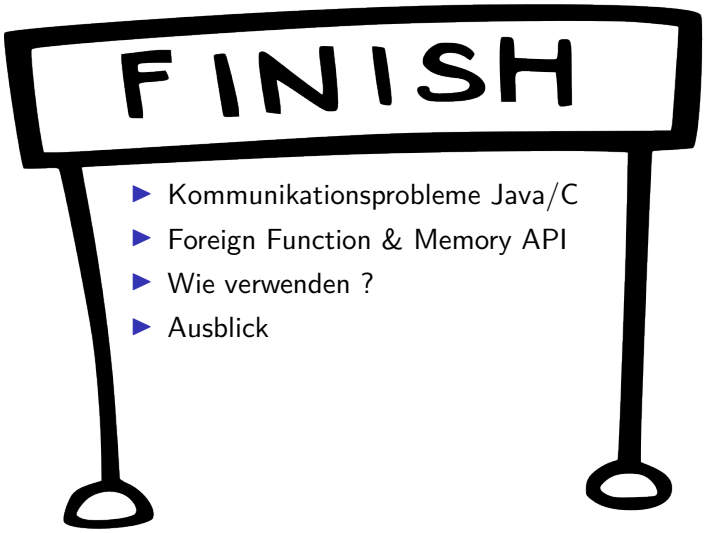
Javas neue Gesprächskultur
– ganz wie in Panama –

Bernd Müller
Ostfalia


FINISH
Kommunikationsprobleme Java/C
Foreign Function & Memory API
Wie verwenden ?
Ausblick

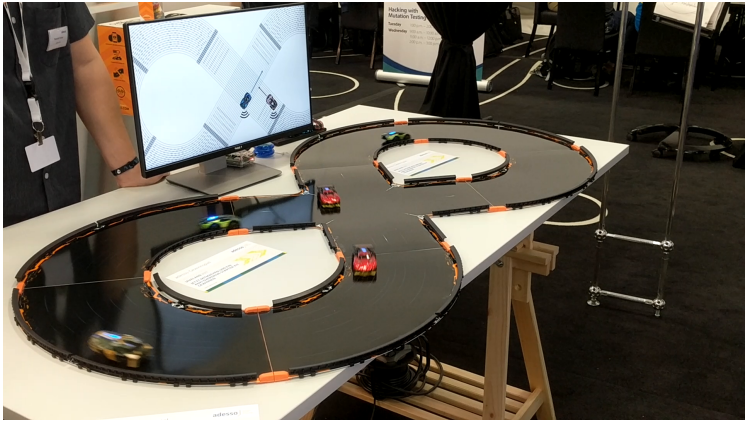
Vorstellung Referent

- Prof. Informatik (Ostfalia, HS Braunschweig/Wolfenbüttel)
- Buchautor (JSF, JPA, Seam, ...)

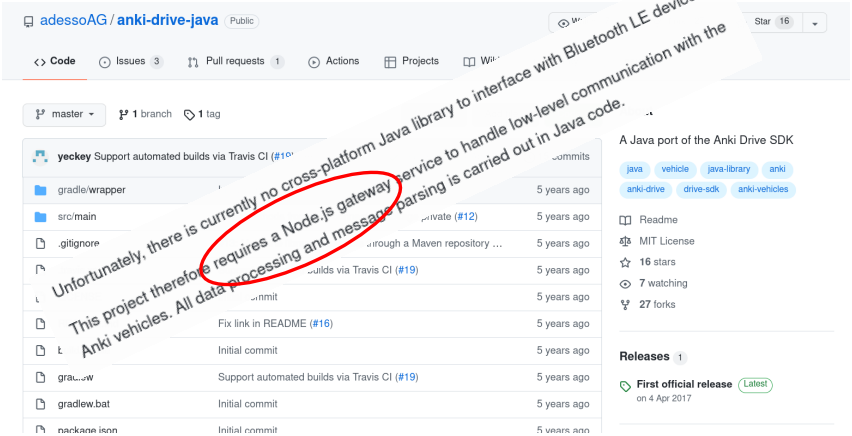
- Mitglied EGs JSR 344 (JSF 2.2) und JSR 338 (JPA 2.1)
- Geschäftsführer PMST GmbH
- JUG Ostfalen (Mitorganisator)
- Beirat der Java aktuell
- bernd.mueller@ostfalia.de
- @berndmuller
- BerndMuller

Motivation

JavaLand 2017, Adesso-Stand

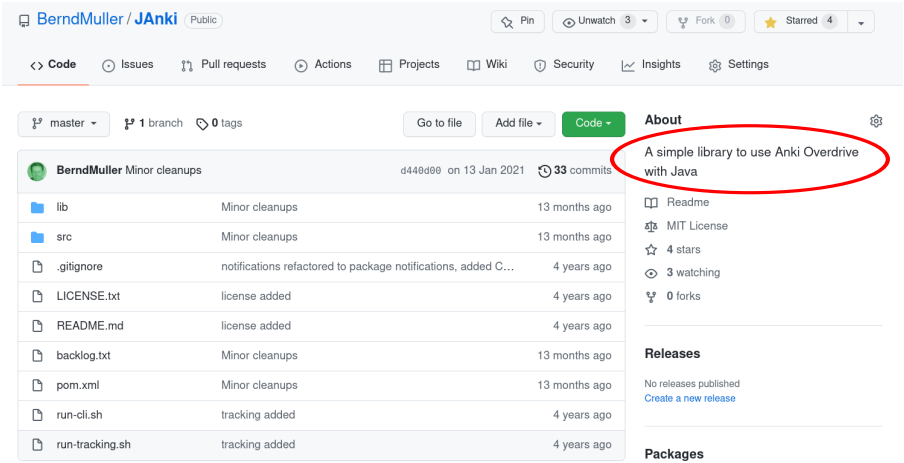


GitHub: adessoAG/anki-drive-java

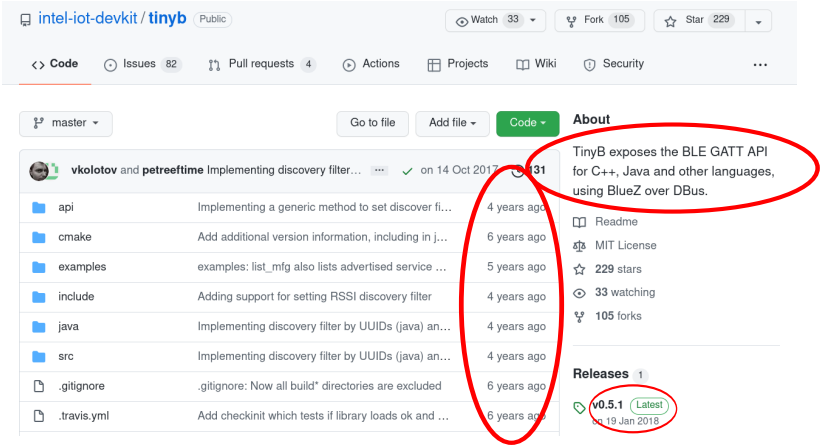


JSR 82: Review Ballot 2000, FR3 2010

JAnki


BerndMuller / JAnki Public
Pin
Unwatch 3
Fork 0
Starred 4
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
BerndMuller Minor cleanups
d440d00 on 13 Jan 2021
33 commits
lib Minor cleanups 13 months ago
src Minor cleanups 13 months ago
.gitignore notifications refactored to package notifications, added C… 4 years ago
LICENSE.txt license added 4 years ago
README.md license added 4 years ago
backlog.txt Minor cleanups 13 months ago
pom.xml Minor cleanups 13 months ago
run-cli.sh tracking added 4 years ago
run-tracking.sh tracking added 4 years ago
About
A simple library to use Anki Overdrive with Java
Readme
MIT License
4 stars
3 watching
0 forks
Releases
No releases published
Create a new release
Packages

TinyB von Intel



Selbst ist der Mann
und das Ergebnis ist immer ein Erlebnis ;-)

Ziel: Keine Bibliothek, nur JDK + BlueZ

Verallgemeinerung

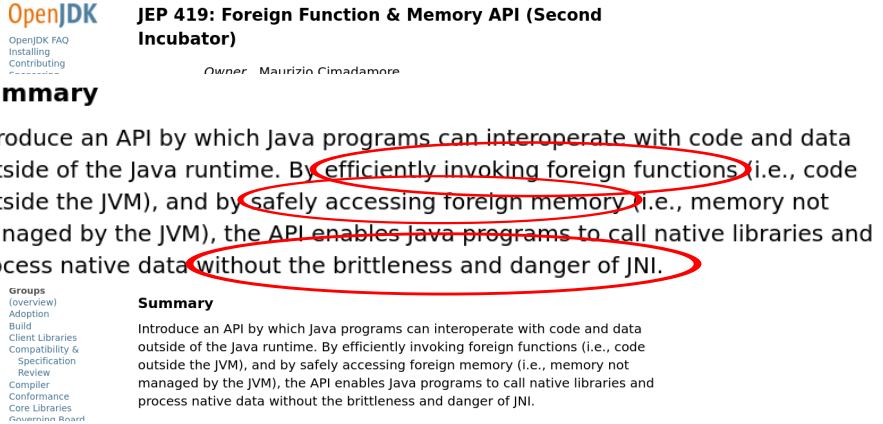
Keine Bibliothek, nur JDK + <your choice>

Das Problem

Das Problem – und hoffentlich die Lösung

- Manchmal genügt JDBC, HTTP, NIO, UNIX-Domain-Sockets, ... nicht
- Sogenannter *Off-Heap-Speicher* (außerhalb JVM) soll direkt zugegriffen werden
- Machen z.B. Tensorflow, Ignite, Lucene, Netty und vielen andere
- Es gibt: JNI, Byte-Buffer-API, `sun.misc.Unsafe`, ...
- Macht aber auf Dauer nicht glücklich ...
- JEPs 191, 370, 383, 389, 393, 412, 419, 424
- Mit Project Panama: Interconnecting JVM and native code

JEP 419: Foreign Function & Memory API



OpenJDK

OpenJDK FAQ
Installing
Contributing

JEP 419: Foreign Function & Memory API (Second Incubator)

Owner Maurizio Cimadamore

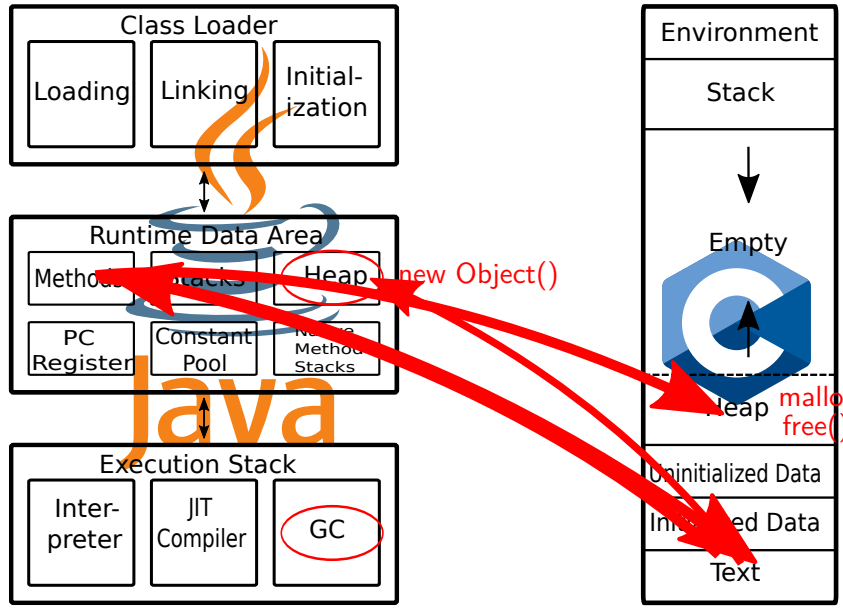
Summary

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI.

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board

Summary

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI.


Class Loader
Loading
Linking
Initial-
ization
Runtime Data Area
Heap
PC
Register
Constant
Pool
Execution Stack
Inter-
preter
JIT
Compiler
GC
new Object()
Environment
Stack
Empty
malloc()
free()
Uninitialized Data
Text
Native Method Interface + Libraries

Lösungsversuch FFM: Konzepte und Klassen (Java 17)

- Allokation von Speicher
  `MemorySegment`, `MemoryAddress`, `SegmentAllocator`, ...
- Lesen und Schreiben von (strukturiertem) Speicher
  `MemoryLayout`, `VarHandle` (`java.lang.invoke`), ...
- Regeln des Lebenszyklus von Ressourcen
  `ResourceScope`
- Aufruf von Funktionen
  `SymbolLookup`, `CLinker`

Beispiel Memory Segment

- Zusammenhängender Speicherbereich
- On- oder Off-Heap
- Nativ, mapped (mmap), Array oder Buffer
- Garantien für räumliche, zeitliche sowie Thread-bezogene Beschränkungen
- ...

Beispiele

Prozess-Id (C)

Prozess-Id des Prozesses: keine Parameter, Integer-Rückgabe (`jps`-Clone)

```
#include <unistd.h>

pid_t getpid(void);
```

Prozess-Id (Java)

```
MethodHandle getpid =  CLinker.getInstance().downcallHandle(
    CLinker.systemLookup().lookup("getpid").get(),
    MethodType.methodType(int.class),
    FunctionDescriptor.of(CLinker.C_INT));

System.out.println("Process Id: " + (int) getpid.invokeExact());
```

Demo Time

Sleepy from slides, we are !

Quicksort

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

Vergleichsmethode/-klasse (à la `Comparator#compare()`)

```
static class Comparator {

  static int compare(MemoryAddress addr1, MemoryAddress addr2) {
    int v1 = MemoryAccess.getIntAtOffset(
                 MemorySegment.globalNativeSegment(),
                 addr1.toRawLongValue());
    int v2 = MemoryAccess.getIntAtOffset(
                 MemorySegment.globalNativeSegment(),
                 addr2.toRawLongValue());
    return v1 - v2;
  }

}
```

Method-Handle auf qsort

```
MethodHandle qsort = CLinker.getInstance().downcallHandle(

  CLinker.systemLookup().lookup("qsort").get(),

  MethodType.methodType(void.class, MemoryAddress.class, long.class,
                        long.class, MemoryAddress.class),

  FunctionDescriptor.ofVoid(C_POINTER, C_LONG, C_LONG, C_POINTER)

  );
```

Method-Handle auf Vergleichsmethode

```
MethodHandle comparHandle = MethodHandles.lookup()

    .findStatic(Comparator.class, "compare",

    MethodType.methodType(int.class, MemoryAddress.class,
                          MemoryAddress.class));
```

Allokation des Arrays und eigentlicher Aufruf

```
try (ResourceScope scope = ResourceScope.newConfinedScope()) {
  MemoryAddress comparFunc = CLinker.getInstance().upcallStub(
    comparHandle,
    FunctionDescriptor.of(C_INT, C_POINTER, C_POINTER), scope);

  MemorySegment array = SegmentAllocator.ofScope(scope)
    .allocateArray(C_INT, new int[] {0, 9, 3, 4, 6, 5, 1, 8, 2, 7});

    qsort.invokeExact(array.address(), 10L, 4L, comparFunc);

    int[] sorted = array.toIntArray();
}
```

Demo Time

Sleepy from slides, we are !

Werkzeugunterstützung

JExtract

- Reduktion des Overheads durch
  - Analyse einer Include-Datei
  - und Generierung der entsprechenden Methoden und Typen für den Aufruf
- Nicht im JDK, evtl. später separat verfügbar

getpid ohne jextract

```
MethodHandle getpid =  CLinker.getInstance().downcallHandle(
    CLinker.systemLookup().lookup("getpid").get(),
    MethodType.methodType(int.class),
    FunctionDescriptor.of(CLinker.C_INT));

System.out.println("Process Id: " + (int) getpid.invokeExact());
```

getpid mit jextract

```
System.out.println("Process Id: " + getpid());
```

Demo Time

Sleepy from slides, we are !

Es geht weiter . . . (Java 17 $\Rightarrow$ 18 $\Rightarrow$ 19)

Quicksort mit Java 17 (Ausschnitt)

```
MethodHandle qsort = CLinker.getInstance().downcallHandle(
    CLinker.systemLookup().lookup("qsort").get(),
    MethodType.methodType(void.class, MemoryAddress.class,
                          long.class, long.class, MemoryAddress.class),
    FunctionDescriptor.ofVoid(C_POINTER, C_LONG, C_LONG, C_POINTER)
  );
```

Quicksort mit Java 18 (Ausschnitt)

```
MethodHandle qsort = CLinker.systemCLinker().downcallHandle(
    CLinker.systemCLinker().lookup("qsort").get(),
    FunctionDescriptor.ofVoid(ADDRESS, JAVA_LONG, JAVA_LONG, ADDRESS)
  );
```

Quicksort mit Java 19 (Ausschnitt)

```
MethodHandle qsort = Linker.nativeLinker().downcallHandle(
    Linker.nativeLinker().defaultLookup().lookup("qsort").get(),
    FunctionDescriptor.ofVoid(ADDRESS, JAVA_LONG, JAVA_LONG, ADDRESS)
  );
```

Demo Time

Sleepy from slides, we are !

Fazit und Ausblick

Persönliche – und damit subjektive – Einschätzung

- Eine wirklich gute Idee
- Versuche JAnki zu migrieren. Schwierigkeit: Komplexität des D-Bus-Interface und eigene Unzulänglichkeiten ;-)
- Wenn GraalVMs Native-Image-Erzeugung FFM unterstützt: ☺
- und was machen andere ?

Tomcat SSL

Tomcat 9.0.55 (remm) **2021-11-10**

Coyote

- Improve performance of Connector shutdown - primarily to reduce the time it takes to run the test suite. (markt)
- Refactor the APR/native connector shutdown to reduce the possibility of a JVM crash during the connector shutdown. (markt)
- #457: Add a `toString()` method to `MimeHeader` to aid debugging. (dblevins)
- Add experimental OpenSSL support through the Panama API incubating in Java 17, with support for OpenSSL 1.1. This no longer requires tomcat-native or APR. Please refer to the `openssl-java17` module from the `main` branch for more details. (remm)
- Fix APR connector stop so it correctly waits for the sendfile thread, if any, to exit. (markt)
- Do not ignore the error condition if the APR connector is not able to open a server socket as continuing in this case will trigger a JVM crash. (markt)

Elasticsearch

Some scattered feedback on the Foreign Linker API

Chris Hegarty chegar999 at gmail.com
Wed Dec 22 09:19:40 UTC 2021

- Previous message (by thread): Calling the Port Audio C API via Panama FFI APIs
- Next message (by thread): Some scattered feedback on the Foreign Linker API
- **Messages sorted by:** [date] [thread] [subject] [author]

Hi,

As part of a recent prototyping effort, we evaluated replacing the usage of JNA in the core of the Elasticsearch server with the incubating Foreign Linker API.

TL;DR things just worked, perf improved, and we really like the restricted native access.

Our usage (at least in this particular case) is not really performance sensitive, in that we reach into native to setup syscall filtering and check resource limits - but hey, faster is always better! We measured approximate perf improvements between 8 and 20 times faster for simple downcalls. The reason the numbers vary so much is more to do with JNA rather than Panama, since the Panama times from run to run were

JDK-Integration

Fragen und Anmerkungen

Vortrag und Code

https://github.com/BerndMuller/panama-jfs-2022