Android : la programmation concurrente

Kotlin

* Présentation

Il est nécessaire d'utiliser la programmation concurrente dans les applications Android afin de n'avoir aucune lenteur au niveau de l'interface graphique. Il est donc nécessaire d'attribuer une tâche entière à la gestion de l'interface graphique et de ses actions rapides et des tâches parallèles dédiés aux traitements longs.

Il existe trois mécanismes permettant de répondre à ce besoin :

* Le thread : permet de créer une nouvelle tâche ne pouvant communiquer directement avec l'interface graphique.
* La tâche asynchrone : permet de créer une nouvelle tâche de manière structurée et pouvant communiquer directement avec l'interface graphique.
* Le handler : permet à plusieurs tâches parallèles de communiquer via un canal commun et standardisé vers l'interface graphique.
* Problématique
* Présentation

Lorsqu'un traitement long doit être exécuté sur une application, celui-ci doit être traité dans un nouveau thread. Effectivement, si un traitement long est exécuté sur le thread gérant l'interface graphique alors elle ne pourra être mise à jour qu'au moment où le traitement sera terminé. Ce qui aura pour effet de rendre l'utilisation de l'application désagréable.

* Démonstration

Cette démonstration présente une application avec une mauvaise gestion de son traitement long. Lorsque l'on clique sur le bouton "traitement long" l'application se fige pendant 60 secondes.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.threadwithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_hello"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Hello World!"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_hello"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long"   
        />   
   
    <Button   
        android:id="@+id/btn\_coucou"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickCoucou"   
        app:layout\_constraintTop\_toBottomOf="@id/btn\_traitement\_long"  
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Afficher coucou"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Une TextView qui affiche "HelloWorld".
* Un bouton "Traitement long" qui exécute un traitement long sur le thread de l'IHM (Thread UI).
* Un bouton "Afficher coucou" qui affiche un toast.

MainActivity.kt

package fr.acos.threadwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.util.Log   
import android.view.View   
import android.widget.Toast   
   
class MainActivity : AppCompatActivity()   
{   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long.   
     \*/   
    fun onClickTraitementLong(view: View)   
    {   
            //Exécute 60 fois   
            for (i in 1..60)   
            {   
                //Fait dormir le thread UI 1 seconde   
                Thread.sleep(1000)   
                //Affichage dans les logs de l'évolution de l'exécution   
                Log.i("XXX","Evolution : $i / 60")   
            }   
    }   
   
    /\*\*   
     \* Fonction permettant de voir la réactivité de l'IHM   
     \*/   
    fun onClickCoucou(view: View)   
    {   
        //Affichage d'un toast   
        Toast.makeText(this,"Coucou",toast.LENGTH\_LONG).show();   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickTraitementLong qui exécute un traitement long sur le thread de l'IHM. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".
* Une fonction onClickCoucou qui affiche un toast. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".

Si on clique sur le bouton "Traitement long" puis sur le bouton "Afficher coucou", le toast ne s'affichera qu'après le traitement long. Ce qui est problématique.

* Les threads
* Présentation

Les threads permettent d'exécuter plusieurs instructions en même temps. Cela permet d'avoir des instructions qui gèrent l'interface graphique et des instructions qui s'occupent des traitements longs en parallèle dans le principal but de ne pas bloquer l'interface graphique.

* Syntaxe

Voici la syntaxe du constructeur de la classe Thread :

fun thread(   
    start: Boolean = true,    
    isDaemon: Boolean = false,    
    contextClassLoader: ClassLoader? = null,    
    name: String? = null,    
    priority: Int = -1,    
    block: () -> Unit   
): Thread

Description des différents paramètres :

|  |  |
| --- | --- |
| Paramètre | Description |
| start | Si la valeur de ce paramètre est égale à true, alors le thread démarre immédiatement après sa création. |
| isDaemon | Un daemon thread en Java est un thread qui n'empêche pas la JVM de se terminer. |
| contextClassLoader | Permet de charger des classes et des ressources dans le thread. |
| name | Nom du thread. |
| priority | Priorité du thread. |
| block | Bloc d'instructions à exécuter dans le thread. C'est le seul paramètre qui est obligatoire. |

* Usage

Exemple de création d'un thread

Thread{traitementLong()}.start()

Dans cet exemple, un nouveau thread va être créé et lancé pour exécuter le traitement long de la fonction traitementLong().

* Démonstration

Cette démonstration permet de voir que le traitement long ne bloque pas l'utilisation de l'IHM.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.threadwithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_hello"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Hello World!"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_hello"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long"   
        />   
   
    <Button   
        android:id="@+id/btn\_coucou"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickCoucou"    
app:layout\_constraintTop\_toBottomOf="@id/btn\_traitement\_long"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Afficher coucou"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Une TextView qui affiche HelloWorld.
* Un bouton "Traitement long" qui exécute un traitement long sur un nouveau thread.
* Un bouton "Afficher coucou" qui affiche un toast.

MainActivity.kt

package fr.acos.threadwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.View   
import android.widget.Toast   
import kotlinx.android.synthetic.main.activity\_main.\*   
   
class MainActivity : AppCompatActivity()   
{   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long.   
     \*/   
    fun onClickTraitementLong(view: View)   
    {   
        Thread({   
            //Exécute 60 fois   
            for (i in 1..60)   
            {   
                //Fait dormir le thread contenant le traitement long   
                Thread.sleep(1000)   
                   
            }   
        }).start()   
    }    
    /\*\*   
     \* Fonction permettant de voir la réactivité de l'IHM   
     \*/   
    fun onClickCoucou(view: View)   
    {   
        //Affichage d'un toast   
        Toast.makeText(this,"Coucou",Toast.LENGTH\_LONG).show()   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickTraitementLong() qui exécute un traitement long sur le thread de l'IHM. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".
* Une fonction onClickCoucou() qui affiche un toast. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".

Si on clique sur le bouton "Traitement long" puis sur le bouton "Afficher coucou", le toast s'affichera aussitôt étant donné que le thread UI est libéré du traitement long grâce au nouveau thread.

* Problématique

La problématique avec l'utilisation des threads est qu'ils ne peuvent pas communiquer avec le thread UI. Il n'est donc pas possible d'interagir avec l'interface graphique. Une solution : runOnUiThread.

* runOnUiThread
* Présentation

runOnUiThread s'utilise en général dans un thread de traitement long et permet d'exécuter une action sur le thread UI dans le but de mettre à jour l'interface utilisateur immédiatement.

* Démonstration

Dans cette démonstration, nous allons créer un compteur qui va de 1 à 60. L'avancée du compteur est dirigée par un traitement long. C'est donc depuis le traitement long que l'on va mettre à jour le compteur en utilisant runOnUiThread.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.threadwithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_hello"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="0"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_hello"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long"   
        />   
   
    <Button   
        android:id="@+id/btn\_coucou"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickCoucou"   
   
app:layout\_constraintTop\_toBottomOf="@id/btn\_traitement\_long"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Afficher coucou"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Une TextView qui affiche les valeurs du compteur.
* Un bouton "Traitement long" qui exécute un traitement long sur un nouveau thread.
* Un bouton "Afficher coucou" qui affiche un toast.

MainActivity.kt

package fr.acos.threadwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.View   
import android.widget.Toast   
import kotlinx.android.synthetic.main.activity\_main.\*   
   
class MainActivity : AppCompatActivity()   
{   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long.   
     \*/   
    fun onClickTraitementLong(view: View)   
    {   
        Thread({   
            //Exécute 60 fois   
            for (i in 1..60)   
            {   
                //Fait dormir le thread contenant le traitement long   
                Thread.sleep(1000)   
                //Affichage de l'évolution dans le thread UI   
                runOnUiThread(   
                        {   
                    tv\_hello.text = "$i"   
                    }   
                )   
   
            }   
        }).start()   
    }   
   
    /\*\*   
     \* Fonction permettant de voir la réactivité de l'IHM   
     \*/   
    fun onClickCoucou(view: View)   
    {   
        //Affichage d'un toast   
        Toast.makeText(this,"Coucou",Toast.LENGTH\_LONG).show();   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickTraitementLong() qui exécute un traitement long sur le thread de l'IHM. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".
* Une fonction onClickCoucou() qui affiche un toast. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".

Si on clique sur le bouton "Traitement long" puis sur le bouton "Afficher coucou", le toast s'affichera aussitôt étant donné que le thread UI est libéré du traitement long grâce au nouveau thread. De plus, le TextView tv\_hello sera mis à jour en temps réel.

* Les tâches asynchrones
* Présentation

La classe AsyncTask permet de mettre en place des tâches asynchrones. Les tâches asynchrones permettent de faciliter l'utilisation des threads. Pour créer une tâche asynchrone, il est nécessaire de définir une classe héritant de la classe AsyncTask et de redéfinir les fonctions suivantes :

|  |  |
| --- | --- |
| Fonctions | Description |
| doInBackground() | Cette fonction contient le contenu du traitement long. Les instructions définies dans cette fonction sont exécutées dans un nouveau thread. |
| onPreExecute() | Cette fonction s'exécute sur le thread UI. Elle s'exécute avant doInBackground(). |
| onPostExecute() | Cette fonction s'exécute sur le thread UI. Elle s'exécute après doInBackground(). |
| onProgressUpdate() | Cette fonction s'exécute sur le thread UI. Elle s'exécute pendant le traitement long donc en même temps que doInBackground(). Cette fonction s'exécute lorsque la fonction publishProgress() est appelée depuis doInBackground(). |

Pour exécuter la tâche asynchrone, il faut créer une instance de la classe héritant de AsyncTask et utiliser la fonction execute().

* Démonstration

Cette démonstration permet de voir comment mettre en place des tâches asynchrones.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.asynctaskwithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_hello"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Hello World!"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <ProgressBar   
        android:id="@+id/pb\_evolution"   
        style="?android:attr/progressBarStyleHorizontal"   
        android:layout\_width="match\_parent"   
        android:layout\_height="wrap\_content"   
        android:orientation="horizontal"   
        android:max="20"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_hello"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong"   
        app:layout\_constraintTop\_toBottomOf="@id/pb\_evolution"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long"   
        />   
   
    <Button   
        android:id="@+id/btn\_coucou"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickCoucou"   
   
app:layout\_constraintTop\_toBottomOf="@id/btn\_traitement\_long"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Afficher coucou"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Une barre de progression qui évolue en même temps que le traitement long.
* Une TextView qui affiche "HelloWorld".
* Un bouton "Traitement long" qui exécute un traitement long sur un nouveau thread.
* Un bouton "Afficher coucou" qui affiche un toast.

MainActivity.kt

package fr.acos.asynctaskwithkotlin   
   
import android.os.AsyncTask   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.View   
import android.widget.Toast   
import kotlinx.android.synthetic.main.activity\_main.\*   
   
class MainActivity : AppCompatActivity()   
{   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    /\*\*   
     \* Fonction permettant de voir la réactivité de l'IHM   
     \*/   
    fun onClickCoucou(view: View)   
    {   
        //Affichage d'un toast   
        Toast.makeText(this,"Coucou", toast.LENGTH\_LONG).show();   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long.   
     \*/   
    fun onClickTraitementLong(view: View)   
    {   
        val traitementLong = MonTraitementLong()   
        traitementLong.execute("Enfin !")   
    }   
   
    /\*\*   
     \*   
     \* AsyncTask<A, B, C>   
     \*     A : Type des données passées au traitement long : fonction   
     \*doInBackground()   
     \*     B : type des données envoyées depuis le traitement long   
     \*au thread UI via onProgressUpdate   
     \*     C : type de la donnée retournée par le traitement long :   
     \*fonction doInBackground()   
     \*/   
    inner class MonTraitementLong() : AsyncTask<String, Int, String>() {   
        /\*\*   
         \* Fonction exécutée avant le traitement long sur le thread UI   
         \*/   
        override fun onPreExecute() {   
            super.onPreExecute()   
            Toast.makeText(this@MainActivity,"Début du traitement   
!!",Toast.LENGTH\_LONG).show()   
        }   
   
        /\*\*   
         \* Fonction exécutant le traitement long sur un nouveau thread   
         \*/   
        override fun doInBackground(vararg params: String?): String {   
            //Exécute 60 fois   
            for (i in 1..20)   
            {   
                //Fait dormir le thread contenant le traitement long   
                Thread.sleep(1000)   
                //Appel de la fonction onProgressUpdate() sur le Thread UI.   
                publishProgress(i)   
            }   
            return "Fin du traitement, message : ${params[0]}"   
        }   
   
        /\*   
         \* Fonction exécutée pendant le traitement long sur le thread UI   
         \* Cette fonction est appelée à l'appel de la fonction   
         \* publishProgress()   
         \*/   
        override fun onProgressUpdate(vararg values: Int?) {   
            // \* pour le spread operator   
            super.onProgressUpdate(\*values)   
            pb\_evolution.progress = values[0]!!.toInt()   
        }   
   
        /\*\*   
         \* Fonction exécutée aprés le traitement long sur le thread UI   
         \*/   
        override fun onPostExecute(result: String?) {   
            super.onPostExecute(result)   
            Toast.makeText(this@MainActivity,"$result",Toast  
            .LENGTH\_LONG).show()   
        }   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickTraitementLong() qui exécute la tâche asynchrone.
* Une fonction onClickCoucou() qui affiche un toast. La fonction s'exécute au moment où l'on clique sur le bouton "Traitement long".
* Une classe MonTraitementLong qui hérite de AsyncTask.

La classe MonTraitementLong contient les fonctions suivantes :

* onPreExecute qui affiche un toast avant que le traitement long commence.
* doInBackground qui contient le traitement long et lance onProgressUpdate grâce à la fonction publishProgress.
* onProgressUpdate qui fait évoluer la barre de progression.
* onPostExecute qui affiche un toast lorsque le traitement long est terminé.

Cet exemple permet de montrer que le code est bien structuré et qu'il est facile de maîtriser le moment ou s'exécute un traitement par rapport au traitement long d'un thread.

* Les Handlers
* Présentation

Un Handler permet :

* à différents threads de communiquer entre eux,
* de réceptionner les messages envoyés par des threads et ensuite d'exécuter une action sur l'IHM par exemple.

La démonstration suivante ne présente pas toutes les possibilités offertes par les handlers.

* Démonstration

Dans cette démonstration, trois traitements longs vont pouvoir être lancés. Ces traitements longs envoient des messages au handler créé sur le thread UI pour mettre à jour l'IHM. Concrètement, il y a trois diffuseurs de messages et un récepteur de messages.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.handlerwithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_hello"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Hello World!"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long1"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong1"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_hello"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long N°1"   
        />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long2"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong2"   
   
        app:layout\_constraintTop\_toBottomOf="@id/  
        btn\_traitement\_long1"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long N°2"   
        />   
   
    <Button   
        android:id="@+id/btn\_traitement\_long3"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickTraitementLong3"   
   
app:layout\_constraintTop\_toBottomOf="@id/btn\_traitement\_long2"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Traitement long N°3"   
        />   
   
    <Button   
        android:id="@+id/btn\_coucou"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:onClick="onClickCoucou"   
   
app:layout\_constraintTop\_toBottomOf="@id/btn\_traitement\_long3"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        android:text="Afficher coucou"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Une TextView qui affiche "HelloWorld".
* Un bouton "Traitement long N°1" qui exécute un traitement long sur un nouveau thread.
* Un bouton "Traitement long N°2" qui exécute un traitement long sur un nouveau thread.
* Un bouton "Traitement long N°3" qui exécute un traitement long sur un nouveau thread
* Un bouton "Afficher coucou" qui permet d'afficher un toast et de montrer que l'IHM n'est pas bloquée.

Chaque traitement long envoie un message au handler au début du traitement, au milieu du traitement et à la fin du traitement.

MainActivity.kt

package fr.acos.handlerwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.os.Handler   
import android.os.Message   
import android.util.Log   
import android.view.View   
import android.widget.Toast   
   
class MainActivity : AppCompatActivity()   
{   
    var handler = MonHandler()   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
        Log.i("XXX","GO")   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long numéro 1.   
     \*/   
    fun onClickTraitementLong1(view: View)   
    {   
        Log.i("XXX","onClickTraitementLong1")   
   
        Thread({   
            val msgDebut = Message()   
            msgDebut.what = 1   
            msgDebut.arg1 = 1   
            //Envoi d'un message au handler pour lui indiquer que   
            //le traitement commence   
            handler.sendMessage(msgDebut)   
   
            //Exécute 20 fois   
            for (i in 1..20)   
            {   
                //Fait dormir le thread contenant le traitement long   
                Thread.sleep(1000)   
                if(i == 10) {   
                    val msgMiChemin = Message()   
                    msgMiChemin.what = 2   
                    msgMiChemin.arg1 = 1   
                    //Envoi d'un message au handler pour lui   
                    //indiquer que le traitement est à la moitié   
                    //de son exécution   
                    handler.sendMessage(msgMiChemin)   
                }   
            }   
            val msgFin = Message()   
            msgFin.what = 3   
            msgFin.arg1 = 1   
            //Envoi d'un message au handler pour lui indiquer   
            //que le traitement est terminé   
            handler.sendMessage(msgFin)   
        }).start()   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long numéro 2.   
     \*/   
    fun onClickTraitementLong2(view: View)   
    {   
        Log.i("XXX","onClickTraitementLong2")   
        Thread({   
            val msgDebut = Message()   
            msgDebut.what = 1   
            msgDebut.arg1 = 2   
            //Envoi d'un message au handler pour lui indiquer que   
            //le traitement commence   
            handler.sendMessage(msgDebut)   
            //Exécute 15 fois   
            for (i in 1..15)   
            {   
                //Fait dormir le thread contenant le traitement long   
                Thread.sleep(1000)   
                if(i == 8) {   
                    val msgMiChemin = Message()   
                    msgMiChemin.what = 2   
                    msgMiChemin.arg1 = 2   
                    //Envoi d'un message au handler pour lui   
                    //indiquer que le traitement est à la moitié   
                    //de son exécution   
                    handler.sendMessage(msgMiChemin)   
                }   
            }   
            val msgFin = Message()   
            msgFin.what = 3   
            msgFin.arg1 = 2   
            //Envoi d'un message au handler pour lui indiquer   
            //que le traitement est terminé   
            handler.sendMessage(msgFin)   
        }).start()   
    }   
   
    /\*\*   
     \* Fonction contenant le traitement long  numéro 3.   
     \*/   
    fun onClickTraitementLong3(view: View)   
    {   
        Log.i("XXX","onClickTraitementLong3")   
        Thread({   
            val msgDebut = Message()   
            msgDebut.what = 1   
            msgDebut.arg1 = 3   
            //Envoi d'un message au handler pour lui indiquer   
            //que le traitement commence   
            handler.sendMessage(msgDebut)   
            //Exécute 10 fois   
            for (i in 1..10)   
            {   
                //Fait dormir le thread contenant le traitement long   
                Thread.sleep(1000)   
                if(i == 5) {   
                    val msgMiChemin = Message()   
                    msgMiChemin.what = 2   
                    msgMiChemin.arg1 = 3   
                    //Envoi d'un message au handler pour lui   
                    //indiquer que le traitement est à la moitié   
                    //de son exécution   
                    handler.sendMessage(msgMiChemin)   
                }   
            }   
            val msgFin = Message()   
            msgFin.what = 3   
            msgFin.arg1 = 3   
            //Envoi d'un message au handler pour lui indiquer que   
            //le traitement est terminé   
            handler.sendMessage(msgFin)   
        }).start()   
    }   
   
    /\*\*   
     \* Fonction permettant de voir la réactivité de l'IHM   
     \*/   
    fun onClickCoucou(view: View)   
    {   
        //Affichage d'un Toast   
        Toast.makeText(this,"Coucou", Toast.LENGTH\_LONG).show();   
    }   
   
   
    fun information(message:String)   
    {   
        Toast.makeText(this,message, Toast.LENGTH\_LONG).show();   
    }   
   
    inner class MonHandler : Handler()   
    {   
        override fun handleMessage(msg: Message) {   
            Log.i("XXX","Wololo ${msg.what}")   
            super.handleMessage(msg)   
            when (msg.what)   
            {   
                1 -> information("Un traitement long numéro   
${msg.arg1} vient de commencer")   
                2 -> information("le traitement long numéro   
${msg.arg1} est à mi-chemin")   
                3 -> information("Un traitement long numéro   
${msg.arg1} est terminé")   
            }   
        }   
    }   
}

Dans cet exemple, le contrôleur contient :

* Une fonction onClickTraitementLong1() qui contient un traitement long. Cette fonction envoie un message au handler au début du traitement :

            val msgDebut = Message()   
            msgDebut.what = 1   
            msgDebut.arg1 = 1   
            handler.sendMessage(msgDebut)

Puis au milieu du traitement :

            val msgMiChemin = Message()   
            msgMiChemin.what = 2   
            msgMiChemin.arg1 = 3   
            handler.sendMessage(msgMiChemin)

Puis à la fin du traitement :

            val msgFin = Message()   
            msgFin.what = 3   
            msgFin.arg1 = 3   
            handler.sendMessage(msgFin)

* Une fonction onClickTraitementLong2() qui a le même comportement que onClickTraitementLong1().
* Une fonction onClickTraitementLong3() qui a le même comportement que onClickTraitementLong1().
* Une classe MonHandler héritant de Handler :

inner class MonHandler : Handler()

Elle permet de créer un objet Handler qui réceptionne les messages de tous les threads grâce à la fonction handleMessage() :

override fun handleMessage(msg: Message) {   
            Log.i("XXX","Wololo ${msg.what}")   
            super.handleMessage(msg)   
            when (msg.what)   
            {   
                1 -> information("Un traitement long numéro   
${msg.arg1} vient de commencer")   
                2 -> information("le traitement long numéro   
${msg.arg1} est à mi-chemin")   
                3 -> information("Un traitement long numéro   
${msg.arg1} est terminé")   
            }   
        }

Et selon le message, la fonction met à jour l'IHM grâce à la fonction information() de l'activité MainActivity.

fun information(message:String)   
    {   
        Toast.makeText(this,message, Toast.LENGTH\_LONG).show();   
    }

* Les coroutines

Kotlin a mis au point un concept de tâches concurrentes coopératives appelé coroutine, permettant de simplifier l'écriture des threads et des tâches asynchrones et d'optimiser la mémoire grâce à un système de fonctions concurrentes suspendues et reprises. Ce concept est encore expérimental aujourd'hui et ne sera pas traité dans le livre.