Introduction to background processing in Android - Tutorial

Lars Vogel (c) 2014, 2017 vogella GmbH – Version 3.5, 14.09.2017

Table of Contents

This tutorial introduces the concept of asynchronous processing in Android applications.

1. Background processing in Android (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#background-processing-in-android)

1.1. Why using concurrency? (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#why-using-concurrency)

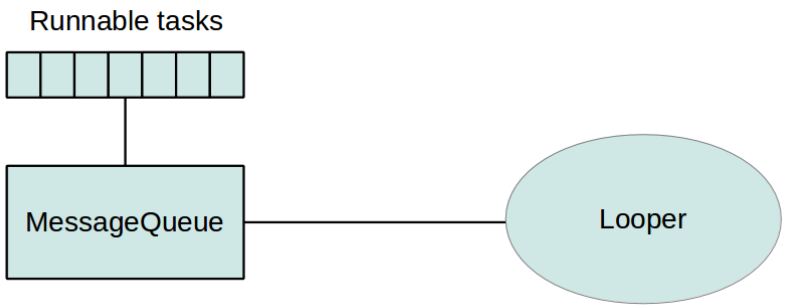
By default, application code runs in the main thread. Every statement is therefore executed in sequence. If you perform a long lasting operation, the application blocks until the corresponding operation has finished.

To provide a good user experience all potentially slow running operations in an Android application should run asynchronously. This can be archived via concurrency constructs of the Java language or of the Android framework. Potentially slow operations are for example network, file and database access and complex calculations.

> Android enforces a worst case reaction time of applications. If an *activity* does not react within 5 seconds to user input, the Android system displays an *Application not responding* (ANR) dialog. From this dialog the user can choose to stop the application.

1.2. Main thread (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#n thread)

Android modifies the user interface and handles input events from one single thread, called the *main thread*. Android collects all events in this thread in a queue and processes this queue with an instance of the `Looper` class.



1.3. Threading in Android (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#t in-android)

Android supports the usage of the `Thread` class to perform asynchronous processing. Android also supplies the `java.util.concurrent` package to perform something in the background. For example, by using the `ThreadPools` and `Executor` classes.

If you need to update the user interface from a new `Thread`, you need to synchronize with the main thread. Because of this restrictions, Android developer typically use Android specific code constructs.

Standard Android provides the `android.os.Handler` class or the `AsyncTasks` classes. More sophisticated approaches are based on open source solutions which are using callbacks.

For example, the RxJava open source framework allow to specify an observer of an observable stream of data. Once the events happen the observer is called by the framework. You can configure in which thread the observer and observable run.

2. Handler (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/a

2.1. Purpose of the Handler class (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#p of-the-handler-class)

A `Handler` object registers itself with the thread in which it is created. It provides a channel to send data to this thread, for example the main thread. The data which can be posted via the `Handler` class can be an instance of the `Message` or the `Runnable` class. A Handler is particular useful if you have want to post multiple times data to the main thread.

2.2. Creating and reusing instances of Handlers (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#c and-reusing-instances-of-handlers)

To implement a handler subclass it and override the `handleMessage()` method to process messages. You can post messages to it via the `sendMessage(Message)` or via the `sendEmptyMessage()` method. Use the `post()` method to send a `Runnable` to it.

To avoid object creation, you can also reuse the existing `Handler` object of your activity.

JAVA

```
// Reuse existing handler if you don't
// have to override the message processing
handler = getWindow().getDecorView().getHandler();
```

The `View` class allows you to post objects of type `Runnable` via the `post()` method.

2.3. Example (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#e

The following code demonstrates the usage of an handler from a view. Assume your activity uses the following layout.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:indeterminate="false"
        android:max="10"
        android:padding="4dip" >
    </ProgressBar>

     <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" >
      </TextView>
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="startProgress"
        android:text="Start Progress" >
    </Button>

</LinearLayout>
```

With the following code the `ProgressBar` get updated once the users presses the `Button`.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.TextView;

public class ProgressTestActivity extends Activity {
    private ProgressBar progress;
    private TextView text;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        progress = (ProgressBar) findViewById(R.id.progressBar1);
        text = (TextView) findViewById(R.id.textView1);

    }

    public void startProgress(View view) {
        // do something long
        Runnable runnable = new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i <= 10; i++) {
                    final int value = i;
                     doFakeWork();
                    progress.post(new Runnable() {
                        @Override
                        public void run() {
                            text.setText("Updating");
                            progress.setProgress(value);
                        }
                    });
                }
            }
        };
        new Thread(runnable).start();
    }

    // Simulating something timeconsuming
    private void doFakeWork() {
        SystemClock.sleep(5000);e.printStackTrace();
    }

}
```


3. AsyncTask (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/a

===

The `AsyncTask` class allows to run instructions in the background and to synchronize again with the main thread. It also reporting progress of the running tasks. AsyncTasks should be used for short background operations which need to update the user interface.

To use `AsyncTask` you must subclass it. The parameters are the following `AsyncTask <TypeOfVarArgParams, ProgressValue, ResultValue>`.

An `AsyncTask` is started via the `execute()` method. This `execute()` method calls the `doInBackground()` and the `onPostExecute()` method.

returned from `doInBackground()` method. This parameter is passed to `onPostExecute()` as a parameter.

The `doInBackground()` method contains the coding instruction which should be performed in a background thread. This method runs automatically in a separate `Thread`.

The `onPostExecute()` method synchronizes itself again with the user interface thread and allows it to be updated. This method is called by the framework once the `doInBackground()` method finishes.

3.1. Parallel execution of several AsyncTasks (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#parallel-execution-of-several-asynctasks)

By default, `AsyncTask` tasks are executed sequence (for Android versions higher than Android 3.0). To run AsyncTasks in sequence use the `executeOnExecutor()` method specifying `AsyncTask.THREAD_POOL_EXECUTOR` as first parameter.

```
// Assume ImageLoader extends AsyncTask
ImageLoader imageLoader = new ImageLoader( imageView );

// Execute in parallel
imageLoader.executeOnExecutor( AsyncTask.THREAD_POOL_EXECUTOR,
"http://url.com/image.png" );
```

TXT

3.2. Example: AsyncTask (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#example-asynctask)

The following code demonstrates how to use the `AsyncTask` to download something from the Internet. The code requires the `android.permission.INTERNET` permission in your Android manifest.

```
// imports cut out for brevity

public class ReadWebpageAsyncTask extends Activity {
    private TextView textView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textView = (TextView) findViewById(R.id.TextView01);
    }

    private class DownloadWebPageTask extends AsyncTask<String, Void, String>
{
        @Override
        protected String doInBackground(String... urls) {
            // we use the OkHttp library from
https://github.com/square/okhttp
            OkHttpClient client = new OkHttpClient();
            Request request =
                    new Request.Builder()
                    .url(urls[0])
                    .build();
                Response response = client.newCall(request).execute();
                if (response.isSuccessful()) {
                    return response.body().string();
                }
            }
            return "Download failed";
        }

        @Override
        protected void onPostExecute(String result) {
            textView.setText(result);
        }
    }

    // Triggered via a button in your layout
    public void onClick(View view) {
        DownloadWebPageTask task = new DownloadWebPageTask();
        task.execute(new String[] { "http://www.vogella.com/index.html" });

    }
}
```


4. Background processing and lifecycle handling (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/a processing-and-lifecycle-handling)

4.1. Retaining state during configuration changes (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#r state-during-configuration-changes)

One challenge in using threads is to consider the lifecycle of the application. The Android system may kill your *activity* or trigger a configuration change which will also restart your activity.

You also need to handle open dialogs, as dialogs are always connected to the activity which created them. In case the activity gets restarted and you access an existing dialog you receive a `View not attached to window manager` exception.

4.2. Using the application object to store objects (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#u the-application-object-to-store-objects)

You can implement an `Application` class for your Android application.

To use your application class assign the classname to the `android:name` attribute of your application.

```
        <activity android:name=".ThreadsLifecycleActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

The application class is automatically created by the Android runtime and is available unless the whole application process is terminated.

This class can be used to access objects which should be cross activities or available for the whole application lifecycle. In the `onCreate()` method you can create objects and make them available via public fields or `getter` methods.

The `onTerminate()` method in the application class is only used for testing. If Android terminates the process in which your application is running all allocated resources are automatically released.

You can access the Application via the `getApplication()` method in your activity.

5. Headless Fragments and background processing (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#headless-fragments-and-background-processing)

Fragments which do return null in the `onCreateView` methods are called *headless fragments*, as the provide no views. Headless fragments can retain there fields between configuration changes via the `setRetainInstance(true)` method call.

For example, if you define an `AsyncTask` as field in a headless fragment, it can continue to run during configuration changgs.

6. About this website (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#this-website)

(http://www.vogella.com/code/index.html)

7. Links and Literature (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#links-and-literature)

7.1. Concurrency Resources (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#concurrency-resources)

Java Concurrency (http://www.vogella.com/tutorials/JavaConcurrency/article.html)

Networking with Android
(http://www.vogella.com/tutorials/AndroidNetworking/article.html)

7.2. Android Resources (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#a resources)

Introduction to Android Development (http://www.vogella.com/tutorials/Android/article.html)

Tutorial about the usage of fragments (http://www.vogella.com/tutorials/AndroidFragments/article.html)

Tutorial about Android services (http://www.vogella.com/tutorials/AndroidServices/article.html)

7.3. vogella GmbH training and consulting support (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html#v gmbh-training-and-consulting-support)

TRAINING (http://www.vogella.com/training/)	**SERVICE & SUPPORT** (http://www.vogella.com/consulting/)
The vogella company provides comprehensive training and education services (http://www.vogella.com/training/) from experts in the areas of Eclipse RCP, Android, Git, Java, Gradle and Spring. We offer both public and inhouse training. Whichever course you decide to take, you are guaranteed to experience what many before you refer to as "The best IT class I have ever attended" (http://www.vogella.com/training/).	The vogella company offers expert consulting (http://www.vogella.com/consulting/) services, development support and coaching. Our customers range from Fortune 100 corporations to individual developers.

Appendix A: Copyright and License (http://www.vogella.com/tutorials/AndroidBackgroundProcessing/a and-license)

Version 3.5
Last updated 2018-02-13 11:59:09 +01:00