PROGRAMMATION DE COMPOSANTS MOBILES (ANDROID)

WIESLAW ZIELONKA

Service

Composant android de l'application sans interface graphique. Utilisé pour de long calcul et des longues opérations qui peuvent (doivent?) être exécutées en arrière plan.

Une fois mis en marche le service peut continuer à s'exécuter même quand l'utilisateur passe à une autre application.

Un composant android (activity ou autre) peut être lié (bind) au service et interagir directement avec celui-ci.

Le service n'est pas un nouveau processus ni un nouveau thread.
Le service ne démarre pas de nouveau thread, le code de service s'exécuterons le thread principal de l'application. Donc si le service doit exécuter une tâche en arrière plan c'est à vous de créer un thread dans le service pour cette tâche.

Service

la classe Service est abstract. La seule méthode abstrait (donc la seule à redéfinir obligatoirement) est la méthode onBind()

Type de service :

1. service en arrière plan
2. service en avant plan
3. service lié (bound)

Chaque service commence l'exécution comme le service en arrière plan.

Service

Le service est exécuté dans le thread principal de l'application.

Dans la majorité des cas **il faut créer explicitement un Thread à exécuter dans le service** et les opérations du service doivent être effectuées dans ce thread.

Besoin d'un service ou non? Si vous avez une tâche à effectuer mais uniquement quand l'activité est visible (par exemple jouer la musique uniquement quand l'activité est visible) alors le service ne sert à rien, utilisez : les Threads dans l'activité (ou HandlerThread).

Service

```
class MyService : Service() {


    /* onBind() : non-utilisé par les services background/foreground
     * mais nécessaire à définir */
    override fun onBind(intent: Intent): IBinder? {  TODO() }

    override fun onCreate() { /* ici la création MediaPlayer */ }

    override fun onStartCommand(intent: Intent, flags: Int,
                                startId: Int): Int {
        /* implementer les opérations start/pause/stop
         * sur le mediaPlayer */
    }

    override fun onDestroy() {  /*stopper MediaPlayer et libérer
                                 * les ressources */ }
}
```

On peut demander de faire démarrer le service plusieurs fois, le système fournit unique startId, une sorte de jeton qui permet identifier les démarrages.

Construire un Service

Ecrire une classe dérivée de Service et implémenter les méthodes suivantes :

- `onStartCommand()` – le service peut s'exécuter à l'infini, par défaut en arrière plan. Il doit s'arrêter lui-même en exécutant **`selfStop()`** ou être arrêter par un autre composant avec **`stopService().`** Pas besoin d'implémenter pour le service de type bound.
- `onBind()` – tjrs à implémenter mais retourner tout de suite null pour le service de type "started". Pour le service de type "bound" implémenter l'interface de communication avec le client et retourner l'objet **`IBinder.`**
- `onCreate()` - exécutée une seule fois à la création du service (avant `onStartCommand()` ou `onBind()`)
- `onDestroy()` – appelée quand le service est détruit ou terminé. Libérer les ressources utilisées.

onStartCommand

```
onStartCommand(Intent intent, int flags, int startId) : Int
```

onStartCommand() doit terminer en retournant une de trois valeurs :

- **`Service.START_NOT_STICKY`** – si le service tué par Android alors n'essayer pas de le recréer (sauf s'il y a des Intents en attente).
- **`Service.START_STICKY`** - si le service tué par android faire le recréer et lui passer Intent null (sauf s'il y a des Intents en attente, dans ce cas envoyer Intent suivant). Approprié pour MediaPlayer.
- **`Service.START_REDELIVER_INTENT`** – si le service tué par android alors le redémarrer avec le même intent. Approprier pour téléchargement de fichiers.

flags : est soit 0 soit combinaison de `START_FLAG_REDELIVERY` et `START_FLAG_RETRY` pour indiquer `si c'est la nouvelle demande de service ou ancienne demande` re-envoyée après l'interruption du service

`startId` – unique id pour identifier la tâche

démarrer le service

Démarrer le service :

remplacer par le nom de votre service

```
val intent = Intent(context = this, Service::class.java )
```

Et on démarre le service avec l'appel à :

```
applicationContext.startService(intent)
```

Au démarrage le service est en mode background.

Arrêter le service

- stopSelf() pour que le service qui s'arrête lui même
- context.stopService(id : Int) pour arrêter le service de l'extérieur (par exemple depuis l'activité)

Le service continue à s'exécuter même si l'application n'est plus visible sur l'écran mais reste en background.

Par contre si l'application est fermée définitivement alors le service s'arrête aussi.

Déclarer le service dans AndroidManifest

Il faut déclarer le service dans AndroidManifest.xml :

```
<application ... >

  <service android:name=".MonService"
    android:exported="false"
    android:description="ne touche pas à mon service" />

</application>
```

`ed="false"` – le service privé non disponible pour d'autres applications
`android:description` – une description de service visible par l'utilisateur

L'utilisateur peut voir les services actifs et supprimer les services qu'il ne connaît pas. L'attribut `description` donne à l'utilisateur des infos concernant le service et permettent de prendre une décision si le service est à supprimer ou non.

Service "foreground"

Service "foreground"

Service foreground est visible par l'utilisateur parce qu'il s'affiche dans les notifications.

Impossible de supprimer la notification correspondante au service foreground tant que ce service est actif.

Exemples :

- le service qui joue la musique et qui peut afficher le titre de chanson jouée en ce moment dans les notifications
- le service qui affiche une distance parcourue (sur une montre android)

Service "foreground"

Pour que le service puisse passer en mode "foreground" il faut ajouter une permission dans AndroidManifest.xml :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>

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

    <application ...>
        ...
    </application>
</manifest>
```

Notez que uses-permission s'ajoute à l'extérieur de la balise application.

Service

```
class MyService : Service() {

    override fun onBind(intent: Intent): IBinder? {  TODO() }

    override fun onCreate() {       }//facultatif

    override fun onStartCommand(intent: Intent, flags: Int,
                                startId: Int): Int {

            .....
        /* le service passe en mode  foreground */
        startForeground( id : Int,  notification : Notification)
            ......

    }

    override fun onDestroy() {  //stopper MediaPlayer }
}
```

Le service passe en mode foreground en exécutant `startForeground()` avec comme deuxième paramètre la notification qui s'affiche tant que le service reste en mode foreground. Le paramètre id : identifiant de la notification permet de supprimer la notification

```
notificationManager.cancel((id)
```

et repasser en mode background.

passer en mode "foreground"

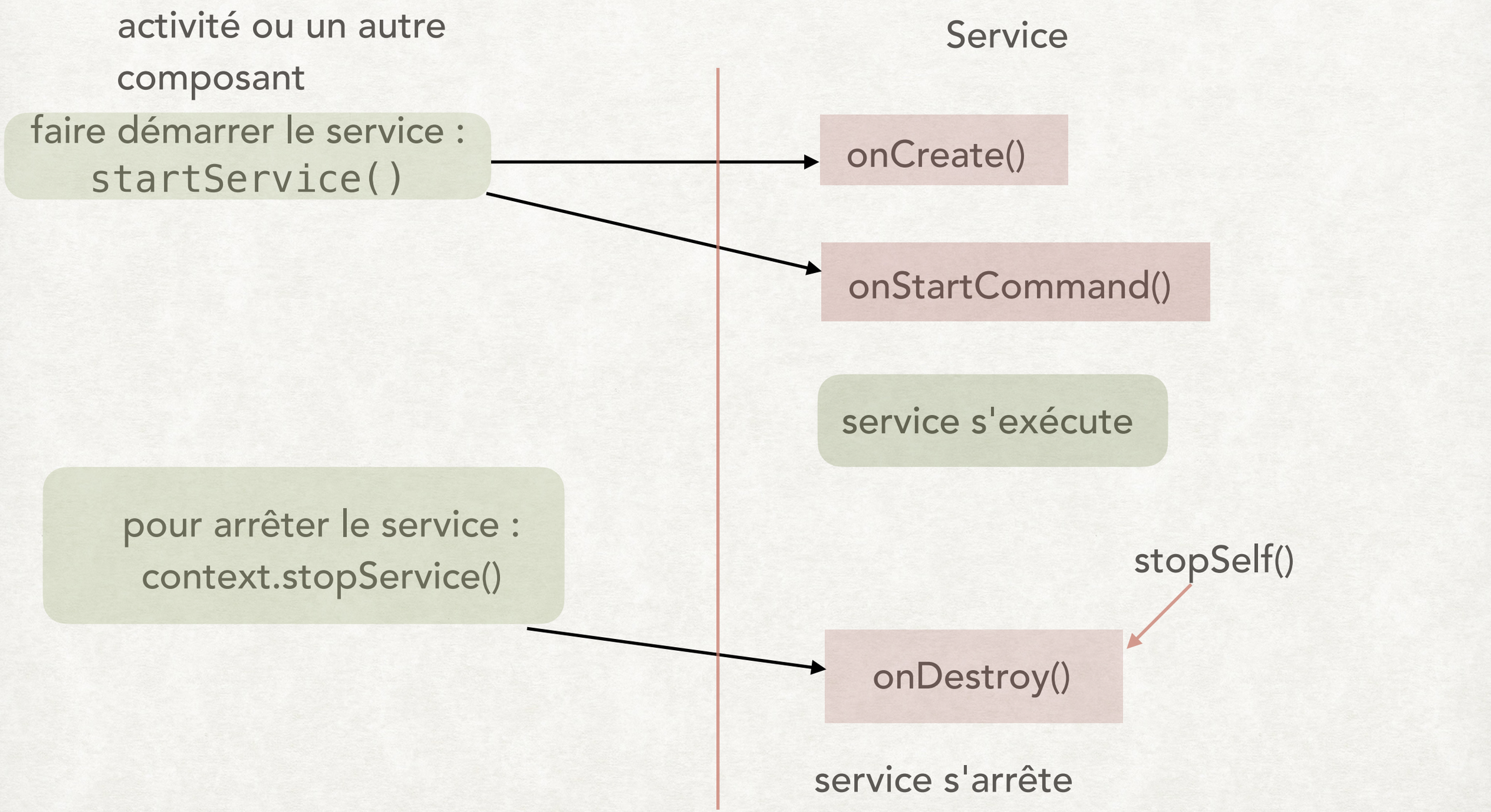
Pour passer en mode "foreground le service, déjà en marche, doit fait lui même l'appel :

```
startForeground( id: Int, notification : Notification )
```

`startForeground()` passe une notification à `NotificationManager`.
L'utilisateur peut "voir" que le service tourne en regardant les notifications.

Tant que startForeground() n'est pas exécuté le service reste en mode background, sans vraiment pouvoir interagir avec l'utilisateur.

Cycle de vie de service (started)



Le service peut s'arrêter aussi lui même juste en terminant l'a tâche à exécuter,` dans ce cas pas la peine de faire appel à stopService()

quitter le mode "foreground"

Pour revenir en mode background le service doit exécuter :

```
stopForeground(STOP_FOREGROUND_REMOVE)
```

Cela enlève aussi la notification.

Comment passer l'information vers un service ?

La question ne se pose pas pour le service lié (bound). L'activité lié au service peut appeler les méthodes du service.

Pour le service background/foreground on passe l'information dans le intent envoyé vers le service et que le service récupère comme le premier paramètre de onStartCommand().

Rappelons que Intent possède un Bundle et nous pouvons y mettre les valeurs à l'aide

```
intent.put(key, value)
```

et le service peut récupérer ces valeurs à l'aide

```
intent.getExtra...(key, ...)
```

Comment passer l'information vers un service ?

Mais l'information à faire passer vers un service c'est souvent une action à effectuer et la donnée associée c'est un objet de type Uri (Uniforme Resource Identifier).

Intent possède deux attributs :
String : intent.action
Uri : intent.data
qui permettent de spécifier l'action que l'activité demande au service et Uri de ressource à utiliser pour effectuer cette action.

Comment passer l'information vers un service ?

Par exemple l'activité qui veut qu'un service joue la musique en utilisant un MediaPlayer peut préparer Intent suivant :

```
val serviceIntent = Intent(this,MusicService::class.java)
.apply {
    action = "start_action"
    data = getUriFromRes(this@MainActivity, id)
}
startService(serviceIntent)
```

où getUriFromRes() une fonction (à écrire!) qui construit Uri à partir de l'identifiant id de la ressources.

Et la communication de service vers l'activité ?

Mais parfois le service doit aussi passer l'information vers l'activité. Par exemple pour le service qui joue la musique il est naturel d'informer l'activité quel est l'état de MediaPlayer.