1. **Window animate**

setFocusedApp:

AMS通过startActivity函数调用该函数：

WindowManager: Input focus has changed to null

InputDispatcher: FocusL:Window{facbf56 u0 com.htc.launcher/com.htc.launcher.Launcher}

public void setFocusedApp(IBinder token, boolean moveFocusNow) {

synchronized(mWindowMap) {

final AppWindowToken newFocus;

if (token == null) {

newFocus = null;

} else {

**//获取当前新窗口的AppWindowToken**

newFocus = findAppWindowToken(token);

}

final boolean changed = mFocusedApp != newFocus;

if (changed) {

**//将当前新窗口赋值给mFocusedApp**

**mFocusedApp** = newFocus;

mInputMonitor.setFocusedAppLw(newFocus);

setFocusedStackFrame();

SurfaceControl.openTransaction();

try {

setFocusedStackLayer();

} finally {

SurfaceControl.closeTransaction();

}

}

if (moveFocusNow && changed) {

final long origId = Binder.clearCallingIdentity();

**//计算焦点窗口为mCurrentFocus赋值**

updateFocusedWindowLocked(UPDATE\_FOCUS\_NORMAL, true /\*updateInputWindows\*/);

Binder.restoreCallingIdentity(origId);

}

}

}

updateFocusedWindowLocked：

private boolean updateFocusedWindowLocked(int mode, boolean updateInputWindows) {

**//计算新的焦点窗口为mCurrentFocus 赋值**

WindowState newFocus = computeFocusedWindowLocked();

if (mCurrentFocus != newFocus) {

final WindowState oldFocus = mCurrentFocus;

**mCurrentFocus** = newFocus;

mLosingFocus.remove(newFocus);

int focusChanged = mPolicy.focusChangedLw(oldFocus, newFocus);

if (mode != UPDATE\_FOCUS\_WILL\_ASSIGN\_LAYERS) {

**//此处mCurrentFocus = null**

mInputMonitor.setInputFocusLw(**mCurrentFocus**, updateInputWindows);

}

return true;

}

return false;

}

computeFocusedWindowLocked：

private WindowState computeFocusedWindowLocked() {

final int displayCount = mDisplayContents.size();

for (int i = 0; i < displayCount; i++) {

final DisplayContent displayContent = mDisplayContents.valueAt(i);

WindowState win = findFocusedWindowLocked(displayContent);

if (win != null) {

return win;

}

}

return null;

}

findFocusedWindowLocked：

private WindowState findFocusedWindowLocked(DisplayContent displayContent) {

final WindowList windows = displayContent.getWindowList();

for (int i = windows.size() - 1; i >= 0; i--) {

final WindowState win = windows.get(i);

**//使用View.VISIBLE做完relayoutwindow的窗口返回true**

**//该处只有切换新窗口之前的焦点窗口返回true**

if (!win.canReceiveKeys()) {

continue;

}

AppWindowToken wtoken = win.mAppToken;

// If this window's application has been removed, just skip it.

if (wtoken != null && (wtoken.removed || wtoken.sendingToBottom)) {

continue;

}

if (wtoken != null && win.mAttrs.type != TYPE\_APPLICATION\_STARTING &&

mFocusedApp != null) {

ArrayList<Task> tasks = displayContent.getTasks();

for (int taskNdx = tasks.size() - 1; taskNdx >= 0; --taskNdx) {

AppTokenList tokens = tasks.get(taskNdx).mAppTokens;

int tokenNdx = tokens.size() - 1;

for ( ; tokenNdx >= 0; --tokenNdx) {

final AppWindowToken token = tokens.get(tokenNdx);

**//遍历所有的窗口返回当前的canReceiveKeys的焦点窗口**

if (wtoken == token) {

break;

}

**//当之前设置的mFocusedApp 不为canReceiveKeys窗口返回null**

if (mFocusedApp == token) {

**//此处返回null**

return null;

}

}

}

}

if (DEBUG\_FOCUS\_LIGHT) Slog.v(TAG, "findFocusedWindow: Found new focus @ " + i + " = " + win);

return win;

}

if (DEBUG\_FOCUS\_LIGHT) Slog.v(TAG, "findFocusedWindow: No focusable windows.");

return null;

}

prepareAppTransition：

public void prepareAppTransition(int transit, boolean alwaysKeepCurrent) {

synchronized(mWindowMap) {

**//ams调用prepareAppTransition函数设置mNextAppTransition (mAppTransition.setAppTransition)**

if (!mAppTransition.isTransitionSet() || mAppTransition.isTransitionNone()) {

mAppTransition.setAppTransition(transit);

} else if (!alwaysKeepCurrent) {

if (transit == AppTransition.TRANSIT\_TASK\_OPEN

&& mAppTransition.isTransitionEqual(

AppTransition.TRANSIT\_TASK\_CLOSE)) {

// Opening a new task always supersedes a close for the anim.

mAppTransition.setAppTransition(transit);

} else if (transit == AppTransition.TRANSIT\_ACTIVITY\_OPEN

&& mAppTransition.isTransitionEqual(

AppTransition.TRANSIT\_ACTIVITY\_CLOSE)) {

// Opening a new activity always supersedes a close for the anim.

mAppTransition.setAppTransition(transit);

}

}

if (okToDisplay() && mAppTransition.prepare()) {

mSkipAppTransitionAnimation = false;

}

**//发送APP\_TRANSITION\_TIMEOUT消息.**

if (mAppTransition.isTransitionSet()) {

mH.removeMessages(H.APP\_TRANSITION\_TIMEOUT);

mH.sendEmptyMessageDelayed(H.APP\_TRANSITION\_TIMEOUT, 5000);

}

}

}

setAppVisibility:

**//打开窗口调用setAppVisibility：**

WindowManager: setAppVisibility(Token{c7e9071 ActivityRecord{937cdfb u0 com.example.testfl/.MainActivity t163}}, visible=true):

mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT hidden=true hiddenRequested=true

Callers=com.android.server.am.ActivityStack.resumeTopActivityInnerLocked:1954

com.android.server.am.ActivityStack.resumeTopActivityLocked:1637

com.android.server.am.ActivityStackSupervisor.resumeTopActivitiesLocked:2858

com.android.server.am.ActivityStack.completePauseLocked:1098

com.android.server.am.ActivityStack.activityPausedLocked:984

com.android.server.am.ActivityManagerService.activityPaused:6972

android.app.ActivityManagerNative.onTransact:544

com.android.server.am.ActivityManagerService.onTransact:2696

android.os.Binder.execTransact:461

**//关闭窗口调用setAppVisibility**

WindowManager: setAppVisibility(Token{cd4849c ActivityRecord{ec72c0f u0 com.htc.launcher/.Launcher t160}}, visible=false):

mNextAppTransition=TRANSIT\_TASK\_TO\_FRONT hidden=false hiddenRequested=false

Callers=com.android.server.am.ActivityStack.setVisible:1196

com.android.server.am.ActivityStack.ensureActivitiesVisibleLocked:1469

com.android.server.am.ActivityStackSupervisor.ensureActivitiesVisibleLocked:3455

com.android.server.am.ActivityManagerService.updateConfigurationLocked:18576

com.android.server.am.ActivityStack.resumeTopActivityInnerLocked:1984

com.android.server.am.ActivityStack.resumeTopActivityLocked:1637

com.android.server.am.ActivityStackSupervisor.resumeTopActivitiesLocked:2858

com.android.server.am.ActivityStack.completePauseLocked:1098

com.android.server.am.ActivityStack.activityPausedLocked:984

com.android.server.am.ActivityManagerService.activityPaused:6972

android.app.ActivityManagerNative.onTransact:544

public void setAppVisibility(IBinder token, boolean visible) {

AppWindowToken wtoken;

synchronized(mWindowMap) {

wtoken = findAppWindowToken(token);

**//将该窗口从mOpeningApps和mClosingApps容器中删除**

mOpeningApps.remove(wtoken);

mClosingApps.remove(wtoken);

wtoken.waitingToShow = false;

wtoken.hiddenRequested = !visible;

if (okToDisplay() && mAppTransition.isTransitionSet()) {

**//如果该窗口被设置可见加入到mOpeningApps**

if (visible) {

mOpeningApps.add(wtoken);

wtoken.startingMoved = false;

wtoken.mEnteringAnimation = true;

if (wtoken.hidden) {

wtoken.allDrawn = false;

wtoken.deferClearAllDrawn = false;

wtoken.waitingToShow = true;

if (wtoken.clientHidden) {

wtoken.clientHidden = false;

wtoken.sendAppVisibilityToClients();

}

}

} else {

**//如果该窗口被设置不可见加入到mClosingApps**

mClosingApps.add(wtoken);

wtoken.mEnteringAnimation = false;

}

}

}

executeAppTransition:

public void executeAppTransition() {

synchronized(mWindowMap) {

//**setAppTransition后改变mAppTransitionState = APP\_STATE\_READY**

**//同时请求一次layout**

if (mAppTransition.isTransitionSet()) {

**//mAppTransitionState = APP\_STATE\_READY**

mAppTransition.setReady();

final long origId = Binder.clearCallingIdentity();

try {

**//请求一次layout**

performLayoutAndPlaceSurfacesLocked();

} finally {

Binder.restoreCallingIdentity(origId);

}

}

}

}

performLayoutAndPlaceSurfacesLockedInner：

1.performLayoutLockedInner：

由于displayContent.layoutNeeded为false不会调用performLayoutLockedInner做compute frame。

2.针对做完relayoutwindow 的窗口即mHasSurface为true，调用commitFinishDrawingLocked函数来将当前mDrawState为DRAW\_PENDING（调用完relayoutwindow ）设置为READY\_TO\_SHOW。

3.调用handleAppTransitionReadyLocked检查当前窗口的wtoken.allDrawn是否为true。

relayoutWindow：

为WindowState的mViewVisibility赋值，mDrawState = DRAW\_PENDING;

通过WindowState的mViewVisibility之前的值和relayoutwindow参数中的viewVisibility做比较来判断focusMayChange。

当两个值不一样时调用updateFocusedWindowLocked函数进行焦点的切换。

由于当前打开的新窗口已经做完了relayoutwindow操作，所以新窗口的canReceiveKeys函数返回true，所以可以找到新的焦点窗口，更新mCurrentFocus的值为新打开的窗口。

relayoutWindow会调用performLayoutAndPlaceSurfacesLockedInner函数进行layout的动作，由于新窗口的atoken.hidden为true所以窗口处于gone状态,所以新打开的窗口不会做compute layout的动作。

InputDispatcher:FocusE:Window{8b470afu0com.example.testfl/com.example.testfl.MainActivity}

设置焦点窗口。

App调用finishDrawingWindow函数表明app已经绘制完成：

public void finishDrawingWindow(Session session, IWindow client) {

synchronized (mWindowMap) {

**//查找当前绘制完成窗口的windowstate**

WindowState win = windowForClientLocked(session, client, false);

**//调用finishDrawingLocked函数将mDrawState = COMMIT\_DRAW\_PENDING**

if (win != null && win.mWinAnimator.**finishDrawingLocked**()) {

if ((win.mAttrs.flags & FLAG\_SHOW\_WALLPAPER) != 0) {

getDefaultDisplayContentLocked().pendingLayoutChanges |=

WindowManagerPolicy.FINISH\_LAYOUT\_REDO\_WALLPAPER;

}

final DisplayContent displayContent = win.getDisplayContent();

if (displayContent != null) {

displayContent.layoutNeeded = true;

}

**//请求DO\_TRAVERSAL**

requestTraversalLocked();

}

}

}

在performLayoutAndPlaceSurfacesLockedInner函数中因为打开的窗口的mHasSurface为true，

调用commitFinishDrawingLocked函数将mDrawState = READY\_TO\_SHOW。

通过函数w.isDrawnLw()来判断窗口是否绘制完成当绘制完成将updateAllDrawn = true，同时调用updateAllDrawnLocked函数来更新打开窗口的wtoken.allDrawn = true。

final int N = windows.size();

for (i=N-1; i>=0; i--) {

WindowState w = windows.get(i);

if (w.mHasSurface) {

final boolean committed =

winAnimator.commitFinishDrawingLocked();

}

if (w != atoken.startingWindow) {

if (w.isDrawnLw()) {

token.numDrawnWindows++;

updateAllDrawn = true;

}

if (updateAllDrawn) {

updateAllDrawnLocked(displayContent);

}

}

if (mAppTransition.isReady()) {

defaultDisplay.pendingLayoutChanges |= handleAppTransitionReadyLocked(defaultWindows);

}

通过调用handleAppTransitionReadyLocked函数检查mOpeningApps中的窗口的是否绘制完成来goodToGo。

遍历mOpeningApps容器调用setTokenVisibilityLocked其visible= true。

setTokenVisibilityLocked:

boolean setTokenVisibilityLocked(AppWindowToken wtoken, WindowManager.LayoutParams lp,

boolean **visible**, int transit, boolean performLayout, boolean isVoiceInteraction) {

**1:通知app端visible发生改变**

if (wtoken.clientHidden == visible && !wtoken.isAppVisibilitySilentToClient) {

wtoken.clientHidden = !visible;

wtoken.sendAppVisibilityToClients();

} else if (DEBUG\_HTC && wtoken.isAppVisibilitySilentToClient) {

Slog.v(TAG, "Skip sending app visibility to clients: " + wtoken);

}

**2.当窗口的hide状态和设置的可见状态一致说明app的可见状态需要发生改变**

if (wtoken.hidden == visible || (wtoken.hidden && wtoken.mIsExiting)) {

boolean changed = false;

boolean runningAppAnimation = false;

**3.调用applyAnimationLocked函数**

**在该函数中调用mAppTransition.loadAnimation创建Animation**

**同时调用atoken.mAppAnimator.setAnimation为AppWindowAnimator中的animation对象赋值**

if (transit != AppTransition.TRANSIT\_UNSET) {

if (wtoken.mAppAnimator.animation == AppWindowAnimator.sDummyAnimation) {

wtoken.mAppAnimator.animation = null;

}

**//atoken.mAppAnimator.setAnimation(Animation)**

if (applyAnimationLocked(wtoken, lp, transit, visible, isVoiceInteraction)) {

delayed = runningAppAnimation = true;

}

WindowState window = wtoken.findMainWindow();

//TODO (multidisplay): Magnification is supported only for the default display.

if (window != null && mAccessibilityController != null

&& window.getDisplayId() == Display.DEFAULT\_DISPLAY) {

mAccessibilityController.onAppWindowTransitionLocked(window, transit);

}

changed = true;

}

**4.针对不可见的窗口设置成可见的设置changed为true**

final int windowsCount = wtoken.allAppWindows.size();

for (int i = 0; i < windowsCount; i++) {

WindowState win = wtoken.allAppWindows.get(i);

if (win == wtoken.startingWindow) {

continue;

}

if (visible) {

if (!win.isVisibleNow()) {

changed = true;

final DisplayContent displayContent = win.getDisplayContent();

if (displayContent != null) {

displayContent.layoutNeeded = true;

}

}

}

**5.重新设置wtoken.hidden\hiddenRequested的值为performLayout准备**

**wtoken.hidden = wtoken.hiddenRequested = !visible;**

if (!visible) {

unsetAppFreezingScreenLocked(wtoken, true, true);

}

goodToGo之AppWindowAnimator.showAllWindowsLocked()：

boolean showAllWindowsLocked() {

boolean isAnimating = false;

final int NW = mAllAppWinAnimators.size();

for (int i=0; i<NW; i++) {

WindowStateAnimator winAnimator = mAllAppWinAnimators.get(i);

Slog.v(TAG, "performing show on: " + winAnimator);

**//mDrawState = HAS\_DRAWN;**

**//applyEnterAnimationLocked**

**winAnimator.performShowLocked();**

**//isAnimating = true atoken.mAppAnimator.setAnimation**

isAnimating |= winAnimator.isAnimating();

}

return isAnimating;

}

boolean performShowLocked() {

if (mDrawState == READY\_TO\_SHOW && mWin.isReadyForDisplayIgnoringKeyguard()) {

mService.enableScreenIfNeededLocked();

**//为WindowStateAnimator的mAnimation创建Animation对象。**

applyEnterAnimationLocked();

mLastAlpha = -1;

//if (DEBUG\_SURFACE\_TRACE || DEBUG\_ANIM)

Slog.v(TAG, "performShowLocked: mDrawState=HAS\_DRAWN in " + this);

mDrawState = HAS\_DRAWN;

mService.scheduleAnimationLocked();

}

}

applyAnimationLocked：

在该函数中mAppTransition.loadAnimation创建Animation为AppWindowToken.mAppAnimator

设置动画，该动画可以是用户通过overridePendingAppTransition函数自定义的enter、exit的动画，可以是根据transition 过渡类型来使用系统自定义的enter、exit动画。

当window的mDrawState值变为HAS\_DRAWN同时通过applyEnterAnimationLocked函数为

WindowStateAnimator对象创建Animation对象，为窗口动画做准备。在创建的过程中调用mPolicy.selectAnimationLw函数为系统窗口创建Animation对象。

goodToGo之mClosingApps:

调用setTokenVisibilityLocked函数将

**wtoken.hidden = wtoken.hiddenRequested = !visible（true）;**

mOpeningApps.clear();

mClosingApps.clear();

getDefaultDisplayContentLocked().layoutNeeded = true;

由于displayContent.layoutNeeded为true，会调用requestTraversalLocked再一次请求performLayoutandplacesurface：

由于新打开的窗口的gone为false，所以开始对窗口做layoutWindowLw和compute frame的操作。

**调用scheduleAnimationLocked函数进行窗口动画：**

updateAppWindowsLocked：

开始窗口动画或者从运行的动画中获取Transformation，以及检查窗口动画是否结束。

private void updateAppWindowsLocked(int displayId) {

ArrayList<TaskStack> stacks = mService.getDisplayContentLocked(displayId).getStacks();

for (int stackNdx = stacks.size() - 1; stackNdx >= 0; --stackNdx) {

final TaskStack stack = stacks.get(stackNdx);

final ArrayList<Task> tasks = stack.getTasks();

for (int taskNdx = tasks.size() - 1; taskNdx >= 0; --taskNdx) {

final AppTokenList tokens = tasks.get(taskNdx).mAppTokens;

for (int tokenNdx = tokens.size() - 1; tokenNdx >= 0; --tokenNdx) {

final AppWindowAnimator appAnimator = tokens.get(tokenNdx).mAppAnimator;

appAnimator.wasAnimating = appAnimator.animating;

**//使用之前在setTokenVisibilityLocked中为AppWindowAnimator中创建的**

**//animation对象开始做动画**

**//该动画需要在窗口绘制完成mAppToken.allDrawn为true才能开始。**

if (appAnimator.stepAnimationLocked(mCurrentTime, displayId)) {

appAnimator.animating = true;

**mAnimating = mAppWindowAnimating = true;**

} else if (appAnimator.wasAnimating) {

setAppLayoutChanges(appAnimator,

WindowManagerPolicy.FINISH\_LAYOUT\_REDO\_WALLPAPER,

"appToken " + appAnimator.mAppToken + " done", displayId);

}

}

}

}

AppWindowAnimator.stepAnimationLocked

boolean stepAnimationLocked(long currentTime, final int displayId) {

if (mService.okToDisplay()) {

// We will run animations as long as the display isn't frozen.

if ((mAppToken.allDrawn || animating || mAppToken.startingDisplayed)

&& animation != null) {

**//开始该appwindow的动画**

if (!animating) {

long correction = getStartTimeCorrection();

animation.setStartTime(currentTime + correction);

animating = true;

mSkipFirstFrame = false;

}

**//获取该窗口动画的transformation同时检查动画是否还在运行**

if (stepAnimation(currentTime)) {

return true;

}

}

} else if (animation != null) {

animating = true;

animation = null;

}

hasTransformation = false;

if (!animating && animation == null) {

return false;

}

clearAnimation();

animating = false;

transformation.clear();

return false;

}

updateWindowsLocked：

private void updateWindowsLocked(final int displayId) {

++mAnimTransactionSequence;

final WindowList windows = mService.getWindowListLocked(displayId);

**//检查是不是处于锁屏解锁的动作**

**//keyguardGoingAway在wms的keyguardGoingAway函数置为true**

if (mKeyguardGoingAway) {

for (int i = windows.size() - 1; i >= 0; i--) {

WindowState win = windows.get(i);

//检查是否为statusbar窗口

if (!mPolicy.isKeyguardHostWindow(win.mAttrs)) {

continue;

}

final WindowStateAnimator winAnimator = win.mWinAnimator;

**//当statusbar的窗口属性为锁屏的时候而且窗口没有做动画时创建窗口动画**

if ((win.mAttrs.privateFlags & PRIVATE\_FLAG\_KEYGUARD) != 0) {

if (!winAnimator.mAnimating) {

winAnimator.mAnimation = new AlphaAnimation(1.0f, 1.0f);

winAnimator.mAnimation.setDuration(KEYGUARD\_ANIM\_TIMEOUT\_MS);

winAnimator.mAnimationIsEntrance = false;

winAnimator.mAnimationStartTime = -1;

winAnimator.mKeyguardGoingAwayAnimation = true;

}

} else {

mKeyguardGoingAway = false;

winAnimator.clearAnimation();

}

break;

}

}

for (int i = windows.size() - 1; i >= 0; i--) {

WindowState win = windows.get(i);

WindowStateAnimator winAnimator = win.mWinAnimator;

final int flags = win.mAttrs.flags;

boolean canBeForceHidden = mPolicy.canBeForceHidden(win, win.mAttrs);

boolean shouldBeForceHidden = shouldForceHide(win);

if (winAnimator.mSurfaceControl != null) {

final boolean wasAnimating = winAnimator.mWasAnimating;

final boolean nowAnimating = winAnimator.stepAnimationLocked(mCurrentTime);

winAnimator.mWasAnimating = nowAnimating;

Slog.i(TAG,"updateWindowsLocked mAnimating11111" + mAnimating);

**mAnimating |= nowAnimating;**

Slog.i(TAG,"updateWindowsLocked mAnimating22222" + mAnimating);

boolean appWindowAnimating = winAnimator.mAppAnimator != null

&& winAnimator.mAppAnimator.animating;

boolean wasAppWindowAnimating = winAnimator.mAppAnimator != null

&& winAnimator.mAppAnimator.wasAnimating;

boolean anyAnimating = appWindowAnimating || nowAnimating;

boolean anyWasAnimating = wasAppWindowAnimating || wasAnimating;

if (anyAnimating && !anyWasAnimating) {

win.mClient.onAnimationStarted(winAnimator.mAnimatingMove ? -1

: winAnimator.mKeyguardGoingAwayAnimation ? 1

: 0);

} else if (!anyAnimating && anyWasAnimating) {

win.mClient.onAnimationStopped();

}

updateWallpaperLocked：

prepareSurfaceLocked：

在做窗口动画的过程中调用prepareSurfaceLocked函数进行surface属性的设置。

computeShownFrameLocked

void computeShownFrameLocked() {

**//获取mAppAnimator的动画过程中的Transformation**

**//该Transformation的值在updateAppWindowsLocked函数中调用stepAnimationLocked更新**

**//boolean hasMoreFrames = animation.getTransformation(currentTime, transformation);**

**//hasTransformation = hasMoreFrames;**

Transformation appTransformation = (mAppAnimator != null && mAppAnimator.hasTransformation)

? mAppAnimator.transformation : null;

if (selfTransformation || attachedTransformation != null

|| appTransformation != null || screenAnimation) {

final Rect frame = mWin.mFrame;

final float tmpFloats[] = mService.mTmpFloats;

final Matrix tmpMatrix = mWin.mTmpMatrix;

if (appTransformation != null) {

tmpMatrix.postConcat(appTransformation.getMatrix());

}

mHaveMatrix = true;

**//获取appTransformation 中的动画矩阵为windowstate中的mShownFrame赋值**

tmpMatrix.getValues(tmpFloats);

mDsDx = tmpFloats[Matrix.MSCALE\_X];

mDtDx = tmpFloats[Matrix.MSKEW\_Y];

mDsDy = tmpFloats[Matrix.MSKEW\_X];

mDtDy = tmpFloats[Matrix.MSCALE\_Y];

float x = tmpFloats[Matrix.MTRANS\_X];

float y = tmpFloats[Matrix.MTRANS\_Y];

int w = frame.width();

int h = frame.height();

mWin.mShownFrame.set(x, y, x+w, y+h);

mShownAlpha = mAlpha;

**//通过appTransformation.getAlpha()获取动画过渡对象中的透明度值**

if (!mService.mLimitedAlphaCompositing

|| (!PixelFormat.formatHasAlpha(mWin.mAttrs.format)

|| (mWin.isIdentityMatrix(mDsDx, mDtDx, mDsDy, mDtDy)

&& x == frame.left && y == frame.top))) {

if (appTransformation != null) {

mShownAlpha \*= appTransformation.getAlpha();

if (appTransformation.hasClipRect()) {

mClipRect.set(appTransformation.getClipRect());

if (mWin.mHScale > 0) {

mClipRect.left /= mWin.mHScale;

mClipRect.right /= mWin.mHScale;

}

if (mWin.mVScale > 0) {

mClipRect.top /= mWin.mVScale;

mClipRect.bottom /= mWin.mVScale;

}

mHasClipRect = true;

}

}

if (screenAnimation) {

mShownAlpha \*= screenRotationAnimation.getEnterTransformation().getAlpha();

}

} else {

}

return;

}

**//动画结束使用窗口frame值来为showframe赋值**

mWin.mShownFrame.set(mWin.mFrame);

if (mWin.mXOffset != 0 || mWin.mYOffset != 0) {

mWin.mShownFrame.offset(mWin.mXOffset, mWin.mYOffset);

}

mShownAlpha = mAlpha;

mHaveMatrix = false;

mDsDx = mWin.mGlobalScale;

mDtDx = 0;

mDsDy = 0;

mDtDy = mWin.mGlobalScale;

setSurfaceBoundariesLocked

void setSurfaceBoundariesLocked(final boolean recoveringMemory) {

final WindowState w = mWin;

int width;

int height;

**//使用computeShownFrameLocked中赋值的showframe来设置窗口的左上坐标**

float left = w.mShownFrame.left;

float top = w.mShownFrame.top;

**//当窗口的起始左上坐标不等于新赋值的坐标，调用setPosition设置surface的位置**

final boolean surfaceMoved = mSurfaceX != left || mSurfaceY != top;

if (surfaceMoved) {

mSurfaceX = left;

mSurfaceY = top;

mSurfaceControl.setPosition(left, top);

}

final boolean surfaceResized = mSurfaceW != width || mSurfaceH != height;

if (surfaceResized) {

mSurfaceW = width;

mSurfaceH = height;

mSurfaceResized = true;

mSurfaceControl.setSize(width, height);

mSurfaceControl.setMatrix(

mDsDx \* w.mHScale, mDtDx \* w.mVScale,

mDsDy \* w.mHScale, mDtDy \* w.mVScale);

}

}

prepareSurfaceLocked：

if (mLastLayer != mAnimLayer|| mLastAlpha != mShownAlpha|| mLastDsDx != mDsDx|| mLastDtDx != mDtDx|| mLastDsDy != mDsDy|| mLastDtDy != mDtDy|| w.mLastHScale != w.mHScale|| w.mLastVScale != w.mVScale|| mLastHidden) {

displayed = true;

mLastAlpha = mShownAlpha;

mLastLayer = mAnimLayer;

mLastDsDx = mDsDx;

mLastDtDx = mDtDx;

mLastDsDy = mDsDy;

mLastDtDy = mDtDy;

w.mLastHScale = w.mHScale;

w.mLastVScale = w.mVScale;

if (mSurfaceControl != null) {

mSurfaceAlpha = mShownAlpha;

**//设置透明度**

mSurfaceControl.setAlpha(mShownAlpha);

mSurfaceLayer = mAnimLayer;

**//设置层级**

mSurfaceControl.setLayer(mAnimLayer);

**//设置变换矩阵**

mSurfaceControl.setMatrix(

mDsDx \* w.mHScale, mDtDx \* w.mVScale,

mDsDy \* w.mHScale, mDtDy \* w.mVScale);

**//当在createSurfaceLocked将mLastHidden 置为true后而且绘制完成。**

**//relayoutwindow和drawnfinish之后显示窗口**

if (mLastHidden && mDrawState == HAS\_DRAWN) {

if (showSurfaceRobustlyLocked()) {

mLastHidden = false;

if (mIsWallpaper) {

mService.dispatchWallpaperVisibility(w, true);

}

mAnimator.setPendingLayoutChanges(w.getDisplayId(),

WindowManagerPolicy.FINISH\_LAYOUT\_REDO\_ANIM);

} else {

w.mOrientationChanging = false;

}

}

if (mSurfaceControl != null) {

w.mToken.hasVisible = true;

}

}

}

}

}

updateAllDrawnLocked

handleAppTransitionReadyLocked

getDefaultDisplayContentLocked().layoutNeeded = true;

当有starting窗口动画的时候，打开窗口不做窗口动画。

当没有starting窗口的时候，打开窗口需要进行窗口动画。

因为ams在窗口打开的过程中只会调用executeAppTransition（mAppTransitionState = APP\_STATE\_READY）一次，当starting窗口加载窗口动画的时候goodToGo（mAppTransitionState = APP\_STATE\_RUNNING）导致打开窗口不会做窗口动画（handleAppTransitionReadyLocked调用有mAppTransition.isReady()检查）。