Android
Intent et Service

Rémi Forax

Discussion inter-activité

- Problème à résoudre
  - Séparation forte entre chaque application
    - 1 application / 1 processus
  - mais nécessité de communiquer entre application
- Solution : envoie de message synchrone
  - Intra application: facile, même espace mémoire
  - Inter-application: IPC spécifique Android (AIDL)

Intent

Message de communication de haut niveau définie dans l'API Java pour communiquer une action d'un composant (activité, service, etc) vers un autre

Types de transmission d'Intent

- Unicast: vers un composant explicitement nommé
  - Vers une classe Java
- Anycast: vers un composant assurant une certaine action (visualisation, édition, launcher, ...) sur certain type de données
- Multicast: diffusion vers des recepteur de broadcast inscrit pour recevoir un type d'Intent

Par défaut, la communication est uni-directionnel, mais on peut répondre à un Intent par un autre Intent

Structure d'un Intent

- Un Intent est constitué de:
  - Action à réaliser
  - Donnée sur laquelle réaliser l'action sous forme d'URI (setData()) ou d'un type MIME (setType())
  - Paramètre optionnels (EXTRA)

- Création d'un Intent
  - Intent(Context, Class<?>) pour l'appels explicite
  - Intent(String action, URI) pour l'appel implicite
  - addCategory(String category) ajout de catégories
  - putExtra(String key,value)
  - setFlags(flags) permission sur les données, relation activité/BackStack

Actions d'Intent prédéfinies

ACTION_MAIN: action principale

ACTION_VIEW: visualiser une donnée

ACTION_ATTACH_DATAT: attachement de donnée

ACTION_EDIT: Edition de donnée

ACTION_PICK: Choisir un répertoire de donnée

ACTION_CHOOSER: menu de choix pour l'utilisateur

- EXTRA_INTENT contient l'Intent original, EXTRA_TITLE le titre du menu

ACTION_GET_CONTENT: obtenir un contenu suivant un type MIME

ACTION_SEND: envoyé un message (EXTRA_TEXT|EXTRA_STREAM) à un destinataire non spécifié

  ACTION_SEND_TO: on spécifie le destinataire dans l'URI

ACTION_INSERT: on ajoute un élement virge dans le répertoire spécifié par l'URI

ACTION_DELETE: on supprime l'élement désigné par l'URI

ACTION_PICK_ACTIVITY: menu de sélection selon l'EXTRA_INTENT mais ne lance pas l'activité

ACTION_SEARCH: effectue une recherche

etc...

Intent Filter

Définie dans le AndroidManifest.xml dans la balise <activity>

- Nom de l'action
  - <action android:name=”XXX”/>
- Category
  - <category android:name=”XXX”/>
  - Android.intent.category.[DEFAULT|BROWSABLE|TAB|ALTERNATIVE|LAUNCHER|HOME|PREFERENCE|TEST]
- Type MIME
  - <category android:mimeType=”XXX”/>

Exemples

Le classique mon activité déclare une application

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Mon activité sait lire et éditer les images JPEG

```
<intent-filter android:label="@string/jpeg_editor">
  <action android:name="android.intent.action.VIEW" />
  <action android:name="android.intent.action.EDIT" />
  <data android:mimeType="image/jpeg" />
</intent-filter>
```

Service

- Composant d'un application exécutant du code sans partie graphique
  - Opération en arrière plan qui dure longtemp
  - Expose une API visible par d'autre application

- Un service en lancer sur la thread principale
  - L'exécution des méthode onXXX() doivent être brève sinon ANR
    - Les calculs long doivent être réalisé dans de nouvelle thread new Thread(new Runnable() { ... }).start()

Service

Un service est tué par le système si pénurie de ressources (trops de processus, pas assez de mémoire)

- La susceptibilité d'être tué dépend de sa priorité
- La priorité d'un service dépend (par défaut) des applications qui l'utilise

    La priorité est ajustée lors de la connection et la deconnection

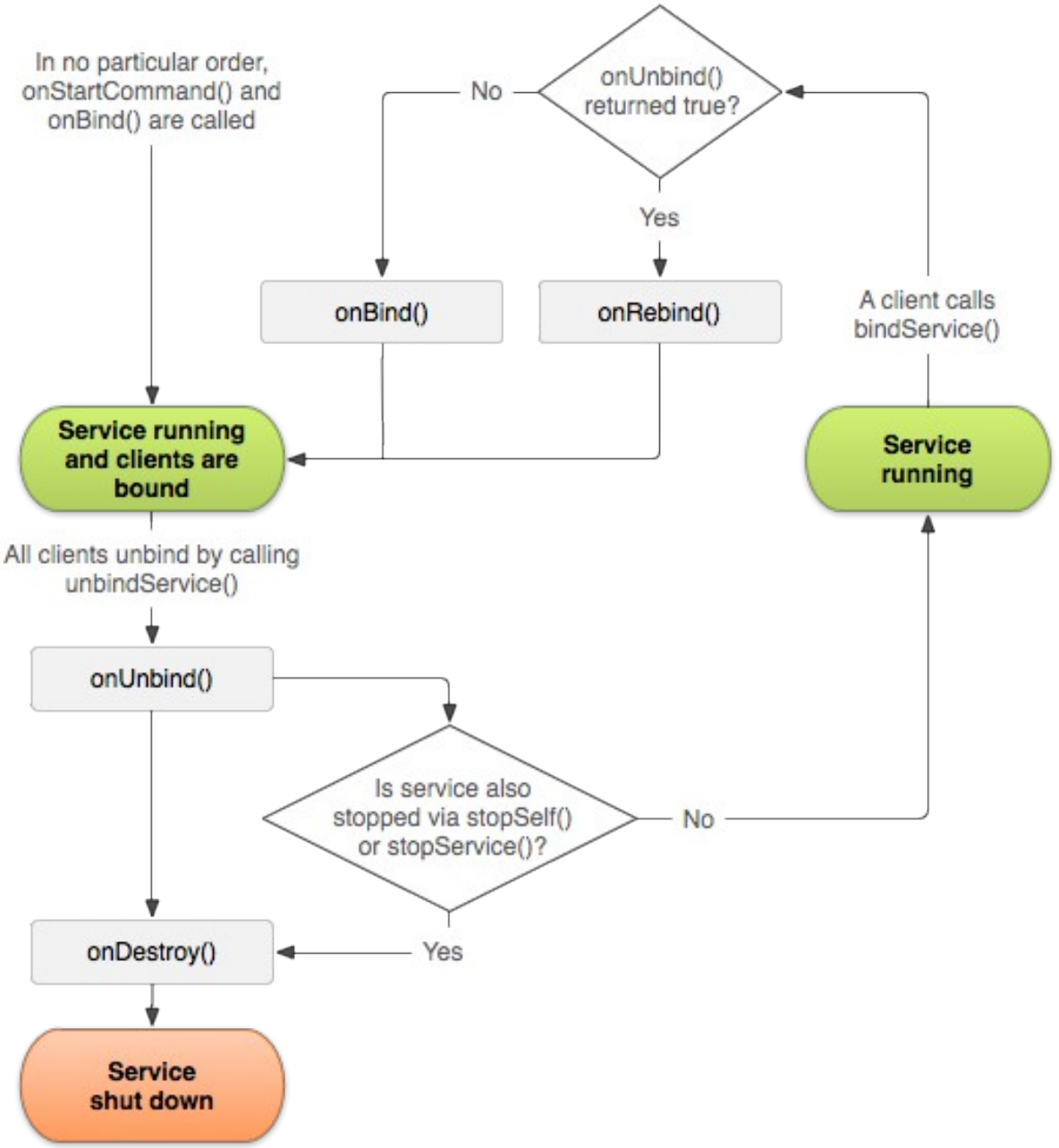
Service

Deux modes d'utilisations

1. Sousmission de travaux asynchrone par un Intent
   - Appel onStartCommand(Intent, flags, startId)
2. Appel à travers un IBinder
   - IBInder onBind()/onRebind()

+ les méthodes onCreate() et onDestroy()

Cycle de vie


In no particular order, onStartCommand() and onBind() are called
No
onUnbind() returned true?
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?
No
Yes
onDestroy()
Service shut down

1. Réagir à un Intent

On demande une réaction à un Intent en appelant Context.startService(Intent)

- La communication est uni-directionnel

- Le service est créé si nécessaire
- Appel onStartCommand(Intent,flags,startId)
  - flags est 0, START_FLAG_REDELIVERY (), START_FLAG_RETRY ()
  - Doit répondre START_STICKY (le service reste), START_NOT_STICKY ou START_DELIVER_INTENT
- Le service peut être stoppé, par lui-même stopSelf, par celui qui l'a lancé stopService(Intent)

Traitement long / IntentService

IntentService, classe spécialisée pour les traitements long

- appel onHandleIntent(Intent) dans une thread de travail (worker thread)

- Une seul thread de travail pour toutes les appels ??

2. Connexion à un Service

Un composant appelle Context.bindService(intent,serviceConnection,flags) pour se lier (bind) à un service

- serviceConnection est un listener
  - onServiceConnected(ComponentName,IBinder)
  - onServiceDisconnected(ComponentName)
- flags:

  BIND_AUTO_CREATE (démarre le service si nécessaire), BIND_ADJUST_WITH_ACTIVITY(monte la priorité au même niveau que l'activité), BIND_WAIVE_PRIORITY (pas de changement de priorité)

=> La méthode IBinder onBind()/onRebind() est appelée sur le service

Local Service / Binder

Si le service est local, il est possible d'utiliser la classe Binder

```
public class HelloService extends Service {
    public class HelloBinder extends Binder {
      public HelloService getHelloService() {
        return HelloService.this;
      }
    }
    private final HelloBinder binder = new HelloBinder();

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

    // API
    public String hello() {
      return “hello local service”;
    } }
```

Et pour une activité ...

```
public class HelloActivity extends Activity {
  HelloService service;
  ...
  @Override protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, HelloService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
  }
  @Override protected void onStop() {
    super.onStop();
    if (service != null) {
      unbindService(connection);
      service = null;
    }
  }
  private final ServiceConnection connection = new ServiceConnection() {
     @Override
      public void onServiceConnected(ComponentName className, IBinder service) {
        HelloBinder binder = (LocalBinder) service;
        service = binder.getService();
        Toast.makeText(HelloActiviy.this, service.hello(), Toast.LENGTH_SHORT).show();
      }
      @Override
      public void onServiceDisconnected(ComponentName className) {
        service = null;
      } }
```

Le Messenger

- Gère une queue de Messages inter-process
  - Messenger.getBinder() crée un IBinder
  - Message.obtain() crée un Message
    - Message.obtain(int what, int arg1, int arg2, Object obj)
    - replyTo (optionel) Messenger pour la réponse
  - Messenger.send(message) permet d'envoyer un message
  - Un Handler permet de recevoir des Messages
    - new Messenger(new Handler() { ... })

Hello avec un Messenger

```
public class HelloService extends Service {
    static final int MSG_HELLO = 1;

    private final Messenger messenger = new Messenger(new Handler() {
      @Override
       public void handleMessage(Message msg) {
           switch (msg.what) {
               case MSG_SAY_HELLO:
                    Message response = Message.obtain(MSG_HELLO, “hello world”);
                    try {
                      response.replyTo.send(msg);
                    } catch (RemoteException e) {
                      // do nothing
                    }
                     break;
               default:
                 super.handleMessage(msg);
           }
       }
    });

    @Override
    public IBinder onBind(Intent intent) {
      return messenger.getBinder();
    } }
```

Activité avec un messenger

```
public class HelloActivity extends Activity {
   Messenger serviceMessenger;
   private final Messenger messenger = new Messenger(new Handler() {
      @Override
       public void handleMessage(Message msg) {
           switch (msg.what) {
               case HelloService.MSG_SAY_HELLO:
                   Toast.makeText(HelloActivity.this, msg.obj, Toast.LENGTH_SHORT).show();
                   break;
               default:
                 super.handleMessage(msg);
           }
       }
    });
   ...
   @Override
   protected void onStart() {
     super.onStart();
     Intent intent = new Intent(this, LocalService.class);
     bindService(intent, connection, Context.BIND_AUTO_CREATE);
   }
   @Override
   protected void onStop() {
     super.onStop();
     if (serviceMessenger != null) {
        unbindService(connection);
        serviceMessenger = null;
     }
   }
```

Activité (suite ...)

```
...
private final ServiceConnection connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName className,
                                   IBinder service) {
      serviceMessenger = new Messenger(service);
      Message msg = Message.obtain(null, MSG_HELLO, 0, 0);
      msg.replyTo = messenger;
      try {
        serviceMessenger.send(msg);
      } catch (RemoteException e) {
         // do nothing
      }
    }
    @Override
    public void onServiceDisconnected(ComponentName className) {
      serviceMessenger = null;
    }
}
```

Service inter-processus

Android Interface Definition Language (AIDL)

- Langage de définition (IDL) pour les appel inter-processus

sort largement du cadre du cours mais la façon de procéder est proche de CORBA ou des RMI

Service en avant-plan

Il est possible de metre un Service en avant-plan

- Apparition dans la zone de notification
- Moins de chance de se faire tuer si manque de ressource

Démarrer le Service avec

- Service.startForeground(int id, Notification notification)

Notifications

Zone centralisée d'affichage d'informations pour l'utilisateur

3 formats

- Petite icône dans la barre de notification
- Vue normale dans le tirroir ouvert
- Vue étendue dans le tirroir ouvert

– Créer une notification
  - NotificationCompat.Builder.build()
– Obtenir le NotificationManager
  - context.getSystemService(Context.NOTIFICATION_SERVICE)
– Soumission de la notification au NotificationManager
  - notificationManager.notify(String tag, int id, Notification notification)
– Supprimer une notification
  - notificationManager.cancel(String tag, int id)

Services Systèmes

- Service pré-existant assurant un accès au matériel ou à des fonctionnalités globales
- Récupération du Service

  context.getSystemManager(String)

  Noms des services (X_SERVICE) sont des constantes dans Context : POWER, ALARM, NOTIFICATION, KEYGUARD, LOCATION, SEARCH, VIBRATOR, CONNECTIVITY, WIFI, DOWNLOAD, etc