Mobile Applications with Android

Running in the background: Services

Joscha Behrmann <behrmann@hm.edu>
Salman Alhadziev <salman.alhadziev0@hm.edu>

What is a service in the context of app-development for Android?

"A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use." [1]

Following this definition, services are widely used in almost every popular application. Services are typically applied to perform any sort of operation in the background, that might possibly block the flow of execution. This includes common I/O-operations such as operating on a file or network-communications with a backend server. Services might also be used to offer functionality that is frequently required by different applications as services can be exported. That way they are accessible not only by the originating app but every other application.

How do I declare and start a service?

First you have to declare the service in your applications manifest-file. Now create your custom service class which has to extend the Service-class which is provided by the Android-SDK. You should now override the service-lifecycle events that you want to use and add your custom code. Finally, start your service by either invoking *startService()* or *bindService()*.

Why can't I just use threads instead of services?

Although the use-case of services and threads seem similar, e.g. the off-loading of possibly blocking or cpu-time consuming tasks there is one crucial difference: *services can run in the background*. This means that services can run even though the application is not currently being interacted with by the user. You should use a thread if there is a tight coupling with the work to be done within the UI, e.g. the UI shows an image that can be edited. The processing for the edit-functionality should be run in a thread. The thread should follow the lifecycle of the UI-activity. A service runs in the background and is only loosely connected to the UI or not at all used by the UI. A service mostly shouldn't care about the current state of the UI.

Can I use services in combination with threads?

Absolutely. You will have to handle the creation, synchronization and stopping of the thread on your own. Make use of your services lifecycle-events to implement this. Beware that you can only modify your UI from within the applications main-thread. If you want to change UI-state from within your service, you can interact with the main-threads message-queue by invoking *post()* on its handler. Take note that the Android-SDK comes with a plethora of concrete service-implementations that might also cover your use-case. For example, the *IntentService*-class will make use of a thread to do its job, so you wouldn't have to implement the threading-behaviour on your own. [2]

What does the lifecycle of a service look like?

The lifecycle-states of your service depend on whether it is used as a bound- or unbound-service. [Figure 1] shows the most useful lifecycle-methods of an unbounded service. Note that there are a couple more callbacks that you can implement. Have a look at the abstract *Service*-class which is the base-class for all services to see what other callbacks you might want to override. [3]

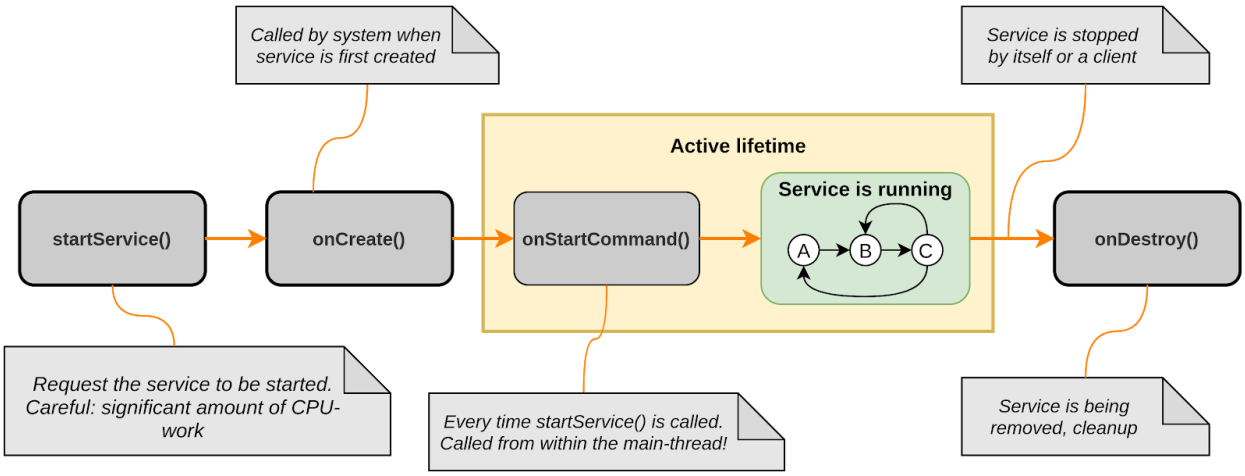

Figure 1: The lifecycle-states of an unbound service.

What's the typical use-case for a service?

As mentioned, services should be used whenever you have to do some processing in the background, e.g. when the user is not interacting with the application. Bounded services are a special form of service that follow a client-server architecture with the service being the server and applications being the client. This way not only the issuing application can use the service but in theory *any* application can access the service and it's interface. A good example would be

the Google Play Service which can be used to access Google-powered features such as Google-maps. [4] So if you want to use some Google-feature in your app, you'll have to query the Google Play Service from within your application.

What types of services are there?

There are three different types of services, background, foreground, and bound, each having their own use cases. Background and foreground can also be classified as "started" services, since they have an identical lifecycle in contrast to the bound service and can be started using the startService() call. A service can also be both started and bound, which is determined by the implemented callback methods (onStartCommand(), onBind()). So to declare a bound service you will have to implement the onBind()-method of the service. If you don't want to support binding, this method should return *null*.

Background Service

The background service can be used to perform operations in the background that are not noticeable to the user. For example, when the user is not interacting with the app, it can compact its storage in the background. The background service can be started by other components like an activity using the startService() function. Once it is started, it runs indefinitely until it is stopped by either itself using stopSelf() inside the service or another component using stopService() from outside [5]. Long-running jobs in the background are likely to be terminated by the system when the memory is low. The system lowers the position of long-running jobs in the list of background tasks over time. A service can be configured to be restarted after it is terminated and the system will restart it as soon as resources are available. The returning value in the onStartCommand() determines how the system should treat the service after it is terminated, e.g. recreate or not recreate the service [6].

Foreground Service

In contrast to the background service, a foreground service performs operations that are noticeable to the user, for example playing music in the background. The user can play a song, close the app, and open a different app while the song is still playing in the background. This approach is suitable for use cases, where the work needs to be done immediately and needs to be executed to completion. In the case of the music player, the user is in charge of when to start and stop the song and does not want the system to randomly stop the music. Therefore a foreground process is rarely killed by the system because the user would notice it and the experience would suffer. The system tries its best to keep it alive. A foreground service *must* display a notification in the notification bar. The user can then start the app from the notification or perform other actions like changing or pausing the song.

Bound Service

A bound service is a special form of service, which provides a client-server interface to other application components and allows them to interact with the service, send requests, receive results, and perform inter-process communication (IPC). They might offer features that are useful to different applications so that these applications can consume their functionality through means of IPC. A bound service allows multiple components to bind to it at one using the bindService() call. The service runs as long as another component is bound to it and does not run in the background indefinitely. It must implement the onBind() callback and return an IBinder, which defines the interface for communication with the service.

Summary

Services are used to perform long-running work in the background. They should be used, if work needs to be executed in the background while the user is not interacting with the app. In the case of *bounded*-services they can also be used to provide a global access-point for some common functionality. Since normally a service, when started, will be executed in the applications main-thread, you should be careful to not process any tasks that might take a long time (*10ms is already a very long time*). You might want to make use of threads from within your service to obtain concurrent execution and thus make use of your hardwares capabilities. A service has no user interface (except foreground-services) so interaction with them is done programmatically and not directly by the user. There are three types of services, background, foreground, and bound, which have their own lifecycle and use cases.

Where can I learn more about services?

Google offers guides and exhaustive references for the Android operating system. To get a feel for background-processing you can start with [7]. This guide also includes an overview of the several service-topics. While developing the references, e.g. [2] and [3], are a really helpful and *up-to-date* source of information. The web is full of android-specific resources, however they might not always represent the current state of the art so be careful to not rely on very old sources as Android is moving *fast*.

References

[1] https://developer.android.com/guide/components/services
[2] https://developer.android.com/reference/android/app/IntentService
[3] https://developer.android.com/reference/android/app/Service
[4] https://developers.google.com/android/guides/overview
[5] https://proandroiddev.com/deep-dive-into-android-services-4830b8c9a09
[6] https://www.big-app.de/alles-zu-services-unter-android/
[7] https://developer.android.com/training/best-background