JNI programming introduction

출처 : 1. The Java™ Native Interface - Programmer's Guide and Specification

2. JNI:JAVA와 C++의 연동

저자 : 1. Sheng Liang

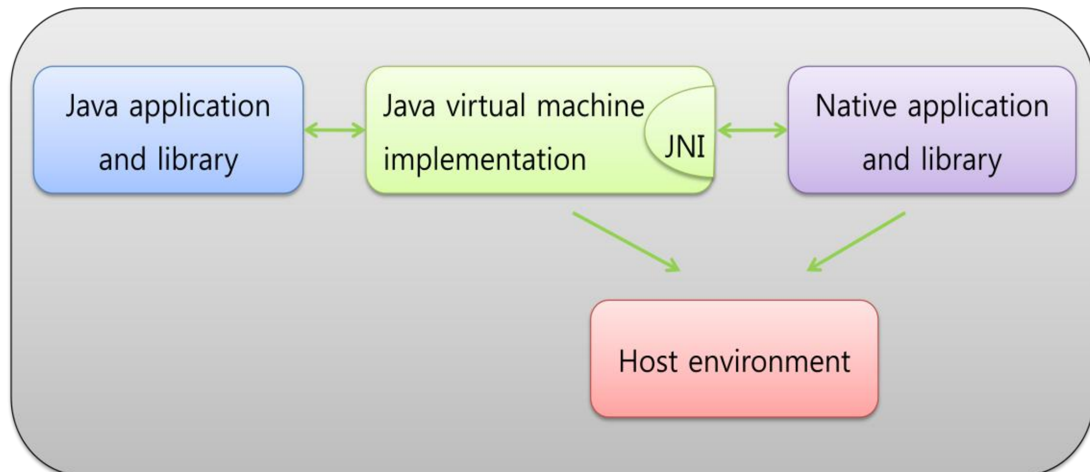
2. Frank Yoon(moses@maru.net)

번역 및 추가: danguria (danguria@gmail.com)

테스트 환경: Ubunto, java1.6, g++, vim

1. What is the JNI
   A. What is the JNI?

      JNI는 자바 어플리케이션과 네이티브 코드가 서로 서로 실행 할 수 있게 해주는 인터페이스를 말합니다.



<Role of JNI>

   B. Why, When to Use the JNI?

      다음은 JNI를 사용하는 이유와 사용해야 하는 경우입니다.

      i. 해당 호스트에 의존적의 기능이라서 Java API에서 제공하지 않는 경우에 사용합니다. 예를 들어 자바에서 제공하지 않는 파일 처리 오퍼레이션 같은 경우 JNI를 통해 사용하면 훨씬 효율 적입니다.
      ii. 기존에 존재하는 네이티브 라이브러리를 자바언어로 다시 재 작성해야 하는 경우, JNI를 통해 기존의 라이브러리를 그대로 사용하는 것이 비용 면에서 훨씬 효율 적입니다.
      iii. 그래픽 라이브러리와 같은 time-critical한 기능을 구현할 때 JNI를 사용하면 속도개선을 할 수 있습니다.

하지만, 장점이 있는 반면 단점도 존재 합니다. 다음은 JNI를 사용하게 되면서 잃게 되는 사항들입니다.

i. Portability

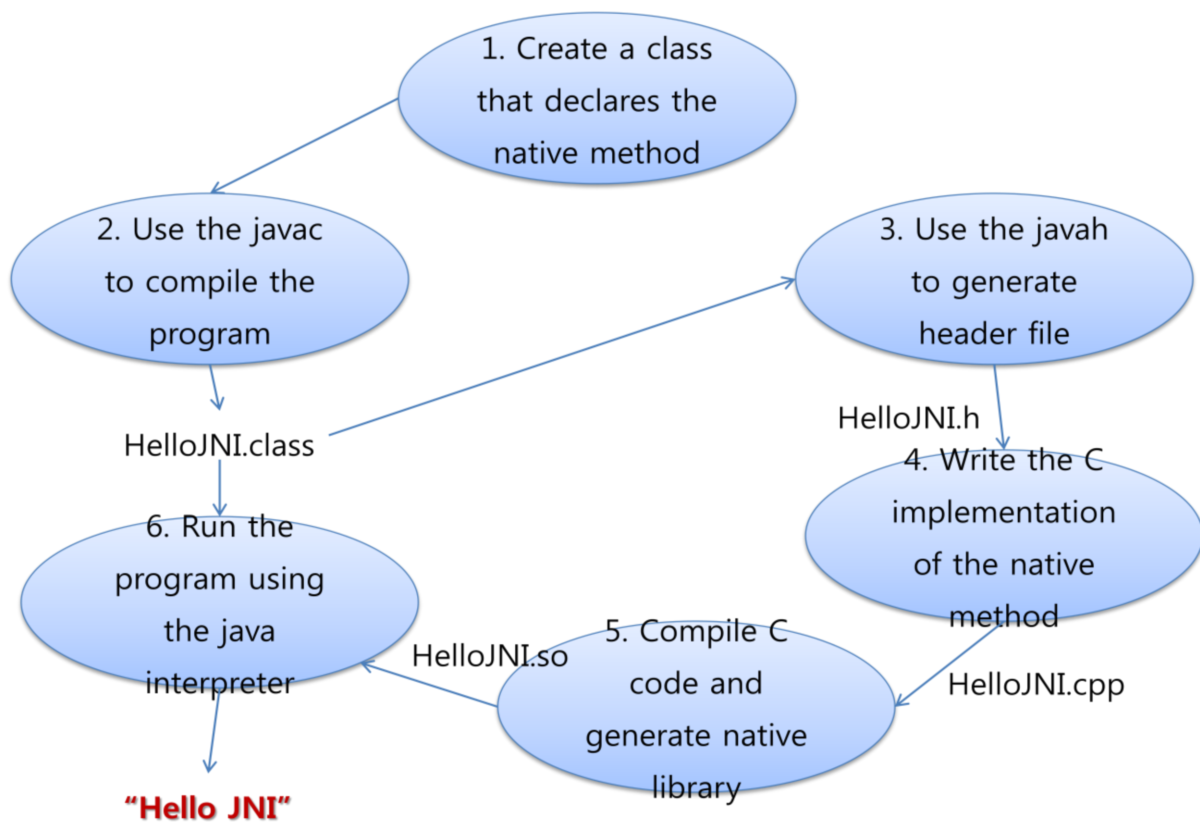
순수한 자바 코드가 호스트에 의존적인 코드와 결합하게 되면서 이식성을 잃게 됩니다.

ii. Safety

자바의 강력한 기능중의 하나인 type checking, garbage collection과 같은 안정성을 잃게 됩니다.

2. Practice JNI

A. Overview



<Steps in Writing and Running the "Hello World" Program>

B. Declare the native method

native 메소드를 포함하는 HelloJNI클래스를 만듭니다.

```
1 public class HelloJNI{
2     native public static void helloJNI();
3
4     static {
5         System.loadLibrary("HelloJNI");
6     }
7 }
```

HelloJNI.java 파일을 모두 작성한 다음 다음과 같이 컴파일하며 HelloJNI.class파일을 생성합니다.

C. Compile the HelloJNI Class

javac컴파일러를 이용하여 이전 단계에서 만든 HelloJNI.java 파일을 컴파일 합니다.

```
javac HelloJNI.java
```

이 결과 같은 디렉터리 안에 HelloJNI.class파일이 생성될 것입니다.

D. Create the Native Method Header File

HelloJNI 클래스에서 만든 native 메소드를 구현하기 위해서 native(C/C++영역)에서는 시그너처를 알아야 합니다. 우리는 컴파일 과정을 알 수 없기 때문에 javah라는 툴이 native 메소드의 시그너처를 알려 주고, native(C/C++영역 이하 생략)에서 사용할 prototype을 포함한 해더 파일을 생성해 줍니다.

다음의 명령어를 통해 해더 파일을 생성할 수 있습니다.

```
javah HelloJNI
```

이 명령어를 실행하기 위해서 HelloJNI.class가 같은 폴더에 있어야 합니다.

생성된 해더 파일을 열어보면 다음과 같은 코드가 들어 있습니다.

```
10  /*
11  * Class:      HelloJNI
12  * Method:     helloJNI
13  * Signature: ()V
14  */
15 JNIEXPORT void JNICALL Java_HelloJNI_helloJNI
16    (JNIEnv *, jclass);
```

주석 처리 된 부분을 보면, HelloJNI 클래스에서 작성한 helloJNI 메소드의 시그너처 정보가 있는 것을 알 수 있습니다. 이 시그너처는 native에서 자바의 클래스, 메소드, 필드 등을 접근할 때 사용되는 중요한 정보입니다.
주석 아래에는 native에서 사용할 helloJNI함수의 prototype이 있습니다.
함수의 naming 규칙은 **Java_클래스명_메소드명** 입니다. javah툴이 자동으로 생성해 주기 때문에 크게 신경 쓰지 않아도 됩니다.

E. Write the native method implementation

이전 단계에서 만든 헤더파일을 이용하여 helloJNI함수를 구현합니다. 다음과 같이 구현합니다.

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 #include "HelloJNI.h"
 5
 6
 7
 8 /*
 9  * Class:       HelloJNI
10  * Method:      helloJNI
11  * Signature: ()V
12  */
13 JNIEXPORT void JNICALL Java_HelloJNI_helloJNI
14    (JNIEnv *env, jclass cls)
15 {
16       cout << "Hello JNI" << endl;
17 }
```

helloJNI함수는 간단하게 “Hello JNI” 문자열을 출력하는 기능을 합니다. 실제 프로그램을 만들 때는 여기에 개발자가 원하는 코드를 넣으면 됩니다. 복잡한 기능을 제공하는 native 메소드의 구현은 The Java™ Native Interface - Programmer's Guide and Specification를 참고하세요.

F. Compile the CPP source and create a native library

"Declare the native method"단계에서 작성한 HelloJNI 클래스를 보면 다음과 같은 코드가 있습니다.

```
static {
    System.loadLibrary("HelloJNI");
}
```

loadLibrary함수는 native 라이브러리를 프로그램에 로딩하는 함수입니다. Native 라이브러리를 만들기 위해서 다음과 같은 명령를 실행 합니다.

```
g++ -c -fPIC -I$JAVA_HOME/include -I$JAVA_HOME/include/linux HelloJNI.cpp
g++ -shared -o libHelloJNI.so HelloJNI.o
```

위의 명령어는 동적 라이브러리를 만드는 명령어 입니다. 자세한 사항은 저의 "공유 라이브러리 만들기" 스프링노트를 참고 하세요.

G. Create Test Program and Run

HelloJNI 클래스를 테스트 하기 위한 HelloJNITest 클래스를 작성하여 잘 만들었는지 확인해 보도록 합니다. 다음과 같이 HelloJNITest.java 파일을 작성합니다.

```
1
2 public class HelloJNITest{
3       public static void main(String[] args){
4             HelloJNI.helloJNI();
5       }
6 }
```

마지막으로 HelloJNITest.java 파일을 컴파일하고 실행 시킵니다.

```
javac HelloJNITest.java
java HelloJNITest
```