Java Native Interface

Johan Östlund

(Tobias Wrigstad)

Java Native Interface

Johan Östlund

(Tobias Wrigstad)

Outline

- Duration: 2–3 hours
- Java Native Interface
  - javah
  - JNI API
- Demo
  - Interfacing with Joystick in C, using the SDL library
  - Adding support for a gamepad in a Java game, through C

Slides & code will be available for download!

Java Native Interface

- Allows Java programs to operate with apps and libs written in other languages, e.g., C, C++, assembler
- Motivation for use
  - No Java standard API support for platform specific program/hardware
  - Need to implement hot-spot in more efficient language/time-critical code
  - Interface Java with legacy application

Native Methods

```
public class Foo
{
   private native int bar();

   public int get42() { return 42; }
}
```

- Declared in the Java code
- Implemented in native language
- Called from Java code or native code
- Can call regular Java methods

Loading the Native (Dynamic) Library

```
public class Foo
{
   ...

   static { System.loadLibrary("Foo"); }
}
```

- Loaded using loadLibrary in the System API class
- Different naming conventions on different platforms
  - libFoo.so—Linux/Solaris
  - libFoo.jnilib—OS X
  - foo.dll—Windows

javah

- Generating a header file for the C library from the .class file
- The generated file includes jni.h, which defines the relevant data structures for interfacing with Java from C

The Generated Foo.h

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

#ifndef _Included_Foo
#define _Included_Foo
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Foo
 * Method:    bar
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_Foo_bar
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

The Generated Foo.h

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

#ifndef _Included_Foo
#define _Included_Foo
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Foo
 * Method:    bar
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_Foo_bar
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

JNIEnv

```
CallObjectMethod,
CallObjectMethodV,
CallObjectMethodA,
CallBooleanMethod,
CallBooleanMethodV,
CallBooleanMethodA,
CallByteMethod,
CallByteMethodV,
CallByteMethodA,
CallCharMethod,
CallCharMethodV,
CallCharMethodA,
CallShortMethod,
CallShortMethodV,
CallShortMethodA,
CallIntMethod,
CallIntMethodV,
CallIntMethodA,
CallLongMethod,
CallLongMethodV,
CallLongMethodA,
CallFloatMethod,
CallFloatMethodV,
...
```

Foo.c in Java

```
int bar() // this and global NS is implicit
{
  int v = this.get42();
  System.out.println(v);
  return v;
}
```

Foo.c in C

```
#include "Foo.h"

JNIEXPORT jint JNICALL
          Java_Foo_bar(JNIEnv* env, jobject this)
{
  /* get class of this (Foo) */
 jclass cls = (*env)->GetObjectClass(env, this);

  /* get handle to get42 method */
  /* ()I means no parameters, return int */
  jmethodID mid = (*env)->GetMethodID
                       (env, cls, "get42", "()I");

  /* check that the method actually exists */
  if (mid == 0)
    {
      return -1;
    }

  /* finally call the method */
  int v = (*env)->CallIntMethod
                                (env, this, mid, 0);

  /* print the value to the terminal */
  printf("The value: %d\n", v);
  return v;
}
```

Compiling

- Platform specifics
  - Include path to Java headers
  - Flag to GCC that you are generating a dynamic library
- On my machine
  - `gcc -dynamiclib -I/.../Headers/ -o libFoo.jnilib Foo.c`
- On your machine
  - I leave that as an exercise

“Java Strings to C\0”

```
/* Java String to null-terminated string */
char* str = (*env)->GetStringUTFChars
                        (env, javaString, 0);

/* Relase reference to the Java string */
(*env)->ReleaseStringUTFChars
                    (env, javaString, str);

/* Creating Java string from char* */
(*env)->NewStringUTF(env, str);
```

Working with Arrays

```
/* Get length of array */
jsize i = (*env)->GetArrayLength
                                    (env, array);


/* Get last object from array */
jobject o = (*jenv)->GetObjectArrayElement
                                (env, array, i-1);
```

Complex Method Calls

```
jmethodID GetMethodID(JNIEnv* e, jclass c,
                      const char *name, const char *sig);
```

- Returns handle to method ID instance or null if no such method exists
- Pass the signature in as a string to allow the Java RTE to find the method

Signature Strings

Type Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L fully-qualified-class ;	fully-qualified-class
[type	type[]
(arg-types) ret-type	method type

```
long f (int n, String s, int[] arr);

                    ==>

(ILjava/lang/String;[I)J
```

Example

```
/* boolean b = this.equals(this); */
jmethodID mid = (*env)->GetMethodID(env, cls,
            "equals", "(Ljava/lang/Object;)B");

jboolean b = (*env)->CallBooleanMethod
                            (env, this, mid, this);
```

Exceptions

- C code can throw and handle exceptions.
  - ExceptionOccurred()
  - ExceptionClear()
  - ThrowNew()
- JNI spec. says no JNI methods are “safe” until all pending exceptions have been “cleared”

Demo

Joystick through SDL

SDL—“Simple Direct Media Layer”

- Cross-platform multimedia library
  - Audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer
- Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, …

Demo

Joystick in Java

Links

- Sun's pages on JNI
  - http://java.sun.com/j2se/1.5.0/docs/guide/jni/
- SDL
  - http://www.libsdl.org/