Android Essentials

AsyncTasks

Hilos en Android

FOLDER IT

Es importante para el usuario que las aplicaciones tengan tiempos de respuesta corto y respondan a la interacción del usuario de manera fluida.

Tenemos aplicaciones intensivas en CPU, pero no tenemos que descuidar la UX tampoco entonces ¿ qué hacemos ? lo mismo que en Java, usamos Hilos

Android tiene un Hilo principal, el Main Thread o UI Thread

Cada vez que lanzamos una aplicación, Android crea un nuevo proceso (al igual que en Linux) y lo ejecuta en su propia Máquina Virtual (Dalvik o ART según el API)

Hilos en Android

FOLDER IT

Android usa el mismo hilo para ejecutar todos los componentes de una app

Cuando invocamos uno de estos componentes se ejecuta sobre el hilo principal

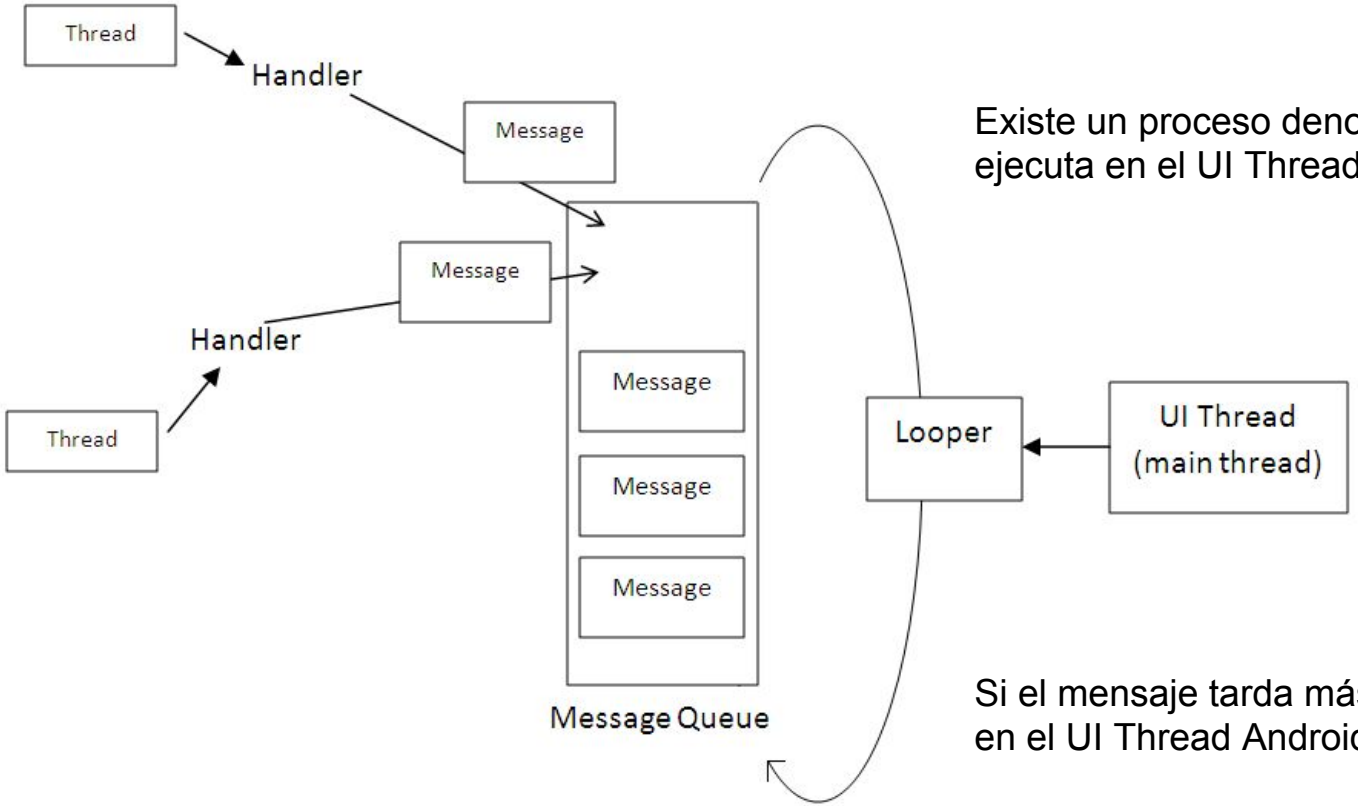
Este hilo es el encargado a su vez de actualizar la UI e interactuar con el usuario

No se deben ejecutar tareas de larga duración en el **UI Thread**, si no bloqueamos la UI.

Cualquier modificación de la UI se realiza mediante el envío de un mensaje a un cola

Android a medida que lo considera apropiado, desencola y procesa los mensajes

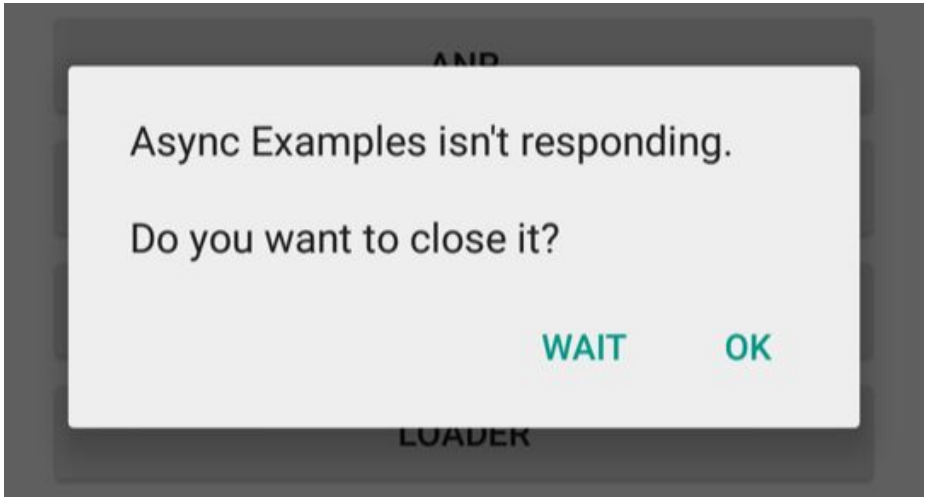
Hilos en Android

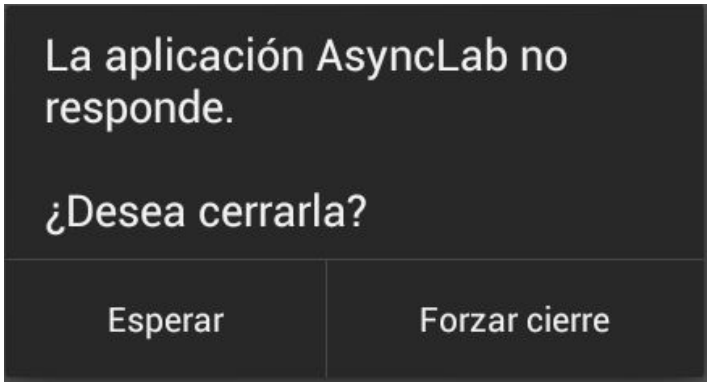


Existe un proceso denominado Looper el cual se ejecuta en el UI Thread

Si el mensaje tarda más de 5 segundo en procesar en el UI Thread Android lanzará un **ANR**

Application Not Responding (ANR)





Consideraciones

FOLDER IT

Para procesos de larga duración necesitamos mecanismos asincronicos

El Android UI Toolkit es “Not Thread-Safe”

Entonces no debemos

Bloquear el UI Thread

Acceder para leer o modificar la UI de Android desde un hilo que no sea el principal

Handlers

FOLDER IT

Para enviar un Mensaje a un Handler, en primer lugar se debe invocar el método obtainMessage() para obtener un objeto mensaje del pool de mensajes.
Para encolar objetos mensajes que contienen datos que serán procesados

```
sendEmptyMessage(int)
sendMessage(Message)
sendMessageAtTime(Message, long)
sendMessageDelayed(Message, long)
```

Para encolar objetos runnables

```
post(Runnable)
postAtTime(Runnable, long)
postDelayed(Runnable, long)
```

AsyncTasks

FOLDER IT

A partir del API 3

La Clase AsyncTask nos permite ejecutar instrucción en segundo plano y sincronizarlas con el hilo principal

A su vez notifica el progreso de las tareas ejecutando

Se debe usar para operaciones cortas en segundo plano que necesitan actualizar la UI

AsyncTasks

FOLDER IT

Que no debemos hacer cuando usamos AsyncTask

- Crear nuestros propios hilos en background
- Terminar hilos en background
- Invocar a métodos que permiten mandar mensajes u objetos Runnables al hilo principal de UI.

Debemos extender a `AsyncTask<Params, Progress, Result>`

AsyncTasks

FOLDER IT

Dado que este clase utiliza `Generics` se necesitan especificar 3 tipos de datos:

Params: El tipo de información que se necesita para procesar la tarea (por ejemplo una URL para descarga).

Progress: EL tipo de información que es pasada dentro de la tarea para indicar el progreso.

Result: El tipo de información que es pasada para post-procesar la tarea cuando la misma es completada.

AsyncTasks

FOLDER IT

`onPreExecute()` : Se ejecutará antes del código principal de nuestra tarea.

`abstract Result doInBackground(Params... params)` : Contendrá el código principal de nuestra tarea.

`onProgressUpdate()` : Se ejecutará cada vez que llamemos al método `publishProgress()` desde el método `doInBackground()`

`void onPostExecute(Result result)` : Se ejecutará cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método `doInBackground()`

`onCancelled()` : Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal

Cliente HTTP

FOLDER IT