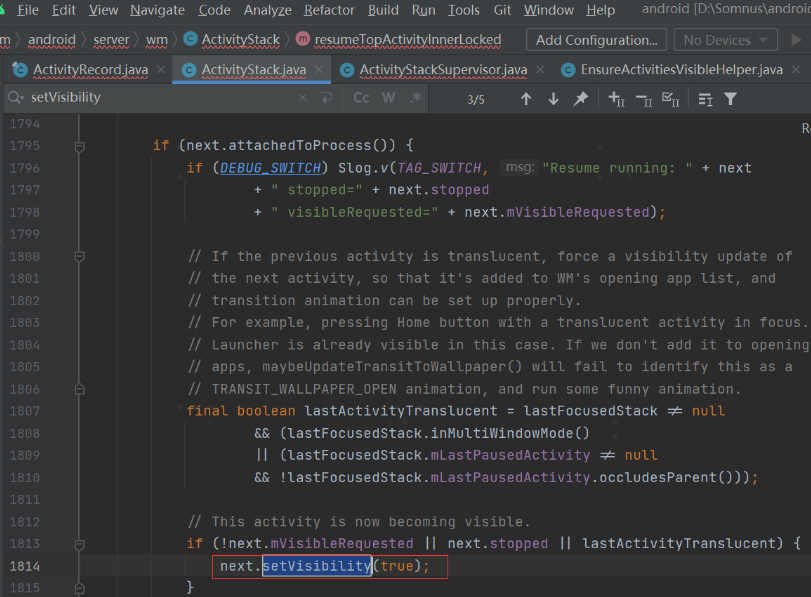
**09-08 09:42:07.741 1479 3574 V WindowManager: Prepare app transition: transit=TRANSIT\_TASK\_OPEN mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT alwaysKeepCurrent=false displayId=0 Callers=com.android.server.wm.DisplayContent.prepareAppTransition:5168 com.android.server.wm.DisplayContent.prepareAppTransition:5162 com.android.server.wm.ActivityStack.resumeTopActivityInnerLocked:2059 com.android.server.wm.ActivityStack.resumeTopActivityUncheckedLocked:1710 com.android.server.wm.RootWindowContainer.resumeFocusedStacksTopActivities:2478**

**但由于我们前面介绍过prepareAppTransition虽然被重复执行，但TRANSIT\_TASK\_OPEN不能覆盖掉TRANSIT\_TASK\_TO\_FRONT。**

**2、setVisibility阶段**

**热启场景在activityPaused阶段执行resumeTop时就会把pause成功的桌面执行setVisibility(false)加到mClosingApps、把要resume的图库执行setVisibility(true)加到mOpeningApps，而不需要依赖completePaused结束的那次ensureActivitiesVisible。**

**resumeTop时图库被setVisibility(true)：**



**09-08 09:42:07.742 1479 3574 V WindowManager: setAppVisibility(Token{5817814 ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, visible=true): mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT visible=false mVisibleRequested=false Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityStack.resumeTopActivityInnerLocked:2141 com.android.server.wm.ActivityStack.resumeTopActivityUncheckedLocked:1710 com.android.server.wm.RootWindowContainer.resumeFocusedStacksTopActivities:2478 com.android.server.wm.ActivityStack.completePauseLocked:1370 com.android.server.wm.ActivityRecord.activityPaused:5475**

**由于热启阶段需要在resumeTop设置ActivityRecord.setVisibility为true时立马去通知对端图库程序立马刷新可见性。**

**09-08 09:42:07.742 1479 3574 V WindowManager: setClientVisible: ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931} clientVisible=true Callers=com.android.server.wm.ActivityRecord.setVisibility:4504 com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityStack.resumeTopActivityInnerLocked:2141 com.android.server.wm.ActivityStack.resumeTopActivityUncheckedLocked:1710 com.android.server.wm.RootWindowContainer.resumeFocusedStacksTopActivities:2478**

**resumeTop时桌面被setVisibility(false)：**



**09-08 09:42:07.745 1479 3574 V WindowManager: setAppVisibility(Token{6d196ad ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}}, visible=false): mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT visible=true mVisibleRequested=true Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityRecord.makeInvisible:5168 com.android.server.wm.EnsureActivitiesVisibleHelper.setActivityVisibilityState:182 com.android.server.wm.EnsureActivitiesVisibleHelper.lambda$Bbb3nMFa3F8er\_OBuKA7-SpeSKo:0 com.android.server.wm.-$$Lambda$EnsureActivitiesVisibleHelper$Bbb3nMFa3F8er\_OBuKA7-SpeSKo.accept:12 com.android.internal.util.function.pooled.PooledLambdaImpl.doInvoke:307**

**热启场景resumeTop最终也会执行ActivityRecord.completeResumeLocked，同样还会重复执行一次setAppVisibility，可以认为热启场景，completeResumeLocked引发的此次setAppVisibility是最关键的一次。**

**09-08 09:42:07.746 1479 3574 V WindowManager: setAppVisibility(Token{5817814 ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, visible=true): mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT visible=false mVisibleRequested=true Callers=com.android.server.wm.ActivityRecord.setVisibility:4405 com.android.server.wm.ActivityRecord.completeResumeLocked:5373 com.android.server.wm.ActivityStack.resumeTopActivityInnerLocked:2273 com.android.server.wm.ActivityStack.resumeTopActivityUncheckedLocked:1710 com.android.server.wm.RootWindowContainer.resumeFocusedStacksTopActivities:2478 com.android.server.wm.ActivityStack.completePauseLocked:1370**

**3、executeAppTransition阶段**

**无论是冷启还是热启，都是由ActivityRecord.completeResumeLocked阶段触发的DisplayContent.executeAppTransition来最终执行AppTransition，不同的是热启阶段触发的completeResumeLocked不是由realStartActivityLocked和minimalResumeActivityLocked触发，而是由resumeTop直接触发。**

**09-08 09:42:07.746 1479 3574 W WindowManager: Execute app transition: mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT, displayId: 0 Callers=com.android.server.wm.RootWindowContainer.executeAppTransitionForAllDisplay:2399 com.android.server.wm.ActivityStackSupervisor.reportResumedActivityLocked:2092 com.android.server.wm.ActivityRecord.completeResumeLocked:5402 com.android.server.wm.ActivityStack.resumeTopActivityInnerLocked:2273 com.android.server.wm.ActivityStack.resumeTopActivityUncheckedLocked:1710**

**4、GOOD TO GO阶段**

**此阶段与冷启场景动效执行流程一样，不再赘述。**

**GoodToGo阶段日志：**

**09-08 09:42:07.753 1479 3574 V WindowManager: Check opening app=ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}: allDrawn=false startingDisplayed=true startingMoved=false isRelaunching()=false startingWindow=Window{bcb050 u0 SnapshotStartingWindow for taskId=5931}**

**09-08 09:42:07.753 1479 3574 V WindowManager: \*\*\*\* GOOD TO GO**

**09-08 09:42:07.754 1479 3574 V WindowManager: New wallpaper target=Window{8a35853 u0 com.wtf.launcher/com.wtf.launcher.Launcher}, oldWallpaper=Window{8a35853 u0 com.wtf.launcher/com.wtf.launcher.Launcher}, openingApps={ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, closingApps={ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}}**

**09-08 09:42:07.754 1479 3574 V WindowManager: New transit away from wallpaper: TRANSIT\_WALLPAPER\_CLOSE**

**09-08 09:42:07.754 1479 3574 V WindowManager: Changing app ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931} visible=false performLayout=false**

**09-08 09:42:07.754 1479 3574 V WindowManager: getAnimationTarget in={ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}}, out={Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}}**

**09-08 09:42:07.754 1479 3574 V WindowManager: Changing app ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815} visible=true performLayout=false**

**09-08 09:42:07.754 1479 3574 V WindowManager: getAnimationTarget in={ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}}, out={Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}}**

**09-08 09:42:07.754 1479 3574 D WindowManager: createAnimationAdapter(): container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}**

**09-08 09:42:07.756 1479 3574 D WindowManager: startAnimation**

**09-08 09:42:07.756 1479 3574 D WindowManager: createAnimationAdapter(): container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}**

**09-08 09:42:07.758 1479 3574 D WindowManager: startAnimation**

**09-08 09:42:07.758 1479 3574 V WindowManager: Now closing app ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}**

**09-08 09:42:07.758 1479 3574 V WindowManager: commitVisibility: ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815}: visible=false mVisibleRequested=false**

**09-08 09:42:07.758 1479 3574 V WindowManager: Now opening app ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}**

**09-08 09:42:07.758 1479 3574 V WindowManager: commitVisibility: ActivityRecord{7ddd3b9 u0 com.wtf.gallery3d/.app.MainActivity t5931}: visible=true mVisibleRequested=true**

**09-08 09:42:07.758 1479 3574 D WindowManager: goodToGo()**

**09-08 09:42:07.758 1479 3574 D WindowManager: createAppAnimations()**

**09-08 09:42:07.758 1479 3574 D WindowManager: Add container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}**

**09-08 09:42:07.758 1479 3574 D WindowManager: Add container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}**

**09-08 09:42:07.769 1479 3574 D WindowManager: startAnimation(): Notify animation start:**

**09-08 09:42:07.769 1479 3574 I WindowManager: Starting remote animation**

**09-08 09:42:07.769 1479 3574 I WindowManager: container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}**

**09-08 09:42:07.769 1479 3574 I WindowManager: Target:**

**09-08 09:42:07.769 1479 3574 I WindowManager: mode=1 taskId=5815 isTranslucent=false clipRect=[0,0][1080,2412] contentInsets=[0,96][0,0] prefixOrderIndex=43 position=[0,0] sourceContainerBounds=[0,0][1080,2412] screenSpaceBounds=[0,0][1080,2412] localBounds=[0,0][1080,2412]**

**09-08 09:42:07.769 1479 3574 I WindowManager: windowConfiguration={ mBounds=Rect(0, 0 - 1080, 2412) mAppBounds=Rect(0, 96 - 1080, 2412) mWindowingMode=fullscreen mDisplayWindowingMode=fullscreen mActivityType=home mAlwaysOnTop=undefined mRotation=ROTATION\_0}**

**09-08 09:42:07.769 1479 3574 I WindowManager: leash=Surface(name=Surface(name=Task=5815)/@0x72710c8 - animation-leash)/@0x1e9744b**

**09-08 09:42:07.769 1479 3574 I WindowManager: container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}**

**09-08 09:42:07.769 1479 3574 I WindowManager: Target:**

**09-08 09:42:07.769 1479 3574 I WindowManager: mode=0 taskId=5931 isTranslucent=false clipRect=[0,0][1080,2412] contentInsets=[0,96][0,0] prefixOrderIndex=47 position=[0,0] sourceContainerBounds=[0,0][1080,2412] screenSpaceBounds=[0,0][1080,2412] localBounds=[0,0][1080,2412]**

**09-08 09:42:07.769 1479 3574 I WindowManager: windowConfiguration={ mBounds=Rect(0, 0 - 1080, 2412) mAppBounds=Rect(0, 96 - 1080, 2412) mWindowingMode=fullscreen mDisplayWindowingMode=fullscreen mActivityType=standard mAlwaysOnTop=undefined mRotation=ROTATION\_0}**

**09-08 09:42:07.769 1479 3574 I WindowManager: leash=Surface(name=Surface(name=Task=5931)/@0xed53246 - animation-leash)/@0xed78e28**

**动效结束：**

**09-08 09:42:08.033 1479 3574 D WindowManager: app-onAnimationFinished(): mOuter=com.android.server.wm.RemoteAnimationController@f52ea08**

**09-08 09:42:08.033 1479 3574 D WindowManager: onAnimationFinished(): mPendingAnimations=2**

**09-08 09:42:08.035 1479 3574 D WindowManager: app-release(): mOuter=com.android.server.wm.RemoteAnimationController@f52ea08**

**09-08 09:42:08.035 1479 3574 D WindowManager: onAnimationFinished(): Notify animation finished:**

**09-08 09:42:08.036 1479 3574 V WindowManager: setClientVisible: ActivityRecord{e45e1e5 u0 com.wtf.launcher/.Launcher t5815} clientVisible=false Callers=com.android.server.wm.ActivityRecord.onAnimationFinished:6772 com.android.server.wm.WindowContainer.doAnimationFinished:2586 com.android.server.wm.WindowContainer.onAnimationFinished:2595 com.android.server.wm.Task.onAnimationFinished:3688 com.android.server.wm.-$$Lambda$dwJG8BAnLlvKNGuDY9U3-haNY4M.onAnimationFinished:2**

**09-08 09:42:08.037 1479 3574 D WindowManager: container=Task{646e936 #5815 visible=true type=home mode=fullscreen translucent=false I=com.wtf.launcher/.Launcher U=0 StackId=1 sz=1}**

**09-08 09:42:08.037 1479 3574 D WindowManager: container=Task{a0bb3fe #5931 visible=true type=standard mode=fullscreen translucent=true A=10180:com.wtf.gallery3d.app.Gallery U=0 StackId=5931 sz=1}**

**09-08 09:42:08.038 1479 3574 I WindowManager: Finishing remote animation**

**setVisibility:**

