

IPV4 over IPV6 隧道协议实验

1.实验目的

IPV4 over IPV6，简称“4over6”是 IPV4 向 IPV6 发展进程中，向纯 IPV6 主干网过渡提出的一种新技术，可以最大程度地继承基于 IPV4 网络和应用，实现 IPV4 向 IPV6 平滑的过渡。

该实验通过实现 IPV4 over IPV6 隧道最小原型验证系统，让同学们对 4over6 隧道的实现原理有更加深刻的认识。

2.实验要求

在安卓设备上实现一个 4over6 隧道系统的客户端程序，内容如下：

- 1) 实现安卓界面程序，显示隧道报文收发状态(java 语言);
- 2) 启用安卓 VPN 服务(java 语言);
- 3) 实现底层通信程序，对 4over6 隧道系统控制消息和数据消息的处理(C 语言)。

3.实验内容

客户端可以分为前台和后台。

- 1) 前台是 java 语言的显示界面
 - 1、进行网络检测并获取上联物理接口 IPV6 地址;
 - 2、启动后台线程;
 - 3、开启定时器刷新界面;
 - 4、界面显示网络状态;
 - 5、开启安卓 VPN 服务。
- 2) 后台是 C 语言客户端与 4over6 隧道服务器之间的数据交互
 - 1、连接服务器;
 - 2、获取下联虚接口 IPV4 地址并通过管道传到前台;
 - 3、获取前台传送到后台的虚接口描述符;

- 4、读写虚接口；
- 5、对数据进行解封装；
- 6、通过 IPV6 套接字与 4over6 隧道服务器进行数据交互；
- 7、实现保活机制，定时给服务器发送 keepalive 消息。

4.实验帮助

4.1 消息类型定义

客户端消息的结构体定义如下：

```
struct Msg
{
    int length;           //整个结构体的字节长度
    char type;           //类型
    char data[4096];     //数据段
};
```

客户端主要用到下面五种消息类型：

类型(char)	长度(int)	数据(char[4096])	备注
100		null	IP 地址请求
101			IP 地址回应
102			上网请求
103			上网回应
104		null	心跳包

其中 101 类型报文的数据段，包含了 IP 地址、路由、三个 DNS，服务器会以字符串的形式把这些信息写入报文数据段，格式为“IP 路由 DNS DNS DNS”，字符串之间以空格隔开，类似“13.8.0.2 0.0.0.0 202.38.120.242 8.8.8.8 202.106.0.20”，收到数据后可以用标准 C 的 sscanf 函数把这些字段分别取出来，也可以直接传送到前台，前台通过 split 函数把数据存储到一个数组中。

4.2 流程解析

该客户端可以分为两大部分，前台和后台，前台是安卓界面的显示，后台是创建 IPV6

套接字，数据的封装与解封装，以及与 4over6 隧道服务端的通信。

整体流如图 4.1 所示：

- 1) 程序启动后，先检查网络状态；
- 2) 获取 IPV6 地址；
- 3) 开启后台线程；
- 4) 开启前台定时器刷新界面(间隔 1 秒)。



图 4.1 整体流程图

4.2.1 后台流程

后台流程图如图 4.2 所示，详细流程解析如下：

- 1) 创建 IPV6 套接字；
- 2) 连接 4over6 隧道服务器；
- 3) 开启定时器线程（间隔 1 秒）：
 - 1、读写虚接口的流量信息写入管道；
 - 2、获取上次收到心跳包距离当前时间的秒数 S；
 - 3、假如 S 大于 60，说明连接超时，就关闭套接字；
 - 4、S 小于 60 就每隔 20 秒给服务器发送一次心跳包。
- 4) 发送消息类型为 100 的 IP 请求消息；
- 5) while 循环中接收服务器发送来的消息，并对消息类型进行判断：
 - 1、 101 类型(IP 响应)：
 - (1) 取出数据段，解析出 IP 地址，路由，DNS；

- (2) 把解析到的 IP 地址，路由，DNS 写入管道；
- (3) 从管道读取前台传送来的虚接口文件描述符；
- (4) 创建读取虚接口线程：
 - A. 持续读取虚接口；
 - B. 记录读取的长度和次数；
 - C. 封装 102(上网请求)类型的报头；
 - D. 通过 IPV6 套接字发送给 4over6 隧道服务器。

2、103 类型(上网应答):

- (1) 取出数据部分；
- (2)写入虚接口；
- (3)存下写入长度和写入次数。

3、104 类型(心跳包):

- (1)记录当前时间到一个全局变量。

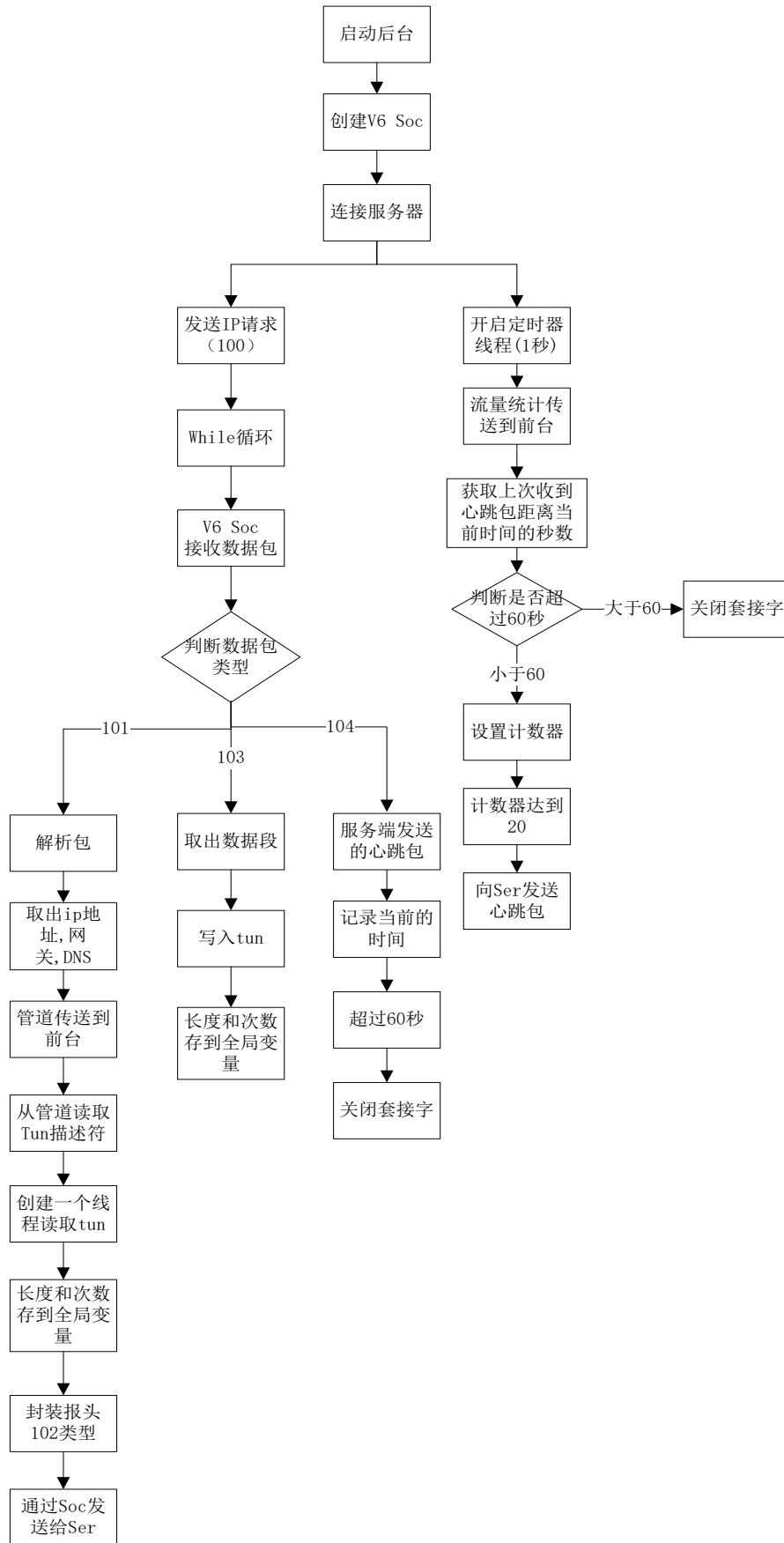


图 4.2 后台详细流程图

4.2.2 前台流程

前台的详细流程图如图 4.3 所示，前台定时器主要功能为定时刷新显示界面，显示流量信息，定时器功能解析如下：

- 1) 开启定时器之前，创建一个读取 IP 信息管道的全局标志位 **flag**，默认置 0；
- 2) 开始读取管道，首先读取 IP 信息管道，判断是否有后台传送来的 IP 等信息；
- 3) 假如没有，下次循环继续读取；
- 4) 有 IP 信息，就启用安卓 VPN 服务(此部分在第五章附录部分有详细解释)；
- 5) 把获取到的安卓虚接口描述符写入管道传到后台；
- 6) 把 **flag** 置 1，下次循环不再读取该 IP 信息管道；
- 7) 读取流量信息管道；
- 8) 从管道读取后台传来的实时流量信息；
- 9) 把流量信息进行格式转换；
- 10) 显示到界面；
- 11) 界面显示的信息有运行时长、上传和下载速度、上传总流量和包数、下载总流量和包数、下联虚接口 **V4** 地址、上联物理接口 **IPV6** 地址。

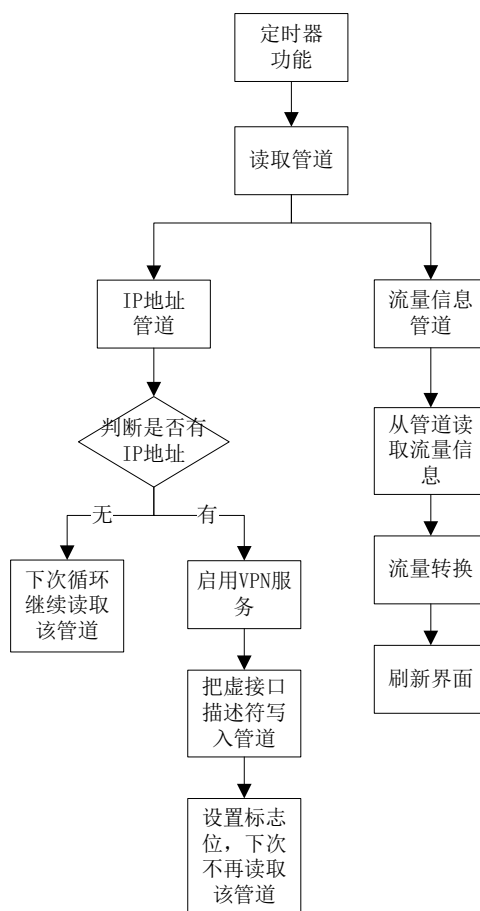


图 4.3 前台详细流程图

4.2.3 前后台通信流程

前台是 java 语言的显示界面，后台是 C 语言的数据交互，前后台之间通过管道进行通信，这里创建了两个管道，一个是 IP 信息管道，一个是流量信息管道，两个管道分开处理相对简单，通信的详细流程如图 4.4 所示。

- 1) 后台获取到 4over6 服务器发送来的 IP 地址等信息；
- 2) 把这些信息解析出来，写入 IP 信息管道；
- 3) 前台从 IP 信息管道读取到 IP 地址等信息；
- 4) 开启安卓 VPN 服务；
- 5) 把安卓虚接口描述符写入 IP 信息管道；
- 6) 后台从 IP 信息管道获取安卓虚接口文件描述符；
- 7) 对该虚接口进行读写操作；
- 8) 记录读写长度和次数；
- 9) 在后台定时器线程，定时写入流量信息管道；

10) 前台从流量信息管道获取流量信息;

11) 定时刷新界面的流量信息。

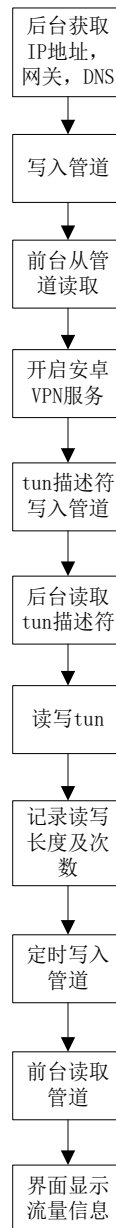


图 4.4 前后台通信流程

4.3 编译环境搭建

实验环境

操作系统: windows 7 x64

开发软件: Android studio 1.4

JDK 版本: 1.8.0_51

NDK 版本: android-ndk-r10e

如果是 x64 位的操作系统, 请直接在以下的网盘中下载 Android studio、JDK、NDK 即可:

链接: <http://pan.baidu.com/s/1mhf47kk> 密码: pt0c

4.3.1 安装 jdk

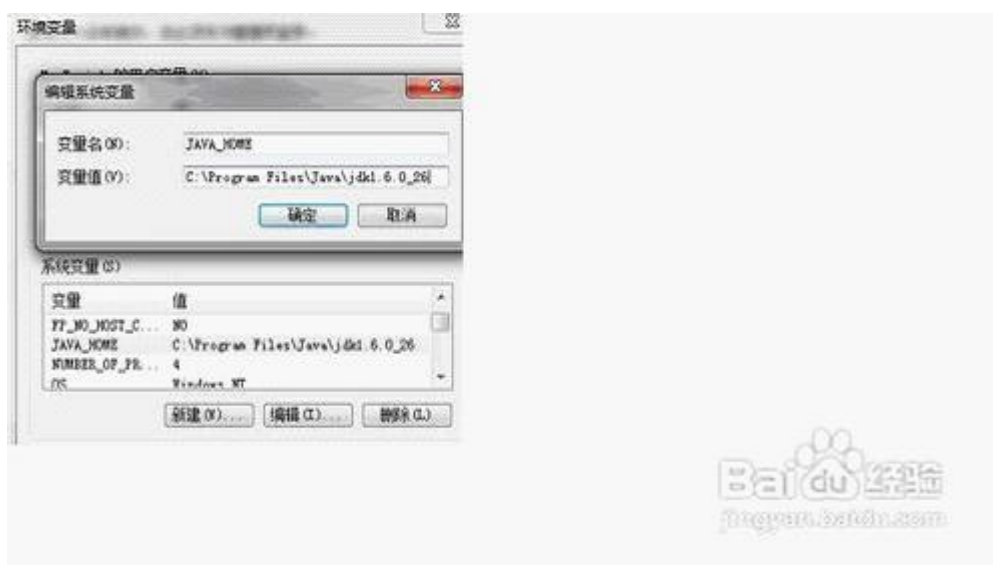
4.3.1.1 下载 jdk

首先查看自己的电脑是 32 位系统还是 64 位系统, 然后选择 java SE Development Kit 中相应的版本来下载。可以到此链接中进行下载 <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>, 下载完成后, 安装, 最好安装在默认目录下, 要不然后面容易出错。

4.3.1.2 配置 jdk 环境变量

安装完成后需要设置环境变量从而使编译器正常使用。右击“计算机”, 选择“属性”, 选择左边“高级系统设置”, 选择上面“高级”选项卡, 点击右下角“环境变量”按钮。接下来弹出的对话框会出现用户变量和系统变量。用户变量对当前登录账户有效, 系统变量对所有用户都有效。

在系统变量里点击新建, 变量名填写 JAVA_HOME, 变量值填写 JDK 的安装路径, 把自己 jdk 的安装路径复制进去就可以。



在系统变量里点击新建变量名填写 CLASSPATH，变量值填写“.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar”。注意不要忘记前面的点和中间的分号。



在系统变量里找到 Path 变量，这是系统自带的，不用新建。双击 Path，由于原来的变量值已经存在，故应在已有的变量后加上“;%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin”。注意前面的分号。



至此，应有的环境变量已经配置完毕。验证的方法：在运行框中输入 cmd 命令，回车后输入 java，按回车出现以下画面



```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\MacIavish>java
Usage: java [-options] class [args...]
        (to execute a class)
or java [-options] -jar jarfile [args...]
        (to execute a jar file)

where options include:
    -client          to select the "client" VM
    -server          to select the "server" VM
                     is a synonym for the "client" VM (deprecated)
                     The default VM is client.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                     A ; separated list of directories, JAR archives,
                     and ZIP archives to search for class files.
    -D<name>=<value>  set a system property
    -verbose[:class[:jnil]
                     enable verbose output
    -version          print product version and exit
    -version:<value>
```

输入 javac,按回车出现以下画面,则表示设置成功。



```
C:\Users\MacIavish>javac
用法: javac <选项> <源文件>
其中, 可能的选项包括:

-g          生成所有调试信息
-g:none     不生成任何调试信息
-g:{lines,vars,source} 只生成某些调试信息
-nowarn     不生成任何警告
-verbose    输出有关编译器正在执行的操作的消息
-deprecation 输出使用已过时的 API 的源位置
-classpath <路径> 指定查找用户类文件和注释处理程序的位置
-cp <路径> 指定查找用户类文件和注释处理程序的位置
-sourcepath <路径> 指定查找输入源文件的位置
-bootclasspath <路径> 指定引导类文件的位置
-extdirs <目录> 指定安装包的扩展目录的位置
-endorsestdirs <目录> 指定签名的标准路径的位置
-proc:{none,only} 控制是否执行注释处理和/或编译
-processor <class1>[,<class2>,<class3>...] 要运行的注释处理程序的名称, 绕过默认的搜索进程
-processorpath <路径> 指定查找注释处理程序的位置
-d <目录> 指定存放生成的类文件的位置
-s <目录> 指定存放生成的源文件的位置
-implicit:{none,class} 指定是否为隐式引用文件生成类文件
-encoding <编码> 指定源文件使用的字符编码
-source <版本> 提供与指定版本的源兼容性
-target <版本> 生成特定 VM 版本的类文件
```

4.3.1.3 配置失败解决方法

假如输入 javac 提示 javac 不是内部或外部命令,说明 javac 没有配置成功,下面是解决方案,假如没有出现此问题,请略过。

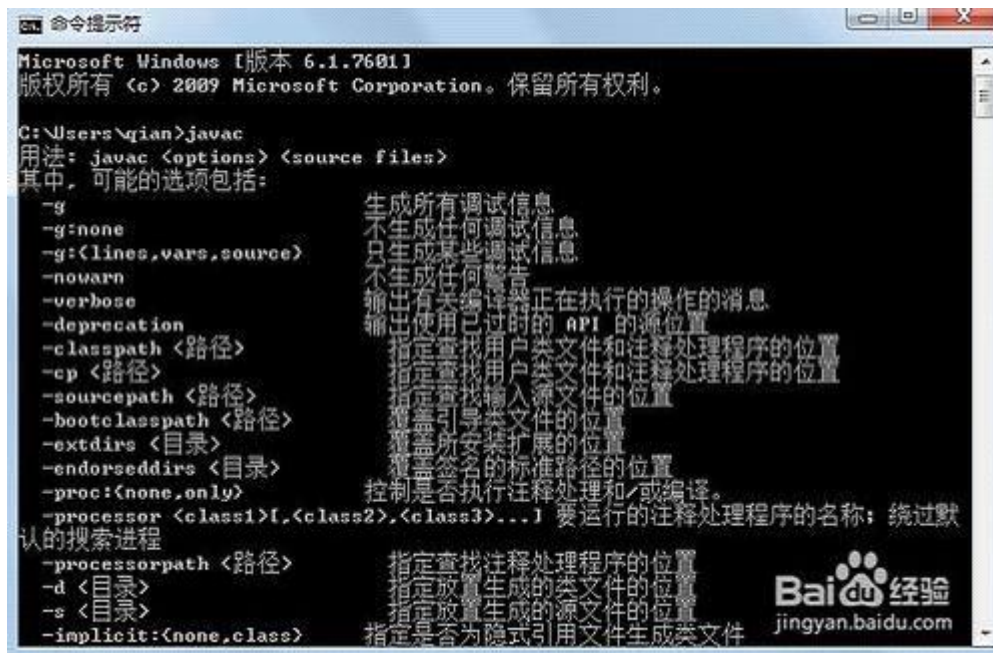
首先右击“我的电脑”,选择“属性”,再选择“高级系统设置”
选择“环境变量”,再次进入环境变量设置界面,在上面一栏选择新建。



变量名“Path”，然后找到自己jdk安装的路径，找到\bin，然后将路径复制到变量值。



最后点击确定，我们再次键入 javac，就会发现问题已经解决了。



```
命令提示符
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

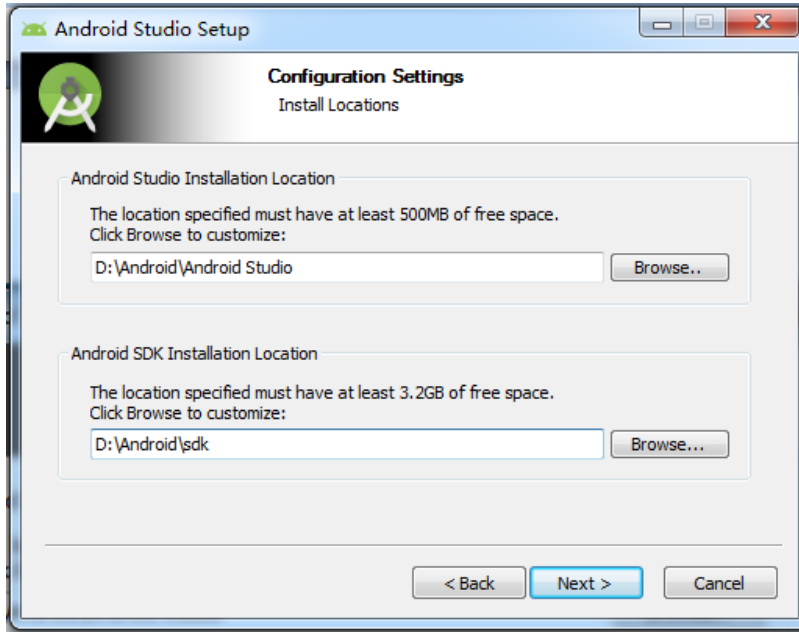
C:\Users\qian>javac
用法: javac <options> <source files>
其中, 可能的选项包括:
-g 生成所有调试信息
-g:none 不生成任何调试信息
-g:<lines,vars,source> 只生成某些调试信息
-nowarn 不生成任何警告
-verbose 输出有关编译器正在执行的操作的消息
-deprecation 输出使用已过时的 API 的源位置
-classpath <路径> 指定查找用户类文件和注释处理程序的位置
-cp <路径> 指定查找用户类文件和注释处理程序的位置
-sourcepath <路径> 指定查找输入源文件的位置
-bootclasspath <路径> 指定引导类文件的位置
-extdirs <目录> 指定安装扩展的位置
-endorseddirs <目录> 指定签名的标准路径的位置
-processor <class1>[,<class2>,<class3>,...] 要运行的注释处理程序的名称; 绕过默认
的搜索进程
-processorpath <路径> 指定查找注释处理程序的位置
-d <目录> 指定放置生成的类文件的位置
-s <目录> 指定放置生成的源文件的位置
-implicit:<none,class> 指定是否为隐式引用文件生成类文件
```

4.3.2 安装 NDK

ndk 下载列表地址: <http://www.cnblogs.com/yaotong/archive/2011/01/25/1943615.html>, 从该网址选择跟自己电脑系统匹配的版本, 下载, 然后放到 D 盘, 最好是根目录, 假如不是根目录, 文件夹命名时候不能有空格和中文, 然后解压, 后面会用到该目录。

4.3.3 安装 android-studio

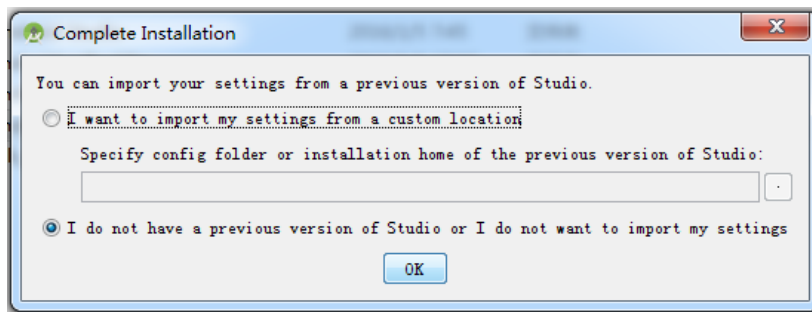
Android studio 下载地址 <http://www.android-studio.org/> 或者 <https://dl.google.com/dl/android/studio/install/1.4.0.10/android-studio-bundle-141.2288178-windows.exe>, 或者安装已下载的文件 android-studio-bundle-141.2288178-windows.exe, 安装过程与普通软件类似, 最好选择一个磁盘比较大的地方, 软件目录和 sdk 目录最好放在同一个路径下, 方便以后使用, 改一下路径, 然后一直下一步, 最后 Finish。



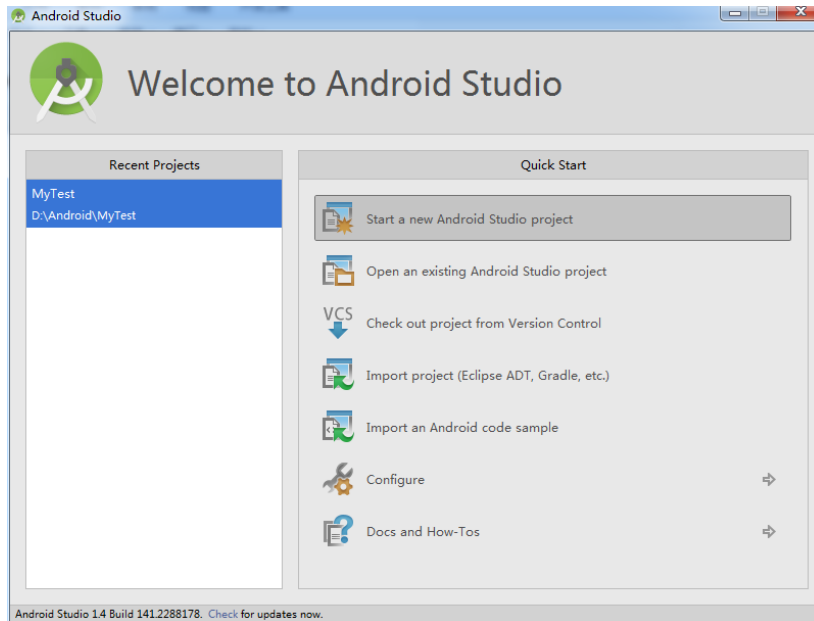
4.4 安卓 NDK+JNI 实例

4.4.1 新建工程

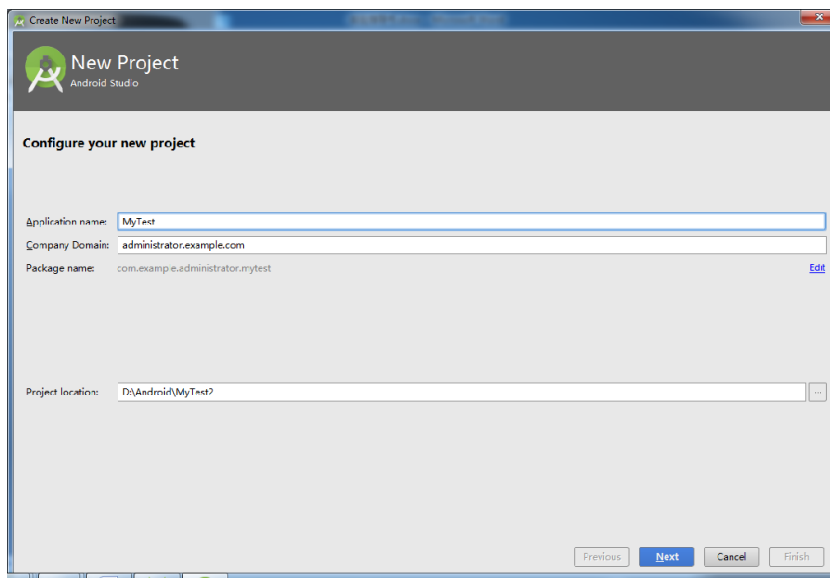
首先打开刚才安装好的 Android studio 软件，首次运行会有这样的提示，直接点击 OK。



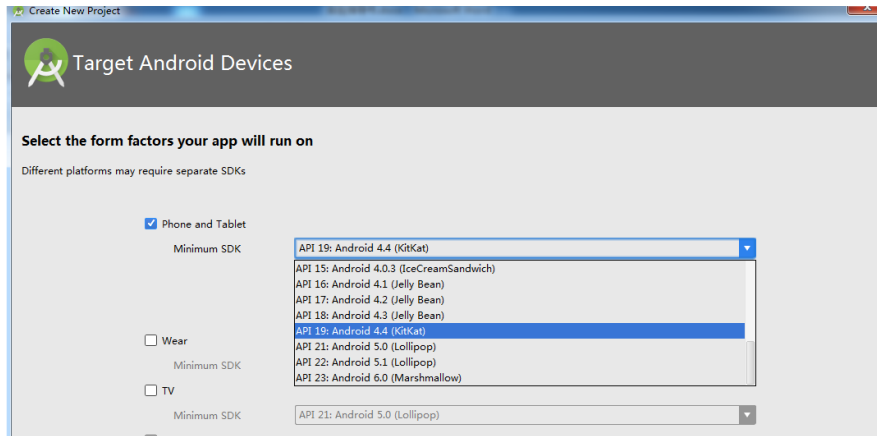
新建工程，选择 Start a new Android Studio project



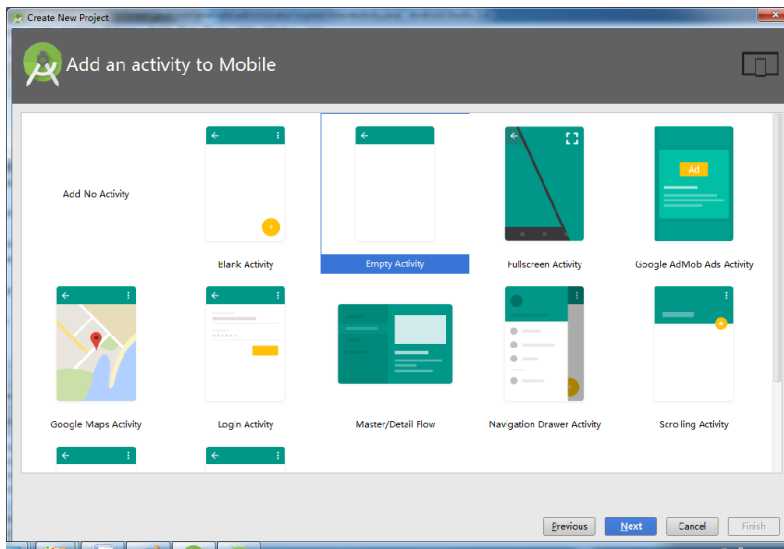
输入工程名称，点击 Next



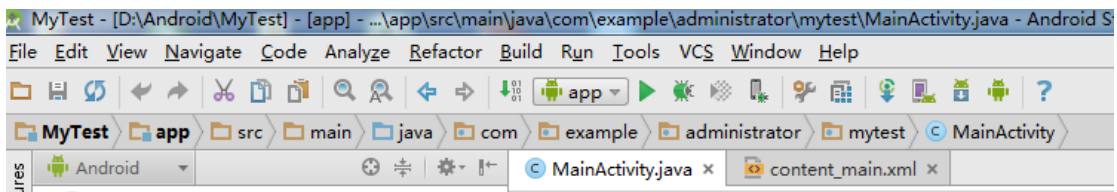
然后选择安卓版本，选择版本的时候要注意，先看一下自己的安卓手机的安卓版本，这里选择的版本一定不能高于手机的安卓版本，要不然你的程序将无法在你的安卓手机上运行，这里选择 Android4.4，然后下一步



选择框架，这里选择 Empty Activity ，空的框架，然后下一步，finish。



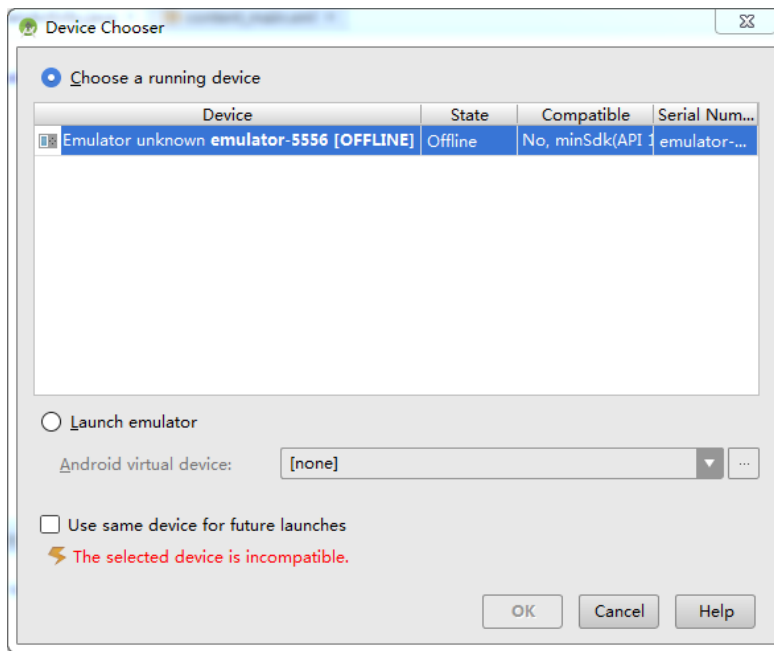
首次运行该软件，电脑需要联网，等待配置完成



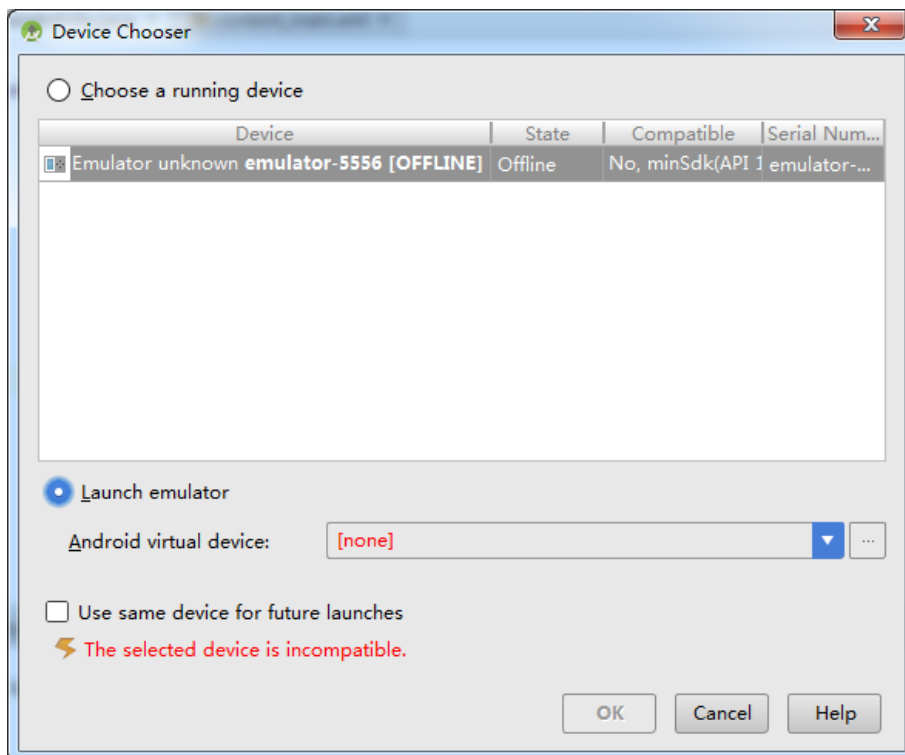
然后点击绿色三角符号，运行程序



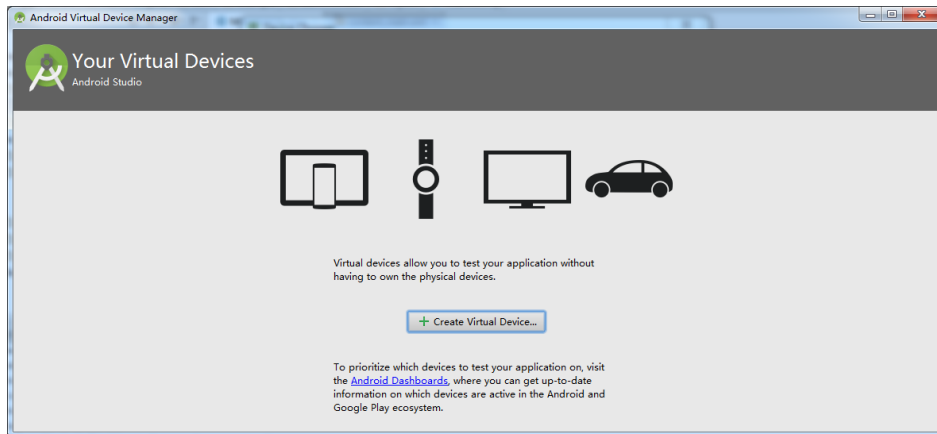
假如此时电脑上已经通过数据线连接了安卓手机，下面的设备列表里就会显示你的手机，选择你的手机，点击 OK 即可



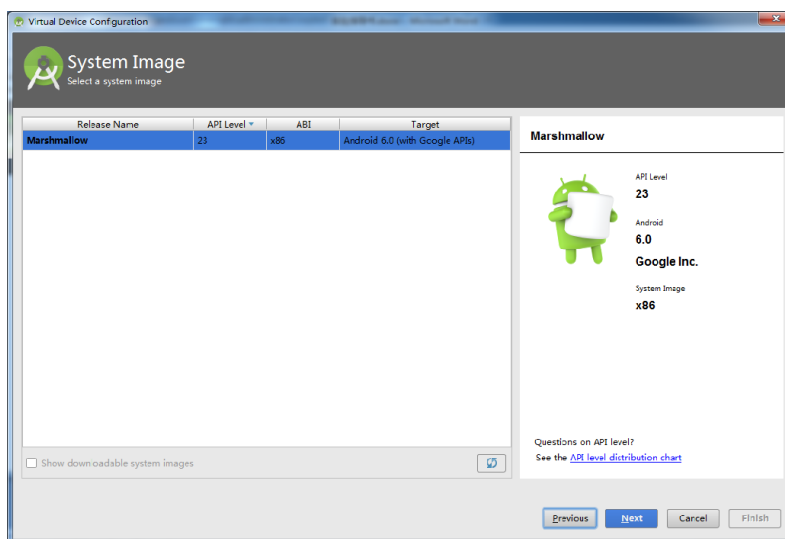
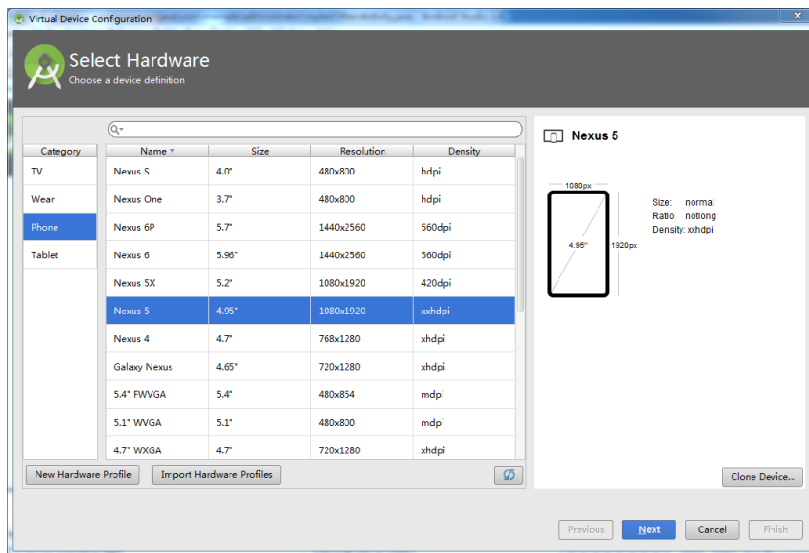
假如电脑上没有连接手机，就选中下面的 Launch emulator，点击

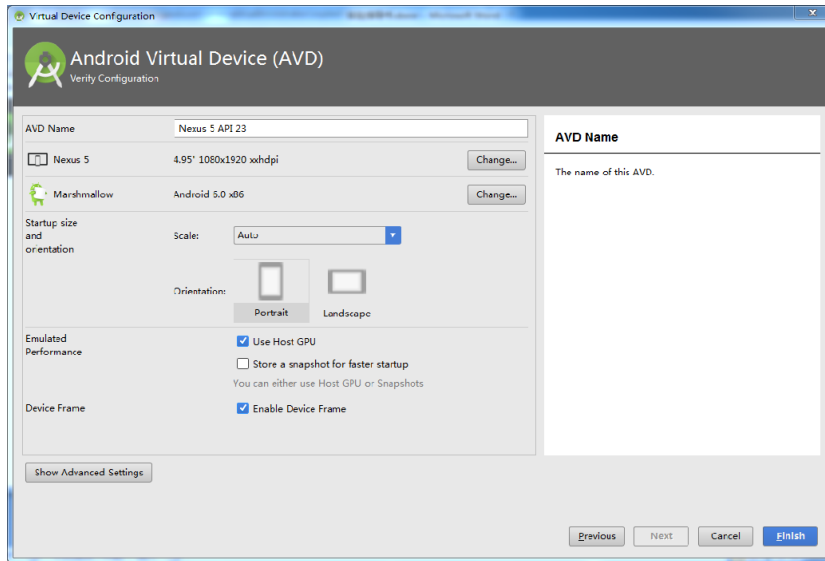


点击 Create Virtual Device

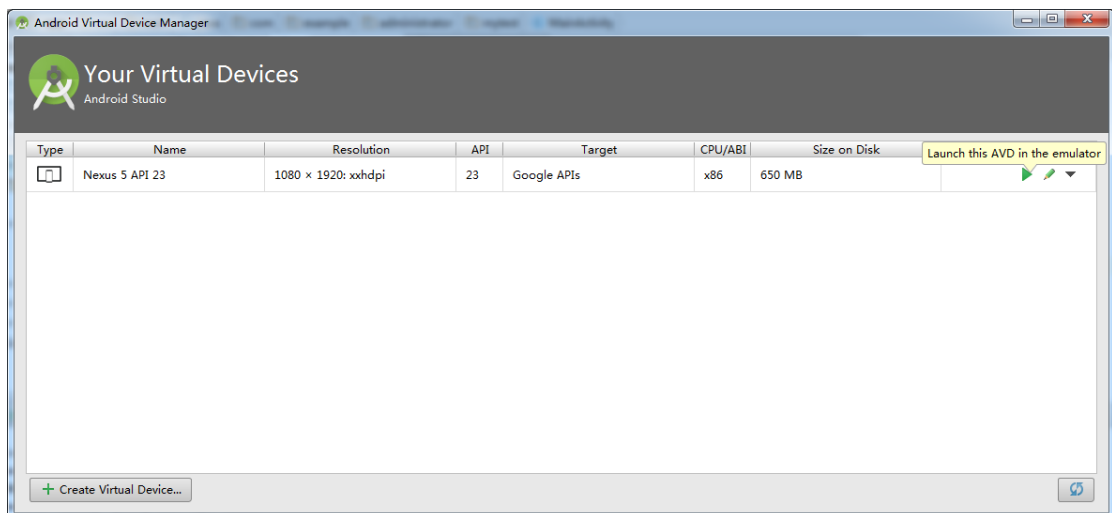


选择虚拟设备尺寸，点击 Next, Next, Finish



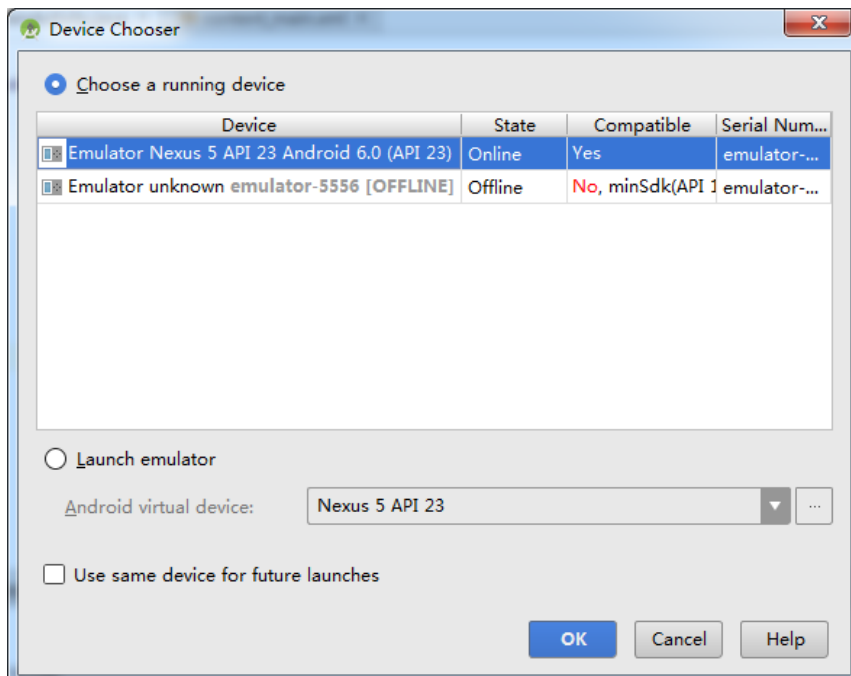


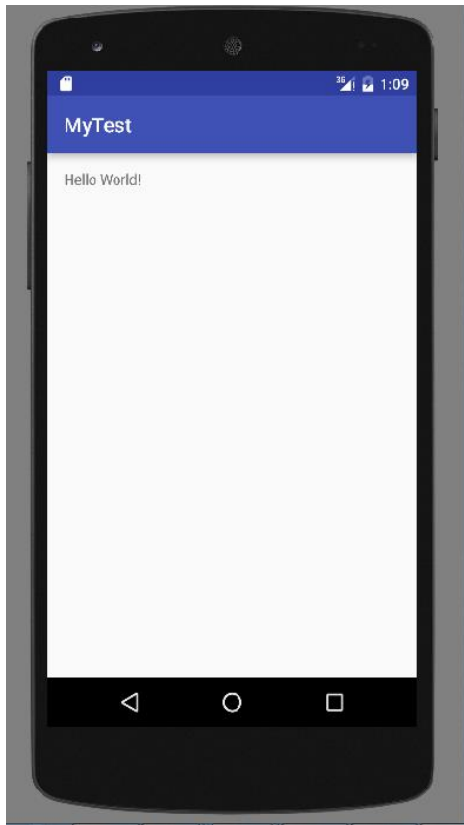
然后点击下面这个绿色三角标志，等待一两分钟，会有一个虚拟的安卓手机显示在桌面上





此时关闭所有无用窗口，再次点击  中的绿色三角符号 (运行按钮)，弹出下面的对话框，选择正在运行的虚拟设备，第一个就是桌面上的虚拟手机，状态是 Online，点击 OK，稍等几秒钟，桌面的安卓手机就会显示出运行界面。

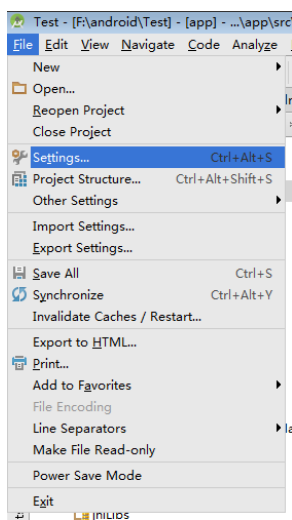




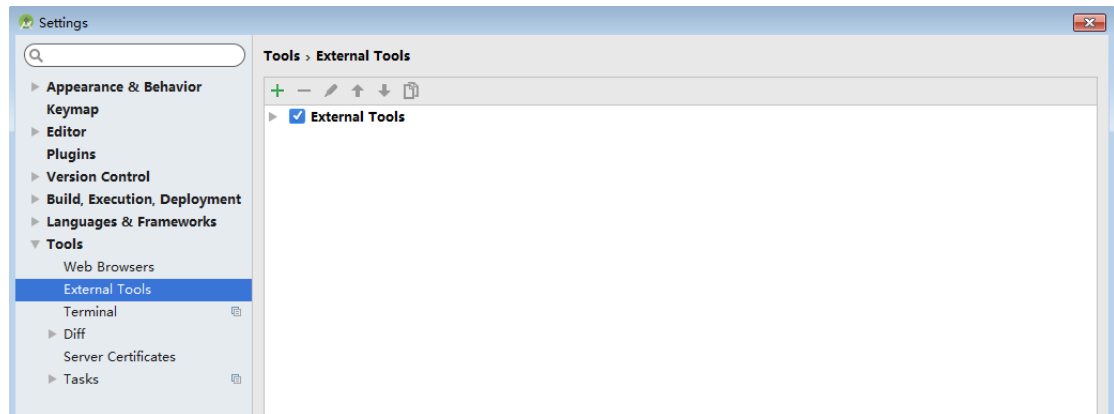
这就是最简单的安卓程序，只是对前台进行了编译，类似于 C 语言的 hello world，因为该实验需要用前台调用后台，后面就开始讲解如何用 java 通过 NDK+JNI 调用 C。

4.4.2 配置 NDK

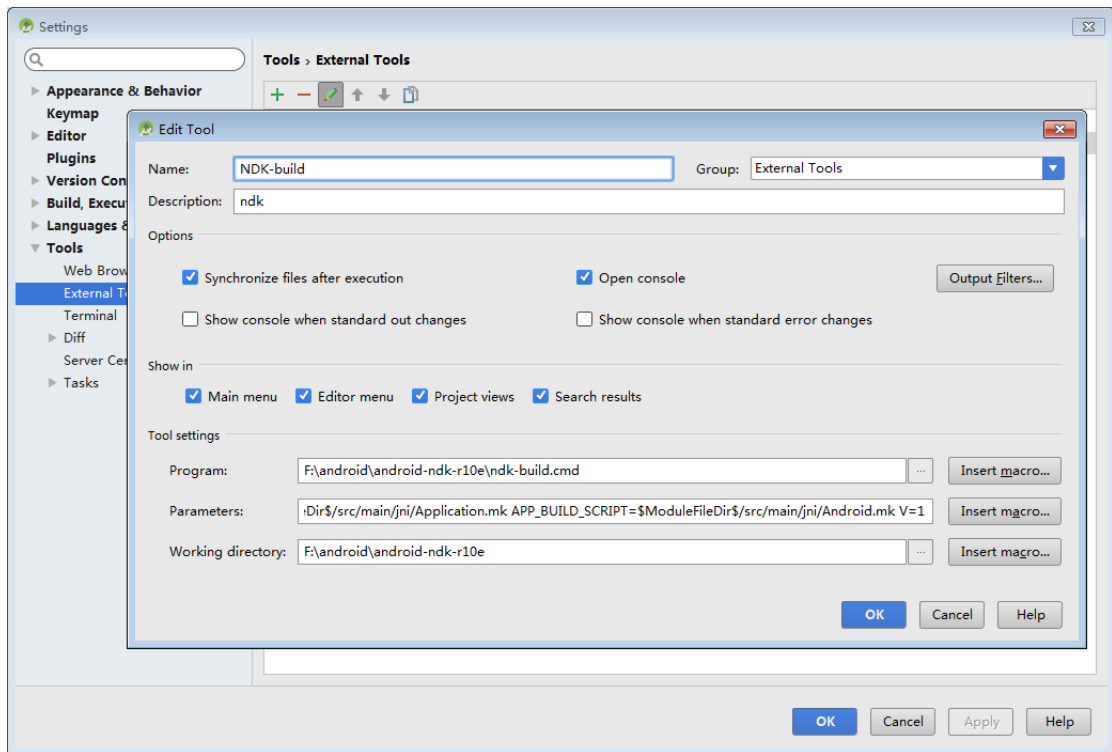
点击 file → settings



打开 tools -> External tools, 假如找不到 External tools, 就在上面的搜索框中输入 External tools, 进行搜索

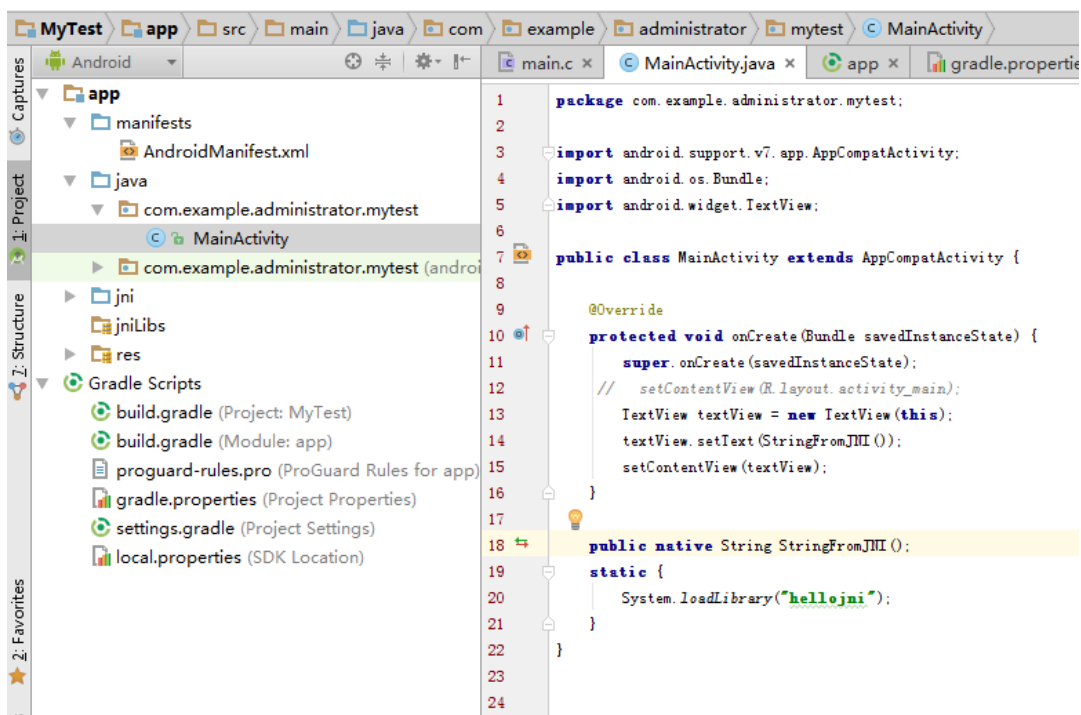


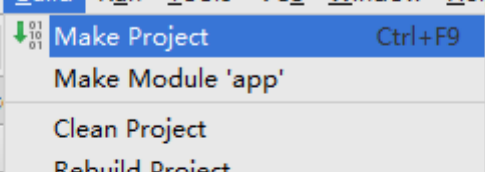
点击左上角的绿色“+”号, 按照下图内容输入 Name, Group, Description, 然后下面三个编辑框输入 Program: 打开自己的 NDK 解压路径, 然后选择 ndk-build.cmd 文件, 然后 Parameters: 这里面是固定内容, 把下面这段代码粘贴进去,
NDK_PROJECT_PATH=\$ModuleFileDir\$/build/intermediates/ndk
NDK_LIBS_OUT=\$ModuleFileDir\$/src/main/jniLibs
NDK_APPLICATION_MK=\$ModuleFileDir\$/src/main/jni/Application.mk
APP_BUILD_SCRIPT=\$ModuleFileDir\$/src/main/jni/Android.mk V=1, 然后 Working directory, 假如第一个 Program 编辑框没有填写错误, 这里会自动填写, 也就是你自己的 NDK 的解压路径, 然后点击 OK, apply, OK, 关闭对话框, 到这里 NDK 就配置完成了。



4.4.3 配置 JNI

我们要用 java 代码调用 C 代码，就要用到 JNI，首先在 MainActivity.java 中的代码做一些修改，添加 JNI 调用接口，代码如下图所示：





factor **Build** Run Tools VCS Window Help

- Make Project Ctrl+F9
- Make Module 'app'
- Clean Project
- Rebuild Project
- Edit Build Types...
- Edit Flavors...
- Edit Libraries and Dependencies...
- Select Build Variant...
- Generate Signed APK...
- Deploy Module to App Engine...

Windows File Explorer window showing the contents of a directory. The address bar shows the path: classes > debug > com > example > administrator > mytest. The search bar contains "搜索 mytest".

The left sidebar shows the "库" (Libraries) section with "本地磁盘 (C:)" (Local Disk C:) selected.

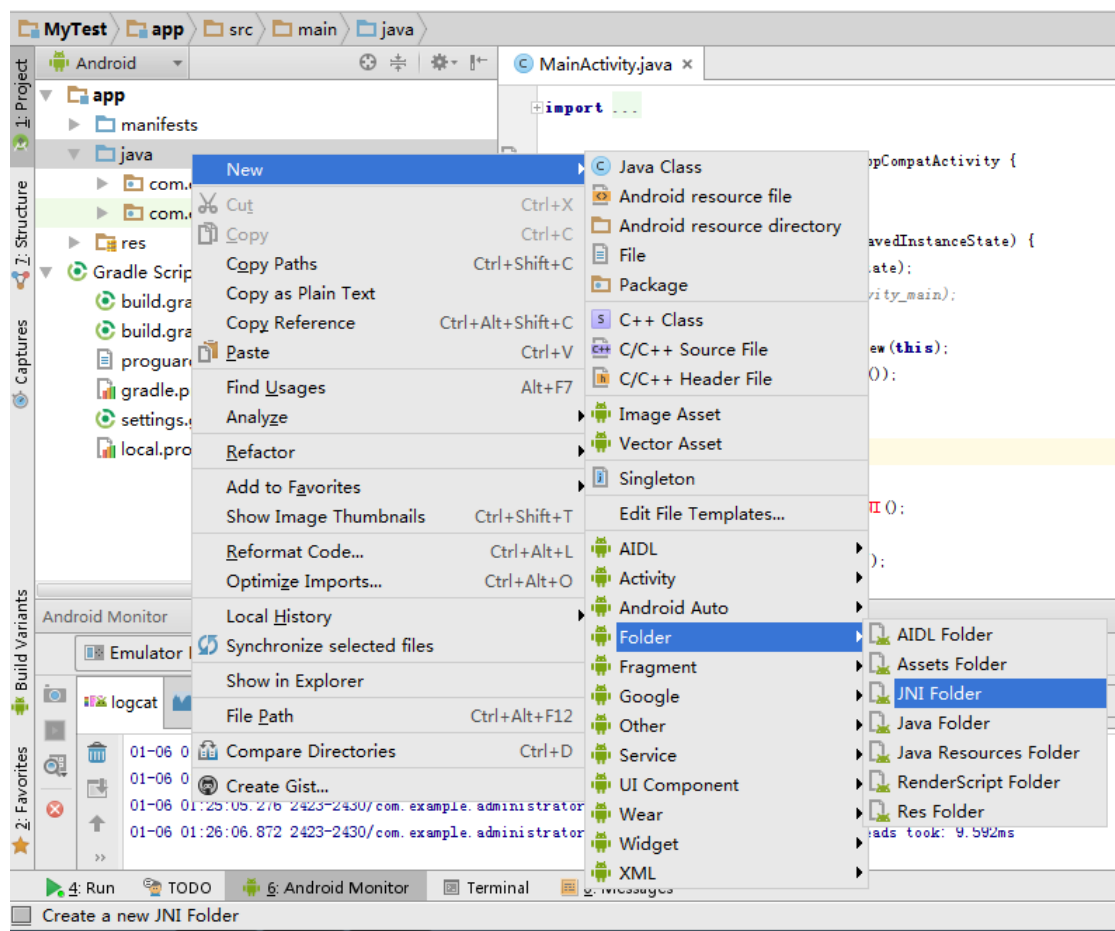
The main pane displays a list of files and folders. The file "MainActivity.class" is selected, and a tooltip is visible showing its details:

- 名称: MainActivity.class
- 修改日期: 2016/1/6 9:55
- 类型: CLASS 文件
- 大小: 971 字节

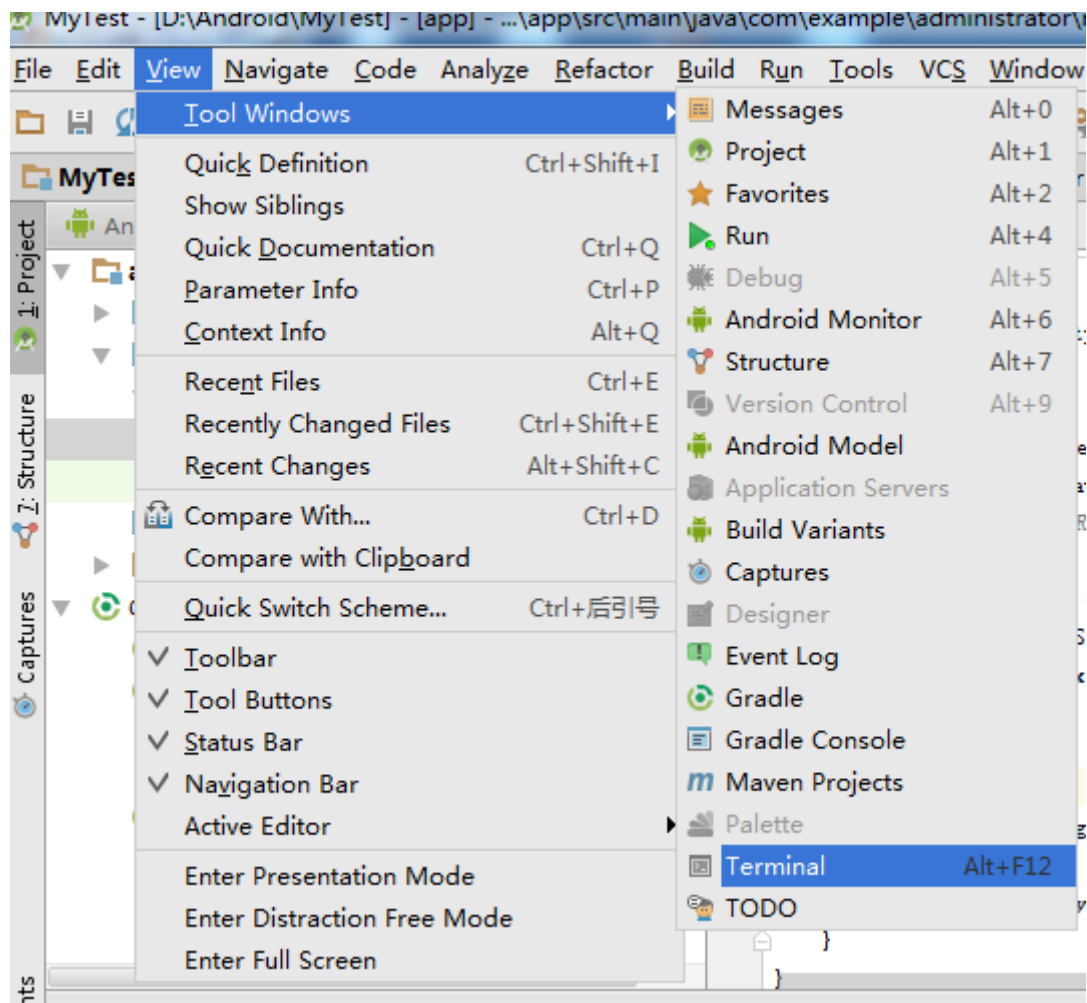
The status bar at the bottom shows:

- MainActivity.class 修改日期: 2016/1/6 9:55
- CLASS 文件
- 大小: 971 字节

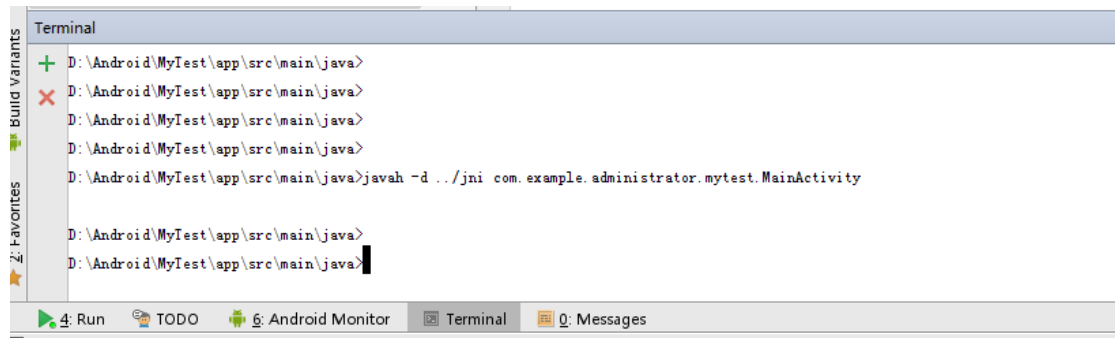
24



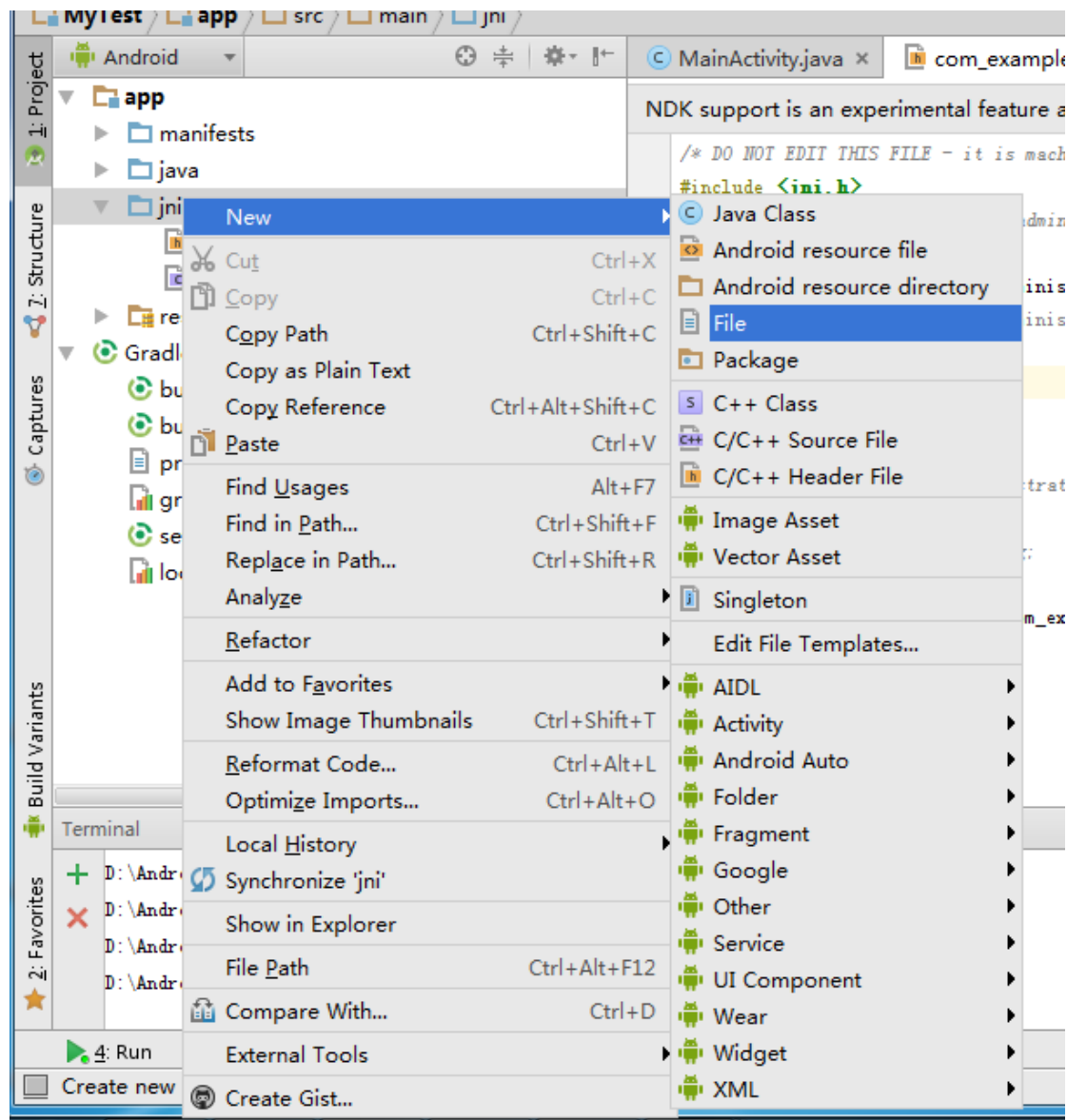
接下来开始生成 JNI 头文件，选择 View → Tool Windows → Terminal ，打开终端，在窗口的最下面显示。

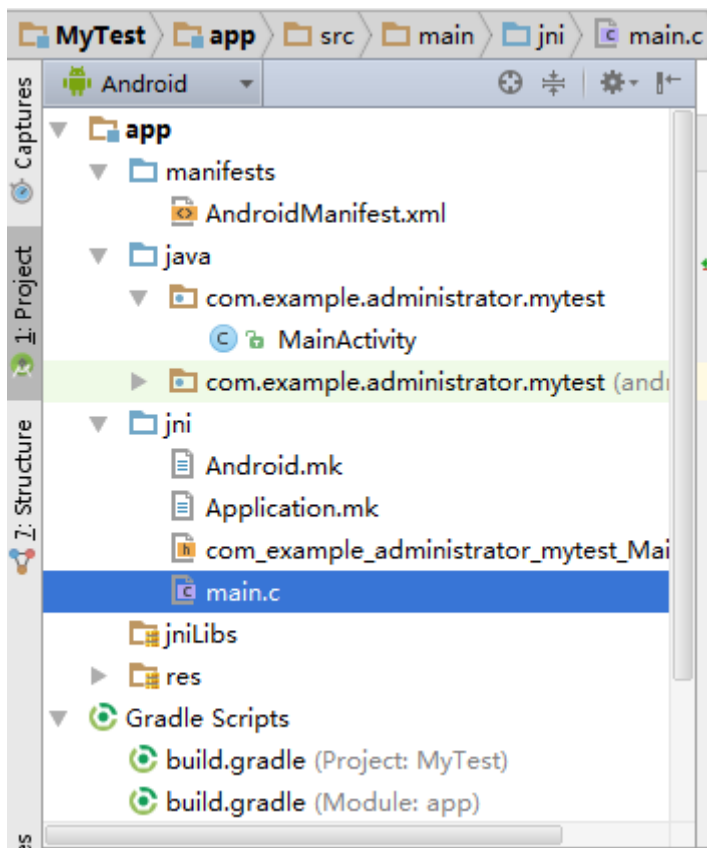


默认是在工程的目录下，我们要进到 `app\src\main\java` 路径下，终端内输入：`cd app\src\main\java`，回车，然后输入 `javah -d ../jni com.example.administrator.mytest.MainActivity`，注意 `/jni` 前面有两个点，然后回车，重新查看 `jni` 目录，发现会多一个文件名为 `com_example_administrator_mytest_MainActivity.h` 的头文件，这就是我们要使用的 C 代码的头文件。

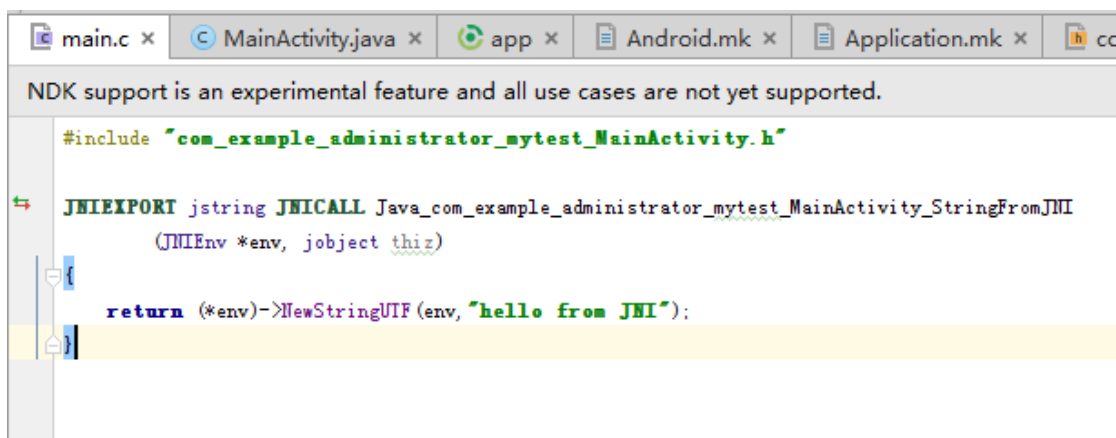


头文件已经有了，接下来该写 C 源文件了，选中 jni 文件夹，右键，new → File，文件名可以根据自己爱好输入，这里输入 main.c，同时可以新建另外两个文件 Android.mk 和 Application.mk，后面要用到。



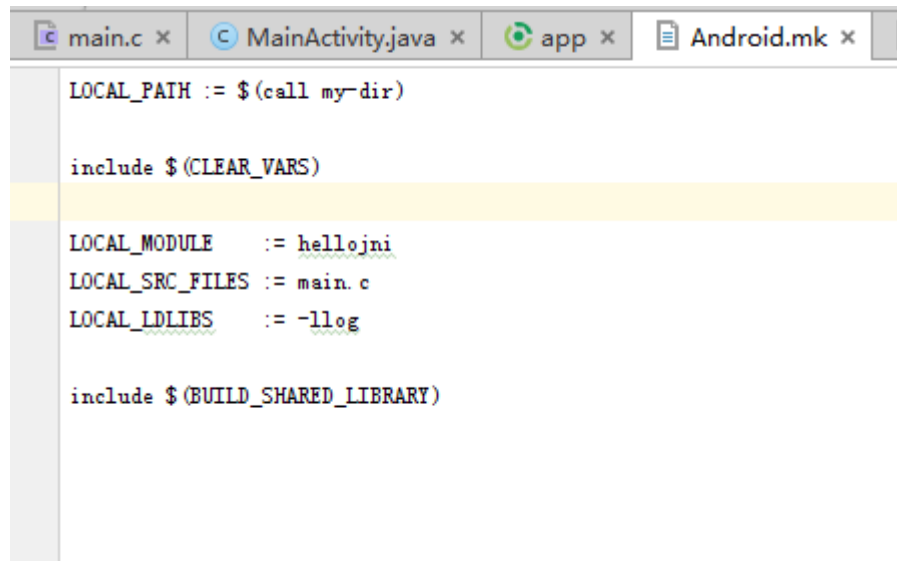


接下来编写 main.c 代码,把 com_example_administrator_mytest_MainActivity.h 文件打开,函数的声明复制到 main.c 里,把参数改一下,同时在 main.c 里包含这个头文件,下面是 main.c 的代码,因为这仅仅是个例子,所以里面代码比较简单,后面开发代码复杂的话,可以先在 linux 环境下进行编译,然后移植到 windows 下, C 文件中只需要把 java 中调用的这个函数接口按照 JNI 的规则命名,其他函数不用修改。



接下来修改刚才新建的 Android.mk 文件和 Application.mk 文件, Android.mk 文件中,

LOCAL_MODULE 代表的是加载的库文件，和 MainActivity.java 中 System.loadLibrary 中的文件名保持一致，LOCAL_SRC_FILES 是 C 代码源文件，文件名不能写错。



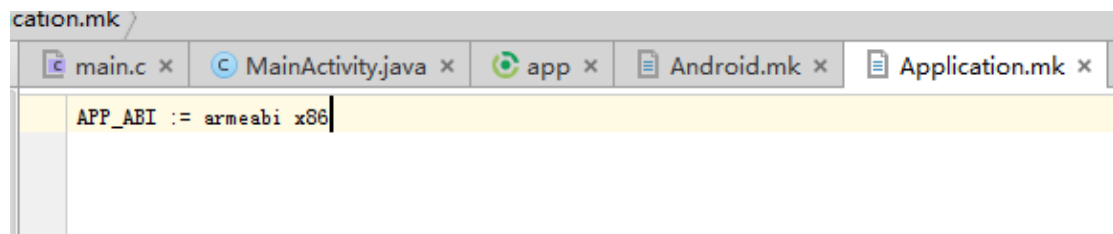
```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := hellojni
LOCAL_SRC_FILES := main.c
LOCAL_LDLIBS := -llog

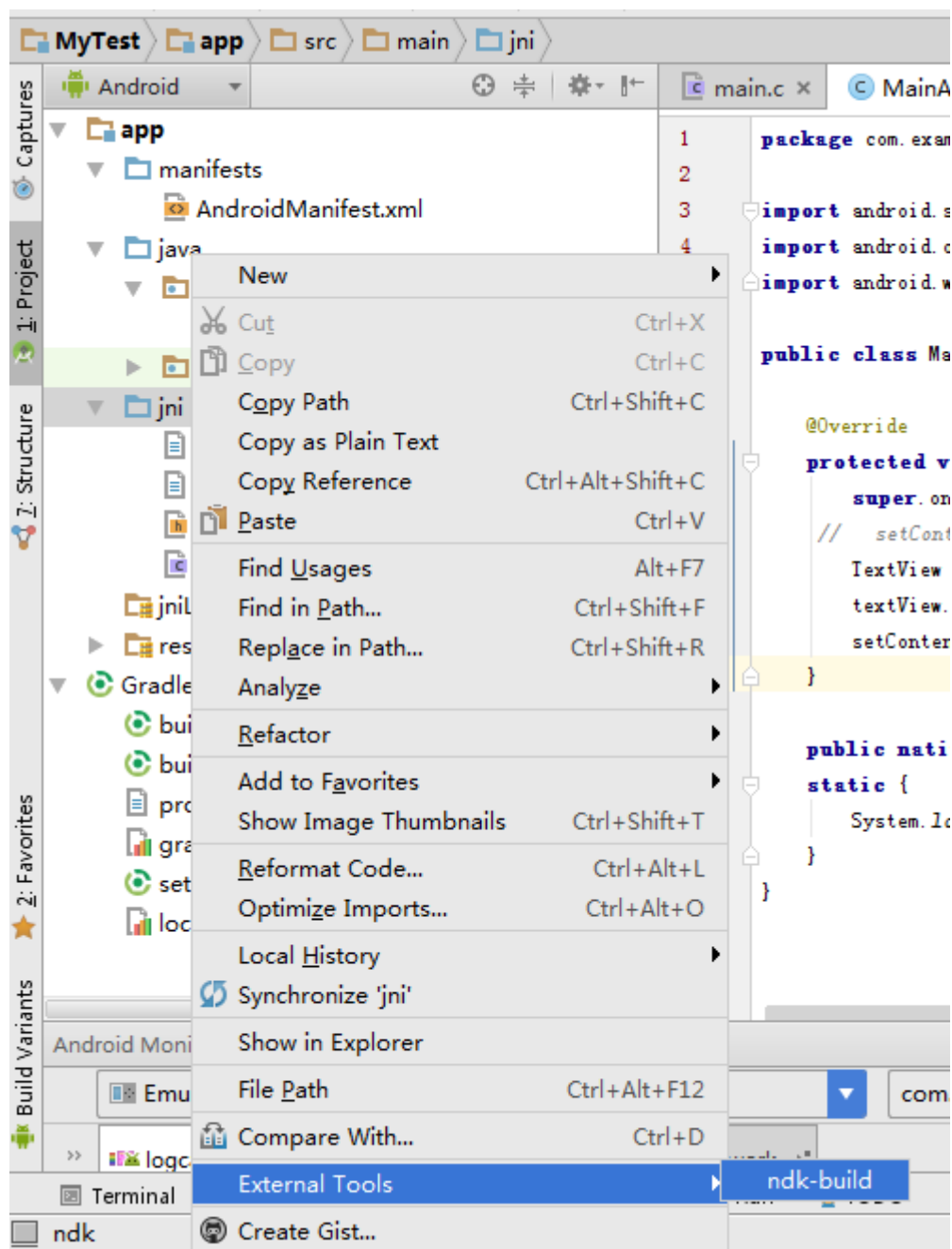
include $(BUILD_SHARED_LIBRARY)
```

Application.mk 文件中 APP_ABI 代表的是运行平台，因为刚才生成的模拟器中是 x86，所以这里就是 armeabi x86，假如电脑连接了安卓手机，可以先试一下是否能运行，假如 armeabi x86 不能运行，就把它换成 armeabi-v7a 试一下。



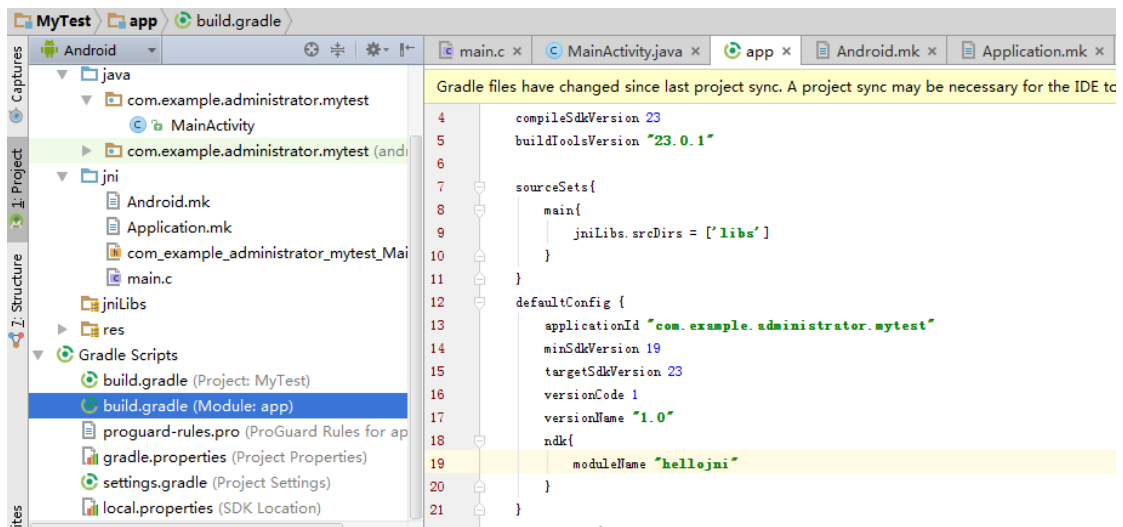
```
APP_ABI := armeabi x86
```

接下来可以生成静态库了，选中 jni 文件夹，右键，External Tools → ndk-build，假如代码没有错误的话，最后会提示出这句话 Process finished with exit code 0。有问题的话，就自己查找是不是哪里代码写错了，当 C 源文件中的代码做出修改，就需要重新 ndk-build 一下，生成最新的.so 静态库。

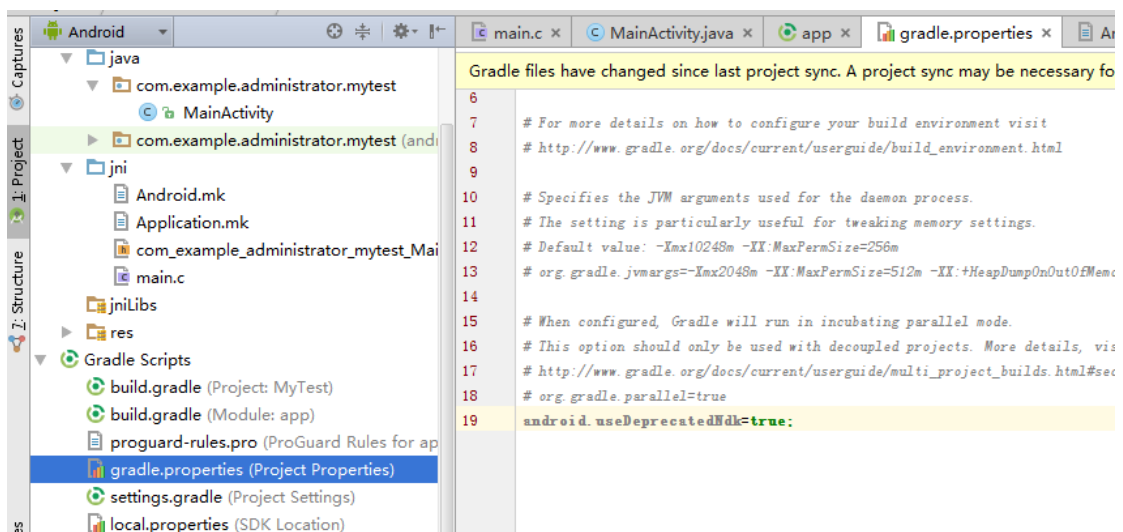


4.4.4 修改配置文件

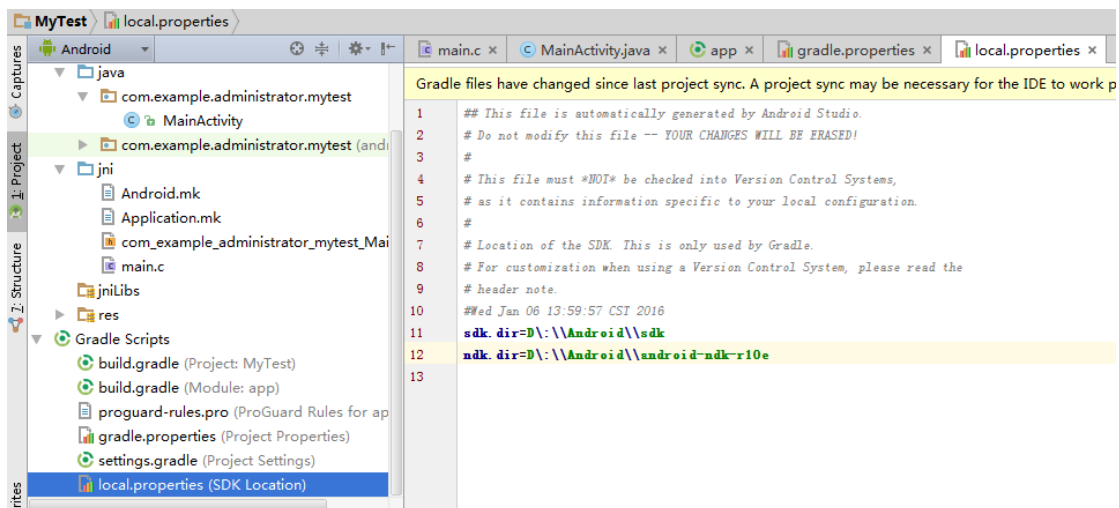
需要修改的配置文件有三个，首先修改 `build.gradle(Module:app)`，有两个 `build.gradle` 文件，注意不要选错，在该文件中需要添加两段代码，分别是第 7 到 11 行代码和 18 到 20 行代码。



然后修改 `gradle.properties` 文件，在该文件的最后一行，添加一句代码 `android.useDeprecatedNdk=true`。

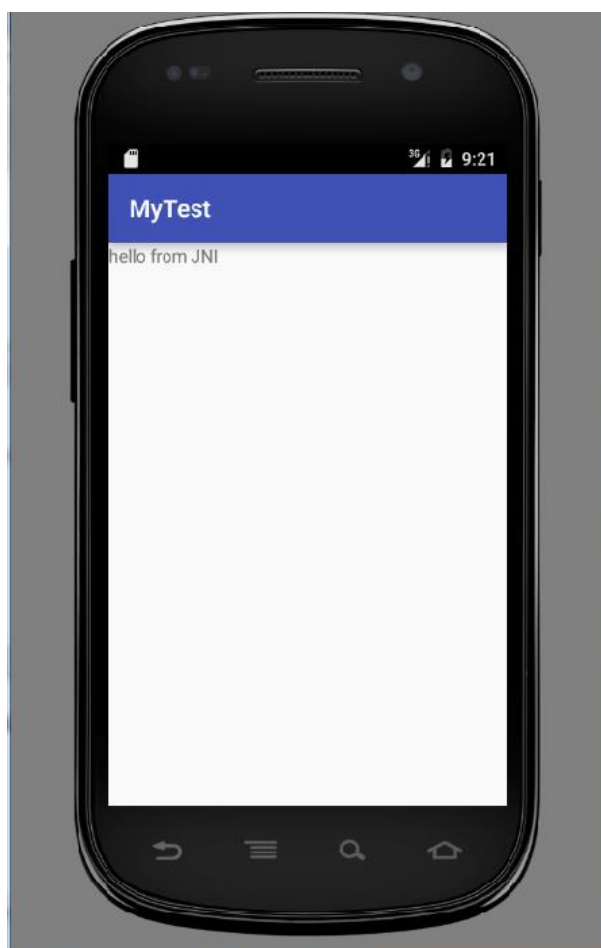


最后一个配置文件，在 `local.properties` 中，把自己的 `ndk` 解压目录添加进去，注意一定要仿照上面 `sdk` 的规则来写，否则编译出错。



4.4.5 运行程序

点击运行按钮，选择正在运行的设备，模拟器或者自己的安卓手机，点击确定，稍等几秒钟查看自己的模拟器，显示 hello from JNI ，这句话就是 C 代码中传过来的，java 成功调用 C 代码。



5.附录

5.1 如何启用安卓 VPN 服务

Android 从 4.0 开始 (API LEVEL 15)，自己带了一个帮助在设备上建立 VPN 连接的解决方案，且不需要 root 权限，本文将对其做一个简单的介绍。

一、基本原理

在介绍如何使用这些新增的 API 之前，先来说说其基本的原理。

Android 设备上,如果已经使用了 VpnService 框架,建立起了一条从设备到远端的 VPN 链接,那么数据包在设备上大致经历了如下四个过程的转换:

1) 应用程序使用 `socket`, 将相应的数据包发送到真实的网络设备上。一般移动设备只有无线网卡, 因此是发送到真实的 WiFi 设备上;

2) Android 系统通过 `iptables`, 使用 NAT, 将所有的数据包转发到 TUN 虚拟网络设备上去, 端口是 `tun0`;

3) VPN 程序通过打开 `/dev/tun` 设备, 并读取该设备上的数据, 可以获得所有转发到 TUN 虚拟网络设备上的 IP 包。因为设备上的所有 IP 包都会被 NAT 转成原地址是 `tun0` 端口发送的, 所以也就是说你的 VPN 程序可以获得进出该设备的几乎所有的数据 (也有例外, 不是全部, 比如回环数据就无法获得);

4) VPN 数据可以做一些处理, 然后将处理过后的数据包, 通过真实的网络设备发送出去。为了防止发送的数据包再被转到 TUN 虚拟网络设备上, VPN 程序所使用的 `socket` 必须先被明确绑定到真实的网络设备上去。

二、代码实现

要实现 Android 设备上的 VPN 程序, 一般需要分别实现一个继承自 `Activity` 类的带 UI 的客户程序和一个继承自 `VpnService` 类的服务程序。

申明权限

要想让你的 VPN 程序正常运行, 首先必须要在 `AndroidManifest.xml` 中显式申明使用 “`android.permission.BIND_VPN_SERVICE`” 权限, 代码如下

“`android:permission=“android.permission.BIND_VPN_SERVICE”`”。

客户程序实现

客户程序一般要首先调用 `VpnService.prepare` 函数:

[java] view plain copy

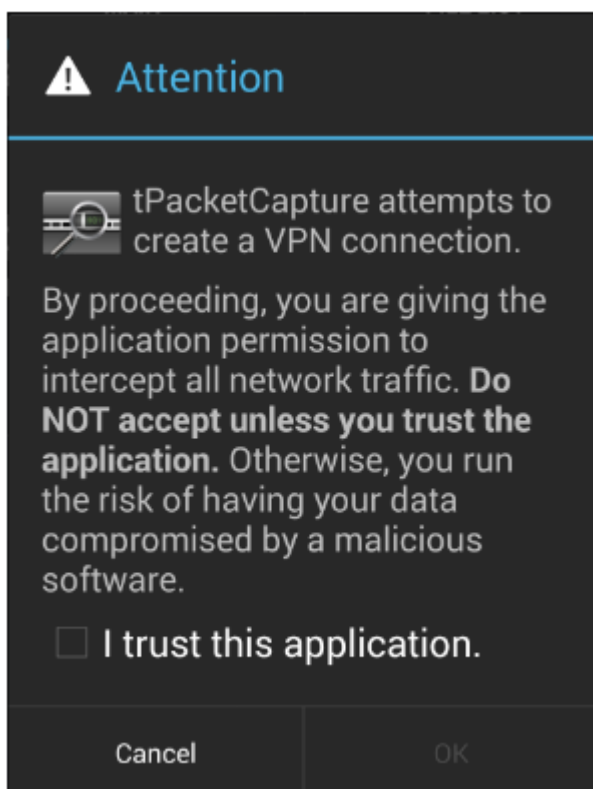
```

1. Intent intent = VpnService.prepare(this);
2. if (intent != null) {
3.     startActivityForResult(intent, 0);
4. } else {
5.     onActivityResult(0, RESULT_OK, null);

```

目前 Android 只支持一条 VPN 连接，如果新的程序想建立一条 VPN 连接，必须先中断系统中当前存在的那个 VPN 连接。

同时，由于 VPN 程序的权利实在太大了，所以在正式建立之前，还要弹出一个对话框，让用户点头确认。



VpnService.prepare 函数的目的，主要是用来检查当前系统中是不是已经存在一个 VPN 连接了，如果有了的话，是不是就是本程序创建的。

如果当前系统中没有 VPN 连接，或者存在的 VPN 连接不是本程序建立的，则 VpnService.prepare 函数会返回一个 intent。这个 intent 就是用来触发确认对话框的，程序会接着调用 startActivityForResult 将对话框弹出来等用户确认。如果用户确认了，则会关闭前面已经建立的 VPN 连接，并重置虚拟端口。该对话框返回的时候，会调用 onActivityResult 函数，并告之用户的选择。

如果当前系统中有 VPN 连接，并且这个连接就是本程序建立的，则函数会返回 null，就

不需要用户再确认了。因为用户在本程序第一次建立 VPN 连接的时候已经确认过了，就不要重复确认了，直接手动调用 `onActivityResult` 函数就行了。

在 `onActivityResult` 函数中，处理相对简单：

[java] [view plaincopy](#)

```
1.  protected void onActivityResult(int request, int result, Intent data) {
2.      if (result == RESULT_OK) {
3.          Intent intent = new Intent(this, MyVpnService.class);
4.          ...
5.          startService(intent);
6.      }
7.  }
```

如果返回结果是 OK 的，也就是用户同意建立 VPN 连接，则将你写的，继承自 `VpnService` 类的服务启动起来就行了。

当然，你也可以通过 `intent` 传递一些别的参数。

服务程序实现

服务程序必须要继承自 `android.net.VpnService` 类：

[java] [view plaincopy](#)

```
1.  public class MyVpnService extends VpnService ...
```

`VpnService` 类封装了建立 VPN 连接所必须的所有函数，后面会逐步用到。

建立链接的第一步是要用合适的参数，创建并初始化好 `tun0` 虚拟网络端口，这可以通过在 `VpnService` 类中的一个内部类 `Builder` 来做到：

[java] [view plaincopy](#)

```
1.  Builder builder = new Builder();
2.  builder.setMtu(...);
3.  builder.addAddress(...);
4.  builder.addRoute(...);
5.  builder.addDnsServer(...);
6.  builder.addSearchDomain(...);
7.  builder.setSession(...);
8.  builder.setConfigureIntent(...);
9.
10. ParcelFileDescriptor interface = builder.establish();
```

可以看到，这里使用了标准的 `Builder` 设计模式。在正式建立（`establish`）虚拟网络接口

之前，需要设置好几个参数，分别是：

1) MTU (Maximun Transmission Unit)，即表示虚拟网络端口的最大传输单元，如果发送的包长度超过这个数字，则会被分包；

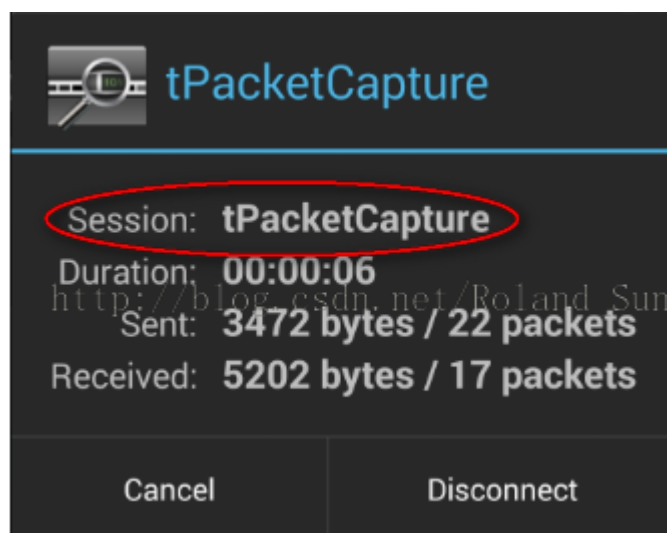
2) Address，即这个虚拟网络端口的 IP 地址；

3) Route，其实这里并不是用来修改 Android 设备上的路由表，而是用来改 iptables 的 NAT 表，只有匹配上的 IP 包，才会被转发到虚拟端口上去。如果是 0.0.0.0/0 的话，则会将所有的 IP 包都通过 NAT 转发到虚拟端口上去；

4) DNS Server，就是该端口的 DNS 服务器地址；

5) Search Domain，就是添加 DNS 域名的自动补齐。DNS 服务器必须通过全域名进行搜索，但每次查找都输入全域名太麻烦了，可以通过配置域名的自动补齐规则予以简化；

6) Session，就是你要建立的 VPN 连接的名字，它将会在系统管理的与 VPN 连接相关的通知栏和对话框中显示出来；



7) Configure Intent，这个 intent 指向一个配置页面，用来配置 VPN 链接。它不是必须的，如果没设置的话，则系统弹出的 VPN 相关对话框中不会出现配置按钮。

最后调用 Builder.establish 函数，如果一切正常的话，tun0 虚拟网络接口就建立完成了。并且，同时还会通过 iptables 命令，修改 NAT 表，将所有数据转发到 tun0 接口上。

此时通过 getFD()函数就能把 tun0 的文件描述符拿出来，虚接口的文件描述符可以在 linux 下用 fcntl.h 中的 read 和 write 函数对虚接口的文件描述符进行读写操作。

这其实基本上就是这个所谓的 VpnService 的全部了，是不是觉得有点奇怪，半点没涉及到建立 VPN 链接的事情。这个框架其实只是可以让某个应用程序可以方便的截获设备上所有发送出去和接收到的数据包，仅此而已。能获得这些数据包，当然可以非常方便的将它们

封装起来，和远端 VPN 服务器建立 VPN 链接，但是这一块 VpnService 框架并没有涉及，留给你的应用程序自己解决。

最后，简单总结一下：

1) VPN 连接对于应用程序来说是完全透明的，应用程序完全感知不到 VPN 的存在，也不需要为支持 VPN 做任何更改；

2) 并不需要获得 Android 设备的 root 权限就可以建立 VPN 连接。你所需要的只是在你应用程序内的 AndroidManifest.xml 文件中申明需要一个叫做“android.permission.BIND_VPN_SERVICE”的特殊权限；

3) 在正式建立 VPN 链接之前，Android 系统会弹出一个对话框，需要用户明确的同意；

4) 一旦建立起了 VPN 连接，Android 设备上所有发送出去的 IP 包，都会被转发到虚拟网卡的网络接口上去（主要是通过 iptables，使用 NAT 转发过来的）；

5) VPN 程序可以通过读取这个接口上的数据，来获得所有设备上发送出去的 IP 包；同时，可以通过写入数据到这个接口上，将任何 IP 数据包插入系统的 TCP/IP 协议栈，最终送给接收的应用程序；

6) Android 系统中同一时间只允许建立一条 VPN 链接。如果有程序想建立新的 VPN 链接，在获得用户同意后，前面已有的 VPN 链接会被中断；

7) 这个框架虽然叫做 VpnService，但其实只是让程序可以获得设备上的所有 IP 数据包。通过前面的简单分析，大家应该已经感觉到了，这个所谓的 VPN 服务，的确可以方便的用来在 Android 设备上建立和远端服务器之间的 VPN 连接，但其实它也可以被用来干很多有趣的事情，比如可以用来做防火墙，也可以用来抓设备上的所有 IP 包。

想更深入的了解 VPN 的话，可以去这里看一下 http://blog.csdn.net/Roland_Sun/article/details/46337171#reply。

注：需要在建立 VPN 时，对 IPv6 的 socket 进行保护，阻止该 socket 通过 VPN，示例：`myVPNService.protect(sockfd)`；`sockfd` 是建立的 socket 文件描述符。

5.2 java 与 c 管道通信示例

Java 写管道示例：

```
File extDir = Environment.getExternalStorageDirectory(); //获取当前路径
File file = new File(extDir, "cmd_pipe");
FileOutputStream fileOutputStream = new FileOutputStream(file);
BufferedOutputStream out = new BufferedOutputStream(fileOutputStream);
out.write(arr, 0, arr.length); //arr 是存放数据的 byte 类型数组
```

```
out.flush();
out.close();
```

java 读管道示例:

```
FileInputStream fileInputStream = new FileInputStream(file);
BufferedInputStream in = new BufferedInputStream(fileInputStream);
readLen = in.read(readBuf); //读取管道
in.close();
```

C 写管道示例:

```
char fifo_buf[100] = "";
int size;
mknod(fifo_name, S_IFIFO | 0666, 0); //创建有名管道
fifo_handle = open(fifo_name, O_RDWR|O_CREAT|O_TRUNC);
size = write(fifo_handle, buf, sizeof(buf));
close(fifo_handle);
```

C 读取管道示例:

```
if ((fifo_handle = open(fifo_name, O_RDWR|O_CREAT)) < 0)
    LOGE("wlh open fifo error %d:%s\n", errno, strerror(errno));
size = read(fifo_handle, fifo_buf, 100);
if (size < 0)
    LOGE("wlh read fifo error");
else if (size > 0)
    close(fifo_handle);
```

该示例仅仅是给学生提供一个使用管道的方法, 学生可以借鉴, 不保证编译通过, 因为 java 操作文件的时候需要捕获异常。

5.3 4over6 隧道服务器地址

4over6 隧道服务器 ipv6 地址: 2402:f000:1:4417::900

和目标服务器的连接的目的端口号为 5678, 使用 TCP 连接。