

NDK Integration (JNI)

Lecture 5

Android Native Development Kit

25 March 2014

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Bibliography

Keywords

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

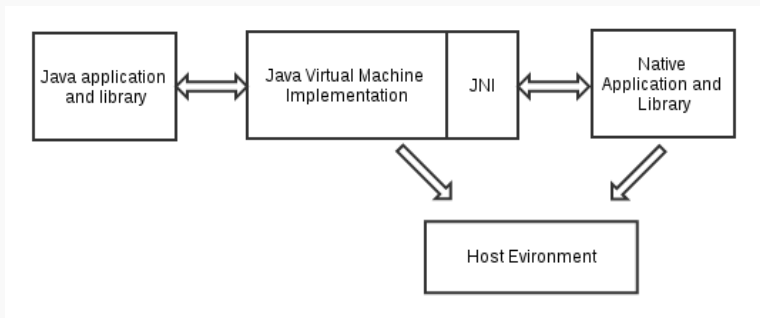
Operations on Strings

Bibliography

Keywords

- ▶ Java Native Interface (JNI)
- ▶ Native Programming Interface
- ▶ Allows Java code to interoperate with apps and libraries written in other programming languages
- ▶ When do we use JNI?
 - ▶ Java library does not support platform-dependent features
 - ▶ Use already existent library written in other language
 - ▶ Time-critical code in a low level language (performance)

- ▶ Two-way interface
- ▶ Allows Java apps to invoke native code and vice versa



- ▶ Support for two types of native code
- ▶ Native libraries
 - ▶ Call functions from native libraries
 - ▶ In the same way they call Java methods
- ▶ Native applications
 - ▶ Embed a Java VM implementation into native apps
 - ▶ Execute components written in Java
 - ▶ e.g. C web browser executes applets in an embedded Java VM implementation

- ▶ Java apps are portable
 - ▶ Runs on multiple platforms
 - ▶ The native component will not run on multiple platforms
 - ▶ Recompile the native code for the new platform
- ▶ Java is type-safe and secure
 - ▶ C/C++ are not
 - ▶ Misbehaving native code can affect the whole application
 - ▶ Security checks when invoking native code
 - ▶ Extra care when writing apps that use JNI
- ▶ Native methods in few classes
 - ▶ Clean isolation between native code and Java app

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Bibliography

Keywords

- ▶ Simple Java app calling a C function to print "Hello World"
 1. Create HelloWorld.java that declares the native method
 2. Compile HelloWorld.java using javac => HelloWorld.class
 3. Use javah -jni to create C header file (function prototype) => HelloWorld.h
 4. Write C implementation on native method => HelloWorld.c
 5. Compile C code into native library => libHelloWorld.so / HelloWorld.dll
 6. Run HelloWorld using java runtime interpreter
 - ▶ The class and native library loaded at runtime

```

class HelloWorld {
    static {
        System.loadLibrary("HelloWorld");
    }
    private native void print();
    public static void main(String [] args) {
        new HelloWorld().print();
    }
}

```

- ▶ In the static initializer we load the native library
 - ▶ The native library on disk is called `libHelloWorld.so` or `HelloWorld.dll`
- ▶ Declare the *print* native method (native modifier)
- ▶ In main we instantiate the HelloWorld class and invoke print native method

- ▶ Compile the Java source code
 - ▶ `javac HelloWorld.java`
 - ▶ Creates `HelloWorld.class` in the same directory
- ▶ Generate JNI-style header file using `javah` tool
 - ▶ `javah -jni HelloWorld`
 - ▶ Generates `HelloWorld.h`
 - ▶ Includes prototype for `Java_HelloWorld_print`
 - ▶ `JNIEXPORT void JNICALL`
`Java_HelloWorld_print (JNIEnv *, jobject);`

```

#include <jni.h>
#include <stdio.h>
#include "HelloWorld.h"
JNIEXPORT void JNICALL
Java_HelloWorld_print(JNIEnv *env, jobject obj)
{
    printf("Hello _World!\n");
    return;
}

```

- ▶ Follow the prototype in the generated header file
- ▶ `jni.h` - needed for native code to call JNI functions

- ▶ Compile C code and build native library

- ▶ Linux:

- ▶ `gcc -I/java/include -I/java/include/linux -shared -o libHelloWorld.so HelloWorld.c`

- ▶ Windows:

- ▶ `cl -IC:\java\include -IC:\java\include\win32 -MD -LD HelloWorld.c -FeHelloWorld.dll`

- ▶ Run java app
 - ▶ `java HelloWorld`
- ▶ Before, set the native library path to the current directory
 - ▶ `java.lang.UnsatisfiedLinkError: no HelloWorld in library path`
`at java.lang.Runtime.loadLibrary(Runtime.java)`
`at java.lang.System.loadLibrary(System.java)`
`at HelloWorld.main(HelloWorld.java)`
 - ▶ Linux: `LD_LIBRARY_PATH=.`
`export LD_LIBRARY_PATH`
 - ▶ Windows: library in the current dir or in a dir from PATH
- ▶ Or specify the native library path when running java
 - ▶ `java -Djava.library.path=. HelloWorld`

Standard JNI

Standard JNI Example

Native Method Arguments

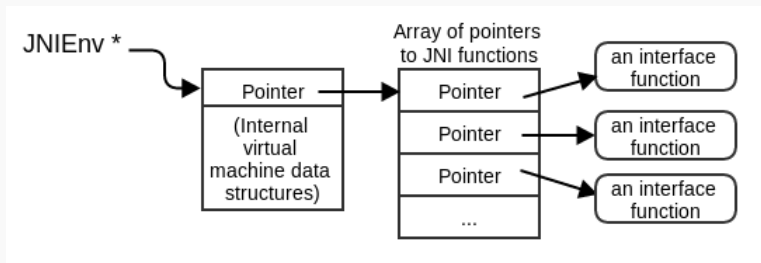
Mapping of Types

Operations on Strings

Bibliography

Keywords

- ▶ Passed into each native method call as the first argument
- ▶ Valid only in the current thread (cannot be used by other threads)
- ▶ Points to a location that contains a pointer to a function table
- ▶ Each entry in the table points to a JNI function
- ▶ Native methods access data structures in the Java VM through JNI functions



- ▶ For C and C++ source files the syntax for calling JNI functions differs
- ▶ C code
 - ▶ JNIEnv is a pointer to a JNINativeInterface structure
 - ▶ Pointer needs to be dereferenced first
 - ▶ JNI functions do not know the current JNI environment
 - ▶ JNIEnv instance should be passed as the first argument to the function call
 - ▶ e.g. `return (*env)->NewStringUTF(env, "Hello!");`
- ▶ C++ code
 - ▶ JNIEnv a C++ class
 - ▶ JNI functions exposed as member functions
 - ▶ JNI functions have access to the current JNI environment
 - ▶ e.g. `return env->NewStringUTF("Hello!");`

- ▶ Second argument depends whether the method is static or instance
- ▶ Instance methods - can be called only on a class instance
- ▶ Static methods - can be called directly from a static context
- ▶ Both can be declared as native
- ▶ For instance native method
 - ▶ Reference to the object on which the method is invoked (*this* in C++)
 - ▶ e.g. jobject `thisObject`
- ▶ For static native method
 - ▶ Reference to the class in which the method is defined
 - ▶ e.g. jclass `thisClass`

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Bibliography

Keywords

- ▶ JNI defines a set of C/C++ types corresponding to Java types
- ▶ Java types
 - ▶ Primitive types: int, float, char
 - ▶ Reference types: classes, instances, arrays
- ▶ The two types are treated differently by JNI
- ▶ int -> jint (32 bit integer)
- ▶ float -> jfloat (32 bit floating point number)

Java Type	JNI Type	C/C++ Type	Size
boolean	jboolean	unsigned char	unsigned 8 bits
byte	jbyte	char	signed 8 bits
char	jchar	unsigned short	unsigned 16 bits
short	jshort	short	signed 16 bits
int	jint	int	signed 32 bits
long	jlong	long long	signed 64 bits
float	jfloat	float	32 bits
double	jdouble	double	64 bits

- ▶ Objects -> opaque references
- ▶ C pointer to internal data structures in the Java VM
- ▶ Objects accessed using JNI functions (JNIEnv interface pointer)
 - ▶ e.g. GetStringUTFChars() function for accessing the contents of a string
- ▶ All JNI references have type jobject
 - ▶ All reference types are subtypes of jobject
 - ▶ Correspond to the most used types in Java
 - ▶ jstring, jobjectArray, etc.

Java Type	Native Type
java.lang.Class	jclass
java.lang. String	jstring
java.lang.Throwable	jthrowable
other objects	jobject
java.lang.Object[]	jobjectArray
boolean[]	jbooleanArray
byte[]	jbyteArray
char[]	jcharArray
short[]	jshortArray
int[]	jintArray
long[]	jlongArray
float[]	jfloatArray
double[]	jdoubleArray
other arrays	jarray

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Bibliography

Keywords

- ▶ String is a reference type in JNI (`jstring`)
- ▶ Cannot be used directly as native C strings
 - ▶ Need to convert the Java string references into C strings and back
 - ▶ No function to modify the contents of a Java string (immutable objects)
- ▶ JNI supports UTF-8 and UTF-16/Unicode encoded strings
 - ▶ UTF-8 compatible with 7-bit ASCII
 - ▶ UTF-8 strings terminated with `'\0'` char
 - ▶ UTF-16/Unicode - 16 bits, not zero-terminated
 - ▶ Two sets of functions
 - ▶ `jstring` is represented in Unicode in the VM

- ▶ `NewStringUTF`, `NewString`
- ▶ `jstring javaString = (*env)->NewStringUTF(env, "Hello!");`
- ▶ Takes a C string, returns a Java string reference type
- ▶ If the VM cannot allocate memory
 - ▶ Returns `NULL`
 - ▶ `OutOfMemoryError` exception thrown in the VM
 - ▶ Native code should not continue

- ▶ GetStringUTFChars, GetStringChars
- ▶ `const jbyte* str = (*env)->GetStringUTFChars(env, javaString, &isCopy);`
- ▶ `const jchar* str = (*env)->GetStringChars(env, javaString, &isCopy);`
- ▶ `isCopy`
 - ▶ `JNI_TRUE` - returned string is a copy of the chars in the original instance
 - ▶ `JNI_FALSE` - returned string is a direct pointer to the original instance (pinned object in heap)
 - ▶ Pass `NULL` if it's not important
- ▶ If the string contains only 7-bit ASCII chars you can use `printf`
- ▶ If the memory allocation fails it returns `NULL`, throws `OutOfMemory` exception

- ▶ Free memory occupied by the C string
- ▶ `ReleaseStringUTFChars`, `ReleaseStringChars`
- ▶ `(*env)->ReleaseStringUTFChars(env, javaString, str);`
- ▶ String should be released after it is used
- ▶ Avoid memory leaks
- ▶ Frees the copy or unpins the instance (copy or not)

- ▶ Get string length
 - ▶ GetStringUTFLength/GetStringLength on the jstring
 - ▶ Or strlen on the GetStringUTFChars result
- ▶ Copy string elements into a preallocated buffer
 - ▶ `(*env)->GetStringUTFRegion(env, javaString, 0, len, buffer);`
 - ▶ start index and length
 - ▶ length can be obtained with GetStringLength
 - ▶ buffer is `char[]`
 - ▶ No memory allocation, no out-of-memory checks

- ▶ GetStringCritical, ReleaseStringCritical
- ▶ Increase the probability to obtain a direct pointer to the string
- ▶ Critical Section between the calls
 - ▶ Must not make blocking operations
 - ▶ Must not allocate new objects in the Java VM
 - ▶ Disable garbage collection when holding a direct pointer to a string
 - ▶ Blocking operations or allocating objects may lead to deadlock
- ▶ No GetStringUTFCritical -> usually makes a copy of the string

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Bibliography

Keywords

- ▶ <http://www.soi.city.ac.uk/~kloukin/IN2P3/material/jni.pdf>
- ▶ <http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html>
- ▶ <http://download.java.net/jdk8/docs/technotes/guides/jni/spec/functions.html>
- ▶ <http://developer.android.com/training/articles/perf-jni.html>
- ▶ Onur Cinar, Pro Android C++ with the NDK, Chapter 3
- ▶ Sylvain Ratabouil, Android NDK, Beginner's Guide, Chapter 3

Standard JNI

Standard JNI Example

Native Method Arguments

Mapping of Types

Operations on Strings

Bibliography

Keywords

- ▶ Java Native Interface
- ▶ Two-way interface
- ▶ Native methods
- ▶ Interface pointer
- ▶ Static methods
- ▶ Instance methods
- ▶ Primitive types
- ▶ Reference types
- ▶ JNI functions
- ▶ Opaque reference
- ▶ Java string reference
- ▶ Conversion operations

To be continued..