Alarmas

Créditos

- Tutorial https://developer.android.com/training/scheduling/alarms.html[1]

[1] Content is licensed under Creative Commons Attribution 2.5

Alarmas

- Clase *AlarmManager*.
- Permiten ejecutar operaciones por tiempo fuera del ciclo de vida de la aplicación.

Características

- Permite lanzar intentos en un tiempo determinado o a intervalos.
- Se puede usar junto con broadcast receivers para comenzar servicios y ejecutar otras operaciones.
- Puede lanzar eventos incluso cuando la aplicación no esté corriendo o aun si el dispositivo está dormido.
- Optimiza recursos. No requiere timers ni servicios corriendo en background.

Nota

- Para operaciones temporizadas que siempre ocurren durante el ciclo de vida de la aplicación, es mejor usar las clases *Handler*, *Timer* y *Thread*.
- De esta forma Android tiene mejor control sobre los recursos.
- Las alarmas repetidas pueden vaciar la pila del dispositivo (ver https://developer.android.com/training/scheduling/alarms.html#tradeoffs)

Clase *Handler*

- Para ejecutar un código dentro de 5 segundos:

```
Handler handler = new Handler();
handler.postDelayed(new Runnable() {
    @Override
    public void run()
    {
        // Corre en el hilo principal
    }
}, 5000);
```

Clases *Timer* y *TimerTask*

- Para ejecutar un código dentro de 5 segundos:

```
TimerTask task = new TimerTask() {
    public void run()
    {
        // Corre en otro hilo
    }};

Timer timer = new Timer("Nombre opcional");
timer.schedule(task, 5000);
```

Otras opciones para *schedule*

void	`schedule(TimerTask task, long delay, long period)` Schedules the specified task for repeated *fixed-delay execution*, beginning after the specified delay.
void	`schedule(TimerTask task, Date time)` Schedules the specified task for execution at the specified time.
void	`schedule(TimerTask task, Date firstTime, long period)` Schedules the specified task for repeated *fixed-delay execution*, beginning at the specified time.
void	`schedule(TimerTask task, long delay)` Schedules the specified task for execution after the specified delay.
void	`scheduleAtFixedRate(TimerTask task, long delay, long period)` Schedules the specified task for repeated *fixed-rate execution*, beginning after the specified delay.
void	`scheduleAtFixedRate(TimerTask task, Date firstTime, long period)` Schedules the specified task for repeated *fixed-rate execution*, beginning at the specified time.

Fixed-delay execution

- In fixed-delay execution, each execution is scheduled relative to the actual execution time of the previous execution. If an execution is delayed for any reason (such as garbage collection or other background activity), subsequent executions will be delayed as well. In the long run, the frequency of execution will generally be slightly lower than the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).

- Fixed-delay execution is appropriate for recurring activities that require "smoothness." In other words, it is appropriate for activities where it is more important to keep the frequency accurate in the short run than in the long run. This includes most animation tasks, such as blinking a cursor at regular intervals. It also includes tasks wherein regular activity is performed in response to human input, such as automatically repeating a character as long as a key is held down.

Fixed-rate execution

- In fixed-rate execution, each execution is scheduled relative to the scheduled execution time of the initial execution. If an execution is delayed for any reason (such as garbage collection or other background activity), two or more executions will occur in rapid succession to "catch up." In the long run, the frequency of execution will be exactly the reciprocal of the specified period (assuming the system clock underlying Object.wait(long) is accurate).

- Fixed-rate execution is appropriate for recurring activities that are sensitive to absolute time, such as ringing a chime every hour on the hour, or running scheduled maintenance every day at a particular time. It is also appropriate for recurring activities where the total time to perform a fixed number of executions is important, such as a countdown timer that ticks once every second for ten seconds. Finally, fixed-rate execution is appropriate for scheduling multiple repeating timer tasks that must remain synchronized with respect to one another.

Alarmas repetidas

1. Escoger el tipo de alarma.
2. Hora de inicio. Si la hora ya pasó la alarma comienza inmediatamente.
3. Intervalo. Por ejemplo, cada hora.
4. Un *PendingIntent* para ejecutar cada vez que la alarma se dispare.

Tipos de alarma

- ELAPSED_REALTIME. El intento se lanza de acuerdo al tiempo que ha pasado desde que el dispositivo fue booteado. No despierta al dispositivo.
- ELAPSED_REALTIME_WAKEUP. Como el previo pero si despierta al dispositivo.
- RTC. Lanza el intento a la hora especificada. No despierta al dispositivo.
- RTC_WAKEUP. Como el previo pero si despierta al dispositivo.

Ejemplo de ELAPSED_REALTIME_WAKEUP

- Despierta al dispositivo en 1 minuto sin repetición.

```
private AlarmManager alarmMgr;
private PendingIntent alarmIntent;
...
alarmMgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context, AlarmReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);

alarmMgr.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
        SystemClock.elapsedRealtime() +
        60 * 1000, alarmIntent);
```

Ejemplo de ELAPSED_REALTIME_WAKEUP

- Despierta al dispositivo en 30 minutos y cada 30 minutos después de eso.

```
// Hopefully your alarm will have a lower frequency than this!
alarmMgr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
        AlarmManager.INTERVAL_HALF_HOUR,
        AlarmManager.INTERVAL_HALF_HOUR, alarmIntent);
```

Ejemplo de RTC

- Despierta todos los días el dispositivo *aproximadamente* a las 2 P.M.

```
// Set the alarm to start at approximately 2:00 p.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 14);

// With setInexactRepeating(), you have to use one of the AlarmManager interval
// constants--in this case, AlarmManager.INTERVAL_DAY.
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
        AlarmManager.INTERVAL_DAY, alarmIntent);
```

Ejemplo de RTC

- Despierta el dispositivo *exactamente* a las 8:30 A.M. y cada 20 minutos a partir de ahí.

```
private AlarmManager alarmMgr;
private PendingIntent alarmIntent;
...
alarmMgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context, AlarmReceiver.class);
alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);

// Set the alarm to start at 8:30 a.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 8);
calendar.set(Calendar.MINUTE, 30);

// setRepeating() lets you specify a precise custom interval--in this case,
// 20 minutes.
alarmMgr.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
        1000 * 60 * 20, alarmIntent);
```

Tiempo inexacto vs exacto

- *setInexactRepeating*() es preferible a *setRepeating*().
- Android puede sincronizar varias alarmas y lanzarlas al mismo tiempo.
- Esto disminuye el gasto de batería.

Cancelar una alarma

```
// If the alarm has been set, cancel it.
if (alarmMgr!= null) {
    alarmMgr.cancel(alarmIntent);
}
```

Alarmas al prender el dispositivo

- Por default las alarmas se cancelan al apagar el dispositivo.
- Para evitar esto, se puede diseñar la aplicación para calendarizar la alarma al prender el dispositivo.

Pasos

1. Poner el permiso RECEIVE_BOOT_COMPLETED en el manifiesto.

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

2. Implementar un *BroadcastReceiver* para recibir el broadcast.

```
public class SampleBootReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {
            // Set the alarm here.
        }
    }
}
```

Pasos

3. Agregar el receptor al manifiesto.

```
<receiver android:name=".SampleBootReceiver"
        android:enabled="false">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"></action>
    </intent-filter>
</receiver>
```

Notar que el receptor está deshabilitado con *android:enabled="false*“.

Pasos

4. Habilitar el receptor desde el programa.

```
ComponentName receiver = new ComponentName(context, SampleBootReceiver.class);
PackageManager pm = context.getPackageManager();

pm.setComponentEnabledSetting(receiver,
        PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
        PackageManager.DONT_KILL_APP);
```

Deshabilitar el receptor

- Si el usuario cancela la alarma.

```
ComponentName receiver = new ComponentName(context, SampleBootReceiver.class);
PackageManager pm = context.getPackageManager();

pm.setComponentEnabledSetting(receiver,
        PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
        PackageManager.DONT_KILL_APP);
```