Delayed Services

Starring

IntentService, Alarm, BroadcastReceiver, WakefulBroadcastReceiver

Featuring

Notification, PendingIntent, WakeLock

Master IAO

Pr. Oussama REDA

IntentService

The IntentService class is:

- **a specialized *subclass of Service***
- ***implements* a background work queue using a single *HandlerThread*.**
- ***work* submitted to an IntentService *is queued for processing* by a HandlerThread, and *processed in order of submission*.**
- ***stops itself* when it has *no more work in its queue*, to avoid consuming unnecessary resources.**

startService(new Intent(context, MyIntentService.class));

starts* the IntentService if it *isn't already running
***enqueues work* to an *already running instance* if there is one**

IntentService

Create an IntentService *subclass*

- ***extend android.app.IntentService***
- ***implement* the abstract *onHandleIntent* method.**
- ***invoke* the single-argument *constructor with a name* for its background thread (naming the thread makes debugging and profiling much easier).**

Example : ==>

IntentService

Create an IntentService *subclass*

```
public class MyIntentService extends IntentService {
    public MyIntentService() {
      super("thread-name");
    }
    protected void onHandleIntent(Intent intent) {
        // executes on the background HandlerThread.
    }
}
```

Create an IntentService *subclass*

```
public class IntentServiceNthPrime extends
IntentService {
    public IntentServiceNthPrime() {
        super("primes");
    }
    protected void onHandleIntent(Intent
    intent) {
            //calculate Nth prime
    }
}
```

in calling Activity

```
private void
LaunchIntentService(int  nthprime) {
// prepare an intent with nthprime
startService(intent);
}
```

System notifications

- **An icon in the notification area, not always at the very top of the device screen.**

- **Once notification received, notification drawer can be opened (more details).**

Usage with services:

inform the user of long time operations results while the user is on something else

System notifications

To build a notification:

NotificationCompat.Builder builder;

***must* include**

- an icon:

setSmallIcon(int icon)

- a title :

setContentTitle(CharSequence title)

- a message :

setContentText(CharSequence text)

All these methods return a *NotificationCompat.Builder*

System notifications

To build a notification:

NotificationManager nm = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

to fire a notification:

nm.notify(*notif_Id*, builder.build());

***notif_Id* is the identifier of the notification (int).**

a method to notify Results (prime of range n) to the user IntentService

Create an IntentService *subclass*

```
public class IntentServiceNthPrime extends IntentService {
    private void notifyUser(int primeToFind, String result) {
String msg = String.format(
"The %sth prime is %s", primeToFind, result);
NotificationCompat.Builder builder =
new NotificationCompat.Builder(this)
.setSmallIcon(R.drawable.prime_notification_icon)
.setContentTitle(getString(R.string.primes_app))
.setContentText(msg);
NotificationManager nm = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
nm.notify(primeToFind, builder.build());
}
  }
```

Create an IntentService *subclass*

```
public class IntentServiceNthPrime extends
IntentService {
    public IntentServiceNthPrime() {
        super("primes");
    }
    protected void onHandleIntent(Intent
    intent) {
            //calculate Nth prime
notifyUser(rangeOfPrime, resultPrimeOfCalculation);
    }
}
```

in calling Activity

```
private void
LaunchIntentService(int  nthprime) {
// prepare an intent with nthprime
startService(intent);
}
```

Alarms

to schedule work :

make use of *postDelayed*, *postAtTime*, *sendMessageDelayed*, and *sendMessageAtTime*.

for short-term scheduling (work should happen while the application is running in the foreground).

Not suitable for work to happen at some distant time t in the future

because: - App terminates/ended/killed before reaching *time t*

- Device may be asleep ==> CPU powered down ==>

==> scheduled tasks cannot be run

Alarms

Solution:

use an AlarmManager. A *scheduling approach* designed for these kind of situations.

***AlarmManager* :**

- A system service ==> cannot be terminated

- If the device is asleep, It can wake it to deliver scheduled alarms to it

Alarms(scheduling)

to get an AlarmManager:

AlarmManager aLm = (AlarmManager)getSystemService(ALARM_SERVICE);

To schedule an alarm to deliver a PendingIntent object at time *t* we can use the set method :

void set(int type, long triggerAtMillis, PendingIntent operation)

Alarms (scheduling)

Example:

long aDelay = TimeUnit.HOURS.toMillis(3L);

long timeToWake = System.currentTimeMillis() + aDelay;

aLm.set(AlarmManager.RTC_WAKEUP, timeToWake, aPendingIntent);

The *aLm* alarm is scheduled to be delivered at time *timeToWake* and trigger *aPendingIntent* intent.

The alarm will awake the device if it is asleep.

The time used is relative to the Unix epoch.

Alarms (scheduling)

Example:

long aDelay = TimeUnit.HOURS.toMillis(3L);

long timeToWake = System.currentTimeMillis() + aDelay;

aLm.set(AlarmManager.RTC_WAKEUP, timeToWake, aPendingIntent);

The *aLm* alarm is scheduled to be delivered at time *timeToWake* and trigger *aPendingIntent* intent.

The alarm will awake the device if it is asleep.

The time used is relative to the Unix epoch.

Alarms (handling)

Example:

long aDelay = TimeUnit.HOURS.toMillis(3L);

long timeToWake = System.currentTimeMillis() + aDelay;

aLm.set(AlarmManager.RTC_WAKEUP, timeToWake, aPendingIntent);

The *aLm* alarm is scheduled to be delivered at time *timeToWake* and trigger *aPendingIntent* intent.

The alarm will awake the device if it is asleep.

The time used is relative to the Unix epoch.

Alarms
handling with
BroadcastReceiver

BroadcastReceiver

Register in manifest

```
<receiver android:name=".Alarm_Receiver">
    <intent-filter>
        <action android:name="runReceiver"/>
    </intent-filter>
</receiver>
```

<intent-filter> **filters intents received by *Alarm_Receiver.* Only intents with the action *runReceiver* cause the receiver to run.**

Alarms
and
BroadcastReceivers

Set an Alarm for a **BroadcastReceiver**

```
Intent intent = new Intent("runReceiver");
intent.putExtra(Alarm_Receiver.MSG, "wake broadcast with alarm examples!");

PendingIntent broadcastIntent = PendingIntent.getBroadcast(
context, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);

AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
long atTime = calculateAlarmTriggerTime();
am.set(AlarmManager.RTC_WAKEUP, atTime, broadcastIntent);
```

Alarms
and
BroadcastReceivers

Set an Alarm for a **BroadcastReceiver**

if it isn't already awake, AlarmManager will

- Wake the device

- Deliver the Intent to the onReceive method of the BroadcastReceiver Alarm_Raceiver .

```
public class Alarm_Receiver extends BroadcastReceiver {
        public static final String MSG = "message";
    @Override
     public void onReceive(Context ctx, Intent intent) {
          // do some work while the device is awake
     }
}
```

Alarms
and
BroadcastReceivers

Because *AlarmManager.RTC_WAKEUP* is the first parameter of the set method of the AlarmManager

==> the device will remain awake (at least until onReceive completes)

==> some work will be done before the device will be allowed to return to sleep.

```
public class Alarm_Receiver extends BroadcastReceiver {
        public static final String MSG = "message";
    @Override
     public void onReceive(Context ctx, Intent intent) {
          // do some work while the device is awake
     }
}
```

BroadcastReceivers and the UI thread

- The system delivers an alarm to a BroadcastReceiver on the main thread

==> Long running operations are not allowed

- BroadcastReceivers Registred in the manifest have limited lifecycle.

- They can only create *toasts* or posted NotificationManager *notifications* as UI elements.

- onReceive () < 10 seconds or its process may be killed.

- The receiver's life is over when onReceive completes.

BroadcastReceivers and the UI thread

In response to the alarm: If the work that we need to do is not intensive

==> complete it during onReceive()

Example:

Posting notifications to the notification drawer

BroadcastReceivers and the UI thread

```
public class AlarmReceiver extends
BroadcastReceiver {

public static final String MSG =
"message";

@Override

public void onReceive(Context
context, Intent intent) {

NotificationCompat.Builder builder =

new
NotificationCompat.Builder(context)
.setSmallIcon(.....)
.setContentTitle(.....)
.setContentText(.....);


NotificationManager nm =
(NotificationManager)
context.getSystemService(
Context.NOTIFICATION_SERVICE);


nm.notify(intent.hashCode(),
builder.build());
```

BroadcastReceivers and Services

onReceive() < 10 seconds is limited budget

Alternative :

Services to the rescue for long-running operations

BroadcastReceivers and Services

onReceive() < 10 seconds is limited budget

Alternative :

Services to the rescue for long-running operations

Plan :

get an IntentService to do the work off the main thread.

when the work is finished the IntentService stops itself.

Beautiful!

Isn't it?!

BroadcastReceivers and Services

It is! but only if the device is awake

If the device is asleep ==> potential problem. Why?

The device is only awake during onReceive's run. but,

starting a Service directly does not involve a BroadcastReceiver so why bother?

Because starting a Service is an asynchronous operation. So?

The device may return to sleep before the service starts up.

==> the work may be delayed till time the device is awake.

BroadcastReceivers and Services

This is *not what we want*! What *we want* is :

the Service starts up and ensure it completes its work

even if the device was asleep the timethe alarm was delivered

It is possible by first starting the service when the device is awake during onReceive run,

then keep the device awake with *WakeLock*

WakeLock is an object used to force the device to stay awake.

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

BroadcastReceivers and Services

To allow a Service to do background work while keeping the CPU powered up, we only need a *partial WakeLock* which keeps the screen off.

How to make use of WakeLocks?

By means of the class *WakefulBroadcastReceiver*

Two static methods:

- ComponentName *startWakefulService*(Context context, Intent intent);

acquires the WakeLock and starts the Service

- boolean *completeWakefulIntent*(Intent intent);

releases the WakeLock when the Service has finished its work

WakefulBroadcastReceivers and Services

```
class Alarm_Receiver extends BroadcastReceiver {
    @Override
     public void onReceive(Context context, Intent intent) {
         Intent service_Intent =
              new Intent( context, MyIntentService.class);
         WakefulBroadcastReceiver
              .startWakefulService(context, serviceIntent);
    }
}
```

WakefulBroadcastReceivers and Services

```
class MyIntentService extends IntentService {
        @Override
         protected final void onHandleIntent(Intent intent) {
             try {
                    // do background work while the CPU is kept awake
             } finally {
                        WakefulBroadcastReceiver. completeWakefulIntent(intent);
              }
        }
}
```