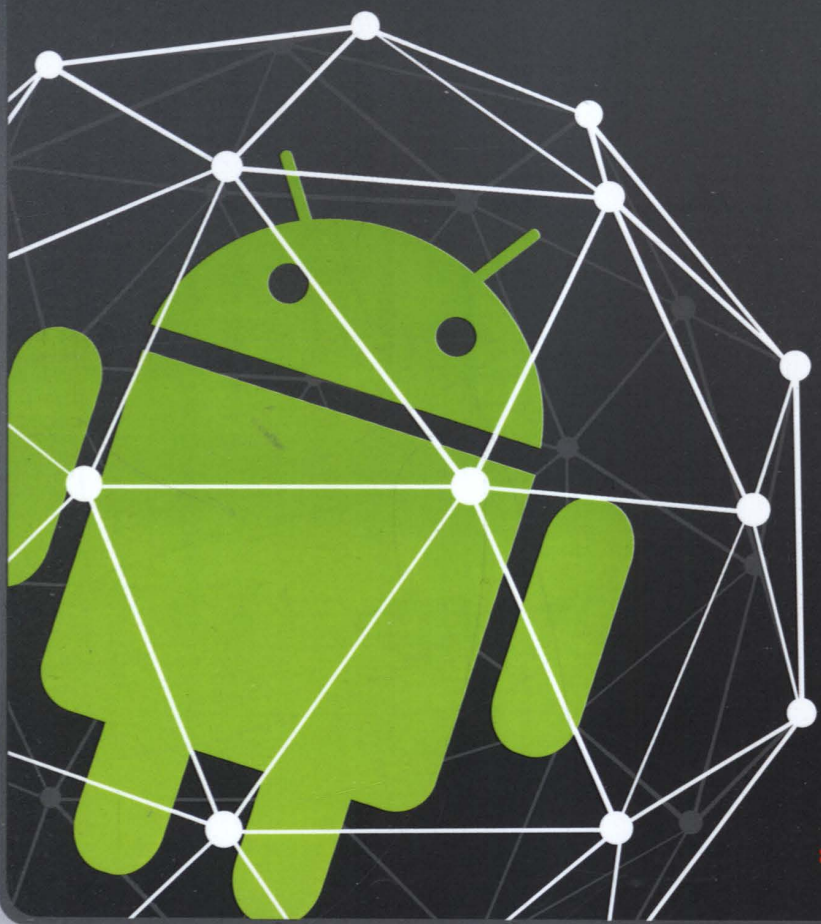
·移动平台开发书库·

本书配套电子课件和源代码
可从www.golden-book.com下载

Android 网络开发从入门到精通

代林峰 等编著



机械工业出版社
CHINA MACHINE PRESS

移动平台开发书库

Android 网络开发从入门到精通

代林峰　等编著

机 械 工 业 出 版 社

在 Android 系统从诞生到现在的短短几年时间里，它凭借操作的易用性和开发的简洁性，已牢牢占据智能手机操作系统市场占有率榜首的位置。而在 Android 应用开发领域中，网络开发一直是贯穿 Android 知识体系的核心内容之一。本书全部内容分为四篇，共计 17 章，循序渐进地讲解了 Android 网络开发方面的知识。本书从搭建开发环境和核心框架分析讲起，依次讲解了 Android 系统概述，Android 网络开发基础，Java 中的网络通信基础，下载、上传数据，Socket 数据通信，处理 XML 数据，WebKit 浏览网页，开发移动网页，开发蓝牙应用程序，开发 Wi-Fi 应用程序，NFC 近场通信技术详解，开发电子邮件应用程序，Android 典型网络应用实践，开发移动微博应用程序，开发 Web 版的电话本管理系统，开发移动微信系统，开发仿陌陌交友系统等高级知识。本书几乎涵盖了 Android 网络开发中的所有主要内容，并且全书内容言简意赅，讲解方法通俗易懂、详细，不但适合应用开发高手们的学习，也特别适合初学者的系统学习。

本书适合 Android 初学者、Android 爱好者、Android 网络开发人员和移动浏览器开发人员，也可以作为相关培训学校和大专院校相关专业的教学用书。

图书在版编目（CIP）数据

Android 网络开发从入门到精通 / 代林峰等编著. —北京：机械工业出版社，2015.10

（移动平台开发书库）

ISBN 978-7-111-52203-4

Ⅰ. ①A… Ⅱ. ①代… 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（2015）第 271402 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：丁 诚 责任编辑：丁 诚 张 恒

责任校对：张艳霞 责任印制：李 洋

保定市中画美凯印刷有限公司印刷

2016 年 10 月第 1 版 • 第 1 次印刷

184mm×260mm • 34 印张 • 841 千字

3001－4500 册

标准书号：ISBN 978-7-111-52203-4

定价：85.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务 网络服务

服务咨询热线：（010）88361066 机 工 官 网：www.cmpbook.com

读者购书热线：（010）68326294 机 工 官 博：weibo.com/cmp1952

（010）88379203 教育服务网：www.cmpedu.com

封面无防伪标均为盗版 金 书 网：www.golden-book.com

前　言

Android 是一款于 2007 年 11 月 5 日发布的基于 Linux 平台的开源手机操作系统，该平台由操作系统、中间件、用户界面和应用软件组成，是首个专为移动终端而打造的移动软件。根据国际数据公司（IDC）公布的统计数据，在 2014 年第一季度，Android 系统和 iOS 系统所占的装机量已达到所有智能手机出货量的 92.3%。在 2014 年头三个月，安装 Android 系统的智能手机数量升至 1.821 亿部。我们有理由相信，在未来一段时间内，Android 将依旧牢牢地占据着智能手机操作系统第一的位置。

市场需求分析

较高的市场占有率造就了更多开发人员关注这款操作系统，当然也不乏很多初学者，所以也就很自然地成就了相关书籍的畅销。但是在市面中已有的书籍中，大多数是入门级的教材，而关于 Android 网络开发领域书籍屈指可数，Android 网络开发领域的专业级书籍更是寥寥无几。

只有更加专业才能造就 Android 开发的殿堂级高手！为了让广大初学者可以对 Android 网络开发有一个更加深入的认识，而不是停留在入门级而止步不前。本书对 Android 网络开发方面的知识进行了细致的分析，“提炼”出了 Android 系统开发的本质，并依此为基础，学以致用地讲解了在现实中开发典型网络项目的实现流程。

本书的内容

本书全部内容共分为 4 篇，共计 17 章，循序渐进地讲解了 Android 网络应用开发方面的知识。本书从搭建开发环境和核心框架分析讲起，依次讲解了 Android 技术概述，Android 技术核心框架分析，Java 中的网络通信基础，WebKit 浏览网页，开发移动网页，开发蓝牙应用程序，开发 Wi-Fi 应用程序，NFC 近场通信技术详解，开发电子邮件应用程序，Android 网络典型应用实践，开发移动微博应用程序，开发 Web 版的电话本管理系统，移动微信系统，仿陌陌交友系统，下载、上传数据，Socket 数据通信，处理 XML 数据等高级知识。本书几乎涵盖了 Android 网络应用开发中的所有主要内容，并且全书内容言简意赅，讲解方法通俗易懂、详细，不但适合应用开发高手们的学习，也特别有利于初学者学习并消化。

本书的版本

Android 系统自 2008 年 9 月发布第一个版本 1.1 以来，截至 2015 年 10 月发布的最新版本 6.0，一共存在十多个版本。由此可见，Android 系统升级频率较快，一年之中至少有两个新版本诞生。但是如果过于追求新版本，会造成力不从心的后果。所以在此建议广大读者：“不必追求最新的版本，我们只需关注最流行的版本即可”。据官方统计，截至 2015 年 10 月，占据前三位的版本分别是 Android 4.2，Android 4.4 和 Android 5.0。

2014 年 10 月，谷歌 I/O 大会在旧金山开幕。会上谷歌发布了 Android 5.0 系统，其正式版本于 2014 年 10 月 16 日推出。本书的内容以编者撰稿时的最新版本 Android 5.0 为基础，并且兼容了 Android 4.4 及其以前的版本，详细讲解了 Android 网络应用开发的相关知识。

本书特色

本书内容丰富、细致、全面。我们的目标是通过本书，提供多本图书的价值，读者可以根据自己的需求有选择地阅读。在内容的编写上，本书具有以下特色。

（1）结构合理

从用户的实际需求出发，科学安排知识结构，内容由浅入深，叙述清楚。全书详细地讲解了和 Android 网络应用开发有关的知识，内容循序渐进，由浅入深。

（2）遵循“理论介绍—演示实例—综合演练”这一主线

为了使广大读者彻底弄清楚 Android 网络应用开发的每一个知识点，在讲解时依次剖析了基本理论、演示实例分析、综合实战演练等内容。遵循了从理论到实践这一学习过程，实现了实践教学这一目标。

（3）易学易懂

本书内容条理清晰、语言简洁，可以帮助读者快速掌握每个知识点。使读者既可以按照本书编排的章节顺序进行学习，也可以根据自己的需求对某一章节进行针对性的学习。

（4）实用性强

本书彻底摒弃枯燥的理论和简单的操作，注重实用性和可操作性，通过简洁的语言和细致的描述，详细讲解了各个知识点的基本知识。

（5）内容全面

本书是市面上内容较全面的一本 Android 网络应用开发书，无论是开发环境搭建，还是各个常用、常见的网络系统，在本书中都有讲述。

读者对象

- Android 编程的初学者。
- 大中专院校的教师和学生。
- Android 编程爱好者。
- 相关培训机构的教师和学员。
- 从事 Android 网络开发的程序员。

致谢

本书的编写人员有代林峰、管西京、周秀、张余、李佐彬、王梦、王书鹏、唐凯、关立勋、张建敏、杨靖宇、谭贞军、杨絮、刘英田、高秀云、任杰、张子帝、黄河、孟娜、杨国华、王南荻、翟明、焦甜甜、张储、刘继虎。在本书的编写过程中，始终本着科学、严谨的态度，力求精益求精，但错误、疏漏之处在所难免，敬请广大读者批评指正。

编　者

目　录

第二篇 核心技术篇

第三篇 技术提高篇

第四篇 综合实战篇

第一篇　基础知识篇

第 1 章　Android 系统概述

Android 是一款用于移动智能设备（手机、平板电脑等）的操作系统，以 Linux 系统作为内核架构。从 2011 年开始到现在，Android 系统一直占据全球智能手机市场占有率第一的宝座。在本章的内容中，将简单地介绍 Android 的发展历程和背景，并介绍搭建 Android 网络应用开发环境的基本知识，为读者进行本书后面知识的学习打下基础。

1.1　智能手机系统介绍

智能手机是指具有像个人计算机那样强大的功能，拥有独立的操作系统，用户可以自行安装应用软件、游戏等第三方提供的应用程序，并且可以通过移动网络接入到无线网络中。在本节的内容中，将详细讲解智能手机的基本知识。

1.1.1　何谓智能手机

一般来说，智能手机必须具备如下的功能。

（1）操作系统必须支持新应用的安装。

（2）芯片拥有高速处理的能力。

（3）可以播放各种音频和视频文件。

（4）具有大容量存储芯片和存储扩展能力。

根据上述功能要求，手机联盟公布了智能手机的主要特点，具体说明如下。

（1）具备普通手机的所有功能，例如拨打、接听电话和收发短信等。

（2）是一个开放性的操作系统，可以安装第三方应用程序，从而实现功能的无限扩展。

（3）具备上网功能，例如可以浏览网页。

（4）具备 PDA 的功能，例如能够实现个人信息管理、日程记事、任务安排、多媒体应用、浏览网页等功能。

（5）扩展性能强，可以根据个人需要扩展手机的功能。

1.1.2　当前主流的智能手机操作系统

在 Android 系统诞生之前，在市面中已经存在了很多优秀的智能手机产品，例如诺基亚中常见的 Symbian 系列和微软的 Windows Mobile 系列等。在当今市面中主流的智能手机系统包括 Windows Phone、iOS 和本书的主角 Android。

1．Windows Phone

微软公司于 2010 年 10 月 11 日正式发布了智能手机操作系统 Windows Phone，并将其使用接口称为“Modern”。Windows Phone 是微软公司的一款杰出触控产品，它将微软旗下的 Xbox Live

游戏、Xbox Music 音乐与独特的视频体验集成至手机中。2011 年 2 月，诺基亚与微软达成全球战略同盟并深度合作共同研发。2011 年 9 月 27 日，微软发布 Windows Phone 7.5。2012 年 6 月 21 日，微软正式发布 Windows Phone 8，采用和 Windows 8 相同的 Windows NT 内核。

2014 年 4 月，微软在 Build2014 开发者大会发布 Windows Phone 8.1。在 2014 年 6 月份，Windows Phone 8 手机部分用户将能收到 Windows Phone 8.1 预览版更新。2015 年 7 月 30 日，微软向全球用户推送 Windows Phone 10。

2．iOS

iOS 作为苹果移动设备 iPhone 和 iPad 的操作系统，在 App Store 的推动之下，成为了世界上引领潮流的操作系统之一。iOS 系统原名为“iPhone OS”，直到 2010 年 6 月 7 日 WWDC 大会上宣布改名为“iOS”。iOS 的用户界面的概念基础是能够使用多点触控直接操作。控制方法包括滑动、轻触开关及按键。与系统交互包括滑动（Swiping）、轻按（Tapping）、挤压（Pinching，通常用于缩小）及反向挤压（Reverse Pinching，通常用于放大）。此外通过其自带的加速器，可以令其旋转设备改变其 y 轴以令屏幕改变方向，这样的设计令 iPhone 更便于使用。

2015 年 10 月，苹果公司推出了版本 iOS 9。

3．Android

Android 系统于 2007 年 11 月 5 日发布，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。

根据国际数据公司（IDC）2015 年 2 月公布的新数据，在 2014 全年第一季度，Android 和 iOS 系统的装机量占到所有智能手机出货量的 96.3%。其中，Android 出货量为 10.59 亿部，同比增长 32%，市场份额为 81.5%，去年同期为 78.7%；iPhone 出货量为 1.927 亿部，同比去年增长 25.6%，市场份额为 14.8%，去年同期为 15.1%。

截止到本书完稿时，Android 系统的最新版本是 Android 6.0，市面中最主流的系统是 Android 5.0。

1.2 Android 5.0 的新功能

美国太平洋时间 2014 年 10 月 15 日 0 时，谷歌（Google）发布了 Android 5.0（Lollipop），如图 1-1 所示。

图 1-1　Google 发布的 Android 5.0

2014 年 11 月 3 日，Android 5.0 Lollipop（棒棒糖）面向用户正式推出。开发者已经可以下载 Android 5.0 Platform（API Level 21）来开发和测试 Android 5.0 应用，并能向 Google Play 发布 Android 5.0 专属的应用程序。

和传统版本相比，Android 5.0 的突出特性如下。

（1）全新的 Material 界面设计

Android 5.0 Lollipop 界面设计的灵感来源于自然、物理学以及基于打印效果的粗体、图表化的设计，换句话说，它的设计是一种基于高品质纸张的效果——扁平、易于操作。

（2）打造健全的 Android 生态系统

Android 将不仅仅是一款智能手机操作系统，而是将成为一款健全的系统出现在所有的电子屏幕中，例如出现在平板电脑上、笔记本电脑上、电视机上、汽车上、手表上、家用电器上等。

（3）全新设计的通知系统

除了界面有较大改变之外，谷歌还调整了通知中心的信息展示规则——最重要的信息将被显示出来，而次要信息则会被隐藏。当然，如果需要查看全部信息，则继续向下滑动即可。

（4）64 位 ART 编译器

从 Android 5.0 版本开始，Dalvik 编译器即将“退休”，系统默认的运行环境是最新的、更高效的 ART。同时，采用了 ART 环境后，Android 能够兼容 ARM、X86 和 MIPS 等架构。

（5）提升了电池续航能力

从 Android 5.0 版本开始，在 Project Volta 中加入了新的工具来帮助开发者更容易地发现并找出应用程序为什么，或者说在哪里特别耗电。还有新的工具来帮助开发者确定某些进程不会被触发（在电池电量不多的时候）。

1.3　Android 的巨大优势

为什么 Android 系统能在这么多的智能系统中脱颖而出，成为市场占有率第一的手机操作系统呢？要分析其原因，需要先了解它的优势，分析究竟是哪些优点获得了厂商和消费者的青睐。在本节的内容中，将详细讲解 Android 系统自身的巨大优势。

1.3.1　优点一——系出名门

Android 系统出身于 Linux，是一款开源的手机操作系统。在 Android 系统取得巨大成功之后，各大手机联盟纷纷加入，这个联盟由包括中国移动、摩托罗拉、高通、HTC 和 T-Mobile 在内的 30 多家技术和无线应用的领军企业组成。通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作伙伴关系，借助建立标准化、开放式的移动电话软件平台，在移动产业内形成了一个开放式的生态系统。

1.3.2　优点二——强大的开发团队

Android 的研发队伍阵容强大，包括 Google、摩托罗拉、HTC（宏达电子）、PHILIPS、T-Mobile、高通、魅族、三星、LG 以及中国移动在内的 34 家企业，这都是在手机领域中享誉盛名的企业。它们都将基于该平台开发手机的新型业务，这样以来应用之间的通用性和互联性将在最大程度上得到保证。并且还成立了手机开放联盟，联盟中的成员名单如下。

1．手机制造商

宏达国际电子（HTC）（Palm 等多款智能手机的代工厂），摩托罗拉（美国最大的手机制造商之一），韩国三星电子（仅次于苹果的全球第二大手机制造商），韩国 LG 电子，中国移动（全球最大的移动运营商之一），日本 KDDI（2900 万用户），日本 NTT DoCoMo（5200 万用户），美国 Sprint Nextel（美国第三大移动运营商，5400 万用户），意大利电信（意大利主要的移动运营商之一，3400 万用户），西班牙 Telefónica（在欧洲和拉美有 1.5 亿用户），T-Mobile（德意志电信旗下公司，在美国和欧洲有 1.1 亿用户）。

2．半导体公司

Audience Corp（声音处理器公司），Broadcom Corp（无线半导体主要提供商），英特尔（Intel），Marvell Technology Group，Nvidia（图形处理器公司），SiRF（GPS 技术提供商），Synaptics（手机用户界面技术），德州仪器（Texas Instruments），高通（Qualcomm），惠普 HP（Hewlett-Packard Development Company,L.P）。

3．软件公司

Aplix，Ascender，eBay 的 Skype，Esmertec，Living Image，NMS Communications，Noser Engineering AG，Nuance Communications，PacketVideo，SkyPop，Sonix Network，TAT-The Astonishing Tribe，Wind River Systems。

1.3.3 优点三——诱人的奖励机制

谷歌为了提高程序员的开发积极性，不但为他们提供了一流的硬件设置，一流的软件服务，而且还提供了振奋人心的奖励机制，例如为了吸引更多的用户使用 Android 进行开发，已经成功举办了奖金为 1000 万美元的开发者竞赛。鼓励开发人员开发出创意十足、非常有用的软件。这种大赛对于开发人员来说，不但能锻炼自己的开发水平，而且高额的奖金也是他们学习的动力。

为了能让 Android 平台吸引更多的关注，谷歌发布了官方 Android 软件下载商店 Android Market，地址是 http://www.Android.com/market/，允许开发人员将应用程序在上面发布，也允许 Android 用户随意下载获取自己喜欢的程序。作为开发者，需要申请开发者账号，申请后才能将自己的程序上传到 Android Market，并且可以对自己的软件进行定价。所以说，只要你的软件程序足够吸引人，你就可以获得很好的金钱回报，从而达到学习、赚钱两不误。

1.3.4 优点四——开源

对开发人员和手机厂商来说，开源意味着 Android 是完全免费使用的。因为源代码公开的原因，所以激发了世界各地无数程序员的热情。于是很多手机厂商都纷纷采用 Android 作为自己产品的操作系统。因为免费，所以降低了成本，提高了利润。而对于开发人员来说，众多厂商采用 Android 系统就意味着人才需求大，所以纷纷加入到 Android 开发大军中来。

1.4 搭建 Android 应用开发环境

在开发 Android 应用程序之前，首先要搭建一个对应的开发环境。而在搭建开发环境前，需要了解安装开发工具所需要的硬件和软件配置条件。在本节的内容中，将详细讲解搭建

Android 应用开发环境的基本知识。

1.4.1　安装 Android SDK 的系统要求

在搭建之前，先确定 Android 应用程序开发对开发环境的要求，具体如表 1-1 所示。

表 1-1　开发系统所需参数

项　　目	版 本 要 求	说　　明	备　　注
操作系统	Windows XP 或以上，或 Vista Mac OS X 10.4.8 或以上，或 Linux Ubuntu Drapper 或以上	根据自己的计算机自行选择	选择自己最熟悉的操作系统
软件开发包	Android SDK	选择最新版本的 SDK	截止到目前，最新手机版本是 Android 6.0
IDE	Eclipse IDE+ADT	Eclipse3.3 (Europa) 或以上 ADT(Android Development Tools)开发插件	选择“for Java Developer”
其他	JDK Apache Ant	Java SE Development Kit 5 或 6 Linux 和 Mac 上使用 Apache Ant 1.6.5+, Windows 上使用 1.7+版本	单独的 JRE 是不可以的，必须要有 JDK，不兼容 GNU Java 编译器

Android 工具是由多个开发包组成的，具体说明如下。

- JDK：可以到网址 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下载。
- Eclipse+Android SDK：可以到 Android 官方网址 http://developer.android.com 下载包含 Eclipse 和 Android SDK 的压缩包。
- 对应的开发插件，最为常用的是 Eclipse ADT 插件。

1.4.2　安装 JDK

JDK（Java Development Kit）是整个 Java 的核心，包括了 Java 运行环境、Java 工具和 Java 基础的类库。在安装 JDK 之前需要先获得 JDK，获得 JDK 的操作流程如下。

（1）登录 Oracle 官方网站，官方下载页面网址为 http://developers.sun.com/downloads/，如图 1-2 所示。

图 1-2　Oracle 官方下载页面

（2）在图 1-2 中可以根据个人需要来选择需要下载的版本，下载界面效果如图 1-3 所示。

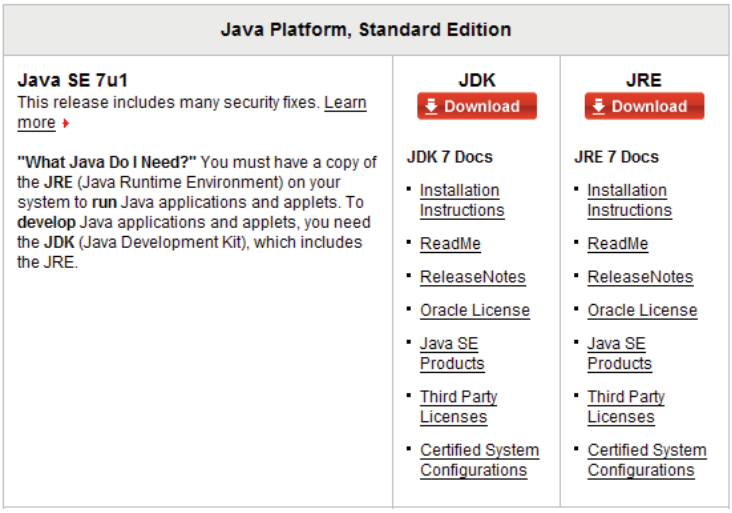


图 1-3　JDK 下载页面效果

（3）在图 1-2 中单击 JDK 下方的 Download 按钮，在弹出的新界面中选择将要下载的 JDK 版本，编者在此选择的是 Windows x86 版本，如图 1-4 所示。

（4）下载完成后双击下载的“.exe”文件开始安装，将弹出“安装向导”对话框，在此单击“下一步”按钮，如图 1-5 所示。

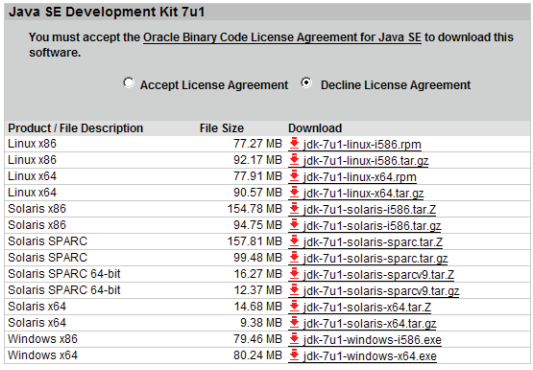


图 1-4　选择 Windows x86 版本

图 1-5　“安装向导”对话框

（6）弹出“安装路径”对话框，在此选择文件的安装路径，如图 1-6 所示。

（7）在此可以设置 JDK 的安装路径单击“更改”按钮可以自定义设置要安装的位置。单击“下一步”按钮开始在安装路径解压缩下载的文件，如图 1-7 所示。

（8）完成后弹出“目标文件夹”对话框，在此单击“更改”按钮可以自定义设置要安装的位置，如图 1-8 所示。

（9）单击“下一步”按钮后开始正式安装，如图 1-9 所示。

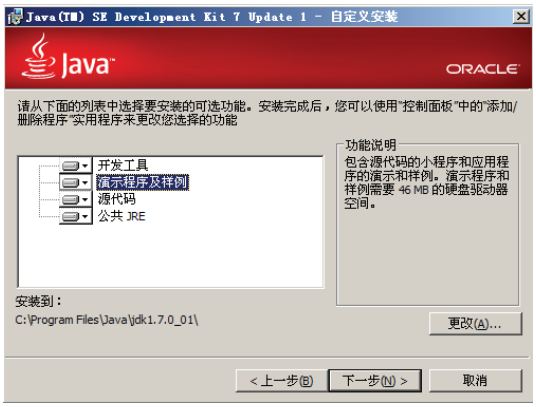


图 1-6　“安装路径”对话框

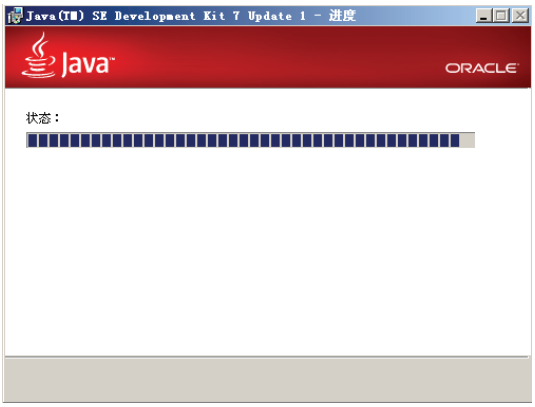


图 1-7　解压缩下载的文件

图 1-8　“目标文件夹”对话框

图 1-9　继续安装

（10）完成后弹出“完成”对话框，单击“完成”按钮后完成整个安装过程，如图 1-10 所示。

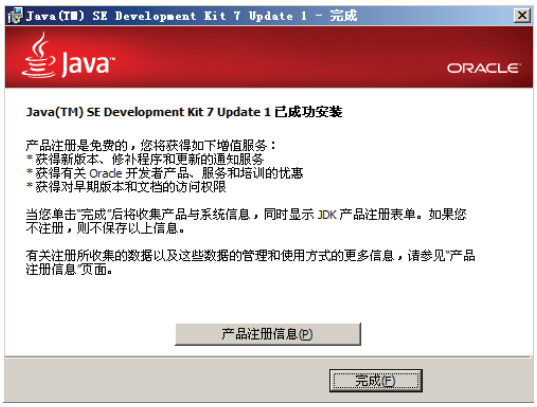


图 1-10　完成安装过程

完成安装后可以检测是否安装成功，检测方法是依次单击计算机中的“开始”→“运行”

按钮，在弹出框中输入“cmd”命令并按〈Enter〉键，在打开的 CMD 窗口中输入命令：

```
java –version
```

如果显示如图 1-11 所示的提示信息，则说明安装成功。

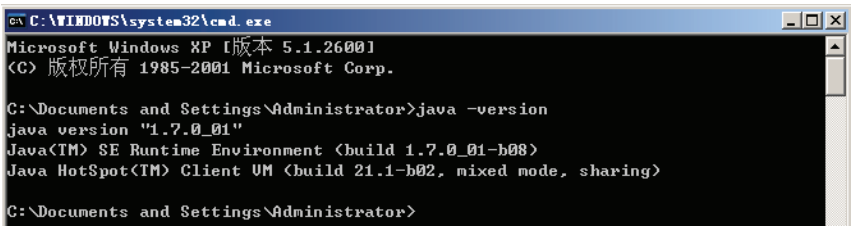


图 1-11　CMD 窗口

如果检测没有安装成功，需要将其目录的绝对路径添加到系统的 PATH 中。具体做法如下。

（1）右键依次单击“我的电脑”→“属性”→“高级”，单击下面的“环境变量”，在下面的“系统变量”处选择“新建”，在变量名中输入 JAVA_HOME，变量值中输入刚才的目录，比如设置为 “C:\Program Files\Java\jdk1.7.0_01”，如图 1-12 所示。

图 1-12　设置系统变量

（2）再次新建一个变量名为 classpath，其变量值如下。

```
.;%JAVA_HOME%/lib/rt.jar;%JAVA_HOME%/lib/tools.jar
```

单击“确定”按钮找到 PATH 的变量，双击或单击编辑，在变量值最前面添加如下值。

```
%JAVA_HOME%/bin;
```

具体如图 1-13 所示。

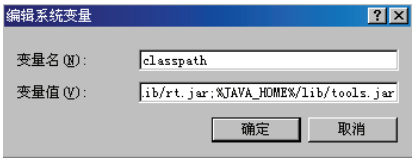


图 1-13　编辑系统变量

（3）再依次单击“开始”→“运行”，在运行框中输入“cmd”命令并按〈Enter〉键，在打开的 CMD 窗口中输入 java –version，此时会显示如图 1-14 所示的安装成功提示。

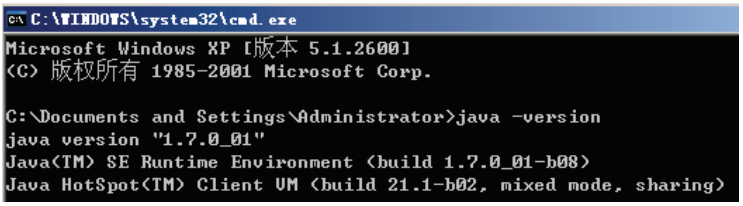


图 1-14　CMD 窗口

注意：上述变量设置中，是按照编者本人的安装路径设置的，编者安装的 JDK 的路径是 C:\Program Files\Java\jdk1.7.0_01。

1.4.3　获取并安装 Eclipse 和 Android SDK

在安装好 JDK 后，接下来需要安装 Eclipse 和 Android SDK。Eclipse 是进行 Android 应用程序开发的一个集成工具，而 Android SDK 是开发 Android 应用程序所必须具备的框架。在 Android 官方公布的最新版本中，已经将 Eclipse 和 Android SDK 这两个工具进行了集成，一次下载即可同时获得这两个工具。获取并安装 Eclipse 和 Android SDK 的具体步骤如下。

（1）登录 Android 的官方网站 http://developer.android.com/index.html，如图 1-15 所示。

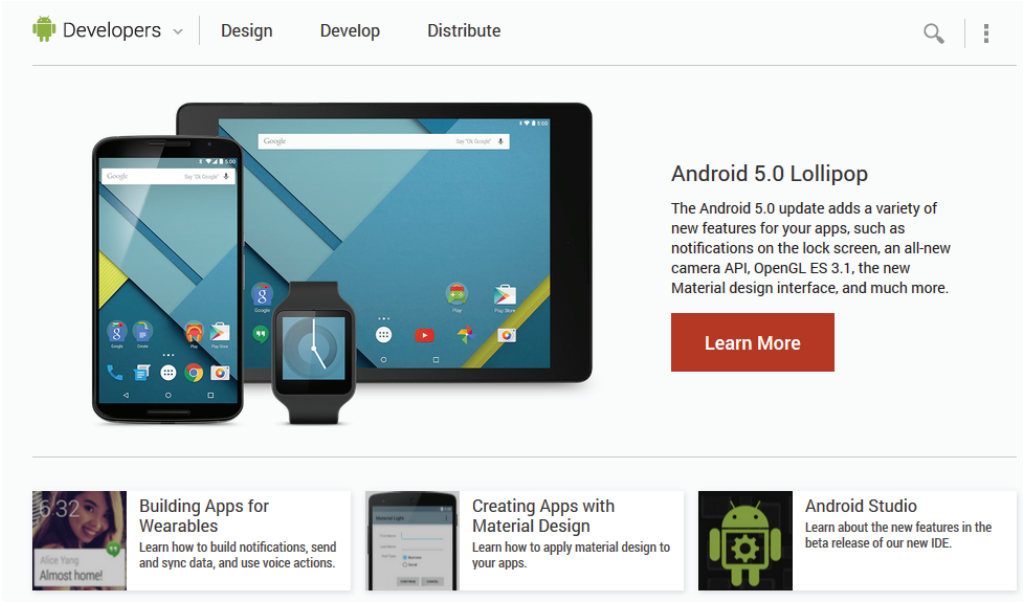


图 1-15　Android 的官方网站

（2）单击网页中的 Get the SDK 链接，如图 1-16 所示。

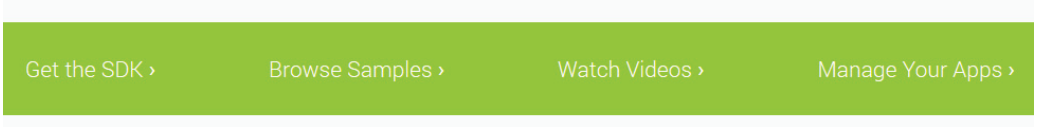


图 1-16　单击 Get the SDK 链接

（3）在弹出的新页面中单击 Download the SDK 按钮，如图 1-17 所示。

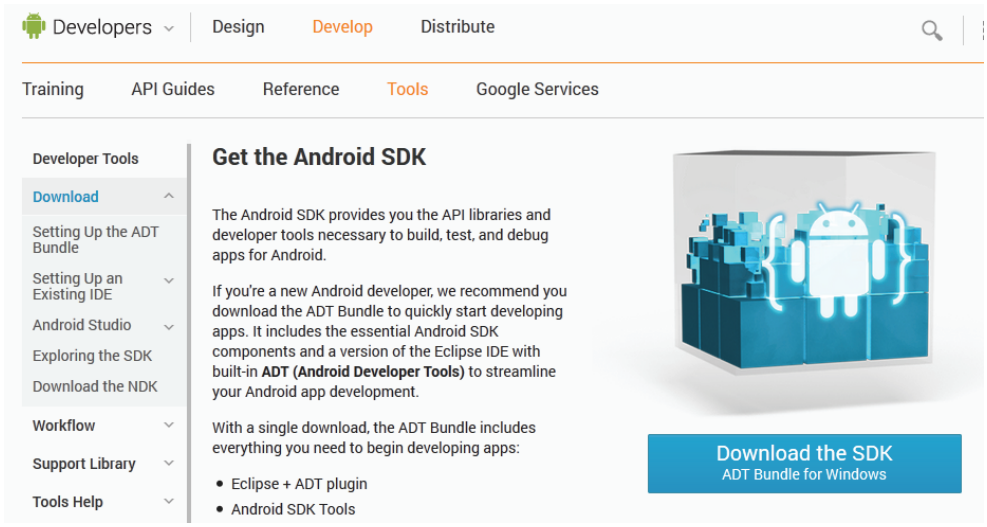


图 1-17　单击 Download the SDK 按钮

（4）在弹出的 Get the Android SDK 界面中勾选 I have read and agree with the above terms and conditions 前面的复选框，然后在下面的单选按钮中选择系统的位数。例如编者的计算机是 32 位的，所以勾选“32-bit”前面的单选按钮，如图 1-18 所示。

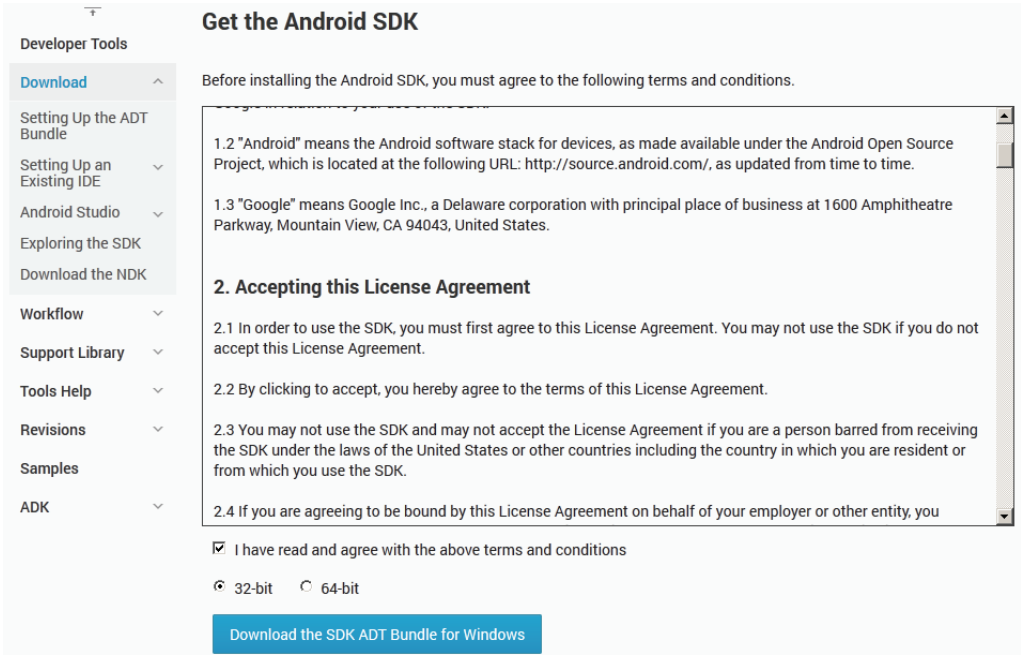


图 1-18　Get the Android SDK 界面

（5）单击图 1-18 中的 Download the SDK ADT Bundle for Windows 按钮后开始下载工作，下载的目标文件是一个压缩包，如图 1-19 所示。

图 1-19　开始下载目标文件压缩包

（6）将下载得到的压缩包进行解压，解压后的目录结构如图 1-20 所示。

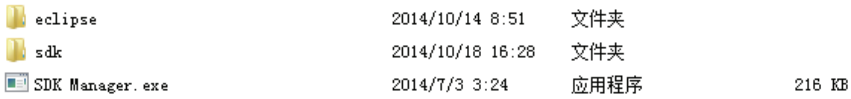


图 1-20　解压后的目录结构

由此可见，Android 官方已经将 Eclipse 和 Android SDK 实现了集成。双击 eclipse 目录中的 eclipse.exe 可以打开 Eclipse，界面效果如图 1-21 所示。

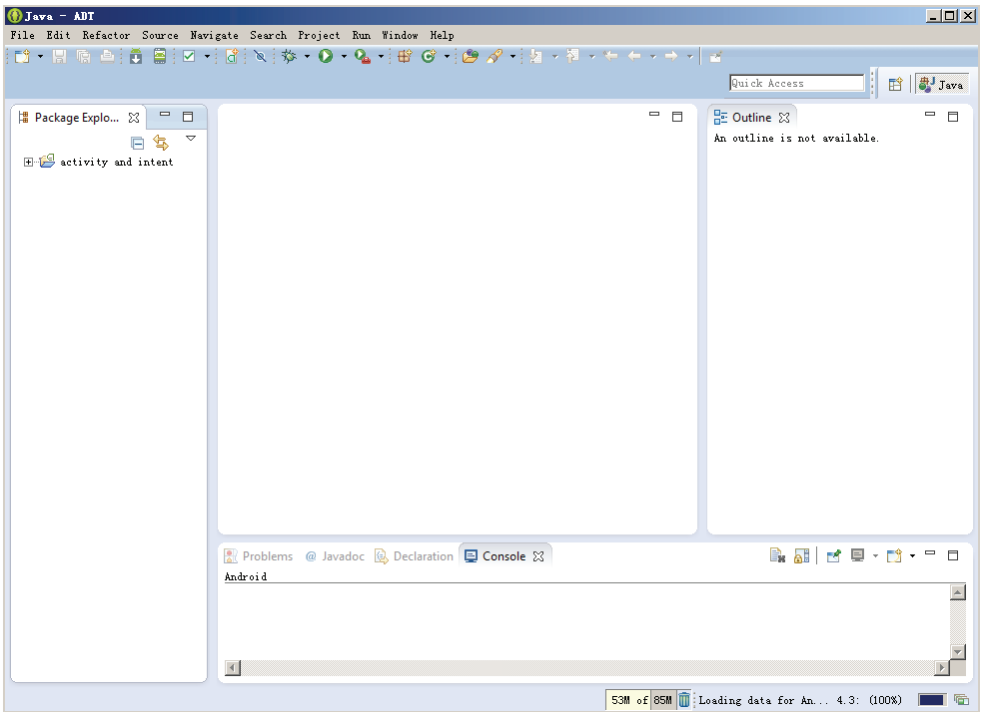


图 1-21　打开 Eclipse 后的界面效果

（7）打开 Android SDK 的方法有两种，第一种是双击下载目录中的“SDK Manager.exe”文件，第二种在是 Eclipse 工具栏中单击图标。打开后的效果如图 1-22 所示，此时会发现当前 Android SDK 的最新版本是 Android L（API 21）。

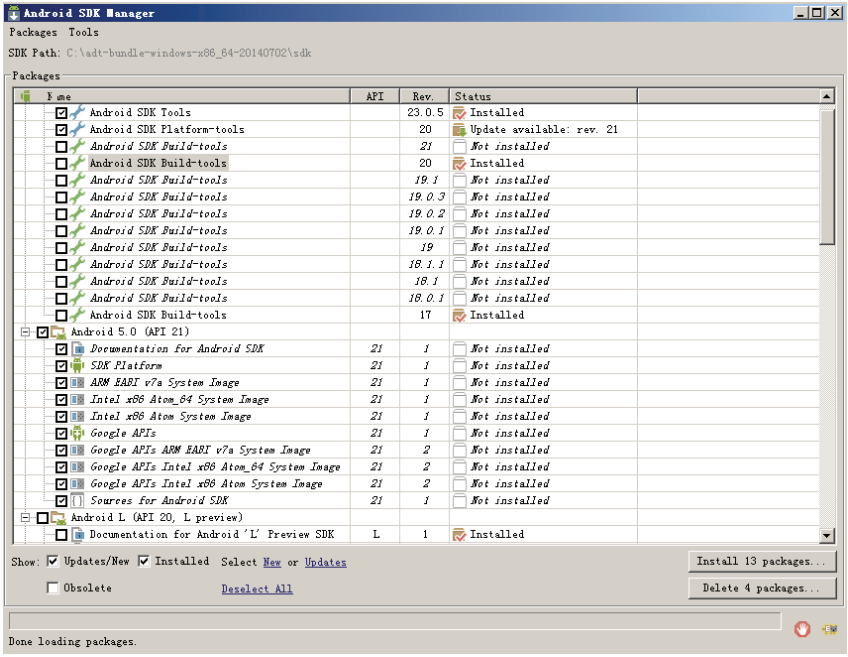


图 1-22　打开 Android SDK 后的界面效果

1.4.4　安装 ADT

Android 为 Eclipse 定制了一个专用插件 ADT（Android Development Tools），此插件为用户提供了一个强大的开发 Android 应用程序的综合环境。ADT 扩展了 Eclipse 的功能，可以让用户快速地建立 Android 项目，创建应用程序界面。要安装 Android Development Tools plug-in，需要首先打开 Eclipse IDE。然后进行如下操作。

（1）打开 Eclipse 后，依次单击菜单栏中的 Help→Install New Software...选项，如图 1-23 所示。

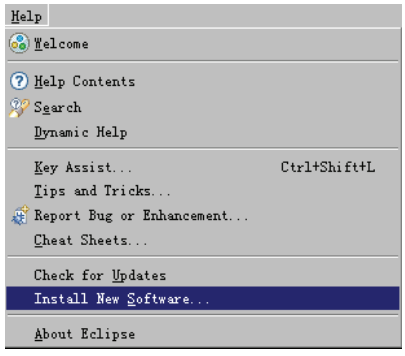


图 1-23　添加插件

（2）在弹出的对话框中单击 Add 按钮，如图 1-24 所示。

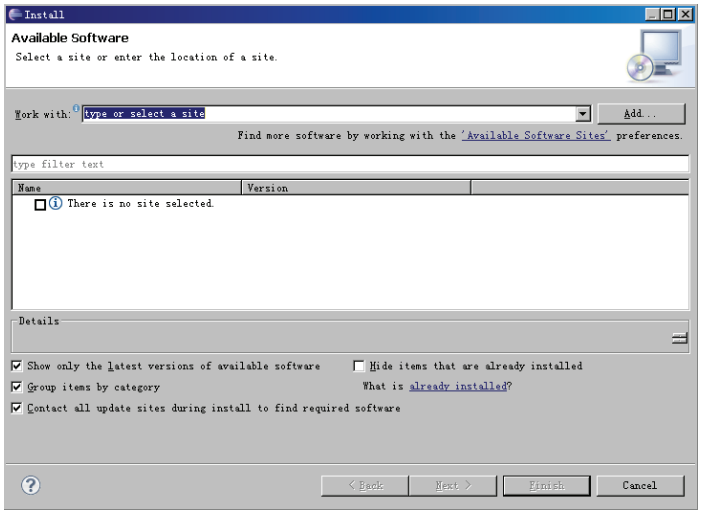


图 1-24　添加插件

（3）在弹出的 Add Site 对话框中分别输入名字和地址，名字可以自己命名，例如“123”，但是在 Location 中必须输入插件的网络地址 http://dl-ssl.google.com/Android/eclipse/，如图 1-25 所示。

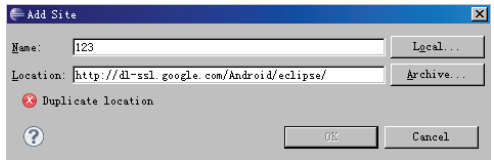


图 1-25　设置地址

（4）单击 OK 按钮，此时在 Install 界面将会显示系统中可用的插件，如图 1-26 所示。

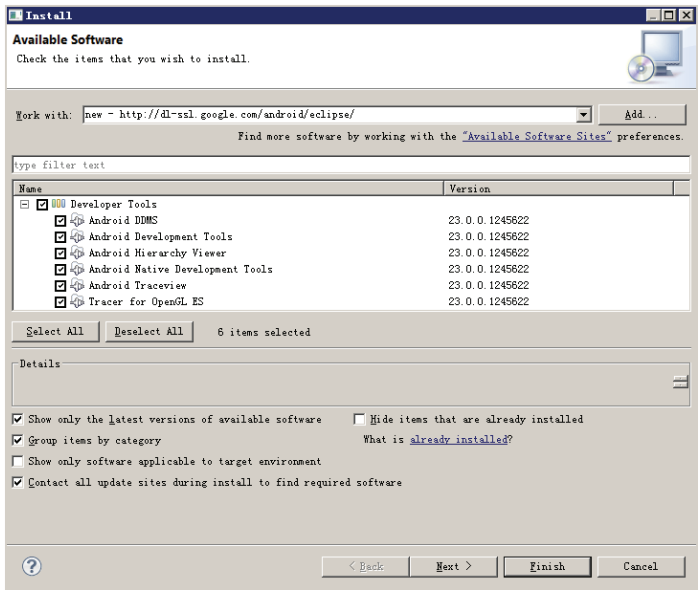


图 1-26　可用插件列表

（5）选中 Android DDMS 和 Android Development Tools，然后单击 Next 按钮打开插件安装详情界面，如图 1-27 所示。

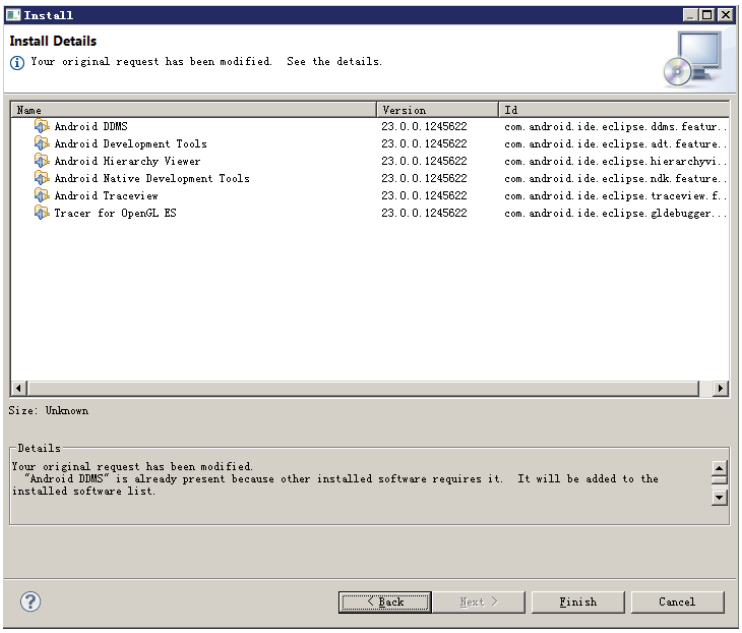


图 1-27　插件安装详情界面

（6）单击 Finish 按钮，开始进行安装，安装进度对话框如图 1-28 所示。

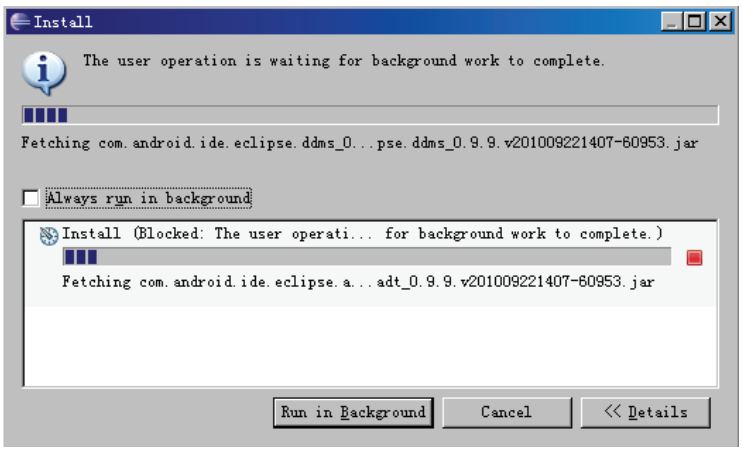


图 1-28　开始安装

注意：在上个步骤中，可能会发生计算插件占用资源的情况，安装过程会有点慢，完成后会提示重启 Eclipse 来加载插件，等重启后就可以使用了。不同版本的 Eclipse 安装插件的方法和步骤是不同的，但是都大同小异，读者可以根据操作提示自行解决。

1.4.5　设定 Android SDK Home

当完成上述插件的安装工作后，此时还不能使用 Eclipse 创建 Android 项目，还需要在

Eclipse 中设置 Android SDK 的主目录。

（1）打开 Eclipse，在菜单中依次单击 Windows→Preferences 项，如图 1-29 所示。

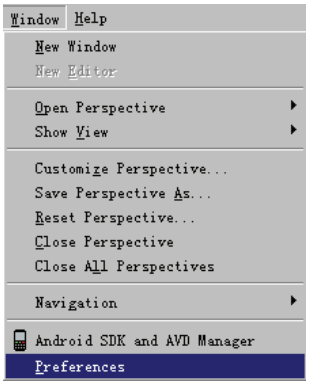


图 1-29　单击 Preferences 项

（2）在弹出的界面左侧可以看到 Android 项，选中 Android 后，在右侧设置 Android SDK 所在目录 SDK Location，单击 OK 按钮完成设置，如图 1-30 所示。

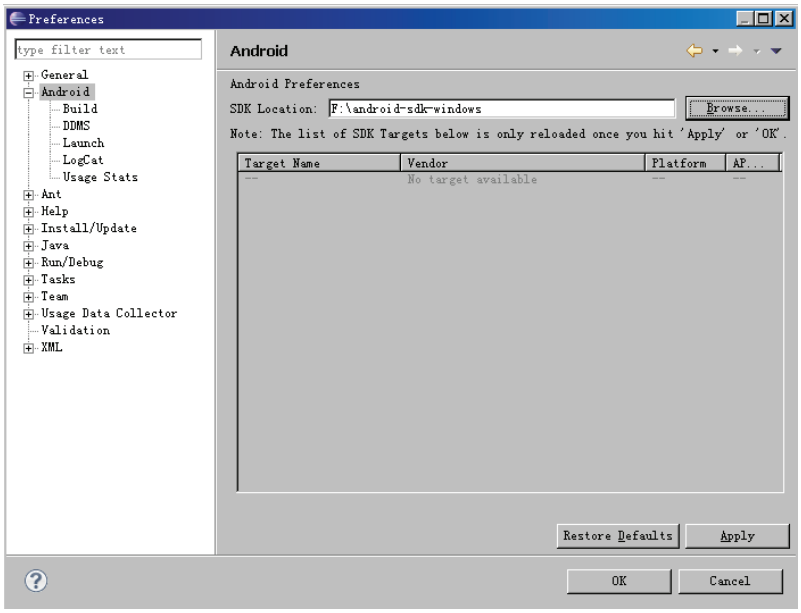


图 1-30　设置 Android SDK 所在目录

1.4.6　验证开发环境

经过前面步骤的讲解，一个基本的 Android 开发环境可以说是搭建完成了。下面通过新建一个项目来验证当前的环境是否可以正常工作。

（1）打开 Eclipse，在菜单中依次选择 File→New→Project 项，在弹出的对话框中可以看到 Android 类型的选项，如图 1-31 所示。

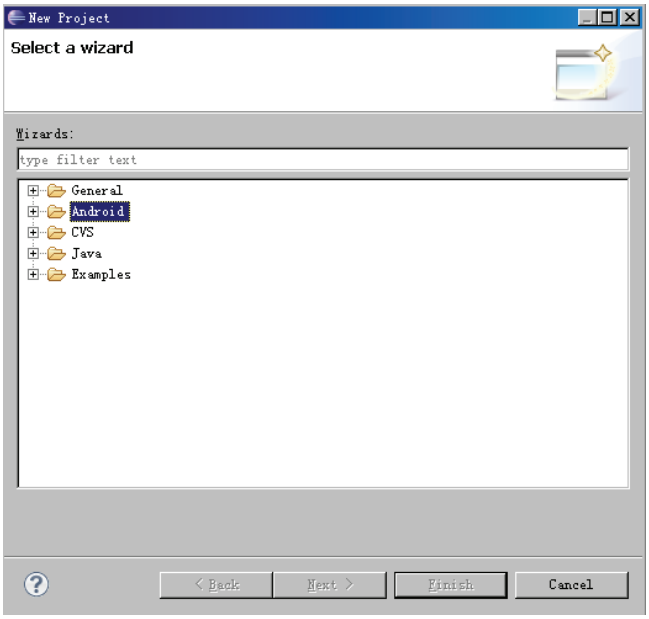


图 1-31　新建项目

（2）在图 1-31 中选择 Android，单击 Next 按钮后打开 New Android Application 对话框，在对应的文本框中输入必要的信息，如图 1-32 所示。

（3）单击 Finish 按钮后，Eclipse 会自动完成项目的创建工作，最后会看到如图 1-33 所示的项目结构。

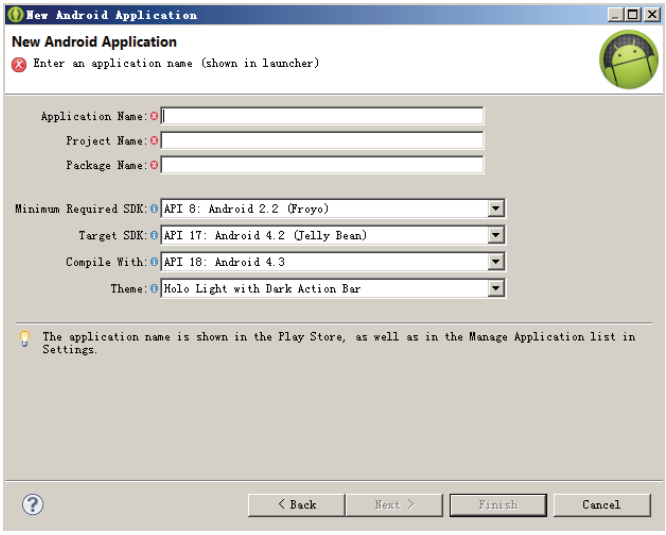


图 1-32　New Android Application 对话框

图 1-33　项目结构

1.4.7　创建 Android 虚拟设备(AVD)

程序开发需要调试，只有经过调试之后才能验证我们的程序是否能正确运行。作为一款

手机操作系统，开发者如何能在计算机平台上调试 Android 应用程序呢？谷歌为开发者提供了模拟器。模拟器是指在计算机上模拟 Android 系统，使用模拟器可以调试并运行开发的 Android 应用程序。由此可见，开发人员不需要使用一个真实的 Android 手机，而是只通过计算机即可模拟运行一个手机环境，进而开发手机应用程序。

谷歌提供的模拟器是 AVD，其全称为 Android 虚拟设备（Android Virtual Device，AVD），每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核、系统图像和数据分区，还可以有自己的 SD 卡和用户数据以及外观显示等。创建 AVD 的基本步骤如下。

（1）单击 Eclips 菜单中的图标，如图 1-34 所示。

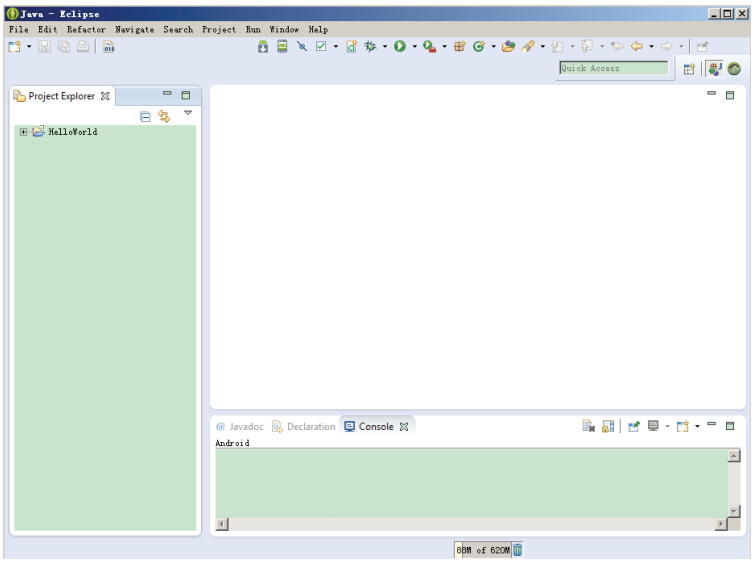


图 1-34　Eclipse 界面

（2）在弹出的 Android Virtual Device（AVD）Manager 界面的左侧导航中选择 Android Virtual Device 选项，如图 1-35 所示。

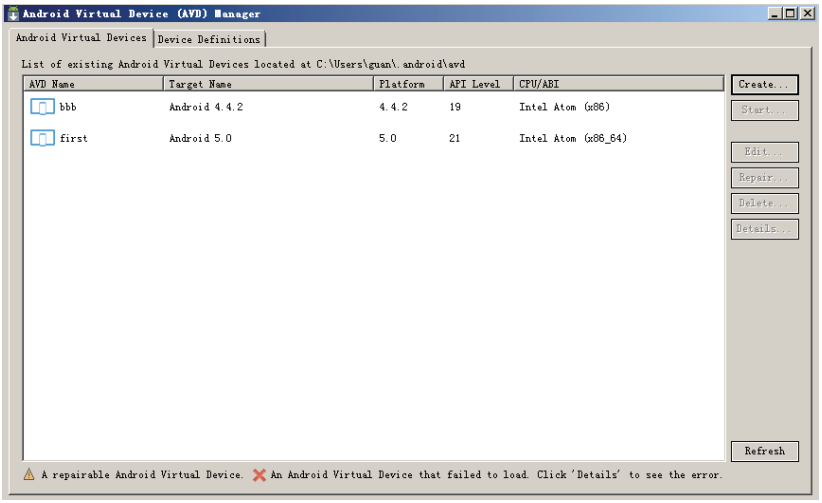


图 1-35　Android Virtual Device（AVD）Manager 界面

在 Virtual Devices 列表中列出了当前已经安装的 AVD 版本，可以通过右侧的按钮来创建、删除或修改 AVD。主要按钮的具体说明如下。

- New...：创建新的 AVD，单击此按钮可在弹出的界面中可以创建一个新 AVD，如图 1-36 所示。
- Edit...：修改已经存在的 AVD。
- Delete...：删除已经存在的 AVD。
- Start...：启动一个 AVD 模拟器。

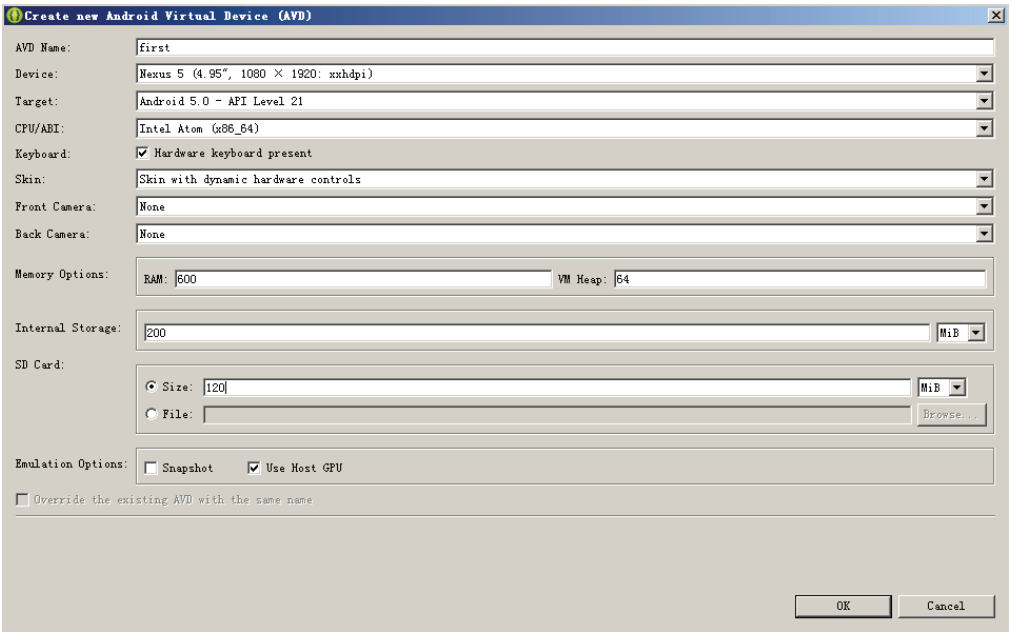


图 1-36　新建 AVD 界面

新建 AVD 界面中，各主要项说明如下。

- AVD Name：在此设置将要创建的 AVD 的名字，可以以英文字符命名。
- Target：在此设置将要创建的 AVD 的 API 版本，例如 Android 2.3、Android 2.3、Android 4.0、Android 5.0 等。
- Device：在此设置将要创建的 AVD 的屏幕分辨率大小。
- CPU/ABI：用于设置当前机器的 CPU。在开发 Android SDK 低版本应用程序时，使用的 Android 模拟器模拟的是 ARM 的体系结构，这个模拟器并不是运行在 X86 上，所以在调试程序的时候运行很慢。针对这个问题， Intel 推出了支持 X86 的 Android 模拟器，这将大大提高软件启动速度和程序的运行速度，将允许 Android 模拟器能够以原始速度（真机运行速度）运行在使用 Intel x86 处理器的计算机中。所以对于使用 Intel x86 计算机开发 Android 应用程序的开发者来说，建议在 CPU/ABI 中选择有 Intel 标识符的选项。

注意：可以在 CMD 中创建或删除 AVD，例如可以使用如下 CMD 命令创建一个 AVD。

```
android create avd --name <your_avd_name> --target <targetID>
```

其中“your_avd_name”是需要创建的 AVD 的名字，在 CMD 窗口界面中如图 1-37 所示。

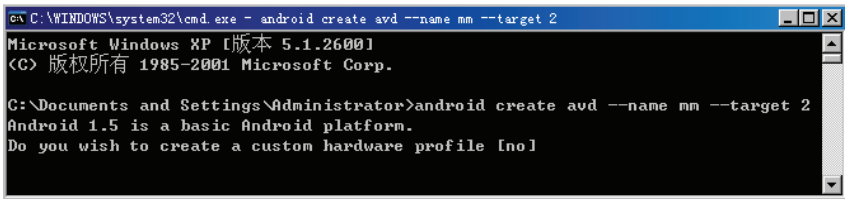


图 1-37　CMD 窗口

1.4.8　启动 AVD 模拟器

在调试 Android 应用程序时需要启动 AVD 模拟器，启动 AVD 模拟器的基本流程如下。

（1）选择图 1-35 列表中名为“win”的 AVD，单击 Start... 按钮后弹出 Launch Option 对话框，如图 1-38 所示。

（2）单击 Launch 按钮后将会运行名为“win”的模拟器，运行界面效果如图 1-39 所示。

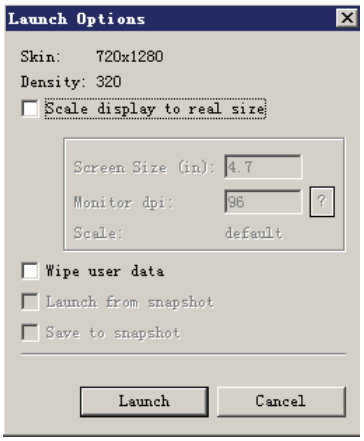


图 1-38　Launch Options 对话框

图 1-39　Android L 模拟器运行界面

注意：技巧——快速安装 SDK 的方法

Android SDK Manager 在线安装的速度非常慢，而且有时容易断开。其实可以先从网络中寻找到 SDK 资源，用迅雷等下载工具下载后，将其放到指定目录后就可以完成安装。具体方法是先下载 android-sdk-windows，然后在 android-sdk-windows 下双击 setup.exe，在更新的过程中会发现安装 Android SDK 的速度是 1Kbit/s，此时打开迅雷，分别输入下面的地址：

https://dl-ssl.google.com/android/repository/platform-tools_r05-windows.zip

```
https://dl-ssl.google.com/android/repository/docs-3.1_r01-linux.zip
https://dl-ssl.google.com/android/repository/android-2.2_r01-windows.zip
https://dl-ssl.google.com/android/repository/android-2.3.3_r01-linux.zip
https://dl-ssl.google.com/android/repository/android-2.1_r01-windows.zip
https://dl-ssl.google.com/android/repository/samples-2.3.3_r01-linux.zip
https://dl-ssl.google.com/android/repository/samples-2.2_r01-linux.zip
https://dl-ssl.google.com/android/repository/samples-2.1_r01-linux.zip
https://dl-ssl.google.com/android/repository/compatibility_r02.zip
https://dl-ssl.google.com/android/repository/tools_r11-windows.zip
https://dl-ssl.google.com/android/repository/google_apis-10_r02.zip
https://dl-ssl.google.com/android/repository/android-2.3.1_r01-linux.zip
https://dl-ssl.google.com/android/repository/usb_driver_r04-windows.zip
https://dl-ssl.google.com/android/repository/googleadmobadssdkandroid-4.1.0.zip
https://dl-ssl.google.com/android/repository/market_licensing-r01.zip
https://dl-ssl.google.com/android/repository/market_billing_r01.zip
https://dl-ssl.google.com/android/repository/google_apis-8_r02.zip
https://dl-ssl.google.com/android/repository/google_apis-7_r01.zip
https://dl-ssl.google.com/android/repository/google_apis-9_r02.zip
……
```

可以继续根据自己开发要求选择不同版本的 API。

下载完后将它们复制到 android-sdk-windows/Temp 目录下，然后再运行 setup.exe，勾选需要的 API 选项，会发现很快就安装好了。记得把原始文件保留好，因为放在 temp 目录下的文件安装好后很快就会被删除。

1.5 第一段 Android 程序

本实例的功能是在手机屏幕中显示问候语“你好我的朋友!”，在具体开始之前先做一个简单的流程规划，如图 1-40 所示。

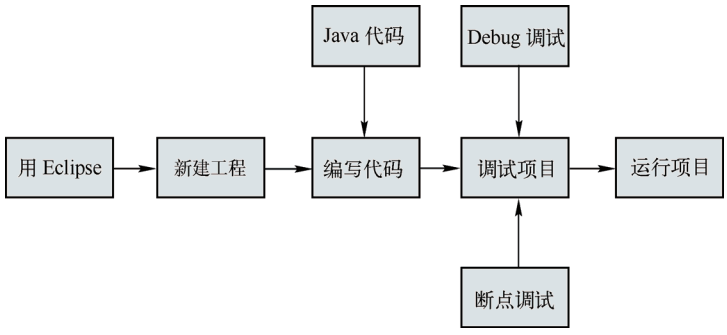


图 1-40　规划流程图

题　　目	目　　的	源 码 路 径
实例 1-1	在手机屏幕中显示问候语	daima\1\first

接下来将详细讲解本实例的具体实现流程。

1.5.1　新建 Android 工程

（1）在 Eclipse 中依次单击 File→New→Project 新建一个工程文件，如图 1-41 所示。

（2）选择 Android Project 项，单击 Next 按钮。

（3）在弹出的 New Android Application 对话框中，设置工程信息，如图 1-42 所示。

在图 1-42 所示的界面中依次设置工程名字、包名字、Activity 名字和应用名字。

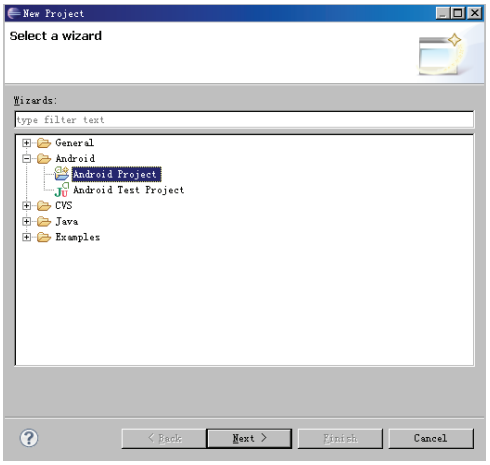


图 1-41　新建工程文件

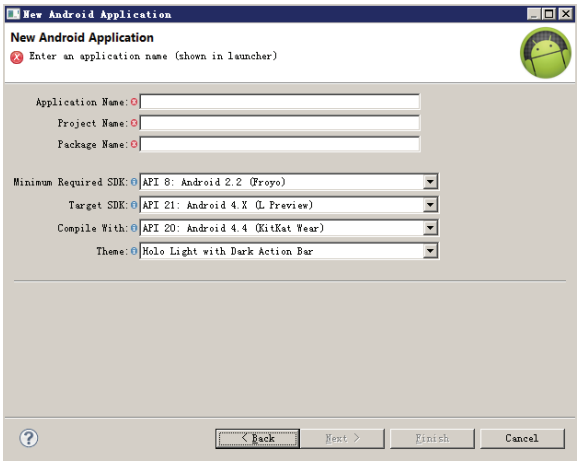


图 1-42　设置工程信息

1.5.2　编写代码

现在已经创建了一个名为“first”的工程文件，现在打开文件 first.java，会显示自动生成的如下代码。

```
package first.a;
import android.app.Activity;
import android.os.Bundle;
public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

如果此时运行程序，将不会显示任何内容。此时可以对上述代码进行稍微的修改，让程序输出“HelloWorld”。具体代码如下。

```
package first.a;
```

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

public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText("你好我的朋友！");
        setContentView(tv);
    }
}
```

经过上述代码的改写后，可以在屏幕中输出“你好我的朋友!”。

1.5.3 调试

Android 调试一般分为 3 个步骤，分别是设置断点、Debug 调试和断点调试。

（1）设置断点

此处的设置断点和 Java 中的方法一样，可以通过双击代码左边的区域进行断点设置，如图 1-43 所示。

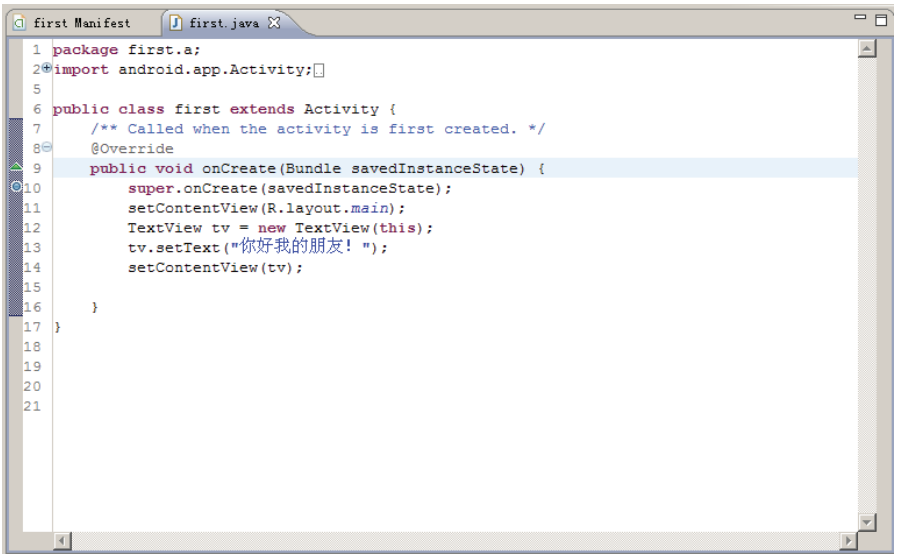


图 1-43　设置断点

为了调试方便，可以设置显示代码的行数。只需在代码左侧的空白部分单击右键，在弹出的菜单中选择 Show Line Numbers 命令，如图 1-44 所示。

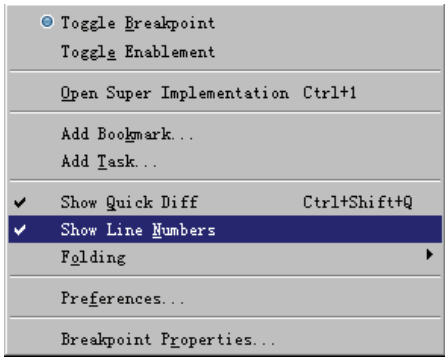


图 1-44　显示行数

（2）Debug 调试

Debug Android 调试项目的方法和普通 Debug Java 调试项目的方法类似，唯一的不同是在选择调试项目时选择 Android Application 命令。具体方法是右键单击项目名，在弹出的菜单中中依次选择 Debug As→Android Application 命令，如图 1-45 所示。

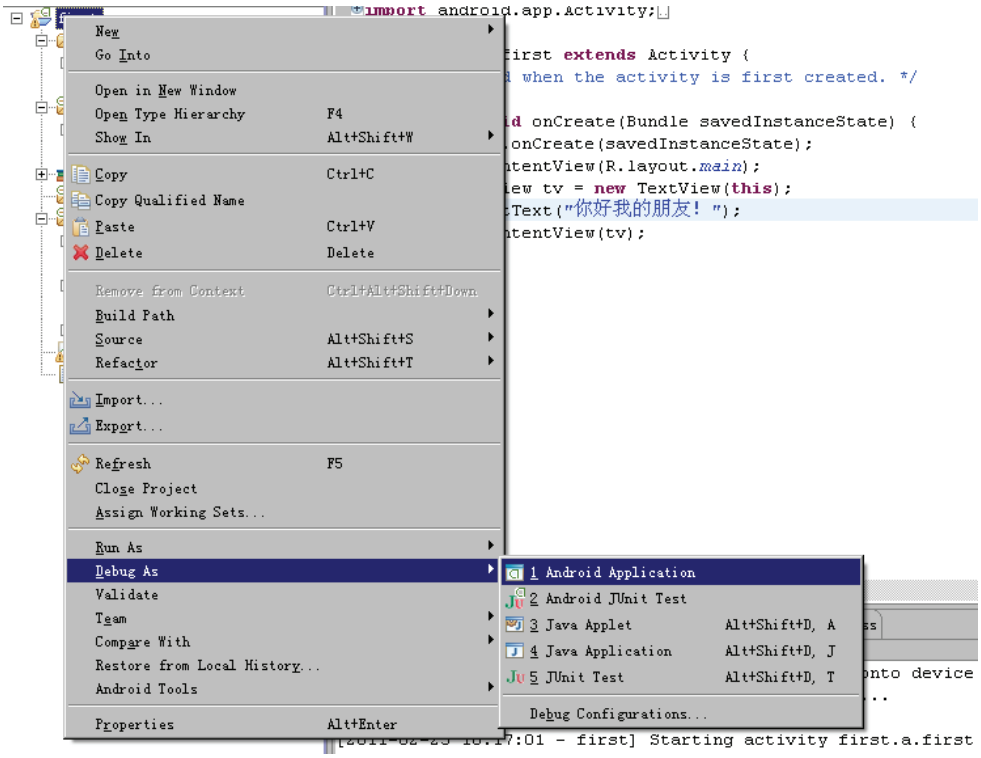


图 1-45　Debug 调试

（3）断点调试

可以进行单步调试，具体调试方法和调试普通 Java 程序的方法类似，调试界面如图 1-46 所示。

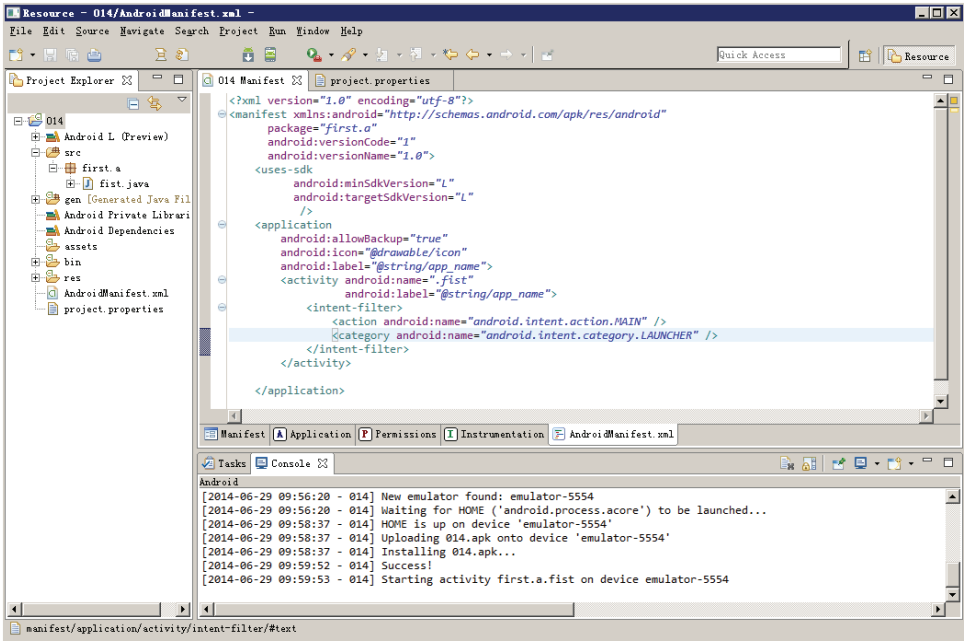


图 1-46　调试界面

1.5.4　运行项目

将上述代码保存后就可运行这段程序了，具体过程如下。

（1）右键单击项目名，在弹出的菜单中依次选择 Run As→Android Application 命令，如图 1-47 所示。

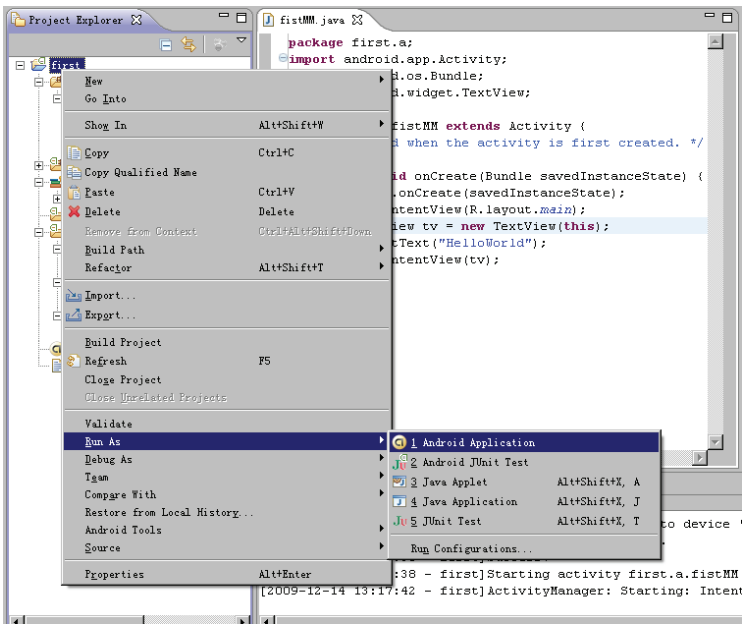


图 1-47　开始调试

（2）此时工程开始运行，运行完成后在屏幕中输出“你好我的朋友!”这段文字，如图 1-48 所示。

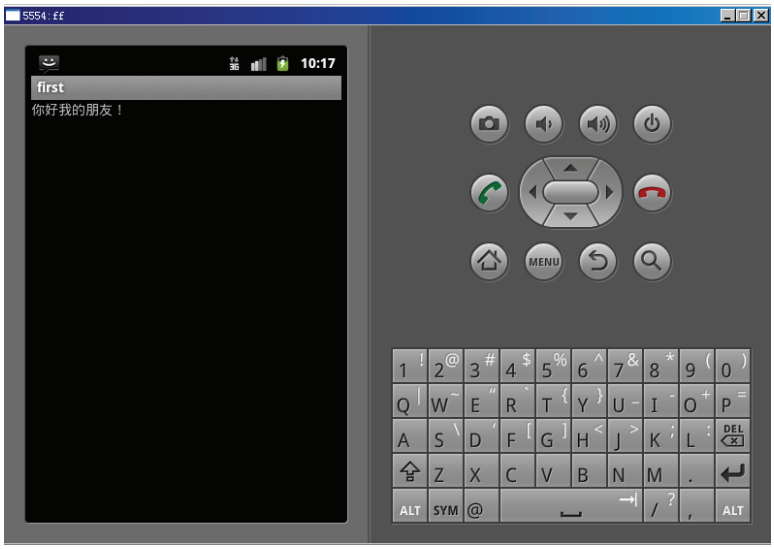


图 1-48　运行结果

第 2 章　Android 网络开发基础

在 Android 开发体系中，网络应用开发是十分重要的一大领域。而网络应用开发领域涉及的知识范围比较广泛，其中主要包括网页开发、远程数据通信、蓝牙通信、WiFi 应用、浏览器开发、流量统计和邮件处理等知识。在学习上述应用开发知识之前，读者需要先具备一些基础性的知识。从本章的内容开始，将简要讲解开发 Android 网络应用项目的基础性知识。

2.1　Android SDK 帮助文档介绍

当读者下载并安装 Android SDK 后，会在安装目录中看到一些安装文件。对开发者来说，其中的帮助文档有着十分重要的参考作用。在 Android SDK 帮助文档中，详细介绍了各个 Android API 接口的语法格式和继承关系，并且介绍了各个接口中的函数和具体用法，这些内容对开发者来说十分重要。

安装 Android SDK 后，其安装目录的结构如图 2-1 所示。

图 2-1　Android SDK 安装后的目录结构

其中主要目录的具体说明如下。

- add-ons：里面包含了官方提供的 API 包，例如常用的 Google Map API（谷歌地图接口）。
- docs：里面包含了帮助文档和说明文档。
- platforms：里面包含了针对每个版本的 SDK 版本，提供了和其对应的 API 包以及一些示例文件，其中包含了各个版本的 Android，如图 2-2 所示。其中“android-21”包含的就是 Android 5.0 的 SDK。

android-10	2014/7/16 21:48	文件夹
android-14	2014/10/13 9:58	文件夹
android-19	2014/10/13 10:11	文件夹
android-20	2014/6/24 8:17	文件夹
android-21	2014/10/18 9:09	文件夹
android-L	2014/10/13 11:20	文件夹

图 2-2　platforms 目录项

- temp：里面包含了一些常用的文件模板。
- tools：包含了一些通用的工具文件。
- usb_driver：包含了 AMD64 和 X86 下的驱动文件。
- SDK Setup.exe：Android 的启动文件。

打开 SDK 帮助文档的方法非常简单，可以使用浏览器打开“docs”目录下的文件 index.html，如图 2-3 所示。然后单击顶部 Developers 中的 Training 链接可以跳转到一个新页面，这个网页就是 SDK 帮助文档的学习主页，界面效果如图 2-4 所示。

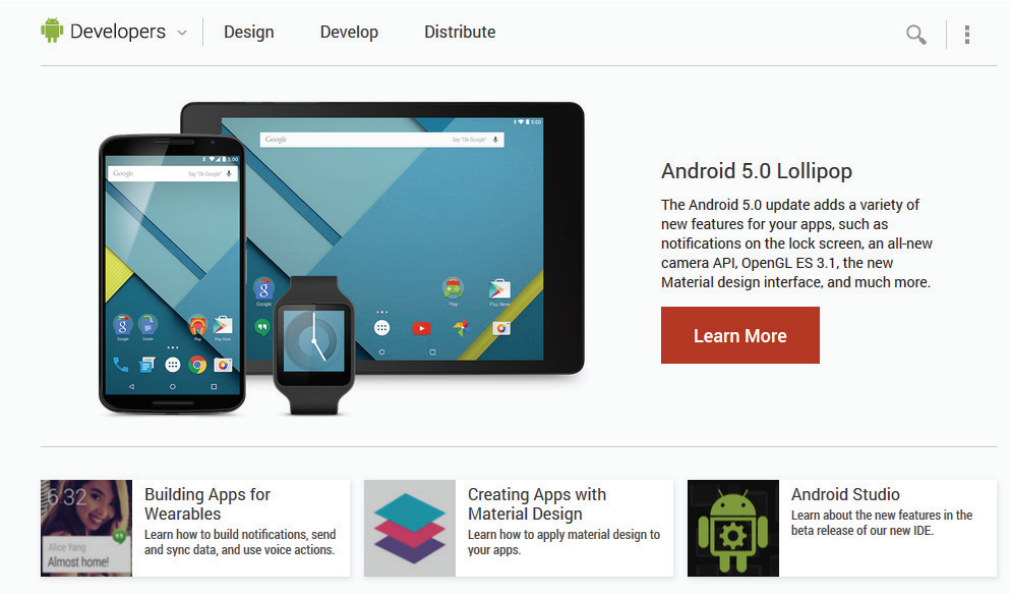


图 2-3　SDK 文档主页

在图 2-4 所示的页面中，介绍了 Android 基本概念和当前常用版本。此 SDK 文件对于初学者来说十分重要，可以帮助读者解决很多常见的问题，是一个很好的学习文档和帮助文档。在图 2-4 所示页面中，左侧是目录索引链接，单击某个链接后可以在右侧界面中显示对应的说明信息。如果要想迅速的理解一个问题或知识点，可以在搜索对话框中通过输入关键字的方式进行快速检索。另外，有很多热心的程序员和学者对帮助文档进行了翻译，读者可以从网络中获取免费的中文版帮助文档。

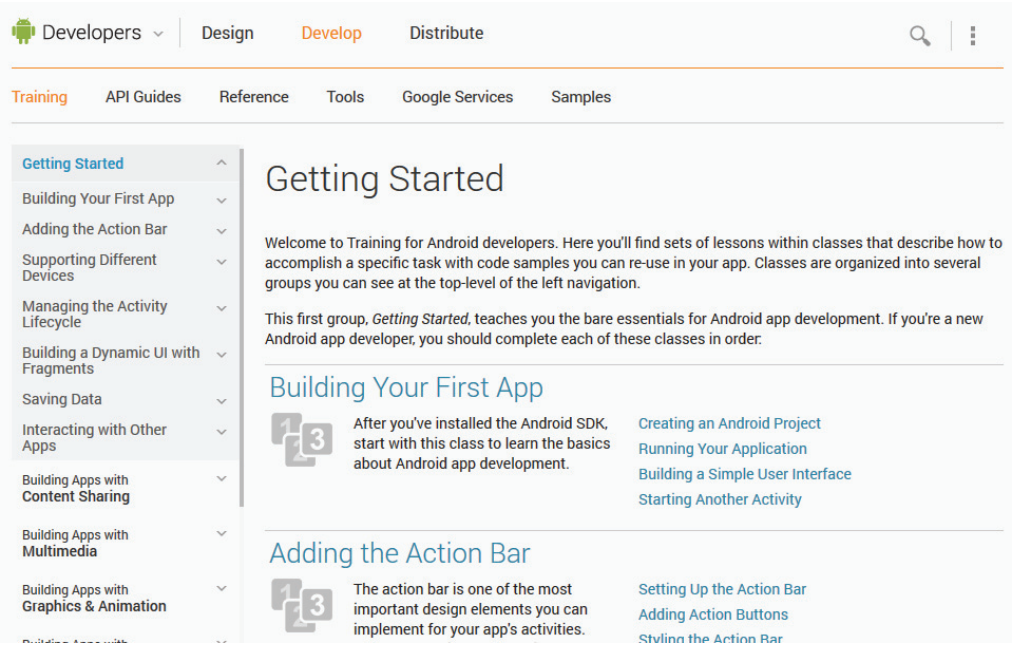


图 2-4　学习界面

2.2　Android 工程文件结构介绍

作为一个基本的 Android 应用程序，其工程文件主要由以下部分组成。

- src 目录：项目源文件都保存在这个目录里面。
- R.java 文件：这个文件是 Eclipse 自动生成的，应用开发者不需要去修改里边的内容。
- Android Library：这个是应用运行的 Android 库。
- assets 目录：里面主要放置多媒体等一些文件。
- res 目录：里面主要放置应用用到的资源文件。
- drawable 目录：主要放置应用用到的图片资源。
- layout 目录：主要放置用到的布局文件。这些布局文件都是 XML 文件。
- values 目录：主要放置字符串（strings.xml）、颜色（colors.xml）、数组（arrays.xml）。
- Androidmanifest.xml：相当于应用的配置文件。在这个文件里边，必须声明应用的名称，应用所用到的 Activity、Service 以及 Receiver 等。

在接下来的内容中，将以本书第 1 章中的实例 1-1 为素材，介绍 Android 工程文件结构的基本知识。

在 Eclipse 中打开实例 1-1 的工程文件，其目录结构如图 2-5 所示。

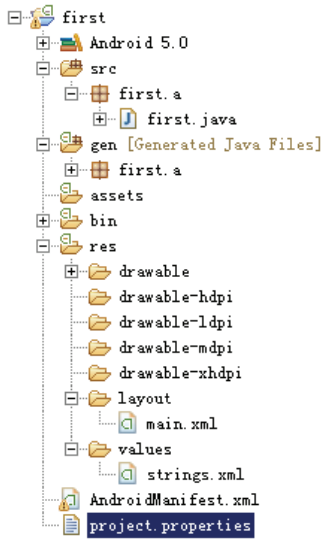


图 2-5　Android 应用工程文件组成

2.2.1　src 目录

与一般的 Java 项目一样，“src”目录下保存的是项目中的所有包及源文件（.java），“res”目录下包含了项目中的所有资源。例如，程序图标（drawable）、布局文件（layout）和常量（values）等。不同的是，在 Java 项目中没有“gen”目录，也没有每个 Android 项目都必须有的 AndroidManfest.xml 文件。

“.java”格式文件是在建立项目时自动生成的，这个文件是只读模式，不能更改。文件 R.java 是定义该项目所有资源的索引文件。先来看看实例 1-1 项目中的文件 R.java，例如下面的代码。

```
/* 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 first.a;
public final class R {
      public static final class attr {
      }
      public static final class drawable {
            public static final int icon=0x7f020000;
      }
      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;
      }
}
```

从上述代码中可以看到定义了很多常量，并且会发现这些常量的名字都与 res 文件夹中的文件名相同，这再次证明了“.java”文件中所存储的就是该项目所有资源的索引。有了这个文件，在程序中使用资源将变得更加方便。由于这个文件不能被手动编辑，所以当我们在项目中加入了新的资源时，只需要刷新一下该项目，“.java”文件会自动生成所有资源的索引。

2.2.2　文件 AndroidManfest.xml

在文件 AndroidManfest.xml 中包含了该项目中所使用的 Activity、Service、Receiver，看下面实例 1-1 项目中的 AndroidManfest.xml 文件。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="first.a"
      android:versionCode="21"
```

```
        android:versionName="1.0">
        <uses-sdk
            android:minSdkVersion="21"
            android:targetSdkVersion="21" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".first"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

在上述代码中，intent-filter 描述了 Activity 启动的位置和时间。每当一个 Activity（或者操作系统）执行一个操作时，将创建一个 Intent 的对象，这个 Intent 对象保存了我们想做什么、想处理什么数据以及数据类型等描述性信息。而 Android 则会和每个 application 所暴露的 intent-filter 数据进行比较，找到最合适的 Activity 来处理调用者所指定的数据和操作。下面我们来仔细分析 AndroidManfest.xml 文件，如表 2-1 所示。

表 2-1　AndroidManfest.xml 分析

参　数	说　明
manifest	根节点，描述了 package 中所有的内容
xmlns:android	包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据
Package	声明应用程序包
application	包含 package 中 application 级别组件声明的根节点。此元素也可包含 application 的一些全局和默认的属性，如标签、icon、主题、必要的权限，等等。一个 manifest 能包含 0 个或 1 个此元素（不能大于 1 个）
android:icon	应用程序图标
android:label	应用程序名字
Activity	用来与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 Activity 所实现，并声明在另外的 Activity 标记中。注意，每一个 Activity 必须有一个<activity>标记对应，无论它给外部使用或是只用于自己的 package 中。如果一个 Activity 没有对应的标记，则不能运行。另外，为了支持运行时查找 Activity，可包含一个或多个<intent-filter>元素来描述 Activity 所支持的操作
android:name	应用程序默认启动的 Activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 Intent Filter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、Icon 和其他信息
action	组件支持的 Intent Action（界面动作）
category	组件支持的 Intent Category。这里指定了应用程序默认启动的 Activity
uses-sdk	该应用程序所使用的 SDK 版本相关信号

2.2.3　定义常量的文件

在实例 1-1 中，文件 String.xml 的代码非常简单，只定义了两个普通的字符串资源。具体

实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, fistMM!</string>
    <string name="app_name">first</string>
</resources>
```

接下来我们来分析 HelloAndroid 项目的布局文件（layout），首先我们打开文件 res\layout\main.xml，其代码如下。

```
<?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="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

在上述代码中，有以下几个布局和参数。

- < LinearLayout></LinearLayout>：线性版面配置，在这个标签中，所有元件都是按由上到下的队列排成的。
- android:orientation：表示这个项目的版面是从上到下垂直地排列其内部的视图。
- android:layout_width：定义当前视图在屏幕上所占的宽度，fill_parent 表示填充整个屏幕。
- android:layout_height：定义当前视图在屏幕上所占的高度，fill_parent 表示填充整个屏幕。
- wrap_content：随着文字栏位的不同而改变这个视图的宽度或高度。

在上述布局代码中，使用 TextView 来配置文本标签 Widget（构件），其中设置的属性 android:layout_width 表示占满整个屏幕的宽度，属性 android:layout_height 表示可以根据文字来改变高度，而属性 android:text 则设置了这个 TextView 要显示的文字内容。在此处引用了 @string 中的 hello 字符串，即文件 String.xml 中的 hello 所代表的字符串资源。

2.3　Android 中的数据存储方式

Android 操作系统提供的是一种公共文件系统，任何应用软件可以使用它来存储和读取文件，该文件也可以被其他的应用软件所读取（会有一些权限控制设定）。在 Android 系统中，所有的应用软件数据（包括文件）为该应用软件所私有。然而，Android 同样也提供了一种标

准方式供应用软件将私有数据开放给其他应用软件所使用。

在 Android 系统中提供了如下 5 种存储数据的方式。

（1）文件存储。

（2）SQLite 数据库方式。

（3）内容提供器（Content Provider）。

（4）SharedPreferences。

（5）网络。

2.3.1 SharedPreferences 存储

SharedPreferences 存储方式是 Android 提供的一种用来存储简单设置信息的机制，经常用于存储常见的欢迎语、登录用户名和密码等信息。SharedPreferences 使用“键-值”对的方式进行存储，这样开发人员可以很方便的实现数据的读取和存入。

通过使用 SharedPreferences 存储方式，可以保存 Android 平台中的 Long 长整形、Int 整形、String 字符串型数据。可以将 SharedPreferences 中的数据分为多种权限，最常用的是设置为全局共享访问。最终会以 XML 方式来保存数据。在处理这些 XML 数据时，Dalvik 会通过自带的底层的本地 XML Parser 进行解析，比如 XMLpull 方式，这种方式会节约内存资源。

在两个 Activity 之间，除了可以通过 Intent 来传递数据外，还可以用 SharedPreferences 共享数据的方式实现数据传递。使用 SharedPreferences 的方法很简单，例如可以先在 A 中设置如下代码。

```
editor sharedata = getSharedPreferences("data", 0).edit();
    sharedata.putString("item","getSharedPreferences");
    sharedata.commit();
```

然后在 B 中编写如下获取设置信息的代码。

```
SharedPreferences sharedata = getSharedPreferences("data", 0);
String data = sharedata.getString("item", null);
Log.v("cola","data="+data);
```

最后可以通过以下 Java 代码将获取的存储数据显示出来。

```
<SPAN class=hilite1>SharedPreferences
</SPAN> sharedata = getSharedPreferences("data", 0);
String data = sharedata.getString("item", null);
Log.v("cola","data="+data);
```

使用 SharedPreferences 的基本方法，基本上和使用 J2SE(java.util.prefs.Preferences)的方法一样，最终目的是用一种简单的、透明的方式保存用户个性化设置的字体、颜色等参数信息。在绝大多数应用程序中，都会提供“设置”或者“首选项”之类的界面，这些设置可以通过 Preferences 来保存。开发者不需要知道信息到底以什么形式保存的，保存在了什么地方。

在接下来的内容中，将通过一个具体实例来讲解 SharedPreferences 存储数据的方法。

实　例	功　能	源 码 路 径
实例 2-1	使用 SharedPreferences 存储数据	daima\2\SharedP

（1）编写文件 SharedPreferencesHelper.java，主要代码如下.

```
public class SharedPreferencesHelper {
        SharedPreferences sp;
        SharedPreferences.Editor editor;
                Context context;
                public SharedPreferencesHelper(Context c,String name){
                context = c;
                sp = context.getSharedPreferences(name, 0);
                editor = sp.edit();
                }
                public void putValue(String key, String value){
                editor = sp.edit();
                editor.putString(key, value);
                editor.commit();
                }
        public String getValue(String key){
                return sp.getString(key, null);
        }
}
```

（2）编写文件 SharedPreferencesUsage.java，主要代码如下。

```
public class SharedPreferencesUsage extends Activity {
        public final static String COLUMN_NAME ="name";
        public final static String COLUMN_MOBILE ="mobile";
        SharedPreferencesHelper sp;

       @Override
       public void onCreate(Bundle savedInstanceState) {
             super.onCreate(savedInstanceState);
             sp = new SharedPreferencesHelper(this, "contacts");
             sp.putValue(COLUMN_NAME, "我爱故乡月");
             sp.putValue(COLUMN_MOBILE, "150xxxxxxxx");

             String name = sp.getValue(COLUMN_NAME);
             String mobile = sp.getValue(COLUMN_MOBILE);

             TextView tv = new TextView(this);
             tv.setText("NAME:"+ name + "\n" + "MOBILE:" + mobile);
                     setContentView(tv);
       }
}
```

通过上述代码，在 SharedPreferences 中存储了“NAME”和“MOBILE”的数据。因为

上述代码中的 pack_name 为：

```
package com.android.SharedPreferences;
```

所以存放数据的路径为：

```
data/data/com.android.SharedPreferences/share_prefs/contacts.xml
```

其中文件 contacts.xml 的内容如下。

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="mobile">150xxxxxxxx</string>
<string name="name">我爱故乡月</string>
</map>
```

执行后的效果如图 2-6 所示。

图 2-6　执行效果

2.3.2　文件存储

虽然使用 SharedPreferences 存储方式的方法非常方便，但是这种方式的缺点也非常明显——只适合存储比较简单的数据。如果想在 Android 系统中存储更多的数据，有多种方法可供我们选择，例如本小节将要讲解的文件存储方式就是一种很好的选择。和传统的在 Java 中实现 I/O 的程序类似，在 Android 中，可以使用方法 openFileInput()和方法 openFileOutput()来读取设备上的文件，例如下面的代码。

```
String FILE_NAME = "tempfile.tmp";
  //初始化
  FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
  FileInputStream fis = openFileInput(FILE_NAME);
```

在上述代码中，方法 openFileInput()和方法 openFileOutput()只能读取该应用目录下的文件，如果读取非其目录下的文件则会抛出异常。如果在调用 FileOutputStream 时指定的文件不存在，Android 会自动创建它。并且在默认情况下，在写入的时候会覆盖原来文件的内容。如果想把新写入的内容附加到原文件内容之后，则可以指定其模式为 Context.MODE_APPEND。在默认情况下，使用方法 openFileOutput()创建的文件只能被其调用的应用程序使用，其他应用程序无法读取这个文件。如果需要在不同的应用中共享数据，可以使用 Content Provider 存储方式实现。

如果应用程序需要使用一些额外的资源文件，例如用于测试音乐播放器是否可以正常工作的 MP3 文件，我们可以将这些测试文件放在应用程序的/res/raw/目录下，例如命名为

mydatafile.mp3。此时就可以在应用程序中使用 getResources()方法获取资源，然后用 openRawResource()方法（不带扩展名的资源文件名）打开这个文件，具体实现代码如下。

```
Resources myResources = getResources();
InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

除了使用方法 openFileInput()和方法 openFileOutput()读写文件外，在 Android 中还可以使用 deleteFile()和 fileList()等方法来操作文件。

2.3.3　SQLite 存储

在 Android 中最为常用的存储方式是 SQLite 存储，这是一个轻量级的嵌入式数据库。SQLite 是 Android 系统自带的一个标准数据库，支持经典的 SQL 语句。SQLite 遵守 ACID 关联式数据库管理系统，是为嵌入式系统所设计的产品，并且目前已经在很多嵌入式产品中使用。SQLite 的突出优点是占用非常低的资源。在嵌入式设备中，可能只需要几百 KB 的内存即可。SQLite 能够支持 Windows、Linux、UNIX 等主流的操作系统，同时能够跟很多程序语言结合使用，例如 C#、PHP 和 Java 等。并且还支持 ODBC 接口，另外和 MySQL、PostgreSQL 这两款开源数据库管理系统相比，SQLite 的处理速度更快。

注意：ACID 是指数据库事务正确执行的四个基本要素的缩写。包含: 原子性(Atomicity)、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。一个支持事务（Transaction）的数据库系统，必需要具有这四种特性，否则在事务过程（Transaction processing）当中无法保证数据的正确性，交易过程极可能达不到交易方的要求。

在接下来的内容中，将通过一个具体实例来讲解使用 SQLite 存储的方法。

实　例	功　能	源 码 路 径
实例 2-2	练习使用 SQLite 来存储数据	daima\2\SQLite

实例文件 UserSQLite.java 的具体实现流程如下。

（1）定义类 DatabaseHelper，此类承于类 SQLiteOpenHelper，具体代码如下。

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
                + " text not null, " + BODY + " text not null " + ");";
        Log.i("haiyang:createDB=", sql);
        db.execSQL(sql);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

在上述代码中，首先分别重写了方法 onCreate()和 onUpgrade()；然后在方法 onCreate()中构造了一条 SQL 语句，并且通过 db.execSQL(sql)执行了这条 SQL 语句。这条 SQL 语句的功能是生成了一张数据库表。

类 SQLiteOpenHelper 是一个辅助类，功能是生成一个数据库，并对数据库的版本进行管理。当在程序当中调用这个类的方法 getWritableDatabase()或者 getReadableDatabase()时，如果当时没有数据，那么 Android 系统就会自动生成一个数据库。

类 SQLiteOpenHelper 是一个抽象类，在 Android 应用项目中通常需要继承这个类。在类 SQLiteOpenHelper 的实现中包含了如下 3 个方法。

- 方法 onCreate（SQLiteDatabase）：在第一次生成数据库时会调用这个方法，一般我们会在这个方法中生成数据库表。
- 方法 onUpgrade（SQLiteDatabase, int, int）：当数据库需要升级的时候，Android 系统会主动的调用这个方法。一般我们在这个方法中删除数据表，并建立新的数据表，当然是否还需要做其他的操作，完全取决于应用的需求。
- 方法 onOpen（SQLiteDatabase）：这是当打开数据库时的回调方法，一般不会用到。

（2）编写按钮处理事件，单击“插入两条记录”按钮后，如果数据成功插入到数据库中的 diary 表中，那么在界面的 title 区域就会显示插入成功提示信息，如图 2-7 所示。

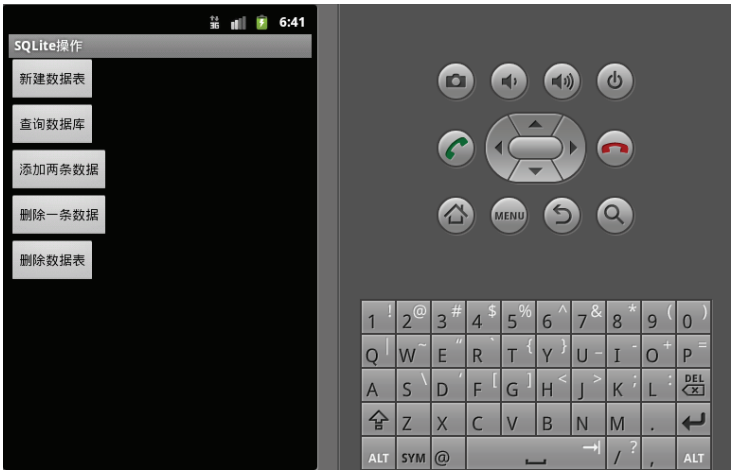


图 2-7　插入成功

如果单击“添加两条数据”按钮，会执行监听器里的 onClick()方法，并最终执行了上述程序里的 insertItem()方法，具体代码如下。

```
/*插入两条数据*/
private void insertItem() {
    /得到一个可写的 SQLite 数据库，如果这个数据库还没有建立/
    /那么 mOpenHelper 辅助类负责建立这个数据库。/
    /如果数据库已经建立，那么直接返回一个可写的数据库。/
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String sql1 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
            + ") values('AA', 'android 好');";
```

```
        String sql2 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
                + ") values('BB', 'android 好');";
        try {
            Log.i("haiyang:sql1=", sql1);
            Log.i("haiyang:sql2=", sql2);
            db.execSQL(sql1);
            db.execSQL(sql2);
            setTitle("插入成功");
        } catch (SQLException e) {
            setTitle("插入失败");
        }
    }
```

在上述代码中，sql1 和 sql2 是构造的两条标准的 SQL 语句，如果读者对 SQL 语句不是很熟悉，可以参考相关的书籍。鉴于本书的重点是 Android，所以对 SQL 语句的知识不进行详细介绍。Log.i()的功能是将参数内容打印到日志中，并且打印级别是 Info 级别；db.execSQL（sql1）表示执行 SQL 语句。

Android 支持 5 种打印输出级别，分别是 Verbose、Debug、Info、Warning、Error，在应用程序中最常用的是 Info 级别，即将一些自己需要知道的信息打印出来，如图 2-8 所示。

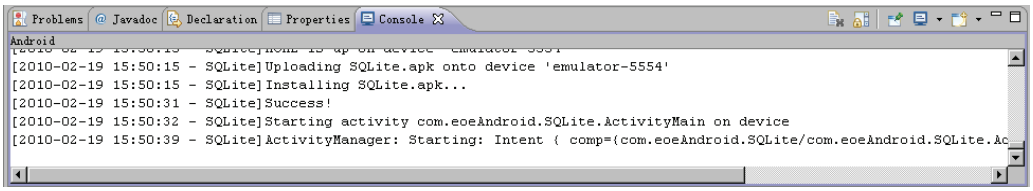


图 2-8　打印输出级别

（3）单击“查询数据库”按钮，在屏幕界面的标题（title）区域会显示当前数据表中数据的条数。因为刚才我们插入了两条数据，所以现在单击“查询数据库”按钮后会显示为两条记录的提示，如图 2-9 所示。

图 2-9　查询数据

单击“查询数据库”按钮后会执行监听器里的 onClick()方法，并最终执行了上述程序里的方法 showItems()，具体代码如下。

```
    /*在屏幕的 title 区域显示当前数据表当中的数据的条数*/
    private void showItems() {
        /得到一个可写的数据库/
        SQLiteDatabase db = mOpenHelper.getReadableDatabase();
        String col[] = { TITLE, BODY };
        Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null);
        /通过 getCount()方法，可以得到 Cursor 当中数据的个数。/
        Integer num = cur.getCount();
        setTitle(Integer.toString(num) + " 条记录");
    }
}
```

在上述代码中，语句“Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null)”的功能是，将查询到的数据放到一个 Cursor 当中。在这个 Cursor 里封装了数据表 TABLE_NAME 中的所有条列。

方法 query()的功能是查询数据，包含了如下 7 个参数。

- 第 1 个参数：是数据库中表的名字，比如在这个例子中，表的名字就是 TABLE_NAME，也就是“diary”。
- 第 2 个参数：是我们想要返回数据包含的列的信息。在这个例子当中我们想要得到的列有 title、body，我们把这两个列的名字放到字符串数组中。
- 第 3 个参数：selection，相当于 SQL 语句的 where 部分，如果想返回所有的数据，那么就直接置为 null。
- 第 4 个参数：selectionArgs，在 selection 部分有可能用到“?”，那么在 selectionArgs 定义的字符串会代替 selection 中的“?”。
- 第 5 个参数：groupBy，定义查询出来的数据是否分组，如果为 null 则说明不用分组。
- 第 6 个参数：having，相当于 SQL 语句当中的 having 部分。
- 第 7 个参数：orderBy，用于描述我们期望的返回值是否需要排序，如果设置为 null 则说明不需要排序。

注意：Cursor 在 Android 当中是一个非常有用的接口，通过 Cursor 我们可以对从数据库查询出来的结果集进行随机的读写访问。

（4）单击“删除一条数据”按钮后，如果成功删除会在屏幕的 title 区域看到文字提示，如图 2-10 所示。

现在再次单击“查询数据库”按钮，会发现数据库中的记录少了一条，如图 2-11 所示。

当单击“删除一条数据”按钮，程序会执行监听器中的 onClick 方法，并最终执行了上述程序里的 deleteItem()方法，其实现代码如下。

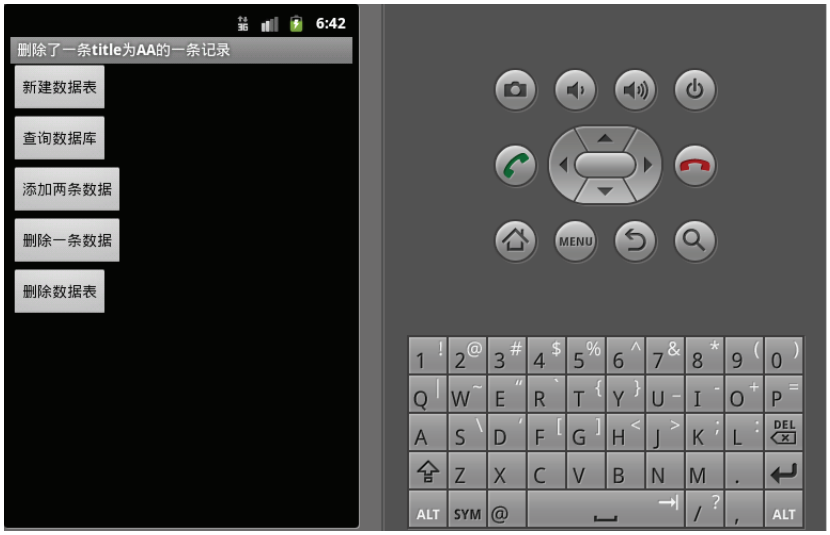


图 2-10　删除一条数据

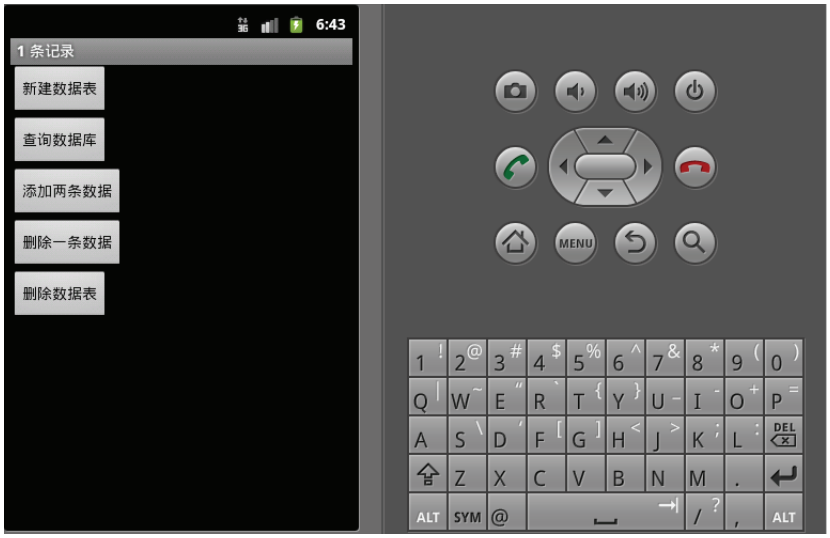


图 2-11　查询数据

```
/*删除其中的一条数据*/
private void deleteItem() {
        try {
                SQLiteDatabase db = mOpenHelper.getWritableDatabase();
                db.delete(TABLE_NAME, " title = 'AA'", null);
                setTitle("删除了一条 title 为 AA 的一条记录");
        } catch (SQLException e) {
        }
}
```

在上述代码中，通过“db.delete(TABLE_NAME, " title = 'haiyang'", null)”语句删除了一条 title 为“AA”的数据。如果有很多条 title 为“AA”的数据，则会全部删除。方法 delete()中各个参数的具体说明如下。

- 第 1 个参数：表示数据库表名，在这里是 TABLE_NAME，也就是 diary。
- 第 2 个参数：相当于 SQL 语句当中的 where 部分，也就是描述了删除的条件。

如果在第 2 个参数当中有“？”，那么第 3 个参数中的字符串会依次替换在第 2 个参数当中出现的“？”。

（5）单击“删除数据表”按钮后可以删除表 diary，如图 2-12 所示。

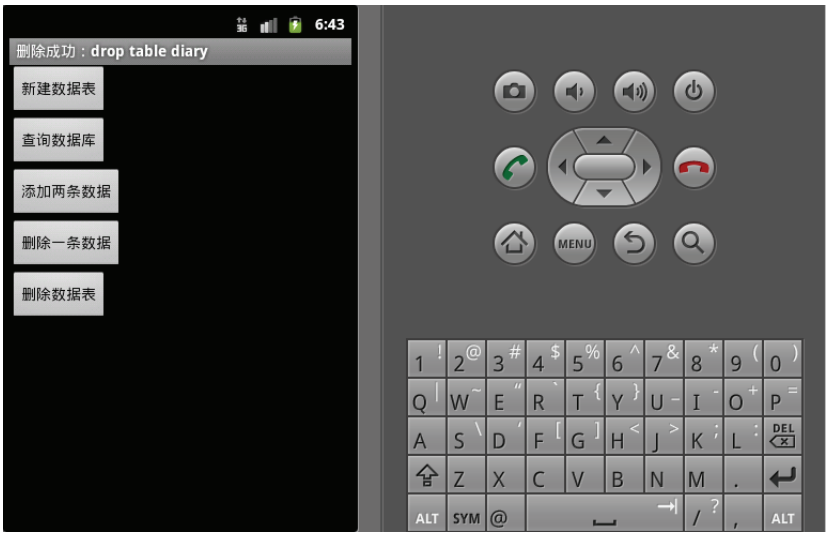


图 2-12　删除表

单击“删除数据表”按钮后会执行方法 dropTable()，具体代码如下。

```
    /*删除数据表*/
    private void dropTable() {
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        String sql = "drop table " + TABLE_NAME;
        try {
            db.execSQL(sql);
            setTitle("删除成功：" + sql);
        } catch (SQLException e) {
            setTitle("删除错误");
        }
    }
```

在上述代码中，构造了一个标准的删除数据表的 SQL 语句，然后执行语句 db.execSQL(sql)。

（6）此时如果单击其他按钮可能会出现运行异常，如果单击“新建数据表”按钮，执行效果如图 2-13 所示。

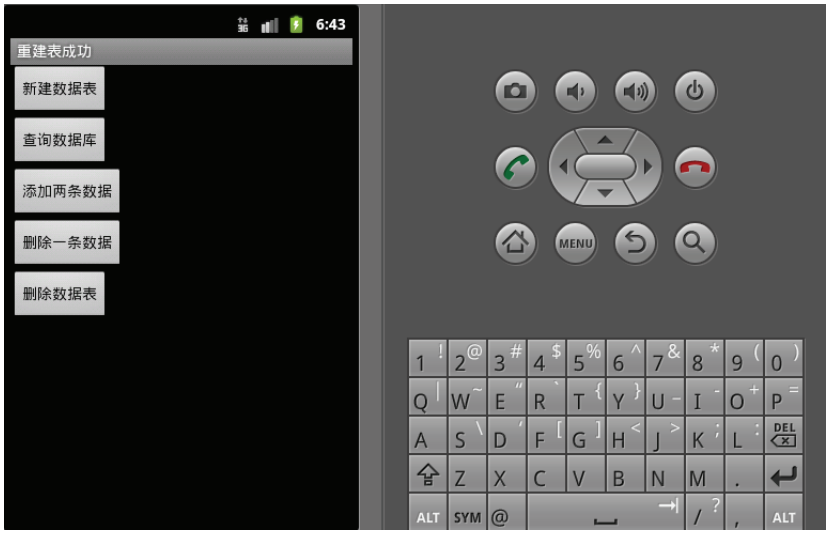


图 2-13　新建表

此时再单击“查询数据库”按钮可以查看里边是否有数据存在，如图 2-14 所示。

图 2-14　显示 0 条记录

单击“新建数据表”按钮后会执行方法 CreateTable()，此方法的功能是重新建立数据表。具体代码如下。

```
/*重新建立数据表*/
private void CreateTable() {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
            + " text not null, " + BODY + " text not null " + ");";
    Log.i("haiyang:createDB=", sql);
```

```
            try {
                db.execSQL("DROP TABLE IF EXISTS diary");
                db.execSQL(sql);
                setTitle("重建数据表成功");
            } catch (SQLException e) {
                setTitle("重建数据表错误");
            }
        }
```

在上述代码中，sql 变量表示使用的是标准的 SQL 语句，功能是按要求建立一张新表。“db.execSQL("DROP TABLE IF EXISTS diary")”表示如果存在表 diary 则先将其删除，因为在同一个数据库中不能出现两张同名字的表；“db.execSQL(sql)”用于执行 SQL 语句，这条 SQL 语句的功能是建立一个新表。

2.3.4 Content Provider 存储

在 Android 系统中的数据是私有的，这些数据包括文件数据、数据库数据和一些其他类型的数据。在 Android 系统中的两个程序之间可以进行数据交换，这个功能是通过 Content Provider 实现的。

1．Content Provider 基础

类 Content Provider 实现了一组标准的方法接口，从而能够让其他的应用保存或读取此 Content Provider 的各种数据类型。在程序中可以通过实现 Content Provider 抽象接口的方式将自己的数据显示出来，而在外界不会看到这个显示数据在应用当中是如何存储的。我们无需关心是用数据库存储还是用文件存储。外界可以通过这套标准的、统一的接口在程序中实现数据交互，即可以读取程序里的数据，也可以删除程序里的数据。

在现实中有如下几种比较常见的 Content Provider 接口。

（1）ContentResolver 接口

外部程序可以通过 ContentResolver 接口访问 Content Provider 提供的数据。在 Activity 当中，可以通过方法 getContentResolver()获取当前应用的 ContentResolver 实例。ContentResolver 提供的接口需要和 Content Provider 中需要实现的接口相对应。接口 ContentResolver 中的常用方法如下。

- query（Uri uri, String[] projection, String selection, String[] selectionArgs,String sortOrder）：通过 Uri 进行查询，返回一个 Cursor。
- insert（Uri url, ContentValues values）：将一组数据插入到 Uri 指定的地方。
- update（Uri uri, ContentValues values, String where, String[] selectionArgs）：更新 Uri 指定位置的数据。
- delete（Uri url, String where, String[] selectionArgs）：删除指定 Uri 并且符合一定条件的数据。

（2）Content Provider 和 ContentResolver 中的 URI

在 Content Provider 和 ContentResolver 中，通常有两种使用 URI 的形式，一种是指定所有的数据，另一种是只指定某个 ID 的数据。例如下面的代码。

```
content://contacts/people/          //此 URI 指定的就是全部的联系人数据
content://contacts/people/1         //此 URI 指定的是 ID 为 1 的联系人的数据
```

在上边用到的 URI 一般由如下 3 部分组成。

- 第 1 部分是“content://”。
- 第 2 部分是要获得数据的一个字符串片段。
- 第 3 部分是 ID（如果没有指定 ID，那么表示返回全部）。

因为 URI 通常比较长，而且有时候容易出错，并且难以理解。所以在 Android 中定义了一些辅助类和常量来代替这些长字符串的使用，例如下边的代码。

```
Contacts.People.CONTENT_URI （联系人的 URI）
```

2．使用 Content Provider

为了使读者掌握 Content Provider 存储的用法，接下来将通过一个具体实例的实现过程，详细讲解在 Android 中使用 Content Provider 存储数据的基本流程。

实　例	功　能	源 码 路 径
实例 2-3	使用 SQLite 来存储数据	daima\2\ContentProviderP

主程序文件 ActivityMain.java 的具体代码如下。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Cursor c = getContentResolver().query(Phones.CONTENT_URI, null, null, null, null);
    startManagingCursor(c);
    ListAdapter adapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_2, c,
                    new String[] { Phones.NAME, Phones.NUMBER },
                    new int[] { android.R.id.text1, android.R.id.text2 });
    setListAdapter(adapter);
}
```

对上述代码的具体说明如下。

（1）方法 getContentResolver()：得到应用的 ContentResolver 实例。

（2）方法 query（Phones.CONTENT_URI, null, null, null, null）：是 ContentResolver 中的方法，用于查询所有联系人，并返回一个 Cursor。此方法中各个参数的具体说明如下：

- 第 1 个参数为 Uri，在此例中的 URI 是联系人的 URI。
- 第 2 个参数是一个字符串的数组，数组里边的每一个字符串都是数据表中某一列的名字，它指定返回数据表中那些列的值。
- 第 3 个参数相当于 SQL 语句的 where 部分，描述哪些值是我们需要的。
- 第 4 个参数是一个字符串数组，里边的值依次代替在第三个参数中出现的“?”。
- 第 5 个参数指定了排序的方式。

（3）startManagingCursor(c)语句：让系统来管理生成的 Cursor。

（4）ListAdapter adapter = new SimpleCursorAdapter(this,Android.R.layout.simple_list_item_2,

c, new String[] { Phones.NAME, Phones.NUMBER }, new int[] { Android.R.id.text1, Android.R.id.text2 })：用于生成一个 SimpleCursorAdapter。

（5）setListAdapter(adapter)：将 ListView 和 SimpleCursorAdapter 进行绑定。

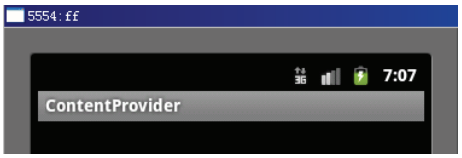


图 2-15　初始效果

运行后的效果如图 2-15 所示。

我们可以在联系人列表中添加几条数据，具体添加流程如下。

（1）单击模拟器的键，在弹出的界面中单击 Contacts 按钮，如图 2-16 所示。

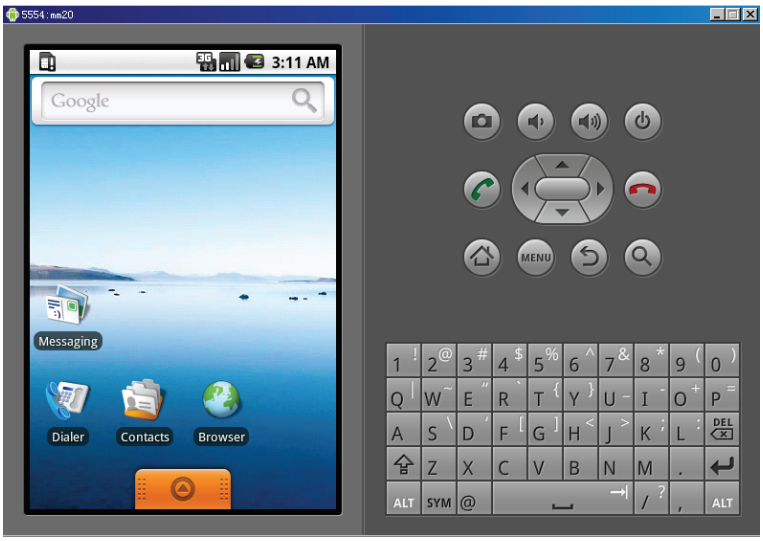


图 2-16　出现的桌面

（2）单击 MENU 项，在弹出界面中单击 Creat new contact 选项，如图 2-17 所示。

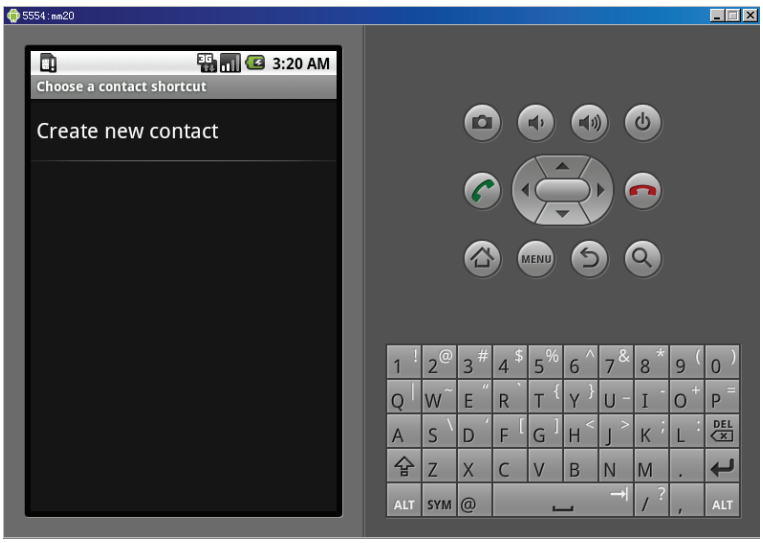


图 2-17　单击 New contact 按钮

（3）添加联系人姓名和电话号码信息，如图 2-18 所示。

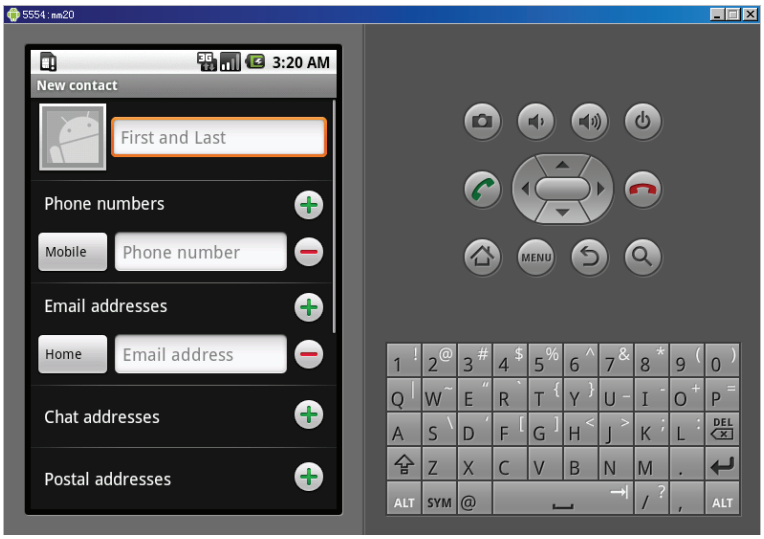


图 2-18　添加联系人姓名和电话号码

（4）单击 Save 按钮添加新建的联系人信息，如图 2-19 所示。

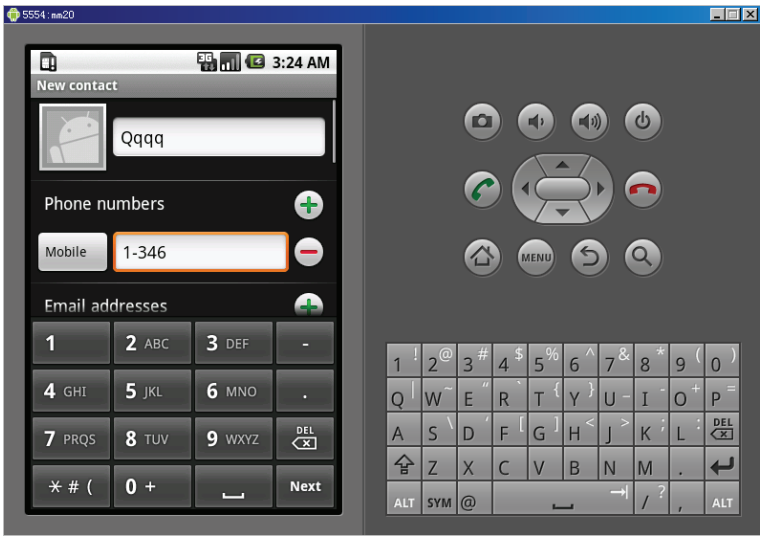


图 2-19　单击 Save 按钮以保存

通过上述操作步骤后，即可添加一条联系人的数据，如图 2-20 所示。

2.3.5　网络存储

在 Android 系统中，可以通过网络实现数据存储工作。在早期版本中，曾经支持用 XMPP Service 和 Web Service 进行远程访问，但是从 Android SDK 1.0 以后不再支持 XMPP Service，而且访问 Web Service 的 API 也全部升级了。

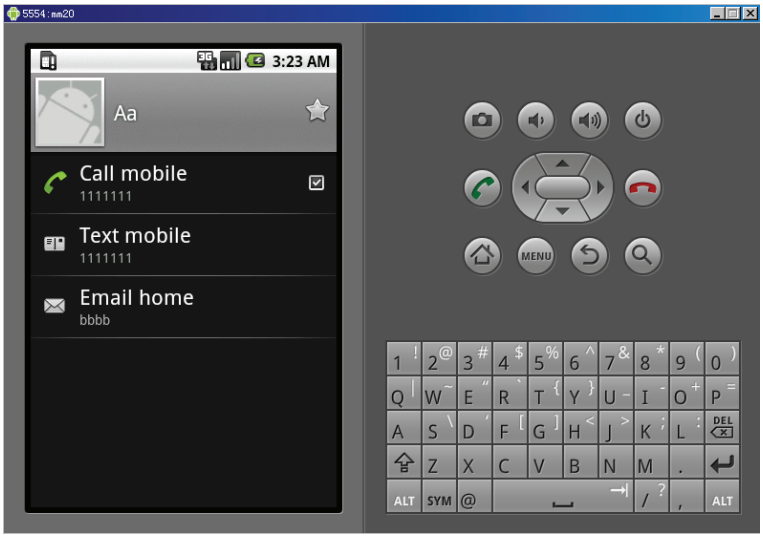


图 2-20　添加后的数据

网络存储在需要及时更新类型的项目中比较常用，例如可以在网络中通过邮政编码来查询该地区的天气预报。实现原理是以 POST 方式发送请求到 webservicex.net 站点，访问 WebService.webservicex.net 站点上提供查询天气预报的服务，具体信息请参考其 WSDL 文档，其网址是：

```
http://www.webservicex.net/WeatherForecast.asmx?WSDL
```

登录后可以查询某地的实时天气状况，输入和输出信息的具体说明如下。

- 输入：某个城市的邮政编码。
- 输出：该邮政编码对应城市的天气预报。

要想实现通过邮政编码来查询该地区的天气预报的功能，可以通过如下过程实现。

（1）如果需要访问外部网络，则需要在文件 AndroidManifest.xml 中加入如下申请权限许可的代码。

```
<!--权限 -->
  <uses-permission Android:name="Android.permission.INTERNET" />
```

（2）以 HTTP POST 的方式发送，SERVER_URL 并不是指 WSDL 的 URL，而是服务本身的 URL。具体实现的代码如下。

```
private static final String SERVER_URL = "http://www.webservicex.net/WeatherForecast.asmx/GetWeatherByZipCode"; //定义需要获取的内容来源地址
    HttpPost request = new HttpPost(SERVER_URL); //根据内容来源地址创建一个 HTTP 请求
    // 添加一个变量
    List <NameValuePair> params = new ArrayList <NameValuePair>();
    // 设置一个华盛顿区号
    params.add(new BasicNameValuePair("ZipCode", "200120"));   //添加必须的参数
    request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8)); //设置参数的编码
```

```
try
{
HttpResponse httpResponse = new DefaultHttpClient().execute(request); //发送请求并获取反馈
   // 解析返回的内容
   if(httpResponse.getStatusLine().getStatusCode() != 404)
   {
      String result = EntityUtils.toString(httpResponse.getEntity());
      Log.d(LOG_TAG, result);
}
} catch (Exception e) {
   Log.e(LOG_TAG, e.getMessage());
}
```

通过上述代码，使用 HTTP 从 webservicex 获取 ZipCode 为“200120”（美国 WASHINGTON D.C）的内容，其返回的内容如下。

```
<WeatherForecasts xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http: //www.w3.org/
2001/XMLSchema-instance" xmlns="http://www.webservicex.net">
   <Latitude>38.97571</Latitude>
   <Longitude>77.02825</Longitude>
   <AllocationFactor>0.024849</AllocationFactor>
   <FipsCode>11</FipsCode>
   <PlaceName>WASHINGTON</PlaceName>
   <StateCode>DC</StateCode>
   <Details>
      <WeatherData>
         <Day>Saturday, April 25, 2009</Day>
        <WeatherImage>http://forecast.weather.gov/images/wtf/sct.jpg</WeatherImage>
         <MaxTemperatureF>88</MaxTemperatureF>
         <MinTemperatureF>57</MinTemperatureF>
         <MaxTemperatureC>31</MaxTemperatureC>
         <MinTemperatureC>14</MinTemperatureC>
      </WeatherData>
      <WeatherData>
         <Day>Sunday, April 26, 2009</Day>
        <WeatherImage>http://forecast.weather.gov/images/wtf/few.jpg</WeatherImage>
         <MaxTemperatureF>89</MaxTemperatureF>
         <MinTemperatureF>60</MinTemperatureF>
         <MaxTemperatureC>32</MaxTemperatureC>
         <MinTemperatureC>16</MinTemperatureC>
      </WeatherData>
…
   </Details>
</WeatherForecasts>
```

通过上述实现过程，演示了如何在 Android 中通过网络获取数据。要掌握这方面的内容，开发者需要熟悉 java.net.*，Android.net.*这两个包的内容，具体信息请读者参阅相关文档。

2.4 访问操作 SD 卡（手机中的存储卡）

在 Android 平台中，可以在如下两个位置对文件进行读写操作。

- SD 卡。
- 手机的存储文件夹。

使用 I/O 技术可以对上述位置存储的文件进行操作。但是基于 SD 卡的特殊性，我们需要事先实现程序对 SD 卡的访问，才能操作 SD 卡中的文件。SD 卡是当前智能手机的一部分，我们经常在 SD 卡中存储大量的文件，例如音乐、视频和游戏。因为 SD 卡的重要性，所以不可避免的需要涉及操作 SD 卡中文件的知识。

其实访问 SD 卡中数据的方法与在 Java 中进行文件读取操作的方法十分类似，只需要注意正确地设置文件的位置和文件名即可。

在 Android 模拟器中，可以使用 FAT32 格式的磁盘镜像作为 SD 卡的模拟，具体过程如下。

（1）进入 Android SDK 目录下的 tools 子目录，然后运行如下命令。

```
mksdcard -l sdcard 512M /your_path_for_img/sdcard.img
```

通过上述命令创建了一个 512MB 大小的 SD 卡镜像文件。

（2）通过如下命令运行模拟器的时候指定路径，在此需要使用完整路径。

```
emulator -sdcard /your_path_for_img/sdcard.img
```

这样在模拟器中就可以使用/sdcard 这个路径来指向模拟的 SD 卡了。

接下来需要复制本机文件到 SD 卡中，甚至需要管理 SD 卡中的文件内容。通过如下两种方案可以实现上述功能。

（1）在 Linux 系统下可以挂载成一个 loop 设备，例如先创建一个名为“android_sdcard”的目录，然后执行下面的命令。

```
mount -o loop sdcard.img android_sdcard
```

这样可以通过管理这个目录的方式管理 sdcard 内容。

（2）在 Windows 可视环境下也可以用 mtools 来实现管理，并且也可以用 Android SDK 自带的如下命令（这个命令在 Linux 下面也可以用）实现。

```
adb push local_file sdcard/remote_file
```

在执行完上面的命令后，需要执行下面的命令启动 Android 模拟器。

```
emulator -avd avd1 -sdcard card/mycard.img
```

如果在 Eclipse 开发环境中，可以在 Run Configuration 对话框中设置启动参数。当然，也可以在 Preferences 对话框中设置默认启动参数。这样在新建立的 Android 工程中就自动加入了装载 SD 卡虚拟文件的命令行参数。

在接下来的内容中，将通过一个具体实例的实现过程讲解读取 SD 卡中数据的方法。

实　例	功　能	源 码 路 径
实例 2-4	操作内存和 SD 卡中的文件	daima\2\SDP

2.4.1　解决思路

移动手机的存储控件分为内存控件和存储卡控件。在本实例的屏幕中添加了两个按钮，分别用于添加和删除内存或存储卡内的文件。并且准备使用 3 个 Activity，其中主程序是 Entry Activity，另外两个分别用于处理内存卡和存储卡。当用户选择内存或存储卡后，将以列表形式显示里面的所有的目录和文件名，并在 Menu 菜单中显示“添加”或“删除”按钮。单击“添加”按钮后会显示一个添加菜单，实现添加文件功能。当单击“删除”按钮后可以删除指定的文件。

2.4.2　具体实现

本实例的实现文件是 SDC.java、SDC_1.java 和 SDC_2.java，接下来将分别介绍上述文件的具体实现流程。

（1）编写文件 SDC.java，具体实现流程如下。

- 用方法 getFilesDir()获取 SD 卡的目录，设置当 SD 卡无插入时 myButton2 处于不能按的状态。对应代码如下。

```
/* 取得目前 File 目录 */
fileDir = this.getFilesDir();
/* 取得 SD 卡目录 */
sdcardDir = Environment.getExternalStorageDirectory();
/* 当 SD 卡无插入时将 myButton2 设成不能按 */
if (Environment.getExternalStorageState().equals(Environment.MEDIA_REMOVED))
{
  myButton2.setEnabled(false);
}
```

- 分别定义按钮单击处理事件 setOnClickListener()和 setOnClickListener()，具体代码如下。

```
myButton1.setOnClickListener(new Button.OnClickListener()
{
  @Override
  public void onClick(View arg0)
  {
    String path = fileDir.getParent() + java.io.File.separator
        + fileDir.getName();
```

```
            showListActivity(path);
        }
    });
    myButton2.setOnClickListener(new Button.OnClickListener()
    {
        @Override
        public void onClick(View arg0)
        {
            String path = sdcardDir.getParent() + sdcardDir.getName();
            showListActivity(path);
        }
    });
}
```

- 定义方法 showListActivity(String path)，并定义一个 Intent 对象 intent，然后将路径传到 SDC_1。具体代码如下。

```
    private void showListActivity(String path)
    {
        Intent intent = new Intent();
        intent.setClass(SDC.this, SDC_1.class);
        Bundle bundle = new Bundle();
        /* 将路径传到 example_1 */
        bundle.putString("path", path);
        intent.putExtras(bundle);
        startActivity(intent);
    }
}
```

（2）编写文件 SDC_1.java，具体实现流程如下。

- 将主 Activity 传来的 path（路径）字符串作为传入路径，如果不存在这个路径，则使用 java.io.File 创建一个新的。具体代码如下。

```
public class SDC_1 extends ListActivity
{
    private List<String> items = null;
    private String path;
    protected final static int MENU_NEW = Menu.FIRST;
    protected final static int MENU_DELETE = Menu.FIRST + 1;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.ex06_09_1);
        Bundle bunde = this.getIntent().getExtras();
        path = bunde.getString("path");
        java.io.File file = new java.io.File(path);
```

```
        /* 当该目录不存在时将创建目录 */
        if (!file.exists())
        {
          file.mkdir();
        }
        fill(file.listFiles());
    }
```

❑ 通过 onOptionsItemSelected 根据单击的 MENU 选项实现添加或删除操作，具体代码如下。

```
public boolean onOptionsItemSelected(MenuItem item)
{
  super.onOptionsItemSelected(item);
  switch (item.getItemId())
  {
    case MENU_NEW:
      /* 点击添加 MENU */
      showListActivity(path, "", "");
      break;
    case MENU_DELETE:
      /* 点击删除 MENU */
      deleteFile();
      break;
  }
  return true;
}
```

❑ 使用 onCreateOptionsMenu(Menu menu)添加需要的 MENU，具体代码如下。

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  super.onCreateOptionsMenu(menu);
  /* 添加 MENU */
  menu.add(Menu.NONE, MENU_NEW, 0, R.string.strNewMenu);
  menu.add(Menu.NONE, MENU_DELETE, 0, R.string.strDeleteMenu);
  return true;
}
```

❑ 当单击文件名后获取文件内容，具体代码如下。

```
protected void onListItemClick
(ListView l, View v, int position, long id)
{
  File file = new File
  (path + java.io.File.separator + items.get(position));
```

```
    /* 点击文件取得文件内容 */
    if (file.isFile())
    {
      String data = "";
      try
      {
        FileInputStream stream = new FileInputStream(file);
        StringBuffer sb = new StringBuffer();
        int c;
        while ((c = stream.read()) != -1)
        {
          sb.append((char) c);
        }
        stream.close();
        data = sb.toString();
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
      showListActivity(path, file.getName(), data);
    }
  }
```

❑ 使用方法 fill(File[] files)将内容填充到文件，具体代码如下。

```
  private void fill(File[] files)
  {
    items = new ArrayList<String>();
    if (files == null)
    {
      return;
    }
    for (File file : files)
    {
      items.add(file.getName());
    }
    ArrayAdapter<String> fileList = new ArrayAdapter<String>
    (this,android.R.layout.simple_list_item_1, items);
    setListAdapter(fileList);
  }
```

❑ 使用 showListActivity 来显示已经存在的文件列表，具体代码如下。

```
  private void showListActivity
  (String path, String    ilename, String data)
```

```
{
    Intent intent = new Intent();
    intent.setClass(SDC_1.this, SDC_2.class);
    Bundle bundle = new Bundle();
    /* 文件路径 */
    bundle.putString("path", path);
    /* 文件名 */
    bundle.putString("  ilename",   ilename);
    /* 文件内容 */
    bundle.putString("data", data);
    intent.putExtras(bundle);
    startActivity(intent);
}
```

❑ 使用方法 deleteFile()删除选定的文件，具体代码如下。

```
    private void deleteFile()
    {
        int position = this.getSelectedItemPosition();
        if (position >= 0)
        {
            File file = new File(path + java.io.File.separator +
            items.get(position));
            /* 删除文件 */
            file.delete();
            items.remove(position);
            getListView().invalidateViews();
        }
    }
}
```

（3）编写文件 SDC_2.java，具体实现流程如下。

❑ 设置 myEditText1 来放置文件内容，然后定义 Bundle 对象 bunde 来获取路径 path 和数据 data。具体代码如下。

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.ex06_09_2);
    /* 放置文件内容的 EditText */
    myEditText1 = (EditText) findViewById(R.id.myEditText1);
    Bundle bunde = this.getIntent().getExtras();
    path = bunde.getString("path");
    data = bunde.getString("data");
    fileName = bunde.getString("fileName");
    myEditText1.setText(data);
}
```

- 使用 onOptionsItemSelected()根据用户选择而进行操作，当选择 MENU_SAVE 时会保存这个文件。具体代码如下。

```
public boolean onOptionsItemSelected(MenuItem item)
{
   super.onOptionsItemSelected(item);
   switch (item.getItemId())
   {
      case MENU_SAVE:
         saveFile();
         break;
   }
   return true;
}
```

- 使用 onCreateOptionsMenu(Menu menu)添加一个 MENU，具体代码如下。

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   super.onCreateOptionsMenu(menu);
   /* 添加 MENU */
   menu.add(Menu.NONE, MENU_SAVE, 0, R.string.strSaveMenu);
   return true;
}
```

- 使用方法 saveFile()保存一个文件。定义 LayoutInflater 对象 factory 用于跳出存档，然后通过 myDialogEditText()获取 Dialog 中的 EditText，最后实现存档处理。具体代码如下。

```
   private void saveFile()
   {
      /* 跳出存档的 Dialog */
      LayoutInflater factory = LayoutInflater.from(this);
      final View textEntryView = factory.inflate
      (R.layout.save_dialog, null);
      Builder mBuilder1 = new AlertDialog.Builder(SDC_2.this);
      mBuilder1.setView(textEntryView);
      /* 取得 Dialog 里的 EditText */
      myDialogEditText = (EditText) textEntryView.findViewById
                                          (R.id.myDialogEditText);
      myDialogEditText.setText(fileName);
      mBuilder1.setPositiveButton
      (
         R.string.str_alert_ok,new DialogInterface.OnClickListener()
         {
            public void onClick(DialogInterface dialoginterface, int i)
```

```
                {
                    /* 存档 */
                    String Filename = path + java.io.File.separator
                            + myDialogEditText.getText().toString();
                    java.io.BufferedWriter bw;
                    try
                    {
                        bw = new java.io.BufferedWriter(new java.io.FileWriter(
                                new java.io.File(Filename)));
                        String str = myEditText1.getText().toString();
                        bw.write(str, 0, str.length());
                        bw.newLine();
                        bw.close();
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
                    /* 回到 SDC_1 */
                    Intent intent = new Intent();
                    intent.setClass(SDC_2.this, SDC_1.class);
                    Bundle bundle = new Bundle();
                    /* 将路径传到 SDC_1 */
                    bundle.putString("path", path);
                    intent.putExtras(bundle);
                    startActivity(intent);
                    finish();
                }
            });
        mBuilder1.setNegativeButton(R.string.str_alert_cancel, null);
        mBuilder1.show();
    }
}
```

执行后的效果如图 2-21 所示，当单击一个按钮后会显示对应的存储信息，如图 2-22 所示。当单击图 2-22 中的“menu”后，会弹出两个 menu 选项，如图 2-23 所示。此时，可以通过这两个选项分别管理存储卡中的数据。

图 2-21　初始效果

图 2-22　SD 卡的文件信息

图 2-23　单击 MENU 按钮

注意：在“Eclipse+Android SDK”开发环境中，可以在可视化模式下管理 SD 卡中的文件，具体方法如下。

（1）单击 Eclipse 右上角的 DDMS 选项卡，如图 2-24 所示。

（2）在右侧列表中单击 mnt 选项，其中的 sdcard 文件夹就是系统模拟的 SD 卡目录，如图 2-25 所示。

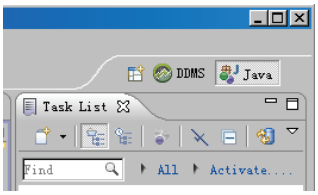


图 2-24　“DDMS”选项卡

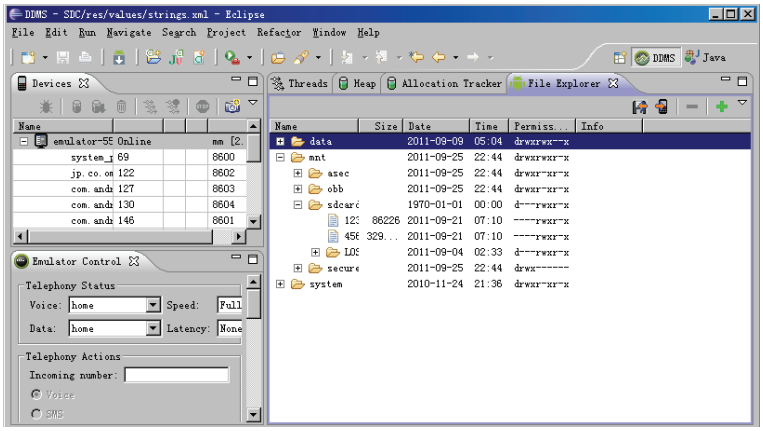


图 2-25　SD 卡目录

（3）通过顶部中的工具按钮可以对 SD 卡进行操作，如图 2-26 所示。

图 2-26　操作 SD 卡的按钮

图 2-22 中操作按钮的具体说明如下。

- ：下载 SD 卡中的文件到本地。
- ：上传本地文件到 SD 卡。

-  ：在 SD 卡中新建文件。
-  ：删除 SD 卡中的某个文件。

2.5　总结和网络开发有关的包

在 Android 系统中进行网络项目开发时，需要用到 Android SDK 中为我们提供的包，这些包的具体说明如表 2-2 所示。

表 2-2　Android SDK 中和网络有关的包

java.net	提供与联网有关的类，包括流和数据包（datagram）sockets、Internet 协议和常见的 HTTP 处理。该包是一个多功能网络资源。有经验的 Java 开发人员可以立即使用此包创建应用程序
java.io	虽然没有提供显式的联网功能，但是仍然非常重要。该包中的类由其他 Java 包中提供的 socket 和连接使用，并且还用于与本地文件（在与网络进行交互时会经常出现）的交互
java.nio	包含表示特定数据类型的缓冲区的类。适合用于两个基于 Java 语言的端点之间的通信
org.apache.*	表示许多为 HTTP 通信提供精确控制和功能的包，可以将 Apache 视为流行的开源 Web 服务器
android.net	除核心 java.net.* 类以外，包含额外的网络访问 socket。该包包括 URI 类，后者频繁用于 Android 应用程序开发，而不仅仅是传统的联网方面
android.net.http	包含处理 SSL 证书的类
android.net.wifi	包含在 Android 平台上管理有关 WiFi（802.11 无线 Ethernet）所有方面的类。并不是所有设备都配备了 WiFi 功能，特别是 Android 在 Motorola 和 LG 等手机制造商的“翻盖手机”领域获得了成功
android.telephony.gsm	包含用于管理和发送 SMS（文本）消息的类。一段时间后，可能会引入额外的包来为非 GSM 网络提供类似的功能，比如 CDMA 或 android.telephony.cdma 等网络

第二篇　核心技术篇

第 3 章　Java 中的网络通信基础

因为大多数 Android 应用程序是使用 Java 语言编写的，所以在 Android 网络应用程序中，通常使用 Java 网络包来传输网络数据。另外，在 Android 平台中，通常需要使用 Java 中的网络通信协议来开发 HTTP 通信项目。在本章的内容中，将详细讲解在 Android 系统中所用到的 Java 网络包技术，并详细讲解这些网络包的基本用法。

3.1　Java 中的网络包

在 Java 中提供了专门的包来支持网络应用，在包 java.net 中通过类 URL 和类 URLConnection 以编程方式访问 Web 服务，通过类 URLDecoder 和类 URLEncoder 提供了普通字符串和 application/x-www-form-urlencoded MIME 字符串相互转换的静态方法。

3.1.1　InetAddress 类详解

在 Java 应用中，使用类 InetAddress 来处理 IP 地址的数据传输工作。在类 InetAddress 中还存在如下的两个子类。

- Inet4Address：表示 Internet Protocol version 4（IPv4）地址。
- Inet6Address：表示 Internet Protocol version 6（IPv6）地址。

在 Java 应用中，类 InetAddress 没有构造器，只是提供了如下两个静态方法来获取 InetAddress 实例。

- getByName(String host)：根据主机获取对应的 InetAddress 对象。
- getByAddress(byte[] addr)：根据原始 IP 地址来获取对应的 InetAddress 对象。

在类 InetAddress 中，可以通过如下三个方法来获取 InetAddress 实例对应的 IP 地址和主机名。

- String getCanonicalHostName()：获取此 IP 地址的完全限定域名。
- String getHostAddress()：返回该 InetAddress 实例对应的 IP 地址字符串（以字符串形式）。
- String getHostName()：获取此 IP 地址的主机名。

在类 InetAddress 中包含了如下的两个重要方法。

- getLocalHost()：获取本机 IP 地址对应的 InetAddress 实例。
- isReachable()：测试是否可以到达该地址，该方法的实现将尽最大努力试图到达主机，但防火墙和服务器配置可能阻塞请求，使其在某些特定的端口可以访问时处于不可达的状态。如果可以获得权限，则将使用 ICMP ECHO REQUEST（PING 信息）进行应答；否则它将试图在目标主机的端口上建立 TCP 连接。

3.1.2　URLDecoder 类和 URLEncoder 类

在 Java 应用中，类 URLDecoder 和类 URLEncoder 的功能是实现普通字符串和 application/x-www-form-urlencoded MIME 字符串的相互转换。虽然 application/x-www-form-urlencoded MIME 不是普通的字符串，但是在现实应用中经常见到，例如搜索引擎网址中看似是乱码的内容，如图 3-1 所示。

图 3-1　MIME 字符串

当 URL 地址中包含非西欧字符的字符串时，系统会将这些非西欧字符串转换成如图 3-1 所示的特殊字符串。在编程过程中可以将普通字符串和这种特殊字符串相关转换，此功能是通过使用类 URLDecoder 和类 URLEncoder 实现的。

- 类 URLDecoder：包含一个 decode(String s,String enc)静态方法，它可以将看上去是乱码的特殊字符串转换成普通字符串。
- 类 URLEncoder：包含一个 encode(String s,String enc)静态方法，它可以将普通字符串转换成 application/x-www-form-urlencoded MIME 字符串。

在实际应用中，无须转换仅包含西欧字符的普通字符串和 application/x-www-form-urlencoded MIME 字符串。但是需要转换包含中文字符的普通字符串，转换的方法是每个中文字符占 2 个字节，每个字节可以转换成 2 个十六进制的数字，所以每个中文字符将转换成“%XX%XX”的形式。当采用不同的字符集时，每个中文字符对应的字节数并不完全相同，所以使用类 URLEncoder 和类 URLDecoder 进行转换时，也需要指定字符集。

3.1.3　URL 和 URLConnection

在计算机应用技术中，URL 是 Uniform Resource Locator 的缩写，意思是统一资源定位器。URL 是指向互联网“资源”的指针，这里的资源可以是简单的文件或目录，也可以是对更为复杂的对象引用，例如对数据库或搜索引擎的查询。在大多数情况下，URL 是由协议名、主机、端口和资源组成的，URL 需要满足如下的格式。

```
protocol://host:port/resourceName
```

例如下面就是一个 URL 地址。

```
http://www.163.com
```

JDK 为用户提供了一个 URI 类，其代表一个统一资源标识符。Java 的 URI 不能用于定位任何资源，它唯一的作用是解析。在 URL 中包含了一个可以打开到达该资源的输入流，因此可以将 URL 类理解为 URI 的一个特例。

在 URL 类中，提供了多个可以创建 URL 对象的构造器，一旦获得了 URL 对象之后，就可以调用下面的方法来访问该 URL 对应的资源。

- String getFile()：获取此 URL 的资源名。

- String getHost()：获取此 URL 的主机名。
- String getPath()：获取此 URL 的路径部分。
- int getPort()：获取此 URL 的端口号。
- String getProtocol()：获取此 URL 的协议名称。
- String getQuery()：获取此 URL 的查询字符串部分。
- URLConnection openConnection()：返回一个 URLConnection 对象，它表示到 URL 所引用的远程对象的连接。
- InputStream openStream()：打开与此 URL 的连接，并返回一个用于读取该 URL 资源的 InputStream。

在 URL 中，可以使用方法 openConnection()返回一个 URLConnection 对象，该对象表示应用程序和 URL 之间的通信链接。应用程序可以通过 URLConnection 实例向此 URL 发送请求，并读取 URL 引用的资源。

创建一个和 URL 连接的，并发送请求，读取此 URL 引用的资源的步骤如下：

（1）通过调用 URL 对象 openConnection()方法来创建 URLConnection 对象。

（2）设置 URLConnection 的参数和普通请求属性。

（3）如果只是发送 GET 方式请求，使用方法 connect 建立和远程资源之间的实际连接即可；如果需要发送 POST 方式的请求，需要获取 URLConnection 实例对应的输出流来发送请求参数。

（4）远程资源变为可用，程序可以访问远程资源的头字段或通过输入流读取远程资源的数据。

在建立和远程资源的实际连接之前，可以通过如下方法来设置请求头字段。

- setAllowUserInteraction：设置该 URLConnection 的 allowUserInteraction 请求头字段的值。
- setDoInput：设置该 URLConnection 的 doInput 请求头字段的值。
- setDoOutput：设置该 URLConnection 的 doOutput 请求头字段的值。
- setIfModifiedSince：设置该 URLConnection 的 ifModifiedSince 请求头字段的值。
- setUseCaches：设置该 URLConnection 的 useCaches 请求头字段的值。

除此之外，还可以使用如下方法来设置或增加通用头字段。

- setRequestProperty(String key, String value)：设置该 URLConnection 的 key，请求头字段的值为 value。
- addRequestProperty(String key, String value)：为该 URLConnection 的 key 请求头字段的增加 value 值，该方法并不会覆盖原请求头字段的值，而是将新值追加到原请求头字段中。

当发现远程资源可以使用后，使用如下方法访问头字段和内容。

- Object getContent()：获取该 URLConnection 的内容。
- String getHeaderField(String name)：获取指定响应头字段的值。
- getInputStream()：返回该 URLConnection 对应的输入流，用于获取 URLConnection 响应的内容。
- getOutputStream()：返回该 URLConnection 对应的输出流，用于向 URLConnection 发送请求参数。

❑ getHeaderField：根据响应头字段来返回对应的值。

因为在程序中需要经常访问某些头字段，所以 Java 为用户提供了如下方法来访问特定响应头字段的值。

❑ getContentEncoding：获取 content-encoding 响应头字段的值。

❑ getContentLength：获取 content-length 响应头字段的值。

❑ getContentType：获取 content-type 响应头字段的值。

❑ getDate()：获取 date 响应头字段的值。

❑ getExpiration()：获取 expires 响应头字段的值。

❑ getLastModified()：获取 last-modified 响应头字段的值。

在编程过程中，无需担心普通 Java 平台和 Android 平台的差异，URLConnection 在 Java 中的用法就是在 Android 应用中的用法。在接下来的内容中，将通过一个具体实例来讲解在 Android 系统中使用类 URLConnection 的基本方法。

实　例	功　能	源 码 路 径
实例 3-1	在手机屏幕中显示 QQ 空间中的照片	daima\3\QQ

本实例的功能是，在 Gallery 中显示某个指定 QQ 空间中的照片，这样做的好处是节约手机的内置存储空间。在具体实现上，需要将 URL 网址的相片实时处理下载后，以 InputStream 转换为 Bitmap，这样才能放入 BaseAdapter 中取用。在运行实例前，需要预先准备照片并上传到网络空间中，在获取相片的连接后，再以 String 数组方式放在程序中，并对 BaseAdapter 稍作修改，再加上对 URL 对象的访问以及 URLConnection 连接的处理。

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，在里面插入了一个 Gallery 控件来实现滑动相簿效果。具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/myLinearLayout"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <Gallery
   android:id="@+id/myGallery01"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   </Gallery>
</LinearLayout>
```

（2）编写主程序文件 QQ.java，其具体实现流程如下。

❑ 分别声明在 Gallery 中要显示的 5 副图片的地址栏字符串，具体代码如下。

```
public class QQ extends Activity
{
    private Gallery myGallery01;
  /* 地址栏字符串 */
```

```
        private String[] myImageURL = new String[]
          {
              "http://b236.photo.store.qq.com/psb?/"
                  + "V14NjoEt1VPfcS/vJ1LfTwqLKvNO2BB0PinvINjCSdCV3RcE4BhdhAD3Z4!/b/
dFr6toyNCAAA&bo=GgNIAgAAAAABB3M!&rf=viewer_4",
              "http://b241.photo.store.qq.com/psb?/"
                  + "V14NjoEt1VPfcS/*vD8rcyIXKGzlpNd8UPb45YdQrZy5DlNh*OWgtMAsDs!/b/
dJvMqI8ACAAA&bo=*gKAAgAAAAABB14!&rf=viewer_4",
              "http://b239.photo.store.qq.com/psb?/"
                  + "V14NjoEt1VPfcS/Xm942*QSSEV2kCwETcv9e4PHFtRcWrs1OnaTomAR.jM!/b/
dHu8gI58CAAA&bo=IANYAgAAAAABB1k!&rf=viewer_4",
              "http://b27.photo.store.qq.com/http_imgload.cgi?/"
                  + "rurl4_b=2a9dcf1fd909a7ed3ce8951f73860898bb7ff57a8cb7747c9f0eb6a02124850b709
c0b86f086a4ba5653eeb71dd4b01e4a58f407e2eec9433cd8d4bc0b88fda56260c2c8beb34ebab77b610c7131393f82e7
74ef&a=27&b=27",
              "http://b27.photo.store.qq.com/http_imgload.cgi?/"
                  + "rurl4_b=2a9dcf1fd909a7ed3ce8951f73860898158d252489f84e7d2a83d44c01b7bb12b2c
19ca0efdd555dba788407fd01e9de45524b11a9793f532624197bc8d14c84ae78ddebafe4357e4eedc60e9e5102243674
90bf&a=27&b=27" };
```

- 引入布局文件 main.xml，定义类成员 myContext 即 Context 对象，然后设置只有一个参数 C 的构造器。具体代码如下。

```
        public void onCreate(Bundle savedInstanceState)
        {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
          myGallery01 = (Gallery) findViewById(R.id.myGallery01);
          myGallery01.setAdapter(new myInternetGalleryAdapter(this));
        }
        /*  用 BaseAdapter */
        public class myInternetGalleryAdapter extends BaseAdapter
        {
          /*  类成员 myContext，Context 对象  */
          private Context myContext;
          private int mGalleryItemBackground; }
          /*构造器只有一个参数，即要存储的 Context */
          public myInternetGalleryAdapter(Context c)
          {
            this.myContext = c;
            TypedArray a = myContext
                .obtainStyledAttributes(R.styleable.Gallery);
            /*     获取 Gallery 属性的 Index id */
            mGalleryItemBackground = a.getResourceId(
                R.styleable.Gallery_android_galleryItemBackground, 0);
            /*  把对象的 styleable 属性能够反复使用  */
```

```
        a.recycle();
    }
```

- 定义方法 getCount()来返回全部已定义图片的总量，定义方法 getItem(int position)获取当前容器中图像数的数组 ID。具体代码如下。

```
/*    返回全部已定义图片的总量  */
public int getCount()
{
   return myImageURL.length;
}
/*  使用 getItem 方法获取当前容器中图像数的数组 ID */
public Object getItem(int position)
{
   return position;
}
public long getItemId(int position)
{
   return position;
}
```

- 定义方法 getScale，利用 getScale 根据中央位移量返回 views 的大小。具体代码如下。

```
/*  根据中央位移量，利用 getScale 返回 views 的大小(0.0f to 1.0f) */
public float getScale(boolean focused, int offset)
{
   /* Formula: 1 / (2 ^ offset) */
   return Math.max(0, 1.0f / (float) Math.pow(2, Math.abs(offset)));
}
```

执行后将在 Gallery 中显示指定的图片，如图 3-2 所示。

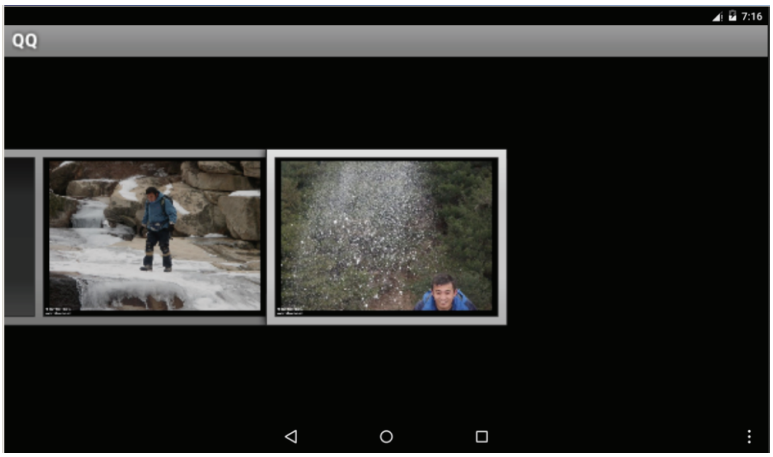


图 3-2　执行效果

3.1.4 HttpURLConnection

在 Java 应用程序中，URLConnection 是一个典型的抽象类。类 URLConnection 有两个直接子类，分别是 HttpURLConnection 和 JarURLConnection。通常 URL 可以通过给构造器传一个 String 类型的参数的方式，生成一个指向特定地址的 URL 实例。

在 Java 应用程序中，每个 HttpURLConnection 实例都可生成单个请求，但是其他实例可以透明地共享连接到 HTTP 服务器的基础网络。当生成请求后，在 HttpURLConnection 的 InputStream 或 OutputStream 上调用 close() 方法来释放与此实例关联的网络资源，但这对共享的持久连接没有任何影响。如果在调用 disconnect() 时发生持久连接空闲的情况，则可能会关闭基础套接字。

在日常应用中，经常不需要将网络中的图片保存在到我们的手机中，只是在线浏览一下内容而已。此时可以使用类 HttpURLConnection 打开网络链接，这样就可以获取链接中的数据。例如在下面的实例中，使用类 HttpURLConnection 连接并获取网络中的数据，将获取的数据用 InputStream 的方式保存在内存空间中。本实例的具体实现流程如下。

实　例	功　能	源 码 路 径
实例 3-2	在手机屏幕中显示网络中的图片	daima\3\tu

（1）编写布局文件 main.xml，主要实现代码如下。

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:background="@drawable/white"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >
   <TextView
      android:id="@+id/myTextView1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/app_name"/>
   <Button
      android:id="@+id/myButton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/str_button1" />
   <ImageView
      android:id="@+id/myImageView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center" />
</LinearLayout>
```

（2）编写主程序文件 tu.java，首先通过方法 getURLBitmap()将图片作为参数传入到创建

的 URL 对象中，然后通过方法 getInputStream()获取连接图片的 InputStream。文件 tu.java 的主要实现代码如下。

```
public class tu extends Activity
{
  private Button mButton1;
  private TextView mTextView1;
  private ImageView mImageView1;
  String uriPic = " http://b236.photo.store.qq.com/psb?/%22+%20%22V14NjoEt1VPfcS/vJ1LfTwqLKv
  NO2BB0PinvINjCSdCV3RcE4BhdhAD3Z4!/b/dFr6toyNCAAA&bo=GgNIAgAAAAABB3M!&rf=
viewer_4";
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mButton1 = (Button) findViewById(R.id.myButton1);
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mImageView1 = (ImageView) findViewById(R.id.myImageView1);
    mButton1.setOnClickListener(new Button.OnClickListener()
    {
      @Override
      public void onClick(View arg0)
      {
        /*  在 ImageView 中设置 Bitmap */
        mImageView1.setImageBitmap(getURLBitmap());
        mTextView1.setText("");
      }
    });
  }

  public Bitmap getURLBitmap()
  {
    URL imageUrl = null;
    Bitmap bitmap = null;
    try
    {
      /* new URL 对象将网址传入  */
      imageUrl = new URL(uriPic);
    }
    catch (MalformedURLException e)
    {
      e.printStackTrace();
    }
    Try

    {
```

```
            /* 取得连接 */
            HttpURLConnection conn = (HttpURLConnection) imageUrl
                  .openConnection();
            conn.connect();
            /* 取得返回的 InputStream */
            InputStream is = conn.getInputStream();
            /* 将 InputStream 变成 Bitmap */
            bitmap = BitmapFactory.decodeStream(is);
            /* 关闭 InputStream */
            is.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return bitmap;
    }
}
```

执行本实例代码，单击“单击后获取网络上的图片”按钮后会显示指定网址的图片，如图 3-3 所示。

图 3-3　执行效果

3.2　Android 网络接口

在 Android 系统中，可以使用网络接口 android.net.http 来处理网络中的 HTTP 请求。在 Android 应用程序中，android.net.http 是 android.net 中的一个典型包，在里面主要包含了处理 SSL 证书的各种类。在本节的内容中，将详细讲解 Android 网络接口的基本知识。

3.2.1　android.net.http 中的类

在 Android API 接口中，android.net.http 里面存在如下的 4 个类。

- AndroidHttpClient。
- SslCertificate。
- SslCertificate.DName。
- SslError。

上述各个类的具体说明如表 3-1 所示。

表 3-1　android.net.http 中的类

AndroidHttpClient	Apache DefaultHttpClient 面向 Android 系统开发的具有合理的默认配置以及注册框架的类，同样允许用户添加 HttpRequestInterceptor 类
HttpResponseCache	将 HTTP 和 HTTPS 的响应连接缓存到文件系统，这样当被再次使用时可以节省时间和带宽
SslCertificate	实现 SSL（密套接字协议层）认证信息（认证细节）的类
SslCertificate.DName	专用名字帮助类：一个三元组包括域名（CN）、组织结构（O）、次级组织结构（OU）
SslError	表示了一组一个或多 SSL 错误和与它们相关联的 SSL 证书

在 Android 应用程序中，“android.net.*”是通过对 Apache 的 HttpClient 封装来实现 HTTP 编程接口功能的。并且同时还提供了 HTTP 请求队列管理，以及 HTTP 连接池管理，以提高并发请求情况下（如转载网页时）的处理效率，除此之外还有网络状态监视等功能接口。

3.2.2　实战演练——在手机屏幕中传递 HTTP 参数

HTTP 是一种典型的网络传输协议，现实中的大多数网页都是通过“HTTP://WWW.”的形式实现数据显示的。在具体应用时，一些重要的 HTTP 数据都是通过其参数传递的。在本节的内容中，将通过一个具体实例来讲解在手机屏幕中传递 HTTP 参数的流程。

实　例	功　能	源 码 路 径
实例 3-3	在手机屏幕中传递 HTTP 参数	daima\3\httpSHI

1．实现思路

在计算机网络技术中，和 HTTP 有关的网络协议是 HTTP protocol。在 Android SDK 中内置了 Apache 的 HttpClient 模块，通过这些模块可以方便地编写出和 HTTP 有关的应用程序。在本实例中首先建立了和 HTTP 的连接代码，只有在连接之后才能获取 Web Server 返回的结果。然后插入了两个按钮，一个用于以 POST 方式获取网站数据，另外一个用于以 GET 方式获取数据，并以 TextView 对象来显示由服务器端返回的网页内容来显示连接结果。

2．具体实现

（1）编写布局文件 main.xml，主要代码如下。

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

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
      android:id="@+id/myTextView1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/title"/>
    <Button
      android:id="@+id/myButton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/str_button1" />
      <Button
      android:id="@+id/myButton2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/str_button2" />
</LinearLayout>
```

（2）编写文件 httpSHI.java，其具体实现流程如下。

- 引用 apache.http 中的相关类来实现 HTTP 联机操作，然后引用 java.io 和 java.util 相关类来读写信息。具体代码如下。

```
/*引用 apache.http 相关类来建立 HTTP 联机*/
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;
/*必需引用 java.io  与 java.util 相关类来读写档案*/
import irdc.httpSHI.R;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

- 使用 OnClickListener 方法监听单击后的第一个按钮事件，声明网址字符串并以 POST 方式建立联机，最后通过 mTextView1.setText 方法输出提示字符。具体代码如下。

```
/*设定 OnClickListener 方法来聆听 OnClick 事件*/
mButton1.setOnClickListener(new Button.OnClickListener()
{
  /*覆写 onClick 事件*/
  @Override
  public void onClick(View v)
  {
    /*声明网址字符串*/
    String uriAPI = "http://www.dubblogs.cc:8751/Android/Test/API/Post/index.php";
    /*建立 HTTP POST 联机*/
    HttpPost httpRequest = new HttpPost(uriAPI);
    /*
     * POST 运行传送变量必须用 NameValuePair[]数组存储
    */
    List <NameValuePair> params = new ArrayList <NameValuePair>();
    params.add(new BasicNameValuePair("str", "I am Post String"));
    try
    {
      httpRequest.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
      /*取得 HTTP 输出*/
      HttpResponse httpResponse = new DefaultHttpClient().execute(httpRequest);
      /*如果状态码为 200 */
      if(httpResponse.getStatusLine().getStatusCode()== 200)
      {
        /*获取应答字符串*/
        String strResult = EntityUtils.toString(httpResponse.getEntity());
        mTextView1.setText(strResult);
      }
      else
      {
        mTextView1.setText("Error Response: "+httpResponse.getStatusLine().toString());
      }
    }
    catch (ClientProtocolException e)
    {
      mTextView1.setText(e.getMessage().toString());
      e.printStackTrace();
    }
    catch (IOException e)
    {
      mTextView1.setText(e.getMessage().toString());
      e.printStackTrace();
    }
```

```
        catch (Exception e)
        {
            mTextView1.setText(e.getMessage().toString());
            e.printStackTrace();
        }
    }
});
```

- 使用 OnClickListener 方法来监听单击第二个按钮的事件，声明网址字符串并建立 GET 方式的联机功能，然后分别实现发出 HTTP 获取请求、获取应答字符串和删除冗余字符操作，最后通过 mTextView1.setText 方法输出提示字符。具体代码如下。

```
mButton2.setOnClickListener(new Button.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // TODO Auto-generated method stub
        /*声明网址字符串*/
        String uriAPI = "http://www.XXXX.cc:8751/index.php?str=I+am+Get+String";
        /*建立 HTTP GET 联机*/
        HttpGet httpRequest = new HttpGet(uriAPI);
        try
        {
            /*发出 HTTP 获取请求*/
            HttpResponse httpResponse = new DefaultHttpClient().execute(httpRequest);
            /*若状态码为 200 ok*/
            if(httpResponse.getStatusLine().getStatusCode()== 200)
            {
                /*获取应答字符串*/
                String strResult = EntityUtils.toString(httpResponse.getEntity());
                /*删除冗余字符*/
                strResult = eregi_replace("(\r\n|\r|\n|\n\r)","",strResult);
                mTextView1.setText(strResult);
            }
            else
            {
                mTextView1.setText("Error Response: "+httpResponse.getStatusLine().toString());
            }
        }
        catch (ClientProtocolException e)
        {
            mTextView1.setText(e.getMessage().toString());
            e.printStackTrace();
```

```
            }
            catch (IOException e)
            {
                mTextView1.setText(e.getMessage().toString());
                e.printStackTrace();
            }
            catch (Exception e)
            {
                mTextView1.setText(e.getMessage().toString());
                e.printStackTrace();
            }
        }
    });
```

❑ 定义替换字符串函数 eregi_replace 来替换掉一些非法字符，具体代码如下。

```
    /* 字符串替换函数 */
    public String eregi_replace(String strFrom, String strTo, String strTarget)
    {
        String strPattern = "(?i)"+strFrom;
        Pattern p = Pattern.compile(strPattern);
        Matcher m = p.matcher(strTarget);
        if(m.find())
        {
            return strTarget.replaceAll(strFrom, strTo);
        }
        else
        {
            return strTarget;
        }
    }
```

（3）在文件 AndroidManifest.xml 中声明网络连接权限，具体代码如下。

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

执行后的效果如图 3-4 所示，单击屏幕中的按钮能够以不同的方式获取 HTTP 参数。

图 3-4　单击“使用 PSST 方式”按钮后的效果

在 Android 系统中打开链接的通用代码如下。

```
Intent it = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.baidu.com"));
it.setClassName("com.Android.browser", "com.android.browser.BrowserActivity");
getContext().startActivity(it);
```

在 Android 系统中打开本地网页的通用代码如下。

```
Intent intent=new Intent();
intent.setAction("android.intent.action.VIEW");
Uri CONTENT_URI_BROWSERS = Uri.parse("content://com.android.htmlfileprovider/sdcard/123.html");
intent.setData(CONTENT_URI_BROWSERS);
intent.setClassName("com.android.browser", "com.android.browser.BrowserActivity");
startActivity(intent);
```

第 4 章　下载、上传数据

下载是指通过网络进行传输文件，把互联网或其他电子计算机上的信息保存到本地计算机上的一种网络活动。下载可以显式或隐式地进行，只要是获得本地计算机上所没有的信息的活动，都可以认为是下载，如在线观看。“上传”的反义词是“下载”，上传就是将信息从本地计算机传递到中央计算机（远程计算机）系统上，让网络上的人都能看到。将制作好的网页、文字、图片等发布到互联网，以便让其他人浏览、欣赏。这一过程称为上传。在 Android 网络开发中，下载和上传是两个十分常见的操作。在本章的内容中，将详细讲解在 Android 手机中实现远程下载、上传数据的基本知识，为读者步入本书后面知识的学习打下基础。

4.1　下载网络中的图片数据

在 Android 系统应用中，获取网络中的图片是一件耗时的操作，如果直接获取有可能会出现应用程序无响应的情况。对于这种情况，一般的解决方法就是使用线程来实现比较耗时的操作。在 Android 网络应用中，有如下三种获取网络图片的方法。

（1）直接获取，演示代码如下。

```
mImageView = (ImageView)this.findViewById(R.id.imageThreadConcept) ;
Drawable drawable = loadImageFromNetwork(IMAGE_URL);
mImageView.setImageDrawable(drawable) ;
```

对应的公用方法的实现代码如下。

```
private Drawable loadImageFromNetwork(String imageUrl)
{
    Drawable drawable = null;
    try {
        // 可以在这里通过文件名来判断，是否本地有此图片
        drawable = Drawable.createFromStream(
                new URL(imageUrl).openStream(), "image.jpg");
    } catch (IOException e) {
        Log.d("test", e.getMessage());
    }
    if (drawable == null) {
        Log.d("test", "null drawable");
    } else {
        Log.d("test", "not null drawable");
```

```
        }
        return drawable ;
    }
```

（2）后台线程获取 URL 图片，演示代码如下。

```
    mImageView = (ImageView)this.findViewById(R.id.imageThreadConcept) ;
    new Thread(new Runnable(){
        Drawable drawable = loadImageFromNetwork(IMAGE_URL);
        @Override
        public void run() {

            // post()  用于到 UI 主线程中更新图片
            mImageView.post(new Runnable(){
             @Override
             public void run() {
                 // TODO Auto-generated method stub
                 mImageView.setImageDrawable(drawable) ;
             }}) ;
             }

    }).start()   ;
```

（3）AsyncTask 获取 URL 图片，演示代码如下。

```
    mImageView = (ImageView)this.findViewById(R.id.imageThreadConcept) ;
    new DownloadImageTask().execute(IMAGE_URL) ;
    private class DownloadImageTask extends AsyncTask<String, Void, Drawable>
    {

            protected Drawable doInBackground(String... urls) {
                return loadImageFromNetwork(urls[0]);
            }

            protected void onPostExecute(Drawable result) {
                mImageView.setImageDrawable(result);
            }
    }
```

在接下来的内容中，将通过一个具体实例的实现过程，来讲解在 Android 手机中下载远程网络图片的方法。

题　目	目　的	源 码 路 径
实例 4-1	在 Android 手机中下载网络中的图片	daima\4\GetAPicture

本实例的具体实现流程如下。

（1）在布局文件 main.xml 中设置一个网址文本框，主要代码如下。

```
<EditText
  android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="http://xxxx.jpg"
android:id="@+id/path"
/>
```

在上述代码中，http://xxxx.jpg 是网络中一副图片的地址。

（2）编写主程序文件 GetAPictureFromInternetActivity.java，主要实现代码如下。

```
public class GetAPictureFromInternetActivity extends Activity {
    private EditText pathText;
    private ImageView imageView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pathText = (EditText) this.findViewById(R.id.path);
        imageView = (ImageView) this.findViewById(R.id.imageView);
    }
    public void showimage(View v){
     String path = pathText.getText().toString();
        try {
            Bitmap bitmap = ImageService.getImage(path);
            imageView.setImageBitmap(bitmap);
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(getApplicationContext(), R.string.error, 1).show();
        }
    }
}
```

执行后的效果如图 4-1 所示。

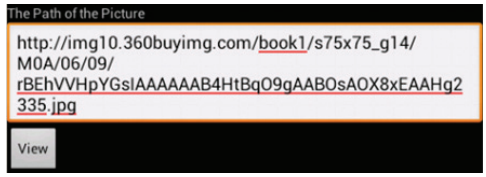


图 4-1　执行效果

4.2　下载网络中的 JSON 数据

JavaScript 对象表示法（JavaScript Object Notation，JSON），是一种轻量级的数据交换格式。JSON 是 JavaScript（Standard ECMA-262 3rd Edition - December 1999）的一个子集，采用

完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯（包括 C、C++、C#、Java、JavaScript、Perl、Python 等）。易于阅读和编写，同时也易于解析和生成，这些特性使 JSON 成为理想的数据交换语言。在本节的内容中，将详细讲解在 Android 系统中获取 JSON 数据的基本知识。

4.2.1 JSON 基础

简单来说，JSON 是 JavaScript 中的对象和数组，所以就是对象和数组这两种结构，通过这两种结构可以表示各种复杂的结构。

（1）对象

对象在 JSON 中表示为“{}”括起来的内容，数据结构为 {key：value,key：value,...}的键值对的结构，在面向对象的语言中，key 为对象的属性，value 为对应的属性值，所以很容易理解，取值方法为 对象.key 获取属性值，这个属性值的类型可以是 数字、字符串、数组、对象几种。

（2）数组

数组在 JSON 中是中括号“[]”括起来的内容，数据结构为 ["java","javascript","vb",...]，取值方式和所有语言一样，使用索引获取，字段值的类型可以是数字（Number）、字符串（String）、数组、对象（Object）几种。

经过对象、数组这两种结构就可以组合成复杂的数据结构。

和 XML 一样，JSON 也是基于纯文本的数据格式。JSON 的数据格式非常简单，可以用 JSON 传输一个简单的 String、Number、Boolean，也可以传输一个数组，或者一个复杂的 Object。

用 JSON 表示 String、Number 和 Boolean 的方法非常简单，例如用 JSON 表示一个简单的 String 数据“abc”，则其表示格式为：

```
"abc"
```

除了字符“"”，“\”，“/”和一些控制符（\b，\f，\n，\r，\t）需要编码外，其他 Unicode 字符可以直接输出。一个完整的 String 表示结构，如图 4-2 所示。

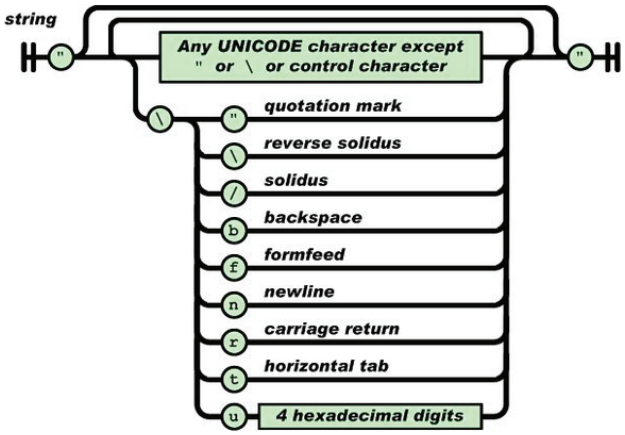


图 4-2　完整的 String 表示结构

4.2.2　远程下载服务器中的 JSON 数据

在接下来的内容中，将通过一个具体实例的实现过程，来详细讲解在 Android 系统中远程下载服务器中的 JSON 数据的方法。

题　　目	目　　的	源 码 路 径
实例 4-2	在手机屏幕中显示 QQ 空间中的照片	daima\4\json

本实例的具体实现流程如下。

（1）使用 Eclipse 新建一个 JavaEE 工程作为服务器端，设置工程名为“ServerForJSON”。自动生成工程文件后，打开文件 web.xml 进行配置，配置后的代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/
xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>ServerForJSON</display-name>
    <servlet>
        <display-name>NewsListServlet</display-name>
        <servlet-name>NewsListServlet</servlet-name>
        <servlet-class>com.guan.server.xml.NewsListServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NewsListServlet</servlet-name>
        <url-pattern>/NewsListServlet</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

（2）编写业务接口 Bean 的实现文件 NewsService.java，具体代码如下。

```
public interface NewsService {
    /**
     * 获取最新的视频资讯
     * @return
     */
    public List<News> getLastNews();
}
```

设置业务 Bean 的名称为 NewsServiceBean，实现文件 NewsServiceBean.java 的具体代码如下。

```
package com.guan.server.service.implement;
import java.util.ArrayList;
import java.util.List;
import com.guan.server.domain.News;
import com.guan.server.service.NewsService;
public class NewsServiceBean implements NewsService {
    /**
     * 获取最新的视频资讯
     * @return
     */
    public List<News> getLastNews(){
        List<News> newes = new ArrayList<News>();
        newes.add(new News(10, "aaa", 20));
        newes.add(new News(45, "bbb", 10));
        newes.add(new News(89, "Android is good", 50));
        return newes;
    }
}
```

（3）创建一个名为“News”的实现类，实现文件 News.java 的具体代码如下。

```
package com.guan.server.domain;
public class News {
    private Integer id;
    private String title;
    private Integer timelength;
    public News(Integer id, String title, Integer timelength) {
        this.id = id;
        this.title = title;
        this.timelength = timelength;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Integer getTimelength() {
        return timelength;
    }
    public void setTimelength(Integer timelength) {
```

```
        this.timelength = timelength;
    }
}
```

（4）编写文件 NewsListServlet，具体实现代码如下。

```
public class NewsListServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private NewsService newsService = new NewsServiceBean();
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        doPost(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        List<News> newes = newsService.getLastNews();//获取最新的视频资讯
            //     [{id:20,title:"xxx",timelength:90},{id:10,title:"xbx",timelength:20}]
            StringBuilder json = new StringBuilder();
            json.append('[');
            for(News news : newes){
                json.append('{');
                json.append("id:").append(news.getId()).append(",");
                json.append("title:\"").append(news.getTitle()).append("\",");
                json.append("timelength:").append(news.getTimelength());
                json.append("},");
            }
            json.deleteCharAt(json.length() - 1);
            json.append(']');
            request.setAttribute("json", json.toString());
            request.getRequestDispatcher("/WEB-INF/page/jsonnewslist.jsp").forward(request, response);
    }
}
```

（5）新建一个 JSP 文件 jsonnewslist.jsp，并引入 JSON 功能，具体实现代码如下。

```
<%@ page language="java" contentType="text/plain; charset=UTF-8" pageEncoding="UTF-8"%>${json}
```

（6）使用 Eclipse 新建一个名为“GetNewsInJSONFromInternet”的 Android 工程文件，在文件 AndroidManifest.xml 中申明对网络权限的应用，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.guan.internet.json"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="com.guan.internet.json.MainActivity"
```

```
                android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="8" />
<!-- 访问 internet 权限 -->
<uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

（7）编写主界面布局文件 mian.xml，具体实现代码如下。

```
<?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="wrap_content"
    android:id="@+id/listView"
    />
</LinearLayout>
```

在上述代码中，通过 ListView 控件列表显示获取的 JSON 数据。其中 ListView 的 Item 显示的数据为 item.xml，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content">
  <TextView
   android:layout_width="200dp"
   android:layout_height="wrap_content"
   android:id="@+id/title"
  />
    <TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:id="@+id/timelength"
  />
</LinearLayout>
```

（8）编写文件 MainActivity.java，功能是获取 JSON 数据并显示数据，具体实现代码如下。

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ListView listView = (ListView) this.findViewById(R.id.listView);
        String length = this.getResources().getString(R.string.length);
        try {
            List<News> newes = NewsService.getJSONLastNews();
            List<HashMap<String, Object>> data = new ArrayList<HashMap<String,Object>>();
            for(News news : newes){
                HashMap<String, Object> item = new HashMap<String, Object>();
                item.put("id", news.getId());
                item.put("title", news.getTitle());
                item.put("timelength", length+ news.getTimelength());
                data.add(item);
            }
            SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.item,
                    new String[]{"title", "timelength"}, new int[]{R.id.title, R.id.timelength});
            listView.setAdapter(adapter);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（9）编写文件 NewsService.java，定义方法 getJSONLastNews()请求前面搭建的 JavaEE 服务器，当获取 JSON 输入流后解析 JSON 的数据，并返回集合中的数据。文件 NewsService.java 的具体实现代码如下。

```
public class NewsService {
    /**
     * 获取最新视频资讯
     * @return
     * @throws Exception
     */
    public static List<News> getJSONLastNews() throws Exception{
        String path = "http://192.165.1.100:8080/ServerForJSON/NewsListServlet";
        HttpURLConnection conn = (HttpURLConnection) new URL(path).openConnection();
        conn.setConnectTimeout(5000);
        conn.setRequestMethod("GET");
        if(conn.getResponseCode()== 200){
            InputStream json = conn.getInputStream();
            return parseJSON(json);
```

```
            }
            return null;
        }
        private static List<News> parseJSON(InputStream jsonStream) throws Exception{
            List<News> list = new ArrayList<News>();
            byte[] data = StreamTool.read(jsonStream);
            String json = new String(data);
            JSONArray jsonArray = new JSONArray(json);
            for(int i = 0; i < jsonArray.length() ; i++){
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                int id = jsonObject.getInt("id");
                String title = jsonObject.getString("title");
                int timelength = jsonObject.getInt("timelength");
                list.add(new News(id, title, timelength));
            }
            return list;
        }
    }
```

到此为止，整个实例介绍完毕，执行后将成功获取服务器端 JSON 的数据。

4.3 下载某个网页的源码

当在 Android 系统中加载非本地的 HTML 文件时，会对此 HTML 进行缓存操作，同时会建一个数据库，用以保存 URL 地址对应的缓存文件名称、打开时间等信息。我们可以将 WebView 加载的 URL 地址作为查询条件以获取对应的缓存文件名称，匹配出的缓存文件就是完整的 HTML 代码。

题　目	目　的	源 码 路 径
实例 4-3	在手机屏幕中显示 QQ 空间中的照片	daima\4\WebCodeViewer

本实例的具体实现流程如下。

（1）编写界面布局文件 main.xml，具体实现代码如下。

```
<?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/path"
    />
    <EditText
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="http://www.baidu.com"
    android:id="@+id/path"
    />
    <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"
    android:onClick="showhtml"
    />
    <ScrollView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        />
    </ScrollView>
</LinearLayout>
```

（2）编写文件 HtmlService.java，定义一个获取网页代码的业务类 HtmlService，主要代码如下。

```
public class HtmlService {
    /**
     * 获取网页源码
     * @param path 网页路径
     * @return
     */
    public static String getHtml(String path) throws Exception {
        HttpURLConnection conn = (HttpURLConnection)new URL(path).openConnection();
        conn.setConnectTimeout(5000);
        conn.setRequestMethod("GET");
        if(conn.getResponseCode()== 200){
            InputStream inStream = conn.getInputStream();
            byte[] data = StreamTool.read(inStream);
            return new String(data);
        }
        return null;
    }
}
```

（3）编写文件 StreamTool.java，功能是将流转换为字节数组，主要代码如下。

```
public static byte[] read(InputStream inStream) throws Exception{
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    byte[] buffer = new byte[1024];
    int len = 0;
    while( (len = inStream.read(buffer)) != -1){
        outputStream.write(buffer, 0, len);
    }
    inStream.close();
    return outputStream.toByteArray();
}
```

（4）编写文件 WebCodeViewerActivity.java，当单击屏幕中的 GetWebCode 按钮时会获取指定网页的源码。文件 WebCodeViewerActivity.java 的具体实现代码如下。

```
public class WebCodeViewerActivity extends Activity {
    private EditText pathText;
    private TextView textView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        pathText = (EditText) this.findViewById(R.id.path);
        textView = (TextView) this.findViewById(R.id.textView);
    }
    public void showhtml(View v) {
        String path = pathText.getText().toString();
        try {
            String html = HtmlService.getHtml(path);
            textView.setText(html);
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(getApplicationContext(), R.string.error, Toast.LENGTH_LONG).show();
        }
    }
}
```

执行后的效果如图 4-3 所示。

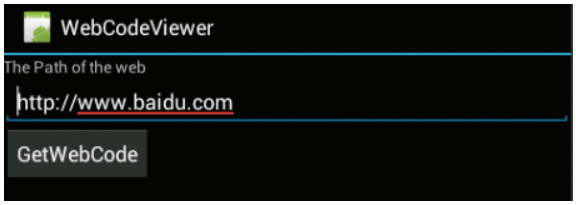


图 4-3　执行效果

4.4　多线程下载

线程可以理解为下载的通道，一个线程就是一个文件的下载通道，多线程也就是同时开启几个下载通道。当服务器提供下载服务时，下载者是共享带宽的，在优先级相同的情况下，服务器会对总下载线程进行平均分配。如果线程越多，下载的速度就会越快。在本节的内容中，将详细讲解在 Android 系统中实现多线程下载的过程。

4.4.1　多线程下载文件的过程

在 Android 系统中，实现多线程下载的基本过程如下。

（1）获取下载文件的长度，然后设置本地文件的长度。

```
HttpURLConnection.getContentLength();//获取下载文件的长度
RandomAccessFile file = new RandomAccessFile("QQWubiSetup.exe","rwd");
    file.setLength(filesize);//设置本地文件的长度
```

（2）根据文件长度和线程数目计算每条线程下载的数据长度和下载位置。假如文件的长度为 6MB，线程数为 3，那么每条线程下载的数据长度为 2MB，每条线程开始下载的位置如下图 4-4 所示。

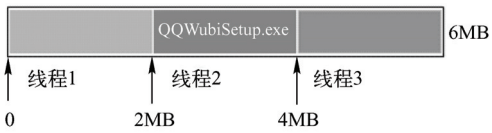


图 4-4　每条线程开始下载的位置

例如 10MB 大小文件，则使用 3 个线程来下载，具体说明如下。

- 线程下载的数据长度：使用表达式(10%3 == 0 ？ 10/3:10/3+1)进行计算，其中第 1 和第 2 个线程下载长度是 4MB，第 3 个线程下载长度为 2MB。
- 下载开始位置：使用表达式（线程 id*每条线程下载的数据长度）进行计算。
- 下载结束位置：使用表达式（(线程 id+1）*每条线程下载的数据长度-1）进行计算。

（3）使用 HTTP 的 Range 头字段指定每条线程从文件的什么位置开始下载，下载到什么位置为止，例如指定从文件的 2MB 位置开始下载，下载到位置(4MB-1byte)为止，代码如下。

```
HttpURLConnection.setRequestProperty("Range", "bytes=2097152-4194303");
```

（4）保存文件，使用类 RandomAccessFile 指定每条线程从本地文件的什么位置开始写入数据。

```
RandomAccessFile threadfile = new RandomAccessFile("QQWubiSetup.exe ","rwd");
threadfile.seek(2097152);//从文件的什么位置开始写入数据
```

4.4.2　在 Android 系统中实现多线程下载

在接下来的实例中，演示了在 Android 平台中通过 HTTP 协议实现断点续传下载的方法。

本实例是一个 HTTP 协议多线程断点下载应用程序，直接使用单线程下载 HTTP 文件对初学者来说是一件非常简单的事。多线程断点需要具备如下的功能。

- 多线程下载。
- 支持断点。

题　目	目　的	源 码 路 径
实例 4-4	下载在线铃声	daima\4\Multiple

本实例的功能是在 Android 手机中在线下载铃声，具体实现流程如下。

（1）打开 Eclipse，新建一个名为“MultipleThreadDownload”的动态 Web 工程。然后将一个 MP3 文件保存在 WebContent 目录下，最后发布服务器端的 Web 工程程序。

（2）打开 Eclipse，新建一个名为“MultipleThreadDownloadrAndroid”的 Android 工程。然后编写主程序文件 main.xml，具体实现代码如下。

```
<?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"
    >
    <!-- 下载路径提示文字 -->
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/path"
        />
    <!-- 下载路径输入框，此处为了方便测试，我们设置了默认的路径，可以根据需要在用户界面处修改 -->
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="http://192.164.1.100:8080/ServerForMultipleThreadDownloader/CNNRecordingFromWangjialin.mp3"
        android:id="@+id/path"
        />
     <!-- 水平 LinearLayout 布局，包裹下载按钮和暂停按钮 -->
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        >
        <!-- 下载按钮，用于触发下载事件 -->
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
                android:text="@string/button"
                android:id="@+id/downloadbutton"
                />
            <!-- 暂停按钮，在初始状态下为不可用 -->
            <Button
             android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/stopbutton"
                android:enabled="false"
                android:id="@+id/stopbutton"
                />
        </LinearLayout>
        <!-- 水平进度条，用图形化的方式实时显示进步信息 -->
        <ProgressBar
            android:layout_width="fill_parent"
            android:layout_height="18dp"
            style="?android:attr/progressBarStyleHorizontal"
            android:id="@+id/progressBar"
            />
        <!-- 文本框，用于显示实时下载的百分比 -->
         <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:id="@+id/resultView"
            />
    </LinearLayout>
```

（3）创建数据库管理类 DBOpenHelper，实现文件 DBOpenHelper.java 的具体代码如下。

```
    /**
     * SQLite 管理器，实现创建数据库和表，但版本变化时实现对表的数据库表的操作
     */
    public class DBOpenHelper extends SQLiteOpenHelper {
        private static final String DBNAME = "eric.db";        //设置数据库的名称
        private static final int VERSION = 1;                  //设置数据库的版本
        /**
         * 通过构造方法
         * @param context 应用程序的上下文对象
         */
        public DBOpenHelper(Context context) {
            super(context, DBNAME, null, VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {              //建立数据表
            db.execSQL("CREATE TABLE IF NOT EXISTS filedownlog (id integer primary key
```

```
autoincrement, downpath varchar(100), threadid INTEGER, downlength INTEGER)");
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            //当版本变化时系统会调用该回调方法
            db.execSQL("DROP TABLE IF EXISTS filedownlog");
            //此处是删除数据表，在实际的业务中一般是需要数据备份的
            onCreate(db);
            //调用 onCreate 方法重新创建数据表，也可以自己根据业务需要创建新的的数据表
        }
    }
```

（4）建立数据库业务操作类 FileService，此类的实现文件是 FileService.java，具体代码如下。

```
    /**
     * 业务 Bean，实现对数据的操作
     */
    public class FileService {
        private DBOpenHelper openHelper;                    //声明数据库管理器
        public FileService(Context context) {
            openHelper = new DBOpenHelper(context);     //根据上下文对象实例化数据库管理器
        }
        /**
         * 获取特定 URI 的每条线程已经下载的文件长度
         * @param path
         * @return
         */
        public Map<Integer, Integer> getData(String path){
            SQLiteDatabase db = openHelper.getReadableDatabase();
    //获取可读的数据库句柄，一般情况下在该操作的内部实现中其返回的其实是可写的数据库句柄
            Cursor cursor = db.rawQuery("select threadid, downlength from filedownlog where
downpath=?", new String[]{path}); //根据下载路径查询所有线程下载数据，返回的 Cursor 指向第一条记录之前
            Map<Integer, Integer> data = new HashMap<Integer, Integer>(); //建立一个哈希表用于存
放每条线程的已经下载的文件长度
            while(cursor.moveToNext()){        //从第一条记录开始开始遍历 Cursor 对象
                data.put(cursor.getInt(0), cursor.getInt(1));
                                                //把线程 id 和该线程已下载的长度设置进 data 哈希表中
                data.put(cursor.getInt(cursor.getColumnIndexOrThrow("threadid")), cursor.getInt
(cursor.getColumnIndexOrThrow("downlength")));
            }
            cursor.close();             //关闭 cursor，释放资源
            db.close();                 //关闭数据库
            return data;                //返回获得的每条线程和每条线程的下载长度
        }
        /**
         * 保存每条线程已经下载的文件长度
```

```
     * @param path  下载的路径
     * @param map  现在的 id 和已经下载的长度的集合
     */
    public void save(String path,    Map<Integer, Integer> map){
        SQLiteDatabase db = openHelper.getWritableDatabase();   //获取可写的数据库句柄
        db.beginTransaction(); //开始事务，因为此处要插入多批数据
        try{
            for(Map.Entry<Integer, Integer> entry : map.entrySet()){
        //采用 For-Each 的方式遍历数据集合
                db.execSQL("insert into filedownlog(downpath, threadid, downlength) values(?,?,?)",
                        new Object[]{path, entry.getKey(), entry.getValue()});
    //插入特定下载路径特定线程 ID 已经下载的数据
            }
            db.setTransactionSuccessful();       //设置事务执行的标志为成功
        }finally{                          //此部分的代码肯定是被执行的，如果不杀死虚拟机的话
            db.endTransaction();    //结束一个事务，如果事务设立了成功标志，则提交事务，
                                     否则会滚事务
        }
        db.close();                        //关闭数据库，释放相关资源
    }
    /**
     * 实时更新每条线程已经下载的文件长度
     * @param path
     * @param map
     */
    public void update(String path, int threadId, int pos){
        SQLiteDatabase db = openHelper.getWritableDatabase();   //获取可写的数据库句柄
        db.execSQL("update filedownlog set downlength=? where downpath=? and threadid=?",
                new Object[]{pos, path, threadId});              //更新特定下载路径下特定线
                                                                  程已经下载的文件长度
        db.close(); //关闭数据库，释放相关的资源
    }
    /**
     * 当文件下载完成后，删除对应的下载记录
     * @param path
     */
    public void delete(String path){
        SQLiteDatabase db = openHelper.getWritableDatabase();   //获取可写的数据库句柄
        db.execSQL("delete from filedownlog where downpath=?", new Object[]{path});
                                                                  //删除特定下载路径的所有线程记录
        db.close(); //关闭数据库，释放资源
    }
}
```

（5）编写文件下载类 FileDownloader，此类调用类 DownloadThread 实现具体的下载功能。类 FileDownloader 在文件 FileDownloader.java 中定义，具体实现代码如下。

```
public class FileDownloader {
    private static final String TAG = "FileDownloader";  //设置标签，方便 Logcat 日志记录
    private static final int RESPONSEOK = 200;           //响应码为 200，即访问成功
    private Context context;                             //应用程序的上下文对象
    private FileService fileService;                     //获取本地数据库的业务 Bean
    private boolean exited;                              //停止下载标志
    private int downloadedSize = 0;                      //已下载文件长度
    private int fileSize = 0;                            //原始文件长度
    private DownloadThread[] threads;                    //根据线程数设置下载线程池
    private File saveFile;                               //数据保存到的本地文件
    private Map<Integer, Integer> data = new ConcurrentHashMap<Integer, Integer>();
                                                         //缓存各线程下载的长度
    private int block;                                   //每条线程下载的长度
    private String downloadUrl;                          //下载路径

    /**
     * 获取线程数
     */
    public int getThreadSize() {
        return threads.length;                           //根据数组长度返回线程数
    }

    /**
     * 退出下载
     */
    public void exit(){
        this.exited = true; //设置退出标志为 true
    }
    public boolean getExited(){
        return this.exited;
    }
    /**
     * 获取文件大小
     * @return
     */
    public int getFileSize() {
        return fileSize;                                 //从类成员变量中获取下载文件的大小
    }
    /**
     * 累计已下载大小
     * @param size
     */
    protected synchronized void append(int size) {  //使用同步关键字解决并发访问问题
        downloadedSize += size;                          //把实时下载的长度加入到总下载长度中
    }
    /**
```

```
     * 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 synchronized void update(int threadId, int pos) {
        this.data.put(threadId, pos);                    //把制定线程 ID 的线程赋予最新的下载长度，以
                                                           前的值会被覆盖掉
        this.fileService.update(this.downloadUrl, threadId, pos);    //更新数据库中指定线程的下
                                                                      载长度
    }
    /**
     * 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;                          //对上下文对象赋值
            this.downloadUrl = downloadUrl;                  //对下载的路径赋值
            fileService = new FileService(this.context);
    //实例化数据操作业务 Bean，此处需要使用 Context，因为此处的数据库是应用程序私有
            URL url = new URL(this.downloadUrl);             //根据下载路径实例化 URL
            if(!fileSaveDir.exists()) fileSaveDir.mkdirs();  //如果指定的文件不存在，则创
                                                               建目录，此处可以创建多层目录
            this.threads = new DownloadThread[threadNum];    //根据下载的线程数创建下载
                                                               线程池
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                                                           //建立一个远程连接句柄，此时尚未真正连接
            conn.setConnectTimeout(5*1000); //设置连接超时时间为 5 秒
            conn.setRequestMethod("GET");  //设置请求方式为 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, */*");
                                                           //设置客户端可以接受的媒体类型
            conn.setRequestProperty("Accept-Language", "zh-CN");   //设置客户端语言
            conn.setRequestProperty("Referer", downloadUrl);
                                                           //设置请求的来源页面，便于服务端进行来源统计
            conn.setRequestProperty("Charset", "UTF-8"); //设置客户端编码
            conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 4.0; 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 3.30729)");          //设置用户代理
            conn.setRequestProperty("Connection", "Keep-Alive");     //设置 Connection 的方式
            conn.connect();  //和远程资源建立真正的连接，但尚无返回的数据流
            printResponseHeader(conn);         //答应返回的 HTTP 头字段集合
            if (conn.getResponseCode()==RESPONSEOK) {
```

```
                                    //此处的请求会打开返回流并获取返回的状态码，用于检查是否请求成
                                      功，当返回码为 200 时执行下面的代码
                                this.fileSize = conn.getContentLength();//根据响应获取文件大小
                                if (this.fileSize <= 0) throw new RuntimeException("Unkown file size ");
                                    //当文件大小为小于等于零时抛出运行时异常
                                String filename = getFileName(conn)                ;//获取文件名称
                                this.saveFile = new File(fileSaveDir, filename);    //根据文件保存目录和文
                                                                                     件名构建保存文件
                                Map<Integer, Integer> logdata = fileService.getData(downloadUrl);
                                                    //获取下载记录
                                if(logdata.size()>0){    //如果存在下载记录
                                    for(Map.Entry<Integer, Integer> entry : logdata.entrySet())
                                                    //遍历集合中的数据
                                        data.put(entry.getKey(), entry.getValue());
                                                    //把各条线程已经下载的数据长度放入 data 中
                                }
                                if(this.data.size()==this.threads.length){
                                    //如果已经下载的数据的线程数和现在设置的线程数相同时则计算所
                                     有线程已经下载的数据总长度
                                    for (int i = 0; i < this.threads.length; i++) {
                                                    //遍历每条线程已经下载的数据
                                        this.downloadedSize += this.data.get(i+1);
                                                    //计算已经下载的数据之和
                                    }
                                    print("已经下载的长度"+ this.downloadedSize + "个字节");
                                                    //打印出已经下载的数据总和
                                }

                                this.block = (this.fileSize % this.threads.length) ==0? this.fileSize /
this.threads.length : this.fileSize / this.threads.length + 1;    //计算每条线程下载的数据长度
                            }else{
                                print("服务器响应错误:" + conn.getResponseCode() + conn.getResponse
Message());                                         //打印错误
                                throw new RuntimeException("server response error ");
                                                    //抛出运行时服务器返回异常
                            }
                        } catch (Exception e) {
                            print(e.toString());            //打印错误
                            throw new RuntimeException("Can't connection this url");
                                                    //抛出运行时无法连接的异常
                        }
                    }
                    /**
                     * 获取文件名
                     */
                    private String getFileName(HttpURLConnection conn) {
```

```
                String filename = this.downloadUrl.substring(this.downloadUrl.lastIndexOf('/') + 1)
                                //从下载路径的字符串中获取文件名称
                if(filename==null || "".equals(filename.trim())){       //如果获取不到文件名称
                    for (int i = 0;; i++) {                             //无限循环遍历
                        String mine = conn.getHeaderField(i);
                                //从返回的流中获取特定索引的头字段值
                        if (mine == null) break;//如果遍历到了返回头末尾这退出循环
                        if("content-disposition".equals(conn.getHeaderFieldKey(i).toLowerCase())){
                                //获取 content-disposition 返回头字段，里面可能会包含文件名
                            Matcher m = Pattern.compile(".*filename=(.*)").matcher(mine.toLowerCase());
                                //使用正则表达式查询文件名
                            if(m.find()) return m.group(1);       //如果有符合正则表达规则的字符串
                        }
                    }
                    filename = UUID.randomUUID()+ ".tmp";       //由网卡上的标识数字(每个网卡都有唯一的标识号)以及 CPU 时钟的唯一数字生成的一个 16 字节的二进制作为文件名
                }
                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{  //进行下载，并抛出异常给调用者，如果有异常的话
                try {
                    RandomAccessFile randOut = new RandomAccessFile(this.saveFile, "rwd");
                    if(this.fileSize>0) randOut.setLength(this.fileSize);   //设置文件的大小
                    randOut.close();  //关闭该文件，使设置生效
                    URL url = new URL(this.downloadUrl);  //A URL instance specifies the location of a resource on the internet as specified by RFC 1738
                    if(this.data.size() != this.threads.length){  //如果原先未曾下载或者原先的下载线程数与现在的线程数不一致
                        this.data.clear();  //Removes all elements from this Map, leaving it empty.
                        for (int i = 0; i < this.threads.length; i++) {       //遍历线程池
                            this.data.put(i+1, 0);              //初始化每条线程已经下载的数据长度为 0
                        }
                        this.downloadedSize = 0;                //设置已经下载的长度为 0
                    }
                    for (int i = 0; i < this.threads.length; i++) {//开启线程进行下载
                        int downloadedLength = this.data.get(i+1);      //通过特定的线程 ID 获取该线程已经下载的数据长度
                        if(downloadedLength < this.block && this.downloadedSize < this.fileSize){//判
```

```
断线程是否已经完成下载，否则继续下载
                                this.threads[i] = new DownloadThread(this, url, this.saveFile, this.block,
this.data.get(i+1), i+1); //初始化特定 id 的线程
                                this.threads[i].setPriority(7);  //设置线程的优先级，Thread.NORM_
PRIORITY = 5 Thread.MIN_PRIORITY = 1 Thread.MAX_PRIORITY = 10
                                this.threads[i].start();    //启动线程
                            }else{
                                this.threads[i] = null;    //表明在线程已经完成下载任务
                            }
                        }
                        fileService.delete(this.downloadUrl);//如果存在下载记录，删除它们，然后重新添加
                        fileService.save(this.downloadUrl, this.data);    //把已经下载的实时数据写入数据库
                        boolean notFinished = true;//下载未完成
                        while (notFinished) {//  循环判断所有线程是否完成下载
                            Thread.sleep(900);
                            notFinished = false;//假定全部线程下载完成
                            for (int i = 0; i < this.threads.length; i++){
                                if (this.threads[i] != null && !this.threads[i].isFinished()) {
                                                                    //如果发现线程未完成下载
                                    notFinished = true;             //设置标志为下载没有完成
                                    if(this.threads[i].getDownloadedLength()== -1){
                                        //如果下载失败,再重新在已经下载的数据长度的基础上下载
                                        this.threads[i] = new DownloadThread(this, url, this.saveFile,
this.block, this.data.get(i+1), i+1);
                                        //重新开辟下载线程
                                        this.threads[i].setPriority(7);         //设置下载的优先级
                                        this.threads[i].start();                //开始下载线程
                                    }
                                }
                            }
                            if(listener!=null) listener.onDownloadSize(this.downloadedSize);
                                        //通知目前已经下载完成的数据长度
                        }
                        if(downloadedSize == this.fileSize) fileService.delete(this.downloadUrl);
                                        //下载完成删除记录
                    } catch (Exception e) {
                        print(e.toString());       //打印错误
                        throw new Exception("File downloads error");  //抛出文件下载异常
                    }
                    return this.downloadedSize;
                }
                /**
                 * 获取 Http 响应头字段
                 * @param http   HttpURLConnection 对象
                 * @return 返回头字段的 LinkedHashMap
                 */
                public static Map<String, String> getHttpResponseHeader(HttpURLConnection http) {
```

```
        Map<String, String> header = new LinkedHashMap<String, String>();
            //使用 LinkedHashMap 保证写入和遍历的时候的顺序相同，而且允许空值存在
        for (int i = 0;; i++) {     //此处为无限循环，因为不知道头字段的数量
            String fieldValue = http.getHeaderField(i);
                        //getHeaderField(int n)用于返回 第 n 个头字段的值。

            if (fieldValue == null) break; //如果第 i 个字段没有值了，则表明头字段部分已经循
                                            环完毕，此处使用 Break 退出循环
            header.put(http.getHeaderFieldKey(i), fieldValue);    //getHeaderFieldKey(int n)用于
                                                                   返回 第 n 个头字段的键。
        }
        return header;
    }
    /**
     * 打印 Http 头字段
     * @param http HttpURLConnection 对象
     */
    public static void printResponseHeader(HttpURLConnection http){
        Map<String, String> header = getHttpResponseHeader(http);      //获取 HTTP 响应头字段
        for(Map.Entry<String, String> entry : header.entrySet()){
                //使用 For-Each 循环的方式遍历获取的头字段的值，此时遍历的循序和输入
                 的顺序相同
            String key = entry.getKey()!=null ? entry.getKey()+ ":" : "";
                //当有键的时候这获取键，如果没有则为空字符串
            print(key+ entry.getValue()); //答应键和值的组合
        }
    }

    /**
     * 打印信息
     * @param msg  信息字符串
     */
    private static void print(String msg){
        Log.i(TAG, msg);       //使用 LogCat 的 Information 方式打印信息
    }
}
```

（6）类 DownloadThread 在文件 DownloadThread.java 中定义，具体实现代码如下。

```
/**
 * 下载线程，根据具体下载地址、保持到的文件、下载块的大小、已经下载的数据大小等信息进
行下载
 *
 */
public class DownloadThread extends Thread {
    private static final String TAG = "DownloadThread"; //定义 TAG，方便日子的打印输出
    private File saveFile;                          //下载的数据保存到的文件
```

```
        private URL downUrl;                    //下载的 URL
        private int block;                      //每条线程下载的大小
        private int threadId = -1;              //初始化线程 id 设置
        private int downloadedLength;           //该线程已经下载的数据长度
        private boolean finished = false;       //该线程是否完成下载的标志
        private FileDownloader downloader;      //文件下载器
        public DownloadThread(FileDownloader downloader, URL downUrl, File saveFile, int block, int
downloadedLength, int threadId) {
            this.downUrl = downUrl;
            this.saveFile = saveFile;
            this.block = block;
            this.downloader = downloader;
            this.threadId = threadId;
            this.downloadedLength = downloadedLength;
        }

        @Override
        public void run() {
            if(downloadedLength < block){//未下载完成
                try {
                    HttpURLConnection http = (HttpURLConnection) downUrl.openConnection();
                                                    //开启 HttpURLConnection 连接
                    http.setConnectTimeout(5 * 1000); //设置连接超时时间为 5 秒钟
                    http.setRequestMethod("GET");    //设置请求的方法为 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, */*");
                                                    //设置客户端可以接受的返回数据类型
                    http.setRequestProperty("Accept-Language", "zh-CN");
                                                    //设置客户端使用的语言问中文
                    http.setRequestProperty("Referer", downUrl.toString());
                                                    //设置请求的来源，便于对访问来源进行统计
                    http.setRequestProperty("Charset", "UTF-8");  //设置通信编码为 UTF-8
                    int startPos = block * (threadId - 1) + downloadedLength;//开始位置
                    int endPos = block * threadId -1;               //结束位置
                    http.setRequestProperty("Range", "bytes=" + startPos + "-"+ endPos);
            //设置获取实体数据的范围,如果超过了实体数据的大小会自动返回实际的数据大小
                    http.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 4.0;
Windows NT 4.2; 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 3.4.30729)");                                   //客户端用户代理
                    http.setRequestProperty("Connection", "Keep-Alive");    //使用长连接
                    InputStream inStream = http.getInputStream(); //获取远程连接的输入流
                    byte[] buffer = new byte[1024];                 //设置本地数据缓存的大小为 1MB
                    int offset = 0;                                 //设置每次读取的数据量
                    print("Thread " + this.threadId + " starts to download from position "+ startPos);
                                                    //打印该线程开始下载的位置
```

```
                RandomAccessFile threadFile = new RandomAccessFile(this.saveFile, "rwd");
        //If the file does not already exist then an attempt will be made to create it and it require that every update to the file's content be written synchronously to the underlying storage device.
                threadFile.seek(startPos);                    //文件指针指向开始下载的位置
                while (!downloader.getExited() && (offset = inStream.read(buffer, 0, 1024)) != -1) { //但用户没有要求停止下载，同时没有到达请求数据的末尾时候会一直循环读取数据
                    threadFile.write(buffer, 0, offset);      //直接把数据写到文件中
                    downloadedLength += offset;               //把新下载的已经写到文件中的数据加入到下载长度中
                    downloader.update(this.threadId, downloadedLength);
                        //把该线程已经下载的数据长度更新到数据库和内存哈希表中
                    downloader.append(offset);
                        //把新下载的数据长度加入到已经下载的数据总长度中
                }           //该线程下载数据完毕或者下载被用户停止
                threadFile.close();
                inStream.close();
                if(downloader.getExited())
                {
                    print("Thread " + this.threadId + " has been paused");
                }
                else
                {
                    print("Thread " + this.threadId + " download finish");
                }

                this.finished = true;     //设置完成标志为 true，无论是下载完成还是用户主动中断下载
            } catch (Exception e) { //出现异常
                this.downloadedLength = -1;               //设置该线程已经下载的长度为-1
                print("Thread "+ this.threadId+ ":"+ e);  //打印出异常信息
            }
        }
    }
    /**
     * 打印信息
     * @param msg 信息
     */
    private static void print(String msg){
        Log.i(TAG, msg);        //使用 Logcat 的 Information 方式打印信息
    }
    /**
     * 下载是否完成
     * @return
     */
    public boolean isFinished() {
        return finished;
```

```
        }
        /**
         * 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 getDownloadedLength() {
            return downloadedLength;
        }
    }
```

（7）在类 FileDownloader 中调用了类 DownloadProgressListener 来监听下载进度，类 DownloadProgressListener 在文件 DownloadProgressListener.java 中定义，具体实现代码如下。

```
public interface DownloadProgressListener {
    /**
     * 下载进度监听方法 获取和处理下载点数据的大小
     * @param size 数据大小
     */
    public void onDownloadSize(int size);
}
```

（8）编写文件主 Activity 文件 MultipleThreadDownloadAndroid.java，具体实现代码如下。

```
/**
* 主界面，负责下载界面的显示、与用户交互、响应用户事件等
*
*/
    public class MultipleThreadDownloadAndroid
    extends Activity {
        private static final int PROCESSING = 1;          //正在下载实时数据传输 Message 标志
        private static final int FAILURE = -1;            //下载失败时的 Message 标志
        private EditText pathText;                        //下载输入文本框
        private TextView resultView;                      //现在进度显示百分比文本框
    private Button downloadButton;                        //下载按钮，可以触发下载事件
    private Button stopbutton;                            //停止按钮，可以停止下载
    private ProgressBar progressBar;                      //下载进度条，实时图形化的显示进度信息
    //hanlder 对象的作用是用于向创建 Hander 对象所在的线程所绑定的消息队列发送消息并处
     理消息
    private Handler handler = new UIHander();
    private final class UIHander extends Handler{
     /**
    * 系统会自动调用的回调方法，用于处理消息事件
    * Mesaage 一般会包含消息的标志和消息的内容以及消息的处理器 Handler
    */
        public void handleMessage(Message msg) {
```

```
                switch (msg.what) {
                case PROCESSING:                        //下载时
                    int size = msg.getData().getInt("size"); //从消息中获取已经下载的数据长度
                    progressBar.setProgress(size);      //设置进度条的进度
                    float num = (float)progressBar.getProgress() / (float)
                    progressBar.getMax();               //计算已经下载的百分比，此处需要转换为
                                                         浮点数计算
                    int result = (int)(num * 100);      //把获取的浮点数计算结构转化为整数
                    resultView.setText(result+ "%");    //把下载的百分比显示在界面显示控件上
                    if(progressBar.getProgress()== progressBar.getMax()){
                    //当下载完成时
                        Toast.makeText(getApplicationContext(), R.string.
                        success, Toast.LENGTH_LONG).show(); //使用 Toast 技术提示用户下载完成
                    }
                    break;
                case -1:                                //下载失败时
                    Toast.makeText(getApplicationContext(), R.string.error,
                    Toast.LENGTH_LONG).show();          //提示用户下载失败
                    break;
                }
            }
        }
        @Override
        public void onCreate(Bundle savedInstanceState) {
            //应用程序启动时会首先调用且在应用程序整个生命周期中只会调用一次，适合于初始
              化工作
            super.onCreate(savedInstanceState);
            //使用父类的 onCreate 用做屏幕主界面的底层和基本绘制工作
            setContentView(R.layout.main);              //根据 XML 界面文件设置主界面
            pathText = (EditText) this.findViewById(R.id.path);    //获取下载 URL 的文本输入框对象
            resultView = (TextView) this.findViewById(R.id.resultView);
            //获取显示下载百分比文本控件对象
            downloadButton = (Button) this.findViewById(R.id.downloadbutton);
            //获取下载按钮对象
            stopbutton = (Button) this.findViewById(R.id.stopbutton);
            //获取停止下载按钮对象
            progressBar = (ProgressBar) this.findViewById(R.id.progressBar);
            //获取进度条对象
            ButtonClickListener listener = new ButtonClickListener();
            //声明并定义按钮监听器对象
            downloadButton.setOnClickListener(listener);    //设置下载按钮的监听器对象
            stopbutton.setOnClickListener(listener);        //设置停止下载按钮的监听器对象
        }
    /**
    * 按钮监听器实现类
    *
```

```
        */
        private final class ButtonClickListener implements View.OnClickListener{
            public void onClick(View v) {                    //该方法在注册了该按钮监听器的对象被单击
                                                               时会自动调用，用于响应单击事件
                switch (v.getId()) {                         //获取单击对象的 ID
                case R.id.downloadbutton:                        //当单击下载按钮时
                    String path = pathText.getText().toString();  //获取下载路径
                    if(Environment.getExternalStorageState().equals
                    (Environment.MEDIA_MOUNTED)){        //获取 SDCard 是否存在，当 SDCard 存在时
                        File saveDir = Environment.getExternalStorageDirectory();
                                                               //获取 SDCard 根目录文件
                        File saveDir1 = Environment.getExternalStoragePublicDirectory(Environment.
DIRECTORY_MOVIES);
                        File saveDir11 =    getApplicationContext().getExternalFilesDir(Environment.
DIRECTORY_MOVIES);
                        download(path, saveDir11);           //下载文件
                    }else{                                   //当 SDCard 不存在时
                        Toast.makeText(getApplicationContext(), R.string.
                        sdcarderror, Toast.LENGTH_LONG).show();       //提示用户 SDCard 不存在
                    }
                    downloadButton.setEnabled(false);        //设置下载按钮不可用
                    stopbutton.setEnabled(true);             //设置停止下载按钮可用
                    break;
                case R.id.stopbutton:                            //当单击停止下载按钮时
                    exit(); //停止下载
                    downloadButton.setEnabled(true);         //设置下载按钮可用
                    stopbutton.setEnabled(false);            //设置停止按钮不可用
                    break;
                }
            }
            private DownloadTask task;                       //声明下载执行者
            /**
             * 退出下载
             */
            public void exit(){
                if(task!=null) task.exit();                  //如果有下载对象时，退出下载
            }
            /**
             * 下载资源，生命下载执行者并开辟线程开始现在
             * @param path    下载的路径
             * @param saveDir     保存文件
             */
            private void download(String path, File saveDir){    //此方法运行在主线程
                task = new DownloadTask(path, saveDir);      //实例化下载任务
                new Thread(task).start();                    //开始下载
            }
```

```
        /*
         * UI 控件画面的重绘(更新)是由主线程负责处理的，如果在子线程中更新 UI 控件的值，更新后的值不会重绘到屏幕上
         */
        private final class DownloadTask implements Runnable{
            private String path;                    //下载路径
            private File saveDir;                   //下载到保存到的文件
            private FileDownloader loader;          //文件下载器(下载线程的容器)
            /**
             * 构造方法，实现变量初始化
             * @param path    下载路径
             * @param saveDir     下载要保存到的文件
    */
            public DownloadTask(String path, File saveDir) {
                this.path = path;
                this.saveDir = saveDir;
            }
            /**
             * 退出下载
             */
            public void exit(){
                if(loader!=null) loader.exit();     //如果下载器存在的话则退出下载
            }
            DownloadProgressListener downloadProgressListener = new
            DownloadProgressListener() {            //开始下载，并设置下载的监听器
                /**
                 * 下载的文件长度会不断的被传入该回调方法
                 */
                public void onDownloadSize(int size) {
                    Message msg = new Message();            //新建立一个 Message 对象
                    msg.what = PROCESSING;                  //设置 ID 为 1；
                    msg.getData().putInt("size", size);     //把文件下载的 size 设置进 Message 对象
                    handler.sendMessage(msg);               //通过 handler 发送消息到消息队列
                }
            };
            /**
             * 下载线程的执行方法，会被系统自动调用
             */
            public void run() {
                try {
                    loader = new FileDownloader(getApplicationContext(),
                    path, saveDir, 3);                              //初始化下载
                    progressBar.setMax(loader.getFileSize());       //设置进度条的最大刻度
                    loader.download(downloadProgressListener);
                } catch (Exception e) {
                    e.printStackTrace();
```

```
                    handler.sendMessage(handler.obtainMessage(FAILURE));
                    //下载失败时向消息队列发送消息
                    /*Message message = handler.obtainMessage();
                    message.what = FAILURE;*/
                }
            }
        }
    }
    }
```

（9）在文件 AndroidManifest.xml 中声明使用网络的权限和操作 SDCard 的权限，具体实现代码如下。

```
    <!-- 访问 internet 权限 -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <!-- 在 SDCard 中创建与删除文件权限 -->
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <!-- 往 SDCard 写入数据权限 -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

到此为止，整个实例介绍完毕，执行后的效果如图 4-5 所示。

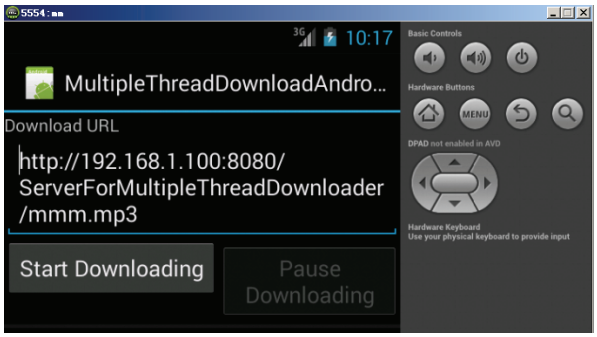


图 4-5　执行效果

4.5　上传文件到远程服务器

在本节的内容中，将通过一个具体实例的实现过程，详细讲解在 Android 系统中上传文件的基本方法。

题目	目的	源码路径
实例 4-5	上传文件到远程服务器	daima\4\chuan

本实例的具体实现流程如下。

（1）编写界面布局文件 main.xml，主要代码如下。

```
    <TextView
```

```
    android:id="@+id/myText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_title"
    android:textSize="20sp"
    android:textColor="@drawable/black"
    android:layout_x="10px"
    android:layout_y="12px"
  >
  </TextView>
  <TextView
    android:id="@+id/myText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:textColor="@drawable/black"
    android:layout_x="10px"
    android:layout_y="52px"
  >
  </TextView>
  <TextView
    android:id="@+id/myText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:textColor="@drawable/black"
    android:layout_x="10px"
    android:layout_y="102px"
  >
  </TextView>
  <Button
    android:id="@+id/myButton"
    android:layout_width="92px"
    android:layout_height="49px"
    android:text="@string/str_button"
    android:textSize="15sp"
    android:layout_x="90px"
    android:layout_y="170px"
  >
  </Button>
```

（2）编写主程序文件 chuan.java，其具体实现流程如下。

❑ 分别声明变量 newName、uploadFile 和 actionUrl，具体代码如下。

```
public class chuan extends Activity
{
  /* 变量声明
```

```
     * newName：上传后在服务器上的文件名称
     * uploadFile：要上传的文件路径
     * actionUrl：服务器上对应的程序路径 */
    private String newName="image.jpg";
    private String uploadFile="/data/data/irdc.example9/image.jpg";
    private String actionUrl="http://125.125.0.1/upload/upload.jsp";
    private TextView mText1;
    private TextView mText2;
    private Button mButton;
```

❑ 通过 mText1 对象获取文件路径，根据 mText2 对象设置上传网址，单击按钮后调用上传方法 uploadFile()。具体代码如下。

```
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   mText1 = (TextView) findViewById(R.id.myText2);
   mText1.setText("文件路径：\n"+uploadFile);
   mText2 = (TextView) findViewById(R.id.myText3);
   mText2.setText("上传网址：\n"+actionUrl);
   /* 设置 mButton 的 onClick 事件处理 */
   mButton = (Button) findViewById(R.id.myButton);
   mButton.setOnClickListener(new View.OnClickListener()
   {
      public void onClick(View v)
      {
         uploadFile();
      }
   });
}
```

❑ 定义方法 uploadFile()将文件上传至 Server，具体代码如下。

```
/* 上传文件至 Server 的方法 */
private void uploadFile()
{
   String end = "\r\n";
   String twoHyphens = "--";
   String boundary = "*****";
   try
   {
      URL url =new URL(actionUrl);
      HttpURLConnection con=(HttpURLConnection)url.openConnection();
      /* 允许 Input、Output，不使用 Cache */
      con.setDoInput(true);
      con.setDoOutput(true);
      con.setUseCaches(false);
```

```
    /* 设置传送的 method=POST */
    con.setRequestMethod("POST");
    /* setRequestProperty */
    con.setRequestProperty("Connection", "Keep-Alive");
    con.setRequestProperty("Charset", "UTF-8");
    con.setRequestProperty("Content-Type",
                           "multipart/form-data;boundary="+boundary);
    /* 设置 DataOutputStream */
    DataOutputStream ds =
      new DataOutputStream(con.getOutputStream());
    ds.writeBytes(twoHyphens + boundary + end);
    ds.writeBytes("Content-Disposition: form-data; " +
                  "name=\"file1\";filename=\"" +
                  newName +"\"" + end);
    ds.writeBytes(end);
    /* 取得文件的 FileInputStream */
    FileInputStream fStream = new FileInputStream(uploadFile);
    /* 设置每次写入 1024bytes */
    int bufferSize = 1024;
    byte[] buffer = new byte[bufferSize];
    int length = -1;
    /* 从文件读取数据至缓冲区 */
    while((length = fStream.read(buffer)) != -1)
    {
      /* 将资料写入 DataOutputStream 中 */
      ds.write(buffer, 0, length);
    }
    ds.writeBytes(end);
    ds.writeBytes(twoHyphens + boundary + twoHyphens + end);
    fStream.close();
    ds.flush();
    /* 取得 Response 内容 */
    InputStream is = con.getInputStream();
    int ch;
    StringBuffer b =new StringBuffer();
    while( ( ch = is.read() ) != -1 )
    {
      b.append( (char)ch );
    }
    /* 将 Response 显示在 Dialog 对话框中 */
    showDialog(b.toString().trim());
    /* 关闭 DataOutputStream */
    ds.close();
  }
  catch(Exception e)
  {
```

```
            showDialog(""+e);
        }
    }
```

❑ 定义方法 showDialog(String mess)来显示提示对话框，具体实现代码如下。

```
    /*  显示 Dialog 的方法*/
    private void showDialog(String mess)
    {
       new AlertDialog.Builder(example9.this).setTitle("Message")
        .setMessage(mess)
        .setNegativeButton("确定",new DialogInterface.OnClickListener()
        {
           public void onClick(DialogInterface dialog, int which)
           {
           }
        })
        .show();
    }
}
```

执行后单击“上传”按钮可以将指定的文件上传到服务器，如图 4-6 所示。

图 4-6　执行效果

4.6　GET 上传数据

在 Andorid 网络开发应用中，可以通过 GET 方式或 POST 方式上传数据。在本节的内容中，将详细讲解使用 GET 方式上传数据的基本方法。

4.6.1　使用 GET 方式上传数据的流程

通过 GET 和 POST 方式上传数据的具体区别如下。

❑ GET 上传的数据一般是很小的并且安全性不高的数据。

❑ POST 上传的数据适用于数据量大，数据类型复杂，数据安全性要求较高的地方。

在 Android 网络开发应用中，采用 GET 方式向服务器传递数据的基本步骤如下。

（1）利用 Map 集合对数据进行获取并进行数据处理，例如：

```
if (params!=null&&!params.isEmpty()) {
  for (Map.Entry<String, String> entry:params.entrySet()) {
    sb.append(entry.getKey()).append("=");
    sb.append(URLEncoder.encode(entry.getValue(),encoding));
    sb.append("&");
     }
    sb.deleteCharAt(sb.length()-1);

    }
```

（2）新建一个 StringBuilder 对象，例如：

```
sb=new StringBuilder()
```

（3）新建一个 HttpURLConnection 的 URL 对象，打开连接并传递服务器的 path，例如：

```
connection=(HttpURLConnection) new URL(path).openConnection();
```

（4）设置超时和连接的方式，例如：

```
connection.setConnectTimeout(5000);
connection.setRequestMethod("GET");
```

4.6.2　实战演练——采用 GET 方法向服务器传递数据

在本节的内容中，将通过一个具体实例的实现过程，介绍在 Android 系统中采用 GET 方式向服务器传递数据的基本方法。

题　　目	目　　的	源 码 路 径
实例 4-6	在 Android 系统中采用 GET 方式向服务器传递数据	daima\4\get

本实例的具体实现流程如下。

（1）打开 Eclipse，新建一个名为“ServerForGETMethod”的 Web 工程，并自动生成配置文件 web.xml。

（2）创建一个名为“ServletForGETMethod”的 Servlet，功能是接收并处理通过 GET 方式上传的数据。实现文件 ServletForGETMethod.java 的具体代码如下。

```
@WebServlet("/ServletForGETMethod")
public class ServletForGETMethod extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
//          String name= request.getParameter("name");
            String age= request.getParameter("age");
            System.out.println("name: " + name );
```

```
                System.out.println("age: " + age );

            }
        }
```

在上述代码中，为了避免出现中文乱码的问题，特意实现了 ISO8855-1 编码和 UTF-8 编码的转换处理。通过下面的代码，可以很好地解决乱码问题。

```
        <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
        <%
        String zh_value=new String(request.getParameter("zh_value").getBytes("ISO-8855-1"),"UTF-8")
        %>
```

由此可见，在使用 GET 方式传递数据时，需要使用如下的代码声明当前页的字符集。

```
        pageEncoding="UTF-8"
```

（3）在配置文件 web.xml 中配置 ServletForGETMethod，具体实现代码如下。

```
        <?xml version="1.0" encoding="UTF-8"?>
        <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/
xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.
sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
          <display-name>ServerForGETMethod</display-name>
          <servlet>
            <display-name>ServletForGETMethod</display-name>
            <servlet-name>ServletForGETMethod</servlet-name>
            <servlet-class>com.guan.internet.servlet.ServletForGETMethod</servlet-class>
          </servlet>
          <servlet-mapping>
            <servlet-name>ServletForGETMethod</servlet-name>
            <url-pattern>/ServletForGETMethod</url-pattern>
          </servlet-mapping>
          <welcome-file-list>
            <welcome-file>index.html</welcome-file>
            <welcome-file>index.htm</welcome-file>
            <welcome-file>index.jsp</welcome-file>
            <welcome-file>default.html</welcome-file>
            <welcome-file>default.htm</welcome-file>
            <welcome-file>default.jsp</welcome-file>
          </welcome-file-list>
        </web-app>
```

（4）打开 Eclipse，新建一个名为“UserInformation”的 Android 工程。然后编写界面布局文件 main.xml，具体实现代码如下。

```
        <?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/title"
        />
        <EditText
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:id="@+id/title"
        />
        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/length"
        />
        <EditText
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:numeric="integer"
          android:id="@+id/length"
        />
        <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button"
        android:onClick="save"
        />
    </LinearLayout>
```

（5）编写文件 UserInformationActivity.java，具体实现代码如下。

```
    public class UserInformationActivity extends Activity {
            private EditText titleText;
                private EditText lengthText;
                @Override
                public void onCreate(Bundle savedInstanceState) {
                      super.onCreate(savedInstanceState);
                      setContentView(R.layout.main);
                      titleText = (EditText) this.findViewById(R.id.title);
                      lengthText = (EditText) this.findViewById(R.id.length);
                }

                public void save(View v){
                  String title = titleText.getText().toString();
```

```
                String length = lengthText.getText().toString();
                try {
                    boolean result = false;

                    result = UserInformationService.save(title, length);

                    if(result){
                        Toast.makeText(this, R.string.success, 1).show();
                    }else{
                        Toast.makeText(this, R.string.fail, 1).show();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(this, R.string.fail, 1).show();
                }
            }
        }
```

（6）编写业务类的实现文件 UserInformationService.java，主要实现代码如下。

```
        public class UserInformationService {
            public static boolean save(String title, String length) throws Exception{
                String path = "http://192.165.1.100:8080/ServerForGETMethod/ServletForGETMethod";
                Map<String, String> params = new HashMap<String, String>();
                params.put("name", title);
                params.put("age", length);
                return sendGETRequest(path, params, "UTF-8");
            }
            /**
             * 发送 GET 请求
             * @param path 请求路径
             * @param params 请求参数
             * @return
             */
            private static boolean sendGETRequest(String path, Map<String, String> params, String encoding)
throws Exception{
                StringBuilder sb = new StringBuilder(path);
                if(params!=null && !params.isEmpty()){
                    sb.append("?");
                    for(Map.Entry<String, String> entry : params.entrySet()){
                        sb.append(entry.getKey()).append("=");
                        sb.append(URLEncoder.encode(entry.getValue(), encoding));
                        sb.append("&");
                    }
                    sb.deleteCharAt(sb.length() - 1);
                }
                HttpURLConnection conn = (HttpURLConnection) new URL(sb.toString()).openConnection();
```

```
                conn.setConnectTimeout(5000);
                conn.setRequestMethod("GET");
                if(conn.getResponseCode()== 200){
                        return true;
                }
                return false;
        }
}
```

（7）编写配置文件 AndroidManifest.xml，声明网络访问权限，主要代码如下。

```
<uses-sdk android:minSdkVersion="18" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name="com.guan.internet.userInformation.get.UserInformationActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
     <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

到此为止，整个实例讲解完毕，执行后的效果如图 4-7 所示。输入用户名和年龄后单击 save 按钮，会将输入的数据上传至服务器。

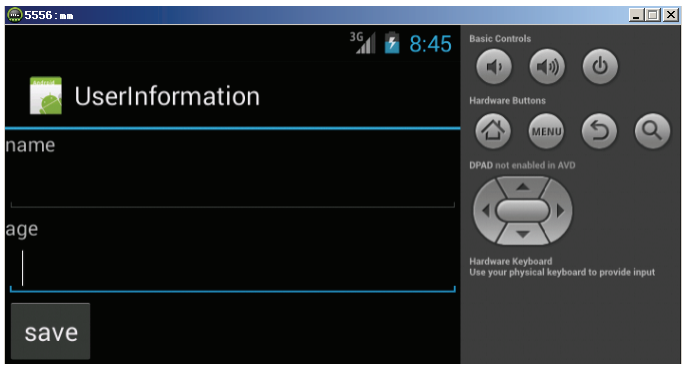


图 4-7　执行效果

4.7　POST 上传数据

在 Android 网络应用中，采用 POST 方式向服务器传递数据的基本步骤如下。

（1）利用 Map 集合对数据进行获取并进行数据处理，例如：

```
if (params!=null&&!params.isEmpty()) {
  for (Map.Entry<String, String> entry:params.entrySet()) {
    sb.append(entry.getKey()).append("=");
    sb.append(URLEncoder.encode(entry.getValue(),encoding));
    sb.append("&");
     }
    sb.deleteCharAt(sb.length()-1);
    }
```

（2）新建一个 StringBuilder 对象，得到 POST 传给服务器的数据，例如：

```
sb=new StringBuilder()
byte[] data=sb.toString().getBytes();
```

（3）新建一个 HttpURLConnection 的 URL 对象，打开连接并传递服务器的 path，例如：

```
connection=(HttpURLConnection) new URL(path).openConnection();
```

（4）设置超时和允许对外连接数据，例如：

```
connection.setDoOutput(true);
```

（5）设置连接的 setRequestProperty 属性，例如：

```
connection.setRequestProperty("Content-Type","application/x-www-form-urlencoded");
connection.setRequestProperty("Content-Length", data.length+"");
```

（6）得到连接输出流，例如：

```
outputStream =connection.getOutputStream();
```

（7）把得到的数据写入输出流中并刷新，例如：

```
 outputStream.write(data);
 outputStream.flush();
```

在接下来的内容中，也将通过一个具体实例的实现过程，介绍在 Android 系统中采用 POST 方式向服务器传递数据的基本方法。

题　　目	目　　的	源 码 路 径
实例 4-7	在 Android 系统中采用 POST 方式向服务器传递数据	daima\4\post

本实例的具体实现流程如下。

（1）打开 Eclipse，新建一个名为“ServerForPOSTMethod”的 Web 工程，并自动生成配置文件 web.xml。

（2）创建一个名为“ServletForPOSTMethod”的 Servlet，功能是接收并处理通过 POST

方式上传的数据。实现文件 ServletForPOSTMethod.java 的具体代码如下。

```
@WebServlet("/ServletForPOSTMethod")
public class ServletForPOSTMethod extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String name= request.getParameter("name");
        String age= request.getParameter("age");
        System.out.println("name from POST method: " + name );
        System.out.println("age from POST method: " + age );
    }
}
```

（3）在配置文件 web.xml 中配置 ServletForGETMethod，具体实现代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/
xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.
sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>ServerForPOSTMethod</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

（4）打开 Eclipse，新建一个名为“POST”的 Android 工程。然后编写界面布局文件 main.xml，具体实现代码如下。

```
<?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/title"
    />
    <EditText
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
```

```
    android:id="@+id/title"
    />
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/length"
    />
    <EditText
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:numeric="integer"
     android:id="@+id/length"
    />
    <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"
    android:onClick="save"
    />
</LinearLayout>
```

（5）编写文件 UploadUserInformationByPOSTActivity.java，具体实现代码如下。

```
public class UploadUserInformationByPOSTActivity extends Activity {
    private EditText titleText;
    private EditText lengthText;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        titleText = (EditText) this.findViewById(R.id.title);
        lengthText = (EditText) this.findViewById(R.id.length);
    }
    public void save(View v){
     String title = titleText.getText().toString();
     String length = lengthText.getText().toString();
     try {
            boolean result = false;
            result = UploadUserInformationByPostService.save(title, length);

            if(result){
                    Toast.makeText(this, R.string.success, 1).show();
                }else{
                    Toast.makeText(this, R.string.fail, 1).show();
                }
            } catch (Exception e) {
                e.printStackTrace();
```

```
                Toast.makeText(this, R.string.fail, 1).show();
            }
        }
    }
```

（6）编写业务类的实现文件 UploadUserInformationByPostService.java，主要实现代码如下。

```
public class UploadUserInformationByPostService {
    public static boolean save(String title, String length) throws Exception{
        String path = "http://192.165.1.100:8080/ServerForPOSTMethod/ServletForPOSTMethod";
        Map<String, String> params = new HashMap<String, String>();
        params.put("name", title);
        params.put("age", length);
        return sendPOSTRequest(path, params, "UTF-8");
    }
    /**
     * 发送 POST 请求
     * @param path  请求路径
     * @param params  请求参数
     * @return
     */
    private static boolean sendPOSTRequest(String path, Map<String, String> params, String encoding)
throws Exception{
        StringBuilder sb = new StringBuilder();
        if(params!=null && !params.isEmpty()){
            for(Map.Entry<String, String> entry : params.entrySet()){
                sb.append(entry.getKey()).append("=");
                sb.append(URLEncoder.encode(entry.getValue(), encoding));
                sb.append("&");
            }
            sb.deleteCharAt(sb.length() - 1);
        }
        byte[] data = sb.toString().getBytes();
        HttpURLConnection conn = (HttpURLConnection) new URL(path).openConnection();
        conn.setConnectTimeout(5000);
        conn.setRequestMethod("POST");
        conn.setDoOutput(true);//允许对外传输数据
        conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        conn.setRequestProperty("Content-Length", data.length+"");
        OutputStream outStream = conn.getOutputStream();
        outStream.write(data);
        outStream.flush();
        if(conn.getResponseCode()== 200){
            return true;
        }
        return false;
    }
```

```
    }
```

（7）编写配置文件 AndroidManifest.xml，声明网络访问权限，主要代码如下。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.guan.internet.userInformation.post"
    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="com.guan.internet.userInformation.post.UploadUserInformationByPOSTActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
<uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

整个实例讲解完毕，执行后的效果如图 4-8 所示。输入用户名和年龄后单击 save 按钮，会将输入的数据上传至服务器。

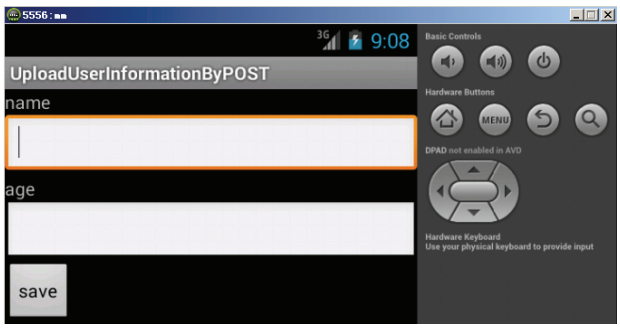


图 4-8　执行效果

第 5 章　Socket 数据通信

在网络传输应用中，通常使用 TCP、IP 和 UDP 这三种协议实现数据传输。在传输网络数据的过程中，需要通过一个双向的通信连接实现数据的交互。在这个传输过程中，通常将这个双向链路的一端称为 Socket，一个 Socket 通常由一个 IP 地址和一个端口号来确定。在 Java 编程应用中，Socket 是 Java 网络编程的核心，而 Java 是 Andord 应用开发的主流语言，所以在本章的内容中，将详细讲解在 Android 系统中使用 Socket 实现通信的基础知识。

5.1　Socket 编程初步

在网络编程中的两个主要问题，一个是如何准确地定位网络上的一台或多台主机；另一个就是找到主机后如何可靠高效地进行数据传输。在 TCP/IP 中 IP 层主要负责网络主机的定位、数据传输的路由，由 IP 地址可以唯一地确定 Internet 上的一台主机。而 TCP 层则提供面向应用可靠的或非可靠的数据传输机制，它是网络编程的主要对象。目前较为流行的网络编程模型是客户机/服务器（C/S）结构。即通信双方一方作为服务器等待客户提出请求并予以响应；客户则在需要服务时向服务器提出申请。服务器一般作为守护进程始终运行，监听网络端口，一旦有客户请求，就会启动一个服务进程来响应该客户，同时自己继续监听服务端口，使后来的客户也能及时得到服务。在接下来的内容中，将简要讲解 TCP/IP 和 UDP 的基本知识。

5.1.1　TCP/IP 基础

TCP/IP 是 Transmission Control Protocol/Internet Protocol 的简写，译名为传输控制协议/因特网互联协议，又名网络通信协议，是 Internet 最基本的协议、Internet 国际互联网络的基础，由网络层的 IP 和传输层的 TCP 组成。TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。TCP/IP 协议采用了四层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。也就是说，TCP 负责发现传输的问题，一旦发现问题便发出信号要求重新传输，直到所有数据安全正确地传输到目的地。而 IP 的功能是给因特网的每一台计算机规定一个地址。

TCP/IP 不是 TCP 和 IP 这两个协议的合称，而是指因特网整个 TCP/IP 协议族。从协议分层模型方面来讲，TCP/IP 由四个层次组成，分别是网络接口层、网络层、传输层、应用层。

其实 TCP/IP 并不完全符合 OSI 的七层参考模型，OSI（Open System Interconnect）是传统的开放式系统互联参考模型，是一种通信协议的七层抽象的参考模型，其中每一层执行某一特定任务。该模型的目的是使各种硬件在相同的层次上可以相互通信，这七层分别为：物

理层、数据链路层（网络接口层）、网络层（网络层）、传输层（传输层）、会话层、表示层和应用层。而 TCP/IP 采用了四层的层级结构，每一层都呼叫它的下一层所提供的网络来完成自己的需求。由于 TCP/IP 的设计者注重的是网络互联，允许通信子网（网络接口层）采用已有的或是将来有的各种协议，所以这个层次中没有提供专门的协议。实际上，TCP/IP 协议可以通过网络接口层连接到任何网络上，例如 X.25 交换网或 IEEE 802 局域网。

5.1.2 UDP

UDP 是 User Datagram Protocol 的简称，是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地、到达目的地的时间以及内容的正确性都是不能被保证的。

在现实网络数据传输过程中，大多数功能是由 TCP 和 UDP 实现，在接下来的内容中，将列出上述两种协议的主要特点，以便读者可以区分这两种数据传输协议。

TCP 的主要特点如下。

- 面向连接的协议，在 socket 之间进行数据传输之前必然要建立连接，所以在 TCP 中需要连接时间。
- TCP 传输数据大小限制，一旦连接建立起来，双方的 socket 就可以按统一的格式传输大的数据。
- TCP 是一个可靠的协议，它确保接收方完全正确地获取发送方所发送的全部数据。

UDP 的主要特点如下。

- 每个数据报中都给出了完整的地址信息，因此无需要建立发送方和接收方的连接。
- UDP 传输数据时是有大小限制的，每个被传输的数据报必须限定在 64KB 之内。
- UDP 是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方。

在日常应用中，可以根据如下两点来选择使用哪一种传输协议。

（1）TCP 在网络通信上有极强的生命力，例如远程连接协议（Telnet）和文件传输协议（FTP）都需要不定长度的数据被可靠地传输。但是可靠的传输是对数据内容正确性的检验必然占用计算机的处理时间和网络的带宽，因此 TCP 传输的效率不如 UDP 高。

（2）UDP 操作简单，而且需要较少的监护，因此通常用于对传输可靠性要求低的应用程序中。例如视频会议系统，并不要求音频视频数据绝对的正确，只要保证连贯性就可以了，这种情况下使用 UDP 会更合理一些。

5.1.3 基于 Socket 的 Java 网络编程

网络上的两个程序通过一个双向的通讯连接实现数据的交换，双向链路的一端称为一个 Socket。Socket 通常用来实现客户端和服务端的连接。Socket 是 TCP/IP 中的一个十分流行的数据传输方式，一个 Socket 由一个 IP 地址和一个端口号唯一确定。但是，Socket 所支持的协议种类也不止 TCP/IP 一种，因此两者之间是没有必然联系的。在 Java 环境下，Socket 编程主要是指基于 TCP/IP 的网络编程。

1．Socket 通信的过程

Server（服务）端 Listen（监听）某个端口是否有连接请求，Client（客户）端向 Server 端

发出 Connect（连接）请求，Server 端向 Client 端发回 Accept（接受）消息。一个连接就建立起来了。Server 端和 Client 端都可以通过 Send，Write 等方法与对方通信。

在 Java 网络编程应用中，对于一个功能齐全的 Socket 来说，其工作过程包含如下的基本步骤。

1）创建 Socket。

2）打开连接到 Socket 的输入/输出流。

3）按照一定的协议对 Socket 进行读/写操作。

4）关闭 Socket。

2．创建 Socket

在 Java 网络编程应用中，在包 java.net 中提供了两个类 Socket 和 ServerSocket，分别用来表示双向连接的客户端和服务端。这是两个封装得非常好的类，其中包含了如下的构造方法。

- Socket(InetAddress address, int port)。
- Socket(InetAddress address, int port, boolean stream)。
- Socket(String host, int prot)。
- Socket(String host, int prot, boolean stream)。
- Socket(SocketImpl impl)。
- Socket(String host, int port, InetAddress localAddr, int localPort)。
- Socket(InetAddress address, int port, InetAddress localAddr, int localPort)。
- ServerSocket(int port)。
- ServerSocket(int port, int backlog)。
- ServerSocket(int port, int backlog, InetAddress bindAddr)。

在上述构造方法中，参数 address、host 和 port 分别是双向连接中另一方的 IP 地址、主机名和端口号，参数 stream 指明 socket 是流 Socket 还是数据报 Socket，参数 localPort 表示本地主机的端口号，参数 localAddr 和 bindAddr 是本地机器的地址（ServerSocket 的主机地址），参数 impl 是 Socket 的父类，既可以用来创建 ServerSocket 又可以用来创建 Socket。参数 count 则表示服务端所能支持的最大连接数。例如：

```
Socket client = new Socket("127.0.01.", 80);
ServerSocket server = new ServerSocket(80);
```

注意：必须小心地选择端口号，每一个端口号提供一种特定的服务，只有给出正确的端口号，才能获得相应的服务。0～1023 的端口号为系统所保留，例如 HTTP 服务的端口号为 80，Telnet 服务的端口号为 21，FTP 服务的端口号为 23，所以我们在选择端口号时，最好选择一个大于 1023 的数以防止发生冲突。另外，在创建 Socket 时如果发生错误，将产生 IOException，在程序中必须相应地作出处理，在创建 Socket 或 ServerSocket 必须捕获或抛出异常。

5.2　TCP 编程详解

TCP/IP 是一种可靠的网络协议，能够在通信的两端各建立一个 Socket，从而在通信的两端之间形成网络虚拟链路。一旦建立了虚拟的网络链路，两端的程序就可以通过虚拟链路进

行通信。Java 语言对 TCP 网络通信提供了良好的封装，通过 Socket 对象可以代表两端的通信端口，并通过 Socket 产生的 I/O 流进行网络通信。在本章的内容中，将首先详细讲解 Java 应用中 TCP 编程的基本知识，为读者步入本章后面的 Android 编程打下基础。

5.2.1 使用 ServletSocket

在 Java 程序中，使用类 ServerSocket 接受其他通信实体的连接请求时，对象 ServerSocket 的功能是监听来自客户端的 Socket 连接，如果没有连接则一直处于等待状态。在类 ServerSocket 中包含了如下监听客户端连接请求的方法。

- Socket accept()：如果接收到一个客户端 Socket 的连接请求，该方法将返回一个与客户端 Socket 对应的 Socket，否则该方法将一直处于等待状态，线程也被阻塞。

为了创建 ServerSocket 对象，ServerSocket 类为我们提供了如下构造器。

- ServerSocket(int port)：用指定的端口 port 创建一个 ServerSocket，该端口应该是有一个有效的端口整数值：0～65535。
- ServerSocket(int port,int backlog)：增加一个用来改变连接队列长度的参数 backlog。
- ServerSocket(int port,int backlog,InetAddress localAddr)：在机器存在多个 IP 地址的情况下，允许通过 localAddr 这个参数来指定将 ServerSocket 绑定到指定的 IP 地址。

当使用 ServerSocket 后，需要使用 ServerSocket 中的 close()方法关闭该 ServerSocket。在通常情况下，因为服务器不会只接受一个客户端请求，而是会不断地接受来自客户端的所有请求，所以可以通过循环来不断地调用 ServerSocket 中的 accept()方法，例如下面的代码。

```
//创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
ServerSocket ss = new ServerSocket(30000);
//采用循环不断接受来自客户端的请求
while (true)
{
    //每当接受到客户端 Socket 的请求，服务器端也对应产生一个 Socket
    Socket s = ss.accept();
    //下面就可以使用 Socket 进行通信了
...
}
```

在上述代码中，创建的 ServerSocket 没有指定 IP 地址，所以 ServerSocket 会绑定到本机默认的 IP 地址。在代码中使用 40000 作为该 ServerSocket 的端口号，通常推荐使用 10000 以上的端口，主要是为了避免与其他应用程序的通用端口冲突。

5.2.2 使用 Socket

在客户端可以使用 Socket 构造器实现和指定服务器的连接，在 Socket 中可以使用如下的两个构造器。

- Socket(InetAddress/String remoteAddress, int port)：创建连接到指定远程主机、远程端口的 Socket，该构造器没有指定本地地址和本地端口，两个参数默认使用本地主机的

默认 IP 地址和系统动态指定的 IP 地址。

- Socket(InetAddress/String remoteAddress, int port, InetAddress localAddr, int localPort)：创建连接到指定远程主机、远程端口的 Socket，并指定本地 IP 地址和本地端口号，适用于本地主机有多个 IP 地址的情形。

在使用上述构造器指定远程主机时，既可使用 InetAddress 来指定，也可以使用 String 对象指定，在 Java 中通常使用 String 对象指定远程 IP，例如 192.168.2.23。当本地主机只有一个 IP 地址时，建议使用第一个方法，第一个方法更简单，例如下面的代码。

```
//创建连接到本机、30000 端口的 Socket
Socket s = new Socket("127.0.0.1" , 30000);
```

当程序执行上述代码后会连接到指定服务器，服务器端的 ServerSocket 的 accept()方法向下执行，于是服务器端和客户端就产生了一对互相连接的 Socket。上述代码连接到“远程主机”的 IP 地址是 127.0.0.1，此 IP 地址总是代表本机的 IP 地址。因为编者示例程序的服务器端、客户端都是在本机中运行，所以 Socket 连接到远程主机的 IP 地址使用 127.0.0.1。

当客户端、服务器端产生对应的 Socket 之后，程序无须再区分服务器端和客户端，而是通过各自的 Socket 进行通信。在 Socket 中提供如下两个方法获取输入流和输出流。

- InputStream getInputStream()：返回该 Socket 对象对应的输入流，让程序通过该输入流从 Socket 中取出数据。
- OutputStream getOutputStream()：返回该 Socket 对象对应的输出流，让程序通过该输出流向 Socket 中输出数据。

例如下面是一段 TCP 的服务器端程序。

源码路径：daima\5\tcpudp\src\Server.java

```
import java.net.*;
import java.io.*;
public class Server
{
    public static void main(String[] args)
            throws IOException
    {
        //创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
        ServerSocket ss = new ServerSocket(30000);
        //采用循环不断接受来自客户端的请求
        while (true)
        {
            //每当接受到客户端 Socket 的请求，服务器端也对应产生一个 Socket
            Socket s = ss.accept();
            //将 Socket 对应的输出流包装成 PrintStream
            PrintStream ps = new PrintStream(s.getOutputStream());
            //进行普通 IO 操作
            ps.println("圣诞快乐！ ");
            //关闭输出流，关闭 Socket
            ps.close();
```

```
                s.close();
            }
        }
    }
```

通过上述代码建立了 ServerSocket 监听，并且使用 Socket 获取了输出流，所以执行后不会显示任何信息。

而下面是一段 TCP 的客户端程序。

源码路径：光盘:\daima\5\tcpudp\src\Client.java

```
import java.net.*;
import java.io.*;
public class Client
{
    public static void main(String[] args)
            throws IOException
    {
            Socket socket = new Socket("127.0.0.1" , 30000);
            //将 Socket 对应的输入流包装成 BufferedReader
            BufferedReader br = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));
            //进行普通 IO 操作
            String line = br.readLine();
            System.out.println("来自服务器的数据：" + line);
            //关闭输入流、socket
            br.close();
            socket.close();
    }
}
```

上述代码使用 Socket 建立了与指定 IP、指定端口的连接，并使用 Socket 获取输入流读取数据。执行后的效果如图 5-1 所示。

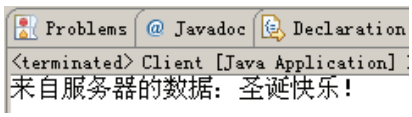


图 5-1　执行效果

由此可见，一旦使用 ServerSocket 和 Socket 建立网络连接之后，程序通过网络通信与普通 I/O 并没有太大的区别。如果先运行上面程序中的 Server 类，将看到服务器一直处于等待状态，因为服务器使用了死循环来接受来自客户端的请求；再运行 Client 类，将可看到程序输出：“来自服务器的数据：圣诞快乐！”，这表明客户端和服务器端通信成功。上述代码为了突出通过 ServerSocket 和 Socket 建立连接、并通过底层 I/O 流进行通信的主题，程序没有进行异常处理，也没有使用 finally 块来关闭资源。

5.2.3　TCP 中的多线程

在本章 5.2.2 的实例中，Server 和 Client 只是进行了简单的通信操作，当服务器接收到客户端连接之后，服务器向客户端输出一个字符串，而客户端也只是读取服务器的字符串后就退出了。在实际应用中，客户端可能需要和服务端保持长时间通信，即服务器需要不断地读取客户端数据，并向客户端写入数据，客户端也需要不断地读取服务器数据，并向服务器写入数据。

当使用 readLine()方法读取数据时，如果在该方法成功返回之前线程被阻塞，则程序无法继续执行。所以此服务器很有必要为每个 Socket 单独启动一条线程，每条线程负责与一个客户端进行通信。另外，因为客户端读取服务器数据的线程同样会被阻塞，所以系统应该单独启动一条线程，该线程专门负责读取服务器数据。

假设要开发一个聊天室程序，在服务端应该包含多个线程，其中每个 Socket 对应一个线程，该线程负责读取 Socket 对应输入流的数据（从客户端发送过来的数据），并将读到的数据向每个 Socket 输出流发送一遍（将一个客户端发送的数据“广播”给其他客户端），因此需要在服务端使用列表（List）来保存所有的 Socket。在具体实现这个聊天室程序时，为服务器提供了如下两个类。

- 创建 ServerSocket 监听的主类。
- 处理每个 Socket 通信的线程类。

接下来介绍具体实现流程，首先看下面的一段代码。

源码路径：daima\5\tcpudp\src\liao\server\IServer.java

```
package liao.server;
import java.net.*;
import java.io.*;
import java.util.*;
public class IServer
{
    //定义保存所有 Socket 的 ArrayList
    public static ArrayList<Socket> socketList = new ArrayList<Socket>();
      public static void main(String[] args)
            throws IOException
      {
            ServerSocket ss = new ServerSocket(30000);
            while(true)
            {
                  //此行代码会阻塞，将一直等待别人的连接
                  Socket s = ss.accept();
                  socketList.add(s);
                  //每当客户端连接后启动一条 ServerThread 线程为该客户端服务
                  new Thread(new Serverxian(s)).start();
            }
      }
}
```

在上述代码中，服务端只负责接受客户端 Socket 的连接请求，每当客户端 Socket 连接到该 ServerSocket 之后，程序将对应 Socket 加入 socketList 集合中保存，并为该 Socket 启动一个线程，该线程负责处理该 Socket 所有的通信任务。

然后看服务端线程类文件的主要代码。

源码路径：daima\5\tcpudp\src\liao\server\Serverxian.java

```
//负责处理每个线程通信的线程类
public class Serverxian implements Runnable
{
    //定义当前线程所处理的 Socket
    Socket s = null;
    //该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    public Serverxian(Socket s)
            throws IOException
    {
            this.s = s;
            //初始化该 Socket 对应的输入流
            br = new BufferedReader(new InputStreamReader(s.getInputStream()));
    }
    public void run()
    {
            try
            {
                    String content = null;
                    //采用循环不断从 Socket 中读取客户端发送过来的数据
                    while ((content = readFromClient()) != null)
                    {
                            //遍历 socketList 中的每个 Socket，
                            //将读到的内容向每个 Socket 发送一次
                            for (Socket s : IServer.socketList)
                            {
                                    PrintStream ps = new PrintStream(s.getOutputStream());
                                    ps.println(content);
                            }
                    }
            }
            catch (IOException e)
            {
                    //e.printStackTrace();
            }
    }
    //定义读取客户端数据的方法
    private String readFromClient()
    {
            try
```

```
            {
                return br.readLine();
            }
            //如果捕捉到异常，表明该 Socket 对应的客户端已经关闭
            catch (IOException e)
            {
                //删除该 Socket。
                IServer.socketList.remove(s);
            }
            return null;
        }
    }
```

在上述代码中，服务端线程类会不断读取客户端数据，在获取时使用 readFromClient()方法来读取客户端数据。如果读取数据过程中捕获到 IOException 异常，则说明此 Socket 对应的客户端 Socket 出现了问题，程序就会将此 Socket 从 socketList 中删除。当服务线程读到客户端数据之后会遍历整个 socketList 集合，并将该数据向 socketList 集合中的每个 Socket 发送一次，该服务线程将把从 Socket 中读到的数据向 socketList 中的每个 Socket 转发一次。

接下来开始客户端的编码工作，在本应用程序的每个客户端中应该包含如下两个线程。

- 第一个：功能是读取用户的键盘输入，并将用户输入的数据写入 Socket 对应的输出流中。
- 第二个：功能是读取 Socket 对应输入流中的数据（从服务器发送过来的数据），并将这些数据打印输出。其中负责读取用户键盘输入的线程由 Myclient 负责，也就是由程序的主线程负责。

客户端主程序文件的主要代码如下。

源码路径：daima\5\tcpudp\src\liao\server\Iclient.java

```
public class IClient
{
    public static void main(String[] args)
        throws IOException
    {
        Socket s = s = new Socket("127.0.0.1" , 30000);
        //客户端启动 ClientThread 线程不断读取来自服务器的数据
        new Thread(new ClientThread(s)).start();
        //获取该 Socket 对应的输出流
        PrintStream ps = new PrintStream(s.getOutputStream());
        String line = null;
        //不断读取键盘输入
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        while ((line = br.readLine()) != null)
        {
            //将用户的键盘输入内容写入 Socket 对应的输出流
            ps.println(line);
```

```
        }
    }
}
```

在上述代码中，当线程读到用户键盘输入的内容后，会将用户键盘输入的内容写入该 Socket 对应的输出流。当主线程使用 Socket 连接到服务器之后，会启动 ClientThread 来处理该线程的 Socket 通信。

最后编写客户端的线程处理文件，此线程负责读取 Socket 输入流中的内容，并将这些内容在控制台打印出来。具体代码如下。

源码路径：daima\5\tcpudp\src\liao\server\Clientxian.java

```
public class Clientxian implements Runnable
{
    //该线程负责处理的 Socket
    private Socket s;
    //该现成所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    public Clientxian(Socket s)
        throws IOException
    {
        this.s = s;
        br = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
    }
    public void run()
    {
        try
        {
            String content = null;
            //不断读取 Socket 输入流中的内容，并将这些内容打印输出
            while ((content = br.readLine()) != null)
            {
                System.out.println(content);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

上述代码能够不断获取 Socket 输入流中的内容，当获取 Socket 输入流中的内容后，直接将这些内容打印在控制台。先运行上面程序中的 IServer 类，该类运行后作为本应用程序的服务器，不会得到任何输出。接着可以运行多个 IClient——相当于多个聊天室客户端登录该服

务器，此时可以看到在任何一个客户端通过键盘输入一些内容后单击〈Enter〉键，将可看到所有客户端（包括自己）都会在控制台收到他刚刚输入的内容，这就简单实现了一个聊天室的功能。

5.2.4　实现非阻塞 Socket 通信

在 Java 应用程序中，可以使用 NIO（New I/O）API 来开发高性能网络服务器。当程序执行输入、输出操作后，在这些操作返回之前会一直阻塞该线程，服务器必须为每个客户端都提供一个独立线程进行处理。这说明前面的程序是基于阻塞式 API 的，当服务器需要同时处理大量客户端请求时，这种做法会降低性能。

在 Java 应用程序中可以用 NIO API 使服务器提供一个或有限几个线程来同时处理连接到服务器上的所有客户端。在 Java 的 NIO 中，为非阻塞式的 Socket 通信提供了下面的特殊类。

（1）Selector：是 SelectableChannel 对象的多路复用器，所有希望采用非阻塞方式进行通信的 Channel 都应该注册到 Selector 对象。可通过调用此类的静态 open()方法来创建 Selector 实例，该方法将使用系统默认的 Selector 来返回新的 Selector。Selector 可以同时监控多个 SelectableChannel 的 I/O 状况，是非阻塞 I/O 的核心。一个 Selector 实例有如下 3 个 SelectionKey 的集合。

- 所有 SelectionKey 集合：代表了注册在该 Selector 上的 Channel，这个集合可以通过 keys()方法返回。
- 被选择的 SelectionKey 集合：代表了所有可通过 select()方法监测到、需要进行 I/O 处理的 Channel，这个集合可以通过 selectedKeys()方法返回。
- 被取消的 SelectionKey 集合：代表了所有被取消注册关系的 Channel，在下一次执行 select()方法时，这些 Channel 对应的 SelectionKey 会被彻底删除，程序通常无须直接访问该集合。

除此之外，Selector 还提供了如下和 select()相关的方法。

- int select()：监控所有注册的 Channel，当它们中间有需要处理的 I/O 操作时，该方法返回，并将对应的 SelectionKey 加入被选择的 SelectionKey 集合中，该方法返回这些 Channel 的数量。
- int select(long timeout)：可以设置超时的 select()操作。
- int selectNow()：执行一个立即返回的 select()操作，相对于无参数的 select()方法而言，该方法不会阻塞线程。
- Selector wakeup()：使一个还未返回的 select()方法立刻返回。

（2）SelectableChannel：它代表可以支持非阻塞 I/O 操作的 Channel 对象，可以将其注册到 Selector 上，这种注册的关系由 SelectionKey 实例表示。在 Selector 对象中，可以使用 select()方法设置允许应用程序同时监控多个 I/O Channel。Java 程序可调用 SelectableChannel 中的 register()方法将其注册到指定 Selector 上，当该 Selector 中某些 SelectableChannel 上有需要处理的 I/O 操作时，程序可以调用 Selector 实例的 select()方法以获取它们的数量，并通过 selectedKeys()方法返回它们对应的 SelectKey 集合。这个集合的作用巨大，因为通过该集合就可以获取所有需要处理 I/O 操作的 SelectableChannel 集。

对象 SelectableChannel 支持阻塞和非阻塞两种模式，其中所有 Channel 默认都是阻塞模

式，我们必须使用非阻塞式模式才可以利用非阻塞 IO 操作。

在 SelectableChannel 中提供了如下两个方法来设置和返回该 Channel 的模式状态。

- SelectableChannel configureBlocking(boolean block)：设置是否采用阻塞模式。
- boolean isBlocking()：返回该 Channel 是否是阻塞模式。

不同的 SelectableChannel 所支持的操作不一样，例如 ServerSocketChannel 代表一个 ServerSocket，它就只支持 OP_ACCEPT 操作。在 SelectableChannel 中提供 ValidOps()方法返回一个 bit mask，表示这个 channel 上支持的 IO 操作来返回它支持的所有操作。

除此之外，SelectableChannel 还提供了如下方法获取它的注册状态。

- boolean isRegistered()：返回该 Channel 是否已注册在一个或多个 Selector 上。
- SelectionKey keyFor(Selector sel)：返回该 Channel 和 Selector 之间的注册关系，如果不存在注册关系，则返回 null。
- SelectionKey：该对象代表 SelectableChannel 和 Selector 之间的注册关系。
- ServerSocketChannel：支持非阻塞操作，对应于 java.net.ServerSocket 类，提供了 TCP 协议 I/O 接口，只支持 OP_ACCEPT 操作。该类也提供了 accept()方法，功能相当于 ServerSocket 提供的 accept()方法。
- SocketChannel：支持非阻塞操作，对应于 java.net.Socket 类，提供了 TCP 协议 I/O 接口，支持 OP_CONNECT，OP_READ 和 OP_WRITE 操作。这个类还实现了 ByteChannel 接口、ScatteringByteChannel 接口和 GatheringByteChannel 接口，所以可以直接通过 SocketChannel 来读写 ByteBuffer 对象。

服务器上的所有 Channel 都需要向 Selector 注册，包括 ServerSocketChannel 和 SocketChannel。该 Selector 则负责监视这些 Socket 的 I/O 状态，当其中任意一个或多个 Channel 具有可用的 I/O 操作时，该 Selector 的 select()方法将会返回大于 0 的整数，该整数值就表示该 Selector 上有多少个 Channel 具有可用的 I/O 操作，并提供了 selectedKeys()方法来返回这些 Channel 对应的 SelectionKey 集合。正是通过 Selector 才使得服务端只需要不断地调用 Selector 实例的 select()方法，这样就可以知道当前所有 Channel 是否有需要处理的 I/O 操作。当 Selector 上注册的所有 Channel 都没有需要处理的 I/O 操作时，将会阻塞 select()方法，此时调用该方法的线程被阻塞。

我们继续以聊天室为例，讲解非阻塞 Socket 通信在 Java 应用项目中的实现过程。我们的目标是，在服务端使用循环不断地获取 Selector 的 select()方法返回值，当该返回值大于 0 时就处理该 Selector 上被选择 SelectionKey 所对应的 Channel。在具体实现时，服务端使用 ServerSocketChannel 来监听客户端的连接请求，程序先调用它的 socket()方法获得关联 ServerSocket 对象，再用该 ServerSocket 对象绑定到指定监听的 IP 和端口。最后在服务端调用 Selector 的 select()方法来监听所有 Channel 上的 I/O 操作。

接下来开始具体编码，其中服务端的主要代码如下。

源码路径：daima\5\tcpudp\src\feizu\feizuServer.java

```
public class feizuServer
{
    //用于检测所有 Channel 状态的 Selector
    private Selector selector = null;
```

```
//定义实现编码、解码的字符集对象
private Charset charset = Charset.forName("UTF-8");
  public void init()throws IOException
  {
      selector = Selector.open();
      //通过 open 方法来打开一个未绑定的 ServerSocketChannel 实例
      ServerSocketChannel server = ServerSocketChannel.open();
      InetSocketAddress isa = new InetSocketAddress(
            "127.0.0.1", 30000);
      //将该 ServerSocketChannel 绑定到指定 IP 地址
      server.socket().bind(isa);
      //设置 ServerSocket 以非阻塞方式工作
      server.configureBlocking(false);
      //将 server 注册到指定 Selector 对象
      server.register(selector, SelectionKey.OP_ACCEPT);
      while (selector.select() > 0)
      {
            //依次处理 selector 上的每个已选择的 SelectionKey
            for (SelectionKey sk : selector.selectedKeys())
            {
                  //从 selector 上的已选择 Key 集中删除正在处理的 SelectionKey
                  selector.selectedKeys().remove(sk);
                  //如果 sk 对应的通道包含客户端的连接请求
                  if (sk.isAcceptable())
                  {
                        //调用 accept 方法接受连接，产生服务器端对应的 SocketChannel
                        SocketChannel sc = server.accept();
                        //设置采用非阻塞模式
                        sc.configureBlocking(false);
                        //将该 SocketChannel 也注册到 selector
                        sc.register(selector, SelectionKey.OP_READ);
                        //将 sk 对应的 Channel 设置成准备接受其他请求
                        sk.interestOps(SelectionKey.OP_ACCEPT);
                  }
                  //如果 sk 对应的通道有数据需要读取
                  if (sk.isReadable())
                  {
                        //获取该 SelectionKey 对应的 Channel，该 Channel 中有可读的数据
                        SocketChannel sc = (SocketChannel)sk.channel();
                        //定义准备执行读取数据的 ByteBuffer
                        ByteBuffer buff = ByteBuffer.allocate(1024);
                        String content = "";
                        //开始读取数据
                        try
                        {
                              while(sc.read(buff) > 0)
```

```
                        {
                            buff.flip();
                            content += charset.decode(buff);
                        }
                        //打印从该 sk 对应的 Channel 里读取到的数据
                        System.out.println("=====" + content);
                        //将 sk 对应的 Channel 设置成准备下一次读取
                        sk.interestOps(SelectionKey.OP_READ);
                    }
                    //如果捕捉到该 sk 对应的 Channel 出现了异常，即表明该 Channel
                    //对应的 Client 出现了问题，所以从 Selector 中取消 sk 的注册
                    catch (IOException ex)
                    {
                        //从 Selector 中删除指定的 SelectionKey
                        sk.cancel();
                        if (sk.channel() != null)
                        {
                            sk.channel().close();
                        }
                    }
                    //如果 content 的长度大于 0，即聊天信息不为空
                    if (content.length() > 0)
                    {
                        //遍历该 selector 里注册的所有 SelectKey
                        for (SelectionKey key : selector.keys())
                        {
                            //获取该 key 对应的 Channel
                            Channel targetChannel = key.channel();
                            //如果该 channel 是 SocketChannel 对象
                            if (targetChannel instanceof SocketChannel)
                            {
                                //将读到的内容写入该 Channel 中
                                SocketChannel dest = (SocketChannel)targetChannel;
                                dest.write(charset.encode(content));
                            }
                        }
                    }
                }
            }
        }
    }

    public static void main(String[] args)
        throws IOException
    {
        new feizuServer().init();
```

```
    }
}
```

通过上述代码，可在启动时马上建立一个监听连接请求的 ServerSocketChannel，并将该 Channel 注册到指定的 Selector，接着程序直接采用循环方法，不断地监控 Selector 对象的 select()方法的返回值，当该返回值大于 0 时处理该 Selector 上所有被选择的 SelectionKey。处理指定的 SelectionKey 之后立即将在 Selector 中的被选择的 SelectionKey 集合中删除该 SelectionKey。服务端的 Selecto 仅需要监听连接和读数据这两种操作，在处理连接操作时只需将接受连接后产生的 SocketChannel 注册到指定 Selector 对象即可。当处理读数据操作后，系统先从该 Socket 中读取数据，再将数据写入在 Selector 上注册的所有 Channel 中。

接下来开始编写客户端的代码，本应用的客户端程序需要如下两个线程。

- 负责读取用户的键盘输入，并将输入的内容写入 SocketChannel 中。
- 不断查询 Selector 对象的 select()方法的返回值。

客户端的主要代码如下。

源码路径：daima\5\tcpudp\src\feizu\feizuClient.java

```
public class feizuClient{
    //定义检测 SocketChannel 的 Selector 对象
    private Selector selector = null;
    //定义处理编码和解码的字符集
    private Charset charset = Charset.forName("UTF-8");
    //客户端 SocketChannel
    private SocketChannel sc = null;
    public void init()throws IOException
    {
        selector = Selector.open();
        InetSocketAddress isa = new InetSocketAddress("127.0.0.1", 30000);
        //调用 open 静态方法创建连接到指定主机的 SocketChannel
        sc = SocketChannel.open(isa);
        //设置该 sc 以非阻塞方式工作
        sc.configureBlocking(false);
        //将 SocketChannel 对象注册到指定 Selector
        sc.register(selector, SelectionKey.OP_READ);
        //启动读取服务器端数据的线程
        new ClientThread().start();
        //创建键盘输入流
        Scanner scan = new Scanner(System.in);
        while (scan.hasNextLine())
        {
            //读取键盘输入
            String line = scan.nextLine();
            //将键盘输入的内容输出到 SocketChannel 中
            sc.write(charset.encode(line));
        }
    }
```

```
//定义读取服务器数据的线程
private class ClientThread extends Thread
{
    public void run()
    {
        try
        {
            while (selector.select() > 0)
            {
                //遍历每个有可用 IO 操作 Channel 对应的 SelectionKey
                for (SelectionKey sk : selector.selectedKeys())
                {
                    //删除正在处理的 SelectionKey
                    selector.selectedKeys().remove(sk);
                    //如果该 SelectionKey 对应的 Channel 中有可读的数据
                    if (sk.isReadable())
                    {
                        //使用 NIO 读取 Channel 中的数据
                        SocketChannel sc = (SocketChannel)sk.channel();
                        ByteBuffer buff = ByteBuffer.allocate(1024);
                        String content = "";
                        while(sc.read(buff) > 0)
                        {
                            sc.read(buff);
                            buff.flip();
                            content += charset.decode(buff);
                        }
                        //打印输出读取的内容
                        System.out.println("聊天信息：" + content);
                        //为下一次读取作准备
                        sk.interestOps(SelectionKey.OP_READ);
                    }
                }
            }
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}
  public static void main(String[] args)
    throws IOException
{
    new feizuClient().init();
  }
```

```
}
```

上述客户端代码只有一条 SocketChannel，当此 SocketChannel 注册到指定的 Selector 后，程序会启动另一条线程来监测该 Selector。

在使用 NIO 来实现服务器时，甚至无须使用 ArrayList 来保存服务器中所有的 SocketChannel，因为所有的 SocketChannel 都需要注册到指定的 Selector 对象。除此之外，当客户端关闭时会导致服务器对应的 Channel 也抛出异常，而且在使用 NIO 时只有一条线程，如果该异常得不到处理将会导致整个服务器退出，所以程序捕捉了这种异常，并在处理异常时从 Selector 删除异常的 Channel 的注册。

5.3　UDP 编程

Java 为我们提供了 DatagramSocket 对象作为基于 UDP 的 Socket，可以使用 DatagramPacket 代表 DatagramSocket 发送或接收的数据报。

5.3.1　使用 DatagramSocket

DatagramSocket 本身只是码头，不维护状态，不能产生 I/O 流，其唯一的功能是接收和发送数据报。Java 语言使用 DatagramPacket 代表数据报，DatagramSocket 的接收和发送数据功能都是通过 DatagramPacket 对象实现的。

在 DatagramSocket 中有如下三个构造器。

- DatagramSocket()：负责创建一个 DatagramSocket 实例，并将该对象绑定到本机默认的 IP 地址、本机所有可用端口中随机选择的某个端口。
- DatagramSocket(int prot)：负责创建一个 DatagramSocket 实例，并将该对象绑定到本机默认的 IP 地址、指定的端口。
- DatagramSocket(int port, InetAddress laddr)：负责创建一个 DatagramSocket 实例，并将该对象绑定到指定的 IP 地址、指定的端口。

在 Java 程序中，通过上述任意一个构造器即可创建一个 DatagramSocket 实例。在创建服务器时必须创建指定端口的 DatagramSocket 实例，目的是保证其他客户端可以将数据发送到该服务器。一旦得到了 DatagramSocket 实例，就可以通过下面的两个方法接收和发送数据。

- receive(DatagramPacket p)：从该 DatagramSocket 中接收数据报。
- send(DatagramPacket p)：以该 DatagramSocket 对象向外发送数据报。

在使用 DatagramSocket 发送数据报时，DatagramSocket 并不知道将该数据报发送到何处，而是由 DatagramPacket 决定数据报的去处。就像码头并不知道每个集装箱的目的地，码头只是将这些集装箱发送出去，而集装箱本身包含了该集装箱的目的地。

当 C/S 程序使用 UDP 时，实际上并没有明确的服务器和客户端区分，因为双方都需要先建立一个 DatagramSocket 对象，用来接收或发送数据报，然后使用 DatagramPacket 对象作为传输数据的载体。通常固定 IP、固定端口的 DatagramSocket 对象所在的程序被称为服务器，因为该 DatagramSocket 可以主动接收客户端数据。

在 DatagramPacket 中包含了如下常用的构造器。

- DatagramPacket(byte buf[],int length)：以一个空数组来创建 DatagramPacket 对象，该对象的作用是接收 DatagramSocket 中的数据。
- DatagramPacket(byte buf[], int length, InetAddress addr, int port)：以一个包含数据的数组来创建 DatagramPacket 对象，创建该 DatagramPacket 时还指定了 IP 地址和端口——这就决定了该数据报的目的地。
- DatagramPacket(byte[] buf, int offset, int length)：以一个空数组来创建 DatagramPacket 对象，并指定接收到的数据放入 buf 数组中时从 offset 开始，最多放 length 个字节。
- DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)：创建一个用于发送的 DatagramPacket 对象，也多指定了一个 offset 参数。

在接收数据前，应该采用上面的第一个或第三个构造器生成一个 DatagramPacket 对象，给出接收数据的字节数组及其长度。然后调用 DatagramSocket 中的 receive()方法等待数据报的到来，此方法将一直等待（阻塞调用该方法的线程），直到收到一个数据报为止。例如下面的代码。

```
        //创建接受数据的 DatagramPacket 对象
DatagramPacket packet=new DatagramPacket(buf, 256);
//接收数据
socket.receive(packet);
```

在发送数据之前，调用第二个或第四个构造器创建 DatagramPacket 对象，此时的字节数组里存放了想发送的数据。除此之外，还要给出完整的目的地址，包括 IP 地址和端口号。发送数据是通过 DatagramSocket 的方法 send()实现的，方法 send()根据数据报的目的地址来传递数据报。例如下面的代码。

```
//创建一个发送数据的 DatagramPacket 对象
DatagramPacket packet = new DatagramPacket(buf, length, address, port);
//发送数据报
socket.send(packet);
```

DatagramPacket 为我们提供了 getData()方法，此方法可以返回 DatagramPacket 对象中封装的字节数组。

当服务器（也可以为客户端）接收到一个 DatagramPacket 对象后，如果想向该数据报的发送者“反馈”一些信息，但由于 UDP 是面向非连接的，所以接收者并不知道每个数据报由谁发送过来，但程序可以调用 DatagramPacket 的如下三个方法来获取发送者的 IP 和端口信息。

- InetAddress getAddress()：返回某台机器的 IP 地址，当程序准备发送此数据报时，该方法返回此数据报的目标机器的 IP 地址；当程序刚刚接收到一个数据报时，该方法返回该数据报的发送主机的 IP 地址。
- int getPort()：返回某台机器的端口，当程序准备发送此数据报时，该方法返回此数据报的目标机器的端口；当程序刚刚接收到一个数据报时，该方法返回该数据报的发送主机的端口。
- SocketAddress getSocketAddress()：返回完整的 SocketAddress，通常由 IP 地址和端口组成。当程序准备发送此数据报时，该方法返回此数据报的目标 SocketAddress；当程

序刚刚接收到一个数据报时，该方法返回该数据报的源 SocketAddress。

上述 getSocketAddress 方法的返回值是一个 SocketAddress 对象，该对象实际上就是一个 IP 地址和一个端口号，也就是说 SocketAddress 对象封装了一个 InetAddress 对象和一个代表端口的整数，所以使用 SocketAddress 对象可以同时代表 IP 地址和端口。

例如下面是一段实现 UDP 的服务端代码。

源码路径：daima\5\tcpudp\src\UdpServer.java

```
public class UdpServer
{
    public static final int PORT = 30000;
    //定义每个数据报的最大大小为 4K
    private static final int DATA_LEN = 4096;
    //定义该服务器使用的 DatagramSocket
    private DatagramSocket socket = null;
    //定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
    //以指定字节数组创建准备接受数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff , inBuff.length);
    //定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket;
    //定义一个字符串数组，服务器发送该数组的元素
    String[] books = new String[]
    {
        "AAA",
        "BBB",
        "CCC",
        "DDD"
    };
    public void init()throws IOException
    {
        try
        {
            //创建 DatagramSocket 对象
            socket = new DatagramSocket(PORT);
            //采用循环接受数据
            for (int i = 0; i < 1000 ; i++ )
            {
                //读取 Socket 中的数据，读到的数据放在 inPacket 所封装的字节数组里。
                socket.receive(inPacket);
                //判断 inPacket.getData()和 inBuff 是否是同一个数组
                System.out.println(inBuff == inPacket.getData());
                //将接收到的内容转成字符串后输出
                System.out.println(new String(inBuff ,
                    0 , inPacket.getLength()));
                //从字符串数组中取出一个元素作为发送的数据
```

```
                byte[] sendData = books[i % 4].getBytes();
                //以指定字节数组作为发送数据、以刚接受到的 DatagramPacket 的
                //源 SocketAddress 作为目标 SocketAddress 创建 DatagramPacket。
                outPacket = new DatagramPacket(sendData ,
                    sendData.length , inPacket.getSocketAddress());
                //发送数据
                socket.send(outPacket);
            }
        }
        //使用 finally 块保证关闭资源
        finally
        {
            if (socket != null)
            {
                socket.close();
            }
        }
    }
    public static void main(String[] args)
        throws IOException
    {
        new UdpServer().init();
    }
}
```

上述代码使用 DatagramSocket 实现了 C/S 结构的网络通信程序，其中服务端使用 1000 次循环来读取 DatagramSocket 中的数据报，每当读到内容之后便向该数据报的发送者送回一条信息。

接下看客户端的实现代码，客户端代码与服务端类似，也是采用循环，不断地读取用户键盘输入，每当读到用户输入内容后就将该内容封装成 DatagramPacket 数据报，再将该数据报发送出去。然后把 DatagramSocket 中的数据读入接收用的 DatagramPacket 中（实际上是读入该 DatagramPacket 所封装的字节数组中）。例如下面是一段实现 UDP 的客户端代码。

源码路径：daima\5\tcpudp\src\UdpClient.java

```
public class UdpClient{
    //定义发送数据报的目的地
    public static final int DEST_PORT = 30000;
    public static final String DEST_IP = "127.0.0.1";
    //定义每个数据报的最大大小为 4K
    private static final int DATA_LEN = 4096;
    //定义该客户端使用的 DatagramSocket
    private DatagramSocket socket = null;
    //定义接收网络数据的字节数组
    byte[] inBuff = new byte[DATA_LEN];
```

```
    //以指定字节数组创建准备接受数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
        new DatagramPacket(inBuff , inBuff.length);
    //定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket outPacket = null;
    public void init()throws IOException{
        try
        {
            //创建一个客户端 DatagramSocket，使用随机端口
            socket = new DatagramSocket();
            //初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
            outPacket = new DatagramPacket(new byte[0] , 0 ,
                InetAddress.getByName(DEST_IP) , DEST_PORT);
            //创建键盘输入流
            Scanner scan = new Scanner(System.in);
            //不断读取键盘输入
            while(scan.hasNextLine())
            {
                //将键盘输入的一行字符串转换字节数组
                byte[] buff = scan.nextLine().getBytes();
                //设置发送用的 DatagramPacket 里的字节数据
                outPacket.setData(buff);
                //发送数据报
                socket.send(outPacket);
                //读取 Socket 中的数据，读到的数据放在 inPacket 所封装的字节数组里。
                socket.receive(inPacket);
                System.out.println(new String(inBuff , 0 ,
                    inPacket.getLength()));
            }
        }
        //使用 finally 块保证关闭资源
        finally
        {
            if (socket != null)
            {
                socket.close();
            }
        }
    }
    public static void main(String[] args)
        throws IOException
    {
        new UdpClient().init();
    }
}
```

上述代码通过 DatagramSocket 实现了发送并接收 DatagramPacket 的功能，具体实现与服

务器的实现代码基本相似。而客户端与服务端的唯一区别是服务器所在 IP 地址和端口是固定的，所以客户端可以直接将该数据报发送给服务器，而服务器需要根据接收到的数据报决定“反馈”数据报的目的地。

5.3.2 使用 MulticastSocket

DatagramSocket 只允许将数据报发送给指定的目标地址，而 MulticastSocket 可以将数据报以广播的方式发送到数量不等的多个客户端。如果要使用多点广播，需要让一个数据报标有一组目标主机地址，当发出数据报后，整个组的所有主机都能收到该数据报。IP 多点广播（或多点发送）实现可以将单一信息发送到多个接收者，功能是设置一组特殊网络地址作为多点广播地址，每一个多点广播地址都被看做一个组，当客户端需要发送、接收广播信息时，只需加入到该组即可。

IP 为多点广播提供了这批特殊的 IP 地址，这些 IP 地址的范围是 224.0.0.0 至 235.255.255.255。

MulticastSocket 类既可以将数据报发送到多点广播地址，也可以接收其他主机的广播信息。MulticastSocket 类是 DatagramSocket 类的一个子类，当要发送一个数据报时，可使用随机端口创建 MulticastSocket，也可以在指定端口来创建 MulticastSocket。

在类 MulticastSocket 中提供了如下三个构造器。

- public MulticastSocket()：使用本机默认地址、随机端口来创建一个 MulticastSocket 对象。
- public MulticastSocket(int portNumber)：使用本机默认地址、指定端口来创建一个 MulticastSocket 对象。
- public MulticastSocket(SocketAddress bindaddr)：使用本机指定 IP 地址、指定端口来创建一个 MulticastSocket 对象。

在创建一个 MulticastSocket 对象后，需要将该 MulticastSocket 加入到指定的多点广播地址。在 MulticastSocket 中使用 jionGroup()方法加入到一个指定的组，使用 leaveGroup()方法从一个组中脱离出去。这两个方法的具体说明如下。

- joinGroup(InetAddress multicastAddr)：将该 MulticastSocket 加入到指定的多点广播地址。
- leaveGroup(InetAddress multicastAddr)：使该 MulticastSocket 离开指定的多点广播地址。

在某些系统中可能有多个网络接口，这可能会对多点广播带来问题，此时程序需要在一个指定的网络接口上监听，通过调用 setInterface 可选择 MulticastSocket 所使用的网络接口，也可以使用 getInterface 方法查询 MulticastSocket 监听的网络接口。

如果创建只发送数据报的 MulticastSocket 对象，则只需使用默认地址和随机端口即可。如果创建接收用的 MulticastSocket 对象，则该 MulticastSocket 对象必须具有指定端口，否则发送方无法确定发送数据报的目标端口。

虽然 MulticastSocket 实现发送/接收数据报的方法与 DatagramSocket 的完全一样，但是 MulticastSocket 比 DatagramSocket 多了下面的方法。

```
setTimeToLive(int ttl)
```

参数“ttl”设置数据报最多可以跨过多少个网络，具体说明如下。

- 为 0 时：指定数据报应停留在本地主机。
- 为 1 时：指定数据报发送到本地局域网。
- 为 32 时：只能发送到本站点的网络上。
- 为 64 时：数据报应保留在本地区。
- 为 128 时：数据报应保留在本大洲。
- 为 255 时：数据报可发送到所有地方。
- 为 1 时：是默认值。

在使用 MulticastSocket 实现多点广播时，所有通信实体都是平等的，都将自己的数据报发送到多点广播 IP 地址，并使用 MulticastSocket 接收其他人发送的广播数据报。例如在下面的代码中，使用 MulticastSocket 实现了一个基于广播的多人聊天室，程序只需要一个 MulticastSocket，两个线程，其中 MulticastSocket 既用于发送，也用于接收，其中一个线程分别负责接受用户键盘输入，并向 MulticastSocket 发送数据，另一个线程则负责从 MulticastSocket 中读取数据。

源码路径：daima\5\tcpudp\src\manySocket.java

```
import java.awt.*;
import java.net.*;
import java.io.*;
import java.util.*;
//让该类实现 Runnable 接口，该类的实例可作为线程的 target
public class manySocket implements Runnable
{
    //使用常量作为本程序的多点广播 IP 地址
    private static final String IP
            = "230.0.0.1";
    //使用常量作为本程序的多点广播目的的端口
    public static final int PORT = 30000;
    //定义每个数据报的最大大小为 4KB
    private static final int LEN = 2048;
    //定义本程序的 MulticastSocket 实例
    private MulticastSocket socket = null;
    private InetAddress bAddress = null;
    private Scanner scan = null;
    //定义接收网络数据的字节数组
    byte[] inBuff = new byte[LEN];
    //以指定字节数组创建准备接受数据的 DatagramPacket 对象
    private DatagramPacket inPacket =
            new DatagramPacket(inBuff , inBuff.length);
    //定义一个用于发送的 DatagramPacket 对象
    private DatagramPacket oPacket = null;
    public void init()throws IOException
    {
            try
            {
```

```
            //创建用于发送、接收数据的 MulticastSocket 对象
            //因为该 MulticastSocket 对象需要接收，所以有指定端口
            socket = new MulticastSocket(PORT);
            bAddress = InetAddress.getByName(IP);
            //将该 socket 加入指定的多点广播地址
            socket.joinGroup(bAddress);
            //设置本 MulticastSocket 发送的数据报被回送到自身
            socket.setLoopbackMode(false);
            //初始化发送用的 DatagramSocket，它包含一个长度为 0 的字节数组
            oPacket = new DatagramPacket(new byte[0] , 0 ,
                  bAddress , PORT);
            //启动以本实例的 run()方法作为线程体的线程
            new Thread(this).start();
            //创建键盘输入流
            scan = new Scanner(System.in);
            //不断读取键盘输入
            while(scan.hasNextLine())
            {
                  //将键盘输入的一行字符串转换字节数组
                  byte[] buff = scan.nextLine().getBytes();
                  //设置发送用的 DatagramPacket 里的字节数据
                  oPacket.setData(buff);
                  //发送数据报
                  socket.send(oPacket);
            }
      }
      finally
      {
            socket.close();
      }
}

public void run()
{
      try
      {
            while(true)
            {
                  //读取 Socket 中的数据，读到的数据放在 inPacket 所封装的字节数组里。
                  socket.receive(inPacket);
                  //打印输出从 socket 中读取的内容
                  System.out.println("聊天信息：" + new String(inBuff , 0 ,
                        inPacket.getLength()));
            }
      }
      //捕捉异常
      catch (IOException ex)
```

```
        {
            ex.printStackTrace();
            try
            {
                if (socket != null)
                {
                    //让该 Socket 离开该多点 IP 广播地址
                    socket.leaveGroup(bAddress);
                    //关闭该 Socket 对象
                    socket.close();
                }
                System.exit(1);
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args)
        throws IOException
    {
        new manySocket().init();
    }
}
```

上述代码的实现流程如下。

- 在 init()方法中创建一个 MulticastSocket 对象，因为需要使用该对象接收数据报，所以为此 Socket 对象设置使用固定端口。
- 将该 Socket 对象添加到指定的多点广播 IP 地址。
- 设置该 Socket 发送的数据报回送到自身，即该 Socket 可以接收到自己发送的数据报。
- 使用 MulticastSocket 发送并接收数据报，与使用 DatagramSocket 并没有区别。

5.4　在 Android 中使用 Socket 实现数据传输

通过本章前面内容的学习，已经了解了在 Java 应用中 Socket 网络编程的基本知识。在 Android 平台中，可以使用相同的方法用 Socket 实现数据传输功能。在本节的内容中，将通过一个具体实例的实现过程，来讲解在 Android 中使用 Socket 实现数据传输的基本方法。

题　　目	目　　的	源 码 路 径
实例 5-1	使用 Socket 实现数据传输	daima\5\socket

本实例的具体实现流程如下。

（1）首先实现服务器端，使用 Eclipse 新建一个名为“android_server”的 Java 工程，然后

编写服务器端的实现文件 AndroidServer.java，功能是创建 Socket 对象 client 以接受客户端请求，并创建 BufferedReader 对象 in 向服务器发送消息。文件 AndroidServer.java 的具体实现代码如下。

```
public class AndroidServer implements Runnable{
    public void run() {
        try {
            ServerSocket serverSocket=new ServerSocket(54321);
            while(true)
            {
                System.out.println("等待接收用户连接：");
                //接受客户端请求
                Socket client=serverSocket.accept();
                try
                {
                    //接受客户端信息
                    BufferedReader in=new BufferedReader(new InputStreamReader(client.
getInputStream()));
                    String str=in.readLine();
                    System.out.println("read:    "+str);
                    //向服务器发送消息
                    PrintWriter out=new PrintWriter(new BufferedWriter(new OutputStreamWriter
(client.getOutputStream())),true);
                    out.println("return        "+str);
                    in.close();
                    out.close();
                }catch(Exception ex)
                {
                    System.out.println(ex.getMessage());
                    ex.printStackTrace();
                }
                finally
                {
                    client.close();
                    System.out.println("close");
                }
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
    public static void main(String [] args)
    {
        Thread desktopServerThread=new Thread(new AndroidServer());
        desktopServerThread.start();
    }
```

```
}
```

（2）开始实现客户端的测试程序，使用 Eclipse 新建一个名为“testSocket”的 Android 工程，编写布局文件 main.xml，在主界面中插入一个信息输入文本框和一个“发送”按钮。文件 main.xml 的具体实现代码如下。

```
<?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">
    <EditText android:id="@+id/edit" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/but1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="发送" />
    <TextView android:id="@+id/text1" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/hello" />
</LinearLayout>
```

（3）编写测试文件 TestSocket.java，功能是获取输入框的文本信息，并将信息发送到“192.168.2.113”。文件 TestSocket.java 的具体实现代码如下。

```
//客户端的实现
public class TestSocket extends Activity {
    private TextView text1;
    private Button but1;
    private EditText edit1;
    private final String DEBUG_TAG="mySocketAct";
      public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
          text1=(TextView)findViewById(R.id.text1);
          but1=(Button)findViewById(R.id.but1);
          edit1=(EditText)findViewById(R.id.edit);
          but1.setOnClickListener(new Button.OnClickListener()
          {
              @Override
              public void onClick(View v) {
                  Socket socket=null;
                  String mesg=edit1.getText().toString()+"\r\n";
                  edit1.setText("");
                  Log.e("dddd", "sent id");

                  try {
                      socket=new Socket("192.168.2.113",54321);
                      //向服务器发送信息
                      PrintWriter out=new PrintWriter(new BufferedWriter(new OutputStreamWriter
(socket.getOutputStream())),true);
```

```
                    out.println(mesg);

                    //接受服务器的信息
                    BufferedReader br=new BufferedReader(new InputStreamReader(socket.
getInputStream()));
                    String mstr=br.readLine();
                    if(mstr!=null)
                    {
                        text1.setText(mstr);
                    }else
                    {
                        text1.setText("数据错误");
                    }
                    out.close();
                    br.close();
                    socket.close();
                } catch (UnknownHostException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }catch(Exception e)
                {
                    Log.e(DEBUG_TAG,e.toString());
                }
            }
        });
    }
}
```

（4）在文件 AndroidManifest.xml 中添加访问网络的权限，具体代码如下。

```
<!-- 添加可以通迅协议 -->
<uses-permission android:name="android.permission.INTERNET" />
```

到此为止，整个实例介绍完毕，执行后的效果如图 5-2 所示。

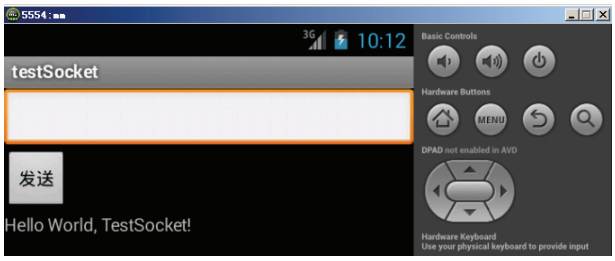


图 5-2　执行效果

第 6 章　处理 XML 数据

可扩展标记语言（Extensible Markup Language，XML）与 HTML 一样，都是标准通用标记语言（Standard Generalized Markup Language，SGML）。通过使用 XML 技术可以实现对数据的存储。在本章的内容中，将详细讲解在 Android 系统中处理 XML 数据的基本知识，为读者步入本书后面知识的学习打下基础。

6.1　XML 技术基础

XML 是Internet环境中跨平台的，依赖于内容的技术，是当前处理结构化文档信息的有力工具。XML 是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用简便的方式建立，虽然 XML 占用的空间要比二进制数据多，但是 XML 极其简单易于掌握和使用。在本节的内容中，将简要介绍 XML 技术的基本知识。

6.1.1　XML 概述

XML 与Access、Oracle和SQL Server等数据库不同，数据库提供了更强有力的数据存储和分析能力，例如数据索引、排序、查找、相关一致性等，XML 仅仅是展示数据。XML 与其他数据表现形式最大的不同是它极其简单。

XML 的简单使其易于在任何应用程序中读写数据，这使 XML 很快成为数据交换的唯一公共语言，虽然不同的应用软件也支持其他的数据交换格式，但不久之后它们都将支持 XML，那就意味着程序可以更容易的与 Windows、Mac OS、Linux 以及其他平台下产生的信息结合，然后可以很容易加载 XML 数据到程序中并解析，并以 XML 格式输出结果。

为了使 SGML 用户界面友好，XML 重新定义了 SGML 的一些内部值和参数，去掉了大量很少用到的功能。XML 保留了 SGML 的结构化功能，使网站设计者可以自定义文档类型，XML 同时也推出一种新型文档类型，使开发者也可以不必定义文档类型。

因为 XML 是W3C制定的，XML 的标准化工作由 W3C 的 XML 工作组负责，该小组成员由来自各个地方和行业的专家组成。因为 XML 是一个公共格式，所以无须担心 XML 技术会成为少数公司的盈利工具，XML 不是一种依附于特定浏览器的语言。

6.1.2　XML 的语法

XML 用来存储数据，对 HTML 语言进行扩展，它和 HTML 分工很明确，XML 是用来存储数据，而 HTML 是用来如何表现数据的，下面通过一段程序代码进行讲解，其代码(6-2.xml)如下。

```
<?xml version="1.0" encoding="utf-8"?>
<book>
<person>
<first>Kiran</first>
<last>Pai</last>
<age>22</age>
</person>
<person>
<first>Bill</first>
<last>Gates</last>
<age>46</age>
</person>
<person>
<first>Steve</first>
<last>Jobs</last>
<age>40</age>
</person>
</book>
```

上面的代码只要符合语法还可以写成汉语，如下面（6-3.xml）代码。

```
<?xml version="1.0" encoding="utf-8"?>
    <项目>
        <名>天上星</名>
        <电子邮件>tianshangxing@hotmail.com</电子邮件>
        <住宅>何国何市何区何街道何番号</住宅>
        <电话>86-021-742745674</电话>
        <一言>XML 学习</一言>
    </项目>
```

从上面两段代码可以看出，XML 的标记方式完全可以自由定义，不受约束，它只是用来存储信息。除了第一行固定以外，其他的只需前后标签一致，末标签不能省略， XML 语法格式总结如下。

- 在第一行必须对 XML 进行声明，也即声明 XML 的版本。
- 它的标记和 HTML 一样是成对出现。
- XML 对标记的大小写敏感。
- XML 标记是用户自定义的，但是每一个标记必须有结束标记。

6.1.3 获取 XML 文档

获取 XML 文档十分简单，下面通过一个简单的 Java 代码来获取 6.12 节中 6-2.xml 文件的信息，其代码如下。

```
import java.io.File;
import org.w3c.dom.Document;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
```

```
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
public class ReadAndPrintXMLFile{
public static void main (String argv []){
try {
    DocumentBuilderFactory docBuilderFactory
= DocumentBuilderFactory.newInstance();
                DocumentBuilder docBuilder
= docBuilderFactory.newDocumentBuilder();
                Document doc = docBuilder.parse (new File("6-2.xml"));
                doc.getDocumentElement ().normalize ();
                System.out.println ("Root element of the doc is "
 + doc.getDocumentElement().getNodeName());
                NodeList listOfPersons = doc.getElementsByTagName("person");
                int totalPersons = listOfPersons.getLength();
                System.out.println("Total no of people : " + totalPersons);
                for(int s=0; s<listOfPersons.getLength() ; s++){
                    Node firstPersonNode = listOfPersons.item(s);
                    if(firstPersonNode.getNodeType() == Node.ELEMENT_NODE){
                        Element firstPersonElement = (Element)firstPersonNode;
                       NodeList firstNameList =
 firstPersonElement.getElementsByTagName("first");
                        Element firstNameElement
= (Element)firstNameList.item(0);
                        NodeList textFNList = firstNameElement.getChildNodes();
                        System.out.println("First Name : " +
                                ((Node)textFNList.item(0)).getNodeValue().trim());
                      NodeList lastNameList
= firstPersonElement.getElementsByTagName("last");
                        Element lastNameElement = (Element)lastNameList.item(0);
                        NodeList textLNList = lastNameElement.getChildNodes();
                        System.out.println("Last Name : " +
                                 ((Node)textLNList.item(0)).getNodeValue().trim());
                        NodeList ageList
 = firstPersonElement.getElementsByTagName("age");
                        Element ageElement = (Element)ageList.item(0);
                        NodeList textAgeList = ageElement.getChildNodes();
                        System.out.println("Age : " +
 ((Node)textAgeList.item(0)).getNodeValue().trim());
                    }    }  }
            catch (SAXParseException err)

{
                    System.out.println ("** Parsing error" + ", line "
                                                + err.getLineNumber () + ", uri " + err.getSystemId ());
                    System.out.println(" " + err.getMessage ());     }
```

```
            catch (SAXException e) {
                        Exception x = e.getException ();
                        ((x == null) ? e : x).printStackTrace ();
            }
            catch (Throwable t) {
                        t.printStackTrace ();
  }
                }
}
```

用户在 Java API 还可以找到更多操作 XML 文档的方法，执行上述代码后得到如图 6-1 所示的结果。

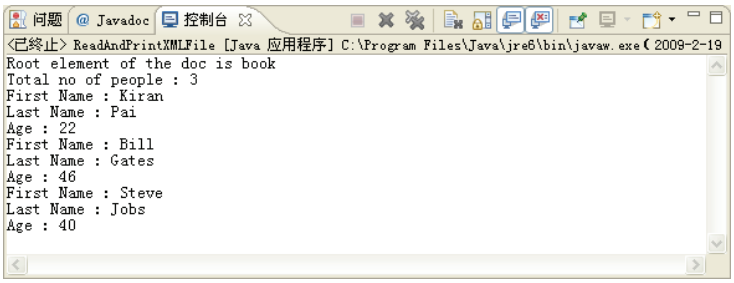


图 6-1　获取 XML 文档

注意：读者需要注意的是，XML 主要用来存储信息，不负责显示在页面。获取 XML 文档的方法有很多，也并不是只有 Java 语言，还有许多语言都可以调用，如 C#、PHP 和 ASP 等，也包括 HTML 语言。

6.2　使用 SAX 解析 XML 数据

SAX，全称 Simple API for XML，既是一种接口，也是一个软件包。SAX 最初是由 David Megginson 采用 Java 语言开发，之后 SAX 很快在 Java 开发者中流行起来。San 现在负责管理其原始 API 的开发工作，这是一种公开的、开源软件。不同于其他大多数 XML 标准的是，SAX 没有必须遵守的标准 SAX 参考版本。因此，SAX 的不同实现可能采用区别很大的接口。在本节的内容中，将简要介绍 SAX 技术的基本知识。

6.2.1　SAX 的原理

SAX 的工作原理简单地说就是对文档进行顺序扫描，当扫描到文档开始与结束、元素开始与结束、文档结束等地方时通知事件处理函数，由事件处理函数做出相应动作，然后继续同样的扫描，直至文档结束。

大多数 SAX 实现都会产生以下五种类型的事件。

- ❑ 在文档的开始和结束时触发文档处理事件。
- ❑ 在文档内每一个 XML 元素接受解析的前后触发元素事件。
- ❑ 任何元数据通常都由单独的事件交付。

- 在处理文档的 DTD 或 Schema 时产生 DTD 或 Schema 事件。
- 产生错误事件以通知主机应用程序解析错误。

6.2.2　基于对象和基于事件的接口

语法分析器有两类接口：基于对象接口和基于事件的接口。DOM 是基于对象的语法分析器的标准的 API。作为基于对象的接口，DOM 通过在内存中显示地构建对象树来与应用程序通信。对象树是 XML 文件中元素树的精确映射。

DOM 易于学习和使用，因为它与常用的 XML 文档关系密切，两者的语法十分相似。在现实应用中，DOM 的最常见应用便是操作 XML 文件中的数据，例如，浏览器程序和编译器程序。

然而，对于大多数应用程序，处理 XML 文档只是其众多任务中的一种。例如，记账软件包可能导入 XML 格式的发票数据，但这不是其主要用途。计算账户余额、跟踪支出以及使付款与发票匹配才是主要活动。DOM 模型不太适合记帐应用程序，因为在那种情况下，应用程序必须在内存中维护数据的两个副本（一个是 DOM 树，另一个是应用程序自己的结构）。在内存维护两次数据会使应用程序运行效率下降。对于桌面应用程序来说，这可能不是主要问题，但是它可能会导致服务器瘫痪。对于不以 XML 为中心的应用程序，SAX 是明智的选择。实际上，SAX 并不在内存中显式地构建文档树。它使应用程序能用最有效率的方法存储数据。

应用程序如何在 XML 树及其自身数据结构之间进行映射，如图 6-2 所示。

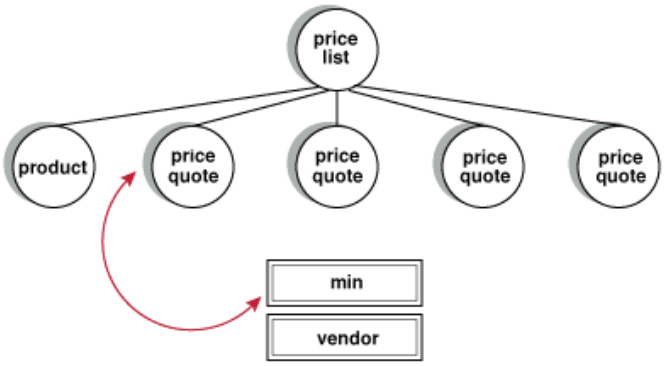


图 6-2　将 XML 结构映射成应用程序结构

SAX 是基于事件的接口，基于事件的语法分析器将事件发送给应用程序。这些事件类似于用户界面事件，例如，浏览器中的 ONCLICK 事件或者 Java 中的 AWT/Swing 事件。

事件通知应用程序发生了某件事并需要应用程序作出反应，在浏览器中，通常为响应用户操作而生成事件：当用户单击按钮时，按钮产生一个 ONCLICK 事件。

在 XML 语法分析器中，事件与用户操作无关，而与正在读取的 XML 文档中的元素有关。在 XML 语法分析器中可以处理以下方面的事件。

- 元素开始和结束标记。
- 元素内容。

- 实体。
- 语法分析错误。

语法分析器在读取文档时生成事件的过程，如图 6-3 所示。

图 6-3　语法分析器生成事件

这两种 API 中没有一种在本质上是最好的，它们适用于不同的需求。在需要更多控制时使用 SAX；要增加方便性时，则使用 DOM，例如，DOM 在脚本语言中很流行。

采用 SAX 的主要原因是效率。SAX 比 DOM 做的事要少，但提供了对语法分析器的更多控制。当然，如果语法分析器的工作减少，则意味着开发者有更多的工作要做。而且，正如我们已讨论的，SAX 比 DOM 消耗的资源要少，因为它不需要构建文档树。在 XML 早期，DOM 得益于 W3C 批准的官方 API 这一身份。逐渐地，开发者选择了功能性而放弃了方便性，并转向了 SAX。

SAX 的主要限制是它无法向后浏览文档。实际上，激发一个事件后，语法分析器就将其“忘记”。如我们所看到的，应用程序必须显式地缓冲其感兴趣的事件。

6.2.3　常用的接口和类

在现实开发应用中，SAX 将其事件分为如下的接口。

- ContentHandler：定义与文档本身关联的事件（例如，开始和结束标记）。大多数应用程序都注册这些事件。
- DTDHandler：定义与 DTD 关联的事件。然而，它不定义足够的事件来完整地报告 DTD。如果需要对 DTD 进行语法分析，请使用可选的 DeclHandler。DeclHandler 是 SAX 的扩展，并且不是所有的语法分析器都支持它。
- EntityResolver：定义与装入与实体关联的事件。只有少数几个应用程序注册这些事件。
- ErrorHandler：定义错误事件。许多应用程序注册这些事件以便用它们自己的方式报错。

为简化工作，SAX 在 DefaultHandler 类中提供了这些接口的默认实现。在大多数情况下，为应用程序扩展 DefaultHandler 并覆盖相关的方法要比直接实现一个接口更容易。

1．XMLReader

如果为事件注册处理器并启动语法分析器，应用程序应该使用 XMLReader 接口，实现方法是使用 XMLReader 的 parse()方法来启动，具体语法格式如下。

```
parser.parse(args[0]);
```

XMLReader 中的主要方法如下。

（1）parse()：对 XML 文档进行语法分析。parse()有两个版本，一个接受文件名或 URL，另一个接受 InputSource 对象。

（2）setContentHandler()、setDTDHandler()、setEntityResolver()和 setErrorHandler()：为应用程序注册事件处理器。

（3）setFeature()和 setProperty()：控制语法分析器如何工作。它们采用一个特性或功能标识（一个类似于名称空间的 URI 和值）。功能采用 Boolean 值，而特性则采用“对象”。

最常用的 XMLReaderFactory 功能如下。

（1）http:// xml.org/sax/features/namespaces：所有 SAX 语法分析器都能识别它。如果将它设置为 true（默认值），则在调用 ContentHandler 的方法时，语法分析器将识别出名称空间并解析前缀。

（2）http://xml.org/sax/features/validation：它是可选的。如果将它设置为 true，则验证语法分析器将验证该文档。非验证语法分析器忽略该功能。

2．XMLReaderFactory

XMLReaderFactory 用于创建语法分析器对象，它定义了 createXMLReader()的如下两个版本。

- 一个采用语法分析器的类名作为参数。
- 一个从 org.xml.sax.driver 系统特性中获得类名称。

对于 Xerces，类是 org.apache.xerces.parsers.SAXParser。应该使用 XMLReaderFactory，因为它易于切换至另一种 SAX 语法分析器。实际上，只需要更改下面一行然后重新编译即可。

```
XMLReaderparser=XMLReaderFactory.createXMLReader(
"org.apache.xerces.parsers.SAXParser");
```

为获得更大的灵活性，应用程序可以从命令行读取类名或使用不带参数的 createXMLReader()方法。因此可以不重新编译就可以更改语法分析器。

3．InputSource

InputSource 控制语法分析器如何读取文件，包括 XML 文档和实体。在大多数情况下，文档是从 URL 装入的。但是，有特殊需求的应用程序可以覆盖 InputSource。例如，可以用来从数据库中装入文档。

4．ContentHandler

ContentHandler 是最常用的 SAX 接口，因为它定义 XML 文档的事件。ContentHandler 声明以下几个事件。

（1）startDocument()/endDocument()：通知应用程序文档的开始或结束。

（2）startElement()/endElement()：通知应用程序标记的开始或结束。属性作为 Attributes 参数传递。即使只有一个标记，“空”元素（例如，<imghref="logo.gif"/>）也生成 startElement()和 endElement()。

（3）startPrefixMapping()/endPrefixMapping()：通知应用程序名称空间作用域。几乎不需要该信息，因为当 http://xml.org/sax/features/namespaces 为 true 时，语法分析器已经解析了名称空间。

（4）当语法分析器在元素中发现文本时，characters()/ignorableWhitespace()会通知应用程序。语法分析器负责将文本分配到几个事件（为了更好地管理其缓冲区）。ignorableWhitespace 事

件用于接收元素内容中可忽略空白的通知。

（5）processingInstruction()：将处理指令通知应用程序。

（6）skippedEntity()：通知应用程序已经跳过了一个实体。

（7）setDocumentLocator()：将 Locator 对象传递到应用程序。不需要 SAX 语法分析器提供 Locator，但是如果它提供了，则必须在任何其他事件之前激活该事件。

5．属性

在 startElement()事件中，应用程序在 Attributes 参数中接收属性列表。

```
Stringattribute=attributes.getValue("","price");
```

Attributes 定义了下列方法。

- getValue(i)/getValue(qName)/getValue(uri,localName)：返回第 i 个属性值或给定名称的属性值。
- getLength()：返回属性数目。
- getQName(i)/getLocalName(i)/getURI(i)：返回限定名（带前缀）、本地名（不带前缀）和第 i 个属性的名称空间 URI。
- getType(i)/getType(qName)/getType(uri,localName)：返回第 i 个属性的类型或者给定名称的属性类型。类型为字符串，即在 DTD 中所使用的：“CDATA”、“ID”、“IDREF”、“IDREFS”、“NMTOKEN”、“NMTOKENS”、“ENTITY”、“ENTITIES” 或“NOTATION”。

注意：Attributes 参数仅在 startElement()事件期间可用。如果在事件之间需要它，则需用 AttributesImpl 复制一个。

6．定位器

Locator 为应用程序提供行和列的位置。不需要语法分析器来提供 Locator 对象。Locator 定义下列方法。

- getColumnNumber()：返回当前事件结束时所在的列。在 endElement()事件中，它将返回结束标记所在的最后一列。
- getLineNumber()：返回当前事件结束时所在的行。在 endElement()事件中，它将返回结束标记所在的行。
- getPublicId()：返回当前文档事件的公共标识。
- getSystemId()：返回当前文档事件的系统标识。

7．DTDHandler

DTDHandler 声明两个与 DTD 语法分析器相关的事件。具体如下。

- notationDecl()：通知应用程序已经声明了一个标记。
- nparsedEntityDecl()：通知应用程序已经发现了一个未经过语法分析的实体声明。

8．EntityResolver

EntityResolver 接口仅定义一个事件 resolveEntity()，它返回 InputSource。因为 SAX 语法分析器已经可以解析大多数 URL，所以应用程序很少实现 EntityResolver。例外情况是对于目录文件它将公共标识解析成系统标识。如果在应用程序中需要目录文件，请下载 NormanWalsh 的目录软件包。

9．ErrorHandler

ErrorHandler 接口定义错误事件。处理这些事件的应用程序可以提供定制错误处理。安装了定制错误处理器后，语法分析器不再抛出异常。抛出异常是事件处理器的责任。接口定义了与错误的三个级别或严重性对应的三个方法。

- warning()：警示不是由 XML 规范定义的错误。例如，当没有 XML 声明时，某些语法分析器发出警告，它不是错误，但是它可能值得注意。
- error()：警示由 XML 规范定义的错误。
- fatalError()：警示由 XML 规范定义的致命错误。

10．SAXException

SAX 定义的大多数方法都可以抛出 SAXException。当对 XML 文档进行语法分析时，SAXException 会抛出一个错误，这里的错误可以是语法分析错误也可以是事件处理器中的错误。要报告来自事件处理器的其他异常，可以将异常封装在 SAXException 中。

6.2.4　实战演练——在 Android 系统中使用 SAX 解析 XML 数据

Android 中可以使用标准的 XML 生成器、解析器、转换器 API，对 XML 进行解析和转换。本实例的功能是，在 Android 系统中使用 SAX 技术解析并生成 XML。

题　　目	目　　的	源 码 路 径
实例 6-1	在 Android 系统中解析和生成 XML	daima\6\XML_Parser

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，具体实现代码如下。

```
<?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>
```

（2）编写解析功能的核心文件 SAXForHandler.java，主要实现代码如下。

```
public class SAXForHandler extends DefaultHandler {
    private static final String TAG = "SAXForHandler";
    private List<Person> persons;
    private String perTag ;//通过此变量，记录前一个标签的名称。
    Person person;//记录当前 Person
```

```
    public List<Person> getPersons() {
        return persons;
    }
    //适合在此事件中触发初始化行为。
    public void startDocument() throws SAXException {
        persons = new ArrayList<Person>();
        Log.i(TAG , "***startDocument()***");
    }
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        if("person".equals(localName)){
            for ( int i = 0; i < attributes.getLength(); i++ ) {
                Log.i(TAG ,"attributeName:" + attributes.getLocalName(i)
                        + "_attribute_Value:" + attributes.getValue(i));
                person = new Person();
                person.setId(Integer.valueOf(attributes.getValue(i)));
            }
        }
        perTag = localName;
        Log.i(TAG , qName+"***startElement()***");
    }

    public void characters(char[] ch, int start, int length) throws SAXException {
        String data = new String(ch, start, length).trim();
       if(!"".equals(data.trim())){
            Log.i(TAG ,"content: " + data.trim());
       }
       if("name".equals(perTag)){
                person.setName(data);
        }else if("age".equals(perTag)){
                person.setAge(new Short(data));
        }
    }
    public void endElement(String uri, String localName, String qName)
            throws SAXException {
        Log.i(TAG , qName+"***endElement()***");
        if("person".equals(localName)){
            persons.add(person);
            person = null;
        }
        perTag = null;
    }
    public void endDocument() throws SAXException {
        Log.i(TAG , "***endDocument()***");
    }
}
```

（3）编写单元测试文件 PersonServiceTest.java，具体代码如下。

```
public void testSAXGetPersons() throws Throwable{
    InputStream inputStream = this.getClass().getClassLoader().
                getResourceAsStream("wang.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());
    }
}
```

此时使用 Eclipse 启动 Android 模拟器，执行后的效果如图 6-4 所示。

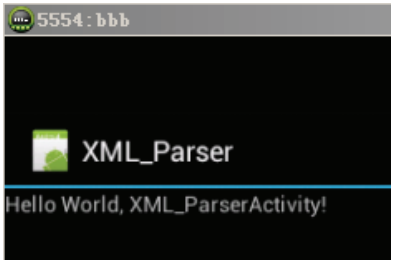


图 6-4　执行效果

（4）开始具体测试，在 Eclipse 中导入本实例项目，在“Outline”面板中右键单击 testSAXGetPersons()，如图 6-5 所示。在弹出命令中依次选择 Run As→Android JUnit Test 命令，如图 6-6 所示。

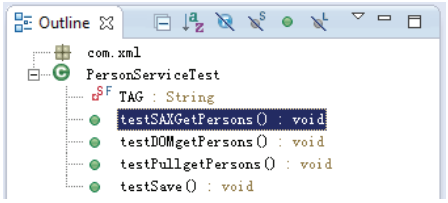


图 6-5　右键单击 testDOMgetPersons()

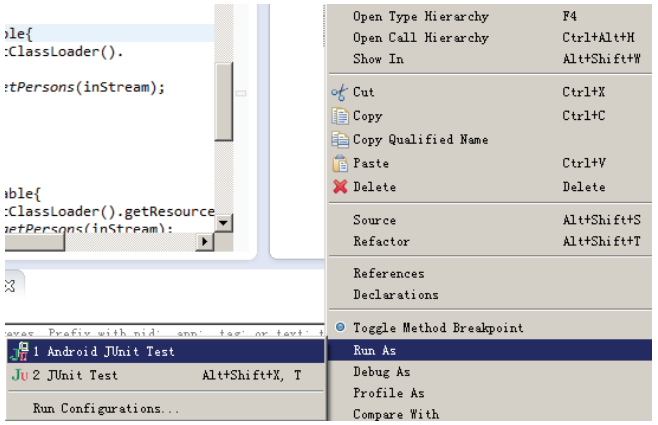


图 6-6　选择 Android JUnit Test 子菜单

此时将在 LogCat 中显示测试的解析结果，如图 6-7 所示。

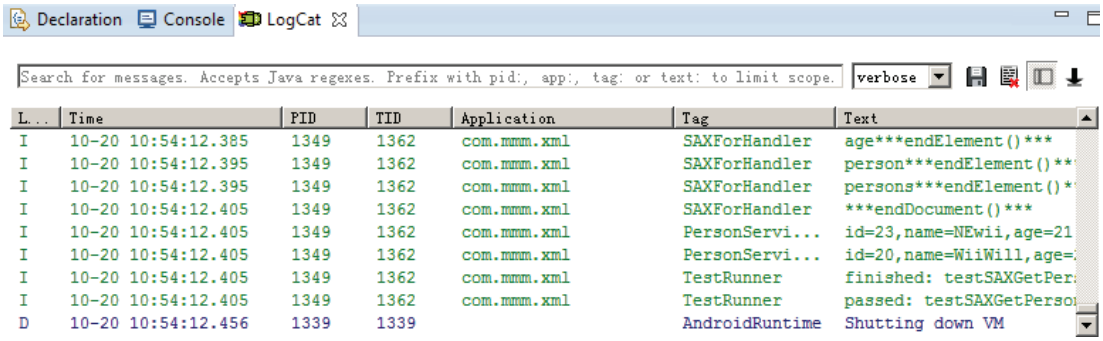


图 6-7　解析结果

注意：如果 Android 下的 Eclipse 界面中没有 LogCat 面板，只需依次单击 Eclipse 菜单栏中的 Window→show view→other→Android 命令，然后选择 LogCat 后即可在 Eclipse 界面看到 LogCat 面板。

6.3　使用 DOM 解析 XML

文件对象模型（Document Object Model，DOM），是 W3C 组织推荐的处理可扩展置标语言的标准编程接口。在本节的内容中，将详细讲解在 Android 系统中使用 DOM 解析 XML 的基本知识。

6.3.1　DOM 概述

DOM 可以以一种独立于平台和语言的方式访问和修改一个文档的内容和结构。换句话说，这是表示和处理一个 HTML 或 XML 文档的常用方法。有一点很重要，DOM 的设计是以对象管理组织（OMG）的规约为基础的，因此可以用于任何编程语言。最初人们把它认为是一种让 JavaScript 在浏览器间可移植的方法，不过 DOM 的应用已经远远超出这个范围。DOM 技术使得用户页面可以动态地变化，如可以动态地显示或隐藏一个元素，改变它们的属性，增加一个元素等，DOM 技术使得页面的交互性大大地增强。

DOM 实际上是以面向对象方式描述的文档模型。DOM 定义了表示和修改文档所需的对象、这些对象的行为和属性以及这些对象之间的关系。可以把 DOM 认为是页面上数据和结构的一个树形表示，不过页面当然可能并不是以这种树的方式具体实现。

通过 JavaScript 可以重构整个 HTML 文档，可以添加、移除、改变或重排页面上的项目。要想改变页面，JavaScript 需要获得对 HTML 文档中所有元素进行访问的入口。这个入口，连同对 HTML 元素进行添加、移动、改变或移除的方法和属性，都是通过 DOM 来获得的。

6.3.2　DOM 的结构

根据 W3C DOM 规范，DOM 是 HTML 与 XML 的 API，DOM 将整个页面映射为一个由层次节点组成的文件。有一级、二级、三级共三个级别，各个级别的具体说明如下。

1．一级 DOM

一级 DOM 在 1998 年 10 月份成为 W3C 的提议内容，由 DOM 核心与 DOM HTML 两个模块组成。DOM 核心能映射以 XML 为基础的文档结构，允许获取和操作文档的任意部分。DOM HTML 通过添加 HTML 专用的对象与函数对 DOM 核心进行了扩展。

2．二级 DOM

鉴于一级 DOM 仅以映射文档结构为目标，二级 DOM 则更为宽广。通过对原有 DOM 的扩展，二级 DOM 通过对象接口增加了对鼠标和用户界面事件（DHTML 长期支持鼠标与用户界面事件）、范围、遍历（重复执行 DOM 文档）和层叠样式表（CSS）的支持。同时也对一级 DOM 的核心进行了扩展，从而可支持 XML 命名空间。

在二级 DOM 中，引进了如下所示的新 DOM 模块来处理新的接口类型。

- DOM 视图：描述跟踪一个文档的各种视图（使用 CSS 样式设计文档前后）的接口。
- DOM 事件：描述事件接口。
- DOM 样式：描述处理基于 CSS 样式的接口。
- DOM 遍历与范围：描述遍历和操作文档树的接口。

3．三级 DOM

三级 DOM 通过引入统一方式载入和保存文档和文档验证方法对 DOM 进行进一步扩展，三级 DOM 包含一个名为“DOM 载入与保存”的新模块，DOM 核心扩展后可支持 XML1.0 的所有内容，包扩 XML Infoset、XPath、和 XML Base。

4．“0 级”DOM

当阅读与 DOM 有关的材料时，可能会遇到“参考 0 级 DOM”的情况。在此需要注意的是，并没有标准被称为 0 级 DOM，它仅是 DOM 历史上的一个参考点。0 级 DOM 被认为是在 Internet Explorer 4.0 与 Netscape Navigator4.0 上被支持的最早的 DHTML。

5．节点

根据 DOM，HTML 文档中的每个成分都是一个节点。关于使用节点的具体规则，DOM 是这样规定的。

- 整个文档是一个文档节点。
- 每个 HTML 标签是一个元素节点。
- 包含在 HTML 元素中的文本是文本节点。
- 每一个 HTML 属性是一个属性节点。
- 注释属于注释节点。

6．Node 的层次

在 DOM 中，各个节点之间彼此都有着等级关系。HTML 文档中的所有节点组成了一个文档树（或节点树）。HTML 文档中的每个元素、属性、文本等都代表着树中的一个节点。树起始于文档节点，并由此继续延伸，直到处于这棵树最低级别的所有文本节点为止。例如图 6-8 演示了一个文档树（节点树）的结构。

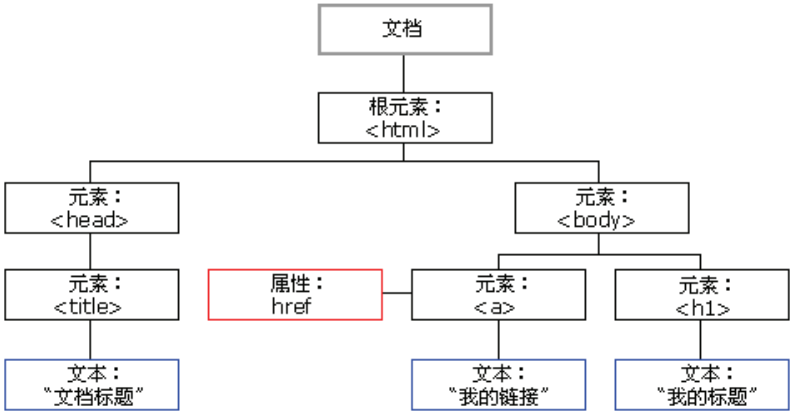


图 6-8　一个文档树（节点树）的结构

7．文档树（节点树）

请读者看如下的 HTML 文档。

```
<html>
<head>
<title>DOM Tutorial</title>
</head>
<body>
<h1>DOM Lesson one</h1>
<p>Hello world!</p>
</body>
</html>
```

在上述代码中，所有的节点彼此间都存在关系，具体说明如下。

- 除文档节点之外的每个节点都有父节点。例如<head> 和 <body> 的父节点是 <html> 节点，文本节点“Hello world!”的父节点是 <p> 节点。
- 大部分元素节点都有子节点。例如<head> 节点有一个子节点：<title>。<title> 节点也有一个子节点：文本节点“DOM Tutorial”。
- 当节点有同一个父节点时，它们就是同辈（同级节点）。例如<h1> 和 <p>是同辈，因为它们的父节点均是 <body> 节点。
- 节点也可以拥有后代，后代指某个节点的所有子节点，或者这些子节点的子节点，以此类推。例如所有的文本节点都是 <html>节点的后代，而第一个文本节点是 <head> 节点的后代。
- 节点也可以拥有先辈。先辈是某个节点的父节点，或者父节点的父节点，以此类推。例如所有的文本节点都可把 <html> 节点作为先辈节点。

6.3.3　实战演练——在 Android 系统中使用 DOM 解析 XML 数据

本实例的功能是，在 Android 系统中使用 DOM 技术来解析并生成 XML。

题　　目	目　　的	源 码 路 径
实例 6-2	在 Android 系统中解析和生成 XML	daima\6\XML_Parser

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，具体实现代码如下。

```
<?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>
```

（2）编写解析功能的核心文件 DOMPersonService.java，具体实现流程如下。

- 创建 DocumentBuilderFactory 对象 factory，并调用 newInstance()创建新实例。
- 创建 DocumentBuilder 对象 builder，DocumentBuilder 将实现具体的解析工作以创建 Document 对象。
- 解析目标 XML 文件以创建 Document 对象。

文件 DOMPersonService.java 的具体实现代码如下。

```
public class DOMPersonService {
    public static List<Person> getPersons(InputStream inStream) throws Exception{
        List<Person> persons = new ArrayList<Person>();
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(inStream);
        Element root = document.getDocumentElement();
        NodeList personNodes = root.getElementsByTagName("person");
        for(int i=0; i < personNodes.getLength() ; i++){
            Element personElement = (Element)personNodes.item(i);
            int id = new Integer(personElement.getAttribute("id"));
            Person person = new Person();
            person.setId(id);
            NodeList childNodes = personElement.getChildNodes();
            for(int y=0; y < childNodes.getLength() ; y++){
                if(childNodes.item(y).getNodeType()==Node.ELEMENT_NODE){
                    if("name".equals(childNodes.item(y).getNodeName())){
                        String name = childNodes.item(y).getFirstChild().getNodeValue();
                        person.setName(name);
                    }else if("age".equals(childNodes.item(y).getNodeName())){
                        String age = childNodes.item(y).getFirstChild().getNodeValue();
                        person.setAge(new Short(age));
                    }
                }
```

```
                }
                persons.add(person);
            }
            inStream.close();
            return persons;
        }
    }
```

（3）编写单元测试文件 PersonServiceTest.java，具体代码如下。

```
public void testDOMgetPersons() throws Throwable{
    InputStream inStream = this.getClass().getClassLoader().
                getResourceAsStream("wang.xml");
    List<Person> persons = DOMPersonService.getPersons(inStream);
    for(Person person : persons){
        Log.i(TAG, person.toString());
    }
}
```

（4）开始具体测试，在 Eclipse 中导入本实例项目，在 Outline 面板中右键单击 testDOMgetPersons() : void，如图 6-9 所示。在弹出命令中依次选择 Run As→Android JUnit Test 命令，如图 6-10 所示。

此时将在 Logcat 中显示测试的解析结果，如图 6-11 所示。

注意：SAX 和 DOM 的对比

DOM 解析器，是通过将 XML 文档解析成树形模型并将其放入内存来完成解析工作的，然后对文档的操作都是在这个树形模型上完成的。这个在内存中的文档树将是文档实际大小的几倍。这样做的好处是结构清晰、操作方便，而带来的麻烦就是极其耗费系统资源。

SAX 解析器，正好克服了 DOM 的缺点，分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中，这对于大型文档来说是个巨大的优点。事实上，应用程序甚至不必解析整个文档，它可以在某个条件得到满足时停止解析。

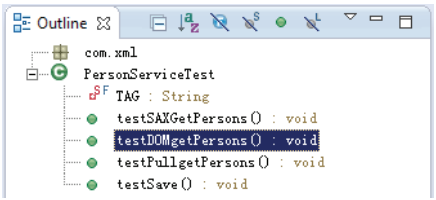


图 6-9　右键单击 testDOMgetPersons()

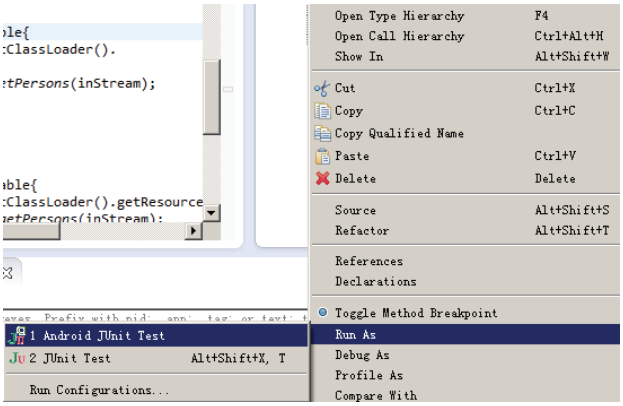


图 6-10　选择 Android JUnit Test 子菜单

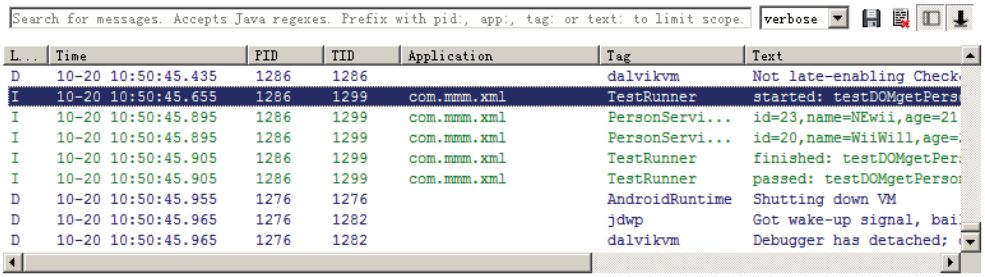


图 6-11　解析结果

表 6-1 列出了 SAX 和 DOM 在一些方面的对比。

表 6-1　SAX 和 DOM 的对比

SAX	DOM
顺序读入文档并产生相应事件，可以处理任何大小的 XML 文档	在内存中创建文档树，不适于处理大型 XML 文档
只能对文档按顺序解析一遍，不支持对文档的随意访问	可以随意访问文档树的任何部分，没有次数限制
只能读取 XML 文档内容，而不能修改	可以随意修改文档树，从而修改 XML 文档
开发上比较复杂，需要自己来实现事件处理器	易于理解，易于开发
对开发人员而言更灵活，可以用 SAX 创建自己的 XML 对象模型	已经在 DOM 基础之上创建好了文档树

6.4　PULL 解析技术

在 Android 网络开发应用中，除了可以使用 SAX 和 DOM 技术解析 XML 文件外，还可以使用 Android 系统内置的 PULL 解析器解析 XML 文件。在本节的内容中，将详细讲解使用 PULL 技术解析 XML 文件的具体过程。

6.4.1　PULL 解析原理

PULL 解析器的运行方式与 SAX 解析器相似，也提供了类似的功能事件，例如开始元素和结束元素事件，使用 parser.next()可以进入下一个元素并触发相应事件。事件将作为数值代码被发送，因此可以使用一个 Switch 控件对感兴趣的事件进行处理。当元素开始解析时，调用 parser.nextText()方法可以获取下一个 Text 类型元素的值。

PULL 解析器的源码及文档下载网址：http://www.xmlpull.org/

在解析过程中，PULL 是采用事件驱动方式进行解析的，当 PULL 解析器开始解析之后，可以调用它的 next()方法来获取下一个解析事件（就是开始文档，结束文档，开始标签，结束标签），当解析到某个元素时可以调用 XmlPullParser 的 getAttributte()方法来获取属性的值，也可调用它的 nextText()方法获取本节点的值。

6.4.2　实战演练——在 Android 系统中使用 PULL 解析 XML 数据

本实例的功能是，在 Android 系统中使用 PULL 技术来解析并生成 XML。

题　　目	目　　的	源 码 路 径
实例 6-3	在 Android 系统中使用 PULL 解析 XML 文件	daima\6\XML_Parser

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，具体实现代码如下。

```
<?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>
```

（2）编写解析功能的核心文件 PullPersonService.java，具体实现流程如下。

- 创建 DocumentBuilderFactory 对象 factory，并调用 newInstance()创建新实例。
- 创建 DocumentBuilder 对象 builder，DocumentBuilder 将实现具体的解析工作以创建 Document 对象。
- 解析目标 XML 文件以创建 Document 对象。

文件 PullPersonService.java 的具体实现代码如下。

```
public class PullPersonService {
    public static void save(List<Person> persons, OutputStream outStream) throws Exception{
        XmlSerializer serializer = Xml.newSerializer();
        serializer.setOutput(outStream, "UTF-8");
        serializer.startDocument("UTF-8", true);
        serializer.startTag(null, "persons");
        for(Person person : persons){
            serializer.startTag(null, "person");
            serializer.attribute(null, "id", person.getId().toString());
            serializer.startTag(null, "name");
            serializer.text(person.getName());
            serializer.endTag(null, "name");
            serializer.startTag(null, "age");
            serializer.text(person.getAge().toString());
            serializer.endTag(null, "age");

            serializer.endTag(null, "person");
        }
        serializer.endTag(null, "persons");
        serializer.endDocument();
```

```
        outStream.flush();
        outStream.close();
    }
    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;
    }
}
```

（3）编写单元测试文件 PersonServiceTest.java，具体代码如下。

```
public void testPullgetPersons() throws Throwable{
    InputStream inStream = this.getClass().getClassLoader().getResourceAsStream("wang.xml");
    List<Person> persons = PullPersonService.getPersons(inStream);
```

```
    for(Person person : persons){
        Log.i(TAG, person.toString());
    }
}
```

（4）开始具体测试，在 Eclipse 中导入本实例项目，在 Outline 面板中右键单击 testPullgetPersons()，如图 6-12 所示。在弹出命令中依次选择 Run As→Android JUnit Test 选项，如图 6-13 所示。

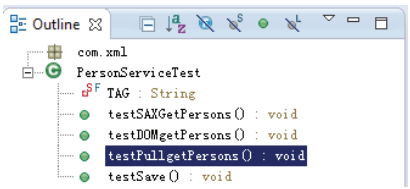


图 6-12　右键单击 testDOMgetPersons()

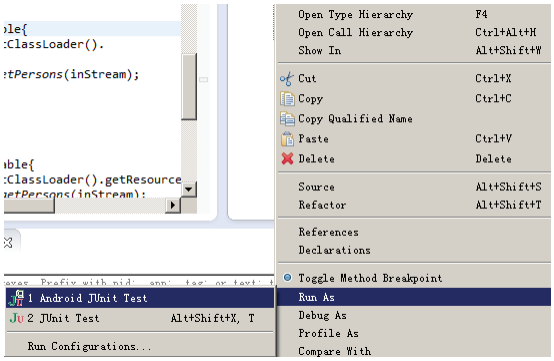


图 6-13　选择 Android JUnit Test 选项

此时将在 Logcat 中显示测试的解析结果，如图 6-14 所示。

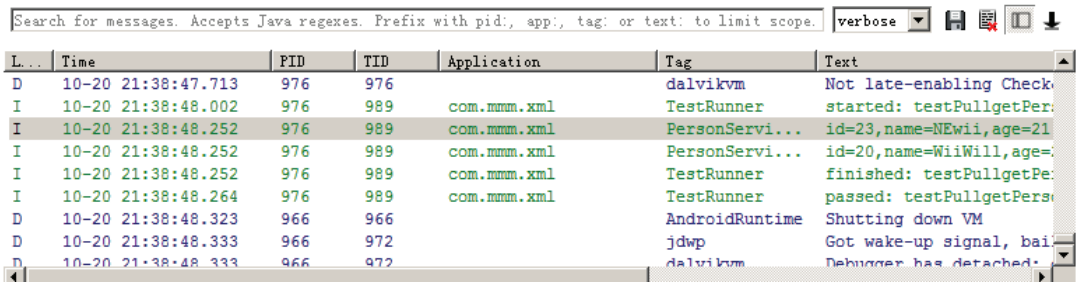


图 6-14　解析结果

6.5　实战演练——三种解析方式的综合演练

在本节的内容中，将通过一个具体实例的实现过程，来综合演示使用 SAX、DOM 和 PULL 技术解析 XML 数据的具体过程。

题　目	目　的	源 码 路 径
实例 6-4	综合演示使用 SAX、DOM 和 PULL 技术解析 XML 数据	daima\6\XML_Parser

本实例的具体实现流程如下。

（1）编写主界面的布局文件 main.xml，在主界面中插入三个 Button 按钮。具体实现代码

如下。

```
<?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/btnSAX"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="SAX 方式解析" />
    <Button
        android:id="@+id/btnPULL"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pull 方式解析" />
     <Button
        android:id="@+id/btnDOM"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Dom 方式解析" />
</LinearLayout>
```

执行后的主界面效果如图 6-15 所示。

图 6-15　执行效果

（2）编写列表界面文件 list.xml，功能是列表显示解析后的结果。具体实现代码如下。

```
<?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:id="@+id/textName"
        android:layout_width="fill_parent"
```

```
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/textId"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:visibility="gone"
            />
</LinearLayout>
```

（3）编写 XML 文件 channels.xml，本实例的目的就是使用 SAX、DOM 和 PULL 技术解析此 XML 文件中的数据。文件 channels.xml 的具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<channel>
<item id="0" url="http://www.baidu.com">百度</item>
<item id="1" url="http://www.qq.com">腾讯</item>
<item id="2" url="http://www.sina.com.cn">新浪</item>
<item id="3" url="http://www.taobao.com">淘宝</item>
</channel>
```

（4）编写对应的 XML 实体对象文件 channel.java，具体实现代码如下。

```
public class channel {
    private String id;
    private String url;
    private String name;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getUrl() {
        return url;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

（5）编写文件 XMLParserActivity.java，用于响应单击主界面三个按钮的事件处理程序。

具体实现代码如下。

```
public class XMLParserActivity extends Activity {
    private Button btnSax;
    private Button btnPull;
    private Button btnDom;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnSax = (Button) findViewById(R.id.btnSAX);
        btnPull = (Button) findViewById(R.id.btnPULL);
        btnDom = (Button) findViewById(R.id.btnDOM);
        btnSax.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                Intent intent = new Intent();
                intent.setClass(XMLParserActivity.this, SAXPraserDemo.class);
                startActivity(intent);
            }
        });
        btnPull.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                Intent intent = new Intent();
                intent.setClass(XMLParserActivity.this, PullPraserDemo.class);
                startActivity(intent);
            }
        });
        btnDom.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                Intent intent = new Intent();
                intent.setClass(XMLParserActivity.this, DomPraserDemo.class);
                startActivity(intent);
            }
        });
    }
}
```

（6）单击“SAX 方式解析”按钮后，触发 SAXPraserDemo 列表以显示结果，此功能的实现文件是 SAXPraserDemo.java，具体实现代码如下。

```
public class SAXPraserDemo extends ListActivity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        SimpleAdapter adapter = null;
        try {
            adapter=new SimpleAdapter(this, getData(), R.layout.list,new String[]{"name","id"},new   int[]{
                R.id.textName,R.id.textId
            });
        } catch (ParserConfigurationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SAXException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        setListAdapter(adapter);
    }
    private List<Map<String, String>> getData() throws ParserConfigurationException, SAXException, IOException
    {
        List<channel> list;
        list=getChannelList();
        List<Map<String, String>> mapList=new ArrayList<Map<String,String>>();
        for(int i=0;i<list.size();i++)
        {
            Map<String, String> map=new HashMap<String, String>();
            map.put("name", list.get(i).getName());
            map.put("id", list.get(i).getId());
            mapList.add(map);
        }
        return mapList;

    }

    private List<channel> getChannelList() throws ParserConfigurationException, SAXException, IOException
    {
        //实例化一个 SAXParserFactory 对象
        SAXParserFactory factory=SAXParserFactory.newInstance();
        SAXParser parser;
        //实例化 SAXParser 对象，创建 XMLReader 对象，解析器
        parser=factory.newSAXParser();
        XMLReader xmlReader=parser.getXMLReader();
```

```
        //实例化 handler，事件处理器
        SAXPraserHelper helperHandler=new SAXPraserHelper();
        //解析器注册事件
        xmlReader.setContentHandler(helperHandler);
        //读取文件流
        InputStream stream=getResources().openRawResource(R.raw.channels);
        InputSource is=new InputSource(stream);
        //解析文件
        xmlReader.parse(is);
        return helperHandler.getList();
    }

}
```

在上述代码中，调用文件 SAXPraserHelper.java，此实现了具体的解析工作，文件具体实现代码如下。

```
public class SAXPraserHelper extends DefaultHandler {
    final int ITEM = 0x0005;
    List<channel> list;
    channel chann;
    int currentState = 0;
    public List<channel> getList() {
        return list;
    }
    /*
     * 接口字符块通知
     */
    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        // TODO Auto-generated method stub
        // super.characters(ch, start, length);
        String theString = String.valueOf(ch, start, length);
        if (currentState != 0) {
            chann.setName(theString);
            currentState = 0;
        }
        return;
    }
    /*
     * 接收文档结束通知
     */
    @Override
    public void endDocument() throws SAXException {
        // TODO Auto-generated method stub
        super.endDocument();
```

```
    }
    /*
     * 接收标签结束通知
     */
    @Override
    public void endElement(String uri, String localName, String qName)
                throws SAXException {
        // TODO Auto-generated method stub
        if (localName.equals("item"))
            list.add(chann);
    }
    /*
     * 文档开始通知
     */
    @Override
    public void startDocument() throws SAXException {
        // TODO Auto-generated method stub
        list = new ArrayList<channel>();
    }
    /*
     * 标签开始通知
     */
    @Override
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        // TODO Auto-generated method stub
        chann = new channel();
        if (localName.equals("item")) {
            for (int i = 0; i < attributes.getLength(); i++) {
                if (attributes.getLocalName(i).equals("id")) {
                    chann.setId(attributes.getValue(i));
                } else if (attributes.getLocalName(i).equals("url")) {
                    chann.setUrl(attributes.getValue(i));
                }
            }
            currentState = ITEM;
            return;
        }
        currentState = 0;
        return;
    }
}
```

从本步骤的实现过程可以看出，使用 SAX 解析 XML 的基本步骤如下。

- 实例化一个工厂 SAXParserFactory。
- 实例化 SAXPraser 对象，创建 XMLReader 解析器。

- 实例化 handler 处理器。
- 解析器注册一个事件。
- 读取文件流。
- 解析文件。

（7）单击“PULL 方式解析”按钮后触发 PullPraserDemo 列表显示结果，此功能的实现文件是 PullPraserDemo.java，具体实现代码如下。

```
public class PullPraserDemo extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        SimpleAdapter adapter = new SimpleAdapter(this, getData(),
                R.layout.list, new String[] { "id", "name" }, new int[] {
                        R.id.textId, R.id.textName });
        setListAdapter(adapter);
    }
    private List<Map<String, String>> getData() {
        List<Map<String, String>> list = new ArrayList<Map<String, String>>();
        XmlResourceParser xrp = getResources().getXml(R.xml.channels);
        try {
            // 直到文档的结尾处
            while (xrp.getEventType() != XmlResourceParser.END_DOCUMENT) {
                // 如果遇到了开始标签
                if (xrp.getEventType() == XmlResourceParser.START_TAG) {
                    String tagName = xrp.getName();// 获取标签的名字
                    if (tagName.equals("item")) {
                        Map<String, String> map = new HashMap<String, String>();
                        String id = xrp.getAttributeValue(null, "id");// 通过属性名来获取
                                                                      // 属性值
                        map.put("id", id);
                        String url = xrp.getAttributeValue(1);// 通过属性索引来获取属性值
                        map.put("url", url);
                        map.put("name", xrp.nextText());
                        list.add(map);
                    }
                }
                xrp.next();// 获取解析下一个事件
            }
        } catch (XmlPullParserException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
        return list;
    }
}
```

（8）单击“DOM 方式解析”按钮后，触发 DomPraserDemo 列表以显示结果，此功能的实现文件是 DomPraserDemo.java，具体实现代码如下。

```
public class DomPraserDemo extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        SimpleAdapter adapter = new SimpleAdapter(this, getData(),
                R.layout.list, new String[] { "id", "name" }, new int[] {
                        R.id.textId, R.id.textName });
        setListAdapter(adapter);
    }
    private List<Map<String, String>> getData() {
        List<Map<String, String>> list = new ArrayList<Map<String, String>>();
        InputStream stream = getResources().openRawResource(R.raw.channels);
        List<channel> channlist = DomParserHelper.getChannelList(stream);
        for (int i = 0; i < channlist.size(); i++) {
            Map<String, String> map = new HashMap<String, String>();
            channel chann = (channel) channlist.get(i);
            map.put("id", chann.getId());
            map.put("url", chann.getUrl());
            map.put("name", chann.getName());
            list.add(map);
        }
        return list;
    }
}
```

在上述代码中，调用文件 DomParserHelper.java，此实现了具体的解析工作。文件具体实现代码如下。

```
public class DomParserHelper {
    public static List<channel> getChannelList(InputStream stream)
    {
        List<channel> list=new ArrayList<channel>();
        //得到 DocumentBuilderFactory 对象, 由该对象可以得到 DocumentBuilder 对象
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        try {
            //得到 DocumentBuilder 对象
            DocumentBuilder builder=factory.newDocumentBuilder();
            //得到代表整个 XML 的 Document 对象
            Document document=builder.parse(stream);
```

```
                //得到“根节点”
                Element root=document.getDocumentElement();
                //获取根节点的所有 items 的节点
                NodeList items=root.getElementsByTagName("item");
                //遍历所有节点
                for(int i=0;i<items.getLength();i++)
                {
                    channel chann=new channel();
                    Element item=(Element)items.item(i);
                    chann.setId(item.getAttribute("id"));
                    chann.setUrl(item.getAttribute("url"));
                    chann.setName(item.getFirstChild().getNodeValue());
                    list.add(chann);
                }
            } catch (ParserConfigurationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (SAXException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return list;
        }
    }
```

从本步骤的实现过程可以看出，使用 DOM 解析 XML 的基本步骤如下。

- 调用 DocumentBuilderFactory.newInstance()方法得到 DOM 解析器工厂类实例。
- 调用解析器工厂实例类的 newDocumentBuilder()方法得到 DOM 解析器对象。
- 调用 DOM 解析器对象的 parse() 方法解析 XML 文档得到代表整个文档的 Document 对象。

到此为止，整个实例的实现过程讲解完毕。因为是解析的同一个目标文件，所以单击任意一个按钮后都会显示一样的效果，具体效果如图 6-16 所示。

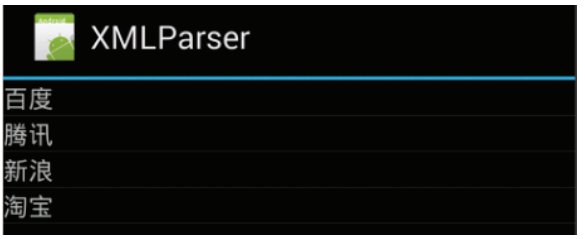


图 6-16　解析后的效果

第 7 章　WebKit 浏览网页

在 Android 系统中，WebKit 是一个系统内置的浏览器框架。WebKit 是一个经典的、开源的浏览器网页排版引擎，内含 WebCore 排版引擎和 JSCore 引擎，而这两个引擎是来自于 KDE 项目的 KHTML 和 KJS 开源项目。在 Android 平台的 Web 引擎框架中，采用了 WebKit 项目中的 WebCore 和 JSCore 部分，上层由 Java 语言封装，并且作为 API 提供给 Android 应用开发者，而底层使用 WebKit 核心库（WebCore 和 JSCore）进行网页排版。本章将详细讲解 WebKit 浏览器的基本知识，并讲解开发 Android 网页应用程序的基本方法。

7.1　WebKit 类库介绍

在 Android 源码系统中，WebKit 系统分成 Java 和 WebKit 库两个部分。

- Java 层：负责与 Android 应用程序进行通信。
- WebKit 类库：因为是由 C/C++实现的，所以也被称为 C 层库，WebKit 类库部分负责实际的网页排版处理。

在开发 Android 网络应用程序时，和开发过程密切相关的是 WebKit 类库模块。在本节的内容中，将详细讲解 Android 系统中 WebKit 类库的基本知识。

7.1.1　主要类

在 Android 系统中，WebKit 模块的 Java 层一共由 41 个文件组成，其中主要的类的具体说明如下。

（1）WebView

类 WebView 是 WebKit 模块 Java 层的视图类，所有需要使用 Web 浏览功能的 Android 应用程序都要创建该视图对象以显示和处理请求的网络资源。目前，WebKit 模块支持 HTTP、HTTPS、FTP 以及 JavaScript 请求。WebView 作为应用程序的 UI 接口，为用户提供了一系列的网页浏览、用户交互接口，客户程序通过这些接口访问 WebKit 核心代码。

（2）WebViewDatabase

类 WebViewDatabase 是 WebKit 模块中针对 SQLiteDatabase 对象的封装，用于存储和获取浏览器运行时保存的缓冲数据、历史访问数据、浏览器配置数据等。该对象是一个单实例对象，通过 getInstance 方法获取 WebViewDatabase 的实例。WebViewDatabase 是 WebKit 模块中的内部对象，仅供 WebKit 框架内部使用。

（3）WebViewCore

类 WebViewCore 是 Java 层与 C 层 WebKit 核心库的交互类，客户程序调用 WebView 的网页浏览相关操作转发给 BrowserFrame 对象。当 WebKit 核心库完成实际的数据分析和处理

后会回调 WebViweCore 中定义的一系列 JNI 接口，这些接口会通过类 CallbackProxy（代理类）将相关事件通知相应的 UI 对象。

（4）CallbackProxy

类 CallbackProxy 是一个代理类，用于实现 UI 线程和 WebCore 线程之间的交互。类 CallbackProxy 定义了一系列与用户相关的通知方法，当 WebCore 完成相应的数据处理后会调用 CallbackProxy 类中对应的方法，这些方法通过消息的方式间接调用相应的处理对象的方法。

（5）BrowserFrame

类 BrowserFrame 负责 URL 资源的载入、访问历史的维护、数据缓存等操作，该类会通过 JNI 接口直接与 WebKit 的 C 层库进行交互。

（6）JWebCoreJavaBridge

类 JWebCoreJavaBridge 的功能是，为 Java 层 WebKit 代码提供了与 C 层 WebKit 核心部分的 Timer（计时模块）和 Cookies（Cookies 身份验证模块）操作相关的方法。

（7）DownloadManagerCore

类 DownloadManagerCore 是一个下载管理核心类，主要负责网络资源的下载管理，所有的 Web 下载操作均由该类统一管理。该类实例运行在 WebKit 线程当中，与 UI 线程的交互是通过调用 CallbackProxy 对象中相应的方法完成。

（8）WebSettings

类 WebSettings 描述了 Web 浏览器访问相关用户的配置信息。

（9）DownloadListener

类 DownloadListener 负责实现侦听下载操作事件的接口，如果客户代码实现该接口，则在下载开始、失败、挂起、完成等情况下，DownloadManagerCore 对象会调用客户代码中实现的 DwonloadListener 方法。

（10）WebBackForwardList

类 WebBackForwarList 负责维护用户访问的历史记录，该类为客户程序提供操作访问浏览器历史数据的相关方法。

（11）WebViewClient

在类 WebViewClient 中定义了一系列事件方法，如果 Android 应用程序设置了 WebViewClient 派生对象，则在页面载入、资源载入、页面访问错误等情况发生时，该派生对象的相应方法会被调用。

（12）WebBackForwardListClient

类 WebBackForwardListClient 定义了对访问历史操作时可能产生的事件接口，当用户实现了该接口，则在操作访问历史时（访问历史移除、访问历史清空等）用户会得到通知。

（13）WebChromeClient

类 WebChromeClient 定义了与浏览窗口修饰相关的事件。例如接收到 Title、接收到 ICON、进度变化时，WebChromeClient 的相应方法会被调用。

7.1.2　使用内置浏览器打开网页

在 Android 系统中有一个内置浏览器，通过这个内置浏览器可以打开网页。在下面的实例中定义了一个 ListView 控件，在里面的列表中显示了 4 个菜单，单击菜单后会连接到指定

的页面。当单击 ListView 中的某一个选项后，会通过 Intent(Intent.ACTION_VIEW,uri)来打开内置的浏览器，并浏览 ListView 中创建的网页 URL。

题　目	目　的	源 码 路 径
实例 7-1	使用内置浏览器打开网页	daima\7\nei

编写主程序文件，其具体实现流程如下。

（1）通过 findViewById 构造器创建 ListView 与 TextView 对象，将文件 string.xml 中的字符串信息导入到列表中。具体实现代码如下。

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  /*通过 findViewById 构造器创建 ListView 与 TextView 对象*/
  mListView1 =(ListView) findViewById(R.id.myListView1);
  mTextView1 = (TextView) findViewById(R.id.myTextView1);
  mTextView1.setText(getResources().getString(R.string.hello));
  /*将列表通过 string.xml 中导入*/
  myFavor = new String[] {
        getResources().getString
        (R.string.str_list_url1),
        getResources().getString
      (R.string.str_list_url2),
      getResources().getString
      (R.string.str_list_url3),
      getResources().getString
      (R.string.str_list_url4)
};
```

（2）将自定义 ArrayAdapter 对象中的信息传入到 ListView 列表中，然后打开 ListAdapter 的可选菜单选项，最后设置单击 ListView 列表选项的处理事件。具体实现代码如下。

```
/*自定义 ArrayAdapter 准备传入 ListView 中,并将 myFavor 列表以参数传入*/
ArrayAdapter<String> adapter = new
ArrayAdapter<String>
(example4.this, android.R.layout.simple_list_item_1, myFavor);
/*将自定义完成的 ArrayAdapter 传入自定义的 ListView 中*/
mListView1.setAdapter(adapter);
/*将 ListAdapter 的可选菜单选项打开*/
mListView1.setItemsCanFocus(true);
/*设置 ListView 菜单选项设为每次只能单一选项*/
mListView1.setChoiceMode
(ListView.CHOICE_MODE_SINGLE);
/*设置 ListView 选项的 nItemClickListener*/
mListView1.setOnItemClickListener
```

```
(new ListView.OnItemClickListener()
{
```

（3）当用户单击一个 Item 选项后，进行比较处理，并从文件 string.xml 中读取出对应的 URL 网址字符串，然后将字符串转换为 URL 对象。具体实现代码如下。

```
/*覆盖 OnItemClick 方法*/
public void onItemClick
(AdapterView<?> arg0, View arg1, int arg2,long arg3)
{
  // TODO Auto-generated method stub
  /*若所选菜单的文字与 myFavor 字符串数组第一个文字相同*/
  if(arg0.getAdapter().getItem(arg2).toString()==
  myFavor[0].toString())
  {
    /*取得网址并调用 goToUrl()方法*/
    myUrl=getResources().getString(R.string.str_url1);
    goToUrl(myUrl);
  }
  /*若所选菜单的文字与 myFavor 字符串数组第二个文字相同*/
  else if (arg0.getAdapter().getItem(arg2).toString()==
  myFavor[1].toString())
  {
    /*取得网址并调用 goToUrl()方法*/
    myUrl=getResources().getString(R.string.str_url2);
    goToUrl(myUrl);
  }
  /*若所选菜单的文字与 myFavor 字符串数组第三个文字相同*/
  else if (arg0.getAdapter().getItem(arg2).toString()==
  myFavor[2].toString())
  {
    /*取得网址并调用 goToUrl()方法*/
    myUrl=getResources().getString(R.string.str_url3);
    goToUrl(myUrl);
  }
  /*若所选菜单的文字与 myFavor 字符串数组第四个文字相同*/
  else if (arg0.getAdapter().getItem(arg2).toString()==
  myFavor[3].toString())
  {
    /*取得网址并调用 goToUrl()方法*/
    myUrl=getResources().getString(R.string.str_url4);
    goToUrl(myUrl);
  }
  /*以上皆非*/
  else
  {
    /*显示错误信息*/
```

```
                mTextView1.setText("Ooops!!出错了");
            }
        }
    });
}
```

（4）定义方法 goToUrl(String url)来打开网址为 URL 的网页，具体实现代码如下。

```
/*打开网页的方法*/
private void goToUrl(String url)
{
    Uri uri = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    startActivity(intent);
}
```

执行后列表将显示 4 个菜单，如图 7-1 所示。当单击一个菜单后，会跳转到对应的目标页面。

图 7-1　4 个菜单

7.2　Android 5.0 中的 WebView

文件 WebView.java 实现了类 WebView，此类是 WebKit 模块 Java 层的视图类，所有需要使用 Web 浏览功能的 Android 应用程序都要创建该视图对象以显示和处理请求的网络资源。目前，WebKit 模块支持 HTTP、HTTPS、FTP 以及 JavaScript 请求。从 Android 5.0 开始，引入了 Mixed Content（图文内容混合）模式和第三方 Cookie。在本节的内容中，将详细讲解 Android 5.0 系统中 WebView 的核心架构知识，为读者深入理解 Android 网络系统的架构打下基础。

7.2.1　WebView 架构基础

在 Android 5.0 系统中，文件 WebView.java 是一个内置的支持浏览器的视图 View，此文件位于如下的目录中：

```
frameworks\base\core\java\android\webkit
```

WebView 作为应用程序的 UI 接口，为用户提供了一系列的网页浏览、用户交互接口，客户程序通过这些接口访问 WebKit 核心代码。WebView 实现了 WebKit 的最核心功能。近年来随着用户对网络安全的重视，谷歌公司为了创建一个安全、稳定和快速的通用浏览器系统，特意推出了 Chromium 引擎驱动，这也是谷歌官方浏览器 Chrome 的引擎驱动。在全新的 Android 5.0 系统中，WebView 将开始采用 Chromium 引擎驱动。

浏览 Android 5.0 源码时会发现，之前版本中的“external/WebKit”目录已经被移除掉，取而代之的是“chromium_org”。由此可见 Chromium 已经完全取代了之前的 WebKit for Android。虽然 Chromium 完全取代了以前的 WebKit for Android，但是 Android WebView 的 API 接口并没有变，与老的版本完全兼容。这样带来的好处是基于 WebView 构建的 APP，无需做任何修改即可享受 Chromium 内核的高效与强大。

在 Android 5.0 系统中，“frameworks/base/core/java/android/webkit”目录下的各个文件如图 7-2 所示。

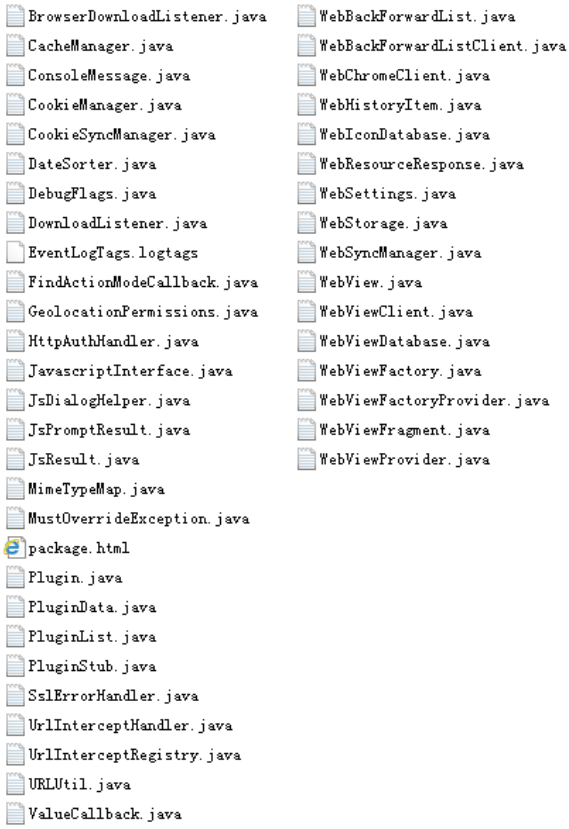


图 7-2　“frameworks/base/core/java/android/webkit”目录下的文件

在 Android 5.0 中，设计者设计出了一个 WebViewProvider 接口，WebView 本身并不实现具体的功能，而是将所有的处理功能交给 WebViewProvider 来实现。而 WebViewProvider 只是一个接口，至于具体的解决方案由实现者来决定到底采用哪一种引擎。其实从 Android 4.1 系统开始就采用了这种架构模式，这说明谷歌那时便已经开始为以 Chromium 为驱动的

WebView 做准备。Android 5.0 之前的版本，通过 WebViewClassic 实现 WebViewProvider 接口。Android 5.0 系统中，则通过 WebViewChromium 实现 WebViewProvider 接口。

WebViewFactory 也采用了相似的结构，设置了实例化 WebViewFactoryProvider 的具体策略。在 WebViewFactoryProvider 中有一个十分关键的接口 createWebView，功能是创建具体的 WebViewProvider。

Android 5.0 中 WebView 的代码结构如图 7-3 所示。

由此可见，传统的基于 AOSP master 架构由 WebKit 实现的仍然保留，因为采用了灵活的架构，所以可以在 AOSP 和 Chromium 两种核心之间非常容易切换。

在 Content API 之上，Chromium 的 WebView 实现封装了一个新的类 AwContents，该类主要基于 ContentViewCore 类的实现，不同的是，AwContents 需要基于一个原来存在于“chrome/”目录下的模块（图 7-3 中的 Browser Components），但是 AwContents 不应该依赖该目录，所以，将 Chrome 中的一些功能实现模块化处理是 Chromium 的一个发展方向。目前，一些功能模块已经从 Chrome 中抽取出来了，具体请浏览“components/”。

因为在 AwContents 中提供的不是 WebView 的 API，所以需要借助中间桥接层，将 AwContents 桥接到 WebView，这就是上图中的桥接模块，该模块位于 Android 5.0 源代码中的“frameworks/webview/chromium/java/com/android/webview/chromium/”目录下。类 WebView Chromium 和类 WebViewChromiumFactory 作为 WebView 的具体实现，需要依赖于 Chromium 项目的 AwContents 模块来实现。整体模块架构如图 7-4 所示。

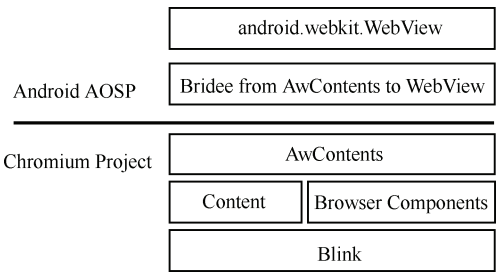


图 7-3　Android 5.0 中 WebView 的代码结构

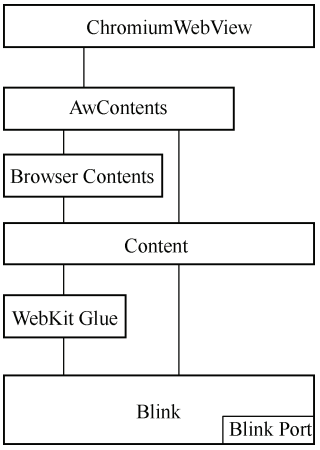


图 7-4　AwContents 模块的架构图

AwContents 基于 Content 之上，专门针对 WebView 的需求进行封装，这个封装只是针对 Android 平台实现。同样道理，WebView 也是基于 Content API（Web Contents 和 ContentViewCore 等）之上的，这一点它同 Content Shell 和 Chromium 浏览器没有大的不同，区别在于它们对很多 Delegate 类的实现不同，这是 Content API 用于让使用者参与内部逻辑和实现的过程。具体来说，主要有以下两个方面的不同。

（1）渲染机制

因为 WebView 提供的是一个 View 控件，所以 View 控件的容器可能会接受储存在 CPU 中的结构，例如 bitmap，也可能是储存在 GPU 内存中的结构，例如 surface，所以它需要提供

两种不同的输出结果。

Chromium 引入了一种新的合成器 UberCompositor++，该合成器支持输出到 GPU 和 CPU 内存的两种方式。对于 Compositor 的结果输出到给定 View 的 GPU 内存的这种方式来说，关键点在于实现 AwContents.InternalAccessDelegate 接口的 requestDrawGL 方法。与 requestDrawGL 实现有关的代码不多，只有文件 DrawGLFunctor.java 和文件 GraphicsUtils.java，还有与之关联的本地实现文件 draw_gl_functor.cpp、raphic_buffer_impl.cpp 和 graphics_utils.cpp，本地代码位于“frameworks/webview/chromium/plat_support”目录下，此部分并不仅使用了 Android SDK 和 NDK 的 API，而且也涉及到了 Android 源码。

（2）进程

目前 WebView 只支持单进程，未来版本应该支持多进程方式。单进程意味着没有办法使用 Android 的 isolated UID 机制，因此，某种程度上来讲，安全性降低了，而且页面的渲染崩溃会导致使用 WebView 的应用程序崩溃。

7.2.2　WebView 类简介

在 Android 5.0 系统中，WebView 类的实现文件 WebView.java 位于如下的目录中。

```
frameworks\base\core\java\android\webkit
```

类 WebView 继承于 AbsoluteLayout，实现 OnGlobalFocusChangeListener 和 OnHierarchy ChangeListener，类 WebView 的构造函数的具体实现代码如下。

```
@SuppressWarnings("deprecation")
uper() call into deprecated base class constructor.
    protected WebView(Context context, AttributeSet attrs, int defStyle,
            Map<String, Object> javaScriptInterfaces, boolean privateBrowsing) {
        super(context, attrs, defStyle);
        if (context == null) {
            throw new IllegalArgumentException("Invalid context argument");
        }
        sEnforceThreadChecking = context.getApplicationInfo().targetSdkVersion >=
                Build.VERSION_CODES.JELLY_BEAN_MR2;
        checkThread();
        if (DebugFlags.TRACE_API) Log.d(LOGTAG, "WebView<init>");

        ensureProviderCreated();
        mProvider.init(javaScriptInterfaces, privateBrowsing);
        CookieSyncManager.setGetInstanceIsAllowed();
    }
    public void addJavascriptInterface(Object object, String name) {
        checkThread();
        if (DebugFlags.TRACE_API) Log.d(LOGTAG, "addJavascriptInterface=" + name);
        mProvider.addJavascriptInterface(object, name);
    }
    public boolean canGoBack() {
```

```
            checkThread();
            return mProvider.canGoBack();
        }
    public boolean canGoBackOrForward(int steps) {
            checkThread();
            return mProvider.canGoBackOrForward(steps);
        }
    public void goForward() {
            checkThread();
            if (DebugFlags.TRACE_API) Log.d(LOGTAG, "goForward");
            mProvider.goForward();
        }
      public boolean canZoomIn() {
            checkThread();
            return mProvider.canZoomIn();
        }
    //注入所提供的 Java 对象到这个 Web 视图
        public void addJavascriptInterface(Object object, String name) {
            checkThread();
            if (DebugFlags.TRACE_API) Log.d(LOGTAG, "addJavascriptInterface=" + name);
            mProvider.addJavascriptInterface(object, name);
        }
    //获取此的 WebView 是否有背历史的项目
        public boolean canGoBack() {
            checkThread();
            return mProvider.canGoBack();
        }
    //获取页面是否可以回去或转发的步骤给定数量
        public boolean canGoBackOrForward(int steps) {
            checkThread();
            return mProvider.canGoBackOrForward(steps);
        }
    //获取此的 WebView 是否有历史前进的项目
        public void goForward() {
            checkThread();
            if (DebugFlags.TRACE_API) Log.d(LOGTAG, "goForward");
            mProvider.goForward();
        }
    //获取此的 WebView 是否可以将其放大
        public boolean canZoomIn() {
            checkThread();
            return mProvider.canZoomIn();
        }
    //告诉这个 Web 视图，以清除其内部的前进/后退清单
        public void clearHistory() {
            checkThread();
```

```
            if (DebugFlags.TRACE_API) Log.d(LOGTAG, "clearHistory");
            mProvider.clearHistory();
        }
    //获取第一个子串组成的物理位置的地址
        public static String findAddress(String addr) {
            return getFactory().getStatics().findAddress(addr);
        }
    //获取的 HTML 内容的高度
        public int getContentHeight() {
            checkThread();
            return mProvider.getContentHeight();
        }
    //获取当前页面的原始 URL
        public String getOriginalUrl() {
            checkThread();
            return mProvider.getOriginalUrl();
        }
    //获取当前页面的进度
        public int getProgress() {
            checkThread();
            return mProvider.getProgress();
        }
```

7.2.3　WebViewProvider 接口

接口 WebViewProvider 的实现文件是 WebViewProvider.java，在里面定义了很多接口函数，主要实现代码如下。

```
public interface WebViewProvider {
    public void init(Map<String, Object> javaScriptInterfaces,
            boolean privateBrowsing);
    public void setHorizontalScrollbarOverlay(boolean overlay);
    public void setVerticalScrollbarOverlay(boolean overlay);
    public boolean overlayHorizontalScrollbar();
    public boolean overlayVerticalScrollbar();
    public int getVisibleTitleHeight();
    public SslCertificate getCertificate();
    public void setCertificate(SslCertificate certificate);
    public void savePassword(String host, String username, String password);
    public void setHttpAuthUsernamePassword(String host, String realm,
            String username, String password);
    public String[] getHttpAuthUsernamePassword(String host, String realm);
    public void destroy();
    public void setNetworkAvailable(boolean networkUp);
    public WebBackForwardList saveState(Bundle outState);
    public boolean savePicture(Bundle b, final File dest);
```

```
public boolean restorePicture(Bundle b, File src);
public WebBackForwardList restoreState(Bundle inState);
public void loadUrl(String url, Map<String, String> additionalHttpHeaders);
public void loadUrl(String url);
public void postUrl(String url, byte[] postData);
public void loadData(String data, String mimeType, String encoding);
public void loadDataWithBaseURL(String baseUrl, String data,
        String mimeType, String encoding, String historyUrl);
public void evaluateJavaScript(String script, ValueCallback<String> resultCallback);
public void saveWebArchive(String filename);
public void saveWebArchive(String basename, boolean autoname, ValueCallback<String> callback);
public void stopLoading();
public void reload();
public boolean canGoBack();
public void goBack();
  public boolean canGoForward();
public void goForward();
public boolean canGoBackOrForward(int steps);
public void goBackOrForward(int steps);
public boolean isPrivateBrowsingEnabled();
public boolean pageUp(boolean top);
public boolean pageDown(boolean bottom);
public void clearView();
public Picture capturePicture();
public PrintDocumentAdapter createPrintDocumentAdapter();
public float getScale();
public void setInitialScale(int scaleInPercent);
public void invokeZoomPicker();
public HitTestResult getHitTestResult();
public void requestFocusNodeHref(Message hrefMsg);
public void requestImageRef(Message msg);
public String getUrl();
public String getOriginalUrl();
public String getTitle();
public Bitmap getFavicon();
public String getTouchIconUrl();
public int getProgress();
public int getContentHeight();
public int getContentWidth();
public void pauseTimers();
public void resumeTimers();
public void onPause();
public void onResume();
public boolean isPaused();
public void freeMemory();
public void clearCache(boolean includeDiskFiles);
```

```
    public void clearFormData();
    public void clearHistory();
    public void clearSslPreferences();
    public WebBackForwardList copyBackForwardList();
    public void setFindListener(WebView.FindListener listener);
    public void findNext(boolean forward);
    public int findAll(String find);
    public void findAllAsync(String find);
    public boolean showFindDialog(String text, boolean showIme);
    public void clearMatches();
    public void documentHasImages(Message response);
    public void setWebViewClient(WebViewClient client);
    public void setDownloadListener(DownloadListener listener);
    public void setWebChromeClient(WebChromeClient client);
    public void setPictureListener(PictureListener listener);
    public void addJavascriptInterface(Object obj, String interfaceName);
    public void removeJavascriptInterface(String interfaceName);
    public WebSettings getSettings();
    public void setMapTrackballToArrowKeys(boolean setMap);
    public void flingScroll(int vx, int vy);
    public View getZoomControls();
    public boolean canZoomIn();
    public boolean canZoomOut();
    public boolean zoomIn();
    public boolean zoomOut();
    public void dumpViewHierarchyWithProperties(BufferedWriter out, int level);
    public View findHierarchyView(String className, int hashCode);
        public boolean shouldDelayChildPressedState();
        public AccessibilityNodeProvider getAccessibilityNodeProvider();
        public void onInitializeAccessibilityNodeInfo(AccessibilityNodeInfo info);
        public void onInitializeAccessibilityEvent(AccessibilityEvent event);
        public boolean performAccessibilityAction(int action, Bundle arguments);
        public void setOverScrollMode(int mode);
        public void setScrollBarStyle(int style);
        public void onDrawVerticalScrollBar(Canvas canvas, Drawable scrollBar, int l, int t,
                int r, int b);
        public void onOverScrolled(int scrollX, int scrollY, boolean clampedX, boolean clampedY);
        public void onWindowVisibilityChanged(int visibility);
        public void onDraw(Canvas canvas);
        public void setLayoutParams(LayoutParams layoutParams);
        public boolean performLongClick();
        public void onConfigurationChanged(Configuration newConfig);
        public InputConnection onCreateInputConnection(EditorInfo outAttrs);
        public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event);
        public boolean onKeyDown(int keyCode, KeyEvent event);
……
```

在 Android 5.0 系统中，通过 WebViewChromium 实现 WebViewProvider 接口。另外，WebViewFactory 也采用了相似的结构，决定如何实例化 WebViewFactoryProvider，WebViewFactoryProvider 的关键的接口是 createWebView，功能是创建具体的 WebViewProvider。

7.2.4 WebViewChromium 详解

在 Android 5.0 系统中，WebViewChromium 实现 WebViewProvider 接口。WebViewChromium 在如下所示的文件中定义。

```
frameworks\webview\chromium\java\com\android\webview\chromium\WebViewChromium.java
```

在文件 WebViewChromium.java 中实现了 WebViewProvider 中各个方法的具体功能，主要实现代码如下。

```
.....
    public void init(final Map<String, Object> javaScriptInterfaces,
            final boolean privateBrowsing) {
        if (privateBrowsing) {
            mFactory.startYourEngines(true);
            final String msg = "Private browsing is not supported in WebView.";
            if (mAppTargetSdkVersion >= Build.VERSION_CODES.KITKAT) {
                throw new IllegalArgumentException(msg);
            } else {
                Log.w(TAG, msg);
                TextView warningLabel = new TextView(mWebView.getContext());
                warningLabel.setText(mWebView.getContext().getString(
                        com.android.internal.R.string.webviewchromium_private_browsing_warning));
                mWebView.addView(warningLabel);
            }
        }
        if (mAppTargetSdkVersion >= Build.VERSION_CODES.JELLY_BEAN_MR2) {
            mFactory.startYourEngines(false);
            checkThread();
        } else if (!mFactory.hasStarted()) {
            if (Looper.myLooper() == Looper.getMainLooper()) {
                mFactory.startYourEngines(true);
            }
        }
        final boolean isAccessFromFileURLsGrantedByDefault =
                mAppTargetSdkVersion < Build.VERSION_CODES.JELLY_BEAN;
        final boolean areLegacyQuirksEnabled =
                mAppTargetSdkVersion < Build.VERSION_CODES.KITKAT;
        mContentsClientAdapter = new WebViewContentsClientAdapter(mWebView);
        mWebSettings = new ContentSettingsAdapter(new AwSettings(
                mWebView.getContext(), isAccessFromFileURLsGrantedByDefault,
```

```
                areLegacyQuirksEnabled));
        mRunQueue.addTask(new Runnable() {
                @Override
                public void run() {
                    initForReal();
                    if (privateBrowsing) {
                        destroy();
                    }
                }
        });
    }
    private void initForReal() {
        mAwContents = new AwContents(mFactory.getBrowserContext(), mWebView,
                new InternalAccessAdapter(), mContentsClientAdapter, new AwLayoutSizer(),
                mWebSettings.getAwSettings());
        if (mAppTargetSdkVersion >= Build.VERSION_CODES.KITKAT) {
            AwContents.setShouldDownloadFavicons();
        }
    }
    void startYourEngine() {
        mRunQueue.drainQueue();
    }
    private RuntimeException createThreadException() {
        return new IllegalStateException(
                "Calling View methods on another thread than the UI thread.");
    }
……
```

上面只是列出了几个方法，具体内容读者可以参阅 Android 5.0 的源码。在 Android 5.0 系统中，WebViewChromium 通过调用 Chromium 项目的 AwContents 模块来实现具体功能。

7.2.5　WebViewChromiumFactoryProvider 详解

在 Android 5.0 系统中，文件 WebViewChromiumFactoryProvider.java 的功能是实例化 WebViewFactoryProvider，此文件位于如下的目录中。

```
frameworks\webview\chromium\java\com\android\webview\chromium\
```

文件 WebViewChromiumFactoryProvider.java 也同样需要依赖于 Chromium 项目的 AwContents 模块来实现具体功能，主要实现代码如下。

```
    private void ensureChromiumStartedLocked(boolean onMainThread) {
        assert Thread.holdsLock(mLock);
        if (mStarted) {
            return;
        }
Looper looper = !onMainThread ? Looper.myLooper() : Looper.getMainLooper();
```

```
        Log.v("WebViewChromium", "Binding Chromium to the " +
                (onMainThread ? "main":"background") + " looper " + looper);
        ThreadUtils.setUiThread(looper);
        if (ThreadUtils.runningOnUiThread()) {
            startChromiumLocked();
            return;
        }
        //通过 CookieManager 实现线程安全的方式启动 Chromium
        ThreadUtils.postOnUiThread(new Runnable() {
            @Override
            public void run() {
                synchronized (mLock) {
                    startChromiumLocked();
                }
            }
        });
        while (!mStarted) {
            try {
                mLock.wait();
            } catch (InterruptedException e) {
                //继续尝试…最终用户界面将处理我们发送的任务
            }
        }
    }
    private void startChromiumLocked() {
        assert Thread.holdsLock(mLock) && ThreadUtils.runningOnUiThread();
        //通知返回路径
        mLock.notifyAll();
        if (mStarted) {
            return;
        }
        if (Build.IS_DEBUGGABLE) {
            CommandLine.initFromFile(COMMAND_LINE_FILE);
        } else {
            CommandLine.init(null);
        }
        CommandLine cl = CommandLine.getInstance();
        cl.appendSwitch("enable-dcheck");
        if (!cl.hasSwitch("disable-webview-gl-mode")) {
            cl.appendSwitch("testing-webview-gl-mode");
        }
        Context context = ActivityThread.currentApplication();
        if (context.getApplicationInfo().targetSdkVersion < Build.VERSION_CODES.KITKAT) {
            cl.appendSwitch("enable-webview-classic-workarounds");
        }
        //无须提取额外的系统图像
```

```
        ResourceExtractor.setMandatoryPaksToExtract("");
        try {
            LibraryLoader.ensureInitialized();
        } catch(ProcessInitException e) {
            throw new RuntimeException("Error initializing WebView library", e);
        }
        PathService.override(PathService.DIR_MODULE, "/system/lib/");
        final int DIR_RESOURCE_PAKS_ANDROID = 3003;
        PathService.override(DIR_RESOURCE_PAKS_ANDROID,
                "/system/framework/webview/paks");
        AwBrowserProcess.start(ActivityThread.currentApplication());
        initPlatSupportLibrary();
        if (Build.IS_DEBUGGABLE) {
            setWebContentsDebuggingEnabled(true);
        }
        mStarted = true;
        for (WeakReference<WebViewChromium> wvc : mWebViewsToStart) {
            WebViewChromium w = wvc.get();
            if (w != null) {
                w.startYourEngine();
            }
        }
        mWebViewsToStart.clear();
        mWebViewsToStart = null;
    }
    @Override
    public Statics getStatics() {
        synchronized (mLock) {
            if (mStaticMethods == null) {
                ensureChromiumStartedLocked(true);
                mStaticMethods = new WebViewFactoryProvider.Statics() {
                    @Override
                    public String findAddress(String addr) {
                        return ContentViewStatics.findAddress(addr);
                    }
                    @Override
                    public void setPlatformNotificationsEnabled(boolean enable) {
                    }
                    @Override
                    public String getDefaultUserAgent(Context context) {
                        return AwSettings.getDefaultUserAgent();
                    }
                    @Override
                    public void setWebContentsDebuggingEnabled(boolean enable) {
                        if (!Build.IS_DEBUGGABLE) {
                            WebViewChromiumFactoryProvider.this.
```

```
                                        setWebContentsDebuggingEnabled(enable);
                                    }
                                }
                            };
                        }
                    }
                    return mStaticMethods;
                }
```

7.2.6 AwContents 架构

在 Content API 之上，Chromium 的 WebView 封装了一个新的类 AwContents，该类主要基于 ContentViewCore 类的实现。不同的是，AwContents 需要基于一个原来存在于“chrome/”目录下的模块 BrowserComponents。但是因为 AwContents 不应该依赖该目录，所以将 Chrome 中的一些浏览器模块化是 Chromium 的一个发展方向。

AwContents 模块的实现文件是 AwContents.java，在如下的目录中实现。

```
external\chromium_org\android_webview\java\src\org\chromium\android_webview\
```

因为在文件 AwContents.java 中没有实现 WebView API 的任何源码，所以接下来需要使用桥接层将 AwContents 连接到 WebView。文件 AwContents.java 的主要实现代码如下。

```
        @JNINamespace("android_webview")
public class AwContents {
    private static final String TAG = "AwContents";
    private static final String WEB_ARCHIVE_EXTENSION = ".mht";
    //避免在输出时发生缩放
    private static final float ZOOM_CONTROLS_EPSILON = 0.007f;
    public static class HitTestData {
        public int hitTestResultType;
        public String hitTestResultExtraData;
        //用在 requestFocusNodeHref 和 requestImageRef
        public String href;
        public String anchorText;
        public String imgSrc;
    }
    public AwContents(AwBrowserContext browserContext, ViewGroup containerView,
            InternalAccessDelegate internalAccessAdapter, AwContentsClient contentsClient,
            boolean isAccessFromFileURLsGrantedByDefault, AwLayoutSizer layoutSizer,
            boolean supportsLegacyQuirks) {
        this(browserContext, containerView, internalAccessAdapter, contentsClient,
                layoutSizer, new AwSettings(containerView.getContext(),
                        isAccessFromFileURLsGrantedByDefault, supportsLegacyQuirks));
    }
    public AwContents(AwBrowserContext browserContext, ViewGroup containerView,
            InternalAccessDelegate internalAccessAdapter, AwContentsClient contentsClient,
```

```
            AwLayoutSizer layoutSizer, AwSettings settings) {
        mBrowserContext = browserContext;
        mContainerView = containerView;
        mInternalAccessAdapter = internalAccessAdapter;
        mContentsClient = contentsClient;
        mLayoutSizer = layoutSizer;
        mSettings = settings;
        mDIPScale = DeviceDisplayInfo.create(mContainerView.getContext()).getDIPScale();
        mLayoutSizer.setDelegate(new AwLayoutSizerDelegate());
        mLayoutSizer.setDIPScale(mDIPScale);
        mWebContentsDelegate = new AwWebContentsDelegateAdapter(contentsClient, mContainerView);
        mContentsClientBridge = new AwContentsClientBridge(contentsClient);
        mZoomControls = new AwZoomControls(this);
        mIoThreadClient = new IoThreadClientImpl();
        mInterceptNavigationDelegate = new InterceptNavigationDelegateImpl();
        AwSettings.ZoomSupportChangeListener zoomListener =
                new AwSettings.ZoomSupportChangeListener() {
                    @Override
                    public void onGestureZoomSupportChanged(boolean supportsGestureZoom) {
                        mContentViewCore.updateMultiTouchZoomSupport(supportsGestureZoom);
                        mContentViewCore.updateDoubleTapDragSupport(supportsGestureZoom);
                    }
                };
        mSettings.setZoomListener(zoomListener);
        mDefaultVideoPosterRequestHandler = new DefaultVideoPosterRequestHandler(mContentsClient);
        mSettings.setDefaultVideoPosterURL(
                mDefaultVideoPosterRequestHandler.getDefaultVideoPosterURL());
        mSettings.setDIPScale(mDIPScale);
        mScrollOffsetManager = new AwScrollOffsetManager(new AwScrollOffsetManagerDelegate(),
                new OverScroller(mContainerView.getContext()));
        setOverScrollMode(mContainerView.getOverScrollMode());
        setScrollBarStyle(mInternalAccessAdapter.super_getScrollBarStyle());
        mContainerView.addOnLayoutChangeListener(new AwLayoutChangeListener());
        setNewAwContents(nativeInit(mBrowserContext));
        onVisibilityChanged(mContainerView, mContainerView.getVisibility());
        onWindowVisibilityChanged(mContainerView.getWindowVisibility());
    }
    /**
     * AwContents 实例的初始化
     */
    private void setNewAwContents(int newAwContentsPtr) {
        if (mNativeAwContents != 0) {
            destroy();
            mContentViewCore = null;
        }
        assert mNativeAwContents == 0 && mCleanupReference == null && mContentViewCore == null;
```

```
        mNativeAwContents = newAwContentsPtr;
        //绑定所有信息到本机
        mCleanupReference = new CleanupReference(this, new DestroyRunnable(mNativeAwContents));
        int nativeWebContents = nativeGetWebContents(mNativeAwContents);
        mContentViewCore = createAndInitializeContentViewCore(
                mContainerView, mInternalAccessAdapter, nativeWebContents,
                new AwGestureStateListener(), mContentsClient.getContentViewClient(),
                mZoomControls);
        nativeSetJavaPeers(mNativeAwContents, this, mWebContentsDelegate, mContentsClientBridge,
                mIoThreadClient, mInterceptNavigationDelegate);
        mContentsClient.installWebContentsObserver(mContentViewCore);
        mSettings.setWebContents(nativeWebContents);
        nativeSetDipScale(mNativeAwContents, (float) mDIPScale);
        updateGlobalVisibleRect();
        mContentViewCore.onShow();
    }
     /**
      * 呼叫"source" AwContents，打开一个弹出式窗口
      */
     public void supplyContentsForPopup(AwContents newContents) {
        int popupNativeAwContents = nativeReleasePopupAwContents(mNativeAwContents);
        if (popupNativeAwContents == 0) {
            Log.w(TAG, "Popup WebView bind failed: no pending content.");
            if (newContents != null) newContents.destroy();
            return;
        }
        if (newContents == null) {
            nativeDestroy(popupNativeAwContents);
            return;
        }
        newContents.receivePopupContents(popupNativeAwContents);
     }
     private void receivePopupContents(int popupNativeAwContents) {
        //保存现有视图状态
        final boolean wasAttached = mIsAttachedToWindow;
        final boolean wasViewVisible = mIsViewVisible;
        final boolean wasWindowVisible = mIsWindowVisible;
        final boolean wasPaused = mIsPaused;
        final boolean wasFocused = mContainerViewFocused;
        final boolean wasWindowFocused = mWindowFocused;
        // 处理现有的 mContentViewCore 和 mNativeAwContents.
        if (wasFocused) onFocusChanged(false, 0, null);
        if (wasWindowFocused) onWindowFocusChanged(false);
        if (wasViewVisible) setViewVisibilityInternal(false);
        if (wasWindowVisible) setWindowVisibilityInternal(false);
        if (!wasPaused) onPause();
```

```
        //处理 GL 资源
        setNewAwContents(popupNativeAwContents);
        //刷新所有的视图状态
        if (!wasPaused) onResume();
        if (wasAttached) onAttachedToWindow();
        onSizeChanged(mContainerView.getWidth(), mContainerView.getHeight(), 0, 0);
        if (wasWindowVisible) setWindowVisibilityInternal(true);
        if (wasViewVisible) setViewVisibilityInternal(true);
        if (wasWindowFocused) onWindowFocusChanged(wasWindowFocused);
        if (wasFocused) onFocusChanged(true, 0, null);
    }

    /**
     *删除此对象的本地副本
     */
    public void destroy() {
        if (mCleanupReference != null) {
            mContentViewCore.destroy();
            mNativeAwContents = 0;
            if (mIsAttachedToWindow) {
                if (mPendingDetachCleanupReferences == null) {
                    mPendingDetachCleanupReferences = new ArrayList<CleanupReference>();
                }
                mPendingDetachCleanupReferences.add(mCleanupReference);
            } else {
                mCleanupReference.cleanupNow();
            }
            mCleanupReference = null;
        }
        assert !mContentViewCore.isAlive();
        assert mNativeAwContents == 0;
    }
```

由此可见，AwContents 和之前版本的 WebView 是一个概念，用于存放网页渲染的结果。并且 AwContents 都是基于 Android SDK/NDK 开发，并没有使用 Android Source 中未公开的 API 和库。

7.2.7　实现 Mixed Content 模式

在 Android 5.0 的 WebView 系统中，将通过函数 setMixedContentMode 和 getMixedContentMode 实现网页的 Mixed Content 模式。函数 setMixedContentMode 和 getMixedContentMode 在文件 platform/frameworks/base/core/java/android/webkit/WebSettings.java 中定义，具体的定义原型如下。

```
public abstract void setMixedContentMode(int mode);
public abstract int getMixedContentMode();
public static final int MIXED_CONTENT_ALWAYS_ALLOW = 0;
```

```
public static final int MIXED_CONTENT_NEVER_ALLOW = 1;
public static final int MIXED_CONTENT_COMPATIBILITY_MODE = 2;
```

函数 setMixedContentMode 和 getMixedContentMode 在文件 AwSettings.java 中定义，具体实现代码如下。

```
public void setMixedContentMode(int mode) {
    synchronized (mAwSettingsLock) {
        if (mMixedContentMode != mode) {
            mMixedContentMode = mode;
            mEventHandler.updateWebkitPreferencesLocked();
        }
    }
}
public int getMixedContentMode() {
    synchronized (mAwSettingsLock) {
        return mMixedContentMode;
    }
}
```

7.2.8 引入第三方 Cookie

在 Android 5.0 的 WebView 系统中，将通过函数 setAcceptThirdPartyCookies 和 acceptThirdPartyCookies 在网页中引入第三方 Cookie。函数 setAcceptThirdPartyCookies 和 acceptThirdPartyCookies 在文件 platform/frameworks/base/core/java/android/webkit/CookieManager.java 中定义，具体定义原型如下。

```
    protected Object clone() throws CloneNotSupportedException {
    throw new CloneNotSupportedException("doesn't implement Cloneable");
    }
    public static synchronized CookieManager getInstance() {
    return WebViewFactory.getProvider().getCookieManager();
    }
    public synchronized void setAcceptCookie(boolean accept) {
    throw new MustOverrideException();
    }
    public synchronized boolean acceptCookie() {
    throw new MustOverrideException();
    }
    public void setAcceptThirdPartyCookies(WebView webview, boolean accept) {
    throw new MustOverrideException();
    }
    public boolean acceptThirdPartyCookies(WebView webview) {
    throw new MustOverrideException();
    }
    public void setCookie(String url, String value) {
```

```
throw new MustOverrideException();
}
public void setCookie(String url, String value, ValueCallback<Boolean> callback) {
throw new MustOverrideException();
}
public String getCookie(String url) {
throw new MustOverrideException();
}
public String getCookie(String url, boolean privateBrowsing) {
throw new MustOverrideException();
}
public synchronized String getCookie(WebAddress uri) {
throw new MustOverrideException();
}
public void removeSessionCookie() {
throw new MustOverrideException();
}
public void removeSessionCookies(ValueCallback<Boolean> callback) {
throw new MustOverrideException();
}
@Deprecated
public void removeAllCookie() {
throw new MustOverrideException();
}
public void removeAllCookies(ValueCallback<Boolean> callback) {
throw new MustOverrideException();
}
public synchronized boolean hasCookies() {
throw new MustOverrideException();
}
public synchronized boolean hasCookies(boolean privateBrowsing) {
throw new MustOverrideException();
}
@Deprecated
public void removeExpiredCookie() {
throw new MustOverrideException();
}
public void flush() {
flushCookieStore();
}
protected void flushCookieStore() {
}
public static boolean allowFileSchemeCookies() {
return getInstance().allowFileSchemeCookiesImpl();
}
protected boolean allowFileSchemeCookiesImpl() {
```

```
        throw new MustOverrideException();
        }
        // Static for backward compatibility.
        public static void setAcceptFileSchemeCookies(boolean accept) {
        getInstance().setAcceptFileSchemeCookiesImpl(accept);
        }
        protected void setAcceptFileSchemeCookiesImpl(boolean accept) {
        throw new MustOverrideException();
        }
}
```

7.2.9 实战演练——在手机屏幕中浏览网页

在 Android 系统中，可以使用内置的 WebKit 引擎中的 WebView 迅速浏览网页。在本实例中，通过 WebView.loadUrl 来加载网址。所以从 EditText 传入要浏览的网址后，就可以在 WebView 中加载网页的内容了。

题　　目	目　　的	源 码 路 径
实例 7-2	在手机屏幕中浏览网页	daima\7\wang

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，在里面插入一个 WebView 控件。主要实现代码如下。

```
        <!-- 建立一个 TextView -->
        <TextView
        android:id="@+id/myTextView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
        <!-- 建立一个 EditText -->
        <EditText
        android:id="@+id/myEditText1"
        android:layout_width="267px"
        android:layout_height="40px"
        android:textSize="18sp"
        android:layout_x="5px"
        android:layout_y="32px"
        />
        <!-- 建立一个 ImageButton -->
        <ImageButton
        android:id="@+id/myImageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/white"
        android:src="@drawable/go"
```

```
android:layout_x="275px"
android:layout_y="35px"
/>
<!-- 建立一个 WebView -->
<WebView
android:id="@+id/myWebView1"
android:layout_height="330px"
android:layout_width="300px"
android:layout_x="7px"
android:layout_y="90px"
android:background="@drawable/black"
android:focusable="false"
/>
```

（2）编写文件 wang.java，通过 setOnClickListener 监听按钮单击事件，单击网址后面的箭头后会抓取 EditText 中的数据，然后打开此网址，并在 WebView 中显示网页内容。具体实现代码如下。

```
package irdc.wang;
import irdc.wang.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.webkit.WebView;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      mImageButton1 = (ImageButton)findViewById(R.id.myImageButton1);
      mEditText1 = (EditText)findViewById(R.id.myEditText1);
      mWebView1 = (WebView) findViewById(R.id.myWebView1);
      /*当单击箭头后*/
      mImageButton1.setOnClickListener(new ImageButton.OnClickListener()
      {
         @Override
         public void onClick(View arg0)
         {
               mImageButton1.setImageResource(R.drawable.go_2);
               /*抓取 EditText 中的数据*/
               String strURI = (mEditText1.getText().toString());
               /*    WebView 显示网页内容    */
               mWebView1.loadUrl(strURI);
```

```
                    Toast.makeText(
                            example2.this,getString(R.string.load)+strURI,
                                    Toast.LENGTH_LONG)
                            .show();
                }
            }
        });
    }
}
```

执行上述代码后会显示一个文本框，在文本框中可以输入一个网页地址，当单击后面的按钮后会显示此网页的内容。例如输入http://www.sohu.com后的效果如图 7-5 所示。

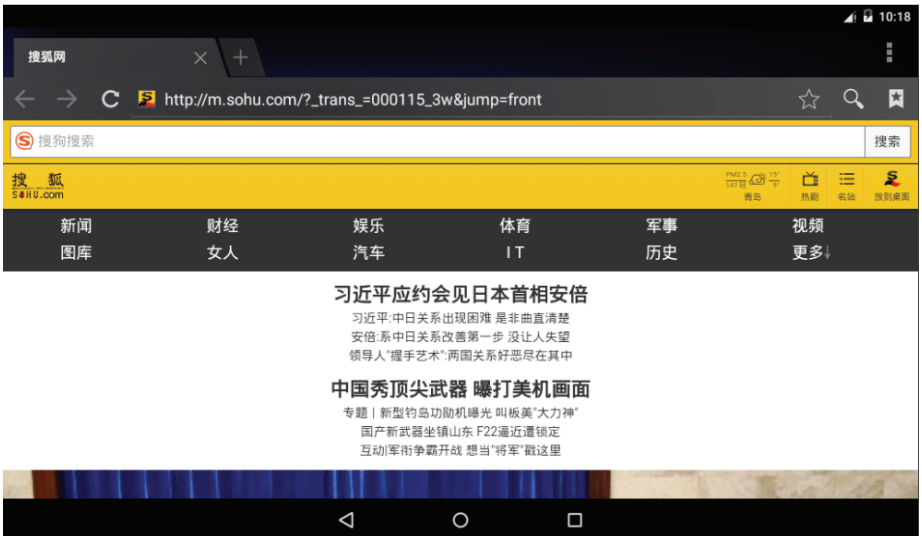


图 7-5　来到搜狐网主页

第三篇　技术提高篇

第 8 章　开发移动网页

随着移动设备的发展和普及，移动 Web 项目开发将成为现阶段一个新兴的热点，所以开发能在 Android 手机上浏览的网页是很有必要的。本书前面所讲解的 HTML、CSS、JavaScript 技术都是网页开发技术，用这三种技术开发的网页能够在手机屏幕上正常浏览吗？答案是肯定的，但是前提是需要进行一些变动。在本章的内容中，将详细讲解为 Android 系统开发移动网页的基本知识。

8.1　第一段 Android 网页代码

接下来将以一个具体的例子作为开始，本实例使用 HTML 和 CSS 技术实现了 4 个简单的网页开发。

实　例	功　能	源 码 路 径
实例 8-1	编写一个适用于 Android 系统的网页	daima\8\first\

8.1.1　编写 HTML 文件

其中主页文件 index.html 的源代码如下。

```
<html>
    <head>
        <title>aaa</title>
        <link rel="stylesheet" href="desktop.css" type="text/css" />
    <body>
        <div id="container">
            <div id="header">
                <h1><a href="./">AAAA</a></h1>
                <div id="utility">
                    <ul>
                        <li><a href="about.html">关于我们</a></li>
                        <li><a href="blog.html">博客</a></li>
                        <li><a href="contact.html">联系我们</a></li>
                    </ul>
                </div>
                <div id="nav">
                    <ul>
                        <li><a href="bbb.html">Android 之家</a></li>
                        <li><a href="ccc.html">电话支持</a></li>
```

```
                        <li><a href="ddd.html">在线客服</a></li>
                        <li><a href="http://www.aaa.com">在线视频</a></li>
                    </ul>
                </div>
            </div>
            <div id="content">
                <h2>About</h2>
                <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为的，
希望事实如此....</p>
            </div>
            <div id="sidebar">
                <img alt="好图片" src="aaa.png">
                <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也这么认为的，
希望事实如此....</p>
            </div>
            <div id="footer">
                <ul>
                    <li><a href="bbb.html">Services</a></li>
                    <li><a href="ccc.html">About</a></li>
                    <li><a href="ddd.html">Blog</a></li>
                </ul>
                <p class="subtle">巅峰卓越</p>
            </div>
        </div>
    </body>
</html>
```

根据“样式和表现相分离”的原则，需要单独写一个 CSS 文件，通过这个 CSS 文件来修饰上述网页的样式。

注意：在现实的应用开发中，最好将桌面浏览器的样式表和 Android 样式表划清界限。编者建议编写两个完全独立的文件。当然还有另一种做法是把所有的 CSS 规则放到一个单一的样式表中，但是这种做法的缺点如下。

- 文件太长了就显得麻烦，不利于维护。
- 把太多不相关的桌面样式规则发送到手机上，会浪费网络带宽和存储空间。

8.1.2 编写 CSS 文件

开始编写 CSS 文件，为了适应 Android 系统，编写下面的 link 标签。

```
<link rel="stylesheet" type="text/css"
  href="android.css" media="only screen and (max-width: 480px)" />
<link rel="stylesheet" type="text/css"
  href="desktop.css" media="screen and (min-width: 481px)" />
```

在上述代码中，最明显的变动是浏览器宽度的变化，即：

```
max-width: 480px
```

```
        min-width: 481px
```

这是因为手机屏幕和计算机屏幕的宽度是不一样的（当然长度也不一样，但是都具有下拉功能），480 像素是 Android 系统的标准宽度，上述代码的功能是不管浏览器的窗口是多大，桌面用户看到的都是文件 desktop.css 中样式修饰的页面，宽度都是用如下的代码来设置的。

```
        max-width: 480px
        min-width: 481px
```

在本实例中有如下两个 CSS 文件。

- 文件 desktop.css：在开发计算机页面时编写的样式文件，是为 HTML 页面服务的。
- 文件 Android.css：是一个新文件，通过这个文件可以将网页显示在 Android 手机中。当开发者开发出完整的 Android.css 文件后，可以直接在 HTML 文件中将如下代码删除，即不再用这段代码修饰文件了。

```
<link rel="stylesheet" type="text/css"
 href="desktop.css" media="screen and (min-width: 481px)" />
```

此时用 Chrome 浏览器打开修改后的 HTML 文件，无论从 Android 手机浏览器还是计算机桌面浏览器中打开，执行后都将得到一个完整的页面展示。此时的完整代码如下。

```
        <html>
            <head>
                <title>AAAA</title>
                <link rel="stylesheet" type="text/css" href="android.css" media="only screen and (max-width:
480px)" />
                <link rel="stylesheet" type="text/css" href="desktop.css" media="screen and (min-width:
481px)" />
                <!--[if IE]>
                    <link rel="stylesheet" type="text/css" href="explorer.css" media="all" />
                <![endif]-->
                <script type="text/javascript" src="jquery.js"></script>
                <script type="text/javascript" src="android.js"></script>
            <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
            </head>
            <body>
                <div id="container">
                  <div id="header">
                        <h1><a href="./">AAAA</a></h1>
                        <div id="utility">
                            <ul>
                                <li><a href="about.html">关于我们</a></li>
                                <li><a href="blog.html">博客</a></li>
                                <li><a href="contact.html">联系我们</a></li>
```

```
                    </ul>
                </div>
                <div id="nav">
                    <ul>
                        <li><a href="bbb.html">Android 之家</a></li>
                        <li><a href="ccc.html">电话支持</a></li>
                        <li><a href="ddd.html">在线客服</a></li>
                        <li><a href="http://www.aaa.com">在线视频</a></li>
                    </ul>
                </div>
            </div>
            <div id="content">
                <h2>About</h2>
                <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为的，
希望事实如此....</p>
            </div>
            <div id="sidebar">
                <img alt="好图片" src="aaa.png">
                <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为的，
希望事实如此....</p>
            </div>
            <div id="footer">
                <ul>
                    <li><a href="bbb.html">Services</a></li>
                    <li><a href="ccc.html">About</a></li>
                    <li><a href="ddd.html">Blog</a></li>
                </ul>
                <p class="subtle">巅峰卓越</p>
            </div>
        </div>
    </body>
</html>
</html>
```

而文件 desktop.css 的代码如下。

```
For example:
body {
    margin:0;
    padding:0;
    font: 75% "Lucida Grande", "Trebuchet MS", Verdana, sans-serif;
}
```

执行效果如图 8-1 所示。

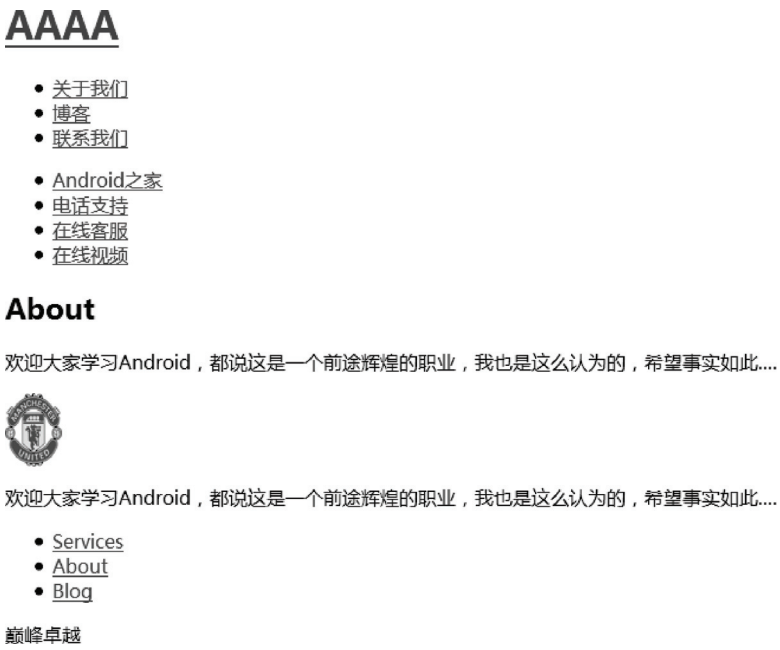


图 8-1　执行效果

8.1.3　控制页面的缩放

除非明确告知 Android 浏览器，否则它会认为页面宽度是 980px。当然这在大多数情况下能工作得很好，因为计算机已经适应了这个宽度。但是如果针对小尺寸屏幕的 Android 手机，则必须做一些调整，必须在 HTML 文件的 head 元素中添加 viewport 的元标签，确保移动浏览器获取屏幕的大小，代码如下。

```
<meta name="viewport" content="user-scalable=no, width=device-width" />
```

这样就实现了屏幕的自动缩放功能，可以根据屏幕的大小显示大小不同的页面。无须担心加上 viewport 元标签后在计算机上的显示效果，因为桌面浏览器会忽略 wiewport 元标签。

如果不设置 viewport 的宽度，页面在加载后会缩小。缩放后的大小是不确定的，因为 Android 浏览器的设置项允许用户设置默认的缩放大小，选项有大、中（默认）、小。即使设置过 viewport 宽度，这个设置项也会影响页面的缩放大小。

8.2　为 Android 中的网页添加 CSS 样式

接着上一节的演示代码继续讲解，开始编写样式文件 android.css，此文件的功能是使网页在 Android 手机上完整并优美地显示。

8.2.1　编写基本的样式

这里的基本样式是指诸如背景颜色、字体大小、字体颜色等样式。接下来以上一节中的实例为基础进行功能扩展，具体实现流程如下。

（1）在文件 android.css 中设置<body>元素的如下基本样式。

```
body {
    background-color: #ddd;        /* 背景颜色 r */
    color: #222;                   /* 字体颜色 */
    font-family: Helvetica;        /* 字体 */
    font-size: 14px;               /* 字体大小 */
    margin: 0;                     /* 外边距 */
    padding: 0;                    /* 内边距 */
}
```

（2）开始处理<header>中的 内容，包含主要入口的链接（也就是 logo）和一级、二级站点导航。第一步是把 logo 链接的格式调整为可以点击的标题栏，将下面的代码加入到文件 android.css 中。

```
#header h1 {
    margin: 0;
    padding: 0;
}
#header h1 a {
    background-color: #ccc;
    border-bottom: 1px solid #666;
    color: #222;
    display: block;
    font-size: 20px;
    font-weight: bold;
    padding: 10px 0;
    text-align: center;
    text-decoration: none;
}
```

（3）用同样的方式格式化一级和二级导航的<ul>元素。在此只需用通用的标签选择器（也就是#header ul）即可，而不必再设置标签<ID>，也就不必设置如下的样式。

- #header ul。
- #utility。
- #header ul。
- #nav。

实现此过程的代码如下。

```
#header ul {
    list-style: none;
    margin: 10px;
    padding: 0;
}
#header ul li a {
```

```
        background-color: #FFFFFF;
        border: 1px solid #999999;
        color: #222222;
        display: block;
        font-size: 17px;
        font-weight: bold;
        margin-bottom: -1px;
        padding: 12px 10px;
        text-decoration: none;
    }
```

（4）为 content 和 sidebar div 添加点内边距，在文字和屏幕边缘之间空出一定距离。具体代码如下。

```
    #content, #sidebar {
        padding: 10px;
    }
```

（5）接下来设置<footer>中内容的样式，<footer>里面的内容比较简单，只需将 display 设置为 none 即可。具体代码如下。

```
    #footer {
        display: none;
    }
```

上述代码在计算机中执行的效果如图 8-2 所示。

图 8-2　计算机中的执行效果

在 Android 模拟器中的执行效果如图 8-3 所示。

图 8-3　在 Android 模拟器中的执行效果

因为添加了自动缩放，并且添加了修饰 Menu 的样式，所以整个界面很协调。

8.2.2　添加视觉效果

为了使页面变得更加精彩，可以尝试加一些充满视觉效果的样式。

（1）给<header>文字加 lpx 向下的白色阴影，背景加上 CSS 渐变效果。具体代码如下。

```
#header h1 a {
    text-shadow: 0px 1px 1px #fff;
    background-image: -webkit-gradient(linear, left top, left bottom, from(#ccc), to(#999));
}
```

对于上述代码有两点说明。

- text-shadow：参数从左到右分别表示水平偏移、垂直偏移、模糊效果和颜色。在大多数情况下，可以将文字设置成上面代码中的数值。在大部分浏览器上，将模糊范围设置为 0px 也能看到效果。但 Andorid 要求模糊范围最少是 lpx，如果设置成 0px，则在 Android 设备上将显示不出文字阴影。
- -webkit-gradient：功能是让浏览器在运行时产生一张渐变的图片。因此，可以把 CSS 渐变功能用在任何指定图片（比如背景图片或者列表式图片）URL 的地方。参数从左到右的排列顺序分别是：渐变类型（可以是 linear 或者 radial）、渐变起点（可以是 left top、left bottom、right top 或者 right bottom）、渐变终点、起点颜色、终点颜色。

在此需要注意，在上述赋值中不能颠倒描述渐变起点、终点常量（left top、left bottom、right top. right bottom）的水平和垂直顺序。也就是说 top left、bottom left、top right 和 bottom right 是不合法的值。

（2）给导航菜单加上圆角样式，代码如下。

```
#header ul li:first-child a {
    -webkit-border-top-left-radius: 8px;
    -webkit-border-top-right-radius: 8px;
}
#header ul li:last-child a {
    -webkit-border-bottom-left-radius: 8px;
    -webkit-border-bottom-right-radius: 8px;
}
```

在上述代码中，使用属性“-webkit-border- radius”描述角的方式，定义了列表中第一个元素的上两个角和最后一个元素的下两个角为以 8 像素为半径的圆角。此时在 Android 模拟器中的执行效果如图 8-4 所示。

图 8-4　在 Android 中的执行效果

此时会发现列表显示样式变为了圆角样式，整个外观显得更加圆滑和自然。

8.3　为 Android 网页添加 JavaScript 特效

经过前面的步骤，一个基本的 HTML 页面就设计完成了，并且这个页面可以在 Android 手机上完美显示。为了使页面更加完美，在接下来的内容中，将详细讲解在上述页面中添加 JavaScript 行为特效的基本知识。

8.3.1　jQuery 框架介绍

jQuery 是继 prototype 之后又一个优秀的 JavaScript 框架。jQuery 是一个轻量级的 JS 库，压缩后只有 21kB 的大小。jQuery 不但兼容 CSS3，而且还兼容各种浏览器。jQuery 不但使用户可以更方便地处理 HTML Documents、Events 和动画元素，并且方便地为网站提供 Ajax（异步 JavaScript 和 XML）交互。jQuery 能够使用户的 HTML 页面保持代码和内容相分离，即不用在 HTML 里面插入一堆 JS 来调用命令，只需定义 ID 即可。

1．语法

jQuery 是为 HTML 元素的选取编制的，通过 jQuery 可以对 HTML 元素执行某些操作。使用 jQuery 的基础语法格式如下。

```
$(selector).action()
```

- 美元符号：定义 jQuery。
- 选择符（selector）：“查询”和“查找” HTML 元素。
- action()：执行对元素的操作动作。

例如下面的代码。

```
$(this).hide()          //隐藏当前元素
$("p").hide()           //隐藏所有段落
$("p.test").hide()      //隐藏所有 class="test" 的段落
$("#test").hide()       //隐藏所有 id="test" 的元素
```

2．简单实用

通过如下代码来认识 jQuery 的强大功能。

```
<html>
<head>
<script type="text/javascript" src="/jquery/jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
   $("button").click(function(){
      $("#test").hide();
   });
});
</script>
</head>

<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p id="test">This is another paragraph.</p>
<button type="button">Click me</button>
</body>

</html>
```

上述代码演示了 jQuery 中函数 hide()的基本用法，功能是隐藏当前的 HTML 元素。执行效果如图 8-5 所示，只显示一个按钮。单击这个按钮后，会隐藏所有的 HTML 元素，包括这个按钮，此时页面一片空白。

图 8-5　未被隐藏时

注意：本书的重点不是 jQuery，所以不再对其使用知识进行讲解。读者可以参阅其他书籍或网上教程来学习。

8.3.2　使网页支持动态行为

本小节继续以前面的实例 8-1 为基础，下面的步骤是给页面添加一些 JavaScript 元素，使页面支持一些基本的动态行为。在具体实现的时候，使用了前面介绍的 jQuery 框架。本实例的目的是，让用户控制是否显示页面顶部的导航栏。实现上述功能的具体流程如下。

（1）隐藏<header>中的 ul 元素，它在用户第一次加载页面后就不会显示。具体代码如下。

```
#header u1．hide{
display：none;
)
```

（2）定义显示和隐藏菜单的按钮，具体代码如下。

```
<div class=”leftButton” onclick=“toggleMenu() “>Menu< / div>
```

定义一个带有 leftButton 类的 div 元素，然后将其放在 header 里面。下面是完整 CSS 样式的代码。

```
#header div.leftButton {
    position: absolute;
    top: 7px;
    left: 6px;
    height: 30px;
    font-weight: bold;
    text-align: center;
    color: white;
    text-shadow: rgba (0,0,0,0.6) 0px -1px 1px;
    line-height: 28px;
    border-width: 0 8px 0 8px;
    -webkit-border-image: url(images/button.png) 0 8 0 8;
}
```

上述代码的具体说明如下。

- “position: absolute”：从顶部开始，设置 position 为 absolute，相当于把这个 div 元素从 HTML 文件流中去掉，从而可以设置自己的最上面和最左面的坐标。
- “height: 30px”：设置高度为 30px。
- “font-weight: bold”：定义文字格式为粗体，白色带有一点向下的阴影，在元素里居中显示。
- “text-shadow: rgba”：rgb(255，255，255)、rgb(100%，100%，l0096)格式和#FFFFFF 格式是一个原理，都是设置颜色值的。在 rgba()函数中，它的第 4 个参数用来定义 alpha 值（透明度），取值范围从 0 到 1。其中 0 表示完全透明，1 表示完全不透明，0 到 1 之间的小数表示不同程度的半透明。

- line-height：控制元素中的文字向下移动的距离，使之不会与上边框齐平。
- border-width 和-webkit-border-image：这两个属性一起决定把一张图片的一部分放入某一元素的边框中去。如果元素大小由于文字的增减而改变，图片会自动拉伸适应这样的变化。这一点意味着只需要不多的图片、少量的工作、低带宽和更少的加载时间。
- border-width：让浏览器把元素的边框定位在距上 0px、距右 8px、距下 0px、距左 8px 的地方（4 个参数从上开始，以顺时针方向为顺序），不需要指定边框的颜色和样式。定义好边框宽度之后，就可以确定放进去的图片了。
- url(images/button.png) 0 8 0 8：5 个参数从左到右分别是图片的 URL、上边距、右边距、下边距、左边距（再一次，从上且顺时针开始）。URL 可以是绝对路径或者相对路径，后者是相对于样式表所在的位置，而不是引用样式表的 HTML 页面的位置。

（3）在 HTML 文件中插入 JavaScript 的代码，将对 aaa.js 和 bbb.js 的引用写到 HTML 文件中。

```
<script type="text/javascript" src="aaa.js"></script>
<script type="text/javascript" src="bbb.js"></script>
```

文件 bbb.js 的主要作用是让用户显示或者隐藏 nav 菜单，具体代码如下。

```
1  if (window.innerWidth && window.innerWidth <= 480) {
2      $(document).ready(function(){
3          $('#header ul').addClass('hide');
4          $('#header').append('<div class="leftButton" onclick="toggleMenu()">Menu</div>');
5      });
6      function toggleMenu() {
7          $('#header ul').toggleClass('hide');
8          $('#header .leftButton').toggleClass('pressed');
9      }
10 }
```

对上述代码的具体说明如下。

第 1 行：括号中的代码，表示当 window 对象的 innerWidth 属性存在并且 innerWidth 小于等于 480px（这是大部分手机合理的最大宽度值）时才执行函数内部。这一行保证只有当用户用 Android 手机或者类似屏幕大小的设备访问这个页面时，才会执行上述代码。

第 2 行：使用函数 document ready，此函数是“网页加载完成”函数。这段代码的功能是当网页加载完成之后才运行里面的代码。

第 3 行：使用了典型的 jQuery 代码，目的是选择 header 中的 ul 元素并且往其中添加 hide 类。此处的 hide 内容是前面 CSS 文件中的选择器，这行代码的执行效果是隐藏 header 的 ul 元素。

第 4 行：为 header 添加按钮，目的是可以“显示/隐藏”菜单。

第 7 行：在函数 toggleMenu()中，用 jQuery 的 toggleClass()函数添加或删除所选择对象中的某个类。这里应用了 header 的 ul 里的 hide 类。

第 8 行：在 header 的 leftButton 里添加或删除 pressed 类，类 pressed 的具体代码如下。

```
#header div.pressed {
    -webkit-border-image: url(images/button_clicked.png) 0 8 0 8;
}
```

通过上述样式和 JavaScript 行为设置之后，Menu 会默认隐藏链接内容，单击之后才会在下方显示链接信息，如图 8-6 所示。

图 8-6　下方显示信息

8.4　在 Android 网页中使用 Ajax 特效

Ajax 是指异步 JavaScript 和 XML 技术，是 Asynchronous JavaScript and XML 的缩写。Ajax 不是一种新的编程语言，而是一种用于创建更好、更快以及交互性更强的 Web 应用程序的技术。通过使用 Ajax 和 JavaScript，可以将 XMLHttpRequest 对象直接与服务器进行通信。通过这个对象，JavaScript 可以在不重载页面的情况下与 Web 服务器进行数据交换。Ajax 在浏览器与 Web 服务器之间使用异步数据传输（HTTP 请求），这样就可以使网页从服务器中请求少量的信息，而不是整个页面。

在本节的内容中，将详细讲解在 Android 系统中为网页添加 Ajax 特效的基本知识。

在接下来的内容中，将通过一个具体实例来讲解在 Android 网页中使用 Ajax 技术的过程。

实　例	功　能	源 码 路 径
实例 8-2	在 Android 系统中开发一个 Ajax 网页	daima\8\gaoji\

（1）编写一个名为 android.html 的 HTML 文件，具体代码如下。

```
<html>
    <head>
        <title>Jonathan Stark</title>
```

```
        <meta name="viewport" content="user-scalable=no, width=device-width" />
        <link rel="stylesheet" href="android.css" type="text/css" media="screen" />
        <script type="text/javascript" src="jquery.js"></script>
        <script type="text/javascript" src="android.js"></script>
    </head>
    <body>
        <div id="header"><h1>AAA</h1></div>
        <div id="container"></div>
    </body>
</html>
```

（2）编写样式文件 android.css，主要代码如下。

```
body {
    background-color: #ddd;
    color: #222;
    font-family: Helvetica;
    font-size: 14px;
    margin: 0;
    padding: 0;
}
#header {
    background-color: #ccc;
    background-image: -webkit-gradient(linear, left top, left bottom, from(#ccc), to(#999));
    border-color: #666;
    border-style: solid;
    border-width: 0 0 1px 0;
}
#header h1 {
    color: #222;
    font-size: 20px;
    font-weight: bold;
    margin: 0 auto;
    padding: 10px 0;
    text-align: center;
    text-shadow: 0px 1px 1px #fff;
    max-width: 160px;
    overflow: hidden;
    white-space: nowrap;
    text-overflow: ellipsis;
}
ul {
    list-style: none;
    margin: 10px;
    padding: 0;
}
ul li a {
```

```
            background-color: #FFF;
            border: 1px solid #999;
            color: #222;
            display: block;
            font-size: 17px;
            font-weight: bold;
            margin-bottom: -1px;
            padding: 12px 10px;
            text-decoration: none;
        }
        ul li:first-child a {
            -webkit-border-top-left-radius: 8px;
            -webkit-border-top-right-radius: 8px;
        }
        ul li:last-child a {
            -webkit-border-bottom-left-radius: 8px;
            -webkit-border-bottom-right-radius: 8px;
        }
        ul li a:active, ul li a:hover {
            background-color: blue;
            color: white;
        }
        #content {
            padding: 10px;
            text-shadow: 0px 1px 1px #fff;
        }
        #content a {
            color: blue;
        }
```

（3）继续编写如下 HTML 文件。

- about.html。
- blog.html。
- contact.html。
- consulting-clinic.html。
- index.html。

为了简单起见，上述文件的实现代码都是一样的，具体代码如下。

```
    <html>
        <head>
            <title>AAA</title>
            <meta name="viewport" content="user-scalable=no, width=device-width" />
            <link rel="stylesheet" type="text/css" href="android.css" media="only screen and (max-width:
480px)" />
            <link rel="stylesheet" type="text/css" href="desktop.css" media="screen and (min-width:
481px)" />
```

```
                <!--[if IE]>
                    <link rel="stylesheet" type="text/css" href="explorer.css" media="all" />
                <![endif]-->
                <script type="text/javascript" src="jquery.js"></script>
                <script type="text/javascript" src="android.js"></script>
            <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
            </head>
            <body>
                <div id="container">
              <div id="header">
                        <h1><a href="./">AAAA</a></h1>
                        <div id="utility">
                            <ul>
                                <li><a href="about.html">AAA</a></li>
                                <li><a href="blog.html">BBB</a></li>
                                <li><a href="contact.html">CCC</a></li>
                            </ul>
                        </div>
                        <div id="nav">
                            <ul>
                                <li><a href="bbb.html">DDD</a></li>
                                <li><a href="ccc.html">EEE</a></li>
                                <li><a href="ddd.html">FFF</a></li>
                                <li><a href="http://www.aaa.com">GGG</a></li>
                            </ul>
                        </div>
                    </div>
                    <div id="content">
                        <h2>About</h2>
                        <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为的，
希望事实如此....</p>
                    </div>
                    <div id="sidebar">
                        <img alt="好图片" src="aaa.png">
                        <p>欢迎大家学习 Android，都说这是一个前途辉煌的职业，我也是这么认为的，
希望事实如此....</p>
                    </div>
                    <div id="footer">
                        <ul>
                            <li><a href="bbb.html">Services</a></li>
                            <li><a href="ccc.html">About</a></li>
                            <li><a href="ddd.html">Blog</a></li>
                        </ul>
                        <p class="subtle">巅峰卓越</p>
                    </div>
                </div>
```

```
    </body>
</html>
```

（4）编写 JavaScript 文件 android.js，在此文件中使用了 Ajax 技术。具体实现代码如下。

```
var hist = [];
var startUrl = 'index.html';
$(document).ready(function(){
    loadPage(startUrl);
});
function loadPage(url) {
    $('body').append('<div id="progress">wait for a moment...</div>');
    scrollTo(0,0);
    if (url == startUrl) {
        var element = ' #header ul';
     } else {
        var element = ' #content';
     }
     $('#container').load(url + element, function(){
        var title = $('h2').html() || '你好!';
        $('h1').html(title);
        $('h2').remove();
        $('.leftButton').remove();
        hist.unshift({'url':url, 'title':title});
        if (hist.length > 1) {
            $('#header').append('<div class="leftButton">'+hist[1].title+'</div>');
            $('#header.leftButton').click(function(e){
                $(e.target).addClass('clicked');
                var thisPage = hist.shift();
                var previousPage = hist.shift();
                loadPage(previousPage.url);
            });
        }
        $('#container a').click(function(e){
            var url = e.target.href;
            if (url.match(/aaa.com/)) {
                e.preventDefault();
                loadPage(url);
            }
        });
        $('#progress').remove();
    });
}
```

对于上述代码的具体说明如下。

- 第 1～5 行：使用了 jQuery 的 document.ready 函数，目的是使浏览器在加载页面完成后运行 loadPage()函数。
- 剩余的行数是函数 loadPage(url)部分，此函数的功能是载入地址为 URL 的网页，但是在载入时使用了 Ajax 技术特效。具体说明如下。
- 第 7 行：为了使 Ajax 效果能够显示出来，在 loadPage()函数启动时，在 body 中增加一个正在加载的 div。
- 第 9～13 行：如果没有在调用函数的时候指定 url（比如第一次在 document ready 函数中调用），url 将会是 undefined，这一行会被执行。第 9 行和第 10 行是 jQuery 的 load()函数样例。load()函数在为页面增加简单快速的 Ajax 特效上非常出色。如果把这一行翻译出来，它的意思是“从 index.html 中找出所有#header 中的 ul 元素，并把它们插入到当前页面的#container 元素中”。当 url 参数有值的时候，执行第 12 行。从效果上看是：“从传给 loadPage()函数的 url 中得到#content 元素，并把它们插入当前页面的#container 元素中”。

（5）最后的修饰

为了能使设计的页面体现出 Ajax 效果，还需要继续设置样式文件 android.css。

- 为了能够显示出“加载中…”的样式，需要在文件 android.css 中添加如下对应的修饰代码。

```
#progress {
    -webkit-border-radius: 10px;
    background-color: rgba(0,0,0,.7);
    color: white;
    font-size: 18px;
    font-weight: bold;
    height: 80px;
    left: 60px;
    line-height: 80px;
    margin: 0 auto;
    position: absolute;
    text-align: center;
    top: 120px;
    width: 200px;
}
```

- 用边框图片修饰返回按钮，并清除默认的点击后高亮显示的效果。在文件 android.css 中添加如下的修饰代码。

```
#header div.leftButton {
    font-weight: bold;
    text-align: center;
    line-height: 28px;
    color: white;
    text-shadow: 0px -1px 1px rgba(0,0,0,0.6);
    position: absolute;
```

```
        top: 7px;
        left: 6px;
        max-width: 50px;
        white-space: nowrap;
        overflow: hidden;
        text-overflow: ellipsis;
        border-width: 0 8px 0 14px;
        -webkit-border-image: url(images/back_button.png) 0 8 0 14;
        -webkit-tap-highlight-color: rgba(0,0,0,0);
    }
```

在 Android 中执行上述文件，在加载时会显示“wait for a moment...”的提示，如图 8-7 所示。在滑动选择某个链接的时候，被选中的按钮会显示不同的颜色，如图 8-8 所示。

图 8-7　提示特效

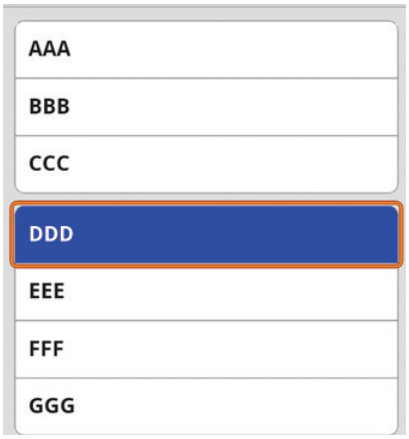


图 8-8　被选择的不同颜色

而文件 android.html 的执行效果和其他文件相比略有不同，如图 8-9 所示。

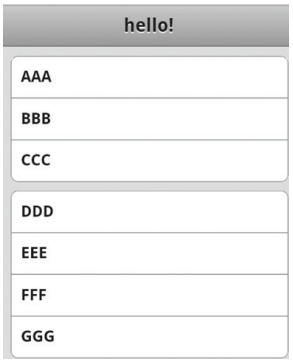


图 8-9　文件 android.html 的执行效果

8.5　使用第三方框架实现动画效果

除了可以为 Android 网页添加 Ajax 特效之外，还可以为其添加动画效果，这样可以让

Android 网页更加充满活力。在本节的内容中，将详细讲解为 Android 网页添加动画效果的基本知识。

8.5.1 一个开源框架——JQTouch

JQTouch 提供了具有一系列功能的 jQuery 插件，这些功能可以为手机浏览器 WebKit 服务。JQTouch 使构建基于 Android 和 iOS 的应用变得更加容易，只需要开发人员掌握少量的 HTML、CSS 和 JavaScript 知识，就能够创建可在 WebKit 浏览器上（iOS、Android、Palm Pre）运行的手机应用程序。

读者可以在 JQTouch 的官方地址 http://www.jqtouch.com 中下载资源，JQTouch 完全开源，下载后可以直接使用。

8.5.2 一个简单应用

接下来将通过一个具体实例的实现过程，详细讲解使用 JQTouch 框架为 Android 网页实现动画效果的过程。

实　例	目　的	源 码 路 径
实例 8-3	在 Android 系统中使用 JQTouch 框架开发网页	daima\8\donghua\

首先编写一个名为 index.html 的 HTML 文件，具体代码如下。

```
<!DOCTYPE html>
<html>
    <head>
        <title>AAA</title>
        <link type="text/css" rel="stylesheet" media="screen" href="jqtouch/jqtouch.css">
        <link type="text/css" rel="stylesheet" media="screen" href="themes/jqt/theme.css">
        <script type="text/javascript" src="jqtouch/jquery.js"></script>
        <script type="text/javascript" src="jqtouch/jqtouch.js"></script>
        <script type="text/javascript">
            var jQT = $.jQTouch({
                icon: 'kilo.png'
            });
        </script>
    </head>
    <body>
        <div id="home">
            <div class="toolbar">
                <h1>Data</h1>
                <a class="button flip" href="#settings">Settings</a>
            </div>
            <ul class="edgetoedge">
                <li class="arrow"><a href="#dates">Dates</a></li>
                <li class="arrow"><a href="#about">About</a></li>
```

```
                </ul>
            </div>
            <div id="about">
                <div class="toolbar">
                    <h1>About</h1>
                    <a class="button back" href="#">Back</a>
                </div>
                <div>
                    <p>Choose you food.</p>
                </div>
            </div>
            <div id="dates">
                <div class="toolbar">
                    <h1>Time</h1>
                    <a class="button back" href="#">Back</a>
                </div>
                <ul class="edgetoedge">
                    <li class="arrow"><a id="0" href="#date">AAA</a></li>
                    <li class="arrow"><a id="1" href="#date">BBB</a></li>
                    <li class="arrow"><a id="2" href="#date">CCC</a></li>
                    <li class="arrow"><a id="3" href="#date">DDD</a></li>
                    <li class="arrow"><a id="4" href="#date">EEE</a></li>
                    <li class="arrow"><a id="5" href="#date">FFF</a></li>
                </ul>
            </div>
            <div id="date">
                <div class="toolbar">
                    <h1>Time</h1>
                    <a class="button back" href="#">Back</a>
                    <a class="button slideup" href="#createEntry">+</a>
                </div>
                <ul class="edgetoedge">
                    <li id="entryTemplate" class="entry" style="display:none">
                        <span class="label">Label</span> <span class="calories">000</span> <span
class="delete">Delete</span>
                    </li>
                </ul>
            </div>
            <div id="createEntry">
                <div class="toolbar">
                    <h1>WHY</h1>
                    <a class="button cancel" href="#">Cancel</a>
                </div>
                <form method="post">
```

```
                    <ul class="rounded">
                        <li><input type="text" placeholder="Food" name="food" id="food"
autocapitalize="off" autocorrect="off" autocomplete="off" /></li>
                        <li><input type="text" placeholder="Calories" name="calories" id="calories"
autocapitalize="off" autocorrect="off" autocomplete="off" /></li>
                        <li><input type="submit" class="submit" name="waction" value="Save Entry" /></li>
                    </ul>
                </form>
            </div>
            <div id="settings">
                <div class="toolbar">
                    <h1>Control</h1>
                    <a class="button cancel" href="#">Cancel</a>
                </div>
                <form method="post">
                    <ul class="rounded">
                        <li><input placeholder="Age" type="text" name="age" id="age" /></li>
                        <li><input placeholder="Weight" type="text" name="weight" id="weight" /></li>
                        <li><input placeholder="Budget" type="text" name="budget" id="budget" /></li>
                        <li><input type="submit" class="submit" name="waction" value="Save
Changes" /></li>
                    </ul>
                </form>
            </div>
        </body>
    </html>
```

接下来开始对上述代码进行详细讲解。

（1）通过如下代码启用 JQTouch 和 jQuery。

```
<script type="text/javascript" src="jqtouch/jquery.js"></script>
<script type="text/javascript" src="jqtouch/jqtouch.js"></script>
```

（2）实现 home 面板，具体代码如下。

```
<div id="home">
    <div class="toolbar">
        <h1>Data</h1>
        <a class="button flip" href="#settings">Settings</a>
    </div>
    <ul class="edgetoedge">
        <li class="arrow"><a href="#dates">Dates</a></li>
        <li class="arrow"><a href="#about">About</a></li>
    </ul>
</div>
```

对应的效果如图 8-10 所示。

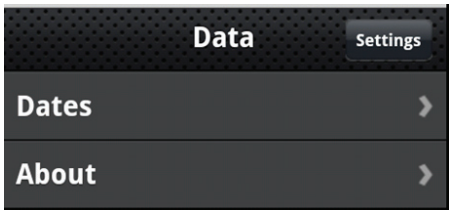


图 8-10　home 面板

（3）实现 about 面板，具体代码如下。

```
<div id="about">
    <div class="toolbar">
        <h1>About</h1>
        <a class="button back" href="#">Back</a>
    </div>
    <div>
        <p>Choose you food.</p>
    </div>
</div>
```

对应的效果如图 8-11 所示。

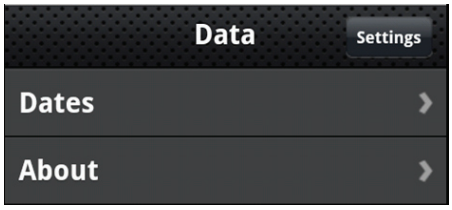


图 8-11　about 面板

（4）实现 dates 面板，具体代码如下。

```
<div id="dates">
    <div class="toolbar">
        <h1>Time</h1>
        <a class="button back" href="#">Back</a>
    </div>
    <ul class="edgetoedge">
        <li class="arrow"><a id="0" href="#date">AAA</a></li>
        <li class="arrow"><a id="1" href="#date">BBB</a></li>
        <li class="arrow"><a id="2" href="#date">CCC</a></li>
        <li class="arrow"><a id="3" href="#date">DDD</a></li>
        <li class="arrow"><a id="4" href="#date">EEE</a></li>
        <li class="arrow"><a id="5" href="#date">FFF</a></li>
    </ul>
</div>
```

对应的效果如图 8-12 所示。

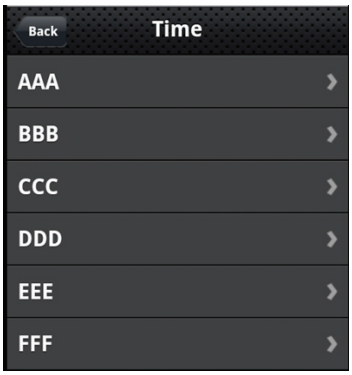


图 8-12　dates 面板

（5）实现 date 面板，具体代码如下。

```
        <div id="date">
            <div class="toolbar">
                <h1>Time</h1>
                <a class="button back" href="#">Back</a>
                <a class="button slideup" href="#createEntry">+</a>
            </div>
            <ul class="edgetoedge">
                <li id="entryTemplate" class="entry" style="display:none">
                    <span class="label">Label</span> <span class="calories">000</span> <span class=
"delete">Delete</span>
                </li>
            </ul>
        </div>
```

（6）实现 settings 面板，具体代码如下。

```
        <div id="settings">
            <div class="toolbar">
                <h1>Control</h1>
                <a class="button cancel" href="#">Cancel</a>
            </div>
            <form method="post">
                <ul class="rounded">
                    <li><input placeholder="Age" type="text" name="age" id="age" /></li>
                    <li><input placeholder="Weight" type="text" name="weight" id="weight" /></li>
                    <li><input placeholder="Budget" type="text" name="budget" id="budget" /></li>
                    <li><input type="submit" class="submit" name="waction" value="Save Changes" /></li>
                </ul>
            </form>
        </div>
```

对应的效果如图 8-13 所示。

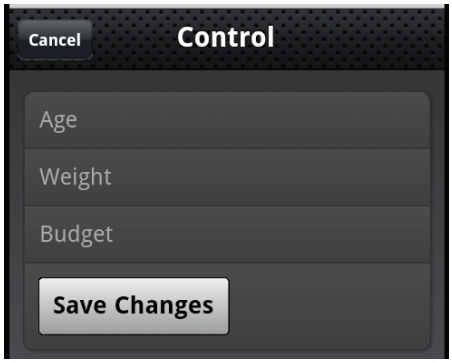


图 8-13　settings 面板

接下来看样式文件 theme.css，此样式文件非常简单，功能是对 index.html 中的元素进行修饰。其实图 8-10、图 8-11、图 8-12 和图 8-13 都是经过 theme.css 修饰之后的显示效果。主要代码如下。

```
body {
    background: #000;
    color: #ddd;
}
#jqt > * {
    background: -webkit-gradient(linear, 0% 0%, 0% 100%, from(#333), to(#5e5e65));
}
#jqt h1, #jqt h2 {
    font: bold 18px "Helvetica Neue", Helvetica;
    text-shadow: rgba(255,255,255,.2) 0 1px 1px;
    color: #000;
    margin: 10px 20px 5px;
}
/* @group Toolbar */
#jqt .toolbar {
    -webkit-box-sizing: border-box;
    border-bottom: 1px solid #000;
    padding: 10px;
    height: 45px;
    background: url(img/toolbar.png) #000000 repeat-x;
    position: relative;
}
#jqt .black-translucent .toolbar {
    margin-top: 20px;
}
#jqt .toolbar > h1 {
    position: absolute;
    overflow: hidden;
```

```
        left: 50%;
        top: 10px;
        line-height: 1em;
        margin: 1px 0 0 -75px;
        height: 40px;
        font-size: 20px;
        width: 150px;
        font-weight: bold;
        text-shadow: rgba(0,0,0,1) 0 -1px 1px;
        text-align: center;
        text-overflow: ellipsis;
        white-space: nowrap;
        color: #fff;
    }
    #jqt.landscape .toolbar > h1 {
        margin-left: -125px;
        width: 250px;
    }
    #jqt .button, #jqt .back, #jqt .cancel, #jqt .add {
        position: absolute;
        overflow: hidden;
        top: 8px;
        right: 10px;
        margin: 0;
        border-width: 0 5px;
        padding: 0 3px;
        width: auto;
        height: 30px;
        line-height: 30px;
        font-family: inherit;
        font-size: 12px;
        font-weight: bold;
        color: #fff;
        text-shadow: rgba(0, 0, 0, 0.5) 0px -1px 0;
        text-overflow: ellipsis;
        text-decoration: none;
        white-space: nowrap;
        background: none;
        -webkit-border-image: url(img/button.png) 0 5 0 5;
    }
    #jqt .button.active, #jqt .cancel.active, #jqt .add.active {
        -webkit-border-image: url(img/button_clicked.png) 0 5 0 5;
        color: #aaa;
    }
    #jqt .blueButton {
        -webkit-border-image: url(img/blueButton.png) 0 5 0 5;
```

```
            border-width: 0 5px;
        }
        #jqt .back {
            left: 6px;
            right: auto;
            padding: 0;
            max-width: 55px;
            border-width: 0 8px 0 14px;
            -webkit-border-image: url(img/back_button.png) 0 8 0 14;
        }
        #jqt .back.active {
            -webkit-border-image: url(img/back_button_clicked.png) 0 8 0 14;
        }
        #jqt .leftButton, #jqt .cancel {
            left: 6px;
            right: auto;
        }
        #jqt .add {
            font-size: 24px;
            line-height: 24px;
            font-weight: bold;
        }
        #jqt .whiteButton,
        #jqt .grayButton, #jqt .redButton, #jqt .blueButton, #jqt .greenButton {
            display: block;
            border-width: 0 12px;
            padding: 10px;
            text-align: center;
            font-size: 20px;
            font-weight: bold;
            text-decoration: inherit;
            color: inherit;
        }

        #jqt .whiteButton.active, #jqt .grayButton.active, #jqt .redButton.active, #jqt .blueButton.active,
#jqt .greenButton.active,
        #jqt .whiteButton:active, #jqt .grayButton:active, #jqt .redButton:active, #jqt .blueButton:active,
#jqt .greenButton:active {
            -webkit-border-image: url(img/activeButton.png) 0 12 0 12;
        }
        #jqt .whiteButton {
            -webkit-border-image: url(img/whiteButton.png) 0 12 0 12;
            text-shadow: rgba(255, 255, 255, 0.7) 0 1px 0;
        }
        #jqt .grayButton {
            -webkit-border-image: url(img/grayButton.png) 0 12 0 12;
```

```
        color: #FFFFFF;
    }
```

到此为止，整个实例全部讲解完毕，此时每一个页面的切换功能都具有了动画效果，如图 8-14 所示。

图 8-14　闪烁的动画效果

因为图片体现不出动画效果，所以建议读者在模拟器上亲自实践体验。

8.6　为网页增加数据存储功能

大多数应用程序都需要某种持久化的方式来存储有用的数据。对于网络应用程序来说，这个任务一般会交给服务器端的数据库或者在浏览器中的 Cookie 来完成。伴随着 HTML 5 的出现，Web 开发者有了另外两种选择：Web Storage 和 Web SQL Database。在本节的内容中，将详细讲解为 Android 网页增加数据存储功能的方法。

8.6.1　在 Android 网页中使用 Web Storage

Web Storage 有两种形式，分别是 localStorage（本地存储）和 sessionStorage（会话存储）。这两种形式都允许开发者使用 JavaScript 设置“键值对”，并在重新加载不同页面的时候读出里面的数据。这点和 Cookie 机制非常类似。但是和 Cookie 不同的是，Web Storage 的数据是完全存储在客户端的，无须通过浏览器请求再传输到服务器。由此可见，和 Cookie 方式相比，Web Storage 可以在本地存储更多的数据。

localStorage 和 sessionStorage 在功能上是一样的，只是在持久性和范围上有所不同。

（1）tocalStorage

即使已经关闭了浏览窗口，数据也会被保存起来并可用于所有来自同源（必须有相同的域名、协议和端口）窗口（或者标签页）的加载，这对于参数设置或是偏好设置之类的功能非常有用。

（2）sessionStorage

数据存储在窗口对象中，对于其他窗口或者标签页来说不可见，并且当窗口关闭时会丢失数据。sessionStorage 可用于特殊的窗口状态，例如一个标签页的高亮状态或者一个表格的

排序状态。在现实应用中，sessionStorage 都可以用 localStorage 来代替。但是在关闭窗口或者标签页时，会丢失 sessionStorage 存储的数据。

设置参数的方法非常简单，例如：

```
localStorage.setItem('age'.40);
```

访问一个存储的数据的方法也非常简单，例如：

```
var age=localStorage.getItem('age');
```

可以用如下代码删除一个特定的键值对。

```
localStorage. removeItem('age');
```

或者用如下代码删除所有的键值对。

```
localStorage.clear();
```

假设键是有效的 JavaScript token，比如没有空格、没有除下画线之外的标点，则可以使用如下的代码实现。

```
localStorage.age=4e                    //设置 age 的值
var age=localStorage.age;              //取得 age 的值
delete localStorage.age;               //从存储中删除 age
```

其实 localStorage 和 sessionStorage 中的键是分开存储的。即使在两种存储中使用同样的键名，也是没有冲突的。

在下面的内容中，将举例说明将用户设置保存到本地和将选中的数据保存到 SessionStorage 中的方法。

1．将用户设置保存到本地

接下来介绍一个实际的例子，目的是更新本章前面例子中的 Settings 面板，此时可以让它把表单数据存储到 localStorage 当中。接下来会有相当数量的 JavaScript 代码，但是并不准备把它们都放进 HTML 文件的 head 标签中。为了保持代码的工整性，在存放 HTML 文档的相同路径下创建一个名为 123.js 的文件，同时更新 HTML 文件中的 head 部分来引用 123.js。

```
<head>
<title>123< / title>
<link type="text/css" rel="stylesheet" media="screen"
href=”jqtouch/jqtouch.css“>
<link type="text/css" rel="stylesheet" media="screen"
href="themes/jqt/theme.css">
<script type="text/javasc ript" src=” "jqtouch/jquery.js"></ script>
<script type="text/javasc ript" src="jqtouch/jqtouch.js"></ script>
<script type="text/javascript" src="123.js"< / script>
</head>
```

由此可见，在更新的同时也从 HTML 的 head 部分删除了 jQTouch 的构造函数。其实它并

没有被删除，只是被移到文件 123.js 中。但是需要确保从 HTML 的 head 部分中删除上述提到的内容，并且在相同路径下建立如下内容的 123.js 文件，然后在浏览器中刷新主 HTML 文件以确保它仍旧可以工作。

```
var jQT=$.jQTouch({
icon:'123.png'
});
```

代码经过重新组织以后，接下来就可以添加保存设置代码。需要重写 settings 表单的提交动作的代码，并用一个名为 saveSettings()的自定义函数来替换。因为用到了 jQTouch，所以只需要修改 Document Ready 函数中如下的一行代码，即将下面的代码加入到 123.js 中。

```
$(document).ready(function(){
$('#settings form') .submit(saveSettings);
```

这段代码的作用是，当用户提交 settings 表单时，用 saveSettings()函数代替表单的提交动作。当调用 saveSettings()函数时，会用 jQTouch 的 val()函数获得表单的 3 个输入项，并且分别存入 localSto rage 这个同名变量中。将函数加入 123.js 中的代码如下。

```
function saveSettings()  {
    localStorage.age=$('#age').val();
    localStorage.budget=$('#budget').val();
    localStorage.weight=$('#weight').val.();
    jQT.goBack();
    return false;
}
```

一旦数值被保存，便调用 jQTouch 的 goBack()函数来关闭面板并返回前一个页面。接下来返回 false 给触发这个函数的提交事件，以防进行默认的提交操作。如果没有这行代码，现有页面会被重新加载一次。

到此为止已经可以运行程序，前往 Settings 面板输入设置信息，然后提交表单将设置存储到 locaIStorage 中。由于在提交表单时没有清空相关字段，所以当用户再次访问 Settings 面板时，仍然会存在上次输入的数据。然而这不是因为使用了 localStorage 保存的缘故，而是因为只要在输入后不进行清理操作，这些字段就会一直在那里。因此，当用户下次重新运行程序并访问 Settings 面板时，即使这些字段被保存了也仍会丢失。

为了解决这个问题，需要使用函数 loadSettings()来加载配置，所以在文件 123.js 中添加如下函数。

```
function loadSettings()  {
    $('#age').val(localStorage.age);
    $('#budget').val(localStorage.budget);
    $('#weight').val(localStorage.weight);
)
```

函数 loadSettings()正好和函数 saveSettings()相反，它使用 local Storage 中的相应字段来调

用 jQTouch 的函数 val()，目的是设置 Settings 表单中的 3 个字段值。

接下来需要触发已有的 loadSettings()函数，当程序启动时进行触发动作。为了实现这个功能，在文件 123.js 的 document ready 函数中添加如下代码。

```
$(document).ready(function(){
    $('#settings form').submit(saveSettings);
    loadSettings();
})
```

上述代码只是在程序启动时加载配置，这样会存在一个漏洞：如果当用户访问 Settings 面板并改动了一些值，然后单击 Cancel 按钮而非提交表单，程序下次就不能加载正确的配置。在这种情况下，当用户再次访问 Settings 面板时会显示最新的修改。这不是因为这些新的数据被保存了（实际上并没有），而只是界面显示的问题。如果用户关闭并重新启动程序，显示的数据又会变成之前保存的数据，因为函数 loadSettings()会在程序启动的时候还原字段中的数据。

其实有很多方法可以改变这种情况，但最合适的方法是每次当移动 Settings 面板时，无论进入还是退出，都需要刷新显示的数据。在 jQTouch 中提供了一种简单的方法将函数 loadSettings()和 Settings 面板的 pageAnimationStart 事件绑定起来。将刚刚加入的代码用如下加粗的代码来代替。

```
$(document).ready(function(){
    $('#settings form').submit(saveSettings);
    $('#settings') .bind('pageAnimationStart',loadSettings);
}) ;
```

现在文件 123.js 中的 JavaScript 代码为 Settings 面板提供了持久化的数据存储。以下是当前文件 123.js 中的代码。

```
var jQT=$.jQTouch({
    icon: '123.png'
    >);
    $(document).ready(function(){
    $('#settings form').submit(saveSettings);
    $('#settings').bind('pageAnimationStart',loadSettings);
});
function loadSettings()  {
    $('#age').val(localStorage.age);
    $('#budget').val(localStorage.budget);
    $('#weight').val(localStorage.weight);
)
function saveSettings()  {
    localStorage.age=$('#age').val();
    localStorage.budget=$('#budget').val();
    localStorage.weight=$('#weight').val();
    jQT.goBack();
```

```
        return false;
    }
```

2．将选中的数据保存到 Session Storage 中

因为希望允许用户在数据库中添加或者删除条目，所以需要支持 Date 面板上提供“+”按钮和 Delete 按钮。在具体实现时，第一步是当用户从 Dates 面板访问 Date 面板时，需要让 Date 面板获知用户单击的是哪个条目。有了这个信息，就可以计算出合适的日期的上下文语境。为了实现这个功能，需要在文件 123.js 的函数 document reday 中添加如下的代码。

```
$(document).ready(function(){
$('#settings form').submit(saveSettings);
$('#settings').bind('pageAnimationStart',loadSettings);
$('#dates li a').click(function(){                    //(1)
var dayOffset=this.id;                                //(2)
var date=new Date();                                  //(3)
date.setDate(date.getDate() - dayOffset);
sessionStorage.currentDate=date.getHonth()+1+'/'+
date.getDate()+'/'+
date.getFullYear();                                   //(4)
refreshEntries();                                     //(5)
});
});
```

（1）jQTouch 中的 click()函数将它随后的 JavaScript 代码和 Dates 面板中的 click 事件的链接绑定起来。

（2）获取被单击对象的 ID，并且将其存入 dayOffset 变量中。Dates 面板中的链接都有从 0～5 范围的 ID，所以被点击的链接的 ID 就相当于点击日期所需要计算的天数，比如，过去 0 天表示今天，过去 1 天表示昨天，过去 2 天表示前天，依此类推。

（3）创建了一个新的 JavaScript 日期对象，并且将其存入一个名为 date 的变量中。首先，这个 date 会被设置成创建的日期，所以在下一行中会从 getDate()函数的结果中减去 dayOffset，再用 setDate()函数将日期设置成选中的日期。其中 dayOffset 为 0 表示今天，1 表示昨天，依此类推。

（4）生成了一个“MM/DD/YYYY”格式的日期字符串，并将其作为当前日期(currentDate)存入 sessionStorage。

（5）调用 refreshEnt ries()函数。此函数的作用是，基于用户单击 Dates 面板的日期值来更新 Date 面板。现在只要使用用户选中的日期来更新 Dates 面板工具栏上的标题，就能看到它起的具体作用。如果没有这个函数，运行后只会出现“Date”字样。函数 refreshEntries()应添加到文件 123.js 中，具体代码如下。

```
function refreshEntries()  {
    var currentDate=sessionStorage.currentDate;
    $('#date   hl').text (currentDate);
  )
```

8.6.2　在 Android 网页中使用 Web SQL Database

在所有 HTML 5 的特性中，Web SQL Database 的功能非常强大。Web SQL Database 提供给开发者一个简单而强大的 JavaScript 数据库 API，使得可以在一个本地 SQLite 数据库中持久保存数据。其实 Web SQL Database 并非是 HTML 5 的一部分，它其实是摆脱了 HTML 5 的细则，并将其融入到自己的细则当中。

开发者可以使用标准 SQL 语句来创建数据表及插入、更新、查找和删除行。JavaScript Database API 甚至能够支持 Transaction（事务）。SQL 有与生俱来的复杂性，但无论如何，这是一个改变游戏规则的变化，所以关注它本身可能会更有意义。在接下来的内容中，将详细讲解使用 Web SQL Database 操作数据库的基本知识。

1．创建数据库

现在用户已经选择好 Date 面板所需的数据了，在编写 createEntry()函数之前，还要建立一个数据库来存储提交的数据（这是个一次性的操作）。可以在文件 123.js 中添加如下代码来实现这项功能。

```
var db;                                                          //(1)
$(dacument).ready(function(){
    $('#settings form') .submit (saveSettings);
    $('#settings').bind('pageAnimationSta rt',loadSettings);
    $('#dates li a').click(functian(){
        var dayOffset=this.id;
        var date=new Date();
        date.setDate(date.getDate() - dayOffset);
        sessionStorage.currentDate=date.getMonth()+1+'/'+
        date．getDate()+'/'+
        date．getFullYear();
        refreshEntries();
    });
        var shortName='123';                                     //(2)
        var version='1.0';
        var displayName='123';
        var maxSize=65536;
        db=openDatabase(shortName,version,displayName,maxSize);  //(3)
        db.transaction(                                          //(4)
        function(transaction){                                   //(5)
            transaction.executeSql(                              //(6)
            'CREATE TABLE IF NOT EXISTS entries '+
            '   (id INTEGER NOT NULL PRIMARY KEY AUTOINCREHENT, '+
            '       date DATE NOT NULL, food TEXT NOT NULL, '+
            'calories INTEGER NOT NULL); '
            );
        }
    );
});
```

（1）首先要注意的是，在程序里有一个名为 db 的全局变量。一旦建立一个数据库连接，这个变量就会保存连接的引用。它之所以被设置成全局变量，是因为在程序的各个地方都将会用到它。

（2）接下来的 4 行代码定义了调用 openDatabase( )需要的参数，具体说明如下。

- shortName：一个指向硬盘上的数据库文件的字符串。
- version：当需要修改数据库模式时，用来管理升级和向后兼容的数字。例如，每次程序启动时检查 version 字段，如果过期则创建一个新的数据库，并且将老数据库中的数据移到新数据库中。
- displayName：表示用户显示的字符串。
- maxSize：允许数据库增长到的最大 KB 数。

（3）当参数被设置以后，这行代码将会调用 openDatabase( )并且将数据库连接存储到 db 变量中。如果数据库不存在，则会新建一个。

（4）所有的数据库查询都必须放在一个事务的上下文当中，所以这里调用了一个 db 对象的 transaction 方法。而接下来的几行会构造一个函数作为唯一的参数传入 transaction 方法。

（5）从本行开始是一个匿名函数，将 transaction 对象当作参数传入。

（6）一旦进入这个函数，调用 transaction 对象的 executeSql()方法来执行一个标准的 CREATE TABLE 查询语句。其中的 IF NOT EXISTS 子句，能够防止已经存在数据表的情况下再次创建数据表的情形。

如果再次启动程序，会在 Android 手机上创建一个名为“123”的数据库。在 Chrome 的桌面版中，实际上是可以浏览并操作客户端的数据库的，只需要导航到 View→Developer→Developer Tools，而后点击 Storage 选项。在 Chrome 桌面版中，Developer Tools 对于调试来说是相当有用的。在默认情况下，这个工具栏会作为浏览器的一个面板出现。如果点击浮动图标，这个面板会作为一个独立的窗口显示。通过点击数据库名，这个界面甚至允许执行任意的 SQL 查询语句。

2．插入行

现在已经有一个配置好的数据库来接收数据条目，可以生成一个 createEntry()函数了。首先，重写#createEntry 表单的提交事件，通过将 createEntry()函数和文件 123.js 中 document ready 函数中的提交事件绑定到一起来达成此目的。下面只显示了前几行，详情参见粗体代码。

```
$(document).ready(function(){
$（'#createEntry form'）.submit(createEntry);
$('#settings form').submit(saveSettings);
$('#settings').bind('pageAnimationStart',loadSettings);
```

这样，每当用户提交#createEntry 表单时就会调用 createEntry()函数。接下来，在文件 123.js 中添加如下的代码，以在数据库中创建记录。

```
function createEntry()  {
    var date=sessionStorage.currentDate;                    //(1)
    var calories=$('#calories') .val();
    var food=$('#food').val();
```

```
        db.transaction(                                          //(2)
          function(transaction){
            transaction.executeSql(
            'INSERT INTO entries (date, calories, food) VALUES(?,?,?);',
             [date, calories, food],
            functiont){
                refreshEntries();
                jQT．goBack();
                ),
                errorHandler
                );
            }
          );
      return false;
    )
```

（1）这部分包含了将会在 SQL 查询语句中使用的一些变量。前面用户在 Dates 面板上点击的日期数据已经保存在 sessionStorage.currentDate 中，使用之前在 Settings 表单中用到的相同方法，可以从表单中获得另外两个值（calories 和 food）。

（2）这段代码打开一个数据库事务，并且执行 executeSql()调用。这时会传入方法 executeSql()中如下的 4 个参数。

- “INSERT INTO entries (date, calories, food) VALUES(?,?,?)”：是即将被执行的语句，问号代表数据占位符。
- “[date,calories, food]”：这个数值数组将被传入数据库当中，和 SQL 语句中的问号占位符一一对应。
- “function() {refreshEnt ries();iQT.goBack();}”：如果 SQL 查询语句成功返回，这个匿名函数将会被调用。
- errorHandler：如果执行 SQL 语句失败，将会执行错误处理函数。

假设插入数据正确，将会执行作为第 3 个参数的匿名函数。这个函数会调用 refreshEnt ries()函数。现在这个函数只能更新 Date 面板的标题，但是之后就会看到创建的条目在列表当中显示。

如果插入不成功则会执行 errorHandler()函数。在文件 123.js 中添加如下代码。

```
    function errorHandler(transaction, error){
        alert('Oops. Error was'+error.message+'    (Code'+error.code+')'):
        return true;
        )
```

上述错误处理函数会传入两个参数：transaction 对象和 error 对象。这里使用 error 对象来提示用户错误信息和抛出的错误码。错误处理函数必须返回 true 或 false。如果一个错误处理函数返回 true，比如这是一个致命的错误，执行过程会被停止并且整个事务将会回滚，如果错误处理函数返回 false，比如这不是一个致命的错误，则执行过程会继续进行。

在有些情况下，可能需要根据不同的错误类型来判断返回值为 true 或 false。其实除了 error

对象以外，函数 errorHandler()接收了一个 transaction 对象。如果没有建立语句当中提到的 errors 表的话，下面的代码将不会被执行。

```
function errorHandler(transaction, error){
    alert('Oops. Error was'+error.message+'    (Code'+error.code+-).);
    t ransaction.executeSql('INSERT INTO errors (code, message) VALUES(7,?);',
    [error.code,error.message]);
    return false;
}
```

如果希望执行 executeSql()函数中的语句，那么错误处理函数就必须返回 false。如果返回的是 true（或者没有返回值），则在包括错误处理在内的整个事务当中的 SQL 语句都将会回滚，这样就达不到希望的效果了。

3．检索行及处理结果集

接下来需要扩展 refreshEntries()函数以完成更多功能，而不只是将选择的日期数据更新到标签栏上。具体来讲，下面将会查询数据库中被选择的日期条目，并使用 HTML 中隐藏的 entryTemplate 把它们添加到#date ul 元素中。在这里把 Date 面板中的代码再次展示出来，它们已经在文件 index.html 中了，所以无需添加。

```
<div id="date">
<div class="toolbar">
<hl>Date< / hl>
<a class="button back" href="#">Back</a>
<a class="button slideup"href="#createEntry">+</a>
</div>
<ul class="edgetoedge">
<li id="ent ryTemplate"class="entry "style="display:none">
<span class="label ">Label</span>
<span class="calories ">OOO</span>
<span class="delete">Delete</span>
</li>
</ul>
</div>
```

注意上面的加粗斜体代码：li 元素的样式属性已经设置成“display:none”，这会使该元素不在页面上显示出来。这样做是因为使用了 HTML 代码段作为数据库行显示的模板。

下面是完整的 refreshEntries()函数，需要将文件 123.js 中现有的函数 refreshEntries()的代码替换成如下的代码。

```
function refreshEntries()   {
    var currentDate=sessionStorage.currentDate;                    //(1)
    $('#date hl').text(currentDate);
    $('#date ul ti:gt(0) ').remove();                              //(2)
    db.transaction(                                                //(3)
    function(transaction)   {
```

```
                transaction.executeSql(
                'SELECT 木 FROM entries WHERE date=7 0RDER BY food;',          //(4)
                 [currentDate],                                                //(5)
                function (transaction.result){                                 //(6)
                    for (var i=0;i<result.rows.length; i++){
                        var    row=result.rows.item(i);                        //(7)
                        var newEntryRow=$('#ent ryTemplate').clone();          //(8)
                        newEnt ryRow. removeAttr('id');
                        newEntryRaw.removeAttr('style');
                        newEnt ryRow.data('entryId'.row.id);                   //(9)
                        newEnt ryRow.appendTo('#date uI');                     //(10)
                        newEnt ryRow.find('*label').text(row.food);
                        newEnt ryRow.find('.calories').text(row.calories);
                    }
            ),
            errorHandler
            )
            );
            )
            );
        }
```

（1）使用 sessionStorage 中保存的 currentDate 值来设置 Date 面板中的标题栏。

（2）使用 jQTouch 的 gt()函数（gt 代表“greater than”）来查找并移除任何索引大于 0 的 li 元素。

（3）这 3 行用于设置数据库事务和 executeSql 语句。

（4）本行包含了 executeSql 函数的第一个参数，这是一个简单的 SELECT 语句，问号代表一个数据占位符。

（5）这是只有一个元素的数组，包含了现有被检索到的数据。在 SQL 查询语句中，问号将会被该元素代替。

（6）当完成正确的查询工作时会调用这个匿名函数，会接收到两个参数：transaction 和 result。

（7）本行使用 rows 的 item()方法把查询到的行的内容设置到 row 变量中。

（8）在这行代码中，使用 clone()方法来复制模板 li，并且在下两行中删除掉它的 id 和 style 属性。删除掉 style 属性会使这行可见，而删除掉 id 是很重要的，否则在页面的结尾中就会出现有多个相同 id 的数据项。

（9）本行中将 row 的 id 属性作为数据存入 li 元素中，需要这个数据是因为在删除的时候知道这是哪条记录。

（10）这行代码将 li 元素加入其父元素 uL 当中。接下来的两行用 row 中相应的数据更新 li 的子元素 label 和 calories。

经过上述操作处理后，Date 面板会操作从数据库中检索出来的数据，设置每一行显示一个 li 元素，每一行都有一个 lable、calories 和 Delete 按钮。如果再创建一些行，就需要加上一些 CSS，让界面变得更美观。所以将下面的 CSS 代码保存到文件 123.css 中，并且需要将这

个文件放在和 HTML 同一层目录下。

```
        #date ut li{
            position:relative;
            }
            #date ul li span{
                color：#FFFFFF;
                text-shadow:0 lpx 2px rgba(0,0,0,.7);
                }
#date ul li.delete {
    position: absolute;
    top: 5px;
    right: 6px;
    font-size: 12px;
   line-height: 30px;
    padding: 0 3px;
    border-width: 0 5px;
    -webkit-barder-image: url(themes/jqt/img/button.png) 0 5 0 5;
}
```

然后在文件 index.html 的 head 部分添加如下代码以便连接到文件 123.css。

```
<link type="text/css"rel="stylesheet"media="sc reen"href="123.css">
```

尽管现在 Delete 按钮看起来更像一个按钮了，但是点击这些按钮时却没有任何反应。这是因为它是使用〈span〉标签来实现的按钮，这在 HTML 页面中是一个无法交互的元素。

4．删除行

为了让 Delete 按钮起作用，需要使用 jQuery 将它们和点击事件绑定起来。与 Dates 面板中的条目不同，Date 面板中的条目不是静态的，这说明它们的添加和删除跟用户的 session 相关。其实当启动程序时，Date 面板上并没有可见的条目，因此也没有什么可以绑定到 click 事件上。

解决方案是当 Delete 按钮在 refreshEntries()函数被创建时，将它们绑定到 click 事件上。为了做到这点，需要在 for 循环的最后添加如下加粗的代码：

```
newEntryRow.find('.calories').text( row.calories);
newEntryRow.find('.delete').click(function(){                    //(1)
    var clickedEntry=$(this).parent();                           //(2)
    var clickedEntryId=clickedEntry.data('entryId');             //(3)
    deleteEntryById(clickedEntryId);                             //(4)
    clickedEntry.slideUp();
});
```

（1）这个函数只在 date 元素的 ID 中寻找任何有 delete 这个类的元素，并设置它们的 click()函数。这个 click()函数允许用一个匿名函数作为参数来处理点击事件。

（2）当 click 事件被触发以后，Delete 按钮的父元素（这里是 li）被找到，然后被保存到 clickedEntry 变量中。

（3）这行代码使用函数 refreshEntries()创建的 li 元素保存的 entryId 值来设置 clickedEntryId 变量。

（4）这行代码将被点击元素的 ID 传入 deleteEntryById()函数中，接下来的一行代码会调用 jQuery 的 slideUp()函数将 li 元素从界面上移除。

在文件 123.js 中添加 deleteEnt ryById()函数，功能是从数据库中移除一条记录。

```
function deleteEntryById(id)   {
      db.transaction(
      function(transaction){
            transaction.executeSql('DELETE FROM entries WHERE id=?;'
            [id],    null, errorHandler);
      }}
      );
}
```

然后打开一个事务，并将一个参数为 transaction 对象的回调函数作为参数传入 transaction 中，然后执行 executeSql()方法。方法中的 SQL 查询语句和被点击记录的 ID 作为两个参数传入。第 3 个参数是成功处理函数，因为不需要，所以被指定为 null。再看第 4 个参数，将其指定为一直使用的错误处理函数。虽然用了很多描述才完成这项功能，但实际上代码并不多。

在演示文件 123.中，与数据库交互操作相关的全部 JavaScript 代码如下。

```
var jQT=$.jQTouch(t
      icon:'123.png'
   });
   var db;
   $(document).ready(function(){
      $('#createEntry form') .submit(createEntry);
      $('#settings form').submit( saveSettings);
      $('#settings').bind('pageAnimationStart',loadSettings);
      $('#dates li a').click(function(){
      var dayOffset=this.id;
      var date=new Date();
      date.setDate(date.getDate()-dayOffset);
      sessionStorage.currentDate=date.getMonth()+l+'/'+
                                          date.getDate() + '/l +
                                 date.getFullYear(); .
      refreshEntries();
      });
        var shortName = '123';
        var version = '1.0';
        var displayName = '123';
          var maxSize = 65536;
          db = openDatabase(shortName, version, displayName, maxSize);
          db.transaction(
            function(transaction) {
                transaction.executeSql(
```

```
                    'CREATE TABLE IF NOT EXISTS entries. +
                    '    (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, ' +
                    '    date DATE NOT NULL, food TEXT NOT NULL, . +
                                .        calories INTEGER NOT NULL) ; '
                    );
                }
            );
    });
    function loadSettings() t
            $( '#age').val(localStorage.age) ;
            $( '#budget').val(localStorage.budget) ;
            $( '#weight').val(localStorage.weight) ;
      }
    function saveSettings() {
            localStorage.age = $('#age') .val() ;
            localStorage.budget = $( '#budget') .val() ;
            localStorage.weight =   $( '#weight') .val() ;
            iQT.goBack() ;
        return false,
      }
    function createEntry(){
         var date=sessionStorage.currentDate .
         vat calories = $( '#calories').val();
         var food = $('#food').val();
         db.transaction(
          function(transaction) {
              transaction.executeSql(
                    'INSERT INTO entries(date, calories, food)VALUES(?, ?, ?);',
                  [date, calories, foodl,
    function(){
              refreshEnt ries () ;
              jQT.goBack() ;
            },
  errorHandler
                        );
                }
            );
       return false,
      )
  function refreshEntries(){
          var currentDate=sessionStorage. currentDate ,
          $( '#date   hi') .text (currentDate) ;
          $( '#date   ul   li:gt (0) ' ) . remove () ;
db.transaction(
        fulclction(transaction) {
            transaction. executeSql(
```

```
                    'SELECT* FROM entries WHERE date = ? ORDER BY food;',
                                        [currentDate],
                function (transaction, result){
                    for(var i=o; i<result.rows.length; i++){
                                var row:result.rows.item(i);
                                var newEntryRow=$( '#entryTemplate') .clone()
                                newEnt ryRow. removeAtt r( 'id ' )    ,
                                newEnt ryRow. removeAttr( 'style')    ;
                                newEntryRow.data( 'entryld',    row.id) ;
                                newEntryRow.appendTo( '#date ul')i
 newEntryRow.find('.label').text( row.food) ,
                                newEntryRow . find (    '    .calo ries')    .text ( row.calo ries) ;
                                newEnt ryRow.find ( '.delete').click(function() {
                            var clickedEntry = $(this).parent();
                        var clickedEntrId = clickedEntry.data('entryId);
                                deleteEnt ryById ( clickedEnt ryId)*
                                    clickedEntry.slideUp();
                    });
                                }
                     },
 errorHandler
                     );
                }
         );
 }
function deleteEntryById(id) {
                db.transaction (
        function(transaction) {
                 transaction.executeSql('DE LETE FROM entries WHERE id=?;',
                Lidl, null, errorHandler);
                 }
         );
 }
function errorHandler(transaction, error) {
      alert('Oops.Error was '+error.message+' (Code '+error.code+')');
          return.true,
}
```

第 9 章　开发蓝牙应用程序

蓝牙这一名称来自于十世纪的一位丹麦国王 Harald Blatand，Blatand 在英文里可以被解释为 Bluetooth。在现实应用中，通过蓝牙技术可以有效地简化移动通信终端设备之间的通信，也能够成功地简化设备与因特网之间的通信，从而使数据传输变得更加迅速、高效。在本章的内容中，将详细介绍在 Android 系统中开发蓝牙应用程序的基本知识，为读者步入本书后面知识的学习打下基础。

9.1　蓝牙技术基础

蓝牙技术的数据传输速率为 1Mbit/s，采用时分双工传输方式实现全双工传输。在本节的内容中，将首先讲解蓝牙技术的发展历程。

9.1.1　蓝牙技术的发展历程

蓝牙技术由瑞典爱立信公司创制，爱立信早在 1994 年就已进行研发。1997 年，爱立信与其他设备生产商联系，并激发了他们对该项技术的浓厚兴趣。1998 年 2 月，5 个跨国大公司，包括爱立信、诺基亚、IBM、东芝及 Intel，组成了一个特殊兴趣小组（SIG），他们共同的目标是创建一项全球性的小范围无线通信技术，即现在的蓝牙。

Bluetooth 无线技术规格供全球的成员公司免费使用。许多行业的制造商都积极地在其产品中实施此技术，以减少使用零乱的电线，实现无缝连接、流传输立体声，传输数据或进行语音通信。Bluetooth 技术在 2.4 GHz 波段运行，该波段是一种无需申请许可证的工业、科技、医学无线电波段。正因如此，使用 Bluetooth 技术不需要支付任何费用。但必须向手机提供商注册使用 GSM 或 CDMA，除了设备费用外，不需要为使用 Bluetooth 技术再支付任何费用。

Bluetooth 技术得到了空前广泛的应用，集成该技术的产品从手机、汽车到医疗设备，使用该技术的用户从消费者、工业市场到企业等等，不一而足。低功耗，小体积以及低成本的芯片解决方案使得 Bluetooth 技术甚至可以应用于极微小的设备中。

9.1.2　低功耗蓝牙的特点

低功耗蓝牙（Bluetooth Low Energy，BLE）是对传统蓝牙 BR/EDR 技术的补充。尽管 BLE 和传统蓝牙都称之为蓝牙标准，且共享射频，但是 BLE 是一个完全不一样的技术。BLE 不具备与传统蓝牙 BR/EDR 的兼容性，它是专为小数据率、离散传输的应用而设计的。

在实际应用过程中，BLE 的低功耗并不是通过优化空中的无线射频传输实现的，而是通过改变协议的设计来实现的。为了实现极低的功耗效果，通常 BLE 协议设计为：在不必要射

频的时候，将空中射频彻底关断。

与传统蓝牙 BR\EDR 相比，BLE 通过如下三大特性实现低功耗效果。

- 缩短无线开启时间。
- 快速建立连接。
- 降低收发峰值功耗（具体由芯片决定）。

缩短无线开启时间的第一个技巧是只用 3 个“广播”信道，第二个技巧是通过优化协议栈来降低工作周期。一个在广告的设备可以自动和一个在搜索的设备快速建立连接，所以可以在 3 毫秒内完成连接的建立和数据的传输。

在现实应用中，低功耗设计可能会带来一些牺牲，例如音频数据无法通过 BLE 来进行传输。尽管如此，BLE 仍然是一种非常出色的技术，依然会支持跳频（37 个数据信道），并且采用了一种改进的 GFSK 调制方法来提高链路的稳定性。BLE 也仍是非常安全的技术，因为在芯片级中提供了 128 bit AES 加密。

单模设备可以作为 Master 或者 Slave，但是不能同时充当两种角色。这意味着 BLE 只能建立简单的星状拓扑，不能实现散射网。在 BLE 的无线电规范中，定义了低功耗蓝牙的最高数据率为 305Kbit/s，但是，这只是理论数据。在实际应用中，数据的吞吐量取决于上层协议栈。而 UART 的速度、处理器的能力和主设备都会影响数据吞吐能力。

高数据吞吐能力的 BLE 只有通过私有方案或者基于 GATT Nnotification（蓝牙协议）才能实现。事实上，如果是高数据率或高数据量的应用，蓝牙 BR/EDR 通常显得更加省电。

9.1.3　低功耗蓝牙的架构

BLE 协议架构总体上分成三层，从下到上分别是：控制器（Controller）、主机（Host） 和应用端（Apps）。三者可以在同一芯片类中实现，也可以分别在不同芯片内实现。Controller 是处理射频数据的解析、接收和发送；Host 是控制不同设备之间如何进行数据交换；Apps 实现具体应用。

（1）控制器 Controller

Controller 实现射频相关的模拟和数字部分，完成最基本的数据发送和接收，Controller 对外接口是天线，对内接口是主机控制器接口（Host Controller Interface，HCI）；控制器包含物理层（Physical Layer，PHY），链路层（Linker Layer，LL），直接测试模式（Direct Test Mode，DTM）以及主机控制器接口 HCI；

- 物理层 PHY。GFSK 信号调制，2402MHz～2480MHz，40 个通道（channel），每两个 channel 间隔 2MHz（经典蓝牙协议是 1MHz），数据传输速率是 1Mbit/s。
- 直接测试模式 DTM。为射频物理层测试接口，供射频数据分析使用。
- 链路层 LL。基于 PHY 之上，实现数据通道分发，状态切换，数据包校验，加密等；LL 分 2 种通道：广播通道（advertising channels）和数据通道（data channels）；广播通道有 3 个：37ch（2402MHz）、38ch（2426MHz）、39ch（2480MHz），每次广播都会向这 3 个通道同时发送（并不会在这 3 个通道之间跳频），以防止某个通道被其他设备阻塞，以至于设备无法配对或广播数据，之所以定 3 个广播通道仅是一种权衡，少了可能会被阻塞，多了则会加大功耗，还有一个有意思的事情是，三个广播通道刚好避开了 WiFi 的 1ch、6ch、11ch，所以在 BLE 广播的时候，不至于被 WiFi 影响；

当 BLE 匹配之后， LL 由广播通道切换到数据通道，数据通道 37 个，数据传输的时候会在这 37 个通道间切换，切换规则在设备间匹配时约定。

（2）主机 Host/控制器 Controller/接口 HCI

HCI 作为一种接口，存在于主机 Host 和控制器 Controller 当中，控制器 Host 通过 HCI 发送数据和事件给主机，主机 Host 通过 HCI 发送命令和数据给控制器 Controller。HCI 逻辑上定义了一系列的命令、事件；物理上有 UART、SDIO、USB，实际上可能包含里面的任意 1 种或几种。

9.1.4 低功耗蓝牙分类

BLE 通常应用在传感器和智能手机或者平板电脑的通信中。到目前为止，只有很少的智能机和平板电脑支持 BLE，如 iPhone 4S 以后的 Apple 手机、Motorola Razr 和 the new iPad 及其以后的 iPad。Android 手机也逐渐支持 BLE，Android 的 BLE 标准在 2013 年 7 月 24 日发布。智能机和平板会带双模蓝牙的基带和协议栈，协议栈中包括 GATT 及以下的所有部分，但是没有 GATT 之上的具体协议。所以，这些具体的协议需要在应用程序中实现，实现时需要基于各个 GATT API 集。这样有利于在智能机端简单地实现具体协议，也可以在智能机端简单地开发出一套基于 GATT 的私有协议。

在现实应用中，低功耗蓝牙分为单模（Bluetooth Smart）和双模（Bluetooth Smart Ready）两种设备。BLE 和蓝牙 BR/EDR 有所区分，可以用三种方式将蓝牙技术集成到具体设备中。因为不再是蓝牙设备都可以和另一个蓝牙设备进行互联，所以准确描述产品中蓝牙的版本是非常地重要的。在接下来的内容中，将详细讲解单模蓝牙和双模蓝牙的基本知识。

（1）单模蓝牙

单模蓝牙设备被称为 Bluetooth Smart 设备，并且有专用的 Logo，如图 9-1 所示。

图 9-1 Bluetooth Smart 设备

在现实应用中，手表、运动传感器等小型设备通常是基于低功耗单模蓝牙的。为了实现极低的功耗，在硬件和软件上都进行了优化，这样的设备只能支持 BLE。单模蓝牙芯片往往是一个带有单模蓝牙协议栈的产品，这个协议栈通常是芯片商免费提供的。

（2）双模蓝牙

双模蓝牙设备被称为 Bluetooth Smart Ready 设备，并且有专用的 Logo，如图 9-2 所示。

双模设备支持蓝牙 BR/EDR 和 BLE。在双模设备中，BR/EDR 和 BLE 技术使用同一个射频前端和天线。典型的双模设备有智能手机、平板电脑、PC 和 Gateway。这些设备可以接收到通过 BLE 或者蓝牙 BR/EDR 设备发送过来的数据，这些设备往往都有足够的供电能力。双模设备和 BLE 设备通信的功耗低于双模设备和蓝牙 BR/EDR 设备通信的功耗。在使用双模解

决方案时，需要使用一个外部处理器才可以实现蓝牙协议栈。

图 9-2　Bluetooth Smart Ready 设备

9.2　分析 Android 系统中的蓝牙模块

在 Android 系统包含了对蓝牙网络协议栈的支持，这使得蓝牙设备能够无线连接到其他蓝牙设备以交换数据。Android 的应用程序框架提供了访问蓝牙功能的 API。这些 API 让应用程序能够无线连接其他蓝牙设备，实现点对点，或点对多的无线交互功能。

Android 平台的蓝牙系统是基于 BlueZ 实现的，是通过 Linux 中一套完整的蓝牙协议栈开源实现。当前 BlueZ 被广泛应用于各种 Linux 版本中，并被芯片公司移植到各种芯片平台上。在 Linux 2.6 内核中已经包含了完整的 BlueZ 协议栈，在 Android 系统中已经移植并嵌入进了 BlueZ 的用户空间实现，并且随着硬件技术的发展而不断更新。

在 Android 系统的蓝牙模块中，除了内核支持外，还需要有用户空间的 BlueZ 的支持。Android 系统中蓝牙模块的基本层次结构如图 9-3 所示。

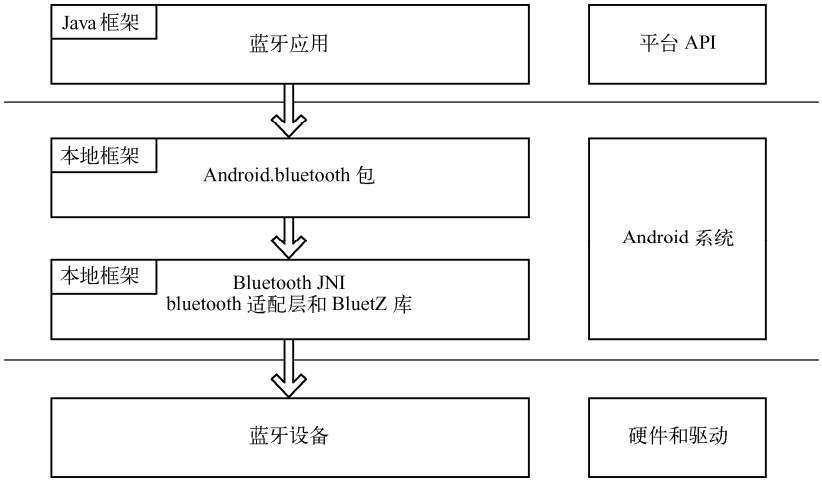


图 9-3　Android 蓝牙系统的层次结构

在图 9-3 所示的结构中，从上到下主要包括 Java 框架中的 BlueTooth 类、Android 适配库、BlueZ 库、驱动程序和协议，这几部分的具体说明如下。

（1）BlueZ 库（蓝牙设备管理库）

Android 蓝牙设备管理库的路径如下。

```
external/bluez/
```

可以分别生成库 libbluetooth.so、libbluedroid.so 和 hcidump 等众多相关的工具和库。BlueZ 库提供了对用户空间蓝牙的支持，在里面包含了主机控制协议 HCI 以及其他众多内核实现协议的接口，并且实现了所有蓝牙应用模式 Profile。

（2）JNI 部分

此部分的代码路径如下。

```
frameworks/base/core/jni/
```

（3）Java 框架层

Java 框架层的实现代码保存在如下的路径中。

```
frameworks/base/core/java/android/bluetooth        //蓝牙部分对应的应用程序的 API
frameworks/base/core/java/android/Server           //蓝牙的服务部分
```

蓝牙的服务部分负责管理并使用底层本地服务，并封装成系统服务。而在 android.bluetooth 部分中则包含了各个蓝牙平台的 API 部分，以供应用程序层所使用。

（4）BlueTooth 的适配库

BlueTooth 适配库的代码路径如下。

```
system/bluetooth/
```

此层用于生成库 libbluedroid.so 以及相关工具和库，能够实现对蓝牙设备的管理，例如蓝牙设备的电源管理。

9.3 Android 系统的低功耗蓝牙协议栈

从 Android 4.2 版本开始，Google 便更换了 Android 的蓝牙协议栈，将 Bluez 换成 BlueDroid。从 Android 4.3 版本开始，提供了对蓝牙 4.0 BLE 的支持。在本节的内容中，将详细讲解 Android 系统中的蓝牙 4.0 BLE 的基本知识。

9.3.1 Android 低功耗蓝牙协议栈基础

为了确保 Android 系统可以更好的支持蓝牙 4.0 BLE，Broadcom 公司特意推出了适应于 Android 平台的开源低功耗蓝牙协议栈 BlueDroid，其开发文档和 API 是开源代码，在如下的地址中保存。

```
https://github.com/briandbl/framework
```

在上述开源代码中，低功耗蓝牙 API 支持 Android 平台上的低功耗蓝牙通信功能。通过使用 BlueDroid 协议栈，Android 应用程序可以枚举、发现并访问低功耗蓝牙的外部设备，并且实现低功耗蓝牙规范。

从 Android 4.2 版本开始，低功耗蓝牙模块的整体结构如图 9-4 所示。

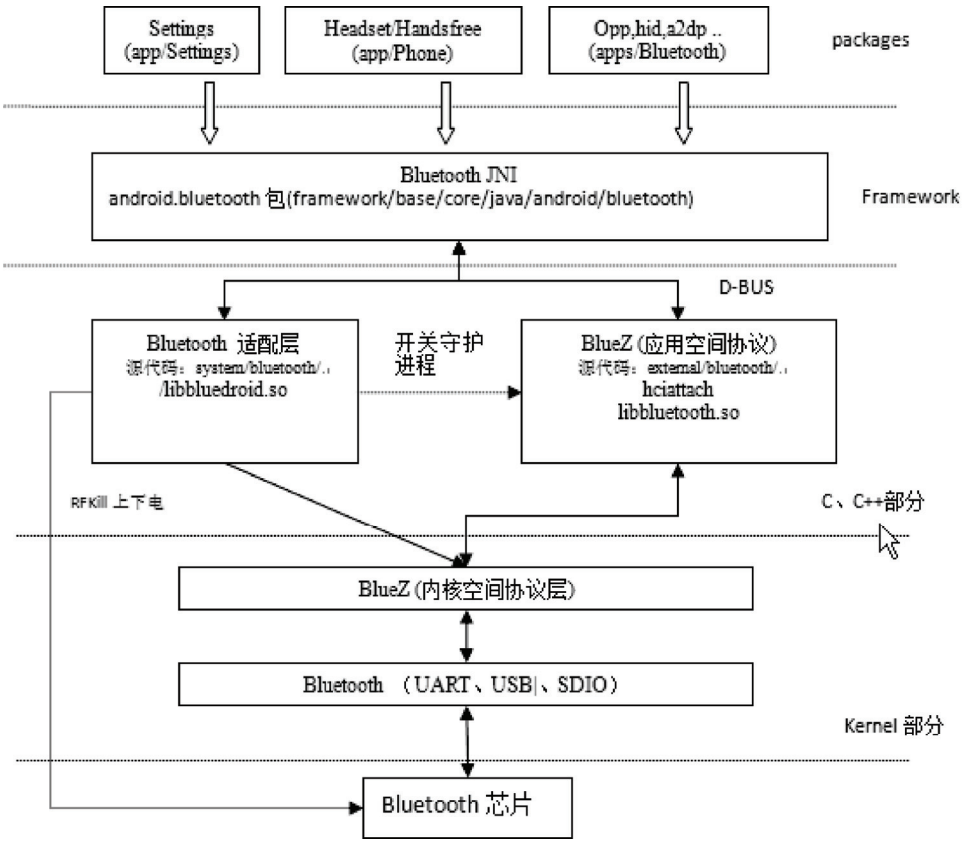


图 9-4　低功耗蓝牙模块的整体结构

注意：虽然从 Android 4.2 版本开始，JNI 部分的代码在 packages 层中实现。但是为了便于读者从视觉上更加容易接受，特将 JNI 部分绘制在了 Framework 层中。

9.3.2　低功耗蓝牙 API 详解

在接下来的内容中，将详细讲解主要 API 的基本功能和具体原理。

（1）本地蓝牙适配器设备

本功能不是由 Broadcom 公司提供，而是由 Android SDK 提供，源码位于如下的目录中。

```
framework/base/core/java/android.bluetooth/BluetoothAdapter.java
```

文件 BluetoothAdapter.java 实现了所有蓝牙交互的入口。通过使用类 BluetoothAdapter 可以实现如下的功能。

- 发现其他的蓝牙设备，查询匹配的设备集。
- 使用一个已知蓝牙地址来初始化蓝牙设备 BluetoothDevice。
- 创建一个能够监听其他设备通信的类 BluetoothSocket。

文件 BluetoothAdapter.java 的主要实现代码如下。

```
public static synchronized BluetoothAdapter getDefaultAdapter() {
            if (sAdapter == null) {
        IBinder b = ServiceManager.getService(BLUETOOTH_MANAGER_SERVICE);
        if (b != null) {
            IBluetoothManager managerService = IBluetoothManager.Stub.asInterface(b);
            sAdapter = new BluetoothAdapter(managerService);
        } else {
            Log.e(TAG, "Bluetooth binder is null");
        }
    }
    return sAdapter;
}
BluetoothAdapter(IBluetoothManager managerService) {
            if (managerService == null) {
        throw new IllegalArgumentException("bluetooth manager service is null");
    }
    try {
        mService = managerService.registerAdapter(mManagerCallback);
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    mManagerService = managerService;
    mLeScanClients = new HashMap<LeScanCallback, GattCallbackWrapper>();
}
public BluetoothDevice getRemoteDevice(byte[] address) {
            if (address == null || address.length != 6) {
        throw new IllegalArgumentException("Bluetooth address must have 6 bytes");
    }
    return new BluetoothDevice(String.format("%02X:%02X:%02X:%02X:%02X:%02X",
            address[0], address[1], address[2], address[3], address[4], address[5]));
}
public int getState() {
    try {
        synchronized(mManagerCallback) {
            if (mService != null)
            {
                int state=    mService.getState();
                if (VDBG) Log.d(TAG, "" + hashCode() + ": getState(). Returning " + state);
                return state;
            }
        }
    } catch (RemoteException e) {Log.e(TAG, "", e);}
    if (DBG) Log.d(TAG, "" + hashCode() + ": getState() :    mService = null. Returning STATE_OFF");
    return STATE_OFF;
}
public String getAddress() {
    try {
        return mManagerService.getAddress();
```

```
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return null;
    }
    public String getName() {
        try {
            return mManagerService.getName();
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return null;
    }
    public int getScanMode() {
        if (getState() != STATE_ON) return SCAN_MODE_NONE;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.getScanMode();
            }
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return SCAN_MODE_NONE;
    }
    public boolean setScanMode(int mode, int duration) {
        if (getState() != STATE_ON) return false;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.setScanMode(mode, duration);
            }
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return false;
    }
    public boolean setScanMode(int mode) {
        if (getState() != STATE_ON) return false;
        return setScanMode(mode, getDiscoverableTimeout());
    }
    public int getDiscoverableTimeout() {
        if (getState() != STATE_ON) return -1;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.getDiscoverableTimeout();
            }
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return -1;
    }
    public void setDiscoverableTimeout(int timeout) {
        if (getState() != STATE_ON) return;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) mService.setDiscoverableTimeout(timeout);
            }
```

```
        } catch (RemoteException e) {Log.e(TAG, "", e);}
    }
    public boolean startDiscovery() {
        if (getState() != STATE_ON) return false;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.startDiscovery();
            }
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return false;
    }
    public boolean cancelDiscovery() {
        if (getState() != STATE_ON) return false;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.cancelDiscovery();
            }
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return false;
    }
    public boolean isDiscovering() {
        if (getState() != STATE_ON) return false;
        try {
            synchronized(mManagerCallback) {
                if (mService != null ) return mService.isDiscovering();
            }
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return false;
    }
    public Set<BluetoothDevice> getBondedDevices() {
        if (getState() != STATE_ON) {
            return toDeviceSet(new BluetoothDevice[0]);
        }
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return toDeviceSet(mService.getBondedDevices());
            }
            return toDeviceSet(new BluetoothDevice[0]);
        } catch (RemoteException e) {Log.e(TAG, "", e);}
        return null;
    }
    public int getConnectionState() {
        if (getState() != STATE_ON) return BluetoothAdapter.STATE_DISCONNECTED;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.getAdapterConnectionState();
```

```
            }
        } catch (RemoteException e) {Log.e(TAG, "getConnectionState:", e);}
        return BluetoothAdapter.STATE_DISCONNECTED;
    }
    public int getProfileConnectionState(int profile) {
        if (getState() != STATE_ON) return BluetoothProfile.STATE_DISCONNECTED;
        try {
            synchronized(mManagerCallback) {
                if (mService != null) return mService.getProfileConnectionState(profile);
            }
        } catch (RemoteException e) {
            Log.e(TAG, "getProfileConnectionState:", e);
        }
        return BluetoothProfile.STATE_DISCONNECTED;
    }
    public BluetoothServerSocket listenUsingRfcommOn(int channel) throws IOException {
        BluetoothServerSocket socket = new BluetoothServerSocket(
                BluetoothSocket.TYPE_RFCOMM, true, true, channel);
        int errno = socket.mSocket.bindListen();
        if (errno != 0) {
            throw new IOException("Error: " + errno);
        }
        return socket;
    }
    public BluetoothServerSocket listenUsingRfcommWithServiceRecord(String name, UUID uuid)
            throws IOException {
        return createNewRfcommSocketAndRecord(name, uuid, true, true);
    }
    private BluetoothServerSocket createNewRfcommSocketAndRecord(String name, UUID uuid,
            boolean auth, boolean encrypt) throws IOException {
        BluetoothServerSocket socket;
        socket = new BluetoothServerSocket(BluetoothSocket.TYPE_RFCOMM, auth,
                                encrypt, new ParcelUuid(uuid));
        socket.setServiceName(name);
        int errno = socket.mSocket.bindListen();
        if (errno != 0) {
            throw new IOException("Error: " + errno);
        }
        return socket;
    }
```

（2）请求远程蓝牙设备

本功能也是由 Android SDK 提供的，源码位于如下的目录中。

```
framework/base/core/java/android.bluetooth/BluetoothDevice.java
```

文件 BluetoothDevice.java 代表一个远程蓝牙设备，可以支持 BLE 低功耗设备、BR/EDR

设备或 Dual-mode 类型的设备。通过使用类 BluetoothDevice 可以实现如下的功能。

- 请求获取远程蓝牙设备的连接。
- 查询获取远程蓝牙设备的名称、地址、类和链接状态。

文件 BluetoothDevice.java 的主要实现代码如下。

```
static IBluetooth getService() {
        synchronized (BluetoothDevice.class) {
                if (sService == null) {
            BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
            sService = adapter.getBluetoothService(mStateChangeCallback);
            }
        }
        return sService;
    }
    static IBluetoothManagerCallback mStateChangeCallback = new IBluetoothManagerCallback.Stub() {

        public void onBluetoothServiceUp(IBluetooth bluetoothService)
                throws RemoteException {
            synchronized (BluetoothDevice.class) {
                sService = bluetoothService;
            }
        }
        public void onBluetoothServiceDown()
            throws RemoteException {
            synchronized (BluetoothDevice.class) {
                sService = null;
            }
        }
    };
        if (!BluetoothAdapter.checkBluetoothAddress(address)) {
            throw new IllegalArgumentException(address + " is not a valid Bluetooth address");
        }
        mAddress = address;
    }
    @Override
    public boolean equals(Object o) {
        if (o instanceof BluetoothDevice) {
            return mAddress.equals(((BluetoothDevice)o).getAddress());
        }
        return false;
    }
    @Override
    public int hashCode() {
        return mAddress.hashCode();
    }
    public static final Parcelable.Creator<BluetoothDevice> CREATOR =
```

```
            new Parcelable.Creator<BluetoothDevice>() {
        public BluetoothDevice createFromParcel(Parcel in) {
            return new BluetoothDevice(in.readString());
        }
        public BluetoothDevice[] newArray(int size) {
            return new BluetoothDevice[size];
        }
    };
```

（3）实现客户端的低功耗蓝牙规范

在 Broadcom 公司提供的源码中，文件 BleClientProfile.java 实现客户端的低功耗蓝牙规范。在应用中要想访问远程设备中的低功耗蓝牙规范，就必须继承类 BleClientProfile，并且需要提供要访问的规范的必须参数和服务标识。通过 BleClientProfile 的派生类可以发起一个远程设备的连接，并且一个 BleClientProfile 类可能会包含多个 BleClientService 对象的实例。文件 BleClientProfile.java 的具体实现代码如下。

```
    //下面是构造方法，功能是给当前规范的 UUID 和客户端应用上下文创建一个 BleClientProfile
    public BleClientProfile(Context context, BleGattID profileUuid)
    {
        Log.d(TAG, "new profile" + profileUuid.toString());
        this.mContext = context;
        this.mAppUuid = profileUuid;
        this.mConnectedDevices = new ArrayList<BluetoothDevice>();
        this.mConnectingDevices = new ArrayList<BluetoothDevice>();
        this.mDisconnectingDevices = new ArrayList<BluetoothDevice>();
        this.mClientIDToDeviceMap = new HashMap<Integer, BluetoothDevice>();
        this.mDeviceToClientIDMap = new HashMap<BluetoothDevice, Integer>();
        this.mCallback = new BleClientCallback();
        this.mSvcConn = new GattServiceConnection(context);
    }
    /**
     * 初始化 BleClientProfile 对象
     */
    public void init(ArrayList<BleClientService> requiredServices,
            ArrayList<BleClientService> optionalServices)
    {
        Log.d(TAG, "init (" + this.mAppUuid + ")");
        this.mRequiredServices = requiredServices;
        this.mOptionalServices = optionalServices;
        IBinder b = ServiceManager.getService(BleConstants.BLUETOOTH_LE_SERVICE);
                if (b == null) {
            throw new RuntimeException("Bluetooth Low Energy service not available");
        }
        this.mSvcConn.onServiceConnected(null, b);
    }
    /**
```

```
 * 清楚和此规范有关的资源
 */
public synchronized void finish()
{
    if (this.mSvcConn != null) {
        this.mContext.unbindService(this.mSvcConn);
        this.mSvcConn = null;
    }
}
@Override
/**
 *返回此规范是否已经成功注册到蓝牙协议栈中.
 * @see {@link #registerProfile()}
 */
public boolean isProfileRegistered()
{
    Log.d(TAG, "isProfileRegistered (" + this.mAppUuid + ")");
    return this.mClientIf != BleConstants.GATT_SERVICE_PRIMARY;
}
/**
 * 注册规范到蓝牙协议栈
 */
public int registerProfile()
{
    int ret = BleConstants.GATT_SUCCESS;
    Log.d(TAG, "registerProfile (" + this.mAppUuid + ")");

            if (this.mClientIf == BleConstants.GATT_SERVICE_PRIMARY)
    {
        try
        {
            this.mService.registerApp(this.mAppUuid, this.mCallback);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
            ret = BleConstants.SERVICE_UNAVAILABLE;
        }
    }
    return ret;
}
/**
 *注销蓝牙协议栈中的规范
 */
public void deregisterProfile()
{
    Log.d(TAG, "deregisterProfile (" + this.mAppUuid + ")");
    if (this.mClientIf != BleConstants.GATT_SERVICE_PRIMARY)
```

```
            try {
                this.mService.unregisterApp(this.mClientIf);
            } catch (RemoteException e) {
                Log.e(TAG, "deregisterProfile() - " + e.toString());
            }
    }
    /**
     * 设置一个活跃连接设备的加密等级
     */
    public void setEncryption(BluetoothDevice device, byte action)
    {
        try
        {
            this.mService.setEncryption(device.getAddress(), action);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
        }
    }
    /**
     * 当请求后台连接时，定义本地设备扫描远程低功耗设备的强度
     */
    public void setScanParameters(int scanInterval, int scanWindow)
    {
        try
        {
            this.mService.setScanParameters(scanInterval, scanWindow);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
        }
    }
    /**
     * 建立一个到远程设备的 GATT 连接
     */
    public int connect(BluetoothDevice device)
    {
        Log.d(TAG, "connect (" + this.mAppUuid + ")" + device.getAddress());
        int ret = BleConstants.GATT_SUCCESS;
        synchronized (this.mConnectingDevices) {
            this.mConnectingDevices.add(device);
        }
        synchronized (this.mDisconnectingDevices) {
            this.mDisconnectingDevices.remove(device);
        }
        try
        {
            this.mService.open(this.mClientIf, device.getAddress(), true);
```

```
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
            ret = BleConstants.GATT_ERROR;
        }
        return ret;
    }
    /**
     * 准备一个到远程蓝牙设备的后台连接
     */
    public int connectBackground(BluetoothDevice device)
    {
        Log.d(TAG,
                "connectBackground (" + this.mAppUuid + ")" + device.getAddress());
        int ret = BleConstants.GATT_SUCCESS;
        synchronized (this.mConnectingDevices) {
            this.mConnectingDevices.add(device);
        }
        synchronized (this.mDisconnectingDevices) {
            this.mDisconnectingDevices.remove(device);
        }
        try
        {
            this.mService.open(this.mClientIf, device.getAddress(), false);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
            ret = BleConstants.GATT_ERROR;
        }
        return ret;
    }
    /**
     * 停止监听远程蓝牙设备试图发起的链接
     */
    public int cancelBackgroundConnection(BluetoothDevice device)
    {
        Log.d(TAG, "cancelBackgroundConnection (" + this.mAppUuid
                + ") - device " + device.getAddress());

        int ret = BleConstants.GATT_SUCCESS;
        try
        {
            this.mService.close(this.mClientIf, device.getAddress(), 0, false);
        } catch (RemoteException e) {
            Log.e(TAG, e.toString());
            ret = BleConstants.GATT_ERROR;
        }
        return ret;
```

```
}
/**
 *断开一个到远程设备的 GATT 连接
 */
public int disconnect(BluetoothDevice device)
{
    Log.d(TAG,
            "disconnect (" + this.mAppUuid + ") - device " + device.getAddress());

    synchronized (this.mDisconnectingDevices) {
        this.mDisconnectingDevices.add(device);
    }
    int ret = BleConstants.GATT_SUCCESS;
    try
    {
        this.mService.close(this.mClientIf,
                device.getAddress(),
                ((Integer) this.mDeviceToClientIDMap.get(device)).intValue(),
                true);
    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
        ret = BleConstants.GATT_ERROR;
    }
    return ret;
}
/**
 * 刷新当前客户端的规范
 */
public int refresh(BluetoothDevice device)
{
    Log.d(TAG,
            "refresh (" + this.mAppUuid + ") - address = " + device.getAddress());
    if (isDeviceDisconnecting(device)) {
        Log.d(TAG, "refresh (" + this.mAppUuid
                + ") - Device unavailable!");
        return BleConstants.GATT_ERROR;
    }
    this.mRequiredServices.get(BleConstants.GATT_SERVICE_PRIMARY).refresh(device);
    return BleConstants.GATT_SUCCESS;
}
/**
 * 刷新当前规范包含的特定服务
 */
public int refreshService(BluetoothDevice device, BleClientService service)
{
    Log.d(TAG, "refreshService (" + this.mAppUuid + ") address = s "
```

```
                + device.getAddress() + "service = " + service.getServiceId());
        return 0;
    }
    /**
     * 在已经连接的设备列表中查找指定蓝牙设备的地址
     */
    public BluetoothDevice findConnectedDevice(String address)
    {
        BluetoothDevice ret = null;
        synchronized (this.mConnectedDevices) {
            for (int i = 0; i != this.mConnectedDevices.size(); i++) {
                BluetoothDevice d = (BluetoothDevice) this.mConnectedDevices.get(i);
                if (address.equalsIgnoreCase(d.getAddress())) {
                    ret = d;
                    break;
                }
            }
        }
        return ret;
    }
    /**
     * 返回当前连接和等待连接中的所有远程设备集合
     */
    public BluetoothDevice[] getPendingConnections()
    {
        return (BluetoothDevice[]) this.mConnectingDevices.toArray(new BluetoothDevice[0]);
    }
    /**
     *设置一个蓝牙设备地址，在等待连接设备列表中查找一个远程设备
     */
    public BluetoothDevice findDeviceWaitingForConnection(String address)
    {
        BluetoothDevice ret = null;
        synchronized (this.mConnectingDevices) {
            for (int i = 0; i < this.mConnectingDevices.size(); i++) {
                BluetoothDevice d = (BluetoothDevice) this.mConnectingDevices.get(i);
                if (address.equalsIgnoreCase(d.getAddress())) {
                    ret = d;
                    break;
                }
            }
        }
        return ret;
    }
```

（4）创建一个代表客户端角色设备上的低功耗蓝牙服务派生类

在 Broadcom 公司提供的源码中，文件 BleClientService.java 定义了一个派生类，此派生

类代表了客户端角色设备上的低功耗蓝牙服务。通过这个派生类可以允许应用程序读写低功耗蓝牙服务的特征，并且在特征改变时注册通知。文件 BleClientService.java 的主要实现代码如下。

```
//定义代表客户端的低功耗服务
public abstract class BleClientService
{
    private static String TAG = "BleClientService";
    private BleClientProfile mProfile = null;
    private BleGattID mServiceId = null;
    private HashMap<BluetoothDevice, ArrayList<ServiceData>> mdeviceToDataMap =
            new HashMap<BluetoothDevice, ArrayList<ServiceData>>();
    private BleCharacteristicDataCallback mCallback =
            new BleCharacteristicDataCallback();
    private boolean mReadDescriptors = true;
    /**
     *创建一个新的低功耗蓝牙服务的 UUID
     *
     * @param serviceId
     */
    public BleClientService(BleGattID serviceId)
    {
        mServiceId = serviceId;
            if (mServiceId.getServiceType()== BleConstants.GATT_UNDEFINED)
            mServiceId.setServiceType(BleConstants.GATT_SERVICE_PRIMARY);
    }
    /**
     * 返回服务的 UUID
     */
    public BleGattID getServiceId()
    {
        return mServiceId;
    }
    /**
     *写操作远程设备上的一个特性
     */
    public int writeCharacteristic(BluetoothDevice remoteDevice, int instanceId,
            BleCharacteristic characteristic)
    {
        Log.d(TAG, "writeCharacteristic");
        int ret = BleConstants.GATT_SUCCESS;
        int connID = BleConstants.GATT_INVALID_CONN_ID;
            if ((connID = mProfile.getConnIdForDevice(remoteDevice)) == BleConstants.GATT_
INVALID_CONN_ID) {
            return BleConstants.GATT_INVALID_CONN_ID;
        }
```

```
ServiceData s = getServiceData(remoteDevice, instanceId);
        if (s == null) {
    return ret;
}
s.writeIndex = s.characteristics.indexOf(characteristic);
if ((s.characteristics != null) && (s.writeIndex >= BleConstants.GATT_SERVICE_PRIMARY)) {
    Log.d(TAG, "writeCharacteristic found characteristic in array:");
    Log.d(TAG,
            "Service = [instanceID = " + instanceId + " svcid = "
                    + mServiceId.toString() + " serviceType = "
                    + mServiceId.getServiceType());
    Log.d(TAG, "CharID = [instanceID = " + characteristic.getInstanceID()
            + " svcid = " + characteristic.getID().toString());
    BleGattID svcId = new BleGattID(instanceId, mServiceId.getUuid(),
            mServiceId.getServiceType());
    BleGattID cID = characteristic.getID();
    BluetoothGattCharID charID = new BluetoothGattCharID(svcId, cID);
    try
    {
        if (characteristic.isDirty()) {
                if (characteristic.getWriteType()== BleConstants.GATT_SUCCESS)
            characteristic.setWriteType(2);
            characteristic.setDirty(false);
            mProfile.getGattService().writeCharValue(connID, charID,
                    characteristic.getWriteType(), characteristic.getAuthReq(),
                    characteristic.getValue());
        }
        else if (!characteristic.getDirtyDescQueue().isEmpty()) {
            ArrayList<BleDescriptor> descList =
                    characteristic.getDirtyDescQueue();
            BleDescriptor descObj = descList.get(0);
            Log.d(TAG, "writeCharacteristic - descriptor = "
                    + descObj.getID().toString());
            if (descObj.isDirty()) {
                BluetoothGattCharDescrID descID = new BluetoothGattCharDescrID(
                        svcId, cID, descObj.getID());
                descObj.setDirty(false);
                mProfile.getGattService().writeCharDescrValue(connID,
                        descID, descObj.getWriteType(), descObj.getAuthReq(),
                        descObj.getValue());
            }
        }
        else
        {
            onWriteCharacteristicComplete(0, remoteDevice, characteristic);
        }
```

```
            } catch (RemoteException e) {
                ret = BleConstants.GATT_ERROR;
            }
        } else {
            onWriteCharacteristicComplete(0, remoteDevice, characteristic);
        }
        return ret;
    }
```

（5）定义服务器端的角色的低功耗规范

在 Broadcom 公司提供的源码中，文件 BleServerProfile.java 定义了服务器端的角色的低功耗规范，在创建一个新的低功耗规范之前，需要先继承于这个类，并提供标识要访问规范所必须的参数和服务。通常来说，一个 BleServerProfile 派生的类包含一个或多个 BleServerService 对象。在 BleServerProfile 派生的类中，包含低功耗规范中定义服务的 BleServerService 对象的集合。文件 BleServerProfile.java 的主要实现代码如下。

```
public abstract class BleServerProfile
{
    private static final boolean D = true;
    private static final String TAG = "BleServerProfile";
    private Context mCtxt = null;
    private BleGattID mAppid;
    ArrayList<BleServerService> mServiceArr = null;
    private HashMap<String, Integer> mConnMap = null;
    private HashMap<Integer, Integer> mMtuMap = null;
    private IBluetoothGatt mService;
    private int mSvcCreated = 0;
    private int mSvcStarted = 0;
    private byte mAppHandle = –1;
    private int mProfileStatus = 2;
    private GattServiceConnection mSvcConn;
    public BleServerProfile(Context ctxt, BleGattID appId,
            ArrayList<BleServerService> serviceArr)
    {
        mAppid = appId;
        mCtxt = ctxt;
        mServiceArr = serviceArr;
        mConnMap = new HashMap<String, Integer>();
        mMtuMap = new HashMap<Integer, Integer>();
        mSvcConn = new GattServiceConnection(null);
        Intent i = new Intent();
        i.setClassName("com.broadcom.bt.app.system",
                "com.broadcom.bt.app.system.GattService");
        mCtxt.bindService(i, mSvcConn, 1);
        throw new RuntimeException("Not implemented");
    }
```

```
/*取消和此规范相关的资源*/
public synchronized void finish()
{
    if (mSvcConn != null) {
        mCtxt.unbindService(mSvcConn);
        mSvcConn = null;
    }
}
public void finalize()
{
    finish();
}
byte getAppHandle()
{
    return mAppHandle;
}
HashMap<String, Integer> getConnMap() {
    return mConnMap;
}
/*初始化相关的服务*/
void initProfile()
{
    Log.i("BleServerProfile", "initProfile()");
    try {
        mService.registerServerProfileCallback(mAppid,
                new BleServerProfileCallback(this));
    } catch (Throwable t) {
        Log.e("BleServerProfile", "Unable to start profile", t);
    }
}
void notifyAction(int event)
{
            if ((event == 0) && (++mSvcCreated == mServiceArr.size()))
    {
        Log.i("BleServerProfile",
                "All services created successfully. Calling onInitialized");
        onInitialized(true);
            } else if ((event == 4) && (--mSvcCreated == 0))
    {
        Log.i("BleServerProfile",
                "All services stopped successfully. Calling onStopped");
        onStopped();
            } else if ((event == 2) && (++mSvcStarted == mServiceArr.size()))
    {
        Log.i("BleServerProfile",
                "All services started successfully. Calling onStarted");
```

```
            onStarted(true);
                } else if (event == 1) {
            Log.i("BleServerProfile",
                        "One of the services creation failed. Calling onInitialized");
            mProfileStatus = 2;
            onInitialized(false);
                } else if (event == 3) {
            Log.i("BleServerProfile",
                        "One of the services start failed. Calling onStarted");
            mProfileStatus = 2;
            onStarted(false);
        } else {
            Log.e("BleServerProfile", "Unknown action from a service");
        }
    }
    /*启用和此规范有关的所有服务*/
    public void startProfile()
    {
        Log.i("BleServerProfile", "startProfile()");
                if (mService == null) {
            Log.i("BleServerProfile", "Remote service object is null.. Returning..");
            return;
        }
        for (int i = 0; i < mServiceArr.size(); i++) {
            if (!((BleServerService) mServiceArr.get(i)).isRegistered()) {
                Log.i("BleServerProfile",
                            "One of the services is not registered. Stopping all the services");
                stopProfile();
                return;
            }
            ((BleServerService) mServiceArr.get(i)).startService();
        }
    }
    /*停止和此规范有关的所有服务*/
    public void stopProfile()
    {
        Log.i("BleServerProfile", "stopProfile()");
        for (int i = 0; i < mServiceArr.size(); i++)
            ((BleServerService) mServiceArr.get(i)).stopService();
    }
    /*注销所有相关的服务*/
    public void finishProfile()
    {
        Log.i("BleServerProfile", "finishProfile()");
        for (int i = 0; i < mServiceArr.size(); i++) {
            ((BleServerService) mServiceArr.get(i)).deleteService();
```

```
        }
        try
        {
            mService.unregisterServerProfileCallback(mAppHandle);
        } catch (Throwable t) {
            Log.e("BleServerProfile", "Unable to stop profile", t);
            return;
        }
    }
    /*为连接设置最大传输单元*/
    public void setMtuSize(int connId, int mtuSize)
    {
        Log.i("BleServerProfile", "setMtuSize");
        mMtuMap.put(Integer.valueOf(connId), Integer.valueOf(mtuSize));
    }
    /*为一个活跃的连接设置需要的加密等级*/
    public void setEncryption(String bdaddr, byte action)
    {
        try
        {
            mService.setEncryption(bdaddr, action);
        } catch (Throwable t) {
            Log.e("BleServerProfile", "Unable to set encryption for connection", t);
        }
    }
    /*当已经请求一个后台链接时，定义本地设备扫描远程低功耗设备的强度*/
    public void setScanParameters(int scanInterval, int scanWindow)
    {
        try
        {
            mService.setScanParameters(scanInterval, scanWindow);
        } catch (Throwable t) {
            Log.e("BleServerProfile", "Unable to set scan parameters", t);
        }
    }
    /*打开一个外设 GAP 客户端的连接*/
    public void open(String bdaddr, boolean isDirect)
    {
        try
        {
            mService.GATTServer_Open(mAppHandle, bdaddr, isDirect);
        } catch (Throwable t) {
            Log.e("BleServerProfile", "Unable to open Gatt connection", t);
        }
    }
    /*取消一个正在进行中对外设 GATT 客户端的打开操作*/
```

```
    public void cancelOpen(String bdaddr, boolean isDirect)
    {
        try
        {
            mService.GATTServer_CancelOpen(mAppHandle, bdaddr, isDirect);
        } catch (Throwable t) {
            Log.e("BleServerProfile", "Unable to open Gatt connection", t);
            return;
        }
    }
    /*关闭一个到远程低功耗规范客户端的连接*/
    public void close(String bdaddr)
    {
        try
        {
            mService.GATTServer_Close(((Integer) mConnMap.get(bdaddr))
                    .intValue());
        } catch (Throwable t) {
            Log.e("BleServerProfile", "Unable to open Gatt connection", t);
            return;
        }
    }
```

（6）创建低功耗服务

在 Broadcom 公司提供的源码中，文件 BleServerService.java 创建了一个低功耗服务，这是服务器端角色上的低功耗规范的一部分。在 BleServerService 的派生类中包含了一个或多个 BleCharacteristic 对象。在应用程序中，需要重写类 BleServerService 来实现一个服务。文件 BleServerService.java 的主要实现代码如下。

```
public abstract class BleServerService
{
    private final String TAG = "BleServerService";
    private HashMap<Integer, BleCharacteristic> mCharHdlMap = null;
    private HashMap<Integer, BleServerService> mServiceHdlMap = null;
    private HashMap<Integer, AttributeRequestInfo> mAttrReqMap = null;
    private ArrayList<BleCharacteristic> mCharQueue = null;
    private ArrayList<BleDescriptor> mDirtyDescQueue = null;
    private BleGattID mServiceId;
    private BleGattID mAppUuid;
    private BleServerProfile mProfileHandle;
    private IBluetoothGatt mService;
    private int mSvcHandle = -1;
    private byte mSupTransport;
    private BleServiceCallback mGattServiceCallback;
    private boolean isServiceAvailable = false;
    private int mSvcInstance = 0;
```

```
private boolean isPrimary = false;
private int mNumHandles;
private final int CHAR_ADDED = 0;
private final int CHAR_DESC_ADDED = 1;
private final int ATTRIBUTE_WRITE = 2;
private final int ATTRIBUTE_READ = 3;
private final int HDL_VAL_INDICATION = 4;
private final int HDL_VAL_NOTIFICATION = 5;
private final int MTU_EXCHANGE = 6;
private final int EXECUTE_WRITE = 7;
private Handler mHandler = new Handler()
{
    public void handleMessage(Message msg)
    {
    }

};
int getConnId(String address)
{
            if (this.mProfileHandle == null)
        return -1;
    HashMap connMap = this.mProfileHandle.getConnMap();
    return ((Integer) connMap.get(address)).intValue();
}
/*构造函数，使用给定的 ID 构造一个低功耗服务*/
public BleServerService(BleGattID serviceId, int numHandles)
{
    /**
     * TODO: implement
     */
    this.mServiceId = serviceId;
    this.mNumHandles = numHandles;
    this.mSupTransport = 2;
    this.mGattServiceCallback = new BleServiceCallback(this);
    this.mCharHdlMap = new HashMap();
    this.mServiceHdlMap = new HashMap();
    this.mCharQueue = new ArrayList();
    this.mAttrReqMap = new HashMap();

            if (this.mServiceId.getServiceType()== -1)
        this.mServiceId.setServiceType(0);
    throw new RuntimeException("not implemented");
}
/*构造函数，使用给定的 ID 构造一个新的低功耗服务*/
public BleServerService(BleGattID serviceId, byte supTransport, int numHandles)
{
```

```
        this.mServiceId = serviceId;
        this.mNumHandles = numHandles;
        this.mSupTransport = supTransport;
        this.mGattServiceCallback = new BleServiceCallback(this);
        this.mCharHdlMap = new HashMap();
        this.mCharQueue = new ArrayList();
        this.mServiceHdlMap = new HashMap();
        this.mAttrReqMap = new HashMap();
                if (this.mServiceId.getServiceType()== -1)
            this.mServiceId.setServiceType(0);
        throw new RuntimeException("not implemented");
    }
    /*初始化服务*/
    protected void initService()
    {
        if (this.mService != null)
            try {
                this.mService.registerServerServiceCallback(this.mServiceId,
                        this.mAppUuid, this.mGattServiceCallback);
            } catch (Throwable t) {
                Log.e("BleServerService", "initService", t);
            }
    }
    /*注册服务到蓝牙协议栈*/
    public void createService()
    {
        if (this.mService != null)
            try {
                this.mService.GATTServer_CreateService(this.mProfileHandle.getAppHandle(),
                        this.mServiceId, this.mNumHandles);
            } catch (Throwable t)
            {
                Log.e("BleServerService", "createService", t);
            }
    }
    /*从蓝牙协议栈注销服务*/
    public void deleteService()
    {
        if (this.mService != null)
            try {
                this.mService.GATTServer_DeleteService(this.mSvcHandle);
            } catch (Throwable t) {
                Log.e("BleServerService", "deleteService ", t);
            }
    }
    /*启用服务*/
```

```
public void startService()
{
    if (this.mService != null)
        try {
            this.mService.GATTServer_StartService(this.mSvcHandle, this.mSupTransport);
        } catch (Throwable t) {
            Log.e("BleServerService", "startService ", t);
        }
}
/*停止服务*/
public void stopService()
{
    if (this.mService != null) {
        this.mProfileHandle.notifyAction(4);
        try {
            this.mService.unregisterServerServiceCallback(this.mSvcHandle);
            this.mService.GATTServer_StopService(this.mSvcHandle);
        } catch (Throwable t) {
            Log.e("BleServerService", "stopService ", t);
        }
    }
}
/*为此服务添加一个包含的服务*/
public void addIncludedService(BleServerService service)
{
    if (this.mService != null)
        try {
            if (service.isRegistered()) {
                this.mServiceHdlMap.put(Integer.valueOf(service.getServiceHandle()),
                        service);
                this.mService.GATTServer_AddIncludedService(this.mSvcHandle,
                        service.getServiceHandle());
            }
            else {
                Log.i("BleServerService",
                        "addIncludedService: Service to be included is not registered.");
            }
        } catch (Throwable t) {
            Log.e("BleServerService", "addIncludedService", t);
        }
}
/*更新一个特性或描述符*/
public void updateCharacteristic(BleCharacteristic charObj)
{
    addCharacteristic(charObj);
}
```

```
/*当客户端已经请求读或写一个本地特性属性后发送一个响应*/
public void sendResponse(String address, int transId, byte[] data, int statusCode)
{
    Log.d("BleServerService", "sendResponse() address = " + address + ", transId = "
            + transId + ",statusCode = " + statusCode);

            if (this.mService == null) {
        Log.e("BleServerService", "sendResponse(): error. GattService not available");
        return;
    }
    AttributeRequestInfo attrInfo = (AttributeRequestInfo) this.mAttrReqMap
            .remove(Integer.valueOf(transId));
            if (attrInfo == null)
    {
        Log.e("BleServerService",
                "sendResponse() error. attrInfo not found with transId " + transId);
        return;
    }
    byte[] dataToSend = null;
            if (attrInfo.mOffset == 0) {
        dataToSend = data;
    } else {
        dataToSend = new byte[data.length - attrInfo.mOffset];
        System.arraycopy(data, attrInfo.mOffset, dataToSend, 0, dataToSend.length);
    }
    try
    {
        this.mService
                .GATTServer_SendRsp(
                        attrInfo.mConnId,
                        attrInfo.mTransId,
                        (byte) statusCode,
                        attrInfo.mAttrHandle,
                        attrInfo.mOffset,
                        dataToSend,
                        (byte) 0,
                        false);
    } catch (Throwable t)
    {
        Log.e("BleServerService", "sendResponse(): error", t);
    }
}
```

（7）描述低功耗蓝牙服务的特性

在 Broadcom 公司提供的源码中，文件 BleCharacteristic.java 可以描述低功耗蓝牙服务的特性。在特性中包含了描述符、实际值和元数据，提供了表现格式或便于阅读值的描述。文

件 BleCharacteristic.java 的主要实现代码如下。

```
public class BleCharacteristic extends BleAttribute
        implements Parcelable
{
    private static final String TAG = "BleCharacteristic";
    private HashMap<BleGattID, BleDescriptor> mDescriptorMap = new HashMap<BleGattID,
BleDescriptor>();
    private ArrayList<BleDescriptor> mDirtyDescQueue = new ArrayList<BleDescriptor>();
    private int mProp;
    private int mWriteType;
    private byte mAuthReq;
    private int mPermission = 0;
    /** @hide */
    @SuppressWarnings({
            "rawtypes", "unchecked"
    })
    public static final Parcelable.Creator<BleCharacteristic> CREATOR = new Parcelable.Creator()
    {
        @Override
        public BleCharacteristic createFromParcel(Parcel source) {
            return new BleCharacteristic(source);
        }
    };
    /*获取 GATT 的 ID 值*/
    private BleGattID getBleGattId(int handle)
    {
        for (Map.Entry<BleGattID, Integer> entry : mHandleMap.entrySet()) {
                if (handle == entry.getValue().intValue()) {
                return entry.getKey();
            }
        }
        return null;
    }
    /**
     *返回该特征的实例的 ID。
     *实例 ID 的 BLE 配置文件和服务用于标识属于一个给定的实例的服务或轮廓的特征。
     */
    public int getInstanceID()
    {
        return mID.getInstanceID();
    }
    /**
     *指定一个实例 ID 的这一特性
     *
     * @see {@link #getInstanceID()}
     */
```

```
public void setInstanceID(int instanceID)
{
    mID.setInstanceId(instanceID);
}
/**
 *根据特性向一个给定的偏移量设置原始值的字节
 */
public byte setValue(byte[] value, int offset, int len, int handle, int totalsize,
        String address)
{
    int uuid = -1;
    int uuidType = -1;
    Log.e("BleCharacteristic", "#### handle is " + handle + " total size is "
            + totalsize);
    BleGattID gattUuid = getBleGattId(handle);
            if (gattUuid == null) {
        Log.e("BleCharacteristic", "setValue: Invalid handle");
        return BleConstants.GATT_INVALID_HANDLE;
    }
    if (gattUuid.equals(mID)) {
        Log.i("BleCharacteristic", "##Writing a characteristic value..");
        Log.i("BleCharacteristic", "##offset=" + offset + " mMaxLength="
                + mMaxLength + " totalsize=" + totalsize);
        return setValue(value, offset, len, gattUuid, totalsize, address);
    }
    BleDescriptor descObj = mDescriptorMap.get(gattUuid);
    if (descObj != null) {
        Log.i("BleCharacteristic", "##Writing descriptor value..");
        Log.i("BleCharacteristic",
                "##offset=" + offset + " mMaxSize=" + descObj.getMaxLength() + " totalsize="
                        + totalsize + "desc uuid =" + descObj.getID());
        if (offset > descObj.getMaxLength())
            return BleConstants.GATT_INVALID_OFFSET;
        if (offset + totalsize > descObj.getMaxLength())
            return BleConstants.GATT_INVALID_ATTR_LEN;
        Log.i("BleCharacteristic", "find the user defined descriptor ");
        return descObj.setValue(value, offset, len, gattUuid, totalsize, address);
    }
    Log.e("BleCharacteristic", "Failed to write the value correctly!!!");
    return -127;
}
```

（8）低功耗描述符

在 Broadcom 公司提供的源码中，文件 BleDescriptor.java 是 BleCharacteristic 的一部分，定义了一个低功耗描述符。文件 BleDescriptor.java 的主要实现代码如下。

```
public class BleDescriptor extends BleAttribute
```

```
        implements Parcelable
{
    private static final String TAG = "BleDescriptor";
    private BleCharacteristic mCharObj;
    protected HashMap<String, Integer> mClientcfgMap = new HashMap();
    /** @hide */
    @SuppressWarnings({
            "unchecked", "rawtypes"
    })
    public static final Parcelable.Creator<BleDescriptor> CREATOR = new Parcelable.Creator()
    {
        public BleDescriptor createFromParcel(Parcel source) {
            return new BleDescriptor(source);
        }
        public BleDescriptor[] newArray(int size)
        {
            return new BleDescriptor[size];
        }
    };

    /**
     *从一个给定的偏移设置原始值的字节的描述符
     *
     * @return {@link BleConstants#GATT_SUCCESS if successful}
     */
    @Override
    public byte setValue(byte[] value, int offset, int length, BleGattID gattUuid,
            int totalSize, String address)
    {
        int uuidType = gattUuid.getUuidType();
        int uuid = -1;
        Log.e("BleDescriptor", "#### UUID type=" + gattUuid.getUuidType());
                if (uuidType == 2) {
            uuid = gattUuid.getUuid16();
                    if (uuid == -1) {
                Log.e("BleDescriptor", "setValue: Invalid handle (UUID16 not found)");
                return 1;
            }
        }
                if (uuid == 10500) {
            Log.i("BleDescriptor", "##Writing a Presentation format..");
                } else if (uuid == 10498) {
            Log.i("BleDescriptor", "##Writing a characteristic client config");
            if (totalSize > this.mMaxLength)
                return 13;
            int valueInt = 0;
```

```
                for (int i = 0; i < length; i++) {
                    int shift = (length - 1 - i) * 8;
                    valueInt += ((value[i] & 0xFF) << shift);
                }
                this.mClientcfgMap.put(address, Integer.valueOf(valueInt));
            } else if (gattUuid.equals(this.mID)) {
                Log.i("BleDescriptor", "##Writing a descriptor value..");
                Log.i("BleDescriptor", "##offset=" + offset + " mMaxLength=" + this.mMaxLength
                        + " length=" + length);
                super.setValue(value, offset, length, gattUuid, totalSize, address);
            }
            this.mDirty = true;
            return 0;
        }
}
```

（9）标识低功耗蓝牙规范、服务和特性

在 Broadcom 公司提供的源码中，文件 BleGattID.java 定义了一个标识低功耗蓝牙规范、服务和特性的类，此类使用 16 位或 128 位的 UUID 来标识一个给定的低功耗蓝牙实体，这个实体包含规范、服务和特性。文件 BleGattID.java 的主要实现代码如下。

```
/**
 *标识一个蓝牙 GATT 特性或属性
 */
public final class BleGattID extends BluetoothGattID
        implements Parcelable
{
    private static final String BASE_UUID_TPL = "%08x-0000-1000-8000-00805f9b34fb";
    @SuppressWarnings({
            "rawtypes", "unchecked"
    })
    public static final Parcelable.Creator<BleGattID> CREATOR = new Parcelable.Creator() {
        public BleGattID createFromParcel(Parcel source) {
            int instId = source.readInt();
            int type = source.readInt();
            int serviceType = source.readInt();
                    if (type == 16) {
                String sUuid = source.readString();
                return new BleGattID(instId, sUuid, serviceType);
            }
            int uuid = source.readInt();
            return new BleGattID(instId, uuid, serviceType);
        }
        public BleGattID[] newArray(int size)
        {
            return new BleGattID[size];
```

```
        }
    };
```

（10）为远程蓝牙设备提供额外信息

在Broadcom公司提供的源码中，文件BleAdapter.java为远程蓝牙设备提供了额外的信息，能够判断远程设备是否是低功耗设备、BR/EDR 传统蓝牙设备或双模设备。文件BleAdapter.java的主要实现代码如下。

```
/**
 *提供帮助的功能和相关的常数扩展蓝牙功能的低能耗信息。
 */
public class BleAdapter
{
    private static final String TAG = "BleAdapter";
    private static final boolean D = true;
    private static final int API_LEVEL = 5;
    private static IBluetoothGatt mService;
    private GattServiceConnection mSvcConn;
    private Context mContext;
    /**
     * 设置远程 ACTION_FOUND 设备的额外信息
     *
     * @see {@link #DEVICE_TYPE_BREDR}, {@link #DEVICE_TYPE_BLE},
     *          {@link #DEVICE_TYPE_DUMO}
     */
    public static final String EXTRA_DEVICE_TYPE = "android.bluetooth.device.extra.DEVICE_TYPE";
    public static final byte DEVICE_TYPE_BREDR = 1;
    public static final byte DEVICE_TYPE_BLE = 2;
    public static final byte DEVICE_TYPE_DUMO = 3;
    public static final String ACTION_UUID = "android.bluetooth.le.device.action.UUID";
    public static final String EXTRA_UUID = "android.bluetooth.le.device.extra.UUID";
    public static final String EXTRA_DEVICE = "android.bluetooth.le.device.extra.DEVICE";
    private static boolean startService() {
        if (mService != null)
            return true;
        IBinder service = ServiceManager.getService(BleConstants.BLUETOOTH_LE_SERVICE);
        if (service != null)
            mService = IBluetoothGatt.Stub.asInterface(service);
        return mService != null;
    }
    /**
     * 构建一种新的 BleAdapter 对象
     */
    public BleAdapter(Context ctx) {
        this.mContext = ctx;
                if (startService()==false)
```

```
            throw new RuntimeException("failed connecting to service");
        this.init();
    }

    /**
     * 启动远程设备中的蓝牙服务，发现使用{@link #ACTION_UUID}的意图
     */
    public static boolean getRemoteServices(String deviceAddress)
    {
        if (!startService())
            throw new RuntimeException("service not available");
        BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();
                if (adapter == null)
            return false;
        try {
            mService.getUUIDs(deviceAddress);
            return true;
        } catch (RemoteException e) {
            e.printStackTrace();
            if (D)
                Log.e(TAG, "error", e);
        }
        return false;
    }
```

（11）保存和 GATT 相关的常量

在 Broadcom 公司提供的源码中，文件 BleConstants.java 定义保存了各种和 GATT 相关的常量，这些常量用于表示各种和实现低功耗功能函数的属性和返回值。文件 BleConstants.java 的主要实现代码如下。

```
public abstract class BleConstants
{
    public static final int GATT_UNDEFINED = -1;
    public static final int GATT_SERVICE_CREATION_SUCCESS = 0;
    public static final int GATT_SERVICE_CREATION_FAILED = 1;
    public static final int GATT_SERVICE_START_SUCCESS = 2;
    public static final int GATT_SERVICE_START_FAILED = 3;
    public static final int GATT_SERVICE_STOPPED = 4;
    public static final int SERVICE_UNAVAILABLE = 1;
    public static final int GATT_SERVICE_PRIMARY = 0;
    public static final int GATT_SERVICE_SECONDARY = 1;
    public static final int GATT_SERVER_PROFILE_INITIALIZED = 0;
    public static final int GATT_SERVER_PROFILE_UP = 1;
    public static final int GATT_SERVER_PROFILE_DOWN = 2;
    public static final int GATT_SUCCESS = 0;
    public static final int GATT_INVALID_HANDLE = 1;
```

```
public static final int GATT_READ_NOT_PERMIT = 2;
public static final int GATT_WRITE_NOT_PERMIT = 3;
public static final int GATT_INVALID_PDU = 4;
public static final int GATT_INSUF_AUTHENTICATION = 5;
public static final int GATT_REQ_NOT_SUPPORTED = 6;
public static final int GATT_INVALID_OFFSET = 7;
public static final int GATT_INSUF_AUTHORIZATION = 8;
public static final int GATT_PREPARE_Q_FULL = 9;
public static final int GATT_NOT_FOUND = 10;
public static final int GATT_NOT_LONG = 11;
public static final int GATT_INSUF_KEY_SIZE = 12;
public static final int GATT_INVALID_ATTR_LEN = 13;
public static final int GATT_ERR_UNLIKELY = 14;
public static final int GATT_INSUF_ENCRYPTION = 15;
public static final int GATT_UNSUPPORT_GRP_TYPE = 16;
public static final int GATT_INSUF_RESOURCE = 17;
public static final int GATT_ILLEGAL_PARAMETER = 135;
public static final int GATT_NO_RESOURCES = 128;
public static final int GATT_INTERNAL_ERROR = 129;
public static final int GATT_WRONG_STATE = 130;
public static final int GATT_DB_FULL = 131;
public static final int GATT_BUSY = 132;
public static final int GATT_ERROR = 133;
public static final int GATT_CMD_STARTED = 134;
public static final int GATT_PENDING = 136;
public static final int GATT_AUTH_FAIL = 137;
public static final int GATT_MORE = 138;
public static final int GATT_INVALID_CFG = 139;
public static final byte GATT_AUTH_REQ_NONE = 0;
public static final byte GATT_AUTH_REQ_NO_MITM = 1;
public static final byte GATT_AUTH_REQ_MITM = 2;
public static final byte GATT_AUTH_REQ_SIGNED_NO_MITM = 3;
public static final byte GATT_AUTH_REQ_SIGNED_MITM = 4;
public static final int GATT_PERM_READ = 1;
public static final int GATT_PERM_READ_ENCRYPTED = 2;
public static final int GATT_PERM_READ_ENC_MITM = 4;
public static final int GATT_PERM_WRITE = 16;
public static final int GATT_PERM_WRITE_ENCRYPTED = 32;
public static final int GATT_PERM_WRITE_ENC_MITM = 64;
public static final int GATT_PERM_WRITE_SIGNED = 128;
public static final int GATT_PERM_WRITE_SIGNED_MITM = 256;
public static final byte GATT_CHAR_PROP_BIT_BROADCAST = 1;
public static final byte GATT_CHAR_PROP_BIT_READ = 2;
public static final byte GATT_CHAR_PROP_BIT_WRITE_NR = 4;
public static final byte GATT_CHAR_PROP_BIT_WRITE = 8;
public static final byte GATT_CHAR_PROP_BIT_NOTIFY = 16;
```

```
public static final byte GATT_CHAR_PROP_BIT_INDICATE = 32;
public static final byte GATT_CHAR_PROP_BIT_AUTH = 64;
public static final byte GATT_CHAR_PROP_BIT_EXT_PROP = -128;
public static final byte SVC_INF_INVALID = -1;
public static final int GATTC_TYPE_WRITE_NO_RSP = 1;
public static final int GATTC_TYPE_WRITE = 2;
public static final int GATT_FORMAT_RES = 0;
public static final int GATT_FORMAT_BOOL = 1;
public static final int GATT_FORMAT_2BITS = 2;
public static final int GATT_FORMAT_NIBBLE = 3;
public static final int GATT_FORMAT_UINT8 = 4;
public static final int GATT_FORMAT_UINT12 = 5;
public static final int GATT_FORMAT_UINT16 = 6;
public static final int GATT_FORMAT_UINT24 = 7;
public static final int GATT_FORMAT_UINT32 = 8;
public static final int GATT_FORMAT_UINT48 = 9;
public static final int GATT_FORMAT_UINT64 = 10;
public static final int GATT_FORMAT_UINT128 = 11;
public static final int GATT_FORMAT_SINT8 = 12;
public static final int GATT_FORMAT_SINT12 = 13;
public static final int GATT_FORMAT_SINT16 = 14;
public static final int GATT_FORMAT_SINT24 = 15;
public static final int GATT_FORMAT_SINT32 = 16;
public static final int GATT_FORMAT_SINT48 = 17;
public static final int GATT_FORMAT_SINT64 = 18;
public static final int GATT_FORMAT_SINT128 = 19;
public static final int GATT_FORMAT_FLOAT32 = 20;
public static final int GATT_FORMAT_FLOAT64 = 21;
public static final int GATT_FORMAT_SFLOAT = 22;
public static final int GATT_FORMAT_FLOAT = 23;
public static final int GATT_FORMAT_DUINT16 = 24;
public static final int GATT_FORMAT_UTF8S = 25;
public static final int GATT_FORMAT_UTF16S = 26;
public static final int GATT_FORMAT_STRUCT = 27;
```

到此为止，Broadcom 公司推出的低功耗蓝牙协议栈 BlueDroid 的开发文档和 API 源码分析结束。本书只是分析了主要的模块类，其他的类的实现代码的功能和原理请读者参阅其源码中的注释说明。

9.4　总结和蓝牙相关的类

经过本章前面内容的学习，已经了解了 Android 系统中蓝牙的基本知识。根据上述从底层到应用的学习，了解了 Android 蓝牙系统的工作原理和机制。在本节的内容中，将详细讲解 Android 蓝牙系统中的相关知识。

9.4.1 BluetoothSocket 类

1．定义

类 BluetoothSocket 的格式如下。

```
public static class Gallery.LayoutParams extends ViewGroup.LayoutParams
```

类 BluetoothSocket 的结构如下。

```
java.lang.Object
android.view. ViewGroup.LayoutParams
android.widget.Gallery.LayoutParams
```

2．公共方法

（1）public void close ()

功能：马上关闭该端口并且释放所有相关的资源。在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。

异常：IOException。

（2）public void connect ()

功能：尝试连接到远程设备。如果该方法没有返回异常值，则表示该端口已经建立。在蓝牙适配器中，设备的查找工作是一个烦琐的过程，肯定会降低一个设备的连接效率。此外的设备查询工作并不是由活动所管理的，而是作为一个系统服务来运行的，所以即使它不能直接请求一个查询，应用程序也总会调用 cancelDiscovery()方法。close()方法可以用来放弃从另一线程而来的调用。

异常：IOException，表示一个错误，例如连接失败。

（3）public InputStream getInputStream ()

功能：通过连接的端口获得输入数据流。即使该端口未连接，也会返回该输入的数据流。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立为止。

返回值：输入流。

异常：IOException。

（4）public OutputStream getOutputStream ()

功能：通过连接的端口获得输出数据流。即使该端口未连接，该输出数据流也会返回。不过在该数据流上的操作将抛出异常，直到相关的连接已经建立。

返回值：输出流。

异常：IOException。

（5）public BluetoothDevice getRemoteDevice ()

功能：获得该端口正在连接或者已经连接的远程设备。

返回值：远程设备。

9.4.2 BluetoothServerSocket 类

1．定义

类 BluetoothServerSocket 的格式如下。

```
public final class BluetoothServerSocket extends Object implements Closeable
```

类 BluetoothServerSocket 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothServerSocket
```

2．公共方法

（1）public BluetoothSocketaccept (int timeout)

功能：阻塞直到在超时时间内连接建立。在一个成功建立的连接上返回一个已连接的 BluetoothSocket 类。每当返回该调用的时候，它可以再使调用去接收以后新来的连接。同样，close()方法可以用来放弃从另一线程来的调用。

参数 timeout：表示阻塞超时时间。

返回值：已连接的 BluetoothSocket。

异常：IOException，表示出现错误，比如该调用被放弃或超时。

（2）public void close ()

功能：马上关闭端口，并释放所有相关的资源。在其他线程的该端口中引起阻塞，从而使系统马上抛出一个 IO 异常。关闭 BluetoothServerSocket，不会关闭接受自 accept()的任意 BluetoothSocket。

异常：IOException。

9.4.3　BluetoothAdapter 类

1．定义

类 BluetoothAdapter 的格式如下。

```
public final class BluetoothAdapter extends Object
```

类 BluetoothAdapter 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothAdapter
```

注意：大部分方法需要 BLUETOOTH 权限，还有一些方法同时需要 BLUETOOTH_ADMIN 权限。

2．常量

（1）String ACTION_DISCOVERY_FINISHED

广播事件：本地蓝牙适配器已经完成设备的搜寻过程。需要 BLUETOOTH 权限来接收。

常量值：android.bluetooth.adapter.action.DISCOVERY_FINISHED。

（2）String ACTION_DISCOVERY_STARTED

广播事件：本地蓝牙适配器已经开始了对远程设备的搜寻过程。它通常涉及到一个大概 12 秒的查询扫描过程，紧跟着是一个对每个获取到的蓝牙名称的新设备的页面扫描。用户会发现一个把 ACTION_FOUND 常量通知为远程蓝牙设备的注册。

常量值：android.bluetooth.adapter.action.DISCOVERY_STARTED。

（3）String ACTION_LOCAL_NAME_CHANGED

广播活动：本地蓝牙适配器已经更改了它的蓝牙名称。该名称对远程蓝牙设备是可见的，它总是包含了一个带有名称的 EXTRA_LOCAL_NAME 附加域。需要 BLUETOOTH 权限来接收。

常量值：android.bluetooth.adapter.action.LOCAL_NAME_CHANGED"

（4）String ACTION_REQUEST_DISCOVERABLE

Activity 活动：显示一个请求被搜寻模式的系统活动。如果未打开当前蓝牙模块，该活动将请求用户打开蓝牙模块，被搜寻模式和 SCAN_MODE_CONNECTABLE_DISCOVERABLE 等价。当远程设备执行查找进程的时候，它允许其发现该蓝牙适配器。从隐私安全考虑，Android 不会将被搜寻模式设置为默认状态。

Android 运用回收方法 onActivityResult(int, int, Intent)来传递该活动结果的通知。被搜寻的时间（以秒为单位）将通过 resultCode 值来显示。如果用户拒绝被搜寻，或者设备产生了错误，则通过 RESULT_CANCELED 值来显示。

当扫描模式变化时，应用程序可以通过 ACTION_SCAN_MODE_CHANGED 值来监听全局的消息通知。比如，当设备停止被搜寻以后，该消息可以被系统通知给应用程序。需要 BLUETOOTH 权限。

常量值：android.bluetooth.adapter.action.REQUEST_DISCOVERABLE

（5）String ACTION_REQUEST_ENABLE

Activity 活动：显示一个允许用户打开蓝牙模块的系统活动。当打开蓝牙模块后，或者当用户决定不打开蓝牙模块时，系统活动将返回该值。Android 运用回收方法 onActivityResult(int, int, Intent)来传递该活动结果的通知。如果蓝牙模块被打开，将通过 resultCode 值 RESULT_OK 来显示。如果用户拒绝该请求，或者设备产生了错误，则通过 RESULT_CANCELED 值来显示。每当蓝牙模块被打开或者关闭时，应用程序可以通过 ACTION_STATE_CHANGED 值来监听全局的消息通知。需要 BLUETOOTH 权限。

常量值：android.bluetooth.adapter.action.REQUEST_ENABLE。

（6）String ACTION_SCAN_MODE_CHANGED

广播活动：指明蓝牙扫描模块或者本地适配器已经发生变化，它总是包含 EXTRA_SCAN_MODE 和 EXTRA_PREVIOUS_SCAN_MODE。这两个附加域各自包含了新的和旧的扫描模式。需要 BLUETOOTH 权限

常量值：android.bluetooth.adapter.action.SCAN_MODE_CHANGED。

（7）String ACTION_STATE_CHANGED

广播活动：本来的蓝牙适配器的状态已经改变，例如蓝牙模块已经被打开或者关闭。它总是包含 EXTRA_STATE 和 EXTRA_PREVIOUS_STATE。这两个附加域各自包含了新的和旧的状态。需要 BLUETOOTH 权限接收

常量值：android.bluetooth.adapter.action.STATE_CHANGED。

（8）int ERROR

功能：标记该类的错误值。确保和该类中的任意其他整数常量不相等。它为需要一个标记错误值的函数提供了便利。例如：

Intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, BluetoothAdapter.ERROR)

常量值：-2147483648 (0x80000000)

（9）String EXTRA_DISCOVERABLE_DURATION

功能：试图在 ACTION_REQUEST_DISCOVERABLE 常量中作为一个可选的整型附加域，来为短时间内的设备发现请求一个特定的持续时间。默认值为 120 秒，超过 300 秒的请求将被限制。这些值是可以变化的。

常量值：android.bluetooth.adapter.extra.DISCOVERABLE_DURATION。

（10）String EXTRA_LOCAL_NAME

功能：试图在 ACTION_LOCAL_NAME_CHANGED 常量中作为一个字符串附加域，来请求本地蓝牙的名称。

常量值：android.bluetooth.adapter.extra.LOCAL_NAME。

（11）String EXTRA_PREVIOUS_SCAN_MODE

功能：试图在 ACTION_SCAN_MODE_CHANGED 常量中作为一个整型附加域，来请求以前的扫描模式。可能的取值如下。

- SCAN_MODE_NONE。
- SCAN_MODE_CONNECTABLE。
- SCAN_MODE_CONNECTABLE_DISCOVERABLE。

常量值：android.bluetooth.adapter.extra.PREVIOUS_SCAN_MODE。

（12）String EXTRA_PREVIOUS_STATE

功能：试图在 ACTION_STATE_CHANGED 常量中作为一个整型附加域，来请求以前的供电状态。可能的取值如下。

- STATE_OFF。
- STATE_TURNING_ON。
- STATE_ON。
- STATE_TURNING_OFF。

常量值：android.bluetooth.adapter.extra.PREVIOUS_STATE。

（13）String EXTRA_SCAN_MODE

功能：试图在 ACTION_SCAN_MODE_CHANGED 常量中作为一个整型附加域，来请求当前的扫描模式。可能的取值如下。

- SCAN_MODE_NONE。
- SCAN_MODE_CONNECTABLE。
- SCAN_MODE_CONNECTABLE_DISCOVERABLE。

常量值：android.bluetooth.adapter.extra.SCAN_MODE。

（14）String EXTRA_STATE

功能：试图在 ACTION_STATE_CHANGED 常量中作为一个整型附加域，来请求当前的供电状态。可能的取值如下。

- STATE_OFF。
- STATE_TURNING_ON。

- STATE_ON。
- STATE_TURNING_OFF。

常量值：android.bluetooth.adapter.extra.STATE。

（15）int　SCAN_MODE_CONNECTABLE

功能：指明在本地蓝牙适配器中，查询扫描功能失效，但页面扫描功能有效。因此该设备不能被远程蓝牙设备发现，但如果以前曾经发现过该设备，则远程设备可以对其进行连接。

常量值：21 (0x00000015)。

（16）int　SCAN_MODE_CONNECTABLE_DISCOVERABLE

功能：指明在本地蓝牙适配器中，查询扫描功能和页面扫描功能都有效。因此该设备既可以被远程蓝牙设备发现，也可以被其连接。

常量值：23 (0x00000017)。

（17）int　SCAN_MODE_NONE

功能：指明在本地蓝牙适配器中，查询扫描功能和页面扫描功能都失效。因此该设备既不可以被远程蓝牙设备发现，也不可以被其连接。

常量值：20 (0x00000014)。

（18）int　STATE_OFF

功能：指明本地蓝牙适配器模块已经关闭。

常量值：10 (0x0000000a)。

（19）int　STATE_ON

功能：指明本地蓝牙适配器模块已经打开，并且准备被使用。

（20）int　STATE_TURNING_OFF

功能：指明正在关闭本地蓝牙适配器模块，本地客户端可以立刻尝试友好地断开任意的外部连接。

常量值：13 (0x0000000d)。

（21）int　STATE_TURNING_ON

功能：指明正在打开本地蓝牙适配器模块，本地客户在尝试使用这个适配器之前需要为STATE_ON 的状态而等待。

常量值：11 (0x0000000b)。

3．公共方法

（1）public boolean cancelDiscovery ()

功能：取消当前的设备发现查找进程，需要 BLUETOOTH_ADMIN 权限。因为对蓝牙适配器而言，查找工作需要消耗大量的能耗，因此这个方法必须在尝试连接到远程设备前使用connect()方法进行调用。

返回值：成功则返回 true，有错误则返回 false。

（2）public static boolean checkBluetoothAddress (String address)

功能：验证蓝牙地址，字母必须为大写才有效。

参数 address：字符串形式的蓝牙模块地址。

返回值：地址正确则返回 true，否则返回 false。

（3）public boolean disable ()

功能：关闭本地蓝牙适配器，不能在没有明确关闭蓝牙的用户动作中使用。没有用户的直接同意，蓝牙永远不能被禁止。这个 disable()方法只提供了一个应用，该应用包含了一个改变系统设置的用户界面，例如“电源控制”应用。

9.4.4　BluetoothClass.Service 类

类 BluetoothClass.Service 的格式如下。

```
public static final class BluetoothClass.Service extends Object
```

类 BluetoothClass.Service 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothClass.Service
```

类 BluetoothClass.Service 用于定义所有的服务类常量，任意的 BluetoothClass 由 0 或多个服务类编码组成。在类 BluetoothClass.Service 中包含了如下的常量。

- int AUDIO。
- int CAPTURE。
- int INFORMATION。
- int LIMITED_DISCOVERABILITY。
- int NETWORKING。
- int OBJECT_TRANSFER。
- int POSITIONING。
- int RENDER。
- int TELEPHONY。

9.4.5　BluetoothClass.Device.Major 类

类 BluetoothClass.Device.Major 的格式如下。

```
public static class BluetoothClass.Device.Major extends Object
```

类 BluetoothClass.Device.Major 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothClass.Device.Major
```

类 BluetoothClass.Device.Major 用于定义所有的主要设备类常量，各个常量如下。

- int　AUDIO_VIDEO。
- int　COMPUTER。
- int　HEALTH。
- int　IMAGING。
- int　MISC。

- int NETWORKING。
- int PERIPHERAL。
- int PHONE。
- int TOY。
- int UNCATEGORIZED。
- int WEARABLE。

9.4.6 BluetoothClass.Device 类

类 BluetoothClass.Device 的格式如下。

```
public final class BluetoothClass.Device extends Object
```

类 BluetoothClass.Device 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothClass.Device
```

类 BluetoothClass.Device 用于定义所有的设备类的常量，每个 BluetoothClass 实现了两种设备类型的编码，分别是主要设备和次要设备。里面的常量分别代表主要设备和次要设备，其中 BluetoothClass.Device.Major 的常量只能代表主要设备类。

BluetoothClass.Device 有一个内部类，此内部类定义了所有的主要设备类常量。内部类的定义格式如下。

```
class BluetoothClass.Device.Major
```

9.4.7 BluetoothClass 类

1．定义

类 BluetoothClass 的格式如下。

```
public final class BluetoothClass extends Object implements Parcelable
```

类 BluetoothClass 的结构如下。

```
java.lang.Object
android.bluetooth.BluetoothClass
```

类 BluetoothClass 代表了一个描述了设备通用特性和功能的蓝牙类。比如一个蓝牙类会指定诸如电话、计算机或耳机的通用设备类型，可以提供诸如音频或者电话的服务。每个蓝牙类都由 0 个或更多的服务类，以及一个设备类组成。设备类将被分解成主要和较小的设备类部分。

使用 getBluetoothClass()方法可以获取为远程设备所提供的类。

2．内部类

类 BluetoothClass 有如下的两个内部类。

（1）class BluetoothClass.Device：用于定义所有设备类的常量。

（2）class BluetoothClass.Service：用于定义所有服务类的常量。

9.5　实战演练——开发一个蓝牙控制器

本实例为在 Android 设备中开发一个蓝牙控制器，通过这个控制器可以实现如下的功能。

- 打开蓝牙。
- 关闭蓝牙。
- 允许搜索。
- 开始搜索。
- 客户端。
- 服务器端。
- OBEX 服务器。

题　　目	目　　的	源 码 路 径
实例 9-1	开发一个 Android 蓝牙控制器	daima\9\Activity01

9.5.1　界面布局

本实例主界面布局文件 main.xml，其主要实现代码如下。

```
<?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:padding="10dip">
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="打开蓝牙"
            android:onClick="onEnableButtonClicked" />
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="关闭蓝牙"
            android:onClick="onDisableButtonClicked" />
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="允许搜索"
            android:onClick="onMakeDiscoverableButtonClicked" />
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="开始搜索"
            android:onClick="onStartDiscoveryButtonClicked" />
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="客户端"
            android:onClick="onOpenClientSocketButtonClicked" />
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="服务器端"
            android:onClick="onOpenServerSocketButtonClicked" />
```

```
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="OBEX 服务器"
            android:onClick="onOpenOBEXServerSocketButtonClicked" />
</LinearLayout>
```

执行之后的界面效果如图 9-5 所示。

图 9-5　执行效果

服务器端的界面布局文件是 server_socket.xml，主要实现代码如下。

```
<?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:padding="10dip">
    <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="Stop server"
            android:onClick="onButtonClicked" />
    <ListView android:id="@+id/android:list" android:layout_width="fill_parent"
            android:layout_height="fill_parent" />
</LinearLayout>
```

其他几个界面的布局文件和上述文件类似，为节省本书篇幅，将不再一一列出。

9.5.2　响应单击按钮

编写主界面的程序文件 Activity01.java，功能是根据用户单击屏幕中的按钮来调用对应的处理函数，例如单击“服务器端”按钮会执行函数 onOpenServerSocketButtonClicked(View view)。文件 Activity01.java 的主要实现代码如下。

```
public class Activity01 extends Activity
{
    /* 取得默认的蓝牙适配器 */
```

```
        private BluetoothAdapter _bluetooth  = BluetoothAdapter.getDefaultAdapter();
/*  请求打开蓝牙  */
        private static final int REQUEST_ENABLE= 0x1;
/*  请求能够被搜索  */
        private static final int REQUEST_DISCOVERABLE = 0x2;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
/*  开启蓝牙  */
public void onEnableButtonClicked(View view)
{
    //打开蓝牙
    _bluetooth.enable();
}
/*  关闭蓝牙  */
public void onDisableButtonClicked(View view)
{
    _bluetooth.disable();
}
/*  使设备能够被搜索  */
public void onMakeDiscoverableButtonClicked(View view)
{
    Intent enabler = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    startActivityForResult(enabler, REQUEST_DISCOVERABLE);
}
/*  开始搜索  */
public void onStartDiscoveryButtonClicked(View view)
{
    Intent enabler = new Intent(this, DiscoveryActivity.class);
    startActivity(enabler);
}
/*  客户端  */
public void onOpenClientSocketButtonClicked(View view)
{
    Intent enabler = new Intent(this, ClientSocketActivity.class);
    startActivity(enabler);
}
/*  服务端  */
public void onOpenServerSocketButtonClicked(View view)
{
    Intent enabler = new Intent(this, ServerSocketActivity.class);
    startActivity(enabler);
```

```
    }
    /* OBEX 服务器 */
    public void onOpenOBEXServerSocketButtonClicked(View view)
    {
        Intent enabler = new Intent(this, OBEXActivity.class);
        startActivity(enabler);
    }
}
```

9.5.3 和指定的服务器建立连接

编写程序文件 ClientSocketActivity.java，功能是创建一个 Socket 连接，以便和指定的服务器建立连接。文件 ClientSocketActivity.java 的主要实现代码如下。

```
public class ClientSocketActivity   extends Activity
{
    private static final String TAG = ClientSocketActivity.class.getSimpleName();
    private static final int REQUEST_DISCOVERY = 0x1;;
    private Handler _handler = new Handler();
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
        WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
        setContentView(R.layout.client_socket);
        if (!_bluetooth.isEnabled()) {
            finish();
            return;
        }
        Intent intent = new Intent(this, DiscoveryActivity.class);
        /* 提示选择一个要连接的服务器 */
        Toast.makeText(this, "select device to connect", Toast.LENGTH_SHORT).show();
        /* 跳转到搜索的蓝牙设备列表区，进行选择 */
        startActivityForResult(intent, REQUEST_DISCOVERY);
    }
    /* 选择了服务器之后进行连接 */
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode != REQUEST_DISCOVERY) {
            return;
        }
        if (resultCode != RESULT_OK) {
            return;
        }
        final BluetoothDevice device = data.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
        new Thread() {
            public void run() {
                /* 连接 */
```

```
                    connect(device);
                };
            }.start();
        }
        protected void connect(BluetoothDevice device) {
            BluetoothSocket socket = null;
            try {
                //创建一个 Socket 连接：只需要服务器在注册时的 UUID 号
                socket = device.createRfcommSocketToServiceRecord(UUID.fromString("a60f35f0-
b93a-11de-8a39-08002009c666"));
                //连接
                socket.connect();
            } catch (IOException e) {
                Log.e(TAG, "", e);
            } finally {
                if (socket != null) {
                    try {
                        socket.close();
                    } catch (IOException e) {
                        Log.e(TAG, "", e);
                    }
                }
            }
        }
            }
```

9.5.4　搜索附近的蓝牙设备

编写程序文件 DiscoveryActivity.java，功能是搜索设备附近的蓝牙设备，并在列表中显示搜索到的蓝牙设备。文件 DiscoveryActivity.java 的主要实现代码如下。

```
public class DiscoveryActivity    extends ListActivity
{
    private Handler _handler = new Handler();
    /*  取得默认的蓝牙适配器  */
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    /*  用来存储搜索到的蓝牙设备  */
    private List<BluetoothDevice> _devices = new ArrayList<BluetoothDevice>();
    /*  是否完成搜索  */
    private volatile boolean _discoveryFinished;
    private Runnable _discoveryWorkder = new Runnable() {
        public void run()
        {
            /*  开始搜索  */
            _bluetooth.startDiscovery();
            for (;;)
            {
```

```
                    if (_discoveryFinished)
                    {
                        break;
                    }
                    try
                    {
                        Thread.sleep(100);
                    }
                    catch (InterruptedException e){}
                }
            }
        };
        /**
         * 接收器
         * 当搜索蓝牙设备完成时调用
         */
        private BroadcastReceiver _foundReceiver = new BroadcastReceiver() {
            public void onReceive(Context context, Intent intent) {
                /* 从 intent 中取得搜索结果数据 */
                BluetoothDevice device = intent
                        .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                /* 将结果添加到列表中 */
                _devices.add(device);
                /* 显示列表 */
                showDevices();
            }
        };
        private BroadcastReceiver _discoveryReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent)
            {
                /* 卸载注册的接收器 */
                unregisterReceiver(_foundReceiver);
                unregisterReceiver(this);
                _discoveryFinished = true;
            }
        };
        protected void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND, Window
Manager.LayoutParams.FLAG_BLUR_BEHIND);
            setContentView(R.layout.discovery);
            /* 如果蓝牙适配器没有打开，则结果 */
            if (!_bluetooth.isEnabled())
            {
```

```
                finish();
                return;
            }
            /* 注册接收器 */
            IntentFilter discoveryFilter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_
FINISHED);
            registerReceiver(_discoveryReceiver, discoveryFilter);
            IntentFilter foundFilter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(_foundReceiver, foundFilter);
            /* 显示一个对话框,正在搜索蓝牙设备 */
            SamplesUtils.indeterminate(DiscoveryActivity.this, _handler, "Scanning...", _discovery
Workder, new OnDismissListener() {
                public void onDismiss(DialogInterface dialog)
                {
                    for (; _bluetooth.isDiscovering();)
                    {
                        _bluetooth.cancelDiscovery();
                    }
                    _discoveryFinished = true;
                }
            }, true);
        }
        /* 显示列表 */
        protected void showDevices()
        {
            List<String> list = new ArrayList<String>();
            for (int i = 0, size = _devices.size(); i < size; ++i)
            {
                StringBuilder b = new StringBuilder();
                BluetoothDevice d = _devices.get(i);
                b.append(d.getAddress());
                b.append('\n');
                b.append(d.getName());
                String s = b.toString();
                list.add(s);
            }
            final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.
simple_list_item_1, list);
            _handler.post(new Runnable() {
                public void run()
                {
                    setListAdapter(adapter);
                }
            });
        }
```

```
    protected void onListItemClick(ListView l, View v, int position, long id)
    {
        Intent result = new Intent();
        result.putExtra(BluetoothDevice.EXTRA_DEVICE, _devices.get(position));
        setResult(RESULT_OK, result);
        finish();
    }
}
```

9.5.5 建立和 OBEX 服务器的数据传输

编写程序文件 OBEXActivity.java，功能是实现与 OBEX 服务器的数据传输。OBEX 全称为 Object Exchange，中文对象交换，所以称之为对象交换协议。OBEX 协议通过使用“PUT”和“GET”命令实现在不同的设备、不同的平台之间方便、高效地交换信息。支持的设备广泛，例如 PC、PDA、电话、摄像头、自动答录机、计算器、数据采集器和手表等。文件 OBEXActivity.java 的主要实现代码如下。

```
public class OBEXActivity extends Activity
{
        private static final String TAG = "@MainActivity";
        private Handler _handler = new Handler();
        private BluetoothServerSocket _server;
        private BluetoothSocket _socket;
        private static final int OBEX_CONNECT  = 0x80;
        private static final int OBEX_DISCONNECT = 0x81;
        private static final int OBEX_PUT = 0x02;
        private static final int OBEX_PUT_END = 0x82;
        private static final int OBEX_RESPONSE_OK = 0xa0;
        private static final int OBEX_RESPONSE_CONTINUE = 0x90;
        private static final int BIT_MASK = 0x000000ff;
    Thread t = new Thread()
    {
        public void run()
        {
            try
            {
                _server = BluetoothAdapter.getDefaultAdapter().listenUsingRfcommWithService
Record("OBEX", null);
                new Thread()
                {
                    public void run()
                    {
                        Log.d("@Rfcom", "begin close");
                        try
                        {
                            _socket.close();
```

```
                                }
                                catch (IOException e)
                                {
                                        Log.e(TAG, "", e);
                                }
                                Log.d("@Rfcom", "end close");
                        };
                }.start();
                _socket = _server.accept();
                reader.start();
                Log.d(TAG, "shutdown thread");
        }
        catch (IOException e)
        {
                e.printStackTrace();
        }
    };
};
Thread reader = new Thread()
{
    public void run()
    {
        try
        {
            Log.d(TAG, "getting inputstream");
            InputStream inputStream = _socket.getInputStream();
            OutputStream outputStream = _socket.getOutputStream();
            Log.d(TAG, "got inputstream");
            int read = -1;
            byte[] bytes = new byte[2048];
            ByteArrayOutputStream baos = new ByteArrayOutputStream(bytes.length);
            while ((read = inputStream.read(bytes)) != -1)
            {
                baos.write(bytes, 0, read);
                byte[] req = baos.toByteArray();
                int op = req[0] & BIT_MASK;
                Log.d(TAG, "read:" + Arrays.toString(req));
                Log.d(TAG, "op:" + Integer.toHexString(op));
                switch (op)
                {
                    case OBEX_CONNECT:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_OK, 0, 7, 16, 0, 4,
0 });
                        break;
                    case OBEX_DISCONNECT:
                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_OK, 0, 3, 0 });
```

```
                                        break;
                                    case OBEX_PUT:
                                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
CONTINUE, 0, 3, 0 });
                                        break;
                                    case OBEX_PUT_END:
                                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
OK, 0, 3, 0 });
                                        break;

                                    default:
                                        outputStream.write(new byte[] { (byte) OBEX_RESPONSE_
OK, 0, 3, 0 });
                                }
                                Log.d(TAG, new String(baos.toByteArray(), "utf-8"));
                                baos = new ByteArrayOutputStream(bytes.length);
                            }
                            Log.d(TAG, new String(baos.toByteArray(), "utf-8"));
                        }
                        catch (IOException e)
                        {
                            e.printStackTrace();
                        }
                    };
                };
                private Thread put = new Thread() {
                    public void run()
                    {
                    };
                };
                public void onCreate(Bundle savedInstanceState)
                {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.obex_server_socket);
                    t.start();
                }
                protected void onActivityResult(int requestCode, int resultCode, Intent data)
                {
                    Log.d(TAG, data.getData().toString());
                    switch (requestCode)
                    {
                        case (1):
                                    if (resultCode == Activity.RESULT_OK)
                            {
                                Uri contactData = data.getData();
                                @SuppressWarnings("deprecation")
```

```
                    Cursor c = managedQuery(contactData, null, null, null, null);
                    for (; c.moveToNext();)
                    {
                        Log.d(TAG, "c1----------------------------------------");
                        dump(c);
                        Uri uri = Uri.withAppendedPath(data.getData(), ContactsContract.
Contacts.Photo.CONTENT_DIRECTORY);
                        @SuppressWarnings("deprecation")
                        Cursor c2 = managedQuery(uri, null, null, null, null);
                        for (; c2.moveToNext();)
                        {
                            Log.d(TAG, "c2----------------------------------------");
                            dump(c2);
                        }
                    }
                }
                break;
            }
        }
```

9.5.6　实现蓝牙服务器端的数据处理

编写文件 ServerSocketActivity.java，功能是实现蓝牙服务器端的数据处理，建立服务器端和客户端的连接和监听工作，其中的监听工作和停止服务器工作由独立的函数实现。文件 ServerSocketActivity.java 的主要实现代码如下。

```
public class ServerSocketActivity extends ListActivity
{
    /* 一些常量，代表服务器的名称 */
    public static final String PROTOCOL_SCHEME_L2CAP = "btl2cap";
    public static final String PROTOCOL_SCHEME_RFCOMM = "btspp";
    public static final String PROTOCOL_SCHEME_BT_OBEX = "btgoep";
    public static final String PROTOCOL_SCHEME_TCP_OBEX = "tcpobex";
    private static final String TAG = ServerSocketActivity.class.getSimpleName();
    private Handler _handler = new Handler();
    /* 取得默认的蓝牙适配器 */
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    /* 蓝牙服务器 */
    private BluetoothServerSocket _serverSocket;
            /* 线程——监听客户端的链接 */
    private Thread _serverWorker = new Thread() {
        public void run() {
            listen();
        };
    };
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
                WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
        setContentView(R.layout.server_socket);
        if (!_bluetooth.isEnabled()) {
            finish();
            return;
        }
        /* 开始监听 */
        _serverWorker.start();
    }
    protected void onDestroy() {
        super.onDestroy();
        shutdownServer();
    }
    protected void finalize() throws Throwable {
        super.finalize();
        shutdownServer();
    }
    /* 停止服务器 */
    private void shutdownServer() {
        new Thread() {
            public void run() {
                _serverWorker.interrupt();
                if (_serverSocket != null) {
                    try {
                        /* 关闭服务器 */
                        _serverSocket.close();
                    } catch (IOException e) {
                        Log.e(TAG, "", e);
                    }
                    _serverSocket = null;
                }
            };
        }.start();
    }
    public void onButtonClicked(View view) {
        shutdownServer();
    }
    protected void listen() {
        try {
            /* 创建一个蓝牙服务器
             * 参数分别：服务器名称、UUID
             */
            _serverSocket = _bluetooth.listenUsingRfcommWithServiceRecord(PROTOCOL_SCHEME_
RFCOMM,
                    UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));
```

```
        /* 客户端连线列表 */
        final List<String> lines = new ArrayList<String>();
        _handler.post(new Runnable() {
            public void run() {
                lines.add("Rfcomm server started...");
                ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                        ServerSocketActivity.this,
                        android.R.layout.simple_list_item_1, lines);
                setListAdapter(adapter);
            }
        });
        /* 接受客户端的连接请求 */
        BluetoothSocket socket = _serverSocket.accept();
        /* 处理请求内容 */
        if (socket != null) {
            InputStream inputStream = socket.getInputStream();
            int read = -1;
            final byte[] bytes = new byte[2048];
            for (; (read = inputStream.read(bytes)) > -1;) {
                final int count = read;
                _handler.post(new Runnable() {
                    public void run() {
                        StringBuilder b = new StringBuilder();
                        for (int i = 0; i < count; ++i) {
                            if (i > 0) {
                                b.append(' ');
                            }
                            String s = Integer.toHexString(bytes[i] & 0xFF);
                            if (s.length() < 2) {
                                b.append('0');
                            }
                            b.append(s);
                        }
                        String s = b.toString();
                        lines.add(s);
                        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                                ServerSocketActivity.this,
                                android.R.layout.simple_list_item_1, lines);
                        setListAdapter(adapter);
                    }
                });
            }
        }
    } catch (IOException e) {
        Log.e(TAG, "", e);
    } finally {
```

```
        }
    }
}
```

在文件 AndroidManifest.xml 中声明对蓝牙设备的使用权限，具体代码如下。

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

到此为止，整个实例全部实现，本实例需要在机器上进行测试。

第 10 章　开发 Wi-Fi 应用程序

Wi-Fi 是一种可以将个人计算机、手持设备（如 PDA、手机）等终端以无线的方式互相连接的技术。Wi-Fi 是一个无线网路通信技术的品牌，由 Wi-Fi 联盟（Wi-Fi Alliance）所持有。目的是改善基于 IEEE 802.11 标准的无线网路产品之间的互通性。在本章的内容中，将简要介绍在 Android 平台中开发 Wi-Fi 相关应用的基本知识。

10.1　了解 Wi-Fi 系统的结构

要想完全掌握 Wi-Fi 应用开发技术，需要从底层源码的分析开始，需要先了解它的底层结构。在本节的内容中，将简要讲解 Wi-Fi 系统底层结构的基本知识。

10.1.1　Wi-Fi 概述

在 Android 系统中存在了一个无线控制模块，打开方式是依次单击 Menu→Settings→Wireless$networks→Wi-Fi settings 命令，来到如图 10-1 所示的界面。

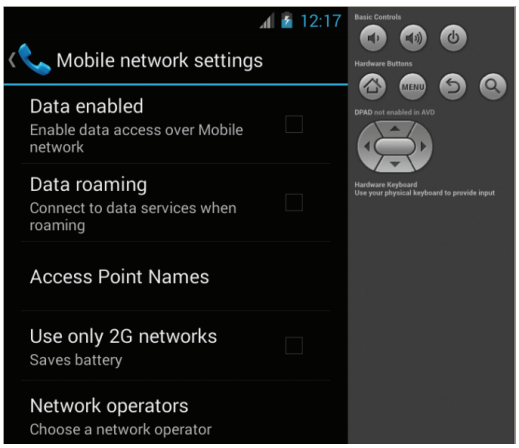


图 10-1　Wi-Fi 控制界面

10.1.2　Wi-Fi 层次结构

Wi-Fi 系统的上层接口包括数据部分和控制部分，数据部分通常是一个和以太网卡类似的网络设备，控制部分用于实现接入点操作和安全验证处理。

在软件层，Wi-Fi 系统包括 Linux 内核程序和协议，还包括本地部分、Java 框架类。Wi-Fi 系统向 Java 应用程序层提供了控制类的接口。

Android 平台中 Wi-Fi 系统的基本层次结构如图 10-2 所示。

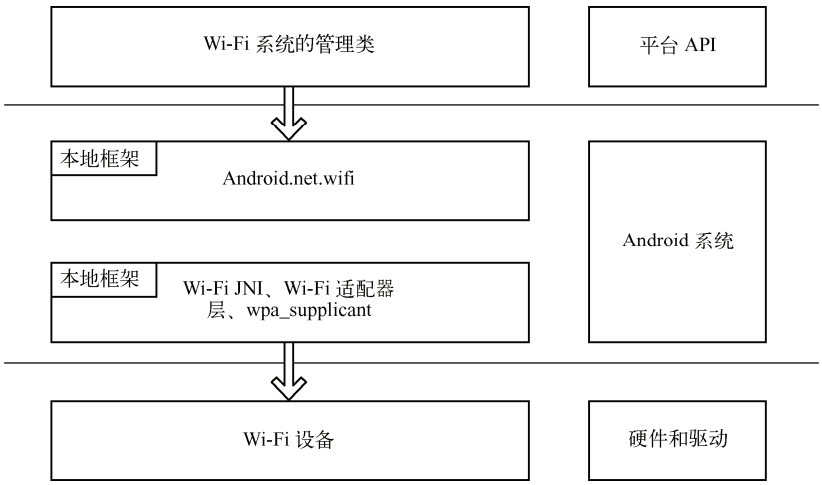


图 10-2　Wi-Fi 系统的层次结构

由图 10-2 可知，Android 的 Wi-Fi 系统从上到下主要包括 Java 框架类、Android 适配器库、wpa_supplicant 守护进程、驱动程序和协议，这几部分的系统结构如图 10-3 所示。

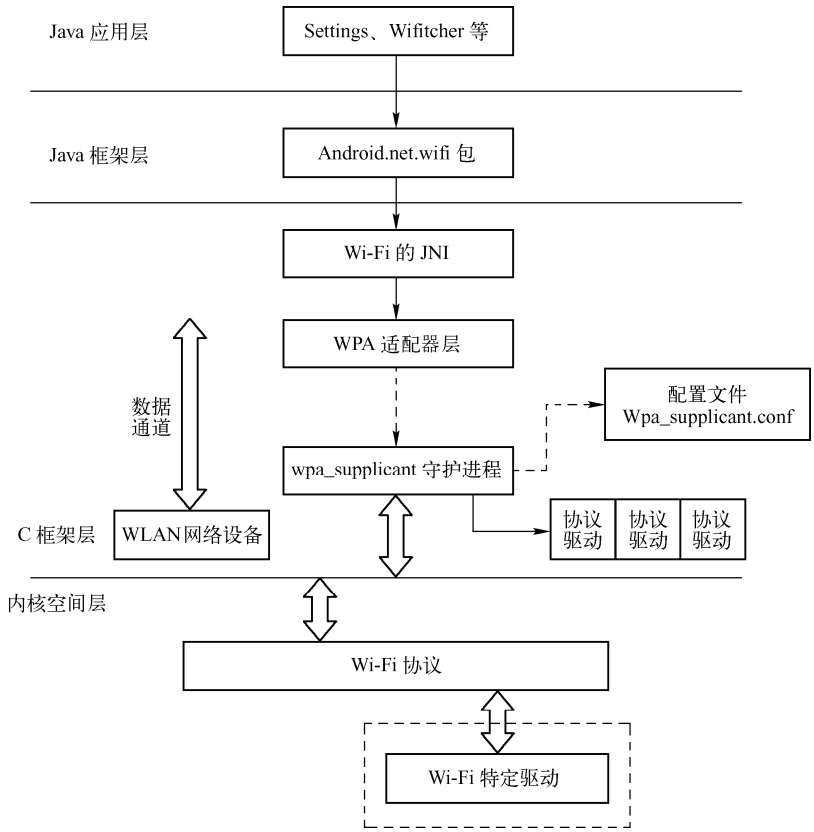


图 10-3　Wi-Fi 的系统结构

10.2　常用的 Wi-Fi 接口

在开发 Android 网络应用程序时，需要使用图 10-3 中的框架层提供的接口来实现 Wi-Fi 通信功能。其中最主要的接口有三类，分别是 WiFiManger、WifiService 和 WifiWatchdogService。在本节的内容中，将详细讲解这三类 Wi-Fi 接口的基本知识。

10.2.1　WifiManger 接口

WifiManger 为 Wi-Fi 与外界的接口，用户通过它来访问 Wi-Fi 的核心功能。WifiWatchdogService 这一系统组件也是通过 WifiManger 来执行一些具体操作。

在应用层开发 Wi-Fi 程序，其实就是使用类 WifiManager 来开发应用程序。在此类中提供了监控 Wi-Fi 状态的方法，其状态主要有如下 5 种。

- WifiManager.WIFI_STATE_DISABLING：表示 Wi-Fi 正在关闭。
- WifiManager.WIFI_STATE_DISABLED：表示 Wi-Fi 已经关闭。
- WifiManager.WIFI_STATE_ENABLING：表示 Wi-Fi 正在打开。
- WifiManager.WIFI_STATE_ENABLED：表示 Wi-Fi 已经打开。
- WifiManager.WIFI_STATE_UNKNOWN：表示 Wi-Fi 无法识别。

第二点是需要在程序中声明一些相关的权限，和 Wi-Fi 权限相关的操作权限主要有如下四个。

- CHANGE_NETWORK_STATE：允许修改网络状态的权限。
- CHANGE_WIFI_STATE：允许修改 Wi-Fi 状态的权限。
- ACCESS_NETWORK_STATE：允许访问网络状态的权限。
- ACCESS_WIFI_STATE：允许访问 Wi-Fi 状态的权限。

10.2.2　WifiService 接口

WifiService 是服务器端的实现，作为 Wi-Fi 的核心，处理实际的驱动加载、扫描、链接、断开等操作，以及底层上报的事件处理。对于主动的命令控制，Wi-Fi 是一个简单的封装，针对来自客户端的控制命令，调用相应的 WifiNative 底层来实现。

当接收到客户端的命令后，一般会将其转换成对应的自身消息塞入消息队列中，以便客户端的调用可以及时返回，然后在 WifiHandler 的 handleMessage()中处理对应的消息。而底层上报的事件，WifiService 则通过启动 WifiStateTracker 来负责处理。WifiStateTracker 和 WifiMonitor 的具体功能如下。

- WifiStateTracker：除了负责 Wi-Fi 的电源管理模式等功能外，其核心部分是 WifiMonitor 所实现的事件轮询机制，以及消息处理函数 handleMessage()。
- WifiMonitor：通过开启一个 MonitorThread 来实现事件的轮询，轮询的关键函数是前面提到的阻塞式函数 WifiNative.waitForEvent()。获取事件后，WifiMonitor 通过一系列的 Handler 通知给 WifiStateTracker。这里 WifiMonitor 的通知机制是将底层事件转换成 WifiStateTracker 所能识别的消息，塞入 WifiStateTracker 的消息循环中，最终在 handleMessage()中由 WifiStateTracker 完成对应的处理。

WifiStateTracker 也是 Wi-Fi 与外界的接口，它不像 WifiManger 那样直接被实例化来操作，而是通过 Intent 机制来发消息通知给在客户端注册的 BroadcastReceiver，以完成和客户端的接口。

10.2.3 WifiWatchdogService 接口

WifiWatchdogService 是 ConnectivityService 所启动的服务，但它并不是通过 Binder 来实现的服务。它的作用是监控同一个网络内的接入点（Access Point），如果当前接入点的 DNS 无法探测（通过 ping 命令实现）通，就自动切换到下一个接入点。WifiWatchdogService 通过 WifiManger 和 WifiStateTracker 辅助完成具体的控制动作。在 WifiWatchdogService 初始化时，通过 registerForWifiBroadcasts 注册获取网络变化的 BroadcastReceiver，也就是捕获 WifiStateTracker 所发出的通知消息，并开启一个 WifiWatchdogThread 线程来处理获取的消息。通过更改 Setting.Secure.WIFI_WARCHDOG_ON 的配置，可以开启和关闭 WifiWatchdog Service。

10.2.4 实战演练——在 Android 系统中控制 Wi-Fi

了解了 Wi-Fi 的基本知识后，接下来将通过一个具体实例的实现过程，详细讲解在 Android 系统中开发 Wi-Fi 应用程序的基本流程。在本实例中新建了一个 Android 应用程序，在 main.xml 中添加 8 个按钮，单击这 8 个按钮可以分别控制手机中的 Wi-Fi，本实例几乎涵盖了现实应用中所有的 Wi-Fi 控制功能。

题　目	目　的	源 码 路 径
实例 10-1	控制 Wi-Fi	daima\10\Wifiguan

本实例的具体实现流程如下。

（1）编写界面布局文件 main.xml，具体代码如下。

```
<?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="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<Button
     android:id="@+id/startButton"
     android:layout_width="300dp"
     android:layout_height="wrap_content"
     android:text="打开 Wi-Fi 网卡"
    />
```

```
<Button
        android:id="@+id/stopButton"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:text="关闭 Wi-Fi 网卡"
       />
<Button
        android:id="@+id/checkButton"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:text="检查 Wi-Fi 网卡状态"
       />
</LinearLayout>
```

（2）编写文件 Main.java，获取屏幕中的按钮单击焦点，根据单击的按钮实现对应的功能。文件 Main.java 的主要实现代码如下。

```
public class Main extends Activity implements OnClickListener {
        // 右侧滚动条按钮
        private ScrollView sView;
        private Button openNetCard;
        private Button closeNetCard;
        private Button checkNetCardState;
        private Button scan;
        private Button getScanResult;
        private Button connect;
        private Button disconnect;
        private Button checkNetWorkState;
        private TextView scanResult;
        private String mScanResult;
        private WifiAdmin mWifiAdmin;
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                mWifiAdmin = new WifiAdmin(Main.this);
                init();
        }
        /**
         * 按钮等控件的初始化
         */
        public void init() {
                sView = (ScrollView) findViewById(R.id.mScrollView);
                openNetCard = (Button) findViewById(R.id.openNetCard);
                closeNetCard = (Button) findViewById(R.id.closeNetCard);
                checkNetCardState = (Button) findViewById(R.id.checkNetCardState);
```

```
        scan = (Button) findViewById(R.id.scan);
        getScanResult = (Button) findViewById(R.id.getScanResult);
        scanResult = (TextView) findViewById(R.id.scanResult);
        connect = (Button) findViewById(R.id.connect);
        disconnect = (Button) findViewById(R.id.disconnect);
        checkNetWorkState = (Button) findViewById(R.id.checkNetWorkState);

        openNetCard.setOnClickListener(Main.this);
        closeNetCard.setOnClickListener(Main.this);
        checkNetCardState.setOnClickListener(Main.this);
        scan.setOnClickListener(Main.this);
        getScanResult.setOnClickListener(Main.this);
        connect.setOnClickListener(Main.this);
        disconnect.setOnClickListener(Main.this);
        checkNetWorkState.setOnClickListener(Main.this);
    }
    /**
     * WIFI_STATE_DISABLING 0 WIFI_STATE_DISABLED 1 WIFI_STATE_ENABLING 2
     * WIFI_STATE_ENABLED 3
     */
    public void openNetCard() {
        mWifiAdmin.openNetCard();
    }
    public void closeNetCard() {
        mWifiAdmin.closeNetCard();
    }
    public void checkNetCardState() {
        mWifiAdmin.checkNetCardState();
    }
    public void scan() {
        mWifiAdmin.scan();
    }
    public void getScanResult() {
        mScanResult = mWifiAdmin.getScanResult();
        scanResult.setText(mScanResult);
    }
    public void connect() {
        mWifiAdmin.connect();
        startActivity(new Intent(android.provider.Settings.ACTION_WIFI_SETTINGS));
    }
    public void disconnect() {
        mWifiAdmin.disconnectWifi();
    }
    public void checkNetWorkState() {
        mWifiAdmin.checkNetWorkState();
    }
```

```
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.openNetCard:
            openNetCard();
            break;
        case R.id.closeNetCard:
            closeNetCard();
            break;
        case R.id.checkNetCardState:
            checkNetCardState();
            break;
        case R.id.scan:
            scan();
            break;
        case R.id.getScanResult:
            getScanResult();
            break;
        case R.id.connect:
            connect();
            break;
        case R.id.disconnect:
            disconnect();
            break;
        case R.id.checkNetWorkState:
            checkNetWorkState();
            break;
        default:
            break;
        }
    }
}
```

（3）将对 Wi-Fi 的相关操作都封装在类 WifiAdmin 中，以后开启或关闭等相关操作可以直接调用这个类的相关方法。类 WifiAdmin 在文件 WifiAdmin.java 中定义，主要实现代码如下。

```
public class WifiAdmin {
    private final static String TAG = "WifiAdmin";
    private StringBuffer mStringBuffer = new StringBuffer();
    private List<ScanResult> listResult;
    private ScanResult mScanResult;
    // 定义 WifiManager 对象
    private WifiManager mWifiManager;
    // 定义 WifiInfo 对象
    private WifiInfo mWifiInfo;
```

```
// 网络连接列表
private List<WifiConfiguration> mWifiConfiguration;
// 定义一个 WifiLock
WifiLock mWifiLock;
/**
 * 构造方法
 */
public WifiAdmin(Context context) {
        mWifiManager = (WifiManager) context
                        .getSystemService(Context.WIFI_SERVICE);
        mWifiInfo = mWifiManager.getConnectionInfo();
}
/**
 * 打开 Wifi 网卡
 */
public void openNetCard() {
        if (!mWifiManager.isWifiEnabled()) {
                mWifiManager.setWifiEnabled(true);
        }
}
/**
 * 关闭 Wifi 网卡
 */
public void closeNetCard() {
        if (mWifiManager.isWifiEnabled()) {
                mWifiManager.setWifiEnabled(false);
        }
}
/**
 * 检查当前 Wifi 网卡状态
 */
public void checkNetCardState() {
        if (mWifiManager.getWifiState()== 0) {
                Log.i(TAG, "网卡正在关闭");
        } else if (mWifiManager.getWifiState()== 1) {
                Log.i(TAG, "网卡已经关闭");
        } else if (mWifiManager.getWifiState()== 2) {
                Log.i(TAG, "网卡正在打开");
        } else if (mWifiManager.getWifiState()== 3) {
                Log.i(TAG, "网卡已经打开");
        } else {
                Log.i(TAG, "---_---晕.....没有获取到状态---_---");
        }
}
/**
 * 扫描周边网络
```

```
 */
public void scan() {
        mWifiManager.startScan();
        listResult = mWifiManager.getScanResults();
        if (listResult != null) {
                Log.i(TAG, "当前区域存在无线网络，请查看扫描结果");
        } else {
                Log.i(TAG, "当前区域没有无线网络");
        }
}
/**
 * 得到扫描结果
 */
public String getScanResult() {
        // 每次点击扫描之前清空上一次的扫描结果
        if (mStringBuffer != null) {
                mStringBuffer = new StringBuffer();
        }
        // 开始扫描网络
        scan();
        listResult = mWifiManager.getScanResults();
        if (listResult != null) {
                for (int i = 0; i < listResult.size(); i++) {
                        mScanResult = listResult.get(i);
                        mStringBuffer = mStringBuffer.append("NO.").append(i + 1)
                                        .append(" :").append(mScanResult.SSID).append("->")
                                        .append(mScanResult.BSSID).append("->")
                                        .append(mScanResult.capabilities).append("->")
                                        .append(mScanResult.frequency).append("->")
                                        .append(mScanResult.level).append("->")
                                        .append(mScanResult.describeContents()).append("\n\n");
                }
        }
        Log.i(TAG, mStringBuffer.toString());
        return mStringBuffer.toString();
}
/**
 * 连接指定网络
 */
public void connect() {
        mWifiInfo = mWifiManager.getConnectionInfo();
}
/**
 * 断开当前连接的网络
 */
public void disconnectWifi() {
```

```
        int netId = getNetworkId();
        mWifiManager.disableNetwork(netId);
        mWifiManager.disconnect();
        mWifiInfo = null;
    }
    /**
     * 检查当前网络状态
     *
     * @return String
     */
    public void checkNetWorkState() {
        if (mWifiInfo != null) {
            Log.i(TAG, "网络正常工作");
        } else {
            Log.i(TAG, "网络已断开");
        }
    }
    /**
     * 得到连接的 ID
     */
    public int getNetworkId() {
        return (mWifiInfo == null) ? 0 : mWifiInfo.getNetworkId();
    }
    /**
     * 得到 IP 地址
     */
    public int getIPAddress() {
        return (mWifiInfo == null) ? 0 : mWifiInfo.getIpAddress();
    }
    // 锁定 WifiLock
    public void acquireWifiLock() {
        mWifiLock.acquire();
    }
    // 解锁 WifiLock
    public void releaseWifiLock() {
        // 判断时候锁定
        if (mWifiLock.isHeld()) {
            mWifiLock.acquire();
        }
    }
    // 创建一个 WifiLock
    public void creatWifiLock() {
        mWifiLock = mWifiManager.createWifiLock("Test");
    }
    // 得到配置好的网络
```

```
    public List<WifiConfiguration> getConfiguration() {
        return mWifiConfiguration;
    }
    // 指定配置好的网络进行连接
    public void connectConfiguration(int index) {
        // 索引大于配置好的网络索引返回
        if (index >= mWifiConfiguration.size()) {
            return;
        }
        // 连接配置好的指定 ID 的网络
        mWifiManager.enableNetwork(mWifiConfiguration.get(index).networkId,
                true);
    }
    // 得到 MAC 地址
    public String getMacAddress() {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.getMacAddress();
    }
    // 得到接入点的 BSSID
    public String getBSSID() {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.getBSSID();
    }
    // 得到 WifiInfo 的所有信息包
    public String getWifiInfo() {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.toString();
    }
    // 添加一个网络并连接
    public int addNetwork(WifiConfiguration wcg) {
        int wcgID = mWifiManager.addNetwork(mWifiConfiguration.get(3));
        mWifiManager.enableNetwork(wcgID, true);
        return wcgID;
    }
}
```

（4）在文件 AndroidManifest.xml 中声明 Wi-Fi 权限，具体代码如下。

```
    <!-- 以下是使用 wifi 访问网络所需的权限 -->
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"></uses-permission>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
```

到此为止，整个实例介绍完毕，执行之后的效果如图 10-4 所示。

本实例和本章上一个实例相比，最大的升级是增加了“扫描结果”功能。单击“扫描结果”按钮后，会显示扫描到的 Wi-Fi 源，如图 10-5 所示。

图 10-4　执行效果

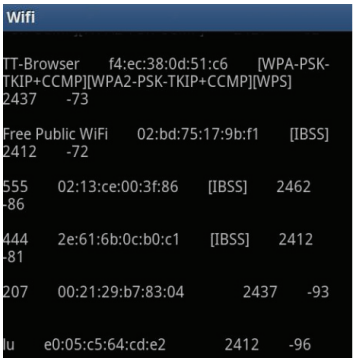


图 10-5　显示扫描结果

注意：在本实例中，连接 Wi-Fi 是比较复杂的，这需要在程序中进行连接，此时会有如下两种可能情况。

（1）Wi-Fi 没有密码，可以直接连接。

（2）Wi-Fi 有密码，在程序中给出密码，然后连接。

最简单的解决方案是直接让程序跳到系统设置的 Wi-Fi 的页面，然后手动去设置。其实 Android 版的 QQ 就是这么做的。

第 11 章　NFC 近场通信技术详解

近场通信（Near Field Communication，NFC）技术由非接触式射频识别（RFID）演变而来，由飞利浦半导体（现为恩智浦半导体）、诺基亚和索尼共同研制开发，其基础是 RFID 及互连技术。NFC 是一种短距高频的无线电技术，在 13.56MHz 频率，运行于 20cm 距离内。其传输速度有 106 Kbit/s、212 Kbit/s 或者 424 Kbit/s 三种。目前近场通信已通过且成为 ISO/IEC IS 18092 国际标准、ECMA-340 标准与 ETSI TS 102 190 标准。NFC 采用主动和被动两种读取模式。在本章的内容中，将详细讲解在 Android 设备中使用 NFC 技术的基本知识。

11.1　近场通信技术基础

NFC 近场通信技术是由 RFID 及互联互通技术整合演变而来，在单一芯片上结合感应式读卡器、感应式卡片和点对点的功能，能在短距离内与兼容设备进行识别和数据交换。工作频率为 13.56MHz，但是使用这种技术进行手机支付的用户必须更换特制的手机。目前这项技术在日韩被广泛应用。手机用户凭借配置了支付功能的手机就可以行遍全国：他们的手机可以用作机场登机验证、大厦的门禁钥匙、交通一卡通、信用卡、支付卡等。在本节的内容中，将简要讲解 NFC 技术的基本知识。

11.1.1　NFC 技术的特点

NFC 是基于 RFID 技术发展起来的一种近距离无线通信技术。与 RFID 一样，近场通信信息也是通过频谱中无线频率部分的电磁感应耦合方式进行传递，但两者之间还是存在很大的区别。近场通信的传输范围比 RFID 小，RFID 的传输范围为 0～1m，但由于近场通信采取了独特的信号衰减技术，相对于 RFID 来说近场通信具有成本低、带宽高、能耗低等特点。

在现实应用中，近场通信技术的主要特征如下。

- 用于近距离（10cm 以内）安全通信的无线通信技术。
- 射频频率：13.56MHz。
- 射频兼容：ISO 14443、ISO 15693、Felica 标准。
- 数据传输速度：106Kbit/s、212 Kbit/s、424Kbit/s。

11.1.2　NFC 的工作模式

在现实应用中，NFC 技术有如下的三种工作模式。

- 卡模式（Card emulation）：此模式其实就是相当于一张采用 RFID 技术的 IC 卡，可以替代当前市面中的大多数 IC 卡，例如场合商场刷卡、公交卡、门禁管制、车票和门票等。卡模式有一个极大的优点，那就是卡片通过非接触读卡器的 RF 域来供电，即便是寄主设备（如手机）没电也可以正常工作。
- 点对点模式（P2P mode）：此模式和红外线差不多，可用于数据交换，只是传输距离较短，但是传输创建速度较快，传输速度也快，且功耗低。将两个具备 NFC 功能的设备连接，能实现数据的点对点传输，如下载音乐、交换图片或者同步设备地址表。因此通过 NFC，多个设备如数码相机、PDA、计算机和手机之间都可以交换资料或者服务。
- 读卡器模式（Reader/writer mode）：作为非接触读卡器使用，比如从海报或者展览信息电子标签上读取相关信息。

11.1.3 NFC 和蓝牙的对比

在现实应用中，NFC 和蓝牙（Bluetooth）都是短程通信技术，而且都被集成到手机中。但 NFC 不需要复杂的设置程序，简单性的甚至可以超越于蓝牙连接方式。NFC 优于蓝牙的地方在于设置程序较短，但无法达到低功率蓝牙（Bluetooth Low Energy）的传输速度。在两台 NFC 设备的相互连接并识别的过程中，连接速度会快于依靠人工设置的蓝牙连接方式，会至少块十分之一秒。

NFC 的最大数据传输量是 424Kbit/s，这远小于 Bluetooth V2.1 (2.1 Mbit/s)。虽然 NFC 在传输速度与距离比不上蓝牙，但是 NFC 技术不需要电源，对于手机或其他电子产品来说，NFC 的使用比较方便。NFC 的短距离通信特性正是其优点，由于耗电量低、一次只和一台机器链接，所以拥有较高的保密性与安全性，NFC 技术用于信用卡交易时可有效避免被盗用。NFC 的目标并非是取代蓝牙等其他无线技术，而是在不同的场合、不同的领域能起到相互补充的作用。

NFC 技术和蓝牙技术相比，主要支持的功能参数如表 11-1 所示。

表 11-1　NFC 技术和蓝牙技术的参数对比

说　　明	NFC	Bluetooth	Bluetooth Low Energy
RFID 兼容	ISO 18000-3	active	active
标准化机构	ISO/IEC	Bluetooth SIG	Bluetooth SIG
网络标准	ISO 13157 etc.	IEEE 802.15.1	IEEE 802.15.1
网络类型	Point-to-point	WPAN	WPAN
加密	not with RFID	available	available
范围	< 0.2 m	~10 m (class 2)	~1 m (class 3)
频率	13.56 MHz	2.4-2.5 GHz	2.4-2.5 GHz
Bit rate	424 kbit/s	2.1 Mbit/s	~1.0 Mbit/s
设置程序	< 0.1 s	< 6 s	< 1 s
功耗	< 15mA (read)	varies with class	< 15 mA (xmit)

11.2　射频识别技术详解

射频识别（Radio Frequency Identification，RFID）技术，又称无线射频识别，是 NFC 技术的一个子集。RFID 是一种通信技术，可以通过无线电信号识别特定目标并读写相关数据，而无需识别系统与特定目标之间建立的机械或光学接触。在现实中常用的 RFID 技术有低频（125KHz~134.2 KHz）、高频（13.56MHz）、超高频，微波等技术。RFID 读写器也分为移动式的和固定式的，目前 RFID 技术应用很广，如图书馆、门禁系统和食品安全溯源等。在本节的内容中，将详细讲解射频识别技术 RFID 的基本知识。

11.2.1　RFID 技术简介

RFID 是一种无线通信技术，可以通过无线电信号识别特定目标并读写相关数据。从概念上来讲，RFID 类似于条码扫描，对于条码技术而言，它是将已编码的条形码附着于目标物，并使用专用的扫描读写器利用光信号将信息由条形磁传送到扫描读写器；而 RFID 则使用专用的 RFID 读写器及专门的可附着于目标物的 RFID 标签，利用频率信号将信息由 RFID 标签传送至 RFID 读写器。

无线电的信号是通过转换成无线电频率的电磁场，把附着在物品标签上的数据传送出去，以自动辨识与追踪该物品。某些标签在识别时从识别器发出的电磁场中就可以得到能量，从而不需要电池；有的标签本身拥有电源，并可以主动发出无线电波（转换成无线电频率的电磁场）。标签包含了电子存储的信息，数米之内都可以识别。与条形码不同的是，射频标签不需要处在识别器视线之内，也可以嵌入在物体之内。

在现实的应用中，有许多行业都运用了射频识别技术。将标签附着在一辆正在生产中的汽车，可以追踪此车在生产线上的进度。在仓库中可以追踪药品的所在。射频标签也可以附于牲畜与宠物上，方便对牲畜与宠物的积极识别（积极识别意思是防止数只牲畜使用同一个身份）。射频识别的身份识别卡可以使员工得以进入锁住的建筑场所，汽车上的射频应答器也可以用于公路和停车场的收费。

某些射频标签可以附在衣物、个人财物上，甚至植入人体之内。由于可能会在未经本人许可的情况下读取个人信息，这项技术会有侵犯个人隐私忧患。

11.2.2　RFID 技术的组成

从结构上讲 RFID 是一种简单的无线系统，只有两个基本组件，该系统用于控制、检测和跟踪物体。系统由一个询问器和多个应答器组成。在最初的技术领域中，应答器是指能够传输信息、回复信息的电子模块。近年来，由于射频技术发展迅猛，应答器有了新的说法和含义，又被叫做智能标签或标签。RFID 电子标签的阅读器通过天线与 RFID 电子标签进行无线通信，可以实现对标签识别码和内存数据的读或写操作。RFID 技术可识别高速运动的物体并可同时识别多个标签，操作快捷方便。

RFID 的具体组成部分如下。

- 应答器：由天线，耦合元件及芯片组成，一般来说都是用标签作为应答器，每个标签具有唯一的电子编码，附着在物体上用于标识目标对象。
- 阅读器：由天线，耦合元件，芯片组成，读取（有时还可以写入）标签信息的设备，可设计为手持式 RFID 读写器（如 C5000W）或固定式读写器。
- 应用软件系统：是应用层软件，主要是把收集的数据做进一步处理。

11.2.3 RFID 技术的特点

射频识别系统最重要的优点是非接触识别，它能穿透雪、雾、冰、涂料、尘垢和条形码无法被使用的恶劣环境阅读标签，并且阅读速度极快，大多数情况下不到 100 毫秒。有源式射频识别系统的速写能力也是重要的优点。可用于流程跟踪和维修跟踪等交互式业务。

制约射频识别系统发展的主要问题是不兼容的标准。射频识别系统的主要厂商提供的都是专有的系统，导致不同的应用和不同的行业采用不同厂商的频率和协议标准，这种混乱和割据的状况已经制约了整个射频识别行业的增长。许多欧美组织正在着手解决这个问题，并已经取得了一些成绩。标准化必将刺激射频识别技术的大幅度发展和广泛应用。

RFID 技术的主要特点如下。

- 快速扫描：RFID 辨识器可以同时辨识读取数个 RFID 标签。
- 体积小型化、形状多样化：RFID 在读取时并不受尺寸大小与形状限制，不需要为了读取的精确度而配合适当的纸张固定尺寸和印刷品质。此外，RFID 标签更可向小型化与多样形态发展，以应用于不同产品。
- 抗污染能力和耐久性：传统条形码的载体是纸张，因此容易受到污染，但 RFID 对水、油和化学药品等物质具有很强的抵抗性。此外，由于条形码是附于塑料袋或外包装纸箱上，所以特别容易受到折损；RFID 卷标是将数据存在芯片中，因此可以免受污损。
- 可重复使用：条形码被印刷到附着物上之后就无法更改，RFID 标签则可以重复地新增、修改、删除 RFID 卷标内储存的数据，方便信息的更新。
- 穿透性和无屏障阅读：在被覆盖的情况下，RFID 能够穿透纸张、木材和塑料等非金属或非透明的材质，并能够进行穿透性通信。而条形码扫描机必须在近距离而且没有物体阻挡的情况下，才可以辨读条形码。
- 数据的记忆容量大：一维条形码的容量是 50B，二维条形码最大的容量可储存 2B～300B，RFID 最大的容量则有几 M。随着记忆载体的发展，数据容量也有不断扩大的趋势。未来物品携带的数据量会越来越大，对卷标的容量需求也相应增加。
- 安全性：由于 RFID 承载的是电子式信息，其数据内容可经由密码保护，使其内容不易被伪造及变更。

RFID 因其所具备的远距离读取、高储存量等特性而备受瞩目。它不仅可以帮助一个企业大幅提高货物信息管理的效率，还可以帮助销售企业和制造企业互联，通过接收反馈信息，需求信息，以优化整个供应链。

11.2.4　RFID 技术的工作原理

RFID 技术的基本工作原理是：当标签进入磁场后，接收解读器发出的射频信号，凭借感应电流所获得的能量发送出存储在芯片中的产品信息（Passive Tag，无源标签或被动标签），或者由标签主动发送某一频率的信号（Active Tag，有源标签或主动标签），解读器读取信息并解码后，送至中央信息系统进行有关的数据处理。

一套完整的 RFID 系统， 是由阅读器与电子标签（TAG）也就是所谓的应答器及应用软件系统三个部份所组成，其工作原理是阅读器发射一特定频率的无线电波能量给应答器，用以驱动应答器电路将内部的数据送出，此时阅读器便依序接收解读数据，送给应用程序做相应的处理。

以 RFID 卡片阅读器及电子标签之间的通信及能量感应方式来区分，可以大致上将 RFID 分成感应耦合（Inductive Coupling）及后向散射耦合（BackscatterCoupling）两种。通常低频的 RFID 大都采用第一种方式，而较高频大多采用第二种方式。

阅读器根据使用的结构和技术不同可以分为读或读/写装置，是 RFID 系统信息控制和处理中心。阅读器通常由耦合模块、收发模块、控制模块和接口单元组成。阅读器和应答器之间一般采用半双工通信方式进行信息交换，同时阅读器通过耦合给无源应答器提供能量和时序。在实际应用中，可进一步通过 Ethernet 或 WLAN 等实现对物体识别信息的采集、处理及远程传送等管理功能。应答器是 RFID 系统的信息载体，应答器大多是由耦合原件（线圈、微带天线等）和微芯片组成的无源单元。

11.3　Android 系统中的 NFC

NFC 通信技术总是由一个发起者和一个接收者组成。通常发起者主动发送电磁场为被动式接受者提供电源。其工作的基本原理和收音机类似。正是由于被动式接受者可以通过发起者提供电源，因此接收可以是非常简单的形式，比如标签、卡和 Sticker 的形式。另外，NFC 也支持点到点的通信（peer to peer），此时参与通信的双方都有电源支持。在 Android 系统的 NFC 模块应用中，Android 手机通常是作为通信中的发起者，也就是作为 NFC 的读写器。Android 手机也可以模拟作为 NFC 通信的接收者，并且从 Android 2.3.3 起也支持点对点通信。Android 系统支持如下的 NFC 标准。

- NfcANFC-A (ISO 14443-3A)。
- NfcBNFC-B (ISO 14443-3B)。
- NfcFNFC-F (JIS 6319-4)。
- NfcVNFC-V (ISO 15693)。
- IsoDepISO-DEP (ISO 14443-4)。
- MifareClassic。
- MifareUltralight。

在 Android 系统中，NFC 模块从上到下的结构如图 11-1 所示。

在本节的内容中，将详细讲解 Android 系统中 NFC 模块源码的基本知识。

```
---------- /system/framework/framework.jar----------

android.nfc                    标准接口 （NFCAdapter/NfcManager）
android.nfc.tech               标签技术

---------- /system/Nfc.apk----------

com.android.nfc                NFC 服务相关
        .DeviceHost                底层设备接口原型
        .NfcService                NFC 服务 实现 DeviceHostListener 接口
com.android.nfc.dhimpl     NFC 功能底层实现-com.android.nfc.DeviceHost (NXP)
        .NativeNfcManager          implements DeviceHost
        JNI-> com_android_nfc_NativeNfcManager.cpp (libnfc_jni.so)
        .NativeNfcSecureElement
        JNI-> com_android_nfc_NativeNfcSecureElement.cpp (libnfc_jni.so)

---------- /system/lib/libnfc___.so----------

libnfc-nxp => libnfc.so, libnfc_ndef.so
libnfc-nci => libnfc-nci.so
```

图 11-1　NFC 模块从上到下的结构

11.3.1　分析 Java 层

在 Android 系统中，NFC 模块的 Java 层代码位于如下的目录中。

```
\frameworks\base\core\java\android\nfc\
```

在上述目录中，包含了用来与本地 NFC 适配器进行交互的顶层类，这些类可以表示被检测到的 TAG 和用 NDEF（NFC 组织约定的 NFC tag 中的数据格式）。在 NFC 的 Java 层中，常用的顶层类如下。

（1）NfcManager

类 NfcManager 在文件\frameworks\base\core\java\android\nfc\NfcManager.java 中定义，这是一个 NFC Adapter 的管理器，可以列出所有此 Android 设备中被支持的 NFC Adapter，只不过大部分 Android 设备只有一个 NFC Adapter，所以在大部分情况下可以直接用静态方法 getDefaultAdapter(context)来取适配器。文件\frameworks\base\core\java\android\nfc\ NfcManager.java 的具体实现代码如下。

```
public final class NfcManager {
    private final NfcAdapter mAdapter;
    public NfcManager(Context context) {
        NfcAdapter adapter;
        context = context.getApplicationContext();
        if (context == null) {
            throw new IllegalArgumentException(
                    "context not associated with any application (using a mock context?)");
```

```
            }
            try {
                adapter = NfcAdapter.getNfcAdapter(context);
            } catch (UnsupportedOperationException e) {
                adapter = null;
            }
            mAdapter = adapter;
        }
        /**
         * 得到这个设备的默认 NFC 适配器
         *
         */
        public NfcAdapter getDefaultAdapter() {
            return mAdapter;
        }
    }
```

（2）NfcAdapter

类在文件\frameworks\base\core\java\android\nfc\NfcAdapter.java 中定义，此类表示本设备的 NFC Adapter，可以定义 Intent 来请求将系统检测到 TAG 的提醒发送到 Activity，并提供方法去注册前台 TAG 提醒发布和前台 NDEF 推送。 前台 NDEF 推送是当前 Android 版本唯一支持的点对点 NFC 通信方式。文件\frameworks\base\core\java\android\nfc\NfcAdapter.java 的具体实现代码如下。

```
    public final class NfcAdapter {
        static final String TAG = "NFC";
        public static final String ACTION_NDEF_DISCOVERED = "android.nfc.action.NDEF_
        DISCOVERED";
        @SdkConstant(SdkConstantType.ACTIVITY_INTENT_ACTION)
        public static final String ACTION_TECH_DISCOVERED = "android.nfc.action.TECH_
        DISCOVERED";
        @SdkConstant(SdkConstantType.ACTIVITY_INTENT_ACTION)
        public static final String ACTION_TAG_DISCOVERED = "android.nfc.action.TAG_
        DISCOVERED";
        public static final String ACTION_TAG_LEFT_FIELD = "android.nfc.action.TAG_LOST";
        public static final String EXTRA_TAG = "android.nfc.extra.TAG";
        public static final String EXTRA_NDEF_MESSAGES = "android.nfc.extra.NDEF_MESSAGES";
        public static final String EXTRA_ID = "android.nfc.extra.ID";
        @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
        public static final String ACTION_ADAPTER_STATE_CHANGED =
                "android.nfc.action.ADAPTER_STATE_CHANGED";
        public static final String EXTRA_ADAPTER_STATE = "android.nfc.extra.ADAPTER_STATE";
        public static final int STATE_OFF = 1;
        public static final int STATE_TURNING_ON = 2;
        public static final int STATE_ON = 3;
        public static final int STATE_TURNING_OFF = 4;
```

```
public static final int FLAG_NDEF_PUSH_NO_CONFIRM = 0x1;
public static final String ACTION_HANDOVER_TRANSFER_STARTED =
        "android.nfc.action.HANDOVER_TRANSFER_STARTED";
public static final String ACTION_HANDOVER_TRANSFER_DONE =
        "android.nfc.action.HANDOVER_TRANSFER_DONE";
public static final String EXTRA_HANDOVER_TRANSFER_STATUS =
        "android.nfc.extra.HANDOVER_TRANSFER_STATUS";
public static final int HANDOVER_TRANSFER_STATUS_SUCCESS = 0;
public static final int HANDOVER_TRANSFER_STATUS_FAILURE = 1;
public static final String EXTRA_HANDOVER_TRANSFER_URI =
        "android.nfc.extra.HANDOVER_TRANSFER_URI";
static boolean sIsInitialized = false;
static INfcAdapter sService;
static INfcTag sTagService;
static HashMap<Context, NfcAdapter> sNfcAdapters = new HashMap();
static NfcAdapter sNullContextNfcAdapter;
final NfcActivityManager mNfcActivityManager;
final Context mContext;
public interface OnNdefPushCompleteCallback {
    public void onNdefPushComplete(NfcEvent event);
}
public interface CreateNdefMessageCallback {
    public NdefMessage createNdefMessage(NfcEvent event);
}
public interface CreateBeamUrisCallback {
    public Uri[] createBeamUris(NfcEvent event);
}
private static boolean hasNfcFeature() {
    IPackageManager pm = ActivityThread.getPackageManager();
    if (pm == null) {
        Log.e(TAG, "Cannot get package manager, assuming no NFC feature");
        return false;
    }
    try {
        return pm.hasSystemFeature(PackageManager.FEATURE_NFC);
    } catch (RemoteException e) {
        Log.e(TAG, "Package manager query failed, assuming no NFC feature", e);
        return false;
    }
}
public static synchronized NfcAdapter getNfcAdapter(Context context) {
    if (!sIsInitialized) {
        /* 这意味着拥有 NFC 装置*/
        if (!hasNfcFeature()) {
            Log.v(TAG, "this device does not have NFC support");
```

```
                throw new UnsupportedOperationException();
            }
            sService = getServiceInterface();
            if (sService == null) {
                Log.e(TAG, "could not retrieve NFC service");
                throw new UnsupportedOperationException();
            }
            try {
                sTagService = sService.getNfcTagInterface();
            } catch (RemoteException e) {
                Log.e(TAG, "could not retrieve NFC Tag service");
                throw new UnsupportedOperationException();
            }
            sIsInitialized = true;
        }
        if (context == null) {
            if (sNullContextNfcAdapter == null) {
                sNullContextNfcAdapter = new NfcAdapter(null);
            }
            return sNullContextNfcAdapter;
        }
        NfcAdapter adapter = sNfcAdapters.get(context);
        if (adapter == null) {
            adapter = new NfcAdapter(context);
            sNfcAdapters.put(context, adapter);
        }
        return adapter;
    }
    /** 获取 NFC 服务接口的 */
    private static INfcAdapter getServiceInterface() {
        IBinder b = ServiceManager.getService("nfc");
        if (b == null) {
            return null;
        }
        return INfcAdapter.Stub.asInterface(b);
    }
    public static NfcAdapter getDefaultAdapter(Context context) {
        if (context == null) {
            throw new IllegalArgumentException("context cannot be null");
        }
        context = context.getApplicationContext();
        if (context == null) {
            throw new IllegalArgumentException(
                    "context not associated with any application (using a mock context?)");
        }
```

```
        /* 一致性验证 */
        NfcManager manager = (NfcManager) context.getSystemService(Context.NFC_SERVICE);
        if (manager == null) {
            // NFC not available
            return null;
        }
        return manager.getDefaultAdapter();
    }
    @Deprecated
    public static NfcAdapter getDefaultAdapter() {
        Log.w(TAG, "WARNING: NfcAdapter.getDefaultAdapter() is deprecated, use " +
                "NfcAdapter.getDefaultAdapter(Context) instead", new Exception());

        return NfcAdapter.getNfcAdapter(null);
    }
    public boolean isEnabled() {
        try {
            return sService.getState()== STATE_ON;
        } catch (RemoteException e) {
            attemptDeadServiceRecovery(e);
            return false;
        }
    }
    public int getAdapterState() {
        try {
            return sService.getState();
        } catch (RemoteException e) {
            attemptDeadServiceRecovery(e);
            return NfcAdapter.STATE_OFF;
        }
    }
    public boolean enable() {
        try {
            return sService.enable();
        } catch (RemoteException e) {
            attemptDeadServiceRecovery(e);
            return false;
        }
    }
    public boolean disable() {
        try {
            return sService.disable(true);
        } catch (RemoteException e) {
            attemptDeadServiceRecovery(e);
            return false;
        }
```

```
}
public void setBeamPushUris(Uri[] uris, Activity activity) {
    if (activity == null) {
        throw new NullPointerException("activity cannot be null");
    }
    if (uris != null) {
        for (Uri uri : uris) {
            if (uri == null) throw new NullPointerException("Uri not " +
                    "allowed to be null");
            String scheme = uri.getScheme();
            if (scheme == null || (!scheme.equalsIgnoreCase("file") &&
                    !scheme.equalsIgnoreCase("content"))) {
                throw new IllegalArgumentException("URI needs to have " +
                        "either scheme file or scheme content");
            }
        }
    }
    mNfcActivityManager.setNdefPushContentUri(activity, uris);
}
public void setBeamPushUrisCallback(CreateBeamUrisCallback callback, Activity activity) {
    if (activity == null) {
        throw new NullPointerException("activity cannot be null");
    }
    mNfcActivityManager.setNdefPushContentUriCallback(activity, callback);
}
public void setNdefPushMessage(NdefMessage message, Activity activity,
        Activity ... activities) {
    int targetSdkVersion = getSdkVersion();
    try {
        if (activity == null) {
            throw new NullPointerException("activity cannot be null");
        }
        mNfcActivityManager.setNdefPushMessage(activity, message, 0);
        for (Activity a : activities) {
            if (a == null) {
                throw new NullPointerException("activities cannot contain null");
            }
            mNfcActivityManager.setNdefPushMessage(a, message, 0);
        }
    } catch (IllegalStateException e) {
        if (targetSdkVersion < android.os.Build.VERSION_CODES.JELLY_BEAN) {
            Log.e(TAG, "Cannot call API with Activity that has already " +
                    "been destroyed", e);
        } else {
            // Prevent new applications from making this mistake, re-throw
            throw(e);
```

```
            }
        }
    }
    public void setNdefPushMessage(NdefMessage message, Activity activity, int flags) {
        if (activity == null) {
            throw new NullPointerException("activity cannot be null");
        }
        mNfcActivityManager.setNdefPushMessage(activity, message, flags);
    }
    public void setNdefPushMessageCallback(CreateNdefMessageCallback callback, Activity activity,
            Activity ... activities) {
        int targetSdkVersion = getSdkVersion();
        try {
            if (activity == null) {
                throw new NullPointerException("activity cannot be null");
            }
            mNfcActivityManager.setNdefPushMessageCallback(activity, callback, 0);
            for (Activity a : activities) {
                if (a == null) {
                    throw new NullPointerException("activities cannot contain null");
                }
                mNfcActivityManager.setNdefPushMessageCallback(a, callback, 0);
            }
        } catch (IllegalStateException e) {
            if (targetSdkVersion < android.os.Build.VERSION_CODES.JELLY_BEAN) {
                Log.e(TAG, "Cannot call API with Activity that has already " +
                        "been destroyed", e);
            } else {
                throw(e);
            }
        }
    }
    public void setNdefPushMessageCallback(CreateNdefMessageCallback callback, Activity activity,
            int flags) {
        if (activity == null) {
            throw new NullPointerException("activity cannot be null");
        }
        mNfcActivityManager.setNdefPushMessageCallback(activity, callback, flags);
    }
    public void setOnNdefPushCompleteCallback(OnNdefPushCompleteCallback callback,
            Activity activity, Activity ... activities) {
        int targetSdkVersion = getSdkVersion();
        try {
            if (activity == null) {
                throw new NullPointerException("activity cannot be null");
            }
```

```
                mNfcActivityManager.setOnNdefPushCompleteCallback(activity, callback);
                for (Activity a : activities) {
                    if (a == null) {
                        throw new NullPointerException("activities cannot contain null");
                    }
                    mNfcActivityManager.setOnNdefPushCompleteCallback(a, callback);
                }
            } catch (IllegalStateException e) {
                if (targetSdkVersion < android.os.Build.VERSION_CODES.JELLY_BEAN) {
                    Log.e(TAG, "Cannot call API with Activity that has already " +
                            "been destroyed", e);
                } else {
                    throw(e);
                }
            }
        }
        public void enableForegroundDispatch(Activity activity, PendingIntent intent,
                IntentFilter[] filters, String[][] techLists) {
            if (activity == null || intent == null) {
                throw new NullPointerException();
            }
            if (!activity.isResumed()) {
                throw new IllegalStateException("Foreground dispatch can only be enabled " +
                        "when your activity is resumed");
            }
            try {
                TechListParcel parcel = null;
                if (techLists != null && techLists.length > 0) {
                    parcel = new TechListParcel(techLists);
                }
                ActivityThread.currentActivityThread().registerOnActivityPausedListener(activity,
                        mForegroundDispatchListener);
                sService.setForegroundDispatch(intent, filters, parcel);
            } catch (RemoteException e) {
                attemptDeadServiceRecovery(e);
            }
        }
        public void disableForegroundDispatch(Activity activity) {
            ActivityThread.currentActivityThread().unregisterOnActivityPausedListener(activity,
                    mForegroundDispatchListener);
            disableForegroundDispatchInternal(activity, false);
        }
```

（3）NdefMessage 和 NdefRecord

NDEF 是 NFC 组织定义的数据结构，用来将有效的数据存储到 NFC TAG 中，比如文本、URL 和其他 MIME 类型。一个 NdefMessage 扮演一个容器，这个容器存储发送和读到的数据。

一个 NdefMessage 对象包含 0 或多个 NdefRecord，每个 NDEF record 有一个类型，比如文本，URL，智慧型海报/广告，或其他 MIME 数据。在 NDEFMessage 中的第一个 NfcRecord 的类型，功能是发送 TAG 到一个 Android 设备上的 Activity。其中类 NdefMessage 在文件 \frameworks\base\core\java\android\nfc\NdefMessage.java 中定义，具体实现代码如下。

```
public final class NdefMessage implements Parcelable {
    private final NdefRecord[] mRecords;
    public NdefMessage(byte[] data) throws FormatException {
        if (data == null) throw new NullPointerException("data is null");
        ByteBuffer buffer = ByteBuffer.wrap(data);
        mRecords = NdefRecord.parse(buffer, false);
        if (buffer.remaining() > 0) {
            throw new FormatException("trailing data");
        }
    }
    public NdefMessage(NdefRecord record, NdefRecord ... records) {
        // 验证
        if (record == null) throw new NullPointerException("record cannot be null");
        for (NdefRecord r : records) {
            if (r == null) {
                throw new NullPointerException("record cannot be null");
            }
        }
        mRecords = new NdefRecord[1 + records.length];
        mRecords[0] = record;
        System.arraycopy(records, 0, mRecords, 1, records.length);
    }
    public NdefMessage(NdefRecord[] records) {
        // 验证
        if (records.length < 1) {
            throw new IllegalArgumentException("must have at least one record");
        }
        for (NdefRecord r : records) {
            if (r == null) {
                throw new NullPointerException("records cannot contain null");
            }
        }
        mRecords = records;
    }
    public NdefRecord[] getRecords() {
        return mRecords;
    }
    public int getByteArrayLength() {
        int length = 0;
        for (NdefRecord r : mRecords) {
            length += r.getByteLength();
```

```
        }
        return length;
    }
    public byte[] toByteArray() {
        int length = getByteArrayLength();
        ByteBuffer buffer = ByteBuffer.allocate(length);

        for (int i=0; i<mRecords.length; i++) {
            boolean mb = (i == 0);    // first record
            boolean me = (i == mRecords.length - 1);    // last record
            mRecords[i].writeToByteBuffer(buffer, mb, me);
        }
        return buffer.array();
    }
    @Override
    public int describeContents() {
        return 0;
    }
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeInt(mRecords.length);
        dest.writeTypedArray(mRecords, flags);
    }
    public static final Parcelable.Creator<NdefMessage> CREATOR =
            new Parcelable.Creator<NdefMessage>() {
        @Override
        public NdefMessage createFromParcel(Parcel in) {
            int recordsLength = in.readInt();
            NdefRecord[] records = new NdefRecord[recordsLength];
            in.readTypedArray(records, NdefRecord.CREATOR);
            return new NdefMessage(records);
        }
        @Override
        public NdefMessage[] newArray(int size) {
            return new NdefMessage[size];
        }
    };
    @Override
    public int hashCode() {
        return Arrays.hashCode(mRecords);
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
```

```
            NdefMessage other = (NdefMessage) obj;
            return Arrays.equals(mRecords, other.mRecords);
        }
        @Override
        public String toString() {
            return "NdefMessage " + Arrays.toString(mRecords);
        }
    }
```

（4）TAG

类 TAG 在文件\frameworks\base\core\java\android\nfc\Tag.java 中定义，此类用于标示一个被动的 NFC 目标，比如 TAG、CARD 和钥匙挂扣，甚至是一个电话模拟的 NFC 卡。当一个 TAG 被检测到时，一个 TAG 对象将被创建并且封装到一个 Intent 里，然后 NFC 发布系统将这个 Intent 用 startActivity()发送到注册了接受这种 Intent 的 Activity 里。可以用方法 getTechList()来得到这个 TAG 支持的技术细节，并创建一个 android.nfc.tech 提供的相应的 TagTechnology 对象。文件 Tag.java 的具体实现代码如下。

```
public final class Tag implements Parcelable {
    final byte[] mId;
    final int[] mTechList;
    final String[] mTechStringList;
    final Bundle[] mTechExtras;
    final int mServiceHandle;
    final INfcTag mTagService;
    int mConnectedTechnology;
    public Tag(byte[] id, int[] techList, Bundle[] techListExtras, int serviceHandle,
            INfcTag tagService) {
        if (techList == null) {
            throw new IllegalArgumentException("rawTargets cannot be null");
        }
        mId = id;
        mTechList = Arrays.copyOf(techList, techList.length);
        mTechStringList = generateTechStringList(techList);
        // 确保 mTechExtras 和 mTechList 一致
        mTechExtras = Arrays.copyOf(techListExtras, techList.length);
        mServiceHandle = serviceHandle;
        mTagService = tagService;
        mConnectedTechnology = -1;
    }
    public static Tag createMockTag(byte[] id, int[] techList, Bundle[] techListExtras) {
        return new Tag(id, techList, techListExtras, 0, null);
    }
```

（5）“tech\”子目录

除此之外，在 frameworks\base\core\java\android\nfc\tech\目录中包含了查询 TAG 属性和进

行 I/O 操作的类。这些类分别标示一个 TAG 支持的不同的 NFC 技术标准，如图 11-2 所示。

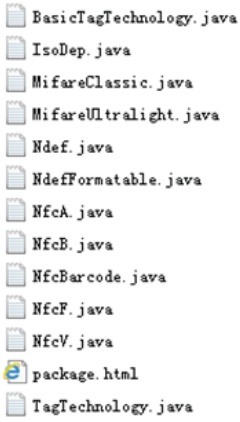


图 11-2　frameworks\base\core\java\android\nfc\tech\目录

在图 11-2 所示的目录中，类 TagTechnology 是如表 11-2 中所示的类必须要实现的。

表 11-2　必须实现 TagTechnology 的类

类	说　明
NfcA	支持 ISO 14443-3A 标准的操作。Provides access to NFC-A (ISO 14443-3A) properties and I/O operations.
NfcB	Provides access to NFC-B (ISO 14443-3B) properties and I/O operations.
NfcF	Provides access to NFC-F (JIS 6319-4) properties and I/O operations.
NfcV	Provides access to NFC-V (ISO 15693) properties and I/O operations.
IsoDep	Provides access to ISO-DEP (ISO 14443-4) properties and I/O operations.
Ndef	提供对那些被格式化为 NDEF 的 tag 的数据的访问和其他操作。

而类 NdefFormatable 对那些可以被格式化成 NDEF 格式的 TAG 提供了一个格式化的操作，文件 frameworks\base\core\java\android\nfc\tech\NdefFormatable.java 的具体实现代码如下。

```
public final class NdefFormatable extends BasicTagTechnology {
    private static final String TAG = "NFC";
    public static NdefFormatable get(Tag tag) {
        if (!tag.hasTech(TagTechnology.NDEF_FORMATABLE)) return null;
        try {
            return new NdefFormatable(tag);
        } catch (RemoteException e) {
            return null;
        }
    }
    void format(NdefMessage firstMessage, boolean makeReadOnly) throws IOException,
            FormatException {
        checkConnected();
```

```
try {
    int serviceHandle = mTag.getServiceHandle();
    INfcTag tagService = mTag.getTagService();
    int errorCode = tagService.formatNdef(serviceHandle, MifareClassic.KEY_DEFAULT);
    switch (errorCode) {
        case ErrorCodes.SUCCESS:
            break;
        case ErrorCodes.ERROR_IO:
            throw new IOException();
        case ErrorCodes.ERROR_INVALID_PARAM:
            throw new FormatException();
        default:
            throw new IOException();
    }
    // 检查并查看是否工作
    if (!tagService.isNdef(serviceHandle)) {
        throw new IOException();
    }
    if (firstMessage != null) {
        errorCode = tagService.ndefWrite(serviceHandle, firstMessage);
        switch (errorCode) {
            case ErrorCodes.SUCCESS:
                break;
            case ErrorCodes.ERROR_IO:
                throw new IOException();
            case ErrorCodes.ERROR_INVALID_PARAM:
                throw new FormatException();
            default:
                throw new IOException();
        }
    }
    // 选择只读
    if (makeReadOnly) {
        errorCode = tagService.ndefMakeReadOnly(serviceHandle);
        switch (errorCode) {
            case ErrorCodes.SUCCESS:
                break;
            case ErrorCodes.ERROR_IO:
                throw new IOException();
            case ErrorCodes.ERROR_INVALID_PARAM:
                throw new IOException();
            default:
                throw new IOException();
        }
    }
} catch (RemoteException e) {
```

```
                Log.e(TAG, "NFC service dead", e);
            }
        }
    }
```

类 MifareClassic 在文件 frameworks\base\core\java\android\nfc\tech\MifareClassic.java 中定义，如果 Android 设备支持 MIFARE，则提供针对 MIFARE Classic 目标的属性和 I/O 操作。文件 MifareClassic.java 的具体实现代码如下。

```
public final class MifareClassic extends BasicTagTechnology {
    private static final String TAG = "NFC";
    public static final byte[] KEY_DEFAULT =
            {(byte)0xFF,(byte)0xFF,(byte)0xFF,(byte)0xFF,(byte)0xFF,(byte)0xFF};
    public static final byte[] KEY_MIFARE_APPLICATION_DIRECTORY =
            {(byte)0xA0,(byte)0xA1,(byte)0xA2,(byte)0xA3,(byte)0xA4,(byte)0xA5};
    public static final byte[] KEY_NFC_FORUM =
            {(byte)0xD3,(byte)0xF7,(byte)0xD3,(byte)0xF7,(byte)0xD3,(byte)0xF7};
    public static final int TYPE_UNKNOWN = -1;
    public static final int TYPE_CLASSIC = 0;
    public static final int TYPE_PLUS = 1;
    public static final int TYPE_PRO = 2;
    public static final int SIZE_1K = 1024;
    public static final int SIZE_2K = 2048;
    public static final int SIZE_4K = 4096;
    public static final int SIZE_MINI = 320;
    public static final int BLOCK_SIZE = 16;
    private static final int MAX_BLOCK_COUNT = 256;
    private static final int MAX_SECTOR_COUNT = 40;
    private boolean mIsEmulated;
    private int mType;
    private int mSize;
    public static MifareClassic get(Tag tag) {
        if (!tag.hasTech(TagTechnology.MIFARE_CLASSIC)) return null;
        try {
            return new MifareClassic(tag);
        } catch (RemoteException e) {
            return null;
        }
    }
    public MifareClassic(Tag tag) throws RemoteException {
        super(tag, TagTechnology.MIFARE_CLASSIC);
        NfcA a = NfcA.get(tag);    // MIFARE Classic is always based on NFC a
        mIsEmulated = false;
        switch (a.getSak()) {
        case 0x01:
        case 0x08:
```

```
                mType = TYPE_CLASSIC;
                mSize = SIZE_1K;
                break;
            case 0x09:
                mType = TYPE_CLASSIC;
                mSize = SIZE_MINI;
                break;
            case 0x10:
                mType = TYPE_PLUS;
                mSize = SIZE_2K;
                // SecLevel = SL2
                break;
            case 0x11:
                mType = TYPE_PLUS;
                mSize = SIZE_4K;
                // Seclevel = SL2
                break;
            case 0x18:
                mType = TYPE_CLASSIC;
                mSize = SIZE_4K;
                break;
            case 0x28:
                mType = TYPE_CLASSIC;
                mSize = SIZE_1K;
                mIsEmulated = true;
                break;
            case 0x38:
                mType = TYPE_CLASSIC;
                mSize = SIZE_4K;
                mIsEmulated = true;
                break;
            case 0x88:
                mType = TYPE_CLASSIC;
                mSize = SIZE_1K;
                // NXP-tag: false
                break;
            case 0x98:
            case 0xB8:
                mType = TYPE_PRO;
                mSize = SIZE_4K;
                break;
            default:
                throw new RuntimeException(
                        "Tag incorrectly enumerated as MIFARE Classic, SAK = " + a.getSak());
            }
        }
```

类MifareUltralight在文件frameworks\base\core\java\android\nfc\tech\MifareUltralight.java中定义，如果 Android 设备支持 MIFARE，则提供针对 MIFARE Ultralight 目标的属性和 I/O 操作。

```
public final class MifareUltralight extends BasicTagTechnology {
    private static final String TAG = "NFC";
    public static final int TYPE_UNKNOWN = -1;
    public static final int TYPE_ULTRALIGHT = 1;
    public static final int TYPE_ULTRALIGHT_C = 2;
    public static final int PAGE_SIZE = 4;
    private static final int NXP_MANUFACTURER_ID = 0x04;
    private static final int MAX_PAGE_COUNT = 256;
    public static final String EXTRA_IS_UL_C = "isulc";
    private int mType;
    public static MifareUltralight get(Tag tag) {
        if (!tag.hasTech(TagTechnology.MIFARE_ULTRALIGHT)) return null;
        try {
            return new MifareUltralight(tag);
        } catch (RemoteException e) {
            return null;
        }
    }
    public MifareUltralight(Tag tag) throws RemoteException {
        super(tag, TagTechnology.MIFARE_ULTRALIGHT);
        NfcA a = NfcA.get(tag);
        mType = TYPE_UNKNOWN;
        if (a.getSak()== 0x00 && tag.getId()[0] == NXP_MANUFACTURER_ID) {
            Bundle extras = tag.getTechExtras(TagTechnology.MIFARE_ULTRALIGHT);
            if (extras.getBoolean(EXTRA_IS_UL_C)) {
                mType = TYPE_ULTRALIGHT_C;
            } else {
                mType = TYPE_ULTRALIGHT;
            }
        }
    }
    public byte[] readPages(int pageOffset) throws IOException {
        validatePageIndex(pageOffset);
        checkConnected();

        byte[] cmd = { 0x30, (byte) pageOffset};
        return transceive(cmd, false);
    }
    public void writePage(int pageOffset, byte[] data) throws IOException {
        validatePageIndex(pageOffset);
        checkConnected();
        byte[] cmd = new byte[data.length + 2];
        cmd[0] = (byte) 0xA2;
```

```
            cmd[1] = (byte) pageOffset;
            System.arraycopy(data, 0, cmd, 2, data.length);
            transceive(cmd, false);
        }
        public void setTimeout(int timeout) {
            try {
                int err = mTag.getTagService().setTimeout(
                        TagTechnology.MIFARE_ULTRALIGHT, timeout);
                if (err != ErrorCodes.SUCCESS) {
                    throw new IllegalArgumentException("The supplied timeout is not valid");
                }
            } catch (RemoteException e) {
                Log.e(TAG, "NFC service dead", e);
            }
        }
        public int getTimeout() {
            try {
                return mTag.getTagService().getTimeout(TagTechnology.MIFARE_ULTRALIGHT);
            } catch (RemoteException e) {
                Log.e(TAG, "NFC service dead", e);
                return 0;
            }
        }
        private static void validatePageIndex(int pageIndex) {
            if (pageIndex < 0 || pageIndex >= MAX_PAGE_COUNT) {
                throw new IndexOutOfBoundsException("page out of bounds: " + pageIndex);
            }
        }
    }
```

11.3.2 分析 JNI 部分

在 Android 系统中，NFC 模块的 JNI 代码在如下的目录中。

```
\packages\apps\Nfc\nxp\jni
```

JNI 代码向上会跟 Framework 层的 Java 代码进行交互，向下会跟 libnfc 层进行交互。JNI 层的核心文件是 com_android_nfc_NativeNfcManager.cpp，功能是初始化并启动 NFC 服务，并扫描和读取 TAG。文件 com_android_nfc_NativeNfcManager.cpp 的主要实现代码如下。

```
static void client_kill_deferred_call(void* arg)
{
    struct nfc_jni_native_data *nat = (struct nfc_jni_native_data *)arg;

    nat->running = FALSE;
```

```
}
static void kill_client(nfc_jni_native_data *nat)
{
    phDal4Nfc_Message_Wrapper_t    wrapper;
    phLibNfc_DeferredCall_t         *pMsg;
    usleep(50000);
    ALOGD("Terminating client thread...");
    pMsg = (phLibNfc_DeferredCall_t*)malloc(sizeof(phLibNfc_DeferredCall_t));
    pMsg->pCallback = client_kill_deferred_call;
    pMsg->pParameter = (void*)nat;
    wrapper.msg.eMsgType = PH_LIBNFC_DEFERREDCALL_MSG;
    wrapper.msg.pMsgData = pMsg;
    wrapper.msg.Size          = sizeof(phLibNfc_DeferredCall_t);
    phDal4Nfc_msgsnd(gDrvCfg.nClientId, (struct msgbuf *)&wrapper, sizeof(phLibNfc_Message_t), 0);
}
static void nfc_jni_ioctl_callback(void *pContext, phNfc_sData_t *pOutput, NFCSTATUS status) {
    struct nfc_jni_callback_data * pCallbackData = (struct nfc_jni_callback_data *) pContext;
    LOG_CALLBACK("nfc_jni_ioctl_callback", status);
    /*报告回调状态并唤醒并用方*/
    pCallbackData->status = status;
    sem_post(&pCallbackData->sem);
}
static void nfc_jni_deinit_download_callback(void *pContext, NFCSTATUS status)
{
    struct nfc_jni_callback_data * pCallbackData = (struct nfc_jni_callback_data *) pContext;
    LOG_CALLBACK("nfc_jni_deinit_download_callback", status);
    /*报告回调状态并唤醒并用方*/
    pCallbackData->status = status;
    sem_post(&pCallbackData->sem);
}

static int nfc_jni_download_locked(struct nfc_jni_native_data *nat, uint8_t update)
{
     uint8_t OutputBuffer[1];
     uint8_t InputBuffer[1];
     struct timespec ts;
     NFCSTATUS status = NFCSTATUS_FAILED;
     phLibNfc_StackCapabilities_t caps;
     struct nfc_jni_callback_data cb_data;
     bool result;
     /*  创建本地信号*/
     if (!nfc_cb_data_init(&cb_data, NULL))
     {
         goto clean_and_return;
     }
     if(update)
```

```
{
    //deinit
    TRACE("phLibNfc_Mgt_DeInitialize() (download)");
    REENTRANCE_LOCK();
    status = phLibNfc_Mgt_DeInitialize(gHWRef, nfc_jni_deinit_download_callback, (void *)&cb_data);
    REENTRANCE_UNLOCK();
    if (status != NFCSTATUS_PENDING)
    {
        ALOGE("phLibNfc_Mgt_DeInitialize() (download) returned 0x%04x[%s]", status,
        nfc_jni_get_ status_name(status));
    }
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += 5;
    /*  等待回调响应*/
    if(sem_timedwait(&cb_data.sem, &ts))
    {
        ALOGW("Deinitialization timed out (download)");
    }
    if(cb_data.status != NFCSTATUS_SUCCESS)
    {
        ALOGW("Deinitialization FAILED (download)");
    }
    TRACE("Deinitialization SUCCESS (download)");
}
result = performDownload(nat, false);

if (!result) {
    status = NFCSTATUS_FAILED;
    goto clean_and_return;
}
TRACE("phLibNfc_Mgt_Initialize()");
REENTRANCE_LOCK();
status = phLibNfc_Mgt_Initialize(gHWRef, nfc_jni_init_callback, (void *)&cb_data);
REENTRANCE_UNLOCK();
if(status != NFCSTATUS_PENDING)
{
    ALOGE("phLibNfc_Mgt_Initialize() (download) returned 0x%04x[%s]", status, nfc_jni_get_
    status_name(status));
    goto clean_and_return;
}
TRACE("phLibNfc_Mgt_Initialize() returned 0x%04x[%s]", status, nfc_jni_get_ status_name
(status));
if(sem_wait(&cb_data.sem))
{
   ALOGE("Failed to wait for semaphore (errno=0x%08x)", errno);
   status = NFCSTATUS_FAILED;
```

```
            goto clean_and_return;
        }
        /*初始化状态*/
        if(cb_data.status != NFCSTATUS_SUCCESS)
        {
            status = cb_data.status;
            goto clean_and_return;
        }
        /* ====== CAPABILITIES ======= */
        REENTRANCE_LOCK();
        status = phLibNfc_Mgt_GetstackCapabilities(&caps, (void*)nat);
        REENTRANCE_UNLOCK();
        if (status != NFCSTATUS_SUCCESS)
        {
            ALOGW("phLibNfc_Mgt_GetstackCapabilities returned 0x%04x[%s]", status, nfc_jni_get_
            status_name(status));
        }
        else
        {
            ALOGD("NFC capabilities: HAL = %x, FW = %x, HW = %x, Model = %x, HCI = %x,
            Full_FW = %d, Rev = %d, FW Update Info = %d",
                    caps.psDevCapabilities.hal_version,
                    caps.psDevCapabilities.fw_version,
                    caps.psDevCapabilities.hw_version,
                    caps.psDevCapabilities.model_id,
                    caps.psDevCapabilities.hci_version,
                    caps.psDevCapabilities.full_version[NXP_FULL_VERSION_LEN-1],
                    caps.psDevCapabilities.full_version[NXP_FULL_VERSION_LEN-2],
                    caps.psDevCapabilities.firmware_update_info);
        }
        /*下载成功*/
        status = NFCSTATUS_SUCCESS;
clean_and_return:
    nfc_cb_data_deinit(&cb_data);
    return status;
}
static int nfc_jni_configure_driver(struct nfc_jni_native_data *nat)
{
        char value[PROPERTY_VALUE_MAX];
        int result = FALSE;
        NFCSTATUS status;
        gDrvCfg.nClientId = phDal4Nfc_msgget(0, 0600);

        TRACE("phLibNfc_Mgt_ConfigureDriver(0x%08x)", gDrvCfg.nClientId);
        REENTRANCE_LOCK();
        status = phLibNfc_Mgt_ConfigureDriver(&gDrvCfg, &gHWRef);
```

```
    REENTRANCE_UNLOCK();
    if(status == NFCSTATUS_ALREADY_INITIALISED) {
        ALOGW("phLibNfc_Mgt_ConfigureDriver() returned 0x%04x[%s]", status, nfc_jni_get_
        status_name(status));
    }
    else if(status != NFCSTATUS_SUCCESS)
    {
        ALOGE("phLibNfc_Mgt_ConfigureDriver() returned 0x%04x[%s]", status, nfc_jni_get_
        status_name(status));
        goto clean_and_return;
    }
    TRACE("phLibNfc_Mgt_ConfigureDriver() returned 0x%04x[%s]", status, nfc_jni_get_status_
    name(status));

    if(pthread_create(&(nat->thread), NULL, nfc_jni_client_thread, nat) != 0)
    {
        ALOGE("pthread_create failed");
        goto clean_and_return;
    }
    driverConfigured = TRUE;
clean_and_return:
    return result;
}
static int nfc_jni_unconfigure_driver(struct nfc_jni_native_data *nat)
{
    int result = FALSE;
    NFCSTATUS status;
    /*取消驱动*/
    TRACE("phLibNfc_Mgt_UnConfigureDriver()");
    REENTRANCE_LOCK();
    status = phLibNfc_Mgt_UnConfigureDriver(gHWRef);
    REENTRANCE_UNLOCK();
    if(status != NFCSTATUS_SUCCESS)
    {
        ALOGE("phLibNfc_Mgt_UnConfigureDriver() returned error 0x%04x[%s] -- this should
        never happen", status, nfc_jni_get_status_name( status));
    }
    else
    {
        ALOGD("phLibNfc_Mgt_UnConfigureDriver() returned 0x%04x[%s]", status, nfc_jni_get_
        status_name(status));
        result = TRUE;
    }
    driverConfigured = FALSE;
    return result;
}
```

11.3.3　分析底层

在 Android 系统中，NFC 模块的底层部分由驱动部分和 libnfc 库部分两大类。驱动部分在“device”目录中实现，而“libnfc-nci”和“libnfc-nxp”目录中的底层文件则负责 NFC 数据的读取和解析工作。

11.4　在 Android 系统中开发 NFC App 的方法

当 Android 手机开启了 NFC 程序，并且检测到一个 TAG 后，TAG 分发系统会自动创建一个封装了 NFC TAG 信息的 Intent。如果多于一个应用程序能够处理这个 Intent，那么手机就会弹出一个对话框，让用户选择处理该 TAG 的 Activity。在 TAG 分发系统中定义了 3 种 Intent，按优先级从高到低排列如下。

- NDEF_DISCOVERED。
- TECH_DISCOVERED。
- TAG_DISCOVERED。

当 Android 设备检测到有 NFC TAG 靠近时，会根据 Action 申明的顺序向对应的 Activity 发送包含 NFC 消息的 Intent。因为此处使用的 intent-filter 的 Action 类型为 TECH_DISCOVERED，所以可以处理所有类型为 ACTION_TECH_DISCOVERED 的、并且使用的技术为在 nfc_tech_filter.xml 文件中定义的类型的 TAG。

当 Android 手机检测到一个 TAG 时，启用 Activity 的匹配过程如图 11-3 所示。

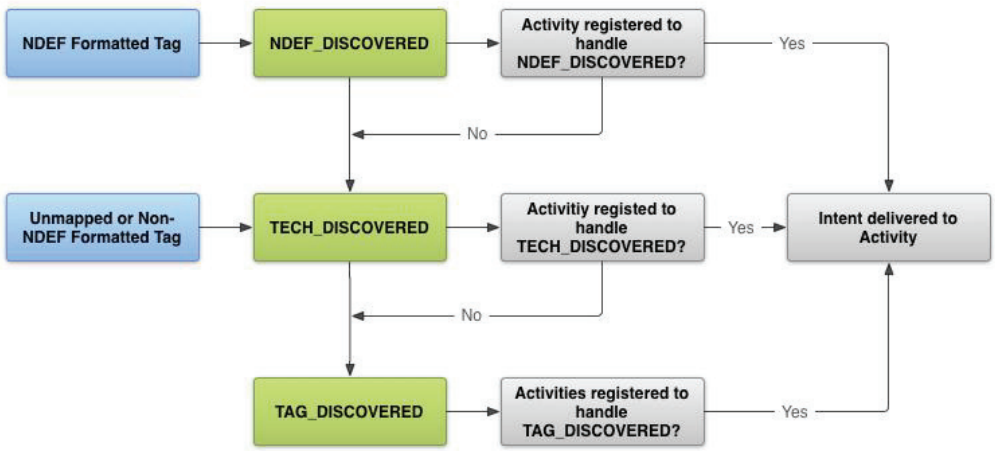


图 11-3　启用 Activity 的匹配过程

在 Android 系统中，开发 NFC App 的基本流程如下。

（1）在相关的 androidManifest 文件中设置 NFC 权限，具体代码如下。

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

（2）设置 SDK 的级别限制，例如设置为 API 10。

```
<uses-sdk android:minSdkVersion="10"/>
```

（3）声明特殊功能的限制权限，通过如下代码声明可以让应用程序在 Google Play 上声明使用者必须拥有 NFC 功能。

```
<uses-feature android:name="android.hardware.nfc" android:required="true" />
```

（4）实现 NFC 标签过滤

在 Activity 的 Intent 过滤 XML 声明中，可以同时声明过滤如下三种 Action，但是需要提前获知系统在发送 Intent 时拥有的优先级。

❑ 动作一：过滤 ACTION_TAG_DISCOVERED，代码如下。

```
<intent-filter>
    <action android:name="android.nfc.action.TAG_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

这个最简单，也是最后一个被尝试接受 intent 的选项。

❑ 动作二：过滤 ACTION_NDEF_DISCOVERED，代码如下。

```
<intent-filter>
<action android:name="android.nfc.action.NDEF_DISCOVERED"/>
<category android:name="android.intent.category.DEFAULT"/>
<data android:mimeType="text/plain" />
 </intent-filter>
```

其中最重要的是 data 的 mimeType 类型，此定义越准确，Intent 指向这个 Activity 的成功率就越高，否则系统可能不会发出想要的 NDEF intent 了。

❑ 动作三：过滤 ACTION_TECH_DISCOVERED。

首先需要在<project-path>/res/xml 下面创建一个过滤规则文件，可以任意命名，例如可以命名为 nfc_tech_filter.xml。这个里面定义的是 NFC 实现的各种标准，每一个 NFC 卡都会符合多个不同的标准。可以在检测到 NFC 标签后，使用 getTechList()方法来查看所检测的 TAG 到底支持哪些 NFC 标准。在一个 nfc_tech_filter.xml 文件中可以定义多个<tech-list>结构组，每一组表示只接受同时满足这些标准的 NFC 标签。比如 A 组表示，只有同时满足 IsoDep、NfcA、NfcB、NfcF 这 4 个标准的 NFC 标签的 Intent 才能进入。B 组表示，只有同时满足 NfcV、Ndef、NdefFormatable、MifareClassic 和 MifareUItralight 这 5 个标准的 NFC 标签的 Intent 才能进入。A 与 B 组之间的关系就是只要满足其中一个就可以了。换句话说，NFC 标签技术满足 A 的声明也可以，满足 B 的声明也可以。

```
    <resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list> ------------------------------A 组
    <tech>android.nfc.tech.IsoDep</tech> <tech>android.nfc.tech.NfcA</tech>
<tech>android.nfc.tech.NfcB</tech> <tech>android.nfc.tech.NfcF</tech>
    </tech-list>
    <tech-list>----------------------------------------B 组
```

```
    <tech>android.nfc.tech.NfcV</tech> <tech>android.nfc.tech.Ndef</tech>
<tech>android.nfc.tech.NdefFormatable</tech>
<tech>android.nfc.tech.MifareClassic</tech>
<tech>android.nfc.tech.MifareUltralight</tech>
    </tech-list>
    </resources>
```

在 androidManifest 文件中，声明 xml 过滤的举例代码如下。

```
<activity>
  <intent-filter>
<action android:name="android.nfc.action.TECH_DISCOVERED"/>
</intent-filter>
<meta-data    android:name="android.nfc.action.TECH_DISCOVERED"          android:resource=
"@xml/nfc_tech_filter" />----------这个就是你的资源文件名
  </activity>
```

（5）创建 NFC 标签的前台分发系统

什么是 NFC 的前台发布系统？就是当打开应用的时候，通过前台发布系统的设置，可以让已经启动的 Activity 拥有更高的优先级，且依据在代码中定义的标准来过滤和处理 Intent，而不是让其他声明了 Intent Filter 的 Activity 来干扰，甚至连自己声明在文件 androidManifest 中的 Intent Filter 都不能干扰。也就是说，前台分发的优先级大于 Intent Filter。此时有如下的两种情况。

- 第一种情况：当 Activity 没有启动的时候去扫描 TAG，那么系统中所有的 Intent Filter 都将一起参与过滤。
- 第二种情况：当 Actiity 启动去扫描 TAG 时，将直接使用在前台分发系统代码中写入的过滤标准。如果这个标准没有命中任何 Intent，那么系统将使用所有 Activity 声明的 Intent Filter xml 来过滤。

例如在 OnCreate 中，可以添加如下的代码。

```
    mPendingIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
            // 做一个 IntentFilter 过滤你想要的 action 这里过滤的是 ndef
            IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        //如果对 action 的定义有更高的要求，比如 data 的要求，可以使用如下的代码来定义 intentFilter
        //          try {
        //              ndef.addDataType("*/*");
        //          } catch (MalformedMimeTypeException e) {

        //              // TODO Auto-generated catch block

        //              e.printStackTrace();

        //          }
          //生成 intentFilter
```

```
                mFilters = new IntentFilter[] {
                        ndef,
                };
                // 做一个 tech-list。可以看到是二维数据，每一个一维数组之间的关系是或，但是一个一
维数组之内的各个项就是与的关系了
                mTechLists = new String[][] {
                        new String[] { NfcF.class.getName()},
                        new String[]{NfcA.class.getName()},
                        new String[]{NfcB.class.getName()},
                        new String[]{NfcV.class.getName()}
                        };
```

在 onPause 和 onResume 中需要加入如下相应的代码。

```
    public void onPause() {
      super.onPause();
    //反注册 mAdapter.disableForegroundDispatch(this);
    }
    public void onResume() {
      super.onResume();
    //设定 intentfilter 和 tech-list。如果两个都为 null 就代表优先接收任何形式的 TAG Action。也就是
    //说系统会主动发 TAG intent。
    mAdapter.enableForegroundDispatch(this, mPendingIntent, mFilters, mTechLists);
    }
```

11.5 实战演练——使用 NFC 发送消息

当 Android 设备检测到有 NFC TAG 时，预期的行为是触发最合适的 Activity 来处理检测到的 TAG，这是因为 NFC 通常是在非常近的距离才起作用(<4m)。如果在此时需要用户来选择合适的应用来处理 TAG，则很容易断开与 TAG 之间的通信，因此我们需要选择合适的 Intent Filter 只处理想读写的 TAG 类型。Android 系统支持两种 NFC 消息发送机制，分别是 Intent 发送机制和前台 Activity 消息发送机制。

- Intent 发送机制：当系统检测到 TAG 时，Android 系统提供 manifest 中定义的 Intent Filter 来选择合适的 Activity 来处理对应的 TAG，当有多个 Activity 可以处理对应的 TAG 类型时，则会显示 Activity 选择窗口由用户选择，如图 11-4 所示。

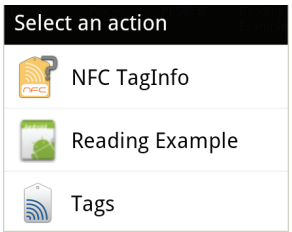


图 11-4 选择窗口

- 前台 Activity 消息发送机制：允许一个在前台运行的 Activity 在读写 NFC TAG 时具有优先权，此时如果 Android 检测到有 NFC TAG，且如果前台允许的 Activity 可以处理该种类型的 TAG，则该 Activity 具有优先权，不出现 Activity 选择窗口。

上述两种方法基本上都是使用 Intent Filter 来指明 Activity 可以处理的 TAG 类型，一个是使用 Android 的 Manifest 来说明，一个是通过代码来声明。

在接下来的内容中，将通过一个具体实例的实现过程，来讲解在 Android 系统中使用 NFC 消息发送机制的基本方法。

实　例	功　能	源 码 路 径
实例 11-1	演示 NFC 消息发送机制的基本用法	daima\11\NFCEX

本实例的具体实现流程如下。

（1）在文件 AndroidManifest.xml 中声明 NFC 权限，具体实现代码如下。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.pstreets.nfc"
        android:versionCode="1"
        android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />
     <uses-permission android:name="android.permission.NFC" />
     <uses-feature android:name="android.hardware.nfc" android:required="true" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".NFCDemoActivity"
                  android:label="@string/app_name"
                  android:launchMode="singleTop">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
              <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
              <data android:mimeType="text/plain" />
            </intent-filter>
            <intent-filter
                    >
                    <action
                        android:name="android.nfc.action.TAG_DISCOVERED"
                        >
                    </action>
                    <category
                        android:name="android.intent.category.DEFAULT"
                        >
                    </category>
              </intent-filter>
             <!-- Add a technology filter -->
            <intent-filter>
```

```
                <action android:name="android.nfc.action.TECH_DISCOVERED" />
            </intent-filter>
            <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/filter_nfc"
            />
        </activity>
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

这样通过上述声明代码，当 Android 检测到有 TAG 时，会显示 Activity 选择窗口，显示效果如前面的图 11–4 所示。

（2）编写布局文件 main.xml，功能是通过文本控件显示当前的扫描状态，具体实现代码如下。

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:text="@string/title">
  <TableLayout
        android:id="@+id/purchScanTable1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TableRow
            android:id="@+id/table1Row1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
            <TextView
                android:id="@+id/status_label"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Current Status:    " />
            <TextView
                android:id="@+id/status_data"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text=" Scan a Tag" />
        </TableRow>
        <View
```

```
                android:id="@+id/purchScanES1"
                android:layout_width="match_parent"
                 android:layout_height="55px"
                 android:layout_below="@id/status_label"
                 android:background="#000000" />
        <TableRow
                 android:id="@+id/table1Row2"
                 android:layout_width="wrap_content"
                 android:layout_height="wrap_content">
                 <TextView
                        android:id="@+id/block_0_label"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="BLOCK 0:    " />
                 <TextView
                        android:id="@+id/block_0_data"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text=" " />
        </TableRow>
        <TableRow
                 android:id="@+id/table1Row3"
                 android:layout_width="wrap_content"
                 android:layout_height="wrap_content">
                 <TextView
                        android:id="@+id/block_1_label"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="BLOCK 1:    " />
                 <TextView
                        android:id="@+id/block_1_data"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text=" " />
        </TableRow>
</TableLayout>
<View
                android:id="@+id/purchScanES1"
                android:layout_width="match_parent"
                 android:layout_height="75px"
                 android:layout_below="@id/purchScanTable1"
                 android:background="#000000" />
<Button
                        android:id="@+id/clear_but"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
```

```
                android:layout_below="@id/purchScanES1"
                android:gravity="center_horizontal"
                android:text="Clear" />
    </RelativeLayout>
```

（3）编写程序文件 NFCDemoActiviy.java，当在前台运行 NFCDemoActiviy 时，如果希望只由它来处理 Mifare 类型的 TAG，此时可以使用前台消息发送机制。文件 NFCDemoActiviy.java 的具体实现代码如下。

```
public class NFCDemoActivity extends Activity {
    private NfcAdapter mAdapter;
    private PendingIntent mPendingIntent;
    private IntentFilter[] mFilters;
    private String[][] mTechLists;
    private TextView mText;
    private int mCount = 0;
    @Override
    public void onCreate(Bundle savedState) {
        super.onCreate(savedState);
        setContentView(R.layout.foreground_dispatch);
        mText = (TextView) findViewById(R.id.text);
        mText.setText("Scan a tag");
        mAdapter = NfcAdapter.getDefaultAdapter(this);
        mPendingIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        // 为 MIME 设置一个过滤器
        IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_TECH_DISCOVERED);
        try {
            ndef.addDataType("*/*");
        } catch (MalformedMimeTypeException e) {
            throw new RuntimeException("fail", e);
        }
        mFilters = new IntentFilter[] {
                ndef,
        };
        // 为 MifareClassic 设置 Tag 列表
        mTechLists = new String[][] { new String[] { MifareClassic.class.getName() } };
    }

    @Override
    public void onResume() {
        super.onResume();
        mAdapter.enableForegroundDispatch(this, mPendingIntent, mFilters, mTechLists);
    }
    @Override
    public void onNewIntent(Intent intent) {
```

```
            Log.i("Foreground dispatch", "Discovered tag with intent: " + intent);
            mText.setText("Discovered tag " + ++mCount + " with intent: " + intent);
        }
        @Override
        public void onPause() {
            super.onPause();
            mAdapter.disableForegroundDispatch(this);
        }
    }
```

这样在执行本实例后，每当靠近一次 TAG，计数就会增加 1。执行效果如图 11-5 所示。

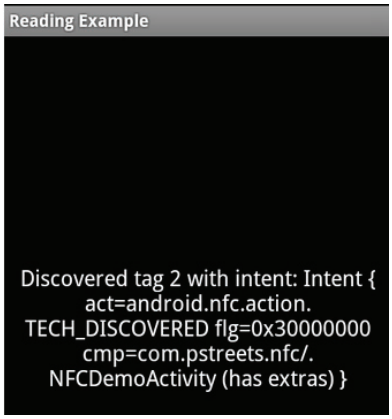


图 11-5　执行效果

第 12 章　开发电子邮件应用程序

自从互联网诞生以来，电子邮件（E-Mail）就成为吸引用户使用网络的主要原因之一。无论是朋友之间的祝福和交流，还是商务活动中的信息往来，都离不开电子邮件。在本章的内容中，将详细介绍在 Android 平台中开发邮件应用程序的基本知识。

12.1　在 Android 中发送邮件的方式

在开发 Android 网络应用程序时，可以有多种实现邮件发送的方式。在本节的内容中，将详细讲解使用 Android 系统发送邮件的常见方式，并通过具体实例来讲解其实现过程。

12.1.1　使用 Intent 方式

在 Android 系统中，可以使用 Intent 调用内置的邮件系统的方法来发送邮件。在具体实现时，需要调用 Intent 中的 putExtra()方法将邮件的各个部分发送 E-Mail 程序。方法 putExtra()的语法格式如下。

```
intent.putExtra(android.content.Intent.EXTRA_STREAM,URL url);
```

第一个参数 EXTRA_STREAM 表示传输的数据，各个取值的具体说明如下。

- EXTRA_E-MAIL：发送的是收件人地址。
- EXTRA_SUBJECT：邮件标题。
- EXTRA_TEXT：邮件文本内容。
- EXTRA_STREAM：邮件附件。

第二个参数 url 表示传递对象的 URL 地址。

在使用 Intent 调用内置邮件系统时，使用的行为是 android.content.Intent.ACTION_SEND。实际上在 Android 系统中使用的邮件发送服务是调用 Gmail 程序，而并不是直接使用 SMTP 协议。

在接下来的内容中，将通过一个具体实例来讲解在发送短信时实现 E-Mail 邮件通知的实现过程。在本实例中，当用户收到一条短信后，先用 Toast 提示获取了短信，然后使用 E-Mail 发送提示到用户的邮箱中，这样就可以将重要的短信放在邮箱中保存，从而不用担心短信容量的问题了。

实　例	功　能	源码路径
实例 12-1	在收到短信时实现 E-Mail 邮件通知	daima\12\tong

在具体实现上，先在后台设计一个 BroadcastReseiver 用于等待接收短信。当接收到短信以后，使用 Bundle 的方式来封装短信的内容，然后通过 Intent 方式返回给主程序 Activity。因为 Receiver 无法直接发送 E-Mail，所以需要将控制权回给主程序，通过主程序来运行发送 E-Mail 的工作。当主程序收到 Bundle 后，会以 Bundle.getString 方法来取得返回短信的内容，然后以 Intent.setType(“plain/text”)来设置要打开的 Intent 类型，并以关键程序 Intent.putExtra (android.content.Intent.EXTRA_E-MAIL,strE-mailReciver)来指定要打开的是发送 E-Mail 所需要的 Extra 参数，当 Android 系统收到这些参数后，就会打开内置的 E-Mail 发送程序。读者在此需要注意的是，在模拟器运行后会显示“No application can perform this action”的提示，而在真实机器上不会出现此问题。

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，通过 TextView 控件显示界面文字，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:background="@drawable/white"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
>
   <TextView
      android:id="@+id/myTextView1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
   />
</LinearLayout>
```

（2）编写文件 AndroidManifest.xml，向系统注册一个常驻的 BroadcastReseiver，并设置这个 Reseiver 的 intent-filter，让其 SMSreceiver 针对收到短信事件做出反应，并声明 android.permission.RECEIVE_SMS 权限。文件 AndroidManifest.xml 的具体实现代码如下。

```
 <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
     </activity>
     <!-- 建立 receiver 来聆听系统广播信息 -->
     <receiver android:name="irdc.tong.SMSreceiver">
     <!-- 设置要捕捉的信息名是 provider 中 Telephony.SMS_RECEIVED -->
   <intent-filter>
     <action
        android:name="android.provider.Telephony.SMS_RECEIVED" />
   </intent-filter>
   </receiver>
     </application>
```

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

（3）编写程序文件 tong.java，通过 try 语句获取短信传来的 bundle 堆信息，并取出 bunde 中的字符串，然后自定义 Intent 来发送 E-Mail 邮件信息，同时设置邮件格式为“plain/text”，并分别取得 EditText01、EditText02、EditText03 和 EditText 04 的值作为收件人地址、附件、主题和正文，最后将取得的字符串放入 mE-mailIntent 中。文件 tong.java 的具体实现代码如下。

```
package irdc.tong;
import irdc.tong.R;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
public class tong extends Activity
{
  /*声明一个 TextView,String 数组与两个文本字符串变量*/
  private TextView mTextView1;
  public String[] strE-mailReciver;
  public String strE-mailSubject;
  public String strE-mailBody;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*通过 findViewById 构造器创建 TextView 对象*/
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mTextView1.setText("等待中...");
    try
    {
      /*取得短信传来的 bundle*/
      Bundle bunde = this.getIntent().getExtras();
      if (bunde!= null)
      {
        /*将 bunde 内的字符串取出*/
        String sb = bunde.getString("STR_INPUT");
        /*自定义一 Intent 来运行寄送 E-mail 的工作*/
        Intent mE-mailIntent =
        new Intent(android.content.Intent.ACTION_SEND);
        /*设置邮件格式为"plain/text"*/
        mE-mailIntent.setType("plain/text");
        /*
        * 取得 EditText01,02,03,04 的值作为
```

```
                * 收件人地址,附件,主题,正文
                */
                strE-mailReciver =new String[]{"jay.mingchieh@gmail.com"};
                strE-mailSubject = "你有一封短信!!";
                strE-mailBody = sb.toString();
                /*将取得的字符串放入 mE-mailIntent 中*/
                mE-mailIntent.putExtra(android.content.Intent.EXTRA_E-MAIL,
                strE-mailReciver);
                mE-mailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT,
                strE-mailSubject);
                mE-mailIntent.putExtra(android.content.Intent.EXTRA_TEXT,
                strE-mailBody);
                startActivity(Intent.createChooser(mE-mailIntent,
                getResources().getString(R.string.str_message)));
            }
            else
            {
                finish();
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

（4）编写文件 SMSreceiver.java，使用 telephoney.gsm.SmsMessage 来接收短信，使用 Toast 类来通知用户收到短信，主要代码如下。文件 SMSreceiver.java 的具体实现代码如下。

```
/*引用 BroadcastReceiver 类*/
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
/*引用 telephoney.gsm.SmsMessage 来收取短信*/
import android.telephony.gsm.SmsMessage;
/*引用 Toast 类来告知用户收到短信*/
import android.widget.Toast;
/*自定义继承自 BroadcastReceiver 类,聆听系统服务广播的信息 */
public class SMSreceiver extends BroadcastReceiver
{
    /*
    * 声明静态字符串,并使用
    * android.provider.Telephony.SMS_RECEIVED
```

```
* 作为 Action 为短信的依据
*/
private static final String mACTION =
"android.provider.Telephony.SMS_RECEIVED";
private String str_receive="收到短信!";
@Override
public void onReceive(Context context, Intent intent)
{
  // TODO Auto-generated method stub
  Toast.makeText(context, str_receive.toString(),
  Toast.LENGTH_LONG).show();
  /*判断传来 Intent 是否为短信*/
  if (intent.getAction().equals(mACTION))
  {
    /*建构一字符串集合变量 sb*/
    StringBuilder sb = new StringBuilder();
    /*接收由 Intent 传来的数据*/
    Bundle bundle = intent.getExtras();
    /*判断 Intent 是有数据*/
    if (bundle != null)
    {
      /* pdus 为 android 内置短信参数 identifier
       * 通过 bundle.get("")返回一包含 pdus 的对象*/
      Object[] myOBJpdus = (Object[]) bundle.get("pdus");
      /*构造短信对象 array,并依据收到的对象长度来创建 array 的大小*/
      SmsMessage[] messages = new SmsMessage[myOBJpdus.length];
      for (int i = 0; i<myOBJpdus.length; i++)
      {
        messages[i] =
        SmsMessage.createFromPdu((byte[]) myOBJpdus[i]);
      }
      /* 将送来的短信合并自定义信息于 StringBuilder 当中 */
      for (SmsMessage currentMessage : messages)
      {
        sb.append("接收到来自:\n");
        /* 收信人的电话号码 */
        sb.append(currentMessage.getDisplayOriginatingAddress());
        sb.append("\n------传来的短信------\n");
        /* 取得传来信息的 BODY */
        sb.append(currentMessage.getDisplayMessageBody());
        Toast.makeText
        (
          context, sb.toString(), Toast.LENGTH_LONG
        ).show();
      }
```

```
            }
            /*  以 Notification(Toase)显示来讯信息    */
            Toast.makeText
            (
               context, sb.toString(), Toast.LENGTH_LONG
            ).show();
            /*  返回主 Activity */
            Intent i = new Intent(context, tong.class);
            /*自定义一 Bundle*/
            Bundle mbundle = new Bundle();
            /*将短信信息以 putString()方法存入自定义的 bundle 内*/
            mbundle.putString("STR_INPUT",    sb.toString());
            /*将自定义 bundle 写入 Intent 中*/
            i.putExtras(mbundle);
            /*设置 Intent 的 Flag 以一个全新的 task 来运行*/
            i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(i);
        }
    }
}
```

实例执行后的效果如图 12-1 所示。如果收到一条短信则会显示提示信息，并自动生成一条邮件提示，如图 12-2 所示。

在模拟器中也可以运行本项目，方法是在 Eclipse 中打开 DDMS 面板，在 Telephony Actions 中设置手机号码和短信内容，并单击 Send 按钮，如图 12-3 所示。

图 12-1　初始运行效果

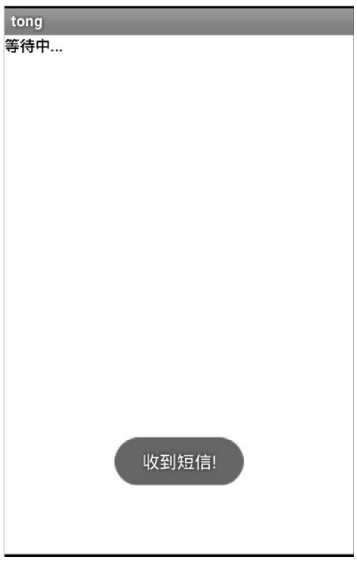


图 12-2　收到提示信息

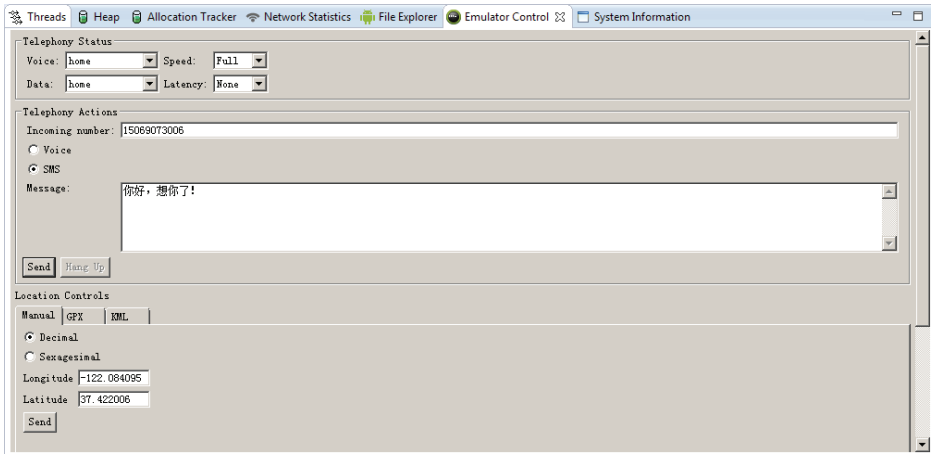


图 12-3　Eclipse 的 DDMS 面板

12.1.2　使用 SmsManager 收发邮件

在 Android 系统中，除了可以使用 Intent 调用内置邮件系统发送邮件外，还可以使用类 SmsManager 来收发邮件。在 Android 平台中，类 SmsManager 是用来管理短信服务操作，像发送数据、文本和数据单元等短信服务消息。可以通过调用 SmsManager.getDefault()来获取 SmsManager 对象。

1. 常量

- public static final int RESULT_ERROR_GENERIC_FAILURE：表示普通错误，值为 1(0x00000001)。
- public static final int RESULT_ERROR_NO_SERVICE：表示当前服务不可用，值为 4 (0x00000004)。
- public static final int RESULT_ERROR_NULL_PDU：表示没有提供 pdu，值为 3 (0x00000003)。
- public static final int RESULT_ERROR_RADIO_OFF：表示无线广播被明确地关闭，值为 2 (0x00000002)。
- public static final int STATUS_ON_ICC_FREE：表示自由空间，值为 0 (0x00000000)。
- public static final int STATUS_ON_ICC_READ：表示接收且已读，值为 1 (0x00000001)。
- public static final int STATUS_ON_ICC_SENT：表示存储且已发送，值为 5 (0x00000005)。
- public static final int STATUS_ON_ICC_UNREAD：表示接收但未读，值为 3 (0x00000003)。
- public static final int STATUS_ON_ICC_UNSENT：表示存储但未发送，值为 7 (0x00000007)。

2. 公有方法

（1）ArrayList<String> divideMessage(String text)

功能：当短信超过 SMS 消息的最大长度时，将短信分割为几块。

参数 text：初始的消息，不能为空。

返回值：有序的 ArrayList<String>，可以重新组合为初始的消息。

（2）static SmsManager getDefault()

功能：获取 SmsManager 的默认实例。

返回值：SmsManager 的默认实例。

（3）void SendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)

功能：发送一个基于 SMS 的数据到指定的应用程序端口。

参数介绍如下。

- destinationAddress：消息的目标地址。
- scAddress：服务中心的地址或为空，则使用当前默认的 SMSC。
- destinationPort：消息的目标端口号。
- data：消息的主体，即消息要发送的数据。
- sentIntent：如果不为空，当消息成功发送或失败，这个 PendingIntent 就广播。结果代码是 Activity.RESULT_OK 表示成功，或为 RESULT_ERROR_GENERIC_FAILURE、RESULT_ERROR_RADIO_OFF、RESULT_ERROR_NULL_PDU 之一则表示错误。当为 RESULT_ERROR_GENERIC_FAILURE 时，sentIntent 可能包括额外的“错误代码”，例如通常包含一个无线电广播技术特定的值，这只在修复故障时有用。每一个基于 SMS 的应用程序控制检测 sentIntent。如果 sentIntent 是空，调用者将检测所有未知的应用程序，这将导致在检测的时候发送较小数量的 SMS。
- deliveryIntent：如果不为空，当消息成功传送到接收者，这个 PendingIntent 就广播。

异常：如果 destinationAddress 或 data 为空时，抛出 IllegalArgumentException 异常。

（4）void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList <String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent>　deliverIntents)

功能：发送一个基于 SMS 的多部分的文本，调用者通过调用 divideMessage(String text) 将消息分割成正确的大小。

参数介绍如下。

- destinationAddress：消息的目标地址。
- scAddress：服务中心的地址，为空则使用当前默认的 SMSC。
- parts：有序的 ArrayList<String>，可以重新组合为初始的消息。
- sentIntents：是一组 PendingIntent。
- deliverIntents：是一组 PendingIntent。

异常：如果 destinationAddress 或 data 为空时，抛出 IllegalArgumentException 异常。

（5）void sendTextMessage(String destinationAddress, String scAddress, String text, Pending Intent sentIntent, PendingIntent deliveryIntent)

功能：发送一个基于 SMS 的文本。参数的意义和异常与前面 4 个公有方法中的一样，在此不再累述。

在下面的实例中定义了两个 EditText 控件，分别用于获取收信人电话和短信正文，并设置了判断手机号码规范化的方法，并且设置了短信的字数不超过 70 个字符。本实例通过

SmsManage 对象的 sendTextMessage()方法来完成的。在 sendTextMessage()方法中要传入 5 个值，分别是：收件人地址 String、发送地址 String、正文 String、发送服务 PendingIntent 和送达服务 PendingIntent。

实　例	功　能	源码路径
实例 12-2	使用 SmsManager 实现一个邮件发送程序	daima\12\jiandan

本实例的具体实现流程如下。

（1）编写布局文件 main.xml，主要代码如下。

```
    <TextView
      android:id="@+id/widget27"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/str_textview"
      android:textSize="16sp"
      android:layout_x="0px"
      android:layout_y="12px"
    >
    </TextView>
    <EditText
      android:id="@+id/myEditText1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text=""
      android:textSize="18sp"
      android:layout_x="60px"
      android:layout_y="2px"
    >
    </EditText>
    <EditText
      android:id="@+id/myEditText2"
      android:layout_width="fill_parent"
      android:layout_height="223px"
      android:text=""
      android:textSize="18sp"
      android:layout_x="0px"
      android:layout_y="52px"
    >
    </EditText>
    <Button
      android:id="@+id/myButton1"
      android:layout_width="162px"
      android:layout_height="wrap_content"
      android:text="@string/str_button1"
      android:layout_x="80px"
```

```
    android:layout_y="302px"
>
</Button>
```

（2）编写主程序文件 jiandan.java，声明一个 Button 和两个 EditText，EditText 供获取输入收信人电话号码和短信内容，Button 按钮用于激活发信处理程序。通过方法 PendingIntent.getBroadcast() 自定义了 PendingIntent 并进行了 Broadcast 广播，然后使用 SmsManager.getDefault()预先构建的 SmsManager 对象，使用 sendTextMessage()方法将有关的数据以参数形式带入，这样即可完成发短信的任务。文件 jiandan.java 的具体代码如下。

```
public class jiandan extends Activity
{
  /*声明变量一个 Button 与两个 EditText*/
  private Button mButton1;
  private EditText mEditText1;
  private EditText mEditText2;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /*
    * 通过 findViewById 构造器来建构
    * EditText1,EditText2 与 Button 对象
    */
    mEditText1 = (EditText) findViewById(R.id.myEditText1);
    mEditText2 = (EditText) findViewById(R.id.myEditText2);
    mButton1 = (Button) findViewById(R.id.myButton1);
    /*将默认文字加载 EditText 中*/
    mEditText1.setText("请输入号码");
    mEditText2.setText("请输入内容!!");
    /*设置 onClickListener 让用户单击 EditText 时做出反应*/
    mEditText1.setOnClickListener(new EditText.OnClickListener()
    {
      public void onClick(View v)
      {
        /*单击 EditText 时清空正文*/
        mEditText1.setText("");
      }
    }
    );
    /*设置 onClickListener 让用户单击 EditText 时做出反应*/
    mEditText2.setOnClickListener(new EditText.OnClickListener()
    {
      public void onClick(View v)
```

```
        {
            /*单击 EditText 时清空正文*/
            mEditText2.setText("");
        }
    }
    );
    /*设置 onClickListener 让用户单击 Button 时做出反应*/
    mButton1.setOnClickListener(new Button.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            /*由 EditText1 取得短信收件人电话*/
            String strDestAddress = mEditText1.getText().toString();
            /*由 EditText2 取得短信文字内容*/
            String strMessage = mEditText2.getText().toString();
            /*建构一取得 default instance 的 SmsManager 对象 */
            SmsManager smsManager = SmsManager.getDefault();
            // TODO Auto-generated method stub
            /*检查收件人电话格式与短信字数是否超过 70 字符*/
            if(isPhoneNumberValid(strDestAddress)==true &&
                iswithin70(strMessage) ==true)
            {
                try
                {
                    /*
                    * 两个条件都检查通过的情况下,发送短信
                    * 先建构一 PendingIntent 对象并使用 getBroadcast()广播
                    * 将 PendingIntent,电话,短信文字等参数
                    * 传入 sendTextMessage()方法发送短信
                    */
                    PendingIntent mPI = PendingIntent.getBroadcast
                    (jiandan.this, 0, new Intent(), 0);
                    smsManager.sendTextMessage
                    (strDestAddress, null, strMessage, mPI, null);
                }
                catch(Exception e)
                {
                    e.printStackTrace();
                }
                Toast.makeText
                (
                    jiandan.this,"送出成功!!" ,
                    Toast.LENGTH_SHORT
                ).show();
                mEditText1.setText("");
                mEditText2.setText("");
```

```
            }
            else
            {
               /*  电话格式与短信文字不符合条件时,以 Toast 提醒  */
               if (isPhoneNumberValid(strDestAddress) ==false)
               { /*且字数超过 70 字符*/
                  if(iswithin70(strMessage) ==false)
                  {
                     Toast.makeText
                     (
                        jiandan.this,
                        "电话号码格式错误+短信内容超过 70 字,请检查!!",
                        Toast.LENGTH_SHORT
                     ).show();
                  }
                  else
                  {
                     Toast.makeText
                     (
                        jiandan.this,
                        "电话号码格式错误,请检查!!" ,
                        Toast.LENGTH_SHORT
                     ).show();
                  }
               }
               /*字数超过 70 字符*/
               else if (iswithin70(strMessage) ==false)
               {
                  Toast.makeText
                  (
                     jiandan.this,
                     "短信内容超过 70 字,请删除部分内容!!",
                     Toast.LENGTH_SHORT
                  ).show();
               }
            }
         }
      });

   }
   /*检查字符串是否为电话号码的方法,并返回 true or false 的判断值*/
   public static boolean isPhoneNumberValid(String phoneNumber)
   {
      boolean isValid = false;
      /*  可接受的电话格式有:
       * ^\\(? :  可以使用  "("  作为开头
       * (\\d{3}):  紧接着三个数字
       * \\)? :  可以使用")"接续
```

```
        * [- ]? : 在上述格式后可以使用具选择性的 "-".
        * (\\d{3}) : 再紧接着三个数字
        * [- ]? : 可以使用具选择性的 "-" 接续.
        * (\\d{5})$: 以五个数字结束.
        * 可以比较下列数字格式:
        * (123)456-7890, 123-456-7890, 1234567890, (123)-456-7890
        */
        String expression =
        "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{5})$";
        /* 可接受的电话格式有:
        * ^\\(? : 可以使用 "(" 作为开头
        * (\\d{3}): 紧接着三个数字
        * \\)? : 可以使用")"接续
        * [- ]? : 在上述格式后可以使用具选择性的 "-".
        * (\\d{4}) : 再紧接着四个数字
        * [- ]? : 可以使用具选择性的 "-" 接续.
        * (\\d{4})$: 以四个数字结束.
        * 可以比较下列数字格式:
        * (02)3456-7890, 02-3456-7890, 0234567890, (02)-3456-7890
        */
        String expression2=
        "^\\(?(\\d{3})\\)?[- ]?(\\d{4})[- ]?(\\d{4})$";
        CharSequence inputStr = phoneNumber;
        /*创建 Pattern*/
        Pattern pattern = Pattern.compile(expression);
        /*将 Pattern 以参数传入 Matcher 作 Regular expression*/
        Matcher matcher = pattern.matcher(inputStr);
        /*创建 Pattern2*/
        Pattern pattern2 =Pattern.compile(expression2);
        /*将 Pattern2 以参数传入 Matcher2 作 Regular expression*/
        Matcher matcher2= pattern2.matcher(inputStr);
        if(matcher.matches()||matcher2.matches())
        {
            isValid = true;
        }
        return isValid;
    }
    public static boolean iswithin70(String text)
    {
        if (text.length()<= 70)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

```
}
```

（3）最后编写文件 AndroidManifest.xml，向系统注册一个常驻的 BroadcastReseiver，并设置这个 Reseiver 的 intent-filter，让其 SMSreceiver 针对收到短信事件做出反应，并声明 android.permission.RECEIVE_SMS 权限。文件 AndroidManifest.xml 的具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="irdc.tong"
   android:versionCode="1"
   android:versionName="1.0.0">
      <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name="irdc.tong.tong"
         android:label="@string/app_name">
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
      </activity>
      <!-- 建立 receiver 来聆听系统广播信息 -->
      <receiver android:name="irdc.tong.SMSreceiver">
      <!-- 设置要捕捉的信息名是 provider 中 Telephony.SMS_RECEIVED -->
   <intent-filter>
      <action
         android:name="android.provider.Telephony.SMS_RECEIVED" />
   </intent-filter>
   </receiver>
      </application>
<uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
</manifest>
```

执行后的效果如图 12-4 所示，输入手机号码，编写短信内容后，单击“发送”按钮后即可完成短信发送功能，系统会提示发送成功信息，如图 12-5 所示。

图 12-4　初始效果

图 12-5　发送成功

如果短信内容和收信人号码格式不规范，会输出对应的错误提示。

12.2 向本地联系人发送邮件

在下面的实例中，利用 Android 提供的 Intent 接口实现邮件发送功能。在发送邮件功能中，使用了 Intent 行为 android.content.Intent.ACTION_SEND。在本实例中，使用的邮件发送服务是调用 Gmail 程序，而并不是直接使用 SMTP 协议。本实例实现了如下的功能。

- 验证用户输入是否为正确的邮箱格式。
- 使用者既可以手动输入邮箱，也可以长按邮箱文本框跳到联系人那里找到联系人，得到联系人的邮箱后返回。
- 发送邮件。

实　例	功　能	源码路径
实例 12-3	向本地联系人发送邮件	daima\12\sendE-mail

在接下来的内容中，将详细讲解本实例的具体实现过程。

12.2.1 界面布局

编写布局文件 main.xml，分别提供了 4 个 EditText 表单供用户输入收件人、附件、主题和邮件内容信息，并通过 Button 控件提供一个发送按钮。文件 main.xml 的具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
   android:id="@+id/widget34"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@drawable/white"
   xmlns:android="http://schemas.android.com/apk/res/android"
   >
   <TextView
      android:id="@+id/myTextView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/str_receive"
      android:layout_x="60px"
      android:layout_y="22px"
   >
   </TextView>
   <TextView
      android:id="@+id/myTextView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/str_cc"
      android:layout_x="60px"
```

```
    android:layout_y="82px"
>
</TextView>
<EditText
    android:id="@+id/myEditText1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="12px"
>
</EditText>
<EditText
    android:id="@+id/myEditText2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="72px"
>
</EditText>
<Button
    android:id="@+id/myButton1"
    android:layout_width="wrap_content"
    android:layout_height="124px"
    android:text="@string/str_button"
    android:layout_x="0px"
    android:layout_y="2px"
>
</Button>
<TextView
    android:id="@+id/myTextView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/str_subject"
    android:layout_x="60px"
    android:layout_y="142px"
>
</TextView>
<EditText
    android:id="@+id/myEditText3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="120px"
    android:layout_y="132px"
```

```
        >
        </EditText>
        <EditText
           android:id="@+id/myEditText4"
           android:layout_width="fill_parent"
           android:layout_height="209px"
           android:textSize="18sp"
           android:layout_x="0px"
           android:layout_y="202px"
        >
         </EditText>
    </AbsoluteLayout>
```

12.2.2 编写主程序文件

编写主程序文件 sendE-mailActivity.java，具体实现过程如下。

- 通过函数 isEmail()验证邮箱格式是否正确。
- 当从一个 Activity 跳转到另一个 Activity 时，为了实现得到需要的数据后返回到原有的 Activitiy 所在的控件上，上述跳转功能都需要借助于 StartAcivity()方法实现。如果想实现跳转并取值返回，就需要用到 startActivityForResult(intent,requestCode)，其中 requestCode 为一个 Activity 要返回的值的依据，可以为任意的 int 类型。可以自己定义常量，也可以自己指定数字。程序覆盖了 onActivityResult()这个方法，程序收到 result 后，再重新写回原本需要加载的控件上。本例中调了文本框的长按事件，文本框长按即自行跳转到联系人页面上，单击需要的联系人名称，则返回该联系人的邮箱号回到主程序窗口并加载到文本上。
- 通过 OnLongClickListener searhE-mail 事件处理程序，用户长按文本框后会通过 Content Provider 查找并跳转到联系人中心，查找用户并返回邮箱。如果要公开设备中的联系信息数据，可以创建或者使用一个 Content Provider。Content Provider 是一个能使所有应用程度都能被存储和检索数据的对象，是唯一在包和包之间分享数据的方法，这是因为不存在那种供所有的包来共享的一般存储空间。在 Android 系统中自带了一些 Content Provider，用于处理一般的数据类型，例如音频、视频、图片和个人联系信息等。虽然可以从 Provider 包中看到一些 Android 自带的 Content Provider，但是 Content Provider 如何存储数据，决定于这个 Content Provider 如何实现。但是所有的 Content Provider 必须实现一种一般的约定规则，才能实现数据查询并返回结果。然而，一个 Content Provider 能够实现自定义的方法，使得在处理一些特定的数据时，对于数据的存储/检索更加简单。
- 实现邮件发送处理。在 EditText、Button 控件中通过构造一个自定义的 Intent（android.content.Intent.ACTION_SEND）作为传送 E-Mail 的 Activity。在这个 Intent 中还必须使用 setType()来决定 E-Mail 的格式，使用 putExtra() 来置入寄件人（EXTRA_E-MAIL）、主题（EXTRA_SUBJECT）、邮件内容（EXTRA_TEXT）以及其他 E-Mail 的字段（EXTRA_BCC、EXTRA_CC）。

文件 sendE-mailActivity.java 的具体实现代码如下。

```
public class sendE-mailActivity extends Activity {
    private Button myButton;
    private EditText myEditText;
    private EditText myEditText2;
    private EditText myEditText3;
    private EditText myEditText4;
    private String[] strE-mailReciver;
    private String strE-mailSubject;
    private String[] strE-mailCC;
    private String strE-mailBody;
    private static final int PICK_CONTACT_SUBACTIVITY = 2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myButton=(Button)findViewById(R.id.myButton1);
        myEditText=(EditText)findViewById(R.id.myEditText1);
        myEditText2=(EditText)findViewById(R.id.myEditText2);
        myEditText3=(EditText)findViewById(R.id.myEditText3);
        myEditText4=(EditText)findViewById(R.id.myEditText4);
        myButton.setEnabled(false);
        myEditText.setOnLongClickListener(searhE-mail);
        myButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent mailIntent=new Intent(android.content.Intent.ACTION_SEND);
                mailIntent.setType("plain/test");
                strE-mailReciver=new String[]{ myEditText.getText().toString() };
                strE-mailCC=new String[]{myEditText2.getText().toString()};
                strE-mailSubject=myEditText3.getText().toString();
                strE-mailBody=myEditText4.getText().toString();
                mailIntent.putExtra(android.content.Intent.EXTRA_E-MAIL, strE-mailReciver);
                mailIntent.putExtra(android.content.Intent.EXTRA_CC, strE-mailCC);
                mailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, strE-mailSubject);
                mailIntent.putExtra(android.content.Intent.EXTRA_TEXT, strE-mailBody);
            startActivity(Intent.createChooser(mailIntent, getResources().getString (R.string.send)));
            /*   第二种方法
            Uri uri=Uri.parse("mailto:terryyhl@gmail.com");
            Intent MymailIntent=new Intent(Intent.ACTION_SEND,uri);
            startActivity(MymailIntent);
            */

            /* 第三种方法
            Intent testintent=new Intent(Intent.ACTION_SEND);
            String[] tos={"terryyhl@gmail.com"};
```

```
                String[] ccs={"kalaicheng@hotmail.com"};
                testintent.putExtra(Intent.EXTRA_E-MAIL, tos);
                testintent.putExtra(Intent.EXTRA_CC, ccs);
                testintent.putExtra(Intent.EXTRA_TEXT, "这是内容");
                testintent.putExtra(Intent.EXTRA_SUBJECT, "这是标题");
                testintent.setType("message/rfc822");
                startActivity(Intent.createChooser(testintent, "发送"));

                */
                /*
                 /*第四种方法，传附件
                 Intent testN=new Intent(Intent.ACTION_SEND);
                  testN.putExtra(Intent.EXTRA_SUBJECT, "标题");
                  testN.putExtra(Intent.EXTRA_STREAM, "file:///sdcard/music.mp3");
                  startActivity(Intent.createChooser(testN, "发送"));
                 */
                }
            });


        myEditText.setOnKeyListener(new OnKeyListener() {
                @Override
                public boolean onKey(View v, int keyCode, KeyEvent event) {
                    // TODO Auto-generated method stub
                    if(isE-mail(myEditText.getText().toString()))
                    {
                        myButton.setEnabled(true);
                    }
                    else
                    {
                        myButton.setEnabled(false);
                    }
                    return false;
                }
            });
    }
    // 检查 E-Mail 格式
    public static boolean isE-mail(String strE-mail) {
      String                                    strPattern                                    =
"^[a-zA-Z][\\w\\.-]*[a-zA-Z0-9]@[a-zA-Z0-9][\\w\\.-]*[a-zA-Z0-9]\\.[a-zA-Z][a-zA-Z\\.]*[a-zA-Z]$";
      Pattern p = Pattern.compile(strPattern);
      Matcher m = p.matcher(strE-mail);
      return m.matches();
    }
    private OnLongClickListener searhE-mail=new OnLongClickListener(){
       public boolean onLongClick(View arg0) {
```

```
            Uri uri=Uri.parse("content://contacts/people");
            Intent intent=new Intent(Intent.ACTION_PICK,uri);
            startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);
                return false;
                }
    ;
};
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
 switch (requestCode) {
        case PICK_CONTACT_SUBACTIVITY:
                final Uri uriRet=data.getData();
                if(uriRet!=null)
                {
                    try {
                        Cursor c=managedQuery(uriRet, null, null, null, null);
                        c.moveToFirst();
                        //取得联系人的姓名
                        String strName=c.getString(c.getColumnIndexOrThrow(People.NAME));
                        //取得联系人的 E-Mail
                        String[] PROJECTION=new String[]{
                                Contacts.ContactMethods._ID,
                                Contacts.ContactMethods.KIND,
                                Contacts.ContactMethods.DATA
                        };
                        //查询指定人的 E-Mail
                         Cursor newcur=managedQuery(
                                 Contacts.ContactMethods.CONTENT_URI,
                                 PROJECTION,
                                 Contacts.ContactMethods.PERSON_ID+"=\'"
                                 +c.getLong(c.getColumnIndex(People._ID))+"\'",
                                 null, null);
                        startManagingCursor(newcur);
                        String E-mail="";
                        if(newcur.moveToFirst())
                        {
                            E-mail=newcur.getString(newcur.getColumnIndex
                                    (Contacts.ContactMethods.DATA));
                            myEditText.setText(E-mail);
                        }
                    } catch (Exception e) {
                        // TODO: handle exception
                        Toast.makeText(sendE-mailActivity.this, e.toString(), 1000).show();
                    }
                }
                break;
            default:
```

```
                    break;
                }
                super.onActivityResult(requestCode, resultCode, data);
            }
        ;
```

在上述实现代码中，还演示了在 Android 系统中发送 E-Mail 的其他方法：

- 方法 1：使用 parse 方法。

```
Uri uri=Uri.parse("mailto:terryyhl@gmail.com");
Intent MymailIntent=new Intent(Intent.ACTION_SEND,uri);
startActivity(MymailIntent);
```

- 方法 2：使用 Intent 创建新界面。

```
Intent testintent=new Intent(Intent.ACTION_SEND);
String[] tos={"terryyhl@gmail.com"};
String[] ccs={"kalaicheng@hotmail.com"};
testintent.putExtra(Intent.EXTRA_E-MAIL, tos);
testintent.putExtra(Intent.EXTRA_CC, ccs);
testintent.putExtra(Intent.EXTRA_TEXT, "这是内容");
testintent.putExtra(Intent.EXTRA_SUBJECT, "这是标题");
testintent.setType("message/rfc822");
startActivity(Intent.createChooser(testintent, "发送"));
```

- 方法 3：将 SD 卡中的音乐文件作为附件发送。

```
Intent testN=new Intent(Intent.ACTION_SEND);
testN.putExtra(Intent.EXTRA_SUBJECT, "标题");
testN.putExtra(Intent.EXTRA_STREAM, "file:///sdcard/music.mp3");
startActivity(Intent.createChooser(testN, "发送"));
```

本实例执行后的效果如图 12-6 所示。

图 12-6　执行效果

第 13 章　Android 网络典型应用实践

经过本书前面内容的学习，Android 网络开发技术已经讲解结束。在本章的内容中，将详细介绍 Android 在网络领域的常用模块，不但探讨了新技术，而且对前面的内容进行整体回顾，以稳固所学知识。

13.1　测试网络下载速度

用户一般比较关心网速的问题，大家都不喜欢过慢的下载速度。所以在 Anroid 平台中，很有必要开发一个网速测试程序。在接下来的演示代码中，将通过下载文件的大小和当前读取的字节数，在固定的时间内检测网络速度。

（1）首先定义网络信息类 NetWorkSpeedInfo，在里面设置了需要的常量值。

```
packagecc.androidos.speed;
publicclassNetWorkSpeedInfo
{
    /**网速*/
    publiclongspeed=0;
    /**已完成的字节*/
    publiclonghadFinishedBytes=0;
    /**文件的总大小*/
    publiclongtotalBytes=1024;
    /**网络类型，3G/GSM*/
    publicintnetworkType=0;
    /**下载进度 0----100*/
    publicintdownloadPercent=0;
}
```

（2）开始编写一个名为 SpeedActivity 的主 Activity，获取网络速度。

```
packagecc.androidos.speed;
importandroid.app.Activity;
importandroid.graphics.Bitmap;
importandroid.graphics.BitmapFactory;
importandroid.os.Bundle;
importandroid.os.Handler;
importandroid.os.Message;
importandroid.util.Log;
importandroid.view.View;
```

```
importandroid.widget.Button;
importandroid.widget.ImageView;
importandroid.widget.TextView;
publicclassSpeedActivityextendsActivity
{
TextViewfileLength=null;
TextViewspeed=null;
TextViewhasDown=null;
TextViewpercent=null;
Stringurl="";
ImageViewimageView=null;
byte[]imageData=null;
NetWorkSpeedInfonetWorkSpeedInfo=null;
privatefinalintUPDATE_SPEED=1;
@Override
publicvoidonCreate(BundlesavedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
hasDown=(TextView)findViewById(R.id.hasDown);
fileLength=(TextView)findViewById(R.id.fileLength);
speed=(TextView)findViewById(R.id.speed);
percent=(TextView)findViewById(R.id.percent);
imageView=(ImageView)findViewById(R.id.ImageView01);
Buttonb=(Button)findViewById(R.id.Button01);
url=getString(R.string.image_url);
netWorkSpeedInfo=newNetWorkSpeedInfo();
b.setOnClickListener(newView.OnClickListener()
{
@Override
publicvoidonClick(Viewarg0)
{
//下载文件线程
newThread()
{
@Override
publicvoidrun()
{
imageData=ReadFile.getFileFromUrl(url,
netWorkSpeedInfo);
stop();
}
}.start();
//获取速度，加载字节，更新视图线程
newThread()
{
```

```
@Override
publicvoidrun()
{
while(netWorkSpeedInfo.hadFinishedBytes<netWorkSpeedInfo.totalBytes)
{
netWorkSpeedInfo.downloadPercent=(int)(((double)netWorkSpeedInfo.hadFinishedBytes/
(double)netWorkSpeedInfo.totalBytes)*100);
try
{
sleep(1500);
}
catch(InterruptedExceptione)
{
e.printStackTrace();
}
Log.e("update,sendthemessagetoupdate","");
handler.sendEmptyMessage(UPDATE_SPEED);
}
//已经完成的
if(netWorkSpeedInfo.hadFinishedBytes==netWorkSpeedInfo.totalBytes)
{
netWorkSpeedInfo.downloadPercent=(int)(((double)netWorkSpeedInfo.hadFinishedBytes/
(double)netWorkSpeedInfo.totalBytes)*100);
handler.sendEmptyMessage(UPDATE_SPEED);
Log.e("update",
",sendthemessagetoupdateandstop");
stop();
}
}
}.start();
}
});
}
privateHandlerhandler=newHandler()
{
@Override
publicvoidhandleMessage(Messagemsg)
{
intvalue=msg.what;
switch(value)
{
caseUPDATE_SPEED:
updateView();
break;
default:
break;
```

```
}
}
};
privatevoidupdateView()
{
speed.setText(netWorkSpeedInfo.speed+"bytes/s");
hasDown.setText(netWorkSpeedInfo.hadFinishedBytes+"bytes");
fileLength.setText(netWorkSpeedInfo.totalBytes+"");
percent.setText(netWorkSpeedInfo.downloadPercent+"%");
if(imageData!=null)
{
Bitmapb=BitmapFactory.decodeByteArray(imageData,0,
imageData.length);
imageView.setImageBitmap(b);
}
}
}
```

（3）编写读取指定 Web 文件的代码，用于通过读取这个指定的文件来测试速度。

```
packagecc.androidos.speed;
importjava.io.InputStream;
importjava.net.URL;
importjava.net.URLConnection;
importandroid.util.Log;
publicclassReadFil{
publicstaticbyte[]getFileFromUrl(Stringurl,NetWorkSpeedInfonetWorkSpeedInfo)
{
intcurrentByte=0;
intfileLength=0;
longstartTime=0;
longintervalTime=0;
byte[]b=null;
intbytecount=0;
URLurlx=null;
URLConnectioncon=null;
InputStreamstream=null;
tr{
Log.d("URL:",url);
urlx=newURL(url);
con=urlx.openConnection();
con.setConnectTimeout(20000);
con.setReadTimeout(20000);
fileLength=con.getContentLength();
stream=con.getInputStream();
netWorkSpeedInfo.totalBytes=fileLength;
```

```
b=newbyte[fileLength];
startTime=System.currentTimeMillis();
while((currentByte=stream.read())!=-1)
{
netWorkSpeedInfo.hadFinishedBytes++;
intervalTime=System.currentTimeMillis()-startTime;
if(intervalTime==0){
netWorkSpeedInfo.speed=1000;
}else{
netWorkSpeedInfo.speed=(netWorkSpeedInfo.hadFinishedBytes/intervalTime)*1000;
}
if(bytecount<fileLength){
b[bytecount++]=(byte)currentByte;
}
}
}
catch(Exceptione)
{
Log.e("exception:",e.getMessage()+"");
}
finall{
tr{
if(stream!=null)
{
stream.close();
}
}
catch(Exceptione)
{
Log.e("exception:",e.getMessage());
}
}
returnb;
}
}
```

13.2　通过 Hander 实现异步消息处理

在 Android 系统中，可以与服务器端实现 HTTP 通信，并解析 XML 数据，通过 Handler 实现异步消息处理。在接下来的演示代码中，分别实现了如下三个重要操作。

（1）HTTP 通信：与服务器端做 HTTP 通信，分别以 GET 方式和 POST 方式做演示。

（2）XML 解析：可以用两种方式解析 XML，分别是 DOM 方式和 SAX 方式。

（3）异步消息处理：通过 Handler 实现异步消息处理，以一个自定义的异步下载类来说明

Handler 的用法。

13.2.1 实现 HTTP 通信和 XML 解析的演示

（1）首先定义一个名为 MySAXHandler 的类，此类继承于 DefaultHandler，功能是实现指定 XML 的 SAX 解析器。并且在下面的代码中使用了 SAX 流式解析方式，通过事件模型解析 XML，这种方式只能顺序解析。

```
packagecom.webabcd.communication;
importorg.xml.sax.Attributes;
importorg.xml.sax.SAXException;
importorg.xml.sax.helpers.DefaultHandler;
//采用 DOM-W3C 标准，需要把 XML 数据全部加载完成后才能对其做解析，可对树做任意遍历
publicclassMySAXHandlerextendsDefaultHandler{
privatebooleanmIsTitleTag=false;
privatebooleanmIsSalaryTag=false;
privatebooleanmIsBirthTag=false;
privateStringmResult="";
//打开 XML 文档的回调函数
@Override
publicvoidstartDocument()throwsSAXException{
//TODOAuto-generatedmethodstub
super.startDocument();
}

//关闭 XML 文档的回调函数
@Override
publicvoidendDocument()throwsSAXException{
//TODOAuto-generatedmethodstub
super.endDocument();
}
//一发现元素开始标记就回调此函数
@Override
publicvoidstartElement(Stringuri,StringlocalName,StringqName,
Attributesattributes)throwsSAXException{
if(localName=="title")
mIsTitleTag=true;
elseif(localName=="salary")
mIsSalaryTag=true;
elseif(localName=="dateOfBirth")
mIsBirthTag=true;
elseif(localName=="employee")
mResult+="\nname:"+attributes.getValue("name");
}
//一发现元素结束标记就回调此函数
```

```
@Override
publicvoidendElement(Stringuri,StringlocalName,StringqName)
throwsSAXException{
if(localName=="title")
mIsTitleTag=false;
elseif(localName=="salary")
mIsSalaryTag=false;
elseif(localName=="dateOfBirth")
mIsBirthTag=false;
}
//一发现元素值或属性值就回调此函数
@Override
publicvoidcharacters(char[]ch,intstart,intlength)
throwsSAXException{
if(mIsTitleTag)
mResult+=newString(ch,start,length);
elseif(mIsSalaryTag)
mResult+="salary:"+newString(ch,start,length);
elseif(mIsBirthTag)
mResult+="dateOfBirth:"+newString(ch,start,length);
}

publicStringgetResult(){
returnmResult;
}
}
```

（2）在下面的演示代码中主要定义了如下两个核心方法。

- 方法 httpGetDemo()：以 HTTP 协议的 get()方法获取远程页面响应的内容。
- 方法 httpPostDemo()：以 HTTP 协议的 post()方法向远程页面传递参数，并获取其响应的内容。

```
packagecom.webabcd.communication;
importjava.io.BufferedInputStream;
importjava.io.BufferedReader;
importjava.io.IOException;
importjava.io.InputStream;
importjava.io.InputStreamReader;
importjava.net.HttpURLConnection;
importjava.net.URL;
importjava.net.URLConnection;
importjava.util.ArrayList;
importjava.util.HashMap;
importjava.util.Map;
```

```
importjavax.xml.parsers.DocumentBuilder;
importjavax.xml.parsers.DocumentBuilderFactory;
importjavax.xml.parsers.SAXParser;
importjavax.xml.parsers.SAXParserFactory;
importorg.apache.http.HttpEntity;
importorg.apache.http.HttpResponse;
importorg.apache.http.client.entity.UrlEncodedFormEntity;
importorg.apache.http.client.methods.HttpPost;
importorg.apache.http.impl.client.DefaultHttpClient;
importorg.apache.http.message.BasicNameValuePair;
importorg.apache.http.protocol.HTTP;
importorg.apache.http.util.ByteArrayBuffer;
importorg.apache.http.util.EncodingUtils;
importorg.w3c.dom.Document;
importorg.w3c.dom.Element;
importorg.w3c.dom.NodeList;
importorg.xml.sax.InputSource;
importorg.xml.sax.XMLReader;
importandroid.app.Activity;
importandroid.os.Bundle;
importandroid.view.View;
importandroid.widget.Button;
importandroid.widget.TextView;
publicclassMainextendsActivity{
privateTextViewtextView;
@Override
publicvoidonCreate(BundlesavedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
textView=(TextView)this.findViewById(R.id.textView);
Buttonbtn1=(Button)this.findViewById(R.id.btn1);
btn1.setText("httpgetdemo");
btn1.setOnClickListener(newButton.OnClickListener(){
publicvoidonClick(Viewv){
httpGetDemo();
}
});
Buttonbtn2=(Button)this.findViewById(R.id.btn2);
btn2.setText("httppostdemo");
btn2.setOnClickListener(newButton.OnClickListener(){
publicvoidonClick(Viewv){
httpPostDemo();
}
});
Buttonbtn3=(Button)this.findViewById(R.id.btn3);
btn3.setText("DOM 解析 XML");
```

```
btn3.setOnClickListener(newButton.OnClickListener(){
publicvoidonClick(Viewv){
DOMDemo();
}
});
Buttonbtn4=(Button)this.findViewById(R.id.btn4);
btn4.setText("SAX 解析 XML");
btn4.setOnClickListener(newButton.OnClickListener(){
publicvoidonClick(Viewv){
SAXDemo();
}
});
}
//Android 调用 HTTP 协议的 get()方法
//本例：以 HTTP 协议的 get()方法获取远程页面响应的内容
privatevoidhttpGetDemo(){
try{
//模拟器测试时，请使用外网地址
URLurl=newURL("http://xxx.xxx.xxx");
URLConnectioncon=url.openConnection();
Stringresult="httpstatuscode:"+((HttpURLConnection)con).getResponseCode()+"\n";
InputStreamis=con.getInputStream();
BufferedInputStreambis=newBufferedInputStream(is);
ByteArrayBufferbab=newByteArrayBuffer(32);
intcurrent=0;
while((current=bis.read())!=-1){
bab.append((byte)current);
}
result+=EncodingUtils.getString(bab.toByteArray(),HTTP.UTF_8);
bis.close();
is.close();
textView.setText(result);
}catch(Exceptione){
textView.setText(e.toString());
}
}
//Android 调用 HTTP 协议的 post()方法
//本例：以 HTTP 协议的 post()方法向远程页面传递参数，并获取其响应的内容
privatevoidhttpPostDemo(){
try{
//模拟器测试时，请使用外网地址
Stringurl="http://5billion.com.cn/post.php";
Map<String,String>data=newHashMap<String,String>();
data.put("name","webabcd");
data.put("salary","100");
DefaultHttpClienthttpClient=newDefaultHttpClient();
```

```
HttpPosthttpPost=newHttpPost(url);
ArrayList<BasicNameValuePair>postData=newArrayList<BasicNameValuePair>();
for(Map.Entry<String,String>m:data.entrySet()){
postData.add(newBasicNameValuePair(m.getKey(),m.getValue()));
}
UrlEncodedFormEntityentity=newUrlEncodedFormEntity(postData,HTTP.UTF_8);
httpPost.setEntity(entity);
HttpResponseresponse=httpClient.execute(httpPost);
Stringresult="httpstatuscode:"+response.getStatusLine().getStatusCode()+"\n";
HttpEntityhttpEntity=response.getEntity();
InputStreamis=httpEntity.getContent();
result+=convertStreamToString(is);
textView.setText(result);
}catch(Exceptione){
textView.setText(e.toString());
}
}
//以 DOM 方式解析 XML
privatevoidDOMDemo(){
try{
DocumentBuilderFactorydocFactory=DocumentBuilderFactory.newInstance();
DocumentBuilderdocBuilder=docFactory.newDocumentBuilder();
Documentdoc=docBuilder.parse(this.getResources().openRawResource(R.raw.employee));
ElementrootElement=doc.getDocumentElement();
NodeListemployeeNodeList=rootElement.getElementsByTagName("employee");
textView.setText("DOMDemo"+"\n");
Stringtitle=rootElement.getElementsByTagName("title").item(0).getFirstChild().getNodeValue();
textView.append(title);
for(inti=0;i<employeeNodeList.getLength();i++){
ElementemployeeElement=((Element)employeeNodeList.item(i));
Stringname=employeeElement.getAttribute("name");
Stringsalary=employeeElement.getElementsByTagName("salary").item(0).getFirstChild().getNodeValue();
StringdateOfBirth=employeeElement.getElementsByTagName("dateOfBirth").item(0).getFirstChild().ge
tNodeValue();
textView.append("\nname:"+name+"salary:"+salary+"dateOfBirth:"+dateOfBirth);
}
}catch(Exceptione){
textView.setText(e.toString());
}
}
//以 SAX 方式解析 XML
privatevoidSAXDemo(){
try{
SAXParserFactorysaxFactory=SAXParserFactory.newInstance();
SAXParserparser=saxFactory.newSAXParser();
XMLReaderreader=parser.getXMLReader();
```

```
MySAXHandlerhandler=newMySAXHandler();
reader.setContentHandler(handler);
reader.parse(newInputSource(this.getResources().openRawResource(R.raw.employee)));
Stringresult=handler.getResult();
textView.setText("SAXDemo"+"\n");
textView.append(result);
}catch(Exceptione){
textView.setText(e.toString());
}
}
//辅助方法，用于把流转换为字符串
privateStringconvertStreamToString(InputStreamis){
BufferedReaderreader=newBufferedReader(newInputStreamReader(is));
StringBuildersb=newStringBuilder();
Stringline=null;
try{
while((line=reader.readLine())!=null){
sb.append(line+"\n");
}
}catch(IOExceptione){
e.printStackTrace();
}finally{
try{
is.close();
}catch(IOExceptione){
e.printStackTrace();
}
}
returnsb.toString();
}
}
```

13.2.2　使用 Handler 实现异步消息处理

在接下来的演示代码中，以一个可以实时汇报下载进度的异步下载类为例，开发了一个 Android 类库。在以下演示代码中此类库的名字为 webabcd_util。打开 Eclipse，依次单击 New→Java Project，然后在项目上单击右键并依次选择 Build Path→Add Libraries→User Library→User Libraries→New，这样可为类库起一个名字，然后选中这个类库，单击 Add JARs 导入 Android 的 jar 包。最后在项目上单击右键，依次选择 Build Path→Add Libraries→User Library 选择 Android 库。

```
packagewebabcd.util;
importjava.io.BufferedReader;
importjava.io.File;
importjava.io.FileOutputStream;
```

```
importjava.io.InputStream;
importjava.io.InputStreamReader;
importjava.net.URL;
importjava.net.URLConnection;
importorg.apache.http.protocol.HTTP;
importandroid.os.Handler;
importandroid.os.Message;
importandroid.util.Log;
//以一个异步下载实例，来演示 Android 的异步消息处理（用 Handler 的方式）
publicclassDownloadManagerAsync{
publicDownloadManagerAsync(){
}
//实例化自定义的 Handler
EventHandlermHandler=newEventHandler(this);
//按指定 url 地址下载文件到指定路径
publicvoiddownload(finalStringurl,finalStringsavePath){
newThread(newRunnable(){
publicvoidrun(){
try{
sendMessage(FILE_DOWNLOAD_CONNECT);
URLsourceUrl=newURL(url);
URLConnectionconn=sourceUrl.openConnection();
InputStreaminputStream=conn.getInputStream();
intfileSize=conn.getContentLength();
Filesavefile=newFile(savePath);
if(savefile.exists()){
savefile.delete();
}
savefile.createNewFile();
FileOutputStreamoutputStream=newFileOutputStream(
savePath,true);
byte[]buffer=newbyte[1024];
intreadCount=0;
intreadNum=0;
intprevPercent=0;
while(readCount<fileSize&&readNum!=-1){
readNum=inputStream.read(buffer);
if(readNum>-1){
outputStream.write(buffer);
readCount=readCount+readNum;
intpercent=(int)(readCount*100/fileSize);
if(percent>prevPercent){
//发送下载进度信息
sendMessage(FILE_DOWNLOAD_UPDATE,percent,
readCount);
```

```
prevPercent=percent;
}
}
}
outputStream.close();
sendMessage(FILE_DOWNLOAD_COMPLETE,savePath);
}catch(Exceptione){
sendMessage(FILE_DOWNLOAD_ERROR,e);
Log.e("MyError",e.toString());
}
}
}).start();
}
//读取指定 URL 地址的响应内容
publicvoiddownload(finalStringurl){
newThread(newRunnable(){
publicvoidrun(){
try{
sendMessage(FILE_DOWNLOAD_CONNECT);
URLsourceUrl=newURL(url);
URLConnectionconn=sourceUrl.openConnection();
conn.setConnectTimeout(3000);
BufferedReaderreader=newBufferedReader(
newInputStreamReader(conn.getInputStream(),
HTTP.UTF_8));
Stringline=null;
StringBuffercontent=newStringBuffer();
while((line=reader.readLine())!=null){
content.append(line);
}
reader.close();
sendMessage(FILE_DOWNLOAD_COMPLETE,content.toString());
}catch(Exceptione){
sendMessage(FILE_DOWNLOAD_ERROR,e);
Log.e("MyError",e.toString());
}
}
}).start();
}
//向 Handler 发送消息
privatevoidsendMessage(intwhat,Objectobj){
//构造需要向 Handler 发送的消息
Messagemsg=mHandler.obtainMessage(what,obj);
//发送消息
mHandler.sendMessage(msg);
```

```
}
privatevoidsendMessage(intwhat){
Messagemsg=mHandler.obtainMessage(what);
mHandler.sendMessage(msg);
}
privatevoidsendMessage(intwhat,intarg1,intarg2){
Messagemsg=mHandler.obtainMessage(what,arg1,arg2);
mHandler.sendMessage(msg);
}
privatestaticfinalintFILE_DOWNLOAD_CONNECT=0;
privatestaticfinalintFILE_DOWNLOAD_UPDATE=1;
privatestaticfinalintFILE_DOWNLOAD_COMPLETE=2;
privatestaticfinalintFILE_DOWNLOAD_ERROR=-1;
//自定义的 Handler
privateclassEventHandlerextendsHandler{
privateDownloadManagerAsyncmManager;
publicEventHandler(DownloadManagerAsyncmanager){
mManager=manager;
}
//处理接收到的消息
@Override
publicvoidhandleMessage(Messagemsg){
switch(msg.what){
caseFILE_DOWNLOAD_CONNECT:
if(mOnDownloadConnectListener!=null)
mOnDownloadConnectListener.onDownloadConnect(mManager);
break;
caseFILE_DOWNLOAD_UPDATE:
if(mOnDownloadUpdateListener!=null)
mOnDownloadUpdateListener.onDownloadUpdate(mManager,
msg.arg1);
break;
caseFILE_DOWNLOAD_COMPLETE:
if(mOnDownloadCompleteListener!=null)
mOnDownloadCompleteListener.onDownloadComplete(mManager,
msg.obj);
break;
caseFILE_DOWNLOAD_ERROR:
if(mOnDownloadErrorListener!=null)
mOnDownloadErrorListener.onDownloadError(mManager,
(Exception)msg.obj);
break;
default:
break;
}
```

```
}
}
//定义连接事件
privateOnDownloadConnectListenermOnDownloadConnectListener;
publicinterfaceOnDownloadConnectListener{
voidonDownloadConnect(DownloadManagerAsyncmanager);
}
publicvoidsetOnDownloadConnectListener(OnDownloadConnectListenerlistener){
mOnDownloadConnectListener=listener;
}
//定义下载进度更新事件
privateOnDownloadUpdateListenermOnDownloadUpdateListener;
publicinterfaceOnDownloadUpdateListener{
voidonDownloadUpdate(DownloadManagerAsyncmanager,intpercent);
}
publicvoidsetOnDownloadUpdateListener(OnDownloadUpdateListenerlistener){
mOnDownloadUpdateListener=listener;
}
//定义下载完成事件
privateOnDownloadCompleteListenermOnDownloadCompleteListener;
publicinterfaceOnDownloadCompleteListener{
voidonDownloadComplete(DownloadManagerAsyncmanager,Objectresult);
}
publicvoidsetOnDownloadCompleteListener(
OnDownloadCompleteListenerlistener){
mOnDownloadCompleteListener=listener;
}
//定义下载异常事件
privateOnDownloadErrorListenermOnDownloadErrorListener;
publicinterfaceOnDownloadErrorListener{
voidonDownloadError(DownloadManagerAsyncmanager,Exceptione);
}
publicvoidsetOnDownloadErrorListener(OnDownloadErrorListenerlistener){
mOnDownloadErrorListener=listener;
}
}
```

然后调用上面自定义的 Android 类库，在项目上单击右键，然后依次选择 Properties→Java Build Path→Projects→Add 命令来引用上面的类库。

```
packagecom.webabcd.handler;
importandroid.app.Activity;
importandroid.os.Bundle;
importandroid.widget.TextView;
importwebabcd.util.DownloadManagerAsync;
```

```
publicclassMainextendsActivityimplements
DownloadManagerAsync.OnDownloadCompleteListener,
DownloadManagerAsync.OnDownloadUpdateListener,
DownloadManagerAsync.OnDownloadErrorListener{
TextViewtxt;
/**Calledwhentheactivityisfirstcreated.*/
@Override
publicvoidonCreate(BundlesavedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
DownloadManagerAsyncmanager=newDownloadManagerAsync();
manager.setOnDownloadCompleteListener(this);
manager.setOnDownloadUpdateListener(this);
manager.download("http://files.cnblogs.com/webabcd/Android.rar","/sdcard/Android.rar");
txt=(TextView)this.findViewById(R.id.txt);
txt.setText("开始下载");
}
publicvoidonDownloadComplete(DownloadManagerAsyncmanager,Objectresult){
txt.setText("下载完成");
}
publicvoidonDownloadUpdate(DownloadManagerAsyncmanager,intpercent){
txt.setText("下载进度："+String.valueOf(percent)+"%");
}
publicvoidonDownloadError(DownloadManagerAsyncmanager,Exceptione){
txt.setText("下载出错");
}
}
```

13.3 实现网络多线程断点下载

在现实应用中，直接使用单线程下载 HTTP 文件是一件非常简单的事。其实还可以尝试使用多线程断点下载。在本节将通过一个具体实例的实现过程，详细讲解在 Android 系统中实现网络多线程断点下载的方法。

13.3.1 实现原理

要想从文件的指定位置处开始文件下载，可以通过 HTTP 请求信息头来设置，需要设置 HTTP 请求信息头的“Range”属性。在解决了多线程下载问题后，接下来开始解决支持断点下载问题。整个过程非常简单，只需将下载的进度保存到文件中即可，但是在 Android 中却不能这么做。在 Android 平台中，需要向文件中写入下载的文件数据，还需要向另一个文件写入下载进度，在这个过程中通常会导致某个文件的内容没有被写入错误。所以我们就不能以文件的方式来保存下载进度，但可以通过数据库的方式来保存下载进度。

13.3.2　具体实现

（1）创建 Android 工程，整个工程的相关参数如下。

- Project name：MulThreadDownloader。
- BuildTarget：Android 4.4。
- Application name：多线程断点下载。
- Package name：com.changcheng.download。
- Create Activity：MulThreadDownloader。

（2）编写文件 AndroidManifest.xml，在此文件中主要设置如下三个权限。

- 在 SDCard 中创建与删除文件权。
- 在 SDCard 文件中选取内容。
- 在 SDCard 文件中写入数据。

具体代码如下。

```
<?xmlversion="1.0"encoding="utf-8"?>
<manifestxmlns:android="http://schemas.android.com/apk/res/android"
package="com.changcheng.download"
android:versionCode="1"
android:versionName="1.0">
<applicationandroid:icon="@drawable/icon"android:label="@string/app_name">
<activityandroid:name=".MulThreadDownloader"
android:label="@string/app_name">
<intent-filter>
<actionandroid:name="android.intent.action.MAIN"/>
<categoryandroid:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
<uses-sdkandroid:minSdkVersion="20"/>
<!--在 SDCard 中创建与删除文件权限-->
<uses-permissionandroid:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!--往 SDCard 写入数据权限-->
<uses-permissionandroid:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!--访问 internet 权限-->
<uses-permissionandroid:name="android.permission.INTERNET"/>
</manifest>
```

（3）编写文件 strings.xml，具体代码如下。

```
<?xmlversion="1.0"encoding="utf-8"?>
<resources>
<stringname="hello">HelloWorld,DownloadActivity!</string>
<stringname="app_name">多线程断点下载</string>
<stringname="path">下载路径</string>
<stringname="downloadbutton">下载</string>
```

```
<stringname="sdcarderror">SDCard 不存在或者写保护</string>
</resources>
```

（4）编写 UI 布局文件 main.xml，具体代码如下。

```
<?xmlversion="1.0"encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<!--下载路径-->
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/path"
/>
<EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="http://www.winrar.com.cn/download/wrar380sc.exe"
android:id="@+id/path"
/>
<!--下载按钮-->
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/downloadbutton"
android:id="@+id/button"
/>
<!--进度条-->
<ProgressBar
android:layout_width="fill_parent"
android:layout_height="20dip"
style="?android:attr/progressBarStyleHorizontal"
android:id="@+id/downloadbar"/>
<!--进度%-->
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:gravity="center"
android:id="@+id/resultView"
/>
</LinearLayout>
```

（5）编写文件 MulThreadDownloader.java，具体代码如下。

```
packagecom.changcheng.download;
importjava.io.File;
importcom.changcheng.net.download.DownloadProgressListener;
importcom.changcheng.net.download.FileDownloader;
importcom.changcheng.download.R;
importandroid.app.Activity;
importandroid.os.Bundle;
importandroid.os.Environment;
importandroid.os.Handler;
importandroid.os.Message;
importandroid.view.View;
importandroid.widget.Button;
importandroid.widget.EditText;
importandroid.widget.ProgressBar;
importandroid.widget.TextView;
importandroid.widget.Toast;
publicclassMulThreadDownloaderextendsActivity{
privateEditTextpathText;
privateProgressBarprogressBar;
privateTextViewresultView;
privateHandlerhandler=newHandler(){
@Override
publicvoidhandleMessage(Messagemsg){
if(!Thread.currentThread().isInterrupted()){
switch(msg.what){
case1:
//获取当前文件下载的进度
intsize=msg.getData().getInt("size");
progressBar.setProgress(size);
intresult=(int)(((float)size/(float)progressBar.getMax())*100);
resultView.setText(result+"%");
if(progressBar.getMax()==size){
Toast.makeText(MulThreadDownloader.this,"文件下载完成",1).show();
}
break;
case-1:
Stringerror=msg.getData().getString("error");
Toast.makeText(MulThreadDownloader.this,error,1).show();
break;
}
}
super.handleMessage(msg);
}
};
@Override
publicvoidonCreate(BundlesavedInstanceState){
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
pathText=(EditText)this.findViewById(R.id.path);
progressBar=(ProgressBar)this.findViewById(R.id.downloadbar);
resultView=(TextView)this.findViewById(R.id.resultView);
Buttonbutton=(Button)this.findViewById(R.id.button);
button.setOnClickListener(newView.OnClickListener(){
@Override
publicvoidonClick(Viewv){
Stringpath=pathText.getText().toString();
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
//下载文件需要很长的时间，主线程是不能够长时间被阻塞，如果主线程被长时间阻塞，那么
//Android 被回收应用
download(path,Environment.getExternalStorageDirectory());
}else{
Toast.makeText(MulThreadDownloader.this,R.string.sdcarderror,1).show();
}
}
});
}
/**
*下载文件
*@parampath 下载路径
*@paramsaveDir 文件保存目录
*/
//对于 Android 的 UI 控件，只能由主线程负责显示界面的更新，其他线程不能直接更新 UI 控件的
//显示
publicvoiddownload(finalStringpath,finalFilesaveDir){
newThread(newRunnable(){
@Override
publicvoidrun(){
FileDownloaderdowner=newFileDownloader(MulThreadDownloader.this,path,saveDir,3);
progressBar.setMax(downer.getFileSize());//设置进度条的最大刻度
try{
downer.download(newDownloadProgressListener(){
@Override
publicvoidonDownloadSize(intsize){
Messagemsg=newMessage();
msg.what=1;
msg.getData().putInt("size",size);
handler.sendMessage(msg);//发送消息
}});
}catch(Exceptione){
Messagemsg=newMessage();
msg.what=-1;
msg.getData().putString("error","下载失败");
```

```
handler.sendMessage(msg);
}
}
}).start();
}
}
```

（6）编写文件 FileDownload.java，具体代码如下。

```
packagecom.changcheng.net.download;
importjava.io.File;
importjava.io.RandomAccessFile;
importjava.net.HttpURLConnection;
importjava.net.URL;
importjava.util.LinkedHashMap;
importjava.util.Map;
importjava.util.UUID;
importjava.util.concurrent.ConcurrentHashMap;
importjava.util.regex.Matcher;
importjava.util.regex.Pattern;
importcom.changcheng.download.service.FileService;
importandroid.content.Context;
importandroid.util.Log;
/**
*文件下载器
*/
publicclassFileDownloader{
privateContextcontext;
privateFileServicefileService;
privatestaticfinalStringTAG="FileDownloader";
/*已下载文件大小*/
privateintdownloadSize=0;
/*原始文件大小*/
privateintfileSize=0;
/*线程数*/
privateDownloadThread[]threads;
/*下载路径*/
privateURLurl;
/*本地保存文件*/
privateFilesaveFile;
/*下载记录文件*/
privateFilelogFile;
/*缓存各线程最后下载的位置*/
privateMap<Integer,Integer>data=newConcurrentHashMap<Integer,Integer>();
/*每条线程下载的大小*/
privateintblock;
privateStringdownloadUrl;//下载路径
```

```
/**
*获取线程数
*/
publicintgetThreadSize(){
returnthreads.length;
}
/**
*获取文件大小
*@return
*/
publicintgetFileSize(){
returnfileSize;
}
/**
*累计已下载大小
*@paramsize
*/
protectedsynchronizedvoidappend(intsize){
downloadSize+=size;
}
/**
*更新指定线程最后下载的位置
*@paramthreadId 线程 id
*@parampos 最后下载的位置
*/
protectedvoidupdate(intthreadId,intpos){
this.data.put(threadId,pos);
}
/**
*保存记录文件
*/
protectedsynchronizedvoidsaveLogFile(){
this.fileService.update(this.downloadUrl,this.data);
}
/**
*构建文件下载器
*@paramdownloadUrl 下载路径
*@paramfileSaveDir 文件保存目录
*@paramthreadNum 下载线程数
*/
publicFileDownloader(Contextcontext,StringdownloadUrl,FilefileSaveDir,intthreadNum){
try{
this.context=context;
this.downloadUrl=downloadUrl;
fileService=newFileService(context);
this.url=newURL(downloadUrl);
```

```
if(!fileSaveDir.exists())fileSaveDir.mkdirs();
this.threads=newDownloadThread[threadNum];
HttpURLConnectionconn=(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;MSIE8.0;WindowsNT5.2;Trident/4.0;.NETCLR1.1.4322;.NETCLR2.0.50727;.NETCLR3.0.04506.30;.NETCLR3.0.4506.2152;.NETCLR3.5.30729)");
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)thrownewRuntimeException("无法获知文件大小");
Stringfilename=getFileName(conn);
this.saveFile=newFile(fileSaveDir,filename);/*保存文件*/
Map<Integer,Integer>logdata=fileService.getData(downloadUrl);
if(logdata.size()>0){
data.putAll(logdata);
}
this.block=this.fileSize/this.threads.length+1;
if(this.data.size()==this.threads.length){
for(inti=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{
thrownewRuntimeException("服务器响应错误");
}
}catch(Exceptione){
print(e.toString());
thrownewRuntimeException("连接不到下载路径");
}
}
/**
*获取文件名
*/
privateStringgetFileName(HttpURLConnectionconn){
Stringfilename=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(inti=0;;i++){
Stringmine=conn.getHeaderField(i);
if(mine==null)break;
if("content-disposition".equals(conn.getHeaderFieldKey(i).toLowerCase())){
Matcherm=Pattern.compile(".*filename=(.*)").matcher(mine.toLowerCase());
if(m.find())returnm.group(1);
}
}
filename=UUID.randomUUID()+".tmp";//默认取一个文件名
}
returnfilename;
}
/**
*开始下载文件
*@paramlistener 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
*@throwsException
*/
publicintdownload(DownloadProgressListenerlistener)throwsException{
try{
if(this.data.size()!=this.threads.length){
this.data.clear();
for(inti=0;i<this.threads.length;i++){
this.data.put(i+1,this.block*i);
}
}
for(inti=0;i<this.threads.length;i++){
intdownLength=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
RandomAccessFilerandOut=newRandomAccessFile(this.saveFile,"rw");
if(this.fileSize>0)randOut.setLength(this.fileSize);
randOut.seek(this.data.get(i+1));
this.threads[i]=newDownloadThread(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);
booleannotFinish=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(inti=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
RandomAccessFilerandOut=newRandomAccessFile(this.saveFile,"rw");
randOut.seek(this.data.get(i+1));
this.threads[i]=newDownloadThread(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(Exceptione){
print(e.toString());
thrownewException("下载失败");
}
returnthis.downloadSize;
}
/**
*获取 Http 响应头字段
*@paramhttp
*@return
*/
publicstaticMap<String,String>getHttpResponseHeader(HttpURLConnectionhttp){
Map<String,String>header=newLinkedHashMap<String,String>();
for(inti=0;;i++){
Stringmine=http.getHeaderField(i);
if(mine==null)break;
header.put(http.getHeaderFieldKey(i),mine);
}
returnheader;
}
/**
*打印 Http 头字段
*@paramhttp
*/
publicstaticvoidprintResponseHeader(HttpURLConnectionhttp){
Map<String,String>header=getHttpResponseHeader(http);
for(Map.Entry<String,String>entry:header.entrySet()){
Stringkey=entry.getKey()!=null?entry.getKey()+":":"";
print(key+entry.getValue());
}
}
privatestaticvoidprint(Stringmsg){
Log.i(TAG,msg);
```

```
}
}
```

（7）编写文件 DownloadProgressListener.java，具体代码如下。

```
packagecom.changcheng.net.download;
publicinterfaceDownloadProgressListener{
publicvoidonDownloadSize(intsize);
}
```

（8）编写文件 FileService.java，具体代码如下。

```
packagecom.changcheng.download.service;
importjava.util.HashMap;
importjava.util.Map;
importandroid.content.Context;
importandroid.database.Cursor;
importandroid.database.sqlite.SQLiteDatabase;
/**
*业务 bean
*
*/
publicclassFileService{
privateDBOpenHelperopenHelper;
publicFileService(Contextcontext){
openHelper=newDBOpenHelper(context);
}
/**
*获取线程最后下载位置
*@parampath
*@return
*/
publicMap<Integer,Integer>getData(Stringpath){
SQLiteDatabasedb=openHelper.getReadableDatabase();
Cursorcursor=db.rawQuery("selectthreadid,positionfromfiledownwheredownpath=?",newString[]{path});

Map<Integer,Integer>data=newHashMap<Integer,Integer>();
while(cursor.moveToNext()){
data.put(cursor.getInt(0),cursor.getInt(1));
}
cursor.close();
db.close();
returndata;
}
/**
*保存下载线程初始位置
*@parampath
```

```
*@parammap
*/
publicvoidsave(Stringpath,Map<Integer,Integer>map){//intthreadid,intposition
SQLiteDatabasedb=openHelper.getWritableDatabase();
db.beginTransaction();
try{
for(Map.Entry<Integer,Integer>entry:map.entrySet()){
db.execSQL("insertintofiledown(downpath,threadid,position)values(?,?,?)",
newObject[]{path,entry.getKey(),entry.getValue()});
}
db.setTransactionSuccessful();
}finally{
db.endTransaction();
}
db.close();
}
/**
*实时更新线程的最后下载位置
*@parampath
*@parammap
*/
publicvoidupdate(Stringpath,Map<Integer,Integer>map){
SQLiteDatabasedb=openHelper.getWritableDatabase();
db.beginTransaction();
try{
for(Map.Entry<Integer,Integer>entry:map.entrySet()){
db.execSQL("updatefiledownsetposition=?wheredownpath=?andthreadid=?",
newObject[]{entry.getValue(),path,entry.getKey()});
}
db.setTransactionSuccessful();
}finally{
db.endTransaction();
}
db.close();
}
/**
*当文件下载完成后，清掉该文件对应的下载记录
*@parampath
*/
publicvoiddelete(Stringpath){
SQLiteDatabasedb=openHelper.getWritableDatabase();
db.execSQL("deletefromfiledownwheredownpath=?",newObject[]{path});
db.close();
}

}
```

（9）编写文件 DownloadThread.java，具体代码如下。

```
packagecom.changcheng.net.download;
importjava.io.InputStream;
importjava.io.RandomAccessFile;
importjava.net.HttpURLConnection;
importjava.net.URL;
importandroid.util.Log;
publicclassDownloadThreadextendsThread{
privatestaticfinalStringTAG="DownloadThread";
privateRandomAccessFilesaveFile;
privateURLdownUrl;
privateintblock;
/*下载开始位置*/
privateintthreadId=-1;
privateintstartPos;
privateintdownLength;
privatebooleanfinish=false;
privateFileDownloaderdownloader;
publicDownloadThread(FileDownloaderdownloader,URLdownUrl,RandomAccessFilesaveFile,intblock,i
ntstartPos,intthreadId){
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
publicvoidrun(){
if(downLength<block){//未下载完成
try{
HttpURLConnectionhttp=(HttpURLConnection)downUrl.openConnection();
http.setRequestMethod("GET");
http.setRequestProperty("Accept","image/gif,image/jpeg,image/pjpeg,image/pjpeg,application/x-shockw
ave-flash,application/xaml+xml,application/vnd.ms-xpsdocument,application/x-ms-xbap,application/x-m s-
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;MSIE8.0;WindowsNT5.2;Trident/4.0;.
NETCLR1.1.4322;.NETCLR2.0.50727;.NETCLR3.0.04506.30;.NETCLR3.0.4506.2152;.NETCLR3.5.30729)");
http.setRequestProperty("Connection","Keep-Alive");
InputStreaminStream=http.getInputStream();
```

```
intmax=block>1024?1024:(block>10?10:1);
byte[]buffer=newbyte[max];
intoffset=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);
intspare=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(Exceptione){
this.downLength=-1;
print("线程"+this.threadId+":"+e);
}
}
}
privatestaticvoidprint(Stringmsg){
Log.i(TAG,msg);
}
/**
*下载是否完成
*@return
*/
publicbooleanisFinish(){
returnfinish;
}
/**
*已经下载的内容大小
*@return 如果返回值为-1,代表下载失败
*/
publiclonggetDownLength(){
returndownLength;
}
}
```

（10）编写文件 DBOpenHelper.java，具体代码如下。

```
packagecom.changcheng.download.service;
importandroid.content.Context;
```

```
importandroid.database.sqlite.SQLiteDatabase;
importandroid.database.sqlite.SQLiteOpenHelper;
publicclassDBOpenHelperextendsSQLiteOpenHelper{
privatestaticfinalStringDBNAME="download.db";
privatestaticfinalintVERSION=2;
publicDBOpenHelper(Contextcontext){
super(context,DBNAME,null,VERSION);
}

@Override
publicvoidonCreate(SQLiteDatabasedb){
db.execSQL("CREATETABLEIFNOTEXISTSfiledown(idintegerprimarykeyautoincrement,downpathva
rchar(100),threadidINTEGER,positionINTEGER)");
}
@Override
publicvoidonUpgrade(SQLiteDatabasedb,intoldVersion,intnewVersion){
db.execSQL("DROPTABLEIFEXISTSfiledown");
onCreate(db);
}
}
```

13.4 判断当前网络中 GPRS 和 Wi-Fi 的状态

在 Android 开发过程中，特别是在开发和网络相关的一些应用时，很可能会用到网络连接状态，包括 GPRS、Wi-Fi 等。其实解决这些问题的方法很简单，Android 提供了两个类，一个是 ConnectivityManager，一个是 NetworkInfo，通过这两个类即可判断当前网络 GPRS 和 Wi-Fi 的状态。

13.4.1 ConnectivityManager 类和 NetworkInfo 类

在 Android 开发过程中，通过类 ConnectivityManager 可以实现管理和网络连接相关的操作，例如相关的 TelephonyManager 可以管理与手机、运营商等的相关信息，而 WifiManager 则管理与 Wi-Fi 相关的信息。

要想访问网络状态，首先得添加如下权限。

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

类 NetworkInfo 包含了对 Wi-Fi 和 Mobile 两种网络连接模式的详细描述，通过其 getState() 方法获取的 State 对象则代表着连接成功与否的状态。

例如下面的代码演示了这两个类的基本用法。

```
publicvoidtestConnectivityManager(){
ConnectivityManagerconnManager=(ConnectivityManager)this
.getSystemService(CONNECTIVITY_SERVICE);
//获取代表联网状态的 NetWorkInfo 对象
```

```
NetworkInfonetworkInfo=connManager.getActiveNetworkInfo();
//获取当前的网络连接是否可用
booleanavailable=networkInfo.isAvailable();
if(available){
Log.i("通知","当前的网络连接可用");
}
else{
Log.i("通知","当前的网络连接可用");
}
Statestate=connManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).getState();
if(State.CONNECTED==state){
Log.i("通知","GPRS 网络已连接");
}
state=connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI).getState();
if(State.CONNECTED==state){
Log.i("通知","Wi-Fi 网络已连接");
}
//跳转到无线网络设置界面
startActivity(newIntent(android.provider.Settings.ACTION_WIRELESS_SETTINGS));
//跳转到无限 Wi-Fi 网络设置界面
startActivity(newIntent(android.provider.Settings.ACTION_WIFI_SETTINGS));
}
```

了解了类 ConnectivityManager 和类 NetworkInfo 的基本用法后，就可以实现判断当前网络 GPRS 和 Wi-Fi 状态的方法了。例如通过下面的代码就可以实现。

```
importandroid.content.Context;
importandroid.net.ConnectivityManager;
importandroid.net.NetworkInfo;
publicclassTestextendsActivity{
privateConnectivityManagercm;
privateNetworkInfoinfo;
/**Calledwhentheactivityisfirstcreated.*/
publicvoidonCreate(BundlesavedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
cm=(ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);
if(cm.getNetworkInfo(1).getState()==NetworkInfo.State.CONNECTED)
{
text.setText("Wi-Fi 已经连接");
}
else{
text.setText("Wi-Fi 未连接");
}
if(cm.getNetworkInfo(0).getState()==NetworkInfo.State.CONNECTED)
{
text.setText("GPRS 已经连接");
```

```
}
else{
text.setText("GPRS 未连接");
}
}
}
```

通过下面的代码可以判断是否有网络连接。

```
public boolean isNetworkConnected(Context context) {
if (context != null) {
                ConnectivityManager mConnectivityManager = (ConnectivityManager) context
                        .getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo mNetworkInfo = mConnectivityManager.getActiveNetworkInfo();
if (mNetworkInfo != null) {
return mNetworkInfo.isAvailable();
                }
            }
return false;
        }
```

通过下面的代码可以判断 Wi-Fi 网络是否可用。

```
public boolean isWifiConnected(Context context) {
if (context != null) {
                ConnectivityManager mConnectivityManager = (ConnectivityManager) context
                        .getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo mWiFiNetworkInfo = mConnectivityManager
                        .getNetworkInfo(ConnectivityManager.TYPE_WIFI);
if (mWiFiNetworkInfo != null) {
return mWiFiNetworkInfo.isAvailable();
                }
            }
return false;
        }
```

通过下面的代码可以判断 Mobile 网络是否可用。

```
public boolean isMobileConnected(Context context) {
if (context != null) {
                ConnectivityManager mConnectivityManager = (ConnectivityManager) context
                        .getSystemService(Context.CONNECTIVITY_SERVICE);
                NetworkInfo mMobileNetworkInfo = mConnectivityManager
                        .getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
if (mMobileNetworkInfo != null) {
return mMobileNetworkInfo.isAvailable();
                }
            }
return false;
```

```
    }
```

通过下面的代码可以获取当前网络连接的类型信息。

```
public static int getConnectedType(Context context) {
if (context != null) {
        ConnectivityManager mConnectivityManager = (ConnectivityManager) context
                .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo mNetworkInfo = mConnectivityManager.getActiveNetworkInfo();
if (mNetworkInfo != null && mNetworkInfo.isAvailable()) {
return mNetworkInfo.getType();
        }
    }
return -1;
    }
```

13.4.2　在程序启动时对网络状态进行判断

在使用 Android 系统连接网络的时候，并不是每次都能连接到网络，最好在程序启动的时候对网络的状态进行判断，如果没有网络则及时提醒用户进行设置。要判断网络状态，首先需要有相应的权限，允许访问网络状态权限的代码如下。

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

具体的判断代码如下。

```
private boolean NetWorkStatus() {
boolean netSataus = false;
        ConnectivityManager cwjManager = (ConnectivityManager) getSystemService
(Context.CONNECTIVITY_SERVICE);
cwjManager.getActiveNetworkInfo();
if (cwjManager.getActiveNetworkInfo() != null) {
netSataus = cwjManager.getActiveNetworkInfo().isAvailable();
        }
if (netSataus) {
            Builder b = new AlertDialog.Builder(this).setTitle("没有可用的网络")
                .setMessage("是否对网络进行设置？");
            b.setPositiveButton("是", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int whichButton) {
                Intent mIntent = new Intent("/");
                ComponentName comp = new ComponentName(
                        "com.android.settings",
                        "com.android.settings.WirelessSettings");
mIntent.setComponent(comp);
mIntent.setAction("android.intent.action.VIEW");
                // 如果在设置完成后需要再次进行操作，可以重写操作代码，在这里
```

```
不再重写
            startActivityForResult(mIntent,0);
                        }
                    }).setNeutralButton("否", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
            dialog.cancel();
                        }
                    }).show();
                }
            return netSataus;
            }
```

13.5 开启或关闭 APN

虽然 Android 对于网络接入点（Access Point Name，APN）的 API 没有公开，但是参考源代码，然后进行数据库操作，这样系统会自动监听数据库的变化，从而实现开启或者关闭 APN。读者在获取 Android 的源码后，可以重点研究文件 frameworks/base/core/Java/android/provider/Telephony.java 中的类，此类的核心是 URI 和数据库字段“content://telephony/carriers”，这个字段可以在文件 Telephony.java 中找到。

下面的演示代码实现了如下两个功能。

（1）当开启 APN 的时候，设置一个正确的移动或者联通的 APN。

（2）关闭的时候设置一个错误 APN 就会自动关闭网络。

首先定义继承于 Activity 的类 Main，代码如下。

```
package cc.mdev.Demo;
import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
public class Main extends Activity {
    Uri uri = Uri.parse("content://telephony/carriers");
    @Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
    Button open= (Button) findViewById(R.id.open);
    Button close= (Button) findViewById(R.id.close);
open.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
    public void onClick(View v) {
    openAPN();
        }
        });
    close.setOnClickListener(new View.OnClickListener() {
        @Override
    public void onClick(View v) {

    closeAPN();
        }
        });
        }
    public void openAPN(){
        List list = getAPNList();
    for (APN apn : list) {
        ContentValues cv = new ContentValues();
    cv.put("apn", APNMatchTools.matchAPN(apn.apn));
    cv.put("type", APNMatchTools.matchAPN(apn.type));
    getContentResolver().update(uri, cv, "_id=?", new String[]{apn.id});
        }
        }
    public void closeAPN(){
        List list = getAPNList();
    for (APN apn : list) {
        ContentValues cv = new ContentValues();
    cv.put("apn", APNMatchTools.matchAPN(apn.apn)+"mdev");
    cv.put("type", APNMatchTools.matchAPN(apn.type)+"mdev");
    getContentResolver().update(uri, cv, "_id=?", new String[]{apn.id});
        }
        }
    private List getAPNList(){
        String tag = "Main.getAPNList()";
        //current 不为空表示可以使用的 APN
        String projection[] = {"_id,apn,type,current"};
        Cursor cr = this.getContentResolver().query(uri, projection, null, null, null);
        List list = new ArrayList();
    while(cr!=null && cr.moveToNext()){
        Log.d(tag, cr.getString(cr.getColumnIndex("_id")) + " " + cr.getString(cr.getColumnIndex("apn")) +
" " + cr.getString(cr.getColumnIndex("type"))+ " " + cr.getString(cr.getColumnIndex("current")));
        APN a = new APN();
        a.id = cr.getString(cr.getColumnIndex("_id"));
        a.apn = cr.getString(cr.getColumnIndex("apn"));
        a.type = cr.getString(cr.getColumnIndex("type"));
    list.add(a);
        }
```

```
if(cr!=null)
cr.close();
return list;
    }
public static class APN{
    String id;
    String apn;
    String type;
    }
    }
```

然后实现不同运营商的 APN，演示代码如下。

```
package cc.mdev.apn;
public final class APNMatchTools {
public static class APNNet{
    /*
    *  中国移动 cmwap
    */
public static String CMWAP = "cmwap";
    /**
    *  中国移动 cmnet
    */
public static String CMNET = "cmnet";
    //中国联通 3GWAP 设置中国联通 3G 因特网设置中国联通 WAP 设置中国联通因特网设置
    //3gwap 3gnet uniwap uninet
    /**
    * 3G wap  中国联通 3gwap APN
    */
public static String GWAP_3 = "3gwap";
    /**
    * 3G net  中国联通 3gnet APN
    */
public static String GNET_3="3gnet";
    /**
    * uni wap  中国联通 uni wap APN
    */
public static String UNIWAP="uniwap";
    /**
    * uni net  中国联通 uni net APN
    */
public static String UNINET="uninet";
    }
public static String matchAPN(String currentName) {
if("".equals(currentName) || null==currentName){
return "";
    }
```

```
currentName = currentName.toLowerCase();
if(currentName.startsWith(APNNet.CMNET))
return APNNet.CMNET;
else if(currentName.startsWith(APNNet.CMWAP))
return APNNet.CMWAP;
else if(currentName.startsWith(APNNet.GNET_3))
return APNNet.GNET_3;
else if(currentName.startsWith(APNNet.GWAP_3))
return APNNet.GWAP_3;
else if(currentName.startsWith(APNNet.UNINET))
return APNNet.UNINET;
else if(currentName.startsWith(APNNet.UNIWAP))
return APNNet.UNIWAP;
else if(currentName.startsWith("default"))
return "default";
else return "";
    }
    }
```

第 14 章　开发移动微博应用程序

微博是微博客（MicroBlog）的简称，是一个基于用户关系的信息分享、传播以及获取平台，用户可以通过 Web、WAP 以及各种客户端登录，以 140 字左右的文字更新显示信息，并实现即时分享。在本章的内容中，将详细介绍在 Android 系统中开发微博项目的基本知识。

14.1　微博介绍

在当前的互联网时代中，使用微博的用户越来越多，最早的微博是美国的 Twitter，根据相关公开数据，截至 2010 年 1 月份，该产品在全球已经拥有 7500 万注册用户。2009 年 8 月份中国最大的门户网站新浪网推出“新浪微博”内测版，成为门户网站中第一家提供微博服务的网站，从此微博正式进入人们视野。

（1）微博的特点

微博广泛分布在桌面、浏览器、移动终端等多个平台上，有多种商业模式并存，并有形成多个垂直细分领域的可能。但是无论哪种商业模式，都离不开用户体验的特性和基本功能。

（2）手机微博

微博的主要发展平台是手机，微博以计算机为服务器以手机为平台，把每个手机用户用无线的手机连在一起，让每个手机用户不再使用计算机就可以发布自己的最新信息，并和好友分享自己的快乐。

微博之所以要限定 140 个字符，就是源于从手机发短信最多的字符就是 140 个。由此可见，微博从诞生之初就同手机密不可分，这更是在互联网形态中最大的亮点。微博对互联网的重大意义就在于建立了手机和互联网应用的无缝连接，培养手机用户使用手机上网的习惯，增强移动端同互联网的互动，从而使手机用户顺利过渡到无线互联网用户。在目前应用中，手机和微博应用有如下三种结合形式。

- 通过短信和彩信

短、彩信形式是同移动运营商合作，用户所花的短、彩信费用由运营商收取，这种形式覆盖的人群比较广泛，只要能发短信就能更新微博。但对用户来说更新成本太大，并且彩信限制 50KB 大小的弊端严重影响了图片的清晰度。最关键的是这种方式只能提供更新，而无法看到其他人的更新，这种单向的信息传输方式大大降低了用户的参与感和互动性。

- 通过 WAP 版网站

各微博网站基本都有自己的 WAP 版，用户可以通过登录 WAP 或通过安装客户端连接到 WAP 版。这种形式只要手机能上网就能连接到微博，就可以更新也可以浏览、回复和评论，所需费用也就是浏览过程中所用的流量费。但目前国内的流量费还相对较高，网速也相对较慢，如果要上传大图片，速度会非常慢。

❑ 通过手机客户端

手机客户端分如下两种。

- 微博网站开发的基于 WAP 的快捷方式版。用户通过客户端直接连接到经过美化和优化的 WAP 版微博网站。这种形式的用户行为依赖于用户的主动性，也就是用户想更新和浏览微博的时候才会打开客户端，其实也就相当于在手机端增加了一个微博网站快捷方式。
- 利用微博网站提供的 API 开发的第三方客户端。国际上比较有名的是 Twitter 的客户端，国内比较著名的是新浪和 Hesine（和信）提供的客户端。其中 Gravity 是专门为 Twitter 开发的，需要通过主动联网登陆的，但操作架构和界面经过合理设计，用户体验非常好。和信是国内公司开发的，目前不但支持 Twitter，还支持国内的各主流微博。与其他客户端不同的是，和信的客户端是利用 IP Push 技术提供微博更新和下发通道，不但能够大大提升用户更新微博的速度，而且能将微博消息推送到用户的手机中，用户不用主动登陆微博就能实现浏览和互动。和信支持的系统平台比较多，但是缺点是在非智能机上的体验还不是很好。而新浪微博的 API 平台开放了包括微博、评论、用户及关系在内的二十余类接口，深受国内程序员的喜好。

14.2　微博开发必备技术介绍

在本节的内容中，将简单介绍在 Android 平台中开发微博系统所需要的基本技术，为读者步入本书后面知识的学习打下基础。

14.2.1　XML-RPC 技术

开发微博应用程序的关键技术是 RPC，RPC 是 Remote Procedure Call 的缩写，意为“远程过程调用”。XML-RPC 是一种统一标准的规范，是通过 HTTP 的方式连接运行的，以传送符合 XML-RPC 格式的请求来调用远程服务器上的某个程序，进而运行微博的功能。现在许多网络服务业者都会以 XML-RPC 的方式提供给软件开发者一个系统介接的管道，让开发者能够根据业者定义好的方式，以 XML-RPC 的方式来使用该网站的某些功能。当前许多市面中的微博也都支持 XML-RPC 的介接方式。

XML-RPC 的原理是 XML-RCP 工具把传入的参数组合成 XML，然后用通过 HTTP 协议发给服务器，服务器回复 XML 格式数据，再由专业工具解析给调用者。

在 XML-RPC 标准中，规定 XML 内容的规则如下。

```
<xml version=”1.0”?>
<methodCall>
   <methodName>要调用的 method name</methodName>
      <params>
      <params>参数 1</param>
      <params>参数 2</param>
      <param>参数 n</param>
      </params>
</methodCall>
```

Android 本身并不支持 XML-RPC 协议，需要下载相应的工具。读者可以从如下地址下载 XML-RPC 相关工具。

```
http://code.google.com/p/android-xmlrpc/downloads/list
```

下面的代码演示了使用 XML-RPC 协议实现微博客户端的基本过程。

```
package org.xmlrpc;
import java.net.URI;
import java.util.HashMap;
import java.util.Map;
import org.apache.http.conn.HttpHostConnectException;
import org.xmlrpc.android.XMLRPCClient;
import org.xmlrpc.android.XMLRPCException;
import org.xmlrpc.android.XMLRPCFault;
import org.xmlrpc.android.XMLRPCSerializable;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;
import android.widget.EditText;
import android.widget.Toast;
import android.widget.Button;
import android.content.DialogInterface.OnCancelListener;
import android.view.View.OnClickListener;
import android.view.View;
public class TestBlog extends Activity {
   private XMLRPCClient client;
   private URI uri;
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.test_blog);
        Button btn = (Button) findViewById(R.id.send);
        btn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                post();
            }
        });
   }

   void post() {
            String blogid = ((EditText) findViewById(R.id.blogid_edit)).getText()
                                .toString();              //ID,
            String username = ((EditText) findViewById(R.id.username_edit))
                                .getText().toString();    //用户名
            String password = ((EditText) findViewById(R.id.password_edit))
                                .getText().toString();    // 密码
```

```
                String title = ((EditText) findViewById(R.id.title_edit)).getText()
                                .toString();            //标题
                String content = ((EditText) findViewById(R.id.content_edit)).getText()
                                .toString();            // 正文
                uri = URI.create("http://blog.csdn.net/" + blogid
                                + "/services/metablogapi.aspx");
                client = new XMLRPCClient(uri);
                Map<String, Object> structx = new HashMap<String, Object>();
                structx.put("title", title);
                structx.put("description", content);
                Object[] params = new Object[] { blogid, username, password, structx,
                        true };

                try {
                        client.callEx("metaWeblog.newPost", params);
                        Toast.makeText(this, "OK", 10000).show();
                } catch (XMLRPCException e) {
                        Toast.makeText(this, "ERROR" + e, 10000).show();
                }
        }
}
```

14.2.2　Meta Weblog API 客户端

Meta Weblog API 是博客园发布的一款功能强大的客户端，其登录地址是：http://www.cnblogs.com/<用户名>/services/metaweblog.aspx。Meta Weblog API 支持通过 XML-RPC 的方法在软件中编辑及浏览日志，其中最为常用的 API 如下。

- 发布新文章(metaWeblog.newPost)。
- 获取分类(metaWeblog.getCategories)。
- 最新文章(metaWeblog.getRecentPosts)。
- 新建文章分类(wp.newCategory)。
- 上传图片音频或视频(metaWeblog.newMediaObject)。

14.3　分析腾讯 Android 版微博 API

作为一名 Android 学习者来说，独立开发微博系统的难度比较大。当前可以获取很多微博系统的开源代码，开发人员利用它们提供的 API 接口，就可以方便的开发出 Android 版的微博系统。比较有代表性的 Android 版的微博系统有新浪微博和腾讯微博。在本节的内容中，将首先讲解腾讯微博系统的 API 接口。

14.3.1　源码和 jar 包下载

因为当前腾讯微博提供的 Java(Android) SDK 功能过低，所以特意集成了一个 Java SDK 包，此包适用于 Android 系统。在包中包含了腾讯微博大部分的 API，主要特点如下。

- 用法简单：微博、评论、转发、私信同一个实体类。
- 方便扩展：可以根据需要修改源代码或是继承 QqTSdkService 类，为了能升级版本建议采用继承的方式。

在压缩包 Java SDK 中，QqTAndroidSdk-1.0.0.jar 是 SDK 的主代码，其中 QqTSdk-ServiceImpl 包含了所有接口的实现。各个包的具体说明如下。

- QqTAndroidSdk-1.0.0.jar：这是主程序代码，可以直接解压缩查看源码。
- google code 源码的下载地址：http://code.google.com/p/qq-t-java-sdk/source/browse/。
- github 源码的下载地址：https://github.com/Trinea/qq-t-java-sdk。
- 压缩包 JavaCommon-1.0.0.jar：是 QqTAndroidSdk 依赖的公用处理包，包含了字符串、list、数组、map、json 工具类等。

14.3.2 具体使用

在具体使用之前，先参考腾讯微博的 API 文档说明，地址是：http://wiki.open.qq.com/wiki/website/API%E5%88%97%E8%A1%A8，如图 14-1 所示。

图 14-1　腾讯微博的 API 文档页面

在编码使用 API 时，都需要先新建 QqTSdkService 类对象并进行初始化工作。例如下面的初始化代码。

```
/**
 * 分别设置应用的 key、secret(腾讯提供)。用户的 accesstoken 和 tokenSecret(OAuth 获取)
 * 请用自己的相应字符串替换，否则无法成功发送和获取数据
 **/
QqTAppAndToken qqTAppAndToken = new QqTAppAndToken();
```

```
qqTAppAndToken.setAppKey("***"); // ***用应用 key 替换
qqTAppAndToken.setAppSecret("***"); // ***用应用 secret 替换
qqTAppAndToken.setAccessToken("***"); // ***用用户 accesstoken 替换
qqTAppAndToken.setTokenSecret("***"); // ***用用户 tokenSecret 替换

/** 新建 QqTSdkService 对象，并设置应用信息和用户访问信息 **/
QqTSdkService qqTSdkService = new QqTSdkServiceImpl();
qqTSdkService.setQqTAppAndToken(qqTAppAndToken);
```

接下来开始对接口进行详细介绍，并讲解使用 QqTAndroidSdk-1.0.0.jar 中的 API 的方法。腾讯微博中的接口主要分成下面的几大类。

1. 时间线（微博列表）

这里的 20 个接口包含了腾讯微博的如下四个部分的 API

（1）时间线中，除 statuses/ht_timeline_ext（话题时间线）以外的 15 个 API。

（2）私信相关中，收件箱、发件箱的两个 API。

（3）数据收藏中，收藏的微博列表和获取已订阅话题列表的两个 API。

（4）微博相关中，获取单条微博的转发或点评列表的 API。

以获取首页信息为例，示例代码如下。

```
QqTTimelinePara qqTTimelinePara = new QqTTimelinePara();
/** 设置分页标识 **/
qqTTimelinePara.setPageFlag(0);
/** 设置起始时间 **/
qqTTimelinePara.setPageTime(0);
/** 每次请求记录的条数 **/
qqTTimelinePara.setPageReqNum(QqTConstant.VALUE_PAGE_REQ_NUM);
/** 可以设置拉取类型，可取值 QqTConstant 中 VALUE_STATUS_TYPE_TL_… **/
qqTTimelinePara.setStatusType(QqTConstant.VALUE_STATUS_TYPE_TL_ALL);
/** 可以设置微博内容类型，可取值 QqTConstant 中 VALUE_CONTENT_TYPE_TL… **/
qqTTimelinePara.setContentType(QqTConstant.VALUE_CONTENT_TYPE_TL_ALL);
List<QqTStatus> qqTStatusList = qqTSdkService.getHomeTL(qqTTimelinePara);
assertTrue(qqTStatusList != null);
```

这样 qqTStatusList 就保存了首页的 20 条数据，可以自行设置不同的类型参数。如果想获取更多时间线数据，请读者参考腾讯微博 Java(Android) SDK 时间线 API 的详细介绍。

2. 微博新增 API

在本书成稿时，腾讯微博新增加了 8 个 API，分别是在微博相关中的发表一条微博、转播一条微博、回复一条微博、发表一条带图片微博、点评一条微博、发表音乐微博、发表视频微博、发表心情帖子。在新的 API 中发表一条微博和发表一条带图片微博的操作合二为一。

另外，还新增了私信相关中的发私信操作一条微博的功能，以新增一条微博为例，演示代码如下。

```
qqTSdkService.addStatus("第一条状态哦", null);
```

其中第一个参数为状态内容，第二个参数为图片地址，不传图片为空即可。或者在如下复杂代码中，status 可以设置其他地理位置信息。

```
QqTStatusInfoPara status = new QqTStatusInfoPara();
status.setStatusContent("发表一条带图片微博啦");
/** 发表带图微博，设置图片路径 **/
status.setImageFilePath("/mnt/sdcard/DCIM/Camera/IMAG2150.jpg");
assertTrue(qqTSdkService.addStatus(status, qqTAppAndToken));
```

3. 操作一条微博

（1）在微博相关中，删除一条微博的 API。

（2）在私信相关中，删除私信的 API

（3）在数据收藏中，收藏微博、取消收藏微博、订阅话题、取消订阅话题的 4 个 API。以收藏一条微博为例，示例代码如下：

```
qqTSdkService.collect(12121);
```

其中参数为微博 id。

4. 关系链列表（用户列表）

这 10 个接口包含腾讯微博关系链相关中的互听关系链列表（对某个用户而言，既是他的听众又被他收听）、其他帐号听众列表、其他帐号收听的人列表、其他帐户特别收听的人列表、黑名单列表、我的听众列表、我的听众列表（只包含名字）、我收听的人列表、我收听的人列表（只包含名字）、我的特别收听列表。

以获取自己的收听用户为例，示例代码如下。

```
QqTUserRelationPara qqTUserRelationPara = new QqTUserRelationPara();
qqTUserRelationPara.setReqNumber(QqTConstant.VALUE_PAGE_REQ_NUM);
qqTUserRelationPara.setStartIndex(0);
List<QqTUser> qqTUserList = qqTSdkService.getSelfInterested(qqTUserRelationPara);
```

5. 用户建立关系

这 6 个接口包含腾讯微博关系链相关中的收听某个用户、取消收听某个用户、特别收听某个用户、取消特别收听某个用户、添加某个用户到黑名单、从黑名单中删除某个用户。

以关注某些用户为例，示例代码如下。

```
qqTSdkService.interestedInOther("wenzhang,li_nian,mayili007", null)
```

6. 帐户相关

这 7 个接口包含腾讯微博帐户相关中的获取自己的详细资料、更新用户信息、更新用户头像信息、更新用户教育信息、获取其他人资料、获取一批人的简单资料、验证账户是否合法（是否注册微博）。

以获取自己的资料为例，示例代码如下。

```
QqTUser qqTUser = qqTSdkService.getSelfInfo();
```

7. 搜索相关

这 3 个接口包含腾讯微博搜索相关中的搜索用户、搜索微博、通过标签搜索用户。以搜索微博为例，示例代码如下。

```
public void testSearchStatus() {
    QqTSearchPara qqTSearchPara = new QqTSearchPara();
    qqTSearchPara.setKeyword("iphone");
    qqTSearchPara.setPage(1);
    qqTSearchPara.setPageSize(QqTConstant.VALUE_PAGE_REQ_NUM);
    List<QqTStatus> qqTStatusList = qqTSdkService.searchStatus(qqTSearchPara);
    assertTrue(qqTStatusList != null);
}
```

8. 热度趋势相关

这两个接口包含腾讯微博热度趋势中的话题热榜、转播热榜用户。以话题热榜为例，示例代码如下。

```
public void testGetHotTopics() {
    QqTHotStatusPara qqTHotStatusPara = new QqTHotStatusPara();
    qqTHotStatusPara.setReqNum(QqTConstant.VALUE_PAGE_REQ_NUM);
    qqTHotStatusPara.setLastPosition(0);
    /**
     * 1 话题名，2 搜索关键字 3 两种类型都有
     **/
    qqTHotStatusPara.setType(Integer.toString(1));
    List<QqTTopicSimple> hotTopicsList = qqTSdkService.getHotTopics(qqTHotStatusPara);
    assertTrue(hotTopicsList != null);
}
```

9. 数据更新

这一个接口为腾讯微博数据更新相关中的查看数据更新条数，示例代码如下。

```
public void testGetUpdateInfoNum() {
    /** 设置 clearType，对应 QqTConstant.VALUE_CLEAR_TYPE_… **/
    QqTUpdateNumInfo qqTUpdateNumInfo = qqTSdkService.getUpdateInfoNum(true, QqTConstant.
VALUE_CLEAR_TYPE_HOME_PAGE);
    assertTrue(qqTUpdateNumInfo != null);
}
```

10. 发起话题

这两个接口是为腾讯微博中的话题应用服务的，可以根据话题名称查询话题 ID 和根据话题 ID 获取话题相关信息。示例代码如下。

```
public void testGetTopicInfoByIds() {
    /** 先得到话题 id **/
    Map<String, String> topicIdAndName = qqTSdkService.getTopicIdByNames("袁莉闪婚,美汁源
```

```
下架,iphone");
                if (topicIdAndName != null) {
                    /** 话题 id 列表，以逗号分隔 **/
                    List<QqTStatus> qqtStatusList = qqTSdkService.getTopicInfoByIds(ListUtils.join(new
ArrayList<String>(topicIdAndName.keySet())));
                    assertTrue(qqtStatusList != null);
                } else {
                    assertTrue(false);
                }
            }
```

11. 标签相关

这两个接口为腾讯微博标签相关中的添加标签和删除标签 API，示例代码如下。

```
        public void testDeleteTag() {
            /** 删除自己的 tag，先获取自己的资料，从中取中 tag id **/
            QqTUser qqTUser = qqTSdkService.getSelfInfo();
            if (qqTUser != null && qqTUser.getTagMap() != null && qqTUser.getTagMap().size() > 0) {
                /** 删除 tag **/
                for (Map.Entry<String, String> tag : qqTUser.getTagMap().entrySet()) {
                    qqTSdkService.deleteTag(tag.getKey());
                }
            } else {
                assertTrue(false);
            }
        }
```

14.4 详解 Android 版新浪微博

新浪微博在国内较早地推出微博应用。读者可以登录 http://open.weibo.com/wiki/SDK 获取在 Android 平台使用新浪微博的详细资料，在网站中也可以获取开源代码。

在 Android 中使用新浪微博开发平台 API 的基本步骤如下。

1. 通过官方网址下载 SDK

当前的最普遍版本是 3.1.1，下载页面的地址是：

http://open.weibo.com/wiki/SDK#AndroidSDK

此页面的界面效果如图 14-2 所示。

2. 认证

在 SDK 中有完整的如何通过 oauth 认证的演示实例，认证和使用流程如下。

（1）在/weibo4android/src/weibo4android/Weibo.java 设置 App Key 和 App Secret（在官方网站新建应用可获得）。

```
        public static String CONSUMER_KEY = "2664209963";
        public static String CONSUMER_SECRET = "b428615797a5d676d428cd146c040399";
```

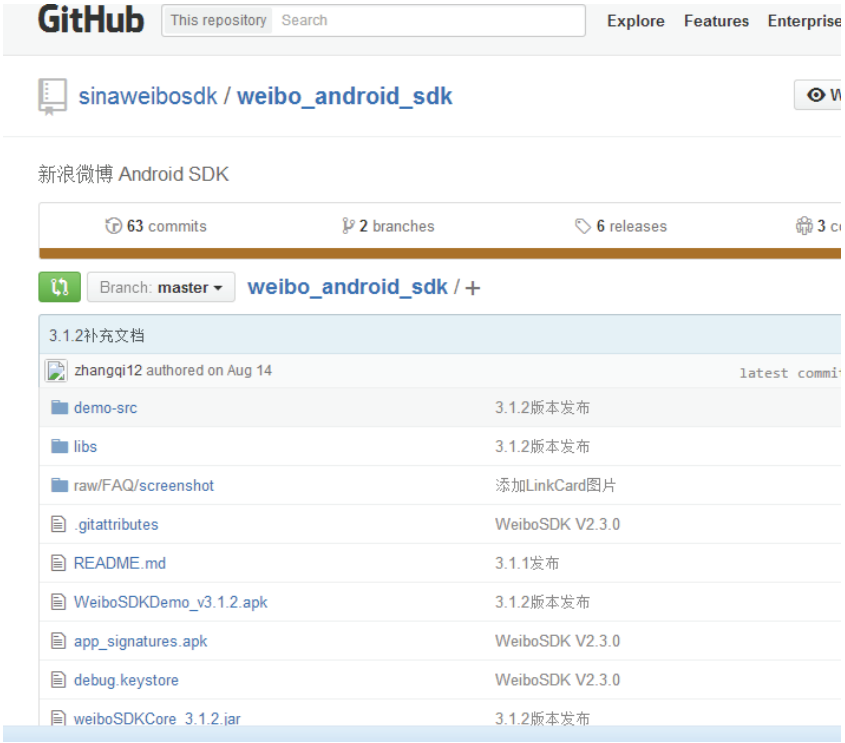


图 14-2　下载 SDK 页面

（2）在/weibo4android/examples/weibo4android/androidexamples/AndroidExample.java 中，将 App Key 和 App Secret 设置到系统类中。

```
System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
```

（3）通过 HTTP POST 方式向服务提供方请求获得 RequestToken。

```
RequestToken    requestToken =weibo.getOAuthRequestToken("weibo4android://OAuthActivity");
```

（4）将用户引导至授权页面。

```
Uri uri = Uri.parse(requestToken.getAuthenticationURL()+ "&display=mobile");
startActivity(new Intent(Intent.ACTION_VIEW, uri));
```

（5）授权页面要求用户输入用户名和密码，授权完成后，服务提供方会通过回调 URL 将用户引导回客户端的 OAuthActivity 页面。

```
<activity android:name=".OAuthActivity">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="weibo4android" android:host="OAuthActivity" />
```

```
        </intent-filter>
            </activity>
```

（6）客户端根据临时令牌和用户授权码从服务提供方那里获取访问令牌 (Access Token)。

```
Uri uri=this.getIntent().getData();
RequestToken requestToken= OAuthConstant.getInstance().getRequestToken();
AccessToken    accessToken=requestToken.getAccessToken(uri.getQueryParameter("oauth_verifier"));
```

（7）获得访问令牌后便可使用 API 接口获得和操作用户数据。

```
Weibo weibo=OAuthConstant.getInstance().getWeibo();
weibo.setToken(OAuthConstant.getInstance().getToken(), OAuthConstant.getInstance().getTokenSecret());
String[] args = new String[2];
args[0]=OAuthConstant.getInstance().getToken();
args[1]=OAuthConstant.getInstance().getTokenSecret();
try {
        GetFollowers.main(args);//返回用户关注对象列表，并返回最新微博文章
} catch (Exception e) {
        e.printStackTrace();
}
```

在上述步骤中，weibo4android 是 XML 文件中定义的索引名，在上面步骤（5）中的 XML 代码中，<data android:scheme="weibo4android" android:host="OAuthActivity" />部分的索引名是自定义的，与 Java 代码中的 URL 相匹配即可

在本书接下来的内容中，将不再剖析 Android 版新浪微博的实现源码，而是以此为基础，讲解二次扩展开发的基本知识。

14.4.1 新浪微博图片缩放的开发实例

在 Android 开发过程中，有时会用到图片缩放效果，即单击图片时显示缩放按钮，一定时间后自动消失。接下来将根据新浪微博的图片缩放原理编写演示代码。

（1）UI 布局文件的演示代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android：orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/layout1"
                >
        <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                android:id="@+id/rl"
                    >
            <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
```

```
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_weight="19"
                android:scrollbars="none"
                android:fadingEdge="vertical"
                android:layout_gravity="center"
                android:gravity="center"
            >
        <HorizontalScrollView
                android:layout_height="fill_parent"
                android:layout_width="fill_parent"
                android:scrollbars="none"
                 android:layout_gravity="center"
                android:gravity="center"
                android:id="@+id/hs"
                    >
                <LinearLayout
                        android：orientation="horizontal"
                        android:layout_width="fill_parent"
                        android:layout_height="fill_parent"
                        android:id="@+id/layoutImage"
                        android:layout_gravity="center"
                        android:gravity="center"
                        >
                        <ImageView
                            android:layout_gravity="center"
                            android:gravity="center"
                                android:id="@+id/myImageView"
                                android:layout_width="fill_parent"
                                android:layout_height="fill_parent"
                                android:layout_weight="19"
                                android:paddingTop="5dip"
                                android:paddingBottom="5dip"

                                />
                </LinearLayout>
        </HorizontalScrollView >
        </ScrollView>

            <ZoomControls android:id="@+id/zoomcontrol"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
            android:layout_alignParentBottom="true"
    >
</ZoomControls>
        </RelativeLayout>
```

```
</FrameLayout>
```

（2）用 Java 编写主程序代码，代码如下。

```
package com.Johnson.image.zoom;
import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnKeyListener;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.os.Bundle;
import android.os.Handler;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.ZoomControls;
public class MainActivity extends Activity {
        /** Called when the activity is first created. */
    private final int LOADING_IMAGE = 1;
    public static String KEY_IMAGEURI = "ImageUri";
    private ZoomControls zoom;
    private ImageView mImageView;
    private LinearLayout layoutImage;
    private int displayWidth;
    private int displayHeight;
    /**图片资源*/
    private Bitmap bmp;
    /**宽的缩放比例*/
    private float scaleWidth = 1;
    /**高的缩放比例*/
    private float scaleHeight = 1;
    /**用来计数放大+1      缩小-1*/
    private int      zoomNumber=0;
    /**单击屏幕显示缩放按钮，三秒消失*/
    private int showTime=3000;
```

```
    RelativeLayout rl;
    Handler mHandler = new Handler();
    private Runnable task = new Runnable() {
        public void run() {
            zoom.setVisibility(View.INVISIBLE);
        }
    };
            @Override
            public void onCreate(Bundle savedInstanceState) {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.main);
        //showDialog(LOADING_IMAGE);
        //图片是从网络上获取的话，需要加入滚动条
            bmp=BitmapFactory.decodeResource(getResources(), R.drawable.image);
        //removeDialog(LOADING_IMAGE);
             initZoom();
}
    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case LOADING_IMAGE: {
            final ProgressDialog dialog = new ProgressDialog(this);
            dialog.setOnKeyListener(new OnKeyListener() {
                @Override
                public boolean onKey(DialogInterface dialog, int keyCode,
                        KeyEvent event) {
                    if (keyCode == KeyEvent.KEYCODE_BACK) {
                        finish();
                    }
                    return false;
                }
            });
            dialog.setMessage("正在加载图片请稍后...");
            dialog.setIndeterminate(true);
            dialog.setCancelable(true);
            return dialog;
        }
        }
        return null;
    }
    public void initZoom() {
        /* 取得屏幕分辨率大小 */
        DisplayMetrics dm = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(dm);
```

```
displayWidth = dm.widthPixels;
displayHeight = dm.heightPixels;
mImageView = (ImageView) findViewById(R.id.myImageView);
mImageView.setImageBitmap(bmp);
layoutImage = (LinearLayout) findViewById(R.id.layoutImage);
mImageView.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {
      // TODO Auto-generated method stub
                                    /**
                                     * 在图片上和整个 view 上同时添加单击监听捕捉屏幕
                                     * 单击事件，来显示放大缩小按钮
                                     * */
      zoom.setVisibility(View.VISIBLE);
      mHandler.removeCallbacks(task);
      mHandler.postDelayed(task, showTime);
   }
});
layoutImage.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {
      // TODO Auto-generated method stub
      zoom.setVisibility(View.VISIBLE);
      mHandler.removeCallbacks(task);
      mHandler.postDelayed(task, showTime);
   }
});
zoom = (ZoomControls) findViewById(R.id.zoomcontrol);
zoom.setIsZoomInEnabled(true);
zoom.setIsZoomOutEnabled(true);
// 图片放大
zoom.setOnZoomInClickListener(new OnClickListener() {
   public void onClick(View v) {
      big();
   }
});
// 图片减小
zoom.setOnZoomOutClickListener(new OnClickListener() {
   public void onClick(View v) {
      small();
   }
});
zoom.setVisibility(View.VISIBLE);
mHandler.postDelayed(task, showTime);
```

```
}
@Override
public boolean onTouchEvent(MotionEvent event) {
    // TODO Auto-generated method stub
        /**
                * 在图片上和整个 view 上同时添加单击监听捕捉屏幕
                * 单击事件，来显示放大缩小按钮
                * */
    zoom.setVisibility(View.VISIBLE);
    mHandler.removeCallbacks(task);
    mHandler.postDelayed(task, showTime);
    return false;
}
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // TODO Auto-generated method stub
    super.onKeyDown(keyCode, event);

    return true;
}
/* 图片缩小的 method */
private void small() {
    --zoomNumber;
    int bmpWidth = bmp.getWidth();
    int bmpHeight = bmp.getHeight();

    Log.i("","bmpWidth = " + bmpWidth + ", bmpHeight = " + bmpHeight);
    /* 设置图片缩小的比例 */
    double scale = 0.8;
    /* 计算出这次要缩小的比例 */
    scaleWidth = (float) (scaleWidth * scale);
    scaleHeight = (float) (scaleHeight * scale);
    /* 产生 reSize 后的 Bitmap 对象 */
    Matrix matrix = new Matrix();
    matrix.postScale(scaleWidth, scaleHeight);
    Bitmap resizeBmp = Bitmap.createBitmap(bmp, 0, 0, bmpWidth, bmpHeight,
        matrix, true);
    mImageView.setImageBitmap(resizeBmp);
    /* 限制缩小尺寸 */
    if ((scaleWidth * scale * bmpWidth < bmpWidth / 4
        || scaleHeight * scale * bmpHeight > bmpWidth /4
        || scaleWidth * scale * bmpWidth > displayWidth / 5
        || scaleHeight * scale * bmpHeight > displayHeight / 5)&&(zoomNumber==-1) ){
```

```
            zoom.setIsZoomOutEnabled(false);
            } else {
            zoom.setIsZoomOutEnabled(true);
            }
            zoom.setIsZoomInEnabled(true);
            System.gc();
        }
        /* 图片放大的 method */
        private void big() {
            ++zoomNumber;
            int bmpWidth = bmp.getWidth();
            int bmpHeight = bmp.getHeight();
            /* 设置图片放大的比例 */
            double scale = 1.25;
            /* 计算这次要放大的比例 */
            scaleWidth = (float) (scaleWidth * scale);
            scaleHeight = (float) (scaleHeight * scale);
            /* 产生 reSize 后的 Bitmap 对象 */
            Matrix matrix = new Matrix();
            matrix.postScale(scaleWidth, scaleHeight);
            Bitmap resizeBmp = Bitmap.createBitmap(bmp, 0, 0, bmpWidth, bmpHeight,
                    matrix, true);
            mImageView.setImageBitmap(resizeBmp);
            /* 限制放大尺寸 */
            if (scaleWidth * scale * bmpWidth > bmpWidth * 4
                    || scaleHeight * scale * bmpHeight > bmpWidth * 4
                    || scaleWidth * scale * bmpWidth > displayWidth * 5
                    || scaleHeight * scale * bmpHeight > displayHeight * 5) {
                zoom.setIsZoomInEnabled(false);
            } else {
                zoom.setIsZoomInEnabled(true);
            }
            zoom.setIsZoomOutEnabled(true);
        System.gc();
        }
    }
```

14.4.2 添加分享到新浪微博

在过去浏览论坛或者博客的时候，每一个论坛/博客都需要一个专用帐号，但是现在会发现很多网站都有一个“用新浪微博登陆”，“用 QQ 帐号登录”之类的字样。这样经过授权以后就可以用新浪或这腾讯的帐号登录到不同论坛或者博客了，这确实是挺方便的事情，可以直接为社区带来不少的用户流量。

Android 系统有内置的分享功能，但是内置的分享功能只有在安装该应用的时候才会被显

示在列表中，下面的是 Android 系统内置的分享功能，如图 14-3 所示。

图 14-3　Android 系统内置的分享功能

选中图 14-4 中的“分享”选项后可以看到“新浪微博”，这个是编者自己添加的。这个列表的加载代码如下。

```
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        ShareAdapter mAdapter = new ShareAdapter(mContext, intent);
        //对话框的适配器
        public class ShareAdapter extends BaseAdapter {
            private final static String PACKAGENAME = "com.sina.weibo";
            private Context mContext;
            private PackageManager mPackageManager;
            private Intent mIntent;
            private LayoutInflater mInflater;
            private List<ResolveInfo> mList;
            private List<DisplayResolveInfo> mDisplayResolveInfoList;
            public ShareAdapter(Context context, Intent intent) {
                mContext = context;
                mPackageManager = mContext.getPackageManager();
                mIntent = new Intent(intent);
                mInflater =
(LayoutInflater)mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                mList = mContext.getPackageManager().queryIntentActivities(intent,
                        PackageManager.MATCH_DEFAULT_ONLY);
                // 排序
                ResolveInfo.DisplayNameComparator comparator = new
ResolveInfo.DisplayNameComparator(
                        mPackageManager);
                Collections.sort(mList, comparator);
                mDisplayResolveInfoList = new ArrayList<DisplayResolveInfo>();
                if (mList == null || mList.isEmpty()) {
                    mList = new ArrayList<ResolveInfo>();
                }
                final int N = mList.size();
```

```
            for (int i = 0; i < N; i++) {
                ResolveInfo ri = mList.get(i);
                CharSequence label = ri.loadLabel(mPackageManager);
                DisplayResolveInfo d = new DisplayResolveInfo(ri, null, null, label, null);
                mDisplayResolveInfoList.add(d);
            }
            //考虑是否已安装新浪微博，如果没有则自行添加
            if(!isInstallApplication(mContext, PACKAGENAME)){
                Intent i = new Intent(mContext, ShareActivity.class);
                Drawable d = mContext.getResources().getDrawable(R.drawable.sina);
                CharSequence label = mContext.getString(R.string.about_sina_weibo);
                DisplayResolveInfo dr = new DisplayResolveInfo(null, i, null, label, d);
                mDisplayResolveInfoList.add(0, dr);
            }
        }
        @Override
        public int getCount() {
            return mDisplayResolveInfoList.size();
        }
        @Override
        public Object getItem(int position) {
            return mDisplayResolveInfoList.get(position);
        }
        @Override
        public long getItemId(int position) {
            return position;
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            View item;
            if(convertView == null) {
                item = mInflater.inflate(R.layout.share_item, null);
            } else {
                item = convertView;
            }
            DisplayResolveInfo info = mDisplayResolveInfoList.get(position);
            ImageView i = (ImageView) item.findViewById(R.id.share_item_icon);
            if(info.mDrawable == null){
                i.setImageDrawable(info.mResoleInfo.loadIcon(mPackageManager));
            }else{
                i.setImageDrawable(info.mDrawable);
            }
            TextView t = (TextView) item.findViewById(R.id.share_item_text);
            t.setText(info.mLabel);
            return item;
        }
```

```
        public ResolveInfo getResolveInfo(int index){
            if(mDisplayResolveInfoList == null){
                return null;
            }
            DisplayResolveInfo d = mDisplayResolveInfoList.get(index);
            if(d.mResoleInfo == null){
                return null;
            }
            return d.mResoleInfo;
        }
        //返回跳转 intent
        public Intent getIntentForPosition(int index) {
            if(mDisplayResolveInfoList == null){
                return null;
            }
            DisplayResolveInfo d = mDisplayResolveInfoList.get(index);
            Intent i = new Intent(d.mIntent == null ? mIntent : d.mIntent);
            i.addFlags(Intent.FLAG_ACTIVITY_FORWARD_RESULT |
Intent.FLAG_ACTIVITY_PREVIOUS_IS_TOP);
            if(d.mResoleInfo != null){
                ActivityInfo a = d.mResoleInfo.activityInfo;
                i.setComponent(new ComponentName(a.applicationInfo.packageName, a.name));
            }
            return i;
        }
        //检查是否安装该 app
        boolean isInstallApplication(Context context, String packageName){
            try {
                mPackageManager
                        .getApplicationInfo(packageName,
PackageManager.GET_UNINSTALLED_PACKAGES);
                return true;
            } catch (NameNotFoundException e) {
                return false;
            }

        }
        /**
         * 打包数据 vo
         * @author Administrator
         */
        class DisplayResolveInfo {
            private Intent mIntent;
            private ResolveInfo mResoleInfo;
            private CharSequence mLabel;
            private Drawable mDrawable;
```

```
                DisplayResolveInfo(ResolveInfo resolveInfo, Intent intent,
                        CharSequence info, CharSequence label, Drawable d) {
                    this.mIntent = intent;
                    this.mResoleInfo = resolveInfo;
                    this.mLabel = label;
                    this.mDrawable = d;
                }
            }
        }
```

以上就加载了弹出框的数据适配器，如果系统已经安装则直接读取系统的分享，没有则添加。当单击分享微博的时候需要一系列的验证和授权，此处采用的机制是先获取requestToken，然后通过 requestToken 获取 AccessToken，然后才可以分享微博。

接下来在单击“分享到微博”的时候进行用户认证，这里的认证是读取新浪提供的页面，下面是部分代码。

```
Weibo weibo = new Weibo();
RequestToken requestToken = weibo.getOAuthRequestToken("yunmai://ShareActivity");
Log.i(TAG, "token:" + requestToken.getToken() + ",tokenSecret:" + requestToken.getTokenSecret());
OAuthConstant.getInstance().setRequestToken(requestToken);
Uri uri = Uri.parse(requestToken.getAuthenticationURL() + "&display=mobile");
url = uri.toString();
```

上面的地址 url 就是请求新浪时提供的登录界面的地址了，此时会涉及到 WebView 的使用。在 Android 里面 WebView 其实就是一个小型浏览器，功能很强大，强大到可以执行脚本。有了地址即可以通过 webview.loadurl(url)请求登录界面。也许读者可能会想自己设计一个登录界面，但是新浪官方说明：通过 getXauthAccessToken 方式认证可以自行设计登录界面，其他认证方式是不能够自行设计的。单击“授权”按钮时需要跳转到自己的 Activity 中，此处的配置是需要在 androidmanifest.xml 中配置的，例如设置跳转到 shareactivity.java。

```
<activity
            android:name="cn.yunmai.cclauncher.ShareActivity"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:host="ShareActivity" android:scheme="yunmai" />
            </intent-filter>
        </activity>
```

在此需要注意的是，在<data>标签中的内容需要和显示授权窗口中的 weibo.getOAuthRequestToken("yunmai://ShareActivity")相对应。当授权完成之后就会跳转到自己定义的 Activity，授权完成之后就会将微博发送到跳转之后的 Activity。在单击发送按钮的时，先获取刚才授权成功的 RequestToken，然后再获取 accessToken，最后发送微博。在此需要注意的

是，RequestToken 只需获取一次即可，获取的口令被保存为 accessToken，以后使用时只需直接调用 accessToken。单击发送按钮的操作代码如下。

```
        Uri uri = this.getIntent().getData();
        RequestToken requestToken = OAuthConstant.getInstance().getRequestToken();
        AccessToken accessToken = requestToken.getAccessToken(uri.getQueryParameter("oauth_verifier"));
        saveAccessToken(accessToken);//保存 accessToken
        Log.i(TAG, "oauth_verifier:" + uri.getQueryParameter("oauth_verifier") +
                                    ",Token" + accessToken.getToken() + ",TokenSecret:" + accessToken.
getTokenSecret());
        OAuthConstant.getInstance().setAccessToken(accessToken);
```

下面代码所执行的操作是获取授权之后的 accessToken，然后发送微博。

```
        Weibo weibo = OAuthConstant.getInstance().getWeibo();
        weibo.setToken(OAuthConstant.getInstance().getToken(), OAuthConstant.getInstance().getTokenSecret());
        Status s = weibo.updateStatus(mEdit.getText().toString());
```

Status 返回了一些详细的信息，有发送时间和用户 ID 等，这就样就完成了分享功能。

14.4.3　通过 JSON 对象获取登录新浪微博

可以引用新浪开发包中的各种类，在 Android 中通过 JSON 对象的方式登录新浪微博。在下面的代码中，1 代表登陆成功，0 代表登陆失败。并通过 verifyCredentials()方法请求新浪微博服务器返回 JSON 对象。

```
        package com.sfc.ui;
        import java.util.ArrayList;
        import java.util.List;
        import com.sfc.ui.adapter.LoginListAdapter;
        import weibo4j.User;                                    //这是新浪开发包中的实体类
        import weibo4j.Weibo;                                   //这是新浪开发包中的类
        import weibo4j.WeiboException;                          //这是新浪开发包中的类
        import android.app.Activity;
        import android.app.AlertDialog;
        import android.app.ProgressDialog;
        import android.os.Bundle;
        import android.os.Handler;
        import android.os.Message;
        import android.util.Log;
        import android.view.View;
        import android.view.View.OnClickListener;
        import android.widget.Button;
        import android.widget.ListView;
        import android.widget.Toast;
        public class LoginActivity extends Activity implements Runnable {
          private Button loginButton;
          private ListView listView;
          private ProgressDialog loginDialog;
```

```
private Thread loginThread;
private Handler handler;
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.login);
 loginButton = (Button)findViewById(R.id.loginButton);
 List<String> list = new ArrayList<String>();
 list.add("随便看看");
 list.add("推荐用户");
 list.add("热门转发");
 listView = (ListView)findViewById(R.id.listView);
 loginThread = new Thread(this);

 handler = new Handler(){
          //1 代表登陆成功  0 代表登陆失败
        public void handleMessage(Message msg) {
         loginDialog.cancel();
            switch (msg.what) {
           case 1:
           Toast.makeText(LoginActivity.this, "登陆成功  ", 3000).show();
                break;
           case 0:
            Toast.makeText(LoginActivity.this, "登陆失败", 3000).show();
            break;
           }
        };
 };
 listView.setAdapter(new LoginListAdapter(this,list));
 loginButton.setOnClickListener(new OnClickListener(){
  public void onClick(View v) {
   loginDialog = new ProgressDialog(LoginActivity.this);
   loginDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
   loginDialog.setMessage("登陆服务器");
   loginDialog.show();
   loginThread.start();
  }
 });
}
public void run() {
 Log.e("loginThread","start");
 Weibo weibo = new Weibo("XXX@sina.com","XXX"); //新浪微博用户名和密码
 weibo.setHttpConnectionTimeout(5000);
 Message    msa = new Message();
 try {
  User user = weibo.verifyCredentials();              //该方法会请求新浪微博服务器返回 JSON 对象
  msa.what=1;
 } catch (WeiboException e) {
```

```
            msa.what=0;
          }
        }
      }
```

14.4.4　实现 OAuth 认证

OAuth 协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同，OAuth 授权不会使第三方触及用户的帐号信息（如用户名与密码），即第三方无需使用用户的用户名与密码就可以申请获得该用户资源的授权，因此 OAuth 是安全的。

新浪微博为了实现自身的安全性，采用了 Oauth 协议认证方式。虽然下面的一段代码比较简单，但是实现了新浪微博的 OAuth 认证。

```
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            Weibo weibo = new Weibo();
            //根据 App Key 第三方应用向新浪获取 requestToken
            RequestToken requestToken = weibo.getOAuthRequestToken();
            System.out.println("1.....Got request token 成功");
            System.out.println("Request token: "+ requestToken.getToken());
            System.out.println("Request token secret: "+ requestToken.getTokenSecret());
            AccessToken accessToken = null;
           //用户从新浪获取 verifier_code 如果是 Android 或 iOS 应用可以 callback =json&userId=
xxs&password=XXX
            System.out.println("Open the following URL and grant access to your account:");
            System.out.println(requestToken.getAuthorizationURL());
            BareBonesBrowserLaunch.openURL(requestToken.getAuthorizationURL());
           //用户输入验证码授权信任第三方应用
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            while (null == accessToken) {
                System.out.print("Hit enter when it's done.[Enter]:");
                String pin = br.readLine();
                System.out.println("pin: " + br.toString());
                try{
                  //通过传递 requestToken 和用户验证码获取 AccessToken
                    accessToken = requestToken.getAccessToken(pin);
                } catch (WeiboException te) {
                    if(401 == te.getStatusCode()){
                      System.out.println("Unable to get the access token.");
                    }else{
                      te.printStackTrace();
                    }
                }
            }
            System.out.println("Got access token.");
            System.out.println("Access token: "+ accessToken.getToken());
            System.out.println("Access token secret: "+ accessToken.getTokenSecret());
```

```
//使用 AccessToken 来操作用户的所有接口
/* Weibo weibo=new Weibo();
 以后就可以用下面 accessToken 访问用户的资料了
 * weibo.setToken(accessToken.getToken(), accessToken.getTokenSecret());
 //发布微博
 Status status = weibo.updateStatus("test message6 ");
 System.out.println("Successfully updated the status to ["
 + status.getText() + "].");
      try {
 Thread.sleep(3000);
 } catch (InterruptedException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }*/
      System.exit(0);
 } catch (WeiboException te) {
      System.out.println("Failed to get timeline: " + te.getMessage());
      System.exit( -1);
 } catch (IOException ioe) {
      System.out.println("Failed to read the system input.");
      System.exit( -1);
 }
```

文件 OAuthActivity.java 实现了得到 OAuth 后的提示处理，验证成功后会得到 AccessToken 的 key 和 Secret，可以使用这两个参数进行授权登录。文件 OAuthActivity.java 的具体实现代码如下。

```
public class OAuthActivity extends Activity {
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.timeline);
            Uri uri=this.getIntent().getData();
            try {
                  RequestToken requestToken= OAuthConstant.getInstance().getRequestToken();
                  AccessToken accessToken=requestToken.getAccessToken(uri.getQueryParameter
                   ("oauth_verifier"));
                  OAuthConstant.getInstance().setAccessToken(accessToken);
                  TextView textView = (TextView) findViewById(R.id.TextView01);
                  textView.setText("得到 AccessToken 的 key 和 Secret,可以使用这两个参数进行授权
                  登录了.\n Access token:\n"+accessToken.getToken()+"\n Access token secret:\n"+
                  accessToken.getTokenSecret());
            } catch (WeiboException e) {
                  e.printStackTrace();
            }
            Button button=    (Button) findViewById(R.id.Button01);
            button.setText("某一话题下的微博");
            button.setOnClickListener(new Button.OnClickListener()
           {
```

```
        public void onClick( View v )
        {
                Weibo weibo=OAuthConstant.getInstance().getWeibo();
                weibo.setToken(OAuthConstant.getInstance().getToken(),
                OAuthConstant.getInstance().getTokenSecret());
                List<Status> friendsTimeline;
                    try {
                        friendsTimeline = weibo.getTrendStatus("seaeast", new Paging(1,20));
                        StringBuilder stringBuilder = new StringBuilder("1");
                    for (Status status : friendsTimeline) {
                        stringBuilder.append(status.getUser().getScreenName() + "说:"
                                + status.getText() + "-------------------------\n");
                    }
                    TextView textView = (TextView) findViewById(R.id.TextView01);
                    textView.setText(stringBuilder.toString());
                    } catch (WeiboException e) {
                        e.printStackTrace();
                    }

        }
    } );
  }
}
```

这样在执行后会显示一个触发 OAuth 认证的按钮，如图 14-4 所示。单击 GoGo 按钮后会得到 OAuth 认证后的 key 和 Secret，如图 14-5 所示。

图 14-4　触发按钮

图 14-5　得到 key 和 Secret

14.4.5　获取用户信息

通过文件 GetFollowers.java 可以返回用户关注对象列表，并返回最新微博文章。文件

GetFollowers.java 的具体实现代码如下。

```
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            List<User> list = weibo.getFollowersStatuses();
            for(User user : list) {
                System.out.println(user.toString());
            }
        } catch (WeiboException e) {
            e.printStackTrace();
        }
    }
```

通过文件 GetFriends.java 获取当前用户的朋友信息，具体实现代码如下。

```
    public static void main(String[] args) {
        try {
            System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
            System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            Weibo weibo = new Weibo();
            weibo.setToken(args[0], args[1]);
            try {
                List<User> list= weibo.getFriendsStatuses();
                System.out.println("Successfully get Friends to [" + list + "].");
            } catch (Exception e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            System.exit(0);
        } catch (Exception ioe) {
            System.out.println("Failed to read the system input.");
            System.exit( -1);
        }
    }
}
```

通过文件 GetUserInfo.java 获取当前登录用户的基本信息，具体实现代码如下。

```
public class GetUserInfo {

    /**
     * 根据用户 ID 获取用户资料（授权用户）
     * @param args
```

```
     * @throws UnsupportedEncodingException
     */
    public static void main(String[] args) throws UnsupportedEncodingException {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            String screen_name="李开复";
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            User user = weibo.showUser(screen_name);
            System.out.println(user.toString());
        } catch (WeiboException e) {
            e.printStackTrace();
        }
    }
}
```

14.4.6　关注用户

通过文件 ShowFriendships.java 显示当前 ID 用户和某个用户的关系，具体实现代码如下。

```
public class ShowFriendships {
    /**
     * 返回两个用户关系的详细情况
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
    System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            //自己与该关注对象的关系
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            JSONObject object = weibo.showFriendships(args[2]);//args[2]:关注用户的 id
            JSONObject source = object.getJSONObject("source");
            JSONObject target = object.getJSONObject("target");
            System.out.println(source.getString("screen_name")+"与"+target.getString("screen_
             name")+"互为关注");
            //两个用户关系的详细情况
            object =    weibo.showFriendships(args[3],args[4]);
            source = object.getJSONObject("source");
            target = object.getJSONObject("target");
            System.out.println(source.getString("screen_name")+"与"+target.getString("screen_name")+"互
            为关注");
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }
}
```

通过文件 ExistsFriendship.java 提醒当前 ID 用户是否关注这个用户，具体实现代码如下。

```
public static void main(String[] args) {
    System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
    System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
    try {
        //args[2]:自己的 id；args[3]:关注对象的 id
        Weibo weibo = new Weibo();
        weibo.setToken(args[0],args[1]);
        boolean bool = weibo.existsFriendship(args[2],args[3]);//args[2]:关注用户的 id
        System.out.println(bool);
    } catch (WeiboException e) {
        e.printStackTrace();
    }
}
```

通过文件 CreateFriendship.java 实现关注某一个用户的功能，具体实现代码如下。

```
public static void main(String[] args) {
    System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
    System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
    try {
        Weibo weibo = new Weibo();
        weibo.setToken(args[0],args[1]);
        User user = weibo.createFriendship(args[2]);//args[2]:关注用户的 id
        System.out.println(user.toString());
    } catch (WeiboException e) {
        e.printStackTrace();
    }
}
```

通过文件 DestroyFriendship.java 取消关注某一个用户，具体实现代码如下。

```
public static void main(String[] args) {
    System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
    System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
    try {
        Weibo weibo = new Weibo();
        weibo.setToken(args[0],args[1]);
        User user = weibo.destroyFriendship(args[2]);//args[2]:关注用户的 id
        System.out.println(user.toString());
    } catch (WeiboException e) {
```

```
                e.printStackTrace();
            }
        }
```

14.4.7　实现收藏功能

通过文件 GetFavorites.java 获取当前 ID 用户的收藏列表，具体实现代码如下。

```
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            List<Status> list = weibo.getFavorites();
            for(Status status : list) {
                System.out.println(status.toString());
            }
        } catch (WeiboException e) {
            e.printStackTrace();
        }
    }
```

通过文件 CreateFavorite.java 实现添加收藏功能，具体实现代码如下。

```
public class CreateFavorite {
    /**
     * 添加收藏
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
    System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            Status status = weibo.createFavorite(Long.parseLong(args[2]));
            System.out.println(status.toString());
        } catch (WeiboException e) {
            e.printStackTrace();
        }
    }
}
```

通过文件 DestroyFavorite.java 实现取消收藏功能，具体实现代码如下。

```
public class DestroyFavorite {
```

```
    /**
     * 删除当前用户收藏的微博信息
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            weibo.destroyFavorite(Long.parseLong(args[2]));
            List<Status> list = weibo.getFavorites();
            for(Status status : list) {
                System.out.println(status.toString());
            }
        } catch (WeiboException e) {
            e.printStackTrace();
        }
    }
}
```

14.4.8 实现微博操作功能

通过文件 DeleteComment.java 删除已经发布的某条评论，具体实现代码如下。

```
public class DeleteComment {
    /**
     * 删除评论,只能删除自己发布的评论
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            Status status = weibo.updateStatus("test4us");
            Thread.sleep(1000);
            String sid = status.getId()+"";
            System.out.println(sid + " : "+ status.getText()+"    "+status.getCreatedAt());
            Comment comment = weibo.updateComment("comment4u", sid, null);
            System.out.println(comment.getId() + " : " + comment.getText() + "    " + comment.
            getCreatedAt());
            Thread.sleep(1000);
            weibo.destroyComment(comment.getId());
        } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
    }
```

通过文件 DeleteStatus.java 删除某一条微博信息，具体实现代码如下。

```
public class DeleteStatus {
    /**
     * 删除一条微博信息
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            //先发表一篇微博
            Status status = weibo.updateStatus("测试测试");
            System.out.println(status.getId() + " : "+ status.getText()+"    "+status.getCreatedAt());
            //删除刚发表的微博
            status = weibo.destroyStatus(status.getId());
            List<Status> list = weibo.getUserTimeline(args[2]);//args[2]:用户 id
            for(Status st : list) {//遍历当前微博信息
                System.out.println(st.getId() + " : "+ st.getText()+"    "+st.getCreatedAt());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

通过文件 ForwardStatus.java 根据微博 ID 和用户 ID 跳转到单条微博页面，具体实现代码如下。

```
public class ForwardStatus {
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            List<Status> list = weibo.getUserTimeline();
            if(list.size() > 0) {
```

```
                    //args[2]:用户的 id
                    String url = "http://api.t.sina.com.cn/"+args[2]+"/statuses/"+list.get(0).getId();
                    //打开单条微博信息页面
                    Runtime.getRuntime().exec("cmd /c start "+url);
                }
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

通过文件 GetRepostTimeline.java 返回一条原创微博消息的最新 n 条转发微博消息，但是本接口无法对非原创微博进行查询。具体实现代码如下。

```
public class GetRepostTimeline {
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
            weibo.setToken(args[0],args[1]);
            List <Status> list =    weibo.getreposttimeline(args[2]);
            for(Status status:list){
            System.out.println(status.toString());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

通过文件 GetStatus.java 获取单条 ID 的微博信息，并同时返回作者信息。具体实现代码如下。

```
public class GetStatus {
    /**
     * 获取单条 ID 的微博信息，作者信息将同时返回
     * @param args
     */
    public static void main(String[] args) {
        System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
        System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
        try {
            Weibo weibo = new Weibo();
```

```
                weibo.setToken(args[0],args[1]);
                List<Status> list = weibo.getUserTimeline(args[2], new Paging(1).count(4));
                if(list.size() > 0) {
                    Status status = weibo.showStatus(list.get(0).getId());
                    System.out.println( status.getId() + "    : "+status.getText());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

通过文件 Reply.java 对一条微博评论信息进行回复，具体实现代码如下。

```
    public class Reply {
        public static void main(String[] args) {
            System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
            System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            try {
                Weibo weibo = new Weibo();
                weibo.setToken(args[0],args[1]);
                List<Status> list = weibo.getUserTimeline(args[2]);
                if(list.size() > 0) {
                    //最新一条微博信息 ID
                    String sid = list.get(0).getId()+"";
                    List<Comment> comments = weibo.getComments(sid);
                    if(comments.size() > 0) {
                        String cid = comments.get(0).getId()+"";//评论的 id
                        Comment status = weibo.reply(sid, cid, "回复内容");//args[3]：回复内容
                        System.out.println(status.toString());
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

通过文件 Repost.java 转发一条微博信息，具体实现代码如下。

```
    public class Repost {
        public static void main(String[] args) {
            System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
            System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            try {
                Weibo weibo = new Weibo();
```

```
                weibo.setToken(args[0],args[1]);
                String sid = "11814288270";
                Thread.sleep(1000);
                //args[2]：添加转发的信息
                Status status = weibo.repost(sid, args[2]);
                System.out.println(status.toString());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

通过文件 RepostByMe.java 获取用户最新转发的 n 条微博消息，具体实现代码如下。

```
    public class RepostByMe {
        public static void main(String[] args) {
            System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
            System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            try {
                Weibo weibo = new Weibo();
                weibo.setToken(args[0],args[1]);
                List <Status> list =    weibo.getrepostbyme(args[2]);
                for(Status status:list){
                System.out.println(status.toString());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

通过文件 UpdateComment.java 对一条微博信息进行评论，具体实现代码如下。

```
    public class UpdateComment {
        public static void main(String[] args) {
            System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
            System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            try {
                Weibo weibo = new Weibo();
                weibo.setToken(args[0],args[1]);
                Status status = weibo.updateStatus("test.....");
                Thread.sleep(1000);
                String sid = status.getId()+"";
                Comment comment = weibo.updateComment("1", sid, null);
                System.out.println(comment.getId() + " : " + comment.getText() + "   " +
```

```
                comment.getCreatedAt());
                Thread.sleep(1000);
                comment = weibo.updateComment("2", sid, null);
                System.out.println(comment.getId() + " : " + comment.getText() + "    " +
                comment.getCreatedAt());
                Thread.sleep(1000);
                comment = weibo.updateComment("3", sid, null);
                System.out.println(comment.getId() + " : " + comment.getText() + "    " +
                comment.getCreatedAt());
                Thread.sleep(1000);
                comment = weibo.updateComment("4", sid, null);
                System.out.println(comment.getId() + " : " + comment.getText() + "    " +
                comment.getCreatedAt());
                Thread.sleep(1000);
                comment = weibo.updateComment("5", sid, null);
                System.out.println(comment.getId() + " : " + comment.getText() + "    " +
                comment.getCreatedAt());
                Thread.sleep(1000);
                comment = weibo.updateComment("6", sid, null);
                System.out.println(comment.getId() + " : " + comment.getText() + "    " +
                comment.getCreatedAt());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

通过文件 UpdateStatus.java 发布一条新的微博信息，具体实现代码如下。

```
    public class UpdateStatus {
        public static void main(String[] args) {
            System.setProperty("weibo4j.oauth.consumerKey", Weibo.CONSUMER_KEY);
            System.setProperty("weibo4j.oauth.consumerSecret", Weibo.CONSUMER_SECRET);
            try {
                Weibo weibo = new Weibo();
                weibo.setToken(args[0],args[1]);
                Status status = weibo.updateStatus("夜深了，人睡了。");
                System.out.println(status.getId() + " : "+ status.getText()+"    "+status.getCreatedAt());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

第 15 章　开发 Web 版的电话本管理系统

经过本书前面内容的学习，已经掌握了 HTML5、jQuery Mobile 和 PhoneGap 的基本知识。在本章的内容中，将综合运用本书前面所学的知识，并结合使用 CSS 和 JavaScript 的技术，开发一个在 Android 平台运行的电话本管理系统。

15.1　需求分析

本实例使用“HTML 5+jQuery Mobile+PhoneGap”技术实现一个经典的电话本管理工具，实现对设备内联系人信息的管理，包括添加新信息、删除信息、快速搜索信息、修改信息、更新信息等功能。在本节的内容中，将对本项目进行必要的需求性分析。

15.1.1　产生背景

随着网络与信息技术的发展，很多陌生人之间也都有了或多或少的联系。如何更好地管理这些信息是每个人必须面临的问题，特别是那些很久没有联系的朋友，再次见面无法马上回忆关于这个人的信息，造成了一些不必要的尴尬。基于上述种种原因，开发一套通讯录管理系统很重要。

另外，随着移动设备平台的发展，以 Android 为代表的智能手机系统已经普及到普通消费者。智能手机设备已经成为了人们生活中必不可少的物品。在这种历史背景之下，手机通讯录变得愈发重要，已经成为人们离不开的联系人系统。

本系统的主要目的是为了更好的管理每个人的通讯录，给每个人提供一个井然有序的管理平台，防止手工管理混乱而造成的不必要麻烦。

15.1.2　功能分析

通过市场调查可知，一个完整的电话本管理系统应该包括：添加模块、主窗体模块、信息查询模块、信息修改模块、系统管理模块。本系统主要实现设备内联系人信息的管理，包括添加、修改、查询和删除。整个系统模块划分如图 15-1 所示。

（1）系统管理模块

用户通过此模块来管理设备内的联系人信息，在屏幕下方提供了实现系统管理的 5 个按钮。

- 搜索：单击此按钮后能够快速搜索设备内需要搜索的联系人信息。
- 添加：单击此按钮后能够向设备内添加新的联系人信息。
- 修改：单击此按钮后能够修改设备内已经存在的某条联系人信息。
- 删除：单击此按钮后删除设备内已经存在的某条联系人信息。

❑ 更新：单击此按钮后能够更新设备的所有联系人信息。

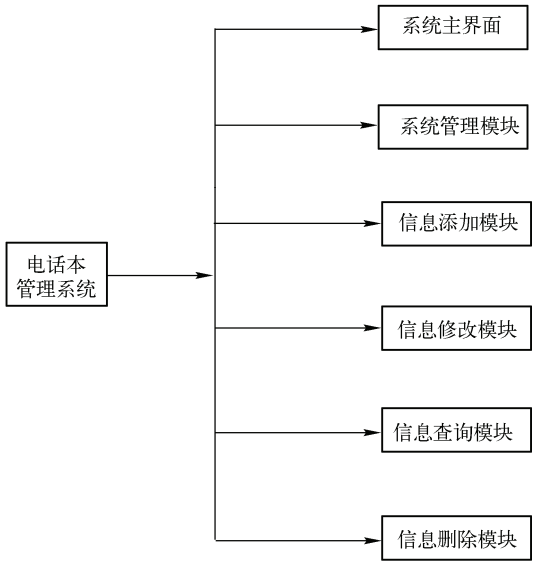


图 15-1　系统构成模块图

（2）系统主界面

在系统主屏幕界面中显示了两个操作按钮，通过这两个按钮可以快速进入本系统的核心功能。

❑ 查询：单击此按钮后能够来到系统搜索界面，能够快速搜索设备内我们需要的联系人信息。

❑ 管理：单击此按钮后能够来到系统管理模块的主界面。

（3）信息添加模块

通过此模块能够向设备中添加新的联系人信息。

（4）信息修改模块

通过此模块能够修改设备内已经存在的联系人信息。

（5）信息删除模块

通过此模块能够删除设备内已经存在的联系人信息。

（6）信息查询模块

通过此模块能够搜索设备内需要查询的联系人信息。

15.2　创建 Android 工程

（1）启动 Eclipse，依次选中 File→New→Other 选项，然后在树形结构中找到 Android 节点。并单击 Android Project，在项目名称栏中输入 phonebook。

（2）单击 Next 按钮，选择目标 SDK，在此选择 4.3。单击 Next 按钮，在其中填写包名 com.example.web_dhb，如图 15-2 所示。

图 15-2　创建 Android 工程

（3）单击 Next 按钮，此时将成功构建一个标准的 Android 项目。图 15-3 展示了当前项目的目录结构。

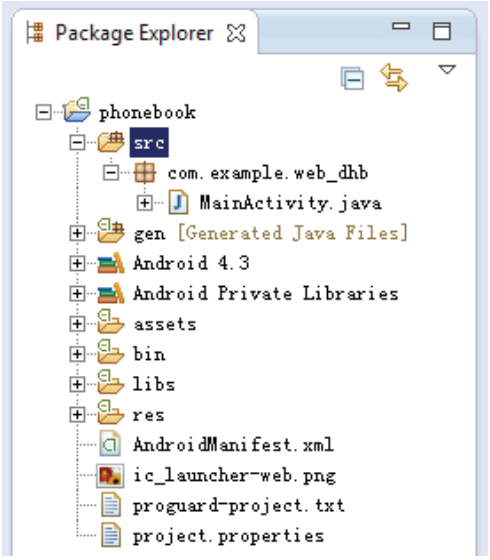


图 15-3　创建的 Android 工程

（4）修改文件 MainActivity.java，为此文件添加执行 HTML 文件的代码，主要代码如下。

```
public class MainActivity extends DroidGap {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.loadUrl("file:///android_asset/www/main.html");
    }
}
```

15.3　实现系统主界面

在本实例中，系统主界面的实现文件是 main.html，主要实现代码如下。

```
                    <script src="./js/jquery.js"></script>
                    <script src="./js/jquery.mobile-1.2.0.js"></script>
                    <script src="./cordova-2.1.0.js"></script>

            </head>
            <body>
                    <!-- Home -->
                    <div data-role="page" id="page1" style="background-image: url(./img/bg.gif);" >
                      <div data-theme="e" data-role="header">
                            <h2>电话本管理中心</h2>
                      </div>
                      <div data-role="content" style="padding-top:200px;">
                          <a data-role="button" data-theme="e" href="./select.html" id="chaxun"  data-icon=
"search" data-iconpos="left" data-transition="flip">查询</a>
                          <a data-role="button" data-theme="e" href="./set.html" id="guanli"  data-icon=
"gear" data-iconpos="left"> 管理 </a>
                      </div>
                      <div data-theme="e" data-role="footer" data-position="fixed">
                          <span class="ui-title">免费组织制作 v1.0</span>
                      </div>
                       <script type="text/javascript">
                          sessionStorage.setItem("uid","");
                           $('#page1').bind('pageshow',function(){
                              $.mobile.page.prototype.options.domCache = false;
                           });
                           // 等待加载 PhoneGap
                           document.addEventListener("deviceready", onDeviceReady, false);
                          // PhoneGap 加载完毕
                          function onDeviceReady() {
                            var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
                            db.transaction(populateDB, errorCB);
                          }
                            // 填充数据库
                             function populateDB(tx) {
                                  tx.executeSql('CREATE TABLE IF NOT EXISTS `myuser` (`user_id`
integer primary key autoincrement ,`user_name` VARCHAR( 25 ) NOT NULL ,`user_phone` varchar( 15 ) NOT
NULL ,`user_qq` varchar( 15 ) ,`user_email` VARCHAR( 50 ),`user_bz` TEXT)');

                             }
                             // 事务执行出错后调用的回调函数
                             function errorCB(tx, err) {
                                  alert("Error processing SQL: "+err);
                             }
                          </script>
                    </div>
                </body>
```

```
</html>
```

执行后的效果如图 15-4 所示。

图 15-4　执行效果

15.4　实现信息查询模块

信息查询模块的功能是快速搜索设备内需要查询的联系人信息。单击图 15-3 中的“查询”按钮后会跳转到查询界面，如图 15-5 所示。

图 15-5　查询界面

在查询界面上面的表单中可以输入搜索关键字，然后单击“查询”按钮后会在下方显示搜索结果。信息查询模块的实现文件是 select.html，主要实现代码如下。

```
        <script src="./js/jquery.js"></script>
        <script src="./js/jquery.mobile-1.2.0.js"></script>
        <!-- <script src="./cordova-2.1.0.js"></script> -->
    </head>
    <body>
    <body>
```

```
                    <!-- Home -->
                    <div data-role="page" id="page1">
                        <div data-theme="e" data-role="header">
                            <a data-role="button" href="./main.html" data-icon="back" data-iconpos="left" class="ui-btn-left">返回</a>
                            <a data-role="button" href="./main.html" data-icon="home" data-iconpos="right" class="ui-btn-right">首页</a>
                            <h3> 查询</h3>
                            <div >
                                <fieldset data-role="controlgroup" data-mini="true">
                                    <input name="" id="searchinput6" placeholder="输入联系人姓名" value="" type="search" />
                                </fieldset>
                            </div>
                            <div>
                              <input type="submit" id="search" data-theme="e" data-icon="search" data-iconpos="left" value="查询" data-mini="true" />
                            </div>
                        </div>
                        <div data-role="content">
                            <div class="ui-grid-b" id="contents" >
                                </div >
                        </div>
                        <script>
                            //App custom javascript
                          var u_name="";
                          <!-- 查询全部联系人 -->
                          // 等待加载 PhoneGap
                          document.addEventListener("deviceready", onDeviceReady, false);
                          // PhoneGap 加载完毕
                              function onDeviceReady() {
                                var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
                                db.transaction(queryDB, errorCB);  //调用 queryDB 查询方法，以及 errorCB 错误回调方法
                              }
                           // 查询数据库
                           function queryDB(tx) {
                                tx.executeSql('SELECT * FROM myuser', [], querySuccess, errorCB);
                            }
                           // 查询成功后调用的回调函数
                           function querySuccess(tx, results) {
                                var len = results.rows.length;
                                var str="<div class='ui-block-a' style='width:90px;'> 姓 名 </div><div class='ui-block-b'>电话</div><div class='ui-block-c'>拨号</div>";
                                console.log("myuser table: " + len + " rows found.");
                                for (var i=0; i<len; i++){
```

```
                                        //写入到 logcat 文件
                                        str                    +="<div                    class='ui-block-a'
style='width:90px;'>"+results.rows.item(i).user_name+"</div><div
class='ui-block-b'>"+results.rows.item(i).user_phone
                                            +"</div><div                                    class='ui-block-c'><a
href='tel:"+results.rows.item(i).user_phone+'"    data-role='button' class='ui-btn-right' >拨打  </a></div>";
                                    }
                                    $("#contents").html(str);
                                }
                                // 事务执行出错后调用的回调函数
                                function errorCB(err) {
                                    console.log("Error processing SQL: "+err.code);
                                }
                                <!-- 查询一条数据  -->
                                $("#search").click(function(){
                                    var searchinput6 = $("#searchinput6").val();
                                    u_name = searchinput6;
                                    var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
                                    db.transaction(queryDBbyone, errorCB);
                                });
                                function queryDBbyone(tx){
                                    tx.executeSql("SELECT * FROM myuser where user_name like '%"+u_name+"%'",
[], querySuccess, errorCB);
                                }
                            </script>
                        </div>
                    </body>
                </html>
```

15.5 实现系统管理模块

系统管理模块的功能是管理设备内的联系人信息，单击图 15-3 中的“管理”按钮后跳转到系统管理界面，如图 15-6 所示。

图 15-6　系统管理界面

在图 15-5 所示的界面中提供了实现系统管理的 5 个按钮，具体说明如下。

- 搜索：单击此按钮后能够快速搜索设备内需要查询的联系人信息。
- 添加：单击此按钮后能够向设备内添加新的联系人信息。
- 修改：单击此按钮后能够修改设备内已经存在的某条联系人信息。
- 删除：单击此按钮后删除设备内已经存在的某条联系人信息。
- 更新：单击此按钮后能够更新设备的所有联系人信息。

系统管理模块的实现文件是 set.html，主要实现代码如下。

```
<body>
    <!-- Home -->
    <div data-role="page" id="set_1"    data-dom-cache="false">
        <div data-theme="e" data-role="header" >
            <a  data-role="button"  href="main.html"  data-icon="home"  data-iconpos="right"
class="ui-btn-right">  主页</a>
            <h1>管理</h1>
            <a  data-role="button"  href="main.html"  data-icon="back"  data-iconpos="left"
class="ui-btn-left">后退  </a>
            <div >
                <span id="test"></span>
                 <fieldset data-role="controlgroup" data-mini="true">
                      <input  name=""  id="searchinput1"  placeholder="输入查询人的姓名"
value="" type="search" />
                  </fieldset>
            </div>
            <div>
                <input   type="submit"   id="search"   data-inline="true"   data-icon="search"
data-iconpos="top" value="搜索" />
                <input    type="submit"    id="add"    data-inline="true"    data-icon="plus"
data-iconpos="top"   value="添加"/>
                <input   type="submit"   id="modfiry"data-inline="true"   data-icon="minus"
data-iconpos="top" value="修改" />
                <input   type="submit"   id="delete"   data-inline="true"   data-icon="delete"
data-iconpos="top" value="删除" />
                <input   type="submit"   id="refresh"   data-inline="true"   data-icon="refresh"
data-iconpos="top" value="更新" />
            </div>
        </div>
        <div data-role="content">
          <div class="ui-grid-b" id="contents">
              </div >
        </div>
        <script type="text/javascript">
            $.mobile.page.prototype.options.domCache = false;
            var u_name="";
            var num="";
```

```
        var strsql="";
    <!-- 查询全部联系人 -->
    // 等待加载 PhoneGap
    document.addEventListener("deviceready", onDeviceReady, false);
    // PhoneGap 加载完毕
     function onDeviceReady() {
          var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
          db.transaction(queryDB, errorCB);  //调用 queryDB 查询方法，以及 errorCB
错误回调方法
        }
      // 查询数据库
    function queryDB(tx) {
          tx.executeSql('SELECT * FROM myuser', [], querySuccess, errorCB);
    }
      // 查询成功后调用的回调函数
    function querySuccess(tx, results) {
          var len = results.rows.length;
          var str="<div class='ui-block-a'> 编 号 </div><div class='ui-block-b'> 姓 名
</div><div class='ui-block-c'>电话</div>";
          for (var i=0; i<len; i++){
              str +="<div class='ui-block-a'><input type='checkbox' class='idvalue'
value="+results.rows.item(i).user_id+" /></div><div class='ui-block-b'>"+results.rows.item(i).user_name
                  +"</div><div
class='ui-block-c'>"+results.rows.item(i).user_phone+"</div>";
          }
          $("#contents").html(str);
    }
    // 事务执行出错后调用的回调函数
    function errorCB(err) {
          console.log("Error processing SQL: "+err.code);
    }
    <!-- 查询一条数据 -->
    $("#search").click(function(){
          var searchinput1 = $("#searchinput1").val();
          u_name = searchinput1;
          var db = window.openDatabase("Database", "1.0", "PhoneGap myuser",
          200000);
          db.transaction(queryDBbyone, errorCB);
    });
    function queryDBbyone(tx){
          tx.executeSql("SELECT * FROM myuser where user_name like
          '%"+u_name+"%'", [], querySuccess, errorCB);
    }
    $("#delete").click(function(){
          var len = $("input:checked").length;
          for(var i=0;i<len;i++){
```

```
                num +=","+$("input:checked")[i].value;
        }
        num=num.substr(1);
        var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
        db.transaction(deleteDBbyid, errorCB);
});
function deleteDBbyid(tx){
        tx.executeSql("DELETE FROM `myuser` WHERE user_id in("+num+")", [],
        queryDB, errorCB);
}
        $("#add").click(function(){
                $.mobile.changePage ('add.html', 'fade', false, false);
        });
        $("#modfiry").click(function(){
                if($("input:checked").length==1){
                    var userid=$("input:checked").val();
                    sessionStorage.setItem("uid",userid);
                    $.mobile.changePage ('modfiry.html', 'fade', false, false);
                }else{
                      alert("请选择要修改的联系人，并且每次只能选择一位");
                }

        });
        //=============与手机联系人 同步数据=====================
        $("#refresh").click(function(){
                // 从全部联系人中进行搜索
        var options = new ContactFindOptions();
        options.filter="";
        var filter = ["displayName","phoneNumbers"];
        options.multiple=true;
        navigator.contacts.find(filter, onTbSuccess, onError, options);
        });
        // onSuccess: 返回当前联系人结果集的快照
function onTbSuccess(contacts) {
        // 显示所有联系人的地址信息
        var str="<div class='ui-block-a'>编号</div><div class='ui-block-b'>姓名
        </div><div class='ui-block-c'>电话</div>";
        var phone;
        var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
        for (var i=0; i<contacts.length; i++){
                for(var j=0; j< contacts[i].phoneNumbers.length; j++){
                    phone = contacts[i].phoneNumbers[j].value;
                }
                strsql +="INSERT INTO `myuser` (`user_name`,`user_phone`) VALUES
                ('"+contacts[i].displayName+"',"+phone+");#";
        }
```

```
                db.transaction(addBD, errorCB);
            }
            // 更新插入数据
            function addBD(tx){
                strs=strsql.split("#");
                for(var i=0;i<strs.length;i++){
                    tx.executeSql(strs[i], [], [], errorCB);
                }
                var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
                db.transaction(queryDB, errorCB);
            }
            // onError: 获取联系人结果集失败
            function onError() {
                console.log("Error processing SQL: "+err.code);
            }
        </script>
    </div>
</body>
```

15.6 实现信息添加模块

在图 15-5 所示的界面中提供了实现系统管理的 5 个按钮，如果单击“添加”按钮则会跳转到信息添加界面，通过此界面可以向设备中添加新的联系人信息，如图 15-7 所示。

图 15-7 信息添加界面

信息添加模块的实现文件是 add.html，主要实现代码如下。

```
<body>
 <!-- Home -->
      <div data-role="page" id="page1">
          <div data-theme="e" data-role="header">
              <a data-role="button" id="tjlxr" data-theme="e" data-icon="info"
data-iconpos="right" class="ui-btn-right">保存</a>
              <h3>添加联系人 </h3>
              <a data-role="button" id="czlxr" data-theme="e" data-icon="refresh"
data-iconpos="left" class="ui-btn-left"> 重置</a>
          </div>
          <div data-role="content">
              <form action="" data-theme="e" >
                  <div data-role="fieldcontain">
                      <fieldset data-role="controlgroup" data-mini="true">
                          <label for="textinput1"> 姓 名 ： <input name="" id="textinput1"
placeholder="联系人姓名" value="" type="text" /></label>
                      </fieldset>
                      <fieldset data-role="controlgroup" data-mini="true">
                          <label for="textinput2"> 电 话 ： <input name="" id="textinput2"
placeholder="联系人电话" value="" type="tel" /></label>
                      </fieldset>
                      <fieldset data-role="controlgroup" data-mini="true">
                          <label for="textinput3">QQ ： <input name="" id="textinput3"
placeholder="" value="" type="number" /></label>
                      </fieldset>
                       <fieldset data-role="controlgroup" data-mini="true">
                           <label for="textinput4">Emai ： <input name="" id="textinput4"
placeholder="" value="" type="email" /></label>
                       </fieldset>
                       <fieldset data-role="controlgroup">
                           <label for="textarea1"> 备注：</label>
                           <textarea name="" id="textarea1" placeholder=""
data-mini="true"></textarea>
                       </fieldset>
                  </div>
                  <div>
                      <a data-role="button" id="back" data-theme="e" >返回</a>
                  </div>
              </form>
          </div>
          <script type="text/javascript">
          $.mobile.page.prototype.options.domCache = false;
          var textinput1 = "";
            var textinput2 = "";
```

```
var textinput3 = "";
var textinput4 = "";
var textarea1    = "";
  $("#tjlxr").click(function(){
      textinput1 =    $("#textinput1").val();
      textinput2 =    $("#textinput2").val();
      textinput3 =    $("#textinput3").val();
      textinput4 =    $("#textinput4").val();
      textarea1    =     $("#textarea1").val();
    var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
    db.transaction(addBD, errorCB);
  });
  function addBD(tx){
      tx.executeSql("INSERT    INTO    `myuser`    (`user_name`,`user_phone`,`user_qq`,
`user_email`,`user_bz`) VALUES ('"+textinput1+"','"+textinput2+"','"+textinput3+"','"+textinput4+"','"+textarea1+"')",
[], successCB, errorCB);
  }
  $("#czlxr").click(function(){
      $("#textinput1").val("");
      $("#textinput2").val("");
      $("#textinput3").val("");
      $("#textinput4").val("");
      $("#textarea1").val("");
  });
  $("#back").click(function(){
      successCB();
  });
  // 等待加载 PhoneGap
  document.addEventListener("deviceready", onDeviceReady, false);
  // PhoneGap 加载完毕
  function onDeviceReady() {
    var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
    db.transaction(populateDB, errorCB);
  }
    // 填充数据库
    function populateDB(tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS `myuser` (`user_id` integer
primary key autoincrement ,`user_name` VARCHAR( 25 ) NOT NULL ,`user_phone` varchar( 15 ) NOT
NULL ,`user_qq` varchar( 15 ) ,`user_email` VARCHAR( 50 ),`user_bz` TEXT)');
    }
    // 事务执行出错后调用的回调函数
    function errorCB(tx, err) {
        alert("Error processing SQL: "+err);
    }
```

```
                    // 事务执行成功后调用的回调函数
                    function successCB() {
                        $.mobile.changePage ('set.html', 'fade', false, false);
                    }
                </script>
            </div>
        </body>
```

15.7　实现信息修改模块

在图 15-5 所示的界面中，如果先勾选一个联系人信息，然后单击“修改”按钮后会来到信息修改界面，通过此界面可以修改这条被选中联系人的信息，如图 15-8 所示。

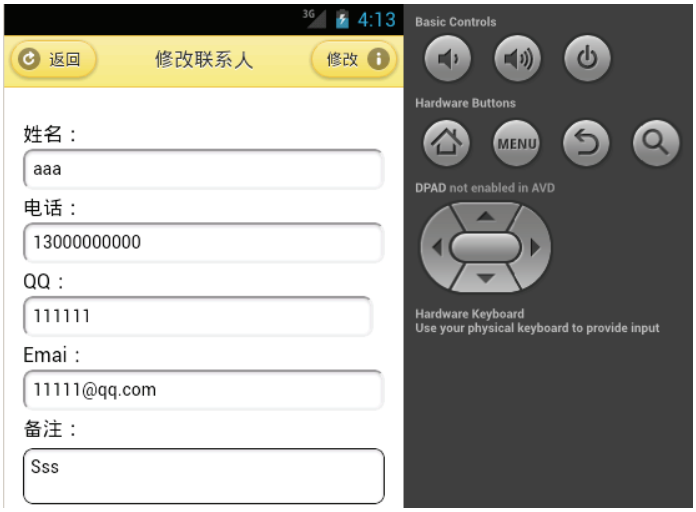


图 15-8　信息修改界面

信息修改模块的实现文件是 modfiry.html，主要实现代码如下。

```
        <script type="text/javascript" src="./js/jquery.js"></script>
        </head>
        <body>
          <!-- Home -->
                <div data-role="page" id="page1">
                    <div data-theme="e" data-role="header">
                        <a data-role="button" id="tjlxr" data-theme="e" data-icon="info"
data-iconpos="right" class="ui-btn-right">修改</a>
                        <h3>修改联系人 </h3>
                        <a data-role="button" id="back" data-theme="e" data-icon="refresh"
data-iconpos="left" class="ui-btn-left"> 返回</a>
                    </div>
```

```
                    <div data-role="content">
                        <form action="" data-theme="e" >
                            <div data-role="fieldcontain">
                                <fieldset data-role="controlgroup" data-mini="true">
                                    <label for="textinput1"> 姓 名 ： <input name="" id="textinput1"
placeholder="联系人姓名" value="" type="text" /></label>
                                </fieldset>
                                <fieldset data-role="controlgroup" data-mini="true">
                                    <label for="textinput2"> 电 话 ： <input name="" id="textinput2"
placeholder="联系人电话" value="" type="tel" /></label>
                                </fieldset>
                                <fieldset data-role="controlgroup" data-mini="true">
                                    <label for="textinput3">QQ ： <input name="" id="textinput3"
placeholder="" value="" type="number" /></label>
                                </fieldset>
                                 <fieldset data-role="controlgroup" data-mini="true">
                                     <label for="textinput4">Emai ： <input name="" id="textinput4"
placeholder="" value="" type="email" /></label>
                                 </fieldset>
                                 <fieldset data-role="controlgroup">
                                     <label for="textarea1"> 备注：</label>
                                     <textarea name="" id="textarea1" placeholder="" data-mini="true">
</textarea>
                                 </fieldset>
                            </div>
                        </form>
                    </div>
                    <script type="text/javascript">
                    $.mobile.page.prototype.options.domCache = false;
                    var textinput1 = "";
                      var textinput2 = "";
                      var textinput3 = "";
                      var textinput4 = "";
                      var textarea1   = "";
                      var uid = sessionStorage.getItem("uid");
                      $("#tjlxr").click(function(){

                          textinput1 =   $("#textinput1").val();
                          textinput2 =   $("#textinput2").val();
                          textinput3 =   $("#textinput3").val();
                          textinput4 =   $("#textinput4").val();
                          textarea1   =    $("#textarea1").val();
                        var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
                          db.transaction(modfiyBD, errorCB);
                      });
                        function modfiyBD(tx){
```

```
                    tx.executeSql("UPDATE  `myuser`SET   `user_name`='"+textinput1+"',`user_phone`
='"+textinput2+"',`user_qq`="+textinput3      +",`user_email`='"+textinput4+"',`user_bz`='"+textarea1+"'      WHERE
user_id="+uid, [], successCB, errorCB);
                }
                $("#back").click(function(){
                    successCB();
                });
                document.addEventListener("deviceready", onDeviceReady, false);
                // PhoneGap 加载完毕
                function onDeviceReady() {
                  var db = window.openDatabase("Database", "1.0", "PhoneGap myuser", 200000);
                  db.transaction(selectDB, errorCB);
                }
                  function selectDB(tx) {
                      tx.executeSql("SELECT * FROM myuser where user_id="+uid, [], querySuccess,
errorCB);
                  }
                  // 事务执行出错后调用的回调函数
                  function errorCB(tx, err) {
                        alert("Error processing SQL: "+err);
                  }
                  // 事务执行成功后调用的回调函数
                  function successCB() {
                        $.mobile.changePage ('set.html', 'fade', false, false);
                  }
                  function querySuccess(tx, results) {
                  var len = results.rows.length;
                  for (var i=0; i<len; i++){
                        //写入到 logcat 文件
                          $("#textinput1").val(results.rows.item(i).user_name);
                          $("#textinput2").val(results.rows.item(i).user_phone);
                          $("#textinput3").val(results.rows.item(i).user_qq);
                          $("#textinput4").val(results.rows.item(i).user_email);
                          $("#textarea1").val(results.rows.item(i).user_bz);
                  }
            }
             </script>
        </div>
    </body>
    </html>
```

15.8　实现信息删除模块和更新模块

在图 15-5 所示的界面中，如果先勾选一个联系人信息，然后单击“删除”按钮后会删除这条被勾选的联系人信息。信息删除模块的功能在文件 set.html 中实现，相关的

实现代码如下。

```
function deleteDBbyid(tx){
    tx.executeSql("DELETE FROM `myuser` WHERE user_id in("+num+")", [], queryDB, errorCB);
}
```

在图 15-5 所示的界面中，如果单击“更新”按钮则会更新整个设备内的联系人信息。信息更新模块的功能在文件 set.html 中实现，相关的实现代码如下。

```
$("#refresh").click(function(){
    // 从全部联系人中进行搜索
    var options = new ContactFindOptions();
    options.filter="";
    var filter = ["displayName","phoneNumbers"];
    options.multiple=true;
    navigator.contacts.find(filter, onTbSuccess, onError, options);
});
```

第 16 章　开发移动微信系统

微信是腾讯公司于 2011 年 1 月推出的一款通过网络快速发送语音短信、视频、图片和文字，支持多人群聊的手机聊天软件。用户可以通过微信与好友进行形式上更加丰富的联系。微信软件本身完全免费，使用时产生的上网流量费由网络运营商收取。2012 年 9 月 17 日，微信注册用户过 2 亿。在本章的内容中，将简要介绍在 Android 平台中开发一个微信系统的基本思路和流程。

16.1　微信系统基础

微信系统和 QQ 聊天软件类似，也是一款通讯工具，能够实现在线实时交流。在本节的内容中，将简要讲解微信系统的基本知识。

16.1.1　微信的特点

微信是一种更快速的即时通讯工具，具有零资费、跨平台沟通、显示实时输入状态等功能，与传统的短信沟通方式相比，更灵活、智能，且节省资费。微信的具体特点如下。

- 支持发送语音短信、视频、图片（包括表情）和文字。
- 支持多人群聊。
- 支持查看所在位置附近使用微信的人（LBS 功能）。
- 支持腾讯微博、QQ 邮箱、漂流瓶、语音记事本、QQ 同步助手等插件功能。
- 支持视频聊天。
- 微行情：支持及时查询股票行情。
- 多平台，支持 iPhone、Android、Windows Phone、塞班、blackberry 平台的手机之间相互收发消息，省流量。

16.1.2　微信和 Q 信的关系

Q 信是另一款腾讯手机软件 QQ 通讯录中的一个功能，与微信功能极其相似。但却是两个不同的软件，可以进行以下区分。

（1）微信上也集成很多插件，如 QQ 邮箱助手、QQ 离线助手、通讯录安全助手等，Q 信则只是 QQ 通讯录上的一个功能，没有下层插件。

（2）微信的好友是基于手机通讯录中的联系人和 QQ 上的好友，而 Q 信只基于手机通讯录中的联系人。

（3）Q 信上能够显示好友手机号码，微信只能显示称呼或者备注名字，不能显示手机号码。

Q 信和微信虽然基本功能一样，可以通过网络快速发送（需消耗少量网络流量）语音短信、视频、图片和文字，支持多人群聊，但却是两个独立的软件，属腾讯的不同产品。

16.2 使用 Android ViewPager

在 Android 系统中，谷歌提供了 ViewPager 实现多页面滑动切换以及动画效果。在本章的微信系统中，将使用 ViewPager 来实现界面之间的切换工作。在下面的内容中，将通过一个实例来讲解使用 Android ViewPager 的基本流程。

实　例	功　能	源码路径
实例 16-1	使用 Android ViewPager 实现切换	daima\16\DWinterTabDemo

本实例的具体实现流程如下。

（1）本实例需要用到 ViewPager 控件，这是 SDK 中自带的一个附加包的一个类，可以用来实现屏幕间的切换。使用本控件需要在工程中包 android-support-v4.jar，放在 libs 文件夹中。当然也可以从网上搜索最新的版本。本实例的工程目录如图 16-1 所示。

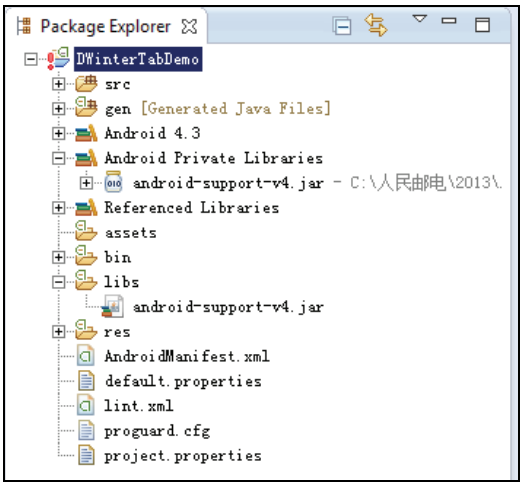


图 16-1　工程目录

（2）开始设计界面，整个设计工作非常简单，第一行为 3 个头标，第二行为动画图片，第三行为页卡内容展示。布局代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:umadsdk="http://schemas.android.com/apk/res/com.LoveBus"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
<LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="fill_parent"
```

```
            android:layout_height="100.0dip"
            android:background="#FFFFFF" >
    <TextView
                android:id="@+id/text1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_weight="1.0"
                android:gravity="center"
                android:text="页卡 1"
                android:textColor="#000000"
                android:textSize="22.0dip" />
    <TextView
                android:id="@+id/text2"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_weight="1.0"
                android:gravity="center"
                android:text="页卡 2"
                android:textColor="#000000"
                android:textSize="22.0dip" />
    <TextView
                android:id="@+id/text3"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_weight="1.0"
                android:gravity="center"
                android:text="页卡 3"
                android:textColor="#000000"
                android:textSize="22.0dip" />
    </LinearLayout>
    <ImageView
            android:id="@+id/cursor"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:scaleType="matrix"
            android:src="@drawable/a" />

    <android.support.v4.view.ViewPager
            android:id="@+id/vPager"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_weight="1.0"
            android:background="#000000"
            android:flipInterval="30"
            android:persistentDrawingCache="animation" />
```

```
</LinearLayout>
```

因为本项目需要展示 3 个页卡，所以还需要 3 个页卡内容的界面设计，这里只设置了背景颜色，能起到区别作用即可。具体代码如下。

```
<?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:background="#158684" >
</LinearLayout>
```

（3）接下来开始具体编码工作，首先进行初始化的工作，实现变量的定义的代码如下。

```
private ViewPager mPager;//页卡内容
private List<View> listViews; // Tab 页面列表
private ImageView cursor;// 动画图片
private TextView t1, t2, t3;// 页卡头标
private int offset = 0;// 动画图片偏移量
private int currIndex = 0;// 当前页卡编号
private int bmpW;// 动画图片宽度
```

❑ 初始化头标并响应点击事件的实现代码如下。

```
/**
     * 初始化头标
 */
    private void InitTextView() {
        t1 = (TextView) findViewById(R.id.text1);
        t2 = (TextView) findViewById(R.id.text2);
        t3 = (TextView) findViewById(R.id.text3);
        t1.setOnClickListener(new MyOnClickListener(0));
        t2.setOnClickListener(new MyOnClickListener(1));
        t3.setOnClickListener(new MyOnClickListener(2));
    }
/**
     * 头标点击监听
 */
    public class MyOnClickListener implements View.OnClickListener {
        private int index = 0;

        public MyOnClickListener(int i) {
            index = i;
        }
        @Override
        public void onClick(View v) {
            mPager.setCurrentItem(index);
```

```
        }
    };
```

❑ 初始化页卡内容区的实现代码如下。

```
/**
     * 初始化 ViewPager
 */
    private void InitViewPager() {
        mPager = (ViewPager) findViewById(R.id.vPager);
        listViews = new ArrayList<View>();
        LayoutInflater mInflater = getLayoutInflater();
        listViews.add(mInflater.inflate(R.layout.lay1, null));
        listViews.add(mInflater.inflate(R.layout.lay2, null));
        listViews.add(mInflater.inflate(R.layout.lay3, null));
        mPager.setAdapter(new MyPagerAdapter(listViews));
        mPager.setCurrentItem(0);
        mPager.setOnPageChangeListener(new MyOnPageChangeListener());
    }
```

❑ 将 3 个页卡界面装入其中，默认显示第一个页卡。在此处还需要通过代码实现一个适配器，具体代码如下。

```
/**
     * ViewPager 适配器
 */
    public class MyPagerAdapter extends PagerAdapter {
        public List<View> mListViews;
        public MyPagerAdapter(List<View> mListViews) {
            this.mListViews = mListViews;
        }
        @Override
        public void destroyItem(View arg0, int arg1, Object arg2) {
            ((ViewPager) arg0).removeView(mListViews.get(arg1));
        }
        @Override
        public void finishUpdate(View arg0) {
        }
        @Override
        public int getCount() {
            return mListViews.size();
        }
        @Override
        public Object instantiateItem(View arg0, int arg1) {
            ((ViewPager) arg0).addView(mListViews.get(arg1), 0);
            return mListViews.get(arg1);
        }
```

```
        @Override
        public boolean isViewFromObject(View arg0, Object arg1) {
            return arg0 == (arg1);
        }
        @Override
        public void restoreState(Parcelable arg0, ClassLoader arg1) {
        }
        @Override
        public Parcelable saveState() {
            return null;
        }
        @Override
        public void startUpdate(View arg0) {
        }
    }
```

到此为止，就实现了各页卡的装入和卸载工作。

❑ 初始化动画的实现代码如下。

```
/**
     * 初始化动画
 */
    private void InitImageView() {
        cursor = (ImageView) findViewById(R.id.cursor);
        bmpW = BitmapFactory.decodeResource(getResources(), R.drawable.a)
                .getWidth();// 获取图片宽度
        DisplayMetrics dm = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(dm);
        int screenW = dm.widthPixels;// 获取分辨率宽度
        offset = (screenW / 3 - bmpW) / 2;// 计算偏移量
        Matrix matrix = new Matrix();
        matrix.postTranslate(offset, 0);
        cursor.setImageMatrix(matrix);// 设置动画初始位置
    }
```

❑ 实现页卡切换监听事件的处理代码如下。

```
/**
     * 页卡切换监听
 */
    public class MyOnPageChangeListener implements OnPageChangeListener {
        int one = offset * 2 + bmpW;// 页卡 1 ->页卡 2 偏移量
        int two = one * 2;// 页卡 1 ->页卡 3 偏移量
        @Override
        public void onPageSelected(int arg0) {
            Animation animation = null;
            switch (arg0) {
            case 0:
```

```
                    if (currIndex == 1) {
                        animation = new TranslateAnimation(one, 0, 0, 0);
                    } else if (currIndex == 2) {
                        animation = new TranslateAnimation(two, 0, 0, 0);
                    }
                    break;
                case 1:
                    if (currIndex == 0) {
                        animation = new TranslateAnimation(offset, one, 0, 0);
                    } else if (currIndex == 2) {
                        animation = new TranslateAnimation(two, one, 0, 0);
                    }
                    break;
                case 2:
                    if (currIndex == 0) {
                        animation = new TranslateAnimation(offset, two, 0, 0);
                    } else if (currIndex == 1) {
                        animation = new TranslateAnimation(one, two, 0, 0);
                    }
                    break;
                }
                currIndex = arg0;
                animation.setFillAfter(true);// True:图片停在动画结束位置
                animation.setDuration(300);
                cursor.startAnimation(animation);
            }
            @Override
            public void onPageScrolled(int arg0, float arg1, int arg2) {
            }
            @Override
            public void onPageScrollStateChanged(int arg0) {
            }
        }
```

到此为止，整个实例介绍完毕，执行后会实现动画切换效果，如图 16-2 所示。

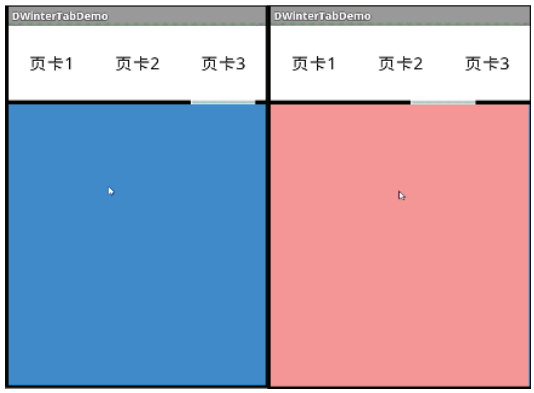


图 16-2　执行效果

16.3 开发一个微信系统

在本节的内容中，将通过一个具体实例的实现过程，讲解开发一个 Android 微信系统的基本流程。

实　例	功　能	源码路径
实例 16-2	开发一个微信系统	daima\16\Weixin

16.3.1 启动界面

使用过微信的用户都知道，每次启动程序都会出现一个启动画面，如果是第一次使用则还会出现后面的导航界面。当本实例启动后会进入第一个 Activity，此 Activity 就是一个启动画面，之后会在这个 Activity 里面设置一个 Handler 去延迟（1 秒，数值可以自己设定）执行启动导航界面的 Activity。

启动界面的 UI 文件是 appstart.xml，具体代码如下。

```
<?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/welcome" >
</LinearLayout>
```

启动界面的实现文件是 Appstart.java，具体代码如下。

```
package cn.buaa.myweixin;
import android.os.Bundle;
import android.os.Handler;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.WindowManager;
public class Appstart extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.appstart);
        new Handler().postDelayed(new Runnable(){
            public void run(){
                Intent intent = new Intent (Appstart.this,Welcome.class);
                startActivity(intent);
                Appstart.this.finish();
            }
        }, 1000);
    }
```

```
}
```

执行后的效果如图 16-3 所示。

图 16-3　执行效果

16.3.2　系统导航界面

进入系统后，会在界面下方显示导航选项卡，分别显示“微信”“通讯录”“朋友们”和“设置”4 个选项，如图 16-4 所示。

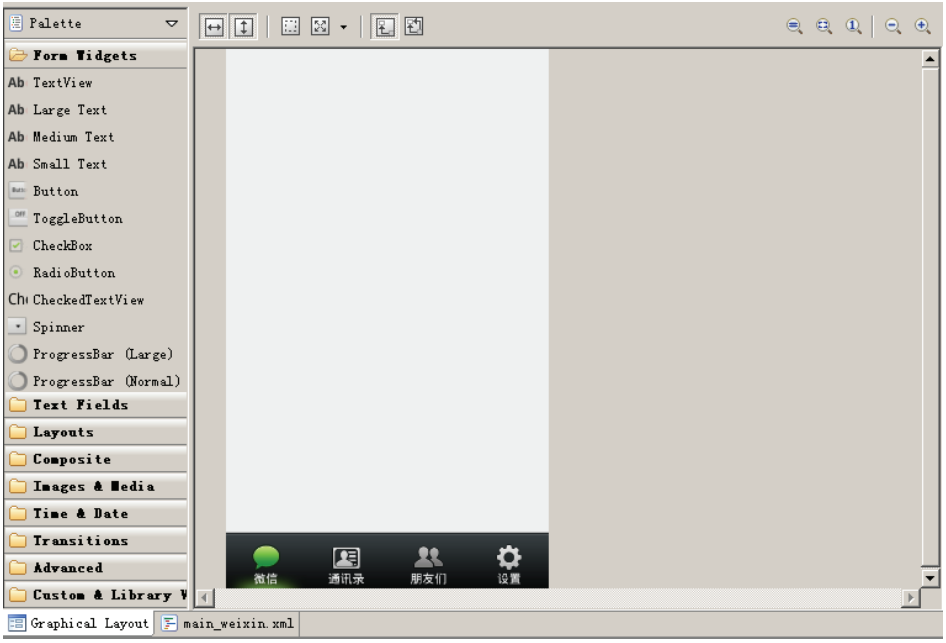


图 16-4　导航选项卡的设计界面

导航界面的布局文件是 main_weixin.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mainweixin"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="#eee" >
<RelativeLayout
        android:id="@+id/main_bottom"
        android:layout_width="match_parent"
        android:layout_height="55dp"
        android:layout_alignParentBottom="true"
        android:orientation="vertical"
        android:background="@drawable/bottom_bar">
<ImageView
    android:id="@+id/img_tab_now"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="matrix"
    android:layout_gravity="bottom"
            android:layout_alignParentBottom="true"
    android:src="@drawable/tab_bg" />
<LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:paddingBottom="2dp">

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:layout_weight="1">
<ImageView
        android:id="@+id/img_weixin"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:scaleType="matrix"
            android:clickable="true"
            android:src="@drawable/tab_weixin_pressed" />
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
            android:text="微信"
            android:textColor="#fff"
            android:textSize="12sp" />
        </LinearLayout>
        <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:layout_weight="1">
    <ImageView
            android:id="@+id/img_address"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:scaleType="matrix"
                android:clickable="true"
                android:src="@drawable/tab_address_normal" />
    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="通讯录"
            android:textColor="#fff"
            android:textSize="12sp" />
        </LinearLayout>
        <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:layout_weight="1">
    <ImageView
            android:id="@+id/img_friends"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:scaleType="matrix"
                android:clickable="true"
                android:src="@drawable/tab_find_frd_normal" />
    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="朋友们"
            android:textColor="#fff"
            android:textSize="12sp" />
        </LinearLayout>

        <LinearLayout
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:layout_weight="1">
    <ImageView
            android:id="@+id/img_settings"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:scaleType="matrix"
                android:clickable="true"
                android:src="@drawable/tab_settings_normal" />
    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="设置"
            android:textColor="#fff"
            android:textSize="12sp" />
        </LinearLayout>

    </LinearLayout>

    </RelativeLayout>
    <LinearLayout
            android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_above="@id/main_bottom"
            android:orientation="vertical" >
    <android.support.v4.view.ViewPager
        android:id="@+id/tabpager"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" >
    </android.support.v4.view.ViewPager>
    </LinearLayout>
    </RelativeLayout>
```

对应的实现文件是 MainWeixin.java，具体代码如下。

```
package cn.buaa.myweixin;
import java.util.ArrayList;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.support.v4.view.PagerAdapter;
import android.support.v4.view.ViewPager;
```

```
import android.support.v4.view.ViewPager.OnPageChangeListener;
import android.view.Display;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.View;
import android.view.WindowManager;
import android.view.WindowManager.LayoutParams;
import android.view.animation.Animation;
import android.view.animation.TranslateAnimation;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.PopupWindow;
public class MainWeixin extends Activity {
    public static MainWeixin instance = null;
    private ViewPager mTabPager;
    private ImageView mTabImg;// 动画图片
    private ImageView mTab1, mTab2, mTab3, mTab4;
    private int zero = 0;// 动画图片偏移量
    private int currIndex = 0;// 当前页卡编号
    private int one;// 单个水平动画位移
    private int two;
    private int three;
    private LinearLayout mClose;
    private LinearLayout mCloseBtn;
    private View layout;
    private boolean menu_display = false;
    private PopupWindow menuWindow;
    private LayoutInflater inflater;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_weixin);
        // 启动 Activity 时不自动弹出软键盘
        getWindow().setSoftInputMode(
                WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN);
        instance = this;
        mTabPager = (ViewPager) findViewById(R.id.tabpager);
        mTabPager.setOnPageChangeListener(new MyOnPageChangeListener());
        mTab1 = (ImageView) findViewById(R.id.img_weixin);
        mTab2 = (ImageView) findViewById(R.id.img_address);
        mTab3 = (ImageView) findViewById(R.id.img_friends);
        mTab4 = (ImageView) findViewById(R.id.img_settings);
        mTabImg = (ImageView) findViewById(R.id.img_tab_now);
        mTab1.setOnClickListener(new MyOnClickListener(0));
        mTab2.setOnClickListener(new MyOnClickListener(1));
        mTab3.setOnClickListener(new MyOnClickListener(2));
        mTab4.setOnClickListener(new MyOnClickListener(3));
```

```
        Display currDisplay = getWindowManager().getDefaultDisplay();// 获取屏幕当前分辨率
        int displayWidth = currDisplay.getWidth();
        int displayHeight = currDisplay.getHeight();
        one = displayWidth / 4; // 设置水平动画平移大小
        two = one * 2;
        three = one * 3;
        // 将要分页显示的 View 装入数组中
        LayoutInflater mLi = LayoutInflater.from(this);
        View view1 = mLi.inflate(R.layout.main_tab_weixin, null);
        View view2 = mLi.inflate(R.layout.main_tab_address, null);
        View view3 = mLi.inflate(R.layout.main_tab_friends, null);
        View view4 = mLi.inflate(R.layout.main_tab_settings, null);
        // 每个页面的 view 数据
        final ArrayList<View> views = new ArrayList<View>();
        views.add(view1);
        views.add(view2);
        views.add(view3);
        views.add(view4);
        // 填充 ViewPager 的数据适配器
        PagerAdapter mPagerAdapter = new PagerAdapter() {
            @Override
            public boolean isViewFromObject(View arg0, Object arg1) {
                return arg0 == arg1;
            }
            @Override
            public int getCount() {
                return views.size();
            }
            @Override
            public void destroyItem(View container, int position, Object object) {
                ((ViewPager) container).removeView(views.get(position));
            }
            @Override
            public Object instantiateItem(View container, int position) {
                ((ViewPager) container).addView(views.get(position));
                return views.get(position);
            }
        };
        mTabPager.setAdapter(mPagerAdapter);
    }
    /**
     * 头标点击监听
     */
    public class MyOnClickListener implements View.OnClickListener {
        private int index = 0;
        public MyOnClickListener(int i) {
            index = i;
        }
```

```
        public void onClick(View v) {
            mTabPager.setCurrentItem(index);
        }
    };
    /*
     * 页卡切换监听(原作者:D.Winter)
     */
    public class MyOnPageChangeListener implements OnPageChangeListener {
        public void onPageSelected(int arg0) {
            Animation animation = null;
            switch (arg0) {
            case 0:
                mTab1.setImageDrawable(getResources().getDrawable(
                        R.drawable.tab_weixin_pressed));
                if (currIndex == 1) {
                    animation = new TranslateAnimation(one, 0, 0, 0);
                    mTab2.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_address_normal));
                } else if (currIndex == 2) {
                    animation = new TranslateAnimation(two, 0, 0, 0);
                    mTab3.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_find_frd_normal));
                } else if (currIndex == 3) {
                    animation = new TranslateAnimation(three, 0, 0, 0);
                    mTab4.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_settings_normal));
                }
                break;
            case 1:
                mTab2.setImageDrawable(getResources().getDrawable(
                        R.drawable.tab_address_pressed));
                if (currIndex == 0) {
                    animation = new TranslateAnimation(zero, one, 0, 0);
                    mTab1.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_weixin_normal));
                } else if (currIndex == 2) {
                    animation = new TranslateAnimation(two, one, 0, 0);
                    mTab3.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_find_frd_normal));
                } else if (currIndex == 3) {
                    animation = new TranslateAnimation(three, one, 0, 0);
                    mTab4.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_settings_normal));
                }
                break;
            case 2:
                mTab3.setImageDrawable(getResources().getDrawable(
                        R.drawable.tab_find_frd_pressed));
```

```
                    if (currIndex == 0) {
                        animation = new TranslateAnimation(zero, two, 0, 0);
                        mTab1.setImageDrawable(getResources().getDrawable(
                                R.drawable.tab_weixin_normal));
                    } else if (currIndex == 1) {
                        animation = new TranslateAnimation(one, two, 0, 0);
                        mTab2.setImageDrawable(getResources().getDrawable(
                                R.drawable.tab_address_normal));
                    } else if (currIndex == 3) {
                        animation = new TranslateAnimation(three, two, 0, 0);
                        mTab4.setImageDrawable(getResources().getDrawable(
                                R.drawable.tab_settings_normal));
                    }
                    break;
                case 3:
                    mTab4.setImageDrawable(getResources().getDrawable(
                            R.drawable.tab_settings_pressed));
                    if (currIndex == 0) {
                        animation = new TranslateAnimation(zero, three, 0, 0);
                        mTab1.setImageDrawable(getResources().getDrawable(
                                R.drawable.tab_weixin_normal));
                    } else if (currIndex == 1) {
                        animation = new TranslateAnimation(one, three, 0, 0);
                        mTab2.setImageDrawable(getResources().getDrawable(
                                R.drawable.tab_address_normal));
                    } else if (currIndex == 2) {
                        animation = new TranslateAnimation(two, three, 0, 0);
                        mTab3.setImageDrawable(getResources().getDrawable(
                                R.drawable.tab_find_frd_normal));
                    }
                    break;
                }
                currIndex = arg0;
                animation.setFillAfter(true);// True:图片停在动画结束位置
                animation.setDuration(150);// 动画持续时间
                mTabImg.startAnimation(animation);// 开始动画
            }

            public void onPageScrolled(int arg0, float arg1, int arg2) {
            }

            public void onPageScrollStateChanged(int arg0) {
            }
        }
        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
        //获取 back 键
                if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount()== 0) {
```

```
            if (menu_display) {
                // 如果 Menu 已经打开，先关闭 Menu
                menuWindow.dismiss();
                menu_display = false;
            } else {
                Intent intent = new Intent();
                intent.setClass(MainWeixin.this, Exit.class);
                startActivity(intent);
            }
        }

        else if (keyCode == KeyEvent.KEYCODE_MENU) { // 获取 Menu 键
            if (!menu_display) {
                // 获取 LayoutInflater 实例
                inflater = (LayoutInflater) this
                        .getSystemService(LAYOUT_INFLATER_SERVICE);
                // 这里的 main 布局是在 inflate 中加入的
                // 该方法返回的是一个 View 的对象，是布局中的根
                layout = inflater.inflate(R.layout.main_menu, null);
                // 后两个参数是 width 和 height
                menuWindow = new PopupWindow(layout, LayoutParams.FILL_PARENT,
                        LayoutParams.WRAP_CONTENT);
                menuWindow.showAtLocation(this.findViewById(R.id.mainweixin),
                        // 设置 layout 在 PopupWindow 中显示的位置
                        Gravity.BOTTOM | Gravity.CENTER_HORIZONTAL, 0, 0);
                mClose = (LinearLayout) layout.findViewById(R.id.menu_close);
                mCloseBtn = (LinearLayout) layout
                        .findViewById(R.id.menu_close_btn);
                // 下面对每一个 Layout 进行点击事件的注册
                mCloseBtn.setOnClickListener(new View.OnClickListener() {
                    public void onClick(View arg0) {
                        Intent intent = new Intent();
                        intent.setClass(MainWeixin.this, Exit.class);
                        startActivity(intent);
                        // 响应点击事件之后关闭 Menu
                        menuWindow.dismiss();
                    }
                });
                menu_display = true;
            } else {
                // 如果当前已经为显示状态，则隐藏起来
                menuWindow.dismiss();
                menu_display = false;
            }
            return false;
        }
        return false;
    }
```

```
        // 设置标题栏右侧按钮的作用
        public void btnmainright(View v) {
            Intent intent = new Intent(MainWeixin.this, MainTopRightDialog.class);
            startActivity(intent);
        }
        public void startchat(View v) {
            Intent intent = new Intent(MainWeixin.this, ChatActivity.class);
            startActivity(intent);
        }
        public void exit_settings(View v) {
            Intent intent = new Intent(MainWeixin.this, ExitFromSettings.class);
            startActivity(intent);
        }
        // 手机摇一摇
        public void btn_shake(View v) {
            Intent intent = new Intent(MainWeixin.this, ShakeActivity.class);
            startActivity(intent);
        }
    }
```

16.3.3 系统登录界面

为了保证系统的安全，设置只有合法用户才能登录系统，为此专门设置了一个登录表单界面。具体 UI 界面如图 16-5 所示。

图 16-5　系统的登录表单界面

系统登录界面的布局文件是 login.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#eee"
    android:orientation="vertical"
    android:gravity="center_horizontal">
<RelativeLayout
        android:id="@+id/login_top_layout"
        android:layout_width="fill_parent"
        android:layout_height="45dp"
        android:layout_alignParentTop="true"
        android:background="@drawable/title_bar">
<Button
            android:id="@+id/login_reback_btn"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:text="返回"
            android:textSize="26sp"
            android:textColor="#fff"
            android:onClick="login_back"
            android:background="@drawable/title_btn_back"/>
<TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:textSize="20sp"
            android:textStyle="bold"
            android:textColor="#ffffff"
            android:text="登录"/>
</RelativeLayout>
    <EditText
        android:id="@+id/login_user_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/login_top_layout"
        android:textColor="#000"
        android:textSize="15sp"
        android:layout_marginTop="25dp"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:singleLine="true"
```

```
        android:background="@drawable/login_editbox"
        android:hint="QQ 号/微信号/手机号（请输入 buaa）"/>
    <EditText
        android:id="@+id/login_passwd_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/login_user_edit"
        android:textColor="#000"
        android:textSize="15sp"
        android:layout_marginTop="25dp"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:background="@drawable/login_editbox"
        android:password="true"
        android:singleLine="true"
        android:hint="密码(请输入 123)"/>
    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_below="@+id/login_passwd_edit">
    <Button
            android:id="@+id/forget_passwd"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="23dp"
            android:layout_marginTop="5dp"
            android:text="忘记密码?"
            android:textSize="16sp"
            android:textColor="#00f"
            android:background="#0000"
            android:onClick="login_pw" />
    <Button
            android:id="@+id/login_login_btn"
            android:layout_width="90dp"
            android:layout_height="40dp"
            android:layout_marginRight="20dp"
            android:layout_alignParentRight="true"
            android:text="登录"
            android:background="@drawable/btn_style_green"
            android:textColor="#ffffff"
            android:textSize="18sp"
            android:onClick="login_mainweixin"/>
    </RelativeLayout>
```

```
</RelativeLayout>
```

对应的实现文件是 login.java，具体代码如下。

```
package cn.buaa.myweixin;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
public class Login extends Activity {
    private EditText mUser; //  账号编辑框
    private EditText mPassword; //  密码编辑框
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        mUser = (EditText)findViewById(R.id.login_user_edit);
        mPassword = (EditText)findViewById(R.id.login_passwd_edit);
    }
    public void login_mainweixin(View v) {
     //判断账号和密码
     if("weixin".equals(mUser.getText().toString()) && "123".equals(mPassword.getText().toString()))
        {
            Intent intent = new Intent();
            intent.setClass(Login.this,LoadingActivity.class);
            startActivity(intent);
         }
          //判断账号和密码
        else if("".equals(mUser.getText().toString()) || "".equals(mPassword.getText().toString()))
     {
     new AlertDialog.Builder(Login.this)
                .setIcon(getResources().getDrawable(R.drawable.login_error_icon))
                .setTitle("登录错误")
                .setMessage("微信账号或者密码不能为空，\n 请输入后再登录！")
                .create().show();
         }
        else{
     new AlertDialog.Builder(Login.this)
                .setIcon(getResources().getDrawable(R.drawable.login_error_icon))
                .setTitle("登录失败")
```

```
                    .setMessage("微信账号或者密码不正确，\n 请检查后重新输入！")
                    .create().show();
                }
            }
            //标题栏返回按钮
        public void login_back(View v) {
         this.finish();
            }
            //忘记密码按钮
        public void login_pw(View v) {
         Uri uri = Uri.parse("http://3g.qq.com");
         Intent intent = new Intent(Intent.ACTION_VIEW, uri);
         startActivity(intent);
            }
    }
```

登录成功后调用文件 LoadingActivity.java 进入系统主界面，此文件的实现代码如下。

```
    package cn.buaa.myweixin;
    import android.os.Bundle;
    import android.os.Handler;
    import android.app.Activity;
    import android.content.Intent;
    import android.view.Menu;
    import android.view.WindowManager;
    import android.widget.Toast;
    public class LoadingActivity extends Activity{

        @Override
        public void onCreate(Bundle savedInstanceState) {
            // TODO Auto-generated method stub
            super.onCreate(savedInstanceState);
            setContentView(R.layout.loading);

            new Handler().postDelayed(new Runnable(){
                public void run(){
                    Intent intent = new Intent (LoadingActivity.this,Whatsnew.class);
                    startActivity(intent);
                    LoadingActivity.this.finish();
                    Toast.makeText(getApplicationContext(), "登录成功", Toast.LENGTH_SHORT).
show();
                }
            }, 200);
        }
    }
```

16.3.4　发送信息界面

为了达到在线交流目的，系统提供了发送信息界面，此界面和 QQ 聊天界面类似，呢狗狗调用输入法输入文本信息。具体 UI 界面如图 16-6 所示。

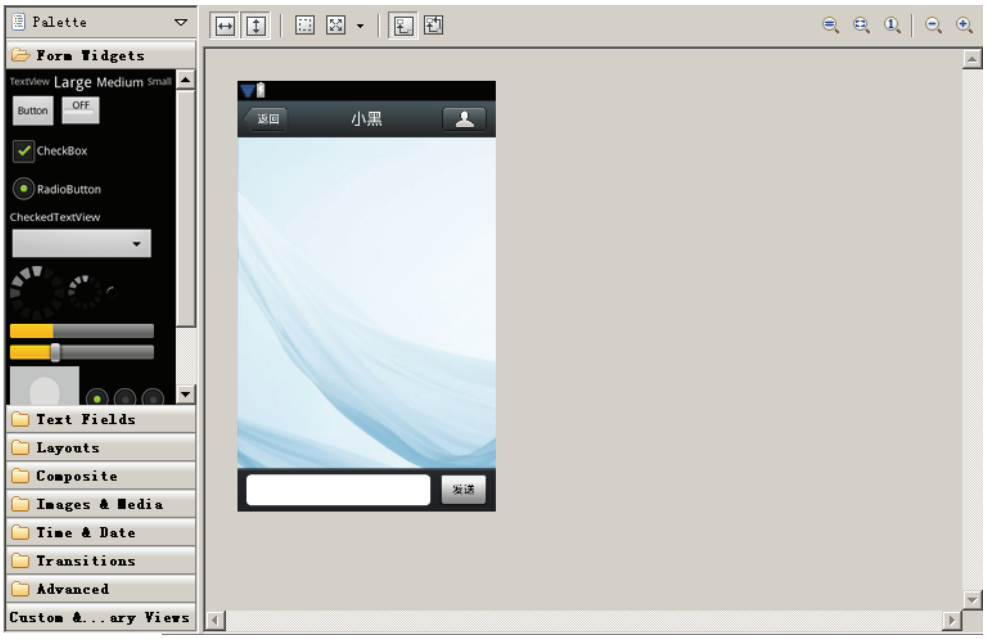


图 16-6　发送信息界面

发送信息界面的布局文件是 chat_xiaohei.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@drawable/chat_bg_default" >
<RelativeLayout
        android:id="@+id/rl_layout"
            android:layout_width="fill_parent"
            android:layout_height="45dp"
            android:background="@drawable/title_bar"
            android:gravity="center_vertical"   >
            <Button
          android:id="@+id/btn_back"
          android:layout_width="70dp"
          android:layout_height="wrap_content"
          android:layout_centerVertical="true"
          android:text="返回"
          android:textSize="26sp"
```

```
                android:textColor="#fff"
                android:onClick="chat_back"
                android:background="@drawable/title_btn_back"/>
            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="小黑"
            android:layout_centerInParent="true"
            android:textSize="20sp"
                    android:textColor="#ffffff" />
                <ImageButton
                    android:id="@+id/right_btn"
                    android:layout_width="67dp"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_centerVertical="true"
            android:layout_marginRight="5dp"
            android:src="@drawable/mm_title_btn_contact_normal"
                    android:background="@drawable/title_btn_right" />
            </RelativeLayout>
<RelativeLayout
        android:id="@+id/rl_bottom"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:background="@drawable/chat_footer_bg" >
<Button
        android:id="@+id/btn_send"
        android:layout_width="60dp"
        android:layout_height="40dp"
        android:layout_alignParentRight="true"
        android:layout_marginRight="10dp"
        android:layout_centerVertical="true"
        android:text="发送"
        android:background="@drawable/chat_send_btn" />
<EditText
        android:id="@+id/et_sendmessage"
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:layout_toLeftOf="@id/btn_send"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:background="@drawable/login_edit_normal"
        android:layout_centerVertical="true"
        android:singleLine="true"
```

```
            android:textSize="18sp"/>
    </RelativeLayout>
    <ListView
        android:id="@+id/listview"
        android:layout_below="@id/rl_layout"
        android:layout_above="@id/rl_bottom"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
         android:divider="@null"
        android:dividerHeight="5dp"
        android:stackFromBottom="true"
        android:scrollbarStyle="outsideOverlay"
        android:cacheColorHint="#0000"/>
    </RelativeLayout>
```

对应的实现文件是 ChatActivity.java，具体代码如下。

```
    package cn.buaa.myweixin;

    import java.util.ArrayList;
    import java.util.Calendar;
    import java.util.List;
    import android.app.Activity;
    import android.content.Intent;
    import android.graphics.drawable.LevelListDrawable;
    import android.os.Bundle;
    import android.text.Editable;
    import android.view.View;
    import android.view.View.OnClickListener;
    import android.view.WindowManager;
    import android.widget.Button;
    import android.widget.EditText;
    import android.widget.ListView;
    public class ChatActivity extends Activity implements OnClickListener{
        private Button mBtnSend;
        private Button mBtnBack;
        private EditText mEditTextContent;
        private ListView mListView;
        private ChatMsgViewAdapter mAdapter;
        private List<ChatMsgEntity> mDataArrays = new ArrayList<ChatMsgEntity>();
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.chat_xiaohei);
            //启动 Activity 时不自动弹出软键盘
            getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_
            ALWAYS_HIDDEN);
```

```
        initView();
        initData();
    }
    public void initView()
    {
        mListView = (ListView) findViewById(R.id.listview);
        mBtnSend = (Button) findViewById(R.id.btn_send);
        mBtnSend.setOnClickListener(this);
        mBtnBack = (Button) findViewById(R.id.btn_back);
        mBtnBack.setOnClickListener(this);
        mEditTextContent = (EditText) findViewById(R.id.et_sendmessage);
    }
    private String[]msgArray = new String[]{"有大", "有！？", "我也有", "那上吧",
                                            "打啊！你放大啊", "你不？留人头那！。",
                                            "不解释", "....",};

    private String[]dataArray = new String[]{"2012-09-01 18:00", "2012-09-01 18:10",
                                            "2012-09-01 18:11", "2012-09-01 18:20",
                                            "2012-09-01 18:30", "2012-09-01 18:35",
                                            "2012-09-01 18:40", "2012-09-01 18:50"};
    private final static int COUNT = 8;
    public void initData()
    {
        for(int i = 0; i < COUNT; i++)
        {
            ChatMsgEntity entity = new ChatMsgEntity();
            entity.setDate(dataArray[i]);
            if (i % 2 == 0)
            {
                entity.setName("小黑");
                entity.setMsgType(true);
            }else{
                entity.setName("人马");
                entity.setMsgType(false);
            }
            entity.setText(msgArray[i]);
            mDataArrays.add(entity);
        }
     mAdapter = new ChatMsgViewAdapter(this, mDataArrays);
        mListView.setAdapter(mAdapter);
    }
     public void onClick(View v) {
        // TODO Auto-generated method stub
        switch(v.getId())
        {
            case R.id.btn_send:
```

```
                send();
                break;
            case R.id.btn_back:
                finish();
                break;
        }
    }
    private void send()
    {
        String contString = mEditTextContent.getText().toString();
        if (contString.length() > 0)
        {
            ChatMsgEntity entity = new ChatMsgEntity();
            entity.setDate(getDate());
            entity.setName("人马");
            entity.setMsgType(false);
            entity.setText(contString);
            mDataArrays.add(entity);
            mAdapter.notifyDataSetChanged();
            mEditTextContent.setText("");
            mListView.setSelection(mListView.getCount() - 1);
        }
    }
    private String getDate() {
        Calendar c = Calendar.getInstance();
        String year = String.valueOf(c.get(Calendar.YEAR));
        String month = String.valueOf(c.get(Calendar.MONTH));
        String day = String.valueOf(c.get(Calendar.DAY_OF_MONTH) + 1);
        String hour = String.valueOf(c.get(Calendar.HOUR_OF_DAY));
        String mins = String.valueOf(c.get(Calendar.MINUTE));
        StringBuffer sbBuffer = new StringBuffer();
        sbBuffer.append(year + "-" + month + "-" + day + " " + hour + ":" + mins);
        return sbBuffer.toString();
    }
    public void head_xiaohei(View v) {        //标题栏返回按钮
     Intent intent = new Intent (ChatActivity.this,InfoXiaohei.class);
        startActivity(intent);
    }
}
```

16.3.5　摇一摇界面

“摇一摇”是微信的特色功能，通过摇动手机的方式可以实现一个操作功能，例如发送一幅图片，查找到一个好友等。具体 UI 界面如图 16-7 所示。

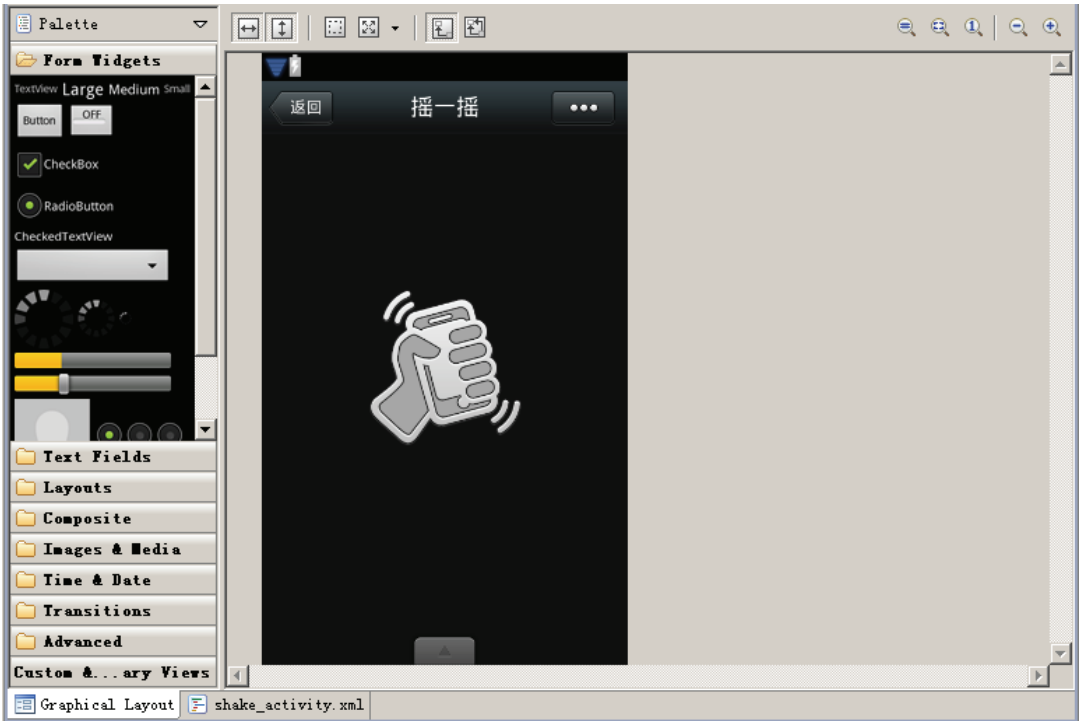


图 16-7　摇一摇设计界面

摇一摇设计界面的布局文件是 shake_activity.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="#111">
<RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_centerInParent="true" >
<ImageView
        android:id="@+id/shakeBg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@drawable/shakehideimg_man2" />
<LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:orientation="vertical" >
```

```
<RelativeLayout
        android:id="@+id/shakeImgUp"
        android:layout_width="fill_parent"
        android:layout_height="190dp"
        android:background="#111">
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:src="@drawable/shake_logo_up" />
</RelativeLayout>
<RelativeLayout
        android:id="@+id/shakeImgDown"
        android:layout_width="fill_parent"
        android:layout_height="190dp"
        android:background="#111">
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:src="@drawable/shake_logo_down" />
</RelativeLayout>
</LinearLayout>
</RelativeLayout>
<RelativeLayout
        android:id="@+id/shake_title_bar"
        android:layout_width="fill_parent"
        android:layout_height="45dp"
        android:background="@drawable/title_bar"
        android:gravity="center_vertical"    >
<Button
    android:layout_width="70dp"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:text="返回"
    android:textSize="26sp"
    android:textColor="#fff"
    android:onClick="shake_activity_back"
    android:background="@drawable/title_btn_back"/>
        <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="摇一摇"
        android:layout_centerInParent="true"
        android:textSize="20sp"
```

```
            android:textColor="#ffffff" />
    <ImageButton
            android:layout_width="67dp"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_centerVertical="true"
            android:layout_marginRight="5dp"
            android:src="@drawable/mm_title_btn_menu"
            android:background="@drawable/title_btn_right"
            android:onClick="linshi"/>
        </RelativeLayout>
    <SlidingDrawer
            android:id="@+id/slidingDrawer1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:content="@+id/content"
            android:handle="@+id/handle" >
    <Button
                android:id="@+id/handle"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@drawable/shake_report_dragger_up" />
    <LinearLayout
                android:id="@+id/content"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:background="#f9f9f9" >
    <ImageView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:scaleType="fitXY"
                    android:src="@drawable/shake_line_up" />
    </LinearLayout>
    </SlidingDrawer>
    </RelativeLayout>
```

对应的实现文件是 ShakeActivity.java，具体代码如下。

```
    public class ShakeActivity extends Activity{
        ShakeListener mShakeListener = null;
        Vibrator mVibrator;
        private RelativeLayout mImgUp;
        private RelativeLayout mImgDn;
        private RelativeLayout mTitle;
        private SlidingDrawer mDrawer;
        private Button mDrawerBtn;
        @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.shake_activity);
        mVibrator = (Vibrator)getApplication().getSystemService(VIBRATOR_SERVICE);
        mImgUp = (RelativeLayout) findViewById(R.id.shakeImgUp);
        mImgDn = (RelativeLayout) findViewById(R.id.shakeImgDown);
        mTitle = (RelativeLayout) findViewById(R.id.shake_title_bar);
        mDrawer = (SlidingDrawer) findViewById(R.id.slidingDrawer1);
        mDrawerBtn = (Button) findViewById(R.id.handle);
        mDrawer.setOnDrawerOpenListener(new OnDrawerOpenListener()
        {   public void onDrawerOpened()
            {
    mDrawerBtn.setBackgroundDrawable(getResources().getDrawable(R.drawable.shake_report_drag
    ger_down));
                TranslateAnimation titleup = new
TranslateAnimation(Animation.RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,0f,Animation.RELAT
IVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,-1.0f);
                titleup.setDuration(200);
                titleup.setFillAfter(true);
                mTitle.startAnimation(titleup);
            }
        });
         /* 设定 SlidingDrawer 被关闭的事件处理 */
        mDrawer.setOnDrawerCloseListener(new OnDrawerCloseListener()
        {   public void onDrawerClosed()
            {
    mDrawerBtn.setBackgroundDrawable(getResources().getDrawable(R.drawable.shake_report_drag
    ger_up));
                TranslateAnimation titledn = new
TranslateAnimation(Animation.RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,0f,Animation.RELAT
IVE_TO_SELF,-1.0f,Animation.RELATIVE_TO_SELF,0f);
                titledn.setDuration(200);
                titledn.setFillAfter(false);
                mTitle.startAnimation(titledn);
            }
        });
        mShakeListener = new ShakeListener(this);
      mShakeListener.setOnShakeListener(new OnShakeListener() {
            public void onShake() {
                startAnim();   //开始摇一摇手掌动画
                mShakeListener.stop();
                startVibrato(); //开始振动
                new Handler().postDelayed(new Runnable(){
                    public void run(){
                        Toast mtoast;
```

```
                                        mtoast = Toast.makeText(getApplicationContext(),
                                                "抱歉，暂时没有找到\n 在同一时刻摇一摇的人。\n
再试一次吧！", 10);
                                                mtoast.show();
                                                mVibrator.cancel();
                                                mShakeListener.start();
                                        }
                                    }, 2000);
                                }
                            });
                    }
                    //定义摇一摇动画
                    public void startAnim () {
                        AnimationSet animup = new AnimationSet(true);
                        TranslateAnimation mytranslateanimup0 = new TranslateAnimation
                        (Animation.RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,0f,Animation.
                        RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,-0.5f);
                        mytranslateanimup0.setDuration(1000);
                        TranslateAnimation mytranslateanimup1 = new TranslateAnimation
                        (Animation.RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,0f,Animation.
                        RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,+0.5f);
                        mytranslateanimup1.setDuration(1000);
                        mytranslateanimup1.setStartOffset(1000);
                        animup.addAnimation(mytranslateanimup0);
                        animup.addAnimation(mytranslateanimup1);
                        mImgUp.startAnimation(animup);
                        AnimationSet animdn = new AnimationSet(true);
                        TranslateAnimation mytranslateanimdn0 = new TranslateAnimation
                        (Animation.RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,0f,Animation.
                        RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,+0.5f);
                        mytranslateanimdn0.setDuration(1000);
                        TranslateAnimation mytranslateanimdn1 = new TranslateAnimation
                        (Animation.RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,0f,Animation.
                        RELATIVE_TO_SELF,0f,Animation.RELATIVE_TO_SELF,-0.5f);
                        mytranslateanimdn1.setDuration(1000);
                        mytranslateanimdn1.setStartOffset(1000);
                        animdn.addAnimation(mytranslateanimdn0);
                        animdn.addAnimation(mytranslateanimdn1);
                        mImgDn.startAnimation(animdn);
                    }
                    //定义振动
                    public void startVibrato(){
                        mVibrator.vibrate( new long[]{500,200,500,200}, -1); //第一个｛｝里面是节奏数组，第二
个参数是重复次数，-1 为不重复，非-1 为从 pattern 的指定下标开始重复
                    }
                    public void shake_activity_back(View v) {        //标题栏返回按钮
```

```
        this.finish();
          }
        public void linshi(View v) {        //标题栏
              startAnim();
          }
        @Override
        protected void onDestroy() {
              super.onDestroy();
              if (mShakeListener != null) {
                    mShakeListener.stop();
              }
        }
}
```

文件 ShakeListener.java 的功能是通过重力感应器实现重力监听，这是实现“摇一摇”功能的基础。文件 ShakeListener.java 的具体实现代码如下。

```
package cn.buaa.myweixin;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
/**
 * 一个检测手机摇晃的监听器
 */
public class ShakeListener implements SensorEventListener {
      // 速度阈值，当摇晃速度达到该值后产生作用
      private static final int SPEED_SHRESHOLD = 3000;
      // 两次检测的时间间隔
      private static final int UPTATE_INTERVAL_TIME = 70;
      // 传感器管理器
      private SensorManager sensorManager;
      // 传感器
      private Sensor sensor;
      // 重力感应监听器
      private OnShakeListener onShakeListener;
      // 上下文
      private Context mContext;
      // 手机上一个位置时重力感应坐标
      private float lastX;
      private float lastY;
      private float lastZ;
      // 上次检测时间
      private long lastUpdateTime;
      // 构造器
```

```
public ShakeListener(Context c) {
        // 获得监听对象
        mContext = c;
        start();
}
// 开始
public void start() {
        // 获得传感器管理器
        sensorManager = (SensorManager) mContext
                        .getSystemService(Context.SENSOR_SERVICE);
        if (sensorManager != null) {
                // 获得重力传感器
                sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        }
        // 注册
        if (sensor != null) {
                sensorManager.registerListener(this, sensor,
                                SensorManager.SENSOR_DELAY_GAME);
        }
}
// 停止检测
public void stop() {
        sensorManager.unregisterListener(this);
}
// 设置重力感应监听器
public void setOnShakeListener(OnShakeListener listener) {
        onShakeListener = listener;
}
// 重力感应器感应获得变化数据
public void onSensorChanged(SensorEvent event) {
        // 现在检测时间
        long currentUpdateTime = System.currentTimeMillis();
        // 两次检测的时间间隔
        long timeInterval = currentUpdateTime - lastUpdateTime;
        // 判断是否达到了检测时间间隔
        if (timeInterval < UPTATE_INTERVAL_TIME)
                return;
        // 现在的时间变成 last 时间
        lastUpdateTime = currentUpdateTime;
        // 获得 x,y,z 坐标
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];
        // 获得 x,y,z 的变化值
        float deltaX = x - lastX;
        float deltaY = y - lastY;
```

```
                float deltaZ = z - lastZ;
                // 将现在的坐标变成 last 坐标
                lastX = x;
                lastY = y;
                lastZ = z;
                double speed = Math.sqrt(deltaX * deltaX + deltaY * deltaY + deltaZ
                        * deltaZ)
                        / timeInterval * 10000;
                Log.v("thelog", "==========log==================");
                // 达到速度阀值，发出提示
                if (speed >= SPEED_SHRESHOLD) {
                    onShakeListener.onShake();
                }
            }
            public void onAccuracyChanged(Sensor sensor, int accuracy) {
            }
            // 摇晃监听接口
            public interface OnShakeListener {
                public void onShake();
            }
        }
```

第四篇　综合实战篇

第 17 章　开发仿陌陌交友系统

交友是人们为了摆脱自己单身的生活，而去结交认识他人的过程。交友的类型可以是女朋友或者男朋友，也可以是普通朋友。在本章的内容中，将详细讲解在 Android 系统中开发一款仿陌陌系统的交友软件，为读者掌握 Android 应用开发的核心技术打下基础。

17.1　陌陌介绍

陌陌是一款基于地理位置的移动社交工具，可以通过陌陌认识周围范围内的陌生人，查看对方的个人信息和位置，免费发送短信、语音、照片以及精准的地理位置。陌陌专注于移动互联网，专注于移动社交，专注于社交模式探索并满足人们的社交愿望。公司于 2011 年 3 月份成立。

17.1.1　陌陌发展现状

陌陌是陌陌科技开发的首个基于 iPhone 和 Android、Windows Phone 的手机应用，有别于微信、微博、QQ、YY、MSN、群群、遇见等手机社交软件。通过陌陌可以提供真实的位置信息，解决了以往社交软件过于虚幻，缺乏真实的线下互动的问题。2011 年 8 月 3 日，陌陌 iOS 版正式上线。

2013 年 4 月 24 日，在由艾瑞咨询举办的 2012～2013 中国移动互联网应用评选活动上，陌陌获得中国移动互联网应用年度最具创新力大奖。13 年 4 月 15 日，陌陌 3.4 版本上线。新增附近群组搜索，创建好友间多人对话，微博好友推荐功能。

2014 年 12 月 12 日，陌陌科技登陆纳斯达克。

17.1.2　陌陌特点介绍

（1）社交模式

根据 GPS 搜寻和定位身边的陌生人和群组，高效快捷的建立联系，节省沟通的距离成本。

（2）免费传递

可以方便的通过陌陌免费发送短信、语音、照片以及精准的地理位置，与他人进行各种互动。

（3）递送提示

即时了解信息送达的状态，“送达、已读”等提示能让用户及时掌握信息是否被对方看到。

（4）个人资料

可以在资料页存放八张照片，以及签名、职业、爱好等信息，以增进其他人对用户的了解。

（5）场景表情

表情商店提供丰富的表情，让聊天不再单调，更加的生动活泼，符合移动社交的聊天习惯。

（6）会员服务

可享受陌陌不断推出的各种增值及专属服务，包括基础会员服务、上限提升服务、表情商店服务等。

（7）隐私保护

可以随时把厌恶的人拉入黑名单，还可以对他人的不良行为进行举报，并且有多种隐身模式。

（8）平台支持

全面支持多种 iOS 设备，以及 Android 2.3 及以上版本的手机，支持各种网络接入方式。

17.2　实现系统欢迎界面

运行本陌陌系统后，将首先显示一个系统欢迎界面，以一副图片作为背景，下方显示“注册”和“登录”按钮，如图 17-1 所示。

图 17-1　系统欢迎界面

在本节的内容中，将详细讲解系统欢迎界面的具体实现过程。

17.2.1 欢迎界面布局

本系统欢迎界面 Activity 是.activity.WelcomeActivity，对应界面布局文件是 activity_welcome.xml，功能是通过 ImageView 控件显示背景图片，且在界面下方通过两个 Button 控件显示“注册”和“登录”按钮，具体实现代码如下。

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/pic_index_background"
    android:orientation="vertical" >
    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true" >
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:scaleType="center"
            android:src="@drawable/pic_index_logo" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_alignParentTop="true"
            android:scaleType="center"
            android:visibility="gone" />
    </RelativeLayout>
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:scaleType="center"
        android:src="@drawable/pic_index_copyright" />
    <LinearLayout
        android:id="@+id/welcome_linear_ctrlbar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:background="@drawable/bg_welcome_ctrlbar"
        android:gravity="center_horizontal|bottom"
        android:orientation="vertical"
        android:paddingBottom="15dip"
        android:paddingLeft="5dip"
```

```
android:paddingRight="5dip"
android:paddingTop="13dip" >
<LinearLayout
    android:id="@+id/welcome_linear_avatars"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal" >
    <include
        android:id="@+id/welcome_include_member_avatar_block0"
        android:layout_weight="1"
        layout="@layout/include_welcome_item" />
    <include
        android:id="@+id/welcome_include_member_avatar_block1"
        android:layout_weight="1"
        layout="@layout/include_welcome_item" />
    <include
        android:id="@+id/welcome_include_member_avatar_block2"
        android:layout_weight="1"
        layout="@layout/include_welcome_item" />
    <include
        android:id="@+id/welcome_include_member_avatar_block3"
        android:layout_weight="1"
        layout="@layout/include_welcome_item" />
    <include
        android:id="@+id/welcome_include_member_avatar_block4"
        android:layout_weight="1"
        layout="@layout/include_welcome_item" />
    <include
        android:id="@+id/welcome_include_member_avatar_block5"
        android:layout_weight="1"
        layout="@layout/include_welcome_item" />
</LinearLayout>
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal"
    android:visibility="invisible" >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:src="@drawable/ic_index_totaluser" />
    <com.immomo.momo.android.view.HandyTextView
```

```
                android:id="@+id/welcome_htv_usercount"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom"
                android:layout_marginLeft="5dip"
                android:layout_marginRight="5dip"
                android:text="0"
                android:textColor="#FFFFFFFF"
                android:textSize="18sp" />
            <com.immomo.momo.android.view.HandyTextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom"
                android:text="位用户在你身边"
                android:textColor="#FFFFFFFF"
                android:textSize="13sp"
                android:textStyle="bold" />
        </LinearLayout>
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal" >
            <Button
                android:id="@+id/welcome_btn_register"
                android:layout_width="100dip"
                android:layout_height="40dip"
                android:layout_margin="5dip"
                android:background="@drawable/btn_default_blue"
                android:text="注册"
                android:textColor="#FFFFFFFF" />
            <Button
                android:id="@+id/welcome_btn_login"
                android:layout_width="100dip"
                android:layout_height="40dip"
                android:layout_margin="5dip"
                android:background="@drawable/btn_default_white"
                android:text="登录"
                android:textColor="#ff465579" />
            <ImageButton
                android:id="@+id/welcome_ibtn_about"
                android:layout_width="wrap_content"
                android:layout_height="40dip"
                android:layout_margin="5dip"
                android:layout_marginLeft="10dip"
```

```
                android:background="@drawable/btn_default_white"
                android:src="@drawable/ic_welcome_about_normal" />
        </LinearLayout>
    </LinearLayout>
```

17.2.2　欢迎界面 Activity

欢迎界面 Activity 的实现文件是 WelcomeActivity.java，功能是监听用户单击屏幕操作，根据用户单击的图标或按钮跳转到注册界面、登录界面或帮助界面。文件 WelcomeActivity.java 的具体实现代码如下。

```
public class WelcomeActivity extends BaseActivity implements OnClickListener {
    private LinearLayout mLinearCtrlbar;
    private LinearLayout mLinearAvatars;
    private Button mBtnRegister;
    private Button mBtnLogin;
    private ImageButton mIbtnAbout;
    private View[] mMemberBlocks;
    private String[] mAvatars = new String[] { "welcome_0", "welcome_1",
            "welcome_2", "welcome_3", "welcome_4", "welcome_5" };
    private String[] mDistances = new String[] { "0.84km", "1.02km", "1.34km",
            "1.88km", "2.50km", "2.78km" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_welcome);
        initViews();
        initEvents();
        initAvatarsItem();
        showWelcomeAnimation();
    }
    @Override
    protected void initViews() {
        mLinearCtrlbar = (LinearLayout) findViewById(R.id.welcome_linear_ctrlbar);
        mLinearAvatars = (LinearLayout) findViewById(R.id.welcome_linear_avatars);
        mBtnRegister = (Button) findViewById(R.id.welcome_btn_register);
        mBtnLogin = (Button) findViewById(R.id.welcome_btn_login);
        mIbtnAbout = (ImageButton) findViewById(R.id.welcome_ibtn_about);
    }
    @Override
    protected void initEvents() {
        mBtnRegister.setOnClickListener(this);
        mBtnLogin.setOnClickListener(this);
        mIbtnAbout.setOnClickListener(this);
    }
    private void initAvatarsItem() {
```

```
        initMemberBlocks();
        for (int i = 0; i < mMemberBlocks.length; i++) {
            ((ImageView) mMemberBlocks[i]
                    .findViewById(R.id.welcome_item_iv_avatar))
                    .setImageBitmap(mApplication.getAvatar(mAvatars[i]));
            ((HandyTextView) mMemberBlocks[i]
                    .findViewById(R.id.welcome_item_htv_distance))
                    .setText(mDistances[i]);
        }
    }
    private void initMemberBlocks() {
        mMemberBlocks = new View[6];
        mMemberBlocks[0] = findViewById(R.id.welcome_include_member_avatar_block0);
        mMemberBlocks[1] = findViewById(R.id.welcome_include_member_avatar_block1);
        mMemberBlocks[2] = findViewById(R.id.welcome_include_member_avatar_block2);
        mMemberBlocks[3] = findViewById(R.id.welcome_include_member_avatar_block3);
        mMemberBlocks[4] = findViewById(R.id.welcome_include_member_avatar_block4);
        mMemberBlocks[5] = findViewById(R.id.welcome_include_member_avatar_block5);
        int margin = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, 4, getResources()
                        .getDisplayMetrics());
        int widthAndHeight = (mScreenWidth - margin * 12) / 6;
        for (int i = 0; i < mMemberBlocks.length; i++) {
            ViewGroup.LayoutParams params = mMemberBlocks[i].findViewById(
                    R.id.welcome_item_iv_avatar).getLayoutParams();
            params.width = widthAndHeight;
            params.height = widthAndHeight;
            mMemberBlocks[i].findViewById(R.id.welcome_item_iv_avatar)
                    .setLayoutParams(params);
        }
        mLinearAvatars.invalidate();
    }
    private void showWelcomeAnimation() {
        Animation animation = AnimationUtils.loadAnimation(
                WelcomeActivity.this, R.anim.welcome_ctrlbar_slideup);
        animation.setAnimationListener(new AnimationListener() {
            @Override
            public void onAnimationStart(Animation animation) {
                mLinearAvatars.setVisibility(View.GONE);
            }
            @Override
            public void onAnimationRepeat(Animation animation) {
            }
            @Override
            public void onAnimationEnd(Animation animation) {
                new Handler().postDelayed(new Runnable() {
                    @Override
                    public void run() {
```

```
                                        mLinearAvatars.setVisibility(View.VISIBLE);
                                    }
                                }, 800);
                            }
                        });
                        mLinearCtrlbar.startAnimation(animation);
                    }
                    @Override
                    public void onClick(View v) {
                        switch (v.getId()) {
                        case R.id.welcome_btn_register:
                            startActivity(RegisterActivity.class);
                            break;
                        case R.id.welcome_btn_login:
                            startActivity(LoginActivity.class);
                            break;
                        case R.id.welcome_ibtn_about:
                            startActivity(AboutTabsActivity.class);
                            break;
                        }
                    }
                }
```

17.3　实现系统注册界面

当在欢迎界面单击“注册”按钮后会跳转到系统注册界面，如图 17-2 所示。

图 17-2　系统注册界面

在本节的内容中，将详细讲解系统注册界面的具体实现过程。

17.3.1 注册界面布局

在系统注册界面中的布局文件是 activity_register.xml，功能是在上方显示注册表单供用户输入 11 位手机号码，在下方显示“返回”和“下一步”按钮。文件 activity_register.xml 的具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/background_normal"
    android:orientation="vertical" >
    <include
        android:id="@+id/reg_header"
        layout="@layout/include_header" />
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_below="@+id/reg_header"
        android:orientation="vertical" >
        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_weight="1"
            android:orientation="vertical" >
            <ViewFlipper
                android:id="@+id/reg_vf_viewflipper"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:flipInterval="1000"
                android:persistentDrawingCache="animation" >
                <include
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    layout="@layout/include_register_phone" />
                <include
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    layout="@layout/include_register_verify" />
                <include
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    layout="@layout/include_register_setpwd" />
```

```
            <include
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                layout="@layout/include_register_baseinfo" />
            <include
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                layout="@layout/include_register_birthday" />
            <include
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                layout="@layout/include_register_photo" />
        </ViewFlipper>
    </LinearLayout>
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/bg_unlogin_bar"
        android:gravity="center_vertical"
        android:orientation="horizontal"
        android:paddingBottom="4dip"
        android:paddingLeft="8dip"
        android:paddingRight="8dip"
        android:paddingTop="4dip" >
        <Button
            android:id="@+id/reg_btn_previous"
            android:layout_width="wrap_content"
            android:layout_height="42dip"
            android:layout_marginRight="9dip"
            android:layout_weight="1"
            android:background="@drawable/btn_bottombar"
            android:gravity="center"
            android:textColor="@color/profile_bottom_text_color"
            android:textSize="14sp" />
        <Button
            android:id="@+id/reg_btn_next"
            android:layout_width="wrap_content"
            android:layout_height="42dip"
            android:layout_marginLeft="9dip"
            android:layout_weight="1"
            android:background="@drawable/btn_bottombar"
            android:gravity="center"
            android:textColor="@color/profile_bottom_text_color"
            android:textSize="14sp" />
    </LinearLayout>
```

```
    </LinearLayout>
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/reg_header"
        android:background="@drawable/bg_topbar_shadow"
        android:focusable="true" />

</RelativeLayout>
```

17.3.2 注册界面 Activity

注册界面 Activity 的实现文件是 RegisterActivity.java，功能是监听用户单击屏幕操作，根据用户在表单中输入的注册信息进行验证。本系统设置的合法手机号码是“12345678901”，如果输入其他号码会输出“已经注册的提示”。文件 RegisterActivity.java 的具体实现代码如下。

```
public class RegisterActivity extends BaseActivity implements OnClickListener,
        onNextActionListener {
    private HeaderLayout mHeaderLayout;
    private ViewFlipper mVfFlipper;
    private Button mBtnPrevious;
    private Button mBtnNext;
    private BaseDialog mBackDialog;
    private RegisterStep mCurrentStep;
    private StepPhone mStepPhone;
    private StepVerify mStepVerify;
    private StepSetPassword mStepSetPassword;
    private StepBaseInfo mStepBaseInfo;
    private StepBirthday mStepBirthday;
    private StepPhoto mStepPhoto;
    private int mCurrentStepIndex = 1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);
        initViews();
        mCurrentStep = initStep();
        initEvents();
        initBackDialog();
    }
    @Override
    protected void onDestroy() {
        PhotoUtils.deleteImageFile();
        super.onDestroy();
    }
    @Override
    protected void initViews() {
```

```
        mHeaderLayout = (HeaderLayout) findViewById(R.id.reg_header);
        mHeaderLayout.init(HeaderStyle.TITLE_RIGHT_TEXT);
        mVfFlipper = (ViewFlipper) findViewById(R.id.reg_vf_viewflipper);
        mVfFlipper.setDisplayedChild(0);
        mBtnPrevious = (Button) findViewById(R.id.reg_btn_previous);
        mBtnNext = (Button) findViewById(R.id.reg_btn_next);
    }
    @Override
    protected void initEvents() {
        mCurrentStep.setOnNextActionListener(this);
        mBtnPrevious.setOnClickListener(this);
        mBtnNext.setOnClickListener(this);
    }
    @Override
    public void onBackPressed() {
        if (mCurrentStepIndex <= 1) {
            mBackDialog.show();
        } else {
            doPrevious();
        }
    }
    @Override
    public void onClick(View arg0) {
        switch (arg0.getId()) {
        case R.id.reg_btn_previous:
            if (mCurrentStepIndex <= 1) {
                mBackDialog.show();
            } else {
                doPrevious();
            }
            break;
        case R.id.reg_btn_next:
            if (mCurrentStepIndex < 6) {
                doNext();
            } else {
                if (mCurrentStep.validate()) {
                    mCurrentStep.doNext();
                }
            }
            break;
        }
    }
    @SuppressWarnings("deprecation")
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
```

```
switch (requestCode) {
case PhotoUtils.INTENT_REQUEST_CODE_ALBUM:
    if (data == null) {
        return;
    }
    if (resultCode == RESULT_OK) {
        if (data.getData()== null) {
            return;
        }
        if (!FileUtils.isSdcardExist()) {
            showCustomToast("SD 卡不可用,请检查");
            return;
        }
        Uri uri = data.getData();
        String[] proj = { MediaStore.Images.Media.DATA };
        Cursor cursor = managedQuery(uri, proj, null, null, null);
        if (cursor != null) {
            int column_index = cursor
                    .getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
            if (cursor.getCount() > 0 && cursor.moveToFirst()) {
                String path = cursor.getString(column_index);
                Bitmap bitmap = BitmapFactory.decodeFile(path);
                if (PhotoUtils.bitmapIsLarge(bitmap)) {
                    PhotoUtils.cropPhoto(this, this, path);
                } else {
                    mStepPhoto.setUserPhoto(bitmap);
                }
            }
        }
    }
    break;
case PhotoUtils.INTENT_REQUEST_CODE_CAMERA:
    if (resultCode == RESULT_OK) {
        String path = mStepPhoto.getTakePicturePath();
        Bitmap bitmap = BitmapFactory.decodeFile(path);
        if (PhotoUtils.bitmapIsLarge(bitmap)) {
            PhotoUtils.cropPhoto(this, this, path);
        } else {
            mStepPhoto.setUserPhoto(bitmap);
        }
    }
    break;
case PhotoUtils.INTENT_REQUEST_CODE_CROP:
    if (resultCode == RESULT_OK) {
        String path = data.getStringExtra("path");
        if (path != null) {
```

```
                        Bitmap bitmap = BitmapFactory.decodeFile(path);
                        if (bitmap != null) {
                            mStepPhoto.setUserPhoto(bitmap);
                        }
                    }
                }
                break;
            }
        }
        @Override
        public void next() {
            mCurrentStepIndex++;
            mCurrentStep = initStep();
            mCurrentStep.setOnNextActionListener(this);
            mVfFlipper.setInAnimation(this, R.anim.push_left_in);
            mVfFlipper.setOutAnimation(this, R.anim.push_left_out);
            mVfFlipper.showNext();
        }
        private RegisterStep initStep() {
            switch (mCurrentStepIndex) {
            case 1:
                if (mStepPhone == null) {
                    mStepPhone = new StepPhone(this, mVfFlipper.getChildAt(0));
                }
                mHeaderLayout.setTitleRightText("注册新账号", null, "1/6");
                mBtnPrevious.setText("返　　回");
                mBtnNext.setText("下一步");
                return mStepPhone;
            case 2:
                if (mStepVerify == null) {
                    mStepVerify = new StepVerify(this, mVfFlipper.getChildAt(1));
                }
                mHeaderLayout.setTitleRightText("填写验证码", null, "2/6");
                mBtnPrevious.setText("上一步");
                mBtnNext.setText("下一步");
                return mStepVerify;
            case 3:
                if (mStepSetPassword == null) {
                    mStepSetPassword = new StepSetPassword(this,
                            mVfFlipper.getChildAt(2));
                }
                mHeaderLayout.setTitleRightText("设置密码", null, "3/6");
                mBtnPrevious.setText("上一步");
                mBtnNext.setText("下一步");
                return mStepSetPassword;
            case 4:
```

```
                if (mStepBaseInfo == null) {
                    mStepBaseInfo = new StepBaseInfo(this, mVfFlipper.getChildAt(3));
                }
                mHeaderLayout.setTitleRightText("填写基本资料", null, "4/6");
                mBtnPrevious.setText("上一步");
                mBtnNext.setText("下一步");
                return mStepBaseInfo;
            case 5:
                if (mStepBirthday == null) {
                    mStepBirthday = new StepBirthday(this, mVfFlipper.getChildAt(4));
                }
                mHeaderLayout.setTitleRightText("您的生日", null, "5/6");
                mBtnPrevious.setText("上一步");
                mBtnNext.setText("下一步");
                return mStepBirthday;
            case 6:
                if (mStepPhoto == null) {
                    mStepPhoto = new StepPhoto(this, mVfFlipper.getChildAt(5));
                }
                mHeaderLayout.setTitleRightText("设置头像", null, "6/6");
                mBtnPrevious.setText("上一步");
                mBtnNext.setText("注      册");
                return mStepPhoto;
            }
            return null;
        }
        private void doPrevious() {
            mCurrentStepIndex--;
            mCurrentStep = initStep();
            mCurrentStep.setOnNextActionListener(this);
            mVfFlipper.setInAnimation(this, R.anim.push_right_in);
            mVfFlipper.setOutAnimation(this, R.anim.push_right_out);
            mVfFlipper.showPrevious();
        }
        private void doNext() {
            if (mCurrentStep.validate()) {
                if (mCurrentStep.isChange()) {
                    mCurrentStep.doNext();
                } else {
                    next();
                }
            }
        }
        private void initBackDialog() {
            mBackDialog = BaseDialog.getDialog(RegisterActivity.this, "提示",
```

```
                "确认要放弃注册么?", "确认", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                        finish();
                    }
                }, "取消", new DialogInterface.OnClickListener() {

                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.cancel();
                    }
                });
        mBackDialog.setButton1Background(R.drawable.btn_default_popsubmit);

    }
    @Override
    protected void putAsyncTask(AsyncTask<Void, Void, Boolean> asyncTask) {
        super.putAsyncTask(asyncTask);
    }
    @Override
    protected void showCustomToast(String text) {
        super.showCustomToast(text);
    }
    @Override
    protected void showLoadingDialog(String text) {
        super.showLoadingDialog(text);
    }
    @Override
    protected void dismissLoadingDialog() {
        super.dismissLoadingDialog();
    }
    protected int getScreenWidth() {
        return mScreenWidth;
    }
    protected BaseApplication getBaseApplication() {
        return mApplication;
    }
    protected String getPhoneNumber() {
        if (mStepPhone != null) {
            return mStepPhone.getPhoneNumber();
        }
        return "";
```

```
        }
    }
```

如果注册手机号合法，则弹出输入验证码界面，如图 17-3 所示。

图 17-3　输入验证码界面

17.3.3　输入验证码界面 Activity

输入验证码界面 Activity 的实现文件是 StepVerify.java，功能是验证注册用户输入的验证码是否合法。在本项目中，设置的固定验证码是“123456”。文件 StepVerify.java 的具体实现代码如下。

```
public class StepVerify extends RegisterStep implements OnClickListener,
        TextWatcher {
    private HandyTextView mHtvPhoneNumber;
    private EditText mEtVerifyCode;
    private Button mBtnResend;
    private HandyTextView mHtvNoCode;
    private static final String PROMPT = "验证码已经发送到* ";
    private static final String DEFAULT_VALIDATE_CODE = "123456";
    private boolean mIsChange = true;
```

```
private String mVerifyCode;
private int mReSendTime = 60;
private BaseDialog mBaseDialog;
public StepVerify(RegisterActivity activity, View contentRootView) {
    super(activity, contentRootView);
    handler.sendEmptyMessage(0);
}
@Override
public void initViews() {
    mHtvPhoneNumber = (HandyTextView) findViewById(R.id.reg_verify_htv_phonenumber);
    mHtvPhoneNumber.setText(PROMPT + getPhoneNumber());
    mEtVerifyCode = (EditText) findViewById(R.id.reg_verify_et_verifycode);
    mBtnResend = (Button) findViewById(R.id.reg_verify_btn_resend);
    mBtnResend.setEnabled(false);
    mBtnResend.setText("重发(60)");
    mHtvNoCode = (HandyTextView) findViewById(R.id.reg_verify_htv_nocode);
    TextUtils.addUnderlineText(mContext, mHtvNoCode, 0, mHtvNoCode
            .getText().toString().length());
}
@Override
public void initEvents() {
    mBtnResend.setOnClickListener(this);
    mHtvNoCode.setOnClickListener(this);
    mEtVerifyCode.addTextChangedListener(this);
}
@Override
public void doNext() {
    putAsyncTask(new AsyncTask<Void, Void, Boolean>() {
        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            showLoadingDialog("正在验证,请稍后...");
        }
        @Override
        protected Boolean doInBackground(Void... params) {
            try {
                Thread.sleep(2000);
                if (DEFAULT_VALIDATE_CODE.equals(mVerifyCode)) {
                    return true;
                }
            } catch (InterruptedException e) {

            }
```

```
                return false;
            }
            @Override
            protected void onPostExecute(Boolean result) {
                super.onPostExecute(result);
                dismissLoadingDialog();
                if (result) {
                    mIsChange = false;
                    mOnNextActionListener.next();
                } else {
                    mBaseDialog = BaseDialog.getDialog(mContext, "提示", "验证码错误",
                            "确认", new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog,
                                        int which) {
                                    mEtVerifyCode.requestFocus();
                                    dialog.dismiss();
                                }
                            });
                    mBaseDialog.show();
                }
            }
        });
    }
    @Override
    public boolean validate() {
        if (isNull(mEtVerifyCode)) {
            showCustomToast("请输入验证码");
            mEtVerifyCode.requestFocus();
            return false;
        }
        mVerifyCode = mEtVerifyCode.getText().toString().trim();
        return true;
    }
    @Override
    public boolean isChange() {
        return mIsChange;
    }
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.reg_verify_btn_resend:
            handler.sendEmptyMessage(0);
```

```
                    break;
                case R.id.reg_verify_htv_nocode:
                    showCustomToast("抱歉,暂时不支持此操作");
                    break;
                }
            }
            @Override
            public void afterTextChanged(Editable s) {
            }
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count,
                    int after) {
            }
            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {
                mIsChange = true;
            }
            Handler handler = new Handler() {
                @Override
                public void handleMessage(Message msg) {
                    super.handleMessage(msg);
                    if (mReSendTime > 1) {
                        mReSendTime--;
                        mBtnResend.setEnabled(false);
                        mBtnResend.setText("重发(" + mReSendTime + ")");
                        handler.sendEmptyMessageDelayed(0, 1000);
                    } else {
                        mReSendTime = 60;
                        mBtnResend.setEnabled(true);
                        mBtnResend.setText("重　　发");
                    }
                }
            };
        }
```

17.3.4　设置密码界面 Activity

如果输入的验证码合法，单击“下一步”按钮后会跳转到设置密码界面，在界面上方显示两个文本框供用户分别输入登录密码和确认密码，在界面下方显示“上一步”和“下一步”按钮，如图 17-4 所示。

设置密码界面 Activity 的实现文件是 StepSetPassword.java，功能是验证注册用户输入的两个密码完全一致，并且是 6 位以上。文件 StepSetPassword.java 的具体实现代码如下。

图 17-4　设置密码界面

```
public class StepSetPassword extends RegisterStep implements TextWatcher {
    private EditText mEtPwd;
    private EditText mEtRePwd;
    private boolean mIsChange = true;
    public StepSetPassword(RegisterActivity activity, View contentRootView) {
        super(activity, contentRootView);
    }
    @Override
    public void initViews() {
        mEtPwd = (EditText) findViewById(R.id.reg_setpwd_et_pwd);
        mEtRePwd = (EditText) findViewById(R.id.reg_setpwd_et_repwd);
    }
    @Override
    public void initEvents() {
        mEtPwd.addTextChangedListener(this);
        mEtRePwd.addTextChangedListener(this);
    }
    @Override
    public void doNext() {
        mIsChange = false;
        mOnNextActionListener.next();
    }
```

```
@Override
public boolean validate() {
    String pwd = null;
    String rePwd = null;
    if (isNull(mEtPwd)) {
        showCustomToast("请输入密码");
        mEtPwd.requestFocus();
        return false;
    } else {
        pwd = mEtPwd.getText().toString().trim();
        if (pwd.length() < 6) {
            showCustomToast("密码不能小于 6 位");
            mEtPwd.requestFocus();
            return false;
        }
    }
    if (isNull(mEtRePwd)) {
        showCustomToast("请重复输入一次密码");
        mEtRePwd.requestFocus();
        return false;
    } else {
        rePwd = mEtRePwd.getText().toString().trim();
        if (!pwd.equals(rePwd)) {
            showCustomToast("两次输入的密码不一致");
            mEtRePwd.requestFocus();
            return false;
        }
    }
    return true;
}
@Override
public boolean isChange() {
    return mIsChange;
}
@Override
public void afterTextChanged(Editable s) {
}
@Override
public void beforeTextChanged(CharSequence s, int start, int count,
        int after) {
}
@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
```

```
                mIsChange = true;
            }
        }
```

17.3.5 设置用户名界面 Activity

如果输入的密码合法，单击“下一步”按钮后会跳转到设置用户名界面，在界面上方显示一个文本框供用户输入用户名，显示一个单选按钮供用户选择性别，在界面下方显示“上一步”和“下一步”按钮，如图 17-5 所示。

图 17-5　设置用户名界面

设置用户名界面 Activity 的实现文件是 StepBaseInfo.java，功能是验证是否输入用户名并选择性别。文件 StepBaseInfo.java 的具体实现代码如下。

```
public class StepBaseInfo extends RegisterStep implements TextWatcher,
        OnCheckedChangeListener {
    private EditText mEtName;
    private RadioGroup mRgGender;
    private RadioButton mRbMale;
    private RadioButton mRbFemale;
    private boolean mIsChange = true;
    private boolean mIsGenderAlert;
    private BaseDialog mBaseDialog;
    public StepBaseInfo(RegisterActivity activity, View contentRootView) {
        super(activity, contentRootView);
    }
    @Override
```

```
public void initViews() {
        mEtName = (EditText) findViewById(R.id.reg_baseinfo_et_name);
        mRgGender = (RadioGroup) findViewById(R.id.reg_baseinfo_rg_gender);
        mRbMale = (RadioButton) findViewById(R.id.reg_baseinfo_rb_male);
        mRbFemale = (RadioButton) findViewById(R.id.reg_baseinfo_rb_female);
}
@Override
public void initEvents() {
        mEtName.addTextChangedListener(this);
        mRgGender.setOnCheckedChangeListener(this);
}
@Override
public void doNext() {
        mOnNextActionListener.next();
}
@Override
public boolean validate() {
        if (isNull(mEtName)) {
                showCustomToast("请输入用户名");
                mEtName.requestFocus();
                return false;
        }
        if (mRgGender.getCheckedRadioButtonId() < 0) {
                showCustomToast("请选择性别");
                return false;
        }
        return true;
}
@Override
public boolean isChange() {
        return mIsChange;
}
@Override
public void onCheckedChanged(RadioGroup group, int checkedId) {
        mIsChange = true;
        if (!mIsGenderAlert) {
                mIsGenderAlert = true;
                mBaseDialog = BaseDialog.getDialog(mContext, "提示", "注册成功后性别将不可更改",
                                "确认", new DialogInterface.OnClickListener() {
                                        @Override
                                        public void onClick(DialogInterface dialog, int which) {
                                                dialog.dismiss();
                                        }
                                });
```

```
                mBaseDialog.show();
            }
            switch (checkedId) {
            case R.id.reg_baseinfo_rb_male:
                mRbMale.setChecked(true);
                break;
            case R.id.reg_baseinfo_rb_female:
                mRbFemale.setChecked(true);
                break;
            }
        }
        @Override
        public void afterTextChanged(Editable s) {
        }
        @Override
        public void beforeTextChanged(CharSequence s, int start, int count,
                int after) {
        }
        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
            mIsChange = true;
        }
    }
```

17.3.6 设置生日界面 Activity

如果设置的用户名和性别合法，单击“下一步”按钮后会跳转到设置生日界面，在界面上方显示年、月、日供用户选择生日，在界面下方显示“上一步”和“下一步”按钮，如图 17-6 所示。

图 17-6 设置生日界面

设置生日界面 Activity 的实现文件是 StepBirthday.java，功能是验证用户设置的年龄的合法性，系统要求的合法年龄范围在“12～100”岁之间。文件 StepBirthday.java 的具体实现代码如下。

```
public class StepBirthday extends RegisterStep implements OnDateChangedListener {
    private HandyTextView mHtvConstellation;
    private HandyTextView mHtvAge;
    private DatePicker mDpBirthday;
    private Calendar mCalendar;
    private Date mMinDate;
    private Date mMaxDate;
    private Date mSelectDate;
    private static final int MAX_AGE = 100;
    private static final int MIN_AGE = 12;
    public StepBirthday(RegisterActivity activity, View contentRootView) {
        super(activity, contentRootView);
        initData();
    }
    private void flushBirthday(Calendar calendar) {
        String constellation = TextUtils.getConstellation(
                calendar.get(Calendar.MONTH),
                calendar.get(Calendar.DAY_OF_MONTH));
        mSelectDate = calendar.getTime();
        mHtvConstellation.setText(constellation);
        int age = TextUtils.getAge(calendar.get(Calendar.YEAR),
                calendar.get(Calendar.MONTH),
                calendar.get(Calendar.DAY_OF_MONTH));
        mHtvAge.setText(age + "");
    }
    private void initData() {
        mSelectDate = DateUtils.getDate("19900101");
        Calendar mMinCalendar = Calendar.getInstance();
        Calendar mMaxCalendar = Calendar.getInstance();
        mMinCalendar.set(Calendar.YEAR, mMinCalendar.get(Calendar.YEAR)
                - MIN_AGE);
        mMinDate = mMinCalendar.getTime();
        mMaxCalendar.set(Calendar.YEAR, mMaxCalendar.get(Calendar.YEAR)
                - MAX_AGE);
        mMaxDate = mMaxCalendar.getTime();
        mCalendar = Calendar.getInstance();
        mCalendar.setTime(mSelectDate);
        flushBirthday(mCalendar);
        mDpBirthday.init(mCalendar.get(Calendar.YEAR),
                mCalendar.get(Calendar.MONTH),
                mCalendar.get(Calendar.DAY_OF_MONTH), this);
    }
```

```
    @Override
    public void initViews() {
        mHtvConstellation = (HandyTextView) findViewById(R.id.reg_birthday_htv_constellation);
        mHtvAge = (HandyTextView) findViewById(R.id.reg_birthday_htv_age);
        mDpBirthday = (DatePicker) findViewById(R.id.reg_birthday_dp_birthday);
    }
    @Override
    public void initEvents() {
    }
    @Override
    public void doNext() {
        mOnNextActionListener.next();
    }
    @Override
    public boolean validate() {
        return true;
    }

    @Override
    public boolean isChange() {
        return false;
    }
    @Override
    public void onDateChanged(DatePicker view, int year, int monthOfYear,
            int dayOfMonth) {
        mCalendar = Calendar.getInstance();
        mCalendar.set(year, monthOfYear, dayOfMonth);
        if (mCalendar.getTime().after(mMinDate)
                || mCalendar.getTime().before(mMaxDate)) {
            mCalendar.setTime(mSelectDate);
            mDpBirthday.init(mCalendar.get(Calendar.YEAR),
                    mCalendar.get(Calendar.MONTH),
                    mCalendar.get(Calendar.DAY_OF_MONTH), this);
        } else {
            flushBirthday(mCalendar);
        }
    }

}
```

17.3.7 设置头像界面 Activity

如果设置的年龄合法，单击“下一步”按钮后会跳转到设置头像界面，在界面上方显示选择图片按钮和拍照按钮供用户快速设置头像，在界面下方显示“上一步”和“注册”按钮，如图 17-7 所示。

图 17-7　设置头像界面

设置头像界面 Activity 的实现文件是 StepPhoto.java，功能是验证用户是否设置了头像。文件 StepPhoto.java 的具体实现代码如下。

```
public class StepPhoto extends RegisterStep implements OnClickListener {
    private HandyTextView mHtvRecommendation;
    private ImageView mIvUserPhoto;
    private LinearLayout mLayoutSelectPhoto;
    private LinearLayout mLayoutTakePicture;
    private LinearLayout mLayoutAvatars;
    private View[] mMemberBlocks;
    private String[] mAvatars = new String[] { "welcome_0", "welcome_1",
            "welcome_2", "welcome_3", "welcome_4", "welcome_5" };
    private String[] mDistances = new String[] { "0.84km", "1.02km", "1.34km",
            "1.88km", "2.50km", "2.78km" };
    private String mTakePicturePath;
    private Bitmap mUserPhoto;
    private EditTextDialog mEditTextDialog;
    public StepPhoto(RegisterActivity activity, View contentRootView) {
        super(activity, contentRootView);
        initAvatarsItem();
    }
    private void initAvatarsItem() {
        initMemberBlocks();
        for (int i = 0; i < mMemberBlocks.length; i++) {
            ((ImageView) mMemberBlocks[i]
                    .findViewById(R.id.welcome_item_iv_avatar))
                    .setImageBitmap(getBaseApplication().getAvatar(mAvatars[i]));
```

```
                ((HandyTextView) mMemberBlocks[i]
                        .findViewById(R.id.welcome_item_htv_distance))
                        .setText(mDistances[i]);
            }
        }
        private void initMemberBlocks() {
            mMemberBlocks = new View[6];
            mMemberBlocks[0] = findViewById(R.id.reg_photo_include_member_avatar_block0);
            mMemberBlocks[1] = findViewById(R.id.reg_photo_include_member_avatar_block1);
            mMemberBlocks[2] = findViewById(R.id.reg_photo_include_member_avatar_block2);
            mMemberBlocks[3] = findViewById(R.id.reg_photo_include_member_avatar_block3);
            mMemberBlocks[4] = findViewById(R.id.reg_photo_include_member_avatar_block4);
            mMemberBlocks[5] = findViewById(R.id.reg_photo_include_member_avatar_block5);
            int margin = (int) TypedValue.applyDimension(
                    TypedValue.COMPLEX_UNIT_DIP, 4, mContext.getResources()
                            .getDisplayMetrics());
            int widthAndHeight = (getScreenWidth() - margin * 12) / 6;
            for (int i = 0; i < mMemberBlocks.length; i++) {
                ViewGroup.LayoutParams params = mMemberBlocks[i].findViewById(
                        R.id.welcome_item_iv_avatar).getLayoutParams();
                params.width = widthAndHeight;
                params.height = widthAndHeight;
                mMemberBlocks[i].findViewById(R.id.welcome_item_iv_avatar)
                        .setLayoutParams(params);
            }
            mLayoutAvatars.invalidate();
        }
        public void setUserPhoto(Bitmap bitmap) {
            if (bitmap != null) {
                mUserPhoto = bitmap;
                mIvUserPhoto.setImageBitmap(mUserPhoto);
                return;
            }
            showCustomToast("未获取到图片");
            mUserPhoto = null;
            mIvUserPhoto.setImageResource(R.drawable.ic_common_def_header);
        }
        public String getTakePicturePath() {
            return mTakePicturePath;
        }
        @Override
        public void initViews() {
            mHtvRecommendation = (HandyTextView) findViewById(R.id.reg_photo_htv_recommendation);
            mIvUserPhoto = (ImageView) findViewById(R.id.reg_photo_iv_userphoto);
            mLayoutSelectPhoto = (LinearLayout) findViewById(R.id.reg_photo_layout_selectphoto);
            mLayoutTakePicture = (LinearLayout) findViewById(R.id.reg_photo_layout_takepicture);
```

```
        mLayoutAvatars = (LinearLayout) findViewById(R.id.reg_photo_layout_avatars);
    }
    @Override
    public void initEvents() {
        mHtvRecommendation.setOnClickListener(this);
        mLayoutSelectPhoto.setOnClickListener(this);
        mLayoutTakePicture.setOnClickListener(this);
    }
    @Override
    public boolean validate() {
        if (mUserPhoto == null) {
            showCustomToast("请添加头像");
            return false;
        }
        return true;
    }
    @Override
    public void doNext() {
        putAsyncTask(new AsyncTask<Void, Void, Boolean>() {
            @Override
            protected void onPreExecute() {
                super.onPreExecute();
                showLoadingDialog("请稍后,正在提交...");
            }
            @Override
            protected Boolean doInBackground(Void... params) {
                try {
                    Thread.sleep(2000);
                    return true;
                } catch (InterruptedException e) {
                }
                return false;
            }
            @Override
            protected void onPostExecute(Boolean result) {
                super.onPostExecute(result);
                dismissLoadingDialog();
                if (result) {
                    mActivity.finish();
                }
            }

        });
    }
    @Override
    public boolean isChange() {
```

```
                return false;
            }
            @Override
            public void onClick(View v) {
                switch (v.getId()) {
                case R.id.reg_photo_htv_recommendation:
                    mEditTextDialog = new EditTextDialog(mContext);
                    mEditTextDialog.setTitle("填写推荐人");
                    mEditTextDialog.setButton("取消",
                            new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog, int which) {
                                    mEditTextDialog.cancel();
                                }
                            }, "确认", new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog, int which) {
                                    String text = mEditTextDialog.getText();
                                    if (text == null) {
                                        mEditTextDialog.requestFocus();
                                        showCustomToast("请输入推荐人号码");
                                    } else {
                                        mEditTextDialog.dismiss();
                                        showCustomToast("您输入的推荐人号码为:" + text);
                                    }
                                }
                            });
                    mEditTextDialog.show();
                    break;
                case R.id.reg_photo_layout_selectphoto:
                    PhotoUtils.selectPhoto(mActivity);
                    break;
                case R.id.reg_photo_layout_takepicture:
                    mTakePicturePath = PhotoUtils.takePicture(mActivity);
                    break;
                }
            }
        }
```

设置头像完毕后，单击“注册”按钮完成注册。

17.4 实现系统主界面

当用户输入合法的注册信息登录陌陌后，会首先显示系统主界面，如图 17-8 所示。

图 17-8　系统主界面

在本节的内容中，将详细讲解系统主界面的具体实现过程。

17.4.1　主界面布局

系统主界面的布局文件是 activity_maintabs.xml，功能是使用 TabWidget 控件将屏幕界面分割成 5 个部分。文件 activity_maintabs.xml 的具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ffffffff" >
        <RelativeLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" >
            <FrameLayout
                android:id="@android:id/tabcontent"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:layout_above="@android:id/tabs"
                android:background="@color/background_normal" />
```

```
                <TabWidget
                    android:id="@android:id/tabs"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:layout_alignParentBottom="true"
                    android:divider="@null" />
            </RelativeLayout>
        </LinearLayout>
    </TabHost>
```

17.4.2 实现主界面 Activity

主界面 Activity 的实现文件是 MainTabActivity.java，功能是通过函数 initTabs()初始化显示 TabWidget 控件中的内容，默认设置为显示“附近的人”。文件 MainTabActivity.java 的具体实现代码如下。

```
public class MainTabActivity extends TabActivity {
    private TabHost mTabHost;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maintabs);
        initViews();
        initTabs();
    }
    private void initViews() {
        mTabHost = getTabHost();
    }
    private void initTabs() {
        LayoutInflater inflater = LayoutInflater.from(MainTabActivity.this);
        View nearbyView = inflater.inflate(
                R.layout.common_bottombar_tab_nearby, null);
        TabHost.TabSpec nearbyTabSpec = mTabHost.newTabSpec(
                NearByActivity.class.getName()).setIndicator(nearbyView);
        nearbyTabSpec.setContent(new Intent(MainTabActivity.this,
                NearByActivity.class));
        mTabHost.addTab(nearbyTabSpec);
        View nearbyFeedsView = inflater.inflate(
                R.layout.common_bottombar_tab_site, null);
        TabHost.TabSpec nearbyFeedsTabSpec = mTabHost.newTabSpec(
                NearByFeedsActivity.class.getName()).setIndicator(
                nearbyFeedsView);
        nearbyFeedsTabSpec.setContent(new Intent(MainTabActivity.this,
                NearByFeedsActivity.class));
        mTabHost.addTab(nearbyFeedsTabSpec);
```

```
                View sessionListView = inflater.inflate(
                        R.layout.common_bottombar_tab_chat, null);
                TabHost.TabSpec sessionListTabSpec = mTabHost.newTabSpec(
                        SessionListActivity.class.getName()).setIndicator(
                        sessionListView);
                sessionListTabSpec.setContent(new Intent(MainTabActivity.this,
                        SessionListActivity.class));
                mTabHost.addTab(sessionListTabSpec);
                View contactView = inflater.inflate(
                        R.layout.common_bottombar_tab_friend, null);
                TabHost.TabSpec contactTabSpec = mTabHost.newTabSpec(
                        ContactTabsActivity.class.getName()).setIndicator(contactView);
                contactTabSpec.setContent(new Intent(MainTabActivity.this,
                        ContactTabsActivity.class));
                mTabHost.addTab(contactTabSpec);
                View userSettingView = inflater.inflate(
                        R.layout.common_bottombar_tab_profile, null);
                TabHost.TabSpec userSettingTabSpec = mTabHost.newTabSpec(
                        UserSettingActivity.class.getName()).setIndicator(
                        userSettingView);
                userSettingTabSpec.setContent(new Intent(MainTabActivity.this,
                        UserSettingActivity.class));
                mTabHost.addTab(userSettingTabSpec);
        }
}
```

17.4.3　实现“附近的人”界面

在系统主界面中，中间大部分内容显示的是系统“附近的人”信息，此功能的实现布局文件是 common_bottombar_tab_nearby.xml，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="0dip"
    android:layout_height="40dip"
    android:layout_weight="1"
    android:background="@drawable/bg_tb_item_center"
    android:paddingBottom="2dip" >
    <com.immomo.momo.android.view.HandyTextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:drawableTop="@drawable/ic_tab_nearby"
        android:gravity="center_horizontal"
        android:text="附近"
```

```
            android:textColor="@color/maintab_text_color"
            android:textSize="11sp"
            android:shadowDx="0.0"
            android:shadowDy="-1.0"
            android:shadowRadius="1.0"/>
    </RelativeLayout>
```

“附近的人”界面 Activity 的实现文件是 NearByActivity.java，功能是在顶部显示“附近”、“群组”和“个人”选项卡，并监听用户单击屏幕事件，根据用户操作执行对应的事件处理函数。例如单击搜索图标可以根据关键字快速检索附近的人。文件 NearByActivity.java 的具体实现代码如下。

```
public class NearByActivity extends TabItemActivity {
    private HeaderLayout mHeaderLayout;
    private HeaderSpinner mHeaderSpinner;
    private NearByPeopleFragment mPeopleFragment;
    private NearByGroupFragment mGroupFragment;
    private NearByPopupWindow mPopupWindow;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nearby);
        initPopupWindow();
        initViews();
        initEvents();
        init();
    }
    @Override
    protected void initViews() {
        mHeaderLayout = (HeaderLayout) findViewById(R.id.nearby_header);
        mHeaderLayout.initSearch(new OnSearchClickListener());
        mHeaderSpinner = mHeaderLayout.setTitleNearBy("附近",
                new OnSpinnerClickListener(), "附近群组",
                R.drawable.ic_topbar_search,
                new OnMiddleImageButtonClickListener(), "个人", "群组",
                new OnSwitcherButtonClickListener());
        mHeaderLayout.init(HeaderStyle.TITLE_NEARBY_PEOPLE);
    }
    @Override
    protected void initEvents() {
    }
    @Override
    protected void init() {
        mPeopleFragment = new NearByPeopleFragment(mApplication, this, this);
        mGroupFragment = new NearByGroupFragment(mApplication, this, this);
```

```
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.nearby_layout_content, mPeopleFragment).commit();
    }
    private void initPopupWindow() {
        mPopupWindow = new NearByPopupWindow(this);
        mPopupWindow.setOnSubmitClickListener(new onSubmitClickListener() {
            @Override
            public void onClick() {
                mPeopleFragment.onManualRefresh();
            }
        });
        mPopupWindow.setOnDismissListener(new OnDismissListener() {
            @Override
            public void onDismiss() {
                mHeaderSpinner.initSpinnerState(false);
            }
        });
    }
    public class OnSpinnerClickListener implements onSpinnerClickListener {
        @Override
        public void onClick(boolean isSelect) {
            if (isSelect) {
                mPopupWindow
                        .showViewTopCenter(findViewById(R.id.nearby_layout_root));
            } else {
                mPopupWindow.dismiss();
            }
        }
    }
    public class OnSearchClickListener implements onSearchListener {
        @Override
        public void onSearch(EditText et) {
            String s = et.getText().toString().trim();
            if (TextUtils.isEmpty(s)) {
                showCustomToast("请输入搜索关键字");
                et.requestFocus();
            } else {
                ((InputMethodManager) getSystemService(INPUT_METHOD_SERVICE))
                        .hideSoftInputFromWindow(NearByActivity.this
                                .getCurrentFocus().getWindowToken(),
                                InputMethodManager.HIDE_NOT_ALWAYS);
                putAsyncTask(new AsyncTask<Void, Void, Boolean>() {
                    @Override
                    protected void onPreExecute() {
                        super.onPreExecute();
```

```
                                mHeaderLayout.changeSearchState(SearchState.SEARCH);
                            }
                            @Override
                            protected Boolean doInBackground(Void... params) {
                                try {
                                    Thread.sleep(2000);
                                } catch (InterruptedException e) {
                                    e.printStackTrace();
                                }
                                return false;
                            }
                            @Override
                            protected void onPostExecute(Boolean result) {
                                super.onPostExecute(result);
                                mHeaderLayout.changeSearchState(SearchState.INPUT);
                                showCustomToast("未找到搜索的群");
                            }
                        });
                    }
                }
            }
            public class OnMiddleImageButtonClickListener implements
                    onMiddleImageButtonClickListener {
                @Override
                public void onClick() {
                    mHeaderLayout.showSearch();
                }
            }
            public class OnSwitcherButtonClickListener implements
                    onSwitcherButtonClickListener {
                @Override
                public void onClick(SwitcherButtonState state) {
                    FragmentTransaction ft = getSupportFragmentManager()
                            .beginTransaction();
                    ft.setCustomAnimations(R.anim.fragment_fadein,
                            R.anim.fragment_fadeout);
                    switch (state) {
                    case LEFT:
                        mHeaderLayout.init(HeaderStyle.TITLE_NEARBY_PEOPLE);
                        ft.replace(R.id.nearby_layout_content, mPeopleFragment)
                                .commit();
                        break;

                    case RIGHT:
                        mHeaderLayout.init(HeaderStyle.TITLE_NEARBY_GROUP);
```

```
                        ft.replace(R.id.nearby_layout_content, mGroupFragment).commit();
                        break;
                    }
                }
            }
            @Override
            public void onBackPressed() {
                if (mHeaderLayout.searchIsShowing()) {
                    clearAsyncTask();
                    mHeaderLayout.dismissSearch();
                    mHeaderLayout.clearSearch();
                    mHeaderLayout.changeSearchState(SearchState.INPUT);
                } else {
                    finish();
                }
            }
        }
```

17.4.4　实现“附近的群组”界面

当在顶部 

单击“群组”选项卡后，会在系统主界面中间显示系统“附近的群组”信息，此功能的实现布局文件是 fragment_nearbygroup.xml，具体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <com.immomo.momo.android.view.MoMoRefreshExpandableList
        android:id="@+id/nearby_group_mmrelv_list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:cacheColorHint="@color/transparent"
        android:divider="@null"
        android:fadingEdge="none"
        android:listSelector="@drawable/list_selector_transition" >
    </com.immomo.momo.android.view.MoMoRefreshExpandableList>
    <LinearLayout
        android:id="@+id/nearby_group_layout_cover"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:clickable="true" >
        <include
            layout="@layout/include_nearby_group_header"
            android:visibility="invisible" />
```

```
    </LinearLayout>
</FrameLayout>
```

“附近的群组”界面 Activity 的实现文件是 NearByGroupFragment.java，功能是在系统主界面中间加载显示附近的群组信息，并通过 onRefresh()函数进行刷新以及显示最新的群主。文件 NearByGroupFragment.java 的具体实现代码如下。

```
public class NearByGroupFragment extends BaseFragment implements
        OnClickListener, OnItemClickListener, OnRefreshListener,
        OnCancelListener {
    private LinearLayout mLayoutCover;
    private MoMoRefreshExpandableList mMmrelvList;
    private NearByGroupAdapter mAdapter;
    public NearByGroupFragment() {
        super();
    }
    public NearByGroupFragment(BaseApplication application, Activity activity,
            Context context) {
        super(application, activity, context);
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        mView = inflater.inflate(R.layout.fragment_nearbygroup, container,
                false);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
    @Override
    protected void initViews() {
        mLayoutCover = (LinearLayout) findViewById(R.id.nearby_group_layout_cover);
        mMmrelvList = (MoMoRefreshExpandableList) findViewById(R.id.nearby_group_ mmrelv_list);
    }
    @Override
    protected void initEvents() {
        mLayoutCover.setOnClickListener(this);
        mMmrelvList.setOnItemClickListener(this);
        mMmrelvList.setOnRefreshListener(this);
        mMmrelvList.setOnCancelListener(this);
    }
    @Override
    protected void init() {
        getGroups();
    }
    private void getGroups() {
        if (mApplication.mNearByGroups.isEmpty()) {
```

```
                putAsyncTask(new AsyncTask<Void, Void, Boolean>() {
                    @Override
                    protected void onPreExecute() {
                        super.onPreExecute();
                        showLoadingDialog("正在加载,请稍后...");
                    }
                    @Override
                    protected Boolean doInBackground(Void... params) {
                        return JsonResolveUtils.resolveNearbyGroup(mApplication);
                    }
                    @Override
                    protected void onPostExecute(Boolean result) {
                        super.onPostExecute(result);
                        dismissLoadingDialog();
                        if (!result) {
                            showCustomToast("数据加载失败...");
                        } else {
                            mAdapter = new NearByGroupAdapter(mApplication,
                                    mContext, mApplication.mNearByGroups);
                            mMmrelvList.setAdapter(mAdapter);
                            mMmrelvList.setPinnedHeaderView(mActivity
                                    .getLayoutInflater().inflate(
                                            R.layout.include_nearby_group_header,
                                            mMmrelvList, false));
                        }
                    }
                });
            } else {
                mAdapter = new NearByGroupAdapter(mApplication, mContext,
                        mApplication.mNearByGroups);
                mMmrelvList.setAdapter(mAdapter);
                mMmrelvList.setPinnedHeaderView(mActivity.getLayoutInflater()
                        .inflate(R.layout.include_nearby_group_header, mMmrelvList,
                                false));
            }
        }
        @Override
        public void onRefresh() {
            putAsyncTask(new AsyncTask<Void, Void, Boolean>() {
                @Override
                protected Boolean doInBackground(Void... params) {
                    try {
                        Thread.sleep(2000);
                    } catch (InterruptedException e) {
                    }
```

```
                    return null;
                }
                @Override
                protected void onPostExecute(Boolean result) {
                    super.onPostExecute(result);
                    mMmrelvList.onRefreshComplete();
                }
            });
        }
        @Override
        public void onCancel() {
            clearAsyncTask();
            mMmrelvList.onRefreshComplete();
        }
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        }
        @Override
        public void onClick(View v) {
            if (mMmrelvList.ismHeaderViewVisible()) {
                mAdapter.onPinnedHeaderClick(mMmrelvList.getFirstItemPosition());
            } else {
                mAdapter.onPinnedHeaderClick(1);
            }
        }
    }
```

到此为止，本章仿陌陌系统的主要内容介绍完毕。篇幅所限，本书没有讲解找回密码、聊天交流、设置、留言板等信息。有关这方面的具体内容，请读者参考本书网络资源中的源代码。

Android 网络开发从入门到精通

本书全部内容分为四篇，共计 17 章，循序渐进地讲解了 Android 网络开发方面的知识。本书从搭建开发环境和核心框架分析讲起，依次讲解了 Android 系统概述，Android 网络开发基础，Java 中的网络通信基础，下载、上传数据，Socket 数据通信，处理 XML 数据，WebKit 浏览网页，开发移动网页，开发蓝牙应用程序，开发 Wi-Fi 应用程序，NFC 近场通信技术详解，开发电子邮件应用程序，Android 典型网络应用实践，开发移动微博应用程序，开发 Web 版的电话本管理系统，开发移动微信系统，开发仿陌陌交友系统等高级知识。本书几乎涵盖了 Android 网络开发中的所有主要内容，并且全书内容言简意赅，讲解方法通俗易懂、详细，不但适合应用开发高手们的学习，也特别适合初学者的系统学习。

本书适合 Android 初学者、Android 爱好者、Android 网络开发人员和移动浏览器开发人员，也可以作为相关培训学校和大专院校相关专业的教学用书。

51CTO

为中华崛起传播智慧

地址：北京市百万庄大街22号
邮政编码：100037

电话服务
服务咨询热线：010-88361066
读者购书热线：010-68326294
010-88379203

网络服务
机工官网：www.cmpbook.com
机工官博：weibo.com/cmp1952
金书网：www.golden-book.com
教育服务网：www.cmpedu.com
封面无防伪标均为盗版

策划编辑◎丁诚 / 封面设计◎道乐文化

上架指导 计算机/移动开发

ISBN 978-7-111-52203-4

机械工业出版社 微信服务号

计算机分社 微信服务号

定价：85.00元