# 交叉编译：

* 在一个平台下，编译出另一个平台能够执行的二进制的代码
* 平台：windows，mac os，linux
* 处理器：x86，arm，mips

## 交叉编译的原理：

* 源代码->编译(编译成机器可识别的汇编语言)->链接(数据存放到哪个地方)->可执行程序
* 模拟其他平台的特性，模拟其他处理器可识别的指令集

## 交叉编译的工具链

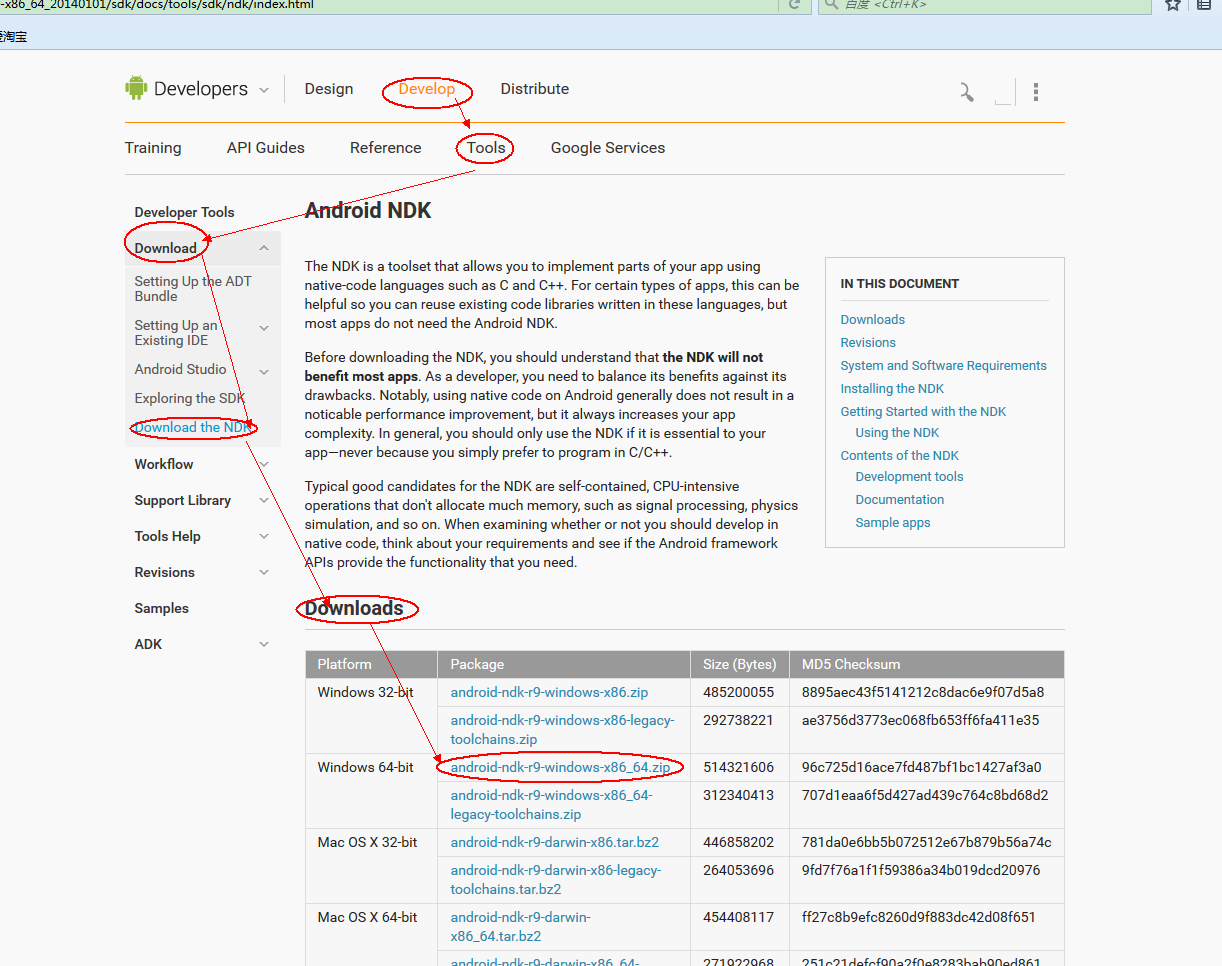
* 多个工具的集合，一个工具使用完后接着调用下一个工具

## 常见工具（做jni开发要用到的工具）

* NDK：native developement kit：本地开发工具集，谷歌提供给android工程师专门用来做jni开发的，在java上层应用程序中使用c语言的类库。就是模拟其他平台特性来编译代码的工具。它允许你使用c/c++实现应用的一部分功能
* CDT: C/C++ developement tools：c/c++开发工具包，要在eclipse中写c/c++代码，就可以用它，但不是必须的。不过用它会更方便一些（高亮显示c/c++语言关键字）
* cygwin：一个模拟器，可以在windows下运行linux指令
* 其中，NDK和CDT是我们要直接用到的工具

## 获取常见工具：

* 获取CDT：
  + 如果你使用的eclipse是从sdk里面解压出来的，他本来就带有CDT
  + 如果不是，就需要安装，CDT也是eclipse的一个插件，和安装ADT差不多
* 获取NDK:
  + 打开android帮助文档



* NDK解压后，放哪里都可以（路径不能有空格）

# NDK目录结构：

* docs:帮助文档，HOTW.html
* build/tools：linux的批处理文件，”.sh”后缀。Windows的是”.bat”后缀
* platforms：编译c代码需要使用的头文件和类库。里面有跟Android版本对应的版本。使用的是哪个android版本开发，就是用那个版本的头文件和类库
  + 里面有个jni.h，是c和java之间做翻译用的
* prebuilt：预编译使用的二进制可执行文件
* sample：jni的使用例子
* source：ndk的源码
* toolchains：工具链，里面有ndk要使用到的工具，开发环境自动使用，我们不用管。里面有各种cpu架构需要使用到的工具链
* 根目录下的ndk-build.cmd: 编译打包c代码的一个指令

# 使用JNI

* 因为c代码就是在java中使用jni被调用的，所以需要一个android项目、
* 这个项目只能在arm架构的手机上运行

## 实现本地方法

* Andorid中，使用到jni的有，音频播放MeidaPlayer
  + 音频视频都需要操作硬件播放，所以它使用jni调用c代码进行音视频播放
* 在android项目中，写一个要被我们调用的c代码
* 在项目根目录下创建jni文件夹
* 在jni文件中创建一个c文件
* 在java代码中，创建一个本地方法helloFromC，这个本地方法没有方法体。这个方法由c语言来实现它，这个方法一旦调用，调用的就是c语言

|  |
| --- |
| //实现一个本地方法，这个方法被c实现，一旦调用，就会调用c的代码  **public** **native** String helloFromC(); |

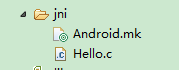
* 在c文件中定义函数，实现这个本地方法

|  |
| --- |
| **#include** <stdio.h>  **#include** <stdlib.h>  **#include** <jni.h>  /\*  \* 要实现的方法返回什么类型，就是什么类型  \* > 这里，要是实现的方法是String类型  \* > 在jni.h中明确定义了，c语言中，使用jstring来表示java中的String  \*  \* 要实现的方法的方法名：Java\_包名（点要变成\_）\_类名\_方法名  \*  \* 参数：这里的参数不是传过来的参数，不过还是需要有两个参数  \* > env: 运行环境的指针（这是个一级指针），就是java虚拟机所在的内存地址。把这个一级指针，传递个env这个二级指针（存放了运行环境的指针）  \* JNIEnv在jni.h中定义了typedef const struct JNINativeInterface\* JNIEnv;  \* JNIEnv是一个结构体的指针，env是一个二级指针  \*  \* >obj:这个函数是哪个对象调用的，就把哪个对象传进来  \* java的对象用jobject表示  \*/  jstring Java\_com\_example\_ **helloword\_jni\_MainActivity\_helloFromC**(JNIEnv\* env, jobject obj) {  //c语言的字符串  **char**\* cstr = "hello from c";  /\*  \* 把c语言的字符串，转换成java语言的字符串  \* jstring (\*NewStringUTF)(JNIEnv\*, const char\*);  \* 这是一个函数指针，在jni.h中的JNINativeInterface（运行环境）中，有大量的这种函数指针，是Google提供给开发者使用的  \* 具体这个指针所指的函数在哪里，就不用管了。直接调用函数指针，使用函数就可以了  \*/  //有两种写法，调用这个函数指针  //拿到运行环境的结构体（JNINativeInterface），调用NewStringUTF函数  //jstring jstr = (\*(\*env)).NewStringUTF(env, cstr);  //jni中，最常用的写法是，拿到运行环境结构体的指针，调用NewStringUTF  jstring jstr = (\*env)->NewStringUTF(env, cstr);  //返回一个java的字符串  **return** jstr;  } |

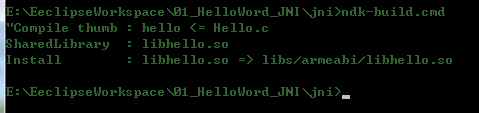
## 使用ndk-build.cmd，编译打包c代码

* + - 把NDK的路径，配置到环境变量
    - 进入到c代码所在的路径，使用命令提示符编译打包c代码
    - 编译之前，要定义android.mk文件（在ndk帮助文档，ANDROID-MK.html中有说明）
    - 在android项目的jni文件夹中，定义Android.mk文件
    - 内容：

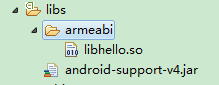
|  |
| --- |
| LOCAL\_PATH := $(call my-dir)  include $(CLEAR\_VARS)  LOCAL\_MODULE := hello-jni  LOCAL\_SRC\_FILES := Hello.c  include $(BUILD\_SHARED\_LIBRARY) |



* 编译



* 编译完成后，lib下会有新文件



* .so文件就是c编译打包后的类库，现在就可以调用实现本地方法的c代码了

## 调用本地方法

* 调用本地方法之前，要在静态代码块里，导入类库

# 本地方法实现加法

# 使其他处理器架构也能运行

* 因为ndk打包的类库，默认只打包arm架构的类库，所以其他架构不能使用
* 在Application.mk中，进行配置，就能打包其他架构能够使用的类库(NDK帮助文档CPU-X86.html有说明)
* Application.mk也是在jni目录下创建，文件里面直接这样写就行

|  |
| --- |
| APP\_ABI := armeabi armeabi-v7a x86 |

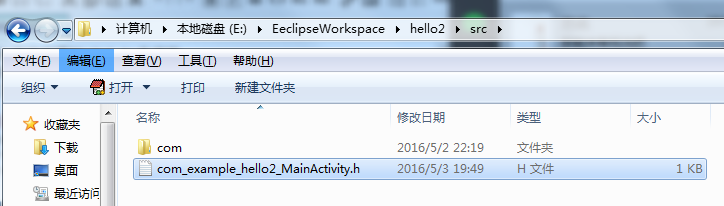
|  |
| --- |
| APP\_ABI := all |

# 常见错误

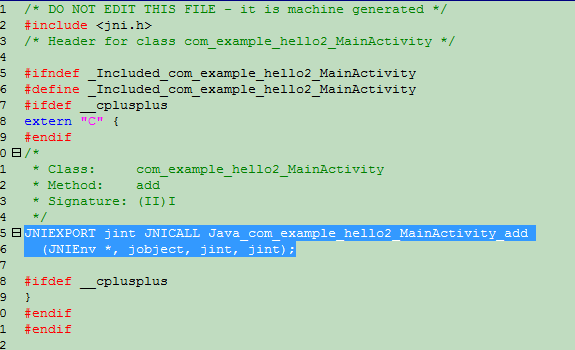
* findLibrary returned null
  + CPU平台不匹配
  + 加载类库时，写错类库名字
* 本地方法找不到
  + 忘记加载类库
  + c代码中方法名写错了

# javah

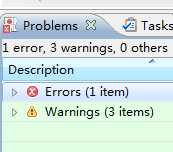
* 自动生成本地方法，要在c中实现时的方法名
* jdk1.7：在src目录下执行javah 包名.类名
* jdk1.6：在bin/classes目录下执行
* 执行后，会在src目录下生成一个文件



* 里面有我们要在c里实现本地方法时的方法名



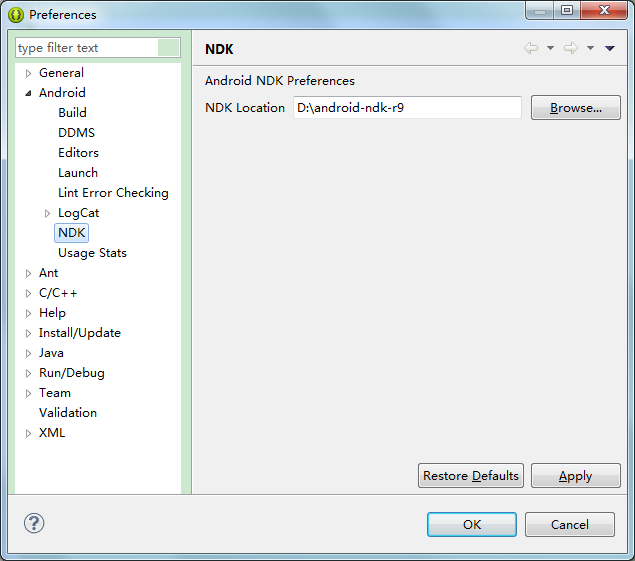
* 把方法名，复制到c中，把参数加上参数名即可
* 这个文件在eclipse中会报错
  + 把文件删掉，反正用完了
  + 或者



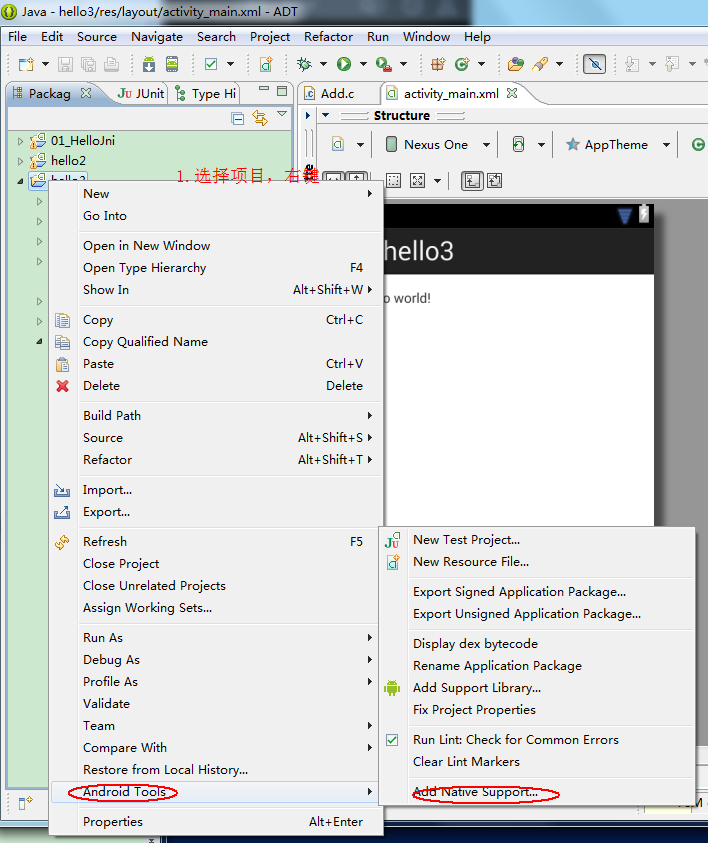
* + 右键把错误删掉就行

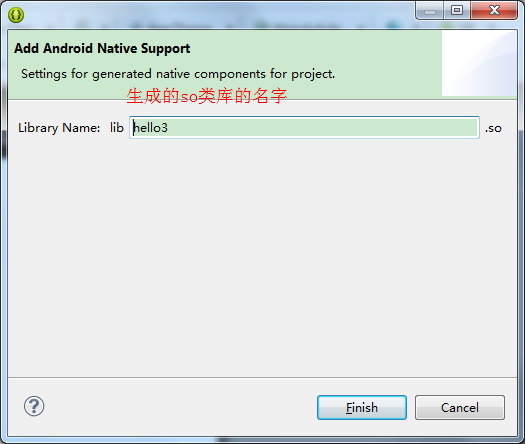
# 配置ndk到开发环境和关联源码(导入include路径)

* 配置ndk到开发环境



* 配置完ndk，可以自动生成jni目录，里面有c文件和Android.mk，不过生成的c文件时cpp（c++的），改成c就行

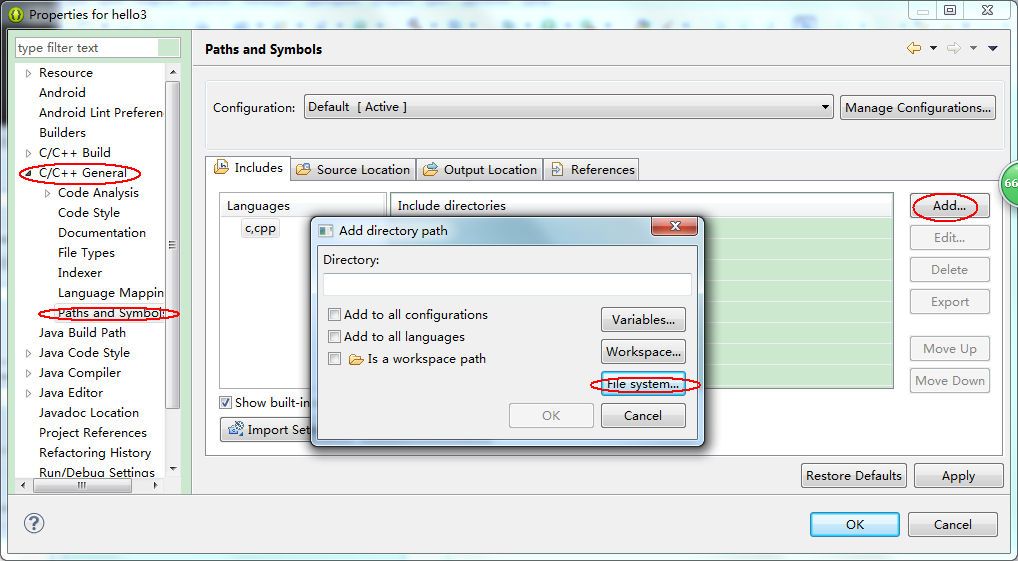




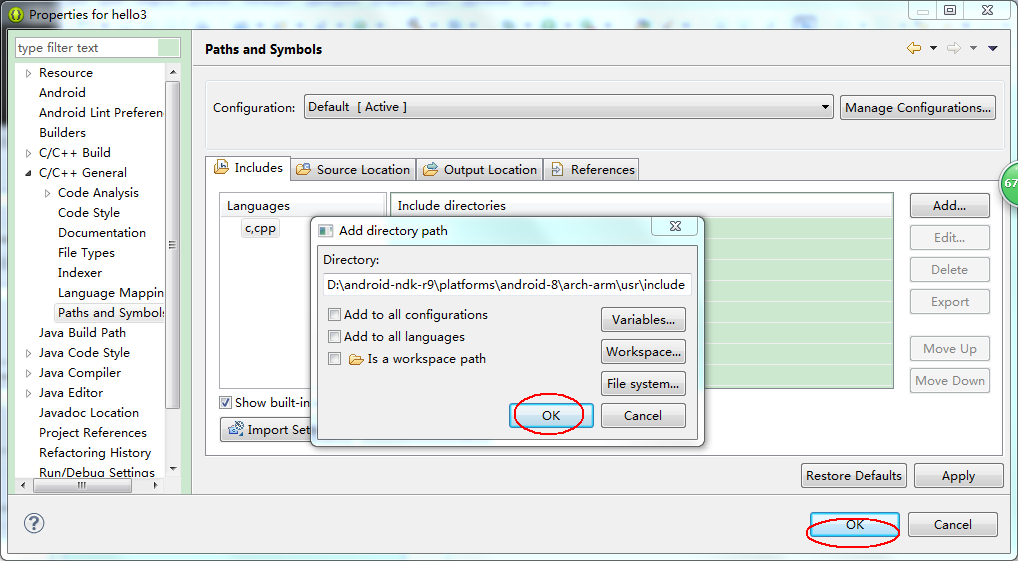
* 生成之后的jni文件夹是删不掉的，如果类库名写错，只能重新创建一个项目了



* 这里有个警告，是因为没有关联源码
* 关联源码：项目🡪右键🡪Properties



* 选择到ndk的platforms目录，然后选择当前项目Android版本对应的版本，然后选择一个处理器架构的文件夹（哪个架构无所谓），然后就是usr/include



* 这时候，就可以像java一样，ctrl左键，查看源码了。也可以提示出函数
* 项目写好后，不用自己手动打包类库，先调用。因为，部署项目的时候，会自动打包编译出类库，然后在执行项目。每次部署都会自动编译打包类库

# 密码加密

* + 使用ndk很大一方面是为了防止反编译
  + Java比较容易被反编译，c就很难
  + 所以对密码进行加密的操作，写在c中，然后打包成类库，会安全很多
  + C代码打包成了so类库，c文件就没作用了，删不删无所谓。Java中直接加载类库，然后调用类库里的方法来使用的
  + 要对类库反编译，很难
  + 反编译方法：



* + 密码加密原理：对用户设置的密码，进行加密算法计算后，得到加密后的密码，保存在数据库。每次用户要输入密码是，把输入的密码再进行一次加密计算，得到的结果，如果和数据库中的密码一样，密码正确，否则反之
  + 如果密码加密的算法，在java中，会被反编译出来。就知道了加密的过程，就能破解密码

# 字符串加密

# 把数组传递个c进行加密

# 黑马秀秀

* 美图秀秀，美化图片的方法，都是本地方法，在JNI类里面（反编译后可以看到）
* 我们拿到他所有的本地方法(类名和包名不能改变它，因为打包类库的时候用的是原来的包名类名，改了就不能用类库了)，然后拿到本地方法编译出来的类库(在apk里面可以拿到，解压apk)
* 类库的名字就是：so文件，掐头去尾（前面的lib去掉，后面的.so去掉），就是类库名了
* 这个类库是arm架构的，所以只能在arm架构的平台上运行
* 因为它的所有美化方法都是在JNI这个类里的本地方法中，所以我们直接用它的本地方法就行了。就做出一个自己的秀秀软件

# 在c中，调用java的代码

* 用处：把C加载资源的进度，显示在activity，要用c来调用java的方法
* C调用java，要用到反射

## 在c中，把字符串输出到logcat

* Android.mk文件增加LOCAL\_LDLIBS += -llog
* C代码中增加，这是google定义好的c函数
* **#include** <android/log.h>
* **#define** LOG\_TAG "System.out"
* **#define** LOGD(...) \_\_android\_log\_print(ANDROID\_LOG\_DEBUG, LOG\_TAG, \_\_VA\_ARGS\_\_)
* **#define** LOGI(...) \_\_android\_log\_print(ANDROID\_LOG\_INFO, LOG\_TAG, \_\_VA\_ARGS\_\_)
* 然后直接使用
* LOGI("info\n");
* LOGD("debug\n");

## c调用java中的代码

* 看视频，因为不重要