


Android Studio使用JNI


0x01 前言

本文讲述使用Android Studio通过静态注册、动态注册使用JNI的方法，以及加载第三方so文件的方法

0x02 Android Studio静态注册的方式使用JNI

1. 添加native接口






```
public class MainActivity extends Activity implements OnClickListener {

    static{
        System.loadLibrary("JniTest");
    }

    private native int Add(double num1,double num2);
    private native int Sub(double num1,double num2);
    private native int Mul(double num1,double num2);
    private native int Div(double num1,double num2);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    }
}
```





在Java类中使用System.loadLibrary("JniTest")加载我们要写的so库名称，Add/Sub/Mul/Div这四个方法在Java类中声明就可以使用了。

2.Build->Make Project

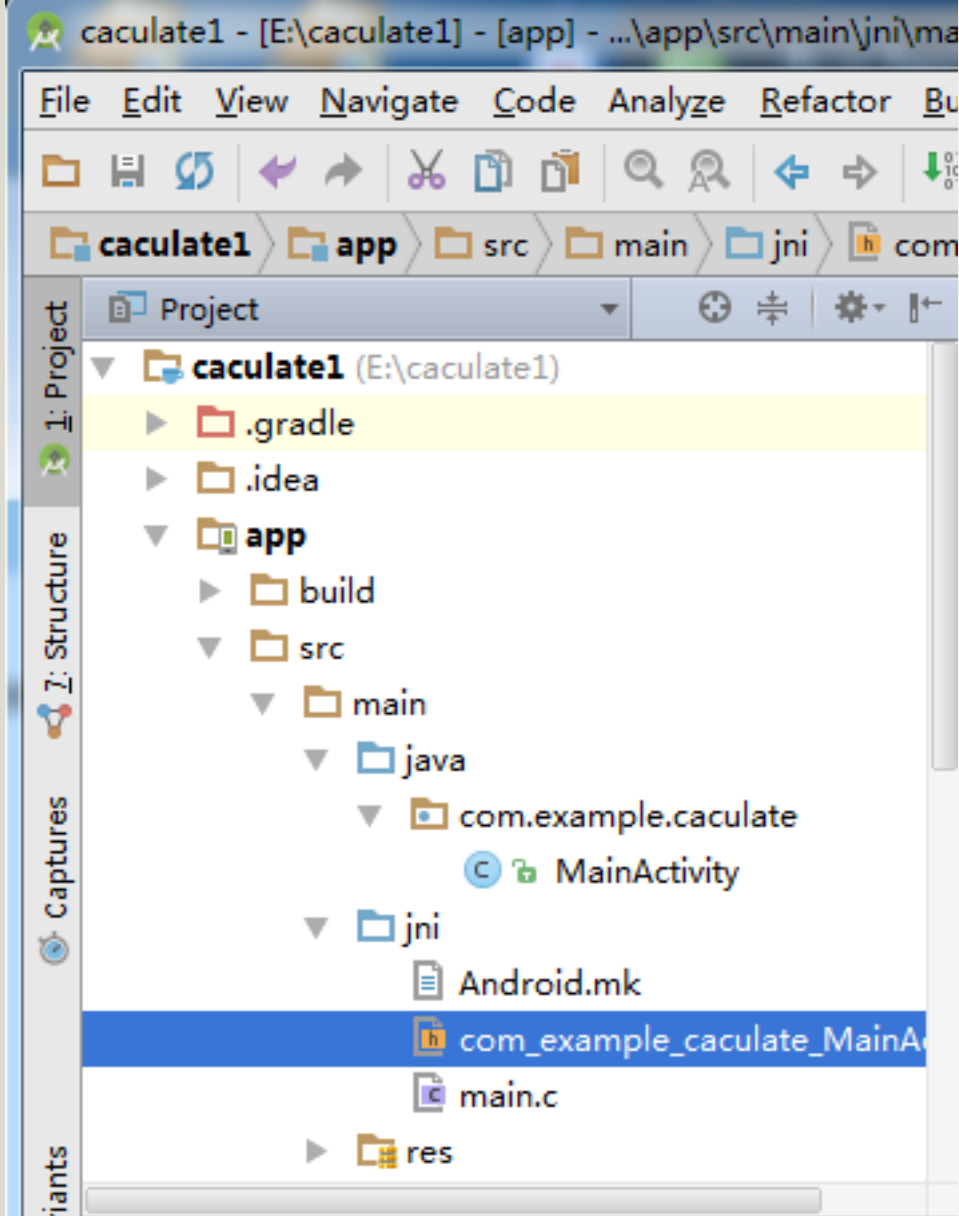
验证工程中并无其他错误，并对工程进行编译，生成.class文件
在Build/intermediates/classes/debug里面

3.javah 生成.h文件

cmd 进入工程目录在工程的app/src/main/java ， 执行 javah com.example.caculate.MainActivity 命令，就会生成so库所需要的.h文件

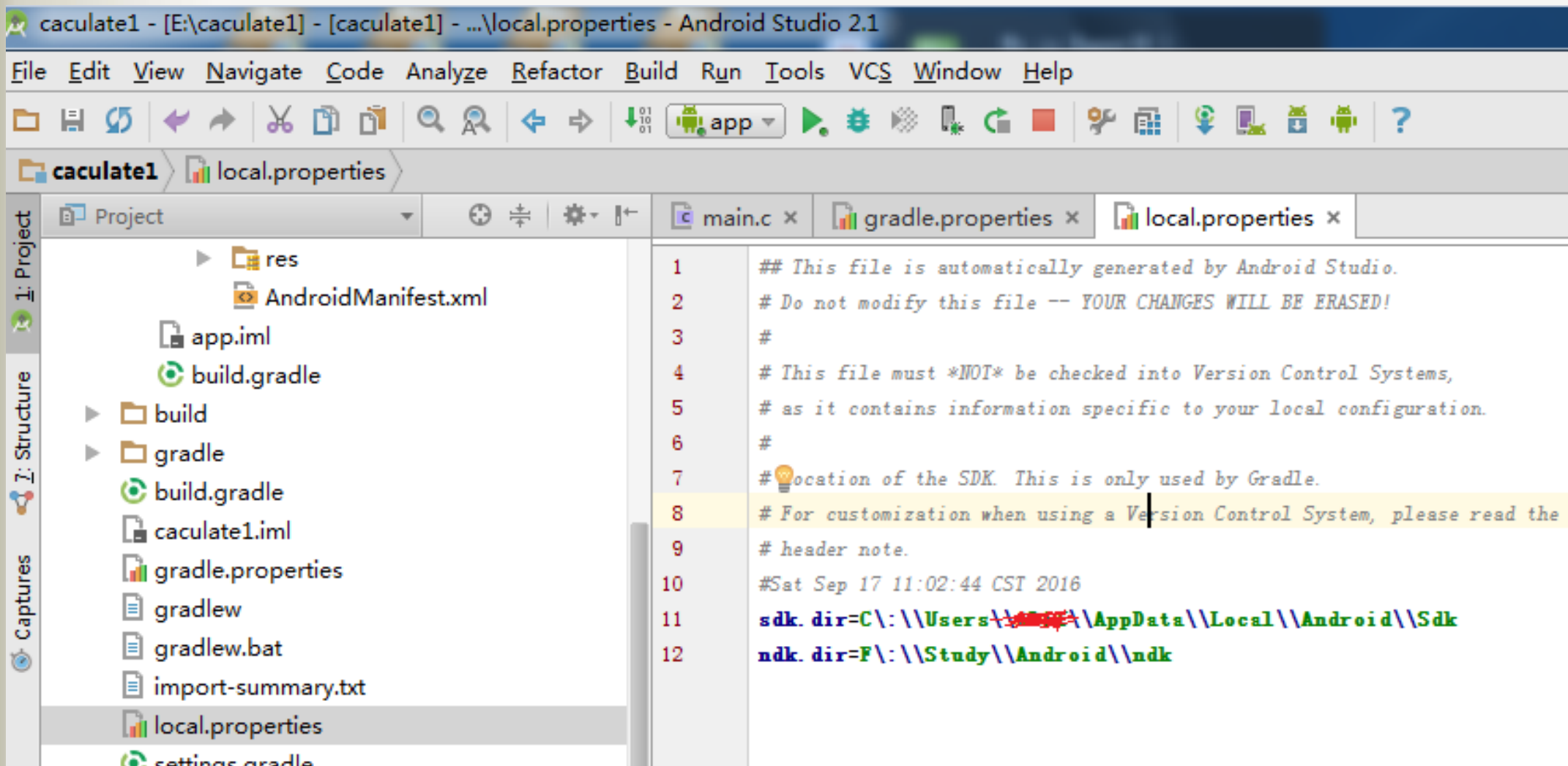
在/app/src/main/下建立jni目录，将.h拷贝进去

4.jni目录下新建一个.c文件，完成so库的函数实现(注:这里的Android.mk文件并不需要，可以不写，用法在下面会提到)



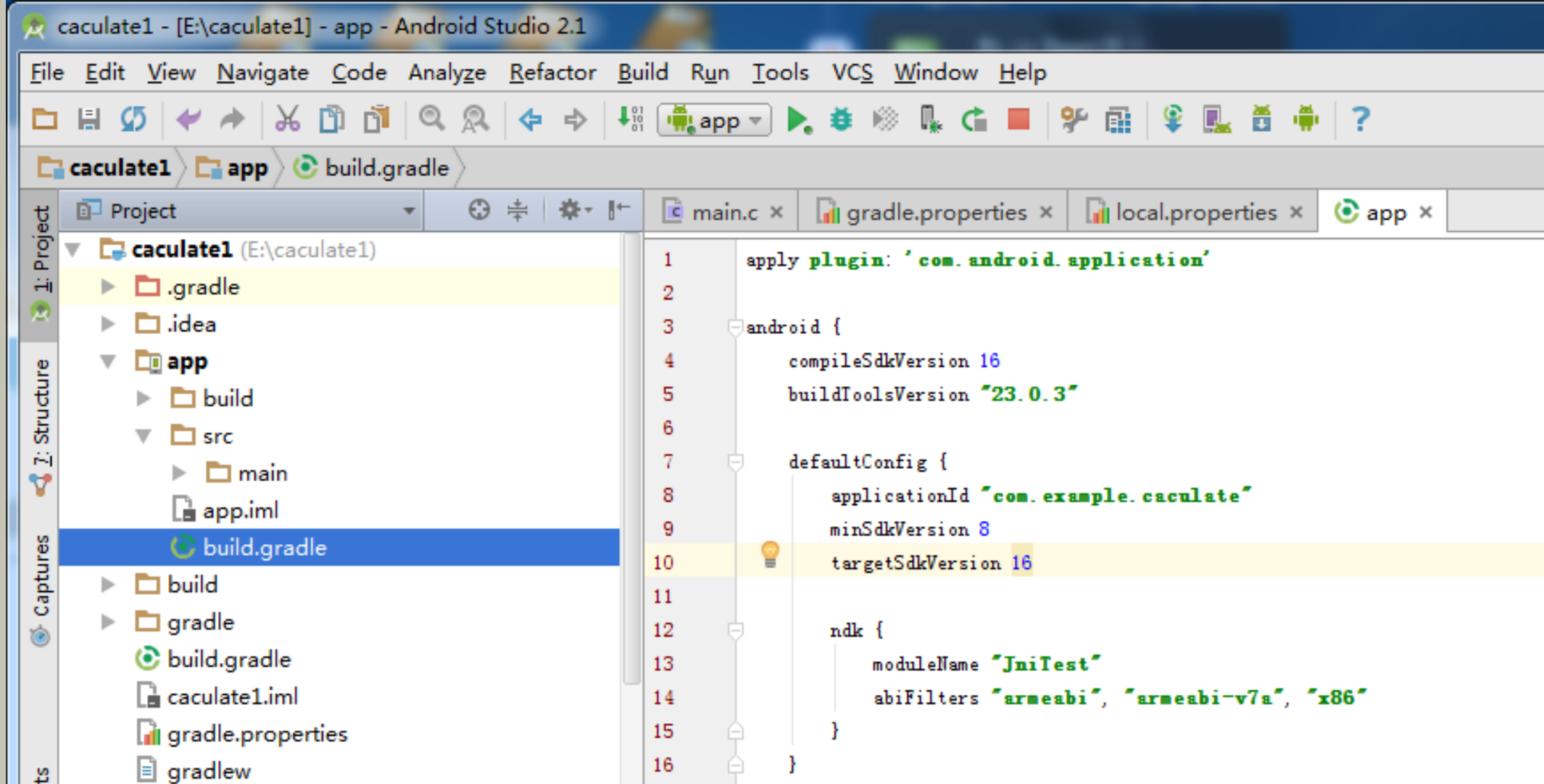
5.Build->Make Project 会报错，这里在android studio中添加ndk路径编译生成so文件

①在local.properties文件中增加ndk的路径



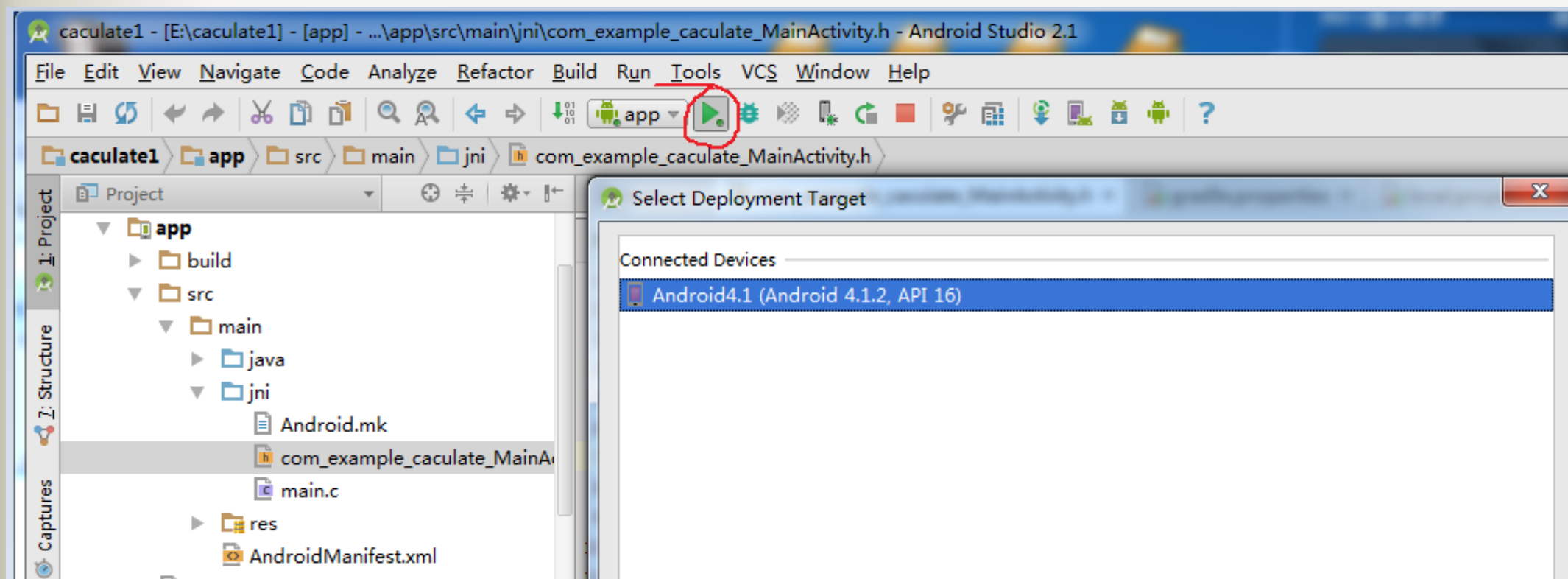
②在/app/目录下的build.gradle文件里增加 so的名称

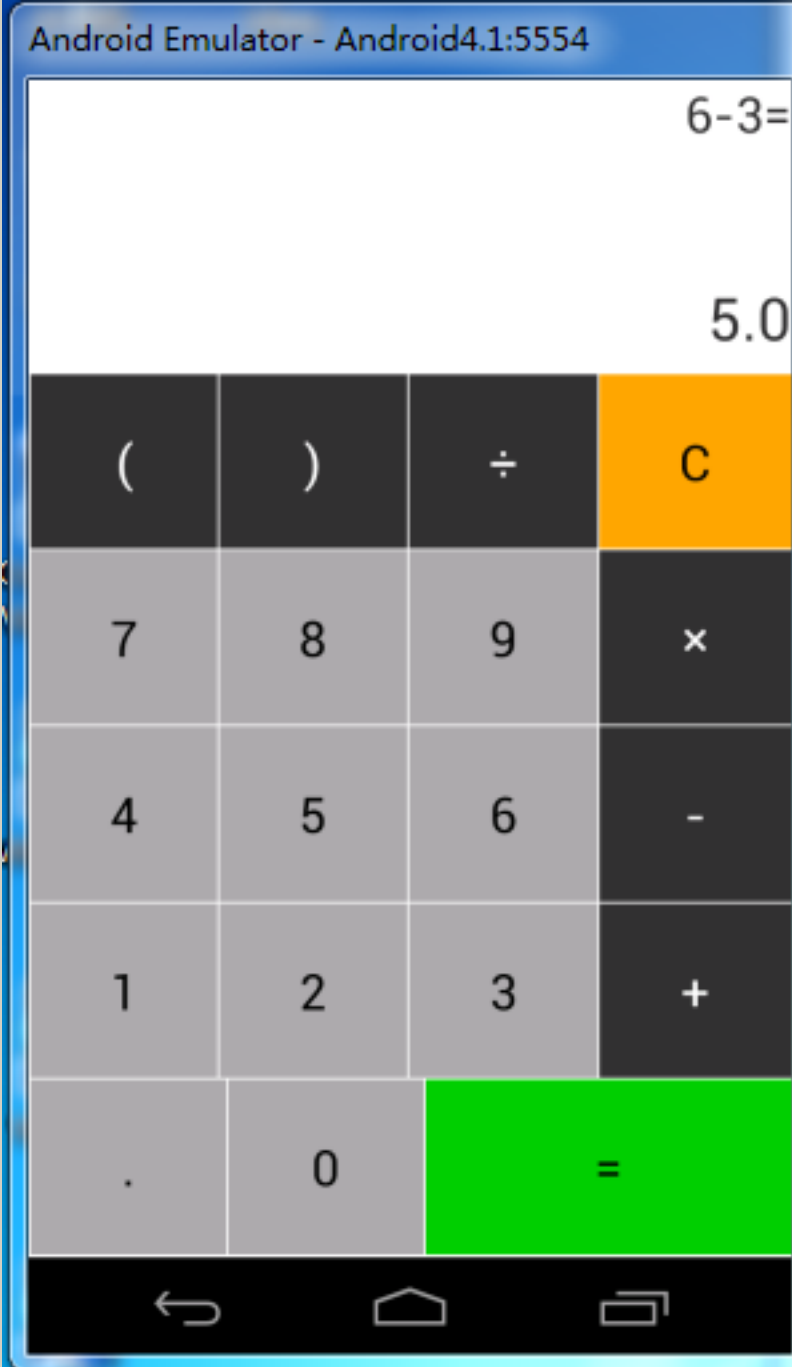
```
ndk {
    moduleName "JniTest"
    abiFilters "armeabi", "armeabi-v7a", "x86"
}
```



6.Build->Make Project 就可以编译出so文件了
生成的so在app/Build/intermediates/ndk/debug/lib下面

7.点击运行，就可以使用我们实现的so中的代码了





(注:在so库中实现在原结果基础上加2)

使用模拟器利用busybox安装grep命令之后，我们可以通过/proc/<pid>/maps文件中保存的进程加载模块，查看我们写的so文件是否被加载了。

```
1!root@android:/data/busybox # cat /proc/29466/maps|grep libJniTest.so
5a953000-5a955000 r-xp 00000000 1f:01 906 /data/data/com.example.caculate/lib/libJniTest.so
5a955000-5a956000 r--p 00001000 1f:01 906 /data/data/com.example.caculate/lib/libJniTest.so
5a956000-5a957000 rw-p 00002000 1f:01 906 /data/data/com.example.caculate/lib/libJniTest.so
root@android:/data/busybox #
```

8.我们JNI的实现文件

①com_example_caculate_MainActivity.h文件

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_example_caculate_MainActivity */

#ifndef _Included_com_example_caculate_MainActivity
#define _Included_com_example_caculate_MainActivity
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      com_example_caculate_MainActivity
 * Method:     Add
 * Signature:  (DD)I
 */
JNIEXPORT jint JNICALL Java_com_example_caculate_MainActivity_Add
    (JNIEnv *, jobject, jdouble, jdouble);

/*
 * Class:      com_example_caculate_MainActivity
 * Method:     Sub
 * Signature:  (DD)I
 */
JNIEXPORT jint JNICALL Java_com_example_caculate_MainActivity_Sub
```

```
(JNIEnv *, jobject, jdouble, jdouble);

/*
 * Class:      com_example_caculate_MainActivity
 * Method:     Mul
 * Signature:  (DD)I
 */
JNIEXPORT jint JNICALL Java_com_example_caculate_MainActivity_Mul
    (JNIEnv *, jobject, jdouble, jdouble);

/*
 * Class:      com_example_caculate_MainActivity
 * Method:     Div
 * Signature:  (DD)I
 */
JNIEXPORT jint JNICALL Java_com_example_caculate_MainActivity_Div
    (JNIEnv *, jobject, jdouble, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```



②so库中实现函数的main.c文件



```
#include <jni.h>
#define jintJNICALL
#ifndef _Included_com_example_caculate_MainActivity
#define _Included_com_example_caculate_MainActivity
#ifdef __cplusplus
extern "C" {
#endif

JNIEXPORT jintJNICALL Java_com_example_caculate_MainActivity_Add
    (JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    return (jint)(num1 + num2+2);
}

JNIEXPORT jintJNICALL Java_com_example_caculate_MainActivity_Sub
    (JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    return (jint)(num1 - num2+2);
}

JNIEXPORT jintJNICALL Java_com_example_caculate_MainActivity_Mul
    (JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    return (jint)(num1 * num2+2);
}

JNIEXPORT jintJNICALL Java_com_example_caculate_MainActivity_Div
    (JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    if (num2 == 0) return 0;
    return (jint)(num1 / num2+2);
}
#ifdef __cplusplus
}
#endif
#endif
```



0x03 Android Studio动态注册的方式使用JNI

1.jni目录下直接编写so库中的.c文件

JNI_Onlad会作为so库被加载后的第一个执行函数，最后通过RegisterNatives函数将JNI函数注册



```
#include <jni.h>
#include <stdio.h>
//#include <assert.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

JNIEXPORT jint JNICALL native_Add
(JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    return (jint)(num1 + num2 +1);
}

JNIEXPORT jint JNICALL native_Sub
(JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    return (jint)(num1 - num2 +1);
}

JNIEXPORT jint JNICALL native_Mul
(JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    return (jint)(num1 * num2 +1);
}

JNIEXPORT jint JNICALL native_Div
(JNIEnv *env, jobject obj, jdouble num1, jdouble num2)
{
    if (num2 == 0) return 0;
    return (jint)(num1 / num2 +1);
}

//Java和JNI函数的绑定表
static JNINativeMethod gMethods[] = {
    {"Add", "(DD)I", (void *)native_Add},
    {"Sub", "(DD)I", (void *)native_Sub},
    {"Mul", "(DD)I", (void *)native_Mul},
    {"Div", "(DD)I", (void *)native_Div},
};

//注册native方法到java中
static int registerNativeMethods(JNIEnv* env, const char* className,
                                JNINativeMethod* gMethods, int numMethods)
{
    jclass clazz;
    clazz = (*env)->FindClass(env, className);
    if (clazz == NULL) {
        return JNI_FALSE;
    }
    if ((*env)->RegisterNatives(env, clazz, gMethods,numMethods) < 0){
        return JNI_FALSE;
    }

    return JNI_TRUE;
}
```



```
}

int register_ndk_load(JNIEnv *env)
{

    return registerNativeMethods(env, "com/example/caculate/MainActivity",
                                gMethods,sizeof(gMethods) / sizeof(gMethods[0]));
    //NELEM(gMethods));
}


```

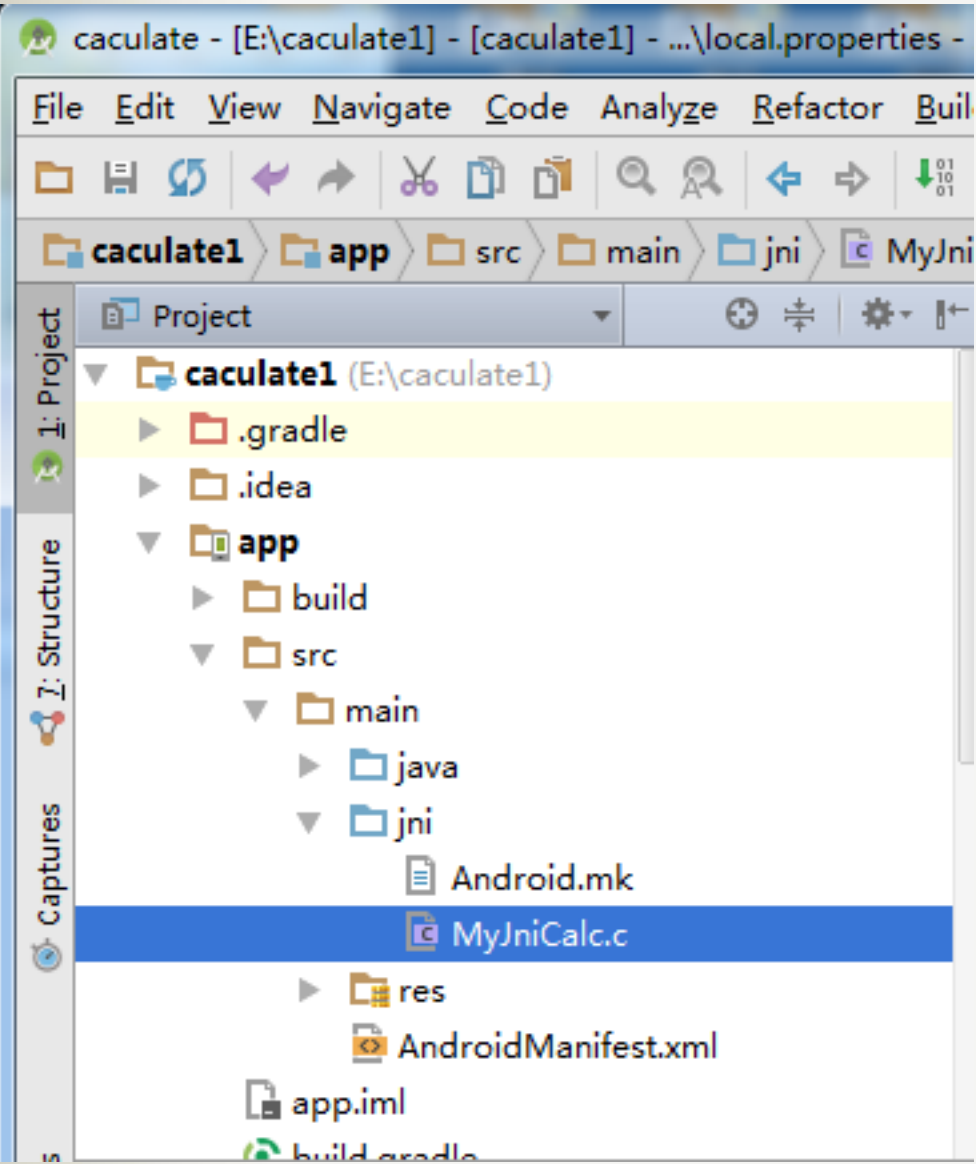
```
JNIEXPORT jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv* env = NULL;
    jint result = -1;

    if ((*vm)->GetEnv(vm, (void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        return result;
    }

    register_ndk_load(env);

    // 返回jni的版本
    return JNI_VERSION_1_4;
}


```



(注:这里Android.mk文件也不需要，在下面会提到)

2.在Java类中添加native接口，加载JniTest.so，以及声明native函数，声明之后，函数可以直接使用。



```
public class MainActivity extends Activity implements OnClickListener {

    static{
        System.loadLibrary("JniTest");
    }

    private native int Add(double num1,double num2);
    private native int Sub(double num1,double num2);
    privatenative int Mul(double num1,double num2);
    privatenative int Div(double num1,double num2);

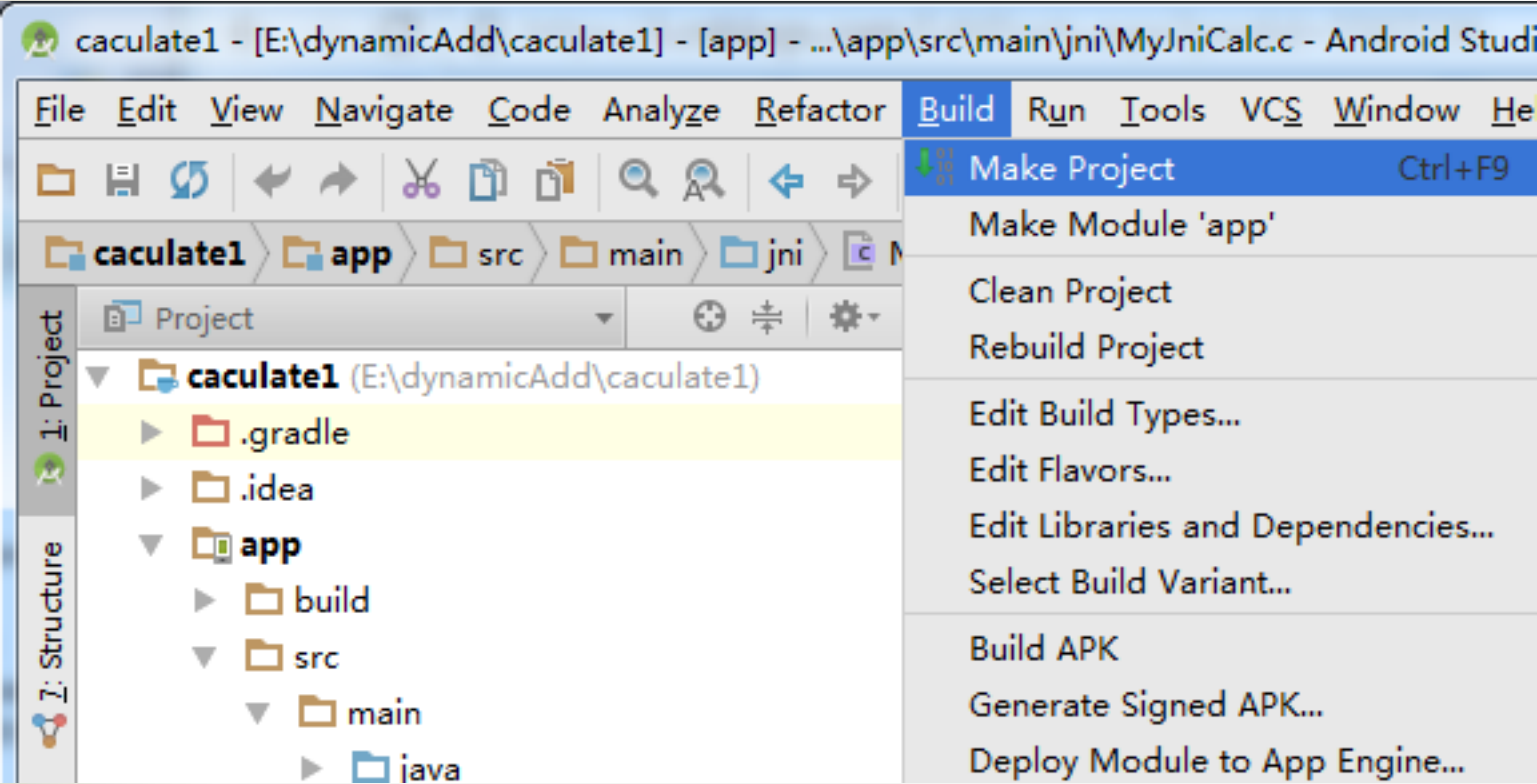
}


```

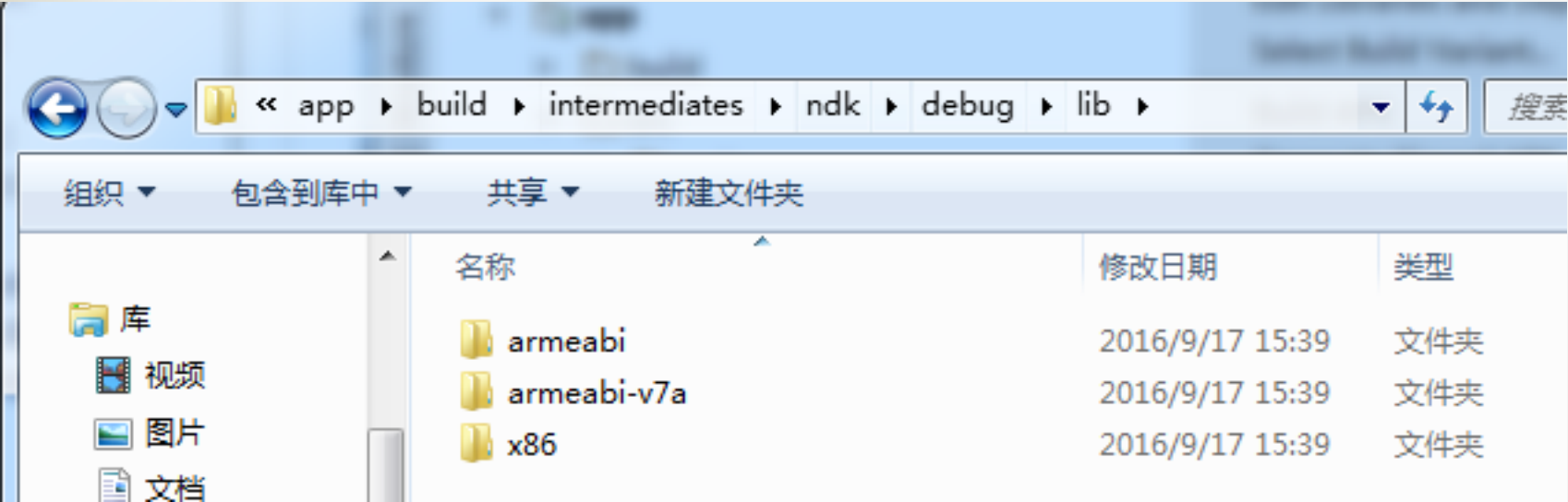
```
@Override
protected void onCreate(Bundle savedInstanceState) {
}
}
```



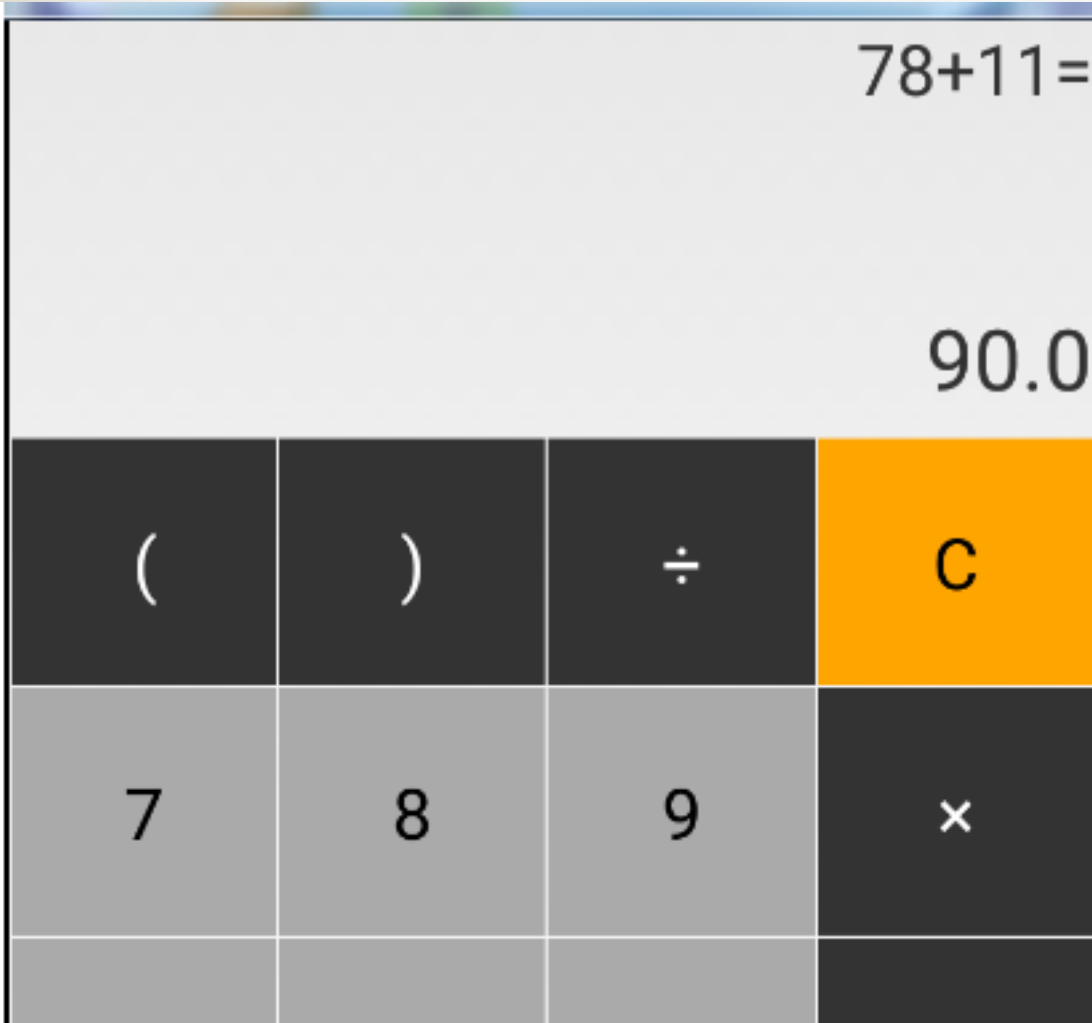
3.make Project 生成so库,如果之前没有修改/app/build.gradle文件和在local.properties中指定ndk路径则会报错。报错参考静态使用JNI中的步骤5，指定ndk路径，并且指定so名称和输出的架构。



生成成功，则在/app/build/intermediates/ndk/debug/lib目录下生成相应的so文件



4.在虚拟机中运行



(注： 这里的结果在原结果上加1)

0x04 Android Studio加载第三方库

1.使用ndk生成so文件

- ①建立jni目录(随便在哪个地方)
- ②编写so库中的.c函数， 这里完全使用0x03 Android Studio动态注册的方式使用JNI中的c文件， 使用动态注册的方式
- ③编写Android.mk文件(单独使用ndk编译so文件的时候需要用到)

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib -llog
LOCAL_PRELINK_MODULE := false
LOCAL_MODULE := JniTest
LOCAL_SRC_FILES := MyJniCalc.c
LOCAL_SHARED_LIBRARIES := libandroid_runtime
include $(BUILD_SHARED_LIBRARY)
```

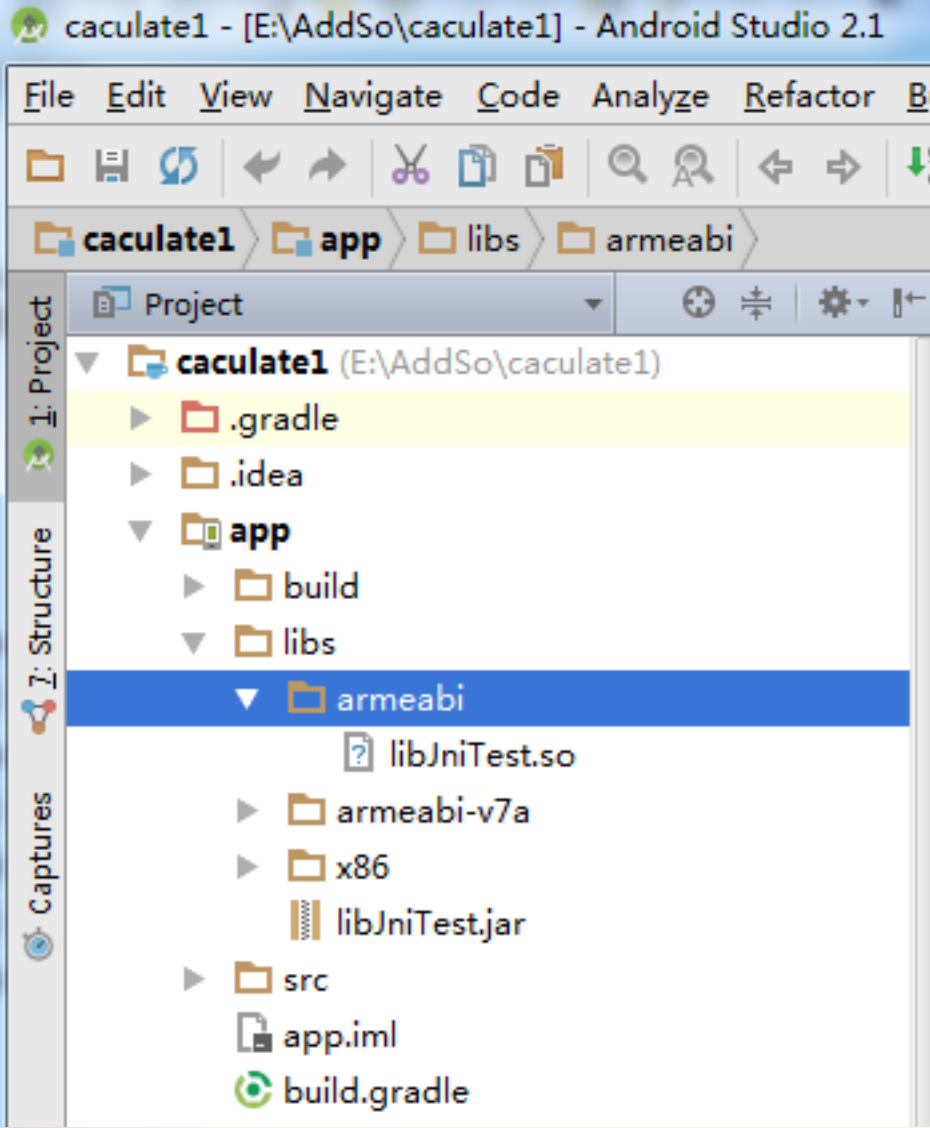
④使用cmd进入jni目录，执行ndk-build，生成so文件

```
E:\AddSo\jni>ndk-build
[armeabi] Compile thumb      : JniTest <= MyJniCalc.c
[armeabi] SharedLibrary      : libJniTest.so
[armeabi] Install             : libJniTest.so => libs/armeabi/libJniTest.so
```

2.加载生成的so文件，打包进apk中

①我们在/app目录下建立libs目录，将我们的so文件拷贝进去

这里可以使用ndk自己生成的so文件，也可以使用其他第三方so文件



②改写/app/build.gradle文件， 编译的时候将so生成上图中的libJniTest.jar文件

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 16
    buildToolsVersion "23.0.3"
```

```

defaultConfig {
    applicationId "com.example.caculate"
    minSdkVersion 8
    targetSdkVersion 16
}

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.txt'
    }
}

sourceSets.main {
    jni.srcDirs = []
    jniLibs.srcDir 'src/main/libs'
}

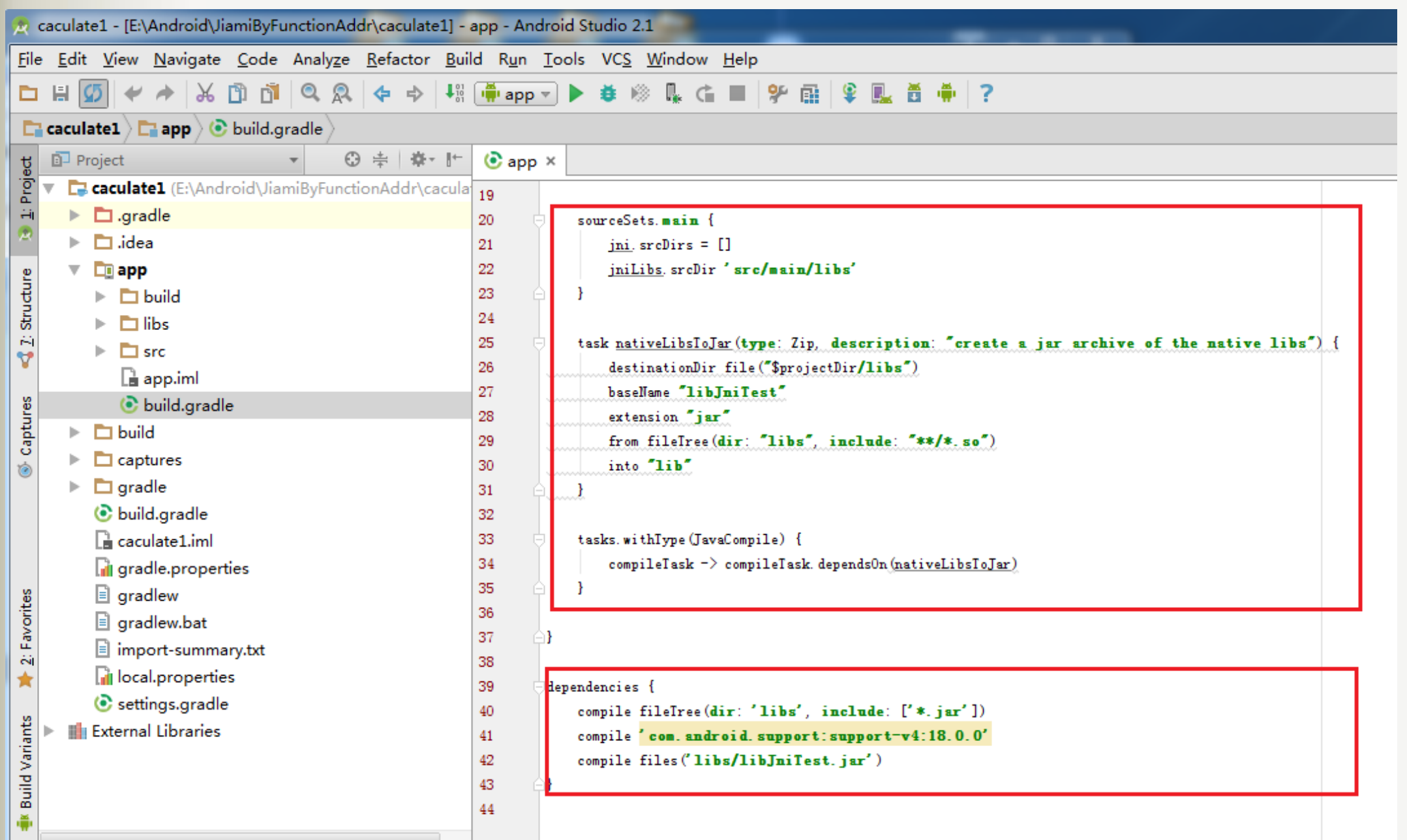
task nativeLibsToJar(type: Zip, description: "create a jar archive of the native libs") {
    destinationDir file("$projectDir/libs")
    baseName "libJniTest"
    extension "jar"
    from fileTree(dir: "libs", include: "**/*.so")
    into "lib"
}

tasks.withType(JavaCompile) {
    compileTask -> compileTask.dependsOn(nativeLibsToJar)
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:18.0.0'
    compile files('libs/libJniTest.jar')
}

```

其中into "lib" 是生成的jar文件存放的目录, include: "**/*.so" 为所依赖的so文件



③在Java类中添加so的接口

```
public class MainActivity extends Activity implements OnClickListener {

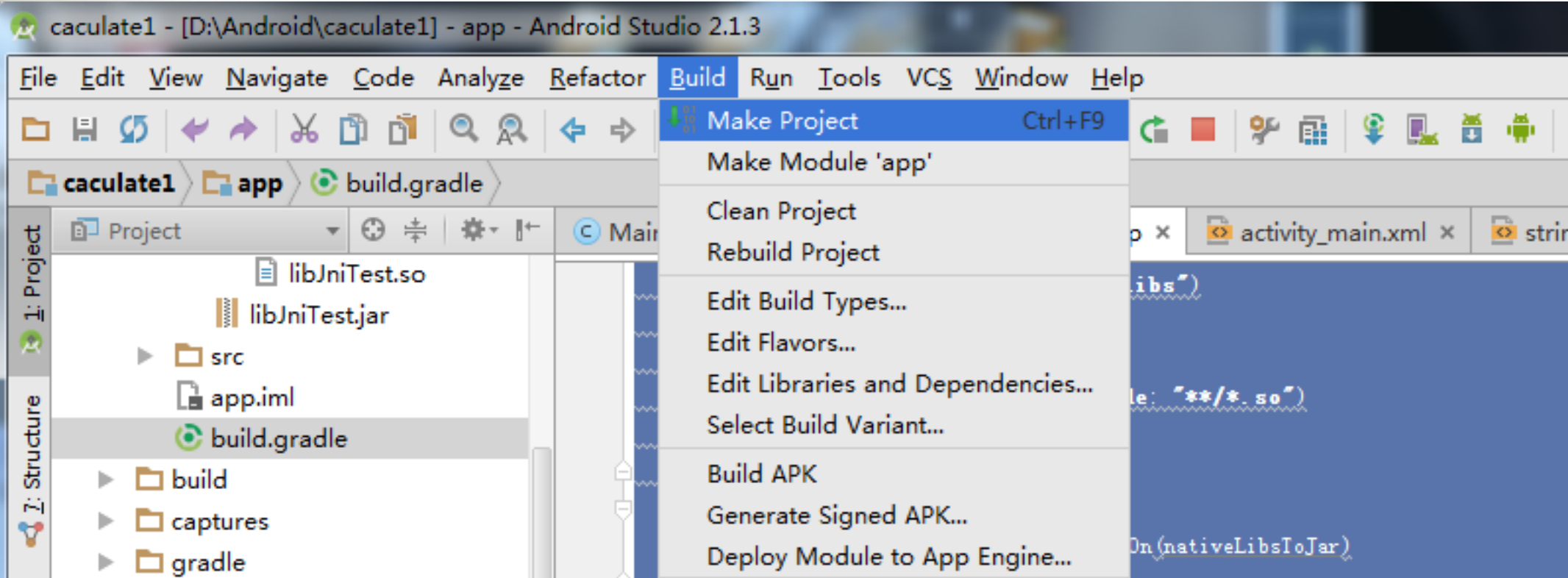
    static{
        System.loadLibrary("JniTest");
    }

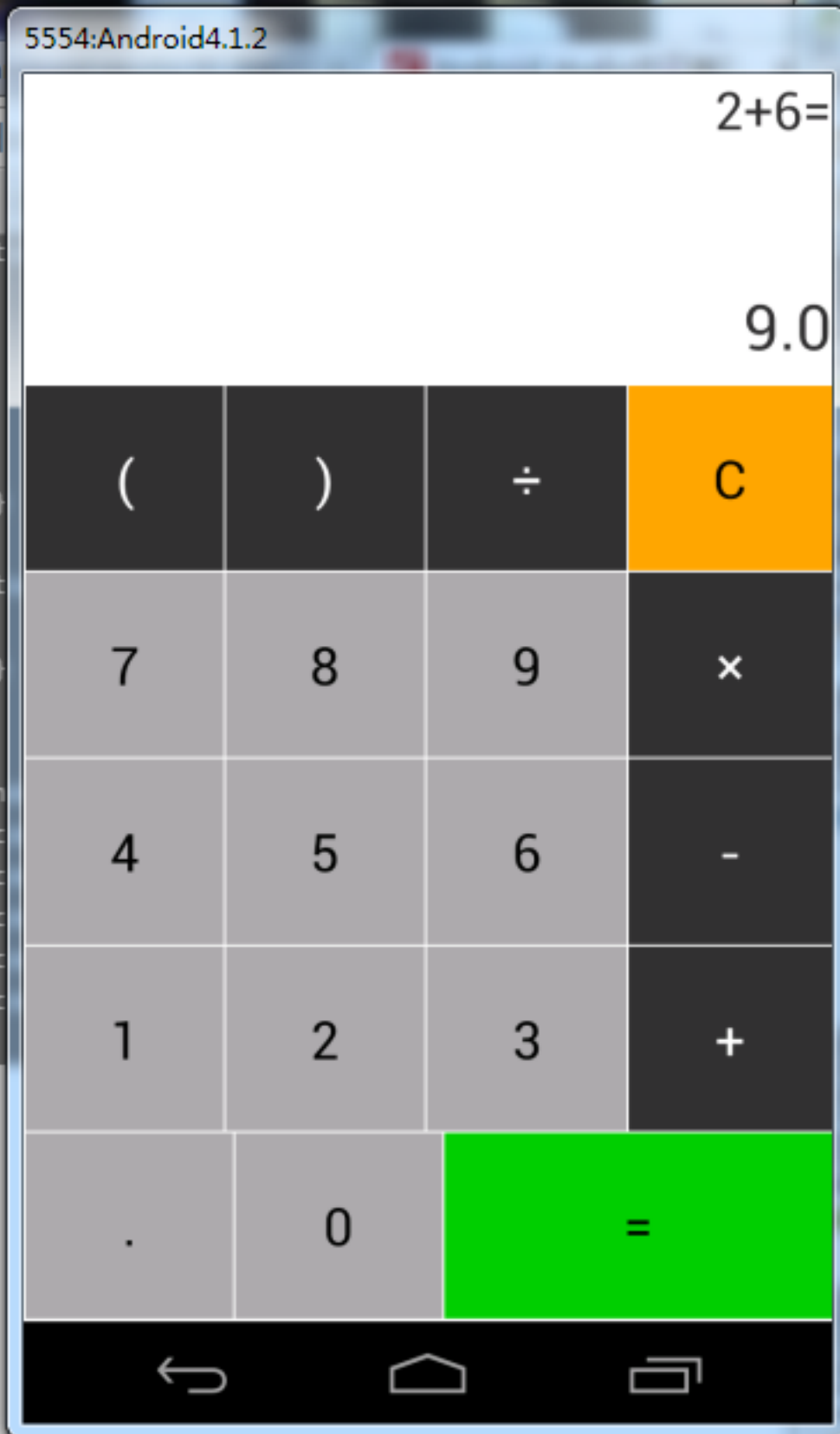
    private native int Add(double num1,double num2);
    private native int Sub(double num1,double num2);
    private native int Mul(double num1,double num2);
    private native int Div(double num1,double num2);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
    }
}
```

4.Make Project，点击运行

这里不需要指定ndk路径，so库的名称也不需要再/app/build.gradle中指定，在Android.mk中指定





(注:这里结果在原结果上加1)

0x05 总结

这里是Android Studio中的使用方法，刚开始也是各种百度，弄得很复杂，在后面的学习中也懂得了很多，所以今天将使用方法重新整理了一遍，希望大家可以不要浪费太多时间。**静态注册方法**主要要用javah生成.h文件，显得比较复杂，每次添加函数都要重新生成.h文件，不过如果知道函数名称的格式也可以不生成.h文件，实践中，在jni目录下只有main.c文件也是可以运行成功的。**动态注册方法**直接在函数中注册比较容易动态扩展，但是需要对注册的数据类型有所了解，可以参考静态方法生成的.h文件中有对应的数据类型。**加载第三方库**感觉是最实用的，不管是so文件加密，还是使用不开源的so库，都需要加载已经生成的so文件。

注意/app/build.gradle中的sdk版本要改成自己android studio中sdk有的版本，如果没有也可根据android studio提示下载。

代码下载：<https://github.com/LycorisGuard/android>

最后编辑于2016.9.17

分类: 安卓

好文要顶

关注我

收藏该文

ciyze

关注 - 8

粉丝 - 18

+加关注

0

推荐

0

反对

« 上一篇：[Win7 x64下进程保护与文件保护\(ObRegisterCallbacks\)](#)
» 下一篇：[病毒分析要掌握的技能](#)

posted on 2016-05-19 00:33 [ciyze](#) 阅读(8607) 评论(3) [编辑](#) [收藏](#)

努力加载评论中...

- 【推荐】腾讯云海外1核2G云服务器低至2折，半价续费券限量免费领取！
- 【推荐】阿里云双11返场来袭，热门产品低至一折等你来抢！
- 【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】天翼云双十一翼降到底，云主机11.11元起，抽奖送大礼
- 【推荐】流程自动化专家UiBot，体系化教程成就高薪RPA工程师
- 【优惠】七牛云采购嘉年华，云存储、CDN等云产品低至1折

京东云

最强“可扩展平台”
云计算核“芯”引擎

11.11
京东云
年终采购季

云主机
最低 1 折

每人限购三台

立即秒杀

相关博文：

- 呕心沥血Android studio使用JNI实例
 - android studio JNI使用
 - android studio JNI使用
 - Android Studio中JNI -- 1 -- 配置方法
 - Androidstudio使用JNI实例
- » 更多推荐...

阿里云研究中心16本白皮书全套下载！涵盖人工智能、云计算等多项领域

腾讯云

11.11 智慧上云

爆品限时购
云服务器 1核2G 首年 88元

最新 IT 新闻：

- 35岁艺人高以翔录节目时猝死！猝死急救黄金四分钟一定要了解！
 - 奥迪计划6年内在德裁员9500人 节约66亿美元转型电动化
 - 买手机被骗6000元 理工男网购被骗后攻破骗子后台
 - 高思教育正式更名为爱学习教育集团 并获腾讯D+轮战略投资
 - 华为子公司底价拍得12万平商住地做人才房，售价不高于10050元/㎡
- » 更多新闻...