MICROOH 麦可网

Android-从程序员到架构师之路

出品人：Sundy
讲师：高焕堂（台湾）

http://www.microoh.com

C05_c

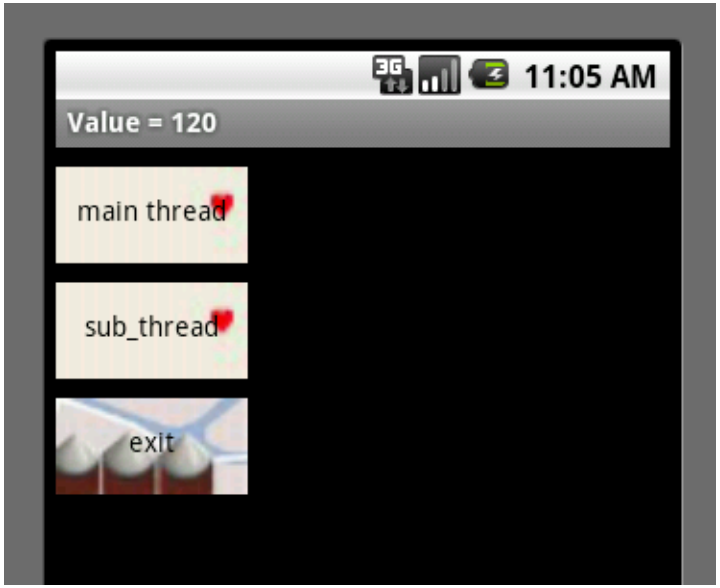
JNI：多个Java线程进入本地函数(c)

By 高煥堂

3、细说JNIEnv对象

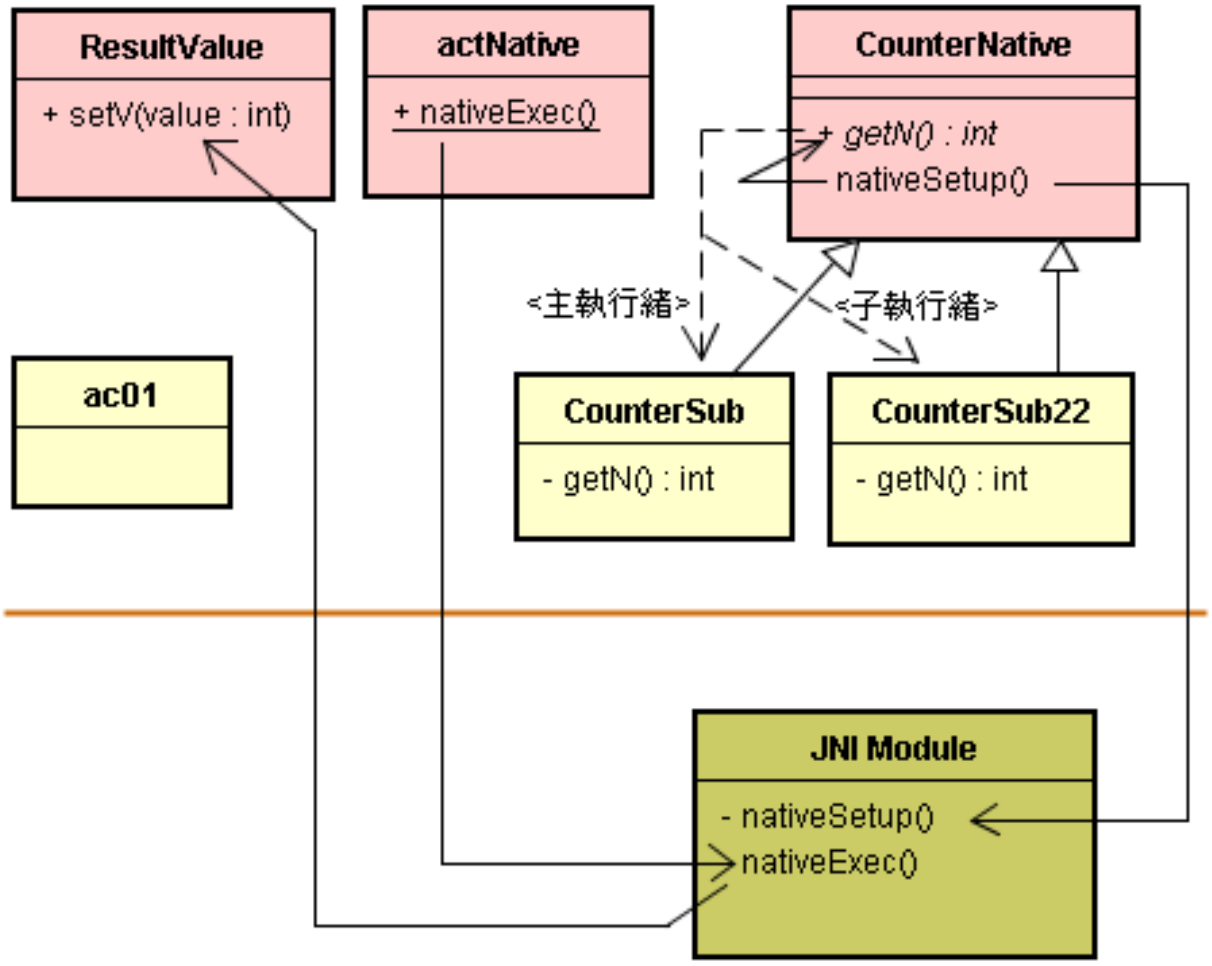
举例说明

- 主、子执行绪都能进入JNI层的C函数，又反过来进入Java层去调用Java函数。



- 按下<main thread>，主线程先进入C层去执行nativeSetup()函数，完毕后返回ac01。
- 再进入C层去执行nativeExec()函数，随后线程进入Java层的setV()函数。

- 按下<sub thread>，主线程就诞生一个子线程去执行Task的run()函数，然后依循刚才主线程的路径走一遍。


ResultValue
+ setV(value : int)
actNative
+ nativeExec()
CounterNative
+ getN() : int
nativeSetup()
<主執行緒>
<子執行緒>
ac01
CounterSub
- getN() : int
CounterSub22
- getN() : int
JNI Module
- nativeSetup()
nativeExec()

```
// actNative.java
public class actNative {
        public static native void nativeExec();
}
```

```
// CounterNative.java
abstract public class CounterNative {
        private int numb;
        public ResultValue rvObj;

        static {  System.loadLibrary("MyJT001");   }
        public CounterNative(){
                rvObj = new ResultValue();
                numb = getN();
                nativeSetup( rvObj );
            }
        abstract protected int getN();
        private native void nativeSetup( Object obj );
 }
```

```
// ResultValue.java
public class ResultValue {
   private int mValue;
   private String mThreadName;
   public int getValue(){  return mValue;  }
   private void setV(int value){
       mValue = value;
  }}
```

```
/*  com.misoo.counter.CounterNative.c  */
// ……..
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, jobject refer) {
	jclass clazz = (*env)->GetObjectClass(env, thiz);
	m_object = (jobject)(*env)->NewGlobalRef(env, thiz);
	m_fid = (*env)->GetFieldID(env, clazz, "numb", "I");

	jclass rvClazz = (*env)->GetObjectClass(env, refer);
	m_rv_object = (jobject)(*env)->NewGlobalRef(env, refer);
	m_rv_mid = (*env)->GetMethodID(env, rvClazz, "setV", "(I)V");
}
```

```
JNIEXPORT void 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);
}
```

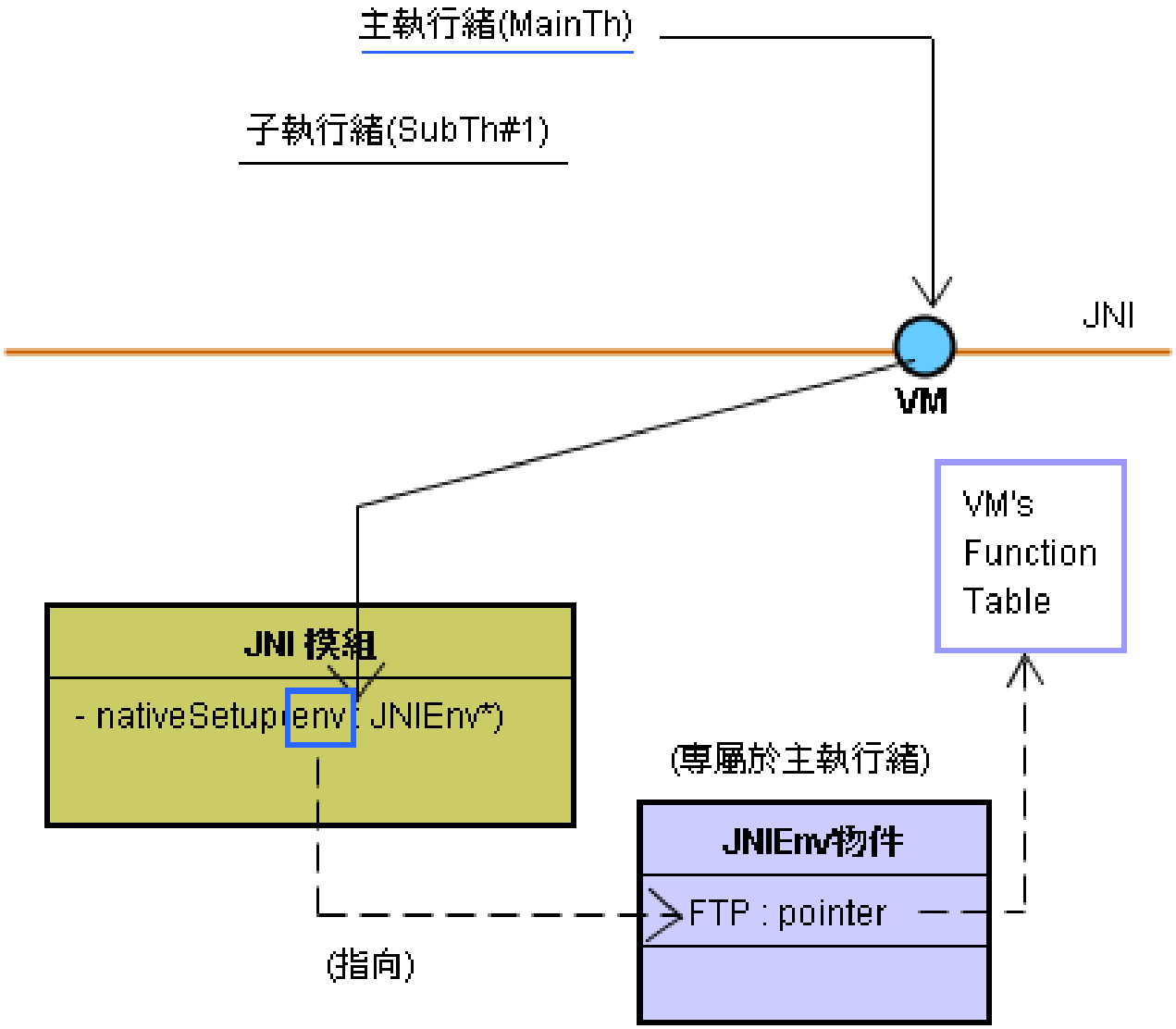
- 当Java 层的主线程准备进来执行这nativeSetup()函数时，VM就会诞生一个JNIEnv类别(或C结构)的对象，这个对象专属于主线程。
- 接着，将该对象的指针传递给nativeSetup()函数的第1个参数，如下：

```
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeSetup
        (JNIEnv *env, ......) {
            // .........
        }
```

- 不仅仅针对主线程而已，VM也替其它线程创建JNIEnv对象，也在该线程进入JNI层C函数时将其指针传递给第1个参数。
- 因此，不同的线程进入到nativeSetup()函数时，其所带进来的env参数值都是不一样的。

- 这样安排的好处之一是：每一个线程都不共享JNIEnv对象，此对象可以储存该线程相关的数据值，如此可以避免线程因共享对象或数据而引发的线程冲突问题，已就是有效提升了JNI环境下的多线程的安全性。

- JNIEnv对象内含一个指针，正指向VM的函数表(Function Table)。


主執行緒(MainTh)
子執行緒(SubTh#1)
JNI
VM
VM's
Function
Table
JNI 模組
- nativeSetup(env : JNIEnv*)
(專屬於主執行緒)
JNIEnv物件
FTP : pointer
(指向)

- 每一个线程第一次进入VM调用本地函数时，VM会替它诞生一个相对映的JNIEnv对象。
- Java层的线程调用C层的本地函数时，该线程必然经过VM，且VM一定替它诞生相对映的JNIEnv对象。

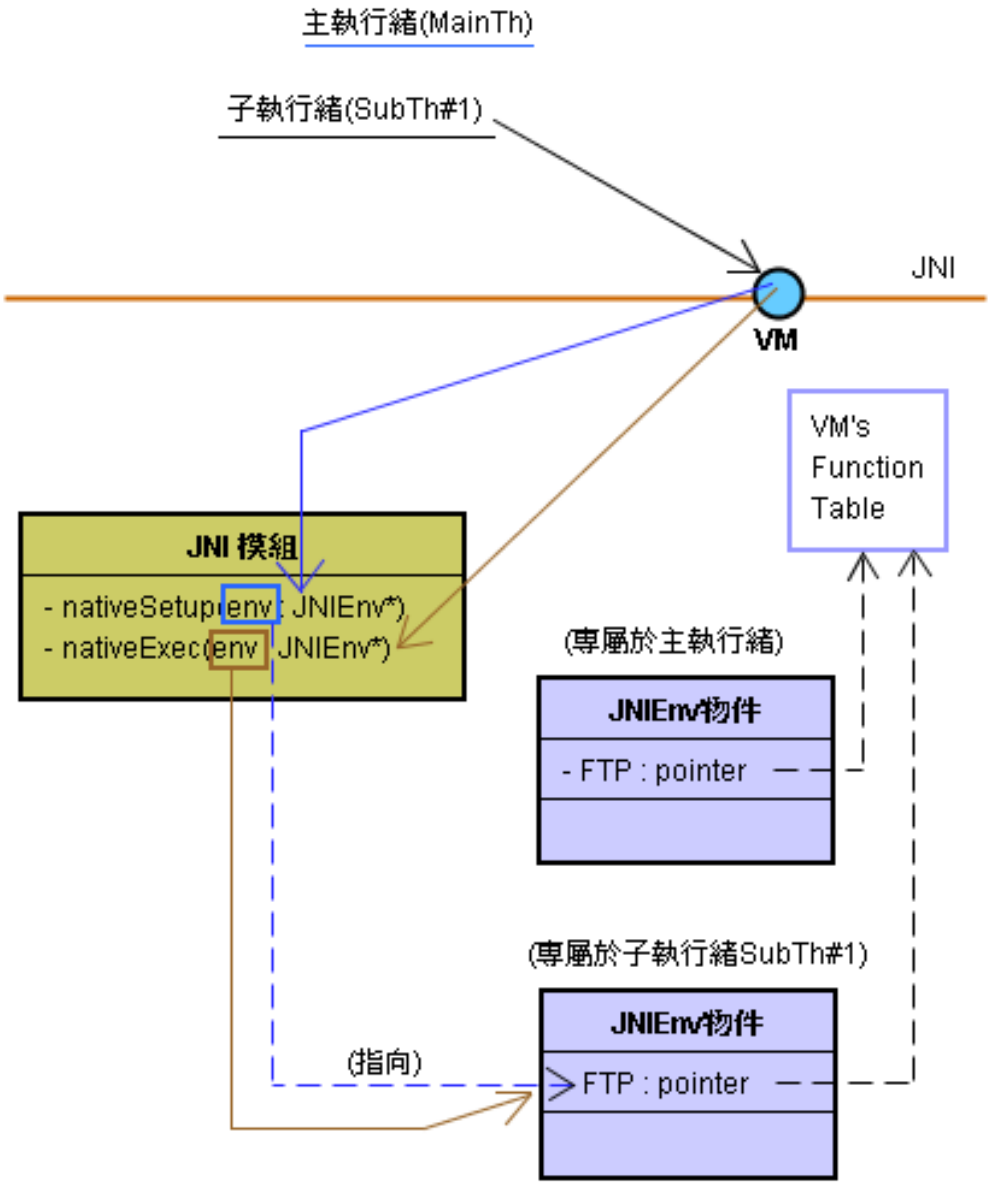
- 所以一个线程每次调用本地函数时，都会将其对映的JNIEnv对象指针值传递给本地函数。
- 每一个线程都有其专属的JNIEnv对象，所以不同的线程(例如th1和th2)调用同一个本地函数(例如f1(JNIEnv* env, ...)函数)时，这本地函数所接到的env值是不一样的。

线程不共享JNIEnv对象，成为”单线程”开发，不必烦恼线程安全问题，让本地函数的撰写单纯化了。

簡而言之…

- 在预设情形下，在某个线程第一次进入VM去执行JNI层C函数时，VM就会替它诞生专属的JNIEnv对象。只要该线程还存在着，就会一直保留它所专属的JNIEnv对象。

- 一个线程经常会多次进入VM去执行JNI层C函数，其中，每一次进入时，VM都会将其专属的JNIEnv对象指针传递给C函数的第1个参数(即env)。
- 因此，同一个线程每回进入C函数时，所带进来的env参数值都是相同的。如下图：


主執行緒(MainTh)
子執行緒(SubTh#1)
JNI
VM
VM's
Function
Table
JNI 模組
- nativeSetup(env : JNIEnv*)
- nativeExec(env : JNIEnv*)
(專屬於主執行緒)
JNIEnv物件
- FTP : pointer
(專屬於子執行緒SubTh#1)
JNIEnv物件
(指向)
FTP : pointer

- 由于某个线程(如子线程SubTh#1)先后执行nativeSetup()和nativeExec()两个函数，其带进来的env指标值都相同，其都指向同一个JNIEnv对象(即该线程专属的对象)，因此在两个函数里皆可以透过env指针而去取得该对象里的数据值，因而达成共享数据的目的。
- 采取JNIEnv机制，既能避免多线程的相互冲突，还能达成跨函数的数据共享。

```
// CounterSub.java
package com.misoo.pk01;
import com.misoo.counter.CounterNative;
public class CounterSub extends CounterNative{
        protected int getN() {  return 15;  }
}
```

```
// CounterSub22.java
package com.misoo.pk01;
import com.misoo.counter.CounterNative;
public class CounterSub22 extends CounterNative{
        protected int getN() {  return 10;  }
}
```

```
// ac01.java
// ........
public class ac01 extends Activity
                          implements OnClickListener {
  private Thread t;
  private static Handler h;
  @Override
  public void onCreate(Bundle savedInstanceState){
        //.........
        h = new Handler(){
        public void handleMessage(Message msg) {
              setTitle("Value = " + cn2.rvObj.getValue());
   }};  }
```

```
@Override public void onClick(View v) {
        switch(v.getId()){
        case 101:
                cn1 = new CounterSub();
                 actNative.nativeExec();
                setTitle("Value = " + cn1.rvObj.getValue());
                break;
        case 102:
                t = new Thread(new Task());
                t.start();
break;
        case 103:   finish();  break;
}}
```

```
class Task implements Runnable {
        public void run() {
           cn2 = new CounterSub22();
            actNative.nativeExec();
            h.sendEmptyMessage(MODE_PRIVATE);
  }}}
```

~ Continued ~