Android - Tâches asynchrones

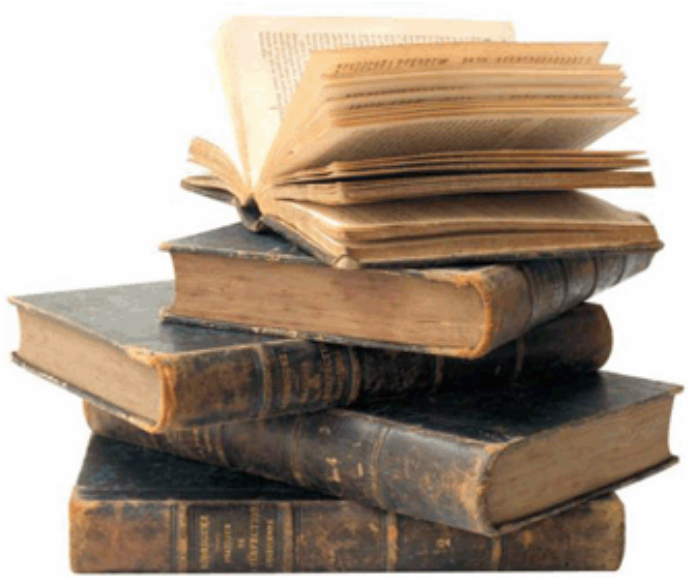
Jérémy S. Cochoy

INRIA Paris-Saclay | jeremy.cochoy@gmail.com

Decembre 2015

Sommaire

La documentation



Votre nouveau livre de chevet.

https ://developer.android.com/guide/index.html

AsyncTask

Pour effectuer une tâche asynchrone...

Pour exécuter une tâche de quelques secondes, on utilise la classe `AsyncTask` qui contiendra le code à exécuter en parallèle de l'activité en cours.

C'est une classe générique !

On hérite de `AsyncTask`

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long>
...
```

AsyncTask

Pour effectuer une tâche asynchrone...

Pour exécuter une tâche de quelques secondes, on utilise la classe `AsyncTask` qui contiendra le code à exécuter en parallèle de l'activité en cours.

C'est une classe générique !

On hérite de `AsyncTask`

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long>
...
```

AsyncTask

Pour effectuer une tâche asynchrone...

Pour exécuter une tâche de quelques secondes, on utilise la classe `AsyncTask` qui contiendra le code à exécuter en parallèle de l'activité en cours.

C'est une classe générique !

On hérite de `AsyncTask`

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long>
...
```

Types génériques

Les types utilisés par `AsyncTask` sont :

- Params - Le type des arguments passés au thread lors de son lancement.
- Progress - Le type de l'unité de mesure de la progression communiqué durant le calcul en arrière-plan.
- Result - Le type du résultat renvoyé après le calcul.

doInBackground()

Où placer la tâche couteuse ?

La tâche à effectuer en arrière-plan se place dans `doInBackground`.

Arguments et retour

Elle prend un(des) argument(s) de type *Params* et renvoie un résultat de type *Result*.

VarArgs

Le nombre d'arguments est variable. On les récupère sous la forme d'un tableau.

doInBackground()

Où placer la tâche couteuse ?

La tâche à effectuer en arrière-plan se place dans `doInBackground`.

Arguments et retour

Elle prend un(des) argument(s) de type *Params* et renvoie un résultat de type *Result*.

VarArgs

Le nombre d'arguments est variable. On les récupère sous la forme d'un tableau.

doInBackground()

Où placer la tâche couteuse ?

La tâche à effectuer en arrière-plan se place dans `doInBackground`.

Arguments et retour

Elle prend un(des) argument(s) de type *Params* et renvoie un résultat de type *Result*.

VarArgs

Le nombre d'arguments est variable. On les récupère sous la forme d'un tableau.

doInBackground()

La méthode isCancelled()

Lors de l'exécution de votre tâche, c'est à vous de réagir dans le cas où celle-ci est annulée dans l'activité principale via la méthode `cancel()`. Pour cela, regardez la valeur de retour de `isCancelled()`.

La méthode publishProgress

A chaque modification du niveau de progression de votre tâche, c'est à vous de communiquer cette information via `publishProgress()`.

doInBackground()

La méthode isCancelled()

Lors de l'exécution de votre tâche, c'est à vous de réagir dans le cas où celle-ci est annulée dans l'activité principale via la méthode `cancel()`. Pour cela, regardez la valeur de retour de `isCancelled()`.

La méthode publishProgress

A chaque modification du niveau de progression de votre tâche, c'est à vous de communiquer cette information via `publishProgress()`.

Exemple de doInBackground

```
protected Long doInBackground(URL... urls)
{
  int count = urls.length;
  long totalSize = 0;
  for (int i = 0; i < count; i++) {
    totalSize += Downloader.downloadFile(urls[i]);
    publishProgress((int) ((i / (float) count) * 100));
    // Fin de l'execution si cancel() est appelee.
    if (isCancelled()) break;
  }
  return totalSize;
}
```

onProgressUpdate()

La méthode onProgressUpdate()

C'est la méthode appelée lors d'une évolution de la progression. Elle s'exécute dans le thread de l'activité et peut donc accéder à ses variables membres.

Exemple de onProgressUpdate().

```
protected void onProgressUpdate(Integer... progress)
{
  mProgressBar.setProgress(progress[0]);
}
```

Pre-process

onPreExecute()

onPreExecute est exécutée avant le traitement de la tâche. Elle est appelée dans le thread de l'activité.

Exemple :

```
protected void onPreExecute()
{
  //Traitement a effectuer
}
```

Post-process

onPostExecute()

onPostExecute est exécutée après le traitement de la tâche. Elle est appelée dans le thread de l'activité.

Exemple :

```
protected void onPostExecute(Long result)
{
  showDialog("Downloaded " + result + " bytes");
}
```

Exécuter notre tâche

execute()

Pour exécuter notre tâche, on crée une instance de notre classe et on appelle la méthode execute().

Exemple :

```
private class DownloadFilesTask extends
                AsyncTask<URL, Integer, Long> { ... }

// Pour executer notre tache asynchrone :
new DownloadFilesTask().execute(url1, url2, url3);
```

Thread et Handler

Thread

La classe `Thread` permet l'exécution asynchrone d'une tâche pouvant se maintenir durant tout le temps de vie de l'application. On peut dériver `Thread` et surcharger `run()`, ou bien passer un objet de type `Runnable`.

Handler

La classe `Handler` permet la communication entre les threads (notamment entre le thread de l'activité et un de vos threads), via un système de queue de messages.

Executor

Pour gérer l'exécution de plusieurs tâches sans devoir explicitement créer les threads, jetez un œil à `Executor`.

ThreadPoolExecutor

Si vous avez un nombre important de tâches à exécuter, jetez un œil à `ThreadPoolExecutor`.

Conclusion

Les tâches asynchrones

Les tâches asynchrones vous permettent d'effectuer des opérations couteuses comme le téléchargement de fichiers, des requêtes à une base de données, des calculs couteux en performance, etc. sans provoquer le blocage de votre activité.

Pour me contacter : jeremy.cochoy@gmail.com, merci et à bientôt.