AS2.2使用CMake方式进行JNI/NDK开发

ITGeGe · 1 年前

之前写过一篇比较水的文章Android手机控制电脑撸出HelloWorld

里面用到了JNI/NDK技术

知 写文章 登录

这篇文章给大家介绍下JNI/NDK开发。采用的是Android Studio2.2开发环境，使用CMake方式进行开发。

> JNI（Java Native Interface）是Java与C/C++进行通信的一种技术，使用JNI技术，可以java调用C/C++的函数对象等等，Android中的Framework层与Native层就是采用的JNI技术。

我们知道，Android系统是基于Linux开发，采用的是linux内核 ，Android APP开发大部分也要和系统打交道，只是Android FrameWork 帮我们处理了和系统相关的操作， 我们从Android 系统的分成结构可以看出，Android FrameWork是通过JNI与底层的C/C++库交互，例如：FreeType，OpenGL，SQLite，音视频等等。

知 写文章 登录

如果我们程序也需要调用自己的C/C++函数库，就必须用到JNI/NDK开发。

NDK配置(最新的CMake方式)

Android Studio2.2版本已经完全支持ndk开发了。而且默认采用CMake方式。（传统方式不过多介绍了）

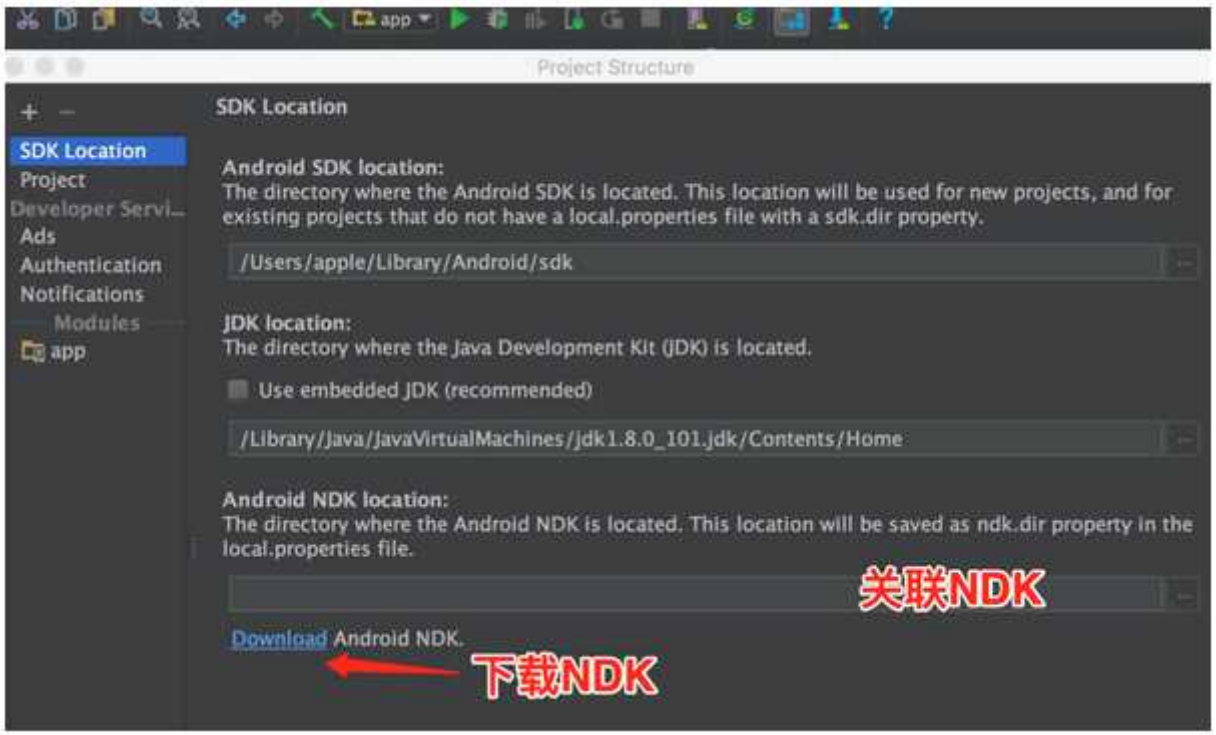
> CMake的优势

知 写文章 登录

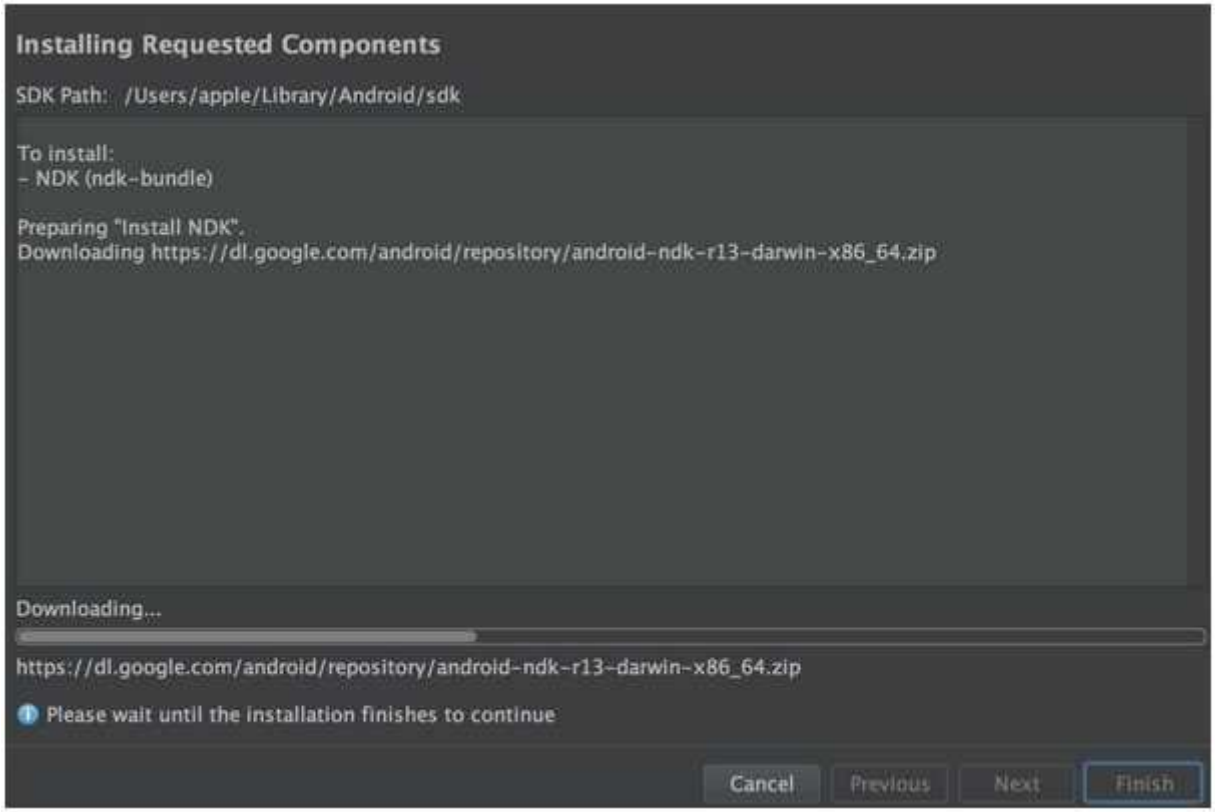
2. java引用的C/C++中的方法，可以直接ctrl+左键进入
3. 对于include的头文件，或者库，也可以直接的进入
4. 不需要配置命令行操作,手动的生成头文件,不需要配置android.useDeprecatedNdk=true属性

下载

首先需要下载NDK，来到设置界面点击下载NDK

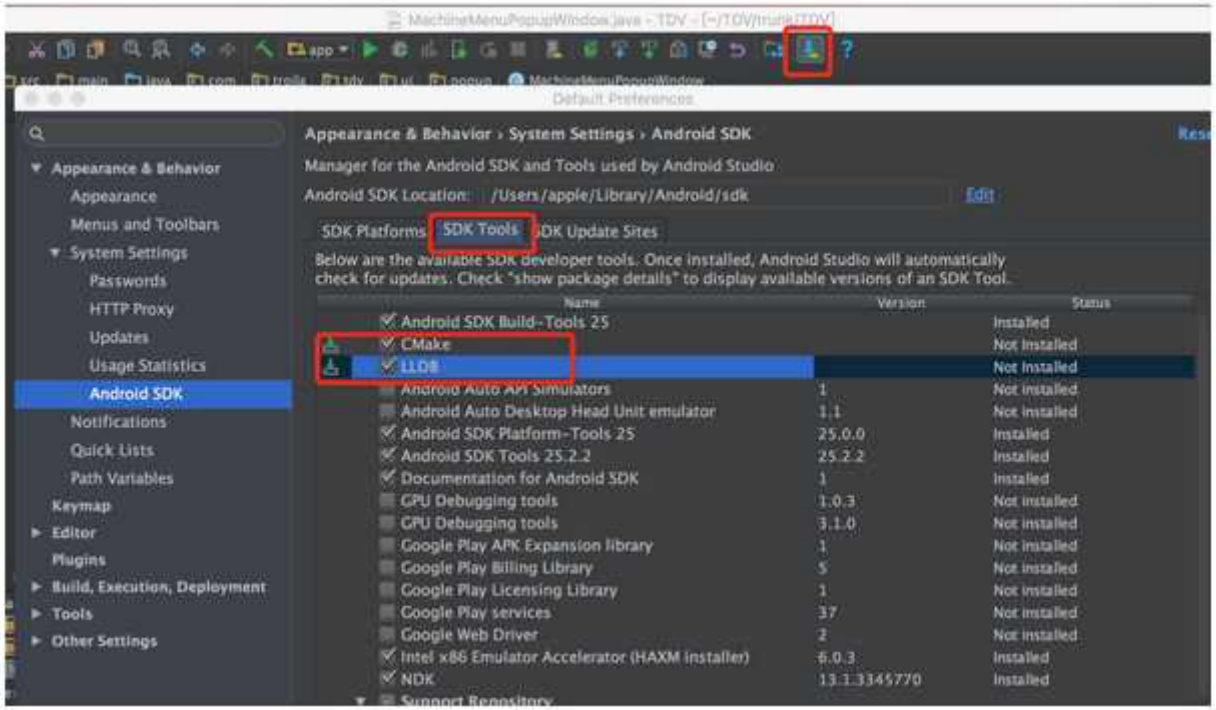


知 写文章 登录



安装完NDK，还可以选择配置一些工具。

知
写文章
登录

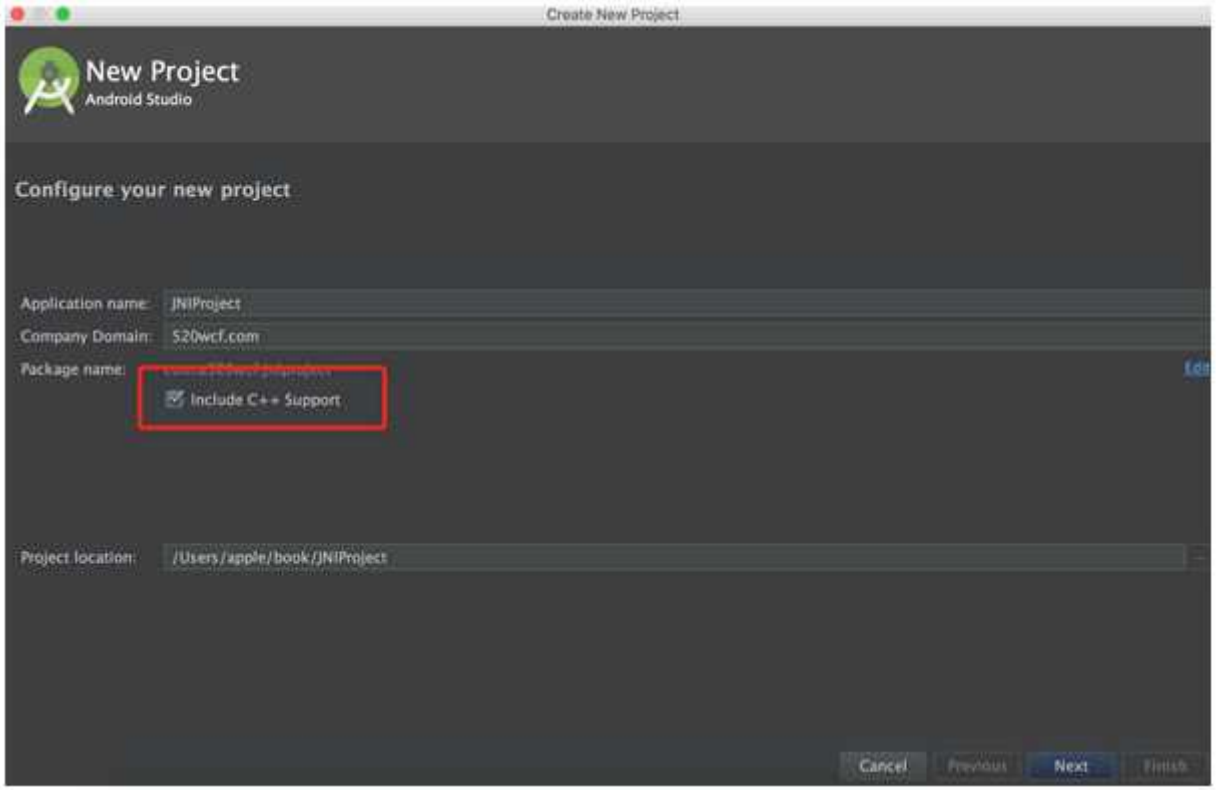


1. CMake: 外部构建工具。如果你准备只使用 ndk-build 的话，可以不使用它。（Android Studio2.2默认采用CMake）
2. LLDB: Android Studio上面调试本地代码的工具。

创建项目

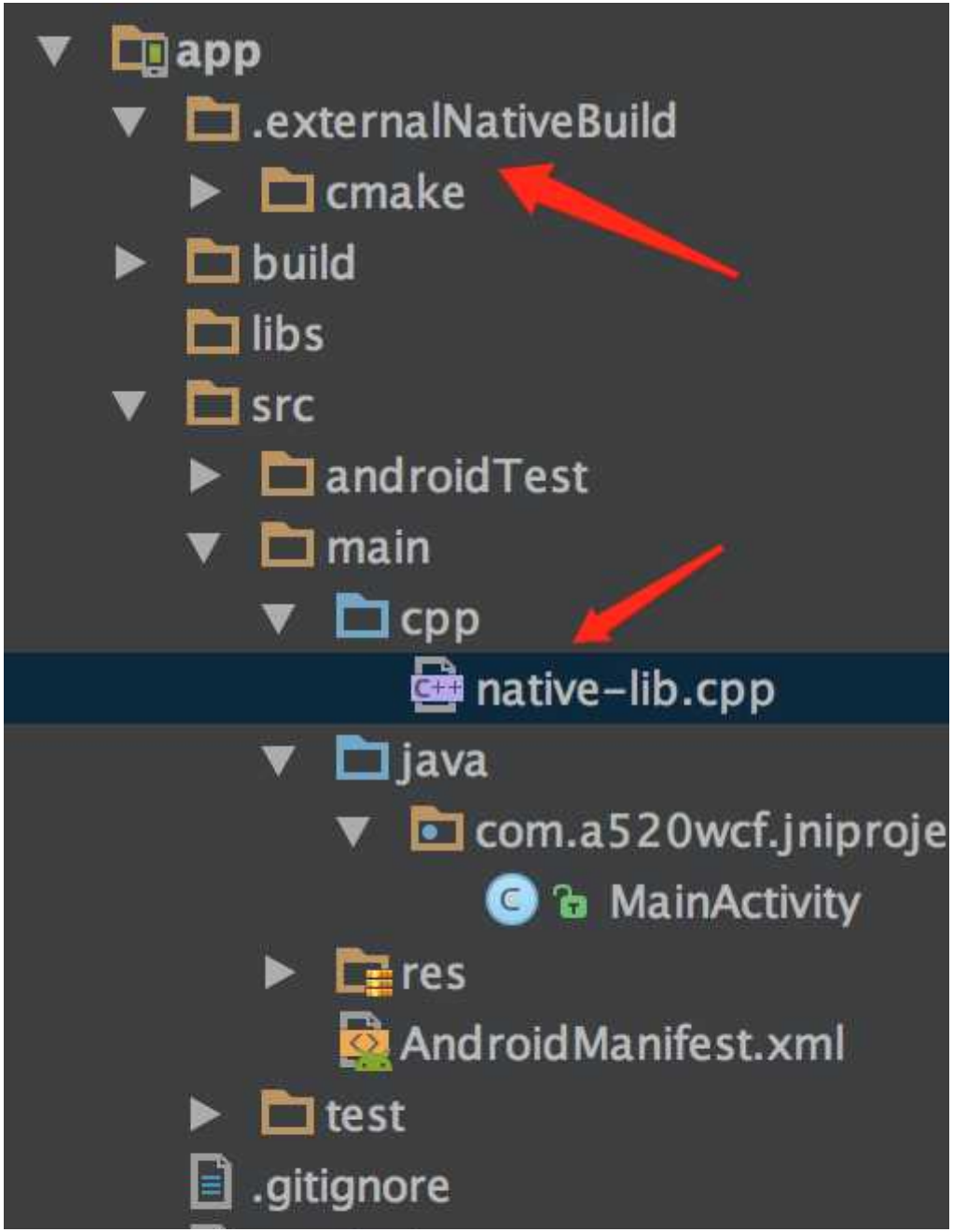
Android Studio升级到2.2版本之后，在创建新的project时，界面上多了一个Include C++ Support的选项。勾选它之后将会创建一个默认的C++与JAVA混编的Demo程序。

知 写文章 登录



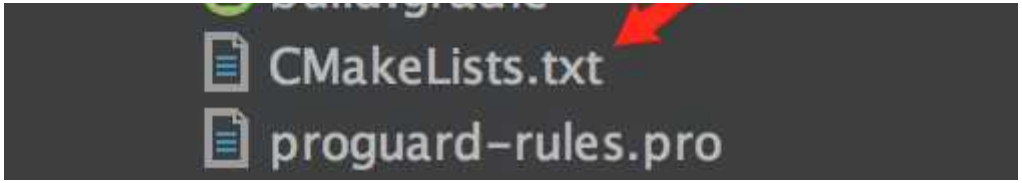
然后一路 Next，直到 Finish 为止即可。

知
写文章
登录


app
.externalNativeBuild
cmake
build
libs
src
androidTest
main
cpp
native-lib.cpp
java
com.a520wcf.jniproje
MainActivity
res
AndroidManifest.xml
test
.gitignore

知

写文章 登录



上面图的这三个文件都是默认生成的NDK项目的一部分：

1. .externalNativeBuild文件夹：cmake编译好的文件, 显示支持的各种硬件等信息。系统生成。

2. cpp文件夹：存放C/C++代码文件，native-lib.cpp文件是默认生成的，可更改。需要自己编写。

3. CMakeLists.txt文件：CMake脚本配置的文件。需要自己配置编写。

app/build.gradle也有所不同

```
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.0"
    defaultConfig {
        applicationId "com.a520wcf.jniproject"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
        externalNativeBuild {
            cmake {
                cppFlags ""  //如果使用C++11的标准,则使用cppFlags "-std=c++11"
            }
        }
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    externalNativeBuild {
        cmake {
            path "CMakeLists.txt" //配置文件路径
        }
    }
}
```

知 写文章 登录

如果你在创建工程选择C++11的标准,则使用cppFlags “-std=c++11”

```
externalNativeBuild {
    cmake {
        cppFlags "-std=c++11"
    }
}
```

> 来看一下，CMakeLists.txt文件中的具体配置

这个文件#开头的全是注释，里面不是注释的只有下面的内容。

```
cmake_minimum_required(VERSION 3.4.1) #指定cmake版本


add_library( #生成函数库的名字
             native-lib
             SHARED  #生成动态函数看
             src/main/cpp/native-lib.cpp ) #依赖的cpp文件

find_library( #设置path变量的名称
              log-lib
              #指定要查询库的名字
              log ) #在ndk开发包中查询liblog.so函数库(默认省略lib和.so),路径赋值给log-lib

target_link_libraries( #目标库,和上面生成的函数库名字一至
                       native-lib
                       #连接的库,根据log-lib变量对应liblog.so函数库
                       ${log-lib} )
```

> java代码

```java
public class MainActivity extends AppCompatActivity {

    // 加载函数库
    static {
        System.loadLibrary("native-lib");
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Example of a call to a native method
        TextView tv = (TextView) findViewById(R.id.sample_text);
        tv.setText(stringFromJNI());
    }

    /**本地方法，当前方法是通过C/C++代码实现*/
    public native String stringFromJNI();
}
```

上面java代码中的 stringFromJNI()方法用native关键字修饰，这个方法是通过C/C++代码实现的。

> native-lib.cpp 代码

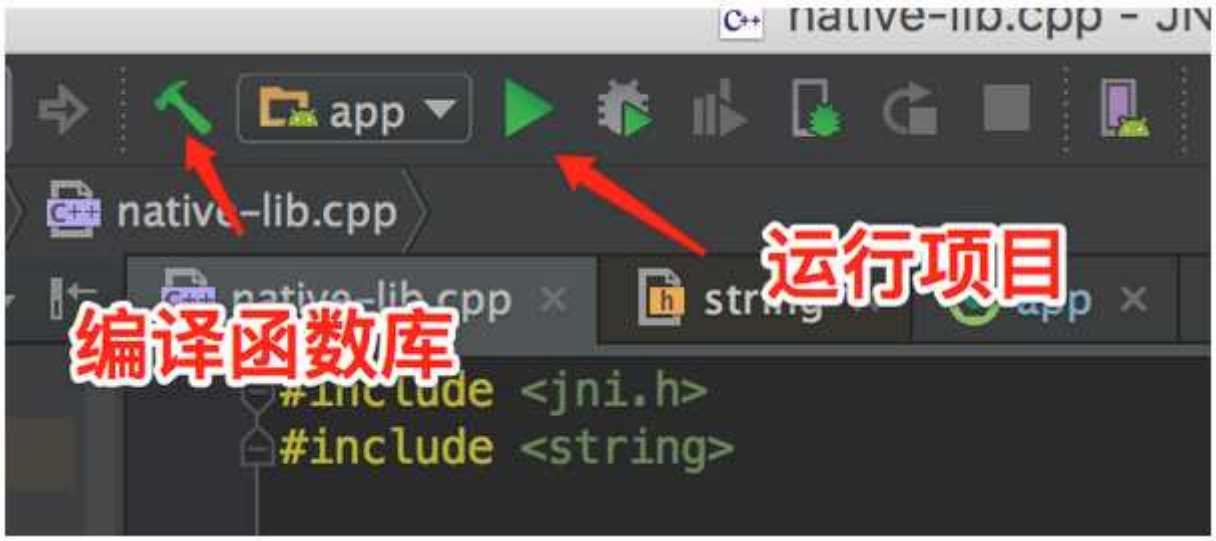
```cpp
#include <jni.h>
#include <string>

extern "C"
jstring
Java_com_a520wcf_jniproject_MainActivity_stringFromJNI(
        JNIEnv *env,
        jobject /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF(hello.c_str());
}
```

知 写文章 登录

上面的C++代码，定义的函数名是固定写法，Java_包名_类名_Java中方法名 ，通过这种命名方式就可以唯一对应到java中具体的方法，从而具体实现java中的native方法。

运行项目

修改完C/C++代码需要点击“锤子”图标进行编译，然后运行项目。



运行代码，就能看到效果，调用了C++方法在界面上显示了Hello from C++字符串。

如果你不是使用CMake而是使用传统方式进行开发，这时候就会使用了ndk -build来编译C/C++文件为so文件。

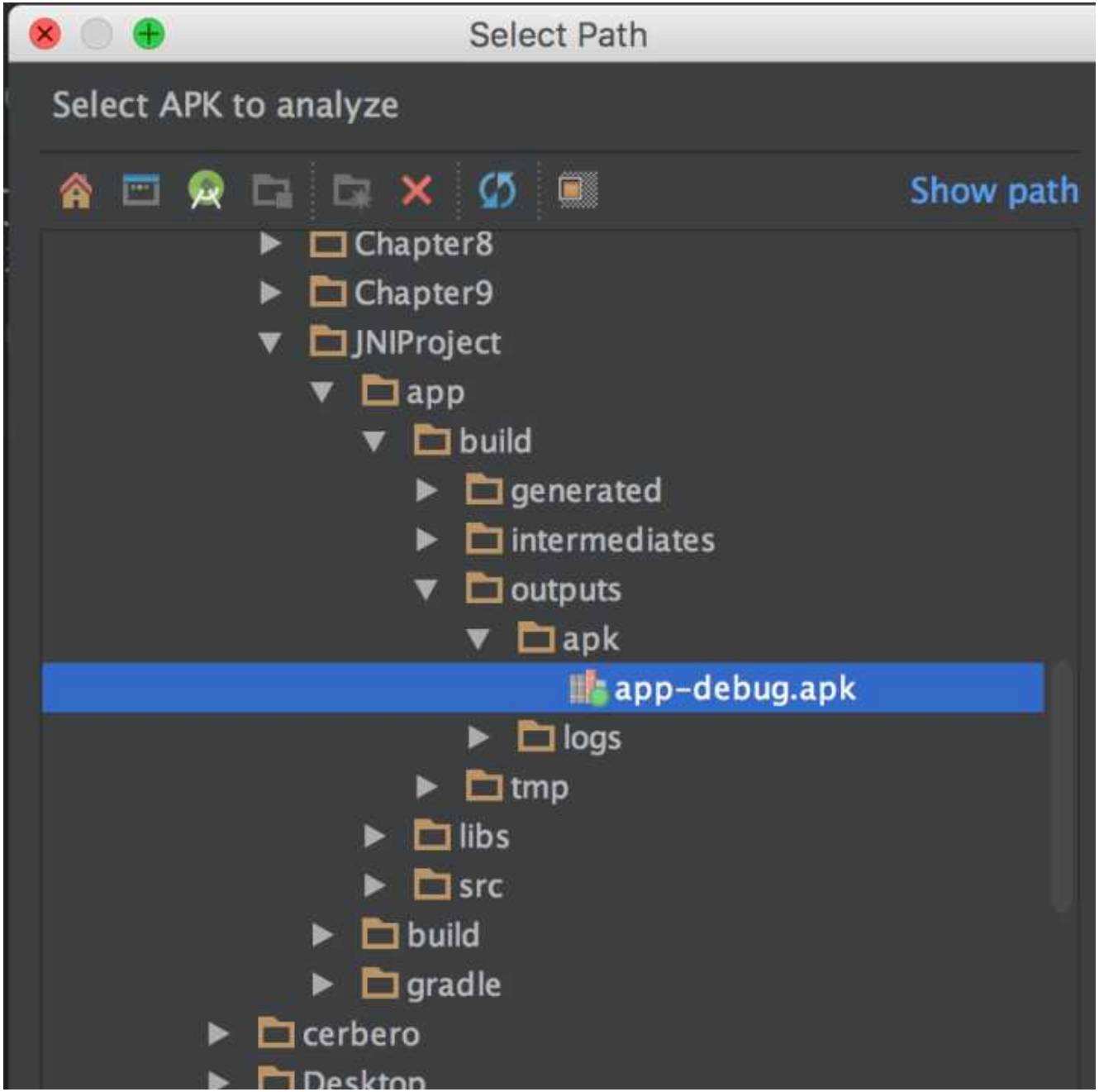
> 那么，我们安装运行的apk中，有对应的so文件吗？

知 写文章 登录

选择 apk，并点击 OK。

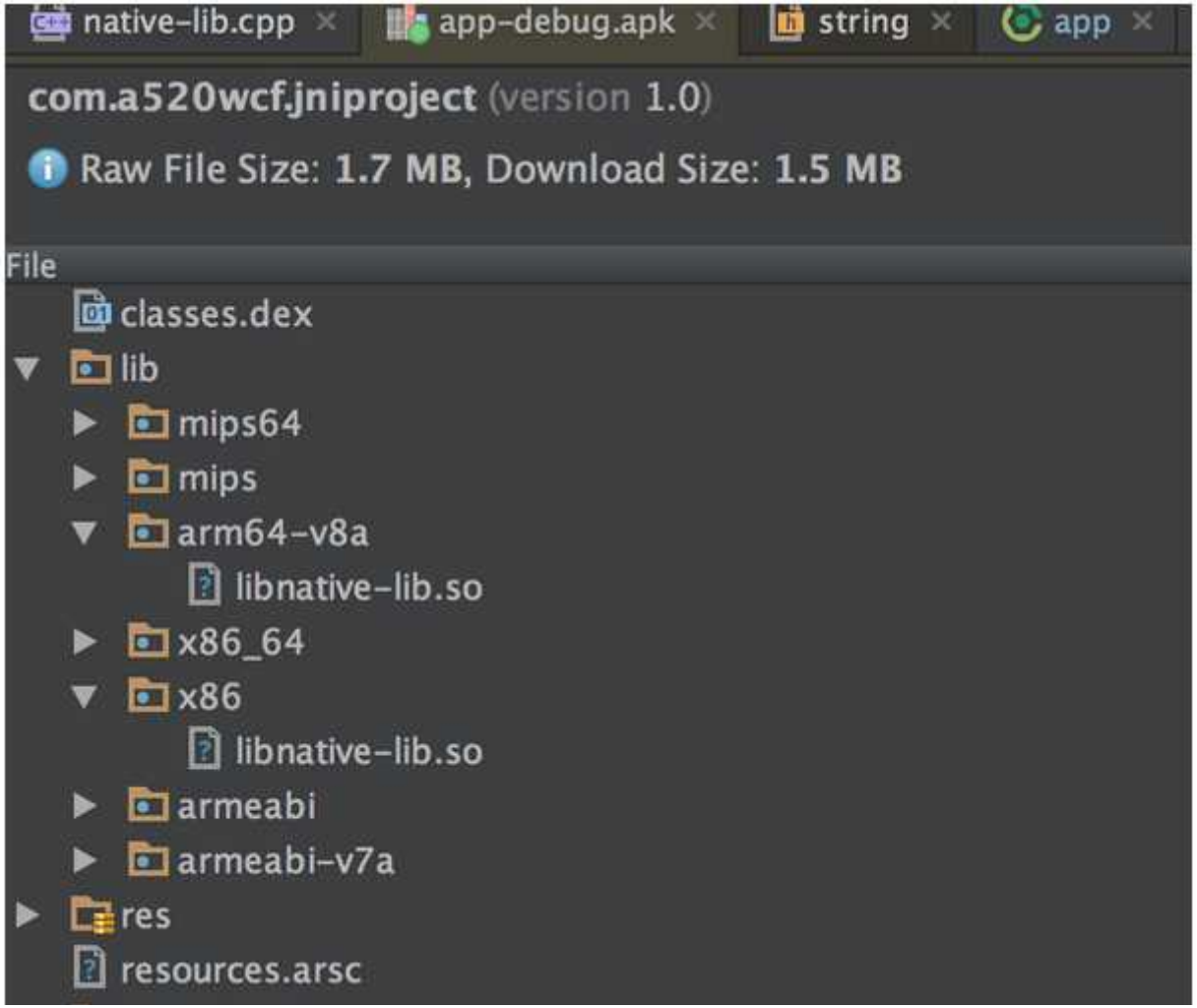
当前项目debug阶段的apk默认路径为 app/build/outputs/apk/app-debug.apk

知 写文章 登录


Select Path
Select APK to analyze
Show path
Chapter8
Chapter9
JNIProject
app
build
generated
intermediates
outputs
apk
app-debug.apk
logs
tmp
libs
src
build
gradle
cerbero
Desktop

知
写文章 登录

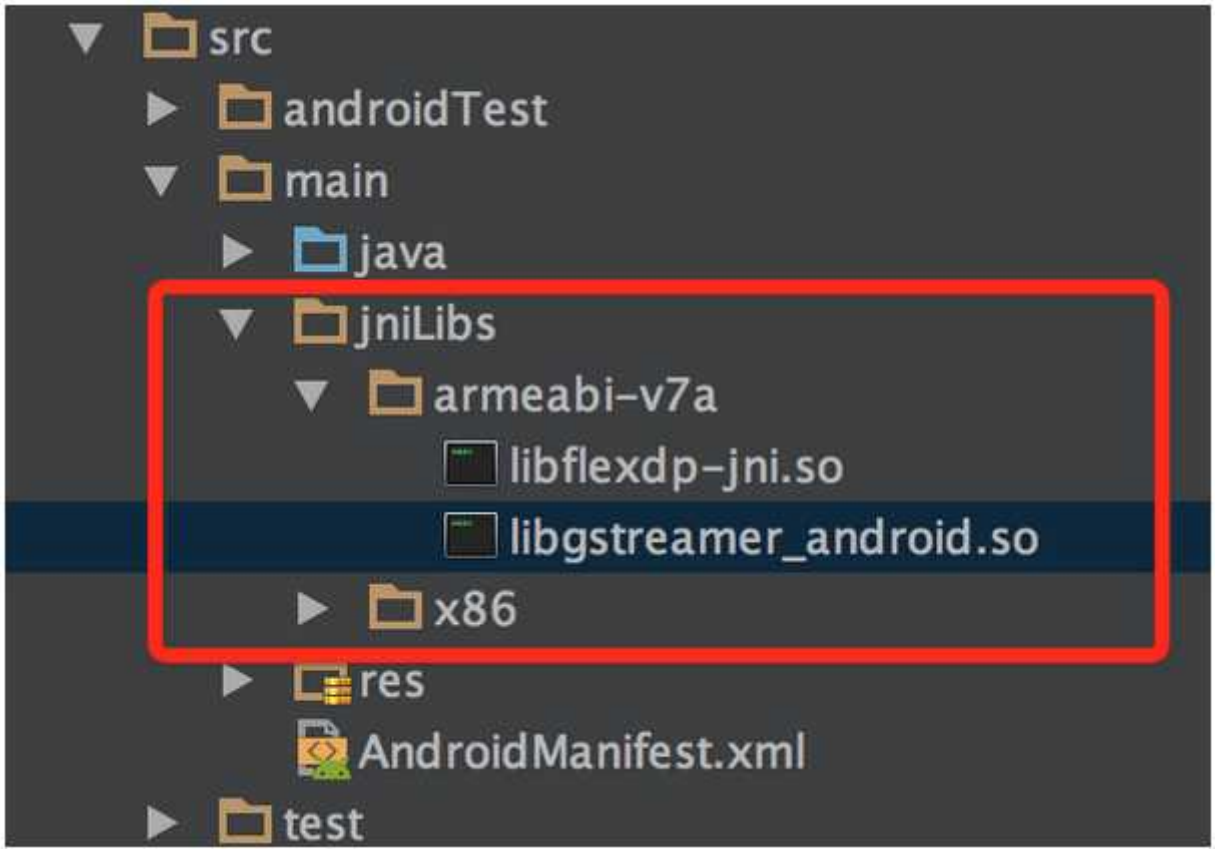
如下图，在 APK Analyzer 窗口中，选择 lib/x86/，可以看见 libnative-lib.so 。



知
写文章
登录

.so文件是动态函数库，写好的c/c++代码默认打包成函数库，就没法看到代码，只能使用了。

如果我们想在工程中使用其他人编译好的函数库，只需要根据不同的cpu架构把函数库在src/main/jniLibs目录下。



在java代码中也需要引入相应的函数库，编写一样的native方法。

手动添加native方法

知 写文章 登录

上面我们主要介绍程序自动生成的代码，接下来我们自己动手写写。
我们也可以在MainActivity中写一个native方法。

```
//手动添加native方法
public native String stringFromJNI2();
```

有红色警告，因为当前方法并没有找到对应的底层代码的实现。我们可以在报错的地方按下万能的快捷键alt+回车。

选择第一项，就会自动生成对应的底层方法。

```
JNIEXPORT jstring JNICALL
Java_com_a520wcf_jniproject_MainActivity_stringFromJNI2(JNIEnv *env, jobject instance) {

    // TODO


    return env->NewStringUTF(returnValue);
}
```

参考之前的方法，照着葫芦画瓢，把错误先修复下。

知 写文章 登录

```
extern "C"
jstring
Java_com_a520wcf_jniproject_MainActivity_stringFromJNI2(JNIEnv *env, jobject instance) {

    std::string hello = "Hello from C++ 2";
    return env->NewStringUTF(hello.c_str());
}
```

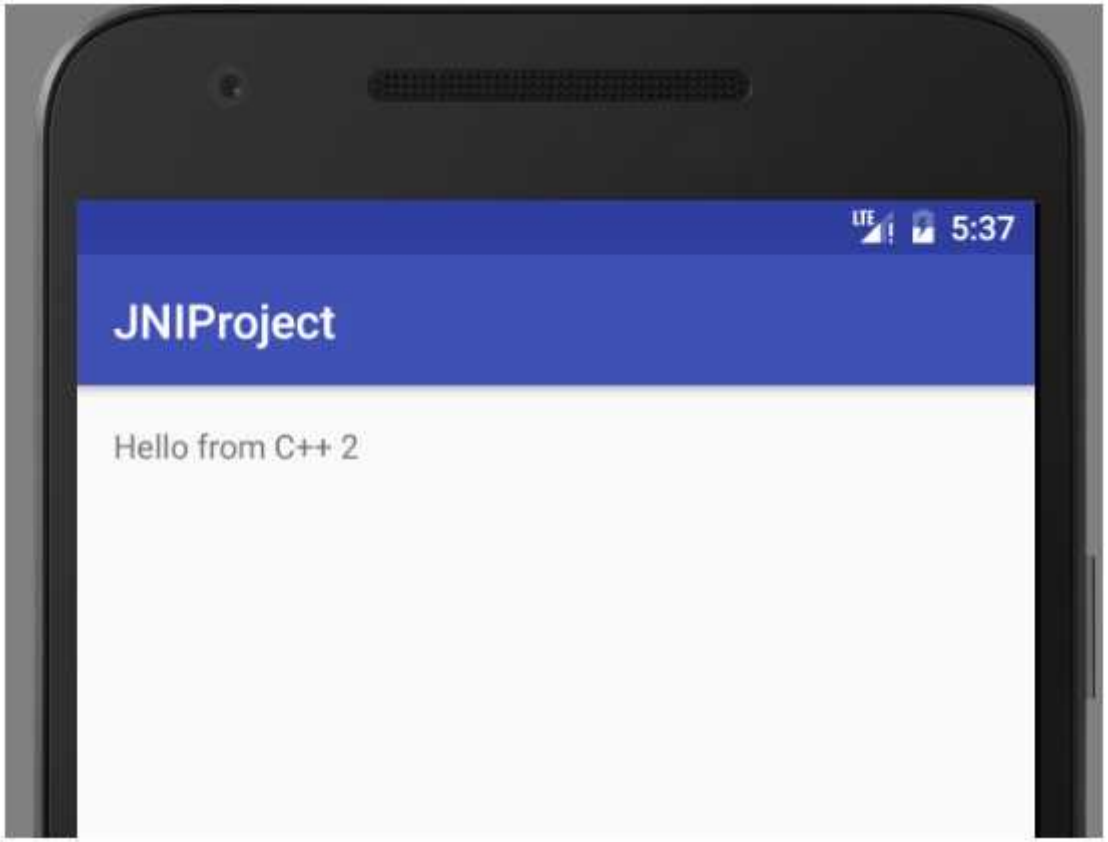
修改MainActivity代码，调用我们写的native方法。

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5
6      TextView tv = (TextView) findViewById(R.id.sample_text);
7      tv.setText(stringFromJNI2());//调用新写的native方法
8  }
```

编译运行当前程序。

运行结果:

知 写文章 登录



可以看到我们成功调用了我们自己创建的native方法。

来自：于连林520wcf

知 写文章 登录

Android Java Native Interface

Android NDK

5

收藏 分享 举报

还没有评论

写下你的评论...