JNI & NDK 雕龍小技

內容：

1. JNI 複習

茲複習一下 JNI 接口的角色。在 Android 環境裡，Java 層與 C 層之間會有密切的溝通。此時 JNI(Java Native Interface)就扮演雙方溝通的接口了。藉由 JNI 接口，可將 Java 層的類的函數之實作部份挖空，而移到本地的 C 函數來實作之。例如，原來在 Java 層有個完整的 Java 類如下：

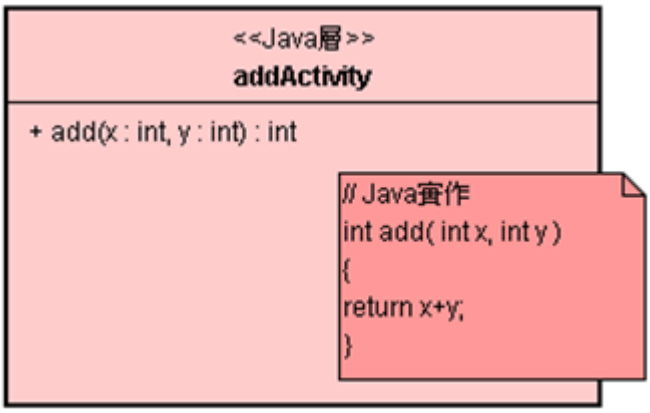


圖 1　一般的 Java 類

這是一個完整的 Java 類，其 add()函數裡有完整的實作(Implementation)程序碼。如果從這 Java 類裡移除掉 add()函數的實作程序碼(就像抽象類裡的抽象函數一般)，就成爲本地(Native)函數了；然後依循 JNI 接口協定而以 C 語言來實作之，如下：

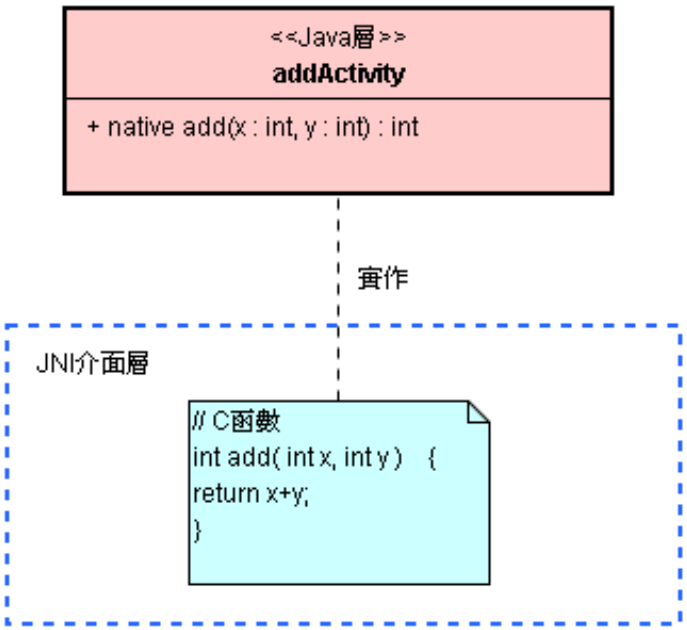


圖 2　以 C 語言來實作 Java 類的函數

這個 add()函數仍然是 Java 類的一部分，只是它是用 C 語言來實作而已。爲什麼要將 Java 類的 add()函數挖空呢? 本地程序碼與平台設備息息相關，意味著它與平台設備的相依性很高，也就是它不具備跨平台的特性。所以，在設計整個應用軟件時，通常會從執行速度、跨平台能力等不同角度來做評估，以便取得最合乎需要的規劃。在本地的 C 程序碼裡，可以創建 C++類的對象，並調用其函數如下圖：

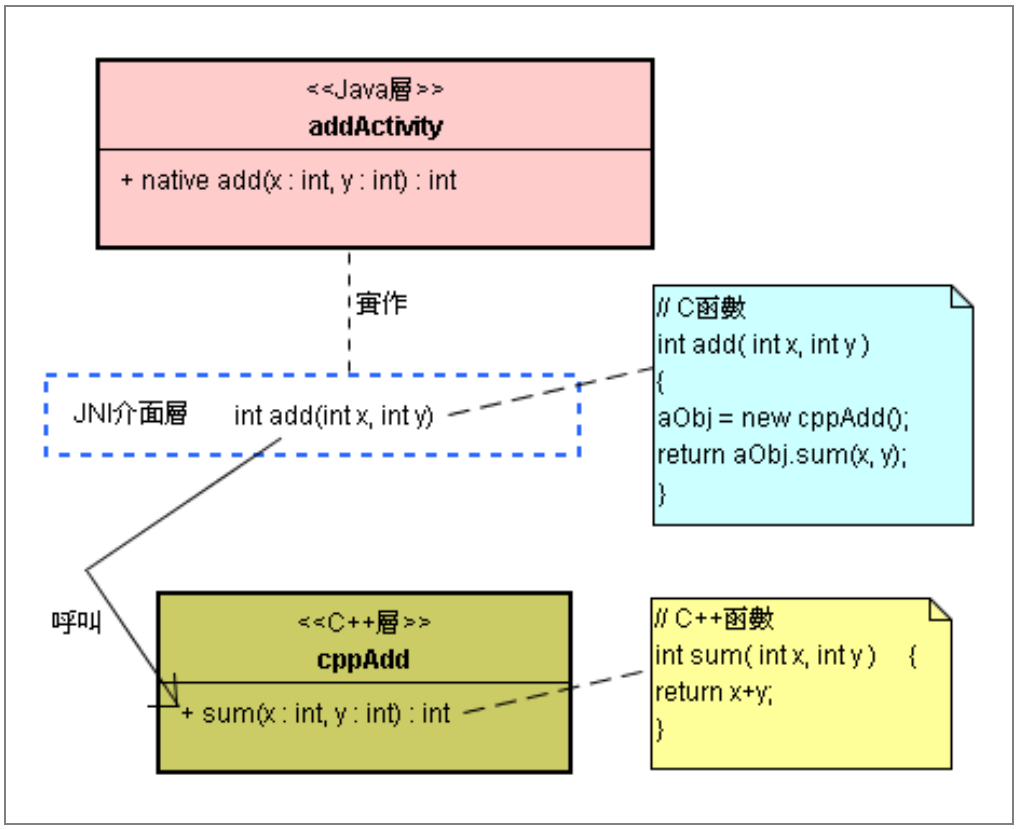


圖 3　Java 類透過本地函數來調用 C++函數

藉由 JNI 接口就能讓 Java 類與 C++類互相溝通起來了。這是 Android 框架的重要機制。

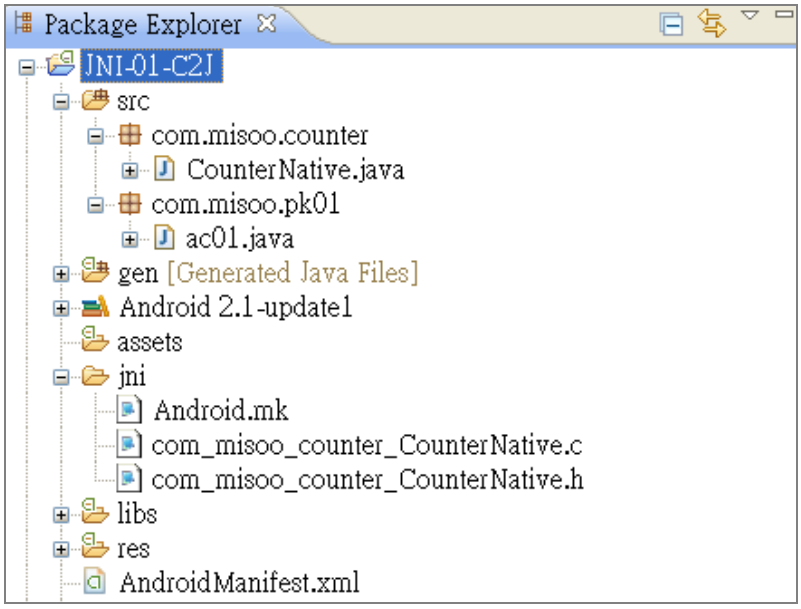
從上圖看來，只看到上層的 Java 函數調用中間 JNI 層的 C 函數，再往下調用 C++層的函數。然而，在 Android 環境裡，從 C/C++層函數反過來調用 Java 層函數，反而是更關鍵性的機制。

2. 從本地 C 函數調用 Java 函數

2.1 範例

茲舉例說明如何活用 JNI 和 NDK 來讓本地 C 函數順利調用 Java 函數。

- 建立 Android 開發專案：JNI-01-C2J，如下：



其中，CounterNative 類裡含有 3 個本地函數，但其程序碼被挖空了，然後實作於 com_misoo_counter_CounterNative.c 程序檔裡。透過 Android NDK 工具，就能將 com_misoo_counter_CounterNative.c 程序加以編譯、連結成爲*.so 程序庫，放置於/libs/目錄裡。

- 設計架構圖，如下：

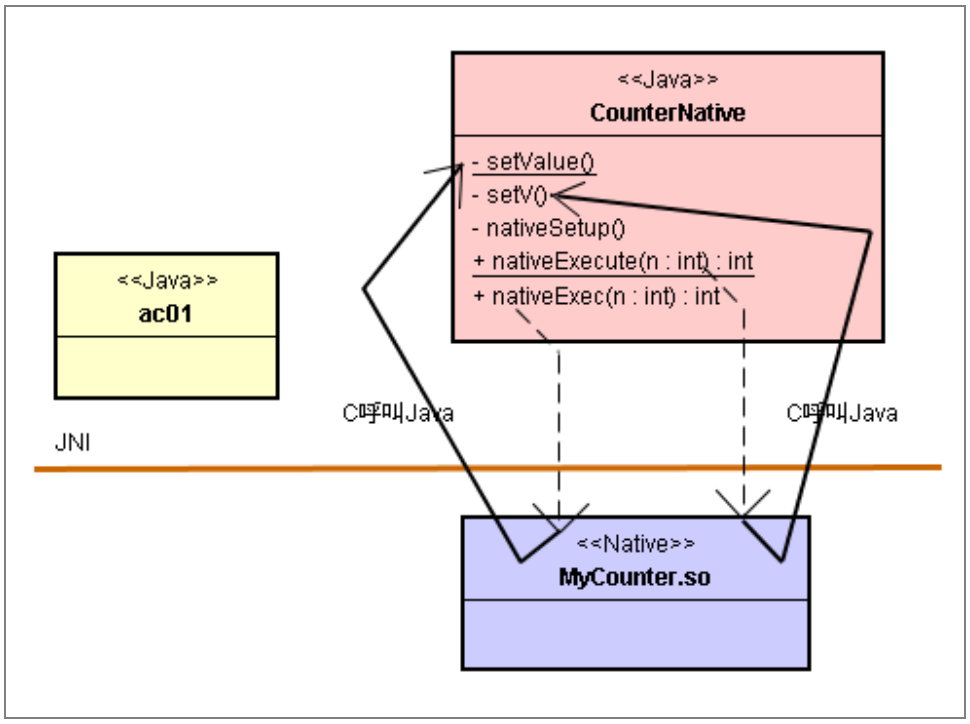


圖 4　C 層本地函數調用 Java 函數

在 CounterNative 類裡，有 3 個本地(Native)函數：靜態(static)的 nativeExecute()和一般的 nativeSetup()及 nativeExec()。這些函數會實作於 C 模塊(即 MyCounter.so)裡。其中，靜態 nativeExecute()會調用 Java 層的一般的 setV()函數；而一般的 nativeExec()會調用 Java 層的靜態 setValue()函數。

- 撰寫 CounterNative.java 類，如下：

```
// CounterNative.java
package com.misoo.counter;
import com.misoo.pk01.ac01;
import android.os.Handler;
import android.os.Message;

public class CounterNative {
        private static Handler h;
          static {
                System.loadLibrary("MyCounter");
```

```
            }
        public CounterNative(){
            h = new Handler(){
            public void handleMessage(Message msg) {
                    ac01.ref.setTitle(msg.obj.toString());
                }};
            nativeSetup();
        }
        private static void setValue(int value){
            String str = "Value(static) = " + String.valueOf(value);
             Message m = h.obtainMessage(1, 1, 1, str);
             h.sendMessage(m);
         }
        private void setV(int value){
            String str = "Value = " + String.valueOf(value);
             Message m = h.obtainMessage(1, 1, 1, str);
             h.sendMessage(m);
          }
        private native void nativeSetup();
        public native static void nativeExecute(int n);
        public native void nativeExec(int n);
    }
```

在這範例裡，將由ac01類調用CounterNative類的建構函數，此函數創建了一個Handler對象，並且調用本地的nativeSetup()函數。

隨後，ac01將調用靜態的nativeExecute()函數，此函數則反過來調用Java層一般的setV()函數。接著，ac01將調用一般的nativeExec()函數，此函數則反過來調用Java層的靜態setValue()函數。

- 編譯 JNI-01-C2J 專案，產出*.class 檔案。
- 使用 javah 工具從*.class 產出*.h 標頭檔，如下：

/ *com_misoo_counter_CounterNative.h* */*

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_misoo_counter_CounterNative */

#ifndef _Included_com_misoo_counter_CounterNative
#define _Included_com_misoo_counter_CounterNative
```

```
#ifdef __cplusplus
extern "C" {
#endif
/*
  * Class:         com_misoo_counter_CounterNative
  * Method:        nativeSetup
  * Signature: ()V
  */
JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeSetup
    (JNIEnv *, jobject);
/*
  * Class:         com_misoo_counter_CounterNative
  * Method:        nativeExecute
  * Signature: (I)V
  */
JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeExecute
    (JNIEnv *, jclass, jint);
/*
  * Class:         com_misoo_counter_CounterNative
  * Method:        nativeExec
  * Signature: (I)V
  */
JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeExec
    (JNIEnv *, jobject, jint);
#ifdef __cplusplus
}
#endif
#endif
```

- 撰寫*.c 實作檔案，如下：

/ *com.misoo.counter.CounterNative.c* */*

```
#include "com_misoo_counter_CounterNative.h"
jclass       m_class;
jobject      m_object;
jmethodID m_mid_static, m_mid;

JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeSetup
    (JNIEnv *env, jobject thiz)
{
     jclass clazz = (*env)->GetObjectClass(env, thiz);
     m_class = (jclass)(*env)->NewGlobalRef(env, clazz);
     m_object = (jobject)(*env)->NewGlobalRef(env, thiz);
     m_mid_static = (*env)->GetStaticMethodID(env, m_class, "setValue", "(I)V");
```

```
    m_mid = (*env)->GetMethodID(env, m_class, "setV", "(I)V");
    return;
 }
JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeExecute
   (JNIEnv *env, jclass clazz, jint n)
{
      int i, sum = 0;
      for(i=0; i<=n; i++)     sum+=i;
      (*env)->CallVoidMethod(env, m_object, m_mid, sum);
      return;
 }
JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeExec
   (JNIEnv *env, jobject thiz, jint n)
{
      int i, sum = 0;
      for(i=0; i<=n; i++)    sum+=i;
      (*env)->CallStaticVoidMethod(env, m_class, m_mid_static, sum);
      return;
 }
```

- 使用 NDK 環境，編譯、連結而產出*.so 程序庫(Library)。
- 將*.so 程序庫拷貝到 JNI-01-C2J 專案裡。
- 編修 ac01.java 類，如下：

// ac01.java

```
package com.misoo.pk01;
import com.misoo.counter.CounterNative;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;

public class ac01 extends Activity implements OnClickListener {
   static public ac01 ref;
   private Button btn, btn2, btn3;
   private CounterNative cn;

   @Override public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      ref = this;
```

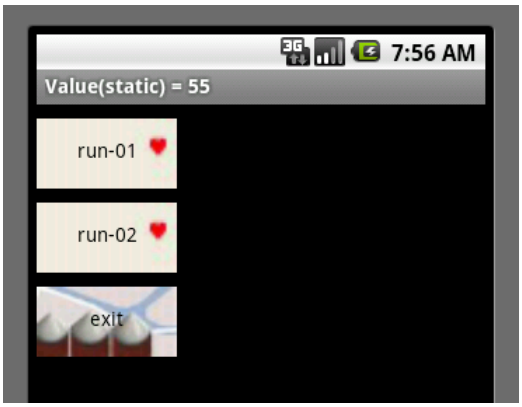
```
    LinearLayout layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    btn = new Button(this);       btn.setId(101);
    btn.setBackgroundResource(R.drawable.heart);
    btn.setText("run-01");        btn.setOnClickListener(this);
    LinearLayout.LayoutParams param =
          new LinearLayout.LayoutParams(100,50);
    param.topMargin = 10;       layout.addView(btn, param);

    btn2 = new Button(this);     btn2.setId(102);
    btn2.setBackgroundResource(R.drawable.heart);
    btn2.setText("run-02");       btn2.setOnClickListener(this);
    layout.addView(btn2, param);
    btn3 = new Button(this);     btn3.setId(103);
    btn3.setBackgroundResource(R.drawable.gray);
    btn3.setText("exit");          btn3.setOnClickListener(this);
    layout.addView(btn3, param);
    setContentView(layout);
    //-----------------------------------------------------------------
    cn = new CounterNative();
  }
@Override public void onClick(View v) {
     switch(v.getId()){
     case 101:   cn.nativeExec(10);
                  break;
     case 102:   CounterNative.nativeExecute(11);
                  break;
     case 103:   finish();
                  break;
  }}}
```

2.2 範例執行

當你從畫面裡按下<run-01>按鈕，就執行指令：

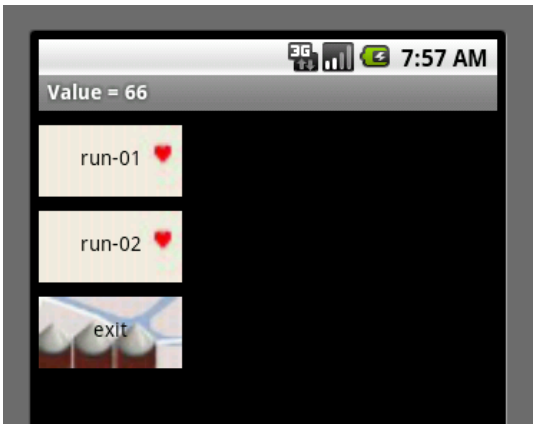
```
cn.nativeExec(10);
```

就調用C函數：com_misoo_counter_CounterNative_nativeExec()，計算出sum值之後，透過VM的CallVoidMethod()函數而調用到目前Java對象的setValue()函數，把sum值傳入Java層，並顯示出來，如下：



當你從畫面裡按下<run-02>按鈕，就執行指令：cn.nativeExecute(11);

此時轉而調用 C 函數：com_misoo_counter_CounterNative_nativeExecute()，計算出 sum 值之後，透過 VM 的 CallVoidMethod()函數而調用到目前 Java 對象的 setV()函數，把 sum 值傳入 Java 層，並顯示出來，如下：



以上說明了 Java 與 C 函數之間的相互調用機制。接下去，將展現出更多的美妙用法和技巧。

3. 從本地 C 函數創建 Java 對象

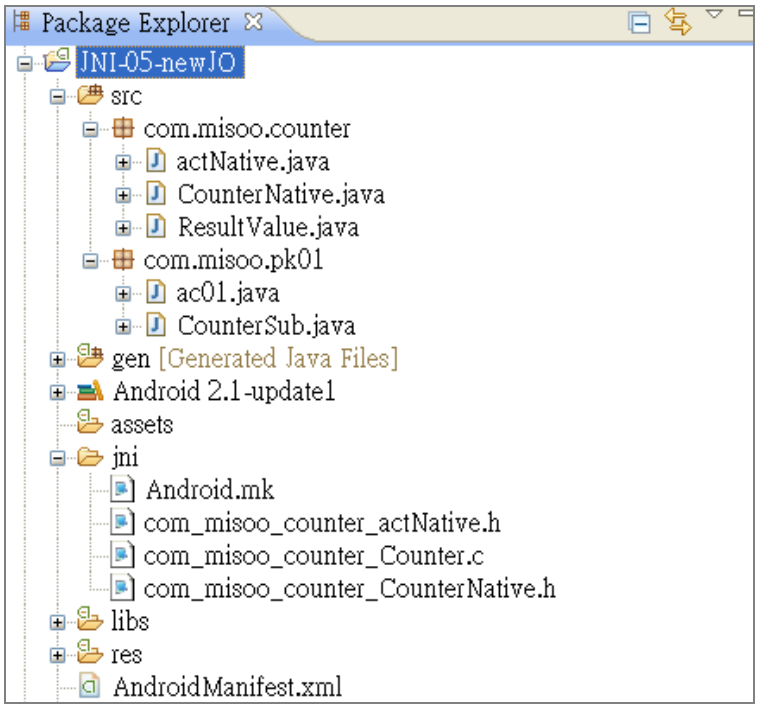
3.1 前言

在前面的範例裡，都是先創建 Java 層對象，然後將該對象的參考(Reference)傳遞給 C 模塊。例如，在上一個範例裡，由 CounterNative 去創建 ResultValue 類的對象。然後將 ResultValue 對象參考傳遞給 C 模塊。本節的範例將改由 C 模塊來創建 Java 層的 ResultValue 對象。

3.2 範例

茲舉例說明如何由 C 模塊來創建 Java 層的對象。

- 建立 Android 開發專案：JNI-05-newJO，如下：



改由 C 模塊來創建 Java 層的 ResultValue 對象，其意味著 C 模塊擁有較大的掌控權。也就是說，整個應用程序的控制中心點，從 Java 層轉移到本地的 C 模塊。如果你決定由 C 模塊來主導系統的執行，這項技巧是非常重要的。

本範例，除了不再由 CounterNative 類負責創建 ResultValue 對象之外，其他部份與上一個範例都相同。

- 設計架構圖，如下：

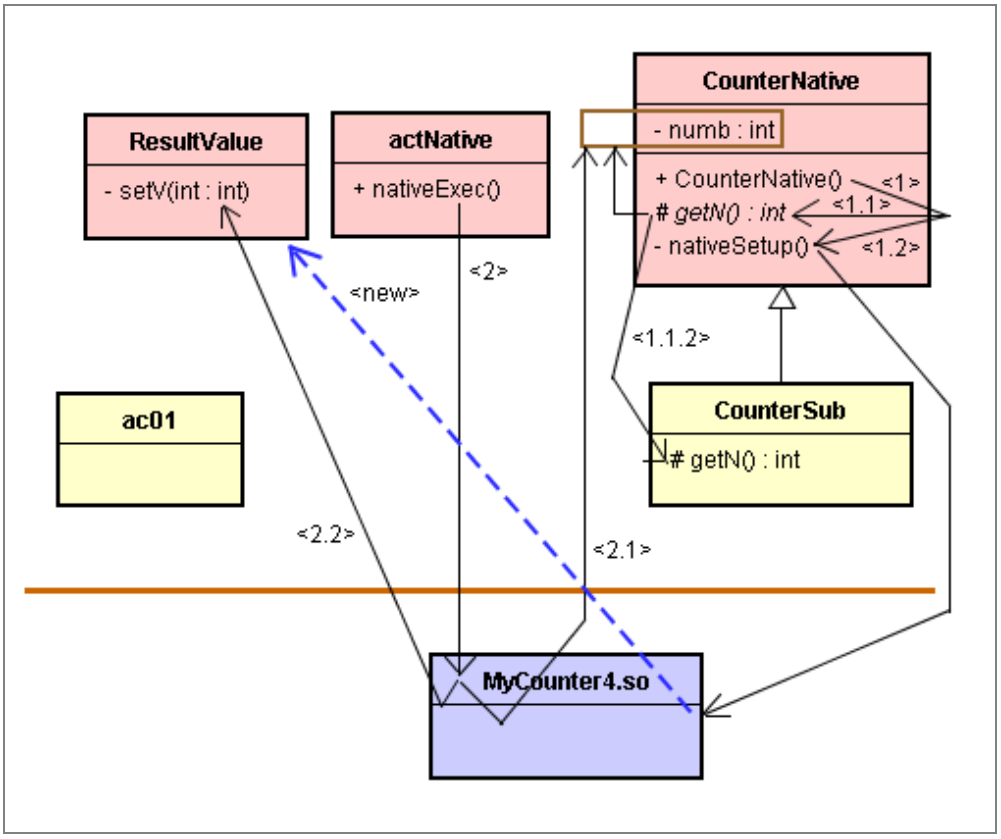


圖 5　C 模塊創建 Java 的對象

此範例執行到 C 模塊的 nativeSetup()函數時，會透過 VM 而調用 ResultValue 類的建構函數去創建 ResultValue 對象。

- 撰寫 Java 層的 actNative、CounterNative 和 ResultValue 類，如下：

// actNative.java

```
package com.misoo.counter;

public class actNative {
        public static native Object nativeExec();
}
```

上述 actNative 類裡定義了 1 個本地函數。

// CounterNative.java

```
package com.misoo.counter;

abstract public class CounterNative {
          private int numb;
          static {    System.loadLibrary("MyCounter5");      }
          public CounterNative(){
                        numb = getN();    nativeSetup();
           }
          abstract protected int getN();
          private native void nativeSetup();
     }
```

這 CounterNative 類裡定義了 1 個抽象函數，以及 1 個本地函數。抽象函數是由 CounterSub 子類來實作；而本地函數則由 C 模塊來實作。

// ResultValue.java

```
package com.misoo.counter;

public class ResultValue {
     private int mValue;

     public int getValue(){      return mValue;      }
     private void setV(int value){      mValue = value;      }
}
```

此類裡的 setV()函數讓 C 模塊來調用，以便送來計算結果；而 getValue()函數則讓 ac01 類來調用，以便將計算結果顯示於畫面上。

- 編譯 JNI-05-newJO 專案，產出*.class 檔案。
- 使用 javah 工具從*.class 產出*.*h* 標頭檔。
- 撰寫*.c 實作檔案，如下：

/ com.misoo.counter.Counter.c */*

```
#include <android/log.h>
#include "com_misoo_counter_actNative.h"
#include "com_misoo_counter_CounterNative.h"
jobject      m_object, m_rv_object ;
jfieldID    m_fid;
jmethodID m_rv_mid;

JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeSetup
        (JNIEnv *env, jobject thiz) {
     jclass clazz = (*env)->GetObjectClass(env, thiz);
     m_object = (jobject)(*env)->NewGlobalRef(env, thiz);
     m_fid = (*env)->GetFieldID(env, clazz, "numb", "I");

     jclass rvClazz = (*env)->FindClass(env, "com/misoo/counter/ResultValue");
     jmethodID constr = (*env)->GetMethodID(env, rvClazz, "<init>", "()V");
     jobject ref = (*env)->NewObject(env, rvClazz, constr);
     m_rv_object = (jobject)(*env)->NewGlobalRef(env, ref);
     m_rv_mid = (*env)->GetMethodID(env, rvClazz, "setV", "(I)V");
     return;
}
JNIEXPORT jobject JNICALL Java_com_misoo_counter_actNative_nativeExec
   (JNIEnv *env, jclass clazz) {
      int n, i, sum = 0;
      n = (int)(*env)->GetObjectField(env, m_object, m_fid);
      for(i=0; i<=n; i++)
               sum+=i;
      (*env)->CallVoidMethod(env, m_rv_object, m_rv_mid, sum);
      return m_rv_object;
}
```

- 使用 NDK 環境，編譯、連結而產出*.so 程序庫(Library)。
- 將*.so 程序庫拷貝到 JNI-05-newJO 專案裡。
- 撰寫 CounterSub.java 類，如下：

// CounterSub.java

```
package com.misoo.pk01;
import com.misoo.counter.CounterNative;

public class CounterSub extends CounterNative{
        protected int getN() {    return 16;   }
```

```
}
```

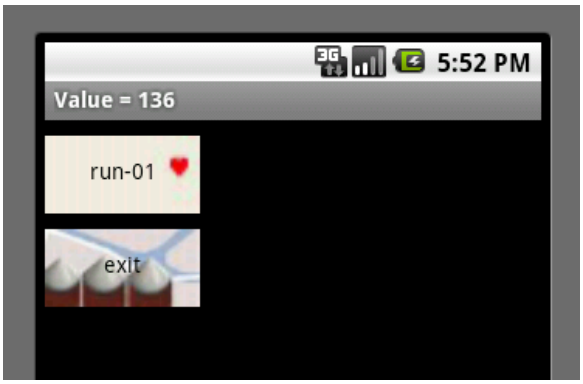
- 編修 ac01.java 類，如下：

// ac01.java

```
package com.misoo.pk01;
import com.misoo.counter.CounterNative;
import com.misoo.counter.ResultValue;
import com.misoo.counter.actNative;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;

public class ac01 extends Activity implements OnClickListener {
   private Button btn, btn3;
   private CounterNative cn;

   @Override public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      LinearLayout layout = new LinearLayout(this);
      layout.setOrientation(LinearLayout.VERTICAL);
      btn = new Button(this);       btn.setId(101);
      btn.setBackgroundResource(R.drawable.heart);
      btn.setText("run-01");        btn.setOnClickListener(this);
      LinearLayout.LayoutParams param =
            new LinearLayout.LayoutParams(100,50);
      param.topMargin = 10;
      layout.addView(btn, param);

      btn3 = new Button(this);       btn3.setId(103);
      btn3.setBackgroundResource(R.drawable.gray);
      btn3.setText("exit");             btn3.setOnClickListener(this);
      layout.addView(btn3, param);
      setContentView(layout);
      //-------------------------------------------
      cn = new CounterSub();
   }
@Override public void onClick(View v) {
       switch(v.getId()){
       case 101:    ResultValue rvObj = (ResultValue)actNative.nativeExec();
                    setTitle("Value = " + rvObj.getValue());      break;
       case 103:     finish();    break;
```

```
}}}
```

當你從畫面裡按下<run-01>按鈕，就執行指令：actNative.nativeExec();
此時，ac01 調用 actNative 類裡的 nativeExec()本地函數，轉而調用到 C 模塊的 nativeExec()函數，其先取得 Java 層 CounterNative 對象裡的 numb 值，計算出 sum 值之後，再調用該新對象的 setV()函數，就把 sum 值傳送到新對象裡。最後，nativeExec()函數將新對象參考傳給 Java 層，讓 ac01 將 sum 值顯示於畫面上，如下的畫面：



以上說明了如何從 C 模塊來創建 Java 層的對象。

4. 深層 C++對象調用 Java 層函數

4.1 前言

在上一個範例裡，由 JNI 層的 C 函數調用 Java 層的函數。相對上，這 C 函數是主動調用者；而 Java 函數是被調用者；其 C 模塊掌握了控制權，成爲整體軟件系統的控制點。本節將延續上一節的議題，更進一步將控制點往更底層移動，也就是由更下層的 C++對象來調用 Java 層的函數。如下圖：

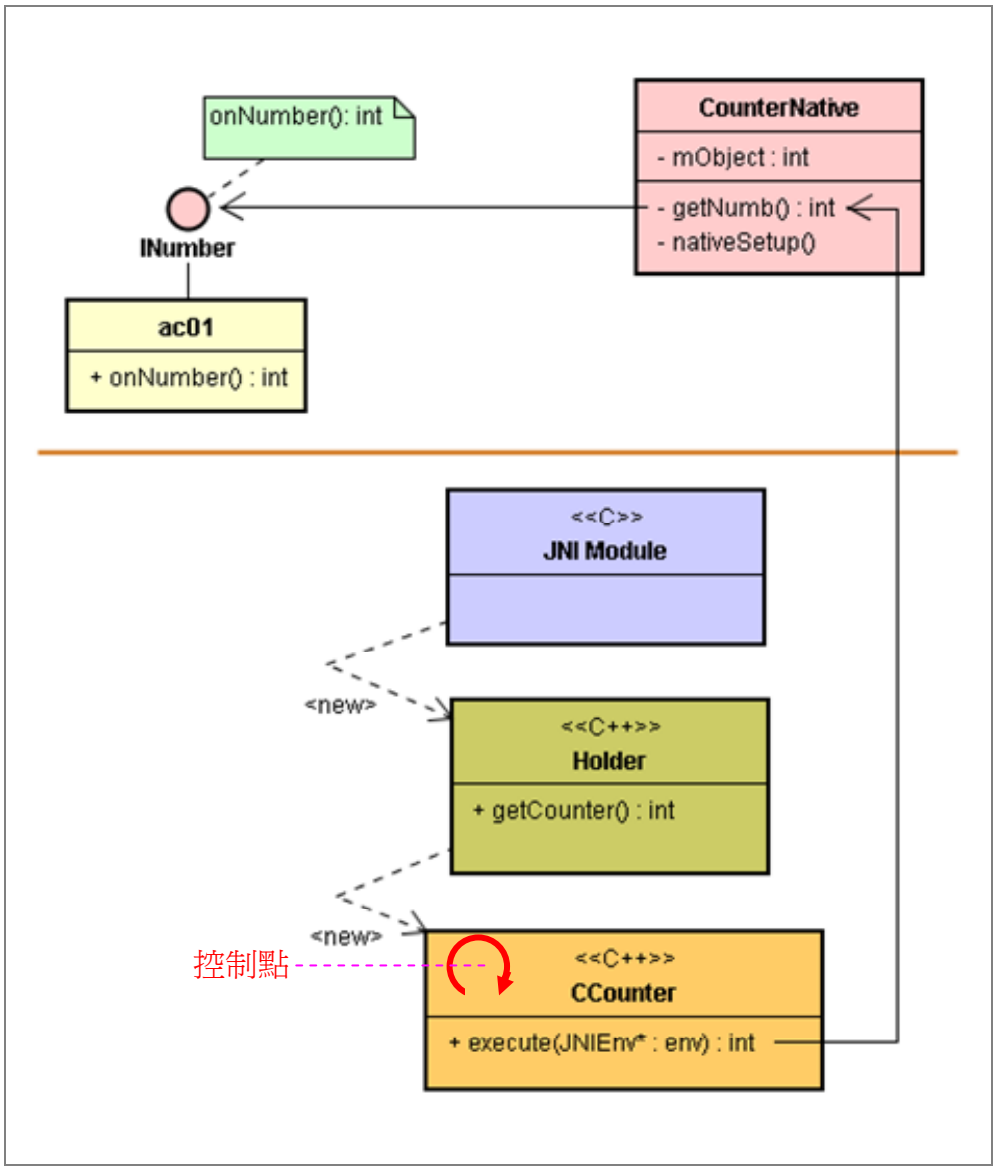


圖 6　C++對象是控制點

當 C++層對象擁有主導權，它想調用 Java 層函數時，它必須取得 JNIEnv*指標，才能調用 VM 的函數，間接地調用到 Java 層的函數。此時，取得 JNIEnv*指標的途徑有二：

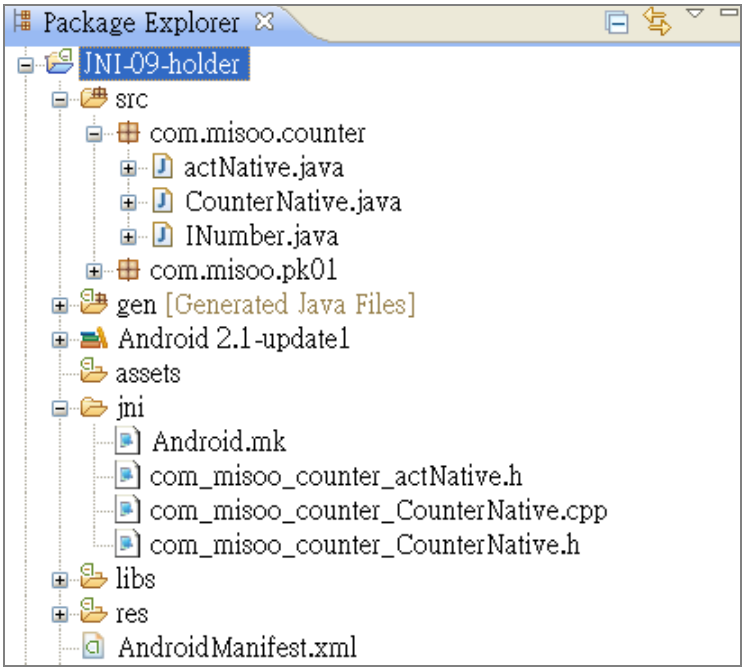
- 從 C 層的本地函數將 JNIEnv*指標傳遞給 C++對象。
- 由 C++對象的執行緒(Thread)向 VM 取得。

由於本節暫時不涉及執行緒議題，所以採取上述的第一種途徑。

4.2 範例

茲舉例說明如何讓深層 C++對象能調用 Java 層的函數。

- 建立 Android 開發專案：JNI-09-holder，如下：



- 設計架構圖，如下：

茲定義 INumber 接口，由 ac01 類來實作之。C 層模塊的 nativeSetup()函數先創建 Holder 對象，並建立它與 Java 層 CounterNative 對象之間的相互連結關係，如下圖：

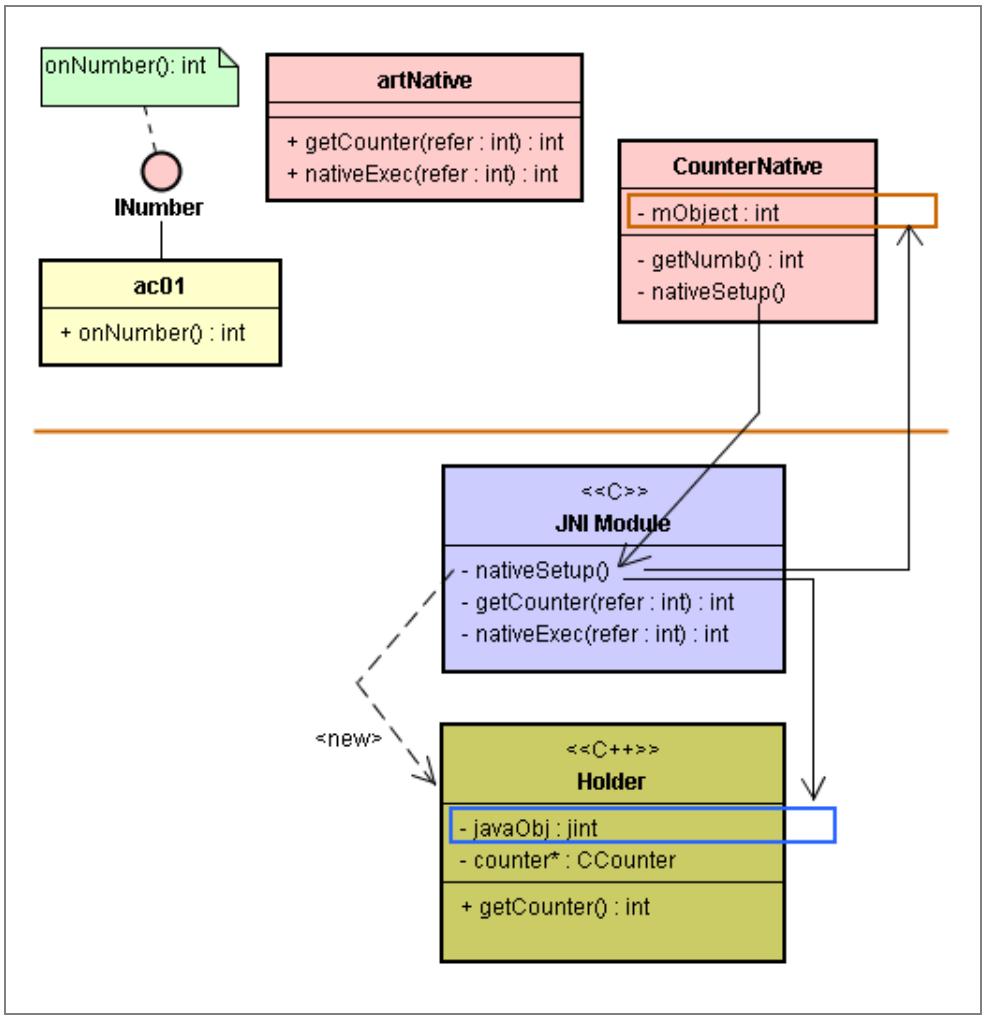


圖 7　先建立雙向連結關係

這個 Holder 對象還可以創建更底層的 CCounter 對象，如下圖：

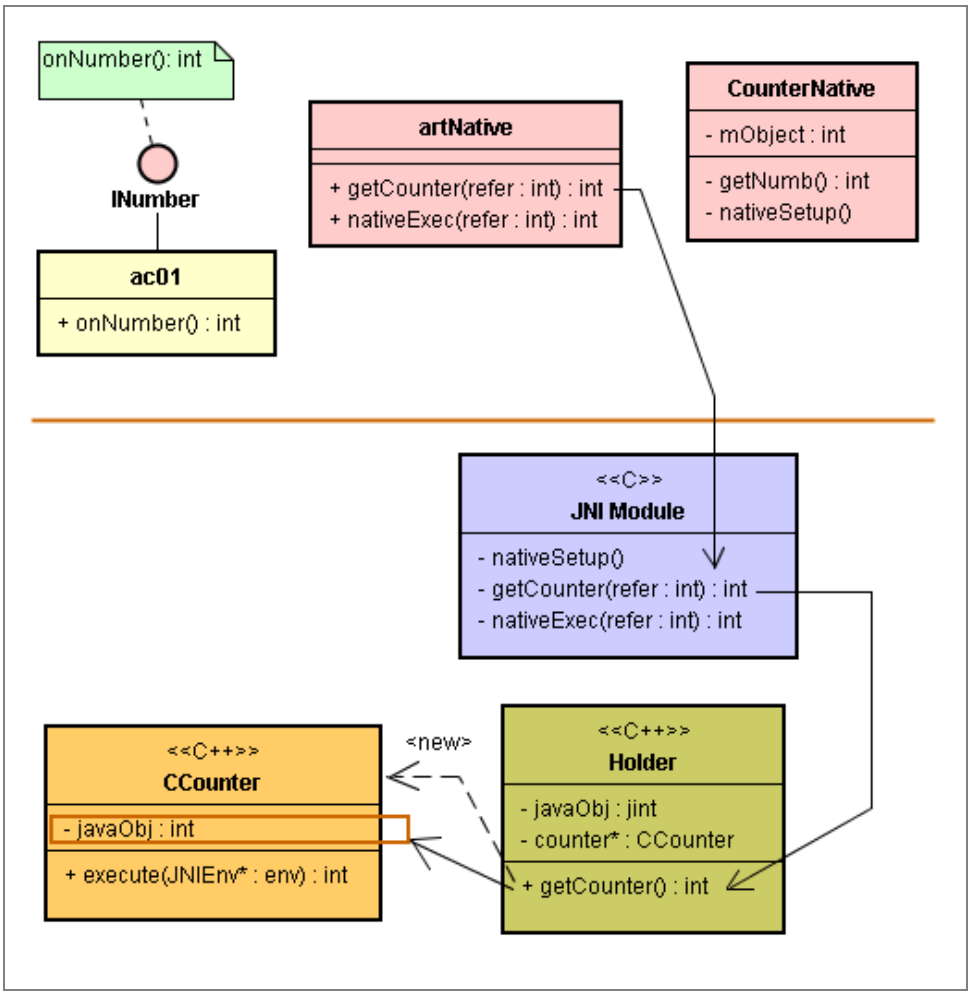


圖 8　創建更底層的 C++對象

創建 CCounter 對象時，它內部還沒有 n 值。當 CCounter 對象執行 execute()函數時，需要 n 值才能計算出 sum 值。此時，可藉由 JNIEnv*指標而調用到 CounterNative 類的 getNumb()函數，再透過 INmuber 接口調用到 ac01 的 onNumber()函數，以便取得 n 值。

CCounter對象就如同一部汽車的引擎部份，而ac01對象則如同輪胎部分，基於這個比喻，可以看出 CCounter 類居於主導者的地位。如下圖：

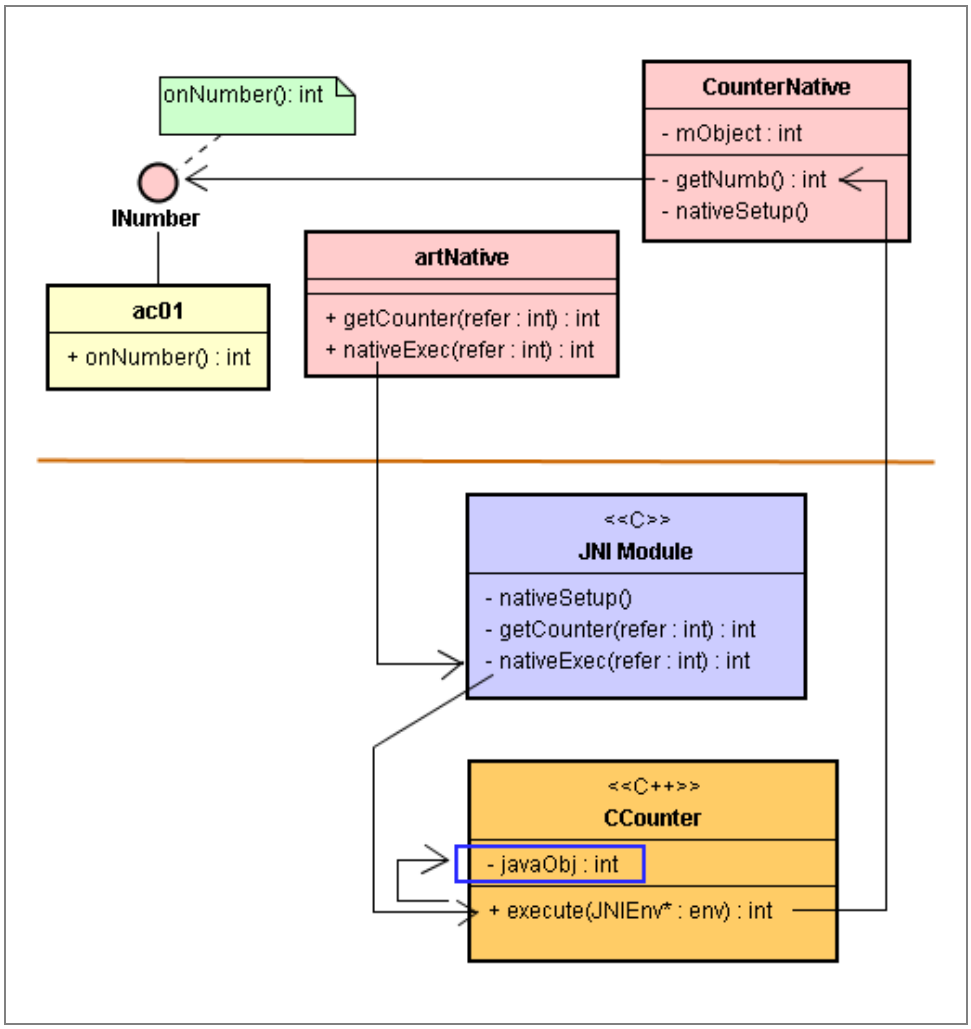


圖 9　底層 C++對象掌握控制權

依循這項技巧，底層 C++對象就能隨時主動調用上層的 Java 對象。也就是，此刻 C++對象擁有主動調用的控制權。

- 定義 INumber 接口，如下：

// INumber.java

```
package com.misoo.counter;

public interface INumber {
    int onNumber();
}
```

- 撰寫 Java 層的 actNative 和 CounterNative 類，如下：

// actNative.java

```
package com.misoo.counter;

public class actNative {
        public static native int getCounter(int refer);
        public static native int nativeExec(int refer);
}
```

上述的 actNative 類裡定義了 1 個本地函數。

// CounterNative.java

```
package com.misoo.counter;

public class CounterNative {
        public int mObject;
        private INumber listener;

        static {    System.loadLibrary("MyCounter9");    }
        public CounterNative()
                        {      nativeSetup();    }
        public void setOnNumber(INumber plis)
                        {    listener = plis;    }
        private int getNumb()
                        {    return listener.onNumber();    }
        private native void nativeSetup();
    }
```

這 CounterNative 類裡定義了 nativeSetup()本地函數。它將創建 C++對象來與 CounterNative 對象相互對映。

- 編譯 JNI-09-holder 專案，產出*.class 檔案。
- 使用 javah 工具從*.class 產出*.*h* 標頭檔。

- 撰寫*.cpp 實作檔案，如下：

/ com.misoo.counter.CounterNative.cpp */*

```
#include "com_misoo_counter_actNative.h"
#include "com_misoo_counter_CounterNative.h"

class CCounter {
      jint javaObj;
public:
      CCounter(int jo)   {   javaObj = jo;     }
      int execute(JNIEnv *env) {
           jobject jo = (jobject)javaObj;
           jclass joClazz = (jclass)env->GetObjectClass(jo);
           jmethodID mid = env->GetMethodID(joClazz, "getNumb", "()I");
           int numb = (int)env->CallIntMethod(jo, mid);

           int i, sum = 0;
           for(i=0; i<=numb; i++)    sum+=i;
           return sum;
   }};
//----------------------------------------
class Holder{
public:
      jint javaObj;
      CCounter *counter;
public:
      int getCounter() {
         counter = new CCounter(javaObj);
         return (int)counter;
   }};

JNIEXPORT void JNICALL Java_com_misoo_counter_CounterNative_nativeSetup
   (JNIEnv *env, jobject thiz) {
     Holder *ho = new Holder();      // create Holder object

     jclass clazz = (jclass)env->GetObjectClass(thiz);
     jfieldID fid = (jfieldID)env->GetFieldID(clazz, "mObject", "I");
     env->SetIntField(thiz, fid, (jint)ho); // set reference for Java object

     jobject gThiz = (jobject)env->NewGlobalRef(thiz);
     ho->javaObj = (jint)gThiz;
}
JNIEXPORT jint JNICALL Java_com_misoo_counter_actNative_getCounter
   (JNIEnv *env, jclass clazz, jint refer){
```

```
        Holder *ho = (Holder*)refer;
        return (jint)ho->getCounter();    // ask Holder to create CCounter object
}
JNIEXPORT jint JNICALL Java_com_misoo_counter_actNative_nativeExec
   (JNIEnv *env, jclass clazz, jint refer) {
        CCounter *co = (CCounter*)refer;
        return (jint)co->execute(env);
}
```

關於nativeSetup()函數的動作

上述nativeSetup()函數裡先創建一個C++層的Holder對象，然後此對象的指標值儲存於CounterNative對象的mObject屬性裡。同時，也將CounterNative對象的指標值儲存於Holder對象的javaObj屬性裡，如此建立了CounterNative對象與CCounter對象之雙向連結。如下圖：

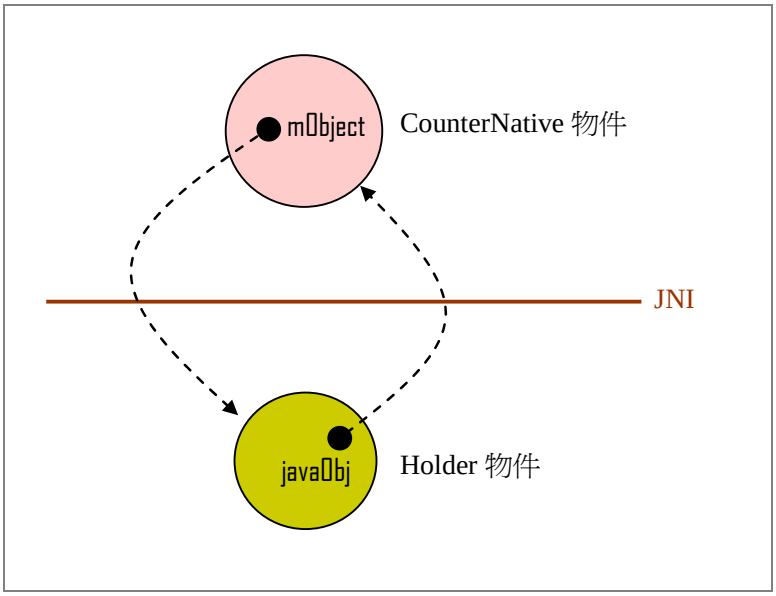


圖 10　Holder 對象與 Java 對象之雙向連結機制

關於getCounter()函數的動作

Holder 類的 getCounter()函數創建一個 CCounter 對象，並且讓 CCounter 對象連結到 CounterNative 對象。如下圖：

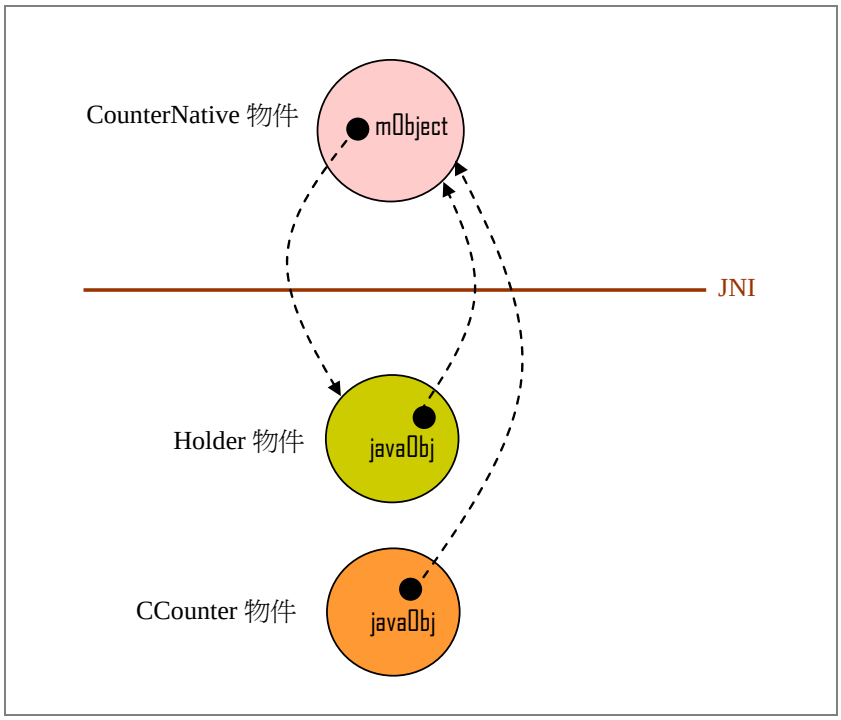


圖 11　CCounter 對象與 Java 對象的連結機制

- 使用 NDK 環境，編譯、連結而產出*.so 程序庫(Library)。
- 將*.so 程序庫拷貝到 JNI-09-holder 專案裡。
- 編修 ac01.java 類，如下：

// ac01.java

```
package com.misoo.pk01;
import com.misoo.counter.CounterNative;
import com.misoo.counter.INumber;
import com.misoo.counter.actNative;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
```

```
public class ac01 extends Activity implements OnClickListener , INumber{
   private Button btn, btn2, btn3;
   private CounterNative cn;
   private int counter;

   @Override public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      LinearLayout layout = new LinearLayout(this);
      layout.setOrientation(LinearLayout.VERTICAL);
      btn = new Button(this);          btn.setId(101);
      btn.setBackgroundResource(R.drawable.heart);
      btn.setText("getCounter");      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.heart);
      btn2.setText("execute");         btn2.setOnClickListener(this);
      layout.addView(btn2, param);
      btn3 = new Button(this);        btn3.setId(103);
      btn3.setBackgroundResource(R.drawable.gray);
      btn3.setText("exit");               btn3.setOnClickListener(this);
      layout.addView(btn3, param);
      setContentView(layout);
      //-----------------------------------------------
      cn = new CounterNative();
      cn.setOnNumber(this);
    }
@Override public void onClick(View v) {
       switch(v.getId()){
       case 101:    counter = actNative.getCounter(cn.mObject);
                    setTitle("getCounter OK");      break;
       case 102:    int sum = actNative.nativeExec(counter);
                    setTitle("Sum = " + sum);       break;
       case 103:    finish();      break;
  }}
@Override public int onNumber() {      return 11;    }
}
```

當你從畫面裡按下<run-01>按鈕，就執行指令：

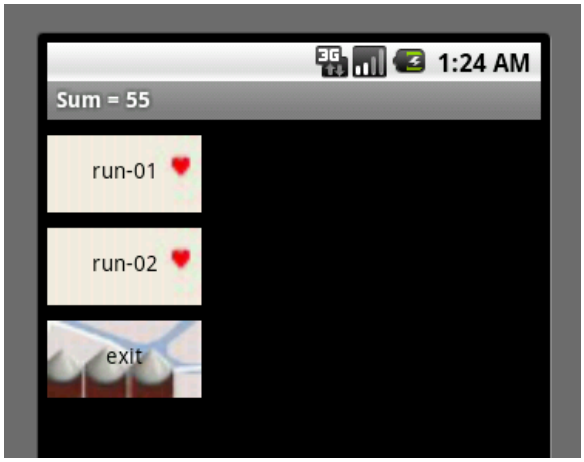
```
counter = actNative.getCounter(cn.mObject);
```

此時，cn.mObject就是Holder對象的參考值，所以C模塊的getCounter()函數就創建CCounter新對象，並且將新對象的參考值回傳給ac01，存入counter變量裡。

當你從畫面裡按下<run-02>按鈕，就執行指令：

```
int sum = actNative.nativeExec(counter);
```

此時，ac01 將 CCounter 對象參考傳遞給 JNI 模塊的 nativeExec()函數，它就調用 CCounter 的 execute()函數，計算出 sum 值之後，把 sum 值傳送到 Java 層顯示出來，如下的畫面：



以上說明了如何底層 C++對象如何取得 Java 層對象裡的資料。◆