利用 JNI 技术实现 Java 对系统进程的查询

Access system process in Java by JNI technology

马俊飞　（山东大学计算机系 250001）

摘要：本文利用 JNI 技术使 Java 与 C/C++的 DLL 进行信息通信和调用，实现了在 JAVA 中对系统进程的访问。

关键词：JAVA　JNI　Native　DLL　本地代码

1　JNI 技术简介

JNI（Java Native Interface）是 JAVA 的一种 Native 接口技术，它将 JAVA 代码与本地代码连接起来，是双向的，如同桥梁和纽带。在这里主要介绍 JAVA 与 C/C++的通信和调用。

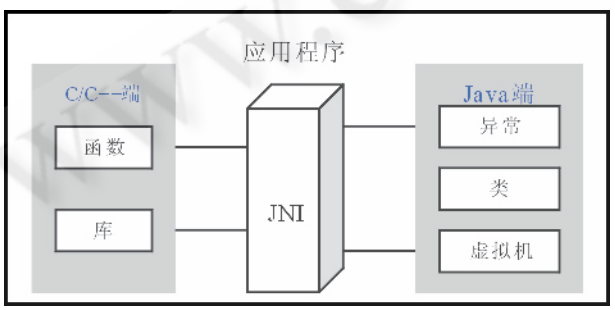

图 1

1.1　JNI 支持两种本地代码

（1）本地共享库。利用 JNI 可以编写本地方法，使得 JAVA 应用程序可以调用在本地库中实现的各种函数，包括 C 程序或 C++类等。对本地方法的调用就像调用由 JAVA 实现的方法一样，只不过是由其他语言编写，驻留在本地库中。

（2）本地应用。JNI 支持 Invocation（调用）接口，使得 JAVA 虚拟机可以被嵌入 C/C++本地应用程序中。

1.2　JAVA 与本地代码之间基本数据的类型映射

利用 JNI 设计应用程序中，必然涉及到 JAVA 与本地代码之间的信息传递。就有一个数据类型的映射问题。

JAVA 语言中基本数据类型的储存长度与具体的平台系统无关，但 C 语言则不然。如：int 类型在 JAVA 中总是 32 位的整数，而在 C 中也有 int 的类型，但其储存与系统平台有关，有的系统中为 16 位，而另外一些系统中则是 32 位。为了解决这种类型的匹配和平台问题，JNI 在头文件 jni.h 中为 C/C++定义了相应的数据类型映射关系如下表：

Java 语言类型	C/C++语言类型	字节数（位数）
boolean	jboolean	1（无符号 8 位）
byte	jbyte	1（有符号 8 位）
char	jchar	2（无符号 16 位）
short	jshort	2（有符号 16 位）
int	jint	4（有符号 32 位）
long	jlong	8（有符号 64 位）
float	jfloat	4
double	jdouble	8
void	jvoid	N/a

1.3　JAVA 与本地代码之间引用类型的映射

所有的 JAVA 对象都是通过引用来传递的。为了在 C/C++代码中存取 JAVA 对象，JNI 定义了相应的引用类型，JNI 就是通过这种引用形式将 JAVA 对象传递给本地方法。而这种引用是 Opaque（不透明的）引用，其指向 JAVA 虚拟机内部数据结构的 C 指针类型，其具体的数据结构对编程者是不透明的。最基本的是 jobject，在此基础上引伸出一系列子类型。

Java 端	C 端
All object	jobject
java. lang. Class object	jclass
java. lang. String object	jstring
(arrays)	jarray
Object[ ]	jobjectArray
boolean[ ]	jbooleanArray
byte[ ]	jbyteArray
char[ ]	jcharArray
short[ ]	jshortaArray
int[ ]	jintArray
long[ ]	jlongArray
float[ ]	jfloatArray
double[ ]	jdoubleArray
java. lang. Throwable objects	jthrowable

但是在用 C/C + + 来实现本地方法时,并不能对所定义的 Opaque 引用类型进行直接存取,还必须借助 JNI 提供的专门函数,这些函数通过 JNIEnv 接口指针来定位(后面将对用到的函数做具体介绍)。

1.4 JNI 的编程步骤

书写 JNI 编程必须按照图 2 的步骤来进行。

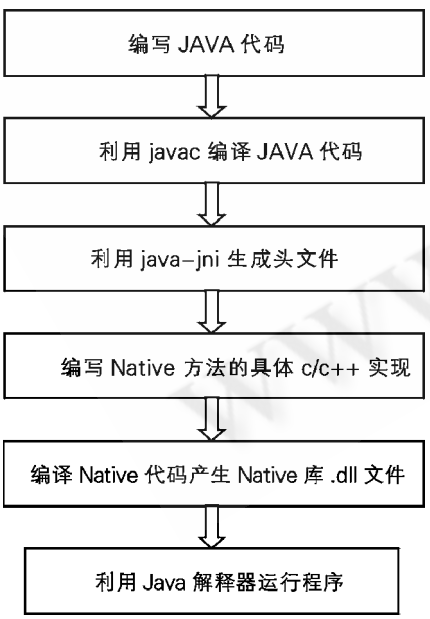

图 2

2 查询系统进程的解决方案

2.1 进程简介

进程是操作系统中的一个重要概念,进程是一个开始执行但是还没有结束程序的实例,是可执行文件的具体实现。当应用程序被系统调用到内存后,系统会给程序分配一定的资源(内存、设备等),然后进行一系列的复杂操作,使应用程序变成进程以供系统调用。系统中只有进程而没有应用程序。

为了区分各个不同的进程,系统给每个进程分配了一个 ID(如同身份证)以便识别。每个进程分为新建、运行、阻塞、就绪、完成 5 个状态。因此,实时监控系统中各个应用程序所对应的进程,对于计算机用户来说尤其重要。有些恶意的程序是无法从任务栏和任务管理器看到的。本例程序是在 JAVA 环境下显示系统当前运行的所有程序。

2.2 程序设计方案

(1) 由于 JAVA 语言的特性之一是可移植性, JAVA 本身不能很好的支持对系统进程的访问,而本地代码(C/C + +)可以很好的实现对系统进程的访问。因此在 JAVA 中只能通过 JNI(Java Native Interface)本地接口技术来实现与本地代码(C/C + +)的 DLL(动态连接库)信息通信,由本地代码(C/C + +)去访问系统的资源,以实现在 JAVA 环境中对系统进程的访问。

查询系统进程,在本地代码(C/C + +)中需要调用函数:CreateToolhelp32Snapshot( )用来获得某一时刻系统的进程、堆、模块或线程得到快照信息。CreateToolhelp32Snapshot( )函数包含在 ToolHelp32 函数组。ToolHelp32 函数组是一组寄存在 Kernel32. dll 中的 Windows API 函数,它能通过快照(Snapshot)获得驻留在系统内存中的进程表、线程表、模块表和堆表,并提供函数来枚举系统中的进程、线程、模块信息。该函数组中常用的函数包括 CreateToolhelp32Snapshot( )、ProcessFirst( )和 ProcessNext( ),用来获取所有进程。在 VC + +6.0 下,通过新建一个 Win32 Dynamic - link Library 工程,在程序中将利用这些函数获得系统的进程。

```
.....
HANDLE snapshot;
//获得某一时刻系统进程、堆、模块或线程的快照信息
```

```
snapshot = CreateToolhelp32Snapshot ( TH32CS _ SNAPPRO-
CESS,0 ) ;
    ……
PROCESSENTRY32 processListStr;
jboolean return_value;
processListStr. dwSize = sizeof( PROCESSENTRY32 ) ;
//获的系统进程链表中下一个进程的信息
return _ value = Process32Next ( ( HANDLE ) snapshot,
&processListStr) ;
……
```

（2）为了实现 JAVA 对本地代码（C/C ++）所编写的 DLL（动态连接库）的调用，在 JAVA 中必须先编写所需要的 JAVA 类，命名为：systemProcess. java

```
public class systemProcess
{
static
{
    //调用本地代码编写 DLL( 动态连接库) ,process 为 C/C
++生成的 DLL 文件名
    System. loadLibrary( " process" ) ;
}
//声明所需要的本地方法
public static native int processHandle( ) throws Exception; //
获取进程快照表的快照信息
public static native boolean processData( int snapshot,dataP-
rocess ds) throws Exception;
    //用来获取其他进程
}
```

在这里，必须声明所要用的本地方法。声明时要带关键词“native”，以指明该方法不是一个常规方法，而是一个本地方法。

在实际调用本地方法之前，要先装载实现该方法的本地库。是在静态初始化器（static initializer）中，由调用 System. loadLibrary 方法来实现，Java 虚拟机在调用该 Java 类中的任何方法之前将自动调用静态初始化器来装载 native 共享库。

然后，用 javac 编译 systemProcess. java 得到相应的 . class 文件。再利用命令 javah 把所生成的 . class 文件生成一个 . h 的 JNI 类型的头文件，用来由本地代码（C/C ++）调用。JNI 要求在每个 native 方法的实现中至少带两个参数。在这里第一个参数是 JNIEnv 接口指针，正是它使 native 代码可以对从 Java 引用程序中传来的参数和对象进行存取；第二个参数是 jclass（或 jobject），是对当前对象本身的引用。

注意，使用命令 javah 时，后面的 class 文件不带后缀。生成一个和 Java 类名相同的头文件，在这里为 systemProcess. h，它为本地方法的实现提供了一个 C/C ++ 函数署名（Signature），是不能修改的。

（3）由于 JAVA 与本地代码是两种不同的编译环境，所以信息传递的方式不同。在 JAVA 中字符串是 16 位的 Unicode 字符序列，而在 C/C ++ 中是 8 位以 Null（0）结尾的字符数组。JAVA 字符串 String 是一种对象，在 Native 方法中对它的使用要通过 Opaque 引用 jstring。但是 jstring 与常规的 C/C ++ 字符串（char *）类型不同，Java 中所以不能直接操作，必须由 JNI 提供函数来支持 Unicode 和 UTF－8 字符的转换。

下表是两种不同字符串的转换函数：

Const jbyte * GetStringUTFChars (JNIEnv *,jstring,jboolean *) void ReleaseStringUTFChars (JNIEnv *,jstring,const jbyte *)	获取或释放一个指向 UTF－8 格式字符串内容的指针，可以返回字符串的一个复制这两个函数必须对应出现，否则容易造成内存泄漏
Jstring NewStringUTF (JNIEnv *,const char *)	根据 Unicode 格式字符串创建一个 Java String 实例，不能创建返回 null

```
……
PROCESSENTRY32 processListStr;
jboolean return_value;
processListStr. dwSize = sizeof( PROCESSENTRY32 ) ;
//获的系统进程链表中下一个进程的信息
return _ value = Process32Next ( ( HANDLE ) snapshot,
&processListStr) ;
……..
//创建一个 java String 实例
    ( env) - > NewStringUTF( processListStr. szExeFile)
……
```

（4）Java 语言中有两种成员变量，对象成员（instance field）和静态成员（static field）。前者是实例化之后的变量，后者是一个类所拥有的，不依具体的对象而变。

为了从本地代码中获得所返回的数据，在 java 中定义了所调用和存取的对象成员变量。命名为：dataProcess. java

```
class dataProcess
```

```
{
String fileName;
int PProcessID;
int currentThread;
int classBase;
//在 java 中设置从本地代码中返回的数据
public void setName(String name) //设置返回的进程文件名
{ fileName = name;}
public String getName()
{ return fileName;}
public void setPProcessID(int PID) //设置返回的进程 PID
{ PProcessID = PID;}
public int getPProcessID()
{ return PProcessID; }
public void setThread(int thread) //设置返回的进程开启线程数
{ currentThread = thread; }
public int getThread()
{ return currentThread; }
public void setClassBase(int base) //设置返回的进程优先级
{classBase = base; }
public int getClassBase()
{ return classBase; }
}
```

要在本地代码中获取或设定 Java 对象成员的变量必须按以下步骤做:

① 首先要得到所要引用或设定对象成员所属的类,调用 JNI 函数:

```
jclass jcls = (env) -> GetObjectClass (dataProcess); //在本地代码中获取 JAVA 中对象所属的类
```

② 根据所得的类、JAVA 对象方法的名称及其署名,得到对象方法的类型为 jmethodID 的标识号(ID):

```
//在本地代码中获取 JAVA 类中的方法的标识号
jmethodID jmid = (env) -> GetMethodID (jcls, (类) "setName", (方法名) "(Ljava/lang/String;)V"); (方法签名)
```

③ 然后通过 Call < type > Method 方法,传递所需要的参数。

```
//在本地代码中设定 JAVA 对象成员的变量
(env) -> CallVoidMethod (dataProcess, (方法名) jmid, (方法标识号) (env) -> NewStringUTF(processListStr.szExeFile)); (从本地代码返回的数据)
```

注 type: Object, Boolean, Byte, Char, Short, Int, Long, Float, Double, Void

要想在本地方法中对 JAVA 成员变量进行存取,或对 JAVA 方法调用,需要用到类型的署名。对于基本数据类型,JNI 用一个大写字母作为署名,而一个类的类型变量的署名以大写的 L 开头,后接完整的类名(包括包名),并用斜杠"/"代替逗号,最后以分号结束。

(5) 通过以上工作,可以由 JAVA 调用的 DLL 就可以在本地语言 VC + +6.0 下编写成功,对于具体的如何编写一个 DLL,在本文中不作详细说明。

3 结束语

JNI 技术为解决 JAVA 访问系统资源等硬件设施、提高 JAVA 运行速度以及与其他语言进行通信提供了一个很好的途径。但是由于使用了本地代码,此时的 JAVA 程序不再具有跨平台性,虽然用 JAVA 编写的那部分代码仍具有可移植性,而本地代码却要重新修改和编译。但 JNI 技术的应用,保证了 JAVA 仍具有类型安全性,而本地代码却不然,因此用 JNI 开发应用程序要格外小心,否则容易造成整个应用程序的崩溃。

参考文献

1 《最新 Java2 核心技术 卷Ⅱ:高级性能》,机械工业出版社,2003.1。

2 《Java 高级实用编程》,清华大学出版社,2004.1。

3 《新编 Windows API 参考大全》,电子工业出版社,2001.4。

4 java.sun.com/docs/books/tutorial/native1.1/

5 www.javaworld.com/javaworld/jw-10-1999/jw-10-jni.html

6 www-106.ibm.com/developerworks/edu/j-dw-javajni-i.html