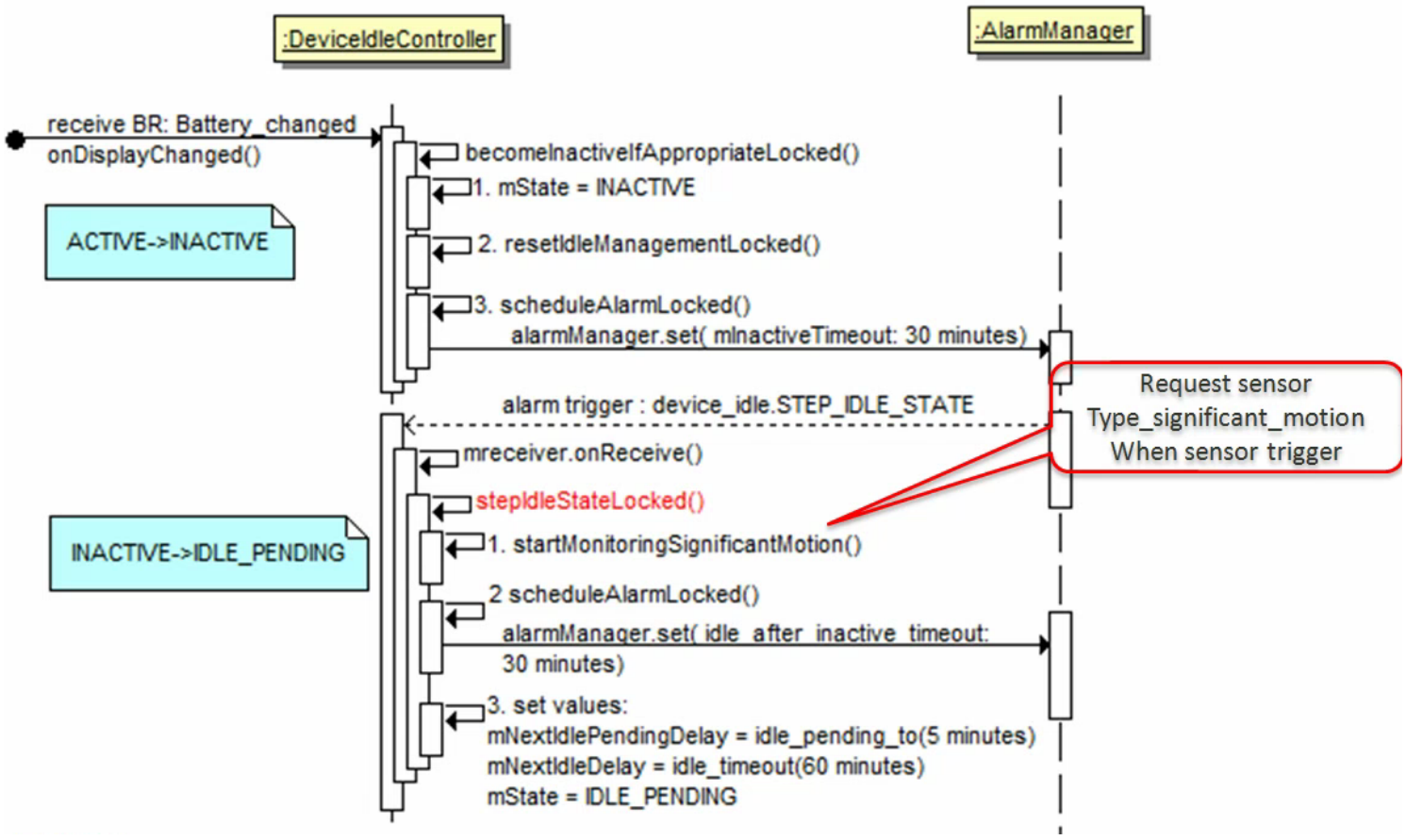
Android (/tags/#Android) PowerManager (/tags/#PowerManager) Doze (/tags/#Doze)

Android电源管理之Doze模式专题系列（三）

状态切换剖析之ACTIVE-->INACTIVE

Posted by Cheson on March 4, 2017

上一篇 Doze代码分布和状态机切换 (https://chendongqi.github.io/blog/2017/03/01/pm_doze_statemachine/)一文中介绍了Doze各个状态的含义和状态机切换的触发条件，这一篇中将从代码角度来剖析ACTIVE–>INACTIVE状态之间的切换。



上图即为从ACTIVE–>INACTIVE以及从INACTIVE–>IDLE_PENDING的时序图，本篇中先来看第一部分，如何从ACTIVE切换到INACTIVE。在ACTIVE状态下需要同时满足两个条件，不插充电器和灭屏，设备才会进入到INACTIVE状态。在DeviceIdleController.java中注册了一个广播接收器。

```
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override public void onReceive(Context context, Intent intent) {
            if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction())) {
                int plugged = intent.getIntExtra("plugged", 0);
                updateChargingLocked(plugged != 0);
            } else if (ACTION_STEP_IDLE_STATE.equals(intent.getAction())) {
                synchronized (DeviceIdleController.this) {
                    stepIdleStateLocked();
                }
            }
        }
    };
```

当接收到电池广播时，调用updateChargingLocked来更新充电状态

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

若从充电状态切换到了不充电状态，那么调用becomeInactiveIfAppropriateLocked来判断是否让设备进入INACTIVE状态

```
    void becomeInactiveIfAppropriateLocked() {
        if (DEBUG) Slog.d(TAG, "becomeInactiveIfAppropriateLocked()");
        if (((!mScreenOn && !mCharging) || mForceIdle) && mEnabled && mState == STATE_ACTIVE)
            // Screen has turned off; we are now going to become inactive and start
            // waiting to see if we will ultimately go idle.
            mState = STATE_INACTIVE;
            if (DEBUG) Slog.d(TAG, "Moved from STATE_ACTIVE to STATE_INACTIVE");
            resetIdleManagementLocked();
            scheduleAlarmLocked(mInactiveTimeout, false);
            EventLogTags.writeDeviceIdle(mState, "no activity");
        }
    }
```

这里刨去mForceIdle这个变量不管，需要的条件是，灭屏&&未充电，当前的状态mState为ACTIVE，并且Doze 功能是开启的，也就是说mEnabled为true。那么为什么又定义了mEnabled这个变量呢，这里再展开来说明下。首先这个变量的初始化在DeviceIdleController的onStart的时候。

```
    mEnabled = getContext().getResources().getBoolean(
                    com.android.internal.R.bool.config_enableAutoPowerModes);
```

由此可知这个值是从配置文件中读取出来的，可以从frameworks/base/core/res/res/values/config.xml中看到这个值的说明，并且看到了默认值为false。那么这么来说介绍了半天的Doze功能岂不是默认关闭的？

```
    <!-- Set this to true to enable the platform's auto-power-save modes like doze and
         app standby.  These are not enabled by default because they require a standard
         cloud-to-device messaging service for apps to interact correctly with the modes
         (such as to be able to deliver an instant message to the device even when it is
         dozing).  This should be enabled if you have such services and expect apps to
         correctly use them when installed on your device.  Otherwise, keep this disabled
         so that applications can still use their own mechanisms. -->
    <bool name="config_enableAutoPowerModes">false</bool>
```

非也，从之前介绍的信息中以及对config_enableAutoPowerModes的注释我们可以知道，在Doze的IDLE模式下是阻止app的网络连接的，那么那些需要实时推送消息的app怎么办呢？有一个方案就是利用google的cloud-to-device messaging来作为统一的消息推送服务器，也就是GCM。那么为了google为了要推广此项功能就在这里做了手脚了，在此设下一个小机关默认关闭 Doze，那么什么时候打开呢？要推理出来应该不难，google的那套东西基本都在GMS包中了，所以我们GMS包中再找找线索。果不其然，在GMS包中就将这个配置的值做了一次overlay了。也就是说正常情况下是只有带GMS的版本才能支持doze模式的。当然，作为强大的ODM厂商，我们也只需动动手指，让非GMS Load也支持doze功能。

```
    <!-- Enable doze mode -->
    <bool name="config_enableAutoPowerModes">true</bool>
```

继续言归正传，上面分析了接收到电池广播开始产生的变化，如果充电状态切换到拔出并且当时的屏幕状态mScreenOn为false的话，那么就触发进入INACTIVE。同理我们来看下如果是屏幕状态改变将经历怎么的变化。

DeviceIdleController中也定义了一个监听display的监听器

```
    private final DisplayManager.DisplayListener mDisplayListener
            = new DisplayManager.DisplayListener() {
        @Override public void onDisplayAdded(int displayId) {
        }

        @Override public void onDisplayRemoved(int displayId) {
        }

        @Override public void onDisplayChanged(int displayId) {
            if (displayId == Display.DEFAULT_DISPLAY) {
                synchronized (DeviceIdleController.this) {
                    updateDisplayLocked();
                }
            }
        }
    };
```

当有onDisplayChanged时调用updateDisplayLocked来更新display的状态

```
    void updateDisplayLocked() {
        mCurDisplay = mDisplayManager.getDisplay(Display.DEFAULT_DISPLAY);
        // We consider any situation where the display is showing something to be it on,
        // because if there is anything shown we are going to be updating it at some
        // frequency so can't be allowed to go into deep sleeps.
        boolean screenOn = mCurDisplay.getState() == Display.STATE_ON;
        if (DEBUG) Slog.d(TAG, "updateDisplayLocked: screenOn=" + screenOn);
        if (!screenOn && mScreenOn) {
            mScreenOn = false;
            if (!mForceIdle) {
                becomeInactiveIfAppropriateLocked();
            }
        } else if (screenOn) {
            mScreenOn = true;
            if (!mForceIdle) {
                becomeActiveLocked("screen", Process.myUid());
            }
        }
    }
```

这一段的逻辑和充电状态是一模一样的，当从亮屏切换到灭屏时触发becomeInactiveIfAppropriateLocked来进行是否进入INACTIVE的判断。其他不再重复。

到这里就讲完了如何从拔充电器或者灭屏来触发进入INACTIVE状态，那么进入INACTIVE时还需要做什么呢？我们再回过头来分析下becomeInactiveIfAppropriateLocked这个函数做了什么事情。

```
    void becomeInactiveIfAppropriateLocked() {
        if (DEBUG) Slog.d(TAG, "becomeInactiveIfAppropriateLocked()");
        if (((!mScreenOn && !mCharging) || mForceIdle) && mEnabled && mState == STATE_ACTIVE)
            // Screen has turned off; we are now going to become inactive and start
            // waiting to see if we will ultimately go idle.
            mState = STATE_INACTIVE;
            if (DEBUG) Slog.d(TAG, "Moved from STATE_ACTIVE to STATE_INACTIVE");
            resetIdleManagementLocked();
            scheduleAlarmLocked(mInactiveTimeout, false);
            EventLogTags.writeDeviceIdle(mState, "no activity");
        }
    }
```

当满足进入INACTIVE状态的条件时，先将设备状态mState置为INACTIVE，然后调用了resetIdleManagementLocked()和scheduleAlarmLocked(mInactiveTimeout, false)来做一些事情。先来看下resetIdleManagementLocked()

```
    void resetIdleManagementLocked() {
        mNextIdlePendingDelay = 0;
        mNextIdleDelay = 0;
        cancelAlarmLocked();
        cancelSensingAlarmLocked();
        cancelLocatingLocked();
        stopMonitoringSignificantMotion();
        mAnyMotionDetector.stop();
    }
```

这个函数中所做的事情就是重置，可以这样理解，从ACTIVE状态切换成INACTIVE状态是一个新的进入IDLE的开始，所有需要重置之前所有设过的ALARM、注册的Sensor等状态。然后再来看scheduleAlarmLocked(mInactiveTimeout, false)做了什么

```
    void scheduleAlarmLocked(long delay, boolean idleUntil) {
        if (DEBUG) Slog.d(TAG, "scheduleAlarmLocked(" + delay + ", " + idleUntil + ")");
        if (mSigMotionSensor == null) {
            // If there is no significant motion sensor on this device, then we won't schedule
            // alarms, because we can't determine if the device is not moving.  This effective
            // turns off normal exeuction of device idling, although it is still possible to
            // manually poke it by pretending like the alarm is going off.
            return;
        }
        mNextAlarmTime = SystemClock.elapsedRealtime() + delay;
        if (idleUntil) {
            mAlarmManager.setIdleUntil(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                    mNextAlarmTime, mAlarmIntent);
        } else {
            mAlarmManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                    mNextAlarmTime, mAlarmIntent);
        }
    }
```

这个方法中就是设置了一个Alarm在后面来唤醒系统做一些事情，那么问题来了：唤醒系统做什么？多久以后唤醒？

先解释唤醒系统做什么，还记得前面的状态机切换吗？INACTIVE状态之后需要切换到IDLE_PENDING状态，那么30分钟后系统处于suspend的状态怎么去切换呢，这里就是通过设置一个定时的ALARM来唤醒系统进行状态机的切换。这里也先透个底，doze模式下各个状态之间的切换基本上都是以这种Alarm的方式来驱动的。来看下这个Alarm中设置的 mAlarmIntent验证下。

```
    Intent intent = new Intent(ACTION_STEP_IDLE_STATE)
                .setPackage("android")
                .setFlags(Intent.FLAG_RECEIVER_REGISTERED_ONLY);
    mAlarmIntent = PendingIntent.getBroadcast(getContext(), 0, intent, 0);
```

```
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override public void onReceive(Context context, Intent intent) {
            if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction())) {
                int plugged = intent.getIntExtra("plugged", 0);
                updateChargingLocked(plugged != 0);
            } else if (ACTION_STEP_IDLE_STATE.equals(intent.getAction())) {
                synchronized (DeviceIdleController.this) {
                    stepIdleStateLocked();
                }
            }
        }
    };
```

果然方法就如所推测的那样，通过Alarm发送一个延时的广播，接收到ACTION_STEP_IDLE_STATE的广播之后调用stepIdeStateLocked方法来做状态的切换，注意：此方法为**DeviceIdleController**中状态机切换的核心方法。

再者就是这个闹钟唤醒的时间mInactiveTimeout，在DeviceIdleController服务onStart的时候这个值被初始化了

```
mInactiveTimeout = mConstants.INACTIVE_TIMEOUT;
```

而INACTIVE_TIMEOUT的值为

```
INACTIVE_TIMEOUT = mParser.getLong(KEY_INACTIVE_TIMEOUT,
                    !COMPRESS_TIME ? 30 * 60 * 1000L : 3 * 60 * 1000L);
```

这个COMPRESS_TIME默认定义成了false，也就是不压缩时间，所以我们取到的默认值为30分钟。介绍此处也就是说不同的ODM就可以修改这个值来使得更快的实现Doze下的状态转换以更早进入IDLE，此处是可以斟酌的。

以上就将ACTIVE如何切换到INACTIVE的流程以及进入到INACTIVE状态时所做的事。

参考资料：

eCourse：M Doze&AppStandby (https://onlinesso.mediatek.com/Pages/eCourse.aspx?001=002&002=002002&003=002002001&itemId=560&csId=%257B433b9ec7-cc31-43c3-938c-6dfd42cf3b57%257D%2540%257Bad907af8-9a88-484a-b020-ea10437dadf8%257D)

eService：关于doze模式是否支持的疑问 (http://eservice.mediatek.com/eservice-portal/issue_manager/update/2062164)

PREVIOUS
ANDROID性能优化之进阶资料 **(/2017/03/04/PERFORMANCE_DOC/)**

NEXT
ANDROID电源管理之DOZE模式专题系列（四） **(/2017/03/04/PM_DOZE_INACTIVE_TO_PENDING/)**

FEATURED TAGS (/tags/)

前端 (/tags/#前端) Android (/tags/#Android) frameworks (/tags/#frameworks) AlarmManager (/tags/#AlarmManager) Performance (/tags/#Performance) systrace (/tags/#systrace) PowerManager (/tags/#PowerManager) Wakelock (/tags/#Wakelock) Guitar (/tags/#Guitar) 民谣 (/tags/#民谣) 赵雷 (/tags/#赵雷) Doze (/tags/#Doze) Android Performance Patterns (/tags/#Android Performance Patterns)

FRIENDS

待遇见志同道合的你 (https://github.com)　小明 (http://www.betterming.cn)