PDM Avanzado Threads

Sumario

Introdución

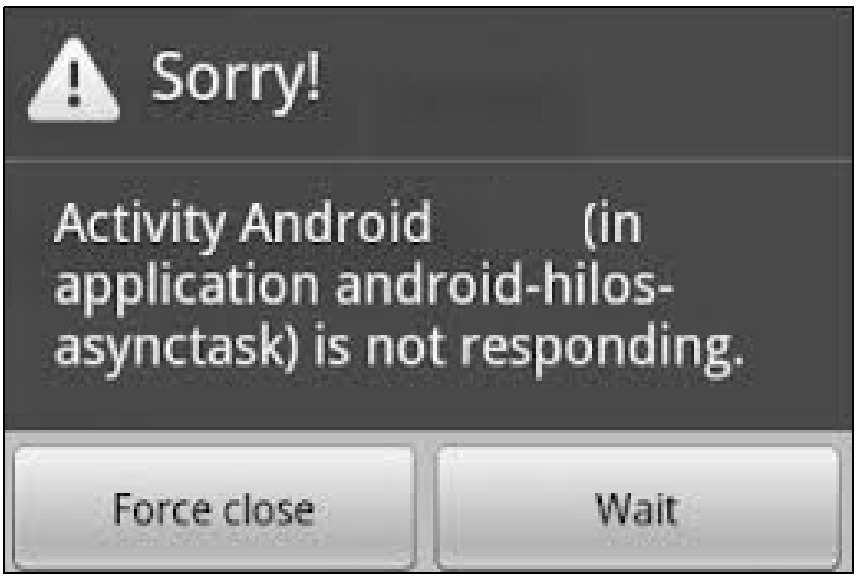
Máis información en: http://developer.android.com/guide/components/processes-and-threads.html

Os threads vannos permitir ter diferentes fíos de execución, separados do fío principal (chamado main) onde se atenden os eventos do usuario na interface.

É moi importante ter dúas consideracións cando estamos a manexar o thread principal:

- Non deixar bloqueado o thread principal, por exemplo, cargando unha imaxe.
- Non se pode acceder os elementos da interface (botóns,?.) dende un fío diferente ao principal.

O primeiro é necesario xa que podemos provocar un ANR (Application Not Responding), no que aparece un dialogbox indicando que a aplicación deixou de funcionar.



Imaxinemos que queremos cargar unha imaxe dende Internet. Isto pode levar moito tempo, e a aplicación non pode esperar a cargar a imaxe.

Para isto, creamos un novo fío de execución.

Temos varias formas de facelo,con AsyncTask e con Threads.

Threads

Cando creamos un obxecto da clase Thread podemos sobrescribir o método run ou crealo mandando no construtor un obxecto de interface runable que implemente o método run.

Por exemplo:

```
private void crearThread(){
        Thread fio = new Thread(){

             public void run(){
             for (int a=0;a<10;a++){
             try {
Thread.sleep(1000);
             Log.i("THREAD",String.valueOf(a));
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
             }
             }// FIN DO RUN
                    };
        fio.start();

}
```

Nota: A liña 7 indica que o Thread 'durma' durante 1 segundo (1000 mseg).

A outra forma sería:

```
private void crearThreadRunable(){
        Thread fio = new Thread(new Runnable() {

public void run() {
// TODO Auto-generated method stub
             for (int a=0;a<10;a++){
             try {
Thread.sleep(1000);
             Log.i("THREAD",String.valueOf(a));
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
             }// Fin do for
} //Fin do run()
   } //Fin do runnable
        ); // Fin no new Thread

        fio.start();
    }// FIN DO METODO
```

Nestes dous exemplos estamos 'parando' a execución do Thread un segundo (sleep(1000)) e imprimindo no log a valor do contador a.

Agora supoñamos que queremos modificar un texto (TextView) cos datos do contador (variable a).

Para iso temos que ter unha referencia da caixa de texto dentro do fío e **iso non é posible**.

O seguinte código da un erro xa que non podemos referenciar á caixa de texto dentro de fío.

```
private void crearThread(){
       final TextView texto = (TextView)findViewById(R.id.txtContador);
       Thread fio = new Thread(){
```

```
            public void run(){
            for (int a=0;a<10;a++){
            try {
Thread.sleep(1000);
            texto.setText(String.valueOf(a));
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
            }
            }
              };
              fio.start();

}// FIN DO crearThread
```

Para solucionalo, temos a opción de usar un Handler, que vén a ser coma unha ponte entre un fío e o fío principal.

Paso de mensaxes

Máis información en: http://developer.android.com/reference/android/os/Handler.html

Nota: A clase Handler pertence ao paquete **android.os**, cando importedes a clase escoller **android.os.Handler**.

Como indicamos antes non podemos referenciar ningún elemento da interface dentro dun fío.

Nos casos en que isto sexa necesario teremos que utilizar un **AsyncTask** ou un **Thread con paso de mensaxes**.

Un **Handler** é unha ponte para pasar información dende o Thread a un procedemento no que podemos acceder ao fío principal.

```
private Handler ponte = new Handler(){
    @Override
    public void handleMessage(Message msg) {

}

}; // Fin do Handler
```

O parámetro **msg** vai recibir información dende o fío, e dende o Handler si imos poder referenciar os elementos gráficos da UI.

```
private Handler ponte = new Handler(){
    @Override
    public void handleMessage(Message msg) {
            TextView texto = (TextView)findViewById(R.id.txtContador);
            texto.setText(String.valueOf(msg.arg1));
}
}; // Fin do Handler

private void crearThread(){

        Thread fio = new Thread(){

            public void run(){
            for (int a=0;a<10;a++){
            try {
Thread.sleep(1000);
            Message msg = new Message();
            msg.arg1=a;
                    ponte.sendMessage(msg);
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} // fin do catch
```

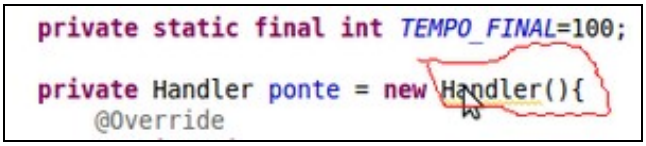
```
            } // Fin do for
            } // Fin do run
        }; // fin do new Thread
        fio.start();

}// Fin do crearThread
```

Fixarse como o msg pódese utilizar para enviar datos de moi diversas formas:

- msg.setData(data): sendo data un obxecto da clase Bundle (almacena pares da datos cos métodos setTIPODATO e recupera con getTIPODATO).
- msg.obj: un Object
- msg.arg1: un enteiro
- msg.arg2: outro enteiro.

Ata o de aquí non hai problema ou si....se o deixades así podedes ver coma vos da un aviso na clase Handle:



Para solucionar o problema temos que ter unha referencia débil da Activity na que queremos acceder ós elementos gráficos.

Debido a que o Handler manexa unha cola de mensaxes, é recomendable que sexa definido como Static, xa que dita cola é compartida por todos os obxectos handler que teñamos, é o procesar dita mensaxe non pode ser destruída polo proceso de garbagecollection se o handler non está definido como de clase.

Se cambiamos sen máis a clase a ?Static? imos atopar o problema de que cando queiramos referenciar un método ou un obxecto fora da clase Handler (que estea definido na clase Activity), non podemos facelo.

Podedes facer a proba:

```
private static Handler ponte = new Handler(){
    @Override
    public void handleMessage(Message msg) {
            TextView texto = (TextView)findViewById(R.id.txtContador);
            texto.setText(String.valueOf(msg.arg1));
}
}; // Fin do Handler
```

Neste caso sería máis conveniente utilizar a clase Asynctask (a veremos a continuación), pero a modo de curiosidade imos ver como podemos solucionar o problema:

Para solucionalo, temos que facer que exista unha referencia ?débil? da Activity principal dentro do Handler.

Para facelo temos que crear un construtor no handler onde lle imos pasar "this" como parámetro e que será a referencia débil:

Cambiamos polo tanto a forma anterior e creamos un obxecto dunha clase Handler definida previamente:

Nota: A clase activity principal neste exemplo se chama "OutraActividade". Deberedes cambiala en función do voso nome.

Código a poñer dentro da activity de nome OutraActividade.java Pasamos de:

```
private Handler ponte = new Handler(){
    @Override
    public void handleMessage(Message msg) {
            TextView texto = (TextView)findViewById(R.id.txtContador);
            texto.setText(String.valueOf(msg.arg1));
}
```

```
}; // Fin do Handler
```

A

```
private static class ClassPonte extends Handler {

private WeakReference<OutraActividade> mTarget=null;

ClassPonte(OutraActividade target) {
            mTarget = new WeakReference<OutraActividade>(target);
}

@Override
public void handleMessage(Message msg) {

OutraActividade target = mTarget.get();

            TextView texto1 = (TextView)target.findViewById(R.id.txtContador);
            texto1.setText(String.valueOf(msg.arg1));
            }
}; // Fin do Handler

private ClassPonte ponte = new ClassPonte(this);
```

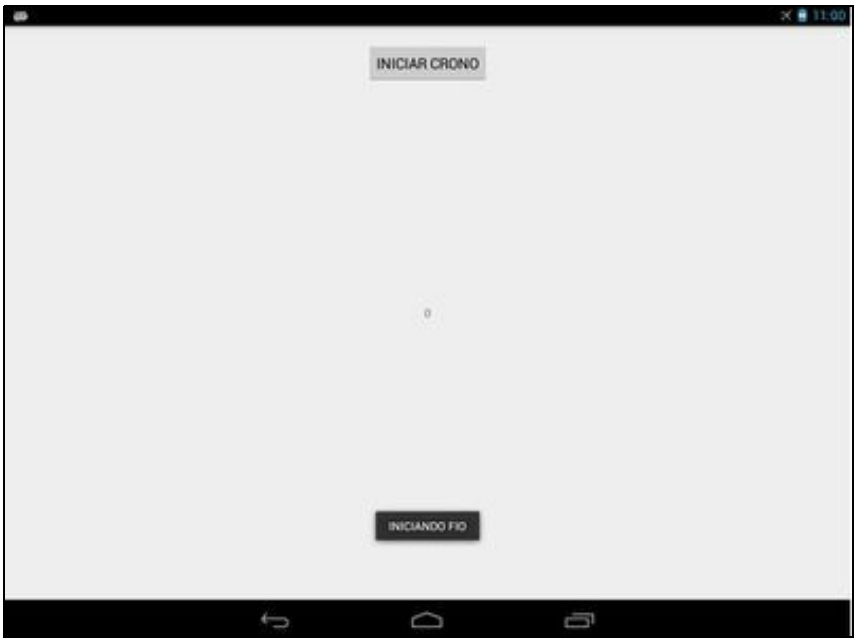
Fixarse coma dentro da clase ClassPonte, no método handleMessage podemos facer referencia a método non static grazas o obxecto target:

```
OutraActividade target = mTarget.get();
```

Caso práctico

O obxectivo desta práctica e ver como funciona un Thread e como podemos usar un Handler para pasar mensaxes dende o fío a activity e poder acceder os elementos gráficos da mesma.

Consta dun botón cun TextView onde se vai a amosar un contador. Cando se preme sobre o botón o contador empeza a funcionar.



Creamos a activity

- Nome do proxecto: **UD3_01_Threads**
- Nome da activity: **UD3_01_Threads.java**

Código do layout xml

```
<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"
    tools:context="${relativePackage}.${activityClass}" >

    <TextView
        android:id="@+id/UD3_01_txtContador"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="0" />

    <Button
        android:id="@+id/UD3_01_btnCrono"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="21dp"
        android:text="INICIAR CRONO" />

</RelativeLayout>
```

Código da clase UD3_01_Threads

Obxectivo: Utilizar un Thread con paso de mensaxes mediante Handler.

NOTA: Neste exemplo o código da clase Hander e o código da clase Thread están definidos dentro da propia Activity. Nada impide que se definan en clases separadas.

```
import java.lang.ref.WeakReference;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class UD3_01_Threads extends Activity {

private final int TEMPO_CRONO=10;

// INICO DA CLASE HANDLER
private static class ClassPonte extends Handler {

private WeakReference<UD3_01_Threads> mTarget = null;

ClassPonte(UD3_01_Threads target) {
mTarget = new WeakReference<UD3_01_Threads>(target);
}
```

```
@Override
public void handleMessage(Message msg) {

UD3_01_Threads target = mTarget.get();
TextView texto1 = (TextView) target
.findViewById(R.id.UD3_01_txtContador);

if (msg.arg2==1){
Toast.makeText(target.getApplicationContext(), "ACABOUSE O CRONO", Toast.LENGTH_LONG).show();
texto1.setText(String.valueOf(0));
}
else {
texto1.setText(String.valueOf(msg.arg1));
}
}
}; // Fin do Handler

private ClassPonte ponte = new ClassPonte(this);


private class MeuFio extends Thread{

public void run(){
             for (int a=0;a<=TEMPO_CRONO;a++){
             try {
Thread.sleep(1000);
Message msg = new Message();
msg.arg1=a;
             ponte.sendMessage(msg);
             } catch (InterruptedException e) {
// TODO Auto-generated catch block
             e.printStackTrace();
             }
             }
             Message msgFin = new Message();
             msgFin.arg2=1;
ponte.sendMessage(msgFin);
}// FIN DO RUN
};

private Thread meuFio;

private void xestionarEventos(){
Button btnCrono = (Button)findViewById(R.id.UD3_01_btnCrono);
btnCrono.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub

if ((meuFio==null) || (!meuFio.isAlive())){
Toast.makeText(getApplicationContext(), "INICIANDO FIO", Toast.LENGTH_LONG).show();
meuFio = new MeuFio();
meuFio.start();
}
else {
Toast.makeText(getApplicationContext(), "NON TE DEIXO INICIAR O FIO ATA QUE REMATE :)", Toast.LENGTH_LONG).show();
}
}
});
}

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.activity_ud3_01__threads);

        xestionarEventos();
    }
}
```

- Liña 15: Definimos o tempo do crono. Cando chegue a ese valor para.

- Liñas 18-41: Definimos a clase Handler que vai capturar as mensaxes enviadas dende o Thread.
    - ◊ Liñas 33-36: Utilizamos a propiedade arg2 do Message para saber cando acaba o contador (chega ao valor indicado na liña 15). Dende o Thread imos enviar un valor igual a 1 para indicalo.
    - ◊ Liña 38: Asinamos o valor enviado na propiedade arg1 do Message ao EditText da activity principal.
- Liña 43: Definimos o obxecto da clase Handler que vai utilizar o Thread.
- Liñas 46-64: Definimos a clase Thread que vai utilizar a activity principal para o contador.
    - ◊ Liña 53: Dentro do bucle enviamos en Message na propiedade arg1 o valor do contador.
    - ◊ Liñas 60-62: Cando sae do bucle enviamos na propiedade arg2 o valor 1 para indicar á ponte que xa rematou o fío.
- Liña 66: Definimos o obxecto da clase Thread.
- Liña 73: Xestionamos o evento Click do botón.
    - ◊ Liñas 76-80: En caso de que o fío non se iniciara nunca ou se xa rematou, instanciamos un novo fío.

-- Ángel D. Fernández González e Carlos Carrión Álvarez -- (2014).