TP:AsyncTask

Introduction

Qu'est-ce qu'une AsyncTask ?

une AsyncTask permet de réaliser des tâches de manière asynchrone, à la manière de la classe Thread. Cela permet d'effectuer un traitement en arrière plan sur une application Android sans ralentir la navigation, et de mettre à jour l'interface de l'application en fin de traitement.

Pourquoi utiliser une AsyncTask?

Lors du développement d'une application, il faut bien avoir en tête que toutes les tâches consommatrices de ressources (requêtes http, calculs lourds, …) doivent se faire dans un Thread séparé. En effet, le système affiche un message d'erreur et ferme l'application lorsque le Thread principal (appelé UI Thread) est bloqué trop longtemps. Donc on doit éviter de bloquer le UI thread. Et puis, on ne peut pas faire des opération sur composant d'UI dans l'autre thread sauf le UI thread.

Pour simplifier les opérations, Android 1.5 propose la classe "android.os.AsyncTask". Il convient idéalement pour les opérations de courtes (quelques secondes au plus.)

Comment utiliser une AsyncTask?

Une AsyncTask est définie par 3 types génériques

```
public abstract class AsyncTask<Params, Progress, Result>
```

- Params: le type des paramètres envoyés à la tâche lors de l'exécution.
- Progress:le type des unités progrès publié pendant le calcul de l'arrière-plan.
- Result:le type du résultat du calcul de l'arrière-plan.

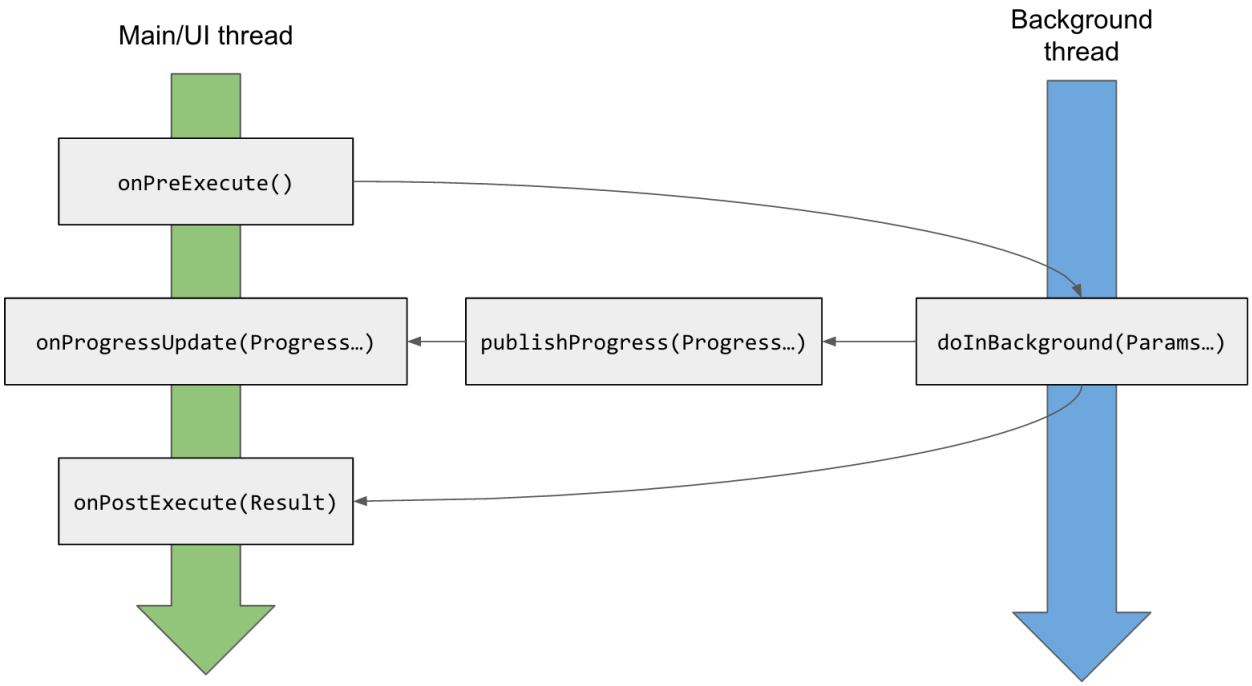
Lorsqu'une AsyncTask est exécutée, la tâche passe par 4 étapes :

- onPreExecute: appelé sur **le UI thread** avant que la tâche est exécutée. Cette étape est normalement utilisée pour configurer la tâche, par exemple en montrant une barre de progression dans l'interface utilisateur.

- doInBackground: appelée sur **le thread d'arrière-plan** immédiatement après que onPreExecute() fin de l'exécution. Cette étape sert à effectuer le calcul de fond qui peut prendre un certain temps. Les paramètres de la tâche asynchrone sont passés à cette étape. Le résultat du calcul doit être retourné par cette étape et va être retransmis à la dernière étape. Cette étape permet également publishProgress(Progress...) de publier une ou plusieurs unités de progrès. Ces valeurs sont publiés sur le thread d'interface Utilisateur, à l'étape de onProgressUpdate(Progress…).

- onProgressUpdate:appelé sur **le UI thread** après un appel à publishProgress(Progress...). Le moment de l'exécution n'est pas défini. Cette méthode est utilisée pour afficher toute forme de progrès dans l'interface utilisateur, tandis que le calcul de l'arrière-plan est en cours d'exécution. Par exemple, il peut être utilisé pour animer une barre de progression ou d'afficher les journaux dans un champ de texte.

- onPostExecute: appelée sur **le UI thread** après que le calcul de fond se termine. Le résultat du calcul fond est passé à cette étape comme un paramètre.



Annulation d'une tâche

Une tâche peut être annulée à tout moment en appelant cancel(boolean). Appel de cette méthode provoquera les appels suivants à isCancelled() retourner true. Après avoir appelé cette méthode, onCancelled(Object), au lieu de onPostExecute(Object) sera appelé après le retour de doInBackground(Object[]). Pour s'assurer qu'une tâche

est annulée dès que possible, vous devez toujours vérifier la valeur de retour de isCancelled() périodiquement de doInBackground(Object[]), si possible (à l'intérieur d'une boucle par exemple.)

Filetage des règles

- L'instance de tâche doit être créé sur le UI thread.
- Execute(params...) doit être appelée sur le UI thread.
- N'appelez pas onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manuellement.
- La tâche peut être exécutée qu'une seule fois (une exception sera levée si l'exécution d'une deuxième tentative.)

Objectif du TP

Le but de ce TP est d'apprendre à créer une AsyncTask. Nous allons donc d'abord créer un nouveau projet appelé *AsyncTask*. Nous allons faire en sorte d'effectuer un traitement long de manière asynchrone.

Pour obtenir la version initial de ce TP:

https://drive.google.com/drive/folders/0B17qz2T2rrSWUU45aVdFaUtsWTg

Pour obtenir la version finale de ce TP :

https://drive.google.com/drive/folders/0B17qz2T2rrSWdFlCaXpYc3lYVkU

Déroulement du TP

Commençons par créer un projet Android Studio de cette façon :
"Start a new Android Studio project" -> "Minimum SDK API 3: Android 1.5" -> "Empty Activity"

Etape 1:

Premièrement, nous allons modifier le layout **main.xml** en lui ajoutant un Button et une ProgressBar. Le premier servira à lancer le traitement ; la seconde à afficher la progression du traitement.

Etape 2:

Nous allons d'abord récupérer les composants définis dans le layout puis ajouter un listener sur le bouton afin qu'à chaque appui on exécute une nouvelle instance de BigCalcul.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // On récupère les composants de notre layout
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);
    mButton = (Button) findViewById(R.id.button);

    // On met un Listener sur le bouton
    mButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View arg0) {
            BigCalcul calcul = new BigCalcul();
            calcul.execute();
        }
    });
}
```

Etape 3:

Passons ensuite à l'écriture de notre classe BigCalcul, qui hérite d'AsyncTask.

```
private class BigCalcul extends AsyncTask<Void, Integer, Void>
{

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        Toast.makeText(getApplicationContext(), "Début du traitement asynchrone", Toast.LENGTH_LONG).show();
    }

    @Override
    protected void onProgressUpdate(Integer... values){
        super.onProgressUpdate(values);
        // Mise à jour de la ProgressBar
        mProgressBar.setProgress(values[0]);
    }

    @Override
    protected Void doInBackground(Void... arg0) {

        int progress;
        for (progress=0;progress<=100;progress++)
        {
            for (int i=0; i<100000000; i++){}
            //la méthode publishProgress met à jour l'interface en invoquant la méthode onProgressUpdate
            publishProgress(progress);
            progress++;
        }
        return null;
    }

    @Override
    protected void onPostExecute(Void result) {
        Toast.makeText(getApplicationContext(), "Le traitement asynchrone est terminé", Toast.LENGTH_LONG).show();
    }
}
```

Etape 4:
Nous ajoutons un bouton pour annuler cette tâche

```
stopButton = (Button) findViewById(R.id.stop_button);
stopButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(calcul.getStatus()== AsyncTask.Status.RUNNING && calcul!=null)
            calcul.cancel(true);
    }
});
```

```
protected Void doInBackground(Void... params) {
    for(int progress=0; progress<100; progress++){
        for(int i=0; i<100000000;i++){}
        if(isCancelled())
            return null;
        publishProgress(progress);
    }
    return null;
}
```

Fin du TP

C'est tout pour ce TP qui, je l'espère, vous a permis de comprendre comment créer une simple asyncTask. Cependant, il existe bien d'autres choses à faire avec la AsyncTask mais nous n'avons pas le temps de les aborder ici. Je vous invite alors à visiter ces liens pour en apprendre davantage sur la AsyncTask :

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