frameworks/base/core/java/android/app/AlarmManager.java

frameworks/base/services/core/java/com/android/server/AlarmManagerService.java

frameworks/base/services/core/jni/com\_android\_server\_AlarmManagerService.cpp

1 AlarmManager的使用

1.1概述

1.1.1 AlarmManager的作用

在特定的时刻为我们广播一个指定的Intent。简单的说就是我们设定一个时间，然后在该时间到来时，AlarmManager为我们广播一个我们设定的Intent。

1.1.2 AlarmManager的分类

public static final int RTC\_WAKEUP = 0;//硬件闹钟，当闹钟发射时唤醒手机休眠

public static final int RTC = 1;//硬件闹钟，不会唤醒手机，当手机休眠时不发射闹钟

public static final int ELAPSED\_REALTIME\_WAKEUP = 2;//真实时间流逝闹钟，具备唤醒功能

public static final int ELAPSED\_REALTIME = 3;//真实时间流逝闹钟，不会唤醒设备；如果设备睡着了，它将关闭，直到下一次设备唤醒时，它才会被发射

static const clockid\_t android\_alarm\_to\_clockid[N\_ANDROID\_TIMERFDS] = {

CLOCK\_REALTIME\_ALARM,//RealTime Alarmtimer，借助RTC，不会因为内核进入cpuidle或者suspend，而停止工作，具备唤醒功能

CLOCK\_REALTIME,//系统实时时间，从Epoch计时，可以被用户更改以及adjtime和NTP影响

CLOCK\_BOOTTIME\_ALARM,//BootTime Alarmtimer，这是healthd所使用的timer clockid，与CLOCK\_MONOTONIC类似，但是当suspend时，会依然增加。

CLOCK\_BOOTTIME,//从系统启动这一刻开始计时，包括休眠时间，受到settimeofday的影响

CLOCK\_MONOTONIC,//是单调时间，即从某个时间点开始到现在过去的时间。用户不能修改这个时间，但是当系统进入休眠（suspend）时，CLOCK\_MONOTONIC是不会增加的

CLOCK\_REALTIME,//系统实时时间，从Epoch计时，可以被用户更改以及adjtime和NTP影响

};

不过看高通的做法是，alarm wakeup功能做在pmic上了

[29843.351214] :(0)[ 1400|system\_server ] suspend ns: 29843351204396\x09suspend cycles: 2503675344600

[29843.351204] :(0)[ 1400|system\_server ] GICv3: gic\_show\_pending\_irqs: 67 triggered pmic controller

[29843.351204] :(0)[ 1400|system\_server ] Suspended for 218.747 seconds

[29843.351204] :(0)[ 1400|system\_server ] resume cycles: 2507875308316

[29843.351275] :(0)[ 1400|system\_server ] Resume caused by IRQ 603, qpnp\_rtc\_alarm

1.1.3 AlarmManager提供的方法

public void set(@AlarmType int type, long triggerAtMillis, PendingIntent operation)//设置一个闹钟

public void setRepeating(@AlarmType int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)//设置一个会重复的闹钟

public void setInexactRepeating(@AlarmType int type, long triggerAtMillis, long intervalMillis, PendingIntent operation)//设置一个重复闹钟的不精确版本，它相对而言更节能一些，因为系统可能会将几个差不多的闹钟合并为一个来执行，减少设备的唤醒次数

内置的几个interval为：

INTERVAL\_FIFTEEN\_MINUTES

INTERVAL\_HALF\_HOUR

INTERVAL\_HOUR

INTERVAL\_HALF\_DAY

INTERVAL\_DAY

如果你将其设为DAY，那么可能这一天中的所有闹钟都会被合并掉。

public void cancel(PendingIntent operation)//取消一个设置的闹钟

public void setTimeZone(String timeZone)//设置系统的默认时区。需要android.permission.SET\_TIME\_ZONE权限

1.2使用方法

1、获取系统服务ALARM\_SERVICE：

AlarmManager alarmMgr = (AlarmManager) getSystemService(ALARM\_SERVICE);

2、闹钟一般都是通过发出一个广播来实现的，定义一个PendingIntent发出广播：

// 创建一个广播事件的意图

Intent intent = new Intent(ALARM\_EVENT);

// 创建一个用于广播的延迟意图

PendingIntent pIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.FLAG\_UPDATE\_CURRENT);

// 从系统服务中获取闹钟管理器

AlarmManager alarmMgr = (AlarmManager) getSystemService(ALARM\_SERVICE);

Calendar calendar = Calendar.getInstance();

calendar.setTimeInMillis(System.currentTimeMillis());

// 给当前时间加上若干秒

calendar.add(Calendar.SECOND, mDelay);

PendingIntent：简单的说就是在Intent上在加个指定的动作。Intent的话，我们还需要在执行startActivity、startService或sendBroadcast才能使Intent有用。而PendingIntent的话就是将这个动作包含在内了，如PendingIntent.getBroadcast就包含了sendBroadcast的动作

3、调用AlarmManager的set方法开启闹钟：

// 开始设定闹钟，延迟若干秒后，携带延迟意图发送闹钟广播

alarmMgr.set(AlarmManager.RTC\_WAKEUP, calendar.getTimeInMillis(), pIntent);

mDesc = DateUtil.getNowTime() + " 设置闹钟";

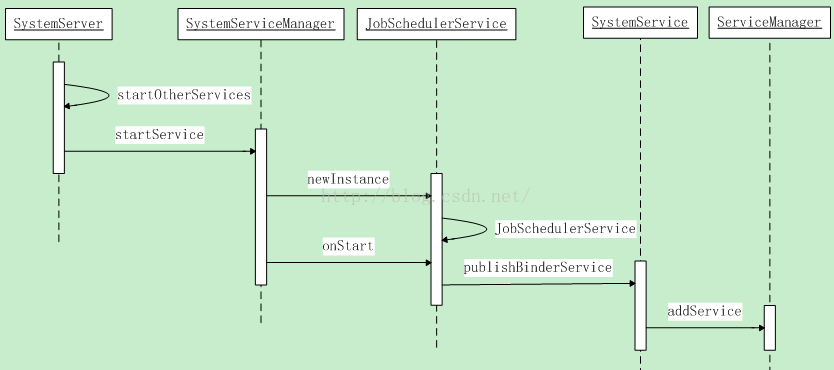
tv\_alarm.setText(mDesc);

2 Android之AlarmManagerService源码分析

AlarmManager实质上是一个全局定时器，是Android常用的一种系统服务级别的提示服务，在指定时间或周期性启动其他组件(Activity、Service、BroadcastReceiver)。

第一章专门介绍了AlarmManager定时器的使用方法，获取到服务后，调用该服务的一些设置方法，在设定时间到达后就会启动指定的组件。

AlarmManagerService和JobSchedulerService一样都是系统服务，故它们的启动流程也类似，先看下时序图：

Zygote进程启动后会启动System进程，在System进程启动过程中会启动系统中的关键服务，如AMS、PMS、JobSchedulerService。

SystemServer启动AlarmManagerService服务调用的是SystemServiceManager类的startService方法：

private void startOtherServices() {

try {

...

mSystemServiceManager.startService(AlarmManagerService.class);

...

} catch (RuntimeException e) {

Slog.e("System", "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Slog.e("System", "\*\*\*\*\*\*\*\*\*\*\*\* Failure starting core service", e);

}

}

在开启AlarmManagerService服务时，会创建服务的实例，看下该服务的创建过程：

frameworks/base/services/core/java/com/android/server/AlarmManagerService.java

final AlarmHandler mHandler = new AlarmHandler();

final Constants mConstants;

public AlarmManagerService(Context context) {

super(context);

// 初始化Handler和常量Constants类

mConstants = new Constants(mHandler);

}

初始化Handler的代码后面调用时再分析，这里先看下常量类的实现

/\*\*

\* 该类中所有的时间单位都是毫秒。

\* 这些常量保持与系统全局设置一致。

\* 任何访问该类或该类中的字段都要持有AlarmManagerService.mLock锁

\*/

private final class Constants extends ContentObserver {

// Key names stored in the settings value. 在设置中保存的键值

private static final String KEY\_MIN\_FUTURITY = "min\_futurity";

private static final String KEY\_MIN\_INTERVAL = "min\_interval";

private static final String KEY\_ALLOW\_WHILE\_IDLE\_SHORT\_TIME = "allow\_while\_idle\_short\_time";

private static final String KEY\_ALLOW\_WHILE\_IDLE\_LONG\_TIME = "allow\_while\_idle\_long\_time";

private static final String KEY\_ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION

= "allow\_while\_idle\_whitelist\_duration";

private static final String KEY\_LISTENER\_TIMEOUT = "listener\_timeout";

private static final long DEFAULT\_MIN\_FUTURITY = 5 \* 1000;

private static final long DEFAULT\_MIN\_INTERVAL = 60 \* 1000;

private static final long DEFAULT\_ALLOW\_WHILE\_IDLE\_SHORT\_TIME = DEFAULT\_MIN\_FUTURITY;

private static final long DEFAULT\_ALLOW\_WHILE\_IDLE\_LONG\_TIME = 9\*60\*1000;

private static final long DEFAULT\_ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION = 10\*1000;

private static final long DEFAULT\_LISTENER\_TIMEOUT = 5 \* 1000;

// Minimum futurity of a new alarm，新报警器的最小将来数

public long MIN\_FUTURITY = DEFAULT\_MIN\_FUTURITY;

// Minimum alarm recurrence interval，最小报警重复间隔

public long MIN\_INTERVAL = DEFAULT\_MIN\_INTERVAL;

// Minimum time between ALLOW\_WHILE\_IDLE alarms when system is not idle.

public long ALLOW\_WHILE\_IDLE\_SHORT\_TIME = DEFAULT\_ALLOW\_WHILE\_IDLE\_SHORT\_TIME;

// Minimum time between ALLOW\_WHILE\_IDLE alarms when system is idling.

public long ALLOW\_WHILE\_IDLE\_LONG\_TIME = DEFAULT\_ALLOW\_WHILE\_IDLE\_LONG\_TIME;

// BroadcastOptions.setTemporaryAppWhitelistDuration() to use for FLAG\_ALLOW\_WHILE\_IDLE.

public long ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION = DEFAULT\_ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION;

// Direct alarm listener callback timeout

public long LISTENER\_TIMEOUT = DEFAULT\_LISTENER\_TIMEOUT;

private ContentResolver mResolver;

private final KeyValueListParser mParser = new KeyValueListParser(',');

private long mLastAllowWhileIdleWhitelistDuration = -1;

public Constants(Handler handler) {

super(handler);

// 更新可以开始执行flag为ALLOW\_WHILE\_IDLE的alarm的最小时间间隔

updateAllowWhileIdleMinTimeLocked();

updateAllowWhileIdleWhitelistDurationLocked();

}

// 系统启动后会调用该方法，注册数据库监听

public void start(ContentResolver resolver) {

mResolver = resolver;

// 监听数据库变化

mResolver.registerContentObserver(Settings.Global.getUriFor(Settings.Global.ALARM\_MANAGER\_CONSTANTS), false, this);

updateConstants();

}

// 更新可以开始执行flag为ALLOW\_WHILE\_IDLE的alarm的最小时间间隔

public void updateAllowWhileIdleMinTimeLocked() {

mAllowWhileIdleMinTime = mPendingIdleUntil != null

? ALLOW\_WHILE\_IDLE\_LONG\_TIME : ALLOW\_WHILE\_IDLE\_SHORT\_TIME;

}

public void updateAllowWhileIdleWhitelistDurationLocked() {

if (mLastAllowWhileIdleWhitelistDuration != ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION) {

mLastAllowWhileIdleWhitelistDuration = ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION;

BroadcastOptions opts = BroadcastOptions.makeBasic();

opts.setTemporaryAppWhitelistDuration(ALLOW\_WHILE\_IDLE\_WHITELIST\_DURATION);

mIdleOptions = opts.toBundle();

}

}

@Override

public void onChange(boolean selfChange, Uri uri) {

// 数据库内容变化时，更新一些本地变量

updateConstants();

}

......

}