# TODO

<integer name="def\_screen\_off\_timeout">60000</integer>

<bool name="def\_lockscreen\_disabled">false</bool>`

settings get secure lockscreen.disabled

# 概述

Keyguard锁屏的加载可能和我们想象的不一样，它是在灭屏的时候加载，这样可以确保的屏幕亮起来的时候，用户能第一时间看到锁屏界面。灭屏存在多种方式，例如按power键、屏幕超时等，这些灭屏的方式虽然不一样，但是殊途同归，我们主要分析按power键时，Keyguard的加载流程

锁屏界面的加载通常在android中有两种方式触发：android系统开机和screenOff(灭屏)后，再screenOn;

两个package，一个是frameworks/base/package/Keyguard主要为系统中锁屏模块的代码，例如我们看到的锁屏界面，以及画开锁屏界面会有密码，图案等不同的解锁界面等，主要是和锁屏的view相关的代码。  
 二是frameworks/base/package/SystemUI这个代表的是系统UI，状态栏，通知中心显示，最近任务列表，锁屏的都在这里面控制，只看锁屏模块的话，keyguard相当于处理显示view，而SystemUI是属于对keyguard进行管理的，控制其显示逻辑的。

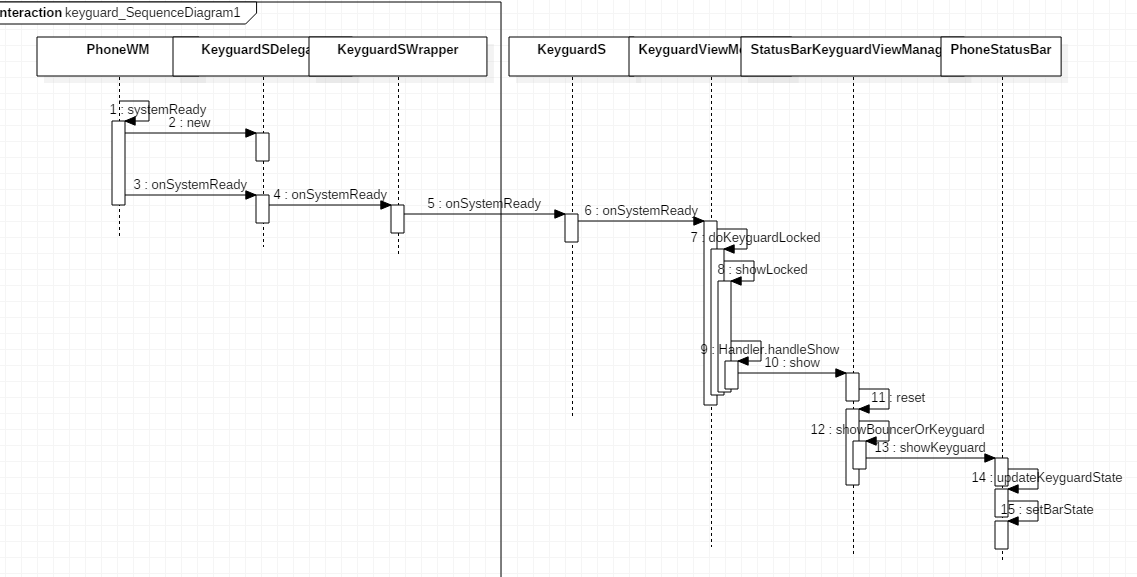
Keyguard是SystemUI 的模块依赖：

LOCAL\_STATIC\_JAVA\_LIBRARIES := \Keyguard，LOCAL\_RESOURCE\_DIR := \frameworks/base/packages/Keyguard/res \  
SystemUI编译需要是要依赖Keyguard模块的。  
修改Keyguard模块的代码后编译此模块不会产生相应的apk而是要在继续编译SystemUI模块的代码产生的SystemUI.apk放到手机里才会起作用，也就是说锁屏不会产生一个单独的apk文件，而是由SystemUI.apk包含了现有手机的锁屏模块。

锁屏界面称为Keyguard（常见为锁屏界面时钟，通知界面）

* 解锁界面称为Bouncer （常见为滑开锁屏界面，显示密码，图案等解锁方式界面）

# Keyguard与SystemUI



## doKeyguardLocked

1、mExternallyEnabled;默认为true，如果其它应用设置keyguard不显示，则直接return不显示;

2、如果keyguard当前正在显示，则不用管它，重置;

3、如果安装向导未执行完毕，即设备未完成初始化绑定等操作，也不去显示keyguard;

4、如果当前屏幕为灭屏状态，也不去显示keyguard;

5、Without this, settings is not enabled until the lock screen first appears(我这边没有比较好的说法);

6、如果上述条件都不满足则使用showLocked()方法开始显示keyguard。

**private void** doKeyguardLocked(Bundle options) {  
 *// if another app is disabling us, don't show* **if** (!mExternallyEnabled) {  
 **if** (DEBUG) Log.d(TAG, **"doKeyguard: not showing because externally disabled"**);  
  
 *// note: we \*should\* set mNeedToReshowWhenReenabled=true here, but that makes  
 // for an occasional ugly flicker in this situation:  
 // 1) receive a call with the screen on (no keyguard) or make a call  
 // 2) screen times out  
 // 3) user hits key to turn screen back on  
 // instead, we reenable the keyguard when we know the screen is off and the call  
 // ends (see the broadcast receiver below)  
 // TODO: clean this up when we have better support at the window manager level  
 // for apps that wish to be on top of the keyguard* **return**;  
 }  
  
 *// if the keyguard is already showing, don't bother* **if** (mStatusBarKeyguardViewManager.isShowing()) {  
 **if** (DEBUG) Log.d(TAG, **"doKeyguard: not showing because it is already showing"**);  
 resetStateLocked();  
 **return**;  
 }  
  
 *// In split system user mode, we never unlock system user.* **if** (!mustNotUnlockCurrentUser()  
 || !mUpdateMonitor.isDeviceProvisioned()) {  
  
 *// if the setup wizard hasn't run yet, don't show* **final boolean** requireSim = !SystemProperties.getBoolean(**"keyguard.no\_require\_sim"**, **false**);  
 **final boolean** absent = SubscriptionManager.isValidSubscriptionId(  
 mUpdateMonitor.getNextSubIdForState(IccCardConstants.State.ABSENT));  
 **final boolean** disabled = SubscriptionManager.isValidSubscriptionId(  
 mUpdateMonitor.getNextSubIdForState(IccCardConstants.State.PERM\_DISABLED));  
 **final boolean** lockedOrMissing = mUpdateMonitor.isSimPinSecure()  
 || ((absent || disabled) && requireSim);  
  
 **if** (!lockedOrMissing && shouldWaitForProvisioning()) {  
 **if** (DEBUG) Log.d(TAG, **"doKeyguard: not showing because device isn't provisioned"** + **" and the sim is not locked or missing"**);  
 **return**;  
 }  
  
 **if** (mLockPatternUtils.isLockScreenDisabled(KeyguardUpdateMonitor.getCurrentUser())  
 && !lockedOrMissing) {  
 **if** (DEBUG) Log.d(TAG, **"doKeyguard: not showing because lockscreen is off"**);  
 **return**;  
 }  
  
 **if** (mLockPatternUtils.checkVoldPassword(KeyguardUpdateMonitor.getCurrentUser())) {  
 **if** (DEBUG) Log.d(TAG, **"Not showing lock screen since just decrypted"**);  
 *// Without this, settings is not enabled until the lock screen first appears* setShowingLocked(**false**);  
 hideLocked();  
 mUpdateMonitor.reportSuccessfulStrongAuthUnlockAttempt();  
 **return**;  
 }  
 }  
  
 **if** (DEBUG) Log.d(TAG, **"doKeyguard: showing the lock screen"**);  
 showLocked(options);  
}

## reset()方法去重置mStatusBarView的state，先来看reset()方法

sdf’

public void reset() {

if (mShowing) {

if (mOccluded) {

mPhoneStatusBar.hideKeyguard();

mPhoneStatusBar.stopWaitingForKeyguardExit();

mBouncer.hide(false /\* destroyView \*/);

} else {

showBouncerOrKeyguard();

}

KeyguardUpdateMonitor.getInstance(mContext).sendKeyguardReset();

updateStates();

}

在reset()方法中会去判断keyguard是否被其它的窗口中断mOccluded，是则不显示keyguard;否则的就执行showBouncerOrKeyguard()方法;

showBouncerOrKeyguard()方法使用KeyguardBouncer.java的needsFullscreenBouncer()方法判断显示常规锁屏还是Bouncer安全锁屏（比如图案锁屏、密码锁屏、PIN码锁屏等）;

---------------------

## 加载锁屏界面showBouncerOrKeyguard

**protected void** showBouncerOrKeyguard() {  
 **if** (mBouncer.needsFullscreenBouncer()) {  
 *// The keyguard might be showing (already). So we need to hide it.* mPhoneStatusBar.hideKeyguard();  
 mBouncer.show(**true** */\* resetSecuritySelection \*/*);  
 } **else** {  
 mPhoneStatusBar.showKeyguard();  
 mBouncer.hide(**false** */\* destroyView \*/*);  
 mBouncer.prepare();  
 }  
}

此方法中就是对于显示Bouncer还是Keyguard的区分，先执行了KeyguardBouncer.java中的needsFullscreenBouncer()方法

### needsFullscreenBouncer

**public boolean** needsFullscreenBouncer() {  
 ensureView();  
 **if** (**mKeyguardView** != **null**) {  
 SecurityMode mode = **mKeyguardView**.getSecurityMode();  
 **return** mode == SecurityMode.***SimPin*** || mode == SecurityMode.***SimPuk***;  
 }  
 **return false**;  
}

可以看出当解锁方式为SimPin和SimPuk时才会返回为true，对应着sim卡的两种状态，这个时候要显示一个全屏的界面，必须要输入正确相应的密码才能进入，否则不能使用手机。

## 锁屏方式

Settings应用包中的ChooseLockGeneric.java.setUnlockMethod

**private boolean** setUnlockMethod(String unlockMethod) {  
 EventLog.*writeEvent*(EventLogTags.***LOCK\_SCREEN\_TYPE***, unlockMethod);  
  
 **if** (***KEY\_UNLOCK\_SET\_OFF***.equals(unlockMethod)) {  
 updateUnlockMethodAndFinish(  
 DevicePolicyManager.***PASSWORD\_QUALITY\_UNSPECIFIED***, **true** */\* disabled \*/* );  
 } **else if** (***KEY\_UNLOCK\_SET\_NONE***.equals(unlockMethod)) {  
 updateUnlockMethodAndFinish(  
 DevicePolicyManager.***PASSWORD\_QUALITY\_UNSPECIFIED***, **false** */\* disabled \*/* );  
 } **else if** (***KEY\_UNLOCK\_SET\_MANAGED***.equals(unlockMethod)) {  
 maybeEnableEncryption(DevicePolicyManager.PASSWORD\_QUALITY\_MANAGED, **false**);  
 } **else if** (***KEY\_UNLOCK\_SET\_PATTERN***.equals(unlockMethod)) {  
 maybeEnableEncryption(  
 DevicePolicyManager.***PASSWORD\_QUALITY\_SOMETHING***, **false**);  
 } **else if** (***KEY\_UNLOCK\_SET\_PIN***.equals(unlockMethod)) {  
 maybeEnableEncryption(  
 DevicePolicyManager.***PASSWORD\_QUALITY\_NUMERIC***, **false**);  
 } **else if** (***KEY\_UNLOCK\_SET\_PASSWORD***.equals(unlockMethod)) {  
 maybeEnableEncryption(  
 DevicePolicyManager.***PASSWORD\_QUALITY\_ALPHABETIC***, **false**);  
 } **else** {  
 Log.*e*(***TAG***, **"Encountered unknown unlock method to set: "** + unlockMethod);  
 **return false**;  
 }  
 **return true**;  
}

设置，更新解锁方式和加密等操作都是通过此类来完成的。我们看到它设置锁屏方式就是设置了不同的DevicePolicyManager，这样的话我们就需要看在锁屏中的代码，根据搜关键字DevicePolicyManager我们就可以发现使如何设置当前的锁屏方式的。  
 keyguard包中的代码KeyguardSecurityModel.java：

KeyguardSecurityModel

**public enum** SecurityMode {  
 ***Invalid***, *// NULL state* ***None***, *// No security enabled* ***Pattern***, *// Unlock by drawing a pattern.* ***Password***, *// Unlock by entering an alphanumeric password* ***PIN***, *// Strictly numeric password* ***SimPin***, *// Unlock by entering a sim pin.* ***SimPuk*** *// Unlock by entering a sim puk*}

### getSecurityMode

SecurityMode getSecurityMode() {  
 KeyguardUpdateMonitor monitor = KeyguardUpdateMonitor.*getInstance*(**mContext**);  
  
 **if** (SubscriptionManager.isValidSubscriptionId(  
 monitor.getNextSubIdForState(IccCardConstants.State.***PIN\_REQUIRED***))) {  
 **return** SecurityMode.***SimPin***;  
 }  
  
 **if** (**mIsPukScreenAvailable** && SubscriptionManager.isValidSubscriptionId(  
 monitor.getNextSubIdForState(IccCardConstants.State.***PUK\_REQUIRED***))) {  
 **return** SecurityMode.***SimPuk***;  
 }  
  
 **final int** security = **mLockPatternUtils**.getActivePasswordQuality(  
 KeyguardUpdateMonitor.*getCurrentUser*());  
 **switch** (security) {  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_NUMERIC***:  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_NUMERIC\_COMPLEX***:  
 **return** SecurityMode.***PIN***;  
  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_ALPHABETIC***:  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_ALPHANUMERIC***:  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_COMPLEX***:  
 **case** DevicePolicyManager.PASSWORD\_QUALITY\_MANAGED:  
 **return** SecurityMode.***Password***;  
  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_SOMETHING***:  
 **return** SecurityMode.***Pattern***;  
 **case** DevicePolicyManager.***PASSWORD\_QUALITY\_UNSPECIFIED***:  
 **return** SecurityMode.***None***;  
  
 **default**:  
 **throw new** IllegalStateException(**"Unknown security quality:"** + security);  
 }

从而可以看出设置和锁屏中通过LockPatternUtils设置和保存不同的密码等来区分当前的解锁方式，然后再看到它的调用出的代码KeyguardSecurityContainer类，此类是屏幕 解锁界面的父view，同过此view跟新当前解锁方式view:

com/android/keyguard/KeyguardSecurityContainer.java

### getSecurityView

**private** KeyguardSecurityView getSecurityView(SecurityMode securityMode) {

**int** layoutId = getLayoutIdFor(securityMode);

}

**protected int** getLayoutIdFor(SecurityMode securityMode) {  
 **switch** (securityMode) {  
 **case *Pattern***: **return** R.layout.*keyguard\_pattern\_view*;  
 **case *PIN***: **return** R.layout.*keyguard\_pin\_view*;  
 **case *Password***: **return** R.layout.*keyguard\_password\_view*;  
 **case *SimPin***: **return** R.layout.*keyguard\_sim\_pin\_view*;  
 **case *SimPuk***: **return** R.layout.*keyguard\_sim\_puk\_view*;  
 **default**:  
 **return** 0;  
 }  
}

这样就可以看到不同的解锁方式对应者不同的自定义布局例如：  
密码布局为：KeyguardPasswordView.java  
Pin码为：KeyguardPINView.java  
图案为：KeyguardPatternView

Fas

常规锁屏即为滑动锁屏界面，一般滑动即可解锁，称之为notification keyguard;这个类型的keyguard已经和statusbar融为一体了，可以通过PhoneStatusBar.java的对象直接进行控制;

2、Bouncer安全锁屏;比如密码、图案、PIM码、PUK码等锁屏方式的锁屏界面，通过KeyguardBouncer.java来开始控制show()和hide();

## KeyguardBouncer. show

首先调用ensureView()方法去加载keyguard\_bouncer view

KeyguardHostView.java的showPrimarySecurityScreen()方法;

public void showPrimarySecurityScreen() {

if (DEBUG) Log.d(TAG, "show()");

mSecurityContainer.showPrimarySecurityScreen(false);

继续调用KeyguardSecurityContainer.java的showPrimarySecurityScreen()方法，先去获取锁屏方式;

继续往下将获取到的锁屏方式securityMode作为参数调用showSecurityScreen()方法;这个方法主要是用来根据securityMode显示锁屏view的

## xiaojie

、首先判断传入进来的securityMode是否已经被显示;

2、调用getSecurityView()方法获取给定的锁屏view;

3、调用KeyguardSecurityView.java的onPause()方法暂停显示旧锁屏view，onResume()方法开始显示新的锁屏view;KeyguardSecurityView.java是一个接口类，其内部方法都是抽象的只有声明没有实现，其方法实现都是在继承于这个接口的类中。

而在keyguard中主要是KeyguardAbsKeyInputView.java、KeyguardPasswordView.java、KeyguardPatternView.java等等这些类继承于此接口实现其内部方法，这些类就是具体的锁屏界面view显示;

## 开机显示keyguard的总结：

1、在KeyguardViewMediator.java的onSystemReady()方法内调用doKeyguardLocked()开始锁屏加载流程;

2、setKeyguardEnabled();其他应用程序或者服务可以调用setKeyguardEnabled()方法请求禁止锁屏;

3、KeyguardViewMediator.java在keyguard中起着主要调度的作用，主要负责

1）查询锁屏状态，当前是锁屏还是解锁状态;在锁屏状态下，会限制输入事件。

2）PhoneWindowManager.java通过mKeyguardDelegate对象(KeyguardServiceDelegate.java)来使能KeyguardViewMediator.java，调用其中的方法;

3）响应SIM卡状态变化并对锁屏界面做相应的调整onSimStateChanged();

4、判断keyguard是否被禁止、keyguard当前是否正在显示等等即当前是否可以显示keguard，可以显示的话继续调用showLocked()方法;

5、调用handleShow()方法，调用StatusBarKeyguardViewManager.java的show()开始显示keyguard锁屏界面;

6、调用reset()方法，调用showBouncerOrKeyguard()方法判断是显示正常锁屏界面还是安全锁屏界面;显示正常锁屏的话直接调用PhoneStatusBar.java的showKeyguard()或者hideKeyguard()方法;如果显示安全锁屏界面的话则调入KeyguardBouncer.java类内;

7、调用KeyguardBouncer.java的show()方法;使用ensureView()方法去加载实例化布局;调用KeyguardHostView.java的showPrimarySecurityScreen()方法去显示安全锁屏界面;

8、KeyguardHostView.java的showPrimarySecurityScreen()方法会调入到KeyguardSecurityContainer.java的showPrimarySecurityScreen()方法中来;

9、调用showSecurityScreen()方法，根据锁屏方式来加载不同的锁屏view;

10、KeyguardSecurityView.java是一个接口类，其内部方法都是抽象的只有声明没有实现，其方法实现都是在继承于这个接口的类中。

而在keyguard中主要是KeyguardAbsKeyInputView.java、KeyguardPasswordView.java、KeyguardPatternView.java等等Keyguard\*View.java这些类继承于此接口实现其内部方法，这些类就是具体的锁屏界面view显示;

系统灭屏Screen off之后的keguard加载流程：

android系统中的自动灭屏跟Power按键之后灭屏流程可能有点区别，但是由于主要是分析灭屏之后keyguard加载，所以只需要关心keguard在系统灭屏之后的加载流程。

这里以按power键灭屏为例，分析其流程：

KeyguardDelegate.onFinishedGoingToSleep(why);

1. mStatusBarKeyguardViewManager.onFinishedGoingToSleep();

继续调用KeyguardSecurityContainer.java的showPrimarySecurityScreen()方法，根据上面的分析知道，此处先会去获取当前锁屏方式，然后根据得到的锁屏方式去加载锁屏界面;

至此完成keyguard在screen off状态下的加载流程分析;

正常灭屏显示keyguard流程总结：

1、不管是按Power键还是自动灭屏，都会执行到PowerManagerService.java的gotoSleep()方法;

2、在这个方法内通过一系列的调用，调入到PhoneWindowManager.java的finishedGoingToSleep()方法;

3、在PhoneWindowManager.java类中通过KeyguardServiceDelegate.java类的对象mKeyguardDelegate来使能KeyguardViewMediator.java;

4、而KeyguardViewMediator.java作为keyguard的调度者，从这里开始keyguard的加载;

5、最终在KeyguardSecurityContainer.java的showPrimarySecurityScreen()方法内去实现根据锁屏方式加载锁屏界面;

## Keyguard与SystemUI的交互

既然Keyguard是作为SystemUI的library存在的，那么在SystemUI中调用Keyguard中的类，就很方便了，只需要直接导入类即可，但是在Keyguard中如何和SystemUI通信呢？主要关注三个类：

* KeyguardViewMediator.java此类是在SystemUI中做统一调度的，也就是像我们长熟悉的熄屏，亮屏，锁屏等的处理都是在这里面的做的，它是一个对Keyguard的调度者。
* KeyguardUpdateMonitor.java，从此类的说明中就可以看的出来，它是来处理锁屏更新操作的类，KeyguardViewMediator.java等对于锁屏更新的额相关处理都是在这里面进行的操作。
* KeyguardUpdateMonitorCallback.java，作为更新之后的回调类，当我们跟新一些状态之后，还需要更新之后反馈一些状态时就会需要用到此类，就像我们进行网络请求，不能就发送网络请求后就不管了，需要反馈结果是连接成功了，还是失败了，需要有明确的信息帮助我们进行下一步操作

KeyguardUpdateMonitor的函数：回调的经典函数KeyguardUpdateMonitorCallback cb = **mCallbacks**.get(i).get();

**private final** ArrayList<WeakReference<KeyguardUpdateMonitorCallback>>  
 **mCallbacks** = Lists.*newArrayList*();

### registerCallback

**public void** registerCallback(KeyguardUpdateMonitorCallback callback) {  
 **if** (***DEBUG***) Log.*v*(***TAG***, **"\*\*\* register callback for "** + callback);  
 *// Prevent adding duplicate callbacks* **for** (**int** i = 0; i < **mCallbacks**.size(); i++) {  
 **if** (**mCallbacks**.get(i).get() == callback) {  
 **if** (***DEBUG***) Log.*e*(***TAG***, **"Object tried to add another callback"**,  
 **new** Exception(**"Called by"**));  
 **return**;  
 }  
 }  
 **mCallbacks**.add(**new** WeakReference<KeyguardUpdateMonitorCallback>(callback));  
 removeCallback(**null**); *// remove unused references* sendUpdates(callback);  
}

### removeCallback

**public void** removeCallback(KeyguardUpdateMonitorCallback callback) {  
 **if** (***DEBUG***) Log.*v*(***TAG***, **"\*\*\* unregister callback for "** + callback);  
 **for** (**int** i = **mCallbacks**.size() - 1; i >= 0; i--) {  
 **if** (**mCallbacks**.get(i).get() == callback) {  
 **mCallbacks**.remove(i);  
 }  
 }  
}

## 禁用锁屏

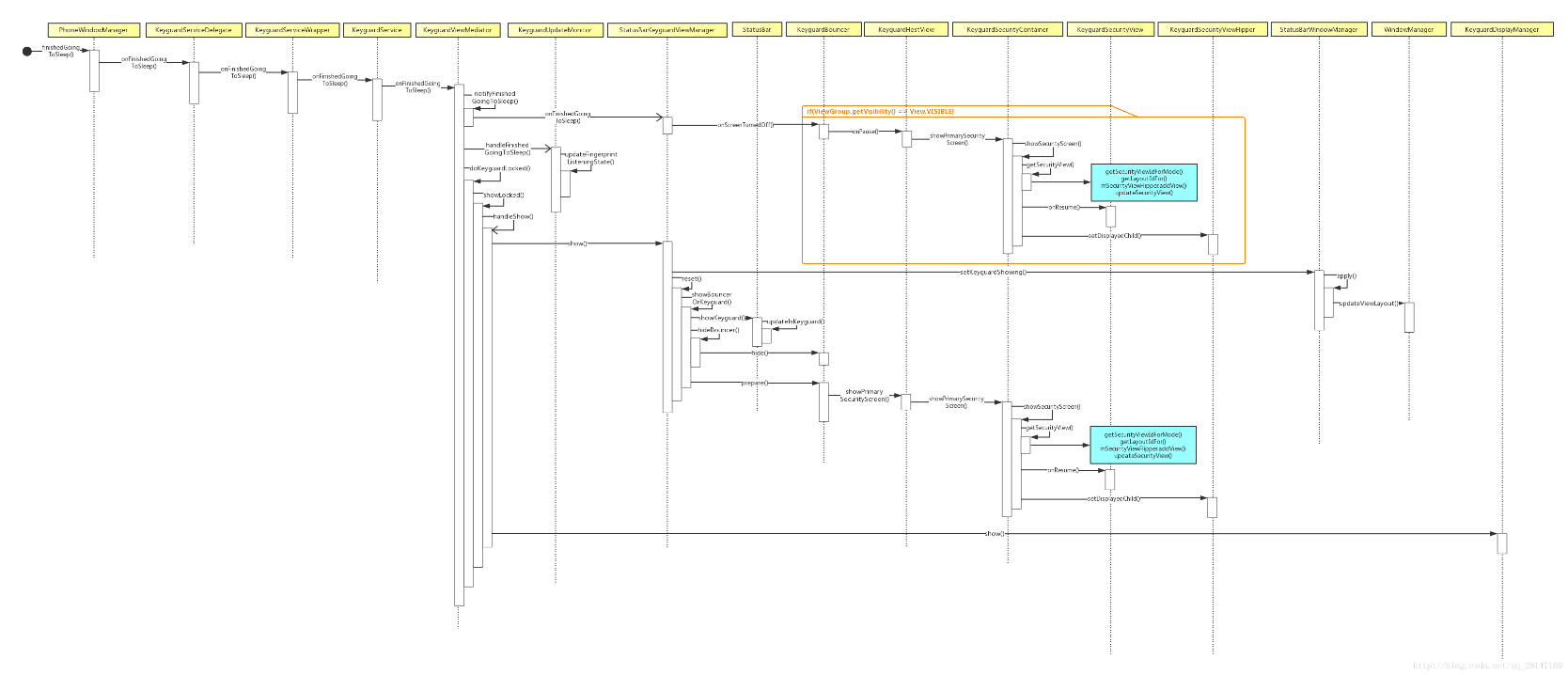
采用aidl通信技术

@Override  
**public void** enableKeyguard(**boolean** enabled) {  
 mKeyguardDelegate.setKeyguardEnabled(enabled);  
}

@Override *// Binder interface***public void** setKeyguardEnabled(**boolean** enabled) {  
 **mKeyguardViewMediator**.setKeyguardEnabled(enabled);  
}

**mKeyguardViewMediator**.setKeyguardEnabled(enabled);

# 灭屏时序图



所以在绘制完成之后，KeyguardViewMediator的handleShow()会调用KeyguardDisplayManager的show()方法，此时Keyguard就已经绘制完毕了，当然因为已经灭屏了，所以你并不能看到界面，当下次亮屏的时候，就会看到绘制好了的keyguard

这种设计想法很好的

### 时序图

# 锁屏与系统服务

<https://blog.csdn.net/azhengye/article/details/50419409>

在bindService中调用了bindServiceAsUser绑定指定intent的service，config\_keyguardComponent的定义如下

## 调起

### PhoneWM. systemReady

**public void** systemReady() {  
 mKeyguardDelegate = **new** KeyguardServiceDelegate(mContext,  
 **this**::onKeyguardShowingStateChanged);  
 mKeyguardDelegate.onSystemReady();

PhoneWM.init

mKeyguardDelegate = **new** KeyguardServiceDelegate(mContext,  
 **new** StateCallback() {  
 @Override  
 **public void** onTrustedChanged() {  
 mWindowManagerFuncs.notifyKeyguardTrustedChanged();  
 }  
 });

## 绑定

无论是在systemReady或systemBooted，都调用了KeyguardServiceDelegate对象的bindService方法

@Override  
**public void** systemBooted() {  
 bindKeyguard();  
 **synchronized** (mLock) {  
 mSystemBooted = **true**;  
 **if** (mSystemReady) {  
 mKeyguardDelegate.onBootCompleted();  
 }  
 }  
 startedWakingUp();  
 screenTurningOn(**null**);  
 screenTurnedOn();  
}

@Override  
**public void** onSystemUiStarted() {  
 bindKeyguard();  
}

systemBooted里绑定了就不在systemReady里再去绑定，自己测试的时候是在systemBooted绑定的

**private void** bindKeyguard() {  
 **synchronized** (mLock) {  
 **if** (mKeyguardBound) {  
 **return**;  
 }  
 mKeyguardBound = **true**;  
 }  
 mKeyguardDelegate.bindService(mContext);  
}

### KeyguardServiceDelegate如何绑定KeyguardService的

/frameworks/base/services/core/java/com/android/server/policy/keyguard/KeyguardServiceDelegate.java

## KeyguardServiceDelegate bindService

**public void** bindService(Context context) {  
 **final** ComponentName keyguardComponent = ComponentName.unflattenFromString(  
 resources.getString(com.android.internal.R.string.config\_keyguardComponent));  
 intent.setComponent(keyguardComponent);  
  
 **if** (!context.bindServiceAsUser(intent, mKeyguardConnection,  
 Context.BIND\_AUTO\_CREATE, mHandler, UserHandle.SYSTEM)) {  
 Log.v(TAG, **"\*\*\* Keyguard: can't bind to "** + keyguardComponent);  
 mKeyguardState.showing = **false**;  
 mKeyguardState.showingAndNotOccluded = **false**;  
 mKeyguardState.secure = **false**;  
 **synchronized** (mKeyguardState) {  
 *// TODO: Fix synchronisation model in this class. The other state in this class  
 // is at least self-healing but a race condition here can lead to the scrim being  
 // stuck on keyguard-less devices.* mKeyguardState.deviceHasKeyguard = **false**;  
 }  
 } **else** {  
 **if** (DEBUG) Log.v(TAG, **"\*\*\* Keyguard started"**);  
 }  
}

在bindService中调用了bindServiceAsUser绑定指定intent的service，config\_keyguardComponent的定义如下

*<!-- Keyguard component -->*<**string name="config\_keyguardComponent" translatable="false"**>com.android.systemui/com.android.systemui.keyguard.KeyguardService</**string**>

当绑定成功后会调用mKeyguardConnection里的onServiceConnected方法

## KeyguardConnection里的onServiceConnected

# KeyguardScrim

WindowManager.LayoutParams.TYPE\_KEYGUARD\_SCRIM;

# 案例

com.android.settings W/ProcStatsManager: No process com.android.systemui/10016 for service com.android.systemui.keyguard.KeyguardService

8701-8733/system\_process W/ActivityManager: Scheduling restart of crashed service com.android.systemui/.keyguard.KeyguardService in 20999ms

8701-8733/system\_process I/ActivityManager: Force stopping service ServiceRecord{9237fbb u0 com.android.systemui/.keyguard.KeyguardService}

8701-8732/system\_process V/KeyguardServiceDelegate: \*\*\* Keyguard disconnected (boo!)

8701-8749/system\_process W/KeyguardServiceDelegate: onScreenTurningOn(): no keyguard service!

## QA

### QA，VVV为何启动大APP必崩，其他的都是正常的

跟kjimode有关，因为systemui（当不在系统私有目录的时候）被kill了，会杀一次。

Data/app,显然是，

### 为何手动 kill –systemui没有复现

### Phone.showScrim干嘛

**public void** removeWindowLw(WindowState win) {  
 **if** (mStatusBar == win) {  
 mStatusBar = **null**;  
 mStatusBarController.setWindow(**null**);  
 mKeyguardDelegate.showScrim();  
 } **else**

### crim(视图内容是啥？？

## 参考

基于android7.1.1锁屏模块分析

https://www.jianshu.com/p/f8ead0a44f5b