UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

disim

Mobile Programming Laboratory

ANDROID
Background Services

Teachers

Ing. Tarquini Francesco, Ph.D

Ph.D in Computer Science Engineering

francesco.tarquini@univaq.it

Ing. D’Errico Leonardo

Ph.D Student in Computer Science Engineering

leonardo.derrico@graduate.univaq.it

Teaching Materials

Available on MOODLE platform

http://www.didattica.univaq.it

Google Drive Repository

https://drive.google.com/drive/folders/1ISqZfn0i9Ub3eWNXbvW00rd0hD9ya8OL?usp=sharing

Topics

- Threads
- Background Services
  - Service
  - IntentService
- BroadcastReceivers
  - Manifest-Declared
  - Context-Registered

Threads

When an application is launched, the system creates a thread of execution for the application, called "main."

This thread is very important because it is in charge of dispatching events to the appropriate user interface widgets, including drawing events.

When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly.

Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI.

If the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog.

Threads

When the application has to work for complex operations, the developer has to implement these operations in a new Thread.

In Java he can easily create a new Thread.

```
Thread thread = new Thread(new Runnable() {
   @Override
   public void run() {
      // Complex Operations
   }
});
thread.start();
```

Background Services

A Service is an application component that can perform long-running operations in the background.

It does not provide a user interface

A Service can handle:

- network transactions;
- play music;
- perform file I/O;
- interact with a content provider;

Background Services

Three type of services:

Foreground: performs some operation that is noticeable to the user, such as play audio track;

Background: performs an operation that isn’t directly noticed by user, such as to compact applications storage;

Bound: a service that offers a client-server interface that allows components to interact with the services, send requests, receive results, etc…

Background Services - Service

To create a service you must create a subclass of Service or use one of its existing subclass.

In your implementation, you must override some callback methods:

onCreate: the system invokes this method to perform one-time;

onStartCommand: the system invokes this method by calling startService() when another component requests that the service be started;

onDestroy: the system invokes this method when the service is no longer used and is being destroyed.

Background Services - Service

Any application component can use the service by starting it with an Intent.

A component such as an Activity can start the service by calling **startService**() and passing an Intent, the Intent is passed to the **onStartCommand**(), and the service continues to run until it stops itself with **stopSelf**() or a component stops it by calling **stopService**()

All services must be declared in your application's manifest file, as child of the **<application>** element.

```
<application ...>

        ...

  <service android:name=".MyService" />

</application>
```

Background Services - Service

A component can start a service in bound way, calling the **bindService**() method.

The service receives the Intent in **onBind**() and not in **onStartCommand**().

The bound service **stops itself** when the request is completed.

Background Services - Service

There are two classes you can extend to create a started service:

Service: the basic class. When you extend this class, it’s important to create a new thread in which the service can complete all of its work

IntentService: it is a Service subclass that uses a worker thread to handle all of the start requests, one at a time.

IntentService is the best option if you don’t require that your service handle multiple requests simultaneously.

Background Services - Service

To work in a separate thread you must create a class that extends Handler and handle the requests overriding **handleMessage**() method.

When the request is completed, you remember to stop the service by **stopSelf**() method.

```
public class MyService extends Service {

   ...

   /* Handler */

   private class ServiceHandler extends Handler {

      ServiceHandler(Looper looper) {
         super(looper);
      }

      @Override
      public void handleMessage(Message msg) {
         super.handleMessage(msg);

         // Work in background and in separate thread

         stopSelf(msg.arg1);
      }
   }
}
```

Background Services - Service

When the service is created, you create a separate thread and link it to your handler.

```
public class MyService extends Service {

  private ServiceHandler handler;

  @Override
  public void onCreate() {
     super.onCreate();

     HandlerThread thread = new HandlerThread(
           MyService.class.getSimpleName(), Process.THREAD_PRIORITY_BACKGROUND);
     thread.start();
     handler = new ServiceHandler(thread.getLooper());
  }

          ...
}
```

Background Services - Service

The system invokes onStartCommand() and passes here the Intent that include the data of service. To send these data to the handle you must create a Message and invoke handleMessage() of the handler's instance.

```
public class MyService extends Service {

        ...

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {

          Message message = new Message();
          message.arg1 = startId;
          handler.handleMessage(message);

          return START_STICKY;
        }
        ...
}
```

Background Services - Service

onStartCommand() method must return an integer. The integer value describes how the system should continue the service in the event that the system kills it.

The return value must be one of the following constants:

START_NOT_STICKY: If the system kills the service, do not recreate the service unless there are pending intents to deliver;

START_STICKY: If the system kills the service, recreate the service and call onStartCommand(), but do not redeliver the last intent, unless there are pending intents to start the service;

START_REDELIVER_INTENT: If the system kills the service, recreate the service and call onStartCommand() with the last intent that was delivered to the service.

Background Services - IntentService

Features:

- it creates a **default worker thread** separated from application’s main thread;
- it creates a **work queue** that passes one intent at a time your onHandleIntent();
- it **stops the service** after all of the start requests are handled

```
public class MyIntentService extends IntentService {

   private static final String NAME =
MyIntentService.class.getSimpleName();

   public MyIntentService() {
       super(NAME);
   }

   @Override
   protected void onHandleIntent(@Nullable Intent intent) {

   }
}
```

Background Services - IntentService

A component can start or stop the service calling the following methods:

Start

```
Intent startIntent = new Intent(getApplicationContext(), MyService.class);
startIntent.putExtra("key", "value");
startService(startIntent);
```

Stop

```
Intent stopIntent = new Intent(getApplicationContext(), MyService.class);
stopService(stopIntent);
```

BroadcastReceiver

The recommended way to send and receive status from service is to use a **LocalBroadcastManager**, which limits broadcast Intent objects to components in your own app

Android apps can send or receive broadcast messages from the Android system and other Android apps.

such as using Publish-subscribe design pattern

Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive it.

BroadcastReceiver

The system automatically sends broadcasts when various system events occurred, such as when the system switches in and out of airplane mode.

- Only apps that are subscribed can receive these broadcasts.

The broadcast message itself is wrapped in an Intent object whose action string identifies the event that occurred.

- The Intent may also include additional data bundled.

BroadcastReceiver

The system automatically sends broadcasts when various system events occurred, such as when the system switches in and out of airplane mode.

Only apps that are subscribed can receive these broadcasts.

The broadcast message itself is wrapped in an Intent object whose action string identifies the event that occurred.

The Intent may also include additional data bundled.

Apps can receive broadcasts in two ways:

through **manifest-declared** receivers: your app always receive broadcasts, also if the app is closed.

through **context-registered** receivers: your app can receive broadcasts only when the component, where receiver is registered, is in foreground.

BroadcastReceiver - Manifest-Declared

You perform the following steps:

Specify the <receiver> element in your app’s manifest as a child of <application>

```
<application ...>

  <receiver android:name=".MyReceiver">
     <intent-filter>
        <action android:name="android.intent.action.AIRPLANE_MODE" />
     </intent-filter>
  </receiver>

</application>
```

Subclass BroadcastReceiver and override onReceive()

```
public class MyReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
     // My implementation
  }
}
```

BroadcastReceiver - Context-Registered

You perform the following steps:

Subclass BroadcastReceiver and override onReceive()

```
public class MyReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
     // My implementation
  }
}
```

Create an instance of MyReceiver, an IntentFilter and register the receiver by calling registerReceiver()

```
MyReceiver receiver = new MyReceiver();
IntentFilter filter = new IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
registerReceiver(receiver, filter);
```

To stop receiving broadcasts, call unregisterReceiver

```
unregisterReceiver(receiver);
```

BroadcastReceiver - Context-Registered

There are three ways to send broadcast:

sendBroadcast: sends the broadcasts to all receiver in an undefined order.

sendOrderedBroadcast: send the broadcasts to one receiver at a time. The order receivers run in can be controlled with the android:priority attribute

LocalBroadcastManager.sendBroadcast: sends broadcasts to receivers that are in the same app ad the senders.

BroadcastReceiver - Context-Registered

To send an implicit Intent to all apps

```
Intent broadcastIntent = new Intent();
broadcastIntent.setAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
broadcastIntent.putExtra("key", "value");
sendBroadcast(broadcastIntent);
```

Do not broadcast sensitive information using an implicit Intent use a LocalBroadcastManager (this is much more efficient)

```
Intent broadcastIntent = new Intent();
broadcastIntent.setAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
broadcastIntent.putExtra("key", "value");
LocalBroadcastManager.getInstance(getApplicationContext()).sendBroadcast(broadcastIntent);
```