

宝贵建议请发送至：wangzhenyang@itcast.cn



黑马程序员

itheima.com

-编程，始于黑马

Android 课程同步笔记

Beta 0.01 版

By 阳哥

Android-JNI-02NDK&JNI 入门

1. NDK 简介 (★★)

1.1 NDK 产生的背景

Android 平台从诞生起，就已经支持 C、C++ 开发。众所周知，Android 的 SDK 基于 Java 实现，这意味着基于 Android SDK 进行开发的第三方应用都必须使用 Java 语言。但这并不等同于“第三方应用只能使用 Java”。在 Android SDK 首次发布时，Google 就宣称其虚拟机 Dalvik 支持 JNI 编程方式，也就是第三方应用完全可以通过 JNI 调用自己的 C 动态库，即在 Android 平台上，“Java+C”的编程方式是一直都可以实现的。

不过，Google 也表示，使用原生 SDK 编程相比 Dalvik 虚拟机也有一些劣势，Android SDK 文档里，找不到任何 JNI 方面的帮助。即使第三方应用开发者使用 JNI 完成了自己的 C 动态链接库（so）开发，但是 so 如何和应用程序一起打包成 apk 并发布？这里面也存在技术障碍。比如程序更加复杂，兼容性难以保障，无法访问 Framework API，Debug 难度更大等。开发者需要自行斟酌使用。

于是 NDK 就应运而生了。NDK 全称是 Native Development Kit。

NDK 的发布，使“Java+C”的开发方式终于转正，成为官方支持的开发方式。NDK 将是 Android 平台支持 C 开发的开端。

1.2 为什么使用 NDK

- ◆ 1.代码的保护。由于 apk 的 java 层代码很容易被反编译，而 C/C++ 库反编译难度较大。
- ◆ 2.可以方便地使用现存的开源库。大部分现存的开源库都是用 C/C++ 代码编写的。
- ◆ 3.提高程序的执行效率。将要求高性能的应用逻辑使用 C 开发，从而提高应用程序的执行效率。
- ◆ 4.便于移植。用 C/C++ 写得库可以方便在其他的嵌入式平台上再次使用。

1.3 NDK 简介

◆ 1.NDK 是一系列工具的集合

NDK 提供了一系列的工具，帮助开发者快速开发 C (或 C++) 的动态库，并能自动将 so 和 java 应用一起打包成 apk。NDK 集成了交叉编译器，并提供了相应的 mk 文件隔离 CPU、平台、ABI 等差异，开发人员只需要简单修改 mk 文件（指出“哪些文件需要编译”、“编译特性要求”等），就可以创建出 so。

◆ 2.NDK 提供了一份稳定、功能有限的 API 头文件声明


Google 明确声明该 API 是稳定的，在后续所有版本中都稳定支持当前发布的 API。从该版本的 NDK 中看出，这些 API 支持的功能非常有限，包含有：C 标准库（libc）、标准数学库（libm）、压缩库（libz）、Log 库（liblog）。

1.4 NDK 的安装

◆ 1.NDK 的下载

NDK 的官方下载地址 <http://developer.android.com/tools/sdk/ndk/index.html>，由于官方网址在国外，国内访问不了，必须得翻墙。因此我提供了下载好的 NDK 工具放在百度网盘上供大家下载。

<http://pan.baidu.com/s/1jGpCDKi>

 android-ndk-r9b-windows-x86.zip	2014/4/17 19:09	好压 ZIP 压缩文件	445,243 KB
--	-----------------	-------------	------------

◆ 2.将 NDK 解压到一个不包含空格和中文的目录下

本人将 NDK 解压在 D:\software\ndkr9\android-ndk-r9b 中。

1.5 NDK 目录结构说明

名称	修改日期	类型	大小
build	2015/1/15 15:53	文件夹	
docs	2015/1/15 15:53	文件夹	
platforms	2015/1/15 15:52	文件夹	
prebuilt	2015/1/15 15:52	文件夹	
samples	2015/1/15 15:52	文件夹	
sources	2015/1/15 15:52	文件夹	
tests	2015/1/15 15:53	文件夹	
toolchains	2015/1/15 15:53	文件夹	
documentation.html	2012/8/21 13:23	HTML 文档	1 KB
find-win-host.cmd	2013/8/8 5:20	Windows 命令脚本	1 KB
GNUmakefile	2012/8/21 13:23	文件	2 KB
ndk-build	2013/9/9 12:38	文件	10 KB
ndk-build.cmd	2015/1/15 18:42	Windows 命令脚本	1 KB
ndk-depends.exe	2013/8/30 22:07	应用程序	134 KB
ndk-gdb	2013/6/11 13:31	文件	24 KB
ndk-gdb.py	2013/8/13 11:32	PY 文件	33 KB
ndk-gdb-py	2013/6/11 13:31	文件	1 KB
ndk-gdb-py.cmd	2013/8/5 15:14	Windows 命令脚本	1 KB
ndk-stack.exe	2013/8/30 22:06	应用程序	164 KB
ndk-which	2012/9/10 12:06	文件	2 KB
README.TXT	2012/8/21 13:23	TXT 文件	2 KB
RELEASE.TXT	2013/10/29 15:25	TXT 文件	1 KB
remove-windows-symlink.sh	2013/8/5 15:14	SH 文件	2 KB

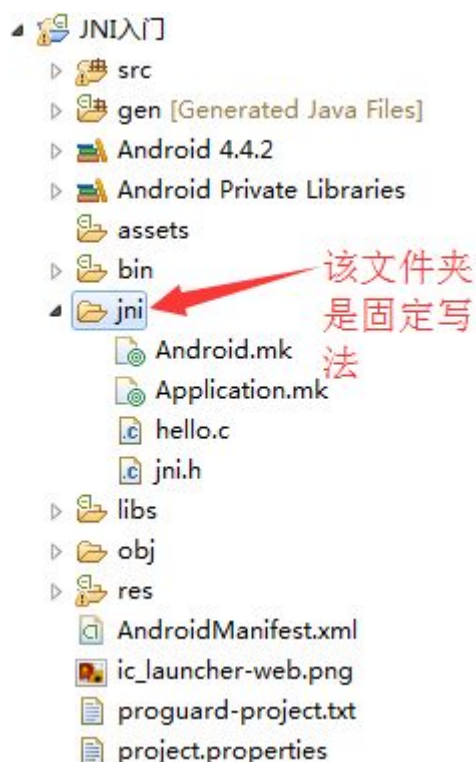
- ◆ build：该目录存放的使用 NDK 的 mk 脚本，mk 脚本指定了编译参数
- ◆ docs：该目录存放的是 NDK 的使用帮助文档
- ◆ platforms：这里面存放的是与各个 Android 版本相关的平台（x86，arm，mips）相关 C 语言库和头文件
- ◆ prebuilt：预编译工作目录
- ◆ samples：存放的是演示程序
- ◆ sources：存放的是 NDK 工具链的 C 语言源码
- ◆ tests：测试相关的文件

- ◆ toolchains : 工具链，存放了三种架构的静态库等文件
- ◆ ndk-build.cmd : Window 平台使用 NDK 的命令
- ◆ ndk-build : Linux 平台使用 NDK 的命令

2. JNI 入门 (★★★)

下面通过一个简单的 JNI 案例来演示如何使用 JNI 编程。

- 1 创建一个新的 Android 工程《JNI 入门》，工程的最终目录结构如下图所示。



- 2 在 MainActivity.java 类中定义一个 native 方法

```
//定义一个 native 方法，意思是该方法的具体实现交给 C 语言实现  
public native String helloC();
```

- 3 在工程跟目录下创建一个文件夹 jni，该目录名称是约定（约定优于配置）好的，不能是其他名字。

- 4 在 jni 目录下创建 hello.c 源文件，文件名可以按照见名知意的规则来创建。hello.c 代码清单如下。

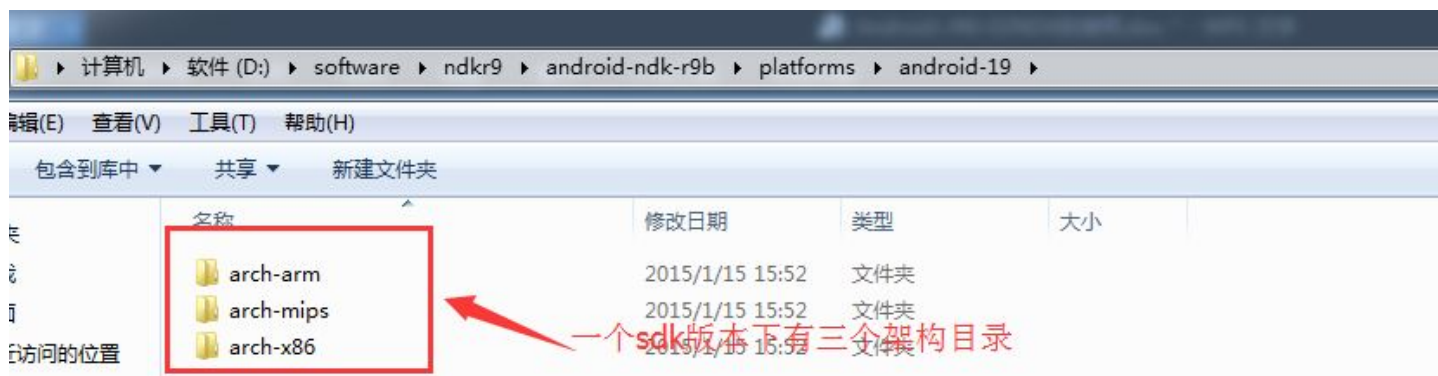

```
#include<stdio.h>//引入头文件
//引入 jni.h jni.h 文件里面定义了 jni 的规范，jni.h 在 ndk 的目录中找到，然后放到当前工程
中的 jni 目录下即可
#include<jni.h>
//定义在 MainActivity.java 类中的 helloC 对应的 C 语言函数
jstring Java_com_itheima_jnihello_MainActivity_helloC(JNIEnv* env, jobject obj) {
    char* str = "hello from C";
    //调用 jni.h 中定义的创建字符串函数
    jstring string = (*env).NewStringUTF(env, str);
    return string;
}
```

Tips : 上面的代码虽然简单但是关于 jni.h 头文件和方法名必须单独说明。

◆ 1) jni.h 头文件位于 NDK 安装目录下/platforms/android-*/(某平台)/usr/include 目录中，如下图



上面的某平台指 CPU 的三种架构如下图。我们选择任意一架构皆可，但是对于手机来说 CPU 用 arm 架构的最多，x86 次之，mips 架构最少。



◆ 2) JNI 中 C 源文件方法名的命名规则

这里的命名规则指用于跟 java 文件中 native 方法对应的 C 语言方法，而 C 语言中的其他方法命名只要符合 C 语言规则就行。jstring Java_com_itheima_jnihello_MainActivity_helloC(JNIEnv* env, jobject obj) 中，jstring 是方法返回值类型，我们可以把 jstring 看成是 java 中 String 跟 C 语言中 char* 类型的一个中

间转换类型，java 跟 C 语言的数据类型是不一样的，他们之间要想互相调用就必须通过一种中介来实现，这个中介就是在 jni.h 头文件中定义的。关于更多的转换类型，在本文档的第 2 章会有更详细的说明。

方法名第一个字母必须是 Java，首单词大写，然后下划线_，然后将该方法所在的包、类、方法用 “_” 连接起来，比如 com.itheima.jnihello.MainActivity 类中的 helloC 方法，转变成 C 语言中的方法名为 **Java_com_itheima_jnihello_MainActivity_helloC**。

方法的形参有两个是必须的也就是不管 java 中的方法是否有形参，但是 C 语言中对应的方法必须有 JNIEnv* env，和 jobject obj，如果 java 方法中还用其他形参，那么在 C 语言中严格按照顺序排在 jobject obj 参数的后面即可。

上面的 env 代表指向 JVM 的指针，obj 是调用该方法的 java 对象。

5

使用 NDK 工具将 hello.c 编译成 hello.so 文件

◆ 为了方便直接在控制台中使用 NDK 工具的 ndk-build.cmd 命令，我们首先将 ndk-build.cmd 所在的目录设置成系统环境变量。环境变量配置好以后，在命令行中输入 ndk-build.cmd 会有如下提示：



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\thinkpad>ndk-build.cmd

C:\Users\thinkpad>rem This is a Windows cmd.exe script used to invoke the NDK-specific GNU Make executable

C:\Users\thinkpad>call "D:\software\ndkr9\android-ndk-r9b\find-win-host.cmd" NDK_WIN_HOST
Android NDK: Could not find application project directory !
Android NDK: Please define the NDK_PROJECT_PATH variable to point to it.
D:\software\ndkr9\android-ndk-r9b\build\core\build-local.mk:130: *** Android NDK: Aborting . Stop.

C:\Users\thinkpad>
```

◆ 将当前目录切换到 hello.c 所在的工程目录

这时候如果直接输入 ndk-build.cmd 那么会出现如下异常：

```
F:\heima33\JNI\JNI02\code\01JNI入门\jni>ndk-build.cmd
Android NDK: Could not find application project directory !
Android NDK: Please define the NDK_PROJECT_PATH variable to point to it.
F:\heima33\JNI\JNI02\code\01JNI入门\jni>ndk-build.cmd
ld-local.mk:130: *** Android NDK: Aborting . Stop.
F:\heima33\JNI\JNI02\code\01JNI入门\jni>ndk-build.cmd hello.c
```

出现这种错误时因为我们并没有告诉 ndk 我们要将那个 C 语言源代码编译成目标文件。为了告诉 ndk 要将那个 C 源文件编译成目标文件，我们需要在工程中的 jni 目录中添加 Android.mk 配置文件。

6 在当前工程的 jni 目录下添加 Android.mk 配置文件，该配置文件可以从 ndk 安装目录的实例代码中拷贝，然后修改。

Android.mk 文件清单如下，我们只需要修改 `LOCAL_MODULE` 和 `LOCAL_SRC_FILES` 两个参数即可。

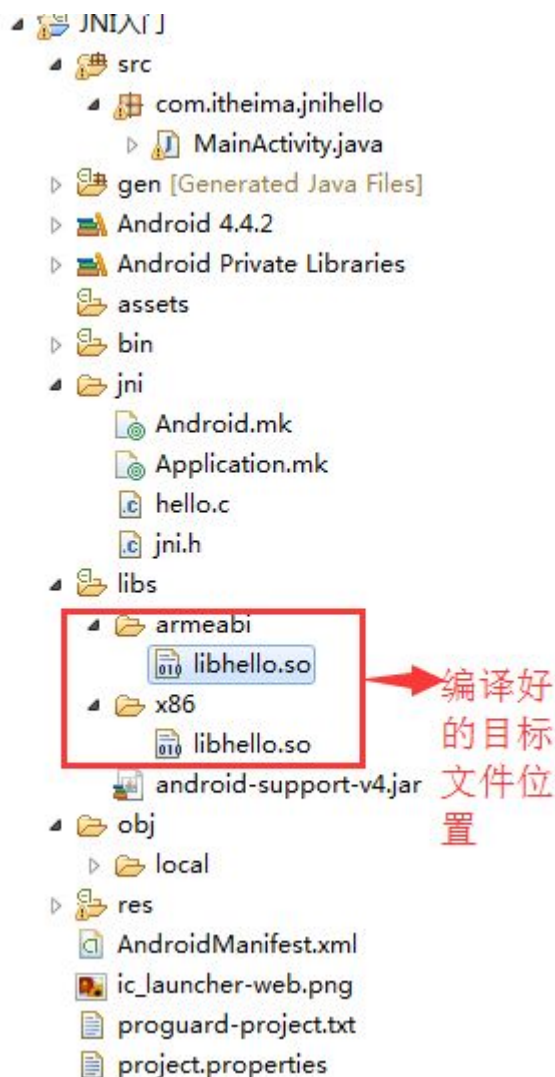
`LOCAL_MODULE` 参数是指定编译后的目标文件的名称，其实编译好的目标文件名为 `libhello.so`，`LOCAL_SRC_FILES` 指定了要编译的源文件。

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := hello
LOCAL_SRC_FILES := hello.c
include $(BUILD_SHARED_LIBRARY)
```

7 在 cmd 中，将当前目录切换到 hello.c 所在目录，然后重新执行 `ndk-build.cmd` 命令，这次成功编译，cmd 显示效果如下图所示。

```
C:\Users\thinkpad\workspace\JNI入门\jni>ndk-build.cmd
C:\Users\thinkpad\workspace\JNI入门\jni>rem This is a Windows cmd.exe script use
d to invoke the NDK-specific GNU Make executable
C:\Users\thinkpad\workspace\JNI入门\jni>call "D:\software\ndkr9\android-ndk-r9b\
find-win-host.cmd" NDK_WIN_HOST
[armeabi] Compile thumb   : hello <= hello.c
[armeabi] SharedLibrary   : libhello.so
[armeabi] Install        : libhello.so => libs/armeabi/libhello.so
[x86] Compile             : hello <= hello.c
[x86] SharedLibrary      : libhello.so
[x86] Install             : libhello.so => libs/x86/libhello.so
C:\Users\thinkpad\workspace\JNI入门\jni>
```


查看项目目录结构，发现在 libs 目录中多了两个文件夹 armeabi 和 x86，这两个文件夹下分别包含了一个 libhello.so 动态链接库。这也代表着当前工程中的动态库支持 arm 架构和 x86 架构的 cpu。



Tips：可能你的并没有同时生成这两个文件，是因为我的工程中引入了 Application.mk 文件，因此你需要引入该文件。

Application.mk 文件清单：

```
# Build both ARMv5TE and ARMv7-A machine code.  
APP_ABI := armeabi x86
```

该清单其实只有一行内容 第一行是注释。APP_ABI 参数指定要生成的目标文件支持的平台都有哪些 默认是 armeabi

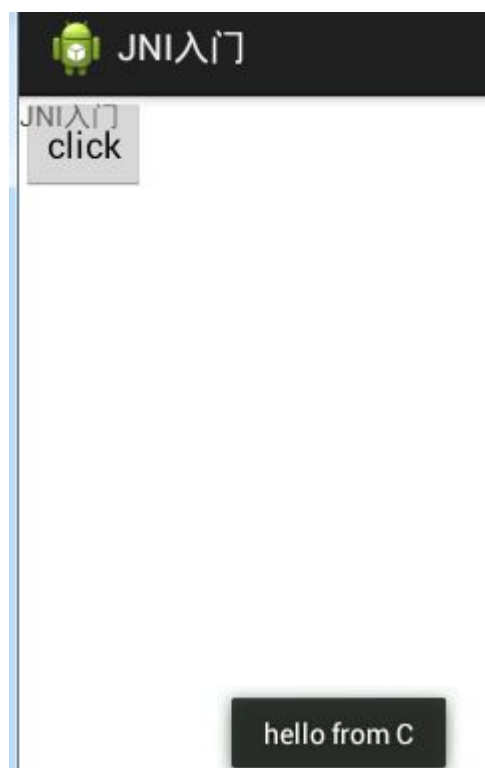
如果想支持多个平台只需要空一格然后写出其他平台名字即可。

8

在 MainActivity.java 中调用 C 语言

```
public class MainActivity extends Activity {  
    //加载 libhello.so 动态库，但是我们加载的时候必须去掉 lib 和后缀  
    static{  
        System.loadLibrary("hello");  
    }  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
    //定义一个 native 方法，意思是该方法的具体实现交给 C 语言实现  
    public native String helloC();  
    //点击按钮调用 C 语言方法  
    public void click(View view){  
        Toast.makeText(this, helloC(), 1).show();  
    }  
}
```

运行上面工程，效果如下：



Tips：如果我们编译的 arm 平台的 so 文件，但是却部署到了 x86 平台的模拟器上，那么运行的时候会报找不到 libhello.so 的异常。

3. JNI 规范 (★)

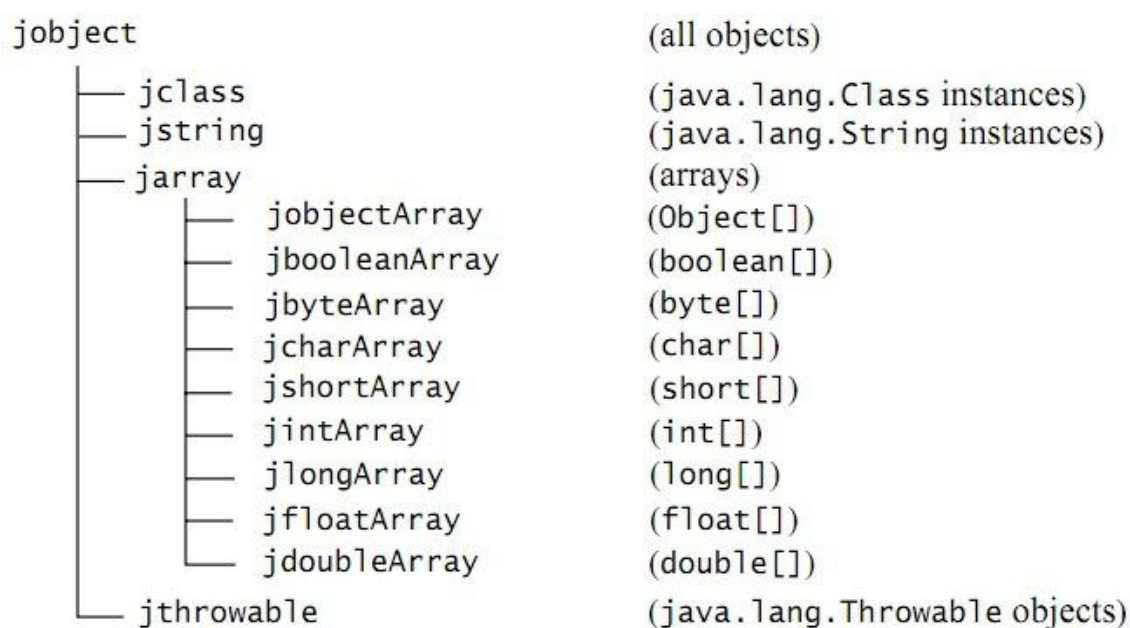
3.1 JNI 数据类型和数据结构

◆ 1) 基本数据类型

JNI 基本类型和本地等效类型的对应表格如下：

Java 类型	本地 C 类型	实际表示的 C 类型 (Win32)	说明
boolean	jboolean	unsigned char	无符号，8 位
byte	jbyte	signed char	有符号，8 位
char	jchar	unsigned short	无符号，16 位
short	jshort	short	有符号，16 位
int	jint	long	有符号，32 位
long	jlong	__int64	有符号，64 位
float	jfloat	float	32 位
double	jdouble	double	64 位
void	void	N/A	N/A

◆ 2) 引用类型，JNI 还包含了很对对应于不同 Java 对象的引用类型，JNI 引用类型的组织层次如下图所示：



3.2 JNI 接口函数命名方式

◆ 1) 类型签名

Java 虚拟机的类型签名如下：

类型签名	Java 类型
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
Lfully-qualified-class;	全限定类
[type	type[] 数组
(argtypes)rettype	方法类型

例如，Java 方法 `int feet(int n, String s,int [] arr)`的类型签名如下：

`(ILjava/lang/String;[I) I`

圆括号里面为参数，I 表示第一个参数 int 型，Ljava/lang/String;表示第二个参数为全限定 Java.lang.String 类型，[I 表示第三个参数为 int 型的数组，圆括号后面为返回值类型，I 表示返回值为 int 型。

◆ 2) 一般函数的 JNI 接口函数命名方式

一般 JNI 接口函数命名如下：

Java_包名_类名_方法名。

例如：某工程下 com/itheima 包下 MainActivity 类的 `int getIntFromC()`方法的 C 语言实现函数命名如下：

`jint Java_com_itheima_MainActivity_getIntFromC(JNIEnv* env,jobject obj)`

其中，包名所包含的 “/” 应全部以下划线替代，其本地实现的参数和返回值也应转换为 JNI 类型。

◆ 3) 重载函数的 JNI 接口函数命名方式

重载函数的 JNI 实现在一般函数的 JNI 实现之外，还应添加上类型签名以作为同名函数之间的区别，其接口函数命名如下：Java_包名_类名_方法名_参数签名。

例如：某工程下 com/itheima 包下 MainActivity 类的 int getIntFromC(int n, String s,int [] arr)方法的 C 语言实现函数命名如下：

```
jint Java_com_itheima_MainActivity_getIntFromC_ILJava_lang_String_2_3I
```

```
(JNIEnv* env, jobject obj, jint n, jstring s, jintarray arr)。
```

JNI 在函数命名时采用名字扰乱方案，以保证所有的 Unicode 字符都能

转换为有效的 C 函数名，所有的 “/” ,无论是包名中的还是全限定类名中的，均使用 “_” 代替，用_0,,,_9 来代替转义字符，如下：

转义字符序列	表示
_0XXXX	Unicode 字符 XXXX
1	字符 “”
_2	签名中的字符 “; ”
_3	签名中的字符 “[”

3.3 JNI 函数与 API

在本文档中我们所主要需要关心的是 C/C++ 数据类型与 JNI 本地类型之间的转化过程，这个过程某些数据的转换需要使用 JNIEnv 对象的一系列方法来完成。

◆ 1) jstring 转换为 C 风格字符串

```
char* test = (char*)(*env)->GetStringUTFChars(env,jstring,NULL);
```

使用完毕后，应调用：

```
(*env)->ReleaseStringUTFChars(env, jstring, test);
```

释放资源。

◆ 2) C 风格字符串转换为 jstring


```
char charStr[50];
```

```
jstring jstr;
```

```
jstr = env -> NewStringUTF(charStr);
```

◆ 3) C 语言中获取的一段 char* 的 buffer 传递给 Java

在 jni 中 new 一个 byte 数组，然后使用

(*env)->SetByteArrayRegion(env, bytearray, 0, len, buffer) 操作将 buffer 拷贝到数组中。

这种方式主要是针对 buffer 中存在 “\0” 的情况，如果以 C 风格字符串的方式读入，就会损失 “\0” 之后的字符。

◆ 4) 数组操作

JNI 函数	功能
GetArrayLength	返回数组中的元素数
NewObjectArray	创建一个指定长度的原始数据类型数组
GetObjectArrayElement	返回 Object 数组的元素
SetObjectArrayElement	设置 Object 数组的元素
GetObjectArrayRegion	将原始数据类型数组中的内容拷贝到预先分配好的内存缓存中
SetObjectArrayRegion	设置缓存中数组的值
ReleaseObjectArrayRegion	释放 GetObjectArrayRegion 分配的内存

Tips：对 int，char 等基本数据类型的数组操作，将相关 Object 名称替换为对应基本数据类型名称即为相关函数。

数组操作的方法选择基于使用者的需求而定，如果使用者需要在内存中拷贝数组并对其进行操作那么一般使用

GetObjectArrayRegion 和 SetObjectArrayRegion 函数，否则一般使用 SetObjectArrayElement 和

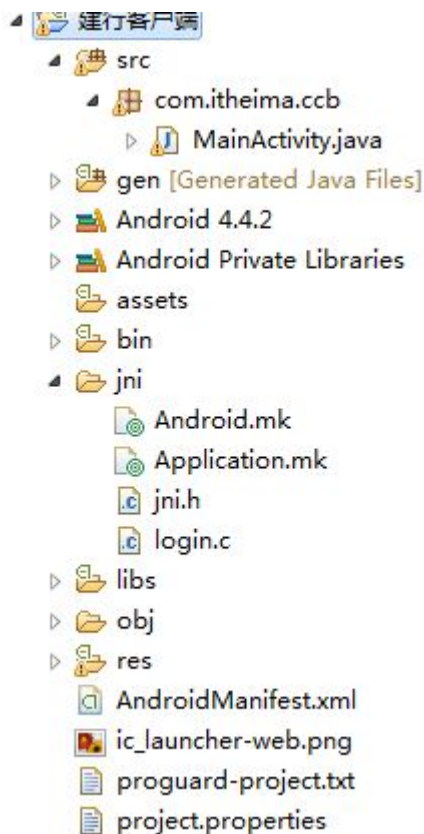
GetObjectArrayElement 函数。

4. 案例-银行登录系统 (★★)

需求：假设银行的登陆模块是用 C 语言来编写的，但是我们的 Android 应用想登陆银行系统，那么就需要通过 JNI 来实现了。

1

创建一个新 Android 工程《建行客户端》，工程目录结构如下图。



2

在工程中创建 jni 文件夹，然后将 jni.h、Android.mk、Application.mk 从 JNI 入门工程拷贝进去。

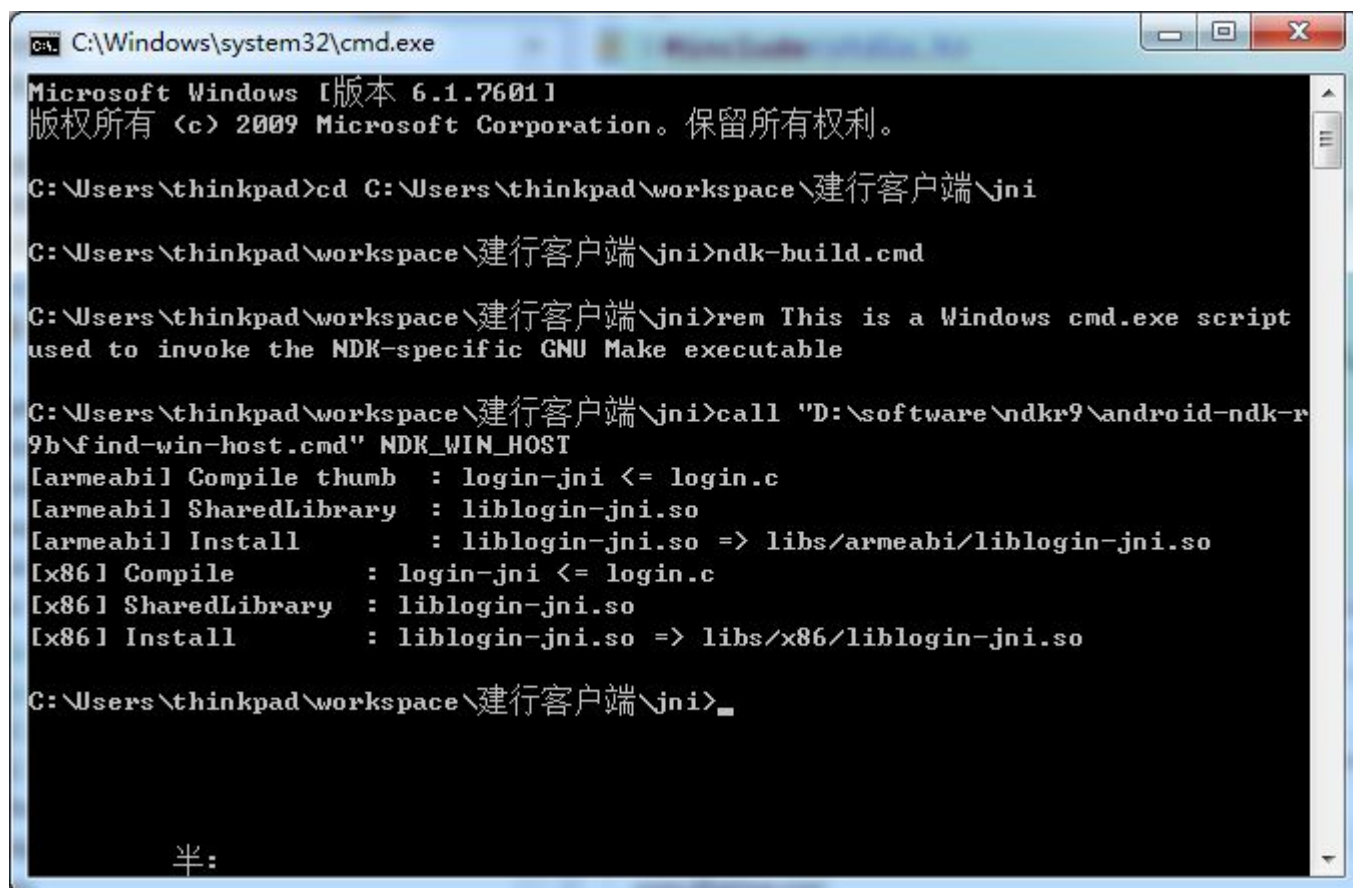
3

在 jni 目录下创建 login.c 文件，在该文件中实现登录业务逻辑。代码清单如下。

```
#include<stdio.h>
//系统在查找头文件的时候""中的文件会去本地搜索，<>中的文件会去系统目录中搜索，因为 jni.h
在当前目录中所以用""将 jni.h 引起来，可以加快搜索速度
#include"jni.h"
int login(int card,int pwd){
    //真实的业务逻辑要复杂的多，这里只简单的返回银行卡号和密码号
    return card+pwd;
}
```

```
jint Java_com_itheima_ccb_MainActivity_login(JNIEnv* env, jobject obj, jint
card, jint pwd){
    return login(card, pwd);
}
```

4 是用 ndk 工具，将 login.c 编译成动态库文件。编译前修改 Android.mk 文件的 LOCAL_SRC_FILES := login.c



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\thinkpad>cd C:\Users\thinkpad\workspace\建行客户端\jni

C:\Users\thinkpad\workspace\建行客户端\jni>ndk-build.cmd

C:\Users\thinkpad\workspace\建行客户端\jni>rem This is a Windows cmd.exe script
used to invoke the NDK-specific GNU Make executable

C:\Users\thinkpad\workspace\建行客户端\jni>call "D:\software\ndkr9\android-ndk-r
9b\find-win-host.cmd" NDK_WIN_HOST
[armeabi] Compile thumb   : login-jni <= login.c
[armeabi] SharedLibrary   : liblogin-jni.so
[armeabi] Install        : liblogin-jni.so => libs/armeabi/liblogin-jni.so
[x86] Compile             : login-jni <= login.c
[x86] SharedLibrary      : liblogin-jni.so
[x86] Install            : liblogin-jni.so => libs/x86/liblogin-jni.so

C:\Users\thinkpad\workspace\建行客户端\jni>_

半:
```

5 编写在 MainActivity.java 类

```
public class MainActivity extends Activity {
    static{
        System.loadLibrary("login-jni");
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public native int login(int card, int pwd);
    public void login(View view){
```

```
EditText et_card = (EditText) findViewById(R.id.et_card);
EditText et_pwd = (EditText) findViewById(R.id.et_pwd);
int card = Integer.valueOf(et_card.getText().toString());
int pwd = Integer.valueOf(et_pwd.getText().toString());
int result = login(card, pwd);
Toast.makeText(this, ""+result, 1).show();
}
```

布局文件比较简单，这里就不再给出。

运行上面的代码，运行结果如下：



5. CDT 插件的安装 (★)

5.1 CDT 简介

CDT 项目致力于为 Eclipse 平台提供功能完全的 C/C++ 集成开发环境 (Integrated Development

Environment , IDE)。CDT 是完全用 Java 实现的开放源码项目（根据 Common Public License 特许的），它作为 Eclipse SDK 平台的一组插件。这些插件将 C/C++ 透视图添加到 Eclipse 工作台（Workbench）中，现在后者可以用许多视图和向导以及高级编辑和调试支持来支持 C/C++ 开发。

5.2 CDT 的下载

CDT 插件可以通过 eclipse 的在线安装，但是受限于跨国家网络访问，一般不是很好用。因此这里我主要给大家说的是如何离线安装。

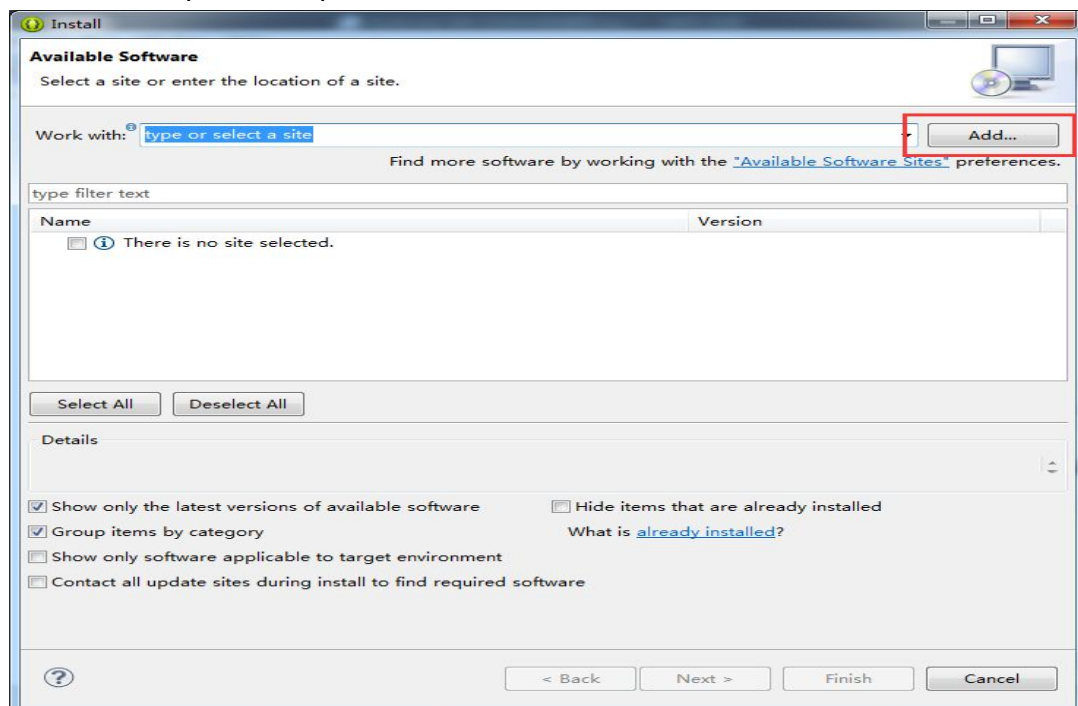
1 下载 CDT 离线安装包。针对不同版本 eclipse 的 cdt 安装包如下，大家可以从我的百度网盘上直接下载。考虑到我们大部分都用的最新的 ADT 因此建议选择 8.5.0 版本的 CDT。

cdt-8.5.0-for-Eclipse-Luna <http://pan.baidu.com/s/1c0m1k0w>

cdt-8.3.0-for-Eclipse-Kepler <http://pan.baidu.com/s/1kT21QOf>

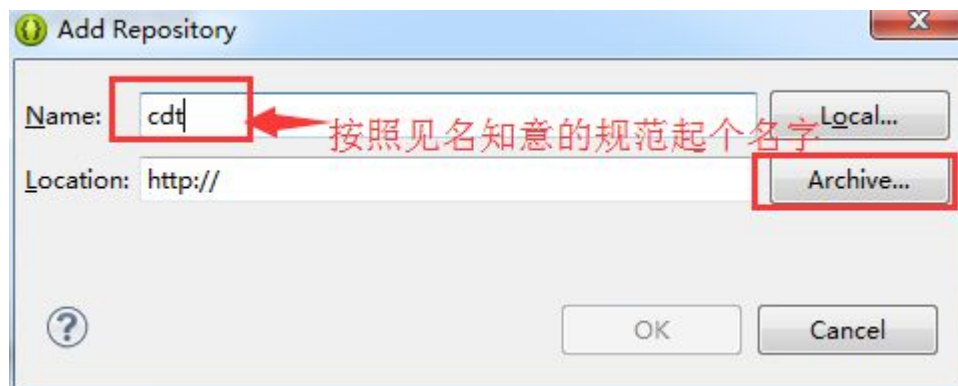
cdt-8.1.2-for-Eclipse-Juno <http://pan.baidu.com/s/1qWazjBI>

2 选择 eclipse 的 Help->Install New Software...，弹出如下对话框



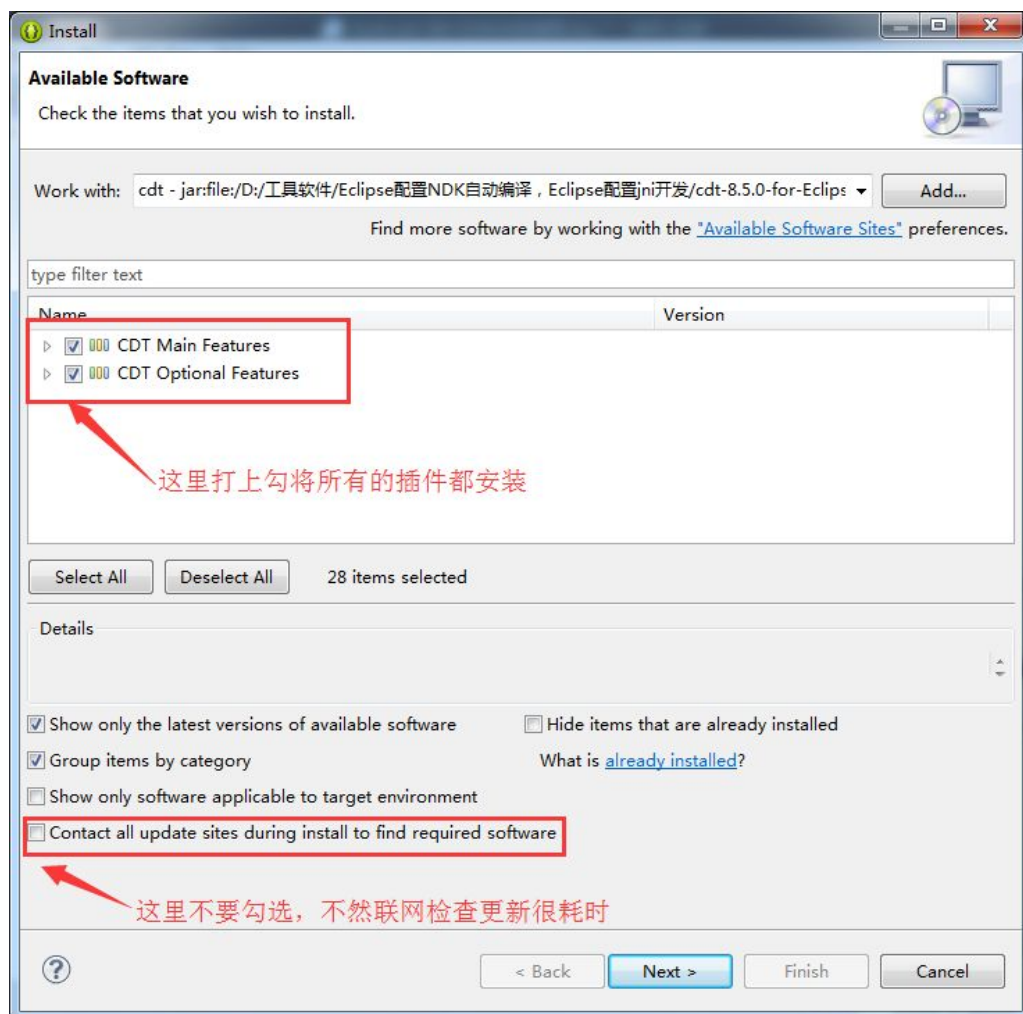
3

点击 Add 按钮，在弹出的对话框中输入 Name。在 Location 栏如果输入一个 http 地址是让 eclipse 自动从网络上下载安装，这里我们点击 Archive 按钮找到我们事先下载好的离线安装包。然后点击 OK。



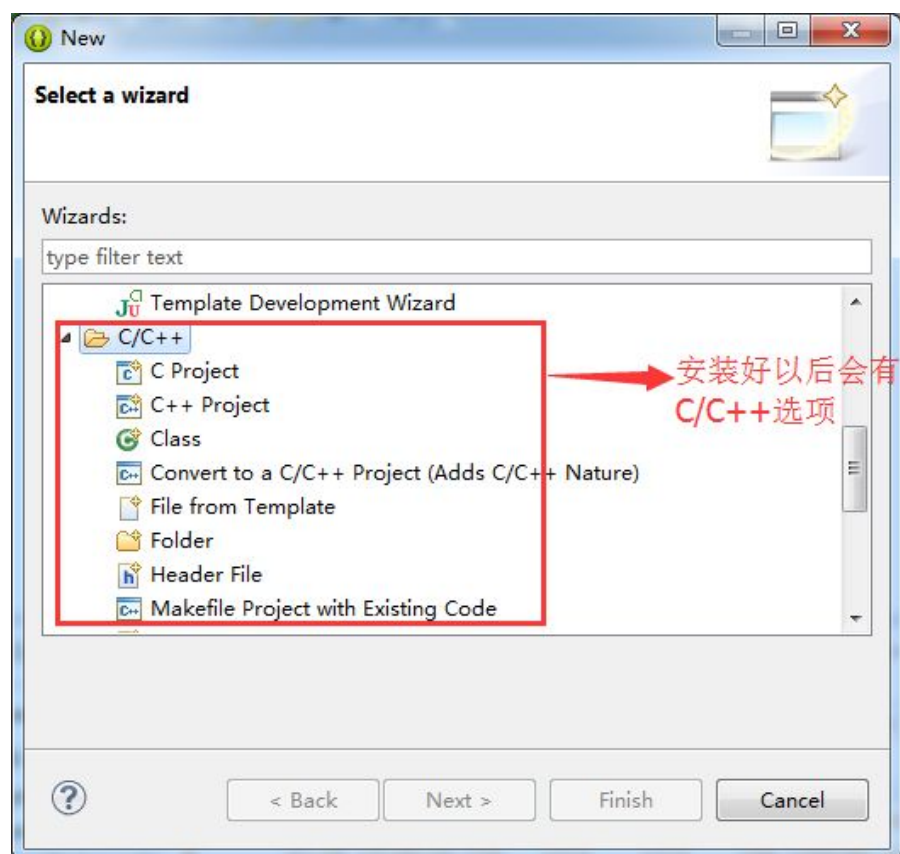
4

将 CDT 所有的插件勾选上，同时将最下面的联网检查更新去掉勾选，然后点击 Next，直到 Finish。

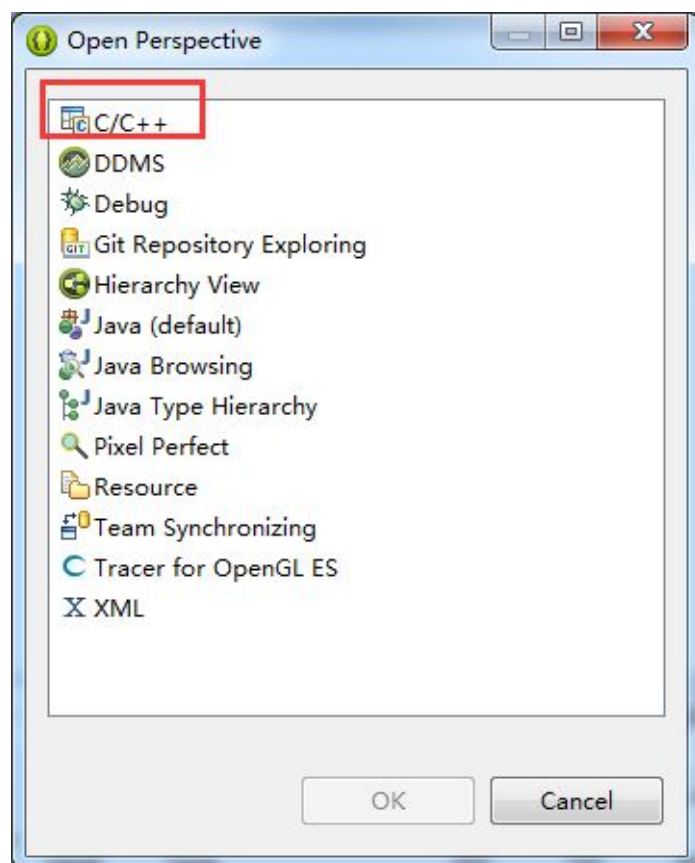


5

安装好以后在 File->New->Other 中会有 C/C++ 选项，如下图。



在 Open Perspective 中也多了 C/C++ 视图可选项，如下图。



安装好以后，我们就可以在 eclipse 中开发我们的 C/C++ 工程了。不过对我们 Android 开发人员来说用到的机会不是很多。就算是开发 C/C++ 工程，大多数程序员也不会选择在 eclipse 平台上进行开发。Eclipse 更多的是专注于 Java 语言项目的开发，比如 JavaEE、Android。

至此，本文档完！

2015 年 1 月 28 日 星期三 21:37:10
北京市海淀区东北旺中路东馨园小区