Java，C/C++都是目前主流的编程语言，其互操作Java Native Interface (JNI)技术也被大部分程序员熟知。在Android开发中，存在大量的Java/C++语言互操作调用，我们以Android支持的Java语言标准（java8）和运行Java的ART虚拟机为语言实现，进行原理说明。

1. Java/C++影响语言互操作各纬度对比



1. Java/C++语言互操作JNI机制使用介绍
2. JNI简介

JNI规范规定了如何跟C/C++/汇编进行互操作。

1. Java调用C和C++

1.java中使用native关键字标记

2.使用工具javah或者手工按照JNI命名规范命名对应函数(dynamic linking需要)

3.使用C/C++实现对应的函数即可

备注：JNI中有两种将java和native函数进行绑定方式：通过JNI\_OnLoad->env->RegisterNatives绑定或者通过Dynamic linkers resolve entries based on their names。

1. C/C++调用Java

C/C++调用Java，基本通过env里面提供的一组函数进行完成，学会了这些函数就可以访问Java对象里面的field，method进行各种调用了。

1. Java/C++语言互操作实现原理
2. Java是如何知道对应C/C++实现的函数在那个so里面？

在写JNI相关代码时候，程序员必须写一个类似的代码：

System.loadLibrary("hello");

System.loadLibrary (libcore/ojluni/src/main/java/java/lang/System.java)

->Runtime.getRuntime().loadLibrary0(caller, name) (java/lang/Runtime.java)

->Runtime.nativeLoad()

-------JNI------------------

->Runtime\_nativeLoad (ojluni/src/main/native/Runtime.c, libopenjdk.so)

->JVM\_NativeLoad (art/openjdkjvm/OpenjdkJvm.cc, libopenjdkjvm.so）

->vm->LoadNativeLibrary (JavaVMExt::LoadNativeLibrary, jni/java\_vm\_ext.cc)

在http://java\_vm\_ext.cc中Libraries和SharedLibrary数据结构，记录所有被加载的so，并且在LoadNativeLibrary中回调"JNI\_OnLoad"函数进行初始化，有兴趣的可以跟踪一下代码。

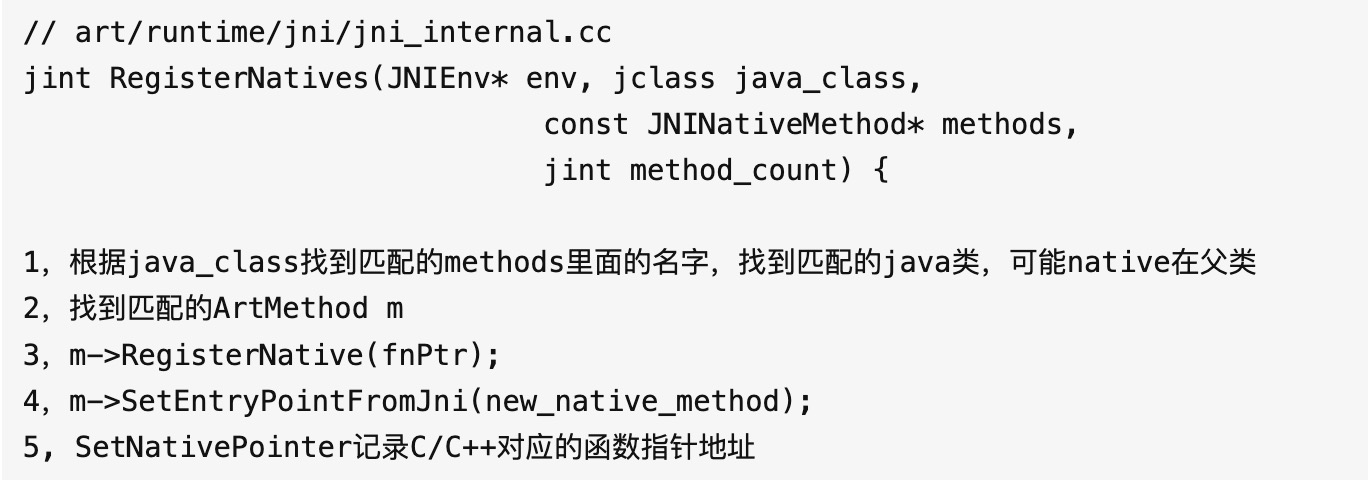
这个里面有个奇怪问题，就是为啥System.loadLibrary可以找到对应的native函数，在虚拟机中针对llibopenjdk.so，在虚拟机创建的时候就会直接去把java基础库libcore的native函数进行动态绑定，给Java一个基础的运行环境，这样就不会出现互相依赖的问题。

Runtime::InitNativeMethods（art/runtime/runtime.cc）

1. Java的native函数跟对应C++函数是如何绑定的

所谓绑定，即调用Java native函数的时候可以找到匹配的C/C++函数。Java中提供了两种不同的绑定方法：

JNI\_OnLoad/env->RegisterNatives动态注册



Dynamic linkers resolve动态衔接

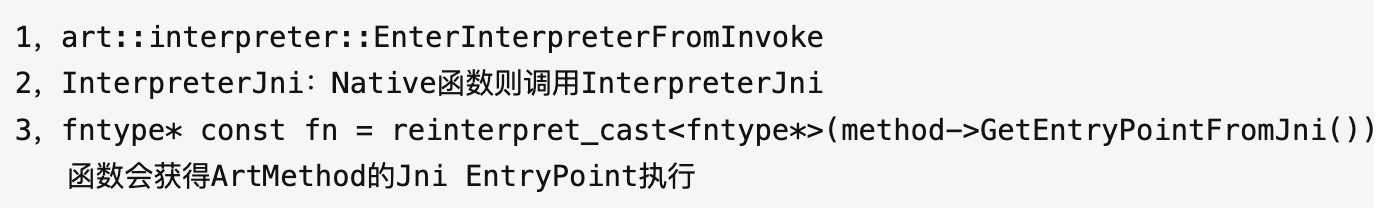
动态衔接是在函数真正调用的时候，按需lazy的方式的查找，在调用中会触发查找流程



1. Java方法如何调用到C/C++函数

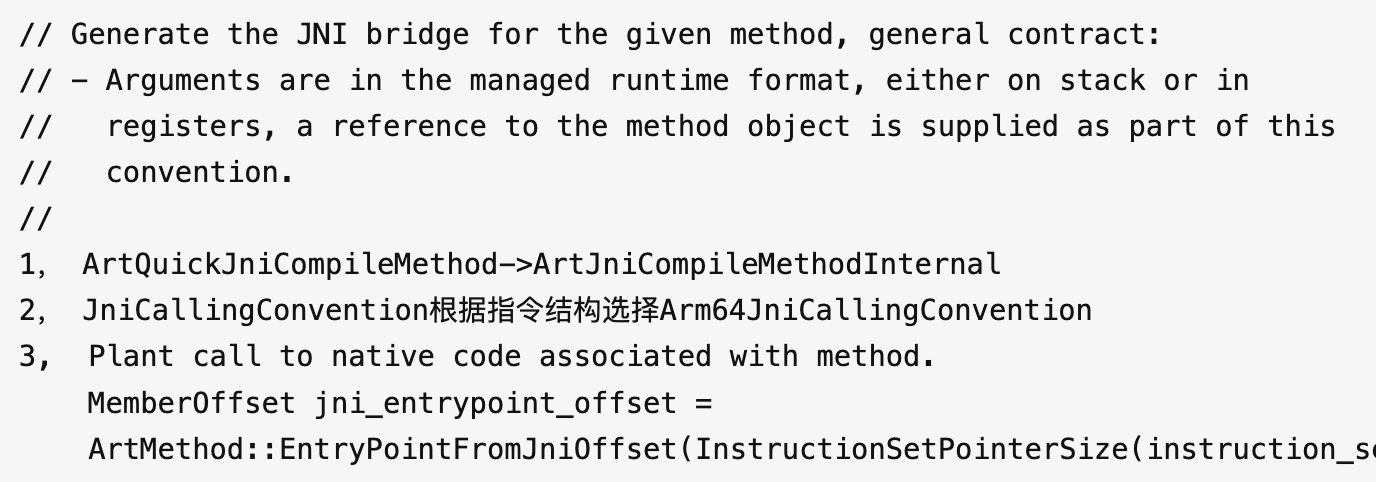
java调用绑定到native函数的C/C++函数，java执行分两种方式，一种就是解释器执行，另外就是JIT/AOT产生的机器指令代码。

解释器执行调用native函数流程



JIT/AOT编译后代码调用native函数流程

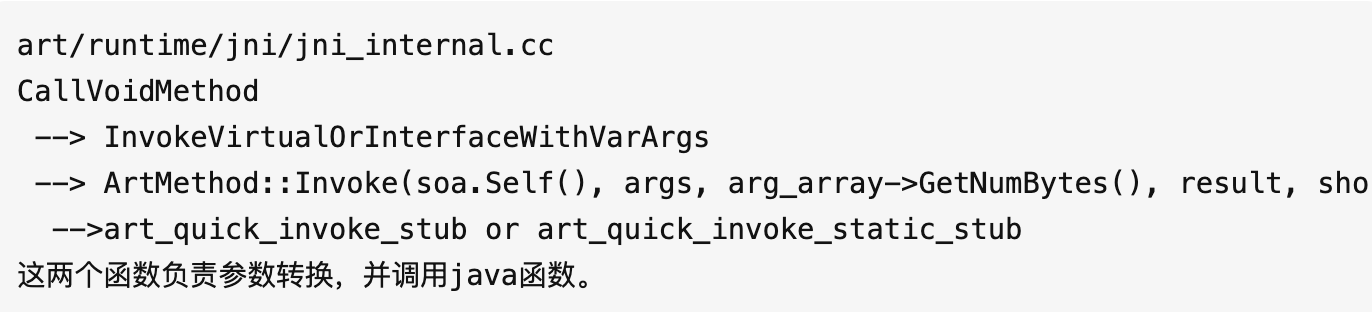
JIT调用optimizingCompiler::JitCompile编译一个Native Java方法，会产生对应的stub代码



编译器调用ArtJniCompileMethodInternal为每个native java函数产生一段glue代码，按照C/C++调用规范，安装jni\_entrypoint\_offset 对应的ArtMethod记录的native地址。Java代码调用native函数的时候，基本就是先调用编译产生的jni bridge代码，为调用c/c++函数准备好参数，处理好返回值。这块实现细节比较多，会根据fast，critical不同的标记，产生相应的优化代码，还需要处理好gc，local reference的问题。

1. C/C++函数调用Java对象中方法

这个流程相对来说，不太复杂，以CallVoidMethod实现为例，简单看一下ART虚拟机的实现流程：



1. C++/Java对象内存回收机制如何正确保障的

在JNI中C++对象和Java对象完全分属到两个不同世界。

C++申请和访问所有java对象，必须通过Env里面相关函数进行，不能直接访问对应Java对象。C++通过env里面的函数返回是java local reference：soa.AddLocalReference

Reference：可以理解是一个句柄，ART里面是一个IndirectReferfenceTable维护，本身IndirectReferfenceTable也是GC一个Root节点，因此Local表里面的对象是无法回收的，这样可以保证C++持有的对象句柄，不会被释放，除非函数执行完或者显示的调用delete

ScopedObjectAccess：可以保证在调用jni函数的之后，访问管理对象时候调用，一是将jobject加入LocalReference二是线程切换为kRunnable状态，即离开安全区。

Java对象也可以通过long类型地址，持有c++对象的指针，通过提供显示的销毁函数或者实现finalize() 进行生命周期管理，在Android中使用非常多。

1. 异常和unwind

在JNI中C++异常和Java异常完全分属到两个不同世界。C++异常不影响Java异常，Java异常也不会影响Java，他们的实现方式完全不同，一个依赖elf里面的异常table表，另外依赖java虚拟机实现。