1、编写静态方法（用java声明）-->编译生成class文件--->编译生成h文件---->编写C文件（用C/C++实现）---->配置NDK---->配置so库---->在Activity调用（Java调用C/C++）。

2、JNI和NDK是面试经常被问到的热点问题，在这里做一个记录。

1）阐述你对JNI的理解

JNI的全称是Java Native Interface（Java本地接口）是一层接口，是用来沟通Java代码和C/C++代码的，是Java和C/C++之间的桥梁。通过JNI，Java可以完成对外部C/C++编写的库函数的调用，相对的，外部C/C++也能调用Java中封装好的类和方法。

Java的优点是跨平台，和操作系统之间的调用由JVM完成，但是一些和操作系统相关的操作就无法完成，JNI的出现刚好弥补了这个缺陷，也完善了Java语言，将java扩展得更为强大。

JNI的应用方面：

实际中的驱动都是C/C++开发的,通过JNI,Java可以调用C开发好的驱动，从而扩展Java虚拟机的能力。另外，在高效率的数学运算、游戏的实时渲染、音视频的编码和解码等方面，一般都是用C开发的。

2）NDK的理解

NDK(Native Development Kit)是Android所提供的一个工具集合，通过NDK可以在Android更加方便地通过JNI来调用本地代码（C/C++）。NDK提供了交叉编译器，开发时只需要修改mk文件就能生成特定的CPU平台的动态库。

1. Diffie-Hellman(迪菲-赫尔曼)秘钥交换

Curve25519 是目前最高水平的 Diffie-Hellman函数，适用于广泛的场景，由Daniel J. Bernstein教授设计。由于NIST P-256的设计过程不透明，有来历不明的参数，被广泛怀疑有后门，所以设计了Curve25519，Curve25519的设计过程完全公开，没有任何来历不明的参数。

对称加密是最快速、最简单的一种加密方式，加密（encryption）与解密（decryption）用的是同样的密钥（secret key）,这种方法在密码学中叫做对称加密算法。

不对称加密算法使用两把完全不同但又是完全匹配的一对钥匙—公钥和私钥。在使用不对称加密算法加密文件时，只有使用匹配的一对公钥和私钥，才能完成对明文的加密和解密过程。加密明文时采用公钥加密，解密密文时使用私钥才能完成，而且发信方（加密者）知道收信方的公钥，只有收信方（解密者）才是唯一知道自己私钥的人。不对称加密算法的基本原理是，如果发信方想发送只有收信方才能解读的加密信息，发信方必须首先知道收信方的公钥，然后利用收信方的公钥来加密原文；收信方收到加密密文后，使用自己的私钥才能解密密文。显然，采用不对称加密算法，收发信双方在通信之前，收信方必须将自己早已随机生成的公钥送给发信方，而自己保留私钥。由于不对称算法拥有两个密钥，因而特别适用于分布式系统中的数据加密。广泛应用的不对称加密算法有RSA算法和美国国家标准局提出的DSA。以不对称加密算法为基础的加密技术应用非常广泛。

1. Base64:byte[] 数组与String之间的转换

CRC32/64：String转换为32位或64位数值

5、JNI 调用的步骤

1）声明native 方法

// 1\声明一个native 方法 该方法用于调用C代码中的函数

public native String helloJni();

2）在android项目中创建jni文件夹，并添加c文件

3）在C文件中，声明与native 方法对应的C函数

/\*\*Eclipse找到src文件夹（Studio找到java文件夹），使用Javah 包名.类名命令可以直接生成对应的C函数

\* 返回值：jstring 对应java当中的String 类

\* 方法名：Java\_包名\_类名\_方法名

\* 参数一：JNIEnv\* env java虚拟机的运行环境

\* 参数二：jobject obj 调用该方法的java对象，在下面的代码中obj是MainActivity

\*/

结构类型如：jstring Java\_com\_itcast\_hello\_jni\_MainActivity\_helloJni(JNIEnv\* env,jobject obj){}

4）添加C代码，并返回对应的值

const char\* pstr= "hello from c";

//(\*\*env).NewStringUTF(env,pstr);

return (\*env)->NewStringUTF(env,pstr);

5）添加Android.mk 配置文件:

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_MODULE := hello # 要编译的模块的名称

LOCAL\_SRC\_FILES := hello.c # 该模块对应的源文件，如果有多个文件，以空格分隔

include $(BUILD\_SHARED\_LIBRARY)

\* 上面的配置只会生成arm处理器的so库

\* 如果想要生成所有的处理器的so库，要创建Application.mk配置文件，加上下面这句话：

APP\_ABI := all

6）将C文件编译为so 动库联接库(必须以管理员权限运行命令提示符窗口)

打开jni文件夹，所在的目录，执行ndk-build命令编译。 1、 将ndk的路径添加至系统path 环境变量 2、 注意编译时，C代码中，应 #include

7）在java文件中，添加so库文件,添加静态代码块

static{

System.loadLibrary("hello");

}

8）注意

\* 静态代码块中System.loadLibrary("hello");的名字和创建的C文件名一致。

\* LOCAL\_MODULE := hello # 要编译的模块的名称，和创建的C文件名一致。

\* LOCALSRCFILES := hello.c # 该模块对应的源文件，和创建的C文件名一致；如果有多个文，以空格分隔

===============================================================================