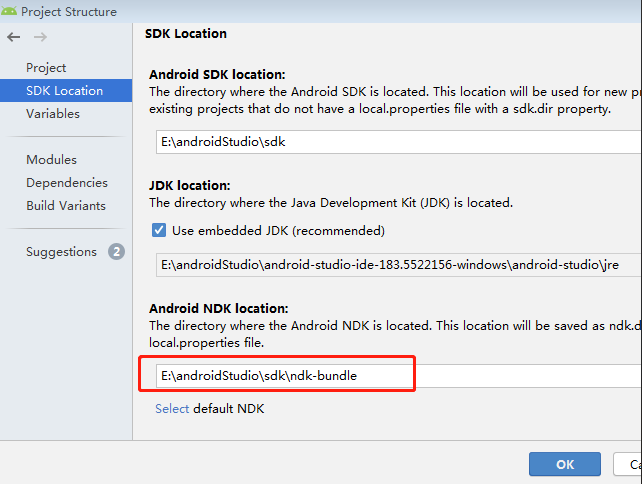
**整体流程**

配置NDK路径-----Java类中定义native方法-----生成.h头文件----编写c文件实现头文件----编写mk文件配置.so------rebuild生成.so文件

**1.配置NDK路径**

下载好NDK后，配置到Androidstudio项目中，



配置好后，会在项目的local.properties中生成

**ndk.dir**=**E\:\\androidStudio\\sdk\\ndk-bundle**

**2.Java类中定义native方法**

新建一个类，类中定义一个native方法

**package** com.shinhoandroid.test0730;

**public class** JNITest {

**public static native** String getStr();

**public static native int** add(**int** a,**int** b);

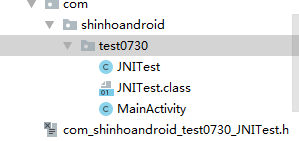
}

通过Terminal控制台，先生成class文件，在生成头文件（.h文件）

E:\test\test0730\app\src\main\java\com\shinhoandroid\test0730>javac JNITest.java

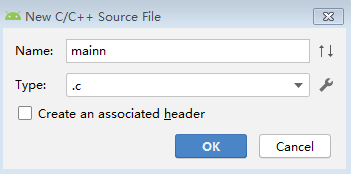
E:\test\test0730\app\src\main\java\com\shinhoandroid\test0730>cd ../../../

E:\test\test0730\app\src\main\java>javah -jni com.shinhoandroid.test0730.JNITest



**3.编写c文件实现头文件**

在main文件夹下新建一个jni文件夹把生成的.h文件拷贝进去，先编写一个c文件，实现.h文件的方法，名字可以随便取。



main.c文件实现.h文件的方法

#include **<jni.h>**

#include**<stdio.h>**

*//自己创建一个c文件，实现自己定义的native方法，也就是.h文件中的方法*

*//引入自己生成的.h头文件*

#include **"com\_shinhoandroid\_test0730\_JNITest.h"**

*//返回一个字符串*

*//JNIEXPORT jstring JNICALL Java\_com\_shinhoandroid\_test0730\_JNITest\_getStr*

*//重写com\_shinhoandroid\_test0730\_JNITest.h中的方法*

**JNIEXPORT** jstring **JNICALL** Java\_com\_shinhoandroid\_test0730\_JNITest\_getStr

(JNIEnv \*env, jclass jobj) {

**return** (\*env)->NewStringUTF(env,**"HelloWorld 我是用jni调用出来的字符串"**);

}

*//返回 a+b的结果*

**JNIEXPORT** jint **JNICALL** Java\_com\_shinhoandroid\_test0730\_JNITest\_add

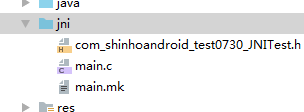
(JNIEnv \*env, jclass jobj, jint a, jint b){

**return** a+b;

}

**4.编写mk文件配置.so**

右击jni->New->File ，新建一个main.mk文件



main.mk文件

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_MODULE := app

LOCAL\_SRC\_FILES := E:\test\test0730\app\src\main\jni\main.c

LOCAL\_C\_INCLUDES += E:\test\test0730\app\src\main\jni

include $(BUILD\_SHARED\_LIBRARY)

//LOCAL\_PATH := $(call my-dir) 文件的开头必须从定义这个变量开始，这个变量用于在开发过程中定位 Android.mk文件在项目中的路径；my-dir是一个宏函数，由系统提供，调用这个函数会返回当前文件所在目录的路径

//include $(CLEAR\_VARS)这个 CLEAR\_VARS变量是系统提供的，它指向一个特殊的 GNU Makefile文件，它会清除很多 LOCAL\_XXX的变量（如：LOCAL\_MODULE、LOCAL\_SRC\_FILES、LOCAL\_STATIC\_LIBRARIES等）；为什么要清除？因为很多的控制文件都是在单个 GUN make中生成的，这些变量都是GUN make中的全局变量，如果不清除，上一次构建的数据信息会影响下一次的构建

//LOCAL\_MODULE := app 生成.so的名字，这里APP表示生成的名字为libapp.so

//LOCAL\_SRC\_FILES := E:\test\test0730\app\src\main\jni\main.c c文件的位置

//LOCAL\_C\_INCLUDES += E:\test\test0730\app\src\main\jni .h文件的位置

//include $(BUILD\_SHARED\_LIBRARY)是系统的提供的一个变量，它指向 GUN Makefile的一个脚本，这个脚本会收集你定义的所有 LOCAL\_XXX形式的变量，然后根据这些变量构建 .so动态库

= 是最基本的赋值

:= 是覆盖之前的值

?= 是如果没有被赋值过就赋予等号后面的值

+= 是添加等号后面的值

在module的build.gradle中配置

在defaultConfig中配置ndk

ndk {

abiFilters **'armeabi-v7a'**,**'x86'**

}

在Android中配置

externalNativeBuild {

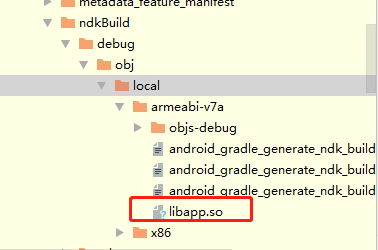
ndkBuild {

path file(**'src/main/jni/main.mk'**)

}

}

**5.rebuild生成.so文件**



没有生成，可以尝试删除build文件夹，重新rebuild。

**6.使用**

*//名字必须和build.gradle中的moduleName一致*

System.*loadLibrary*(**"app"**);

**tv** = findViewById(R.id.***tv***);

**tv**.setText(JNITest.*getStr*());