Working in the Background - Services, Threads, Handlers, Async Tasks

Penyo P. Atanasov

Astea Solutions

Make your application responsive

- UI Thread
- Make sure you respond within 5 seconds
- Do expensive operations in a background service (relying on notifications to prompt users to go back to your activity)
- Do expensive work in a background thread

Services

- Service - faceless task that runs in the background.
- Managed from other application components including:
  - ✓ Services
  - ✓ Activities
  - ✓ Broadcast Receivers

Services

- Perform long-running operations
  - ✓ Downloading resources
  - ✓ Server synchronization
  - ✓ Etc ...
- Supply functionality of one application to other applications
  - ✓ Account service
  - ✓ Connectivity service
  - ✓ Audio service
  - ✓ Etc... - see Context constants

Creating a Service

- AndroidManifest.xml
  - ✓ <service>
  - ✓ <intent-filter>
  - ✓ <uses-permission>

- Java file – onBind(Intent intent)

Example

AndroidManifest.xml:

```
<service
        android:name=".NewsService">
        <intent-filter>
                <action
                        android:name="bg.sofia.uni.fmi.NEWS_SERVICE" />
        </intent-filter>
</service>
```

NewsService.java:

```
public class NewsService extends Service {
        ...
        @Override
        public void onCreate() {
                super.onCreate();
                mHandler = new Handler();
        }

        @Override
        public IBinder onBind(Intent intent) {
                return new NewsServiceBinder(this);
        }
        ...
}
```

Initializing the Service

- bindService(Intent service, ServiceConnection conn, int flags):
  - ✓ Starts a Service which lives as long as the Activity/Service, that started it, is living
  - ✓ An instance of the Service can be obtained through the ServiceConnection

Initializing the Service

- startService(Intent intent):
  - ✓ Starts the Service independently of the lifecycle of the Activity/Service that has started it
  - ✓ Overrides bindService and in order to stop the Service a consequent stopService(Intent service) invocation should be made
  - ✓ onStart(Intent intent, int startId)
  - ✓ onStartCommand(Intent intent, int flags, int startId)
- A service can be stopped by the OS!

Started Services modes

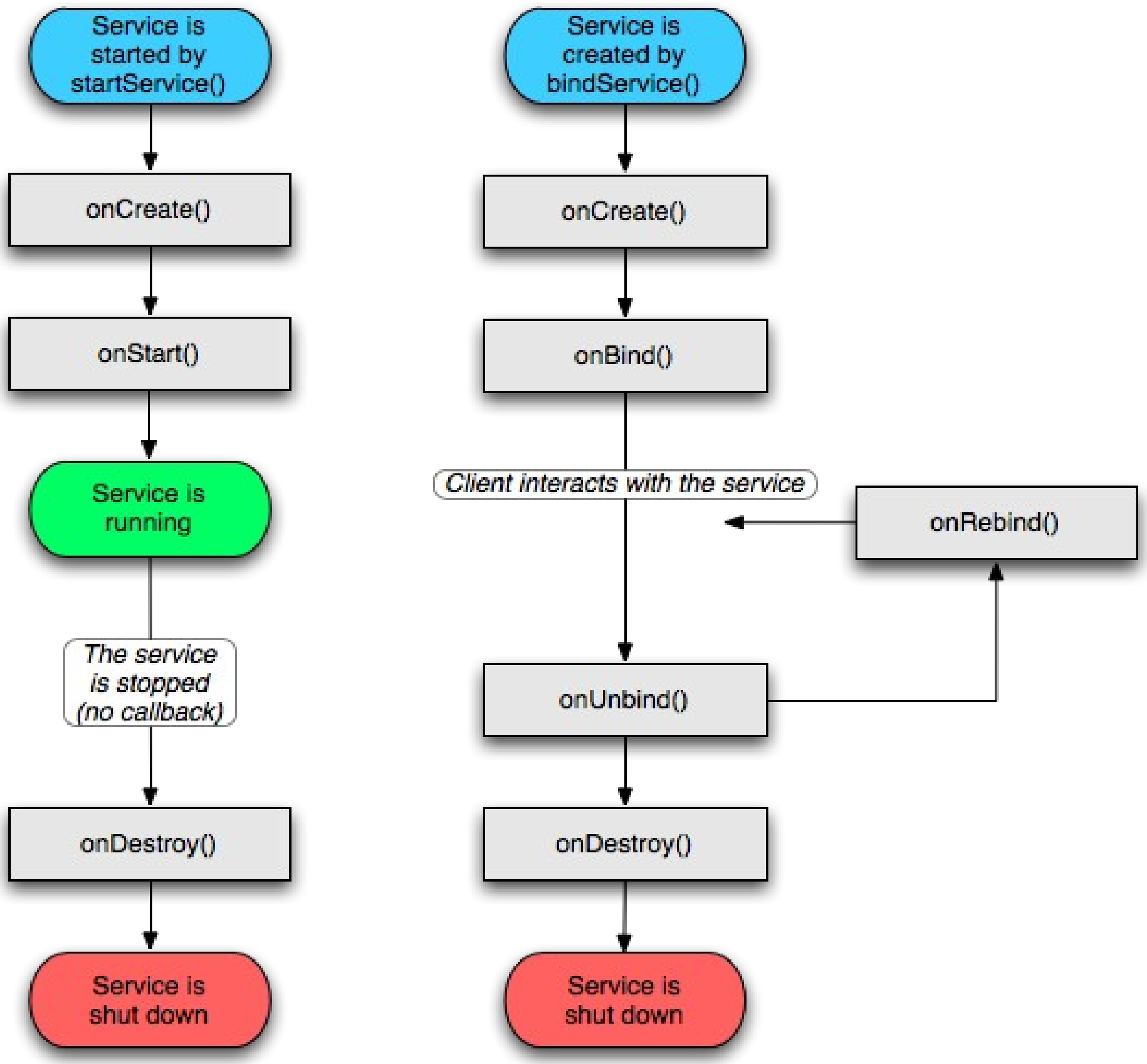
int onStartCommand (Intent intent, int flags, int startId):

- START_STICKY
- START_NOT_STICKY
- START_REDELIVER_INTENT
- START_FLAG_REDELIVERY
- START_FLAG_RETRY

Example

```
...
private NewsService mService;
private ServiceConnection mConnection = new ServiceConnection() {

        @Override
        public void onServiceDisconnected(ComponentName name) {
                mService = null;
        }

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
                NewsServiceBinder binder = (NewsServiceBinder)service;
                mService = binder.getService();
        }
};

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        bindService(new Intent("bg.sofia.uni.fmi.NEWS_SERVICE"), mConnection,
        Service.BIND_AUTO_CREATE);
}
...
```

Service lifecycle


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Runnable

- A command that can be executed
- Often used to run code in a different Thread
- run()

Background threads

- Use them for time all time-comsuming processing like:
  - File operations
  - Network lookups
  - Database transactions
  - Complex calculations
  - Etc.
- Multiple threads

Initializing a Thread

- Override run()
- Provide a Runnable instance
- Start()
- setPriority()
- setDaemon()

Managing Threads

- Deprecated – stop(), suspend()
- Running:

```
public void run() {
    while(<boolean>){
        <do some processing here>
    }
}
```

Example

```
// A method called on the main GUI thread.
private void mainThreadProcessing() {
	// This moves the time consuming operation to a child thread.
	Thread thread = new Thread(null, doBackgroundThreadProcessing,
	                                                  “Background”);
	thread.start();
}

// A Runnable executed in the background processing method.
private Runnable doBackgroundThreadProcessing = new Runnable() {
	public void run() {
		backgroundThreadProcessing();
	}
};

// Method which does some processing in the background.
private void backgroundThreadProcessing() {
[ … Time consuming operations … ]
}
```

Handlers

- Handlers and Threads
- UI and background Threads synchronization
- Allows posting methods on the thread where the handler was created

Handlers

- Posts can be delayed using postDelay and postAtTime

- Usage
  - Schedule Messages/Runnables
  - Enqueue actions to be perfomed on a different Thread

Using Handlers

```
// Initialize a handler on the main thread.
private Handler handler = new Handler();

// Method which does some processing in the background.
private void backgroundThreadProcessing() {
	[ ... Time consuming operations ... ]
	handler.post(doUpdateGUI);
}

// Runnable that executes the update GUI method.
private Runnable doUpdateGUI = new Runnable() {
	public void run() {
		updateGUI();
	}
};

private void updateGUI() {
	[ ... Open a dialog or modify a GUI element ... ]
}
```

AIDL

- Android Interface Definition Language: Provides support for interprocess communication between services and application components
  - OS-level primitives
  - Process boundaries
  - Independent applications

Implementing AIDL
(data types)

- Java language primitives (int, boolean, float, char, etc.)
- String and CharSequence values
- List (including generic) objects, where each element is a supported type.
- Map (not including generic) objects in which each key and element is a supported type
- Other AIDL-generated interfaces(an import statement is always needed for these)
- Classes that implement the Parcelable interface. An import statement is always needed for these.

Implementing AIDL

- Java interface-similar syntax
  - specify a fully qualified package name
  - import all the packages required

- Methods can take zero or more parameters and return void or a supported type

AIDL file:

```
package bg.uni.sofia.fmi;

interface NewsUpdater {

	void scheduleNewsUpdate(long milis);

}
```

FmiNewsService class:

```
@Override
public IBinder onBind(Intent intent) {
	return new NewsUpdater.Stub() {

		@Override
		public void scheduleNewsUpdate(long milis) throws RemoteException {
			FmiNewsService.this.scheduleNewsUpdate(milis);
		}
	};
}
```

NewsReader activity class:

```
private NewsUpdater mService;
private ServiceConnection mConnection = new ServiceConnection() {

	@Override
	public void onServiceDisconnected(ComponentName name) {
		mService = null;
	}

	@Override
	public void onServiceConnected(ComponentName name, IBinder service) {
		mService = NewsUpdater.Stub.asInterface(service);
	}
};
```

AsyncTasks

- Perform background operations
- Publish results on the UI thread
- No Threads and/or Handlers
- Must be created on the
  UI Thread

AsyncTasks

- Defined by 3 generic types
  - ✓ Params
  - ✓ Progress
  - ✓ Result

- Lifecycle
  - ✓ onPreExecute()
  - ✓ abstract Result doInBackground(Params... params)
  - ✓ onProgressUpdate(Progress...)
  - ✓ onPostExecute(Result)

Rules

- The task instance should be created on the UI Thread
- execute(Params ...) should be invoked on the UI Thread
- The task can be executed ONLY ONCE
- Do not invoked its methods manually

Example

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}

new DownloadFilesTask().execute(url1, url2, url3);
```

Questions ?