A safari walk-through into JNI within Android™ OS

Exploiting JNI capabilities taking advantage of Android™ NDK

Antonio Troina
Matr. 708267, (antonio.troina@mail.polimi.it)

Introductory report for the M.Sc. thesis in Computer Science Engineering
Reviser: PhD. Patrick Bellasi (bellasi@elet.polimi.it)

Last update: October 29, 2012

Abstract

This article aims to briefly describe, in the form of some simple and annotated tutorials, how a developer can take advantage of some JNI capabilities under Android operating system to let the Java and Native environments communicate to each other. This operation has been made quite easy by the Android's native development kit (NDK). Special attention will be given to the callback mechanism, which is by far the most complex, nevertheless the most challenging, under the developer's point of view.

1 Introduction

The coming of Android in the smart-phones market would suggest that Java is definitely enough to rule this galaxy. Moreover, lately, the little-green-robot's operative system has approached, silently but firmly, to the embedded world which, typically, wasn't famous for being dominated by the Java language, particularly for its performance-oriented needs. That being said, who's responsible for connecting the high-level Java application layer to the native world, and the other way around? Yes, this filling is JNI, which was initially released in early 1997. With JNI, the developer can achieve two main goals: reusing his native code within a Java environment, and optimising the execution with regard to performances, so that intensive operations can run natively, instead of being interpreted, as Java pattern requires - except for the peculiar case of the *JIT-ed* code, where the bytecode is compiled *Just In Time* to run natively (this operation commonly runs at launch time, but can happen at install time, or at method invoke time).
So far, some basic examples are available (mainly on-line) which, however, are often more theoretical than practical, therefore this article is meant to be a concrete hands-on guide, to discover - some of - the secrets behind this powerful instrument which is JNI.

1.1 JNI, Android and NDK

JNI per se basically needs two components to be used: the *javah* JDK tool, which builds c-style header files from a given Java class (that will be implemented afterwards in a proper native source file, which includes the mentioned header), and the *jni.h* header file, which maps the Java types to their native counterparts. The whole flow (shown in Figure 1) mainly lies in four steps:

- implement a *Java class*, declare the methods you want to call on the native environment as `native`, and compile it
- generate the header file through the `javah -jni` command
- implement as native C/C++ code the function whose signatures have been generated during the step above
- compile the file above as a shared library, which will be loaded by the java class

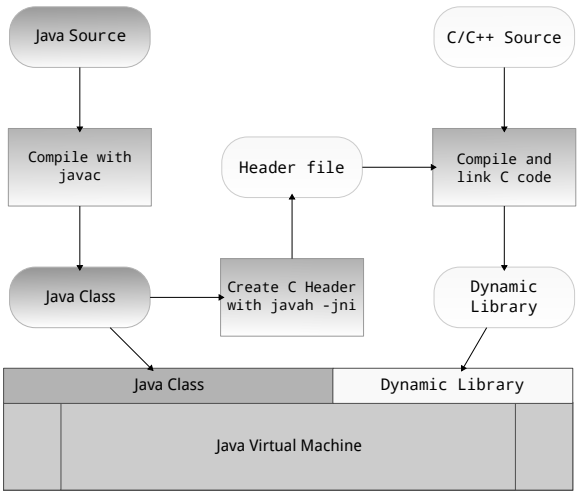

Figure 1: JNI flow

The focus of this work though points to explore JNI within the Android context, therefore our starting point will be the Android NDK, which basically consists of a ready-to-use tool-set (available on `http://developer.android.com/tools/sdk/ndk/index.html#Downloads`). The use

of the NDK condenses the steps above in just one main step, which will do almost everything at once, through the `ndk-build` command.

2 Getting started

[Environment setup - NDK, Eclipse...]

3 Tutorial1

[Tutorial1, direct call Java2C, simple example of callback]

4 Tutorial2

[Tutorial2, callback from a native thread]

5 Tutorial3

[Tutorial2, interaction Service - native thread, callback]

6 Conclusions

This report and all the source code are publicly available through the git repository at `https://github.com/thoeni/ndk-tutorials` or at `https://bitbucket.org/atroina/ndk-tutorials`.

References

[1] Liang, S.: The Java Native Interface. Addison-Wesley (1999)

[2] Gargenta, A.: Jni reference example (Oct. 2012)