Java II

:: Activity

Activity: Services (90m)

(Due: Thu, 08 Aug | Status: Not Completed)

This lesson will discuss Services, which allow applications to run operations in the background, outside of the main thread.

Objectives & Outcomes

Upon completion of this Activity, the learner will be able to:

- Explain the advantages of using Services to run tasks in the background and pass data back to the primary activities.
- Apply Services to mobile applications for efficient processing of data as background threads.

Level of Effort

This activity should take approximately 90m to complete. It will require:

- 90m Research
- 0m Prep & Delivery
- 0m Work

If you find that this activity takes you significantly less or more time than this estimate, please contact me for guidance.

Reading & Resources

Android Services Reference (helpful)

Vogella - Android Services Tutorial (necessary)

How To Avoid Context Leaks When Using Handlers (related)

Handlers are used in Services to communicate data back to an Activity. If a Handler is set up improperly it can cause massive memory leaks. This very short tutorial explains how to avoid this problem.

Services Video (necessary) Approx. Duration: 21m

This video shows how to create an Intent Service.

Parsing a JSON Array to a ListView – Part 1 (helpful) Approx. Duration: 19m

This video shows how to read a JSON String from a file and parse that JSON array to a Listview.

Parsing a JSON Array to a ListView – Part 2 (helpful) Approx. Duration: 13m

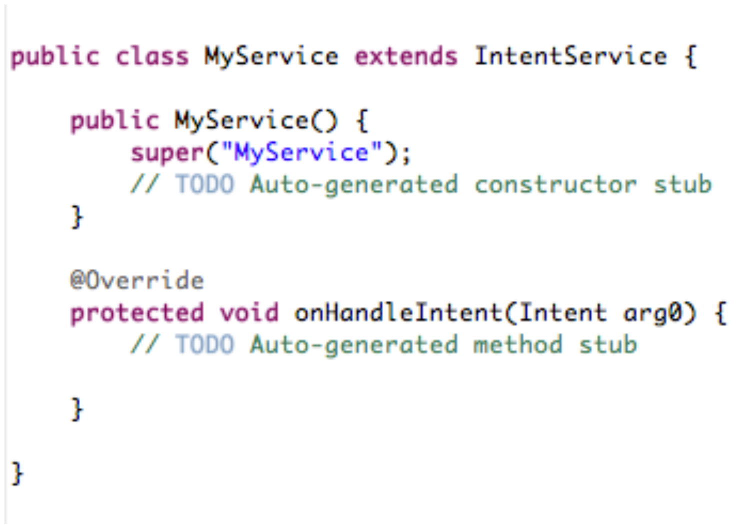
Continuation of JSON parsing to a ListView.

Instructions

Services provide applications with the ability to run tasks in the background, outside of the main thread. They are typically used for long running tasks such as optimizing a database, reading and writing files, but can also be used for shorter processes such as accessing http content. A service can be used to play music, or to perform periodic background updates to data used by an application, such as weather data that is constantly changing. Like an asynchronous task, a service can be used for processing that cannot be performed instantaneously, or is dependent on external factors such as internet access where response time is outside of our control.

Creating a Service

Services are created by extending the Service class or one of its associated subclasses. The most common implementation of a service is IntentService, a service that is started in an Activity. The IntentService Class contains a member method onHandleIntent which is executed when the service is started. This method may be thought of as the equivalent of the doInBackground method of an AsyncTask and this is where you will put your service code body.



```java
public class MyService extends IntentService {

    public MyService() {
        super("MyService");
        // TODO Auto-generated constructor stub
    }

    @Override
    protected void onHandleIntent(Intent arg0) {
        // TODO Auto-generated method stub

    }

}
```

The Service must be declared in the application manifest file.

```
<service android:name="MyService"></service>
```

Creating a Handler

Services may be created to collect information that will be used by the launching Activity. The communication between processes is handled by the Handler Class. The Handler Class is used to pass messages between background threads, such as Services, and the main application thread.

```
final Handler serviceHandler = new Handler() {
    public void handleMessage(Message message) {

        Object returnedObect = message.obj;

        // Test if status is OK.  Only test the returned Object if your
        // service is sending an Object through the message.
        if (message.arg1 == RESULT_OK && returnedObect != null) {

            // Do post service processing here
        }
    }
};
```

The Handler Class has a method handleMessage which takes a Message as an input parameter. The handleMessage method can be used similar to the way the onPostExecute method is used upon completion of an AsyncTask. It should perform post service processing only after validating the Message contents that it will be using. The Message Class contains two integers, arg1 and arg2 which may be used to return status values. The Message class also contains an Object which gives the designer the freedom to pass any type of data within the Message. While the onPostExecute method is automatically invoked upon completion of an AsyncTask, the handleMessage is explicitly called when the Service sends a message.

Creating the Messenger

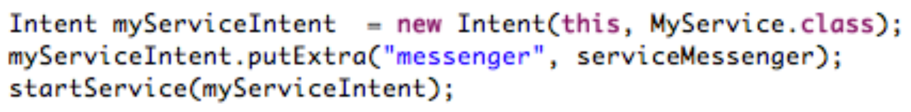
Once you have designed your Service Class and instantiated the Handler in your launching Activity you need a way for the Service to access that Handler so it can communicate with the Activity. The Messenger Class in instantiated in the calling activity and passed to the Service to provide a reference to the Handler.

```
Messenger serviceMessenger = new Messenger(serviceHandler);
```

The Messenger constructor accepts the Handler that will be used for communication between processes.

Starting the Service

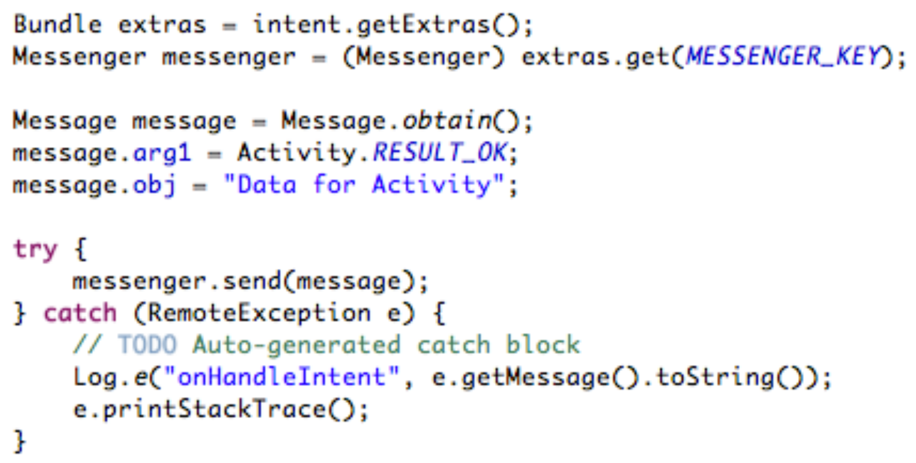
To interact with a Service, the Activity will contain an instantiation of a Handler and a Messenger. The Activity has to command the operating system to start the Service. This operation is done through an Intent Class. The Intent Class is a container for delivering information to the Android operating system, as well as, in this case the Service being started. The Intent constructor takes as input parameters the calling Activity's context, and the Service Class to start. Then the Intent is loaded with any data that is to be passed to this Service. You can put any type of data into the intent using the "putExtra" method, where you enter two parameters. The first parameter is a key string, giving a name to the data for the recipient to access. The second parameter in the putExtra method is the data, which can be of any type of data, even a class such as Messenger. The Messsenger is put into the Intent for the Service to access so it can respond to the calling Activity. Note the Intent only delivers in one direction and that is the reason the Messenger is delivered to the Service so it can then communication back through the handler referenced in the messenger.



```
Intent myServiceIntent  = new Intent(this, MyService.class);
myServiceIntent.putExtra("messenger", serviceMessenger);
startService(myServiceIntent);
```

Service Message to the Activity

The system will kill off an IntentService once it has run its course. As such, the Activity typically handles communication with the service via the Messenger once the operation is complete. One method of handling the communication is to pass a constant value through the messenger, indicating to the Activity whether or not the service was successful in completing its task. This is done with the constant *Activity.RESULT_CANCELED* and *Activity.RESULT_OK*. These are attached to the Message's arg1 property and sent back to the handler in the calling Activity. Along with the result, you can attach an object to the Message's obj property to send back to the calling Activity. In the onHandleIntent method of the Service the following code sends the message to the Handler.



```
Bundle extras = intent.getExtras();
Messenger messenger = (Messenger) extras.get(MESSENGER_KEY);

Message message = Message.obtain();
message.arg1 = Activity.RESULT_OK;
message.obj = "Data for Activity";

try {
    messenger.send(message);
} catch (RemoteException e) {
    // TODO Auto-generated catch block
    Log.e("onHandleIntent", e.getMessage().toString());
    e.printStackTrace();
}
```

The Service has access to the Intent as it is passed into the onHandleIntent method. From this Intent you can get the extras which are returned in a Bundle. The Message is obtained from the operating system. The operating system has a pool of Messages so new ones are not constantly allocated and using up memory. The Service then loads the Message with data to be sent to the Activity. Finally, the message is sent by passing the message into the Messenger send method.

Deliverables

There is no deliverable information associated with this activity