What does a JVM have to do when calling a native method?

Asked 6 years, 1 month ago Active 3 years, 3 months ago Viewed 4k times

20

15

What are the usual steps that the JVM runtime has to perform when calling a Java method that is declared as `native`?

How does a HotSpot 1.8.0 JVM implement a JNI function call? What checking steps are involved (e.g. unhandled exceptions after return?), what bookkeeping has the JVM to perform (e.g. a local reference registry?), and where does the control go after the call of the native Java method? I would also appreciate it if someone could provide the entry point or important methods from the native HotSpot 1.8.0 code.

Disclaimer: I know that I can read the code myself but a prior explanation helps in quickly finding my way through the code. Additionally, I found this question worthwhile to be Google searchable. ;)

java jvm java-native-interface

asked Jul 14 '14 at 22:03

box
2,766 2 21 32

1 Answer

Active | Oldest | Votes

39

Calling a JNI method from Java is rather expensive comparing to a simple C function call. HotSpot typically performs most of the following steps to invoke a JNI method:

1. Create a stack frame.
2. Move arguments to proper register or stack locations according to ABI.
3. Wrap object references to JNI handles.
4. Obtain `JNIEnv*` and `jclass` for static methods and pass them as additional arguments.
5. Check if should call `method_entry` trace function.
6. Lock an object monitor if the method is `synchronized`.
7. Check if the native function is linked already. Function lookup and linking is performed lazily.
8. Switch thread from `in_java` to `in_native` state.
9. **Call the native function**
10. Check if safepoint is needed.
11. Return thread to `in_java` state.
12. Unlock monitor if locked.
13. Notify `method_exit`.
14. Unwrap object result and reset JNI handles block.
15. Handle JNI exceptions.

The source code for this procedure can be found at SharedRuntime::generate_native_wrapper.

As you can see, an overhead may be significant. But in many cases most of the above steps are not necessary. For example, if a native method just performs some encoding/decoding on a byte array and does not throw any exceptions nor it calls other JNI functions. For these cases HotSpot has a non-standard (and not known) convention called `Critical Natives`, discussed here.

edited Jun 7 '17 at 21:58

answered Jul 14 '14 at 23:08

apangin
70.5k 9 148 181

You left out a few. Pin the argument objects before calling the method and unpin them afterwards. Pop any frames pushed, and release any local refs created by the method after it returns. – Marquis of Lorne Jul 15 '14 at 22:54

@EJP What do you mean by pinning argument objects? HotSpot does not support object pinning in the conventional sense. No local reference is actually "released". JVM resets the whole JNI handle block at once just by two machine instructions (I've mentioned this), and any created local JNI reference automatically becomes garbage. – apangin Jul 16 '14 at 1:56