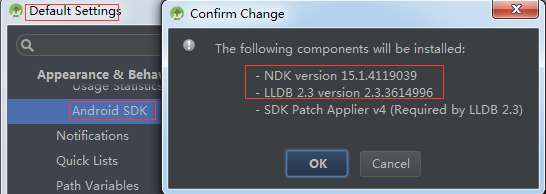
# JNI和NDK编程

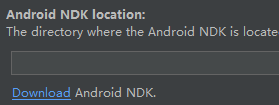
Java JNI本意为Java Native Interface(java本地接口), 是为方便java调用C或者C++等本地代码所封装的一层接口. 由于Java的跨平台性导致本地交互能力的不好, 一些和操作系统相关的特性Java无法完成, 于是Java提供了JNI专门用于和本地代码交互.

NDK是android所提供的一个工具合集, 通过NDK可以在Android中更加方便地通过JNI来访问本地代码. NDK还提供了交叉编译工具, 开发人员只需要简单的修改mk文件就可以生成特定的CPU平台的动态库. 好处如下:

* + 提供代码的安全性. so库反编译比较困难
  + 可以很方便地使用目前已有的C/C++开源库
  + 提高程序在某些特定情形下得执行效率, 但是不能明显提高Android程序的的性能.

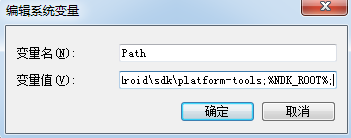
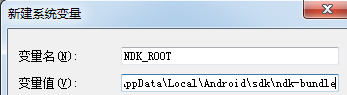
安装非常简单，可以用as的窗口下载，有两个入口





如果下载不了，访问不了官网，[否则自己网上去手动下载](https://developer.android.google.cn/ndk/downloads/index.html)

安装好的NDk一般位于你的sdk文件夹下的ndk-bundle。可以看到里面有ndk-build文件，然后将该路径配置到你系统变量的path里面去，将NDK\_ROOT追加到Path环境变量下;%NDK\_ROOT%



## JNI的开发流程

大体步骤就是在java中声明native方法, 然后调用c/c++实现的native的方法. 下面开始具体步骤

### Java中声明native方法

创建一个java类

|  |
| --- |
| public class HelloJni {  static {  System.*loadLibrary*("HelloJni");  }  public native String get();  public native void set(String str);  // 主函数  public static void main(String arg[]) {  HelloJni jniTest = new HelloJni();  System.*out*.println(jniTest.get());  jniTest.set("java写入数据");  } } |

首先是通过静态代码块先加载so库, 然后声明两个native方法提供使用. 这两个方法需要在JNI中实现. 加载so库名称填入的虽然是HelloJni, 但是so库全名称应该是libHelloJni.so,这是加载so库的规范.

### Java->class->JNI头文件

在Studio中进行终端命令行工具.执行如下命令，将在操作的路径下生成c语言头文件k\_art\_ch14Ndk\_HelloJni.h。并把头文件移动到app/src/main/jni下。不要写中文的硬编码



|  |
| --- |
| /\* DO NOT EDIT THIS FILE - it is machine generated \*/ #include <jni.h> /\* Header for class k\_art\_ch14Ndk\_HelloJni \*/  #ifndef \_Included\_k\_art\_ch14Ndk\_HelloJni #define \_Included\_k\_art\_ch14Ndk\_HelloJni /\*这个宏定义是必须的, 作用是指定extern”C”内部的函数采用C语言的命名风格来编译.  如果设定那么当JNI采用C++来实现时, 由于C/C++编译过程对函数的命名风格不同,  这将导致JNI在链接时无法根据函数名找到具体的函数, 那么JNI调用肯定会失效.\*/ #ifdef \_\_cplusplus **extern** "C" { #endif /\*  \* Class: k\_art\_ch14Ndk\_HelloJni  \* Method: get  \* Signature: ()Ljava/lang/String;  \*/  //函数名:格式遵循:Java\_包名\_类名\_方法名包名之间的.分割全部替换成\_分割.  /\* 参数: jstring是代表String类型参数. 具体的类型关系后面会说明.  JNIEnv \*: 表示一个指向JNI环境的指针, 可以通过它来访问JNI提供的方法.  jobject: 表示java对象中的this.  JNIEXPORT和JNICALL: 这是JNI种所定义的宏, 可以在jni.h这个头文件查到\*/ JNIEXPORT jstring JNICALL Java\_k\_art\_ch14Ndk\_HelloJni\_get  (JNIEnv \*, jobject);  /\*  \* Class: k\_art\_ch14Ndk\_HelloJni  \* Method: set  \* Signature: (Ljava/lang/String;)V  \*/ JNIEXPORT **void** JNICALL Java\_k\_art\_ch14Ndk\_HelloJni\_set  (JNIEnv \*, jobject, jstring);  #ifdef \_\_cplusplus } #endif #endif |

### 实现JNI.cpp

JNI方法是指的Java中声明的native方法, 这里可以选择c++和c来实现. 过程都是类似的. 只有少量的区别, 这里两种都实现一下

在工程的主目录创建一个子目录, 名称任意, 然后将之前通过javah命令生成的.h头文件复制到创建的目录下, 接着创建k\_art\_ch14Ndk\_HelloJni.cpp实现如下:

|  |
| --- |
| #include "k\_art\_ch14Ndk\_HelloJni.h" JNIEXPORT jstring JNICALL Java\_k\_art\_ch14Ndk\_HelloJni\_get(JNIEnv \*env, jobject jObj){  // return env->NewStringUTF("I`m Str from jni libs!");  //return (\*env)->NewStringUTF(env, "Hello world from jni)");//C语言格式，文件名应为xxx.c  **return** env->NewStringUTF((**char** \*)"Hello JIN");//C++格式，文件名应为xxx.cpp  } |

### 编译配置，编译C/C++文件为.so库。

在jni目录下新建Android.mk(必须是这个名称Android.mk)文件，如下图所示：

Android.mk指定.so文件名称：

|  |
| --- |
| LOCAL\_PATH := $(call my-dir)  include $(CLEAR\_VARS)  LOCAL\_MODULE := HelloJni //要生成的so库名，但实际为libHelloJni.so  LOCAL\_SRC\_FILES := k\_art\_ch14Ndk\_HelloJni.cpp //指定编译的文件  include $(BUILD\_SHARED\_LIBRARY) |

Application.mk指定生成不同平台的.so库：

# 注释掉了，不写会生成全部支持的平台。目前支持：

# APP\_ABI := armeabi arm64-v8a armeabi-v7a mips mips64 x86 x86\_64

三种常用平台，不写则全部生成（armeabi arm64-v8a armeabi-v7a mips mips64 x86 x86\_64）。架构不同有的不兼容。不写则全部生成。

也可以在.\app\build.gradle指定：

android.defaultConfig.ndk {

moduleName "HelloJni"

ldLibs "log", "z", "m"

// abiFilters "armeabi", "armeabi-v7a", "x86"

abiFilters "armeabi-v7a"

}

## QA

F:\K\tools\android-ndk-r13b-windows-x86\_64\android-ndk-r13b>ndk-build.cmd F:\K\dev\dpad\3288\dpad-explorer/app/src/main/java/jni

Android NDK: Could not find application project directory !

Android NDK: Please define the NDK\_PROJECT\_PATH variable to point to it.

F:\K\tools\android-ndk-r13b-windows-x86\_64\android-ndk-r13b\build\\..\build\core\build-local.mk:151: \*\*\* Android NDK: Aborting . Sto

[Android Studio上进行NDK编程之Hello world](http://blog.csdn.net/mingyueyixi/article/details/65449723)