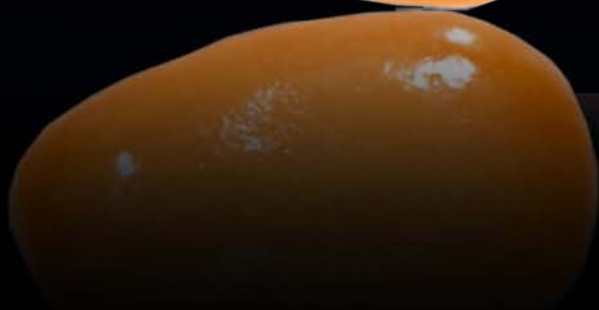


移动开发经典丛书

构建图形丰富与更佳性能的原生应用



Pro Android C++ with the NDK

Android C++ 高级编程——使用NDK

[美] Onur Cinar
于红 余建伟 冯艳红 著
译



Apress®

清华大学出版社

移动开发经典丛书

Android C++高级编程 ——使用 NDK

[美] Onur Cinar 著
于红 余建伟 冯艳红 译

清华大学出版社

北 京

Onur Cinar

Pro Android C++ with the NDK

EISBN: 978-1-4302-4827-9

Original English language edition published by Apress Media. Copyright © 2012 by Apress Media.
Simplified Chinese-Language edition copyright © 2013 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2013-4603

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

Android C++高级编程——使用 NDK / (美) 辛纳 (Cinar,O.) 著；于红，余建伟，冯艳红 译.
—北京：清华大学出版社，2014
(移动开发经典丛书)

书名原文：Pro Android C++ with the NDK

ISBN 978-7-302-34301-1

I. ①A… II. ①辛… ②于… ③余… ④冯… III. ①移动终端—应用程序—程序设计
IV. ①TN929.53

中国版本图书馆 CIP 数据核字(2013)第 251610 号

责任编辑：王 军 于 平

装帧设计：牛艳敏

责任校对：邱晓玉

责任印制：

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

装 订 者：

经 销：全国新华书店

开 本：185mm×260mm 印 张：22.5 字 数：548 千字

版 次：2014 年 1 月第 1 版 印 次：2014 年 1 月第 1 次印刷

印 数：1~4000

定 价：59.80 元

产品编号：

译者序

Android 是 Google 公司基于 Linux 平台开发的开源手机操作系统，自然要对 C、C++ 提供原生的支持。通过 Google 发布的 Android 手机 NDK(Native Development Kit)，应用程序可以非常方便地实现 Java 与 C/C++ 代码的相互沟通。合理地使用 NDK，可以提高应用程序的执行效率。所以，对于 Android 开发人员来说，NDK 是必须掌握的工具。

本书的作者 Onur Cinar 在美国宾州费城 Drexel 大学获得计算机科学理学学士学位。他有 17 年的移动通信领域大规模复杂软件项目的设计、开发和管理经验。自 Android 平台问世以来，Onur Cinar 一直积极从事 Android 开发工作，目前在微软 Skype 分部担任 Android 平台的 Skype 客户端高级产品工程经理。出版了多部 Android 开发应用方面的图书。

本书提供了 Android NDK 开发的全面信息，介绍了从 NDK 开发环境搭建的每一步细节，NDK 的基本概念和体系结构，具体的开发流程和方法。同时还比较详尽地介绍了 Android NDK 对 C、C++ 标准库的支持。是一本关于 Android NDK 开发的全新入门指南。

本书译者团队具有丰富的系统设计与开发经验。于红在计算机应用技术领域工作 20 余年，承担 Java 语言程序设计、Visual Basic 等程序设计、操作系统等课程的教学任务，熟悉程序设计类课程的教学规律，具有较强的语言组织和语言表达能力，在国内外期刊及知名国际会议上发表论文 40 余篇，其中被三大检索收录 22 篇，出版教材 3 部。余建伟先后就职于腾讯(大连)无线研发中心和东软(大连)集团有限公司。主要从事移动互联网应用开发以及嵌入式产品的设计和开发。从 2010 年至今一直从事 Android 应用与游戏开发和 Framework 内核以及 Android 底层开发技术的研究，对 Android 内核有较为深刻的理解。此外对移动互联网产品交互设计和产品运营也有一定的研究。目前已出版一本译著《Android 4 高级编程(第 3 版)》。冯艳红从事计算机应用技术教学及研究工作 7 年多，主要承担 Java 语言程序设计、C 语言程序设计、C++ 面向对象的程序设计等课程的教学工作，参与了多个项目的开发，具有较强的理论基础和程序开发经验。

于红负责翻译第 3~10 章，余建伟负责翻译第 11~14 章，冯艳红负责翻译第 1 章、第 2 章、前言、内封、封底等。参与翻译活动的还有孙庚、王芳、黄璐、史鹏辉、崔春雷、何南、孙京恩、王美妮；另外李青、孔亮亮、王宁三位同学协助完成本书部分内容的翻译，没有他们的帮助不可能完成本书的翻译工作，在此对他们表示衷心的感谢。由于时间仓促、译者的水平及精力有限，一定存在谬误，希望读者们多多批评指正。

译者

作者简介



Onur Cinar 有超过 17 年的移动和通信领域大规模复杂软件项目的设计、开发和管理经验。他的专业技能包括 VoIP、视频通信、移动应用程序、网格计算和不同平台上的网络技术。从 Android 平台问世他就一直积极从事这方面的工作。他是 Apress 出版的 *Android Apps with Eclipse* 一书的作者。他在美国宾州费城 Drexel 大学获得计算机科学理学学士学位。现就职于微软 Skype 分部，任 Android 平台的 Skype 客户端高级产品工程经理。

技术审校者简介



Grant Allen 在 IT 领域工作了 20 年，主要职位是 CTO、企业架构师和数据库架构师。他曾经在全球各地的私企、学术界和政府部门工作过，专长是进行全球通用的系统设计、开发和性能优化。他经常在产业界和学术界的会议上发言，涉及的主题从数据挖掘到兼容性，以及诸如数据库(DB2、Oracle、SQL Server 以及 MySQL)，内容管理，协作，颠覆性的创新和像 Android 这样的移动生态系统之类的技术。

他的第一个 Android 应用程序是一个提醒他完成所有其他未完成的 Android 项目的任务列表。

Grant 在谷歌工作，利用空闲时间攻读博士学位，博士论文的研究题目是构建创新性的高技术环境。

他是 *Beginning DB2: From Novice to Professional* (Apress, 2008)的作者，*Oracle SQL Recipes: A Problem-Solution Approach* (Apress, 2010)和 *The Definitive Guide to SQLite, 2nd Edition* (Apress, 2010)的主编。

前言

Android 是移动电话市场的主要角色而且其市场份额正在持续增长。它是第一个完整的、开放的、免费的移动平台，该平台给移动应用开发者提供了无限的机会。

虽然 Android 平台的官方程序语言是 Java，但应用开发者不限于仅使用 Java 技术。

Android 允许应用开发者通过 Android 原生开发包(NDK)使用诸如 C 和 C++之类的原生代码语言实现他们的部分应用。本书中我们将学习如何用 Android NDK 通过原生代码语言去实现自己的 Android 应用中对性能要求较高的部分。

本书介绍了原生应用开发、可用的原生 API 以及故障排除技术的详细叙述，包括用按步骤的指导和屏幕截图以帮助 Android 开发人员迅速达到开发原生应用的目的。

主要内容：

- 在主要的操作系统上安装 Android 原生开发环境。
- 使用 Eclipse 集成开发环境开发原生代码。
- 使用 Java 原生接口(JNI)将原生代码与 Java 代码连接。
- 用 SWIG 自动生成 JNI 代码。
- 用 POSIX 和 Java 线程开发多线程原生应用。
- 用 POSIX sockets 开发网络原生应用。
- 用 logging、GDB 和 Eclipse 调试器调试原生代码。
- 用 Valgrind 分析内存问题。
- 用 GProf 测试应用性能。
- 用 SIMD/NEON 优化原生代码。

下载代码

读者可以在 www.apress.com 上下载本书源代码。

联系作者

读者可以通过作者的 Android C++ with the NDK 网站 <http://www.zdo.com/android-c++-with-the-ndk> 联系作者。

目 录

第 1 章	Android 平台上的 C++入门	1
1.1	Microsoft Windows	1
1.1.1	在 Windows 平台上下载并安装 JDK 开发包	2
1.1.2	在 Windows 平台上下载并安装 Apache ANT	5
1.1.3	在 Windows 平台上下载并安装 Android SDK	7
1.1.4	在 Windows 平台上下载并安装 Cygwin	8
1.1.5	在 Windows 平台上下载并安装 Android NDK	11
1.1.6	在 Windows 平台上下载并安装 Eclipse	13
1.2	Apple Mac OS X	14
1.2.1	在 Mac 平台上安装 Xcode	14
1.2.2	验证 Mac 平台的 Java 开发包	15
1.2.3	验证 Mac 平台上的 Apache ANT	15
1.2.4	验证 GNU Make	16
1.2.5	在 Mac 平台上下载并安装 Android SDK	16
1.2.6	在 Mac 平台上下载并安装 Android NDK	18
1.2.7	在 Mac 平台上下载并安装 Eclipse	19
1.3	Ubuntu Linux	20
1.3.1	检查 GNU C 库版本	20
1.3.2	激活在 64 位系统上支持 32 位的功能	21

1.3.3	在 Linux 平台上下载并安装 Java 开发工具包(JDK)	21
1.3.4	在 Linux 平台上下载并安装 Apache ANT	22
1.3.5	在 Linux 平台上下载并安装 GNU Make	22
1.3.6	在 Linux 平台上下载并安装 Android SDK	23
1.3.7	在 Linux 平台上下载并安装 Android NDK	24
1.3.8	在 Linux 平台上下载并安装 Eclipse	25
1.4	下载并安装 ADT	26
1.4.1	安装 Android 平台包	29
1.4.2	配置模拟器	30
1.5	小结	33
第 2 章	深入了解 Android NDK	35
2.1	Android NDK 提供的组件	35
2.2	Android NDK 的结构	36
2.3	以一个示例开始	36
2.3.1	指定 Android NDK 的位置	37
2.3.2	导入示例项目	37
2.3.3	向项目中添加原生支持	39
2.3.4	运行项目	40
2.3.5	用命令行对项目进行构建	41
2.3.6	检测 Android NDK 项目的结构	42
2.4	构建系统	42
2.4.1	Android.mk	43
2.4.2	Application.mk	53

2.5	使用 NDK-Build 脚本	54	4.3.1	接口文件	86
2.6	排除构建系统故障	55	4.3.2	在命令行方式下调用 SWIG	89
2.7	小结	56	4.3.3	将 SWIG 集成到 Android 构建过程中	90
第 3 章	用 JNI 实现与原生代码通信	57	4.3.4	更新 Activity	92
3.1	什么是 JNI	57	4.3.5	执行应用程序	93
3.2	以一个示例开始	57	4.3.6	剖析生成的代码	93
3.2.1	原生方法的声明	58	4.4	封装 C 语言代码	94
3.2.2	加载共享库	58	4.4.1	全局变量	94
3.2.3	实现原生方法	59	4.4.2	常量	95
3.3	数据类型	64	4.4.3	只读变量	96
3.3.1	基本数据类型	64	4.4.4	枚举	97
3.3.2	引用类型	64	4.4.5	结构体	100
3.4	对引用数据类型的操作	65	4.4.6	指针	101
3.4.1	字符串操作	65	4.5	封装 C++ 代码	101
3.4.2	数组操作	67	4.5.1	指针、引用和值	102
3.4.3	NIO 操作	68	4.5.2	默认参数	103
3.4.4	访问域	69	4.5.3	重载函数	104
3.4.5	调用方法	71	4.5.4	类	104
3.4.6	域和方法描述符	72	4.6	异常处理	106
3.5	异常处理	75	4.7	内存管理	107
3.5.1	捕获异常	75	4.8	从原生代码中调用 Java	108
3.5.2	抛出异常	75	4.8.1	异步通信	108
3.6	局部和全局引用	76	4.8.2	启用 Directors	109
3.6.1	局部引用	76	4.8.3	启用 RTTI	109
3.6.2	全局引用	76	4.8.4	重写回调方法	109
3.6.3	弱全局引用	77	4.8.5	更新 HelloJni Activity	110
3.7	线程	78	4.9	小结	110
3.7.1	同步	78	第 5 章	日志、调试及故障处理	111
3.7.2	原生线程	79	5.1	日志	111
3.8	小结	79	5.1.1	框架	111
第 4 章	使用 SWIG 自动生成 JNI 代码	81	5.1.2	原生日志 API	112
4.1	什么是 SWIG	81	5.1.3	受控制的日志	114
4.2	安装	82	5.1.4	控制台日志	118
4.2.1	Windows 平台上 SWIG 的 安装	82	5.2	调试	119
4.2.2	在 Mac OS X 下安装	83	5.2.1	预备知识	119
4.2.3	在 Ubuntu Linux 下安装	85	5.2.2	调试会话建立	120
4.3	通过示例程序试用 SWIG	86	5.2.3	建立调试示例	121

5.2.4 启动调试器.....	121	第 7 章 原生线程.....	155
5.3 故障处理.....	126	7.1 创建线程示例项目.....	155
5.3.1 堆栈跟踪分析.....	127	7.1.1 创建 Android 项目.....	155
5.3.2 对 JNI 的扩展检查.....	128	7.1.2 添加原生支持.....	157
5.3.3 内存问题.....	130	7.1.3 声明字符串资源.....	157
5.3.4 strace.....	133	7.1.4 创建简单的用户界面.....	157
5.4 小结.....	134	7.1.5 实现 Main Activity.....	159
第 6 章 Bionic API 入门.....	135	7.1.6 生成 C/C++ 头文件.....	162
6.1 回顾标准库.....	135	7.1.7 实现原生函数.....	163
6.2 还有另一个 C 库.....	136	7.1.8 更新 Android.mk 构建脚本.....	165
6.2.1 二进制兼容性.....	136	7.2 Java 线程.....	165
6.2.2 提供了什么.....	136	7.2.1 修改示例应用程序使之能够 使用 Java 线程.....	165
6.2.3 缺什么.....	137	7.2.2 执行 Java Threads 示例.....	166
6.3 内存管理.....	137	7.2.3 原生代码使用 Java 线程的 优缺点.....	167
6.3.1 内存分配.....	137	7.3 POSIX 线程.....	168
6.3.2 C 语言的动态内存管理.....	138	7.3.1 在原生代码中使用 POSIX 线程.....	168
6.3.3 C++ 的动态内存管理.....	139	7.3.2 用 pthread_create 创建线程.....	168
6.4 标准文件 I/O.....	141	7.3.3 更新示例应用程序以 使用 POSIX 线程.....	169
6.4.1 标准流.....	141	7.3.4 执行 POSIX 线程示例.....	174
6.4.2 使用流 I/O.....	141	7.4 从 POSIX 线程返回结果.....	174
6.4.3 打开流.....	142	7.5 POSIX 线程同步.....	176
6.4.4 写入流.....	143	7.5.1 用互斥锁同步 POSIX 线程.....	176
6.4.5 流的读取.....	145	7.5.2 使用信号量同步 POSIX 线程.....	180
6.4.6 搜索位置.....	148	7.6 POSIX 线程的优先级和 调度策略.....	180
6.4.7 错误检查.....	149	7.6.1 POSIX 的线程调度策略.....	181
6.4.8 关闭流.....	149	7.6.2 POSIX Thread 优先级.....	181
6.5 与进程交互.....	150	7.7 小结.....	181
6.5.1 执行 shell 命令.....	150	第 8 章 POSIX Socket API: 面向 连接的通信.....	183
6.5.2 与子进程通信.....	150	8.1 Echo Socket 示例应用.....	183
6.6 系统配置.....	151	8.1.1 Echo Android 应用项目.....	184
6.6.1 通过名称获取系统属性值.....	152	8.1.2 抽象 echo activity.....	184
6.6.2 通过名称获取系统属性.....	152		
6.7 用户和组.....	153		
6.7.1 获取应用程序用户和组 ID.....	153		
6.7.2 获取应用程序用户名.....	154		
6.8 进程间通信.....	154		
6.9 小结.....	154		

8.1.3	echo 应用程序字符串资源	188	10.3.1	创建本地 Socket: socket	237
8.1.4	原生 echo 模块	188	10.3.2	将本地 socket 与 Name 绑定: bind	238
8.2	用 TCP sockets 实现面向连接的 通信	191	10.3.3	接受本地 Socket: accept	240
8.2.1	Echo Server Activity 的 布局	192	10.3.4	原生本地 Socket Server	240
8.2.2	Echo Server Activity	193	10.4	将本地 Echo Activity 添加到 Manifest 中	242
8.2.3	实现原生 TCP Server	194	10.5	运行本地 Sockets 示例	243
8.2.4	Echo 客户端 Activity 布局	206	10.6	异步 I/O	243
8.2.5	Echo 客户端 Activity	208	10.7	小结	244
8.2.6	实现原生 TCP 客户端	210	第 11 章	支持 C++	245
8.2.7	更新 Android Manifest	213	11.1	支持的 C++运行库	245
8.2.8	运行 TCP Sockets 示例	214	11.1.1	GAbi++ C++运行库	246
8.3	小结	217	11.1.2	STLport C++运行库	246
第 9 章	POSIX Socket API: 无连接的 通信	219	11.1.3	GNU STL C++运行库	246
9.1	将 UDP Server 方法添加到 Echo Server Activity 中	219	11.2	指定 C++运行库	246
9.2	实现原生 UDP Server	220	11.3	静态运行库与动态运行库	247
9.2.1	创建 UDP Socket: socket	220	11.4	C++异常支持	247
9.2.2	从 Socket 接收数据报: recvfrom	221	11.5	C++ RTTI 支持	248
9.2.3	向 Socket 发送数据报: sendto	223	11.6	C++标准库入门	249
9.2.4	原生 UDP Server 方法	224	11.6.1	容器	249
9.3	将原生 UDP Client 方法加入 Echo Client Activity 中	225	11.6.2	迭代器	250
9.4	实现原生 UDP Client	226	11.6.3	算法	251
9.5	运行 UDP Sockets 示例	228	11.7	C++运行库的线程安全	251
9.5.1	连通 UDP 的模拟器	228	11.8	C++运行库调试模式	251
9.5.2	启动 Echo UDP Client	229	11.8.1	GNU STL 调试模式	251
9.6	小结	229	11.8.2	STLport 调试模式	252
第 10 章	POSIX Socket API: 本地通信	231	11.9	小结	253
10.1	Echo Local Activity 布局	231	第 12 章	原生图形 API	255
10.2	Echo Local Activity	232	12.1	原生图形 API 的可用性	255
10.3	实现原生本地 Socket Server	237	12.2	创建一个 AVI 视频播放器	256
			12.2.1	将 AVILib 作为 NDK 的 一个导入模块	256
			12.2.2	创建 AVI 播放器 Android 应用程序	258
			12.2.3	创建 AVI Player 的 Main Activity	258

12.2.4	创建 Abstract Player Activity.....	262
12.3	使用 JNI 图形 API 进行 渲染	269
12.3.1	启用 JNI Graphics API.....	269
12.3.2	使用 JNI Graphics API.....	270
12.3.3	用 Bitmap 渲染来更新 AVI Player	271
12.3.4	运行使用 Bitmap 渲染的 AVI Player	278
12.4	使用 OpenGL ES 渲染	279
12.4.1	使用 OpenGL ES API	279
12.4.2	启用 OpenGL ES 1.x API	279
12.4.3	启用 OpenGL ES 2.0 API	280
12.4.4	用 OpenGL ES 渲染来 更新 AVI Player.....	280
12.5	使用原生 Window API 进行 渲染	290
12.5.1	启用原生 Window API	290
12.5.2	使用原生 Window API	291
12.5.3	用原生 window 渲染器来 更新 AVI Player.....	293
12.5.4	EGL 图形库	301
12.6	小结	301
第 13 章	原生音频 API	303
13.1	使用 OpenSL ES API	303
13.1.1	与 OpenSL ES 标准的 兼容性	304
13.1.2	音频许可	304
13.2	创建 WAVE 音频播放器	304
13.2.1	将 WAVELib 作为 NDK 导入模块	304
13.2.2	创建 WAVE 播放器 Android 应用程序	306
13.2.3	创建 WAVE 播放器 主 Activity	306
13.2.4	实现 WAVE Aduio 播放	310
13.3	运行 WAVE Audio Player	327
13.4	小结	328
第 14 章	程序概要分析和 NEON 优化	329
14.1	用 GNU Profiler 度量性能	329
14.1.1	安装 Android NDK Profiler	329
14.1.2	启用 Android NDK Profiler	330
14.1.3	使用 GNU Profiler 分析 gmon.out 文件	331
14.2	使用 ARM NEON Intrinsics 进行优化	332
14.2.1	ARM NEON 技术概述	333
14.2.2	给 AVI Player 添加一个 亮度过滤器	333
14.2.3	为 AVI 播放器启用 Android NDK Profiler	336
14.2.4	AVI Player 程序概要 分析	337
14.2.5	使用 NEON Intrinsics 优化 Brightness Filter	338
14.3	自动向量化	342
14.3.1	启用自动向量化	343
14.3.2	自动向量化问题的发现和 排除	344
14.4	小结	344

第 1 章

Android 平台上的 C++入门

毋庸置疑，探索和实践是学习的最佳方法。本书一开始就为读者讲解功能完备的开发环境，使读者可以在以后各章的学习过程中用实例进行探索和实验。Android C++开发环境主要由以下几部分构成：

- Android 软件开发包(Software Development Kit, SDK)
- Android 原生开发包(Native Development Kit, NDK)
- Eclipse 上的 Android 开发工具(Android Development Tools, ADT)插件
- Java 开发包(Java Development Kit, JDK)
- Apache ANT 构建系统
- GNU Make 构建系统
- Eclipse IDE

本章循序渐进地讲解正确配置 Android C++开发环境的步骤，Android 开发工具可以在以下三种操作系统平台上运行：

- Microsoft Windows
- Apple Mac OS X
- Linux

由于不同操作系统的需求和安装步骤差异较大，因此下面将分别阐述不同操作系统上 Android C++开发环境的安装步骤，可以跳过你不使用的操作系统。

1.1 Microsoft Windows

Android 开发工具可以在 Windows XP(仅限于 32 位)、Vista 或 Windows 7 中运行。在本节中你需要下载并安装以下组件：

- Java JDK 6
- Apache ANT 构建系统
- Android SDK

- Cygwin
- Android NDK
- Eclipse IDE

1.1.1 在 Windows 平台上下下载并安装 JDK 开发包

Android 开发工具要求必须安装 JDK(Java Development Kit), 不能只安装 JRE(Java Runtime Edition), 在安装 Android 开发工具之前需要先安装 Java JDK 6。

注意:

为遵守上述版本号, Android 开发工具只支持版本 5 或者 6 的 Java 编译器。用 JDK 6 更简单且不易出错。

Android 开发工具支持多种发行版本的 JDK, 例如: IBM JDK、Open JDK 以及 Oracle JDK(即以前的 Sun JDK)。因为 Oracle JDK 支持的平台较多, 本书使用 Oracle JDK 为例进行讲解。

请访问 www.oracle.com/technetwork/java/javase/downloads/index.html 网站, 按照以下步骤下载 Oracle JDK:

(1) 如图 1-1 所示, 单击 JDK 6 下载按钮开始下载, 本书编写时最新版本的 Oracle JDK 6 是 Update 33。

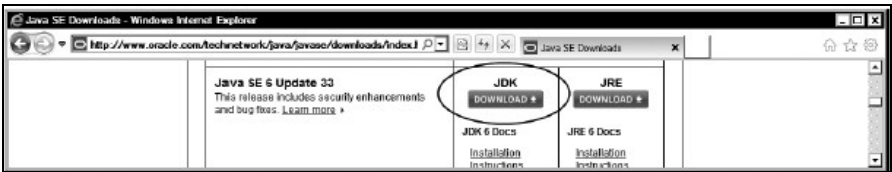


图 1-1 Oracle JDK 6 下载按钮

- (2) 单击 Oracle JDK 6 Download 按钮之后进入支持平台的 Oracle JDK 6 安装包清单页面。
- (3) 选中 Accept License Agreement 选项并下载 Windows x86 安装包, 如图 1-2 所示。

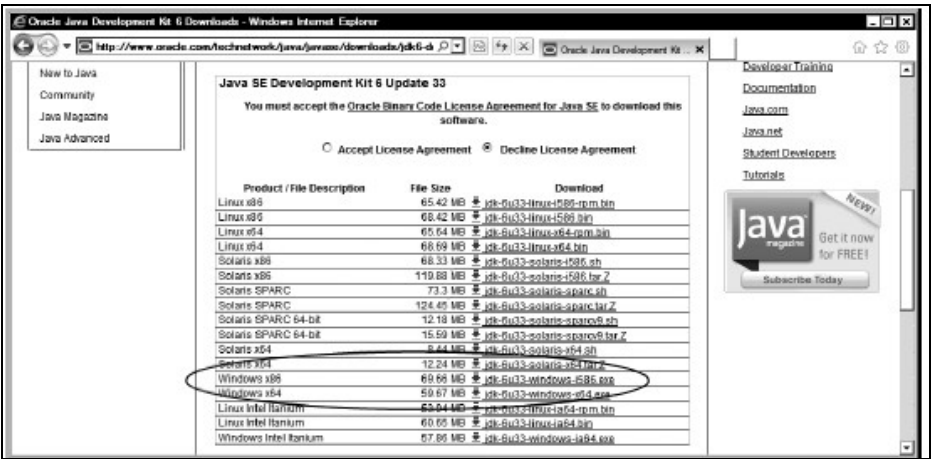


图 1-2 下载 Windows x86 下的 Oracle JDK 6

现在可以安装了。Windows 平台下的 Oracle JDK 6 安装包带有图形安装向导，它将指导你完成 JDK 的安装。安装向导首先安装 JDK，然后安装 JRE。在安装过程中，安装向导会让你指定安装目录以及要安装的组件，如图 1-3 所示。当然，此处可以使用默认值。此时，要记住 JDK 的安装目录以备环境变量设置时使用。

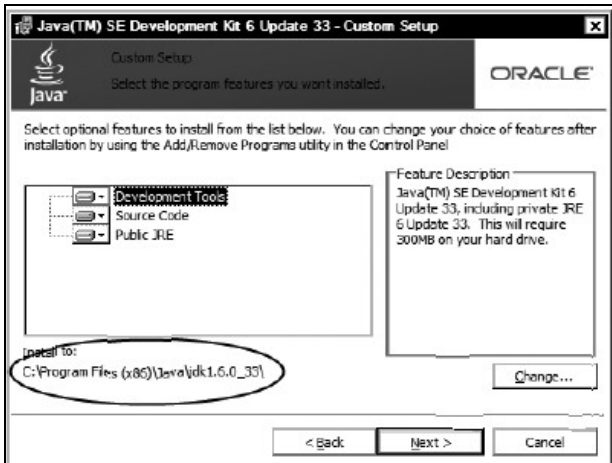


图 1-3 Oracle JDK 6 安装目录

安装完成时 JDK 准备就绪，安装向导不会自动将 Java 二进制目录加入系统可执行文件搜索路径，即 PATH 环境变量中，这一步要在 JDK 安装的最后一步手工完成：

- (1) 在 Start 按钮菜单中选择 Control Panel。
- (2) 单击 System 图标进入 System Properties 对话框。
- (3) 单击 Advanced 选项卡，然后单击此选项卡中的 Environment Variables 按钮，如图 1-4 所示。

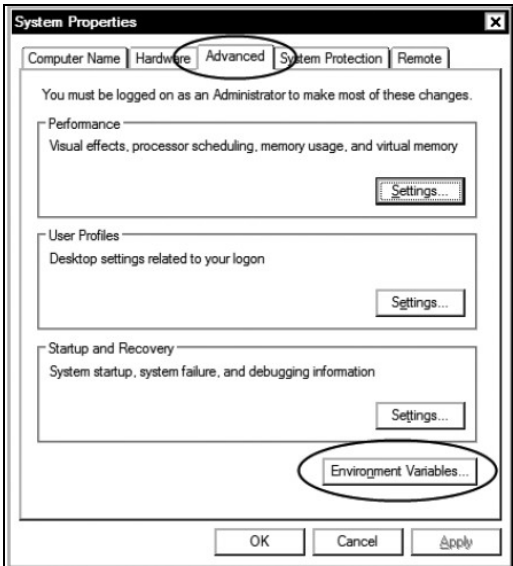


图 1-4 System Properties 对话框

(4) 单击 Environment Variables 按钮将启动 Environment Variables 对话框，该对话框由两部分构成：上面是 User Environment Variables(用户环境变量)，下面是 System Environment Variables(系统环境变量)。

(5) 在系统变量部分单击 New 按钮定义新的环境变量，如图 1-5 所示。

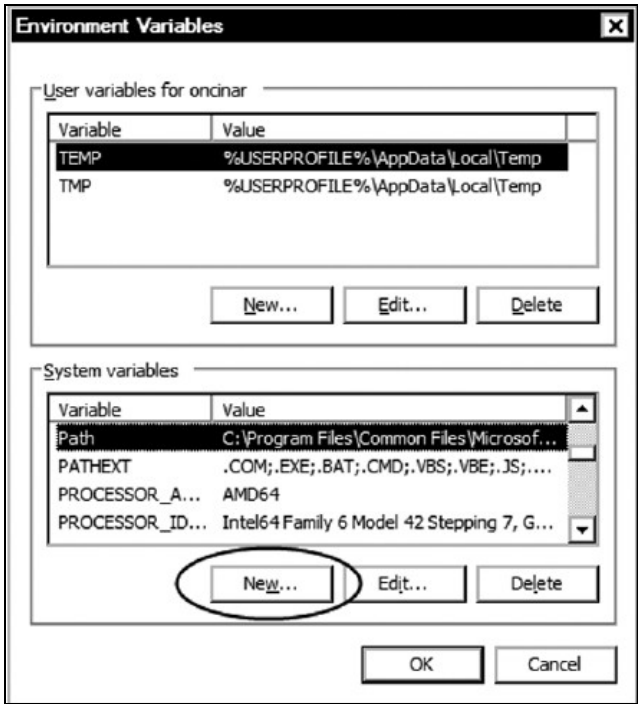


图 1-5 Environment Variables 对话框

(6) 将变量名设置成 JAVA_HOME，变量值设置成在前面的安装过程中记录下来的 Oracle JDK 的安装目录，如图 1-6 所示。

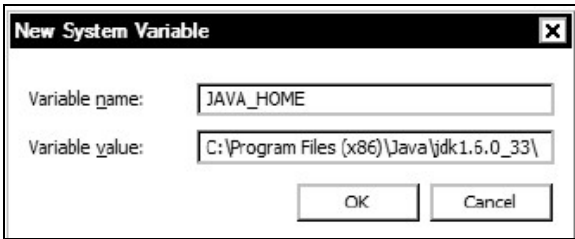


图 1-6 新的 JAVA_HOME 环境变量

(7) 单击 OK 按钮保存环境变量。

(8) 在系统变量列表中，双击 PATH 变量，并将;%JAVA_HOME%\bin 追加到变量值后面，如图 1-7 所示。

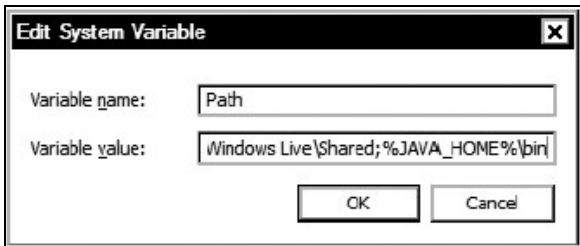


图 1-7 将 Oracle JDK 二进制路径追加到系统 PATH 变量中

现在 Oracle JDK 成为系统可执行文件搜索路径的一部分了，且该地址很容易找到。为了验证安装是否成功，选择 Start | Accessories | Command Prompt，打开一个命令提示窗口，在命令提示符下执行 `javac -version`。如果安装成功，就会看到 Oracle JDK 版本号，如图 1-8 所示。



图 1-8 Oracle JDK 安装的有效性验证

1.1.2 在 Windows 平台上下下载并安装 Apache ANT

Apache ANT 是命令行构建工具，其任务是驱动根据目标和任务所描述的任何类型过程。Android 开发工具要求安装 Apache ANT 1.8 及以后版本，在本书编写时，最新版本是 Apache ANT 1.8.4。

请访问 <http://ant.apache.org/bindownload.cgi> 网站下载 Apache ANT，下载安装包为 ZIP 格式，如图 1-9 所示。安装步骤如下：

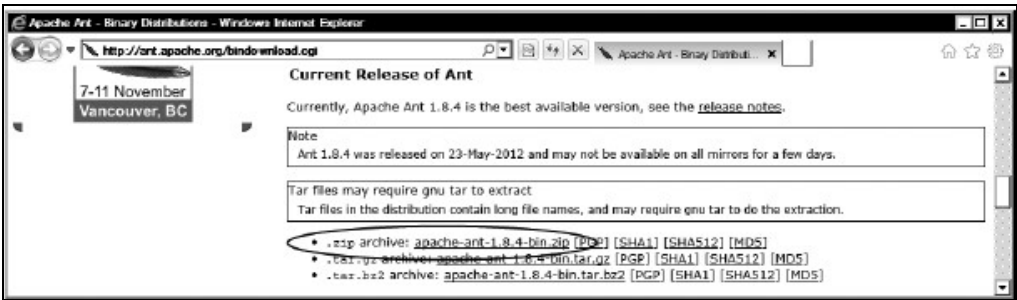


图 1-9 ZIP 格式的 Apache ANT 下载安装包

- (1) Windows 操作系统支持 ZIP 文件，当下载完成时右击该 ZIP 文件。
- (2) 在上下文菜单中选择 Extract All 打开 Extract Compressed Folder 向导。
- (3) 单击 Browse 按钮，选择目标目录，如图 1-10 所示。因为 ZIP 文件已经包含一个名为 `apache-ant-1.8.4` 的目录用来保存 Apache ANT 文件，因此不需要建立专用的空白目标目录。本书中 `C:\android` 目录是用来保存 Android 开发工具及其相关工具的根目录，要记

住目标目录名以备后面设置环境变量时使用。

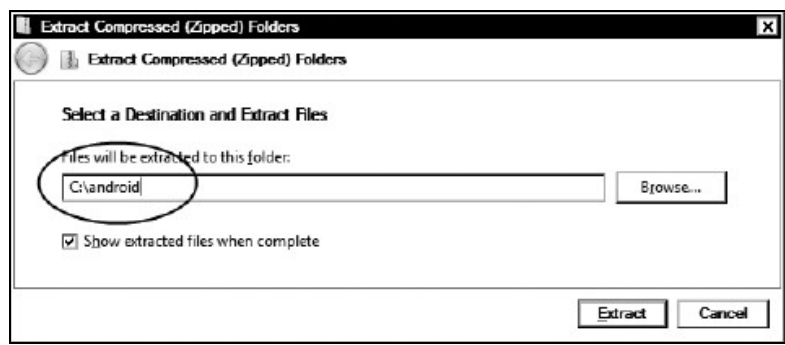


图 1-10 解压缩 Apache ANT ZIP 文档

(4) 单击 Extract 按钮安装 Apache ANT。

Apache ANT 安装完成后，按以下步骤将其二进制路径追加到系统可执行文件搜索路径中：

- (1) 在 System Properties 界面打开 Environment Variables 对话框。
- (2) 在系统变量部分单击 New 按钮定义一个新的环境变量。
- (3) 将变量名设置成 ANT_HOME ， 变量值设置成前面记下的 Apache ANT 安装目录 (例如 C:\android\apache-ant-1.8.4)， 如图 1-11 所示。

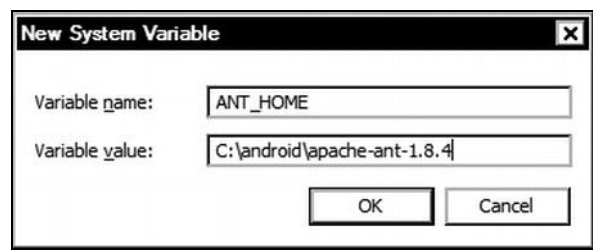


图 1-11 新建 ANT_HOME 环境变量

- (4) 单击 OK 按钮保存新的环境变量。
- (5) 在系统变量列表中，双击 PATH 变量，将;% ANT_HOME%\bin 追加到变量值后面， 如图 1-12 所示。

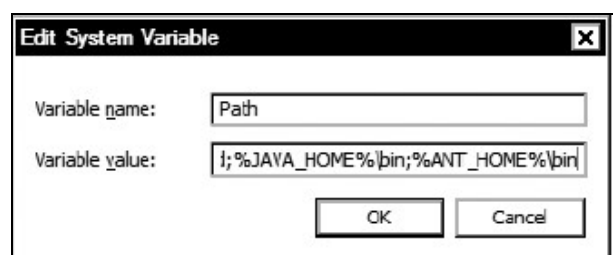


图 1-12 将 Apache ANT 二进制路径追加到系统 PATH 变量中

安装完成后，Apache ANT 被添加到系统可执行文件搜索路径中。为了验证安装是否成功，打开一个命令提示窗口。在命令提示符下执行 ant -version。如果安装成功，就会看到 Apache ANT 版本号，如图 1-13 所示。

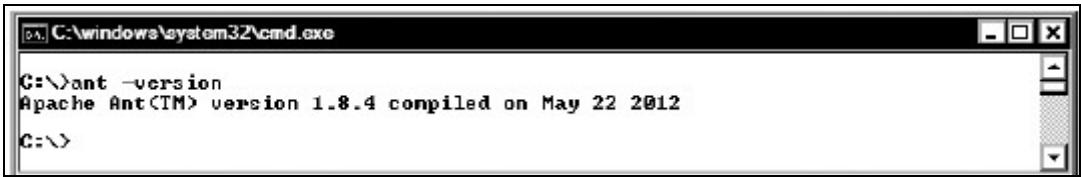


图 1-13 Apache ANT 安装的有效性验证

1.1.3 在 Windows 平台上下载并安装 Android SDK

Android 软件开发包(SDK)是开发工具链的核心组件，它提供框架 API 库和构建、测试及调试 Android 应用所需要的开发人员工具。

访问 <http://developer.android.com/sdk/index.html> 网站下载 Android SDK。本书编写时，Android SDK 的最新版本是 R20。当前提供两种类型的安装包：图形安装程序和 ZIP 文档。尽管图形安装程序被认为是主要的安装包，但是众所周知，在某些平台上它还存在问题。单击链接 Other platforms 并下载 Android SDK ZIP 文档，如图 1-14 所示。然后按照以下步骤进行操作：



图 1-14 Android SDK 下载页

(6) 下载完成后，右击 ZIP 文件，在上下文菜单中选择 Extract All 打开 Extract Compressed Folder 向导。

(7) 单击 Browse 按钮，选择目标目录。因为 ZIP 文件中已经包含一个名为 android-sdk-windows 的子目录，且其中包含 Android SDK 文件，因此不需要创建专用的空白目标目录。要记住目标目录名以备后面设置环境变量时使用。

(8) 单击 Extract 按钮安装 Android SDK。

安装完成后，按以下步骤将 Android SDK 的二进制路径追加到系统可执行文件搜索路径中。

- (1) 在 System Properties 界面打开 Environment Variables 对话框。
- (2) 在系统变量部分单击 New 按钮定义一个新的环境变量。

(3) 将变量名设置成 ANDROID_SDK_HOME,将变量值设置成前面记下的 Android SDK 安装目录(例如 C:\android\android-sdk-windows), 如图 1-15 所示。

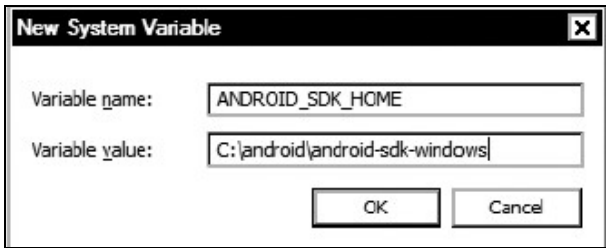


图 1-15 ANDROID_SDK_HOME 环境变量

(4) 单击 OK 按钮保存新的环境变量。
(5) 有三个重要的目录需要添加到系统可执行搜索路径中: SDK根目录、保存Android 平台独立SDK工具的工具目录和保存Android的平台工具目录, 不考虑不存在平台工具 目录的情况。在Environment Variables对话框中的系统变量列表中, 双击PATH变量并将 ;%ANDROID_SDK_HOME%; %ANDROID_SDK_HOME%\tools; %ANDROID_SDK_HOME%\ platform-tools追加到变量值后面, 如图 1-16 所示。

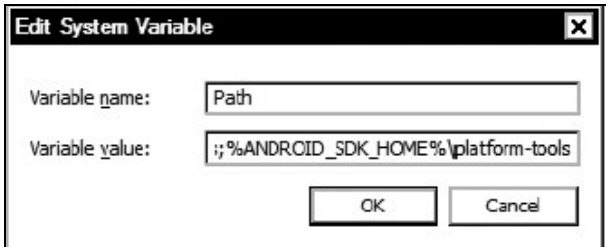


图 1-16 将 Android SDK 二进制路径追加到系统 PATH 变量中

为了验证安装是否成功, 打开一个命令提示窗口, 在命令提示符下执行'SDK Manager'(命令中包括引号)。如果安装成功, 就会看到 Android SDK Manager 管理器, 如 图 1-17 所示。

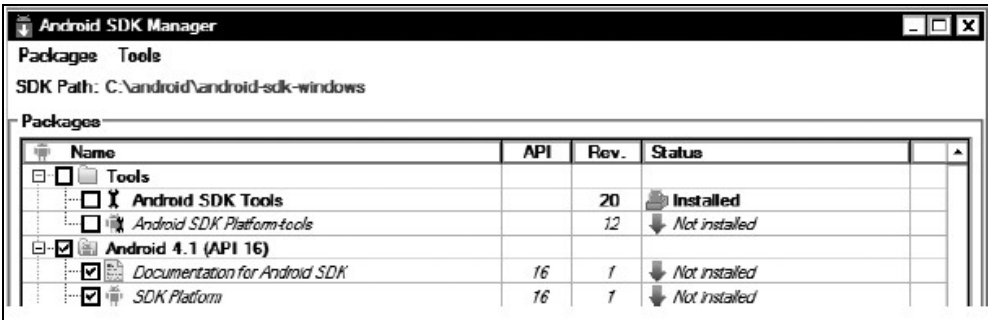


图 1-17 Android SDK Manager 应用

1.1.4 在 Windows 平台上下载并安装 Cygwin

Android原生开发包工具(Android Native Development Kit, NDK)最初设计在类UNIX系

统上工作，NDK的一些组件是shell脚本，这些脚本不能直接在Windows操作系统上执行。尽管Android NDK的最新版本表明它在独立性和自我打包方面有进步，但是它仍然需要在主机上安装Cygwin才能进行完整的操作。Cygwin是一个Windows操作系统上的类UNIX环境和命令行接口，它是基于UNIX应用程序的，包括允许运行Android NDK构建系统的shell。在本书编写时，Android NDK要求安装Cygwin 1.7 才能运行。访问 <http://cygwin.com/install.html> 网站并下载Cygwin安装程序setup.exe(如图 1-18 所示)。

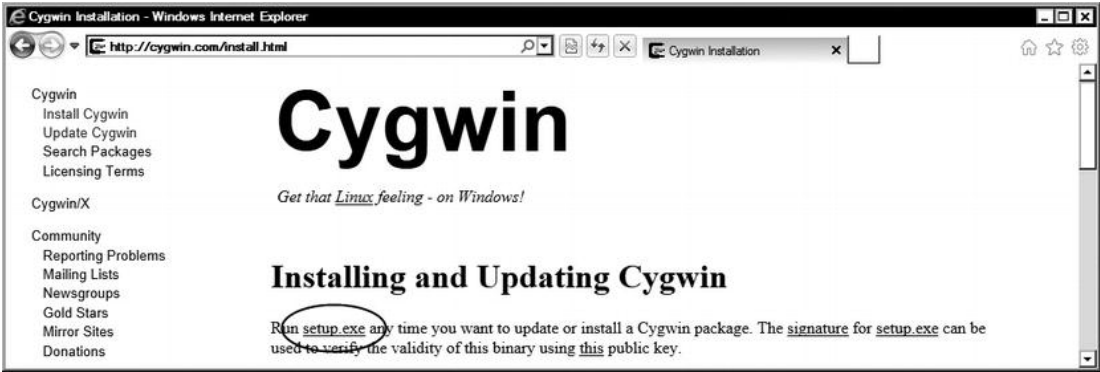


图 1-18 下载 Cygwin 安装程序

启动安装程序之后，可以看见 Cygwin 安装向导欢迎界面，单击 Next 按钮按照以下步骤完成安装操作：

- (1) 安装程序会让用户选择下载源，选择默认选项 Install from Internet，并单击 Next 按钮继续。
- (2) 在下一个对话框中，安装程序会让用户选择 Cygwin 的安装目录，如图 1-19 所示。默认情况下，Cygwin 被安装在 C:\cygwin 目录下。注意要记住目标目录名以备后面使用，然后单击 Next 按钮。

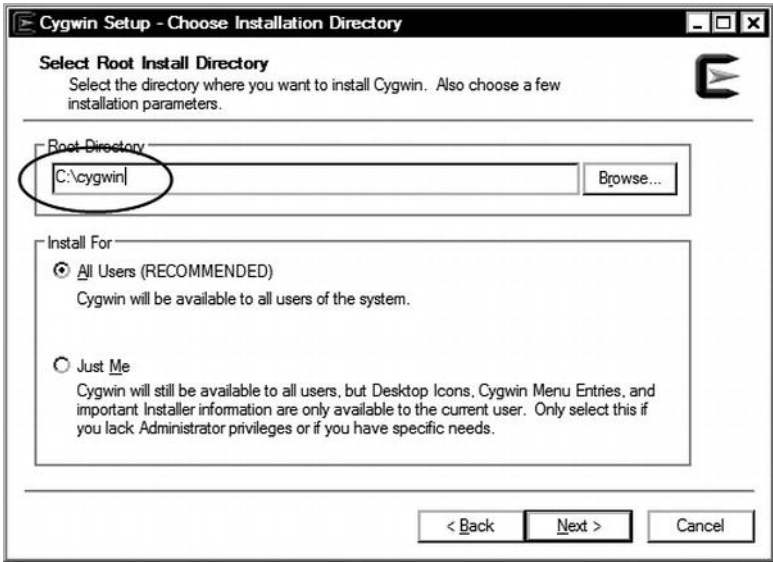


图 1-19 选择 Cygwin 安装目录

(3) 下一个对话框将让用户选择原生包目录，这是一个用于下载文件包的临时目录。使用默认值，然后单击 Next 按钮。

(4) 下一个对话框将让用户选择 Internet 连接类型，除非需要用代理访问 Internet，否则选择默认项 Direct Connection，然后单击 Next 按钮继续。

(5) 安装程序将让用户选择下载站点，从镜像站点列表中随机选择一个站点或者选择一个离安装站点地理位置最近的站点，然后单击 Next 按钮。

(6) Cygwin 不是单个的应用程序，是包含多个应用程序的巨大的软件分布。在下一个对话框中，Cygwin 安装程序会为用户提供一个可用包列表，Android NDK 要求安装 GNU Make 3.8.1 及以后版本以正常使用其功能。在搜索框中输入关键字 make 对包列表进行过滤，展开 Devel 目录，选择 GNU Make 包，如图 1-20 所示。单击 Next 按钮开始安装。

安装完成后，要把 Cygwin 二进制路径添加到系统可执行搜索路径中。

- (1) 从 System Properties 打开 Environment Variables 对话框。
- (2) 在系统变量部分单击 New 按钮定义一个新的环境变量。
- (3) 将变量名设置成 CYGWIN_HOME, 将变量值设置成 Cygwin 安装目录(例如 C:\cygwin), 如图 1-21 所示。
- (4) 在 Environment Variables 对话框中的系统变量列表中双击 PATH 变量，并将 ;%CYGWIN_HOME%\bin 追加到变量值后面，如图 1-22 所示。

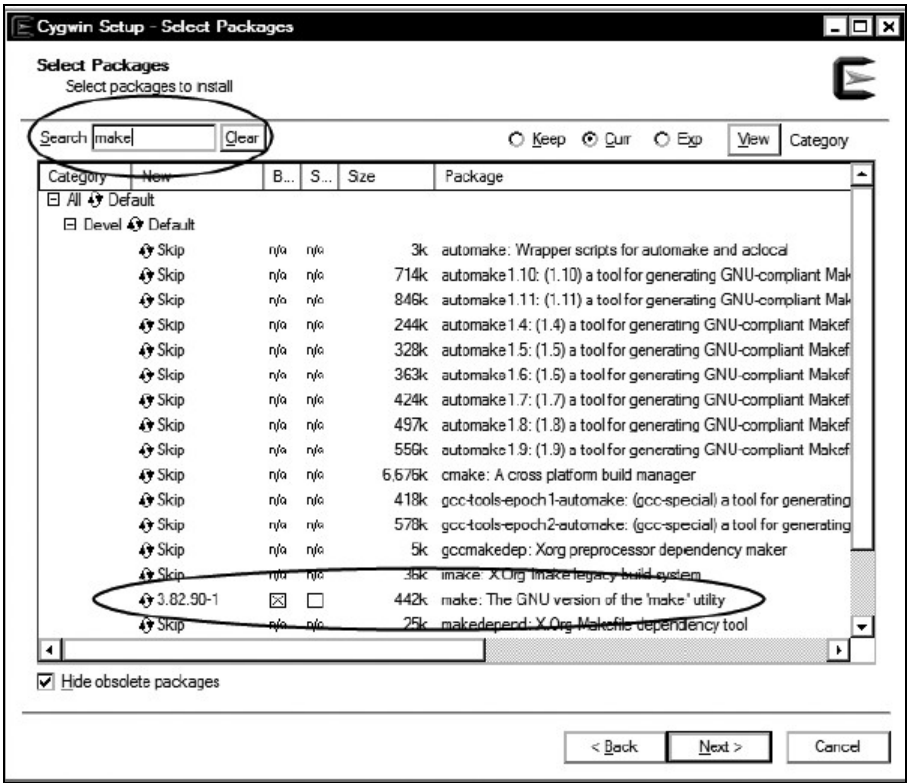


图 1-20 选择 GNU Make 包

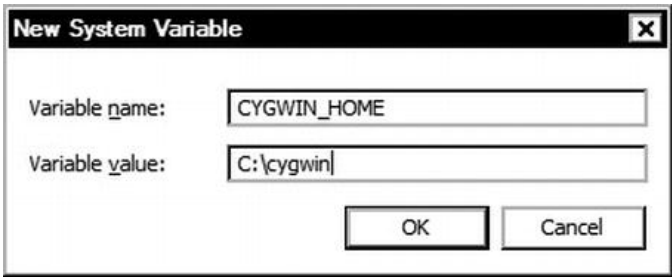


图 1-21 CYGWIN_HOME 环境变量

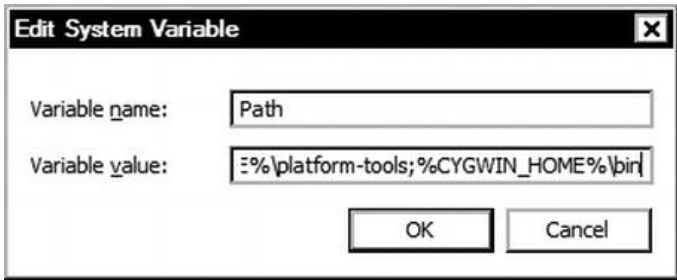


图 1-22 将 Cygwin 二进制路径追加到系统 PATH 变量中

完成了上述安装步骤后，Cygwin 工具成为系统可执行搜索路径的一部分。为了验证安装是否成功，打开一个命令提示窗口，在命令提示符下执行 `make -version`。如果安装成功，则会显示 GNU Make 的版本号，如图 1-23 所示。

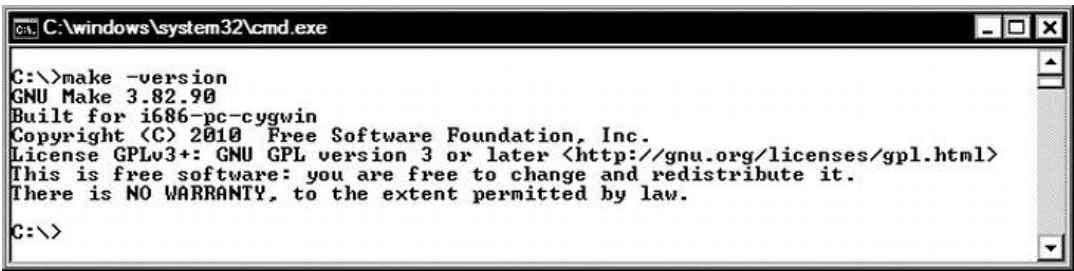


图 1-23 验证 Cygwin 安装结果

1.1.5 在 Windows 平台上下下载并安装 Android NDK

Android 原生开发工具包(Native Development Kit, NDK)是 Android SDK(Software Development Kit)的伴随工具，它可以让用户用诸如 C++的原生编程语言开发 Android 应用程序。Android NDK 提供了头文件、库和交叉编译器工具链。本书编写时，Android NDK 的最新版本是 R8。请定位到 <http://developer.android.com/tools/sdk/ndk/index.html> 网站并找到图 1-24 所示的 Downloads 部分。下载步骤如下：



图 1-24 Android NDK 下载页面

(1) Android NDK 安装包以 ZIP 文档的形式提供。下载完成时，右击 ZIP 文件并在上下文菜单中选择 Extract All 选项，打开 Extract Compressed Folder 向导。

(2) 用 Browse 按钮选择目标目录。因为 ZIP 文件中已经包含一个名为 android-ndk-r8 的子目录，其中包含 Android NDK 文件，因此不需要建立专用的空白目标目录。要记住目标目录名以备后面设置环境变量时使用。

(3) 单击 Extract 按钮安装 Android NDK。

安装完成后，按以下步骤将 Android SDK 的二进制路径追加到系统可执行搜索路径中：

(1) 同样，在 System Properties 界面打开 Environment Variables 对话框。

(2) 在系统变量部分单击 New 按钮定义一个新的环境变量，将变量名设置成 ANDROID_NDK_HOME，将变量值设置成前面记下的 Android NDK 安装目录(例如 C:\android\android-ndk-r8)，如图 1-25 所示。

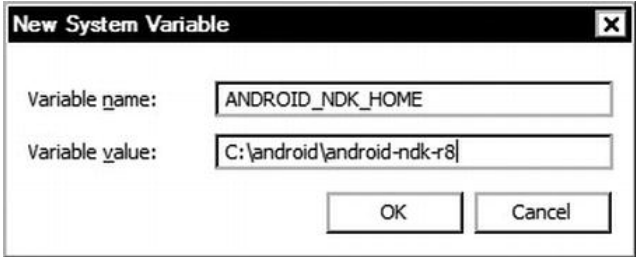


图 1-25 ANDROID_NDK_HOME 环境变量

(3) 单击 OK 按钮保存该新环境变量。

(4) 在环境变量对话框中的系统变量列表中，双击 PATH 变量，并将;%ANDROID_NDK_HOME%追加到变量值后面，如图 1-26 所示。

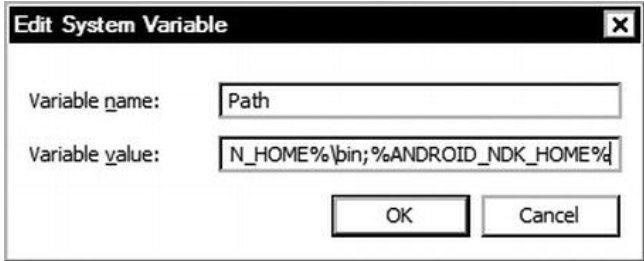


图 1-26 将 Android NDK 二进制路径追加到系统 PATH 变量中

现在可以很容易地访问 Android NDK。为了验证安装是否成功，打开一个命令提示窗口，在命令提示符下执行 `ndk-build`。如果安装成功，就会看到 NDK 给出的关于项目目录的提示，如图 1-27 所示。

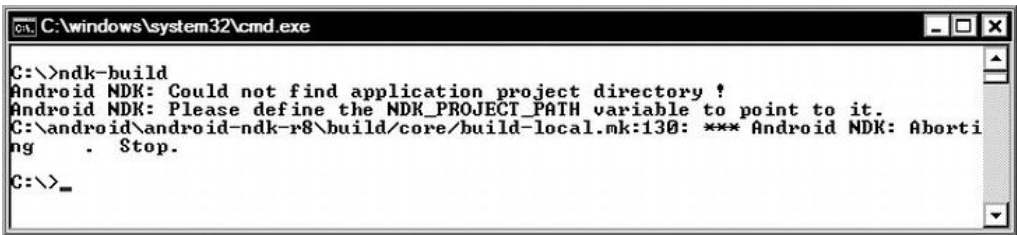


图 1-27 Android NDK 安装结果验证

1.1.6 在 Windows 平台上下下载并安装 Eclipse

Eclipse 是高度可扩展的、多语言集成的开发环境，尽管原生 Android 开发不要求必须安装 Eclipse，但是因为 Eclipse 提供了高度集成的编码环境，将其与 Android 工具结合使用可以简化应用程序的开发过程。本书编写时，Eclipse 的最新版本是 Juno 4.2，请访问 <http://www.eclipse.org/downloads/> 网站下载 Eclipse，如图 1-28 所示，下载步骤如下：

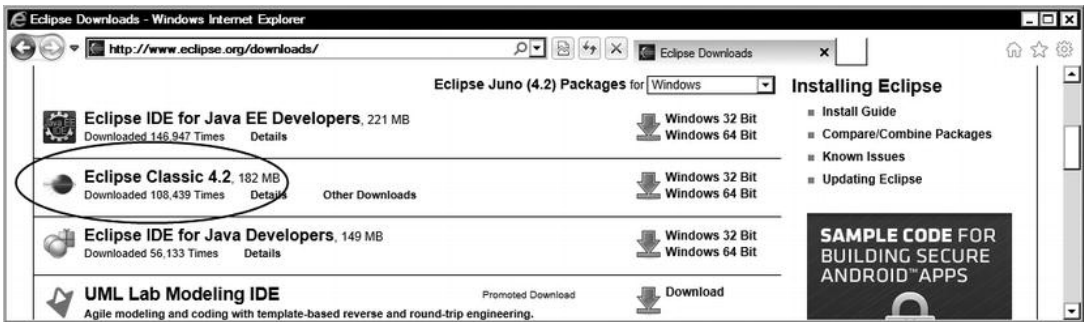


图 1-28 Eclipse 下载页面

- (1) 从列表中下载 Eclipse Classic for Windows 32 Bit，Eclipse 安装包以 ZIP 文档形式提供。
- (2) 下载完成时，右击该 ZIP 文件，并在上下文菜单中选择 Extract All 选项，打开 Extract Compressed Folder 向导。
- (3) 用 Browse 按钮选择目标目录，因为 ZIP 文件中已经包含一个名为 eclipse 的子目录，其中包含 Eclipse 文件，因此不需要建立专用的空白目标目录。要记住目标目录名以备后面设置环境变量时使用。
- (4) 单击 Extract 按钮安装 Eclipse。
- (5) 为了方便访问 Eclipse，进入 Eclipse 安装目录。
- (6) 右击 Eclipse 二进制，并选择 Send | Desktop 在 Windows 桌面上建立 Eclipse 的快捷键。

为了验证 Eclipse 安装结果的有效性，双击 Eclipse 图标。如果安装成功，将看到图 1-29

所示的 Eclipse Workspace Launcher 对话框。

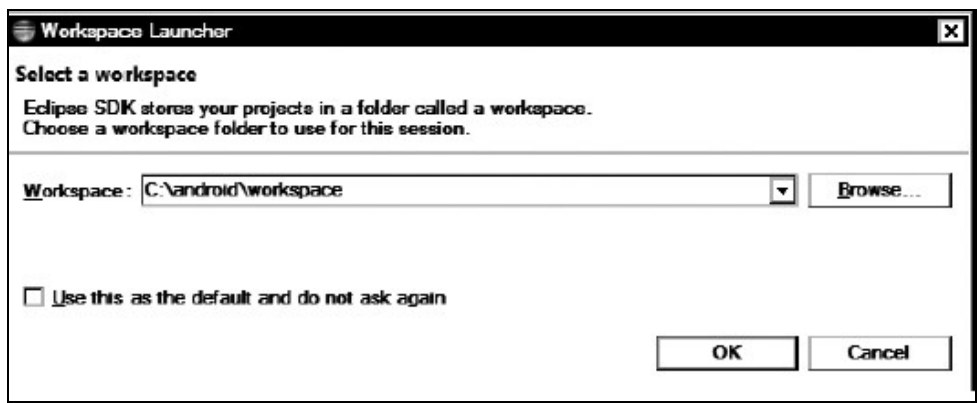


图 1-29 Eclipse 安装结果验证

1.2 Apple Mac OS X

Android 开发工具要求 Mac OS X 10.5.8 及后续版本和 x86 系统。因为 Android 开发工具最初被设计成在类 UNIX 操作系统上工作，可以直接通过 OS X 或者 Xcode 开发工具在平台上使用它的大多数扩展功能。在本节中，用户需要下载并安装以下组件：

- Xcode
- Java JDK 6
- Apache ANT Build System
- GNU Make
- Android SDK
- Android NDK
- Eclipse IDE

1.2.1 在 Mac 平台上安装 Xcode

Xcode 可以为 OS X 平台上的应用程序开发提供开发工具。它可以在 Mac OS X 安装介质中找到，或者通过 Mac App Store 免费获取。访问 <https://developer.apple.com/xcode/> 网站可以获取更多信息。启动 Xcode 安装程序将进入 Xcode 安装向导，向导程序将引导你完成安装过程。

- (1) 同意许可协议。
- (2) 选择目标目录。
- (3) 安装向导会显示可以安装的 Xcode 组件列表，从列表中选择 UNIX 开发工具包，如图 1-30 所示。
- (4) 单击 Continue 按钮开始安装。

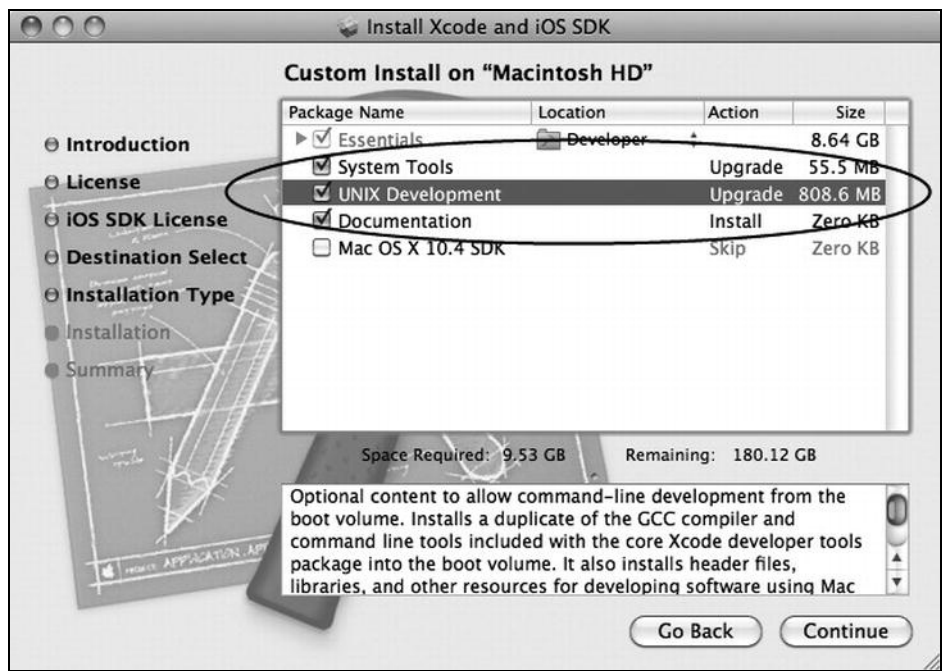


图 1-30 Xcode 定制安装程序对话框

1.2.2 验证 Mac 平台的 Java 开发包

Android 开发工具要求事先安装 JDK(Java Development Kit)6 才能运行,苹果 Mac OS X 操作系统已经配置了 JDK,所配置的 JDK 是基于 Oracle JDK 的,但是由苹果来配置 JDK 可以更好地与 Mac OS X 集成。可以通过 Software Update 获取 JDK 的新版本。必须确保安装的是 JDK 6 及以后版本。为了验证 JDK 的安装效果,打开一个终端窗口,在命令行方式下执行 `javac -version`。如果 JDK 安装正确,会看到 JDK 的版本号,如图 1-31 所示。

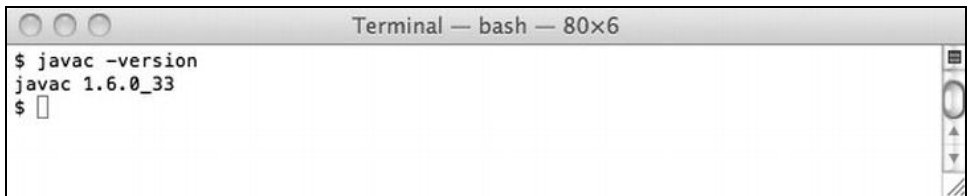


图 1-31 验证 JDK

1.2.3 验证 Mac 平台上的 Apache ANT

Apache ANT 是命令行构建工具,它可以驱动任何根据目标和任务描述的过程,Android 开发工具要求安装 Apache ANT 1.8 及以后版本才能运行构建过程,Apache ANT 作为 Xcode 的 UNIX 开发包的一部分安装到系统中。为了验证 Apache ANT 的安装效果,打开一个终端窗口并在命令行方式下执行 `ant -version`。如果安装成功,会看到 Apache ANT 的版本号,如图 1-32 所示。

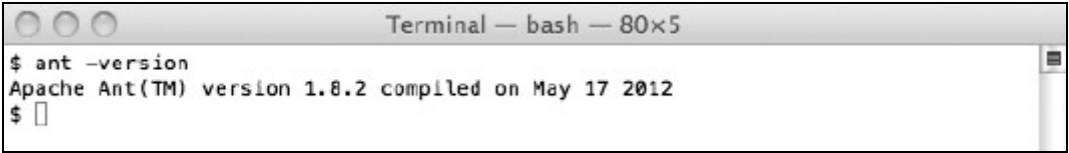


图 1-32 验证 Apache ANT

1.2.4 验证 GNU Make

GNU Make 是在应用程序源代码中控制生成程序中的可执行程序部分及其他部分的构建工具。Android NDK 要求安装 GNU Make 3.8.1 及以后版本。GNU Make 是作为 Xcode 的 UNIX 开发包的一部分安装到系统中。为了验证 GNU Make 的安装效果，打开一个终端窗口，在命令行方式下执行 `make -version`。如果安装成功，会看到 GNU Make 的版本号，如图 1-33 所示。

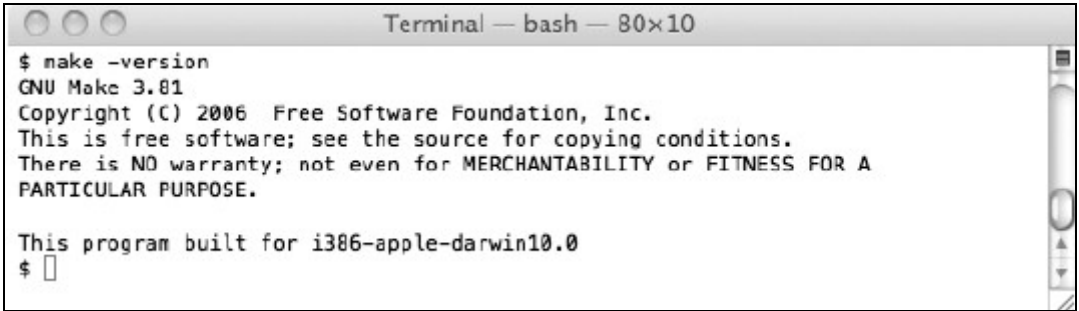


图 1-33 验证 GNU Make

1.2.5 在 Mac 平台上下下载并安装 Android SDK

Android 软件开发包(SDK)是开发工具链的核心组件，它提供了构建、测试和调试 Android 应用程序所需要的框架 API 库和开发工具。本书编写时，Android SDK 的最新版本是 R20，请访问 <http://developer.android.com/sdk/index.html> 网站下载 Android SDK，如图 1-34 所示。安装步骤如下：



图 1-34 Android SDK 下载页面

- (1) 单击 Download the SDK for Mac 按钮开始下载 SDK 安装包。
- (2) Android SDK 安装包以 ZIP 文档方式提供，OS X 提供对 ZIP 文档的原生支持。如果使用 Safari 浏览器，则 ZIP 文件会在下载后自动解压，也可以双击下载的 ZIP 文件以压缩文件夹的形式打开文件。
- (3) 使用 Finder 将 android-sdk-macosx 目录拖放到目标文件夹下，如图 1-35 所示。本书中/android 目录是保存 Android 开发工具及其相关组件的根目录。



图 1-35 将 Android SDK 安装到目标位置

为了便于访问 Android SDK，要把 Android SDK 二进制路径追加到系统可执行搜索路径中。打开一个终端窗口并执行下列命令，如图 1-36 所示：

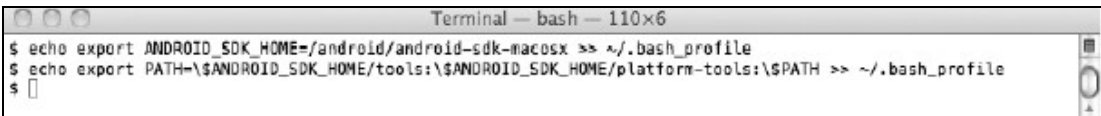


图 1-36 将 SDK 二进制路径追加到系统 PATH 变量中

- `echo export ANDROID_SDK_HOME=/android/android-sdk-macosx > > ~/.bash_profile`
- `echo export PATH = \${ANDROID_SDK_HOME}/tools:\${ANDROID_SDK_HOME}/platformtools:\$PATH > > ~/.bash_profile`

为了验证 Android SDK 的安装效果，打开一个新的终端窗口并在命令行方式下执行 `android -h`。如果安装成功，将会看到图 1-37 显示的帮助信息。

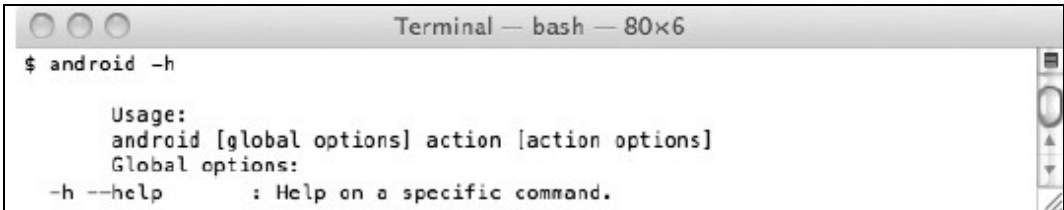


图 1-37 验证 Android SDK 安装效果

1.2.6 在 Mac 平台上下载并安装 Android NDK

Android 原生开发工具包(NDK)是 Android SDK 的伴随工具,它可以让用户用诸如 C++ 之类的原生编程语言开发 Android 应用程序。Android NDK 提供头文件、库和交叉编译器工具链。本书编写时,Android NDK 的最新版本是 R8。请到 <http://developer.android.com/tools/sdk/ndk/index.html> 网站下载 Android NDK, 下载部分如图 1-38 所示。下载步骤如下:



图 1-38 Android NDK 下载页面

- (1) 单击下载安装包, Android NDK 安装包以 BZIP'ed TAR 文档形式提供, OS X 不会自动解压这种类型的文件。
- (2) 为了自动解压文件, 打开一个终端窗口。
- (3) 进入目标目录/android。
- (4) 执行 `tar jxvf ~/Downloads/android-ndk-r8-darwin-x86.tar.bz2`, 如图 1-39 所示。

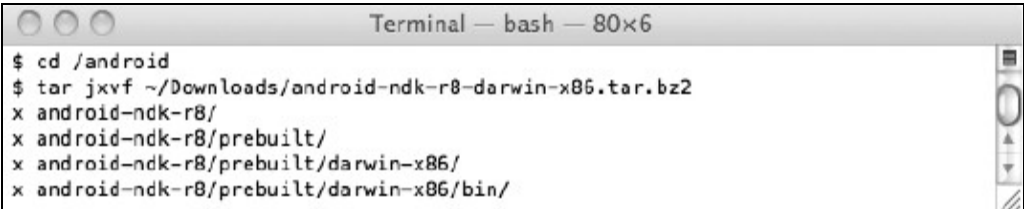


图 1-39 安装 Android NDK

为了便于访问, 要把 Android NDK 二进制路径追加到系统可执行搜索路径中, 打开终端窗口并执行下列命令(如图 1-40 所示)。

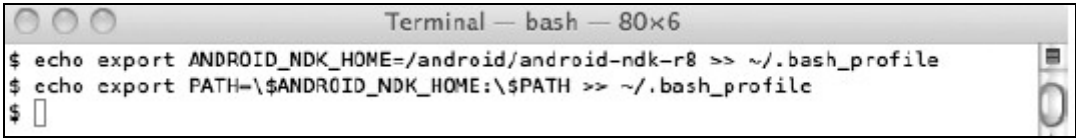


图 1-40 将 Android NDK 二进制路径追加到系统 PATH 变量中

- `echo export ANDROID_NDK_HOME=/android/android-ndk-r8 > > ~/.bash_profile`
- `echo export PATH = \${ANDROID_NDK_HOME}:\$PATH > > ~/.bash_profile`

打开一个新的终端窗口并在命令行方式下执行命令 `ndk-build` 以验证 Android NDK 的安装效果。如果安装成功，就会看到 NDK 给出的关于项目目录的提示，如图 1-41 所示。

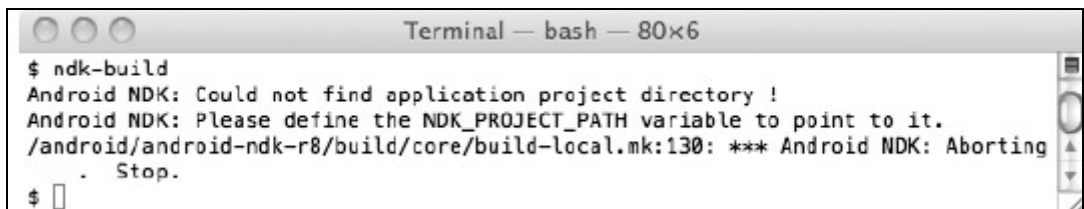


图 1-41 验证 Android NDK

1.2.7 在 Mac 平台上下下载并安装 Eclipse

Eclipse 是高度可扩展的、多语言集成开发环境，尽管原生 Android 开发不要求必须安装 Eclipse，但是 Eclipse 提供了高度集成的编码环境，它与 Android 工具结合使用可以简化应用程序的开发过程。本书编写时，Eclipse 的最新版本是 Juno 4.2。请访问 <http://www.eclipse.org/downloads/> 网站下载 Eclipse，如图 1-42 所示，下载步骤如下：

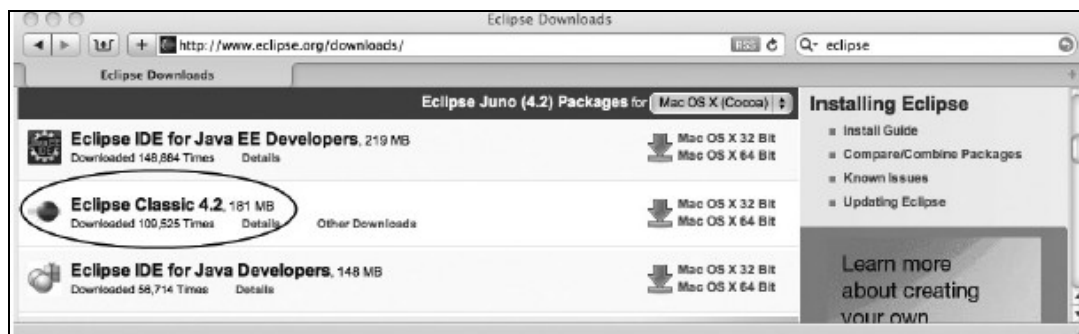


图 1-42 Eclipse 下载页面

- (1) 从列表中下载 Eclipse Classic for Mac OS X 32 Bit。Eclipse 安装包以 GZIP'ed TAR 形式提供；如果使用 Safari 浏览器，文件会自动解压，但是下载后不会自动提取。
- (2) 为了手动释放文件，打开一个终端窗口并进入 `/android` 的目标目录。
- (3) 执行 `tar xvf ~/Downloads/eclipse-SDK-4.2-macosx-cocoa.tar`，如图 1-43 所示。

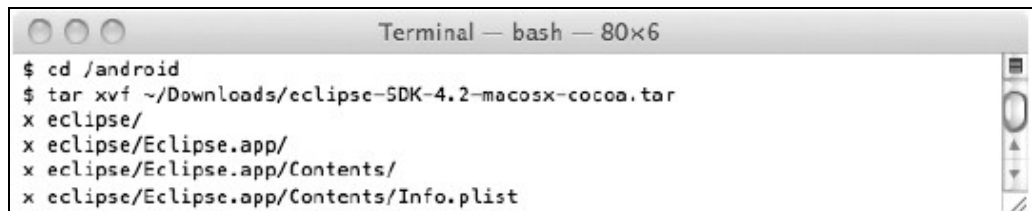


图 1-43 安装 Eclipse

为了便于访问 Eclipse，可以按照如下步骤将 Eclipse 添加到 dock 中：

- (1) 进入 Eclipse 安装目录。
- (2) 将 Eclipse 应用程序拖放到 Dock 中，如图 1-44 所示。



图 1-44 将 Eclipse 加入停靠栏

为了验证 Eclipse 安装结果的有效性，双击 Eclipse 图标。如果安装成功，你就会看到图 1-45 所示的 Eclipse Workspace Launcher 对话框。

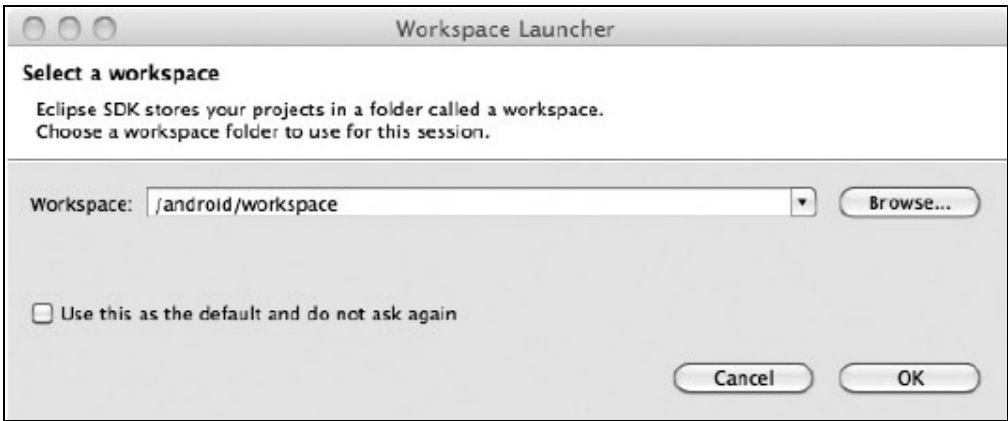


图 1-45 验证 Eclipse

1.3 Ubuntu Linux

Android 开发工具要求安装 Ubuntu Linux 8.04 32-bit 以及后续版本或者安装支持 GNU C Library(glibc)2.7 及以后版本的其他 Linux。在本节中用户需要下载并安装以下组件：

- Java JDK 6
- Apache ANT Build System
- GNU Make
- Android SDK
- Android NDK
- Eclipse IDE

1.3.1 检查 GNU C 库版本

可以通过在终端窗口中运行 `ldd --version` 来检查 GNU C 库版本，如图 1-46 所示。

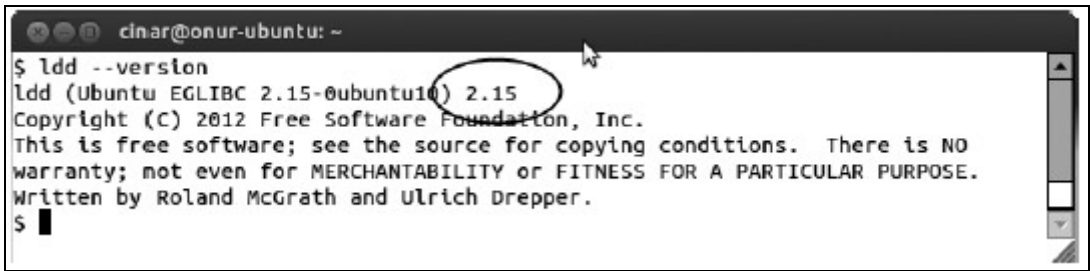


图 1-46 检查 GNU C 库版本

1.3.2 激活在 64 位系统上支持 32 位的功能

在 64 位 Linux 发布之后，Android 开发工具要求安装 32 位支持包。为了安装 32 位支持包，打开一个终端窗口并执行 `sudo apt-get install ia32-libs-multiarch`，如图 1-47 所示。

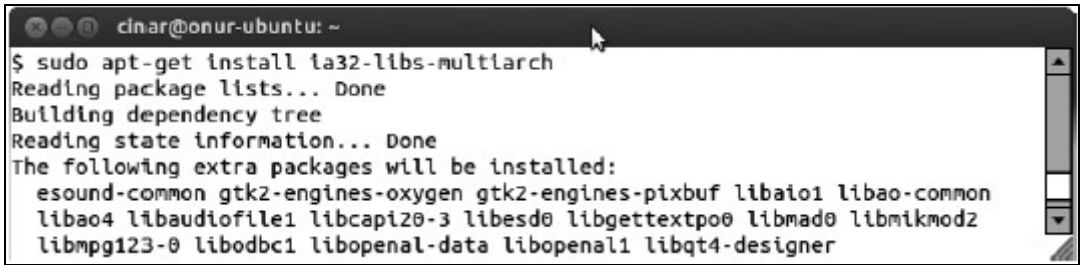


图 1-47 安装 ia32-libs-multiarch

1.3.3 在 Linux 平台上下下载并安装 Java 开发工具包(JDK)

Android 开发工具要求安装 JDK 6 才能运行。不能只安装 JRE(Java Runtime Edition)，在安装 Android 开发工具之前需要先安装 Java JDK 6。除了针对 Java(gcj)GNU Compiler 以外，Android 开发工具支持多种发行版本的 JDK，例如 IBM JDK、Open JDK 以及 Oracle JDK(以前称为 Sun JDK)。由于许可问题，Oracle JDK 不能用于 Ubuntu 软件库。本书以 Open JDK 为例进行讲解。为了安装 Open JDK，打开一个终端窗口并执行 `sudo apt-get install openjdk-6-jdk`，如图 1-48 所示。

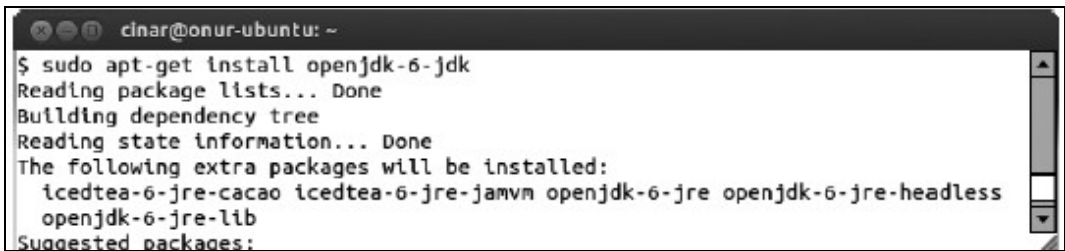


图 1-48 安装 Open JDK 6

为了验证 Open JDK 安装结果的有效性，打开一个终端窗口并在命令行方式下执行

java -version。如果安装成功，就会看到如图 1-49 所示的 Open JDK 版本号。

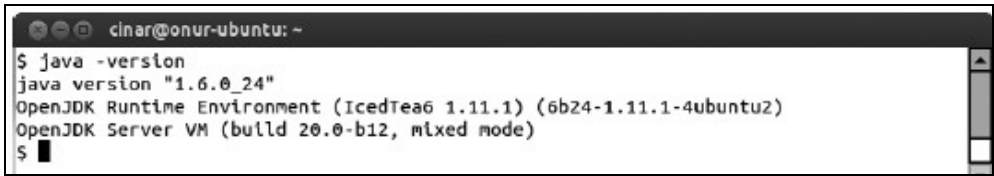


图 1-49 验证 Open JDK 的安装效果

1.3.4 在 Linux 平台上下载并安装 Apache ANT

Apache ANT 是命令行构建工具，可以驱动任何根据目标和任务描述的过程。Android 开发工具要求安装 Apache ANT 1.8 及以后版本，Apache ANT 通过 Ubuntu 软件库提供。为了安装 Apache ANT，打开一个终端窗口并执行 `sudo apt-get install ant`，如图 1-50 所示。

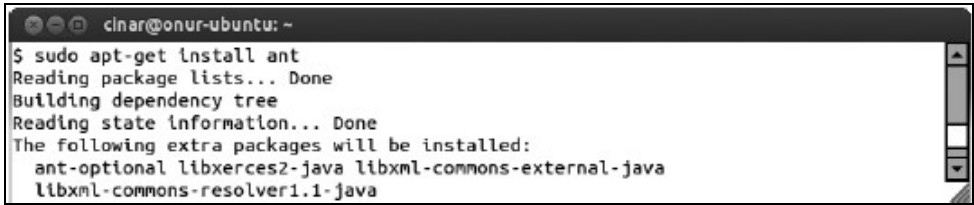


图 1-50 安装 Apache ANT

为了验证 Apache ANT 的安装效果，打开一个终端窗口并在命令行方式下执行 `ant -version`。如果安装成功，将会显示 Apache ANT 的版本号，如图 1-51 所示。

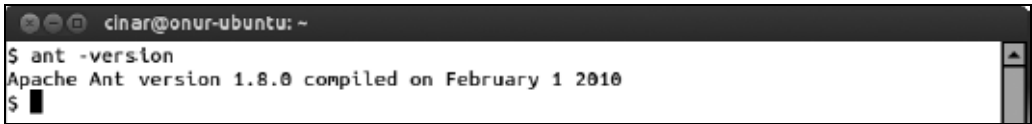


图 1-51 验证 Apache ANT 安装效果

1.3.5 在 Linux 平台上下载并安装 GNU Make

GNU Make 是一种构建工具，用于控制应用程序源代码的可执行代码和其他部分代码的生成。Android NDK 要求安装 GNU Make 3.8.1 及以后版本。GNU Make 是由 Ubuntu 软件库提供的。为了安装 GNU Make，打开一个终端窗口并执行 `sudo apt-get install make`，如图 1-52 所示。

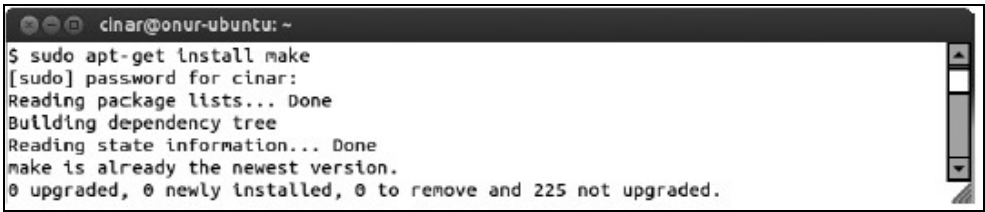


图 1-52 安装 GNU Make

为了验证 GNU Make 的安装效果，打开一个终端窗口并在命令行方式下执行 `make -version`。如果安装成功，将会显示 GNU Make 的版本号，如图 1-53 所示。

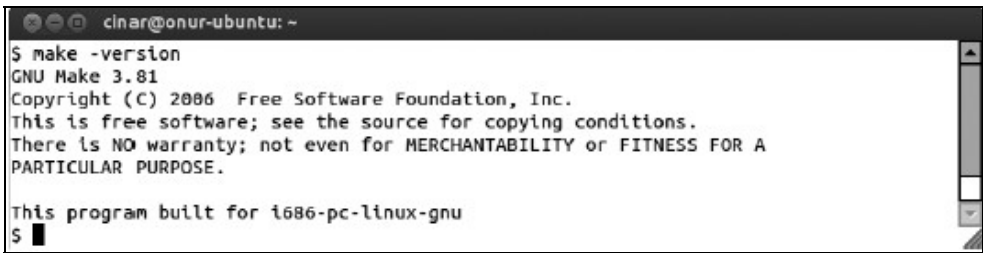


图 1-53 验证 GNU Make 安装效果

1.3.6 在 Linux 平台上下下载并安装 Android SDK

Android 软件开发包 (SDK) 是开发工具链的核心组件，它提供了构建、测试和调试 Android 应用程序所需要的框架 API 库和开发工具。本书编写时，Android SDK 的最新版本是 R20，请访问 <http://developer.android.com/sdk/index.html> 网站下载 Android SDK，如图 1-54 所示。安装步骤如下：



图 1-54 Android SDK 下载页面

- (1) Android SDK 安装包以 GZIP'ed TAR 方式提供， 打开一个终端窗口并进入目标目录。本书中~/android 目录是保存 Android 开发工具及其相关组件的根目录。
- (2) 在命令行方式下执行 `tar zxvf ~/Downloads/android-sdk_r20-linux.tgz` 解压 Android SDK，如图 1-55 所示。



图 1-55 安装 Android SDK

为了便于访问 Android SDK，应该把 Android SDK 二进制路径追加到系统可执行搜索路径中。假设使用 BASH shell， 打开一个终端窗口并执行下列命令(如图 1-56 所示)：

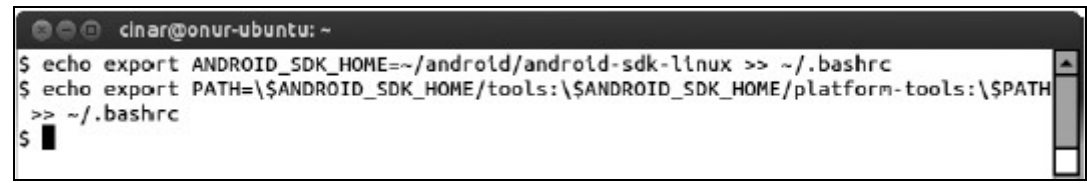


图 1-56 将 SDK 二进制路径追加到系统 PATH 变量中

- `echo export ANDROID_SDK_HOME = ~/.android/android-sdk-linux >> ~/.bashrc`
- `echo export PATH = ~/.android/android-sdk-linux/tools:~/.android/android-sdk-linux/platformtools:~/.android/android-sdk-linux/platform-tools:$PATH >> ~/.bashrc`

为了验证 Android SDK 的安装效果，打开一个新的终端窗口并在命令行方式下执行 `android -h`。如果安装成功，将会看到图 1-57 显示的帮助信息。

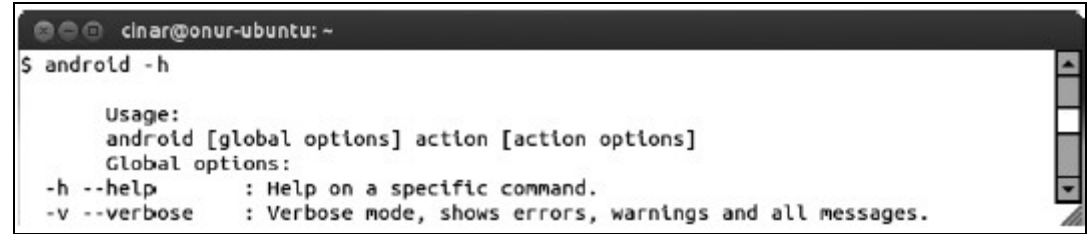


图 1-57 验证 AndroidSDK 安装效果

1.3.7 在 Linux 平台上下下载并安装 Android NDK

Android 原生开发工具包(NDK)是 Android SDK 的伴随工具，可以让用户用诸如 C++ 的原生编程语言开发 Android 应用程序。Android NDK 提供了头文件、库和交叉编译器工具链。本书编写时，Android NDK 的最新版本是 R8。请到 <http://developer.android.com/tools/sdk/ndk/index.html> 网站下载 Android NDK，下载部分如图 1-58 所示。下载步骤如下：



图 1-58 Android NDK 下载页面

- (1) 打开终端窗口进入目标目录~/android。
- (2) Android NDK 安装包以 BZIP’ed TAR 的形式提供，执行 tar jxvf ~/Downloads/android-ndk-r8-linux-x86.tar.bz2 解压文件，如图 1-59 所示。



图 1-59 安装 Android NDK

为了便于访问，要把 Android NDK 二进制路径追加到系统可执行文件搜索路径中。打开一个终端窗口并执行下列命令(如图 1-60 所示)：

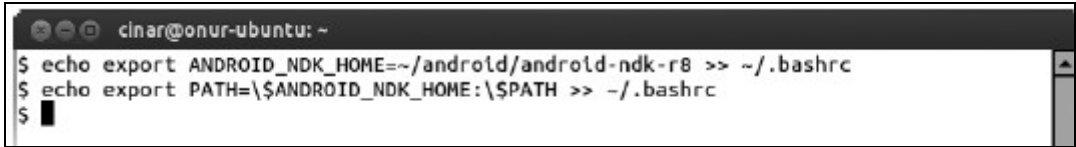


图 1-60 将 Android NDK 二进制路径追加到系统 PATH 变量中

- echo export ANDROID_NDK_HOME = ~/android/android-ndk-r8 > > ~/.bashrc
 - echo export PATH = \\$ANDROID_NDK_HOME:\\$PATH > > ~/.bashrc
- 打开一个终端窗口，在命令行方式下执行命令 ndk-build 以验证 Android NDK 的安装效果。如果安装成功，会看到 NDK 给出的关于项目目录的提示，如图 1-61 所示。

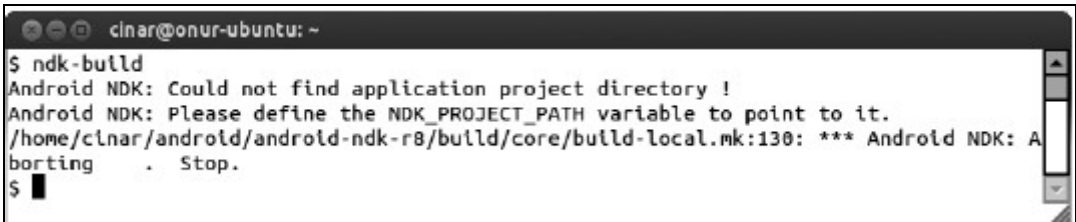


图 1-61 验证 Android NDK 安装情况

1.3.8 在 Linux 平台上下下载并安装 Eclipse

Eclipse 是高度可扩展的、多语言集成开发环境。尽管原生 Android 应用程序开发不要求必须安装 Eclipse，但是 Eclipse 提供了高度集成的编码环境，它与 Android 工具结合使用从而简化应用程序的开发。本书编写时，Eclipse 的最新版本是 Juno 4.2，请访问 <http://www.eclipse.org/downloads/> 网站下载 Eclipse，如图 1-62 所示，下载步骤如下：

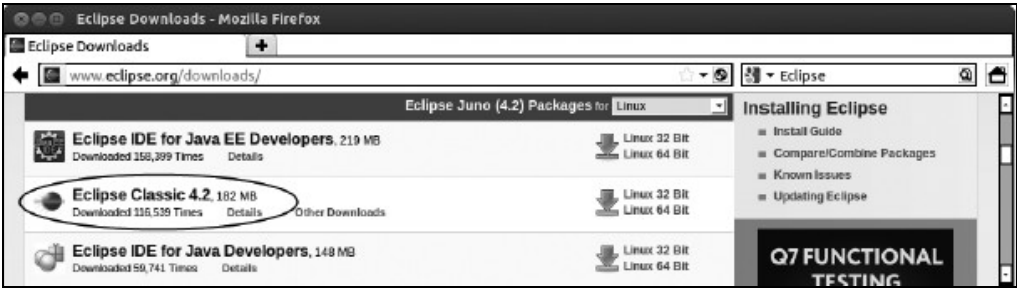


图 1-62 Eclipse 下载页面

- (1) 从列表中下载 Eclipse Classic for Linux 32 Bit。
- (2) 打开一个终端窗口并进入目标目录~/android。
- (3) Eclipse 安装包以 GZIP'ed TAR 形式提供，通过在命令行方式下执行命令 `tar xvf ~/Downloads/eclipse-SDK-4.2- linux-gtk.tar.gz` 进行文件解压缩，如图 1-63 所示。



图 1-63 安装 Eclipse

为了验证 Eclipse 安装结果的有效性，进入 eclipse 目录并在命令行方式下执行命令 `./eclipse`。如果安装成功，就会看到如图 1-64 所示的 Eclipse Workspace Launcher 对话框。

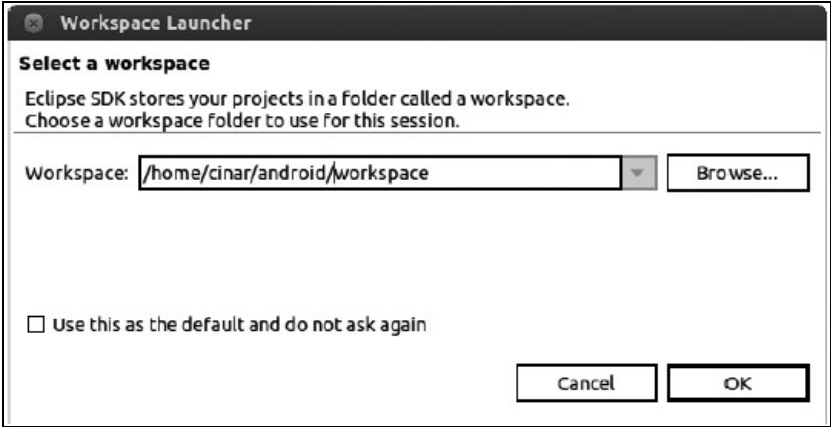


图 1-64 验证 Eclipse 安装结果

1.4 下载并安装 ADT

Android 开发工具(Android Development Tools, ADT)是 Android C++开发环境的平台独立组件，它必须安装在所有三种操作系统上。

Eclipse 平台是以插件的概念为基础构建而成,ADT 是 Eclipse 平台上用于进行 Android 应用程序开发的插件集合,ADT 是遵循开源 Apache 许可的免费软件。关于 ADT 最新版本的更多信息以及最新的安装步骤请参见 <http://developer.android.com/sdk/eclipse-adt.html> 网站上 Eclipse 页面的 ADT 插件部分内容。我们将使用 Eclipse 的 Install New Software 向导来安装 ADT:

- (1) 在顶级菜单栏中选择 Help | Install New Software 启动安装向导,如图 1-65 所示。

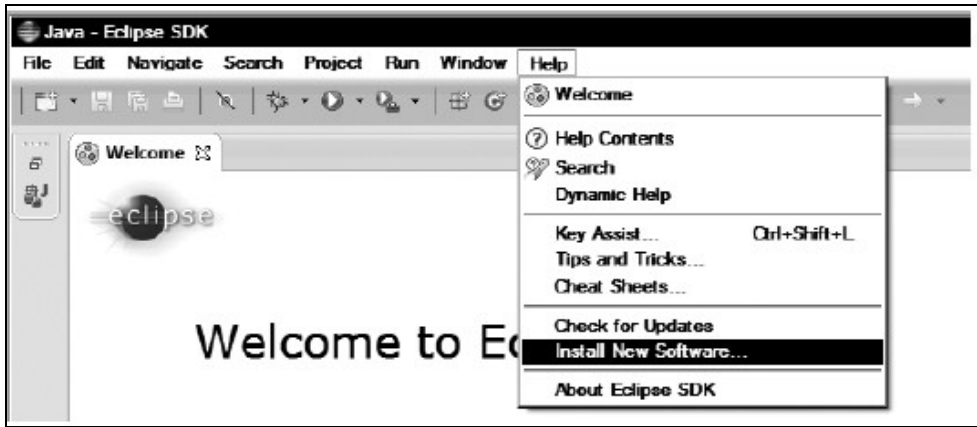


图 1-65 Eclipse 安装新软件

(2) 安装向导将启动并显示可用插件列表。因为 ADT 不是 Eclipse 官方软件库的组成部分,用户需要先添加 Android 的 Eclipse 软件库作为一个新软件站点。为完成此项任务,单击 Add 按钮,如图 1-66 所示。

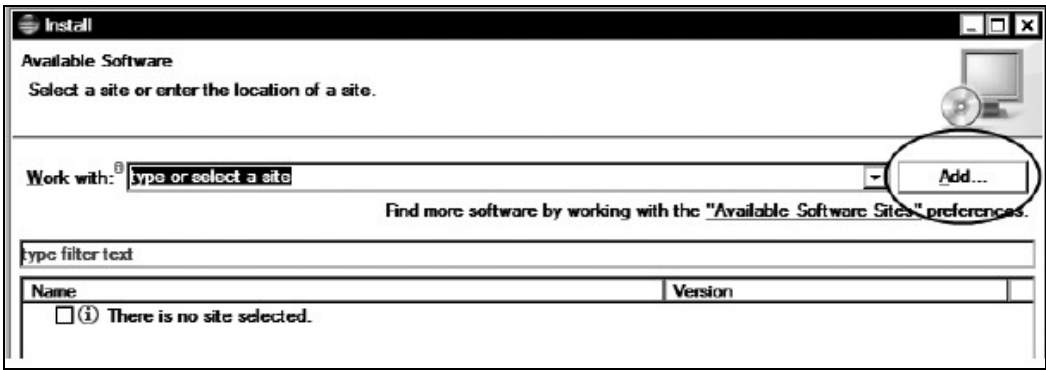


图 1-66 添加新软件库

(3) 出现 Add Repository 对话框。在 Name 字段中输入 Android ADT,并在 Location 字段中输入 Android 的 Eclipse 软件库的 URL:<https://dl-ssl.google.com/android/eclipse/>,如图 1-67 所示。



图 1-67 添加 Android ADT 软件库

- (4) 单击 OK 按钮添加新软件站点。
- (5) Install New Software 向导将显示可用 ADT 插件列表，如图 1-68 所示。其中的每一个插件对 Android 应用程序开发都至关重要，强烈推荐安装所有插件。

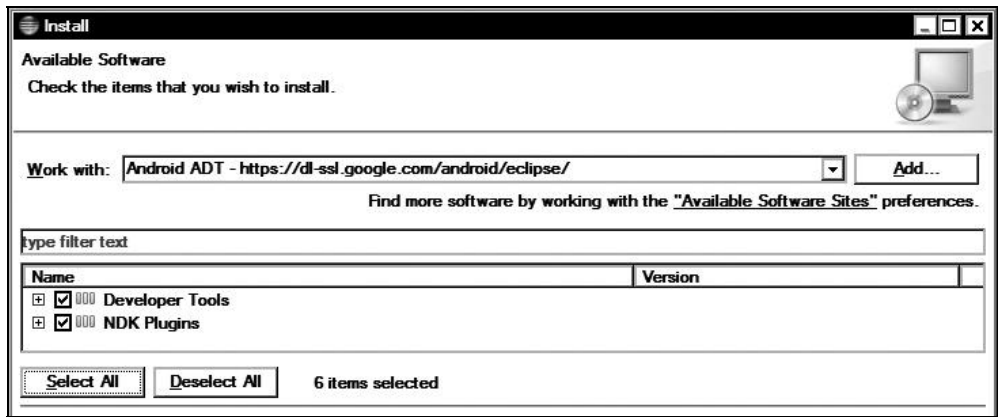


图 1-68 安装 ADT

- (6) 单击 Select All 按钮选择所有的 ADT 插件。
 - (7) 单击 Next 按钮进入下一步操作。
 - (8) Eclipse 将浏览选中的插件列表以将所有的相关组件追加到列表中，然后会显示最后的下载列表让用户核查。单击 Next 按钮进入下一步操作。
 - (9) ADT 含有一系列遵守不同许可协议的第三方组件。在安装过程中，Eclipse 会显示每个软件的许可协议，让用户接收许可协议的内容才可以继续安装过程。选择接收许可协议，单击 Finish 按钮开始安装过程。
- 在未签约的 JAR 文件中的 ADT 插件会触发安全警告，如图 1-69 所示。单击 OK 按钮忽略警告并继续安装。当 ADT 插件安装完成时，Eclipse 需要重启从而使所做的修改生效。



图 1-69 安全警告

重启时，ADT 将询问 Android SDK 的位置。选择 Use existing SDKs，并使用 Browse 按钮选择 Android SDK 的安装目录，如图 1-70 所示。

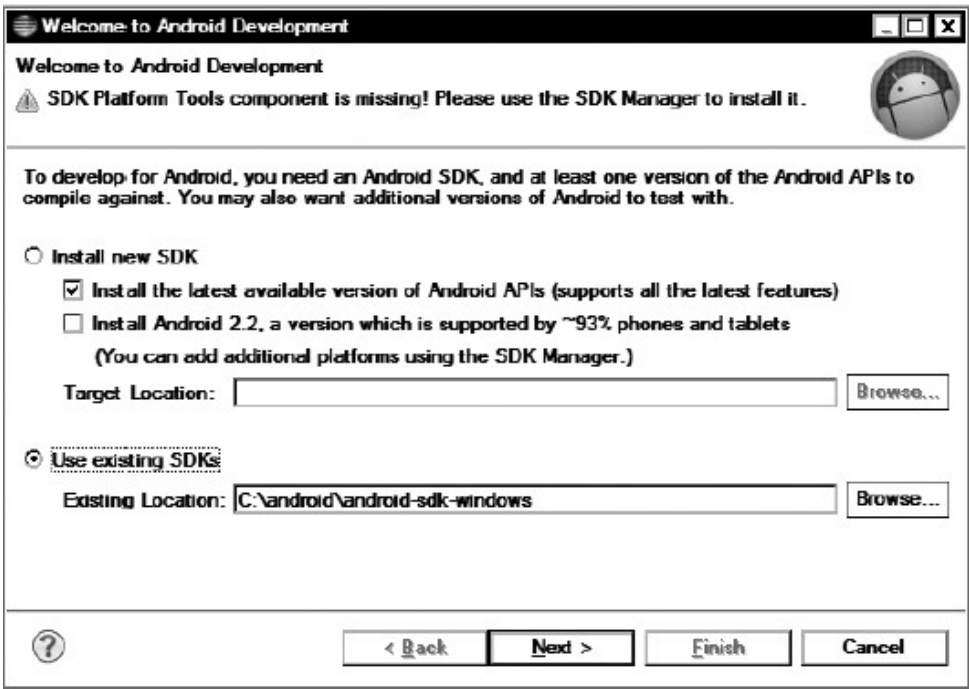


图 1-70 选择 Android SDK 位置

单击 Next 按钮进入下一步操作。

1.4.1 安装 Android 平台包

选择了 Android SDK 位置之后，ADT 验证 Android SDK 和 Android Platform 包。Android SDK 安装程序只包含 Android 开发工具，Android Platform 包需要单独安装，这样才能构建 Android 应用程序。验证完成后，显示 SDK 验证警告对话框，如图 1-71 所示。



图 1-71 ADT Android SDK 验证

单击 Open SDK Manager 按钮启动 Android SDK Manager，按照图 1-72 所示的步骤进行操作。

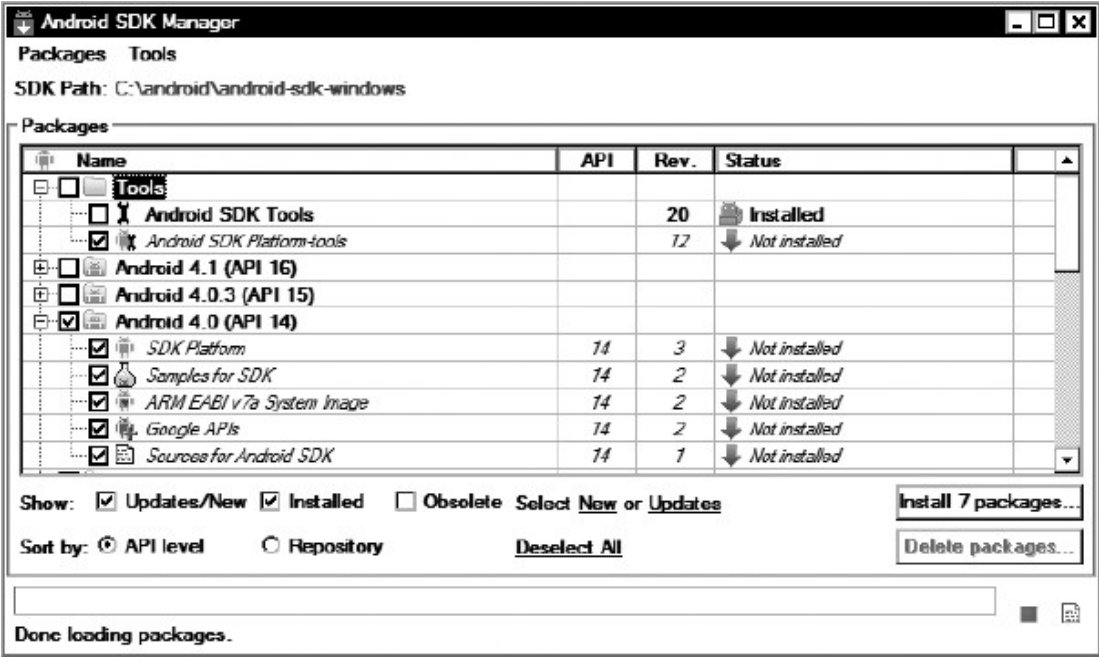


图 1-72 Android SDK 管理器

- (1) 从可用包列表中展开 Tools 目录，并选择 Android SDK Platform-Tools。
 - (2) 选择 Android 4.0 (API 14)类别。
 - (3) 单击 Install N Packages 按钮开始安装。
- Android SDK 管理器将显示选中包的许可协议，接受所有的许可协议继续安装。

1.4.2 配置模拟器

Android SDK 带有一个功能齐全的模拟器，该模拟器是一个在用户的机器上运行的虚拟设备。Android 模拟器可以让你在机器上本地开发和测试 Android 应用程序，而不需要使用物理设备。

Android 模拟器运行一个包括 Linux 内核的完整 Android 系统栈。它是可以模仿真实设备的所有硬件和软件特性的完全虚拟的设备，每一个特征都可以用 Android 虚拟设备管理器(Android Virtual Device, AVD)来定制。如图 1-73 所示，启动 AVD Manager 主菜单中选择 Window | AVD Manager。

单击 AVD Manager 对话框右侧的 New 按钮定义一个新的模拟器配置，如图 1-74 所示。

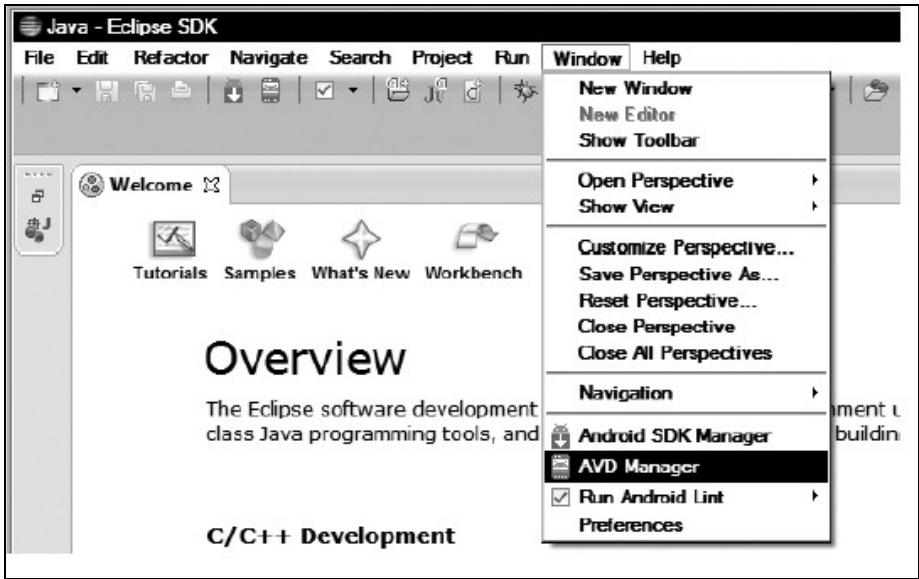


图 1-73 AVD Manager 菜单

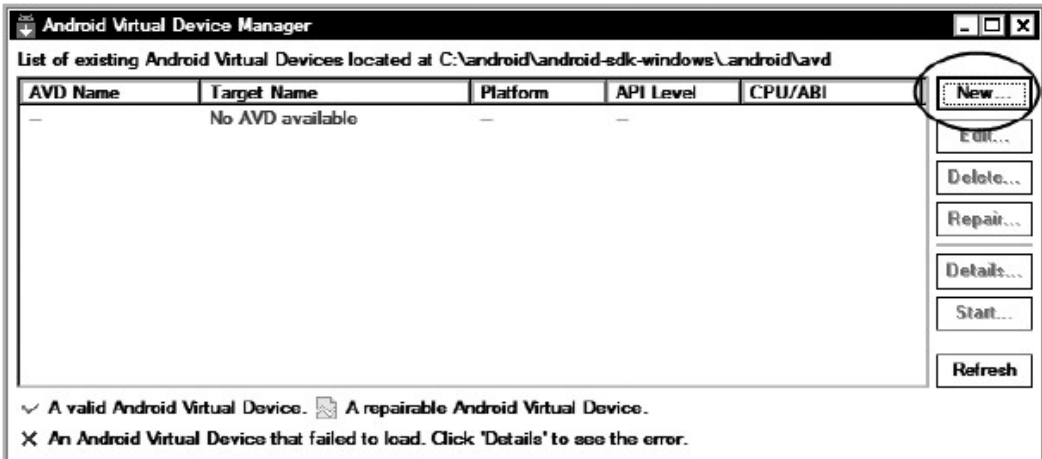


图 1-74 AVD Manager

在本书中，通常在完成了资源配置之后使用 Android 模拟器。推荐使用下面的虚拟机配置执行本书中的示例代码，按照下面的说明设置各字段值，如图 1-75 所示。

- Name 参数应设置为 Android_14。
- Target 参数应设置为 Android 4.0 – API Level 14。
- SD Card 的大小应设置为至少 128 MB。

其他参数可以使用默认值。

为了验证新定义的模拟器配置，打开 AVD Manager，从列表中选择所配置的模拟器名称。单击 Start 按钮启动模拟器实例。如果配置成功，将会出现模拟器(如图 1-76)所示。

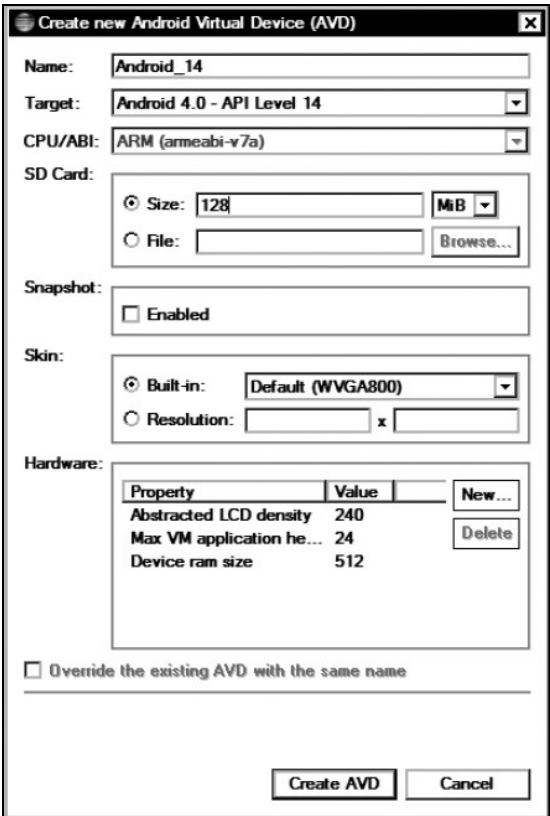


图 1-75 新建模拟器配置



图 1-76 新定义的模拟器配置运行

1.5 小结

本章通过在目标操作系统上安装 Android 开发工具及其相关组件配置你的 Android C++开发环境，定义了 Android 模拟器配置以执行以后章节中将出现的示例代码，第 2 章将详细介绍 Android NDK。

第 2 章

深入了解 Android NDK

在第 1 章中我们通过安装 Android 开发工具及其相关工具配置了开发环境。在这些工具中，Android 原生开发包(NDK)将用于 Android 平台上的 C++开发。Android NDK 是 Android 软件开发包(SDK)的相关工具集，用来扩展 Android SDK 的功能，从而使开发人员能够使用机器代码生成的编程语言(如 C、C++和汇编语言)实现一些对代码性能要求较高的模块并将这些模块嵌入到 Android 应用程序中。

本章开始深入探讨 Android NDK。我们将使用 Android NDK 自带的 hello-jni 示例程序来学习 Android NDK 的构建系统。

2.1 Android NDK 提供的组件

Android NDK 不是一个单独的工具；它是一个包含 API、交叉编译器、链接程序、调试器、构建工具、文档和示例应用程序的综合工具集。以下是 Android NDK 的一些主要组件：

- ARM、x86 和 MIPS 交叉编译器
- 构建系统
- Java 原生接口头文件
- C 库
- Math 库
- POSIX 线程
- 最小的 C++库
- ZLib 压缩库
- 动态链接库
- Android 日志库
- Android 像素缓冲区库

- Android 原生应用 APIs
- OpenGL ES 3D 图形库
- OpenSL ES 原生音频库
- OpenMAX AL 最小支持

2.2 Android NDK 的结构

在安装过程中，所有的 Android NDK 组件都被安装在目标目录下。下面介绍一些重要文件和子目录。

- **ndk-build**: 该 shell 脚本是 Android NDK 构建系统的起始点。本章将在深入学习 Android NDK 构建系统的同时详细阐述 **ndk-build**。
- **ndk-gdb**: 该 shell 脚本允许用 GUN 调试器调试原生组件。第 5 章讨论原生组件调试时将详细阐述 **ndk-gdb**。
- **ndk-stack**: 该 shell 脚本可以帮助分析原生组件崩溃时的堆栈追踪。第 5 章讨论原生组件的故障排除和故障分析时将详细阐述 **ndk-stack**。
- **build**: 该目录包含了 Android NDK 构建系统的所有模块。本章将详细介绍 Android NDK 构建系统。
- **platforms**: 该目录包含了支持不同 Android 目标版本的头文件和库文件。Android NDK 构建系统会根据具体的 Android 版本自动引用这些文档。
- **samples**: 该目录包含了一些示例应用程序，这些程序可以体现 Android NDK 的性能。示例项目对于学习如何使用 Android NDK 的特性很有帮助。
- **sources**: 该目录包含了可供开发人员导入到现有的 Android NDK 项目的一些共享模块。
- **toolchains**: 该目录包含目前 Android NDK 支持的不同目标机体系结构的交叉编译器。Android NDK 目前支持 ARM、X86 和 MIPS 机体系结构。Android NDK 构建系统根据选定的体系结构使用不同的交叉编译器。

Android NDK 最重要的组件是它的构建系统，它包含了所有的其他组件。想要更好地了解构建系统的工作原理，先看一个示例。

2.3 以一个示例开始

下面以 Android NDK 自带的 **hello-jni** 示例应用程序开始讲解。之后可以通过修改它来展示 Android NDK 构建系统所提供的不同功能，例如：

- 建立一个共享库
- 建立多种共享库
- 建立静态库
- 利用共享库共享通用模块
- 在多种 NDK 项目间共享模块

- 使用预建库
- 建立独立的可执行文件
- 其他构建系统变量和宏
- 定义新变量和条件操作

打开第 1 章安装好的 Eclipse IDE。虽然 Android NDK 不要求必须使用 IDE，但使用 IDE 可以帮助用户直观地检查项目结构和构建流。在启动阶段，Eclipse 会让你选择工作区；可以选择默认值并继续。

2.3.1 指定 Android NDK 的位置

因为是首次用工作区进行 Android NDK 开发，所以需要指定 Android NDK 的位置。

(1) 在 Windows 和 Linux 平台上，在主菜单栏中选择 Preference 菜单项。在 Mac OS X 平台上，使用 Eclipse 中的应用程序菜单并且选择 Preferences 菜单项。

(2) 如图 2-1 所示，Preferences 对话框左边的窗格包含了一个树状的 preference 分类列表。展开 Android 然后在树状列表中选择 NDK。

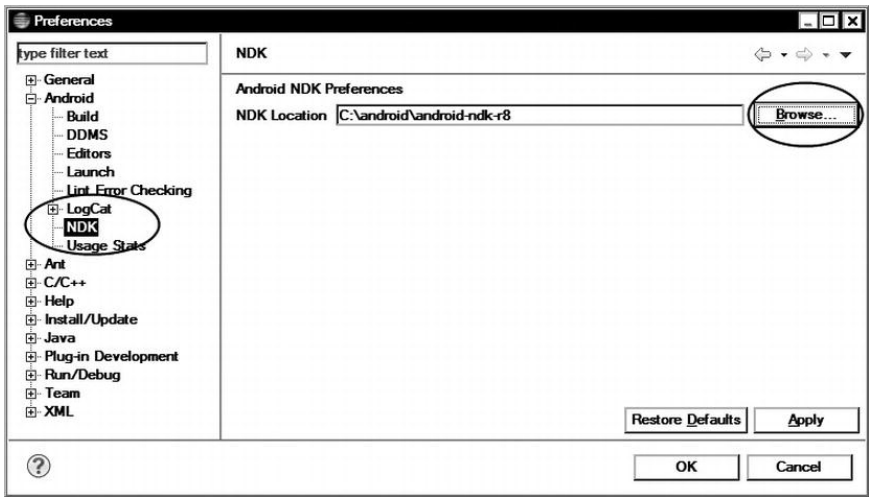


图 2-1 选择的 Android NDK 位置

(3) 在右边的窗格中，单击 Browse 按钮，并利用文件浏览器选择 Android NDK 安装位置。

选择的 NDK 位置仅对当前 Eclipse 工作区有效。如果以后要用其他工作区，需要重复以上过程。

2.3.2 导入示例项目

上节已经讲过，在 samples 目录下包含了 Android NDK 安装程序自带的示例应用程序。现在可以使用其中的示例应用程序。

在主菜单栏中选择 File，然后选择 Import 菜单项打开 Import 向导。在导入资源列表中，展开 Android 并选择 Existing Android Code into Workspace，如图 2-2 所示。单击 Next 按钮

进行下一步。

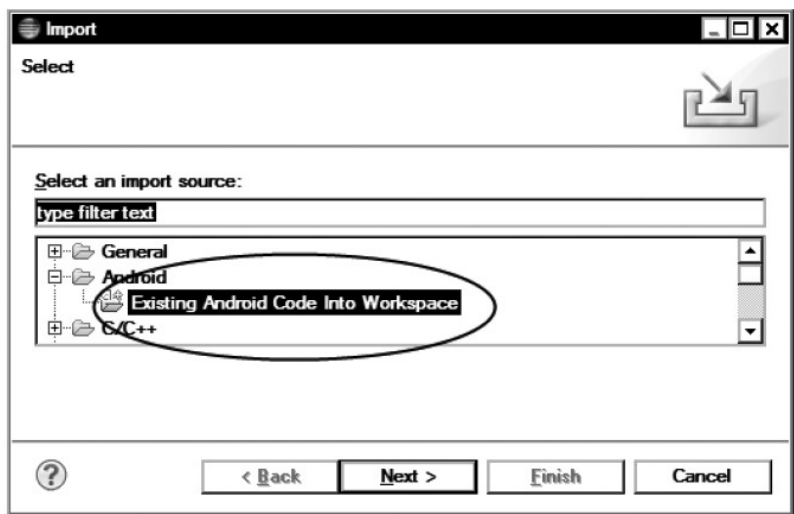


图 2-2 将现有的 Android 代码导入到工作区中

如图 2-3 所示，使用 Browse 按钮打开文件资源管理器并进入<Android NDK>/samples/hello-jni 目录。hello-jni 项目是一个简单的“Hello World”的 Android NDK 项目。项目目录包含实际项目和测试项目。为了简化问题，先不选择测试项目，只选主项目。不对 Android NDK 安装目录做任何修改是保证安全的正确做法。选中 Copy projects into workspace 选项让 Eclipse 将项目代码复制到工作区，这样就可以对副本而不是原始项目进行操作。单击 Next 按钮开始将项目导入工作区。

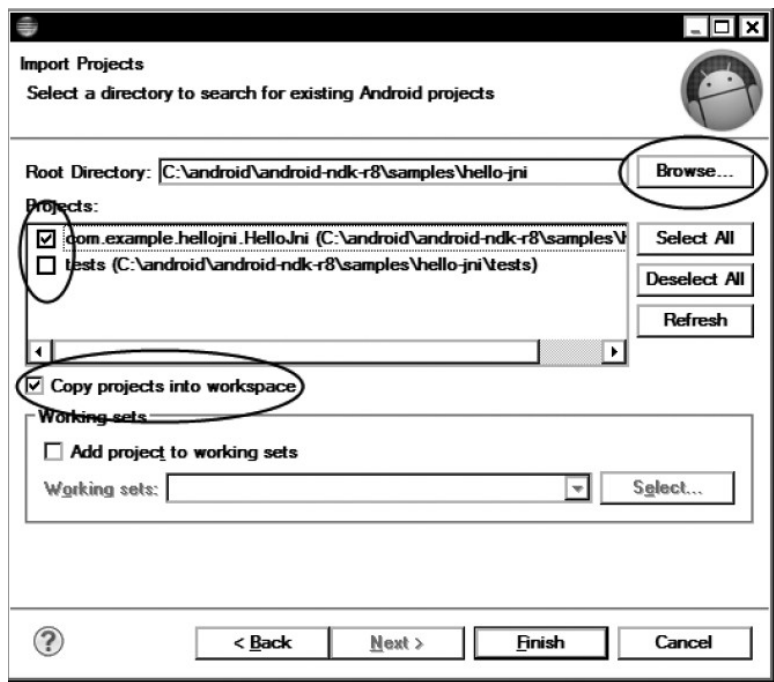


图 2-3 导入 hello-jni Android NDK 项目

如图 2-4 所示，在导入过程的最后一步，控制台会出现一个错误信息。回顾第 1 章的内容，那里只用 SDK Manager 下载了适用于 Android 4.0(API Level 14)平台的 APIs。而 hello-jni 项目是基于 Android 1.5(API Level 3)开发的。



图 2-4 不能解决目标 API Level 3

API Levels 是向后兼容的，所以不用下载 API Level 3，而是在 Eclipse 的 Project Explorer 视图中右击 com.example. hellojni. HelloJni 项目，在上下文菜单中选择 Properties 打开项目属性对话框。项目属性对话框左边的窗格包含了一个树状的项目属性分类列表。在树状列表中选择 Android，并在右边窗格中选择 Android 4.0 作为项目构建目标(如图 2-5 所示)。

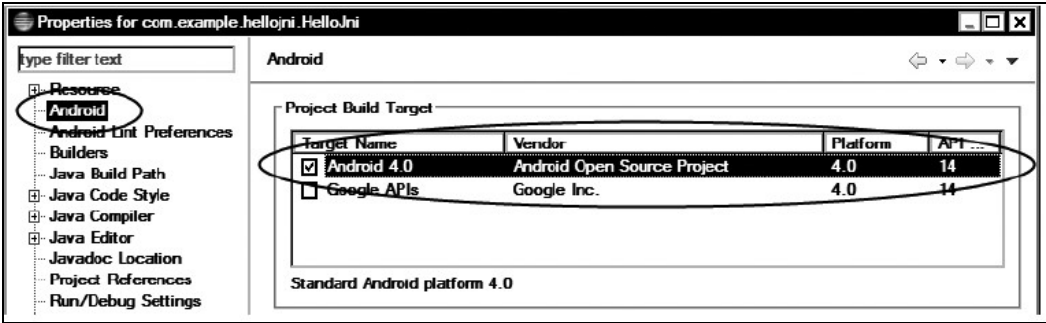


图 2-5 将 Android 4.0 选为项目构建目标

单击 OK 按钮确认上述更改。Eclipse 会用已选的项目构建目标来重建项目。

2.3.3 向项目中添加原生支持

Import Android Project 向导只将项目作为 Android Java 项目导入。为了让构建流包含原生组件，需要手动添加原生支持。在 Eclipse 的 Project Explorer 视图中右击 com.example. hellojni.HelloJni 项目，鼠标停在 Android Tools 菜单项上并在上下文菜单中选择 Add Native Support。打开 Add Android Native Support 对话框，如图 2-6 所示。由于该项目已经包含了一个原生项目，所以库名可以保持不变，单击 Finish 按钮继续。

如果是第一次向 Java-only 项目中添加原生支持，可以在该对话框中指定首选的共享库名，在将构建文件自动生成为进程的一部分时会使用该首选共享库名称。

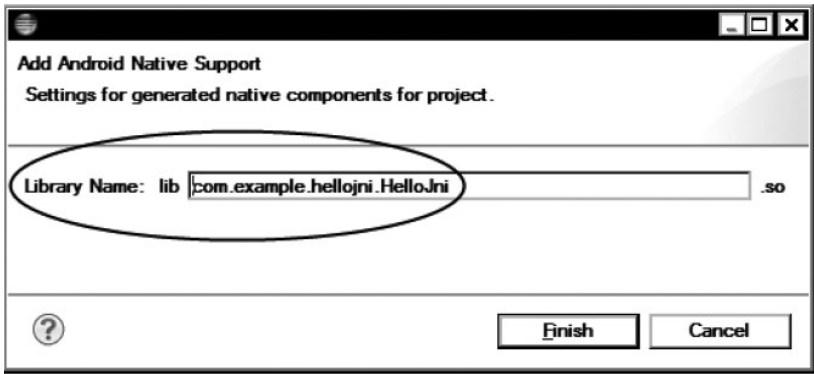


图 2-6 添加 Android 原生支持

2.3.4 运行项目

现在已经完成了项目的编写，可以在 Android 模拟器上运行该项目。在主菜单上选择 Run，然后在子菜单中选择 Run。因为是第一次运行该项目，Eclipse 会通过 Run As 对话框让用户选择该项目的运行方式。在列表中选择 Android Application 并单击 OK 按钮继续。将会启动 Android 模拟器；Eclipse 会自动部署并执行该项目，如图 2-7 所示。Android 模拟器是一个虚拟机，它完全启动 Android 操作系统可能需要花几分钟的时间。



图 2-7 运行原生项目的 Android 模拟器

或许你已经注意到：运行该项目的过程和运行 Java-only 项目的过程一模一样。因为向项目自动添加原生支持的同时已经在用户不知道的情况下将必要步骤包含在构建过程中了。仍然可以在 Console 视图中查看 Android NDK 构建系统发来的消息，如图 2-8 所示。

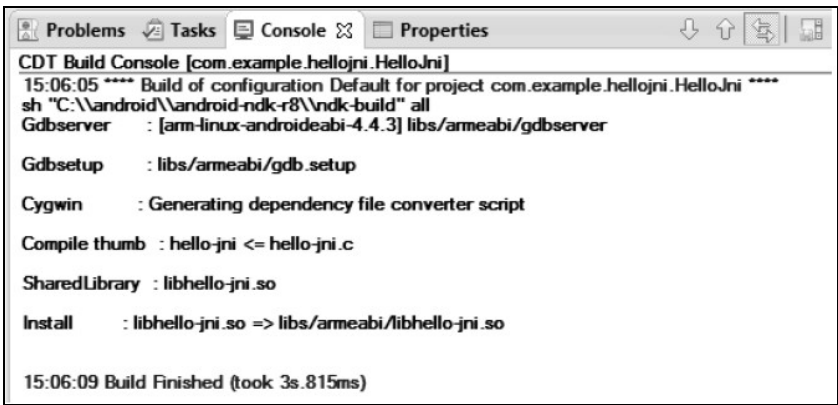


图 2-8 显示 Android NDK 构建信息的 Console 视图

尽管 Eclipse 可以很好地简化整个构建和部署过程，但就像本章之前所提到过的，Eclipse 不是构建 Android NDK 项目的必要条件，整个构建过程也可以用命令行方式执行。

2.3.5 用命令行对项目进行构建

为了在命令行方式下构建 hello-jni 项目，首先在 Windows 中打开命令提示符或在 Mac OS X 或 Linux 中打开终端窗口，并将 hello-jni project 所在目录更改为当前工作目录。用原生组件构建 Android 项目需要两步：第一步构建原生组件，第二步构建 Java 应用程序并将 Java 应用程序与其原生组件打包。为构建原生组件，在命令行方式下执行 ndk-build。ndk-build 是一个调用 Android 构建系统的辅助脚本。如图 2-9 所示，Android NDK 构建脚本会在构建过程中输出进度消息。



图 2-9 用 ndk-build 对原生组件进行构建

现在完成了原生组件的构建，可以继续第二步。Android SDK 构建系统是基于 Apache ANT 的。因为这是第一次用命令行构建项目，所以首先应该生成 Apache ANT 构建文件。在命令行中执行 android update project -p . -n hello-jni -t android-14 --subprojects 命令来生成 Apache ANT 构建文件，如图 2-10 所示。

现在 Apache ANT 构建文档的编写已经完成，可以通过在命令行方式下执行“ant debug”命令构建项目，Apache ANT 将构建 Java 文件并将该 Java 文件与原生组件打成一个可安装 Android 包，即 APK 文件。通过上述操作可以看出，构建带有原生构件的 Android 应用最简单的方式是使用 Eclipse，因为不需要记住每一个构建操作步，所以不易出错。

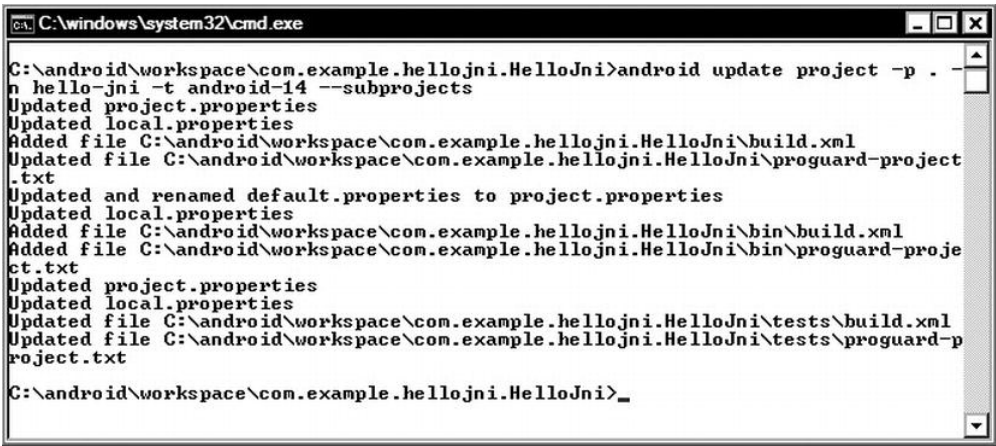


图 2-10 生成 Apache ANT 构建文件

2.3.6 检测 Android NDK 项目的结构

现在重新回到 Eclipse 环境下学习带有原生组件的 Android 应用程序的结构。如图 2-11 所示，带有原生组件的 Android 项目包含一组附加的目录和文件。

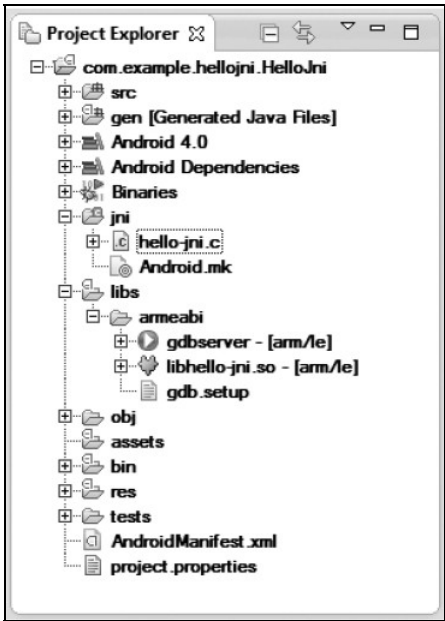


图 2-11 Hello-jni Android NDK 项目的结构

- **jni:** 该目录包含原生组件的源代码以及描述原生组件构建方法的 `Android.mk` 构建文件。Android NDK 构建系统将该目录作为 NDK 项目目录并希望在项目根目录中找到它。
- **Libs:** 在 Android NDK 构建系统的构建过程中创建该目录。它包含指定的目标机体系结构的独立子目录,例如 ARM 的 `armeabi`。在打包过程中该目录被包含在 APK 文件中。
- **Obj:** 这是一个中间目录,编译源代码后所产生的目标文件都保存在该目录下。开发人员最好不要访问该目录。

Android NDK 项目最重要的组件是 `Android.mk` 构建文件,该文档描述了原生组件。理解构建系统是熟练运用 Android NDK 及其所有组件的关键。

2.4 构建系统

Android NDK 的构建系统是基于 GUN Make 的。该构建系统的主要目的是使开发人员能够用很短的构建文档来描述原生的 Android 应用程序;该构建系统还处理了包括替开发人员指定工具链、平台、CPU 和 ABI 等很多细节。封装该构建过程可以在不改变构建文件的情况下,使 Android NDK 的后续更新添加更多对工具链、平台以及系统接口的支持。

Android NDK 构建系统是由多种 GUN Makefile 片段构成的。该构建系统包括基于渲

染构建过程的不同类型 NDK 项目所需要的必要片段。如图 2-12 所示，这些构建系统片段可以在 Android NDK 安装程序的 build/core 子目录中找到。虽然开发人员并不会直接接触到这些文件，但知道它们的位置对与构建系统相关的故障很有帮助。

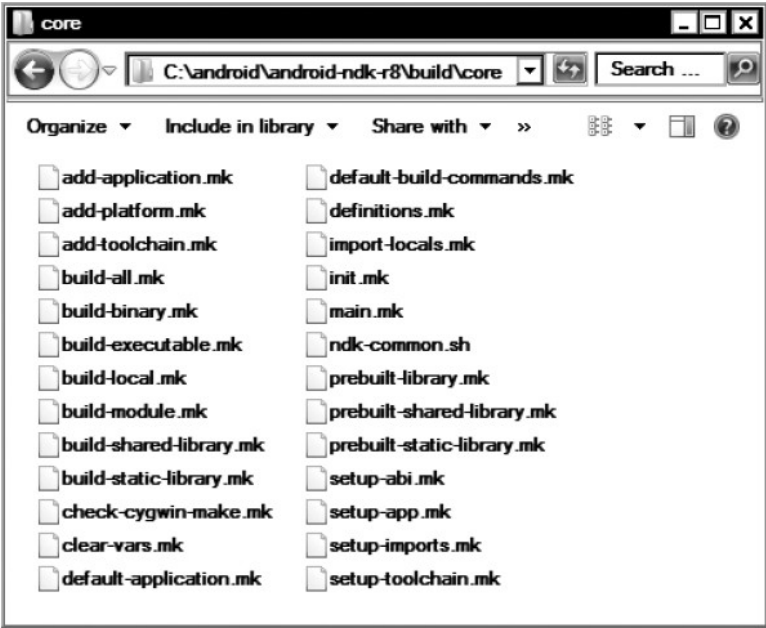


图 2-12 Android NDK 构建系统片段

除了这些片段，Android NDK 构建系统还要依赖另外两个文件：Android.mk 和 Application.mk，这两个文件应该作为 NDK 项目的一部分由开发人员提供，让我们来回顾一下。

2.4.1 Android.mk

Android.mk 是一个向 Android NDK 构建系统描述 NDK 项目的 GUN Makefile 片段。它是每一个 NDK 项目的必备组件。构建系统希望它出现在 jni 子目录中。在 Eclipse 的 Project Explorer 中，双击 Android.mk 文件在编辑视图中打开它。程序清单 2-1 显示了 hello-jni 项目中 Android.mk 文件的内容。

程序清单 2-1 hello-jni 项目下 Android.mk 文件的内容

```
# Copyright (C) 2009 The Android Open Source Project
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)
```

为了更好地理解它的句法，我们逐行分析。因为这个是一个 GUN Makefile 片段，所以它的句法和其他 Makefile 是一样的。每行都包含一个单独的指令，以“#”开头的是注释行，GUN Make 工具不处理它们。根据命名规范，变量名要大写。

注释块后的第一条指令是用来定义 LOCAL_PATH 变量的。根据 Android 构建系统的要求，Android.mk 文档必须以 LOCAL_PATH 变量的定义开头。

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

Android 构建系统利用 LOCAL_PATH 来定位源文件。因为将该变量设置为硬编码值并不合适，所以 Android 构建系统提供了一个名为 my-dir 的宏功能。通过将该变量设置为 my-dir 宏功能的返回值，可以将其放在当前目录下。

Android 构建系统将 CLEAR_VARS 变量设置为 clear-vars.mk 片段的位置。包含 Makefile 片段可以清除除了 LOCAL_PATH 以外的 LOCAL_<name> 变量，例如 LOCAL_MODULE 与 LOCAL_SRC_FILES 等。

```
Include $(CLEAR_VARS)
```

这样做是因为 Android 构建系统在单次执行中解析多个构建文件和模块定义，而 LOCAL_<name> 是全局变量。清除它们可以避免冲突，每一个原生组件被称为一个模块。

LOCAL_MODULE 变量用来给这些模块设定一个唯一的名称。下面的代码将该模块的名称设为 hello-jni：

```
LOCAL_MODULE := hello-jni
```

因为模块名称也被用于给构建过程所生成的文件命名，所以构建系统给该文件添加了适当的前缀和后缀。本例中，hello-jni 模块会生成一个共享库文件且构建系统会将它命名为 libhello-jni.so。

用 LOCAL_SRC_FILES 变量定义用来建立和组装这个模块的源文件列表。

```
LOCAL_SRC_FILES := hello-jni.c
```

这里，hello-jni 模块只由一个源文件生成，而 LOCAL_SRC_FILES 变量可以包含用空

格分开的多个源文件名。

至此，**Android.mk** 文件中定义的构建系统变量简单描述了原生项目。编译和生成实际模块的构建系统还需要包含合适的构建系统片段，具体需要包含哪些片段取决于想要生成模块的类型。

1. 构建共享库

为了建立可供主应用程序使用的模块，必须将该模块变成共享库。Android NDK 构建系统将 **BUILD_SHARED_LIBRARY** 变量设置成 **build-shared-library.mk** 文件的保存位置。该 **Makefile** 片段包含了将源文件构建和组装成共享库的必要过程：

```
include $(BUILD_SHARED_LIBRARY)
```

hello-jni 是一个简单的模块；然而，除非你的模块需要特殊处理，否则 **Android.mk** 文档将会包含一模一样的流程和指令。

2. 构建多个共享库

基于不同的应用程序的体系结构，一个单独的 **Android.mk** 文档可能产生多个共享库模块。为了达到这个目的，需要如程序清单 2-2 所示在 **Android.mk** 文档中定义多个模块。

程序清单 2-2 带有多个共享库模块的 **Android.mk** 构建文件

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

#
# 模块 1
#
include $(CLEAR_VARS)

LOCAL_MODULE := module1
LOCAL_SRC_FILES := module1.c
include $(BUILD_SHARED_LIBRARY)

#
# 模块 2
#
include $(CLEAR_VARS)

LOCAL_MODULE := module2
LOCAL_SRC_FILES := module2.c

include $(BUILD_SHARED_LIBRARY)
```

在处理完这个 **Android.mk** 构建文档之后，Android NDK 构建系统会产生 **libmodule1.so** 和 **libmodule2.so** 两个共享库。

3. 构建静态库

Android NDK 构建系统也支持静态库。实际的 Android 应用程序并不直接使用静态库，并且应用程序包中也不包含静态库。静态库可以用来构建共享库。例如，在将第三方代码添加到现有原生项目中时，不用直接将第三方源代码包括在原生项目中，而是将第三方代码编译成静态库然后并入共享库，如程序清单 2-3 所示。

程序清单 2-3 在 Android.mk 文件中使用静态库

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

#
# 第三方 AVI 库
#
include $(CLEAR_VARS)

LOCAL_MODULE := avilib
LOCAL_SRC_FILES := avilib.c platform_posix.c

include $(BUILD_STATIC_LIBRARY)

#
# 原生模块
#
include $(CLEAR_VARS)

LOCAL_MODULE := module
LOCAL_SRC_FILES := module.c

LOCAL_STATIC_LIBRARIES := avilib

include $(BUILD_SHARED_LIBRARY)
```

在将第三方代码模块生成静态库之后，共享库就可以通过将它的模块名添加到 `LOCAL_STATIC_LIBRARIES` 变量中来使用该模块。

4. 用共享库共享通用模块

静态库可以保证源代码模块化；但是，当静态库与共享库相连时，它就变成了共享库的一部分。在多个共享库的情况下，多个共享库与同一个静态库连接时，需要将通用模块的多个副本与不同共享库重复相连，这样就增加了应用程序的大小。在这种情况下，不用构建静态库，而是将通用模块作为共享库建立起来，而动态连接依赖模块以便消除重复的副本(见程序清单 2-4)。

程序清单 2-4 Android.mk 文件中共享库之间的代码共享

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

```

#
# 第三方 AVI 库
#
include $(CLEAR_VARS)

LOCAL_MODULE := avilib
LOCAL_SRC_FILES := avilib.c platform_posix.c

include $(BUILD_SHARED_LIBRARY)

#
# 原生模块 1
#
include $(CLEAR_VARS)

LOCAL_MODULE := module1
LOCAL_SRC_FILES := module1.c

LOCAL_SHARED_LIBRARIES := avilib

include $(BUILD_SHARED_LIBRARY)

#
# 原生模块 2
#
include $(CLEAR_VARS)

LOCAL_MODULE := module2
LOCAL_SRC_FILES := module2.c

LOCAL_SHARED_LIBRARIES := avilib

include $(BUILD_SHARED_LIBRARY)

```

5. 在多个 NDK 项目间共享模块

同时使用静态库和共享库时，可以在模块间共享通用模块。但要说明的是，所有这些模块必须属于同一个 NDK 项目。从 R5 版本开始，Android NDK 也允许在 NDK 项目间共享和重用模块。考虑前面讲过的示例，可以通过以下步骤在多个 NDK 项目间共享 avilib 模块：

- 首先，将 avilib 源代码移动到 NDK 项目以外的位置，例如：C:\android\shared-modules\avilib。为了避免命名冲突，目录结构也可以包含模块提供者的名字，例如：C:\android\shared-modules\transcode\avilib。

注意：

在 Android NDK 构建系统中，共享模块路径不能包含空格。

- 作为共享模块，avilib 需要自己的 Android.mk 文件，如程序清单 2-5 所示。

程序清单 2-5 共享 avilib 模块的 Android.mk 文件

```

LOCAL_PATH := $(call my-dir)

#
#第三方 AVI 库
#
include $(CLEAR_VARS)

LOCAL_MODULE := avilib
LOCAL_SRC_FILES := avilib.c platform_posix.c

include $(BUILD_SHARED_LIBRARY)

```

- 现在，可以将 avilib 模块从 NDK 项目的 Android.mk 文件中移除。为了使用这个共享模块，将以 transcode/avilib 为参数调用函数宏 import-module 部分添加在构建文档的末尾，如程序清单 2-6 所示。为了避免构建系统的冲突，应该将 import-module 函数宏调用放在 Android.mk 文档的末尾。

程序清单 2-6 使用共享模块的 NDK 项目

```

#
# 原生模块
#
include $(CLEAR_VARS)

LOCAL_MODULE := module
LOCAL_SRC_FILES := module.c
LOCAL_SHARED_LIBRARIES := avilib

include $(BUILD_SHARED_LIBRARY)

$(call import-module,transcode/avilib)

```

- import-module 函数宏需要先定位共享模块，然后再将它导入到 NDK 项目中。默认情况下，import-module 函数宏只搜索<Android NDK>/sources 目录。为了搜索 c:\android\shared-modules 目录，定义一个名为 NDK_MODULE_PATH 的新环境变量并将它设置成共享模块的根目录，例如：c:\android\shared-modules。

6. 用 Prebuilt 库

使用共享模块要求有共享模块的源代码，Android NDK 构建系统简单地把这些源文件包含在 NDK 项目中并每次构建它们。自 R5 版本以后，Android NDK 也提供对 Prebuilt 库的支持。在下面的情况下，Prebuilt 库是非常有用的：

- 想在不发布源代码的情况下将你的模块发布给他人。
- 想使用共享模块的预建版来加速构建过程。

尽管已经被编译了，但预建模块仍需要一个 `Android.mk` 构建文档，如程序清单 2-7 所示。

程序清单 2-7 预构建共享模块的 `Android.mk` 文件

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

#
# 第三方预构建 AVI 库
#
include $(CLEAR_VARS)

LOCAL_MODULE := avilib
LOCAL_SRC_FILES := libavilib.so

include $(PREBUILT_SHARED_LIBRARY)
```

`LOCAL_SRC_FILES` 变量指向的不是源文件，而是实际 Prebuilt 库相对于 `LOCAL_PATH` 的位置。

注意：

Prebuilt 库定义中不包含任何关于该库所构建的实际机器体系结构的信息。开发人员需要确保 Prebuilt 库是为与 NDK 项目相同的机器体系结构而构建的。

`PREBUILT_SHARED_LIBRARY` 变量指向 `prebuilt-shared-library.mk` Makefile 片段。它什么都没有构建，但是它将 Prebuilt 库复制到了 NDK 项目的 `libs` 目录下。通过使用 `PREBUILT_STATIC_LIBRARY` 变量，静态库可以像共享库一样被用作 Prebuilt 库，NDK 项目可以像普通共享库一样使用 Prebuilt 库了。

```
...
LOCAL_SHARED_LIBRARIES :=avilib
...
```

7. 构建独立的可执行文件

在 Android 平台上使用原生组件的推荐和支持的方法是将它们打包成共享库。但是，为了方便测试和进行快速原型设计，Android NDK 也支持构建独立的可执行文件。这些独立的可执行文件是不用打包成 APK 文件就可以复制到 Android 设备上的常规 Linux 应用程序，而且它们可以直接执行，而不通过 Java 应用程序加载。生成独立可执行文件需要在 `Android.mk` 构建文档中导入 `BUILD_EXECUTABLE` 变量，而不是导入 `BUILD_SHARED_LIBRARY` 变量，如程序清单 2-8 所示。

程序清单 2-8 独立可执行模块的 `Android.mk` 文件

```
#
# 独立的可执行的原生模块
#
include $(CLEAR_VARS)
```

```

LOCAL_MODULE := module
LOCAL_SRC_FILES := module.c

LOCAL_STATIC_LIBRARIES := avilib

include $(BUILD_EXECUTABLE)

```

BUILD_EXECUTABLE 变量指向 build-executable.mk Makefile 片段，该片段包含了在 Android 平台上生成独立可执行文件的必要步骤。独立可执行文件以与模块相同的名称被放在 libs/<machine architecture>目录下。尽管放在该目录下，但在打包阶段它并没有被包含在 APK 文件中。

8. 其他构建系统变量

除了在前几节提到的变量之外，Android NDK 构建系统还支持其他变量，本节将对这些变量进行简要说明。

构建系统定义的变量有：

- **TARGET_ARCH:** 目标 CPU 体系结构的名称，例如 arm
- **TARGET_PLATFORM:** 目标 Android 平台的名称，例如：android-3
- **TARGET_ARCH_ABI:** 目标 CPU 体系结构和 ABI 的名称，例如：armeabi-v7a
- **TARGET_ABI:** 目标平台和 ABI 的串联，例如：android-3-armeabi-v7a

可被定义为模块说明部分的变量有：

- **LOCAL_MODULE_FILENAME:** 可选变量，用来重新定义生成的输出文件名称。默认情况下，构建系统使用 LOCAL_MODULE 的值作为生成的输出文件名称，但变量 LOCAL_MODULE_FILENAME 可以覆盖 LOCAL_MODULE 的值。
- **LOCAL_CPP_EXTENSION:** C++源文件的默认扩展名是.cpp。这个变量可以用来为 C++源代码指定一个或多个文件扩展名。

```

...
LOCAL_CPP_EXTENSION :=.cpp .cxx
...

```

- **LOCAL_CPP_FEATURES:** 可选变量，用来指明模块所依赖的具体 C++特性，如 RTTI、exceptions 等。

```

...
LOCAL_CPP_FEATURES :=rtti
...

```

- **LOCAL_C_INCLUDES:** 可选目录列表，NDK 安装目录的相对路径，用来搜索头文件。

```

...
LOCAL_C_INCLUDES :=sources/shared-module
LOCAL_C_INCLUDES :=$(LOCAL_PATH)/include
...

```

- **LOCAL_CFLAGS:** 一组可选的编译器标志，在编译 C 和 C++源文件的时候会被传送给编译器。

```
...
LOCAL_CFLAGS :=-DNDEBUG -DPORT=1234
...
```

- **LOCAL_CPP_FLAGS:** 一组可选的编译标志，在只编译 C++源文件时被传送给编译器。
- **LOCAL_WHOLE_STATIC_LIBRARIES:** LOCAL_STATIC_LIBRARIES 的变体，用来指明应该被包含在生成的共享库中的所有静态库内容。

小贴士：

当几个静态库之间有循环依赖时，LOCAL_WHOLE_STATIC_LIBRARIES 很有用。

- **LOCAL_LDLIBS:** 链接标志的可选列表，当对目标文件进行链接以生成输出文件时该标志将被传送给链接器。它主要用于传送要进行动态链接的系统库列表。例如：要与 Android NDK 日志库链接，使用以下代码：

```
LOCAL_LDFLAGS :=-llog
```

- **LOCAL_ALLOW_UNDEFINED_SYMBOLS:** 可选参数，它禁止在生成的文件中进行缺失符号检查。若没有定义，链接器会在符号缺失时生成错误信息。
- **LOCAL_ARM_MODE:** 可选参数，ARM 机器体系结构特有变量，用于指定要生成的 ARM 二进制类型。默认情况下，构建系统在拇指模式下用 16 位指令生成，但该变量可以被设置为 arm 来指定使用 32 位指令。

```
LOCAL_ARM_MODE :=arm
```

该变量改变了整个模块的构建系统行为；可以用.arm 扩展名指定只在 arm 模式下构建特定文件。

```
LOCAL_SRC_FILES :=file1.c file2.c.arm
```

- **LOCAL_ARM_NEON:** 可选参数，ARM 机器体系结构特有变量，用来指定在源文件中应该使用的 ARM 高级单指令流多数据流(Single Instruction Multiple Data, SIMD)(a.k.a. NEON)内联函数。

```
LOCAL_ARM_NEON :=true
```

该变量改变了整个模块的构建系统行为；可以用.neon 扩展名指定只构建带有 NEON 内联函数的特定文件。

```
LOCAL_SRC_FILES :=file1.c file2.c.neon
```

- **LOCAL_DISABLE_NO_EXECUTE:** 可选变量，用来禁用 NX Bit 安全特性。NX Bit 代表 Never Execute(永不执行)，它是在 CPU 中使用的一项技术，用来隔离代码

区和存储区。这样可以防止恶意软件通过将它的代码插入应用程序的存储区来控制应用程序。

```
LOCAL_DISABLE_NO_EXECUTE :=true
```

- **LOCAL_EXPORT_CFLAGS:** 该变量记录一组编译器标志，这些编译器标志会被添加到通过变量 `LOCAL_STATIC_LIBRARIES` 或 `LOCAL_SHARED_LIBRARIES` 使用本模块的其他模块的 `LOCAL_CFLAGS` 定义中。

```
LOCAL_MODULE := avilib
...
LOCAL_EXPORT_CFLAGS := - DENABLE_AUDIO
...
LOCAL_MODULE := module1
LOCAL_CFLAGS := - DDEBUG
...
LOCAL_SHARED_LIBRARIES := avilib
```

编译器在构建 `module1` 时会以 `-DENABLE_AUDIO -DDEBUG` 标志执行。

- **LOCAL_EXPORT_CPPFLAGS:** 和 `LOCAL_EXPORT_CFLAGS` 一样，但是它是 C++ 特定代码编译器标志。
- **LOCAL_EXPORT_LDFLAGS:** 和 `LOCAL_EXPORT_CFLAGS` 一样，但用作链接器标志。
- **LOCAL_EXPORT_C_INCLUDES:** 该变量允许记录路径集，这些路径会被添加到通过变量 `LOCAL_STATIC_LIBRARIES` 或 `LOCAL_SHARED_LIBRARIES` 使用该模块的 `LOCAL_C_INCLUDES` 定义中。
- **LOCAL_SHORT_COMMANDS:** 对于有大量资源或独立的静态/共享库的模块，该变量应该被设置为 `true`。诸如 Windows 之类的操作系统只允许命令行最多输入 8 191 个字符；该变量通过分解构建命令使其长度小于 8 191 个字符。在较小的模块中不推荐使用该方法，因为使用它会让构建过程变慢。
- **LOCAL_FILTER_ASM:** 该变量定义了用于过滤来自 `LOCAL_SRC_FILES` 变量的装配文件的应用程序。

9. 其他的构建系统函数宏

本节概括了 Android NDK 构建系统支持的其他函数宏。

- **all-subdir-makefiles:** 返回当前目录的所有子目录下的 `Android.mk` 构建文件列表。例如，调用以下命令可以将子目录下的所有 `Android.mk` 文件包含在构建过程中：

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

- **this-makefile:** 返回当前 `Android.mk` 构建文件的路径。
- **parent-makefile:** 返回包含当前构建文件的父 `Android.mk` 构建文件的路径。
- **grand-parent-makefile:** 和 `parent-makefile` 一样但用于祖父目录。

10. 定义新变量

开发人员可以定义其他变量来简化他们的构建文件。以 `LOCAL_` 和 `NDK_` 前缀开头的名称预留给 Android NDK 构建系统使用。建议开发人员定义的变量以 `MY_` 开头，如程序清单 2-9 所示。

程序清单 2-9 表示开发人员定义的中间变量的使用方法的 `Android.mk` 文件

```
...
MY_SRC_FILES := avilib.c platform_posix.c
LOCAL_SRC_FILES := $(addprefix avilib/, $(MY_SRC_FILES))
...
```

11. 条件操作

`Android.mk` 构建文件也可以包含关于这些变量的条件操作，例如：在每个体系结构中包含一个不同的源文件集，如程序清单 2-10 所示。

程序清单 2-10 包含条件操作的构建文件 `Android.mk`

```
...
ifeq ($(TARGET_ARCH),arm)
    LOCAL_SRC_FILES += armonly.c
else
    LOCAL_SRC_FILES += generic.c
endif
...
```

2.4.2 Application.mk

`Application.mk` 是 Android NDK 构建系统使用的一个可选构建文件。和 `Android.mk` 文件一样，它也被放在 `jni` 目录下。`Application.mk` 也是一个 GUN Makefile 片段。它的目的是描述应用程序需要哪些模块；它也定义所有模块的通用变量。以下是 `Application.mk` 构建文件支持的变量：

- **APP_MODULES:** 默认情况下，Android NDK 构建系统构建 `Android.mk` 文件声明的所有模块。该变量可以覆盖上述行为并提供一个用空格分开的、需要被构建的模块列表。
- **APP_OPTIM:** 该变量可以被设置为 `release` 或 `debug` 以改变生成的二进制文件的优化级别。默认情况下使用的是 `release` 模式，并且此时生成的二进制文件被高度优化。该变量可以被设置为 `debug` 模式以生成更容易调试的未优化二进制文件。
- **APP_CLAGS:** 该变量列出了一些编译器标志，在编译任何模块的 C 和 C++ 源文件时这些标志都会被传给编译器。
- **APP_CPPFLAGS:** 该变量列出了一些编译器标志，在编译任何模块的 C++ 源文件时这些标志都会被传给编译器。

- **APP_BUILD_SCRIPT:** 默认情况下, Android NDK 构建系统在项目的 jni 子目录下查找 Android.mk 构建文件。可以用该变量改变上述行为, 并使用不同的生成文件。
- **APP_ABI:** 默认情况下, Android NDK 构建系统为 armeabi ABI 生成二进制文件。可以用该变量改变上述行为, 并为其他 ABI 生成二进制文件, 例如:

```
APP_ABI := mips
```

另外, 可以设置多个 ABI

```
APP_ABI := armeabi mips
```

为所有支持的 ABI 生成二进制文件

```
APP_ABI := all
```

- **APP_STL:** 默认情况下, Android NDK 构建系统使用最小 STL 运行库, 也被称为 system 库。可以用该变量选择不同的 STL 实现。

```
APP_STL :=stlport_shared
```

- **APP_GNUSTL_FORCE_CPP_FEATURES:** 与 LOCAL_CPP_EXTENSIONS 变量相似, 该变量表明所有模块都依赖于具体的 C++特性, 如 RTTI、exceptions 等。
- **APP_SHORT_COMMANDS:** 与 LOCAL_SHORT_COMMANDS 变量相似, 该变量使得构建系统在有大量源文件的情况下可以在项目中使用更短的命令。

2.5 使用 NDK-Build 脚本

如前所述, 可以通过执行 ndk-build 脚本启动 Android NDK 构建系统。该脚本用一组参数使维护和控制构建过程更容易。

- 默认情况下, ndk-build 脚本应该在主项目目录中执行。-C 参数可以用于指定命令行中 NDK 项目的位置, 这样一来 ndk-build 脚本可以从任意的位置开始。

```
ndk-build -C /path/to/the/project
```

- 如果源文件没被修改, Android NDK 构建系统不会重构建目标。可以用-B 执行 ndk-build 脚本来强制重构建所有源代码。

```
ndk-build -B
```

- 为了清理生成的二进制文件和目标文件, 可以在命令行执行 ndk-build clean 命令。Android NDK 构建系统会删除生成的二进制文件。

```
ndk-build clean
```

- Android NDK 构建系统依赖于 GNU Make 工具对模块进行构建。默认情况下, GNU Make 工具一次执行一句构建命令, 等这一句完成以后再执行下一句。如果在命令

行使用-j 参数, GNU Make 就可以并行执行构建命令。另外,也可以通过指定该参数之后的数字来指定并行执行的命令总数。

```
ndk-build -j 4
```

2.6 排除构建系统故障

Android NDK 构建系统有大量的日志以支持构建系统相关的故障排除, 本节将简要阐述构建系统故障排除。

在命令行输入 ndk-build NDK_LOG=1 便可启用 Android NDK 构建系统内部状态日志功能。Android NDK 构建系统会产生大量的日志, 日志消息的前缀是 Android NDK(如图 2-13 所示)。

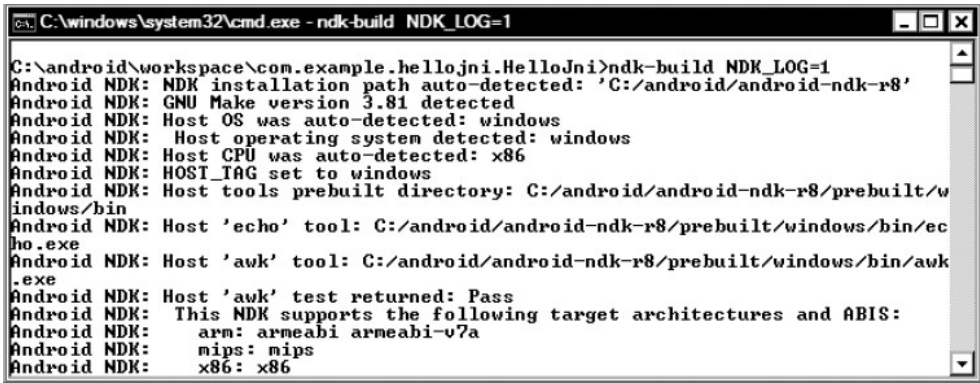


图 2-13 ndk-build 脚本显示调试信息

如果只想看实际执行的构建命令, 可以在命令行输入 ndk-build V=1。Android NDK 将会只显示构建命令, 如图 2-14 所示。



图 2-14 ndk-build 脚本显示构建命令

2.7 小结

本章引导读者学习了在 Eclipse IDE 环境下构建 NDK 项目的方法，在此过程中讲述了很多 Android NDK 构建系统的知识，第 3 章将继续学习用 Android NDK 构建的原生组件如何与实际的 Java 应用程序通信。