Let's Talk About Foreign Functions In Java

Deepu K Sasidharan
@deepu105 | deepu.tech
Deepu K Sasidharan

JHipster co-lead developer
Creator of KDash, JDL Studio
Developer Advocate @ Okta
OSS aficionado, author, speaker, polyglot dev

@deepu105
deepu.tech
deepu105
What is FFI
Foreign Function Interface (FFI)

- Call routines from another program regardless of the language
- Most modern languages provide this feature in intuitive ways
- Term originated from common LISP
- Most languages use C/C++ calling conventions
Why FFI

- Interact with legacy apps
- Access features not available in the language
- Use native libraries
- Access functions or programs on the host OS
- GPU and CPU offloading (Cuda, OpenCL, OpenGL, Vulcan, DirectX…)
- Multiprecision arithmetic, Matrix multiplications
- Deep learning (Tensorflow, cuDNN, Blas…)
- OpenSSL, V8, and many more
A history of FFI in Java
Java Native Interface (JNI)

- Native interface access for C/C++/Assembly
- Fastest solution in Java
- Complicated to use and brittle
- Not very secure and could cause memory safety issues
- Overhead and performance loss is possible
- Difficult to debug
- Depends on Java devs to write safe C binding code manually
Java Native Access (JNA)

- Native interface access for C/C++/Assembly
- Much simpler to use
- Dynamic binding. No need to write any C binding code
- Widely used and mature library
- Uses reflection
- Built on top of JNI
- Has performance overhead and can be slower than JNI
- Difficult to debug

https://github.com/java-native-access/jna
Java Native Runtime (JNR)

- Native interface access for C/C++/Assembly
- Easy to use
- Dynamic binding. No need to write any C binding code
- Modern API
- Comparable performance to JNI
- Built on top of JNI
- Difficult to debug

https://github.com/jnr/jnr-ffi
Project Panama

https://foojay.io/today/project-panama-for-newbies-part-1/
Foreign-Memory Access API

- Safely and efficiently access foreign memory outside of the Java heap
  - Consistent API for different types of memory
  - JVM memory safety should not be compromised
  - Explicit memory deallocation
  - Interact with different kinds of memory resources, including off-heap or native memory.
- JEP-370 - First incubator in JDK 14
- JEP-383 - Second incubator in JDK 15
- JEP-393 - Third incubator in JDK 16
- Combined as Foreign Function & Memory API
Foreign Linker API

- API for statically-typed, pure-Java access to native code
  - Focus on Ease of use, flexibility and performance
  - Initial support for C interop
  - Call native code in a .dll/.so/.dylib
  - Create a native function pointer to a Java method which can be passed to code in a native library
- JEP-389 - First incubator in JDK 16
- Combined as Foreign Function & Memory API
Vector API

- API for reliable and performant vector computations
  - Platform agnostic
  - Clear and concise API
  - Reliable runtime compilation and performance
  - Graceful degradations
- JEP-338 - First incubator in JDK 16
- JEP-414 - Second incubator in JDK 7
- JEP-417 - Third incubator in JDK 18
Foreign Function & Memory API

- Evolution of the Foreign-Memory Access API and the Foreign Linker API
  - Same goals and features as the original two (Ease of use, safety, performance, generality)
- JEP-412 - First incubator in JDK 17
- JEP-419 - Second incubator in JDK 18
Jextract

- A simple command line tool
- Generates a Java API from one or more native C headers
- Shipped with OpenJDK Panama builds
- Makes working with large C headers a cakewalk

Generate Java API for OpenGL

```
jextract --source -t org.opengl \ 
   -I /usr/include /usr/include/GL/glut.h
```
JNI vs Panama
getpid with JNI

class Main {
  public static void main(String[] args) {
    System.out.println("my process id: " + getpid());
  }

  private static native int getpid();
}

Implement C class

```c
#include <unistd.h>
#include "Main.h"

JNIEXPORT jint JNICALL Java_Main_getpid
  (JNIEnv *env, jclass cls) {
  // call the actual C function to get the process id!
  return getpid();
}
```

Generate header

```
javac -h . Main.java
```

Compile C code to dynamic lib

```
System.loadLibrary("main");
```

```
java Main.java
```
getpid with Panama (2 ways)

```java
import org.unix.Unix;
import org.unix.Unix.Unix_errno;
import org.unix.Unix.Unix_proc;
import org.unix.Unix.Unix_proc_id;
import org.unix.Unix.Unix_stat;
import org.unix.Unix.Unix_strerror;
import org.unix.Unix.Unix_strerror_errno;
import org.unix.Unix.Unix_pwd;

public static void main(String[] args) throws Throwable {
    var linker : CLinker = CLinker.getInstance();
    var lookup : SymbolLookup = CLinker.systemLookup();
    var getpid : MethodHandle = linker.downcallHandle(
        lookup.lookup(name: "getpid").get(),
        MethodType.methodType(int.class),
        FunctionDescriptor.of(CLinker.O_INT));
    System.out.println((int) getpid.invokeExact());
}
```

```
jextract --source -t org.unix \
-1 /usr/include /usr/include/unistd.h
```

```
java Main.java
```

```
java Main.java
```
Benchmark
# Benchmark on OpenJDK 17

Full benchmark (average time, smaller is better)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mode</th>
<th>Cnt</th>
<th>Score</th>
<th>Error</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFIBenchmark.JNI</td>
<td>avgt</td>
<td>40</td>
<td>49.182 ± 1.079</td>
<td>ns/op</td>
<td></td>
</tr>
<tr>
<td>FFIBenchmark.panamaDowncall</td>
<td>avgt</td>
<td>40</td>
<td>50.746 ± 0.702</td>
<td>ns/op</td>
<td></td>
</tr>
<tr>
<td>FFIBenchmark.panamaJExtract</td>
<td>avgt</td>
<td>40</td>
<td>48.838 ± 1.461</td>
<td>ns/op</td>
<td></td>
</tr>
</tbody>
</table>

[https://github.com/deepu105/Java-FFI-benchmarks](https://github.com/deepu105/Java-FFI-benchmarks)
So are we there yet?
Project panama current state

OpenJDK 17

- Can already work with languages that has C interop
  - like C/C++, Fortran, Rust, etc
- Performance on par with JNI
  - Hopefully this will be improved further
- Jextract makes it really easy to use native libs
- Memory safe and less brittle than JNI
- Native/off-heap memory access
- Documentation needs huge improvement
  - its an incubator feature so this is expected
Learn more

- [https://foojay.io/today/project-panama-for-newbies-part-1/](https://foojay.io/today/project-panama-for-newbies-part-1/)
- [https://hg.openjdk.java.net/panama/dev/raw-file/4810a7de75cb/doc/panama_foreign.html#using-panama-foreign-jdk](https://hg.openjdk.java.net/panama/dev/raw-file/4810a7de75cb/doc/panama_foreign.html#using-panama-foreign-jdk) (some examples are outdated for current API)
Thank You

Deepu K Sasidharan

@deepu105  |  deepu.tech

https://deepu.tech/tags#rust