JNI 攻略之十一－－启动虚拟机调用 java 类

江苏 无锡 缪小东

一、一个简单的 java 程序

下面是一个简单的 java 程序！主要是供虚拟机直接调用的！我们可以编写任意的 java 程序，然后在 c 中调用虚拟机执行它！通常我们会有此场景：某些本地系统可能不支持多线程，但你对 java 的多线程又比较精通，同时对 c 还是比较了解，此时我们就可以在 c 中调用虚拟机，执行改线程程序！更一般的：以前有一个用 c 写的服务器程序，它可以介绍客户的申请，当然这种申请在某一个时刻肯定是很多很多的，因此我们就想让这些申请交由 java 的线程完成！此时就可以使用 c 调用虚拟机执行线程操作！对用户隐藏了内部调用虚拟机执行线程的细节！

```
//示例程序 Prog.java
public class Prog {
    public static void main(String[] args) {
        System.out.println("Hello World " + args[0]);
    }
}
```

二、调用虚拟机执行 java 程序的代码

下面是启动虚拟机执行 java 程序的过程。（在此之前请先阅读上面一篇）

```
/* invoke&exec.c */
#include <stdio.h>
#include <jni.h>

int main() {
    int res;
    JavaVM *jvm;
    JNIEnv *env;
    JavaVMInitArgs vm_args;
    JavaVMOption options[3];                    //以上是启动虚拟机的一些参数

    jclass cls , strcls;                        //将要被虚拟机执行的类和创建的 String 类

    jmethodID mid ;                             //类的方法标识符
    jstring jstr ;                              //输入参数
    jobjectArray args ;                         //主函数输入参数

     vm_args.version=JNI_VERSION_1_2;           //这个字段必须设置为该值

     /*设置初始化参数*/
     options[0].optionString = "-Djava.compiler=NONE";
```

```
    options[1].optionString = "-Djava.class.path=.";
    options[2].optionString = "-verbose:jni";           //用于跟踪运行时的信息
    /*版本号设置不能漏*/
    vm_args.version = JNI_VERSION_1_2;
    vm_args.nOptions = 3;
    vm_args.options = options;
    vm_args.ignoreUnrecognized = JNI_TRUE;
    res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
    if (res < 0) {
      fprintf(stderr, "Can't create Java VMn");
      //exit(1);
    }

    //建立一个类及其方法，调用其方法的过程。和前面操作 java 的方法完全一样哦！
    cls = (*env)->FindClass(env, "Prog");
    if (cls == NULL) {          goto destroy;    }
    mid = (*env)->GetStaticMethodID(env, cls, "main","([Ljava/lang/String;)V");
    if (mid == NULL) {          goto destroy;    }
    jstr = (*env)->NewStringUTF(env, " This Method is called from C!");
    if (jstr == NULL) {         goto destroy;    }
    strcls = (*env)->FindClass(env, "java/lang/String");
    args = (*env)->NewObjectArray(env, 1, strcls, jstr);
    if (args == NULL) {         goto destroy;    }
    (*env)->CallStaticVoidMethod(env, cls, mid, args);
destroy:
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*jvm)->DestroyJavaVM(jvm);
    fprintf(stdout, "Java VM destory.n");
}
```

三、执行结果

以下是执行结果的主要部分！

```
[Dynamic-linking native method java.lang.Float.intBitsToFloat ... JNI]
[Dynamic-linking native method java.lang.Double.longBitsToDouble ... JNI]
[Dynamic-linking native method java.lang.Float.floatToIntBits ... JNI]
[Dynamic-linking native method java.lang.Double.doubleToLongBits ... JNI]
[Dynamic-linking native method java.lang.Object.registerNatives ... JNI]
[Registering JNI native method java.lang.Object.hashCode]
[Registering JNI native method java.lang.Object.wait]
    ……
[Registering JNI native method sun.misc.Unsafe.getShortVolatile]
[Registering JNI native method sun.misc.Unsafe.staticFieldOffset]
```

……
[Dynamic-linking native method java.io.FileOutputStream.writeBytes ... JNI]

Hello World This Method is called from C!
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Java VM destory.

后面 3 个分别是 C 中的输出、main 方法的输入－－调用 aaa XXXX > 11.txt 和 c 中最后的输出！
执行此 exe 程序，必须将 Prog.class 放到同一个目录中！