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

Android电源管理之Doze模式专题系列（四）

状态切换剖析之INACTIVE-->IDLE_PENDING

Posted by Cheson on March 4, 2017

本篇介绍Doze模式的第二种状态切换，从INACTIVE到IDLE_PENDING。

首先回顾下 Doze模式专题（三）——状态切换剖析之ACTIVE–>INACTIVE (https://chendongqi.github.io/blog/2017/03/04/pm_doze_active_to_inactivie/)一文中介绍的后半部分，以续上这一篇的内容。之前已经介绍了进入INACTIVE状态之后会做两件事，第一是重置所有变量、Alarm及Sensor的状态等以迎接新一轮的状态机切换；第二就是设置了一个Alarm在30分钟（默认时间，可修改）后唤醒系统，此Alarm就是为下一个状态切换埋下的触发器。那么我们这一篇就接着这个Alarm的代码来继续分析。

这个Alarm在定时到了之后会发送一个action为ACTION_STEP_IDLE_STATE的广播，这个action也就是标示Doze模式状态切换的action

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

在BroadcastReceiver中接受到广播后调用stepIdleStateLocked来进行下一步的状态切换动作

```
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();
            }
        }
    }
};
```

在stepIdleStateLocked方法中针对当前所处的不同状态有不同的处理逻辑，这篇中我们只看当前为INACTIVE的状态。

```
case STATE_INACTIVE:
    // We have now been inactive long enough, it is time to start looking
    // for significant motion and sleep some more while doing so.
    startMonitoringSignificantMotion();// 监听SignificantMotion这个Sensor的数据
    scheduleAlarmLocked(mConstants.IDLE_AFTER_INACTIVE_TIMEOUT, false);// 设一个Alarm来触发了
    // Reset the upcoming idle delays.
    mNextIdlePendingDelay = mConstants.IDLE_PENDING_TIMEOUT;
    mNextIdleDelay = mConstants.IDLE_TIMEOUT;
    mState = STATE_IDLE_PENDING;
    if (DEBUG) Slog.d(TAG, "Moved from STATE_INACTIVE to STATE_IDLE_PENDING.");
    EventLogTags.writeDeviceIdle(mState, "step");
    break;
```

这个方法中总的来说做了三件事：1）调用startMonitoringSignificantMotion()开始监听

SignificantMotion这个Sensor的数据；2）调用scheduleAlarmLocked(mConstants.IDLE_AFTER_INACTIVE_TIMEOUT, false)方法再次设一个Alarm来触发下一个状态的切换；3）重置已将到来的状态切换的时间。

首先来看startMonitoringSignificantMotion()

```
    void startMonitoringSignificantMotion() {
        if (DEBUG) Slog.d(TAG, "startMonitoringSignificantMotion()");
        if (mSigMotionSensor != null && !mSigMotionActive) {
            mSensorManager.requestTriggerSensor(mSigMotionListener, mSigMotionSensor);
            mSigMotionActive = true;
        }
    }
```

这个方法只是注册了Sensor的监听器，具体的行为需要看listener。在看具体逻辑之前需要先介绍下这个SignificantMotionDetecter的Sensor，此Sensor为M版本中新加的一个Sensor，一下为google官方文档中的一段说明：

> *A significant motion detector triggers when the detecting a “significant motion”: a motion that might lead to a change in the user location. Examples of such significant motions are: walking or biking sitting in a moving car, coach or train Examples of situations that do not trigger significant motion: phone in pocket and person is not moving phone is on a table and the table shakes a bit due to nearby traffic or washing machine At the high level, the significant motion detector is used to reduce the power consumption of location determination. When the localization algorithms detect that the device is static, they can switch to a low power mode, where they rely on significant motion to wake the device up when the user is changing location.*

概括的来说这个Sensor也就是检测有意义的行为的，细微的非用户行为的不会触发此Sensor，这个比之前的一些Sensor如加速度传感器、GSensor等的数据变化来的更加有意义。

大致了解了SignificantMotion之后再来继续看它的listener，当onTrigger被回调时，调用significantMotionLocked来处理

```
    private final TriggerEventListener mSigMotionListener = new TriggerEventListener() {
        @Override public void onTrigger(TriggerEvent event) {
            synchronized (DeviceIdleController.this) {
                significantMotionLocked();
            }
        }
    };
```

然后重置Sensor的active状态以可以被再次注册，再调用handleMotionDetectedLocked来继续处理，注意这里的Delay时间为10分钟

```
    void significantMotionLocked() {
        if (DEBUG) Slog.d(TAG, "significantMotionLocked()");
        // When the sensor goes off, its trigger is automatically removed.
        mSigMotionActive = false;
        handleMotionDetectedLocked(mConstants.MOTION_INACTIVE_TIMEOUT, "motion");// 10 * 60 *
    }
```

而最后真正处理的逻辑就在handleMotionDetectedLocked中了

```
    void handleMotionDetectedLocked(long timeout, String type) {
        // The device is not yet active, so we want to go back to the pending idle
        // state to wait again for no motion.  Note that we only monitor for significant
        // motion after moving out of the inactive state, so no need to worry about that.
        if (mState != STATE_ACTIVE) {
            scheduleReportActiveLocked(type, Process.myUid());
            mState = STATE_ACTIVE;
            mInactiveTimeout = timeout;
            EventLogTags.writeDeviceIdle(mState, type);
            becomeInactiveIfAppropriateLocked();
        }
    }
```

在这个方法中如果当前状态不是ACTIVE，就会做几件事情，首先调用scheduleReportActiveLocked来报告当前的状态

```
    case MSG_REPORT_ACTIVE: {
            String activeReason = (String)msg.obj;
            int activeUid = msg.arg1;
            boolean needBroadcast = msg.arg2 != 0;
            EventLogTags.writeDeviceIdleOffStart(
                    activeReason != null ? activeReason : "unknown");
            mLocalPowerManager.setDeviceIdleMode(false);
            try {
                /// M: integrate Doze and App Standby @{
                if(null != getDataShapingService()) {
                    mDataShapingManager.setDeviceIdleMode(false);
                }
                /// integrate Doze and App Standby @}
                mNetworkPolicyManager.setDeviceIdleMode(false);
                mBatteryStats.noteDeviceIdleMode(false, activeReason, activeUid);
            } catch (RemoteException e) {
            }
            if (needBroadcast) {
                getContext().sendBroadcastAsUser(mIdleIntent, UserHandle.ALL);
            }
            EventLogTags.writeDeviceIdleOffComplete();
```

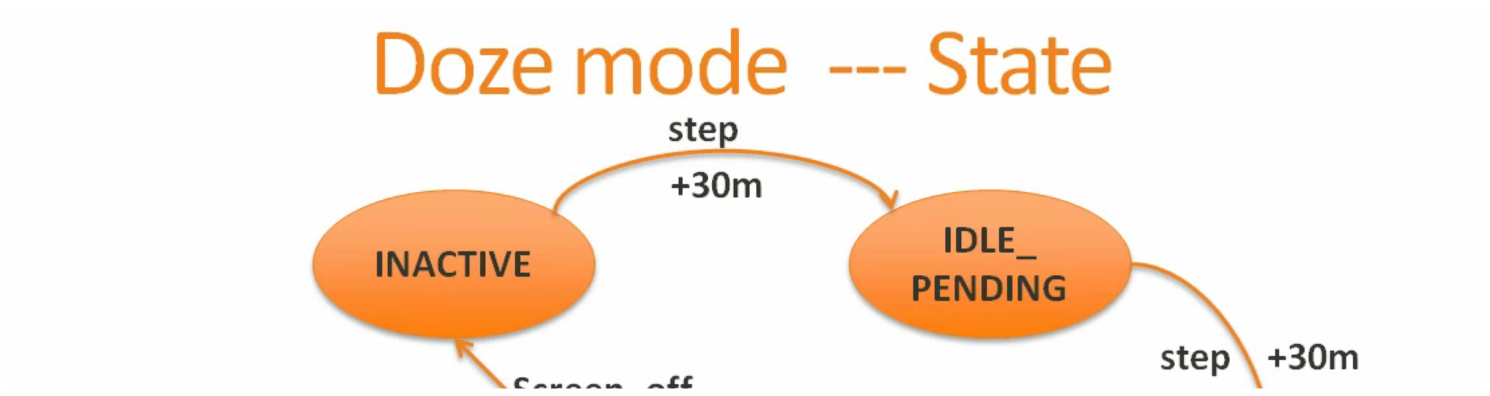
这里会通知各个Manager来设置DeviceIdle的状态为false。然后就当前状态改写成为ACTIVE，并且将mInactiveTimeout的时间设置成了10分钟，也就是说如果由于SignificantMotion检测到的动作而退出了INACTIVE状态，那么再次从INACTIVE切换到IDLE_PENDING的时间就从初识的30分钟缩短到了10分钟。然后调用becomeInactiveIfAppropriateLocked来判断是否可以再次直接进入到INACTIVE状态。

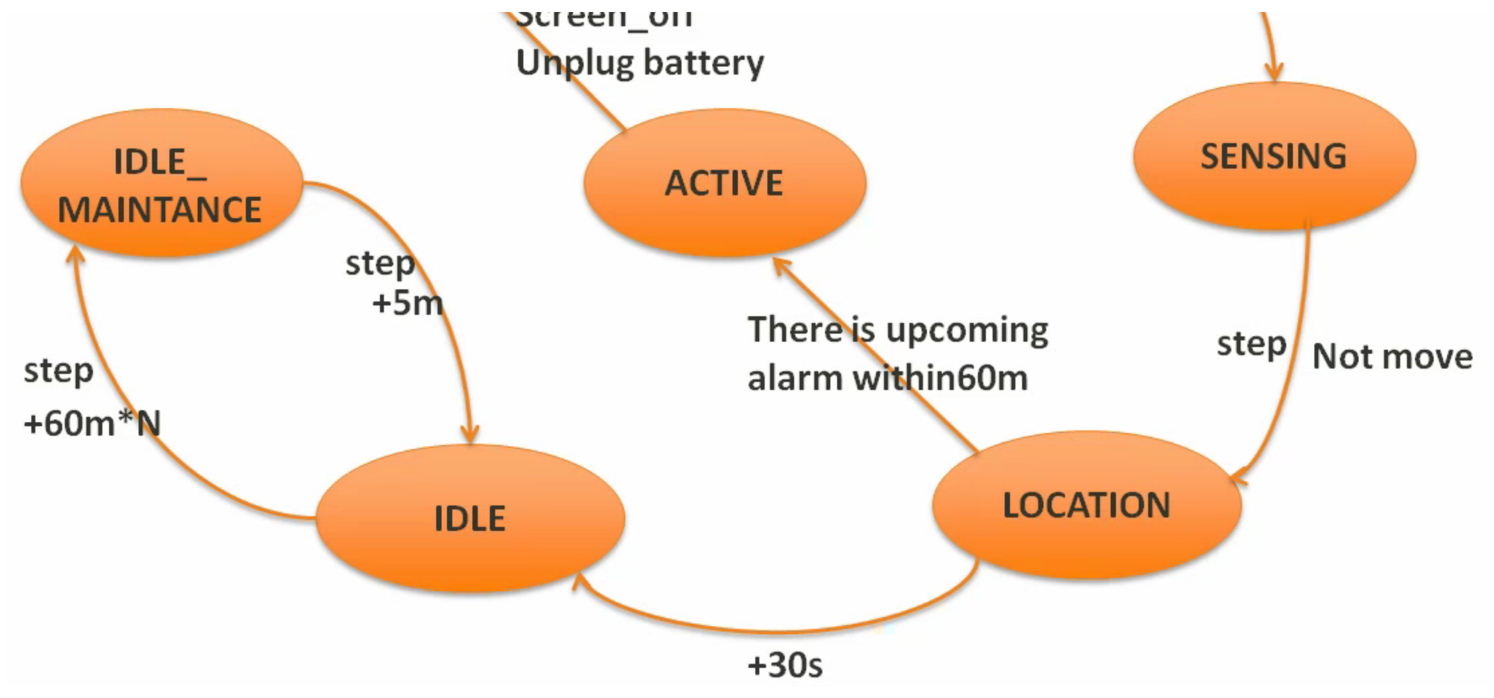
介绍完了SignificantMotion的监听，我们继续来看在INACTIVE状态中将要切换到IDLE_PENDING状态时做的第二件事情scheduleAlarmLocked(mConstants.IDLE_AFTER_INACTIVE_TIMEOUT, false)，这里就是触发下一次状态切换的定时器，将在30分钟后触发进行下一次的状态切换，也就是从IDLE_PENDINGQ切换到SENSING。

然后第三件事就是重置几个Delay的时间以及完成当前状态的修改，正式切入到IDLE_PENDING

```
    // Reset the upcoming idle delays.
    mNextIdlePendingDelay = mConstants.IDLE_PENDING_TIMEOUT;// 30 * 1000L
    mNextIdleDelay = mConstants.IDLE_TIMEOUT;// 60 * 60 * 1000L
    mState = STATE_IDLE_PENDING;
```

这里的delay时间是什么呢？还记得 Doze代码分布和状态机切换 (https://chendongqi.github.io/blog/2017/03/01/pm_doze_statemachine/)介绍中的状态机切换图吗





30秒就是从LOCATION状态切换到IDLE的delay时间，而60分钟就是一个IDLE周期的超时时间，超时之后就进入到IDLE_MAINTANCE

以上就是从INACTIVE切换到IDLE_PENDING的整个流程。

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（三）
(/2017/03/04/PM_DOZE_ACTIVE_TO_INACTIVIE/)

NEXT
ANDROID电源管理之DOZE模式专题系列（五）
(/2017/03/04/PM_DOZE_PENDING_TO_SENSING/)

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)