开源项目：Java Native Access

用更容易的方法来调用本地代码

By Jeff Friesen, JavaWorld.com, 02/05/08

原文地址：

http://www.javaworld.com/javaworld/jw-02-2008/jw-02-opensourcejava-jna.html

如果在 Java 程序中你使用 Java Native Interface(JNI) 来调用某个特定平台下的本地库文件，你就会发现这个过程很单调、乏味。Jeff Friesen 一直在介绍一个知名度很低的 Java 开源项目：Java Native Access---它能够避免因使用 JNI 导致的错误和乏味，同时它还能让你通过编程的方式调用 C 语言库。

在 Java 语言没有提供必要的 APIs 的情况下，Java 程序使用 Java Native Interface (JNI)来调用特定平台下的本地库是必要的。例如：在 Windows XP 平台中，我使用过 JNI 来调用通用串行总线和基于 TWAIN 的扫描仪器的库；在更古老的 Windows NT 平台中，调用过智能卡的库。

我按照一个基本的、乏味的流程来解决这些问题：首先，我创建一个 Java 类用来载入 JNI-friendly 库（这个库能过访问其他的库）并且声明这个类的本地方法。然后，在使用 JDK 中的 javah 工具为 JNI-friendly 库中的函数---函数和这个类中的本地方法一一对应---创建一个代理。最后，我使用 C 语言写了一个库并用 C 编译器编译了这些代码。

尽管完成这些流程并不是很困难，但是写 C 代码是一个很缓慢的过程---例如： C 语言中的字符串处理是通过指针来实现的，这会很复杂的。而且，使用 JNI 很容易出现错误，导致内存泄漏、很难找到程序崩溃的原因。

在 Java 开源系列的第二篇文章中，我要介绍一个更简单、更安全的解决方法：Todd Fast and Timothy Wall 的 Java Native Access (JNA) 项目。JNA 能够让你在 Java 程序中调用本地方法时避免使用 C 和 Java Native Interface。在这篇文章中，让我以简要的介绍 JNA 和运行示例必需的软件来开始下面的内容。然后，向你展示如何使用 JNA 将 3 个 Windows 本地库中的有用代码移植到 Java 程序中。

1、Get started with JNA（JNA 入门）

Java Native Access 项目 在 Java.net 上，你可以到这个网站上现在这个项目的代码和在线帮助文档。虽然在下载有 5 个相关的 jar 文件，在本文中你仅仅需要下载其中的 jna.jar 和 example.jar。

Jna.jar 提供基本的、运行这些示例文件必需的 jna 运行环境。这个 jna.jar 文件除了有 Unix、Linux、Windows 和 Mac OS X 平台相关的 JNT-friendly 本地库外，还包含其他几个类包。每一个本地库都是用来访问相对应平台下的本地方法的。

example.jar 包含了不同的示例来表明 JNA 的用途。其中的一个例子是使用 JNA 来实现一个在不同平台下的透明视窗技术的 API。在文章最后的示例中将要展示如何使用这个 API 修复上个月的文章关于 VerifyAge2 应用中辨认透明效果的问题。

2、获取本地时间（Get local time）

如果你在 Java Native Access 首页 看过“JNA 如何入门”，你就会知道一个很简单的关于调用 Windows 平台下的 API 函数：GetSystemTime（） 的 JNA 示例。这个不完整的例子只是展示了 JNA 的基本特点。（在例子的基础上，我做了一个更完整的基于 Windows 的例子来介绍 JNA）我在 Windows 平台下完善了这个例子来介绍 JNA。

第一例子基于 Windows GetLocalTime() API 函数返回本地当前的时间和日期。和 GetSystemTime()不同的是，返回的时间/日期是协调通用时间(UTC)格式的，GetLocalTime()返回的时间/日期信息的格式是根据当前时区来表示。

在一个 Java 程序中使用 JNA 调用 GetLocalTime，你需要知道这个函数所在的 Windows 平台下的动态链接库（DLL）的名称（和可能所在的地理区域）。我们发现 GetLocalTime()和 GetSystemTime 在同一个 DLL 文件中：kernel32.dll。你还需要知道 GetLocalTime()在 C 语言环境中的申明。申明如下 Listing 1：

Listing 1. GetLocalTime 在 C 语言中的申明

```
typedef struct
{
   WORD wYear;
   WORD wMonth;
   WORD wDayOfWeek;
   WORD wDay;
   WORD wHour;
   WORD wMinute;
   WORD wSecond;
   WORD wMilliseconds;
}
SYSTEMTIME, *LPSYSTEMTIME;

VOID GetLocalTime(LPSYSTEMTIME lpst);
```

这个基于 C 语言的申明表明传到这个函数的参数数目和类型。在这个例子中，

只有一个参数---一个指向 Windows SYSTEMTIME 结构体的指针。而且，每个结构体成员的类型是 16bit 长度的无符号整型。根据这些信息，你能够创建一个完全描述 GetLocalTime()函数的接口，如 Listing 2 中所示：

Listing 2. Kernel32.java

```
// Kernel32.java

import com.sun.jna.*;
import com.sun.jna.win32.*;

public interface Kernel32 extends StdCallLibrary
{
   public static class SYSTEMTIME extends Structure
   {
      public short wYear;
      public short wMonth;
      public short wDayOfWeek;
      public short wDay;
      public short wHour;
      public short wMinute;
      public short wSecond;
      public short wMilliseconds;
   }

   void GetLocalTime (SYSTEMTIME result);
}
```

2.1、Kernel32 接口（The Kernel32 interface）

因为 JNA 使用通过一个接口来访问某个库中的函数，Listing 2 表示了一个描述 GetLocalTime()的接口。根据约定，我把接口命名为 Kernel32 是因为 GetLocalTime()在 Windows 的 kernel32.dll 库。

这个接口必须继承 com.sun..jna.Library 接口。因为 Windows API 函数遵循 stdcall 调用协议（stdcall calling convention），为 Windows API 申明的接口也必须继承 com.sun.jna.win32. StdCallLibrary 接口。因此这个接口共继承了 Library 和 com.sun.jna.win32.StdCall 两个接口。

在前面，你已经知道了 GetLocalTime() 需要一个指向 SYSTEMTIME 结构体的指针作为它唯一的参数。因为 Java 不支持指针，JNA 是通过申明一个

com.sun.jna.Structure 的子类来代替的。根据 java 文档中抽象类的概念，在参数环境中，Structure 相当于 C 语言的 struct*。

在SYSTEMTIME类中的字段和C结构体中的相对应的属性字段的顺序是一一对应的。保证字段顺序的一致性是非常重要的。例如，我发现交换 wYear 和 wMonth 会导致 wYear 和 wMonth 值互换。

每个字段在 java 中是 short integer 类型的。按照 JNA 首页上“默认类型映射”章节给出的提示，这个 short integer 分配类型是正确。然而，我们应该知道一个重要的区别：Windows 平台下的 WORD 类型等同于 C 语言环境中的 16-bit 的无符号的 short integer，而 java 中 short integer 是 16-bit 有符号的 short integer。

一个类型映射的问题

通过比较一个 API 函数返回的整型值，你会发现 Windows/C 语言的无符号整型和 Java 语言的有符号整型的 JNA 类型映射是有问题的。在比较的过程中，如果你不细心，那么错误的执行过程可能导致决定性情况。导致这种后果是因为忘记任何数值的符号位的确定是根据：在无符号整型的情况下会被解释为正号，而在有符号整型的进制中被理解为负号的。

2.2、通过 Kernel32 获取本地时间（Access the local time with Kernel32）

JNA 首页上的 GetSystemTime()示例已经表明必须使用预先申明的接口为本地库分配一个实例对象。你可以通过 com.sun.jna.Native 类中静态公用方法 loadLibrary(String name, Class interfaceClass)来完成上述的目标。Listing 3 所示：

Listing 3. LocalTime.java

```
// LocalTime.java

import com.sun.jna.*;

public class LocalTime
{
    public static void main (String [] args)
    {
        Kernel32 lib = (Kernel32) Native.loadLibrary ("kernel32",
                                                      Kernel32.class);
        Kernel32.SYSTEMTIME time = new Kernel32.SYSTEMTIME ();
        lib.GetLocalTime (time);
        System.out.println ("Year is "+time.wYear);
        System.out.println ("Month is "+time.wMonth);
        System.out.println ("Day of Week is "+time.wDayOfWeek);
        System.out.println ("Day is "+time.wDay);
        System.out.println ("Hour is "+time.wHour);
```

```
        System.out.println ("Minute is "+time.wMinute);
        System.out.println ("Second is "+time.wSecond);
        System.out.println ("Milliseconds are "+time.wMilliseconds);
    }
}
```

Listing 3 执行 Kernel32 lib = (Kernel32) Native.loadLibrary ("kernel32", Kernel32.class);来分配一个 Kernel32 实例对象并且装载 kernel32.dll。因为 kernel32.dll 是 Windows 平台下标准的 dll 文件，所以不要指定访问这个库的路径。然而，如果找不到这个 dll 文件，loadLibrary()会抛出一个 UnsatisfiedLinkError 异常。

Kernel32.SYSTEMTIME time = new Kernel32.SYSTEMTIME ();创建了一个 SYSTEMTIME 结构体的示例。初始化后下面是 lib.GetLocalTime (time);，这句话使用本地的时间/日期来给这个实例赋值。几个 System.out.println()语句是输出这些值。

2.3、编译和运行这个应用（Compile and run the application）

这部分很容易。假设 jna.jar、Kernel32.java 和 LocalTime.java 是放在当前文件夹中，调用 java –cp jna.jar;. LocalTime.java 来编译这个应用的源代码。如果在 Windows 平台下，调用 invoke java –cp jna.jar;. LocalTime 来运行这个应用。你可以得到类似与 Listing 4 的输出结果：

Listing 4. 从 LocalTime.java 生成的输出

```
Year is 2007
Month is 12
Day of Week is 3
Day is 19
Hour is 12
Minute is 35
Second is 13
Milliseconds are 156
```

3、获取操纵杆信息（Accessing joystick device info）

上面的例子已经介绍了 JNA，但是这个获取本地时间和日期的例子并没有很好的

利用这个技术，甚至也没有体现 JNI 的价值。Java 语言中的 System.currentTimeMillis()函数已经以毫秒的格式返回了这些信息。因为 Java 语言没有为游戏控制器提供 API，所以获取操纵杆的信息更适合 JNA 的使用。

例如，你要构建一个平台无关的 Java 库，而且这些库使用 JNA 调用 Linux, Mac OS X, Windwos 和 Unix 平台中本地的操纵杆 API。为了简洁和方便起见，这个例子仅仅是调用 Windows 平台下的操纵杆 API。而且我将重点介绍这个 API 很小的一部分。

类似 GetLocalTime()，第一步是辨别出操作杆 API 的 DLL，这个 DLL 是 winmm.dll，和 kernel32.dll 在同一个文件夹中，它包含了操作杆的 API 和其他的多媒体 APIs。还需知道要被使用的操作杆函数基于 C 语言的声明。这些函数声明已经在 Listing 5 中列出来了。

Listing 5. C-based declarations for some Joystick API functions

```
#define MAXPNAMELEN 32

typedef struct
{
   WORD  wMid;                  // manufacturer identifier
   WORD  wPid;                  // product identifier
   TCHAR szPname [MAXPNAMELEN]; // product name
   UINT  wXmin;                 // minimum x position
   UINT  wXmax;                 // maximum x position
   UINT  wYmin;                 // minimum y position
   UINT  wYmax;                 // maximum y position
   UINT  wZmin;                 // minimum z position
   UINT  wZmax;                 // maximum z position
   UINT  wNumButtons;           // number of buttons
   UINT  wPeriodMin;            // smallest supported polling interval
when captured
   UINT  wPeriodMax;            // largest supported polling interval
when captured
}
JOYCAPS, *LPJOYCAPS;

MMRESULT joyGetDevCaps(UINT IDDevice, LPJOYCAPS lpjc, UINT cbjc);

UINT joyGetNumDevs(VOID);
```

3.1、操作杆 API 的函数（Functions of the Joystick API）

在Windows平台下是通过以joy作为函数名开始的函数以及被各种函数调用的结构体来实现操作杆API的。例如，joyGetNumDevs()返回的是这个平台下支持的操作杆设备最多的数目；joyGetDevCaps()返回的是每个连接上的操纵杆的质量。

joyGetDevCaps()函数需要3个参数：

- 处在0到joyGetNumDevs()-1之间的操作杆ID
- 保存返回的质量信息的JOYCAPS结构体的地址
- JOYCAPS结构体的字节大小

虽然它的结果不同，这个函数返回的是一个32位的无符号整型结果，而且0表示一个已经连接的操纵杆。

JOYCAPS结构体有3种类型。Windows平台下的WORD（16位无符号短整型）类型对应的是Java语言中16位有符号短整型。除此之外，Windows下的UINT（32位无符号整型）类型是和Java语言中32位有符号整型相对应的。而Windows平台上的text character就是TCHAR类型。

微软通过TCHAR类型使开发人员能够从ASCII类型的函数参数平滑的转移到Unicode字符类型的函数参数上。而且，拥有text类型参数的函数的实现是通过宏转变为对应的ASCII或者wide-character的函数。例如，joyGetDevCaps()是一个对应joyGetDevCapsA() 和 joyGetDevCapsW()的宏。

3.2、使用 TCHAR（Working with TCHAR）

使用TCHAR和将TCHAR转变的宏会导致基于C语言的申明向基于JNA接口的转换变得有点复杂—你在使用ASCII或者wide-character版本的操纵杆函数吗？两种版本都在如下的接口中展示了：

Listing 6. WinMM.java

```
// WinMM.java

import com.sun.jna.*;
import com.sun.jna.win32.*;
```

```
public interface WinMM extends StdCallLibrary
{
   final static int JOYCAPSA_SIZE = 72;

   public static class JOYCAPSA extends Structure
   {
      public short wMid;
      public short wPid;
      public byte szPname [] = new byte [32];
      public int wXmin;
      public int wXmax;
      public int wYmin;
      public int wYmax;
      public int wZmin;
      public int wZmax;
      public int wNumButtons;
      public int wPeriodMin;
      public int wPeriodMax;
   }

   int joyGetDevCapsA (int id, JOYCAPSA caps, int size);

   final static int JOYCAPSW_SIZE = 104;

   public static class JOYCAPSW extends Structure
   {
      public short wMid;
      public short wPid;
      public char szPname [] = new char [32];
      public int wXmin;
      public int wXmax;
      public int wYmin;
      public int wYmax;
      public int wZmin;
      public int wZmax;
      public int wNumButtons;
      public int wPeriodMin;
      public int wPeriodMax;
   }

   int joyGetDevCapsW (int id, JOYCAPSW caps, int size);

   int joyGetNumDevs ();
}
```

Listing 6 没有介绍 JNA 的新特性。实际上，JNA 强调了对本地库的接口命名规则。同时，还展示了如何将 TCHAR 映射到 Java 语言中的 byte 和 char 数组。最后，它揭示了以常量方式声明的结构体的大小。Listing 7 展示了当调用 joyGetDevCapsA() 和 joyGetDevCapsW()时如何使用这些常量。

Listing 7. JoystickInfo.java

```
// JoystickInfo.java

import com.sun.jna.*;

public class JoystickInfo
{
   public static void main (String [] args)
   {
      WinMM lib = (WinMM) Native.loadLibrary ("winmm", WinMM.class);
      int numDev = lib.joyGetNumDevs ();

      System.out.println ("joyGetDevCapsA() Demo");
      System.out.println ("--------------------\n");

      WinMM.JOYCAPSA caps1 = new WinMM.JOYCAPSA ();
      for (int i = 0; i < numDev; i++)
           if (lib.joyGetDevCapsA (i, caps1, WinMM.JOYCAPSA_SIZE) == 0)
           {
               String pname = new String (caps1.szPname);
               pname = pname.substring (0, pname.indexOf ('\0'));
               System.out.println ("Device #"+i);
               System.out.println ("  wMid = "+caps1.wMid);
               System.out.println ("  wPid = "+caps1.wPid);
               System.out.println ("  szPname = "+pname);
               System.out.println ("  wXmin = "+caps1.wXmin);
               System.out.println ("  wXmax = "+caps1.wXmax);
               System.out.println ("  wYmin = "+caps1.wYmin);
               System.out.println ("  wYmax = "+caps1.wYmax);
               System.out.println ("  wZmin = "+caps1.wZmin);
               System.out.println ("  wZmax = "+caps1.wZmax);
               System.out.println ("  wNumButtons =
"+caps1.wNumButtons);
               System.out.println ("  wPeriodMin =
"+caps1.wPeriodMin);
```

```
                System.out.println ("  wPeriodMax =
"+caps1.wPeriodMax);
                System.out.println ();
            }

        System.out.println ("joyGetDevCapsW() Demo");
        System.out.println ("--------------------\n");

        WinMM.JOYCAPSW caps2 = new WinMM.JOYCAPSW ();
        for (int i = 0; i < numDev; i++)
            if (lib.joyGetDevCapsW (i, caps2, WinMM.JOYCAPSW_SIZE) == 0)
            {
                String pname = new String (caps2.szPname);
                pname = pname.substring (0, pname.indexOf ('\0'));
                System.out.println ("Device #"+i);
                System.out.println ("  wMid = "+caps2.wMid);
                System.out.println ("  wPid = "+caps2.wPid);
                System.out.println ("  szPname = "+pname);
                System.out.println ("  wXmin = "+caps2.wXmin);
                System.out.println ("  wXmax = "+caps2.wXmax);
                System.out.println ("  wYmin = "+caps2.wYmin);
                System.out.println ("  wYmax = "+caps2.wYmax);
                System.out.println ("  wZmin = "+caps2.wZmin);
                System.out.println ("  wZmax = "+caps2.wZmax);
                System.out.println ("  wNumButtons =
"+caps2.wNumButtons);
                System.out.println ("  wPeriodMin =
"+caps2.wPeriodMin);
                System.out.println ("  wPeriodMax =
"+caps2.wPeriodMax);
                System.out.println ();
            }
    }
}
```

尽管和 LocalTime 这个示例类似，JoystickInfo 执行 WinMM lib = (WinMM) Native.loadLibrary ("winmm", WinMM.class);这句话来获取一个 WinMM 的实例，并且载入 winmm.dll。它还执行 WinMM.JOYCAPSA caps1 = new WinMM.JOYCAPSA (); 和 WinMM.JOYCAPSW caps2 = new WinMM.JOYCAPSW ();初始化必需的结构体实例。

3.3、编译和运行这个程序（Compile and run the application）

假如 jna.jar,WinMM.java 和 JoystickInfo.java 在同一个文件夹中，调用 javac -cp jna.jar;. JoystickInfo.java 来编译这个应用的源代码。
在 windows 平台下，调用 java -cp jna.jar;. JoystickInfo 就可以运行这个应用程序了。如果没有操纵杆设备，你应该得到 Listing 8 中的输出。

将 C 语言中的 string 类型转换为 Java 语言的 String 类型
pname = pname.substring (0, pname.indexOf ('\0')); 这段代码将一个 C string 转换成了 Java string. 如果不使用这个转换，C 语言的 string 结束符'\0'和 string 后面的无用字符都会成为 Java 语言中 String 实例对象的内容。

Listing 8. 输出操纵杆信息（Output of JoystickInfo）

```
joyGetDevCapsA() Demo
---------------------

joyGetDevCapsW() Demo
---------------------
```

上面的输出是因为每次调用 joyGetDevCap()返回的是一个非空值，这表示没有操纵杆/游戏控制器设备或者是出现错误。为了获取更多有意思的输出，将一个设备连接到你的平台上并且再次运行 JoystickInfo。如下，将一个微软 SideWinder 即插即用游戏触摸板设备联上之后我获取了如下的输出：

Listing 9. 操纵杆连接上之后的运行结果（Output after running JoystickInfo with a joystick attached）

```
joyGetDevCapsA() Demo
---------------------

Device #0
  wMid = 1118
  wPid = 39
  szPname = Microsoft PC-joystick driver
  wXmin = 0
  wXmax = 65535
  wYmin = 0
  wYmax = 65535
  wZmin = 0
```

```
  wZmax = 65535
  wNumButtons = 6
  wPeriodMin = 10
  wPeriodMax = 1000

joyGetDevCapsW() Demo
---------------------

Device #0
  wMid = 1118
  wPid = 39
  szPname = Microsoft PC-joystick driver
  wXmin = 0
  wXmax = 65535
  wYmin = 0
  wYmax = 65535
  wZmin = 0
  wZmax = 65535
  wNumButtons = 6
  wPeriodMin = 10
  wPeriodMax = 1000
```

4、窗口透明度（Transparent windows）

在这系列文章中上篇文章是关于 Bernhard Pauler's 气泡提示（balloontip）工程的。我构建了一个叫做 VerifyAge 的、包含有一个气泡提示的 GUI 应用。Figure 1 中显示了这个 GUI 应用的一个小问题：这个气泡提示的没有经过修饰的对话框部分遮住了应用窗口的一部分边框,导致了无法点击这个边框的最小化和最大化按钮，并且使整个 GUI 很难看.

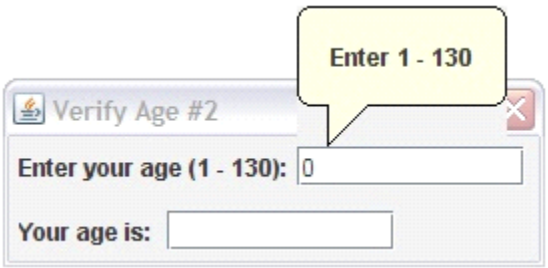

Figure 1. It is not possible to minimize the GUI when the balloon tip is displayed

尽管未修饰部分的对话框不能显示气泡提示的透明度，java 语言不支持窗口透明度。幸运的是，我们可以通过使用 com.sun.jna.examples.WindowUtils 类调用 JNA 的 examples.jar 文件来解决这个问题。

WindowUtils 提供在 Unix，Linux，Mac OS X 和 Windows 平台上使用 JNA’s 来实现窗口透明的工具方法。例如， public static void setWindowMask(final Window w, Icon mask) 让你根据像素而不是通过预定的掩罩（mask）参数来选取某部分的窗口。这个功能将在 Listing 10 中展示：

Listing 10. Using JNA to render a window transparent

```
// Create a mask for this dialog. This mask has the same shape as the
// dialog's rounded balloon tip and ensures that only the balloon tip
// part of the dialog will be visible. All other dialog pixels will
// disappear because they correspond to transparent mask pixels.

// Note: The drawing code is based on the drawing code in
// RoundedBalloonBorder.

Rectangle bounds = getBounds ();
BufferedImage bi = new BufferedImage (bounds.width, bounds.height,
                                      BufferedImage.TYPE_INT_ARGB);
Graphics g = bi.createGraphics ();
g.fillRoundRect (0, 0, bounds.width, bounds.height-VERT_OFFSET,
                 ARC_WIDTH*2, ARC_HEIGHT*2);
g.drawRoundRect (0, 0, bounds.width-1, bounds.height-VERT_OFFSET-1,
                 ARC_WIDTH*2, ARC_HEIGHT*2);
int [] xPoints = { HORZ_OFFSET, HORZ_OFFSET+VERT_OFFSET, HORZ_OFFSET };
int [] yPoints = { bounds.height-VERT_OFFSET-1,
bounds.height-VERT_OFFSET
                   -1, bounds.height-1 };
g.fillPolygon (xPoints, yPoints, 3);
g.drawLine (xPoints [0], yPoints [0], xPoints [2], yPoints [2]);
g.drawLine (xPoints [1], yPoints [1], xPoints [2], yPoints [2]);
g.dispose ();
WindowUtils.setWindowMask (this, new ImageIcon (bi));
```

在 Listing 10 中的代码段是从本文代码文档（code archive）里的加强版的 VerifyAge2 应用中的 TipFrame 的构造函数结尾部分摘录的。这个构造函数定义了围绕提示气泡的掩罩（mask）的形状，在这个形状范围里描绘不透明的像素。

假如你当前文件夹中有 examples.jar， jna.jar， 和 VerifyAge2.java，调用

javac -cp examples.jar;balloontip.jar VerifyAge2.java 来编译源文件.然后调用 java -Dsun.java2d.noddraw=true -cp examples.jar;balloontip.jar;. VerifyAge2 运行这个应用. Figure 2 展示了透明示例.

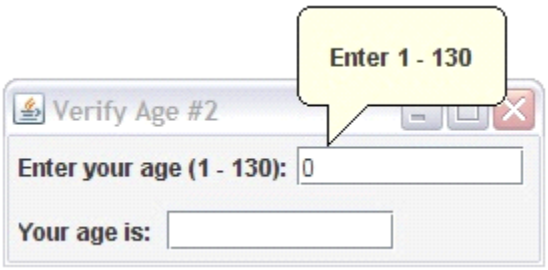

Figure 2. You can now minimize the GUI when a balloon tip appears

5、总结（In conclusion）

JNA 项目有很长的历史了（追溯到 1999 年），但是它第一次发布是在 2006 年 11 月。从此以后它慢慢的被需要将本地 C 代码整合到 Java 工程中的开发者注意到了。因为 JNA 能够用来解决 JuRuby 中常见一个问题：缺乏对 POSIX 调用的支持（lack of support for POSIX calls），它也在 JRuby 程序员中掀起些波浪。JNA 也同样被作为实现用低级 C 代码继承 Ruby 的一种解决方案（extending Ruby with low-level C code）。

我喜欢使用 JNA 来工作，相信你也会发现它比使用 JNI 来访问本地代码更简单、更安全。无需多言，JNA 还有更多的特性在本文中没有体现出来。查阅它的资源部分：获取这个开源 java 项目更多的信息（learn more about this open source Java project）。用它做 demo，而且在论坛（discussion forum ）上共享你的经验。 下一个月我会带着另一个开源项目回来的，这个开源项目会给你每天的 java 开发带来益处。

附录：WindowUtils.setWindowMask()的替代品

在刚刚写完这篇文章后，我发现 java 语言支持在 6u10 版本中支持窗口的透明和形状定制。读完 Kirill Grouchnikov 的博客后，我用 WindowUtils.setWindowMask()的替代品修改了 VerifyAge2，如下：

```
// Create and install a balloon tip shape to ensure that only this part
// of the dialog will be visible.

Rectangle bounds = getBounds ();
GeneralPath gp;
gp = new GeneralPath (new RoundRectangle2D.Double (bounds.x, bounds.y,
```

```
                                                    bounds.width,
                                                    bounds.height-
                                                    VERT_OFFSET,
                                                    ARC_WIDTH*2-1,
                                                    ARC_HEIGHT*2-1));
gp.moveTo (HORZ_OFFSET, bounds.height-VERT_OFFSET);
gp.lineTo (HORZ_OFFSET, bounds.height);
gp.lineTo (HORZ_OFFSET+VERT_OFFSET+1, bounds.height-VERT_OFFSET);
AWTUtilities.setWindowShape (this, gp);
```

这段代码使用新类 AWTUtilities（在 com.sun.awt 包中），而且 public void setWindowShape(Window w, Shape s)函数将 TipFrame 和 JDialog 窗口设置气泡形状。

作者资料

Jeff Friesen 是一名自由的软件开发人员，同时也是 Java 技术领域的的教育者。在他的 javajeff.mb.ca 网站上可以获取他发布的所有的 Java 文章和其他资料。

译者联系方式

联系方式：MSN/E-Mail:nathanlhb@hotmail.com
Blog:http://blog.csdn.net/liu251