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：

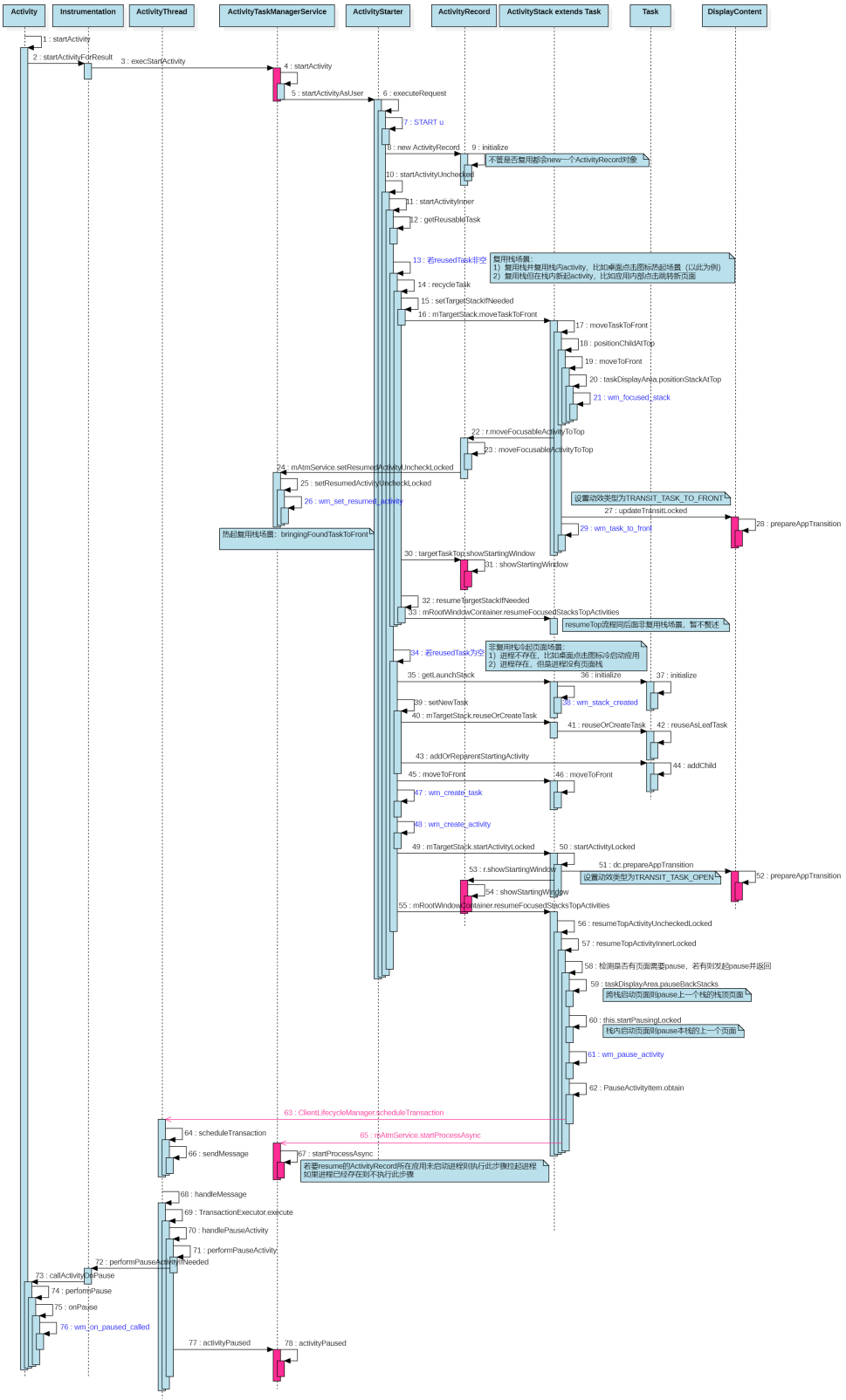
adb shell wm logging enable-text
WM_DEBUG_APP_TRANSITIONS WM_DEBUG_REMOTE_ANIMATIONS

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。




```
void startActivityLocked(ActivityRecord r, @Nullable Task topTask, boolean newTask,
        boolean isTaskSwitch, ActivityOptions options, @Nullable ActivityRecord sourceRecord) {
    final ActivityRecord pipCandidate = findEnterPipOnTaskSwitchCandidate(topTask);
    Task rTask = r.getTask();
    final boolean allowMoveToFront = options == null || !options.getAvoidMoveToFront();
    final boolean isOrHasTask = rTask == this || hasChild(rTask);
    // mLaunchTaskBehind tasks get placed at the back of the task stack.
    if (!r.mLaunchTaskBehind && allowMoveToFront && (!isOrHasTask || newTask)) {
        // Last activity in task had been removed or ActivityManagerService is reusing task.
        // Insert or replace.
        // Might not even be in.
        positionChildAtTop(rTask);
    }
    Task task = null;
```

```
if (!newTask && isOrhasTask && !r.shouldBeVisible()) {
    ActivityOptions.abort(options);
    return;
}

// Place a new activity at top of root task, so it is next to interact with the user.

// If we are not placing the new activity frontmost, we do not want to deliver the
// onUserLeaving callback to the actual frontmost activity
final Task activityTask = r.getTask();
if (task == activityTask && mChildren.indexOf(task) != (getChildCount() - 1)) {
    mTaskSupervisor.mUserLeaving = false;
    if (DEBUG_USER_LEAVING) Slog.v(TAG_USER_LEAVING,
            "startActivity() behind front, mUserLeaving=false");
}

task = activityTask;

// Slot the activity into the history root task and proceed
ProtoLog.i(WM_DEBUG_ADD_REMOVE, "Adding activity %s to task %s "
                + "callers: %s", r, task, new RuntimeException("here").fillInStackTrace());

// The transition animation and starting window are not needed if {@code allowMoveToFront}
// is false, because the activity won't be visible.
if ((!isActivityTypeHomeOrRecents() || hasActivity()) && allowMoveToFront) {
    final DisplayContent dc = mDisplayContent;
    if (DEBUG_TRANSITION) Slog.v(TAG_TRANSITION,
            "Prepare open transition: starting " + r);
    // TODO(shell-transitions): record NO_ANIMATION flag somewhere.
    if ((r.intent.getFlags() & Intent.FLAG_ACTIVITY_NO_ANIMATION) != 0) {
        dc.prepareAppTransition(TRANSIT_NONE);
        mTaskSupervisor.mNoAnimActivities.add(r);
    } else {
        int transit = TRANSIT_OLD_ACTIVITY_OPEN;
        if (newTask) {
            if (r.mLaunchTaskBehind) {
                transit = TRANSIT_OLD_TASK_OPEN_BEHIND;
            } else {
                // If a new task is being launched, then mark the existing top activity as
                // supporting picture-in-picture while pausing only if the starting activity
                // would not be considered an overlay on top of the current activity
                // (eg. not fullscreen, or the assistant)
                enableEnterPipOnTaskSwitch(pipCandidate,
                        null /* toFrontTask */, r, options);
                transit = TRANSIT_OLD_TASK_OPEN;
            }
        }
        dc.prepareAppTransition(TRANSIT_OPEN);
        mTaskSupervisor.mNoAnimActivities.remove(r);
    }
    boolean doShow = true;
    if (newTask) {
        // Even though this activity is starting fresh, we still need
        // to reset it to make sure we apply affinities to move any
        // existing activities from other tasks in to it.
        // If the caller has requested that the target task be
        // reset, then do so.
        if ((r.intent.getFlags() & Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED) != 0) {
            resetTaskIfNeeded(r, r);
            doShow = topRunningNonDelayedActivityLocked(null) == r;
        }
    } else if (options != null && options.getAnimationType()
            == ActivityOptions.ANIM_SCENE_TRANSITION) {
        doShow = false;
    }
    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) {
        // Figure out if we are transitioning from another activity that is
        // "has the same starting icon" as the next one.  This allows the
        // window manager to keep the previous window it had previously
        // created, if it still had one.
        Task baseTask = r.getTask();
        if (baseTask.isEmbedded()) {
            // If the task is embedded in a task fragment, there may have an existing
            // starting window in the parent task. This allows the embedded activities
            // to share the starting window and make sure that the window can have top
            // z-order by transferring to the top activity.
            baseTask = baseTask.getParent().asTaskFragment().getTask();
        }

        final ActivityRecord prev = baseTask.getActivity(
                a -> a.mStartingData != null && a.showToCurrentUser());
        mWmService.mStartingSurfaceController.showStartingWindow(r, prev, newTask,
                isTaskSwitch, sourceRecord);
    }
} else {
```

```
        // If this is the first activity, don't do any fancy animations,
        // because there is nothing for it to animate on top of.
        ActivityOptions.abort(options);
    }
}

/** On Task switch, finds the top activity that supports PiP. */
@Nullable
static ActivityRecord findEnterPipOnTaskSwitchCandidate(@Nullable Task topTask) {
```

最后调用了showStartingWindow后面分析

prepareAppTransition

```
     */
    @Deprecated
    void prepareAppTransition(@WindowManager.TransitionType int transit) {
        prepareAppTransition(transit, 0 /* flags */);
    }

    /**
     * @deprecated new transition should use {@link #requestTransitionAndLegacyPrepare(int, int)}
     */
    @Deprecated
    void prepareAppTransition(@WindowManager.TransitionType int transit,
            @WindowManager.TransitionFlags int flags) {
        final boolean prepared = mAppTransition.prepareAppTransition(transit, flags);
        if (prepared && okToAnimate() && transit != TRANSIT_NONE) {
            mSkipAppTransitionAnimation = false;
        }
    }
```

跳过过度动画=false，也就是不跳过

这里的判断是上面的结果是true，并且传入进来的transit不是TRANSIT_NODE标志，就需要执行过度动画

mAppTransition:

```
mAppTransition = new AppTransition(mWmService.mContext, mWmService, this);
```

mAppTransition.prepareAppTransition

AppTransition.java

```
1359            pw.print(prefix); pw.print("mLastChangingApp=");
1360            pw.println(mLastChangingApp);
1361        }
1362    }
1363
1364    boolean prepareAppTransition(@TransitionType int transit, @TransitionFlags int flags) {
1365        if (mDisplayContent.mTransitionController.isShellTransitionsEnabled()) {
1366            return false;
1367        }
1368        mNextAppTransitionRequests.add(transit);
1369        mNextAppTransitionFlags |= flags;
1370        updateBooster(); //更新启动优先级
1371        removeAppTransitionTimeoutCallbacks();
1372        mHandler.postDelayed(mHandleAppTransitionTimeoutRunnable,
1373                APP_TRANSITION_TIMEOUT_MS);
1374        return prepare();
1375    }
1376
```

添加activity组件切换类型

移除组件切换超时回调 下面重新post一个新的超时回

post一个超时回调

调用prepare

removeAppTransitionTimeoutCallbacks：默认是5000毫秒

```
void removeAppTransitionTimeoutCallbacks() {
    mHandler.removeCallbacks(mHandleAppTransitionTimeoutRunnable);
}
```

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

```
private boolean prepare() {
    if (!isRunning()) {
        setAppTransitionState(APP_STATE_IDLE)
        notifyAppTransitionPendingLocked();
        return true;
    }
    return false;
}
```

setAppTransitionState：

```
private void setAppTransitionState(int state) {
    mAppTransitionState = state;
    updateBooster();
}
```

notifyAppTransitionFinishedLocked：

```
public void notifyAppTransitionFinishedLocked(IBinder token) {
    for (int i = 0; i < mListeners.size(); i++) {
        mListeners.get(i).onAppTransitionFinishedLocked(token);
    }
}
```

mListener是数组存放ApptTransitionListener:

```
private final ArrayList<AppTransitionListener> mListeners = new ArrayList<>();
```

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。

```
// TODO(shell-transitions): record NO_ANIMATION flag somewhere.
if ((r.intent.getFlags() & Intent.FLAG_ACTIVITY_NO_ANIMATION) != 0) {
    dc.prepareAppTransition(TRANSIT_NONE);
    mTaskSupervisor.mNoAnimActivities.add(r);
} else {
    int transit = TRANSIT_OLD_ACTIVITY_OPEN;
    if (newTask) {
        if (r.mLaunchTaskBehind) {
            transit = TRANSIT_OLD_TASK_OPEN_BEHIND;
        } else {
            // If a new task is being launched, then mark the existing top activ
            // supporting picture-in-picture while pausing only if the starting
            // would not be considered an overlay on top of the current activity
            // (eg. not fullscreen, or the assistant)
            enableEnterPipOnTaskSwitch(pipCandidate,
                    null /* toFrontTask */, r, options);
            transit = TRANSIT_OLD_TASK_OPEN;
        }
    }
    dc.prepareAppTransition(TRANSIT_OPEN);
    mTaskSupervisor.mNoAnimActivities.remove(r);
}
boolean doShow = true;
```

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：

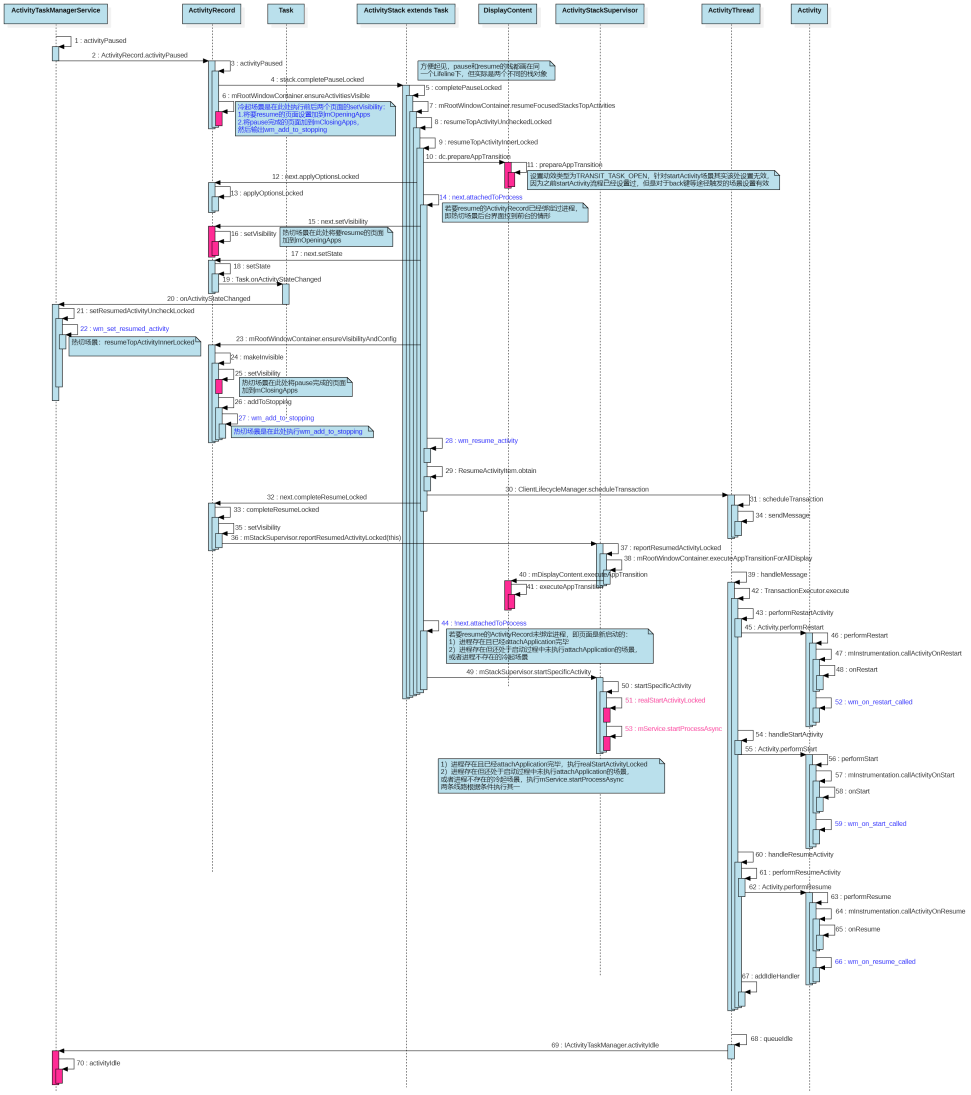
```
        }
    } else {
        if (DEBUG_TRANSITION) Slog.v(TAG_TRANSITION, "Prepare open transition: no previous");
        if (mTaskSupervisor.mNoAnimActivities.contains(next)) {
            anim = false;
            dc.prepareAppTransition(TRANSIT_NONE);
        } else {
            dc.prepareAppTransition(TRANSIT_OPEN);
        }
    }
```

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：

```
void activityPaused(boolean timeout) {
    ProtoLog.v(WM_DEBUG_STATES, "Activity paused: token=%s, timeout=%b", token,
            timeout);

    final TaskFragment taskFragment = getTaskFragment();
    if (taskFragment != null) {
        removePauseTimeout();

        final ActivityRecord pausingActivity = taskFragment.getPausingActivity();
        if (pausingActivity == this) {
            ProtoLog.v(WM_DEBUG_STATES, "Moving to PAUSED: %s %s", this,
                    (timeout ? "(due to timeout)" : " (pause complete)"));
            mAtmService.deferWindowLayout();
            try {
                taskFragment.completePause(true /* resumeNext */, null /* resumingActivity */);
            } finally {
                mAtmService.continueWindowLayout();
            }
            return;
        } else {
            EventLogTags.writeWmFailedToPause(mUserId, System.identityHashCode(this),
                    shortComponentName, pausingActivity != null
                            ? pausingActivity.shortComponentName : "(none)");
            if (isState(PAUSING)) {
                setState(PAUSED, "activityPausedLocked");
                if (finishing) {
                    ProtoLog.v(WM_DEBUG_STATES,
                            "Executing finish of failed to pause activity: %s", this);
                    completeFinishing("activityPausedLocked");
                }
            }
        }
    }

    mDisplayContent.handleActivitySizeCompatModeIfNeeded(this);
    mRootWindowContainer.ensureActivitiesVisible(null, 0, !PRESERVE_WINDOWS);
}

/**
 * Schedule a pause timeout in case the app doesn't respond. We don't give it much time because
```

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。

```
void ensureActivitiesVisible(ActivityRecord starting, int configChanges,
        boolean preserveWindows) {
    ensureActivitiesVisible(starting, configChanges, preserveWindows, true /* notifyClients */);
}

/**
 * @see #ensureActivitiesVisible(ActivityRecord, int, boolean)
 */
void ensureActivitiesVisible(ActivityRecord starting, int configChanges,
        boolean preserveWindows, boolean notifyClients) {
    if (mTaskSupervisor.inActivityVisibilityUpdate()
            || mTaskSupervisor.isRootVisibilityUpdateDeferred()) {
        // Don't do recursive work.
        return;
    }

    try {
        mTaskSupervisor.beginActivityVisibilityUpdate();
        // First the front root tasks. In case any are not fullscreen and are in front of home.
        for (int displayNdx = getChildCount() - 1; displayNdx >= 0; --displayNdx) {
            final DisplayContent display = getChildAt(displayNdx);
            display.ensureActivitiesVisible(starting, configChanges, preserveWindows,
                    notifyClients);
        }
    } finally {
        mTaskSupervisor.endActivityVisibilityUpdate();
    }
}
```

图库ensure执行的分支：

```
    if (reallyVisible) {
        if (r.finishing) {
            return;
        }
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Make visible? " + r
                    + " finishing=" + r.finishing + " state=" + r.getState());
        }
        // First: if this is not the current activity being started, make
        // sure it matches the current configuration.
        if (r != mStarting && mNotifyClients) {
            r.ensureActivityConfiguration(0 /* globalChanges */, mPreserveWindows,
                    true /* ignoreVisibility */);
        }

        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;
            }
```

makeVisibleAndRestartIfNeeded:

```
    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 && !r.isState(INITIALIZING)) {
            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 */);
        }
    }
}
```

添加到 mOpenling列表

启动Activity流程

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

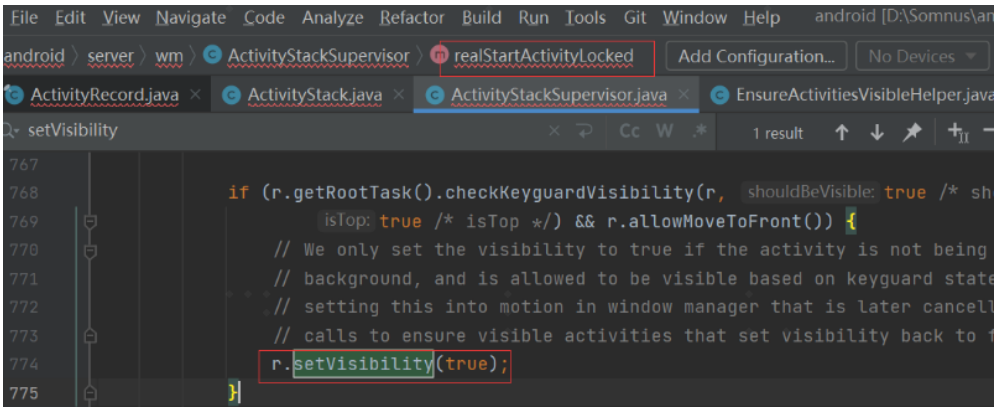
com.android.server.wm.-$$Lambda$EnsureActivitiesVisibleHelper$Bbb3nMFa3F8er_OBuKA7-SpeSKo.accept:12 com.android.internal.util.function.pooled.PooledLambdaImpl.doInvoke:307

桌面ensure执行的分支：

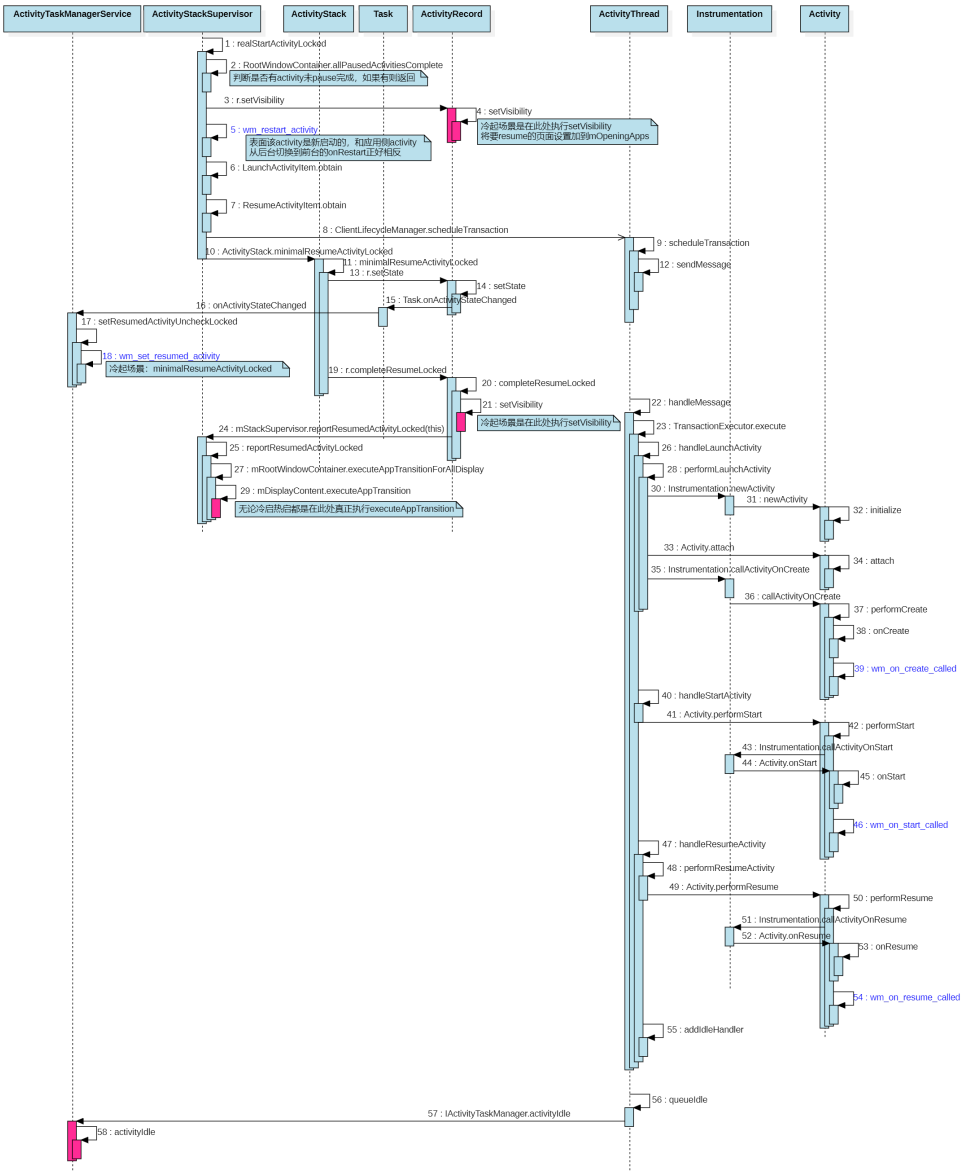
```
        if (!r.attachedToProcess()) {          图库走这里
            makeVisibleAndRestartIfNeeded(mStarting, mConfigChanges, isTop,
                    resumeTopActivity && isTop, r);
        } else if (r.mVisibleRequested) {
            // If this activity is already visible, then there is nothing 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.
        mConfigChanges |= r.configChangeFlags;
    } else {
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Make invisible? " + r
                    + " finishing=" + r.finishing + " state=" + r.getState()
                    + " containerShouldBeVisible=" + mContainerShouldBeVisible
                    + " behindFullyOccludedContainer=" + mBehindFullyOccludedContainer
                    + " mLaunchTaskBehind=" + r.mLaunchTaskBehind);
        }
        r.makeInvisible();   桌面走这里
    }
```

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：

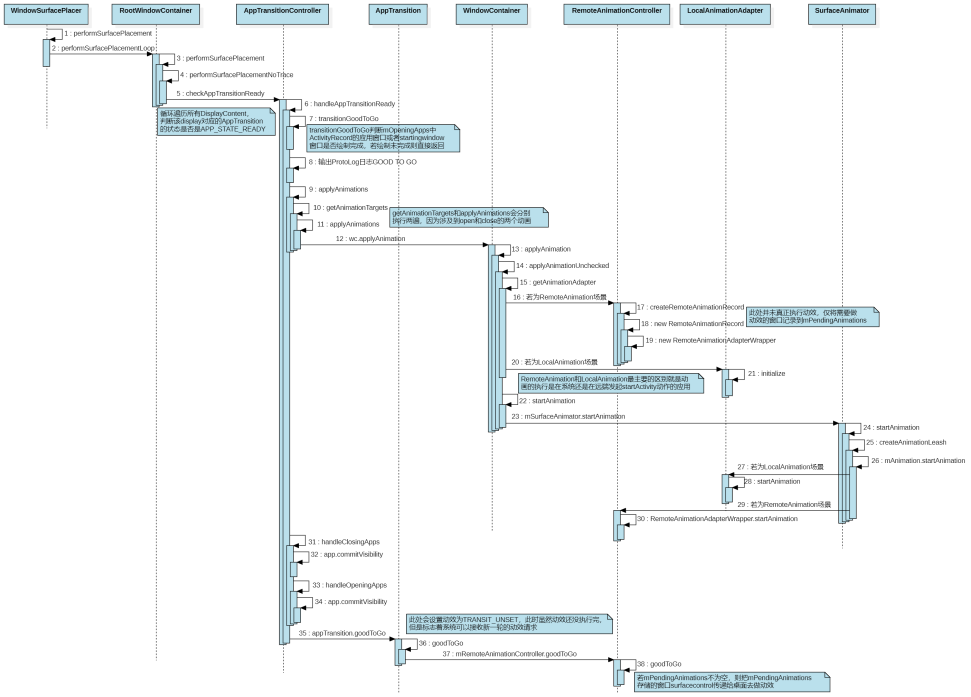
1）首先调用AppTransition.isTransitionSet()判断是否有设置transit，若未设置则该函数不产生任何作用。

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：

```
void executeAppTransition() {
    mTransitionController.setReady(this);
    if (mAppTransition.isTransitionSet()) { 是否设置了transit
        ProtoLog.w(WM_DEBUG_APP_TRANSITIONS,
                "Execute app transition: %s, displayId: %d Callers=%s",
                mAppTransition, mDisplayId, Debug.getCallers(5));
        mAppTransition.setReady(); 设置为READY状态
        mWmService.mWindowPlacerLocked.requestTraversal();
    }
}
                                        post刷新
```

requestTraversal：

```
    void requestTraversal() {
        if (mTraversalScheduled) {
            return;
        }

        // Set as scheduled even the request will be deferred because mDeferredRequests is also
        // increased, then the end of deferring will perform the request.
        mTraversalScheduled = true;
        if (mDeferDepth > 0) {
            mDeferredRequests++;
            if (DEBUG) Slog.i(TAG, "Defer requestTraversal " + Debug.getCallers(3));
            return;
        }
        mService.mAnimationHandler.post(mPerformSurfacePlacement);
    }
```

performSurfacePlacement:

```
final void performSurfacePlacement() {
    performSurfacePlacement(false /* force */);
}

final void performSurfacePlacement(boolean force) {
    if (mDeferDepth > 0 && !force) {
        mDeferredRequests++;
        return;
    }
    int loopCount = 6;
    do {
        mTraversalScheduled = false;
        performSurfacePlacementLoop();
        mService.mAnimationHandler.removeCallbacks(mPerformSurfacePlacement);
        loopCount--;
    } while (mTraversalScheduled && loopCount > 0);
    mService.mRoot.mWallpaperActionPending = false;
}

private void performSurfacePlacementLoop() {
    if (mInLayout) {
        if (DEBUG) {
            throw new RuntimeException("Recursive call!");
        }
        Slog.w(TAG, "performLayoutAndPlaceSurfacesLocked called while in layout. Callers="
                + Debug.getCallers(3));
        return;
    }

    // TODO(multi-display):
    final DisplayContent defaultDisplay = mService.getDefaultDisplayContentLocked();
    if (defaultDisplay.mWaitingForConfig) {
        // Our configuration has changed (most likely rotation), but we
        // don't yet have the complete configuration to report to
        // applications.  Don't do any window layout until we have it.
        return;
    }

    if (!mService.mDisplayReady) {
        // Not yet initialized, nothing to do.
        return;
    }

    mInLayout = true;

    if (!mService.mForceRemoves.isEmpty()) {
        // Wait a little bit for things to settle down, and off we go.
        while (!mService.mForceRemoves.isEmpty()) {
            final WindowState ws = mService.mForceRemoves.remove(0);
            Slog.i(TAG, "Force removing: " + ws);
            ws.removeImmediately();
        }
        Slog.w(TAG, "Due to memory failure, waiting a bit for next layout");
        Object tmp = new Object();
        synchronized (tmp) {
            try {
                tmp.wait(250);
            } catch (InterruptedException e) {
            }
        }
    }

    try {
        mService.mRoot.performSurfacePlacement();

        mInLayout = false;

        if (mService.mRoot.isLayoutNeeded()) {
            if (++mLayoutRepeatCount < 6) {
```

RootWindowContainer.performSurfacePlacement:

RootWindowContainer.java

```
776         }
777
778         return leakedSurface || killedApps;
779     }
780
781     void performSurfacePlacement() {
782         Trace.traceBegin(TRACE_TAG_WINDOW_MANAGER, "performSurfacePlacement");
783         try {
784             performSurfacePlacementNoTrace();
785         } finally {
786             Trace.traceEnd(TRACE_TAG_WINDOW_MANAGER);
787         }
788     }
789
```

performSurfacePlacementNoTrace:

```
// "Something has changed!  Let's make it correct now."
// TODO: Super long method that should be broken down...
void performSurfacePlacementNoTrace() {
    if (DEBUG_WINDOW_TRACE) {
        Slog.v(TAG, "performSurfacePlacementInner: entry. Called by "
                + Debug.getCallers(3));
    }

    int i;

    if (mWmService.mFocusMayChange) {
        mWmService.mFocusMayChange = false;
        mWmService.updateFocusedWindowLocked(
                UPDATE_FOCUS_WILL_PLACE_SURFACES, false /*updateInputWindows*/);
    }

    mHoldScreen = null;
    mScreenBrightnessOverride = PowerManager.BRIGHTNESS_INVALID_FLOAT;
    mUserActivityTimeout = -1;
    mObscureApplicationContentOnSecondaryDisplays = false;
    mSustainedPerformanceModeCurrent = false;
    mWmService.mTransactionSequence++;

    // TODO(multi-display): recents animation & wallpaper need support multi-display.
    final DisplayContent defaultDisplay = mWmService.getDefaultDisplayContentLocked();
    final WindowSurfacePlacer surfacePlacer = mWmService.mWindowPlacerLocked;

    if (SHOW_LIGHT_TRANSACTIONS) {
        Slog.i(TAG,
                ">>> OPEN TRANSACTION performLayoutAndPlaceSurfaces");
    }
    Trace.traceBegin(TRACE_TAG_WINDOW_MANAGER, "applySurfaceChanges");
    mWmService.openSurfaceTransaction();
    try {
        applySurfaceChangesTransaction();
    } catch (RuntimeException e) {
        Slog.wtf(TAG, "Unhandled exception in Window Manager", e);
    } finally {
        mWmService.closeSurfaceTransaction("performLayoutAndPlaceSurfaces");
        Trace.traceEnd(TRACE_TAG_WINDOW_MANAGER);
        if (SHOW_LIGHT_TRANSACTIONS) {
            Slog.i(TAG,
                    "<<< CLOSE TRANSACTION performLayoutAndPlaceSurfaces");
        }
    }

    // Send any pending task-info changes that were queued-up during a layout deferment
    mWmService.mAtmService.mTaskOrganizerController.dispatchPendingEvents();
    mWmService.mAtmService.mTaskFragmentOrganizerController.dispatchPendingEvents();
    mWmService.mSyncEngine.onSurfacePlacement();
    mWmService.mAnimator.executeAfterPrepareSurfacesRunnables();

    checkAppTransitionReady(surfacePlacer);

    // Defer starting the recents animation until the wallpaper has drawn
    final RecentsAnimationController recentsAnimationController =
            mWmService.getRecentsAnimationController();
    if (recentsAnimationController != null) {
        recentsAnimationController.checkAnimationReady(defaultDisplay.mWallpaperController);
    }

    for (int displayNdx = 0; displayNdx < mChildren.size(); ++displayNdx) {
        final DisplayContent displayContent = mChildren.get(displayNdx);
```

checkAppTransitionReady:

```
private void checkAppTransitionReady(WindowSurfacePlacer surfacePlacer) {
    // Trace all displays app transition by Z-order for pending layout change.
    for (int i = mChildren.size() - 1; i >= 0; --i) {
        final DisplayContent curDisplay = mChildren.get(i);

        // If we are ready to perform an app transition, check through all of the app tokens
        // to be shown and see if they are ready to go.
        if (curDisplay.mAppTransition.isReady()) {   已经准备好了
            // handleAppTransitionReady may modify curDisplay.pendingLayoutChanges.
            curDisplay.mAppTransitionController.handleAppTransitionReady();
            if (DEBUG_LAYOUT_REPEATS) {
                surfacePlacer.debugLayoutRepeats("after handleAppTransitionReady",
                        curDisplay.pendingLayoutChanges);
            }
        }

        if (curDisplay.mAppTransition.isRunning() && !curDisplay.isAppTransitioning()) {
            // We have finished the animation of an app transition. To do this, we have
            // delayed a lot of operations like showing and hiding apps, moving apps in
            // Z-order, etc.
            // The app token list reflects the correct Z-order, but the window list may now
            // be out of sync with it. So here we will just rebuild the entire app window
            // list. Fun!
            curDisplay.handleAnimatingStoppedAndTransition();
            if (DEBUG_LAYOUT_REPEATS) {
                surfacePlacer.debugLayoutRepeats("after handleAnimStopAndXitionLock",
                        curDisplay.pendingLayoutChanges);
            }
        }
    }
}
```

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。

```
/**
 * Handle application transition for given display.
 */
void handleAppTransitionReady() {
    mTempTransitionReasons.clear();
    if (!transitionGoodToGo(mDisplayContent.mOpeningApps, mTempTransitionReasons)
            || !transitionGoodToGo(mDisplayContent.mChangingContainers, mTempTransitionReasons)
            || !transitionGoodToGoForTaskFragments()) {
        return;
    }
    Trace.traceBegin(Trace.TRACE_TAG_WINDOW_MANAGER, "AppTransitionReady");

    ProtoLog.v(WM_DEBUG_APP_TRANSITIONS, "**** GOOD TO GO");   关键Log
    // TODO(b/205335975): Remove window which stuck in animatingExit status. Find actual cause.
    mDisplayContent.forAllWindows(WindowState::cleanupAnimatingExitWindow,
```

transitionGoodToGo:

```
private boolean transitionGoodToGo(ArraySet<? extends WindowContainer> apps,
        ArrayMap<WindowContainer, Integer> outReasons) {
    ProtoLog.v(WM_DEBUG_APP_TRANSITIONS,
            "Checking %d opening apps (frozen=%b timeout=%b)...", apps.size(),
            mService.mDisplayFrozen, mDisplayContent.mAppTransition.isTimeout());
    if (mDisplayContent.mAppTransition.isTimeout()) {
        return true;
    }
    final ScreenRotationAnimation screenRotationAnimation = mService.mRoot.getDisplayContent(
            Display.DEFAULT_DISPLAY).getRotationAnimation();

    // Imagine the case where we are changing orientation due to an app transition, but a
    // previous orientation change is still in progress. We won't process the orientation
    // change for our transition because we need to wait for the rotation animation to
    // finish.
    // If we start the app transition at this point, we will interrupt it halfway with a
    // new rotation animation after the old one finally finishes. It's better to defer the
    // app transition.
    if (screenRotationAnimation != null && screenRotationAnimation.isAnimating()
            && mDisplayContent.getDisplayRotation().needsUpdate()) {
        ProtoLog.v(WM_DEBUG_APP_TRANSITIONS,
                "Delaying app transition for screen rotation animation to finish");
        return false;
    }
    for (int i = 0; i < apps.size(); i++) {
        WindowContainer wc = apps.valueAt(i);
        final ActivityRecord activity = getAppFromContainer(wc);
        if (activity == null) {
            continue;
        }
        ProtoLog.v(WM_DEBUG_APP_TRANSITIONS,
                "Check opening app=%s: allDrawn=%b startingDisplayed=%b "
                        + "startingMoved=%b isRelaunching()=%b startingWindow=%s",
                activity, activity.allDrawn, activity.startingDisplayed,
                activity.startingMoved, activity.isRelaunching(),
                activity.mStartingWindow);
        final boolean allDrawn = activity.allDrawn && !activity.isRelaunching();
        if (!allDrawn && !activity.startingDisplayed && !activity.startingMoved) {
            return false;
        }
        if (allDrawn) {
            outReasons.put(activity, APP_TRANSITION_WINDOWS_DRAWN);
        } else {
            outReasons.put(activity,
                    activity.mStartingData instanceof SplashScreenStartingData
                            ? APP_TRANSITION_SPLASH_SCREEN
                            : APP_TRANSITION_SNAPSHOT);
        }
    }

    // We also need to wait for the specs to be fetched, if needed.
    if (mDisplayContent.mAppTransition.isFetchingAppTransitionsSpecs()) {
        ProtoLog.v(WM_DEBUG_APP_TRANSITIONS, "isFetchingAppTransitionSpecs=true");
        return false;
    }

    if (!mDisplayContent.mUnknownAppVisibilityController.allResolved()) {
        ProtoLog.v(WM_DEBUG_APP_TRANSITIONS, "unknownApps is not empty: %s",
                mDisplayContent.mUnknownAppVisibilityController.getDebugMessage());
        return false;
    }

    // If the wallpaper is visible, we need to check it's ready too.
    return !mWallpaperControllerLocked.isWallpaperVisible()
            || mWallpaperControllerLocked.wallpaperTransitionReady();
}

private boolean transitionGoodToGoForTaskFragments() {
    if (mDisplayContent.mAppTransition.isTimeout()) {
```

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}

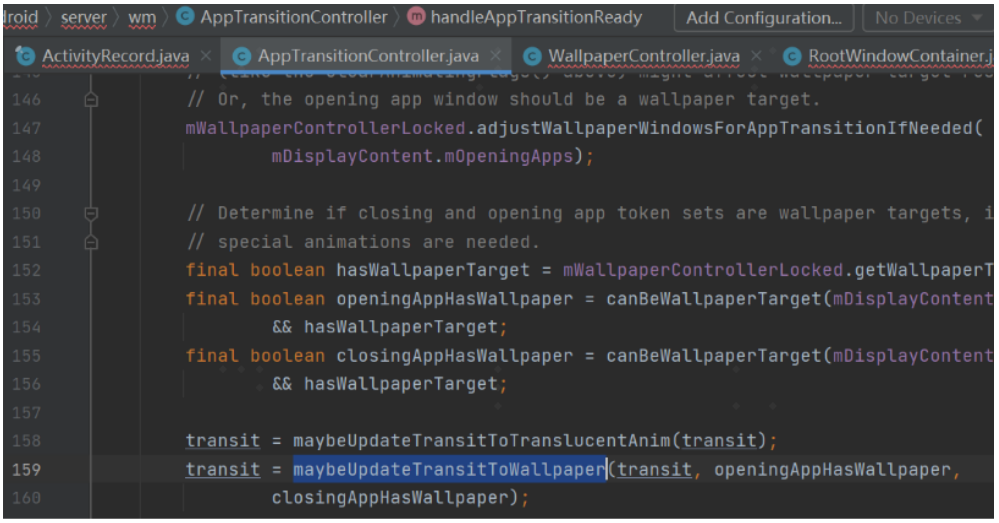
桌面打开应用就属于有startingwindow绘制完成但主窗口并未绘制完毕的情形。

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：

```
final int layoutRedo;
mService.mSurfaceAnimationRunner.deferStartingAnimations();
try {
    applyAnimations(openingAppsForAnimation, closingAppsForAnimation, transit, animLp,
            voiceInteraction);
    handleClosingApps();
    handleOpeningApps();
    handleChangingApps(transit);
```

applyAnimations：

```
private void applyAnimations(ArraySet<ActivityRecord> openingApps,
        ArraySet<ActivityRecord> closingApps, @TransitionOldType int transit,
        LayoutParams animLp, boolean voiceInteraction) {
    if (transit == WindowManager.TRANSIT_OLD_UNSET
            || (openingApps.isEmpty() && closingApps.isEmpty())) {
        return;
    }

    final ArraySet<WindowContainer> openingWcs = getAnimationTargets(
            openingApps, closingApps, true /* visible */);
    final ArraySet<WindowContainer> closingWcs = getAnimationTargets(
            openingApps, closingApps, false /* visible */);
    applyAnimations(openingWcs, openingApps, transit, true /* visible */, animLp,
            voiceInteraction);
    applyAnimations(closingWcs, closingApps, transit, false /* visible */, animLp,
            voiceInteraction);
    final RecentsAnimationController rac = mService.getRecentsAnimationController();
    if (rac != null) {
        rac.sendTasksAppeared();
    }

    for (int i = 0; i < openingApps.size(); ++i) {
        openingApps.valueAtUnchecked(i).mOverrideTaskTransition = false;
    }
    for (int i = 0; i < closingApps.size(); ++i) {
        closingApps.valueAtUnchecked(i).mOverrideTaskTransition = false;
    }
```

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:

```
private void applyAnimations(ArraySet<WindowContainer> wcs, ArraySet<ActivityRecord> apps,
        @TransitionOldType int transit, boolean visible, LayoutParams animLp,
        boolean voiceInteraction) {
    final int wcsCount = wcs.size();
    for (int i = 0; i < wcsCount; i++) {
        final WindowContainer wc = wcs.valueAt(i);
        // If app transition animation target is promoted to higher level, SurfaceAnimator
        // triggers WC#onAnimationFinished only on the promoted target. So we need to take care
        // of triggering AR#onAnimationFinished on each ActivityRecord which is a part of the
        // app transition.
        final ArrayList<ActivityRecord> transitioningDescendants = new ArrayList<>();
        for (int j = 0; j < apps.size(); ++j) {
            final ActivityRecord app = apps.valueAt(j);
            if (app.isDescendantOf(wc)) {
                transitioningDescendants.add(app);
            }
        }
        wc.applyAnimation(animLp, transit, visible, voiceInteraction, transitioningDescendants);
    }
}
```

applyAnimations:

WindowContainer.java

```
2868     * @return {@code true} when the container applied the app transition, {@code false} if the
2869     *         app transition is disabled or skipped.
2870     *
2871     * @see #getAnimationAdapter
2872     */
2873    boolean applyAnimation(WindowManager.LayoutParams lp, @TransitionOldType int transit,
2874            boolean enter, boolean isVoiceInteraction,
2875            @Nullable ArrayList<WindowContainer> sources) {
2876        if (mWmService.mDisableTransitionAnimation) {
2877            ProtoLog.v(WM_DEBUG_APP_TRANSITIONS_ANIM,
2878                    "applyAnimation: transition animation is disabled or skipped. "
2879                            + "container=%s", this);
2880            cancelAnimation();
2881            return false;
2882        }
2883
2884        // Only apply an animation if the display isn't frozen. If it is frozen, there is no reason
2885        // to animate and it can cause strange artifacts when we unfreeze the display if some
2886        // different animation is running.
2887        try {
2888            Trace.traceBegin(TRACE_TAG_WINDOW_MANAGER, "WC#applyAnimation");
2889            if (okToAnimate()) {
2890                ProtoLog.v(WM_DEBUG_APP_TRANSITIONS_ANIM,
2891                        "applyAnimation: transit=%s, enter=%b, wc=%s",
2892                        AppTransition.appTransitionOldToString(transit), enter, this);
2893                applyAnimationUnchecked(lp, enter, transit, isVoiceInteraction, sources);
2894            } else {
2895                cancelAnimation();
2896            }
2897        } finally {
2898            Trace.traceEnd(TRACE_TAG_WINDOW_MANAGER);
2899        }
2900
2901        return isAnimating();
2902    }
```

applyAnimationUnchecked:

```
protected void applyAnimationUnchecked(WindowManager.LayoutParams lp, boolean enter,
        @TransitionOldType int transit, boolean isVoiceInteraction,
        @Nullable ArrayList<WindowContainer> sources) {
    final Task task = asTask();
    if (task != null && !enter && !task.isActivityTypeHomeOrRecents()) {
        final InsetsControlTarget imeTarget = mDisplayContent.getImeTarget(IME_TARGET_LAYERING);
        final boolean isImeLayeringTarget = imeTarget != null && imeTarget.getWindow() != null
                && imeTarget.getWindow().getTask() == task;
        // Attach and show the IME screenshot when the task is the IME target and performing
        // task closing transition to the next task.
        if (isImeLayeringTarget && AppTransition.isTaskCloseTransitOld(transit)) {
            mDisplayContent.showImeScreenshot();
        }
    }
    final Pair<AnimationAdapter, AnimationAdapter> adapters = getAnimationAdapter(lp,
            transit, enter, isVoiceInteraction);
    AnimationAdapter adapter = adapters.first;
    AnimationAdapter thumbnailAdapter = adapters.second;
    if (adapter != null) {
        if (sources != null) {
            mSurfaceAnimationSources.addAll(sources);
        }

        AnimationRunnerBuilder animationRunnerBuilder = new AnimationRunnerBuilder();

        if (isTaskTransitOld(transit)) {
            animationRunnerBuilder.setTaskBackgroundColor(getTaskAnimationBackgroundColor());
            // TODO: Remove when we migrate to shell (b/202383002)
            if (mWmService.mTaskTransitionSpec != null) {
                animationRunnerBuilder.hideInsetSourceViewOverflows(
                        mWmService.mTaskTransitionSpec.animationBoundInsets);
            }
        }

        final ActivityRecord activityRecord = asActivityRecord();
        if (activityRecord != null && isActivityTransitOld(transit)
                && adapter.getShowBackground()) {
            final @ColorInt int backgroundColorForTransition;
            if (adapter.getBackgroundColor() != 0) {
                // If available use the background color provided through getBackgroundColor
                // which if set originates from a call to overridePendingAppTransition.
                backgroundColorForTransition = adapter.getBackgroundColor();
            } else {
                // Otherwise default to the window's background color if provided through
                // the theme as the background color for the animation - the top most window
                // with a valid background color and showBackground set takes precedence.
                final Task arTask = activityRecord.getTask();
                backgroundColorForTransition = ColorUtils.setAlphaComponent(
                        arTask.getTaskDescription().getBackgroundColor(), 255);
            }
            animationRunnerBuilder.setTaskBackgroundColor(backgroundColorForTransition);
        }

        animationRunnerBuilder.build()
                .startAnimation(getPendingTransaction(), adapter, !isVisible(),
                        ANIMATION_TYPE_APP_TRANSITION, thumbnailAdapter);

        if (adapter.getShowWallpaper()) {
            getDisplayContent().pendingLayoutChanges |= FINISH_LAYOUT_REDO_WALLPAPER;
        }
```

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。

```
Pair<AnimationAdapter, AnimationAdapter> getAnimationAdapter(WindowManager.LayoutParams lp,
        @TransitionOldType int transit, boolean enter, boolean isVoiceInteraction) {
    final Pair<AnimationAdapter, AnimationAdapter> resultAdapters;
    final int appRootTaskClipMode = getDisplayContent().mAppTransition.getAppRootTaskClipMode();

    // Separate position and size for use in animators.
    final Rect screenBounds = getAnimationBounds(appRootTaskClipMode);
    mTmpRect.set(screenBounds);
    if (this.asTask() != null && isTaskTransitOld(transit)) {
        this.asTask().adjustAnimationBoundsForTransition(mTmpRect);
    }
    getAnimationPosition(mTmpPoint);
    mTmpRect.offsetTo(0, 0);

    final RemoteAnimationController controller =
            getDisplayContent().mAppTransition.getRemoteAnimationController();
    final boolean isChanging = AppTransition.isChangeTransitOld(transit) && enter
            && isChangingAppTransition();

    // Delaying animation start isn't compatible with remote animations at all.
    if (controller != null && !mSurfaceAnimator.isAnimationStartDelayed()) {
        final Rect localBounds = new Rect(mTmpRect);
        localBounds.offsetTo(mTmpPoint.x, mTmpPoint.y);
        final RemoteAnimationController.RemoteAnimationRecord adapters =
                controller.createRemoteAnimationRecord(this, mTmpPoint, localBounds,
                        screenBounds, (isChanging ? mSurfaceFreezer.mFreezeBounds : null));
        if (!isChanging) {
            adapters.setMode(enter
                    ? RemoteAnimationTarget.MODE_OPENING
                    : RemoteAnimationTarget.MODE_CLOSING);
        }
        resultAdapters = new Pair<>(adapters.mAdapter, adapters.mThumbnailAdapter);
    } else if (isChanging) {
        final float durationScale = mWmService.getTransitionAnimationScaleLocked();
        final DisplayInfo displayInfo = getDisplayContent().getDisplayInfo();
        mTmpRect.offsetTo(mTmpPoint.x, mTmpPoint.y);

        final AnimationAdapter adapter = new LocalAnimationAdapter(
                new WindowChangeAnimationSpec(mSurfaceFreezer.mFreezeBounds, mTmpRect,
                        displayInfo, durationScale, true /* isAppAnimation */,
                        false /* isThumbnail */),
                getSurfaceAnimationRunner());

        final AnimationAdapter thumbnailAdapter = mSurfaceFreezer.mSnapshot != null
                ? new LocalAnimationAdapter(new WindowChangeAnimationSpec(
```

createRemoteAnimationRecord：

```
 */
RemoteAnimationRecord createRemoteAnimationRecord(WindowContainer windowContainer,
        Point position, Rect localBounds, Rect endBounds, Rect startBounds) {
    ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "createAnimationAdapter(): container=%s",
            windowContainer);
    final RemoteAnimationRecord adapters = new RemoteAnimationRecord(windowContainer, position,
            localBounds, endBounds, startBounds);
    mPendingAnimations.add(adapters);
    return adapters;
}
```

RemoteAnimationRecord：

```
*/
public class RemoteAnimationRecord {
    RemoteAnimationAdapterWrapper mAdapter;
    RemoteAnimationAdapterWrapper mThumbnailAdapter = null;
    RemoteAnimationTarget mTarget;
    final WindowContainer mWindowContainer;
    final Rect mStartBounds;
    private @RemoteAnimationTarget.Mode int mMode = RemoteAnimationTarget.MODE_CHANGING;
                                    构造函数
    RemoteAnimationRecord(WindowContainer windowContainer, Point endPos, Rect localBounds,
            Rect endBounds, Rect startBounds) {
        mWindowContainer = windowContainer;
        if (startBounds != null) {
            mStartBounds = new Rect(startBounds);
            mAdapter = new RemoteAnimationAdapterWrapper(this, endPos, localBounds, endBounds,
                    mStartBounds);
            if (mRemoteAnimationAdapter.getChangeNeedsSnapshot()) {
                final Rect thumbnailLocalBounds = new Rect(startBounds);
                thumbnailLocalBounds.offsetTo(0, 0);
                // Snapshot is located at (0,0) of the animation leash. It doesn't have size
                // change, so the startBounds is its end bounds, and no start bounds for it.
                mThumbnailAdapter = new RemoteAnimationAdapterWrapper(this, new Point(0, 0),
                        thumbnailLocalBounds, startBounds, new Rect());
            }
        } else {
            mAdapter = new RemoteAnimationAdapterWrapper(this, endPos, localBounds, endBounds,
                    new Rect());
            mStartBounds = null;
        }
    }
```

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

```
void startAnimation(Transaction t, AnimationAdapter anim, boolean hidden,
        @AnimationType int type,
        @Nullable OnAnimationFinishedCallback animationFinishedCallback,
        @Nullable Runnable animationCancelledCallback,
        @Nullable AnimationAdapter snapshotAnim) {
    ProtoLog.v(WM_DEBUG_ANIM, "Starting animation on %s: type=%d, anim=%s",
            this, type, anim);

    // TODO: This should use isVisible() but because isVisible has a really weird meaning at
    // the moment this doesn't work for all animatable window containers.
    mSurfaceAnimator.startAnimation(t, anim, hidden, type, animationFinishedCallback,
            animationCancelledCallback, snapshotAnim, mSurfaceFreezer);
}
```

SurfaceAnimator.startAnimation：

```
    */
    void startAnimation(Transaction t, AnimationAdapter anim, boolean hidden,
            @AnimationType int type,
            @Nullable OnAnimationFinishedCallback animationFinishedCallback,
            @Nullable Runnable animationCancelledCallback,
            @Nullable AnimationAdapter snapshotAnim, @Nullable SurfaceFreezer freezer) {
        cancelAnimation(t, true /* restarting */, true /* forwardCancel */);
        mAnimation = anim;
        mAnimationType = type;
        mSurfaceAnimationFinishedCallback = animationFinishedCallback;
        mAnimationCancelledCallback = animationCancelledCallback;
        final SurfaceControl surface = mAnimatable.getSurfaceControl();
        if (surface == null) {
            Slog.w(TAG, "Unable to start animation, surface is null or no children.");
            cancelAnimation();
            return;
        }
        mLeash = freezer != null ? freezer.takeLeashForAnimation() : null;
        if (mLeash == null) {
            mLeash = createAnimationLeash(mAnimatable, surface, t, type,
                    mAnimatable.getSurfaceWidth(), mAnimatable.getSurfaceHeight(), 0 /* x */,
                    0 /* y */, hidden, mService.mTransactionFactory);
            mAnimatable.onAnimationLeashCreated(t, mLeash);
        }
        mAnimatable.onLeashAnimationStarting(t, mLeash);
        if (mAnimationStartDelayed) {
            ProtoLog.i(WM_DEBUG_ANIM, "Animation start delayed for %s", mAnimatable);
            return;
        }
        mAnimation.startAnimation(mLeash, t, type, mInnerAnimationFinishedCallback);
        if (ProtoLogImpl.isEnabled(WM_DEBUG_ANIM)) {
            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw);
            mAnimation.dump(pw, "");
            ProtoLog.d(WM_DEBUG_ANIM, "Animation start for %s, anim=%s", mAnimatable, sw);
        }
        if (snapshotAnim != null) {
            mSnapshot = freezer.takeSnapshotForAnimation();
            if (mSnapshot == null) {
                Slog.e(TAG, "No snapshot target to start animation on for " + mAnimatable);
                return;
            }
            mSnapshot.startAnimation(t, snapshotAnim, type);
        }
    }

    void startAnimation(Transaction t, AnimationAdapter anim, boolean hidden,
            @AnimationType int type) {
        startAnimation(t, anim, hidden, type, null /* animationFinishedCallback */,
                null /* animationCancelledCallback */, null /* snapshotAnim */, null /* freezer */);
    }
```

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：

RemoteAnimationController.java

```
498
499         @Override
500         public boolean getShowWallpaper() {
501             return false;
502         }
503
504         @Override
505         public void startAnimation(SurfaceControl animationLeash, Transaction t,
506                 @AnimationType int type, @NonNull OnAnimationFinishedCallback finishCallback) {
507             ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "startAnimation");
508
509             if (mStartBounds.isEmpty()) {
510                 // Restore position and stack crop until client has a chance to modify it.
511                 t.setPosition(animationLeash, mPosition.x, mPosition.y);
512                 t.setWindowCrop(animationLeash, mEndBounds.width(), mEndBounds.height());
513             } else {
514                 // Offset the change animation leash to the relative start position in parent.
515                 // (mPosition) is the relative end position in parent container.
516                 // (mStartBounds - mEndBounds) is the position difference between start and end.
517                 // (mPosition + mStartBounds - mEndBounds) will be the relative start position.
518                 t.setPosition(animationLeash, mPosition.x + mStartBounds.left - mEndBounds.left,
519                         mPosition.y + mStartBounds.top - mEndBounds.top);
520                 t.setWindowCrop(animationLeash, mStartBounds.width(), mStartBounds.height());
521             }
522             mCapturedLeash = animationLeash;
523             mCapturedFinishCallback = finishCallback;
524             mAnimationType = type;
525         }
```

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。

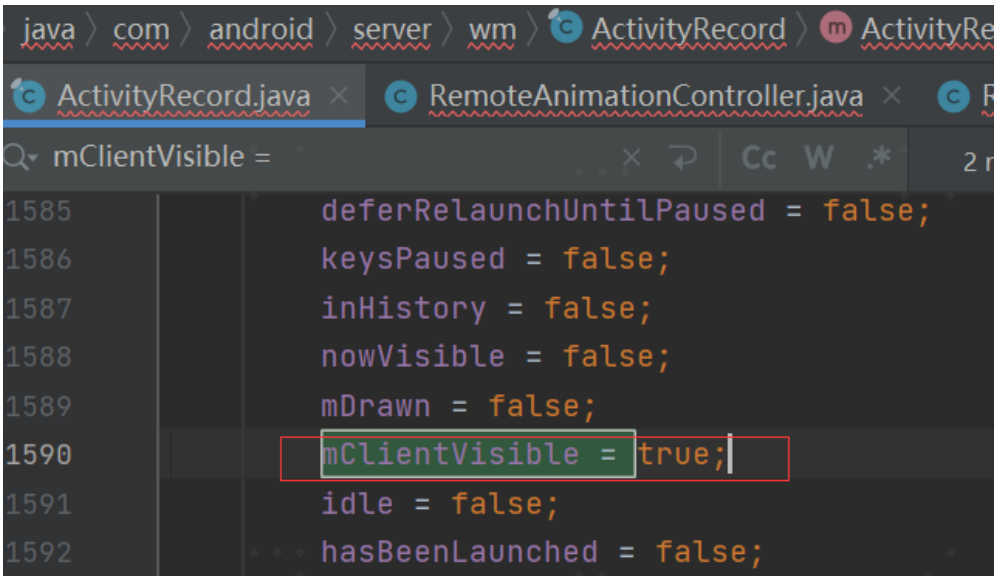
ActivityRecord.postApplyAnimation函数去尝试调用setClientVisible：

```
 */
private void postApplyAnimation(boolean visible, boolean fromTransition) {
    final boolean usingShellTransitions = mTransitionController.isShellTransitionsEnabled();
    final boolean delayed = isAnimating(PARENTS | CHILDREN,
            ANIMATION_TYPE_APP_TRANSITION | ANIMATION_TYPE_WINDOW_ANIMATION
                    | ANIMATION_TYPE_RECENTS);
    if (!delayed && !usingShellTransitions) {
        // We aren't delayed anything, but exiting windows rely on the animation finished
        // callback being called in case the ActivityRecord was pretending to be delayed,
        // which we might have done because we were in closing/opening apps list.
        onAnimationFinished(ANIMATION_TYPE_APP_TRANSITION, null /* AnimationAdapter */);
        if (visible) {
            // The token was made immediately visible, there will be no entrance animation.
            // We need to inform the client the enter animation was finished.
            mEnteringAnimation = true;
            mWmService.mActivityManagerAppTransitionNotifier.onAppTransitionFinishedLocked(
                    token);
        }
    }

    // If we're becoming visible, immediately change client visibility as well. there seem
    // to be some edge cases where we change our visibility but client visibility never gets
    // updated.
    // If we're becoming invisible, update the client visibility if we are not running an
    // animation. Otherwise, we'll update client visibility in onAnimationFinished.
    if (visible || !isAnimating(PARENTS, ANIMATION_TYPE_APP_TRANSITION | ANIMATION_TYPE_RECENTS)
            || usingShellTransitions) {
        setClientVisible(visible);
    }
```

ActivityRecord.onAnimationFinished函数去尝试调用setClientVisible：

```
@Override
protected void onAnimationFinished(@AnimationType int type, AnimationAdapter anim) {
    super.onAnimationFinished(type, anim);

    Trace.traceBegin(TRACE_TAG_WINDOW_MANAGER, "AR#onAnimationFinished");
    mTransit = TRANSIT_OLD_UNSET;
    mTransitFlags = 0;
    mNeedsAnimationBoundsLayer = false;

    setAppLayoutChanges(FINISH_LAYOUT_REDO_ANIM | FINISH_LAYOUT_REDO_WALLPAPER,
            "ActivityRecord");

    clearThumbnail();
    setClientVisible(isVisible() || mVisibleRequested);

    getDisplayContent().computeImeTargetIfNeeded(this);

    ProtoLog.v(WM_DEBUG_ANIM, "Animation done in %s"
            + ": reportedVisible=%b okToDisplay=%b okToAnimate=%b startingDisplayed=%b",
            this, reportedVisible, okToDisplay(), okToAnimate(), startingDisplayed);
```

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：

```
}
                    setVisibility
// In the case where we are making an app visible but holding off for
// we still need to tell the client to make its windows visible so th
// Otherwise, we will wait on performing the transition until all win
// drawn, they never will be, and we are sad.
setClientVisible(true);

requestUpdateWallpaperIfNeeded();

ProtoLog.v(WM_DEBUG_ADD_REMOVE, "No longer Stopped: %s", this);
mAppStopped = false;
```

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():

```
private void handleOpeningApps() {
    final ArraySet<ActivityRecord> openingApps = mDisplayContent.mOpeningApps;
    final int appsCount = openingApps.size();

    for (int i = 0; i < appsCount; i++) {
        final ActivityRecord app = openingApps.valueAt(i);
        ProtoLog.v(WM_DEBUG_APP_TRANSITIONS, "Now opening app %s", app);

        app.commitVisibility(true /* visible */, false /* performLayout */);

        // In case a trampoline activity is used, it can happen that a new ActivityRecord is
        // added and a new app transition starts before the previous app transition animation
        // ends. So we cannot simply use app.isAnimating(PARENTS) to determine if the app must
        // to be added to the list of tokens to be notified of app transition complete.
        final WindowContainer wc = app.getAnimatingContainer(PARENTS,
                ANIMATION_TYPE_APP_TRANSITION);
        if (wc == null || !wc.getAnimationSources().contains(app)) {
            // This token isn't going to be animating. Add it to the list of tokens to
            // be notified of app transition complete since the notification will not be
            // sent be the app window animator.
            mDisplayContent.mNoAnimationNotifyOnTransitionFinished.add(app.token);
        }
        app.updateReportedVisibilityLocked();
        app.waitingToShow = false;
```

handleClosingApps():

```
private void handleClosingApps() {
    final ArraySet<ActivityRecord> closingApps = mDisplayContent.mClosingApps;
    final int appsCount = closingApps.size();

    for (int i = 0; i < appsCount; i++) {
        final ActivityRecord app = closingApps.valueAt(i);
        ProtoLog.v(WM_DEBUG_APP_TRANSITIONS, "Now closing app %s", app);

        app.commitVisibility(false /* visible */, false /* performLayout */);
        app.updateReportedVisibilityLocked();
        // Force the allDrawn flag, because we want to start
        // this guy's animations regardless of whether it's
        // gotten drawn.
        app.allDrawn = true;
        // Ensure that apps that are mid-starting are also scheduled to have their
        // starting windows removed after the animation is complete
        if (app.mStartingWindow != null && !app.mStartingWindow.mAnimatingExit) {
            app.removeStartingWindow();
        }

        if (mDisplayContent.mAppTransition.isNextAppTransitionThumbnailDown()) {
            app.attachThumbnailAnimation();
        }
    }
}
```

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。

```
void goodToGo(@WindowManager.TransitionOldType int transit) {
    ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "goodToGo()");
    if (mCanceled) {
        ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS,
                "goodToGo(): Animation canceled already");
        onAnimationFinished();
        invokeAnimationCancelled("already_cancelled");
        return;
    }

    // Scale the timeout with the animator scale the controlling app is using.
    mHandler.postDelayed(mTimeoutRunnable,
            (long) (TIMEOUT_MS * mService.getCurrentAnimatorScale()));
    mFinishedCallback = new FinishedCallback(this);

    // Create the app targets
    final RemoteAnimationTarget[] appTargets = createAppAnimations();
    if (appTargets.length == 0 && !AppTransition.isKeyguardOccludeTransitOld(transit)) {
        // Keyguard occlude transition can be executed before the occluding activity becomes
        // visible. Even in this case, KeyguardService expects to receive binder call, so we
        // don't cancel remote animation.
        ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS,
                "goodToGo(): No apps to animate, mPendingAnimations=%d",
                mPendingAnimations.size());
        onAnimationFinished();
        invokeAnimationCancelled("no_app_targets");
        return;
    }

    if (mOnRemoteAnimationReady != null) {
        mOnRemoteAnimationReady.run();
        mOnRemoteAnimationReady = null;
    }

    // Create the remote wallpaper animation targets (if any)
    final RemoteAnimationTarget[] wallpaperTargets = createWallpaperAnimations();

    // Create the remote non app animation targets (if any)
    final RemoteAnimationTarget[] nonAppTargets = createNonAppWindowAnimations(transit);

    mService.mAnimator.addAfterPrepareSurfacesRunnable(() -> {
        try {
            linkToDeathOfRunner();
            ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "goodToGo(): onAnimationStart,"
                            + " transit=%s, apps=%d, wallpapers=%d, nonApps=%d",
                    AppTransition.appTransitionOldToString(transit), appTargets.length,
                    wallpaperTargets.length, nonAppTargets.length);
            mRemoteAnimationAdapter.getRunner().onAnimationStart(transit, appTargets,
                    wallpaperTargets, nonAppTargets, mFinishedCallback);
        } catch (RemoteException e) {
            Slog.e(TAG, "Failed to start remote animation", e);
            onAnimationFinished();
        }
        if (ProtoLogImpl.isEnabled(WM_DEBUG_REMOTE_ANIMATIONS)) {
            ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "startAnimation(): Notify animation start:");
            writeStartDebugStatement();
        }
    });
    setRunningRemoteAnimation(true);
}

void cancelAnimation(String reason) {
```

createAppAnimations:

```
private RemoteAnimationTarget[] createAppAnimations() {
    ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "createAppAnimations()");
    final ArrayList<RemoteAnimationTarget> targets = new ArrayList<>();
    for (int i = mPendingAnimations.size() - 1; i >= 0; i--) {
        final RemoteAnimationRecord wrappers = mPendingAnimations.get(i);
        final RemoteAnimationTarget target = wrappers.createRemoteAnimationTarget();
        if (target != null) {
            ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "\tAdd container=%s",
                    wrappers.mWindowContainer);
            targets.add(target);
        } else {
            ProtoLog.d(WM_DEBUG_REMOTE_ANIMATIONS, "\tRemove container=%s",
                    wrappers.mWindowContainer);

            // We can't really start an animation but we still need to make sure to finish the
            // pending animation that was started by SurfaceAnimator
            if (wrappers.mAdapter != null
                    && wrappers.mAdapter.mCapturedFinishCallback != null) {
                wrappers.mAdapter.mCapturedFinishCallback
                        .onAnimationFinished(wrappers.mAdapter.mAnimationType,
                                wrappers.mAdapter);
            }
            if (wrappers.mThumbnailAdapter != null
                    && wrappers.mThumbnailAdapter.mCapturedFinishCallback != null) {
                wrappers.mThumbnailAdapter.mCapturedFinishCallback
                        .onAnimationFinished(wrappers.mThumbnailAdapter.mAnimationType,
                                wrappers.mThumbnailAdapter);
            }
            mPendingAnimations.remove(i);
        }
    }
    return targets.toArray(new RemoteAnimationTarget[targets.size()]);
}
```

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

```
        // Set focus to the top running activity of this task and move all its parents to top.
        top.moveFocusableActivityToTop(reason);

        if (DEBUG_TRANSITION) Slog.v(TAG_TRANSITION, "Prepare to front transition: task=" + tr);
        if (noAnimation) {
            mDisplayContent.prepareAppTransition(TRANSIT_NONE);
            mTaskSupervisor.mNoAnimActivities.add(top);
            ActivityOptions.abort(options);
        } else {
            updateTransitLocked(TRANSIT_TO_FRONT, options);
        }

        // If a new task is moved to the front, then mark the existing top activity as
```

```
    private void updateTransitLocked(@WindowManager.TransitionType int transit,
            ActivityOptions options) {
        if (options != null) {
            ActivityRecord r = topRunningActivity();
            if (r != null && !r.isState(RESUMED)) {
                r.updateOptionsLocked(options);
            } else {
                ActivityOptions.abort(options);
            }
        }
        mDisplayContent.prepareAppTransition(transit);
    }
```

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：

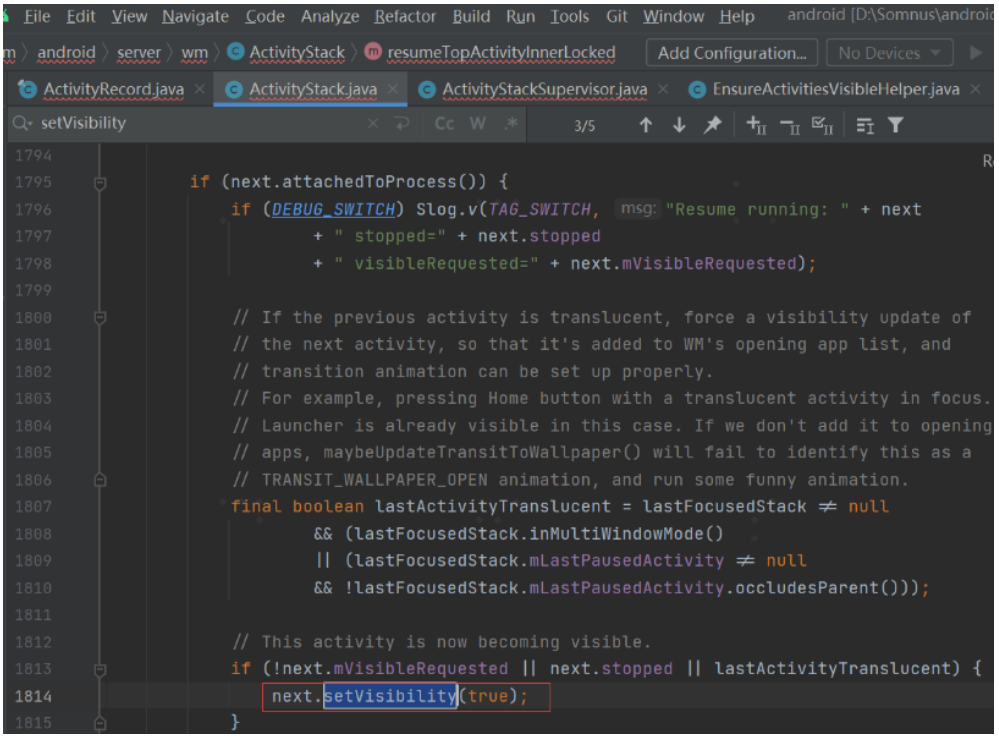
09-08 09:42:07.741 1479 3574 V WindowManager: Prepare app transition: transit=TRANSIT_TASK_OPEN mNextAppTransition=TRANSIT_TASK_TO_FRONT alwaysKeepCurrent=false displayId=0 Callers=com.android.server.wm.DisplayContent.prepareAppTransition:5168 com.android.server.wm.DisplayContent.prepareAppTransition:5162 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)：

```
        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.
        mConfigChanges |= r.configChangeFlags;
    } else {
        if (DEBUG_VISIBILITY) {
            Slog.v(TAG_VISIBILITY, "Make invisible? " + r
                    + " finishing=" + r.finishing + " state=" + r.getState()
                    + " containerShouldBeVisible=" + mContainerShouldBeVisible
                    + " behindFullyOccludedContainer=" + mBehindFullyOccludedContainer
                    + " mLaunchTaskBehind=" + r.mLaunchTaskBehind);
        }
        r.makeInvisible();
    }
```

图库走这里

桌面走这里

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:

```
@VisibleForTesting
void setVisibility(boolean visible, boolean deferHidingClient) {
    final AppTransition appTransition = getDisplayContent().mAppTransition;

    // Don't set visibility to false if we were already not visible. This prevents WM from
    // adding the app to the closing app list which doesn't make sense for something that is
    // already not visible. However, set visibility to true even if we are already visible.
    // This makes sure the app is added to the opening apps list so that the right
    // transition can be selected.
    // TODO: Probably a good idea to separate the concept of opening/closing apps from the
    // concept of setting visibility...
    if (!visible && !mVisibleRequested) {

        if (!deferHidingClient && mLastDeferHidingClient) {
            // We previously deferred telling the client to hide itself when visibility was
            // initially set to false. Now we would like it to hide, so go ahead and set it.
            mLastDeferHidingClient = deferHidingClient;
            setClientVisible(false);
        }
        return;
    }

    ProtoLog.v(WM_DEBUG_APP_TRANSITIONS,
            "setAppVisibility(%s, visible=%b): %s visible=%b mVisibleRequested=%b Callers=%s",
            token, visible, appTransition, isVisible(), mVisibleRequested,
            Debug.getCallers(6));

    // Before setting mVisibleRequested so we can track changes.
    mTransitionController.collect(this);

    onChildVisibilityRequested(visible);

    final DisplayContent displayContent = getDisplayContent();
    displayContent.mOpeningApps.remove(this);
    displayContent.mClosingApps.remove(this);
    waitingToShow = false;
    setVisibleRequested(visible);
    mLastDeferHidingClient = deferHidingClient;

    if (!visible) {
        // If the app is dead while it was visible, we kept its dead window on screen.
        // Now that the app is going invisible, we can remove it. It will be restarted
        // if made visible again.
        removeDeadWindows();
        // If this activity is about to finish/stopped and now becomes invisible, remove it
        // from the unknownApp list in case the activity does not want to draw anything, which
        // keep the user waiting for the next transition to start.
        if (finishing || isState(STOPPED)) {
            displayContent.mUnknownAppVisibilityController.appRemovedOrHidden(this);
        }
    } else {
        if (!appTransition.isTransitionSet()
                && appTransition.isReady()) {
            // Add the app mOpeningApps if transition is unset but ready. This means
            // we're doing a screen freeze, and the unfreeze will wait for all opening
            // apps to be ready.
            displayContent.mOpeningApps.add(this);
        }
        startingMoved = false;
```