* Java程序向C程序传入一个字符串，C程序对该字符串转成大写形式后回传给Java程序
* C程序向Java程序返回一个定长的整型数组成的数组
* 传递字符串数组
* C程序向Java程序传递对象数组
* 自定义对象为参数和返回值

作为主调方的Java源程序TestJNI.java如下。

代码清单15-4 在Linux平台上调用C函数的例程——TestJNI.java

1. public class TestJNI

2. {

3. static

4. {

5. System.loadLibrary("testjni");//载入静态库，test函数在其中实现

6. }

7.

8. private native void testjni(); //声明本地调用

9.

10. public void test()

11. {

12. testjni();

13. }

14.

15. public static void main(String args[])

16. {

17. TestJNI haha = new TestJNI();

18. haha.test();

19. }

20. }

TestJNI.java声明从libtestjni.so（注意Linux平台的动态链接库文件的扩展名是.so）中调用函数testjni()。

在Linux平台上，遵循JNI规范的动态链接库文件名必须以“lib”开头。例如在上面的Java程序中指定的库文件名为“testjni”，则实际的库文件应该命名为“libtestjni.so”。

编译TestJNI.java，并为C程序生成头文件：

java TestJNI.java

javah TestJNI

提供testjni()函数的testjni.c源文件如下。

代码清单15-5 在Linux平台上调用C函数的例程——testjni.c

#include <stdio.h>

#include <TestJNI.h>

JNIEXPORT void JNICALL Java\_TestJNI\_testjni(JNIEnv \*env, jobject obj){

printf("haha---------go into c!!!\n");

}

编写Makefile文件如下，JDK安装的位置请读者自行调整：

libtestjni.so:testjni.o

gcc -rdynamic -shared -o libtestjni.so testjni.o

testjni.o:testjni.c TestJNI.h

gcc -c testjni.c -I./ -I/usr/java/jdk1.6.0\_00/include -I/usr/java/jdk1.6.0\_00/include/linux

在Makefile文件中，我们描述了最终的 libtestjin.so依赖于目标文件testjni.o，而testjni.o则依赖于testjni.c源文件和TestJNI.h头文件。请注 意，我们在将testjni.o连接成动态链接库文件时使用了“-rdynamic”选项。

执行make命令编译testjni.c。Linux平台和在Windows平台上类似，有3种方法可以让Java程序找到并装载动态链接库文件。

— 将动态链接库文件放置在当前路径下。

— 将动态链接库文件放置在LD\_LIBRARY\_PATH环境变量所指向的路径下。注意这一点和Windows平台稍有区别，Windows平台参考PATH环境变量。

— 在启动JVM时指定选项“-Djava.library.path”，将动态链接库文件放置在该选项所指向的路径下。

从下一节开始，我们开始接触到在JNI框架内Java调用C程序的一些高级话题，包括如何传递参数、如何传递数组、如何传递对象等。

各种类型数据的传递是跨平台、跨语言互操作的永恒话题，更复杂的操作其实都可以分解为各种 基本数据类型的操作。只有掌握了基于各种数据类型的互操作，才能称得上掌握了JNI开发。从下一节开始，环境和步骤不再是阐述的重点，将不再花费专门的篇 幅，例程中的关键点将成为我们关注的焦点。

**15.2.2.3 传递字符串**

到目前为止，我们还没有实现Java程序向C程序传递参数，或者C程序向Java程序传递参数。本例程将由Java程序向C程序传入一个字符串，C程序对该字符串转成大写形式后回传给Java程序。

Java源程序如下。

代码清单15-6 在Linux平台上调用C函数的例程——Sample1

public class Sample1

{

public native String stringMethod(String text);

public static void main(String[] args)

{

System.loadLibrary("Sample1");

Sample1 sample = new Sample1();

String text = sample.stringMethod("Thinking In Java");

System.out.println("stringMethod: " + text);

}

}

Sample1.java以“Thinking In Java”为参数调用libSample1.so中的函数stringMethod()，在得到返回的字符串后打印输出。

Sample1.c的源程序如下。

代码清单15-7 在Linux平台上调用C函数的例程——Sample1.c

#include <Sample1.h>

#include <string.h>

JNIEXPORT jstring JNICALL Java\_Sample1\_stringMethod(JNIEnv \*env, jobject obj, jstring string)

{

const char \*str = (\*env)->GetStringUTFChars(env, string, 0);

char cap[128];

strcpy(cap, str);

(\*env)->ReleaseStringUTFChars(env, string, str);

int i=0;

for(i=0;i<strlen(cap);i++)

\*(cap+i)=(char)toupper(\*(cap+i));

return (\*env)->NewStringUTF(env, cap);

}

首先请注意函数头部分，函数接收一个jstring类 型的输入参数，并输出一个jstring类型的参数。jstring是jni.h中定义的数据类型，是JNI框架内特有的字符串类型，因为jni.h在 Sample1.h中被引入，因此在Sample1.c中无须再次引入。

程序的第4行是从JNI调用上下文中获取UTF编码的输入字符，将其放在指针str所指向 的一段内存中。第9行是释放这段内存。第13行是将经过大写转换的字符串予以返回，这一句使用了NewStringUTF()函数，将C语言的字符串指针 转换为JNI的jstring类型。JNIEnv也是在jni.h中定义的，代表JNI调用的上下文，GetStringUTFChars()、 ReleaseStringUTFChars()和NewStringUTF()均是JNIEnv的函数。

**15.2.2.4 传递整型数组**

本节例程将首次尝试在JNI框架内启用数组：C程序向Java程序返回一个定长的整型数组成的数组，Java程序将该数组打印输出。

Java程序的源代码如下。

代码清单15-8 在Linux平台上调用C函数的例程——Sample2

public class Sample2

{

public native int[] intMethod();

public static void main(String[] args)

{

System.loadLibrary("Sample2");

Sample2 sample=new Sample2();

int[] nums=sample.intMethod();

for(int i=0;i<nums.length;i++)

System.out.println(nums[i]);

}

}

Sample2.java调用libSample2.so中的函数intMethod()。Sample2.c的源代码如下。

代码清单15-9 在Linux平台上调用C函数的例程——Sample2.c

#include <Sample2.h>

JNIEXPORT jintArray JNICALL Java\_Sample2\_intMethod(JNIEnv \*env, jobject obj)

{

inti = 1;

jintArray array;//定义数组对象

array = (\*env)-> NewIntArray(env, 10);

for(; i<= 10; i++)

(\*env)->SetIntArrayRegion(env, array, i-1, 1, &i);

/\* 获取数组对象的元素个数 \*/

int len = (\*env)->GetArrayLength(env, array);

/\* 获取数组中的所有元素 \*/

jint\* elems = (\*env)-> GetIntArrayElements(env, array, 0);

for(i=0; i<len; i++)

printf("ELEMENT %d IS %d\n", i, elems[i]);

return array;

}

Sample2.c涉及了两个jni.h定义的整型数 相关的数据类型：jint和jintArray，jint是在JNI框架内特有的整数类型。程序的第7行开辟出一个长度为10 的jint数组。然后依次向该数组中放入元素1-10。第11行至第16行不是程序的必须部分，纯粹是为了向读者们演示GetArrayLength() 和GetIntArrayElements()这两个函数的使用方法，前者是获取数组长度，后者则是获取数组的首地址以便于遍历数组。

设置 数组里面的元素

**方法解释**

(\*env)->SetIntArrayRegion(env,array,start,len,buffer) , 从start开始复制长度为len 的数据 buffer到 array 中

**15.2.2.5 传递字符串数组**

本节例程是对上节例程的进一步深化：虽然仍然是传递数组，但是数组的基类换成了字符串这样一种对象数据类型。Java程序将向C程序传入一个包含中文字符的字符串，C程序并没有处理这个字符串，而是开辟出一个新的字符串数组返回给Java程序，其中还包含两个汉字字符串。

Java程序的源代码如下。

代码清单15-10 在Linux平台上调用C函数的例程——Sample3

public class Sample3

{

public native String[] stringMethod(String text);

public static void main(String[] args) throws java.io.UnsupportedEncodingException

{

System.loadLibrary("Sample3");

Sample3 sample = new Sample3();

String[] texts = sample.stringMethod("java编程思想");

for(int i=0;i<texts.length;i++)

{

texts[i]=new String(texts[i].getBytes("ISO8859-1"),"GBK");

System.out.print( texts[i] );

}

System.out.println();

}

}

Sample3.java调用libSample3.so中的函数stringMethod()。Sample3.c的源代码如下：

代码清单15-11 在Linux平台上调用C函数的例程——Sample3.c

#include <Sample3.h>

#include <string.h>

#include <stdlib.h>

#define ARRAY\_LENGTH 5

JNIEXPORT jobjectArray JNICALL Java\_Sample3\_stringMethod(JNIEnv \*env, jobject obj, jstring string)

{

jclass objClass = (\*env)->FindClass(env, "java/lang/String");

jobjectArray texts= (\*env)->NewObjectArray(env, (jsize)ARRAY\_LENGTH, objClass, 0);

jstring jstr;

char\* sa[] = { "Hello,", "world!", "JNI", "很", "好玩" };

int i=0;

for(;i<ARRAY\_LENGTH;i++)

{

jstr = (\*env)->NewStringUTF( env, sa[i] );

(\*env)->SetObjectArrayElement(env, texts, i, jstr);//必须放入jstring

}

return texts;

}

第9、10行是我们需要特别关注的地方：JNI框架并 没有定义专门的字符串数组，而是使用jobjectArray——对象数组，对象数组的基类是jclass，jclass是JNI框架内特有的类型，相当 于Java语言中的Class类型。在本例程中，通过FindClass()函数在JNI上下文中获取到java.lang.String的类型 （Class），并将其赋予jclass变量。

在例程中我们定义了一个长度为5的对象数组texts，并在程序的第18行向其中循环放入预先定义好的sa数组中的字符串，当然前置条件是使用NewStringUTF()函数将C语言的字符串转换为jstring类型。

本例程的另一个关注点是C程序向Java程序传递的中文字符，在Java程序中能否正常显 示的问题。在笔者的试验环境中，Sample3.c是在Linux平台上编辑的，其中的中文字符则是用支持GBK的输入法输入的，而Java程序采用 ISO8859\_1字符集存放JNI调用的返回字符，因此在“代码清单15-10在Linux平台上调用C函数的例程——Sample3”的第14行中将 其转码后输出。

15.2.2.6 传递对象数组

本节例程演示的是C程序向Java程序传递对象数组，而且对象数组中存放的不再是字符串，而是一个在Java中自定义的、含有一个topic属性的MailInfo对象类型。

MailInfo对象定义如下。

代码清单15-12 在Linux平台上调用C函数的例程——MailInfo

public class MailInfo {

public String topic;

public String getTopic()

{

return this.topic;

}

public void setTopic(String topic)

{

this.topic=topic;

}

}

Java程序的源代码如下。

代码清单15-13 在Linux平台上调用C函数的例程——Sample4

public class Sample4

{

public native MailInfo[] objectMethod(String text);

public static void main(String[] args)

{

System.loadLibrary("Sample4");

Sample4 sample = new Sample4();

MailInfo[] mails = sample.objectMethod("Thinking In Java");

for(int i=0;i<mails.length;i++)

System.out.println(mails[i].topic);

}

}

Sample4.java调用libSample4.so中的objectMethod()函数。Sample4.c的源代码如下。

代码清单15-14 在Linux平台上调用C函数的例程——Sample4.c

#include <Sample4.h>

#include <string.h>

#include <stdlib.h>

#define ARRAY\_LENGTH 5

JNIEXPORT jobjectArray JNICALL Java\_Sample4\_objectMethod(JNIEnv \*env, jobject obj, jstring string)

{

jclass objClass = (\*env)->FindClass(env, "java/lang/Object");

jobjectArray mails= (\*env)->NewObjectArray(env, (jsize)ARRAY\_LENGTH, objClass, 0);

jclass objectClass = (\*env)->FindClass(env, "MailInfo");

jfieldID topicFieldId = (\*env)->GetFieldID(env, objectClass, "topic", "Ljava/lang/String;");

int i=0;

for(;i<ARRAY\_LENGTH;i++)

{

(\*env)->SetObjectField(env, obj, topicFieldId, string);

(\*env)->SetObjectArrayElement(env, mails, i, obj);

}

return mails;

}

程序的第9、10行读者们应该不会陌生，在上一节的例 程中已经出现过，不同之处在于这次通过FindClass()函数在JNI上下文中获取的是java.lang.Object的类型（Class），并将 其作为基类开辟出一个长度为5的对象数组，准备用来存放MailInfo对象。

程序的第12、13行的目的则是创建一个jfieldID类型的变量，在JNI中，操作对 象属性都是通过jfieldID进行的。第12行首先查找得到MailInfo的类型（Class），然后基于这个jclass进一步获取其名为 topic的属性，并将其赋予jfieldID变量。

程序的第18、19行的目的是循环向对象数组中放入jobject对象。 SetObjectField()函数属于首次使用，该函数的作用是向jobject的属性赋值，而值的内容正是Java程序传入的jstring变量 值。请注意在向对象属性赋值和向对象数组中放入对象的过程中，我们使用了在函数头部分定义的jobject类型的环境参数obj作为中介。至此，JNI框 架固有的两个环境入参env和obj，我们都有涉及。

**一、定义作为输入和返回的自定义类 （仅提供两个简单类型和一个打印函数）**

1.package com.example.jniexample;

2.

3.import android.util.Log;

4.

5.public class JNIParam {

6. public int mInt;

7. public String mString;

8.

9. JNIParam(){

10. mInt = 0;

11. mString = "0";

12. }

13.

14. public void print(String tag){

15. Log.d(tag, String.format("print: mInt=%d, mString=%s", mInt, mString));

16. }

17.}

二. 定义和JNI通信的java封装

功能：测试输入参数和返回值都为自定义对象的本地方法，同时提供必要的调试打印

1.package com.example.jniexample;

2.

3.public class JNIParamTest {

4. private final String TAG = "JNIParamTest";

5.

6. static {

7. System.loadLibrary("JNIParamTest");

8. };

9.

10. public void runParamTest(){

11. JNIParam paramIn = new JNIParam();

12. paramIn.print(TAG);

13.

14. JNIParam paramOut = doTest(paramIn);

15. if( paramOut != null ) paramOut.print(TAG);

16. }

17.

18. //JNI

19. private native JNIParam doTest(JNIParam paramIn);

20.}

三、实现JNI的本地实现

a. 头文件 （JNIParamTest.h)

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

2.#include <jni.h>

3./\* Header for class com\_example\_jniexample\_JNIParamTest \*/

4.

5.#ifndef \_Included\_com\_example\_jniexample\_JNIParamTest

6.#define \_Included\_com\_example\_jniexample\_JNIParamTest

7.#ifdef \_\_cplusplus

8.extern "C" {

9.#endif

10./\*

11. \* Class: com\_example\_jniexample\_JNIParamTest

12. \* Method: doTest

13. \* Signature: (Lcom/example/jniexample/JNIParam;)Lcom/example/jniexample/JNIParam;

14. \*/

15.JNIEXPORT jobject JNICALL Java\_com\_example\_jniexample\_JNIParamTest\_doTest

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

17.

18.#ifdef \_\_cplusplus

19.}

20.#endif

21.#endif

b. c文件 （JNIParamTest.c)

1.#include "JNIParamTest.h"

2.

3.#include "android/log.h"

4.#define TAG "JNI\_ParamTest"

5.#define LOGD(fmt, args...) \_\_android\_log\_print(ANDROID\_LOG\_DEBUG, TAG, fmt, ##args)

6.

7./\*

8. \* Class: com\_example\_jniexample\_JNIParamTest

9. \* Method: doTest

10. \* Signature: (Lcom/example/jniexample/JNIParam;)Lcom/example/jniexample/JNIParam;

11. \*/

12.JNIEXPORT jobject JNICALL Java\_com\_example\_jniexample\_JNIParamTest\_doTest

13. (JNIEnv \*env, jobject thiz, jobject paramIn)

14.{

15. jclass paramInClass = (\*env)->GetObjectClass(env, paramIn);

16. if( paramInClass){

17. jboolean iscopy;

18. jfieldID intId = (\*env)->GetFieldID(env, paramInClass, "mInt", "I");

19. jint num = (int)(\*env)->GetIntField(env, paramIn, intId);

20. LOGD("num = %d", num);

21.

22. jfieldID strId = (\*env)->GetFieldID(env, paramInClass, "mString", "Ljava/lang/String;");

23. jstring str = (jstring)(\*env)->GetObjectField(env, paramIn, strId);

24. const char \*locstr = (\*env)->GetStringUTFChars(env, str, &iscopy);

25. LOGD("str = %s", locstr);

26.

27. (\*env)->ReleaseStringUTFChars(env, str, locstr);

28. }

29.

30. jclass cls = (\*env)->FindClass(env, "com/example/jniexample/JNIParam");

31. jmethodID id = (\*env)->GetMethodID(env, cls, "<init>", "()V");

32.

33. jobject paramOut = (\*env)->NewObjectA(env, cls, id, 0);

34.

35. jfieldID intId = (\*env)->GetFieldID(env, cls, "mInt", "I");

36. (\*env)->SetIntField(env, paramOut, intId, 1);

37.

38. jfieldID strId = (\*env)->GetFieldID(env, cls, "mString", "Ljava/lang/String;");

39. (\*env)->SetObjectField(env, paramOut, strId, (jstring)(\*env)->NewStringUTF(env, "1"));

40.

41. return paramOut;

42.}

四、提供一个测试环境

1.package com.example.jniexample;

2.

3.import android.os.Bundle;

4.import android.app.Activity;

5.import android.view.Menu;

6.

7.public class JNIExample extends Activity {

8.

9. @Override

10. public void onCreate(Bundle savedInstanceState) {

11. super.onCreate(savedInstanceState);

12. setContentView(R.layout.activity\_jniexample);

13.

14. JNIParamTest test = new JNIParamTest();

15. test.runParamTest();

16. }

17.

18. @Override

19. public boolean onCreateOptionsMenu(Menu menu) {

20. getMenuInflater().inflate(R.menu.activity\_jniexample, menu);

21. return true;

22. }

23.}

实现了一个Activity, 启动就调用

五、测试结果： （logcat 查看）

1.D/JNIParamTest( 6135): print: mInt=0, mString=0

2.D/JNI\_ParamTest( 6135): num = 0

3.D/JNI\_ParamTest( 6135): str = 0

4.D/JNIParamTest( 6135): print: mInt=1, mString=1