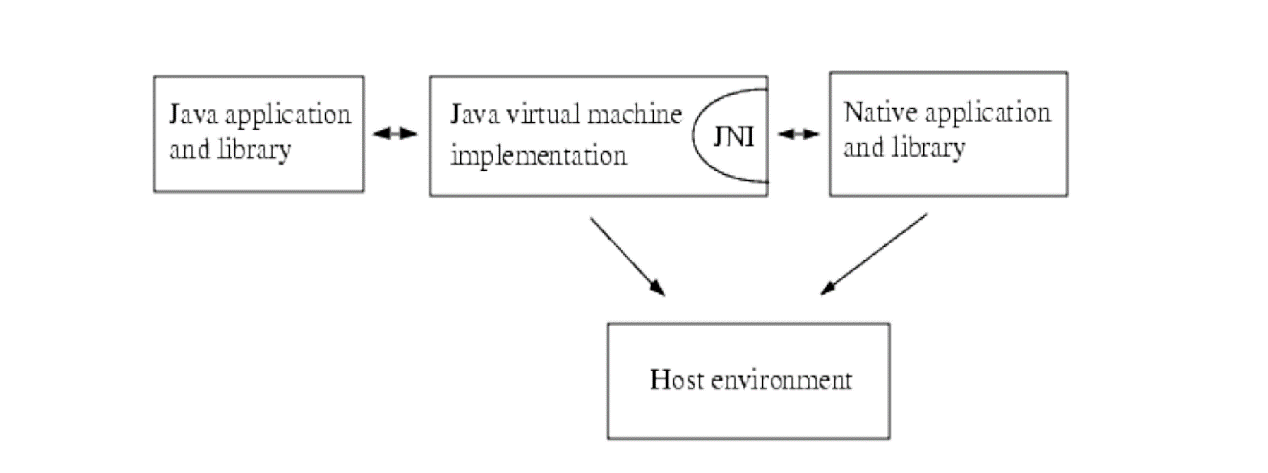
什么是JNI更多详细的说明？

Java native Interface

Java 与C/C++ 相互调用的一套API

JNI 是 JAVA 平台的一个重要特征，使用它我们可以重用以前用 C/C++写的大量代码。

Java 平台和本地环境的简介



Host environment

系统环境代指本地操作系统环境，它有自己的本地库和 CPU 指令集。

本地程序和本地库都使用CC++ 这样的本地语言编写，然后编译成本地环境下可以运行的二进制代码。 所以本地程序和本地库一般都会依赖这个本地的系统环境。

JNI 的角色

JNI的强大就在于它使用java平台的同时，还可以重用原来的本地代码。作为虚拟机实现的一部分，JNI 允许java 和本地代码间的双向交互。

什么场合下使用JNI

1. 程序当中用到了 JAVA API 不提供的特殊系统环境才会有的特征。而跨进程操作又不现实。
2. 你可能想访问一些己有的本地库，而他们并不是用java语言编写
3. JAVA 程序当中的一部分代码对效率要求非常高，如算法计算，图形渲染等

JNI 和NDK关系

 JNI是**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "_blank)**语言提供的Java和C/C++相互沟通的机制，Java可以通过JNI调用本地的C/C++代码，本地的C/C++的代码也可以调用java代码。JNI 是本地编程接口，Java和C/C++互相通过的接口。Java通过C/C++使用本地的代码的一个关键性原因在于C/C++代码的高效性。

NDK是一系列工具的集合。它提供了一系列的工具，帮助开发者快速开发C（或C++）的动态库，并能自动将so和java应用一起打包成apk。这些工具对开发者的帮助是巨大的。它集成了交叉编译器，并提供了相应的mk文件隔离CPU、平台、ABI等差异，开发人员只需要简单修改mk文件（指出“哪些文件需要编译”、“编译特性要求”等），就可以创建出so。它可以自动地将so和Java应用一起打包，极大地减轻了开发人员的打包工作。

# 开始使用JNI

步骤：

1. 编写native 方法
2. Javah命令，生成.h 头文件

Javah 生成的函数名是： Java\_完整类名\_函数名

正常情况下我们都要使用这个名字，如果名字不对就访问不了。

然而，动态注册可以，不过这里就不讲解了，思维集中一下。

1. 复制.h头文件到CPP工程中
2. 复制jni.h和jni\_md.h 文件到工程中(jdk 中可以找到)
3. 实现.h 头文件中声明的函数
4. 生成一个dll 动态文件
5. 加载dll 文件到动态库中

举例说明：

Static 函数

## 静态库和动态库

这些库是指函数库。就是我们把一些公用的函数制作而生成的函数库。

静态库一般是 .a 文件

静态库在程序编译时会被连接(包含.h)到目标代码中，程序运行时将不再需要该静态库

动态库.dll /.so 在编译时并不会连接到目标代码中，而是程序运行时才会载入，因此在程序运行时还需要动态存在

## JNIEnv 说明：

结构体指针，代表java运行环境，可以通过JNIEnv 来调用JAVA中的一些代码。调用java中的代码

env 是二级指针。

为什么用二级指针？

C语音中：

JNIEnv 在C语言中是一个结构体指针的别名，所以是二级指针

C++ 中：

JNIEnv 就是一个结构体的名字，这个结构体里面有一个接口，用于将C++的调用转换成C语音的调用。当我们想要在CPP 里面使用的时候，我们就可以不需要这个指针了。

所以，如果C 中使用的是一级指针，那么C++就不能使用了。这个JNI接口就没有办法通用了

C++ 中不需要在调用函数的时候传入env，原因就是C++ 中有this关键子。This的意义和java一样，代表就是这个对象本身。

### 总结：

在C++中：  
JNIEnv 是一个结构体的别名  
env 一级指针  
在C中：  
JNIEnv 结构体指针的别名  
env二级指针

再举例说明一个定义一个非静态的函数

JNI数据类型：

定义一个非static 的函数。作用和上面举例的一样

每个native 函数，都至少有两个参数( JNIEnv\*, jclass 或者 jobject).

当native方法为静态方法时： jclass 代表为native方法所属类的class对象(JNITest.class)

当native 方法为非静态方法时： jobject代表的native方法所属的对象

# JNI 基本数据

Java 基本数据类型与JNI数据类型的映射关系

Java 类型->JNI 类型—》C类型

Boolean jboolean;

byte jbyte;

char jchar;

short jshort;

int jint;

long jlong;

float jfloat;

doublc jdouble;

void void

C 语言对应的类型在jni.h 中看就好了

引用类型（对象）

String 类型对应关系

String jstring

其他引用类型

Object jobject

基本数据类型的数组

Byte[] jByteArray

引用类型 的数组

Object[] (String[]) jobjcetArray

### 举例说明得到一个字符串，然后修改java中字符串的值

GetStringUTFChars (JNIEnv \*, jstirng, jboolean \*) 关于第三个参数的说明：

当从 JNI 函数 GetStringChars 中返回得到字符串 B 时，如果 B 是原始字符串java.lang.String 的拷贝，则 isCopy 被赋值为 JNI\_TRUE。如果 B 和原始字符串指向的是 JVM 中的同一份数据，则 isCopy 被赋值为 JNI\_FALSE。当 isCopy 值为JNI\_FALSE 时，本地代码决不能修改字符串的内容，否则 JVM 中的原始字符串也会被修改，这会打破 JAVA 语言中字符串不可变的规则。  
通常，因为你不必关心 JVM 是否会返回原始字符串的拷贝，你只需要为 isCopy  
传递 NULL 作为参数。

JVM 是否会通过拷贝原始 Unicode 字符串来生成 UTF-8 字符串是不可以预测的，程序员最好假设它会进行拷贝，而这个操作是花费时间和内存的。一个典型  
JVM 会在 heap 上为对象分配内存。