Services et BroadcastReceivers sous android

Plan

I. Services

I.1. Qu'est-ce qu'un service ?

Tout comme les activités, les services possèdent un cycle de vie ainsi qu'un contexte qui contient des informations spécifiques sur l'application et qui constitue une interface de communication avec le restant du système. Ainsi, on peut dire que les services sont des composants très proches des activités (et beaucoup moins des receivers, qui eux ne possèdent pas de contexte). Cette configuration leur prodigue la même grande flexibilité que les activités. En revanche, à l'opposé des activités, les services ne possèdent pas d'interface graphique : c'est pourquoi on les utilise pour effectuer des travaux d'arrière-plan [1].

I.2. Exemples

Un exemple typique est celui du lecteur de musique. Vous laissez à l'utilisateur l'opportunité de choisir une chanson à l'aide d'une interface graphique dans une activité, puis il est possible de manipuler la chanson dans une seconde activité qui nous montre un lecteur avec des commandes pour modifier le volume ou mettre en pause. Mais si l'utilisateur veut regarder une page web en écoutant la musique ? Comme une activité a besoin d'afficher une interface graphique, il est impossible que l'utilisateur regarde autre chose que le lecteur quand il écoute la musique. On pourrait éventuellement envisager de passer par un receiver, mais celui-ci devrait résoudre son exécution en dix secondes, ce n'est donc pas l'idéal pour un lecteur. La solution la plus évidente est bien sûr de faire jouer la musique par un service, comme ça votre client pourra utiliser une autre application sans pour autant que la musique s'interrompe. Un autre exemple est celui du lecteur d'e-mails qui va vérifier ponctuellement si vous avez reçu un nouvel e-mail [1].

I.3. Les types des services

Il existe deux types de services [1]:

Les plus courants sont les services *locaux* (on trouve aussi le terme *started* ou *unbound service*), où l'activité qui lance le service et le service en lui-même appartiennent à la même application.

Il est aussi possible qu'un service soit lancé par un composant qui appartient à une autre application, auquel cas il s'agit d'un service *distant* (on trouve aussi le terme bound *service*). Dans ce cas de figure, il existe toujours une interface qui permet la communication entre le processus qui a appelé le service et le processus dans lequel s'exécute le service. Cette communication permet d'envoyer des requêtes ou récupérer des résultats par exemple. Le fait de communiquer entre plusieurs processus s'appelle l'IPC. Il peut bien sûr y avoir plusieurs clients liés à un service.

Il est aussi possible que le service expérimente les deux statuts à la fois. Ainsi, on peut lancer un service local et lui permettre d'accepter les connexions distantes par la suite.

I.4. Cycle de vie d'un service

De manière analogue aux activités, les services traversent plusieurs étapes pendant leur vie et la transition entre ces étapes est matérialisée par des méthodes de *callback*. Cependant, le cycle des services contient beaucoup moins d'étapes. La figure suivante est un schéma qui résume ce fonctionnement [1].

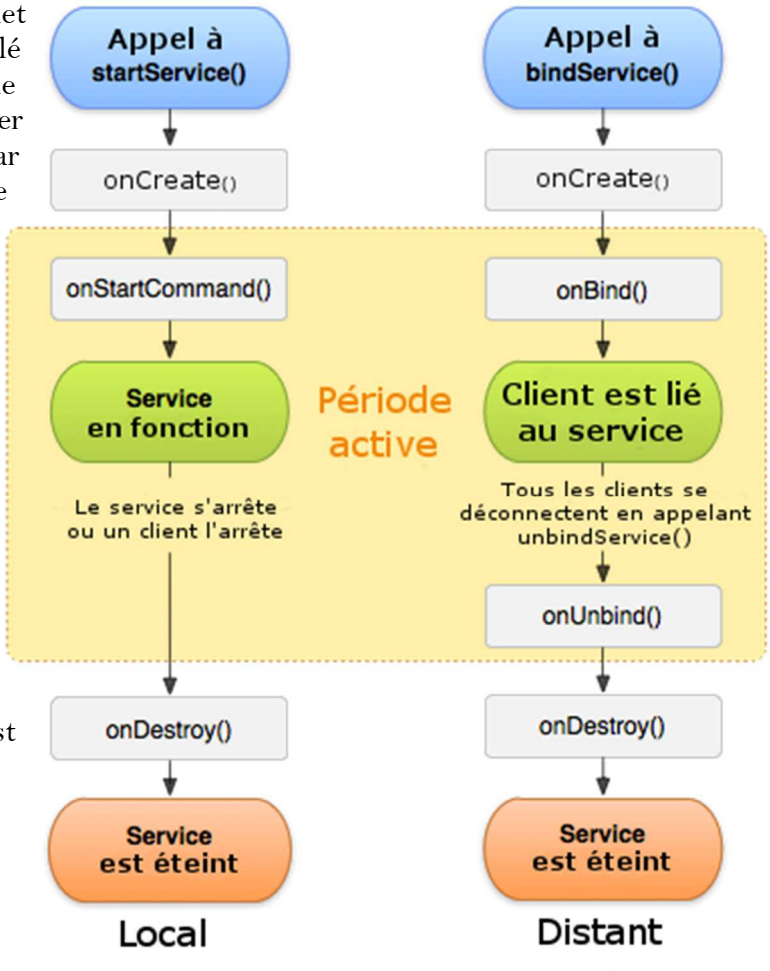

Figure 1 : cycle de vie d'un service local et d'un service distant [1].

I.5. Lancer un service

I.5.1. Lancer un service local

Un service local est lancé à partir d'une activité avec la méthode **ComponentName startService(Intent service)**. La variable retournée donne le package accompagné du nom du composant qui vient d'être lancé.

Exemple :

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

I.5.2. Lancer un service distant

Un service distant est lancé à partir d'une activité avec la méthode boolean bindService(Intent service, ServiceConnection conn, int flags).

I.6. Arrêter un service

Un service local est lancé à partir d'une activité avec la méthode boolean stopService (Intent service).

Exemple :

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

I.7. Ajout d'un Service à une application

I.7.1. La classe Service

La classe Service du package android.app permet de déclarer un service de base. Le code sera alors exécuté dans le thread principal, alors ce sera au développeur de créer un nouveau thread pour ne pas perturber l'exécution du thread UI [1].

Service est une classe abstraite qui contient une seule méthode abstraite qui est **IBinder onBind(Intent intent).**

Exemple :

```
package com.serv;
import android.app.Service;
import android.os.IBinder;
public class MonService extends Service{
        @Override
        public IBinder onBind(Intent intent) {
                return null;
        }
}
```

I.7.2. La classe IntentService

La classe IntentService du package android.app est une sous classe de la classe Service. Cette classe va créer elle-même un thread et gérer les requêtes que vous lui enverrez dans une file. À chaque fois que vous utiliserez startService() pour lancer ce service, la requête sera ajoutée à la file et tous les éléments de la file seront traités par ordre d'arrivée. Le service s'arrêtera dès que la file sera vide [1].

I.7.3. Déclarer le service dans le AndroidManifest.xml

Utiliser la balise <service> et donner dans android :name le nom du package et de la classe du service.

Exemple :

```
<service android:name=" com.serv.MonService"

</service>
```

II. BroadcastReceivers

II.1. Qu'est-ce qu'un BroadcastReceiver ?

- Pour pouvoir recevoir des intents, Android vous permet de créer une classe qui implémente BroadcastReceiver. Ces objets sont conçus pour recevoir des intents (intentions) et appliquer des comportements spécifiques à votre code.
- L'interface BroadcastReceiver ne possède qu'une seule méthode onReceive() que votre classe devra implémenter.
- Un BroadcastReceiver ne vit que le temps de traiter votre onReceive(). L'instance peut être supprimée par le Garbage Collector.
- Les receivers sont limités : ils ne peuvent pas ouvrir de boîte de dialogue par exemple. Le système Android envoie l'intention à tous les BroadCast Receiver abonnés par ordre de priorité (priorité de votre BroadCast dans le fichier AndroidManifest.xml).
- Si un BroadCast souhaite interrompre la réception du Broadcast à ceux d'un niveau inférieure de priorité, il faut utiliser la méthode abortBroadcast() [2].

II.2. La classe BroadcastReceiver

BroadcastReceiver est une classe du package android.content, elle possède une seule méthode abstraite **void onReceive(Context context, Intent intent)**.

Exemple :

```
BroadcastReceiver receiver ;

receiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {
      }
    };
```

II.3. Définir l'action dans l'Intent

Pour définir l'action d'un Intent, utiliser le constructeur **Intent(String action)** ou la méthode **Intent setAction(String action)**.

Exemple :

```
Intent i1= new Intent("com.serv.Info1");

Intent i2= new Intent();
i2.setAction("com.serv.Info2");
```

II.4. Diffuser un Intent

Utiliser la méthode **void sendBroadcast(Intent intent)** pour diffuser un intent.

Exemple :

```
Intent i1= new Intent("com.serv.Info1");
sendBroadcast(i1);
```

II.5. S'enregistrer à l'écoute des diffusions d'une action

Pour s'enregistrer à l'écoute des diffusions d'une action, utiliser la méthode **Intent registerReceiver (BroadcastReceiver receiver, IntentFilter filter)**.

Exemple :

```
registerReceiver(receiver, new IntentFilter("com.serv.Info1");
// Après cet enregistrement chaque fois qu'un Intent qui a pour
// action "com.serv.Info1" est diffusé, la méthode onReceive de
// receiver est exécutée et elle a comme paramètre l'Intent
// diffusé
```

II.6. Se désenregistrer de l'écoute des diffusions

Pour se désenregistrer de l'écoute des diffusions, utiliser la méthode **void unregisterReceiver (BroadcastReceiver receiver)**.

III. Application

III.1. Enoncé

Donner le code d'une application android qui permet d'afficher le temps système. Cette application utilise un Service qui permet d'obtenir le temps système et de le diffuser. Un BraodcastReceiver permet d'obtenir le temps système et de l'afficher dans l'activité. L'interface de l'activité est la suivante :

III.2. Solution

```
package com.serv;
public class MainActivity extends Activity {
  private Button btnDemarrer;  private Button btnArreter;  private TextView tvTemps;
  private BroadcastReceiver receiver;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    init();
  }
  private void init() {
    btnDemarrer = (Button) findViewById(R.id.btnDemarrer);
    btnArreter = (Button) findViewById(R.id.btnArreter);
    tvTemps = (TextView) findViewById(R.id.tvTemps);
    ajouterEcouteur();
  }
  private void ajouterEcouteur() {
    btnDemarrer.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
        demarrer();
      }
    });
    btnArreter.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
        arreter();
      }
    });
    receiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {
          afficher(intent);
      }
    };
  }
  protected void afficher(Intent intent) {
    String temps = intent.getStringExtra("temps");
    tvTemps.setText(temps);
  }
  protected void arreter() {
    Intent i = new Intent(this, PremierService.class);
    stopService(i);
  }
  protected void demarrer() {
    Intent i = new Intent(this, PremierService.class);
    startService(i);
  }
  @Override
  protected void onResume() {
    super.onResume();
    registerReceiver(receiver, new IntentFilter(PremierService.ACTION_TEMPS));
  }
```

```
  @Override
  protected void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
  }
}
```

```
package com.serv;
//imports
public class PremierService extends Service {
  public static final String ACTION_TEMPS = "com.serv.temps";
  private Thread th;  private boolean actif;
  @Override
  public IBinder onBind(Intent intent) {
    Log.i("Service", "onBind");
    return null;
  }
  @Override
  public void onCreate() {
    Log.i("Service", "onCreate");
    actif = true;
    super.onCreate();
    afficherTemps();
  }
  private void afficherTemps() {
    Runnable r = new Runnable() {
      @Override
      public void run() {
        while (actif) {
          afficher();
        }
      }
    };
    th = new Thread(r);
    th.start();
  }
  protected void afficher() {
    try {
      String s = DateFormat.format("MM/dd/yy hh:mm:ss",
          System.currentTimeMillis()).toString();
      Log.i("", s);
      envoyerMessage(s);
       th.sleep(1000);   // ou SystemClock.sleep(1000);
    } catch (InterruptedException e) { e.printStackTrace();  }
  }
  private void envoyerMessage(String temps) {
    Intent intent = new Intent(ACTION_TEMPS);
    intent.putExtra("temps", temps);
    sendBroadcast(intent);
  }
  @Override
  public void onDestroy() {
    Log.i("", "onDestroy");
    actif = false;
    super.onDestroy();
  }
}
```

Références :

[1] https://openclassrooms.com/courses/creez-des-applications-pour-android/les-services-3
[2] http://nbenbourahla.developpez.com/tutoriels/android/broadcast-receiver-android/