Breve Tutorial sobre cómo usar JNI

Ing. Lamberto Maza Casas

2 de julio de 2015

1. Crear el código Java

Primero escribimos la clase java EjemploString

```
package u4.jni00;

public class EjemploString {

    public native String replaceString(String sourceString,
            String strToReplace, String replaceString);

    static {
        System.loadLibrary("BibliotecaString");
    }

    public static void main(String[] args) {
        EjemploString ex = new EjemploString();
        String str1 = "";
        String str2 = "";
        str1 = "Sky Black";
        str2 = ex.replaceString(str1, "Black", "Blue");
        System.out.println("La cadena antes: " + str1);
        System.out.println("La cadena despues: " + str2);
    }
}//end class EjemploString
```

2. Crear el código y las bibliotecas nativas

Para escribir el código en C++, debemos utilizar la herramienta javah (incluida en el JDK) para generar un archivo de cabecera. Este archivo de cabecera contiene los prototipos de las funciones que deben implementarse en C++. En primer lugar se compila el código java y, a continuación se ejecuta esta herramienta con el archivo class. Opcionalmente para compilar la clase EjemploString podemos utilizar la herramienta ant con un archivo build.xml como el siguiente:

```
<project name="JNI_EjemploString" basedir="." default="main">

    <property name="src.dir"     value="newpkgroot"/>
```

```
    <property name="build.dir"   value="Build"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="imagenes.dir" value="${classes.dir}/Imagenes"/>
    <property name="jar.dir"     value="${build.dir}/jar"/>

    <property name="main-class"  value="u4.jni00.EjemploString"/>

    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
<!--
<copy todir="${imagenes.dir}">
<fileset dir="${src.dir}/images"/>
</copy>
-->
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
<jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
<!--
cd Build/classes/
java -Djava.library.path=~/BibliotecaString/   u4.jni00.EjemploString
En el directorio ~/BibliotecaString/ debe estar el archivo
libBibliotecaString.so
-->
    </target>

    <target name="clean-build" depends="clean,jar"/>

    <target name="main" depends="clean,run"/>

</project>
```

Las ubicaciones de los archivos build.xml y EjemploString.java antes de compilar usando ant deben ser las siguientes:

```
.
|-- BibliotecaString
```

```
|-- build.xml
`-- newpkgroot
    `-- u4
        `-- jni00
            `-- EjemploString.java
```

Después de ejecutar el comando ant compile debemos tener algo como esto:

```
.
|-- BibliotecaString
|-- Build
|   `-- classes
|       `-- u4
|           `-- jni00
|               `-- EjemploString.class
|-- build.xml
|-- newpkgroot
    `-- u4
        `-- jni00
            `-- EjemploString.java
```

Al ejecutar javah debemos especificar el nombre de la clase (no el nombre del archivo) como primer parámetro. Despues necesitamos crear la biblioteca BibliotecaString: Si trabajamos en linux deberemos crear una biblioteca dinámica que deberá llamarse libBibliotecaString.so, y si trabajamos en Windows deberemos crear una biblioteca de enlace dinámico (DLL) que debera llamarse BibliotecaString.dll
En linux podemos usar la herramienta make con el siguiente archivo Makefile:

```
INCLUDE_DIRS:= ./BibliotecaString/include /usr/lib/jvm/java-6-openjdk/include/
SOURCE_DIRS:= ./BibliotecaString/src/
CFILES:= bibliotecastring.cpp
#ASM_FILES:= startup_ARMCM4.S
INCLUDE_FLAGS:=$(patsubst %,-I%,$(INCLUDE_DIRS))
#TCHIP=TM4C123GH6PM
#CC=arm-none-eabi-gcc
#CC=avr-gcc
#CC=gcc
CC=g++
CFLAGS=-g -Wall -fPIC

#ld -G BibliotecaString/objectfiles/*.o -o BibliotecaString/lib/libBibliotecaString.so -lc -l
#ld: BibliotecaString/objectfiles/bibliotecastring.o: relocation R_X86_64_PC32 against symbol
#ld: final link failed: Bad value
#make: *** [libBibliotecaString.so] Error 1    (2015.07.02)

#CPPFLAGS= $(INCLUDE_FLAGS) -D__AVR_LIBC_DEPRECATED_ENABLE__
CPPFLAGS= $(INCLUDE_FLAGS)
vpath %.h $(INCLUDE_DIRS)
vpath %.cpp *.o $(SOURCE_DIRS)
COBJECTS:= $(patsubst %.cpp,%.o,$(CFILES))

all: libBibliotecaString.so
##libBibliotecaString.so: bibliotecastring.cpp
#libBibliotecaString.so: $(COBJECTS)
# gcc -c -I/usr/lib/jvm/java-6-openjdk/include/ MyCJavaInterface.c -o MyCJavaInterface.o
# gcc -c -I MyCJavaInterface.c -o MyCJavaInterface.o
```

```
# ld -G MyCJavaInterface.o -o libMyCJavaInterface.so -lm -lc -lpthread
$(COBJECTS): %.o: %.cpp
mkdir -p BibliotecaString/objectfiles
$(CC) $(CFLAGS) $(CPPFLAGS) -c $< -o BibliotecaString/objectfiles/$@

libBibliotecaString.so: $(COBJECTS)
mkdir -p BibliotecaString/lib
ld -G BibliotecaString/objectfiles/*.o -o BibliotecaString/lib/$@ -lc -lpthread

.PHONY: clean rebuild
clean:
rm -frv BibliotecaString/lib/
rm -frv BibliotecaString/objectfiles/

rebuild: clean all
```

En windows podemos usar por ejemplo el IDE DevC++ en la version 5.11 (32 y 64 bits) para crear la biblioteca DLL. En el DevC++ se crea un proyecto DLL, se escribe el codigo de las funciones C/C++. y se compila para obtener la DLL. Puede que nos marque error(es) como que no encuentra algunoc archivos como jni.h. Hay que buscar en Tools (o Herramientas) las opciones del compilador para agregar directorios como

```
C:\Program Files\Java\jdk_8u45_algo\include
```

por último, para ejecutar el archivo .class de java desde la línea de comandos debemos hacer lo siguiente:

```
cd Build/classes
java -Djava.library.path=RutaANuestrabiblioteca pkgname.ClasePrincipal
```

En linux RutaANuestrabiblioteca será la ruta al archivo libBibliotecaString.so, mientras que en windows, RutaANuestrabiblioteca será la ruta al archivo bibliotecaString.dll.
Por otra parte si usamos un IDE como el Netbeans, deberemos

```
Para ejecutar en el Netbeans de debe agregar en el menu
run
   \set project configuration
                              \customize

donde dice run, en VM option escribir
-Djava.library.path=rutaparala_biblioteca.dll

o bien en linux
-Djava.library.path=rutaparala_libbiblioteca.so
```

Este HowTo esta hecho muy a prisa. Pero espero que le sea util a alguien. (2015.07.02)