AsyncTask

Benjamin Lefebvre M2 E-Services FI
13/10/2017

Résumé

Les AsyncTask sont devenus incontournables pour les applications Android actuelles. Elles permettent de gérer correctement les problèmes de gestion des lourds traitements en les exécutant en parallèle du reste de l'application. Ainsi, elle évite le blocage de cette dernière.

Dans ce TP, nous verrons comment créer des AsyncTask, comment utiliser toutes ses composantes afin d'en tirer toutes les performances, et analyserons la pertinence de celle-ci.

Prérequis

- Savoir programmer en Android
- Comprendre le concept d'exécution en parallèle

Code source

Le code source de ce TP est disponible aux adresses suivantes :

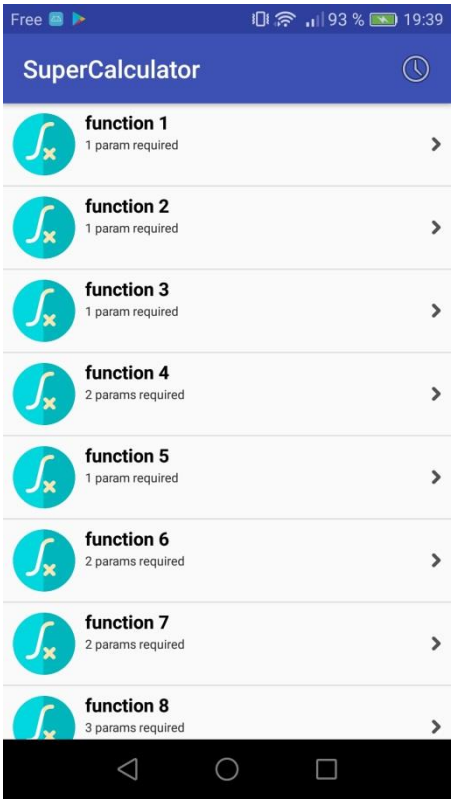
- Version initiale https://github.com/benji62f/AsyncTask-initial
- Version finale https://github.com/benji62f/AsyncTask

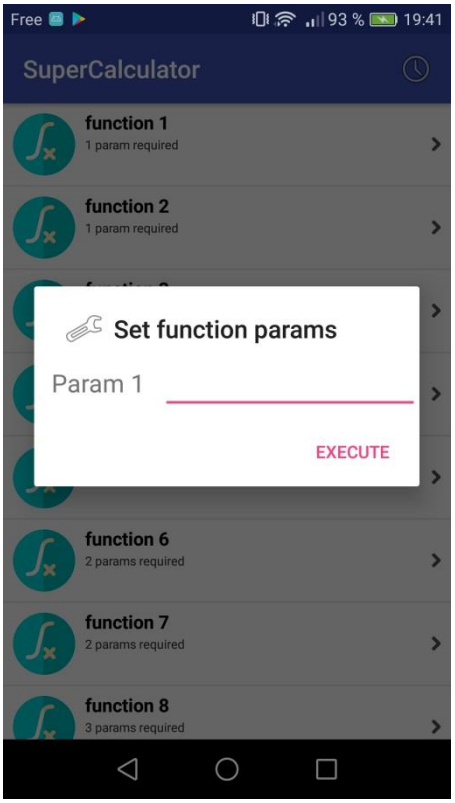
Explications du TP

Etape 1 : constater le problème sans AsyncTask

Installez puis lancez l'application « SuperCalculator » sur votre téléphone.

Lancez une fonction dans la liste, saisissez des paramètres (peu importe lesquels, c'est factice…), puis cliquez sur « Execute ».





Constatez le comportement de l'application.

Etape 2 : créer une AsyncTask

Pour l'heure, la simulation de calcul se fait de manière séquentielle, ce qui a pour effet de bloquer l'application et par conséquence de retirer la main de l'utilisateur. Pour y remédier, nous allons créer une AsyncTask pour que le traitement se fasse en parallèle.

Rendez-vous dans `com.lille1.bl.supercalculator.prgm.MathFunction`

L'AsyncTask étant déjà créée (classe interne : `private class ComplexeCalculation`), il n'est nécessaire que de l'instancier. Pour se faire, remplacez l'appel de la fonction par la ligne d'instanciation commentée dans le code.

Remplacez

```
// params contient les paramètres saisis lors de l'exécution de la
fonction de calcul
public void execute(double[] params){
    /**
     * AsyncTask
     */
    new ComplexeCalculation().execute(params);

    /**
     * Sequential code
     */
    makeAVeryComplexeCalculation(params);
}
```

Par

```
// params contient les paramètres saisis lors de l'exécution de la
fonction de calcul
public void execute(double[] params){
    /**
     * AsyncTask
     */
    new ComplexeCalculation().execute(params);

    /**
     * Sequential code
     */
    //makeAVeryComplexeCalculation(params);
}
```

➔ Le traitement se fait maintenant en parallèle (on observe qu'il n'y a plus de blocage), ce qui est une bonne chose. Mais, où est-donc le résultat ?

Etape 3 : comprendre le fonctionnement de l'AsyncTask

Tout d'abord, jetons un coup d'œil à l'AsyncTask en elle-même afin d'en comprendre son fonctionnement.

Nous avons appelé la méthode `execute`, et notre simulation de calcul s'est lancée (bon, effectivement, il n'y a pas vraiment eu de preuve de cela pour l'instant, mais ça va venir)… on peut donc en conclure qu'un appel est automatiquement fait à la méthode `doInBackground`.

```
private class MyTask extends AsyncTask<double[], Void, Double> {
    protected Double doInBackground(double[]... params) {

    }
}
```

Observez maintenant les 3 types d'objets qu'il y a entre les chevrons lors de l'héritage, à savoir `<double[], Void, Double>`.

Le premier : Souvenez-vous, `double[]` est le type de notre tableau de paramètres que nous avons passés à la méthode `execute` de notre AsyncTask.

Constatez l'utilisation de ce tableau dans les paramètres d'entrée de la méthode `doInBackground`.

Faites de même avec le troisième type, `Double`, qui lui est utilisé comme valeur de retour de cette même méthode.

Etape 4 : traiter le résultat de l'AsyncTask

Vous l'aurez compris, le type `Double` évoqué précédemment correspond au type de retour de la méthode `doInBackground` de notre AsyncTask. Mais alors, comment récupérer cette valeur ?

Dé-commentez au sein de l'AsyncTask la méthode `onPostExecute`.

```
protected void onPostExecute(Double result) {
    //...

    activity.getResults().add(new Result(getName(), this.params, result));
// records the result in the result list
}
```

Constatez que cette méthode prend en paramètre un `Double`, soit la valeur de retour de la méthode `doInBackground`.

Tout comme `doInBackground`, il n'y a pas besoin de l'appeler explicitement, tout est géré automatiquement une fois que l'invocation de `execute` est faite sur l'instance de l'AsyncTask. Ainsi, `onPostExecute` sera directement exécutée après la terminaison de `doInBackground`.

➔ Ici, nous utilisons `onPostExecute` pour enregistrer le résultat du calcul effectué, dans une liste. On peut très bien imaginer remplacer cela par un enregistrement en base de données.

Testez le bon fonctionnement de l'enregistrement, en lançant une fonction de calcul depuis l'application, puis en allant consulter l'historique des résultats après avoir attendu 10 secondes.

Etape 5 : ajouter une notification de démarrage

L'AsyncTask dispose également d'une méthode `onPreExecute`, qui sera exécutée automatiquement comme `onPostExecute`, à la différence que celle-ci le sera avant `doInBackground`.

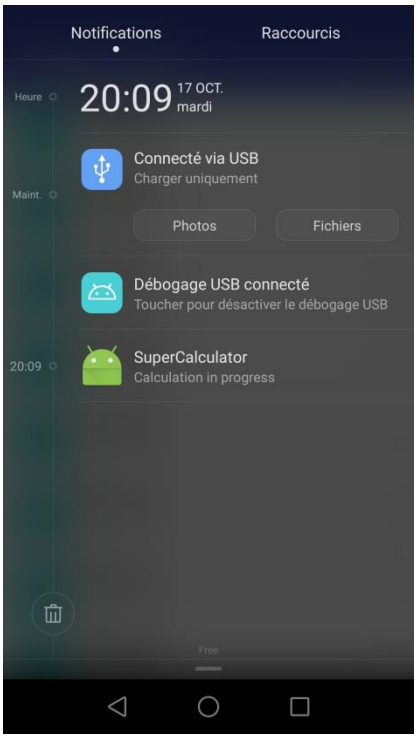
A l'intérieur, nous pouvons écrire un *feedback* pour signifier à l'utilisateur le démarrage de la tâche demandée.

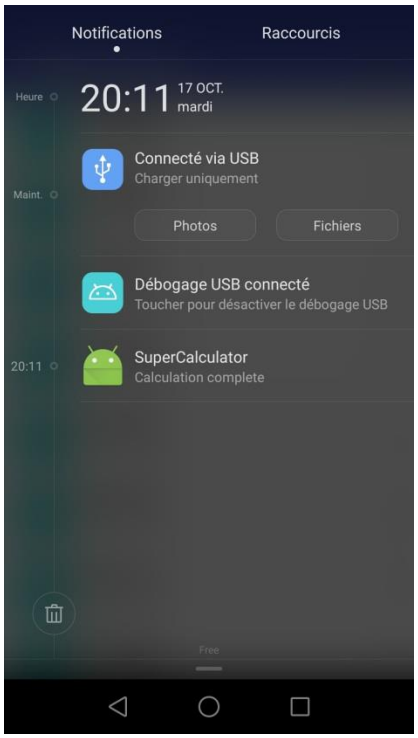
Dé-commentez `onPreExecute`, qui se chargera d'écrire une notification dans la barre des tâches du système Android.

```
protected void onPreExecute(){
    mNotifyManager =
            (NotificationManager)
activity.getApplicationContext().getSystemService(Context.NOTIFICATION_SERV
ICE);
    mBuilder = new
NotificationCompat.Builder(activity.getApplicationContext());
    mBuilder.setContentTitle("SuperCalculator")
            .setContentText("Calculation in progress")
            .setSmallIcon(android.R.drawable.ic_media_play);
    // Displays the progress bar for the first time.
    id = View.generateViewId();
    mNotifyManager.notify(id, mBuilder.build());
}
```

➔ A ce stade, la notification se lance bien, mais ne sert pas à grand-chose. En effet, elle n'indique jamais lorsque la tâche s'est terminée.

Dé-commentez alors le code restant dans `onPostExecute`, qui justement se charge de mettre à jour la notification une fois la tâche achevée.





Etape 6 : ajouter une barre de progression

Et pourquoi ne pas ajouter une barre de progression dans notre notification ? Cela permettra à l'utilisateur d'être informé en temps réel de l'état d'avancement de son AsyncTask.

Pour cela, on a la méthode `onProgressUpdate` ! Qui sera bien entendu appelée pendant le fonctionnement de l'AsyncTask, donc pendant que le `doInBackground` travaille.

Dé-commentez la méthode `onProgressUpdate`.

```
protected void onProgressUpdate(Integer... progress) {
    mBuilder.setProgress(100, progress[0], false)
            .setContentText(name + ": " + progress[0] + " %");
    mNotifyManager.notify(id, mBuilder.build());
}
```

Corrigez ceci :

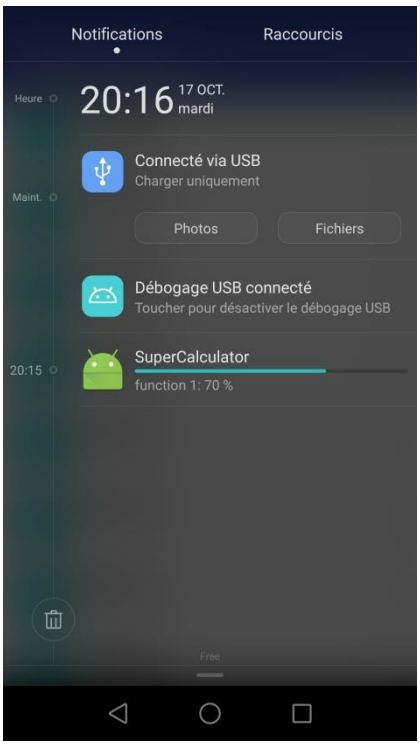
➔ A ce stade, vous devriez constater une erreur : Android Studio indique que cette méthode n'est jamais utilisée. Il s'agit d'une erreur de type… à vous de la réparer. Un indice, c'est par ici : `AsyncTask<double[], Void, Double>`.

Afin que la méthode `onProgressUpdate` puisse indiquer le pourcentage de progression, elle doit être elle-même au courant de ce dernier… pour ce faire, il faut le lui indiquer.

Dé-commentez cette ligne dans la méthode `doInBackground`.

```
publishProgress(i*10); // calls onProgressUpdate
```

Testez l'application et vérifier le bon fonctionnement de la notification.



Synthèse

Utilité

Une AsyncTask permet de traiter un calcul, un long traitement, un téléchargement, etc., de manière asynchrone c'est-à-dire de façon parallèle par rapport au processus de l'application. Cela permet à l'utilisateur de ne pas subir de blocage lors de ces traitements.

Mise en œuvre

Fonctionnement

Pour mettre en place une AsyncTask, il faut créer une classe héritant de la classe `AsyncTask`. Puis, il faut redéfinir les fonctions suivantes :

- `doInBackground` (obligatoire, contient le traitement à effectuer)
- `onPreExecute` (facultatif)
- `onProgressUpdate` (facultatif)
- `onPostExecute` (facultatif)

Pour démarrer une AsyncTask, il suffit d'appeler sa méthode `execute(…)`. Ce seul appel va déclencher tout le déroulement de l'AsyncTask automatiquement, dans l'ordre suivant :

- `onPreExecute` -> `doInBackground` -> `onPostExecute`
- `onProgressUpdate` doit être déclenchée dans `doInBackground` via la méthode `publishProgress(…)`

Typage

- Lors de l'héritage de la classe AsyncTask, il faut lui passer 3 types comme suit :

```
AsyncTask<param_doInBackground, param_onProgressUpdate, retour_doInBackground>
```

- Les AsyncTask ne travaillent qu'avec les objets (donc pas de types primitifs).

Informations complémentaires

- Documentation officielle : https://developer.android.com/reference/android/os/AsyncTask.html