Android Doze模式分析

2016-09-13 10:10:19 0个评论 来源：zhenjie.Chang的专栏 收藏 我要投稿

Doze模式是Android6.0上新出的一种模式，是一种全新的、低能耗的状态，在后台只有部分任务允许运行，其他都被强制停止。当用户一段时间没有使用手机的时候，Doze模式通过延缓app后台的CPU和网络活动减少电量的消耗。PowerManagerService中也有Doze模式，和此处的Doze模式不一样，其实此处叫DeviceIdle模式更容易区分

如果一个用户断开了充电连接，关屏不动手机一段时间之后，设备进入Doze模式。在Doze模式中，系统尝试去通过减少应用的网络访问和CPU敏感的服务来保护电池。它也阻止应用通过访问网络，并且延缓应用的任务、同步和标准alarms。

系统定期退出Doze模式（maintenancewindow）去让app完成他们被延缓的动作。在maintenancewindow期间，系统运行所有挂起的同步、任务和alarms，同时也能访问网络

Doze模式的限制。

1.网络接入被暂停

2.系统忽略wakelocks

3.标准的AlarmManageralarms被延缓到下一个maintenancewindow

4.如果你需要在Doze状态下启动设置的alarms，使用setAndAllowWhileIdle()或者setExactAndAllowWhileIdle()。

5.当有setAlarmClock()的alarms启动时，系统会短暂退出Doze模式

6.系统不会扫描Wi-Fi

7.系统不允许syncadapters运行

8.系统不允许JobScheduler运行

Doze模式在系统中主要有DeviceIdleController来驱动。下面我们来分析下DeviceIdleController

文章 推荐

· Android二维码扫描
· Android热补丁动态修复技术（四）：自
· Android中基于RxJava的响应式编程
· android 中vector的用法的一些看法
· Android 6.0 IMS流程(一)——IMS开机
· HTTPS理论基础及其在Android中的最佳实
· 安卓Zygote详解
· 关于Fragment的方方面面

点击排行

· [Android Studio 权威教程] 断点调
· RecyclerView完全解析,让你从此爱上它
· Android M新控件之AppBarLayout，Nav
· Android Studio中如何引用图片资源
· Android之开发常用颜色
· Android Studio安装配置详细步骤（图
· 框架模式 MVC 在Android中的使用
· Android的AlertDialog详解

```
mSystemServiceManager.startService(DeviceIdleController.class);
```


同样，在SystemServiceManager中的startService方法中利用反射的方法构造DeviceIdleController对象，然后调用DeviceIdleController的onStart方法来初始化。

DeviceIdleController的构造方法

```
public DeviceIdleController(Context context) {
        super(context);
        mConfigFile = new AtomicFile(new File(getSystemDir(), "device
        mHandler = new MyHandler(BackgroundThread.getHandler().getLoo
    }
```

构造方法很简单，只有两步

1.创建一个deviceidle.xml文件，该文件位于data/system/目录下。

2.创建了一个Handler用来处理消息

onStart方法

```
public void onStart() {
        ……
        synchronized (this) {
           //第1步，获取Doze模式是否默认开启
            mEnabled = getContext().getResources().getBoolean(
                    com.android.internal.R.bool.config_enableAutoPow
           //第2步，从systemConfig中读取默认的系统应用的白名单
            SystemConfig sysConfig = SystemConfig.getInstance();
            ArraySet<string> allowPowerExceptIdle = sysConfig.getAll
            for (int i=0; i<allowpowerexceptidle.size(); string="" p
            for (int i=0; i<allowpower.size(); string="" pkg="allowP
```

这个方法中大致可以分为6部分

第1步：从配置文件中获取Doze模式的开关值，默认为false

第2步：从SystemConfig中读取Doze模式系统应用的白名单，这个白名单是已经在系统配置文件中配置好的，位于手机目录system/ect/sysconfig中。

收集了配置的除了Idle模式都可以运行的白名单

第3步：从SystemConfig中读取Doze模式的白名单

第2步和第3步主要用于读取Doze模式下系统应用的白名单。

第4步：读取deviceidle.xml文件，解析xml文件并将用户应用的白名单读入内存

第5步：设置Doze模式的白名单，通过updateWhitelistAppIdsLocked()方法将系统应用白名单和用户应用的白名单合并，然后将白名单设置到PowerManagerService中

```
mLocalPowerManager.setDeviceIdleTempWhitelist(mTempWhitelistAppIdArr
```

第6步：初始化一些变量，默认系统的屏幕为开启，Doze模式默认为ACTIVE，默认为充电模式等

第7步：和PowerManagerService类似，将DeviceIdleController注册到ServiceManager和LocalService中。

OnStart一共大致有以上7个步骤，主要作用是读取系统白名单和应用白名单，并设置到PowerManagerService中。

下一步同样和PowerManangerService一样，当SystemSerivceReady之后回调onBootPhase方法，这个方法也比较简单，主要是也是做一些初始化的功能。

OnBootPhase方法

```
public void onBootPhase(int phase) {
        if (phase == PHASE_SYSTEM_SERVICES_READY) {
            synchronized (this) {
                ……
                //初始化部分传感器服务
                mSensorManager = (SensorManager) getContext().getSys
                mSigMotionSensor = mSensorManager.getDefaultSensor(S
                mLocationManager = (LocationManager) getContext().ge
                        Context.LOCATION_SERVICE);
                ……
                mAnyMotionDetector = new AnyMotionDetector(
                        (PowerManager) getContext().getSystemService
                        mHandler, mSensorManager, this);

                ……
                //注册电池变化等广播
                IntentFilter filter = new IntentFilter();
                filter.addAction(Intent.ACTION_BATTERY_CHANGED);
                filter.addAction(ACTION_STEP_IDLE_STATE);
                getContext().registerReceiver(mReceiver, filter);
                ……

               //更新白名单mLocalPowerManager.setDeviceIdleWhitelist(m
                //注册屏幕显示的回调方法
                mDisplayManager.registerDisplayListener(mDisplayList
                updateDisplayLocked();
            }
        }
    }
```

可以看出onBootPhase方法主要是初始化一些传感器，注册了电池改变的广播接收器，注册了屏幕显示变化的回调方法，最后调用updateDisplayLocked方法。

由于DeviceIdleController刚启动，初始化的时候，screenOn默认为true且屏幕状态未发生变化，最后直接调用becomeActiveLocked()方法，将Doze状态设置为ACTIVE,并初始化一些值。

至此，DeviceIdleController就基本启动和初始化完成了。逻辑比较简单，下面我们将讨论Doze模式的几种状态及切换逻辑

DeviceIdleController在onBootPhase方法中，注册了一个ACTION_BATTERY_CHANGED广播接收器，当电池状态发生变化的时候触发该广播接收器的onReceive方法。

```
if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction())) {
                int plugged = intent.getIntExtra("plugged", 0);
                updateChargingLocked(plugged != 0);
            }
```

当Battery状态发生变化的时候，首先先根据Intent的参数判断是否是充电状态，然后调updateChargingLocked方法

```
void updateChargingLocked(boolean charging) {
        if (DEBUG) Slog.i(TAG, "updateChargingLocked: charging=" + cl
        if (!charging && mCharging) {
            mCharging = false;
            if (!mForceIdle) {
                becomeInactiveIfAppropriateLocked();
            }
        } else if (charging) {
            mCharging = charging;
            if (!mForceIdle) {
                becomeActiveLocked("charging", Process.myUid());
            }
        }
    }
```

该方法中，根据当前的充电状态，如果是由充电状态变为未充电状态的时候，调用becomeInactiveIfAppropriateLocked()方法，修改Doze模式的状态为InActive状态，如果是充电状态则直接调用becomeActiveLocked方法修改Doze模式的状态为Active状态。

同时在onBootPhase方法中我们还设置了displayListener的监听方法，当屏幕显示状态发生变化的时候，回调该接口。

```
public void onDisplayChanged(int displayId) {
```

```
            if (displayId == Display.DEFAULT_DISPLAY) {
                synchronized (DeviceIdleController.this) {
                    updateDisplayLocked();
                }
            }
```

当Display发生变化的时候，最终调用updateDisplayLocked来更新状态

```
void updateDisplayLocked() {
        mCurDisplay = mDisplayManager.getDisplay(Display.DEFAULT_DISP
        boolean screenOn = mCurDisplay.getState() == Display.STATE_ON
        if (!screenOn && mScreenOn) {
            mScreenOn = false;
            if (!mForceIdle) {
                becomeInactiveIfAppropriateLocked();
            }
        } else if (screenOn) {
            mScreenOn = true;
            if (!mForceIdle) {
                becomeActiveLocked("screen", Process.myUid());
            }
        }
    }
```

逻辑也比较简单，首先获得当前显示状态，如果显示状态是由亮屏到灭屏，那么调用becomeInactiveIfAppropriateLocked()方法，将Doze模式的状态设置为InActive，如果当前状态是亮屏状态，这直接调用becomeActiveLocked将Doze模式的状态设置为Active。

Doze模式状态转换分析

becomeInactiveIfAppropriateLocked方法分析：

```
void becomeInactiveIfAppropriateLocked() {
        if (((!mScreenOn && !mCharging) || mForceIdle) && mEnabled &&
            mState = STATE_INACTIVE;
            resetIdleManagementLocked();
            scheduleAlarmLocked(mInactiveTimeout, false);
        }
    }
```

首先屏幕灭屏并且不是充电状态，Doze状态为Active的时候，

1.将状态修改为InActice

2.重置一些变量及状态

3.设置一个定时器，在一段时间mInactiveTimeout后触发。时间mInactiveTimeout=30min

```
void resetIdleManagementLocked() {
        //重置Idle和maintance状态中的时间间隔
        mNextIdlePendingDelay = 0;
        mNextIdleDelay = 0;
        //取消定时器
        cancelAlarmLocked();
        //取消传感器和定位监测
        cancelSensingAlarmLocked();
        cancelLocatingLocked();
        stopMonitoringSignificantMotion();
        mAnyMotionDetector.stop();
    }
```

将mNextIdlePendingDelay和mNextIdleDelay两个时间变量重置，取消定时器，停止定位和运动，位置检测。

becomeActiveLocked方法分析：

```
void becomeActiveLocked(String activeReason, int activeUid) {
        if (mState != STATE_ACTIVE) {
            scheduleReportActiveLocked(activeReason, activeUid);
            mState = STATE_ACTIVE;
            mInactiveTimeout = mConstants.INACTIVE_TIMEOUT;
            resetIdleManagementLocked();
        }
    }
```

主要是将Doze的状态设置为ACTIVE状态，并重置相关的变量。执行scheduleReportActiveLocked方法，该方法发送了一个MESSAGE_REPORT_ACTIVE的消息给Handler。

```
case MSG_REPORT_ACTIVE: {
                    String activeReason = (String)msg.obj;
                    int activeUid = msg.arg1;
                    boolean needBroadcast = msg.arg2 != 0;
```

```
                    mLocalPowerManager.setDeviceIdleMode(false);
                    try {
                        mNetworkPolicyManager.setDeviceIdleMode(fals
                        mBatteryStats.noteDeviceIdleMode(false, acti
                    } catch (RemoteException e) {
                    }
                    if (needBroadcast) {
                        getContext().sendBroadcastAsUser(mIdleIntent
                    }
                } break
```

Handler处理逻辑，主要告诉PowerManagerService退出Doze模式，网络和电量统计退出Doze模式。

该方法主要作用是修改Doze模式为Active,并通知相应的服务退出Doze模式。

下一步我们接着分析InActive状态。当拔掉充电电源的时候或者屏幕灭屏的时候，调用becomeInactiveIfAppropriateLocked进入InActive状态，当30min后，定时器触发后，广播接收器接收到ACTION_STEP_IDLE_STATE定时器触发，并调用了setIdleStateLocked方法。

setIdleStateLocked方法关键代码：

```
switch (mState) {
            case STATE_INACTIVE:
                startMonitoringSignificantMotion();
                scheduleAlarmLocked(mConstants.IDLE_AFTER_INACTIVE_TI
                mNextIdlePendingDelay = mConstants.IDLE_PENDING_TIMEO
                mNextIdleDelay = mConstants.IDLE_TIMEOUT;
                mState = STATE_IDLE_PENDING;
                break;
                ……
```

1.调用startMonitoringSignificantNotion()方法，注册SigMotion传感器，监测重要的运动事件。

2.重新设置一个定时器，同样在30min后触发。

3.设置mNextIdlePendingDelay的时间为5min,mNextIdleDelay的时间为60min

4.修改当前的Doze状态为IDLE_PENDING状态

startMonitoringSignificantNotion()方法注册了运动监测的传感器，当传感器触发的时候，回调SigMotionListener接口，最终调用significantMotionLocked()方法来处理

```
void significantMotionLocked() {
        mSigMotionActive = false;
        handleMotionDetectedLocked(mConstants.MOTION_INACTIVE_TIMEOU
    }

    void handleMotionDetectedLocked(long timeout, String type) {
        if (mState != STATE_ACTIVE) {
            scheduleReportActiveLocked(type, Process.myUid());
            mState = STATE_ACTIVE;
            mInactiveTimeout = timeout;
            cancelSensingAlarmLocked();
            becomeInactiveIfAppropriateLocked();
        }
    }
```

处理逻辑，当前的状态为Idle_pending，所以调用scheduleReportActiveLocked方法通知相关的服务退出Doze模式，将当前状态修改为Active,最终调用becomeInactiveIfAppropriateLocked重新进入InActive状态。根据此处逻辑可知，如果传感器监测到有特殊的运动就随时返回到InActive状态。

如果没有运动30min后触发定时器，再次进入setIdleStateLocked方法

```
case STATE_IDLE_PENDING:
                mState = STATE_SENSING;
                scheduleSensingAlarmLocked(mConstants.SENSING_TIMEOU
                cancelLocatingLocked();
                mAnyMotionDetector.checkForAnyMotion();
                mNotMoving = false;
                mLocated = false;
                mLastGenericLocation = null;
                mLastGpsLocation = null;
                break;
```

此时将当前的Doze状态设置为STATE_SENSING,同时设置了定时器，4min后触发。同时开启运动检测，mAnyMotionDetector.checkForAnyMotion()，该方法利用传感器，检测手机是否移动，我们来看下关键代码实现。

```
if (!mMeasurementInProgress && mAccelSensor != null) {
            if (mSensorManager.registerListener(mListener, mAccelSen
                    SAMPLING_INTERVAL_MILLIS * 1000)) {
                mWakeLock.acquire();
                mMeasurementInProgress = true;
                mRunningStats.reset();
            }

            Message msg = Message.obtain(mHandler, mMeasurementTimeo
            msg.setAsynchronous(true);
            mHandler.sendMessageDelayed(msg, ACCELEROMETER_DATA_TIME
        }
```

此处注册了个传感器，然后接受传感器的数据，然后延迟3s执行方法。

首先，接收数据接口，数据接口会判断当前的数据信息是否足够，若足够则调用status=stopOrientationMeasurementLocked()，根据当前传感器的数据计算出当前手机的状态，回调.onAnyMotionResult(status)方法。

或者，延迟3s再收集数据，调用status=stopOrientationMeasurementLocked()，根据当前传感器的数据计算出当前手机的状态，回调.onAnyMotionResult(status)方法。

```
public void onAnyMotionResult(int result) {
        if (result == AnyMotionDetector.RESULT_MOVED) {
            synchronized (this) {
                handleMotionDetectedLocked(mConstants.INACTIVE_TIMEO
            }
        } else if (result == AnyMotionDetector.RESULT_STATIONARY) {
            if (mState == STATE_SENSING) {
                synchronized (this) {
                    mNotMoving = true;
                    stepIdleStateLocked();
                }
            }
……
    }
```

当检测到手机的状态后，根据状态处理不同的逻辑

1.当手机处于MOVE状态的时候，执行handleMotionDetectedLocked方法，将状态重置为INACTIVE状态，通知各个服务退出IDLE模式。

2.当手机固定不动的时候，讲notMoveing变量置为true,同时执行

setIdleStateLocked方法，进入下一个状态。

```
case STATE_SENSING:
                mState = STATE_LOCATING;
                scheduleSensingAlarmLocked(mConstants.LOCATING_TIMEO
                mLocating = true;
                mLocationManager.requestLocationUpdates(mLocationReq
                        mHandler.getLooper());
                if (mLocationManager.getProvider(LocationManager.GPS_
                    mHaveGps = true;
                    mLocationManager.requestLocationUpdates(Location
                            mGpsLocationListener, mHandler.getLooper
                } else {
                    mHaveGps = false;
                }
                break;
```

当移动监测完成，或者4min后定时器触发，当前状态为STATE_LOCATION状态。在设置30s的定时器，同时调用系统定位，当定位完成回调定位完成接口

```
void receivedGenericLocationLocked(Location location) {
       ……
        mLocated = true;
        if (mNotMoving) {
            stepIdleStateLocked();
        }
    }

    void receivedGpsLocationLocked(Location location) {
        ……
        mLocated = true;
        if (mNotMoving) {
            stepIdleStateLocked();
        }
    }
```

当接收到定位信息且当前手机位置没有移动，就进入IDLE状态。

```
case STATE_LOCATING:
          cancelSensingAlarmLocked();
          cancelLocatingLocked();
          mAnyMotionDetector.stop();
```

```
        case STATE_IDLE_MAINTENANCE:
            scheduleAlarmLocked(mNextIdleDelay, true);
            mNextIdleDelay = (long)(mNextIdleDelay * mConstants.IDLE_F
            mNextIdleDelay = Math.min(mNextIdleDelay, mConstants.MAX_I
            mState = STATE_IDLE;
            mHandler.sendEmptyMessage(MSG_REPORT_IDLE_ON);
            break;
```

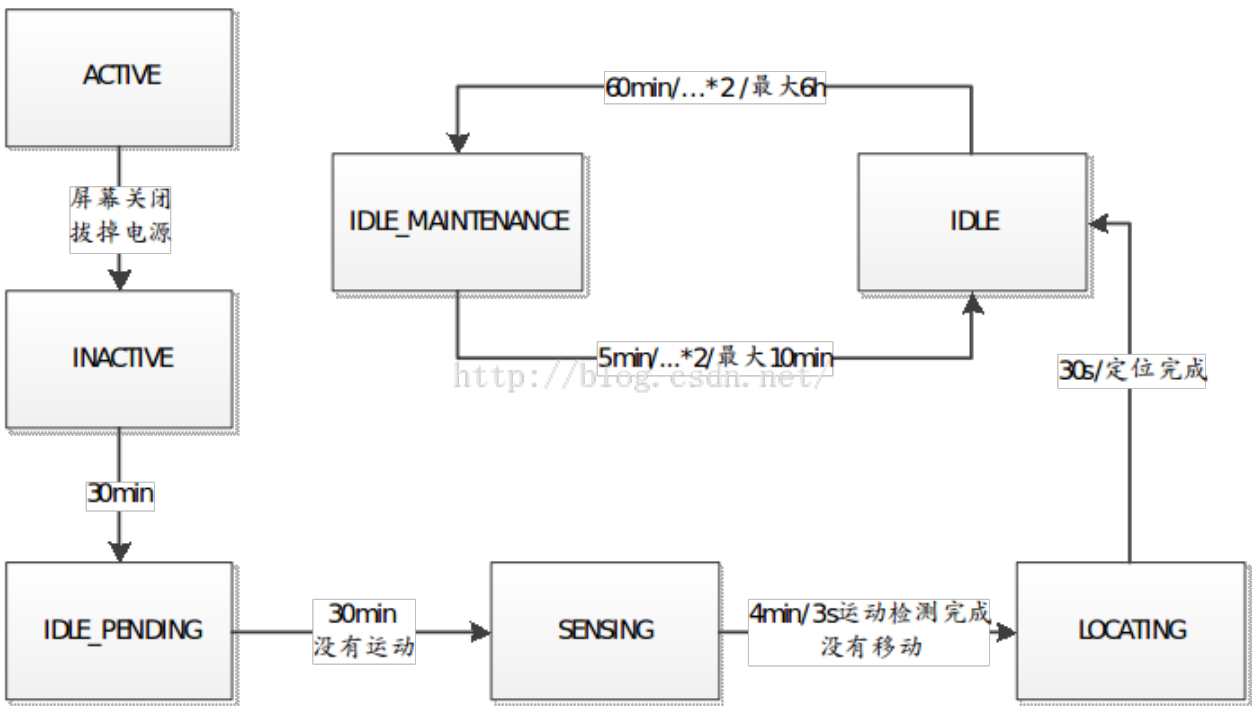
先取消上个状态的没有完成的定时器和定位，取消运动监测。设置下一个定时器mNextIdleDelay=60min后触发。修改mNextIdleDelay的值为当前的2倍。最大值为6h,将状态设置为IDLE状态，发送IDLE的handler消息，接收到消息通知相关的服务进入IDLE状态。

当60min过后，定时器触发，进入IDLE_MAINTENANCE状态。设置定时器5min后触发，修改下次时间为2倍，最大时间为10min,发送MSG_REPORT_IDLE_OFF消息，在Handler中处理，通知各个服务退出IDLE状态。

```
case STATE_IDLE:
                scheduleAlarmLocked(mNextIdlePendingDelay, false);
                mNextIdlePendingDelay = Math.min(mConstants.MAX_IDLE_
                        (long)(mNextIdlePendingDelay * mConstants.IDL
                mState = STATE_IDLE_MAINTENANCE;
                mHandler.sendEmptyMessage(MSG_REPORT_IDLE_OFF);
                break;
```

下次触发再次进入IDLE状态。

Doze模式的几个状态转换基本分析完成，下面是各个状态转换的流程图



Doze模式总共有7中状态，手机被操作的时候为Active状态，当手机关闭屏幕或者拔掉电源的时候，手机开始进入Doze模式，经过一系列的状态后最终进入IDLE状态，此时属于深度休眠状态，系统中的网络，Wakelock等服务都会停止运行，当60min过后，系统进入IDLE_MAINTENANCE状态，此时集中处理相关的任务，5min后再次进入IDLE状态，每次进入IDLE状态，时间都会是上次的2倍，最大时间限制为6h.

在这7中状态中，随时都会返回ACTIVE或者INACTIVE状态。当手机运动，或者点亮屏幕，插上电源等，系统会返回到ACTIVIE和INACTIVE状态。

PowerManagerService中Doze状态的处理

以上分析完了Doze几个状态之间的转换，下面我们分析下PowerManagerService中关于Doze的处理逻辑。

我们知道，当Doze模式转换到Idle状态之后，就会通知相关的服务进入IDLE状态，其中PowerManangerService处理的方法为localPowerManager.setDeviceIdle(true)

```
void setDeviceIdleModeInternal(boolean enabled) {
        synchronized (mLock) {
            if (mDeviceIdleMode != enabled) {
                mDeviceIdleMode = enabled;
                updateWakeLockDisabledStatesLocked();
            }
        }
    }
```

将PowerManagerService中mDeviceIdleMode的值设置为true，然后调用

updateWakeLockDisableStatesLocked()方法更新wakeLock的状态。

```
1  private void updateWakeLockDisabledStatesLocked() {
2          boolean changed = false;
3          final int numWakeLocks = mWakeLocks.size();
4          for (int i = 0; i < numWakeLocks; i++) {
5              final WakeLock wakeLock = mWakeLocks.get(i);
6              if ((wakeLock.mFlags & PowerManager.WAKE_LOCK_LEVEL_MASK
7                      == PowerManager.PARTIAL_WAKE_LOCK) {
8                  if (setWakeLockDisabledStateLocked(wakeLock)) {
9                      changed = true;
10                     if (wakeLock.mDisabled) {
11                         //当前wakeLock的mDisabled变量为true，释放掉该wak
12                         notifyWakeLockReleasedLocked(wakeLock);
13                     } else {
14                         notifyWakeLockAcquiredLocked(wakeLock);
15                     }
16                 }
17             }
18         }
19         if (changed) {
20             mDirty |= DIRTY_WAKE_LOCKS;
21             updatePowerStateLocked();
22         }
23     }
```

在更新WakeLock的信息时，遍历所有的wakeLock,只处理wakeLock的类型为PARTIAL_WAKE_LOCK的wakeLock。调用setWakeLockDisabledStateLocked(wakeLock)方法更新wakeLock的disable的状态值。在该方法中根据当前wakelock所有应用程序的appid来判断该程序在IDLE状态是是否可以正常运行，如果该appid合法且该appid不在设置的白名单中，该应用程序在IDLE状态时是不能运行的，将该wakeLock的disabled的值置为true

如果该appid在白名单中，那么该程序在IDLE状态允许运行，将该wakeLock的disable状态置为false.

```
1  private boolean setWakeLockDisabledStateLocked(WakeLock wakeLock) {
2          if ((wakeLock.mFlags & PowerManager.WAKE_LOCK_LEVEL_MASK)
3                  == PowerManager.PARTIAL_WAKE_LOCK) {
4              boolean disabled = false;
5              if (mDeviceIdleMode) {
6                  final int appid = UserHandle.getAppId(wakeLock.mOwne
7                  if (appid >= Process.FIRST_APPLICATION_UID &&
8                          Arrays.binarySearch(mDeviceIdleWhitelist, ap
9                          Arrays.binarySearch(mDeviceIdleTempWhitelist
10                         mUidState.get(wakeLock.mOwnerUid,
11                                 ActivityManager.PROCESS_STATE_CACHED_
12                                 > ActivityManager.PROCESS_STATE_FORE
13                     disabled = true;
14                 }
15             }
16             if (wakeLock.mDisabled != disabled) {
17                 wakeLock.mDisabled = disabled;
18                 return true;
19             }
20         }
21         return false;
22     }
```

如果该wakeLock的disabled变量更新成功，当wakeLock的disabled的值为true，表示IDLE状态次wakeLock不可用，调用notifyWakeLockReleasedLocked(wakelock)通知释放该wakeLock。

当wakeLock的disabled的值为false，表示IDLE状态此wakeLock可用，调用notifyWakeLockAcquiredLocked(wakeLock)通知获取该wakeLock。

相关TAG标签 Android Doze 模式分析

上一篇：【Android】Retrofit的使用(3)-对Retrofit进行简单的封装

下一篇：AndroidStudio使用SVN管理代码

相关文章

android libaray的创建和使用
android中json转换成List
android AudioTrack不能播放awr
android-仿QQtab
Android之ImageSwitcher,Gallery用法
android中调用webservice
Android ListView Adapter
Android异步操作AsyncTask
android database sqlite的使用
Android面试系列之一

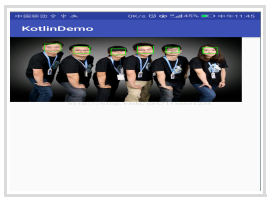
热门专题推荐 python div+css css教程 html5 html教程 jquery Android SDK php mysql oracle

图文推荐

Android零基础入门第

一个Activity中多个F

系统或JDK为32位时安



安卓人脸检测之FaceD

登录

来说两句吧...

畅言一下

还没有评论，快来抢沙发吧！

红黑联盟正在使用畅言