Services & *thread* UI

Jean-Marc Lecarpentier
Université de Caen

Au programme

- Problématique de performances et de réactivité
- Gestion des exécutions en arrière-plan
- Outils Android de gestion des threads

Problématique

- Exécution de l'app sur le *thread* principal
  ⇒ problèmes potentiels
- Tâches "longues" affectent la réactivité de l'app
  ⇒ utilisateurs frustrés

  ⇒ erreurs, risques d'ANR

Solutions

- *thread* principal : alléger, alléger, alléger !!
- Tâches longues ⇒ lancer en arrière-plan
- Planifier des tâches à intervalles réguliers
- Utiliser les *Loaders* (cf. cours n°3 - Stockage)

Android et Threads

- “UI Thread” = activité en cours d’exécution
- “Worker thread” = *thread* séparé du *thread* UI
  = tâche en arrière-plan
- Problème : “Tout” est exécuté par défaut sur le *thread* UI

Solution Java

- Utiliser Java pour gérer les threads
- new Thread() et objets Runnable
- Mauvaise solution à priori
- ⇒ Utiliser les outils Android

Outils Android

- AsyncTask
  pour des tâches ponctuelles
  qui “renvoient” des données
  au *thread* UI

- IntentService
  pour des tâches plus longues
  et qui doivent tourner même
  quand l’app n’est pas au 1er
  plan

AsyncTask

- Utilitaire pour exécuter une tâche en dehors de la thread UI

- Peu fiable si Activity redémarre (changt d'orientation)

- Callbacks :
  doInBackground() pour programmer la tâche en arrière plan (worker thread)
  onPostExecute() pour "renvoyer" des données une fois terminé

- execute() pour lancer AsyncTask

AsyncTask

- Exemple : AsyncTask dans Activity

```
public class MyActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      // démarre l'activité
     …
     // démarre le worker
      DoSomethingTask worker = new DoSomethingTask();
     worker.execute(stringParameter);
   }

   private class DoSomethingTask extends AsyncTask<String, Void, String> {
       // démarre le worker thread
       protected String doInBackground(String... urls) {
            // fait une opération en arrière-plan
           return myString;
       }

       // permet de mettre à jour l'UI
       protected void onPostExecute(String result) {
           (TextView) findById(R.id.mon_text).setText(result);
       }
   }
}
```

http://developer.android.com/reference/android/os/AsyncTask.html

AsyncTask

- Inconvénient : AsyncTask inclus dans Activity
  ⇒ AsyncTask non utilisable ailleurs

- Solution :
  - Créer AsyncTask indépendant
  - Créer interface implémentée par Activity
  - AsyncTask.onPostExecute envoie les données à Activity via l'interface

http://stackoverflow.com/questions/12575068/how-to-get-the-result-of-onpostexecute-to-main-activity-because-asynctask-is-a#answer-12575319

Multi-threading

- Autres possibilités : utiliser les threads Java et les Runnable, ou bien les HandlerThread de Android

- Problématique : gérer le nombre de threads lancées en parallèle et s'assurer que le système va pouvoir y répondre

- Android propose une file d'attente pour éviter de multiplier les threads

IntentService

- Crée un worker qui exécute les Intent reçues
- File d'attente : une seule Intent exécutée à la fois
  ⇒ pas de risque lié au multi-threading
- Pas de moyen d'arrêter une tache démarrée
- Arrête le service quand la file est vide
- Pas de communication avec le *thread* UI
  ⇒ utiliser d'autres moyens (Toast, Notifications, etc)
- Solution pratique pour la plupart des cas

IntentService

- Dérive de Service, et ajoute la gestion des *threads*
  ⇒ il suffit d'implémenter le callbak onHandleIntent

- Utiliser Intent pour ajouter une tache à la file d'attente

- Exemple :

```
public class MyIntentService extends IntentService {
  // constructeur obligatoire qui doit appeler le parent
  public HelloIntentService() {
      super("MyIntentService");
  }
  @Override
  protected void onHandleIntent(Intent intent) {
    // faire le traitement
  }
}
```

Communication avec l'Activity

- IntentService n'a pas d'UI
- Alertes "envoyées" à l'utilisateur avec :
  - Toast
  - Notification (Status Bar)
  - Broadcast Manager

Tâches programmées

- AlarmManager pour lancer tâches :
    - intervalles réguliers
    - horaires donnés
- http://developer.android.com/training/scheduling/alarms.html

En bref

- Attention à ne pas charger la thread UI
- IntentService est suffisant dans la plupart des cas
- AlarmManager pour simplement programmer des tâches

Aller plus loin

- Guide des bonnes pratiques
  http://developer.android.com/training/best-background.html

- Services
  http://developer.android.com/guide/components/services.html