



AS2.2使用CMake方式进行JNI/NDK开发



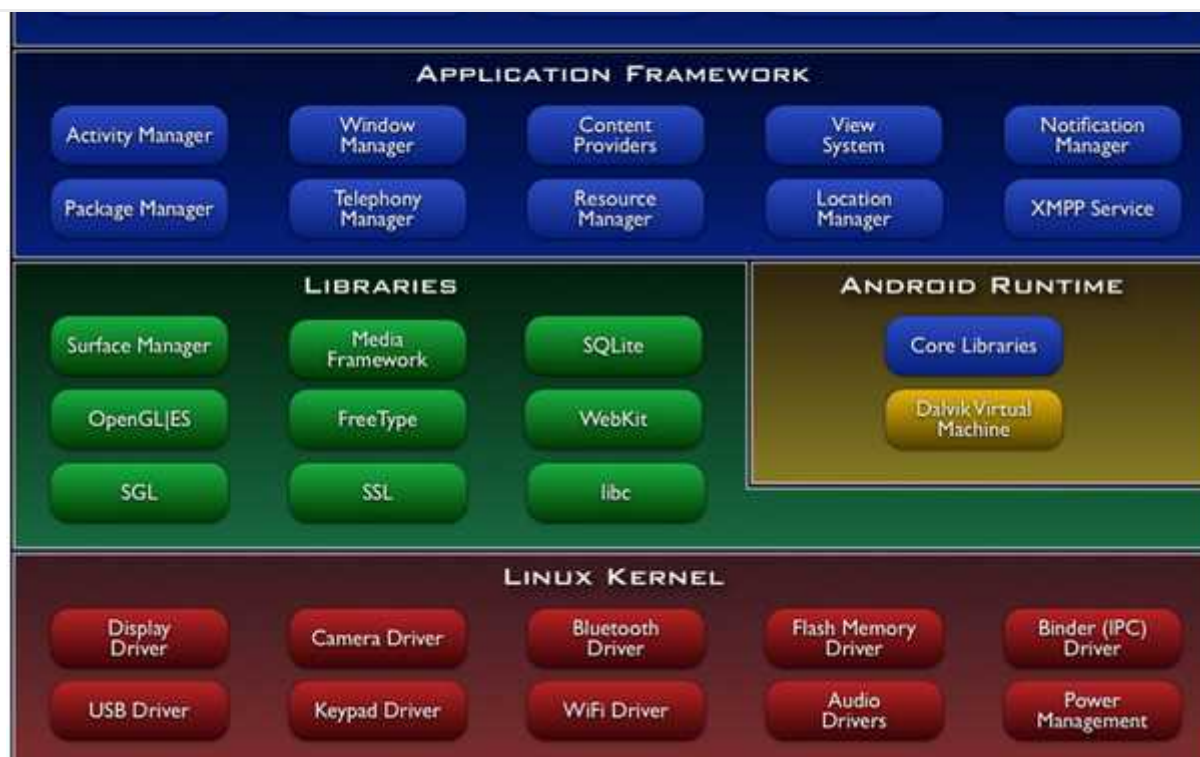
ITGeGe · 1 年前

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。

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。



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

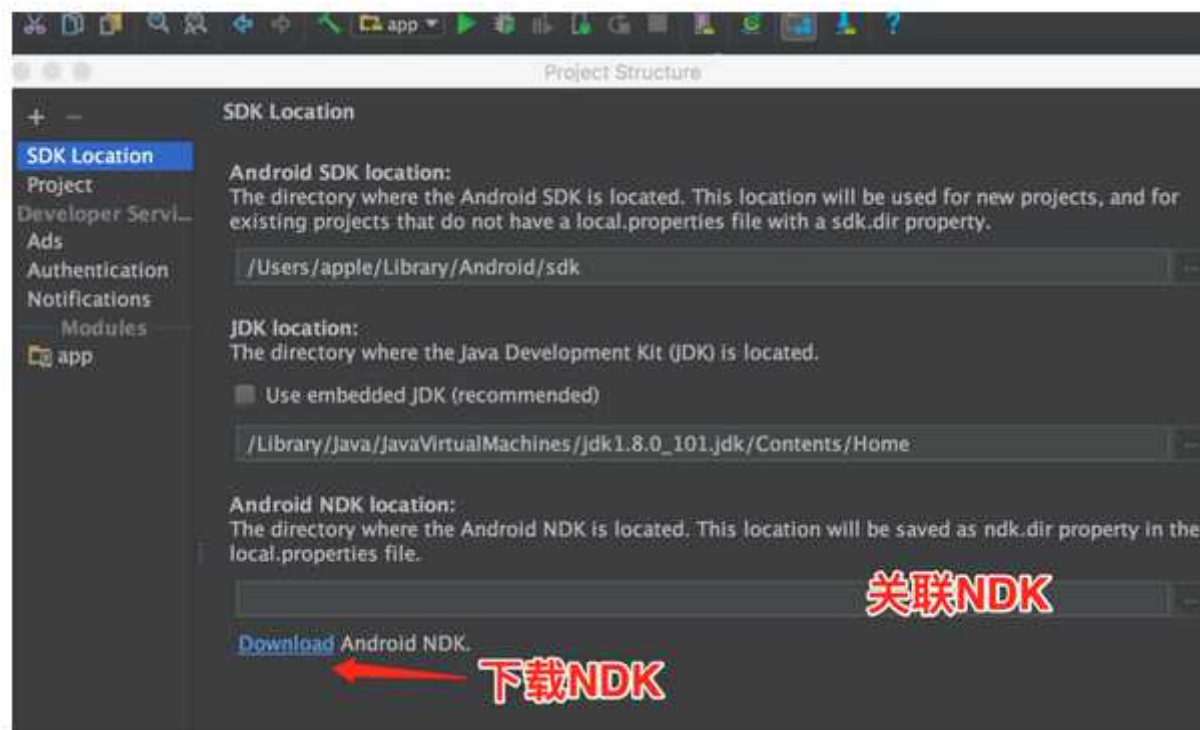
1. 可以直接的在C/C++代码中加入断点，进行调试

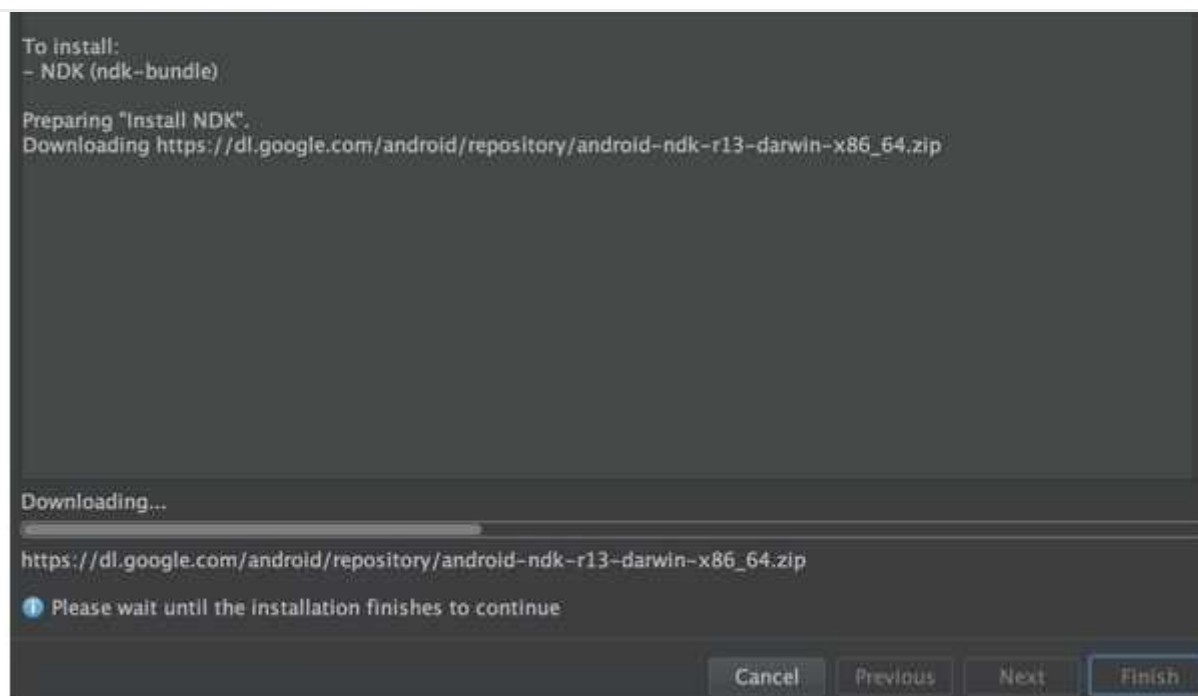
3. 对于include的头文件，或者库，也可以直接的进入

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

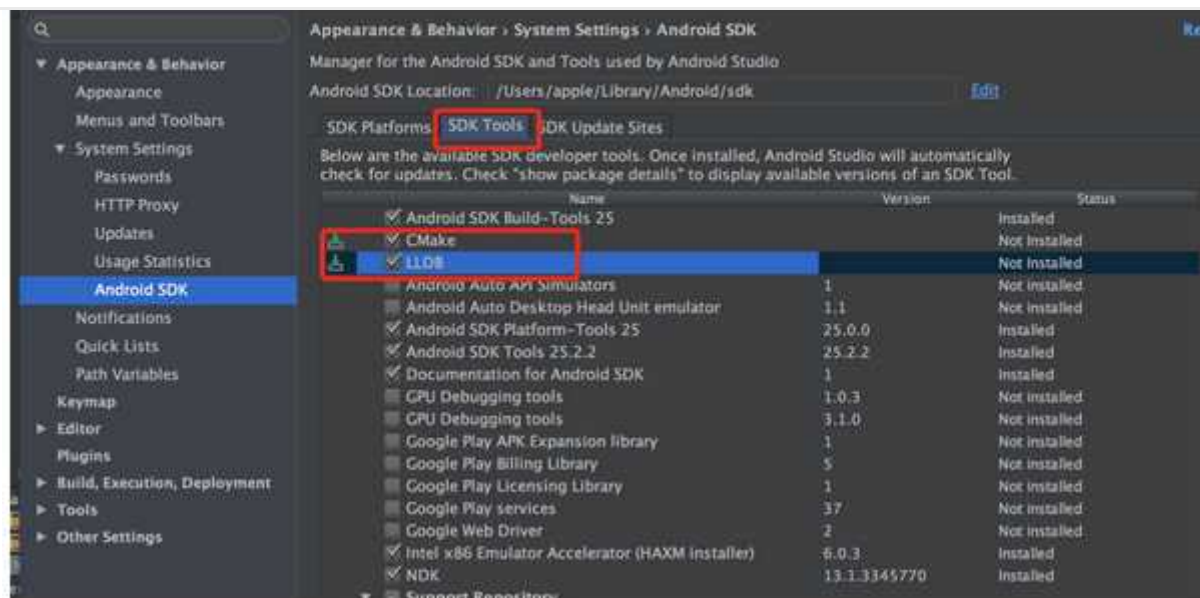
下载

首先需要下载NDK，来到设置界面点击下载NDK





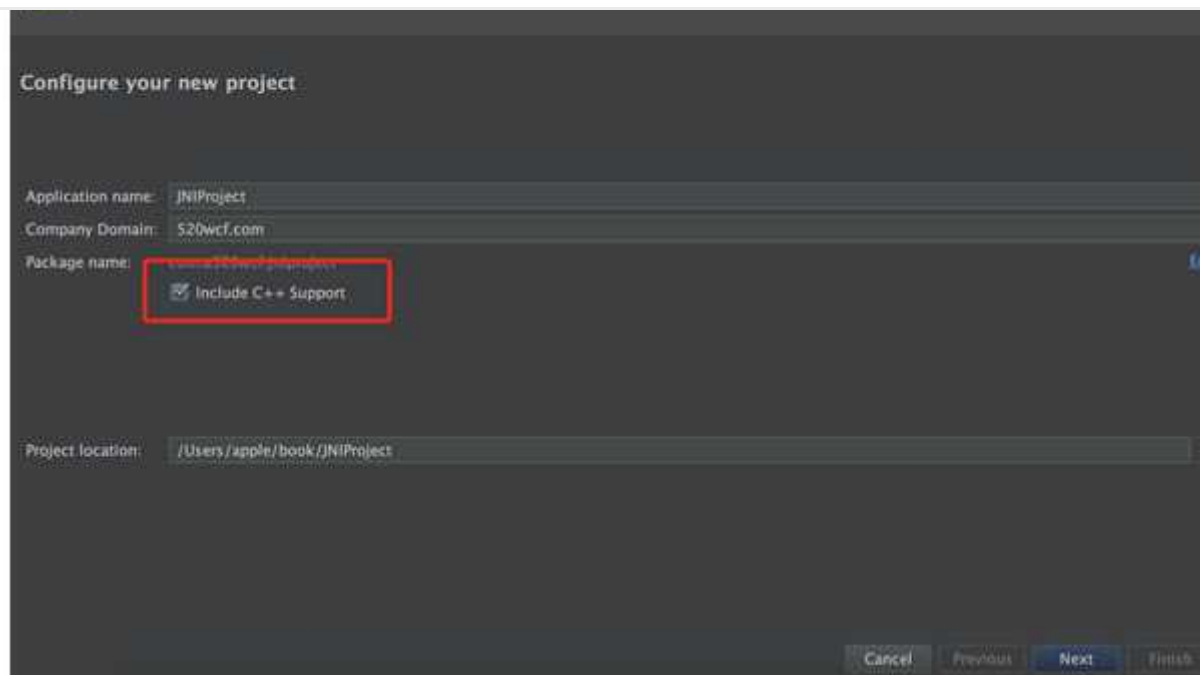
安装完NDK，还可以选择配置一些工具。



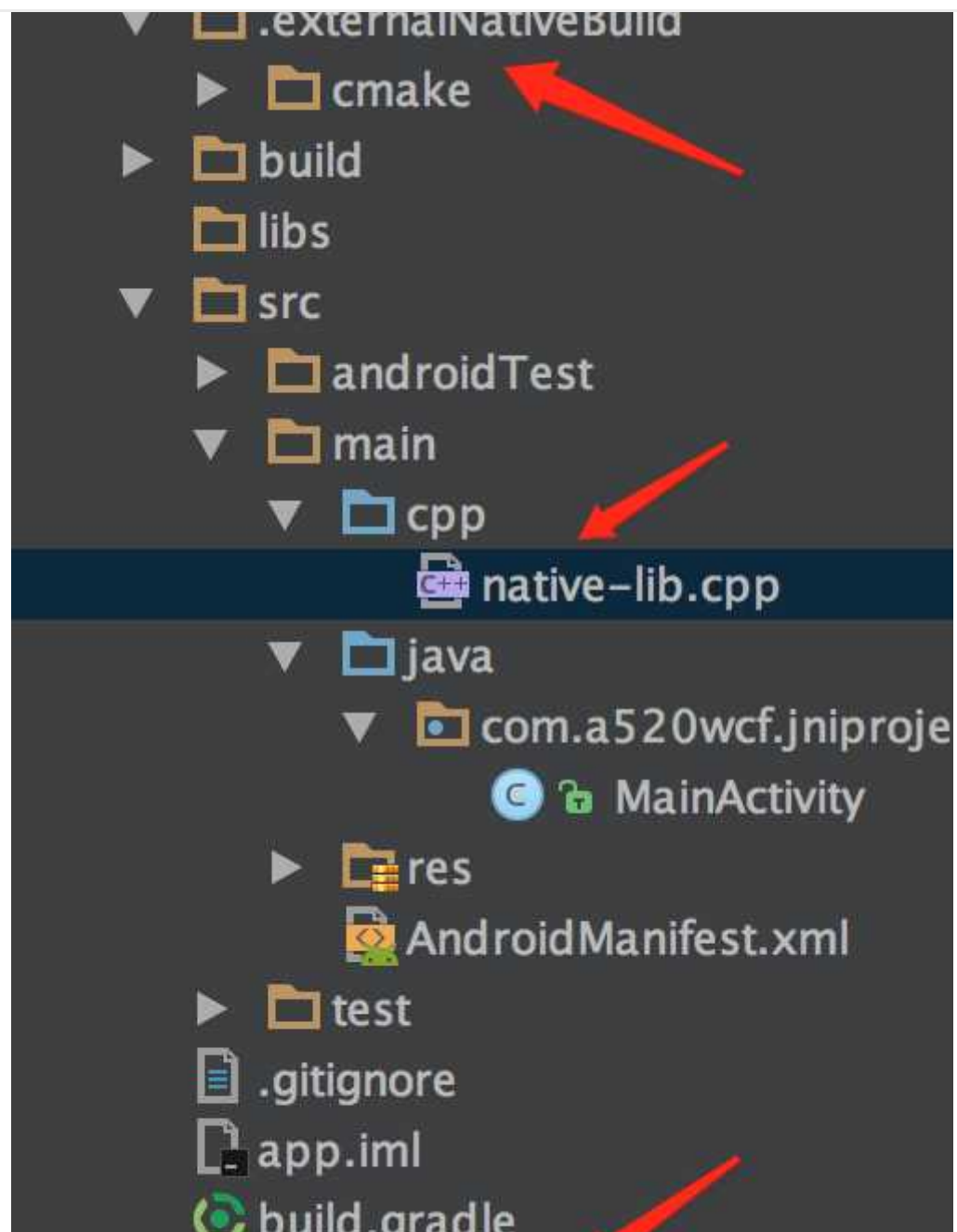
1. CMake: 外部构建工具。如果你准备只使用 ndk-build 的话，可以不使用它。（Android Studio 2.2默认采用CMake）
2. LLDB: Android Studio上面调试本地代码的工具。

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。



然后一路 Next，直到 Finish 为止即可。





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.as20wcf.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
        }
    }
}
```

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。

java代码

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

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。

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？

如果想验证一下apk是否有so文件，我们可以使用 APK Analyzer查看。
选择 Build > Analyze APK。

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

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

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

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。

参考之前的方法，照着葫芦画瓢，把错误先修复下。

修改MainActivity代码，调用我们写的native方法。

编译运行当前程序。

运行结果:

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](#)

本文由 [于连林520wcf](#) 发布于blog.csdn.net/yulianlin...

Android

Java Native Interface

Android NDK



5

☆ 收藏


📄 分享

⚠️ 举报



...

知

 写文章

登录

写下你的评论...