《深入理解 JNI》：JNI 基础

JNI 简介

Java Native Interface（JNI）是Java平台的一个标准接口，它允许Java代码与其他语言编写的代码进行交互。这种交互能力极大地扩展了Java的应用范围，使得Java程序可以调用系统级的库或者执行高性能计算，这些往往是纯Java代码难以高效完成的。

从Java 1.1版本开始，JNI标准就成为Java平台的一部分。JNI最初的设计目的是为了本地已编译语言，特别是C和C++，与其他语言的交互也只要调用约定受支持即可，并非局限于这两种语言。

JNI在整个Java运行体系中的工作原理类似一个中间人。Java代码编译后生成字节码，在Java虚拟机（JVM）中运行。当Java程序需要调用本地代码（如C或C++编写的函数）时，通过JNI提供的机制来进行。同样，本地代码如果需要与Java程序交互，例如调用Java中的方法或者操作Java对象，也是借助JNI来实现的。

在调用本地方法时，Java虚拟机会提供JNI接口指针给本地方法。这个接口指针对于本地方法的执行非常关键，它允许本地方法访问JVM内部的数据结构，如Java对象的引用、类信息等。并且，虚拟机会保证在从相同的Java线程中对本地方法进行多次调用时，传递给本地方法的这个接口指针是相同的；而当本地方法被不同的Java线程调用时，它接受的是不同的JNI接口指针。

使用JNI可以带来很多好处。例如，当需要与旧的库进行交互时，这些库可能是用C或C++编写的，如果重新用Java实现这些库的功能可能成本很高，而JNI就提供了一种方便的方式来调用这些现有库。在与硬件进行交互的场景中，硬件操作通常需要更底层的控制，本地代码在这方面更擅长，通过JNI可以很好地让Java程序利用这些硬件操作。另外，对于一些对性能要求极高的场景，某些算法用C或C++实现可能比Java实现更高效，JNI使得Java能够调用这些高性能的本地实现。

然而，JNI的使用也存在一些挑战。最大的一个挑战就是平台可移植性的丧失。由于本地代码是针对特定平台编译的，如果要在不同平台上运行包含JNI调用的Java程序，就需要为每个平台提供相应的本地库，这无疑增加了开发和维护的成本。不过，在考虑到一些特殊需求（如前面提到的与旧库交互、硬件交互和高性能计算）时，这种可移植性的代价是在可接受范围之内的。

JNI为Java与其他语言的交互提供了一种标准化的、通用的方法，使得开发者能够在Java的跨平台优势和其他语言在特定功能上的优势之间找到平衡，进而构建出功能更强大、性能更优化的应用程序。

环境搭建

在使用JNI进行开发之前，需要确保相关的工具和环境正确安装和配置。以下是搭建JNI开发环境的关键步骤：

安装Java Development Kit (JDK)

JDK是Java开发的核心工具包，它包含了Java编译器、Java运行时环境（JRE）以及其他开发工具。JNI依赖于JDK提供的头文件和库文件，因此必须首先安装JDK。

- 下载JDK

  ：访问Oracle官方网站或其他JDK提供商的网站，下载适合你操作系统的JDK版本。

- 安装JDK

  ：按照下载的安装包提示完成JDK的安装。

- 配置环境变量

  ：安装完成后，需要配置系统的环境变量，确保Java的可执行文件路径被添加到PATH环境变量中。同时，设置JAVA_HOME环境变量指向JDK的安装目录。

安装Native Development Kit (NDK)

NDK是一套工具集，允许开发者使用C和C++编写代码，并将其编译成Android平台上的原生库。如果你的JNI项目是为Android平台开发的，那么安装NDK是必要的。

- 下载NDK

  ：可以通过Android Studio的SDK Manager来下载NDK，或者直接从Android开发者官方网站下载。

- 解压NDK

  ：下载完成后，解压NDK到一个合适的目录。

配置Android Studio（如果适用）

如果你是在Android平台上进行JNI开发，那么Android Studio是一个很好的集成开发环境（IDE）选择。

- 安装Android Studio

  ：如果尚未安装，从Android开发者官方网站下载并安装Android Studio。

- 配置NDK路径

  ：在Android Studio中，通过SDK Manager设置NDK的路径。

- 创建新项目

：在创建新的Android项目时，可以选择包含C++支持的模板，这样Android Studio会自动配置好JNI相关的设置。

安装C/C++编译器

对于非Android平台的JNI开发，或者需要自行编译本地库的情况，你需要安装C或C++编译器。

- Windows

  ：可以使用MinGW或者Microsoft Visual C++ Build Tools。

- macOS

  ：Xcode命令行工具包含了Clang编译器。

- Linux

  ：大多数发行版的包管理器都提供了GCC编译器。

验证安装

安装和配置完成后，可以通过简单的命令来验证环境是否搭建成功。

验证Java安装：

```
java -version
javac -version
```

这两个命令应该显示已安装的Java版本信息。

验证C/C++编译器安装** (以GCC为例)

```
gcc --version
```

或者在Windows上：

```
g++ --version
```

这些命令应该显示已安装的编译器版本信息。

编写和运行第一个JNI程序

环境搭建完成后，可以尝试编写一个简单的JNI程序来验证一切是否正常工作。这个程序通常包括一个Java类，该类声明了一个本地方法，然后有一个对应的C或C++文件实现这个本地方法。编译Java类生成头文件，然后编写C/C++代码实现头文件中声明的函数，编译生成共享库，最后运行Java程序加载共享库并调用本地方法。

通过以上步骤，你应该能够成功搭建起JNI的开发环境，并准备好开始进行JNI编程。记住，JNI编程需要对Java和C/C++都有一定的了解，因为你需要在这两种语言之间进行交互。

第一个JNI示例

现在我们将通过一个简单的示例来了解JNI的基本工作流程。这个示例将展示如何从Java代码中调用一个C函数。

编写Java类

首先，我们需要创建一个Java类，其中包含一个本地方法声明。本地方法使用`native`关键字标记，但不需要提供方法体。

```
public class HelloWorldJNI {
   // 声明本地方法
   public native void sayHello();

   // 加载包含本地方法实现的共享库
   static {
      System.loadLibrary("hello-jni");
   }

   // 主方法，用于测试本地方法
   public static void main(String[] args) {
      new HelloWorldJNI().sayHello();
   }
}
```

在上面的代码中，`sayHello`是一个本地方法，它将在C代码中实现。`System.loadLibrary`方法用于加载包含本地方法实现的共享库。库的名称是`hello-jni`，但在实际的系统中，这个名称可能会根据操作系统的不同而有所变化（例如，在Windows上可能是`hello-jni.dll`，在Linux上可能是`libhello-jni.so`）。

生成头文件

使用`javac`命令编译Java类，然后使用`javah`工具生成C语言的头文件。`javah`工具会根据Java类中的本地方法声明生成相应的C函数原型。

```
javac HelloWorldJNI.java
javah HelloWorldJNI
```

这将生成一个名为`HelloWorldJNI.h`的头文件，其中包含了`sayHello`方法的C语言声明。

实现本地方法

接下来，我们需要编写C代码来实现头文件中声明的函数。创建一个C文件，例如`hello-jni.c`，并实现`sayHello`函数。

```
#include <jni.h>
#include <stdio.h>
#include "HelloWorldJNI.h"

JNIEXPORT void JNICALL Java_HelloWorldJNI_sayHello(JNIEnv *env, jobject thisObj) {
    printf("Hello from C!₩n");
}
```

在这个C函数中，我们使用了`JNIEXPORT`和`JNICALL`宏来确保函数可以被Java虚拟机正确调用。`JNIEnv *env`参数提供了访问JNI函数表的指针，而`jobject thisObj`参数是对调用该本地方法的Java对象的引用。

编译共享库

现在我们需要编译C代码并生成共享库。这个步骤会根据操作系统的不同而有所变化。

在Linux上：

```
  gcc -shared -fPIC -o libhello-jni.so hello-jni.c -I${JAVA_HOME}/include -I${JAVA_HOME}/include/linux
```

在macOS上：

```
 gcc -dynamiclib -o libhello-jni.dylib hello-
jni.c -I${JAVA_HOME}/include -I${JAVA_HOME}/include/darwin
```

在Windows上：
使用MinGW或Visual Studio的命令行工具来编译生成DLL文件。

运行Java程序

最后，我们可以运行Java程序来测试本地方法。确保共享库位于Java的库路径中，然后运行Java类。

```
java HelloWorldJNI
```

如果一切设置正确，你应该会在控制台看到输出：

```
Hello from C!
```

这个简单的示例展示了JNI的基本使用流程。通过这个过程，你可以看到Java如何调用C代码，以及如何在这两种语言之间进行交互。在实际的项目中，JNI的使用可能会更加复杂，涉及到更多的数据类型转换、对象操作和异常处理等。但这个基础示例为理解JNI的工作原理提供了一个良好的起点。