IntentService

Par Florian Fournier
dans le cadre d'Android2ee

Des notions de bases sur les Intents, Service et BroadCastReceiver sont nécessaires pour ce tutoriel.

Qu'est ce que sont les IntentService ? Pourquoi les utiliser ? Peut être que ce sont des questions que vous vous êtes posés, nous allons essayer d'y répondre dans ce tutoriel:

Table des matières

1 Description

Tout d'abord les IntentService sont un type de service Android qui permet de gérer les requêtes asynchrones. Ici, le mot requête doit être considéré comme une action ou une demande de traitement.

Ces requêtes sont exécutées dans un « worker », par conséquent, les traitements sont exécutés les uns après les autres au sein de ce thread (qui n'est pas le thread U.I.). Les IntentService vous permettent de décharger le Thread UI de traitements lourds tout en ayant les avantages des services qui sont indépendants du cycle de vie des activités, ou bien lancer ces traitements qui pourront être exécutés dans un autre processus (c'est la notion de RemoteService).

Les IntentService s'arrêtent d'eux-mêmes à la fin des traitements demandés. Pour rajouter un nouveau traitement, il vous suffit de le rajouter dans la liste d'attente de l'IntentService.
Par contre pour récupérer les résultats du traitement, il vous faudra transférer le résultat par message (Intent, broadcast, handler, etc.) au(x) destinataire(s) voulu(s). C'est l'inconvénient de l'IntentService qui n'a pas accès directement au Thread UI, contrairement aux AsyncTask.

Si nous résumons rapidement, les IntentService sont utiles pour gérer des requêtes multiples de manière asynchrone (bien que leur exécution soit synchrone dans le worker). Ainsi, nous avons l'avantage du cycle de vie des Services, de l'arrêt automatique de celui-ci quand l'action est terminée et de l'exécution du traitement en tâche de fond dans un seul concept. L'inconvénient est l'absence d'accès direct à l'interface graphique.

2 Implémentation d'un IntentService.

Prenons l'exemple d'implémentation fourni par la documentation de Google :

```
public class HelloIntentService extends IntentService {
    /**
     * Un constructeur est requis, et doit appeler la méthode
     * superIntentService(String)
     * constructeur avec un nom pour le « worker thread »
     */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /**
     * L'IntentService appelle la méthode par défaut du « worker thread » avec
     * l'intent passé en paramètre de la méthode. Quand cette méthode est
termine
     * le service s'arrête de lui-même
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normalement le traitement lourd se fait ici comme télécharger un
fichier
        // A ne pas faire mais dans cet exemple une attente de 5 secondes est
        // faites, ceci ne bloque pas le « thread UI » puisque nous sommes dans
le
        // « worker thread »
        long endTime = System.currentTimeMillis() + 5 * 1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                } catch (Exception e) { }
            }
        }
    }
}
```

On a un peu de mal à savoir comment le Service fonctionne ainsi. Où se trouve notre file d'attente d'intention ? Comment le « worker thread » est lancé ?

Regardons alors directement dans le code des IntentService

```
private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }

        @Override
        public void handleMessage(Message msg) {
            onHandleIntent((Intent)msg.obj);
           stopSelf(msg.arg1); // arrêt du Service
        }
    }
```

Premièrement, une classe interne « ServiceHandler » est crée avec comme paramètre de son constructeur un Looper (qui est une gestion de queue de message).

Cette classe hérite d'un Handler qui permet de recevoir des messages et de les traiter.

Lors de la réception d'un de ces messages (dans la fonction handleMessage), nous pouvons voir que la fonction onHandleIntent est invoquée, le traitement (réel) se fera au sein de cette méthode.

L'appel à la fonction stopSelf arrête ensuite le service (StopSelf est une fonction de la classe Service Android).

On comprend comment le Service s'arrête de lui même. Mais, qui lui passe ce Looper et comment est-ce qu'il gère sa file d'attente ? Regardons dans la fonction onCreate de la classe IntentService.

```
  public abstract class IntentService extends Service {
//blabla
@Override
    public void onCreate() {
        super.onCreate();
        HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");
        thread.start(); // notre « worker thread »
        mServiceLooper = thread.getLooper();// son Looper
        mServiceHandler = new ServiceHandler(mServiceLooper);// passage du Looper au
ServiceHandler
    }
```

Un nouveau thread est créé à partir de la classe HandlerThread et son Looper est passé à la classe ServiceHandler que nous venons de voir. Nous avons trouvé notre « worker thread ».

Depuis la fonction onStart, le service gère la file d'attente via les intentions par l'envoie de messages à notre « worker thread ».

```
  public abstract class IntentService extends Service {
//blabla
@Override
    public void onStart(Intent intent, int startId) {
        Message msg = mServiceHandler.obtainMessage();// Créé un message pré-rempli
        msg.arg1 = startId;//affecte un identifiant au message (utilisé pour arrêter ou pas le
service)
        msg.obj = intent;// l'action associée au message
        mServiceHandler.sendMessage(msg); // ajoute le message dans la file d'attente du Looper
    }

     // Start Service
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        onStart(intent, startId);
        return mRedelivery ? START_REDELIVER_INTENT : START_NOT_STICKY;
    }
```

La méthode onStart est appelée depuis la méthode onStartCommand. Cette fonction est invoquée automatiquement à chaque appel lors du démarrage d'un service par la méthode startService (c.f.le cycle de vie des Services Android).

Par conséquent, pour ajouter une nouvelle demande il vous suffit d'appeler la fonction startService appartenant à la classe Context en lui passant l'Intent qui porte les paramètres du traitement.

L'envoie du message au ServiceHandler est effectué au sein de la méthode onStart de l'IntentService par l'appel à la fonction sendMessage. Dans ce message nous trouvons l'intent passé en paramètre et le startId, qui identifiera ce service lors de l'appel du stopSelf.

Une option permet de redémarrer le service lorsque celui-ci a été détruit par le système pour des questions de place mémoire. Ce redémarrage (cette restitution de l'Intent d'où la notion de Redelivery), renverra seulement l'Intent le plus récent. Il suffit d'appeler la fonction setIntentRedelivery(true) dans le onCreate ou le constructeur de l'IntentService. Cette option peut être utile si vous avez une seule action à réaliser. Il est tout de même plus pertinent de connaître l'état des actions non effectuées pour pouvoir les relancer plutôt que de s'appuyer sur ce flag.

Dans la fonction onDestroy, on nettoie le Looper permettant de vider et arrêter la queue de messages en cours, et de ce fait de terminer le thread proprement.

```
   public abstract class IntentService extends Service {
//blabla
    @Override
     public void onDestroy() {
          mServiceLooper.quit(); // nettoie la queue de messages et stopper le Thread
     }
```

Pour résumer les IntentService fournissent :

- un « worker thread » qui exécute tout les intents passés à la fonction onStartCommand (via la méthode Context.StartService(Intent intent))
- une file d'attente sur les intents passés qui est gérée par la fonction sendMessage (du ServiceHandler)
- le traitement des intents qui se fait depuis la fonction onHandleIntent de votre classe qui hérite d'IntentService. Cette méthode s'exécute dans le « worker thread », pas de souci de multi threading (synchronised et autre mutex)
- arrête le service quand le traitement est terminé
- retourne null sur la fonction onBind par défaut (par défaut on n'a pas besoin de s'abonner au Service)

Nous avons vu comment les IntentService fonctionnaient mais est ce que l'on pourrait avoir un exemple concret ?

3 Exemple d'un IntentService

Essayons de mettre en place un Intent Service permettant de récupérer le contenue d'une page web (code source) et de l'afficher depuis une TextView prévue à cet effet.

Pour la communication entre votre IntentService et l'activité nous passerons pas un BroadcastReceiver. Ce principe est la manière naturelle sous Android pour effectuer ce dialogue et permet de s'adapter aux envoies multiples.

Voici l'IntentService :

```
package com.android2ee.tutorial.intentservice.;
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URLConnection;
import java.net.URL;

import com.android2ee.tutorial.intentservice.MainActivity.MyReceiver;

import android.app.IntentService;
import android.content.Intent;

public class DownloadSourceService extends IntentService {

    public static final String URL = "urlpath";
    public static final String SOURCE_URL = "destination_source";

    public DownloadSourceService() {
            super("DownloadService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
            String urlPath = intent.getStringExtra(URL);
            InputStream is = null;
            BufferedReader r = null;
            StringBuilder result = null;

            // on récupère les données depuis l'url
            try {
                            URL aURL = new URL(urlPath);
                            URLConnection conn = aURL.openConnection();
                            conn.connect();
                            is = conn.getInputStream();
                            r = new BufferedReader(new       InputStreamReader(is));
                                    result = new StringBuilder();
                                    String line;
                    while ((line = r.readLine()) != null) {
                             result.append(line);
                    }
            } catch (IOException e) {
                    // message d'erreur

            } finally {
                    // on ferme bien tout les flux
                    if ( r != null) {
                            try {
                                    r.close();
                            } catch (IOException e) {
                                    // message d'erreur
                            }
                    }
            }
            // maintenant on transmet le résultat
            // on pourrait avoir un Handler, BroadCast, Notification, etc.
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction(MyReceiver.ACTION_RESP);
```

```
            broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
            broadcastIntent.putExtra(SOURCE_URL, result.toString());
            sendBroadcast(broadcastIntent);

        }
    }
```

L'implémentation est très simple au final, il suffit seulement de développer son traitement depuis la fonction onHandleIntent et ensuite de choisir une méthode de diffusion du résultat à la fin de celle-ci.

Il existe plusieurs méthodes de diffusion possibles, parmi celles-ci vous pouvez avoir :

- *un broadcast* : permet d'avoir une diffusion au sens large
- *un handler* : permet d'avoir une diffusion vers un correspondant unique.
- *une notification* : permet d'avoir une diffusion vers le centre de notification du téléphone
- *un intent* : permet de lancer une action en fin de traitement.
- *un listener* : permet d'avoir une diffusion sur l'ensemble des personnes abonnées.
- *etc.*

Voici l'Activité qui lancera le service et ainsi, qui récupérera le message répondu par celui-ci et l'affichera sur sa vue.

```
    package com.android2ee.tutorial.intentservice.;

    import android.os.Bundle;
    import android.app.Activity;
    import android.content.BroadcastReceiver;
    import android.content.Context;
    import android.content.Intent;
    import android.content.IntentFilter;
    import android.widget.TextView;

    public class IntentServiceActivity extends Activity {

        public class MyReceiver extends BroadcastReceiver {
            public static final String ACTION_RESP
="com.myapp.intent.action.TEXT_TO_DISPLAY";

            @Override
            public void onReceive(Context context, Intent intent) {
                String text =
intent.getStringExtra(DownloadSourceService.SOURCE_URL);
                // send text to display
                TextView result = (TextView) findViewById(R.id.text_result);
                result.setText(text);

            }
        }

        private MyReceiver receiver;
```

```
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		// on initialise notre broadcast
		receiver = new MyReceiver();
		// on lance le service
		Intent msgIntent = new Intent(this, DownloadSourceService.class);

	msgIntent.putExtra(DownloadSourceService.URL,"http://api.openweathermap.org/data
/2.5/weather?q=London&mode=xml");
		startService(msgIntent);
	}

	@Override
	protected void onResume() {
		super.onResume();
		// on déclare notre Broadcast Receiver
		IntentFilter filter = new IntentFilter(MyReceiver.ACTION_RESP);
		filter.addCategory(Intent.CATEGORY_DEFAULT);
		registerReceiver(receiver, filter);
	}

	@Override
	protected void onPause() {
		super.onPause ();
		// on désenregistre notre broadcast
		unregisterReceiver(receiver);
	}

}
```

Ne pas oublier de déclarer l'IntentService dans le manifest comme un Service classique.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.android2ee.tutorial.intentservice."
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="19" />

    <!-- Internet permission, as we are accessing to url -->
   <uses-permission android:name="android.permission.INTERNET" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.android2ee.tutorial.intentservice.MainActivity"
         android:label="@string/app_name" >
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- Declaring Service in Manifest -->
        <service
            android:name=".DownloadSourceService"
            android:exported="false" />
    </application>

</manifest>
```

Enfin le fichier activity_main qui reste très simple :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/text_result"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

4 Conlusion

Les IntentService sont intéressants lorsque vous avez une tâche de fond régulière au sein de la vie de votre application. Par exemple :

- Flux RSS ;
- Mise à jour de la base de données ;
- Téléchargement d'images à partir d'un ensemble d'U.R.L.
- etc.

Avec les IntentService, le développement de ces tâches seront simples comparé aux autres moyens que nous avons, dû à la simplicité de ce pattern. Cette méthode vous donne les avantages des services Android qui peuvent être disponibles depuis votre application (ou depuis les autres applications, si celui-ci a été exporté depuis le manifest).

Utiliser réellement les IntentService quand vous avez des requêtes multiples, ceux-ci vous simplifieront vraiment la vie sur le cycle de vie de ce traitement ainsi que la gestion de sa file

d'attente. Vous pouvez passer par les Intent des paramètres pour différentier le fonctionnement à réaliser du traitement en tâche de fond.

Mais surtout ce DesignPattern est excellent pour mettre en place une architecture simple qui vous permet d'enchaîner des traitements dans un autre Thread que le ThreadUI. Cela vous permet d'éviter de mettre en place une architecture complexe (un PoolExecutor, des Services, des Threads de traitements) pour un traitement qui ne l'est pas. A noter que si vous effectuez une application complexe vous ne pourrez pas vous appuyer exclusivement sur ce pattern et vous devrez mettre en place une véritable architecture robuste.

5 Approfondissons le sujet

Et si nous prenions du temps à regarder réellement l'enchaînement des actions bas niveaux, pour comprendre au mieux ce pattern.

Voici le diagramme de séquence que l'on obtiendrais :

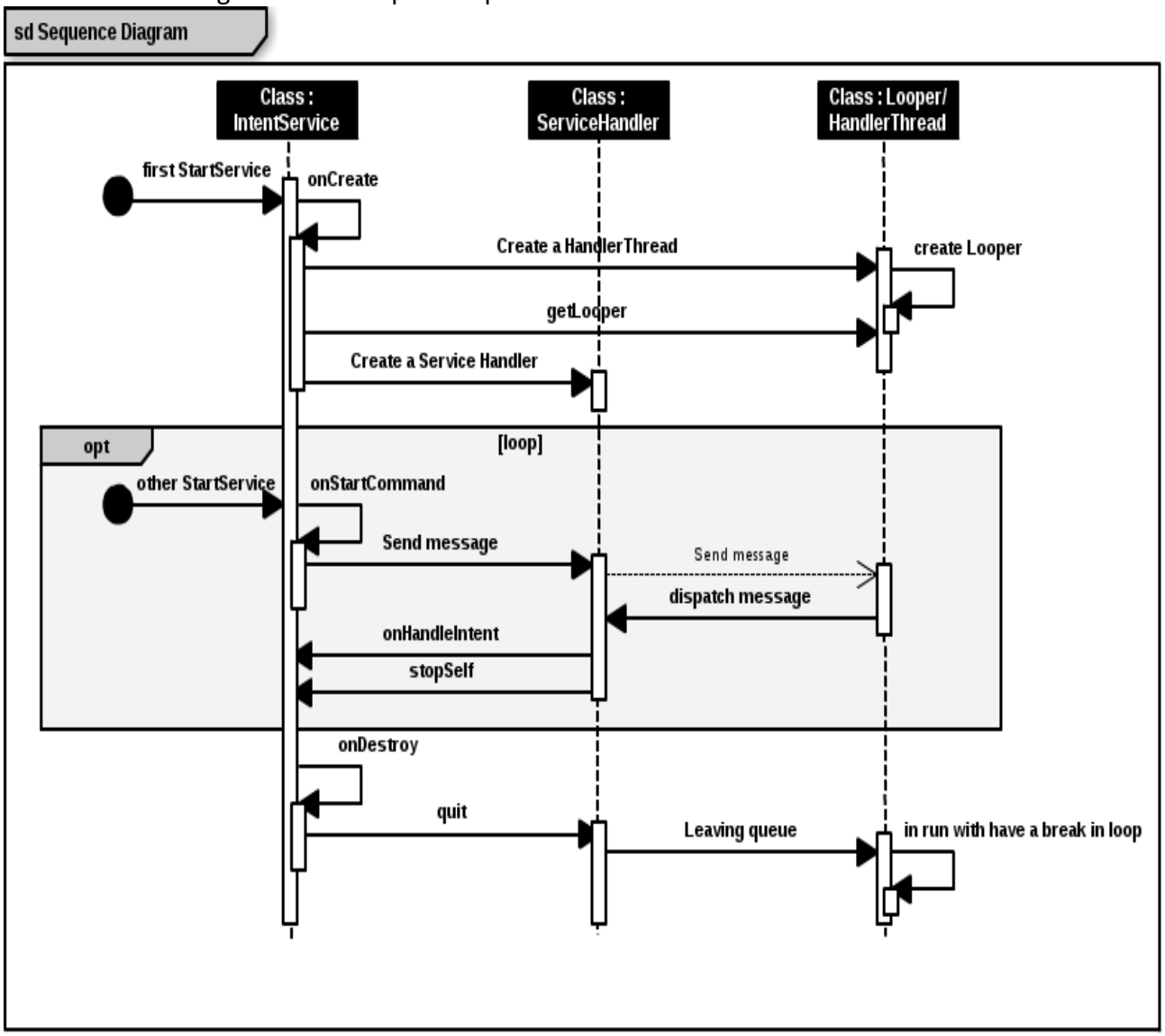
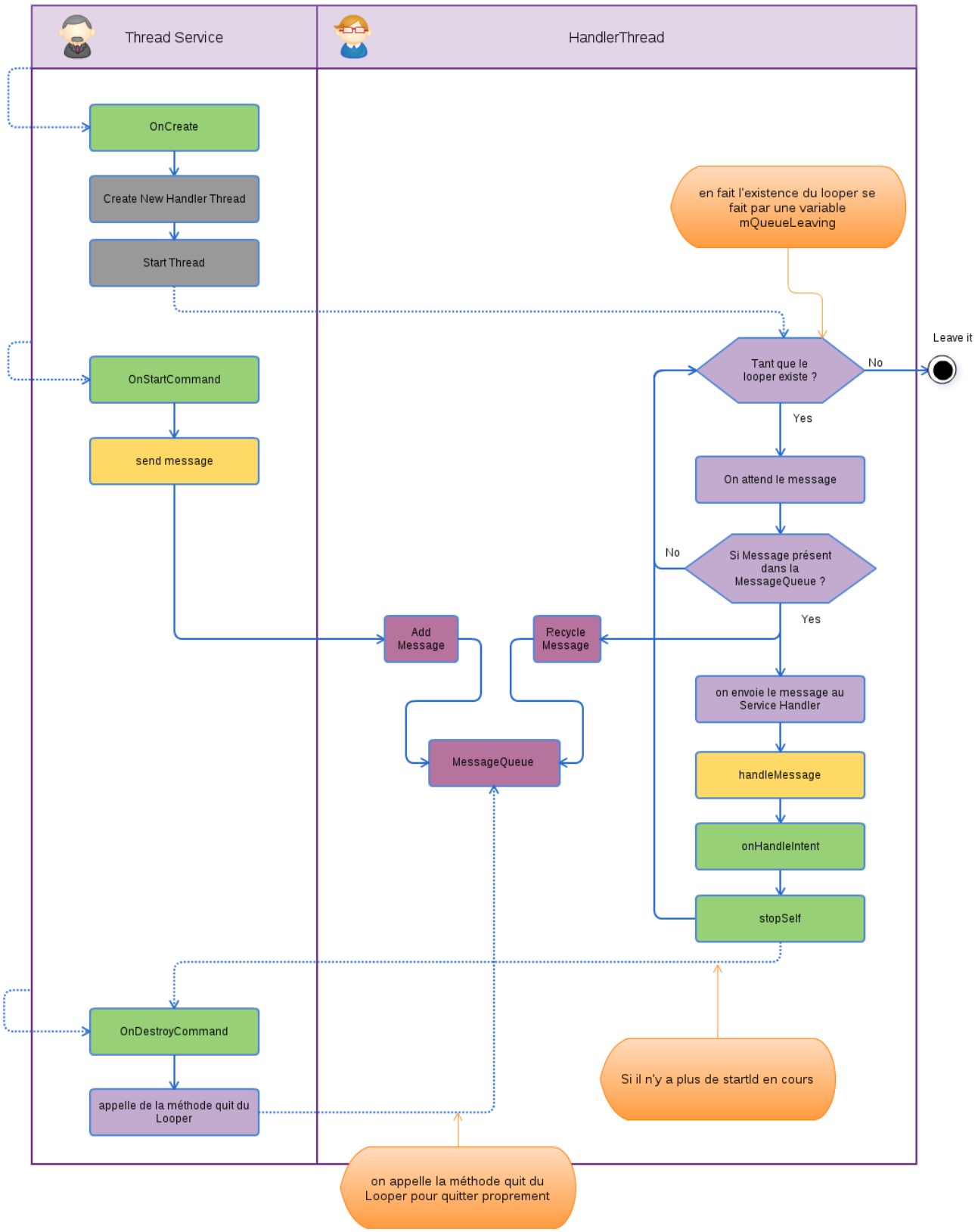


Illustration 1: Diagramme de séquence

Le fonctionnement de l'IntentService est basé sur l'envoie et la réception de message qui nous permet d'avoir un traitement asynchrone. Tout traitement de fond se réalise dans le « worker thread » (l'Handler Thread), donc dans le même thread. Par conséquent nous aurons une exécution séquentielle des traitements.

Pour comprendre le fonctionnement, voici un schéma sur les traitements par Thread :



Légende :

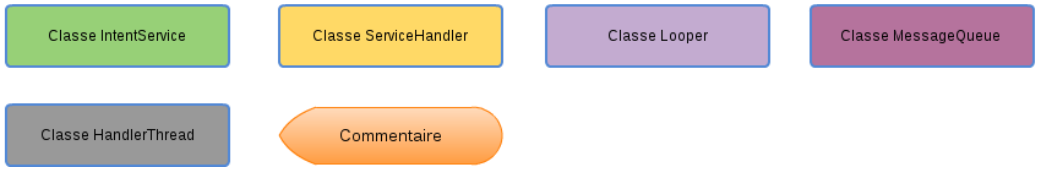


Illustration 2: Traitements par Thread

Le ThreadService est le Thread UI pour ce qui ne le savait pas.

Le traitement est réalisé depuis le « worker thread », la gestion asynchrone des messages est réalisée par le biais de la message queue et du Looper qui ira lire périoridiquement les messages présents dans la message queue.

Le service ne s'arrêtera de lui-même que si aucune intention n'est en cours (startId).

6 Tutorial

Le projet sous Github : https://github.com/MathiasSeguy-Android2EE/IntentServiceTutorial