FFM API — ничего общего с радио

Настя Лисицкая

Яндекс

Кто я

- Больше 8 лет в Яндексе занимаюсь java разработкой
- Сейчас тимлид в команде разработки продуктивности и международных сценариев в Алисе
- Участник рабочей группы java-комьюнити в Яндексе

О чем песня

1) Зачем оно нам надо
2) Общая вводная в FFM API
3) Как попробовали:
   - Примеры
   - Проблемы
4) Планы/выводы

Что на входе

1) Базовые сценарии в Алисе реализованы на:
   - C++
   - Kotlin
2) Есть использование JNI
3) В проде на Java 22

Project Panama: Interconnecting JVM and native code

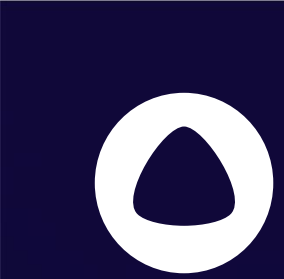
Foreign Function & Memory API — adds support for foreign memory access and foreign function calls

Jextract — a tool which mechanically generate Java bindings from native library headers

Vector API — adds vectorization support in Java through JVM intriniscs

openjdk.org

inside.java



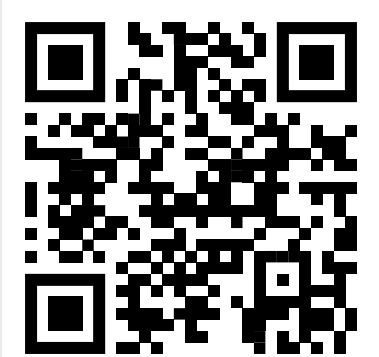
JEP 469: Vector API (Eighth Incubator) — Java 23

JEP 454: Foreign Function & Memory API

Release 22

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.



openjdk.org

History

- The so-called «Foreign Memory Access API» was presented in the incubator stage back in March 2020 in Java 14 (JEP 370)
- One year later, the «Foreign Linker API» was introduced in the incubator stage in Java 16 (JEP 389)
- In Java 17, the two APIs were merged into the «Foreign Function & Memory API», and this unified API was presented once again as an incubator version (JEP 412)
- In Java 19, the FFM API was promoted to the preview stage (JEP 424)
- In Java 22, the API was declared ready for production and finalized in March 2024 after a long development and maturation period (JEP 454)

Goals

Productivity — Replace the brittle machinery of native methods and the Java Native Interface (JNI) with a concise, readable, and pure-Java API.

Performance — Provide access to foreign functions and memory with overhead comparable to, if not better than, JNI and sun.misc.Unsafe.

Broad platform support — Enable the discovery and invocation of native libraries on every platform where the JVM runs.

Uniformity — Provide ways to operate on structured and unst[illegible]ed d[illegible]a, [illegible] unli[illegible]ited s[illegible]n multiple kinds of mem[illegible]e.g., n[illegible] memo[illegible]rsist[illegible] mo[illegible] a[illegible] man[illegible]ged h[illegible]ap m[illegible]

So[illegible]ndne[illegible] — [illegible] allocated and de[illegible]loca[illegible] a[illegible]ss [illegible]ultiple t[illegible]

Int[illegible]grity — Allow programs to perform unsafe operations with native code and data, but war[illegible] us[illegible]out such operations by default.

ПРОДАНО

Понятное API

Не хуже по перфомансу

Не только C/C++

Description

The Foreign Function & Memory API (FFM API) defines classes and interfaces so that client code in libraries and applications can

- Control the allocation and deallocation of foreign memory (MemorySegment, Arena, and SegmentAllocator)
- Manipulate and access structured foreign memory (MemoryLayout and VarHandle), and
- Call foreign functions (Linker, SymbolLookup, FunctionDescriptor, and MethodHandle)

The FFM API resides in the java.lang.foreign package of the java.base module

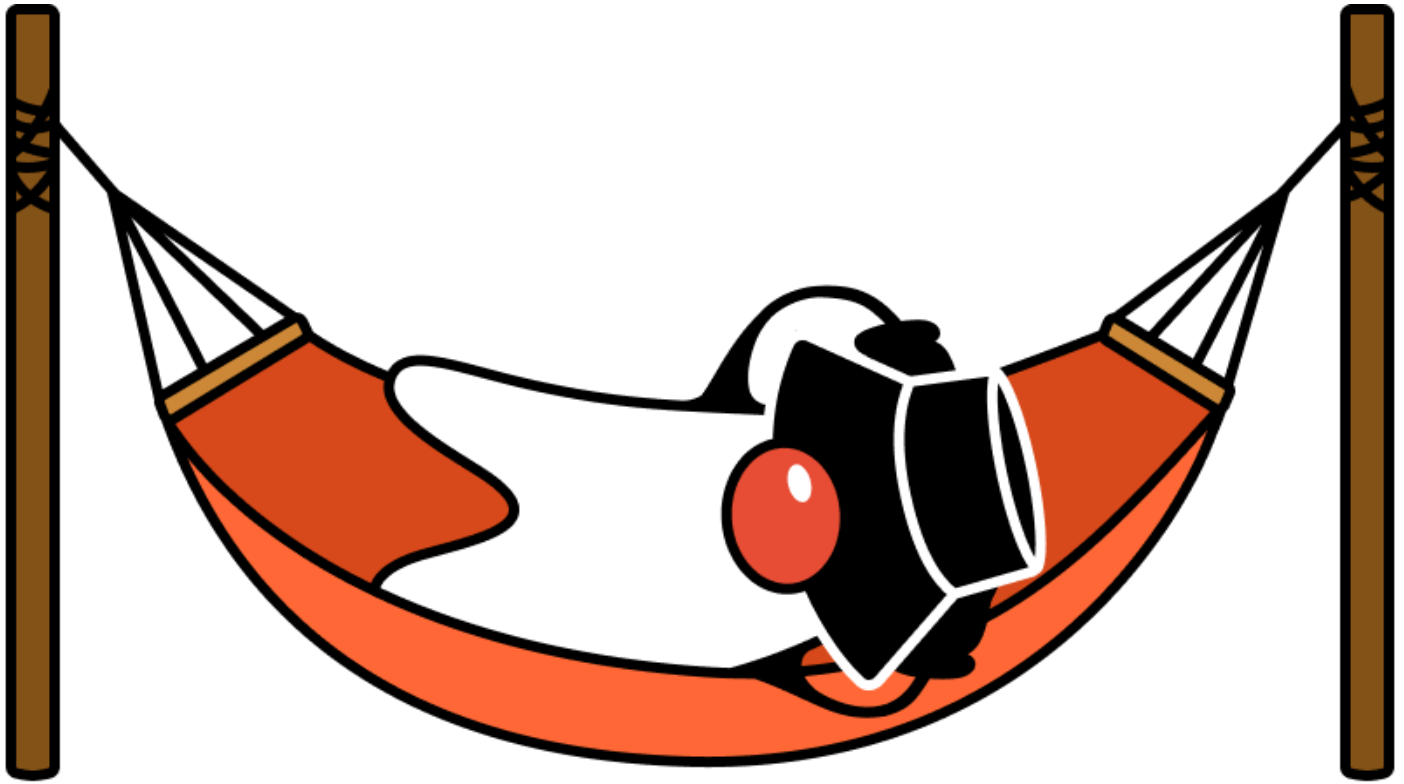
Норм дока от Oracle

docs.oracle.com

Из доки

```
try (Arena arena = Arena.ofConfined()) {
    // Allocate off-heap memory and copy the argument, a Java string, into off-heap memory
    MemorySegment nativeString = arena.allocateUtf8String(s);

    // Link and call the C function strlen
    // Obtain an instance of the native linker
    Linker linker = Linker.nativeLinker();

    // Locate the address of the C function signature
    SymbolLookup stdLib = linker.defaultLookup();
    MemorySegment strlen_addr = stdLib.find("strlen").get();

    // Create a description of the C function
    FunctionDescriptor strlen_sig = FunctionDescriptor.of(ValueLayout.JAVA_LONG, ValueLayout.ADDRESS);

    // Create a downcall handle for the C function
    MethodHandle strlen = linker.downcallHandle(strlen_addr, strlen_sig);

    // Call the C function directly from Java
    return (long)strlen.invokeExact(nativeString);
}
```

Ну все, дело в Paname



Простой пример (нормализация текста на C++)

```
#include <library/cpp/langs/langs.h>

…

namespace nnlu {

  std::string NormalizeText(std::string_view text, ELanguage lang);

  …

} // namespace nnlu
```

В коде вызов:

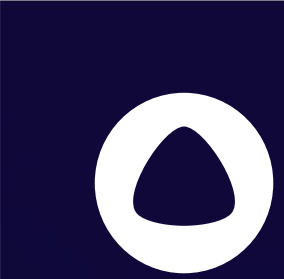
```java
/**
 * Binding for normalization
 */
@ParametersAreNonnullByDefault
public class Normalizer {

  static {
    System.loadLibrary("normalization_java");
  }

  @Nonnull
  public String normalizeText(String text, Lang lang) {
    Objects.requireNonNull(text, "text argument is null");
    Objects.requireNonNull(lang, "lang argument is null");

    return normalizeText(text, lang.getCode());
  }

  private native String normalizeText(String text, String lang);
}
```

JNI — .h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class ru_yandex_alice_libs_normalization_Normalizer */

#ifndef _Included_ru_yandex_alice_libs_normalization_Normalizer
#define _Included_ru_yandex_alice_libs_normalization_Normalizer
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     ru_yandex_alice_libs_normalization_Normalizer
 * Method:    normalizeText
 * Signature: (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_ru_yandex_alice_libs_normalization_Normalizer_normalizeText
  (JNIEnv *, jobject, jstring, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

JNI — .cpp

```
...
JNIEXPORT jstring JNICALL Java_ru_yandex_alice_nlu_libs_normalization_Normalizer_normalizeText
  (JNIEnv *jenv, jobject, jstring text, jstring lang) {
    const char* k_text = jenv->GetStringUTFChars(text, 0);
    const char* k_lang = jenv->GetStringUTFChars(lang, 0);
    auto t = std::string(k_text);
    auto l = LanguageByName(std::string(k_lang));
    auto res =  NNlu::NormalizeText(t, l);
    if (res.empty()) {
        return nullptr;
    }
    jstring normalized = jenv->NewStringUTF(res.c_str());
    jenv->ReleaseStringUTFChars(text, k_text);
    jenv->ReleaseStringUTFChars(lang, k_lang);
    return normalized;
  }
```



Постараемся поймать волну FFM API

Найти свою библиотеку

```
SymbolLookup symbolLookup = SymbolLookup.loaderLookup();// is dynamic with respect to the
// libraries associated with the class loader, i.e., by System.load(String) or System.loadLibrary(String)

or

SymbolLookup symbolLookup = Linker.nativeLinker().defaultLookup();//for libraries that are
// commonly used on the OS and processor combination supported by that Linker

or

try (Arena arena = Arena.ofConfined()) {
  SymbolLookup symbolLookup = SymbolLookup.libraryLookup("libnormalization.so/dylib/dll",
arena);
  …
}
```

Найти метод в библиотеке

```
// Returns the address of the symbol with the given name.
MemorySegment memorySegment = symbolLookup.find(”NormalizeText").orElseThrow();
```

MemorySegment

You can access off-heap or on-heap memory with the Foreign Function and Memory (FFM) API through the MemorySegment interface. Each memory segment is associated with, or backed by, a contiguous region of memory.

Heap segment

- ofArray
- ofBuffer — direct ByteBuffer

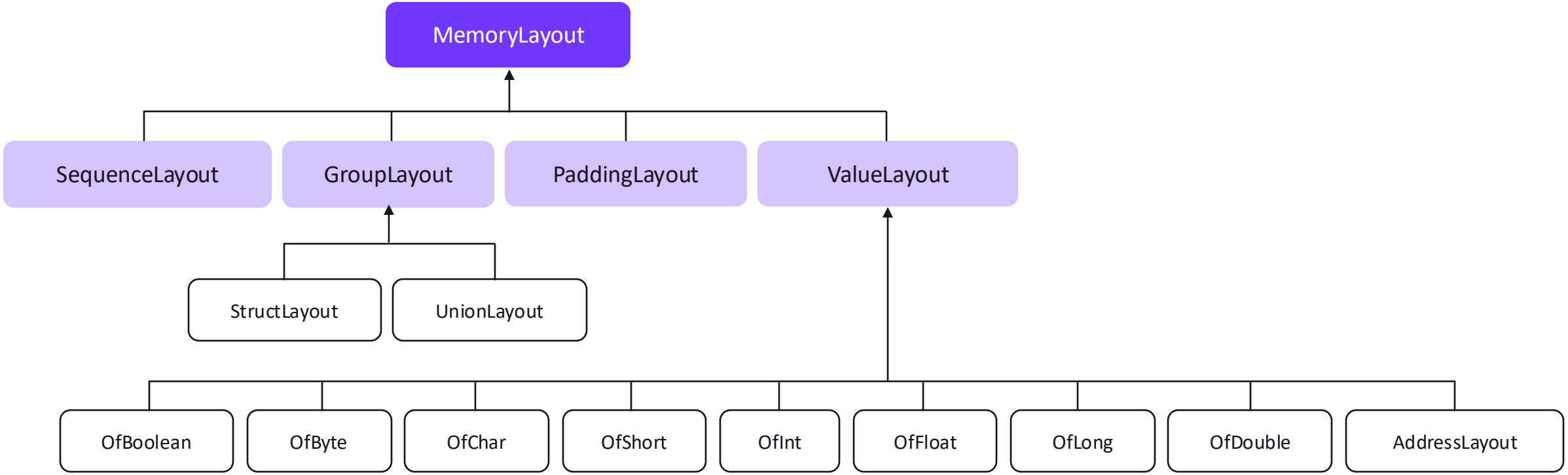
Native segment

Use an arena to allocate a memory segment

- Arena.allocate(long, long)
- By mapping a file into a new off-heap region (FileChannel.map)
- ofBuffer

Описать структуры данных с MemoryLayout

Composite types such as struct and union types are modeled with the GroupLayout interface, which is a supertype of StructLayout and UnionLayout.



C types mapping in Linux/x64

C type	Layout	Java carrier
bool	JAVA_BOOLEAN	byte
char	JAVA_BYTE	byte
short	JAVA_SHORT	short, char
int	JAVA_INT	int
long	JAVA_LONG	long
long long	JAVA_LONG	long
float	JAVA_FLOAT	float
double	JAVA_DOUBLE	double
char* int** …	ADDRESS	MemorySegment
struct Point { int x; int y; }; union Choice {float a; int b; }; …	MemoryLayout.structLayout(...) MemoryLayout.unionLayout(...)	MemorySegment

SequenceLayout

A compound layout that denotes a homogeneous repetition of a given element layout. The repetition count is said to be the sequence layout's element count. A sequence layout can be thought of as a struct layout where the sequence layout's element layout is repeated a number of times that is equal to the sequence layout's element count.

```
SequenceLayout abcLayout
 = MemoryLayout.sequenceLayout(7,
 MemoryLayout.structLayout(
   ValueLayout.JAVA_INT.withName("a"),
   ValueLayout.JAVA_INT.withName("b"),
   ValueLayout.JAVA_INT.withName("c")
));
```

Нужный нам метод

```
...
  std::string NormalizeText(std::string_view text, ELanguage lang);
...
```

Может пригодиться оберточка (печалька) Normalizer.h

```
#ifndef _FF_Normalizer
#define _FF_Normalizer
#ifdef __cplusplus
extern "C" {
#endif

char* NormalizeText(const char* text, const char* lang);

#ifdef __cplusplus
}
#endif
#endif //_FF_Normalizer
```

Может пригодиться оберточка (двойная печалька) .cpp

```
#include " Normalizer.h"

…

char* NormalizeText(const char* text, const char* lang) {
            auto l = LanguageByName(lang);
            auto res = nnlu::NormalizeText(std::string(text), l);
    return strdup(res.c_str());
}
```

Описать сигнатуру функции

```
char* NormalizeText(const char* text, const char* lang);
```

The first argument of the FunctionDescriptor::of method is the layout of the native function's return value.

```
FunctionDescriptor FUNCTION_DESCRIPTOR = FunctionDescriptor.of(
    ValueLayout.ADDRESS,   // Return type
    ValueLayout.ADDRESS,   // const char* text
    ValueLayout.ADDRESS    // const char* lang
);
```

ValueLayout.ADDRESS. This represents the only argument for strlen, a pointer to a string.

Как объявлен метод в доке

```
// Lookup the function in the standard native library
MethodHandle strlen = Linker.nativeLinker().downcallHandle(
        linker.defaultLookup().find("strlen").get(),
        FunctionDescriptor.of(ValueLayout.JAVA_LONG, ValueLayout.ADDRESS)
);
```

Объявим наш метод

```
// Lookup the custom function in the native library
MethodHandle ffNormalizeText = Linker.nativeLinker().downcallHandle(
        SymbolLookup.loaderLookup().find("NormalizeText").orElseThrow(),
        FUNCTION_DESCRIPTOR
);
```

Как будто поймать волну становится сложнее

Выделить в памяти свои аргументы

```
try (Arena arena = Arena.ofConfined()) {
  // Allocate off-heap memory
  MemorySegment textSegment = arena.allocateFrom(text); //text:String
  MemorySegment langSegment = arena.allocateFrom(langCode); // langCode:String
} // Off-heap memory is deallocated
```

public java. lang. foreign. MemorySegment allocate(java.lang.foreign.MemoryLayout layout)
public java. lang. foreign. MemorySegment allocateFrom(java.lang.foreign.ValueLayout.OfInt layout, int value)

Arena

An arena controls the lifecycle of native memory segments, providing both flexible allocation and timely deallocation.

An arena has a scope — the arena scope.

Scope — specifies when the region of memory that backs the memory segment will be deallocated and is no longer valid.

Arena

Arena::ofConfined — provides a bounded and deterministic lifetime. Its scope is alive from when it's created to when it's closed. A confined arena has an owner thread. This is typically the thread that created it. Only the owner thread can access the memory segments allocated in a confined arena. You'll get an exception if you try to close a confined arena with a thread other than the owner thread.

Arena::ofShared, has no owner thread. Multiple threads may access the memory segments allocated in a shared arena. In addition, any thread may close a shared arena, and the closure is guaranteed to be safe and atomic.

UnsupportedOperationException

```
try (Arena arena = Arena. ofAuto()){ … }
// or

try (Arena arena = Arena.global()){ … }

// or
arena.close();

java.lang.UnsupportedOperationException: Attempted to close a non-closeable session

    at java.base/jdk.internal.foreign.MemorySessionImpl.nonCloseable(MemorySessionImpl.java:318)
    at java.base/jdk.internal.foreign.GlobalSession.justClose(GlobalSession.java:71)
    at java.base/jdk.internal.foreign.MemorySessionImpl.close(MemorySessionImpl.java:232)
    at java.base/jdk.internal.foreign.ArenaImpl.close(ArenaImpl.java:50)
```

Arena::ofAuto

This is an area that's managed, automatically, by the garbage collector. Any thread can access memory segments allocated by an automatic arena. If you call Arena::close on an automatic arena, you'll get a UnsupportedOperationException.

Arena::global

The global arena features an unbounded lifetime. Any thread can access memory segments allocated with this arena. In addition, the region of memory of these memory segments is never deallocated; if you call Arena::close on a global arena, you'll get a UnsupportedOperationException.

Custom arenas

```
class SlicingArena implements Arena {
  final Arena arena = Arena.ofConfined();
  final SegmentAllocator slicingAllocator;

  (long size) {
    slicingAllocator =
    SegmentAllocator.slicingAllocator(arena.allocate(size));
  }

  public MemorySegment allocate(long byteSize, long
  byteAlignment) {
    return slicingAllocator.allocate(byteSize,
    byteAlignment);
  }

  public MemorySegment.Scope scope() {
    return arena.scope();
  }

  public void close() {
    arena.close();
  }
}
```

```
try (Arena slicingArena = new SlicingArena(1000)) {
  for (int i = 0; i < 10; i++) {
    MemorySegment s =
    slicingArena.allocateFrom(JAVA_INT, 1, 2, 3, 4, 5);
    ...
  }
} // all memory allocated is released here
```

Дошли до вызова метода

```
private final FunctionDescriptor FUNCTION_DESCRIPTOR = FunctionDescriptor.of(
  ValueLayout.ADDRESS,   // Return type
  ValueLayout.ADDRESS,   // const char* text
  ValueLayout.ADDRESS    // const char* lang
);

...
System.loadLibrary("normalization_java");
// Allocate memory and call the function
try (Arena arena = Arena.ofConfined()) {
  // Lookup the function in the native library
  MethodHandle normalizeText = Linker.nativeLinker().downcallHandle(
    SymbolLookup.loaderLookup().find("NormalizeText").orElseThrow(),
    FUNCTION_DESCRIPTOR
  );
// УРА, делаем вызов:
  MemorySegment segment = (MemorySegment) normalizeText.invokeExact(arena.allocateFrom(text),
  arena.allocateFrom(langCode));
```

В консольке

```
WARNING: A restricted method in java.lang.foreign.Linker has been called
WARNING: java.lang.foreign.Linker::downcallHandle has been called by
ru.yandex.alice.nlu.libs.normalization.Normalizer$NormalizerSettings in an unnamed
module
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this
module
WARNING: Restricted methods will be blocked in a future release unless native access
is enabled
```

--enable-native-access*

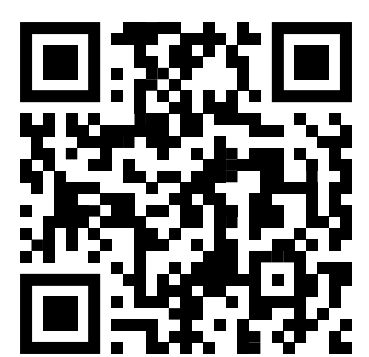
--enable-native-access=M

--enable-native-access=M1,M2,M3

--enable-native-access=ALL-UNNAMED

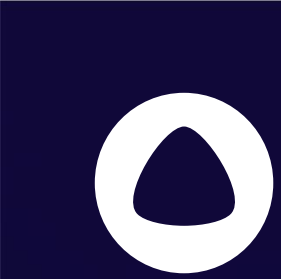
JEP 472: Prepare to Restrict the Use of JNI (Release 24)

When the --enable-native-access option is present, any use of unsafe methods from outside the list of specified modules causes an IllegalCallerException to be thrown, rather than a warning to be issued.



openjdk.org

Что у нас там вернулось…



Как получить значение String по ссылке

docs.oracle.com

stackoverflow.com

MemorySegment::reinterpret

MemorySegment::reinterpret method enables you to work with zero length memory segments so that you can safely access them and attach them to an existing arena so that the lifetime of the region of memory backing the segment can be managed automatically.

MemorySegment::reinterpret

1) reinterpret(long)
2) reinterpret(Arena, Consumer)
3) reinterpret(long, Arena, Consumer)

Params:

- newSize — the size of the returned segment
- arena — the arena to be associated with the returned segment
- cleanup — the cleanup action that should be executed when the provided arena is closed (can be null)

Returns: a new segment that has the same address as this segment, but with the new size and its scope set to that of the provided arena.

Пример очистки из доки

```
// Locate the address of free()
var free_addr = Linker.nativeLinker().defaultLookup().find("free").orElseThrow();

// Create a downcall handle for free()
MethodHandle free = linker.downcallHandle(
    free_addr,
    FunctionDescriptor.ofVoid(ValueLayout.ADDRESS)
);

// This reintepret method:
// 1. Resizes the memory segment so that it's equal to byteSize
// 2. Associates it with an existing arena
// 3. Invokes free() to deallocate the memory allocated previously
//    when its arena is closed

Consumer<MemorySegment> cleanup = s -> {
  try {
    free.invokeExact(s);
  } catch (Throwable e) {
    throw new RuntimeException(e);
  }
};
```

Неправильный size в reinterpret

```
java.lang.IndexOutOfBoundsException: Out of bound access on segment MemorySegment{ address: 0x6000030b2820, byteSize: 24 }; new offset = 24; new length = 1

  at java.base/jdk.internal.foreign.AbstractMemorySegmentImpl.outOfBoundException(AbstractMemorySegmentImpl.java:434)
  at java.base/jdk.internal.foreign.AbstractMemorySegmentImpl.apply(AbstractMemorySegmentImpl.java:415)
  at java.base/jdk.internal.foreign.AbstractMemorySegmentImpl.apply(AbstractMemorySegmentImpl.java:70)
  at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:98)
  at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:124)
  at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:448)
  at java.base/jdk.internal.foreign.AbstractMemorySegmentImpl.checkBounds(AbstractMemorySegmentImpl.java:404)
  at java.base/jdk.internal.foreign.AbstractMemorySegmentImpl.checkAccess(AbstractMemorySegmentImpl.java:364)
  at java.base/java.lang.invoke.VarHandleSegmentAsBytes.checkAddress(VarHandleSegmentAsBytes.java:81)
  at java.base/java.lang.invoke.VarHandleSegmentAsBytes.get(VarHandleSegmentAsBytes.java:107)
  at java.base/jdk.internal.foreign.AbstractMemorySegmentImpl.get(AbstractMemorySegmentImpl.java:728)
  at java.base/jdk.internal.foreign.StringSupport.strlenByte(StringSupport.java:197)
  at java.base/jdk.internal.foreign.StringSupport.chunkedStrlenByte(StringSupport.java:152)
...
```

Note: `MemorySegment::reinterpret` is a restricted method, which, if used incorrectly, might crash the JVM or silently result in memory corruption. See Restricted Methods for more information.

Restricted Methods

Some methods in the Foreign Function and Memory (FFM) API are unsafe and therefore restricted. If used incorrectly, restricted methods can crash the JVM and may silently result in memory corruption.

```
MemorySegment.reinterpret(long)

MemorySegment.reinterpret(long, Arena, Consumer)

MemorySegment.reinterpret(Arena, Consumer)
```

These methods allows you to change the size and lifetime of an existing segment by creating a new alias to the same region of memory. See Foreign Functions That Return Pointers for more information.

The spatial or temporal bounds associated with the memory segment alias returned by these methods might be incorrect. For example, consider a region of memory that's 10 bytes long that's backing a zero-length memory segment. An application might overestimate the size of the region and use `MemorySegment::reinterpret` to obtain a segment that's 100 bytes long. Later, this might result in attempts to dereference memory outside the bounds of the region, which might cause a JVM crash or, even worse, result in silent memory corruption.

docs.oracle.com

Что получилось

```
private final FunctionDescriptor FUNCTION_DESCRIPTOR = FunctionDescriptor.of(
    ValueLayout.ADDRESS,   // Return type
    ValueLayout.ADDRESS,   // const char* text
    ValueLayout.ADDRESS    // const char* lang
);

...
try (Arena arena = Arena.ofConfined()) {
  // Lookup the function in the native library
  MethodHandle normalizeText = Linker.nativeLinker().downcallHandle(
    SymbolLookup.loaderLookup().find("NormalizeText ").orElseThrow(),
    FUNCTION_DESCRIPTOR
  );
  MemorySegment segment = (MemorySegment)
  normalizeText.invokeExact(arena.allocateFrom(text), arena.allocateFrom(langCode));
    MemorySegment reinterpret = segment.reinterpret(Long.MAX_VALUE);
    return reinterpret.getString(0);
} // Off-heap memory is deallocated
```

Как-то все сложно… но есть jextract

The jextract tool parses header (.h) files of native libraries, and generates Java code, called bindings, which use the Foreign Function and Memory API (FFM API) under the hood, that can be used by a client to access the native library.

github.com

Чуть-чуть про jextract

```
$ jextract
 --include-dir /path/to/mylib/include
 --output src
 --target-package org.jextract.mylib
 --library mylib
 /path/to/mylib/include/mylib.h
```

Other Languages

As noted in the introduction, jextract currently only supports C header files, but many other languages also support C interop, and jextract/FFM can still be used to talk to libraries written in those language through an intermediate C layer. The table below describes how to do this for various different langauges:

Language	Method of access
C++	C++ allows declaring C methods using extern "C", and many C++ libraries have a C interface to go with them. Jextract can consume such a C interface, which can then be used to access the library in question.
Rust	The Rust ecosystem has a tool called cbindgen which can be used to generate a C interface for a Rust library. Such a generated C interface can then be consumed by jextract, and be used to access the library in question.

FATAL: Unexpected exception java.lang.RuntimeException: Unsupported language: C++ occurred

Меняем на вызовы jextract

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


try (Arena arena = Arena.ofConfined()) {
    MemorySegment segment = FF_Normalizer_h.ffaNormalizeText(
        arena.allocateFrom(text),
        arena.allocateFrom(langCode));


    MemorySegment reinterpret = segment.reinterpret(Long.MAX_VALUE);

    return reinterpret.getString(0);
} // Off-heap memory is deallocated
```

Бенч на простеньких тестах

Benchmark	Mode	Cnt	Score	Error	Units
NormalizeWithBenchTest.normalizeUsingFF	avgt	10	58.778 ±	0.185	us/op
NormalizeWithBenchTest.normalizeUsingJNI	avgt	10	5.197 ±	0.011	us/op
NormalizeWithBenchTest.normalizeUsingJextract	avgt	10	5.227 ±	0.075	us/op
NormalizeWithBenchTest.turnOn10PercentUsingFF	avgt	10	8.952 ±	0.264	us/op
NormalizeWithBenchTest.turnOn10PercentUsingJNI	avgt	10	5.205 ±	0.028	us/op
NormalizeWithBenchTest.turnOn10PercentUsingJextract	avgt	10	5.165 ±	0.218	us/op

Вдохновившись jextractom

```
private static class NormalizerSettings {
  private static final SymbolLookup SYMBOL_LOOKUP = SymbolLookup.loaderLookup();
  public static final MemorySegment ADDR = SYMBOL_LOOKUP.find("NormalizeText").orElseThrow();
  private static final FunctionDescriptor FUNCTION_DESCRIPTOR = FunctionDescriptor.of(
      ValueLayout.ADDRESS,   // Return type
      ValueLayout.ADDRESS,   // const char* text
      ValueLayout.ADDRESS    // const char* lang  );
  public static final MethodHandle HANDLE = Linker.nativeLinker().downcallHandle(ADDR, FUNCTION_DESCRIPTOR);
} …

// Allocate memory and call the function
    try (Arena arena = Arena.ofConfined()) {

        // Allocate off-heap memory
        MemorySegment textSegment = arena.allocateFrom(text);
        MemorySegment langSegment = arena.allocateFrom(langCode);

        MemorySegment segment = (MemorySegment) NormalizerSettings.HANDLE.invokeExact(textSegment, langSegment);
        MemorySegment reinterpret = segment.reinterpret(Long.MAX_VALUE);

        return reinterpret.getString(0);
    } // Off-heap memory is deallocated
```

Со статической инициализацией

Benchmark	Mode	Cnt	Score	Error	Units
NormalizeWithBenchTest.normalizeUsingFF	avgt	10	5.086 ±	0.168	us/op
NormalizeWithBenchTest.normalizeUsingJNI	avgt	10	5.306 ±	0.170	us/op
NormalizeWithBenchTest.normalizeUsingJextract	avgt	10	5.213 ±	0.262	us/op
NormalizeWithBenchTest.turnOn10PercentUsingFF	avgt	10	4.854 ±	0.193	us/op
NormalizeWithBenchTest.turnOn10PercentUsingJNI	avgt	10	5.345 ±	0.065	us/op
NormalizeWithBenchTest.turnOn10PercentUsingJextract	avgt	10	4.737 ±	0.067	us/op

ATTENTION

bugs.openjdk.org

bugs.openjdk.org

```
ByteBuffer byteBufferText = ByteBuffer.wrap(text.getBytes());
MemorySegment textSegment = MemorySegment.ofBuffer(byteBufferText);
// OR
// MemorySegment textSegment = MemorySegment.ofArray(text.getBytes());

try (Arena arena = Arena.ofConfined()) {
    MemorySegment langSegment = arena.allocateFrom(langCode);
    MemorySegment segment = (MemorySegment)
    FFNormalizerSettings.HANDLE.invokeExact(textSegment, langSegment);
    MemorySegment reinterpret = segment.reinterpret(Long.MAX_VALUE);
    return reinterpret.getString(0, StandardCharsets.UTF_8);
}
```

```
java.lang.IllegalArgumentException: Heap segment not allowed: MemorySegment{ heapBase:
[B@3a7442c7, address: 0x0, byteSize: 24 }
```

```java
/**
 * {@return a linker option used to mark a foreign function as <em>critical</em>}
 * <p>
 * A critical function is a function that has an extremely short running time in
 * all cases (similar to calling an empty function), and does not call back into
 * Java (e.g. using an upcall stub).
 * <p>
 * Using this linker option is a hint that some implementations may use to apply
 * optimizations that are only valid for critical functions.
 * <p>
 * Using this linker option when linking non-critical functions is likely to have
 * adverse effects, such as loss of performance or JVM crashes.
 * <p>
 * Critical functions can optionally allow access to the Java heap. This allows
 * clients to pass heap memory segments as addresses, where normally only off-heap
 * memory segments would be allowed. The memory region inside the Java heap is
 * exposed through a temporary native address that is valid for the duration of
 * the function call. Use of this mechanism is therefore only recommended when a
 * function needs to do short-lived access to Java heap memory, and copying the
 * relevant data to an off-heap memory segment would be prohibitive in terms of
 * performance.
 *
 * @param allowHeapAccess whether the linked function should allow access to the
 *        Java heap.
 */
static Option critical(boolean allowHeapAccess) {
    return allowHeapAccess
        ? LinkerOptions.Critical.ALLOW_HEAP
        : LinkerOptions.Critical.DONT_ALLOW_HEAP;}
```

github.com

JNI & critical

If possible, the VM returns a pointer to the primitive array; otherwise, a copy is made. However, there are significant restrictions on how these functions can be used.

… Inside a critical region, native code must not call other JNI functions, or any system call that may cause the current thread to block and wait for another Java thread. (For example, the current thread must not call read on a stream being written by another Java thread.)

Ссылка на статью

HANDLE_CRITICAL

```
public static final MethodHandle HANDLE_CRITICAL;
static {
    Linker linker = Linker.nativeLinker();
    HANDLE_CRITICAL = linker.downcallHandle(
            ADDR,
            FUNCTION_DESCRIPTOR,
            // allow access to the Java heap
            Linker.Option.critical(true)
    );
}
```

HANDLE_CRITICAL

```
ByteBuffer byteBufferText = ByteBuffer.wrap(text.getBytes());
MemorySegment textSegment = MemorySegment.ofBuffer(byteBufferText);
// OR
// MemorySegment textSegment = MemorySegment.ofArray(text.getBytes());

try (Arena arena = Arena.ofConfined()) {
    MemorySegment langSegment = arena.allocateFrom(langCode);
    MemorySegment segment = (MemorySegment)
    FFNormalizerSettings.HANDLE_CRITICAL.invokeExact(textSegment, langSegment);
    MemorySegment reinterpret = segment.reinterpret(Long.MAX_VALUE);
    return reinterpret.getString(0);
}
```

Еще один бенч

Benchmark	Mode	Cnt	Score	Error	Units
NormalizeWithBenchTest.longTextUsingJNI	avgt	10	64.043 ±	2.519	us/op
NormalizeWithBenchTest.longTextUsingJextract	avgt	10	63.324 ±	2.055	us/op
NormalizeWithBenchTest.longTextUsingFF	avgt	10	63.849 ±	2.736	us/op
NormalizeWithBenchTest.longTextUsingFFCritical	avgt	10	64.286 ±	3.517	us/op
NormalizeWithBenchTest.testDuckMultilinguaJNI	avgt	10	4.944 ±	0.094	us/op
NormalizeWithBenchTest.testDuckMultilingualJextract	avgt	10	4.831 ±	0.473	us/op
NormalizeWithBenchTest.testDuckMultilingualFF	avgt	10	4.782 ±	0.166	us/op
NormalizeWithBenchTest.testDuckMultilingualFFCritical	avgt	10	4.357 ±	0.155	us/op
NormalizeWithBenchTest.turnOn10PercentUsingJNI	avgt	10	5.345 ±	0.065	us/op
NormalizeWithBenchTest.turnOn10PercentUsingJextract	avgt	10	4.737 ±	0.067	us/op
NormalizeWithBenchTest.turnOn10PercentUsingFF	avgt	10	4.854 ±	0.193	us/op
NormalizeWithBenchTest.turnOn10PercentUsingFFCritical	avgt	10	4.588 ±	0.127	us/op

Неуловимое падение в тесте

uncaught exception:

address -> 0x600000276370

what() -> "util/charset/wide.h:366: failed to decode UTF-8 string at pos 26 in string "\xD0\x9C\xD0\xBE\xD1\x91 \xD0\xB2\xD1\x81\xD1\x91 123 \xD0\x90\xD0\23092\tv\x82\x8A\x15""

type -> yexception

Disconnected from the target VM, address: '127.0.0.1:51106', transport: 'socket'

На почитать:

JEP454

Дока от Oracle

Jextract Guide

Спасибо!

Настя Лисицкая

Яндекс