<http://www.cnblogs.com/moon1992/p/5258245.html>

**阅读目录**

* [一. 什么是 native Method](http://www.cnblogs.com/moon1992/p/5258245.html#_label0)
* [二. 为什么要使用 native Method](http://www.cnblogs.com/moon1992/p/5258245.html#_label1)
* [三. JVM怎样使 native Method 跑起来](http://www.cnblogs.com/moon1992/p/5258245.html#_label2)
* [四. Native关键字示例：Java调用C语言本地库](http://www.cnblogs.com/moon1992/p/5258245.html#_label3)
* [五. 使用批处理文件编译运行程序](http://www.cnblogs.com/moon1992/p/5258245.html#_label4)

[回到顶部](http://www.cnblogs.com/moon1992/p/5258245.html#_labelTop)

### 一. 什么是 native Method

　　简单地讲，一个 native Method 就是一个java调用非java代码的接口。一个 native Method 是这样一个java的方法：该方法的实现由非java语言实现，比如C。这个特征并非java所特有，很多其它的编程语言都有这一机制，比如在C++中，你可以用extern "C"告知C++编译器去调用一个C的函数。

　　"A native method is a Java method whose implementation is provided by non-java code."

　　在定义一个 native Method 时，并不提供实现体（有些像定义一个java interface），因为其实现体是由非java语言在外面实现的(通常是C或C++等)。例如：

[复制代码](javascript:void(0);)

1 package java.lang;

2

3 public class Object {

4 ......

5 public final native Class<?> getClass();

6 public native int hashCode();

7 protected native Object clone() throws CloneNotSupportedException;

8 public final native void notify();

9 public final native void notifyAll();

10 public final native void wait(long timeout) throws InterruptedException;

11 ......

12 }

[复制代码](javascript:void(0);)

　　标识符 native 可以与所有其它的java标识符连用，但是abstract除外。这是合理的，因为 native 暗示这些方法是有实现体的，只不过这些实现体是非java的，但是abstract却显然的指明这些方法无实现体。 native 与其它java标识符连用时，其意义同非 native Method 并无差别。

　　一个 native Method 方法可以返回任何java类型，包括非基本类型，而且同样可以进行异常控制。这些方法的实现体可以自制一个异常并且将其抛出，这一点与java的方法非常相似。

　　native Method 的存在并不会对其他类调用这些本地方法产生任何影响，实际上调用这些方法的其他类甚至不知道它所调用的是一个本地方法。JVM将控制调用本地方法的所有细节。

　　如果一个含有本地方法的类被继承，子类会继承这个本地方法并且可以用java语言重写这个方法（这个似乎看起来有些奇怪），同样的如果一个本地方法被 final标识，它被继承后不能被重写。

　　本地方法非常有用，因为它有效地扩充了jvm，事实上，我们所写的java代码已经用到了本地方法，在sun的java的并发（多线程）的机制实现中，许多与操作系统的接触点都用到了本地方法，这使得java程序能够超越java运行时的界限。有了本地方法，java程序可以做任何应用层次的任务。

[回到顶部](http://www.cnblogs.com/moon1992/p/5258245.html#_labelTop)

### 二. 为什么要使用 native Method

　　java使用起来非常方便，然而有些层次的任务用java实现起来不容易，或者我们对程序的效率很在意时，问题就来了。

1. 与java环境外交互：  
　　有时java应用需要与java外面的环境交互。这是本地方法存在的主要原因，你可以想想java需要与一些底层系统如操作系统或某些硬件交换信息时的情况。本地方法正是这样一种交流机制：它为我们提供了一个非常简洁的接口，而且我们无需去了解java应用之外的繁琐的细节。

2. 与操作系统交互：  
　　JVM支持着java语言本身和运行时库，它是java程序赖以生存的平台，它由一个解释器（解释字节码）和一些连接到本地代码的库组成。然而不管怎样，它毕竟不是一个完整的系统，它经常依赖于一些底层（underneath在下面的）系统的支持。这些底层系统常常是强大的操作系统。通过使用本地方法，我们得以用java实现了jre的与底层系统的交互，甚至JVM的一些部分就是用C写的，还有，如果我们要使用一些java语言本身没有提供封装的操作系统的特性时，我们也需要使用本地方法。

3. Sun's Java  
　　Sun的解释器是用C实现的，这使得它能像一些普通的C一样与外部交互。jre大部分是用java实现的，它也通过一些本地方法与外界交互。例如：类java.lang.Thread 的 setPriority()方法是用java实现的，但是它实现调用的是该类里的本地方法setPriority0()。这个本地方法是用C实现的，并被植入JVM内部，在Windows 95的平台上，这个本地方法最终将调用Win32 SetPriority() API。这是一个本地方法的具体实现由JVM直接提供，更多的情况是本地方法由外部的动态链接库（external dynamic link library）提供，然后被JVM调用。

[回到顶部](http://www.cnblogs.com/moon1992/p/5258245.html#_labelTop)

### 三. JVM怎样使 native Method 跑起来

　　我们知道，当一个类第一次被使用到时，这个类的字节码会被加载到内存，并且只会回载一次。在这个被加载的字节码的入口维持着一个该类所有方法描述符的list，这些方法描述符包含这样一些信息：方法代码存于何处，它有哪些参数，方法的描述符（public之类）等等。

　　如果一个方法描述符内有 native, 这个描述符块将有一个指向该方法的实现的指针。这些实现在一些DLL文件内，但是它们会被操作系统加载到java程序的地址空间。当一个带有本地方法的类被加载时，其相关的DLL并未被加载，因此指向方法实现的指针并不会被设置。当本地方法被调用之前，这些DLL才会被加载，这是通过调用System.loadLibrary()实现的。

　　最后需要提示的是，使用本地方法是有开销的，它丧失了Java的很多好处。如果别无选择，我们可以选择使用本地方法。

　　可以将native方法比作Java程序同Ｃ程序的接口，其实现步骤为：

1. 在Java中声明 native方法，然后编译；
2. 用javah产生一个.h文件；
3. 写一个.cpp文件实现 native 导出方法，其中需要包含第二步产生的.h文件（注意其中又包含了JDK带的jni.h文件）；
4. 将第三步的.cpp文件编译成动态链接库文件；
5. 在Java中用 System.loadLibrary()方法加载第四步产生的动态链接库文件，这个 native方法就可以在Java中被访问了。(实际上已经在第1步的Java代码上提前写进去了)

[回到顶部](http://www.cnblogs.com/moon1992/p/5258245.html#_labelTop)

### 四. Native关键字示例：Java调用C语言本地库

　　代码功能：Java与C语言混合编程，实现两个数字相加。即在Java的类中声明 native 方法，而具体实现由C语言完成。

#### 4.1  Java程序编写：

[复制代码](javascript:void(0);)

1 //fileName: TestAdd

2 public class TestAdd

3 {

4 public static void main(String[] args)

5 {

6 System.loadLibrary("NativeAdd");//加载由C编译器生成的DLL文件。

7

8 NativeAdd na = new NativeAdd();

9 System.out.println("3 + 4 = " + na.add(3, 4));

10 }

11 }

12

13 class NativeAdd

14 {

15 public native int add(int x, int y);

16 }

[复制代码](javascript:void(0);)

　　上面这段代码中 System.loadLibrary("NativeAdd"); 加载了动态类库，在Windows下加载的就是NativeAdd.dll，在Linux中加载的就是libNativeAdd.so，本文使用的Windows，所以后面使用NativeAdd.dll来表示NativeAdd动态链接库。注意不可以在代码中写上扩展名.dll或者.so，还要保证NativeAdd.dll在path路径中, 或者在与.class文件在同一个文件夹中，否则无法加载动态链接库。这个NativeAdd.dll是我们后面需要编译出来的东西.

　　使用命令行 javac TestAdd.java 可以看到生成文件TestAdd.class文件和NativeAdd.class文件，不过现在还不能运行TestAdd.class，因为我们还没有生成NativeAdd.dll，下面介绍如何生成NativeAdd.dll文件。

#### 4.2  使用javah生成头文件

　　使用命令行 javah NativeAdd 生成NativeAdd.h，注意此处不能写成 javah TestAdd ，因为我们使用 native 关键字修饰的方法在NativeAdd类中，而不是TestAdd类中。也就是说：native 关键字修饰的方法在那个类中，就使用javah命令生成对应的头文件，然后使用C语言实现这个方法。

　　执行完上面的命令之后，可以自动生成一个新文件：NativeAdd.h，代码如下：

[复制代码](javascript:void(0);)

1 /\* DO NOT EDIT THIS FILE - it is machine generated \*/

2 #include <jni.h>

3 /\* Header for class Add \*/

4

5 #ifndef \_Included\_Add

6 #define \_Included\_Add

7 #ifdef \_\_cplusplus

8 extern "C" {

9 #endif

10 /\*

11 \* Class: Add

12 \* Method: add

13 \* Signature: (II)I

14 \*/

15 JNIEXPORT jint JNICALL Java\_Add\_add

16 (JNIEnv \*, jobject, jint, jint);

17

18 #ifdef \_\_cplusplus

19 }

20 #endif

21 #endif

[复制代码](javascript:void(0);)

　　我们可以看到其中有一个函数声明 JNIEXPORT jint JNICALL Java\_Add\_add (JNIEnv \*, jobject, jint, jint); ，这个头文件我们不用管也不用做任何修改，他是有JNI(Java Native Interface)自动生成的，我们要做的就是编写这个函数的函数体。

#### 4.3  实现头文件中的函数

　　这里以C语言为例，介绍如何实现上面的函数，并生成DLL文件。

4.3.1 创建DLL工程

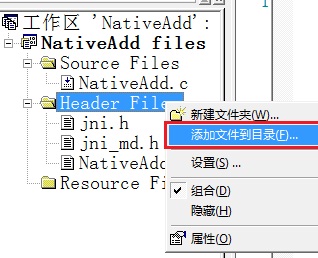
打开VC++6.0，执行：文件-->新建-->工程-->Win32 Dynamic-Link Library，输入工程名NativeAdd，选择工程路径到适当位置-->创建一个空的DLL工程。

4.3.2 创建.c文件

执行：文件-->新建-->文件-->C/C++Source File，输入文件名为NativeAdd.c，注意这里最好加上扩展名为.c，否则默认为.cpp文件，我们是使用C语言实现头文件中声明的函数，而非C++，为避免不必要的问题，最后还是加上扩展名.c，以免编译出错。

4.3.3 添加头文件

这里需要添加的头文件有三个，分别是：NativeAdd.h, jni.h, jni\_md.h，其中NativeAdd.h是我们刚才使用javah命令生成的，jni.h在 \Java\jdk1.8.0\_05\include ，jni.md\_h在 \Java\jdk1.8.0\_05\include\win32 。将这两个文件拷贝到工程文件夹中。右击Header File-->添加文件到目录，选择我们刚刚拷贝进来的三个头文件NativeAdd.h, jni.h, jni\_md.h。



4.3.4 编写C程序

代码如下：即将头文件的函数声明拷贝到c文件中，添加形参，编写函数体将函数实现即可。然后保存编译运行，即可在Debug文件夹下即生成NativeAdd.dll文件

[复制代码](javascript:void(0);)

1 #include <stdio.h>

2 #include "NativeAdd.h"

3

4 JNIEXPORT jint JNICALL Java\_NativeAdd\_add

5 (JNIEnv \* env, jobject obj, jint x, jint y)

6 {

7 return x + y;

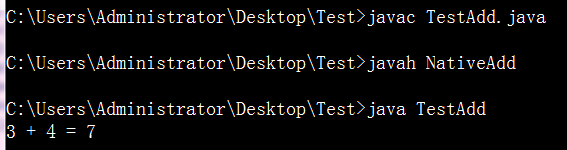
8 }

[复制代码](javascript:void(0);)

#### 4.4  运行Java程序，调用DLL文件。

　　将上一步骤中的NativeAdd.dll文件拷贝到TestAdd.class所在的文件中。注意dll的文件名要与Java代码中 System.loadLibrary("NativeAdd"); 保持一致。如果不一致可以自己修改文件名。

　　执行命令号 java TestAdd 即可看到运行输出结果为7，即:3 + 4 = 7。



[回到顶部](http://www.cnblogs.com/moon1992/p/5258245.html#_labelTop)

### 五. 使用批处理文件编译运行程序

[批处理](http://baike.baidu.com/link?url=pXallf65jPeTUg4z31NpVgFHBnSMY5PD4KuYkSGQL-Zoz7Vv8wm0OBYMWnbTCzflsxSQtX3bRgWATb2u2rVb1a)(Batch)，也称为批处理脚本。批处理就是对某对象进行批量的处理，通常被认为是一种简化的脚本语言，它应用于DOS和Windows系统中。批处理文件的扩展名为.bat 。批处理文件的编写非常简单，只是相当于把DOS命令行的指令一条条的写到一个文本文件中，然后修改文本文件的扩展名为.bat。双击运行会自动用DOS窗口打开并依次执行批处理文件中的指令。

[复制代码](javascript:void(0);)

1 echo 清理以前生成的文件

2 del \*.class

3 del \*.dll

4 del \*.exp

5 del \*.lib

6 del \*.obj

7 del \*.h

8

9 pause

10

11 echo 编译java文件并生成.h文件

12 javac TestAdd.java

13 javah NativeAdd

14

15 pause

16

17 echo 生成.dll文件

18 cl -I "%JAVA\_HOME%\include" -I "%JAVA\_HOME%\include\win32" -LD NativeAdd.c -FeNativeAdd.dll

19

20 echo 运行

21 java TestAdd

22

23 pause

[复制代码](javascript:void(0);)

　　文件名build&run.bat，用到了几个批处理命令，这里解释一下这个批处理文件的意思。

* echo  命令：将echo后面跟的内容输出到DOS窗口中，起到提示作用，相当于C中的printf。
* del   命令：删除文件，\*.class表示删除所有的class文件，在编译运行前我们先删除上次编译生成的文件，以免对本次编译运行结果的产生干扰。
* pause 命令：批处理暂停，在屏幕输出"按任意键继续..."，按任意键可以继续执行批处理文件。
* cl    命令：VC编译器，在DOS下输入cl会打印相关信息。-I 指定头文件的搜索路径，注意路径要用""，这里指定的分别是jni.h和jni\_md.h所在的路径。-LD 表示创建一个动态连接库。-Fe 设置最终可执行文件的存放路径及（或）文件名。

　　有个这个批处理文件，我们就不用使用VC创建工程的方法去生成dll文件了，而是直接使用批处理文件中的cl命令。只需要将build&run.bat，TestAdd.java，NativeAdd.c这三个文件放到同一个文件夹中，双击运行bat文件，批处理文件会完成上面的命令。

**参考：** [Java的native关键字](http://blog.csdn.net/jiakw_1981/article/details/3073613)

http://www.cnblogs.com/moon1992/