Services

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Services

- A *service* is a component which runs in the background, without direct interaction with the user.
- It is faceless components that can run in the background e.g. music player, network download etc
- A service can allow other components to bind to it, in order to interact with it and perform interprocess communication
- A service runs in the main thread of the application that hosts it, by default

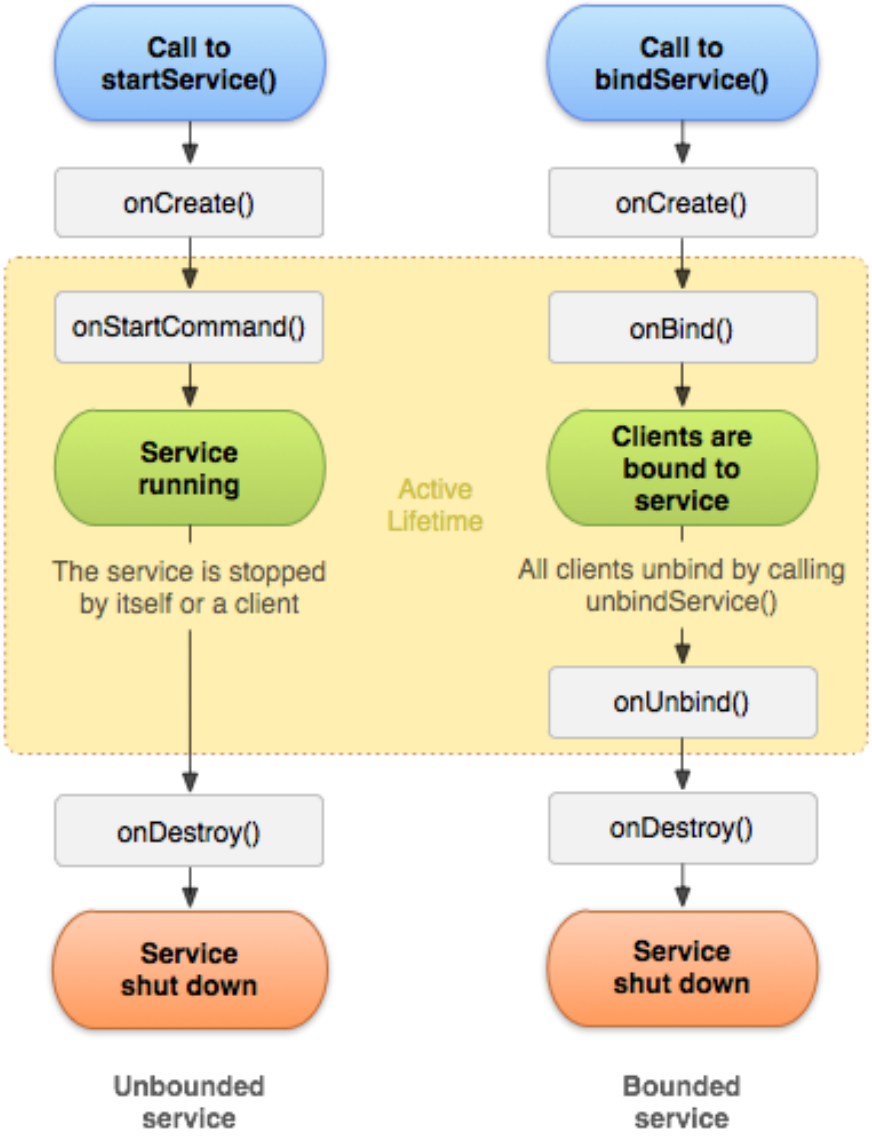
Benefits of using services

- If you use asynchronous processing in *activities* or *fragments* the corresponding threads are still connected to the life-cycle of the corresponding *activity*. The Android system may decide to terminate them at any point in time.
- *Services* run with a higher priority than inactive or invisible *activities* and therefore it is less likely that the Android system terminates them.
- Defining your own *services* allows you to design very responsive applications. You can fetch the data via a *service* and once the application is started by the user, it can present fresh data to the user.

Services – Forms

- Services may take two forms:
- **A started service:** The service is created when another component calls startService(). The service then runs indefinitely and must stop itself by calling stopSelf(). Another component can also stop the service by calling stopService(). When the service is stopped, the system destroys it
- **A bound service** The service is created when another component (a client) calls bindService(). The client then communicates with the service through an IBinder interface. The client can close the connection by calling unbindService(). Multiple clients can bind to the same service and when all of them unbind, the system destroys the service. (The service does *not* need to stop itself.)

Service Life Cycle



Defining new services

- Every Android application can define and start new *services*
- To create a service, a subclass of service should be created
- In implementation, callback methods should be implemented(onCreate, onStartCommand, onDestroy, onBind)

Services Handling After being Killed

- This depends on the return value from onStartCommand(), which must be one of the following constants:

- **START_NOT_STICKY** If the system kills the service after onStartCommand() returns, *do not* recreate the service, unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.
- **START_STICKY** If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand(), but *do not* redeliver the last intent. Instead, the system calls onStartCommand() with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands, but running indefinitely and waiting for a job.
- **START_REDELIVER_INTENT** If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand() with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.

Services in Manifest file

- All services must be represented by <service> elements in the manifest file (AndroidManifest.xml)
- Required Attributes: android:name

```
<service
android:name="MyService“
 android:icon="@drawable/icon“
 android:label="@string/service_name" >
</service>
```

Service Implementation

```
public class MyService extends Service {
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
//TODO do something useful
      return Service.START_NOT_STICKY;
}
 @Override public IBinder onBind(Intent intent) {
//TODO for communication return IBinder implementation
      return null;
}
 }
```

Starting a Service

- An Android component (service, receiver, activity) can start and trigger a service via the startService(intent) method. This method call starts the service if it is not running.

```
Intent service = new Intent(context, MyService.class);
context.startService(service);
```

- This call starts the services by calling onCreate() method and onStartCommand in sequence
- If startService(intent) is called while the service is running, its onStartCommand() is also called. Therefore your service needs to be prepared that onStartCommand() can be called several times.

Stopping a Service

- A service can be stopped by using the stopService() method.

- A service can stop itself by calling the stopSelf() method.

Services Mp3 Demo

- ***Play mp3 Services Demo:-***

IntentServices

- The IntentService class is available as a standard implementation of Service that **has its own thread** where it schedules its work to be done.
- Once the service is done, the instance of IntentService terminate itself automatically. Examples for its usage would be to download a certain resources from the Internet.
- The IntentService class offers the onHandleIntent() method which will be asynchronously called by the Android system.
- To create an IntentService component for your app, define a class that extends IntentService, and within it, define a method that overrides onHandleIntent()

Bound Services

- **IBinder:** Base Interface for a remotable object. This interface describes the abstract protocol for interacting with a remotable object
- **Binder:** Base class for a remotable object, the core part of a lightweight remote procedure call mechanism defined by IBinder. This class is an implementation of IBinder that provides the standard support creating a local implementation of such an object.

Bound Services

- For creating a services that supports binding must return an implementation of IBinder. There are three ways to define this interface
- Extending the Binder Class
- Using a Messenger
- Using AIDL

Bound Services – Extending Binder Class

- Create an interface by extending Binder Class
- Return an instance of it from onBind
- The client receives the Binder and can use it to directly access public methods available in either the Binder implementation or even the Service

Binding a Service

- Services also can be started via the bindService() method call. This allows to communicate directly with the service.

```
Intent intent = new Intent(this, LocalService.class);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

- The first parameter of bindService() is an Intent that explicitly names the service to bind
- The second parameter is the ServiceConnection object.
- The third parameter is a flag indicating options for the binding. It should usually be BIND_AUTO_CREATE in order to create the service if its not already alive. Other possible values are BIND_DEBUG_UNBIND and BIND_NOT_FOREGROUND, or 0 for none.

Binding a Service

- To bind to a service from client, client must:
- Implement ServiceConnection, this implementation must override two callback methods:
- onServiceConnected() : The system calls this to deliver the IBinder returned by the service's onBind() method.
- onServiceDisconnected() : The Android system calls this when the connection to the service is unexpectedly lost, such as when the service has crashed or has been killed. This is *not* called when the client unbinds.
- Call bindService(), passing the ServiceConnection implementation.
- When the system calls your onServiceConnected() callback method, you can begin making calls to the service, using the methods defined by the interface.
- To disconnect from the service, call unbindService().

Bound Services – Using Messenger

- The service implements a **Handler** that receives a callback for each call from a client.
- The Handler is used to create a **Messenger** object (which is a reference to the Handler).
- The Messenger creates an **IBinder** that the service returns to clients from onBind().
- Clients use the IBinder to instantiate the Messenger (that references the service's Handler), which the client uses to send Message objects to the service.
- The service receives each Message in its Handler—specifically, in the handleMessage() method.

Android Client-Server Interaction via Bound Services Across Processes Using Inter Process Communications (IPC)