[java native方法及JNI实例](http://blog.csdn.net/xw13106209/article/details/6989415)

**1.参考文献：**

<http://blog.csdn.net/youjianbo_han_87/article/details/2586375>

<http://blog.csdn.net/yangjiali014/article/details/1633017>

<http://blog.chinaunix.net/space.php?uid=7437948&do=blog&id=2054823>

<http://www.iteye.com/topic/72543>

<http://www.enet.com.cn/article/2007/1029/A20071029886398.shtml>

<http://blog.csdn.net/heqingrong623/article/details/3906350>

参考1:[用JNI调用C或C++动态联接库原来如此简单](http://blog.chinaunix.net/space.php?uid=13756968&do=blog&id=2873440)

参考2：[JNI技术实践小结](http://www.iteye.com/topic/304594)

参考3：[jni简单实例](http://www.cnblogs.com/sunnychuh/archive/2011/07/17/2108852.html)

## 2.概述

今天在看java多线程编程的时候，发现Thread这个类中有多个native方法，以前从来没有见过这种方法，因此对于比较好奇，查阅了一些资料，现在整理一下，以作备忘。

### 2.1.native关键字用法

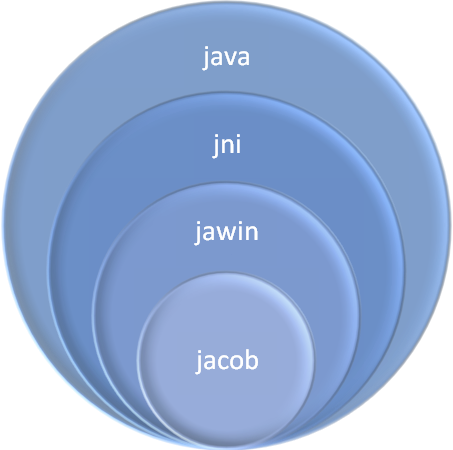
native是与C++联合开发的时候用的！**使用native关键字说明这个方法是原生函数，也就是这个方法是用C/C++语言实现的，并且被编译成了DLL，由java去调用**。 这些函数的实现体在DLL中，JDK的源代码中并不包含，你应该是看不到的。对于不同的平台它们也是不同的。这也是java的底层机制，实际上java就是在不同的平台上调用不同的native方法实现对操作系统的访问的。总而言之：

1. native 是用做java 和其他语言（如c++）进行协作时使用的，也就是native 后的函数的实现不是用java写的。
2. 既然都不是java，那就别管它的源代码了，我们只需要知道这个方法已经被实现即可。
3. native的意思就是通知操作系统， 这个函数你必须给我实现，因为我要使用。 所以native关键字的函数都是操作系统实现的， java只能调用。
4. java是跨平台的语言，既然是跨了平台，所付出的代价就是牺牲一些对底层的控制，而java要实现对底层的控制，就需要一些其他语言的帮助，这个就是native的作用了

### 2.2JNI简介

native方法是通过java中的JNI实现的。JNI是Java Native Interface的 缩写。从Java 1.1开始，Java Native Interface (JNI)标准成为java平台的一部分，它允许Java代码和其他语言写的代码进行交互。***JNI一开始是为了本地已编译语言，尤其是C和C++而设计 的，但是它并不妨碍你使用其他语言，只要调用约定受支持就可以了。***使用java与本地已编译的代码交互，通常会丧失平台可移植性。但是，有些情况下这样做是可以接受的，甚至是必须的，比如，使用一些旧的库，与硬件、操作系统进行交互，或者为了提高程序的性能。JNI标准至少保证本地代码能工作在任何Java 虚拟机实现下。

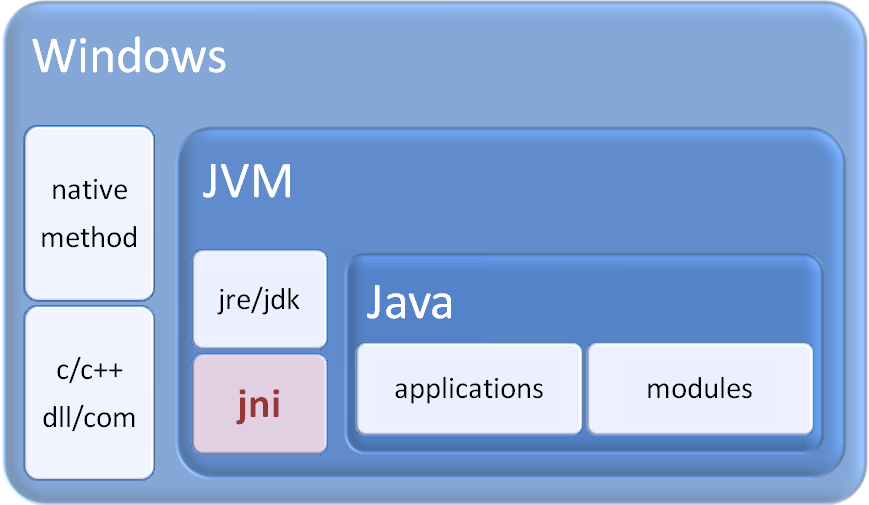
**目前java与dll交互的技术主要有3种：jni，jawin和jacob。**Jni（Java Native Interface）是sun提供的java与系统中的原生方法交互的技术（在windows\linux系统中，实现java与native method互调）。目前只能由c/c++实现。后两个都是sourceforge上的开源项目，同时也都是基于jni技术的windows系统上的一个应用库。Jacob（Java-Com Bridge）提供了java程序调用microsoft的com对象中的方法的能力。而除了com对象外，jawin（Java/Win32 integration project）还可以win32-dll动态链接库中的方法。**就功能而言：jni >> jawin>jacob，其大致的结构如下图：**



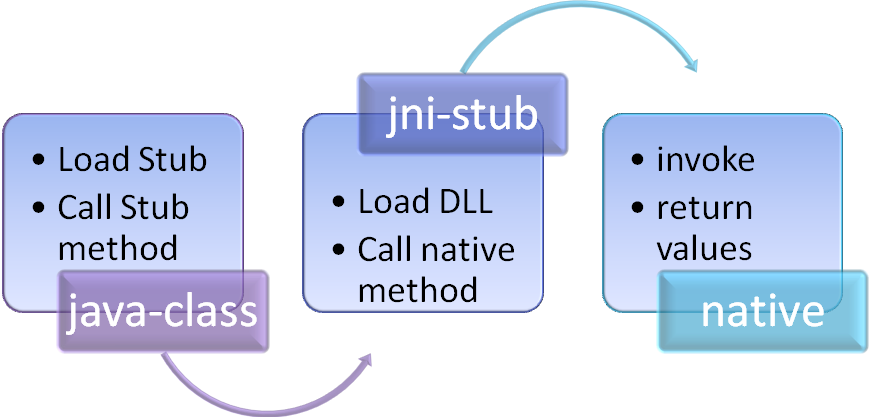
**就易用性而言，正好相反：jacob>jawin>>jni。**

Jvm封装了各种操作系统实际的差异性的同时，提供了jni技术，使得开发者可以通过java程序（代码）调用到操作系统相关的技术实现的库函数，从而与其他技术和系统交互，使用其他技术实现的系统的功能；同时其他技术和系统也可以通过jni提供的相应原生接口开调用java应用系统内部实现的功能。

在windows系统上，一般可执行的应用程序都是基于native的PE结构，windows上的jvm也是基于native结构实现的。Java应用体系都是构建于jvm之上。



Jni对于应用本身来说，可以看做一个代理模式。对于开发者来说，需要使用c/c++来实现一个代理程序（jni程序）来实际操作目标原生函数，java程序中则是jvm通过加载并调用此jni程序来间接地调用目标原生函数。



### 2.3JN的书写步骤

1. 编写带有native声明的方法的java类，生成.java文件
2. 使用javac命令编译所编写的java类，生成.class文件
3. 使用javah -jni java**类名**生成扩展名为h的头文件，也即生成.h文件
4. 使用C/C++（或者其他编程想语言）实现本地方法，创建.h文件的实现，也就是创建.cpp文件实现.h文件中的方法
5. 将C/C++编写的文件生成动态连接库，生成dll文件

## 3.JNI实例

下列是所有操作都是在目录：**D:\JNI**下进行的，这样做的好处是便于控制。还有另外一个要求是我们的java类不含包名，当前我只测试成功不含包名的类型。

### 3.1.编写带有native声明的方法的java类：HelloWorld.java

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. **public** **class** HelloWorld {
2. **public** **native** **void** displayHelloWorld();// java native方法申明
4. **static** {
5. System.loadLibrary("HelloWorldImpl");// 装入动态链接库，"HelloWorldImpl"是要装入的动态链接库名称。
6. }
8. **public** **static** **void** main(String[] args) {
9. // TODO Auto-generated method stub
10. HelloWorld helloWorld = **new** HelloWorld();
11. helloWorld.displayHelloWorld();
12. }
13. }

### 3.2.使用javac命令编译所编写的java类

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. d:\JNI>javac HelloWorld.java

执行完上述命令以后生成**D:\JNI\HelloWorld.class**文件

### 3.3.使用javah -jni java类名生成扩展名为h的头文件

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. d:\JNI>javah -jni HelloWorld

执行完上述命令以后生成**D:\JNI\HelloWorld.h**文件，该文件内容如下：

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. /\* DO NOT EDIT THIS FILE - it is machine generated \*/
2. #include <jni.h>
3. /\* Header for class HelloWorld \*/
5. #ifndef \_Included\_HelloWorld
6. #define \_Included\_HelloWorld
7. #ifdef \_\_cplusplus
8. extern "C" {
9. #endif
10. /\*
11. \* Class:     HelloWorld
12. \* Method:    displayHelloWorld
13. \* Signature: ()V
14. \*/
15. JNIEXPORT **void** JNICALL Java\_HelloWorld\_displayHelloWorld
16. (JNIEnv \*, jobject);
18. #ifdef \_\_cplusplus
19. }
20. #endif
21. #endif

这里我们可以这样理解：这个h文件相当于我们在java里面的接口，这里声明了一个 Java\_HelloWorld\_displayHelloWorld (JNIEnv \*, jobject);方法，然后在我们的本地方法里面实现这个方法，也就是说我们在编写C/C++程序的时候所使用的方法名必须和这里的一致

### 3.4.使用C/C++实现本地方法

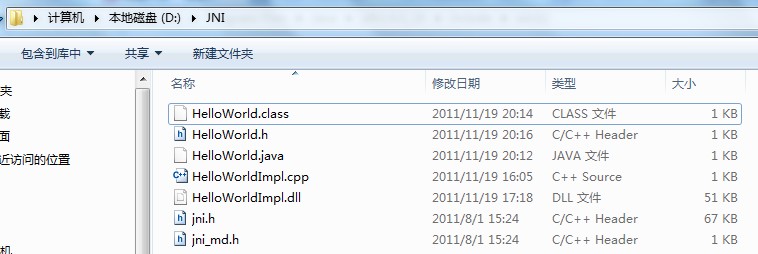
创建HelloWorldImpl.cpp，代码如下所示：

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. #include "HelloWorld.h"
2. #include <stdio.h>
3. #include <jni.h>
4. /\*
5. \* Class:     HelloWorld
6. \* Method:    displayHelloWorld
7. \* Signature: ()V
8. \*/
9. JNIEXPORT **void** JNICALL Java\_HelloWorld\_displayHelloWorld
10. (JNIEnv \*, jobject)
11. {
12. printf("Hello World!\n");
13. **return**;
14. }

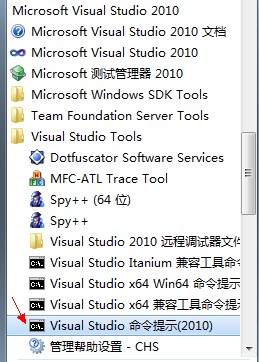
### 3.5.将C/C++编写的文件生成动态连接库

将**D:\Program Files\Java\jdk1.6.0\_26\include\jni.h**和**D:\Program Files\Java\jdk1.6.0\_26\include\win32\jni\_md.h**这两个文件拷贝到**D:\JNI\**目录下。与HelloWorldImpl.cpp同目录，目录结构如下图所示：



### 3.7 执行 cl/LD D:\JNI\HelloWorldImpl.cpp  得到HelloWorldImpl.dll文件

我使用的是visual studio 2010，要使用其中的cl命令，必须打开visual studio 命令行，如下图所示：

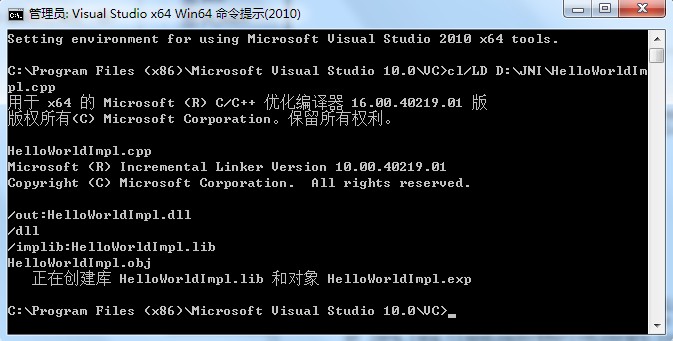


然后再命令行中输入如下命令

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. cl/LD D:\JNI\HelloWorldImpl.cpp

具体如下图所示：

  
执行完上述命令以后，我们在C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC可以看到生成的四个文件，分别是：

* ﻿﻿HelloWorldImpl.dll
* HelloWorldImpl.exp
* HelloWorldImpl.lib
* HelloWorldImpl.obj

将其中的HelloWorldImpl.dll拷贝到**D:\JNI\**目录下。

### 3.8.执行class得到结果

在cmd中运行：

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. d:\JNI>java HelloWorld

具体如下图所示：

http://hi.csdn.net/attachment/201111/19/0_13217061542EyJ.gif

## 4.在eclipse下运行

* 4.1在eclipse下创建一个叫做jnitest的project
* 4.2添加一个同3.1一样的HelloWorld.java
* 4.3保存HelloWorld.java以后在jnitest\bin目录下会生成HelloWorld.class。
* 4.4根据根据HelloWorld.class生成HelloWorld.h文件
* 4.5创建HelloWorldImpl.cpp来实现HelloWorld.h中的方法
* 4.6使用Visual studio 2010生成HelloWorldImpl.dll
* 4.7在Eclipse中运行HelloWorld程序，报错如下：

**[java]** [view plaincopy](http://blog.csdn.net/xw13106209/article/details/6989415)

1. java.lang.UnsatisfiedLinkError: no HelloWorldImpl in java.library.path
2. at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1738)
3. at java.lang.Runtime.loadLibrary0(Runtime.java:823)
4. at java.lang.System.loadLibrary(System.java:1028)
5. at HelloWorld.<clinit>(HelloWorld.java:6)

* 4.8将HelloWorldImpl.dll拷贝到C:\Windows\System32
* 4.9再次执行HelloWorld程序，程序正常运行，console输出“Hello World!”