HSBC

Android App Development using Kotlin

Infosys®
Navigate your next

Android App Development Using Kotlin

Background Processes in Android

Background Processes

1. Services
2. Threads and Handlers
3. AsyncTask class
4. Intent Service
5. RxJava / RxKotlin with RxAndroid

Threads and Processes in Android

- All the processing of UI components occur on “**main Thread”** or “**UI Thread**“.
- This thread is responsible for task such as rendering UI and dispatching events to the UI.
- All the components of an application run in the same **process** by the **main thread (UI Thread).**
- Every new component, by default, will run in the same thread. This is called “Single Thread Model”.

Disadvantage of Single Thread Model

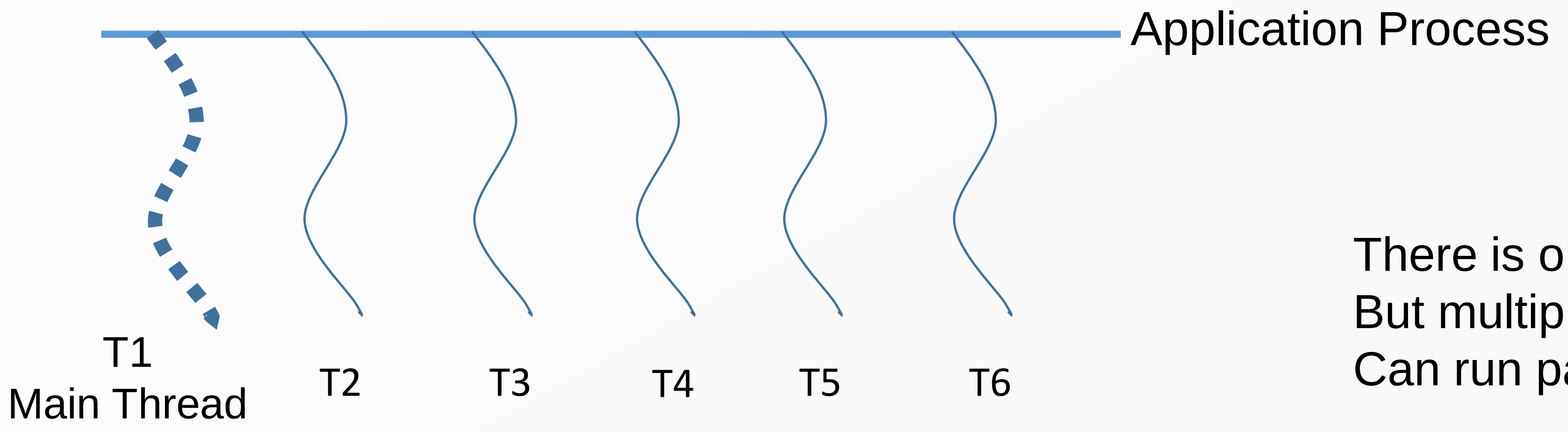
- Running various components by a single thread (UI/Main Thread) affects the performance and cause the application non responsive.
- To overcome the above disadvantage use Worker/Helper Thread
- **Worker thread(s)** runs in background in parallel with the “main” thread, making the main “UI” thread more responsive.
- This is a very common technique that you will do every day when developing Android apps.

Different ways of creating Helper/Worker Thread

1. Implementing Thread using the Java (Thread class or Implementing runnable interface) and update the UI using Handler
2. AsyncTask
3. IntentService(instead of Service class)
4. Using RxJava/RxAndroid with RxAndroid

Threads

- Thread is a processing resource which can execute a task using some CPU resources.
- In Android, each application can be thought of - one process.
- Each process, in turn has multiple threads running multiple different tasks.



There is only one Main thread,
But multiple worker threads
Can run parallelly in a process.

Threads

- Each process, however has **only one main thread** also known as **UI thread**. Main thread is dedicated to simple tasks and user interaction events only.

- Any task exceeding 5 seconds cannot be run on main thread. If done so, android system makes the app unresponsive and app stops running. Hence worker threads are needed in order to perform such heavy processing.

Demo on Using Threads in Android

Handlers

- As discussed earlier, long running background tasks must be run in a separate thread called Helper/ Worker thread.
- Traditional thread can run a Task in the background, but, it fails when the UI thread has to be updated from the same traditional thread.
- When tried to communicate directly with the main or UI thread, the android system throws an exception and does not allow the direct communication.
- In order to work around this problem, Handlers can be used along with threads.

Handlers (Cont.)

- Handlers maintain a queue known as MessageQueue, which enqueues all the requests in the form of messages.
- ‘Each’ message can be understood as a runnable task that can to be executed in the worker thread.
- Handler helps the worker threads to update the UI Thread forming a bridge between the two.

Demo on Handlers in Android

AsyncTask

- AsyncTask simplifies the operation of Threads and Handlers to make the code more readable and clear.
- Allows publishing results from background thread to UI thread.
- Should be used for shorter operations (at most few seconds).
- Example: network calls, I/O operations etc..
- It takes three generic parameters :
  - Params
  - Progress
  - Result

AsyncTask (Cont.)

- Params : data type of parameter array sent for execution
- Progress : data type of progress array published during execution
- Result : data type of result of background processing
- Void can be passed where a generic type is not used:

```
AsyncTask<URL, Void, Long>
```

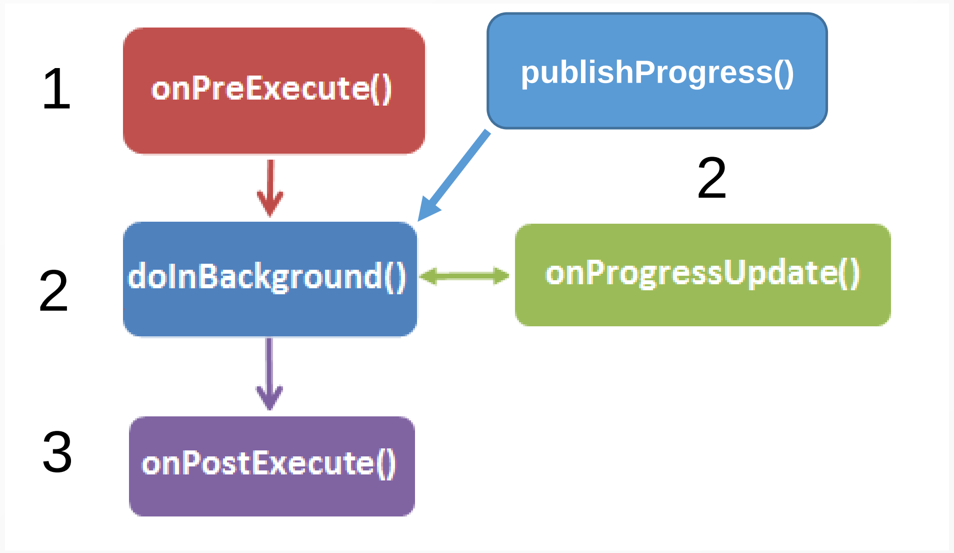
AsyncTask (Cont.)

- To execute an AsyncTask, an instance of AsyncTask is created and execute() method is called on the instance.
- The above method must be called only from the UI thread.
- A task can be executed only once.
- AsyncTask processes requests serially by default, only one thread will be used to service all requests.
- It is tightly bound to an Activity. If the Activity gets killed, the task will be stopped.

Sates of AsyncTask

States of AsyncTask operation:

1. PENDING :AsyncTask has been instantiated(execute() hasn't been called on the instance).
2. RUNNING : execute() has been called.
3. FINISHED



AsyncTask methods

- onPreExecute() : used to setup the task, before execution of the background task (invoked on the UI thread)

- doInBackground(Params...): Used to perform long running background computations, just after onPreExecute() finish its task (invoked on the background thread)

- onProgressUpdate(Progress...): Used to display any form of progress in the user interface while the background computation is still executing. The timing of the execution is undefined, but, after a call to publishProgress(Progress...). (invoked on the UI thread).

- onPostExecute(Result): The result of the background computation is passed to this step as a parameter, after the background computation finishes. (invoked on the UI thread)

AsyncTask

Steps:

1. **Create the layout**
2. **Create the AsyncTask**
3. **Invoke the AsyncTask inside the Activity**

Demo on AsyncTask in Android

Intent Service

- IntentService is an extended Service class which builds on the basics of the Service class.
- It serves the asynchronous service requests in the form of Intents.
- As the clients pass intents through startService(Intent) method call, intent service serves the intents only one at a time.
- Unlike traditional Service, IntentService runs the intents on a worker thread by default.
- Hence there is no need to create and maintain one by ourselves.

Intent Service

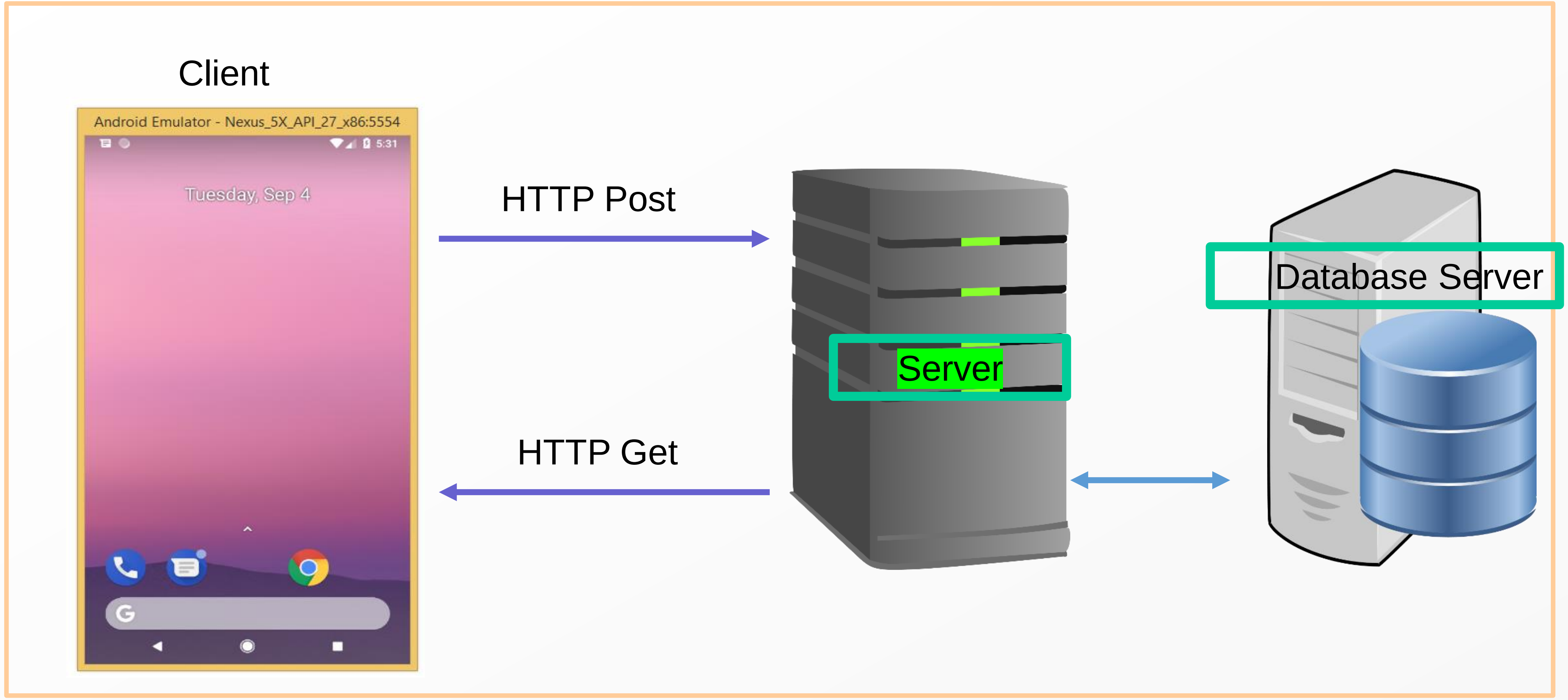
- Once all the intents are handled, the worker thread is automatically stopped. Hence there is no need to call stopSelf() method explicitly unlike in Service.

- The lifecycle of the intent service is very similar to that of Service except for one extra lifecycle method.

  - onCreate()
  - onStartCommand()
  - onDestroy()
  - **onHandleIntent(Intent) – Handles the intent and runs on worker thread**

External Data Handling

Interacting with External Data



Thank You