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

Android电源管理之Doze模式专题系列（七）

状态切换剖析之Locating-->IDLE/IDLE_MAINTENANCE-->IDLE

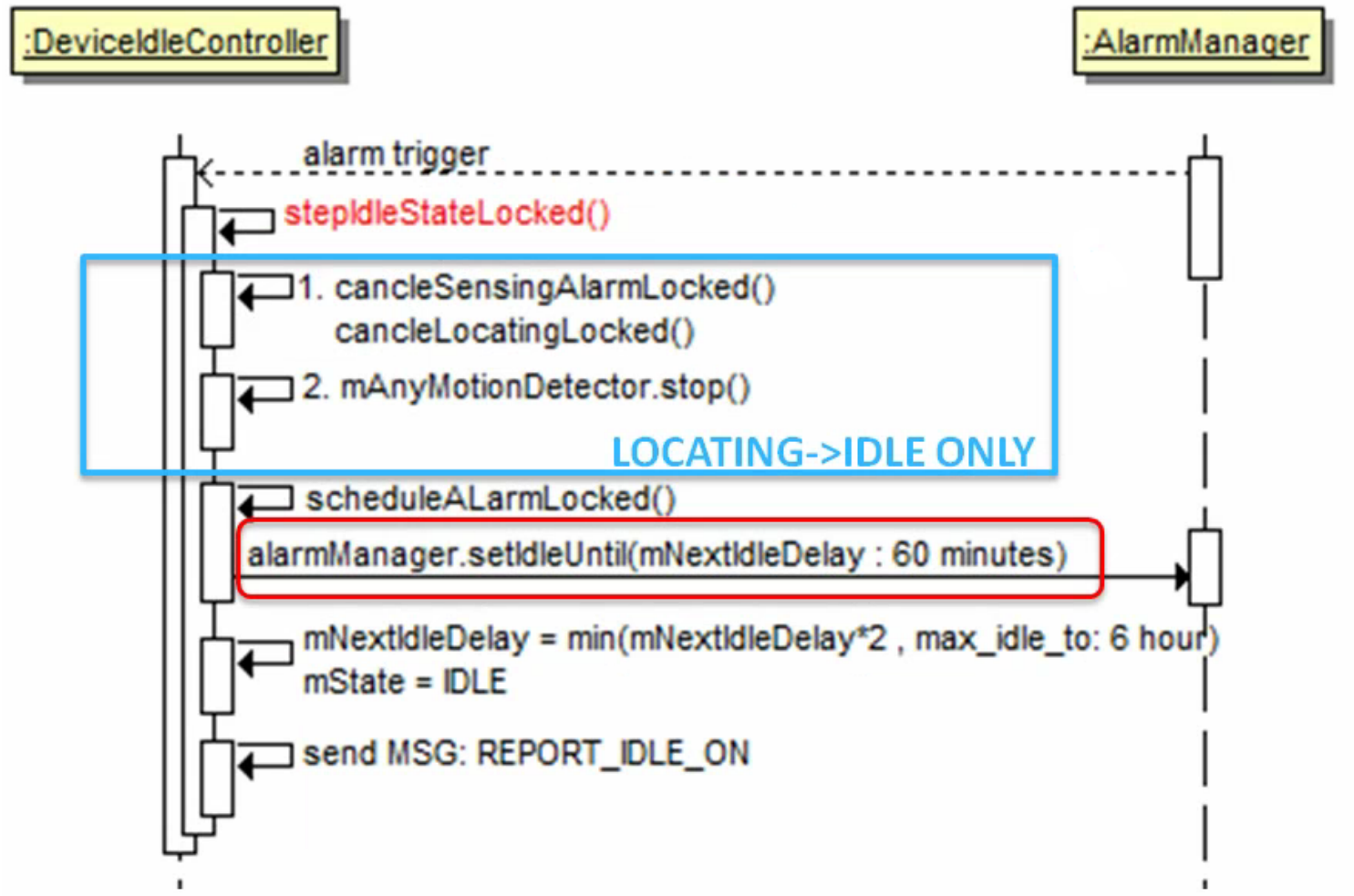
Posted by Cheson on April 7, 2017

在状态切换剖析之SENSING–>LOCATION (https://chendongqi.github.io/blog/2017/03/15/pm_doze_sensing_to_location/)中设备的状态已经进入到了LOCATING，接下来就只差临门一脚设备就能进入一种低功耗状态，也就是IDLE状态，而在IDLE状态下，设备又会定期的进入到IDLE_MAINTENANCE状态来处理之前被pending的任务，然后又进入到IDLE状态，从Locating–>IDLE和IDLE_MAINTENANCE–>IDLE两种状态的切换不过很相似，所以这一篇合在一起介绍。

从代码中也能看到，从LOCATING切换到IDLE只比从IDLE_MAINTENANCE切换到IDLE时多了三个动作，取消之前的Alarm，取消位置服务，停止动作监听。

```
    case STATE_LOCATING:
        cancelSensingAlarmLocked();
        cancelLocatingLocked();
        mAnyMotionDetector.stop();
    case STATE_IDLE_MAINTENANCE:
        scheduleAlarmLocked(mNextIdleDelay, true);
        if (DEBUG) Slog.d(TAG, "Moved to STATE_IDLE. Next alarm in " + mNextIdleDelay + " ms."
        mNextIdleDelay = (long)(mNextIdleDelay * mConstants.IDLE_FACTOR);
        if (DEBUG) Slog.d(TAG, "Setting mNextIdleDelay = " + mNextIdleDelay);
        mNextIdleDelay = Math.min(mNextIdleDelay, mConstants.MAX_IDLE_TIMEOUT);
        mState = STATE_IDLE;
        EventLogTags.writeDeviceIdle(mState, "step");
        mHandler.sendEmptyMessage(MSG_REPORT_IDLE_ON);
        break;
```

上面代码的时序图如下



如果当前为LOCATING状态，当之前在SENSING状态设下的30秒Alarm被触发时，调用stepIdleStateLocked方法，进入从LOCATING到IDLE状态的切换。首先取消Sensing状态下设置的Alarm（mSensingAlarmIntent），然后取消位置监听服务（mGenericLocationListener和mGpsLocationListener），接着停止了动作监听。因为这个case分支里没有break，所以直接进入到下一个case的代码中，也是就直接从IDLE_MAINTENANCE状态切换到IDLE状态。首先也是设置一个Alarm，这个Alarm就是用来唤醒系统进入到IDLE_MAINTENANCE状态处理pending的任务。这个设置Alarm的方式有点特殊，来看下实现。

```
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);
        }
    }
```

第一个需要关注的点是传入的delay时间，并不是一个固定的值，而是动态计算mNextIdleDelay的值作为参数传入。mNextIdleDelay的值计算关系到以下三个地方，首先在状态进入INACTIVE时，mNextIdleDelay就被初始化成了IDLE_TIMEOUT，也就是默认的60分钟。在首次进入IDLE时Alarm的延迟时间就是60分钟，然后mNextIdleDelay的值就被重新计算，乘以IDLE_FACTOR（2），然后和MAX_IDLE_TIMEOUT（6小时）比较取小的那个。

```
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();
    scheduleAlarmLocked(mConstants.IDLE_AFTER_INACTIVE_TIMEOUT, false);
    // Reset the upcoming idle delays.
    mNextIdlePendingDelay = mConstants.IDLE_PENDING_TIMEOUT;
    mNextIdleDelay = mConstants.IDLE_TIMEOUT;
```

```
mNextIdleDelay = (long)(mNextIdleDelay * mConstants.IDLE_FACTOR);
if (DEBUG) Slog.d(TAG, "Setting mNextIdleDelay = " + mNextIdleDelay);
mNextIdleDelay = Math.min(mNextIdleDelay, mConstants.MAX_IDLE_TIMEOUT);
```

```
IDLE_TIMEOUT = mParser.getLong(KEY_IDLE_TIMEOUT, !COMPRESS_TIME ? 60 * 60 * 1000L : 6 * 60 * 1
MAX_IDLE_TIMEOUT = mParser.getLong(KEY_MAX_IDLE_TIMEOUT, !COMPRESS_TIME ? 6 * 60 * 60 * 1000L
IDLE_FACTOR = mParser.getFloat(KEY_IDLE_FACTOR, 2f);
```

因此这里也印证了IDLE时期的持续时间越来越长，第一次为一小时，后面每次都翻倍，但最大不会超过6小时。第二个需要关注的点是调用的设置Alarm的接口。传入的idleUntil参数为true时，会使用setIdleUntil这个接口来设置Alarm，此接口为添加的一个新的方法（ARM板子也同时新加了底层的接口来支持）方法的注释参考AlarmManager.java中

```
/**
  * Schedule an idle-until alarm, which will keep the alarm manager idle until
  * the given time.
  * @hide
  */
public void setIdleUntil(int type, long triggerAtMillis, PendingIntent operation) {
    setImpl(type, triggerAtMillis, WINDOW_EXACT, 0, FLAG_IDLE_UNTIL, operation, null, null);
}
```

其最终实现是在AlarmManagerService.java中的setImplLocked

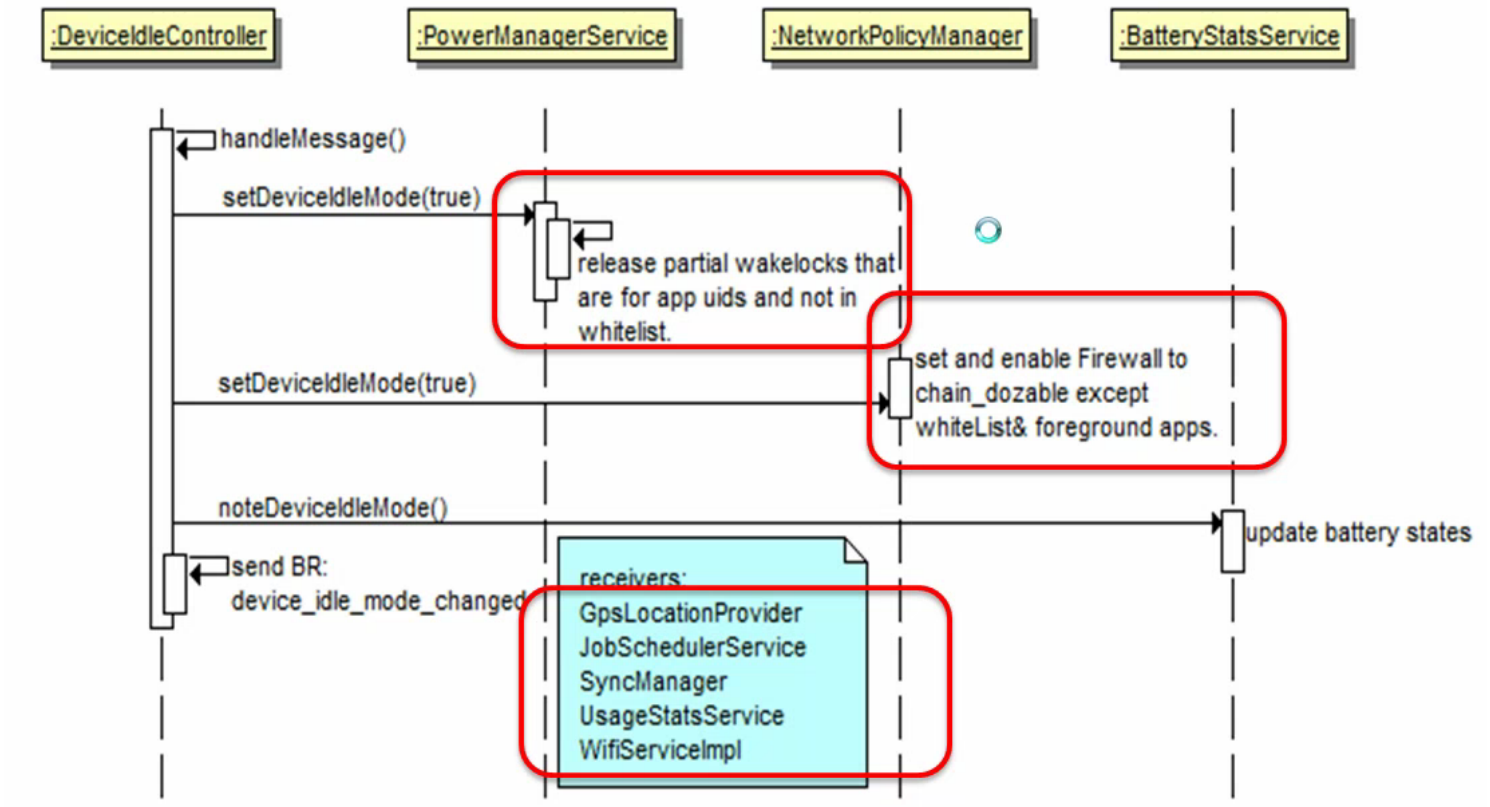
```
} else if (mPendingIdleUntil != null) {
    // We currently have an idle until alarm scheduled; if the new alarm has
    // not explicitly stated it wants to run while idle, then put it on hold.
    if ((a.flags&(AlarmManager.FLAG_ALLOW_WHILE_IDLE
            | AlarmManager.FLAG_ALLOW_WHILE_IDLE_UNRESTRICTED
            | AlarmManager.FLAG_WAKE_FROM_IDLE))
            == 0) {
        mPendingWhileIdleAlarms.add(a);
        return;
    }
}
```

如果由Alarm通过setIdleUntil被schedule之后，那么mPendingIdleUntil就会被赋一个alarm的实例，也就不为空，后面设置的Alarm中的flag如果不包含FLAG_ALLOW_WHILE_IDLE或者FLAG_ALLOW_WHILE_IDLE_UNRESTRICTED或者FLAG_WAKE_FROM_IDLE，那么这些Alarm都会被加入到pending list中去。这里也就实现了进入**IDLE**状态之后，把**Alarm**挂起处理的机制。 切换到IDLE状态的第二步是修改当前状态为IDLE，然后发送一个MSG_REPORT_IDLE_ON的消息。

```
mState = STATE_IDLE;
EventLogTags.writeDeviceIdle(mState, "step");
mHandler.sendEmptyMessage(MSG_REPORT_IDLE_ON);
```

然后是切换IDLE状态的第三步，处理MSG_REPORT_IDLE_ON消息。

```
case MSG_REPORT_IDLE_ON: {
    EventLogTags.writeDeviceIdleOnStart();
    mLocalPowerManager.setDeviceIdleMode(true);
    try {
        /// M: integrate Doze and App Standby @{
        if(null != getDataShapingService()) {
                mDataShapingManager.setDeviceIdleMode(true);
        }
        /// integrate Doze and App Standby @}
        mNetworkPolicyManager.setDeviceIdleMode(true);
        mBatteryStats.noteDeviceIdleMode(true, null, Process.myUid());
    } catch (RemoteException e) {
    }
    getContext().sendBroadcastAsUser(mIdleIntent, UserHandle.ALL);
    EventLogTags.writeDeviceIdleOnComplete();
} break;
```



在消息处理中，google原生的doze方案里直接通过PowerManagerService，NetworkPolicyManager和BatteryStats三个服务的接口设置了doze模式为true，进行了功耗的降低处理。另外发送了ACTION_DEVICE_IDLE_MODE_CHANGED广播进行通知，目前接受该广播的服务有

- WifiServiceImpl
- UsageStatsService
- JobSchedulerService
- SyncManager
- GpsLocationProvider

Doze模式里的功耗相关除了前面提到的挂起Alarm之外，就是在接收到MSG_REPORT_IDLE_ON消息之后的在各个服务中的各自处理了，后续会专门篇章介绍如何实现省电策略的。

从LOCATING状态切换到IDLE状态流程就是如此，从IDLE_MAINTENANCE状态切换到IDLE时，除了三个取消的动作外，其他都一样，不重复说明。

PREVIOUS
吉他谱——赵雷_无法长大 **(/2017/03/16/GUITAR_ZHAOLEI_WUFAZHANGDA/)**

NEXT
ANDROID电源管理之DOZE模式专题系列（八） **(/2017/04/10/PM_DOZE_IDLE_TO_IDLEMAINTANCE/)**

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