ANDROID - Curs 5

Topics:

- Services
- JobScheduler
- AlarmManager
- WorkManager

Background Processing

Every Android app has a main thread which is in charge of handling UI (including measuring and drawing views), coordinating user interactions, and receiving lifecycle events.

Any long-running computations and operations such as decoding a bitmap, accessing the disk, or performing network requests should be done on a separate background thread.

Applications may also require some tasks to run even when the user is not actively using the app such as syncing periodically with a backend server or fetching new content within an app on a periodic basis.

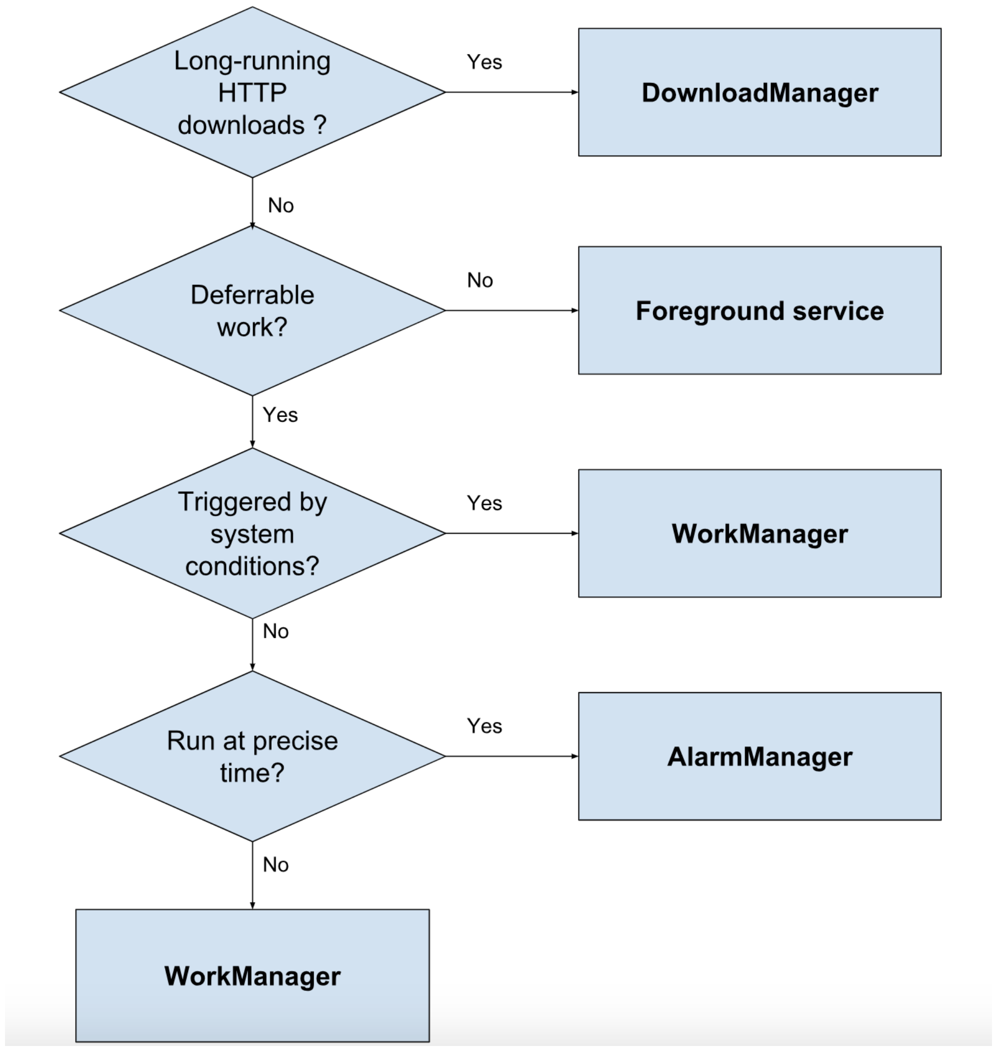
Applications may also require services to run immediately to completion even after the user has completed interacting with the app.

The right solution

? Can the work be deferred, or does it need to happen right away?

? Is the work dependent on system conditions?

? Does the job need to run at a precise time?



Services

A Service is an application component that can perform long-running operations in the background, and it doesn't provide a user interface.

Another application component can start a service, and it continues to run in the background even if the user switches to another application.

A service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Service Types

- **Foreground**
    - A foreground service performs some operation that is noticeable to the user.
    - For example: an audio app would use a foreground service to play an audio track.

- **Background**
    - A background service performs an operation that isn't directly noticed by the user.
    - For example: if an app used a service to compact its storage, that would usually be a background service.

- **Bound**
    - A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication.
    - For example: a music player might find it useful to allow its service to run indefinitely and also provide binding.

Using a foreground service tells the system that the app is doing something important and it shouldn't be killed.

Foreground services are visible to users via a non-dismissible notification in the notification tray.

Implementation Example

- https://developer.android.com/guide/components/services
- https://www.vogella.com/tutorials/AndroidServices/article.html#exercise-define-and-consume-local-service

Job Scheduler

The Android 5.0 Lollipop (API 21) release introduces a job scheduler API via the JobScheduler class.

This API allows to batch jobs when the device has more resources available. In general this API can be used to schedule everything that is not time critical for the user.

Advantages

- Supports batch scheduling of jobs;
- The Android system can combine jobs so that battery consumption is reduced;
- JobManager makes handling uploads easier as it handles automatically the unreliability of the network;
- Survives application restarts.

Examples of use

- Tasks that should be done once the device is connect to a power supply;
- Tasks that require network access or a Wi-Fi connection;
- Task that are not critical or user facing;
- Tasks that should be running on a regular basis as batch where the timing is not critical.

Job Scheduler + Job Service

- **Create a Job** - A unit of work is encapsulated by a **JobInfo** object. This object specifies the scheduling criteria.
- You will construct these JobInfo objects and pass them to the **JobScheduler**
- To implement a Job, extend the **JobService** class and implement the onStartJob and onStopJob.
- When the criteria declared are met, the system will execute this job on your application's **JobService**

Implementation example:

- https://www.vogella.com/tutorials/AndroidTaskScheduling/article.html#the-job-scheduler-api

Alarm Manager

This class provides access to the system alarm services.

These allow you to schedule your application to be run at some point in the future, to run a job at a **precise time**.

Registered alarms are retained while the device is asleep (and can optionally **wake the device** up if they go off during that time), but will be *cleared* if it is turned off and rebooted.

Setup

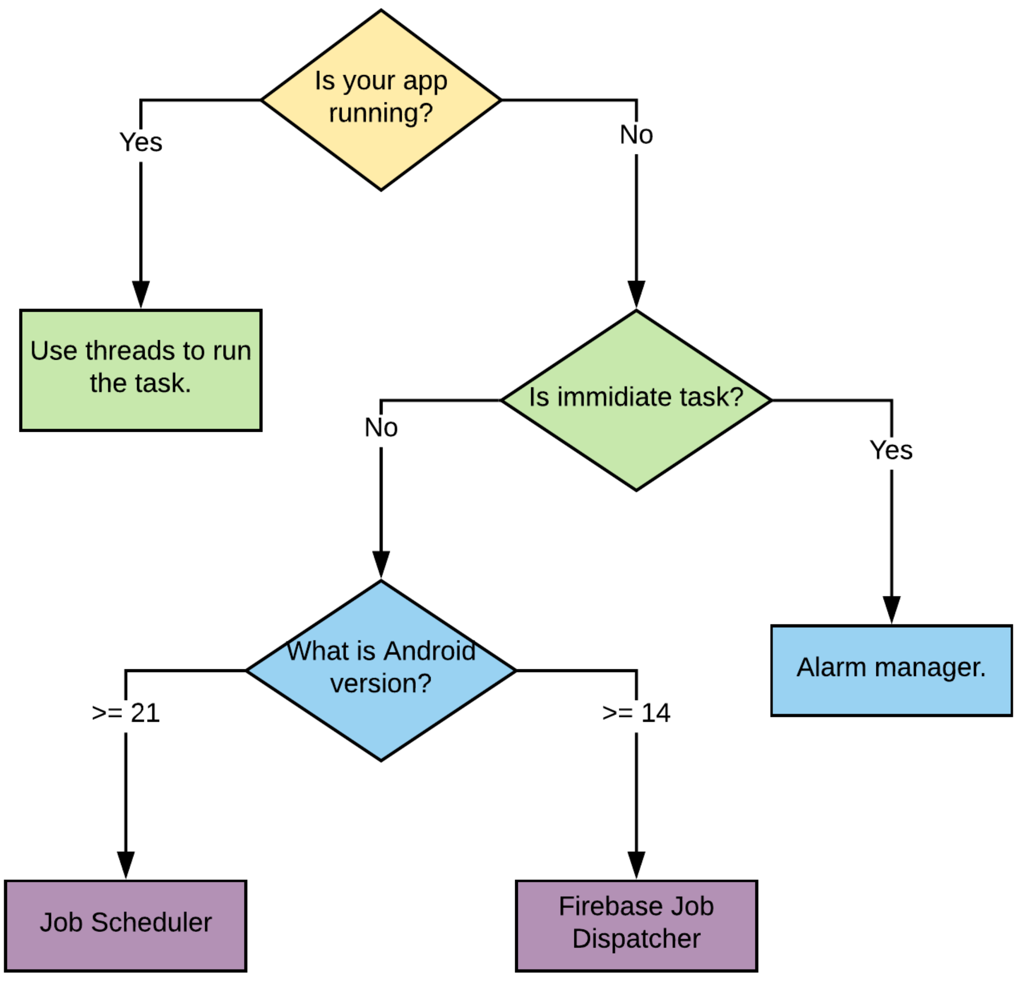
1. Create an AlarmManager instance;
2. Create a Broadcast Receiver;
3. Create a PendingIntent;
    - gives AlarmManager the permission to trigger the wrapped intent as if it is being triggered from your own app(Even if your application is killed)

1. Choosing the Alarm Types;
2. Setting the Alarm.

Implementation Example

- https://androidclarified.com/android-example-alarm-manager-complete-working/
- https://codelabs.developers.google.com/codelabs/android-training-alarm-manager/#0

Work Manager



- **Easy to schedule**
  - The WorkManager API makes it easy to create deferrable, asynchronous tasks and also allows you to specify when they should run.
  - You can create a task and hand over that task to the work manager to run it immediately or whenever the device is under specific conditions (like “only while the device is charging and online”).

- Work manager provides guarantee that your task will run when specific conditions match, even if your app is force-quit or the device is rebooted.

- **Easy to cancel**

WorkManager assigns UUID to the each work/task you schedule. Using this unique id, you can easily cancel any task at any time.

- **Easy to query**

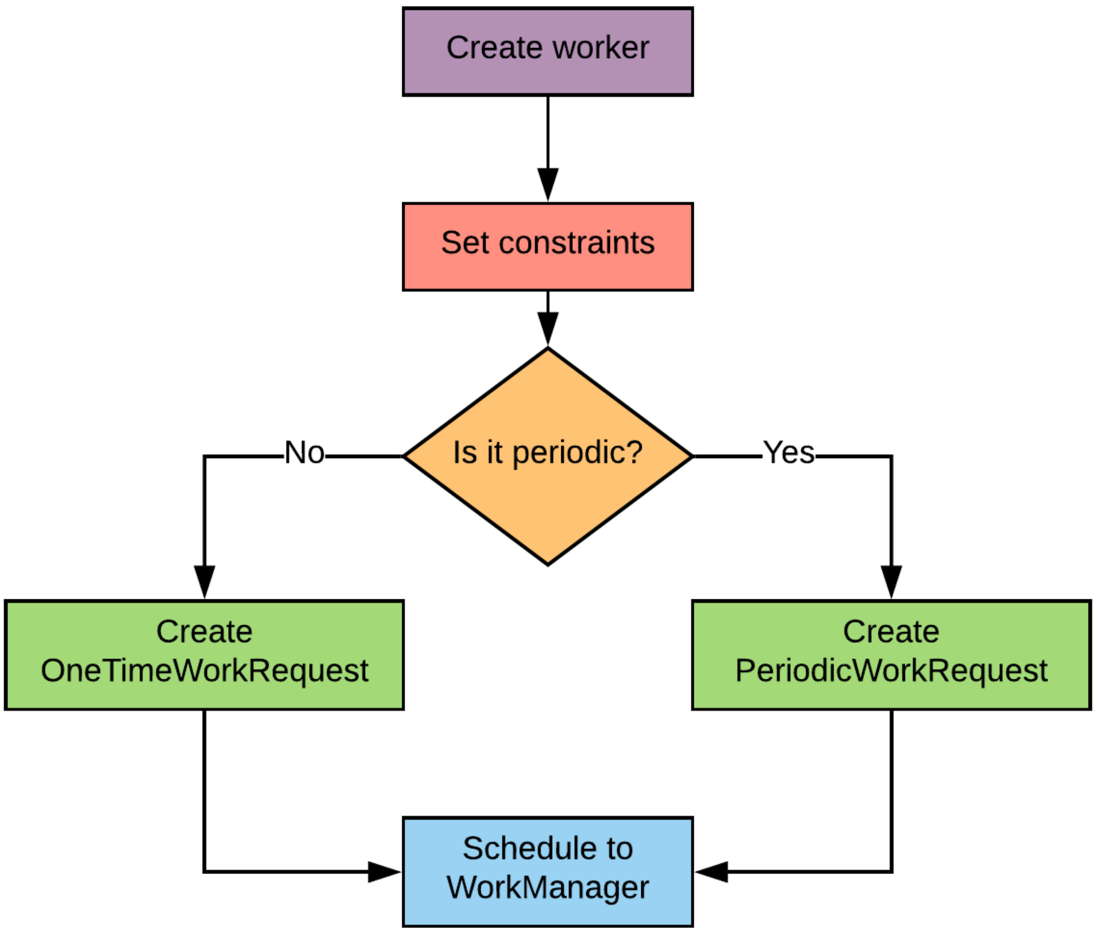
  - You can ask for the status of the task—whether it is running, pending or finished—by using the unique id assigned to each task.
  - Work manager APIs goes beyond only current state of the task and allows tasks to return data in key-value format.
  - Work manager uses LiveData to return the data and state of the work. So, your activity can observe this LiveData and it will get notified whenever the task is finished.

- **Support for all android versions**

  - Work manager APIs supports API 14 or above.
  - WorkManager chooses the appropriate way to run your task based on such factors as the device API level and the app state.
  - If the application is running, work manager APIs will create new thread to run the task.
  - If the application process is not running, it will use JobScheduler APIs or Firebase Job dispatcher or Alarm manager API to run the scheduled task.

Setup

1. Create the work;
2. Define constraints;
3. Create the work request;
4. Schedule the request.



Implementation Example:

- https://medium.com/@kevalpatel2106/exploring-jetpack-scheduling-tasks-with-work-manager-fba20d7c69bf