Android 開發筆記 - 使用 NDK / JNI 實作從底層呼叫上層 (C call Java)

一般使用 JNI 的情境，不外呼從 Java 呼叫 C ，接著在 C (底層)運算完後，把數值透過 return 的方式傳回給 Java (上層)端，這在官方 NDK 教學或是之前的筆記都可以看到簡單的範例：Android 開發教學筆記 - 使用 Android NDK (Native Development Kit)。然而，如果要從 Native C 去呼叫 Java function 的話，就不是那麼直觀，例如在 Unix 系統上的程式開發，則是需取得一個門牌號碼(process id)，接著才跟他溝通，或是直接透過執行新的程式，去指定運行某個 function 的這種架構等。在此使用的環境為 Ubuntu 10.04 desktop 64-bit，提供簡易 C call Java 的範例。

一般 C call Java 的話，則需要建立一個 Java 運行的 JVM 通道，接著才開始找尋物件、函數並開始運作：

@ c_to_java.c

```
#include <stdio.h>
#include <stdlib.h>
#include <jni.h>

int main()
{
JNIEnv *env;
JavaVM *jvm;

JavaVMInitArgs vm_args;
JavaVMOption options[1];

// launch Jvm
options[0].optionString = "-Djava.class.path="; // add user classes
vm_args.version = JNI_VERSION_1_6; //JDK version.
vm_args.options = options;
vm_args.nOptions = 1;
```

```
if( JNI_CreateJavaVM(&jvm, (void*) &env, &vm_args) < 0 )
{
fprintf( stderr , "Launch JVM Error\n" );
exit(1);
}

// find the obj & method
jclass my_class;
jmethodID my_main;

if( !( my_class = (*env)->FindClass( env, "MyJavaClass") ) )
{
fprintf( stderr , "'Class' Not Found\n" );
exit(1);
}

if( !( my_main = (*env)->GetStaticMethodID( env, my_class , "main" ,  "([Ljava/lang/String;)V" ) ) )
fprintf( stderr , "'main' Not Found\n" );
else// Call main function
(*env)->CallStaticVoidMethod( env, my_class, my_main, NULL);

// finish
(*jvm)->DestroyJavaVM(jvm);

return 0;
}


@ MyJavaClass.java


class MyJavaClass
{
public static void main( String []arg)
{
```

```
System.out.println("Hello World");
}
}
```

@ Makefile

```
CC=gcc
INCLUDE=-I/usr/lib/jvm/java-6-sun/include/ -I/usr/lib/jvm/java-6-sun/include/linux/
LIB=/usr/lib/jvm/java-6-sun/jre/lib/amd64/server/libjvm.so

all:
	javac MyJavaClass.java
	$(CC) $(INCLUDE) $(LIB) c_to_java.c
clean:
	rm -f ./a.out  ./*.class
```

@ /etc/ld.so.conf.d/jvm.conf

```
/usr/lib/jvm/java-6-sun/jre/lib
/usr/lib/jvm/java-6-sun/jre/lib/amd64
/usr/lib/jvm/java-6-sun/jre/lib/amd64/server
```

執行結果：

```
$ make
$ ./a.out
Hello World
```

由於我是在 Ubuntu 10.04 desktop 64-bit 環境，安裝 Sun Java6，所以 gcc 編譯要透過 -I 指定 header file 和 libjvm.so 檔案外，在執行 a.out 的時候也會出現找不到 share library 的情，這時候要去設定 /etc/ld.so.conf，在

此是透過建立 /etc/ld.so.conf.d/jvm.conf 並再用 sudo ldconfig -v 進行 loading，之後在執行 a.out 就不會出錯了

接著是 Android 端之 JNI 從底層 C 呼叫上層 Java 端的部分，稍稍不一樣，整個程式運作流程：

1. 啟動 Android app 時，進入程式內的 OnCreate 時，將本身 app 的資訊丟進 C 底層，並使用全域變數紀錄。
2. 由於 C 帶有 Java 端相關資訊，因此 C 可以呼叫 Java 端的相關函數，完成了底層往上層呼叫的動作。

建構於 Android 開發教學筆記 - 使用 Android NDK (Native Development Kit) 的範例：

請留意 NDK 版本，此例是用 android-ndk-r5c，用新版的話，相關路徑要記得更新

目錄結構：

~/android-ndk-r5c
~/workspace/MyNDK
~/workspace/MyNDK/jni
~/workspace/MyNDK/jni/my-jni.c
~/workspace/MyNDK/jni/Android.mk
~/workspace/MyNDK/jni/Makefile (非必要)

@ MyNDK.java

```
package com.example.ndk;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
```

```
import android.widget.TextView;
import java.lang.ref.WeakReference;

public class MyNDK extends Activity {
    /** Called when the activity is first created. */
static int count;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText( stringFromJNI() );
        setContentView(tv);
        count = 0;

        initMyJNI(new WeakReference<MyNDK>(this));
    }

    private static void callback( int pid )
    {
        Log.i("MyNDK","Pid:"+pid+",Count:"+count);
    count++;
    }

    private native void initMyJNI( Object weak_this );

    public native String  stringFromJNI();

    static {
        System.loadLibrary("my-jni");
    }
}
```

@ my_jni.c

```
#include <stdio.h>
#include <string.h>
#include <jni.h>
#include <pthread.h>
#include <android/log.h>

#define LOG_TAG"MyJNI"
#define LOGV(...)__android_log_print( ANDROID_LOG_VERBOSE,LOG_TAG, __VA_ARGS__ )
#define LOGD(...)__android_log_print( ANDROID_LOG_DEBUG,LOG_TAG, __VA_ARGS__ )
#define LOGI(...)__android_log_print( ANDROID_LOG_INFO,LOG_TAG, __VA_ARGS__ )
#define LOGW(...)__android_log_print( ANDROID_LOG_WARN,LOG_TAG, __VA_ARGS__ )
#define LOGE(...)__android_log_print( ANDROID_LOG_ERROR,LOG_TAG, __VA_ARGS__ )

// --- Header Begin ---
#defineJAVA_CALLBACK_FUNCTION_NAME"callback"
#defineJAVA_CALLBACK_FUNCTION_ARGS_TYPE"(I)V"

JNIEXPORT void JNICALL Java_com_example_ndk_MyNDK_initMyJNI( JNIEnv* env, jobject thiz, jobject weak_this
);
void * threadJobRun();

static const char *classPathName = "com/example/ndk/MyNDK";
static JNINativeMethod methods[] = {
{
"initMyJNI" ,
"(Ljava/lang/Object;)V" ,
(void *)Java_com_example_ndk_MyNDK_initMyJNI
}
};

static JavaVM *jvm;// global for JVM init Info
static jobject mObject;// get the object
static jclass mClass;// get the class
```

```
static jmethodID mMethodID;// get the method

static pthread_t thread;

// --- End Of Header ---

static int registerNativeMethods(JNIEnv* env, const char* className, JNINativeMethod* gMethods, int
numMethods)
{
jclass clazz;

if( !( clazz = (*env)->FindClass( env, className ) ) )
{
LOGE( "Unable to get the Class at registerNativeMethods" );
return JNI_FALSE;
}

if( ( (*env)->RegisterNatives( env, clazz, gMethods, numMethods ) ) < 0 )
{
LOGE( "Unable to register the methods at registerNativeMethods" );
return JNI_FALSE;
}
return JNI_TRUE;
}

jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
LOGI("IN JNI_OnLoad");
JNIEnv *env;
if( ( (*vm)->GetEnv( vm, (void **)&env, JNI_VERSION_1_6 ) ) != JNI_OK )
{
LOGE( "Unable to get the env at JNI_OnLoad" );
return -1;
}
if (!registerNativeMethods( env, classPathName, methods, sizeof(methods) / sizeof(methods[0])))
```

```
{
LOGE( "Unable to register native methods at JNI_OnLoad" );
return -1;
}
jvm = vm;
return JNI_VERSION_1_6;
}

JNIEXPORT void JNICALL Java_com_example_ndk_MyNDK_initMyJNI( JNIEnv* env, jobject thiz, jobject weak_this
)
{
jclass clazz;

mClass = NULL;
mObject = NULL;
mMethodID = NULL;

if( !( clazz = (*env)->GetObjectClass( env, thiz ) ) )
{
LOGE( "Unable to get the object class" );
return;
}
if( !( mClass = (*env)->NewGlobalRef( env, clazz ) ) )
{
LOGE( "Unable to get the class ref" );
return;
}
if( !( mObject = (*env)->NewGlobalRef( env, weak_this ) ) )
{
LOGE( "Unable to get the object ref" );
return;
}

if( !( mMethodID = (*env)->GetStaticMethodID( env, mClass, JAVA_CALLBACK_FUNCTION_NAME,
JAVA_CALLBACK_FUNCTION_ARGS_TYPE ) ) )
```

```
{
LOGE( "Unable to get the method ref" );
return;
}

LOGI("IN initMyJNI");

if( mClass && mMethodID )// first call
(*env)->CallStaticVoidMethod( env, mClass , mMethodID , (int)getpid() );

// thread
pthread_create( &thread, NULL, threadJobRun, NULL);
}
void * threadJobRun()
{
JNIEnv *env;
int isAttached;

isAttached = 0;
env = NULL;
LOGI("IN threadJobRun");
if( jvm )
{
LOGI("IN threadJobRun with JVM");

if( ( (*jvm)->GetEnv( jvm, (void**) &env, JNI_VERSION_1_6 ) ) < 0 )
{
LOGI( "Unable to get env at threadJobRun" );
if( ( (*jvm)->AttachCurrentThread( jvm, &env, NULL) ) < 0 )
{
LOGE( "Unalbe to attach current thread at threadJobRun" );
return NULL;
}
isAttached = 1;
}
```

```
while( 1 )
{
sleep(1);

// --- jobs begin
if( mClass && mMethodID )
{
LOGI( "call mClass & mMehtodID" );
(*env)->CallStaticVoidMethod( env, mClass , mMethodID , (int)getpid() );
}
// --- end of jobs
}

if( isAttached )
{
(*jvm)->DetachCurrentThread( jvm );
}
}
return NULL;
}

jstring Java_com_example_ndk_MyNDK_stringFromJNI( JNIEnv* env, jobject thiz )
{
    return (*env)->NewStringUTF(env, "Hello from My JNI !");
}
```

@ Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
```

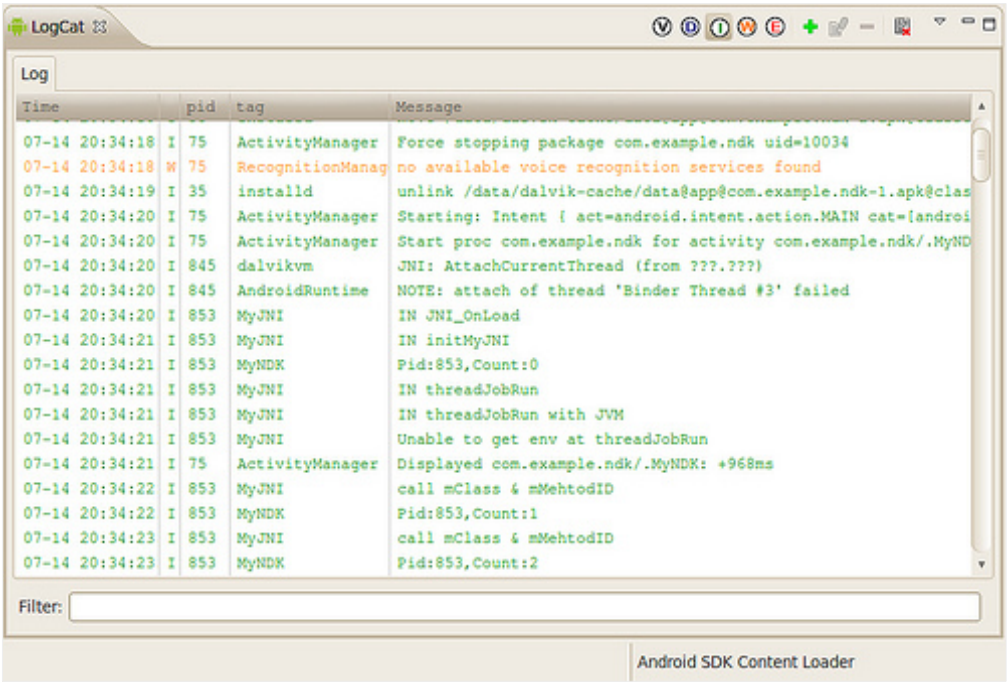
```
LOCAL_MODULE    := my-jni
LOCAL_SRC_FILES := my-jni.c

LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib -llog

include $(BUILD_SHARED_LIBRARY)
```

@ Makefile (非必要)

```
all:
cd ../ && sh ~/android-ndk-r5c/ndk-build
```

其中 Makefile 是我自己工作上使用的，我習慣用 vim + makefile 開發，所以會需要這個檔案，所以此檔非必要，其他檔案如同 Android 開發教學筆記 - 使用 Android NDK (Native Development Kit) 敘述。比較重要的是 my-jni.c 裡頭，有一個 JNI_OnLoad 和 initMyJNI 函數，在 app 一開始執行後，會先透過 JNI_OnLoad 將相關的資訊紀錄在 global variables 中，如 JVM 以及相關動作等，接著在 Java 呼叫 initMyJNI 時，才正式將 Java Object 紀錄起來，例如 Class 、Method 等，另外，在 initMyJNI 時，已經算是做到 C call Java 之 callback 的動作了(first call)，只是我的目標是在 C 層跑一個 thread ，等到有事件時去更新 Java 層，所以有多一個 thread 的運行，在 threadJobRun 中則是 thread 的動作囉。

程式運作過程中，我有透過 Logcat 印出點訊息，可以看得清楚：



其中 tag 為 MyJNI 是指 C 那層的動作，可以看到流程是 IN JNI_OnLoad、IN initMyJNI，接著就是第一次 c call java，在 Java 層印出 Pid:853, Count:0 的訊息，接著才進入 Thread ，然後每秒印出一次 Pid:853, Count:number 等資訊。

其他比較重要的是 C 裡頭也能可以呼叫 logcat ，記得在 Android.mk 中要加入 LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib -llog 才不會編譯失敗(擺放的位置也很重要，擺在上頭會失敗)；而 c call java 的部分，在 Java 那邊定義的函數必須是 static 的，此例為 private static void callback(int pid)，不然也會出錯(因為 C 裡頭是用 GetStaticMethodID)，我有去找一下其他 open source (Kwaak3 / KwaakJNI.java)，寫法也是這樣。

最後一提，上述 C 程式存在很多地方沒寫好，例如資源的釋放等，在此僅供用來了解流程用途。

參考資料：

- kwaak3 - Port of a famous 3d shooter to Android - Google Project Hosting
- 回應 albert_espresso： 由 C 呼叫 java 中的物件--Android 大舞台 文章講義分享 --Google Android 論壇