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

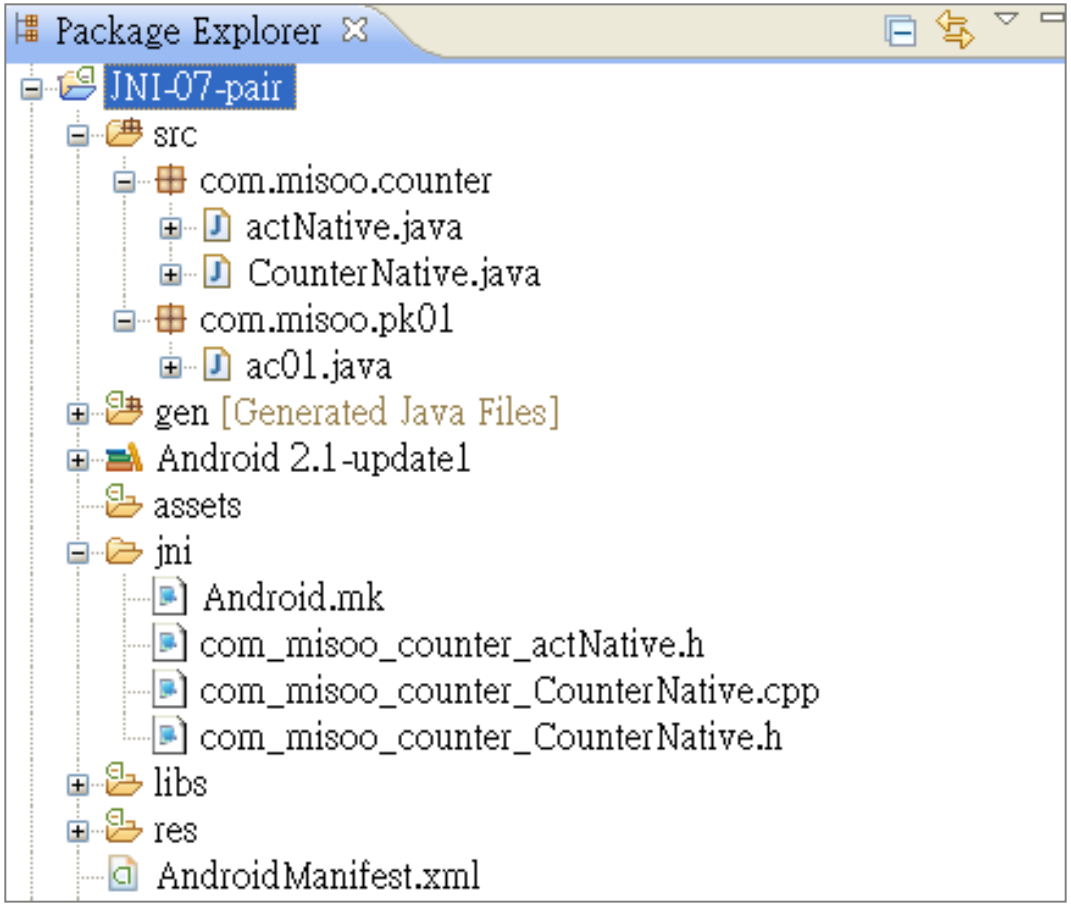
C04_c

JNI：必要的优化设计(c)

By 高煥堂

4、Java与C++对象之间的<单向>对称关连

举例说明


Package Explorer
JNI-07-pair
src
com.misoo.counter
actNative.java
CounterNative.java
com.misoo.pk01
ac01.java
gen [Generated Java Files]
Android 2.1-update1
assets
jni
Android.mk
com_misoo_counter_actNative.h
com_misoo_counter_CounterNative.cpp
com_misoo_counter_CounterNative.h
libs
res
AndroidManifest.xml

```
// CounterNative.java
// ........
public class CounterNative {
        private int mObject;
        static {
                System.loadLibrary("MyCounter7");
              }
        public CounterNative(int numb) {
                 nativeSetup( numb );
            }
        private native void nativeSetup(int n);
}
```

- 当你定义C++类别时，可以将它与JNI的C函数定义在同一个文件(*.so)里，也可定义在独立的档案里。
- 在此范例里，在JNI的C函数文件中，新增一个CCounter类。

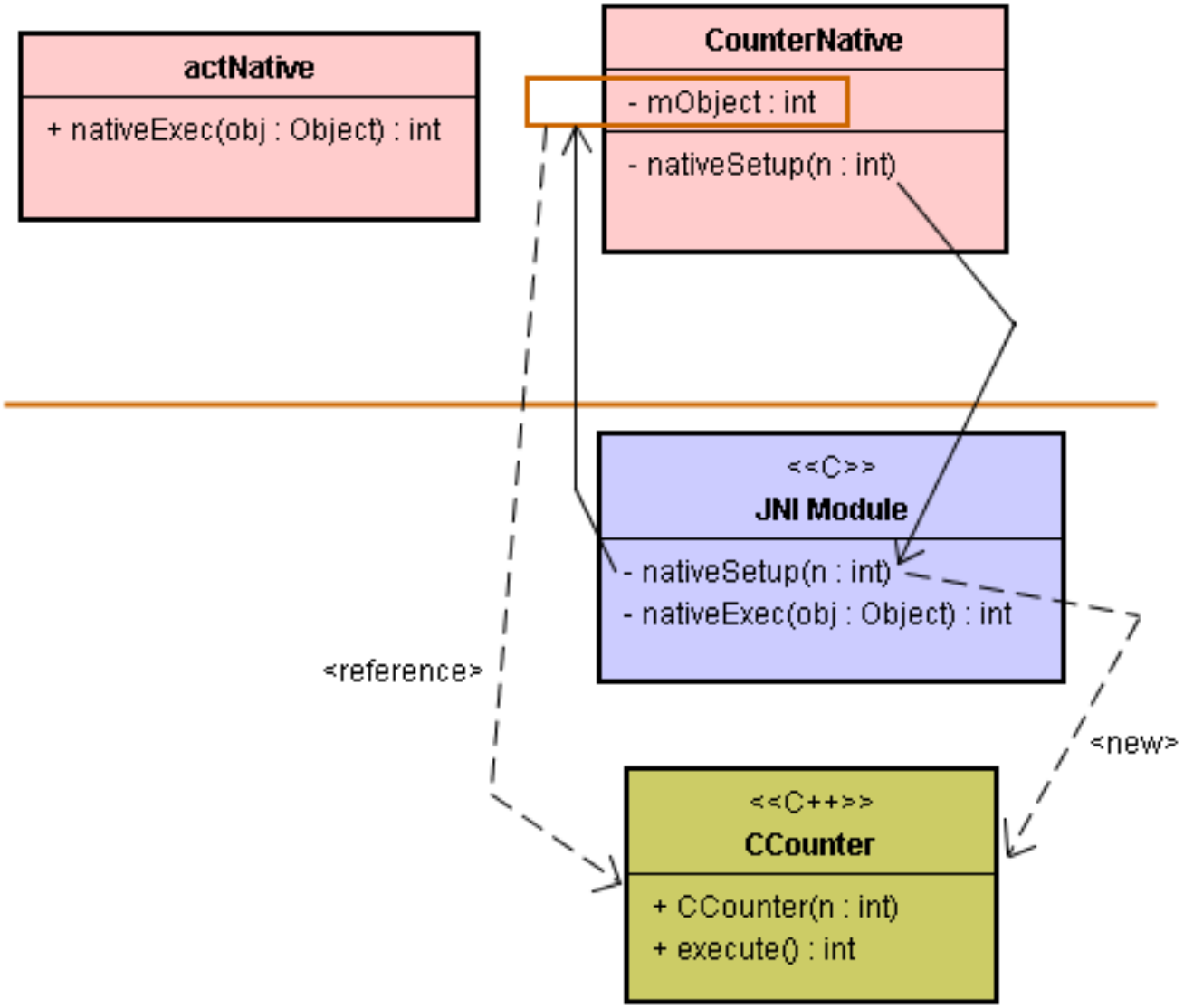
- 例如，在com_misoo_counter_CounterNative.cpp里除了实作本地C函数之外，还定义了一个C++的CCounter类。

```
/* com.misoo.counter.CounterNative.cpp */
#include "com_misoo_counter_actNative.h"
#include "com_misoo_counter_CounterNative.h  ”

class CCounter{
      int n;
   public:
      CCounter(int v) {  n = v;  }
      int execute() {
                int i, sum = 0;
                for(i=0; i<=n; i++)  sum+=i;
                return sum;
  }};
```

```
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeSetup
  (JNIEnv *env, jobject thiz, jint n) {
        CCounter *obj = new CCounter(n);
        jclass clazz = (jclass)env->GetObjectClass(thiz);
        jfieldID fid = (jfieldID)env->GetFieldID(clazz, "mObject", "I");
        env->SetIntField(thiz, fid, (jint)obj);
}

JNIEXPORT jint JNICALL
Java_com_misoo_counter_actNative_nativeExec
  (JNIEnv *env, jclass clazz, jobject obj) {
        jclass objClazz = (jclass)env->GetObjectClass(obj);
        jfieldID fid = env->GetFieldID(objClazz, "mObject", "I");
        jlong p = (jlong)env->GetObjectField(obj, fid);
        CCounter *co = (CCounter*)p;
        return (jint)co->execute();
}
```


actNative
+ nativeExec(obj : Object) : int
CounterNative
- mObject : int
- nativeSetup(n : int)
<<C>>
JNI Module
- nativeSetup(n : int)
- nativeExec(obj : Object) : int
<reference>
<new>
<<C++>>
CCounter
+ CCounter(n : int)
+ execute() : int

- 上述nativeSetup()函数里的指令：

  CCounter *obj = new CCounter(n);

- 创建一个C++层的CCounter对象，并且把n值存入其中。

- 随后，指令：

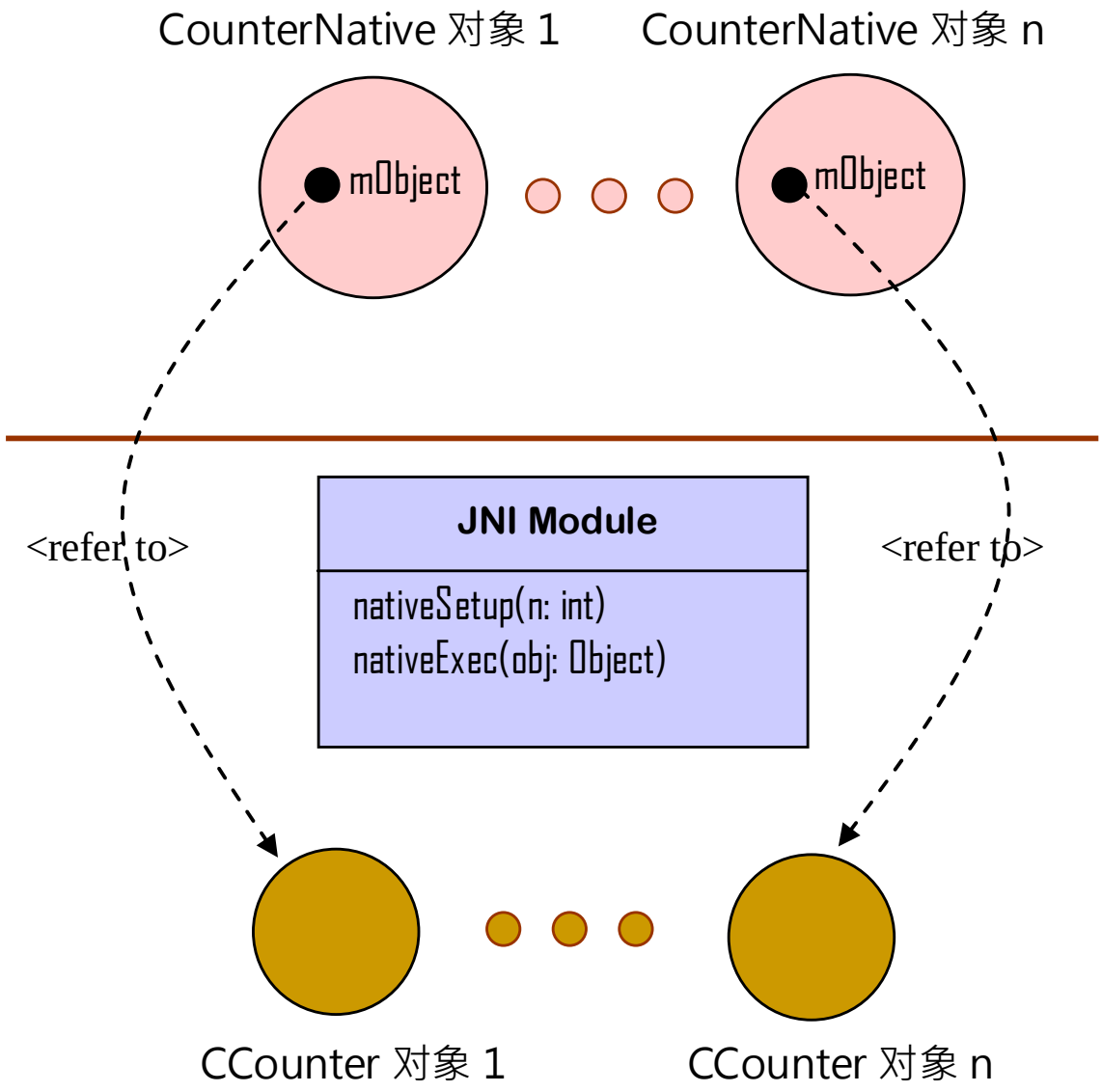
```
jclass clazz = (jclass)env->GetObjectClass(thiz);
jfieldID fid =
    (jfieldID)env->GetFieldID(clazz, "mObject", "I");
```

- 取得该CCounter对象的mObject属性ID。

- 接着，指令：

  env->SetIntField(thiz, fid, (jint)obj);

- 就将CCounter对象的指针值储存于CounterNative对象的mObject属性里，如此建立了CounterNative对象与CCounter对象之连结。


CounterNative 对象 1
CounterNative 对象 n
mObject
mObject
JNI Module
nativeSetup(n: int)
nativeExec(obj: Object)
<refer to>
<refer to>
CCounter 对象 1
CCounter 对象 n

- C模块创建CCounter对象之后，立即将CCounter对象指针储存于CounterNative的mObject属性里。

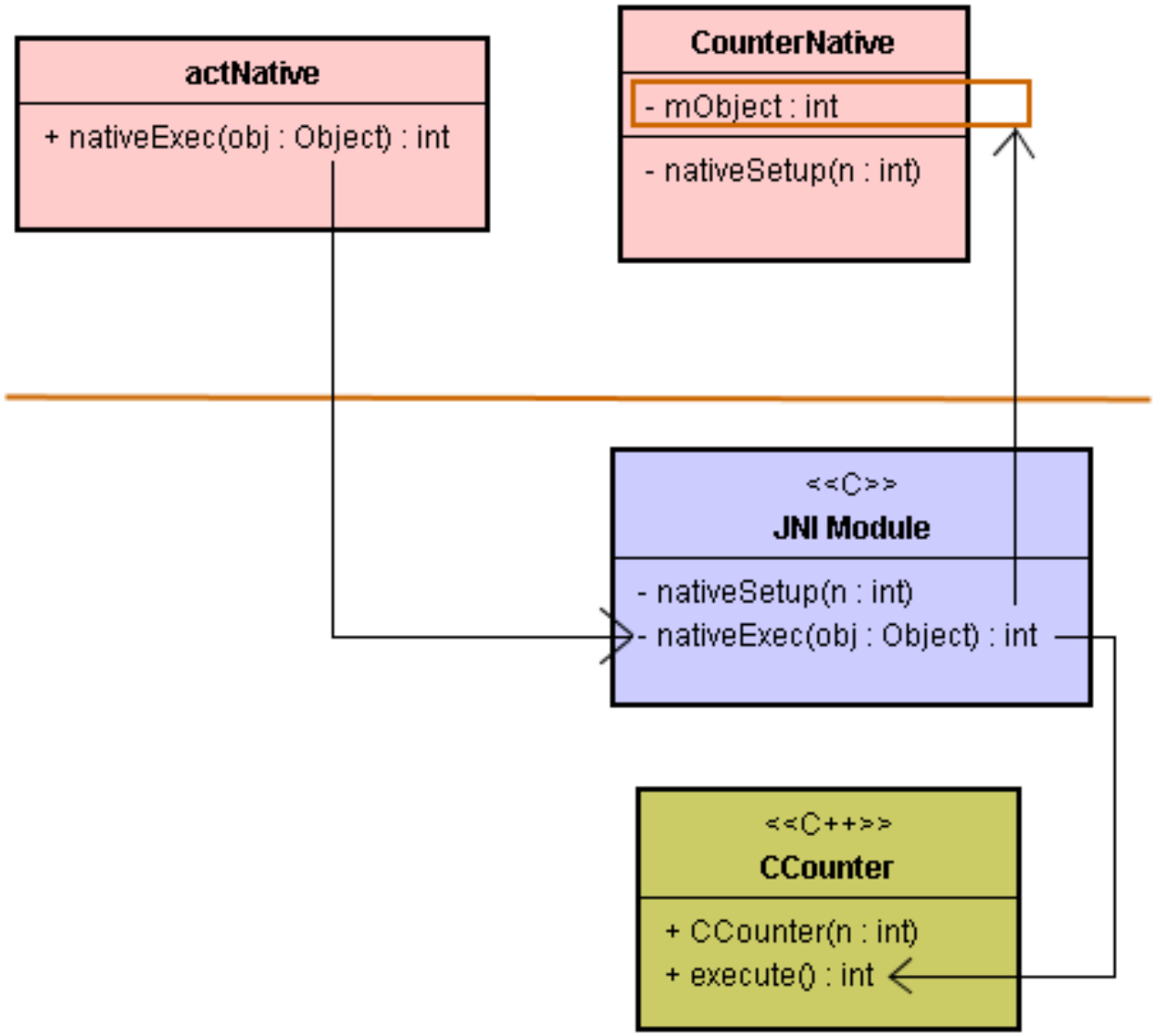
静态对静态，
动态对动态

- C模块来创建C++对象，然后让Java对象与C++对象之间产生成双成对的连结关系。

- C模块本身并不储存Java或C++对象的指针或参考值。而是仅负责创建C++对象，并建立Java与C++的对象间的连结关系。
- 如此，C模块能替众多Java对象服务，而不再与特定的Java对象绑在一起了。

- 一旦解开C模块与C++对象(或Java对象)之间的相依性，C模块就能具有通用性。
- 例如，C层nativeSetup()函数，能为Java层的每一个对象建立其相对映的C++对象。

- 由于C层的nativeSetup()已经变成为通用型的函数了，每次调用它时，只要将特定的CounterNative对象传递给它，就能顺利找到其相对映的CCounter对象了。如下图：


actNative
+ nativeExec(obj : Object) : int
CounterNative
- mObject : int
- nativeSetup(n : int)
<<C>>
JNI Module
- nativeSetup(n : int)
- nativeExec(obj : Object) : int
<<C++>>
CCounter
+ CCounter(n : int)
+ execute() : int

```
// actNative.java
// ……
public class actNative {
        public static native int nativeExec(Object obj);
}
```

- 这nativeExec()先取得CounterNative对象里的mObject属性值，相当于取得CCounter对象的指针了，就能调用CCounter对象的execute()函数了。

- 由于C模块里并没有储存CounterNative对象的指针，所以Java必须将CounterNative对象的参考值传递给JNI层的nativeExec()本地函数，如下指令：

- 編修ac01.java类别：

```
// ac01.java
// ……..
public class ac01 extends Activity implements OnClickListener {
  private CounterNative cn1, cn2;

  @Override public void onCreate(Bundle savedInstanceState){
    //……..
    cn1 = new CounterNative(10);
    cn2 = new CounterNative(12);
  }
```

- ac01.java类：

```
@Override public void onClick(View v) {
        int sum;
        switch(v.getId()){
        case 101:  sum = actNative.nativeExec(cn1);
                   setTitle("Sum = " + sum);
                    break;
        case 102:  sum = actNative.nativeExec(cn2);
                   setTitle("Sum = " + sum);
                    break;
        case 103:  finish();
                    break;
}}}
```

- 指令：

```
actNative.nativeExec( cn1 );
```

- 此时，ac01将CounterNative类别的第1个对象传递给JNI模块的nativeExec()函数，找到相对映的CCounter对象，然后调用它的execute()函数。

```
JNIEXPORT jint JNICALL
Java_com_misoo_counter_actNative_nativeExec
  (JNIEnv *env, jclass clazz, jobject obj) {
        jclass objClazz = (jclass)env->GetObjectClass(obj);
        jfieldID fid = env->GetFieldID(objClazz, "mObject", "I");
        jlong p = (jlong)env->GetObjectField(obj, fid);
        CCounter *co = (CCounter*)p;
        return (jint)co->execute();
}
```

- 这obj参考到CounterNative对象。

- 当其执行到指令：

```
jfieldID fid = env->GetFieldID(objClazz, "mObject", "I");
jlong p = (jlong)env->GetObjectField(obj, fid);
```

- 就从CounterNative对象里取得mObject属性值，并存入p变量里。此p值正是C++层CCounter对象的参考，所以可透过p调用CCounter对象的execute()函数。

~ Continued ~