JNA-JNI 升级版

一、概述

JNA(Java Native Acess)框架是一个开源的 Java 框架,是 SUN 公司主导开发的,建立在经典的 JNI 的基础之上的一个框架， 功能强大,易用,类似于.NET 的 P/Invoke. 我们知道，使用 JNI 调用.dll/.so 共享类库是非常麻烦和痛苦的.如果有一个现有的.dll/.so 文件，如果使用 JNI 技术调用，我们首先需要另外使用 C 语言写一个.dll/.so 共享库，使用 SUN 规定的数据结构替代 C 语言的数据结构,调用已有的 dll/so 中公布的函数.然后再在 Java 中载入这个适配器 dll/so，再编写 Java Native 函数作为 dll 函数中的代理. 经过 2 个繁琐步骤才能在 Java 中调用本地代码. 因此，很少有 Java 程序员愿意编写 dll/so 库中的原生函数的 Java 程序，这也使 Java 语言在客户端上乏善可陈. 可以说 JNI 是 Java 的一大弱点!

二、P/Invoke 和 JNA

而在.NET 平台上,强大的 P/Invoke 技术使我们 Java 程序员非常羡慕. 使用 P/Inovke 技术，只需要使用编写一个.NET 函数,再加上一个声明的标注，就可以直接调用 dll 中的函数. 不需要你再使用 C 语言编写 dll 来适配.

现在，不需要再羡慕.NET 的 P/Invoke 机制了, JNA 把对 dll/so 共享库的调用减少到了和 P/Invoke 相同的程度. 使用 JNA，不需要再编写适配用的.dll/.so，只需要在 Java 中编写一个接口和一些代码，作为.dll/.so 的代理，就可以在 Java 程序中调用 dll/so.

三、快速启动：

下载一个 jar 包,就可以使用 JNA 的强大功能方便地调用动态链接库中的 C 函数.
示例 1:

```
import com.sun.jna.Library;
import com.sun.jna.Native;
import com.sun.jna.Platform;

public class HelloWorld {

    public interface CLibrary extends Library {
        String libraryName = Platform.isWindows() ? "msvcrt" : "c";
        CLibrary INSTANCE = (CLibrary)Native.loadLibrary(libraryName, CLibrary.class);
        void printf(String format, Object... args);
    }

    public static void main(String[] args) {
        CLibrary.INSTANCE.printf("Hello World\n");
```

```
        for (int i = 0; i < args.length; i++) {
            CLibrary.INSTANCE.printf("Argument %d : %s\n", i, args[i]);
        }
    }
}
```

执行以后，可以看到输出内容

```
D:\MyStudy\Java\mycode>java -cp jna.jar;. HelloWorld test adfa lkj
Hello World
Argument 0 : test
Argument 1 : adfa
Argument 2 : lkj
```

这里,程序的打印输出实际上是使用了 msvcrt.dll 这个 C 运行时库中的 printf 函数打印出上面的内容，不需要写 C 代码就可以直接在 Java 中调用外部动态链接库中函数了!

示例 2:

1，在 VS 中选择 C++语言，然后选择创建一个 Win32 程序。 选择 dll 类型。

2，发布的 C 函数是：

```
#define  MYLIBAPI extern "C" __declspec(dllexport)
MYLIBAPI  void  say(wchar_t* pValue);
```

//这个函数的实现是：

```
void say(wchar_t* pValue){
        std::wcout.imbue(std::locale("chs"));
        std::wcout<<L"DLL 程序说：“<<pValue<<std::endl;
}
```

// 它需要传入一个 Unicode 编码的字符数组。然后在控制台上打印出一段中文字符。

3，生成 dll。然后把生成的 dll 文件复制到 Eclipse 项目中，放在项目下面。

4，在 Eclipse 中编写以下代码：

```
import com.sun.jna.Library;
import com.sun.jna.Native;
import com.sun.jna.WString;
```

```
public class TestDll1Service {

    public interface TestDll1 extends Library {
        /**
         * 当前路径是在项目下，而不是 bin 输出目录下。
         */
        TestDll1 INSTANCE = (TestDll1)Native.loadLibrary("TestDll1", TestDll1.class);
        public void say(WString value);

    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        TestDll1.INSTANCE.say(new WString("这段字符串来自 JAVA 程序!"));
        System.out.println("这段话是由 Java 直接打印出来的.");
    }

}
```

5，执行这个 Java 类。可以看到控制台下如下输出：

```
DLL 程序说：这段字符串来自 JAVA 程序!
这段话是由 Java 直接打印出来的.
```

6，上面一行是 C 语言使用 C++的 std::wcout 输出的。
下面一行是 Java 语言输出的。

四、JNA 技术解密：

JNA 是建立在 JNI 技术基础之上的一个 Java 类库，它使您可以方便地使用 java 直接访问动态链接库中的函数。

原来使用 JNI，你必须手工用 C 写一个动态链接库，在 C 语言中映射 Java 的数据类型。

JNA 中，它提供了一个动态的 C 语言编写的转发器，可以自动实现 Java 和 C 的数据类型映射。你不再需要编写 C 动态链接库。

当然，这也意味着，使用 JNA 技术比使用 JNI 技术调用动态链接库会有些微的性能损失。可能速度会降低几倍。但影响不大。

Java—C 和操作系统数据类型的对应表

Java Type	**C Type**	**Native Representation**
boolean	int	32-bit integer (customizable)
byte	char	8-bit integer
char	wchar_t	platform-dependent
short	short	16-bit integer
int	int	32-bit integer
long	long long, __int64	64-bit integer
float	float	32-bit floating point
double	double	64-bit floating point
Buffer Pointer	pointer	platform-dependent (32- or 64-bit pointer to memory)
<T>[] (array of primitive type)	pointer array	32- or 64-bit pointer to memory (argument/return) contiguous memory (struct member)
除了上面的类型，JNA 还支持常见的数据类型的映射。		
String	char*	NUL-terminated array (native encoding or jna.encoding)
WString	wchar_t*	NUL-terminated array (unicode)
String[]	char**	NULL-terminated array of C strings

WString[]	wchar_t**	NULL-terminated array of wide C strings
Structure	struct* struct	pointer to struct (argument or return) (or explicitly) struct by value (member of struct) (or explicitly)
Union	union	same as Structure
Structure[]	struct[]	array of structs, contiguous in memory
Callback	<T> (*fp)()	function pointer (Java or native)
NativeMapped	varies	depends on definition
NativeLong	long	platform-dependent (32- or 64-bit integer)
PointerType	pointer	same as Pointer

五、JNA 编程过程：

JNA 把一个 dll/.so 文件看做是一个 Java 接口。

Dll 是 C 函数的集合、容器，这正和接口的概念吻合。

我们定义这样一个接口，

```
public interface TestDll1 extends Library {
        /**
         * 当前路径是在项目下，而不是 bin 输出目录下。
         */
        TestDll1 INSTANCE = (TestDll1)Native.loadLibrary("TestDll1", TestDll1.class);
        public void say(WString value);
}
```

如果 dll 是以 stdcall 方式输出函数，那么就继承 StdCallLibrary。否则就继承默认的 Library 接口。

接口内部需要一个公共静态常量：instance。

```
TestDll1 INSTANCE = (TestDll1)Native.loadLibrary("TestDll1", TestDll1.class);
```

通过这个常量，就可以获得这个接口的实例，从而使用接口的方法。也就是调用外部 dll 的函数！

注意：

一，Native.loadLibrary()函数有 2 个参数：

1，dll 或者.so 文件的名字，但不带后缀名。这符合 JNI 的规范，因为带了后缀名就不可以跨操作系统平台了。

搜索 dll 的路径是：

1）项目的根路径

2）操作系统的全局路径、

3）path 指定的路径。

2，第二个参数是本接口的 Class 类型。

JNA 通过这个 Class 类型，根据指定的 dll/.so 文件，动态创建接口的实例

二，接口中你只需要定义你需要的函数或者公共变量，不需要的可以不定义。

```
public void say(WString value);
```

参数和返回值的类型，应该和 dll 中的 C 函数的类型一致。

这是 JNA，甚至所有跨平台调用的难点。

这里，C 语言的函数参数是：wchar_t*。

JNA 中对应的 Java 类型是 WStirng。

有过跨语言、跨平台开发的程序员都知道，跨平台、语言调用的难点，就是不同语言之间数据类型不一致造成的问题。绝大部分跨平台调用的失败，都是这个问题造成的。

关于这一点，不论何种语言，何种技术方案，都无法解决这个问题。

这需要程序员的仔细开发和设计。这是程序员的责任。

常见的跨平台调用有：

1，Java 调用 C 语言编写的 dll、.so 动态链接库中的函数。

2，.NET 通过 P/Invoke 调用 C 语言编写的 dll、.so 动态链接库中的函数。

3，通过 WEBService，在 C,C++,Java,.NET 等种种语言间调用。

WebService 传递的是 xml 格式的数据。

即使是强大的 P/Invoke 或者 WebService，在遇到复杂的数据类型和大数据量的传递时，还是会碰到很大的困难。

因为，一种语言的复杂的数据类型，很难用另一种语言来表示。这就是跨平台调用问题的本质。

如，WEBService 调用中，很多语言，如 Java，.NET 都有自动实现的 Java/.NET 类型和 XML 类型之间的映射的类库或者工具。

但是，在现实的编程环境中，如果类型非常复杂，那么这些自动转换工具常常力不从心。要么 Object-XML 映射错误。要么映射掉大量的内存。

如果要使用 WEBService，更为简单直接的做法是手工使用 xml 处理工具提取 xml 中的数据构建对象。或者反过来，手工根据 Object 中的属性值构建 xml 数据。

Java 和 C 语言之间的调用问题，也是如此。

Java 要调用 C 语言的函数，那么就必须严格按照 C 语言要求的内存数量提供 Java 格式的数据。要用 Java 的数据类型完美模拟 C 语言的数据类型。

JNA 已经提供了大量的类型匹配 C 语言的数据类型。

跨平台、跨语言调用的第一原则：就是尽量使用基本、简单的数据类型，尽量少跨语言、平台传递数据！

如果在程序中，有复杂的数据类型和庞大的跨平台数据传递。那么必须另外写一些 Façade 接口，把需要传递的数据类型简化，把需要传递的数据量简化。

否则，不论是实现的难度还是程序的性能都很难提高。

六、JNI 怎么办?

JNI 还是不能废

我们已经见识了 JNA 的强大。JNI 和它相比是多么的简陋啊！

但是，有些需求还是必须求助于 JNI。

JNA 是建立在 JNI 技术基础之上的一个框架。

使用 JNI 技术，不仅可以实现 Java 访问 C 函数，也可以实现 C 语言调用 Java 代码。

而 JNA 只能实现 Java 访问 C 函数，作为一个 Java 框架，自然不能实现 C 语言调用 Java 代码。此时，你还是需要使用 JNI 技术。

JNI 是 JNA 的基础。是 Java 和 C 互操作的技术基础。