Android App Development

Background Tasks

Sabih Agbaria
Android Apps are event based
UI Thread

- By default the app has one thread that fetches events from event queue and executes the callbacks: UI Thread
- If one callback takes a few seconds to complete, the UI will stop updating and look to be freezing

⇒ Any long operation should be done outside of the UI Thread

⇒ Nothing should block UI Thread
Let’s see a “Bad” example

- Let’s do a long-running task in onClick
- run for 3 seconds: see how UI freezes, till operation ends
- run for 30 seconds: Android detects that UI thread is doing too much work and gives the user a chance to kill app
- User uninstalls the app
Not all long-running tasks are equal

- Long running tasks examples:
  - Computations: image processing, next AI move
  - Networking: downloads, http requests, ..
  - Music playing.

- Some are expected to finish in a few seconds, some live as long as the users is using the app
Android supports Java multithreading

- Android supports standard Java classes like Thread and Runnable

- But also Android supplies a higher-level threading models, that fits better with the android framework.

- When possible use Android solutions
Meet AsyncTask

- Hides the creation of a new thread, and simplifies the communication between it and the UI Thread.
- Ideal for operations that take a few seconds: for example network access.

AsyncTask runs in stages:
1. onPreExecute: code to run in UI Thread before the task.
2. doInBackground: the heavy computation
3. onProgressUpdate: report progress, runs in UI Thread
4. onPostExecute: runs in UI Thread

Pre,Post and update are important because only the UI thread can touch Views.
Creating and running AsyncTask

1. Extend the generic AsyncTask class
2. Give type parameters: input, progress and output type
   
   ```java
   AsyncTask<Params, Progress, Result>
   ```
   
   a. **Params** - the type that is passed into the `execute()` method.
   b. **Progress** - the type that is used within the task to track progress.
   c. **Result** - the type that is returned by `doInBackground()`.
3. override methods
4. whenever needed instantiate a task and execute it
   
   ```java
   new ComplexTask().execute(param1,param2);
   ```
Let’s fix the bad example

- Run a 20 seconds task, see how UI is still responsive
- Always show the user some indication that a background task is running
- Make sure AsyncTask is parametrized to make it reusable
Simultaneous AsyncTask

- How many AsyncTasks can run simultaneously?
- The behaviour of execute is version dependent, earlier version allowed one thread, later versions allowed unlimited number
- Recent android versions give you two options:
  - `executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR)`
  - `executeOnExecutor(AsyncTask.SERIAL_EXECUTOR)`
- **THREAD_POOL_EXECUTOR** is a pool of threads that executes the given tasks, the size of the pool is as the number of cpu cores
- **SERIAL_EXECUTOR** is useful if tasks depend on each other in linear way
How to schedule a task for the future

- We want to accomplish some task sometime in the future
- We cannot use AsyncTask because there is no guarantee that the app will remain alive till that point

- This seemingly easy problem requires collaboration between three objects:
  - Service
  - BroadcastReciever
  - AlarmManager
AlarmManager

- It’s a global service provided by android, that allows apps to schedule code execution
- Intended for tasks that is independent of app running state: background or not running
- Can schedule one-time or repeating alarms
- When the alarms fires, AlarmManager broadcasts the registered intent
**BroadcastReceiver**

- It’s one of the four types of application components
- Like activities it should be defined in the manifest file
- Each BroadcastReceiver tells what type of Intents it wants to receive:
  - For example our app can respond to low battery event, by declaring a BroadcastReceiver for this intent.
- To create a broadcastreceiver extend BroadcastReceiver and override onReceive() method
Service

- It’s one of the four types of application components
- Like activities it should be defined in the manifest file
- It executes code that outlives the App (unlike AsyncTask)
- To create a service extend Service or IntentService
The plan

Schedule

Broadcast an Intent

Framework

Our App
Let's implement the plan

- Schedule some code to run in future
- Run code inside a service
Notes

- Alarms are not guaranteed to be delivered on exact time, but can be aggregated to save battery time.
- Alarms do not survive a device reboot!
- WakefulBroadcastReceiver has the ability to wake up the device and requires a special permission (see code)
- IntentService runs in a worker thread, normal service runs in Main thread
Status bar notification

- Notifications are great way to get attention of the user while app is not running or in the background
- For example:
  - show user some new available data
  - notify user of score of his favorite sports team

- Do NOT annoy your users!
- Status Bar notifications can ask the user to do some action
- In many cases the user touches the notification and expects your app to show him the relevant information
- For example the chat thread
Let's notify the user when the algorithms finishes