JNI 技术之静态注册

作者:程姚根　　联系方式:chengyaogen123@163.com

一、JNI技术介绍

JNI是**Java Native Interface**的缩写，中文为JAVA本地调用。它是JAVA的一种机制，通过这种机制可以让**JAVA代码调到C/C++写的接口，也可以在C/C++回调JAVA代码**。它的出现，让JAVA成员和C/C++成员无缝对接,它让JAVA程序员真正做到不需要关心底层细节，只需要调用对应的接口就可以，底层的细节让苦逼的C/C++程序去完善吧。换句话说，JAVA层只需要声明接口就可以了，这个接口具体的实现是通过C/C++语言来编写的。

自从知道了JNI，我再也不鄙视JAVA了，从此膜拜JAVA的设计者，他是多么无敌的存在呀。

为什么要学习JNI技术呢?这门技术在历史的长河里几乎被埋没，自从Android开天辟地的出现后，在一次将它呈现在历史的舞台，只有真正的了解了，才能更好的去驾驭Android。

好了，接下来我们就来一层层的揭开JNI的面纱吧!

二、JAVA 是如何调用到用C/C++编写的接口?

在C/C++的程序员的世界里，我们会经常使用printf等这样的C库函数，我们并没有真的实现这些函数。我们可以很放心的调用他们，因为库中有它的实现，在实际运行的代码的时候，只需要加载动态库就可以了。通过加载动态库我们就可以找到对应的实现。

JNI技术也是如此，在JAVA层，我们调用的接口不是JAVA程序实现的，而是C/C++程序员实现的,那JAVA是如何找到这个接口的呢?呵呵,道理是一样的，JAVA只需要加载库就可以了，这个库中有C/C++程序员编写的对应函数接口。

你明白了吗?下面我们以代码的是形式呈现JAVA层的实现,如下所示:

```
1 class Test{
2
3     //通过"native"关键字申明helloWorld是本地接口
4     //即表明这个函数接口是由C/C++实现的，在JAVA层直接调用就可以了
```

```
5     public native void helloWorld();
6
7     static{
8
9         System.out.printf("Run static section \n");
10        //加载HelloWorld库
11        System.loadLibrary("HelloWorld");
12    }
13
14    public static void main(String[] args)
15    {
16        Test obj1;
17
18        obj1 = new Test();
19        obj1.helloWorld();
20    }
21 }
```

通过上面的代码我们可以知道，JAVA层要调用到C/C++写的代码，JAVA程序员必须做以下事情:

1.通过"native"修饰成员函数接口，表示这个函数接口是由C/C++来实现的
2.通过System.loadLibrary()加载对应的动态库

玩过库的人都会疑问，加载的库写法是不是应该有问题呀?

动态库在windows下叫name.dll ，在Linux 下叫libname.so ,这里怎么没有".dll"和".so"呀？
因为在实际使用的时候，在Linux 平台会扩展成libHelloWorld.so,在Windows平台会扩展成HelloWorld.dll。
所以在这里，我们在加载动态库的时候只需要写上"库名"就可以了。

不对，还有一个疑问,动态库中一般有多个函数接口，JAVA怎么知道哪一个才是它想要调用的接口呀?
是呀,哪一个才是JAVA需要调用的接口呢?不急，我们来看一下C/C++实现的代码,如下所示:

```
1 #include <jni.h>//必须包含
2 #include <stdio.h>
3 #include "Test.h"
4
5 //JNIEXPORT，JNICALL都是一个空的宏，只取标识作用
```

```
 6 JNIEXPORT void JNICALL Java_Test_helloWorld(JNIEnv *env, jobject obj)
 7 {
 8     printf("Hello word\n");
 9     return;
10 }
11
```

这个名字看起来挺长的实际上我们注意看我们会发现:

"Java"　　表明这个函数最后是要被JAVA调用的

"Test"　　在 JAVA层表示类名

"helloWorld"　在JAVA层是成员函数名

参数后面我们在解释。

实际上，当JAVA 加载完动态库后，在它调用"helloWorld()"方法时，它就在函数库中找"Java_Test_helloWorld"这个函数，如果存在，则调用它，如果不存在则报错。这种方法在JNI技术中也叫"静态注册"。

是不是所有的JNI函数，都需要写成上面那种固定的造型呢？

答案：不是的，因为JAVA中存在函数重载，所以C/C++在实现的时候，对应的函数名也是有变化的。

那在C/C++到底，该怎样写?

其实JAVA都给我们做好了，在我们把JAVA程序编译后，我们可以通过JDK给我们提供的javah这个程序从 .class文件中得到C/C++中实现的函数造型。使用方法如下:

javah　-jni　类名

执行完后，在你的本地会生成一个类名命名的.h文件，在我们这里叫做Test.h，下面我们看看Test.h文件的内容:

```
 1 /* DO NOT EDIT THIS FILE - it is machine generated */
 2 #include <jni.h>
 3 /* Header for class Test */
 4
 5 #ifndef _Included_Test
 6 #define _Included_Test
 7 #ifdef __cplusplus
 8 extern "C" {
 9 #endif
10 /*
11  * Class:     Test
12  * Method:    helloWorld
```

```
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_Test_helloWorld
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
```

三、生成动态库，运行JAVA程序

(1)Linux 生成动态库

gcc -fpic -shared -o libHelloWorld.so hello.c -I /home/cyg/workdir/jdk-1.6/include/ -I /home/cyg/workdir/jdk-1.6/include/linux/

gcc : C语言编译器
fpic : gcc 参数，表示生成位置无关的代码
shared : gcc 参数，表示生成动态库
-I : 指定jni.h文件所在的路径,在编译的时候一定指定正确

(2)运行JAVA程序

```
cyg@ubuntu:~/workdir/learn_android/JNI/hello_jni$ ls
hello.c  hello.java  libHelloWorld.so  Test.class  Test.h
cyg@ubuntu:~/workdir/learn_android/JNI/hello_jni$ java Test
Run static section
Exception in thread "main" java.lang.UnsatisfiedLinkError: no HelloWorld in java.library.path
        at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1682)
        at java.lang.Runtime.loadLibrary0(Runtime.java:823)
        at java.lang.System.loadLibrary(System.java:1030)
        at Test.<clinit>(hello.java:12)
```

出错的原因是，没有找到动态库，JAVA提供了两种方法，让其找到动态库:

A.在Linux 下，在LD_LIBRARY_PATH环境变量中指定动态库的路径

```
cyg@ubuntu:~/workdir/learn_android/JNI/hello_jni$ export LD_LIBRARY_PATH=.
cyg@ubuntu:~/workdir/learn_android/JNI/hello_jni$ java Test
Run static section
```

```
Hello word
```

B.运行JAVA程序的时候，通过-Djava.library.path指定动态库的路径

```
cyg@ubuntu:~/workdir/learn_android/JNI/hello_jni$ java -Djava.library.path=. Test
Run static section
Hello word
```