4.8. Отложенное выполнение действий по расписанию

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 4.8. Отложенное выполнение действий по расписанию

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:56

Оглавление

1. Управление отложенными действиями

Любой процесс, даже работающий в фоновом режиме, неминуемо будет расходовать заряд батареи устройства: таймеры, напоминания, сигнальные службы - сервисы, работающие вне жизненного цикла приложения, при завершении работы самого приложения остаются в запущенном состоянии. Учитывая портативность смартфона, при отсутствии бесперебойного источника питания остро стоит вопрос экономии заряда батареи. Именно по этому по истечении определённого времени "простоя" устройства оно сначала "гасит" экран, а затем засыпает. При этом активные процессы тоже переходят в режим сна и перестают выполняться. В ракурсе экономии энергии устройства, режим сна весьма полезен. Однако, при "засыпании" устройства так же останавливаются и фоновые процессы, как правило, выполняющие тяжеловесные задачи (например, загрузку и установку обновлений, подготовку данных к отправке и отправка данных по HTTP, обращение к базе данных и т.п.) Остановка этих процессов влечёт за собой потерю данных, а, возможно, и сбои в работе приложений. Частично данную проблему можно решить за счёт использования механизма запуска процессов по расписанию с использованием PendingIntent - класса ожидающих намерений. PendingIntent получает намерение и выполняет его по требованию, то есть является «управляющим» механизмом для запуска, регламентирования работы и остановки процессов. При наличии такой регулировки приложения и сервисы запускаются по мере необходимости и не тратят заряд батареи на поддержку своей жизнедеятельности в фоновом режиме. В Android имеется несколько вариантов планировщиков, позволяющих организовать расписание для запуска действий.

2. Класс AlarmManager

Механизм сигнализации в Android реализован в классе AlarmManager. Разработчики в документации подчёркивают назначение AlarmManager: запуск процессов по расписанию вне зависимости от состояния активности приложения. То есть данная сигнализация

- активирует приложение, если оно было неактивным;
- может при запуске выводить устройство из режима сна, если устройство было неактивно;
- срабатывает сразу после включения устройства, если в назначенное время устройство было выключено.

Однако, с версии API 19 (Android 4.4.4 KitKat) была произведена оптимизация объекта AlarmManager в пользу экономии ресурса батареи за счёт управления пробуждениями устройства. Появилась возможность явного указания сигнализации на необходимость выводить устройство из режима сна. Без этого указания в режиме сна будильник срабатывать не будет, но сработает сразу же при переходе устройства в активное состояние.

Преимущества сигнализации::

- возможность выполнения ожидающего намерения в определённый срок;
- возможность использования AlarmManager в широковещательной рассылке для запуска сторонних процессов;
- возможность управления другими процессами за рамками жизненного цикла текущего приложения или в режиме сна устройства;
- возможность избежать использования ресурсоёмких служб и таймеров.

> AlarmManager целесообразно использовать для управления действиями за пределами работы текущего приложения.
>
> При активном приложении разработчиками рекомендована организация фоновых потоков с использованием Handler.
>
> Не рекомендуется использовать сигнализацию для организации сетевого взаимодействия. Для этих целей лучше использовать Firebase Cloud Messaging
>
> Альтернативой сигнализации, выводящей устройство из режима сна, является механизм WorkManager

Точкой отсчёта времени срабатывания сигнализации может выступать как момент включения устройства и загрузки операционной системы, так и системное время, установленное на устройстве. Первый вариант более предпочтителен, поскольку не зависит от региональных настроек и не выполняет частых действий по проверке системного времени.

При использовании сигнализации следует придерживаться определённых правил:

- желательно настраивать будильник на неопределённое время. В таком случае система будет запускать один будильник для всех задач, запланированных на ближайший диапазон времени. В противном случае на каждую задачу будет запущен отдельный будильник, что приведёт к нерациональному расходу заряда батареи;
- действия над данными при сетевом взаимодействии выполнять в локальных процессах вне будильника и после его срабатывания;
- не стоит будить устройство без необходимости, лучше использовать отложенную сигнализацию;
- свести к минимуму использование времени часового пояса, по возможности время сигнализации настраивать в режиме отсчёта от момента загрузки устройства.

Объект AlarmManager может быть запущен в трёх режимах:

- одноразовая сигнализация set(Int, Long, PendingIntent!), которая срабатывает ровно один раз по прошествии времени от точки отсчёта и отключается;
- с точным повторением `setRepeating(Int, Long, Long, PendingIntent!), которая срабатывает по прошествии заданного времени и потом повторяется через равные промежутки времени (этот будильник самостоятельно не отключается);
- с приблизительным повторением setInexactRepeating(Int, Long, Long, PendingIntent!), который срабатывает ориентировочно в установленное время от точки отсчёта и повторяется раз в установленный промежуток времени (один раз в день, один раз в час и т.п.)

Из повторяющихся режимов `setInexactRepeating()` более предпочтителен с точки зрения экономии энергии, поскольку в этом режиме сигнализация подстраивается под общее поведение системы и может быть запущена одна на несколько задач, запланированных примерно на одно время.

С API 23 в Android добавлена новая схема управления питанием - режим ожидания с пониженным энергопотреблением (Doze), отключающий все, в том числе и фоновые, пользовательские процессы. Отключение касается и сигнализаций, то есть они работать не будут. В случае необходимости вывода устройства из Doze - состояния AlarmManager запускают методами `setAndAllowWhileIdle()` или

`setExactAndAllowWhileIdle()`. Подробнее об этом статья в документации разработчиков.

Для отмены установленной сигнализации применяется метод cancel(PendingIntent!). Одноразовая сигнализация, как правило, в отмене не нуждается.

Итак, создание сигнализации требует выполнения следующих шагов:

1. Создание класса запускаемого процесса.

```
class ExecProcess: [Activity() | BroadcastReceiver() | Service()] {
...
}
```

Запускаемым процессом может быть

- **активность в текущем приложении**, запуск которой должен быть отложен на какое-то время;
- **приёмник широковещательных сообщений**, предусматривающий выход за пределы контекста запускающего приложения;
- **сервис**, так же работающий за пределами приложения.

2. Создание намерения по запуску процесса

```
val intent = Intent(context, ExecProcess::class.java)
```

3. Создание ожидающего намерения, запускающего подготовленное намерение по расписанию

```
val pIntent = PendingIntent.get[Activity | Broadcast | Service] (context, requestCode, intent, flags)
```

4. Подготовка объекта AlarmManager для принятия ожидающего намерения

```
val alarmManager = getSystemService(Context.ALARM_SERVICE) as AlarmManager
```

5. Установка ожидающего намерения на подготовленный объект сигнализации

```
alarmManager.set[Repeating](typeAlarm, triggerTime, [interval,] pIntent)
```

2.1. Установка сигнализации

Поскольку сигнализация является отложенным процессом, то и запускается она по отложенному намерению PendingIntent. Данный класс является "упаковкой" для обычного намерения Intent и представляет из себя ссылку на данные и действие с ними, размещенные в передаваемом намерении. Таким образом, отложенное намерение выполняет базовое намерение по определённому правилу. PendingIntent имеет смысл применять в случае выхода расписания действий за пределы времени жизни запускающего его приложения, что вполне соответствует режиму работы сигнализации AlarmManager.

Для установки сигнализации, не зависимо от её режима, необходимо определить:

- *тип сигнализации.*

В классе AlarmManager существует четыре типа сигнализаций, использующих разные точки отсчёта времени и имеющие (не имеющие) право выводить устройство из спящего режима.

тип сигнализации	назначение сигнализации	использование
ELAPSED_REALTIME	отсчёт времени начинается от момента загрузки устройства. Если на момент запланированного времени устройство активно, то сигнализация сработает. При нахождении устройства в режиме сна намерение выполнится сразу же после активации устройства.	AlarmManager.ELAPSED_REALTIME
ELAPSED_REALTIME_WAKEUP	отсчёт времени начинается с момента загрузки устройства, но при наступлении времени выполнения намерения устройство выводится из режима сна.	AlarmManager.ELAPSED_REALTIME_WAKEUP
RTC	время проверяется по установленному на устройстве времени. Применяется только в случае необходимости привязки работы приложений к текущему времени. Чувствителен к смене часового пояса в настройках устройства, что влечёт нестабильность работы приложения. Не выводит устройства из режима сна, срабатывает только при активном состоянии устройства. Если в назначенное время устройство было неактивно, то сработает при ближайшем его переходе в активное состояние.	AlarmManager.RTC
RTC_WAKEUP	Время срабатывания рассчитывается от установленного на устройстве времени. Если устройство на момент срабатывания сигнализации находилось в режиме сна, то оно будет активировано и ожидающее намерение сразу же запустит действие.	AlarmManager.RTC_WAKEUP

- *время срабатывания сигнала.*

Устанавливается в зависимости от типа сигнализации. Если тип `RTC`, то используется временная шкала по системному времени, если тип `ELAPSED_REALTIME`, то время устанавливается от момента перезапуска устройства.

Для точных сигналов, работающих по системному времени можно устанавливать время с помощью объекта `Calendar` (используется в создании напоминаний и будильников)

```
val calendar = Calendar.getInstance()
            calendar.timeInMillis = System.currentTimeMillis()
            calendar.add(Calendar.HOUR_OF_DAY,14)
            calendar.set(Calendar.MINUTE,50)
```

Временной параметр определяется методами класса SystemClock. Если время запуска установлено на прошедшее время, то сигнализация сработает сразу же.

метод определения времени	получаемые значения

System.currentTimeMillis()	определяет текущее время в миллисекундах, установленное в настройках устройства. При смене часовых поясов будет работать нестабильно. Рекомендуется использовать при программировании будильников и напоминаний дат в календаре. Для отслеживания изменения настроек времени и даты используют широковещательные сообщения ACTION_TIME_TICK, ACTION_TIME_CHANGED и ACTION_TIMEZONE_CHANGED
uptimeMillis	отсчёт времени в миллисекундах начинается с момента загрузки устройства. При переходе устройства в спящий режим счётчик времени останавливается. При выключении устройства время сбрасывается на 0. Не зависит от текущих настроек локали устройства.
elapsedRealtime elapsedRealtimeNanos	отсчёт времени с момента включения устройства с учётом времени нахождения в спящем режиме. Часы не останавливаются, но при перезагрузке устройства счётчик обнуляется. Не зависит от системных настроек даты и времени на устройстве.

- *временной интервал между срабатываниями.*

Первое срабатывание сигнализации происходит по прошествии времени, переданного во втором аргументе (Long-количество миллисекунд, прошедших от точки отсчёта) методов установки `set()`, `setRepeating()`, `setInexactRepeating()`.

Временной интервал повторных сигналов так же задаётся в миллисекундах и передаётся в методах `setRepeating()`, `setInexactRepeating()` в третьем аргументе. У метода `set()` по понятным причинам данный аргумент отсутствует.

Для неточных сигнализаций (`setInexactRepeating()`) время повтора задаётся константами класса AlarmManager:

раз в день INTERVAL_DAY

раз в 15 минут INTERVAL_FIFTEEN_MINUTES

раз в 12 часов INTERVAL_HALF_DAY

раз в 30 минут INTERVAL_HALF_HOUR

раз в час INTERVAL_HOUR

Примеры установки сигнализации

Для сигнализации

```
val alarmManager = getSystemService(Context.ALARM_SERVICE) as AlarmManager
```

работающей по ожидающему намерению `pIntent`

alarmManager.set(AlarmManager.ELAPSED_REALTIME,pIntent)	одноразовая сигнализация; время запуска не зависит от системного времени; срабатывает сразу, если устройство активно; в режиме сна не запускается, ожидая активации устройства
alarmManager.set(AlarmManager.RTC_WAKEUP, calendar.timeInMillis, pIntent)	одноразовая сигнализация; время срабатывания настроено в объекте Calendar; если к моменту запуска устройство неактивно, то выводит его из режима сна
alarmManager.setRepeating(AlarmManager.RTC, System.currentTimeMillis(), 5000, pIntent)	многоразовая сигнализация; ориентируется по системному времени устройства; первое срабатывание происходит при запуске приложения, остальные точно через каждые 5 секунд; в режиме сна ожидает активации устройства и только после этого срабатывает
alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME, SystemClock.elapsedRealtime()+1500*60, AlarmManager.INTERVAL_FIFTEEN_MINUTES, pIntent)	неточная многократная сигнализация; первое выполнение происходит примерно через полторы минуты после старта приложения, затем повторяется с частотой один раз в 15 минут

Все сигнализации при перезагрузке устройства автоматически отключаются. Восстановление будильника можно регулировать с помощью установки флагов в методах запуска ожидающего намерения.

- *отложенное намерение.*

Последним аргументом методов установки сигнализации является ожидающее намерение. Для запуска у класса PendingIntent предусмотрены методы - аналоги методов запуска класса Context.

Context	PendingIntent
startActivity()	getActivity(Context!, Int, Intent!, Int)
sendBroadcast()	getBroadcast(Context!, Int, Intent!, Int)
startService()	getService(Context!, Int, Intent!, Int)

Методы запуска получают следующие аргументы:

context: Context!	пространство, в котором будет запущено намерение
requestCode: Int	идентификатор запускающего приложения
intent: Intent!	намерение, которое будет запущено в текущей отсылке
flags: Int	целочисленное значение для идентификации режима отправляемого намерения.

Значением флагов может быть либо 0, либо любая константа, поддерживаемая Intent.FillIn(), в частности android.app.PendingIntent.FLAG_ONE_SHOT, android.app.PendingIntent.FLAG_NO_CREATE, android.app.PendingIntent.FLAG_CANCEL_CURRENT, android.app.PendingIntent.FLAG_UPDATE_CURRENT, android.app.PendingIntent.FLAG_IMMUTABLE, android.app.PendingIntent.FLAG_MUTABLE.

В случае использования флага `FLAG_NO_CREATE` метод отправки может вернуть `null`

Два PendingIntent, заданные на одном и том же Intent, считаются одинаковыми и сохраняется только одно отложенное намерение. Для того, чтобы система могла отличать эти намерения, им необходимо назначать различающиеся значения флагов и различные коды requestCode.

Два различных намерения, отправляемые в одно ожидающее намерение, будут помещены в него по очереди. Если при этом первое намерение не успеет отработать, то оно заменится на второе и уже запущено не будет.

2.2. Упражнение 4.8.1 Создание сигнализаций

Создадим приложение AlarmTests с разными видами сигнализаций.

Все активности, широковещательные приёмники и сервисы обязательно должны быть зарегистрированы в файле Manifest.xml!

В новом проекте установите линейную разметку экрана и зафиксируйте горизонтальную ориентацию активности.

Задание 1. Реализовать отложенный запуск активности.

1. Создайте вторую активность с именем ArtActivity.kt без управляющих элементов. Установите в разметке фоновую картинку. Можно по желанию добавить другие элементы. **Зарегистрируйте активность в манифесте**

2. В главной активности опишите разметку кнопки для отложенного запуска второй активности и задайте ей обработчик с именем "oneAlarm"

```
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="oneAlarm"
        android:text="@string/one_alarm"
        android:textSize="25sp"/>
```

3. При нажатии на кнопку запрограммируйте отложенный на 15 секунд запуск второй активности. Для этого:

   а) создайте объект ожидающего намерения на основании намерения по запуску активности

```
val penIntent = PendingIntent.getActivity(applicationContext,1,
        Intent(this,ArtActivity::class.java),PendingIntent.FLAG_UPDATE_CURRENT)
```

б) создайте экземпляр сигнализации

```
val am = getSystemService(Context.ALARM_SERVICE) as AlarmManager
```

и установите на него одноразовый запуск без привязки к системному времени с задержкой в 15000 миллисекунд.

```
am.set(AlarmManager.ELAPSED_REALTIME,SystemClock.elapsedRealtime()+15000,penIntent)
```

в) оповестите пользователей о времени ожидания запуска активности

```
Toast.makeText(applicationContext,"Терпение! Через 15 секунд всё будет!",Toast.LENGTH_LONG).show()
```

Задание 2. Добавьте в приложение будильник. Установку времени реализуйте через диалог `TimePicker`. Предусмотрите повторение сигнала и отключение будильника.

1. Добавьте в разметку главной активности блок для будильника, состоящий из текстовой надписи "БУДИЛЬНИК" и двух кнопок с обработчиками: "Установить" и "Отключить".

```
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="БУДИЛЬНИК"
        android:textSize="25sp" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="repeatAlarm"
            android:text="@string/repeat_alarm" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="clearRep"
            android:text="@string/cancel_rep"/>
    </LinearLayout>
```

2. Поскольку настройка будильника реализуется через диалог выбора времени, создайте класс описания выбора времени с объектом календаря

```
class TimePickerFragment: DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val calendar = Calendar.getInstance()
        var h = calendar.get(Calendar.HOUR_OF_DAY)
        var m = calendar.get(Calendar.MINUTE)
        return TimePickerDialog(activity,activity as TimePickerDialog.OnTimeSetListener,h,m,true)
    }
}
```

3. Будильник будет привязан к системному времени, по этому его запуск реализуйте через приёмник широковещательного сообщения. Зарегистрируйте приёмник в манифесте

```
class AlarmReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {

    }
}
```

Чтобы будильник был настоящим будильником, настройте ему сигнал

```
val alert = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM)
```

и создайте объект `MediaPlayer` для проигрывания сигнала

```
        val mp = MediaPlayer.create(context, alert)
        if (mp != null) {
            mp.setVolume(100f, 100f)
            mp.start()
            mp.setOnCompletionListener { mp -> mp.release() }
        }
```

4. В обработчике кнопки "Установить" создайте и выведите на экран окно диалога

```
val timePicker = TimePickerFragment()
    timePicker.show(supportFragmentManager,"picker")
```

5. Для получения значений из диалога имплементируйте в MainActivity слушателя `TimePickerDialog.OnTimeSetListener` и реализуйте в методе `onTimeSet()` считывание установленного времени в объект Calendar

```
val calendar = Calendar.getInstance()
        calendar.set(Calendar.HOUR_OF_DAY,hourOfDay)
        calendar.set(Calendar.MINUTE,minute)
```

Здесь же создайте намерение относительно широковещательного приёмника

```
val mIntent = Intent(this, AlarmReceiver::class.java)
```

и передайте его в ожидающее намерение `penIntent` в метод `getBroadcast()`. Создайте объект сигнализации (описано в выполнении Задания 1) и установите на него повторяющийся сигнал с привязкой к системному времени и повторением каждую секунду

```
am.setRepeating(AlarmManager.RTC_WAKEUP, calendar.timeInMillis,1000, penIntent)
```

5. Для остановки сигнала будильника определите обработчик для кнопки "Отключить".

   а) создайте намерение относительно `AlarmReceiver`;

   б) создайте ожидающее намерение по запуску Broadcast с теми же значениями аргументов, что и при установке будильника (**!**);

   в) создайте экземпляр сигнализации `AlarmManager`

   г) отмените у сигнализации все установленные сигналы

```
if (penIntent != null && am != null) am.cancel(penIntent)
```

Задание 3. Используйте неточную сигнализацию, выполняющую отправку уведомлений каждые 15 минут.

1. Добавьте в разметку блок из текстового поля с надписью "НАПОМИНАНИЯ" и двумя кнопками с обработчиками "Уведомление" и "Достаточно, я помню".

2. Для отправки уведомлений напишите класс широковещательного приёмника. Зарегистрируйте его в манифесте

```
class AlarmReceiverNotify : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
val CID = "TestNotify"
        val penIntent = PendingIntent.getActivity(context,1,
            Intent(context,ArtActivity::class.java), PendingIntent.FLAG_UPDATE_CURRENT)
        val builder = NotificationCompat.Builder(context,CID)
            .setSmallIcon(android.R.drawable.ic_dialog_email)
            .setContentTitle("Вам письмо")
            .setAutoCancel(true)
            .setContentText("Примите самые искренние пожелания добра и удачи!")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(penIntent)
            .build()
        with(NotificationManagerCompat.from(context)) {
            notify(102, builder)}
   }
}
```

3. Чтобы уведомления приходили на Android Oreo и выше, создайте [канал уведомлений](#).

   Канал создаётся один раз на все уведомления. Не имеет смысла создавать свой канал на каждое отдельное уведомление, поскольку при создании нового канала предыдущий канал заменяется на новый.

   Без создания канала уведомления в версиях API 26 и выше приходить не будут

   Поскольку создание канала не имеет логического отношения к созданию сигнализации, целесообразно выделить его в отдельный метод и вызвать в обработчике кнопки "Уведомление". Можно сам процесс создания канала разместить непосредственно в обработчике.

```
val CID = "TestNotify"
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    val name = "name"
    val descriptionText = "discription"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel(CID, name, importance)
       channel.description = descriptionText
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
}
```

4. В обработчике кнопки "Уведомление" реализуйте установку и запуск неточной сигнализации:

   а) создайте канал уведомлений;

   б) создайте намерение относительно широковещательного приёмника, описанного в п. 2;

   в) создайте ожидающее намерение и задайте ему задачу запуска Broadcast по созданному ранее намерению;

   г) создайте объект сигнализации;

   д) установите на сигнализацию неточные повторяющиеся сигналы, не привязанные к системному времени с повторением каждые 15 минут

```
am.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,
                           SystemClock.uptimeMillis(),
                           AlarmManager.INTERVAL_FIFTEEN_MINUTES,
                           penIntent)
```

5. Определите метод отмены сигнализации в обработчике кнопки "Достаточно, я помню".