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

# Android电源管理之Doze模式专题系列（五）

状态切换剖析之IDLE_PENDING-->SENSING

*Posted by Cheson on March 4, 2017*

依照惯例，在介绍下一个状态切换之前，先来回顾一下上一个状态中的伏笔。在 Doze模式专题（四）——状态切换剖析之INACTIVE–>IDLE_PENDING (https://chendongqi.github.io/blog/2017/03/04/pm_doze_inactive_to_pending/)这篇中讲到了状态切换到IDLE_PENDING时需要做的事情，其中之一就是调用了scheduleAlarmLocked(mConstants.IDLE_AFTER_INACTIVE_TIMEOUT, false)设定了一个Alarm来在30分钟之后作为下一个状态切换的发起。于是，下面的故事就发生在这个动作的30分钟之后。

当Alarm被触发时，Broadcast将再次接收到ACTION_STEP_IDLE_STATE这个广播，然后调用stepIdleStateLocked()这个核心方法来进行状态切换的一系列处理。我们来看一下当前状态为STATE_IDLE_PENDING时会怎么样进行判断和切换。

```
    case STATE_IDLE_PENDING:
        mState = STATE_SENSING;
        if (DEBUG) Slog.d(TAG, "Moved from STATE_IDLE_PENDING to STATE_SENSING.");
        EventLogTags.writeDeviceIdle(mState, "step");
        scheduleSensingAlarmLocked(mConstants.SENSING_TIMEOUT);
        cancelSensingAlarmLocked();
        cancelLocatingLocked();
        mAnyMotionDetector.checkForAnyMotion();
        mNotMoving = false;
        mLocated = false;
        mLastGenericLocation = null;
        mLastGpsLocation = null;
        break;
```

首先将当前的状态mState改写为了STATE_SENSING，然后还是调用了scheduleSensingAlarmLocked来设定了一个4分钟的Alarm，看过前几个状态切换的很容易就明白这个Alarm一定是为了触发下一个状态的切换。然而下一个步骤就有点难以理解了，紧接着又调用了cancelSensingAlarmLocked方法。

```
    void cancelSensingAlarmLocked() {
        if (mSensing) {
            if (DEBUG) Slog.d(TAG, "cancelSensingAlarmLocked()");
            mAlarmManager.cancel(mSensingAlarmIntent);
            mSensing = false;
        }
    }
```

而这个方法取消的Alarm正是中的Intent正是前面设置的mSensingAlarmIntent。这么来看在IDLE_PENDING切换到SENSING状态时，设定了一个Alarm然后又立即取消了，这么来看不是做了两个无效的动作吗？没错，代码看到这里我也没明白这个意义，继续往下看。然后调用了cancelLocatingLocked来取消了位置监听。

```
    void cancelLocatingLocked() {
        if (mLocating) {
            mLocationManager.removeUpdates(mGenericLocationListener);
            mLocationManager.removeUpdates(mGpsLocationListener);
            mLocating = false;
        }
    }
```

其实这mGenericLocationListener和mGpsLocationListener这两个监听器是在SENSING状态时才被注册的，所以这边即将要进入SENSING之前需要先重置下状态。然后调用了mAnyMotionDetector.checkForAnyMotion()，这个函数在AnyMotionDetector.java中

```
    public void checkForAnyMotion() {
      if (DEBUG) Slog.d(TAG, "checkForAnyMotion(). mState = " + mState);
        if (mState != STATE_ACTIVE) {
            mState = STATE_ACTIVE;
            if (DEBUG) Slog.d(TAG, "Moved from STATE_INACTIVE to STATE_ACTIVE.");
            mCurrentGravityVector = null;
            mPreviousGravityVector = null;
            startOrientationMeasurement();
        }
    }
```

这里核心的步骤就是掉用startOrientationMeasurement来进行行为的检测

```
    private void startOrientationMeasurement() {
        if (DEBUG) Slog.d(TAG, "startOrientationMeasurement: mMeasurementInProgress=" +
            mMeasurementInProgress + ", (mAccelSensor != null)=" + (mAccelSensor != null));

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

            Message msg = Message.obtain(mHandler, mMeasurementTimeout);
            msg.setAsynchronous(true);
            mHandler.sendMessageDelayed(msg, ACCELEROMETER_DATA_TIMEOUT_MILLIS);
        }
    }
```

这里注册了一个加速度计的监听器，来检测设备的行动状态。

```
    private final SensorEventListener mListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            int status = RESULT_UNKNOWN;
            synchronized (mLock) {
                Vector3 accelDatum = new Vector3(SystemClock.elapsedRealtime(), event.values[0
                        event.values[1], event.values[2]);
                mRunningStats.accumulate(accelDatum);

                // If we have enough samples, stop accelerometer data acquisition.
                if (mRunningStats.getSampleCount() >= mNumSufficientSamples) {
                    status = stopOrientationMeasurementLocked();
                }
            }
            if (status != RESULT_UNKNOWN) {
                mCallback.onAnyMotionResult(status);
            }
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
        }
    };
```

在最后检测的状态不等于RESULT_UNKNOWN会回调onAnyMotionResult来返回结果，这个回调函数又会把流程带回到DeviceIdleController中。然后状态切换即将开始。

```
@Override
public void onAnyMotionResult(int result) {
    if (DEBUG) Slog.d(TAG, "onAnyMotionResult(" + result + ")");
    if (result == AnyMotionDetector.RESULT_MOVED) {
        if (DEBUG) Slog.d(TAG, "RESULT_MOVED received.");
        synchronized (this) {
            handleMotionDetectedLocked(mConstants.INACTIVE_TIMEOUT, "sense_motion");
        }
    } else if (result == AnyMotionDetector.RESULT_STATIONARY) {
        if (DEBUG) Slog.d(TAG, "RESULT_STATIONARY received.");
        if (mState == STATE_SENSING) {
            // If we are currently sensing, it is time to move to locating.
            synchronized (this) {
                mNotMoving = true;
                stepIdleStateLocked();
            }
        } else if (mState == STATE_LOCATING) {
            // If we are currently locating, note that we are not moving and step
            // if we have located the position.
            synchronized (this) {
                mNotMoving = true;
                if (mLocated) {
                    stepIdleStateLocked();
                }
            }
        }
    }
}
```

当检测结果为MOVED时，调用handleMotionDetectedLocked重新去将状态切成ACTIVE或者INACTIVE（满足INACTIVE条件时），如果结果为RESULT_STATIONARY，当前状态为SENSING时，就调用stopIdleStateLocked来进行状态切换，也就是进入到LOCATING状态，那么如果当前为LOCATING时，需要另外对mLocated变量进行判断才决定是否进行下一次切换，这个状态切换看下一篇。

讲到这里也就已经剖析了从IDLE_PENDING到SENSING的切换，以及在SENSING状态时所做的时和下一次切换的准备。那么再回过头来看下最开始的疑问，在进入SENSING状态时为什么设置了一个ALARM来作为状态切换的唤醒，然后又取消了？因为SENSING状态到下一个状态的切换发起者不是由ALARM来唤醒的，而是根据onAnyMotionResult这个回调函数的结果来处理，所以之前设置的Alarm根本就没有用是可以取消掉的，然后为什么要多此一举呢？也没想明白设计者的目的，也许是为了和之前的几个状态切换保持一致的风格吧，或者以后会更明白的。

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

**NEXT**
ALARMMANAGERSERVICE之ALARMGROUP机制剖析 **(/2017/03/06/ALARMGROUP/)**

**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)

● (https://twitter.com/chendongqi)

● (https://www.zhihu.com/people/chendongqi)

● (http://weibo.com/chendongqi) ● (https://www.facebook.com/chendongqi)

● (https://github.com/chendongqi)

● (https://www.linkedin.com/in/firstname-lastname-idxxxx)



Theme by Cheson (https://github.com/chendongqi/blog) | Star 1