JNI/NDK 开发指南（六）——C/C++ 访问 Java 实例方法和静态方法

作者：woyehua

原文链接：https://ld246.com/article/1541603772855

来源网站：链滴

通过前面5章的学习，我们知道了如何通过JNI函数来访问JVM中的基本数据类型、字符串和数组这些据类型。下一步我们来学习本地代码如何与JVM中任意对象的属性和方法进行交互。比如本地代码调Java层某个对象的方法或属性，也就是通常我们所说的来自C/C++层本地函数的callback（回调）。个知识点分2篇文章分别介绍，本篇先介绍方法回调，在第七章中介绍本地代码访问Java的属性。

在这之前，先回顾一下在Java中调用一个方法时在JVM中的实现原理，有助于下面讲解本地代码调用ava方法实现的机制。写过Java的童鞋都知道，调用一个类的静态方法，直接通过 类名.方法 就可以用。这也太简单了，有什么好讲的呢。。。但在这个调用过程中，JVM是帮我们做了很多工作的。当们在运行一个Java程序时，JVM会先将程序运行时所要用到所有相关的class文件加载到JVM中，并采按需加载的方式加载，也就是说某个类只有在被用到的时候才会被加载，这样设计的目的也是为了提程序的性能和节约内存。所以我们在用类名调用一个静态方法之前，JVM首先会判断该类是否已经加，如果没有被ClassLoader加载到JVM中，JVM会从classpath路径下查找该类，如果找到了，会将其载到JVM中，然后才是调用该类的静态方法。如果没有找到，JVM会抛出java.lang.ClassNotFoundException异常，提示找不到这个类。ClassLoader是JVM加载class字节码文件的一种机制，不太了解童鞋，请移步阅读《深入分析Java ClassLoader原理》一文。其实在JNI开发当中，本地代码也是按上面的流程来访问类的静态方法或实例方法的，下面通过一个例子，详细介绍本地代码调用Java方法程当中的每个步聚：

```
package com.study.jnilearn;

/**
 * AccessMethod.java
 * 本地代码访问类的实例方法和静态方法
 * @author yangxin
 */
public class AccessMethod {
public static native void callJavaStaticMethod();
```

```
public static native void callJavaInstaceMethod();
public static void main(String[] args) {
callJavaStaticMethod();
callJavaInstaceMethod();
}
static {
System.loadLibrary("AccessMethod");
}
}
```

```
package com.study.jnilearn;

/**
● ClassMethod.java
● 用于本地代码调用
● @author yangxin
*/
public class ClassMethod {
```

```
private static void callStaticMethod(String str, int i) {
System.out.format("ClassMethod::callStaticMethod called!-->str=%s," +
" i=%d\n", str, i);
}
private void callInstanceMethod(String str, int i) {
System.out.format("ClassMethod::callInstanceMethod called!-->str=%s, " +
"i=%d\n", str, i);
}
}
```

由AccessMethod.class生成的头文件：

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

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

```
/*
 * Class:     com_study_jnilearn_AccessMethod
 * Method:    callJavaStaticMethod
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_study_jnilearn_AccessMethod_callJavaStaticMethod
(JNIEnv *, jclass);

/*
 * Class:     com_study_jnilearn_AccessMethod
 * Method:    callJavaInstaceMethod
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_study_jnilearn_AccessMethod_callJavaInstaceMethod
(JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

18
19
20
21
22
23
24
25
26
27
28
29

本地代码对头文件中函数原型的实现：

```
// AccessMethod.c

#include "com_study_jnilearn_AccessMethod.h"

/*
● Class:     com_study_jnilearn_AccessMethod
● Method:    callJavaStaticMethod
● Signature: ()V
*/
JNIEXPORT void JNICALL Java_com_study_jnilearn_AccessMethod_callJavaStaticMethod
(JNIEnv *env, jclass cls)
{
jclass clazz = NULL;
jstring str_arg = NULL;
jmethodID mid_static_method;
// 1、从classpath路径下搜索ClassMethod这个类，并返回该类的Class对象
clazz =(*env)->FindClass(env,"com/study/jnilearn/ClassMethod");
if (clazz == NULL) {
return;
}
// 2、从clazz类中查找callStaticMethod方法
mid_static_method = (*env)->GetStaticMethodID(env,clazz,"callStaticMethod","(Ljava/lang/Strng;I)V");
if (mid_static_method == NULL) {
printf("找不到callStaticMethod这个静态方法。");
```

```
return;
}
// 3、调用clazz类的callStaticMethod静态方法
str_arg = (*env)->NewStringUTF(env,"我是静态方法");
(*env)->CallStaticVoidMethod(env,clazz,mid_static_method, str_arg, 100);
// 删除局部引用
(*env)->DeleteLocalRef(env,clazz);
(*env)->DeleteLocalRef(env,str_arg);
}

/*
● Class:     com_study_jnilearn_AccessMethod
● Method:    callJavaInstaceMethod
● Signature: ()V
*/
JNIEXPORT void JNICALL Java_com_study_jnilearn_AccessMethod_callJavaInstaceMethod
(JNIEnv *env, jclass cls)
{
jclass clazz = NULL;
jobject jobj = NULL;
jmethodID mid_construct = NULL;
jmethodID mid_instance = NULL;
jstring str_arg = NULL;
// 1、从classpath路径下搜
```