# Android 的 IPC 機制與 AIDL

## 認識 Android 的 IPC 溝通

### IPC 與 IBinder 接口

IPC(Inter-Process Communication)溝通，就是跨越兩個不同進程(Process)之通訊。

一般而言，一個 Android 應用程式裏的各組件(如 Activity、Service 等)都在同一個進程裡執行。這種在同一進程內的溝通，又稱爲「短程溝通」，意味著，兩個 Activity 在同一個進程(Process)裡執行。相對地，「遠程溝通」的意思是：兩個 Activity 分別在不同的進程裡執行；兩者之間是跨進程(Inter-Process Communication)的溝通，簡稱 IPC 溝通。

然而，在 Android 裡，一個套件(Package)可以含有多個組件(如 Activity、Service 等)，這些組件可以在同一個進程裡執行；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)，兩個組件都在同一個進程裡執行時，雙方溝通既方便又快速。但是，當兩個阻件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。如果，AndroidManifest.xml 內容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.misoo.ex01_09"
        android:versionCode="1"
        android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <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>
        <activity android:name=".Activity_2" >
        </activity>
    </application>
</manifest>
```

這指定了兩個 Activity 組件：ac01 和 Activity_2 是在同一進程裡執行。如果改爲：

```
<activity android:name=".Activity_2" android:process=":remote">
        .....
```

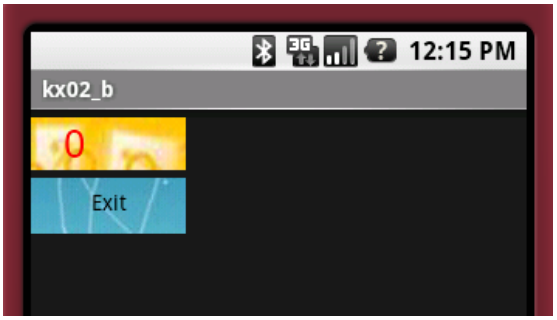
</activity>

則 Activity_2 會在獨立的進程裡執行。於是兩者就必須使用 IPC 機制才能進行跨進程的溝通。一般而言，近程溝通之效率比遠程溝通高很多。因而 Android 裡含有一個 Binder 系統，它提供高效率的遠程溝通。Android 框架裡定義了 IBinder 接口讓雙方進行 IPC 的遠程呼叫。除了 Activity 之間，Activity 與 Service 之間也常見遠程溝通。

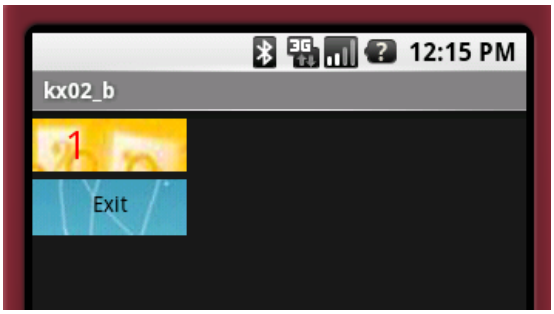
## 範例(一)：Activity 與 Service 之短程溝通

### 操作情境

1. 首先出現ac01畫面，立即啓動myService，定時連續傳來數字，如下：




2. 由於定時連續傳來數字，所以數字會持續遞增，如下：




3. 按下<Exit>按鈕，就結束ac01畫面。

### 撰寫步驟

Step-1: 建立 Android 專案：kx02_b。

Step-2: 定義 IListener接口：

```
/* ---- IListener.java ----*/
package com.misoo.ppvv;
public interface IListener {
	public void update(String s);
}
```

Step-3: 撰寫一個 Row 類別，以配合 listAdapter 類別來產出客製化的 ListView：

/*---- myService.java ----*/

```
package com.misoo.ppvv;
import java.util.Timer;
import java.util.TimerTask;
import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;

public class myService extends Service {
	private static IListener plis;
	private static int k = 0;
	private Timer timer = new Timer();

	 public Handler handler = new Handler(){
		public void handleMessage(Message msg) {
			plis.update(String.valueOf(k++));
		  }
	 };
	@Override	public void onCreate() {
		super.onCreate();
		TimerTask task = new TimerTask() {
		@Override public void run() {
				handler.sendEmptyMessage(0);
			}
		};
		timer.schedule(task,1000, 4000);
	}
	@Override	public IBinder onBind(Intent intent)
			{	return null;	}
	public static void setUpdateListener(IListener listener)
			{	plis = listener;	}
}
```

Step-4:　撰寫 Activity 的子類別：ac01，其程式碼如下：

/*---- ac01.java ----*/

```
package com.misoo.ppvv;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
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 TextView tx;
	private Button btn;
	@Override public void onCreate(Bundle savedInstanceState) {
	    super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.main);
            LinearLayout layout = new LinearLayout(this);
            layout.setOrientation(LinearLayout.VERTICAL);
            LinearLayout.LayoutParams param =
                        new LinearLayout.LayoutParams(100, WC);
            param.topMargin = 5;
            tx = new TextView(this);         tx.setTextSize(26);
            tx.setTextColor(Color.RED);
            tx.setBackgroundResource(R.drawable.x_yellow);
            layout.addView(tx, param);
            btn = new Button(this);              btn.setText("Exit");
            btn.setBackgroundResource(R.drawable.earth);
            btn.setOnClickListener(this);
            layout.addView(btn, param);
            this.setContentView(layout);

            //---------------------------------------------------------------
            myService.setUpdateListener(new UpdateUIListener());
            Intent svc = new Intent(this, myService.class);
            startService(svc);
    }
    @Override protected void onDestroy() {
            super.onDestroy();
            Intent svc = new Intent(this, myService.class);
            stopService(svc);
    }
  class UpdateUIListener implements IListener {
          public void update(String s)    tx.setText("      " + s);
       }
  public void onClick(View v) {    this.finish();    }
}
```

說明

上述 ac01 的指令：

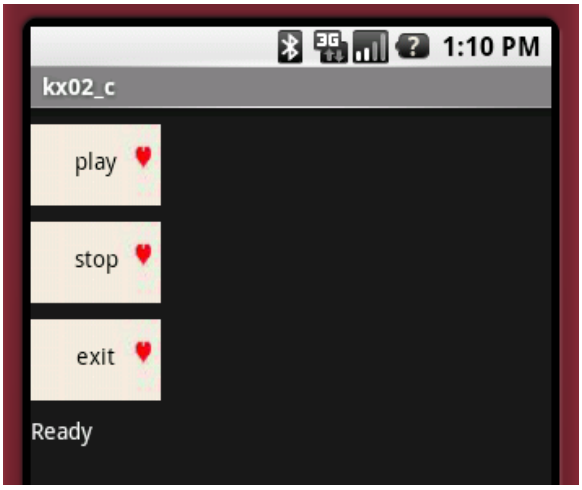
```
myService.setUpdateListener(new UpdateUIListener());
Intent svc = new Intent(this, myService.class);
startService(svc);
```

誕生一個 UpdateUIListner 的對象，然後將此對象之參考值(Reference)傳遞給 myService，存於 plis 參考變數裡。接著，誕生一個 Intent 對象去要求 Android 啓動 myService。就執行到 myService 的 onCreate()函數。此 onCreate()函數誕生一個 Timer 對象，它會定時呼叫 Handler 類別的 handleMessage()函數。此 onCreate()函數就透過 plis 而將資料傳遞給 ac01 裡的 UpdateUIListner 對象，然後將數字呈現於畫面上，並且循環下去。

## 範例(二)：Activity 與 Service 之遠程(IPC)溝通

操作情境

1. 此程式開始執行後，出現畫面如下：




2. 按下<play>，就開始播放 MP3 音樂。
3. 若按下<stop>，就結束播放音樂。
4. 若按下<exit>，程式就結束了。

## 撰寫步驟

Step-1: 建立 Android 專案：kx02_c。

Step-2: 撰寫 Activity 的子類別：ac01，其程式碼如下：

/*---- ac01.java ----*/

```
package com.misoo.kx02c;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.graphics.Color;
import android.os.Bundle;
import android.os.IBinder;
import android.os.Parcel;
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;
	private Button btn, btn2, btn3;
	public TextView tv;
	private IBinder ib = null;

	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.setText("play");          btn.setBackgroundResource(R.drawable.heart);
        btn.setOnClickListener(this);
        LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(80, 50);
        param.topMargin = 10;
        layout.addView(btn, param);

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

        btn3 = new Button(this);   btn3.setId(103);
        btn3.setText("exit");         btn3.setBackgroundResource(R.drawable.heart);
        btn3.setOnClickListener(this);
        layout.addView(btn3, param);

        tv = new TextView(this);   tv.setTextColor(Color.WHITE);
        tv.setText("Ready");
        LinearLayout.LayoutParams param2
                = new LinearLayout.LayoutParams(FP, WC);
        param2.topMargin = 10;
        layout.addView(tv, param2);
        setContentView(layout);
        //------------------------------------------------------------------------
        bindService(new Intent("com.misoo.kx02c.REMOTE_SERVICE"),
                    mConnection, Context.BIND_AUTO_CREATE);
}
private ServiceConnection mConnection = new ServiceConnection() {
            public void onServiceConnected(ComponentName className,
                                                   IBinder ibinder) {
             ib = ibinder;
            }
            public void onServiceDisconnected(ComponentName className) {}
          };
public void onClick(View v) {
        switch (v.getId()) {
        case 101:
              Parcel pc = Parcel.obtain();
              Parcel pc_reply = Parcel.obtain();
              pc.writeString("playing");
              try { ib.transact(1, pc, pc_reply, 0);
                    tv.setText(pc_reply.readString());
              } catch (Exception e) {    e.printStackTrace();     }
              break;
        case 102:
              pc = Parcel.obtain();
              pc_reply = Parcel.obtain();
              pc.writeString("stop");
              try { ib.transact(2, pc, pc_reply, 0);
                    tv.setText(pc_reply.readString());
              } catch (Exception e) {    e.printStackTrace();    }
```

```
                    break;
            case 103:  finish();    break;
    }}}
```

Step-3: 撰寫 Service 的子類別：myService，其程式碼如下：

/*---- myService.java ----*/

```
package com.misoo.kx02c;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class myService extends Service {
      private IBinder mBinder = null;
      @Override       public void onCreate() {
            mBinder = new myBinder(getApplicationContext());  }
      @Override       public IBinder onBind(Intent intent) {    return mBinder;   }
}
```

Step-4: 撰寫 Binder 的子類別：myBinder，其程式碼如下：

/*---- myBinder.java ----*/

```
package com.misoo.kx02c;
import android.content.Context;
import android.media.MediaPlayer;
import android.os.Binder;
import android.os.Parcel;
import android.util.Log;
public class myBinder extends Binder{
      private MediaPlayer mPlayer = null;
      private Context ctx;
      public myBinder(Context cx){    ctx= cx;    }
      @Override       public boolean onTransact(int code, Parcel data, Parcel reply, int flags)
                  throws android.os.RemoteException    {
            reply.writeString(data.readString()+ " mp3");
            if(code == 1) this.play();
            else if(code == 2) this.stop();
            return true;    }
      public void play(){
            if(mPlayer != null) return;
            mPlayer = MediaPlayer.create(ctx, R.raw.test_cbr);
            try { mPlayer.start();
            }catch (Exception e) {       Log.e("StartPlay", "error: " + e.getMessage(), e);}
        }
      public void stop(){
            if (mPlayer != null) {
                  mPlayer.stop();      mPlayer.release();    mPlayer = null; }
  }}
```

說明

上述 ac01 的指令：

```
        public void onCreate(Bundle icicle) {
```

```
    ……..
    bindService(new Intent("com.misoo.kx02c.REMOTE_SERVICE"),
                mConnection, Context.BIND_AUTO_CREATE);
    ……..
  }

  private ServiceConnection mConnection = new ServiceConnection() {
      public void onServiceConnected(ComponentName className,
                                     IBinder ibinder) {
        ib = ibinder;
       }
       public void onServiceDisconnected(ComponentName className) {}
  };
```

程式開始執行時，就誕生一個 ServiceConnection 的對象，並將其參考值存入 mConnection 變數裡。再誕生並送出一個 Intent 對象(把 mConnection 當作參數)給 Android，請它篩選出適合的 Service。

在此範例裡，會配對到 myService，於是執行 onBind()函數，取得 mBinder 參考值。Android 將它當作參數，呼叫了 ac01 裡的 onServiceConnected()函數(因爲 Android 掌握了 mConnection 參考值)：

```
public void onServiceConnected(ComponentName className, IBinder ibinder) {
        ib = ibinder;
      }
```

於是，ib 就參考到 myBinder 對象了。按下<Play>按鈕時，就執行 onClick()：

```
public void onClick(View v) {
    switch (v.getId()) {
    case 101:
        Parcel pc = Parcel.obtain();
        Parcel pc_reply = Parcel.obtain();
        pc.writeString("playing");
        try {
            ib.transact(1, pc, pc_reply, 0);
            tv.setText(pc_reply.readString());




        } catch (Exception e) {    e.printStackTrace();    }
      ……..
}
```

先誕生兩個 Parcel 的對象，做爲後續執行到指令：

```
ib.transact(1, pc, pc_reply, 0);
```

時，作爲其參數，其中 pc 用來傳遞資料給 myBinder，而 pc_reply 則給 myBinder 回傳資料之用。於是，執行到 myBinder 的 onTransact()函數：

```
@Override public boolean onTransact(int code, Parcel data, Parcel reply, int flags)
                    throws android.os.RemoteException     {
        reply.writeString(data.readString()+ " mp3");
        if(code == 1) this.play();
        else if(code == 2) this.stop();
        return true;
    }
```

就依據 ac01 傳來的 code 值而決定呼叫 play()或 stop()函數。如果呼叫到 play()：

```
    public void play(){
        if(mPlayer != null) return;
        mPlayer = MediaPlayer.create(ctx, R.raw.test_cbr);
        try {       mPlayer.start();
        } catch (Exception e) {
              Log.e("StartPlay", "error: " + e.getMessage(), e);
    }}
```

就誕生MediaPlayer對象，就呼叫它的start()函數而開始播放mp3歌曲了。同樣地，如果呼叫到stop()函數：

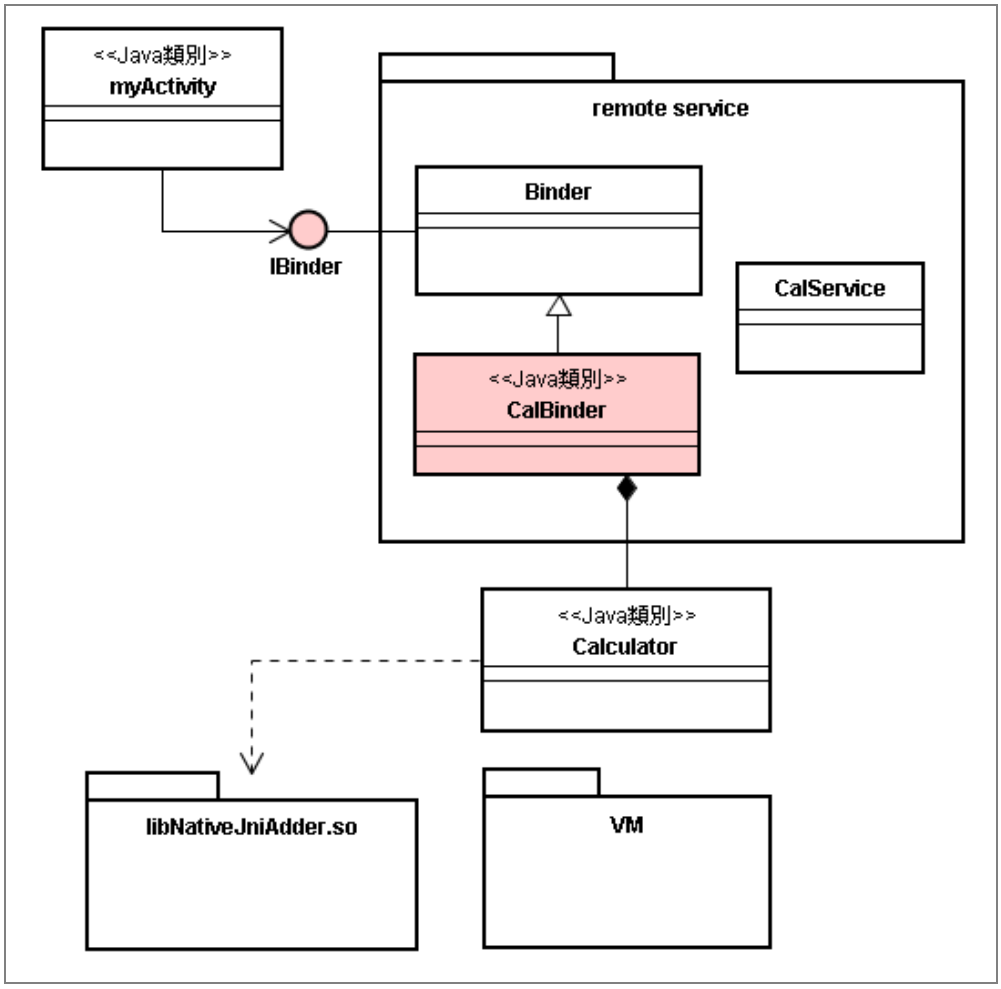
```
    public void stop(){
        if (mPlayer != null) {
                mPlayer.stop();
            mPlayer.release();
            mPlayer = null;
        }}
```

就呼叫MediaPlayer對象的stop()函數而停止播放了。

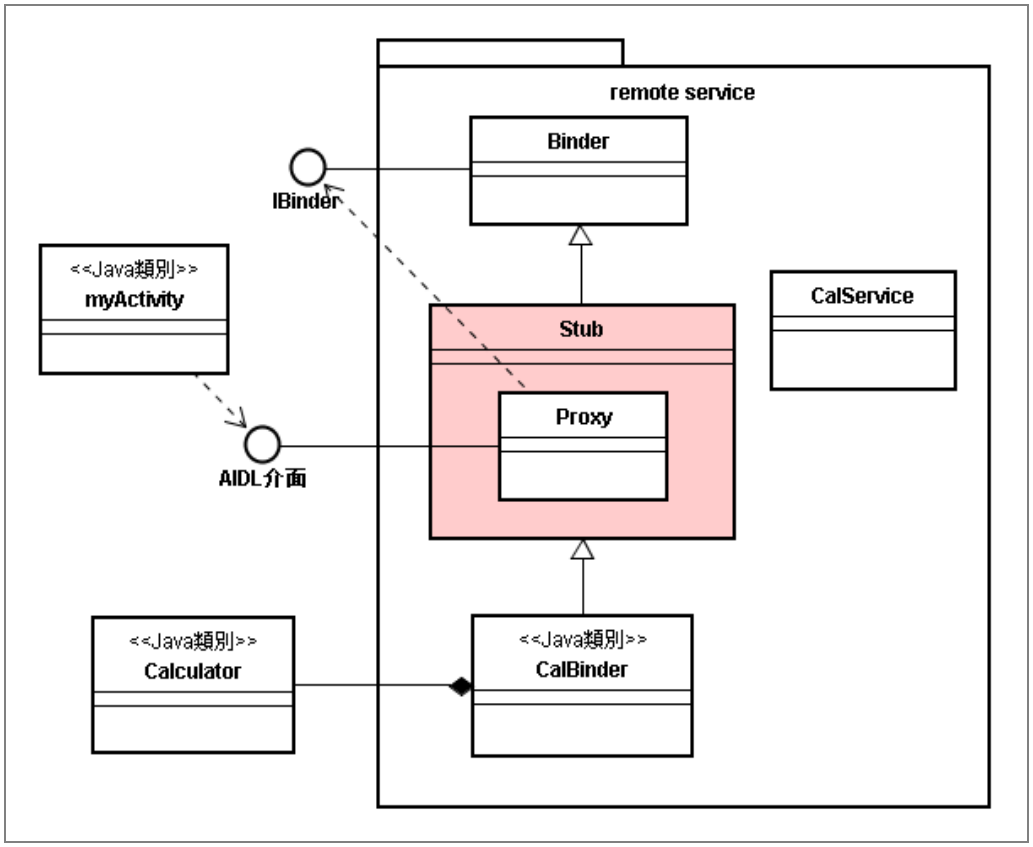
# IBinder 接口與 AIDL

### IBinder 的 Proxy/Stub 機制

AIDL的目的是定義Proxy/Stub來封裝IBinder接口，以便產生更親切貼心的新接口。所以，在應用程式裡，可以選擇使用IBinder接口，也可以使用AIDL來定義出新接口。例如下圖：



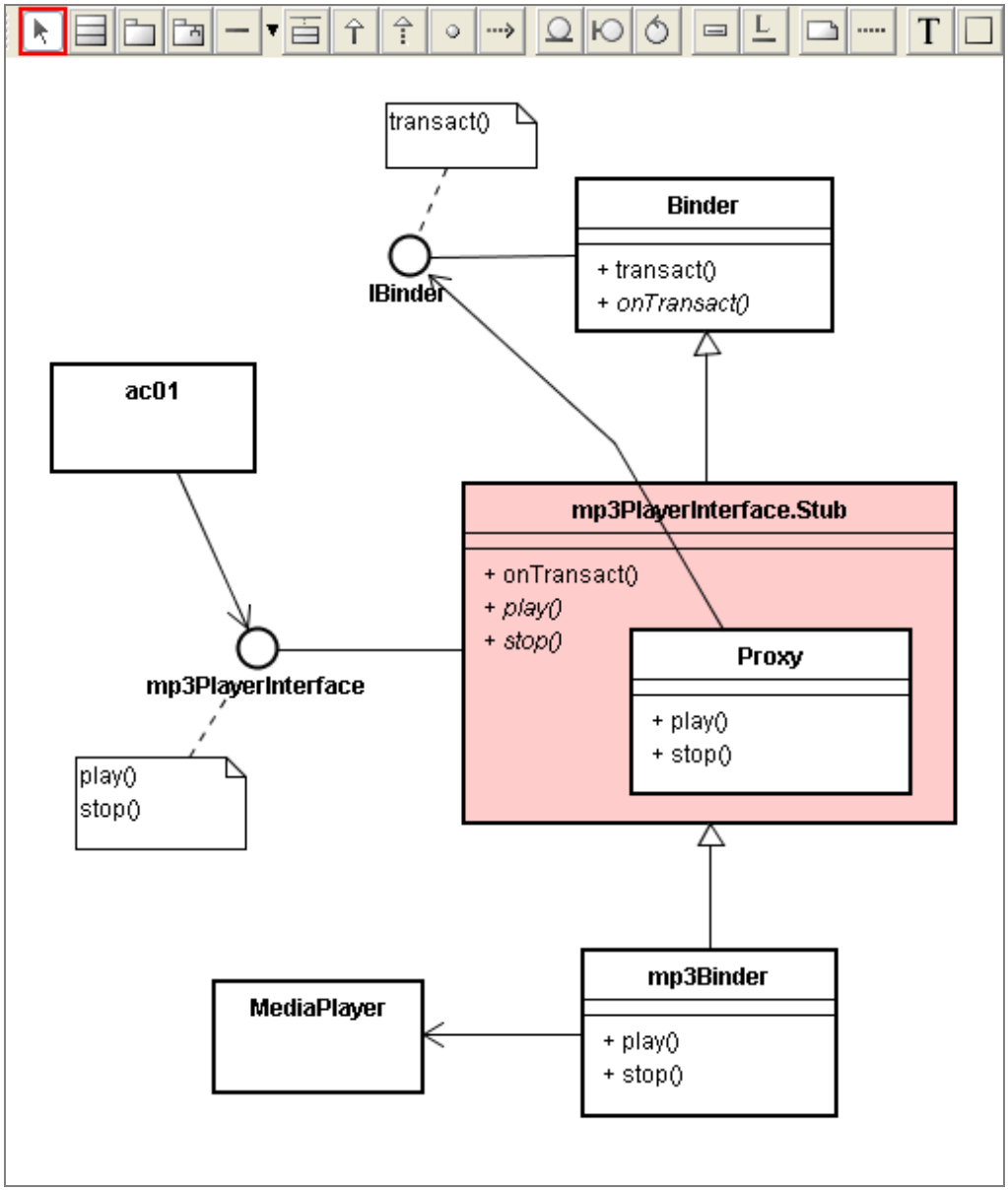

由於IBinder接口只提供單一函數(即transact()函數)來進行遠距溝通，呼叫起來比較不方便。茲設想，如果Calculator類別有多個函數時，myActivity要如何呼叫它們呢？可以呼叫IBinder接口的transact()函數，再轉而呼叫Calculator的各個函數。由於它並不太方便，所以Android提供aidl.exe工具來協助產出Proxy和Stub類別，以化解這個困難，其架構圖如下：




只要你善於使用開發環境的工具(如 Android 的 aidl.exe 軟體工具)自動產生 Proxy 和 Stub 類別的程式碼；那就很方便了。

## Proxy/Stub 機制之範例

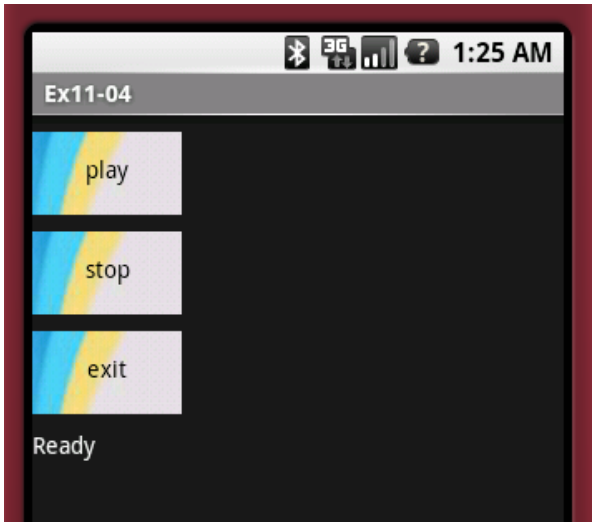
此範例使用 Android-SDK 的/tools/裡的 aidl.exe 工具程式，根據接口定義檔(如下述的 *mp3PlayerInterface.aidl)*而自動產出 Proxy 及 Stub 類別，其結構如下：




藉由開發工具自動產出 Proxy 及 Stub 類別的程式碼，再分別轉交給 ac01 和 mp3Binder 開發者。於是，ac01 和 mp3Binder 兩端的開發者都不必具備 IPC 的知識，因而兩者皆能享受到「不知而亦能用」之美好效益。請看 Android 應用程式碼：

### 操作情境

此程式執行時，出現畫面如下：




## 撰寫程式

Step-1: 建立 Android 應用程式專案：Ex11-04。


Step-2: 定義 mp3PlayerInterface 接口(含 Proxy 和 Stub 類別)。

*// mp3PlayerInterface.aidl*

```
package com.misoo.pkaz;
interface mp3PlayerInterface {
	void play();
	void stop();
}
```

使用 Android-SDK 所含的 aidl.exe 工具，將上述的 mp3PlayerInterface.aidl 檔翻譯成爲下述的 mp3PlayerInterface.java 檔案。

*// mp3PlayerInterface.java*

```
/*
 * This file is auto-generated.    DO NOT MODIFY.
 * Original file: mp3PlayerInterface.aidl
 */
package com.misoo.pkgx;
import java.lang.String;
import android.os.RemoteException;
import android.os.IBinder;
import android.os.IInterface;
```

```
import android.os.Binder;
import android.os.Parcel;
public interface mp3PlayerInterface extends android.os.IInterface
{
/** Local-side IPC implementation stub class. */
public static abstract class Stub extends android.os.Binder implements
com.misoo.pkgx.mp3PlayerInterface
{
private static final java.lang.String DESCRIPTOR =
"com.misoo.pkaz.mp3PlayerInterface";
/** Construct the stub at attach it to the interface. */
public Stub()
{
this.attachInterface(this, DESCRIPTOR);
}
/**
  * Cast an IBinder object into an mp3PlayerInterface interface,
  * generating a proxy if needed.
  */
public static com.misoo.pkgx.mp3PlayerInterface asInterface(android.os.IBinder obj)
{
if ((obj==null)) {
return null;
}
com.misoo.pkgx.mp3PlayerInterface in =
(com.misoo.pkgx.mp3PlayerInterface)obj.queryLocalInterface(DESCRIPTOR);
if ((in!=null)) {
return in;
}
return new com.misoo.pkgx.mp3PlayerInterface.Stub.Proxy(obj);
}
public android.os.IBinder asBinder()
{
return this;
}
public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel reply,
int flags) throws android.os.RemoteException
{
switch (code)
{
case INTERFACE_TRANSACTION:
{
reply.writeString(DESCRIPTOR);
return true;
}
case TRANSACTION_play:
{
data.enforceInterface(DESCRIPTOR);
this.play();
reply.writeNoException();
```

```
return true;
}
case TRANSACTION_stop:
{
data.enforceInterface(DESCRIPTOR);
this.stop();
reply.writeNoException();
return true;
}
}
return super.onTransact(code, data, reply, flags);
}
private static class Proxy implements com.misoo.pkgx.mp3PlayerInterface
{
private android.os.IBinder mRemote;
Proxy(android.os.IBinder remote)
{
mRemote = remote;
}
public android.os.IBinder asBinder()
{
return mRemote;
}
public java.lang.String getInterfaceDescriptor()
{
return DESCRIPTOR;
}
public void play() throws android.os.RemoteException
{
android.os.Parcel _data = android.os.Parcel.obtain();
android.os.Parcel _reply = android.os.Parcel.obtain();
try {
_data.writeInterfaceToken(DESCRIPTOR);
mRemote.transact(Stub.TRANSACTION_play, _data, _reply, 0);
_reply.readException();
}
finally {
_reply.recycle();
_data.recycle();
}
}
public void stop() throws android.os.RemoteException
{
android.os.Parcel _data = android.os.Parcel.obtain();
android.os.Parcel _reply = android.os.Parcel.obtain();
try {
_data.writeInterfaceToken(DESCRIPTOR);
mRemote.transact(Stub.TRANSACTION_stop, _data, _reply, 0);
_reply.readException();
}
```

```
finally {
_reply.recycle();
_data.recycle();
}
}
}
static final int TRANSACTION_play = (IBinder.FIRST_CALL_TRANSACTION + 0);
static final int TRANSACTION_stop = (IBinder.FIRST_CALL_TRANSACTION + 1);
}
public void play() throws android.os.RemoteException;
public void stop() throws android.os.RemoteException;
}
```

從表面上看來，此 mp3PlayerInterface.java 是蠻複雜的，其實它的結構是清晰又簡單的，只要對於類別繼承、反向呼叫和接口等對象導向觀念必須有足夠的認識，就很容易理解了。如果你想細探它的結構，請您閱讀筆者的第 3 本 Android 書籍：*Android 與對象導向技術* 一書的第 16 章。

Step-3: 撰寫 mp3Service 類別。

*// mp3Service.java*

```
package com.misoo.pkgx;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class mp3Service extends Service {
	IBinder ib = null;
	@Override	public void onCreate() {
		super.onCreate();
		ib = new mp3Binder(this.getApplicationContext());
	}
	@Override	public void onDestroy() { }
	@Override	public IBinder onBind(Intent intent) {	return ib;	}
}
```

Step-4: 撰寫 mp3Binder 類別。

*// mp3Binder.java*

```
package com.misoo.pkgx;
import android.content.Context;
import android.media.MediaPlayer;
import android.util.Log;

public class mp3Binder extends mp3PlayerInterface.Stub{
	private MediaPlayer mPlayer = null;
	private Context ctx;
	public mp3Binder(Context cx){ ctx= cx;	}
	public void play(){
```

```
            if(mPlayer != null) return;
            mPlayer = MediaPlayer.create(ctx, R.raw.test_cbr);
            try { mPlayer.start();
            } catch (Exception e) {       Log.e("StartPlay", "error: " + e.getMessage(), e);    }
      }
      public void stop(){
           if (mPlayer != null) { mPlayer.stop();    mPlayer.release();    mPlayer = null; }
      }
}
```

Step-5: 撰寫 myButton 類別。

*// myButton.java*

```
package com.misoo.pkgx;
import android.content.Context;
import android.widget.Button;

public class myButton extends Button{
      public myButton(Context ctx){
            super(ctx);
            super.setBackgroundResource(R.drawable.byw);
      }
      public int get_width(){      return 90;    }
      public int get_height(){    return 50;    }
}
```

Step-6: 撰寫 ac01 類別。

*// ac01.java*

```
package com.misoo.pkgx;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.graphics.Color;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
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;
      private mp3PlayerInterface mp3_interface = null;
       private myButton btn, btn2, btn3;
       public TextView tv;
      public void onCreate(Bundle icicle) {
            super.onCreate(icicle);
            LinearLayout layout = new LinearLayout(this);
```

```
            layout.setOrientation(LinearLayout.VERTICAL);
            btn = new myButton(this);      btn.setId(101);
            btn.setText("play");               btn.setOnClickListener(this);
            LinearLayout.LayoutParams param =
                    new LinearLayout.LayoutParams(btn.get_width(), btn.get_height());
            param.topMargin = 10;
            layout.addView(btn, param);
            btn2 = new myButton(this);      btn2.setId(102);
            btn2.setText("stop");              btn2.setOnClickListener(this);
            layout.addView(btn2, param);

            btn3 = new myButton(this);      btn3.setId(103);
            btn3.setText("exit");              btn3.setOnClickListener(this);
            layout.addView(btn3, param);
            tv = new TextView(this);      tv.setTextColor(Color.WHITE);
            tv.setText("Ready");
            LinearLayout.LayoutParams param2 =
                    new LinearLayout.LayoutParams(FP, WC);
            param2.topMargin = 10;        layout.addView(tv, param2);
            setContentView(layout);
            //-----------------------------------------------------------------------
            startService(new Intent("com.misoo.pkgx.REMOTE_SERVICE"));
            bindService(new Intent("com.misoo.pkgx.REMOTE_SERVICE"),
                            mConnection, Context.BIND_AUTO_CREATE);
          }
          private ServiceConnection mConnection = new ServiceConnection() {
                public void onServiceConnected(ComponentName className, IBinder
ibinder)
                  {    mp3_interface = mp3PlayerInterface.Stub.asInterface(ibinder);
                          //setTitle("service connected.");
                  }
                public void onServiceDisconnected(ComponentName className) {}
          };
         public void onClick(View v) {
           switch(v.getId()){
           case 101:
                  tv.setText("play...");
                  try { mp3_interface.play();
                  } catch (RemoteException e) {    e.printStackTrace();    }
                 break;
           case 102:
                  tv.setText("stop.");
                  try { mp3_interface.stop();
                  } catch (RemoteException e) {    e.printStackTrace();    }
                  break;
           case 103:
                  unbindService(mConnection);
                  stopService(new Intent("com.misoo.pkgx.REMOTE_SERVICE"));
                 finish();    break;
    }}}
```

說明

對於 Anrdoid 的初學者而言， Android 的 AIDL 機制可說是最難弄懂的。

有關於 Android-SDK 所含的 aidl.exe 工具的應用，以及 AIDL 接口之詳細說明，請您閱讀筆者的：

- 第 2 本 Android 書籍：*Android 應用軟體架構設計* 一書的第 6 章。
- 第 3 本 Android 書籍：*Android 與對象導向技術* 一書的第 16 章。

當你能理解Android的 aidl.exe工具之角色，其自動產生Proxy及Stub類別，來包容IPC複雜細節，且可任意抽換IPC機制而不會影響到應用類別，其促進抽換性和生生不息的活力，豈不美哉!

~~ END ~~