移动互联应用开发系列

细说 Android 4.0 NDK 编程

王家林◎著

本书顺应 Android 软件、硬件、云计算整合潮流
详细剖析了 NDK 开发过程中会涉及的各类问题和解决方案
本书还特别介绍了 Android UI 编程技术，对源代码进行了细致的剖析

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

移动互联应用开发系列

细说 Android 4.0 NDK 编程

王家林　著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

通过 NDK，应用程序可以非常方便地实现 Java 与 C/C++代码的相互沟通。本书顺应 Android 软/硬件、云计算整合潮流，详细剖析了 NDK 开发中涉及的各类问题和解决方案：搭建 Android NDK 开发环境的每一步细节，开发第一个 Android NDK 程序，Android NDK 中 Java 与 C/C++代码的互相调用，Facade 设计模式在 NDK 中的美妙应用，NDK 与软/硬件整合，NDK 与云计算等。本书还特别介绍了 Android UI 编程技术。

本书适合从事 Android 开发的人员阅读。

图书在版编目（CIP）数据

细说 Android 4.0 NDK 编程/王家林著. —北京：电子工业出版社，2012.7
（移动互联应用开发系列）
ISBN 978-7-121-16140-7

Ⅰ. ①细… Ⅱ. ①王… 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（2012）第 034874 号

责任编辑：周宏敏　　文字编辑：施易含
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1 092　1/16　印张：11.5　字数：294 千字
印　　次：2012 年 7 月第 1 次印刷
印　　数：4 000 册　　定价：39.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

几乎对任何一位移动互联网应用程序开发者而言，都会面临或者思考这样一个问题：在Android 上，到底什么样的应用程序更赚钱？

笔者认为：符合 Android 软/硬件、云计算整合潮流的应用程序更赚钱。

一般而言，大多数 Android 开发者都考虑如何使自己开发的应用程序“跑”在更多不同设备上，所以在开发应用程序时，Android 应用开发工程师往往会舍弃不同 Android 设备具体的特性，使用各个平台的共同属性来进行应用程序开发和适配，以达到跨平台运行的目的。这样开发出来的程序能够在很多不同的 Android 设备上运行，但在每一个平台上运行的用户体验都会很平庸，结果就不会赚到很多钱。

Android 应用程序跨平台问题的根源在于没有充分发挥 Android 设备本身的硬件特性，此类应用程序很难在这个特定的平台上有出色的表现，也就很难俘获用户的芳心，其结果必然是无法在市场上取得成功。

到目前为止，软/硬件、云计算整合的最大成功者是苹果公司，苹果公司深度整合自己的硬件、平台软件、APP Store、云服务，在商业上获得了空前的成功，我们来看一下 Android 智能手机领域主要的成功者：HTC、摩托罗拉、三星等，都极为擅长软/硬件、云计算的整合，如HTC Sense。

作为 Android 应用程序开发者，如何顺应 Android 软/硬件、云计算整合潮流从而在市场上取得更大的成功呢？

笔者认为：使用 NDK 开发自己的应用程序。

通过 NDK，应用程序可以非常方便地实现 Java 与 C/C++代码的相互沟通，进而达到以下目的：

（1）Java 调用 C/C++的 so 库文件，基于 Android 出色的架构，so 库文件又可以非常方便地通过 HAL 使用硬件的特性，这样开发出的应用程序就具备了独特的市场竞争力。

（2）云服务可以非常方便地集成到 libraries 中，这样，通过 NDK，应用程序就可以非常方便地使用云服务，这是众多云服务提供商希望看到的事情。

（3）此时的应用程序更加具有吸引力，就会促进硬件的销售，而硬件大量的销售又促进了应用程序和云服务在终端市场上的成功。

本书顺应 Android 软/硬件、云计算的整合潮流，详细剖析了 NDK 开发过程中会涉及的各类问题和解决方案，希望本书能够为读者在开发的过程中带来帮助。

本书源代码可在电子工业出版社官方网站（www.phei.com.cn）进行下载，也可通过微盘（关注 http://weibo.com/pheicombook，进入微盘下载）。

目录 CONTENTS

第 1 章　Android 4.0 开发环境搭建和测试 …… 1

1.1　下载所需要的软件 …… 2
1.2　安装所需要的软件 …… 3
1.3　第一个 Android 4.0 程序 …… 11
1.4　剖析 Android 4.0 程序的组织结构 …… 17
1.5　Android 4.0 模拟器无 3G 信号的解决方案 …… 33

第 2 章　使用 C 语言编程 …… 35

2.1　下载并安装 C 语言交叉编译工具链 …… 36
2.2　第一个 C 语言程序 …… 37
2.3　在 Android 上安装、授权、运行 C 语言程序 …… 39
2.4　采用动态链接的方式生成可执行文件并在 Android 上安装、授权、运行 C 程序 …… 43
2.5　解决采用动态链接方式生成的可执行文件执行时的“Segmentation fault”问题 …… 47

第 3 章　搭建 Android NDK 开发环境并开发第一个 Android NDK 程序 …… 49

3.1　下载 Windows 下开发 Android NDK 所需的软件 …… 50
3.2　安装 Windows 下 Android NDK 开发环境 …… 50
3.3　配置 Cygwin …… 52
3.4　开发第一个 Android NDK 程序 …… 54

第 4 章　Android NDK 中的代码调用 …… 61

4.1　NDK 与 JNI 的关系 …… 62
4.2　JNI 中的 JavaVM 与 JNIEnv 对象 …… 62
4.3　Android NDK 中 Java 通过 JNI 调用 C 的步骤 …… 63
4.4　本地 C 代码调用 Java 中的 Method …… 63
4.5　本地 C 代码获得 Java 对象的属性值 …… 71
4.6　多个类中有本地 C 代码的调用 …… 76
4.7　Java、Dalvik VM、C/C++的运行机制与流程 …… 81
4.8　Java 中分配线程调用 C/C++函数 …… 82

第 5 章　NDK 的架构/设计模式及 NDK 与软/硬件整合、云计算 …… 89

5.1　NDK 的架构图及思考 …… 90
5.2　Facade 设计模式剖析 …… 91
5.3　Facade 设计模式在 JNI 中的应用 …… 96
5.4　Facade 设计模式在 NDK 中的应用 …… 97
5.5　NDK 的优势与不足 …… 97
5.6　NDK 与软/硬件整合 …… 98
5.7　NDK 与云计算 …… 99

附录 A　Android UI 编程 …… 101

附录 B　如何成为 Android 高手 V2.0：结合云计算和智能终端、软/硬件整合 …… 159

第1章

Android 4.0开发环境搭建和测试

1.1 下载所需要的软件

Android 开发需要的工具如下所示。

1．JDK 5/6/7

需要注意的是仅有 JRE 是不够的，JRE 是 Java 的运行环境，而 JDK 不仅包含了 JRE，还包含了开发 Java 程序所需要的工具集合。

JDK 可以到 http://www.oracle.com/technetwork/java/javase/downloads/index.html 进行下载。

单击“JDK Download”，如下图所示。

进入下载界面，单击“Accept License Agreement”并选择“Windows x86”进行下载。

2．Eclipse-jee-indigo-SR1

使用 MyEclipse 也可以，但由于 MyEclipse 是收费的，并且插件较多影响运行速度，因此不建议采用。Eclipse 是一个开放源代码的，基于 Java 的可扩展的集成开发环境（IDE）。Eclipse 中可以集成进多种插件，以完成特定语言的开发。

下载地址：http://www.eclipse.org/downloads/。页面中的 Eclipse IDE for Java EE Developers 链接如下图所示。

单击后进入下图所示界面，选择“Windows 32-bit”。

此时进入如下图所示界面，单击“[China] Actuate Shanghai (http)”下载即可。

3．Android SDK

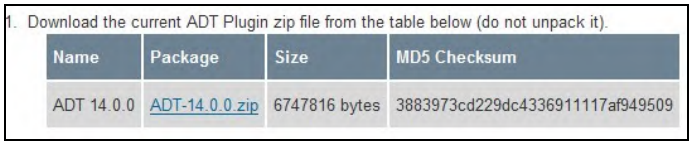
SDK 是开发 Android 应用程序的软件开发工具包。

下载地址：http://developer.android.com/sdk/index.html，进入下载页面后选择“android-sdk_r14-windows.zip”，单击即可下载。

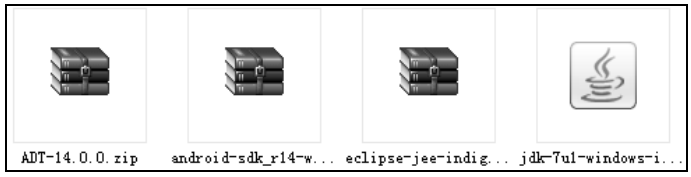
4．Eclipse 的插件 ADT（Android Development Tools）

Android 开发工具（ADT）是一个为 Eclipse IDE 设计的，旨在提供一个强大的、集成的环境来建立 Android 应用程序的插件。ADT 扩展了 Eclipse 的功能，可以快速建立新的 Android 项目，创建一个应用程序界面。它添加了基于 Android 框架 API 的组件，使用 Android SDK 工具调试应用程序，甚至导出签名（或未签名）APKs 以分发应用程序。在 Eclipse 中强烈建议使用 ADT 进行开发，ADT 提供了令人难以置信的提高开发 Android 应用程序的效率。

下载地址：http://developer.android.com/sdk/eclipse-adt.html，如下图所示。



准备好这些工具，我们就可以安装这些软件来搭建 Android 的开发环境了。有一点需要注意，以上的链接部分会由于官方的更新而产生变动，有时下载到的版本不同，但下载的方式一样，如果遇到问题可以参考官方帮助文档。具体的下载文件如下图所示。



1.2　安装所需要的软件

1．安装 JDK 7

（1）找到 JDK 安装文件，双击后，进入运行界面。

单击“下一步”按钮，进入安装路径选择界面。

（2）默认安装的位置是“C:\Program Files\Java\jdk1.7.0_01\”，建议在安装 JDK 的路径中不要出现汉字或空格，以避免出现未知的错误。单击“更改”按钮，将路径修改为“G:\Android\Java\”。

单击“确定”按钮后，界面如下图所示。

将上图中的“演示程序和样例”选中。之后单击“下一步”按钮，进入安装 JDK 的过程。

（3）修改 JRE 安装位置，更改为“G:\Android\Java\jre7\”。

单击“确定”按钮后，进入安装界面。

（4）完成后单击“下一步”按钮，进行安装。

出现安装完成界面后，单击“完成”按钮即可完成 JDK 的安装。

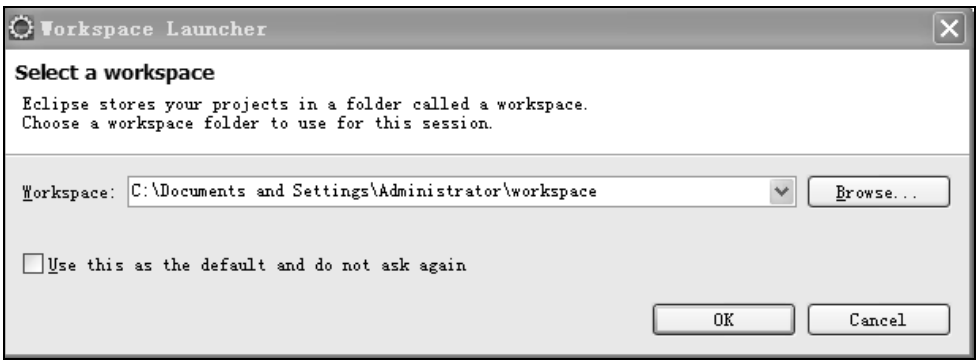
为了测试 JDK 安装是否成功，在 Windows 中，单击“开始”→“运行”，在对话框中输入 cmd，单击“确定”按钮，之后输入 java-version，回车。若出现如下图所示信息，则说明安装成功。

2．安装 Eclipse

在安装完 JDK 之后，开始安装开发工具 Eclipse。Eclipse 的安装比较简单，直接找到文件 eclipse-jee-indigo-SR1-win32.zip，将其解压缩到指定的位置即可。

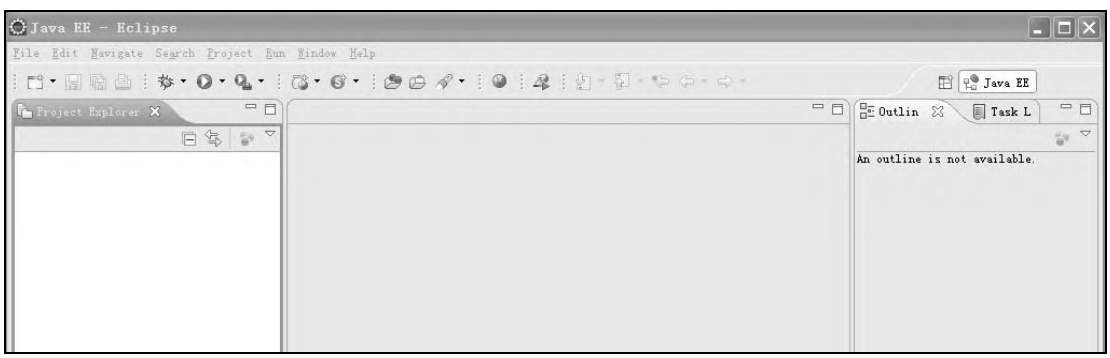
在这里将 Eclipse 放在了 G:\Android 路径下，打开 eclipse 文件夹，双击“eclipse.exe”即可运行 Eclipse。

初次启动 Eclipse，会遇到如下图所示的提示界面。



提示选择自己的工作空间（Workspace）路径，可以单击“Browse…”按钮选择自己的 Workspace 存放路径，选择存放于 G:\Android\eclipse\workspace。如果不希望下次打开 Eclipse 时有该提示，可以单击上图中“Use this as the default and do not ask again”前面的单选框。

单击“OK”按钮，会进入 Eclipse 的首页欢迎界面。关闭“Welcome”界面，即进入了 Eclipse 的主界面，如下图所示。

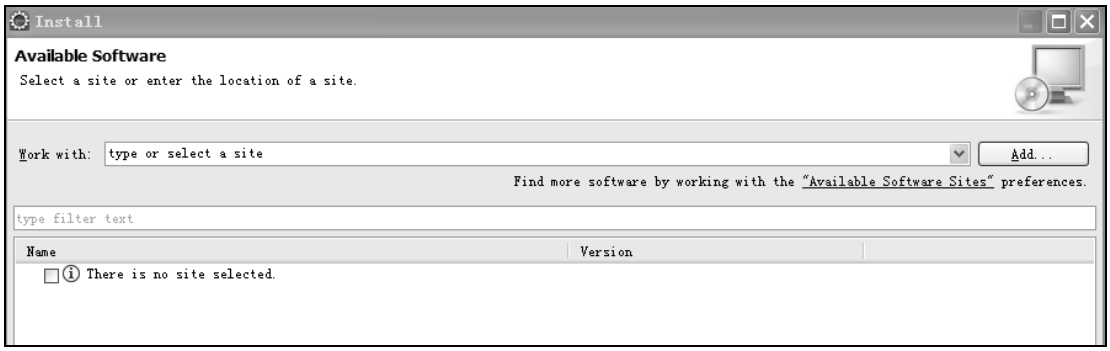


此时 Eclipse 安装完毕。

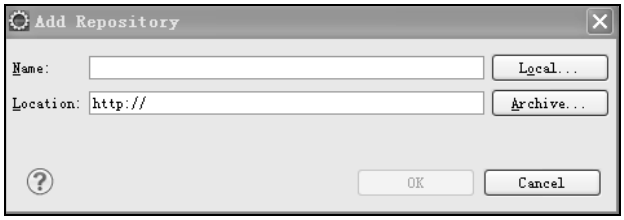
3．安装 Eclipse 的 ADT 插件，用于 Android 快速开发

开发 Android 应用时需要用到 Android 提供的 Eclipse 插件，英文全称是 Android Development Tools，下面为 Eclipse 安装 ADT 插件。

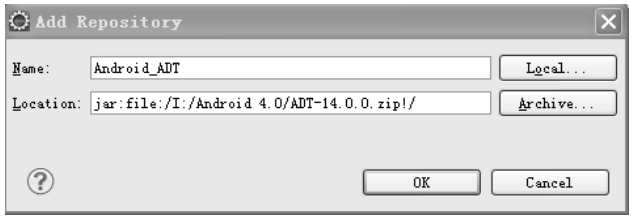
在主界面中选择“Help”→“Install New Software”，此时出现如下图所示的对话框。



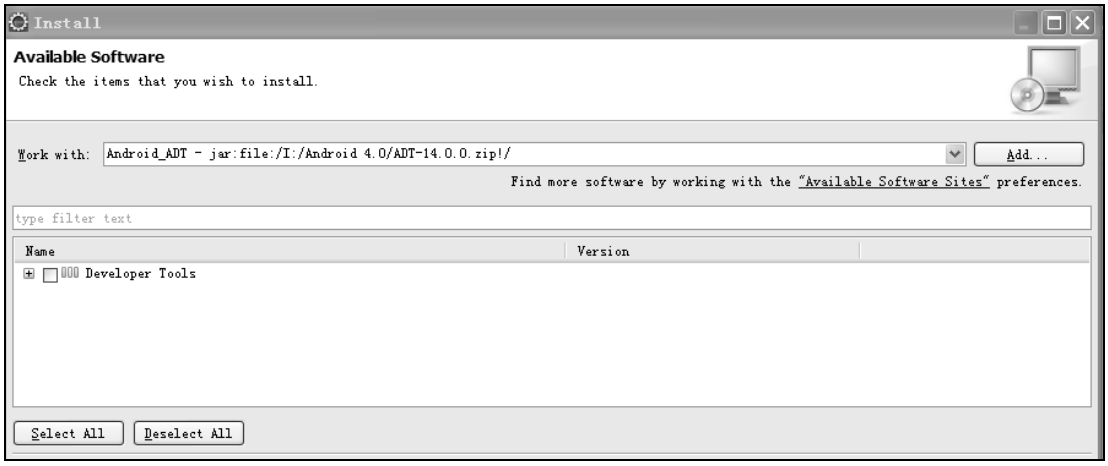
单击“Add”按钮，出现如下图所示的对话框。



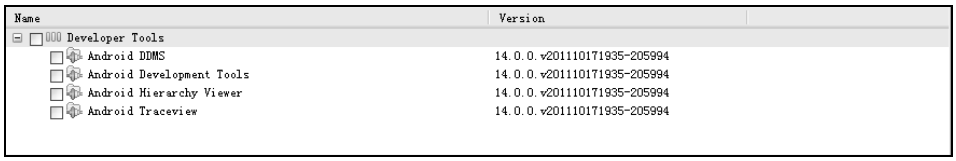
在对话框的 Name 一栏输入“Android_ADT”。然后单击“Archive…”按钮，浏览和选择已经下载的 ADT 插件压缩文件 ADT-14.0.0.zip 的路径，单击“打开”按钮，进入如下图所示的界面。



单击“OK”按钮，返回可用软件的视图，如下图所示。

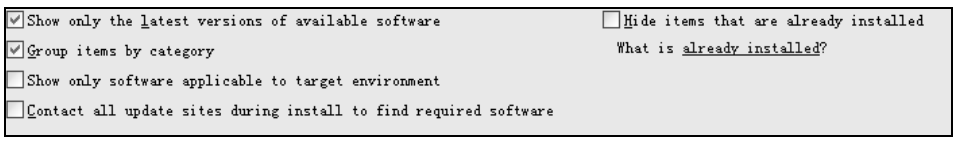


单击“Developer Tool”前的加号，进入如下图所示的界面。

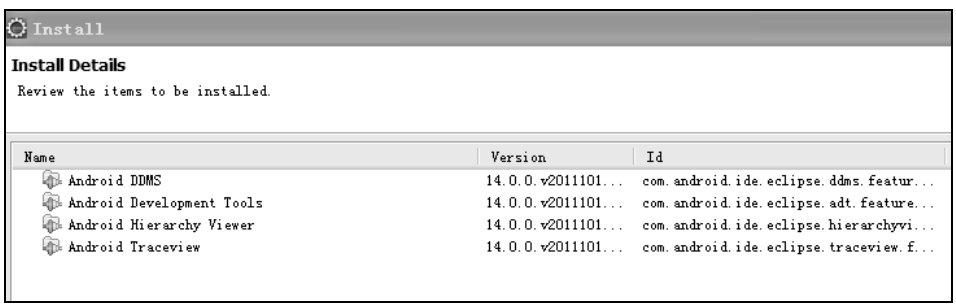


然后选择“Developer Tools”前的复选框，选择所有的内容。

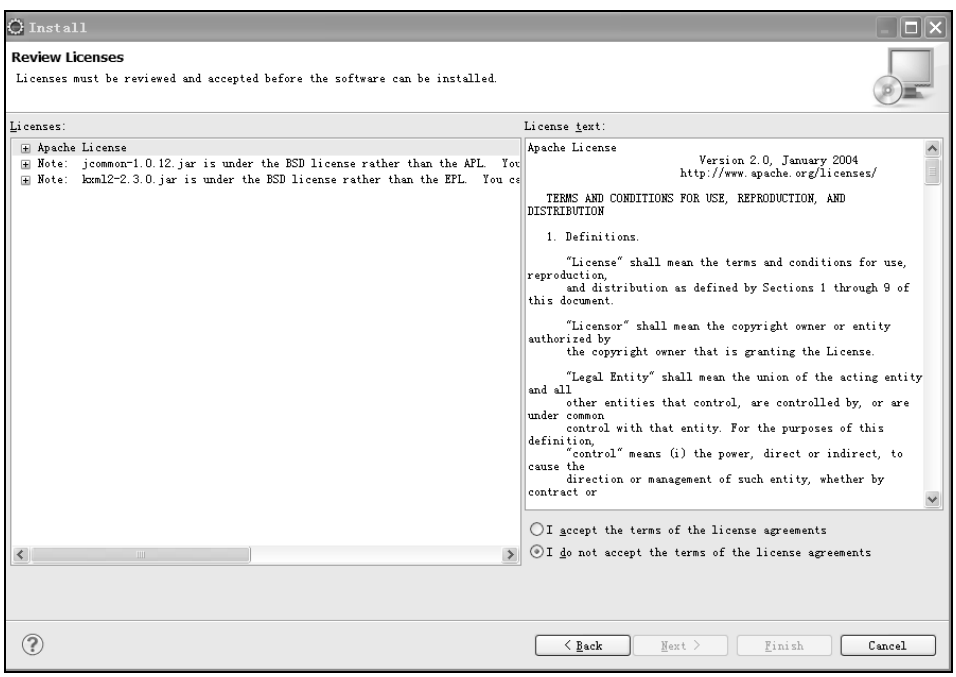
此时把同一个页面下的“Contact all update sites during install to find required software”前面的复选框去掉，因为 ADT 已经是最新的版本，不需要连接网络检查信息，这样会加快安装速度，如下图所示。



单击“Next”按钮，自动运行到如下图所示界面。



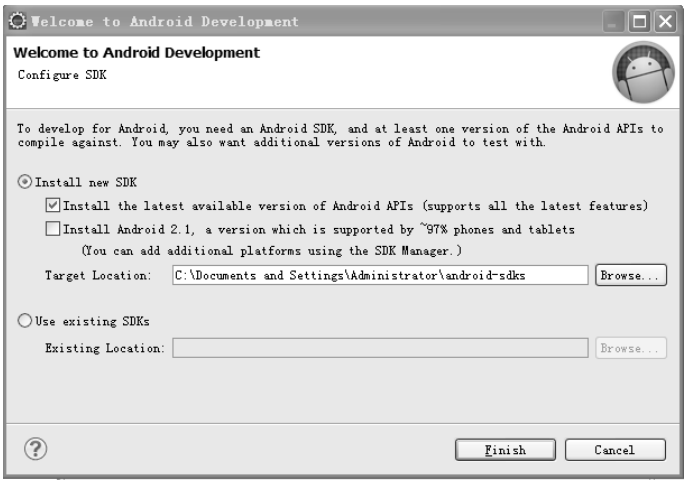
单击“Next”按钮，出现如下图所示界面。



选中“I accept the terms of the license agreements”前的单选框，单击“Finish”按钮，此时会自动弹出如下对话框。

单击“OK”按钮继续安装，然后会自动看到如下图所示的界面。

单击“Restart Now”按钮，此时 Eclipse 自动重新启动，重新启动后，Eclipse 会自动弹出如下图所示的对话框。



接下来会提示我们安装 Android 的 SDK 开发包。单击“Cancel”按钮，此时 Eclipse 的工具栏出现了如下视图。

表明 ADT 插件安装成功。

4．安装 Android SDK

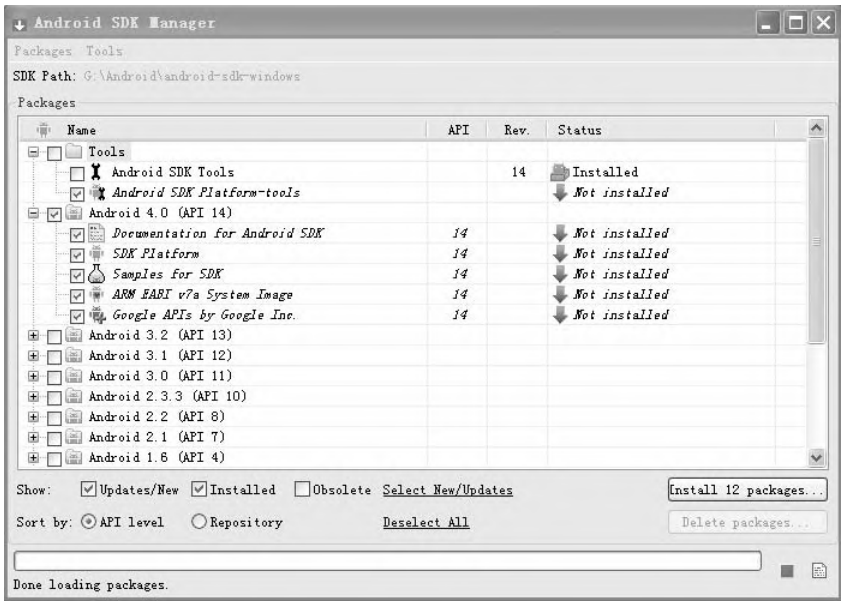
（1）解压 SDK 压缩包。

把 android-sdk_r14-windows.zip 文件解压到计算机上合适位置。笔者是解压至 G:\Android 路径下。

解压缩后，目录如下图所示。

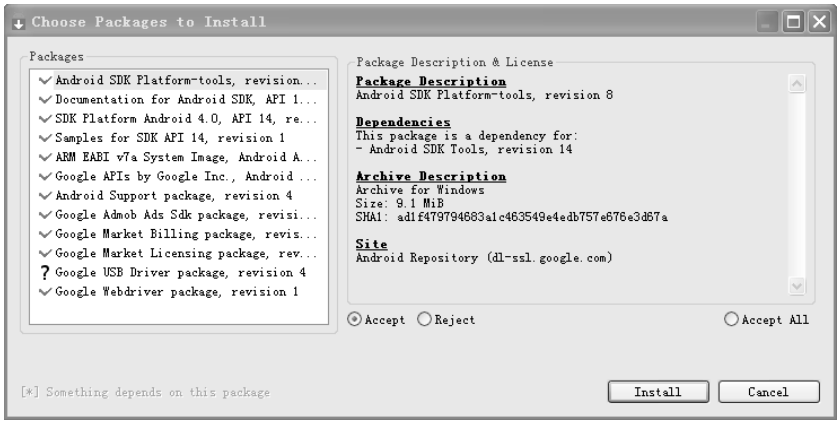
（2）单击 SDK Manager.exe 下载所需要的 Android 4.0 开发平台组件。

此时程序自动运行到如下图所示界面。

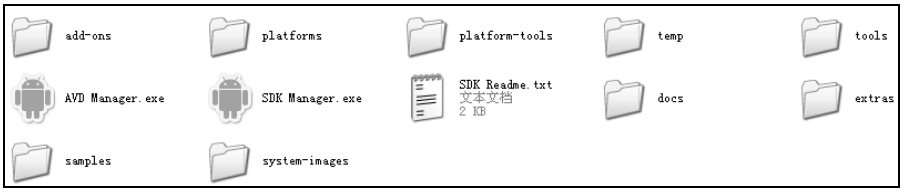


系统默认选择了 Android SDK 4.0 开发所需要的所有组件。

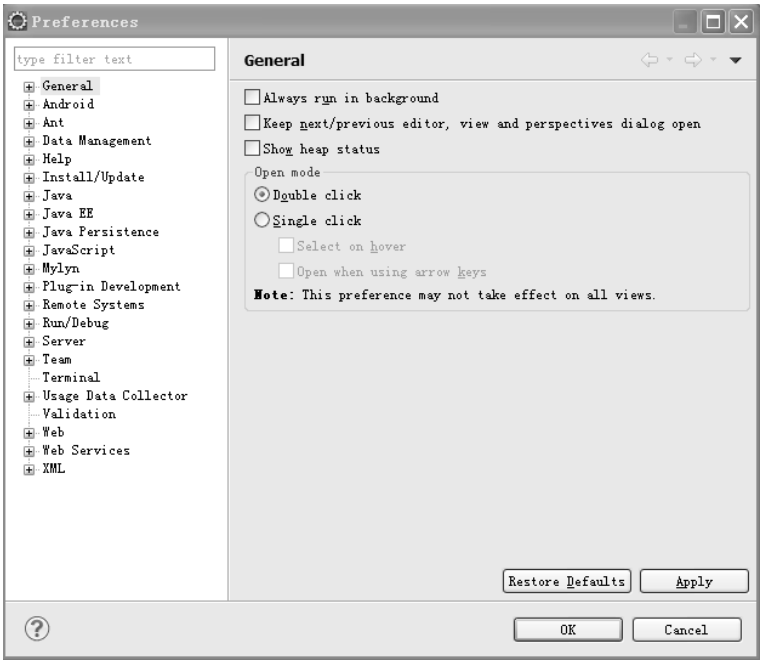
此时，单击“install 12 packages…”，出现如下图所示的界面。



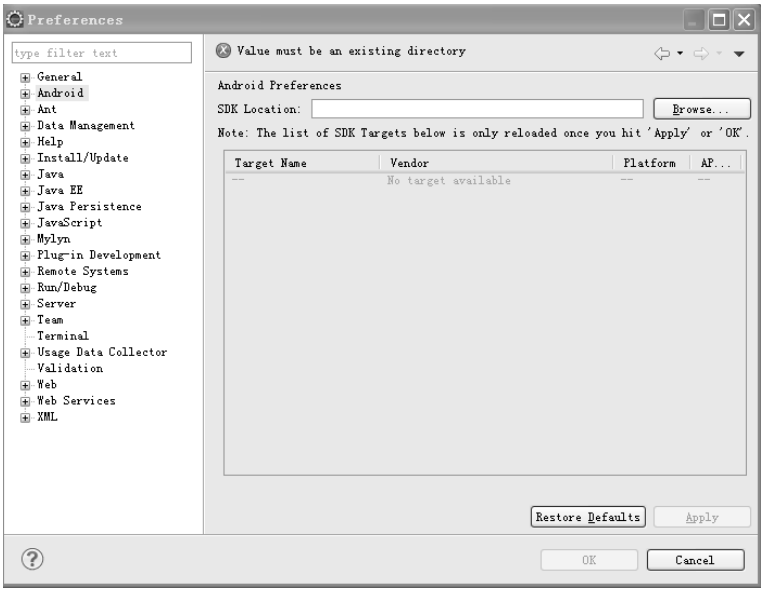
选择右下角的“Accept All”后，单击“Install”按钮，此时程序会自动通过网络下载所需组件。下载过程很长，下载完毕后的文件视图如下图所示。



（3）启动 Eclipse，选择“Window”→“Preferences”，出现如下视图。

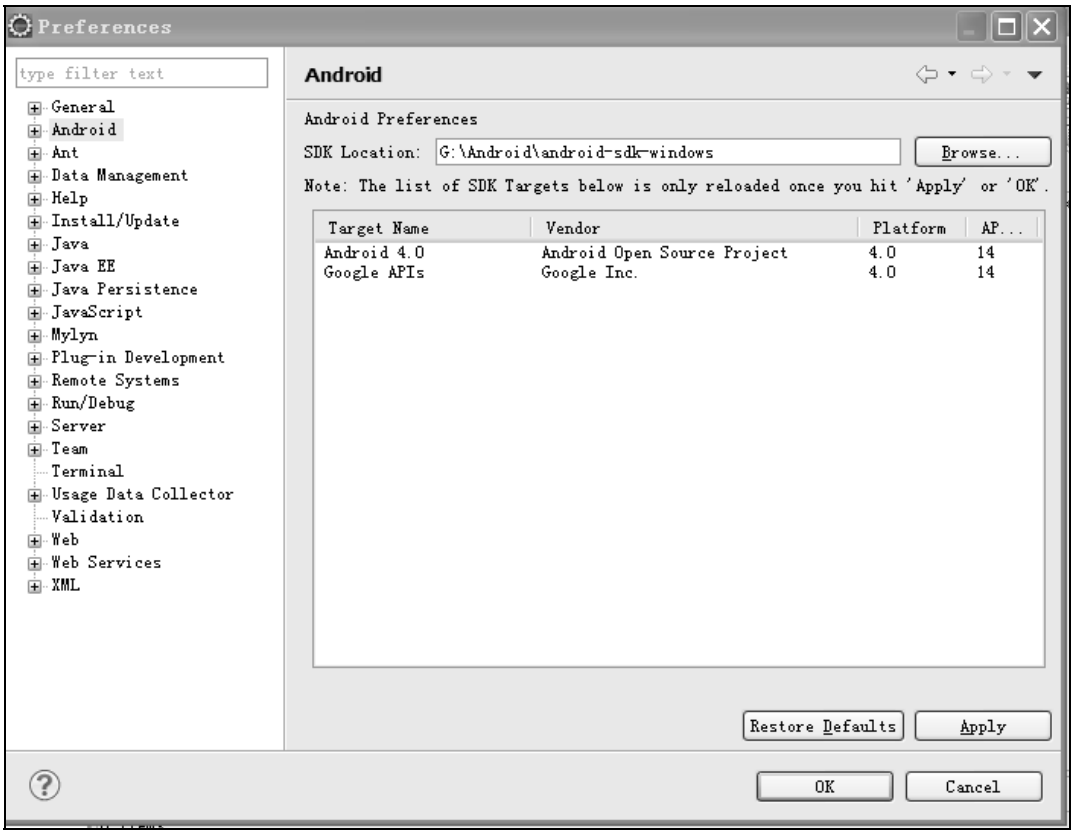


（4）在打开的视图左侧单击“Android”，如下图所示。



在上图右侧的 SDK Location 中单击“Browse…”按钮选择 Android SDK 所在位置笔者选择为“android_sdk_Windows”目录。

单击“确定”按钮后，再单击“Apply”按钮，出现如下视图。



单击“OK”按钮即可完成 Android SDK 的安装。

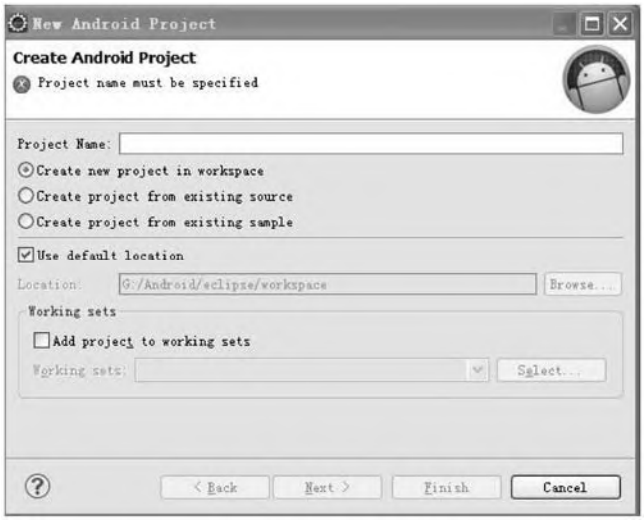
至此，Android 4.0 的开发环境已经搭建完成。

1.3　第一个 Android 4.0 程序

搭建好了开发环境，笔者用一个 HelloWorld 程序的开发来测试开发环境并说明开发流程。

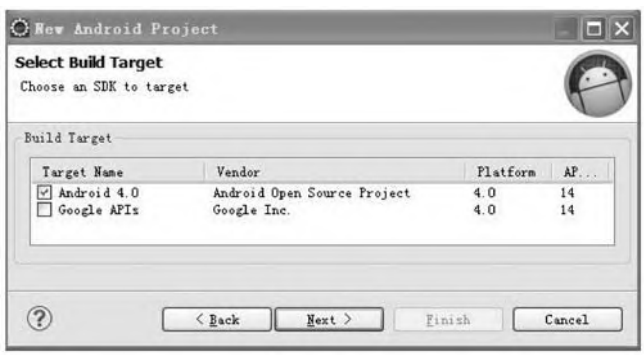
创建 Android Project（HelloWorld 项目）。

打开 Eclipse，单击左上角的“File”之后单击“New”，然后单击“Android Project”，出现如下图所示的窗口。

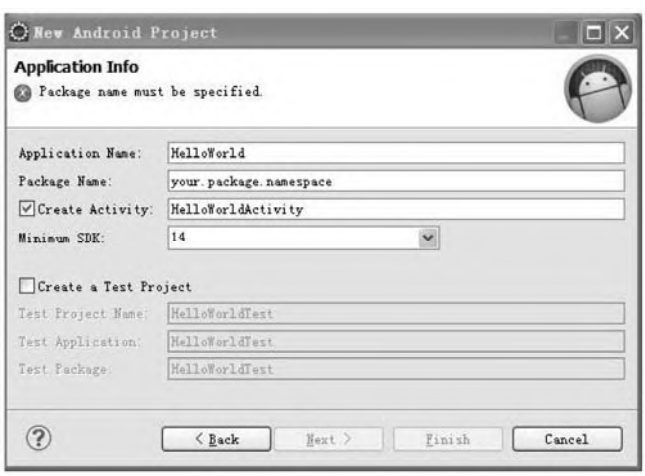


此处设置“Project Name”为“HelloWorld”。

单击“Next”按钮，进入如下视图。



默认已经选择了 Android 4.0 平台，此时我们只需单击“Next”按钮即可，出现如下图所示窗口。



笔者把上图中的 Package Name 改为 com.wangjialin.hello，其他的保持不变。

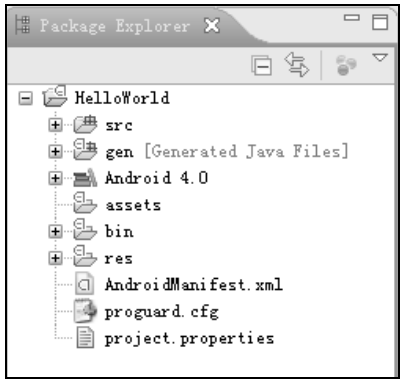
单击“Finish”按钮，即可完成 HelloWorld 项目的创建。

New Android Project 选项卡属性介绍如下表所示。

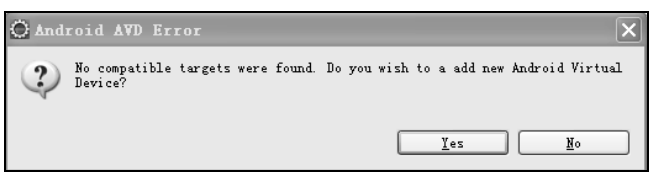
New Android Project 选项卡属性介绍

Project name	项目名称，本例指定为 HelloWorld
Contents	设定项目存放的位置，默认放于 Workspace 中
Build Target	设定项目运行的目标版本，有 Android 4.0 和 Google API 4.0 两个版本，我们使用默认的 Android 4.0，这就意味着这个项目是基于 Android 4.0 版本开发的
Application name	本项目的应用名称为 HelloAndroid 应用名称会在手机程序列表中该应用的图标下方显示，并且在该项目运行时应用名称会在标题栏中显示
Package name	本项目的包名为“com.wangjialin.hello” 包结构是 Java 语言的一种规范
Create Activity	ADT 会根据此名字自动为项目创建同名的 Activity 类，建议以 Activity 作为后缀，方便阅读和理解。该项可选，如果不需要 ADT 自动生成 Activity，则可以不选 本例指定为“HelloWorldActivity”
Min SDK Version	14 这个数字代表了该项目运行的 Android 平台的最低版本是 Android 4.0，默认情况下比 4.0 低的版本都不能运行该项目

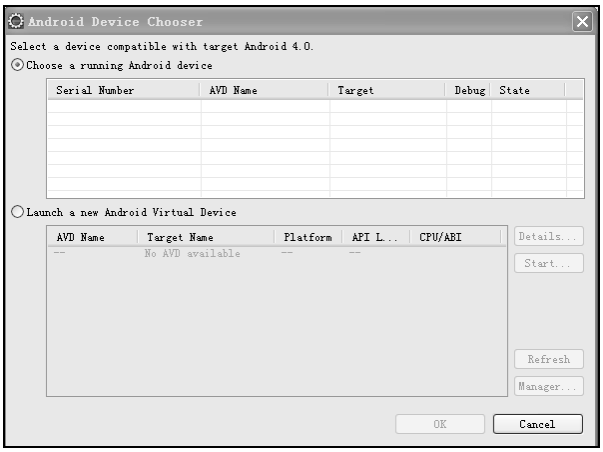
项目创建之后，Package Explorer 结构图如下图所示。



截止目前为止，虽然没有写下任何一行代码，但是该项目已经可以运行了，这是由于我们使用 ADT 生成的每一个项目本身就是一个可运行的项目。此时如果右键单击项目的名称 HelloWorld，并选择“Run As”，接着选择“Android Application”，会弹出如下图所示的对话框。



单击“Yes”按钮，此时弹出如下图所示的对话框。



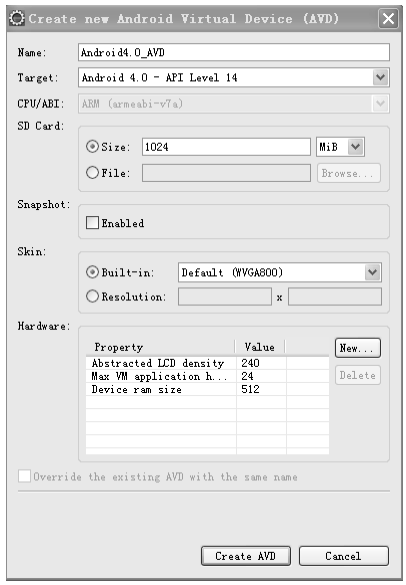
因为还没有创建模拟器，所以没有可供选择的模拟器，单击“Cancel”按钮，进入模拟器管理界面，在管理界面右上角单击“New”按钮，创建一个新的模拟器。

设置相关属性，如下表所示。

属性及作用表

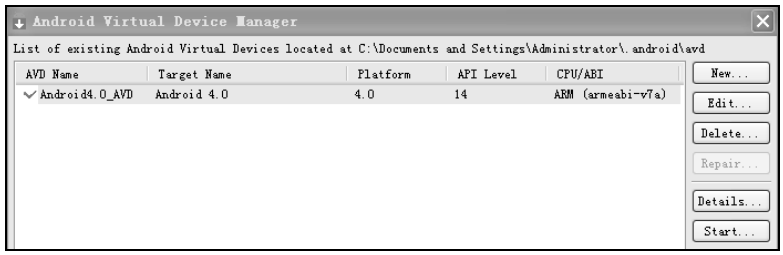
Name	该 AVD 被指定的名称，此处设置为 Android 4.0_AVD
Target	属性是指该 AVD 所搭载的 Android 的版本，本 AVD 选用的是 Android 4.0-API Level 14
SD Card	有 size 选项，是指该 AVD 所安装的 SDCard 的大小，本例为 1024MiB。创建的 SD Card 是以镜像文件的形式存放，file 属性指定该镜像文件存放于硬盘上的位置，可以不进行修改，使用默认设置即可。并非每次创建时都新建一个 SD Card，可以使用以前创建过的 SD Card，并访问其中的数据
Skin	可以指定手机屏幕的格式，在此均采用默认，可以自行设置。分辨率应与开发应用的目标机型进行匹配，若目标机型的分辨率比较特殊，可以选择 Resolution，并指定相应参数

在新建的模拟器中“Name”中输入 Android 4.0_AVD，“Target”中输入 Android 4.0-API Levdl 14，“Size”为 1024，界面如下图所示。

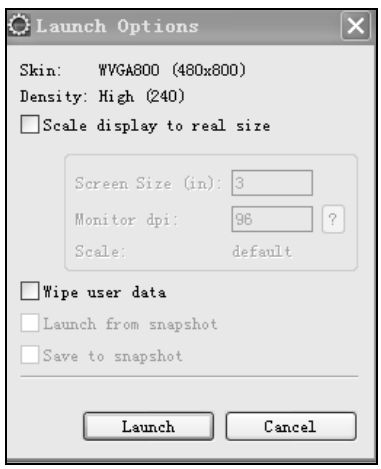


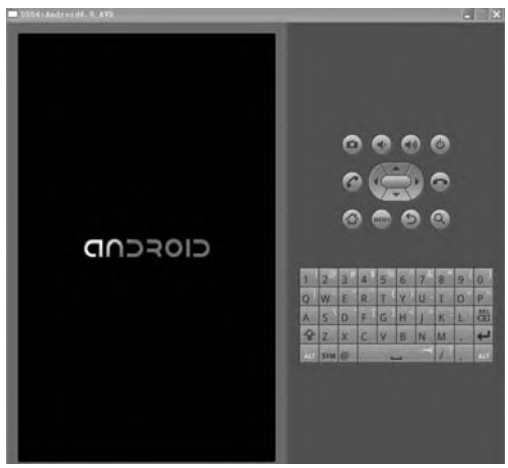
单击“Create AVD”按钮在 AVD 列表中出现刚才创建的 AVD——“Android 4.0_AVD”的信息。

选中 AVD 列表中要启动的 AVD 之后，单击右侧的“Start”按钮，如下图所示。

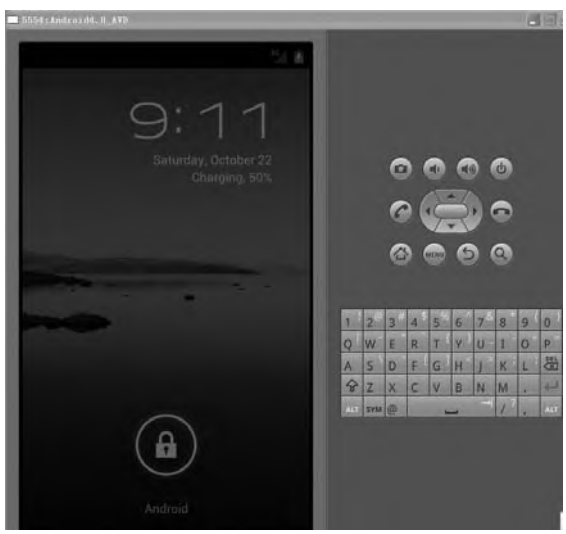


此时出现“Launch Options”界面，如下（左）图所示。单击“Launch”按钮，等待模拟器启动，如下（右）图所示。

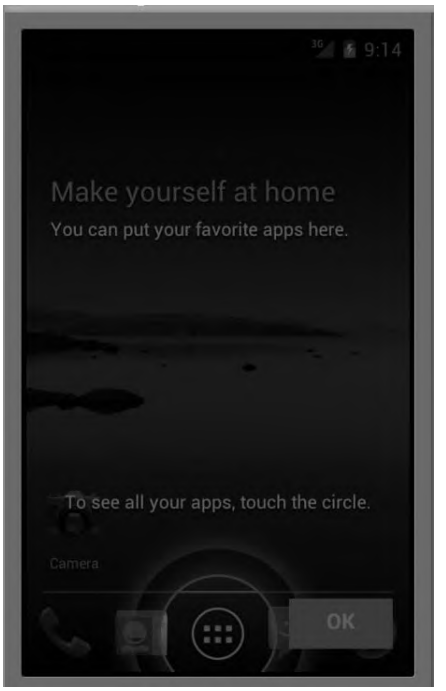




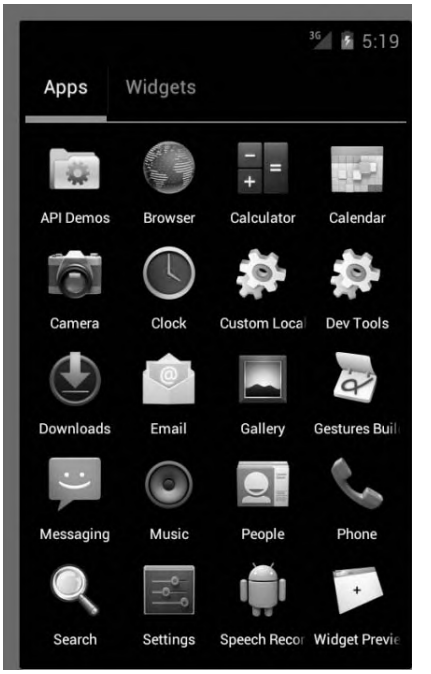
启动完毕后，如下图所示。



单击锁的图标并往右拖曳，解锁屏幕。解锁后，如下（左）图所示。单击“OK”图标就进入了 Android 4.0 的主屏幕，如下（右）图所示。



单击图标，进入应用程序快速启动区，如下（左）图所示。单击右下角的“OK”图标，即可进入程序列表视图，如下（右）图所示。



此时模拟器成功创建并成功运行。

有了模拟器，这时就可以运行第一个 Android 4.0 程序了。

此时如果右键单击项目的名称“HelloWorld”，并选择“Run As”，接着选择“Android Application”，弹出如下（左）图所示的启动模拟器的对话框。成功启动后，进入如下（右）图所示界面。

单击锁的图标并往右拖曳，解锁屏幕。此时进入的视图如下图所示。

表明第一个 Android 4.0 程序已经运行成功。

1.4　剖析 Android 4.0 程序的组织结构

1．目录结构介绍

资源是 Android 应用程序不可或缺的部分。资源是你想包含和引入到应用程序里面的一些

外部元素，如图片、音频、视频、文本字符串、布局、主题等。每个 Android 应用程序包含一个资源目录（res/）和资产目录（assets/），但资产不常用，因为它们的应用很少，仅在需要读取原始字节流时才需要保存数据在 assets/目录。Res/和 assets/目录均在 Android 项目树的顶端，和源代码目录（src/）处在同一级上。资源和资产从表面上看没多大区别，不过总体上，在存储外部内容时资源用得更多。真正的区别在于任何放置在资源目录里的内容可以通过应用程序的 R 类访问，这是被 Android 编译过的。而任何存放在资产目录里的内容会保持它的原始文件格式，为了读取它，必须使用 AssetManager 来以字节流的方式读取文件。所以保持文件和数据在资源中（res/）中会更方便访问。

2．Resource 目录及其下文件详解

res/目录下可以有以下几个子目录，部分目录开发工具并没有自动创建，根据需要可以自行创建，介绍如下所示。

（1）src/：专门存放编写的 Java 源代码的包。

（2）android 4.0/：存放 Android 自身的 jar 包。

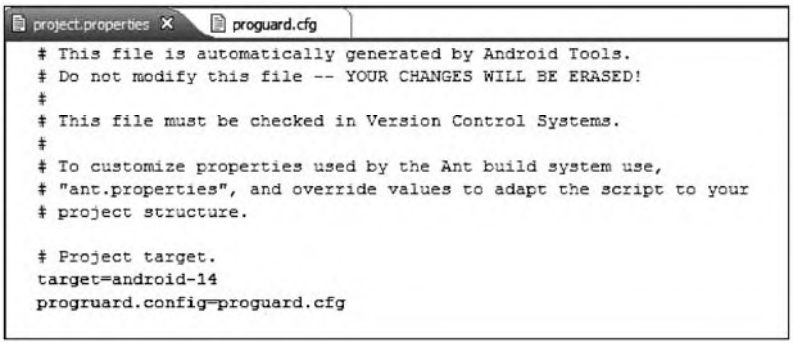
（3）gen/：该目录不用开发人员维护，但又是非常重要的目录。用来存放由 Android 开发工具所生成的目录。该目录下的所有文件都不是我们创建的，而是由 ADT 自动生成的。该目录下的 R.java 文件非常重要。

（4）assets/：该目录用来存放应用中用到的类似于视频文件、MP3 等媒体文件。

（5）res/：res 是 resource 的缩写，称该目录为资源目录。该目录可以存放一些图标、界面文件、应用中用到的文字信息。

（6）AndroidManifest.xml：该文件是功能清单文件，列出了应用中所使用的所有组件，如 Activity，以及后面要学习的广播接收者、服务等组件。

（7）proguard.cfg：Proguard 是 Android 的混淆器的配置文件，用来防止程序被反编译，如果需要使用该功能，只需要在 project.properties 文件中添加一行配置信息 progruard.config=proguard.cfg，如下图所示。



（8）project.properties：该文件存放了项目对应的一些环境配置，如应用要求运行的最低 Android 版本，同时可以包含 proguard 的配置等信息。

资源被编译到最终的 APK 文件里。Android 创建了一个被称为 R 的类，这样在 Java 代码

中可以通过它关联到对应的资源文件。R 类包含的子类的命名由资源在 res/目录下的文件夹名称所决定。

关于 res/子目录的一些补充说明。

（1）res/drawable。

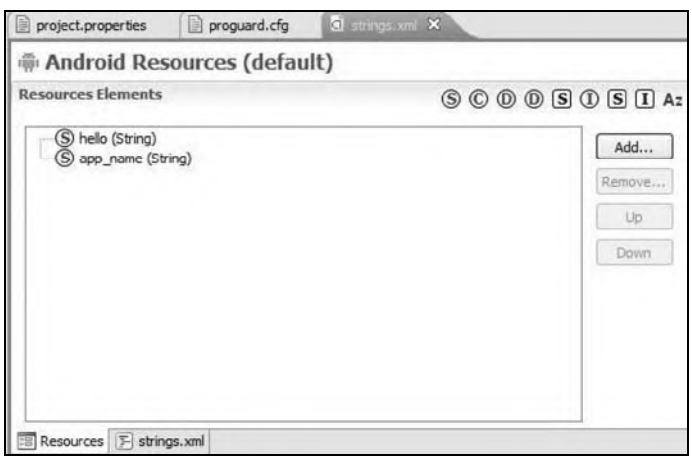
res/目录下有三个 dawable 文件夹——drawable-*dpi，区别只是将图标按分辨率高低来放入不同的目录。drawable-hdpi 用来存放高分辨率的图标；drawable-mdpi 用来存放中等分辨率的图标；drawable-ldpi 用来存放低分辨率的图标。程序运行时可以根据手机分辨率的高低选取相应目录下的图标。不过，如果不想准备过多的图片，那么也可以只准备一张图标，将其放入三个目录的任何一个中去。

（2）res/values/文件夹下常放的文件如下所示。

① strings.xml。

用来定义字符串和数值，在 Activity 中使用 getResources().getString(resourceId)或 getResources().getText(resourceId)取得资源。

在 Eclipse 中对 res/values 下的文件左键双击或右键单击后选择“Open With”→“Android Resource Editor”编辑器打开。默认是 Resources 显示，单击旁边的“strings.xml”，显示代码内容，如下图所示。



HelloWorld 项目的 strings.xml 文件内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World, HelloWorldActivity!</string>
    <string name="app_name">HelloWorld</string>

</resources>
```

每个 string 标签声明了一个字符串，name 属性指定其引用名。

为什么需要把应用中出现的文字单独存放在 strings.xml 文中呢？

原因有二，一是为了国际化，Android 建议将在屏幕上显示的文字定义在 strings.xml 中，如果今后需要进行国际化，例如，我们开发的应用本来是面向国内用户的，当然要在屏幕上使用中文，而如今我们要让应用走向世界，打入日本市场，当然需要在手机屏幕上显示日语，如果没有把文字信息定义在 strings.xml 中，就需要修改程序内容了。但当我们把所有屏幕上出现的文字信息都集中存放在 strings.xml 文件之后，只需要再提供一个 strings.xml 文件，将里面的汉字信息都修改为日语，再运行程序时，Android 操作系统会根据用户手机的语言环境和国家来自动选择相应的 strings.xml 文件，这时手机界面就会显示出日语。这样做非常方便国际化。二是为了减少应用的体积，降低数据冗余。假设在应用中要使用“我们一直在努力”这段文字 10 000 次，如果我们不将“我们一直在努力”定义在 strings.xml 文件中，而是在每次使用时直接写上这几个字，这样下来程序中将有 70 000 个字，这 70 000 个字占 136 KB 的空间。而由于手机的资源有限，其 CPU 的处理能力及内存是非常有限的，136 KB 对于手机程序来说是个不小的空间，我们在开发手机应用时一定要记住“能省内存就省内存”。而如果将这几个字定义在 strings.xml 中，在每次使用到的地方通过 Resources 类来引用该文字，只占用了 14 B，因此对降低应用体积效果是非常有效的。当然我们可能在开发时并不会用到这么多的文字信息，但是“不以善小而不为，不以恶小而为之”，作为手机应用开发人员，一定要养成良好的编程习惯。

② arrays.xml。

用来定义数组，在 Activity 中使用 getResources().getStringArray(resourceId)获取一个 String 数组。

代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="sports">
        <item>足球</item>
        <item>篮球</item>
        <item>太极</item>
        <item>乒乓球</item>
    </string-array>
</resources>
```

③ colors.xml。

用来定义颜色和颜色字串数值，可以在 Activity 中使用 getResources().getDrawable(resourceId)以及 getResources().getColor(resourceId)取得这些资源。代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
  <color name="contents_text">#ff000000</color>
</resources>
```

④ dimens.xml。

用来定义尺寸数据，在 Activity 中使用 getResources().getDimension(resourceId)取得这些资源，代码如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <dimen name="height">80dip</dimen>
</resources>
```

⑤ styles.xml。

定义样式，代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="sharpText">
        <item name="android:textSize">18px</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

需要注意的是，Android 中的资源文件不要以数字作为文件名，否则会导致错误。

（3）res/layout/目录下的布局文件简介。

本例中的布局文件是 ADT 默认自动创建的 main.xml 文件。在 Eclipse 中，双击 main.xml 文件，或者右键单击以选择相应的编辑器，选用“Android Layout Editor”。在编辑区出现“Layout 编辑器”的预览效果。

可以单击 Layout 选项卡旁边的 main.xml，切换到代码模式以编辑。

与在网页中布局中使用 HTML 文件一样，Android 在 XML 文件中使用 XML 元素来设定屏幕布局。每个文件包含整个屏幕或部分屏幕，被编译进一个视图资源，可以被传递给 Activity.setContentView 或被其他布局文件引用。文件保存在工程的 res/layout/ 目录下，它被 Android 资源编辑器编译。

下面介绍 HelloWorld 项目的 main.xml 文件的内容，其代码如下所示。

```
main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
```

```
        android:orientation="vertical" >

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/hello" />

    </LinearLayout>
```

下面逐元素进行分析说明。

① <LinearLayout>元素。

LinearLayout 翻译成中文是线性布局，所谓线性布局就是在该元素下的所有子元素会根据其“orientation”属性的值来决定是按行或者是按列逐个显示。

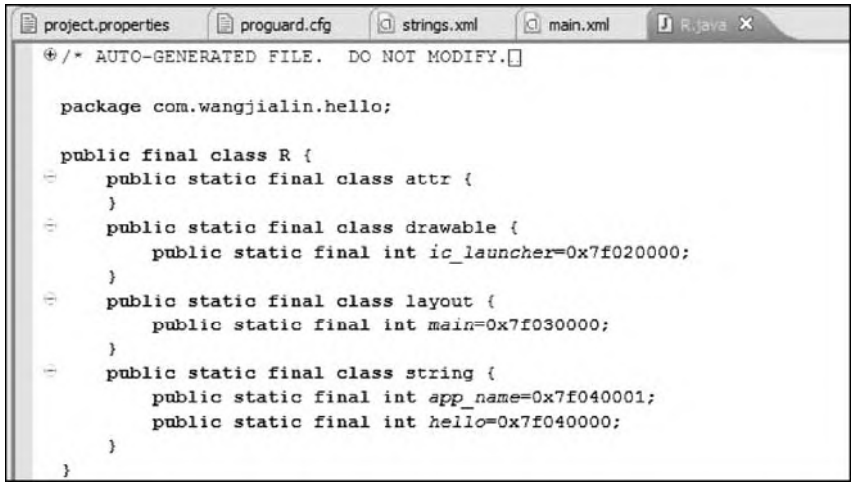
② <TextView>元素。

该元素与 HTML 中的<label>元素比较相似，也是一种显示控件。

其属性 text 指定在该元素上面显示的文字内容。建议将该文字内容在 strings.xml 文件中进行定义，之后再在 main.xml 文件中通过“@string/stringName”的方式进行引用。

3．gen/目录下的 R.java 文件详解

R.java 文件中默认有 attr、drawable、layout、string 四个静态内部类，每个静态内部类分别对应一种资源，如 layout 静态内部类对应 layout 中的界面文件，其中每个静态内部类中的静态常量分别定义一条资源标识符，如“public static final int main=0x7f030000”对应的是 layout 目录下的 main.xml 文件，如下图所示。



由于目前 drawable-*dpi 目录下都只有 icon.png 一个图片文件，因此此时不同像素的同名的 icon.png 文件在 drawable 内部类中只有一个 icon 属性。如果我们在 drawable-*dpi 目录下再添加一幅图片，那么会有什么效果出现呢？如下图所示。

project.properties　proguard.cfg　strings.xml　main.xml　R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.wangjialin.hello;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
        public static final int sharpandroid=0x7f020001;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

sharpandroid.png

现在已经理解了 R.java 文件中内容的来源，也即是当开发者在 res/目录中任何一个子目录中添加相应类型的文件之后，ADT 会在 R.java 文件中相应的匿名内部类当中自动生成一条静态 int 类型的常量，对添加的文件进行索引。如果在 layout 目录下再添加一个新的界面，那么在 public static final class layout 中也会添加相应的静态 int 常量。相反，当在 res 目录下删除任何一个文件后，其在 R.java 中对应的记录会被 ADT 自动删除。例如，在 strings.xml 中添加一条记录，在 R.java 的 string 内部类中也会自动增加一条记录。

R.java 文件会给开发程序带来很大的便利，例如，在程序中使用 public static final int icon= 0x7f020000 就可以找到其对应的 icon.png 图片。

R.java 文件除了有自动标识资源的“索引”功能之外，还有另一个主要的功能，当 res 目录中的某个资源在应用中没有被使用到时，在该应用被编译时系统就不会把对应的资源编译到该应用的 APK 包中，这样可以节省 Android 手机的资源。

4．组件标识符

通过对 R.java 文件的介绍，已经了解了 R 文件的索引作用，它可以检索到应用中需要使用的资源。下面介绍如何通过 R.java 文件来引用所需要的资源。

（1）在 Java 程序当中，可以按照 Java 的语法来引用。

① R.resource_type.resource_name。

需要注意的是，resource_name 不需要文件的后缀名。例如，上面的 icon.png 文件的资源标识符可以通过如下方式获取。

```
R.drawable.icon
```

② android.R.resource_type.resource_name。

Android 系统本身自带了很多的资源，也可以进行引用，只是需要在前面加上“android.”以声明该资源来自 Android 系统。

（2）在 XML 文件中引用资源的语法如下所示。

① @[package:]type/name。

使用自己包下的资源可以省略 package。

在 XML 文件中，main.xml 以及 AndroidMainfest.xml 文件通过@drawable/icon 的方式获取。其中@代表的是 R.java 类，drawable 代表 R.java 中的静态内部类 drawable，/icon 代表静态内部类 drawable 中的静态属性 icon。而该属性可以指向 res 目录下的 drawable-* dpi 中的 icon.png 图标。

其他类型的文件也比较类似。凡是在 R 文件中定义的资源都可以通过@ Static_inner_classes_name/resourse_name 的方式获取，如@id/button，@string/app_name。

② 如果访问的是 Android 系统中带的文件，则要添上包名“android:”，如 android:textColor="@android:color/red"。

（3）@+id/string_name 表达式。

顺便说一下，在布局文件中需要为一些组件添加 id 属性作为标示时，可以使用表达式@+ id/string_name，其中“+”表示在 R.java 的名为 id 的内部类中添加一条记录。如"@+id/button"的含义是在 R.java 文件中的 id 这个静态内部类添加一条常量名为 button，该常量就是该资源的标识符。如果 id 这个静态内部类不存在，则会先生成它。通过该方式生成的资源标识符，仍然可以以@id/string_name 的方式引用。示例代码片段如下所示。

```
<RelativeLayout
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     >
 <Button
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/cancle_button"
     android:layout_alignParentRight="true"
     android:id="@+id/cancle"/>
  <Button
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_toLeftOf="@id/cancle"
     android:layout_alignTop="@id/cancle"
     android:text="@string/ok_button"/>
  </RelativeLayout>
```

其中，android:id="@+id/cancle"将其所在的 Button 标识为 cancle，在第二个 Button 中通过

"@id/cancle"对第一个 Button 进行引用。

5．AndroidManifest.xml 介绍

如果要查看 AndroidManifest.xml 的详细信息，可以到帮助文档中去了解。路径如下：“Dec Guide”→“The AndroidManifest.xml File”超链接。

每个应用程序都有一个功能清单文件 AndroidManifest.xml（一定是这个名字）在它的根目录里。这个清单文件给 Android 系统提供了关于这个应用程序的基本信息，系统在运行任何程序代码之前必须知道这些信息。今后在开发 Activity、Broadcast、Service 过程中都要在 AndroidManifest.xml 中进行定义。另外如果使用到系统自带的服务，如拨号服务、应用安装服务、GPRS 服务等都必须在 AndroidManifest.xml 中声明权限。

AndroidManifest.xml 主要包含以下功能：

- 命名应用程序的 Java 应用包，这个包名用来唯一标识应用程序。
- 描述应用程序的组件——活动、服务、广播接收者、内容提供者；对实现每个组件和公布其功能（比如，能处理哪些意图消息）的类进行命名。这些声明使得 Android 系统了解这些组件以及它们在什么条件下可以被启动。
- 决定应用程序组件运行在哪个进程里面。
- 声明应用程序所必须具备的权限，用于访问受保护的部分 API 以及和其他应用程序交互。
- 声明应用程序其他的必备权限，用于组件之间的交互。
- 列举测试设备 Instrumentation 类，用来提供应用程序运行时所需的环境配置及其他信息，这些声明只在程序开发和测试阶段存在，发布前将被删除。
- 声明应用程序所要求的 Android API 的最低版本级别。
- 列举 application 所需要链接的库。

程序中使用的所有组件都会在功能清单文件中列出来，所以先分析功能清单文件。只有下列元素才是功能清单文件中的合法元素，应用开发者不能添加自己的元素或属性。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <application>
        <activity>
```

```
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> …</intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> … </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> … </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <path-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>
</manifest>
```

下面以 HelloWorld 项目的功能清单文件为例进行讲解。

```
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.wangjialin.hello"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk android:minSdkVersion="14" />

   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:label="@string/app_name"
         android:name=".HelloWorldActivity" >
         <intent-filter >
            <action android:name="android.intent.action.MAIN" />
```

```
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

以下详细介绍各个标签。

（1）<manifest>元素。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.sharpandroid.activity"
     android:versionCode="1"
     android:versionName="1.0">
```

该元素是 AndroidManifest.xml 文件的根元素，为必选。根据 XML 文件的语法，xmlns:android 指定该文件的命名空间。功能清单文件会使用 http://schemas.android.com/apk/res/ android 所指向的一个文件。Package 属性是指定 Android 应用所在的包，以后会经常说到“应用的包”，“应用的包”就是指该属性的内容。Android:versionCode 指定应用的版本号。如果应用需要不断升级，在升级的时候应该修改该值。Android:versionName 是版本名称，名称的取定可根据爱好而定。

（2）<application>元素。

```
<application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name" >
       <activity
          android:label="@string/app_name"
          android:name=".HelloWorldActivity" >
          <intent-filter >
             <action android:name="android.intent.action.MAIN" />

             <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
       </activity>
    </application>
```

<application>是非常重要的一个元素，今后开发的许多组件都会在该元素下定义的。该元素为必选元素。

<application>的 icon 属性是用来设定应用的图标。

该表达式指向的是 icon.png 图片。在 Eclipse 中双击 icon.png 图片，对照关系如下所示。

```
public final class R {
        public static final class attr {
        }
        public static final class drawable {
```

```
        public static final int ic_launcher=0x7f020000;
        public static final int sharpandroid=0x7f020001;
    }
```

<application>的 label 属性用来设定应用的名称。指定其属性值所用的表达式@string/app_name 含义与上面的表达式@drawable/icon 一样，同样是指向 R.java 文件中的 string 静态内部类中的 app_name 属性所指向的资源。在这里它指向的是 strings.xml 文件中的一条记录 app_name，其值为“Android，你好”，因此，这种表达方式等价于“android:label="Android，你好！""。

两者结合起来，当程序发布到模拟器上之后，会在应用程序的快速启动区中显示效果，如下图所示。

（3）<activity>元素。

<activity>元素的作用是注册一个 Activity 信息，当我们在创建 HelloWorld 这个项目时，指定了 Create Activity 属性为 HelloActivity，之后 ADT 在生成项目时自动创建了一个 Activity 名称就是 HelloActivity.java，Activity 在 Android 中属于组件，它需要在功能清单文件中进行配置。

<activity>元素的 name 属性指定的是该 Activity 的类名。我们可以看到这个属性值.HelloActivity 中的“.”，“.”代表的是在上面<manifest>元素的 package 属性中指定的当前包。因而.HelloActivity 的含义等价于 com.wangjialin.helllo.HelloActivity.java。

如果 Acitivity 在应用程序包的子包中，必须在声明的时候在子包前加上“.”。

Activity 只能放在“应用的包”或者其子包里面，而不能在“应用的包”以外的包中，这一点必须牢记。

<activity>元素的 label 属性表示 Activity 所代表的屏幕的标题，其属性值的表达式在上面

已经介绍过了，不再赘述。

该属性值在 AVD 程序运行到该 Activity 所代表的界面时，会在标题上显示该值，如下图所示。

（4）<intent-filter>元素。

翻译成中文是意图过滤器。首先简单介绍什么是意图（Intent）。应用程序的核心组件（活动、服务和广播接收器）通过意图被激活，意图代表的是你要做的一件事情，代表你的目的，Android 寻找一个合适的组件来响应这个意图，如果需要会启动这个组件一个新的实例，并传递给这个意图对象。

组件通过意图过滤器（intent filters）通告它们所具备的功能——能响应的意图类型。由于 Android 系统在启动一个组件前必须知道该组件能够处理哪些意图，那么意图过滤器需要在 manifest 中以<intent-filter>元素指定。一个组件可以拥有多个过滤器，每一个描述该组件具有的不同能力。一个指定目标组件的显式意图将会激活那个指定的组件，意图过滤器不起作用。但是一个没有指定目标的隐式意图只在它能够通过组件过滤器中任一过滤器时才能激活该组件。

第一个过滤器：

```
    <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
```

是最常见的。它表明这个 Activity 将在应用程序加载器中显示，就是用户在设备上看到的可供加载的应用程序列表。换句话说，这个 Activity 是应用程序的入口，是用户选择运行这个应用程序后所见到的第一个 Activity。

（5）权限 Permissions。

HelloWorld 项目的功能清单文件中并没有出现 Permissions 元素，但是 Permission 也是一个非常重要的节点。Permission 是代码对设备上数据的访问限制，这个限制被引入来保护可能会被误用而曲解或破坏用户体验的关键数据和代码，如拨号服务、短信服务等。每个许可被一个唯一的标签所标识。这个标签常常指出了受限的动作。例如，下表是一些 Android 定义的权限。

Android 定义的权限

permisson name 值	作　　用
android.permissin.CALL_PHONE	电话服务权限
android.permission.SEND_SMS	发送短信服务权限
android.permission.RECEIVE_SMS	接收短信服务权限
android.permission.READ_PHONE_STAT	监听电话权限

续表

permisson name 值	作　用
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	在 SD Card 中创建与删除文件权限
android.permission.WRITE_EXTERNAL_STORAGE	往 SD Card 写入数据权限
android.permission.RECEIVE_BOOT_COMPLETED	开机启动，电池电量变化，时间改变等 Intent 权限
android.permission.RECORD_AUDIO	音频刻录权限
android.permission.CAMERA	照相机权限
android.permission.INSTALL_PACKAGES	安装程序权限

如申请发送短信服务的权限需要在功能清单文件中添加如下语句：

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

一个功能（feature）最多只能被一个权限许可保护。如果一个应用程序需要访问一个需要特定权限的功能，它必须在 manifest 元素内使用<uses-permission>元素来声明这一点。这样，当应用程序安装到设备上之后，安装器可以通过检查签署应用程序认证的机构来决定是否授予请求的权限，在某些情况下，会询问用户。如果权限已被授予，那应用程序就能够访问受保护的功能特性。如果没有，访问将失败，但不会给用户任何通知。因此在使用一些系统服务，如拨号、短信、访问互联网、访问 SD Card 时一定要记得添加相应的权限，否则会出现一些难以预料的错误。

应用程序还可以通过权限许可来保护它自己的组件（活动、服务、广播接收器、内容提供者）。它可以利用 Android 已经定义（列在 android.Manifest.permission 里）或其他应用程序已声明的权限许可，或者定义自己的许可。一个新的许可通过<permission>元素声明。例如，一个 Activity 可以用下面的方式保护：

```
<manifest … >
    <permission android:name="com.example.project.DEBIT_ACCT" …/>
    …
    <application …>
        <activity android:name="com.example.project.FreneticActivity" … >
                android:permission="com.example.project.DEBIT_ACCT"
                … >
            …
        </activity>
    </application>
    …
    <uses-permission android:name="com.example.project.DEBIT_ACCT" />
    …
</manifest>
```

注意，在这个例子里，DEBIT_ACCT 许可并非仅仅在<permission>元素中声明，如果该应用程序的其他组件要使用到该组件，那么它同样在<uses-permission>元素里声明。

（6）库 Libraries。

每个应用程序都链接到默认的 Android 库，这个库包含了基础应用程序开发包（实现了基础类，如活动、服务、意图、视图、按钮、应用程序、内容提供者，等等）。

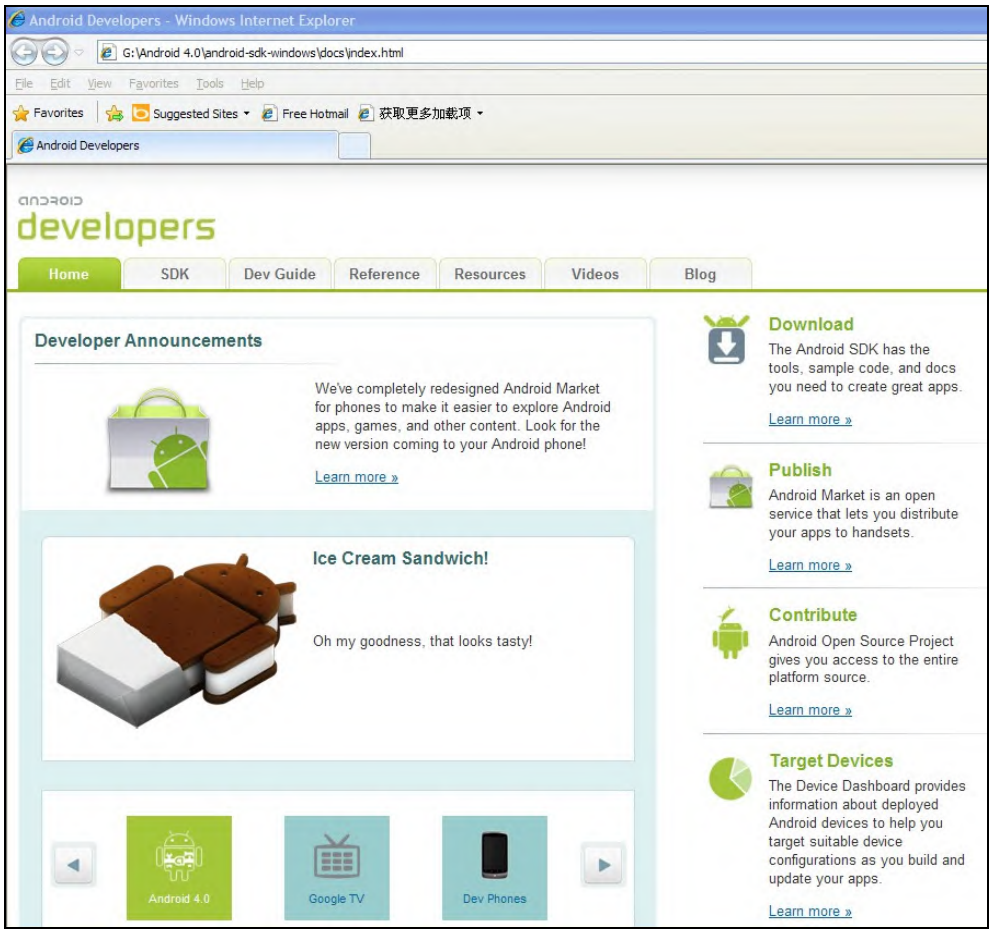
然而，一些包处于它们自己的库中。如果你的应用程序使用了其他开发包中的代码，它必须请求链接到它们。这个 manifest 必须包含一个单独的<uses-library>元素来命名每一个库，如在进行单元测试的时候需要引入其所需要的库。

代码片段如下：

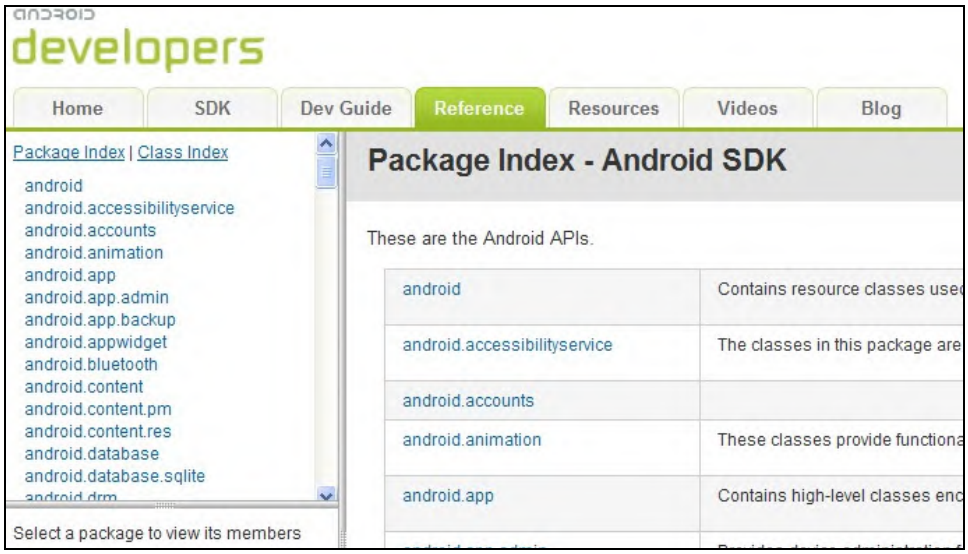
```
<application android:icon="@drawable/icon"
             android:label="@string/app_name">
      <uses-library android:name="android.test.runner" />
  </application>
```

6．如何在文档中查找权限信息

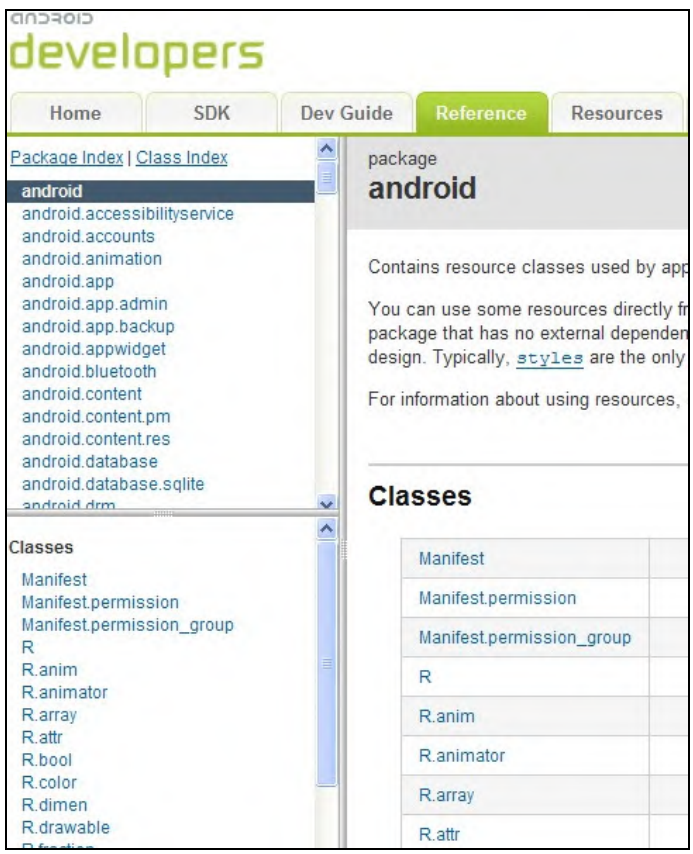
在前面我们介绍了几种常用的权限，但是如何了解 Android 平台中的所有权限信息呢？如查找 CALL_PHONE 这个权限信息，可以按如下步骤操作。在 SDK 解压目录下找到 docs 文档的子目录，如 G：\Android 4.0\android-sdk-windows\docs，运行其中的 index.html 文件，页面如下图所示。



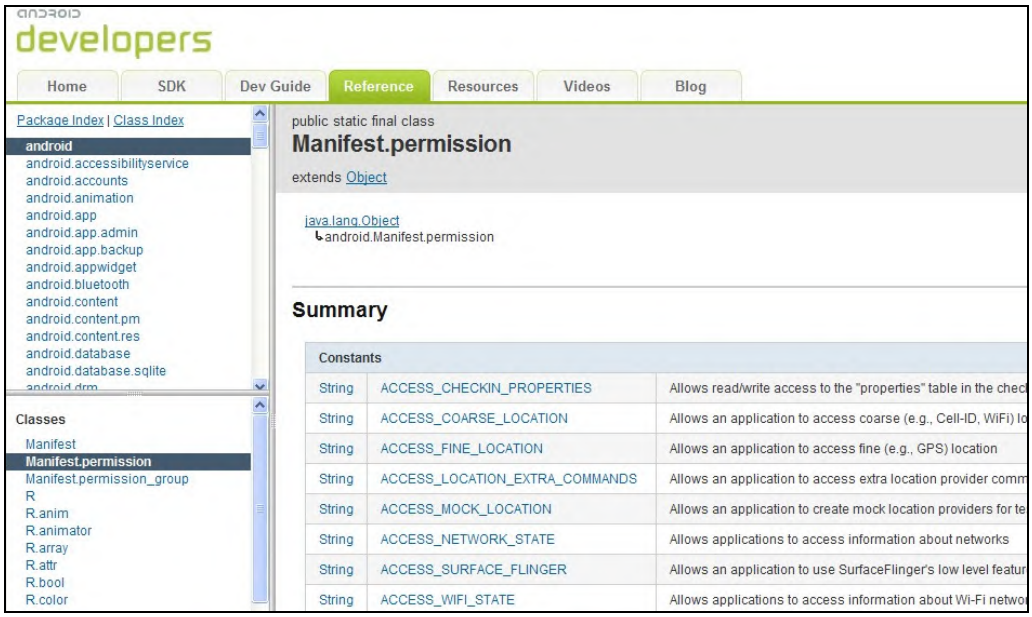
单击“Reference”，如下图所示。



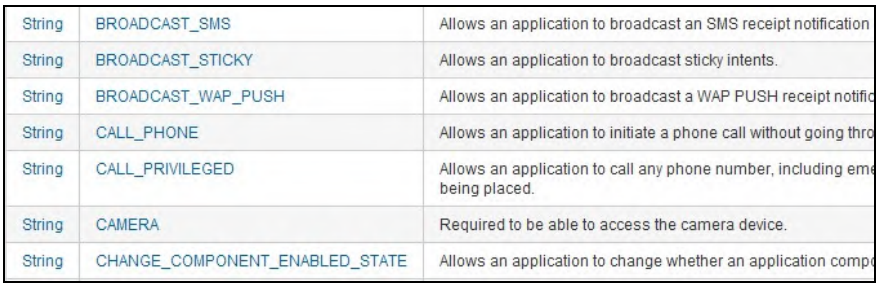
单击方框中的 android 超链接，进入如下界面。



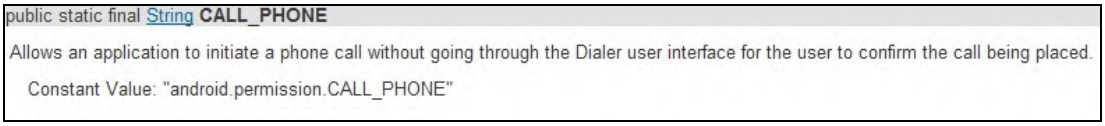
单击方框中的 Mainfest.permission 超链接，进入 Mainfest.permission 类页面，如下图所示。



向下寻找，则会找到 CALL_PHONE 这个权限信息，如下图所示。



单击 CALL_PHONE 超链接，进入下一个界面，如下图所示。

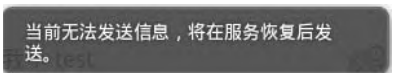


从中得知申请该权限的值为“android.permission.CALL_PHONE”。

以后如果需要申请其他任何权限，都可以通过该方式在 Mainfest.permission 类中找到相关的信息。

1.5 Android 4.0 模拟器无 3G 信号的解决方案

有些时候打开 Android 模拟器时会出现无信号即屏幕左上方显示，而非的情况，并且拨打电话或发短信时，提示“尚未注册网络”错误信息，如下图所示。



该故障的解决方案如下所示。

如果网络状态正常，可以尝试关掉该模拟器，然后重新打开，如果多次打开仍然无效，那么可以根据以下建议进行操作。在没有连接互联网的情况下，进行如下操作。

（1）如果计算机不在一个局域网之中，那么需要将本机 IP 地址设为 DNS 服务器以及默认网关。

右键单击网上邻居，选择“属性”，在网络连接窗口中右键单击“本地连接”，选择“属性”，设置 TCP/IP 属性如下：

- IP 地址：192.168.1.100；
- 子网掩码：255.255.255.0；
- 默认网关：192.168.1.100；
- 首选 DNS 服务器：192.168.1.100。

注意，IP 地址、默认网关、首选 DNS 服务器三项地址必须一致。

（2）计算机在一个局域网内，则可以将默认网关和 DNS 服务器修改为所在局域网的相关地址。一般默认网关的 IP 格式为：*.*.*.1，前三位与本机 IP 一致。如果不成功，可以联系网络管理员获得该地址。

右键单击“网上邻居”，选择“属性”，在网络连接窗口中右键单击“本地连接”，选择“属性”，设置 TCP/IP 属性如下：

- IP 地址：设置成所在局域网的 IP，如 192.168.1.100；
- 子网掩码：设置成所在局域网的掩码，如 255.255.255.0；
- 默认网关：设置成所在局域网的网关，如 192.168.1.1；
- 首选 DNS 服务器：设置成所在局域网的路由器 IP，一般路由器的 IP 格式为*.*.*.1，如 192.168.1.1。通常初学者会出现没有设置网关和 DNS 服务器的错误，这两点一定要注意。

（3）最后一种解决方案是让计算机连上互联网。

第2章

使用C语言编程

2.1 下载并安装C语言交叉编译工具链

1．下载所需软件

因为我们是在 Windows 平台下进行基于 ARM 的 C 语言开发，所以要下载 C 语言交叉编译工具链的 Windows 版本，具体的软件下载地址为 http://www.codesourcery.com/sgpp/lite/arm/portal/package8742/public/arm-none-linux-gnueabi/arm-2011.03-41-arm-none-linux-gnueabi.exe。

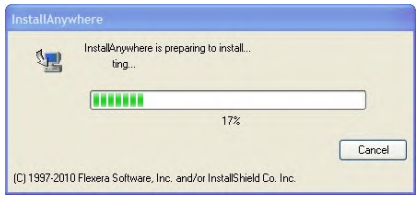
下载时笔者将其放在 G:\NDK\Software 目录中。如果读者想获得其他平台的版本，可以参考以下地址：

http://kristech.pl/dokuwiki/doku.php?id=en:kt-sbc-sam9-1:crosscompiler-sourcery。

2．安装下载的 C 语言交叉编译工具链

（1）双击下载的软件，出现正在准备安装的视图，如下（左）图所示。

（2）当准备环境完毕后会自动进入如下（右）图所示的界面。

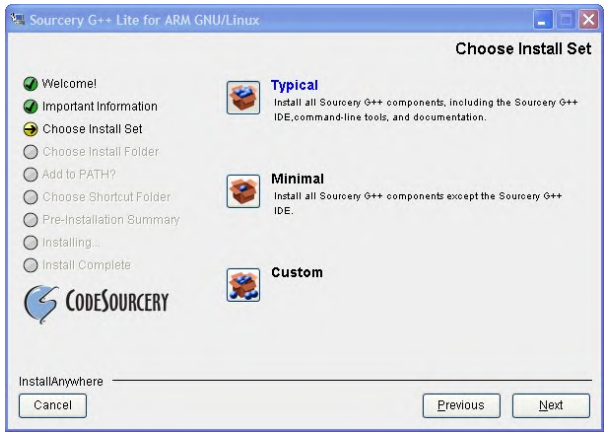


（3）然后软件的安装程序会自动进入安装界面。

单击“Next”按钮进入下一步的安装。

（4）此时选择接收使用许可协议，单击“Next”按钮，进入下一步的安装。

（5）选择默认的“Typical”来进行安装，如下图所示。



单击“Next”按钮进入下一步的安装。

（6）此时安装程序让我们选择程序安装的目录，此时笔者选择安装在 G:\NDK\ProgramFiles 目录中，单击“Next”按钮，进入下一步的安装。

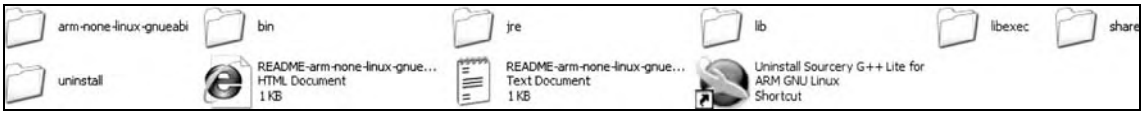
（7）安装程序让我们选择 PATH 路径的设置，选择默认即可，单击“Next”按钮，进入下一步的安装。

（8）此时需要我们确认一下安装的信息，如果觉得有问题可以返回进行修改。

单击“Install”按钮，开始正式的安装。当出现“Done”按钮后，单击“Done”按钮完成 C 语言交叉编译链的安装，此时会弹出帮助的 PDF 文档。

至此，C 语言交叉编译链的安装彻底完成。

打开安装的目录，如下图所示。



打开 bin 文件夹，可以看出里面有很多可执行文件，其实我们使用的核心功能主要有两个：

- 编译——使用的是 arm-none-linux-gnueabi-gcc.exe，每次编译 C 语言文件时需要使用这个命令，这个命令执行的结果会产生.o 文件。
- 链接——使用的是 arm-none-linux-gnueabi-ld.exe，该命令的执行会产生可执行文件。

2.2　第一个 C 语言程序

1．编写 C 语言程序

在 G:\NDK\workspace\Hello 目录下建立一个名为 hello.c 的 C 语言源文件。

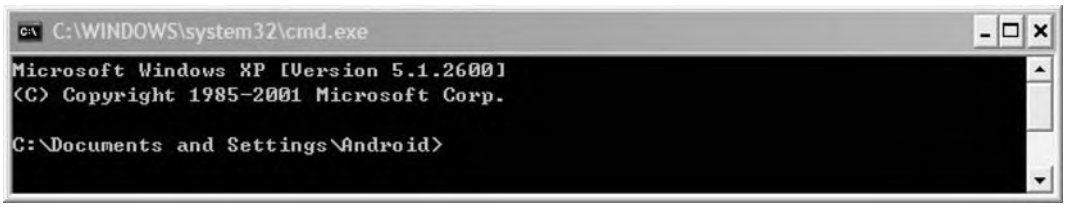
该文件的具体内容如下所示。

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello,Android!\n");
    return 0;
}
```

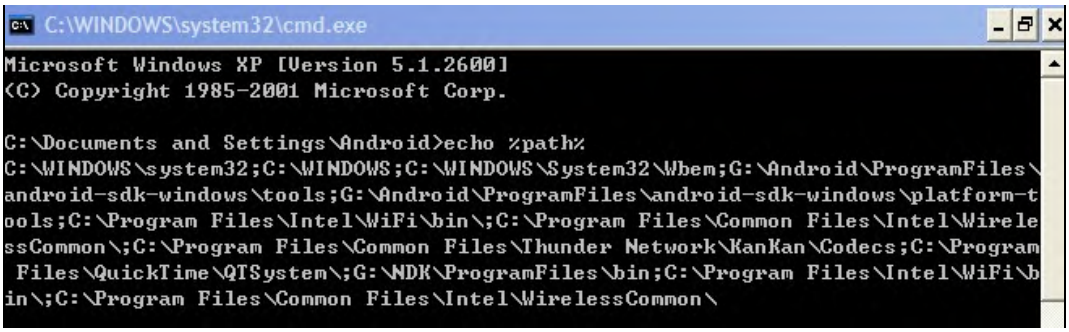
该 C 语言文件的功能就是向控制台输出“Hello，Android!”这样的一个字符串，并换行。

2．编译并链接 C 语言程序

（1）打开 Windows 的 Run 程序，单击“OK”按钮，进入 Windows 的命令窗口，如下图所示。

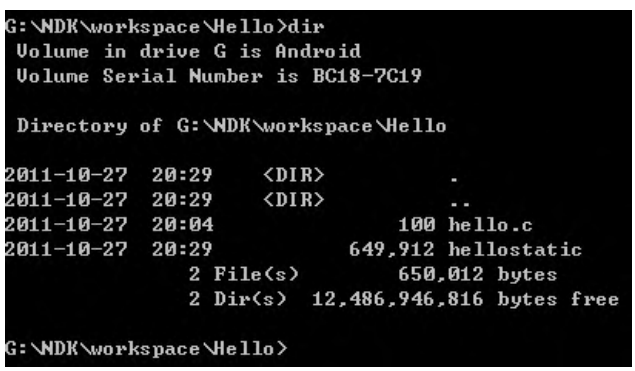


此时，我们使用 echo %path%来查看环境变量，如下图所示。



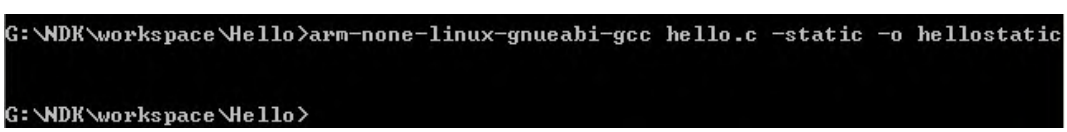
可以看出安装的交叉编译链的 G:\NDK\ProgramFiles\bin 是在系统的环境变量 path 中的，这样我们就可以直接在命令窗口中调用这些命令了。也就是说，我们可以在任何目录路径下执行这些命令。

（2）进入 hello.c 所在的目录，如下图所示。



（3）编译并链接 hello.c。

此时我们使用命令：arm-none-linux-gnueabi-gcc hello.c -static -o hellostatic，如下图所示。



此时我们使用 gcc 命令把 hello.c 通过静态编译的方式编译成目标为 hellostatic 的可执行文件。

我们使用 dir 命令，可以看到 Hello 下产生了目标文件 hellostatic，如下图所示。

```
G:\NDK\workspace\Hello>dir
 Volume in drive G is Android
 Volume Serial Number is BC18-7C19

 Directory of G:\NDK\workspace\Hello

2011-10-27  20:29    <DIR>          .
2011-10-27  20:29    <DIR>          ..
2011-10-27  20:04               100 hello.c
2011-10-27  20:29           649,912 hellostatic
               2 File(s)        650,012 bytes
               2 Dir(s)  12,486,946,816 bytes free

G:\NDK\workspace\Hello>
```

同时打开 G:\NDK\workspace\Hello，发现在文件夹中增加了 hellostatic 目标文件表明编译成功。

非常值得注意的是：

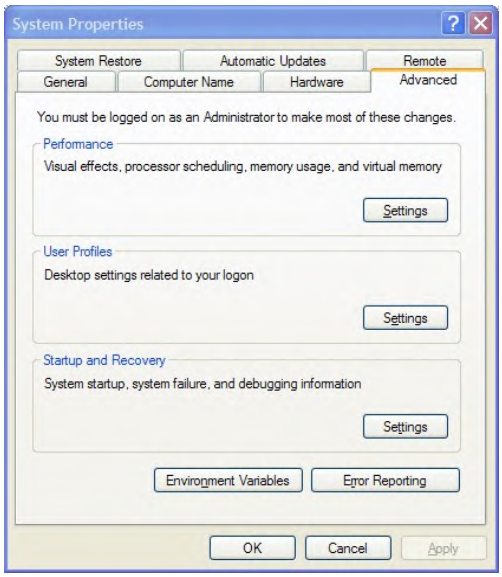
- 这已经是一个可以在 Android 手机上和模拟器上运行的可执行文件。
- 由于 hellostatic 采用了静态链接的原因，所以导致文件的体积非常大，我们只输出一个简短的字符串，最后静态编译链接后的可执行文件大小竟然达到了 635 KB。

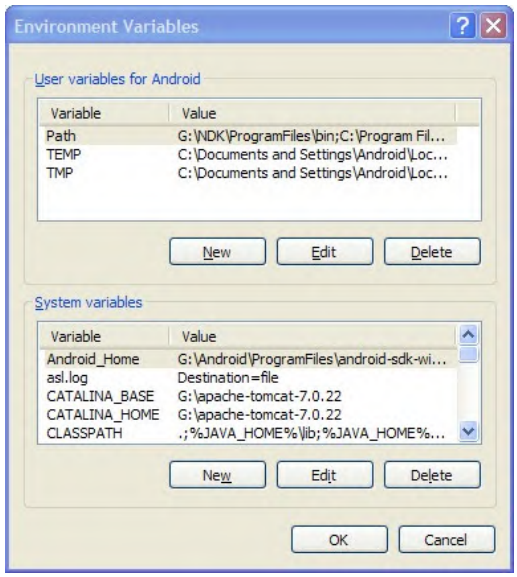
2.3　在 Android 上安装、授权、运行 C 语言程序

1．设置好 Android SDK 的环境变量

右键单击“My Computer”→“Properties”出现如下（左）图所示的界面。

单击“Environment Variables”，出现如下（右）图所示的界面。





可以看出，笔者以前已经设置了 Android_Home，在这里为了方便学习，我们把 Android_Home 修改为新的 SDK，这个 SDK 中只有 Android 4.0 和 Android 2.2 两个平台，在第 1 章中，新安装的 SDK 的目录在 G:\Android 4.0\android-sdk-windows 中，所以，此处就把 Android_ Home 设置为 G:\Android 4.0\android-sdk-windows，此时需要将 Variable Value 值设置为“%Android_Home%\tools;%Android_ Home%\platformtools”。

连续单击“OK”按钮直至完成设置。

2．在 Android 上安装 C 语言程序

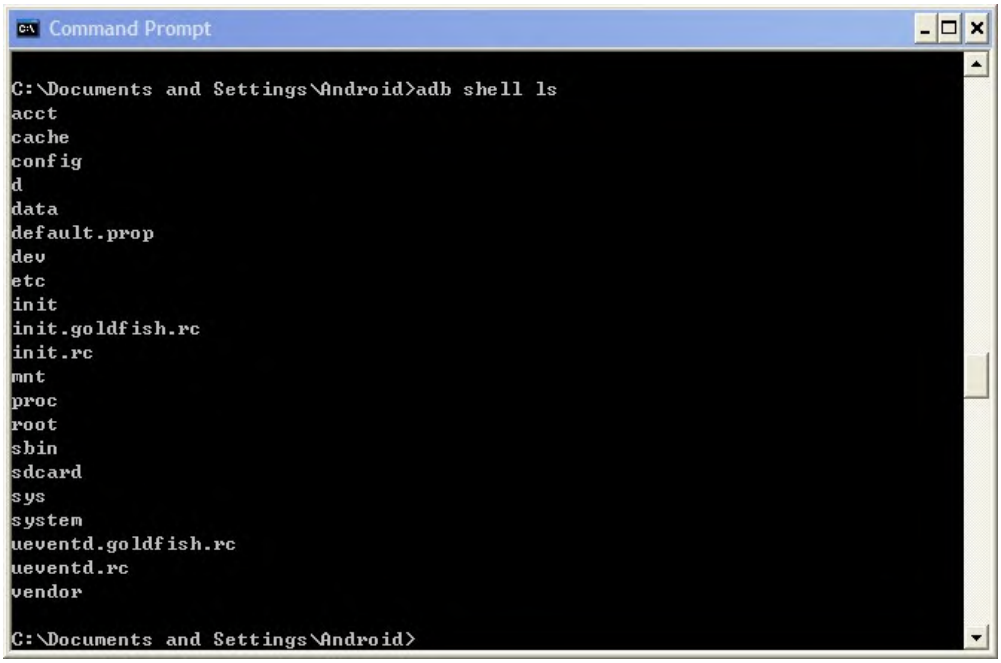
（1）在 Windows 的命令控制台中使用 adb 命令，如下图所示。

```
C:\Documents and Settings\Android>adb
Android Debug Bridge version 1.0.29

 -d                            - directs command to the only connected USB devic
e
                                 returns an error if more than one USB device is
 present.
 -e                            - directs command to the only running emulator.
                                 returns an error if more than one emulator is r
unning.
 -s <serial number>            - directs command to the USB device or emulator w
ith
                                 the given serial number. Overrides ANDROID_SERI
AL
                                 environment variable.
 -p <product name or path>     - simple product name like 'sooner', or
                                 a relative/absolute path to a product
                                 out directory like 'out/target/product/sooner'.

                                 If -p is not specified, the ANDROID_PRODUCT_OUT

                                 environment variable is used, which must
                                 be an absolute path.
 devices                       - list all connected devices
 connect <host>[:<port>]       - connect to a device via TCP/IP
                                 Port 5555 is used by default if no port number
is specified.
 disconnect [<host>[:<port>]]  - disconnect from a TCP/IP device.
                                 Port 5555 is used by default if no port number
is specified.
                                 Using this command with no additional arguments

                                 will disconnect from all connected TCP/IP devic
es.

device commands:
  adb push <local> <remote>    - copy file/dir to device
  adb pull <remote> [<local>]  - copy file/dir from device
  adb sync [ <directory> ]     - copy host->device only if changed
                                 (-l means list but don't copy)
                                 (see 'adb help all')
  adb shell                    - run remote shell interactively
  adb shell <command>          - run remote shell command
  adb emu <command>            - run emulator console command
  adb logcat [ <filter-spec> ] - View device log
  adb forward <local> <remote> - forward socket connections
                                 forward specs are one of:
```

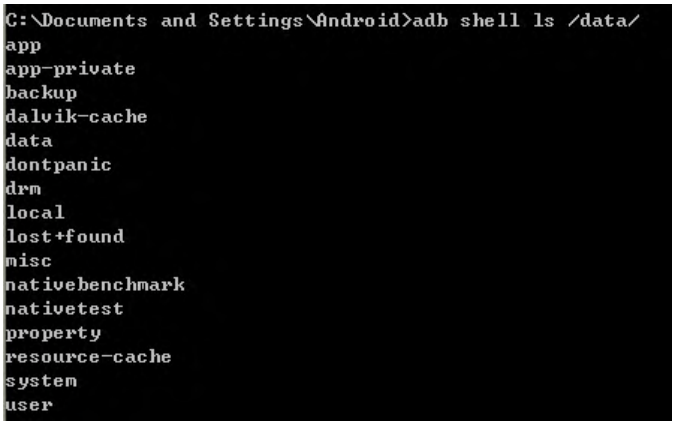
往下拉动命令窗口的时候我们会发现有很多非常有用的 adb 命令。

（2）启动 Android 模拟器。

（3）使用 adb shell ls 命令列出当前模拟器所有的文件，如下图所示。



可以看到文件中有一个 data 的文件夹，我们使用 adb shell ls /data/命令执行，效果如下图所示。



（4）在/data/下建立一个 c 的目录，使用的命令为 adb shell mkdir /data/c。

可以看到，使用 adb shell ls /data/显示出了我们创建的 c 这个文件夹。

（5）在控制台中进入 hellostatic 所在的目录，如下图所示。

```
C:\Documents and Settings\Android>G:

G:\>cd NDK

G:\NDK>cd workspace

G:\NDK\workspace>cd Hello

G:\NDK\workspace\Hello>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

G:\NDK\workspace\Hello>dir
 Volume in drive G is Android
 Volume Serial Number is BC18-7C19

 Directory of G:\NDK\workspace\Hello

2011-10-27  20:29    <DIR>          .
2011-10-27  20:29    <DIR>          ..
2011-10-27  20:04               100 hello.c
2011-10-27  20:29           649,912 hellostatic
               2 File(s)        650,012 bytes
               2 Dir(s)  12,488,245,248 bytes free

G:\NDK\workspace\Hello>
```

（6）使用 adb push hellostatic /data/c 命令安装程序到 c 目录中，如下图所示。

```
G:\NDK\workspace\Hello>adb push hellostatic /data/c
66 KB/s (649912 bytes in 9.484s)

G:\NDK\workspace\Hello>
```

上图表明安装成功。

（7）进入 shell 中，查看安装文件。

使用 adb shell 命令进入 shell，如下（左）图所示。查看文件的内容，如下（右）图所示。

```
G:\NDK\workspace\Hello>adb shell
#
```

```
# cd /data/c
cd /data/c
# ls
ls
hellostatic
#
```

（8）改变 hellostatic 的权限，使我们可以运行该文件，如下（左）图所示。

（9）运行 hellostatic 文件，如下（右）图所示。

```
# chmod 777 ./hellostatic
chmod 777 ./hellostatic
#
```

```
# ./hellostatic
./hellostatic
Hello,Android!
#
```

可以看到，我们成功地执行了 C 程序并打印出了“Hello，Android”字符串。

2.4　采用动态链接的方式生成可执行文件并在 Android 上安装、授权、运行 C 程序

动态链接和静态链接的区别如下。

静态链接：采用静态链接的方式生成 C 可执行文件时，会在链接时把该 C 程序运行所需要的库等一切资源编译进可执行文件中，这样就导致编译链接后的文件非常大，例如，我们前面的 hello.c 的源文件只有 1 KB 的大小，但是通过静态编译链接后却达到了不可思议的 635 KB。

采用静态链接生成的可执行文件在执行时不需要依赖于任何系统的库，一切都是自包含的。

动态链接：在生成可执行文件的时候不包含系统库等常规文件，这就大大缩小了可执行文件的体积。在运行时，可执行文件会动态地从系统获取运行所需要的系统库等文件，从而正常运行。

我们下面采用动态链接的方式运行 C 程序，具体步骤如下所示。

1．使用 arm-none-linux-gnueabi-gcc -c hello.c -o hello.o 命令

编译 hello.c，生成目标文件 hello.o，如下图所示。

```
G:\NDK\workspace\Hello>arm-none-linux-gnueabi-gcc -c hello.c -o hello.o

G:\NDK\workspace\Hello>dir
 Volume in drive G is Android
 Volume Serial Number is BC18-7C19

 Directory of G:\NDK\workspace\Hello

2011-10-28  07:56    <DIR>          .
2011-10-28  07:56    <DIR>          ..
2011-10-27  20:04               100 hello.c
2011-10-28  07:56             1,340 hello.o
2011-10-27  20:29           649,912 hellostatic
               3 File(s)        651,352 bytes
               2 Dir(s)  12,488,237,056 bytes free

G:\NDK\workspace\Hello>
```

由上图可以看出，当我们使用 arm-none-linux-gnueabi-gcc -c hello.c -o hello.o 命令编译 hello.c 后生成了目标文件 hello.o，通过 dir 命令可以看出 Hello 目录中新增了 hello.o 文件，也可以通过 Windows 的文件夹进行查看。

2．链接

因为我们使用的是动态链接，所以在我们要进行链接时就必须利用 Android 下的 Linux 的 Libraries，此类库位于/system/lib/目录下。

此时我们通过 Windows 的命令窗口进入打开的模拟器的根目录，如下（左）图所示。

我们发现根目录下有一个 system 的目录。

进入 system 文件夹，并列出其文件，如下（右）图所示。

```
G:\NDK\workspace\Hello>adb shell
# ls
ls
acct
cache
config
d
data
default.prop
dev
etc
init
init.goldfish.rc
init.rc
mnt
proc
root
sbin
sdcard
sys
system
ueventd.goldfish.rc
ueventd.rc
vendor
```

```
# cd system
cd system
# ls
ls
app
bin
build.prop
etc
fonts
framework
lib
lost+found
media
tts
usr
xbin
#
```

很明显，system 目录下有一个名称为 lib 的目录，进入 lib 目录，并将其内容列出，如下图所示。

```
# cd lib
cd lib
# ls
ls
egl
hw
invoke_mock_media_player.so
libEGL.so
libETC1.so
libFFTEm.so
libGLESv1_CM.so
libGLESv1_enc.so
libGLESv2.so
libGLESv2_dbg.so
libGLESv2_enc.so
libOpenMAXAL.so
libOpenSLES.so
libOpenglSystemCommon.so
libRS.so
libSR_AudioIn.so
libWnnEngDic.so
libWnnJpnDic.so
lib_renderControl_enc.so
libandroid.so
libandroid_runtime.so
libandroid_servers.so
libaudioeffect_jni.so
libaudioflinger.so
libbcc.so
libbcc.so.sha1
libbcinfo.so
libbinder.so
libc.so
libc_malloc_debug_leak.so
libc_malloc_debug_qemu.so
libcamera_client.so
libcameraservice.so
libchromium_net.so
libclcore.bc
libcrypto.so
libctest.so
libcutils.so
libdefcontainer_jni.so
libdiskconfig.so
libdl.so
libdrm1.so
libdrm1_jni.so
libdrmframework.so
libdvm.so
libeffects.so
libemoji.so
libexif.so
```

```
libexif.so
libexpat.so
libext4_utils.so
libfilterfw.so
libfilterpack_imageproc.so
libgabi++.so
libgui.so
libhardware.so
libhardware_legacy.so
libharfbuzz.so
libicui18n.so
libicuuc.so
libinput.so
libjni_latinime.so
libjni_mosaic.so
libjnigraphics.so
libjpeg.so
liblog.so
libm.so
libmedia.so
libmedia_jni.so
libmediaplayerservice.so
libmtp.so
libnativehelper.so
libnetutils.so
libnfc_ndef.so
libpagemap.so
libpixelflinger.so
libpower.so
libpowermanager.so
libreference-ril.so
libril.so
librs_jni.so
librtp_jni.so
libsensorservice.so
libskia.so
libsonivox.so
libsoundpool.so
libspeexresampler.so
libsqlite.so
libsqlite_jni.so
libsrec_jni.so
libssl.so
libstagefright.so
libstagefright_amrnb_common.so
libstagefright_avc_common.so
libstagefright_enc_common.so
libstagefright_foundation.so
libstagefright_omx.so
libstagefright_soft_aacdec.so
libstagefright_soft_amrdec.so
libstagefright_soft_g711dec.so
```

可以看出，lib 下有很多以.so 结尾的文件，同时也有 hw 等文件夹。

为了能够正确地链接程序，我们需要把 Android 系统库下的文件复制到 Windows 平台中，即把/system/lib/目录下的所有文件复制到 Windows 的一个文件夹中，复制的工作是通过 adb pull 来完成的。

首先我们需要在 Windows 上建立一个文件夹，笔者是在 G:\NDK\目录下建立了一个名为 Android_Library 的文件下用来存放 Android 的系统库文件。

准备工作做好后，通过下面一行命令开始复制工作：adb pull /system/lib G:\NDK\Android_Library。

通过复制过程可以看出，我们从 Android 中的/system/lib/下一共复制了 140 个文件。

需要说明的是，读者也可以根据自己的需要逐个进行导出。

接下来的工作就是要进行动态链接了，具体的命令如下所示。

```
    G:\NDK\workspace\Hello>arm-none-linux-gnueabi-ld-entry=main-dynamic-linker/
system/bin/linker-nostdlib-rpath G:\NDK\Android_Library-rpath-link G:\NDK\Android_
Library-L G:\NDK\An- droid_Library-l android_runtime-l c-o hellodynamic hello.o
```

其中参数的具体含义如下。

- arm-none-linux-gnueabi-ld——C 语言链接器命令；
- -entry=main——表明程序的执行入口点是 main 函数；
- -dynamic-linker/system/bin/linker——告诉应用程序在哪里能够找到动态链接器程序，/system/bin/linker 路径是在 Android 模拟器中的，在开发环境中是不存在的；
- -nostdlib——告诉链接器在链接的过程中不要包含标准的 C 库；
- -rpath G:\NDK\Android_Library——告诉可执行文件在运行时从什么地方可以找到 libraries，其作用和环境变量是一致的，因为我们是在 Windows 环境下运行该程序的，所示 libraries 的路径为 G:\NDK\Android_Library；
- -rpath-link G:\NDK\Android_Library——当进行链接时告诉链接器从哪里能够找到 libraries 文件；
- -L G:\NDK\Android_Library——告诉链接器在哪里能够找到 libraries；
- -l android_runtime——告诉链接器可以在库文件 libandroid_runtime.so 中找到所需要的程序，打开 Android 的库文件，可以找到 libandroid_runtime.so；
- -l c——告诉链接器可以在库文件 libc.so 中找需要的程序，打开 Android 的库文件，可以找到 libc.so；
- -o hellodynamic hello.o——表明目标文件是 hello.o，链接后生成的可执行文件为 hellodynamic。

回车后，运行上述的链接命令，链接之后，使用 dir 命令，如下图所示。

```
G:\NDK\workspace\Hello>arm-none-linux-gnueabi-ld -entry=main -dynamic-linker /sy
stem/bin/linker -nostdlib -rpath G:\NDK\Android_Library -rpath-link G:\NDK\Andro
id_Library -L G:\NDK\Android_Library -l android_runtime -l c -o hellodynamic hel
lo.o

G:\NDK\workspace\Hello>dir
 Volume in drive G is Android
 Volume Serial Number is BC18-7C19

 Directory of G:\NDK\workspace\Hello

2011-10-28  17:50    <DIR>          .
2011-10-28  17:50    <DIR>          ..
2011-10-27  20:04               100 hello.c
2011-10-28  07:56             1,340 hello.o
2011-10-28  17:50             2,839 hellodynamic
2011-10-27  20:29           649,912 hellostatic
               4 File(s)        654,191 bytes
               2 Dir(s)  12,452,413,440 bytes free

G:\NDK\workspace\Hello>
```

可以看出链接后的可执行文件 hellodynamic 出现在了列表中。此时的 G:\NDK\workspace\Hello 有 hello.c、hello.o、hellodynamic、hellostatic 文件。

至此，对 hello.o 文件的的链接完毕。

3．在 Android 上安装、授权、运行 C 程序

首先安装经过动态链接后的可执行文件 hellodynamic，具体所示的命令如下所示。

```
    "adb push hellodynamic /data/c"
```

这条指令的意思是将 hellodynamic 文件安装到 Android 模拟器的/data/c 目录下，具体效果如下图所示。

```
G:\NDK\workspace\Hello>adb push hellodynamic /data/c
3 KB/s (2839 bytes in 0.843s)
```

此时，进入 Android 的/data/c 目录下查看我们安装的可执行文件，如下图所示。

```
G:\NDK\workspace\Hello>adb shell
# cd /data/c
cd /data/c
# ls
ls
hellodynamic
hellostatic
# ls -l
ls -l
-rw-rw-rw- root     root         2839 2011-10-28 17:50 hellodynamic
-rwxrwxrwx root     root       649912 2011-10-27 20:29 hellostatic
#
```

从上面的命令可以看出我们在 Android 上成功地安装了 hellodynamic，同时看到 hellodynamic 的体积只有 2 KB 多一点，而 hellostatic 却高达 634 KB。

接下来对 hellodynamic 进行授权，使我们能够具有执行该文件的权限，具体命令如下所示。

```
    "adb shell chmod 777 /data/c/hellodynamic"
```

控制台的运行效果如下所示。

```
G:\NDK\workspace\Hello>adb shell chmod 777 /data/c/hellodynamic

G:\NDK\workspace\Hello>
```

授权后就可以运行 hellodynamic 了，具体指令如下所示。

```
"adb shell /data/c/hellodynamic"
```

运行效果如下图所示。

```
G:\NDK\workspace\Hello>adb shell /data/c/hellodynamic
Hello,Android!
[1]   Segmentation fault      /data/c/hellodyn...

G:\NDK\workspace\Hello>
```

我们成功地运行了 hellodynamic 程序。

不过，读者应该注意到，在执行 hellodynamic 时，在成功打印出“Hello，Android”之后，下面提示了一个“Segmentation fault”的错误，当然这个错误对程序并没有什么影响。关于该错误的解决方法，会在下一节细说。

2.5　解决采用动态链接方式生成的可执行文件执行时的“Segmentation fault”问题

首先我们再来看一下 hello.c 的源代码，如下所示。

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello,Android!\n");
    return 0;
}
```

可以看出我们在源代码中要返回 0，但是把 0 返回给谁呢？没有返回给任何调用者，因为没有任何调用者调用 main 函数，所以就出现了“Segmentation fault”问题。

要解决这个问题，就不要有返回值，修改 hello.c 源代码的内容，如下所示。

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello,Android!\n");
    exit(0);
    //return 0;
}
```

我们在新的 hello.c 中去掉了 return 0 这句代码，使用 exit(0)来结束程序的运行。

具体过程如下。

（1）重新编译 hello.c，使用 arm-none-linux-gnueabi-gcc -c hello.c -o hello.o，如下图所示。

```
G:\NDK\workspace\Hello>arm-none-linux-gnueabi-gcc -c hello.c -o hello.o
hello.c: In function 'main':
hello.c:5:2: warning: incompatible implicit declaration of built-in function 'ex
it'

G:\NDK\workspace\Hello>
```

其中出现的警告并不影响目标文件的生成。

（2）动态链接 hello.o，使用的命令为 arm-none-linux-gnueabi-ld -entry=main -dynamic-linker /sy stem/bin/linker -nostdlib -rpath G:\NDK\Android_Library -rpath-link G:\NDK\Andro id_Library -L G:\NDK\Android_Library -l android_runtime -l c -o hellodynamic hello.o，运行效果如下图所示。

```
G:\NDK\workspace\Hello>arm-none-linux-gnueabi-ld -entry=main -dynamic-linker /sy
stem/bin/linker -nostdlib -rpath G:\NDK\Android_Library -rpath-link G:\NDK\Andro
id_Library -L G:\NDK\Android_Library -l android_runtime -l c -o hellodynamic hel
lo.o

G:\NDK\workspace\Hello>
```

（3）安装 hellodynamic 文件到 Android 模拟器上，效果如下图所示。

```
G:\NDK\workspace\Hello>adb push hellodynamic /data/c
10 KB/s (2900 bytes in 0.281s)

G:\NDK\workspace\Hello>
```

（4）进入 Android 中，将 hellodynamic 的权限加入可执行权限，如下图所示。

```
G:\NDK\workspace\Hello>adb shell
# cd data
cd data
# cd c
cd c
# ls -l
ls -l
-rw-rw-rw- root     root         2900 2011-10-28 20:04 hellodynamic
-rwxrwxrwx root     root       649912 2011-10-27 20:29 hellostatic
# chmod 777 hellodynamic
chmod 777 hellodynamic
# ls -l
ls -l
-rwxrwxrwx root     root         2900 2011-10-28 20:04 hellodynamic
-rwxrwxrwx root     root       649912 2011-10-27 20:29 hellostatic
```

（5）执行 Android 中的 hellodynamic 可执行程序，使用的命令是 adb shell /data/c/hellodynamic，如下图所示。

```
G:\NDK\workspace\Hello>adb shell /data/c/hellodynamic
Hello,Android!

G:\NDK\workspace\Hello>
```

程序成功运行。

至此，我们成功地解决了采用动态链接方式生成的可执行文件执行时的“Segmentation fault”问题。

第3章

搭建Android NDK开发环境并开发第一个Android NDK程序

3.1 下载 Windows 下开发 Android NDK 所需的软件

1．下载并安装版本不低于 1.5 的 Android SDK

笔者在第 1 章中已经非常详细地讲解了如何安装 Android 4.0 的开发环境，读者可以返回第 1 章进行参考。

2．下载并安装 Android NDK 开发包

我们使用最新的“Android NDK，r6b”，具体过程如下所示。

进入 NDK 开发者的官方网站 http://developer.android.com/sdk/ndk/index.html。

选择 android-ndk-r6b-windows.zip，单击即可下载。笔者下载后的 Android NDK 开发包存放于 G:\Android 4.0\Software 目录下。

3．下载 Cygwin

为何要下载 Cygwin？原因是 NDK 的编译、链接命令需要在 Linux 环境下运行，而现在是在 Windows 上开发 Android NDK 程序，所以就需要 Cygwin 这个工具来模拟 Linux 环境，需要说明的是我们需要的 Cygwin 版本是不能够低于 1.7 的。

进入 Cygwin 的官方网站 http://www.cygwin.com/。

单击 http://cygwin.com/install.html 首页左上方的“Install Cygwin”即可进入 Sygwin 的下载、安装界面。

单击 setup.exe 超链接，即可下载 Cygwin，笔者下载后存放在了 G:\Android 4.0\Software 目录下，具体文件与安装包大小如下图所示。

至此，Windows 下开发 Android NDK 所需要的软件准备完毕。

3.2 安装 Windows 下 Android NDK 开发环境

1．安装 Android SDK 开发环境

这一点在第 1 章已经进行了非常详细的讲解，读者如果还没有安装该环境的话，可以参考

第 1 章中的安装过程。

2．安装 Android NDK

安装很简单，只需要把下载的 android-ndk-r6b-windows.zip 解压缩到适当的目录即可，笔者将其解压到了 G:\NDK 目录下。

单击“android-ndk-r6b”，进入文件夹，此时我们看到有一个名为 toolchains 的文件夹，这是 Android NDK 开发的编译工具链，和第 2 章中使用的编译工具链是一致的，双击该文件夹进入，在这里我们看到了 arm-linux-androideabi-4.4.3，这正是编译工具链要使用的包，打开 arm-linux-androideabi-4.4.3 → prebuilt → windows → bin，可以看出，里面出现了 arm-linux-androideabi-gcc.exe 编译 C 源文件的命令以及 arm- linux-androideabi-ld.exe 链接 C 目标文件以生产 C 可执行文件的命令。

3．安装 Cywin

（1）双击 Cywin 的 setup.exe，单击“Next”按钮进入安装的下一步。

（2）我们使用默认的选项，从 Internet 上下载文件，并自动进行安装，单击“Next”按钮进入下一步。笔者使用的默认的安装目录（G:Android 4.0\cygwin），单击“Next”按钮进入下一步。

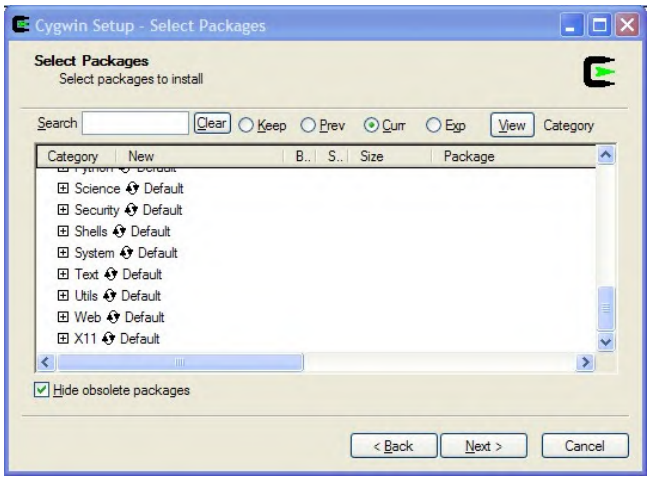
（3）我们使用程序默认选择的目录，单击“Next”按钮进入下一步。

（4）此时要求我们选择网络连接的方式，因为笔者的计算机是直接连上 Internet 的，所以就使用默认的 Direct Connection，单击“Next”按钮进入下一步。

（5）此时让我们选择从哪个镜像站点下载 Cygwin 安装程序，笔者选择的是 http://mirrors.163.com，当然读者也可以根据需要选择其他的站点。

（6）单击“Next”按钮进入下一步的安装，在安装的过程中会弹出如下图所示的对话框。

这个窗口不是运行错误，而是询问我们是否是第一次安装 Cygwin，如果是就单击“OK”按钮继续安装，如果不是则进行 Cygwin 的升级，而不是安装。单击“OK”按钮，此时安装程序弹出了一个大窗口，该窗口是询问我们要安装哪些组件，我们使用默认的选择即可，此时手动缩小窗口，如下图所示。



（7）单击“Next”按钮进入下一步，此时正在下载要安装的文件，我们需要等待下载完毕。

下载完毕后，会自动进入安装过程。此时我们等待安装的完成。

在笔者的安装过程中，因为在笔者的计算机上安装了 360 木马防火墙的原因，所以弹出了如下图所示的窗口，建议读者在安过程中可先关掉防火墙软件。

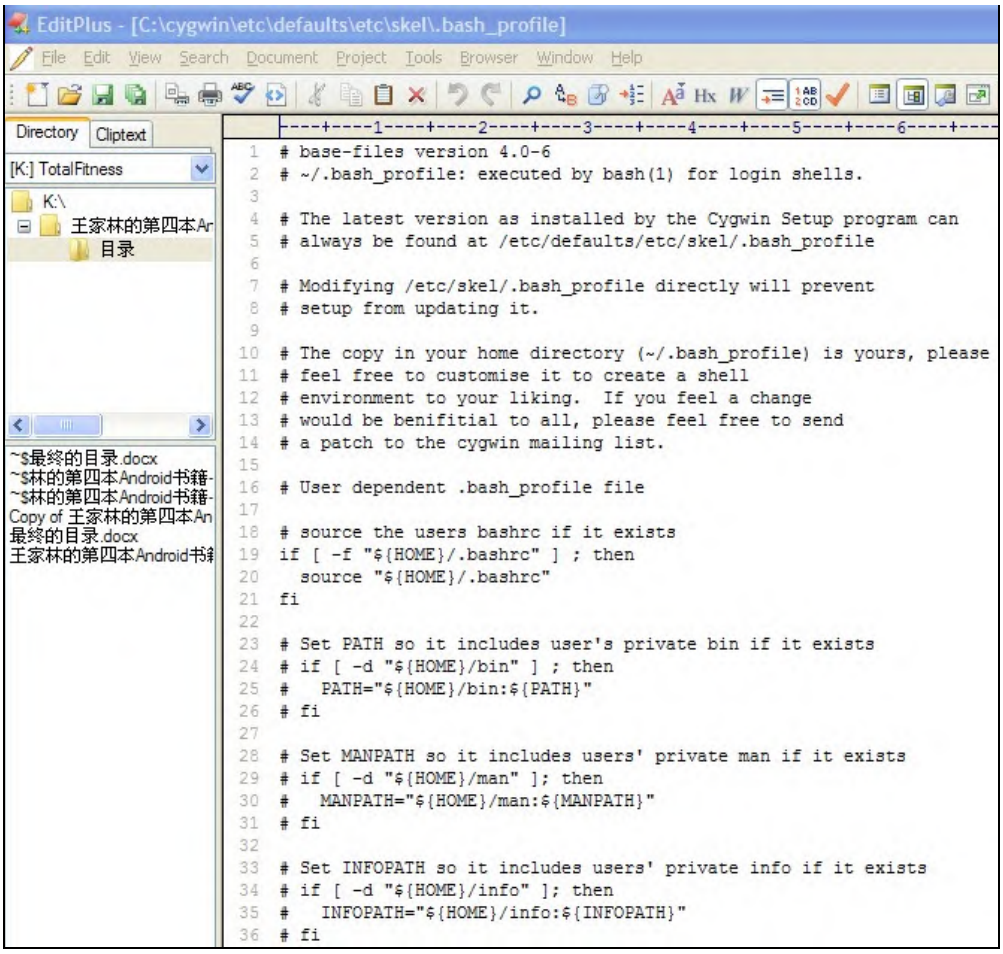
（8）单击“添加信任”即可，此时程序安装完成，使用默认选项，单击“Finish”按钮完成安装。

至此，Android NDK 开发环境安装完毕。

3.3 配置 Cygwin

为了更加方便地进行编译链接 C 程序，我们对 Cygwin 进行适当的设置，进入找到 Cygwin 默认配置文件.bash_profile，在笔者的计算机上，该文件的具体路径为 C:\cygwin\etc\defaults\etc\skel。

使用 EditPlus 打开.bash_profile，如下图所示。



我们在该文件的结尾处加入 NDK 的安装目录的参数设置，具体内容如下所示。

```
NDK=/cygdrive/G:\NDK\android-ndk-r6b
export NDK
```

修改后保存好该文件即可。

接下来启动 Cygwin，选择“start”→“All Programs”→“Cygwin”→“CygwinBashShell”即可启动 Cygwin。

在 shell 中输入 cd $NDK 命令进入 NDK 的安装目录，我们发现此时 cd $NDK 是无效的，因为并没有进入 NDK 安装目录，可以使用 who 命令来查看当前命令行所在的目录。

出现此问题的原因是我们只是修改了 C:\cygwin\etc\defaults\etc\skel 下的.bash_profile 文件，但并没有把该文件复制到用户目录下，笔者的计算机上的用户名称为 Android，所以用户目录的具体路径就是 C:\cygwin\home\Android，此时我们把 C:\cygwin\etc\defaults\etc\skel 下的.bash_profile 复制到 C:\cygwin\home\Android 下，重新开启一个 Cygwin Bash Shell 终端。

此时输入 cd $NDK 命令并按下回车执行该命令。

可以看到，其实我们已经进入了该目录，使用 who 命令可以查看当前所在的完整路径。

至此，设置完毕。

3.4 开发第一个 Android NDK 程序

下面通过一个 Android NDK 程序来初步体验 Android NDK 的开发过程，具体步骤如下所示。

1．创建工程

建立一个工程名称为 HelloAndroidNDK 的 Android 工程。单击“Next”按钮进入下一步。

我们使用默认的 Android 4.0 平台，单击“Next”按钮进入下一步，此时将包的名称命名为 com.wangjialin.ndk.hello，单击“Finish”按钮完成工程的创建。

2．加载本地的 C 库文件，同时声明本地的 C 方法

在此，创建一个 HelloAndroidNDK 的 Java 文件，此时编写 HelloAndroidNDK 的内容如下所示。

```
package com.wangjialin.ndk.hello;

public class HelloAndroidNDK {
    //静态代码块在类加载时会执行，这个时候就会加载本地的 C 库文件
    static{
        //加载本地的 C 库文件
        System.loadLibrary("helloAndroidNDK");
    }

    //声明加载的 C 库中的本地方法
    public native String sayHelloToNDK();
}
```

3．编译 Android 文件

因为使用的是 Eclipse 的默认设置，所以当保存文件时，Eclipse 会自动帮助我们完成 Android 的编译工作，此时会在 bin\classes\com\wangjialin\ndk/hello 目录下找到 Hello Android NDK 被编译后的 class 文件，名称为 HelloAndroidNDK.class，不过读者需要注意的是默认情况下无法通过 Eclipse 找到该文件，需要从 Windows 的文件夹下进入，Hello AndroidNDK.class 的具体 Windows 路径为 G:\Android 4.0\workspace\HelloAndroidNDK\bin\ classes\com\wangjialin\ndk\hello。

4．使用 javah 工具产生 C 语言的.h 头文件

（1）在 Eclipse 的 HelloAndroidNDK 这个工程的根目录下建立一个名称为 jni 的文件夹，将 Folder name 设置为 jni，单击“Finish”按钮完成文件夹的创建。

（2）打开 Windows 的命令窗口，在 G:\Android 4.0\workspace\Hello AndroioNDK 下输入 jni 可进入刚才新建的 jni 的目录中。

（3）使用 javah 工具产生 HelloAndroidNDK.class 的头文件，具体操作命令如下图所示。

```
G:\Android 4.0\workspace\HelloAndroidNDK\jni>javah -classpath ../bin/classes com
.wangjialin.ndk.hello.HelloAndroidNDK
'javah' is not recognized as an internal or external command,
operable program or batch file.

G:\Android 4.0\workspace\HelloAndroidNDK\jni>
```

此时命令提示符提示我们无法识别 javah 命令，这是由于该命令没有加入到 Windows 的 path 路径中的缘故，该文件的路径位于笔者计算机上的 G:\Android\ProgramFiles\Java\bin，将 Variable Value 的值设为“QuickTime\QTSystem\;%JAVA_HOME%bin”；把该路径加入 Windows 的 Path 环境变量中。

单击“OK”按钮确认该设置，也可以把 JAVA_HOME 设置为 Java 的安装目录 G:\Android\ProgramFiles\Java。

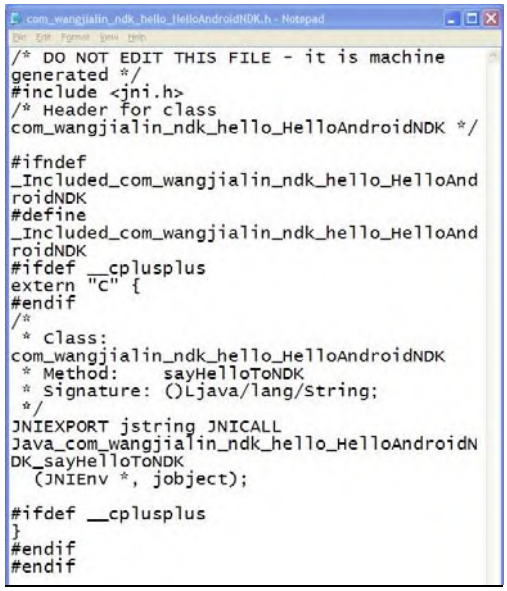
此时重新启动命令窗口，再次进入 G:\Android 4.0\workspace\HelloAndroidNDK\jni，执行 javah -classpath ../bin/classes com.wangjialin.ndk.hello.HelloAndroidNDK 命令，如下图所示。

```
G:\Android 4.0\workspace\HelloAndroidNDK\jni>javah -classpath ../bin/classes com
.wangjialin.ndk.hello.HelloAndroidNDK

G:\Android 4.0\workspace\HelloAndroidNDK\jni>
```

此时我们成功地执行了 javah 命令并生成了头文件，打开 jni 的文件夹，可以看到自动生成的头文件 com_wangjialin_ndk_hello_HelloAndroidNDK.h。

此时刷新 Eclipse 的工程 HelloAndroidNDK，jni 的目录中出现了头文件。

打开 com_wangjialin_ndk_hello_HelloAndroidNDK.h，其具体内容如下所示。



5．编写 C 语言程序

Java 开发者产生 com_wangjialin_ndk_hello_HelloAndroidNDK.h 文件后，就可以把头文件传递给 C 程序开发者，让 C 程序开发者根据 com_wangjialin_ndk_hello_HelloAndroidNDK.h 来

开发 C 程序。此时我们将 HelloAndroidNDK 整个工程复制到 Android NDK 的 samples 目录下。

此时我们在复制后的工程文件的 jni 文件夹中编写 com_wangjialin_ndk_hello_HelloAndroidNDK.c 文件，实现 com_wangjialin_ndk_hello_HelloAndroidNDK.h 的功能，com_wangjialin_ndk_hello_HelloAndroidNDK.c 的内容如下所示。

```
#include "com_wangjialin_ndk_hello_HelloAndroidNDK.h"
JNIEXPORT jstring JNICALL Java_com_wangjialin_ndk_hello_HelloAndroidNDK_
sayHelloToNDK
  (JNIEnv *env, jobject thiz)
{

     return (*env)->NewStringUTF(env, "Hello,Android NDK,this is
     WangJialin!");
}
```

从 com_wangjialin_ndk_hello_HelloAndroidNDK.c 的内容可以看出，我们只是导入了头文件 com_wangjialin_ndk_hello_HelloAndroidNDK.h，并实现了头文件中声明的方法，该方法的声明为：

```
JNIEXPORT jstring JNICALL Java_com_wangjialin_ndk_hello_HelloAndroidNDK_
sayHelloToNDK
  (JNIEnv *, jobject);
```

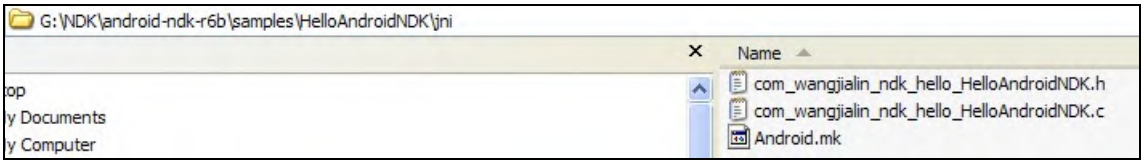
上述过程只是简单地返回了一个“Hello，Android NDK，this is WangJialin!”的字符串，如下所示。

```
  return (*env)->NewStringUTF(env, "Hello,Android NDK,this is
  WangJialin!");
```

6．根据开发的项目修改 Android.mk 文件

从 samples 示例包中的 hello-jni 的 jni 文件夹中复制一份 Android.mk 文件，如下图所示。

把其中的 Android.mk 复制到 samples 下的 jni 文件中，如下图所示。

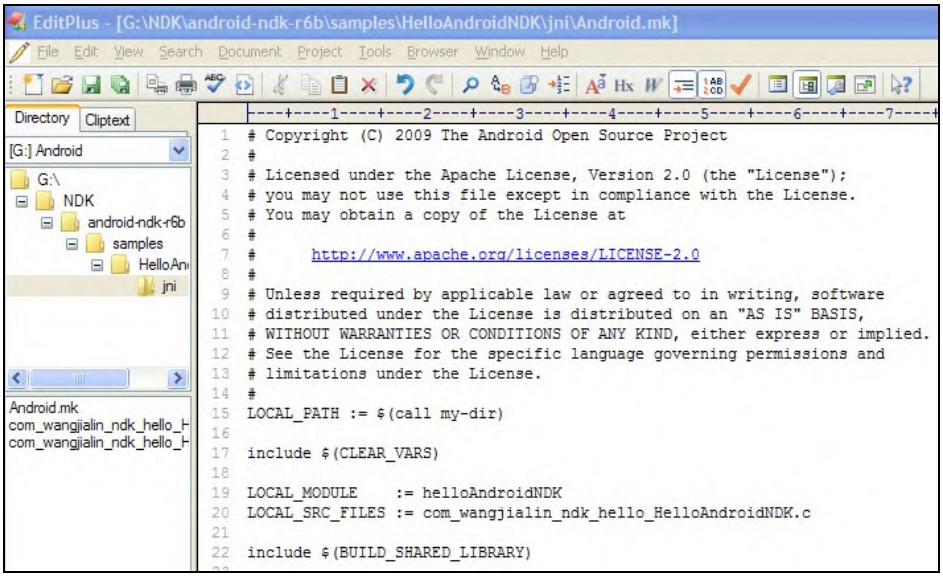


打开复制后的 Android.mk 文件，其初始内容如下所示。

```
# 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)
```

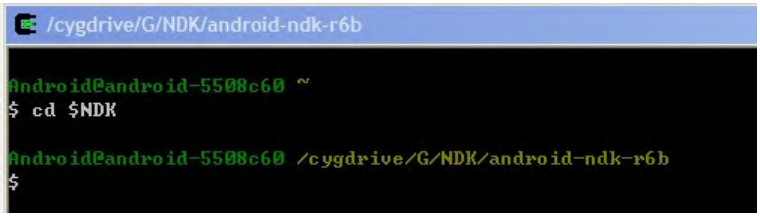
修改 Android.mk 文件，以适应项目编译、链接的需要，修改后的内容如下图所示。



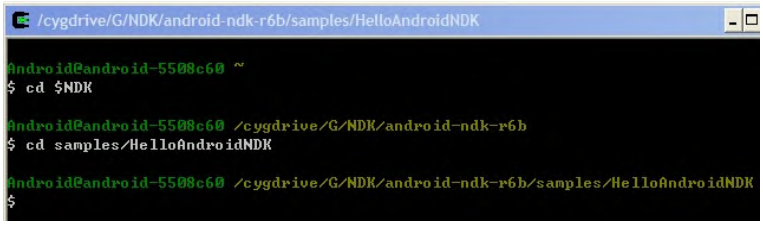
7．编译并链接 C 程序

（1）启动 Cygwin。

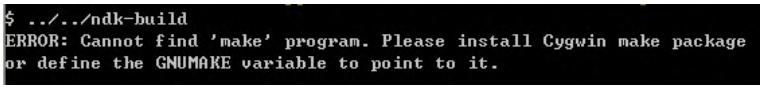
（2）输入 cd $NDK 命令进入 Android NDK 目录，如下图所示。



（3）进入 NDK 目录下的/samples/HelloAndroidNDK 目录，如下图所示。

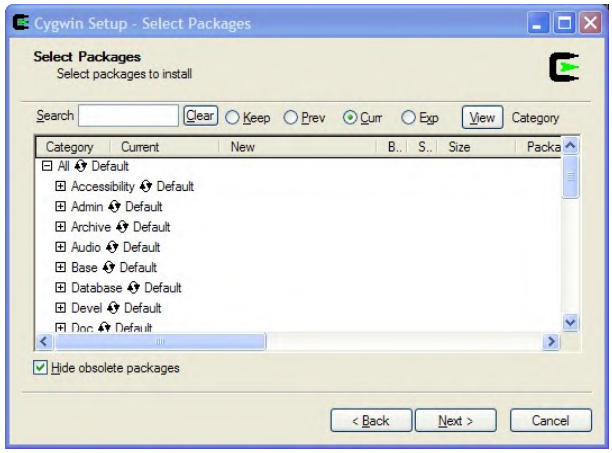


（4）执行 NDK 的编译、链接命令 ndk-build，如下图所示。

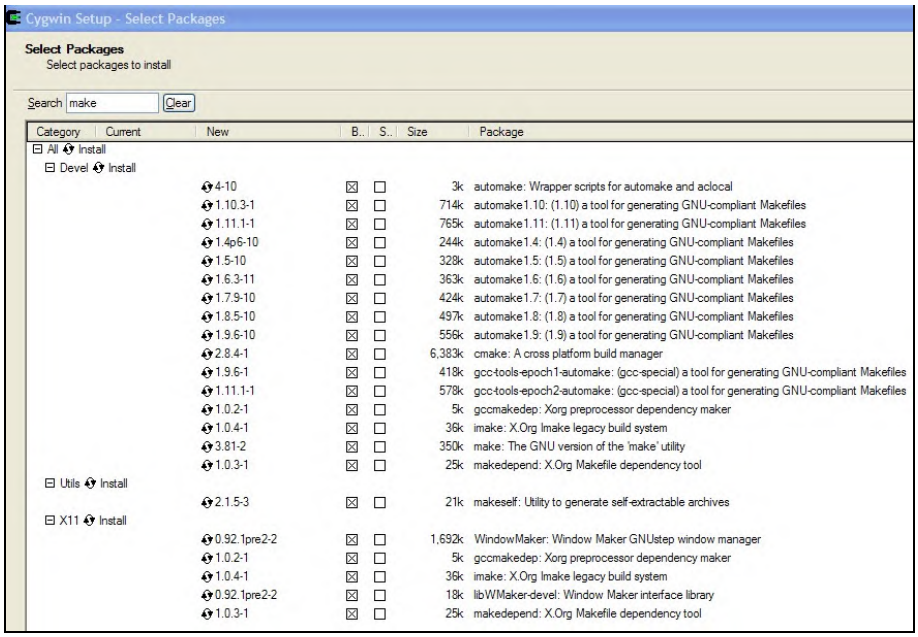


上图显示了错误，错误的原因在于 make 命令不可用，也就是说我们在安装 Cygwin 的时候没有安装 make 命令或者安装出错，这是很多在 Windows 上使用默认方式安装 Cygwin 都会遇到的问题，此时的具体解决方案如下：

和第一次安装 Cygwin 完全一样的过程，在此时安装 Cygwin，Cygwin 会发现系统已经安装了 Cygwin，如果没有新的组件被安装的话，就直接使用原来的安装程序，如果有新的组件安装的话，就把新的组件加入到原来的安装程序中，此时缺少 make 命令，在进入组件时选择的视图如下图所示。



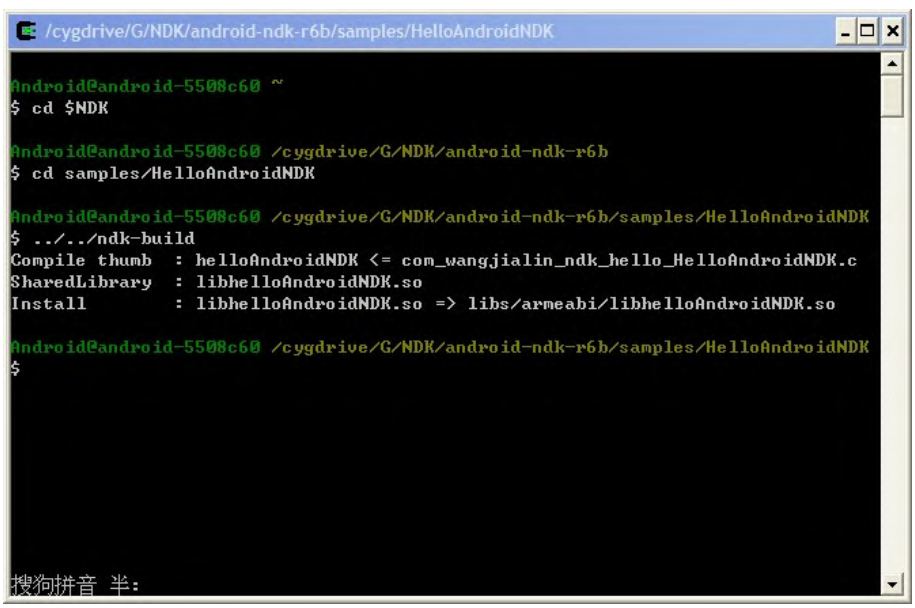
因为默认安装中没有安装 make 命令组件，所以此时我们在 Search 中搜索 make 并把搜索出来的内容全部安装，如下图所示。



这样我们就可以安装 make 命令。

接下来和我们第一次安装的过程是完全一样的，读者可以参考前面的内容。

安装完成后，重新执行上述编译、连接过程，如下图所示。



可以看出此时我们终于编译、连接成功。

此时回到 G:\NDK\android-ndk-r6b\samples\HelloAndroidNDK 中的界面。我们发现新产生了一个 libs 文件夹，里面正是编译、连接好的文件。

8．复制编译、链接好的 libs\armeabi\libhelloAndroidNDK.so 文件，并使用 Java 调用该文件

（1）把复制编译、链接好的 libs\armeabi\libhelloAndroidNDK.so 文件复制到 HelloAndroidNDK 的根目录下。

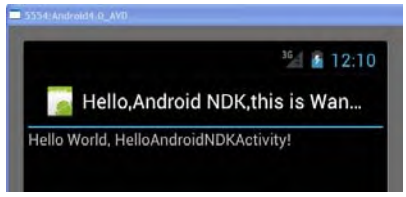
（2）修改 HelloAndroidNDKActivity 的内容，调用 HelloAndroidNDK 类中的本地函数，修改后其内容如下所示。

```
package com.wangjialin.ndk.hello;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroidNDKActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        HelloAndroidNDK helloAndroidNDK = new HelloAndroidNDK();
        String sayHelloToNDK = helloAndroidNDK.sayHelloToNDK();
        this.setTitle(sayHelloToNDK);
    }
}
```

我们在主 Activity 的标题上显示了从本地 C 代码中获得的内容。

因为在 C 代码中的字符串是“Hello，Android NDK，this is WangJialin!”，所以标题应该显示“Hello，Android NDK，this is WangJialin!”。

（3）安装并运行 HelloAndroidNDK，运行的界面如下图所示。



从上图中可以看出，Java 代码成功地调用了本地 C 代码。

第4章

Android NDK中的代码调用

4.1 NDK与JNI的关系

1．JNI简介

（1）JNI是Java语言提供的Java和C/C++相互沟通的机制，Java可以通过JNI调用本地的C/C++代码，本地的C/C++的代码也可以调用Java代码。

（2）通过JNI，Java的多线程代码可以使用本地的C/C++代码，本地C/C++多线程也可调用Java代码。

（3）JNI是Java和C/C++相互通过的接口。

（4）Java通过C/C++使用本地的代码的一个关键性原因在于C/C++代码的高效性。

2．Java通过JNI机制和C/C++沟通的具体步骤

（1）编写包含native本地方法的Java类。

（2）通过javah工具生成C/C++语言的头文件。

（3）使用C/C++语言实现头文件。

（4）使用交叉编译工具对 C/C++本地代码进行编译，最后通过链接生成*.so 可执行的C/C++库。

（5）实际执行Java代码去和本地的C/C++代码相互沟通。

上述过程是传统的Java和C/C++相互的实现步骤，显然这个过程是有些烦琐的。

Android的NDK（Native Development Kit）开发工具集是Android为了更方便Android程序编写者通过 JNI 的机制达到 Java 和本地 C/C++代码相互沟通的强有力武器。通过 Android NDK，Android软件工程师可以很方便地实现Java和本地C/C++代码的相互调用，充分发挥本地硬件的特性和C/C++代码的高效性。

4.2 JNI中的JavaVM与JNIEnv对象

在 3.4 节开发第一个 Android NDK 程序时，我们可以看到通过 javah 生成的头文件com_wangjialin_ndk_hello_HelloAndroidNDK.h的具体内容。

从该生成的代码中可以看到生成的方法声明中有两个参数：JNIEnv *和jobject。

在标准的Java平台下，每一个Process里可以产生很多Java VM对象，每一个Java VM对象都有一个与之对应的Java VM对象，但是在Android平台上，每一个Process只能产生一个Dalvik VM对象，也就是说在一个Android的进程中是通过有且只有一个虚拟器对象来服务所有Java和C/C++代码的。

此时我们来具体分析Android中的JNIEnv对象和Dalvik的Java VM的关系。

（1）JNIEnv *内部包含一个 Pointer， Pointer 指向 Dalvik 的 Java VM 对象的 Fanction Table，JNIEnv *关于程序执行环境的众多函数正是来源于 Dalvik 虚拟机。

（2）Android 中每当一个 Java 线程第一次要调用本地 C/C++代码时，Dalvik 虚拟机实例会为该 Java 线程产生一个 JNIEnv *指针。

（3）Java 每条线程在和 C/C++相互调用时，JNIEnv*是相互独立的，互不干扰，这就提升了并发执行时的安全性。

（4）当本地的 C/C++代码想获得当前线程所要使用的 JNIEnv 时，可以使用 Dalvik VM 对象的 Java VM* jvm→GetEnv()方法，该方法即会返回当前线程所在的 JNIEnv *。

4.3 Android NDK 中 Java 通过 JNI 调用 C 的步骤

Android NDK 中 Java 通过 JNI 调用 C 的具体步骤如下：

（1）在 Eclipse 下创建 Android 工程，该工程需要使用本地的 C 代码。

（2）在创建的 Android 工程建立调用本地 C 代码的 Java 类。

（3）通过 Java 的 javah 生成包含本地 C 代码的 Java 类字节码的头文件。

（4）把此时的 Android 工程复制给 C 代码编写人员。

（5）Android 的 Java 端开发人员和本地的 C 代码编写人员根据头文件分别独立编写自己的代码。

（6）C 代码编写人员编译链接自己的 C 代码，生成*.so 文件，并提供给 Java 端开发人员。

（7）Java 端开发人员调用 C 开发人员提供的*.so 库。

（8）运行包含 Java 和 C 本地代码的工程，产生期望的结果。

Android NDK 中 Java 通过 JNI 调用 C 的案例可参考 3.4 节。

4.4 本地 C 代码调用 Java 中的 Method

1．新建工程

建立一个工程名称为 CallJavaMethodFromNativeCInNDK 的 Android 工程。

单击“Next”按钮进入下一步，选择“Android 4.0”，由于我们使用了 Android 4.0 平台，单击“Next”按钮进入下一步，此时将包的名称命名为 com.wangjialin.ndk.callJavaMethod FromNativeCInNDK。

单击“Finish”按钮完成工程的创建。

2．加载本地的 C 库文件，同时声明本地的 C 方法

在此，创建一个 CallJavaMethodFromNativeCode 的 Java 文件，Modifiers 设置为“public”。

单击“Finish”按钮完成类的创建，此时编写 CallJavaMethodFromNativeCode 的内容如下所示。

```
package com.wangjialin.ndk.callJavaMethodFromNativeCInNDKService;

import android.content.Context;
import android.widget.Toast;

public class CallJavaMethodFromNativeCode {
    static Context mainContext;
    static{
        //加载本地 C 代码
        System.loadLibrary("NativeCodeCallJavaMethod");
    }

    //构造方法进行初始化设置
    public CallJavaMethodFromNativeCode(Context context)
    {
        mainContext = context;
        Toast.makeText(context, "The instructor of CallJavaMethod
        FromNativeCode is invoked",
                Toast.LENGTH_LONG).show();
        //调用本地的 C 代码，进行 C 的初始化工作
        nativeInit();
    }

    /**
     * @param value : 该值来自本地 C 代码
     */
    public  void setMethodByNativeCode(int value)
    {
        Toast.makeText(mainContext, "The Value From C: " + value, Toast.
        LENGTH_LONG).show();
    }

    /**
     * @param value : 该值来自本地 C 代码
     */
    public static void setStaticMethodByNativeCode(int value)
    {
        Toast.makeText(mainContext, "The Value From C: " + value, Toast.
        LENGTH_LONG).show();
    }
```

```
        //C 本地代码的设置方法
        private native void nativeInit();
        //本地 C 代码设置非静态方法
        public native void nativeExecution();
        //本地 C 代码设置静态方法
        public native void nativeStaticExecution();
    }
```

3．编译 Android 文件

与 3.4 节中样，此时会在 G:\Android4.0\workspace\CallJavaMethodFromNativeCInNDK\bin\classes\com\wangjialin\ndk\callJavaMethodFromNativeCInNDKService 目录中找到包含本地调用代码 CallJavaMethodFromNativeCode 被编译后的 class 文件，名称为 CallJavaMethodFromNativeCode.class。

4．使用 javah 工具产生 C 语言的.h 头文件

（1）在 Eclipse 的 CallJavaMethodFromNativeCInNDK 这个工程的根目录下建立一个名称为 jni 的文件夹。

单击“Finish”按钮完成文件夹的创建。

（2）打开 Windows 的命令窗口。

进入刚才新建的 jni 的目录中，具体的命令操作过程如下图所示。

```
C:\Documents and Settings\Android>G:

G:\>cd G:\Android 4.0\workspace\CallJavaMethodFromNativeCInNDK\jni

G:\Android 4.0\workspace\CallJavaMethodFromNativeCInNDK\jni>
```

（3）使用 javah 工具产生 CallJavaMethodFromNativeCode.class 的头文件，具体操作命令如下图所示。

```
G:\Android 4.0\workspace\CallJavaMethodFromNativeCInNDK\jni>javah -classpath ../
bin/classes com.wangjialin.ndk.callJavaMethodFromNativeCInNDKService.CallJavaMet
hodFromNativeCode

G:\Android 4.0\workspace\CallJavaMethodFromNativeCInNDK\jni>
```

此时我们成功地执行了 javah 命令生成了头文件，打开 jni 的文件夹，可以看到自动生成的头文件 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.h。

此时刷新 Eclipse 的工程 CallJavaMethodFromNativeCInNDK，jni 的目录中出现了头文件。打开com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFrom NativeCode.h，其具体内容如下所示。

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_wangjialin_ndk_callJavaMethodFromNativeCInNDK
```

```
    Service_CallJavaMethodFromNativeCode */

#ifndef _Included_com_wangjialin_ndk_callJavaMethodFromNativeCInNDK
Service_CallJavaMethodFromNativeCode
#define _Included_com_wangjialin_ndk_callJavaMethodFromNativeCInNDK
Service_CallJavaMethodFromNativeCode
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class: com_wangjialin_ndk_callJavaMethodFromNativeCInNDK
   Service_CallJavaMethodFromNativeCode
 * Method: nativeInit
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeInit
  (JNIEnv *, jobject);

/*
 * Class: com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_
   CallJavaMethodFromNativeCode
 * Method: nativeExecution
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeExecution
  (JNIEnv *, jobject);

/*
 * Class: com_wangjialin_ndk_callJavaMethodFromNativeCInNDK
   Service_CallJavaMethodFromNativeCode
 * Method: nativeStaticExecution
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeStaticExecution
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

5．编写 C 语言程序

Java 开发者产生 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.h 文件后，就可以把头文件传递给 C 程序开发者，让 C 程序开发者根据 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.h 来开发 C 程序。此时我们将 CallJavaMethodFromNativeCInNDK 整个工程复制到 Android NDK 的 samples 目录下。

此时在复制后的工程文件的 jni 文件夹中编写 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.c 文件，实现 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.h 的功能，com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.c 的内容如下所示。

```
#include "com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_
CallJavaMethodFromNativeCode.h"
jclass mClass;
jobject mObject;
jmethodID mStaticMethodID,mnotStaticMethodID;

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeInit
  (JNIEnv *env, jobject thiz)
{
    jclass clazz = (*env)->GetObjectClass(env,thiz);
    mClass = (jclass)(*env)->NewGlobalRef(env,clazz);
    mObject = (jobject)(*env)->NewGlobalRef(env,thiz);
    mStaticMethodID = (*env)->GetStaticMethodID(env,mClass,"setStatic
    MethodByNativeCode","(I)V");
    mnotStaticMethodID = (*env)->GetMethodID(env,mClass,
    "setMethodByNativeCode","(I)V");
}

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeExecution
  (JNIEnv *env, jobject thiz)
{

    (*env)->CallVoidMethod(env,mObject,mnotStaticMethodID,100);
    return;
}

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeStaticExecution
  (JNIEnv *env, jobject thiz)
{
    (*env)->CallVoidMethod(env,mObject,mStaticMethodID,200);
    return;

}
```

从 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.c 的内容可以看出，我们导入了头文件 com_wangjialin_ndk_callJavaMethodFromNativeCInNDKService_CallJavaMethodFromNativeCode.h，并实现了头文件中声明的方法。

6．根据开发的项目修改 Android.mk 文件

此过程与 3.4 节中一致，读者可自行参考。

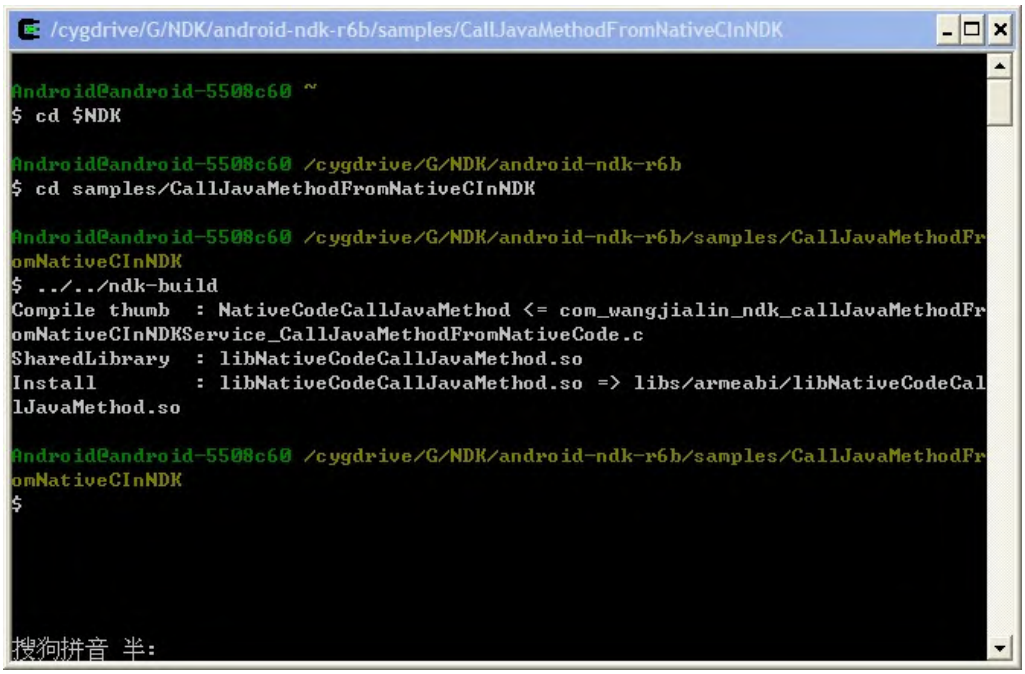
7．编译并链接 C 程序

（1）启动 Cygwin。

（2）输入 cd $NDK 命令进入 Android NDK 目录。

（3）进入 NDK 目录下的/samples/ CallJavaMethodFromNativeCInNDK 目录。

（4）执行 NDK 的编译、链接命令 ndk-build，如下图所示。



可以看出此时编译、连接成功。

此时回到 G:\NDK\android-ndk-r6b\samples\HelloAndroidNDK 中，我们发现新产生了一个 libs 文件夹，里面正是我们编译、连接好的文件。

8. 复制编译、链接好的 libs\armeabi\ libNativeCodeCallJavaMethod.so 文件，并使用 Java 调用该文件

（1）把复制编译、链接好的 libs\armeabi\libNativeCodeCallJavaMethod.so 文件复制到 CallJavaMethodFromNativeCInNDK 的根目录下。

（2）修改 CallJavaMethodFromNativeCInNDKActivity 的内容，调用 CallJavaMethodFromNativeCode 类中的本地函数，修改后其内容如下所示。

```
package com.wangjialin.ndk.callJavaMethodFromNativeCInNDK;

import com.wangjialin.ndk.callJavaMethodFromNativeCInNDKService.
CallJavaMethodFromNativeCode;

import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class CallJavaMethodFromNativeCInNDKActivity extends Activity
implements OnClickListener{
    private Button button1;
    private Button button2;
    private CallJavaMethodFromNativeCode callJavaMethodFromNativeCode;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        button1 = (Button)this.findViewById(R.id.button1);
        button2 = (Button)this.findViewById(R.id.button2);

        button1.setOnClickListener(this);
        button2.setOnClickListener(this);

        callJavaMethodFromNativeCode = new CallJavaMethodFromNativeCode
        (this);
    }

    @Override
    public void onClick(View v) {
        switch(v.getId()){
        case R.id.button1:
            callJavaMethodFromNativeCode.nativeExecution();
            break;
        case R.id.button2:
            callJavaMethodFromNativeCode.nativeStaticExecution();
            break;

        }
    }
}
```

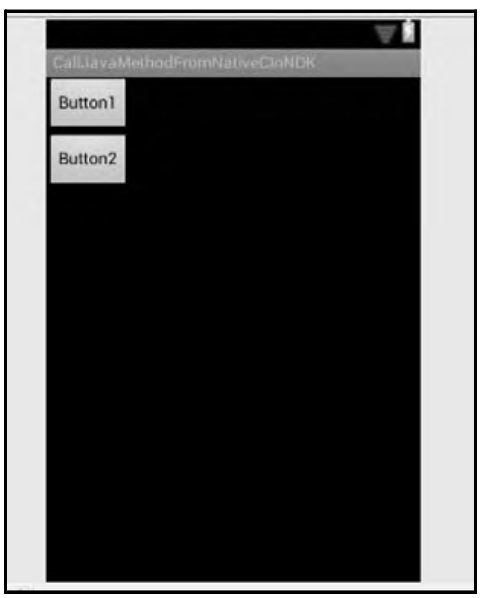
此时我们需要一个 main.xml 的布局文件，其内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Button1" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Button2" />

    </LinearLayout>
```

Main.xml 的 Graphical Layout 如下（左）图所示。

（3）安装并运行 HelloAnCallJavaMethodFromNativeCInNDK，运行的初始界面如下（右）图所示。



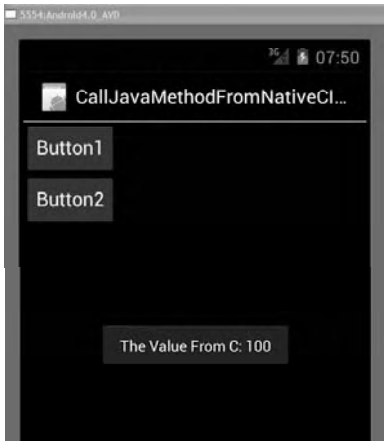
上图（右）表明我们此时成功地调用构造方法并初始化了本地的 C 代码。

单击 Button1，出现如下（左）图所示的界面。

在我们单击 Button1 这个按钮时，Java 代码调用了本地的 C 代码中的 nativeExecution 方法，nativeExecution 反过来调用了 Java 中的 setMethodByNativeCode 方法，并把 C 代码的 100 传入 Java 代码中，此时 Java 通过 Toast 的方式把本地传递的 100 显示出来，C 的代码如下所示。

```
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFromNative
CInNDKService_CallJavaMethodFromNativeCode_nativeExecution
  (JNIEnv *env, jobject thiz)
{

    (*env)->CallVoidMethod(env,mObject,mnotStaticMethodID,100);
    return;
}
```

单击 Button2 按钮，出现如下（右）图所示的视图。



可以看出当我们单击 Button2 时，成功地实现了 Java 代码调用本地 C 代码，本地 C 代码反过来调用 Java 的静态方法，具体过程为：Java 调用本地的 C 方法 nativeStaticExecution，本地的 nativeStaticExecution 反过来调用 Java 的 setStaticMethodByNativeCode，并通过 Toast 显示出来自本地 C 代码中的信息，nativeStaticExecution 的具体内容如下所示。

```
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_callJavaMethodFrom
NativeCInNDKService_CallJavaMethodFromNativeCode_nativeStaticExecution
  (JNIEnv *env, jobject thiz)
{
    (*env)->CallVoidMethod(env,mObject,mStaticMethodID,200);
    return;

}
```

4.5　本地 C 代码获得 Java 对象的属性值

1．新建工程

建立一个工程名称为 GetJavaPropertyFromNativeC 的 Android 工程。

单击“Next”按钮进入下一步，选择“Android 4.0”，将包的名称命名为 com.wangjialin.ndk.getJavaPropertyFromNativeC，单击“Finish”按钮完成工程的创建。

2．加载本地的 C 库文件，同时声明本地的 C 方法

在此，我们创建一个 GetJavaPropertyNative 的 Java 文件，Modifiers 设置为“public”，单击“Finish”按钮完成类的创建，GetJavaPropertyNative 的内容如下所示。

```
package com.wangjialin.ndk.getJavaPropertyFromNativeC;

import android.content.Context;
```

```
import android.widget.Toast;

public class GetJavaPropertyNative {
    private Context context;
    private int numberInJava;
    static{
        System.loadLibrary("getJavaPropertyFromNativeC");
    }

    public GetJavaPropertyNative(Context context)
    {
        this.context = context;
        numberInJava = 1000;
        nativeInit();
    }

    private void showMessage(int value)
    {
        Toast.makeText(context, value + " is from native C", Toast.LENGTH_
        LONG).show();
    }

    private native void nativeInit();

    public native void nativeBusiness();
}
```

3．编译 Android 文件

与 3.4 节中的一样，此时会在 G:\Android 4.0\workspace\GetJavaPropertyFromNativeC\bin\classes\com\wangjialin\ndk\getJavaPropertyFromNativeC 目录中找到包含本地调用代码 GetJavaPropertyNative 被编译后的 class 文件，名称为 GetJavaPropertyNative.class。

4．使用 javah 工具产生 C 语言的.h 头文件

在 Eclipse 的 GetJavaPropertyFromNativeC 这个工程的根目录下建立一个名称为 jni 的文件夹，单击“Finish”按钮完成文件夹的创建。

打开 Windows 的命令窗口，进入刚才新建的 jni 的目录中，具体的命令操作过程如下图所示。

```
C:\Documents and Settings\Android>g:

G:\>cd G:\Android 4.0\workspace\GetJavaPropertyFromNativeC\jni

G:\Android 4.0\workspace\GetJavaPropertyFromNativeC\jni>
```

使用 javah 工具产生 GetJavaPropertyNative.class 的头文件，具体操作命令如下图所示。

```
G:\Android 4.0\workspace\GetJavaPropertyFromNativeC\jni>javah -classpath ../bin/
classes com.wangjialin.ndk.getJavaPropertyFromNativeC.GetJavaPropertyNative

G:\Android 4.0\workspace\GetJavaPropertyFromNativeC\jni>
```

此时我们成功地执行了 javah 命令生成了头文件，打开 jni 的文件夹，可以看到自动生成的头文件 com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJavaPropertyNative.h，。

此时刷新 Eclipse 的工程 GetJavaPropertyFromNativeC，jni 的目录中出现了头文件。

5．编写 C 语言程序

此时我们将 GetJavaPro- pertyFromNativeC 整个工程复制到 Android NDK 的 samples 目录下。

此时我们在复制后的工程文件的 jni 文件夹中编写 com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJavaPropertyNative.c 文件，实现 com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJavaPropertyNative.h 的功能，com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJavaPropertyNative.c 的内容如下所示。

```
#include "com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJava
PropertyNative.h"
jobject mObject;
jmethodID mMethodID;
jfieldID mFieldID;

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_getJavaProperty
FromNativeC_GetJavaPropertyNative_nativeInit
  (JNIEnv *env, jobject thiz)
  {
      jclass clazz = (*env)->GetObjectClass(env,thiz);
      mObject = (jobject)(*env)->NewGlobalRef(env,thiz);
      mMethodID = (*env)->GetMethodID(env,clazz,"showMessage","(I)V");
      mFieldID = (*env)->GetFieldID(env,clazz,"numberInJava","I");
      return;
  }

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_getJavaProperty
FromNativeC_GetJavaPropertyNative_nativeBusiness
  (JNIEnv *env, jobject thiz)
  {
int numberInC,sum = 0;
      numberInC =(int)(*env)->GetObjectField(env,mObject,mFieldID);
      sum = numberInC + numberInC;
      (*env)->CallVoidMethod(env,mObject,mMethodID,sum);
      return;
  }
```

从 com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJavaPropertyNative.c 的内容可以看出，我们导入了头文件 com_wangjialin_ndk_getJavaPropertyFromNativeC_GetJavaPropertyNative.h，并实现了头文件中声明的方法。

6．根据开发的项目修改 Android.mk 文件

此过程与 3.4 节中一致，读者可自行参考。

7．编译并链接 C 程序

启动 Cygwin，输入 cd $NDK 命令进入 Android NDK 目录。进入 NDK 目录下的/samples/GetJavaPropertyFromNativeC 目录。执行 NDK 的编译、链接命令 ndk-build，如下图所示。

可以看出此时编译、连接成功。

此时我们回到 G:\NDK\android-ndk-r6b\samples\GetJavaPropertyFromNativeC 中，我们发现新产生了一个 libs 文件夹，里面正是我们编译、连接好的文件。

8．复制编译、链接好的 libs\armeabi\libgetJavaPropertyFromNativeC.so 文件，并使用 Java 调用该文件

把复制编译、链接好的 libs\armeabi\libgetJavaPropertyFromNativeC.so 文件复制到 GetJavaPropertyFromNativeC 的根目录下。

修改 GetJavaPropertyFromNativeCActivity 的内容，调用 GetJavaPropertyNative 类中的本地函数，修改后其内容如下所示。

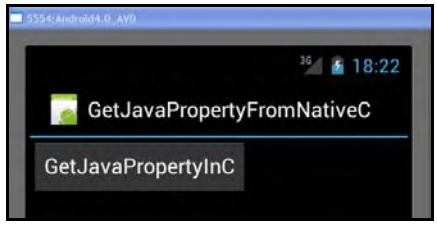
```
package com.wangjialin.ndk.getJavaPropertyFromNativeC;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class GetJavaPropertyFromNativeCActivity  extends Activity
implements OnClickListener{
    private Button button1;
    private Button button2;
    private GetJavaPropertyNative getJavaPropertyNative;
    @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);

       button1 = (Button)this.findViewById(R.id.button1);

       button1.setOnClickListener(this);
       button2.setOnClickListener(this);

       getJavaPropertyNative = new GetJavaPropertyNative(this);
    }
```

```
        @Override
        public void onClick(View v) {
            switch(v.getId()){
            case R.id.button1:
                getJavaPropertyNative.nativeBusiness();
                break;

            }
        }
    }
```

此时我们需要一个 main.xml 的布局文件，其内容与 4.4 节一致。

Main.xml 的 Graphical Layout 如下（右）图所示。

安装并运行 GetJavaPropertyFromNativeC，运行的初始界面如下（右）图所示。



上图表明此时成功地调用构造方法并初始化了本地的 C 代码。

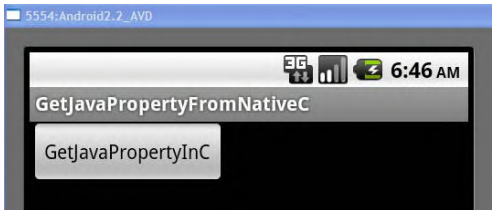
此时当单击“GetJavaPropertyInC”按钮时出现了如下图所示的错误。

```
JNI WARNING: GetObjectField for field 'numberInJava' of expected type L, got I
            in Lcom/wangjialin/ndk/getJavaPropertyFromNativeC/GetJavaPropertyNative;.nativeBus...
"main" prio=5 tid=1 NATIVE
  | group="main" sCount=0 dsCount=0 obj=0x40996460 self=0x12810
  | sysTid=548 nice=0 sched=0/0 cgrp=default handle=1074082952
  | schedstat=( 571317091 2240800906 148 ) utm=32 stm=25 core=0
  at com.wangjialin.ndk.getJavaPropertyFromNativeC.GetJavaPropertyNative.nativeBusiness(Native ...
  at com.wangjialin.ndk.getJavaPropertyFromNativeC.GetJavaPropertyFromNativeCActivity.onClick(G...
  at android.view.View.performClick(View.java:3460)
  at android.view.View$PerformClick.run(View.java:13955)
  at android.os.Handler.handleCallback(Handler.java:605)
  at android.os.Handler.dispatchMessage(Handler.java:92)
  at android.os.Looper.loop(Looper.java:137)
  at android.app.ActivityThread.main(ActivityThread.java:4340)
  at java.lang.reflect.Method.invokeNative(Native Method)
  at java.lang.reflect.Method.invoke(Method.java:511)
  at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:784)
  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:551)
  at dalvik.system.NativeStart.main(Native Method)
VM aborting
Fatal signal 11 (SIGSEGV) at 0xdeadd00d (code=1)
```

从理论上讲，我们是不会出现这个错误的，准确地说这只是 JNI WARNING 而已，笔者为这个问题调试了大约一天的时间，而未能够解决，结论是无法解决，因为我们的 NDK 代码没有任何问题！

于是根据 Android 4.0 模拟器在网络方面的系统错误的处理经验，笔者把该程序放在了 Android 2.2 的平台上，重新使用 Android 2.2 平台编译该程序，并把<uses-sdk android:minSdk Version="14" />修改为<uses-sdk android:minSdkVersion="8" />然后发布后，此时的 Android 2.2 的初始界面如下（左）图所示。

此时单击“GetJavaPropertyInC”按钮出现如下（右）图所示的界面。



此时成功运行了程序。

这表明 Android 4.0 模拟器对该程序的处理是有 bug 的，类似的情况在后面的代码调用章节中也有，读者需多加注意。

至此，本地 C 代码获得 Java 对象的属性值的方法讲解完毕。

4.6 多个类中有本地 C 代码的调用

1. 新建工程

建立一个工程名称为 MultipleNativeC 的 Android 工程。

单击“Next”按钮进入下一步，选择“Android 4.0”，将包的名称命名为 com.wangjialin.ndk.multipleNativeC。

单击“Finish”按钮完成工程的创建。

2．加载本地的 C 库文件，同时声明本地的 C 方法

编写 ShowValueFromNativeC 的内容如下所示。

```
package com.wangjialin.ndk.multipleNativeC;

public class ShowValueFromNativeC {
    public static native void showValueFromNativeC();
}
```

建立第二个包含本地方法的 Java 类，类名为 ShowMessageNative，具体内容如下所示。

```
package com.wangjialin.ndk.multipleNativeC;

import android.content.Context;
import android.widget.Toast;

public class ShowMessageNative {
    private Context context;
    private int numberInJava;
    static{
        System.loadLibrary("multipleNativeC");
    }
    public ShowMessageNative(Context context)
    {
        this.context = context;
        numberInJava = 88;
        nativeInit();
    }

    private void showMessage(int value)
    {
        Toast.makeText(context, value + " is from native C", Toast.
        LENGTH_LONG).show();
    }

    private native void nativeInit();

}
```

3．编译 Android 文件

会在 G:\Android 4.0\workspace\MultipleNativeC\bin\classes\com\wang jialin\ndk\multipleNativeC 目录中找到包含本地调用代码 ShowValueFromNativeC 和 ShowMessage Native 被编译后的 class 文件，名称分别为 ShowValueFromNativeC.class 和 ShowMessageNative. class。

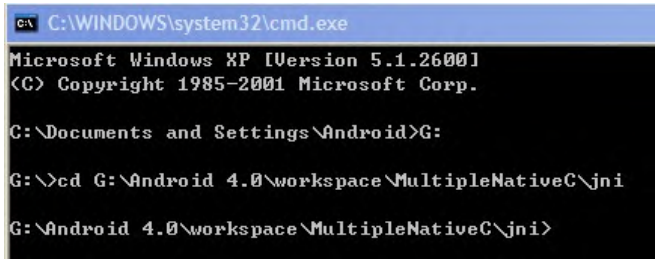
4．使用 javah 工具产生 C 语言的.h 头文件

（1）在 Eclipse 的 GetJavaPropertyFromNativeC 这个工程的根目录下建立一个名称为 jni 的文件夹。

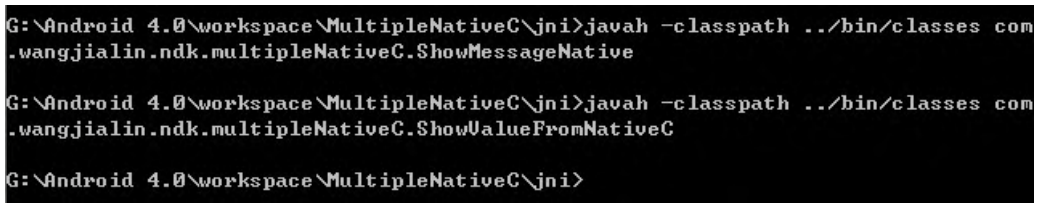
单击“Finish”按钮完成文件夹的创建。

（2）打开 Windows 的命令窗口。

（3）进入刚才新建的 jni 的目录中，具体的命令操作过程如下图所示。



（4）使用 javah 工具产生 ShowMessageNative.class 和 ShowValueFromNativeC.class 的头文件，具体操作命令如下图所示。



（5）此时我们成功地执行了 javah 命令生成了头文件，打开 jni 的文件夹，可以看到自动生成的头文件 com_wangjialin_ndk_multipleNativeC_ShowMessageNative.h 和 com_wangjialin_ndk_multipleNativeC_ShowValueFromNativeC.h。

此时刷新 Eclipse 的工程 MultipleNativeC，jni 的目录中出现了头文件。

打开 com_wangjialin_ndk_multipleNativeC_ShowMessageNative.h，其具体内容如下所示。

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_wangjialin_ndk_multipleNativeC_ShowMessageNative */

#ifndef _Included_com_wangjialin_ndk_multipleNativeC_ShowMessageNative
#define _Included_com_wangjialin_ndk_multipleNativeC_ShowMessageNative
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class: com_wangjialin_ndk_multipleNativeC_ShowMessageNative
 * Method: nativeInit
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_multipleNativeC_Show
MessageNative_nativeInit
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

打开 com_wangjialin_ndk_multipleNativeC_ShowValueFromNativeC.h，其具体内容如下所示。

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_wangjialin_ndk_multipleNativeC_ShowValueFrom
NativeC */

#ifndef _Included_com_wangjialin_ndk_multipleNativeC_ShowValueFrom
NativeC
#define _Included_com_wangjialin_ndk_multipleNativeC_ShowValueFrom
NativeC
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class: com_wangjialin_ndk_multipleNativeC_ShowValueFromNativeC
 * Method: showValueFromNativeC
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_com_wangjialin_ndk_multipleNativeC_Show
ValueFromNativeC_showValueFromNativeC
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

5．编写 C 语言程序

将 MultipleNativeC 整个工程复制到 Android NDK 的 samples 目录下。

此时我们在复制后的工程文件的 jni 文件夹中编写 businessCode.c 文件，实现 com_wangjialin_ndk_multipleNativeC_ShowMessageNative.h 和 com_wangjialin_ndk_multipleNativeC_ShowValueFromNativeC.h 的功能，businessCode.c 的内容如下所示。

```
#include "com_wangjialin_ndk_multipleNativeC_ShowMessageNative.h"
#include "com_wangjialin_ndk_multipleNativeC_ShowValueFromNativeC.h"
jobject mObject;
jmethodID mMethodID;
jfieldID mFieldID;

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_multipleNativeC_Show
MessageNative_nativeInit
  (JNIEnv * env , jobject thiz)
  {
    jclass clazz = (*env)->GetObjectClass(env,thiz);
    mObject = (jobject)(*env)->NewGlobalRef(env,thiz);
    mMethodID = (*env)->GetMethodID(env,clazz,"showMessage","(I)V");
    mFieldID = (*env)->GetFieldID(env,clazz,"numberInJava","I");
    return;
  }

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_multipleNativeC_Show
ValueFromNativeC_showValueFromNativeC
```

```
(JNIEnv * env, jclass clazz)
{
  int valueFromJava,sum=0;
  valueFromJava = (int)(*env)->GetObjectField(env,mObject,mFieldID);
  sum = valueFromJava + 11;
  (*env)->CallVoidMethod(env,mObject,mMethodID,sum);
  return;
}
```

从 businessCode.c 的内容可以看出，我们导入了头文件 com_wangjialin_ndk_multiple NativeC_ShowMessageNative.h 和 com_wangjialin_ndk_multipleNativeC_ShowValueFromNativeC.h，并实现了头文件中声明的方法。

6．根据开发的项目修改 Android.mk 文件

此过程与 3.4 节一致，读者可自行参考。

7．编译并链接 C 程序

启动 Cygwin，输入 cd $NDK 命令进入 NDK 目录下的/samples/ MultipleNativeC 目录，执行 NDK 的编译、链接命令 ndk-build，如下图所示。

```
$ ../../ndk-build
Compile thumb  : multipleNativeC <= businessCode.c
SharedLibrary  : libmultipleNativeC.so
Install        : libmultipleNativeC.so => libs/armeabi/libmultipleNativeC.so

Android@android-5508c60 /cygdrive/G/NDK/android-ndk-r6b/samples/MultipleNativeC
$
```

可以看出此时编译、连接成功。

我们发现新产生了一个 libs 文件夹，里面正是我们编译、连接好的文件。

8．复制编译、链接好的 libs\armeabi\ libmultipleNativeC.so 文件，并使用 Java 调用该文件

把复制编译、链接好的 libs\armeabi\ libmultipleNativeC.so 文件复制到 MultipleNativeC 的根目录下。

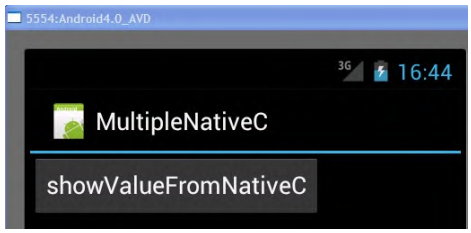
修改 MultipleNativeCActivity 的内容，调用本地函数，修改后其内容如下所示。

```
package com.wangjialin.ndk.multipleNativeC;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class MultipleNativeCActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        new ShowMessageNative(this);
```

```
        }

        public void showValue(View view)
        {
         ShowValueFromNativeC.showValueFromNativeC();
        }
    }
```

此时我们需要一个 main.xml 的布局文件，其内容与 4.4 节一致。

安装并运行 MultipleNativeC，运行的初始界面如下图所示。



上图表明我们此时成功地调用构造方法并初始化了本地的 C 代码。

至此，多个类中有本地 C 代码的调用讲解完毕。

在本地代码中创建 Java 对象、Java 代码调用 C++代码、C++代码调用 Java 代码，实现步骤与上述方法一致，读者可自行实现。

处理异常：

因为 Android NDK 编程涉及 Java、C/C++，所以就比单纯的 Java 或 C/C++更加复杂。到目前为止尚未出现比较好的调试工具，这就直接导致了 NDK 在出现错误时很难调试的问题，同时也就要求我们非常清楚程序运行每一步的流程。

我们在 Java 处理 Android NDK 异常时，基本能够做的就是处理 Java 加载*.so 库的异常，具体示例代码如下所示。

```
static{ try {   System.loadLibrary(…);  }   catch(UnsatisfiedLinkError
ule){//Log.e…//System.out.println… }}
```

该段代码可以让我们捕获本地 C/C++库加载异常。

4.7　Java、Dalvik VM、C/C++的运行机制与流程

在 Android 的 NDK 中，Java、Dalvik VM、C/C++的关系如下：

（1）Java 的 dex 字节码和 C/C++的*.so 同时运行 Dalvik VM 之内，共同使用一个进程空间。

（2）Java 和 C/C++可以相互调用，其调用的关键均是 Dalvik VM。

（3）一般而言，比较经典的模式是 Java 通过 JNI 的 C 组件和 C++相互沟通，一般的业务处理代码是放在 C++中的。

（4）C++代码处于核心的控制地位更具有价值。

当 Java 代码需要 C/C++代码时，在 Dalvik 虚拟机加载进*.so 库时，会先调用 JNI_OnLoad( ) 函数，此时就会把 JavaVM 对象的指针存储于 C 层 JNI 组件的全局环境中，在 Java 层调用 C 层的本地函数时，调用 C 本地函数的线程必然通过 Dalvik VM 来调用 C 层的本地函数，此时 Dalvik 虚拟机会为本地的 C 组件实例化一个 JNIEnv 指针，该指针指向 Dalvik 虚拟机的具体的函数列表，当 JNI 的 C 组件调用 Java 层的方法或者属性时，需要通过 JNIEnv 指针来进行调用。

当 C++组件主动调用 Java 的方法或者属性时，需要通过 JNI 的 C 组件把 JNIEnv 指针传递给 C++组件，此后，C++组件即可通过 JNIEnv 指针来掌控 Java 层代码。

4.8 Java 中分配线程调用 C/C++函数

1．新建工程

建立一个工程名称为 MultipleThreadInJavaInvokeCPP 的 Android 工程。

单击“Next”按钮进入下一步，选择“Andvar 4.0”，如下图所示。将包的名称命名为 com.wangjialin.ndk.multipleThreadInJavaInvokeCPP。单击“Finish”按钮完成工程的创建。

2．加载本地的 C++库文件，同时声明本地方法

编写包含本地方法的类 WorkerNative，如下所示。

```
package com.wangjialin.ndk.multipleThreadInJavaInvokeCPP;

abstract public class WorkerNative {
    private int number;
    public Person person;
    static {
        System.loadLibrary("mutipleThreadInvokeC");
    }

    public WorkerNative()
    {
        person = new Person();
        number = getNumber();
        nativeInitial(person);
    }

    abstract protected int getNumber();
    private native void nativeInitial(Object person);
    public native void doBusiness();

}
```

此时我们需要一个实体对象 Person，其具体内容如下所示。

```
package com.wangjialin.ndk.multipleThreadInJavaInvokeCPP;

public class Person {
    private int age;
    private String threadName;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getThreadName() {
        return threadName;
    }
    public void setThreadName(String threadName) {
        this.threadName = threadName;
    }

    private void setValue(int value)
    {
        age = value;
        threadName = Thread.currentThread().getName() + "is In Method of
        setValue";
    }
}
```

接下来再编写两个 WorkerNative 的具体实现者：SubWorkerNative1 和 SubWorker Native2，具体内容如下所示。

```
package com.wangjialin.ndk.multipleThreadInJavaInvokeCPP;

public class SubWorkerNative1 extends WorkerNative {

    @Override
    protected int getNumber() {
        return 100;
    }

}

package com.wangjialin.ndk.multipleThreadInJavaInvokeCPP;

public class SubWorkerNative2 extends WorkerNative {

    @Override
    protected int getNumber() {
        return 150;
    }

}
```

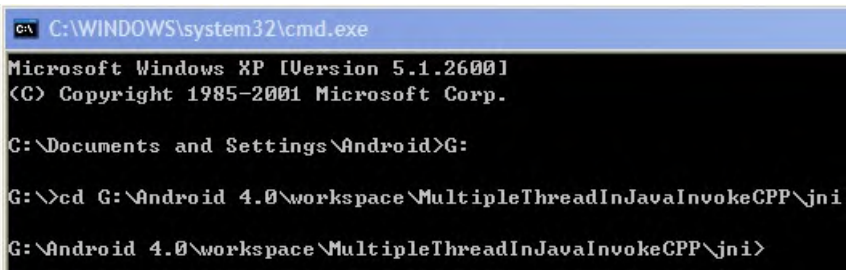
3．编译 Android 文件

在 G:\Android 4.0\workspace\MultipleThreadInJavaInvokeCPP\bin\classes\ com\wangjialin\ndk\multipleThreadInJavaInvokeCPP 目录中找到包含本地调用代码 Worker Native 被编译后的 class 文件，名称为 WorkerNative.class。

4．使用 javah 工具产生 C 语言的.h 头文件

在 Eclipse 的 MultipleThreadInJavaInvokeCPP 这个工程的根目录下建立一个名称为 jni 的文件夹，单击“Finish”按钮完成文件夹的创建。

打开 Windows 的命令窗口，进入刚才新建的 jni 的目录中，具体的命令操作过程如下图所示。



使用 javah 工具产生 WorkerNative.class 的头文件，具体操作命令如下图所示。

此时我们成功地执行了 javah 命令生成了头文件，打开 jni 的文件夹，可以看到自动生成的头文件 com_wangjialin_ndk_multipleThreadInJavaInvokeCPP_WorkerNative.h。

此时刷新 Eclipse 的工程 MultipleThreadInJavaInvokeCPP，jni 的目录中出现了头文件。

5．编写本地 C 语言程序

将 MultipleThreadInJavaInvokeCPP 整个工程复制到 Android NDK 的 samples 目录下。

此时我们在复制后的 jni 文件夹中编写 mutipleThreadInvokeC.c 文件，实现 com_wangjialin_ndk_multipleThreadInJavaInvokeCPP_WorkerNative.h 的功能，mutipleThreadInvokeC.c 的内容如下所示。

```
#include "com_wangjialin_ndk_multipleThreadInJavaInvokeCPP_
WorkerNative.h"
jobject mObject,personObject;
jfieldID ageField;
jmethodID setValueMethodID;

JNIEXPORT void JNICALL Java_com_wangjialin_ndk_multipleThreadInJava
InvokeCPP_WorkerNative_nativeInitial
  (JNIEnv *env, jobject thiz, jobject person)
```

```
    {
      jclass clazz = (*env)->GetObjectClass(env,thiz);
      mObject = (jobject)(*env)->NewGlobalRef(env,thiz);
      ageField = (*env)->GetFieldID(env,clazz,"number","I");

      jclass personClazz = (*env)->GetObjectClass(env,person);
      personObject = (jobject)(*env)->NewGlobalRef(env,person);
      setValueMethodID = (*env)->GetMethodID(env,personClazz,
      "setValue","(I)V");

    }

  JNIEXPORT void JNICALL Java_com_wangjialin_ndk_multipleThreadInJava
  InvokeCPP_WorkerNative_doBusiness
    (JNIEnv *env, jobject thiz)
    {
      int numberFromJava,result = 0;
      numberFromJava = (int)(*env)->GetObjectField(env,mObject,ageField);

      result = numberFromJava + numberFromJava;

      (*env)->CallVoidMethod(env,personObject,setValueMethodID,result);
    }
```

从 mutipleThreadInvokeC.c 的内容可以看出，我们导入了头文件 com_wangjialin_ndk_multipleThreadInJavaInvokeCPP_WorkerNative.h，并实现了头文件中声明的方法。

6．根据开发的项目修改 Android.mk 文件

此过程与 3.4 节一致。

7．编译并链接 C 程序

启动 Cygwin，输入 cd $NDK 命令进入 NDK 目录下的/samples/ MultipleThreadInJavaInvokeC 目录。执行 NDK 的编译、链接命令 ndk-build，如下图所示。

```
$ ../../ndk-build
Compile thumb  : mutipleThreadInvokeC <= mutipleThreadInvokeCPP.c
SharedLibrary  : libmutipleThreadInvokeC.so
Install        : libmutipleThreadInvokeC.so => libs/armeabi/libmutipleThreadInvo
keC.so

Android@android-5508c60 /cygdrive/G/NDK/android-ndk-r6b/samples/MultipleThreadIn
JavaInvokeCPP
$
```

可以看出此时编译、链接成功。

我们发现新产生了一个 libs 文件夹，里面正是我们编译、链接好的文件。

8．复制编译、链接好的 libs\armeabi\ libmutipleThreadInvokeC.so 文件，并使用 Java 调用该文件

把复制编译、链接好的 libs\libmutipleThreadInvokeC.so 文件复制到 JavaInvokeCPP 的根目录下。

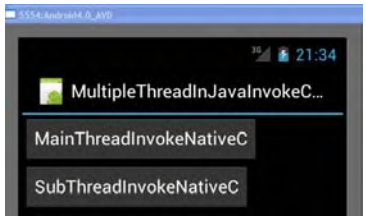
修改 MultipleThreadInJavaInvokeCPPActivity 的内容，调用本地函数，修改后其内容如下所示。

```
package com.wangjialin.ndk.multipleThreadInJavaInvokeCPP;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MultipleThreadInJavaInvokeCPPActivity extends Activity {
    private Button mainThreadButton;
    private Button subThreadButton;
    private Handler handler;
    private WorkerNative native1,native2;
    private Thread thread;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);


        handler = new Handler(){
         @Override
         public void handleMessage(Message msg) {
             Toast.makeText(getApplicationContext(),
                     native2.person.getAge()+""+native2.person.
                     getThreadName(), Toast.LENGTH_LONG).show();
         }
        };

        mainThreadButton = (Button)this.findViewById(R.id.mainThread);
        subThreadButton = (Button)this.findViewById(R.id.subThread);
    }


    public void mainThread(View view)
    {
        native1 = new SubWorkerNative1();
        native1.doBusiness();
        Toast.makeText(getApplicationContext(),
                native1.person.getAge()+ native1.person.getThreadName()
                + "", Toast.LENGTH_LONG).show();
    }

    public void subThread(View view)
    {
        thread = new Thread(new RunTask());
        thread.start();
    }
```

```
        class RunTask implements Runnable{

            @Override
            public void run() {
                native2 = new SubWorkerNative2();
                native2.doBusiness();
                handler.sendEmptyMessage(MODE_PRIVATE);
            }

        }
    }
```

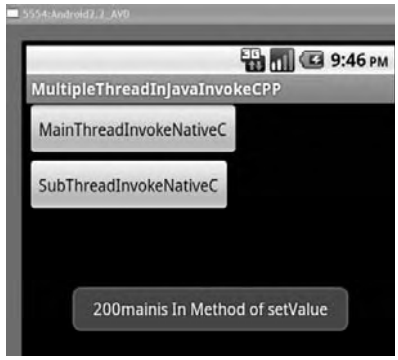
此时我们需要编写界面，main.xml 的具体内容与 4.5 节一致。

安装并运行 MultipleThreadInJavaInvokeCPP，程序运行如下图所示。



把程序发布到 Android 2.2 模拟器上，初始运行界面如下（左）图所示。

单击“MainThreadInvokeNativeC”选项，出现如下（右）图所示的运行效果。



单击“SubThreadInvokeNativeC”选项，出现如下图所示的运行效果。

运行效果正如我们所预期的一样正确。

至此，Java 中分配线程调用 C/C++函数的方法讲解完毕。

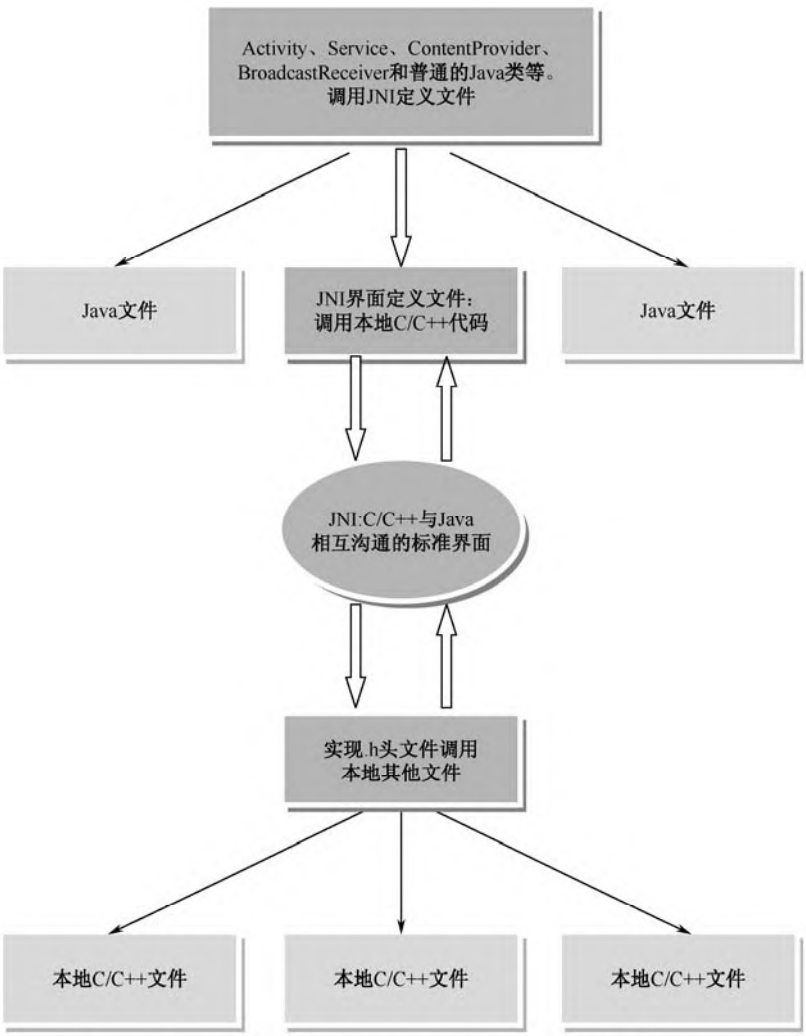
C/C++本地代码通过分配线程调用 Java 函数的实现步骤与上述过程一致，读者可自行实现。

第5章

NDK的架构/设计模式及NDK与软/硬件整合、云计算

5.1 NDK 的架构图及思考

NDK 的架构图如下图所示。



其实，更加理想的情况是 Java 与 JNI 的 C 组件相互沟通，然后 JNI 的 C 组件和 C++组件相互沟通，也就是说 Java 负责整个程序的界面，JNI 的 C 组件控制流程，C++负责实际的业务代码。在 Android NDK 中，更加有意义的是 C++如何掌控 Java。

5.2 Facade 设计模式剖析

1．外观模式解释

Facade 设计模式又叫外观模式。

外观模式（Facade Pattern）是比较常用的一种软件设计模式，属于结构型模式的一种。它是一组具有类似功能的类群提供一个统一的高层接口，比如类库、子系统等。这个高层接口以一种简单一致的界面展现给使用者，从而使得子系统、类群等更容易使用。

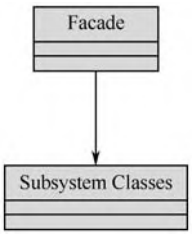
英文定义为：Provide a unified interface to a set of interfaces in a subsystem。Facade defines a higher-level interface that makes the subsystem easier to use。

2．外观模式的 UML 图

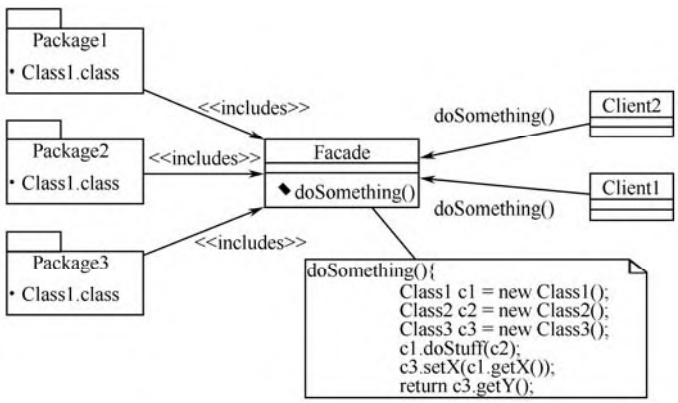
Facade 设计模式涉及以下角色：

- 外观（Facade）角色——为调用方定义简单的调用接口。
- 客户端（Clients）角色——通过 Facade 接口调用提供某功能的内部类群，读写子系统各个接口的数据资源。
- 类群（Packages）角色——功能提供者，指提供功能的类群（模块或子系统）。

外观模式的 UML 图如下图所示。



更具体的 UML 图如下图所示。



3．外观模式深入分析

在真实的应用系统中，一个子系统可能由很多类组成。子系统的客户端为了它们的需要，需要和子系统中的一些类进行交互。客户端和子系统的类进行直接的交互会导致客户端对象和子系统之间高度耦合。任何的类似于对子系统中类的接口的修改，会对依赖于它的所有的客户类造成影响。外观模式（Facade Pattern）为子系统提供了一个更高层次、更简单的接口，从而降低了子系统的复杂度和依赖程度，这使得子系统更易于使用和管理。

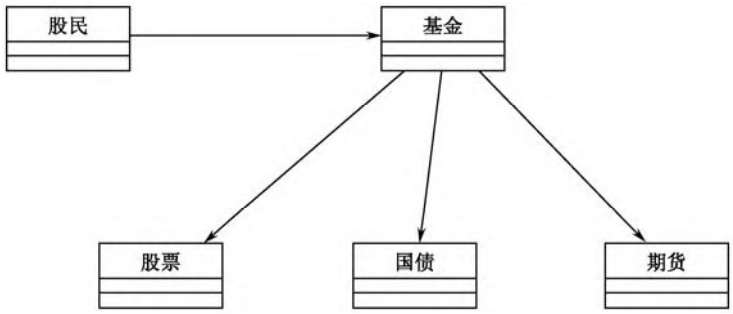
外观是一个能为子系统和客户端提供简单接口的类。当正确地应用外观时，客户不再直接和子系统中的类交互，而是与外观交互。外观承担与子系统中类交互的责任。实际上，外观是子系统与客户的接口，这样外观模式降低了子系统和客户的耦合度。这样，就可以使得当子系统改变时，客户端可以保持不变性。

尽管客户端使用由外观提供的简单接口，但是当需要的时候，客户端还是可以视外观不存在，直接访问子系统中的低层次的接口。这种情况下，它们之间的依赖/耦合度和原来一样。

4．外观模式使用场景分析及代码实现

例如，期货、股票、国债分别属于单一的不同的投资类型，而基金可以把这些类型综合起来，也就是说，基金相当于期货、股票、国债的外观，用户只需要通过基金这个接口进行理财投资即可。

UML 模型图如下图所示。



建立股票类，代码如下所示。

```
package com.diermeng.designPattern.Facade.impl;
/*
 * 股票类
 */
public class Stock {
    /*
     * 购买股票
     */
    public void buy() {
        System.out.println("购买股票");
    }
}
```

建立国债类，代码如下所示。

```
package com.diermeng.designPattern.Facade.impl;
/*
 * 国债类
 */
public class NationalDebt {
    /*
     * 购买国债
     */
    public void buy() {
        System.out.println("购买国债");
    }
}
```

建立期货类，代码如下所示。

```
package com.diermeng.designPattern.Facade.impl;
/*
 * 期货类
 */
public class Futures {

    public void buy() {
        System.out.println("购买期货");
    }
}
```

建立基金类，代码如下所示。

```
package com.diermeng.designPattern.Facade.impl;
/*
 * 基金类
 */
public class Fund {
    //声明股票
    private Stock stock;
    //声明国债
    private NationalDebt guozai;
    //声明期货
    private Futures futures;

    //构造方法 实例化声明的变量
    public Fund() {
        this.guozai = new NationalDebt();
        this.stock = new Stock();
        this.futures = new Futures();
    }

    //A 种基金类型
    public void fundA() {
```

```
            this.guozai.buy();
            this.futures.buy();
        }
        //B 种基金类型
        public void fundB() {
            this.guozai.buy();
            this.stock.buy();
        }
        //C 种基金类型
        public void fundC() {
            this.futures.buy();
            this.stock.buy();
        }

        //D 种基金类型
        public void fundD() {
            this.guozai.buy();
            this.stock.buy();
            this.futures.buy();
        }
    }
```

最后建立测试客户端，代码如下所示。

```
    package com.diermeng.designPattern.Facade.client;

    import com.diermeng.designPattern.Facade.impl.Fund;

    /*
     * 测试客户端
     */
    public class FacadeTest {
        public static void main(String[] args) {
            //建立基金类
            Fund jijin = new Fund();

            //调用相应的方法
            System.out.println("**************************");
            jijin.fundA();
            System.out.println("**************************");
            jijin.fundB();
            System.out.println("**************************");
            jijin.fundC();
            System.out.println("**************************");
            jijin.fundD();
            System.out.println("**************************");
        }
    }
```

输出的结果如下所示。

```
    **************************
```

```
购买国债
购买期货
**************************
购买国债
购买股票
**************************
购买期货
购买股票
**************************
购买国债
购买股票
购买期货
**************************
```

5．外观模式的优缺点分析

（1）优点。

外观模式通过提供一个统一的对外接口，一方面可以避免外部系统和子系统之间的直接联系，从而降低系统间的依赖程度；另一方面，如何外部系统想和子系统进行直接的交互，也可以绕过外观模式，这使得外部系统对子系统的使用非常的灵活。

（2）缺点。

外观模式对外部系统提供的接口是有限的，从这个角度上讲，是限制了外部系统对子系统调用的灵活性。

6．外观模式的实际应用简介

一般而言，外观模式使用于以下场合：

（1）为一个复杂的子系统提供一个简单的接口。子系统往往因为不断地演化而变得越来越复杂，使用外观模式可以保持外部系统对子系统调用的简洁性，而那些需要细节调用的用户却可以越过外观模式直接对子系统进行调用。

（2）引进外观模式可以将一个子系统和使用它的客户端及其他的子系统分离开来，这就提高了子系统的独立性和可移植性。

（3）在构建一个层次化结构的时候，可以使用外观模式定义每一个层次对外交互的接口。此时，层与层之间只需要通过外观进行通信，从而简化层与层之间的依赖关系。

外观模式是一种得到广泛应用的模式，例如我们熟知的 MVC 模式就采用了外观模式。在 MVC 架构模式中，每一层并不需要知道其他层次的细节，只是通过层与层之间的接口调用即可，这大大地方便了开发者。

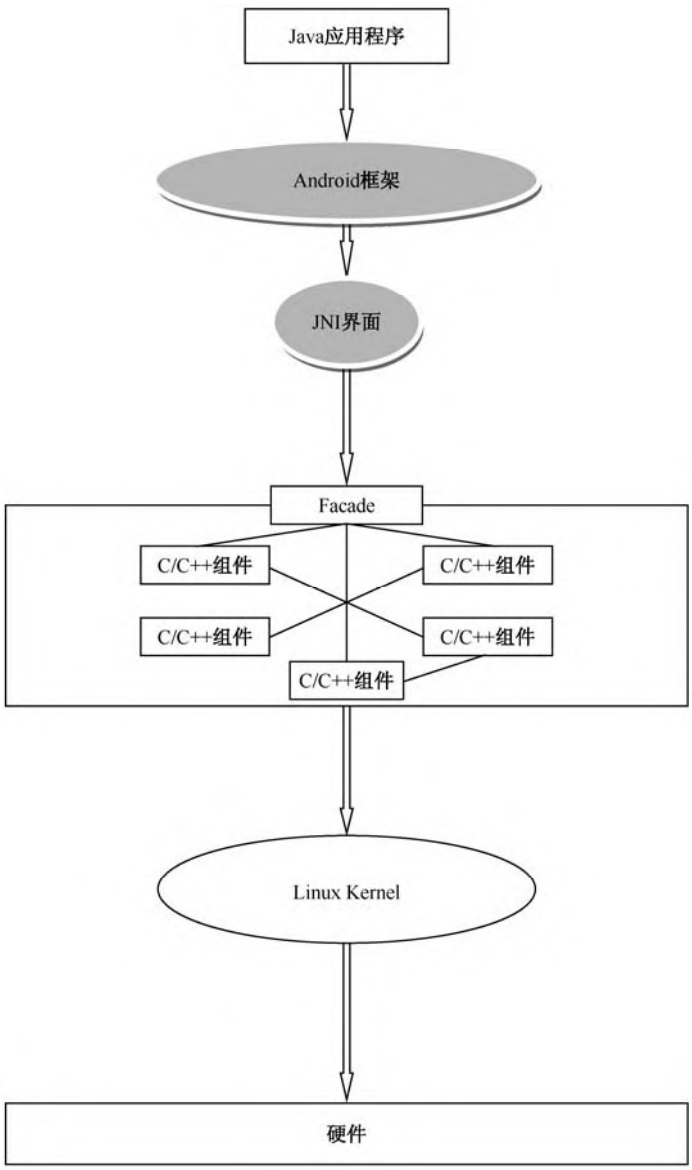
7．温馨提示

一般而言，外观模式是在复杂系统中为子系统提供方便一致的对外接口，但是这并不妨碍外部系统和子系统的直接交互，因为外部系统可以穿越外观而直接与子系统交互。

对于股票、期货、基金而言，基金就相当于它们的外观，而且还可以组合出多个不同的外观来供股民选购。

5.3 Facade 设计模式在 JNI 中的应用

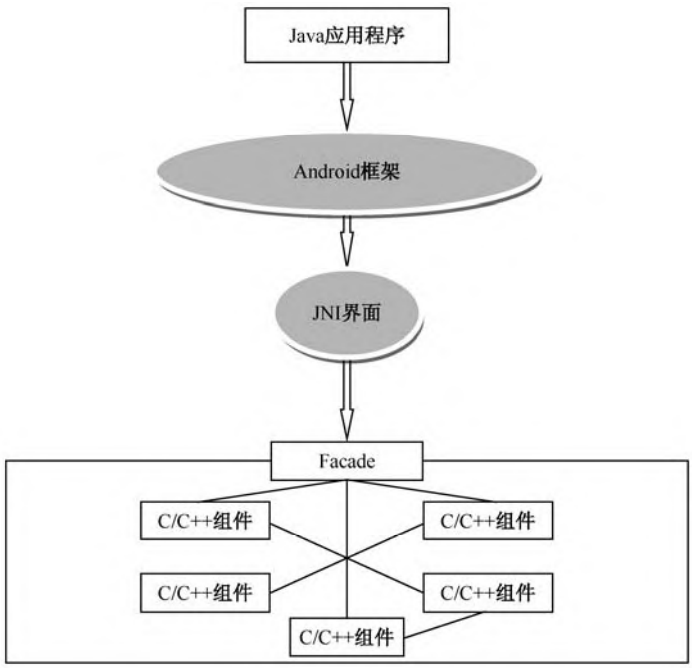
Facade 设计模式在 JNI 中的应用示意图如下图所示。



可以非常清晰地看出，Facade 设计模式提供了封装 C/C++组件的机制，提供本地代码对外服务的统一的接口，这样我们就非常容易根据硬件的实际情况修改内部的本地 C/C++代码，系统就变得“Easy to change”。

5.4　Facade 设计模式在 NDK 中的应用

Facade 设计模式在 NDK 中应用从本质上讲是 Facade 设计模式在 JNI 的应用，如下图所示。



5.5　NDK 的优势与不足

1．NDK 的优势

（1）通过 NDK 提供的一系列工具，可以让软件工程师更方便迅速地开发和调试具有本地 C/C++代码的应用程序。

（2）NDK 可以自动地将本地 so 库和 Java 代码打包到应用程序最终的 apk 包中，极大地提高了打包的效率。

（3）NDK 提供了 C 标准库（libc）、标准数学库（libm）、压缩库（libz）、Log 库（liblog）等，非常方便本地 C/C++代码的开发。

2．NDK 的不足

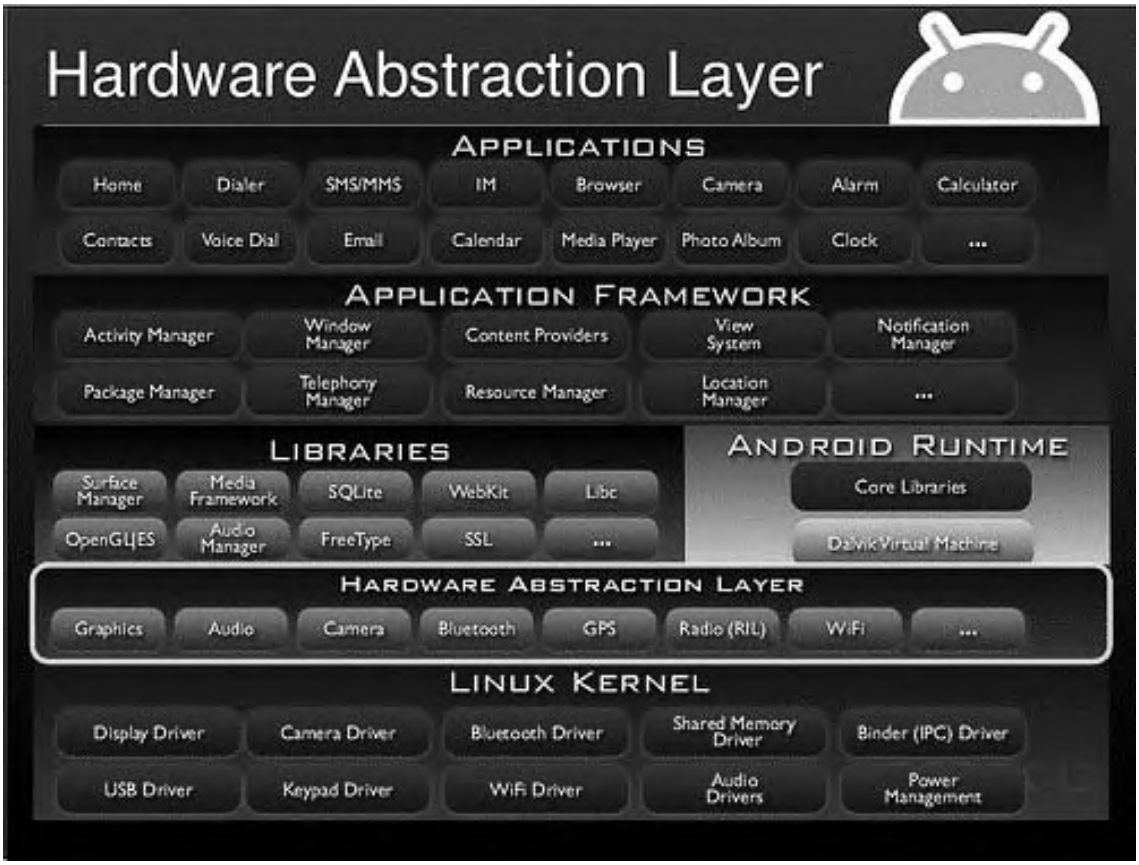
（1）支持的功能有限，可以调用的本地 C/C++库不是非常丰富。

（2）NDK 本地的 C/C++代码无法开发界面等。

（3）采用了 NDK 开发后，应用程序依赖于本地代码，如果要在其他不同的平台上运行，可能需要重新编写和生成 so 库文件。

5.6 NDK 与软/硬件整合

首先我们再来看一下 Android 完整的架构图，如下图所示。



Android 的核心特点在于透过 Android 平台发挥硬件的价值和优势，从这一点上出发，我们就非常容易理解为什么现在做 Android 智能手机挣钱的主要是 HTC、三星、摩托罗拉等，这几家公司均具有雄厚的硬件设计和研发实力，而且我们还可以发现，这几家的 Android 智能手机往往是价格偏高的，即走高质量、高价格之路。

为了传递和扩大硬件的价值和优势，Android 精心打造了 HAL 硬件框架和 Application Framework 软件框架。

NDK 通过 JNI 机制沟通了 Java 与本地的 C/C++代码，而本地的 C/C++代码有可以充分使用硬件的特性。这样开发的系统和程序就具有了独特的竞争力。

NDK 对于软/硬件整合具有重大的意义：

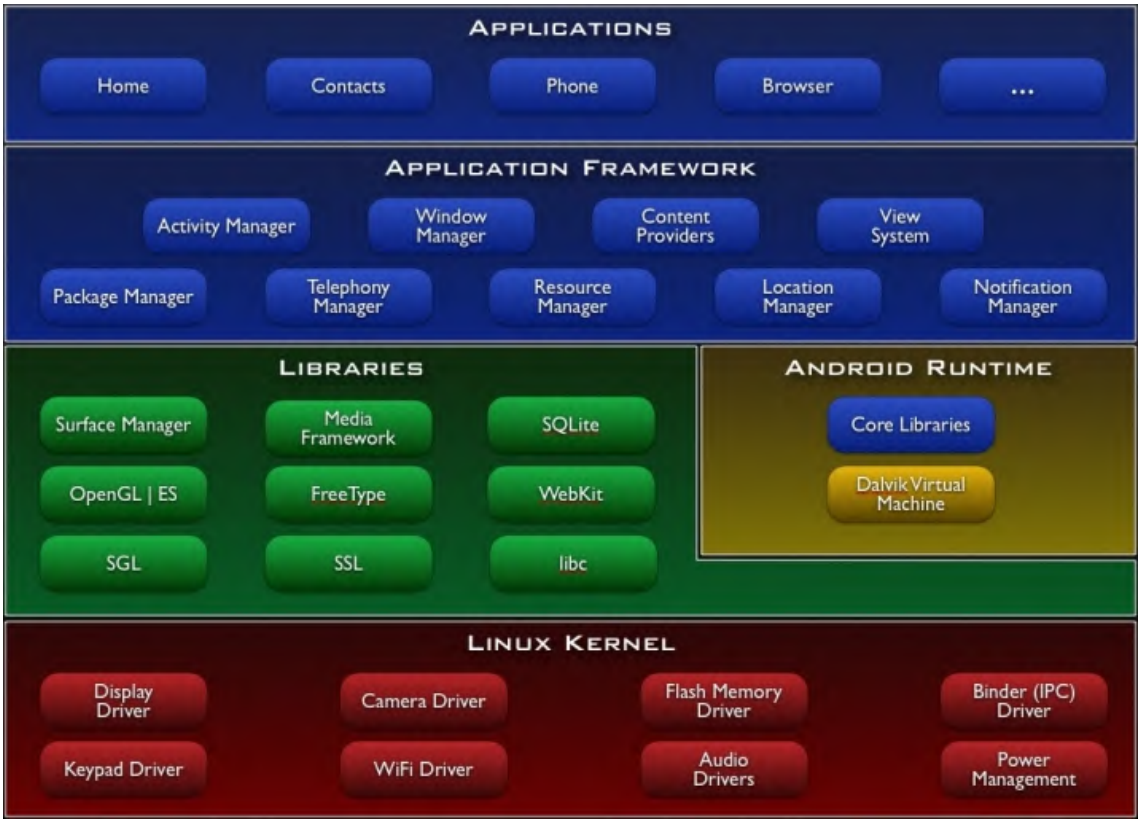
（1）把底层硬件的优势通过 JNI 机制传递给上层的应用程序。

（2）应用程序因为使用了 NDK 提供的硬件特性，就可以具有其他平台硬件没有的特性和优势，非常有利于在市场上占据优势地位。

（3）NDK 是应用软件和硬件相互促进的强有力的手段。

5.7　NDK 与云计算

Android SDK 的 DOC 中的 Android 架构图如下图所示。



一般而言，把云服务的 API 放在 Android 的 libraries 中是一个非常好的选择，这样，Android 的应用程序就可以通过 NDK 来使用云服务，这种方式无论是对于 Android 手机厂商还是对于云服务提供商及 Android 应用程序开发者而言都是相互促进的。

附录 A　Android UI 编程

UI 编程技术是开发过程中常用到的技术，下文对 ListView 异步加载技术，ListView 分页加载技术，Widget 编程，PopupWindow，自定义标题栏，多点触摸、缩放功能、拖拉功能这些可以极大地改善用户体验的技术进行了非常细致的讲解。

一、ListView 异步加载技术

在处理图片时经常会遇到需要从网络上下载图片；从网络上下载的图片需要在显示在本地的 ListView 中；图片会有很多；网络上的图片在服务端会不定期更新等问题，而在 Android 应用程序开发时，比较好的解决办法就是使用 ListView 异步加载技术。

ListView 的异步加载技术涉及到服务器端和 Android 客户端的内容，下面分别进行讲解。

1．服务器端

服务器端主要是提供众多的图片来供客户端使用，具体编程过程如下所示。

（1）创建一个动态的 Web 工程，工程名为 ServerForAsyncPictureLoading。

（2）创建图片所在的文件夹并在其中存放图片。

在 WebContent 目录下创建一个名为 images 的文件夹，然后在该文件夹下放置一些图片。

（3）创建包含访问图片信息的 XML 文件。

在 WebContent 目录下创建名为 listForPictures.xml 的 XML 文件，并在该 XML 文件中加入和图片路径相关的信息，完成后的具体内容如下所示。

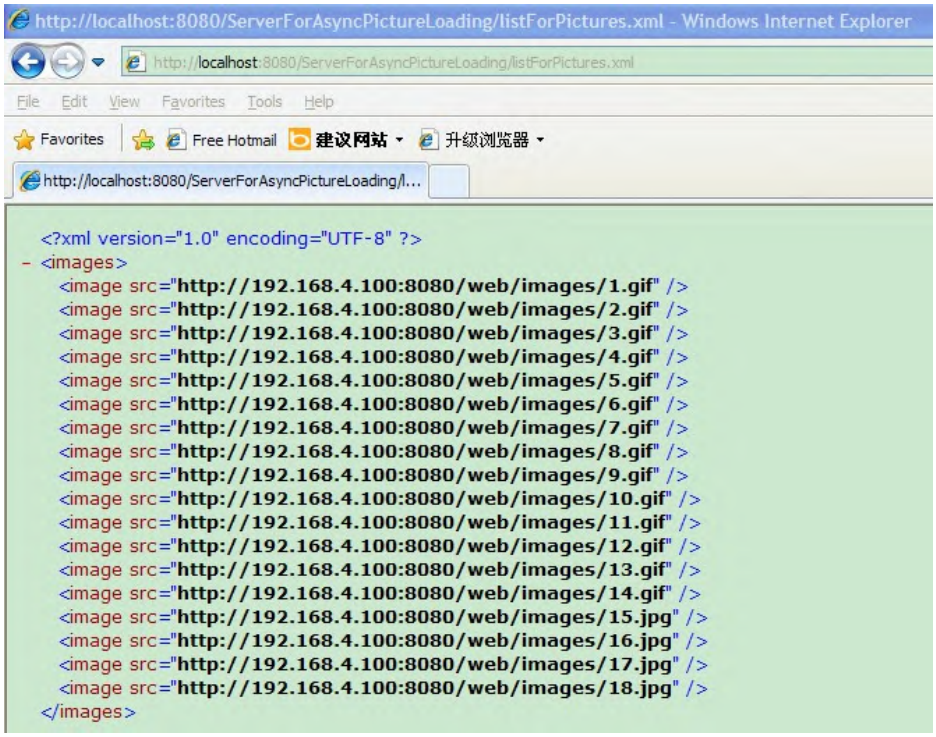
```
    <?xml version="1.0" encoding="UTF-8"?>
    <images>
        <image src="http://192.168.1.39:8080/ServerForAsyncPictureLoading/images/
1.gif"/>
        ...
        <image src="http://192.168.1.39:8080/ServerForAsyncPictureLoading/images/
18.jpg"/>
    </images>
```

（4）部署并测试创建的图片服务器。

发布并运行该 Web 工程，单击“Run As”→“Run on Server”，选择“Choose an existing server”，一切采用默认设置，单击“Finish”即可完成 Web 工程的发布和运行。

输入 http://localhost:8080/ServerForAsyncPictureLoading/listForPictures.xml 路径并发起请求，即可显示 XML 内容。

我们可以在 IE 浏览器中来测试图片 Web 服务器是否构建成功，如下图所示。



测试成功，至此，服务器端的工作完毕。

2．Android 客户端

（1）创建一个名为 AsyncLoadingPictures 的 Android 工程。

（2）创建界面。

界面是在 XML 文件中创建的，因为要用 ListView 来显示图片，所以我们需要修改 main.xml 文件的内容，修改后的代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ListView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/listView"
    />
</LinearLayout>
```

同时因为 ListView 中每个 Item 要显示一张图片，因此我们需要在 layout 目录下创建一个显示 item 的 XML 文件，名称为 list_item.xml。

在 list_item.xml 中只需要简单地显示一张图片即可，其内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  >
  <ImageView
  android:layout_width="50dp"
  android:layout_height="50dp"
  android:layout_marginLeft="5dp"
  android:id="@+id/imageView"
  />
</LinearLayout>
```

（3）创建 ListView 的适配器。

因为 ListView 需要显示内容，而要显示内容就需要内容适配器，我们建立 ImageAdapter 作为 ListView 的适配，ImageAdapter 需要继承自 BaseAdapter，编写其内容如下所示。

```
package com.wangjialin.asynloading.adapter;
import java.io.File;
import java.util.List;
import android.content.Context;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Environment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import com.wangjialin.asynloading.picuture.R;
public class ImageAdapter extends BaseAdapter{
    //存放图片路径的集合
    private List<String> data;
    //显示图片的 Item 的布局文件
    private int sourceid;
    //布局实例化类
    private LayoutInflater layoutInflater;
    //网络上的图片文件文件保存到的本地地点
    File saveDir;
    /**
     * 适配器的构造方法
     * @param data 图片的路径的集合
     * @param context 应用程序所在是上下文
     * @param sourceid 显示单张图片的布局文件
     */
    public ImageAdapter(List<String> data, Context context, int sourceid) {
        this.data = data;
        this.sourceid = sourceid;
```

```
        //实例化话布局文件的实例化器
        layoutInflater = LayoutInflater.from(context);
        //在 SDCard 的根目录下创建一个名称为“cache”的文件夹
        saveDir = new File(Environment.getExternalStorageDirectory(),"cache");
        //当该文件夹不存在时，创建这个文件夹
        if(!saveDir.exists()) saveDir.mkdirs();
    }
    @Override
    public int getCount() {
        //获得显示图片一种有多少张
        return data.size();
    }
    @Override
    public Object getItem(int position) {
        //获得特定位置图片的路径
        return data.get(position);
    }
    @Override
    public long getItemId(int position) {
        //获得图片所在的位置
        return position;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        //声明要一张图片
        ImageView imageView;
        //如果这张图片没有显示过或者没有缓存时
        if(convertView == null){
            //实例化该 Item 的布局文件
            convertView = layoutInflater.inflate(sourceid, null);
            //查询该布局文件中的图片显示控件并赋值给 imageView
            imageView = (ImageView) convertView.findViewById(R.id.imageView);
            //设置对 imageView 的引用，这样就可以重用了
            convertView.setTag(imageView);
        }else{
            //获得 ImageView
            imageView = (ImageView) convertView.getTag();
        }
        //显示图片
        showImageAccordingAsyncTask(imageView, data.get(position));
        //返回整个 Item 的视图
        return convertView;
    }
    /*
    private void showImage(final ImageView imageView, final String imagepath) {
        final Handler handler = new Handler(){
            public void handleMessage(Message msg) {
                imageView.setImageURI((Uri) msg.obj);//显示照片
            }
        };
```

```
            Runnable loadImageTask = new Runnable() {
                public void run() {
                    try {
                        Uri uri = ImageService.getImage(imagepath, saveDir);
                        handler.sendMessage(handler.obtainMessage(1, uri));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            };
            new Thread(loadImageTask).start();
        }
        */
        /**
         * 使用 AsyncTask 来显示图片
         * @param imageView
         * @param path
         */
        private void showImageAccordingAsyncTask(ImageView imageView, String path) {
            //实例化该 AsyncTask
            LoadImageTask task = new LoadImageTask();
            //参数传给了 doInBackground()
            task.execute(imageView, path);
        }
        private final class LoadImageTask extends AsyncTask<Object, Integer, Uri>{
            ImageView imageView;
            String imagepath;
            @Override
            protected Uri doInBackground(Object... params) {//耗时操作,运行在子线程
                //获得布局文件中 ImageView
                imageView = (ImageView) params[0];
                //获得图片的路径
                imagepath = (String) params[1];
                try {
                    //从网络上获得图片并存储在本地的 SDCard 中
                    return ImageService.getImage(imagepath, saveDir);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return null;
            }
            @Override
        protected void onPostExecute(Uri result) {//doInBackground()的结果传入该方
法，运行在 UI 线程
                imageView.setImageURI(result);//显示照片
            }
        }
    }
```

（4）编写业务类 ImageService。

ImageService 负责从网络上下载并在本地的 SDCard 中缓存这些图片。

编写 ImageService 业务代码，内容如下所示。

```
package com.wangjialin.asynloading.utils;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import org.xmlpull.v1.XmlPullParser;
import android.net.Uri;
import android.util.Xml;
public class ImageService {
    /**
     * 获取图片，具有缓存功能
     * @param path
     * @param saveDir
     * @return
     * @throws Exception
     */
    public static Uri getImage(String path, File saveDir) throws Exception{
        //通过 MD5 工具类来生成缓存的文件名
        String filename = MD5.getMD5(path) + path.substring(path.lastIndexOf("."));
        //在 SDCard 的 cache 目录下存储文件
        File imageFile = new File(saveDir, filename);
        //如果文件存在就直接返回在文件的 Uri
        if(imageFile.exists()){
            //返回文件的 Uri
            return Uri.fromFile(imageFile);
        }
        //通过文件的路径建立网络连接
        HttpURLConnection conn = (HttpURLConnection) new URL(path).openConnection();
        //设置超时时间为 5 秒钟
        conn.setConnectTimeout(5000);
        //设置"GET"的方式从网络上获取文件
        conn.setRequestMethod("GET");
        //如果返回的状态码为 200 则表示请求成功
        if(conn.getResponseCode() == 200){
            //把网络上指定路径的图片存放在本地的 SDCard 的 cache 目录中
            copyStream(conn.getInputStream(), new FileOutputStream(imageFile));
            //返回该图片在本地的 Uri
            return Uri.fromFile(imageFile);
        }
        return null;
    }
    /**
     * 把网络上特定的图片下载到本地
```

```
         * @param inputStream
         * @param outStream
         * @throws Exception
         */
        private static void copyStream(InputStream inputStream, OutputStream
outStream) throws Exception{
            //开辟 1KB 的缓存，以提高效率
            byte[] buffer = new byte[1024];
            int len = 0;
            //不断从网络上读取数据并保存到本地
            while((len = inputStream.read(buffer)) != -1){
                outStream.write(buffer, 0, len);
            }
            //关闭网络输入流
            inputStream.close();
            //刷新关闭本地文件输出流
            outStream.close();
        }
        /**
         * 从网络上获取要下载到本地的图片的信息
         * @return
         * @throws Exception
         */
        public static List<String> getImageInfos() throws Exception{
            //这是笔者服务器上图片的 XML 文件的路径
    String path = "http://125.34.55.167:8080/ServerForAsyncPictureLoading/listFor
Pictures.xml";
            //建立网络连接
            HttpURLConnection conn = (HttpURLConnection) new URL(path).openConnection();
            //设置超时时间为 5 秒钟
            conn.setConnectTimeout(5000);
            //通过 GET 的方式获取网络上的文件
            conn.setRequestMethod("GET");
            //如果返回码为 200 则表示请求成功
            if(conn.getResponseCode() == 200){
                //解析 XML 文件
                return parseXML(conn.getInputStream());
            }
            return null;
        }
        /**
         * 解析从网络上下载的 XML 文件
         * @param xml
         * @return
         * @throws Exception
         */
        private static List<String> parseXML(InputStream xml) throws Exception{
            //建立一个元素类型为 String 的 List 来存储解析的 XML 的内容
            List<String> paths = new ArrayList<String>();
            //创建一个 PULL 解析器
```

```
        XmlPullParser pullParser = Xml.newPullParser();
        //设置读取的输入流和输入流的编码
        pullParser.setInput(xml, "UTF-8");
        //获得文档开始节点
        int event = pullParser.getEventType();
        //循环读取输入流并不断解析
        while(event != XmlPullParser.END_DOCUMENT){
            if(event == XmlPullParser.START_TAG){
                //如果是 image 开头的则读取其中的图片的路径
                if("image".equals(pullParser.getName())){
                    paths.add(pullParser.getAttributeValue(0));
                }
            }
            //获得下一个 PULL 时间
            event = pullParser.next();
        }
        //返回从 XML 文件中解析的图片路径
        return paths;
    }
}
```

在 ImageService 中还用到了 PULL 解析器来解析 XML 文件。

3．Pull 解析 XML

（1）新建类，我们还延续上个项目的内容，在 service 层新建类 PullPersonService。

（2）编写 PullPersonService 的业务代码。

```
package com.wangjialin.service;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import org.xmlpull.v1.XmlPullParser;
import android.util.Xml;
import com.wangjialin.domain.Person;
public class PullPersonService {
    public static List<Person> getPersons(InputStream inStream) throws Exception{
        Person person = null;
        List<Person> persons = null;
        XmlPullParser pullParser = Xml.newPullParser();
        pullParser.setInput(inStream, "UTF-8");
        int event = pullParser.getEventType();//触发第一个事件
        while(event!=XmlPullParser.END_DOCUMENT){
            switch (event) {
            case XmlPullParser.START_DOCUMENT:
                persons = new ArrayList<Person>();
                break;
            case XmlPullParser.START_TAG:
                if("person".equals(pullParser.getName())){
                    int id = new Integer(pullParser.getAttributeValue(0));
                    person = new Person();
```

```
                    person.setId(id);
                }
                if(person!=null){
                    if("name".equals(pullParser.getName())){
                        person.setName(pullParser.nextText());
                    }
                    if("age".equals(pullParser.getName())){
                        person.setAge(new Short(pullParser.nextText()));
                    }
                }
                break;
            case XmlPullParser.END_TAG:
                if("person".equals(pullParser.getName())){
                    persons.add(person);
                    person = null;
                }
                break;
            }
            event = pullParser.next();
        }
        return persons;
    }
}
```

说明：

- int eventType = parser.getEventType()是 Pull 解析器的第一个事件。读者可以看到，这个方法的返回值是 int 类型的，这就是前面所提到的，Pull 解析器返回的是一个数字，类似于一个信号。Pull 解析器已经定义了这 5 个常量，而且对于事件，仅仅只有这 5 个。XmlPullParser.START_DOCUMENT：开始解析；XmlPullParser.START_TAG：开始元素；XmlPullParser.TEXT：解析文本；XmlPullParser.END_TAG：结束元素；XmlPullParser.END_DOCUMENT：结束解析。
- parser.getEventType()触发了第一个事件，根据 XML 的语法，也就是从它开始了解析文档。要通过 parser 中最重要的方法 parser.next()触发下一个事件。注意：该方法是有返回值的，在 Pull 触发下一个事件的同时，我们也获得该事件的“信号”。通过获得的信号进行 switch 操作。
- parser.getAttributeValue()：获得相应属性的值。它有两种形式，可以通过属性的索引，也可以通过（命名空间，属性名）进行索引。

（3）编写单元测试方法测试 PullPersonService。

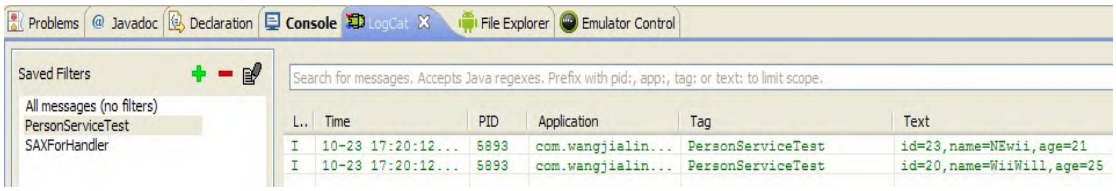
```
package com.wangjialin.xml;
import java.io.InputStream;
import java.util.List;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import android.test.AndroidTestCase;
```

```
    import android.util.Log;
    import com.wangjialin.domain.Person;
    import com.wangjialin.service.DOMPersonService;
    import com.wangjialin.service.PullPersonService;
    import com.wangjialin.service.SAXForHandler;
    public class PersonServiceTest extends AndroidTestCase {
        private static final String TAG = "PersonServiceTest";
        public void testSAXGetPersons() throws Throwable{
            InputStream inputStream = this.getClass().getClassLoader().
                    getResourceAsStream("wangjialin.xml");
            SAXForHandler saxForHandler = new SAXForHandler();
            SAXParserFactory spf = SAXParserFactory.newInstance();
            SAXParser saxParser = spf.newSAXParser();
            saxParser.parse(inputStream, saxForHandler);
            List<Person> persons = saxForHandler.getPersons();
            inputStream.close();
            for(Person person:persons){
            Log.i(TAG, person.toString());
            }
        }
        public void testDOMgetPersons() throws Throwable{
            InputStream inStream = this.getClass().getClassLoader().
                    getResourceAsStream("wangjialin.xml");
            List<Person> persons = DOMPersonService.getPersons(inStream);
            for(Person person : persons){
                Log.i(TAG, person.toString());
            }
        }
        public void testPullgetPersons() throws Throwable{
            InputStream inStream = this.getClass().getClassLoader().getResourceAs
Stream("wangjialin.xml");
            List<Person> persons = PullPersonService.getPersons(inStream);
            for(Person person : persons){
                Log.i(TAG, person.toString());
            }
        }
    }
```

（4）进行测试。

右键单击“PersonServiceTest”的 Outline 视图中的“testPullgetPersons ()”，选择“Run As”下的“Android JUnit Test”自动部署应用程序（此时如果模拟器未启动的话，则会自动启动模拟器来部署应用程序）并进行单元方法测试。

此时单击 LogCat 中的“PersonServiceTest”，显示出解析的数据，如下图所示。



总结：

仅仅是 5 个事件和几个简单的方法，就能解析我们需要的 XML。通过编写简单的 Pull 解析器应用，读者应该也能体会这个默默无闻的高手的能力了吧。它是一个非常优秀的技术，有着 SAX 解析器的性能，在编码方式上，又没有 SAX 那么繁琐。同时没有 DOM 那种繁琐的树状结构。

二、ListView 分页加载技术

如果在 ListView 中显示很多数据，例如，在 ListView 中要显示 500 条数据，此时就需要使用 ListView 的分页加载技术了，因为如果一次性把数据加载进来的话，会很消耗时间，导致用户长期的等待和不友好的用户体验。

下面我们来具体完成整个程序的编写。

（1）创建一个名为 BatchLoadingListView 的 Android 工程。

（2）编写程序的界面。

界面是在 main.xml 中编写，接下来编写 ListView 中每个 Item 的显示方法，list_item.xml 具体内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  >
  <TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textSize="18sp"
  android:id="@+id/name"
  />
</LinearLayout>
```

为了避免不必要的复杂，我们只是让每个 Item 显示文本内容。同时我们需要加载数据的状态显示界面，在这里我们就显示一个圆形的进度图和说明加载的文字，该界面写在了 loading.xml 中，如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:gravity="center"
    >
    <ProgressBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyle"
    />
    <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="22sp"
    android:text="数据正在加载，请稍等..."
    />
</LinearLayout>
```

（3）编写业务类。

业务类主要是数据以及对数据的操作，一般而言，需要分页加载的数据要么是存放的本地中，要么是存放在网络中。为方便，把这些数据放在一个 List 中，业务类 PersonService 的内容如下所示。

```
package com.wangjialin.service;
import java.util.ArrayList;
import java.util.List;
public class PersonService {
    /**
     * 分页数据的来源
     * @param offset
     * @param maxResult
     * @return
     */
public static List<String> getNames(int offset, int maxResult){
    List<String> names = new ArrayList<String>();
    names.add("王家林的第一本 Android 书籍：大话企业级 Android 应用开发实战");
    ... //省略部分代码
    names.add("王家林的第二十本 Android 书籍：细说 Android 的传感器");
    return names;
    }
}
```

可以发现，在调用此方法的业务类的 getNames 方法时，需要传入前面跳过的记录数以及该页的最大记录数。为了简化工作，每一页都返回了同样的数据。

（4）编写控制器类 Activity。

```
package com.wangjialin.batchloading;
import java.util.ArrayList;
import java.util.List;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.AbsListView;
import android.widget.AbsListView.OnScrollListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import com.wangjialin.service.PersonService;
public class BatchLoadingListViewActivity extends Activity {
    //声明 ListView
    ListView listView;
    //声明一个数组适配器
   private ArrayAdapter<String> adapter;
   //声明并实例化一个接受单页数据的集合
   List<String> data = new ArrayList<String>();
   //声明加载的视图
   View loadingView;
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       //实例化加载视图
       loadingView = getLayoutInflater().inflate(R.layout.loading, null);
       //实例数组适配器，此时数据为空
       adapter = new ArrayAdapter<String>(this, R.layout.list_item,R.id.name, data);
       //实例化 ListView
       listView = (ListView) this.findViewById(R.id.listView);
       //设置 listView 的监听器
       listView.setOnScrollListener(new ScrollListener());
       //设置 listView 的适配器
       listView.setAdapter(adapter);
   }
   //声明下一页，在默认情况下下一页的值为 1，即下一页为第二页
   int nextpage = 1;
   //声明当前页的索引，默认赋值为 0，即第一页的索引为 0
   int currentpage = 0;
   //listView 的监听器的实现
   private final class ScrollListener implements OnScrollListener{
    //设置每页的数据的 Item 数据为 20 条
    private int number = 20;
        @Override
        public void onScrollStateChanged(AbsListView view, int scrollState) {
            Log.i("Test", "onScrollStateChanged(scrollState="+ scrollState + ")");
        }
        @Override
        public void onScroll(AbsListView view, int firstVisibleItem,
                int visibleItemCount, int totalItemCount) {
```

```
                Log.i("Test", "onScroll(firstVisibleItem="+ firstVisibleItem+
                        "visibleItemCount="+visibleItemCount+   "totalItemCount=
"+totalItemCount + ")");
                //listView.getLastVisiblePosition()
                if(firstVisibleItem + visibleItemCount == totalItemCount){//下一页
                    if(totalItemCount>0) nextpage = totalItemCount / number + 1;

                    if(currentpage!=nextpage){
                        listView.addFooterView(loadingView);//显示数据正在加载
                        //开线程{得到数据，然后发送消息}
                handler.sendMessageDelayed(handler.obtainMessage(1,  totalItemCount),
3000);
                        currentpage = nextpage;
                    }
                }
            }
        }
        Handler handler = new Handler(){
            public void handleMessage(Message msg) {
                //把数据加载进来
                data.addAll(PersonService.getNames((Integer)msg.obj, 20));
                //通知 ListView 刷新数据列表显示
                adapter.notifyDataSetChanged();
                //当数据加载完后，删除提示
                listView.removeFooterView(loadingView);
            }
        };
    }
```

（5）测试 ListView 分页加载项目。

发布后的初始页面如下（左）图所示，数据加载完成后，如下（中）图所示，当数据位于单页的底部时，会显示加载下一页，如下（右）图所示。

对于 ListView 分页加载技术测试成功。

（6）调试。

为了读者更好地理解代码的工作原理和过程，我们采用调试的方式来观察 ListView 分页加载技术是如何运行的。

首先在 handler 中打上断点，如下图所示。

```
77      Handler handler = new Handler(){
78          public void handleMessage(Message msg) {
79              //把数据加载进来
80              data.addAll(PersonService.getNames((Integer)msg.obj, 20));
81              //通知ListView刷新数据列表显示
82              adapter.notifyDataSetChanged();
83              //当数据加载完后，删除提示
84              listView.removeFooterView(loadingView);
85          }
86      };
```

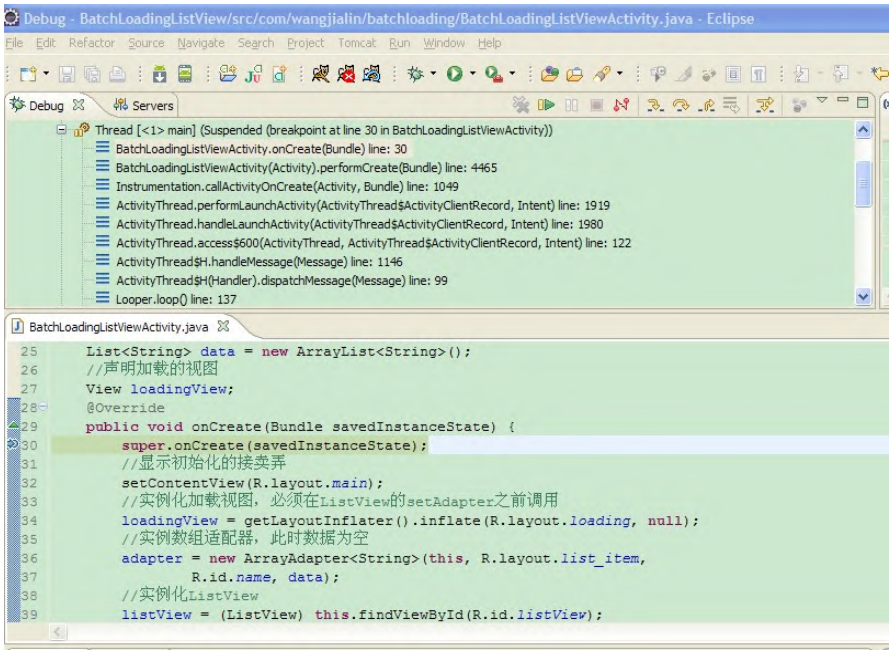
同时在 onCreate 方法中打上断点，如下图所示。

```
28      @Override
29      public void onCreate(Bundle savedInstanceState) {
30          super.onCreate(savedInstanceState);
31          //显示初始化
32          setContentView(R.layout.main);
```

最后在 listView 的监听器的 onScroll 方法中打上断点，如下图所示。

```
58      @Override
59      public void onScroll(AbsListView view, int firstVisibleItem,
60              int visibleItemCount, int totalItemCount) {
61          Log.i("Test", "onScroll(firstVisibleItem="+ firstVisibleItem+
62                  "visibleItemCount="+visibleItemCount+ "totalItemCount=
63          //listView.getLastVisiblePosition()
```

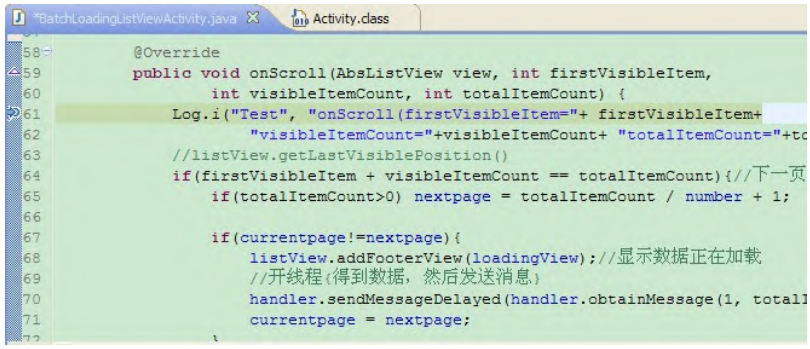
以调试模式运行程序，弹出提示下窗口后单击“Yes”即可进入调试视图，如下图所示。



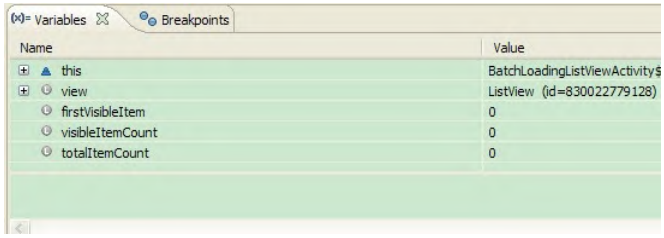
可以看出，此时运行到了 onCreate 方法中。

单步调试，连续按下三次快捷键 F6，进入如下图所示的视图。

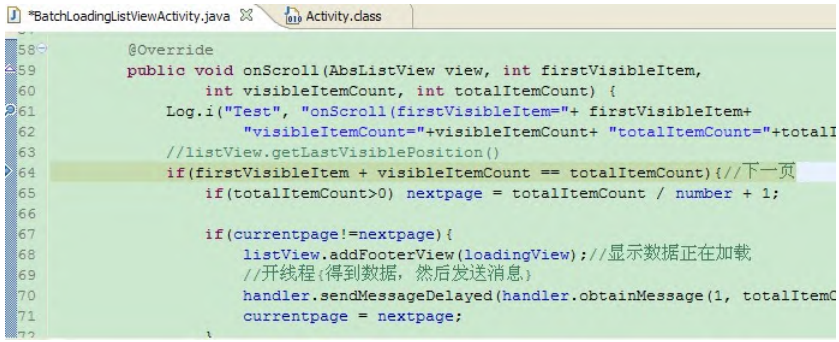
继续按下 F6 快捷键，一直运行到如下图所示区域。



此时可以发现 firstVisibleItem 、visibleItemCount、totalItemCount 的值皆为 0，如下图所示。

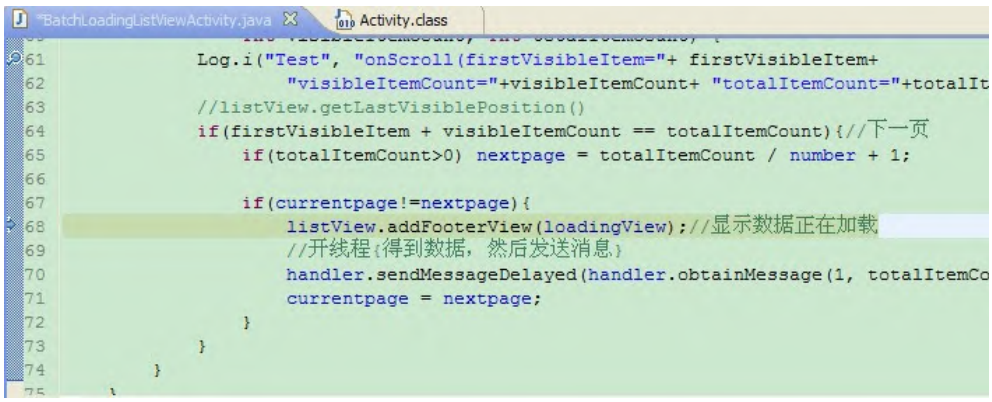


按下快捷键 F6，单步运行到如下图所示区域。

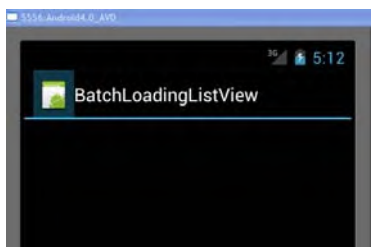


因为 firstVisibleItem、visibleItemCount、totalItemCount 的值皆为 0，所以条件判断是成立，也就进入了条件语句内，currentpage 的初始值为 0，而 nextpage 的初始值为 1，所示一下代码条件判断成立：if(currentpage!=nextpage)。

此时进入判断成功后的代码区域，如下图所示。

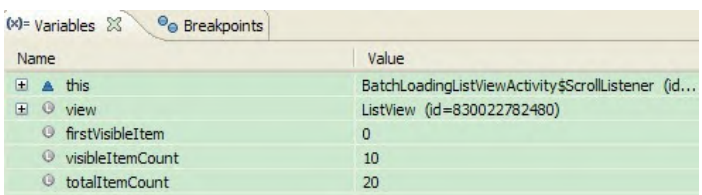


在执行以下代码之前的程序运行效果如下图所示。



按下快捷键 F8，此时程序跳转到如下图所示区域。

一直按下 F8 快捷键，程序运行到如下图所示状态。



此时的程序运行效果如下（左）图所示，当需要加载下一页的数据时，如下（右）图所示。

此时代码运行的区域如下图所示。

```
74          }
75      }
76
77      Handler handler = new Handler(){
78          public void handleMessage(Message msg) {
79              //把数据加载进来
80              data.addAll(PersonService.getNames((Integer)msg.obj, 20));
81              //通知ListView刷新数据列表显示
82              adapter.notifyDataSetChanged();
83              //当数据加载完后，删除提示
84              listView.removeFooterView(loadingView);
85          }
86      };
87  }
```

我们可以发现，此时 obj 的值是 20。

因为此时要加载第二页的数据，读者可以按照上述的思路继续调试下去。

三、Widget 编程实战

1．Widget 应用场景

在 Android 屏幕上长按会弹出对话框如下（左）图所示，单击弹出的对话框中的“Widgets”，弹出系统上安装的 Widget 的对话框，如下（中）图所示。单击其中的一个 Widget 即可把该 Widget 加入到桌面上，例如，选择“Analog Clock”，则会在 Android 的 Home 屏幕上显示并运行该 Widget，如下（右）图所示。

可以看到该 Widget 是一个时钟。

我们需要编写 Widget 的原因在于：

- 提供 Android 的程序的快捷桌面方式；
- 一些需要随时刷新的信息，如新闻、天气、时钟等，需要以 Widget 的方式呈现给用户。

2．Widget 编程实战

（1）建立一个名称为 Widget 的 Android 工程。

（2）建立一个名称为 TimeWidgetProvider 的 Widget，所有 Widget 的实现都必须继承 AppWidgetProvider。

在其中首先实现一些生命周期方法，代码如下所示。

```
    package com.wangjialing.widgetsuit;
    import android.appwidget.AppWidgetManager;
    import android.appwidget.AppWidgetProvider;
    import android.content.Context;
    public class TimeWidgetProvider extends AppWidgetProvider {
        @Override
        public void onDeleted(Context context, int[] appWidgetIds) {
            super.onDeleted(context, appWidgetIds);
        }
        @Override
        public void onEnabled(Context context) {//第一次添加该部件时调用
        }
        @Override
        public void onDisabled(Context context) {
        }
        @Override
        public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
        }
    }
```

我们看一下 AppWidgetProvider 的继承结构，如下图所示。

可以看出 AppWidgetProvider 继承自 BroadcastReceiver，也就是说 AppWidgetProvider 继承者也是一种 BroadcastReceiver，源码如下所示。

```
public class AppWidgetProvider extends BroadcastReceiver {
```

（3）编写 AndroidManifest.xml 中关于 Widget 的配置文件，如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.wangjialing.widgetsuit"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk android:minSdkVersion="8" />
   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:label="@string/app_name"
         android:name=".WidgetActivity" >
         <intent-filter >
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <receiver android:name=".TimeWidgetProvider" >
          <intent-filter> <action android:name="android.appwidget.action.
APPWIDGET_UPDATE" />
          </intent-filter>
          <meta-data  android:name="android.appwidget.provider"  android:
resource="@xml/time_widget" />
       </receiver>
   </application>
</manifest>
```

我们在 receiver 节点中声明了前面建立的 TimeWidgetProvider ，同时在 meta-data 中通过 mete-data 中的 android:resource 属性来声明了 Widget 的属性配置文件，属性配置文件会声明 Widget 在窗口上显示时候的长、宽、背景颜色等信息。

（4）编写 Widget 的配置文件。

在 AndroidManifest.xml 中声明关于 Widget 的配置文件的信息如下所示。

```
<meta-data android:name="android.appwidget.provider" android:resource="@xml/
```

```
time_widget" />
```

因此，我们需要在在 res 目录下新建一个 xml 目录，并在 xml 目录中建立我们的 time_widget.xml 配置文件。

编写 Widget 的属性配置文件 time_widget.xml，内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
        android:minWidth="294dp"
        android:minHeight="72dp"
        android:updatePeriodMillis="0"
        android:initialLayout="@layout/timewidget"
        >
    </appwidget-provider>
```

在配置文件中声明了 Widget 的最小宽度和高度，此时设置更新频率为 0，即不更新，我们采用了在 Service 中更新的方式。同时在配置文件也指定了 Widget 显示时的初始化配置文件为 layout 下的 timewidget.xml。

（5）编写 Widget 的初始化布局文件。

在 Widget 的属性配置文件 time_widget.xml 中指定了 Widget 的初始布局文件为 timewidget.xml。

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      >
      <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="22sp"
      android:text="This is widget"
      android:textColor="#ffffff"
      android:id="@+id/title"
      />
    </LinearLayout>
```

可以看出，我们只是在 Widget 上简单地显示了一个文本界面。

（6）编写业务代码。

TimeWidgetProvider 的业务代码如下所示。

```
    package com.wangjialin.widgetsuit;
    import java.text.SimpleDateFormat;
    import java.util.Date;
```

```
import java.util.Timer;
import java.util.TimerTask;
import android.app.PendingIntent;
import android.app.Service;
import android.appwidget.AppWidgetManager;
import android.content.ComponentName;
import android.content.Intent;
import android.os.IBinder;
import android.widget.RemoteViews;
/**
 * 更新 Widget 的服务
 * @author Android
 */
public class UpdateService extends Service {
    //声明一个定时器
    private Timer timer;
    @Override
    public void onCreate() {
        super.onCreate();
        //实例化定时器为后台线程
        timer = new Timer(true);
        //把任务加入到定时器上，并且设置立即启动该任务，每隔 1 秒钟就执行一次
        timer.schedule(new UpdateTask(), 0, 1000);
    }
    /**
     * 定义定时器任务
     * @author Android
     */
    private final class UpdateTask extends TimerTask{
        //定时器任务运行时会自动运行 run 方法
        public void run() {
            //格式化时间
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
            //获得 Widget 的布局文件
            RemoteViews remoteViews = new RemoteViews(getPackageName(),
R.layout.timewidget);
            //设置 Widget 布局文件中的文本控件显示的内容为格式化后的当前时间
            remoteViews.setTextViewText(R.id.title, dateFormat.format(new Date()));
            //创建一个启动 WidgetActivity 的意图
           Intent intent = new Intent(getApplicationContext(), WidgetActivity.
class);
            //以我们创建的意图为基础创建一个未来执行的意图
            PendingIntent pendingIntent = PendingIntent.getActivity
(getApplicationContext(), 100, intent, 0);
            //设置当文本控件被单击时执行未来的意图
            remoteViews.setOnClickPendingIntent(R.id.title, pendingIntent);
            //获得 Widget 管理器
            AppWidgetManager appWidgetManager = AppWidgetManager.getInstance
(getApplicationContext());
```

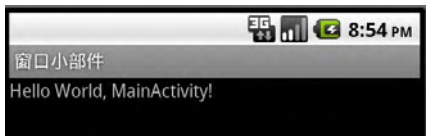
```
        //更新 TimeWidgetProvider 这个 Widget，并指定通过 remoteViews 显示 Widget 新的内容
                appWidgetManager.updateAppWidget(new                ComponentName
(getApplicationContext(), TimeWidgetProvider.class), remoteViews);
            }
        }
        /**
         * 当所有的 TimeWidgetProvider 类型的 Widget 都从 Home 界面上删除
         * 时停止定时器的工作，即停止更新 Widget 的内容
         */
        public void onDestroy() {
            super.onDestroy();
            if(timer!=null){
                timer.cancel();
                timer = null;
            }
        }
        @Override
        public IBinder onBind(Intent intent) {
                    return null;
        }
    }
```

此时我们修改一下 AndroidManifest.xml，并把 UpdateService 加入其中，如下所示。

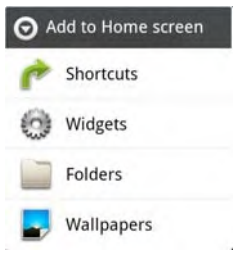
```
    <?xml version="1.0" encoding="utf-8"?>
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
         package="com.wangjialin.widgetsuit"
         android:versionCode="1"
         android:versionName="1.0">
        <application android:icon="@drawable/icon" android:label="@string/app_name">
            <activity android:name=".WidgetActivity"
                      android:label="@string/app_name">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
             <!-- 定义的 Widget -->
             <receiver android:name=".TimeWidgetProvider" >
                <intent-filter>
                   <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
                </intent-filter>
              <meta-data android:name="android.appwidget.provider" android:resource=
"@xml/time_widget" />
            </receiver>
            <!-- 定义的更新 Widget 的服务 -->
            <service android:name=".UpdateService"/>
        </application>
        <uses-sdk android:minSdkVersion="8" />
    </manifest>
```

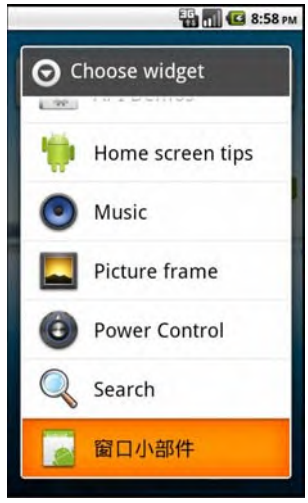
（7）测试 Widget。

安装并运行程序，初始界面如下图所示。

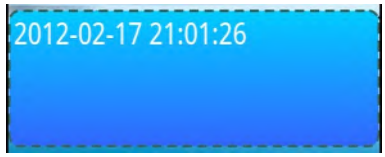


此时进入 Home 程序的主界面，并长按屏幕，弹出如下（左）图所示的对话框。选择“Widgets”，在弹出的对话框中可以看到我们创建的 Widget 已经被自动注册进系统，如下（右）图所示。





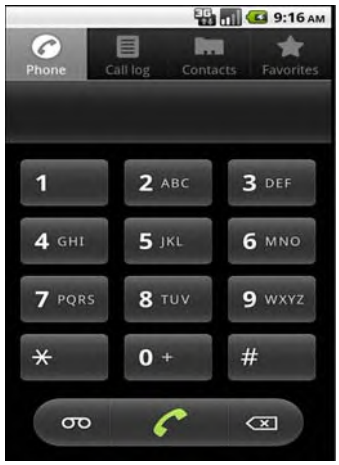
选中“窗口小部件”，此时 Home 的窗口如下（左）图所示，如我们所料，显示了我们定义的 Widget，同时每秒钟更新一次，不断更新效果如下图（右）所示。

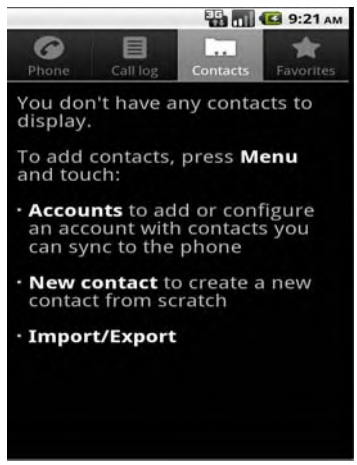


四、自定义 TabHost

1．Tabhost 的应用场景

TabHost 一个比较经典的应用是电话和通讯录程序，如下（左）图所示，我们可以切换不同的 Tab 页面，如切换到联系人列表页面，如下（右）图所示。





TabHost 在 Android 的应用程序开发中有着非常广泛的应用，尤其适用于要在一个 Activity 中展示多个相对独立的功能选项的情况。

2．自定义 TabHost 编程

（1）建立一个名称为 MyTabHost 的 Android 工程。

（2）构建界面。

主界面 main.xml 的内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/tabhost"
    >
    <LinearLayout
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical"
    >
     <!-- TabWidget 的 id 必须为"@android:id/tabs",
     因为 Android 框架的源代码会查询该 ID -->
       <TabWidget
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@android:id/tabs"
       />
```

```
        <!-- 包含 Tab 的具体内容必须是 FrameLayout，
        FrameLayout 的 id 必须为"@android:id/tabcontent"，
          因为 Android 框架的源代码会查询该 ID -->
        <FrameLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@android:id/tabcontent"
        >
          <!-- 第一个标签页 -->
          <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/page1"
            >
            <TextView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="一个标签页"
            />
          </LinearLayout>
          <!-- 第二个标签页 -->
           <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/page2"
            >
            <TextView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="第二个标签页"
            />
          </LinearLayout>
          <!-- 第三个标签页 -->
           <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/page3"
            >
             <TextView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="第三个标签页"
            />
          </LinearLayout>
        </FrameLayout>
      </LinearLayout>
    </TabHost>
```

需要特别强调的是，按照 Android 框架源代码的要求，必须做到：TabWidget 的 id 设置为@android:id/tabs。包含每个 Tab 选项内容的容器必须为 FrameLayout。FrameLayout 的 id 必须

设置为@android:id/tabcontent

（3）编写控制器代码。

控制器为 MyTabHostActivity，其具体内容如下所示。

```
package com.wangjialin.tab;
import android.app.Activity;
import android.os.Bundle;
import android.os.Debug;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TabHost;
import android.widget.TabHost.TabSpec;
import android.widget.TextView;
public class MyTabHostActivity extends Activity {
    //声明一个 TabHost 容器
    private TabHost tabHost;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //方法日志记录，默认的日志文件会存放在 SDCard 下
        Debug.startMethodTracing("MyTabHost");
        //实例化 TabHsot
        tabHost = (TabHost) this.findViewById(R.id.tabhost);
        //初始化设置 TabHost，在该方法的内部会检查 TabWidget 的 id 设置
        tabHost.setup();
        //第一个 Tab 选项
        TabSpec tabSpec1 = tabHost.newTabSpec("page1");
        tabSpec1.setIndicator(createTabView("第一页"));
        tabSpec1.setContent(R.id.page1);
        tabHost.addTab(tabSpec1);
        //第二个 Tab 选项
        TabSpec tabSpec2 = tabHost.newTabSpec("page2");
        tabSpec2.setIndicator(createTabView("第二页"));
        tabSpec2.setContent(R.id.page2);
        tabHost.addTab(tabSpec2);
        //第三个 Tab 选项
        TabSpec tabSpec3 = tabHost.newTabSpec("page3");
        tabSpec3.setIndicator(createTabView("第三页"));
        tabSpec3.setContent(R.id.page3);
        tabHost.addTab(tabSpec3);
        //设置打开 TabHost 时的默认的显示 Tab 页面
        tabHost.setCurrentTab(0);
    }
    @Override
    protected void onStop() {
    //停止往自定义的日志文件中记录日志
    Debug.stopMethodTracing();
```

```
            super.onStop();
        }
        /**
         * 自定义 Tab 的标题视图
         * @param title
         * @return
         */
        private View createTabView(String title){
            //声明并实例化一个线性布局
            LinearLayout linearLayout = new LinearLayout(this);
            //设置线性布局为垂直方向
            linearLayout.setOrientation(LinearLayout.VERTICAL);
            //线性布局的背景颜色为白色
            linearLayout.setBackgroundColor(0xFFFFFF);
            //声明并实例化一个 TextView
            TextView textView = (TextView) new TextView(this);
            //设置 TextView 显示的内容为传递进来的字符串
            textView.setText(title);
            //设置填充方式为 FILL_PARENT
            ViewGroup.LayoutParams params = new ViewGroup.LayoutParams
(ViewGroup.LayoutParams.FILL_PARENT,
                    ViewGroup.LayoutParams.FILL_PARENT);
            //把 TextView 按照指定的方式加入线性布局，并显示指定的内容
            linearLayout.addView(textView, params);
        return linearLayout;
        }
    }
```

（4）测试程序。

运行程序，效果如下图所示。

运行成功。

3．TabHost 调试

为了清晰地观察 TabHost 的运行路径和原理，下面我们一起来剖析一下 TabHost 的源代码。

首先来看一下 TabHost 的 setUp 方法，因为 MyTabHostActivity 调用了该方法。

```
    //实例化 TabHsot
        TabHost = (TabHost) this.findViewById(R.id.tabhost);
        //初始化设置 TabHost，在该方法的内部会检查 TabWidget 的 ID 设置
        tabHost.setup();
```

进入 setup 方法内部：

```
    public void setup() {
```

```
        mTabWidget = (TabWidget) findViewById(com.android.internal.R.id.tabs);
        if (mTabWidget == null) {
            throw new RuntimeException(
    "Your TabHost must have a TabWidget whose id attribute is 'android.R.id.tabs'");
        }
        mTabKeyListener = new OnKeyListener() {
            public boolean onKey(View v, int keyCode, KeyEvent event) {
                switch (keyCode) {
                    case KeyEvent.KEYCODE_DPAD_CENTER:
                    case KeyEvent.KEYCODE_DPAD_LEFT:
                    case KeyEvent.KEYCODE_DPAD_RIGHT:
                    case KeyEvent.KEYCODE_DPAD_UP:
                    case KeyEvent.KEYCODE_DPAD_DOWN:
                    case KeyEvent.KEYCODE_ENTER:
                        return false;
                }
                mTabContent.requestFocus(View.FOCUS_FORWARD);
                return mTabContent.dispatchKeyEvent(event);
            }
        };
        mTabWidget.setTabSelectionListener(new TabWidget.OnTabSelectionChanged() {
            public void onTabSelectionChanged(int tabIndex, boolean clicked) {
                setCurrentTab(tabIndex);
                if (clicked) {
                    mTabContent.requestFocus(View.FOCUS_FORWARD);
                }
            }
        });
        mTabContent = (FrameLayout) findViewById(com.android.internal.R.id.tabcontent);
        if (mTabContent == null) {
            throw new RuntimeException(
                    "Your TabHost must have a FrameLayout whose id attribute is "
                            + "'android.R.id.tabcontent'");
        }
    }
```

源代码中非常明确地指出，如果是通过 findViewById 的方式获得 TabHost 的，就必须调用 setup 方法。

在 setup 方法内部首先通过下述代码来检查 TabWidget 的 id 是否是按照系统的规定进行设置的：

```
mTabWidget = (TabWidget) findViewById(com.android.internal.R.id.tabs);
```

这就是为什么必须把 TabWidget 的 id 设置为@android:id/tabs 的原因：<!-- TabWidget 的 id 必须为@android:id/tabs，因为 Android 框架的源代码会查询该 id。

```
<TabWidget
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:id="@android:id/tabs"
/>
```

如果把 TabWidget 的 id 设置为@android:id/tabs，就会抛出一个运行时异常，如下所示。

```
     if (mTabWidget == null) {
               throw new RuntimeException(
                     "Your TabHost must have a TabWidget whose id attribute is
'android.R.id.tabs'");
           }
```

继续阅读 setup 方法的源代码，我们发现在该方法中最后出现对 FrameLayout 的强制要求：

```
    mTabContent = (FrameLayout) findViewById(com.android.internal.R.id.tabcontent);
           if (mTabContent == null) {
               throw new RuntimeException(
                     "Your TabHost must have a FrameLayout whose id attribute is "
                           + "'android.R.id.tabcontent'");
           }
```

就是说我们必须把 Tab 的内容放在 FrameLayout 中，同时必须该 FrameLayout 的 id 必须为@android:id/tabcontent，而我们的布局文件正是这么做的：<!-- 包含 Tab 的具体内容必须是 FrameLayout，FrameLayout 的 id 必须为"@android:id/tabcontent"，因为 Android 框架的源代码会查询该 id。

```
    <FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@android:id/tabcontent"
    >
```

如果不符合上述两者的要求，就会抛出一个运行时异常：

```
    if (mTabContent == null) {
               throw new RuntimeException(
                     "Your TabHost must have a FrameLayout whose id attribute is "
                           + "'android.R.id.tabcontent'");
           }
```

五、自定义标题栏

由于 Android 手机屏幕的限制，现在越来越多的程序在开发时把 Android 系统的标题栏利用起来作为导航栏，这样一方面可以取得程序界面的一致性，另外一方可以充分利用有限的屏幕空间。

接下来通过一个案例来说明如何自定义标题栏。

（1）建立一个名称为 CustomizedTitle 的 Android 工程。

（2）编写自定义标题的界面。

我们把自定义标题的界面放在一个 layout 目录下的 cutomizedview.xml 中，内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        >
        <TextView
         android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hey,here is an customized title"
        />
    </LinearLayout>
```

我们只是在 XML 文件中简单地显示一段文字，当然，因为这是一个界面描述文件，读者可以根据自己的需要编写任意形态的标题界面。

（3）编写 Activity 控制器。

CustomizedTitleActivity 的内容如下所示。

```
    package com.wangjialin.title;
    import android.app.Activity;
    import android.os.Bundle;
    import android.view.Window;
    public class CustomizedTitleActivity extends Activity {
           @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            ////////////////////////////////////////////////
            //requestWindowFeature(Window.FEATURE_CUSTOM_TITLE)必须在 setContentView
之前被调用
            //原因在于 Android 的界面是先绘制标题，后绘制窗口中的内容的
            //requestWindowFeature(Window.FEATURE_CUSTOM_TITLE)会告知框架我们需要自
定义标题区内容
            ////////////////////////////////////////////////
            requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
            setContentView(R.layout.main);
            //设置自定义的标题
            getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
R.layout.customizedview);
        }
    }
```

源代码非常简单，但是必须要注意：

在 Android 整个界面的绘制中，因为是先绘制标题区，然后再绘制内容显示去，所以 requestWindowFeature(Window.FEATURE_CUSTOM_TITLE)必须在 setContentView 之前被调用。

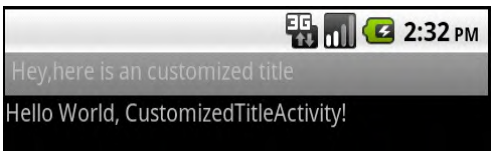
（4）测试。

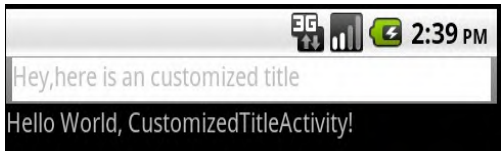
运行我们的程序，效果如下（左）图所示。

把 customizedview.xml 中的 LinearLayout 加上背景色，并设置背景色为白色，修改后代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ffffff"
    >
    <TextView
     android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hey,here is an customized title"
    />
</LinearLayout>
```

此时重新运行程序，如下（右）图所示。





此时出现了一个非常严重的问题：自定义的标题栏布局文件无法全部覆盖整个标题栏。

系统窗口标题的界面文件在 android 系统源代码 core\res\res\layout 位置下，文件名为 screen_custom_title.xml，内容如下所示。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:fitsSystemWindows="true">
    <FrameLayout android:id="@android:id/title_container"
        android:layout_width="match_parent"
        android:layout_height="?android:attr/windowTitleSize"
        style="?android:attr/windowTitleBackgroundStyle">
    </FrameLayout>
    <FrameLayout android:id="@android:id/content"
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="1"
        android:foregroundGravity="fill_horizontal|top"
        android:foreground="?android:attr/windowContentOverlay" />
</LinearLayout>
```

上述的 XML 文件中出现了下面这些属性：

- ?android:attr/windowTitleSize；
- ?android:attr/windowTitleBackgroundStyle；
- ?android:attr/windowContentOverlay。

上述属性的值在 core\res\res\values\themes.xml 文件中被赋值：

```
    <style name="Theme">
     <item name="windowContentOverlay">@android:drawable/title_bar_shadow</item>
     <item name="windowTitleSize">25dip</item>
     <item
name="windowTitleBackgroundStyle">@android:style/WindowTitleBackground</item>
    </style>
```

@android:style/WindowTitleBackground 样式在 core\res\res\values\styles.xml 文件中定义：

```
     <style name="WindowTitleBackground">
           <item name="android:background">@android:drawable/title_bar</item>
    </style>
```

从源代码中可以看出，FrameLayout 是回叠加的，后面的 FrameLayout 会叠加前面的 FrameLayout。

从源代码可以知道，后面的灰色区域的 id 即为@android:id/title_container，我们自定义的白色区域的 id 即为@android:id/content。

因为系统应用了默认的窗口主题，所以要解决问题，就需要修改默认主题的行为：

- 对于@android:id/title_container 而言，我们需要修改 Style 文件的行为 style="?android:attr/windowTitleBackgroundStyle"；
- 对于@android:id/content 而言，我们需要修改 android:foreground="?android:attr/ windowContentOverlay"。

很显然，应用程序开发时，我们是无法修改系统的源代码的，不过可以通过样式文件的继承结构来进行修改。

在工程的 values 文件夹下定义一个名称为 styles.xml 样式文件，并编写其内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <resources>
        <!-- 继承系统默认主题，修改相关的属性值 -->
      <style name="mytheme" parent="android:Theme">
         <!-- 自定义标题所在的区域 -->
        <item name="android:windowContentOverlay">@drawable/contentColor</item>
        <!-- 标题栏的高度 -->
        <item name="android:windowTitleSize">55dip</item>
        <!-- 标题的背景 -->
        <item  name="android:windowTitleBackgroundStyle">@style/myBackgroundStyle
</item>
      </style>
      <drawable name="contentColor">#00000000</drawable>
      <style name="myBackgroundStyle">
        <item name="android:background">@drawable/bg</item>
      </style>
    </resources>
```

在样式文件中，我们不要一个@drawable/bg 文件。

在 res 目录下新建一个 drawable 文件夹，并在该文件夹下建立一个名称为 bg.xml 的文件，其内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
   android:shape="rectangle">
   <gradient
      android:startColor="#00CCFF"
      android:endColor="#3366FF"
      android:angle="270"/>
   <padding android:left="7dp"
      android:top="2dp"
      android:right="7dp"
      android:bottom="2dp" />
</shape>
```

完成上述工作后，我们要在 AndroidManifest.xml 中应用该新的主题，修改后内容如下所示。

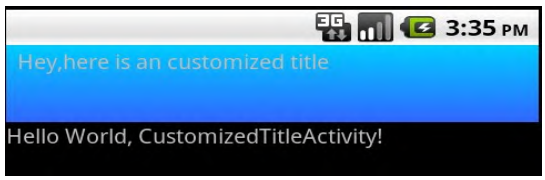
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.wangjialin.title"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk android:minSdkVersion="8" />
   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:label="@string/app_name"
         android:name=".CustomizedTitleActivity"
         android:theme="@style/mytheme" ><!-- 设置主题为自定义的主题 -->
         <intent-filter >
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

另外，我们还需要修改 customizedview.xml 文件，去掉其背景色：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <TextView
    android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

```
        android:text="Hey,here is an customized title"
        />
    </LinearLayout>
```

此时在重新启动我们的程序，效果如下图所示。



总结：

通过样式文件继承的方式，我们修改了系统的默认窗口行为，关键点有两个：

- 标题内容的背景必须为透明，这样就可以看到整个标题的背景；
- 整个标题的背景可以根据需要进行设置。

六、PopupWindow

1．PopupWindow 为 Android 程序带来的视觉冲击

PopupWindow 是一个可以显示在当前 Activity 之上的浮动容器，PopupWindow 弹出的位置是能够改变的，按照有无偏移量，可以分为无偏移和有偏移两种；按照参照对象的不同又可以分为两种：相对某个控件（Anchor 锚点）的位置和在父容器内部的相对位置。

创建 PopupWindow：

```
    LayoutInflater mLayoutInflater = (LayoutInflater) context.getSystemService
(LAYOUT_INFLATER_SERVICE);
    View contentView = mLayoutInflater.inflate(R.layout.xxx, null); //
R.layout.xxx 为界面文件
    PopupWindow pop = new PopupWindow(contentView, LayoutParams.FILL_PARENT,
LayoutParams.WRAP_CONTENT);
    pop.setBackgroundDrawable(new BitmapDrawable());
    pop.setFocusable(true);//取的焦点，创建出来的 popupwindow 默认无焦点
```

显示 PopupWindow 的方法：

- showAsDropDown(View anchor) 定义相对某个控件的位置（正左下方），无偏移；
- showAsDropDown(View anchor, int xoff, int yoff) 定义相对某个控件的位置，有偏移，xoff 是 X 轴的偏移量，yoff 是 Y 轴的偏移量；
- showAtLocation(View parent, int gravity, int x, int y) 定义在父容器的什么位置，gravity 为相对位置，如正中央 Gravity.CENTER、下方 Gravity.BOTTOM、Gravity.RIGHT|Gravity.BOTTOM 右下方等，后面两个参数为 X/Y 轴的偏移量。

关闭 PopupWindow：dismiss() 。

利用 PopupWindow，可以制作出很多动态酷炫的效果，例如，在单击某个按钮时会缓慢的在按钮下面弹出一个菜单，而且在弹出的过程中可以根据需要制作动画效果等。

2．PopupWindow 编程实战

下面通过一个带有动态效果的菜单为例，来细致地说明 PopupWindow 的应用。

（1）建立一个新的名称为 PopupWindow 的 Android 工程。

（2）构建界面文件。

主界面就是一个线性布局中存放一个简单的按钮，指定该按钮的单击响应方法是“openPopWindow”，同时还有一点是值得注意的是我们设置了 LinearLayout 的 id 为 main，这样方便引用。

我们想在单击按钮时弹出一个 PopupWindow 类型的窗口，此窗口具体的内容可以根据需要进行设置，我们把其布局文件放在 layout 文件夹下 pop.xml 文件中，内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="@drawable/popbg"
  >
  <!-- 一个表格视图 -->
  <GridView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:numColumns="4"
  android:horizontalSpacing="10dp"
  android:verticalSpacing="10dp"
  android:id="@+id/gridView"
  />
</LinearLayout>
```

我们设置弹出的窗口的内容为一个 GridView 类型构建的表格视图，同时我们也设置了 LinearLayout 的背景文件，在 res 下新建一个 drawable 文件夹，在其中创建 popbg.xml 文件，如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#F0FBFF"
        android:endColor="#AFDAFC"
        android:angle="270"/
    <padding android:left="7dp"
        android:top="2dp"
        android:right="7dp"
        android:bottom="2dp" />
</shape>
```

GridView 表格中具体每个 Item 内容也可以根据需要进行设置，我们在这里设置上面是一张图片，下面是说明文字，该 Item 的布局文件放在 layout 下的 grid_item.xml 中：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
  android:gravity="center"
  >
  <!-- 一张图片 -->
  <ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/icon"
  />
  <!-- 提示文字 -->
  <TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:textColor="#003399"
  android:textSize="14sp"
  android:gravity="center"
  android:id="@+id/title"
  />
</LinearLayout>
```

（3）创建 Activity 控制器。

```
package com.wangjialin.popwindow;
import java.util.ArrayList;
import java.util.HashMap;
import android.app.Activity;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.view.Gravity;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;
import android.widget.ListAdapter;
import android.widget.PopupWindow;
import android.widget.SimpleAdapter;
public class PopupWindowActivity extends Activity {
    /** 菜单图片 **/
    private int[] menu_image_array = {R.drawable.i1, R.drawable.i2,
    R.drawable.i3, R.drawable.i4,
    R.drawable.i5, R.drawable.i6,
    R.drawable.i7, R.drawable.i8};
```

```
        /** 菜单文字 **/
        private String[] menu_name_array = {"搜索", "文件管理", "下载管理", "全屏", "
网址", "书签", "加入书签", "分享页面"};
        //声明弹出窗口
        PopupWindow popupWindow;
        //声明表格视图
        GridView gridView;
       @Override
       public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);
           //获得 PopupWindow 的视图布局文件
           View popView = getLayoutInflater().inflate(R.layout.pop, null);
           //用获得 PopupWindow 视图布局文件的实例来构造 PopupWindow
           popupWindow = new PopupWindow(popView, ViewGroup.LayoutParams.FILL_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT);
           //设置一张空的背景图片
           popupWindow.setBackgroundDrawable(new BitmapDrawable());
           //取的焦点，创建出来的 popupwindow 默认无焦点
           popupWindow.setFocusable(true);
           //设置 PopupWindow 的动画效果
           popupWindow.setAnimationStyle(R.style.animationStyle);
           //实例化表格视图
           gridView = (GridView) popView.findViewById(R.id.gridView);
           //设置表格视图 Item 的单击事件监听器
           gridView.setOnItemClickListener(new ItemClickListener());
       }
       /**
        * 表格视图的单击事件监听器
        * @author Android
        *
        */
       private final class ItemClickListener implements OnItemClickListener{
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
                //当 PopupWindow 显示时，关闭 PopupWindow
                if(popupWindow.isShowing()) popupWindow.dismiss();
            }
       }
           /**
        * 表格视图的适配器
        * @return
        */
       private ListAdapter createAdapter() {
        //用 HashMap 存放图片与文字，用 ArrayList 来存放 HashMap
        ArrayList<HashMap<String,  Object>>  data  =  new  ArrayList<HashMap
<String,Object>>();
        //遍历数组，以此加入到 ArrayList 中
        for(int i = 0 ; i < menu_image_array.length ; i++){
            HashMap<String, Object> item = new HashMap<String, Object>();
```

```
            item.put("icon", menu_image_array[i]);
            item.put("title", menu_name_array[i]);
            data.add(item);
        }
        //创建适配器
            SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.
grid_item,
                    new String[]{"icon","title"}, new int[]{R.id.icon, R.id.title});
            return adapter;
        }
       /**
        * 单击 Button 时的响应方法
        * @param v
        */
        public void openPopWindow(View v){
            //设置弹出窗口显示位置
       popupWindow.showAtLocation(findViewById(R.id.main), Gravity.BOTTOM|Gravity.
CENTER_HORIZONTAL, 0, 0);
        //设置 GridView 的适配器，关联要显示的数据
        gridView.setAdapter(createAdapter());
       }
    }
```

我们需要在 res 下的 drawable 文件夹中加入图片。

（4）创建 PopupWindow 的动画效果。

在控制器中设置了 PopupWindow 的动画效果：

```
    //设置 PopupWindow 的动画效果
          popupWindow.setAnimationStyle(R.style.animationStyle);
```

我们在 values 文件夹下 styles.xml 文件中创建 animationStyle ：

```
    <?xml version="1.0" encoding="utf-8"?>
    <resources>
      <style name="animationStyle">
        <item name="android:windowEnterAnimation">@anim/enter</item>
        <item name="android:windowExitAnimation">@anim/out</item>
      </style>
    </resources>
```

可以看出，我们设置了 PopupWindow 弹出时的动画效果，也设置了 PopupWindow 退出时的动画效果。

在 res 文件夹下建立一个 anim 文件夹，并在其中建立两个文件：enter.xml 和 out.xml。

Enter.xml 的具体内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <set xmlns:android="http://schemas.android.com/apk/res/android"
       android:shareInterpolator="false">
       <translate        android:fromYDelta="100%p"        android:toYDelta="0"
```

```
android:duration="1000"/>
       <alpha            android:fromAlpha="0.5"            android:toAlpha="1"
android:duration="2000"/>
    </set>
```

Out.xml 的具体内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <set xmlns:android="http://schemas.android.com/apk/res/android"
       android:shareInterpolator="false">
       <translate android:fromYDelta="0" android:toYDelta="100%p" android:duration=
"1000"/>
       <alpha   android:fromAlpha="1"   android:toAlpha="0"   android:duration=
"1000"/>
    </set>
```

动画的内容非常简单，读者可以自行参考 SDK 文档，在此不在赘述。

（5）测试程序。

发布并运行程序，效果如下（左）图所示，单击按钮，会按照设置的动画效果弹出 PopupWindow，效果如下（中）图所示，完全进入后的效果如下（右）图所示。

退出时也会按照我们设置的动画效果进行运行，为节省篇幅，这次不再截图。

七、图片拖拉功能

图片拖拉功能是处理图片时一个有用且常用的功能。由于手机屏幕尺寸的限制，往往无法在手机上一次性显示一张比较大的图片，也就是说，我们在手机上一次性只能看到图片的一部分，此时就可以使用图片的拖拉功能来拖动图片，进而查看图片相应的部分。

下面通过一个例子来实现图片的拖拉功能。

（1）建立一个名称为 DragAndDrop 的 Android 工程。

（2）创建界面。

我们把界面放在默认的布局文件 main.xml 中，读者需要特别注意的是我们的 ImageView 的控制方式被设置成为了 matrix，这样就可以在代码中非常方便对图片进行控制了。

在 main.xml 文件中需要一张图片（图片大小要超过手机屏幕），我们把图片存放 drawable-mdpi 文件夹中。

（3）编写控制器。

DragAndDropActivity 的内容如下所示：

```
package com.wangjialin.picture;
import android.app.Activity;
import android.graphics.Matrix;
import android.graphics.PointF;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;
public class DragAndDropActivity extends Activity {
    //声明图片
    private ImageView imageView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //获得图片
        imageView = (ImageView) this.findViewById(R.id.imageView);
        //设置图片的触摸监听器
        imageView.setOnTouchListener(new ImageTouchListener());
    }
    /**
     * 图片的触摸监听器
     * @author Android
     *
     */
    private class ImageTouchListener implements OnTouchListener{
    //声明一个坐标点
    private PointF startPoint;
    //声明并实例化一个 Matrix 来控制图片
    private Matrix matrix = new Matrix();
    //声明并实例当前的图片的 Matrix
    private Matrix currentMatrix = new Matrix();
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction() & MotionEvent.ACTION_MASK) {
            case MotionEvent.ACTION_DOWN://按下屏幕
                //获得当前按下点的坐标
                startPoint = new PointF(event.getX(), event.getY());
                //把当前的图片 Matrix 设置为按下图片的 Matrix
                currentMatrix.set(imageView.getImageMatrix());
```

```
                break;
            case MotionEvent.ACTION_MOVE://按下屏幕拖动
                //移动的 X 坐标的距离
                float dx = event.getX() - startPoint.x;
                //移动的 Y 坐标的距离
                float dy = event.getY() - startPoint.y;
                //设置 Matrix 当前的 matrix
                matrix.set(currentMatrix);
                //告诉 matrix 要移动的 X 轴和 Y 轴的距离
                matrix.postTranslate(dx, dy);
                break;
            }
            //按照 Matrix 的要求移动照片到达某一个位置
            imageView.setImageMatrix(matrix);
            //返回 True 表明我们会消费掉该动作，不需要父控件进行进一步的处理
            return true;
        }
    }
}
```

（4）测试程序。

发布并运行程序，此时我们可以拖动图片了。

八、多点触摸与缩放功能

在 Android 上查看图片或者浏览网页时，我们往往有把图片或者网页放大或者缩小的需求，这样就能够获得更多的细节信息或者获得更多的全貌信息，多点触摸与缩放功能正是满足这种应用场景的技术。

为节省篇幅，我们使用上一章图片拖拉使用的工程 DragAndDrop。

DragAndDrop 原有内容保持不变，我们所要做的主要是针对图片的多点触摸与缩放功能，因此，我们只需要修改控制器 DragAndDropActivity 即可，修改后其内容如下所示。

```
package com.wangjialin.picture;
import android.app.Activity;
import android.graphics.Matrix;
import android.graphics.PointF;
import android.os.Bundle;
import android.util.FloatMath;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;
public class DragAndDropActivity extends Activity {
    //声明图片
    private ImageView imageView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        //获得图片
        imageView = (ImageView) this.findViewById(R.id.imageView);
        //设置图片的触摸监听器
        imageView.setOnTouchListener(new ImageTouchListener());
    }
    /**
     * 图片的触摸监听器
     * @author Android
     *
     */
    private class ImageTouchListener implements OnTouchListener{
    //声明一个坐标点
    private PointF startPoint;
    //声明并实例化一个 Matrix 来控制图片
    private Matrix matrix = new Matrix();
    //声明并实例当前的图片的 Matrix
    private Matrix currentMatrix = new Matrix();
    //声明缩放时初始的距离
    private float startDistance;
    //拖拉的标识
    private static final int DRAG = 1;
    //缩放的标识
    private static final int ZOOM = 2;
    //标识
    private int mode;
    //缩放的中间点
    private PointF midPoint;
@Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction() & MotionEvent.ACTION_MASK) {
            case MotionEvent.ACTION_DOWN://按下屏幕
                //此时处于拖拉方式下
                mode = DRAG;
                //获得当前按下点的坐标
                startPoint = new PointF(event.getX(), event.getY());
                //把当前的图片 Matrix 设置为按下图片的 Matrix
                currentMatrix.set(imageView.getImageMatrix());
                break;
            case MotionEvent.ACTION_MOVE://按下屏幕拖动
                //根据不同的模式执行相应的缩放或者拖拉操作
                switch (mode) {
                case DRAG:
                    //移动的 X 坐标的距离
                    float dx = event.getX() - startPoint.x;
                    //移动的 Y 坐标的距离
                    float dy = event.getY() - startPoint.y;
                    //设置 Matrix 当前的 matrix
                    matrix.set(currentMatrix);
                    //告诉 matrix 要移动的 X 轴和 Y 轴的距离
```

```
                        matrix.postTranslate(dx, dy);
                        break;
                    case ZOOM:
                        //计算缩放的距离
                        float enddistance = distance(event);
                        //计算缩放的比率
                        float scale = enddistance / startDistance;
                        //设置当前的 Matrix
                        matrix.set(currentMatrix);
                        //设置缩放的参数
                        matrix.postScale(scale, scale, midPoint.x, midPoint.y);
                        break;
                    }
                    break;
                case MotionEvent.ACTION_POINTER_DOWN://已经有一个手指按住屏幕，再有一
个手指按下屏幕就会触发该事件
                    //此时为缩放模式
                    mode = ZOOM;
                    //计算开始时两个点的距离
                    startDistance = distance(event);
                    //在两个点的距离大于 10 时才进行缩放操作
                    if(startDistance > 10){
                        //计算中间点
                        midPoint = mid(event);
                        //得到进行缩放操作之前，照片的缩放倍数
                        currentMatrix.set(imageView.getImageMatrix());
                    }
                    break;
                case MotionEvent.ACTION_POINTER_UP://有一个手指离开屏幕，但还有手指在
屏幕就会触发该事件
                case MotionEvent.ACTION_UP://最后一个手指离开屏幕，就会触发该事件
                    mode = 0;
                    break;
                }
                //按照 Matrix 的要求移动照片到达某一个位置
                imageView.setImageMatrix(matrix);
                //返回 True 表明我们会消费掉该动作，不需要父控件进行进一步的处理
                return true;
            }
        }
        /**
         * 计算两点之间的距离
         * @param event
         * @return
         */
        public static float distance(MotionEvent event) {
            float dx = event.getX(1) - event.getX(0);
            float dy = event.getY(1) - event.getY(0);
            return FloatMath.sqrt(dx * dx + dy * dy);
        }
```

```
    /**
     * 得到两点之间的中间点坐标
     * @param event
     * @return
     */
    public static PointF mid(MotionEvent event) {
        float x = (event.getX(1) + event.getX(0)) / 2;
        float y = (event.getY(1) + event.getY(0)) / 2;
        return new PointF(x, y);
    }
}
```

因为需要测试多点触摸和缩放功能，所以必须把程序发布到 Android 真实的手机上，笔者已经测试过这个代码，读者可以把程序安装在自己的手机上进行测试。

九、Android 中图形编程实战

1．使用 Layer List 制作动态相框

Layer List 是 Android 中的一种制作图形的方式，它是通过叠加若干张图片的方式来形成最终的图片，最终的图片在代码中表现为一个 LayerDrawable 对象。

下面说明通过 Layer List 制作动态相框的方法。

（1）新建一个名称为 LayerList 的 Android 工程。

（2）编写 XML 界面文件。

界面文件默认放在 main.xml 文件中，修改后内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <!-- 显示一张 LayerDrawable -->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/layerlist"
        />
    <!-- 显示一个按钮，当单击该按钮时执行名称为 change 的方法 -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ChangeUser"
        android:onClick="change"/>
</LinearLayout>
```

在 src 下建立一个 drawable 文件夹并在其中建立一个 layerlist.xml 文件，如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <!-- 采用 Layer List 的方式构建图片，后面的 Item 会以指定的设置的方式覆盖前面的区域 -->
    <layer-list xmlns:android="http://schemas.android.com/apk/res/android">
       <item android:drawable="@drawable/faceback" />
       <item android:id="@+id/userimage" android:drawable="@drawable/user"
        android:left="18dp"      android:top="68.0dp"      android:right="18dp"
android:bottom="22dp" />
    </layer-list>
```

在 layerlist.xml 中需要两张图片来合成新的图片，把这两张图片加入到 drawable 中。

（3）编写 Activity 控制器。

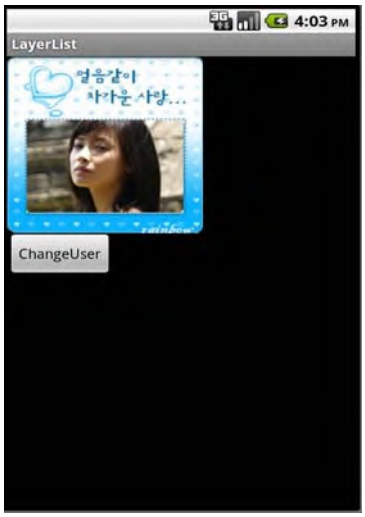
默认的 Activity 控制器为 LayerListActivity，我们在其中编写代码如下：

```
    package com.wangjialin.drawable.layerlist;
    import android.app.Activity;
    import android.graphics.drawable.LayerDrawable;
    import android.os.Bundle;
    import android.view.View;
    import android.widget.ImageView;
    public class LayerListActivity extends Activity {
        //声明我们要显示的合成图片
        private ImageView imageView;
       @Override
       public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);
           //实例化 ImageView 控件
           imageView = (ImageView) this.findViewById(R.id.imageView);
       }
       /**
        * 按钮单击事件响应方法
        * @param v
        */
       public void change(View v){
        //获得我们合成的图片
        LayerDrawable layerDrawable = (LayerDrawable) getResources().getDrawable
(R.drawable.layerlist);
        //置换合同图片中的部分内容已达到相框的效果
        layerDrawable.setDrawableByLayerId(R.id.userimage, getResources().getDrawable
(R.drawable.ic_launcher));
        //显示更改后的效果
        imageView.setImageDrawable(layerDrawable);
       }
    }
```

当我们单击按钮时，会调用单击事件的响应方法 change 方法，然后在通过 setDrawable ByLayerId 重新设置图片，达到动态相框的效果。

（4）测试。

发布并运行程序，初始视图如下（左）图所示，单击“ChangeUser”按钮，更换头像，效果如下（右）图所示。



2．使用 StateList 实现不同状态下图片的切换

在应用程序开发中，我们几乎不可避免要面对这样一种场景：对一个空间，如按钮，当该控件获得焦点时会是一种界面上的表现，当该控件被按下时是另外一种界面上的表现，当空间没有获得焦点时又是另外一种界面上的表现，这就是 StateList 的用武之地。值得注意的是 StateList 所构建的 StateListDrawable 必须要放在 drawab 目录下，而且这是一个 XML 文件。

StateList 最为常用的状态属性是控件被按下：state_pressed；控件获得焦点：state_focuse；控件被选中：state_selected。

下面用一个程序来细致地说明 StateList 的构建。

（1）新建一个名称为 StateList 的 Android 工程。

（2）创建界面。

在我们的主界面文件中声明一个应用了 StateList 的按钮，该按钮会随着在 drawable 文件夹下的 statelist.xml 文件的设置而产生相应的变化。

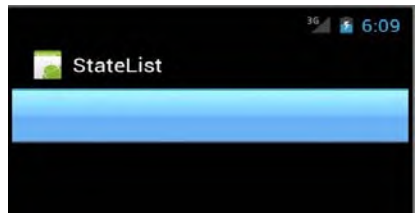
创建 drawable 文件夹并创建 statelist.xml 文件，statelist 的具体内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <!-- 声明一个状态选择器 -->
    <selector xmlns:android="http://schemas.android.com/apk/res/android">
        <!-- 当 item 可用且被按下的情况 -->
        <item      android:state_pressed="true"      android:state_enabled="true"
android:drawable="@drawable/bg_selected" /> <!-- pressed -->
        <!-- 其他情况 -->
        <item android:drawable="@drawable/bg_normal" /> <!-- default -->
    </selector>
```

在 selector 中，需要两张图片。

（3）测试。

发布并运行程序，初始视图如下（左）图所示，当按下按钮时，按钮变化为如下（右）图所示的状态。



3．使用 Level list 揭秘电池电量变换状态

首先来看一下手机电池电量变化的三种状态，如下图所示。

上述电池电量的变化就是在电池电量的不同状态时采用不同的图片去显示电量。

手机电池电量的变换状态就是采用了 Level List 技术。下面通过一个实例来细致地说明如何使用 Level List 技术。

（1）新建一个名称为 LevelList 的 Android 工程。

（2）编写界面文件。

界面在默认的 main.xml 文件中描述，具体内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <!-- 使用 Level List 的图片 -->
     <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/levellist"
    android:id="@+id/imageView"
    />
     <!-- 数字输入框 -->
    <EditText
     android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:numeric="signed"
    android:id="@+id/level"
    />
    <!-- 触发图片切换的按钮 -->
```

```
        <Button
         android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Have a change"
        android:onClick="changeImage"
        />
    </LinearLayout>
```

在 ImageView 中使用了 levellist 图形资源，该图形资源使用了 Level List 技术，levellist.xml 位于 drawable 文件夹中。

Levellist.xml 的具体内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <level-list xmlns:android="http://schemas.android.com/apk/res/android" >
        <!-- 当 level 的值在 0 到 20 之间的时候,使用 faceback 这张图片 -->
        <item     android:drawable="@drawable/faceback"     android:minLevel="0"
android:maxLevel="20" />
        <!-- 当 level 的值在 21 到 100 之间的时候,使用 user 这张图片 -->
        <item       android:drawable="@drawable/user"       android:minLevel="21"
android:maxLevel="100"/>
    </level-list>
```

（3）编写控制器代码。

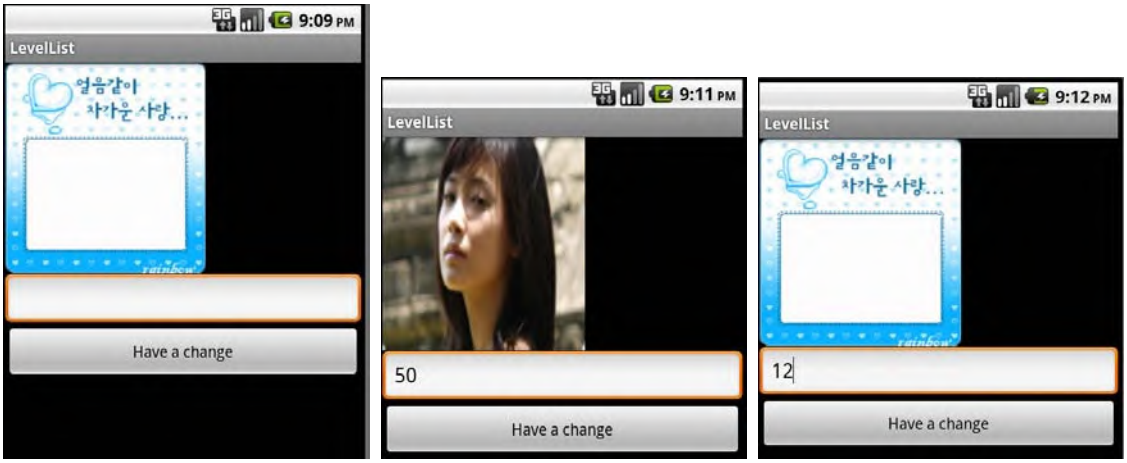
控制器代码位于默认的“LevelListActivity”，具体内容如下所示。

```
    package com.wangjialin.picture.levellist;
    import android.app.Activity;
    import android.graphics.drawable.LevelListDrawable;
    import android.os.Bundle;
    import android.view.View;
    import android.widget.EditText;
    import android.widget.ImageView;
    public class LevelListActivity extends Activity {
            //声明一张图片控件
            private ImageView imageView;
            //声明一个文本输入框
           private EditText levelText;
           @Override
           public void onCreate(Bundle savedInstanceState) {
               super.onCreate(savedInstanceState);
               setContentView(R.layout.main);
               //实例化图片控件
               imageView = (ImageView) this.findViewById(R.id.imageView);
               //实例化文本输入框控件
               levelText = (EditText) this.findViewById(R.id.level);
           }
           /**
            * 按钮单击事件响应方法
            * @param v
```

```
             */
            public void changeImage(View v){
             //获得用户的输入值
             String level = levelText.getText().toString();
             //获得 ImageView 的 LevelListDrawable
             LevelListDrawable levelListDrawable = (LevelListDrawable) imageView.
getDrawable();
             //设置 LevelListDrawable 的 level，从而动态的改变图片的显示
             levelListDrawable.setLevel(Integer.valueOf(level));
            }
    }
```

（4）测试。

发布并运行程序，初始界面如下（左）图所示，在文本输入框中输入 50，单击“Have a change”按钮，效果如下（中）图所示，在文本输入框中输入 12，单击“Have a change”按钮，效果如下（右）图所示。



4．使用 Transition DRAWABLE 实现动态过渡效果

Transition Drawable 是实现两张图片之间动态过渡效果的方式。

（1）新建一个名称为 TransitionDrawable 的 Android 工程。

（2）创建界面。

界面中主要是一个应用了 Transition Drawable 的按钮，在 main.xml 中使用了 Transition Drawable 效果，该效果具体位于 transition.xml 文件中，transition.xml 位于 Drawable 文件夹下。Transition.xml 的具体内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <!-- 具有 Transition 效果 -->
    <transition xmlns:android="http://schemas.android.com/apk/res/android">
        <!-- 第一个具有图片的 Item -->
        <item android:drawable="@drawable/bg_normal" />
        <!-- 第二个具有图片的 Item -->
```

```
        <item android:drawable="@drawable/bg_selected" />
    </transition>
```

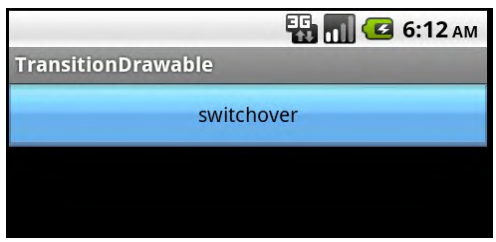
（3）编写控制器文件。

要实现过渡效果，需要在代码中实现，控制器文件位于 TransitionDrawableActivity ，具体内容如下所示。

```
    package com.wangjialin.picture.transitiondrawable;
    import android.app.Activity;
    import android.graphics.drawable.TransitionDrawable;
    import android.os.Bundle;
    import android.view.View;
    import android.widget.Button;
    public class TransitionDrawableActivity extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
        }
        /**
         * 按钮单击时的响应方法
         * @param v
         */
        public void changeImage(View v){
         //获得按钮的背景图
         TransitionDrawable drawable = (TransitionDrawable) ((Button) v).
getBackground();
         //设置渐变时间为 2 秒钟
         drawable.startTransition(2000);
        }
    }
```

（4）测试。

发布并运行程序，初始视图如下（左）图所示，单击“switchover”按钮，出现渐变效果，如下（右）图所示。



Transition Drawable 适合于一些对程序画面细节要求比较严格的程序。

5．使用 Clip DRAWABLE 揭秘水平进度条

水平进度条的效果如下图所示。

系统提供了水平进度条供我们显示进度使用，水平进度条在显示进度时使用的就是 Clip Drawable 技术。

此处需要说明，官方文档对于 Clip Drawable 在举例的时候有一个地方写错了，在获得图片的时候，官方文档使用了：

```
ClipDrawable drawable = (ClipDrawable) imageview.getDrawable();
```

而这显然是无法工作的，因为在界面文件中把 ClipDrawable 设置为了 background，所以，在此处要想获得 ClipDrawable，必须使用：

```
ClipDrawable drawable = (ClipDrawable) imageview.getBackground();
```

接下来，我们通过一个具体的实例来说明 ClipDrawable 使用细节。

（1）新建一个名称为 ClipDrawable 的 Android 工程。

（2）构建界面。

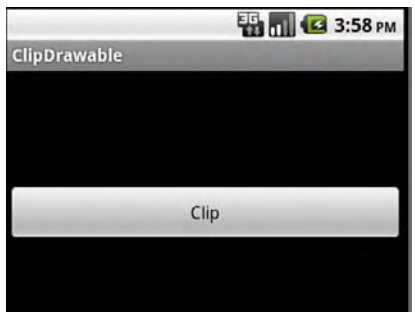
在界面中应用一张采用 Clip 技术的图片，该图片位于 drawable 文件夹中。

Clip.xml 的内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 按照水平方向，从左到右 Clip 一张名为 user 的图片 -->
<clip xmlns:android="http://schemas.android.com/apk/res/android"
   android:drawable="@drawable/user"
   android:clipOrientation="horizontal"
   android:gravity="left"
   />
```

（3）编写控制器代码。

如果此时发布程序，运行效果如下所示：



可以发现，此时无法看见我们设置的图片，这是为什么呢？

原因在于：Clip 类型的图片默认裁剪级别为 0，此时是全部裁剪，导致图片看不见；当级别为 10000 时，不裁剪图片，图片全部可见。

因此，此时我们可以编写控制器代码来设置 Clip 的裁剪级别。

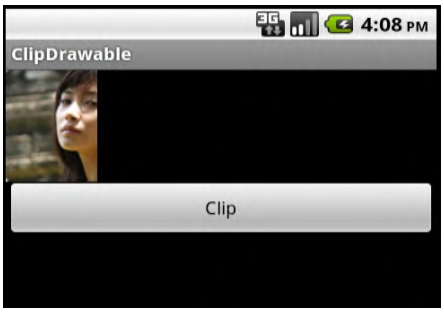
控制器代码代码位于默认的 ClipDrawableActivity 中，其具体内容如下所示。

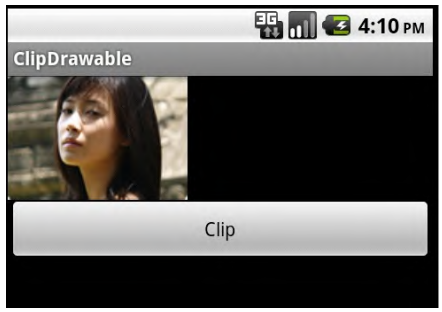
```
package com.wangjialin.picture.clipdrawable;
import android.app.Activity;
import android.graphics.drawable.ClipDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;
public class ClipDrawableActivity extends Activity {
        //声明含有 Clip 的 ImageView
        private ImageView imageView;
       @Override
       public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);
           //实例化含有 Clip 的 ImageView
           imageView = (ImageView) this.findViewById(R.id.imageView);
       }
       /**
        * 按钮单击事件的响应方法
        * @param v
        */
       public void changeImage(View v){
        //获得 ClipDrawable
        ClipDrawable clipDrawable = (ClipDrawable) imageView.getDrawable();
        //设置裁剪级别，Clip 类型的图片默认裁剪级别为 0，此时是全部裁剪，导致图片看不见
        //当级别为 10000 时，不裁剪图片，图片全部可见
        clipDrawable.setLevel(clipDrawable.getLevel()+ 1000);
       }
}
```

（4）测试。

发布并运行程序，默认视图如下（左）图所示，单击“Clip”按钮，裁剪效果如下（右）图所示。

继续连续单击“Clip”按钮，裁剪效果如下（左）图所示，继续单击“Clip”按钮，最终的裁剪效果如下（右）图所示。





6．使用 Shape DRAWABLE 自定义几何图形

Shape Drawable 图形资源是用 XML 文件生成几何图形的一种技术，这些几何图形包括矩形、椭圆形、圆形、三角形等。

下面通过一个具体示例来细致地说明其使用方法。

（1）新建一个名称为 ShapeDrawable 的 Android 工程。

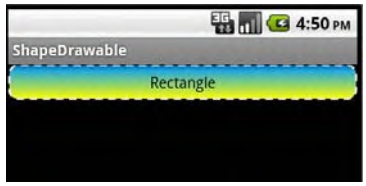
（2）创建界面。

在 main.xml 中应用的 Shape Drawable 资源位于 drawable 文件夹下的 rectangle.xml，内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个矩形 -->
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- 定义渐变，包含渐变角度、渐变的开始颜色和渐变的结束颜色 -->
     <gradient
        android:angle="270"
         android:startColor="#0099FF"
        android:endColor="#FFFF00"
         />
    <!-- 定义图形中的内容和四条边界线的距离 -->
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <!-- 设置圆角半径为 8dp -->
    <corners android:radius="8dp" />
    <!-- 定义矩形的边界线，包含边界线的宽度
     边界线的颜色、边界的破折号的长度为 7dp，破折号的宽度为 5dp-->
    <stroke
        android:width="2dp"
        android:color="#ffffff"
        android:dashWidth="7dp"
        android:dashGap="5dp" />
</shape>
```

（3）测试。

发布并运行程序，效果如下图所示。



十、循环显示视图的 Gallery

1．循环显示视图的 Gallery 原理分析

Gallery 提供一个画廊效果，通过左右滑动，可以水平滚动浏览 Gallery 中的图片。当单击当前图像的后一个图像时，这个图像列表会向左移动一格，当单击当前图像的前一个图像时，这个图像列表会向右移动一样，也可以通过拖动的方式来向左和向右移动图像列表。

默认情况下，Gallery 只能从第一项开始，滚动浏览到最后一项。浏览最后一项后，就不能再滚动，只能回退浏览。

如果我们在达到 Gallery 的最后一张显示的 View 时要能够继续循环浏览，也就是说，在我们浏览到最后一个 View 时，可以继续往下浏览，不过此时向下浏览是从开头开始的，如此循环。

要实现这样的效果，首先要对 Gallery 的 Adapter 进行修改。

无论使用哪个 Adapter，都需要修改两处：

（1）public int getCount() 方法要返回一个很大的数，它表示 Adapter 提供的数据的数量。要实现无限循环，必须把它修改成一个很大的值——Integer.MAX_VALUE。

（2）public View getView(int position, View convertView, ViewGroup parent) 中的 position 值是实现循环的关键，它表示当前要显示的哪一个 VIEW。通过对 position 进行取余运算即可实现循环的效果。

2．循环显示视图的 Gallery 编程实战

（1）新建一个名称为 CyclingGallery 的 Android 工程。

（2）构建 main.xml 文件。

（3）准备一下 Gallery 要使用的图片，放置在“drawable-mdpi”中。

（4）编写自定义的 Gallery 适配器。

此处我们在新建立的包中定义 Gallery 适配器的名称为 GalleryAdapter。

GalleryAdapter 中的内容如下所示。

```
package com.wangjialin.service;
import android.content.Context;
```

```
import android.content.res.TypedArray;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;
import com.wangjialin.gallery.R;
/**
 * Gallery 适配器
 * @author Android
 */
public class GalleryAdapter extends BaseAdapter {
    //Gallery 每个 Item 的背景样式
    int mGalleryItemBackground;
    //上下文
    private Context mContext;
    //图片的 id 数组
private Integer[] mImageIds = {
        R.drawable.sample_0,
          R.drawable.sample_1,
          R.drawable.sample_2,
          R.drawable.sample_3,
          R.drawable.sample_4,
          R.drawable.sample_5,
          R.drawable.sample_6,
          R.drawable.sample_7
    };
    /**
     * 适配器的构造方法
     * @param c
     */
    public GalleryAdapter(Context c) {
        mContext = c;
        TypedArray attr = mContext.obtainStyledAttributes(R.styleable.HelloGallery);
        mGalleryItemBackground = attr.getResourceId(
                R.styleable.HelloGallery_android_galleryItemBackground, 0);
        attr.recycle();
    }
    public int getCount() {
     //返回最大整数，构成无限循环
        return Integer.MAX_VALUE;
    }
    public Object getItem(int position) {
        return position;
    }
    public long getItemId(int position) {
        return position;
    }
    public View getView(int position, View convertView, ViewGroup parent) {
     //新建一个空的 ImageView
        ImageView imageView = new ImageView(mContext);
```

```
        //设置图片来源，注意此处我们采用了求模运算
        imageView.setImageResource(mImageIds[position%8]);
        //设置 Item 加入 Gallery 的参数
        imageView.setLayoutParams(new Gallery.LayoutParams(150, 100));
        //设置缩放方式
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);
        //设置图片的背景
        imageView.setBackgroundResource(mGalleryItemBackground);
        //返回图片
        return imageView;
    }
}
```

此处我们需要一个样式文件 attrs.xml，位于 res/values/目录下。

样式文件 attrs.xml 内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="HelloGallery">
        <attr name="android:galleryItemBackground" />
    </declare-styleable>
</resources>
```

在该文件中，我们运用了 Android 默认的 Gallery 的 Item 样式。

（5）编写控制文件。

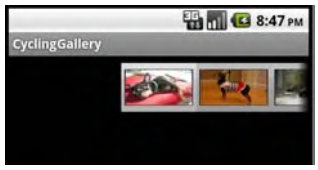
控制器文件位于 CyclingGalleryActivity ，具体内容如下所示。

```
package com.wangjialin.gallery;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Gallery;
import android.widget.Toast;
import com.wangjialin.service.GalleryAdapter;
public class CyclingGalleryActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //获得 Gallery 实例
        Gallery gallery = (Gallery) findViewById(R.id.gallery);
        //设置 Gallery 的适配器
        gallery.setAdapter(new GalleryAdapter(this));
        //设置对 Gallery 的 Item 的监听器
        gallery.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView parent,View v,int position,long id) {
              //弹出一条当前 Item 位置的通知
                Toast.makeText(CyclingGalleryActivity.this,  ""  +  position,
```

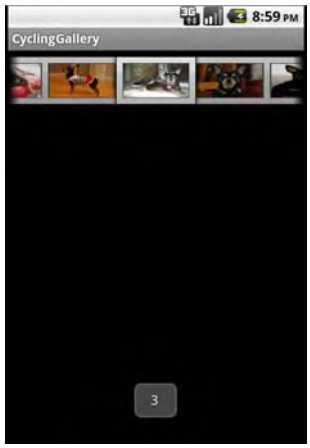
```
Toast.LENGTH_SHORT).show();
                }
            });
        }
    }
```

（6）运行测试程序。

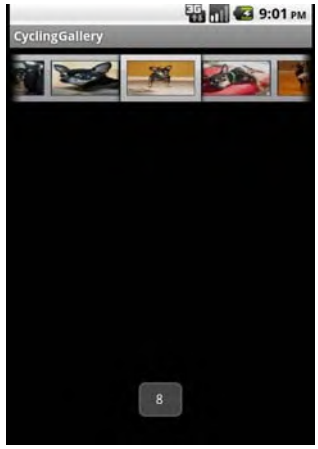
发布并运行程序，如下图所示。



当我们单击第一张图片时，效果如下（左）图所示，可以看出，第一张图片的位置是 0。向后面单击图片会发现图片自动从左往右滑动，如下（右）图所示。



第一轮循环的最后一张图片如下（左）图所示，我们继续单击一张图片即开始循环，如下（右）图所示。



成功实现了 Gallery 循环的效果。

附录 B　如何成为 Android 高手 V2.0：结合云计算和智能终端、软/硬件整合

Android 价值最大化方程式：硬件终端+驱动软件+硬件抽象层+平台系统软件+应用框架+应用软件+内容，其中平台的系统软件需要整合云服务，应用程序通过应用框架的 API 调用云服务。

- 2007 年底 Google 宣布举办总奖金高达 1000 万美元的开发者大奖赛，鼓励程序开发者在 Android 上写出实用而又具有创意的应用程序。
- 2009 年 5 月 27 日，在 Google 的 I/O 开发者聚会上，Google 发布了总奖金接近 2000 万美元的第二次大奖赛的消息，开发者们开始了新一轮的较量。
- 2010 年 Android 就业岗位的需求增长超过 700%，Android 市场增长率高达 1100%。
- 来自 IHS 公司的数据显示：2010 年 Android 市场应用程序的收入增长了 861.5%。
- Android 已经超越苹果成为美国智能手机用户的首选平台，而且在所有的智能手机中增长速度最快。
- CNET 科技资讯网 2011 年 2 月 1 日国际报道：Canalys 研究公司宣布，2010 年第四季度，Google Android 超越诺基亚 Symbian，成为全球第一大智能手机平台。
- 美国移动广告网络公司 Millennial Media 报告称，谷歌 Android 操作系统于 2011 年 4 月份继续在全球智能手机市场上占据主导地位，所占份额为 53%。
- 2011 年 8 月 Google 收购摩托罗拉移动部门，使得 Android 软件和硬件一体化成为趋势。

Android 是 Google 于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称，该平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。

Android 一出生就被打上了富二代的标记，不仅仅是因为它诞生于当今的网络霸主 Google，更主要的是它还有一个空前强大和壮观的开放手机联盟 OHA（Open Handset Alliance）提供的全力支持。OHA 是什么？OHA 涵盖了中国移动、T-Mobile、Sprint 等移动运营商，包括 HTC、Motorola、三星等手机制造商，有以 Google 为代表的手机软件商，还有以 Intel、NVIDIA 为标志的底层硬件厂商和 Astonishing Tribe 等商业运作公司，该组织声称组织的所有成员都会基于 Android 来开发新的手机业务。

但是，要成为 Android 开发高手并不是一件容易的事情。也并不是很多人想象的能够飞快地写出几行漂亮的代码去解决一些困难的问题就是 Android 高手了。真正的 Android 高手需要考虑的问题远远不是写几行漂亮的代码就足够的。下面是成为一名真正的 Android 高手必须掌握和遵循的一些准则：

（1）学会懒惰。

（2）精通 Android 体系架构、MVC、常见的设计模式、控制反转（IoC）。

（3）编写可重用、可扩展、可维护、灵活性高的代码。

（4）高效地编写高效的代码。

（5）学会至少一门服务器端开发技术。

（6）以 Android 软件和硬件深度整合的视角来看待 Android，2011 年 8 月份 Google 收购摩托罗拉移动部门，使得 Android 软件和硬件一体化成为趋势。

一、学会懒惰

没搞错吧？竟然让程序开发人员学会懒惰？程序开发人员可能是世界上最为忙碌的一类人啦！对，没错，学会懒惰！正因为程序开发人员忙碌，正因为程序开发人员可能会在客户无限变化的需求之下没日没夜地加班，所以要学会懒惰，这样，你就可以把更多的时间花在美好的事物身上！

如何懒惰：

- Don’t Reinvent the Wheel（不要重复发明轮子）。
- Inventing the Wheel（发明轮子）。

1．Don’t Reinvent the Wheel（不要重复发明轮子）

“轮子理论”，亦即“不要重复发明轮子”，这是西方国家的一句谚语，原话是：Don’t Reinvent the Wheel，意思是企业中任何一项工作实际上都有人做过，我们所需要做的就是找到做过这件事情的人。拿到软件领域中就是指有的项目或功能，别人已经做过，我们需要用的时候，直接拿来用即可，而不必重新制造。

Android 号称是首个为移动终端打造的真正开放和完整的移动软件。Android 发布后不久 Google 公司就发布了操作系统核心（Kernel）与部分驱动程序的源代码，到目前为止除了 Google Maps 等 Google 公司的核心组件没有开放源代码外，Android 基本完成了完全的开源，这就极大地促进了 Android 的普及和移植。受到 Android 开放行为和开源精神的影响，在世界各地，有成千上万的程序员喜欢和别人分享自己的聪明才智和自己编写的代码。你可以在 Google 的 Android 讨论组或者 Google 搜索引擎上搜索到很多优秀的程序代码。这样做并不是鼓励大家整天等着让别人为你编写代码，而是你可以“站在伟人的肩膀上”，充分发扬“拿来主义”，聪明地应用别人的程序代码可以节省你大量的时间。

下面笔者为大家介绍几个通用的类，这些类来自笔者平日的收集，如果你能把它们加入到你自己的类库中，你迟早会发现自己在进行 Android 开发的时候受益无穷。

（1）从输入流中获取数据并以字节数组返回，这种输入流可以来自 Android 本地也可以来自网络。

```
import java.io.ByteArrayOutputStream;
```

```
import java.io.InputStream;

public class StreamTool {
    /**
     * 从输入流获取数据
     * @param inputStream
     * @return
     * @throws Exception
     */
    public static byte[] readInputStream(InputStream inputStream) throws
    Exception {
        byte[] buffer = new byte[1024]; //可以根据实际需要调整缓存大小
        int len = -1;
        ByteArrayOutputStream outSteam = new ByteArrayOutputStream();
        while( (len = inputStream.read(buffer)) != -1 ){
            outSteam.write(buffer, 0, len);
        }
        outSteam.close();
        inputStream.close();
        return outSteam.toByteArray();
    }
}
```

（2）通过 Android 客户端上传数据到服务器：可以上传简单的表单，也可以方便地上传带有附件的文件，此类远远比 Android 自身的 HttpClient 更高效、更易于使用。

```
import java.io.DataOutputStream;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

public class HttpRequester {

    /**
     * 直接通过 HTTP 协议提交数据到服务器,实现如下表单的提交功能:
     *   <FORM METHOD=POST ACTION="http://192.168.0.200:8080/ssi/fileload/
        test.do" enctype="multipart/form-data">
            <INPUT TYPE="text" NAME="name">
```

```
            <INPUT TYPE="text" NAME="id">
            <input type="file" name="imagefile"/>
           <input type="file" name="zip"/>
       </FORM>
 * @param actionUrl 上传路径(注：避免使用 localhost 或 127.0.0.1 这样的路径
  测试，因为它会指向手机模拟器，可以使用 http://www.guoshiAndroid.cn 或
  http://192.168.1.10:8080 这样的路径测试)
 * @param params 请求参数 key 为参数名,value 为参数值
 * @param file 上传文件
 */
public static String post(String actionUrl, Map<String, String> params,
FormFile[] files) {
   try {
      String BOUNDARY = "----------7d4a6d158c9"; //数据分隔线
      String MULTIPART_FORM_DATA = "multipart/form-data";

      URL url = new URL(actionUrl);
      HttpURLConnection conn = (HttpURLConnection) url.openConnection
      ();
      conn.setConnectTimeout(5* 1000);
      conn.setDoInput(true);//允许输入
      conn.setDoOutput(true);//允许输出
      conn.setUseCaches(false);//不使用 Cache
      conn.setRequestMethod("POST");
      conn.setRequestProperty("Connection", "Keep-Alive");
      conn.setRequestProperty("Charset", "UTF-8");
      conn.setRequestProperty("Content-Type", MULTIPART_FORM_DATA +
      "; boundary=" + BOUNDARY);

      StringBuilder sb = new StringBuilder();
      for (Map.Entry<String, String> entry : params.entrySet()) {//
      构建表单字段内容
         sb.append("--");
         sb.append(BOUNDARY);
         sb.append("\r\n");
         sb.append("Content-Disposition: form-data; name=\""+ entry.
         getKey() + "\"\r\n\r\n");
         sb.append(entry.getValue());
         sb.append("\r\n");
      }
      DataOutputStream outStream = new DataOutputStream(conn.
      getOutputStream());
      outStream.write(sb.toString().getBytes());//发送表单字段数据
      for(FormFile file : files){//发送文件数据
       StringBuilder split = new StringBuilder();
          split.append("--");
          split.append(BOUNDARY);
          split.append("\r\n");
         split.append("Content-Disposition: form-data;name=\""+ file.
```

```
                getFormname()+"\";filename=\""+ file.getFilname() + "\"\r\n");
                split.append("Content-Type: "+ file.getContentType()+"
                \r\n\r\n");
                outStream.write(split.toString().getBytes());
                if(file.getInStream()!=null){
                 byte[] buffer = new byte[1024];
                 int len = 0;
                 while((len = file.getInStream().read(buffer))!=-1){
                     outStream.write(buffer, 0, len);
                 }
                 file.getInStream().close();
                }else{
                 outStream.write(file.getData(), 0, file.getData().length);
                }
                outStream.write("\r\n".getBytes());
            }
            byte[] end_data = ("--" + BOUNDARY + "--\r\n").getBytes();//
            数据结束标志
            outStream.write(end_data);
            outStream.flush();
            int cah = conn.getResponseCode();
            if (cah != 200) throw new RuntimeException("请求 url 失败");
            InputStream is = conn.getInputStream();
            int ch;
            StringBuilder b = new StringBuilder();
            while( (ch = is.read()) != -1 ){
             b.append((char)ch);
            }
            outStream.close();
            conn.disconnect();
            return b.toString();
        } catch (Exception e) {
         throw new RuntimeException(e);
        }
    }

    /**
     * 提交数据到服务器
     * @param actionUrl 上传路径(注：避免使用 localhost 或 127.0.0.1 这样的路径
       测试，因为它会指向手机模拟器，可以使用 http://www.guoshiAndroid.cn 或
       http://192.168.1.10:8080 这样的路径测试)
     * @param params 请求参数 key 为参数名,value 为参数值
     * @param file 上传文件
     */
    public static String post(String actionUrl, Map<String, String> params,
    FormFile file) {
       return post(actionUrl, params, new FormFile[]{file});
    }
```

```
public static byte[] postFromHttpClient(String path, Map<String,
String> params, String encode) throws Exception{
    List<NameValuePair> formparams = new ArrayList<NameValuePair>();//
    用于存放请求参数
    for(Map.Entry<String, String> entry : params.entrySet()){
        formparams.add(new BasicNameValuePair(entry.getKey(),
        entry.getValue()));
    }
    UrlEncodedFormEntity entity = new UrlEncodedFormEntity(formparams,
    "UTF-8");
    HttpPost httppost = new HttpPost(path);
    httppost.setEntity(entity);
    HttpClient httpclient = new DefaultHttpClient();//看做是浏览器
    HttpResponse response = httpclient.execute(httppost);//发送 post
     请求
    return StreamTool.readInputStream(response.getEntity().
    getContent());
}

/**
 * 发送请求
 * @param path 请求路径
 * @param params 请求参数 key 为参数名称 value 为参数值
 * @param encode 请求参数的编码
 */
public static byte[] post(String path, Map<String, String> params,
String encode) throws Exception{
    //String params = "method=save&name="+ URLEncoder.encode("老毕",
    "UTF-8")+ "&age=28&";//需要发送的参数
    StringBuilder parambuilder = new StringBuilder("");
    if(params!=null && !params.isEmpty()){
        for(Map.Entry<String, String> entry : params.entrySet()){
            parambuilder.append(entry.getKey()).append("=")
                .append(URLEncoder.encode(entry.getValue(), encode)).
                 append("&");
        }
        parambuilder.deleteCharAt(parambuilder.length()-1);
    }
    byte[] data = parambuilder.toString().getBytes();
    URL url = new URL(path);
    HttpURLConnection conn = (HttpURLConnection)url.openConnection();
    conn.setDoOutput(true);//允许对外发送请求参数
    conn.setUseCaches(false);//不进行缓存
    conn.setConnectTimeout(5 * 1000);
    conn.setRequestMethod("POST");
    //下面设置 http 请求头
    conn.setRequestProperty("Accept", "image/gif, image/jpeg, image/
    pjpeg, image/pjpeg, application/x-shockwave-flash, application/
    xaml+xml, application/vnd.ms-xpsdocument, application/x-ms-xbap,
```

```
            application/x-ms-application, application/vnd.ms-excel,
            application/ vnd.ms-powerpoint, application/msword, */*");
            conn.setRequestProperty("Accept-Language", "zh-CN");
            conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible;
             MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET CLR
             2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR
             3.5.30729)");
            conn.setRequestProperty("Content-Type", "application/x-www-form-
            urlencoded");
            conn.setRequestProperty("Content-Length", String.valueOf(data.
            length));
            conn.setRequestProperty("Connection", "Keep-Alive");
            //发送参数
            DataOutputStream outStream = new DataOutputStream(conn.
            getOutputStream());
            outStream.write(data);//把参数发送出去
            outStream.flush();
            outStream.close();
            if(conn.getResponseCode()==200){
                return StreamTool.readInputStream(conn.getInputStream());
            }
            return null;
        }
    }
```

2．Inventing the Wheel（发明轮子）

发明轮子？不错，发明轮子！我们不仅要发明轮子，更要努力成为世界上发明轮子的主导力量，唯有这样，才能谈得上中华民族软件大业的真正强大。在 Android 系统中，要发明轮子，就是我们要主动地解决一些世界上他人未解决的难题，或创造新的编程框架，或对 Android 进行深度的改造以适合自己的业务发展需要。Google 发布 Android 后不久，中国移动便投入了大量的人力和物力，在 Android 的基础上创建、融入了自己的业务并开发和封装了拥有新功能和新框架的 OMS，这是 Android 中发明轮子的一个非常重要的例子。可能你会说，发明轮子也太难了吧，别急，我们慢慢来，开发一个特定领域的框架。当然开发一个框架并非易事，但是也不是不可能。首先，我们分析一下框架的魅力源泉，看看 Spring、Struts 等 Java EE 框架，再看看.NET 框架，当然也可以看看发展得如火如荼、层出不穷的 PHP 框架，它们的强大和魅力的源泉都在于：IoC（Inversion of Control）。

Don’t call us, we’ll call you（别找我，我会来找你的）。下面我们就自己发明一个轮子的模型，实际展示一个框架最初核心的类。

下面的类是文件下载类，支持文件的多线程断点续传，使用该类即可安全、高效地下载任何类型的二进制文件。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.RandomAccessFile;
```

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Properties;
import java.util.UUID;
import java.util.concurrent.ConcurrentHashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import cn.guoshiAndroid.service.FileService;

import android.content.Context;
import android.util.Log;
/**
 * 文件下载器
 */
public class FileDownloader {
    private Context context;
    private FileService fileService;

    private static final String TAG = "FileDownloader";
    /* 已下载文件大小 */
    private int downloadSize = 0;
    /* 原始文件大小 */
    private int fileSize = 0;
    /* 线程数 */
    private DownloadThread[] threads;
    /* 下载路径  */
    private URL url;
    /* 本地保存文件 */
    private File saveFile;
    /* 下载记录文件 */
    private File logFile;
    /* 缓存各线程最后下载的位置*/
    private Map<Integer, Integer> data = new ConcurrentHashMap<Integer,
    Integer>();
    /* 每条线程下载的大小 */
    private int block;
    private String downloadUrl;//下载路径
    /**
     * 获取线程数
     */
    public int getThreadSize() {
        return threads.length;
    }
    /**
     * 获取文件大小
     * @return
```

```
     */
    public int getFileSize() {
        return fileSize;
    }
    /**
     * 累计已下载大小
     * @param size
     */
    protected synchronized void append(int size) {
        downloadSize += size;
    }
    /**
     * 更新指定线程最后下载的位置
     * @param threadId 线程 id
     * @param pos 最后下载的位置
     */
    protected void update(int threadId, int pos) {
        this.data.put(threadId, pos);
    }
    /**
     * 保存记录文件
     */
    protected synchronized void saveLogFile() {
        this.fileService.update(this.downloadUrl, this.data);
    }
    /**
     * 构建文件下载器
     * @param downloadUrl 下载路径
     * @param fileSaveDir 文件保存目录
     * @param threadNum 下载线程数
     */
    public FileDownloader(Context context, String downloadUrl, File
    fileSaveDir, int threadNum) {
        try {
            this.context = context;
            this.downloadUrl = downloadUrl;
            fileService = new FileService(context);
            this.url = new URL(downloadUrl);
            if(!fileSaveDir.exists()) fileSaveDir.mkdirs();
            this.threads = new DownloadThread[threadNum];
            HttpURLConnection conn = (HttpURLConnection) url.
            openConnection();
            conn.setConnectTimeout(6*1000);
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "image/gif, image/jpeg,
            image/pjpeg, image/pjpeg, application/x-shockwave-flash,
            application/xaml+xml, application/vnd.ms-xpsdocument,
            application/x-ms-xbap, application/x-ms-application,
            application/vnd.ms-excel, application/vnd.ms-powerpoint,
```

```
            application/msword, */*");
            conn.setRequestProperty("Accept-Language", "zh-CN");
            conn.setRequestProperty("Referer", downloadUrl);
            conn.setRequestProperty("Charset", "UTF-8");
            conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible;
            MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET CLR 1.1.4322; .NET
            CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.4506.
            2152; .NET CLR 3.5.30729)");
            conn.setRequestProperty("Connection", "Keep-Alive");
            conn.connect();
            printResponseHeader(conn);
            if (conn.getResponseCode()==200) {
                this.fileSize = conn.getContentLength();//根据响应获取文件
                大小
                if (this.fileSize <= 0) throw new RuntimeException("1 无
                法获知文件大小 ");

                String filename=getFileName(conn);
                this.saveFile=new File(fileSaveDir, filename);/* 保存文件*/
                Map<Integer, Integer> logdata = fileService.getData
              (downloadUrl);
                if(logdata.size()>0){
                    for(Map.Entry<Integer, Integer> entry : logdata.
                    entrySet())
                        data.put(entry.getKey(), entry.getValue()+1);
                }
                this.block = this.fileSize / this.threads.length + 1;
                if(this.data.size()==this.threads.length){
                    for (int i = 0; i < this.threads.length; i++) {
                        this.downloadSize += this.data.get(i+1)-
                        (this.block * i);
                    }
                    print("已经下载的长度"+ this.downloadSize);
                }
            }else{
                throw new RuntimeException("2 服务器响应错误 ");
            }
        } catch (Exception e) {
            print(e.toString());
            throw new RuntimeException("3 连接不到下载路径 ");
        }
    }
    /**
     * 获取文件名
     */
    private String getFileName(HttpURLConnection conn) {
        String filename = this.url.toString().substring(this.url.
        toString().lastIndexOf('/') + 1);
        if(filename==null || "".equals(filename.trim())){//如果获取不到文件
        名称
```

```
            for (int i = 0;; i++) {
                String mine = conn.getHeaderField(i);
                if (mine == null) break;
                if("content-disposition".equals(conn.getHeaderFieldKey(i).
                toLowerCase())){
                    Matcher m = Pattern.compile(".*filename=(.*)").
                     matcher(mine.toLowerCase());
                    if(m.find()) return m.group(1);
                }
            }
            filename = UUID.randomUUID()+ ".tmp";//默认取一个文件名
        }
        return filename;
    }

    /**
     *  开始下载文件
     * @param listener 监听下载数量的变化,如果不需要了解实时下载的数量,可以设置为 null
     * @return 已下载文件大小
     * @throws Exception
     */
    public int download(DownloadProgressListener listener) throws
    Exception{
        try {
            if(this.data.size() != this.threads.length){
                this.data.clear();
                for (int i = 0; i < this.threads.length; i++) {
                    this.data.put(i+1, this.block * i);
                }
            }
            for (int i = 0; i < this.threads.length; i++) {
                int downLength = this.data.get(i+1) - (this.block * i);
                if(downLength < this.block && this.data.get(i+1)<this.
                fileSize){ //该线程未完成下载时,继续下载
                    RandomAccessFile randOut = new RandomAccessFile(this.
                     saveFile, "rw");
                    if(this.fileSize>0) randOut.setLength(this.fileSize);

                    randOut.seek(this.data.get(i+1));
                    this.threads[i] = new DownloadThread(this, this.url,
                     randOut, this.block, this.data.get(i+1), i+1);
                    this.threads[i].setPriority(7);
                    this.threads[i].start();
                }else{
                    this.threads[i] = null;
                }
            }
            this.fileService.save(this.downloadUrl, this.data);
            boolean notFinish = true;//下载未完成
            while (notFinish) {// 循环判断是否下载完毕
```

```
                Thread.sleep(900);
                notFinish = false;//假定下载完成
                for (int i = 0; i < this.threads.length; i++){
                    if (this.threads[i] != null && !this.threads[i].
                    isFinish()) {
                        notFinish = true;//下载没有完成
                        if(this.threads[i].getDownLength() == -1){//如果
                        下载失败,再重新下载
                            RandomAccessFile randOut = new
                            RandomAccessFile(this.saveFile, "rw");
                            randOut.seek(this.data.get(i+1));
                            this.threads[i] = new DownloadThread(this,
                            this.url, randOut, this.block, this.data.
                            get(i+1), i+1);
                            this.threads[i].setPriority(7);
                            this.threads[i].start();
                        }
                    }
                }
                if(listener!=null) listener.onDownloadSize(this.
                downloadSize);
            }
            fileService.delete(this.downloadUrl);
        } catch (Exception e) {
            print(e.toString());
            throw new Exception("下载失败");
        }
        return this.downloadSize;
    }
    /**
     * 获取 Http 响应头字段
     * @param http
     * @return
     */
    public static Map<String, String> getHttpResponseHeader
    (HttpURLConnection http) {
        Map<String, String> header = new LinkedHashMap<String, String>();
        for (int i = 0;; i++) {
            String mine = http.getHeaderField(i);
            if (mine == null) break;
            header.put(http.getHeaderFieldKey(i), mine);
        }
        return header;
    }
    /**
     * 打印 Http 头字段
     * @param http
     */
    public static void printResponseHeader(HttpURLConnection http){
        Map<String, String> header = getHttpResponseHeader(http);
```

```
            for(Map.Entry<String, String> entry : header.entrySet()){
                String key = entry.getKey()!=null ? entry.getKey()+ ":" : "";
                print(key+ entry.getValue());
            }
        }

        private static void print(String msg){
            Log.i(TAG, msg);
        }

        public static void main(String[] args) {
        /*  FileDownloader loader = new FileDownloader(context, "http://browse.
            babasport.com/ejb3/ActivePort.exe",
                    new File("D:\\androidsoft\\test"), 2);
            loader.getFileSize();//得到文件总大小
            try {
                loader.download(new DownloadProgressListener(){
                    public void onDownloadSize(int size) {
                        print("已经下载: "+ size);
                    }
                });
            } catch (Exception e) {
                e.printStackTrace();
            }*/
        }

    }
```

下面的类是真正支持下载的线程类。

```
    import java.io.InputStream;
    import java.io.RandomAccessFile;
    import java.net.HttpURLConnection;
    import java.net.URL;

    import android.util.Log;

    public class DownloadThread extends Thread {
        private static final String TAG = "DownloadThread";
        private RandomAccessFile saveFile;
        private URL downUrl;
        private int block;
        /* 下载开始位置 */
        private int threadId = -1;
        private int startPos;
        private int downLength;
        private boolean finish = false;
        private FileDownloader downloader;

        public DownloadThread(FileDownloader downloader, URL downUrl,
         RandomAccessFile saveFile, int block, int startPos, int threadId) {
```

```
        this.downUrl = downUrl;
        this.saveFile = saveFile;
        this.block = block;
        this.startPos = startPos;
        this.downloader = downloader;
        this.threadId = threadId;
        this.downLength = startPos - (block * (threadId - 1));
    }

    @Override
    public void run() {
        if(downLength < block){//未下载完成
            try {
                HttpURLConnection http = (HttpURLConnection) downUrl.
                openConnection();
                http.setRequestMethod("GET");
                http.setRequestProperty("Accept", "image/gif, image/jpeg,
                image/pjpeg, image/pjpeg, application/x-shockwave-flash,
                application/xaml+xml, application/vnd.ms-xpsdocument,
                application/x-ms-xbap, application/x-ms-application,
                application/vnd.ms-excel, application/vnd.ms-powerpoint,
                application/msword, */*");
                http.setRequestProperty("Accept-Language", "zh-CN");
                http.setRequestProperty("Referer", downUrl.toString());
                http.setRequestProperty("Charset", "UTF-8");
                http.setRequestProperty("Range", "bytes=" + this.startPos
                + "-");
                http.setRequestProperty("User-Agent", "Mozilla/4.0
                (compatible; MSIE 8.0; Windows NT 5.2; Trident/4.0; .NET
                 CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .
                 NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)");
                http.setRequestProperty("Connection", "Keep-Alive");

                InputStream inStream = http.getInputStream();
                int max = 1024 * 1024;
                byte[] buffer = new byte[max];
                int offset = 0;
                print("线程 " + this.threadId + "从位置"+ this.startPos+ "
                开始下载 ");
                while (downLength < block && (offset = inStream.read(buffer,
                0, max)) != -1) {
                    saveFile.write(buffer, 0, offset);
                    downLength += offset;
                    downloader.update(this.threadId, block * (threadId -
                    1) + downLength);
                    downloader.saveLogFile();
                    downloader.append(offset);
                    int spare = block-downLength;//求剩下的字节数
                    if(spare < max) max = (int) spare;
                }
```

```
                saveFile.close();
                inStream.close();
                print("线程 " + this.threadId + "完成下载 ");
                this.finish = true;
                this.interrupt();
            } catch (Exception e) {
                this.downLength = -1;
                print("线程"+ this.threadId+ ":"+ e);
            }
        }
    }
    private static void print(String msg){
        Log.i(TAG, msg);
    }
    /**
     * 下载是否完成
     * @return
     */
    public boolean isFinish() {
        return finish;
    }
    /**
     * 已经下载的内容大小
     * @return 如果返回值为-1,代表下载失败
     */
    public long getDownLength() {
        return downLength;
    }
}
```

下面为监听器接口，会实时显示下载的大小，在实际使用时建议采用匿名类的方式构建此接口。

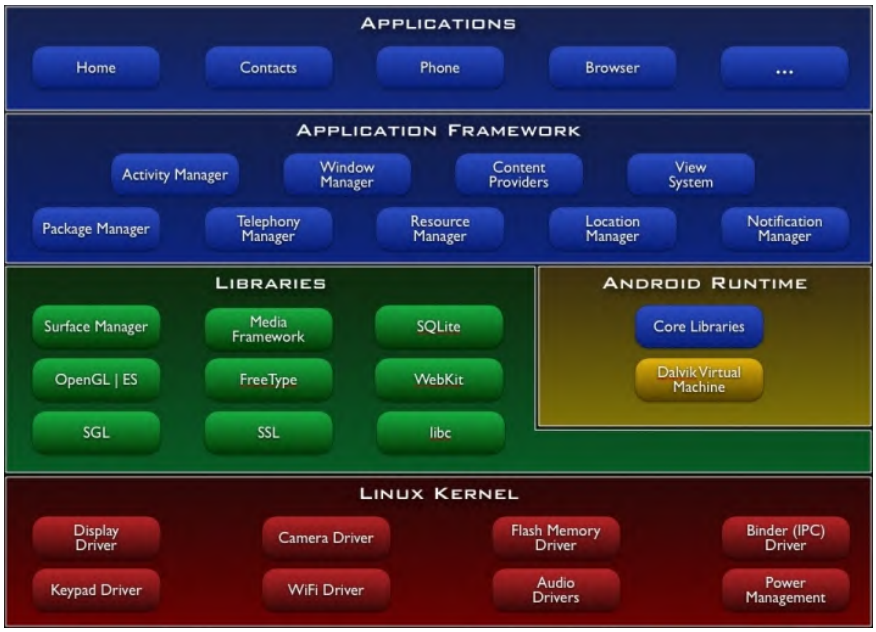
```
public interface DownloadProgressListener {
    public void onDownloadSize(int size);
}
```

上面的 3 个文件在一起就构建起了一个迷你型的 Android 下载框架，这个下载框架可以用于下载任何类型的二进制文件，以后应用程序需要下载的时候直接使用该代码即可。其中 IoC 非常直接地体现了 DownloadProgressListener ，在使用的时候只需要传入该接口一个实现实例即可自动获取实时的下载长度。

二、精通 Android 体系架构、MVC、常见的设计模式、控制反转（IoC）

1．Android 体系架构

请看某著名 IT 公司的一则招聘信息中的一条要求：“熟悉 Android 系统架构及相关技术，1 年以上实际 Android 平台开发经验”，里面非常明确地提到了要求熟练掌握 Android 系统架构，这从某种程度上说明了对 Android 体系架构的理解的重要性，下图为 Android 体系结构图。



很明显，上图包含 4 个主要的层次。

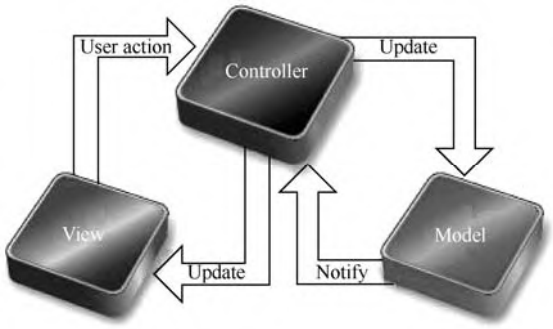
- Linux Kernel：负责硬件的驱动程序、网络、电源、系统安全及内存管理等功能。
- Libraries 和 Android Runtime：Libraries，即 C/C++函数库部分，大多数都是开放源代码的函数库，如 WebKit，该函数库负责 Android 网页浏览器的运行，例如标准的 C 函数库 Libc、OpenSSL、SQLite 等，当然也包括支持游戏开发 2D SGL 和 3D OpenGL | ES，在多媒体方面由 MediaFramework 框架来支持各种影音和图形文件的播放与显示，例如 MPEG4、H.264、MP3、AAC、AMR、JPG 和 PNG 等众多的多媒体文件格式。Android 的 Runtime 负责解释和执行生成的 Dalvik 格式的字节码。
- Application Framework（应用软件架构）：Java 应用程序开发人员主要使用该层封装好的 API 进行快速开发。
- Applications：该层是 Java 的应用程序层，Android 内置的 Google Maps、E-mail、即时通信工具、浏览器、MP3 播放器等处于该层，Java 开发人员开发的程序也处于该层，而且和内置的应用程序具有平等的位置，可以调用内置的应用程序，也可以替换内置的应用程序。

上面的 4 个层次，下层为上层服务，上层需要下层的支持并调用下层的服务，这种严格分层的方式带来极大的稳定性、灵活性和可扩展性，使得不同层的开发人员可以按照规范专心于特定层的开发。

Android 应用程序使用框架的 API 并在框架下运行，这就带来了程序开发的高度一致性，另一方面也告诉我们，要想写出优质高效的程序就必须对整个 Application Framework 进行非常深入的理解。精通 Application Framework，就可以真正地理解 Android 的设计和运行机制，也就更能够驾驭整个应用层的开发。

2．MVC 模式

Android 官方建议应用程序的开发采用 MVC 模式。何谓 MVC？先看看下图。



MVC 是 Model，View，Controller 的缩写，从上图可以看出 MVC 包含 3 个部分。

- 模型（Model）对象：应用程序的主体部分，所有的业务逻辑都应该写在该层。
- 视图（View）对象：应用程序中负责生成用户界面的部分。也是在整个 MVC 架构中用户唯一可以看到的一层，接收用户的输入，显示处理结果。
- 控制器（Control）对象：根据用户的输入，控制用户界面数据显示及更新 Model 对象状态。控制器更重要的一种功能是导航功能，响应用户出发的相关事件，交给 Model 处理。

Android 鼓励弱耦合和组件的重用，在 Android 中 MVC 的具体体现如下。

（1）视图层（View）：一般采用 XML 文件进行界面的描述，使用时可以方便地引入，当然，如何对 Android 了解比较多，就一定可以想到在 Android 中也可以使用 JavaScript+HTML 等的方式作为 View 层，当然这里需要进行 Java 和 JavaScript 之间的通信，幸运的是，Android 提供了它们之间非常方便的通信。

（2）控制层（Controller）：Android 控制层的重任通常落在了众多的 Activity 肩上，这句话也就暗含了不要在 Activity 中写代码，要通过 Activity 交割 Model 业务逻辑层处理，这样做的另外一个原因是 Android 中的 Activity 的响应时间是 5 s，如果耗时的操作放在这里，程序就很容易被回收掉。

（3）模型层（Model）：对数据库、网络等的操作都应该在 Model 里面处理，当然对业务计算等操作也必须放在该层。

3．设计模式和 IoC（控制反转）

毫无疑问，Android 之所以能够成为一个开放的气象万千的移动操作平台，与设计模式的精妙应用是分不开的，只要观察稍加用心，就会发现在 Android 中到处都是设计模式或者设计模式的联合运用，以下设计模式是游刃有余地驾驭 Android 所必须掌握的：

- Template Method 模式；
- Factory Method 模式；

- Observer 模式；
- Abstract Factory 模式；
- Adapter 模式；
- Composite 模式；
- Strategy 模式；
- State 模式；
- Proxy 模式；
- Bridge 模式；
- Iterator 模式；
- Mediator 模式；
- Facade 模式。

Android 框架魅力的源泉在于 IoC，在 Android 应用开发的过程中你会时刻感受到 IoC 带来的巨大方便，就拿 Activity 来说，下面的函数是框架自动调用的。

```
protected void onCreate(Bundle savedInstanceState);
```

不是程序编写者主动去调用，反而是用户写的代码被框架调用，这也就反转了！当然 IoC 本身的内涵远远不止这些，但是从这个例子中也可以看出 IoC 带来的巨大好处。此类的例子在 Android 随处可见，例如，数据库的管理类，Android 中 SAX 的 Handler 的调用等。有时，甚至需要自己编写简单的 IoC 实现，上面展示的多线程现在就是一个说明。

三、编写可重用、可扩展、可维护、灵活性高的代码

Android 应用程序的开发使用 Java 编写，在架构上使用 MVC，鼓励组件之间的弱耦合。开发出编写可重用、可扩展、可维护、灵活性高的代码需要遵循以下原则。

- “开—闭”原则（OCP）：一个软件实体应当对扩展开放，对修改关闭。这个原则说的是，在设计一个模块的时候，应当使这个模块可以在不被修改的前提下被扩展。换言之，应当允许在不必修改源代码的情况下改变这个模块的行为。
- 里氏代换原则（LSP）：一个软件实体如果使用的是一个基类的话，那么一定使用于其子类，而且它根本不能察觉出基类对象和子类对象的区别。
- 依赖倒转原则（DIP）：要依赖于抽象，不要依赖于具体。
- 接口隔离原则（ISP）：使用多个专门的接口比使用单一的总接口要好。一个类对另外一个类的依赖性应当是建立在最小的接口上的。
- 合成/聚合复用原则（CARP）：又称合成复用原则（CRP），就是在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分；新的对象通过向这些对象的委派达到复用已有功能的目的。简而言之就是：要尽量使用合成/聚合，尽量不使用继承。

- 迪米特法则（LoD）：又称最少知识原则（LKP），是说一个对象应当对其他对象尽可能少的了解。狭义的迪米特法则是指如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果其中一个类需要调用另一个类，可以通过第三者转发这个调用。广义的迪米特法则是指一个模块设计得好坏的一个重要的标志就是该模块在多大的程度上将自己的内部数据与实现有关的细节隐藏起来。信息的隐藏非常重要的原因在于，它可以使各个子系统之间脱耦，从而允许它们独立地被开发、优化、使用、阅读及修改。

灵活地使用设计模式可以在面对千变万化的业务需求时编写出可重用、可扩展、可维护、灵活性高的代码。

当然，由于 Android 是运行在移动设备上的，而移动设备的处理能力是有限的，所以有时必须在编写可重用、可扩展、可维护、灵活性高的代码与高效的代码之间做出适当的平衡。

四、高效地编写高效的代码

高效快速地编写代码和编写高效率执行的代码很多时候是对立的。多数情况下，想快速地开发，代码的执行效率往往就会慢下来；想编写高效的代码，开发速度就会慢下来。

不重复发明轮子和发明新的轮子是高效地编写高效代码的正确道路。

五、学会至少一门服务器端开发技术

可能有读者会问：学习 Android 应用程序开发为什么还需要学会至少一门服务器端开发技术？答案如下：一方面 Android 号称是首个为移动终端打造的真正开放和完整的移动软件。作为一种移动终端，必须与服务器端结合才能发挥巨大的作用。简言之，需要云+云端的方式。Android 是为移动互联网量身打造的，移动互联网时代的服务模式是“手机终端+互联网络+应用软件”，移动互联网时代应用技术之一的 Android 只是用于开发移动终端软件，而服务端技术用于开发互联网应用，所以未来移动互联网时代软件的主流应用模式将是“手机客户端+互联网应用服务端”，这种模式要求做移动互联网开发的程序员不但要掌握像 Android 这样的手机终端软件技术，还要掌握开发互联网应用的服务器端技术。目前，软件企业普遍存在这样的问题，即做移动互联网 Android 终端软件开发的程序员不了解 Web 应用技术，而做 Web 应用的程序员不了解移动互联网终端技术，这样就导致客户端与服务端在衔接上出现了问题。目前的现状是：既掌握移动互联网 Android 终端技术，又掌握 Web 应用技术的程序员比较稀缺，随着中国步入移动互联网时代，企业对这种移动互联网综合性人才的需求很旺盛。如果不了解 Web 应用技术，最终会遇到技术和发展的瓶颈；另一方面，Google 联合 OHA 推出 Android 的真正优势之一也在于和移动互联网结合，用意之一也是想开辟新的终端去使用 Google 的优势服务。

服务器端开发技术目前主流的有 Sun 的 Java EE、微软的.NET，以 PHP 和 MySQL 为代表的开源的 LAMP 体系，我们该选择哪一种呢？从理论上讲，很多人倾向于选择 Java EE，毕竟它们都是使用 Java 作为开发语言的，但是很多人面对 Java EE 众多的框架就望而生畏，其实在学习 Java EE 的时候可以从 Struts 入手，随着业务的需求逐步深入。当然，选择微软的.NET 也是不错的选择，毕竟该技术体系也占有很大的市场份额。其实，笔者认为，LAMP 可能是获得

最高“性价比”的选择。一方面 PHP 是现在 Web 方面的主流语言，大多数新型网站尤其是创业性质的网站一般都会选用 PHP 作为服务端开发语言；另一方面，前面也说过，Android 是为移动互联网而生的，两者可以达到完美的契合。

六、以 Android 软件和硬件深度整合的视角看待 Android

2011 年 8 月份 Google 收购摩托罗拉移动部门，使得 Android 软件和硬件一体化成为趋势。该事件将导致以硬件终端、Android 系统软件、Android 应用软件、网络云服务、网络营销为关键组成部分的全方位的激烈竞争。而其核心在于 Android 的软、硬件深度整合。所谓深度整合是指，Android 系统和应用软件要充分发挥硬件的优势，同时把云服务整合到 Android 系统中，应用开发人员可以通过应用框架去调用云服务。笔者会就关于 Android 软、硬件深度整合、云和端专门写一本书来和大家分享该方面的智慧和具体实现，敬请关注。