如果找native方法的Hotspot源码

有道云链接：http://note.youdao.com/noteshare?id=22527f9eeee8d95f1a0145908ad096dd&sub=AFCD50977A5F4891964CE8C96BBBCD6B

hello，大家好，我是江湖人送外号[道格牙]的子牙老师。
大家平时在看jdk源码的时候，是不是看着看着，总是会凑不及防地遇到native方法，然后就束手无策了。

今天我就教大家如何精准定位到Java方法对应的C++代码、如何高效研究Hotspot源码，甚至！教大家如何修改Hotspot源码，拓展反射API，为我所用！

怎么找

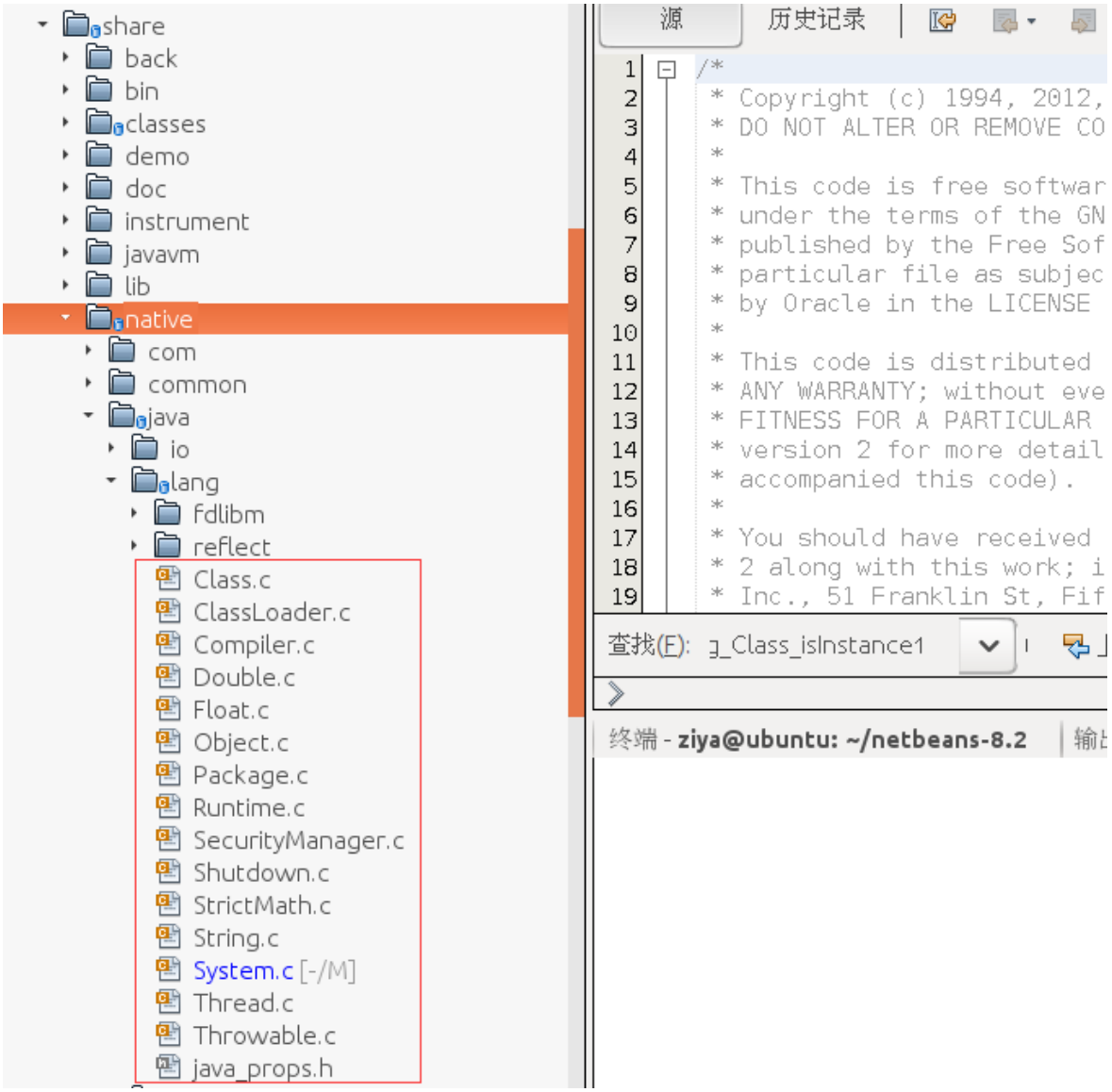
就拿线程的start方法为例吧

```
private native void start0();
```

一、定位文件

如果是系统的native方法，都是很有规律的

start0是Thread类中的方法，Thread类在jdk中有其对应的.c文件：/openjdk/jdk/src/share/native/java/lang/Class.c



有没有看到很多你熟悉的Java类？比如System.c，里面的就是System.java中的所有native方法。

二、找方法

系统提供的JNI模块注册native方法有两种方式，所以找的话也有两种情况：

1. 直接调用JVM模块中的方法，在每个.c文件的头部就可以找到

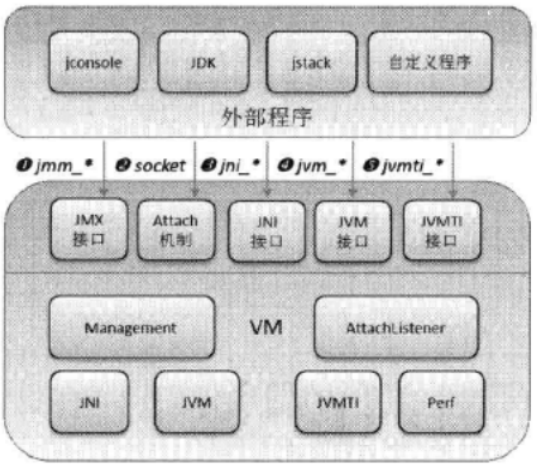
```
static JNINativeMethod methods[] = {
    {"start0",           "()V",        (void *)&JVM_StartThread},
    {"stop0",            "(" OBJ ")V", (void *)&JVM_StopThread},
    {"isAlive",          "()Z",        (void *)&JVM_IsThreadAlive},
    {"suspend0",         "()V",        (void *)&JVM_SuspendThread},
    {"resume0",          "()V",        (void *)&JVM_ResumeThread},
    {"setPriority0",     "(I)V",       (void *)&JVM_SetThreadPriority},
    {"yield",            "()V",        (void *)&JVM_Yield},
    {"sleep",            "(J)V",       (void *)&JVM_Sleep},
    {"currentThread",    "()" THD,     (void *)&JVM_CurrentThread},
    {"countStackFrames", "()I",        (void *)&JVM_CountStackFrames},
    {"interrupt0",       "()V",        (void *)&JVM_Interrupt},
    {"isInterrupted",    "(Z)Z",       (void *)&JVM_IsInterrupted},
    {"holdsLock",        "(" OBJ ")Z", (void *)&JVM_HoldsLock},
    {"getThreads",        "()[" THD,   (void *)&JVM_GetAllThreads},
    {"dumpThreads",      "([" THD ")[[" STE, (void *)&JVM_DumpThreads},
    {"setNativeName",    "(" STR ")V", (void *)&JVM_SetNativeThreadName},
};
```

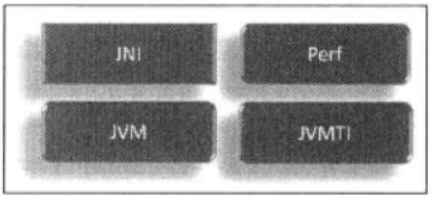
2、JNI模块中定义一个方法，直接找

```
JNIEXPORT void JNICALL
Java_java_lang_System_setIn0(JNIEnv *env, jclass cla, jobject stream)
{
    jfieldID fid =
        (*env)->GetStaticFieldID(env,cla,"in","Ljava/io/InputStream;");
    if (fid == 0)
        return;
    (*env)->SetStaticObjectField(env,cla,fid,stream);
}

JNIEXPORT void JNICALL
Java_java_lang_System_setOut0(JNIEnv *env, jclass cla, jobject stream)
{
    jfieldID fid =
        (*env)->GetStaticFieldID(env,cla,"out","Ljava/io/PrintStream;");
    if (fid == 0)
        return;
    (*env)->SetStaticObjectField(env,cla,fid,stream);
}
```

3、你得理解这张远古图，才能理解JVM各模块的位置及联系





怎么读

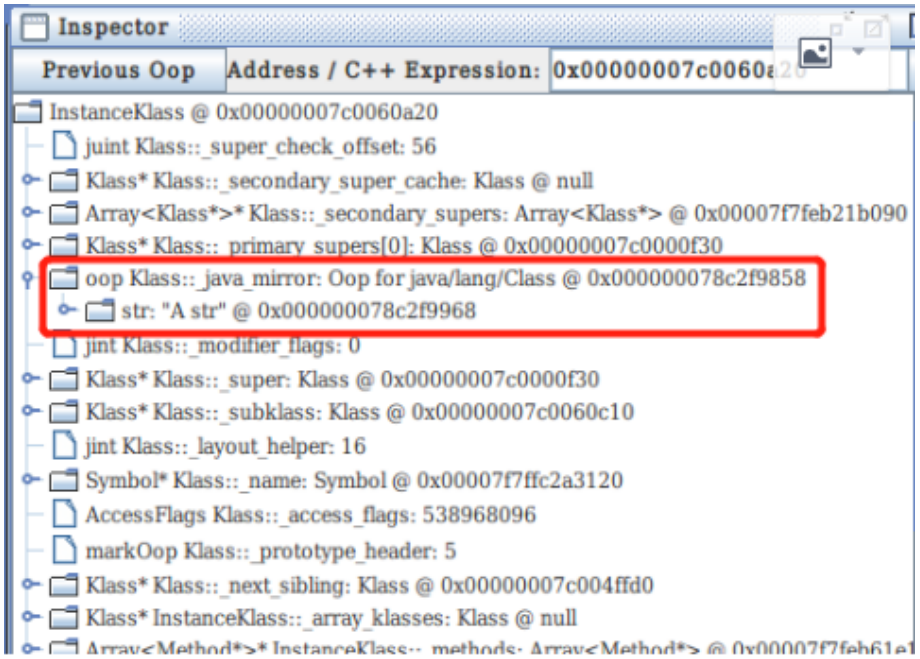
找到native方法对应的Hotspot源码是第一步，接下来就是如何读懂的问题。

Hotspot主要是用C++编写的，所以掌握C++是必要的。对于大家来说，C++最难的应该就是它丰富的语法糖及万恶之源指针了。这些知识，只能通过做项目才能熟练掌握及深入理解。所以掌握C++是基础，你还得有C++的项目开发经验，否则你还是看不懂Hotspot源码。做什么项目？

肯定不是写写常见的数据结构及算法，你得写一些与Hotspot相关的小项目，比如OOP机制、内存池、垃圾收集算法...

掌握了C++，有C++项目开发经验就可以了吗？还不够！你还得对JVM底层原理有深入的理解。这个理解不是看周志明的《深入理解Java虚拟机》就足够的，你得去看一些讲Hotspot源码级别的书籍，做到真正的理解才足够。这边给大家推荐两本：深入Hotspot、解密Java虚拟机底层原理与实现。如果你懒得找，关注公众号【硬核子牙】回复【Hotspot书】获取。

除此之外，还得掌握HSDB这款工具的使用，能熟练地用它去查找JVM内部找到你想要的数据。比如静态属性到底是在堆区还是方法区这样的问题，就可以通过HSDB查看Java类映射的Klass对象，然后通过查看它的属性得到答案。这款工具不知道怎么用？我之前讲的JVM底层原理中有演示。关注公众号【硬核子牙】回复【Hotspot书】获取。



再说下研究Hotspot源码的顺序：先把JVM的启动流程整个看一遍，这时候会碰到很多不知道干啥有啥用的类、不知道为什么存在的流程...不用管，理清主线就可以了。第二步就是去看JVM是如何执行main方法的，这个流程包含类加载的流程及JVM执行方法的流程，同样会遇到各种看不懂。不要灰心，理清主线了解个大概即可。然后就是求甚解的阶段，基于对JVM底层原理的理解仔细读类加载流程、内存初始化、垃圾收集器与内存如何构建起的桥梁、模板引擎执行流生成、封装继承多态的实现原理...这个过程可能会很久很久，所以这个过程不能急，慢慢啃。

读源码需要有单步调试环境，因为你需要看调用链路、变量赋值、指针指向，甚至是运行时代码区。我之前分享过如何搭建单步调试openjdk环境（链接在文末）。相关的软件建议与我的环境保持一致，不然你可能需要花很长时间踩坑。关注公众号【硬核子牙】回复【单步调试】获取相关软件。

黑科技

native方法也找到了，我又是C++大佬，我改怎么验证我的想法及猜测呢？改代码呗！改完后给Java提供调用API呗！JNI行吗？不行！JNI能做的事情都被Hotspot限制死了。你想做的一些事情，只有修改Hotspot源码这一招。我给大家分享一招最简单的。咱们就拓展反射的API吧。

一、增加native方法

/openjdk/jdk/src/share/classes/java/lang/Class.java

```
public native boolean isInstance1(Object obj);
```

二、实现native方法

/openjdk/jdk/src/share/native/java/lang/Class.c

```
JNIEXPORT jboolean JNICALL
Java_java_lang_Class_isInstance1(JNIEnv *env, jobject cls, jobject obj)
{
    if (obj == NULL) {
        return JNI_FALSE;
    }
    return (*env)->IsInstanceOf(env, obj, (jclass)cls);
}
```

三、让编译器知道我增加了方法

/openjdk/jdk/make/mapfiles/libjava/mapfile-vers

```
# Define public interface.

SUNWprivate_1.1 {
        global:
    ......
```

Java_java_lang_Class_isInstance1;

.....

四、编译

```
sudo make all DISABLE_HOTSPOT_OS_VERSION_CHECK=OK ZIP_DEBUGINFO_FILES=0
```

五、运行

```
7  public class Test_1 {
8
9      public static void main(String[] args) {
10         Class clazz = Test_1.class;
11
12         System.out.println(clazz.isInstance1(new Object()));
13         System.out.println(clazz.isInstance1(new Test_1()));
14     }
```

```
Run:  Test_1 (1)
/home/ziya/Documents/openjdk/build/linux-x86_64-normal-server-slo
false
true
```

OK，东风已就位，可以踏上探究Hotspot源码之旅了。