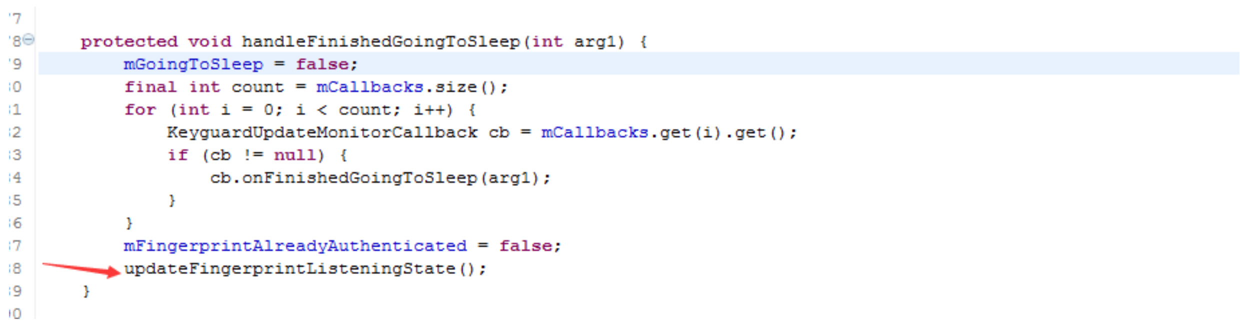
指纹解锁过程

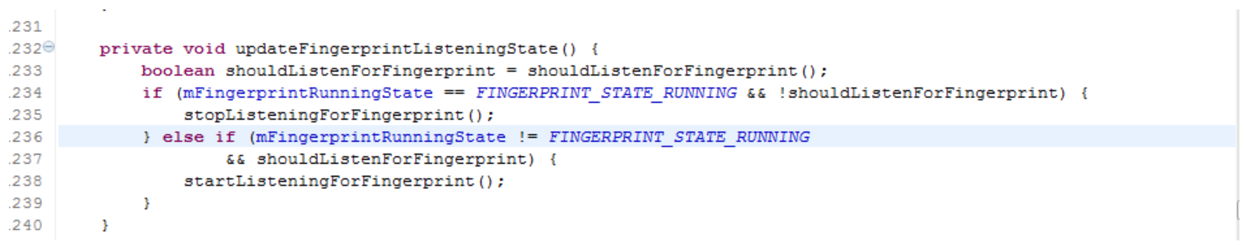
在分析指纹解锁之前我们需要分析SystemUI如何实现指纹监听的

之前我们分析过灭屏下构造锁屏的过程的时候我们会调用
onFinishGoingtoSleep()
最终我们会调用至
handleFinishedGoingToSleep



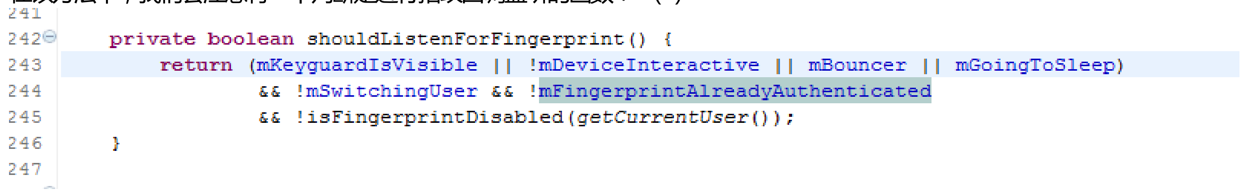
```java
    protected void handleFinishedGoingToSleep(int arg1) {
        mGoingToSleep = false;
        final int count = mCallbacks.size();
        for (int i = 0; i < count; i++) {
            KeyguardUpdateMonitorCallback cb = mCallbacks.get(i).get();
            if (cb != null) {
                cb.onFinishedGoingToSleep(arg1);
            }
        }
        mFingerprintAlreadyAuthenticated = false;
        updateFingerprintListeningState();
    }
```

在这个方法中我们会调用updateFingerprintListeningState()
更新注册监听的状态，并且对mFingerprintAlerdyAuthenticated赋值为false



```java
    private void updateFingerprintListeningState() {
        boolean shouldListenForFingerprint = shouldListenForFingerprint();
        if (mFingerprintRunningState == FINGERPRINT_STATE_RUNNING && !shouldListenForFingerprint) {
            stopListeningForFingerprint();
        } else if (mFingerprintRunningState != FINGERPRINT_STATE_RUNNING
                && shouldListenForFingerprint) {
            startListeningForFingerprint();
        }
    }
```

在该方法中，我们会注意有一个判断是进行指纹回调监听的函数：（ ）



```java
    private boolean shouldListenForFingerprint() {
        return (mKeyguardIsVisible || !mDeviceInteractive || mBouncer || mGoingToSleep)
                && !mSwitchingUser && !mFingerprintAlreadyAuthenticated
                && !isFingerprintDisabled(getCurrentUser());
    }
```

从上面的判断可知，
1 ：我们需要对指纹满足监听的条件是必现是锁屏可见等判断成立

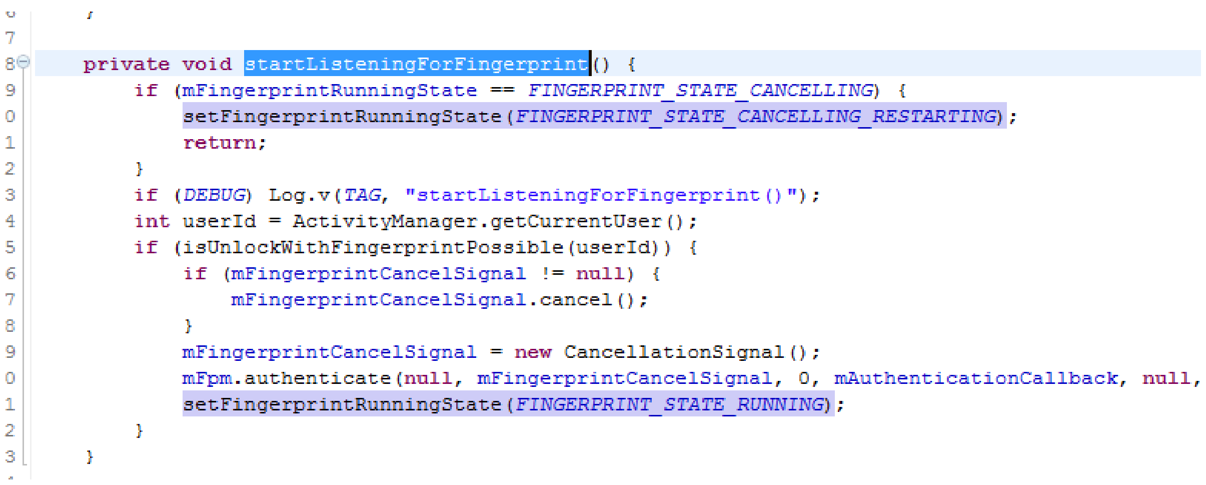
2：我们需要判断当前指纹是否处于FINGERPRINT_STATE_RUNNING状态

 private int mFingerprintRunningState = FINGERPRINT_STATE_STOPPED;

首次进入的时候mFingerprintRunningState的赋值是FINGERPRINT_STATE_STOPPED

所以我们满足第二个条件

走的函数时



```
    private void startListeningForFingerprint() {
        if (mFingerprintRunningState == FINGERPRINT_STATE_CANCELLING) {
            setFingerprintRunningState(FINGERPRINT_STATE_CANCELLING_RESTARTING);
            return;
        }
        if (DEBUG) Log.v(TAG, "startListeningForFingerprint()");
        int userId = ActivityManager.getCurrentUser();
        if (isUnlockWithFingerprintPossible(userId)) {
            if (mFingerprintCancelSignal != null) {
                mFingerprintCancelSignal.cancel();
            }
            mFingerprintCancelSignal = new CancellationSignal();
            mFpm.authenticate(null, mFingerprintCancelSignal, 0, mAuthenticationCallback, null,
            setFingerprintRunningState(FINGERPRINT_STATE_RUNNING);
        }
    }
```

此函数里面做了几件总要的事情：
调用FingerpritManager.authenticate来注册回调AuthenticationCallback

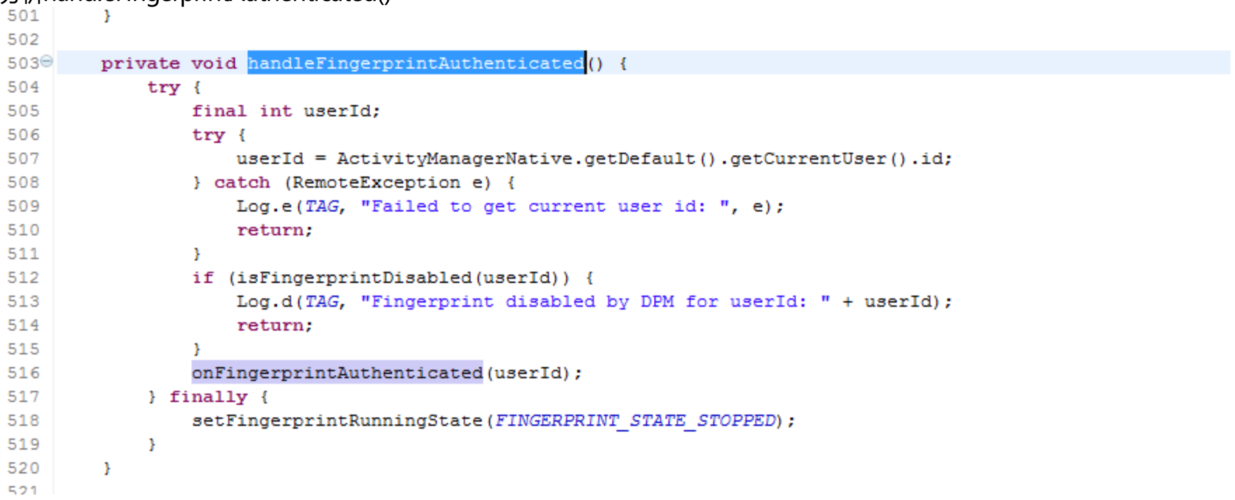
调用方法setFingerprintRunningState()来改变mFingerprintRunningState的状态

指纹解锁：

因为之前我们做个监听了指纹的回调AuthenticationCallback函数
当指纹认证成功过后，我们通过回调如下方法进行解锁操作

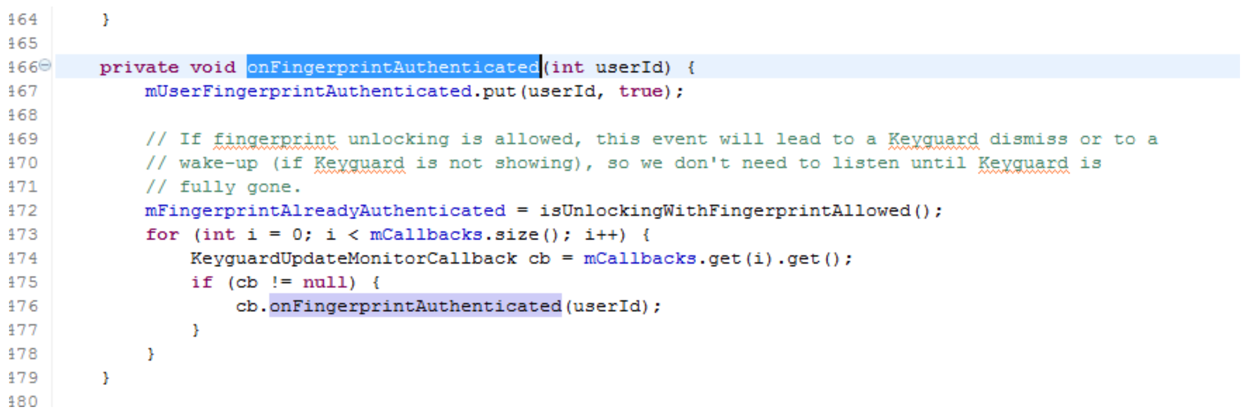
```
@Override
public void onAuthenticationSucceeded(AuthenticationResult result) {
    handleFingerprintAuthenticated();
}
```

分析handleFingerprintAuthenticated()



```
    }

    private void handleFingerprintAuthenticated() {
        try {
            final int userId;
            try {
                userId = ActivityManagerNative.getDefault().getCurrentUser().id;
            } catch (RemoteException e) {
                Log.e(TAG, "Failed to get current user id: ", e);
                return;
            }
            if (isFingerprintDisabled(userId)) {
                Log.d(TAG, "Fingerprint disabled by DPM for userId: " + userId);
                return;
            }
            onFingerprintAuthenticated(userId);
        } finally {
            setFingerprintRunningState(FINGERPRINT_STATE_STOPPED);
        }
    }
```

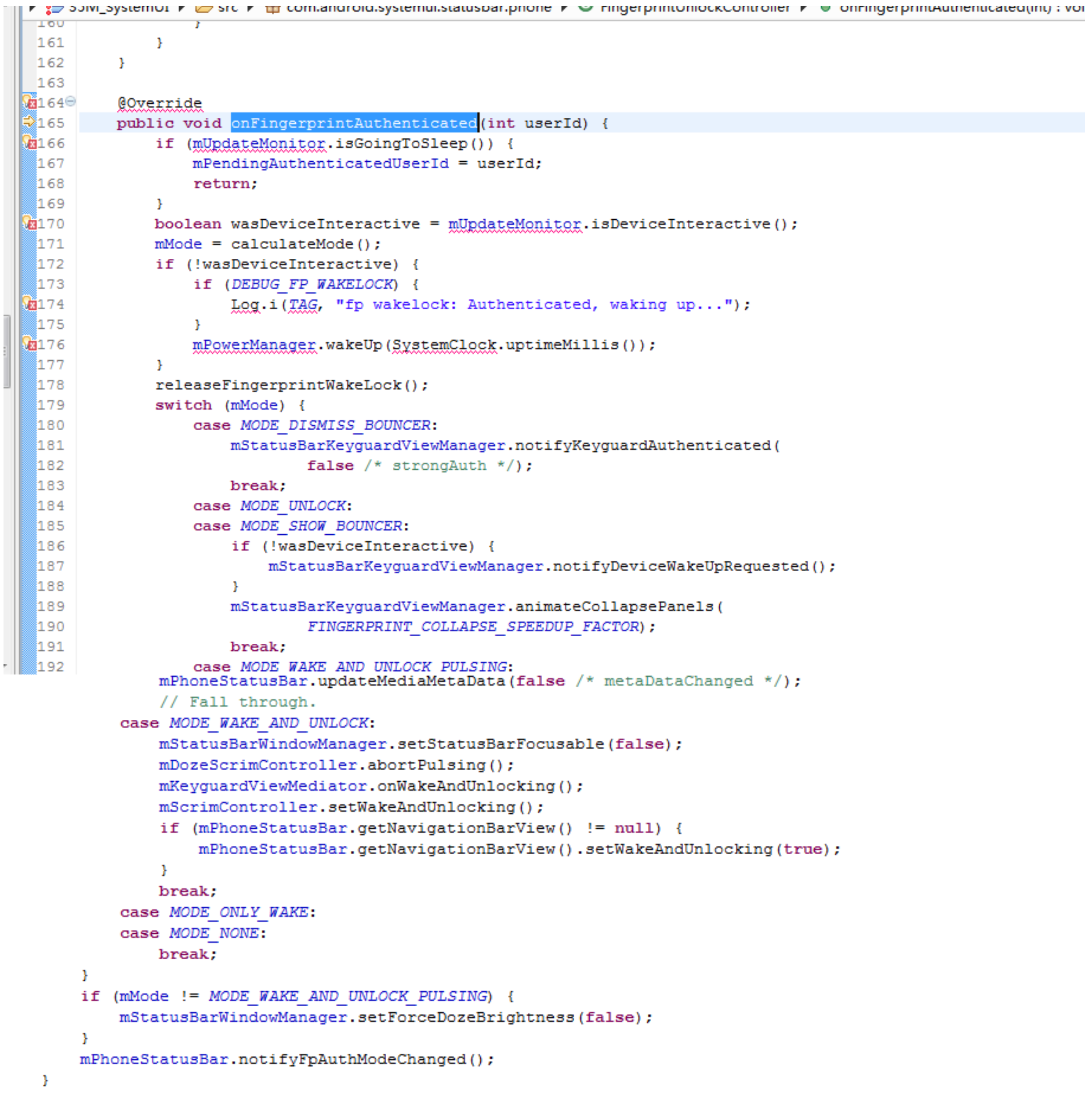
原来还继续调用onFingerprintAuthenticated()



```java
    }

    private void onFingerprintAuthenticated(int userId) {
        mUserFingerprintAuthenticated.put(userId, true);

        // If fingerprint unlocking is allowed, this event will lead to a Keyguard dismiss or to a
        // wake-up (if Keyguard is not showing), so we don't need to listen until Keyguard is
        // fully gone.
        mFingerprintAlreadyAuthenticated = isUnlockingWithFingerprintAllowed();
        for (int i = 0; i < mCallbacks.size(); i++) {
            KeyguardUpdateMonitorCallback cb = mCallbacks.get(i).get();
            if (cb != null) {
                cb.onFingerprintAuthenticated(userId);
            }
        }
    }
```

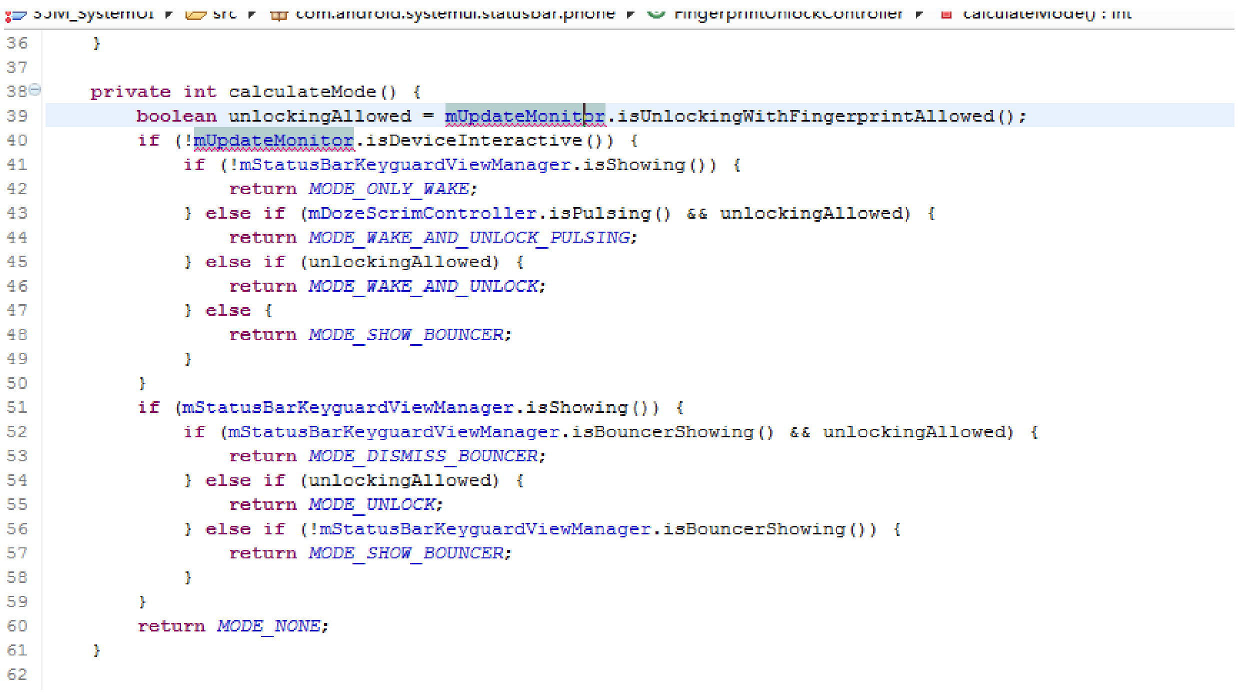
原来是调用了KeyguardUpdateMonitorCallback函数来实现的，以前我们分析了很多锁屏的callback回调接口的调用，现在我们就不分析他是如何注册回调的，我们直接来分析它的调用实现：

FingerprintUnlockControl



```java
        }
    }

    @Override
    public void onFingerprintAuthenticated(int userId) {
        if (mUpdateMonitor.isGoingToSleep()) {
            mPendingAuthenticatedUserId = userId;
            return;
        }
        boolean wasDeviceInteractive = mUpdateMonitor.isDeviceInteractive();
        mMode = calculateMode();
        if (!wasDeviceInteractive) {
            if (DEBUG_FP_WAKELOCK) {
                Log.i(TAG, "fp wakelock: Authenticated, waking up...");
            }
            mPowerManager.wakeUp(SystemClock.uptimeMillis());
        }
        releaseFingerprintWakeLock();
        switch (mMode) {
            case MODE_DISMISS_BOUNCER:
                mStatusBarKeyguardViewManager.notifyKeyguardAuthenticated(
                        false /* strongAuth */);
                break;
            case MODE_UNLOCK:
            case MODE_SHOW_BOUNCER:
                if (!wasDeviceInteractive) {
                    mStatusBarKeyguardViewManager.notifyDeviceWakeUpRequested();
                }
                mStatusBarKeyguardViewManager.animateCollapsePanels(
                        FINGERPRINT_COLLAPSE_SPEEDUP_FACTOR);
                break;
            case MODE_WAKE_AND_UNLOCK_PULSING:
                mPhoneStatusBar.updateMediaMetaData(false /* metaDataChanged */);
                // Fall through.
            case MODE_WAKE_AND_UNLOCK:
                mStatusBarWindowManager.setStatusBarFocusable(false);
                mDozeScrimController.abortPulsing();
                mKeyguardViewMediator.onWakeAndUnlocking();
                mScrimController.setWakeAndUnlocking();
                if (mPhoneStatusBar.getNavigationBarView() != null) {
                    mPhoneStatusBar.getNavigationBarView().setWakeAndUnlocking(true);
                }
                break;
            case MODE_ONLY_WAKE:
            case MODE_NONE:
                break;
        }
        if (mMode != MODE_WAKE_AND_UNLOCK_PULSING) {
            mStatusBarWindowManager.setForceDozeBrightness(false);
        }
        mPhoneStatusBar.notifyFpAuthModeChanged();
    }
```

从上的可知，我们需要获取我们的mode变量的类型，获取方法如下：


SSM_SystemUI ▸ src ▸ com.android.systemui.statusbar.phone ▸ FingerprintUnlockController ▸ calculateMode() : int

```
    }

    private int calculateMode() {
        boolean unlockingAllowed = mUpdateMonitor.isUnlockingWithFingerprintAllowed();
        if (!mUpdateMonitor.isDeviceInteractive()) {
            if (!mStatusBarKeyguardViewManager.isShowing()) {
                return MODE_ONLY_WAKE;
            } else if (mDozeScrimController.isPulsing() && unlockingAllowed) {
                return MODE_WAKE_AND_UNLOCK_PULSING;
            } else if (unlockingAllowed) {
                return MODE_WAKE_AND_UNLOCK;
            } else {
                return MODE_SHOW_BOUNCER;
            }
        }
        if (mStatusBarKeyguardViewManager.isShowing()) {
            if (mStatusBarKeyguardViewManager.isBouncerShowing() && unlockingAllowed) {
                return MODE_DISMISS_BOUNCER;
            } else if (unlockingAllowed) {
                return MODE_UNLOCK;
            } else if (!mStatusBarKeyguardViewManager.isBouncerShowing()) {
                return MODE_SHOW_BOUNCER;
            }
        }
        return MODE_NONE;
    }
```

我们这么多么是，那么这几种模式我们就一一分析：

1:MODE_ONLY_WAKE，仅仅点亮屏幕，这个就是在我们没有锁屏存在的时候调用

2:MODE_WAKE_AND_UNLOCK_PULSING 解锁的过程之一

3：MODE_WAKE_AND_UNLOCK 亮屏解锁

4：MODE_SHOW_BOUNCER 显示锁屏

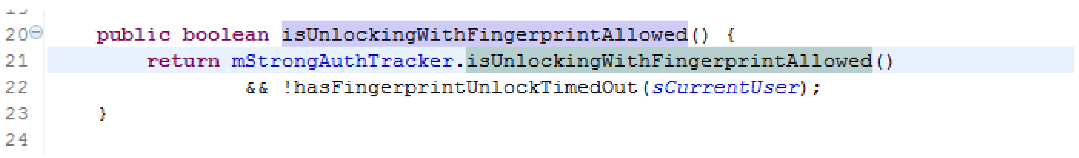
以下三个模式是基于锁屏存在的条件：

1：MODE_DISMISS)BOUNCER 解锁

2:MODE_UNLOCK 解锁

3:MOD_SHOW_BOUNCER

我们来看过重要的参数：



```
    public boolean isUnlockingWithFingerprintAllowed() {
        return mStrongAuthTracker.isUnlockingWithFingerprintAllowed()
                && !hasFingerprintUnlockTimedOut(sCurrentUser);
    }
```

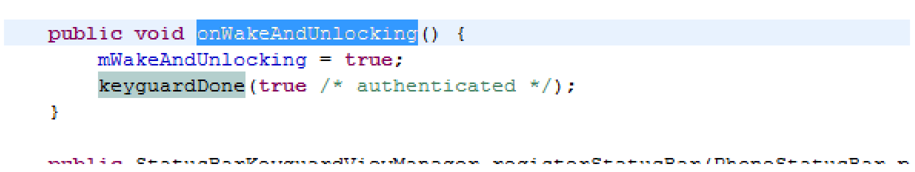
是否允许指纹解锁：

我们现在就来重点分析唤醒解锁：

```
MODE_WAKE_AND_UNLOCK
    mStatusBarWindowManager.setStatusBarFocusable(false);
    mDozeScrimController.abortPulsing();
    mKeyguardViewMediator.onWakeAndUnlocking();
    mScrimController.setWakeAndUnlocking();
    if (mPhoneStatusBar.getNavigationBarView() != null) {
            mPhoneStatusBar.getNavigationBarView().setWakeAndUnlocking(true);
     }
```

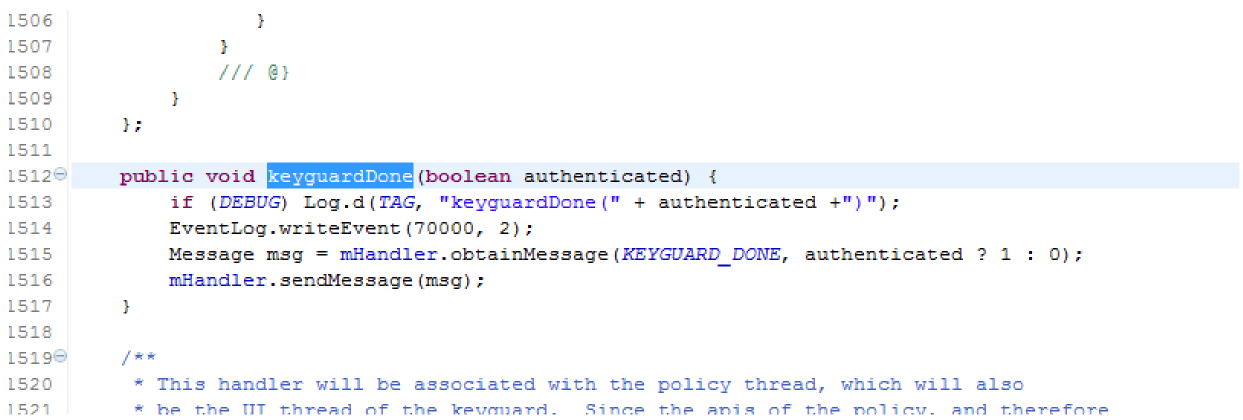
唤醒解锁的话我们主要分析KeyguardViewMediator.onWakeAndUnlocking

KeyguardViewMediator.onWakeAndUnlocking

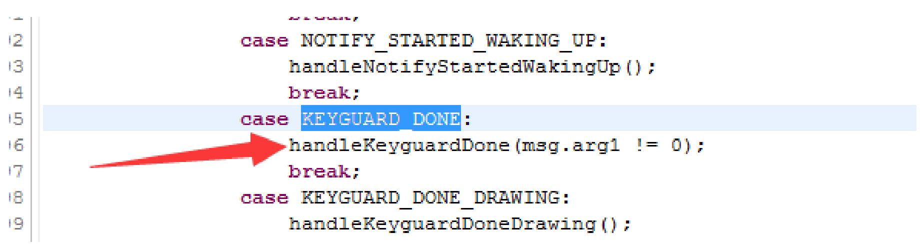


```
    public void onWakeAndUnlocking() {
        mWakeAndUnlocking = true;
        keyguardDone(true /* authenticated */);
    }

    public StatusBarKeyguardViewManager registerStatusBar(PhoneStatusBar p
```

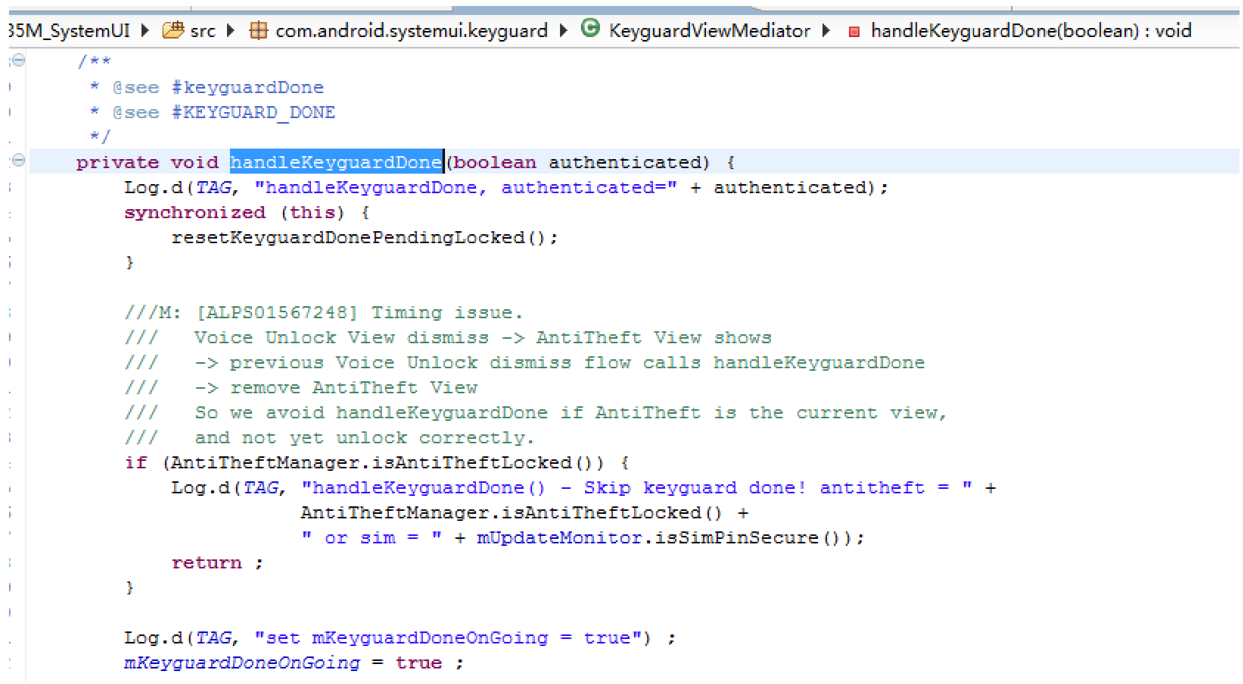
继续调用，那么我们就跟踪下keyguardDone



```
                    }
                }
                /// @}
            }
        };

        public void keyguardDone(boolean authenticated) {
            if (DEBUG) Log.d(TAG, "keyguardDone(" + authenticated +")");
            EventLog.writeEvent(70000, 2);
            Message msg = mHandler.obtainMessage(KEYGUARD_DONE, authenticated ? 1 : 0);
            mHandler.sendMessage(msg);
        }

        /**
         * This handler will be associated with the policy thread, which will also
         * be the UI thread of the keyguard.  Since the apis of the policy, and therefore
```

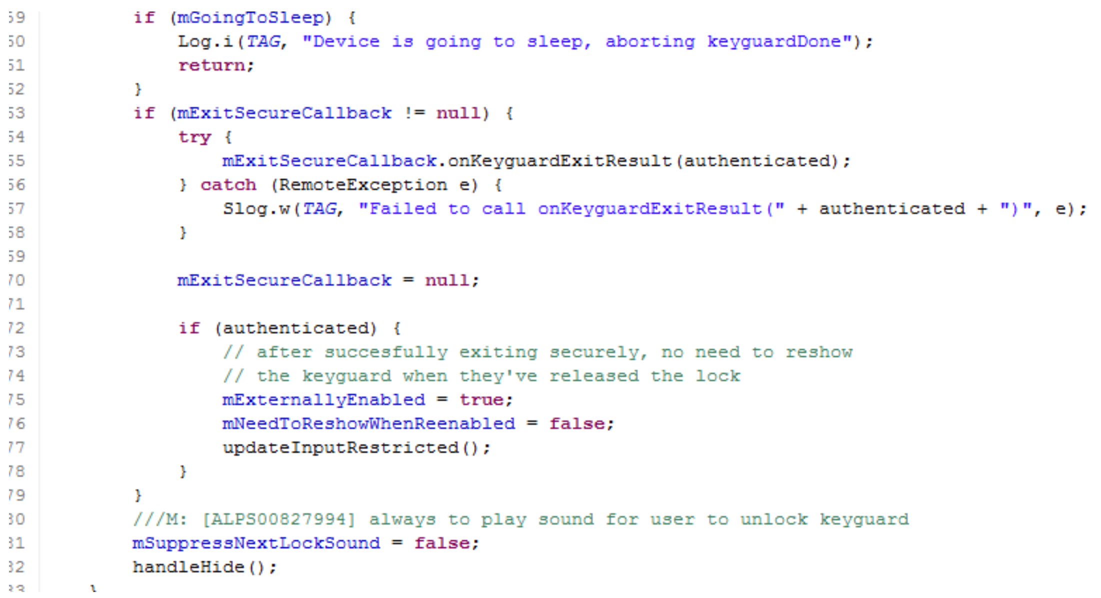


```
                    case NOTIFY_STARTED_WAKING_UP:
                        handleNotifyStartedWakingUp();
                        break;
                    case KEYGUARD_DONE:
                        handleKeyguardDone(msg.arg1 != 0);
                        break;
                    case KEYGUARD_DONE_DRAWING:
                        handleKeyguardDoneDrawing();
```



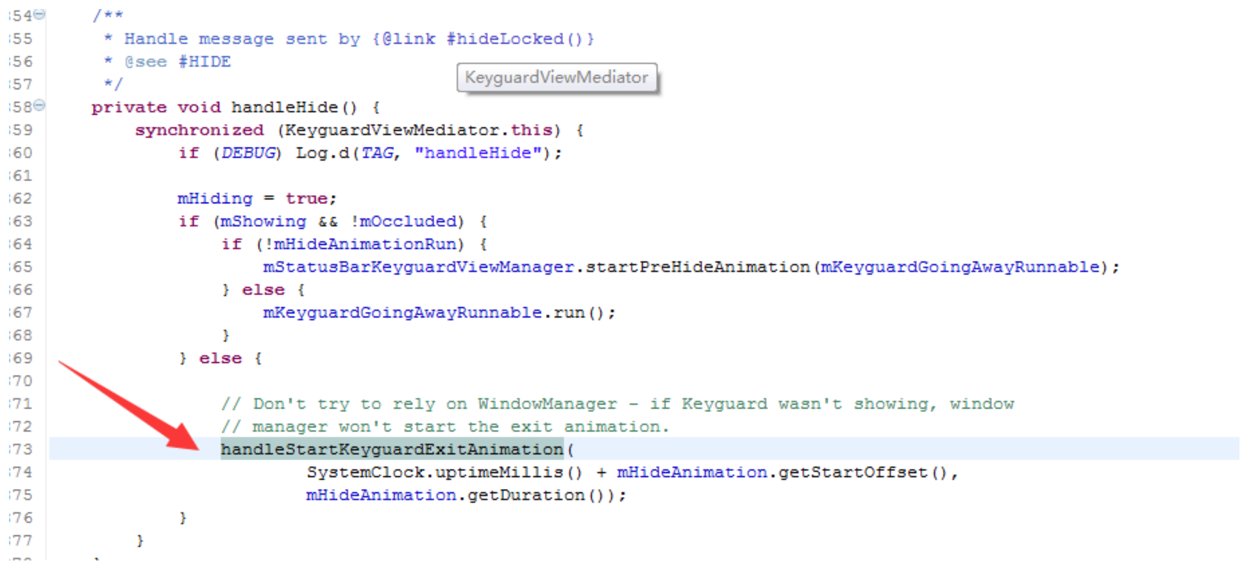
35M_SystemUI ▸ src ▸ com.android.systemui.keyguard ▸ KeyguardViewMediator ▸ handleKeyguardDone(boolean) : void

```
    /**
     * @see #keyguardDone
     * @see #KEYGUARD_DONE
     */
    private void handleKeyguardDone(boolean authenticated) {
        Log.d(TAG, "handleKeyguardDone, authenticated=" + authenticated);
        synchronized (this) {
            resetKeyguardDonePendingLocked();
        }

        ///M: [ALPS01567248] Timing issue.
        ///   Voice Unlock View dismiss -> AntiTheft View shows
        ///   -> previous Voice Unlock dismiss flow calls handleKeyguardDone
        ///   -> remove AntiTheft View
        ///   So we avoid handleKeyguardDone if AntiTheft is the current view,
        ///   and not yet unlock correctly.
        if (AntiTheftManager.isAntiTheftLocked()) {
            Log.d(TAG, "handleKeyguardDone() - Skip keyguard done! antitheft = " +
                       AntiTheftManager.isAntiTheftLocked() +
                       " or sim = " + mUpdateMonitor.isSimPinSecure());
            return ;
        }

        Log.d(TAG, "set mKeyguardDoneOnGoing = true") ;
        mKeyguardDoneOnGoing = true ;
```



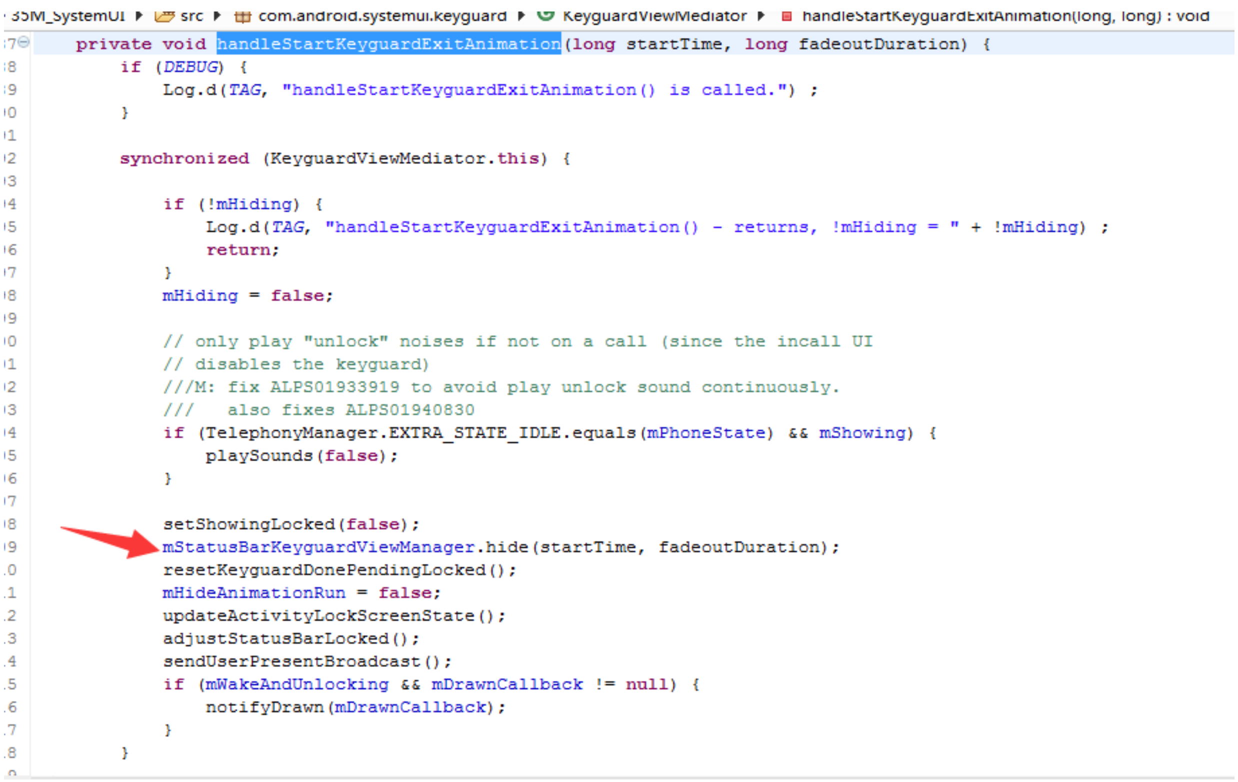
```
        if (mGoingToSleep) {
            Log.i(TAG, "Device is going to sleep, aborting keyguardDone");
            return;
        }
        if (mExitSecureCallback != null) {
            try {
                mExitSecureCallback.onKeyguardExitResult(authenticated);
            } catch (RemoteException e) {
                Slog.w(TAG, "Failed to call onKeyguardExitResult(" + authenticated + ")", e);
            }

            mExitSecureCallback = null;

            if (authenticated) {
                // after succesfully exiting securely, no need to reshow
                // the keyguard when they've released the lock
                mExternallyEnabled = true;
                mNeedToReshowWhenReenabled = false;
                updateInputRestricted();
            }
        }
        ///M: [ALPS00827994] always to play sound for user to unlock keyguard
        mSuppressNextLockSound = false;
        handleHide();
```

看到上面的方法，我们知道前面的一些判断不是关键的，主要的是handlerHide方法



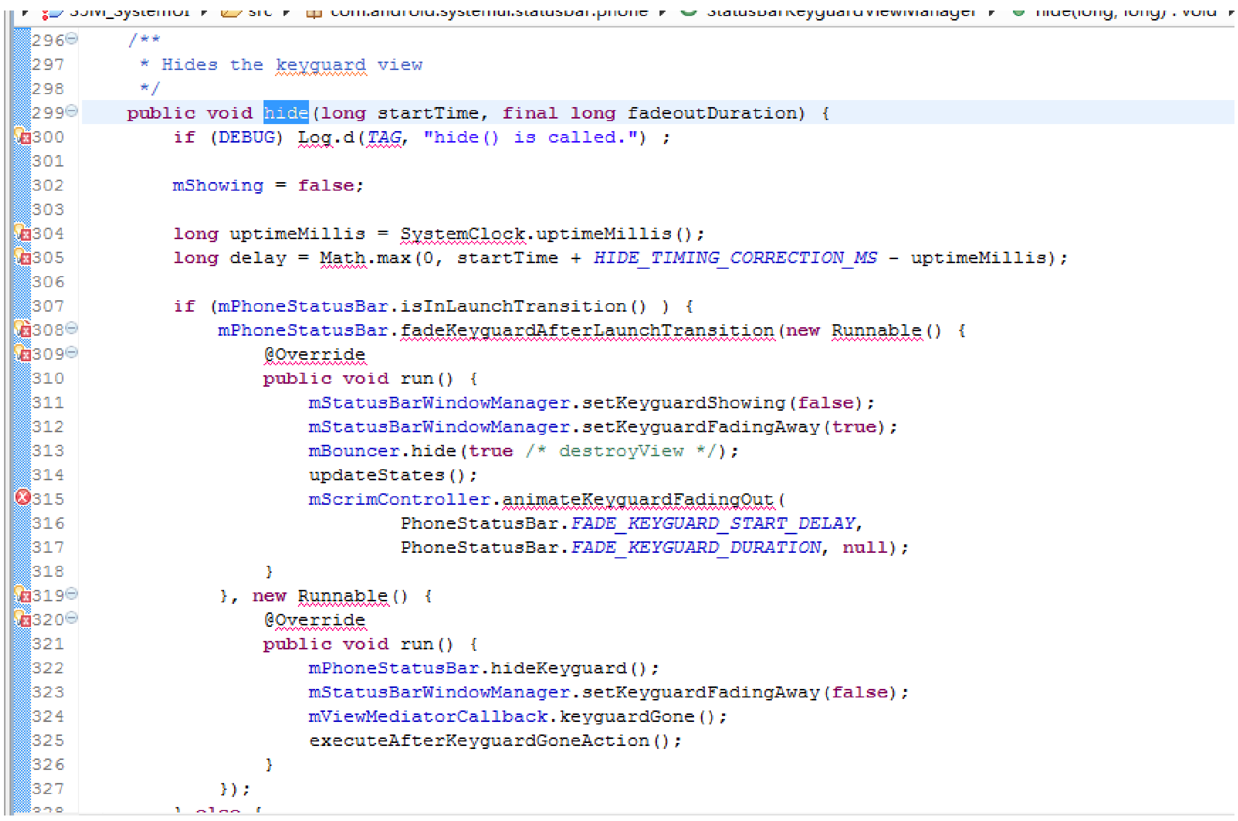
```
    /**
     * Handle message sent by {@link #hideLocked()}
     * @see #HIDE
     */
    private void handleHide() {
        synchronized (KeyguardViewMediator.this) {
            if (DEBUG) Log.d(TAG, "handleHide");

            mHiding = true;
            if (mShowing && !mOccluded) {
                if (!mHideAnimationRun) {
                    mStatusBarKeyguardViewManager.startPreHideAnimation(mKeyguardGoingAwayRunnable);
                } else {
                    mKeyguardGoingAwayRunnable.run();
                }
            } else {

                // Don't try to rely on WindowManager - if Keyguard wasn't showing, window
                // manager won't start the exit animation.
                handleStartKeyguardExitAnimation(
                        SystemClock.uptimeMillis() + mHideAnimation.getStartOffset(),
                        mHideAnimation.getDuration());
            }
        }
```

继续走HandlerStartKeyguardExitAnimation方法

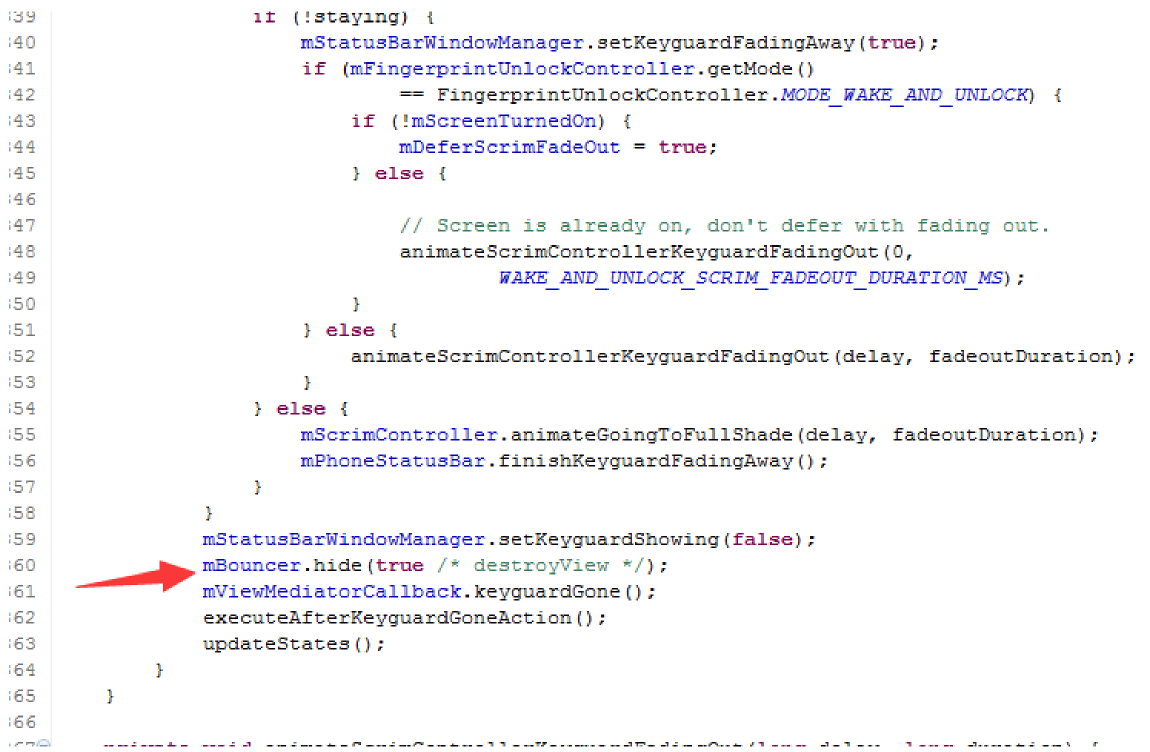


```
35M_SystemUI ▸ src ▸ com.android.systemui.keyguard ▸ KeyguardViewMediator ▸ handleStartKeyguardExitAnimation(long, long) : void
    private void handleStartKeyguardExitAnimation(long startTime, long fadeoutDuration) {
        if (DEBUG) {
            Log.d(TAG, "handleStartKeyguardExitAnimation() is called.") ;
        }

        synchronized (KeyguardViewMediator.this) {

            if (!mHiding) {
                Log.d(TAG, "handleStartKeyguardExitAnimation() - returns, !mHiding = " + !mHiding) ;
                return;
            }
            mHiding = false;

            // only play "unlock" noises if not on a call (since the incall UI
            // disables the keyguard)
            ///M: fix ALPS01933919 to avoid play unlock sound continuously.
            ///   also fixes ALPS01940830
            if (TelephonyManager.EXTRA_STATE_IDLE.equals(mPhoneState) && mShowing) {
                playSounds(false);
            }

            setShowingLocked(false);
            mStatusBarKeyguardViewManager.hide(startTime, fadeoutDuration);
            resetKeyguardDonePendingLocked();
            mHideAnimationRun = false;
            updateActivityLockScreenState();
            adjustStatusBarLocked();
            sendUserPresentBroadcast();
            if (mWakeAndUnlocking && mDrawnCallback != null) {
                notifyDrawn(mDrawnCallback);
            }
        }
```

StatusBarKeyguardViewManager.java
最后调用到StatusBarKeyguardViewManager.java中的方法hide



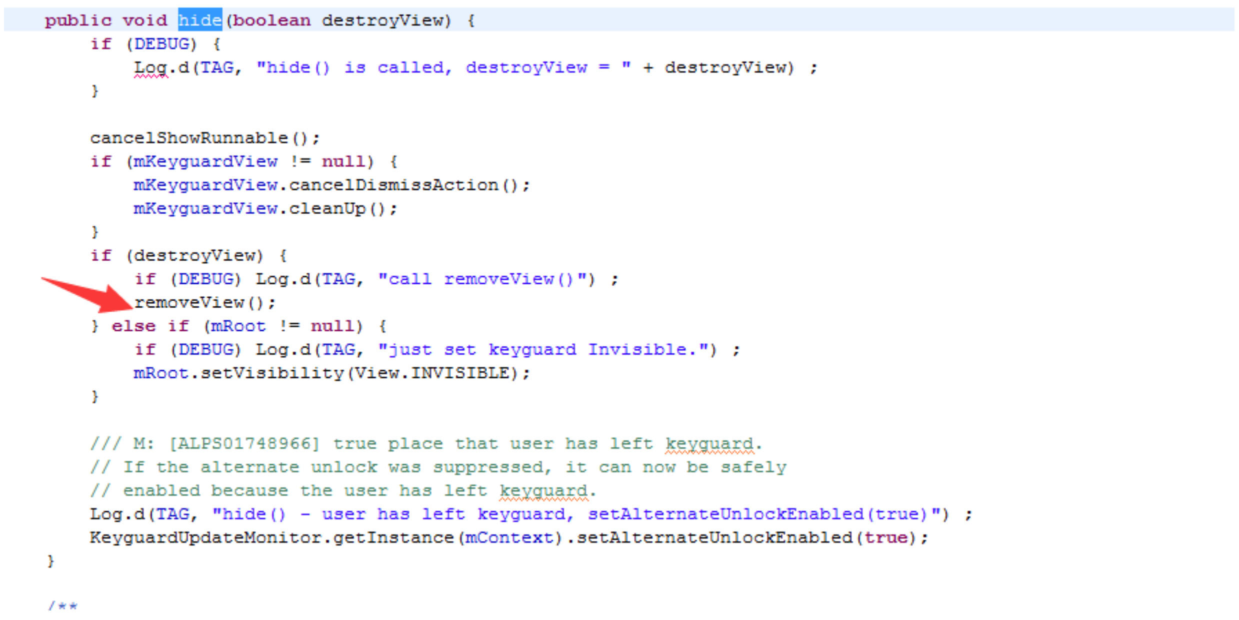
```
35M_SystemUI ▸ src ▸ com.android.systemui.statusbar.phone ▸ StatusBarKeyguardViewManager ▸ hide(long, long) : void
    /**
     * Hides the keyguard view
     */
    public void hide(long startTime, final long fadeoutDuration) {
        if (DEBUG) Log.d(TAG, "hide() is called.") ;

        mShowing = false;

        long uptimeMillis = SystemClock.uptimeMillis();
        long delay = Math.max(0, startTime + HIDE_TIMING_CORRECTION_MS - uptimeMillis);

        if (mPhoneStatusBar.isInLaunchTransition() ) {
            mPhoneStatusBar.fadeKeyguardAfterLaunchTransition(new Runnable() {
                @Override
                public void run() {
                    mStatusBarWindowManager.setKeyguardShowing(false);
                    mStatusBarWindowManager.setKeyguardFadingAway(true);
                    mBouncer.hide(true /* destroyView */);
                    updateStates();
                    mScrimController.animateKeyguardFadingOut(
                            PhoneStatusBar.FADE_KEYGUARD_START_DELAY,
                            PhoneStatusBar.FADE_KEYGUARD_DURATION, null);
                }
            }, new Runnable() {
                @Override
                public void run() {
                    mPhoneStatusBar.hideKeyguard();
                    mStatusBarWindowManager.setKeyguardFadingAway(false);
                    mViewMediatorCallback.keyguardGone();
                    executeAfterKeyguardGoneAction();
                }
            });
        } else {
```



```java
                if (!staying) {
                    mStatusBarWindowManager.setKeyguardFadingAway(true);
                    if (mFingerprintUnlockController.getMode()
                            == FingerprintUnlockController.MODE_WAKE_AND_UNLOCK) {
                        if (!mScreenTurnedOn) {
                            mDeferScrimFadeOut = true;
                        } else {

                            // Screen is already on, don't defer with fading out.
                            animateScrimControllerKeyguardFadingOut(0,
                                    WAKE_AND_UNLOCK_SCRIM_FADEOUT_DURATION_MS);
                        }
                    } else {
                        animateScrimControllerKeyguardFadingOut(delay, fadeoutDuration);
                    }
                } else {
                    mScrimController.animateGoingToFullShade(delay, fadeoutDuration);
                    mPhoneStatusBar.finishKeyguardFadingAway();
                }
            }
            mStatusBarWindowManager.setKeyguardShowing(false);
            mBouncer.hide(true /* destroyView */);
            mViewMediatorCallback.keyguardGone();
            executeAfterKeyguardGoneAction();
            updateStates();
        }
    }

    private void animateScrimControllerKeyguardFadingOut(long delay, long duration) {
```

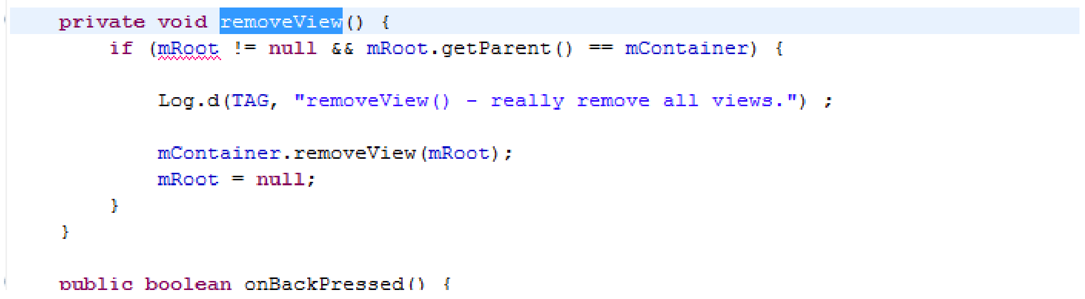
最后调用mBoucher,hide()

KeyguardBouncer.java



```java
    public void hide(boolean destroyView) {
        if (DEBUG) {
            Log.d(TAG, "hide() is called, destroyView = " + destroyView) ;
        }

        cancelShowRunnable();
        if (mKeyguardView != null) {
            mKeyguardView.cancelDismissAction();
            mKeyguardView.cleanUp();
        }
        if (destroyView) {
            if (DEBUG) Log.d(TAG, "call removeView()") ;
            removeView();
        } else if (mRoot != null) {
            if (DEBUG) Log.d(TAG, "just set keyguard Invisible.") ;
            mRoot.setVisibility(View.INVISIBLE);
        }

        /// M: [ALPS01748966] true place that user has left keyguard.
        // If the alternate unlock was suppressed, it can now be safely
        // enabled because the user has left keyguard.
        Log.d(TAG, "hide() - user has left keyguard, setAlternateUnlockEnabled(true)") ;
        KeyguardUpdateMonitor.getInstance(mContext).setAlternateUnlockEnabled(true);
    }

    /**
```

上诉方法中我们知道了最终调用到removeView方法来移除所有的View



```java
    private void removeView() {
        if (mRoot != null && mRoot.getParent() == mContainer) {

            Log.d(TAG, "removeView() - really remove all views.") ;

            mContainer.removeView(mRoot);
            mRoot = null;
        }
    }

    public boolean onBackPressed() {
```

以上我们就完成了整个滑动解锁的过程