How to Run C/C++ Code in a Java Environment Using JNI: A Step-by-Step Guide

Neelesh Janga · Follow

5 min read · Jul 11, 2024

If you've ever wanted to combine the power of C/C++ with the versatility of Java, you're in the right place! In this article, we'll walk you through how to run C/C++ code in a Java environment using Java Native Interface

(JNI). Don't worry if you're new to this — we'll break it down step by step.

Step 1: Create a Native Method in Java

First, let's create a simple Java class with a native method. Native methods are Java methods that are implemented in another programming language, like C or C++.

```
package dev.neelesh.jni;

public class JNIDemo {

    // Declare a native method
    public native void sayHello();

    // Load the library that contains the native method implementation
    static {
        System.loadLibrary("jni_demo");
    }

    public static void main(String[] args) {
        JNIDemo demo = new JNIDemo();
        demo.sayHello();
    }
}
```

This loads the native library (in this case, `jni_demo`) into the Java Virtual Machine (JVM). This native library is a dynamic library that contains the implementation of native methods declared in the Java class.

Here's a breakdown of what it does and why it's important:

Why is `System.loadLibrary` Used?

1. **Linking Native Methods:** The `System.loadLibrary` method links the native methods declared in your Java code to their implementations in the native (C/C++) code. Without this, the JVM wouldn't know where to find the native method implementations.
2. **Loading Native Libraries:** It loads the specified native library (a shared library, usually with extensions like `.dll` on Windows, `.so` on Linux, and `.dylib` on macOS) into memory so that the JVM can call the native methods.

How Does It Work?

1. **Library Name:** The argument to `System.loadLibrary` is the name of the

library without the platform-specific prefix (`lib`) and suffix (e.g., `.dll`

a

`jni_demo.dll` on Windows.

2. **Library Path:** The JVM searches for the specified library in the directories listed in the `java.library.path` system property. This is why we run the Java program with the `-Djava.library.path=./bin` option, ensuring the JVM looks in the `./bin` directory for the `libjni_demo.dylib` file. Note that, we are providing `java.library.path` dynamically using VM Arguments.

Step 2: Compile the Java Class and Generate a Header File

Next, compile your Java class and generate a header file using the `javac` command. This header file will be used to write the C code.

```
javac -h . dev/neelesh/jni/JNIDemo.java
```

This command generates a header file named `dev_neelesh_jni_JNIDemo.h`.

The command is named `dev_neelesh_jni_JNIDemo.h` to follow a specific naming convention that ensures uniqueness and helps the JNI (Java Native Interface) system map Java classes and methods to their native counterparts correctly.

Naming Convention Explained

The naming convention for the header file follows the package and class structure of the Java class being compiled. Here's the breakdown:

1. **Package Structure:**

- Java packages are typically separated by dots (.), but for the purpose of the header file name, these dots are replaced with underscores (_).
- For the class `dev.neelesh.jni.JNIDemo`, the package is `dev.neelesh.jni`.

2. **Class Name:**

- The class name `JNIDemo` is appended to the package name, separated by an underscore.

Pu

`dev.neelesh.jni.JNIDemo` translates to the header file name `dev_neelesh_jni_JNIDemo.h`.

Step 3: Create a C File

Now, let's create a C file that implements the native method. Include the necessary headers and use the header file generated in the previous step.

Create a file named `JNIDemo.c` and add the following code:

```c
#include <jni.h>
#include <stdio.h>
#include "dev_neelesh_jni_JNIDemo.h"

// Implementation of the native method
JNIEXPORT void JNICALL Java_dev_neelesh_jni_JNIDemo_sayHello(JNIEnv *env, jobjec
    printf("Hello viewers - This method is written in C language!\n");
}
```

Header Inclusions:

Virtual Machine (JVM) from C/C++.

- `#include <stdio.h>` : This includes the standard input/output header file in C, which provides functions for performing input and output operations like `printf`.
- `#include "dev_neelesh_jni_JNIDemo.h"` : This includes the header file generated by the `javac -h` command. It contains the function prototypes for the native methods declared in the corresponding Java class.

Native Method Implementation:

- `JNIEXPORT void JNICALL` : These are macros defined in `jni.h` to ensure the correct handling of function exports and calling conventions across different platforms and compilers.
- `Java_dev_neelesh_jni_JNIDemo_sayHello` : This is the name of the function implementing the native method. The naming convention follows the pattern:

  `Java_<package>_<ClassName>_<methodName>`

Function Parameters:

NI

- `JNIEnv *env` : A pointer to the JNI environment, which provides access to JNI functions. It's used for interaction with the JVM, such as calling Java methods, accessing fields, and handling exceptions.
- `jobject obj` : A reference to the calling Java object (an instance of the `JNIDemo` class in this case).

Step 4: Compile the C Code into a Dynamic Library

Next, compile your C code into a dynamic library that the Java program can use. Use the `gcc` compiler for this.

```
sudo gcc -dynamiclib -o ./bin/libjni_demo.dylib jni/JNIDemo.c -I${JAVA_HOME}/inc
```

Let’s break down this command:

- `sudo` : Runs the command with superuser privileges.

- `gcc` : The GNU Compiler Collection command for compiling C and C++

- `-dynamiclib` : Specifies that the output should be a dynamic library (shared library).
- `-o ./bin/libjni_demo.dylib` : Specifies the output file name and location.
- `jni/JNIDemo.c` : The path to your C source file.
- `-I${JAVA_HOME}/include` and `-I${JAVA_HOME}/include/darwin` : Specifies the directories to search for JNI header files.

If this command fails, ensure the `bin` directory exists in your current working directory. You can create it with: `mkdir bin` , and try again.

Step 5: Run the Java Program

Finally, run your Java program and see the magic happen!

```
java -Djava.library.path=./bin dev.neelesh.jni.JNIDemo
```

Th he Java Virtual Machine (JVM) to find the dynamic library.

Conclusion

That's it! You've successfully run C/C++ code within a Java environment using JNI. By following these steps, you can leverage the performance and capabilities of C/C++ while enjoying the flexibility of Java.

Resources

1. **GitHub** — https://github.com/Neelesh-Janga/Java-Code-Snippets/tree/main/src/dev/neelesh/jni

Social Accounts

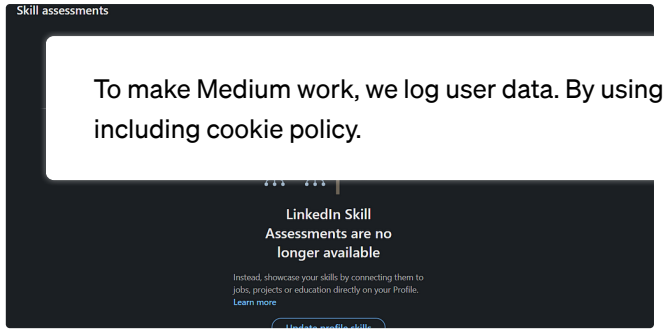
1. **LinkedIn** — https://www.linkedin.com/in/neelesh-janga/
2. **GitHub** — https://github.com/Neelesh-Janga

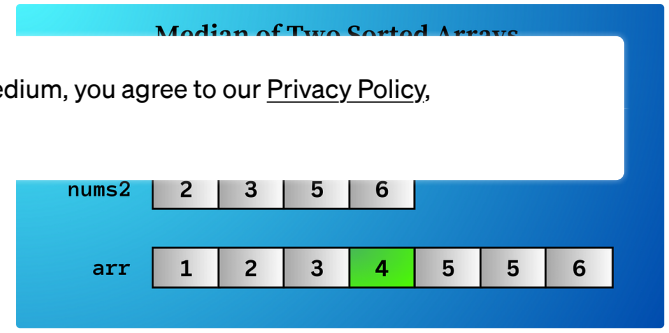
Written by Neelesh Janga

118 Followers

Passionate about Java, Web Development, AI, Coding, DSA, Freelancing , Web development & Security. Always loves to collaborate and work together!!

More from Neelesh Janga





Neelesh Janga

LinkedIn Skill Assessments No Longer Available!

LinkedIn Skill Assessment Badges are a feature on the LinkedIn platform designed t...

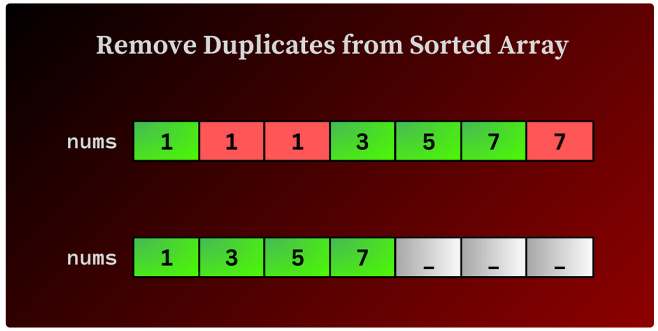
Dec 21, 2023 18 2

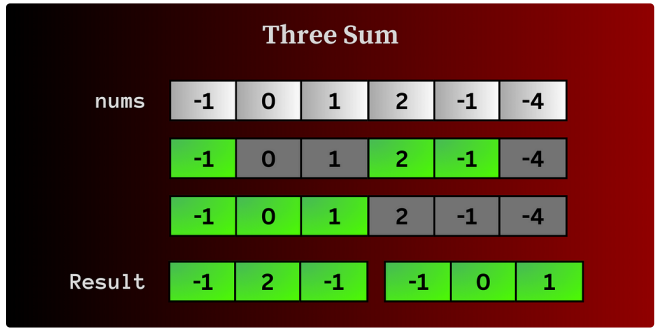
Neelesh Janga

Q-4 LeetCode: Finding Median of Two Sorted Arrays Using Java

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median...

Nov 23, 2023





Neelesh Janga

Q-
Du

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-...

Dec 5, 2023 4

Neelesh Janga

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that...

Dec 1, 2023 4

See all from Neelesh Janga

Recommended from Medium

Hùng Trần in Javarevisited

St
Ja

The tale goes that there is an ancient legacy system, my colleague coded thousands of...

Aug 20 1.3K 39

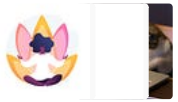
Rabinarayan Patra

Ever wondered why comparing 1==1 returns true, but 128==128 returns false in Java? Let'...

Sep 11 1.8K 47

Lists

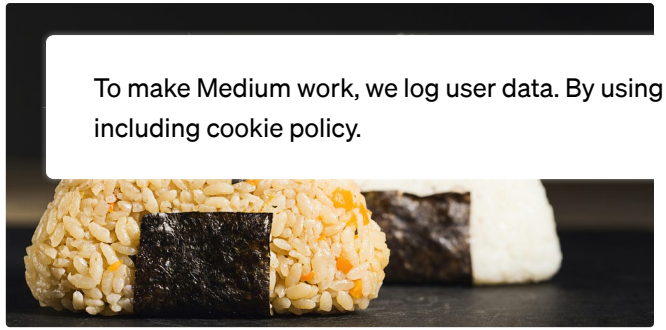
General Coding Knowledge
20 stories · 1666 saves



Stories to Help You Grow as a Software Developer
19 stories · 1435 saves

Coding & Development
11 stories · 858 saves

Good Product Thinking
12 stories · 728 saves

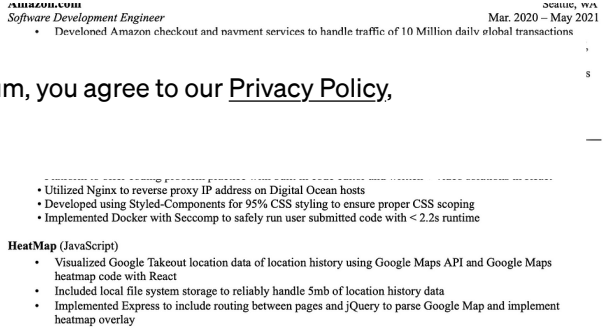


Full Stack Developer

Spring doesn't recommend @Autowired anymore???

There are 3 ways we can inject dependencies and Field Injection is not recommended...

Jul 19 99 4

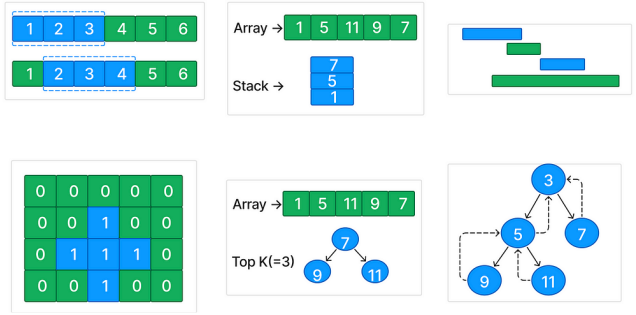


Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 24K 485



Oliver Foster in Stackademic

W
re

My article is open to everyone; non-member readers can click this link to read the full text.

Sep 16 2K 60

Ashish Pratap Singh in AlgoMaster.io

I have solved more than 1500 LeetCode problems, and if there is one thing I have...

Aug 23 2.3K 22

See more recommendations