CMPS 121
Services

Luca de Alfaro

Portions of this presentations are based on content taken from the Android Developers Site. The work was created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.
Services

Services are application components that can run long-lived background tasks. Services run even when the user switches to another app.

A service has two ways of running:

● **Started**: `startService()` has been called. The service can run, until it decides to stop itself.

● **Bound**: An application component can bind a service by calling `bindService()`. A bound service runs only as long as there is something bound to it.
Creating a service

Subclass Service, then override:

- **onStartCommand()** -- called when startService() is called. Then you can call stopSelf() or stopService()
- **onBind()** -- called when bindService() is called. Returns an IBinder (or null if you don't want to be bound).
- **onCreate()** -- called before above methods.
- **onDestroy()** -- called when about to be shut down.
Declaring a service in the manifest

<manifest ...
 ...
 <application ...
  <service android:name=".ExampleService" />
   android:enabled=["true" | "false"]
   android:exported=["true" | "false"]
   android:icon="drawable resource"
   android:label="string resource"
   android:permission="string"
   android:process="string" 
...
  </service>
 ...
 </application>
</manifest>
Creating a Started Service

There are two classes you can subclass:

- **Service**: you need to create a new thread, since it is not created by default.
- **IntentService**: This uses a worker thread to perform the requests, and all you need to do is override `onHandleIntent`. This is the easiest, provided you don't need to handle multiple requests.
The IntentService class

The IntentService class does the following:

- Creates a default worker thread that executes all intents delivered to `onStartCommand()` separate from your application's main thread.
- Creates a work queue that passes one intent at a time to your `onHandleIntent()` implementation, so you never have to worry about multi-threading.
- Stops the service after all start requests have been handled, so you never have to call `stopSelf()`.
- Provides default implementation of `onBind()` that returns null.
- Provides a default implementation of `onStartCommand()` that sends the intent to the work queue and then to your `onHandleIntent()` implementation.

All you have to do is handle `onHandleIntent()`.
public class HelloIntentService extends IntentService {
    // A constructor is required, and must call the super IntentService(String)
    // constructor with a name for the worker thread.
    public HelloIntentService() { super("HelloIntentService"); }

    // The IntentService calls this method from the default worker thread with the intent that started
    // the service. When this method returns, IntentService stops the service, as appropriate.
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a file.
        // For our sample, we just sleep for 5 seconds.
        long endTime = System.currentTimeMillis() + 5*1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                } catch (Exception e) {
                }
            }
        }
    }
}
Extending the Service class

```java
public class HelloService extends Service {
    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // Handler that receives messages from the thread
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {
            // Normally we would do some work here, like download a file.
            // For our sample, we just sleep for 5 seconds.
            long endTime = System.currentTimeMillis() + 5*1000;
            while (System.currentTimeMillis() < endTime) {
                synchronized (this) {
                    try {
                        wait(endTime - System.currentTimeMillis());
                    } catch (Exception e) {
                    }
                }
            }
            // Stop the service using the startId, so that we don't stop
            // the service in the middle of handling another job
            stopSelf(msg.arg1);
        }
    }
}
```
@Override
public void onCreate() {
    // Start up the thread running the service. Note that we create a
    // separate thread because the service normally runs in the process's
    // main thread, which we don't want to block. We also make it
    // background priority so CPU-intensive work will not disrupt our UI.
    HandlerThread thread = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();

    // Get the HandlerThread's Looper and use it for our Handler
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
}
Extending the Service class (cont.)

```java
@override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

    // For each start request, send a message to start a job and deliver the
    // start ID so we know which request we're stopping when we finish the job
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    mServiceHandler.sendMessage(msg);

    // If we get killed, after returning from here, restart
    return START_STICKY;
}
```
Extending the Service class (cont.)

```java
@Override
public IBinder onBind(Intent intent) {
    // We don't provide binding, so return null
    return null;
}

@Override
public void onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
}
```
Starting and stopping a service

**Starting:**

```java
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

**Stopping:**

- It can stop itself calling `stopSelf()`, or another component can call `stopService()`.
- You can call `stopSelf(startId)` passing the id of the call that started it, so it will finish only when all requests are serviced.
Running a service in the foreground

- To request running in foreground: `startForeground()`
- To remove from foreground: `stopForeground()`

```java
Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker_text), System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(this, getText(R.string.notification_title), getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION, notification);
```
Service lifecycle

- Call to `startService()`
  - `onCreate()`
  - `onStartCommand()`
  - **Service running**
    - The service is stopped by itself or a client
    - `onDestroy()`
      - **Service shut down**
- Call to `bindService()`
  - `onCreate()`
  - `onBind()`
  - **Clients are bound to service**
    - All clients unbind by calling `unbindService()`
    - `onUnbind()`
      - **Service shut down**

- **Active Lifetime**
- **Unbounded service**
- **Bounded service**