Android 線程模式簡介

基礎概念

進程與線程

在 Android 框架裡，一個應用套件(Application Package)通常含有多個 Java 類(Class)，這些類可以在同一個進程(Process)裡執行；也可以在不同的進程裡執行。基於 Linux 的安全限制，以及進程的基本特性(例如，不同進程的位址空間是獨立的)，如果兩個類(或其對象)在同一個進程裏執行時，兩者溝通方便也快速。但是，當它們分別在不同的進程裡執行時，兩者溝通就屬於 IPC 跨進程溝通了，不如前者方便，也慢些。

一個進程是一個獨立的執行空間，不會被正在其他進程裡的程序所侵犯。這種保護方法是 Android 的重要安全機制。於是，得先認識進程的內涵，才能進一步了解跨進程 IPC(Inter-Process Communication)機制。

在Android的進程裡，有一個虛擬機(Virtual Machine，簡稱VM)的對象，可執行Java代碼，也引導JNI本地程序的執行，實現Java與C/C++程序之間的溝通；如下圖：

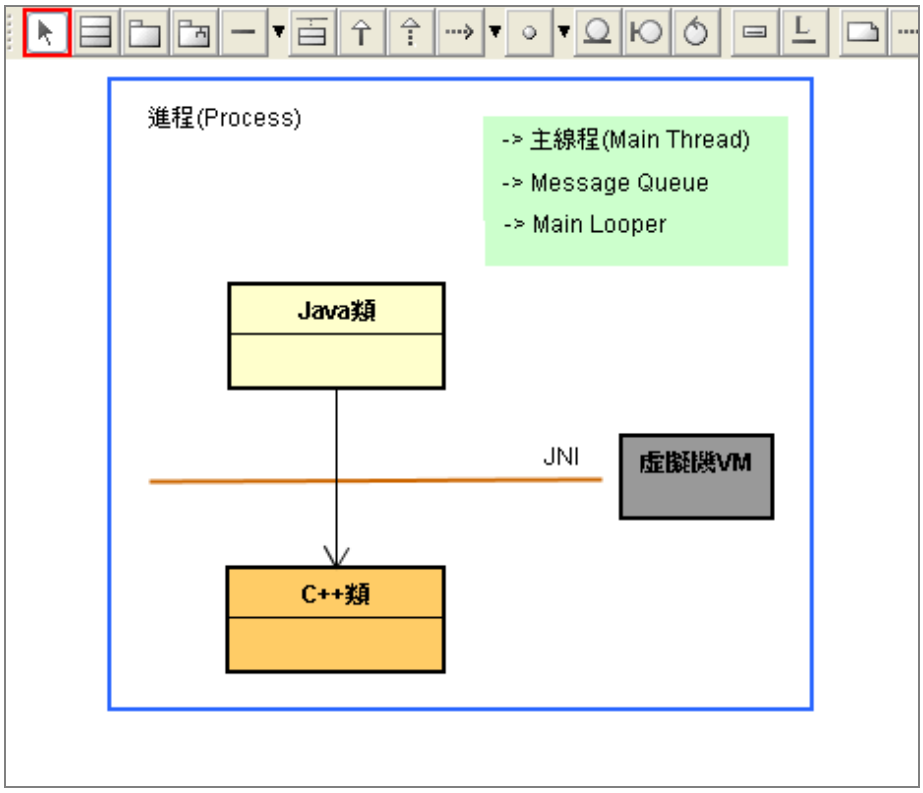


圖 1　Android 進程裡的基本元素

每一個進程在誕生時，都會誕生一個主線程(Main Thread)，以及誕生一個Looper類的對象和一個MQ(Message Queue)資料結構。每當主線程作完事情，就會去執行Looper類。此時，不斷地觀察MQ的動態。如下圖：

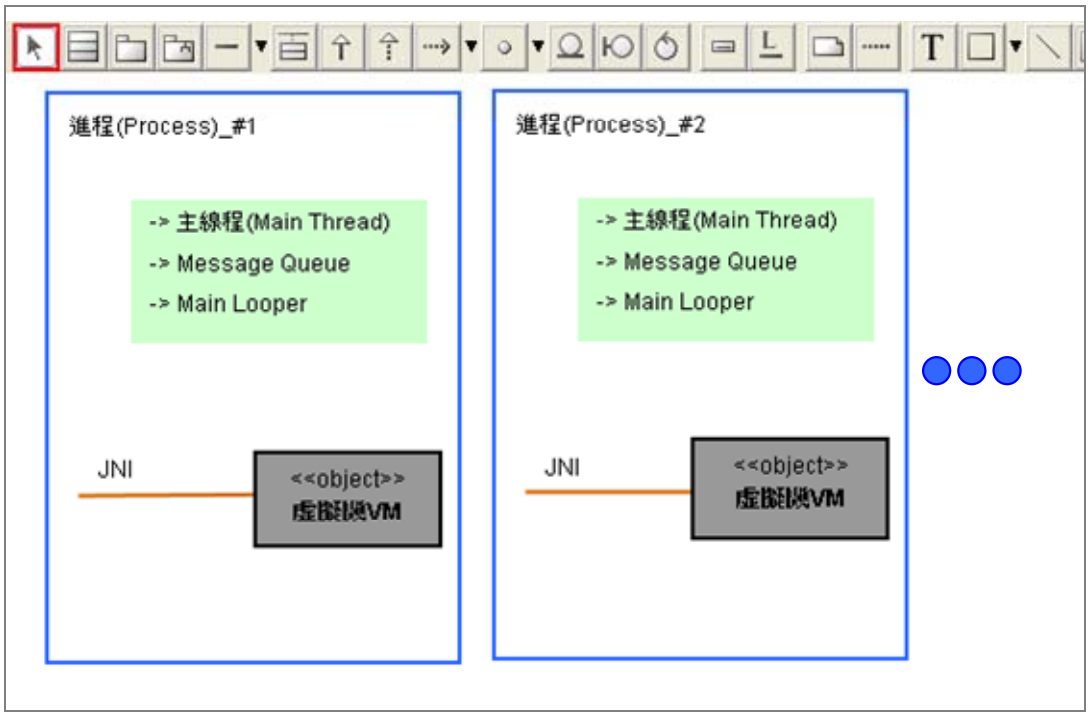


圖 2　Android 內部含有多個進程

主線程最主要的工作就是處理UI畫面的事件(Event)，每當UI事件發生時，Android框架會丟信息(Message)到MQ裡。主線程看到MQ有新的信息時，就取出信息，然後依據信息內容而去執行特定的函數。執行完畢，就再繼續執行Looper類，不斷地觀察MQ的動態。

大家都知道，當兩個類都在同一個進程裡執行時，兩者之間的溝通，只要採取一般的函數調用(Function Call)就行了，既快速又方便。一旦兩個類分別在不同的進程裡執行時，兩者之間的溝通，就不能採取一般的函數調用途徑了。只好採取 IPC 溝通途徑，如下圖：

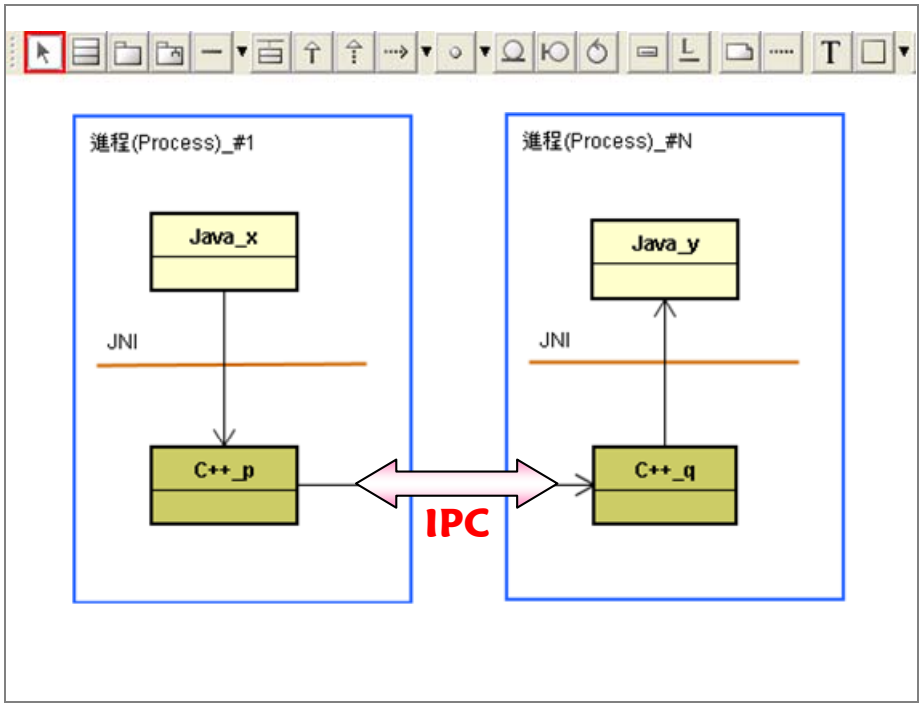


圖 3　Android 框架的 IPC 機制之例

Android 框架的 IPC 溝通仰賴單一的 IBinder 接口。此時 Client 端調用 IBinder 接口的 transact()函數，透過 IPC 機制而調用到遠方(Remote)的 onTransact()函數。例如下圖裡的 myActivity1、myActivity2 和 myService 分別在不同的進程裡執行，透過 C++層的 IBinder 接口進行跨進程的 IPC 溝通。

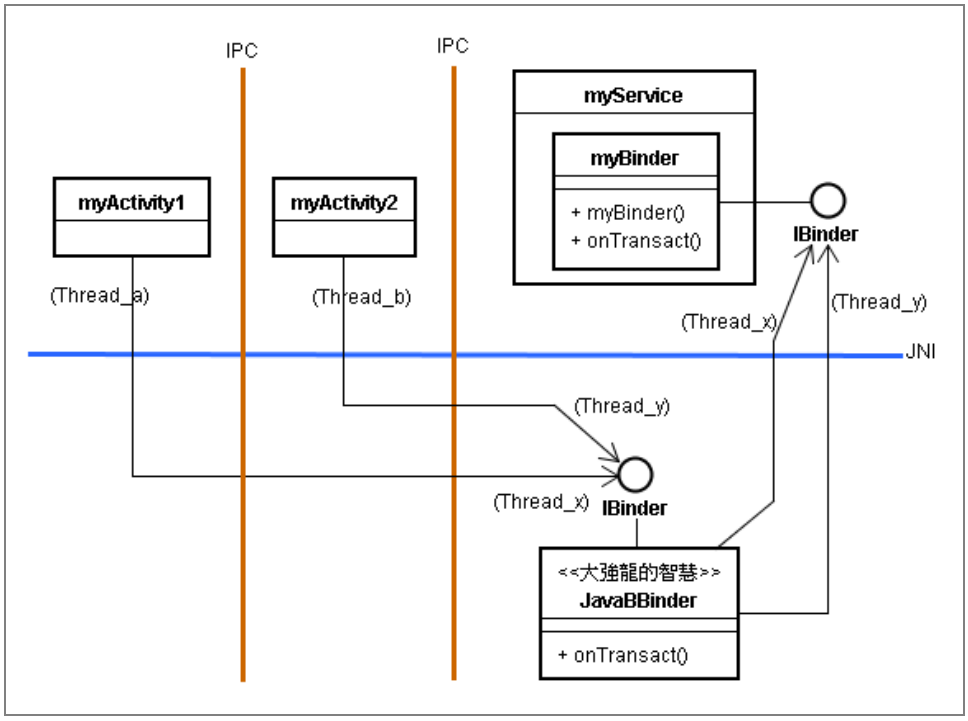


圖 4　Android IPC 機制的 IBinder 接口

在此圖的不同進程裡，各有其主線程(Thread)，這些線程可並行(Concurrent)執行，形成多線程(Multiple-Thread)的執行環境。例如上圖 4-14，myActivity1 和 myActivity2 並行執行，並透過 C++層的 JavaBBinder 類而共享(可能並行)了 Java 層的 myService 類之服務。

認識主線程

在 Android 裏，於預設情形下，一個應用程式內的各元件(如 Activity、BroadcastReceiver 或 Service 等)都會在同一個進程(Process)裏執行，而且由該進程的主線程負責執行之。在 Android 裏，如果有特別指示，也可以讓特定元件在不同的進程裏執行。無論元件在那一個進程裏執行，於預設情形下，他們都是由該進程裏的主線程來負責執行之。例如下述的範例，由一個 Activity 啓動一個 Service，兩者都在同一個進程裏執行。

那麼，主線程除了要處理 Activity 元件的 UI 事件，又要處理 Service 幕後服務工作，通常會忙不過來。該如何化解這種困境呢？此時，多線程(Multi-thread)的並行(Concurrent)概念了，其可以化解主線程太過於忙碌的情形。也就是說，主線程可以誕生多個子線程來分擔其工作，尤其是比較冗長費時的幕後服務工作，例如播放動畫的背景音樂、或從網路下載映片等。於是，主線程就能專心於處理 UI 畫面的事件了。

關於 Remote Service

剛才的範例裏的 Activity、Service 和 BroadcastReceiver 三者都是由該 APK 的預設進程裏執行。由於三者都是在同一進程裏執行，所以它們之間的通訊是屬於進程內的短距通訊。同時，也都由該預設進程裏的主線程負責執行之。

那麼，如果 Activity、Service 和 BroadcastReceiver 三者並不是在同一個進程裏執行時，它們之間的通訊就是跨進程通訊(IPC, Inter-Process Communication)了。當 Activity 與 Service(或 BroadcastReceiver)之間採用 IPC 通訊時，意味著兩者分別在不同的進程裏執行，此時基於一般原則：

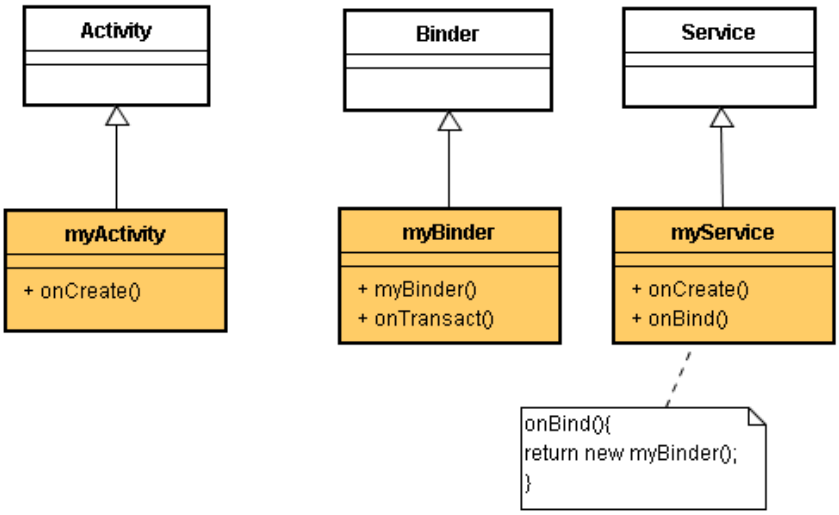
『於預設情形下，Activity、BroadcastReceiver 或 Service 都是由其所屬進程裏的主線程負責執行之。』

可知，雙方是分別由不同(進程)的主線程來執行之。請先看個範例，它由一個 Activity 啓動一個遠距的 Service，兩者分別在不同的進程裏執行。如下述 XML 檔案內容：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

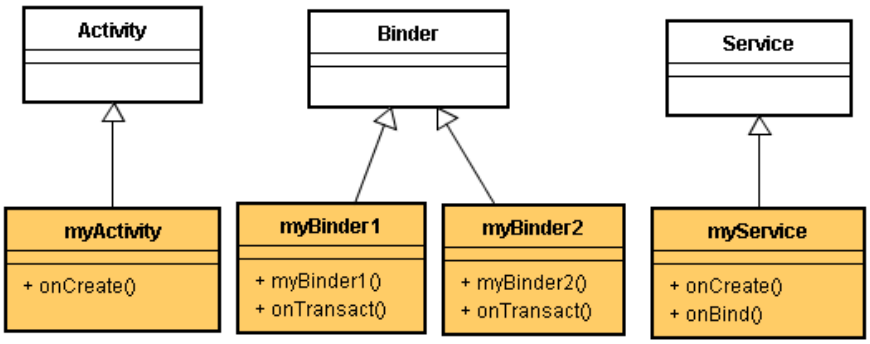
```
    package="com.misoo.kxaa">
    <application android:icon="@drawable/icon">
        <activity android:name=".ac01" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".myService" android:process=":remote">
            <intent-filter>
                <action android:name="com.misoo.kxaa.REMOTE_SERVICE" />
            </intent-filter>
        </service>
    </application>
</manifest>
```

從其中可看到，ac01 是在此應用程式的預設進程裏執行的，而 myService 是在名爲”remote”的進程裏執行的。所以 ac01 是由該預設進程的主線程所執行的，而 myService 則是由 remote”進程的主線程所執行的。此應用程式共含有兩個進程：預設進程和 remote”進程。請繼續看下圖：



其中，myActivity 與 myService 各在不同進程裏執行，兩者都是由各進程的 main thread 所執行。亦即，兩者是由不同的線程所執行。此情形下，兩個類別裏的函數也不宜太費時(例如不宜超過 5 秒鐘)；但必要時可誕生子線程去執行較費時的函數。

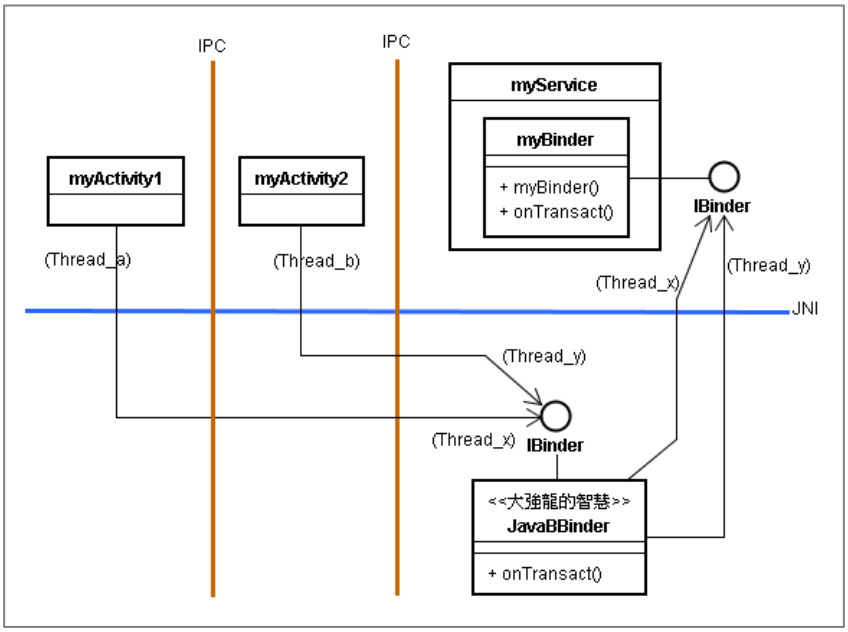
由於 new myBinder()指令寫在 myService::onCreate()內，所以是由 main thread 執行建構式 myBinder()。在 Binding-time 時，Binder System 會從進程的線程池(Thread pool)裏啓動一個線程來執行 myBinder::onTransact()函數。以此類推，下圖的 myBinder1::onTransact()與 myBinder1::onTransact()兩個函數，是由不同的線程分別執行之。



就 Android 的 Java 層應用程式開發者而言，他們可能不會太重視上述的線程機制。然而，對於軟硬整合元件開發者而言，就非常重要了。

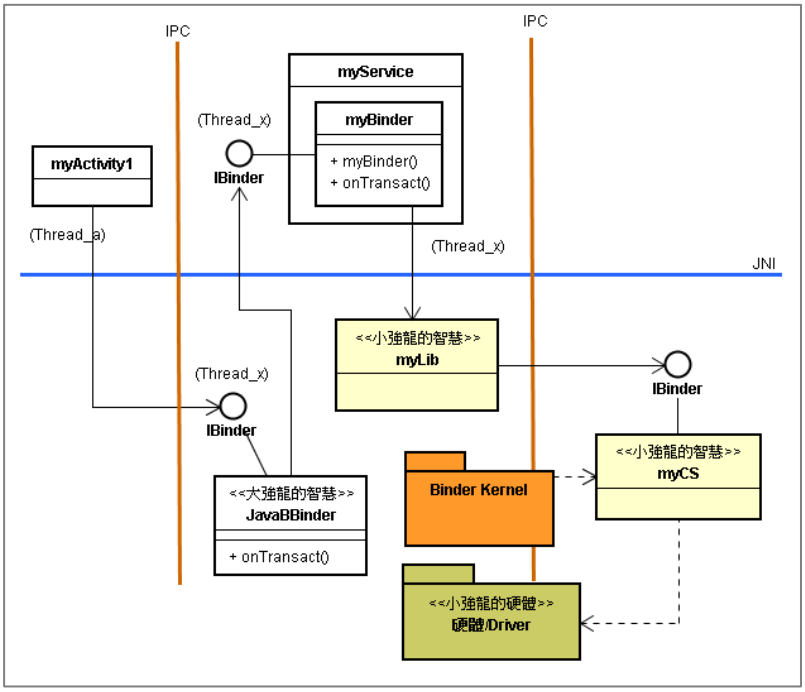
結語

- Binding-time 時，Binder System 會建立 myActivity 與 myBinder(即 myService 的 Interface)之間的連結(Connection)。
- 在 IPC calling-time 時，每次 IPC call，Binder System 會從 Service 進程的 Thread pool 裏啓動一個 Thread 來對應 myActivity 的線程。
- 在 Binding-time 時，Binder System 會建立 myActivity 與 myBinder(即 myService 的 Interface)之間的連結(Connection)。
- 在 IPC calling-time 時，myActivity 的線程與 myBinder 的線程會同步(Synchronize)，讓 myActivity 開發者覺得 IPC 遠端呼叫、跨進程的兩個線程，就如同單一線程一般。如下圖：



- 如果從 Binder 衍生了 myBinder1、myBinder2 和 myBinder3 等子類時，如何替 myService 選擇適當的 myBinder 介面類別呢？
- 如果連續呼叫 bindService()兩次，會 bind 到同一個 myBinder 物件。

- 如果想 Bind 到另一個 myBinder 介面類別之物件，可先 unbind()，就會呼叫到 onBind()函數，來決定 bind 到哪一個物件。
- 其他還有更多變化的結構，如下圖：



主、子線程的通訊模式

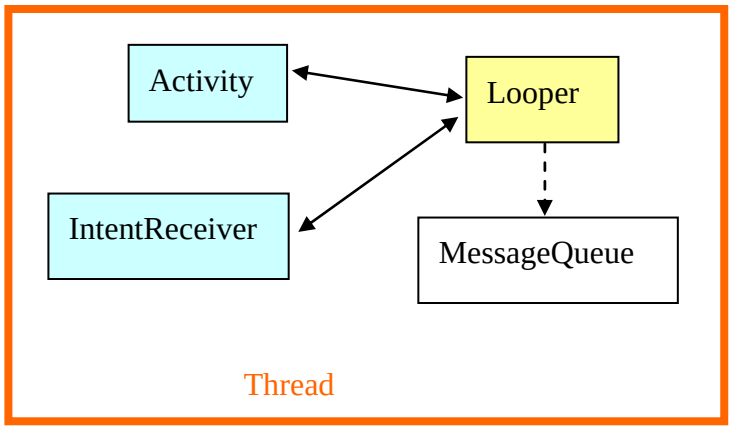
Message Queue 的角色

- 在 Android 程式裏，新誕生一個線程，或稱執行緒(Thread)時，並不會自動建立其 Message Loop。
- Android 裏並沒有 Global 的 Message Queue 資料結構，例如，不同 APK 裏的物件不能透過 Massage Queue 來交換訊息(Message)。
- 一個線程可以誕生一個 Looper 之物件，由它來管理此線程裏的 Message Queue。
- 你可以誕生 Handler 之物件來與 Looper 溝通，以便 push 新訊息到 Message Queue 裏；或者接收 Looper(從 Message Queue 取出)所送來的訊息。
- 線程 A 的 Handler 物件參考可以傳遞給別的線程，讓別的線程 B 或 C 等能送訊息來給線程 A(存於 A 的 Message Queue 裏)。
- 線程 A 的 Message Queue 裏的訊息，只有線程 A 所屬的物件可以處理之。
- 使用 Looper.myLooper 可以取得目前線程的 Looper 物件參考值。
- 使用 mHandler = **new** EevntHandler(Looper.*myLooper*()); 可誕生用來處理目前線程的 Handler 物件；其中，EevntHandler 是 Handler 的子類別。
- 使用 mHandler = **new** EevntHandler(Looper.*getMainLooper()*); 可誕生用來處

理 main 線程的 Handler 物件；其中，EevntHandler 是 Handler 的子類別。

範例之一：Looper 物件之角色

Looper 類別用來管理特定線程內物件之間的訊息交換(Message Exchange)。你的應用程式可以誕生許多個線程，或稱執行緒(Thread)。而一個線程可以誕生許多個物件，這些物件之間常常需要互相交換訊息。如果有這種需要，您可以替線程誕生一個 Looper 類別之物件，來擔任訊息交換的管理工作。Looper 物件會建立一個 MessageQueue 資料結構來存放各物件傳來的訊息(包括 UI 事件或 System 事件等)。如下圖：



每一個線程(Thread，或稱「執行緒」)裏可含有一個 Looper 物件以及一個 MessageQueue 資料結構。在你的應用程式裏，可以定義 Handler 的子類別來接收 Looper 所送出的訊息。

//----- Looper_01 範例 -----

```
package com.misoo.kx04;
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class ac01 extends Activity implements OnClickListener {
    private final int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
    private final int FP = LinearLayout.LayoutParams.FILL_PARENT;
    public TextView tv;
    private EventHandler mHandler;
```

```
private Button btn, btn2, btn3;

public void onCreate(Bundle icicle) {
                super.onCreate(icicle);
                LinearLayout layout = new LinearLayout(this);
                layout.setOrientation(LinearLayout.VERTICAL);

                btn = new Button(this);
                btn.setId(101);
                btn.setBackgroundResource(R.drawable.heart);
                btn.setText("test looper");
                btn.setOnClickListener(this);
                LinearLayout.LayoutParams param =
                      new LinearLayout.LayoutParams(100,50);
                param.topMargin = 10;
                layout.addView(btn, param);

                btn2 = new Button(this);
                btn2.setId(102);
                btn2.setBackgroundResource(R.drawable.ok_blue);
                btn2.setText("exit");
                btn2.setOnClickListener(this);
                layout.addView(btn2, param);

                tv = new TextView(this);
                tv.setTextColor(Color.WHITE);
                tv.setText("");
                LinearLayout.LayoutParams param2 =
                    new LinearLayout.LayoutParams(FP, WC);
                param2.topMargin = 10;
                layout.addView(tv, param2);
                setContentView(layout);
               }
      public void onClick(View v) {
      switch(v.getId()){
      case 101:
            Looper looper;
              looper = Looper.myLooper();
              mHandler = new EventHandler(looper);
              mHandler.removeMessages(0);
              // 清除整個MessageQueue裏的事件，確保不會通知到別人
              String obj = "This my message!";
              Message m = mHandler.obtainMessage(1, 1, 1, obj);
              // 組裝成一個Message物件
              mHandler.sendMessage(m);
              // 將Message物件送入MessageQueue裏
            break;
      case 102:
            finish();
            break;
```

```
            }
        }
//-----------------------------------------------
class EventHandler extends Handler
            {
                public EventHandler(Looper looper) {
                    super(looper);
                }
                @Override
                public void handleMessage(Message msg) {
                    tv.setText((String)msg.obj);
            }
        }
}
```

說明

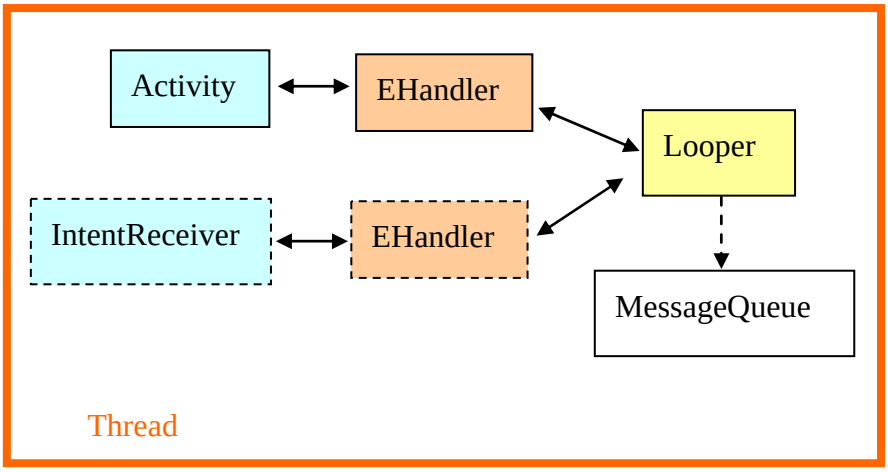
此程式啓動時，目前線程(即主線程, main thread)已誕生了一個Looper物件，並且有了一個MessageQueue資料結構。

指令：looper = Looper.*myLooper*();

就呼叫Looper類別的靜態myLooper()函數，以取得目前線程裏的Looper對象之參考值。

指令：mHandler = **new** EventHandler(looper);

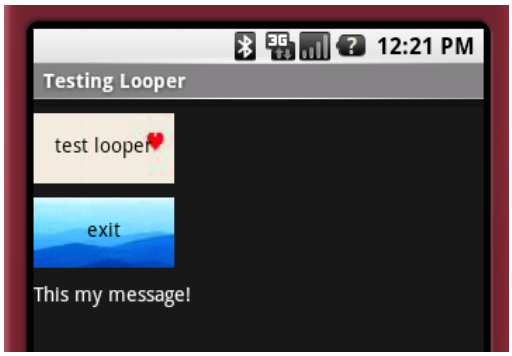
誕生一個EventHandler之物件來與Looper溝通。Activity等物件可以藉由EventHandler物件來將訊息傳給Looper，然後放入MessageQueue裏；EventHandler物件也扮演Listener的角色，可接收Looper物件所送來的訊息。如下圖：



指令：Message m = mHandler.obtainMessage(1, 1, 1, obj);

先誕生一個Message物件，並將資料存入次物件裏。

指令：mHandler.sendMessage(m);
就透過mHandler物件而將訊息m傳給Looper，然後放入MessageQueue裏。
此時，Looper物件看到MessageQueue裏有訊息m，就將它廣播出去，mHandler物件接到此訊息時，會呼叫其handleMessage()函數來處理之，於是輸出"This my message!"於畫面上，如下：



範例之二：由別的線程送訊息到主線程的 Message Queue

//----- Looper_02 範例 -----

```
package com.misoo.kx04;
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class ac01 extends Activity implements OnClickListener {
	private final int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
	private final int FP = LinearLayout.LayoutParams.FILL_PARENT;
	public TextView tv;
	private myThread t;
	private Button btn, btn2, btn3;

	public void onCreate(Bundle icicle) {
			super.onCreate(icicle);
			LinearLayout layout = new LinearLayout(this);
			layout.setOrientation(LinearLayout.VERTICAL);

			btn = new Button(this);
```

```
                    btn.setId(101);
                    btn.setBackgroundResource(R.drawable.heart);
                    btn.setText("test looper");
                    btn.setOnClickListener(this);
                    LinearLayout.LayoutParams param =
                          new LinearLayout.LayoutParams(100,50);
                    param.topMargin = 10;
                    layout.addView(btn, param);

                    btn2 = new Button(this);
                    btn2.setId(102);
                    btn2.setBackgroundResource(R.drawable.ok_blue);
                    btn2.setText("exit");
                    btn2.setOnClickListener(this);
                    layout.addView(btn2, param);

                    tv = new TextView(this);
                    tv.setTextColor(Color.WHITE);
                    tv.setText("");
                    LinearLayout.LayoutParams param2 =
                        new LinearLayout.LayoutParams(FP, WC);
                    param2.topMargin = 10;
                    layout.addView(tv, param2);
                    setContentView(layout);
                  }
        public void onClick(View v) {
        switch(v.getId()){
        case 101:
              t = new myThread();
              t.start();
              break;
        case 102:
             finish();
             break;
        }
    }
//-------------------------------------------------
class EHandler extends Handler {
             public EHandler(Looper looper) {
                  super(looper);
             }
             @Override
             public void handleMessage(Message msg) {
                 tv.setText((String)msg.obj);
        }
    }
//-------------------------------------------------
class myThread extends Thread{
      private EHandler mHandler;
      public void run() {
```

```
            Looper myLooper, mainLooper;
            myLooper = Looper.myLooper();
            mainLooper = Looper.getMainLooper();
            String obj;
            if(myLooper == null){
                    mHandler = new EHandler(mainLooper);
                    obj = "current thread has no looper!";
            }
            else {
                  mHandler = new EHandler(myLooper);
                  obj = "This is from current thread.";
            }
            mHandler.removeMessages(0);
            Message m = mHandler.obtainMessage(1, 1, 1, obj);
            mHandler.sendMessage(m);
        }
    }
}
```

Android會自動替主線程建立Message Queue。在這個子線程裏並沒有建立Message Queue。所以，myLooper值爲null，而mainLooper則指向主線程裏的Looper物件。於是，執行到指令：

mHandler = **new** EHandler(mainLooper); 此mHandler屬於主線程。

指令：mHandler.sendMessage(m);

就將m訊息存入到主線程的Message Queue裏。mainLooper看到Message Queue裏有訊息，就會處理之，於是由主線程執行到mHandler的handleMessage()函數來處理訊息。

由主線程送訊息給子線程

上述範例裏，是由子線程丟訊息給主線程。本節將介紹如何從主線程丟訊息給子線程。其方法是：當子線程執行 run()函數時，就誕生一個子線程的 Handler 物件。之後，當主線程執行 ac01::onClick()函數時，就藉由此 Handler 物件參考而 push 訊息給子線程。例如下述範例：

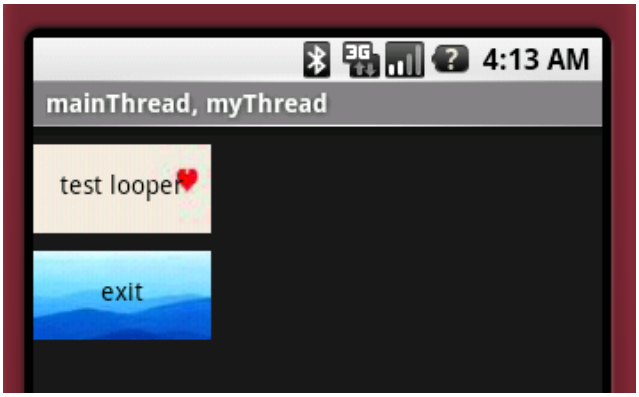
//----- Looper_04 範例 -----

```
package com.misoo.kx04;
import android.app.Activity;
import android.content.Context;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
```

```
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class ac01 extends Activity implements OnClickListener {
	private final int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
	private final int FP = LinearLayout.LayoutParams.FILL_PARENT;
	public TextView tv;
	private myThread t;
	private Button btn, btn2;
	private Handler h;
	private Context ctx;
	public void onCreate(Bundle icicle) {
			super.onCreate(icicle);
			ctx = this;
			LinearLayout layout = new LinearLayout(this);
			layout.setOrientation(LinearLayout.VERTICAL);

			btn = new Button(this);
			btn.setId(101);
			btn.setBackgroundResource(R.drawable.heart);
			btn.setText("test looper");
			btn.setOnClickListener(this);
			LinearLayout.LayoutParams param =
				new LinearLayout.LayoutParams(100,50);
			param.topMargin = 10;
			layout.addView(btn, param);

			btn2 = new Button(this);
			btn2.setId(102);
			btn2.setBackgroundResource(R.drawable.ok_blue);
			btn2.setText("exit");
			btn2.setOnClickListener(this);
			layout.addView(btn2, param);

			tv = new TextView(this);
			tv.setTextColor(Color.WHITE);
			tv.setText("");
			LinearLayout.LayoutParams param2 =
			    new LinearLayout.LayoutParams(FP, WC);
			param2.topMargin = 10;
			layout.addView(tv, param2);
			setContentView(layout);
			//-----------------------
			t = new myThread();
			t.start();
		}
```

```
            public void onClick(View v) {
            switch(v.getId()){
            case 101:
                    String obj = "mainThread";
                    Message m = h.obtainMessage(1, 1, 1, obj);
                    h.sendMessage(m);
                    break;
            case 102:
                   h.getLooper().quit();
                   finish();
                   break;
            }
      }
//-----------------------------------------
public class EventHandler extends Handler {
                public EventHandler(Looper looper) {
                                super(looper);
                 }
                  @Override
                   public void handleMessage(Message msg) {
                                ((Activity)ctx).setTitle((String)msg.obj);
                    }
            }
//-----------------------------------------
class myThread extends Thread{
        public void run() {
              Looper.prepare();
              h = new Handler(){
                        public void handleMessage(Message msg) {
                                EventHandler ha = new
                                  EventHandler(Looper.getMainLooper());
                                String obj = (String)msg.obj + ", myThread";
                                Message m = ha.obtainMessage(1, 1, 1, obj);
                                ha.sendMessage(m);
              }
              };
              Looper.loop();
        }
   }
}
```

當子線程執行 run()函數時，誕生一個主線程的 EventHandler 物件，並且藉之而 push 訊息給主線程了。就進行了兩個線程之間的互相交換訊息，也是兩個函數或物件間之交換訊息。此程式輸出畫面爲：



~~ END ~~