在下面的目录下面创建gradle.properties文件：

* /home/<username>/.gradle/ (Linux)
* /Users/<username>/.gradle/ (Mac)
* C:\Users\<username>\.gradle (Windows)

并在文件中增加：

1. org.gradle.daemon=true

同时修改项目下的gradle.properties文件也可以优化：

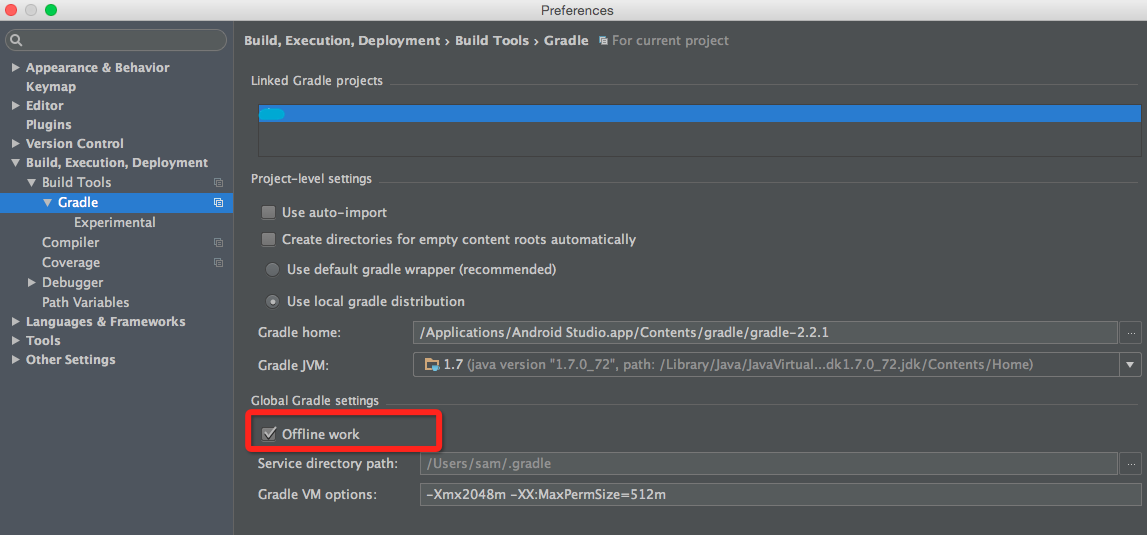
1. # Project-wide Gradle settings.
2. # IDE (e.g. Android Studio) users:
3. # Settings specified in this file will override any Gradle settings
4. # configured through the IDE.
5. # For more details on how to configure your build environment visit
6. # http://www.gradle.org/docs/current/userguide/build\_environment.html
7. # The Gradle daemon aims to improve the startup and execution time of Gradle.
8. # When set to true the Gradle daemon is to run the build.
9. # TODO: disable daemon on CI, since builds should be clean and reliable on servers
10. org.gradle.daemon=true
11. # Specifies the JVM arguments used for the daemon process.
12. # The setting is particularly useful for tweaking memory settings.
13. # Default value: -Xmx10248m -XX:MaxPermSize=256m
14. org.gradle.jvmargs=-Xmx2048m -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8
15. # When configured, Gradle will run in incubating parallel mode.
16. # This option should only be used with decoupled projects. More details, visit
17. # http://www.gradle.org/docs/current/userguide/multi\_project\_builds.html#sec:decoupled\_projects
18. org.gradle.parallel=true
19. # Enables new incubating mode that makes Gradle selective when configuring projects.
20. # Only relevant projects are configured which results in faster builds for large multi-projects.
21. # http://www.gradle.org/docs/current/userguide/multi\_project\_builds.html#sec:configuration\_on\_demand
22. org.gradle.configureondemand=true

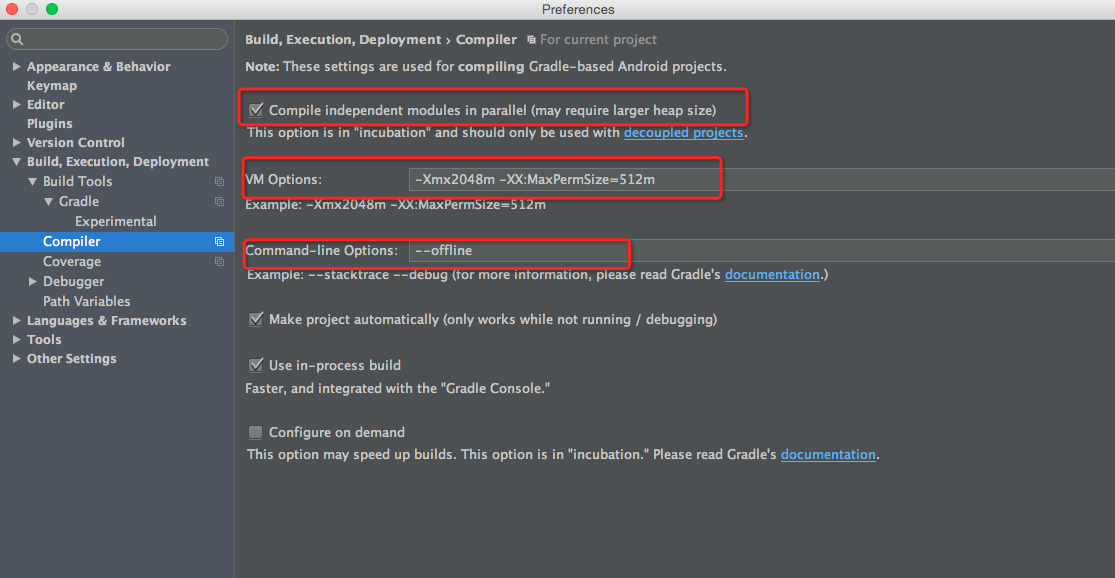
同时上面的这些参数也可以配置到前面的用户目录下的gradle.properties文件里，那样就不是针对一个项目生效，而是针对所有项目生效。

上面的配置文件主要就是做， 增大gradle运行的java虚拟机的大小，让gradle在编译的时候使用独立进程，让gradle可以平行的运行。

**修改android studio配置**

在android studio的配置中，开启offline模式，以及修改配置。实际上的配置和上面的一大段一样，主要是在这个地方配置的只会在ide构建的时候生效，命令行构建不会生效。





**命令行构建**

基于上面的配置，命令行构建时在命令后面加上这个参数即可 --daemon --parallel --offline。

**引入依赖库时使用aar**

使用网上第三方的依赖库时尽量使用aar，可以在maven<http://gradleplease.appspot.com/>或者githu<https://github.com/Goddchen/mvn-repo>搜索。

NDK配置

gradle.proprties

android.useDeprecatedNdk=true

在 studio 打开 Terminal 命令行工具，打开步骤是 View-->Tool Windows-->Terminal （Alt+F12）

在命令行中先进入到工程的 main 目录下  
输入命令：javah -d jni -classpath 自己编译后的 class 文件的绝对路径  
例如：javah -d jni -classpath  
C:\ASworkspace\MyJniTest\app\build\intermediates\classes\debug jary.com.myjnitest.JniTest

依赖刚才生成的 .h 头文件 ； 方法名与 .h 里面方法名保持一致 ；  
最后在 build.gradle defaultConfig 中添加如下代码

ndk {

moduleName "jary" //生成的so名字

abiFilters "armeabi", "armeabi-v7a", "x86" //输出指定三种abi体系结构下的so库。

stl "stlport\_static" //打开.c 的 debug , 下面第 4 点会讲到

}

.so 文件复制出来，如下图所示的放到 jniLibs

NDK debug

buildTypes {

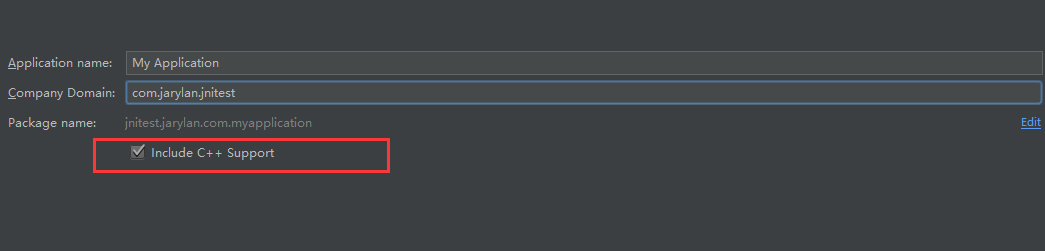
debug {

jniDebuggable true //此句不加在真机上 debug 不受影响，但是在虚拟机上不能 debug

}

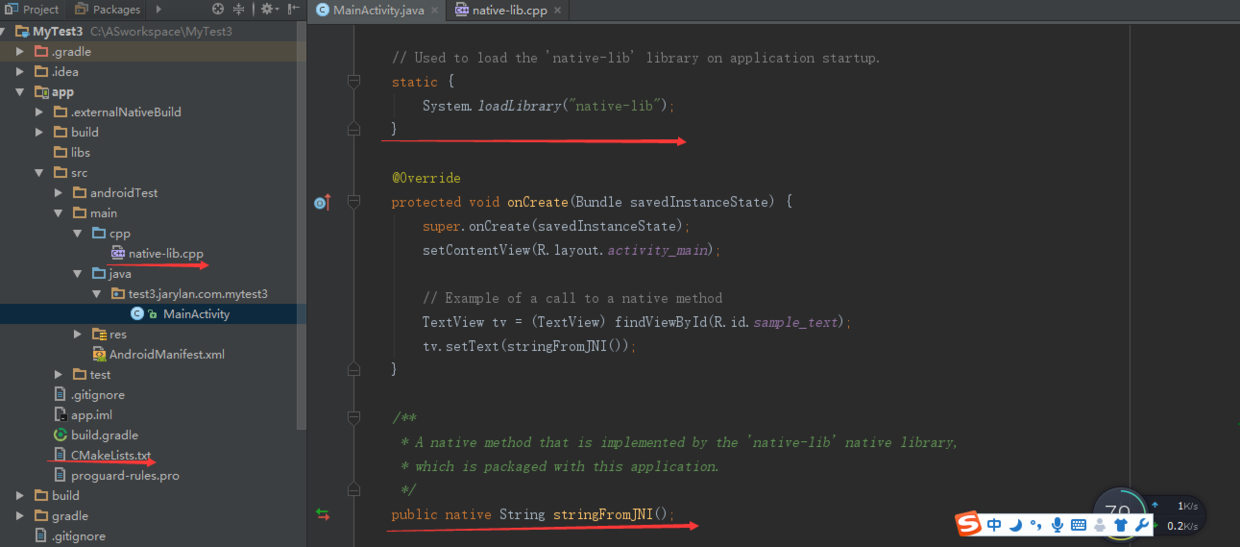
}

在创建项目的时候，记得将此处打上勾，其他默认就行；



Paste\_Image.png

然后 Android Studio 就会自动给你生成 jni demo；

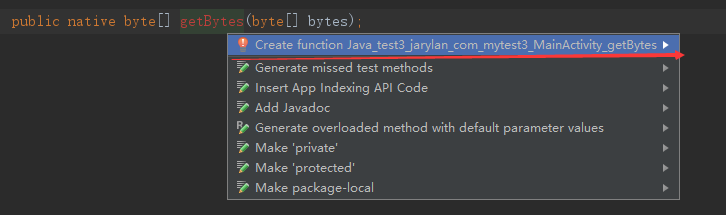


Paste\_Image.png

2、 拓展

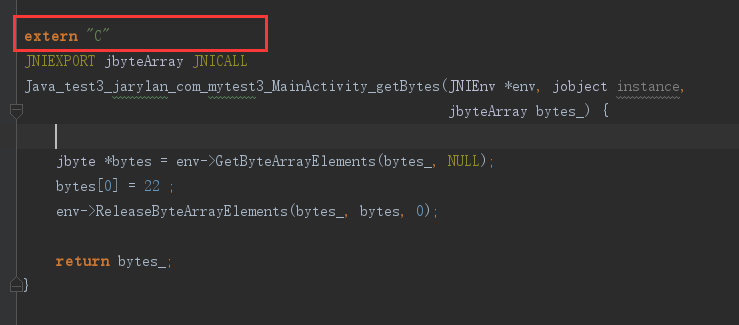
2.1 自定义方法 ：

比如你要自定义个 getBytes() 方法 ，直接声明这个方法，然后 利用快捷键 Alt+Enter 弹出提示，在 native-lib.cpp 中创建该方法（当然也可以自己按照命名方式自己去写）。如下图



Paste\_Image.png

在 native-lib.cpp 中声明后记得在方法名的上面一行添加 extern"C"，否则执行后会找不到该方法。



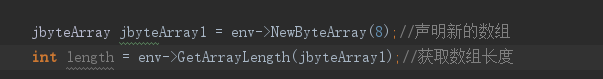
Paste\_Image.png

注 ：如要修改 byte 数组里面的值 ， 需要如上图所示 ，通过转换修改再转换回来；因为在这里 java 基本数据类型都通过 <jni.h> 转换了，具体可以点击进入 jni.h 里面查看源码

http://upload-images.jianshu.io/upload_images/3878387-cc1ff1fff71bb7d9.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240

Paste\_Image.png

如果要声明一个新的 byte[] ，获取数组的长度，如下图



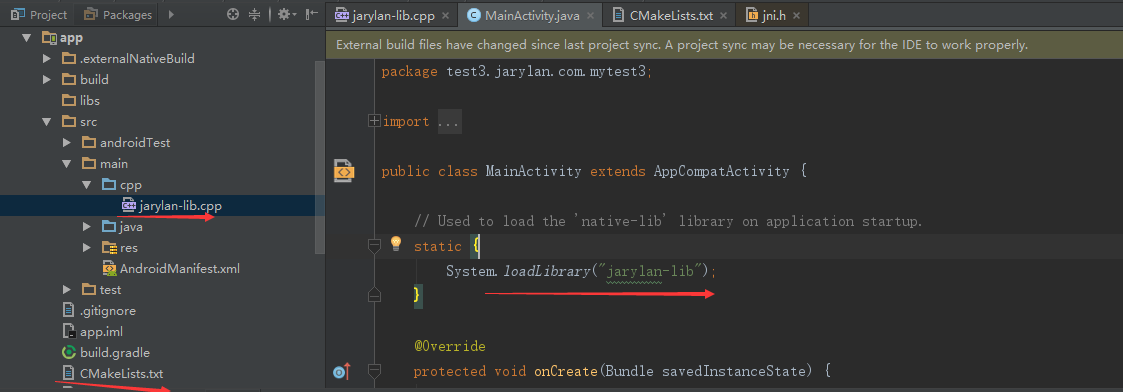
Paste\_Image.png

2.2 debug

直接在 .cpp 文件里面加断点 ； debug 运行就好了 ；

2.３ 修改 so 库名字

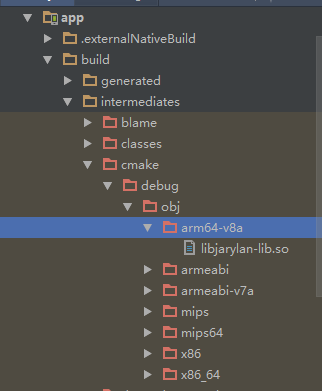
第一步，将 cpp 文件夹下的文件名 修改  
第二步，将此句 System.loadLibrary("native-lib"); 的名字修改  
第三步，将 CMakeLists.txt 文件里面所有 "native-lib" 的字段修改  
然后先 Clean Project ，再重新编译一下项目（ Ctrl + F9）



Paste\_Image.png

2.４ 导出 so 文件

编译之后再如下图路径可以找到编译后的 so 文件：



Paste\_Image.png

这里面有默认 7 中架构的 so 库； 如果你不需要生成那么多 ，可以在  
build.gradle 配置 ；



Paste\_Image.png

Clean Project ， 再重新编译 ，就只生成你配置的那几种了 ；