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

C04_a

JNI：必要的优化设计(a)

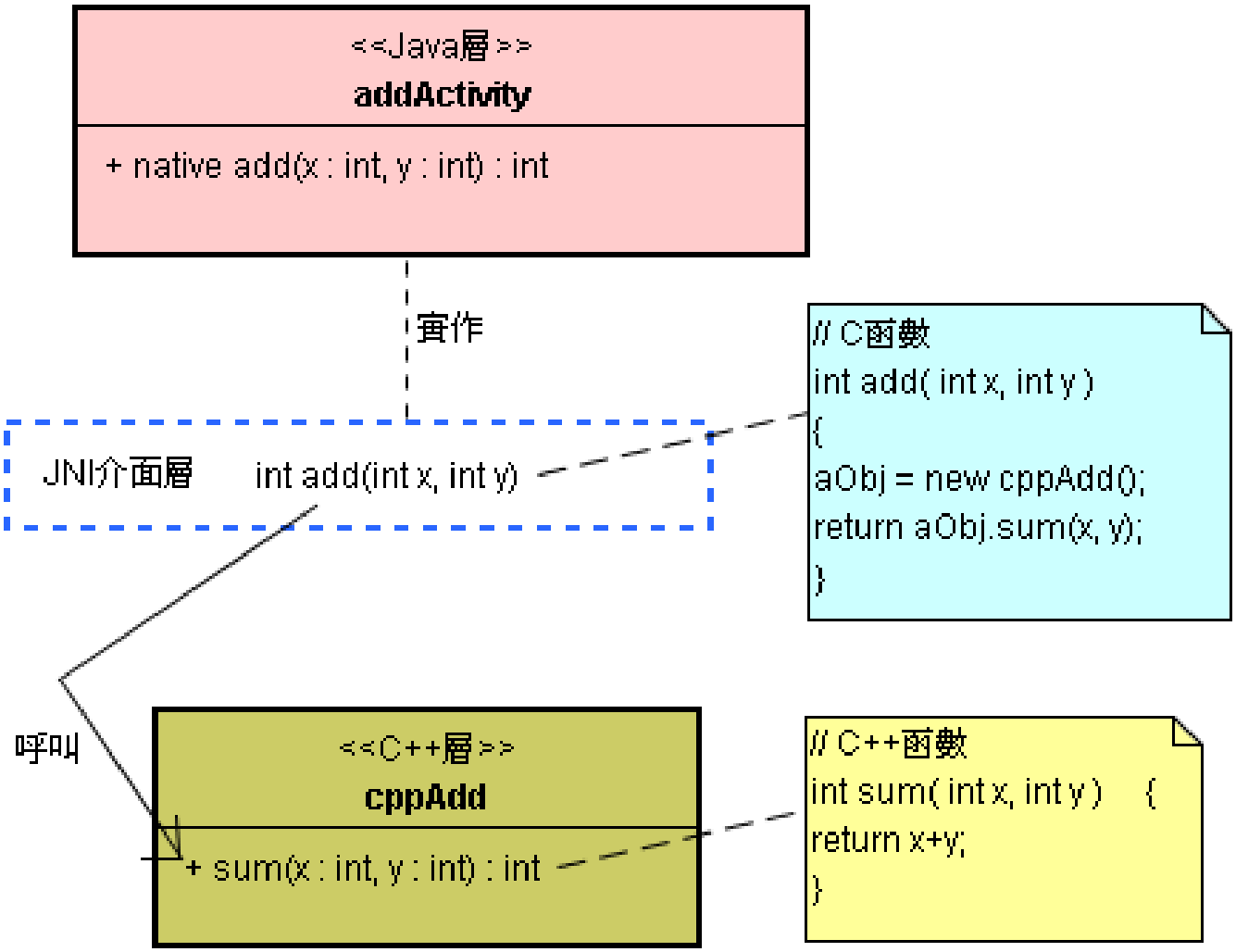
By 高煥堂

内容

1. 创建C++类的对象
2. 优化目的：维护本地函数的稳定性
3. <静态对静态，动态对动态>原则
4. Java与C++对象之间的<单向>对称关连
5. Java与C++对象之间的<双向>对称关连

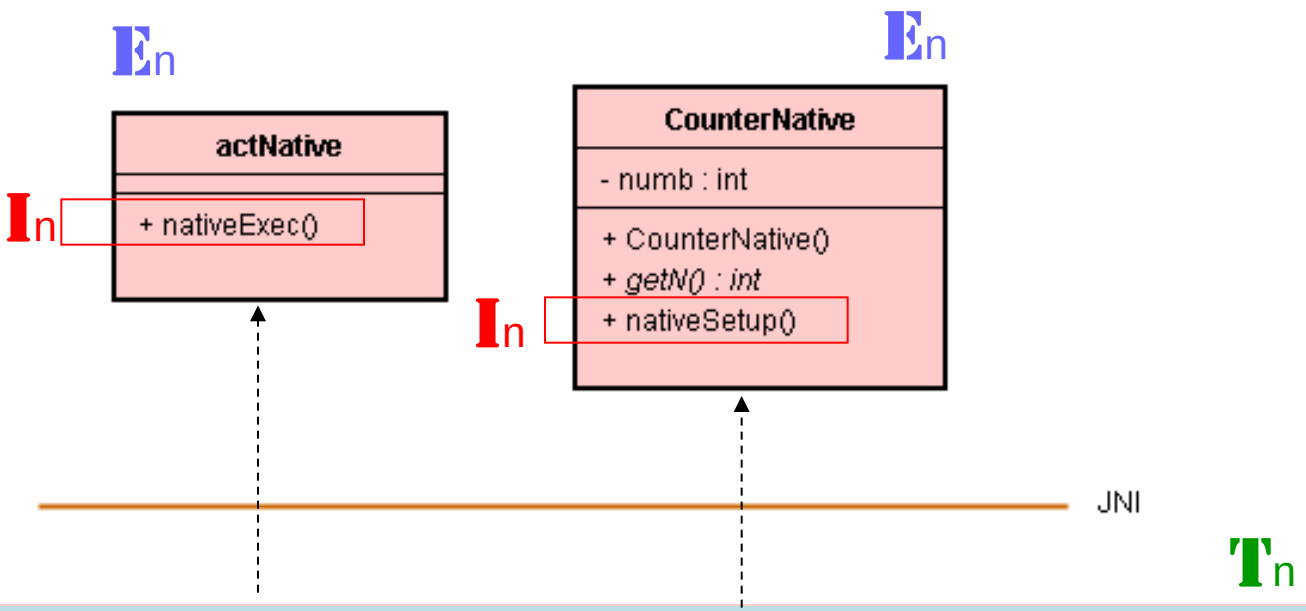
1、创建C++类的对象

- 在JNI的C模块里，不仅能创建Java层的对象，也可以创建C++类别的对象，如下图：


<<Java層>>
addActivity
+ native add(x : int, y : int) : int
實作
// C函數
int add(int x, int y)
{
aObj = new cppAdd();
return aObj.sum(x, y);
}
JNI介面層
int add(int x, int y)
呼叫
<<C++層>>
cppAdd
+ sum(x : int, y : int) : int
// C++函數
int sum(int x, int y) {
return x+y;
}

- 上图的JNI接口层是以C语言实作的本地函数。
- 在逻辑上，这些C函数仍属于Java类(即定义<In>的类)。
- 典型的架构共分为三个层级：Java层、C层和C++层；其间可以互相沟通与合作。

- C和C++代码可以摆在同一个*.so档案里。
- 多个Java类的C函数(即多个<In>的实现代码)可以摆在同一个*.so档案里。

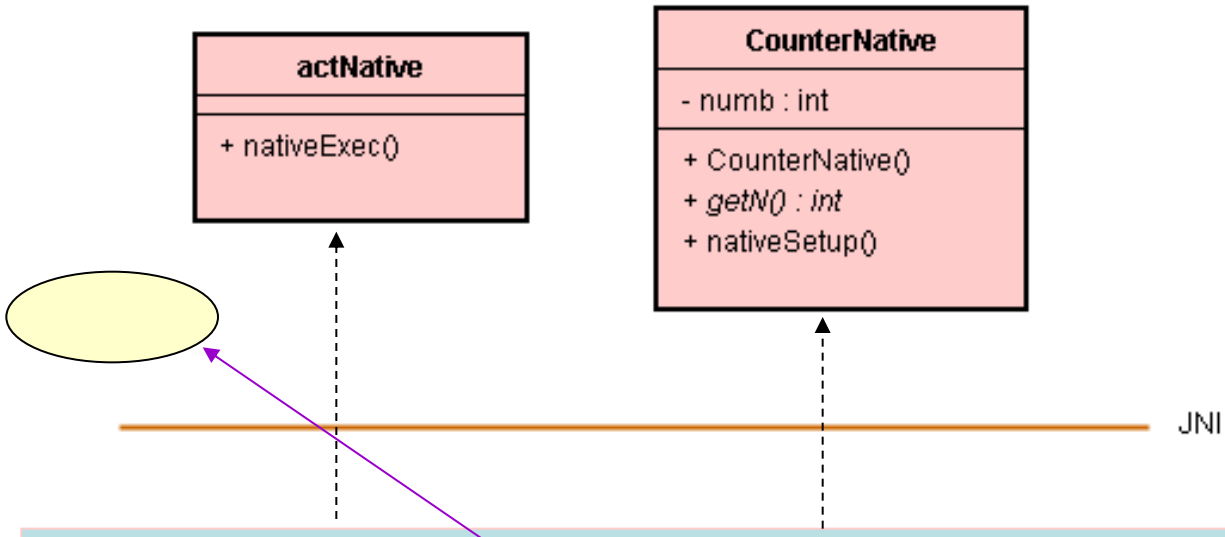



```
/* com.misoo.counter.Counter.c */
// …….
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeSetup
    (JNIEnv *env, jobject thiz) {    // …….    }

JNIEXPORT jobject JNICALL
Java_com_misoo_counter_actNative_nativeExec
      (JNIEnv *env, jclass clazz) {   // ……   }
}
```

2、优化目的：
维护本地函数的稳定性

- 不宜仰赖C层的*.so的全局变量来储存Java层或C++层的对像(指针或参考)。
- 依赖C层(全局或静态变量)来储存C++对象指针，或者储存Java层对象参考，这常常让C层模块与特定C++对象或Java对象绑在一起，产生紧密的相依性，导致系统弹性的下降。
- 本节的范例将以优越的设计化解这项困境。

```
/* com.misoo.counter.Counter.c */
// ……
jobject  m_object, m_rv_object ;

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);
        // ……
        jclass rvClazz = (*env)->FindClass(env,
                            "com/misoo/counter/ResultValue");
        jobject ref = (*env)->NewObject(env, rvClazz, constr);
        m_rv_object = (jobject)(*env)->NewGlobalRef(env, ref);
}
```



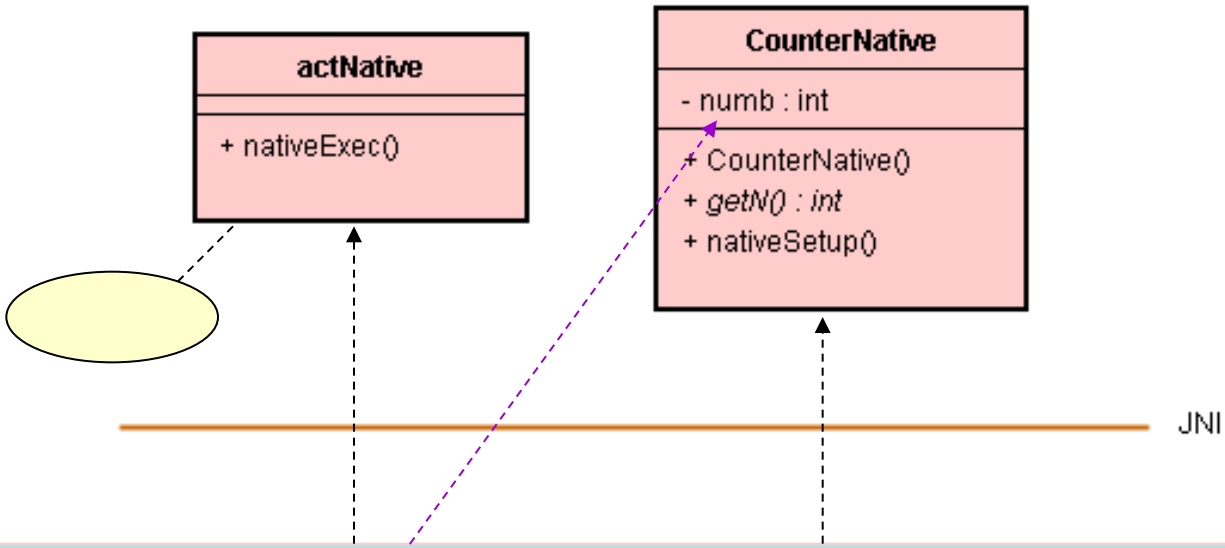

```
/* com.misoo.counter.Counter.c */
jobject   m_object, m_rv_object ;

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

议题

- 由于ResultValue对象是在run-time时期动态创建的，如果有多个对象时，该如何储存呢?
- 如果多个Java线程并行地(Concurrently)执行这个本地函数，共享了m_object和m_rv_object变量，如何确保线程之间不互相冲突呢?

不将Java或C++对象参考
储存于C层的全局变量里；
提升C函数和代码稳定度。

- C层的全局或静态(static)变量只适合储存静态的数据，例如methodID或fieldID值。




```
/* com.misoo.counter.Counter.c */
jfieldID  m_fid;

JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeSetup
    (JNIEnv *env, jobject thiz) {    // …….    }
     //…….
     m_fid = (*env)->GetFieldID(env, clazz, "numb", "I");
}
```

- 这m_fid储存的是类的属性ID，静态对静态关系，是合理的。
- Java层的每一个CounterNative类的对象来调用本地NativeSetup()时，都可利用m_fid值来取得各对象里的numb属性值(无论有多少个Java层的CounterNative对象)。

动态的对象指针又要放在哪里呢?

~ Continued ~