Agendando tarefas no Android

06/02/2015

Em vários tipos de aplicativos, necessitamos que alguma ação seja agendada para ser executada algum tempo depois, ou que seja feita periodicamente fora do ciclo de vida de sua aplicação, para isso o sistema Android disponibiliza a o recurso de *Alarm* através da classe **AlarmManager** que utiliza melhor os recursos do sistema do que se tivessemos controlando através <threads< em="" style="box-sizing: border-box; margin: 0px; padding: 0px; font-family: Ubuntu, sans-serif; text-decoration: none !important;">ou *Timer*.</threads<>

Começamos com um exemplo que executa uma única fez.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | AlarmManager alarmMgr = (AlarmManager) this.getSystemService(Context.ALARM\_SERVICE);    Intent intent = new Intent(this, AlarmReceiver.class);  PendingIntent alarmIntent = PendingIntent.getBroadcast(this, 0, intent, 0);    alarmMgr.set(AlarmManager.ELAPSED\_REALTIME\_WAKEUP,  SystemClock.elapsedRealtime() + 60 \* 1000, alarmIntent); |

Começamos na linha 1 pegando um objeto *AlarmManager* do sistema, é através deste objeto que registraremos os alarmes.

Na linha 3 criamos uma *Intent* que será a ação que queremos executar, que pode ser uma *Activity*, *Service* ou um*BroadcastReceiver*. Para nosso exemplo criamos apenas um *BroadcastReceiver* que “loga” a ação(veja o código no final do post), mas poderia fazer qualquer outra ação, como exibir uma notificação por exemplo.

Na linha 4, criamos um *PendingIntent* de *Broadcast*, para quem não sabe uma *PendingIntent* que é uma *Intent* que outras aplicações podem executar do nosso código com as permissões do nosso aplicativo. Muitos dos serviços do Android que registramos para executar nosso código necessita criar um *PendingIntent*.

Na linha 6, chamamos o método *set(int type, long time , PedingIntent intent)* que agenda uma única execução. No primeiro parâmetro recebemos o tipo de execução que queremos, que podemos utilizar as seguintes constantes.

**ELAPSED\_REALTIME**: Dispara a *PendingIntent* e é baseado na quantidade de tempo desde que o aparelho foi ligado, mas não irá acordar o dispositivo se este estiver hibernando. O tempo contado incluí o tempo em que o dispositivo esteja hibernando.

**ELAPSED\_REALTIME\_WAKEUP** Acorda o dispositivo e dispara o *PendingIntent* após a quantidade de tempo especificada que passou desde o boot do aparelho.

**RTC** Dispara o *PendingIntent* no tempo especificado e não acorda o dispositivo se este estiver hibernando. Ele utiliza o tempo real do relógio do dispositivo, então para agendar utilizando este tipo de alarme devemos usar as classes de tempo do java(*Date*,*Calendar*, etc) para pegar o horário do sistema.

**RTC\_WAKEUP** Acorda o dispositivo e dispara o *PendingIntent* no tempo determinado.

Ainda na linha 6, no segundo parâmetro temos o tempo, lembrando que este tempo é medido em **milissegundos**, no nosso exemplo pegamos o tempo através do método *SystemClock.elapsedRealtime()* que retorna a quantidade de tempo do sistema desde que foi ligado, já que estamos utilizando *ELAPSED\_REALTIME* e somamos a este valor 60 \* 1000 ou 60 segundos.

Agora um alarme que repete!

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | AlarmManager alarmMgr = (AlarmManager) this.getSystemService(Context.ALARM\_SERVICE);    Intent intent = new Intent(this, AlarmReceiver.class);  PendingIntent alarmIntent = PendingIntent.getBroadcast(this, 0, intent, 0);    Calendar calendar = Calendar.getInstance();  calendar.setTimeInMillis(System.currentTimeMillis());  calendar.set(Calendar.HOUR\_OF\_DAY, 14);  long inicio = calendar.getTimeInMillis();    long intervalo =  60 \* 1000;    alarmMgr.setRepeating(AlarmManager.RTC\_WAKEUP, inicio, intervalo, alarmIntent); |

As linhas 1, 3, 4, são iguais ao exmplo anterior, mas agora utilizamos o método *setRepeating* em vez do *set*. O método*setRepeating* recebe como primeiro parâmetro o tipo de alarme, o segundo parâmetro é o tempo que o o alarme irá começar, e como estamos utilizando o tipo de alarme *RTC\_WAKEUP* colocamos o tempo real adquirido através das classes *Date* ou *Calendar*. No exemplo utilzamos um *Calendar* que configuramos o tempo atual e mudamos a hora para executar as 14:00, então nosso exemplo só irá executar a partir das duas da tarde do dia atual.

No terceiro parâmetro colocamos o tempo de intervalo em milissegundos em que o alarme será executado, no nosso exemplo 60 segundos. Também poderíamos utilizar algumas constantes que a classe *AlarmManager* nos fornece como *INTERVAL\_DAY*(um dia), *INTERVAL\_FIFTEEN\_MINUTES*(quinze minutos), *INTERVAL\_HALF\_DAY*(12 horas), *INTERVAL\_HALF\_HOUR*(30 minutos) e*INTERVAL\_HOUR*(uma hora).

E por fim no quarto parâmetro o *PendingIntent* que será executado periodicamente.

Como podemos iniciar um alarme, também podemos cancelar um, através do método *cancel* que recebe a intente que está agendada.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | AlarmManager alarmMgr = (AlarmManager) this.getSystemService(Context.ALARM\_SERVICE);    Intent intent = new Intent(this, AlarmReceiver.class);  PendingIntent alarmIntent = PendingIntent.getBroadcast(this, 0, intent, 0);    alarmMgr.cancel(alarmIntent); |

De forma alternativa ao método *setRepeating* temos o *setInexactRepeating* que permite ao dispositivo sincronizar vários alarme para executarem todos juntos, economizando recursos do aparelho.

Abaixo o *BroadcastReceiver* que estamos utilizando no nosso exemplo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class AlarmReceiver extends BroadcastReceiver {        @Override      public void onReceive(Context context, Intent intent) {          Log.i("ALARME", "O alarme executou as: "+new Date());      }    } |

Lembrando que devemos registrar o *BroadcastReceiver* no manifest.xml.

|  |  |
| --- | --- |
| 1  2  3 | <receiver      android:name=".AlarmReceiver"      android:process=":remote" /> |

Links

<https://developer.android.com/training/scheduling/alarms.html>

<http://developer.android.com/reference/android/app/AlarmManager.html>

<http://developer.android.com/reference/android/app/PendingIntent.html>