JNI

Denne artikel beskriver JNI (Java Native Interface) som er måden hvorpå Java kan kalde native kode.

Den viser et simpelt eksempel og forklarer hvordan man bygger med forskellige compilere.

Den forudsætter kendskab til Java og C men ikke til JNI.

Skrevet den **17. Feb 2010** af **arne_v** I kategorien **Programmering / Java** | ★★★☆☆

[vigtigt: artikel http://www.eksperten.dk/guide/252 er samme artikel som denne - der gik
koks i det i forbindelse med 2 store nedbrud på Eksperten
for mange år siden - først ville jeg ikke slette en af dem af hensyn til
dem som havde "betalt" for artiklen og nu beholder jeg duplikaterne af hensyn til afgivne kommentarer]

Historie:
V1.0 - 11/04/2004 - original
V1.1 - 20/08/2005 - vis også hvordan man fra C kan kalde Java
V1.2 - 16/02/2010 - smårettelser

Baggrund

Java er platforms uafhængigt (eller Java er en platform - hvis man foretrækker den formulering). Det betyder at Java ikke har den samme support for at interface operativ systemet som f.eks. C programmer har. Det er et meget bevidst valg fra SUN's side at et Java program skal kunne køre uændret på alle platforme. Og det udelukker ting som kun kan gøres på Windows, ting som kun kan gøres på Linux etc..

Imidlertid har programmer ude i den virkelige verden sommetider behov for at bruge noget platformt specifikt. Man vil have fat i nogle oplysninger om hvilke diske der sidder i systemet. På Windows vil man have fat i registerings databasen.

Javas løsning på det problem hedder JNI. JNI (Java Native Interface) muliggør at kalde C/C++ kode fra Java kode. Så kan man kode sine operativ system specifikke funktioner i C og kalde den C kode fra Java.

Ens Java kode er naturligvis ikke længere portabelt til alle platforme hvor Java er til rådighed, men kun til de platforme hvor man har lavet den native kode som man skal bruge.

Derfor skal man når behovet for JNI opstår lige overveje om Java nu også er det rette valg af programmerings sprog. Hvis man stadigvæk mener det, så laver man en JNI løsning.

Teori

Princippet i at bruge JNI er:

1) Man laver en Java klasse med nogle metoder som har keyword native i erklæringen og ingen implementation.

2) Man compiler den .java fil og kører javah programmet på den resulterende .class fil. Programmet javah genererer en .h fil med C erklæringer af metoderne.

3) Man skriver en .c/.cpp fil som implementerer funktionerne som erklæret i den genererede .h fil.

4) Man bygger .c/.cpp filen til et modul som kan loades dynamisk. Windows: en .dll fil, Linux: en .so fil.

5) Man bruger Java klassen helt normalt i sin Java kode.

Eksempel kode

MinKlasse.java

```
package minpakke;

// klasse som indeholder de native metoder
public class MinKlasse {
    // native metoder
    public native String dup(String s);
    public native int add(int a, int b);
    // load dynamic library (Win32 .dll file/Linux .so file) når klassen
første gang bruges
    static {
        System.loadLibrary("mitlib");
    }
}
```

minpakke_MinKlasse.h (genereret af javah)

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class minpakke_MinKlasse */

#ifndef _Included_minpakke_MinKlasse
#define _Included_minpakke_MinKlasse
#ifdef __cplusplus
extern "C" {
#endif
/*
* Class:     minpakke_MinKlasse
* Method:    dup
* Signature: (Ljava/lang/String;)Ljava/lang/String;
```

```
*/
JNIEXPORT jstring JNICALL Java_minpakke_MinKlasse_dup
  (JNIEnv *, jobject, jstring);

/*
* Class:     minpakke_MinKlasse
* Method:    add
* Signature: (II)I
*/
JNIEXPORT jint JNICALL Java_minpakke_MinKlasse_add
  (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
}
#endif
#endif
```

minpakke_MinKlasse.c

```
#include <string.h>

#include <jni.h>

#include "minpakke_MinKlasse.h"

/* buffer til at returnere string resultateri (vigtigt: denne kode er ikke
thread safe) */
static char retbuf[1000];

/* funktions erklæring fra genererede .h fil */
JNIEXPORT jstring JNICALL Java_minpakke_MinKlasse_dup(JNIEnv *cntx, jobject
me, jstring s)
{
   const char *s2;
   /* konverter fra Java string til C nul termineret char array */
   s2 = (*cntx)->GetStringUTFChars(cntx,s,0);
   /* dupliker streng */
   strcpy(retbuf,s2);
   strcat(retbuf,s2);
   /* frigiv dynamisk allokeret char array */
   (*cntx)->ReleaseStringUTFChars(cntx,s,s2);
   /* konverter fra nul termineret char array til Java string og returner
denne */
   return (*cntx)->NewStringUTF(cntx,retbuf);
}

/* funktions erklæring fra genererede .h fil */
JNIEXPORT jint JNICALL Java_minpakke_MinKlasse_add(JNIEnv *cntx, jobject me,
jint a, jint b)
{
   int a2,b2;
   /* simpel assign af integers */
```

```
   a2 = a;
   b2 = b;
   /* plus og returner resultat */
   return (a2 + b2);
}
```

TestProgram.java

```
package minpakke;

// test program
public class TestProgram {
    public static void main(String[] args) throws Exception {
        // lav instans af klassen med native metoder
        MinKlasse nat = new MinKlasse();
        // kald native metoder
        System.out.println(nat.dup("abc"));
        System.out.println(nat.dup("123"));
        System.out.println(nat.add(12, 34));
        System.out.println(nat.add(56, 78));
    }
}
```

Byg og kør

Den del der kommer med java er uafhængig af platform, men hvordan man bygger et C modul som kan loades dynamisk er platform og compiler specifikt.

Jeg vil her vise kommandoerne for nogle af de mest gængse platforme/compilere.

Eksemplerne vil bruge command line build. Læserne må selv finde ud af at lave det samme i IDE'en.

Microsoft VC++

* compile med java include og java include win32 directory
* link DLL
* put dir med DLL i PATH

```
javac -classpath .. MinKlasse.java
javah -classpath .. -jni minpakke.MinKlasse
cl /c /I\sunjava\jdk1.3.1\include /I\sunjava\jdk1.3.1\include\win32
minpakke_MinKlasse.c
cl /LD minpakke_MinKlasse.obj /Femitlib.dll
javac -classpath .. TestProgram.java
path=.;%PATH%
java -classpath .. minpakke.TestProgram
```

Borland C++ Builder

* compile med java include og java include win32 directory
* link DLL med C og Windows libs
* put dir med DLL i PATH

```
javac -classpath .. MinKlasse.java
javah -classpath .. -jni minpakke.MinKlasse
bcc32 -c -I\sunjava\jdk1.3.1\include -I\sunjava\jdk1.3.1\include\win32
minpakke_MinKlasse.c
ilink32 -Tpd minpakke_MinKlasse.obj c0d32.obj,mitlib.dll,,cw32.lib import32.lib
javac -classpath .. TestProgram.java
path=.;%PATH%
java -classpath .. minpakke.TestProgram
```

GCC/mingw32

* compile med java include og java include win32 directory
* link DLL og fix calling convention
* put dir med DLL i PATH

```
javac -classpath .. MinKlasse.java
javah -classpath .. -jni minpakke.MinKlasse
gcc -c -I\sunjava\jdk1.3.1\include -I\sunjava\jdk1.3.1\include\win32
minpakke_MinKlasse.c -o minpakke_MinKlasse.obj
gcc -s -shared -Wl,--export-all,--kill-at minpakke_MinKlasse.obj -o mitlib.dll
javac -classpath .. TestProgram.java
path=.;%PATH%
java -classpath .. minpakke.TestProgram
```

GCC/Linux

* compile med java include og java include linux directory
* link so med lib prefix
* put dir med so i LD_LIBRARY_PATH

```
javac -classpath .. MinKlasse.java
javah -classpath .. -jni minpakke.MinKlasse
gcc -c -I/usr/java/jdk1.3.1/include -I/usr/java/jdk1.3.1/include/linux
minpakke_MinKlasse.c -o minpakke_MinKlasse.o
gcc -s -shared minpakke_MinKlasse.o -o libmitlib.so
javac -classpath .. TestProgram.java
LD_LIBRARY_PATH=`pwd`
export LD_LIBRARY_PATH
java -classpath .. minpakke.TestProgram
```

(erstat stier til Java med stier til din Java version på dit system)

Afsluttende kommentarer

Der er masser af ting i forbindelse med argumenterne som jeg ikke er kommet ind på her, men dette burde være nok til at komme igang.

Til andre sprog end C/C++ må man enten selv lave den rette rutine erklæring eller også bruge en C/C++ wrapper.

Bemærk at i J2EE/Java EE kontekst er det kun JCA adaptorer som bør bruge JNI.

Kalde Java fra C/C++

Det er også muligt at kalde Java kode fra et C/C++ program.

Det bruges meget sjældent. Når det bruges er det normalt som launcher af et main program skrevet i Java.

Normalt er det nok at lave en executable jar fil (en jar fil med et manifest med et Main-Class direktiv og evt. suppleret med et Class-Path diretiv). Men hvis man vil have sat specielle options til Java og man ikke kan lide tanken om en BAT fil eller hvis det skal startes som en windows service eller eller hvis man ikke vil være afhængig af at windows associationen til .jar peger korrekt på javaw, så kan man lave sin egen EXE wrapper.

Jeg vil vise 2 eksempler som illustrerer teknikken.

Console app

test\Test.java:

```
package test;

public class Test {
    public static void main(String[] args) {
        for(int i = 0; i < args.length; i++) {
            System.out.println("arg " + i + " = " + args[i]);
        }
        System.out.println("maxmem = " + Runtime.getRuntime().maxMemory());
    }
}
```

startjava.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <jni.h>

void start_java(int maxmem, char *classpath, char *mainclass,int nargs,char
**args)
{
    JavaVMOption options[2];
```

```
    JavaVMInitArgs vm_args;
    JNIEnv *env;
    JavaVM *jvm;
    jint res;
    jclass mainclassptr;
    jmethodID mainmethod;
    jobjectArray mainargs;
    jstring *mainargsptr;
    int i;
    char maxmemopt[16];
    char classpathopt[1024];
    char mainclass2[128];
    sprintf(maxmemopt,"-Xmx%dm",maxmem);
    sprintf(classpathopt,"-Djava.class.path=%s",classpath);
    options[0].optionString = maxmemopt;
    options[1].optionString = classpathopt;
    vm_args.version = JNI_VERSION_1_4;
    vm_args.nOptions = 2;
    vm_args.options = options;
    vm_args.ignoreUnrecognized = JNI_FALSE;
    res = JNI_CreateJavaVM(&jvm,(void **)&env,&vm_args);
    if(res<0)
    {
        printf("Error creating JVM\n");
        return;
    }
    strcpy(mainclass2,mainclass);
    for(i=0;i<strlen(mainclass2);i++)
    {
        if(mainclass2[i]=='.')
        {
            mainclass2[i]='/';
        }
    }
    mainclassptr = (*env)->FindClass(env,mainclass2);
    if(mainclassptr==NULL)
    {
        printf("Error finding class %s\n",mainclass);
        return;
    }
    mainmethod = (*env)->GetStaticMethodID(env,mainclassptr,"main",
"([Ljava/lang/String;)V");
    if (mainmethod==NULL)
    {
        printf("Error getting main method in class %s\n",mainclass);
        return;
    }
    mainargs =
(*env)->NewObjectArray(env,nargs,(*env)->FindClass(env,"java/lang/String"),NUL
L);
    mainargsptr = (jstring *)malloc(nargs*sizeof(jstring));
    for(i=0;i<nargs;i++)
    {
        mainargsptr[i] = (*env)->NewStringUTF(env,args[i]);
        (*env)->SetObjectArrayElement(env,mainargs,i,mainargsptr[i]);
```

```
    }
    (*env)->CallStaticVoidMethod(env,mainclassptr,mainmethod,mainargs);
    free(mainargsptr);
    (*jvm)->DestroyJavaVM(jvm);
    return;
}

int main(int argc,char *argv[])
{
    /*
     * java -Xmx128m -classpath test.jar test.Test A BB CCC
     */
    char *args[] = { "A", "BB", "CCC" };
    start_java(128, "test.jar", "test.Test", 3, args);
    return 0;
}
```

build med Microsoft VC++:

```
cd test
javac Test.java
cd ..
jar cvf test.jar test\Test.class
java -classpath test.jar test.Test X YY ZZZ
cl /I\sunjava\j2sdk1.4.2_02\include /I\sunjava\j2sdk1.4.2_02\include\win32
startjava.c /link C:\sunjava\j2sdk1.4.2_02\lib\jvm.lib
path=C:\sunjava\j2sdk1.4.2_02\jre\bin\client;%PATH%
startjava
```

Windows app

TestGUI.java:

```
import java.awt.*;

import javax.swing.*;

public class TestGUI extends JFrame {
   public TestGUI() {
      setIconImage(Toolkit.getDefaultToolkit().getImage("j.gif"));
      setTitle("Start Java");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      getContentPane().setLayout(new GridLayout(1, 1));
      getContentPane().add(new JLabel("Det virker"));
      pack();
   }
   public static void main(String[] args) {
      TestGUI f = new TestGUI();
      f.setVisible(true);
```

```
   }
}
```

startjava2.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <windows.h>
#include <tchar.h>

#include <jni.h>

#include "j.h"

void start_java(int maxmem, char *classpath, char *mainclass,int nargs,char
**args)
{
    JavaVMOption options[2];
    JavaVMInitArgs vm_args;
    JNIEnv *env;
    JavaVM *jvm;
    jint res;
    jclass mainclassptr;
    jmethodID mainmethod;
    jobjectArray mainargs;
    jstring *mainargsptr;
    int i;
    char maxmemopt[16];
    char classpathopt[1024];
    char mainclass2[128];
    sprintf(maxmemopt,"-Xmx%dm",maxmem);
    sprintf(classpathopt,"-Djava.class.path=%s",classpath);
    options[0].optionString = maxmemopt;
    options[1].optionString = classpathopt;
    vm_args.version = JNI_VERSION_1_4;
    vm_args.nOptions = 2;
    vm_args.options = options;
    vm_args.ignoreUnrecognized = JNI_FALSE;
    res = JNI_CreateJavaVM(&jvm,(void **)&env,&vm_args);
    if(res<0)
    {
        MessageBox(NULL,_T("Error creating JVM"),_T("Error"),MB_OK);
        return;
    }
    strcpy(mainclass2,mainclass);
    for(i=0;i<strlen(mainclass2);i++)
    {
        if(mainclass2[i]=='.')
        {
```

```
            mainclass2[i]='/';
        }
    }
    mainclassptr = (*env)->FindClass(env,mainclass2);
    if(mainclassptr==NULL)
    {
        MessageBox(NULL,_T("Error finding class"),_T("Error"),MB_OK);
        return;
    }
    mainmethod = (*env)->GetStaticMethodID(env,mainclassptr,"main",
"([Ljava/lang/String;)V");
    if (mainmethod==NULL)
    {
        MessageBox(NULL,_T("Error getting main method in
class"),_T("Error"),MB_OK);
        return;
    }
    mainargs =
(*env)->NewObjectArray(env,nargs,(*env)->FindClass(env,"java/lang/String"),NUL
L);
    mainargsptr = (jstring *)malloc(nargs*sizeof(jstring));
    for(i=0;i<nargs;i++)
    {
        mainargsptr[i] = (*env)->NewStringUTF(env,args[i]);
        (*env)->SetObjectArrayElement(env,mainargs,i,mainargsptr[i]);
    }
    (*env)->CallStaticVoidMethod(env,mainclassptr,mainmethod,mainargs);
    free(mainargsptr);
    (*jvm)->DestroyJavaVM(jvm);
    return;
}

int WINAPI WinMain (HINSTANCE hThisInstance,HINSTANCE hPrevInstance,LPSTR
szCmdLine,int iCmdShow)
{
   /*
     * java -Xmx128m -classpath mlf.jar MultiLookAndFeel
     */
    start_java(128, "tg.jar", "TestGUI", 0, NULL);
    return 0;
}
```

j.rc:

```
#include "j.h"

IDI_J ICON "j.ico"
```

j.h:

```
#define IDI_J 101
```

j.ico og j.gif indeholder det icon man vil bruge

build med Microsoft VC++:

```
javac TestGUI.java
jar cvf tg.jar TestGUI.class
java -classpath tg.jar TestGUI
rc j.rc
cl /I\sunjava\j2sdk1.4.2_02\include /I\sunjava\j2sdk1.4.2_02\include\win32
startjava2.c /link j.res C:\sunjava\j2sdk1.4.2_02\lib\jvm.lib user32.lib
gdi32.lib /machine:x86 /subsystem:windows
path=C:\sunjava\j2sdk1.4.2_02\jre\bin\client;%PATH%
startjava2
```

Kommentar af simonvalter d. 03. May 2004 | 1

stadigvæk god ;)

Kommentar af nitte d. 23. May 2004 | 2