

Programación multimedia y dispositivos móviles

11

Tareas en segundo plano

IES Nervión
Miguel A. Casado Alías

Introducción

- En Android, por defecto, las aplicaciones se ejecutan en un solo hilo: Hilo principal / Hilo de la interfaz de usuario
- Si se ejecutan tareas costosas en dicho hilo, la interfaz de usuario puede dejar de responder de forma fluida a las interacciones de usuario
- Dichas tareas se deben ejecutar en hilos en segundo plano
 - Conexiones a internet
 - Accesos a BD
 - Escrituras en ficheros
 - Y en general cualquier tarea “no inmediata”
- Los componentes de la interfaz de usuario (vista) solo son accesibles desde el hilo principal

AsyncTask

- Facilita la creación de un hilo en segundo plano y la comunicación con el hilo principal
- Solo usar para tareas de unos pocos segundos como máximo
- Con la liberación de ViewModel y LiveData la comunicación con el hilo principal es muy sencilla, y además se tiene en cuenta el ciclo de vida, por lo que las AsyncTask pierden protagonismo
- Las AsyncTask lanzadas directamente desde una actividad pueden dar muchos problemas (relacionados con el ciclo de vida sobre todo), pero usadas en conjunción con ViewModel y LiveData pueden ser útiles y seguras
- Por defecto (desde API 11), todas las AsyncTask se ejecutan en el mismo hilo. Si queremos paralelismo entre ellas hay que usar el método **executeOnExecutor()**

Problemas de AsyncTask ([más info aquí](#))

- Si la actividad que lanza una AsyncTask se vuelve a crear por un cambio de configuración, el método `onPostExecute` de la tarea asíncrona no funcionará, pues perdió la referencia
- Si la actividad que lanza una AsyncTask es destruida, eso no implica que la tarea asíncrona sea destruida. La tenemos que destruir nosotros mismos para evitar fuga de memoria
- El método `cancel()` de la tarea asíncrona no destruye el hilo, sino que simplemente marca la tarea como “cancelada”, pero será el programador el que periódicamente tenga que comprobar si el estado es “cancelado” para parar la tarea en ese caso
- No se pueden ejecutar más de 128 tareas asíncronas simultáneamente, con una cola de 10. Si superamos la cifra de 138 AsyncTasks simultaneas la aplicación se cierra

AsyncTask: Tipos genéricos

- AsyncTask es una clase abstracta, por lo que para poder usarla debemos crear una subclase de ella
- Una AsyncTask usa tres tipos genéricos:
 - Params: El tipo de los parámetros que se le pasan a la tarea al ejecutarla
 - Progress: El tipo de los datos de progreso que queremos publicar durante la tarea en segundo plano
 - Result: El tipo del resultado de la tarea en segundo plano
- Si no usamos alguno de los tipos genéricos lo pondremos de tipo Void

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

Params

Progress

Result

Pasos en la ejecución de la AsyncTask

- `onPreExecute()` se ejecuta en el hilo principal justo antes de que se empiece a ejecutar la tarea en segundo plano
- `doInBackground(Params...)` es el único método que se ejecuta en un hilo en segundo plano y se usa para hacer la tarea deseada.
 - Se ejecuta justo después de `onPreExecute()`
 - Recibe como parámetro “Params” y devuelve “Result”
 - Puede hacer llamadas a `publishProgress(Progress...)`
- `onProgressUpdate(Progress...)` se ejecuta en el hilo principal cada vez que se llame a `publishProgress(Progress...)` desde `doInBackground(Params...)`. Sirve para mostrar progreso
- `onPostExecute(Result)` se ejecuta en el hilo principal justo después de que acabe `doInBackground`, y recibe como parámetro el dato devuelto por dicho método

Cancelar una AsyncTask

- Una tarea puede ser cancelada llamando a `cancel(true)`.
 - Devuelve false si la tarea ya se ha completado, o ya se había cancelado, o si no se puede cancelar por cualquier otra razón.
 - Devuelve true en los demás casos
- Deberemos comprobar periódicamente dentro del método `doInBackground(Params...)` si la llamada a `isCanceled()` devuelve true, en cuyo caso deberemos parar la tarea lo antes posible
- Si se ha llamado a `cancel(true)`, en vez de ejecutarse `onPostExecute(Result)` se ejecutará `onCancelled(Result)` cuando finalice el método `doInBackground(Params...)`

Algunas reglas a seguir

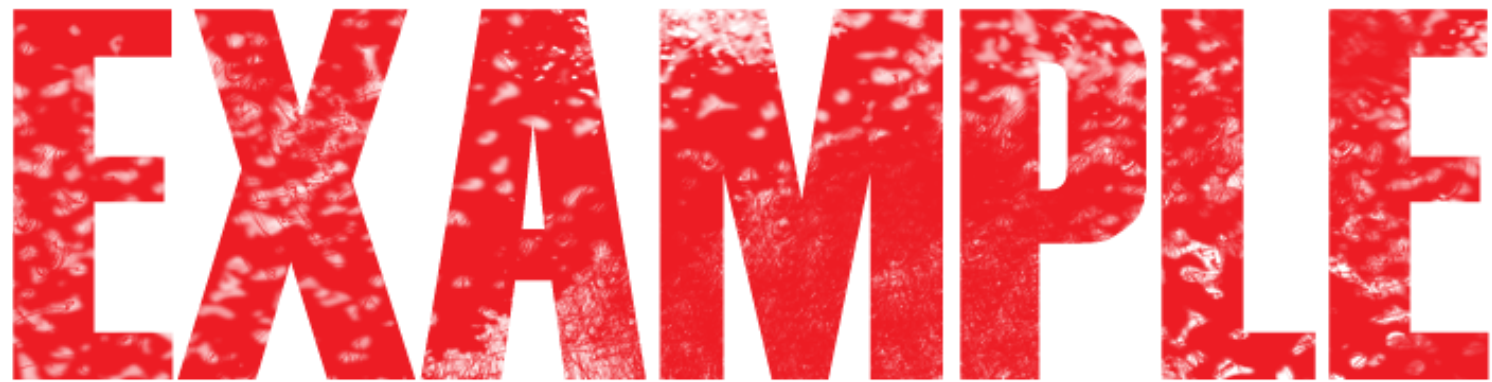
- Las AsyncTasks se deben instanciar en el hilo principal
- El método execute(Params...) debe llamarse desde el hilo principal
- Nunca llamar manualmente a los métodos onPreExecute(), doInBackground(Params...), onProgressUpdate(Progress...), onPostExecute(Result) ni onCancelled(Result)
- Cada AsyncTask solo se puede ejecutar una vez. Se lanzará un excepción si se intenta ejecutar más veces. Tendremos que crear otra instancia para volver a ejecutar la misma tarea otra vez.

AsyncTasks en paralelo

- Por defecto (desde API 11), todas las AsyncTasks se ejecutan en el mismo hilo
- Antes de API 11 (Honeycomb) se ejecutaban en paralelo.
- Si queremos paralelismo entre AsyncTasks en dispositivos con una versión de Android superior a la API 11 hay que usar el método **executeOnExecutor()**

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
    task.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
else
    task.execute();
```

Ejemplo de AsyncTask con ViewModel y LiveData



EXAMPLE

<https://medium.com/google-developers/lifecycle-aware-data-loading-with-android-architecture-components-f95484159de4>

