1. 找到字节码(第一个参数env,第二个参数，要找到的字节码对应类的路径)

jclass clazz = (\*env)->FindClass(env,"包名/类名")

使用C++,JNI中的函数声明：

jclass (\*FindClass)(JNIEnv\*, **const char**\*);

**typedef** \_jclass\* jclass;

**class** \_jclass : **public** \_jobject {};

1. 找到方法(获取java方法签名，可以用javap -s )

jmethodId methoID = (\*env)->GetMethodId(env,clazz,"helloFromJava","()V");

方法签名，如果是一个非静态数据类型，就得把整个路径都写上，并且在最前面加上L，例如String类型的写法是: Ljava/lang/String

1. 创建对象(如果被反射调用的方法与native方法在同一个类中，这一步就是可选的，直接使用传进来的jobject做为对象调用)
2. 通过对象调用方法

(

第一个参数，JNIEnv

第二个参数,要调用方法的对象

第三个参数,要调用的方法的methodID变量

... 可变的参数，调用方法时有参数，那就是后面可变参数要传入的内容，本例中反调调用的方法没有参数

并且被调用的方法，返回值是void，所以也没有接收返回值

)

(\*env)->CallVoidMethod(env,thiz,methoID);

回想下正常的方法调用，

如果是对象调用，需要先拿到类对象，静态方法调用，需要先拿到类名，对应的是第3步

与正常的方法调用，不同的是，多出来了前两步，需要拿到jmethodId ,拿它，又需要先拿到类的字节码，所以，想想，也能理清反射调用的整个流程了。

好了，使用C语言测试没问题，那么用C++来调用java层的静态方法试下

先在java层写个静态方法

public static void helloFromJava()

{

Log.*i*("tag","这里是java层的helloFromJava方法.");

}

C++写的so里面

Java\_com\_jniStudy\_JniTest\_getString(

JNIEnv \*env,

jclass obj, /\* this \*/

jstring js

) {

jclass clazz = env->FindClass("com/jniStudy/JniTest"); //拿到类在内存中的指针? 这里有个小坑，包名类名中间不能用.而用/

jmethodID methoID = env->GetStaticMethodID(clazz, "helloFromJava", "()V"); //拿到的方法结构体指针?

env->CallStaticVoidMethod(obj, methoID);

}

这里有几个小坑需要注意

1、so里方法的第二个参数，由于是反射调用静态方法，所以需要拿到的是类名 jclass

2、C++写的so，不需要再传入环境指针env

3、反射调用静态方法，所以需要使用GetStaticMethodID，CallStaticVoidMethod这两个static方法

补充一下第3步，创建对象用到的方法

jobject obj = env->AllocObject(env,clazz);