Utiliser les services sous Android

Romain Raveaux
Polytech'Tours

1°) Objectif

Dans ce TP, nous allons abordé la notion de service dans l'environnement Android.
A l'instar des Activities, des Intents, les services font partie des briques essentielles d'Android. Ils ne disposent pas d'interface utilisateur mais fonctionnent en arrière-plan pour une période de temps indéfinie. L'exemple le plus courant est le lecteur de musique, qui vous permet d'écouter vos mp3 sans bloquer la navigation sur internet ou consulter la liste de vos contacts. Un service peut également rapatrier des données sur internet tels que des flux RSS.
Dans ce TP, nous élaborerons un service qui récupérera toutes les secondes l'heure courante au format : heure-minute-seconde. Cette information sera envoyée et affichée dans le champ texte de l'activity

Dans ce TP nous aborderons :

- Les services (threads, cycle de vie...)
- Services Locaux (LocalService)
    - Déclaration du service
    - Implémentation des listeners
    - Implémentation d'un binder

2°) Création de l'interface graphique

a°)La vue graphique

Votre application aura comme nom, les noms des deux personnes composants votre binôme. Créer une Activity avec 4 boutons et un champ texte.

b°) Les composants graphiques

Dans la fonction *onCreate* de votre activité récupérer les composants graphiques grâce à la méthode findViewById. Les variables correspondants aux objets graphiques seront déclarés comme champs de la classe de votre activité.

```
/** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
```

c°) Les évènements

Sur chaque bouton, ajouter un *listener* pour capturer l'action d'un click que le bouton.

3°) Les services (threads, cycle de vie...)

Les services ont pour but de réaliser des tâches de fond sans aucune interaction avec l'utilisateur pour une durée indéfinie. Il existe deux type de services :

- les services locaux (ou LocalService) qui s'exécutent dans le même processus que votre application
- Les services distants (ou RemoteService) qui s'exécutent dans un processus différent de celui de application

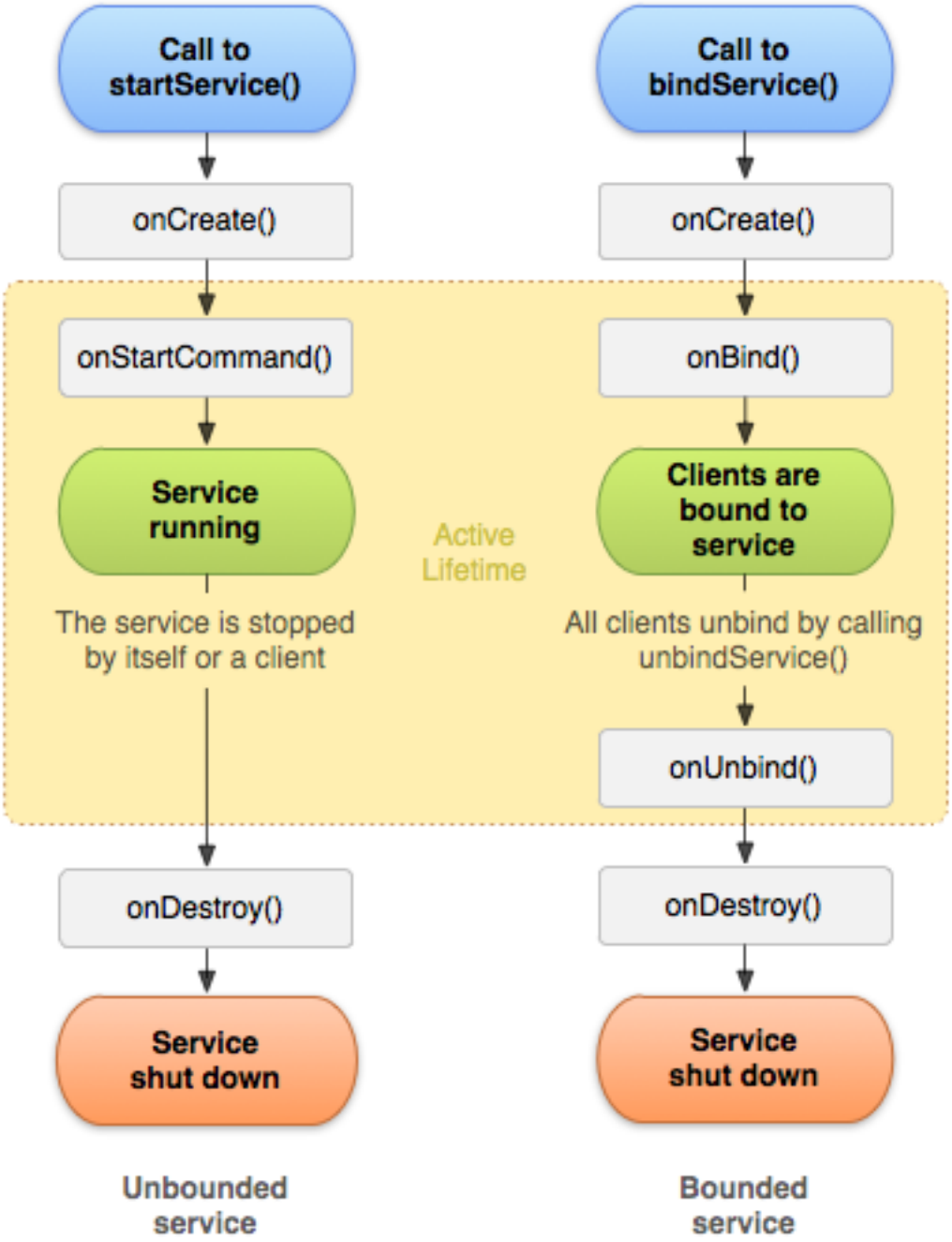
Les services s'exécutent dans le Thread principal du processus parent. Ils doivent être déclarés dans le fichier AndroidManifest.xml:

<service android:name=".subpackagename.ServiceName"/>

Ils doivent étendre la classe Service dont vous devrez surcharger les méthodes suivantes en fonction de vos besoins :

void onCreate(); // initialisation des ressources
void onStart(Intent intent); // SDK<2.0 la tâche de fond démarre
void onStartCommand(Intent intent, int flags, int startId); // SDK>2.0 la tâche de fond démarre
void onDestroy(); // libération des ressources
IBinder onBind(Intent intent); // connexion client distant
boolean onUnbind(Intent intent); // déconnexion d'un client
void onRebind(Intent intent)

Google a publié un post sur la méthode onStartCommand() apparue avec le SDK 2.0 : http://android-developers.blogspot.com/2010/02/service-api-changes-starting-with.html
La méthode onStart() est dépréciée mais doit être redéfinie pour une compatibilité avec les SDK antérieurs (si nécessaire).
Quant au cycle de vie d'un service, Google l'illustre de la manière suivante :



Pour interagir (demarrer/arrêter...) avec un service, deux possibilités s'offrent à nous :

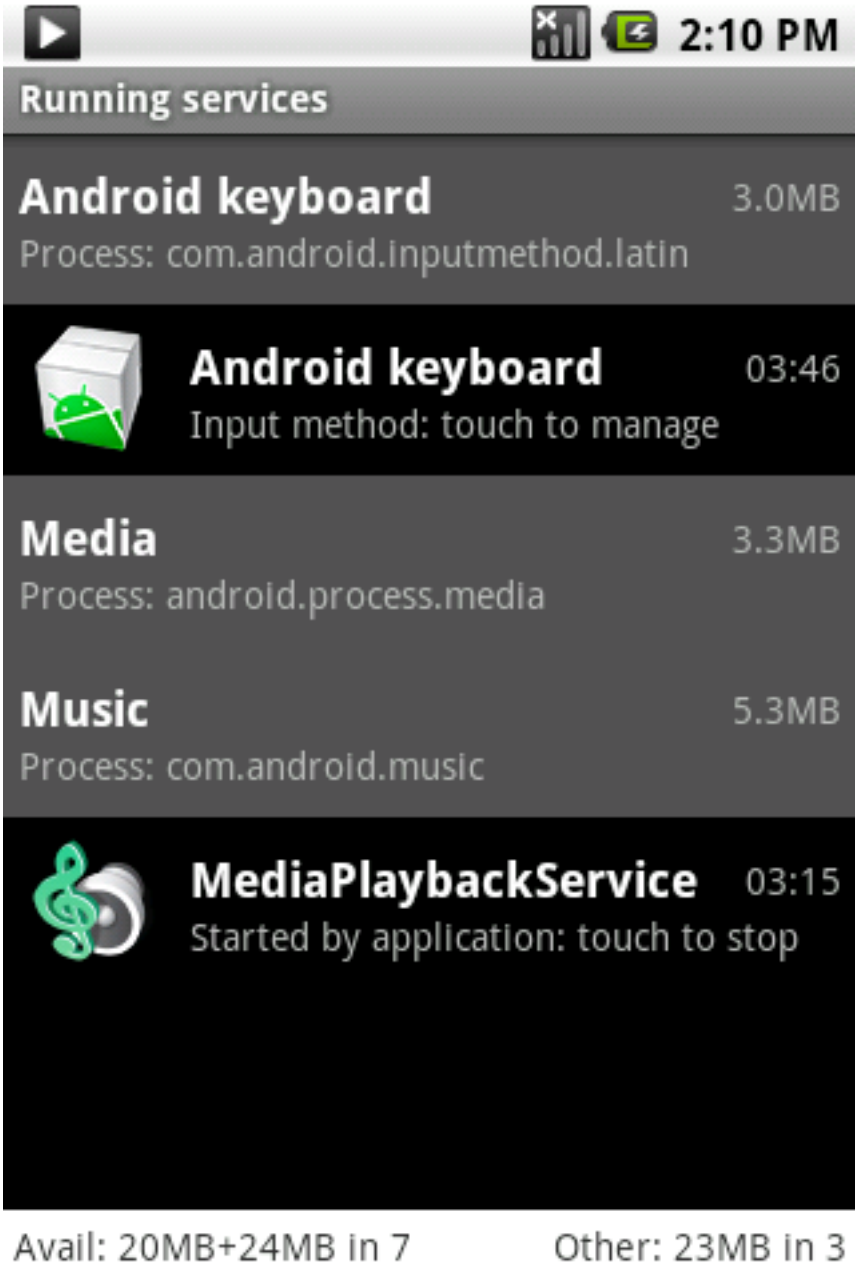
- Soit on appelle la méthode startService() qui invoque la méthode onCreate() puis onStart()

service.startService() | -> onCreate() - > onStartCommand() [service running]
L'appel de la méthode stopService() invoque la méthode onDestroy()

- Soit on appelle la méthode bindService() qui appelle uniquement la méthode onCreate()

activity.bindService() | ->onCreate() [service created]

Il est possible de voir la liste des services exécutés en allant dans Menu > Settings > Applications > Running Services > du téléphone:


2:10 PM
Running services
Android keyboard
3.0MB
Process: com.android.inputmethod.latin
Android keyboard
03:46
Input method: touch to manage
Media
3.3MB
Process: android.process.media
Music
5.3MB
Process: com.android.music
MediaPlaybackService
03:15
Started by application: touch to stop
Avail: 20MB+24MB in 7
Other: 23MB in 3

4°) Services locaux (LocalService)

a°) Définition du service

Un service Local n'est accessible que par les Activity de l'application.
Pour l'exemple, notre service initialise un Timer et une tâche qui sera exécutée toutes les minutes. Nous allons créer une classe héritant de Service et surcharger les méthodes onCreate(), onStart() et onDestroy().

```
public class BackgroundService extends Service {

private Timer timer ;

@Override
public void onCreate() {
    super.onCreate();
    timer = new Timer();
    Log.d(this.getClass().getName(), "onCreate");
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d(this.getClass().getName(), "onStart");
    timer.scheduleAtFixedRate(new TimerTask() {
        public void run() {
            // Executer de votre tâche
        }
    }, 0, 60000);

    return START NOT STICKY;
}

@Override
public void onDestroy() {
    Log.d(this.getClass().getName(), "onDestroy");
    this.timer.cancel();
}
```

b°) A faire :

I°) Modifier votre programme pour qu'il affiche toutes les secondes dans le « Logger » l'heure au format suivant : heure:minute:seconde. Pour se faire vous pourrez utiliser le classe Date.

II°) Les modes d'exécution des services.

A partir du texte suivant extrait de la documentation Android (http://developer.android.com/reference/android/app/Service.html) expliquez quel est l'impact du mot clé `START_NOT_STICKY`.

« For started services, there are two additional major modes of operation they can decide to run in, depending on the value they return from onStartCommand(): START_STICKY is used for services that are explicitly started and stopped as needed, while START_NOT_STICKY or START_REDELIVER_INTENT are used for services that should only remain running while processing any commands sent to them. See the linked documentation for more detail on the semantics. »

c°) Déclaration du service

Nous déclarons le service dans le fichier `AndroidManifest.xml` à l'intérieur de la balise `<application>` :

```
<service android:name=".BackgroundService" />
```

Pour le démarrer, nous faisons appel à la méthode `startService(Intent)` de l'`Activity` prenant en paramêtre un Intent. Ce dernier peut être initialisé de deux manières :

- Soit en lui passant explicitement le context de l'application et la class du service.

```
Intent intent = new Intent(this,BackgroundService.class);
startService(intent);
```

- Soit en déclarant une action dans le fichier `AndroidManifest.xml`...

```
<service android:enabled="true" android:name=".BackgroundService">
    <intent-filter>
        <action android:name=".BackgroundService.ACTION" />
    </intent-filter>
</service>
```

... que nous passons au constructeur de l'intent

```
Intent intent = new Intent(".BackgroundService.ACTION");
startService(intent);
```

Il est possible de passer des objets complets serialisables en paramètre avec les méthodes intent.putExtra(...). Les objets pourront être récupérer dans la méthode `onStart(Intent intent, int startId)` du service avec intent.getExtras().get(String key). Le fonctionnement est similaire à une table de Hash.

Pour arrêter notre service :

```
Intent intent = new Intent(this,BackgroundService.class);
stopService(intent);
```

d°) A faire

I°) Placer le démarrage du service pour qu'il démarre lors de l'appuie sur le bouton « start ».

```
Intent intent = new
Intent(VotreActivity.this,BackgroundService.class);
startService(intent);
```

II°) Placer le stoppage du service pour qu'il s'arrête lors de l'appuie sur le bouton « stop».

```
Intent intent = new Intent(VotreActivity
.this,BackgroundService.class);
stopService(intent);
```

III°) Vérifier dans votre émulateur que le service à bien été créé.

5°) Implémentation des listeners

a°) Quelques concepts JAVA

Les interfaces

Java autorise la séparation entre le code de définition du comportement d'un objet et le code réalisant son implantation.

L'écriture d'une interface, puis d'une classe implantant cette interface réalise cette opération.

Les interfaces déclarent les prototypes des méthodes d'une classe donnée mais ne définissent pas le contenu des méthodes.

http://raphaello.univ-fcomte.fr/javareseau/rmi/Interface.htm

Les variables statiques

Déclarer une variable static (ou meme une fonction) signifie que ce membre n'est pas spécifique à un objet mais à la classe, tous les objets de la classe partagent cette même variable. Et de ce fait tu peux y accéder ainsi : Maclasse.MonVarStatic
ex:

```
class UneClasse {
public static int counter = 0;
}

class Main {
public static void main (String []args) {
UneClasse un = new UneClasse (), deu = new UneClasse ();
un.counter++;
println (UneClasse.counter); // affiche 1
deu.counter++ ;
println (un.counter); // affiche 2
}
}
```

Les Listeners

Dans le contexte d'une interface graphique (Swing, AWT, etc), les listeners permettent au programmeur de réagir suite aux actions de l'utilisateur (clic de souris, touche du clavier enfoncée, etc).

Les « listeners » sont des interface. Ces interfaces fournissent une ou plusieurs méthodes qui peuvent donc être implémentées différemment selon les cas et les besoins, pour répondre aux événements.

b°) Les listeners sur les services

Dans un premier temps, nous allons implémenter un système de listeners très simple. L'interface `IBackgroundServiceListener` implémentée par notre Activity pour écouter les mises à jour du service est la suivante :

Les listeners peuvent être créés sous Eclipse via le menu contextuel :

New->Interface

```
public interface IBackgroundServiceListener {
    public void dataChanged(Object o);
}
```

Nous créons l'interface `IBackgroundService` :

```
public interface IBackgroundService {
    public void addListener(IBackgroundServiceListener listener);
    public void removeListener(IBackgroundServiceListener listener);
}
```

Nous implémentons l'interface dans le service :

Utiliser le mot clé : implements `IBackgroundService` lors de la déclaration de votre service.

```
private ArrayList<IBackgroundServiceListener> listeners = null;
```

Déclarer la variable listeners et allouer de la mémoire pour cette variable dans la fonction public void onCreate().

listeners = new ArrayList< `IBackgroundServiceListener` >();

```
private ArrayList<IBackgroundServiceListener> listeners = null;



// Ajout d'un listener
public void addListener(IBackgroundServiceListener listener) {
    if(listeners != null){
            listeners.add(listener);
    }

}

// Suppression d'un listener
public void removeListener(IBackgroundServiceListener listener) {
    if(listeners != null){
        listeners.remove(listener);
    }
}

// Notification des listeners
```

```
private void fireDataChanged(Object data){
    if(listeners != null){
        for(IBackgroundServiceListener listener: listeners){
            listener.dataChanged(data);
        }
    }
}
```

Il nous reste à mettre à jour la méthode `onDestroy()` du service pour vider la liste des listeners :

```
@Override
public void onDestroy() {
    this.listeners.clear();
    this.timer.cancel();
    Log.d(this.getClass().getName(), "onDestroy");
}
```

d°) A faire

Faites en sorte que l'information d'heure rythmée par le TIMER fasse appel à la méthode fireDataChanged afin de prévenir les listeners écoutant le service d'un changement dans les données.

6°) Implémentation d'un binder

Le binder va nous permettre d'obtenir une référence sur le service en cours. Comme avec un RemoteService, nous allons nous connecter au service et récupérer un Binder. A travers cet objet nous accéderons aux méthodes publiques du service via l'interface IBackgroundService que nous avons définie plus haut. L'avantage de cette solution est d'unifier l'utilisation des LocalService et RemoteService mais surtout de récupérer l'instance du service.
Nous redéfinissons la méthode onBind() :

```
public class BackgroundService extends Service implements
IBackgroundService {

    private Timer timer ;

    private BackgroundServiceBinder binder ;
    private List<IBackgroundServiceListener> listeners = null;

    @Override
    public void onCreate() {
        timer = new Timer();
        binder = new BackgroundServiceBinder(this);
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

}
```

Nous définissons le binder qui nous permettra de récupérer l'instance de notre service :

```
public class BackgroundServiceBinder extends Binder{

    private IBackgroundService service = null;

    public BackgroundServiceBinder(IBackgroundService service) {
        super();
        this.service = service;
    }

    public IBackgroundService getService(){
        return service;
    }
};
```

L'avantage de l'interface `IBackgroundService` est de masquer les méthodes `onCreate()`, `onBind()`... et de ne proposer uniquement `addListener()`, `removeListener()` et pourquoi pas d'autres méthodes métiers.

Il nous reste à nous connecter au service, accéder aux méthodes exposées par l'interface et s'ajouter comme listener pour mettre à jour l'interface utilisateur

7°) Connexion au service

A présent, nous souhaitons que l'activty se connecte au service afin de récupérer une instance du service. Par la même occasion, une fois l'instance du service récupéré, nous ajouterons les listeners qui écouteront les opérations effectuées par le service.

Cette partie du code est à placer lors de l'appui sur le bouton « Connexion ». Le listener créé sera déclaré comme un champ de votre Acitivity.

Les variables « connection » et « service » seront aussi à déclarer comme variables globales à votre Activity.

Définition du listener :

```
Intent intent = new Intent(this,BackgroundService.class);
 //Création des listeners
IBackgroundServiceListener listener = new IBackgroundServiceListener() {
    public void dataChanged(final Object data) {
        VotreActivity.this.runOnUiThread(new Runnable() {
            public void run() {
                // Mise à jour de l'UI
            }
        });
    }
};
```

Au moment voulu, dans la méthode `run()` du Timer nous ferons appel à la méthode privée`fireDataChanged(data)` pour notifier les listeners (dans notre cas, l'Activity) de la mise à jour des données.

Attention ! Si nous nous contentons d'implémenter la méthode `dataChanged()`, l'exception suivante sera levée au runtime :

```
android.view.ViewRoot$CalledFromWrongThreadException:
Only the original thread that created a view hierarchy can touch its views.
```

En effet, la méthode du listener `dataChanged()` est appelée par le Timer, et donc exécutée dans le Thread de celui-ci. L'exception nous indique que toute mise à jour de l'interface utilisateur ne peut se faire que par le Thread responsable de la création des View, Button, TextView... Nous utilisons donc la méthode runOnUiThread de la classe Activity pour poster des instructions dans l'UiThread.

a°) A faire

Modifier le code de la fonction dataChanged afin que l'objet data s'affiche dans le champ texte de l'activité.

Définition de la connexion :

```
//Création de l’objet Connexion
ServiceConnection connection = new ServiceConnection() {
   @Override
public void onServiceConnected(ComponentName name, IBinder service) {
        Log.i("BackgroundService", "Connected!");
        IBackgroundService monservice =
((BackgroundServiceBinder)service).getService();
        monservice.addListener(listener);
    }

    public void onServiceDisconnected(ComponentName name) {
        Log.i("BackgroundService", "Disconnected!");
    }
};


//Connexion au servce
bindService(intent,connection,BIND_AUTO_CREATE);
```

La connexion au service se fait via la méthode bindService (Intent service, ServiceConnection conn, int flags)de la classe Context dont hérite Activity. L'argument flags peut prendre soit 0 soit BIND_AUTO_CREATE pour forcer le service à démarrer, s'il ne l'est pas, au moment de la connexion.

8°) Déconnexion au service

Lors du click sur le bouton déconnexion, placer le code suivant :

```
unbindService(connection);

service.removeListener(listener);
```

9°) Compte Rendu

1°) Le code source commenté

2°) Un rapport décrivant le fonctionnement et le but de ce TP. Qu'avez-vous compris ? Avez-vous rencontré des problèmes ? Ecrivez les éléments qui permettraient à un de vos collègues de faire ce TP facilement.

3°) Le tout doit être déposé sur moodle pour la semaine suivant le TP. (si moodle n'est pas opérationnel, envoyer votre travail par email à votre enseignant (romain.raveaux@univ-tours.fr)

Ce TP est inspiré du tutoriel « Création de Service » écrit par Nicolas DRUET[1], puis adapté pour convenir aux objectifs pédagogiques de Polytech'Tours.

[1] http://blog.developpez.com/android23/p8571/android/creation-de-service/