



【eoe 特刊】第 7 期: **NDK**



发布版本: Ver 1.0.0(build 2009.08.21)

© Copyright 2009 eoeAndroid.com. All Rights Reserved.

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

本期简介

在此之前, Android 平台的第三方应用程序均是依靠基于 Java 的 Dalvik 特制虚拟机进行开发的。原生 NDK 的公布可以让开发者更加直接的接触 Android 系统资源, 并使用传统的 C 或 C++ 语言编写程序, 并在程序封包文件 (.apks) 中直接嵌入原生库文件。NDK 的公布让原来从事 C 语言开发者也参与到 Android 平台, 也让 java 开发者方便的调用底层操作, 特别是游戏开者, 对引擎, 速度等要求较高者, 有了新的工具来实现所需功能, 做出更加完美的程序。

本期由由浅到深从环境部署到实例开发, 为你娓娓道来, 并提供文档提及的源码, 从而使你快速上手。

同时, 此期特刊由网友和特刊组成员热血完成, 在本刊中将为您一一介绍, 为作者和读者搭一桥梁, 相信在大家支持下, 特刊组将不断推出大家需求的文档。

万众期待的 eoeMarket 2.5 corn(玉米版)上线了!

万众期待的玉米版终于在今天上线了!

在 eoeMarket 的核心模块, 我们加入排行榜和标签云。

排行榜中有热度、日期、下载量和喜爱度不同的排行榜。

标签云, 可以让大家通过标签来查找应用。

此外, 登陆和注册画面的输入窗口的小问题也已经修正。网络连接、一些 bug 也已经修正。

这里特别加入了反馈功能, 大家可以直接在客户端, 给我们反馈你对 eoeMarket 的看法和需求。

下载页面: <http://www.eoemarket.com/download>

论坛下载: <http://www.eoeandroid.com/viewthread.php?tid=2675&extra=>



本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!



《Google Android 开发入门与实战》简介

- 作者: 靳岩 姚尚朗
- 出版社: 人民邮电出版社
- 书号: 9787115209306
- 出版日期: 2009 年 7 月
- 开本: 16 开
- 页码: 400

内容特点

国内第一本原创 Android 图书

完全基于 Android 最新的 SDK1.5

全书除了大量小型案例之外还包含了 5 个 Android 平台下的完整商业实例及源码分析, 分别是 RSS 阅读器、基于 GoogleMap 的个人 GPS、豆瓣客户端、在线音乐播放器、手机信息助手

随书附赠的光盘中包含 300 分钟的详细教学视频以及 Android 开发必备的开发资源

读者对于此书内容的疑问可以访问 <http://www.eoeandroid.com> 社区, 作者团队将会及时解答

1.   

目录

本期简介.....	2
万众期待的 eoeMarket 2.5 corn(玉米版)上线了!	3
《Google Android 开发入门与实战》简介.....	4
1.了解 NDK, Android NDK 带来什么.....	7
1.1 前言	7
1.2 误解	7
1.3 NDK 是什么	8
1.4 NDK 带来什么	8
2.环境部署 Windows xp Android NDK 环境搭建.....	10
2.1 ANDROID NDK 简介	10
2.2 搭建环境	10
3.Ubuntu android NDK 配置与开发.....	19
3.1 准备工作:	19
3.2 下面开始 NDK 的配置之旅	19
NDK 自带文档翻译与其它.....	24
4. Android1.5 NDK Release 1 中文说明文档.....	24
4.1 ANDROID NDK 是什么?	24
4.2 NDK 提供了:	24
4.3 NDK 的内容	24
4.4 文档	25
4.5 示例应用	26
4.6 系统和软件要求	26
4.7 安装 NDK	27
4.8 开始使用 NDK	27
4.9 示例使用	28
4.10 编者注:	28
5.Android NDK 概述.....	29
5.1 ANDROID NDK 的目标:	29
5.2 不是 ANDROID NDK 的目标:	30
5.3 NDK 开发实践:	30
5.4 配置 NDK:	31
5.5 放置 C 和 C++代码:	31
5.6 ANDROID.MK 编译脚本:	31
5.7 编写 APPLICATION.MK 编译文件:	32
5.8 调用 NDK 编译系统:	33
5.9 调试支持:	33

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

6.Android.mk 文件语法详解.....	34
7.NDK doc 其余四篇文章译文.....	42
7.1 APPLICATION.MK 文件语法详述.....	42
7.2 ANDROID NDK HOW-TO:.....	44
7.3 ANDROID NDK STABLE APIS:.....	46
7.4 ANDROID SYSTEM IMAGE ISSUES.....	48
8.实例分析与入门实例 NDK 自带实例分析.....	51
8.1 前言:	51
8.2 熟悉环境:	51
8.3 万里长征第一步:	53
8.4 从错误中入门:	55
8.5 试验过程:	57
8.6 小结.....	58
8.7 离胜利还有一步: 利用 SO.....	59
8.8 后言:	60
9.NDK 入门开发实战 Ubuntu 版本.....	61
10.eoeMarket.....	74
11.eoe 特刊小组诚邀您的加入 & 下期预告.....	81
12.介绍特刊组成员.....	82
13.其他.....	83
14.编后语.....	84
15.eoeAndroid 社区最新动向:	85

1.了解 NDK, Android NDK 带来什么

作者: Jack 发表于 @ 2009 年 06 月 28 日 14:33:00

摘自: <http://blog.csdn.net/hhao137/archive/2009/06/28/4304664.aspx>

1.1 前言

6 月 26 日, Google Android 发布了 NDK, 引起了很多开发人员的兴趣。NDK 全称: Native Development Kit。下载地址为: http://developer.android.com/sdk/ndk/1.5_r1/index.html。

1.2 误解

新出生的事物, 除了惊喜外, 也会给我们带来一定的迷惑、误解。

1.2.1 误解一: NDK 发布之前, Android 不支持进行 C 开发

在 Google 中搜索“NDK”, 很多“Android 终于可以使用 C++ 开发”之类的标题, 这是一种对 Android 平台编程方式的误解。其实, 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。

当然这种误解的产生是有根源的: 在 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。我曾经花了不少时间, 安装交叉编译器创建 so, 并通过 asset (资源) 方式, 实现捆绑 so 发布。但这种方式只能属于取巧的方式, 并非官方支持。所以, 在 NDK 出来之前, 我们将“Java+C”的开发模式称之为灰色模式, 即官方既不声明“支持这种方式”, 也不声明“不支持这种方式”。

1.2.2 误解二: 有了 NDK, 我们可以使用纯 C 开发 Android 应用

Android SDK 采用 Java 语言发布, 把众多的 C 开发人员排除在第三方应用开发外 (注意: 我们所有讨论都是基于“第三方应用开发”, Android 系统基于 Linux, 系统级别的开发肯定是支持 C 语言的。)。NDK 的发布, 许多人会误以为, 类似于 Symbian、WM, 在 Android 平台上终于可以使用纯 C、C++ 开发第三方应用了! 其实不然, NDK 文档明确说明: it is not a good way。因为 NDK 并没有提供各种系统事件处理支持, 也没有提供应用程序生命周期维护。此外, 在本次发布的 NDK 中, 应用程序 UI 方面的 API 也没有提供。至少目前来说, 使用纯 C、C++ 开发一个完整应用的条件还不完备。

1.3 NDK 是什么

对 NDK 进行了粗略的研究后,我对“NDK 是什么”的理解如下:

1.3.1 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。这些工具对开发者的帮助是巨大的。

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。

NDK 可以自动地将 so 和 Java 应用一起打包,极大地减轻了开发人员的打包工作。

1.3.2 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 带来什么

1.4.1 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。

使用 NDK,我们可以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。

使用 NDK,我们可以将需要保密的应用逻辑使用 C 开发。毕竟,Java 包都是可以反编译的。

NDK 促使专业 so 组件商的出现。(乐观猜想,要视乎 Android 用户的数量)

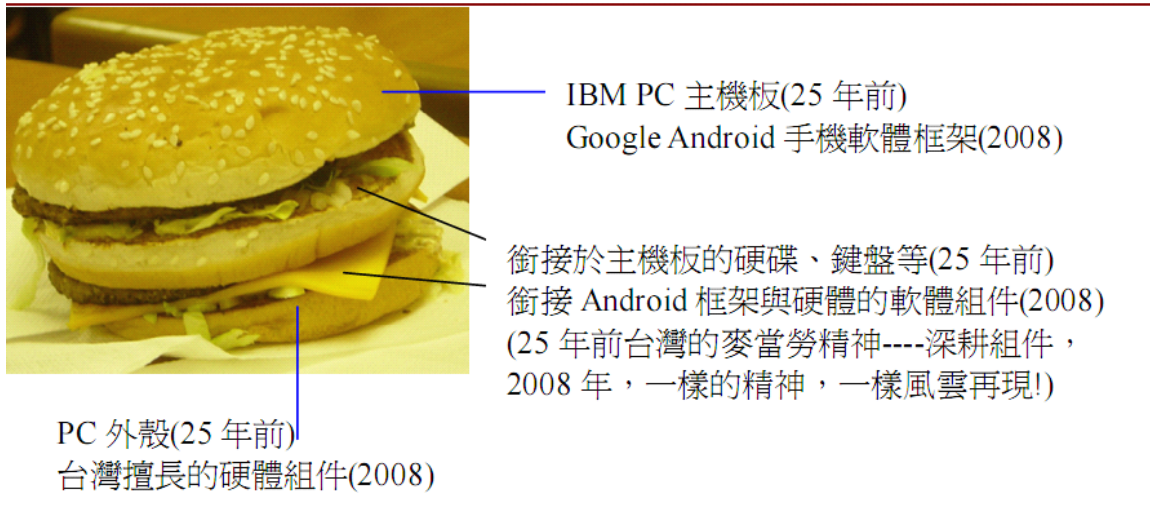
1.4.2 NDK 将是 Android 平台支持 C 开发的开端。

NDK 提供了的开发工具集合,使开发人员可以便捷地开发、发布 C 组件。同时,Google 承诺在 NDK 后续版本中提高“可调试”能力,即提供远程的 gdb 工具,使我们可以便捷地调试 C 源码。在支持 Android 平台 C 开发,我们能感觉到 Google 花费了很大精力,我们有理由憧憬“C 组件支持”只是 Google Android 平台上 C 开发的开端。毕竟,C 程序员仍然是码农阵营中的绝对主力,将这部分人排除在 Android 应用开发之外,显然是不利于 Android 平台繁荣昌盛的。

编者注:

关于 C 组件的前景,推荐参看:高焕堂的《Android_36 技.pdf》其中的 P323 中 11.2 节:发展 Android C 组件的经济意义。特别引用其中的图片,如下:

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!



2.环境部署 Windows xp Android NDK 环境搭建

作者: 曹立明

Email:clm16668@gmail.com

2.1 Android NDK 简介

Android NDK 是运行于 Android 平台上的 Native Development Kit 的缩写。它跟 NDK(日本电波工业株式会社)一点关系都没有 O(∩_∩)O~。

Android 应用开发者可以通过 NDK 调用 C 或 C++本地代码。

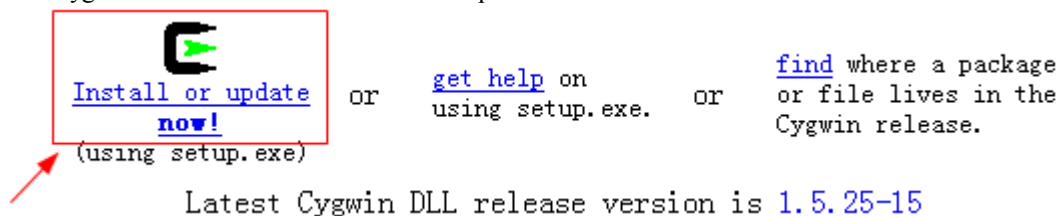
更多介绍参见: http://developer.android.com/sdk/ndk/1.5_r1/index.html

不过国内的朋友暂时只有用代理访问。

给个国内可以下载 NDK 的地址: http://dl.google.com/android/ndk/android-ndk-1.5_r1-windows.zip

2.2 搭建环境

NDK 编译需要用到 Cygwin 中的 make 和 gcc,所以先来下载并安装 Cygwin。先进入 www.cygwin.com, 点击其中的 Install or update now 链接, 如下图:



这将下载一个名为 setup.exe 的安装文件, 下载后双击启动安装。

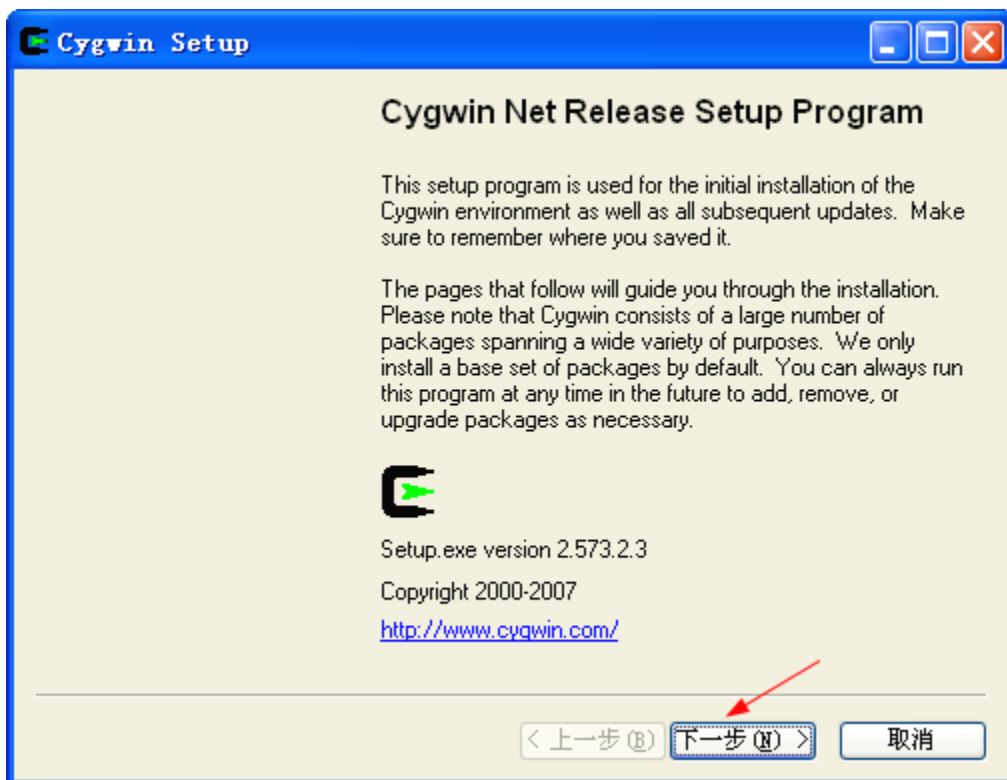
Cygwin 的安装有两种方式, 一种是本地安装, 这里采用另一种在线安装方式。

1: 启动安装

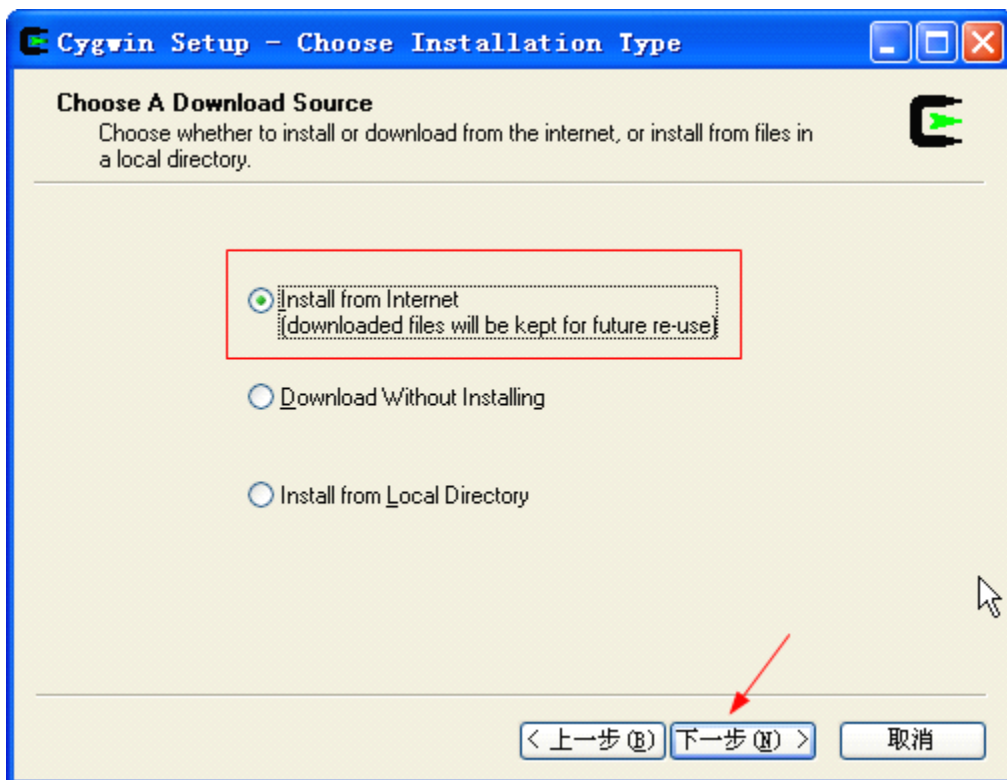


2: 下一步

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

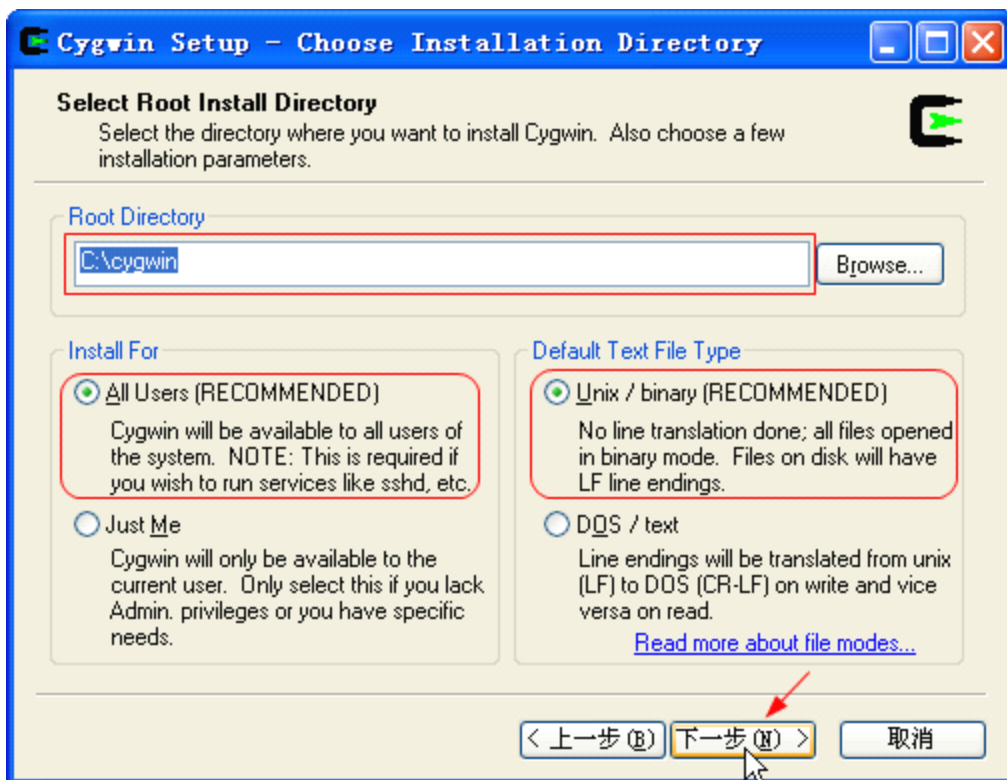


3: 选择在线安装方式

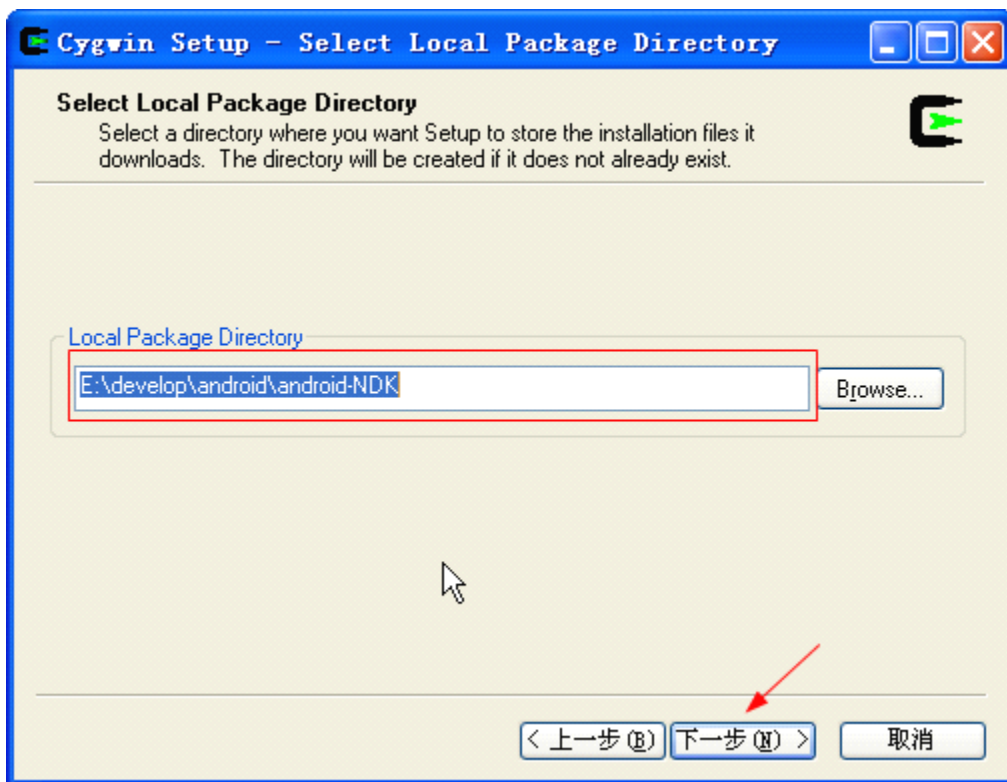


本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

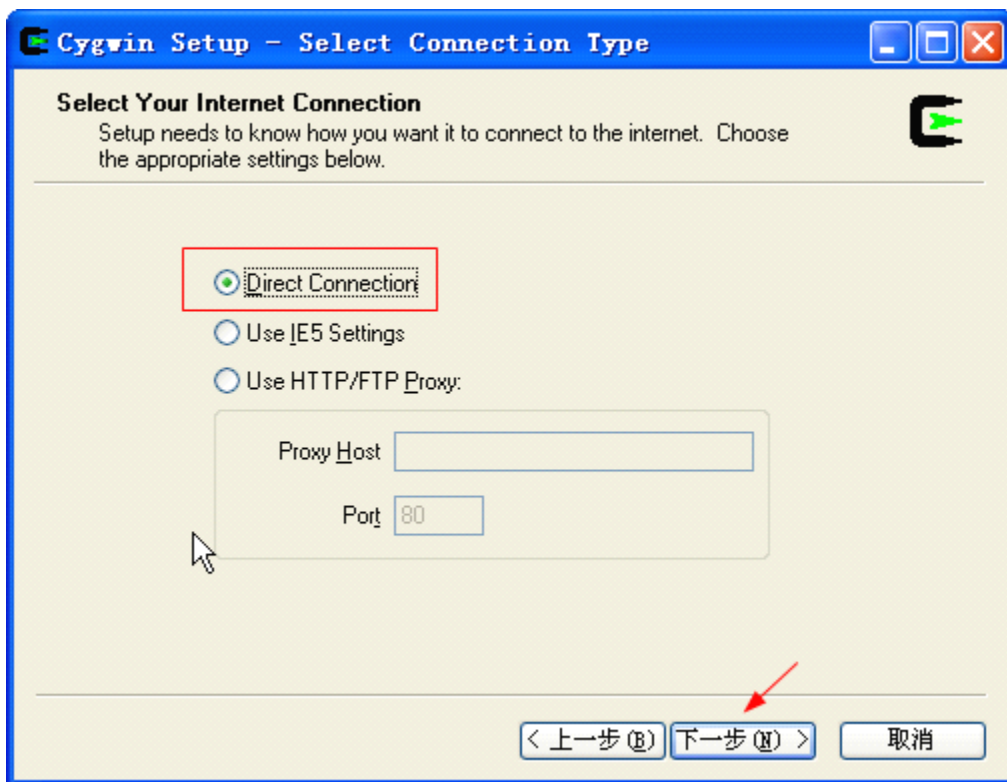
4: 选择安装路径



5.选择下载文件存放的目录

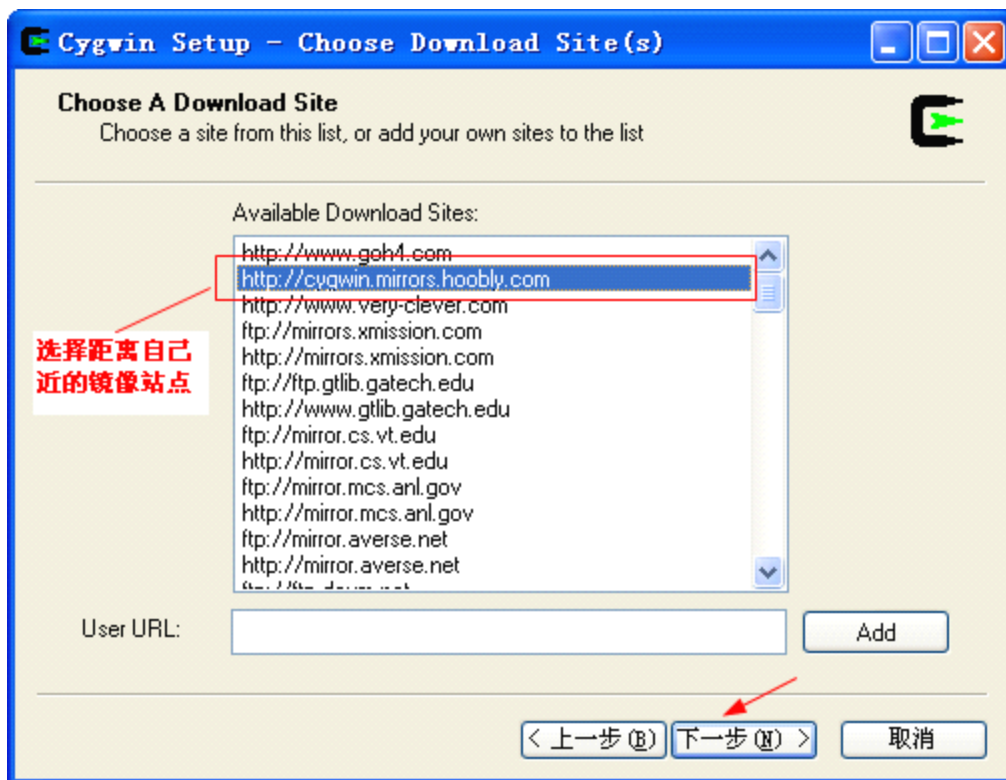


6. 选择网络连接方式

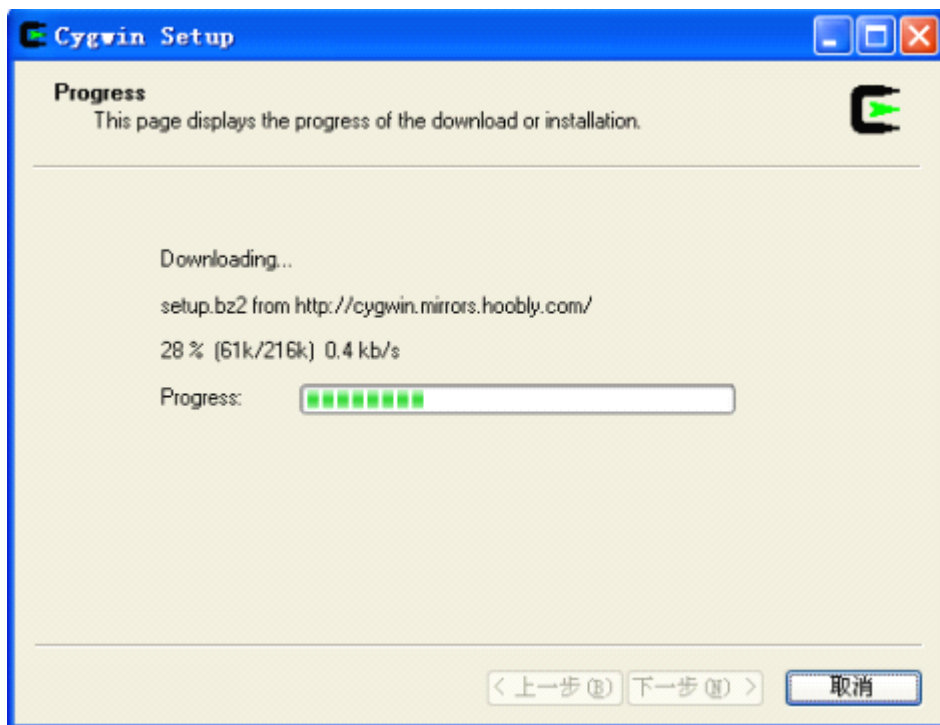


本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！

7. 选择下载镜像站点

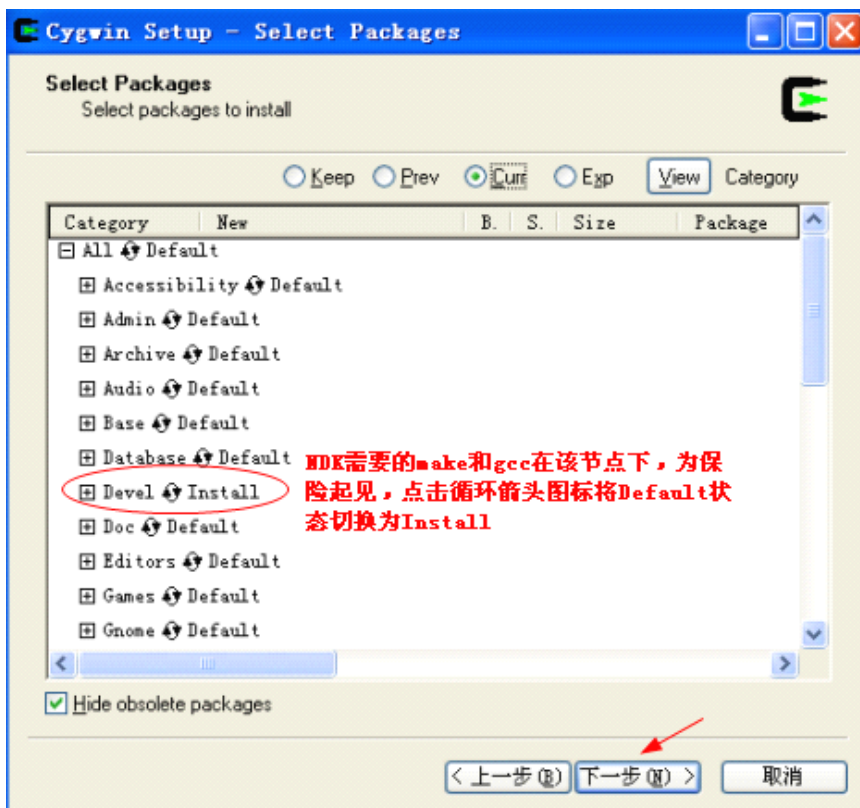


8. 开始下载安装

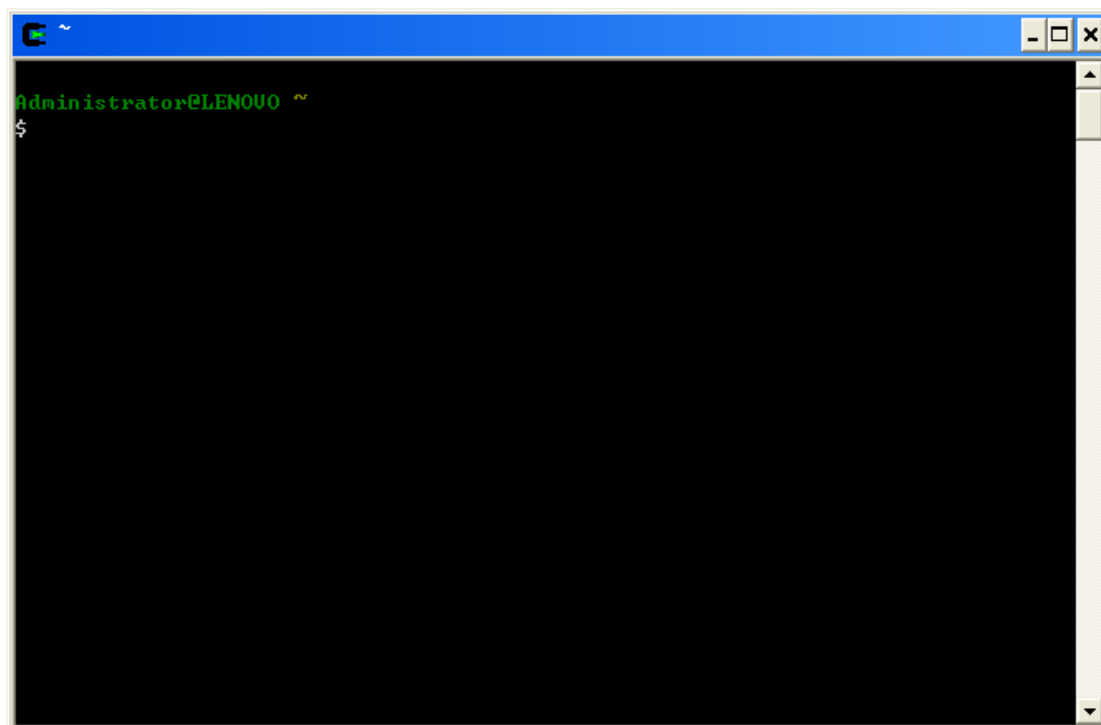


本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

9. 选择安装项

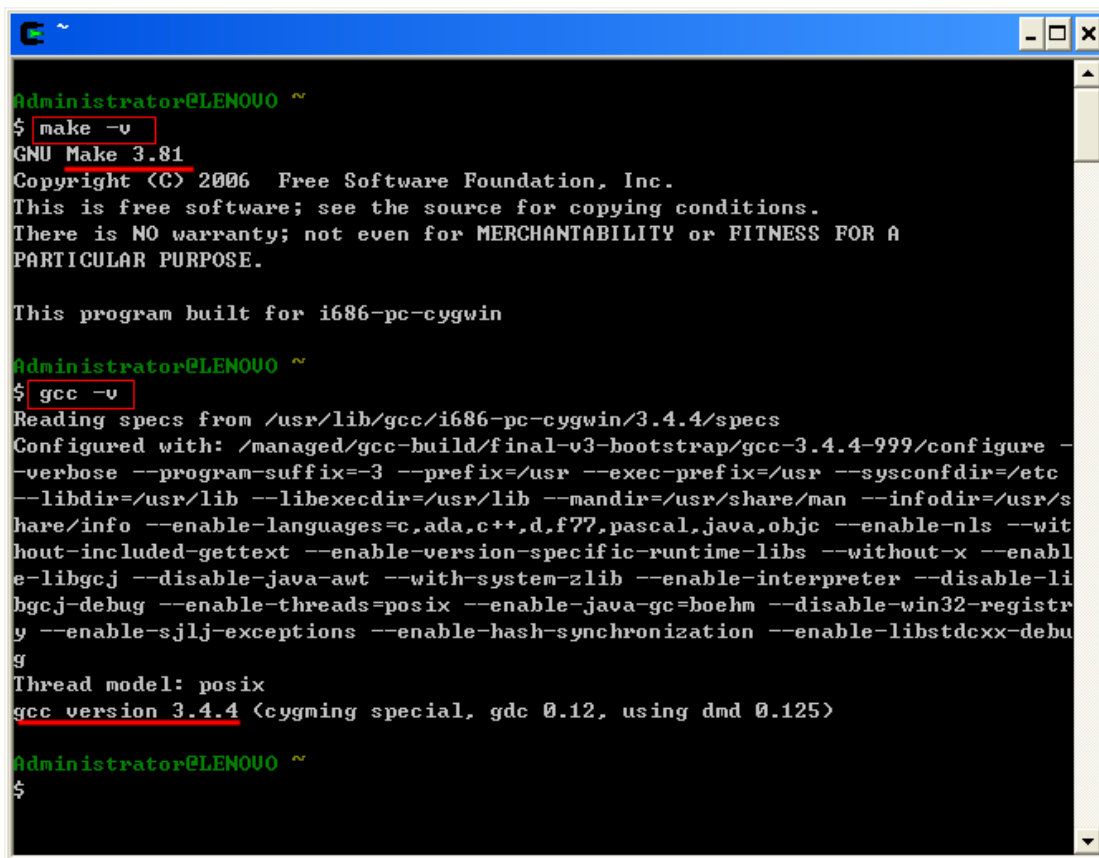


10. 启动 Cygwin,测试安装



本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！

输入命令检查



```
Administrator@LENOVO ~  
$ make -v  
GNU Make 3.81  
Copyright (C) 2006 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions.  
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A  
PARTICULAR PURPOSE.  
  
This program built for i686-pc-cygwin  
  
Administrator@LENOVO ~  
$ gcc -v  
Reading specs from /usr/lib/gcc/i686-pc-cygwin/3.4.4/specs  
Configured with: /managed/gcc-build/final-v3-bootstrap/gcc-3.4.4-999/configure --  
verbose --program-suffix=-3 --prefix=/usr --exec-prefix=/usr --sysconfdir=/etc  
--libdir=/usr/lib --libexecdir=/usr/lib --mandir=/usr/share/man --infodir=/usr/s  
hare/info --enable-languages=c,ada,c++,d,f77,pascal,java,objc --enable-nls --wit  
hout-included-gettext --enable-version-specific-runtime-libs --without-x --enabl  
e-libgcj --disable-java-awt --with-system-zlib --enable-interpreter --disable-li  
bgcj-debug --enable-threads=posix --enable-java-gc=boehm --disable-win32-registr  
y --enable-sjlj-exceptions --enable-hash-synchronization --enable-libstdcxx-debu  
g  
Thread model: posix  
gcc version 3.4.4 (cygming special, gdc 0.12, using dmd 0.125)  
  
Administrator@LENOVO ~  
$
```

出现以上版本信息则表明 make 和 gcc 已经安装成功。

11.赶紧用 NDK 来编译简单例子

A) .设置环境变量

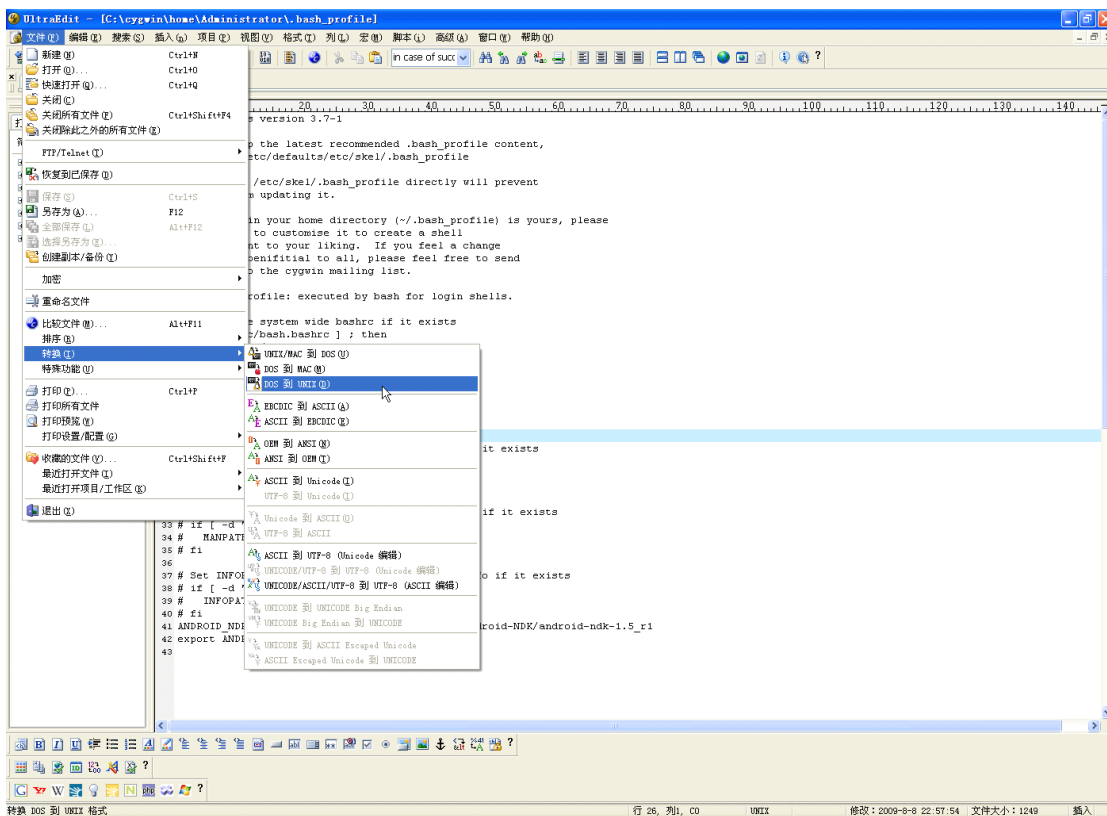
先找到安装路径下的 .bash_profile 文件, 我的环境位于: C:\cygwin\home\Administrator。用 UltraEdit 打开, 添加以下两句:

```
ANDROID_NDK_ROOT=/cygdrive/e/develop/android/android-NDK/android-ndk-1.5_r1  
  
export ANDROID_NDK_ROOT
```

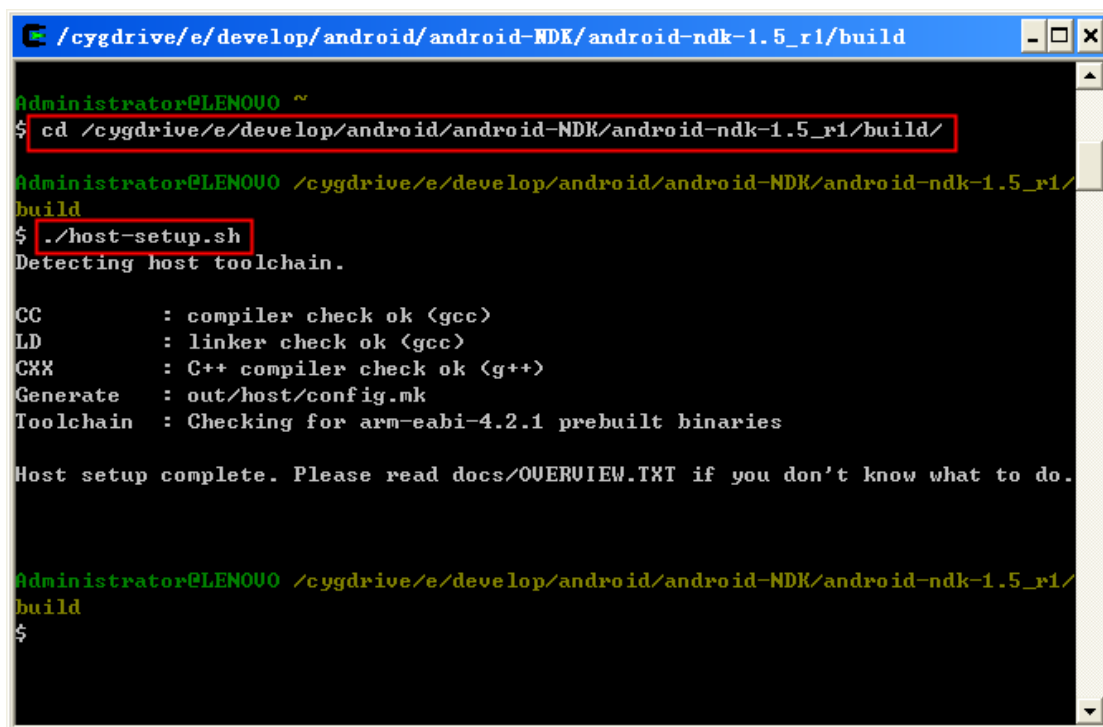
其中前一句需要修改为你自己的路径, 保存后, 重新启动 Cygwin。

注意: 这里如果是中文 OS 使用记事本或者写字板打开编辑后, 重新启动 Cygwin 会报错。可以按照下图方式转换:

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!



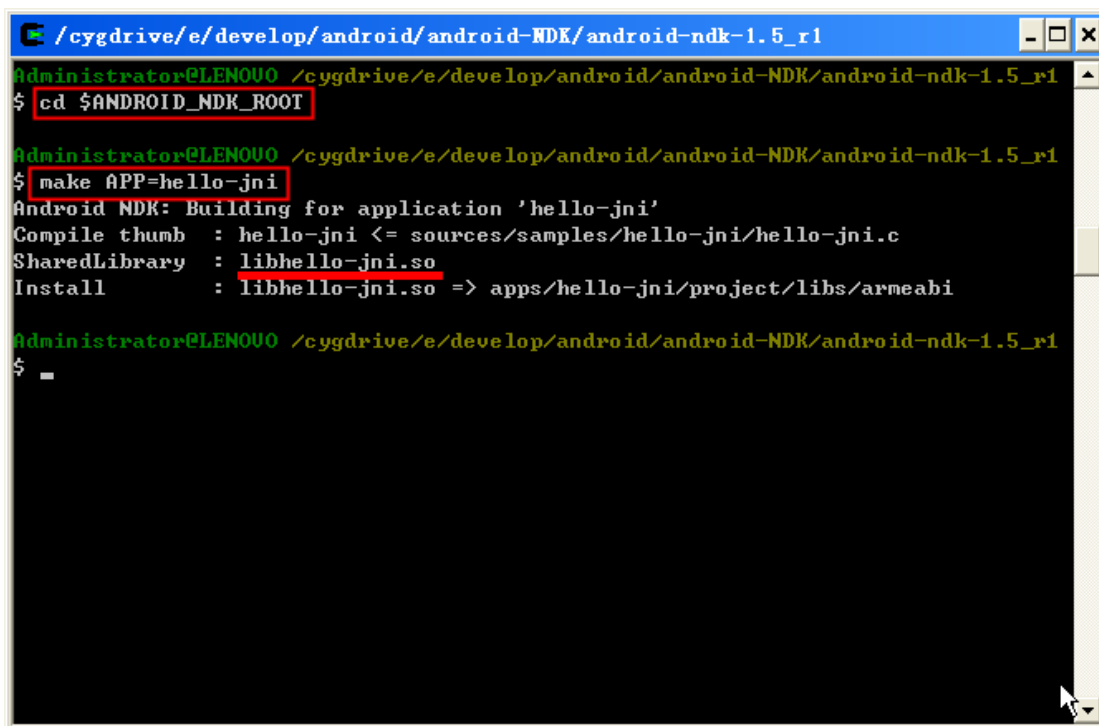
B) .配置 NDK



C).编译例子

本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！

先拷贝 build/out 下的 host 目录以及其下的 config.mk 文件到 NDK 根目录的 out 目录下。然后参照下图编译 hello-jni 例子。



```
/cygdrive/e/develop/android/android-NDK/android-ndk-1.5_r1
Administrator@LENOVO /cygdrive/e/develop/android/android-NDK/android-ndk-1.5_r1
$ cd $ANDROID_NDK_ROOT
Administrator@LENOVO /cygdrive/e/develop/android/android-NDK/android-ndk-1.5_r1
$ make APP=hello-jni
Android NDK: Building for application 'hello-jni'
Compile thumb : hello-jni <= sources/samples/hello-jni/hello-jni.c
SharedLibrary : libhello-jni.so
Install : libhello-jni.so => apps/hello-jni/project/libs/armeabi
Administrator@LENOVO /cygdrive/e/develop/android/android-NDK/android-ndk-1.5_r1
$
```

编辑完成可以看到 NDK 根目录下的 out 目录多出了 apps 子目录, 其他有编译好的共享文件库。

3.Ubuntu android NDK 配置与开发

作者: 情话 love 发表于 @ 2009 年 08 月 18 日

摘自: <http://www.eoeandroid.com/viewthread.php?tid=2406&highlight=ndk>

3.1 准备工作:

先下载 linux 版本的 NDK:

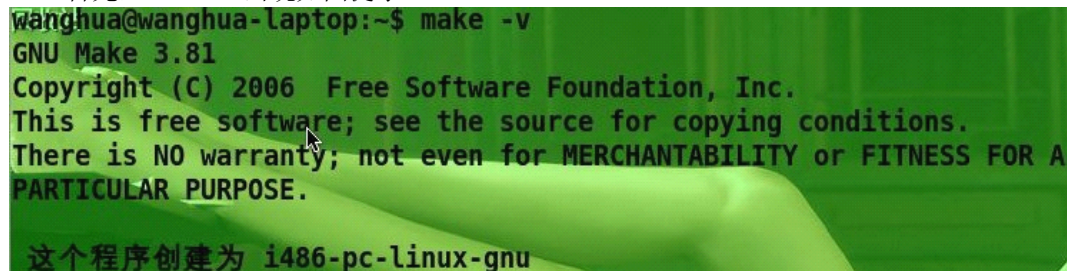
http://developer.android.com/intl/zh-CN/sdk/ndk/1.5_r1/index.html

下载完, 解压, 记住自己的解压目录, 为了下面方便和避免错误, 建议先进入解压的 android-ndk-1.5_r1 目录中

3.2 下面开始 NDK 的配置之旅

第一步: make -v 和 gcc -v 检测

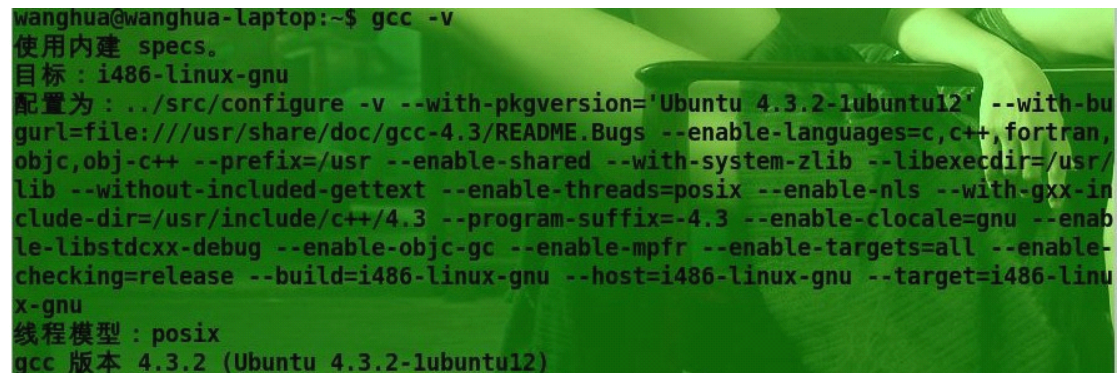
首先, make -v 出现如图提示。



```
wanghua@wanghua-laptop:~$ make -v
GNU Make 3.81
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

这个程序创建为 i486-pc-linux-gnu
```

然后, gcc -v 如图提示:



```
wanghua@wanghua-laptop:~$ gcc -v
使用内建 specs。
目标 : i486-linux-gnu
配置为: ../src/configure -v --with-pkgversion='Ubuntu 4.3.2-1ubuntu12' --with-bu
gurl=file:///usr/share/doc/gcc-4.3/README.Bugs --enable-languages=c,c++,fortran,
objc,obj-c++ --prefix=/usr --enable-shared --with-system-zlib --libexecdir=/usr/
lib --without-included-gettext --enable-threads=posix --enable-nls --with-gxx-in
clude-dir=/usr/include/c++/4.3 --program-suffix=-4.3 --enable-clocale=gnu --enab
le-libstdcxx-debug --enable-objc-gc --enable-mpfr --enable-targets=all --enable-
checking=release --build=i486-linux-gnu --host=i486-linux-gnu --target=i486-linu
x-gnu
线程模型 : posix
gcc 版本 4.3.2 (Ubuntu 4.3.2-1ubuntu12)
```

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

第二步: 检测没有错误, 输入命令: `./build/host-setup.sh`

如果出现如图错误:

```
wanghua@wanghua-laptop:~/Android/android-ndk-1.5_r1$ ./build/host-setup.sh
./build/host-setup.sh: 23: source: not found
Detecting host toolchain.

./build/host-setup.sh: 57: force_32bit_binaries: not found
./build/host-setup.sh: 58: setup_toolchain: not found
./build/host-setup.sh: 60: cannot create : Directory nonexistent
Can't create directory for host config file: out/host
```

第三步: `vim ./build/host-setup.sh`

编辑脚本, 按下字母 `i` 进行修改, 修改 `#!/bin/sh` 为 `#!/bin/bash`

修改完毕: ESC 然后按下: `wq` 保存退出。

继续进行: `./build/host-setup.sh`

如图所示, 运行成功!

```
wanghua@wanghua-laptop:~/Android/android-ndk-1.5_r1$ ./build/host-setup.sh
Detecting host toolchain.

CC      : compiler check ok (gcc)
LD      : linker check ok (gcc)
CXX     : C++ compiler check ok (g++)
Generate : out/host/config.mk
Toolchain : Checking for arm-eabi-4.2.1 prebuilt binaries

Host setup complete. Please read docs/OVERVIEW.TXT if you don't know what to do.
```

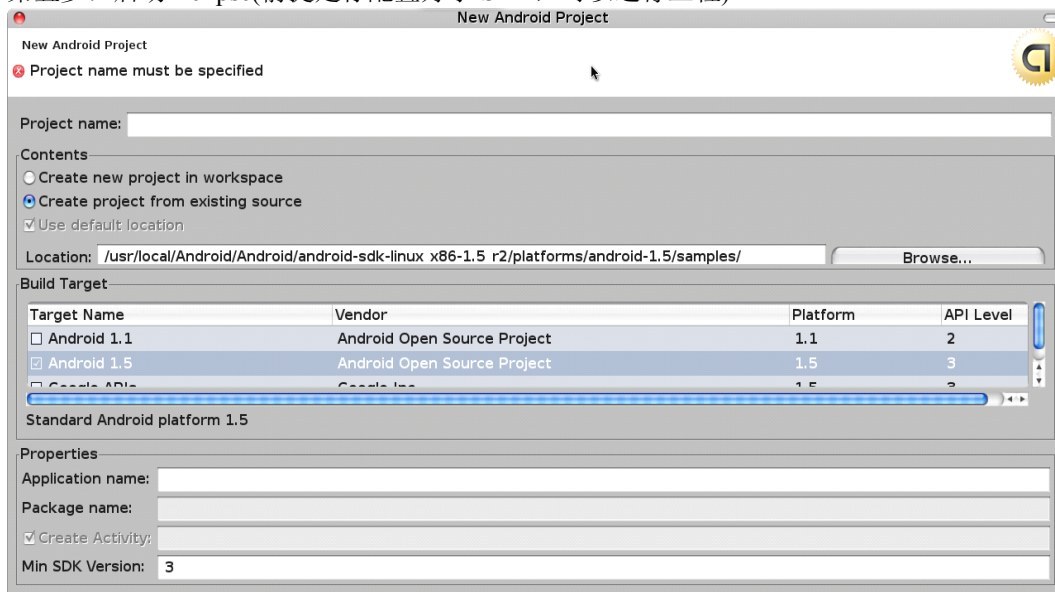
第四步:

```
wanghua@wanghua-laptop:~/Android/android-ndk-1.5_r1$ make APP=hello-jni
Android NDK: Building for application 'hello-jni'
Compile thumb : hello-jni <= sources/samples/hello-jni/hello-jni.c
SharedLibrary : libhello-jni.so
Install       : libhello-jni.so => apps/hello-jni/project/libs/armeabi
```

接着编译 `samples` 里的例子: `make APP=hello-jni`

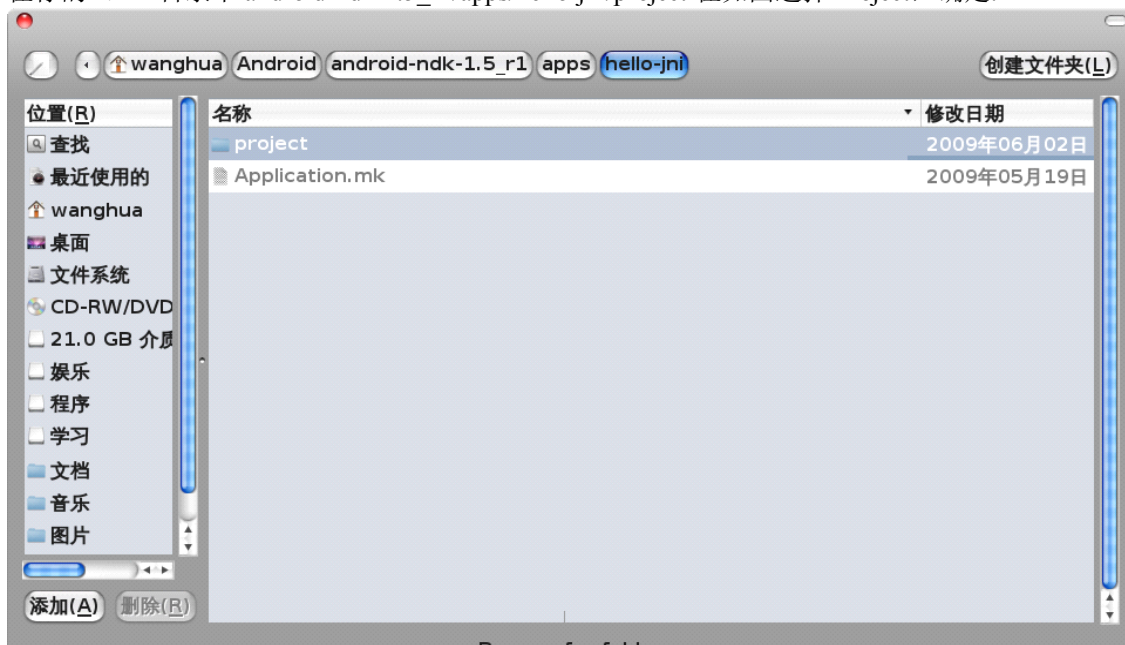
如上图命令提示: `libhello-jni.so` 会生成在 `apps/hello-jni/project/libs/armeabi`

第五步: 启动 Eclipse(前提是你配置好了 SDK, 可以运行工程)



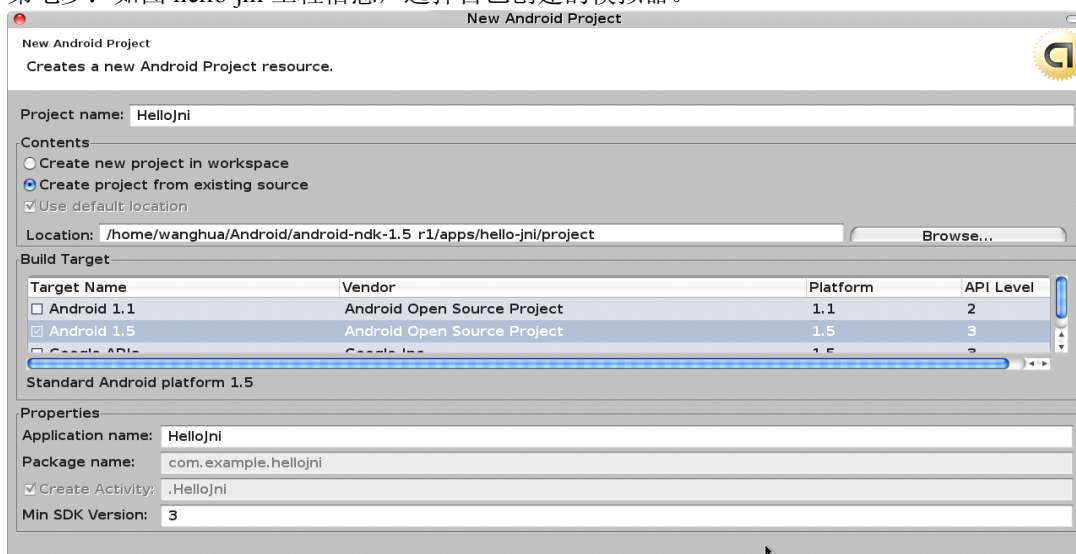
第六步: 选择刚刚编译过的 hello-jni 例子

在你的 NDK 目录下 android-ndk-1.5_r1/apps/hello-jni/project 在如图选择 Project, 确定.

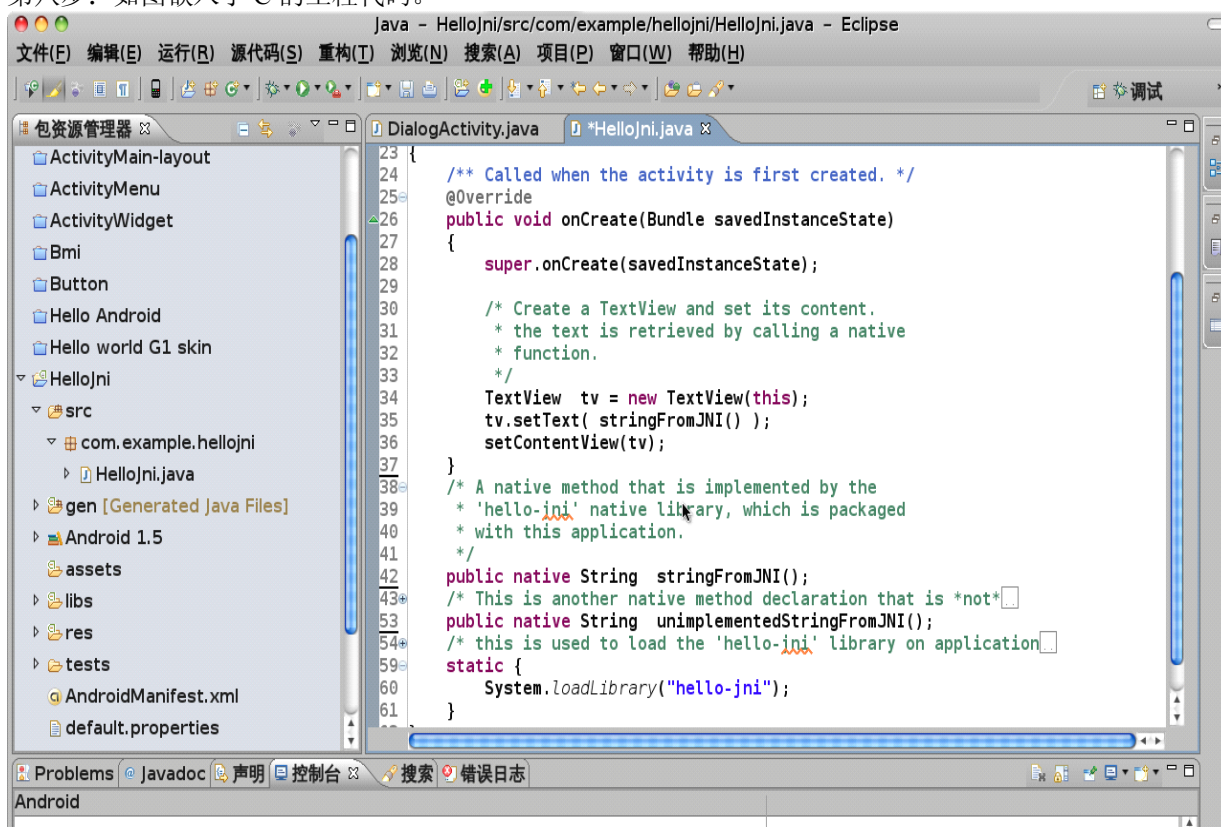


本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

第七步: 如图 hello-jni 工程信息, 选择自己创建的模拟器。



第八步: 如图嵌入了 C 的工程代码。



第九步: 生成。



至此, ubuntu android NDK 配置成功! 现在就可以嵌入 C 代码的程序了

NDK 自带文档翻译与其它

4. Android1.5 NDK Release 1 中文说明文档

作者: Luggie

日期: 2009-6-27 版本: 1.0

Android NDK 是配合 Android SDK 的工具, 用来编译应用的原生代码。他只能与 Android SDK 配合使用, 因此下载 NDK 之前请先安装 Android1.5 SDK。

4.1 Android NDK 是什么?

Android NDK 是编译嵌入在 Android 应用中的原生代码的工具。

Android 应用运行在 Dalvik 虚拟机上。NDK 允许开发者用原生代码 (C 或 C++) 实现应用的一部分。这将给某些应用带来好处, 这种方式可重用代码, 而且在某些情况下可加快运行速度。

4.2 NDK 提供了:

提供了将 C 和 C++ 源代码生成原生代码库的工具和文件

提供了将原生库嵌入 apk 文件的方法

提供了兼容 1.5 版本以上的原生系统头文件和库

提供了文档, 示例和指引

Android 1.5 NDK 支持 ARMv5TE 机器指令集, 提供稳定的 C 库头文件 (C library), JNI 接口和其他的库。

NDK 并不适用于大部分应用。作为开发者, 你应该衡量它的优缺点; 很明显, 用原生代码并不能自动提升性能, 却增加了应用的复杂度。NDK 适合用于独立的、占用内存少、占用较多 CPU 资源的处理, 例如, 信号处理、物理仿真等等。简单的用 C 重写代码一般不会带来性能的大幅提升。不过, NDK 提供了重用现有 C/C++ 的有效途径。

注意, NDK 并不能让你开发纯原生应用。Android 的主要运行时仍然是 Dalvik 虚拟机。

4.3 NDK 的内容

开发工具

NDK 包含了一套交叉编译工具 (编译, linkers 等), 它可生成 Linux, OS X 和 windows (用 Cygwin) 上的原生 ARM 的二进制码。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

他提供了一套原生 API 的系统头文件（兼容今后版本）：

libc (C library) headers

libm (math library) headers

JNI interface headers

libz (Zlib compression) headers

liblog (Android logging) header

A Minimal set of headers for C++ support

NDK 也提供了编译系统，你可以快速编译源代码，而不用处理 toolchain/platform/CPU/ABI 的细节问题。你只需创建很短的编译文件，用来说明哪些源代码需要编译以及编译到哪个目标 Android 应用，编译工具将根据此文件编译并将生成的共享库放到对应的应用下。

重要：除上述的库之外，Android1.5 的原生系统库在以后的版本有可能改变。因此，你的应用应该只适用 NDK 提供的库。

4.4 文档

NDK 包包含了一套文档，描述了 NDK 的功能和创建 Android 应用共享库的使用方法。在这个版本中，文档包含在下载了的 NDK 包中，路径：<ndk>/docs/。文档包含如下文件：

INSTALL.TXT — 说明如何安装、配置 NDK

OVERVIEW.TXT — 概要介绍 NDK 的功能和用法

ANDROID-MK.TXT — 说明 Android.mk 的用法, Android.mk 用来指定需要编译的源代码

APPLICATION-MK.TXT — 说明 Application.mk 的用法 file, Application.mk 用来指定目标应用

HOWTO.TXT — 介绍与 NDK 开发相关的任务.

SYSTEM-ISSUES.TXT — 如果用 NDK 开发，你需要了解 Android 系统映像相关的知识

STABLE-APIS.TXT — NDK 头文件列表

另外，还包含了“bionic”C 库的详细信息，如果用 NDK 开发，你应该了解这些信息。路径：<ndk>/docs/system/libc/：

OVERVIEW.TXT—介绍“bionic”C 库及其特性

本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！

4.5 示例应用

NDK 包含了两个 Android 应用, 用来说明如何在 Android 应用中用原生代码。

hello-jni—该示例调用共享库 (shared library) 的原生方法获取一个字符串, 并显示在应用的界面上。

Two-libs—该示例动态加载一个共享库, 并调用其中的方法, 该方法由一个导入到共享库的静态库实现。

4.6 系统和软件要求

本节说明使用 Android NDK 的系统和软件要求, 以及用 NDK 开发的的平台兼容性问题。

Android SDK

需要完整安装 Android SDK

Android SDK 版本为 1.5 或以上版本

支持的操作系统

Windows XP (32-bit) or Vista (32- or 64-bit)

Mac OS X 10.4.8 or later (x86 only)

Linux (32- or 64-bit, tested on Linux Ubuntu Dapper Drake)

需要的开发工具

在所有操作系统上, 都需要安装 GNU Make 3.81 或更高版本。早期的版本也可能可用, 但未经测试验证。

在 windows 上需要安装较新版本 Cygwin, 包括 gmake 和 gcc 包。

Android 平台兼容性

NDK 生成的原生库只能在 Android 1.5 或更高版本上运行。这是因为相关的 toolchain 和 ABI 发生了变化, 使得原生库与 1.0 和 1.1 的系统映像不兼容。

由于上述原因, 你用 NDK 开发的原生库对应的应用只能运行在 Android 1.5 或更高版本上。为了确保兼容, NDK 开发的应用必须在其 manifest 文件中声明 <uses-library> 元素, 属性为 “android:minSdkVersion=“3””

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

4.7 安装 NDK

安装 NDK 比较简单, 涉及到解压下载到的 NDK 包和运行安装的脚本。

首先需要确认你安装了 Android SDK1.5 或更高版本, 且按需要升级了你的应用和环境。

接下来开始安装:

下载合适的 NDK 包

解压 NDK。解压后, NDK 文件放在 `android-ndk-<version>` 中。你可以重命名此目录也可移动到其他目录下。本文档中 NDK 根目录用 `<ndk>` 表示。

打开命令行窗口, 在 NDK 根目录下执行安装脚本。该脚本用来设置你的环境、生成以后需要的主机配置文件(编译生成共享库时会用到)。脚本路径如下:

```
<ndk>/build/host-setup.sh
```

如果脚本执行成功, 将显示 "Host setup complete."。如果执行失败, 将显示用法说明, 以便你解决问题。

这样便安装完毕, 可以可是使用 NDK 了。

4.8 开始使用 NDK

NDK 安装成功后, 可在 `<ndk>/docs/` 目录下查看相关文档。为了理解 NDK 的目的和用法, 可以看 OVERVIEW.TXT。

以下是 NDK 工具的一般用法:

你的 native 源代码应该放在 `<ndk>/sources/<my_src>/...` 下。也可以将源代码链接到其他目录。这些源代码并不是完全与某一个共享库或 Android 应用绑定在一起, 相反, 他们可通过不同的配置文件生成对应不同 Android 应用的共享库。

创建 `<ndk>/sources/<my_src>/Android.mk` 来指定要编译的源代码。

创建 `<ndk>/apps/<my_app>/Application.mk` 来指定对应的目标 Android 应用。这个文件将一个 Android SDK 应用工程与 `<ndk>/sources/` 中的共享库关联起来, 并指定了接收共享库的应用工程所在目录。

从 NDK 根目录运行 `make` 指令进行源代码编译:

```
$ make APP=<my_app>
```

编译工具将拷贝共享库到应用工程的相应目录。

最后, 用 SDK 工具编译你的 Android 应用。SDK 编译工具将把共享库打到 apk 包中。
本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

更详细的信息请查看 NDK 包中的文档。

4.9 示例使用

NDK 包含了两个例子应用, 通过示例说明在 Android 应用中如何使用原生代码:

hello-jni—该示例调用共享库 (shared library) 的原生方法获取一个字符串, 并显示在应用的界面上。

Two-libs—该示例动态加载一个共享库, 并调用其中的方法, 该方法由一个导入到共享库的静态库实现。

每个示例, 都包含了一个 Android 应用工程、对应的 C 源代码以及必需的 Android.mk 和 Application.mk 文件。应用工程路径是 <ndk>/apps/<app_name>/project/, C 代码路径是 <ndk>/sources/samples/<library>/。安装好 NDK 后, 你可以在 NDK 根目录用以下命令编译出示例的共享库:

```
$ make APP=hello-jni — compiles <ndk>/sources/samples/hello-jni/hello-jni.c and outputs a shared library to <ndk>/apps/hello-jni/project/libs/armeabi/libhello-jni.so.
```

```
$ make APP=two-libs — compiles <ndk>/sources/samples/two-libs/second.c and first.c and outputs a shared library to <ndk>/apps/two-libs/project/libs/armeabi/libtwolib-second.so.
```

然后, 编译对应的 Android 应用。

4.10 编者注:

这篇文章应该算是相应的好, 可惜联系不到作者, 还希望能向他索要更多好文章, 如此的牛人不能认识真是可惜!

5.Android NDK 概述

译者: 曹立明

原文: Android_NDK_Overview

介绍:

Android NDK 是一套工具, 允许 Android 应用开发者嵌入从 C、C++源代码文件编译来的本地机器代码到各自的应用软件包中。

重要:

Android NDK 只能被用于使用该平台的 Cupcake (1.5)或是更新发布的系统映像。

特别指出 1.0 和 1.1 系统映像不支持 NDK, 这是由于在 1.5 发布中对 toolchain 和相关 ABI 做了改变。

5.1 Android NDK 的目标:

Android 虚拟机允许你的应用程序源代码通过 JNI 调用实现本地代码的方法。简而言之, 这意味着:

- 应用程序源代码将用 'native' 关键字声明一个或多个方法来表明这些方法是通过本地代码来实现的。例如:

```
native byte[] loadFile(String filePath);
```

- 你必须提供一个包含这些方法实现的本地共享库, 该共享库将会打包到你的应用程序的 .apk 文件中。

这个共享库需要根据标准的 Unix 的公约来命名, 像 lib<something>.so, 并且应包含标准 JNI 切入点 (以下有更详细的介绍)。例如:

```
libFileLoader.so
```

- 你的应用程序应必须明确地加载这个库。例如, 在应用程序的开始加载它, 只需将以下内容添加到应用程序的源代码中:

```
static {  
    System.loadLibrary("FileLoader");  
}
```

请注意, 在这里, 您不需要使用 'lib' 前缀和 '.so' 后缀。

Android NDK 是对 Android SDK 的补充, 帮助你:

- 生成 JNI 兼容共享库, 这些库可以运行在 ARM 处理器上的 Android 1.5 平台 (及更高版本)。
- 拷贝生成共享库到应用程序工程路径的正确位置, 那么它们将被自动添加到你的最后 (和已签名) 的 .apk 文件中。
- 在以后的 NDK 修订中, 我们打算提供工具通过一个远程的 gdb 连接来帮助调试你的本地代码以及更多的源/符号信息。

此外, Android NDK 规定:

- 一套交叉工具链 (编译器, 连接器等), 可以产生本地的 ARM 二进制文件在 Linux, OS X 和 Windows 操作系统运用 (使用 Cygwin)

- 一套系统头符合于 Android 平台支持的稳定本地 API 列表。这相当于定义都保证支持所有更高版本的平台。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

重要的:

请记住, 在以后的更新和发布的平台, 大多数在 Android 系统映像的本地系统库并没有固定, 可能会被彻底地改变, 甚至删除。

- 一个编译系统, 它允许开发者编写很短的编译文件来描述哪些代码是需要编译的, 并且怎么样编译。该编译系统处理所有的多如毛发的工具/平台/处理器/ABI 的细节。

此外, 更新后的 NDK, 可以添加支持更多的工具, 平台, 系统接口, 而无需改变开发编译文件 (以下有更详细的介绍)。

5.2 不是 Android NDK 的目标:

NDK 不是用来编写通用的运行在 Android 设备上的本地代码。特别指出, 你的应用程序仍然应该用 JAVA 语言来编写, 处理 Android 系统事件, 应避免弹出"应用程序没有响应"对话框或处理 Android 应用程序的生命周期。

但是请注意, NDK 适于有可能采用本地代码编写一个复杂的应用程序与一个小的用于启动/停止的"应用程序封装"。

强烈建议很好的理解 JNI, 因为许多业务在这种环境下需要的具体动作来自于开发者, 而不需要特定的本地代码。这些包括:

- 不能通过本地指针直接访问 VM 对象的内容。例如, 你不能安全地获取一个指针对应一个字符串对象的 16 位字符数组在循环中的迭代。

- 需要明确提到管理本地代码时, 要保持 VM 对象同 JNI 调用间的句柄。

NDK 仅提供系统头, 它针对 Android 平台支持的一套非常受限制的本地 APIs 和库。虽然一个典型 Android 系统映像包含许多本地共享库, 这些共享库应该考虑

执行的细节, 这些细节可能在已经发布的平台和更新之间被彻底改变。

如果一个 Android 系统库不是 NDK 头直接支持的, 那么应用程序不应仅是可用而依赖它, 否则在下一个空中系统更新多种设备后他们将不能使用。

选定的系统库将逐步地添加到一套稳定 NDK 的 API 中。

5.3 NDK 开发实践:

这里有一个非常粗略的概述指导你怎样使用 Android NDK 开发本地代码:

1/ 运行 build/host-setup.sh 配置 NDK

2/ 放置你的代码到目录 sources/<mysrc>

3/ 编写 sources/<mysrc>/Android.mk 文件描述你的代码给 NDK 编译系统

4/ 编写 apps/<myapp>/Application.mk 文件描述你需要 NDK 编译系统编译的应用程序和本地代码

5/ 在 NDK 顶级目录中运行"make app=<myapp>"编译你的本地代码。

译者注: 5/是在 Cygwin 中执行。

如果编译成功, 最后一步将拷贝共享库到你的应用程序工程根目录。然后你就可以通过正常手段生成最后的.apk 文件。

现在, 一些更为详细的信息:

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

5.4 配置 NDK:

按照 docs/INSTALL.TXT 文件中描述安装 NDK 后, 你应该运行'build/host-setup.sh'脚本来配置 NDK。

这个脚本是用来探测你的主机系统并确认一些先决条件。
然后将生成一个配置文件(如: out/host/config-host.mk),它将在随后的 NDK 编译过程中使用到。

在某些情况下, 这可能指示你为开发平台去下载一个包含预编译工具的二进制压缩文件, 然后将它解压缩到 NDK 根目录。
这个提示应包含足够的信息让你做到这一点。

如果你忘记了这一步, 那么在尝试使用 NDK 编译的时候将产生错误信息, 这些信息将告诉你该怎么做。

5.5 放置 C 和 C++代码:

NDK 编译系统预定你的资源在顶级目录'sources'下可见。你应首先创建如下目录:

\$NDK/sources/<mysrc>/

你可以按照你想要的目录结构, 灵活自由组织'sources'下的内容, 这个目录名和结构将不影响最后生成的应用程序包, 因此你不必使用不真实的独特的名字,

如: com.<mycompany>.<myproject>作为应用程序包名。

针对记录, NDK 来自'sources/samples'目录, 该目录自身包含多个简单模块的子目录。

请注意, C 和 C++源是支持的。默认 C++文件扩展名是 NDK 支持的'.cpp',但是其他扩展名也能被很好地处理(详见 docs/ANDROID-MK.TXT 文件)。

可以存储你的源文件在不同的位置, 只要你创建\$NDK/sources/<mysrc>为一个符号链接。进行适当的操作, NDK 编译系统必须能够找到来自\$NDK/sources 下的源文件和编译脚本。

5.6 Android.mk 编译脚本:

Android.mk 文件是一个小编译脚本, 你编写来描述源文件给 NDK 编译系统。它们的语法在文件 docs/ANDROID-MK.TXT 中有详细描述。

简而言之, NDK 组织源文件进入"modules", 这里每个模块可以是下列之一:

- 一个静态库
- 一个共享库

你能定义几个模块在一个单独的 Android.mk 文件中, 或者你能编写几个 Android.mk 文件, 每个文件定义一个单独的模块。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

全部 **Android.mk** 文件在其他编译发生前被编译系统解析。

请注意, 一个单独的 **Android.mk** 可以被编译系统解析多次, 因此不要认为某些变量在它们中没有定义。

By default, the NDK will look for all files that match the following:

默认情况下, NDK 将寻找匹配下列的所有文件:

```
$NDK/sources/*/Android.mk
```

如果你想要定义 **Android.mk** 到子目录, 你应明确地在顶级 **Android.mk** 文件中包含它们。这有一个帮助函数可以做到, 如,使用:

```
include $(call all-subdir-makefiles)
```

这将包含全部在当前编译文件路径下子目录中的全部 **Android.mk** 文件。

5.7 编写 **Application.mk** 编译文件:

虽然一个 **Android.mk** 文件描述模块给编译系统了, 但是, 你还需要编写一个 **Application.mk** 文件描述应用程序以及应用程序需要的模块。

该文件必须位于:

```
$NDK/apps/<myapp>/Application.mk
```

凡是<myapp>是一个应用程序的简短描述名称, 将被用于调用 NDK 编译 (并且不会进入最后 APK 文件)。

该文件用于提供给 NDK 编译下列内容:

- Android 应用程序工程路径的位置。
- 应用程序需要的 NDK 模块的列表。
这应该是一个真正的'共享库'模块的列表。
- 可选信息, 如判断你要发布或者是调整编译, 指定 C 或 C++编译标志等。
- 计划: 具体平台/处理器的清单, 你想要的明确目标 (当前仅支持一个)。

Application.mk 文件的语法非常简单并且在 docs/APPLICATION-MK.TXT 文件中有描述。

你可以定义几个 **Application.mk** 文件建立同一个应用程序的不同编译, 例如:

```
$NDK/apps/release/Application.mk
$NDK/apps/debug/Application.mk
```

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

5.8 调用 NDK 编译系统:

在命令行, 进入 NDK 顶级目录, 然后使用以下命令调用编译系统:

```
make APP=<myapp>
```

这里的‘make’是指 GNU Make, 并且<myapp>是一个‘\$NDK/apps/’中子目录的名称。

这将试图建立相关的全部模块选择, Application.mk 文件列出最终共享库, 如果编译成功, 将拷贝共享库到你的应用程序工程根目录(请注意: unstripped 版本可能是为保留调试目的, 因此没有必要拷贝 unstripped 二进制到设备)。

译者注: unstripped 不知道该如何翻译, 故保留。

5.9 调试支持:

调试本地代码采用 NDK 的初始版本仍然很粗略。

请注意: 我们计划在 NDK 的更新中让调试变得更简单, 这一切无需改变你的源, Android.mk 和 Application.mk 文件。

6.Android.mk 文件语法详解

译者: 游利卡

<http://www.cnblogs.com/pcedb0189/>

注: 6 和 7 这两篇文章都是游利卡第一次尝试翻译 C 编程的文章, 可能有一些地方不是很准确, 希望大家海涵。如果觉得有不准的地方敬请参阅原文。

这篇文档描述了 Android.mk 文件语法, 以及如何将 C 和 C++ 的源文件写进 Android NDK(第一句话, 我不知道我说的是否准确)。

为了能让你更清楚得了解一下的内容, 我们这里假设你已经阅读了 OVERVIEW.TXT 这个文件。

概览:

简而言之, NDK 组织源文件进入 "modules", 这里每个模块可以是下列之一:

- 这个文件只是 GNU 编译文件的很小一部分, 他会在编译系统中被一次或者多次解析。正因为如此, 你需要尽可能得简化这个文件, 并且保证编译时还有任何其他部分没有定义。

- 这个文件的语法允许你将你的源文件写进你的 "模块"。一个模块是如下的文件

- 一个静态库

- 一个动态链接库

只有动态链接库被安装进你的应用程序包里后, 静态库才能被动态链接库所使用。

你可以在每一个 Android.mk 文件中定义一个或多个模块, 你也可以在不同的模块中使用相同的库。

- 编译系统会为你处理很多细节。举个例子, 你不需要列出头文件以及产生在 Android.mk 文件中的详细属性。NDK 编译系统自动帮你计算。

这同样意味着, 当有新的 NDK 版本发布后, 你可以直接从新的工具链和平台上得到支持, 而不用动手去修改你的 Android.mk 文件

注意: 下这里的语法和分布 Android 源代码中的 Android.mk 文件非常像。但编译器执行他们的时候会有所不同, 这里有一个置顶好的国际设通用的决议, 它允许我们的应用开发者重复使用外部库的源码。

示例:

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

在我们描述这里的语法细节时, 让我们用一个简单的 `hello world` 的例子做示范, 下面是具体的文件:

```
sources/helloworld/helloworld.c
```

```
sources/helloworld/Android.mk
```

当'`helloworld.c`'作为一个简单的, JNI 的动态链接库的源文件时, 它会实例化一个本地的方法, 并返回"`hello world`"的字符串。

文件会返回这样的内容:

```
----- cut here -----
```

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

```
include $(CLEAR_VARS)
```

```
LOCAL_MODULE := helloworld
```

```
LOCAL_SRC_FILES := helloworld.c
```

```
include $(BUILD_SHARED_LIBRARY)
```

```
----- cut here -----
```

好, 现在我们这里解释一下几句:

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

一个 `Android.mk` 文件从定义 `LOCAL_PATH` 变量开始, 他位于开发树源文件中。在这个例子里, '`my-dir`'宏的功能由编译器提供, 被用来返回当前目录的地址(这里的当前目录里包括 `Android.mk` 这个文件本身)

```
include $(CLEAR_VARS)
```

`CLEAR_VARS` 这个变量由编译系统提供, 并且指明了一个 GNU `makefile` 文件, 这个功能会清理掉所有的 `LOCAL_XXX`.(例如 `LOCAL_MODULE`, `LOCAL_SRC_FILES`,

`LOCAL_STATIC_LIBRARIES`, 等), 除了 `LOCAL_PATH`。这句话是必须的, 因为如果所有的变量都是全局的, 所有的可控的编译文件都需要在一个单独的 GNU 中被解析并执行。

```
LOCAL_MODULE := helloworld
```

`LOCAL_MODULE` 变量必须被定义, 用来区分你的 `Android.mk` 中的每一个模块。文件名必须是唯一的, 不能有空格。注意这里编译器会为你自动加上一些前缀和后缀, 来保证文件是一致的。在这些语句中, 一个动态链接库模块被命名成'`foo`', 最后会生成是 '`libfoo.so`'。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

IMPORTANT NOTE:

如果你将你的模块命名成"libfoo",编译系统就不会给你加上前缀,但是也同样会生成 libfoo.so 这样的文件。为了支持 Android 平台源码中的 Android.mk 文件,你需要这样做。

```
LOCAL_SRC_FILES := helloworld.c
```

LOCAL_SRC_FILES 变量必须包含一个 C 和 C++源文件的列表,这些会被编译并聚合到一个模块中。注意一下这里并不需要列出头文件和被包含的文件,因为编译系统会自动为你计算相关的属性;源代码中的列表,会直接被传递给编译器,这样做对你来说比较好。

注意一下 C++默认文件的扩展名是'.cpp'. 这里可以通过定义一个

LOCAL_DEFAULT_CPP_EXTENSION 变量,来定义一个不同的 C 文件。不要忘记初始化前面的."点"哦(也就是说".cxx"可以正常工作,但是'cxx'不行)。

```
include $(BUILD_SHARED_LIBRARY)
```

BUILD_SHARED_LIBRARY 这个变量是由系统提供的,并且指定给 GNU Makefile 的脚本,他可以收集所有你定义的从'include \$(CLEAR_VARS)'的 LOCAL_XXX 变量,并且决定哪些要被编译,哪些应该做得更加准确。这里同样也有一个 BUILD_STATIC_LIBRARY 来生成一个静态的库。

在 sources/samples 目录下,还有更多复杂的例子,并且都是你可以找到的带有注释的 Android.mk 文件。

Reference:

这里有一个变量的列表,你可以在你的 Android.mk 中定义,也可以按照你自己的习惯来定义,总之 NDK 的编译系统将会为你存储这些变量名。

- names that begin with LOCAL_ (e.g. LOCAL_MODULE)
- names that begin with PRIVATE_, NDK_ or APP_ (used internally)
- lower-case names (used internally, e.g. 'my-dir')

如果你需要定义你自己的定义你自己的再一个 Android.mk 文件中的变量,我们建议使用 MY_ 的前缀,这里有一个零碎的示例:

----- cut here -----

```
MY_SOURCES := foo.c
```

```
ifneq ($(MY_CONFIG_BAR),)
```

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!


```
MY_SOURCES += bar.c

endif
```

```
LOCAL_SRC_FILES += $(MY_SOURCES)
```

```
----- cut here -----
```

那么, 我们就开始吧:

NDK-provided variables:

```
-----
```

这些 GNU 的变量都是由你的编译系统在你的 `Android.mk` 文件前被解析的。注意下面这些合适的环境下, NDK 可能会多次解析你的 `Android.mk` 文件, 每一次, 这些变量都会有不同的定义。

`CLEAR_VARS`

指向一个编译的脚本, 用来反向定义所有列在 "Module-description" 中的 `LOCAL_XXX` 的变量。你必须在你开始一个新的模块前包含这些脚本, 例如

```
include $(CLEAR_VARS)
```

`BUILD_SHARED_LIBRARY`

指向一个编译脚本, 能收集你所有模块中 `LOCAL_XXX` 的变量, 并且能决定如何编译一个在你源文件中共享出来的目标板。注意你必须在包含这些文件之前, 定义 `LOCAL_MODULE` 和 `LOCAL_SRC_FILES`, 例如

```
include $(BUILD_SHARED_LIBRARY)
```

注意这里会生成这样一个库文件 named `lib$(LOCAL_MODULE).so`

`BUILD_STATIC_LIBRARY`

不同的 `BUILD_SHARED_LIBRARY` 在使用时会被一个静态库所代替。静态库是没有办法复制到你的项目或包中的, 但是却可以被动态链接库调用(详细观察一下下面对这两个变量的描述 `LOCAL_STATIC_LIBRARIES` 和 `LOCAL_STATIC_WHOLE_LIBRARIES`)

```
include $(BUILD_STATIC_LIBRARY)
```

注意这里会生成这样一个文件 named `lib$(LOCAL_MODULE).a`

`TARGET_ARCH`

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

架构中 CPU 的名字已经被 Android 开源代码明确指出了。这里的 arm 包含了任何 ARM-独立结构的架构, 以及每个独立的 CPU 的版本。

TARGET_PLATFORM

Android 平台的名字在 Android.mk 文件中被解析, 从现在开始, 只支持"android-1.5"

TARGET_ARCH_ABI

CPU+ABI 的名字, 目前, 只有 Arm 平台被支持, 这也就意味着

目前只有 Arm v5 以及更高版本的 CPU, 并只支持'softfloat' 其他的 ABIS 将会未来的 NDK 版本中得到支持, 他们会有一个不同的名字, 注意所有基于 ARM 的 ABIs 都会有一个 "TARGET_ARCH"被定义给 arm, 但也有可能有不同的"TARGET_ARCH_ABI"

TARGET_ABI

平台目标板和 abi 的链接, 这里要定义\$(TARGET_PLATFORM)-\$(TARGET_ARCH_ABI), 他们都非常有用, 特别是当你想测试一下具体的系统镜像在一个真实的设备上的时候。

NDK-provided function macros:

下面的都是 GNU 编译出来的宏, 而且他们都必须使用'\$(call <function>)', 才能返回文字化的信息。

my-dir

返回现在 Android.mk 文件所在的目录, 这直接和 NDK 的编译系统相关联, 在 Android.mk 文件中顶一个 LOCAL_PATH 变量非常有用

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

all-subdir-makefiles

返回一个 Android.mk 文件所在位置的列表, 以及当前的 my-dir 的路径。举个例子, 考虑一下下面的检索信息:

hierarchy:

```
sources/foo/Android.mk
```

```
sources/foo/lib1/Android.mk
```

```
sources/foo/lib2/Android.mk
```

如果 sources/foo/Android.mk 中包含这单独的一行

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!


```
include $(call all-subdir-makefiles)
```

这里也会自动去收录 sources/foo/lib1/Android.mk 和 sources/foo/lib2/Android.mk

这个功能被用来提供深度的代码目录, 并将协助编译系统进行分层, 注意这里默认情况下, NDK 只能在 sources/*/Android.mk 中进行查找

```
this-makefile
```

返回当前 Makefile 的路径(也就是那个功能被调用了)

```
parent-makefile
```

返回 Makefile 的包含树。也就是包含 Makefile 当前的文件

```
grand-parent-makefile
```

```
Guess what...
```

```
Module-description variables:
```

```
-----
```

下面的变量也是用来描述一下你准备给编译系统的模块。首先必须在一个 'include \$(CLEAR_VARS)' 和一个 '\$(BUILD_XXXXX)', 做一个定义。写之前, 我们必须保证 \$(CLEAR_VARS) 是一个未定义或者清空了的脚本, 除非在描述中有更加详细的说明。

```
LOCAL_PATH
```

这个变量被用来给出当前的路径, 你必须在你的 Android.mk 文件一开始就定义, 他们可以这样来写

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

这个变量不能被 \$(CLEAR_VARS) 给清除, 所以每一个 Android.mk 文件中只需要一个就够了(除非你在一个文件中定义了多个模块)

```
LOCAL_MODULE
```

这里是你的模块名。每一个模块必须是特殊的, 并且不能包含空格。你必须在包括 \$(BUILD_XXXX) 前定义它。

模块名决定了我们生成的文件的名字, 例如 lib<foo>.so 文件是一个动态链接库中的名字为 <foo> 的模块。然后, 你只需要为编译器提供你的模块的正常的名字(例如 <foo> 就可以了)。(无论是 Android.mk 还是 Application.mk)

```
LOCAL_SRC_FILES
```

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

这里是源文件的列表, 那些会被你的模块编译的源文件列表。只要这些文件提交给编译器, 系统就能自动得为你计算这些属性值

注意这些源代码的名字都是和 LOCAL_PATH 有关的, 你可以这样来使用这些部分, 例如:

```
LOCAL_SRC_FILES := foo.c \
                toto/bar.c
```

注意:在你的编译的文件中, 最好使用 Unix 风格的反斜杠(/), Windows 风格的反斜杠不能被很好得处理。

LOCAL_CPP_EXTENSION

这里有一个可选的变量可以被 C++的扩展文件所支持, 默认的 C++的文件是".cpp"的后缀名, 但是你最好换一下

```
LOCAL_CPP_EXTENSION := .cxx
```

LOCAL_CFLAGS

当编译 C 的原文件的时候, 一个可以选择的编译器设置可以作为信号来发出。

详细说明一下额外添加的路径(跟 NDK 的路径相关的)、宏定义或者是其他的编译设置, 都非常有用

重要: 不要尝试在优化状态下/debug 状态下改变你的 Android.mk 文件, 在你的 Application.mk 中修改的具体信息可能会被自动得处理, 如果想让 NDK 生成有用的数据文件, 最好在 Debugging 状态下

LOCAL_CXXFLAGS

和 LOCAL_CFLAGS 一样的 C++源文件

LOCAL_CPPFLAGS

和 LOCAL_CFLAGS 一样的被用来做 C 和 C++的源文件

LOCAL_STATIC_LIBRARIES

静态库模块的列表(通过 BUILD_STATIC_LIBRARY 来编译), 这些只能在动态链接库的模块中才有意义。

LOCAL_SHARED_LIBRARIES

动态链接库的列表的模块需要运行时间。他们在连接时间及在生成文件中反馈信息都非常有必要。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

注意这个不能直接挂载这些被列出的模块到编译图中,也就是你只能添加这些到 Application.mk 中的需要他们的模块中去。

LOCAL_LDLIBS

当编译你的模块时,这里可能会有一些额外的链接会被用到。这些用"-l"的前缀,来传递具体的系统库的名字非常有用。比如说,下面的例子你可以先告诉链接来生成一个模块,链接到 /system/lib/libz.so at load time

```
LOCAL_LDLIBS := -lz
```

看一下 docs 中的 STABLE-APIS.TXT, 这里对系你能找到的系统库有个详细的说明。

LOCAL_ALLOW_UNDEFINED_SYMBOLS

默认情况下,未定义的参考会遇到"undefined symbol"这样的错误,这里可以帮助你搜集一下代码中的 bug。

然而,如果因为一些原因,你需要这个检查的功能时,请设置这个变量为"true"。但是注意反馈信息可能在运行时失效。

7.NDK doc 其余四篇文章译文

译者: 游利卡

<http://www.cnblogs.com/pcedb0189/>

注: 这四篇文章都是游利卡第一次尝试翻译 C 编程的文章, 可能有一些地方不是很准确, 希望大家海涵。如果觉得有不准的地方敬请参阅原文。

因为文章都比较短小, 所以放在一个大标题下列出。

7.1 Application.mk 文件语法详述

介绍:

这篇文档描述了 Application.mk 文件构建的语法, 及如何在你的应用中描述这些原生的模块。为了能让你更清楚的了解一下的内容, 我们这里假设你已经阅读了 OVERVIEW.TXT 这个文件。

我们希望读者能在读此篇文档时能阅读以下 docs/OVERVIEW.TXT 和 docs/ANDROID-MK.TXT

概览:

Application.mk 就是用来描述你的应用所需要的原声的"模块"(也就是静态库和动态链接库)。每一个 Application.mk 都必须放在应用目录下的子目录, 就像这样:

```
$NDK/apps/<myapp>/Application.mk
```

<myapp> 是一个 NDK 编译系统, 而描述你的应用的简短的名称(在最终生成的动态链接库中, 不需要包含这样的名称)。

作为 GNU Makefile 的一部分, Application.mk 必须要定义以下部分:

APP_MODULES

这个变量是强制性的, 并且会列出所有你的应用所需要的模块(通过 Android.mk 文件来即兴描述)他们必须有一定的空间独立性, 就像 Android.mk 文件中的 LOCAL_MODULE 一样。

APP_PROJECT_PATH

这个变量也是强制性的, 并且会给你的应用工程的根目录一个绝对路径。这是用来复制或者安装一个没有任何版本限制的 JNI 库, 从而给 APK 生成工具一个详细的路径。

APP_OPTIM

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

这个可选的变量可以定义为"发布版"或者是"debug 版"。这是用来在你编译你的项目模块的时候,改进你的优化等级。

release 模式时默认的,也会生成更加优化的二进制文件。"debug 模式"会生成一些没有优化过的二进制文件,但是更容易检测出一些 bug。

这里需要注意一下,其实两种模式都能检测出 bug。但是 release 模式相对来说提供的信息较少。在我们除 bug 的过程中,有一些变量被优化,或者根本就无法被检测出来,代码的重新排序会让这些代码变得更加难以阅读,并且让这些轨迹更加不可靠。

APP_CFLAGS

当要编译模块中的有任何 C 文件的时候,C 编译器的信号就会被发出。这里可以在你的应用中需要这些模块时,进行编译的调整,这样就不用直接来更改 Android.mk 文件本身了

IMPORTANT WARNING: ++++++

信号中所有的路径都需要和 NDK 的顶级目录相关。比如说,如果你已经遵循了下面的安装步骤

+ sources/foo/Android.mk

+ sources/bar/Android.mk

为了在你想要在编译过程中添加到"bar"源码中的 foo/Android.mk 文件,你应该这样做:

+ APP_CFLAGS += -Isources/bar

或者这样

+ APP_CFLAGS += -I\$(LOCAL_PATH)/../bar

+ Using '-I../bar' will *NOT* work since it will be equivalent to

+ '-I\$NDK_ROOT/../../bar' instead.

使用 '-I../bar' 不会起作用,除非他已经被'-I\$NDK_ROOT/../../bar' 所代替

+++++

APP_CXXFLAGS

这里和 APP_CFLAGS 的 C++源文件一样

APP_CPPFLAGS

这里和 APP_CFLAGS 文件已经,并且会传递 C 和 C++的源文件

一个 Application.mk 文件的应该像这样:

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!

----- cut here -----

APP_MODULES := <list of modules>

APP_PROJECT_PATH := <path to project>

----- cut here -----

7.2 Android NDK How-To:

这里是一份 NDK 开发人员的小技巧的集锦

如何强制运行编译命令:

执行"make APP=<yourapp> V=1"这段命令会让编译进行。这里可以用来明确一下哪些部分被编译,并在 NDK 编译系统中检查一下 bug。

(V=1 trick 是来自 Linux 内核编译系统)

如何重构你的源码:

Use GNU Make's "-B" option, as in:

```
make APP=<yourapp> -B
```

如果在你的本地机器码下存储你的源代码呢?

在不同的路径下存储源代码都比直接在\$NDK_ROOT/sources.更加方便。比如说,你想在同一目录下放置这些文件,而不是放置在其他工程文件中,那就要让每一个事务都在控制中,

NDK 编译系统需要直接获取\$NDK_ROOT/sources 的源代码和 \$NDK_ROOT/apps/<name>.下的 Application.mk。本质上说这是为了让实例化变得明理且简单,以及保证所有的功能,都可以自发并独立得管理,并且工作稳定。

你可以使用下面简单的符号链接来限制你的路径。比如

```
$NDK_ROOT/sources/<foo> ----> $PROJECT_PATH/jni/<foo>
```

```
$NDK_ROOT/apps/<foo> ----> $PROJECT_PATH/jni/<foo>
```

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!

`$PROJECT_PATH/jni/<foo>` 包含一个 `Android.mk` 和一个 `Application.mk` 文件。注意所有的像 `$(call my-dir)` 文件都会在编译的时候给出一个 NDK 的路径()也就是

`$NDK_ROOT/sources/<foo>)`

Windows 用户请注意:NDK 只支持 `cygwin`, 如果要是用这些符号必须通过 `"ln -s"` 这样的命令行, 需要这样来写

```
ln -s <target> <link>
```

怎样在你的模块中添加合适的目录声明:

如果你定义了几个模块, 这里通常需要来包含一个模块头, 同时编译另外一个。考虑一下下面的例子吧:

```
$NDK_ROOT/sources/foo/
```

```
Android.mk
```

```
foo.h
```

```
foo.c
```

```
$NDK_ROOT/sources/bar/
```

```
Android.mk
```

```
bar.c
```

当 `'bar.c'` 使用 `'#include <foo.h>'` 时, 你需要在 `bar/Android.mk` 中的模块中增加一个 `'foo'` 路径, 这样才可以编译成功。

下面是一种写法。

```
LOCAL_CPPFLAGS := -I../foo
```

然而有的时候这样可能不起作用, 因为所有的编译都是从 NDK 的根目录发生的(也就是 `$NDK_ROOT`), 这里必须包含相关的文件。下面的几句都需要被翻译:

```
LOCAL_CPPFLAGS := -I$(NDK_ROOT)/../foo
```

这是增加一个没有头文件存在的 C 的目录, 正确的应该这样写:

```
LOCAL_CPPFLAGS := -Isources/foo
```

或者这样:

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!


```
LOCAL_CPPFLAGS := $(LOCAL_PATH)/../foo
```

这里使用 \$(LOCAL_PATH) 相关的路径, 为了防止你将 'foo' 和 'bar' 放到 'sources' 一个比较深的路径。

7.3 Android NDK Stable APIs:

这里列出了 Android NDK 中比较稳定的 APIs 和 ABIs 接口

I. 目的:

每一个 API 都会反映一个头文件的设置, 还有动态链接库中所包含的实例工具, 这些都需要被源代码所链接。

比如, 如果要使用系统库中的 "Foo", 你必须在你代码中包含 <foo.h>, 告诉编译系统你的原生模块需要用到这个文件, 并且在装载的时候在 Android.mk 文件中增加这几句话/system/lib/libfoo.so

```
LOCAL_LDLIBS := -lfoo
```

注意编译系统会为你的源代码自动链接到 C 的库, 数学库和 C++ 的支持库。这里不需要在 LOCAL_LDLIBS 里面列出来。

II. Android 1.5 稳定的接口:

下面列出来的接口都可以被源代码使用, 并可以运行在 1.5 或以上的镜像上。

C 库:

C 库的头文件, 就像他们在 1.5 中定义好的标准的名字(<stdlib.h>, <stdio.h>, etc...) 如果有一个头文件在编译时丢失, 这可能会让我们的编译好的文件无法在 1.5 的系统镜像中起作用。

编译系统会自动将你的模块链接到 C 库, 不需要我们自己添加 LOCAL_LDLIBS

注意 Android C 库包括对 pthread(<pthread.h>) 的支持, 所以 "LOCAL_LIBS := -lpthread" 这句话就不是必须的了。对于真实事件的扩展也是一样的(这个首先是分布在 Linux 系统上的)

**** VERY IMPORTANT NOTE: *******

* 在某种情况下, Kernel 的在 <linux/...> and <asm/...> 的头文件,

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

- * 可能不是很稳定。我们应该避免直接使用他们,
- * 因为他们很有可能在未来的平台版本中被更替,
- * 对于硬件的定义也是一样的。

数学库:

<math.h> 是必须的, 而且数学库会在编译时自动链接到你机器码中的模块, 所以这里没必要通过 LOCAL_LDLIBS 列出一个 "-lm" 列表

C++ Library:

这里对 C++API 的支持很有限, 对于 Android 1.5, 目前被限制于这些头文件

<cstdint>

<new>

<utility>

<stl_pair.h>

他们可能不能包含所有我们所需要的标准的定义。很显然对于 Android1.5 的系统镜像, C++和 RTTI 还不能被很好的支持

C++支持库(-lstdc++)会自动链接到你自己的模块, 所以这里没必要通过 LOCAL_LDLIBS 列出他们。

Android 的日志系统的支持:

<android/log.h> 包括很多不同的定义, 他们可以从你的代码发送日志文件到 Kernel. 请仔细看一下这个文件的内容(build/platforms/android-1.5/common/include/android/log.h), 这里有具体怎么使用它的注释

你可以按照你自己的意愿写一些简单的, 又很有帮助的宏。

如果你这样做了, 你的原声的模块会链接到 to /system/lib/liblog.so, 通过以下的句子

LOCAL_LDLIBS := -llog

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

ZLib Compression Library:

<zlib.h> 和 <zconf.h> 是可以的, 并且可以被 Android1.5 的系统镜像 的 Zlib 库来压缩。与 ZLib 相关的文件, 请看这个页面 <http://www.zlib.net/manual.html>

如果你要使用它, 你的原声的模块会链接到/system/lib/libz.so

LOCAL_LDLIBS := -lz

7.4 Android System Image Issues

这篇文档包含了一些在 Android 系统镜像中已经存在的, 但是 NDK 的开发者需要注意的内容

I. Android 1.5 System Issues:

以下内容都是针对官方出品的 Android 1.5 系统镜像

标准的 C++ 库不被支持:

Android 1.5 的系统不支持 C++ 的标准库, 也没有提供相关应用的源代码。相反, 这里有一个被限制的头文件被提供(详见 docs/STABLE-APIS.TXT), 他们被用来构建 Android 平台, 这些都是被支持的。

这里可以试着修改一下已经存在的 C++ STL 工具, 但是对这部分的支持还不好, 我们建议试着用 uSTL 和 STLport, 如果你真的希望使用 C++ 的话。

No support for C++ exceptions and RTTI:

Android 1.5 的系统镜像欠缺一些必要的 C++ 的扩展功能还有 RTTI。C++ 代码是取决于这些功能的, 他们不可以被链接或者跑在 Android 1.5 系统上。

C Library limitations:

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

在虚拟机中, C 库或 C 语言的东西不会执行所有的功能。很显然, 线程的撤销还不被支持, 具体的信息, 请参见 docs/system/libc/OVERVIEW.TXT

No SysV IPCs in C library:

Unix 的进程间通信的 API 还不能被 C 库所支持, 为了避免拒绝服务。详见 docs/system/libc/SYSV-IPC.TXT

C Library bug: getservbyname() returns port number in incorrect order:

Android1.5 C 库的 getservbyname() 会返回一个通道数, 来给一个特定的网络服务。这个功能会存储它的结果到'struct servent' 这个结构中, 通道数会存在这里。

标准的任务会要求这个值是存储在网络命令(因此我们需要通过 ntohs()转换一下 host 的命令)然而, 目前 1.5 版的工具还有一些 bug, 并会返回一些数字。

This bug will be fixed in future releases of the platform, and applications should not depend on the wrong behaviour in the future. Avoid using this function if possible; if this is not possible, try to use a small wrapper

like the following one:

(这一句, 请大家参照原文, 译者的理解可能和你的理解会有出处)

Bug 会在未来的发布的平台中被修复, 所以我们写的应用程序不应该在将来, 基于这些错误的行为。为了避免使用这些功能, 即使是可能还是不可能, 我们应该像下面那样做一个小的封装:

```
static struct servent*

my_getservbyname(const char* name, const char* proto)

{

    static int    has_bug = -1;

    struct servent* ret;

    if (has_bug < 0) {

        ret = getservbyname("http", NULL);

        has_bug = (ret == NULL || ret->s_port == 80);

    }

}
```

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!


```
ret = getservbyname(name, proto);

if (has_bug)

    ret->s_port = htons(ret->s_port);

}
```

(返回的 `struct servent` 是本地线程相关的, 并且可以被 `caller` 识别。他会在下一个 `call` 中通过这个方法来重写)

动态链接库的限制

Android 的动态链接库在 1.5 版中有很多的限制:

- 目前还没有对 `LD_LIBRARY_PATH`, `LD_PRELOAD`, `RTLD_LOCAL`、以及其他的选择进行支持
- 静态的 C++ 结构会在执行中被调用两次, 这里是一个 C 库实例化顺序所造成的 `bug`。然后, 静态的 C++ 结构在共享库中只被调用一次。
- 静态的结果在一开始就不会被调用, 但是在程序退出、结束的时候则会被调用。
- 错误反馈功能很受限制, 他们只提供了一些比较普通的错误信息, 这些错误信息会让我们很难弄清楚为什么动态库的加载及连接功能会失效。大多数情况下, 错误的主因都是一个丢失的符号。
- 这里有一个 `bug` 会阻止一个建立在第三方的应用程序的共享库。比如, 如果你建立了两个文件, 一个是 `libfoo.so` 另一个是 `libbar.so` 在你的应用中, 并且在 `bar/android.mk` 文件中将列出的 `libfoo.so` 作为一个独立的 `libbar.so`, 如果再加载 `libbar.so`, 就会失败, 除非你已经在你的进程中加载了 `libfoo.so`

8.实例分析与入门实例 NDK 自带实例分析

作者: 向上

EMAIL: xusaomaiss@gmail.com

版本号: 2009-8-19

原创作品, 欢迎转载, 注明出处, 谢谢谢谢!

目标读者: NDK 入门, 有 android 开发基础, 了解 C 语言语法。

要求: 已经阅读过 OVERVIEW.TXT (Android NDK 概述) 等 NDK 自带的几份文档, NDK 环境已经安装好。更多信息阅读【参考资料】

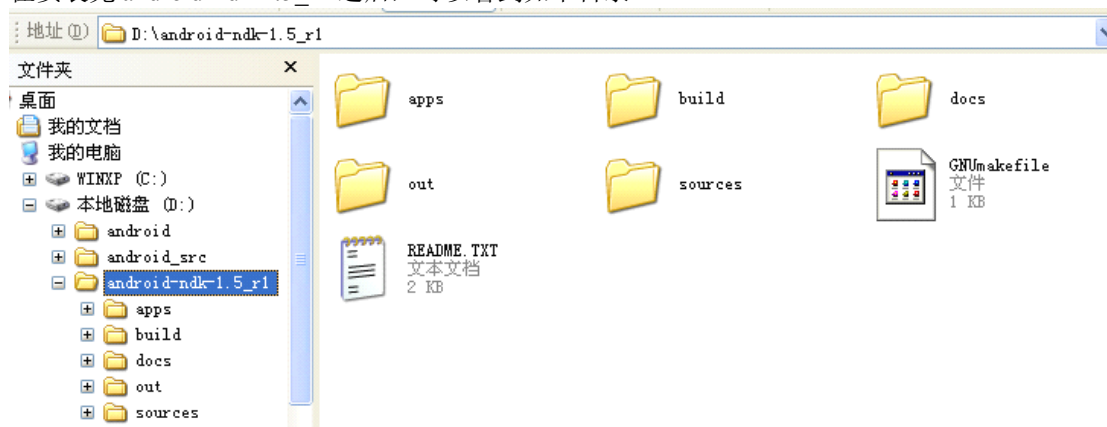
8.1 前言:

在 www.eoeandroid.com 论坛中发了几篇关于 NDK 的帖子, 同时查看了一下相关论坛, 发现现在搞 NDK 的还少, 不过还有很多人在门外观望, NDK 现在相关的中文资料比较少, 能参考的多是 android-ndk-1.5_r1 中自带的英文文档, 为了更快的入门, 本文将以 NDK1.5 自带实例进行分析, 然后在这基础上介绍分析了 hello-jni, 本文的思想是在解决错误的探索中前行。

正文:

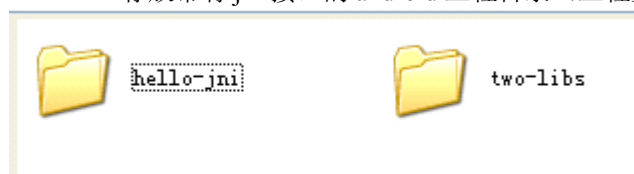
8.2 熟悉环境:

不要急着就去编译代码, 先熟悉一下相关的目录结构, 以便在后面碰到问题时也好查找, 在安装完 android-ndk-1.5_r1 之后, 可以看到如下目录



1. Apps :

存放带有 jni 接口的 android 工程目录 (工程里面有利用 native 关键字定义的 java 函数)



2. Build :

存放着几乎所有的 ndk 编译相关的脚本以及必要的静态链接库。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!



3. Doc :

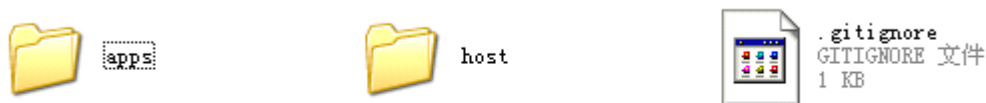
存放着 ndk 的所有“官方”文档，每一篇文档对于 jni 编写者来说这里面的任何一点点资料都是无价的

名称	大小	类型	修改日期
system		文件夹	2009-6-1 23:05
ANDROID-MK.TXT	14 KB	文本文档	2009-6-1 21:38
APPLICATION-MK.TXT	4 KB	文本文档	2009-5-11 15:26
CHANGES.TXT	1 KB	文本文档	2009-6-1 23:04
HOWTO.TXT	3 KB	文本文档	2009-6-1 21:38
INSTALL.TXT	2 KB	文本文档	2009-5-11 15:26
LICENSES.TXT	1 KB	文本文档	2009-5-11 23:37
OVERVIEW.TXT	11 KB	文本文档	2009-5-11 15:26
STABLE-APIS.TXT	4 KB	文本文档	2009-6-1 21:38
SYSTEM-ISSUES.TXT	5 KB	文本文档	2009-6-1 22:26

- INSTALL.TXT — 说明如何安装、配置 NDK
- OVERVIEW.TXT — 概要介绍 NDK 的功能和用法
- ANDROID-MK.TXT — 说明 Android.mk 的用法, Android.mk 用来指定需要编译的源代码
- APPLICATION-MK.TXT — 说明 Application.mk 的用法 file, Application.mk 用来指定目标应用
- HOWTO.TXT — 介绍与 NDK 开发相关的任务.
- SYSTEM-ISSUES.TXT — 如果用 NDK 开发, 你需要了解 Android 系统映像相关的知识
- STABLE-APIS.TXT — NDK 头文件列表
- 另外, 还包含了“bionic”C 库的详细信息, 如果用 NDK 开发, 你应该了解这些信息。路径: <ndk>/docs/system/libc/:
- OVERVIEW.TXT—介绍“bionic”C 库及其特性

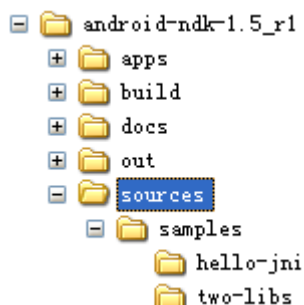
4. Out:

存放一些中间的临时文件，例如 jni 的.c/.cpp 文件编译过程中产生的.o 文件等。



5. Source

存放 jni 文件的.c/.cpp 的源代码文件

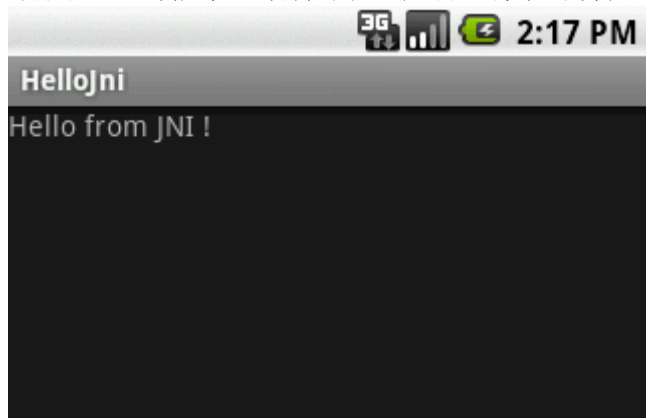


我们需要分析的代码就在 source 里面，让我们先从这里开始。下面说明一下各应用的功能。NDK 包含了两个 Android 应用，用来说明如何在 Android 应用中用原生代码。

- hello-jni—该示例：调用共享库（shared library）的原生方法获取一个字符串，并显示在应用的界面上。
- Two-libs—该示例：动态加载一个共享库，并调用其中的方法，该方法由一个导入到共享库的静态库实现。

8.3 万里长征第一步：

做为一份分析文档，就不详细说明环境的安装与配置了，大家可以参考：Android NDK 开发环境安装和配置 这篇文档，顺利的话，就可以出现如下界面



代码：

//android 中调用的代码

```
public class HelloJni extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        /* Create a TextView and set its content.
         * the text is retrieved by calling a native
         * function.
         */
    }
}
```

本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！


```

        TextView tv = new TextView(this);
        tv.setText( stringFromJNI() );
        setContentView(tv);
    }

    /* A native method that is implemented by the
     * 'hello-jni' native library, which is packaged
     * with this application.
     */
    public native String  stringFromJNI();

    /* This is another native method declaration that is *not*
     * implemented by 'hello-jni'. This is simply to show that
     * you can declare as many native methods in your Java code
     * as you want, their implementation is searched in the
     * currently loaded native libraries only the first time
     * you call them.
     *
     * Trying to call this function will result in a
     * java.lang.UnsatisfiedLinkError exception !
     */
    public native String  unimplementedStringFromJNI();

    /* this is used to load the 'hello-jni' library on application
     * startup. The library has already been unpacked into
     * /data/data/com.example.HelloJni/lib/libhello-jni.so at
     * installation time by the package manager.
     */
    static {
        System.loadLibrary("linhai-jni");
    }
}

```

应该来说上面的代码是很简单的, 从功能上说是 Android NDK 版的 Helloworld.
数据是从 stringFromJNI() 函数获得。

//NDK 中的代码

```

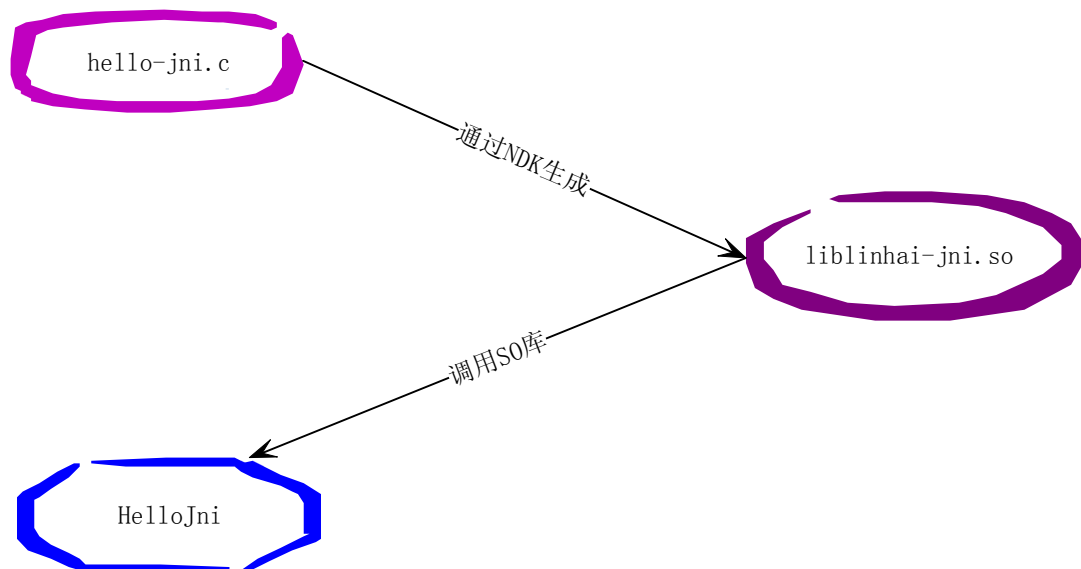
/* This is a trivial JNI example where we use a native method
 * to return a new VM String. See the corresponding Java source
 * file located at:
 *
 * apps/samples/hello-jni/project/src/com/example/HelloJni/HelloJni.java
 */
JNIEXPORT_STRING
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
                                                    jobject thiz )
{
    return (*env)->NewStringUTF(env, "Hello from JNI !");
}

```

分析:

从整个流程上看, 应该是挺简单的, 和原来 VC 中生成 dll 文件, 提供给第三方调用是一样的, 只是原来的 dll 改成 so 文件。

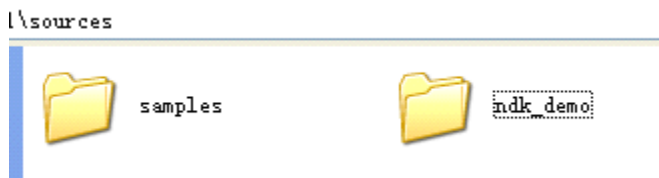
本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!



8.4 从错误中入门:

有人说失败是成功之母, 太多的教程都是教别人如何写一个正确的程序, 对于可能出现的问题很少提及。以致于碰到问题, 不知道如何解决。因此, 先从错误开始, 一步步让你从错误中了解问题的真象。

在看完例子之后, 是不是很想自己也亲手写一个? 让我们从最简单的开始: 把例子 `hello-jni` 拷一份到 `source` 目录下, 重新命名为 `ndk_demo`, 如下:



修改 `mk` 的文件内容, 修改成如下:

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

应该来说, 以上的修改是完全参考 `hello_jni` 的文件进行修改。

第二步: 编译


```

/cygdrive/d/android-ndk-1.5_r1
Administrator@1FFEB2365E03421 ~
$ cd $ANDROID_NDK_ROOT

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$ make APP=ndkdemo_jni
Android NDK: The APP variable contains unknown app names: ndkdemo_jni
Android NDK: Please use one of: hello-jni two-libs
build/core/main.mk:171: *** Android NDK: Aborting . Stop.

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$

```

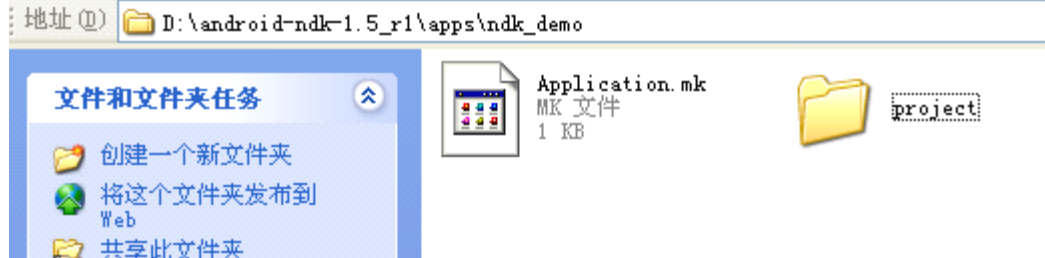
提示错误, 从错误提示中我们可以清楚的看到, 找不到 ndkdemo_jni 这个 app 名称。
 从第二行提示可以看出, 我们可以使用的 APP 参数有两个: hello-jni 和 two-libs
 太神奇了, 这些提示内容丰富的很。按上面的提示, 回想一下 ndk 的目录结构中不是正好有一个 APP 的文件夹吗? 进去看, 参看 apps\hello-jni\Application.mk
 APP_PROJECT_PATH := \$(call my-dir)/project
 APP_MODULES := hello-jni

内容很简单, 不过里面有一个很重要的东西 APP_MODULES:=hello-jni, 于是, 在 APPS 文件夹下面创建一个名为 ndk_demo 的文件夹, 文件夹中包括一个 Application.mk 文件, 并创建一个空的 project 的文件夹, 在 Application.mk 的文件内容:

```

APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES := ndk_demo

```



重新编译, 还是错误, 不过这次可以看到多了一个 ndk_demo

```

/cygdrive/d/android-ndk-1.5_r1
Administrator@1FFEB2365E03421 ~
$ cd $ANDROID_NDK_ROOT

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$ make APP=ndkdemo_jni
Android NDK: The APP variable contains unknown app names: ndkdemo_jni
Android NDK: Please use one of: hello-jni two-libs
build/core/main.mk:171: *** Android NDK: Aborting . Stop.

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$ make APP=ndkdemo_jni
Android NDK: The APP variable contains unknown app names: ndkdemo_jni
Android NDK: Please use one of: hello-jni ndk_demo two-libs
build/core/main.mk:171: *** Android NDK: Aborting . Stop.

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$

```

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

这不是在 apps 下刚刚新建的 ndk_demo 文件夹名吗, 使用 ndk_demo 编译

```
Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$ make APP=ndk_demo
Android NDK: Building for application 'ndk_demo'
Compile thumb : ndk_demo <= sources/ndk_demo/ndkdemo-jni.c
SharedLibrary : libndk_demo.so
Install : libndk_demo.so => apps/ndk_demo/project/libs/armeabi

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$ make APP=ndk_demo
Android NDK: Building for application 'ndk_demo'
make: Nothing to be done for 'all'.

Administrator@1FFEB2365E03421 /cygdrive/d/android-ndk-1.5_r1
$
```

从上可以看出, 编译成功, 第二次编译, 提示没有东西需要编译, 说明, NDK 应该是增量编译, 也就是说如果代码修改了, 则编译修改过的那部份代码, 以提高编译速度。

再查看 apps/ndk_demo/project 文件夹下面多了几个文件夹, 最后的 libndk_demo.so 文件在里面。同时查看 out 文件夹, 里面也多了一个文件夹 ndk_demo, 应正了 out 文件夹是存放一些中间的临时文件, 这也就是在文章开始要先熟悉目录结构的目的。



试验操作过程如下:

8.5 试验过程:

第一次试验, 各参数使用相同变量, 结果成功, 和预期一样, 但从中无法分析出它参数之间的关系。

第二次: 失败, 从错误提示信息中, 得到提示, 应该输入 **Ndk_demo1**, 从中分析出 make APP 参数应该是 apps 目录下的文件夹名称。

第三次: 成功, 验证了推想。

第四次: 上成功的基础上, 修改了 app_modules 参数, 结果失败, 从参数名观察, APP_MODULES

	Apps/工程名	application.mk APP_MODULES	Source/工程名	/Android.mk LOCAL_MODULE	Make 参数	结果
1.	Ndk_demo	Ndk_demo	Ndk_demo	Ndk_demo	Ndk_demo	成功
2.	Ndk_demo1	Ndk_demo	Ndk_demo	Ndk_demo	Ndk_demo	失败
3.	Ndk_demo1	Ndk_demo	Ndk_demo	Ndk_demo	Ndk_demo1	成功
4.	Ndk_demo1	Ndk_demo1	Ndk_demo	Ndk_demo	Ndk_demo1	失败
5.	Ndk_demo1	Ndk_demo1	Ndk_demo	Ndk_demo1	Ndk_demo1	成功
6.	Ndk_demo	Ndk_demo	Ndk_demo2	Ndk_demo	Ndk_demo	成功
7.	Ndk_demo	Ndk_demo	Ndk_demo2	Ndk_demo2	Ndk_demo	失败
8.	Ndk_demo	Ndk_demo2	Ndk_demo2	Ndk_demo2	Ndk_demo	成功
9.						

和 LOCAL_MODULE 两者应该有关联。

第五次: 修改 LOCAL_MODULE 参数结果成功, 验证了推想。

第六次: 修改 source 下的工程文件夹名, 不影响结果, 说明, source 文件夹下的文件夹名不影响编译。查看 Android_NDK_Overview.doc 文档

默认情况下, NDK 将寻找匹配下列的所有文件:

```
$NDK/sources/*/Android.mk
```

如果你想要定义 Android.mk 到子目录, 你应明确地在顶级 Android.mk 文件中包含它们。这有一个帮助函数可以做到, 如,使用:

```
include $(call all-subdir-makefiles)
```

这将包含全部在当前编译文件路径下子目录中的全部 Android.mk 文件。

证明了猜想。

第七次: 为了验证 APP_MODULES 和 LOCAL_MODULE 两者是否有关联, 再次试验。预期失败和真实结果一致。

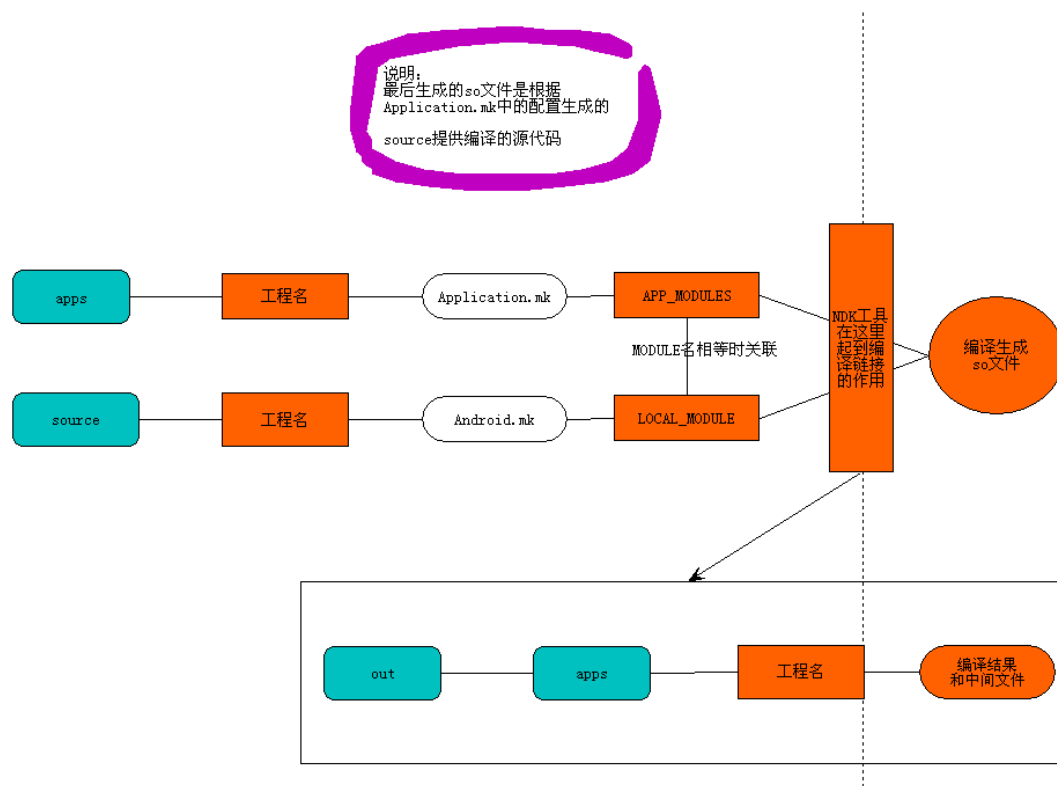
第八次: 验证了 APP_MODULES 和 LOCAL_MODULE 有关联的猜想。

8.6 小结

- 在 make APP 的参数是使用 apps 目录下的文件名做为参数
- APP_MODULES 和 LOCAL_MODULE 必须相等, 最后生成的 so 文件是要根据 APP_MODULES 配置生成的, 在 android.mk 文件中主要是配置源文件的路径等相关信息。

以下是关系图

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!



8.7 离胜利还有一步: 利用 so

把 so 拷到 hello-jni 工程中 hello-jni\project\libs\armeabi 文件夹下, 修改代码

```
static {
    System.loadLibrary("ndk_demo");
}
```

运行代码, 成功。

观察函数名特点, 发现除 java 外, Java_com_example_hellojni_HelloJni_stringFromJNI 中 com_example_hellojni_HelloJni 这个字符串就是包名+类名的组合, 只是把 . 改成了 _ 所以在用 一个 so 接口时, 这点一定要注意。JNI 的命名规则: java_包名_类名_方法名

以下是 JNI 的书写步骤:

1. 编写带有 native 声明的方法的 java 类
2. 使用 javac 命令编译所编写的 java 类
3. 使用 javah ?jni java 类名生成扩展名为 h 的头文件
4. 使用 C/C++ 实现本地方法
5. 将 C/C++ 编写的文件生成动态连接库

从上面流程可以看出, jni 是先写 java 类, 然后通过工具生成头文件, 所以函数接名要使用上面的规则。在使用 NDK 时, 如果要写的 so 是需要提供给其它人使用, 在命名时就要考虑这点,

同时, 为了方便他人使用, 最好提供一下 java 的函数接口包。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

以开源的 box2d 为例, 在提供了 so 文件后, 还提供了 ABox2D . java 文件, 为了方便用户测试接口, 最好提供 helloworld 这样的简单接口, 它与业务无关。基本思想和 windows 上的使用 C 写的 dll 文件和 webservice

提供接口时, 也提供 helloworld 这样的方式一样。

```
package com.boredprogrammers.chronobox.abox2d;

import com.boredprogrammers.chronobox.BallView;

public class ABox2D {
    static {
        System.loadLibrary("abox2d");
    }
    /**
     * Test method, quick sanity check of world.
     * @return
     */
    public native float helloWorld();
    public native void setupDroppingBalls(float width, float height);
    public native void stepDroppingBalls(BallView callback);
}
```

8.8 后言:

这篇文章总结一下就几句话, 不过为了这几句话, 本人做了大量的试验。由于是分析性的文档, 所以文档中大部分是大家可能都了解的知识, 不过文档中重点还是希望把大家平时可能碰到的错误列出来, 以后碰到了, 知道如何解决。如果写的不好, 发邮件过来把我臭骂一顿吧。

9.NDK 入门开发实战 Ubuntu 版本

作者: 王华

MSN: qinghua3344521@msn.com

目标: 利用 NDK 生成 SO 库, 使用 SO 库进行 JNI 调用, 在 Android sdcard 创建文件并写入数据, 同时在 View 上显示文本。

工具: Eclipse, ADT9.0 ,SDK ,NDK

开发平台: Ubuntu8.10

说明: 因为本人 XP 系统原因没有成功安装 Cywin, 下面的操作是 Ubuntu8.10 平台上, 对于 XP 使用 Cywin 的朋友, 操作没有太大的差别。

1) 新建工程,如图:

Project name: Ndk

Contents

- ☐ Create new project in workspace
- ☒ Create project from existing source
- ☒ Use default location

Build Target

Target Name	Vendor
<input type="checkbox"/> Android 1.1	Android Open Source Proje
<input checked="" type="checkbox"/> Android 1.5	Android Open Source Proje

Properties

Application name: Ndk

Package name: com.eoeandroid.ndk

☒ Create Activity: .Ndk

Min SDK Version: 3

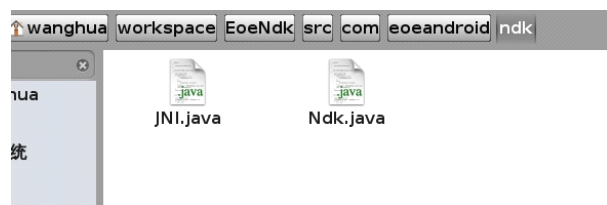
2) 添加
JNI
类,
如
图:



3) 在 JNI 类里面写连个 native 原生函数: printf 和 write, 如图:

```
JNI.java x *Ndk.java
1 package com.adnroid.jni;
2
3 public class JNI{
4     public native void write();
5     public native String printf();
6
7 }
8
```

4) 拷贝工程目录下 src\com\eoandroid\ndk 下的 JNI.java 文件

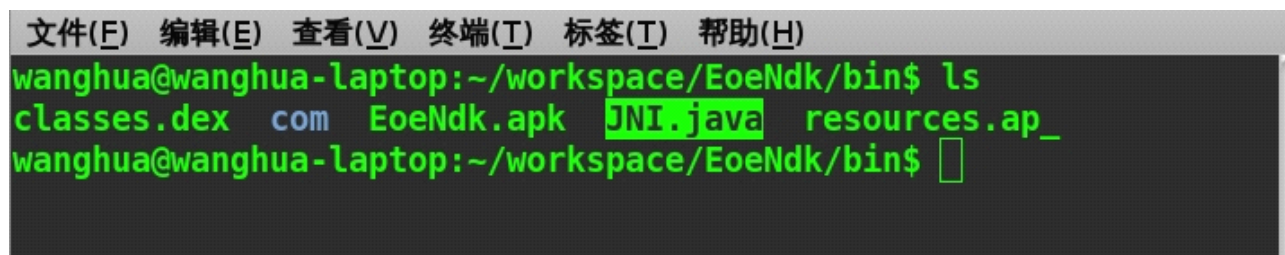


本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

5) 粘贴到工程目录 bin 目录下, 如图:



6) 终端下进入 bin 目录, 可看到 JNI.class 文件, 如图:



7) 输入命令: javac JNI.java 生成 JNI.class 文件, 如图:

```
wanghua@wanghua-laptop:~/workspace/EoeNdk/bin$ javac JNI.java
wanghua@wanghua-laptop:~/workspace/EoeNdk/bin$ ls
classes.dex  EoeNdk.apk  JNI.java
com          JNI.class   resources.ap_
wanghua@wanghua-laptop:~/workspace/EoeNdk/bin$
```


8) 拷贝 bin 目录下的 JNI.class 文件, 如图:



9) 粘贴到 EoeNdk\bin\com\eoeandroid\ndk 下, 替换掉原来文件, 如图:



10) 进入 bin 目录, 输入命令: `:javah -classpath . -jni com.eoeandroid.ndk.JNI` 生成 `com_eoeandroid_ndk_JNI.h` 文件, 如图:

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!



11) 在 bin 目录下建立 C 文件:com.eoeandroid.ndk.JNI.c，如图：
打开编辑文件，如图：

```
*com.eoeandroid.ndk.JNI.c
#include <stdio.h>
#include <stdlib.h>
#include "com_eoeandroid_ndk_JNI.h"

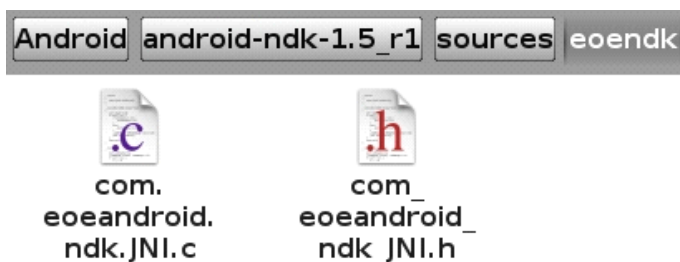
JNIEXPORT jstring JNICALL Java_com_eoeandroid_ndk_JNI_printf
(JNIEnv *e, jobject j){
    FILE *v = fopen("sdcard/eoeandroid.txt", "w+");
    fprintf(v, "hello eoeandroid Ndk Develop");
    fclose(v);
}

JNIEXPORT void JNICALL Java_com_eoeandroid_ndk_JNI_write
(JNIEnv *e, jobject j){
    return (*e)->NewStringUTF(env, "Hello EoeAndroid from
JNI! eoeandroid论坛:www.eoeandroid.com
作者:王华");
}
```

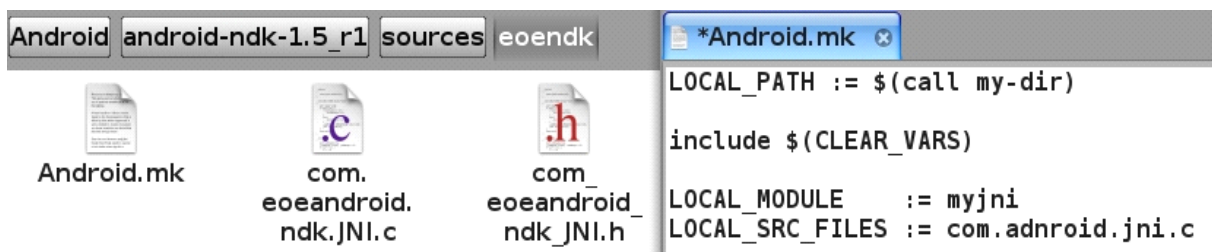
12) 保存上述代码，复制 bin 目录下的两个文件，如图：



13) 在 android-ndk-1.5_r1\sources 目录下,先创建 eoendk 文件夹,然后粘贴两个文件进来, 如图:



14) 在 eoendk 文件里创建 Android.mk 文件, 并编辑内容, 如图:



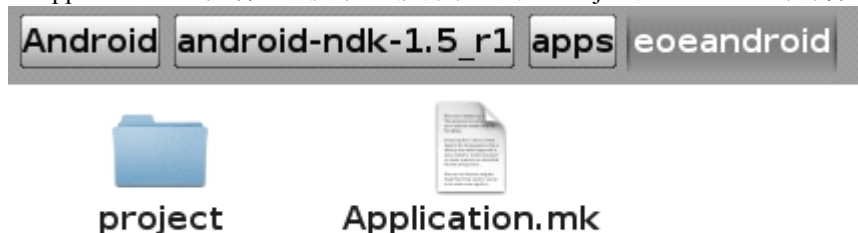
15) 修改代码如图:

```

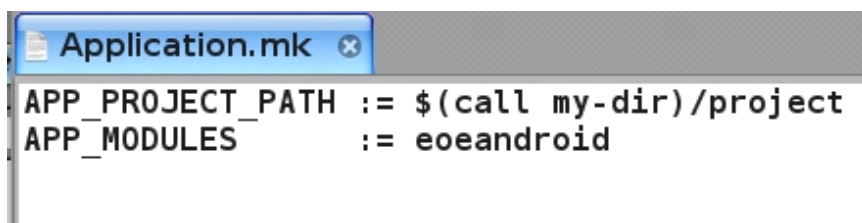
LOCAL_MODULE      := eoeandroid
LOCAL_SRC_FILES   := com.eoeandroid.ndk.JNI.c
  
```

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

- 16) 在 android-ndk-1.5_r1\apps 下创建文件夹 eoeandroid, 并在文件夹内创建 project 文件夹和 Application.mk 文件, 或是从上级目录里的 hello-jni 和 two-libs 里面拷贝, 如图:



- 17) 修改 Application.mk 文件, 修改 APP_MODULES:=eoeandroid, 如下图所示:

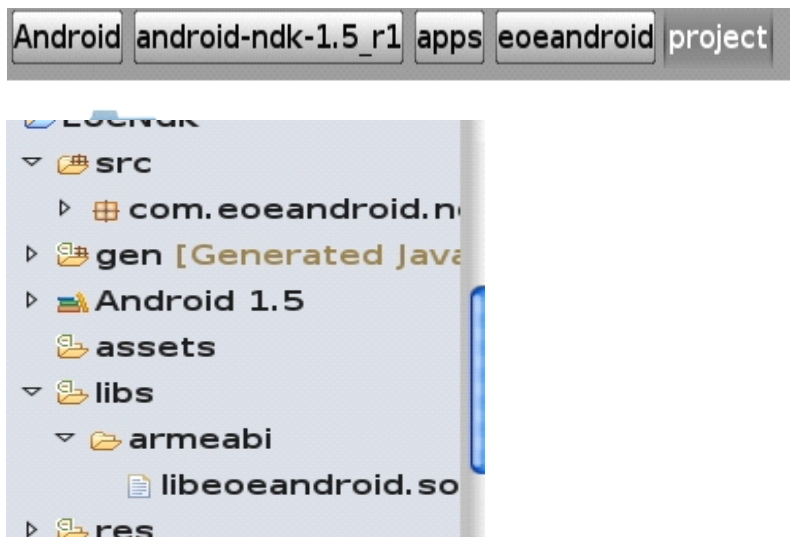


- 18) 进入 android-ndk-1.5_r1 目录, 编译: :make APP=eoeandroid, 没有错误就会出现下图所示:

```
wanghua@wanghua-laptop:~/Android/android-ndk-1.5_r1$ make APP=eoeandroid
Android NDK: Building for application 'eoeandroid'
Compile thumb  : eoeandroid <= sources/eoendk/com.eoeandroid.ndk.JNI.c
SharedLibrary  : libeoeandroid.so
Install        : libeoeandroid.so => apps/eoeandroid/project/libs/armeabi
wanghua@wanghua-laptop:~/Android/android-ndk-1.5_r1$
```

至此, 我们的原生代码已经生成 so 文件

- 19) 拷贝 apps/eoeandroid/project 下的 libs 文件夹, 到项目工程根目录, 或是直接右键工程粘贴进去, 如图:



- 20) 启动 emulator (因为本人 Ubuntu bash 问题) 需要进入 sdk 的 tools 目录, 如图:

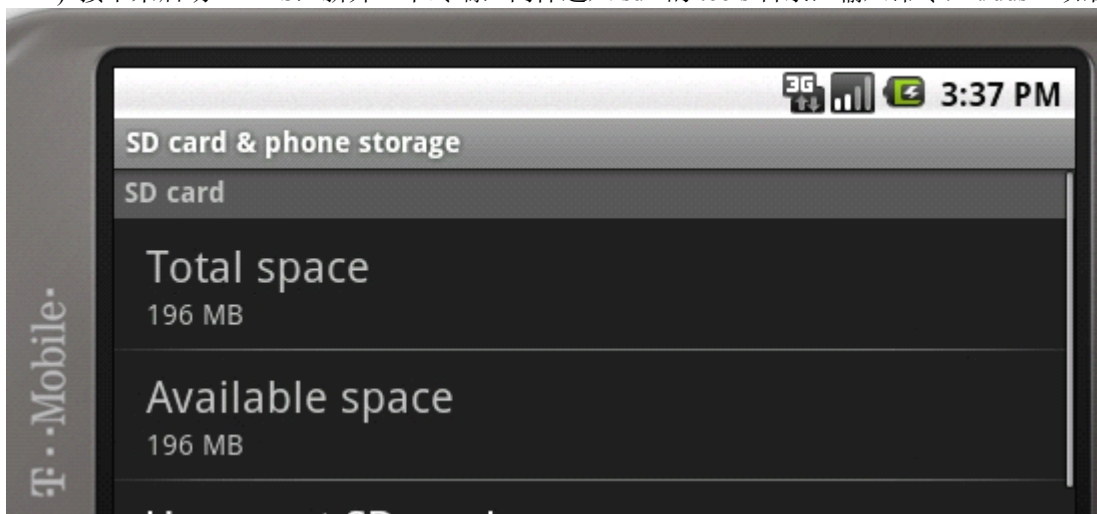
```
wanghua@wanghua-laptop:/usr/local/Android/Android/android-sdk-linux_x86-1.5_r2/tools$ ls
adb          ddms          emulator      lib           sqlite3
android      dmtracedump   hierarchyview mksdcard      traceview
apkbuilder   draw9patch    hprof-conv    NOTICE.txt
wanghua@wanghua-laptop:/usr/local/Android/Android/android-sdk-linux_x86-1.5_r2/tools$
```

启动皮肤 G1, 虚拟 Sdcard 为 200M 的 emulator 模拟器

注: Sdcard 帖子: <http://www.eoeandroid.com/viewthread.php?tid=2406>
不会用的朋友可以看看。



- 21) 启动成功, 进入 setting—>Sd card & phone storage 看一下是否 sdcard 加载成功, 如图显示 196M 表明加载成功, 如图:
- 22) 接下来启动 DDMS, 新开一个终端, 同样进入 sdk 的 tools 目录, 输入命令: `./ddms` 如图:




```
wanghua@wanghua-laptop:/usr/local/Android/Android/android-sdk-linux_x86-1.5_
ools$ ./ddms

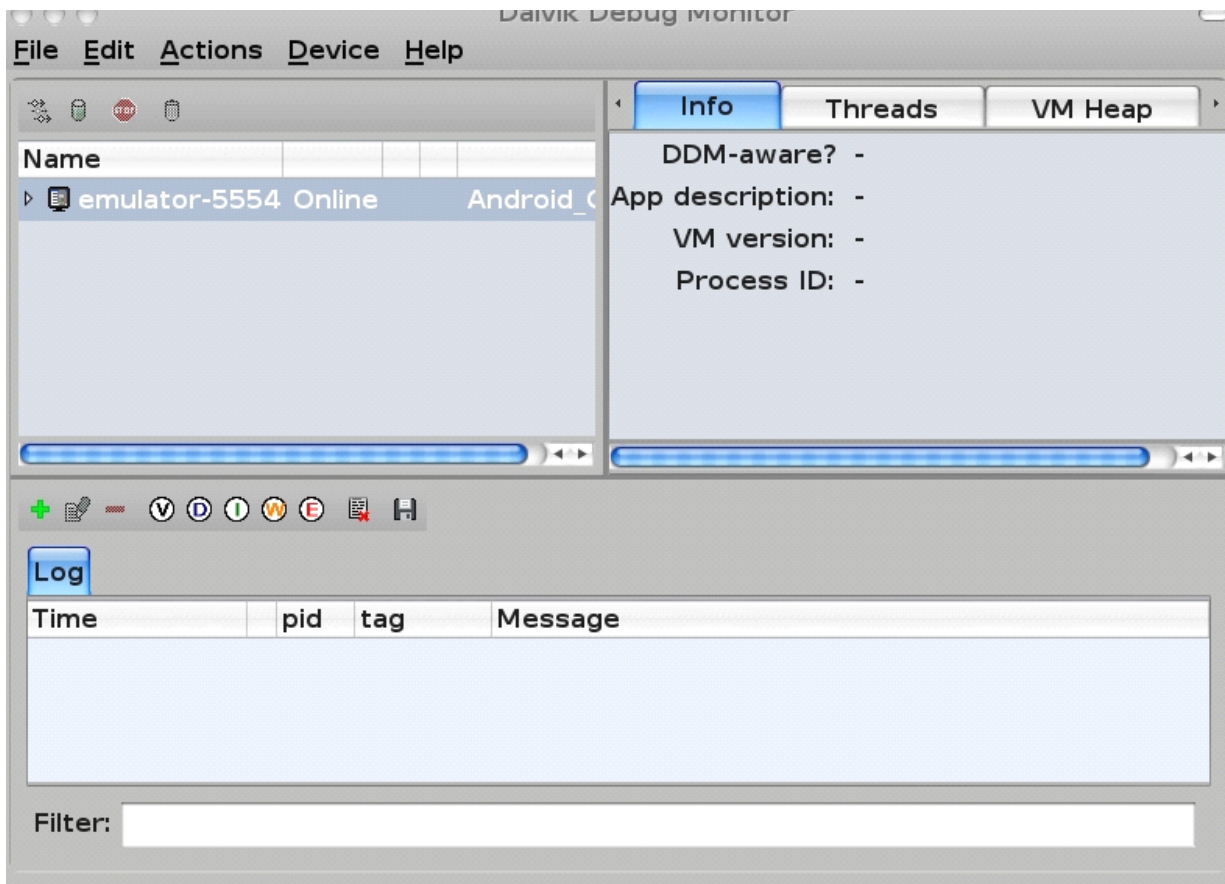
(ddms:7509): Gdk-WARNING **: gdk_window_set_icon_list: icons too large
22:40 E/ddms: Could not open Selected VM debug port (8700). Make sure you do
have another instance of DDMS or of the eclipse plugin running. If it's bei
sed by something else, choose a new port number in the preferences.
22:41 E/ddms: Can't bind to local 8600 for debugger
22:41 E/ddms: Can't bind to local 8601 for debugger
22:41 E/ddms: Can't bind to local 8602 for debugger
22:41 E/ddms: Can't bind to local 8603 for debugger
22:41 E/ddms: Can't bind to local 8604 for debugger
22:41 E/ddms: Can't bind to local 8605 for debugger
22:41 E/ddms: Can't bind to local 8606 for debugger

(ddms:7509): Gtk-WARNING **: gtk_widget_size_allocate(): attempt to allocate
get with width -5 and height 19

(ddms:7509): Gtk-WARNING **: gtk_widget_size_allocate(): attempt to allocate
get with width -5 and height 19

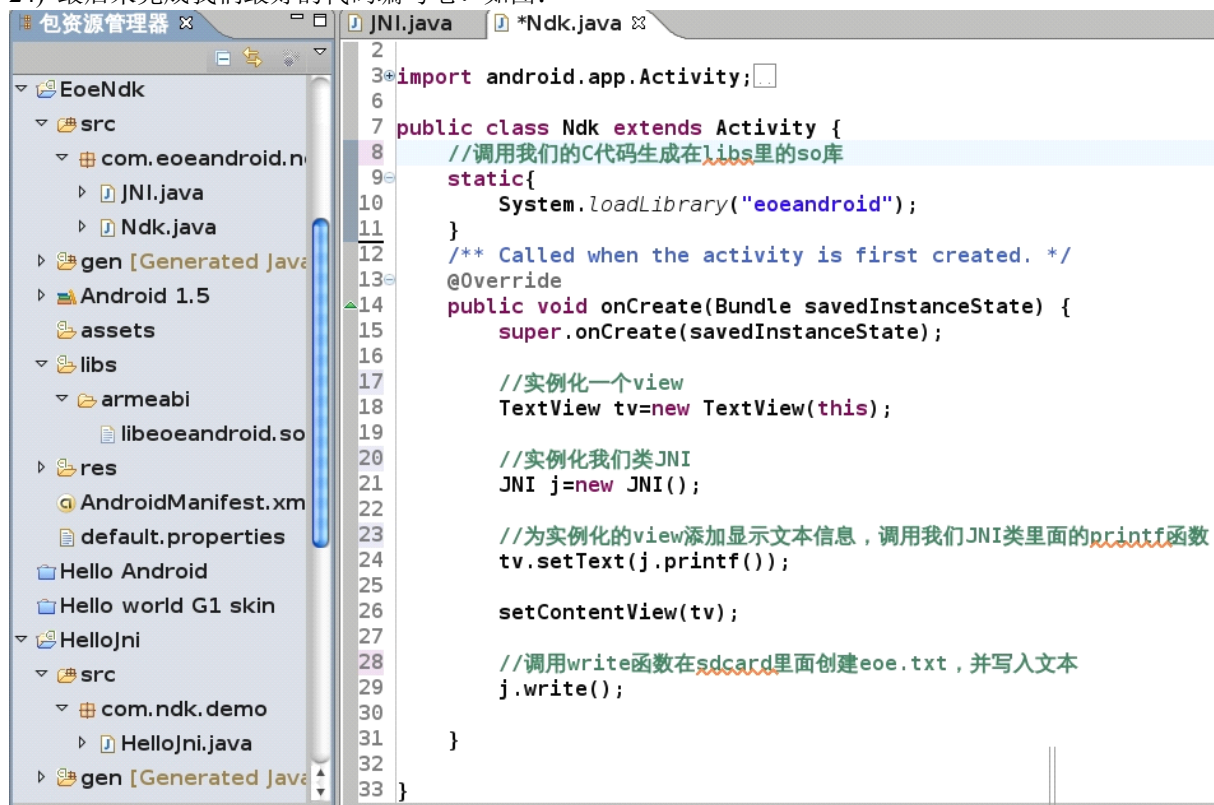
(ddms:7509): Gdk-WARNING **: gdk_window_set_icon_list: icons too large
```

23) 如下图, DDMS 启动成功!



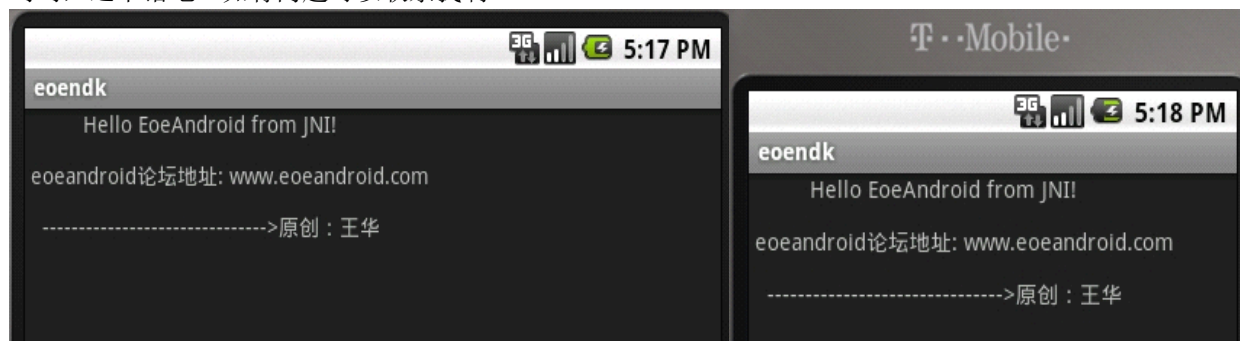
本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

24) 最后来完成我们最好的代码编写吧! 如图:



运行程序,如下所示:

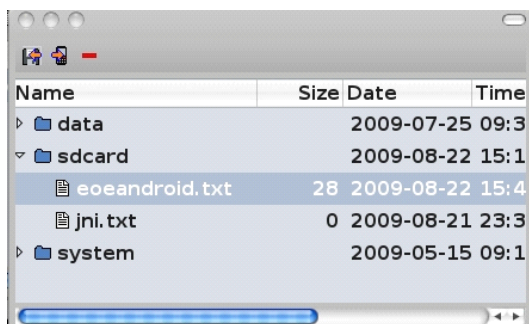
呵呵,还不错吧!如有问题可以联系我啊!



同时也欢迎大家直接到我们 eoeandroid 社区: www.eoeandroid.com 交流学习!

不要以为完事了啊,还没有呢,别忘了我们写了两个方法,一个 printf, 一个 write 啊,接下来看看我们在 sdcard 中创建的文本文件和写入的内容吧!

打开 DDMS 菜单 Device 下的 File Explorer, 展开 sdcard 目录,看到了吧 eoeandroid.txt 文件! 如图:



单击左上方手机图标向左箭头的按钮（Pull File from Device）导出文件，如图：
我导出到桌面了，打开它看看里面什么内容啊！如图：


```
NIEXPORT void JNICALL Java_com_eoeandroid_ndk_JNI_write
(JNIEnv *e, jobject j){
    FILE *v = fopen("sdcard/eoeandroid.txt", "w+");
    fprintf(v, "hello eoeandroid Ndk Develop");
    fclose(v);
}
```



呵呵，看看是不是和我们写的原生代码（C 代码）里面要写入的内容一致啊！！

到此，NDK 入门实例就结束了，你是不是有想法了啊！趁着脑袋热乎，赶紧自己也写一个 NDK 的小实例来炫一把吧！

如果示例有问题或错误，请联系我：qinghua3344521@msn.com

同样也欢迎大家来我们的社区：www.eoeandroid.com 交流学习！

谢谢！

10.eoeMarket

eoeMarket 将从本期开始在每一期特刊中, 为大家推荐一些 eoeMarket 平台中的一些精品的软件, 而且在后期, 我们甚至还会为大家推荐一些优秀的 Android 开发团队。

所以, 如果你有好的应用, 你应该尽快上传到我们的 eoeMarket。

因为是第一期, 可能还有很多朋友并不熟悉我们的 eoeMarket 软件平台, 所以我们会首先占用一定的篇幅给大家着重介绍一下 eoeMarket。

10.1eoeMarket 是什么?

eoeMarket 是由 eoeAndroid 社区推出的一个主要针对中文市场, 和 Android 开发者紧密合作、并给广大玩家提供优秀 Android 应用的集软件发布、搜索、安装于一体的平台。eoeMarket 平台致力于提供更好的 Android 软件服务, 拉近开发者和广大玩家的距离。

eoeMarket 包含两个部分:

web 端: 在 web 端可以发布, 搜索, 收藏应用

手机客户端: 在手机客户端可以浏览, 下载、安装应用

10.2eoeMarket 能给开发者带来什么?

A eoeMarket 对于开发者 在本地市场的 帮助

1. eoeMarket 的用户主要是中文用户, 用户的使用习惯我们更加清楚, 因此我们可以透过 eoeMarket, 制作出更适合中国人的软件。
2. eoeMarket 对于开发者而言, 不仅仅是一款简单的发布软件, 他更是一个平台, 一个具有交互功能的软件平台, 它能为开发者拉近与中国用户之间的距离, 在国内的环境下, 可以与更多的用户进行更好的交流。
3. 如果您的软件足够优秀, eoeMarket 可以帮您推荐给运营商和手机厂商, 如果您的软件可以成为运营商或者手机厂商指定的预装软件, 那您的软件所获得影响力将会成倍得增长!
4. eoeMarket 也是您展示实力的好机会, 优秀软件的作者, 也必定是优秀的您, 那么您更加可以透过 eoeMarket 来彰显自己的实力, 因此, 您能获得更多利润丰厚的外包项目。
5. eoeMarket 的定位决定了 eoeMarket 上的软件会被更多人下载和使用, 这是一次绝佳的给您的软件提供免费的反馈和建议的机会。这样您的软件会获得更多的用户的赞同, 同样也会获得更多的市场认同。

B eoeMarket 对于开发者 在国际市场的 帮助

6. eoeMarket 是针对国内用户的, 而国内有着更为挑剔的用户群, 而 eoeMarket 的环境更为宽松, 在您准备发布自己的软件时, 可以第一时间发布到 eoeMarket 中, 有了如此多的挑剔的玩家来检验, 可以让我们及早发现程序中的 bug, 或者不合理的地方。
当我们改掉这些不合理的地方后, 再上传到 Android Market 必定能征服更多挑剔的用户。

10.3 eoeMarket 能给玩家带来什么?

为方便国内用户更好地安装 Android 软件, 提供更多更好的适合我们中文本土化的应用软件, eoeAndroid 社区特地为广大的开发者和玩家提供了 eoeMarket 平台。
从此以后开发者可以轻松将自己做的最新最酷的软件上传到 eoeMarket, 广大的玩家朋友们也可以很方便地

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

在 eoeMarket 平台下载好玩、好用的本土化应用。

总而言之, eoeMarket 是 eoeAndroid 社区奉献给广大 Android 开发者和玩家的一份厚礼。

10.4 为什么使用 eoeMarket

1. 本土的服务器, 更快的访问速度。
2. 以中文应用软件及游戏为主。
3. 全部都是精选的软件, 不浪费您每一 KB 的流量。

eoeMarket 具有和 Android Market 同样优秀的界面, 只要大家使用过 Android Market, eoeMarket 可以很快上手。

最关键的是 eoeMarket 提供的应用软件是以本地化的中文语言为主的, 大部分软件和游戏都是让大家感到亲切友好的中文版, 再也不用担心下载下来的软件只有难懂的英文, 有 eoeAndroid 社区背后的支持,

eoeMarket 更会为大家推荐出更多更好的优秀软件。

不同于 Android Market 的杂乱, 在 eoeMarket 上, 看到的都是优秀的软件。

10.5 如何使用 eoeMarket

1. 如何注册? 通过 web 注册

您可以在您的电脑里边打开浏览器, 通过网址 <http://www.eoemarket.com/> 进行注册。



发布应用的分享平台.

首页	应用	游戏	主题	widget	下载客户端	关于eoeMarket	联系我们
----	----	----	----	--------	-------	-------------	------

为什么要注册?

注册后可以为您提供个性化的服务;
注册的帐号可以在web和手机客户端通用;
注册的帐号可以申请开发权限

注册新帐号

您的帐户信息 (这是您登录eoeMarket的凭证)

登录名* (请使用3..40位的小写字母, 数字或 - _ @)

Email地址* (请输入可以正常接收邮件的地址)

密码* (请输入您的密码)

确认密码* (请再次输入您的密码)

注册帐号

[About](#) - [Features](#) - [Agreement](#) - [Contact](#)

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

过手机客户端注册



2. 如何下载 eoeMarket 客户端软件?

通过 PC 安装

您可以通过网址: <http://www.eoemarket.com/download> 来下载客户端



通过手机浏览器进行安装

手机浏览器打开网址 <http://www.eoemarket.com/down> 进行安装。



3. 开发者如何上传软件?

注册后, 您可以选择成为开发者, 这样您就可以上传应用软件了。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

开发者管理面板
您个人的应用发布平台...

欢迎您, eoemobile | [首页](#) | [帐户](#) | [开发者面板](#) | [退出](#)

快捷面板
[管理面板](#)
[发布新应用](#)

应用相关
[发布新应用](#)
[应用列表](#)

反馈相关
[应用反馈](#)
[用户留言](#)

统计相关
[统计首页](#)
[应用统计](#)
[标签统计](#)
[技能统计](#)

账户相关
[我的帐户](#)
[收支明细表](#)
[帐户充值](#)
[提现申请](#)

step1、请上传你要发布应用的APK文件（需要签名。）

选择APK文件: No file chosen

发布新应用步骤

step1:上传APK文件

选择需要发布应用的APK文件，上传成功后可以智能分析出您的应用中诸如版本，包名，icon等信息，免去您手工输入；

step2:填写app信息

该步骤您会看到智能分析出的应用信息，除此之外，需要您输入应用的名字，描述，同时需要您选择分类，输入标签、价格等信息。

10.6 eoeMarket 软件推荐

从这期开始小编会为大家推荐 eoeMarket 的优秀软件。如果您认为哪个软件好用的话，也可以直接告诉小编，发邮件到 eoemarket@eoemobile.com

eoeMarket 致力于为用户提供最好的中文化软件，所以我们选择的软件必须都是支持中文的。在这里给大家推荐两款软件：

eoeAppInstaller(安装器)

eoe App 安装器(Installer)可以收集你 SD 上保存的全部.apk 文件,并可以直接安装该 app 或者删除这个 Apk 文件.这是目前小编见过的最好的 apk 安装器，强烈给大家推荐一下，从此后大家可以直接从 sdcard 上安装应用了。

[详细介绍](#)



eoeAppInstaller(安装器)

作者:eoemobile , 分类: [Application - Tools] , 发布时间: 2009-07-08 07:20:37

eoe App安装器(Installer)可以收集你SD上保存的全部.apk文件,并可以直接安装该app或者删除这个Apk文件.

本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！



按揭计算器

按揭计算器是一个很重用的工具，买房子前可一定要计算清楚每个月的还贷啊:)



按揭计算器

作者: titmobile, 分类: [Application - Tools], 发布时间: 2009-08-18 02:58:49
简单实用的按揭计算工具

本文档由 eoeAndroid 社区组织策划，整理及发布，版权所有，转载请保留！

最后小编还要提醒大家,如果大家在 eoeMarket 发现好的中文软件了,别忘了告诉小编一下,小编也去下载一下,给小编发邮件到 eoemarket@eoemobile.com。以后每一期的特刊小编会为大家分享好玩好用的中文软件。

eoeMarket 应用开发者介绍

从这一期起小编也会为大家介绍一些国内游戏的 Android 开发团队和个人。如果您是 Android 开发者并且有了作品的话,可以给小编来信,发邮件到 eoemarket@eoemobile.com。小编会将您推介给全国的 Android 用户,让更多的人了解您和您的作品。

第一期的话,给大家推荐国内一支著名的 Android 开发团队 [eoeMobile](#) 团队。

eoeMobile 是国内最早从事 Android 开发的技术团队之一,他们也出了大陆第一本 Android 的教程

[《Google Android 开发入门与实战》](#)

eoeMobile 制作了大量的中文化软件,包括国内著名网站豆瓣网客户端[伊豆](#)和饭否网的客户端[伊饭](#)大家在 eoeMarket 或者 google Market 通过搜索 eoe 关键字可以获取这些软件。

关于 eoeMobile 团队更详细的信息可以参看他们的官方网站 <http://www.eoemobile.com/>,另外小编发现他们的官方 blog 也不错,大家如果感兴趣的话可以经常去看看:)他们团队的 blog 是 <http://blog.eoemobile.com>。

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!

11.eoe 特刊小组诚邀您的加入 & 下期预告

这次特刊组于 2009-8-9 成立, 由小 E 把我们三人组织到一块, 并确定了主题为 NDK, 由于大家没有接触过, 花了比较多的时间, 2009-8-21 这天, 终于可以出第一个初版了。对于论坛中

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

fuchao01



发表于 13 小时前 | 只看该作者

都快一个月了，第七刊连影都没有

影都

如题

本人感到的特别抱歉，同时对网友对特刊如此高的关注感到高兴，希望我们特刊组没有让大家失望，我们将一如既往的保证特刊的品质。

游利卡废话又来了：

不要觉得，特刊只是少数人的舞台，只要你愿意展现自己，那 eoe 特刊永远向你敞开大门！

下期预告：

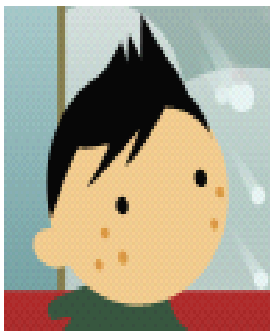
分享你开发中的小技巧！

虽然我们每一位程序人员都做着相同的工作，但是可能一个间歇中发笑的小技巧，则能让你事半功倍！

特刊这几期以来都是着力想办法把特刊做大、做强。所以特刊这次是一个回归，回到我们最普通的开发过程中来。

每个人，都肯定有自己的一些小技巧，每人分享一点！

12.介绍特刊组成员



向上: 负责特刊的编辑

论坛 ID:xusaomaiss

QQ:63590240

Email:xusaomaiss@gmail.com

个人签名: 坚持向上出品必为精品, 向上人生路。



明叔:

论坛 ID:fanth

QQ:1093148692

Email: clm16668@gmail.com



王华: 负责特刊推广及人员联系组织

论坛 ID: 情话 love

QQ: 526155779

个人签名: 活自己, 潇洒自己, 做我们自己的特刊。

特刊组刚刚成立, 无论是技术, 人员等等很多方面都还不完善, 我们相信由于你的加入, 为特刊贡献出自己的那份力量, 会使我们的特刊变的更加强大!

因你的加入我们的团队又强大一分!

-----情话 献上

如果有对 android 开发有兴趣的朋友, 或是对我们的特快有兴趣的朋友都可以联系我, 你就有机会在我们的特刊上崭露头角啊!

联系情话 MSN:qinghua3344521@msn.com

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

大家如何有关于特刊的任何问题,都可与我们联系,希望能得到大家对特刊的反馈,同时也可以告诉我们你的需求,我们将尽量做到最好,做到您满意!

13.其他

BUG 提交 如果你发现文档中翻译不妥的地方,请到如下地址反馈,我们会定期更新、发布更新后的版本

<http://www.eoeandroid.com/viewthread.php?tid=753>

资源下载: 本期文部分中包含的源码请在如下地址下载

<http://www.eoeandroid.com/viewthread.php?tid=753>

关于 eoeAndroid

当前,3G 商业,传统互联网与移动互联网也呈现出全业务发展的融合趋势,电信与互联网行业已经踏入继单机计算时代、传统互联网时代之后的第三个纪元。

由于看好移动互联网和 Android 手机平台的商业前景,同时也拥有专业而独特的产品、技术服务能力,我们聚集了一群热爱 Android 的技术英才,组建了 eoeMobile 团队。

eoeMobile 是一支专注于 Android 平台应用开发、产品运营和相关商业与技术服务的团队,立志于建立中国最大的 Android 应用开发专业社区 eoeAndroid.com,想为 Android 在中国的发展尽自己的微薄之力。

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!

14. 编后语

-游利卡

每到了这个时间，都是我感觉最好与最担心的时候。因为好多天的努力，终于在今天有成果了，在我们特刊一行人的努力下，终于做完了本期的 NDK。

大家可要了解，NDK 的资料在国内外可都是非常短缺的。这次 NDK 的特刊为大家奉献出的资料绝对值得一读。在国内找找还有别的，能比的上我们特刊的，我绝对敢说没有！

另外，从这期特刊开始，我们也开始走一些多元化的路线。我们 eoeMarket 软件平台会给大家推出一的新栏目，也绝对是为国内的开发者提供很大帮助的。

最后还是说一下，咱社区的很多丰富精彩的活动吧，只要翻到下一页就能看到。之前社区有提到过北京的线下活动，而这次我们可以喝 GTUG 联手，在谷歌中国的地方来举办我们的线下活动。

另外游利卡的“游戏诞生记”，喜欢游戏的朋友请多多捧场了！

最后还是那句话，如果你喜欢我们的特刊，同样希望和特刊的朋友一样流芳百世，那就赶紧加入我们 eoe 特刊小组吧！

15.eoeAndroid 社区最新动向:

A Android Hackathon 线下活动, 邀请大家参加

eoeAndroid 社区和 GTUG 联手组织举办 Android Hackathon 主题活动, 旨在通过该系列活动, 向更多的人介绍 Android 历史、技术、商业、发展趋势及相关话题, 引导更多人进入 Android 领域, 分享最新的技术和最有用的经验!

Beijing-GTUG 是由来自国内外的广大 Free Software、Open Source 爱好者自发组成的非盈利性质的技术交流组织, 讨论与谷歌产品开发和开源技术相关的话题。谷歌技术用户组定期举办线上和线下技术交流活动。我们希望能够帮助广大技术爱好者更好的学习、交流最流行的开源技术和 Google 软件开发技术。在谷歌技术用户组您能交到更多朋友, 学习最新的开源知识, 开阔您的眼界, 最近距离地接近 Google 公司, 体验 Google 公司的企业文化, 发现更适合您的事业机会。

2009 年 9 月 6 日下午, 将在 Google 中国举办 Android Hackathon 主题活动第 1 期, 我们诚挚邀请 Beijing-GTUG 会员、eoeAndroid 会员以及其他 android 技术爱好者参加这次活动; 齐聚各路高手, 分享 android 技术心得, 讨论 Android 发展大略。届时来宾有机会参与互动, 并有图书、纪念品赠送, 希望您届时大驾光临!

具体的活动详情: 请访问: <http://www.eoeandroid.com/viewthread.php?tid=2704>

B eoeExam 马上就要拉开帷幕了

一说起考试, 勾起了所有中国人民的心, 有人喜欢考试有人憎恨考试。其实游利卡也不想考试, 而且是恨透了考试的。但是有的时候发现, 对于一些我们用不到的东西, 如果没有一件事情驱动得让我们来驱动一下, 很容易就忘记了。想了好长时间, 发现只有万恶的考试最适合做这种事情。所以也有了 eoeExam 的诞生!

和咱社区的六大版主沟通了一下, 特别是 flylyke 叔, 发现大家也都很支持。所以:

从这期特刊发布后的不久, 在应用六大区, 将会举办一个考试的活动。

eoeExam 考务中心主任: flylyke
eoeExam 考试题目&考官&监考.....: 应用六大区的版主
eoeExam 考试心理咨询: 游利卡
eoeExam 考试作弊投诉中心: Iceskysl
eoeExam 考试费缴纳&拒绝缴纳: haiyangjy

我们的考试计划定于每周一考, 考试的题目现定于 Android 应用的开发。当然, 我们倡导轻松考试, 不建议因为考试增加爱大家的心理压力。所以请大家一定要调节好自己的心态, 不要因为考试压力过大, 而导致自己心理健康问题受影响。我们真不希望, 因为我们的考试而发生类似跳楼自杀等等惨烈的事情!

另外, 大家的考试成绩, 可以作为大家自己的鞭策。也可以用来炫耀了, 哈哈。

考试会尽量照顾大家大部分朋友, 但是也会有一些有难度的题目, 当然, 也会有一些比较雷人的题目。开心考试吗。

本文档由 eoeAndroid 社区组织策划, 整理及发布, 版权所有, 转载请保留!

好了,大家等着我们的考试吧!

C 游戏诞生记

今天不玩游戏,我们作游戏!

我的游戏我制作!

游戏制作人不再是梦想

我做你玩的游戏



在社区呆的时间长的朋友,肯定都知道我们游戏研究院的年度大戏——游戏诞生记!

而我们正在整合最核心的技术成员,现在机会难得,如果你是真正喜欢游戏,并且希望通过自己的双手和我们一起做成一款游戏,那你就应该加入我们。也许你就能成为未来各大公司抢手的游戏制作人,并能在中国游戏史上名流千古。

还是那句话

- 1.如果你喜欢游戏,你那应该加入我们
- 2.如果你想用自己的双手做成一款游戏,你更应该加入我们。
- 3.如果你想和我们一起并肩作战,在开放的 Android 平台,共同开发一款具有 PSP、NDS 水准的掌上游戏,那你怎么能不加入我们呢?

eoeAndroid 社区与 Gphone 中文网联合推出的“游戏诞生记”邀您加入!

我们的讨论群:4276804

三言两语,说不完这件事情。如果看到这个帖子心动了,那最好的办法就是访问
eoeAndroid 专区 <http://www.eoeandroid.com/forumdisplay.php?fid=47>
Gphone 中文网专区: <http://forum.tgbus.com/forumdisplay.php?fid=230>

本文档由 eoeAndroid 社区组织策划,整理及发布,版权所有,转载请保留!