Android (/tags/#Android) frameworks (/tags/#frameworks) AlarmManager (/tags/#AlarmManager)

AlarmManagerService之AlarmGroup机制剖析

对比MTK和展讯的Alarm对齐方案

Posted by Cheson on March 6, 2017

1. 概述

在 AlarmManagerService之设置alarm流程 (https://chendongqi.github.io/blog/2017/02/14/SetAlarmFlow/)一文中介绍了设置一个非精确Alarm的流程，其中包括了对Alarm优化方案的介绍。其中包括了两个步骤，其一为Alarm延时，包含了google原生的方案和MTK定制的方案，MTK的方案是封装在jar包中所以之前没有涉及到个中原理的详细介绍；其二为将Alarm分batch触发，根据触发时间和延时时间，将Alarm分批，减少系统被唤醒的次数。本篇中将在此基础上对MTK的Alarm对齐方案和展讯的方案对比，孰优孰劣，相互借鉴。

2. MTK对齐方案

AlarmManagerService之设置alarm流程 (https://chendongqi.github.io/blog/2017/02/14/SetAlarmFlow/)一文中介绍了在计算Alarm的最大可延迟时间时，MTK加入了自己的优化

```
    maxElapsed = mAmPlus.getMaxTriggerTime(type, triggerElapsed, windowLength, interval, opera
```

之前由于没拿到源码只是给出了一个优化后的结果说明，本节中将从源码角度来详述MTK的优化方案。完整源码参见 AlarmManagerService之MTK AmPlus源码 (https://chendongqi.github.io/blog/2017/03/06/MTK_AlarmGroup_code/)。

而AlarmManagerPlus.java中的getMaxTriggerTime方法只是个代理，而这个类中也之有两个方法，另外一个就是构造函数

```
    public long getMaxTriggerTime(int type, long triggerElapsed,
            long windowLength, long interval, PendingIntent operation) {
        return mPowerSavingUtils.getMaxTriggerTime(type, triggerElapsed,
                windowLength, interval, operation);
    }
```

2.1 延迟时间

功能的实现在PowerSavingUtils.java中

```
    public long getMaxTriggerTime(int type, long triggerElapsed,
            long windowLength, long interval, PendingIntent operation) {

        long maxElapsed;
        long nowElapsed = SystemClock.elapsedRealtime();

        if (windowLength == 0L) {// WINDOW_EXACT
            maxElapsed = getMMaxTriggerTime(type, operation, triggerElapsed);
        } else if (windowLength == -1L) {// WINDOW_HEURISTIC
            maxElapsed = adjustMaxTriggerTime(nowElapsed, triggerElapsed,interval, operation,
        } else {
            maxElapsed = triggerElapsed + windowLength;
        }
        return maxElapsed;
    }
```

当windowLength为WINDOW_EXACT时，调用getMMaxTriggerTime来处理延迟时间，当需要做对齐时，最大延迟时间为触发时间加上5分钟。

```
    private long getMMaxTriggerTime(int type, PendingIntent operation,long triggerAtTime) {

        if ((isAlarmNeedAlign(type, operation, true))) {
            return (triggerAtTime + SCREENOFF_TIME_INTERVAL_THRESHOLD);// 300000L
        }
        return triggerAtTime;
    }
```

当windowLength为WINDOW_HEURISTIC时，调用adjustMaxTriggerTime来计算延迟时间。首先沿用了google的计算延迟时间的算法，然后判断需要做对齐时，进一步判断延时是否超过SCREENOFF_TIME_INTERVAL_THRESHOLD，如果没超过则同样将触发时间加上5分钟来作为最大延迟时间处理。

```
    private long adjustMaxTriggerTime(long now, long triggerAtTime,
            long interval, PendingIntent operation, int type,
            boolean isExactAlarm) {

        long futurity = (interval == 0L) ? triggerAtTime - now : interval;

        if (futurity < MIN_FUZZABLE_INTERVAL) {
            futurity = 0L;
        }

        long maxTriggerAtTime = triggerAtTime + (long) (.75 * futurity);// 沿用google的算法

        if ((isAlarmNeedAlign(type, operation, isExactAlarm)) && (maxTriggerAtTime - triggerAt
            return (triggerAtTime + SCREENOFF_TIME_INTERVAL_THRESHOLD);
        }
        return maxTriggerAtTime;
    }
```

如果windowLength已经非0或-1的值，则将triggerElapsed加上windowLength作为最大延迟时间。

2.2 对齐条件

另外一个比较关键的方法就是isAlarmNeedAlign，在计算延迟时间时都用到了

```
    private boolean isAlarmNeedAlign(int type, PendingIntent operation,
            boolean isExactAlarm) {

        if (!isPowerSavingStart()) {// 未达到Power Saving的条件：1、连接usb 2、灭屏时间未超过30s 3、
            return false;
        }

        boolean isAlarmNeedAlign = false;
        outer: if (type == 0 || type == 2) {// 只需要针对wakeup类型的alarm
            String packageName = operation.getCreatorPackage();
            if (packageName == null) {
                Log.v(TAG, "isAlarmNeedAlign : packageName is null");
            } else {
                for (int i = 0; i < mWhitelist.size(); ++i) {// 过滤白名单
                    if ((mWhitelist.get(i)).equals(packageName)) {
                        Log.v(TAG, "isAlarmNeedAlign : packageName = "
                                + packageName + "is in whitelist");
                        return false;
                    }
                }
                if (isExactAlarm) {
                    PackageManager pm = mContext.getPackageManager();
                    try {

                        ApplicationInfo info = pm.getApplicationInfo(packageName, 0);
                        // 过滤系统alarm
                        if (((info.flags & 0x1) != 0) && (packageName.startsWith("com.android"
                            Log.v(TAG, "isAlarmNeedAlign : "+ packageName + " skip!");
                            break outer;
                        }
                    } catch (PackageManager.NameNotFoundException e) {
                        Log.v(TAG, "isAlarmNeedAlign : packageName not fount");
                        break outer;
                    }
                }
                //Log.v(TAG, "isAlarmNeedAlign = true");
                isAlarmNeedAlign = true;
            }
        }
        return isAlarmNeedAlign;
    }
```

判断是否需要对齐的alarm，只针对type为0或者2的，也就是说会唤醒系统的alarm；过滤掉了白名单中的应用设置的alarm，白名单的定义放在framework/base/core/java/com/mediatek/amplus/config/alarmplus.config，目前只添加了CTS相关的应用

```
com.android.app.cts
com.android.cts.stub
```

针对exact的Alarm，过滤掉了系统的alarm，而剩下的Alarm都是可以被对齐的Alarm，按照计算延时的规则进行处理。

3. 展讯对齐方案

展讯的Alarm对齐方案是直接提供了源码的。在AlarmManagerService的setImpl中计算最大延迟时间时展讯并未做修改，在setImplLocked中在分batch之前加入Alarm对齐的方案。

```
    // Here start to align alarm. It just adjust the 3rd-party alarm without any flag
    final PowerGuruAlarmInfo guruAlarm = mAlignHelper.matchBeatListPackage(a.operation);
    if(mAlignHelper.isUnavailableGMS(guruAlarm)){
        return;
    }
    if(mAlignHelper.getEnable() && mAlignHelper.checkAlignEnable(a.type, guruAlarm)){
        final long alignTime = mAlignHelper.adjustTriggerTime(a.when, a.type);
        if(alignTime > SystemClock.elapsedRealtime()){
            a.maxWhenElapsed = alignTime + (a.maxWhenElapsed - a.whenElapsed);
            a.whenElapsed = alignTime;
            a.tiggerTimeAdjusted = true;
        }else{
            Slog.w(TAG, "the adjusted time is in the past, ignore it");
            a.tiggerTimeAdjusted = false;
        }
    }
```

3.1 心跳列表

第一步首先判断Alarm是否在心跳列表中

```
    final PowerGuruAlarmInfo guruAlarm = mAlignHelper.matchBeatListPackage(a.operation);
```

```
    protected PowerGuruAlarmInfo matchBeatListPackage(final PendingIntent pi){
        if(mBeatlist != null && mBeatlist.size() > 0){// mBeatlist有效性判断
            String pn = pi.getCreatorPackage();
            Intent in = pi.getIntent();
            String action = null;
            String component = null;
            if(in != null){
                action = in.getAction();
                if(in.getComponent() != null){
                    component = in.getComponent().getClassName();
                }
            }
            for(PowerGuruAlarmInfo palarm : mBeatlist){// 遍历列表寻找匹配
                if(pn.equals(palarm.packageName) &&
                ((action != null && action.equals(palarm.actionName)) || (action == null && pa
                ((component != null && component.equals(palarm.componentName)) || (component =
                )
                return palarm;
            }
        }
        return null;
    }
```

那么mBeatlist的列表是怎么来的呢？

```
    protected void updateBeatlist(){
        if(mPowerGuruService != null){
            mBeatlist = mPowerGuruService.getBeatList();
        }else{
            Slog.w(TAG, "updateBeatlist() -> PowerGuruService is not running ???");
        }
    }
```

跳转到PowerGuruService中

```
    @Override
    public List<PowerGuruAlarmInfo> getBeatList() {
        List<PowerGuruAlarmInfo> beatList = new ArrayList<PowerGuruAlarmInfo>();
        synchronized( mHeartbeatListLock){
            for (AlarmInfo item : mHeartbeatAlarmList){
                beatList.add(new PowerGuruAlarmInfo(item.packageName, item.action,
                            item.componentName, item.type, item.isFromGms, item.isAvailable));
            }
            mHeartbeatListUpdate = false;
        }
        return beatList;
    }
```

这里代码表述了心跳列表来自于mHeartbeatAlarmList，而mHeartbeatAlarmList的生成有多种途径，在PowerGuruService中，收到一个NEW_ALARM的消息后会触发一系列的处理动作，其中包含了是否将这个Alarm添加的mHeartbeatAlarmList中去，而能添加有多个判断条件，来一个个看下。

首先是过滤掉不处理的，GMS的单独处理，非第三方的直接过滤掉，白名单（来自于动态创建的system/whiteAppList.xml）的过滤掉

```
    /*check if a GMS alarm*/
    if (isGMSAlarm(alarm)) {

        log("Alarm from GMS app");

        if (processGMSAlarm(alarm)) {
            return;
        }
    }

    /*check if a ThirdParty alarm */
    if (!isThirdParty(alarm) && !alarm.isFromGms) {
        log("Alarm is not from ThirdParty, is not care about!");
        return;
    }

    /*check if in a white list*/
    if (isInWhiteAppList(alarm)) {
        log("Alarm is form WhiteApp, is not care about!");
        return;
    }
```

然后是添加到列表中的条件，其一是Alarm匹配预设的列表（来自于动态创建的system/pwrGuruPreset.xml），从代码中来看该文件被创建后也没有写入过内容，目前应该还没有预设的内容

```
    /*check if a preset Heartbeat alarm or a saved heartbeat alarm*/
    if (isPresetHeartbeatAlarm(alarm)) {
        log("This alarm is a preset Heartbeat alarm! add it to Heartbeat list!");

        addToHeartbeatAlarmList(alarm, true, false);
        return;
    }
```

其二是自学习的添加的心跳Alarm，存放在代码中创建的system/pwrGuruStudied.xml文件中。

```
    /*check if a preset Heartbeat alarm or a saved heartbeat alarm*/
    if (isSavedHeartbeatAlarm(alarm)) {
        log("This alarm is a study saved Heartbeat alarm! add it to Heartbeat list!");

        addToHeartbeatAlarmList(alarm, false, true);
        return;
    }
```

其写入的内容来源为目前的mAlarmList

```
writeAlarmInfoToFile(mStudiedHeartBeatRecordFile, mAlarmList);
```

然而最终写入的方法确并未被调用，展讯的代码也是改的逻辑混乱的

```
case SAVE_STUDY_HEARTBEAT_LIST_TO_DISK:
    log("SAVE_STUDY_HEARTBEAT_LIST_TO_DISK");
    //saveStudyHeartbeatListToDisk(); //just do not need to save to disk
    break;
```

其三为检测是否是心跳alarm并将其添加到列表中

```
/*detec if it is a heartbeat alarm */
if (detectHeartbeatAlarm(alarm)) {
    log("This alarm is a Heartbeat alarm! add it to Heartbeat list!");

    addToHeartbeatAlarmList(alarm, true, false);
    return;
}
```

核心在于如何检测

```
private boolean detectHeartbeatAlarm(AlarmInfo alarm) {

    if (alarm.repeatInterval > mAlarmRepeatIntervalMsLevel1) {// 默认15分钟
        log("alarm interval = " + alarm.repeatInterval + ", > " + mAlarmRepeatIntervalMsLe
        return false;
    }

    if (containHeartbeatAlarmFeature(alarm)) {// 包含关键字
        log(" alarm action = " + alarm.action + " contain Heartbeat alarm feature, it is a
        return true;
    }

    if (alarm.repeatInterval > mAlarmRepeatIntervalMsLevel2) {// 默认10分钟

        processSuspectedHeartbeatAlarm(alarm);

        return false;
    }

    return true;
}
```

上面这段代码的要点有三个：1、重复时间超过15分钟的不属于心跳alarm；2、重复时间15分钟以内action中包含预设的关键字的属于心跳alarm，这里的关键字为：

```
private final String[] mHeartbeatFetureStrings = new String[] {
    "KEEP_ALIVE",
    "KEEPALIVE",
    /*"heartbeat",*/
    "PING",
};
```

看关键字的含义应该是IM应用保活用的相关Alarm；3、重复时间大于10分钟小于15分钟的，目前展讯无处理的方案，只是留了个处理的接口，什么都没做就return了，然后又return false，目前判定为非心跳alarm：

```
    private void processSuspectedHeartbeatAlarm(AlarmInfo alarm) {
        log("suspected heartbeat alarm:");
        dumpAlarm(alarm);// 打印alarm的信息
        //currently just return
        return;

    }
```

小结一下判断为心跳Alarm的逻辑，重复时间超过15分钟的直接判断为非心跳Alarm；15分钟以下10分钟以上的符合心跳特性的Alarm作为心跳Alarm；10分钟以下的作为心跳Alarm。

3.2 对齐处理

前面很大的篇幅都是在判断Alarm是否可以作为心跳Alarm，接下来就是处理该Alarm的动作。首先进一步判断了该Alarm是否真的可以作为心跳Alarm：

```
    if(mAlignHelper.isUnavailableGMS(guruAlarm)){// unavailableGMS的条件
        return;
    }
```

如果Alarm来自于GMS包，并且处于中国网络，未连接VPN并且未连接Wifi，则不作为心跳Alarm使用，这个接触过GMS的应该就非常好理解，中国网络环境对goole的不友好也在这里体现了。

接下来就是重点了，处理Alarm的触发时间

```
    if(mAlignHelper.getEnable() && mAlignHelper.checkAlignEnable(a.type, guruAlarm)){// 进入条件
        final long alignTime = mAlignHelper.adjustTriggerTime(a.when, a.type);
        if(alignTime > SystemClock.elapsedRealtime()){
            a.maxWhenElapsed = alignTime + (a.maxWhenElapsed - a.whenElapsed);
            a.whenElapsed = alignTime;
            a.tiggerTimeAdjusted = true;
        }else{
            Slog.w(TAG, "the adjusted time is in the past, ignore it");
            a.tiggerTimeAdjusted = false;
        }
    }
```

处理流程进入的两个条件是：1、系统中设置的开关开启；2、带WAKEUP的Alarm，并且在灭屏情况下且未插入USB或者AC。这里和MTK的判断条件非常相似，MTK多了在灭屏后30秒的条件。

```
    protected boolean getEnable(){
        int temp = SystemProperties.getInt(ALIGN_ENABLE, 1);
        return temp == 1;
    }

    protected boolean checkAlignEnable(int type, final PowerGuruAlarmInfo palarm){
        boolean isWakeup = (type == RTC_WAKEUP || type == ELAPSED_REALTIME_WAKEUP);
        if(isWakeup && (palarm != null) && mAlignEnable){
            Slog.d(TAG, "checkAlignEnable() return true. palarm = "+ palarm);
            return true;
        }
        return false;
    }
```

然后是对对齐时间的处理，其原理是触发时间以预设的的alignLength来对齐

```
    protected long adjustTriggerTime(long triggerTime, int type){
        final int alignLength = getAlignLength();// 默认30分钟
        Slog.d(TAG, "adjustTriggerTime()  triggerTime = "+triggerTime+", type = "+type+", alig
        if(type == ELAPSED_REALTIME_WAKEUP || type == ELAPSED_REALTIME){// realtime类型才会处理
            triggerTime += System.currentTimeMillis() - SystemClock.elapsedRealtime();// 转RTC
        }
        final CharSequence orgiTrigger = DateFormat.format("yyyy-MM-dd HH:mm:ss", triggerTime)

        long adjustedRtcTime = adjustTriggerTimeInternal(triggerTime, alignLength);// 终极处理
        long nowRTC = System.currentTimeMillis();
        if(adjustedRtcTime < nowRTC){// 善后
            Slog.w(TAG, "the triggerTime is adjuested to past, so it need adjusted to future t
            if(isAlignPoint(nowRTC, alignLength)){
                nowRTC += 2000; //make sure nowRTC can be aligned to future time.
            }
            adjustedRtcTime = adjustTriggerTimeInternal(nowRTC, alignLength);
        }

        final long adjuestedMillis = AlarmManagerService.convertToElapsed(adjustedRtcTime, 0);
        final CharSequence adjuestedTrigger = DateFormat.format("yyyy-MM-dd HH:mm:ss", adjuste
        Slog.d(TAG, "adjustTriggerTime() original-trigger: "+orgiTrigger+ ", adjuested-trigger
        return adjuestedMillis;
    }
```

最终的对齐方法如下代码描述，取时间中的分钟和秒钟部分，如果分钟小于alignLength(默认30分钟)，则分钟数设成30分钟，秒钟数设为0；如果分钟数不是alignLength的整数倍，则改成整数倍，秒钟数设为0；如果最终分钟数大于等于60，则给小时加1，分钟归0。

```
    private long adjustTriggerTimeInternal(long rtcTime, int alignLength){
        if(alignLength <=0){
            throw new IllegalStateException("The align length can not be less than 0 !");
        }
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(rtcTime);
        int minute = calendar.get(Calendar.MINUTE);
        int seconds = calendar.get(Calendar.SECOND);
        int residue = minute / alignLength;
        if (residue == 0){// minute is less than alignLength
            if(minute != 0 || seconds != 0){
                calendar.set(Calendar.MINUTE, alignLength);// 设成30分钟后触发
                calendar.set(Calendar.SECOND, 0);// 秒数为0，为了对齐
            }
        }else{
            if(0 != (minute % alignLength) || 0 != seconds){
                int alignMinute = (residue+1) * alignLength;// 不是30分钟的整数倍，格式成整数倍，例如
                if (alignMinute >= 60){//This case means it should align to next hour.
                    calendar.add(Calendar.HOUR_OF_DAY, 1);
                    calendar.set(Calendar.MINUTE, 0);
                }else{
                    calendar.set(Calendar.MINUTE, alignMinute);
                }
                calendar.set(Calendar.SECOND, 0);
            }
        }
        return calendar.getTimeInMillis();
    }
```

4. MTK和展讯的对比

4.1 触发对齐条件

MTK：灭屏，灭屏超过30s，未连接usb

展讯：灭屏，未连接usb和AC

4.2 优化方法

MTK：处理最大延时时间—— 精准Alarm统一延时5分钟；模糊Alarm用google的算法先计算最大延时时间，和5分钟比较取最大值

展讯：处理触发时间—— 以预设的alignLength（默认30分钟）为标尺来格式化Alarm的触发时间。当触发的分钟数小于30时，设为30；分钟数不为30的整数倍时调整成整数倍；分钟数大于等于60时，小时数加1，分钟归0；秒数都设为0。

4.3 优化的Alarm来源

MTK：带WAKEUP属性；非白名单；非系统（com.android）精准Alarm

展讯：待WAKEUP属性；来自于心跳列表：GMS的单独处理，非第三方的直接过滤掉，白名单（来自于动态创建的system/whiteAppList.xml）的过滤掉；剩下的里面做如下判断：重复时间超过15分钟的直接判断为非心跳Alarm；15分钟以下10分钟以上的符合心跳特性的Alarm作为心跳Alarm；10分钟以下的作为心跳Alarm。

4.4 代码对比

MTK的方案做的比较简洁轻量级，展讯的代码写的比较复杂，很多地方又没有具体实现只是保留了一个初衷。两种方案一个是针对延时时间的优化，一个是对触发时间的对齐，所以并不矛盾，可以实现互补。

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

NEXT
ALARMMANAGERSERVICE之MTK AMPLUS源码
(/2017/03/06/MTK_ALARMGROUP_CODE/)

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)