JNI入门教程之HelloWorld篇

资料引用:http://www.knowsky.com/363050.html

本文讲述如何使用JNI技术实现HelloWorld，目的是让读者熟悉JNI的机制并编写第一个HelloWorld程序。

Java Native Interface（JNI）是Java语言的本地编程接口，是J2SDK的一部分。在java程序中，我们可以通过JNI实现一些用java语言不便实现的功能。通常有以下几种情况我们需要使用JNI来实现。

标准的java类库没有提供你的应用程序所需要的功能，通常这些功能是平台相关的

你希望使用一些已经有的类库或者应用程序，而他们并非用java语言编写的

程序的某些部分对速度要求比较苛刻，你选择用汇编或者c语言来实现并在java语言中调用他们

在《java核心技术》中，作者提到JNI的时候，建议不到万不得已不要使用JNI技术，一方面它需要你把握更多的知识才可以驾驭，一方面使用了JNI你的程序就会丧失可移植性。在本文我们跳过JNI的底层机制，读者最好先把它想象为本地代码和java代码的粘合剂。关系如下图所示：

下面我们开始编写HelloWorld程序，由于涉及到要编写c/c++代码因此我们会在开发中使用Microsoft VC++工具。

编写java代码

我们在硬盘上建立一个hello目录作为我们的工作目录，首先我们需要编写自己的java代码，在java代码中我们会声明native方法，代码非常简单。如下所示

class HelloWorld

{

public native void displayHelloWorld();

static {

System.loadLibrary("hello");

}

public static void main(String[] args) {

new HelloWorld().displayHelloWorld();

}

}

注重我们的displayHelloWorld()方法的声明，它有一个要害字native，表明这个方法使用java以外的语言实现。方法不包括实现，因为我们要用c/c++语言实现它。注重System.loadLibrary("hello")这句代码，它是在静态初始化块中定义的，系统用来装载hello共享库，这就是我们在后面生成的hello.dll（假如在其他的操作系统可能是其他的形式，比如hello.so）

编译java代码

javac HelloWorld.java 生成HelloWorld.class文件

创建.h文件

这一步中我们要使用javah命令生成.h文件，这个文件要在后面的c/c++代码中用到，我们运行

javah HelloWorld。这样我们可以看到在相同目录下生成了一个HelloWorld.h文件，文件内容如下

在此我们不对他进行太多的解释。

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

#include <jni.h>

/\* Header for class HelloWorld \*/

#ifndef \_Included\_HelloWorld

#define \_Included\_HelloWorld

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*

\* Class: HelloWorld

\* Method: displayHelloWorld

\* Signature: ()V

\*/

JNIEXPORT void JNICALL Java\_HelloWorld\_displayHelloWorld

(JNIEnv \*, jobject);

#ifdef \_\_cplusplus

}

#endif

#endif

编写本地实现代码

在这部分我们要用C/C++语言实现java中定义的方法，我们在VC++中新建一个Project,然后创建一个HelloWorldImp.cpp文件，内容如下

#include <jni.h>

#include "HelloWorld.h"

#include <stdio.h>

JNIEXPORT void JNICALL

Java\_HelloWorld\_displayHelloWorld(JNIEnv \*env, jobject obj)

{

printf("Hello world!\n");

return;

}

注重我们这里include了 jni.h和刚才得到的HelloWorld.h文件。因此你要在VC++里面设置好，jni.h在JAVA\_HOME/include里面。编译通过后再生成hello.dll文件。

运行java程序

把上面生成的hello.dll文件复制到我们的工作目录，这时候我们的目录中包括HelloWorld.java，HelloWorld.class和hello.dll文件。运行java HelloWorld命令，则可在控制台看到Hello world

的输出了。

-

Java以其跨平台的特性深受人们喜爱，而又正由于它的跨平台的目的，使得它和本地机器的各种内部联系变得很少，约束了它的功能。解决JAVA对本地操作的一种方法就是JNI。

　　JAVA通过JNI调用本地方法，而本地方法是以库文件的形式存放的（在WINDOWS平台上是DLL文件形式，在UNIX机器上是SO文件形式）。通过调用本地的库文件的内部方法，使JAVA可以实现和本地机器的紧密联系，调用系统级的各接口方法。

　　简单介绍及应用如下：

　　一、JAVA中所需要做的工作

　　在JAVA程序中，首先需要在类中声明所调用的库名称，如下：

　　static {

　　System.loadLibrary(“goodlUCk”);

　　}

　　在这里，库的扩展名字可以不用写出来，究竟是DLL还是SO，由系统自己判定。

　　还需对将要调用的方法做本地声明，要害字为native。且只需要声明，而不需要具体实现。如下：

　　public native static void set(int i);

　　public native static int get();

　　然后编译该JAVA程序文件，生成CLASS，再用JAVAH命令，JNI就会生成C/C++的头文件。

　　例如程序testdll.java，内容为：

　　public class testdll

　　{

　　static

　　{

　　System.loadLibrary("goodluck");

　　}

　　public native static int get();

　　public native static void set(int i);

　　public static void main(String[] args)

　　{

　　testdll test = new testdll();

　　test.set(10);

　　System.out.println(test.get());

　　}

　　}

　　用javac testdll.java编译它，会生成testdll.class。

　　再用javah testdll，则会在当前目录下生成testdll.h文件，这个文件需要被C/C++程序调用来生成所需的库文件。

　　二、C/C++中所需要做的工作

　　对于已生成的.h头文件，C/C++所需要做的，就是把它的各个方法具体的实现。然后编译连接成库文件即可。再把库文件拷贝到JAVA程序的路径下面，就可以用JAVA调用C/C++所实现的功能了。

　　接上例子。我们先看一下testdll.h文件的内容：

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

　　#include

　　/\* Header for class testdll \*/

　　#ifndef \_Included\_testdll

　　#define \_Included\_testdll

　　#ifdef \_\_cplusplus

　　extern "C" {

　　#endif

　　/\*

　　\* Class: testdll

　　\* Method: get

　　\* Signature: ()I

　　\*/

　　JNIEXPORT jint JNICALL Java\_testdll\_get (JNIEnv \*, jclass);

　　/\*

　　\* Class: testdll

　　\* Method: set

　　\* Signature: (I)V

　　\*/

　　JNIEXPORT void JNICALL Java\_testdll\_set (JNIEnv \*, jclass, jint);

　　#ifdef \_\_cplusplus

　　}

　　#endif

　　#endif

　　在具体实现的时候，我们只关心两个函数原型

　　JNIEXPORT jint JNICALL Java\_testdll\_get (JNIEnv \*, jclass); 和

　　JNIEXPORT void JNICALL Java\_testdll\_set (JNIEnv \*, jclass, jint);

　　这里JNIEXPORT和JNICALL都是JNI的要害字，表示此函数是要被JNI调用的。而jint是以JNI为中介使JAVA的int类型与本地的int沟通的一种类型，我们可以视而不见，就当做int使用。函数的名称是JAVA\_再加上java程序的package路径再加函数名组成的。参数中，我们也只需要关心在JAVA程序中存在的参数，至于JNIEnv\*和jclass我们一般没有必要去碰它。

　　好，下面我们用testdll.cpp文件具体实现这两个函数：

　　#include "testdll.h"

　　int i = 0;

　　JNIEXPORT jint JNICALL Java\_testdll\_get (JNIEnv \*, jclass)

　　{

　　return i;

　　}

　　JNIEXPORT void JNICALL Java\_testdll\_set (JNIEnv \*, jclass, jint j)

　　{

　　i = j;

　　}

　　编译连接成库文件，本例是在WINDOWS下做的，生成的是DLL文件。并且名称要与JAVA中需要调用的一致，这里就是goodluck.dll 。把goodluck.dll拷贝到testdll.class的目录下，java testdll运行它，就可以观察到结果了。

　　我的项目比较复杂，需要调用动态链接库，这样在JNI传送参数到C程序时，需要对参数进行处理转换。才可以被C程序识别。

　　大体程序如下：

　　public class SendSMS {

　　static

　　{

　　System.out.println(System.getProperty("java.library.path"));

　　System.loadLibrary("sms");

　　}

　　public native static int SmsInit();

　　public native static int SmsSend(byte[] mobileNo, byte[] smContent);

　　}

　　在这里要注重的是，path里一定要包含类库的路径，否则在程序运行时会抛出异常：

　　java.lang.UnsatisfiedLinkError: no sms in java.library.path

　　at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1491)

　　at java.lang.Runtime.loadLibrary0(Runtime.java:788)

　　at java.lang.System.loadLibrary(System.java:834)

　　at com.mobilesoft.sms.mobilesoftinfo.SendSMS.(SendSMS.java:14)

　　at com.mobilesoft.sms.mobilesoftinfo.test.main(test.java:18)

　　Exception in thread "main"

　　指引的路径应该到.dll文件的上一级，假如指到.dll，则会报：

　　java.lang.UnsatisfiedLinkError: C:\sms.dll: Can't find dependent libraries

　　at java.lang.ClassLoader$NativeLibrary.load(Native Method)

　　at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1560)

　　at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1485)

　　at java.lang.Runtime.loadLibrary0(Runtime.java:788)

　　at java.lang.System.loadLibrary(System.java:834)

　　at com.mobilesoft.sms.mobilesoftinfo.test.main(test.java:18)

　　Exception in thread "main"

　　通过编译，生成com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS.h头文件。（建议使用Jbuilder进行编译，操作比较简单！）这个头文件就是Java和C之间的纽带。要非凡注重的是方法中传递的参数jbyteArray，这在接下来的过程中会重点介绍。

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

　　#include

　　/\* Header for class com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS \*/

　　#ifndef \_Included\_com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS

　　#define \_Included\_com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS

　　#ifdef \_\_cplusplus

　　extern "C" {

　　#endif

　　/\*

　　\* Class: com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS

　　\* Method: SmsInit

　　\* Signature: ()I

　　\*/

　　JNIEXPORT jint JNICALL Java\_com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS\_SmsInit

　　(JNIEnv \*, jclass);

　　/\*

　　\* Class: com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS

　　\* Method: SmsSend

　　\* Signature: ([B[B)I

　　\*/

　　JNIEXPORT jint JNICALL Java\_com\_mobilesoft\_sms\_mobilesoftinfo\_SendSMS\_SmsSend

　　(JNIEnv \*, jclass, jbyteArray, jbyteArray);

　　#ifdef \_\_cplusplus

　　}

　　#endif

　　#endif

　　对于我要调用的C程序的动态链接库，C程序也要提供一个头文件，sms.h。这个文件将要调用的方法罗列了出来。

　　/\*

　　\* SMS API

　　\* Author: yippit

　　\* Date: 2004.6.8

　　\*/

　　#ifndef MCS\_SMS\_H

　　#define MCS\_SMS\_H

　　#define DLLEXPORT \_\_declspec(dllexport)

　　/\*sms storage\*/

　　#define SMS\_SIM 0

　　#define SMS\_MT 1

　　/\*sms states\*/

　　#define SMS\_UNREAD 0

　　#define SMS\_READ 1

　　/\*sms type\*/

　　#define SMS\_NOPARSE -1

　　#define SMS\_NORMAL 0

　　#define SMS\_FLASH 1

　　#define SMS\_MMSNOTI 2

　　typedef struct tagSmsEntry {

　　int index; /\*index, start from 1\*/

　　int status; /\*read, unread\*/

　　int type; /\*-1-can't parser 0-normal, 1-flash, 2-mms\*/

　　int storage; /\*SMS\_SIM, SMS\_MT\*/

　　char date[24];

　　char number[32];

　　char text[144];

　　} SmsEntry;

　　DLLEXPORT int SmsInit(void);

　　DLLEXPORT int SmsSend(char \*phonenum, char \*content);

　　DLLEXPORT int SmsSetSCA(char \*sca);

　　DLLEXPORT int SmsGetSCA(char \*sca);

　　DLLEXPORT int SmsSetInd(int ind);

　　DLLEXPORT int SmsGetInd(void);

　　DLLEXPORT int SmsGetInfo(int storage, int \*max, int \*used);

　　DLLEXPORT int SmsSaveFlash(int flag);

　　DLLEXPORT int SmsRead(SmsEntry \*entry, int storage, int index);

　　DLLEXPORT int SmsDelete(int storage, int index);

　　DLLEXPORT int SmsModifyStatus(int storage, int index); /\*unread -> read\*/

　　#endif

　　在有了这两个头文件之后，就可以进行C程序的编写了。也就是实现对JNI调用的两个方法。在网上的资料中，由于调用的方法实现的都比较简单，（大多是打印字符串等）所以避开了JNI中最麻烦的部分，也是最要害的部分，参数的传递。由于Java和C的编码是不同的，所以传递的参数是要进行再处理，否则C程序是会对参数在编译过程中提出警告，