多線程 JNI Native

Java 多線程執行同一 Native 函數

Java 程式可能會有多個線程幾乎同時先後進入同一個 Native 函數裡執行。

Case 1: 各自使用區域變數(沒有線程安全問題)

只要各線程不共用變數，就不會干擾到別線程執行的正確性了。例如下述範例：

// JTX02.java

```
package com.misoo.thread;
import java.lang.ref.WeakReference;
import com.misoo.pk01.ac01;
import android.os.Handler;
import android.os.Message;

public class JTX02 {
   private static Handler h;
   private static int count;

   static {
              System.loadLibrary("JTX02_jni");
     }
   public JTX02(){
            Init(new WeakReference<JTX02>(this));

            count = 0;
            h = new Handler(){
                   public void handleMessage(Message msg) {
                          count++;
                       if(count == 1)
                            ac01.ref.setTitle("Sum: " + String.valueOf(msg.arg1));
                       else if(count == 2)
                                  ac01.tv.setText("Sum: " + String.valueOf(msg.arg1));
                  }
            };
   }
   public long calculate(){
         Thread t1 = new Thread(){
               public void run() {
                     String sss = JTX02.this.execute();
               }
         };
```

```
        t1.start(); //sub thread
        execute(); //main thread
        return 0;
   }
   @SuppressWarnings("unused")
   private static void callback(int a, int b){
                    Message m = h.obtainMessage(1, a, 3, null);
                      h.sendMessage(m);
     }
   private native void Init(Object weak_this);
   private native String execute();
  }
```

主線程進入 calculate()函數，立即誕生一個小線程，去執行 Native 的 execute()函數，同時自己去執行 execute()函數。

// com_misoo_thread_JTX02.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_misoo_thread_JTX02 */

#ifndef _Included_com_misoo_thread_JTX02
#define _Included_com_misoo_thread_JTX02
#ifdef __cplusplus

extern "C" {
#endif
/*
  * Class:          com_misoo_thread_JTX02
  * Method:        nativeSetup
  * Signature: (Ljava/lang/Object;)V
  */
JNIEXPORT void JNICALL Java_com_misoo_thread_JTX02_Init
   (JNIEnv *, jobject, jobject);

/*
  * Class:          com_misoo_thread_JTX02
  * Method:        execute
  * Signature: ()J
  */
JNIEXPORT jstring JNICALL Java_com_misoo_thread_JTX02_execute
   (JNIEnv *, jobject);


#ifdef __cplusplus
}
#endif
#endif
```

/   com_misoo_thread_JTX02.cpp   */*

```
#include "com_misoo_thread_JTX02.h"
#include <utils/Log.h>
#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>

using namespace android;

jmethodID      mid;
jclass         mClass;       // Reference to JTX02 class
jobject        mObject;      // Weak ref to JTX02 Java object to call on
char sTid[20];

void Thread_sleep(int t)
{
     timespec ts;
     ts.tv_sec = t;
     ts.tv_nsec = 0;
     nanosleep(&ts, NULL);
     return;
}

JNIEXPORT void JNICALL
Java_com_misoo_thread_JTX02_Init(JNIEnv *env, jobject thiz, jobject weak_this)
{
   jclass clazz = env->GetObjectClass(thiz);
   mClass = (jclass)env->NewGlobalRef(clazz);
   mObject    = env->NewGlobalRef(weak_this);
   mid = env->GetStaticMethodID(mClass, "callback","(II)V");
   return;
  }

JNIEXPORT jstring JNICALL
Java_com_misoo_thread_JTX02_execute(JNIEnv *env, jobject thiz){
     int sum = 0;
     for(int i = 0; i<=10; i++){
                 sum += i;
                 Thread_sleep(1);
                }
       //----------------------------------------------------------
     env->CallStaticVoidMethod(mClass, mid, sum, 0);
       //----------------------------------------------------------
     sprintf(sTid, "%lu", 0);
     jstring ret = env->NewStringUTF(sTid);
     return ret;
}
```

無論誰先進入 execute()函數裡執行，由於 sum 等變數都是屬於 execute()函數內部的，每一個線程進來都會誕生自己的 sum 等變數，所以沒有現成安全的顧慮。

Case 2: 使用共用的變數(可能發生線程安全問題)

如果將上述的 execute()函數更改如下：

```
/*    com_misoo_thread_JTX03.cpp         */
#include "com_misoo_thread_JTX03.h"
#include <utils/Log.h>
#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>

using namespace android;

jmethodID      mid;
jclass          mClass;        // Reference to JTX03 class
jobject         mObject;       // Weak ref to JTX03 Java object to call on
char sTid[20];
int sum;

void Thread_sleep(int t)
{
      timespec ts;
      ts.tv_sec = t;
      ts.tv_nsec = 0;
      nanosleep(&ts, NULL);
      return;
}

JNIEXPORT void JNICALL
Java_com_misoo_thread_JTX03_Init(JNIEnv *env, jobject thiz, jobject weak_this)
{
    jclass clazz = env->GetObjectClass(thiz);
    mClass = (jclass)env->NewGlobalRef(clazz);
    mObject    = env->NewGlobalRef(weak_this);
    mid = env->GetStaticMethodID(mClass, "callback","(II)V");
    return;
  }

JNIEXPORT jstring JNICALL
Java_com_misoo_thread_JTX03_execute(JNIEnv *env, jobject thiz){
      sum = 0;
      for(int i = 0; i<=10; i++){
                 sum += i;
```

```
                Thread_sleep(1);
              }
     //---------------------------------------------------------
    env->CallStaticVoidMethod(mClass, mid, sum, 0);
     //---------------------------------------------------------
    sprintf(sTid, "%lu", 0);
    jstring ret = env->NewStringUTF(sTid);
    return ret;
}
```

也就可能發生線程安全問題了。

多個 Java 線程之同步(Synchronization)

　　當會發生線程衝突時，如何呢? 請看範例：

// JTX04.java

```
package com.misoo.thread;
import java.lang.ref.WeakReference;
import com.misoo.pk01.ac01;

import android.os.Handler;
import android.os.Message;
import android.os.Process;

public class JTX04 {
   ………
public long calculate(){
          Thread t1 = new Thread(){
               public void run() {
                     JTX04.this.execute(JTX04.this);
               }
          };
          t1.start();
          try {
             Thread.sleep(2000);
    } catch (InterruptedException e) {
             e.printStackTrace();
      }
      String ss = execute(this);
      ac01.ref.setTitle("ss: " + ss);
    return 0;
  }
  ……….
  private native void Init(Object weak_this);
  private native synchronized String execute(Object oSync);
}
```

/   com_misoo_thread_JTX04.cpp   */*

```
#include "com_misoo_thread_JTX04.h"
#include <utils/Log.h>
#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>
#include "android_runtime/AndroidRuntime.h"

using namespace android;

JavaVM *gJavaVM;
jmethodID      mid;
jclass          mClass;         // Reference to JTX04 class
jobject         mObject;        // Weak ref to JTX04 Java object to call on
char sTid[20];
unsigned int e1;
int x;
int sum;
long test;


JNIEXPORT jstring JNICALL
Java_com_misoo_thread_JTX04_execute(JNIEnv *env, jobject thiz, jobject
syncObj){
     env->MonitorEnter(syncObj);
     sum = 0;
     for(int i = 0; i<=10; i++)
                 {
                   sum += i;
                   Thread_sleep(1);
                 }
       //--------------------------------------------------------
     env->CallStaticVoidMethod(mClass, mid, sum, 666);
       //--------------------------------------------------------

     env->MonitorExit(syncObj);

     long pid = getpid();
     sprintf(sTid, "%lu", test);
     jstring ret = env->NewStringUTF(sTid);
     return ret;
}
………
}
```

執行到指令：

```
    JTX04.this.execute(JTX04.this);
```
和

```
   String ss = execute(this);
```
時都把目前的Java 對象(即JTX04對象)傳遞給Native的execute()函數。

先進入execute()函數的線程會先執行到指令：

```
   env->MonitorEnter(syncObj);
```
也就向JTX04對象索取鑰匙(Key)。由於JTX04對象只要一把鑰匙，所以其他後進入的線程只好停下來等待。當執行到指令：

```
    env->MonitorExit(syncObj);
```
也就把鑰匙(Key)交還給JTX04對象，讓等待中的其他線程可以逐一進入。

如何誕生 Native 層的子線程

在 Java 層的線程可以誕生子線程；而在 Native 函數裡執行的線程也能誕生子線程。

```
/*    com_misoo_thread_JTX05.cpp        */
#include "com_misoo_thread_JTX05.h"
#include <utils/Log.h>
#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>
#include "android_runtime/AndroidRuntime.h"
using namespace android;

JavaVM *gJavaVM;
jmethodID      mid;
jclass         mClass;       // Reference to JTX05 class
jobject        mObject;      // Weak ref to JTX05 Java object to call on
char sTid[20];
unsigned int e1;
int x;
int sum;
pthread_t thread;
void* trRun( void* );

typedef multimap<string, JNIGlobalRef<jobject> *> MapOfObjects;

JNIEXPORT jstring JNICALL
Java_com_misoo_thread_JTX05_execute(JNIEnv *env, jobject thiz, jobject
syncObj){
    ………
    int rr = pthread_create( &thread, NULL, trRun, NULL);
    ………
}
```

```
void* trRun( void* )
{
    .........
}
```

使用函數 pthread_create()函數來誕生 Native 層的子線程。

多個 Native 線程的安全問題

由於 Native 函數裡執行的線程也能誕生子線程，所以也應該注意其線程安全問題。例如：

// JTX07.java

```
package com.misoo.thread;
import java.lang.ref.WeakReference;
import com.misoo.pk01.ac01;

import android.os.Handler;
import android.os.Message;
import android.os.Process;

public class JTX07 {
   private long refer;
   private static Handler h;
   private static int count;

   static {
              System.loadLibrary("JTX07_jni");
     }
   public JTX07(){
            init(new WeakReference<JTX07>(this));
            count = 0;
            h = new Handler(){
                   public void handleMessage(Message msg) {
                          count++;
                       if(count == 1) {
                           ac01.ref.setTitle("env: " + String.valueOf(msg.arg1));
                           }
                       else if(count == 2)
                              ac01.tv.setText("env: " + String.valueOf(msg.arg1));
                 }
            };
   }

   public long calculate(){
      execute(this);
      //this.ssetTitle(ss);
      //ac01.ref.setTitle("ss: " + ss);
     return 0;
   }

   @SuppressWarnings("unused")
   private static void callback(int a, int b){
                    Message m = h.obtainMessage(1, a, 3, null);
                      h.sendMessage(m);
     }
   private native void init(Object weak_this);
   private native String execute(Object oSync);
   //private native String execute(int x);
  }
```

/   com_misoo_thread_JTX07.h   */*

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_misoo_thread_JTX07 */

#ifndef _Included_com_misoo_thread_JTX07
#define _Included_com_misoo_thread_JTX07
#ifdef __cplusplus

extern "C" {
#endif
/*
  * Class:         com_misoo_thread_JTX07
  * Method:        nativeSetup
  * Signature: (Ljava/lang/Object;)V
  */
void JNICALL Java_com_misoo_thread_Init
   (JNIEnv *, jobject, jobject);

/*
  * Class:         com_misoo_thread_JTX07
  * Method:        execute
  * Signature: (J)J
  */
jstring JNICALL Java_com_misoo_thread_Exec
   (JNIEnv *, jobject, jobject);


#ifdef __cplusplus
}
#endif
#endif
```

/   com_misoo_thread_JTX07.cpp   */*
```
#include "com_misoo_thread_JTX07.h"
#include <utils/Log.h>
#include <utils/IPCThreadState.h>
#include <utils/ProcessState.h>
#include "android_runtime/AndroidRuntime.h"

using namespace android;

JavaVM *gJavaVM;
jmethodID      mid;
jclass          mClass;         // Reference to JTX07 class
jobject         mObject;        // Weak ref to JTX07 Java object to call on
char sTid[20];
unsigned int e1;
int x;
int sum;
pthread_t thread;
void* trRun( void* );
void callBack(JNIEnv *);
jobject         mSyncObj;

//typedef multimap<string, JNIGlobalRef<jobject> *> MapOfObjects;

//-------------------------------------------------------------
void Thread_sleep(int t)
{
      timespec ts;
      ts.tv_sec = t;
      ts.tv_nsec = 0;
      nanosleep(&ts, NULL);
      return;
}

void JNICALL
Java_com_misoo_thread_setUp(JNIEnv *env, jobject thiz, jobject weak_this)
{
    jclass clazz = env->GetObjectClass(thiz);
    mClass = (jclass)env->NewGlobalRef(clazz);
    mObject    = env->NewGlobalRef(weak_this);
    mid = env->GetStaticMethodID(mClass, "callback","(II)V");
    return;
  }

jstring JNICALL
Java_com_misoo_thread_Exec(JNIEnv *env, jobject thiz, jobject syncObj){
      mSyncObj = env->NewGlobalRef(syncObj);

      int rr = pthread_create( &thread, NULL, trRun, NULL);
      Thread_sleep(4);
      callBack(env);
        //----------------------------------------------------------
      long pid = getpid();
      sprintf(sTid, "%lu", pid);
      jstring ret = env->NewStringUTF(sTid);
      return ret;
}
//------------------------------------------------------------------
void callBack(JNIEnv *env){
    env->MonitorEnter(mSyncObj);
    sum = 0;
    for(int i = 0; i<=10; i++)
                    {
                      sum += i;
                      Thread_sleep(1);
                    }
        //----------------------------------------------------------
      env->CallStaticVoidMethod(mClass, mid, sum, 666);
        //----------------------------------------------------------
    env->MonitorExit(mSyncObj);
}
//------------------------------------------------------------------

static const char *classPathName = "com/misoo/thread/JTX07";

static JNINativeMethod methods[] = {
        {"init",              "(Ljava/lang/Object;)V",
                                   (void *)Java_com_misoo_thread_setUp},
        {"execute",        "(Ljava/lang/Object;)Ljava/lang/String;",
                                   (void *)Java_com_misoo_thread_Exec}
};

/*
  * Register several native methods for one class.
  */
static int registerNativeMethods(JNIEnv* env, const char* className,
        JNINativeMethod* gMethods, int numMethods)
{
        jclass clazz;

        clazz = env->FindClass(className);
        if (clazz == NULL) {
                LOGE("Native registration unable to find class '%s'", className);
                return JNI_FALSE;
        }
        if (env->RegisterNatives(clazz, gMethods, numMethods) < 0) {
                LOGE("RegisterNatives failed for '%s'", className);
                return JNI_FALSE;
        }
        return JNI_TRUE;
}

static int registerNatives(JNIEnv* env)
{
    if (!registerNativeMethods(env, classPathName,
                                 methods, sizeof(methods) / sizeof(methods[0]))) {
        return JNI_FALSE;
    }

    return JNI_TRUE;
}


// ---------------------------------------------------------------------------

jint JNI_OnLoad(JavaVM* vm, void* reserved)
{

JNIEnv *env;
gJavaVM = vm;
int result;

LOGI("JNI_OnLoad called");
if (vm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        LOGE("Failed to get the environment using GetEnv()");
        return -1;
}

if (registerNatives(env) != JNI_TRUE) {
                LOGE("ERROR: registerNatives failed");
                goto bail;
        }

        result = JNI_VERSION_1_4;

bail:
        return result;
}

//----------------------------------------------------------------------------
void* trRun( void* )
{
        int status;
        JNIEnv *env;
        bool isAttached = false;

        Thread_sleep(1);
        status = gJavaVM->GetEnv((void **) &env, JNI_VERSION_1_4);
        if(status < 0) {
                LOGE("callback_handler: failed to get JNI environment, "
                          "assuming native thread");
                status = gJavaVM->AttachCurrentThread(&env, NULL);
                if(status < 0) {
                        LOGE("callback_handler: failed to attach "
                                  "current thread");
                        return NULL;
                }
                isAttached = true;
        }
        //-----------------------------------------------------------
        callBack(env);
        //-----------------------------------------------------------

        if(isAttached)
                gJavaVM->DetachCurrentThread();
        return NULL;
}
```

主線程誕生了子線程去執行 trRun()函數。必須先調用：

gJavaVM->AttachCurrentThread(&env, NULL);

才能取得子線程自己所屬的 JNIEnv 對象之參考了，並且調用 Callback()函數。之後，主線程也調用同一 Callback 函數。於是，在 Callback()函數裡，使用 env->MonitorEnter()和 env->MonitorExit(mSyncObj);指令來讓各線程能達到同步。

~~ END ~~