



移动开发系列丛书



Google Android

开发入门与实战

靳岩 姚尚朗 编著

☆ 高焕堂 ☆ 邹仕洪 ☆ 霍泰稳 ☆ 何峰
☆ 高昂 ☆ 蔡望勤 ☆ 冯大辉 ☆ 熊节
鼎力推荐

5 大完整商业应用案例：

RSS阅读器、基于Google Map的个人GPS、豆瓣网客户端、
在线音乐播放器、手机信息查看助手

- 全部内容基于Android SDK 1.5
- 随书附赠400多分钟详细教学视频，286段示例代码
- 从事一线开发的作者团队提供在线答疑



DVD视频



人民邮电出版社
POSTS & TELECOM PRESS

推荐序

随着 3G 的到来, 无线带宽越来越高, 使得更多内容丰富的应用程序布置在手机上成为可能, 如视频通话、视频点播、移动互联网冲浪、在线看书/听歌、内容分享等。为了承载这些数据应用及快速部署, 手机功能将会越来越智能, 越来越开放, 为了实现这些需求, 必须有一个好的开发平台来支持, 在此由 Google 公司发起的 OHA 联盟走在了业界的前列, 于 2007 年 11 月推出了开放的 Android 平台, 任何公司及个人都可以免费获取到源代码及开发 SDK。由于其开放性和优异性, Android 平台得到了业界广泛的支持, 其中包括各大手机厂商和著名的移动运营商等。继 2008 年 9 月第一款基于 Android 平台的手机 G1 发布之后, 预计三星、摩托罗拉、索爱、LG、华为等公司都将推出自己的基于 Android 平台的手机, 中国移动也将联合各手机厂商共同推出基于 Android 平台的 OPhone。按目前的发展态势, 我们有理由相信, Android 平台能够在短时间内跻身智能手机开发平台的前列。

由于 Android 平台被推出的时间才一年左右, 了解 Android 平台软件开发技术的程序员还不多, 如何迅速地推广和普及 Android 平台软件开发技术, 让越来越多的人参与到 Android 应用的开发中, 是整个产业链都在关注的一个话题。本书作者较早就从事 Android 的研究与开发工作, 为了帮助开发者更快地进入 Android 开发行列, 他们不辞辛劳地编写了 Android 开发教程——《Google Android 开发入门与实战》。本书系统讲解了 Android 软件开发的基础知识, 图文并茂地帮助读者学习和掌握 SDK、开发流程以及常用的 API 等。书中以讲述实战实例为导向, 用一个个典型应用生动地引领读者进行项目开发实践。作为一本既及时、又翔实、理论与实践相结合的教程, 《Google Android 开发入门与实战》一书很值得入门者阅读。

值得一提的是, 书中的开发实例很有创意, 将传统互联网的内容/服务与移动平台紧密结合起来, 如书中的豆瓣网客户端、在线音乐播放器等应用, 都体现了移动互联网应用所需的创新精神及良好的用户体验理念, 这个设计思路很值得大家去思考和学习。

最后, 祝广大开发者的技术日益精进, 早日开始 Android 开发之旅, 赶上移动互联网的第一班车, 共赢中国 3G 未来!

邹仕洪
网秦副总裁

名家推荐

这是一本很平易近人的 Android 入门书籍，也是开发者及非开发者两相宜的实务书籍，它能陪伴您顺利入门，并驰骋于无限宽广的 Android 系统和应用领域。

—台湾著名技术专家 高焕堂

随着 3G 元年的到来，移动互联网的不断深入发展，G1 和 G2 手机的相继问世，Android 上网本的若隐若现，中国移动、中国联通等运营商即将发布基于 Android 系统手机的新闻不断，越来越多的开发者更清晰地看到了，在 Android 平台上进行软件和服务开发的必要性和广阔前景。

作者历时半年完成本书，并通过通俗的文字、翔实的示例，由浅入深地向读者详细介绍 Android 系统的概念、开发流程以及实际的演示例子。更难能可贵的是，作者还通过和真实生活联系紧密的例子，把 Android 开发与当下流行的地图服务、Web 2.0、SNS 等应用联系起来，给读者展示了使用 Android 进行丰富开发的诸多可能。本书结构合理，行文流畅，示例丰富，是一本很好的 Android 开发入门和实战书籍。

—Google Android 测试工程师 何峰

未来可以预见，基于 Android 的应用需求会越来越多。对于软件从业人员来说，Android 平台的一个吸引人之处是，它包括了开发移动设备应用所需的操作系统、用户界面设计等软件，且没有任何专有权方面的障碍，这也就意味着任何人都可以在其上进行开发。书中提供了一些完整的应用实例和源代码分析，如豆瓣网客户端、在线音乐播放器等，对于想快速上手 Android 的朋友有很大的帮助。

—InfoQ 中文站 (infoq.com/cn) 总编辑 霍泰稳

Android 开放平台是手机炫酷应用最好的实验场之一，与此同时，Android 平台所具有的发展潜力，也必将成为应用开发者创造价值的源泉。在 Android 应用中，出现了很多体验性很好的程序，如果你也希望借助 Android 平台，以全新的方式和富有个性的创意来诠释掌中的移动应用，那么请不要犹豫翻开这本书，她将引导你用自己的创意去征服 Android 手机用户。

—中国科学院 博士 高昂

基于 Android 平台的 G1、G2 手机也已经开始销售，中国移动推出的基于 Android 系统的 OPhone 即将上市。许多人已经在关注这个新的市场了，Android 无疑扮演了寻找这个市场金矿入口的角色，本书的面市恰逢其时，她将对程序员开发个性彰显的 Android 程序产生积极的影响。

—P1.cn 技术总监 蔡望勤

这本书通俗易懂，书中实例把传统互联网的内容与移动平台紧密结合，体现了很好的创新精神，很值得读者阅读。

—支付宝 数据库架构师 冯大辉

在我印象里，本书作者是个乐于探索新知、对技术潮流敏锐的程序员。如今 Android 手机操作系统刚刚出现，他又开始勇于创新地去实践，不仅开发了一系列 Android 应用，更用自己积累的经验和技术写出了这本实战教程，书中实例丰富、案例生动有趣，很值得读者学习。

—Thought Works 咨询师 IT 技术专家 熊节

前言

本书是一本教你开发赚钱的 3G 移动互联网应用的书！

2009 年 1 月 7 日，工业和信息化部为中国移动、中国电信和中国联通发放 3 张第三代移动通信（3G）牌照，此举标志着我国正式进入 3G 时代。中国 3G 正式商用和规模建网，App 商店在全球被运营商和终端厂商热捧：中国移动 App 商店将不久上线，这些背景，共同决定了中国基于 Android 的应用程序开发将在未来处于一个爆发和蓬勃成长的时期，这也为软件开发者提供了一个淘金的好机会，也迫使大量此前不熟悉 Android 开发的程序员要迅速进入此领域，以便在经历了单机计算时代和传统互联网时代之后，能在移动互联网时代的元年开始赶上这趟车，成为了许多程序员的愿望。

Android 平台由互联网与社会信息科技的领袖 Google 公司开发，由 Google 和二十多家业界重量级企业联盟提供持续的技术支持。Google 的开放、自由精神，Android 联盟的雄厚实力，以及 App 商店商业模式带来的巨大活力，使得在 IT 界迅速出现了一大批热爱、追随 Android 平台的开发人员和产品设计人员，为了帮助他们更好地学习 Android 技术，特意创作了本书。

本书内容讲解上深入浅出，实例紧贴市场。在技术理论的讲解方面，本书注重对实际动手能力的指导，在遵循技术研发知识体系的严密性同时，在容易产生错误、不易理解的环节都配以了翔实的开发情景截图，并将重要的知识点、开发技巧以“小实验”、“小提醒”、“小知识”、“注意”等活泼的形式呈现给读者。在程序实例的讲解方面，主要将实例安插在 Android 开发的精髓知识章节，也是最难学习的部分，这为初学者学用结合提供了很好的指导。书中所有案例均来自于实际开发项目，具有很高的商业价值，5 大综合案例分别是 RSS 阅读器、基于 Google Map 的个人 GPS、豆瓣网（Web 2.0）客户端、在线音乐播放器、手机信息助手。这些案例充分体现了移动互联网应用的创新精神，将很好地帮助初学者尽快融入实战角色。

本书内容

笔者在移动终端软件开发以及终端与服务器交互技术方面均有着多年的开发经验，积累了深厚的技术基础，在写本书的过程中，又和来自 Google、微软公司、中国移动、澳洲电信、韩国电信、联想移动等众多电信、互联网行业的移动互联网专家多次切磋，进行了很多交流。本书包括 17 章，主要内容如下。

第1章 掀起你的盖头来——初识 Android。本章介绍了 Android 的来龙去脉,让读者以最短的时间直观地了解到 Android 的基础及发展的历史和趋势,并能清晰地知道自己可以在 Android 上做什么事情。

第2章 工欲善其事 必先利其器——搭建 Android 开发环境。本章介绍了在 Windows、Linux、Mac OS 平台上搭建 Android 开发环境的过程,通过本章的学习,可以很顺利地搭建起自己的开发环境,做好开发前的准备。

第3章 清点可用资本——Android SDK 介绍。主要介绍了 Android SDK 中包含的内容,了解用 SDK 可以干什么。通过本章的学习,可以清晰地知道 SDK 中包含了哪些技术以及用途。

第4章 赚钱的市场——Android Market 及应用发布。本章介绍了与 Android Market 相关的知识、一些特色应用及快速创建自己的 Market 账号。通过本章的学习,了解整个 Market 的使用过程、拥有自己的 Market 账号。

第5章 千里之行 始于足下——第一个应用 HelloWorld。讲解了用 Android 开发的第一个实例,从这里开始,带领读者进入 Android 编程的世界。通过本章的学习,读者了解了 Android 开发是怎么个流程。

第6章 磨刀不误砍柴工——Android 应用程序结构介绍。介绍了 Android 程序的组成部分,让读者搞清楚整个程序是怎么组成,怎么运作的。通过本章的学习,使得读者可以比较清楚地了解 Android 系统的架构组成部分。

第7章 良好的学习开端——Android 基本组件介绍。介绍了 Android 提供的一些组件,使得读者可以很快地掌握并使用这些组件,这是以后开发的基础。通过本章的学习,可以掌握常见组件的使用方法和技巧,为后续快速开发提供帮助。

第8章 移动信息仓库——Android 的数据存储操作。介绍了 Android 上可以使用的几种存储方式以及每种方式适应的场景,使得读者可以很快掌握数据存储操作。通过本章的学习,将学会何时需要存储数据,采用哪种存储方式,以及如何实现存储。

第9章 我来“广播”你的“意图”——Intent 和 Broadcast 面对面。介绍了 Android 提供的 Intents 和 Broadcast 的使用,让读者更清楚地了解 Android 内部的技术,掌握内部消息的流转和控制。

第10章 一切为用户服务——Service 应用实例。介绍了 Android 中 Service 的原理和使用方式,用两个精彩的例子讲述了如何调用系统的 Service 或者创建自己的后台 Service 的应用技术和技巧。

第 11 章 循序渐进——开发 Android 应用的基本步骤。本章整理、归纳了整个 Android 开发的基本流程和步骤，使得读者可以按照这个流程更快速、更合理地开发自己的应用。

第 12 章 Android 综合案例一——RSS 阅读器实例。本章以开发一个网络阅读器为实战案例，实现的功能是读取网络上的 RSS 数据，并对这些数据进行解析、显示。另外，还着重介绍了 XML 和 SAX 的解析方式。通过本章的学习，读者可以掌握如何读取互联网上的数据，完成基于服务端和客户端的应用开发技术。

第 13 章 Android 综合案例二——基于 Google Map 开发个人移动地图。本章通过讲述开发一款个人 GPS 位置跟踪、记录，并在 Google Map 上实时显示的一个应用为主线，介绍了 Google API、LBS 等技术的实现。通过本章的学习，读者可以掌握 Google API 的使用方式和基于位置服务的应用开发技术。

第 14 章 Android 综合案例三——基于 Android 的豆瓣网（Web2.0）移动客户端开发。讲述了基于豆瓣网（Web 2.0）API 的 Android 客户端实现的应用，通过豆瓣网的用户鉴权，读取该用户的一些信息，例如，友邻广播等。通过本章的学习，可以很快掌握如何为 Web2.0 网站开发其对应的客户端软件的技术。

第 15 章 Android 综合案例四——在线音乐播放器。本章讲述了一款多媒体应用的开发，其基于 YOBO API 开发的一款在线音乐播放器。实现了可以读取用户在 YOBO 的个人信息、音乐盒、音乐电台等，以及对其音乐盒中歌曲的在线播放等功能。通过本章的学习，可以掌握网络数据读取、解析、显示和多媒体的播放等开发技术。

第 16 章 Android 综合案例五——手机信息查看助手。本章讲述了一款手机信息查看器的应用开发，用此应用实例可以查看手机的硬件、软件，以及运行时的一些信息，是一款针对单机的应用。通过本章的学习，可以掌握手机工具类底层应用的开发技术。

第 17 章 芝麻开门——Android 底层开发和移植概述。

本书由靳岩、姚尚朗主编，参与资料整理和代码调试的还有靳辉、杨琨、刘婷、周波、陈良、刘玲、王雪明、靳琼、王亮、何雪等朋友，在此一并感谢！附录内容见光盘。

本书适合我吗

当您走进书店，看到书的标题中熟悉的字眼“Android”，想了解这本书是

否适合自己时，下面的提示对您的选购很有帮助：

- 您听说过 Google 出了一款手机操作系统吗？
- 您听说过 Android 吗？
- 您知道 Android Market 吗？
- 您听说过中国移动推出的 OMS 吗？

如果上述问题中有一个以上是肯定的，可以很高兴地告诉您，拿在手中的这本书确实是这个方向上的，下面需要进一步确认：

- 您对软件开发有经验或者有兴趣吗？
- 您对 Java 语言有了解吗？
- 您做过手机应用开发吗？
- 您是 Android 爱好者吗？

如果上述问题，您的回答中有肯定的，那么您已经具备了阅读本书需要的基础，不用担心读不懂了，那么：

- 您想快速了解并进入 Android 应用开发吗？
- 您想找到一本系统介绍 Android 开发的参考资料吗？
- 您想选择一本有原理剖析又有真实例子演示的教材吗？
- 您想选一本通俗易懂，符合自己阅读习惯的图书吗？

如上问题中，如果您有大多数回答都是肯定的，那么非常恭喜您，现在拿着的这本书差不多正是您需要的，可以放心地带回去开始自己的 Android 之旅了。

如果还在犹豫，那么让下面几个提示告诉您，尽早开始学习的重要性：

- 国内 3G 牌照发布后推动了移动互联网的快速发展，而 Android 是一个非常强大的手机开发平台，其让您可以快速切入无线互联网领域；
- 在 Android Market 发布应用的数量在快速增长，早日发布可以体现自己的开发价值和乐趣；
- 中国移动将要推出的 OMS 是基于 Android 的，您开发的 Android 应用可以适用于 OMS 系统；
- 掌握了 Android 开发就可以很快开发出供全球 Android 用户使用的应用，有人已经在 Android Market 赚到钱了！

本书适合

- 具备一定软件开发经验，想快速进入 Android 开发领域的程序员；

- 为了个人兴趣和职业技能储备而想要学习 Android 的爱好者；
- 具备一些手机开发经验的独立开发者、业余爱好者；
- 软件技术企业或培训学校的培训教材。

本书特色

- 本书专门针对中国开发者撰写，更加符合国内开发者的阅读和编程习惯，内容更容易被接受和消化。
- 本书基于最新的 Android SDK 1.5，所有的例子可以在 Android SDK 1.5 的环境中运行。
- 内容涵盖了 Android 开发的大部分场景，从 Android 开发环境搭建、SDK 介绍、Market 申请和使用，到 Android 应用剖析、常见组件介绍、实例演示等方面。
- 技术实现上，包括了 RSS 阅读器、基于位置服务的 Google Map 地图应用、Web 2.0 网站移动客户端、音乐多媒体以及手机信息查看 5 大综合实战案例。
- 本书配套有 400 多分钟的全程开发视频光盘，指导读者快速、无障碍地学通 Android 实战开发技术。
- 本书笔者毫无保留地将上传到 Google Market 里边的开发实例直接放在书中和读者共享。帮助读者多角度地学习到更多的实战技能。

本书约定

本书遵循如下约定。

符号和术语	含 义	示 例
小知识	用于表示对编程以外的技术拓展	小知识： 什么是 FrameLayout？ 一个 FrameLayout 对象就好比一块在屏幕上提前预订好的空白区域，然后可以填充一些元素到里边，比方说一张图片等
小实验	表示对上面讲解知识的使用和编程验证	小实验： 上边的这种布局，不需要运行程序，直接在 Eclipse 里边就可以看到结果显示。所以用 XML 进行布局是非常方便的

续表

符号和术语	含 义	示 例
代码解释	对一段代码进行功能和技术 的总结性说明	代码解释： startActivityForResult() 这个方法启动 Intent，并且当新的 Activity 运行结束的时候，还必须执行原 Activity 里边的回调函数，这个函数叫做 onActivityResult()
学习目标	关键知识点的总结内容	学习目标： • 了解什么是 Service • 如何使用 Service • 学习 Service 的生命周期 • 掌握实例一 定时提醒 • 掌握实例二 音乐播放
注意	对一个技术应用特意的提醒	注意： Android 中的 Alarms 只是在其设置的时间点到达的时候给系统发一个 Alarm 消息唤醒系统，用户不会听到响铃或者感受到振动，如果需要响铃或者振动，则需要在接收到 Alarm 消息时自己播放一段声音或者启动振动

支持网站和社区

本书的支持社区为 eoeAndroid，通过网址 <http://www.eoeAndroid.com> 就可以进行访问。eoeAndroid 是关于 Android 开发的专业社区，如果大家在具体开发中有任何的疑问可以登录社区和广大网友交流。另外，我们在社区里边专门开辟了一个专版来解决读者提出的关于阅读本书当中的问题，如果有勘误或者更新也都会发到这个专版上。

致谢

靳岩（海阳|haiyangjy）

作为一部原创 Android 开发书籍，这本书的写作是比较艰苦的。一方面关于 Android 所有的原始资源和文档都是国外的，所以学习 Android 本身就不是很容易

易，我们在开发一些 Android 应用的时候，都是在国外的论坛进行讨论。另外一方面，我们对自己严格要求，希望能给用户带来一本精品读物，在 Android 的 SDK 从 1.0 升级到 1.1，从 1.1 升级到 1.5 的时候，我们对书中的内容和实例都进行了重新修订，以保证书中所有例子都可以运行在最新的 SDK 上。

虽然写这本书是辛苦的，甚至很多时候是带病写作，但是写这本书也是开心的，也是满怀希望的。在此我特别感谢我的家人，我的爸爸和妈妈，他们理解我的工作，支持我；感谢我的姐姐，经常给我鼓励和激励。另外，要感谢我的好朋友，也是本书的另外一位作者姚尚朗同学。他虽然不幸骨折，但是仍然单手击打键盘进行写作，让我十分敬佩。还要感谢好朋友靳辉同学，是他的加入才让我们把更多的精力投入到本书的编撰之中。感谢杨琨同学，你的加入使得 eoeMobile 团队更加强大。感谢我的母校北京邮电大学，在北京邮电大学我学会了思考问题、分析问题和解决问题的能力。感谢“北邮四人组”，特别是王亮和可可同学，是你们让我有更好的状态写作此书。感谢张欣师兄，和你的每次交谈我都能学到很多知识。

感谢我以前的领导 Tony（陈有杰），感谢你对我们团队的支持以及对本书的推广所做的一切。

感谢我母校的老师，网秦公司副总邹仕洪先生为本书写的推荐序。感谢您对 eoeMobile 团队的支持和鼓励。

我代表 eoeMobile 团队要特别感谢人民邮电出版社的编辑张涛，没有他的热情推动，这本书可能不会这么快地展现在读者面前。感谢人民邮电出版社刘涛副社长，他的谦和和实干精神给我们留下了深刻的印象。另外，我也要特别感谢人民邮电出版社市场部产品经理李大微，是你让更多的读者有机会接触到此书。

最后感谢给本书写推荐语的朋友们，他们是何峰、霍泰稳、高昂、蔡望勤、高焕堂、冯大辉、熊节。谢谢你们的长期支持与鼓励。

姚尚朗（IceskYsl）

虽然在写书之前已经听到无数的人说写一本书是要花费大量时间和精力，我深有感触，特别是想力求写出一本让读者满意的作品的时候，其写作过程中需要承受的压力和历经的辛苦是外人无法体会的，我也无法清晰传达，只能在内心深深体味着。然而，写完本书后，回想一下历时半年的编写过程，我才感到原来有这么多人帮助一直支撑着我写作下去，希望在这里向他们表示谢意。这其中有个我无法忘记的过程，当书稿内容编写一半的时候，在一次运动中不慎将右手腕摔骨折，绑上石膏的我当时心情特别复杂。尽管编写过程是用笔写，但是右手手腕骨折，至少 1~2 个月无法使用的情况下，我

对撰写的进度非常担忧。幸运的是，我克服了这个困难，为了更快恢复健康，我从寒冷的北京回到深圳，期待在温暖的气候下，可以尽快恢复健康，这个决定是正确的，1 个月的时间我就可以拆除石膏，进行功能恢复训练，并及时地恢复到写作的过程中。在此期间，我更加感受到朋友们的支持和鼓励，特别要谢谢女朋友 shine，她的照顾让我更有动力写完本书。另外还要谢谢父母，为了争取进度，上一个春节都没有回家团聚，等到忙完了，一定回家住一段时间。

本书的社区支持联盟

作为一本 Android 原创技术书籍，我们充分吸取 Android 社区反馈的意见和建议，积极修订和完善，力求使得本书能更符合开发者的阅读习惯和使用要求，感谢这些为 Android 发展做出贡献的社区，感谢其对本书的完善做出的积极贡献，社区列举如下，排名不分先后。



编 者

目 录

第 1 章 掀起你的盖头来——初识 Android	1
1.1 认识 Android	1
1.2 Android 的背景	2
1.2.1 Android 的历史	2
1.2.2 Android 的发展	2
1.3 我的 Android 我做主	2
1.3.1 开发基于 Android 平台的应用	3
1.3.2 参加 Android 开发者大赛	3
1.3.3 个人英雄主义再现——得到更多人的认可和尊重	3
1.3.4 获得应有的收益——Android Market	3
1.4 真实体验——Android 模拟器	4
1.4.1 模拟器概述	4
1.4.2 模拟器和真机的区别	4
1.4.3 模拟器使用注意事项	4
1.5 更上一层楼——加入 Android 开发社区	5
1.6 本章小结	6
第 2 章 工欲善其事 必先利其器——搭建 Android 开发环境	7
2.1 开发 Android 应用前的准备	7
2.1.1 Android 开发系统要求	7
2.1.2 Android 软件开发包	7
2.1.3 其他注意事项	8
2.2 Windows 开发环境搭建	8
2.2.1 JDK、Eclipse、Android SDK 软件安装	8
2.2.2 SDK 的家在哪里——设定 Android SDK Home	14
2.2.3 真的准备好了吗——开发环境验证	14
2.2.4 创建 Android 虚拟设备 (AVD)	15
2.3 Linux 一族——Ubuntu 开发环境搭建	17
2.3.1 Java、Eclipse 和 ADT 插件安装	17
2.3.2 设定 Android SDK Home	23
2.4 Mac OS 一族——苹果开发环境搭建	24
2.5 本章小结	24
第 3 章 清点可用资本——Android SDK 介绍	25
3.1 Android SDK 基础	25
3.2 深入探寻 Android SDK 的密码	25

3.2.1	Android SDK 目录结构	25
3.2.2	android.jar 及内部结构	27
3.2.3	SDK 文档及阅读技巧	27
3.2.4	先来热热身——Android SDK 例子解析	28
3.2.5	SDK 提供的工具介绍	31
3.3	Android 典型包分析	33
3.3.1	开发的基石——Android API 核心开发包介绍	33
3.3.2	拓展开发外延——Android 可选 API 介绍	34
3.4	本章小结	34
第 4 章	赚钱的市场——Android Market 及应用发布	35
4.1	Google Market 产生背景与目的	35
4.2	体验“选货”的乐趣——在 G1 上体验 Market 的使用	35
4.3	Android 开发活动及特色应用	37
4.3.1	开发应用的领域	37
4.3.2	Android Market 特色应用一览	38
4.4	你也可以做东家——申请 Market 账号	43
4.4.1	卖东西要先入伙——准备工作	43
4.4.2	入伙过程——申请	44
4.5	开张了——在 Market 上发布应用	45
4.5.1	发布时可能遇到的错误	45
4.5.2	卖东西也要签名——生成签名文件	46
4.5.3	打包、签名、发布应用	48
4.6	本章小结	51
第 5 章	千里之行 始于足下——第一个应用 HelloWorld	52
5.1	HelloWorld 应用分析	52
5.1.1	新建一个 Android 工程	52
5.1.2	填写工程的信息	52
5.1.3	编程实现	53
5.1.4	运行项目	54
5.2	调试项目	54
5.2.1	设置断点	54
5.2.2	Debug 项目	55
5.2.3	断点调试	55
5.3	本章小结	56

第 6 章	磨刀不误砍柴工——Android 应用程序结构介绍	57
6.1	Android 体系结构介绍	57
6.1.1	应用程序 (Application)	57
6.1.2	应用程序框架 (Application Framework)	58
6.1.3	库 (Libraries) 和运行环境 (RunTime)	58
6.2	Android 应用程序组成	59
6.2.1	Activity 介绍	59
6.2.2	Broadcast Intent Receiver 介绍	60
6.2.3	Service 介绍	61
6.2.4	Content Provider 介绍	61
6.3	Android 应用工程文件组成	61
6.4	本章小结	62
第 7 章	良好的学习开端——Android 基本组件介绍	63
7.1	第一印象很重要——界面 UI 元素介绍	63
7.1.1	视图组件 (View)	63
7.1.2	视图容器组件 (Viewgroup)	63
7.1.3	布局组件 (Layout)	64
7.1.4	布局参数 (LayoutParams)	64
7.2	我的美丽我做主——Android 中应用界面布局	64
7.2.1	实例操作演示	65
7.2.2	实例编程实现	66
7.3	不积跬步 无以至千里——常用 widget 组件介绍	75
7.3.1	创建 widget 组件实例	75
7.3.2	按钮 (Button) 介绍与应用	76
7.3.3	文本框 (TextView) 介绍与应用	77
7.3.4	编辑框 (EditText) 介绍与应用	79
7.3.5	多项选择 (CheckBox) 介绍与应用	81
7.3.6	单项选择 (RadioGroup) 介绍与应用	83
7.3.7	下拉列表 (Spinner) 介绍与应用	85
7.3.8	自动完成文本 (AutoCompleteTextView)	87
7.3.9	日期选择器 (DatePicker) 介绍与应用	89
7.3.10	时间选择器 (TimePicker) 介绍与应用	90
7.3.11	滚动视图 (ScrollView) 介绍与应用	91
7.3.12	进度条 (ProgressBar) 介绍与应用	92
7.3.13	拖动条 (SeekBar) 介绍与应用	93

7.3.14	评分组件 (RatingBar) 介绍与应用	94
7.3.15	图片视图 (ImageView) 介绍与应用	95
7.3.16	图片按钮 (ImageButton) 介绍与应用	96
7.3.17	切换图片 (ImageSwitcher&Gallery)	96
7.3.18	网格视图 (GridView) 介绍与应用	99
7.3.19	标签 (Tab) 介绍与应用	101
7.4	友好的菜单——menu 介绍与实例	102
7.4.1	实例操作演示	103
7.4.2	实例编程实现	103
7.5	Android 应用的灵魂——Intent 和 Activity 介绍与实例	106
7.5.1	实例操作演示	106
7.5.2	实例编程实现	106
7.6	用好列表, 做好程序——列表 (ListView) 介绍与实例	111
7.6.1	实例程序演示	111
7.6.2	实例编程实现	112
7.7	友好地互动交流——对话框 (Dialog) 介绍与实例	119
7.8	温馨的提醒——Toast 和 Notification 应用	127
7.8.1	实例操作演示	128
7.8.2	实例编程实现	129
7.9	本章小结	135

第 8 章	移动信息仓库——Android 的数据存储操作	136
8.1	Android 数据存储概述	136
8.2	轻轻地我保护——SharedPreferences 存储	136
8.3	谁的文件, 谁主宰——文件存储	140
8.4	打造自己的数据库存储——SQLite 存储方式	141
8.4.1	Android 中对数据库操作	141
8.4.2	完整地操作数据库——日记本实例	147
8.5	我的数据你来用——ContentProvider 介绍	155
8.5.1	初识 ContentProvider	155
8.5.2	使用 ContentProvider 读取系统数据	156
8.5.3	使用 ContentProvider 操作数据日记本实例	159
8.6	再学一招——网络存储	171
8.7	本章小结	173

第 9 章	我来“广播”你的“意图”——Intent 和 Broadcast 面对面	174
9.1	Android 应用程序的核心——Intent	174
9.1.1	Intent 基础	174
9.1.2	用 Intent 启动一个新的 Activity	174
9.1.3	Intent 详细讲解	177
9.1.4	Android 解析 Intent 实现	179
9.2	用广播告诉你——利用 Intent 来广播(Broadcast)事件	180
9.2.1	实现 Android 中的广播事件	180
9.2.2	Broadcast Receiver 介绍	181
9.3	应用实例详解	181
9.3.1	程序操作演示	182
9.3.2	实例编程实现	182
9.4	本章小结	186
第 10 章	一切为用户服务——Service 应用实例	187
10.1	认识 Service	187
10.2	使用 Service	188
10.3	Service 的生命周期	194
10.4	实例学习 Service	194
10.4.1	精彩实例一——定时提醒	194
10.4.2	精彩实例二——音乐播放器	198
10.5	本章小结	201
第 11 章	循序渐进——开发 Android 应用的基本步骤	202
11.1	兵马未动 粮草先行——应用规划及架构设计	202
11.2	应用开发步骤	202
11.2.1	界面设计始终是第一位——实现 UI	203
11.2.2	必备的动力源泉——数据操作和存储	203
11.2.3	华丽转身——实现多页面跳转	203
11.2.4	始终为用户做好服务——增加 Service	203
11.2.5	细节决定成败——完善应用细节	203
11.3	成功就在眼前——应用测试和发布	204
11.3.1	只欠东风——应用测试	204
11.3.2	可以赚钱了——发布到 Android Market	204
11.4	本章小结	204

第 12 章	Android 综合案例一——RSS 阅读器实例	205
12.1	RSS 介绍	205
12.1.1	RSS 基础	205
12.1.2	RSS 的历史	205
12.1.3	RSS 语法介绍	206
12.2	SAX 介绍	207
12.2.1	SAX 基础	207
12.2.2	使用 SAX 的作用	207
12.2.3	怎样使用 SAX	207
12.3	RSS 阅读器设计	209
12.3.1	RSS 阅读器功能设计	209
12.3.2	RSS 阅读器 UI 和交互流程设计	209
12.4	RSS 阅读器的实现	210
12.4.1	程序实体解析	210
12.4.2	实现一个自己的 ContentHandler	214
12.4.3	Activity 的实现	217
12.5	本章小结	220
第 13 章	Android 综合案例二——基于 Google Map 开发 个人移动地图	221
13.1	项目 UI 规划	221
13.2	数据存储实现	222
13.2.1	设计数据库及表结构	222
13.2.2	设计 SharedPreferences 存储	223
13.3	项目实现流程	223
13.3.1	创建项目工程	223
13.3.2	项目各功能及界面实现	224
13.3.3	实现数据存取	247
13.3.4	实现 Service	252
13.3.5	应用流程整体实现	254
13.3.6	后续开发完善	255
13.4	项目程序演示	255
13.5	项目程序签名、打包和发布	257
13.6	本章小结	259

第 14 章	Android 综合案例三——基于 Android 的豆瓣网 (Web 2.0)	
	移动客户端开发	260
14.1	关于豆瓣网和豆瓣网 API	260
14.1.1	豆瓣网介绍	260
14.1.2	豆瓣网 API 介绍	261
14.1.3	豆瓣网 API 认证	261
14.1.4	豆瓣网 API 快速入门	264
14.1.5	豆瓣网 API Java 库介绍	265
14.2	豆瓣网 (Web 2.0) 客户端设计	268
14.2.1	客户端功能规划设计	268
14.2.2	UI 和交互流程设计	268
14.2.3	数据存储设计	270
14.3	豆瓣网 (Web 2.0) 客户端实现	270
14.3.1	申请 API Key	270
14.3.2	Activity 实现	271
14.4	本章小结	282
第 15 章	Android 综合案例四——在线音乐播放器	283
15.1	关于 YOBO 和 YOBO API	283
15.1.1	YOBO 简介	284
15.1.2	YOBO 功能特点	284
15.1.3	YOBO API 介绍	284
15.2	在线播放器架构设计	285
15.2.1	在线播放器功能规划设计	285
15.2.2	在线播放器 UI 设计	285
15.2.3	在线播放器数据存储设计	286
15.3	在线播放器的编程实现	286
15.3.1	申请 API Key	286
15.3.2	基础功能实现	287
15.3.3	实现 Activity	290
15.3.4	实现 Service	306
15.3.5	后续发展展望	310
15.4	在线音乐播放器应用演示	310
15.5	在线音乐播放器应用打包、签名和发布	311
15.6	本章小结	311

第 16 章	Android 综合案例五——手机信息查看助手	312
16.1	手机信息查看助手功能规划和可行性分析	312
16.1.1	手机信息查看助手功能规划	312
16.1.2	手机信息查看助手可行性分析	312
16.2	手机信息查看助手功能实现	313
16.2.1	手机信息查看助手主界面	313
16.2.2	查看系统信息实现	316
16.2.3	查看硬件信息	320
16.2.4	查看软件信息	324
16.2.5	获取运行时信息	326
16.2.6	文件浏览器	328
16.2.7	项目的细节完善	331
16.3	手机信息查看助手功能展望	332
16.4	项目的打包、签名和发布	332
16.5	本章小结	333
第 17 章	芝麻开门—Android 底层开发和移植概述	334
17.1	Android 原始码下载	334
17.2	实机测试	335
17.2.1	Neo FreeRunner 介绍	335
17.2.2	烧录 androidfs.jffs2	336
17.3	Android 移植技术概论	337
17.3.1	Android 底层技术的重要性	337
17.3.2	Android 移植项目介绍	338
17.3.3	Android 的技术优点	340



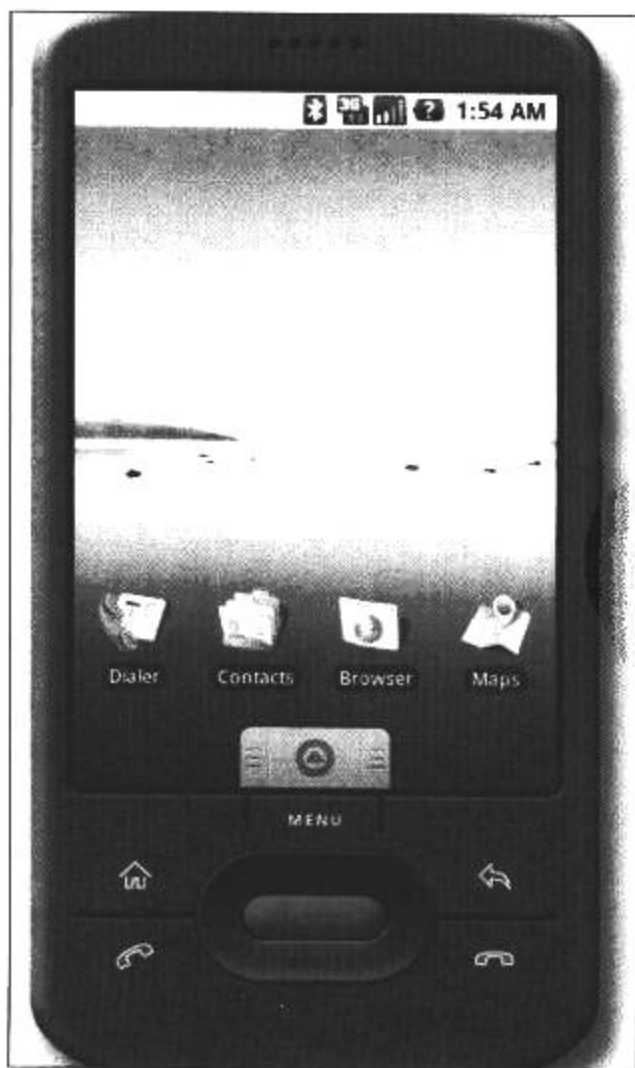
第1章 掀起你的盖头来——

初识 Android

1.1 认识 Android

Android 一词的英文本义指“机器人”，它是 Google 公司于 2007 年 11 月宣布的基于 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。

简单的理解，Android 是基于 Java 并运行在 Linux 内核上的操作系统，这个操作系统是轻量级的，但是功能却很全面。而且在这个系统上，Google 公司已经内置了很多有用的软件，如打电话、发短信等。图 1-1 所示是 Android 的模拟器，从图上可以大致了解 Android 的运行界面。



▲ 图 1-1 Android 模拟器

1.2 Android 的背景

1.2.1 Android 的历史

为了更好地了解 Android，有必要对其历史进行一些讲解。谈到 Android，那首先需要了解的是“开放手机联盟”，其英文名称为 Open Handset Alliance，是美国 Google 公司于 2007 年 11 月宣布组建的一个全球性的联盟组织。这一联盟将会支持 Google 公司发布的手机操作系统或者应用软件，共同开发名为 Android 的开放源代码的移动操作系统。

这个联盟成员数量已经达到了几十家，除了前面提到的 Google 公司外，还包括手机制造商、手机芯片厂商和移动运营商几类。

1.2.2 Android 的发展

从开放手机联盟成立以后，Android 的发展也加快了速度，如下选取几个比较关键的路标来回顾整个 Android 的发展历程。

(1) 开放手机联盟成立。

2007 年 11 月 5 日，34 个联盟成员宣布成立开放手机联盟。

(2) 发布第一版 Android SDK。

2007 年 11 月 12 日宣布发布第一版 Android SDK。

(3) Android 开发者竞赛。

2008 年 4 月 17 日举办 Android 开发竞赛，在规定的时间内提交了 1788 件作品，推动了 Android 开发的应用速度。

(4) Android Market 上线。

2008 年 8 月 28 日，为 Android 平台手机提供软件分发和下载的 Market 正式上线，迅速积累了大量的应用。

(5) T-Mobile G1 上市。

2008 年 9 月 23 日，第一款基于 Android 的手机上市，美国运营商 T-Mobile USA 在纽约正式发布第一款 Google 手机——T-Mobile G1。该款手机为宏达电制造，是世界上第一部使用 Android 操作系统的手机，支持 WCDMA/HSPA 网络，理论下载速率为 7.2Mbit/s，并支持 Wi-Fi。

(6) Android 1.0 SDK release 1 发布。

2008 年 9 月 23 日，Android 1.0 R1 SDK 发布，标志着 Android 系统趋于稳定和成熟，越来越多的开发者加入 Android 开发阵营中。

(7) Android 被宣布开放源代码。

2008 年 10 月 21 日，Android 被宣布开放源代码。

通过了解上述 Android 发展路线，在短短一年多的时间内，其发展速度是惊人的，除了 Android 平台发展外，也带动了周边厂商的发展，如硬件设备、软件 Market 等。

1.3 我的 Android 我做主

了解了 Android 的前世今生，也了解了它将对移动开发带来的影响，作为随时需要跟踪新技术

的程序员不禁要问：我们在 Android 上可以做什么？

1.3.1 开发基于 Android 平台的应用

在 Android 平台上，可以开发出各式各样的应用。Android 的应用程序是用 Java 语言开发的，也就是说，只要会 Java 语言就可以比较容易地开发出 Android 的应用。如果不了解 Java 也没有关系，通过一段时间对 Java 语言的学习，也可以很快进入 Android 开发领域。加上 Android 平台提供了 2D、3D 的图形支持、数据库支持（SQLite），并且集成了浏览器。基于 Android 的平台，我们可以开发出丰富多彩的应用，这些应用可以涉及工具、管理、互联网、游戏等，在此开放的平台，这一切都取决于程序员的自由发挥和创意。

1.3.2 参加 Android 开发者大赛

Google 公司为了吸引更多的开发者参与到 Android 开发中，已经举办了一次奖金为 1000 万美元的开发者竞赛，如果能开发出创意十足，非常有用的软件，在以后 Google 公司或者其他公司举办的开发者大赛中，可以一显身手。

1.3.3 个人英雄主义再现——得到更多人的认可和尊重

在这个开放的平台，任何程序员开发的应用程序都可以放在 Android Market 上，供所有的 Android 手机用户下载体验，用户都可以对此应用加以评价和使用，这是一个展示个人能力和技术魅力的平台。软件开发的个人英雄主义时代将在 Android 移动平台上完美地体现出来，程序员完全可以凭自己的能力，开发一款得到更多赞誉的应用。

1.3.4 获得应有的收益——Android Market

开发出的应用给谁用，怎么提供给别人，程序员能从中得到什么好处呢？这些问题都可以从“Android Market”中得到解答。为了让 Android 平台更多地吸引开发者关注和应用，Google 公司开发了自己的 Android 手机应用软件下载店“Android Market”，它允许研发人员将应用程序在其上发布，当然了，也允许 Android 用户随意将自己喜欢的程序下载到自己的手机设备上。简单地说，Android Market 就是一个软件大市场，作为用户，你可以随时用手机连上去下载自己需要的应用例子；作为开发者，只要申请了“开发者”账号，可以随时将自己的应用上传到 Android Market 上，供其他人下载、使用。而这其中，你是可以给自己开发的软件定价的。

如此一来，只要程序员开发的软件足够吸引人，让用户有足够的理由掏钱购买，那你就获取很好的回报，这也就是本书的目的之一——开发赚钱的 3G 应用，帮助读者尽快进入 Android 开发实战中，更快地按照自己的想法开发出精彩的应用软件。

关于 Android Market 的介绍及其账号的申请、使用，将会在第 4 章讲解，这里不再赘述，Android Market 展示如图 1-2 所示。



▲ 图 1-2 Android Market 展示界面

1.4 真实体验——Android 模拟器

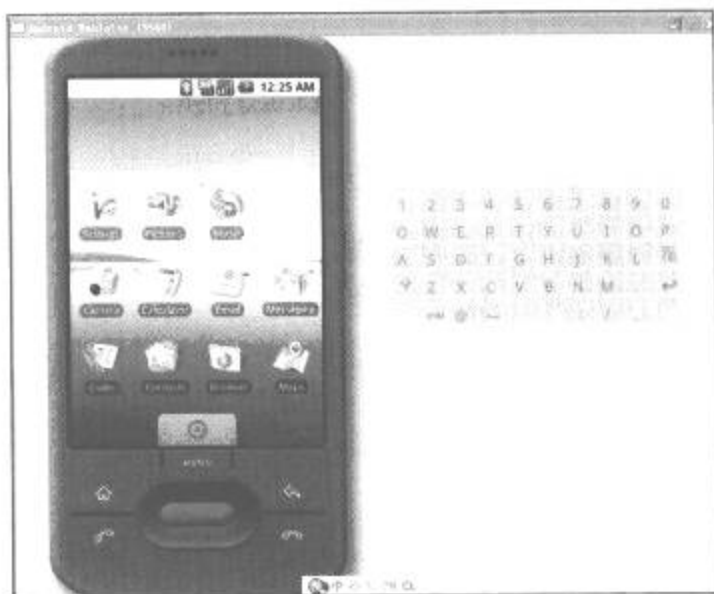
1.4.1 模拟器概述

图 1-3 所示是模拟器的全景图，从图中可以看出模拟器有两部分组成，左边部分模拟手机显示，右边部分模拟手机的键盘输入。在模拟手机的部分内置了一些 Google 公司自带的程序，如打电话、发短信、Google Map 等程序。

1.4.2 模拟器和真机的区别

Android 的模拟器功能很强，但是模拟器仍然只是尽量去模拟手机，有些手机的功能还是模拟不了，下边做一个简单的说明。

- 模拟器不支持呼叫和接听实际来电，但可以通过控制台模拟电话呼叫（呼入和呼出）。
- 模拟器不支持 USB 连接。
- 模拟器不支持相机/视频捕捉。
- 模拟器不支持音频输入（捕捉），但支持输出（重放）。
- 模拟器不支持扩展耳机。
- 模拟器不能确定连接状态。
- 模拟器不能确定电池电量水平和充电状态。
- 模拟器不能确定 SD 卡的插入/弹出。
- 模拟器不支持蓝牙。



▲ 图 1-3 模拟器全景

1.4.3 模拟器使用注意事项

在用模拟器调试开发的应用程序时，常见的有以下两条需要特别注意的问题，现在介绍给读者，以便避免大家再走弯路。

平时使用模拟器 Emulator 测试开发时，可能会遇到计算机提示，系统 C 盘空间不足之类的信息，这是由于 Android 模拟器每次运行时临时生成几个 tmp 后缀的临时文件，一段时间后，其可能占用几 GB 的磁盘空间。所以，需要手工定期清理一下，清理的目录为 C:\Documents and Settings\“用户名”\Local Settings\Temp\AndroidEmulator 文件夹下的内容。笔者有一次在开发当中也遇到了类似的问题。审查程序没有发现什么问题，但是启动模拟器时总启动不起来，后来发现原来 C 盘的剩余空间为零。删去临时文件后，模拟器运行正常。

如果使用 Eclipse 开发工具进行调试，那么第一次运行程序，并启动模拟器的时间是比较长的，大概需要将近一分钟。为了解决启动时间过长的时间，当模拟器启动后，每次运行新的程序不需要关闭旧的模拟器，而是直接在 Eclipse 开发工具里边单击运行即可。

1.5 更上一层楼——加入 Android 开发社区

如果想要快速成为 Android 开发高手，加入 Android 开发者社区是必不可缺的步骤。下边介绍了 Android 官方的社区，以及中国的专业应用开发社区。

通过访问网址 <http://www.Android.com/>，可以访问 Android 的官方网站，在这里可以上传你的应用，查看开发文档，搜索开发资源等，具体界面如图 1-4 和图 1-5 所示。



▲ 图 1-4 开发社区主页



▲ 图 1-5 开发社区可用资源

上面网站中的内容是英文的,对于更多的中国开发者来说,我们极力推荐大家加入 Android 中文专业应用开发社区,可以通过网址 <http://www.coeandroid.com/> 访问,在这里可以获取新的学习资料,开发资源等,如图 1-6 和图 1-7 所示。



▲ 图 1-6 专业的中文 Android 开发社区 (1)



图 1-7 专业的中文 Android 开发社区（2）

1.6 本章小结

本章从介绍 Android 的产生和发展开始，让读者了解了 Android 产生的背景，Android 提供了哪些功能，开发人员可以在此平台上做的事情。通过简单地对 Android 的了解，读者可以对它的应用前景有一个很好的认识。



第2章 工欲善其事 必先利其器—— 搭建 Android 开发环境

学习目标

- 搭建 Android 开发环境的步骤和注意事项; ☐
- Windows 开发环境搭建; ☐
- Linux/Ubuntu 开发环境搭建; ☐
- Mac OS 开发环境搭建。 ☐

2.1 开发 Android 应用前的准备

2.1.1 Android 开发系统要求

开发基于 Android 的应用软件所需要的开发环境如下表所示。

开发系统需求参数

所需项	版本要求	说明	备注
操作系统	Windows XP 或 Vista Mac OS X 10.4.8+ Linux Ubuntu Drapper		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, 其最新版是 1.5
IDE	Eclipse IDE+ADT	Eclipse3.3 (Europa), 3.4 (Ganymede)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 不可以的, 必须要有 JDK), 不兼容 Gnu Java 编译器(gcj)

注: 推荐使用 Eclipse + ADT(Android Development Tools Plug-in) 来开发。

2.1.2 Android 软件开发包

- JDK 1.6, 请到网址 <http://java.sun.com/javase/downloads/index.jsp> 处下载。
- Eclipse 3.4 (Europa), 请到网址 <http://www.eclipse.org/downloads/> 下载 Eclipse IDE for Java Developers。
- Android SDK1.5, 请到网址 <http://developer.android.com> 下载。

- APT 插件。

2.1.3 其他注意事项

Android 的 Eclipse 插件安装需注意, Eclipse 可以到对应的网站下载安装, 如果通过网络远程安装不成功, 可以下载到本地安装。

2.2 Windows 开发环境搭建

首先介绍在 Windows 平台上搭建 Android 开发环境的步骤和注意事项, 其具体步骤如下。

2.2.1 JDK、Eclipse、Android SDK 软件安装

本书所讲的安装是以 Windows XP SP2 为平台, 安装的软件为 JDK 1.5、Eclipse 3.3、ADT1.5、Android SDK。下面具体介绍各自的安装步骤, 因为本书的重点在 Android 开发上, 所以, 安装的步骤在书中就不详细讲了, 我们在配套的视频中有详细的介绍, 如果大家还有疑问可以到 <http://eoeandroid.com> 开发社区查询相关资料。

1. 安装 JDK

安装 Eclipse 的开发环境需要 JRE 的支持, 如果没有 JRE, 则启动 Eclipse 时会报告如图 2-1 所示的错误。

在 Windows 上安装 JRE/JDK 非常简单, 首先在 Sun 官方网站下载, 网址为 <http://developers.sun.com/downloads/>, 如图 2-2 所示。



▲ 图 2-1 没有 JRE 时的报错信息界面



▲ 图 2-2 Sun 官方下载页面

在图 2-2 中可以看到有很多版本,运行 Eclipse 时虽然只需要 JRE 就可以了,但是在开发 Andriod 应用程序的时候,是需要完整的 JDK (JDK 包含了 JRE),且要求其版本在 1.5+ 以上,这里选择 Java SE (JDK) 6,其下载页面如图 2-3 所示。



▲ 图 2-3 JDK 下载页面

在图 2-3 中看到“JDK 6 Update 13”,单击其右侧的“Download”按钮,出现让用户选择其操作系统和语言的界面,如图 2-4 所示。

这里选择 Windows,并且选中“I agree to the Java SE Development Kit 6 License Agreement”,然后单击“Continue”按钮转到下载页面,在下载页面可以看到需要下载的安装文件“jdk-6u13- windows-i586-p.exe”,如图 2-5 所示。

单击下载链接,保存软件即可,如图 2-6 所示。

下载得到的是 exe 安装文件,双击此安装文件后打开安装向导,接受许可证,如图 2-7 所示。

选择需要安装的组件和安装到的路径,如图 2-8 所示。



▲ 图 2-4 选择操作系统和语言界面



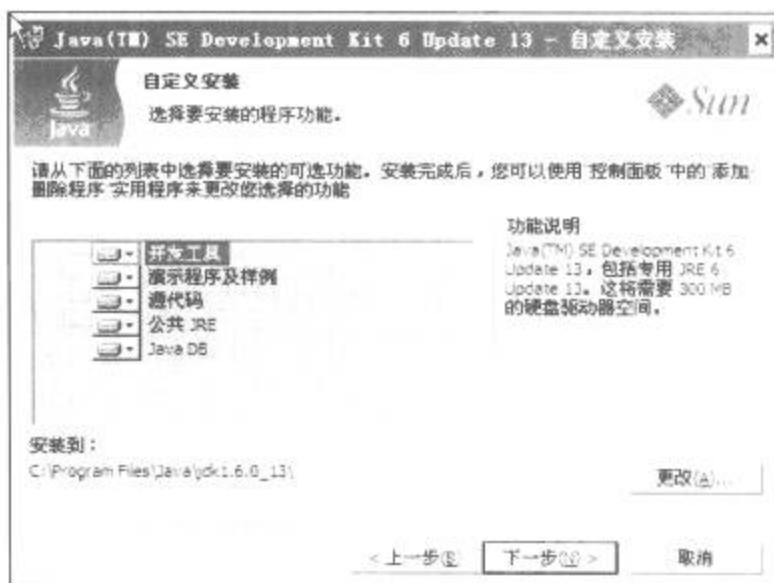
▲ 图 2-5 下载文件



▲ 图 2-6 保存文件



▲ 图 2-7 接受许可证



▲ 图 2-8 选择组件和安装路径

然后单击“下一步”按钮，就可以自动完成安装，这里不做赘述，最后检查以下是否安装成功。

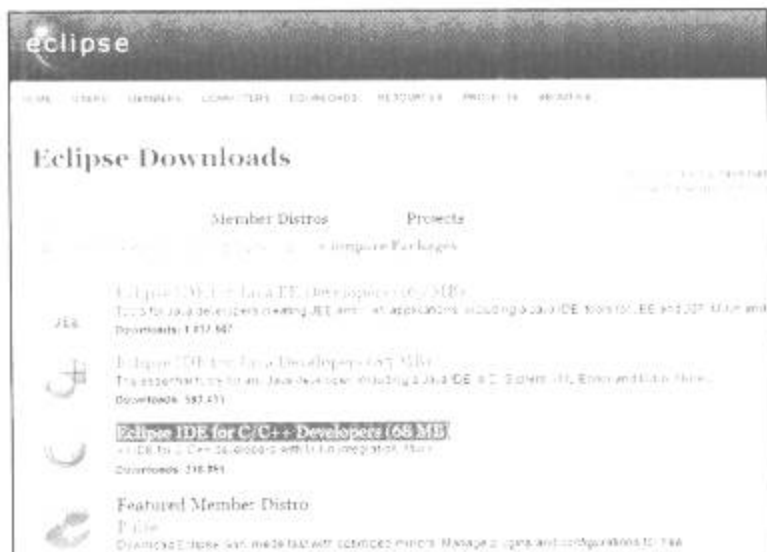
单击“开始”，选择“运行”，在出现的对话框中输入 CMD 命令，打开 CMD 窗口，在 CMD 窗口中输入 `java -version`，如果屏幕出现下面的代码信息，说明 JDK 安装成功了：

```
C:\Documents and Settings\Administrator>java -version
java version "1.6.0_13"
Java(TM) SE Runtime Environment (build 1.6.0_13-b03)
Java HotSpot(TM) Client VM (build 11.3-b02, mixed mode, sharing)
```

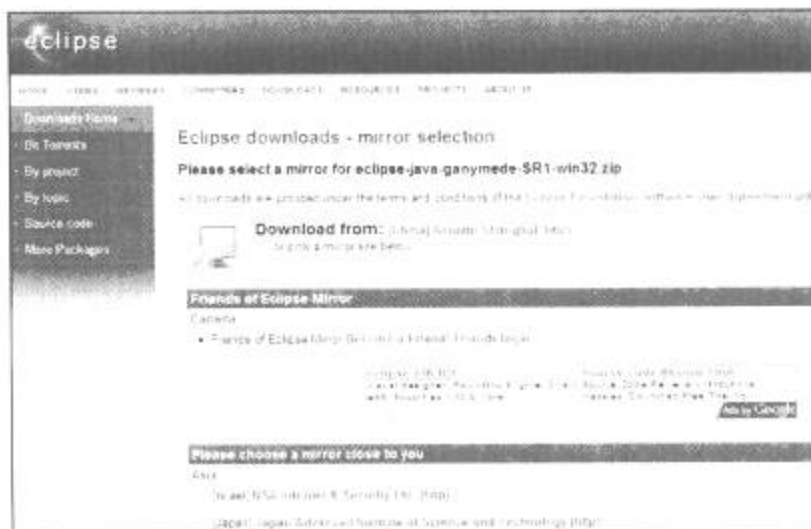
2. 安装 Eclipse

安装好 JDK 后，就可以接着安装 Eclipse 了，打开 Eclipse 的下载页面，具体网址为 <http://www.eclipse.org/downloads/>，如图 2-9 所示。

在图 2-9 中选择“Eclipse IDE for Java Developers (85 MB)”，在其下载的镜像页面，选择离用户最近的镜像即可（一般推荐的下载速度就不错），如图 2-10 所示。



▲ 图 2-9 Eclipse 的下载页面

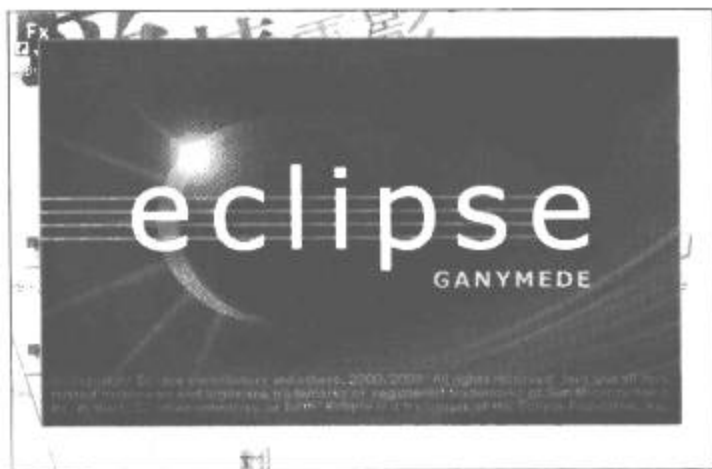


▲ 图 2-10 Eclipse 的镜像下载网站

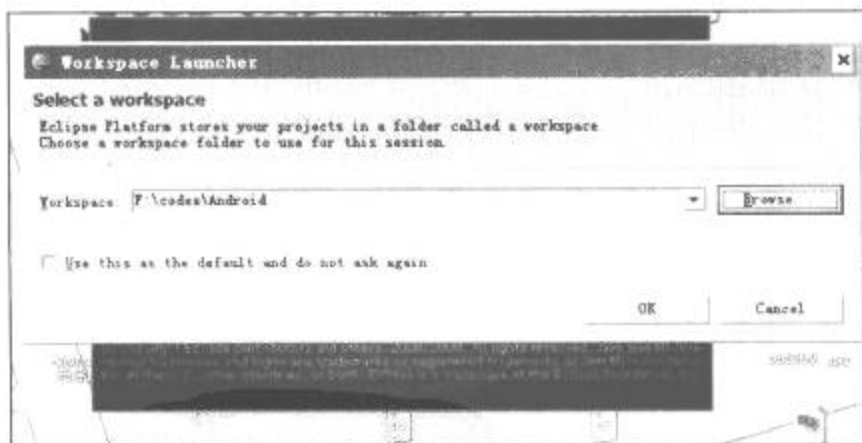
下载完成后，然后找到下载的压缩包“eclipse-java-ganymede-SR1-win32.zip”。Eclipse 无须执行安装程序，解压此压缩文件就可以用，不过一定要先安装 JDK。本文假定 Eclipse 解压后存放的目录为 D:\unsetup\eclipse。

进入解压后的目录，可以看到一个名为“eclipse.exe”的可执行文件，双击此文件直接运行，Eclipse 能自动找到用户先期安装的 JDK 路径，启动界面如图 2-11 所示。

如果用户是第一次安装、启动 Eclipse，将会看到选择工作空间的提示，如图 2-12 所示。



▲ 图 2-11 启动 Eclipse 界面



▲ 图 2-12 选择工作空间

选择工作空间路径，然后单击“OK”按钮，至此，JDK 和 Eclipse 已经安装完毕。

3. 安装 Android SDK

安装好 JDK 和 Eclipse 后，下面需要下载安装 Android 的 SDK，首先打开 Android 开发者社区，对应的网址为 <http://developer.android.com/>，转到 SDK 下载页面（http://developer.android.com/sdk/1.5_r1/index.html），如图 2-13 所示。

在图 2-13 中可以看到，当前最新版本是 2009 年 4 月发布的“Download Android 1.5 SDK, Release 1”，这里选择适应于 Windows 平台的“android-sdk-windows-1.5_r1.zip”，打开下载页面，如图 2-14 所示。



▲ 图 2-13 Andriod SDK 选择页面



▲ 图 2-14 SDK 下载页面

选中 “I agree to the terms of the Android SDK License Agreement.”，单击 “Download” 按钮开始下载，同 Eclipse 一样，不需要安装，解压之后就可以使用，这里把下载后的文件解压存放在 D:\unsetup\android-sdk-windows-1.5_r1 目录下，并将其 tools 目录的绝对路径添加到系统的 PATH 中，打开一个 CMD 窗口，输入 android -h，如果有如下返回则表示 SDK 安装成功：

```
C:\Documents and Settings\Administrator>android -h

Usage:
  android [global options] action [action options]

Global options:
  -h --help      This help.
  -s --silent    Silent mode: only errors are printed out.
  -v --verbose   Verbose mode: errors, warnings and informational messages are printed.
```

4. 安装 ADT

Android 为 Eclipse 定制了一个插件，即 Android Development Tools (ADT)，目的是为用户提供一个强大的综合环境用于开发 Android 应用程序。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，在基于 Android 框架 API 的基础上添加组件，以及用 SDK 工具集调试应用程序，甚至导出签名（或未签名）的 APKs 以便发行应用程序。

安装 Android Development Tools plug-in，打开 Eclipse IDE。依次单击 Menu 项中的 Help → Software Updates 项，如图 2-15 所示。

在弹出的对话框中，选择“Available Software”标签。单击“Add Site...”，输入“http://dl-ssl.google.com/Android/eclipse/”，单击“OK”按钮，如图 2-16 所示。



▲ 图 2-15 Software Updates 界面



▲ 图 2-16 Add Site 界面

然后就可以看到可用的插件，如图 2-17 所示。

在图 2-17 中选择“Android Developer Tools”和“Android DDMS”，单击“install”按钮，然后就可以看到其安装界面，如图 2-18 所示。



▲ 图 2-17 查看可用的插件



▲ 图 2-18 安装界面

安装完成后，会提示用户重启 Eclipse。



注意

不同版本的 Eclipse 安装插件的方法和步骤是不同的，如上演示的是基于 Eclipse 3.4 版本的，如果使用的是 3.3 版本的 Eclipse，那么其步骤大致如下。

- (1) 启动 Eclipse，依次选择 Help → Software Updates → Find and Install...项。
- (2) 在弹出来的对话框中选择“Search for new features to install”项，然后单击“Next”按钮。
- (3) 单击“New Remote Site”项。
- (4) 在弹出的对话框中，给新的站点取个名字（如 Android Plugin），输入 <http://dl-ssl.google.com/Android/eclipse/>后单击“OK”按钮。
- (5) 然后可以看到刚刚添加的站点已经加入搜索的列表，选择其前面的选择框，单击“Finish”按钮。
- (6) 在接下来的操作中，依次选择 Android Plugin → Developer Tools 项，这样会选中“Android

Developer Tools”和“Android DDMS”，然后单击“Next”按钮。

(7) 阅读其协议，选择“Accept terms of the license agreement”，单击“Next”按钮。

(8) 最后单击“Finish”按钮。

(9) 然后会出现“The ADT plugin is not signed; you can accept the installation anyway”，单击“Install All”按钮。

(10) 重启 Eclipse 即可。

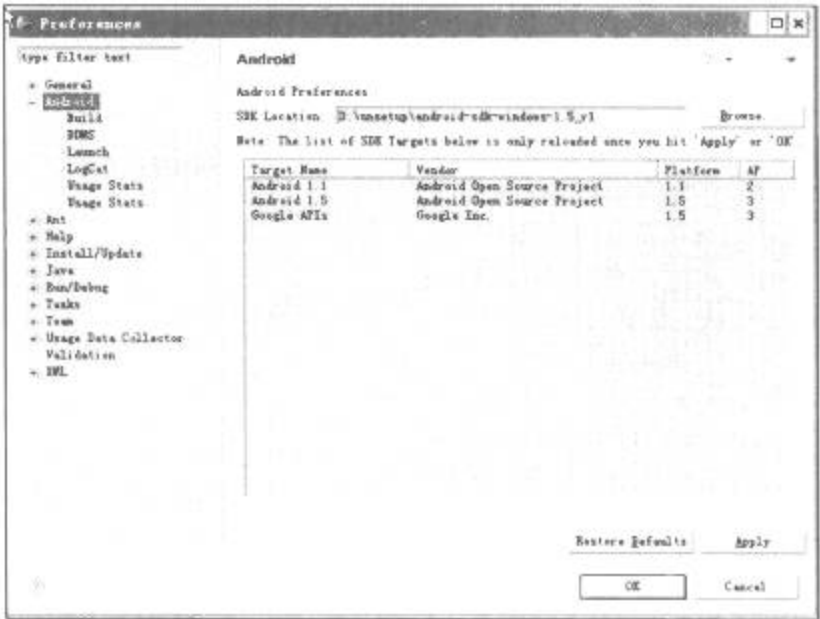
2.2.2 SDK 的家在哪里——设定 Android SDK Home

安装好插件后，还需要做如下配置才可以使用 Eclipse 创建 Android 项目，需要设置 Android SDK 主目录。依次单击“windows→Preferences”项，如图 2-19 所示。

在弹出的界面左侧可以看到“Android”项，选中 Android 后，在右侧设定 Android SDK 所在目录为 SDK Location，单击“OK”按钮完成安装，如图 2-20 所示。



▲ 图 2-19 单击“Preferences”界面



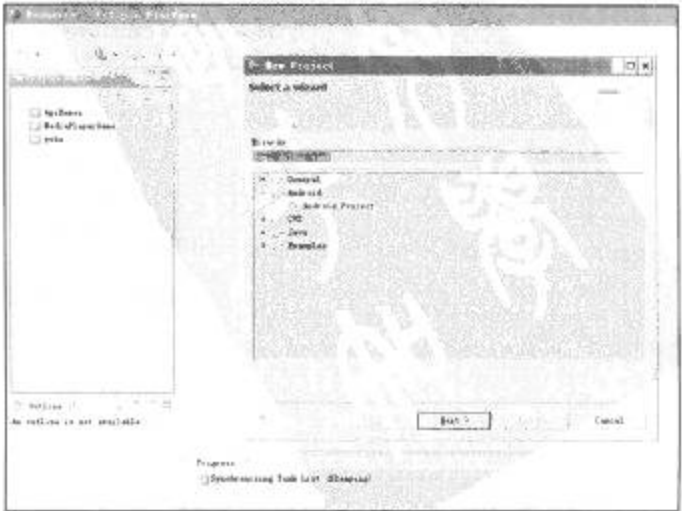
▲ 图 2-20 设置 SDK 路径界面

2.2.3 真的准备好了吗——开发环境验证

上述操作步骤至此，开发环境搭建完成，下面需要新建一个项目来验证环境是否可以正常工作，依次选择“File→New→Project”项，在弹出的对话框上可以看到 Android，如图 2-21 所示。

在图 2-21 上选择“Android”，单击“Next”按钮，打开“New Android Project”对话框，在对应的文本框中输入必要的信息，如图 2-22 所示。

这里不再具体说明项目信息中各项的意义，后面章节会详细介绍。单击“Finish”按钮，会自动完成项目的创建工作，最后将可以看到如图 2-23 所示的

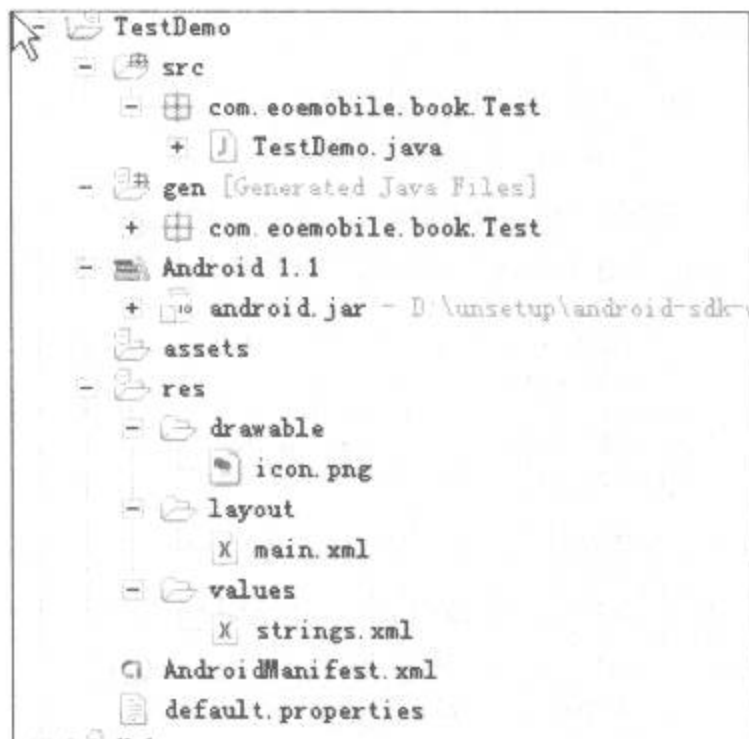


▲ 图 2-21 出现 Android 选项界面

项目结构。



▲ 图 2-22 输入项目信息界面



▲ 图 2-23 目录结构界面

通过上述操作，已经在 Windows 平台上搭建好了开发环境。

2.2.4 创建 Android 虚拟设备 (AVD)

在 Android SDK1.5 版以后的 Android 开发中，必须创建至少一个 AVD，AVD 全称为 Android 虚拟设备 (Android Virtual Device)，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。

由于 Android SDK1.5 以后支持多个平台和外观显示，作为开发者创建不同的 AVD 来模拟和测试不同的平台环境，创建 AVD 方法如下：

(1) 在 CMD 下输入 `android list targets` 查看可用的平台。

```
C:\Documents and Settings\Administrator>android list targets
Available Android targets:
id: 1
  Name: Android 1.1
  Type: Platform
  API level: 2
  Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id: 2
  Name: Android 1.5
  Type: Platform
  API level: 3
  Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id: 3
```

```

Name: Google APIs
Type: Add-On
Vendor: Google Inc.
Description: Android + Google APIs
Based on Android 1.5 (API level 3)
Libraries:
  * com.google.android.maps (maps.jar)
    API for Google Maps
Skins: QVGA-P, HVGA-L, HVGA (default), QVGA-L, HVGA-P

```

如上列举了 3 个 targets, id 分别是 1、2、3。

(2) 创建 AVD。

按照 “android create avd --name <your_avd_name> --target <targetID>” 格式创建 AVD, 其中 “your_avd_name” 是需要创建的 AVD 的名字, 如下:

```

C:\Documents and Settings\Administrator>android create avd --name sdk_1_5_version
--target 2
Android 1.5 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
Created AVD 'sdk_1_5_version' based on Android 1.5

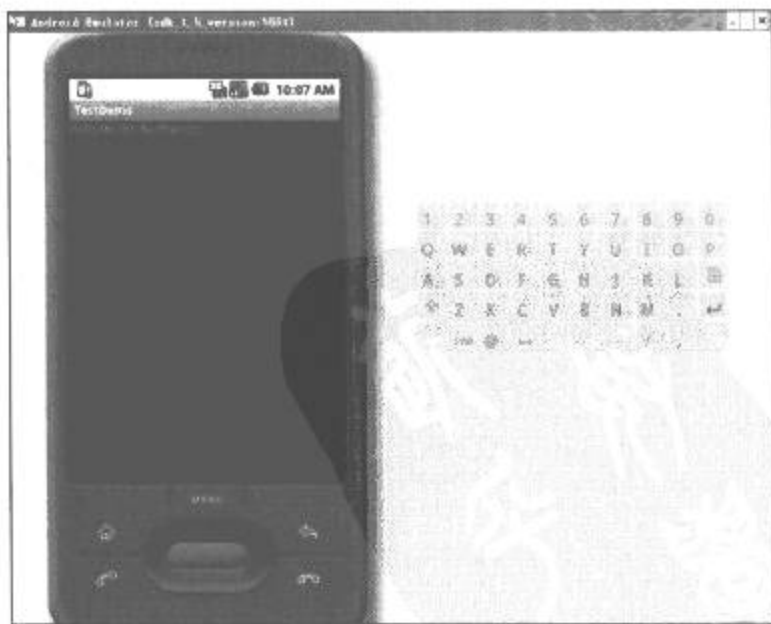
```

这样就创建了一个自定义的 AVD(Android Virtual Device), 然后, 只要在 Eclipse 的 Run Configurations 里面指定一个 AVD, 即在 Target 下选中自己定义的这个 AVD, 即 sdk_1_5_version 就可以运行了, 如图 2-24 所示。

如上我们选择 “sdk_1_5_version”, 单击 “Apply” 按钮后, 单击 “Run” 按钮将运行模拟器, 如图 2-25 所示, 关于应用结构分析和讲解, 以及代码的调试部分内容在本书第 5 章将会详细介绍。



▲ 图 2-24 选择 AVD



▲ 图 2-25 指定 AVD 的模拟器

至此, 在 Windows 平台上的开发环境搭建完成, 安装了运行环境 JDK, 开发工具 Eclipse, Android SDK, 并安装了 ADT 并进行 SDK Home 的配置, 最后创建了一个 Android 虚拟设备 (AVD)。

2.3 Linux 一族——Ubuntu 开发环境搭建

介绍完了在 Windows 平台搭建 Android 开发环境后,接下来讲解如何在 Linux 平台上搭建开发环境,这里采用 Ubuntu 8.04 Desktop 版本作为演示示例。

2.3.1 Java、Eclipse 和 ADT 插件安装

1. Java 和 Eclipse 安装

软件安装是在 Ubuntu 8.04 Desktop 上进行,安装 JDK 1.5、Eclipse 3.4、ADT、Android SDK,在主目录 /home/iceskysl 这里使用 iceskysl 账号,读者可以更换适合自己的账户应用。

在开始安装 Android SDK 之前,必须先安装 Java 和 Eclipse (version 3.3 或 3.4),在下面我们使用 Eclipse 3.4 版的“Ganymede”)进行安装演示,具体步骤如下。

首先,我们开启一个终端(Terminal)窗口(依次执行 Applications → Accessories → 终端项),如图 2-26 所示。

首先安装 Java,在安装中输入下列 JDK 命令,如图 2-27 所示。



▲ 图 2-26 打开 Terminal 界面



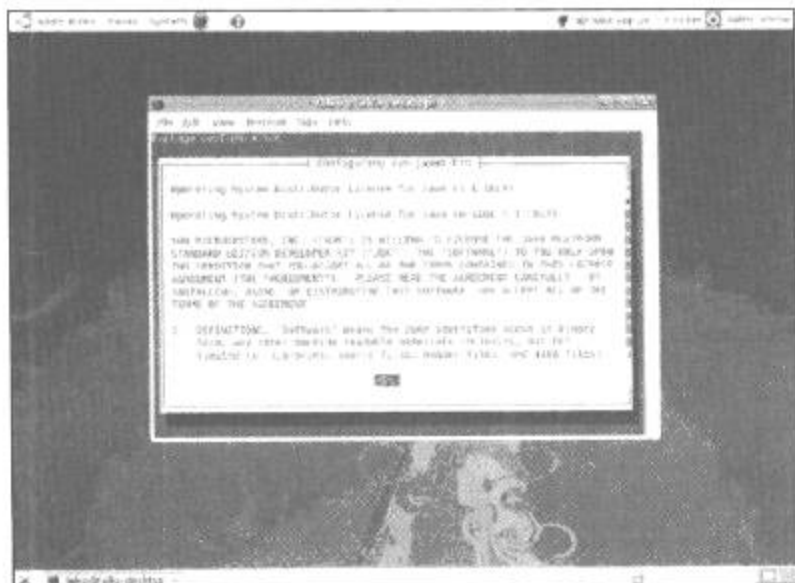
▲ 图 2-27 输入安装 JDK 命令

输入的 TDK 命令: `sudo apt-get install sun-java6-bin`

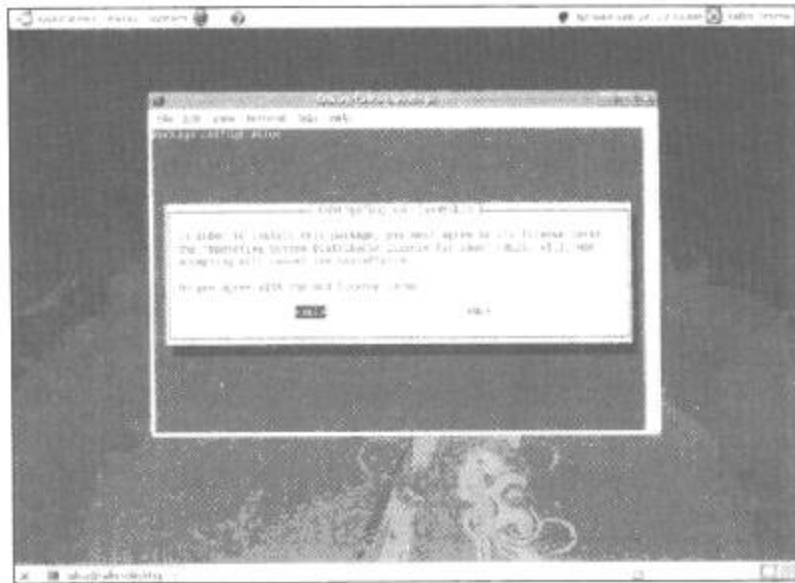
注: 如果读者是在一个 x86_64 系统上运行,也必须安装 `ia32-libs`: `sudo apt-get install ia32-libs`。

在安装时将被询问是否接受 Java license,如图 2-28 和图 2-29 所示。

Android SDK 需要 Eclipse 3.3 以后的版本,在撰写本书稿时,Ubuntu 8.04 只包含 Eclipse 3.2,所以必须手动安装 Eclipse。



▲ 图 2-28 是否接受 Java license



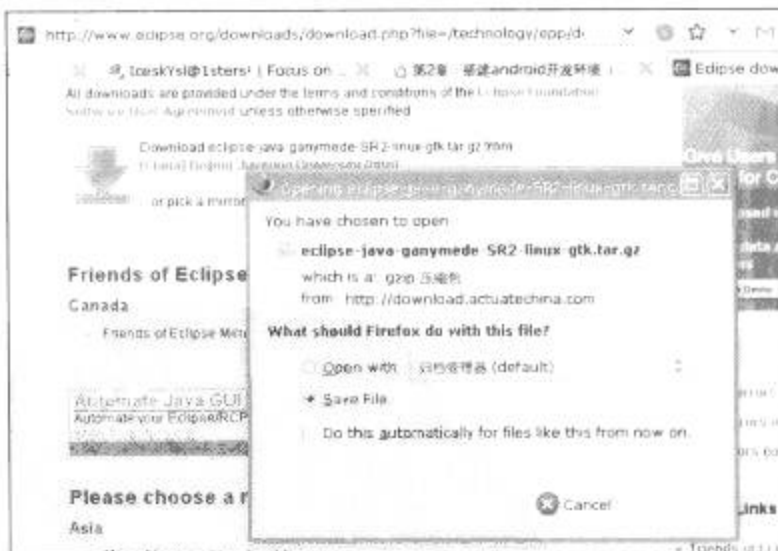
▲ 图 2-29 接受 Java license

打开火狐浏览器，在地址栏中输入网址 <http://www.eclipse.org/downloads/>，在出现的页面上选择一个 Java、Java EE 或 RCP 版本的 Eclipse，这里选择 Eclipse IDE for Java EE Developers for Linux 32bit 下载，如果读者用的是 x86_64 系统的，请选择 Linux 64bit，如图 2-30 所示。

保存这个下载下来的压缩文件到硬盘，如保存到桌面 (/home/iceskysl/Desktop) 中，如图 2-31 所示。



▲ 图 2-30 下载 Eclipse



▲ 图 2-31 保存文件

然后开始安装 Eclipse，我们必须再次开启一个终端，转换到下载文件所存放的目录，然后解压缩现在的文件：

```
iceskysl@IceskYsl:~/Desktop$ cd /home/iceskysl/Desktop/
iceskysl@IceskYsl:~/Desktop$ tar -zxvf eclipse-java-ganymede-SR2-linux-gtk.tar.gz
```

这将建立一个叫做 *Eclipse* 的文件夹，然后，可以删除 Eclipse 所存放的压缩文件，删除命令

如下：

```
rm -f eclipse-java-ganymede-SR2-linux-gtk.tar.gz
```

本文演示的 Eclipse 资料文件夹存放在我的桌面（`/home/iceskysl/Desktop`）上，若不想放在那里，可以移动这个文件夹到我的主目录内（也可以搬离到任何你喜欢的地方），移动命令为：

```
mv eclipse ~
```

在 Eclipse 文件夹内，有一个可执行的文件叫做 `eclipse`，这个文件是用来启动 Eclipse 时所必须执行的。很显然的，若不想用命令方式启动 Eclipse，就需要运行它。在开始菜单中选择“设置”，然后选择“主菜单”项，如图 2-32 所示。

在弹出的窗口中选择“编程”（任何你想执行的类别项），然后单击“新建项目”按钮，如图 2-33 所示。

在弹出的窗口中的名称文本框上输入应用名称（如 Eclipse），然后在命令文本框中输入可执行 Eclipse 的完整路径（如 `/home/iceskysl/eclipse/eclipse`，所存放文件的地方），然后按“确定”按钮，如图 2-34 所示。



▲ 图 2-32 编辑菜单



▲ 图 2-33 主菜单窗口



▲ 图 2-34 创建启动器

现在可以执行 Eclipse 了（依次单击开始→编程→Eclipse），在 Linux 上 Eclipse 的启始和在 Windows 上启动差不多，如果是第一次启动 Eclipse，会要求选择一个工作区，如图 2-35 所示。

选择工作区后，Eclipse 即可完成启动，其启动的界面如图 2-36 所示。

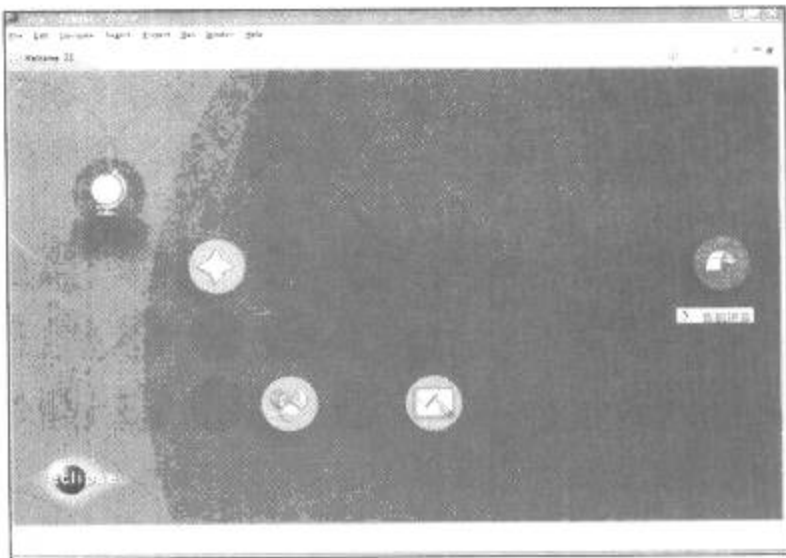
至此，已经完成了 JDK 和 Eclipse 的安装，接着安装 Android SDK。

2. 安装 Android SDK

下载 Android SDK，在浏览器网址栏中输入 `http://developer.android.com/sdk/1.1_r1/index.html`，在出现的下载页面中，选择“`android-sdk-linux_x86-1.1_r1.zip`”，如图 2-37 所示。



▲ 图 2-35 选择工作区



▲ 图 2-36 Eclipse 启动界面

在下载页面中，选择 Linux (i386) 下载，如图 2-38 所示。



▲ 图 2-37 Android SDK 下载页面



▲ 图 2-38 选择下载的版本

把下载下来的文件储存在到硬盘（如桌面）上，如图 2-39 所示。

现在再开启一个终端，然后找到储存下载得到 Android SDK 的文件夹，解压缩此下载的文件，解压缩命令为：

```
cd /home/iceskysl/Desktop/  
unzip android-sdk-linux_x86-1.1_r1.zip
```

加压缩文件完成以后，已经不需要 zip 文件了，所以可以将它删除，删除命令为：

```
rm -f android-sdk-linux_x86-1.1_r1.zip
```

在所储存文件的桌面上，现在有一个文件夹，名字是 Android-sdk-linux_x86-1.1_r1，若不想把此文件放在桌面那里，可以将这个目录移动到主目录上，



▲ 图 2-39 保存下载的文件

移动命令为:

```
mv Android-sdk-linux_x86-1.1_r1/ ~
```

Android-sdk-linux_x86-1.1_r1 目录包含子目录, 名字是 tools (/home/iceskysl/Android-sdk-linux_x86-1.1_r1/tools)。现在我们必须开启 ~/.bashrc, 执行语句如下:

```
gedit ~/.bashrc
```

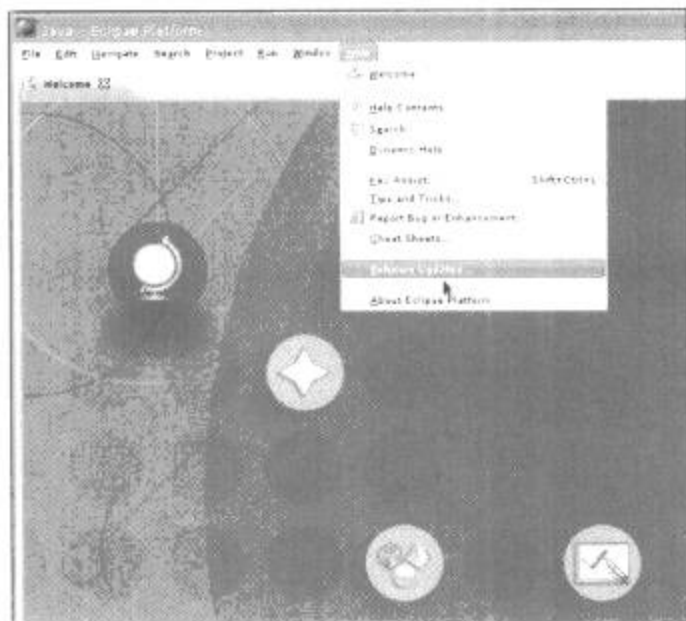
然后加入以下语句, 以便让 Android-sdk-linux_x86-1.1_r1/tools 是在我们设置的 PATH (路径) 上:

```
export PATH=${PATH}:/home/iceskysl/Android-sdk-linux_x86-1.1_r1/tools
```

3. 安装 ADT 插件

接下来的操作如下, 安装 Android Eclipse 开发插件, 其安装过程和 Windows 平台上一样。在 Eclipse 内依次选择 Help→Software Updates...项, 如图 2-40 所示。

转换到 Available Software 标签, 然后单击 “Add Site...” 项, 如图 2-41 所示。



▲ 图 2-40 Software Updates 界面



▲ 图 2-41 “Add Site” 界面

Add Site 界面打开后, 在地址文本框中输入 <http://dl-ssl.google.com/Android/eclipse/>, 然后单击 “Install” 按钮, 如图 2-42 所示。



▲ 图 2-42 添加 ADT 地址

返回到“Available Software”页签项，选择“Developer Tools”（这应该是自动检查 Android Development Tools 和 Android Editors），然后单击“Install...”项，如图 2-43 所示。



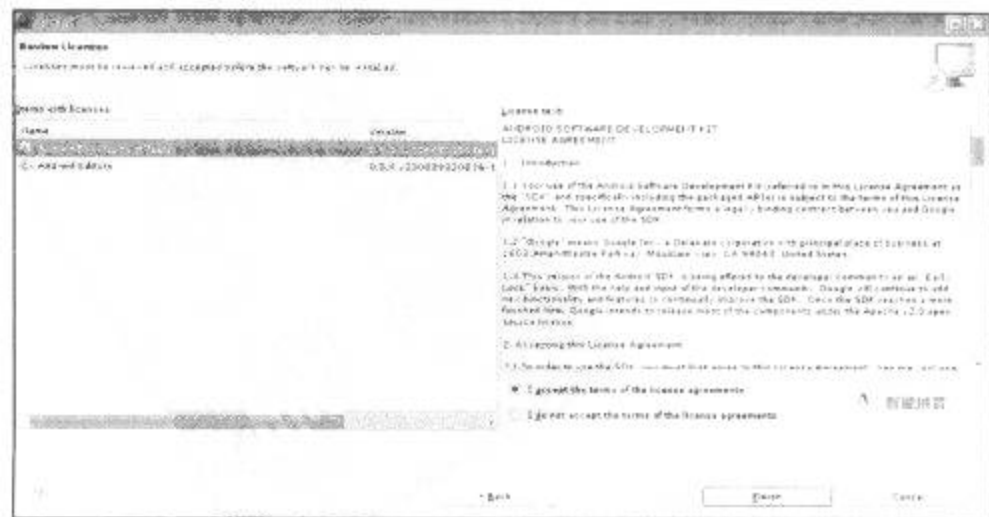
▲ 图 2-43 单击“install”操作界面

在安装的界面中，确认“Android Development Tools”和“Android Editors”已经检查，然后单击“Next”按钮，如图 2-44 所示。

单击“Finish”按钮，如图 2-45 所示。



▲ 图 2-44 选择需要安装的组件



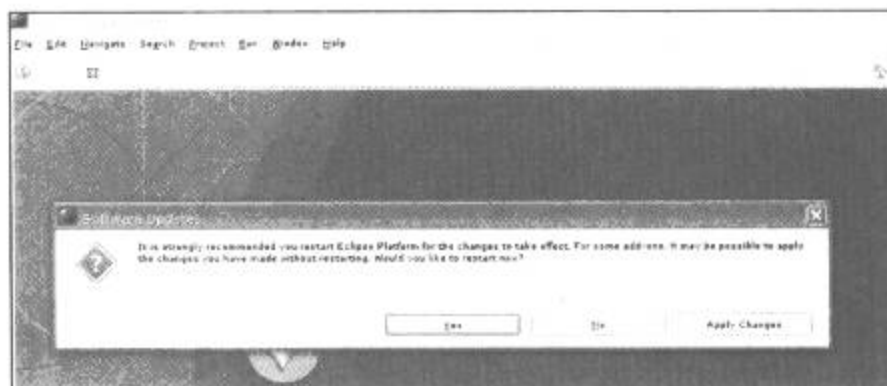
▲ 图 2-45 单击“Finish”操作界面

至此，这个插件现在已经安装完毕，如图 2-46 所示。

Eclipse 必须重新启动以便变更一些设定。安装完成后会提示用户“是否希望重新启动 Eclipse 后继续安装”时，应该选择“Yes”，如图 2-47 所示。



▲ 图 2-46 插件安装完毕

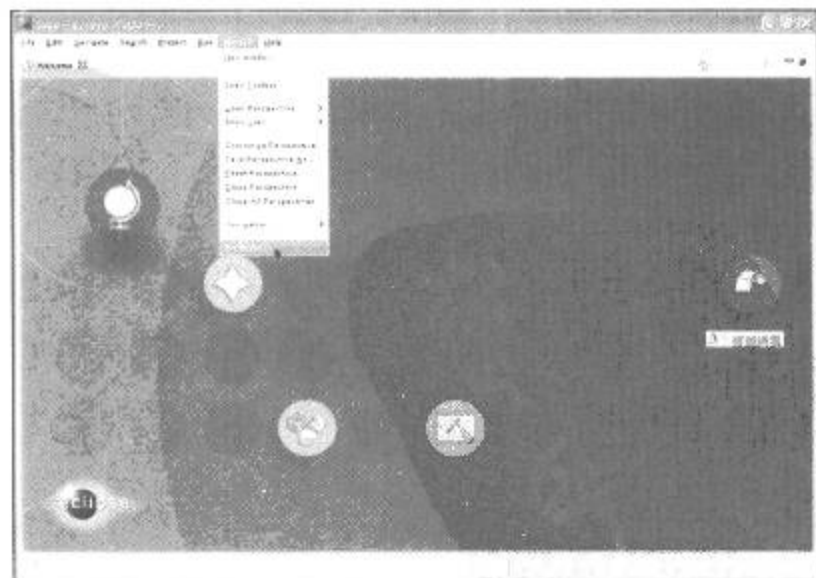


▲ 图 2-47 重启 Eclipse 界面

2.3.2 设定 Android SDK Home

Eclipse 已经再次启动后，我们必须更新 Eclipse preferences 对应到 Android SDK 目录，依次选择 Window → Preferences 项，如图 2-48 所示。

选择 Android 标签，单击“Browse...”项，选择 Android SDK 目录的路径，如图 2-49 所示。



▲ 图 2-48 设置 Android SDK 路径



▲ 图 2-49 单击 Browse 操作

选择 Android SDK 目录 (/home/iceskysl/Android-sdk-linux_x86-1.1_r1)，如图 2-50 所示。

返回到 Preferences 界面，单击“Apply”项，然后再单击“OK”按钮，如图 2-51 所示。



▲ 图 2-50 选择 Android SDK 路径



▲ 图 2-51 单击“Apply”操作界面

依照上述的操作步骤，Linux 的开发环境搭建完成，现在可以使用 Android SDK 去建立自己的 Android 应用了。

2.4 Mac OS 一族——苹果开发环境搭建

在 Mac OS 上搭建开发环境的方法和步骤与在 Ubuntu 上操作大同小异，这里就不再赘述了，参考上节的 Ubuntu 安装的步骤就可以了，只是在下载 Android SDK 的时候，注意选择 Mac OS 版本的。

2.5 本章小结

本章主要介绍了搭建 Android 开发环境需要的条件，诸如系统要求、SDK、IDE 等需求。然后分别介绍了在 Windows、Ubuntu 和 Mac OS 上搭建开发环境的过程和步骤。总体看来，本章简单且重要，好的开始是继续前进的动力。



第3章 清点可用资本—— Android SDK 介绍

学习目标

- 了解 Android 组成和用途; ☐
- 熟悉文档包含的内容; ☐
- 熟悉一些常用和重要的 Tools (工具); ☐
- 熟悉 SDK 附带的 Demo 及其实现技术; ☐
- 熟悉 Android 核心开发包和可选开发包。 ☐

3.1 Android SDK 基础

Android SDK (Software Development Kit) 提供了在 Windows/Linux/Mac 平台上开发 Android 应用的开发组件, 也就是说, 不论读者使用什么样的平台都可以开发出 Android 应用软件, 其包含了在 Android 平台上开发移动应用的各种工具集。它不仅包括了 Android 模拟器和用于 Eclipse 的 Android 开发工具插件 (ADT), 而且包括了各种用来调试、打包和在模拟器上安装应用的工具。

Android SDK 主要是以 Java 语言为基础, 用户可以使用 Java 语言来开发 Android 平台上的软件应用。通过 SDK 提供的一些工具将其打包成 Android 平台使用的 apk 文件, 然后再使用 SDK 中的模拟器 (Emulator) 来模拟和测试该软件在 Android 平台上运行情况 and 效果。

3.2 深入探寻 Android SDK 的密码

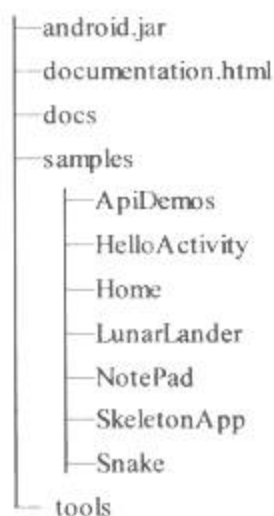
当读者按照本书第 2 章的操作搭建开发环境后, 肯定还记得曾下载过 Android SDK 的包, 当时是将其解压在 D:\unsetup\android-sdk-windows-1.5_r1 目录下了, 那么此文件包到底包含哪些文件, 以及这些文件的作用是什么, 这是本章所要讲述的。

3.2.1 Android SDK 目录结构

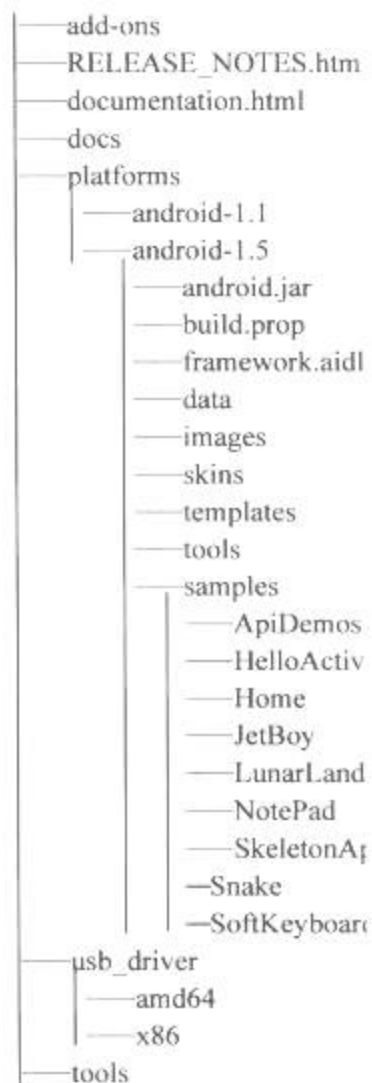
首先来看下 SDK 的目录结构, 这有助于从全局把握 SDK, 为后面的深入学习打下好的基础。

如果用户是从 1.0 或者 1.1 版学习 Android 的话，可能对下面展示的目录结构非常熟悉了，如图 3-1 所示。

新的 Android 1.5 版本 SDK 的目录结构和以前版本有着较大的区别，下面是 android-sdk-windows-1.5_r1 的目录结构，如图 3-2 所示。



▲ 图 3-1 Android SDK1.1 目录结构



▲ 图 3-2 Android SDK1.5 目录结构

首先需要说明的是，Android 1.5 版本的 SDK 增加了对多版本平台的支持，从上面目录的 platforms 下可以看到，其包含 android-1.1 和 android-1.5，也就是说 Android 1.5 版本的 SDK 可以兼容以前的 1.1 版本。

另外，add-ons 目录下的 google_apis-3 是 Google 提供的 API 包，其中主要是 Google Map 的 API，documentation.html 和 docs 目录下的是文档；RELEASE_NOTES.html 是 SDK 的发布说明；usb_driver 目录下包含了 amd64 和 x86 下的驱动文件；tools 目录下包含一些通用的工具文件。

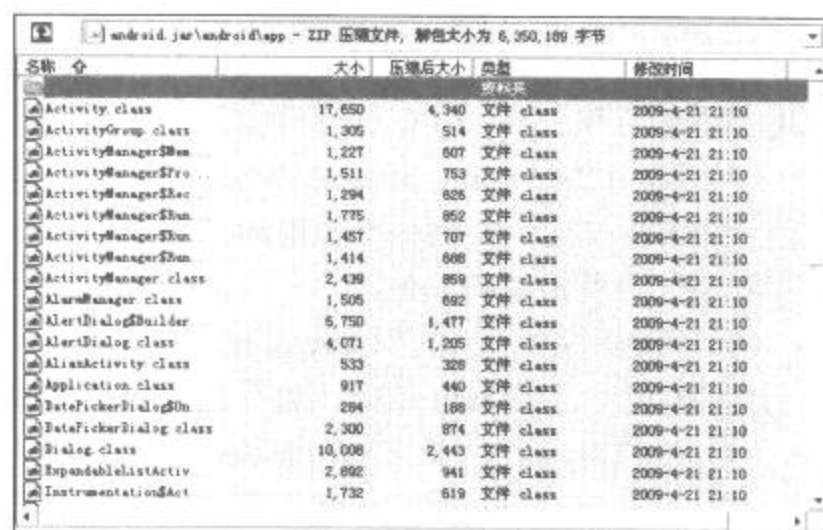
platforms 目录下针对每个版本的 SDK 版本提供了其相对应的 API 包以及一些示例文件，这里主要列举了 android-1.5 文件夹下的内容，其中 android.jar 是包含全部 API 的压缩包，samples 目录下是 SDK 附带的一些例子；skins 目录下则包含其支持的几种外观样式，如 HVGA(default)，HVGA-L，HVGA-P，QVGA-L，QVGA-P；templates 目录下是一些常用的文件模板；tools 目录下则是一些非常实用的工具和辅助工具，这些内容后面会详细介绍。

3.2.2 android.jar 及内部结构

android.jar 是一个标准的压缩包, 其内包含的是编译后的 class 文件, 包含了全部的 API, 我们使用 Windows 系统上的解压缩工具 winRAR 打开此压缩文件, 可以看到其内部结构如图 3-3 和图 3-4 所示。



▲ 图 3-3 android.jar 结构

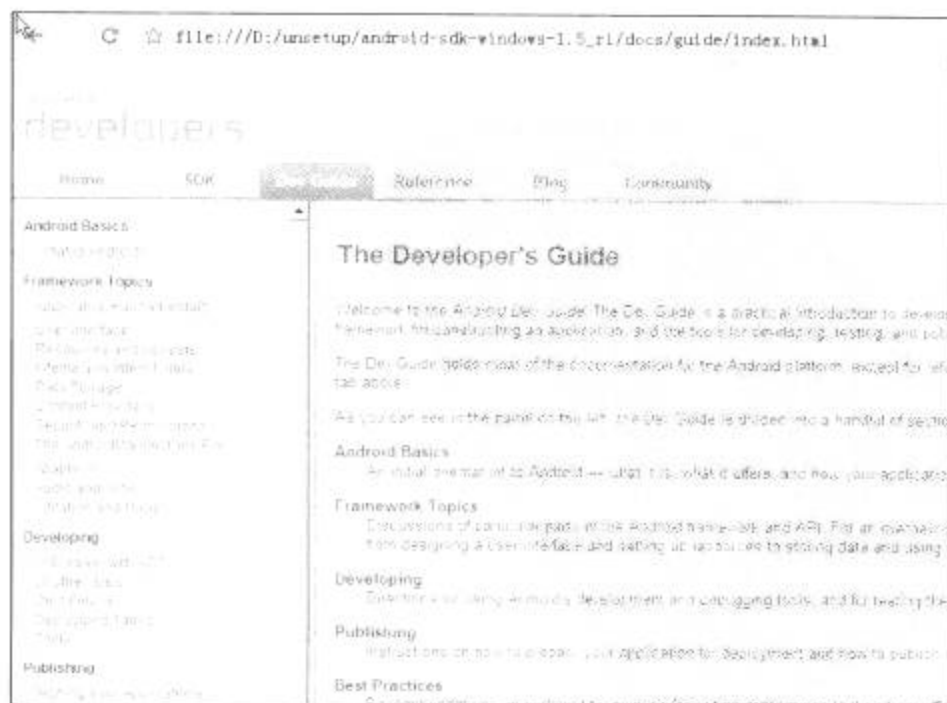


▲ 图 3-4 内部的 class 文件

从图 3-3 和图 3-4 可以了解其 API 的包结构的划分,如 app、content、database、graphics、hardware 等,也就可以大致了解了其模块的划分和结构,有了这种印象,有助于我们阅读和查找 SDK 的文档。

3.2.3 SDK 文档及阅读技巧

通过解压 android.jar 文件,大致了解了其内部 API 的包结构和组织方式,如果想深入了解各个文件包内包含的 API 及每个 API 的用法,那我们必须学会阅读和查找 SDK 的文档。使用 IE 浏览器打开 SDK 目录下的 documentation.html 文件,可以看到如图 3-5 所示的界面。



▲ 图 3-5 SDK 文档首页

在图 3-5 中可以看到，其首页上介绍了 Android 的概念，以及右侧一大堆的链接。读者可能要问了，这么多的文档，应该怎么看呢？哪些是必须先弄明白的？哪些应该先知道个大概，今后引用的时候再细细阅读的？如果没人指点的话，自己学习可能会陷入“文档风暴”中，需要花不少时间才能弄明白各自之间的依赖关系，所以，掌握一些文档的阅读技巧会给自己学习和研究省下不少时间，也能更清晰、有条理地掌握整个文档所要传递的知识和技术信息。现在简要地介绍一种阅读技巧，帮助读者少走弯路，找出学习的技巧。

(1) 首先要理解，什么是 Android？如果读者通过本书的第一章学习初步了解了 Android，若想进一步学习和了解，建议仔细阅读这个文档中的“**What is Android?**”。

(2) 阅读“**Anatomy of an Android Application**”能够知道一个 Android 应用中到底都有些什么东西，如果读完这个文档还不是很清楚的话也没有关系，本书第 6 章会详细介绍 Android 的组成部分和各个部分所扮演的角色。

(3) 接着可以读一下“**Development Tools**”一节的内容，其中会介绍 SDK 中包含的一些工具及工具的作用，但是其介绍的比较简单，我们后面会详细讲解各个工具的作用。

(4) 然后可以看一下“**Application Life Cycle**”，了解 Android 中生命周期的介绍，其中对前台进程、可见进程，后台进程等都做了比较详细的解释。

(5) “**Installing the SDK**”文档介绍了开发环境的搭建，读者通过本书第 2 章的学习，应该对这个文档中介绍的内容非常熟悉了。

(6) 了解了开发环境搭建以后，就动手实践了，先按照“传统习惯”实现一个“**Hello World**”程序，那么请参考“**Hello Android**”这节内容。

(7) “**Hello World**”程序比较简单，完成后就可以开始“**Notepad Application**”的制作和学习了，如果读者真正弄懂了关于这个应用程序的 4 个练习，就已经从初学者转向开发高手了。

(8) 在练习“**Notepad Application**”的时候，可以阅读一下“**Developing Applications**”这节的内容，其用了很多文字，也介绍了更多的细节。其中包括如何实现 UI，数据存储和读取，安全问题，资源引用和国际化等，其中每类下面都有更详细的介绍。

(9) 最后，还要熟悉一下“**Reference Information**”的内容，其中按照包、索引、继承关系介绍了 API 及其使用方法；同时列举了比较典型的视图组件和权限类型，做到心里有数，在以后遇到问题的时候知道在哪里可以查到就可以了。

3.2.4 先来热热身——Android SDK 例子解析

在前面介绍 Android SDK 目录结构的时候，我们看到一个名为“**samples**”的目录，其下存放的是 SDK 附带的几个示例演示应用程序，它们从不同方面展示了 SDK 的特性，我们下面逐一剖析这些 Demo。

1. Hello World 程序 (HelloActivity)

和我们以前学习别的语言的习惯一样，一开始实践开发一个“**HelloWorld**”简单程序，**HelloActivity** 这个例子就是如此，它非常简单，运行该程序时，将在手机（模拟器）上显示“**Hello, World!**”提示，如图 3-6 所示。

该例子虽然简单，其实是“麻雀虽小，五脏俱全”，它涉及 Java 程序，XML 模板文件，字符

定义及其引用等方面的技术，将在本书第5章中了解其详细的结构和各个结构所起的作用。

2. 视图组件应用 (SkeletonApp)

这个示例应用展示了，在 Android 中如何使用其提供的视图组件，例如 EditText、Button、ImageView 和菜单等，并演示了如何操作这些组件，应用运行界面如图 3-7 所示。



▲ 图 3-6 Hello, Activity! 界面



▲ 图 3-7 SkeletonApp 界面

学习 Android 已经提供的视图组件是必要的，本书第7章将介绍常用的视图组件及其使用方法。

3. API 应用实例 (API Demos)

这个例子演示了很多的 API 演示，包括 app、content、graphic、media 等，应用运行界面如图 3-8 所示。

在图 3-8 上可以选择其上的分类，进入查看更加详细的内容，通过这个例子，将了解到其 API 功能非常强大。浏览并熟悉其中包括的所有 API，等自己需要编写应用时，忘记某个 API 使用的时，可以翻阅这个例子的源代码参考。

4. 登月游戏实例 (LunarLander)

这个实例演示的类似于登录月球的小游戏，其通过方向键和点火时机控制画面上不断下坠的飞船，使它可以安全着陆，应用运行界面如图 3-9 所示。



▲ 图 3-8 API Demos 界面



▲ 图 3-9 LunarLander 界面

从技术实现上分析，其演示了如何使用键盘快捷键，如何实现菜单，如何定制自己的视图组件，如何实现线程等技术点，具体实现方式可以参考其源代码。

5. 记事本实例 (NotePad)

一个记事本应用程序，用此程序可以进行新建、编辑、删除等文档编辑操作，应用运行界面如图 3-10 所示。

本实例从技术实现角度分析，其中用到了 SQLite 的数据存储和编辑，ContentProvider 等方面的技术，具体的实现可以参阅其源代码。

6. 贪吃蛇游戏实例 (Snake)

此实例演示的是一个非常经典的贪吃蛇游戏，用方向键控制蛇的前进方向，吞并和其颜色一样的“食物”后可以成长，应用运行界面如图 3-11 所示。



▲ 图 3-10 Note Pad 主界面



▲ 图 3-11 贪吃蛇游戏界面

本实例从技术实现角度来分析，实现了自定义视图组件的实现和控制，实现了游戏类应用开发中相关技术，如果想开发游戏类应用，这无疑是最好的起点。

7. 主题类实例 (Home)

Home 是一款主题类软件实现的示例，它实现一套新的主题界面，并将其注册到系统主题中，使得用户按下“Home”键，可以显示主题并选择使用，应用运行界面如图 3-12 所示。

本实例从技术分析来看，很详细地演示了如何开发主题类应用，通过这个例子的学习，读者可以轻松掌握主题类应用开发步骤和一些注意事项。

8. 软键盘 (SoftKeyboard)

在 Android 1.5 版本的 SDK 中提供了使用软键盘的方法，这个例子就是介绍如何将软键盘绑定到输入框输入事件上，当聚焦到输入框时，将自动显示软键盘，如图 3-13 所示。

这个例子示范了如何在自己的程序中调用和整合软键盘，如果需要给自己的应用添加软键盘支持，可能很需要这个示例。



▲ 图 3-12 Home 选择界面



▲ 图 3-13 软键盘

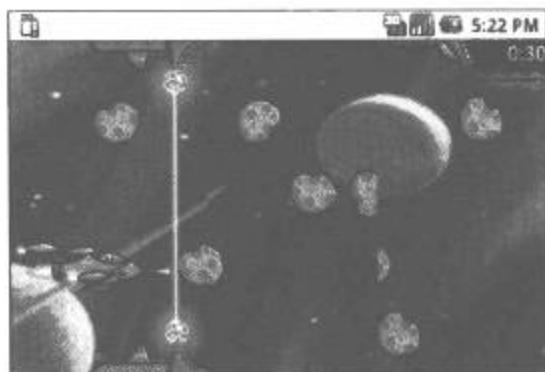
9. JETBOY

这是一款具备声音支持的游戏示例，其模拟演示了如何在游戏中集成 SONiVOX 的 audioINSIDE 技术，这项技术是 SONiVOX 捐赠给开放手机联盟的，以使其可以更加出色的播放背景音乐和情景音乐。

这个例子以一款发射子弹击碎飞来的障碍物，发射子弹、击碎障碍等都有其背景音乐，体验非常棒，图 3-14 是其启动界面，图 3-15 是其运行时的界面。



▲ 图 3-14 启动主界面



▲ 图 3-15 运行时界面

通过这个例子的学习和体验，如果需要介入多媒体游戏效果开发，这无疑是非常好的入门范例。

本节逐一介绍和演示了 Android-SDK 附带的几个例子，并从技术的角度分析其使用到的技术点，使得读者可以更有针对性的选择感兴趣的示例进行学习和研究，下面将继续介绍 Android SDK 中另外一个文件夹 tools 下包含的内容。

3.2.5 SDK 提供的工具介绍

Android SDK 包括各种各样的定制工具，可以帮助读者在 Android 平台上开发移动应用程序。其中最重要的工具是 Android 模拟器和 Eclipse 的 Android 开发工具插件，但 SDK 也包含了各种在模拟器上用于调试、打包和安装的工具，其针对不同的使用场景，能带来很多便利，如下将逐一介绍这些工具及其使用方法。

1. Android 模拟器 (Android Emulator (emulator.exe))

一个运行在计算机上的虚拟移动模拟器，可以使用模拟器在一个实际的 Android 运行环境下设计，调试和测试用户的应用程序。

2. 集成开发环境插件 (Android Development Tools Plugin for the Eclipse IDE , ADT)

用于 Eclipse 集成开发环境的 Android 应用开发工具插件，它为 Eclipse 集成开发环境增加了强大的功能，使得创建和调试 Android 应用程序更加简单和快速。如果使用 Eclipse 来开发 Android 应用，ADT 插件将带来极大的帮助，具体作用如下。

- 可以从 Eclipse 集成开发环境内部访问别的 Android 开发工具。通过它可以进行包括截屏、管理端口转发 (Port-Forwarding)、设置断点、查看线程和进程信息的一系列操作。
- 它提供一个新项目向导，用于快速创建一个新的 Android 应用需要的所有基本文件。
- 它使构建 Android 应用的过程自动化和简单化。
- 它提供一个 Android 代码编辑器，用于为 Android 的 manifest 和资源文件编写有效的 XML。

3. 调试监视服务 (Dalvik Debug Monitor Service (ddms.bat))

它集成在 Dalvik (Android 平台的虚拟机) 中，用于管理运行在模拟器或设备上的进程，并协助进行调试。可以用它来去除进程、选择一个特定程序来调试、生成跟踪数据、查看堆和线程数据、对模拟器或设备进行屏幕快照等。

4. Android 调试桥 (Android Debug Bridge (adb.exe))

它用于向模拟器或手机设备安装应用程序的 apk 文件和从命令行访问模拟器或手机设备。也可以用于将标准的调试器连接到运行在 Android 模拟器或手机设备上的应用代码。

5. Android 资源打包工具 (Android Asset Packaging Tool (aapt.exe))

可以通过 aapt 工具来创建 apk 文件，这些文件包含了 Android 应用程序的二进制文件和资源文件。

6. Android 接口描述语言 (Android Interface Description Language (aidl.exe))

它用来生成进程间接口代码，例如，在一个服务中可能就会用到。

7. SQLite3 数据库 (sqlite3 (sqlite3.exe))

Android 应用程序可以创建和使用 SQLite 数据文件，而开发者和使用者也可以方便的访问这些 SQLite 数据文件。

8. 跟踪显示工具 (Traceview (traceview.bat))

它可以生成跟踪日志数据的图形分析视图，这些跟踪日志数据由 Android 应用程序产生。

9. 创建 SD 卡工具 (mksdcard (mksdcard.exe))

它帮助创建磁盘镜像。这个磁盘镜像可以在模拟器上模拟外部存储卡（如 SD 卡）。

10. DX 工具 (dx (dx.bat))

它将 class 字节码重写为 Android 字节码（存储在 dex 文件中）。

11. 生成 Ant 构建文件 (activityCreator (activitycreator.bat))

它是一个脚本，用来生成 Ant 构建文件。Ant 构建文件用来编译 Android 应用程序。如果在安装了 ADT 插件的 Eclipse 环境下开发，就不需要这个脚本了。

12. Android 虚拟设备 (Android Virtual Devices AVD)

每个 Android 虚拟设备（AVD）模拟了单一的虚拟设备来运行 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。开发者可以创建并保存多种虚拟模拟器配置，每种配置环境有其自己的平台版本，硬件配置以及 SD 卡和用户数据，当然还可以有不同的显示外观等个性化设置，而你在运行的时候只需要指定需要使用哪个即可实现多平台下的模拟测试。

3.3 Android 典型包分析

参考 SDK 文档可以看到，其中有按照包结构查看文档，通过这种方式可以很清晰地了解其整个 API 结构，但是我们觉得还是有必要再对其核心包中的模块以及第三方包加以说明，可以更方便地找到自己需要的东西。

3.3.1 开发的基石——Android API 核心开发包介绍

这些是基本包，它们是通过 Android SDK 来编写应用程序的基石，这里是从最底层到最高层列出并加以简要说明。

- android.util 包含一些底层辅助类，例如，特定的容器类、XML 辅助工具类等。
- android.os 提供基本的操作服务，消息传递和进程间通信 IPC。
- android.graphics 作为核心渲染包，提供图形渲染功能。
- android.text android.text.method android.text.style android.text.util 提供一套丰富的文本处理工具，支持富文本、输入模式等。
- android.database 包含底层 API 处理数据库，方便操作数据库表和数据。
- android.content 提供各种服务访问数据在手机设备上，程序安装到手机设备和其他的相关资源，以及内容提供展示动态数据。
- android.view 核心用户界面框架。
- android.widget 提供标准用户界面元素，lists（列表），buttons（按钮），layout managers（布局管理器）等，是组成我们界面的基本元素。

- android.app 提供高层应用程序模型，实现使用 Activity。
- android.provider 提供方便调用系统提供的 content providers 的接口。
- android.telephony 提供 API 交互和手机设备的通话接口。
- android.webkit 包含一系列工作在基于 Web 内容的 API。

3.3.2 拓展开发外延——Android 可选 API 介绍

除了上面介绍的核心 API 外，Android 还有很多可选 API。Google 和 Sun 公司相同，把部分高端应用作为可选 API 供手机生产商定制不同的硬件支持模块。在 JME 中 Sun 公司是以 JSR 方式公布，谷歌公司（Google）采用了 optional API，包含但不限于如下这些模块。

1. Location-Based Services 定位服务

Android 操作系统支持 GPS API-LBS，可以通过集成 GPS 芯片来接收卫星信号，通过 GPS 全球定位系统中至少 3 颗卫星和原子钟来获取当前手机的坐标数据，通过转换就可以成为地图上的具体位置了，这一误差在手机上可以缩小到 10 米。

同时谷歌正在研制基于基站式的定位技术——MyLocation，可以更快速的定位，与前者 GPS 定位需要花费大约 1 分钟相比基站定位更快。

2. Media APIs 多媒体接口

Android 平台上集成了很多影音解码器以及相关多媒体 API，通过这些可选 API，厂商可以让手机支持 MP3、MP4、高清晰视频播放处理等支持。

3. 3D Graphics with OpenGL 3D 图形处理 OpenGL 可选 API

Android 平台上的游戏娱乐功能，如支持 3D 游戏，或应用场景就需要用到 3D 技术，手机生产厂商根据手机的屏幕，以及定位集成不同等级的 3D 加速图形芯片来加强基于 Android 平台手机的娱乐性。

4. Low-Level Hardware Access 低级硬件访问

这个功能主要用于控制手机的底层方面操作，由于设计底层硬件操作，将主要由各个手机硬件生产厂商来定制，支持不同设备的操作管理等，如蓝牙 Bluetooth、WiFi 无线网络支持等。

3.4 本章小结

本章主要介绍了 Android SDK 的相关内容，包括其文档解读，示例解读，以及相关 API 的介绍，通过本章学习，可以比较清晰地把握 Android SDK 的全貌，熟悉其提供的相关示例，以及附带的工具使用。另外，如果读者遵循本章推荐的文档阅读顺序，可以更加快速、全面的了解整个 Android SDK 的知识。下面介绍一个大家非常关心的内容，Android Market 的申请、使用，让我们一起去体验 Android Market 的应用。



第4章 赚钱的市场—— Android Market 及应用发布

学习目标

- Android Market 介绍; ☐
- Android 开发竞赛相关; ☐
- 解读 10 大获奖典型实例; ☐
- 申请 Market 流程。 ☐

4.1 Google Market 产生背景与目的

北京时间 2008 年 8 月 28 日, Google 公司透露将要推出新一代网上平台 Android Market, 并将其定位为开放的内容分享系统, 也就是应用程序超市。使用这个系统可以帮助使用 Android 操作系统的手机用户在该平台寻找、购买、下载及评级使用 Android 操作系统的手机应用程序和其他内容。这个平台类似于 Apple 给 iPhone 用户提供的 AppStore, 可以连接 Google 在线服务器, 获取最新的应用程序。

Google Android 平台负责人 Eric Chu 称, Android Market 对应用程序开发者最大的吸引力是, 他们可以很轻松地进入这个“市场”。他写道: “与 YouTube 相似, 开发者仅需 3 个简单的步骤即可上传内容, 注册为‘商人’、上传内容并加以描述、发布。我们选择‘Market’这个字眼而不是‘Store’的原因是, 我们觉得应当为开发者创造一个开放的、畅通无阻的环境来发布内容。”

Google 公司表示不会像苹果那样对 iTunes AppStore 内的软件应用进行严格的监管, 这也是 Android Market 称为“Market”的意图。软件开发商只需要注册并一次性支付 25 美元的申请费, 之后就可以在 Android Market 上销售他们的软件, 并从每次交易中获得 70% 的利润, 这类似于苹果的销售模式, 不同的是, 交易所得的其他利润将属于运营商而不是 Google 公司。

4.2 体验“选货”的乐趣——在 G1 上体验 Market 的使用

随着 T-Mobile 与 2008 年 10 月 22 日开始正式在美国销售首款 Android 手机 G1, 使得其提供 Android 应用程序下载服务的 Android Market 也正式揭开神秘的面纱, 如图 4-1 所示。

刚刚上线时，Android Market 提供了大概 50 多种应用程序，包括可从 Amazon MP3 商店下载音乐、比价根据（Compare Everywhere）、烹饪指南（Cooking Capsules Taster）、个人记事（e-ventr）、了解全球各地天气的 iMap Weather、可嵌入 Skype 服务的 iSkoot For Skype，以及多种手机游戏，这些应用都可以免费下载。Market 上线后，吸引了大批的开发者，截至到笔者写此书稿时，已经超过 3000 个应用程序，其主要分为应用程序和游戏两大类。其中应用程序又分成许多小类，每类又按照流程度和发布时间排序，如图 4-2 和图 4-3 所示。

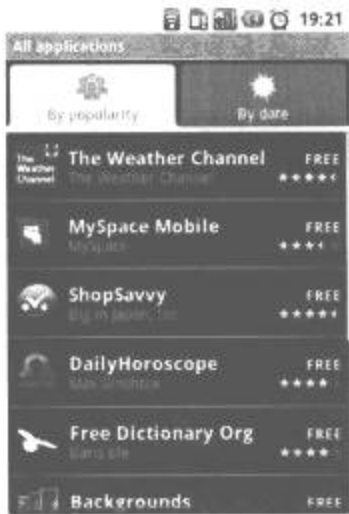


▲ 图 4-1 Android Market 界面



▲ 图 4-2 应用程序分类

游戏大类下同样也分成许多小类，例如动作类、智力、棋牌类等，选择每个应用，可以查看其详细信息，包括应用的简介、等级、下载次数、评分次数、安装者评价等，如图 4-4 和图 4-5 所示。

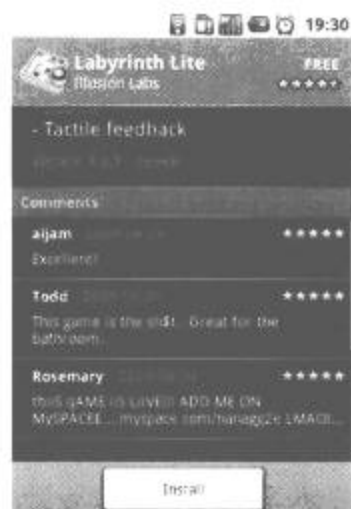


▲ 图 4-3 按照流行度分类

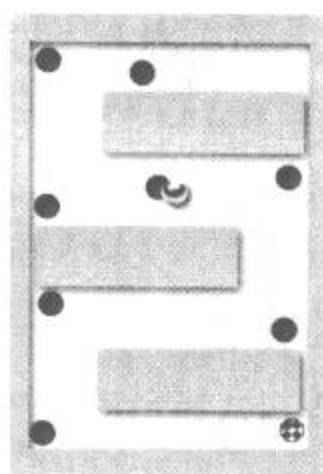


▲ 图 4-4 应用的详细信息

从图 4-4 上面这个应用的介绍文字可以知道，其是一款靠倾斜手机使得小球可以顺利通过陷阱的游戏，如果喜欢这个应用，那么可以单击“Install”按钮，在确认相关权限后，即可进行下载安装，安装成功后即可运行，如图 4-6 所示。



▲ 图 4-5 用户评分和评价



▲ 图 4-6 游戏界面

4.3 Android 开发活动及特色应用

Market 刚刚上线时，其应用程序十分缺乏，而 Android 的未来关键正取决于能让手机用户的生活多姿多彩的炫酷应用程序，而这些应用程序正是出自广大的开发者之手，为了推动 Market 和 Android 开发社区的发展，Google 已悬赏 1000 万美元的奖金举办“Android 开发者大赛”，只要您通过 Android 的平台开发出绝妙的手机应用程序，您就会有机会得到奖金，而且不需要任何附加条件。

4.3.1 开发应用的领域

Google 欢迎所有类型的参赛作品，并会奖励那些兼具创意与实用性，并能够借助 Android 的功能来提供更佳手机体验的应用程序，也可以从以下一些领域着手。

- 社交。
- 媒体消费、管理、修改或共享，如照片等。
- 效率与协作，如电子邮件、即时通信、日历等。
- 游戏。
- 资讯与信息。
- 突破传统用户界面的新构想。
- 混搭式（Mash-up）网络服务。
- 定位服务。
- 社会公益。
- 服务于全球经济发展的各类应用。
- 任何您所热衷的领域。

1. 应用程序的获奖标准不是惟一的，可以从如下方面考虑

- 构思新颖：应用程序是否展现了绝妙的创意，比如社交应用程序中的新视角？
- 有效利用 Android 平台：应用程序能否将 Android 独有的显著特点发挥到了极致，比如内置的定位服务、加速计以及时刻在线网络？

- 完善且具有吸引力：应用程序是否便于使用且具有美感？
- 不可或缺：应用程序能否令用户感到必不可少，比如让用户欲罢不能的游戏或功符号？

2. 哪些人有资格参赛

- Android 开发者大赛面向所有个人、个人开发团队以及商业机构。
- 您必须达到居住国家/地区所规定的法定参赛年龄。
- Google 公司工作人员、开放手机联盟成员或联属公司不能参赛。

4.3.2 Android Market 特色应用一览

1. 第一轮竞赛获奖实例

谷歌刚刚结束 Android 开发人员挑战赛。裁判向创建的 50 个最佳应用程序的每个人奖励 2.5 万美元。没有人能够保证在 Android 手机于晚些时候发布时，这些应用程序是否能火起来和吸引用户。但是，下面是人们可以期待的一些例子。

(1) AndroidScan。

AndroidScan 软件是 Jeffrey Sharkey 制作的。这个软件能够让用户发现和评估带条形码的任何东西的价格。这个软件能够显示 CD、DVD 和图书封面，以及亚马逊网站的详细评估。这个软件还能够搜索十几个商店，提供给商店打电话或者得到方向等选择。

(2) BioWallet。

BioWallet 是用于 Android 操作系统的一种生物身份识别系统。这个软件能够通过观察用户的虹膜进行身份识别，作为一种口令安全措施和提供其他 Android 应用程序的单一登录。

(3) Commandro。

类似于有用的网络应用程序 SNIFF，这个社交网络应用程序使用 GPS 数据显示有关用户和朋友的实时事件和位置信息。用户能够邀请朋友参加活动。当他们确认之后，相关的活动将自动创建并且增加到他们的活动计划中。这个应用程序是 Alex Pisarev 和 Andrey Tapekha 创建的。

(4) PedNav。

PedNav 软件能够帮助用户在城市活动中规划自己的活动。这个软件能够向手机用户提供帮助，让手机用户规划一天的活动，包括饭店、办公室、商店和旅游点等内容。利用当地公司和物流公司创建一个个性化的行程计划，指定什么时候去什么地方和如何去。PedNav 软件是 RouteMe2 技术公司制作的。

(5) Eco2goFor。

Eco2goFor 是一款环保软件，能够让用户跟踪和减少碳排放量并且与全球的其他人共享这个想法。Taneem Talukdar、Gary Pong、Jeff Kao 和 Robert Lam 共同创建的这个软件能够计算旅行的碳排放量并且找到和建议公共交通的替代方法。

(6) Teradesk。

Teradesk 是一个文件虚拟化平台，用于文件存储、远程接入和用户协作。José Augusto Athayde Ferrarini 编写的这个应用软件能够让用户从家里的计算机和手机访问文件，同时为任何类型和任何规格的文件提供多种加密的传输方式，让用户暂停和恢复中断的传输。

(7) SocialMonster.

SocialMonster 是一种特殊的社交规划工具, 让用户发送邀请、聊天和制定计划, 不用给具体的人打电话。这个软件是 Ben Siu-Lung Hui 和 Tommy Ng 制作的。

(8) Multiple Facets Instant Messenger.

Multiple Facets Instant Messenger（多面即时信使）是一种熟悉位置的 XMPP（可扩展消息处理现场协议）即时消息客户端软件。用户能够交换多媒体消息和接收通知和提醒消息。这个应用程序是 Virgil Dobjanschi 创建的。

(9) Mobeedo.

这是一个为这次 Android 竞赛开发的移动搜索应用程序。这个应用程序能够把多媒体信息带给手机用户。这个软件是 Sengaro 编写的。

(10) LReady Emergency Manager.

LReady Emergency Manager 应急管理器是你的家庭中的应急准备和通信工具。这个软件能够让用户发送和接收多频道信息，自动获得基于位置的灾难警报。这个软件是 Chris Hulls、Dilpreet Singh、Luis Carvalho 和 Phuong Nguyen 编写的。

2. 竞赛第二轮获奖实例

竞赛决赛名单正式出炉。有 10 项 Android 应用每项获得 27.5 万美元奖金，另外 10 项应用每项获得 10 万美元奖金，这 20 项应用合计获得 375 万美元奖励，我们选取几个介绍下，希望能给你带来更多的灵感。

(1) \$275,000 获得者作品。

- cab4me₀

作者: Konrad Huebner, Henning Boeger。

cab4me 是 Google Android 开发挑战赛第二轮获奖作品, 由德国的 Both Konrad 和 Henning Boeger 联合开发。**cab4me** 允许你在世界的任何地方轻松地叫车, 你不需要知道当地的出租车公司电话, 也不需要输入你在什么地方, 仅仅通过 Android 手机运行 **cab4me** 这个小软件单击下 **pickup**, **Cab4Me** 就会下载附近区域出租车公司的电话号码等信息, 然后就可以打电话叫出租车了。此时, 用户只需在原地等候即可。**Cab4Me** 还包括对出租车的服务进行评价的功能, 给用户提供参考, 如图 4-7 所示。



▲ 图 4-7 cab4me 界面

- CompareEverywhere.

作者: Jeffrey Sharkey。

CompareEverywhere 是一个条形码扫描软件,可以实时查询到商品 ▲图 4-7 ca
价格、库存等商品相关信息,为用户购物提供最好的便利条件,其主要包含如下功能:

扫描条形码获得商品信息，并且可以与附近的商店商品价格比较；选定商店之后，可以通过地图找到商店位置和行车路线；文本搜索；购买清单，一边买一边划掉清单项目。

其作者是在几年前受到一个文献室用条形码查询图书的存量和价格的启发触发了这个创意,并且参予了 Google 的多种开源项目。此应用仅用两个星期便完成,可见用 Android 开发是多么迅速,如图 4-8 所示。

- Ecorio。

作者: Jeff Kao, Gary Pong, Robert Lam, Taneem Talukdar。

Ecorio 可以通过调用 GPS 信息, 在后台运行, 并计算你每天的碳排放。在积累了一定的数据之后, Ecorio 便会根据你的习惯向你建议减少碳排放的小窍门, 如乘坐公交工具等, 如图 4-9 所示。



▲ 图 4-8 CompareEverywhere



▲ 图 4-9 Ecorio

- GoCart。

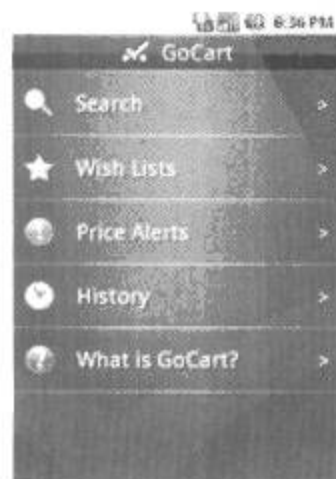
作者: Rylan Barnes。

用户可以通过手机的内置摄像头扫描带条形码的任何物品, 扫描之后可以使用 GoCart 这个软件在网上查询相关物品的信息, 同时还能够参考到这个物品在附近商店最便宜的价格, 而且这个软件还能提供相关类似商品的信息, 可以通过网络与朋友分享你的信息, 如图 4-10 所示。

- Life360。

作者: Chris Hulls, Dilpreet Singh, Luis Carvalho, Phuong Nguyen, Steve Potell。

这是一个以邻里关系为中心的社交网络应用程序。它可以使得用户与家人以及当地社区的联系更为紧密。Life360 的用户彼此之间可以收发紧急警报。无论你在后院烧烤还是希望找人帮忙寻找宠物, 该软件都可以迅速帮你联系到家人和邻居, 如图 4-11 所示。



▲ 图 4-10 GoCart



▲ 图 4-11 Life360

- Locale。

作者: Carter Jernigan, Clare Bayley, Jasper Lin, Christina Wright。

Locale 来自麻省理工学院,该软件允许用户自定义最经常使用的地点,当用户到达这些地点时,软件可以以多种方式对用户做出提示。这一软件目前仅具备框架,但是未来拥有极大潜力。例如,在家庭自动化应用中,如果软件感应到用户正在回家途中,那么将会自动把家中的灯打开,如图 4-12 所示。

- PicSay。

作者: Eric Wijngaard。

PicSay 是一个简单好用运行在 Android 手机上的图像编辑软件,通过 PicSay 可以方便的编辑修改图像、处理照片,同时可以分享到自己的 Blog、Facebook、Flickr、Photobucket 上面,通过 PicSay 还可以同步个人的图片集,如图 4-13 所示。



▲ 图 4-12 Locale



▲ 图 4-13 PicSay

- Softrace。

作者: Staffan Kjellberg, Thomas Kjellberg。

Softrace 使用 Google Android 平台 MyLocation API 和 Map API 开发的一款旅行必备软件,使用了 Android 手机存储中的 SQLite 数据库保存信息,比较简洁大方,如图 4-14 所示。

- TuneWiki。

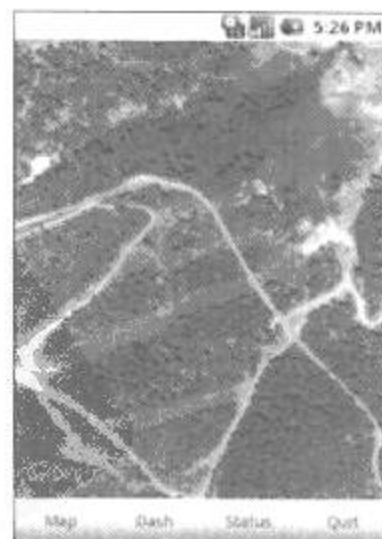
作者: TuneWiki Inc。

使用过 iPod Touch 或 iPhone 的网友一定知道自动匹配歌词等信息的 TuneWiki 软件,如今 Android TuneWiki 也出现了,可以匹配出 Android 手机上音乐的相关信息,同时加入了 GPS 定位相关功能,如图 4-15 所示。

- Wertago。

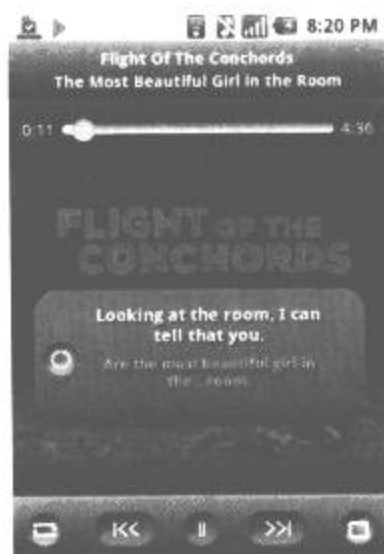
作者: Kelvin Cheung, Teresa Ko, Peter Ree, Robert Sarvis, Douglas Yeung。

Wertago 也算一个社交软件,使用者可以经由这款应用程序与好友分享社群活动,而且还可以向好友推荐他们最喜欢的地方。与许多其他的 Android 应用程序类似,Wertago 同样



▲ 图 4-14 Softrace

采用了 Google 地图的 API (應用程式介面)，使用者可以在地图上对 Club、饭店和剧场标记，并将其推荐给好友，如图 4-16 所示。



▲ 图 4-15 TuneWiki



▲ 图 4-16 Wertago

(2) \$100,000 获得者作品。

- BreadCrumbz。

作者：Amos Yoffe。

BreadCrumbz 是一款不错的 Android 导航软件，BreadCrumbz 导航功能可以通过真实的图片，工作方式和 Google Street View 街景图一样，通过 3G 高速网络比在地图上导航更过瘾，可以提前看到当地的景色，BreadCrumbz 支持创建自己的路线图，同时分享给朋友或者遍布全球的 Android 手机用户，如图 4-17 所示。

- Cooking Capsules。

作者：Mary Ann Cotter, Muthu Ramadoss。

很有趣的一个作品，简而言之就是把食谱存在手机里，它会告诉你你可以去附近哪个超市买哪些食材，买回家之后如何处理，让你可以看着手机做出一道又一道的佳肴，如图 4-18 所示。

除了上面列举的这些获奖应用外，还有很多，诸如 Maverick、PebbleBox、PhoneBook、Piggyback 等非常有特色的应用，这里就不赘述了。



▲ 图 4-17 BreadCrumbz

3. 收费应用列举

上面介绍的都是一些免费的应用，Market 推出收费功能后，很快又有一大批优秀的收费应用出现了，我们这个选取几个介绍一下。

- Bubble-Bash。

Gameloft 公司的 Android 游戏泡泡龙 Bubble-Bash，进入游戏之后，看到的是一片碧海蓝天的夏威夷风景，用 2D 技术画成的画面中我们看到的是非常精美的仿 3D 图像，尤其是对人物方面细节的设计，即使是人物在做动作的时候对各个方面也表现的非常细致，如图 4-19 所示。



▲ 图 4-18 Cooking Capsules



▲ 图 4-19 Bubble-Bash

- Guitar Hero® World Tour.

吉他英雄 (Guitar hero) 这款来自 World Tour Mobile 的游戏风靡了全球, 在 iPhone 上此前 Gameloft 推出的一款其他传说 (Guitar Legend) 相信很多人都玩过, 目前 Hands-On Mobile 在 Market 上对吉他英雄开出了 9.99 美元的价格, 非常火爆, 如图 4-20 所示。



▲ 图 4-20 Guitar hero

随着时间的推移, 还会有越来越多的应用出现, 也会让 Android 用户越来越有的玩。

4.4 你也可以做东家——申请 Market 账号

看了这么多各有特色的应用, 是否想赶快进入 Android 开发世界, 尽快发布自己的软件到 Market 上呢, 我们这节将详细讲解如何申请 Market 账号, 为后续发布软件做好前期准备。

4.4.1 卖东西要先入伙——准备工作

做申请 Android Market 账号之前, 需要有一个 Google 的账号才可以申请一个开发者账号, 并需要缴纳 25 美元的注册费, 所以需要有一张国际信用卡, 无论是 VISA 卡还是 Master 卡都可以。

如果已经做好这些准备工作了，下面进入账号申请的过程。

4.4.2 入伙过程——申请

打开 <http://market.android.com/publish/signup> 页面，输入基本信息，如图 4-21 所示。填写完后，单击“Continue”按钮进入支付引导页面，如图 4-22 所示。

▲ 图 4-21 输入基本资料

▲ 图 4-22 支付引导页面

单击“Continue”按钮进入支付界面，需要输入你的信用卡号和 CVC 码，需要注意其国家选项中尚无“China”，这个不要紧，选择“Hong Kong”就可以了，如图 4-23 所示。

资料填写完成后，转到其确认页面，如图 4-24 所示。

▲ 图 4-23 填写资料

▲ 图 4-24 确认页面

确认后开始支付，支付完成后显示成功消息，并可以看到你的控制台上可以上传软件了，如图 4-25 和图 4-26 所示。



▲ 图 4-25 申请成功



▲ 图 4-26 控制台

至此我们已经申请了自己的 Android Market 账号，Android Market 的账号不需要审核，支付完成后，就开通了，就可以发布自己的应用了。

4.5 开张了——在 Market 上发布应用

上面我们申请开通了自己的 Android Market 账号，已经有发布应用的权限了，但是在第一次使用时还是需要说明一下，发布应用的流程和注意事项。

需要提醒一下的是，我们将发布应用的步骤放在这里，开始的时候或许不是完全可以理解，不过不要紧，现在只需要记住大概的流程和步骤就可以了，等完成自己的第一个应用，需要发布的时候，记得翻回来查看就可以了。

4.5.1 发布时可能遇到的错误

在控制台上选择“Upload an Application”项，输入应用的名字，简介，选择相应的 APK 文件就可以了，如图 4-27 所示。

Android Market 要求发布的应用必须是经过签名的，且不能是 Debug 模式下签名，否则会出现如图 4-28 和图 4-29 所示的错误。

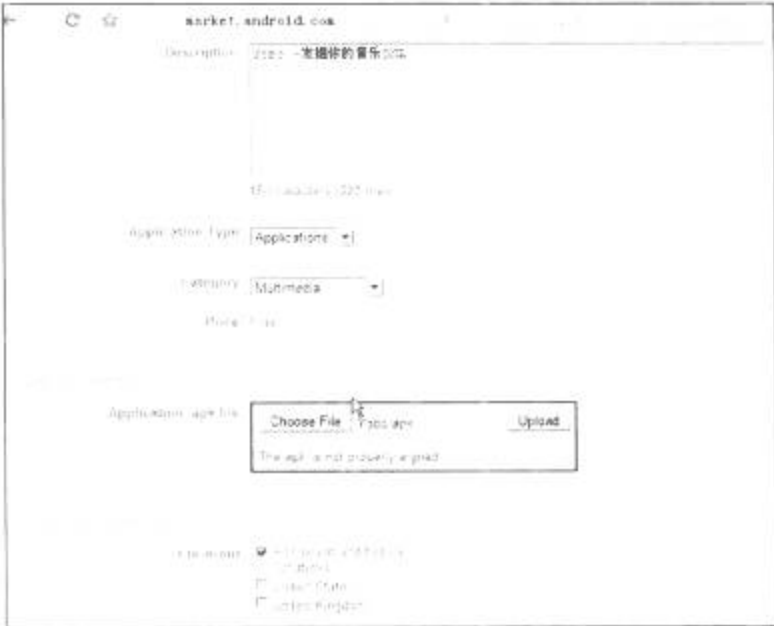


▲ 图 4-27 发布应用界面

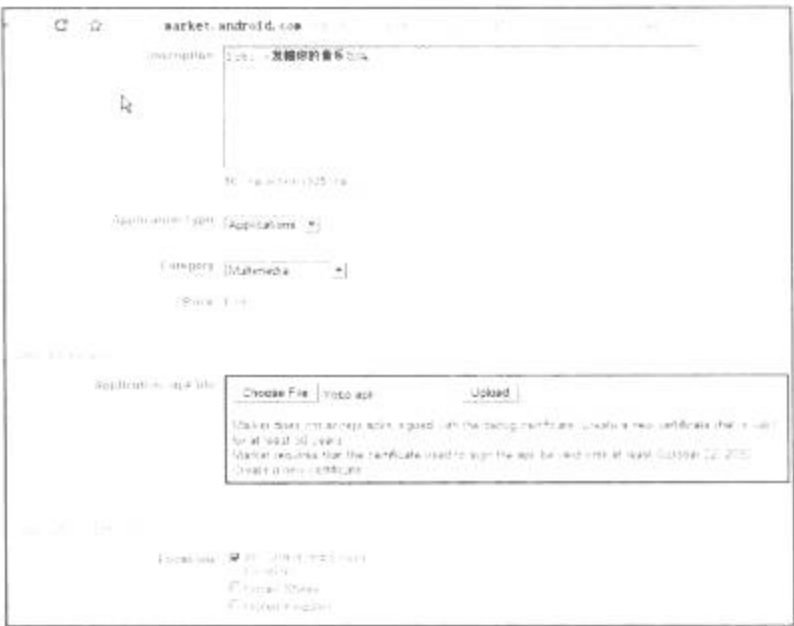
小知识

什么是 APK 文件？

APK 是 Android Package 的缩写，即 Android 安装包。APK 是类似于 Symbian Sis 或 Sisx 的文件格式。通过将 APK 文件直接传入 Android 模拟器或 Android 手机中即可安装。



▲ 图 4-28 未签名的发布



▲ 图 4-29 签名不正确的发布

4.5.2 卖东西也要签名——生成签名文件

从上面可以看到出现的错误都是跟签名相关的，那么，我们这里就来仔细讲解下 Android 需要的签名流程。首先需要确认是否已经安装了必须的两个工具 Keytool 和 Jarsigner，如果你已经按照第 2 章搭建开发环境的话，那么其已经包含在 JDK 中了（还记得在搭建环境时说最好是选择 JDK 而不是 JRE 吗？这就是一个原因）；如果你还没有搭建开发环境，那么请翻回到第 2 章按照说明搭建环境。

生成签名文件有两种方式，一种是使用命令行方式，另一种是借助 ADP 插件生成签名文件，下面就分别讲述。

1. 使用命令行生成签名文件

详细的步骤如下。

- (1) 创建一个专门用来发布应用的文件夹 publish 和存放签名文件的文件夹 publish。
- (2) 输入如下命令，按照其提示依次输入相关信息：

```
F:\publish>"C:\Program Files\Java\jdk1.6.0_10\bin\keytool.exe" -genkey -alias iceskysl_
android.keystore -keyalg RSA -validity 20000 -keystore keys/ic
eskysl_android.keystore
```

- 输入 keystore 密码：【未回显】。
- 再次输入新密码：【未回显】。
- 您的名字与姓氏是什么？
- [Unknown]: YaoShanglang。
- 您的组织单位名称是什么？
- [Unknown]: eoemobile.com。
- 您的组织名称是什么？
- [Unknown]: eoe。

- 您所在的城市或区域名称是什么？
- [Unknown]: beijing。
- 您所在的州或省份名称是什么？
- [Unknown]: beijing。
- 该单位的两字母国家代码是什么？
- [Unknown]: cn。
- CN=YaoShanglang, OU=eoemobile.com, O=coe, L=beijing, ST=beijing, C=cn 正确吗？
- [否]: Y
- 输入<iceskysl_android.keystore>的主密码。

(如果和 keystore 密码相同, 按回车):

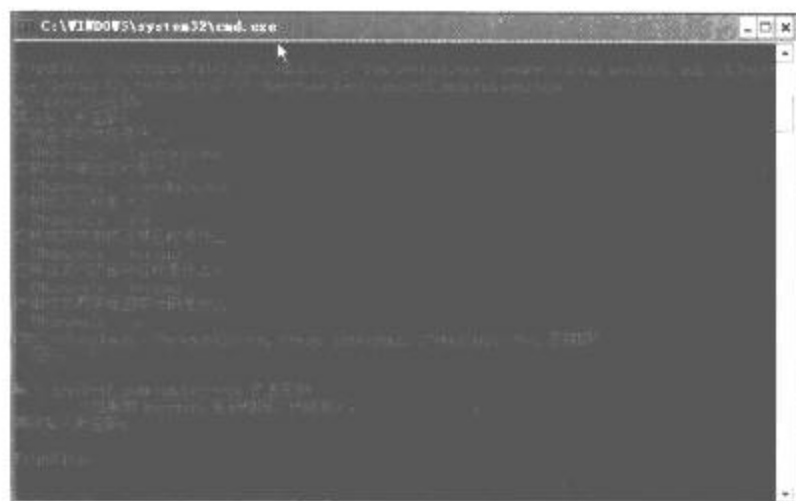
- 再次输入新密码:

其中参数-validity 为证书有效天数, 这里写的大些 20000 天。还有在输入密码时没有回显, 只管输入就可以了, 一般位数建议使用 10 位, 最后需要记下来后面还要用, 整个过程如图 4-30 所示。

如上步骤既可以在当前目录下的 keys 文件夹中生成一个名字为 iceskysl_android.keystore 的签名文件, 我们后面需要拿它来给我们的 APK 文件签名。

2. 使用 ADT 插件生成签名文件

除了上面介绍了如何在命令行下生成签名文件, 还可以在发布应用过程中, 在 ADT 插件中生成项目对应的签名文件, 其步骤为: 在需要发布的项目中上单击鼠标右键, 选择“Android Tools”→“Export Signed Application Package...”项, 如图 4-31 所示。



▲ 图 4-30 生成签名文件



▲ 图 4-31 选择导出功能

在接下来的对话框中选择需要导出的项目, 如图 4-32 所示。

选择需要导出的项目后, 单击“OK”按钮, 打开选择/创建签名文件的界面, 如图 4-33 所示。



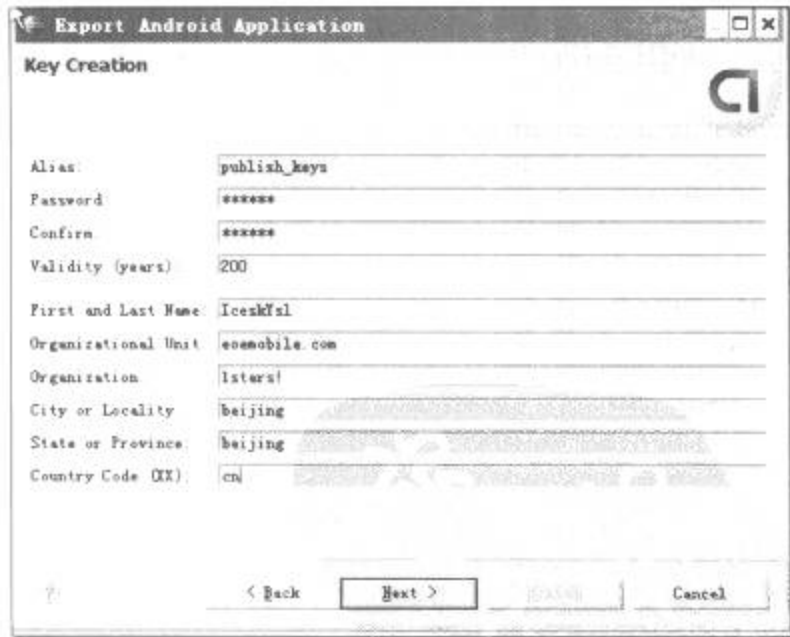
▲ 图 4-32 选项需要导出的项目



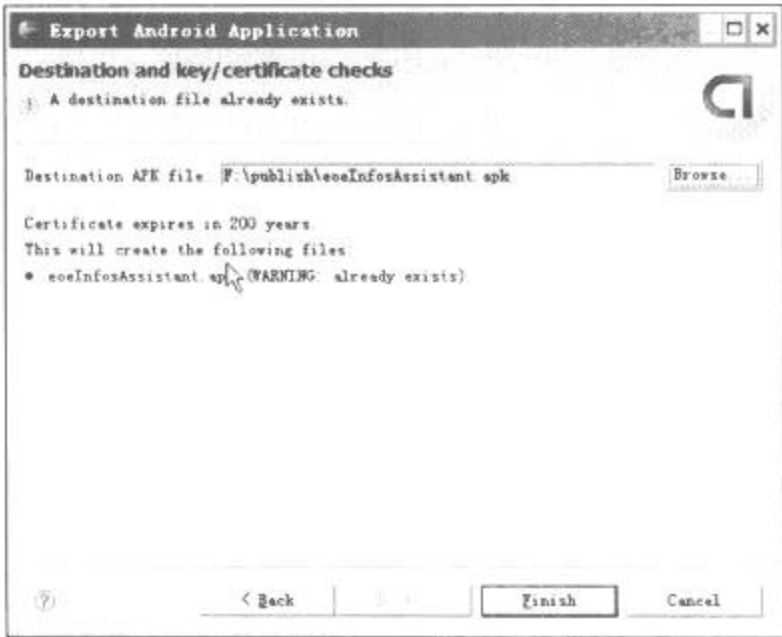
▲ 图 4-33 选择创建新的签名文件

这里选择“Create new keystore”，选择一个存储该签名文件的地址，然后输入密码，单击“Next”按钮，将打开签名文件信息录入界面，如图 4-34 所示。

按照前面使用命令行的方式录入的信息输入相关信息后，单击“Next”按钮，选择签名后的 apk 文件存储路径，如图 4-35 所示。



▲ 图 3-34 输入签名文件信息



▲ 图 4-35 导出签名文件

最后单击“Finish”按钮完成 apk 文件的签名，同时生成相应的签名文件。

4.5.3 打包、签名、发布应用

签名文件生成后，就可以用其对 apk 文件进行签名了，也有两种方式，第一种是使用命令行，第二种是使用 ADT，下面对此详细说明。

1. 使用命令行方式

步骤依次为。

(1) 在 Eclipse 中用“Export the unsigned apk”工具导出项目（后面会再次讲到），保存上面创建的发布目录中，假设为 eoe.apk。

(2) 使用 jarsigner 给 eoe.apk 文件签名，如下所示：

```
F:\publish>"C:\Program Files\Java\jdk1.6.0_12\bin\jarsigner.exe" -verbose -keystore
keys/iceskysl_android.keystore -signedjar eoe_signed.apk eoe.ap
k iceskysl_android.keystore
```

输入密钥库的口令短语：

正在添加： META-INF/MANIFEST.MF

正在添加： META-INF/ICESKYSL.SF

正在添加： META-INF/ICESKYSL.RSA

正在签名： res/drawable/app_launch.png

正在签名： res/drawable/app_remove.png

正在签名： res/drawable/back.png

正在签名： res/drawable/battery.png

...

如上命令即可将未签名 eoe.apk 文件签名为 eoe_signed.apk，如图 4-36 所示。

(3) 最后验证是否签名成功，如下所示：

```
F:\publish>"C:\Program Files\Java\jdk1.6.0_12\bin\jarsigner.exe" -verify eoe_signed.apk
jar 已验证。
```

如果成功，则如上显示：jar 已验证。

2. 使用 ADT 的步骤

在 Eclipse 中，在需要导出的项目上单击鼠标右键，选择“Android Tools”中的“Export the signed Application Package 项”，在选择需要导出的项目后，在其“Keystore selection”界面下选择“Use existing keystore”，选择签名生成的签名文件（这里是 public_keys），如图 4-37 所示。



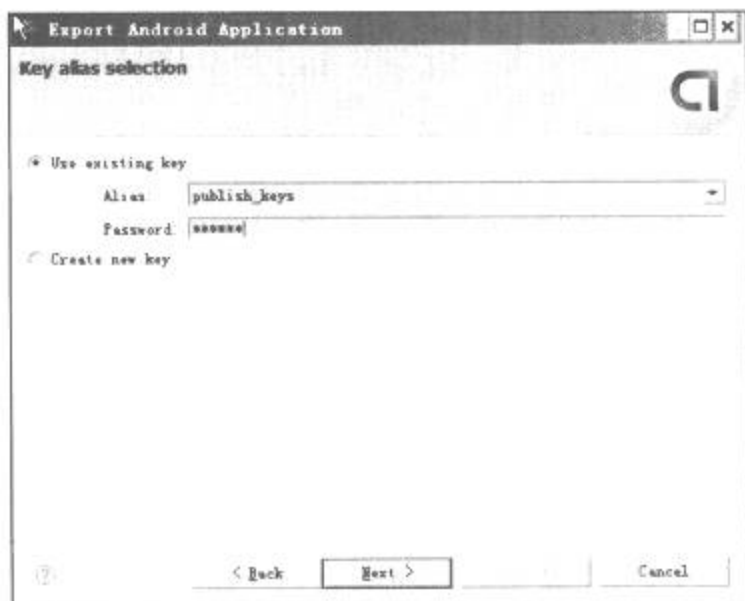
▲ 图 4-36 签名



▲ 图 4-37 选择签名文件

输入签名文件的密码后，单击“Next”按钮后，在其“Key alias selection”中选择 Alias 并输入密码，如图 4-38 所示。

选择导出签名后的文件存储路径，如图 4-39 所示。



▲ 图 4-38 选择 Alias



▲ 图 4-39 选择导出文件路径

最后单击“Finish”按钮完成导出。

至此，已经完成对 APK 文件的签名，然后就可以发布到 Android Market 平台上了，如图 4-40 所示。

发布成功后，即可在控制台上看到其被下载、安装的次数，已经被评的星级，如图 4-41 所示。



▲ 图 4-40 发布到 Android Market 平台



▲ 图 4-41 控制台显示的软件信息

图 4-41 显示每一行是一个应用，其中可以看到应用的名字、版本、分类以及下载次数，安装

次数，评星等，在这里可以大致了解用户开发的应用使用的状况。

4.6 本章小结

本章介绍了 Android Market 的历史背景和定位，接着介绍了 Android 程序开发竞赛的一些情况，并着重介绍了一批非常有特色的应用。后面部分介绍了如何申请 Market 账号以及生成需要的签名文件，如何打包、签名并在 Market 平台上发布应用。

本章内容有些超前，放在这里的目的是让读者在没开始前先了解些非常棒的应用，可以借鉴他们的一些思路，做出更好的应用。



第5章 千里之行 始于足下—— 第一个应用 HelloWorld

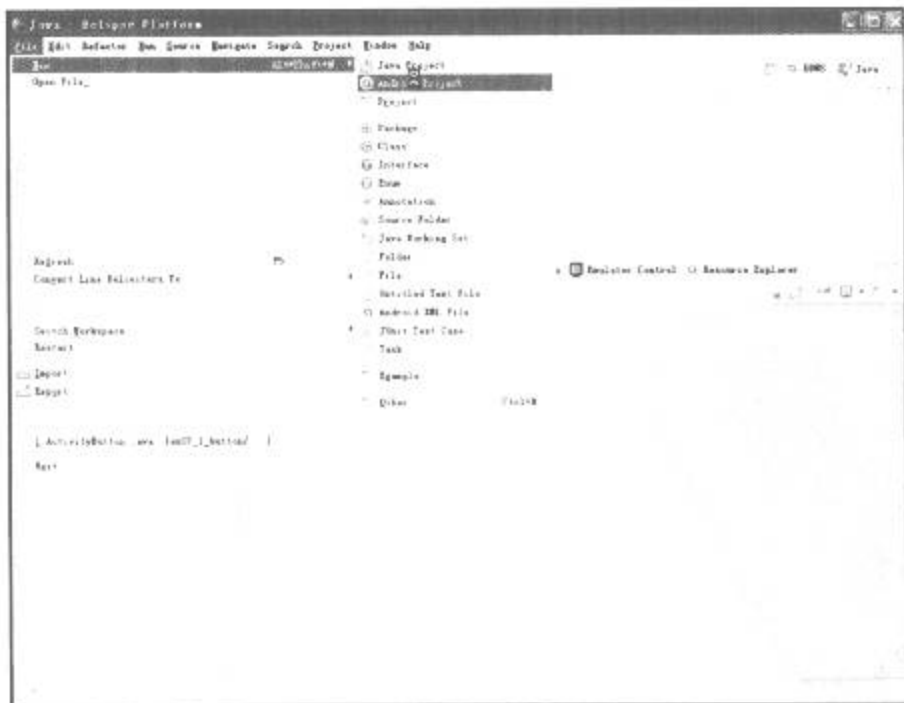
学习目标

- 了解如何快速构建 Android 项目； ☐
- 了解如何运行一个 Android 的项目； ☐
- 对一个 Android 的项目进行调试。 ☐

5.1 HelloWorld 应用分析

5.1.1 新建一个 Android 工程

打开 Eclipse，依次选择 File→New→others 项。出现如图 5-1 所示的界面，选择 Android project。



▲ 图 5-1 新建一个 Android 工程

5.1.2 填写工程的信息

在这一步中，我们需要填写这个工程的信息，这些信息包括：工程的名字、包的名字、Activity

的名字，还有应用的名字。工程的名字和包的名字不用多说，这个和普通的 Java 工程类似。Android 的应用可以说是以 Activity 为单位的，这里我们填写第一个 Activity 的名字。应用的名字，自己可以随便起一个，这个没有多少限制，但是最好让别人能比较容易读懂，而且确实也描述了你这个应用的功用。对于第一个 HelloWorld 应用，大家按照图 5-2 所示进行填写就可以了。

5.1.3 编程实现

我们打开 HelloWorld.java 这个文件，会看到下面的代码：

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

public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```



▲ 图 5-2 填写工程的信息

运行这个程序，但是程序不显示任何东西，我们还是稍微对程序做一些修改，让程序运行出“Hello World”的语句来，修改后的代码如下所示：

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

public class HelloWorld 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("Hello World");
        setContentView(tv);
    }
}
```

这里我们对代码进行简单的讲解，大家看不懂没关系，只要有一个大致的印象即可，在随后的章节当中，我们会对技术的细节进行详细的讲解。

在 Android 当中基本上所有的 UI 都是由 View 或者 View 的子类实现的。View 代表了一块区域，这块区域可以处理事件并且可以主动渲染这块区域。我们看到的第一个 UI 组件是 TextView，当然这个组件也是继承 View 来实现了。目前我们可以简单的认为 TextView 代表了一块区域，这片区域里边有一些文本信息，并且在这块区域里边我们可以处理这部分文本

信息。

5.1.4 运行项目

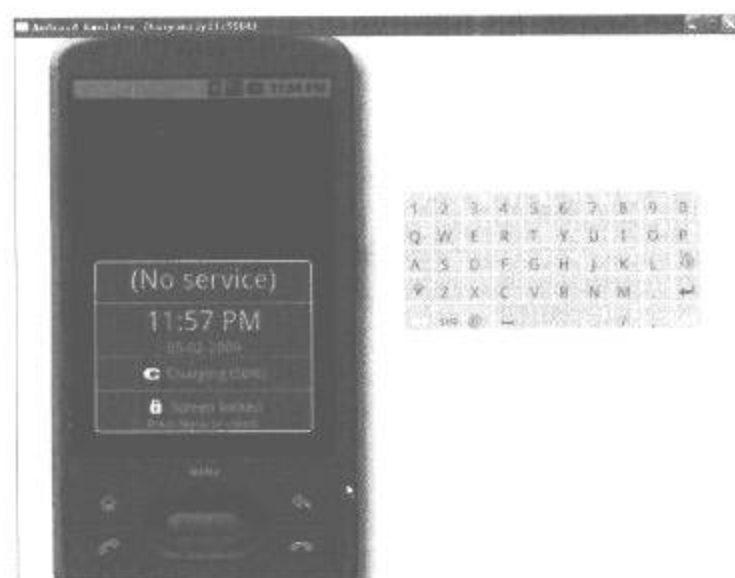
保存好上述代码后，如图 5-3 所示，我们可以运行 Android 程序，单击 HelloWorld 项目，单击 run as 项，然后选择 Android 项目。

当我们运行 Android 项目后，会跳出 Android 的模拟器，这个模拟器的第一次初始化会比较慢一些，稍等片刻会出现如图 5-4 所示的显示。

出现上边的状态是因为加载程序太长，Android 系统会自动会跳到这个状态，不过没关系，我们单击模拟器中的 menu 键，现在我们可以看到程序执行的结果了，如图 5-5 所示。



▲ 图 5-3 运行 HelloWorld



▲ 图 5-4 屏保状态



▲ 图 5-5 执行结果

5.2 调试项目

Android 提供的配套工具是强大的，利用 Eclipse 和 Android 基于 Eclipse 的插件，我们可以在 Eclipse 当中对 Android 的程序进行断点调试，下面就来具体演示一下。

5.2.1 设置断点

和对普通的 Java 应用设置断点一样，我们通过双击代码左边的区域进行断点设置，如图 5-6 所示。



▲ 图 5-6 设置断点

5.2.2 Debug 项目

Debug Android 项目的操作和 Debug 普通 Java 项目类似，只不过在选择调试项目的时候选择 Android Application 即可，如图 5-7 所示。



▲ 图 5-7 Debug 项目

5.2.3 断点调试

我们可以进行单步调试，具体调试和调试普通的 Java 程序类似，如图 5-8 所示。



▲ 图 5-8 断点调试

5.3 本章小结

至此，我们的第一个 HelloWorld 项目就算是完成了，通过这个项目我们了解了如何快速构建一个 Android 的项目工程，以及如何对 Android 的项目进行调试。在后续的章节当中，我们将构建更加复杂和有现实意义的例子，通过这些例子的学习，相信你对 Android 会有更加深刻的了解。



第6章 磨刀不误砍柴工——Android 应用程序结构介绍

学习目标

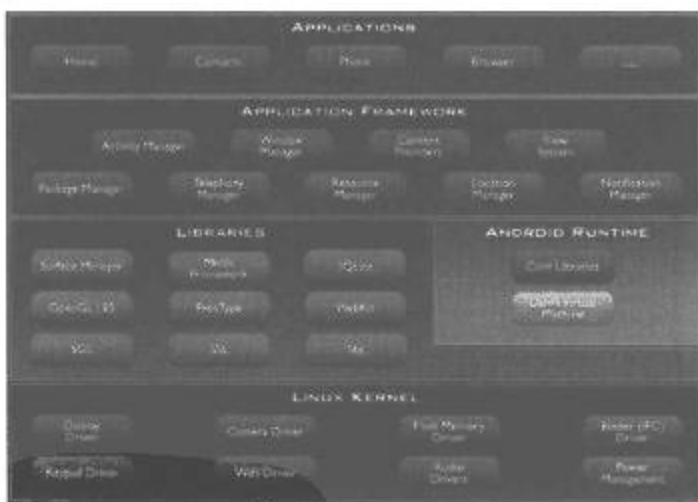
- 掌握 Android 的体系结构; ☐
- 掌握 Android 应用文件的组成结构; ☐
- 掌握 Android 应用的逻辑组成。 ☐

6.1 Android 体系结构介绍

Android 作为一个移动设备的开发平台,其软件层次结构包括了一个操作系统 (OS), 中间件 (MiddleWare) 和应用程序 (Application)。Android 的软件框如图 6-1 所示, 其软件层次结构自上而下可以分为以下几个层次。

- 应用程序 (Application);
- 应用程序框架 (Application Framework);
- 各种库 (Libraries) 和 Android 运行环境 (RunTime);
- 操作系统层 (OS)。

图 6-1 来自 Android 的官方网站, 主要描述了 Android 操作系统的整体架构, 下边将对每一部分进行简单的说明。



▲ 图 6-1 Android 系统架构

6.1.1 应用程序 (Application)

Android 的应用程序通常涉及用户界面和用户交互, 这类程序是用户实实在在能感觉到的。Android 的应用程序目前以 Java 语言进行编写。Android 本身提供了桌面 (Home)、联系人 (Contact)、电话 (Phone) 和浏览器 (Browsers) 等众多的核心应用。同时应用程序的开发者还可以使用应用程序框架层的 API 实现自己的程序, 这也是 Android 开源的巨大潜力的体现。Android 这种开放而强大的平台给每一个程序开发者提供了公平的机会, 每一个开发者都可以加入移动互联的世界, 共同推动移动互联的发展。

6.1.2 应用程序框架 (Application Framework)

普通的开发者可以使用 Android 核心应用程序使用的系统 API。Android 中的应用开发框架设计的非常巧妙，通过这套应用框架，各种组件可以被用户的应用重复利用，各种服务也可以被各种应用重复利用。开发人员应该很好地理解这套框架的工作机制，这样才能开发出更好的应用程序。

下边列举了每个应用程序有可能会使用到的应用框架的部分。

- 一组 Views (UI 组件)。这些 UI 组件包括列表 (List)、文本框 (textbox)、按钮 (button) 等。通过这些 UI 组件，可以构建应用程序的视图部分，这部分也就是用户能看到的部分。
- Content Providers。通俗地理解 Content Provider 就是提供了一种机制，通过这种机制，应用程序可以实现数据的互访和共享。
- Resource Manger。它负责管理非代码的访问。什么是非代码？如本地的图片资源、涉及布局的 XML 文件及国际化的字符串等。
- Notification Manager。它能让程序将自己的警示信息显示在状态栏上。例如，当有短信的时候，可以在状态栏上显示出新短信的信息。
- Activiy Manager。它管理着应用的生命周期，并且提供了应用页面退出的机制。在 Android 应用中，每一个应用一般是由多个页面组成的，而每个页面的单位就是 Activity，再通俗一点讲，可以认为 Android 的应用是由多个 Activity 的交互构成的（当然有些应用只有一个 Activity）。

在后边的章节中，会在实例中对上述的每一个部分进行讲解，现在读者只要有一个大概的印象就行了。

6.1.3 库 (Libraries) 和运行环境 (RunTime)

这部分知识已经开始涉及底层，和应用程序关系不是很密切，所以，我们会用比较少的篇幅进行介绍。大家只要大概了解一下即可。如果对这部分特别感兴趣，可以参考 Android 官方提供的 API 或下载源代码学习。

Android 包含一些 C/C++ 库，这些库能被 Android 系统中不同的组件使用。它们通过 Android 应用程序框架为开发者提供服务。以下是一些核心库。

- 系统 C 库。一个从 BSD 继承来的标准 C 系统函数库 (libc)，它是专门为基于嵌入式 Linux 的设备定制的。
- 媒体库。基于 PacketVideo OpenCORE，该库支持多种常用的音频、视频格式回放和录制，同时支持静态图像文件。编码格式包括 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG。
- Surface Manager。对显示子系统的管理，并且为多个应用程序提供了 2D 和 3D 图层的无缝融合。
- LibWebCore。一个最新的 Web 浏览器引擎，用来支持 Android 浏览器和一个可嵌入的 Web 视图。
- SGL。底层的 2D 图形引擎。
- 3D libraries。基于 OpenGL ES 1.0 APIs 实现，该库可以使用硬件 3D 加速（如果可用），或

者使用高度优化的 3D 软加速。

- FreeType。位图 (bitmap) 和矢量 (vector) 字体显示。
- SQLite。一个对于所有应用程序可用，功能强大的轻型关系型数据库引擎。

Android 包括了一个核心库，该核心库提供了 Java 编程语言核心库的大多数功能。

每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 被设计成一个设备，可以同时高效地运行多个虚拟系统。Dalvik 虚拟机执行 (.dex) 的 Dalvik 可执行文件，该格式文件针对小内存使用做了优化。同时虚拟机是基于寄存器的，所有的类都经由 Java 编译器编译，然后通过 SDK 中的 "dx" 工具转化成 dex 格式，由虚拟机执行。

Dalvik 虚拟机依赖于 Linux 内核的一些功能，例如，线程机制和底层内存管理机制。

Android 的核心系统服务依赖于 Linux 2.6 内核，如安全性、内存管理、进程管理、网络协议栈和驱动模型。Linux 内核也同时作为硬件和软件栈之间的抽象层。

6.2 Android 应用程序组成

一般来说 Android 程序主要由下列 4 部分组成，它们分别是。

- Activity。
- Broadcast Intent Receiver。
- Service。
- Content Provider。

并不是说所有的程序都需要这 4 部分，例如，刚刚完成的 HelloWorld 程序只涉及了 Activity，其他的 3 项并没有涉及。

在 Android 项目里边有一个 XML 的描述文件，叫做 AndroidManifest.xml。在这个文件里边，需要申明用到的上边 4 类组件名称，以及可能为每个组件的功能和需求进行必要的描述。打开第一个程序里边的 AndroidManifest.xml 文件，会看到一些系统自动生成的描述。现在只需要大概了解即可，不必深究，在后边会一点一点说明。接下来我们了解一下这 4 部分每一部分是做什么的，让读者有一个整体认识。

6.2.1 Activity 介绍

Activity 可以算是开发者遇到的最频繁，也是 Android 当中最基本的模块之一。在 Android 的程序当中，Activity 一般代表手机屏幕的一屏。如果把手机比作一个浏览器，那么 Activity 就相当于一个网页。在 Activity 当中，我们可以添加一些 View，并且可以对这些 View 做一些事件的处理。例如，在 Activity 当中可以添加一些 Button 元素，或者 Check box 等。可以看到 Activity 的概念和网页的概念相当类似。一般一个 Android 应用是由多个 Activity 组成的，这多个 Activity 之间可以进行相互跳转，例如，按下一个 Button 按钮后，可能会跳转到其他的 Activity。和网页跳转稍微有些不一样的是，Activity 之间的跳转有可能返回值，例如，从 Activity A 跳转到 Activity B，那么当 Activity B 运行结束的时候，有可能会给 Activity A 一个返回值。这样做在很多时候是相当方便了，

在后边将会介绍到这方面的知识。

当打开一个新的屏幕时，之前一个屏幕会被置为暂停状态，并且压入历史堆栈中。用户可以通过回退操作返回到以前打开过的屏幕。我们可以选择性的移除一些没有必要保留的屏幕，因为 Android 会把每个应用的开始到当前的每一个屏幕保存在堆栈中。

Activity 是由 Android 系统进行维护的，它也有自己的生命周期，即它的一个产生、运行、销毁的一个周期，在这里我们不打算展开讲，在后边有一章会专门介绍一下 Activity 的生命周期，了解 Activity 的生命周期对程序的设计编写是非常有益的。

Intent 和 Intent Filters 介绍。

在这里最好也了解一下 Intent 和 Intent Filters 的一些基本概念。Intent 在 Android 应用中是相当重要的。在 Android 的官方 API 文档里，对 Intent 是这样定义的，An Intent is an abstract description of an operation to be performed。一个 Intent 就是一次对将要执行的操作的抽象描述。Intent 表现的作用有 3 种，我们在后边会一点一点展开的讲，在这里我们先记住最基本，也是最常用的 Intent 的作用。通过 Intent 我们可以在多个 Activity 之间进行跳转。Intent 中两个最重要的部分是 Intent 的动作 (Action) 和动作对应的数据 (Data)。典型的动作类型有，MAIN (Activity 的门户)、VIEW、PICK、EDIT 等。而动作对应的数据则以 URI 的形式进行表示，例如，要查看某一个人的联系方式，需要创建一个动作类型为 VIEW 的 Intent，以及一个表示这个人的 URI。

通过 Intent 也可以进行 Activity 的跳转。下边是实现 Activity 跳转的代码，我们大概看一下，在这里假设有两个 Activity，一个是 Activity A，另一个是 Activity B，代码如下所示：

```
Intent in = new Intent(A.this, B.class);
startActivity(in);
```

通过执行这两句代码，就可以实现从 Activity A 跳转到了 Activity B。

与 Intent 有关系的一个类叫 Intent Filters。如果说 Intent 是一个有效请求，一个 Intent Filters 则用于描述一个 Activity (或者 Intent Receiver) 能够操作哪些 Intent。一个 Activity 如果要显示一个人的联系方式时，需要声明一个 Intent Filters，这个 Intent Filters 要知道怎么去处理 VIEW 动作和表示一个人的 URI。Intent Filter 一般在 AndroidManifest.xml 中定义。

6.2.2 Broadcast Intent Receiver 介绍

可以使用 BroadcastReceiver 来让应用对一个外部的事件做出响应。这是非常有意思的，例如，当电话呼入这个外部事件到来的时候，可以利用 BroadcastReceiver 进行处理。例如，当下载一个程序成功完成的时候，仍然可以利用 BroadcastReceiver 进行处理。BroadcastReceiver 不能生成 UI，也就是说对于用户来说不是透明的，用户是看不到的。BroadcastReceiver 通过 NotificationManager 来通知用户这些事情发生了。BroadcastReceivers 既可以在 AndroidManifest.xml 中注册，也可以在运行时的代码中使用 Context.registerReceiver() 进行注册。只要是注册了，当事件来临的时候，即使程序没有启动，系统也在需要的时候启动程序。各种应用还可以通过使用 Context.sendBroadcast() 将它们自己的 intent broadcasts 广播给其他应用程序。

6.2.3 Service 介绍

Service 是一种程序，它可以运行很长的时间，但是它却没有用户界面。这么说有点枯燥，我们来看个例子。打开一个音乐播放器的程序，这个时候若想上网了，那么，我们打开 Android 的浏览器，这个时候虽然我们已经进入了浏览器这个程序，但是，歌曲的播放并没有停止，而是在后台继续一首接着一首的播放。其实这个播放就是由播放音乐的 Service 进行控制。当然这个播放音乐的 Service 也可以停止，例如，当播放列表里边的歌曲都结束，或者用户按下了停止音乐播放的快捷键等。关于 Service 的生命周期，我们会在后边章节介绍。通过 `startService(Intent service)` 可以来启动一个 Service，通过 `Context.bindService()` 可以绑定一个 Service。

6.2.4 Content Provider 介绍

Android 这个系统和其他的操作系统还不太一样，我们需要记住的是，数据在 Android 当中是私有的，当然这些数据包括文件数据和数据库数据，以及一些其他类型的数据。看了这样的介绍读者可能就会提出问题，难道两个程序之间就没有办法对于数据进行交换？Android 这么优秀的系统不会让这种情况发生的，现在该 ContentProvider 登场了。一个 Content Provider 类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。也就是说，一个程序可以通过实现一个 ContentProvider 的抽象接口将自己的数据暴露出去。外界根本看不到，也不用看到这个应用暴露的数据在应用当中是如何存储的，即用数据库存储还是用文件存储，还是通过网上获得，这些一切都不重要，重要的是外界可以通过这一套标准而统一的接口和程序里的数据打交道。可以读取程序的数据，也可以删除程序的数据，当然中间也会涉及一些权限的问题。下边列举一些较常见的接口，具体的使用方法参见后续内容，这里只是先稍微有个了解就可以了。

- `query(Uri, String[], String, String[], String)`：通过关键字查询数据。
- `insert(Uri, ContentValues)`：将一组数据插入到指定的地方。
- `update(Uri, ContentValues, String, String[])`：更新数据。
- `delete(Uri, String, String[])`：删除数据。

6.3 Android 应用工程文件组成

首先我们来看一个典型的工程文件是什么样子的，如图 6-2 所示。

现在我们来大概了解一下项目里边的每一部分构成。

- 源文件(包含 Activity)。源文件都在上边的 `src` 目录中，在上边源文件只有一个，就是 `HelloWorld.java`。
- `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。里边内容的修改也是由 Eclipse 自动来处理的。R 文件对于应用开发者来说基本上没有什么用，但是对 Android 系统非常有用。在这个文件当中，Android 对下边即将介绍的资源进行了全局索引。在下边介绍的 `res` 文件夹中，内容发生任何变化，`R.java` 都会重新编译，同步更新。
- Android Library。这个是应用运行的 Android 库。图 6-2 显示项目用的是 Android1.1 的库，

而图 6-3 显示项目用的是 Android1.5 的库。



▲ 图 6-2 HelloWorld 工程文件结构



▲ 图 6-3 Android 1.5 下 HelloWorld 工程文件结构

- assets 目录。在这个目录里边主要放置多媒体等一些文件。
- res 目录。这个目录主要放置应用用到的资源文件。下边分 3 个目录进行存放。当这个目录当中的资源文件发生变化的时候，上边我们介绍的 R 文件的内容就会自动发生变化。
 - drawable 目录。这个目录里边主要放置应用到的图片资源。
 - layout 目录。这个目录里边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。这个目录里边主要放置字符串 (strings.xml)、颜色 (colors.xml)、数组 (arrays.xml)。
- AndroidManifest.xml。这个文件相当重要，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。

6.4 本章小结

本章从全局的角度对 Android 程序进行了介绍，首先介绍了 Android 的体系结构，然后在应用的代码级别介绍了 Android 的代码模块组成，最后在应用的工程级别，介绍了一个应用工程中文件和文件夹的意义及一个工程的组成。看完这一章，读者对 Android 有了一个大概的认识，下一章开始将以代码和实例为主讲解 Android 的各个部分。当然，建议在编写代码一段时间后，最好能返回看看这一章，这样才能对 Android 理解越来越深刻。



第7章 良好的学习开端——Android 基本组件介绍

学习目标

- 掌握 Android 中的基本组件; ☐
- 掌握 Activity 和 Intent 的内容; ☐
- 掌握 Android 中的布局; ☐
- 掌握 Android 对话框; ☐
- 掌握 Android 中的 Toast 和 Notification。 ☐

7.1 第一印象很重要——界面 UI 元素介绍

本章是基于 Android 应用开发最为基础和重要的一章，在本章中我们就 Android 当中大量的组件进行了详细地介绍。另外，在这章中也会学习关于 Activity 和 Intent 的内容。

一个 Android 的应用是由一个或者多个 Activity 组成的。Activity 是一个 UI 的容器，其本身并不在用户界面上显示出来。我们在具体编程之前，先简单地了解一下，具体编程当中用到的 UI 基本元素以及它们之间是一种什么样的继承结构。

7.1.1 视图组件 (View)

在 Android 当中 View 类是最基本的一个 UI 类，基本上所有的高级 UI 组件都是继承 View 类而实现的，例如，我们在前边章节中实现的 HelloWorld 例子，其中用到的 TextView (文本框)，以及我们将在后边看到的其他 UI 组件，如 Button (按钮)，List (列表)，EditText (编辑框)，RadioButton (多选按钮)，Checkbox (选择框) 等都是 View 类。

一个视图 (View) 在屏幕上占据了一块矩形区域，它负责渲染这块矩形区域 (如将这块区域变成蓝色或其他颜色)，也可以处理这块矩形区域发生的事件 (如用户单击了这块区域)，并且可以设置这块区域是否可见，是否可以获取焦点等。

7.1.2 视图容器组件 (Viewgroup)

一个 Viewgroup 对象是一个 Android.view.Viewgroup 的实例。就像它的名字显示的意义一样，Viewgroup 的作用就是 View 的容器，它负责对添加进 Viewgroup 的这些 View 进行布局。

当然一个 Viewgroup 也可以加入到另外一个 Viewgroup 里边，因为 Viewgroup 也是继承于 View.Viewgroup 类，它是一个抽象类，也是其他容器类的基类。在下面我们会介绍它的一些实现类。

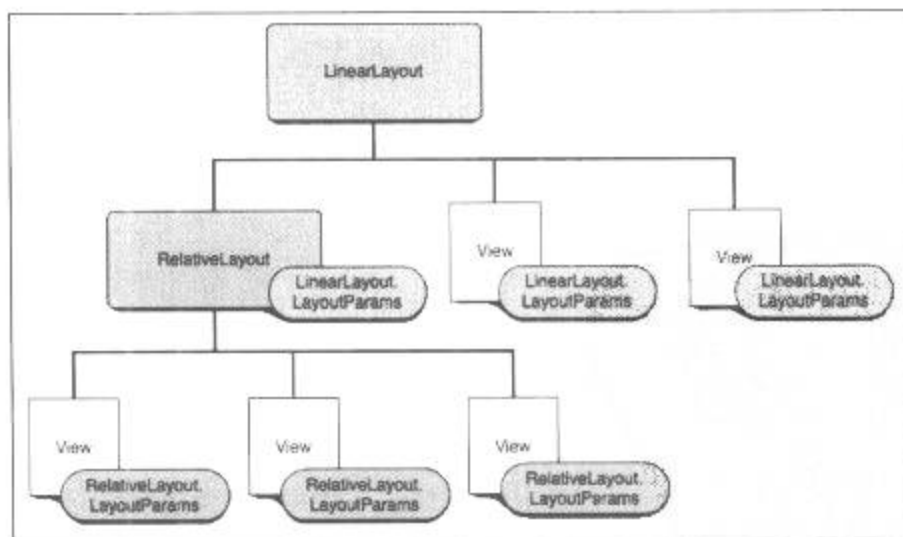
7.1.3 布局组件 (Layout)

Viewgroups 的实现类比较多，在这里我们先只介绍最常用的两个实现类。一个是 LinearLayout，它可以进行水平布局或者竖直布局。如果将 LinearLayout 的布局方向设置为“vertical”，那么表明是竖直布局。也就是说先加入进去的 View 会显示在顶端，后续加入的 View 会显示在底端。如果设置为“horizontal”，那么表明是水平布局，也就是从左到右依次进行排列布局。

RelativeLayout 负责相对布局，用过 CSS 编程的读者应该知道，在 CSS 中经常会用到相对布局，例如，设置 A 显示在 B 的左侧，那么 B 的显示坐标不是固定的，而是相对于 A 的位置。

7.1.4 布局参数 (LayoutParams)

一个 View 加入到一个 Viewgroup 里边，例如，加入到一个 RelativeLayout 里边，那么它在这个 RelativeLayout 里边是怎么显示的呢，如果这个 RelativeLayout 是 50×50 大小，这个 View 到底显示在这个 RelativeLayout 里边的上边？下边？左边？还是右边的？在将每一个 View 加入到这个 RelativeLayout 里边的时候，我们会传递一组值（如果没有传递值，那么系统采用的是默认值，反正肯定会有一组这样的值），这组值封装在 LayoutParams 这个类当中。在显示这个 View 的时候，它的容器会根据传进来的 LayoutParams 进行计算，来确认这个 View 显示的大小和位置，如图 7-1 所示。



▲ 图 7-1 Layout 结构图

7.2 我的美丽我做主——Android 中应用界面布局

在第 7.1 节大概介绍了一下布局 (Layout) 这个概念，也许还不是很明白。没有关系，在本节中将重点研究一些布局方面的知识，并通过一个例子来学习它。通过这个例子可以学到以下

内容:

- 如何使用 FrameLayout 进行布局;
- 如何单独使用 LinearLayout 实现布局;
- 如何单独使用 RelativeLayout 实现布局;
- 如何结合 LinearLayout 和 RelativeLayout 进行布局;
- 如何使用 TableLayout 进行布局。

7.2.1 实例操作演示

(1) 此布局程序运行后的主界面如图 7-2 所示。

(2) 单击第一个按钮后出现一个地图的界面, 如图 7-3 所示。



▲ 图 7-2 主界面



▲ 图 7-3 地图界面

(3) 单击第二个按钮后的界面, 如图 7-4 所示。

(4) 单击“LinearLayout 和 RelativeLayout 互助使用”按钮后界面如图 7-5 所示。

(5) 单击第四个按钮出现如图 7-6 所示的界面。



▲ 图 7-4 单击“演示 RelativeLayout”按钮后界面



▲ 图 7-5 单击“LinearLayout 和 RelativeLayout 互助使用”按钮后界面



▲ 图 7-6 整齐菜单

7.2.2 实例编程实现

1. 第一步

在 Eclipse 中打开 ex07_layout 项目，具体步骤如下所示。

- 新建一个项目。依次单击 File → New → Android Project 项。
- 在新建项目的对话框中，选择 Create project from existing source 项。
- 单击浏览，找到 ex07_layout 项目，然后单击“确定”按钮。

导入项目后，可以看到项目的结构如图 7-7 所示。

关于 Android 应用的程序结构我们在第 6 章当中已经详细介绍了，如果有疑问可以参照第 6 章的内容，这里不再介绍。

2. 第二步

我们首先来看一下 ActivityMain 里边的代码，里边可能有些代码暂时不是很明白，但没关系，随着后面的学习，这些读者都可以逐步掌握。ActivityMain 实现代码如下所示：

```
package com.eoeAndroid.layout;

import Android.app.Activity;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;

public class ActivityMain extends Activity {
    OnClickListener listener0 = null;
    OnClickListener listener1 = null;
    OnClickListener listener2 = null;
    OnClickListener listener3 = null;
    Button button0;
```



▲ 图 7-7 项目结构图

```

Button button1;
Button button2;
Button button3;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    listener0 = new OnClickListener() {
        public void onClick(View v) {
            Intent intent0 = new Intent(ActivityMain.this, ActivityFrameLayout.class);
            setTitle("FrameLayout");
            startActivity(intent0);
        }
    };
    listener1 = new OnClickListener() {
        public void onClick(View v) {
            Intent intent1 = new Intent(ActivityMain.this, ActivityRelativeLayout.class);
            startActivity(intent1);
        }
    };
    listener2 = new OnClickListener() {
        public void onClick(View v) {
            setTitle("这是在 ActivityLayout");
            Intent intent2 = new Intent(ActivityMain.this, ActivityLayout.class);
            startActivity(intent2);
        }
    };
    listener3 = new OnClickListener() {
        public void onClick(View v) {
            setTitle("TableLayout");
            Intent intent3 = new Intent(ActivityMain.this, ActivityTableLayout.class);
            startActivity(intent3);
        }
    };
    setContentView(R.layout.main);
    button0 = (Button) findViewById(R.id.button0);
    button0.setOnClickListener(listener0);
    button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(listener1);
    button2 = (Button) findViewById(R.id.button2);
    button2.setOnClickListener(listener2);
    button3 = (Button) findViewById(R.id.button3);
    button3.setOnClickListener(listener3);
}
}

```

代码解释:

- setContentView(R.layout.main) 负责当前的 Activity 与 main.xml 文件相关联。
- 然后再得到 4 个 Button 的引用, 并且给 Button 设置单击监听器。Button 是最常用的组件之一, 它就是一个按钮, 一般需要设置监听器来处理单击事件。
- 每一个监听器都是跳转到一个新的 Activity。

下面看一下 main.xml 文件当中的代码, 下边的代码描述了 ActivityMain 的布局信息。具体代码如下所示:


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/button0"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="非洲草原: FrameLayout 的使用" />
    <Button android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="个性化表单: RelativeLayout 的使用" />
    <Button android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="LinearLayout 和 RelativeLayout 互助使用" />
    <Button android:id="@+id/button3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="整齐的表单: TableLayout 的使用" />
</LinearLayout>
```

代码解释:

这是一个典型的 `LinearLayout` 布局的使用。在 `HelloWorld` 这个例子中，我们已经碰到了 `LinearLayout` 布局，现在我们来总结一下 `LinearLayout` 布局。

什么是 `LinearLayout`?

小知识

`LinearLayout` 是一种 `Android` 中最常用的布局之一，它将自己包含的子元素按照一个方向进行排列。方向有两种，水平或者竖直。这个方向可以通过设置 `Android:orientation="vertical"` 或者 `Android:orientation="horizontal"` 来实现，所有的元素排列都是一个接着一个的。如果是竖直排列，那么 `LinearLayout` 的元素就一个接着一个的从上到下竖直排列，例如，在 `ActivityMain` 的视图中，就是这样竖直的一个一个接着排列的。如果是水平排列，那么就是 `LinearLayout` 里边的子元素从左到右一个一个的进行排列。

3. 第三步

单击第一个 `Button` 按钮，转到一个地图的界面。此界面是一个 `FrameLayout` 布局。那么，我们学习一下什么是 `FrameLayout`。

什么是 `FrameLayout`?

小知识

一个 `FrameLayout` 对象就好比一块在屏幕上提前预订好的空白区域，然后可以填充一些元素到里边，比方说一张图片等。需要注意的是，所有的元素都被放置在 `FrameLayout` 区域最左上的区域。而且无法为这些元素指定一个确切的位置。如果一个 `FrameLayout` 里边有多个子元素，那么后边的子元素的显示会重叠在前一个元素上。

我们看一下 `activity_frame_layout.xml` 文件里边的代码，我们 `FrameLayout` 布局里填充了一个图片显示组件(`ImageView`)元素，而这个元素的背景就是一个地图，具体代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout Android:id="@+id/left"
xmlns:Android="http://schemas.Android.com/apk/res/Android"
Android:layout_width="fill_parent"
Android:layout_height="fill_parent">
<ImageView Android:id="@+id/photo" Android:src="@drawable/bg"
Android:layout_width="wrap_content"
Android:layout_height="wrap_content" />
</FrameLayout>
```

❏ 代码解释:

- Android:id 定义组件的 id, 在应用程序当中, 我们通过这个 id 就可以访问到定义的这个元素。
- Android:layout_width="fill_parent" 和 Android:layout_height="fill_parent" 表示 FrameLayout 布局可以在 x 轴方向和 y 轴方向填满父容器的空间。
- Android:layout_width="wrap_content" 和 Android:layout_height="wrap_content" 表示 ImageView 元素的长和宽只需要将 bg.jpg 包裹起来即可, 并不需要填满父容器。

4. 第四步

当单击第二个 Button 按钮时, 就看到了如图 7-4 所示的利用 RelativeLayout 构造的视图。那么, 现在就让我们再来学习一下另外一种在 Android 中常见的布局 RelativeLayout。

什么是 RelativeLayout?

❏ 小知识

从类的名字上可以猜测出。这是一个相对布局类。即首先 RelativeLayout 是一个容器, 它里边的元素, 如 Button 按钮等的位置是按照相对位置来计算的, 例如, 有两个 Button 按钮都布局在一个 RelativeLayout 里边, 我们可以定义第二个 Button 在第一个 Button 的上边或者是右边。但到底第二个 Button 在什么位置呢, 它还是依赖于第一个 Button 的位置。需要注意的是, 出于性能上的考虑, 对于相对布局的精确位置的计算只会执行一次, 所以, 如果一个可视化组件 B 依赖于 A, 那么必须要让 A 出现在 B 的前边。

了解了 RelativeLayout 后, 我们来具体看一下这个布局文件的代码, 看到底是怎么生成的当前这个界面的, 代码如下所示:

```
<?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="wrap_content"
Android:background="@drawable/blue" Android:padding="10dip">

<TextView Android:id="@+id/label" Android:layout_width="fill_parent"
Android:layout_height="wrap_content" Android:text="请输入用户名:" />

<!--
这个 EditText 放置在上边 id 为 label 的 TextView 的下边
-->
<EditText Android:id="@+id/entry" Android:layout_width="fill_parent"
Android:layout_height="wrap_content"
Android:background="@Android:drawable/editbox_background"
Android:layout_below="@id/label" />
```

```

<!--
取消按钮和容器的右边齐平，并且设置左边的边距为 10dip
-->
<Button Android:id="@+id/cancel" Android:layout_width="wrap_content"
Android:layout_height="wrap_content" Android:layout_below="@id/entry"
Android:layout_alignParentRight="true"
Android:layout_marginLeft="10dip" Android:text="取消" />

<!--
确定按钮在取消按钮的左侧，并且和取消按钮的高度齐平
-->
<Button Android:id="@+id/ok" Android:layout_width="wrap_content"
Android:layout_height="wrap_content"
Android:layout_toLeftOf="@id/cancel"
Android:layout_alignTop="@id/cancel" Android:text="确定" />

</RelativeLayout>

```

代码解释：

- `Android:id` 定义组件的 id。
- `Android:layout_width` 定义组件的宽度，目前，我们学过的有"wrap_content"（仅仅包裹住内容）和"fill_parent"（填充父容器）。
- `Android:layout_height`，定义组件的高度，和上边一样。
- `Android:background="@drawable/blue"` 定义组件的背景，在这里系统会解析到@drawable/blue 的值为#770000ff，这是颜色的 RGB 码，RGB 是颜色的一种编码表示，具体可以上网查看一下相应资料。
- `Android:padding="10dip"`,dip 的意思是依赖于设备的像素。是描述区域大小的一种单位。和 HTML 类似，在 Android 中也有 padding 和 margin 的概念。padding 表示填充，margin 表示边距。
- `Android:layout_below="@id/label"`，按照字面的意思比较容易理解，就是将当前组件放置于 id 为 label 组件的下方。这就是经典的相对布局。这种布局的好处就是不用关注很多细节，而且它的适配性很强，在不同大小的屏幕或者手机设备上都可以使用。所以，我们在 Android 的界面布局中，推荐这种相对布局，而不推荐按照像素精确布局。当然有些时候按照像素精确布局是必要的，而且有时候是必须的。
- `Android:layout_alignParentRight="true"`，这个也可以从字面上来理解，就是和父容器的右边齐平。这也是相对布局。
- `Android:layout_marginLeft="10dip"`，设置 id 为 Cancel 的 Button 的左边距为 10dips。
- `Android:layout_toLeftOf="@id/cancel"`，设置此组件在 id 为 Cancel 的组件的左边。
- `Android:layout_alignTop="@id/cancel"`，设置此组件和 id 为 Cancel 组件的高度齐平。

小实验

上边的这种布局，不需要运行程序，直接在 Eclipse 里边就可以看到显示结果。所以，我们用 XML 进行布局是非常方便的。因为，我们的修改可以即时的显示出来。双击 relative_layout.xml 文件，然后单击 Layout 按钮，就可以看到布局了。在上面的文件当中，可以对相对布局进行修改。如去掉 `Android:layout_alignParentRight="true"`，或者将 `Android:layout_marginLeft="10dip"` 改为 `Android:layout_marginLeft="120dip"`。看看结果，是不是界面已经变化了？

写程序最重要的是尝试，不要怕出错，多尝试，多调试，进步才会快，这也是笔者学习 Android 的一点体会，显示如图 7-8 所示。

小知识

Android 当中支持的描述大小区域的类型如下。

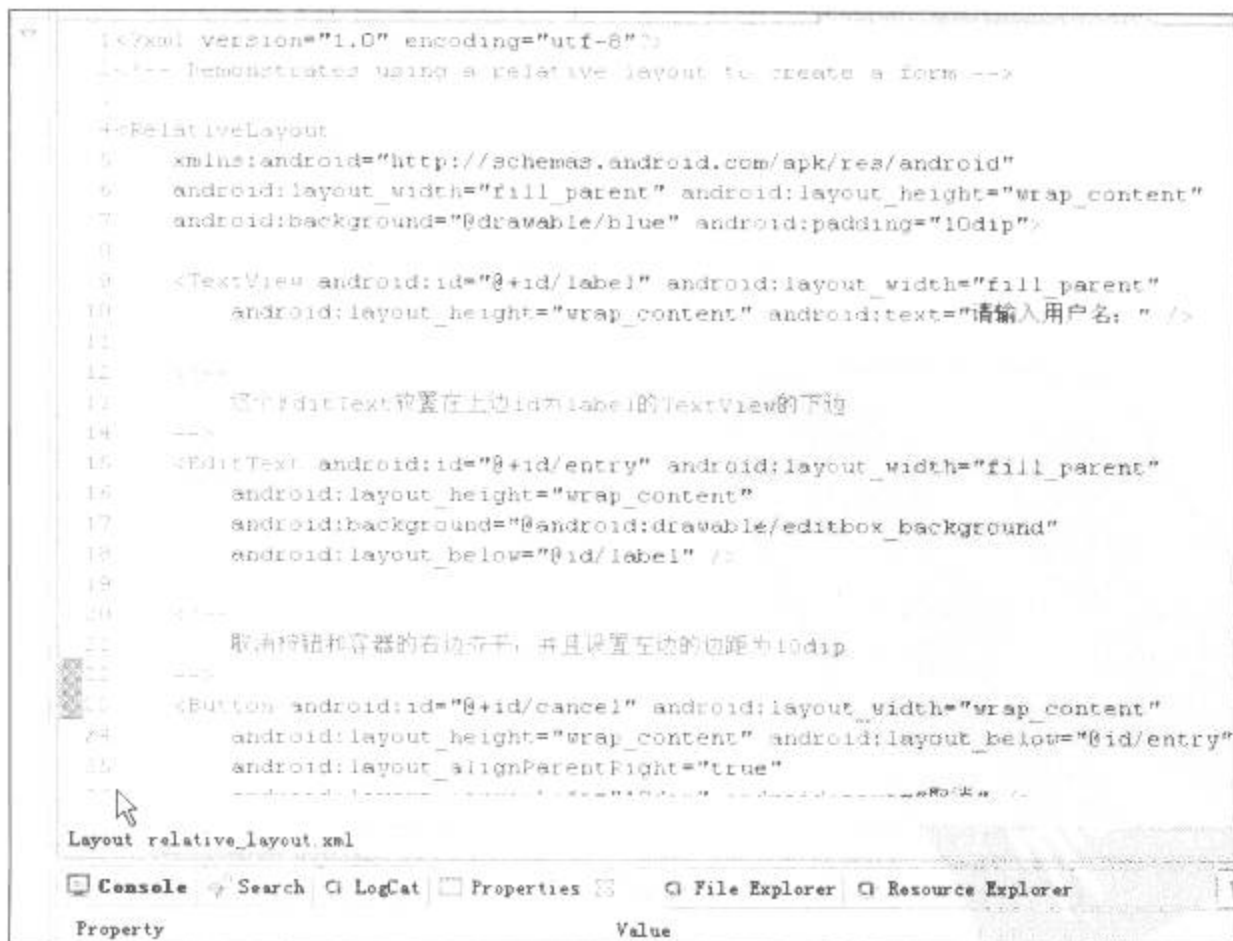
- px (pixels): 像素。
- dip (device independent pixels): 依赖于设备的像素。
- sp (scaled pixels — best for text size): 带比例的像素。
- pt (points): 点。
- in (inches): 英尺。
- mm (millimeters): 毫米。

虽然有这么多的表达方式，但是最常用的是 px 和 dip。

5. 第五步

在程序的主界面单击第三个按钮，即单击“LinearLayout 和 RelativeLayout 互助使用”这个按钮后，出现如图 7-5 的界面。下边来看一下这个比较复杂的界面是如何生成的。

在图 7-5 的文字标识为“第一组第一项”和“第一组第二项”这两个区域是属于一个 RelativeLayout，这个布局在 left.xml 文件当中定义。打开 left.xml 文件，对里边的代码进行简单地分析一下，具体代码如下所示：



▲ 图 7-8 运行及显示的结果界面

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    Android:id="@+id/left"
    xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent">

    <TextView Android:id="@+id/view1" Android:background="@drawable/blue"
        Android:layout_width="fill_parent"
        Android:layout_height="50px" Android:text="第一组第一项" />
    <TextView Android:id="@+id/view2"
        Android:background="@drawable/yellow"
        Android:layout_width="fill_parent"
        Android:layout_height="50px" Android:layout_below="@id/view1"
        Android:text="第一组第二项" />
</RelativeLayout>
```

代码解释:

- 这是一个 RelativeLayout 布局，里边包括了两个 TextView。
- 对于第一个 TextView，定义了它的背景是蓝色：“@drawable/blue”，这个值是在 res/values/string.xml 里边获得的，具体为<drawable name="yellow">#77ffff00</drawable>里边的#77ffff00。
- 对于两个 TextView 都设置了高度为 50 个像素。
- 对于第二个 TextView 还设置了它的位置是在第一个 TextView 的下边，即 Android:layout_height="50px" Android:layout_below="@id/view1"。

在图 7-5 中的文字标识为“第二组第一项”和“第二组第二项”区域是属于另外一个 RelativeLayout，这个 RelativeLayout 在 right.xml 文件里边描述。具体的内容和 left.xml 比较相似，在这里就不一一进行重复的描述了。

打开 ActivityLayout 的文件，看一下里边的代码：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    LinearLayout layoutMain = new LinearLayout(this);
    layoutMain.setOrientation(LinearLayout.HORIZONTAL);
    setContentView(layoutMain);
    LayoutInflater inflate = (LayoutInflater)
        getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    RelativeLayout layoutLeft = (RelativeLayout) inflate.inflate(
        R.layout.left, null);
    RelativeLayout layoutRight = (RelativeLayout) inflate.inflate(
        R.layout.right, null);

    RelativeLayout.LayoutParams relParam = new RelativeLayout.LayoutParams(
        RelativeLayout.LayoutParams.WRAP_CONTENT,
        RelativeLayout.LayoutParams.WRAP_CONTENT);
    layoutMain.addView(layoutLeft, 100, 100);
    layoutMain.addView(layoutRight, relParam);
}
```

代码解释:

- 在以往的 Activity 中，我们让一个 Layout 和一个 Activity 关联，而这个 Layout 是在 XML

文件当中定义的。当然所有的 View 和 Layout 都可以直接在 Java 代码构造。在此 Activity 中，为了方便，我们自己构建一个 Layout，语句为：`LinearLayout layoutMain = new LinearLayout(this)`。

- `layoutMain.setOrientation(LinearLayout.HORIZONTAL)`语句作用，设置加入到这个 `LinearLayout` 中元素的布局方向为竖直方向。
- `setContentView(layoutMain)`。将我们构建的 `layoutMain` 和此 Activity 进行关联。
- `LayoutInflater inflate = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE)`。得到一个 `LayoutInflater` 对象。这个对象可以对 XML 的布局文件进行解析，并且生成一个 View。
- `RelativeLayout layoutLeft = (RelativeLayout) inflate.inflate(R.layout.left, null)`。调用 `inflate` 方法将 `left.xml` 进行解析，并生成一个 `RelativeLayout` 布局。
- `RelativeLayout.LayoutParams relParam = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,RelativeLayout.LayoutParams.WRAP_CONTENT)`。生成一个 `RelativeLayout` 适用的 `LayoutParams`。
- `layoutMain.addView(layoutLeft, 100, 100)`将 `layoutLeft` 添加到 `layoutMain` 当中，其中第一个参数是添加进去的 View，第二个参数是这个 View 的高度，第三个参数是这个 View 的宽度。
- `layoutMain.addView(layoutRight, relParam)`将 `layoutRight` 添加到 `layoutMain` 当中，其中第二个参数就是一个 `RelativeLayout.LayoutParams`。

什么是 `LayoutParams`？虽然在前边我们有过一些介绍，但是在这里再比较详细地介绍一下。

我们可以这么设想一下，在象棋的棋盘上，每一个棋子都占据一个位置，也就是每个棋子都有一个位置的信息，如这个棋子在三行五列，这里的“三行五列”就是棋子的 `LayoutParams`。那么同样的道理，假设在屏幕上一块区域是由一个 Layout 占领的，那么，如果想把一个 View 加入到这个区域去，开发者要么告诉这个 Layout 所需要多高多宽在某个方向，要么 Layout 就按照默认的值进行处理。这里边关于高、宽等的信息都封装在 `LayoutParams` 当中。一般如果将一个 View 添加到一个 Layout 当中，最好告诉 Layout 用户期望的布局方式，也就是将一个认可的 `LayoutParams` 传递进去。

所有的 `LayoutParams` 都是继承 `Android.view.ViewGroup.LayoutParams`。这个类只是简单的描述了宽和高，对于这两者都可以设置为下边 3 个值中的任何一种。

小知识

- 一个确定的值。如在例子当中的 `layoutMain.addView(layoutLeft, 100, 100)`语句，那 100 就是确定的值。
 - `FILL_PARENT`，这个值意味着 View 想和父容器一样大，也就是最小的填充(padding)。
 - `WRAP_CONTENT`，这个值意味着当前 View 的大小只需要包裹住 View 里边的内容即可。
- 有很多继承 `Android.view.ViewGroup.LayoutParams` 的具体实现类，如 `AbsoluteLayout`，它里边的 `LayoutParams` 就多了 X 和 Y 两个属性。

小知识

什么是 **LayoutInflater**?

这个类用来将 XML 的布局文件变成一个 **View** 的类。出于性能上的考虑,从 XML 的布局文件生成一个 **View** 类这个过程是在应用的构建阶段完成的,而不是在应用运行阶段进行的。典型的使用过程是在 **Activity** 的 **oncreate** 方法里边,代码如下所示:

```
LayoutInflater inflate = (LayoutInflater) getSystemService(Context.
    LAYOUT_INFLATER_SERVICE);
LinearLayout layout = (LinearLayout) inflate.inflate(R.layout.ex01_07,null);
```

- **LayoutInflater** 类需要通过 **getSystemService** 方法来获得,而不能直接实例化。这个类负责将一个 xml 的布局文件实例化,从而在代码中可以应用在 xml 定义的组件。

6. 第六步

当我们单击第四个 **Button** 按钮的时候,看到了一个很整齐的表单,这个表单就是由 **TableLayout** 来展示的。那么我们先来了解一下 **TableLayout**。

小知识

什么是 **TableLayout**?

从字面上了解 **TableLayout** 是一种表格式的布局。这种布局会把包含的元素以行和列的形式进行排列。表格的列数为每一行的最大列数。当然表格里边的单元格是可以为空。

在下边的代码当中一共有 3 列,即 3 个 **TableRow**,每一个 **TableRow** 里边都有两个单元格,具体代码如下所示:

```
<TableLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent" Android:layout_height="fill_parent"
    Android:stretchColumns="1">

    <TableRow>
        <TextView Android:text="用户名:" Android:textStyle="bold"
            Android:gravity="right" Android:padding="3dip" />

        <EditText Android:id="@+id/username" Android:padding="3dip"
            Android:scrollHorizontally="true" />
    </TableRow>

    <TableRow>
        <TextView Android:text="登录密码:" Android:textStyle="bold"
            Android:gravity="right" Android:padding="3dip" />

        <EditText Android:id="@+id/password" Android:password="true"
            Android:padding="3dip" Android:scrollHorizontally="true" />
    </TableRow>

    <TableRow Android:gravity="right">

        <Button Android:id="@+id/cancel"
            Android:text="取消" />
    </TableRow>
</TableLayout>
```

```
<Button Android:id="@+id/login"
    Android:text="登录" />
</TableRow>
</TableLayout>
```

❏ 代码解释:

- TableLayout 标签定义了一个表格布局(TableLayout)。
- TableRow 标签定义了表格布局里边的一行。每一行里边可以自由的加入一些组件,比如在上边我们主要添加了按钮组件和文本框组件。

7.3 不积跬步 无以至千里——常用 widget 组件介绍

通过前面的小节学习,读者应该已经对 Android 的组件关系有了一定的了解,掌握了如何编写一个界面的布局,那么现在就需要进入另外一个更具体的内容学习了,也就是那些可以放在布局中的组件。

Android 中提供了一个叫 widget 的包,其包含各种 UI 元素(大部分是可见的)在应用程序的屏幕中使用,如按钮、列表框、进度条、图片显示等,在进行 Android 编程前,读者应对这类组件有个大致的了解,知道每个组件的样式、使用的场景及使用的方法。在以后的编程中才能有的放矢、游刃有余。

本小节将在一个实例中,选取 Android 中比较常用的组件逐一介绍,并附带相关的使用方法和其对应的代码,使得读者能对这部分的内容有个非常直观的学习,后面章节如果用到这部分的内容,可以翻回来查询。

7.3.1 创建 widget 组件实例

在 Eclipse 中创建一个新的项目,名字为 ex07_WidgetDemo,实现步骤为。

(1) 新建一个项目。依次单击 File→New→Android Project 项。

(2) 在新建项目的对话框中,选择 Create new project in workspace 项。

(3) 输入项目的名字 ex07_WidgetDemo 和包的名字及 Activity 的名字。

创建项目的过程如图 7-9 所示。

如图 7-8 所示,在创建 ex07_WidgetDemo 项目的时候,创建了一个 MainActivity,这也就是应用程序的入口,打开其对应的文件 MainActivity.java,可以看到其对应的 onCreate 方法,实现代码如下所示:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```



▲ 图 7-9 创建项目

 代码解释:

从上面代码可以看出，它已经关联了一个布局模板文件 `main.xml`，下面将在这个主界面上放置一些按钮，单击每个按钮将进入和按钮对应的组件演示界面。一下我们将从按钮开始应用开发的组件之旅。

7.3.2 按钮 (Button) 介绍与应用

Button 按钮在整个组件中的地位非常特殊，其一般情况下扮演“终结者”的角色，例如，用户在一个界面上输入一些信息，很多情况下都有一个“确认”或者“取消”的按钮，用户的动作一般在这里结束，然后开始另外一系列的动作。在这个项目的主界面上放置一个按钮，看看它是如何使用的，实现代码如下所示：

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

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="这是 button" />

</LinearLayout>
```

 代码解释:

上述代码所示，在主界面放了一个 `Android:id` 为 `button` 的按钮，并设定其宽度 (`layout_width`) 和高度 (`layout_height`) 都是根据内容调整 (`wrap_content`)，并设定其显示的文字是“这是 button”，然后运行这个例子，可以看到其效果如图 7-10 所示。

然后看看如何监听并响应按钮的“单击”事件，先看实现代码：

```
private void find_and_modify_button() {
    Button button = (Button) findViewById(R.id.button);
    button.setOnClickListener(button_listener);
}
```

 代码解释:

上述代码首先获取模板中的这个 **Button**，然后使用 `setOnClickListener()` 方法设定其被单击的监听器为 `button_listener`，最后还需要申明一下这个 `button_listener`，其代码如下所示：

```
private Button.OnClickListener button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        setTitle("哎呦,button 被点了一下");
    }
};
```

 代码解释:

上述的监听器实现单击事件代码，这里只是使用 `setTitle` 将标题设置为“哎呦，button 被点了一下”，重新运行程序，然后单击按钮，将出现图 7-11 所示的效果。

这里介绍了按钮的使用，包括其定义和设定监听以及监听的响应方法，本书后面还会使用单击按钮，让其打开一个新的 **Activity** 的例子，这里先不细说，等再次用到按钮的地方详细讲解。



▲ 图 7-10 主界面上放个 Button 按钮



▲ 图 7-11 响应按钮被单击事件

7.3.3 文本框 (TextView) 介绍与应用

接着上面创建的这个例子项目，首先需要添加一个按钮，当这个按钮被单击的时候，显示用来讲解 TextView 的界面。和 7.3.2 小节一样，先在 main.xml 文件中添加一个 Button 的组件，并设置其 id 为 text_view_button，实现代码如下所示：

```
<Button Android:id="@+id/text_view_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="TextView" />
```

🔍 代码解释：

这里定义一个 id 为 “text_view_button” 的按钮，并设定其宽、高都是和内容自适应，其显示文字为 “TextView”。

然后在 MainActivity 中添加对这个按钮的单击动作的监听和响应，当发生单击事件的时候，将打开 TextView 的演示界面，首先在 find_and_modify_button 方法中添加获取 text_view_button 按钮的代码，具体如下所示：

```
Button text_view_button = (Button) findViewById(R.id.text_view_button);
text_view_button.setOnClickListener(text_view_button_listener);
```

🔍 代码解释：

和前面类似，这里设置其监听器是 text_view_button_listener，在单击的时候，需要打开新的界面，核心代码如下：

```
private Button.OnClickListener text_view_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, TextActivity.class);
        startActivity(intent);
    }
};
```


代码解释：

从上面的代码可以看出，当单击该按钮的时候，首先创建一个 Intent，然后调用其 setClass 方法设定其需要跳转的 Activity，这里也就是 TextActivity，最后调用 startActivity 打开这个界面。

那么，当前的任务就是创建一个 TextActivity，并在其上显示一个 TextView 的组件，创建 Activity 的方法和前面一样，输入其名字为 TextActivity，然后需要修改一下其 onCreate 方法，让其绑定到指定的模板上，实现代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("ViewTextActivity");
        setContentView(R.layout.text);
    }
}
```

代码解释：

从上述代码可以看出，我们将这个 TextActivity 的布局模板关联到 text.xml 上，并设置其 Title 为“TextActivity”，将在 text.xml 中添加一个 TextView，实现代码如下所示：

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

    <TextView
        android:id="@+id/text_view"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="#ffffff"
        android:padding="10dip"
        android:background="#cc0000"
        android:text="这里是 TextView，你可以在这里输入需要显示的文字信息.."
    />

</LinearLayout>
```

代码解释：

上面代码中已经加了一个 TextView 组件进去，并设定一些属性，Android:id 是这个组件的惟一性标识，Android:textSize 用来设定字体的大小，Android:padding 用来设定组件周围空隙的大小，Android:background 设定其背景颜色，android:textColor 用来设定字体的颜色，Android:text 设定其需要显示的值。结合上面的代码，这里表达的意思就是在周围空出 18dip，使用 20sp 的字，显示“这里是 TextView，你可以在这里输入需要显示的文字信息..”，在这之后，还有需要做一件不可或缺的工作，将 TextActivity 添加到 AndroidManifest.xml 文件中，将如下代码添加到<application>域内实现：

```
<activity android:name="ViewTextActivity"></activity>
```

最后运行一下这个项目，在主界面上单击“ViewText”按钮，可以看到如图 7-12 所示的效果。

在图 7-12 中可以发现，组件按照我们设定的功能正确地显示了，除了上面用到的一些属性外，TextView 还有很多其他的属性，读者可以参考 Android SDK 附带的文档。

TextView 一般使用在需要显示一些信息的时候,其不能输入,只能初始设定或者在程序中修改,如果需要在程序中动态的修改这个值,那么就需要使用其 Android:id 的值,下面编写一个方法来演示一下,实现代码如下所示:

```
private void find_and_modify_text_view() {
    TextView text_view = (TextView) findViewById(R.id.text_view);
    CharSequence text_view_old = text_view.getText();
    text_view.setText("修改前是: " + text_view_old + "\n 修改为:TextView 的值也是可以动态修改的.");
}
```

🔍 代码解释:

这段代码先是使用 findViewById 将放在 main.xml 中的那个 TextView 找出来,然后调用其自身的 getText 方法将其原来的值取出来保存为 text_view_old,然后使用 setText 修改其自身的值。其运行效果如图 7-13 所示。



▲ 图 7-12 TextView 显示界面



▲ 图 7-13 动态修改 TextView 的值

实现的效果如图 7-13 所示,从图可以看到,动态修改 TextView 的值非常直观和方便。

7.3.4 编辑框 (EditText) 介绍与应用

学习完 TextView 组建应用,接着看一个和其应用非常类似的组件 EditText,看此组件的名字就可以猜出它的大致功能,它是编辑框。和前面介绍 TextView 应用一样,首先在程序的主入口界面上添加一个名字为“EditText”的按钮,并设定其监听器在接收到单击事件时,程序打开展示 EditText 的界面,在 XML 文件中添加 Button 的代码如下所示:

```
<Button Android:id="@+id/edit_view_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="EditView" />
```

🔍 代码解释:

定义个 id 为“edit_view_button”的按钮,并设定其显示文字为“EditView”。

其响应方法代码如下:

```
private Button.OnClickListener edit_view_button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, EditTextActivity.class);
        startActivity(intent);
    }
};
```

 代码解释：

此代码实现和前面例子的逻辑基本上是一样的，只是在响应单击按钮的时候，这里创建的 Intent 将要跳转到下一个 Activity，即 EditTextActivity 上。按照这个应用的思路，修改其关联的模板文件，其 onCreate 实现方法代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTitle("EditTextActivity");
    setContentView(R.layout.editview);
}
```

 代码解释：

这里指定其对应的模板文件是 editview.xml，需要在其中添加一个 EditText 的组件，以便看一看其表现形式是怎样的，添加的代码如下所示：

```
<EditText Android:id="@+id/edit_text"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:text="这里可以输入文字" />
```

和前面看到的 TextView 代码差不多，只是其提供一个输入框可以供用户输入，其显示效果如图 7-14 所示。

在图 7-14 中显示的 EditText 中，可以让用户输入文字。通过编程也可以用 getText 获取用户输入的值。下面我们将添加一个功能，在该项目添加一个 Button 按钮，单击此按钮，可以获得 EditText 中的值，并在 Title 上显示。为了实现此功能，需要在 editview.xml 文件中添加一个 Button 组件，并在 EditTextActivity.java 中添加单击该按钮的响应方法。

首先是在 XML 文件中添加一个 Button 组件，实现代码如下所示：

```
<Button Android:id="@+id/get_edit_view_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="获取 EditText 的值" />
```

然后完成其响应代码，具体如下所示：

```
private Button.OnClickListener get_edit_view_button_listener = new
Button.OnClickListener() {
    public void onClick(View v) {
        EditText edit_text = (EditText) findViewById(R.id.edit_text);
        CharSequence edit_text_value = edit_text.getText();
        setTitle("EditText 的值:" + edit_text_value);
    }
};
```

 代码解释：

上述代码就是获取 EditText 的值，并显示在 Title 上，其运行效果如图 7-15 所示。



▲ 图 7-14 EditText 显示效果



▲ 图 7-15 获取 EditText 的值

上面分别讲解了如何使用和操作 `EditView` 组件，通过实例讲解，相信读者已经能掌握这个组件了，它一般使用在需要用户输入信息的时候。

7.3.5 多项选择 (CheckBox) 介绍与应用

除了 `EditText` 组件外，`CheckBox` 也是一个使用频繁的组件，一般用来提供给用户输入信息的组件，可以一次选择多个选项。这个组件解决了在不是很方便的手机屏幕上操作输入时，用选择组件供用户单击输入选项，显得非常有用。在 Android 中，和其他组件一样，也是在 XML 中定义，按照前面讲解的其他组件的编程思路，首先在 `main.xml` 文件中再增加一个按钮“`CheckBox`”，单击该按钮程序运行将展示介绍 `CheckBox` 的界面。

在 `main.xml` 文件中添加如下代码内容：

```
<Button Android:id="@+id/check_box"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="CheckBox" />
```

🔗 代码解释：

定义个 id 为“`check_box`”的按钮，并设定其宽度和高度都是自适应，并指定其显示的文字是“`CheckBox`”。其对应的响应方法代码如下所示：

```
private Button.OnClickListener check_box_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, CheckBoxActivity.class);
        startActivity(intent);
    }
};
```

🔗 代码解释：

和前面一样，在单击按钮的时候，启动 `CheckBoxActivity`，这个 `CheckBoxActivity` 就是用来展示 `CheckBox` 组件的，首先修改其 `onCreate` 方法，实现代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTitle("CheckBoxActivity");
    setContentView(R.layout.check_box);
}
```

 代码解释:

上述代码是将其关联在 `check_box.xml` 的布局模板上，然后需要在里面添加 `CheckBox` 组件，代码如下所示：

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

<CheckBox android:id="@+id/plain_cb"
    android:text="Plain"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>

<CheckBox android:id="@+id/serif_cb"
    android:text="Serif"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:typeface="serif"
/>

<CheckBox android:id="@+id/bold_cb"
    android:text="Bold"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textStyle="bold"
/>

<CheckBox android:id="@+id/italic_cb"
    android:text="Italic"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textStyle="italic"
/>
</LinearLayout>
```

 代码解释:

上述代码中实现添加了 4 个 CheckBox 组件，并设定各自的宽度和高度都是自适应的，程序运行的效果如图 7-16 所示。

然后，需要考虑一下如何处理 `CheckBox` 的选中或者未选中，在这个应用中添加一个 `Button` 按钮，单击 `Button` 按钮时，程序运行获取 4 个 `Checkbox` 的状态，进而将被选中的 `Checkbox` 的值显示在 `Title` 上。

首先在 `check_box.xml` 文件中增加一个 `Button` 按钮, 实现代码如下所示:



▲ 图 7-16 CheckBox 展示页面


```
<Button Android:id="@+id/get_view_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="获取 CheckBox 的值" />
```

然后编写其对应的单击响应方法，实现代码如下所示：

```
private Button.OnClickListener get_view_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        String r = "";
        if (plain_cb.isChecked()) {
            r = r + "," + plain_cb.getText();
        }
        if (serif_cb.isChecked()) {
            r = r + "," + serif_cb.getText();
        }
        if (italic_cb.isChecked()) {
            r = r + "," + italic_cb.getText();
        }
        if (bold_cb.isChecked()) {
            r = r + "," + bold_cb.getText();
        }
        setTitle("Checked: " + r);
    }
};
```

 代码解释：

上述代码功能就是按照预期设定编写的，目的是将被选中的 Checkbox 的值显示在 Title 上，实现效果如图 7-17 所示。



▲ 图 7-17 获取 CheckBox 的状态

7.3.6 单项选择（RadioGroup）介绍与应用

RadioGroup 提供了一种多选一的选择模式，也是经常应用的组件，这里将详细解说它的用法。按照前面的思路，首先在 main.xml 文件中添加一个按钮，单击该按钮会启动这个 RadioGroupActivity，首先添加 Button 的代码，具体如下所示：

```
<Button Android:id="@+id/radio_group_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="RadioGroup" />
```

 代码解释：

这个定义一个 id 为“radio_group_button”的按钮，并设定其宽度和高度都是和内容自适应，并设定其显示文字为“RadioGroup”。

其对应的响应方法代码如下：

```
private Button.OnClickListener radio_group_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, RadioGroupActivity.class);
        startActivity(intent);
    }
};
```

代码解释:

和前面看到的按钮单击响应逻辑一样,没有什么特殊的地方,这里将要打开的是 `RadioGroupActivity`,也就是我们将用来展示 `RadioGroup` 组件,首先修改一下其 `onCreate` 方法,实现代码如下所示:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.radio_group);
    setTitle("RadioGroupActivity");
    mRadioGroup = (RadioGroup) findViewById(R.id.menu);
    Button clearButton = (Button) findViewById(R.id.clear);
    clearButton.setOnClickListener(this);
}
```

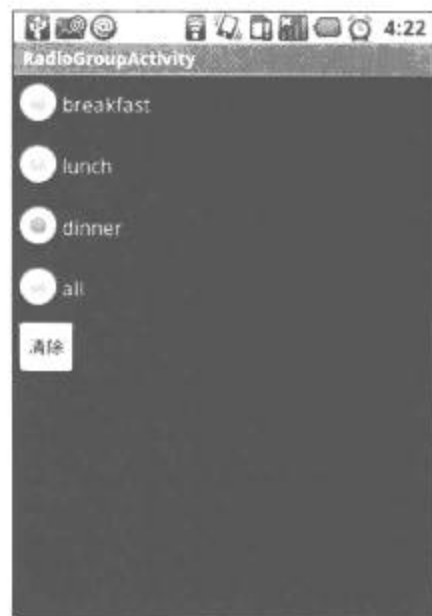
代码解释:

首先将其模板绑定到 `radio_group.xml` 上,然后使用 `setTitle` 修改 Title 为“RadioGroupActivity”,然后获取 `RadioGroup` 对象和按钮对象。首先来看其布局文件,关键代码如下所示:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <RadioGroup
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:checkedButton="@+id/lunch"
        android:id="@+id/menu">
        <RadioButton
            android:text="breakfast"
            android:id="@+id/breakfast"
            />
        <RadioButton
            android:text="lunch"
            android:id="@+id/lunch" />
        <RadioButton
            android:text="dinner"
            android:id="@+id/dinner" />
        <RadioButton
            android:text="all"
            android:id="@+id/all" />
    </RadioGroup>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="清除"
        android:id="@+id/clear" />
</LinearLayout>
```

代码解释:

提供了一个单选组(`RadioGroup`),其中包含了 4 种不同的单选按钮 (`Radio`),以及在下面有个清除 (`clear`) 的按钮。首先看 `RadioGroupActivity` 的代码就是上面说到的部分。这里运行一下程序,看到其效果如图 7-18 所示。



▲ 图 7-18 RadioGroup 界面

7.3.7 下拉列表 (Spinner) 介绍与应用

Spinner 提供一种下拉列表选择的输入方式，在手机应用界面上非常常见，其可以节省有限的屏幕空间占用。按照前面的讲解思路，它的实现也是首先在 main.xml 文件中添加一个按钮，单击按钮会启动这个 SpinnerActivity 文件。添加 Button 的代码如下所示：

```
<Button Android:id="@+id/spinner_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="Spinner" />
```

 代码解释：

定义一个 id 为 “spinner_button” 的按钮，并设定其宽度和高度是自适应，并指定其显示文字为 “Spinner”。

其对应的响应方法代码如下所示：

```
private Button.OnClickListener spinner_button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, SpinnerActivity.class);
        startActivity(intent);
    }
};
```

 代码解释：

上述代码实现的是，可以在该按钮被单击的时候，启动 SpinnerActivity，这个 SpinnerActivity 就是用来展示 Spinner 组件的界面。先创建 SpinnerActivity 这个 Activity，然后修改其 onCreate 方法，设定其对应的模板为 spinner.xml，实现代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTitle("SpinnerActivity");
    setContentView(R.layout.spinner);
}
```

 代码解释：

首先使用 setTitle 方法将其标题设置为 “SpinnerActivity”，并将 spinner.xml 绑定为其模板文件。

然后新建其对应的模板文件 spinner.xml，在其中添加一个 TextView 和一个 Spinner 组件，实现代码如下所示：

```
<TextView
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:text="Spinner_1"
/>
<Spinner Android:id="@+id/spinner_1"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:drawSelectorOnTop="false"
/>
```

这个模板文件中定义了一个 TextView 组件和一个 Spinner 组件。

接着在 AndroidManifest.xml 中添加如下代码：

```
<activity Android:name="SpinnerActivity"></activity>
```

代码解释：

这里定义的 Spinner 组件的 id 为 spinner_1，宽度是占满其父元素（也就是 LinearLayout）的宽，高为自适应。

运行这个实例程序，在出现的主界面上单击“Spinner”按钮，将显示图 7-19 所示的效果。

从图 7-19 上可以看出，界面上已经显示了一个下拉组件，但是其中还没有任何数值。下面就是实现往里面填一些数据，以便可以在列表数据中选择需要的数据。给 Spinner 组件装填数据比较常见的有两种方式。

（1）在编程的时候载入列表数据。

首先定义一下需要载入的数据，具体代码如下所示：

```
private static final String[] mCountries = { "China", "Russia", "Germany",
    "Ukraine", "Belarus", "USA" };
```

这里提供了几个国家名称供用户选择，然后在 onCreate 方法中调用 find_and_modify_view() 来载入数据，find_and_modify_view() 的代码如下所示：

```
private void find_and_modify_view() {
    spinner_c = (Spinner) findViewById(R.id.spinner_1);
    allcountries = new ArrayList<String>();
    for (int i = 0; i < mCountries.length; i++) {
        allcountries.add(mCountries[i]);
    }
    aspnCountries = new ArrayAdapter<String>(this,
        Android.R.layout.simple_spinner_item, allcountries);
    aspnCountries
        .setDropDownViewResource(Android.R.layout.simple_spinner_dropdown_item);
    spinner_c.setAdapter(aspnCountries);
}
```

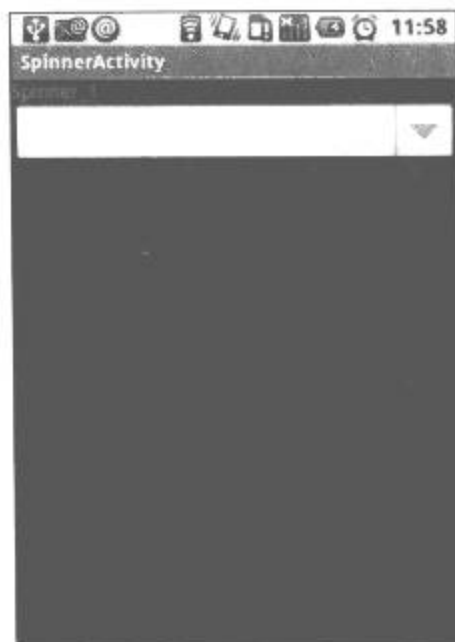
代码解释：

上述代码实现将定义的 mCountries 数据装载到 Spinner 组件中，运行一下实例，可以看到图 7-20 所示的效果。

（2）在 XML 文件中预先定义数据。

除了上面那种指定 Spinner 内容的方法外，还可以在 XML 文件中预先定义数据，为了说明这个用法，在 spinner.xml 这个模板中再添加一个 Spinner 组件，实现代码如下所示：

```
<TextView
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:text="Spinner_2 From arrays xml file"
/>
<Spinner Android:id="@+id/spinner_2"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:drawSelectorOnTop="false"
/>
```



▲ 图 7-19 Spinner 初始界面



▲ 图 7-20 有数据的 Spinner 显示效果

然后在 `SpinnerActivity.java` 中可以这样来初始化其值，实现代码如下所示：

```
spinner_2 = (Spinner) findViewById(R.id.spinner_2);
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.countries, Android.R.layout.simple_spinner_item);

    adapter.setDropDownViewResource(Android.R.layout.simple_spinner_dropdown_item);
    spinner_2.setAdapter(adapter);
```

❏ 代码解释：

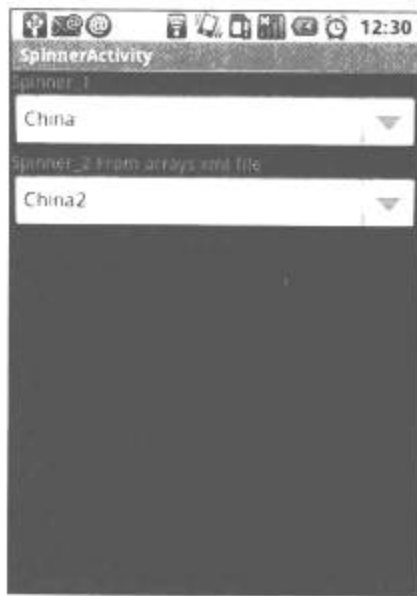
上述代码是将 `R.array.countries` 对应的值载入到 `Spinner_2` 中，而 `R.array.countries` 对应的值是在 XML 文件中预先定义的，其方法是在 `res/values/` 目录下新建一个名为 `array.xml` 的文件，输入如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Used in Spinner/spinner_2.java -->
    <string-array name="countries">
        <item>China2</item>
        <item>Russia2</item>
        <item>Germany2</item>
        <item>Ukraine2</item>
        <item>Belarus2</item>
        <item>USA2</item>
    </string-array>
</resources>
```

❏ 代码解释：

在此文件中预先定义了一组名字为 `countries` 的数组（array），最后使用 `setDropDownViewResource` 将其设置为 `Spinner_2` 的内容，运行程序可以看到图 7-21 所示的效果。

然后单击第二个 Spinner 右侧的下拉箭头，将看到图 7-22 所示的效果。



▲ 图 7-21 Spinner 界面



▲ 图 7-22 Spinner_2 显示效果

上述介绍了 Spinner 组件的使用方法和数值填充两个方法的实现，帮助读者可以很好地掌握这个组件。

7.3.8 自动完成文本（AutoCompleteTextView）

除了上面介绍的 Spinner 组件外，Android 还提供了一种节省界面空间的辅助输入方式

AutoCompleteTextView，看其名字就可以大致了解其特点就是可以自动完成。

和前面实现其他组件的编程思路一样，首先在 main.xml 文件上添加一个打开 AutoCompleteTextViewActivity 的入口按钮，XML 文件代码如下所示：

```
<Button Android:id="@+id/auto_complete_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="AutoCompleteTextView" />
```

 代码解释：

定义一个 id 为“auto_complete_button”的按钮，并设定其显示文字是“AutoCompleteTextView”。然后添加这个按钮的响应方法，实现代码如下所示：

```
private Button.OnClickListener auto_complete_button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, AutoCompleteTextViewActivity.class);
        startActivity(intent);
    }
};
```

 代码解释：

如上代码逻辑是，单击按钮打开 AutoCompleteTextViewActivity。

然后，需要创建 AutoCompleteTextViewActivity.java 文件，并设定其模板为 autocomplete.xml，在 autocomplete.xml 文件中添加一个 AutoCompleteTextView 组件，实现代码如下所示：

```
<AutoCompleteTextView Android:id="@+id/auto_complete"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"/>
```

 代码解释：

定义一个 id 为“auto_complete”的 AutoCompleteTextView 组件，并设定其宽度和高度和内容自适应。

然后在 AutoCompleteTextViewActivity.java 文件中完成将数据绑定到这个组件上。首先定义一组需要绑定的数据：

```
static final String[] COUNTRIES = new String[] {
    "China", "Russia", "Germany",
    "Ukraine", "Belarus", "USA", "Chinal", "Chinal2", "Germany",
    "Russia2", "Belarus", "USA"
};
```

然后在 onCreate 中绑定，实现代码如下所示：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.autocomplete);
    setTitle("AutoCompleteTextViewActivity");
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        Android.R.layout.simple_dropdown_item_1line, COUNTRIES);
    AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.auto_
complete);
    textView.setAdapter(adapter);
}
```

代码解释:

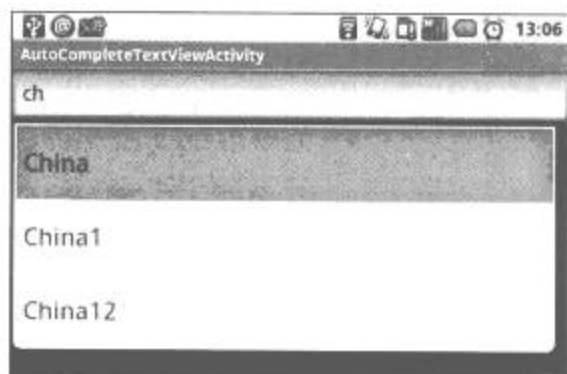
设定布局模板文件和标题后, 根据定义的 COUNTRIES 创建 ArrayAdapter, 并将其绑定到 AutoCompleteTextView 组件上。

最后在 AndroidManifest.xml 文件中添加 Activity 的申明, 实现代码如下所示:

```
<activity
    Android:name="AutoCompleteTextViewActivity"></activity>
```

运行项目, 在主界面上单击“AutoCompleteTextView”按钮, 打开 AutoCompleteTextViewActivity 的界面, 在输入框中输入“ch”, 可以看到其自动显示能匹配的条目, 如图 7-23 所示。

AutoCompleteTextView 的自动完成功能能给用户提供非常棒的体验。若想在上面的基础上进一步学习, 推荐读者仔细阅读 Android SDK 中相关的内容。



▲ 图 7-23 AutoCompleteTextView 显示效果

7.3.9 日期选择器 (DatePicker) 介绍与应用

DatePicker 是一个日期选择的组件, 可以提供快速选择日期的方式, 本小节演示如何使用 DatePicker。首先在 main.xml 文件中添加一个按钮, 用以打开演示 DatePicker 组件的界面, 实现代码如下所示:

```
<Button Android:id="@+id/date_picker_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="DatePicker" />
```

代码解释:

定义一个 id 为“date_picker_button”的按钮, 并设定其显示文字为“DatePicker”。其对应的响应方法代码如下所示:

```
private Button.OnClickListener date_picker_button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, DatePickerActivity.class);
        startActivity(intent);
    }
};
```

代码解释:

上述代码实现了单击按钮“DatePicker”时, 跳转到 DatePickerActivity 上, 和前面的组件实现一样, 创建一个 Activity 组件, 并在其 onCreate 方法中指定需要绑定的模板文件为 date_picker.xml, onCreate 的实现代码如下所示:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTitle("CheckBoxActivity");
    setContentView(R.layout.date_picker);
    DatePicker dp = (DatePicker) this.findViewById(R.id.date_picker);
    dp.init(2009, 5, 17, null);
}
```

代码解释:

在上面的代码中, 实现了获取 date_picker.xml 中的 DatePicker 组件 date_picker, 并设定其开始时间为 2009 年 5 月 17 日, 在 date_picker.xml 文件中添加 DatePicker 组件的代码如下所示:

```
<DatePicker
    Android:id="@+id/date_picker"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content" />
```

代码解释:

定义 DatePicker 组件, 其 id 为 date_picker, 宽度和高度都是自适应。

最后在 AndroidManifest.xml 文件中添加一个 Activity 的申明, 实现代码如下所示:

```
<activity Android:name="DatePickerActivity"></activity>
```

运行程序, 在主界面上单击“DatePicker”按钮, 展示的 DatePicker Activity 如图 7-24 所示。

这里需要提醒的是, DatePicker 还有其他展示方式, 这里所演示的只是其中的一种, 其他展示和使用方式, 相信通过这里的介绍, 读者可以举一反三。



▲ 图 7-24 DatePicker 节目

7.3.10 时间选择器 (TimePicker) 介绍与应用

TimePicker 是时间选择组件, 可以提供快速选择和调整时间的方式, 本小节演示如何来使用 TimePicker 组件。首先在 main.xml 中添加一个按钮用以打开演示 TimePicker 的界面, 实现代码如下所示:

```
<Button Android:id="@+id/time_picker_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="TimePicker" />
```

代码解释:

定义一个 id 为“time_picker_button”的按钮, 并设定其显示文字为“TimePicker”。

其对应的响应方法代码如下所示:

```
private Button.OnClickListener time_picker_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, TimePickerActivity.class);
        startActivity(intent);
    }
};
```

代码解释:

上述代码实现了单击按钮“TimePicker”时, 跳转到 TimePickerActivity 上, 和前面讲解的组件实现一样, 创建一个 Activity, 并在其 onCreate 方法中指定需要绑定的模板为 time_picker.xml, onCreate 实现代码如下所示:

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTitle("TimePickerActivity");
    setContentView(R.layout.time_picker);
    TimePicker tp = (TimePicker)this.findViewById(R.id.time_picker);
    tp.setIs24HourView(true);
}

```

 代码解释：

上述代码中，首先指定其对应的布局模板是 `time_picker.xml`，然后获取其中的 `TimePicker` 组件，进而进行一些设置。接着看其 `time_picker.xml` 中关于 `TimePicker` 的定义，实现代码如下：

```

<TimePicker
    Android:id="@+id/time_picker"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"/>

```

 代码解释：

定义一个 `id` 为 “`time_picker`” 的时间选择组件。

最后在 `AndroidManifest.xml` 中添加一个 `Activity` 的申明，实现的代码如下所示：

```

<activity
    Android:name="TimePickerActivity"></activity>

```

运行程序，在主界面上单击 “`TimePicker`” 按钮，展示的 `TimePickerActivity` 如图 7-25 所示。



▲ 图 7-25 `TimePickerActivity` 界面

这里需要提醒一下的是，`TimePicker` 的属性比较多，定义不同的属性所表现的外观差异也比较大，读者通过了解这里介绍的基础后，可以很快熟悉其他特性。

7.3.11 滚动视图（`ScrollView`）介绍与应用

这里演示的每个组件都是在主界面上增加一个按钮，单击按钮打开其各自的界面。现在看一个主界面，如图 7-26 所示。

从图 7-26 上可以看出，内容差不多已经占满了一个屏幕，再增加按钮或内容的话就被挡住看不见了。若希望在一个屏幕显示内容不完全的情况下，可以往下滚动以显示被挡住的部分。这个时候就需要一个组件 `ScrollView`。

`ScrollView` 的功能主要就是将一个屏幕显示不了的内容，通过滚动显示出来，使用这个组件也比较直观，直接在 `LinearLayout` 外面再增加 `ScrollView` 组件申明即可，实现代码如下所示：

```

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <LinearLayout>....</LinearLayout>
</ScrollView>

```

 代码解释：

这里将 `ScrollView` 套在 `LinearLayout` 外面，使得当 `LinearLayout` 的内容超过一个屏幕的时候，

可以滚动浏览。

上述代码完成后，运行程序，其主界面如图 7-27 所示。



▲ 图 7-26 主界面



▲ 图 7-27 出现了滚动条的界面

7.3.12 进度条 (ProgressBar) 介绍与应用

ProgressBar 是个非常有用的组件，其最直观的感觉就是进度条显示，但是在 Android 中，进度条也有很多种，这里选择两个最常见的说明，首先在 main.xml 中增加一个按钮，实现代码如下所示：

```
<Button Android:id="@+id/progress_bar_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="ProgressBar" />
```

🔗 代码解释：

定义了一个名字为 ProgressBar 的 Button。

然后实现其单击操作，打开进度条演示界面，具体代码如下所示：

```
private Button.OnClickListener progress_bar_button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, ProgressBarActivity.class);
        startActivity(intent);
    }
};
```

🔗 代码解释：

当单击按钮是，启动 ProgressBarActivity。

然后，创建 ProgressBarActivity.java 文件，指定其对应的布局文件是 progress_bar.xml，在 progress_bar.xml 文件中添加两种表现显示的进度条，实现代码如下所示：

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

    <TextView
        Android:layout_width="wrap_content"
```



```

        Android:layout_height="wrap_content"
        Android:text="圆形进度条" />
<ProgressBar
    Android:id="@+id/progress_bar"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"/>

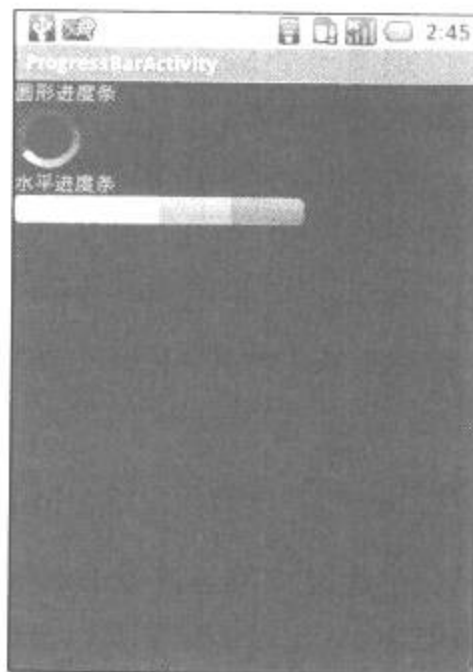
<TextView
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="水平进度条" />
<ProgressBar Android:id="@+id/progress_horizontal"
    style="?Android:attr/progressBarStyle Horizontal"
    Android:layout_width="200dip"
    Android:layout_height="wrap_content"
    Android:max="100"
    Android:progress="50"
    Android:secondaryProgress="75" />
</LinearLayout>

```

代码解释：

上述代码中添加了两个 **ProgressBar** 组件，并制定第二个进度条组件（**progress_horizontal**）为水平进度条，并设定其进度最大值是 100，且有两个进度，第一个进度到 50，第二个进度到 75。运行实例程序，单击“**ProgressBar**”按钮打开其演示界面，可以看到运行效果如图 7-28 所示。

定义的进度条是可以通过代码实现实时更新的，如在资源下载等场景下，这些进度条提示下载进度显得非常友好。



▲ 图 7-28 进度条演示界面

7.3.13 拖动条（SeekBar）介绍与应用

SeekBar 组件和水平的 **ProgressBar** 组件功能有些相似，不过其不同点在于 **SeekBar** 可以被拖动。首先在 **main.xml** 中放置一个演示 **SeekBar** 的按钮，实现代码如下所示：

```

<Button Android:id="@+id/seek_bar_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="SeekBar" />

```

代码解释：

定义一个 id 为“**seek_bar_button**”的按钮，并设定其显示文字为“**SeekBar**”。

按钮的响应方法代码为：

```

private Button.OnClickListener seek_bar_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, SeekBarActivity.class);
        startActivity(intent);
    }
};

```

代码解释：

上述代码实现的是，单击该按钮，跳转到 **SeekBarActivity**。创建一个 **Activity**，并将其模板绑

定到 seek_bar.xml 上, 在 seek_bar.xml 中添加 SeekBar 的组件, 实现代码如下所示:

```
<SeekBar
    Android:id="@+id/seek"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:max="100"
    Android:thumb="@drawable/seeker"
    Android:progress="50"/>
```

❏ 代码解释:

定义一个 SeekBar 组件, 设定其 id 为 seek, 宽度占满屏幕, 并设定其最大值是 100。

最后在 AndroidManifest.xml 中增加 SeekBarActivity 的申明, 实现代码如下所示:

```
<activity Android:name="SeekBarActivity"></activity>
```

运行实例程序, 在其主界面上单击“SeekBar”, 打开演示界面如图 7-29 所示。



▲ 图 7-29 SeekBar 界面

7.3.14 评分组件 (RatingBar) 介绍与应用

在让用户参与评分的时候, 用 RatingBar 组件实现非常方便, 第一方便用户输入, 第二直观。首先在 main.xml 中增加一个打开 RatingBar 界面的按钮, 实现代码如下所示:

```
<Button Android:id="@+id/rating_bar_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="RatingBar" />
```

❏ 代码解释:

定义一个 id 为 “rating_bar_button” 的按钮, 其显示的文字是 “RatingBar”。

其对应的响应代码如下所示:

```
private Button.OnClickListener rating_bar_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, RatingBarActivity.class);
        startActivity(intent);
    }
};
```

❏ 代码解释:

上述代码实现的是, 单击按钮时跳转到 RatingBarActivity, 这个 RatingBarActivity 就是用来演示 RatingBar 的, 创建一个 Activity, 并制定其模板文件为 rating_bar.xml, 然后在模板文件中添加 RatingBar 组件的代码如下所示:

```
<RatingBar Android:id="@+id/rating_bar"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    ratingBarStyleSmall="true" />
```

✎ 代码解释:

定义一个 RatingBar 组件, 设定其 id 为 rating_bar, 宽度和高度都是自适应。

然后在 AndroidManifest.xml 中增加 RatingBarActivity 的申明, 实现代码如下所示:

```
<activity Android:name="RatingBarActivity"></activity>
```

运行实例程序, 在主界面上单击“RatingBar”按钮, 进入 RatingBar 的效果显示页面, 如图 7-30 所示。



▲ 图 7-30 RatingBar 效果

7.3.15 图片视图 (ImageView) 介绍与应用

下面介绍的是 ImageView 组件, 其主要用作展示图片, 在很多场合下使用的非常普遍。首先在 main.xml 中增加一个入口按钮, 实现代码如下所示:

```
<Button Android:id="@+id/image_view_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="ImageView" />
```

✎ 代码解释:

定义一个 id 为“image_view_button”的按钮, 并设定其显示文字为“ImageView”。

其响应单击的代码如下所示:

```
private Button.OnClickListener image_view_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, ImageViewActivity.class);
        startActivity(intent);
    }
};
```

✎ 代码解释:

程序运行后, 单击按钮时跳转到 ImageViewActivity, 该 Activity 就是用来解释 ImageView 的。指定其需要绑定的模板文件是 image_view.xml, 在其中添加一个 ImageView, 实现代码如下所示:

```
<ImageView
    Android:id="@+id/imagebutton"
    Android:src="@drawable/ee"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"/>
```

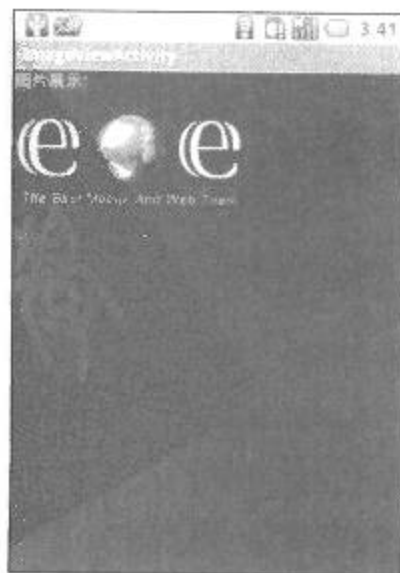
✎ 代码解释:

上述代码中, 指定 Android:src 为一张图片, 该图片位于项目根目录下的 src 下的 drawable, 支持 PNG、JPG、GIF 等图片格式, 此处选择 eoeMobile 团队的 Logo 演示。

最后在 AndroidManifest.xml 中加上对该 Activity 的申请, 实现代码如下所示:

```
<activity Android:name="ImageViewActivity"></activity>
```

运行程序实例, 在主界面上单击“ImageView”按钮, 跳转到 ImageView 界面, 如图 7-31 所示。



▲ 图 7-31 ImageView 界面

7.3.16 图片按钮 (ImageButton) 介绍与应用

接下来要介绍的是 **ImageButton** 组件，其可以提供一个图片按钮。首先在 **main.xml** 中增加一个入口按钮，其实现代码如下所示：

```
<Button Android:id="@+id/image_button_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="ImageButton" />
```

❏ 代码解释：

定义一个 **id** 为 “**image_button_button**” 的按钮，并设定其显示文字为 “**ImageButton**”。

其响应单击的代码如下所示：

```
private Button.OnClickListener image_button_button_listener = new Button.OnClickListener() {
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, ImageButtonActivity.class);
        startActivity(intent);
    }
};
```

❏ 代码解释：

程序运行后，单击按钮时，跳转到 **ImageButtonActivity**，该 **Activity** 就是用来演示 **ImageButton** 组件的。首先指定其需要绑定的模板文件是 **image_button.xml**，在其中添加一个 **ImageButton**，实现代码如下所示：

```
<ImageButton id="@+id/imagebutton"
    Android:src="@drawable/play"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"/>
```

❏ 代码解释：

上述代码中，指定 **Android:src** 为一张图片，该图片位于项目根目录下的 **src** 下的 **drawable** 下，支持 **PNG**、**JPG**、**GIF** 等图片格式，如这里选择一个播放图片。

最后在 **AndroidManifest.xml** 文件中添加上对该 **Activity** 的申请，实现代码如下所示：

```
<activity Android:name="ImageButtonActivity"></activity>
```

运行该程序实例，在主界面上单击 “**ImageButton**” 按钮，跳转到 **ImageButton** 界面，如图 7-32 所示。



▲ 图 7-32 ImageButton 界面

7.3.17 切换图片 (ImageSwitcher&Gallery)

下面要介绍的是 **ImageSwitcher** 和 **Gallery** 这两个组件，使用一个例子介绍这两个组件。这个例子相对比较复杂，其最终需要实现的是在屏幕上面展示一个大图，在屏幕的下面是一组可以滚动的图片，其中用到了一些比较复杂的東西。和前面讲解的其他组件的实例操作一样，首先在 **main.xml** 中增加一个入口按钮，其实现代码如下所示：

```
<Button Android:id="@+id/image_show_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="ImageSwitcher Gallery"/>
```

✎ 代码解释：

定义一个 id 为“image_show_button”的按钮，并设定其显示文字为“ImageSwitcher Gallery”。其响应单击的实现代码如下所示：

```
private Button.OnClickListener image_show_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, ImageShowActivity.class);
        startActivity(intent);
    }
};
```

✎ 代码解释：

程序运行后单击按钮时，跳转到 ImageShowActivity，该 Activity 就是用来演示 ImageSwitcher & Gallery 的。首先指定其需要绑定的模板文件是 image_show.xml，在其中添加 ImageSwitcher & Gallery，实现的代码如下所示：

```
<?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">

    <ImageSwitcher
        Android:id="@+id/switcher"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
        Android:layout_alignParentTop="true"
        Android:layout_alignParentLeft="true" />

    <Gallery Android:id="@+id/gallery"
        Android:background="#55000000"
        Android:layout_width="fill_parent"
        Android:layout_height="60dp"
        Android:layout_alignParentBottom="true"
        Android:layout_alignParentLeft="true"
        Android:gravity="center_vertical"
        Android:spacing="16dp" />
</RelativeLayout>
```

✎ 代码解释：

在这个布局文件里，外层的大 RelativeLayout 组件包含了两个控件：分别是 ImageSwitcher 和 Gallery，ImageSwitcher 是用来图片显示那块区域的控件，Gallery 是来控制底下那个图标列表索引用的。

然后看一下 ImageShowActivity.java 的代码，首先看其 onCreate 方法，实现代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);

    setContentView(R.layout.image_show);
    setTitle("ImageShowActivity");

    mSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
    mSwitcher.setFactory(this);
}
```



```

mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
    Android.R.anim.fade_in));
mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
    Android.R.anim.fade_out));

Gallery g = (Gallery) findViewById(R.id.gallery);
g.setAdapter(new ImageAdapter(this));
g.setOnItemClickListener(this);
}

```

代码解释:

上述代码中, `requestWindowFeature(Window.FEATURE_NO_TITLE)` 是使得这个 Activity 没有 titlebar, 进而这个图片显示区域会增大。Gallery 这个类和 ListView 这个类使用方法差不多, 也是要利用 `setAdapter` 来进行资源的设置。这里对 `BaseAdapter` 进行了封装, 通过 `getView` 这个函数来返回要显示的那个 `ImageView` 的, `getView` 实现代码如下所示:

```

public View getView(int position, View convertView, ViewGroup parent) {
    ImageView i = new ImageView(mContext);
    i.setImageResource(mThumbIds[position]);
    i.setAdjustViewBounds(true);
    i.setLayoutParams(new Gallery.LayoutParams(
        LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
    i.setBackgroundResource(R.drawable.picture_frame);
    return i;
}

```

代码解释:

`getView` 方法里动态生成一个 `ImageView`, 然后利用 `setLayoutParams`、`setImageResource` 和 `setBackgroundResource` 分别设定图片大小、图片源文件和图片背景。当图片被显示到当前屏幕的时候, 这个函数就会被自动回调来提供要显示的 `ImageView`。

下面就来讲解 `ImageSwitcher`, 在 `ImageSwitcherI` 中实现了 `ViewSwitcher.ViewFactory` 这个接口, 这个接口里有个方法 `makeView`, 它的实现代码如下所示:

```

public View makeView() {
    ImageView i = new ImageView(this);
    i.setBackgroundColor(0xFF000000);
    i.setScaleType(ImageView.ScaleType.FIT_CENTER);
    i.setLayoutParams(new ImageSwitcher.LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.FILL_PARENT));
    return i;
}

```

代码解释:

这个方法为 `ImageSwitcher` 返回了一个 `View`。 `ImageSwitcher` 调用过程是这样的, 首先要有一个 `Factory` 为它提供一个 `View`, 然后 `ImageSwitcher` 就可以初始化各种资源了。

注意

在使用一个 `ImageSwitcher` 之前, 一定要调用 `setFactory` 方法, 否则, `setImageResource` 这个方法会出现空指针异常。`ImageSwitcher` 的切换效果就是由上面最后两句实现的, 这里的 `setInAnimation` 是资源被读入到这个 `ImageSwitcher` 中时实现的动画效果, `setOutAnimation` 是资源文件从这个 `ImageSwitcher` 里消失的时候要实现的动画效果, 这里所有的动画都是从 `Android.R` 系统文件里读取的。

最后在 AndroidManifest.xml 文件中添加对该 Activity 的申请，实现代码如下所示：

```
<activity
    Android:name="ImageShowActivity"></activity>
```

运行这个实例程序，在主界面单击“ImageSwitcher Gallery”，其打开的就是这两个组件的显示效果，如图 7-33 所示。

从图 7-33 可以看到，其显示效果和用户体验都相当棒，以后如果需要做相册等图片应用，这两个组件可以很好地实现此功能。



▲ 图 7-33 ImageSwitcher 和 Gallery 效果

7.3.18 网格视图（GridView）介绍与应用

GridView 是格子视图，可以将很多的图片以一定的大小显示出来，在做类似相册、图片浏览等应用时非常有用。实现的操作步骤和前面讲解的其他组件实现一样，首先在 main.xml 文件中添加一个“GridView”按钮，实现代码如下所示：

```
<Button Android:id="@+id/grid_view_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="GridView"/>
```

🔗 代码解释：

定义一个 id 为“grid_view_button”的按钮，并设定其显示文字为“GridView”。其响应实现代码为：

```
private Button.OnClickListener grid_view_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, GridViewActivity.class);
        startActivity(intent);
    }
};
```

🔗 代码解释：

上述代码实现单击按钮后，跳转到 GridViewActivity。首先创建这个类，然后在其 onCreate 方法中添加如下代码：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.grid_view);
    setTitle("GridViewActivity");
    GridView gridview = (GridView) findViewById(R.id.grid_view);
    gridview.setAdapter(new ImageAdapter(this));
}
```

🔗 代码解释：

上述代码首先绑定其模板文件为 grid_view.xml，然后获取模板中的 GridView 组件，接着使用 setAdapter 方法给其绑定一个适配 ImageAdapter。紧接着需要了解的是 ImageAdapter 是如何实现的，实现代码如下所示：

```

public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView==null) { // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // 图片的引用
    private Integer[] mThumbIds = {
        R.drawable.grid_view_01, R.drawable.grid_view_02,
        R.drawable.grid_view_03, R.drawable.grid_view_04,
        R.drawable.grid_view_05, R.drawable.grid_view_06,
        R.drawable.grid_view_07, R.drawable.grid_view_08,
        R.drawable.grid_view_09, R.drawable.grid_view_10,
        R.drawable.grid_view_11, R.drawable.grid_view_12,
        R.drawable.grid_view_13, R.drawable.grid_view_14,
        R.drawable.grid_view_15, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7
    };
}

```

代码解释：

ImageAdapter 继承于 BaseAdapter，其构造方法 ImageAdapter 获取 Context，然后重要的是 getView 的实现，这里创建一个 ImageView，并设定对应的图片。mThumbIds 就是需要显示的图片。最后在 AndroidManifest.xml 中加上对该 Activity 的申请，实现代码如下所示：

```
<activity Android:name="GridViewActivity"></activity>
```

运行实例程序，在主界面单击“GridView”，展示的 GridView Activity 效果如图 7-34 所示。



▲ 图 7-34 GridViewActivity 效果

7.3.19 标签 (Tab) 介绍与应用

Tab 组件可以实现多个标签栏样式的效果, 单击每个标签, 打开其对应的内容界面。这里做简要说明, 首先在 main.xml 里增加一个按钮 “TabView”, 实现代码如下所示:

```
<Button Android:id="@+id/tab_demo_button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="TabView"/>
```

 代码解释:

定义一个 id 为 “tab_demo_button” 的按钮, 并设置其显示文字为 “TabView”。

添加其单击响应实现的方法代码如下所示:

```
private Button.OnClickListener tab_demo_button_listener = new Button.OnClickListener()
{
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setClass(MainActivity.this, TabDemoActivity.class);
        startActivity(intent);
    }
};
```

 代码解释:

单击后启动 Tab Demo Activity。

接着是完成 TabDemoActivity 的实现, 需要集成自 TabActivity, 其实现代码如下所示:

```
public class TabDemoActivity extends TabActivity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("TabDemoActivity");
        TabHost tabHost = getTabHost();
        LayoutInflater.from(this).inflate(R.layout.tab_demo,
            tabHost.getTabContentView(), true);
        tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("tab1")
            .setContent(R.id.view1));
        tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("tab2")
            .setContent(R.id.view2));
        tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("tab3")
            .setContent(R.id.view3));
    }
}
```

 代码解释:

上述代码功能是, 首先是 getTabHost 获取 TabHost, 然后绑定其模板, 进而绑定每个标签的内容关联。接着看模板文件 tab_demo.xml, 实现代码如下所示:

```
<?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">

    <TextView Android:id="@+id/view1"
        Android:background="@drawable/blue"
        Android:layout_width="fill_parent"
```

```

        Android:layout_height="fill_parent"
        Android:text="这里是 Tab1 里的内容。"/>

<TextView Android:id="@+id/view2"
    Android:background="@drawable/red"
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent"
    Android:text="这里是 Tab2, balabalal...." />

<TextView Android:id="@+id/view3"
    Android:background="@drawable/green"
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent"
    Android:text="Tab3"/>

</FrameLayout>

```

代码解释:

这里仅仅放置了 3 个 TextView 组件, 每个标签切换的时候, 显示其各自对应的 TextView, 别忘了和前面一样, 在 AndroidManifest.xml 文件中添加 TabDemoActivity, 最后运行实例程序, 在主界面上单击“TabView”, 其显示的 TabDemoActivity 效果如图 7-35 所示。

最后看一下项目的结构如图 7-36 所示。



▲ 图 7-35 TabDemoActivity 效果



▲ 图 7-36 目录结构

本小节介绍了 Android 中比较常用的 Widget 组件, 通过本小节的学习, 读者应该对 Widget 组件有了一个直观感受, 也了解了其各自的表现形式和基本的使用方法。但是这里需要再次提醒的是, 在介绍的时候都是比较简单直观的介绍, 没有深究每个组件的详细用法, 但是当掌握了这里介绍的基础后, 再去看 SDK 文档或者查 API 的时候会更有的放矢, 运用自如。另外, 这里介绍的都是 Android 提供的组件, 除了使用系统提供的组件, 还可以定制自己的组件, 会在后续的进阶书籍中详细讲解。

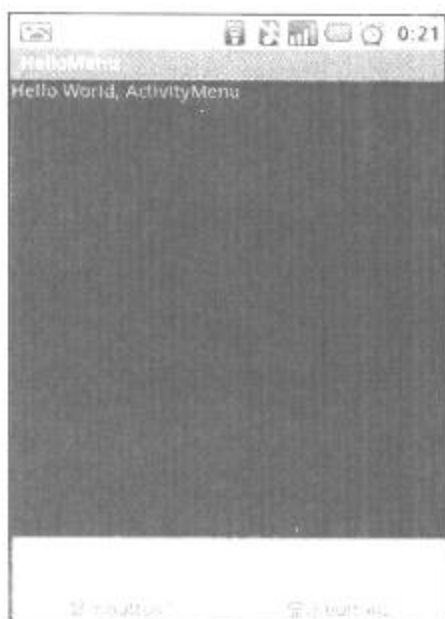
7.4 友好的菜单——menu 介绍与实例

在前边学习了 Android 中常用组件的用法和页面布局的技巧知识。在这一小节中, 要给大家介

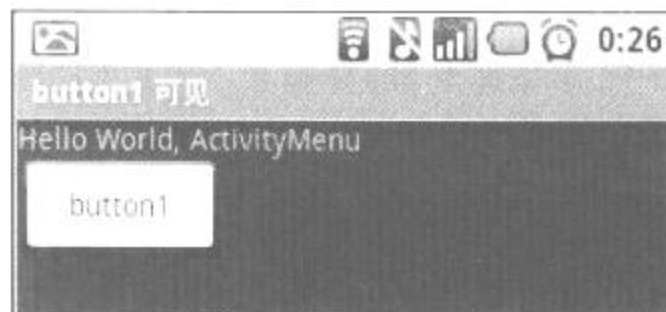
绍 Android 中另外一个比较特别的部分，menu 的使用。

7.4.1 实例操作演示

- (1) 进入程序主界面，并单击 menu 按钮，显示如图 7-37 所示。
- (2) 单击第一个按钮后。显示的界面如图 7-38 所示。



▲ 图 7-37 单击 menu 键后界面



▲ 图 7-38 单击第一个按钮后

7.4.2 实例编程实现

1. 第一步

在 Eclipse 中打开 ex07_menu 项目，具体操作步骤如下。

(1) 新建一个项目。依次单击 File→New→Android Project 项。

(2) 在新建项目的对话框中，选择 Create project from existing source 项。

(3) 单击浏览，找到 ex07_menu 项目，然后单击确定。程序目录结构如图 7-39 所示。

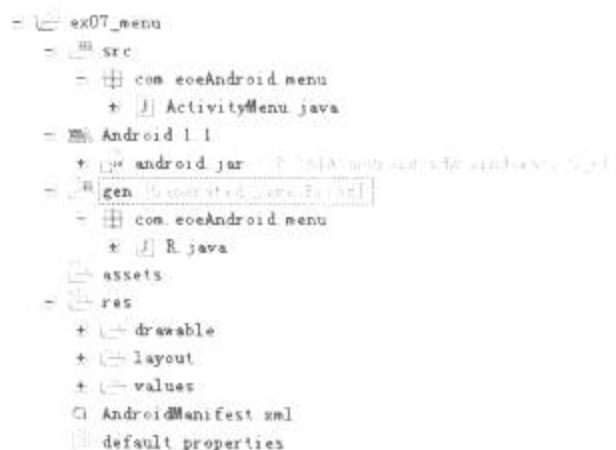
2. 第二步

在这一步当中仍然先看一下布局 XML 文件，在 Eclipse 当中，单击 layout 目录当中的 main.xml 这个 XML 文件，其代码如下所示：

```
<TextView Android:layout_width="fill_parent"
Android:layout_height="wrap_content" Android:text="@string/hello" />
<Button Android:id="@+id/button1"
Android:layout_width="wrap_content"
Android:layout_height="wrap_content" Android:text="@string/button1" />
```

代码解释：

- (1) TextView 这个是介绍过的组件，它代表一个文本框，负责在一块区域上边显示一段文本。



▲ 图 7-39 程序目录结构

(2) Android: layout_width 设置这个 Button 的宽度, 在这个例子中它的值为"wrap_content"。从字面上也很容易了解, Button 的宽度刚好是把显示在 Button 上的内容包裹住。

(3) Android: layout_height 设置这个 Button 的高度, 同理, 在上边这个例子中, 高度也是刚好把 Button 包裹住。

(4) 注意到@这个符号。这个符号提示 XML 文件的解析器应当对@后的名字进行解析, 例如, 在上边的@string/button1, 那么解析器会从 values/string.xml 里边读取 Button1 这个变量的值, 进行替换。也就是@string/button1, 相当于 <string name="button1">button1</string> 中的 Button1。

小实验

设置一个组件的高度和宽度除了用最常用的"wrap_content"外, 还可以用"fill_parent"。如果设置为 Android:layout_width="fill_parent", 表示在宽度方面尽可能地填充父容器。当然这里的值还可以是精确值, 例如, 我们设置为"100px", 表示宽度为 100 个像素。读者可以修改上边的代码, 然后单击编辑器左下角的 layout 按钮看看效果。

3. 第三步

下面来看 onCreate()方法, 具体代码如下所示:

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
button1 = (Button) findViewById(R.id.button1);
button2 = (Button) findViewById(R.id.button2);
button1.setVisibility(View.INVISIBLE);
button2.setVisibility(View.INVISIBLE);
```

代码解释:

上边代码负责显示出 main.xml 描述的 Layout, 并且设置里边的两个 Button 为不可见。

4. 第四步

接下来看 onCreateOptionsMenu()方法。

这个方法负责生成 menu, 它是一个回调方法, 即当按下手机设备上的 menu 按钮的时候, Android 系统才生成一个包含两个子项的菜单。先运行程序直观感觉一下, 然后再详细分析 menu 是怎么生成的, 具有两个步骤。

(1) 在 onCreateOptionsMenu()方法里边, 首先得到 super 函数调用后的返回值。并且在方法的最后进行返回。

(2) 调用 menu.add().给 menu 添加一项。

整个的代码如下边所示:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, ITEM0, 0, "显示 button1");
    menu.add(0, ITEM1, 1, "显示 button2");
```

```

        menu.findItem(ITEM1);
        return true;
    }

```

 代码解释：

- groupId: 分组的 id, 如果不用分组就用 0。
- itemId: menu 当中每一项的 id。这个 id 应该是惟一的。
- order: 排序的信息, 根据这个字段的大小, menu 对其中的项进行排序显示。order 小的显示在前边, order 大的显示在后边。
- titleRes: menu 项的显示信息。

我们在这里不妨多添加几项然后单击运行, 看一下效果怎么样。一般 itemId 需要是惟一的, 这里为了简单, 我们没有保证惟一。请将下面代码拷贝进去:

 小实验

```

menu.add(0, ITEM1, 8, "显示button3");
menu.add(1, ITEM1, 1, "显示button4");
menu.add(1, ITEM1, 1, "显示button5");
menu.add(0, ITEM1, 1, "显示button6");
menu.add(0, ITEM1, 1, "显示button7");

```

5. 第五步

实现 menu 响应单击事件。现在来看一下方法 `onOptionsItemSelected()` (MenuItem item), 由于该方法的前缀是 on, 所以依照惯例这依然是一个回调方法。即这个方法只有在事件触发后才会被 Android 系统自动调用执行。而这个事件就是单击菜单里的某一项 (MenuItem), 看到这个方法传进来的参数是 MenuItem, 也就是 menu 当中的某一项, 实现代码如下所示:

```

public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case ITEM0:
            actionClickMenuItem1();
            break;
        case ITEM1:
            actionClickMenuItem2(); break;
    }
    return super.onOptionsItemSelected(item);
}

```

 代码解释：

如何判断单击的那个 MenuItem 到底是哪一个呢? 这个时候前边设置过的 id 就发挥作用了。是的, 在前面生成一个 menu 的时候, 给每一个 menuItem 都赋予一个惟一的 id, 通过这个惟一的 id, 可以区分到底是单击到那个 menuItem 了。这个 id 通过 `item.getItemId()` 获得。在上边的代码中可以看到, 当用户单击 ITEM0 的时候执行 `actionClickMenuItem1()` 方法, 同理单击到 ITEM1 的时候, 执行 `actionClickMenuItem2()` 方法。这两个方法负责显示出一个 Button, 并且让另外一个 Button 消失。

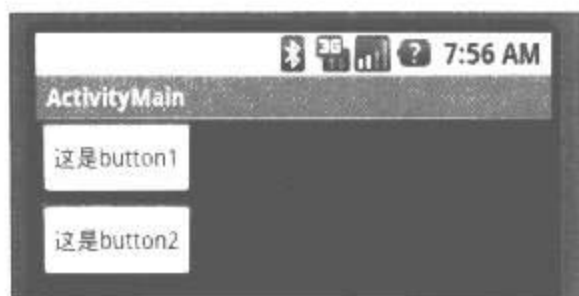
7.5 Android 应用的灵魂——Intent 和 Activity 介绍与实例

前边的一些例子中所有的操作都是在一个 Activity 中进行,但是现实中的应用往往包含好几个,甚至十几个或者几十个 Activity。这些 Activity 是可以相互进行跳转的,并且它们之间也可以相互传递数据。在这个例子中可以学到以下内容。

- (1) Intent 的介绍,第一种 Intent 使用方法的详细介绍。
- (2) 构建一个新的 Activity,然后将这个 Activity 加入到 Android 的 manifest 里。
- (3) 通过 startActivity()切换 Activity。
- (4) 通过 startActivityForResult()调用 Activity。
- (5) 不同的 Activity 之间进行相互数据的传递。

7.5.1 实例操作演示

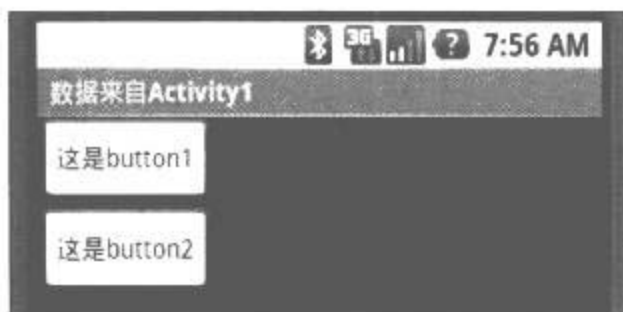
- (1) 程序运行主界面如图 7-40 所示。
- (2) 单击 button1 后进入的界面如图 7-41 所示。
- (3) 单击 button3 后进入的界面如图 7-42 所示。
- (4) 单击 button2 后进入的界面如图 7-43 所示。



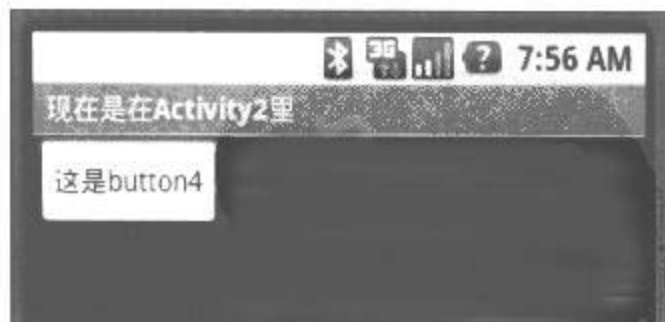
▲ 图 7-40 程序主界面



▲ 图 7-41 单击 button1 后进入界面



▲ 图 7-42 单击 button3 后进入界面



▲ 图 7-43 单击 button2 后进入界面

7.5.2 实例编程实现

1. 第一步

在 Eclipse 中打开 ex07_activity_intent 项目,实现步骤如下所示。

- (1) 新建一个项目。依次单击 File→New→Android Project 项。

(2) 在新建项目的对话框中, 选择 Create project from existing source 项。

(3) 单击浏览, 找到 ex07_activity_intent 项目, 然后单击确定。程序的目录结构如图 7-44 所示。

2. 第二步

首先看一下 ActivityMain 里的构造函数 onCreate, 代码如下所示:

```
setContentView(R.layout.main);
button1 = (Button) findViewById(R.id.button1);
button1.setOnClickListener(listener1);
button2 = (Button) findViewById(R.id.button2);
button2.setOnClickListener(listener2);
```

 代码解释:

通过前面的学习, 上述代码的功能现在应该很容易理解了。首先是先把 layout 目录当中 main.xml 中的 layout 显示出来。然后得到两个 Button 的应用, 依次各绑定一个单击监听器 (OnClickListener)。

3. 第三步

当单击 button2 的时候, 程序跳转到 Activity2。实现代码如下所示:

```
listener2 = new OnClickListener() {
    public void onClick(View v) {
        setTitle("这是在 ActivityMain");
        Intent intent2 = new Intent(ActivityMain.this, Activity2.class);
        startActivity(intent2);
    }
};
```

 代码解释:

当单击 button2 的时候, 系统会反向调用绑定在 button2 上的监听器的 onClick() 方法。

- setTitle("这是在 ActivityMain") 语句设置屏幕中的 title 区域。
- Intent intent2 = new Intent(ActivityMain.this, Activity2.class) 语句用来新建一个 Intent, 这个 Intent 描述了从 ActivityMain 跳转到 Activity2 的一次操作。
- startActivity(intent2) 语句用来启动 intent2, 由 intent2 描述的这次操作才开始正式执行。

小知识

什么是 Intent? 在 Android 的官方文档中是这么定义的, Intent 是一次即将操作的抽象描述。现在理解这个定义还有些抽象, 但是看完这本书就会对这个定义理解了。在 Android 当中, 一共用到了 3 种 Intent, 现在使用的这种 Intent 是第一种, 它的作用就是启动一个新的 Activity 并且可以携带数据。还有两种分别为:

- (1) 通过 Intent 来启动一个服务(Service)。
- (2) 通过 Intent 来广播一个事件。



▲ 图 7-44 程序目录结构

以上两种我们在第 9 章都有详细的介绍和讲解，这里就不再细述了。

4. 第四步

现在打开 Activity2.java 文件，会看到以下代码。

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity2);
    listener = new OnClickListener() {
    public void onClick(View v) {
        finish();
    }
    };
    button = (Button) findViewById(R.id.button4);
    button.setOnClickListener(listener);
    setTitle("现在是在 Activity2 里");
}
```

 代码解释：

由于 Activity2 也是一个新的 Activity，按照以前学过的知识，一个 Activity 一般需要和一个 Layout 进行关联。在这里看到和 layout 目录下边的 activity2.xml 相关联。在 Eclipse 当中，双击打开 activity2.xml 文件，看到一个 LinearLayout 里边包含了一个 Button。上边的代码实现的是，单击 Button 结束当前 Activity (finish())。

5. 第五步

当单击 button1 的时候，从 ActivityMain 跳转到 Activity1，不过这个跳转是有些特殊的，和前边介绍的不太一样。实现代码如下所示：

```
listener1 = new OnClickListener() {
    public void onClick(View v) {
        Intent intent1 = new Intent(ActivityMain.this, Activity1.class);
        intent1.putExtra("activityMain", "数据来自 activityMain");
        startActivityForResult(intent1, REQUEST_CODE);
    }
};
```

 代码解释：

startActivityForResult() 这个方法启动 Intent，并且当新的 Activity 运行结束的时候，还必须执行原 Activity 里的回调函数，这个函数叫做 onActivityResult()。将在后续的步骤当中来介绍这个方法。使用 startActivity() 方法启动一个新的 Activity，这种方法调用后，当新的 Activity 执行完后，不会执行回调函数，也不用返回任何值。也就是说新的 Activity 和旧的 Activity 没有什么关系，属于 Activity 的切换。

Activity 还可以传递数据，最常见的是传递一些名值对，这样在新的 Activity 当中也可以很方便地获得，当然了，通过 Intent 还可以传递其他类型的数据。在上边的例子中，将一组名值对放置进了 Intent，这组名值对可以在 Activity1 这个被 ActivityMain 启动的 Activity 中使用。

- putExtra() 方法将一组名值对添加到 Intent 的 mExtras 变量里边。它是一个 Bundle 类型。

Bundle 类型使用起来有点像 HashMap，而且通过如图 7-45 所示的截图可以看到，Bundle 的实现确实和 HashMap 分不开，最后的名值对其实也是存储到 Bundle 里边的 HashMap 当中了。

- `startActivityForResult(intent1, REQUEST_CODE)` 这个方法有两个参数，第一个参数是 Intent 类型，也就是即将要启动的 Intent。第二个参数是一个整型的请求码。如果请求码大于 0 的时候，当新的 Activity 结束的时候，这个请求码被返回到 `onActivityResult()` 的参数里边。通过不同的请求码使得在 Activity 的回调函数 `onActivityResult()` 中针对不同的 Activity 的返回值进行不同的代码操作。



▲ 图 7-45 Bundle 实现截图

小知识

Bundle 类型，在以后的程序当中也会用到很多 Bundle 类型的变量。现在对 Bundle 类型进行一个简单的介绍。Bundle 是一个类型安全的容器，它的实现其实就是对 HashMap 做了一层封装。对于 HashMap 来讲，任何名值对都可以存进去，值可以是任何的 Java 对象。但是对于 Bundle 不是这样的。同样存的是名值对，但是这个值只能是基本类型，或者基本类型的数组，如 String、int、byte、boolean、char 等。

6. 第六步

单击完 `button1` 后，Activity 进行了跳转，跳转到 `Activity1` 中。在 `Activity1` 中，程序将把从 `ActivityMain` 当中传递过来的值显示在屏幕的标题(title)区域，实现代码如下所示：

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
    data = extras.getString("activityMain");
}
setTitle("现在是在 Activity1 里:" + data);
```

代码解释：

- `getIntent()` 方法返回启动这个 Activity 的 Intent，在这里返回的就是 `ActivityMain` 中定义的 `intent1` 了。
- `getExtras()` 返回这个 Intent 的 Bundle 类型的变量 `eExtras`。在此变量中存储了 Intent 携带的数据。

7. 第七步

单击在 `Activity1` 当中的 `button3` 按钮后，当前 Activity 结束，并将需要返回的数据放置到 Intent 当中，实现代码如下所示：

```
listener1 = new OnClickListener() {
    public void onClick(View v) {
        Bundle bundle = new Bundle();
        bundle.putString("store", "数据来自 Activity1");
        Intent mIntent = new Intent();
```

```

mIntent.putExtras(bundle);
setResult(RESULT_OK, mIntent);
finish();
}
};

```

 代码解释:

- Bundle bundle = new Bundle()这条语句中，我们实现了一个 Bundle 的实例，这个实例用于在 Intent 中携带需要返回的数据。
- bundle.putString("store", "数据来自 Activity1")语句，将 String 类型的名值对放置到 bundle 中。
- mIntent.putExtras(bundle)语句将 Bundle 放置到 Intent 当中。
- setResult()的第一个参数是返回的结果码，结果码设置成 RESULT_OK，在这个例子运行正常。第二个参数就是需要返回给调用此 Activity 的 Intent。
- 调用 finish()结束当前的 Activity。

8. 第八步

下面看一下 ActivityMain 当中的 onActivityResult()方法。在 onClick()方法中，采用了异步的方式来启动 Intent，也就是说，启动完 Intent 后不会等待新的 Activity 返回值，而是等新的 Activity 执行完毕后再执行旧的 Activity 当中的回调函数。这个回调函数就是 onActivityResult(int requestCode, int resultCode, Intent data)。

此回调函数参数的意义为：

- requestCode：这个值就是原始 Intent 当中的 requestCode，例如，在这个例子当中是 REQUEST_CODE。
- resultCode：这个值暗示了在新的 Activity 当中的操作是否合法正确。如果一切都是正确的，那么返回值就是 0，如果返回值不是 0，那就说明程序运行出现了问题。Android 系统提供了一些标准的值，但是这里可以构造自己的值，以便发现程序到底出了什么问题了。
- Intent：当前 Activity 所启动的新的 Activity 执行完毕后返回这个 Intent。一般需要的只是从这个 Intent 当中得到新的 Activity 执行完后的返回值。这些值可以通过 extras.getString()方法获得。

startActivityForResult()和 onActivityResult()可以被认为是一组异步的远程调用。这种形式出现在 Activity 中的调用和 Service 的共享当中。

onActivityResult()中的代码如下所示：

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CODE) {
        if (resultCode == RESULT_CANCELED)
            setTitle("取消");
        else if (resultCode == RESULT_OK) {
            String temp=null;
            Bundle extras = data.getExtras();
            if (extras != null) {
                temp = extras.getString("store");
            }
            setTitle(temp);
        }
    }
}

```

```

}
}
}

```

✎ 代码解释：

- 根据不同的 requestCode 进行不同的处理，当然，在这个例子当中只用到了—种 requestCode。
- 根据不同的 resultCode 进行不同的处理，在这个例子当中有两种 resultCode，当 resultCode == RESULT_CANCELED 时候说明操作失败了；当 resultCode == RESULT_OK，说明一切正常，可以进行下一步的处理。

- 实现把从 Activity1 当中传递过来的值显示在屏幕的标题(title)区域。

9. 第九步

最后看一下 AndroidManifest.xml 文件，将新的 Activity 加进去，实现代码如下所示：

```

<activity Android:name=".Activity1"></activity>
<activity Android:name=".Activity2"></activity>

```

小知识

每一个应用都有一个 AndroidManifest.xml 文件，Android 系统是通过这个文件的内容来了解你的应用的相关信息，例如，通过这个文件可以了解这个应用里边有几个 Activity，有几个 Service，有几个 ContentProvider，还可以通过定义 IntentFilter 来告诉系统期望处理什么样的 Intent 等。

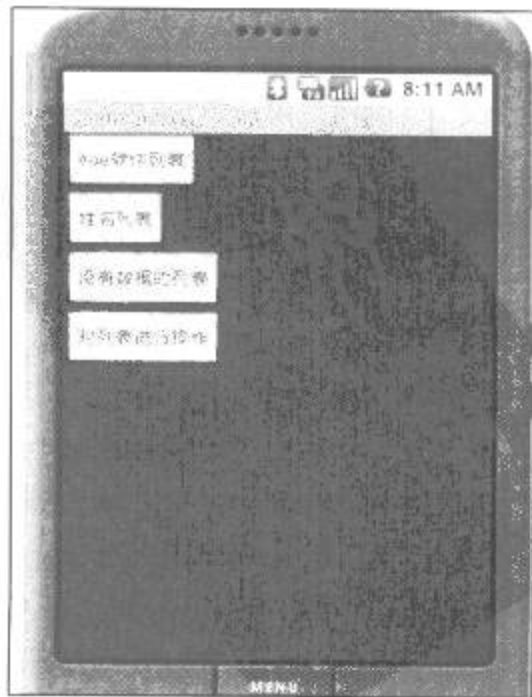
7.6 用好列表，做好程序——列表（ListView）介绍与实例

在先前的内容当中我们已经学习了界面布局、界面跳转，以及经常用的组件，如 Button、TextView 等。在这一节当中，将重点学习 Android 当中另外一个非常重要的一个组件 ListView，也就是常说的列表。运行此程序后，在程序主界面会看到 4 个 Button 按钮。单击每一个 Button 按钮后都会进入一个新的 Activity 界面，这个界面都会呈现出一个 ListView，但是每一个界面都会展现一些不同的东西，方便大家循序渐进的学习。下面先运行一下这个程序，依次单击这 4 个 Button 按钮看一下直观的效果是怎么样的。通过这个例子可以学到：

- 了解和使用 ListView；
- 了解和使用 ListActivity；
- 了解和使用 ListAdapter；
- 用 ArrayAdapter 结合 ListView 进行显示；
- 用 SimpleAdapter 结合 ListView 进行显示；
- 如何处理单击 ListView 的事件；
- 如何处理选择 ListView 的事件。

7.6.1 实例程序演示

(1) 程序运行后的主界面如图 7-46 所示。



▲ 图 7-46 主界面

(2) 单击第一个按钮后界面如图 7-47 所示。

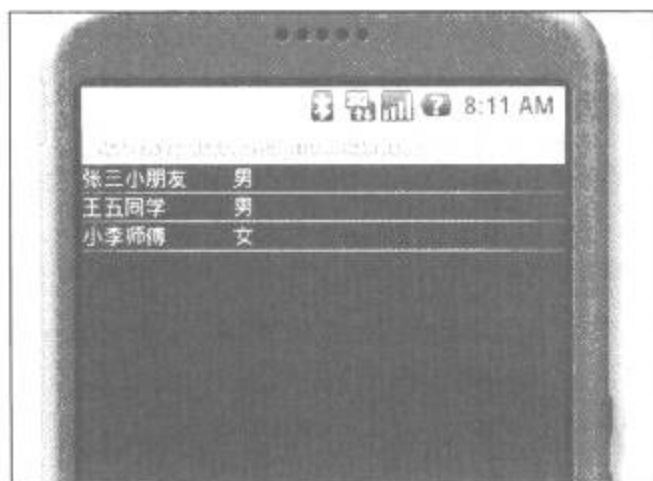
(3) 单击第二个按钮后界面如图 7-48 所示。

(4) 单击第三个按钮后界面如图 7-49 所示。

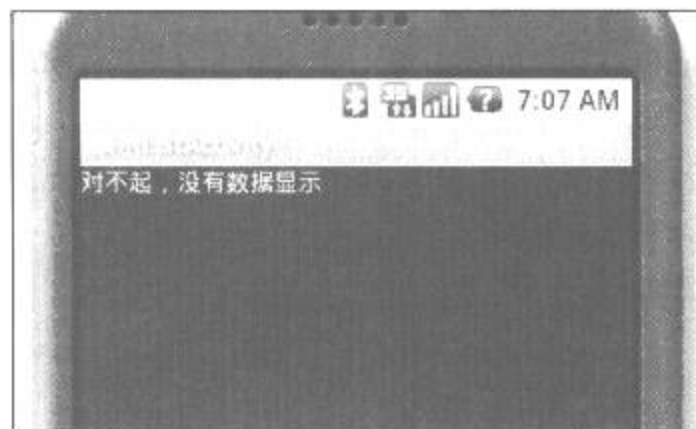
(5) 单击第四个按钮，选中 ListView 中的某一项后，这一项的内容在 title 区域显示出来，如图 7-50 所示。



▲ 图 7-47 单击第一个按钮后



▲ 图 7-48 单击第二个按钮后



▲ 图 7-49 单击第三个按钮后



▲ 图 7-50 单击第四个按钮后选中一项

7.6.2 实例编程实现

其实除了 ArrayAdapter 和 SimpleAdapter 与 ListView 可以关联外，还有一种比较常见的关联用法是使用 SimpleCursorAdapter 与 ListView 关联。当一个 ListView 需要和数据库当中的数据绑定时，SimpleCursorAdapter 的使用更为常见。由于现在还没有学习到对数据库的操作，所以，关于 SimpleCursorAdapter 和 ListView 的结合会放到讲解数据库的时候再进行说明。

1. 第一步

在 Eclipse 中打开 ex07_list 项目，操作步骤如下。

(1) 新建一个项目。依次单击 File → New → Android Project 项。

(2) 在新建项目的对话框中，选择 Create project from existing source 项。

(3) 单击浏览，找到 ex07_list 项目，然后单击确定。

程序目录结构如图 7-51 所示。

2. 第二步

打开 ActivityMain.java 文件，在这文件中的 Activity 和布局文件 main.xml 定义的布局进行关联。在 ActivityMain 中，从 main.xml 里边得到了 4 个 Button 按钮，并对 Button 设置了单击事件，当单击每个 Button 的时候，分别进入不同的 Activity。

在 AndroidManifest.xml 文件当中，我们也对应用当中的 5 个 Activity 进行了声明。其中声明 ActivityMain 为主 Activity，也是应用启动的 Activity，实现代码如下所示：

```
<activity Android:name=".ActivityMain"
    Android:label="Android 中的列表演示">
    <intent-filter>
    <action Android:name="Android.intent.action.MAIN" />
    <category
        Android:name="Android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

 代码解释：

上边定义了我们这个 Activity 的名字叫做 ActivityMain。并且从 intent-filter 中我们定义了里边的动作(action)是"Android.intent.action.MAIN"，分类是"Android.intent.category.LAUNCHER"，通过这两个信息说明了这个 Activity 是程序启动时候的第一个 Activity。

3. 第三步

在程序的主界面单击第一个按钮后进入到 ActivityList1 当中。在 Eclipse 当中双击打开 ActivityList1.java 文件。下边是 onCreate 方法里边的实现代码：

```
listView = new ListView(this);
listView.setAdapter(new ArrayAdapter<String>(this,Android.R.layout.simple_list_item_1,
    data));
setContentView(listView);
```

 代码解释：

- listView = new ListView(this)语句新建了一个 ListView，也就是新建了一个列表。和前边一样，在新建一个组件时会将一个 Context 传递给构造函数。
- listView.setAdapter(new ArrayAdapter<String>(this,Android.R.layout.simple_list_item_1, data))



▲ 图 7-51 程序目录结构

语句让 `listView` 和 `ArrayAdapter` 进行绑定。`ListView` 是一个列表，列表一般都需要显示数据。在上边的代码当中，`listView` 会把 `ArrayAdapter` 里边的数据显示出来。而 `ArrayAdapter` 是由 3 个参数进行构造的。第一个参数，依然是 `Context`，即上下文的引用；第二个参数为一个在 `R` 文件里边定义的 `Layout`，只不过这里用的是系统的 `R` 文件；第三个参数是一个数组，当然数组里边的每一项的类型没有限制。在上边的代码当中，第二个参数为 `Android.R.layout.simple_list_item_1`，这表明 `ListView` 里边的每一项都只有一个 `TextView`；第三个参数用的数字字符串数组。

- `setContentView(listView)` 语句表示将 `listView` 直接进行显示。

小知识

什么是 `ListAdapter`？

`ListAdapter` 继承于 `Adapter`，它是 `ListView` 和其里边数据的适配器。也就是要让一个 `ListView` 显示出来，需要 3 个东西。

- `ListView`（需要被显示的列表）。
- `Data`，和 `ListView` 绑定的数据，一般是一个 `Cursor` 或者一个字符串数组。
- `ListAdapter`，是 `data` 和 `ListView` 的桥梁，起一个适配作用。

在这一节的例子中，读者会学到它的两个直接子类 `ArrayAdapter` 和 `SimpleAdapter` 的应用。

小知识

什么是 `ArrayAdapter`？

`ArrayAdapter` 是 `ListAdapter` 的一个直接子类。如果直译这个名字，可以翻译为数组适配器。很显然，它的作用就是一个数组和 `ListView` 之间的桥梁。它可以将数组里边定义的数据一一对应的显示在 `ListView` 里边。通常由 `ArrayAdapter` 进行适配的 `ListView` 每一项通常只有一个 `TextView`，而 `TextView` 里边显示的内容就是数组里边的对象调用 `toString()` 方法后生成的字符串。

小知识

Android 系统默认的布局方式。

项目当中自己定义的布局方式可以通过 `R.layout.XX` 访问（`XX` 为资源的名字）。Android 系统提供了很多种默认的布局方式，他们通过 `Android.R.layout.XX` 来进行定义。对于 `ListView` 当中的某一项的布局方式，Android 系统也提供了很多种，我们经常用的有下面一些。

- `Android.R.layout.simple_list_item_1`，每一项只有一个 `TextView`。
- `Android.R.layout.simple_list_item_2` 每一项有两个 `TextView`。
- `Android.R.layout.simple_list_item_single_choice`，每一项有一个 `TextView`，但是这一项可以被选择。

4. 第四步

在上边演示了一个 `ListView` 和一个 `ArrayAdapter` 绑定的列表。但是这种绑定有一定的局限性。首先 `ListView` 里边的每一项只有一个 `TextView`，而且 `TextView` 里边的内容都是调用了数组里边每

一个对象的 `toString()` 方法生成的字符串。ActivityList2 中将会演示 SimpleAdapter 和 ListView 的结合。它们生成的列表有了很大的用户可定制性。可以对 ListView 里边的每一项进行个性化布局。在看具体的代码前，最好了解一下什么是 SimpleAdapter。

小知识

什么是 SimpleAdapter?

SimpleAdapter 也是 ListAdapter 的直接子类。通过 SimpleAdapter 可以让 ListView 当中的每一项里边的内容更加个性化。通常将 ListView 中某项的布局信息写在一个 XML 的布局文件当中。这个布局文件通过 `R.layout.XX` (XX 为文件的名称) 获得。

ArrayAdapter 的作用是数组和 ListView 间的桥梁，而 SimpleAdapter 的作用是 ArrayList 和 ListView 的桥梁。需要注意的是，这个 ArrayList 里边的每一项都是一个 `Map<String, ?>` 类型。ArrayList 当中的每一项 Map 对象都和 ListView 当中的一项进行数据绑定和一一对应。

看一下 SimpleAdapter 的构造函数，就会对 SimpleAdapter 更加了解，函数源代码如下所示：

```
public SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int resource,
String[] from, int[] to)
```

函数解释：

- Context，这个不用解释了，在很多类的构造函数里边都会将这个上下文的应用传递进去。
- Data，基于 Map 的 List。Data 里边的每一项都和 ListView 里边的每一项对应。Data 里边的每一项是一个 Map 类型，这个 Map 类里边包含了 ListView 每一行需要的数据。比较常用的用法是：
`data = new ArrayList<Map<String, Object>>();`
- Resoure，这个 Resoure 就是一个 layout，这个 layout 最起码要包含在 to 中出现的那些 View。一般用系统提供的就可以了，当然也可以自己定义。系统提供的有 `Android.R.layout.simple_list_item_single_choice`、`Android.R.layout.two_line_list_item` 等。
- From，这是一个名字的数组，每一个名字是为了在 ArrayList 中的每一个 item 中索引 `Map<String, Object>` 的 Object 用的。
- To，里边是一个 TextView 的数组。这些 TextView 是以 id 的形式来表示的，如 `Android.R.id.text1`，这个 text1 在 layout 当中是可以索引到的。需要注意的是这些 View 必须是 TextView。

接下来打开 ActivityList2 文件，看一下里边的具体实现代码：

```
private void PrepareData() {
    data = new ArrayList<Map<String, Object>>();
    Map<String, Object> item;
    item = new HashMap<String, Object>();
    item.put("姓名", "张三小朋友");
    item.put("性别", "男");
    data.add(item);
    item = new HashMap<String, Object>();
    item.put("姓名", "王五同学");
    item.put("性别", "男");
    data.add(item);
    item = new HashMap<String, Object>();
    item.put("姓名", "小李师傅");
```

```

item.put("性别", "女");
data.add(item);
}

```

代码解释:

方法 `PrepareData()` 负责提前准备好需要在 `ListView` 里边要显示的数据。我们构造了一个 `ArrayList<Map<String, Object>>()`, 这是一个列表, 列表中的每个元素是一个 `Map` 集合。

接着看一下 `onCreate()` 里边的实现代码:

```

super.onCreate(savedInstanceState);
PrepareData();
listView = new ListView(this);
SimpleAdapter adapter = new SimpleAdapter(this, data,
R.layout.listitem, new String[] { "姓名", "性别" }, new int[] {
R.id.mview1, R.id.mview2 });
listView.setAdapter(adapter);
setContentView(listView);

```

代码解释:

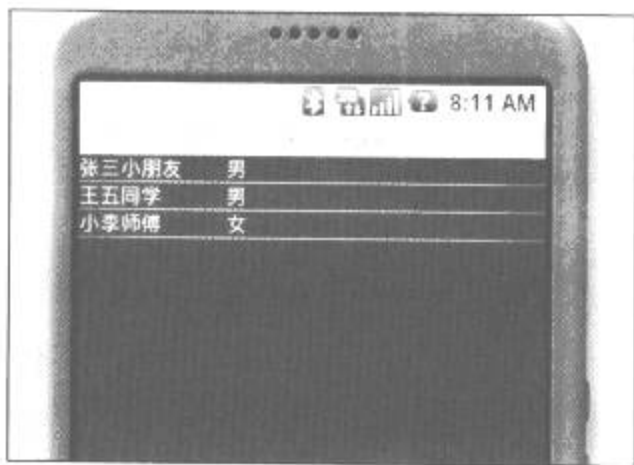
- `PrepareData()` 语句准备一个 `Map` 类型的 `ArrayList` 对象。
- `listView = new ListView(this)` 语句新建一个 `ListView`。
- `SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.list_item, new String[] { "姓名", "性别" }, new int[] { R.id.mview1, R.id.mview2 })`, 这条语句负责生成一个 `SimpleAdapter` 的实例。现在来看一下其中的参数都是什么。
 - `this`, 这个就是当前上下文的 `Context` 引用。
 - `R.layout.list_item` 关联在 `list_item.xml` 文件当中定义的 `Layout`, 这个 `Layout` 规定 `ListView` 当中每一项的布局。
 - `String[] { "姓名", "性别" }`, 这是一个 `String` 的数组, 里边每一个字符串用于访问得到 `Map` 对象当中的值。例如, 在这个例子当中, 在 `PrepareData()` 函数当中执行 `item.put("姓名", "张三")`, 那么在 `ListView` 当中就可以通过 `item.get("姓名")` 将“张三这个值取出来”, 并且在 `ListView` 中的一项进行显示。

- `int[] { R.id.mview1, R.id.mview2 }`, 里边的值就是在 `list_item.xml` 里边定义的两个 `TextView` 的 `Id`。

下面稍微总结一下, `ListView` 的每一项都包含有两个 `TextView`, 并且系统将 `data[i].get("姓名")` 的值显示在 `id` 为 `mview1` 的 `TextView` 上。将 `data[i].get("性别")` 的值显示在 `id` 为 `mview2` 的 `TextView` 上, 实现效果如图 7-52 所示。

小实验

在代码 `SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.list_item, new String[] { "姓名", "性别" }, new int[] { R.id.mview1, R.id.mview2 })` 中, 除了可以使用自己实现的个性化布局 `R.layout.list_item` 外, 还可以直接采用 `Android` 系统给我们提供的默认布局, 如 `Android.R.layout.simple_list_item_2`。在程序中将布局代码进行修改, 然后运行程序, 看一下效果是怎样的, 如图 7-53 所示。



▲ 图 7-52 代码执行后的界面



▲ 图 7-53 使用 Android.R.layout.simple_list_item_2 进行布局

5. 第五步

很多时候的应用并不仅是显示列表就足够了，还需要和列表进行一些交互，例如，单击列表当中的某一项会触发一些动作。而且和列表的交互是经常的，有时候也是必须的。

在第一节当中已经知道如何将一个单击事件的处理和一个 **Button** 关联起来。最经常的办法就是采用监听器的方法。**Button** 采用的是注册 **OnClickListener** 的方式，对应的处理 **List** 的单击事件是把一个 **OnItemClickListener** 注册到 **ListView** 当中。

打开 **ActivityList2** 文件，看 **onCreate()** 里边的部分实现代码：

```
OnItemClickListener listener = new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position,
        long id) {
        setTitle(parent.getItemAtPosition(position).toString());
    }
};
listView.setOnItemClickListener(listener);
```

🔍 代码解释：

- **onItemClick** 是回调函数，即当用户单击了 **ListView** 当中的某一项后，系统会自动执行这个函数。也就是说这个函数不是由用户自己主动调用的，而是由系统反向调用的。这个函数有 4 个参数，第一个参数为被单击的 **ListView**；第二个参数是一个 **View**，代表用户选中的那一项；第三个参数为选中的 **View** 在 **ListView** 当中的位置；第四个参数为被选中那一行的 **id**。

- **getItemAtPosition()** 非常有用并且也经常用，它可以通过 **position** 获得和这一行绑定的数据。在这个例子当中，只是简单地获得了数据，并且调用 **toString()** 方法把数据变成字符串显示在界面的 **title** 区域。

- **listView.setOnItemClickListener(listener)** 语句将监听器注册到 **ListView** 里边。

6. 第六步

在很多情况下每一个 **Activity** 里边都是只有一个 **ListView**，而且这个 **ListView** 都会占满一屏。所以 **Android** 提供了一个类叫做 **ListActivity**。一看名字就可以明白这个类的意义，它既是一个 **Activity** 又有 **ListView** 的特征。

每一个 **ListActivity** 都是和一个布局文件相关联，而且这个布局文件基本上都是按照下列模式固定的。打开 **list3.xml** 文件，具体代码如下所示：


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ListView android:id="@id/android:list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>
    <TextView android:id="@id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="对不起，没有数据显示"/>
</LinearLayout>
```

代码解释：

在这个布局文件当中，必须定义一个 ListView，并且这个 ListView 的 id 是 “@id/android:list”，这个名字是 Android 系统预设的，也就是说在 ListActivity 当中 Android 系统能通过这个预设的 id 得到这个 ListView。

另外一个需要定义但并非必须的是 id 为 “@id/android:empty” 的 TextView。虽然不是必须要定义，但是我们强烈推荐在程序当中来定义这个元素。当 ListView 没有数据可显示时，这个 TextView 可以友好地展示出对用户的提示信息。

打开 ActivityList3 看一下 onCreate() 函数里边的代码，下边的代码和 ActivityList1 当中的类似，在这里就不重复解释了，代码如下所示：

```
setContentView(R.layout.list3);
setListAdapter(new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, data))
```

接下来看一下 onItemClick 方法，实现代码如下所示：

```
protected void onItemClick(ListView listView, View v, int position, long id) {
    super.onItemClick(listView, v, position, id);
    setTitle(listView.getItemAtPosition(position).toString());
}
```

代码解释：

在 ListActivity 里边不用注册监听器，而是直接重写 onItemClick 就可以对单击 ListView 的事件进行处理。onItemClick() 是一个回调函数，由 Android 函数负责调用。

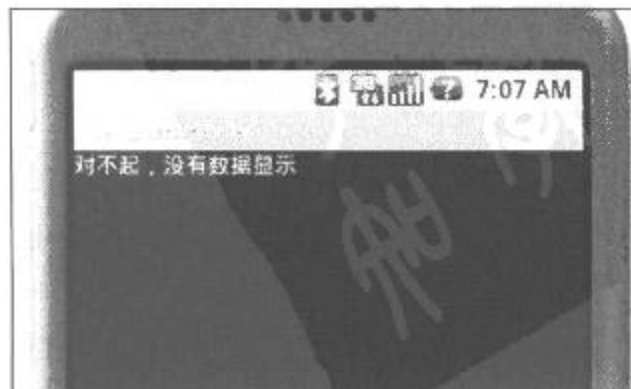
小实验

在 ActivityList3 中我们将对 data 赋值的代码改为：private String[] data = {}，运行修改的程序，单击 button3 就会出现下边的界面，如图 7-54 所示。

7. 第七步

对于 ListView 来说，用户的事件不光只有单击，也就是说不只 onItemClickListener 监听器可以注册到 ListView 里边，比较常见还有 onItemSelectedListener 监听器。当 ListView 里边的某一项被选择（selected）到的时候触发 onItemSelectedListener 监听器里边的方法。

打开 ActivityList4 里边的代码多出了对 onItemClick



▲ 图 7-54 没有数据显示的界面

SelectedListener 监听器的处理外，基本上和 ActivityList1 的代码一致，具体实现代码如下所示：

```
OnItemSelectedListener itemSelectedListener = new OnItemSelectedListener() {

    @Override
    public void onItemSelected(AdapterView<?> parent, View arg1,
        int position, long arg3) {
        setTitle("您选中的软件是: "+parent.getItemAtPosition(position).toString());
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        setTitle("");
    }
};

listView.setOnItemSelectedListener(listener);
```

 代码解释：

- onItemSelected()函数里边的参数和先前回调函数 onItemClick 的一致。即，第一个参数为被单击的 ListView；第二个参数是 View，代表用户选中的那一项；第三个参数为选中的 View 在 ListView 当中的位置；第四个参数为被选中那一行的 id。

- 当 ListView 中的某一项被选中时，通过 getItemAtPosition()得到和这一项绑定的数据对象，并调用数据对象的 toString()方法生成字符串，然后将这个字符串显示在手机界面的 title 区域。

运行程序，单击第四个按钮后进入 ActivityList4，然后使用键盘的上下键来移动选择 ListView，这个时候会看到 title 区域的值在一直的变化。

7.7 友好地互动交流——对话框 (Dialog) 介绍与实例

对话框虽然在程序中不是必备的，但是用好对话框能对我们编写的增色不少。采用对话框可以大大地增强应用的友好性。在这个用户体验至上的时代，好的界面就是成功的一半。所以，要学好对话框的应用，并在适当的地方灵活运用。比较常用的场景是，用户登录、网络正在下载、下载成功或失败的提示。当然还有很多场景可以用到。如来短信了、电池没电了等，一般都需要系统弹出对话框。

运行应用后，在主界面会显示几个 Button，单击每一个 Button 都会呈现出来不同的对话框。通过这个例子可以学到：

- 什么是 Dialog；
- 什么是 AlertDialog；
- 什么是 AlertDialog.Builder；
- 通过 AlertDialog.Builder 构造一个 AlertDialog；
- 什么是 ProgressDialog。

程序操作演示和实现讲解。

1. 第一步

在 Eclipse 中打开 ex07_dialog 项目，具体步骤如下。

(1) 新建一个项目。依次单击 File→New→Android Project 项。

(2) 在新建项目的对话框中，选择 Create project from existing source 项。

(3) 单击浏览，找到 ex07_dialog 项目，然后单击确定。

程序的目录结构如图 7-55 所示。



▲ 图 7-55 程序的目录结构

2. 第二步

首先运行这个实例程序，运行后在手机屏幕里边会出现 4 个按钮。关于主界面的布局在 alert_dialog.xml 文件里，由于界面简单，在这里不对 alert_dialog.xml 文件里边的代码进行讲解，显示的效果如图 7-56 所示。

3. 第三步

单击第一个按钮进入下边的界面，如图 7-57 所示。



▲ 图 7-56 应用主界面



▲ 图 7-57 单击第一个按钮进入的界面

小知识

什么是 Dialog?

Dialog 类，是一切对话框的基类。需要注意的是，Dialog 类虽然可以在界面上显示，但是并非继承于习惯用的 View 类，而是直接从 java.lang.Object 开始构造出的。类似于 Activity，Dialog 也是有生命周期的，它的生命周期由 Activity 来维护。Activity 负责生成、保存、恢复它。在生命周期的每个阶段都有一些回调函数供系统反向调用。

在 Activity 当中用户可以主动调用的函数为：

- `showDialog(int id)`, 负责显示标识为 `id` 的 `Dialog`。这个函数如果调用后, 系统将反向调用 `Dialog` 的回调函数 `onCreateDialog(int id)`。

- `dismissDialog(int id)`, 使标识为 `id` 的 `Dialog` 在界面当中消失。

`Dialog` 有两个比较常见的回调函数, `onCreateDialog(int id)`和 `onPrepareDialog(int id, Dialog dialog)`函数。当在 `Activity` 当中调用了 `showDialog(int id)`后, 如果这个 `Dialog` 是第一次生成, 系统将反向调用 `Dialog` 的回调函数 `onCreateDialog(int id)`, 然后再调用 `onPrepareDialog(int id, Dialog dialog)`; 如果这个 `Dialog` 已经生成, 只不过还没有显示出来, 那么将不会回调 `onCreateDialog(int id)`, 而是直接回调 `onPrepareDialog(int id, Dialog dialog)`方法。

`onPrepareDialog(int id, Dialog dialog)`方法提供了这样一套机制, 即当 `Dialog` 生成但是没有显示出来的时候, 使得有机会在显示前对 `Dialog` 做一些修改, 如对 `Dialog` 标题进行修改等。

了解了 `Dialog` 的一些基本知识后, 来看 `ActivityMain` 文件当中 `onCreate()`方法里边的实现代码:

```
Button button1 = (Button) findViewById(R.id.button1);
button1.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        showDialog(DIALOG1);
    }
});
```

 代码解释:

- `findViewById` 方法通过组件的 `id` 返回这个组件的引用。
- `setOnClickListener` 方法为 `button1` 设置了一个单击监听器。
- `onClick()`为单击 `Button` 后的回调函数。
- `showDialog()`为 `Activity` 里边的函数。负责将 `id` 为 `DIALOG1` 的 `Dialog` 显示出来。

执行 `showDialog(DIALOG1)`代码后, 系统执行回调函数 `onCreateDialog()`, 实现代码如下所示:

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case DIALOG1:
            return buildDialog1(ActivityMain.this);

        case DIALOG2:
            return buildDialog2(ActivityMain.this);

        case DIALOG3:
            return buildDialog3(ActivityMain.this);
        case DIALOG4:
            return buildDialog4(ActivityMain.this);
    }
    return null;
}
```

代码解释:

- 针对不同的 Dialog 的 Id, 生成不同的 Dialog。
- buildDialog1 函数生成第一个要显示的 Dialog。

4. 第四步

接下来看一下 buildDialog1()函数, 其实现代码如下所示:

```
private Dialog buildDialog1(Context context) {
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setIcon(R.drawable.alert_dialog_icon);
    builder.setTitle(R.string.alert_dialog_two_buttons_title);
    builder.setPositiveButton(R.string.alert_dialog_ok,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {

                setTitle("单击对话框上的确定按钮");
            }
        });
    builder.setNegativeButton(R.string.alert_dialog_cancel,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {

                setTitle("单击了对话框上的取消按钮");
            }
        });
    return builder.create();
}
```

代码解释:

- AlertDialog.Builder builder = new AlertDialog.Builder(context)语句首先生成一个 AlertDialog.Builder 的对象, 这样就可以开始构造 AlertDialog。
- builder.setIcon(R.drawable.alert_dialog_icon)语句给 AlertDialog 预设置一个图片。这里预设的是 alert_dialog_icon.png 这张图片。
- builder.setTitle(R.string.alert_dialog_two_buttons_title)语句给 AlertDialog 预设一个标题, 标题的字符串内容在 value 目录的 string.xml 文件里边定义。
- setPositiveButton()这个方法设置确定按钮的一些属性。第一个参数为按钮上显示出来的内容; 第二个参数为 DialogInterface.OnClickListener() 监听器对象, 这个监听器和前边学习的 Button 的单击监听器类似。
- onClick()方法为监听器中的回调方法, 即当单击 Dialog 的按钮时, 系统回调这个方法。一般将对话框处理的逻辑写到回调函数里边。在这个回调函数里边我们通过 setTitle 方法, 将标题栏的内容进行了重新设置。
- setNegativeButton()方法和 setPositiveButton()方法对应, 用于设置、取消按钮的一些属性。
- 预设好所有关于 Dialog 的属性后, 执行 builder.create()后生成一个配置好的 Dialog。

小知识

什么是 AlertDialog? 什么是 AlertDialog.Builder?

AlertDialog 是 Dialog 的一个直接子类, AlertDialog 也是 Android 系统当中最常用的对话框之一。在图 7-57 中显示出来的对话框就是一个 AlertDialog。一个 AlertDialog 可以有两个 Button 或者 3 个 Button, 可以对一个 AlertDialog 设置 title 和 message。不能直接通过 AlertDialog 的构造函数来生成一个 AlertDialog。一般生成 AlertDialog 的时候都是通过它的一个内部静态类 AlertDialog.Builder 来构造的。

5. 第五步

当程序执行完 onCreateDialog()方法后, 接着会执行 onPrepareDialog 方法。也就是在小知识里边总结的: 不管某 Dialog 是不是第一次生成, 在正式显示之前都会执行 onPrepareDialog()函数, 这样可以有机会对这个 Dialog 进行修改。

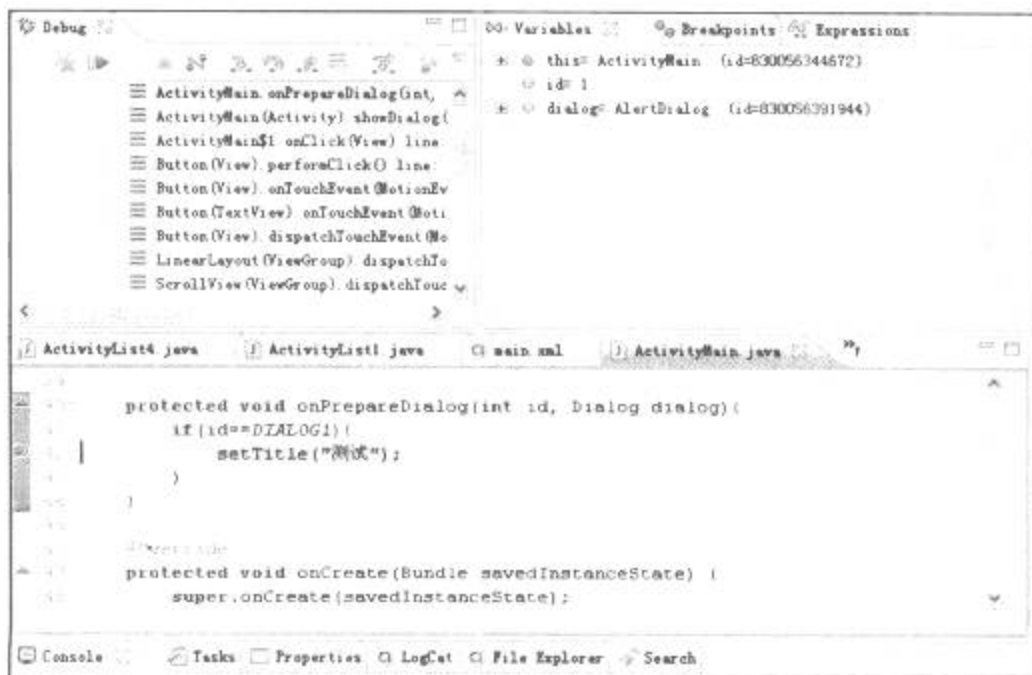
我们可以在 setTitle("测试")这一行设置断点来调试一下, 就会看到结果了, 如图 7-58 所示。

具体代码如下:

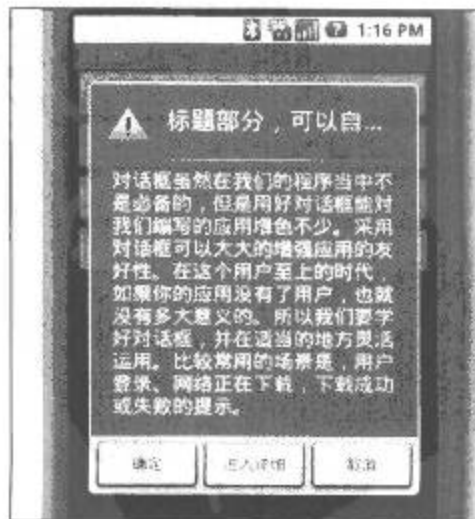
```
protected void onPrepareDialog(int id, Dialog dialog) {
    if (id == DIALOG1) {
        setTitle("测试");
    }
}
```

6. 第六步

在程序主页面中单击第二个 Button, 出现界面如图 7-59 所示。



▲ 图 7-58 执行到断点位置



▲ 图 7-59 单击第二个按钮进入的界面

在这里只需看一下这个 Dialog 是怎么生成的，实现代码如下所示：

```
private Dialog buildDialog2(Context context) {
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setIcon(R.drawable.alert_dialog_icon);
    builder.setTitle(R.string.alert_dialog_two_buttons_msg);
    builder.setMessage(R.string.alert_dialog_two_buttons2_msg);
    builder.setPositiveButton(R.string.alert_dialog_ok,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {

                setTitle("点击了对话框上的确定按钮");
            }
        });
    builder.setNeutralButton(R.string.alert_dialog_something,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {

                setTitle("点击了对话框上的进入详细按钮");
            }
        });
    builder.setNegativeButton(R.string.alert_dialog_cancel,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {

                setTitle("点击了对话框上的取消按钮");
            }
        });
    return builder.create();
}
```

代码解释：

buildDialog2()方法里边的代码很多和 buildDialog1()里边的比较类似。下边具体讲一下不太一样的地方。

- builder.setMessage(R.string.alert_dialog_two_buttons2_msg)，一般一个 Dialog 都是有一个标题（title），然后标题（title）下边有一些信息（message）。
- setNeutralButton()方法负责设置中间按钮的一些属性，具体设置方法和 buildDialog1()里边介绍的一样。

7. 第七步

返回到程序的主界面，单击第三个按钮后出现如图 7-60 所示的界面。

来看一下代码是怎么实现的：

```
private Dialog buildDialog3(Context context) {
    LayoutInflater inflater = LayoutInflater.from(this);
    final View textEntryView = inflater.inflate(
        R.layout.alert_dialog_text_entry, null);
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setIcon(R.drawable.alert_dialog_icon);
```



▲ 图 7-60 单击第三个按钮进入的界面

```

builder.setTitle(R.string.alert_dialog_text_entry);
builder.setView(textEntryView);
builder.setPositiveButton(R.string.alert_dialog_ok,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            setTitle("点击了对话框上的确定按钮");
        }
    });
builder.setNegativeButton(R.string.alert_dialog_cancel,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            setTitle("点击了对话框上的取消按钮");
        }
    });
return builder.create();
}

```

代码解释:

此方法的实现和前边介绍的两个方法的实现比较类似，但是在这个 Dialog 里边，没有设置 Dialog 的 message，而是直接给 Dialog 设置了一个个性化的 View。

- LayoutInflater 类在前边已经介绍过了，通过这个类的 inflate 方法，可以将一个 XML 的布局变成一个 View 实例。
- builder.setView(textEntryView) 语句，此方法是这个 Dialog 的精髓。通过这种方法可以将实现好的个性化的 View 放置到 Dialog 里边去。这里的 textEntryView 和 alert_dialog_text_entry.xml 这个文件定义的布局相关联。

8. 第八步

打开 alert_dialog_text_entry.xml 这个文件，看一下布局的代码，具体如下所示：

```

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

    <TextView
        android:id="@+id/username_view"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:text="用户名"

        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/username_edit"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"

```

```

        Android:capitalize="none"
        Android:textAppearance="?Android:attr/textAppearanceMedium" />

<TextView
    Android:id="@+id/password_view"
    Android:layout_height="wrap_content"
    Android:layout_width="wrap_content"
    Android:layout_marginLeft="20dip"
    Android:layout_marginRight="20dip"
    Android:text="密码"

    Android:textAppearance="?Android:attr/textAppearanceMedium" />

<EditText
    Android:id="@+id/password_edit"
    Android:layout_height="wrap_content"
    Android:layout_width="fill_parent"
    Android:layout_marginLeft="20dip"
    Android:layout_marginRight="20dip"
    Android:capitalize="none"
    Android:password="true"
    Android:textAppearance="?Android:attr/textAppearanceMedium" />

</LinearLayout>

```

代码解释：

这里我们会学习一些新的东西。

- `Android:textAppearance="?Android:attr/textAppearanceMedium"`，这条语句负责设置 `TextView` 里边文字的字体，这个字体是系统里边的。
- `EditText` 是我们新学到的一个组件，它是文本输入框。
- `Android:capitalize="none"`，`Android:capitalize` 属性规定输入的大小写，如果为 `none`，输入的首字母是小写的。
- `Android:password="true"`，说明此输入框为密码输入框。

小实验

对上边 `alert_dialog_text_entry.xml` 文件里边的代码稍作修改。去掉第一个 `TextView` 里边的 `Android:textAppearance="?Android:attr/textAppearanceMedium"`，现在看看字体有什么变化？将 `Android:capitalize="none"` 改成 `Android:capitalize="words"`，当在用户名下边的输入框中输入英文的时候，显示又如何。我们再将第二个 `EditText` 里边的 `Android:password="true"` 改成 `Android:password="false"`，当输入密码的时候显示又如何。只有对程序多做改动，多做变化，才能更好地理解 `Android`。

9. 第九步

返回到程序的主界面，单击最后一个 `Button`，看一下要学习的另外一个 `Dialog:ProgressDialog` 运行效果，如图 7-61 所示。



▲ 图 7-61 单击第四个按钮进入的界面

小知识

什么是 ProgressDialog?

顾名思义, 这个 Dialog 负责给用户显示进度的相关情况, 它是 AlertDialog 的一个子类。在这个例子当中, 我们是实现默认的进度显示。当然可以配置自己的进度条。看一下 ActivityMain 当中的 buildDialog4() 函数, 具体实现代码如下所示:

```
private Dialog buildDialog4(Context context) {
    ProgressDialog dialog = new ProgressDialog(context);
    dialog.setTitle("正在下载歌曲");
    dialog.setMessage("请稍候.....");
    return dialog;
}
```

代码解释:

- 和 AlertDialog 不一样的是, ProgressDialog 不需要通过 Builder 这个内部静态类进行构造, 而是直接使用构造函数进行构造。

- setTitle() 设置标题。
- setMessage() 设置内容。

在这一节当中我们对 Dialog 进行了讨论, 其实在 Android 当中还提供了很多 Dialog, 如 CharacterPickerDialog、TimePickerDialog、DatePickerDialog 等。但是最常用的还是 AlertDialog 和 ProgressDialog, 至于其他的 Dialog, 读者可以参考一下 Android 的 API 文档和 Google 提供的一个 APIdemo 的例子。

7.8 温馨提醒——Toast 和 Notification 应用

前边我们学习的对话框 Dialog 其实已经起到提醒用户的作用, 但是 Android 系统提供了另外一套更加友好的, 轻量级的机制来提醒用户, 这种系统不会打断用户的当前操作, 真可谓非常巧妙。通过下边的例子, 可以学习到:

- 如何使用 Toast 更加友好地提示用户;

- 如何使用 Notification 友好地提示用户。

7.8.1 实例操作演示

- (1) 程序运行主界面如图 7-62 所示。
- (2) 单击第一个 button 进入的界面，如图 7-63 所示。
- (3) 单击“晴空万里”按钮，显示的界面如图 7-64 所示。
- (4) 单击“阴云密布”按钮，显示的界面如图 7-65 所示。
- (5) 单击“大雨连绵”按钮，显示的界面如图 7-66 所示。
- (6) 单击状态栏，拖出显示所有 Notification 的详细信息，如图 7-67 所示。

如图 7-67 所示。

(7) 单击主页面第二个按钮后进入的页面，显示效果如图 7-68 所示。

(8) 单击“短时间显示 Toast”按钮后，显示如图 7-69 所示。

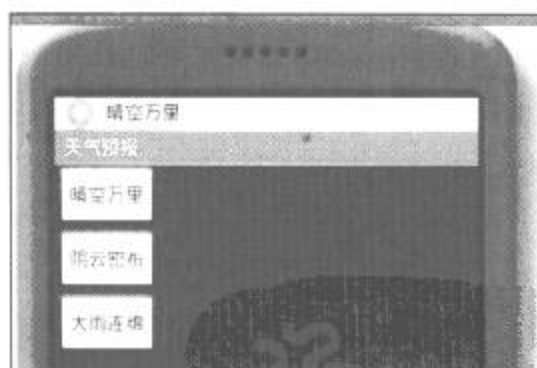
(9) 单击“长时间显示 Toast”按钮后，显示如图 7-70 所示界面。



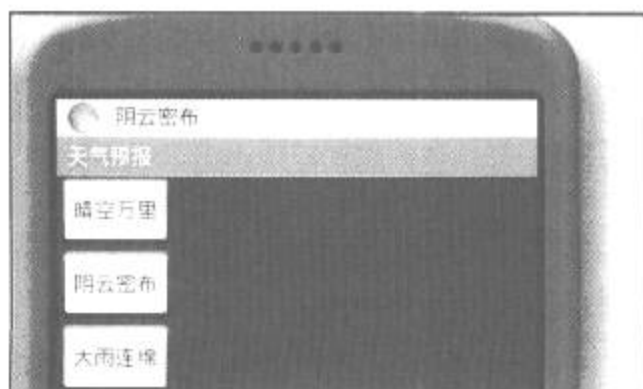
▲ 图 7-62 程序主界面



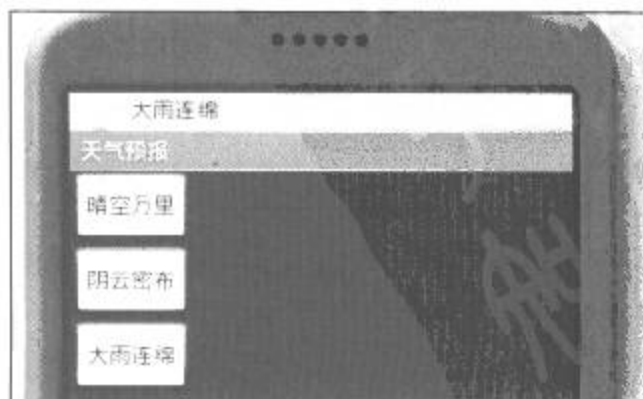
▲ 图 7-63 单击第一个 Button 进入界面



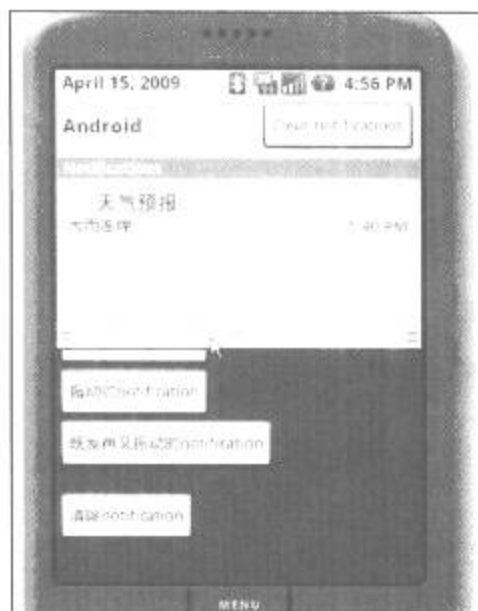
▲ 图 7-64 单击“晴空万里”显示界面



▲ 图 7-65 单击“阴云密布”后界面



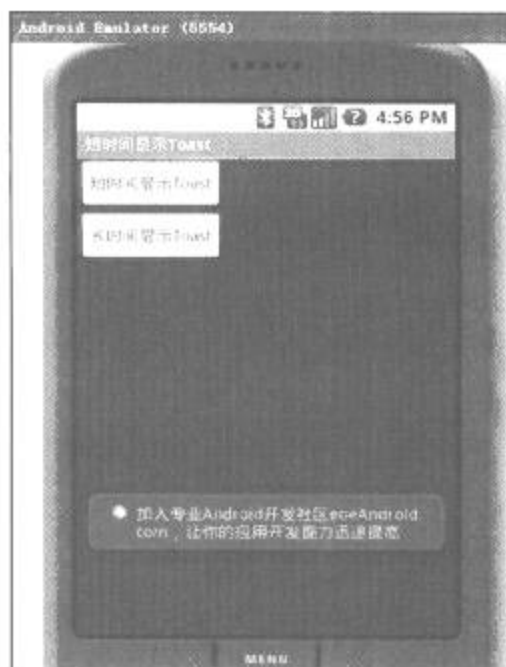
▲ 图 7-66 单击“大雨连绵”后界面



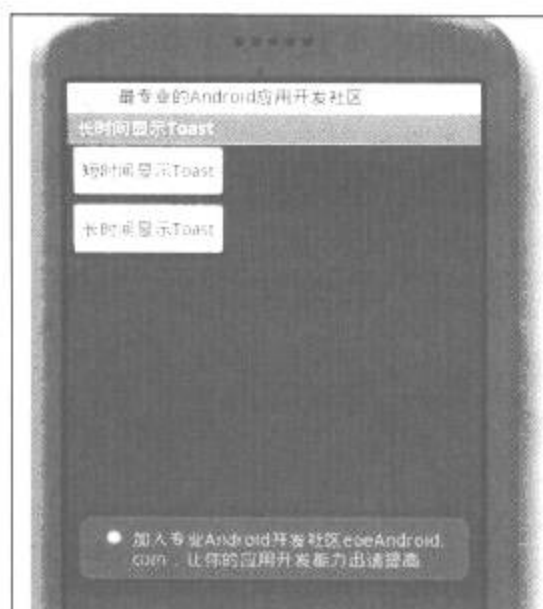
▲ 图 7-67 显示所有 Notification 的详细信息界面



▲ 图 7-68 单击主页面第二个按钮后进入的页面界面



▲ 图 7-69 单击“短时间显示 Toast”后界面



▲ 图 7-70 单击“短时间显示 Toast”后界面

7.8.2 实例编程实现

1. 第一步

在 Eclipse 中打开 ex07_toast_and_notification 项目，具体步骤如下。

- (1) 新建一个项目。依次单击 File→New→Android Project 项。
- (2) 在新建项目的对话框中，选择 Create project from existing source 项。
- (3) 单击浏览，找到 ex07_toast_and_notification 项目，然后单击确定。

程序的目录结构如图 7-71 所示。



▲ 图 7-71 程序的目录结构

2. 第二步

首先来看一下 ActivityMain 文件中的代码，具体如下所示：

```
package com.eoeAndroid.toast_and_notification;

import Android.app.Activity;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;

public class ActivityMain extends Activity {

    OnClickListener listener1 = null;
    OnClickListener listener2 = null;

    Button button1;
    Button button2;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        listener1 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("学习 Notification");
                Intent intent = new Intent(ActivityMain.this,
                    ActivityMainNotification.class);
                startActivity(intent);
            }
        };
        listener2 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("学习 Toast");
                Intent intent = new Intent(ActivityMain.this,
                    ActivityToast.class);
                startActivity(intent);
            }
        };
        setContentView(R.layout.main);
        button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(listener1);
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(listener2);
    }
}
```

 代码解释：

- ActivityMain 显示了两个 Button，并且对两个 Button 绑定了单击监听器(OnClickListener)，当单击这两个 Button 的时候，会跳转到新的 Activity。

3. 第三步

单击第一个 Button 后，程序跳转到 ActivityMainNotification 里边，具体看一下实现代码：

```

package com.eoeAndroid.toast_and_notification;

import Android.app.Activity;
import Android.app.Notification;
import Android.app.NotificationManager;
import Android.app.PendingIntent;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.View;
import Android.widget.Button;

public class ActivityMainNotification extends Activity {
    private static int NOTIFICATIONS_ID = R.layout.activity_notification;
    private NotificationManager mNotificationManager;

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

        Button button;

        mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

        button = (Button) findViewById(R.id.sun_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("晴空万里", "天气预报", "晴空万里", R.drawable.sun);
            }
        });

        button = (Button) findViewById(R.id.cloudy_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("阴云密布", "天气预报", "阴云密布", R.drawable.cloudy);
            }
        });

        button = (Button) findViewById(R.id.rain_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("大雨连绵", "天气预报", "大雨连绵", R.drawable.rain);
            }
        });

        button = (Button) findViewById(R.id.defaultSound);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_SOUND);
            }
        });

        button = (Button) findViewById(R.id.defaultVibrate);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_VIBRATE);
            }
        });
    }
}

```

```

button = (Button) findViewById(R.id.defaultAll);
button.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        setDefault(Notification.DEFAULT_ALL);
    }
});

button = (Button) findViewById(R.id.clear);
button.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        mNotificationManager.cancel(NOTIFICATIONS_ID);
    }
});

}

private void setWeather(String tickerText, String title, String content,
    int drawable) {

    Notification notification = new Notification(drawable, tickerText,
        System.currentTimeMillis());

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, ActivityMain.class), 0);

    notification.setLatestEventInfo(this, title, content, contentIntent);

    mNotificationManager.notify(NOTIFICATIONS_ID, notification);
}

private void setDefault(int defaults) {

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, ActivityMain.class), 0);

    String title = "天气预报";
    String content = "晴空万里";

    final Notification notification = new Notification(R.drawable.sun,
        content, System.currentTimeMillis());

    notification.setLatestEventInfo(this, title, content, contentIntent);

    notification.defaults = defaults;

    mNotificationManager.notify(NOTIFICATIONS_ID, notification);
}
}

```

❏ 代码解释：

- 所有的 Notification 都是由 NotificationManager 来管理，所以，第一步应该得到一个 NotificationManager 的实例，以便管理在这个 Activity 中的 Notification：`mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);`
- `System.currentTimeMillis()` 函数得到系统的当前时间。
- `setWeather()` 函数是 `ActivityMainNotification` 里边比较重要的函数之一。在这个函数里边，实例化了一个 Notification，并且将这个 Notification 显示出来。

- `Notification notification = new Notification(drawable, tickerText, System.currentTimeMillis());`
- 第一个参数为要显示的图片 id。
- 第二个参数为显示的文本文字。
- 第三个参数为 `Notification` 显示的时间，一般是立即显示，那么时间就是 `System.currentTimeMillis()`。

- 构造好一个 `Notification` 后就可以通过 `NotificationManager` 显示出来，需要注意的是，每一个显示的 `Notification` 都有一个惟一的 ID，这个 ID 是应用开发者来制指定的，例如，在这个例子当中，ID 为 `NOTIFICATIONS_ID`。显示 `Notification` 的具体实现代码为：`mNotificationManager.notify(NOTIFICATIONS_ID, notification);`

- 在显示一个 `Notification` 前还有一项工作也是必不可少的，那就是通过方法 `setLatestEventInfo(Context context, CharSequence contentTitle, CharSequence contentText, PendingIntent contentIntent)` 来设置两方面的内容：一方面如图 7-67 所示，当展开 `Notification` 列表的时候，如何呈现 `Notification`；另外一方面设置当单击 `Notification` 时候，如何处理单击。

- `setDefault()` 函数是 `ActivityMainNotification` 里边另外一个重要的函数。在这个函数里边仍然是初始化了一个 `Notification`，但是这里用到了另外一个技巧，那就是给 `Notification` 设置缺省的表现形式，如 `notification.defaults = defaults`。在此程序当中，用到了以下几种基本的表现形式。

- `Notification.DEFAULT_VIBRATE`，表示当前的这个 `Notification` 显示出来的时候，手机会伴随着振动。

- `Notification.DEFAULT_SOUND`，表示当前的这个 `Notification` 显示出来的时候，手机会伴随着音乐。

- `Notification.DEFAULT_ALL`，表示当前的这个 `Notification` 显示出来的时候，手机同时会伴随着振动和音乐。

4. 第四步

当在应用的主界面单击“学习 Toast”按钮后，程序跳转到 `ActivityToast` 中，具体看一下代码是怎么实现的：

```
package com.eoeAndroid.toast_and_notification;

import Android.app.Activity;
import Android.app.Notification;
import Android.app.NotificationManager;
import Android.app.PendingIntent;
import Android.content.Context;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.LayoutInflater;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;
import Android.widget.TextView;
import Android.widget.Toast;

public class ActivityToast extends Activity {
```

```

OnClickListener listener1 = null;
OnClickListener listener2 = null;
Button button1;
Button button2;
private static int NOTIFICATIONS_ID = R.layout.activity_toast;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    listener1 = new OnClickListener() {
        public void onClick(View v) {
            setTitle("短时间显示 Toast");
            showToast	Toast.LENGTH_SHORT);
        }
    };
    listener2 = new OnClickListener() {
        public void onClick(View v) {
            setTitle("长时间显示 Toast");
            showToast	Toast.LENGTH_LONG);
            showNotification();
        }
    };
    setContentView(R.layout.activity_toast);
    button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(listener1);
    button2 = (Button) findViewById(R.id.button2);
    button2.setOnClickListener(listener2);
}

protected void showToast(int type) {

    View view = inflateView(R.layout.toast);

    TextView tv = (TextView) view.findViewById(R.id.content);
    tv.setText("加入专业 Android 开发社区 eoeAndroid.com, 让你的应用开发能力迅速提高");

    Toast toast = new Toast(this);
    toast.setView(view);
    toast.setDuration(type);
    toast.show();
}

private View inflateView(int resource) {
    LayoutInflater vi = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    return vi.inflate(resource, null);
}

protected void showNotification() {

    NotificationManager notificationManager = (NotificationManager) getSystemService(
        NOTIFICATION_SERVICE);

    CharSequence title = "专业的 Android 应用开发社区";
    CharSequence contents = "eoeAndroid.com";

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, ActivityMain.class), 0);

    Notification notification = new Notification(R.drawable.default_icon,

```

```

title, System.currentTimeMillis());

notification.setLatestEventInfo(this, title, contents, contentIntent);

// 100ms 延迟后, 振动 250ms, 停止 100ms 后振动 500ms
notification.vibrate = new long[] { 100, 250, 100, 500 };

notificationManager.notify(NOTIFICATIONS_ID, notification);
}
}

```

代码解释:

Toast 虽然也是提醒用户的一种方式,但是在实现上和 Notification 是不一样的,Notification 需要被 NotificationManager 所管理。而 Toast 是作为 Android 中的 Widget 存在的。从上述代码里边的 showToast 方法中可以看下怎么样使用 Toast 的,具体实现如下。

- 首先需要实例化一个 Toast,通过代码 Toast toast = new Toast(this) 来实现。
- 每一个 Toast 的实例都可以和一个 View 相关,如 toast.setView(view)。
- Toast 的显示长度有长有短,如果能让 Toast 显示的时间长一点,可以通过设置 toast.setDuration Toast.LENGTH_LONG)来实现;如果能让 Toast 显示的时间短一点的话,可以通过 toast.setDuration Toast.LENGTH_SHORT)来实现。

- 设置好这些属性后,就可以让 Toast 显示出来,通过 toast.show()系统就会显示出这个 Toast。

在当前界面单击长时间显示 Toast 这个按钮后,程序不但长时间地显示了这个 Toast 外,我们还使用了 Notification 用于在状态栏中提醒用户。Notification 的使用在前边已经详细讲解了,这里就不再重复,但是在 ActivityToast 中的 showNotification 方法有一个地方是需要注意的,具体为。

- notification.vibrate = new long[] { 100, 250, 100, 500 },这段代码表示,我们对 Notification 设置了振动,并且是 100ms 延迟后,振动 250ms,然后停止 100ms 后再振动 500ms。

5. 第五步

当写完代码后需要在 AndroidManifest.xml 文件当中加入对振动的权限控制,要不然程序会因为没有振动权限而报错,实现代码如下所示:

```
<uses-permission Android:name="Android.permission.VIBRATE" />
```

7.9 本章小结

本章主要对 Android 中最常用的 UI 组件的使用进行了介绍。当然在 Android 中并不是只有上边介绍的这些 UI 组件,还有其他的一些组件,具体的用户可以参照 Android 的 SDK 文档,或者关注 eoeAndroid.com 论坛,笔者会陆续地将其他的一些组件的介绍,以教程的方式发布上去。通过这章的学习,学会了使用 Button 组件,并且可以让 Button 组件去处理一些基本的事件,如单击事件等。也学会了使用 Menu、ListView、Dialog 等组件。另外,也学习到了 Intent,以及如果通过 Intent 来启动一个新的 Activity。后续的章节还有更精彩的内容,让我们继续吧!



第8章 移动信息仓库——Android 的数据存储操作

学习目标

- Android 在存储方面的系统知识； ☐
- 各类存储的使用及步骤； ☐
- SQLite 方式的存储实现； ☐
- ContentProvider 方式的存储实现。 ☐

8.1 Android 数据存储概述

典型的桌面操作系统提供一种公共文件系统——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）。Android 采用了一种不同的系统，在 Android 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。这一章节描述一个应用软件存储和获取数据、开放数据给其他应用软件、从其他应用软件请求数据并且开放它们的多种方式。

在 Android 中，可供选择的存储方式有 SharedPreferences、文件存储、SQLite 数据库方式、内容提供者（Content provider）和网络，我们将在本章详细介绍。

8.2 轻轻地我保护——SharedPreferences 存储

首先介绍的是 SharedPreferences，它是 Android 提供用来存储一些简单的配置信息的一种机制，例如，一些默认欢迎语、登录的用户名和密码等。其以键值对的方式存储，使得我们可以很方便的读取和存入，下面看一个演示的例子。

1. 第一步

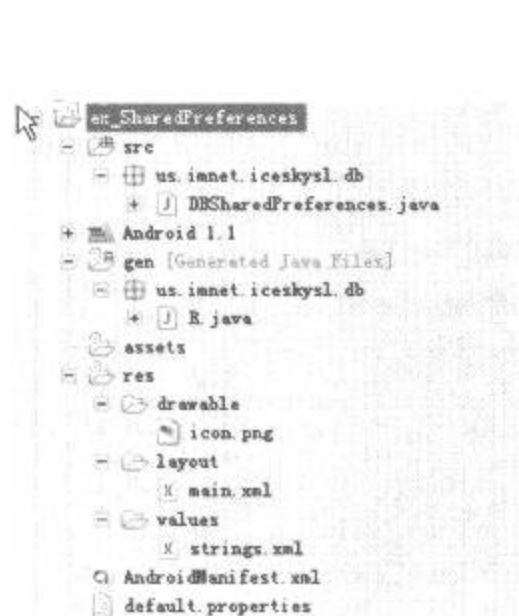
在 Eclipse 中打开 ex_SharedPreferences 项目，其步骤如下所示。

- (1) 新建一个项目。依次单击 File→New→Android Project 项。
- (2) 在新建项目的对话框中，选择 Create project from existing source 项。
- (3) 单击浏览，找到 ex_SharedPreferences 项目，然后单击确定。

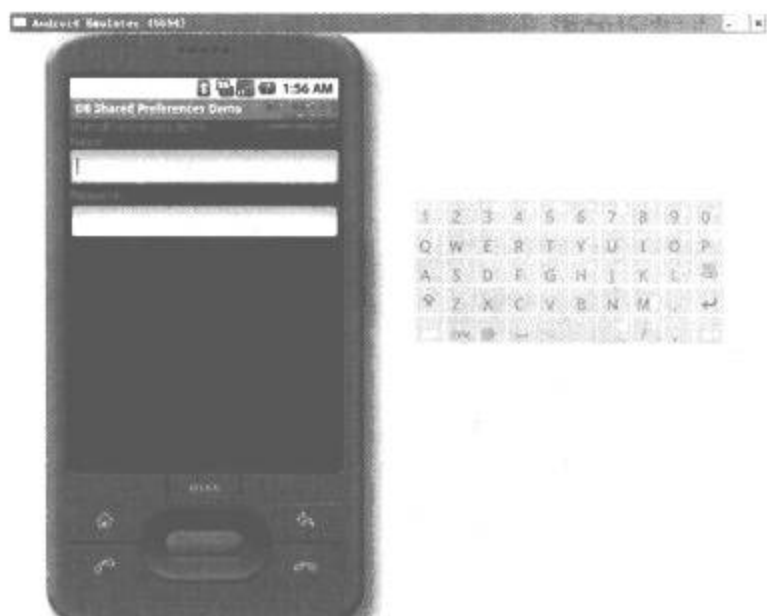
其程序的目录结构如图 8-1 所示。

2. 第二步

单击运行项目, 可以看到主界面如图 8-2 所示, 这个界面的布局信息都在 main.xml 文件当中, 在一个 LinearLayout 当中放了 3 个 TextView 和两个 EditText 组件, 代码如下所示:



▲ 图 8-1 程序目录结构



▲ 图 8-2 主界面

```
<?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="SharedPreferences demo"
        />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Name:" />
    <EditText android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Password:" />
    <EditText android:id="@+id/password"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:password="true"
        android:text="" />
</LinearLayout>
```

代码解释:

如上代码表示, 使用 LinearLayout 布局, 其中放置 3 个用来做界面提示的文本框 (TextView) 组件和两个用于输入 Name 和 Password (注意这里使用了 Android:password="true") 的编辑框

(EditText) 组件, 运行这个应用, 可以看到其界面如图 8-2 所示。

在图 8-2 中, 我们可以看到, 初始状态下两个 EditText 都是空的, 现在输入一些字符, 如图 8-3 所示。

如图 8-3 所示, 我们在 Name 文本框中输入 lceskYsl, 在 Password 文本框中输入 Password, 然后退出这个应用。我们在应用程序列表中找到这个应用, 重新启动, 可以看到其使用了前面输入的名称和 Password, 如图 8-4 所示。



▲ 图 8-3 输入 Name 和 Password



▲ 图 8-4 重新启动应用

由此可见, 应用保存了我们输入的名称和 Password, 现在来看看其实现的代码, 在 DBShared Preferences.java 文件中, 此文件的代码如下所示:

```
package us.imnet.iceskysl.db;

import Android.app.Activity;
import Android.content.SharedPreferences;
import Android.os.Bundle;
import Android.widget.EditText;

public class DBSharedPreferences extends Activity {
    public static final String SETTING_INFOS = "SETTING_Infos";
    public static final String NAME = "NAME";
    public static final String PASSWORD = "PASSWORD";
    private EditText field_name; //接收用户名的组件
    private EditText filed_pass; //接收密码的组件

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //Find View
        field_name = (EditText) findViewById(R.id.name); //首先获取用来输入用户名的组件
        filed_pass = (EditText) findViewById(R.id.password); //同时也需要获取输入密码的组件
        // Restore preferences
        SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0); //获取一个
        SharedPreferences 对象
        String name = settings.getString(NAME, ""); //取出保存的 NAME
        String password = settings.getString(PASSWORD, ""); //取出保存的 PASSWORD
    }
}
```

```

//Set value
field_name.setText(name); //将取出来的用户名赋予 field_name
filed_pass.setText(password); //将取出来的密码赋予 filed_pass

}

@Override
protected void onStop() {
    super.onStop();
    SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0); //首先获取一个
    SharedPreferences 对象
    settings.edit()
    .putString(NAME, field_name.getText().toString())
    .putString(PASSWORD, filed_pass.getText().toString())
    .commit();
    } //将用户名和密码保存进去
}

```

代码解释:

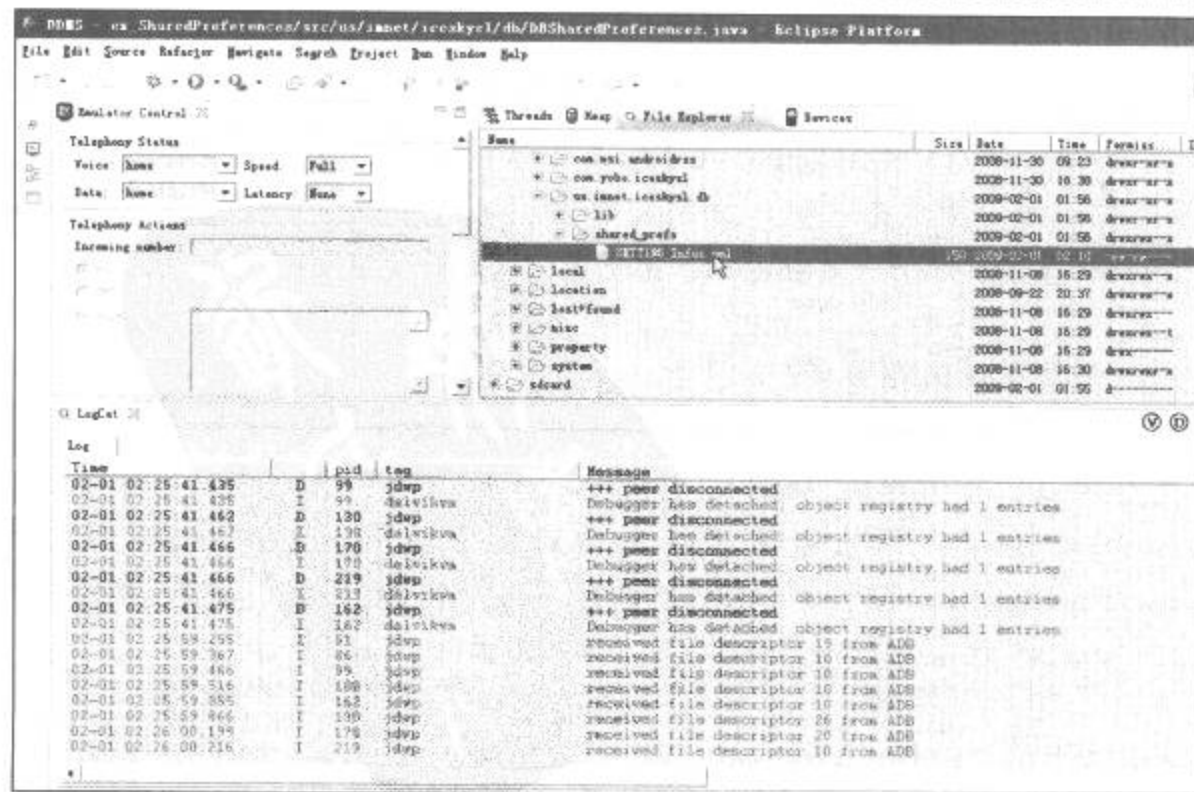
通过上述代码可以看到, 在 `onCreate` 中使用 `findViewById` 得到两个 `EditText` 后, 使用 `getSharedPreferences` 取得 `SharedPreferences` 对象 `settings`, 然后使用 `getString` 取得其中保存的值, 最后使用 `setText` 将其值设置为两个 `EditText` 的值。

而在程序运行 `onStop` 过程, 也就是在程序退出时, 首先使用 `getSharedPreferences` 得到 `settings`; 然后调用 `edit()` 方法使其处于可以编辑状态, 并使用 `putString` 将两个 `EditText` 中的值保存起来; 最后使用 `commit()` 方法提交即可保存。

小知识

SharedPreferences 保存到哪里去了?

SharedPreferences 是以 XML 的格式以文件的方式自动保存的, 在 DDMS 中的 File Explorer 中展开到 `/data/data/<package name>/shared_prefs` 下, 以上面这个为例, 可以看到一个叫做 `SETTING_Infos.xml` 的文件, 如图 8-5 所示。



▲ 图 8-5 SharedPreferences 文件

将其导出到设备中，可以打开这个文件，看到其代码内容为：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="PASSWORD">Password</string>
<string name="NAME">IceskySl</string>
</map>
```

小知识

我们可以通过「getXXX」函数，从 SharedPreferences 中读取不同类型的内容，例如，上面我们使用[getString]读取 String 类型的内容。

注意

Preferences 只能在同一个包内使用，不能在不同的包之间使用。

8.3 谁的文件，谁主宰——文件存储

前面介绍的 Shared Preferences 存储方式非常方便，但是其只适合存储比较简单的数据，如果需要存储更多的数据，可行选择的方式有好几种，这里先给读者介绍文件存储的方法。

和传统的 Java 中实现 I/O 的程序类似，在 Android 中，其提供了 openFileInput 和 openFileOutput 方法读取设备上的文件，下面看个例子代码，具体如下所示：

```
String FILE_NAME = "tempfile.tmp"; //确定要操作文件的文件名
FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE); //初始化
FileInputStream fis = openFileInput(FILE_NAME); //创建写入流
```

代码解释：

上述代码中两个方法只支持读取该应用目录下的文件，读取非其自身目录下的文件将会抛出异常。需要提醒的是，如果调用 FileOutputStream 时指定的文件不存在，Android 会自动创建它。另外，在默认情况下，写入的时候会覆盖原文件内容，如果想把新写入的内容附加到原文件内容后，则可以指定其模式为 Context.MODE_APPEND。

注意

默认情况下，使用 openFileOutput 方法创建的文件只能被其调用的应用使用，其他应用无法读取这个文件，如果需要在不同的应用中共享数据，可以使用 Content Provider 实现，关于 Content Provider 我们将在稍后的内容中介绍。

小知识

资源文件放在哪里？

如果你的应用需要一些额外的资源文件，例如，一些用来测试你写的音乐播放器是否可以正常工作的 MP3 文件，可以将这些文件放在应用程序的/res/raw/下，如 mydatafile.mp3。那么就可以在你的应用中使用 getResources 获取资源后，以 openRawResource 方法（不带后缀的资源文件名）打开这个文件，实现代码如下所示：

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

除了前面介绍的读写文件外，Android 还提供了诸如 deleteFile、fileList 等方法来操作文件，不再赘述。

8.4 打造自己的数据库存储——SQLite 存储方式

8.4.1 Android 中对数据库操作

在前边的章节当中主要学习关于 Android 在布局 and 显示方面的知识，在这一节中将开始学习 Android 应用的另外一个方面：数据存储。用户可以将自己的数据存储到文件系统或者数据库当中，当然最经常的是，用户将自己的数据存储到 SQLite 数据库当中。SQLite 是 Android 所带的一个标准的数据库，它支持 SQL 语句，它是一个轻量级的嵌入式数据库。

在这个例子里边，我们在程序的主界面有一些按钮，通过这些按钮可以对数据库进行标准的增、删、改、查。

通过这个例子我们可以学到：

- 如何新建一个数据库；
- 如何新建数据库里边的数据表；
- 如何删除数据库里边的数据表；
- 如何在数据表中添加新数据；
- 如何删除数据库表中的数据；
- 如何使用 Android 提供的工具 File explore，来查看和删除模拟器当中的数据库表；
- 如何使用 LogCat 来看程序当中打印的日志。

1. 第一步

在 Eclipse 中，打开 ex08_1_SQLite 项目，具体步骤如下。

- 新建一个项目。依次单击 File→New→Android Project 项。
- 在新建项目的对话框中，选择 Create project from existing source 项。
- 单击浏览，找到 ex08_1_SQLite 项目，然后单击确定。

程序的目录结构如图 8-6 所示。

2. 第二步

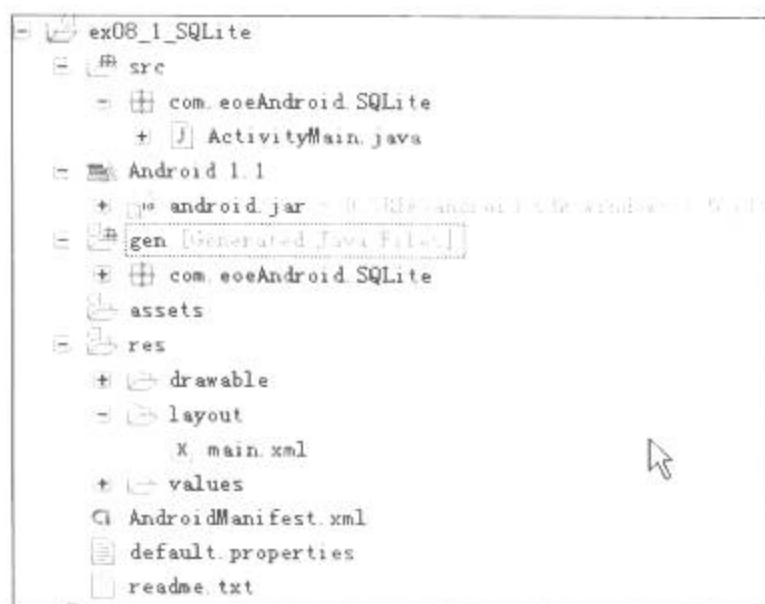
单击运行项目，我们可以看到主界面如图 8-7 所示，这个界面的布局信息都在 main.xml 文件中，在一个 LinearLayout 当中数值排列了 5 个 Button。

3. 第三步

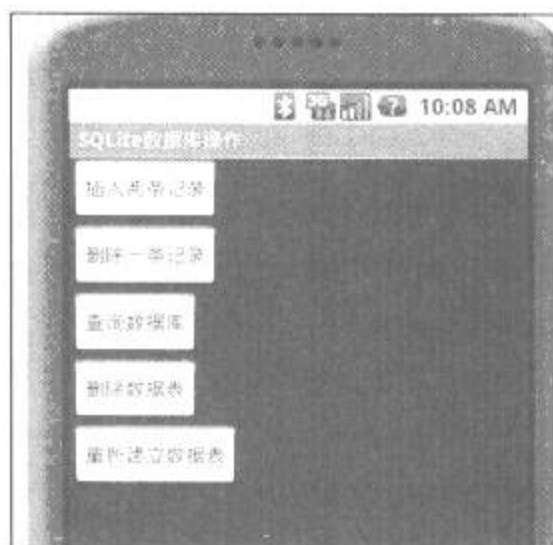
小知识

什么是 SQLiteDatabase?

一个 SQLiteDatabase 的实例代表了一个 SQLite 的数据库，通过 SQLiteDatabase 实例的一些方法，我们可以执行 SQL 语句，对数据库进行增、删、查、改的操作。需要注意的是，数据库对于一个应用来说是私有的，并且在一个应用当中，数据库的名字也是惟一的。



▲ 图 8-6 程序目录结构图



▲ 图 8-7 主界面

小知识

什么是 SQLiteOpenHelper ?

根据这名字，我们可以看出这个类是一个辅助类。这个类主要生成一个数据库，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()` 方法的时候，如果当时没有数据，那么 Android 系统就会自动生成一个数据库。SQLiteOpenHelper 是一个抽象类，我们通常需要继承它，并且实现里边的 3 个函数，具体函数如下所示。

- `onCreate (SQLiteDatabase)`: 在数据库第一次生成的时候会调用这个方法，一般我们在这个方法里边生成数据库表。
- `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里边删除数据表，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。
- `onOpen (SQLiteDatabase)`: 这是当打开数据库时的回调函数，一般也不会用到。

我们在 ActivityMain 文件中看下边这个内部类。DatabaseHelper 类继承 SQLiteOpenHelper，具体代码如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // SQL 语句
        String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
            + " text not null, " + BODY + " text not null " + ")";
        Log.i("haiyang:createDB=", sql);
        //执行这条 SQL 语句
        db.execSQL(sql);
    }
}
```



```

}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
}
}

```

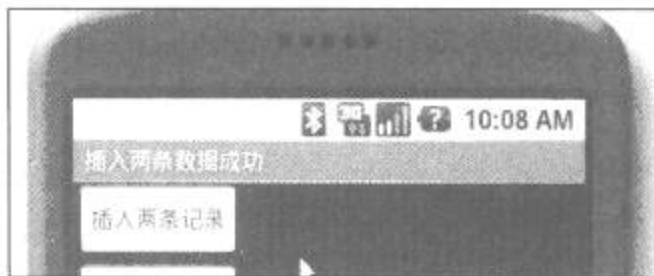
❏ 代码解释：

- DatabaseHelper 类继承了 SQLiteOpenHelper 类，并且重写了 onCreate 和 onUpgrade 方法。
- 在 onCreate() 方法里边首先我们构造一条 SQL 语句，然后调用 db.execSQL(sql) 执行 SQL 语句。这条 SQL 语句为我们生成了一张数据库表。
- 目前我们还不需要升级数据库，所以我们在 onUpgrade() 函数里边没有执行任何操作。

4. 第四步

我们单击插入两条记录的按钮，如果数据成功插入到数据库当中的 diary 表中，那么在界面的 title 区域就会有成功的提示，如图 8-8 所示。

单击这个按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 insertItem 方法，其具体代码如下所示：



```

private void insertItem() {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    //首先生成 SQL 语句
    String sql1 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
    + ") values('haiyang', 'Android的发展真是迅速啊');";
    String sql2 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
    + ") values('icesky', 'Android的发展真是迅速啊');";
    try {
        Log.i("haiyang:sql1=", sql1);
        Log.i("haiyang:sql2=", sql2);
        db.execSQL(sql1);
        db.execSQL(sql2);
        setTitle("插入两条数据成功");
    } catch (SQLException e) {
        setTitle("插入两条数据失败");
    }
}
}

```

❏ 代码解释：

- SQLiteDatabase db = mOpenHelper.getWritableDatabase() 这条语句负责得到一个可写的 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。
- sql1 和 sql2 是我们构造的标准的插入 SQL 语句，如果对 SQL 语句不是很熟悉，可以参考相关的书籍。鉴于本书的重点是在 Android 方面，所以对 SQL 语句的构建不进行详细的介绍。
- Log.i() 会将参数内容打印到日志当中，并且打印级别是 Info 级别，在使用 LogCat 工具的时候我们会进行详细的介绍。
- db.execSQL(sql1) 语句对 SQL 语句进行执行。

小知识

对 Android 的打印级别介绍

Android 支持 5 种打印级别，分别是 Verbose、Debug、Info、Warning、Error，当然我们在程序当中一般用到的是 Info 级别，即将一些自己需要知道的信息打印出来，如图 8-9 所示。



▲ 图 8-9 Android 中的 5 种打印级别

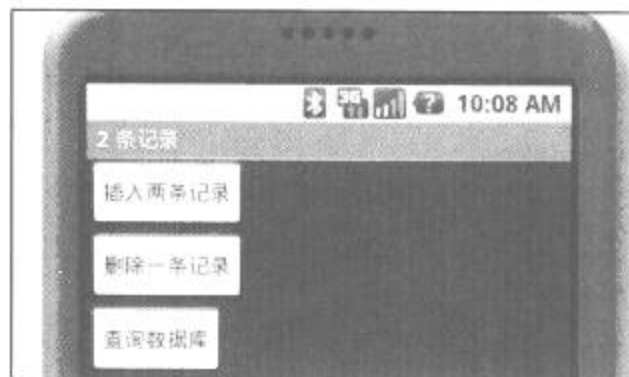
注意

虽然执行 SQL 语句在 Android 当中并没有强制放在 try catch 语句当中，但是我们最好这么做。并在 catch 模块中将错误信息打印在日志当中。这样一方面方便调试，另一方面可以加强程序的健壮性。

5. 第五步

单击查询数据库的按钮，会在界面的 title 区域显示当前数据表当中数据的条数，刚才我们插入了两条，那么现在单击后应该显示为两条，如图 8-10 所示。

单击这个按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 showItems 方法，具体代码如下所示：



▲ 图 8-10 显示两条数据

```
private void showItems() {
    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
    String col[] = { TITLE, BODY };
    //进行数据库查询
    Cursor cur = db.query (TABLE_NAME, col, null, null, null, null, null);
    Integer num = cur.getCount();
    setTitle(Integer.toString(num) + " 条记录");
}
```

代码解释：

- SQLiteDatabase db = mOpenHelper.getReadableDatabase()语句首先得到一个可读的数据库。
- Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null)语句将查询到的数据放到一个 Cursor 当中。这个 Cursor 里边封装了这个数据表 TABLE_NAME 当中的所有条列。query()方法相当的有用，在这里我们简单地讲一下。

- 第一个参数是数据库里边表的名字，比如在我们这个例子，表的名字就是 TABLE_NAME，也就是"diary"。
- 第二个字段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里边来。
- 第三个参数为 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。
- 第四个参数为 selectionArgs。在 selection 部分，你有可能用到“？”，那么在 selectionArgs 定义的字符串会代替 selection 中的“？”。
- 第五个参数为 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。
- 第六个参数为 having，相当于 SQL 语句当中的 having 部分。
- 第七个参数为 orderBy，来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。
- Integer num = cur.getCount()语句通过 getCount()方法，可以得到 Cursor 当中数据的个数。

小知识

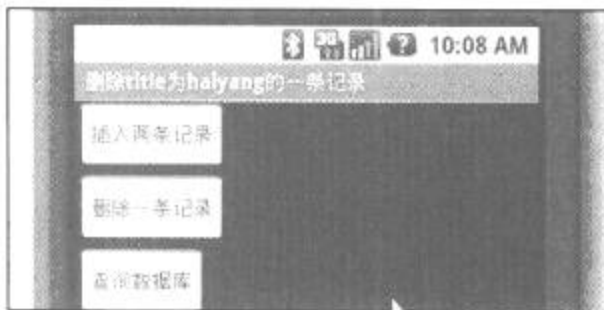
什么是 Cursor ？

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。

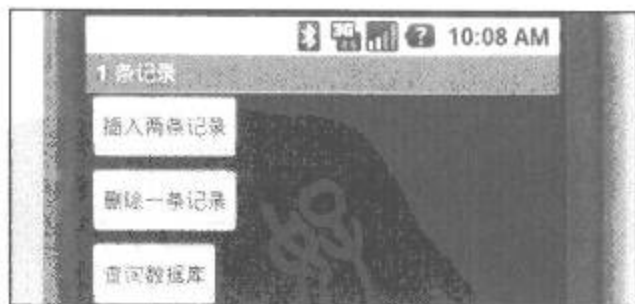
6. 第六步

单击删除一条数据库的按钮后，如果成功删除，我们可以看到在屏幕的标题（title）区域有文字提示，如图 8-11 所示。

现在我们再单击查询数据库按钮，看数据库里边的记录是不是少了一条。单击查询数据库按钮后，出现如图 8-12 所示的界面。



▲ 图 8-11 成功删除一条记录



▲ 图 8-12 还剩一条记录

下面我们来看一下如何删除数据。

单击删除一条记录的按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 deleteItem 方法，其代码如下所示：

```
private void deleteItem() {
    try {
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        //进行删除操作
        db.delete(TABLE_NAME, " title = 'haiyang'", null);
    }
}
```

```

        setTitle("删除 title 为 haiyang 的一条记录");
    } catch (SQLException e) {
    }
}

```

 代码解释：

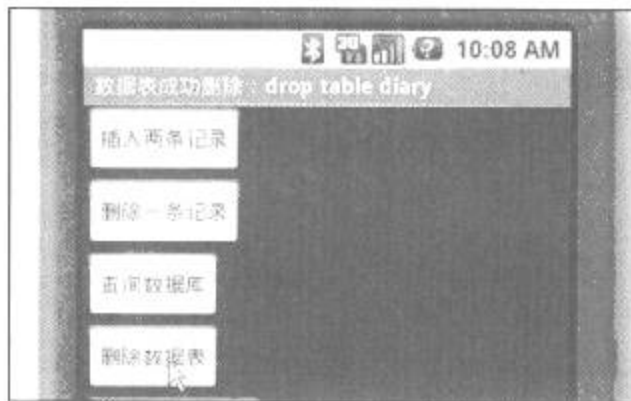
- db.delete(TABLE_NAME, " title = 'haiyang'", null)语句删除了一条 title='haiyang'的数据。当然如果有很多条数据 title 都为'haiyang', 那么一并删除。我们对 delete 方法的参数进行以下介绍。

- 第一个参数是数据库表名, 在这里是 TABLE_NAME, 也就是 diary。
- 第二个参数, 相当于 SQL 语句当中的 where 部分, 也就是描述了删除的条件。
- 如果在第二个参数当中有“?”符号, 那么第三个参数中的字符串会依次替换在第二个参数当中出现的“?”符号。

7. 第七步

单击删除数据表, 我们可以删除 diary 这张数据表, 如图 8-13 所示。

下边我们看在代码部分, 是怎么实现删除的, 具体代码如下所示:



▲ 图 8-13 删除数据库表

```

private void dropTable() {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String sql = "drop table " + TABLE_NAME;
    try {
        //执行 SQL 语句
        db.execSQL(sql);
        setTitle("数据表成功删除: " + sql);
    } catch (SQLException e) {
        setTitle("数据表删除错误");
    }
}

```

 代码解释：

首先我们构造了一个标准的删除数据表的 SQL 语句, 然后执行这条语句 db.execSQL(sql)。

8. 第八步

现在单击其他的按钮, 程序运行时有可能会出现异常, 我们单击重新建立数据表按钮, 如图 8-14 所示。

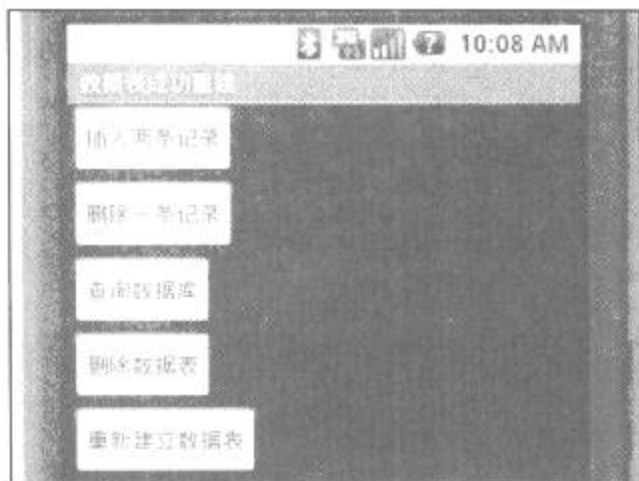
现在我们单击查询数据库, 看里边是否有数据, 如图 8-15 所示。

下边我们看一下程序是如何建立一张新表的, 具体实现代码如下所示:

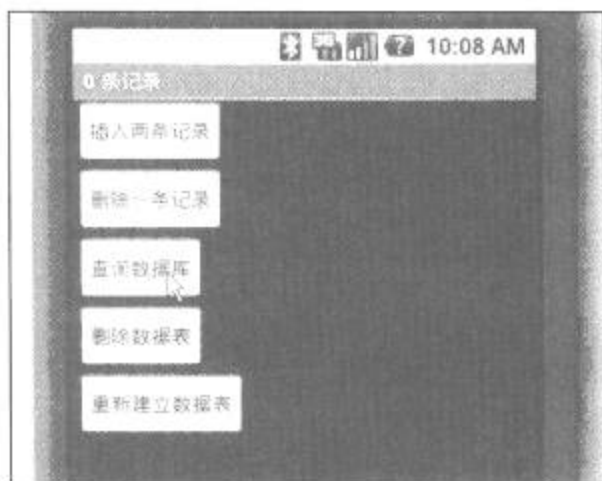
```

private void CreateTable() {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
        + " text not null, " + BODY + " text not null " + ");";
}

```



▲ 图 8-14 重新建立数据库表



▲ 图 8-15 新建的表里边没有数据

```
Log.i("haiyang:createDB=", sql);

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

代码解释：

- sql 变量表示的语句为标准的 SQL 语句，负责按要求建立一张新表。
- db.execSQL("DROP TABLE IF EXISTS diary")语句表示，如果存在 diary 这张表，我们需要先删除，因为在同一个数据库当中不能出现两张同样名字的表。
- db.execSQL(sql)语句执行 SQL 语句，新表建立。

8.4.2 完整地操作数据库——日记本实例

在上一个例子中，我们对 Android 系统自带的 SQLite 数据库进行了初步的学习，了解了一些增、删、改、查的基本工作。在这一节的例子当中，我们做了一个非常简便的日记本程序，虽然没有完善，但是已经是基本可以使用了。在例子当中，我们不但要对数据库进行增、删、改、查的操作，而且还要把数据库当中的数据显示在一个 ListView 当中，通过对 ListView 的操作，实现对数据的增、删、改、查操作。

通过这个例子我们可以学到以下操作：

- 如何对 DatabaseHelper 和 SQLiteDatabase 封装，以便让我们访问数据库更加方便和安全；
- 如何利用 ContentValues 类来代替原始的 SQL 语句进行数据库的操作；
- 如何使用 SimpleCursorAdapter 类和 ListView 配合进行 ListView 的显示。

日记本具体实现步骤如下所述。

1. 第一步

在 Eclipse 中打开 ex08_2_SQLite 项目，具体操作步骤如下。

- 新建一个项目。单击 File→New→Android Project 项。
- 在新建项目的对话框中，选择 Create project from existing source 项。
- 单击浏览，找到 ex08_2_SQLite 项目，然后单击确定。

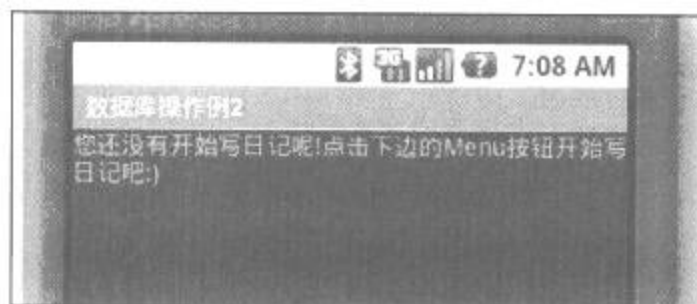
程序的目录结构如图 8-16 所示。

2. 第二步

我们首先运行一下建立的程序，将会出现如图 8-17 所示。



▲ 图 8-16 程序的目录结构



▲ 图 8-17 没有任何数据的程序主界面

程序的主 Activity 是 ActivityMain, 它是一个 ListActivity, 和它关联的布局文件是 diary_list.xml。关于 ListActivity 的介绍。请参阅第 7 章关于 ListView 的介绍。

3. 第三步

在继续操作前，让我们重点关注一下 DiaryDbAdapter 类，这个类封装了 DatabaseHelper 和 SQLiteDatabase 类，使得我们对数据库的操作更加安全和方便。

在 DiaryDbAdapter 的类变量里，主要定义了以下几个变量：

- 数据库、数据表、数据表中列的名字；
- DatabaseHelper 和 SQLiteDatabase 的实例；
- Context 实例。

DatabaseHelper 类的定义和上一个例子一样，只不过这个例子里边，我们在 onUpgrade 增加了升级的代码，具体如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //生成数据库
```

```

        db.execSQL(DATABASE_CREATE);

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS diary");
            onCreate(db);
        }
    }
}

```

代码解释:

在 DiaryDbAdapter 类里, 向外界提供了以下一些方法。

- open(), 调用这个方法后, 如果数据库还没有建立, 那么会建立数据库, 如果数据库已经建立了, 那么会返回可写的数据库实例。
- close(), 调用此方法, DatabaseHelper 会关闭对数据库的访问。
- createDiary (String title, String body) 通过一个 title 和 body 字段在数据库当中创建一条新的纪录。
- deleteDiary (long rowId) 通过记录的 id, 删除数据库中的那条记录。
- getAllNotes() 得到 diary 表中所有的记录, 并且以一个 Cursor 的形式进行返回。
- getDiary (long rowId) 通过记录的主键 id, 得到特定的一条记录。
- updateDiary (long rowId, String title, String body) 更新主键 id 为 rowId 那条记录中的两个字段 title 和 body 字段的内容。

小知识

什么是 ContentValues 类?

ContentValues 类和 Hashtable 比较类似, 它也是负责存储一些名值对, 但是它存储的名值对当中的名是一个 String 类型, 而值都是基本类型。

我们回顾一下, 在上一个例子当中, 我们是通过 SQL 语句进行插入操作, SQL 语句的好处是比较直观, 但是容易出错。但是在这个例子当中我们有更好的办法, 在这里我们将要插入的值都放到一个 ContentValues 的实例当中, 然后执行插入操作, 具体代码如下所示:

```

public long createDiary(String title, String body) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_TITLE, title);
    initialValues.put(KEY_BODY, body);
    Calendar calendar = Calendar.getInstance();
    // 生成年月日字符串
    String created = calendar.get(Calendar.YEAR) + "年" + calendar.get(Calendar.MONTH) + "月" + calendar.get(Calendar.DAY_OF_MONTH) + "日" + calendar.get(Calendar.HOUR_OF_DAY) + "时" + calendar.get(Calendar.MINUTE) + "分";
    initialValues.put(KEY_CREATED, created);
    return mDb.insert(DATABASE_TABLE, null, initialValues);
}

```

代码解释:

- ContentValues initialValues = new ContentValues() 语句实例化一个 contentValues 类。
- initialValues.put (KEY_TITLE, title) 语句将列名和对应的列值放置到 initialValues 里边。
- mDb.insert (DATABASE_TABLE, null, initialValues) 语句负责插入一条新的纪录, 如果插入成功则会返回这条记录的 id, 如果插入失败会返回-1。

在更新一条记录的时候，我们也是采用 ContentValues 的这套机制，具体代码如下所示：

```
public boolean updateDiary(long rowId, String title, String body) {
    ContentValues args = new ContentValues();
    args.put(KEY_TITLE, title);
    args.put(KEY_BODY, body);
    Calendar calendar = Calendar.getInstance();
    String created = calendar.get(Calendar.YEAR) + "年"
        + calendar.get(Calendar.MONTH) + "月"
        + calendar.get(Calendar.DAY_OF_MONTH) + "日"
        + calendar.get(Calendar.HOUR_OF_DAY) + "时"
        + calendar.get(Calendar.MINUTE) + "分";
    args.put(KEY_CREATED, created);

    return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
```

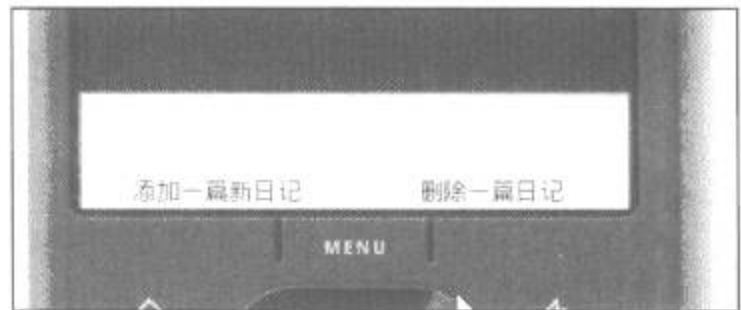
❏ 代码解释：

实现更新一条记录。

4. 第四步

现在返回到程序的主界面，对应的 Activity 是 ActivityMain。

当我们单击 menu 按钮后会出现如图 8-18 所示界面。



▲ 图 8-18 单击 menu 按钮

根据我们第 7 章对 menu 的学习，对单击 menu 里边按钮的处理逻辑全部放在 onOptionsItemSelected 函数里，具体代码如下所示：

```
public boolean onOptionsItemSelected(int featureId, MenuItem item) {
    switch (item.getItemId()) {
        case INSERT_ID:
            createDiary();
            return true;
        case DELETE_ID:
            mDbHelper.deleteDiary(getListView().getSelectedItemId());
            renderListView();
            return true;
    }
    return super.onOptionsItemSelected(featureId, item);
}
```

❏ 代码解释：

- 如果单击添加一篇新日记按钮那么会执行到 createDiary() 语句。
- 如果单击删除一条记录，会执行 mDbHelper.deleteDiary(getListView().getSelectedItemId()) 语句，首先删除当前被选中的某一项所对应的数据库当中的记录。
- renderListView() 语句重新对界面刷新。

在 createDiary() 函数里边的代码如下所示：

```
private void createDiary() {
    Intent i = new Intent(this, ActivityDiaryEdit.class);
    startActivityForResult(i, ACTIVITY_CREATE);
}
```

 代码解释:

- 首先构造了一个 intent, 这个 intent 负责跳转到 ActivityDiaryEdit 里。
- 然后启动这个 intent, 并且需要返回值。

5. 第五步

在 ActivityMain 中, 有多处地方都用到了 renderListView() 函数。在 onCreate 里边用这个函数显示 ListView。当 ListView 需要发生变化后, 例如, 删除了一条记录或者增加了一条记录的时候, 我们调用这个函数进行刷新 ListView, 下边来看一下此函数的实现, 具体代码如下所示:

```
private void renderListView() {
    mDiaryCursor = mDbHelper.getAllNotes();
    startManagingCursor(mDiaryCursor);
    String[] from = new String[] { DiaryDbAdapter.KEY_TITLE,
        DiaryDbAdapter.KEY_CREATED };
    int[] to = new int[] { R.id.text1, R.id.created };
    SimpleCursorAdapter notes = new SimpleCursorAdapter(this,
        R.layout.diary_row, mDiaryCursor, from, to);
    setListAdapter(notes);
}
```

 代码解释:

- mDiaryCursor = mDbHelper.getAllNotes() 语句, 我们首先获取数据库当中的所有数据, 这些数据以 Cursor 的形式存在。
- startManagingCursor (mDiaryCursor) 语句, 我们将生成的 Cursor 交给 Activity 来管理, 这样的好处是系统能自动做很多事情, 比如当程序暂停的时候, 这个系统可以卸载 Cursor 以节省空间, 当程序重新启动的时候系统重新查询生成 Cursor。
- String[] from 里边定义了 ListView 每一排对应的数据是从数据库中的哪个列表里选取。
- 和 SimpleAdapter 类似 int[] to 里边是一个 View 的数组。这些 View 只能是 TextView 或者 ImageView。这些 View 是以 id 的形式来表示的, 如 Android.R.id.text1。
- SimpleCursorAdapter notes = new SimpleCursorAdapter(this, R.layout.diary_row, mDiaryCursor, from, to) 语句生成一个 SimpleCursorAdapter, 我们介绍以下每一个参数的意义。
 - 第一个参数是 Context。
 - 第二个参数为 R.layout.diary_row, 它关联在 diary_row.xml 文件当中定义的 Layout, 这个 Layout 规定 ListView 当中每一项的布局。
 - 第三个参数就是 Cursor。
 - 第四个参数是数据表当中的列的数组, 只有在这里边出现的列名, 数据才会对应的填充在 to 里边对应的 TextView 或者 ImageView 当中。
 - 第五个参数是在 ListView 里边每一项中需要被数据填充的 TextView 或者 ImageView。
- setListAdapter (notes) 语句将 SimpleCursorAdapter 和 ListActivity 里边的 ListView 绑定起来, 至此在界面当中才会显示出列表来。

什么是 SimpleCursorAdapter ?

小知识

在第 7 章, 我们已经介绍过了 ArrayAdapter 和 SimpleAdapter。和它们俩类似, SimpleCursorAdapter 也是集成 Adapter。ArrayAdapter 负责把一个字符串数组中的数据填充到一个 ListView 当中, 而对应的 SimpleCursorAdapter 负责把 Cursor 里面的内容填充到 ListView 当中。通过 SimpleCursorAdapter 可以把数据库当中一列的数据和 ListView 中一排进行对应起来。和前两个 Adapter 类似, 要求和数据进行对应的 View 必须是 TextView 或者 ImageView。

6. 第六步

单击添加一条数据的按钮, 程序运行界面如图 8-19 所示。

这个界面对应的 Activity 是 ActivityDiaryEdit, 对应的布局文件是 diary_edit.xml。对于这个布局文件里边用到了 LinearLayout、TextView 和 EditText, 这些我们都已经讲过, 在这里不再赘述, 具体看一下代码:

```
<?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">

    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/title"
            android:padding="2px" />
        <EditText android:id="@+id/title"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1" />
    </LinearLayout>

    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="@string/body" />
    <EditText android:id="@+id/body" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:layout_weight="1"
        android:scrollbars="vertical" />

    <Button android:id="@+id/confirm" android:text="@string/confirm"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

代码解释:

在一个 LinearLayout 里边我们放置了一些文本框、输入框和一些 Button 按钮。



▲ 图 8-19 新建一篇日记的界面

当程序运行输入内容后，单击确定按钮，日记就会保存到数据库当中。下边来看一下代码具体是怎么执行的。当单击确定按钮后，系统回调执行和按钮绑定的单击监听器里边的 `onClick` 方法，具体代码如下所示：

```
public void onClick(View view) {
    String title = mTitleText.getText().toString();
    String body = mBodyText.getText().toString();
    if (mRowId != null) {
        mDbHelper.updateDiary(mRowId, title, body);
    } else
        mDbHelper.createDiary(title, body);
    Intent mIntent = new Intent();
    setResult(RESULT_OK, mIntent);
    finish();
}
```

 代码解释：

- 首先获得 `EditView` 里边的数据。
- 目前 `mRowId` 为 `null`，所以执行 `mDbHelper.createDiary (title, body)` 语句将数据保存到数据当中。
- `setResult (RESULT_OK, mIntent)` 语句设置返回值。

执行完上边的代码后，系统跳转到 `ActivityMain`，并执行回调函数 `onActivityResult`，具体代码如下所示：

```
protected void onActivityResult(int requestCode, int resultCode,
    Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    renderListView();
}
```

 代码解释：

- 在回调函数中，我们重新对 `ListView` 进行刷新，将所有数据重新显示出。

7. 第七步

单击 `ListView` 里边的条列，可以对刚才保存的数据进行编辑。具体怎么实现这个功能的，我们可以看一下 `ActivityMain` 当中的 `onListItemClick` 方法代码：

```
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Cursor c = mDiaryCursor;
    c.moveToPosition(position);
    Intent i = new Intent(this, ActivityDiaryEdit.class);
    i.putExtra(DiaryDbAdapter.KEY_ROWID, id);
    i.putExtra(DiaryDbAdapter.KEY_TITLE,
        c.getString(c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_TITLE)));
    i.putExtra(DiaryDbAdapter.KEY_BODY,
        c.getString(c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_BODY)));
    startActivityForResult(i, ACTIVITY_EDIT);
}
```

 代码解释：

- `c.moveToPosition (position)` 语句将在 `Cursor` 当中的指针移到 `position` 位置，这个 `position`

是我们单击的这个一列在整个列表中的位置。

- `Intent i = new Intent(this, ActivityDiaryEdit.class)` 语句构造一个跳转到 `ActivityDiaryEdit` 的 intent。
 - `putExtra()` 方法负责将要传递的数据放到 intent 当中。
 - `c.getString(c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_TITLE))` 得到这一条数据中列名为 `title` 的值。
 - `c.getString(c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_BODY))` 得到这一条数据中列名为 `body` 的值。
 - `startActivityForResult(i, ACTIVITY_EDIT)` 语句启动 intent, 发生 Activity 的跳转。
- 我们来看一下 `ActivityDiaryEdit` 中的 `onCreate()` 里边的代码:

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
    String title = extras.getString(DiaryDbAdapter.KEY_TITLE);
    String body = extras.getString(DiaryDbAdapter.KEY_BODY);
    mRowId = extras.getLong(DiaryDbAdapter.KEY_ROWID);

    if (title != null) {
        mTitleText.setText(title);
    }
    if (body != null) {
        mBodyText.setText(body);
    }
}
```

 代码解释:

- 对于 `ActivityDiaryEdit` 这个 Activity 有两个 intent 可以跳转进来, 一个是新建一篇日记的时候, 这个时候 intent 里边的 `extras` 部分没有任何数据。第二种情况是在单击列表的某一个条列的时候, 这个时候的 intent 如上所示会携带 `extras` 数据。所以在 `ActivityDiaryEdit` 中我们通过判断 `extras` 是否为 `null`, 就可以判断是哪种 intent 启动的。

- 当 `extras` 不为 `null`, 我们将 `extras` 里边的数据显示出来。

8. 第八步

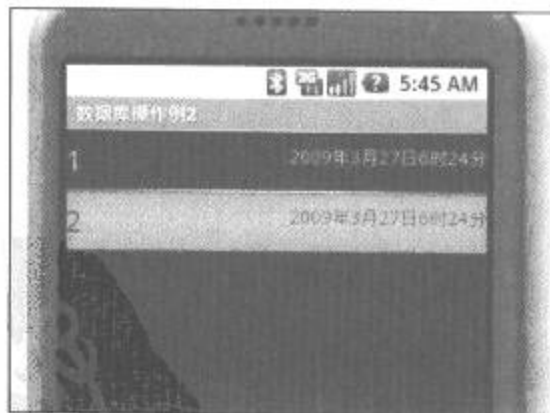
在程序的主界面当中, 上下移动焦点 (可以通过键盘的上下键或者模拟器中的上下按键), 可以对拥有焦点的那一项进行删除, 如图 8-20、图 8-21 和图 8-22 所示。

具体执行 `onMenuItemSelected` 中的代码如下所示:

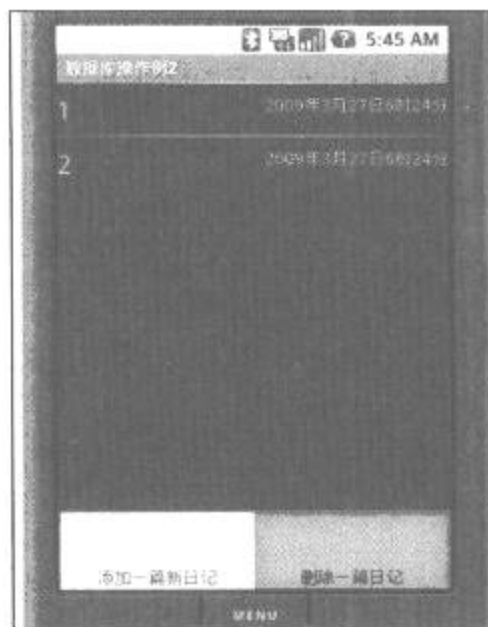
```
mDbHelper.deleteDiary(getListView().getSelectedItemId());
renderListView();
```

 代码解释:

- `getListView()` 方法获取当前的 `ListView` 引用。
- `getSelectedItemId()` 方法得到当前这一列所对应的数据项的 `rowId`, 也就是这条数据在数据库中的主键 `id`。



▲ 图 8-20 上下移动焦点界面



▲ 图 8-21 进行删除界面



▲ 图 8-22 删除后界面

- `mDbHelper.deleteDiary()`方法删去数据库中的这一列数据。
- `renderListView()`负责对列表重新刷新一遍。

至此，对数据库的学习就先告一段落，接下来将学习 `contentProvider`，它是 Android 应用当中非常重要的一部分。而且程序间的大部分数据交换都是通过 `contentProvider` 机制进行。

8.5 我的数据你来用——ContentProvider 介绍

8.5.1 初识 ContentProvider

在第 6 章当中，介绍了组成 Android 程序的主要 4 部分，它们分别是。

- Activity。
- Broadcast Intent Receiver。
- Service。
- Content Provider。

关于 Activity 和相关 View 的部分，已经在前边章节进行了比较详细的介绍，在这一节中，将学习 Android 应用里另外一个非常重要的部分 ContentProvider。

1. 什么是 ContentProvider

Android 这个系统和其他的操作系统还不太一样，读者需要记住的是，数据在 Android 当中是私有的，当然这些数据包括文件数据和数据库数据以及一些其他类型的数据。那这个时候有读者就会提出问题，难道两个程序之间就没有办法对于数据进行交换？Android 这么优秀的系统不会让这种情况发生的。解决这个问题主要靠 ContentProvider。一个 Content Provider 类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。也就是说，一个程序可以通过实现一个 Content Provider 的抽象接口将自己的数据暴露出去。外界根本看不到，也不用看到这个应用暴露的数据在应用当中是如何存储的，或者用数据库存储还是用文件存储，

还是通过网上获得，这些一切都不重要，重要的是外界可以通过这一套标准及统一的接口和程序里的数据打交道，可以读取程序的数据，也可以删除程序的数据，当然，中间也会涉及一些权限的问题。下边列举一些较常见的接口，这些接口如下所示。

- `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. 什么是 ContentResolver

外界的程序通过 `ContentResolver` 接口可以访问 `ContentProvider` 提供的数据，在 `Activity` 当中通过 `getContentResolver()` 可以得到当前应用的 `ContentResolver` 实例。`ContentResolver` 提供的接口和 `ContentProvider` 中需要实现的接口对应，主要有以下几个。

- `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。

3. ContentProvider 和 ContentResolver 中用到的 Uri

在 `ContentProvider` 和 `ContentResolver` 当中用到了 Uri 的形式通常有两种，一种是指定全部数据，另一种是指定某个 ID 的数据。我们看下面的例子。

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

在上边两个类中用到的 Uri 一般由 3 部分组成。

- 第一部分是: `"content://"`。
- 第二部分是要获得数据的一个字符串片段。
- 最后就是 ID (如果没有指定 ID, 那么表示返回全部)。

由于 URI 通常比较长，而且有时候容易出错，且难以理解。所以，在 `Android` 当中定义了一些辅助类，并且定义了一些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)。

8.5.2 使用 ContentProvider 读取系统数据

在这个例子里边，首先在系统的联系人应用当中插入一些联系人信息，然后把这些联系人的名字和电话再显示出来，通过这个例子可以学到。

- 如何在联系人应用当中添加联系人。

- 如何使用系统提供的 ContentProvider。
- 如何使用 ContentResolver 当中的 query() 方法。

具体实现步骤如下所示。

1. 第一步

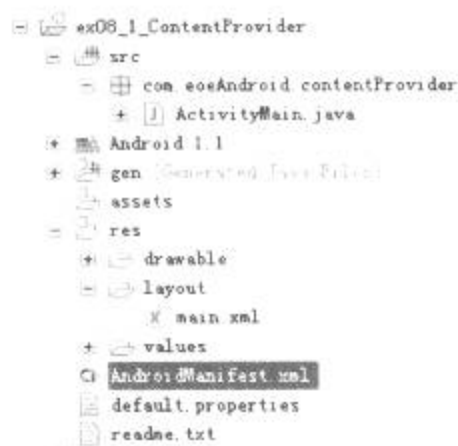
在 Eclipse 中打开 ex09_1_ContentProvider 项目，具体操作如下。

- (1) 新建一个项目，依次单击 File→New→Android Project 项。
- (2) 在新建项目的对话框中，选择 Create project from existing source 项。
- (3) 单击浏览按钮，找到 ex09_1_ContentProvider 项目，然后单击确定按钮。

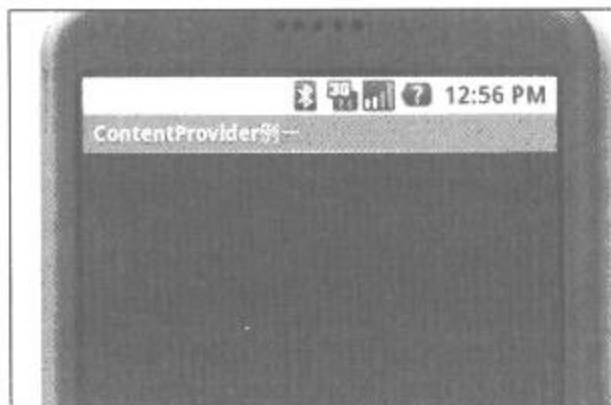
程序的目录结构如图 8-23 所示。

2. 第二步

首先运行这个项目，将会看到如图 8-24 所示的界面。



▲ 图 8-23 程序的目录结构



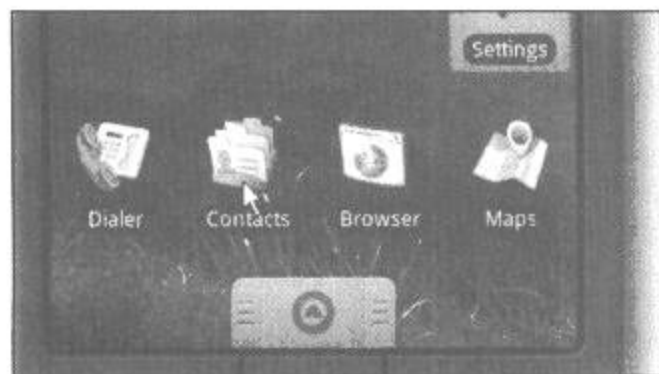
▲ 图 8-24 未添加任何数据的主界面

图 8-24 所示的列表中没有任何数据，接下来的操作是为应用添加几条联系人数据。

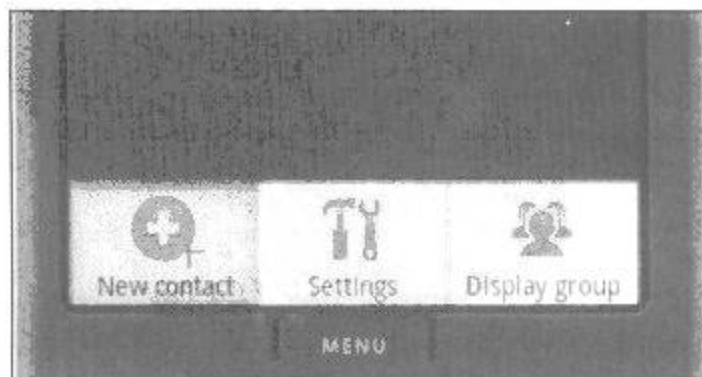
3. 第三步

按照下列图示添加几条数据到联系人列表中，具体步骤如下。

- (1) 单击模拟器的 Home 键，在转出来的界面上，单击桌面上的 Contacts 应用，如图 8-25 所示。
- (2) 进入应用后，单击 MENU 项，在出现的界面上单击 New contact 按钮，如图 8-26 所示。



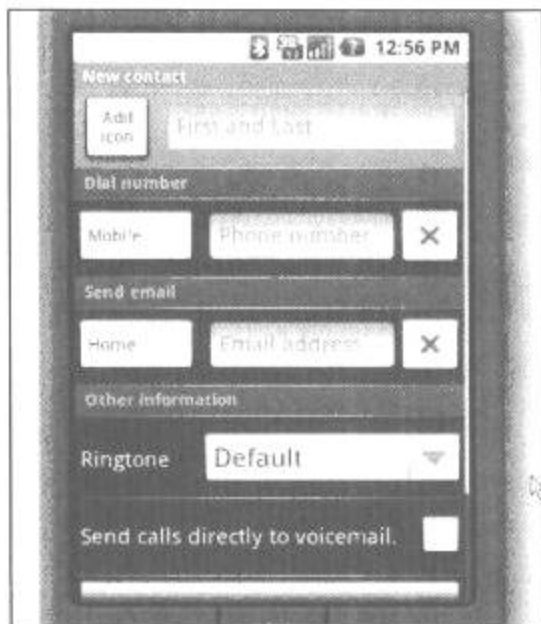
▲ 图 8-25 单击 Contacts 应用



▲ 图 8-26 单击 New contact 按钮

(3) 添加联系人姓名和电话号码信息, 如图 8-27 所示。

(4) 单击 MENU 项, 在返回的界面上单击 Save 项保存, 如图 8-28 所示。



▲ 图 8-27 添加联系人姓名和电话号码

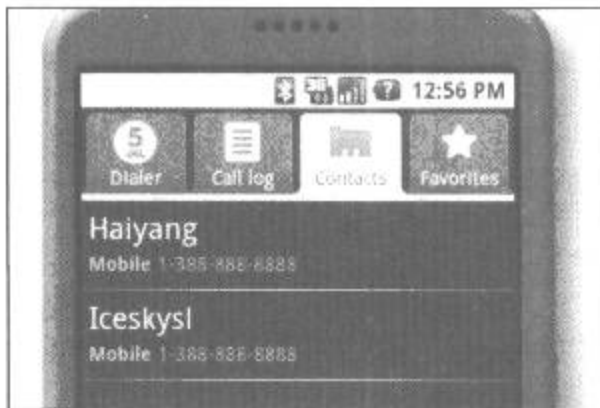


▲ 图 8-28 保存联系人姓名和电话号码信息

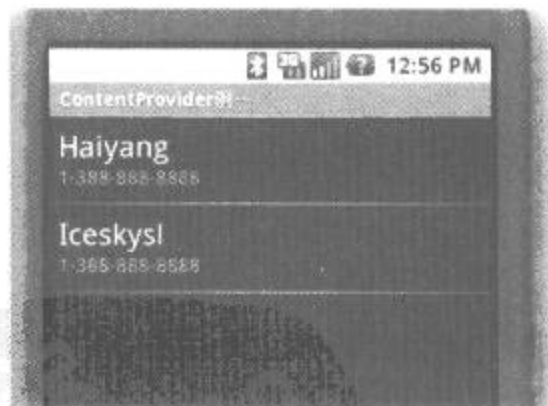
(5) 按照上边的操作步骤, 添加了两条数据后显示如图 8-29 所示。

4. 第四步

再次运行程序, 模拟器显示如图 8-30 所示。



▲ 图 8-29 添加后结果



▲ 图 8-30 模拟器显示

我们看一下程序 ActivityMain 中的 onCreate()方法, 具体代码如下所示:

```
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);
}
```

 代码解释:

- `getContentResolver()`方法得到应用的 `ContentResolver` 实例。
- `query (Phones.CONTENT_URI, null, null, null, null)`。它是 `ContentResolver` 里的方法，负责查询所有联系人，并返回一个 `Cursor`。这个方法参数比较多，每个参数的具体含义如下。
 - 第一个参数为 `Uri`，在这个例子里边这个 `Uri` 是联系人的 `Uri`。
 - 第二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。
 - 第三个参数相当于 SQL 语句的 `where` 部分，描述哪些值是我们需要的。
 - 第四个参数是一个字符串数组，它里边的值依次代替在第三个参数中出现的“?”符号。
 - 第五个参数指定了排序的方式。
- `startManagingCursor(c)`语句让系统来管理生成的 `Cursor`。
- `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 })` 语句生成一个 `SimpleCursorAdapter`。（关于 `SimpleCursorAdapter` 我们在第 7 章已经详细介绍了。）
- `setListAdapter(adapter)`。将 `ListView` 和 `SimpleCursorAdapter` 进行绑定。

8.5.3 使用 ContentProvider 操作数据日记本实例

在上一个例子当中学习了 `ContentProvider`，并且使用了系统中的一个联系人的 `ContentProvider`，在本节当中，我们仍然实现类似第 8 章所讲的日记本的例子，只不过这次我们用 `ContentProvider` 实现，而不是直接用数据库实现。这样外界的程序就可以访问得到日记本这个应用数据。

通过这个例子可以学到。

- 如何实现一个 `ContentProvider`。
- 理解 `UriMatcher` 的含义。
- `onPrepareOptionsMenu` 方法介绍。

具体实现步骤如下所示。

1. 第一步

在 Eclipse 中打开 `ex09_2_contentProvider` 项目，具体操作步骤如下。

- (1) 新建一个项目，依次单击 `File` → `New` → `Android Project` 项。
 - (2) 在新建项目的对话框中，选择 `Create project from existing source` 项。
 - (3) 单击浏览项，找到 `ex09_2_contentProvider` 项目，然后单击确定按钮。
- 程序的目录结构如图 8-31 所示。

2. 第二步

首先运行这个项目，将会看到如图 8-32 所示界面。



▲ 图 8-31 程序的目录结构



▲ 图 8-32 未添加任何数据的主界面

图 8-32 所示的这个界面我们在第 8 章的例子已经见过。本例的主 Activity 仍然是一个 ListActivity，而且关联的布局文件也是 diary_list.xml，关于 ListActivity 的使用方法和详细说明可以参见第 7 章第 6 节。diary_list.xml 的代码如下所示：

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

    <ListView android:id="@+id/android:list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:id="@+id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="您还没有开始写日记呢!单击下边的 Menu 按钮
        开始写日记吧:)" />
</LinearLayout>
```

代码解释：

其中 ListView 的 id 我们必须定义成 Android:id=“@+id/Android:list”，这样系统才可以在 List Activity 里引用的到。

3. 第三步

和第 8 章的前一个日记本例子一样，日记本的数据还是存储在 SQLite 数据库中，但是不一样的是，在这个例子里，执行增、删、改、查操作时不是直接访问数据库，而是通过日记本程序的 ContentProvider 来实现。

我们先来看一下 Diary 这个类，这个类里边有一个内部静态类 DiaryColumns，和它的名字意思一样，这个类里主要定义了日记本数据库的列表字段的名字，具体代码如下所示：

```
public static final class DiaryColumns implements BaseColumns {
    private DiaryColumns() {}
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/diaries");
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.google.diary";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.
```

```
google.diary";
    public static final String DEFAULT_SORT_ORDER = "created DESC";
    public static final String TITLE = "title";
    public static final String BODY = "body";
    public static final String CREATED = "created";
}
```

代码解释:

- **BaseColumns** 是一个接口，里边有两个变量，一个是 `_ID=“_id”`，另一个是 `_COUNT=“count”`。在 Android 当中，每一个数据库表至少有一个字段，而且这个字段是 `_id`。所以当我们构造列名的辅助类时，直接实现 **BaseColumns**，这样我们便默认地拥有了 `_id` 字段。
- 在我们的日记本的数据表里，一共有 4 个字段，分别是：“id”、“title”、“body”、“created”。

小知识

在 Android 的设计“哲学”里是鼓励开发者使用内部类的，这样不但使用方便，而且执行效率也高。

4. 第四步

现在我们来看 **DiaryContentProvider** 类，这个类继承自 **ContentProvider**，在这个类里边实现了 **ContentProvider** 的一些接口方法，并且成为日记本的一个 **ContentProvider**。我们现在通过学习这个类，来详细了解实现一个自定义的 **ContentProvider** 的具体步骤。

(1) 继承 **ContentProvider**。**DiaryContentProvider** 类是继承 **ContentProvider** 类的。

(2) 定义一个 `public static final` 的 **Uri** 类型的变量，并且命名为 `CONTENT_URI`。**Diary Content Provider** 所能处理的 **Uri** 都是基于 `CONTENT_URI` 来构建的，需要注意的是，这个 `CONTENT_URI` 中的内容以 `content://` 开头，并且全部小写，且全局惟一。

下边是在这个例子中的一个普通 **URI**，通过分析这个 **URI**，可以了解 **content URI** 的构成，代码如下所示：

```
| content://com.ex09_2_contentprovider.diarycontentprovider/diaries/1
```

代码解释:

- 第一部分是 `content://`，这部分是一直存在的，也是不用做什么修改的。
- 第二部分是授权（**AUTHORITY**）部分，在这个例子里边就是 `com.ex09_2_contentprovider.diarycontentprovider`，授权部分是惟一的，在程序中一般是我们实现的那个 **Content Provider** 的全称，并且全都小写。这部分是和在 **AndroidManifest.xml** 文件当中的 `<provider Android:name="DiaryContentProvider" Android:authorities="com.ex09_2_contentprovider. diaryconten tprovider" />` 部分对应的。
- 第三部分是请求数据的类型，例如，在这个例子当中定义的类型是 `diaries`。当然这一部分可以是 0 个片段或者多个片段构成，如果 **Content Provider** 只是暴露出了一种类型的数据，那么这部分可以为空，但是如果暴露出了多种，尤其是包含子类的时候，就不能为空，例如，日记本程序里边可以暴露出来两种数据，一种是用户自己的“`diaries/my`”，另一种是其他人的“`diaries/others`”。
- 第四部分就是“1”，当然这部分是允许为空的。如果为空，表示请求全部数据；如果不为

空，表示请求特定 ID 的数据。

(3) 构建用户的数据存储系统。在这个例子当中，是将数据存储到数据库系统当中。通常是将数据存储到数据库系统中，但是也可以将数据存储在其他的地方，如文件系统等。

(4) 实现 `ContentProvider` 这个抽象类的抽象方法，具体如下所示：

- `public boolean onCreate()`，当 `ContentProvider` 生成的时候调用此方法。
- `public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`，此方法返回一个 `Cursor` 对象作为查询结果集。
- `public Uri insert (Uri uri, ContentValues initialValues)`，此方法负责往数据集当中插入一行，并返回这一行的 `Uri`。
- `public int delete (Uri uri, String where, String[] whereArgs)`，此方法负责删除指定 `Uri` 的数据。
- `public int update (Uri uri, ContentValues values, String where, String[] whereArgs)`，此方法负责更新指定 `Uri` 的数据。
- `public String getType (Uri uri)`，返回所给 `Uri` 的 MIME 类型。

(5) 在 `AndroidManifest.xml` 文件中增加 `<provider>` 标签，例如，在这个例子中，实现的代码为：`<provider Android:name="DiaryContentProvider" Android:authorities="com.ex09_2_contentprovider.diarycontentprovider" />`，其中 `Android:name` 为我们实现的这个 content provider 的类名；`Android:authorities` 为 content URI 的第二部分，即授权部分，代码为：`"com.ex09_2_contentprovider.diarycontentprovider"`。

5. 第五步

继续来看 `DiaryContentProvider` 类的代码。此实例的数据是存储在数据库当中，和前面章节学过的数据例子一样，此实例中我们定义了 `DatabaseHelper` 辅助类。这个类在第 8 章中有详细的讲解，这里就不再进行详细说明，具体代码如下所示：

```
private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + DIARY_TABLE_NAME + " ("
            + DiaryColumns._ID + " INTEGER PRIMARY KEY,"
            + DiaryColumns.TITLE + " TEXT," + DiaryColumns.BODY
            + " TEXT," + DiaryColumns.CREATED + " TEXT" + ");");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS notes");
        onCreate(db);
    }
}
```

 代码解释：

`DatabaseHelper` 里操作数据库的辅助类，通过这个类我们可以生成数据库，并且维护这个数据库。

在 `DiaryContentProvider` 中，我们定义了一些变量和常量，其中这些常量主要是描述数据库的信息。

```
private static final String DATABASE_NAME = "database";
private static final int DATABASE_VERSION = 1;
private static final String DIARY_TABLE_NAME = "diary";

private static HashMap<String, String> sDiariesProjectionMap;

private static final int DIARIES = 1;
private static final int DIARY_ID = 2;

private static final UriMatcher sUriMatcher;
```

什么是 UriMatcher?

UriMatcher 是匹配 Uri 的一个辅助类，例如，在我们的 `DiaryContentProvider` 中的 static 模块中，有下边的代码：

❗ 小知识

```
sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
sUriMatcher.addURI(Diary.AUTHORITY, "diaries", DIARIES);
sUriMatcher.addURI(Diary.AUTHORITY, "diaries/#", DIARY_ID);
```

🔍 代码解释：

- `sUriMatcher.addURI(Diary.AUTHORITY, "diaries", sUriMatcher.match(uri))`表示，如果我们的 Uri 是 `content://com.ex09_2_contentprovider.diarycontentprovider/diaries/`，那么 `sUriMatcher.match(uri)` 的返回值就是 `DIARIES`。

- `sUriMatcher.addURI(Diary.AUTHORITY, "diaries/#", DIARY_ID)`表示，如果我们的 Uri 是 `content://com.ex09_2_contentprovider.diarycontentprovider/diaries/id`（其中后边的 id 是一个数字），那么 `sUriMatcher.match(uri)` 的返回值就是 `DIARY_ID`。

通过 UriMatcher 类我们可以很方便地判断一个 Uri 的类型，特别是判断这个 Uri 是对单个数据的请求，还是对全部数据的请求。

6. 第六步

现在我们来查看一下在第四步列出来的抽象方法具体是怎么实现的。我们就从增、删、改、查的顺序说起。

(1) 首先来看插入的方法：`insert()`。

具体实现代码如下所示：

```
@Override
public Uri insert(Uri uri, ContentValues initialValues) {
    if (sUriMatcher.match(uri) != DIARIES) {
        throw new IllegalArgumentException("Unknown URI " + uri);
    }

    ContentValues values;
    if (initialValues != null) {
        values = new ContentValues(initialValues);
    } else {
        values = new ContentValues();
    }

    if (values.containsKey(Diary.DiaryColumns.CREATED) == false) {
```

```

values.put(Diary.DiaryColumns.CREATED, getFormateCreateDate());
}

if (values.containsKey(Diary.DiaryColumns.TITLE) == false) {
    Resources r = Resources.getSystem();
    values.put(Diary.DiaryColumns.TITLE, r
        .getString(Android.R.string.untitled));
}

if (values.containsKey(Diary.DiaryColumns.BODY) == false) {
    values.put(Diary.DiaryColumns.BODY, "");
}

SQLiteDatabase db = mOpenHelper.getWritableDatabase();
long rowId = db.insert(DIARY_TABLE_NAME, DiaryColumns.BODY, values);
if (rowId > 0) {
    Uri diaryUri= ContentUris.withAppendedId(
        Diary.DiaryColumns.CONTENT_URI, rowId);
    return diaryUri;
}

throw new SQLException("Failed to insert row into " + uri);
}

```

代码解释:

- 首先我们通过语句 `sUriMatcher.match(uri) != DIARIES` 对传进来的 Uri 进行了判断, 如果这个 Uri 不是 DIARIES 类型的, 那么这个 Uri 就是一个非法的 Uri。
- `SQLiteDatabase db = mOpenHelper.getWritableDatabase()` 语句负责得到一个 SQLiteDatabase 的实例。
- `db.insert (DIARY_TABLE_NAME, DiaryColumns.BODY, values)` 语句负责插入一条记录到数据库中。
- 需要注意的是, `insert()` 返回的是一个 Uri, 而不是一个记录的 id, 所以, 我们还应该把记录的 id 构造成一个 Uri: `Uri diaryUri= ContentUris.withAppendedId(Diary.DiaryColumns.CONTENT_URI, rowId).withAppendedId()` 方法在下边的小知识当中详细介绍。

小知识

什么是 ContentUris?

ContentUris 是 content URI 的一个辅助类。它有两个方法很有用, 具体如下所示。

- `public static Uri withAppendedId (Uri contentUri, long id)`, 这个方法负责把 id 和 contentUri 连接成一个新的 Uri。比如在我们这个例子当中是这么使用的: `ContentUris.withAppendedId (Diary.DiaryColumns.CONTENT_URI, rowId)`。如果 rowId 为 100 的话, 那么现在的这个 Uri 的内容就是:

`content://com.ex09_2_contentprovider.diarycontentprovider/diaries/100。`

- `public static long parseId (Uri contentUri)`, 这个方法负责把 content URI 后边的 id 解析出来, 比如现在这个 content URI 是 `content://com.ex09_2_contentprovider.diarycontentprovider/diaries/100`, 那么这个函数的返回值就是 100。

(2) 现在来看删除方法: `delete()`。

具体实现代码如下所示:

```

@Override
public int delete(Uri uri, String where, String[] whereArgs) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String rowId = uri.getPathSegments().get(1);
    return db.delete(DIARY_TABLE_NAME, DiaryColumns._ID + "=" + rowId, null);
}

```

✎ 代码解释：

- rowId = uri.getPathSegments().get(1)负责得到 rowId 的值。

getPathSegments()方法得到一个 String 的 List，在我们例子当中 uri.getPathSegments().get(1)为 rowId，如果是 uri.getPathSegments().get(0)那值就为"diaries"。

- db.delete(DIARY_TABLE_NAME, DiaryColumns._ID + "=" + rowId, null)是标准的 SQLite 删除操作。第一个参数数据表的名字，第二个参数相当于 SQL 语句当中的 where 部分。

(3) 更新一条数据的方法：update()。

具体实现代码如下所示：

```

@Override
public int update(Uri uri, ContentValues values, String where,
String[] whereArgs) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String rowId = uri.getPathSegments().get(1);
    return db.update(DIARY_TABLE_NAME, values, DiaryColumns._ID + "="
+ rowId, null);
}

```

✎ 代码解释：

- 首先得到 SQLiteDatabase 的实例，然后得到 rowId，最后再调用 db.update(DIARY_TABLE_NAME, values, DiaryColumns._ID + "=" + rowId, null)语句执行更新工作。

(4) 查寻一条数据的方法：query()。

具体实现代码如下所示：

```

@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

    switch (sUriMatcher.match(uri)) {
        case DIARIES:
            qb.setTables(DIARY_TABLE_NAME);
            break;

        case DIARY_ID:
            qb.setTables(DIARY_TABLE_NAME);
            qb.appendWhere(DiaryColumns._ID + "="
+ uri.getPathSegments().get(1));
            break;

        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }

    String orderBy;
    if (TextUtils.isEmpty(sortOrder)) {
        orderBy = Diary.DiaryColumns.DEFAULT_SORT_ORDER;
    } else {
        orderBy = sortOrder;
    }
}

```

```

    }

    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
    Cursor c = qb.query(db, projection, selection, selectionArgs, null,
        null, null, orderBy);
    return c;
}

```

代码解释:

- **SQLiteQueryBuilder** 是一个构造 SQL 查询语句的辅助类。
- **sUriMatcher.match(uri)**, 根据返回值可以判断这次查询请求时, 它是请求全部数据还是某个 id 的数据。
 - 如果返回值是 **DIARIES**, 那么只需要执行 **qb.setTables (DIARY_TABLE_NAME)** 语句就可以了。
 - 如果返回值是 **DIARY_ID**, 那么还需要将 **where** 部分的参数设置进去, 代码为 **qb.appendWhere(DiaryColumns._ID + "=" + uri.getPathSegments().get(1))**。
- **SQLiteDatabase db = mOpenHelper.getReadableDatabase()**, 得到一个可读的 **SQLiteDatabase** 实例。
- **Cursor c = qb.query (db, projection, selection, selectionArgs, null,null, orderBy)** 语句, 这个查询类似于一个标准的 SQL 查询, 但是这个查询是 **SQLiteQueryBuilder** 来发起的, 而不是 **SQLiteDatabase** 直接发起的, 所以在参数方面略有不同。这个函数为 **query (SQLiteDatabase db, String[] projectionIn, String selection, String[] selectionArgs, String groupBy, String having, String sortOrder, String limit)** 下边将各个参数介绍一下。
 - 第一个参数为要查询的数据库实例。
 - 第二个参数是一个字符串数组, 里边的每一项代表了需要返回的列名。
 - 第三个参数相当于 SQL 语句中的 **where** 部分。
 - 第四个参数是一个字符串数组, 里边的每一项依次替代在第三个参数中出现的问号 (?)。
 - 第五个参数相当于 SQL 语句当中的 **groupby** 部分。
 - 第六个参数相当于 SQL 语句当中的 **having** 部分。
 - 第七个参数描述是怎么进行排序。
 - 第八个参数相当于 SQL 当中的 **limit** 部分, 控制返回的数据的个数。

在 **DiaryContentProvider** 类里边还有两个方法, 一个是 **getType (Uri uri)**, 另一个是 **getFormateCreateDate()**。后者根据时间得到一个我们特定格式的字符串, 比较简单。而前者是必须要重写的方法。下边看一下我们如何重写了 **getType** 方法, 具体代码如下所示:

```

public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
        case DIARIES:
            return DiaryColumns.CONTENT_TYPE;

        case DIARY_ID:
            return DiaryColumns.CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
}

```

✎ 代码解释：

- 此方法返回一个所给 Uri 的指定数据的 MIME 类型。它的返回值如果以 vnd.Android.cursor.item 开头，那么就代表这个 Uri 指定的是单条数据。如果是以 vnd.Android.cursor.dir 开头的话，那么说明这个 Uri 指定的是全部数据。

7. 第七步

在程序主界面单击 MENU 键，程序运行界面如图 8-33 所示。

在 ActivityMain 中执行的代码如下所示：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, MENU_ITEM_INSERT, 0, R.string.menu_insert);
    return true;
}
```

✎ 代码解释：

- 我们给菜单（menu）添加了一项，如图 8-33 所示。

单击添加日记按钮后进入如图 8-34 所示界面。



▲ 图 8-33 单击 MENU 键



▲ 图 8-34 新建日记的页面

图 8-34 所示的这个页面的布局和第 8 章前一个日记本程序里的布局完全一样，关于布局，读者可以参看第 8 章日记本程序例子的讲解。

下面来看一下在 ActivityDiaryEditor 中，是怎么处理两种不同情况进入日记本程序界面的。一种是通过新建日记进入此界面，另一种是经过编辑日记进入此日记运行界面。下面是在 ActivityDiaryEditor 程序中使用的 onCreate() 函数，其代码如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```



```

setTheme(Android.R.style.Theme_Black);
final Intent intent = getIntent();
final String action = intent.getAction();
setContentView(R.layout.diary_edit);
mTitleText = (EditText) findViewById(R.id.title);
mBodyText = (EditText) findViewById(R.id.body);
confirmButton = (Button) findViewById(R.id.confirm);

if (EDIT_DIARY_ACTION.equals(action)) { // 编辑日记
mState = STATE_EDIT;
mUri = intent.getData();
mCursor = managedQuery(mUri, PROJECTION, null, null, null);
mCursor.moveToFirst();
String title = mCursor.getString(1);
mTitleText.setTextKeepState(title);
String body = mCursor.getString(2);
mBodyText.setTextKeepState(body);
setResult(RESULT_OK, (new Intent()).setAction(mUri.toString()));
setTitle("编辑日记");
} else if (INSERT_DIARY_ACTION.equals(action)) { // 新建日记
mState = STATE_INSERT;
setTitle("新建日记");
} else {
Log.e(TAG, "no such action error");
finish();
return;
}

confirmButton.setOnClickListener(new View.OnClickListener() {
public void onClick(View view) {
if (mState == STATE_INSERT) {
insertDiary();
} else {
updateDiary();
}
Intent mIntent = new Intent();
setResult(RESULT_OK, mIntent);
finish();
}

});
}

```

代码解释:

- 通过 `getIntent()` 得到启动当前 Activity 的那个 Intent。
- 通过 `intent.getAction()` 得到这个 intent 的动作 (Action)。在此操作中, 当前的 Activity 是通过单击先前的新建日记的按钮进来的, 所以, 看一下 `ActivityMain` 当中的 `onOptionsItemSelected()` 函数, 了解一下动作 (Action) 是怎么设置的, 此代码如下所示:

```

Intent intent0 = new Intent(this, ActivityDiaryEditor.class);
intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION);
intent0.setData(getIntent().getData());
startActivity(intent0);

```

代码解释:

- 从上述代码可以看到, 通过语句 `intent0.setAction (ActivityDiaryEditor.INSERT_ DIARY_ACTION)` 设置的 action 是 `ActivityDiaryEditor.INSERT_DIARY_ACTION` 实现的。

- 在 ActivityDiaryEditor 中的 onCreate()函数中,当动作(Action)为 INSERT_DIARY_ACTION 时候,我们执行 mState = STATE_INSERT, 也就是标记当前的状态为插入状态。

- 如果当前的动作(Action)是 EDIT_DIARY_ACTION, 那么当前就是编辑状态: mState = STATE_EDIT。

当运行程序填充数据后单击确定按钮,执行 confirmButton 的单击监听器当中的 onClick()函数。这个函数根据不同的状态执行插入或者更新数据的操作。

按照现在的程序执行流程,单击确定按钮后,程序将执行插入操作,实现插入操作代码如下所示:

```
private void insertDiary() {
    String title = mTitleText.getText().toString();
    String body = mBodyText.getText().toString();
    ContentValues values = new ContentValues();
    values.put(Diary.DiaryColumns.CREATED, DiaryContentProvider
        .getFormatCreatedDate());
    values.put(Diary.DiaryColumns.TITLE, title);
    values.put(Diary.DiaryColumns.BODY, body);
    getContentResolver().insert(Diary.DiaryColumns.CONTENT_URI, values);
}
```

代码解释:

- 首先通过 getText()方法将编辑框中的数据都读出来。
- 将要插入的数据都放到一个 ContentValues 的实例当中。
- 调用 getContentResolver()得到当前应用的一个 ContentResolver 的实例。
- insert (Diary.DiaryColumns.CONTENT_URI, values) 语句为 ContentResolver 的插入方法,这个方法有两个参数,第一个参数是数据的 Uri, 第二个参数是包含要插入数据的 ContentValues 的实例。执行这条语句的最终会调用 DiaryContentProvider 中的 insert()方法。

当单击确定按钮后,程序运行出现如图 8-35 所示的界面。

8. 第八步

按方向键向下键将焦点移动到这条数据上,然后单击 MENU 键会出现如图 8-36 所示界面。

图 8-36 所示的这个界面和刚才讲到的单击 MENU 键出现的界面一样,下面看一下实现的代码,以便了解此界面是怎么生成的,具体实现代码如下所示:



▲ 图 8-35 已经插入了一条数据



▲ 图 8-36 单击 MENU 键界面

```

public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);
    final boolean haveItems = getListAdapter().getCount() > 0;
    if (haveItems) {
        if (getListView().getSelectedItemId() > 0) {
            menu.removeGroup(1);
            Uri uri = ContentUris.withAppendedId(getIntent().getData(),
                getSelectedItemId());
            Intent intent = new Intent(null, uri);
            menu.add(1, MENU_ITEM_EDIT, 1, "编辑内容").setIntent(intent);
            menu.add(1, MENU_ITEM_DELETE, 1, "删除当前日记");
        }

    } else {
        menu.removeGroup(1);
    }
    return true;
}

```

代码解释:

- 当前和这个 ListView 相关联的 ListAdapter 里边元素的个数大于零的时候 haveItems 为真，否则为假。
- menu.removeGroup(1)，如果 menu 里边有分组为 1 组的子项的话，那么就先全部删除。
- getIntent().getData()得到的 Uri 是 DiaryColumns.CONTENT_URI。
- 通过 ContentUris.withAppendedId(getIntent().getData(),getSelectedItemId()) 生成一个和 ListView 中获得焦点这一项的数据的 Uri。
- menu.add(1, MENU_ITEM_EDIT, 1, "编辑内容").setIntent(intent)，在 menu 当中增加了一项“编辑内容”，并且这一项和一个 intent 关联，这个 intent 主要用于携带数据，它里边携带的就是一个 Uri 的数据，在下边的 onOptionsItemSelected()函数当中会用到。
- menu.add(1, MENU_ITEM_DELETE, 1, "删除当前日记")，增加另外一项。
- 如果 haveItems 为假，也就是当前和这个 ListView 相关联的 ListAdapter 里边元素的个数为零，即数据显示在 ListView 上的时候，单击 MENU 按钮的话，如果 menu 当中还有第 1 分组的子项的话，就给删除了，实现语句为：menu.removeGroup(1)。

小知识

单击 MENU 按钮的时候，在 Activity 中回调的函数可能有两个。

第一个是 onOptionsItemSelected()，这个函数只在第一次在当前应用当中单击 MENU 键的时候回调，以后不再回调。

第二个是 onPrepareOptionsMenu()，这个函数在每次单击 MENU 键后显示 menu 的时候被系统回调，每次 menu 显示前都会回调此函数。我们一般根据条件改变 menu 显示的逻辑都放在这个函数里边。

当单击 MENU 键时出现 3 个按钮，单击其中的某一个按钮会触发 Android 系统回调 onOptionsItemSelected()函数，此函数实现代码如下所示：

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        // 插入一条数据
    }
}

```

```

case MENU_ITEM_INSERT:
Intent intent0 = new Intent(this, ActivityDiaryEditor.class);
intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION);
intent0.setData(getIntent().getData());
startActivity(intent0);
return true;
// 编辑当前数据内容
case MENU_ITEM_EDIT:
Intent intent = new Intent(this, ActivityDiaryEditor.class);
intent.setData(item.getIntent().getData());
intent.setAction(ActivityDiaryEditor.EDIT_DIARY_ACTION);
startActivity(intent);
return true;
// 删除当前数据
case MENU_ITEM_DELETE:
Uri uri = ContentUris.withAppendedId(getIntent().getData(),
getListView().getSelectedItemId());
getContentResolver().delete(uri, null, null);
renderListView();

}
return super.onOptionsItemSelected(item);
}

```

代码解释:

- 当用户单击添加新日记按钮后, 程序新建一个跳转 Activity 的 intent0, 并且设置了 Action 和 data, 这两个部分在程序跳转到 ActivityDiaryEditor 后会用到。
- 当用户单击编辑按钮后, 程序也新建了一个跳转 Activity 的 intent, 并且也设置了 Action 和 data。同样, 这两部分在程序跳转到 ActivityDiaryEditor 后会用到。需要注意的是, 通过 item.getIntent().getData()得到了所需要的 Uri。
- 当用户单击删除按钮后, 程序通过 ContentUris.withAppendedId(getIntent().getData(), getListView().getSelectedItemId())先得到需要删除数据的 Uri, 然后得到当前的 ContentResolver, 然后调用它的 delete 方法进行删除。
- 删除数据后, 调用 renderListView()函数对 ListView 进行及时刷新。

8.6 再学一招——网络存储

前面介绍的几种存储都是将数据存储在本地上, 除此之外, 还有一种存储(获取)数据的方式, 通过网络来实现数据的存储和获取, 下面看一个在 Android 上调用 WebService 的例子。

注意

在 Android 的早期版本中, 曾经支持过进行 XMPP Service 和 Web Service 的远程访问。Android SDK 1.0 以后的版本对它以前的 API 作了许多变更。Android 1.0 以上版本不再支持 XMPP Service, 而且访问 Web Service 的 API 全部变更。

1. 例子介绍

通过邮政编码查询该地区的天气预报, 以 POST 发送的方式发送请求到 webservicex.net 站点, 访问

WebService.websvcicex.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，网址为：

<http://www.websvcicex.net/WeatherForecast.asmx?WSDL>。

输入：美国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。

2. 实现步骤如下

(1) 如果需要访问外部网络，则需要在 AndroidManifest.xml 文件中加入如下代码申请权限许可：

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

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

```
private static final String SERVER_URL = "http://www.websvcicex.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());
}
```

 代码解释：

如上代码使用 Http 从 websvcicex 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.websvcicex.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>
```

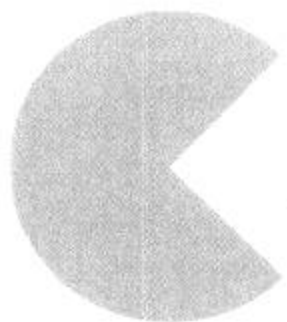


```
<WeatherData>
  <Day>Sunday, April 26, 2009</Day>
  <WeatherImage>http://forecast.weather.gov/images/wtf/few.jpg</WeatherImage>
  <MaxTemperatureF>89</MaxTemperatureF>
  <MinTemperatureF>60</MinTemperatureF>
  <MaxTemperatureC>32</MaxTemperatureC>
  <MinTemperatureC>16</MinTemperatureC>
</WeatherData>
...
</Details>
</WeatherForecasts>
```

这个例子演示了如何在 Android 中通过网络获取数据，掌握该类内容，开发者需要熟悉 `java.net.*`，`Android.net.*` 这两个包的内容，在这就不赘述了，请读者参阅相关文档。

8.7 本章小结

本章介绍了如何在 Android 中实现数据的存储和获取，详细介绍了轻量级的 `SharedPreferences`，比较传统的文件存储和数据库存储，也介绍了 Android 特有的 `ContentProvider` 方式，最后以一个获取天气预报的例子，演示了如何通过网络获取 `WebService` 的数据。通过本章的介绍，读者应该比较全面地了解了 Android 中的存储数据的方式，至于在应用程序中使用哪一种数据存储方式取决于开发者的具体需求，通过这章的介绍，读者将会做出合适的选择。



第9章 我来“广播”你的 “意图”——Intent 和 Broadcast 面对面

学习目标

- 深入了解 Intent; ☐
- 学习如何在 Android 中广播 Intent; ☐
- 学习如何在 Android 监听广播。 ☐

9.1 Android 应用程序的核心——Intent

Intent 可以说是 Android 里边的灵魂，程序跳转和传递数据的时候基本上就是靠 Intent 了，我们在前边的例子当中不止一次地用到了 Intent，也介绍过很多 Intent 的知识，这一章中我们将集中对 Intent 做一个总结。

另外在涉及异步操作或者需要做通知的时候，Android 也给我们提供了非常好的广播机制，在这一章中我们也会对广播机制做一个介绍和总结。

9.1.1 Intent 基础

Intent 在 Android 应用中是相当重要的，理解 Intent 对应用编程很有帮助。在 Android 的官方 API 里边对 Intent 是这样定义的：An intent is an abstract description of an operation to be performed。一个 Intent 就是一次对将要执行的操作的抽象描述。具体有以下 3 种形式：

- 通过 startActivity 方法来启动一个新的 Activity；
- 通过 broadcast Intent 机制可以将一个 Intent 发送给任何对这个 Intent 感兴趣的 BroadcastReceiver。
- 通过 startService (Intent) 或 bindService (Intent, ServiceConnection, int) 来和后台的 Service 进行交互。

关于广播 (Broadcast) 和服务 (Service) 的部分我们会放到后续的部分进行详细的介绍。现在我们首先来看一下如何通过 startActivity 方法来启动一个新的 Activity。

9.1.2 用 Intent 启动一个新的 Activity

Intent 最常用的用途就是连接一个应用当中的各个 Activity，如果我们把 Activity 比作积木的话，

那么 Intent 就好像是胶水，把不同的积木粘起来，构成我们搭建的房子。在程序当中如果要启动一个 Activity 的话，通常会调用 startActivity 方法，并把 Intent 作为参数传进去，代码如下所示：

```
startActivity(myIntent);
```

这个 Intent 或者指定了一个 Activity，或者里边只包含了选定 Activity 的信息，但是具体启动哪个 Activity 是由系统去决定的，Android 的系统负责挑选一个最满足匹配挑选条件的 Activity。

1. 启动一个特定的 Activity

在前边的例子当中我们不止一次的启动特定的 Activity，启动一个特定的 Activity 是非常简单的，只需要将当前的 Context 和这个 Activity 的 class 作为参数构造一个 Intent，然后再将这个 Intent 作为参数传递给 startActivity 方法即可，示例代码如下：

```
Intent intent = new Intent(CurrentActivity.this, OtherActivity.class);
startActivity(intent);
```

当执行了这个 startActivity 的方法后，一个新的 Activity，比如就是我们上边的 OtherActivity 将被创建，并且移到整个 Activity 的堆栈顶部。

2. 启动一个未指明的 Activity

在程序当中很多时候我们并不去关心启动哪个 Activity，而是将我们想启动哪个 Activity 的描述信息放置到 Intent 里边。然后让系统去寻找和匹配合适的 Activity 来启动，这样做是非常有好处的。比如对于第三方的 Activity，它只需要描述自己在什么情况下被执行，如果我们启动 Activity 的描述信息正好和第三方 Activity 的描述信息相匹配的话，那么这个第三方的 Activity 就被启动了。

例如，下边的一段代码：

```
Intent intent = new Intent(Intent.ACTION_DIAL,
    Uri.parse("tel:115-1345"));
startActivity(intent);
```

我们并没有指定哪个 Activity 去执行，只是把我们对启动 Activity 的描述信息放到 Intent 当中。那么当执行 startActivity 方法后，Android 系统就会去寻找，到底哪个 Activity 最合适。最后它根据描述信息找到处理电话的 Activity，然后启动，执行。

3. 处理一个 Activity 的返回值

前边我们介绍了 startActivity 的方法来启动 Activity，这种方法启动的 Activity 执行完后，不会返回数据给前一个 Activity。所以 Android 提供了另外一种方法来处理这种情况。

startActivityForResult 方法用于启动一个 Activity 并且新的 Activity 执行完后需要返回到启动它的 Activity 来执行回调函数。

比如我们用以下代码来启动一个子 Activity，并且当这个子 Activity 返回的时候，我们可以处理子 Activity 返回的数据，代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    listener1 = new OnClickListener() {
        public void onClick(View v) {
            Intent intent1 = new Intent(ActivityMain.this, Activity1.class);
            intent1.putExtra("activityMain", "数据来自 activityMain");
```

```

startActivityForResult(intent1, REQUEST_CODE);
}
};
listener2 = new OnClickListener() {
public void onClick(View v) {
setTitle("这是在 ActivityMain");
Intent intent2 = new Intent(ActivityMain.this, Activity2.class);
startActivity(intent2);

}
};
setContentView(R.layout.main);
button1 = (Button) findViewById(R.id.button1);
button1.setOnClickListener(listener1);
button2 = (Button) findViewById(R.id.button2);
button2.setOnClickListener(listener2);
setTitle("ActivityMain");
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
if (requestCode == REQUEST_CODE) {
if (resultCode == RESULT_CANCELED)
setTitle("取消");
else if (resultCode == RESULT_OK) {
String temp=null;
Bundle extras = data.getExtras();
if (extras != null) {
temp = extras.getString("store");
}
setTitle(temp);
}
}
}
}
}

```

❧ 代码解释:

- 我们通过 `startActivityForResult(intent1, REQUEST_CODE)` 来启动了 `Activity1`。
- 一般如果我们通过调用子 `Activity` 的方式去启动一个 `Activity` 意味着我们想获取子 `Activity` 的返回值,而我们是通过重写当前 `Activity` 当中的 `onActivityResult(int requestCode, int resultCode, Intent data)` 方法来处理这些返回值的。当子 `Activity` 执行完毕的时候,Android 系统会自动会调用这个方法。

接下来我们看一下在 `Activity1` 中是如何将值返回的,代码如下所示:

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity1);
    listener1 = new OnClickListener() {
public void onClick(View v) {
Bundle bundle = new Bundle();
bundle.putString("store", "数据来自 Activity1");
Intent mIntent = new Intent();
mIntent.putExtras(bundle);
setResult(RESULT_OK, mIntent);
finish();
}
}
}

```

```
};
button1 = (Button) findViewById(R.id.button3);
button1.setOnClickListener(listener1);
String data=null;
Bundle extras = getIntent().getExtras();
    if (extras != null) {
        data = extras.getString("activityMain");
    }
setTitle("现在是在 Activity1 里:"+data);
}
```

代码解释:

- 上边代码中最重要的一行代码是: `setResult(RESULT_OK, mIntent)`, 这行代码设置了 Activity1 执行结束时候的返回值。所有的返回值都是放置在一个 intent 里边, 通过 intent 进行传递的。
 - 当执行完 `finish()` 后, Activity1 执行结束, 并且将设的返回值返回给调用它的父 Activity。
- 看完上边的两段代码后也许读者朋友们仍然有疑惑, 上边出现了 `requestCode`、`resultCode` 等, 这些东西是什么, 怎么来理解? 下边我们就来详细了解一下。
- `startActivityForResult (Intent intent, int requestCode)`。
 - 第一个参数是一个 Intent, 这个 Intent 包含了即将启动的 Activity 的信息。
 - 第二个参数是一个整数, 从字面上理解那么就是请求码。这个请求码将会在 `onActivityResult` 方法里边被返回。你可以通过不同的请求码去启动一个 Activity, 这样就可以在 `onActivityResult` 中通过这些不同的请求码来区分不同时候的启动, 从而做出不同的处理。
 - `onActivityResult (int requestCode, int resultCode, Intent data)`。
 - 第一个参数我们在上边已经介绍了, 就是启动新 Activity 时候的请求码。
 - 第二个参数是结果码, 这个参数其实是在子 Activity 的 `setResult (int resultCode, Intent data)` 方法里边设置的。通过不同的结果码, 我们可以了解子 Activity 操作结果的一些信息。比如结果码最经常的有两个, 一个是 `RESULT_OK`, 它代表子 Activity 的操作成功。还有一个是 `RESULT_CANCELED`, 表示子 Activity 里边的操作取消掉了, 或者没有成功。

9.1.3 Intent 详细讲解

在 Android 参考文档中, 对 Intent 的定义是执行某操作的一个抽象描述 (确实很抽象)。我们先来看看这里的抽象描述, 到底描述了什么。

一般, 一个 Intent 里边对某操作的抽象描述包含下边几个部分。

- 对执行动作的描述: 操作 (Action)。
- 对于这次动作相关联的数据进行描述: 数据 (data)。
- 对数据类型的描述: 数据类型 (type)。
- 对执行动作的附加信息进行描述: 类别 (category)。
- 其他一切附加信息的描述: 附件信息 (extras)。
- 对目标组件的描述: 目标组件 (component)。

1. Android 系统自带的 Action 介绍

Android 系统自定义了很多标准的 Action。如 `ACTION_MAIN`, `ACTION_VIEW` 等, 下面我们

就介绍一些比较常见的，也是经常用的标准动作。

- **ACTION_MAIN**，它的值我们比较熟悉："android.intent.action.MAIN"，这个值我们在每个 AndroidManifest.xml 文档中都可以看到。它标记当前 Activity 作为一个程序的入口。
- **ACTION_VIEW**，将数据显示给用户。**ACTION_VIEW** 通常和特定的 data 相配合使用，用于给用户显示数据。
- **ACTION_DIAL**，用于描述给用户打电话的动作，通过和 data 配合使用将会触发给特定 data 的用户打电话。
- **ACTION_PICK**，从特定的一组数据中进行选择数据操作。
- **ACTION_EDIT**，编辑特定的数据。
- **ACTION_DELETE**，删除特定的数据。

2. 与动作(Action)相关联的数据(Data)介绍

Android 中采用指向数据的一个 URI 来表示 data，例如，在联系人应用中，一个指向某联系人的 URI 可能为：content://contacts/1。

这种 URI 表示，通过 ContentURI 这个类来描述，具体可以参考 android.net.ContentURI 类的文档。以联系人应用为例，以下是一些 action / data 对，及其它们要表达的意图：

- **VIEW_ACTION** content://contacts/1 显示标识符为"1"的联系人的详细信息；
- **EDIT_ACTION** content://contacts/1 编辑标识符为"1"的联系人的详细信息；
- **VIEW_ACTION** content://contacts/ 显示所有联系人的列表；
- **PICK_ACTION** content://contacts/ 显示所有联系人的列表，并且允许用户在列表中选择一个人，然后把这个人返回给父 Activity，例如，电子邮件客户端可以使用这个 Intent，要求用户在联系人列表中选择一个人。

3. Intent 中的其他属性介绍

Action 中大多数的情况下只包含上边介绍的 Action 和 Data 属性，但是有时候还会包括下边的属性。

- **类别 (category)**，它为执行动作的附加信息。例如，LAUNCHER_CATEGORY 表示 Intent 的接受者应该在 Launcher 中作为顶级应用出现；而 ALTERNATIVE_CATEGORY 表示当前的 Intent 是一系列的可选动作中的一个。
- **数据类型 (type)**，显式指定 Intent 的数据类型 (MIME)。一般 Intent 的数据类型能够根据数据本身进行判定，但是通过设置这个属性，可以强制采用显式指定的类型而不再进行推导。
- **组件 (component)**，指定 Intent 的目标组件的类名称。通常 Android 会根据 Intent 中包含的其他属性的信息，如 action、data/type、category 进行查找，最终找到一个与之匹配的目标组件。但是，如果 component 这个属性有指定的话，将直接使用它指定的组件，而不再执行上述查找过程。指定了这个属性以后，Intent 的其他所有属性都是可选的。
- **附加信息 (extras)**，是其他所有附加信息的集合。使用 extras 可以为组件提供扩展信息，例如，如果要执行“发送电子邮件”这个动作，可以将电子邮件的标题、正文等保存在 extras 里，传给电子邮件发送组件。

总之，action、data/type、category 和 extras 一起形成了一种级别的语言规范。这种语言使系统能够理解诸如“查看某联系人的详细信息”或“给某人打电话”之类的短语。随着应用不断地加入到系统中，Android 系统可以添加新的 action、data/type、category 来扩展这种语言。当然最受益的还是应用本身，我们可以利用这套语言机制来处理不同的动作和数据。

9.1.4 Android 解析 Intent 实现

在应用中，我们可以以两种形式来使用 Intent。

- 直接 Intent：指定了 component 属性的 Intent（调用 `setComponent(ComponentName)` 或者 `setClass(Context, Class)` 来指定）。通过指定具体的组件类，通知应用启动对应的组件。
- 间接 Intent：没有指定 component 属性的 Intent。这些 Intent 需要包含足够的信息，这样系统才能根据这些信息，在所有的可用组件中，确定满足此 Intent 的组件。

对于直接 Intent，Android 不需要去做解析，因为目标组件已经很明确，Android 需要解析的是那些间接 Intent，通过解析，将 Intent 映射给可以处理此 Intent 的 Activity、Receiver 或 Service。Intent 解析机制主要是通过查找已注册在 AndroidManifest.xml 中的所有 IntentFilter 及其中定义的 Intent，最终找到匹配的 Intent。

在这个解析过程中，Android 是通过 Intent 的 action、type、category 这 3 个属性来进行判断的，判断方法如下。

- 如果 Intent 指明定了 action，则目标组件的 IntentFilter 的 action 列表中就必须包含有这个 action，否则不能匹配。
- 如果 Intent 没有提供 type，系统将从 data 中得到数据类型。和 action 一样，目标组件的数据类型列表中必须包含 Intent 的数据类型，否则不能匹配。
- 如果 Intent 中的数据不是具体内容而是 URI 形式的话，则将根据 Intent 中数据的 scheme（如 http: 或者 mailto:）进行匹配。同上，Intent 的 scheme 必须出现在目标组件的 scheme 列表中。
- 如果 Intent 指定了一个或多个 category，这些类别必须全部出现在组件的类别列表中。如 Intent 中包含了两个类别：LAUNCHER_CATEGORY 和 ALTERNATIVE_CATEGORY，解析得到的目标组件必须至少包含这两个类别。

我们以书中豆瓣客户端中的 AndroidManifest.xml 为例子，进行示例介绍，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.douban.android" android:versionCode="1"
android:versionName="1.0.0">
<uses-permission android:name="android.permission.INTERNET" />
<application android:icon="@drawable/icon"
android:label="@string/app_name">
<activity android:name=".ActivityMain"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category
android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".ActivityAuth"
```

```

    android:label="@string/app_name">
</activity>
<activity android:name=".ActivityShowSaying"
    android:label="@string/app_name">
</activity>

<activity android:name=".ActivityAuthSuccess"
    android:label="@string/app_name">
<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="haiyang" />
</intent-filter>
</activity>
</application>
</manifest>

```

代码解释：

上边例子中的第一个 Activity 是 ActivityMain, 它是应用的入口, 动作(Action)为"android.intent.action.MAIN", ActivityMain 为这个应用中的第一个 Activity。类别(category)为"android.intent.category.LAUNCHER"表示当程序启动的时候, ActivityMain 会被放在启动 Activity 堆栈的顶层。还有一个 Activity 是 ActivityAuthSuccess, 动作(Action)为"android.intent.action.VIEW"。

类别(category)分别为缺省类型"android.intent.category.DEFAULT", android.intent.category.BROWSABLE" 类型, 此类型表明 ActivityAuthSuccess 可以通过浏览器启动。需要注意的是, 我们在这里定义了数据类型的匹配模式为 scheme="haiyang", 也就是说当链接为 "haiyang://" 模式的时候可以触发 ActivityAuthSuccess 这个 Activity 来处理。

9.2 用广播告诉你——利用 Intent 来广播[Broadcast]事件

在前边学习到 Intent 可以用来启动一个新的 Activity, 但是 Intent 的作用远远不止这些。Intent 还有一个重要的机制就是作为不同进程间传递数据和事件的媒介。

通常我们自己的应用或者是 Android 系统本身在某些事件来临的时候会将 Intent 广播出去, 而注册的 Broadcast Receiver 可以监听到这些 Intent, 并且可以获得保存在 Intent 里边的数据。

例如, 在电池电量发生变化、网络连接发生变化的时候或者来电、来短信的时候, Android 系统都会将相关的 Intent 进行广播。如果注册了针对这些事件的 Broadcast Receiver, 那么就可以处理这些事件。

9.2.1 实现 Android 中的广播事件

程序主动广播 Intent 是比较简单。在程序当中只需要构造好一个 Intent, 然后调用 sendBroadcast 进行广播即可, 代码如下所示:

```

public static final String NEW_BROADCAST = "com.eoeandroid.action.NEW_BROADCAST";
Intent intent = new Intent(NEW_BROADCAST);
intent.putExtra("data1", someData);
intent.putExtra("data2", someData);
sendBroadcast(intent);

```

9.2.2 Broadcast Receiver 介绍

不管是系统广播的 Intent 还是其他程序广播的 Intent，如果想接收并且对它处理的话，需要注册一个 Broadcast Receiver，并且一般的要给注册的这个 Broadcast Receiver 设置一个 Intent Filter 来制定当前的 Broadcast Receiver 是对那些 Intent 进行监听。

1. 如何实现一个 Broadcast Receiver

我们可以通过继承 BroadcastReceiver 类，并重写这个类当中的 onReceive 方法来实现。代码如下所示：

```
public class EoeAndroidReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
    }
}
```

在 onReceive 方法里边最好不要有执行超过 5s 的代码，如果这样的话，Android 系统就会弹出一个超时对话框。我们建议如果您有比较耗时的方法，可以把这些方法写在一个线程里边，单独来执行。

我们注册的 BroadcastReceiver 并非一直在后台运行，而是当事件或者相关的 Intent 来临的时候，才会被系统调用。有了这套广播的机制，我们就可以写一些和事件处理相关的程序了。

2. 如何注册和注销 Broadcast Receiver

实现了自己的 BroadcastReceiver 后千万别忘了进行注册，笔者刚开始写代码的时候就经常犯这样的错误，明明写好了自己的 BroadcastReceiver，但是却无法对事件进行处理，后来经过仔细排查，才发现没有将实现的 BroadcastReceiver 注册到系统里边去。

我们可以通过两种方式对 BroadcastReceiver 进行注册。

第一种方法是在 AndroidManifest.xml 文件当中进行注册，这种方法是推荐的，而且也是最常用的。比如我们将上边实现的这个 BroadcastReceiver 注册到系统当中去，实现代码如下：

```
<receiver android:name="EoeAndroidReceiver">
    <intent-filter>
        <action android:name="com.eoeandroid.action.NEW_BROADCAST"/>
    </intent-filter>
</receiver>
```

第二种方式是在代码当中直接进行注册，这种方式虽然不是很推荐，但是在有些时候用起来比较灵活，代码如下所示：

```
IntentFilter filter = new IntentFilter(NEW_BROADCAST);
EoeAndroidReceiver eoeandroidReceiver = new EoeAndroidReceiver ();
registerReceiver(eoeandroidReceiver, filter);
```

如果想将一个已经注册的 BroadcastReceiver 注销的话，可以用下边的代码：

```
unregisterReceiver(eoeandroidReceiver);
```

9.3 应用实例详解

为了更好地学习上边介绍的知识，我们准备了一个实例和大家分享。通过这个实例的学习，帮

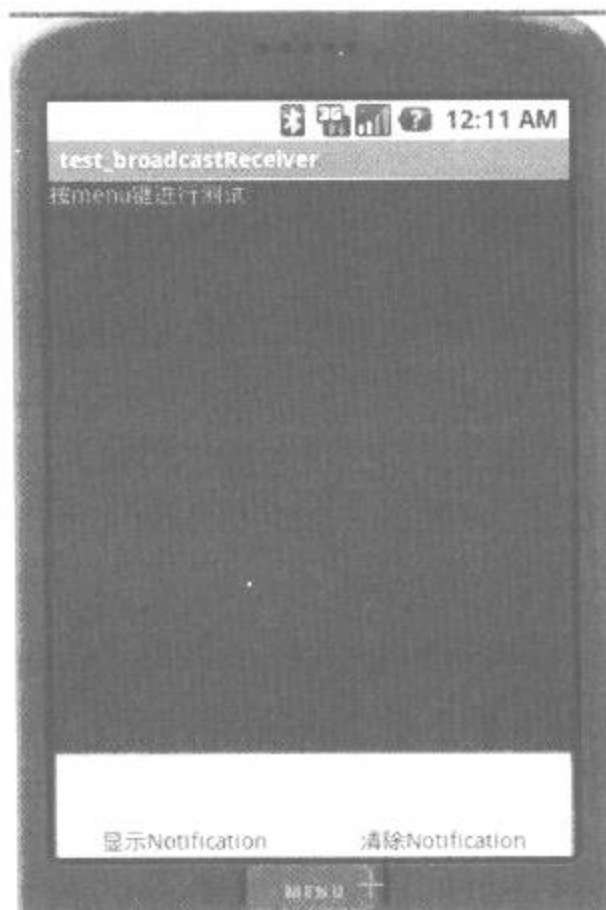
帮助大家活学活用 BroadcastReceiver 和 Android 当中的广播机制。首先看一下这个例子是怎么操作的。

9.3.1 程序操作演示

- (1) 应用运行后主界面如图 9-1 所示。
- (2) 单击 MENU 键后，显示如图 9-2 所示的界面。



▲ 图 9-1 程序主界面



▲ 图 9-2 单击 MENU 后的显示界面

- (3) 单击 MENU 当中的第一项，显示如图 9-3 所示的界面。
- (4) 单击 MENU 中的第二项后，状态栏的 Notification 消失，界面返回到如图 9-4 所示的界面。



▲ 图 9-3 单击 MENU 的第一项后的显示界面



▲ 图 9-4 单击 MENU 第二项后的显示界面

9.3.2 实例编程实现

1. 第一步

在 Eclipse 中打开 ex09_BroadcastReceiver 项目，具体操作步骤如下所示。

- (1) 新建一个项目。依次单击 File→New→Android Project 项。
 - (2) 在新建项目的对话框中, 选择 Create project from existing source 项。
 - (3) 单击浏览, 找到 ex09_BroadcastReceiver 项目, 然后单击确定。
- 程序目录结构如图 9-5 所示。

2. 第二步

首先我们先看一下 AndroidManifest.xml 文件, 代码如下所示:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.eoeandroid.broadcastReceiver">
<application android:icon="@drawable/icon">
<activity android:name=".ActivityMain"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category
android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<receiver android:name="EoeAndroidReceiver1">
<intent-filter>
<action
android:name="com.eoeandroid.action.NEW_BROADCAST_1"/>
</intent-filter>
</receiver>
<receiver android:name="EoeAndroidReceiver2">
<intent-filter>
<action
android:name="com.eoeandroid.action.NEW_BROADCAST_2"/>
</intent-filter>
</receiver>
</application>
</manifest>
```

代码解释:

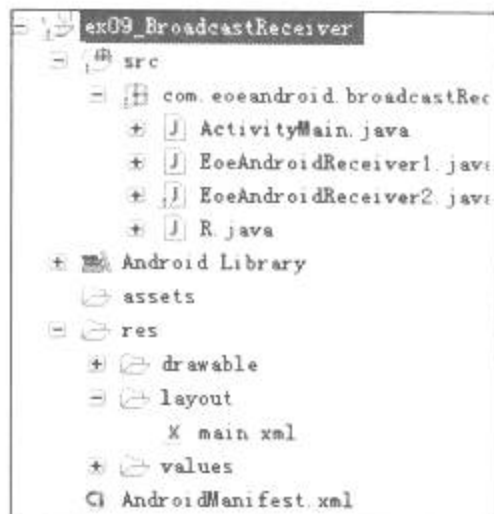
在这个文件里边我们注册了两个 BroadcastReceiver, 一个是 EoeAndroidReceiver1, 另一个是 EoeAndroidReceiver2。前者可以接收 action 为 "com.eoeandroid.action.NEW_BROADCAST_1" 的 Intent, 后者可以接收 action 为 "com.eoeandroid.action.NEW_BROADCAST_2" 的 Intent。

3. 第三步

现在我们来看一下 ActivityMain 的代码, 在 ActivityMain 里边我们对 Intent 进行了广播。

```
package com.eoeandroid.broadcastReceiver;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
```



▲ 图 9-5 程序目录结构图

```

/**
 *
 * @author jinyan
 */
public class ActivityMain extends Activity {

    public static final int ITEM0 = Menu.FIRST;
    public static final int ITEM1 = Menu.FIRST + 1;

    static final String ACTION_1 = "com.eoeandroid.action.NEW_BROADCAST_1";
    static final String ACTION_2 = "com.eoeandroid.action.NEW_BROADCAST_2";

    @Override
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
    }

    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, ITEM0, 0, "显示 Notification");
        menu.add(0, ITEM1, 0, "清除 Notification");
        menu.findItem(ITEM1);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case ITEM0:
                actionClickMenuItem1();
                break;
            case ITEM1:
                actionClickMenuItem2();
                break;
        }
        return true;
    }

    private void actionClickMenuItem1() {
        Intent intent = new Intent(ACTION_1);
        sendBroadcast(intent);
    }

    private void actionClickMenuItem2() {
        Intent intent = new Intent(ACTION_2);
        sendBroadcast(intent);
    }
}

```

代码解释:

- 通过 new Intent (ACTION_1), 我们新建了 Action 为 ACTION_1 的 Intent。
- 通过 sendBroadcast (intent), 我们将这个 Intent 进行了广播。

4. 第四步

当单击 MENU 的第一项后, 程序会执行到 EoeAndroidReceiver1 里边, 我们特地在 EoeAndroid

Receiver1 里边的 onReceive 里边将一个 Notification 显示在了状态栏中，代码如下所示：

```
package com.eoeandroid.broadcastReceiver;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class EoeAndroidReceiver1 extends BroadcastReceiver {
    Context context;
    public static int NOTIFICATION_ID = 21321;

    @Override
    public void onReceive(Context context, Intent intent) {
        this.context = context;
        showNotification();
    }

    private void showNotification() {

        NotificationManager notificationManager = (NotificationManager) context
            .getSystemService(android.content.Context.NOTIFICATION_SERVICE);
        Notification notification = new Notification(R.drawable.icon,
            "在 EoeAndroidReceiver1 中", System.currentTimeMillis());

        PendingIntent contentIntent = PendingIntent.getActivity(context, 0,
            new Intent(context, ActivityMain.class), 0);
        notification.setLatestEventInfo(context, "在 EoeAndroidReceiver1 中", null,
            contentIntent);
        notificationManager.notify(NOTIFICATION_ID, notification);
    }
}
```

❏ 代码解释：

- showNotification() 负责显示一个 Notification。关于 Notification 我们在第 7 章已经有了介绍，如果有遗忘的话，可以回头查阅一下。

单击第二个按钮后，我们的程序开始广播，这个广播被 EoeAndroidReceiver2 所截获，然后就开始执行 EoeAndroidReceiver2 里的 onReceive 方法，代码如下所示：

```
package com.eoeandroid.broadcastReceiver;

import android.app.NotificationManager;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

public class EoeAndroidReceiver2 extends BroadcastReceiver {
    Context context;

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub
        this.context = context;
        DeleteNotification();
    }
}
```

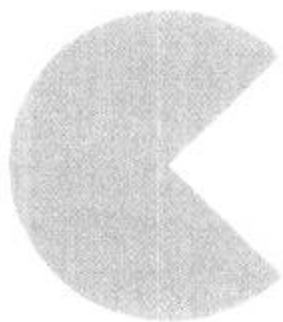
```
private void DeleteNotification() {  
  
    NotificationManager notificationManager = (NotificationManager) context  
        .getSystemService(android.content.Context.NOTIFICATION_SERVICE);  
    notificationManager.cancel(EoeAndroidReceiver1.NOTIFICATION_ID);  
}  
  
}
```

✎ 代码解释：

- DeleteNotification()方法负责将刚才生成的 Notification 从状态栏中删除。需要注意的是，每一个 Notification 都有一个惟一的 id 进行标识，而在程序中的这个 Notification 的 id 为 EoeAndroidReceiver1.NOTIFICATION_ID。

9.4 本章小结

在这一章里我们对 Intent 进行了集中的总结，通过学习 Intent 的作用到深入分析 Intent 的构成，让我们对 Intent 有可更加深刻的理解。通过 Intent 可以启动一个新的 Activity 或者一个 Service。也可以通过广播 Intent 进而在进程间进行事件的传递。通过注册自己的 BroadcastReceiver 可以监听系统或者其他应用发送的事件，并可以对这些事件进行处理。最后通过一个 BroadcastReceiver 的例子演示了 BroadcastReceiver 和广播机制在代码中的使用方法和技巧。



第 10 章 一切为用户服务——

Service 应用实例

学习目标

- 了解什么是 Service; ☐
- 如何使用 Service; ☐
- 学习 Service 的生命周期; ☐
- 掌握实例一：定时提醒; ☐
- 掌握实例二：音乐播放。 ☐

10.1 认识 Service

前面几章讲解了 Activity，本章将要接触一个新的概念——Service。我们先介绍一下在 Android 中，Service 是个什么概念，进而用实例介绍几种使用 Service 的方法；然后会一起探讨 Service 的生命周期，最后使用两个例子详细演示 Service 的使用。

从字面理解 Service 的意义是“服务”，它是在后台运行，不可交互的一个东西。它跟前面讲解的 Activity 级别差不多，但是它不能自己运行，需要通过某一个 Activity 或者其他 Context 对象来调用，如 Context.startService() 和 Context.bindService()两种方式启动 Service。上面两种启动 Service 的方式有所不同，我们会在后面详细说明，现在只要记得它们是不同的就可以了。另外，这里还需要说明的是，如果在 Service 的 onCreate 或者 onStart 做一些很耗时的动作，最好是启动一个新线程来运行这个 Service，因为，如果 Service 是运行在主线程中，会影响到程序的 UI 操作或者阻塞主线程中的其他事情。

那么，什么时候需要使用 Service 呢？这要看程序的应用场景，例如，播放多媒体的时候用户启动了其他 Activity，这个时候程序要在后台继续播放；如检测 SD 卡上文件的变化；或在后台记录用户的地理信息位置的改变等，总之服务嘛，总是藏在后面。比较直观的一个例子就是，一个正在从播放列表中播放歌曲的媒体播放器，在一个媒体播放器的应用中，应该会有多个 Activity，让使用者可以选择歌曲并播放歌曲。然而，音乐播放这个功能并没有对应的 Activity，因为使用者会认为在导航到其他屏幕时音乐应该还在播放的。在这个例子中，媒体播放器这个 Activity 会使用 Context.startService()来启动一个 Service，从而可以在后台保持音乐的播放。同时，系统也将保持这个 Service 一直执行，直到这个 Service 运行结束，我们还可以进行暂停、重播等操作。

10.2 使用 Service

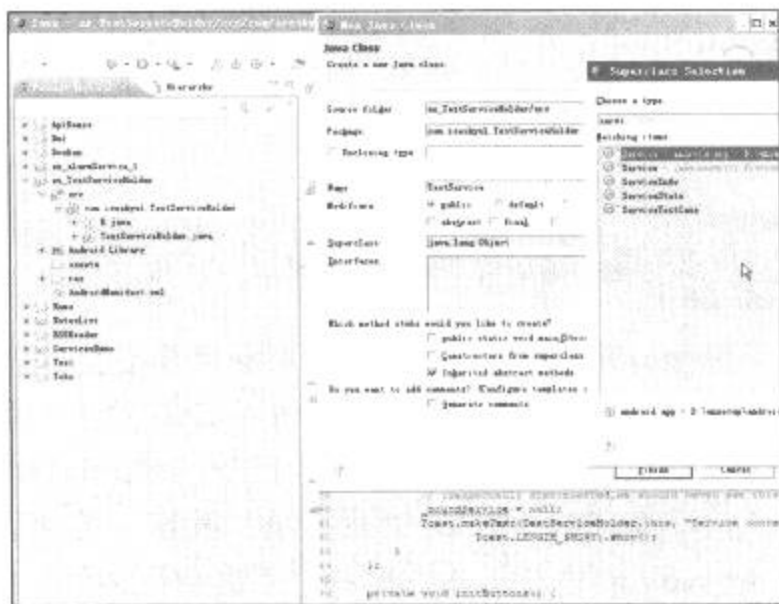
前面一小节讲解了什么是 Service，接下来会用代码来说明一下怎么使用 Service，这里要讲的是 Local Service，也就是用户自己的一个 Service，也可以操作别的应用程序的 Service（如果它允许你那么去做的话），这就涉及一个比较麻烦的技术 interprocess communication（IPC），其提供了在不同的进程之间通信的机制，这个以后再说，通常情况下 Local 的就够用啦。

为了演示 Service 的使用，下面将用一个例子来说明，首先，我们创建一个项目，名字为 TestServiceHolder，操作和前面做法一样，如图 10-1 所示。

然后，还需要创建一个 Service，跟 Activity 一样，首先要写继承自 Android.app.Service 的一个类，在这里命名为 TestService，如图 10-2 所示。



▲ 图 10-1 创建项目



▲ 图 10-2 创建 Service

紧接着，需要在布局文件中添加 4 个按钮，分别用来演示 start_Service（开始服务）、stop_Service（停止服务）、bind_Service（绑定服务）和 unbind_Service（断开服务）操作。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">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

    <Button android:id="@+id/start_Service" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Start Service" />

    <Button android:id="@+id/stop_Service" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Stop Service" />
```

```

<Button Android:id="@+id/bind_Service" Android:layout_width="fill_parent"
Android:layout_height="wrap_content" Android:text="Bind Service" />

<Button Android:id="@+id/unbind_Service" Android:layout_width="fill_parent"
Android:layout_height="wrap_content" Android:text="Unbind Service" />

</LinearLayout>

```

❏ 代码解释:

上述代码中,采用 `LinearLayout` 布局形式,并放置了 4 个按钮,分别对应我们需要测试的 4 种操作。然后需要在项目的配置文件 `AndroidManifest.xml` 中添加对前面创建的 `Service` 的引用,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.iceskysl.TestServiceHolder"
Android:versionCode="1"
Android:versionName="1.0.0">
<application Android:icon="@drawable/icon" Android:label="@string/app_name">
<activity Android:name=".TestServiceHolder"
Android:label="@string/app_name">
<intent-filter>
<action Android:name="Android.intent.action.MAIN" />
<category Android:name="Android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

<Service Android:enabled="true" Android:name=".TestService" />

</application>
</manifest>

```

❏ 代码解释:

上述代码中,我们添加了一行“`<Service Android:enabled="true" Android:name=".TestService" />`”用来申明这个服务可以使用了。

然后我们来写 `TestService` 的代码,实现代码如下:

```

package com.iceskysl.TestServiceHolder;

import Android.app.Notification;
import Android.app.NotificationManager;
import Android.app.PendingIntent;
import Android.app.Service;
import Android.content.Intent;
import Android.os.Binder;
import Android.os.IBinder;
import Android.util.Log;

public class TestService extends Service {

    private static final String TAG = "TestService";
    private NotificationManager _nm;

    @Override
    public IBinder onBind(Intent i) {
        Log.e(TAG, "=====> TestService.onBind");
        return null;
    }

    public class LocalBinder extends Binder {
        TestService getService() {
            return TestService.this;
        }
    }
}

```

```

@Override
public boolean onUnbind(Intent i) {
    Log.e(TAG, "=====> TestService.onUnbind");
    return false;
}

@Override
public void onRebind(Intent i) {
    Log.e(TAG, "=====> TestService.onRebind");
}

@Override
public void onCreate() {
    Log.e(TAG, "=====> TestService.onCreate");
    _nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    showNotification();
}

@Override
public void onStart(Intent intent, int startId) {
    Log.e(TAG, "=====> TestService.onStart");
}

@Override
public void onDestroy() {
    _nm.cancel(R.string.Service_started);
    Log.e(TAG, "=====> TestService.onDestroy");
}

private void showNotification() {
    Notification notification = new Notification(R.drawable.face_1,
        "Service started", System.currentTimeMillis());

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        new Intent(this, TestServiceHolder.class), 0);

    // must set this for content view, or will throw a exception
    notification.setLatestEventInfo(this, "Test Service",
        "Service started", contentIntent);

    _nm.notify(R.string.Service_started, notification);
}
}

```

代码解释:

上述代码中，创建了继承与 `Android.app.Service` 的 `TestService`，并重写了其 `onStart`、`onDestroy` 等方法，这里主要是通过输入 LOG 的方式确定被调用的方法。另外，其用到 `Notification` 是为了明显地表明 `Service` 存活的状态，这样看上去直观一点，更多关于 `Notification` 的内容在以后的例子部分讲解，现在就知道怎么使用就好了。

```

@Override
public void onCreate() {
    Log.e(TAG, "=====> TestService.onCreate");
    _nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    showNotification();
}

```

上述代码作用是，在 `Service` 的几个生命周期函数中加了打印 log 的语句，方便测试和查看。

```

Public class LocalBinder extends Binder {
    TestService getService() {
        return TestService.this;
    }
}

```

 代码解释：

上述这个方法是为了让调用者得到这个 Service 并操作它。

Service 本身的逻辑就这样，你需要做什么就在 onCreate 和 onStart 里添加就好了，需要注意的是如果操作需要花费很多时间，那么最好是用个线程来做以防止其造成用户使用不流畅。

再看一下它的调用者 TestServiceHolder，实现代码如下：

```

package com.iceskysl.TestServiceHolder;

import Android.app.Activity;
import Android.content.ComponentName;
import Android.content.Context;
import Android.content.Intent;
import Android.content.ServiceConnection;
import Android.os.Bundle;
import Android.os.IBinder;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;
import Android.widget.Toast;

public class TestServiceHolder extends Activity {
    private boolean _isBound;
    private TestService _boundService;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        setTitle("Service Test");

        initButtons();
    }

    private ServiceConnection _connection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder Service) {
            _boundService = ((TestService.LocalBinder)Service).getService();

            Toast.makeText(TestServiceHolder.this, "Service connected",
                Toast.LENGTH_SHORT).show();
        }

        public void onServiceDisconnected(ComponentName className) {
            // unexpectedly disconnected, we should never see this happen.
            _boundService = null;
            Toast.makeText(TestServiceHolder.this, "Service disconnected ",
                Toast.LENGTH_SHORT).show();
        }
    };

    private void initButtons() {
        Button buttonStart = (Button) findViewById(R.id.start_Service);
        buttonStart.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {

```

```

startService();
}
});

Button buttonStop = (Button) findViewById(R.id.stop_Service);
buttonStop.setOnClickListener(new OnClickListener() {
public void onClick(View arg0) {
stopService();
}
});

Button buttonBind = (Button) findViewById(R.id.bind_Service);
buttonBind.setOnClickListener(new OnClickListener() {
public void onClick(View arg0) {
bindService();
}
});

Button buttonUnbind = (Button) findViewById(R.id.unbind_Service);
buttonUnbind.setOnClickListener(new OnClickListener() {
public void onClick(View arg0) {
unbindService();
}
});
}

private void startService() {
Intent i = new Intent(this, TestService.class);
this.startService(i);
}

private void stopService() {
Intent i = new Intent(this, TestService.class);
this.stopService(i);
}

private void bindService() {
Intent i = new Intent(this, TestService.class);
bindService(i, _connection, Context.BIND_AUTO_CREATE);
_isBound = true;
}

private void unbindService() {
if (_isBound) {
unbindService(_connection);
_isBound = false;
}
}
}

```

代码解释:

TestServiceHolder 是个很常见的 Activity，在其 onCreate 中，我们设定其对应的布局模板文件为 main.xml，并调用 setTitle 设定其标题为“Service Test”，最好调用自定义的方法 initButtons() 设定几个按钮对应的单击监听器。

上述代码中可以看到两种启动服务的方法，分别是 start 和 bind，当然也是通过 intent 调用的，在 intent 中指明指定要启动的 Service 的名字，stop 也一样，具体实现代码如下所示：

```

private void startService() {
Intent i = new Intent(this, TestService.class);
this.startService(i);
}

```



```
private void stopService() {
    Intent i = new Intent(this, TestService.class);
    this.stopService(i);
}
```

除此之外，还需要一个 ServiceConnection 对象，实现代码如下：

```
private ServiceConnection _connection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder Service) {
        _boundService = ((TestService.LocalBinder)Service).getService();

        Toast.makeText(TestServiceHolder.this, "Service connected",
            Toast.LENGTH_SHORT).show();
    }

    public void onServiceDisconnected(ComponentName className) {
        //连接失败
        _boundService = null;
        Toast.makeText(TestServiceHolder.this, "Service connected",
            Toast.LENGTH_SHORT).show();
    }
};
```

解释代码：

其作用是把 Activity 和特定的 Service 连接在一起，共同生存，具体的生命周期细节下面将详细讲解。

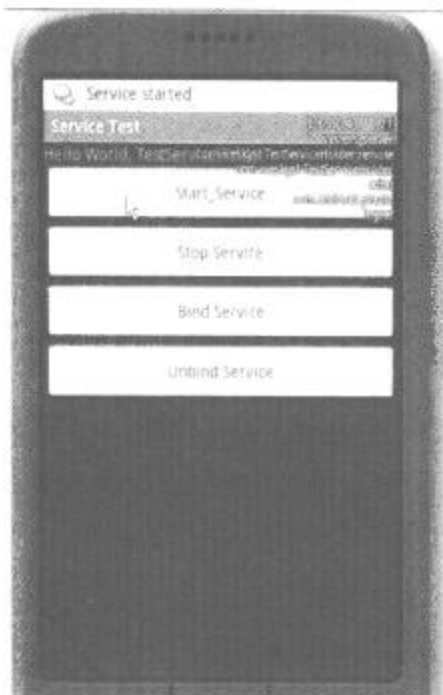
然后看这个例子的运行结果，启动这个应用，可以看到如图 10-3 所示界面。

单击“Start Service”按钮，可以看到如图 10-4 所示的变化。

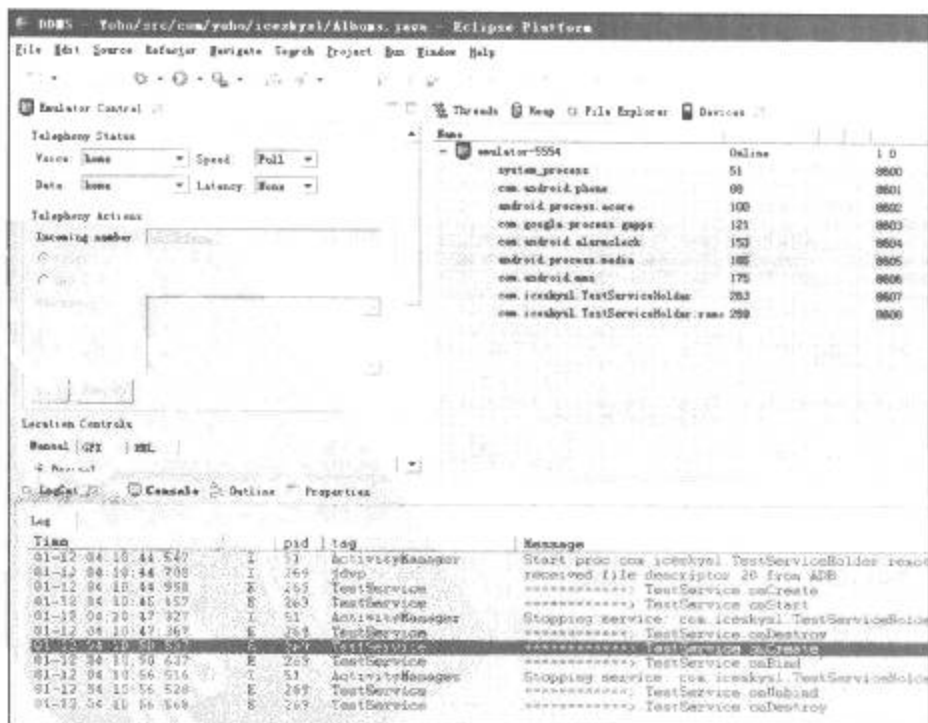
从图 10-4 可以看到标题改变了，表明 Service 正常工作了，依次单击 4 个按钮，可以看到如图 10-5 所示的日志。



▲ 图 10-3 主界面



▲ 图 10-4 单击“Start Service”后的界面



▲ 图 10-5 日志显示

如图 10-5 显示的日志，就是我们加在 TestService 中的，用来标识其被正确调用的结果。到这

里，差不多明白 Service 是如何使用的了，接下来我们会讨论一下 Service 的生命周期。

10.3 Service 的生命周期

Service 的生命周期方法比 Activity 要少一些，只有 onCreate、onStart、onDestroy。从前面实例看到了，有两种方式启动一个 Service，他们对 Service 生命周期的影响是不一样的。

1. 通过 startService 启动

Service 启动的时候会经历生成→开始 (onCreate→onStart) 过程，Service 停止的时候直接进入销毁过程 (onDestroy)。而如果是调用者 (TestServiceHolder) 自己直接退出而没有调用 stopService，Service 会一直在后台运行。直到下次调用者 (TestServiceHolder) 再启动起来，并明确调用 stopService。

2. 通过 bindService

提供 bindService 启动 Service，其只会运行 onCreate，这个时候将 TestServiceHolder 和 TestService 绑定在一起，如果 TestServiceHolder 退出了，Service 就会调用 onUnbind→onDestroyed，所谓绑定在一起就共存亡了。

读者可能要问了，要是这几个方法交织在一起的话，会出现什么情况呢？一个原则是 Service 的 onCreate 的方法只会被调用一次，就是无论多少次的启动或者绑定，Service 只被创建一次。如果先是绑定 (bind) 了，那么启动 (start) 的时候就直接运行 Service 的 onStart 方法，如果先是启动 (start)，那么绑定 (bind) 的时候就直接运行 onBind 方法。如果你先绑定 (bind) 上了，就停止 (stop) 不掉了，也就是 stopService 不能用了，只能先 UnbindService，再 StopService，所以是先启动 (start) 还是先绑定 (bind) 行为是有区别的。

看起来情况很多，比较复杂的样子，如果还是不明白，可以参考本书的示例代码，在搭建的环境中运行一下，多单点按钮，看看其 log 的输出就明白了。我们紧接着还会使用两个实际的例子来进一步说明 Service 的用法。

10.4 实例学习 Service

通过前面对使用方法的代码讲解，若还不是很明白，不用着急，接下来，我们将使用两个更加实际的例子告诉读者怎么使用 Service。

在开始例子之前，大家需要有个概念，Android 系统中已经有很多 Service 了，例如，常见的 AlarmService、GPS Service 等。除了系统自带的 Service，还可以定制自己需要的 Service，如音乐播放 Service 等。接下来的两个例子中，第一个例子演示的是如何调用系统的 Service，后面一个例子将说明我们如何定制、使用自己的 Service。

10.4.1 精彩实例一——定时提醒

首先来看如何使用系统自带的 Service，我们将要创建一个应用，用来演示定时提醒的例子，和前面讲的创建实例的步骤一样，先创建一个名字为 alarmService 的项目。

然后在其布局模板上添加两个按钮，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"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />

    <Button
        android:id="@+id/call_alarm_Service"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="call_alarm_Service"
    />

    <Button
        android:id="@+id/exit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Exit"
    />

</LinearLayout>
```

 代码解释：

上述代码中，两个按钮分别对应的是 call_alarm_Service 和 Exit。

下面将要实现的是单击第一个按钮后，启动一个 Service，然后等待 15 秒后自动停止，或者单击 Exit 按钮停止。实现代码如下：

```
package com.ex_alarmService_1;

import java.util.Calendar;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class alarmService extends Activity implements OnClickListener {
    /** Called when the activity is first created. */
    private static alarmService appRef = null;
    private Button b_call_Service, b_exit_Service;
    boolean k = false;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        appRef = this;

        // 设定模板
        setContentView(R.layout.main);
        b_call_Service = (Button) findViewById(R.id.call_alarm_Service);
```

```

    b_call_Service.setOnClickListener(this);
    b_exit_Service = (Button) findViewById(R.id.exit);
    b_exit_Service.setOnClickListener(this);
}

public static alarmService getApp() {
    return appRef;
}

public void btEvent(String data) {
    setTitle(data);
}
//单击响应方法
public void onClick(View arg0) {

    //如果是启动按钮被单击
    if (arg0 == b_call_Service) {
        setTitle("Waiting... Alarm=5");
        Intent intent = new Intent(alarmService.this, AlarmReceiver.class);
        PendingIntent p_intent = PendingIntent.getBroadcast(
            alarmService.this, 0, intent, 0);
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(System.currentTimeMillis());
        calendar.add(Calendar.SECOND, 5);
        //得到一个定时的服务实例
        AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
        am.set(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
            p_intent);
    }
    //如果是退出按钮被单击
    if (arg0 == b_exit_Service) {
        Intent intent = new Intent(alarmService.this, AlarmReceiver.class);
        PendingIntent p_intent = PendingIntent.getBroadcast(
            alarmService.this, 0, intent, 0);
        AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);

        m.cancel(p_intent);
        finish();
    }
}
}

```

代码解释:

上述代码很简单，取得两个按钮，并设定其被单击时的动作，但是还没完成，需要接着实现 AlarmReceiver，代码如下：

```

package com.ex_alarmService_1;

import Android.content.BroadcastReceiver;
import Android.content.Context;
import Android.content.Intent;

public class AlarmReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent arg1) {
        // TODO Auto-generated method stub
        context.startService(new Intent(context, NotifyService.class));
    }
}

```

代码解释：

这里的 `AlarmReceiver` 继承自 `BroadcastReceiver`，关于 `BroadcastReceiver`，可以这样理解，`Activity` 是应用的“眼睛”，其展示给用户，可以用来交互；而 `BroadcastReceiver` 是“耳朵”，其接收发生的 `Intent`；`Service` 则相对于“手”，其把事情做完。

如此理解上面的代码就很简单了，`AlarmReceiver` 收到 `Intent` 的时候就使用 `context.startService` 启动我们的服务 `NotifyService`。

紧接着，我们看 `NotifyService` 如何实现的，实现代码如下：

```
package com.ex_alarmService_1;
import Android.app.Service;

import Android.content.Intent;
import Android.os.IBinder;
import Android.widget.Toast;
public class NotifyService extends Service {
    @Override

    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public void onCreate() {
        alarmService app = alarmService.getApp();
        app.btEvent("from NotifyService");
        Toast.makeText(this, "hello my iceskysl ", Toast.LENGTH_LONG).show();
    }
}
```

代码解释：

当这个 `Service` 启动，会使用 `btEvent` 改变标题，然后使用 `Toast` 显示一句提示语。

最后，别忘了在 `AndroidManifest.xml` 文件中添加刚刚创建的 `AlarmReceiver` 和 `NotifyService`，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:Android="http://schemas.Android.com/apk/res/Android"
package="com.ex_alarmService_1" Android:versionCode="1"
Android:versionName="1.0.0">
<application Android:icon="@drawable/icon" Android:label="@string/app_name">
<activity Android:name=".alarmService" Android:label="@string/app_name">
<intent-filter>
<action Android:name="Android.intent.action.MAIN" />
<category Android:name="Android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<receiver Android:name=".AlarmReceiver">
</receiver>
<Service Android:name=".NotifyService">
</Service>
</application>
</manifest>
```

操作到这里，这个例子就编写完了，看一下运行的结果，如图 10-6 所示。在图 10-6 中单击“call_alarm_Service”项，出现如图 10-7 所示的界面。



▲ 图 10-6 运行结果

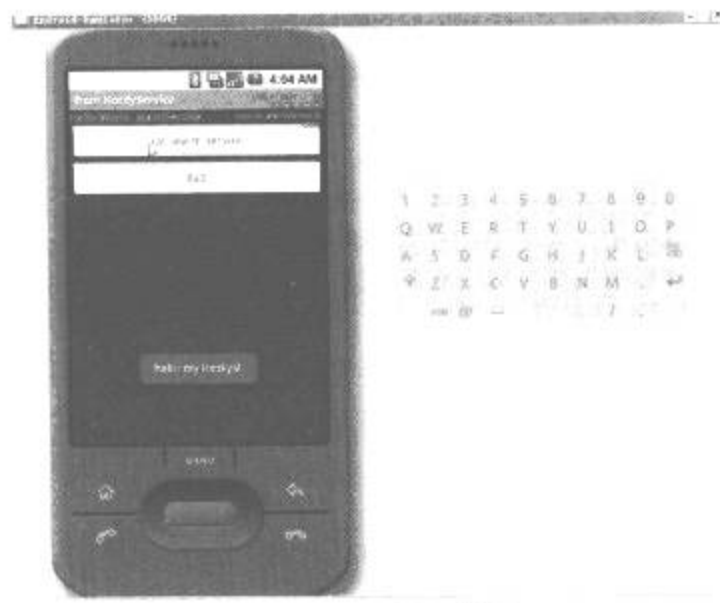
看到标题栏上显示“Waiting... Alarm=5”，5秒后，Service 运行，出现如图 10-8 所示。至此，完成了调用内部 AlarmManager 实现定时提醒的目的。

注意

Android 中的 Alarms 只是在其设置的时间点到达的时候给系统发一个 Alarm 消息唤醒系统，用户不会听到响铃或者感受到振动，如果你需要响铃或者振动，则需要在接受到 Alarm 消息时自己播放一段声音或者启动振动。



▲ 图 10-7 call_alarm_Service 界面



▲ 图 10-8 显示效果

10.4.2 精彩实例二——音乐播放器

前面一个例子说明如何在应用中使用系统 Service，接下来将看看如何实现开发者自己的 Service，这里，我们拿最经典、直观的音乐播放器来说明。

按照我们的设想，需要实现一个音乐播放 Service 的例子，当单击按钮启动播放 Service 后，其在后台播放音乐，当执行其他操作时也不会打断音乐播放。先创建一个项目，名字为 PlayService，如图 10-9 所示。

先给其布局模板文件上添加两个按钮，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"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />

<Button
    android:id="@+id/start"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Start Play"
    />
```



▲ 图 10-9 创建项目

```

<Button
    Android:id="@+id/stop"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:text="Stop Play"
/>

```

```
</LinearLayout>
```

✎ 代码解释：

这个采用 LinearLayout 布局模式，并添加“播放”和“停止”两个控制按钮。

然后实现 PlayService.java，实现代码如下：

```

package com.iceskysl.PlayService;

import Android.app.Activity;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;

public class PlayService extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button1 = (Button)findViewById(R.id.start);
        button1.setOnClickListener(startIt);
        Button button2 = (Button)findViewById(R.id.stop);
        button2.setOnClickListener(stopIt);
    }

    private OnClickListener startIt = new OnClickListener()
    {
        public void onClick(View v)
        {
            startService(new Intent("com.iceskysl.PlayService.START_AUDIO_SERVICE"));
        }
    };

    private OnClickListener stopIt = new OnClickListener()
    {
        public void onClick(View v)
        {
            stopService(new Intent("com.iceskysl.PlayService.START_AUDIO_SERVICE"));
            finish();
        }
    };
}

```

✎ 代码解释：

前面一部分没啥新鲜的知识，主要看后面这个语句：

```

startService(new
Intent("com.iceskysl.PlayService.
START_AUDIO_SERVICE"));

```

其用来启动指定名字的服务，继续操作，下面会创建一个开发者自己的 Service，命名为 Music，如图 10-10 所示。



▲ 图 10-10 创建 Service



注意

其继承自 Android 的 Service 类，添加需要的代码如下所示：

```

package com.iceskysl.PlayService;

import Android.app.Service;
import Android.content.Intent;
import Android.media.MediaPlayer;
import Android.os.IBinder;

public class Music extends Service {
    private MediaPlayer player;

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }

    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
        player = MediaPlayer.create(this, R.raw.gequ);
        player.start();
    }

    public void onDestroy() {
        super.onDestroy();
        player.stop();
    }
}

```



代码解释：

我们首先声明一个 MediaPlayer 对象 player，然后在其 onStart 中播放指定的 MP3 文件（这里是 gequ.mp3），其他的都比较直观，在此不再赘述。

最后在 AndroidManifest.xml 文件中添加对这个 Service 的说明，代码如下所示：

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:Android="http://schemas.Android.com/apk/res/Android"
    package="com.iceskysl.PlayService"
    Android:versionCode="1"
    Android:versionName="1.0.0">
    <application Android:icon="@drawable/icon" Android:label="@string/app_name">
        <activity Android:name=".PlayService"
            Android:label="@string/app_name">
            <intent-filter>
                <action Android:name="Android.intent.action.MAIN" />
                <category Android:name="Android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <Service Android:name=".Music">
        <intent-filter>
        <action Android:name="com.iceskysl.PlayService.START_AUDIO_SERVICE" />
        <category Android:name="Android.intent.category.default" />
        </intent-filter>
        </Service>
    </application>
</manifest>

```

```
</application>
</manifest>
```

解释代码：

可以看到，我们添加了一个名字为“Music”的 Service，其 Action 对应为

```
"com.iceskysl.PlayService.START_AUDIO_SERVICE"
```

最后，运行一下这个应用，看看自己的 Service 是否正常工作了，如图 10-11 所示。

在图 10-11 中单击“Start Play”后，将听到播放歌曲的声音，跳转到其他界面，歌曲依然播放，单击“Stop Play”将停止播放，同时退出应用。

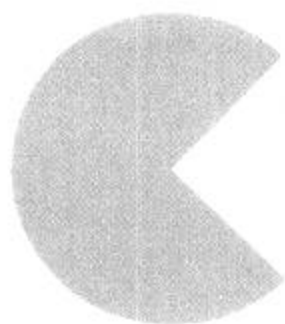


▲ 图 10-11 运行演示

10.5 本章小结

本章主要介绍了 Service 的概念和使用方法，并讨论了不同启动方式下的 Service 不同生命周期，最后使用两个例子讲解了如何在实际应用中使用 Service。

Service 非常强大，我们会在本书后面的实例部分中再次讲解 Service，到时将会展示更多特性和使用技巧。



第 11 章 循序渐进——开发 Android 应用的基本步骤

学习目标

- 掌握 Android 开发的基本流程和步骤; ☐
- 掌握 UI 设计; ☐
- 程序测试和发布。 ☐

11.1 兵马未动 粮草先行——应用规划及架构设计

开发 Android 应用时，在进入具体的编码实现前，需要好好整理一下将要开发的应用步骤，需要整理的主要包括以下内容：

- 项目有哪些功能；
- 需要哪些必须的界面以及界面之间跳转的流程；
- 需要的数据及其数据的来源和格式；
- 是否需要服务端支持；
- 是否需要本地数据库支持；
- 是否需要特殊的权限；
- 是否需要后台进程等。

在开始实现一个新的应用开发时，程序员一般习惯于用纸和笔列出这个应用所必须的功能，该功能所需要的数据，及其展示界面，然后将每个界面上需要显示的元素，用笔画在纸上，并理清每个界面直接的跳转关系，再理清其主要的功能后，这样就有了这个应用的基本开发轮廓了，然后围绕这个应用轮廓进行其他的细化。

依照这个项目的大致轮廓，进一步整理其中每个界面（功能）需要的数据来源，搞清楚数据到底是来源于网络还是本机，再考虑是否有数据存储的必要，如果需要存储的数据，是存储到数据库还是文件比较合适。在数据这部分考虑的差不多了，再深入想想是否需要后台进程，如果需要，那么这个进程是需要长期驻留还是只在应用启动期间在后台运行。这些方面都考虑差不多以后，还需要整理下一其可能会涉及哪些是需要特殊权限的。

11.2 应用开发步骤

在规划完整个应用的主要界面，功能以及其周边的一些东西后，这个时候就可以按部就班地开

发实现了，依照笔者的经验，在开发过程中可以遵守如下流程。

11.2.1 界面设计始终是第一位——实现 UI

按照前面的规划，参考自己画出来的应用界面图，可以很清晰的知道需要几个界面以及界面上的元素，那么在这里就只是按照上述规划，使用 XML 文件描述出 Android 的模板就可以了，例如，使用那种布局模式，在哪个地方放置什么视图组件，组件中将要填充哪些数据等。

关于使用 XML 描述界面模板，可以参考本书第 7 章相关内容，其中比较详细地介绍了常用的布局模式和视图组件的使用。另外，还需要实现其界面上各个组件需要响应的事件，如单击按钮触发的事件，单击“meun”需要展示的菜单，选择菜单需要执行的操作等。

11.2.2 必备的动力源泉——数据操作和存储

在描绘完应用程序的 UI 后，然后需要仔细考虑实现的数据存储策略，按照先前做的项目规划轮廓，我们已经知道需要哪些数据，以及数据的来源。开发项目时，应用项目的数据来源有很多种，例如，最常见的有 SharedPreferences、文件系统、数据库、Content Provider 及网络等方式，这里需要分清楚这些数据是否需要存储，以及存储方式的选择，这部分在本书第 8 章已经比较详细地介绍过了。相信读者通过前面章节的学习，在实现应用开发的时候，可以很清楚的把握自己需要存储哪些数据，以及这些数据的存储方式。

11.2.3 华丽转身——实现多页面跳转

应用的 UI 和数据存储都论证实现后，现在就可以把整个流程连起来了，开发者需要实现各个界面直接的调用和跳转流程，例如是选择菜单跳转，还是单击按钮实现，以及监听事件发生等处理，如果一些跳转暂时无法连起来，那么可以使用其他方式，如在界面上设置一个按钮，单击跳转到目标界面。

这个步骤用来进一步梳理应用的界面，如果整个流程都可以运行起来，那么开发的应用差不多就完成一半了，后面需要做的是进一步完善和细化。

11.2.4 始终为用户做好服务——增加 Service

前面做的工作都是比较“表面”的工作，其主要是实现人-机交互以及前台界面的跳转等，但是还有一个比较重要的内容需要关注，那就是论证是否需要服务（Service），关于 Service 的介绍已经在本书的第 10 章详细地介绍过。而这里需要做的就是实现自己需要的 Service，按照第 10 章介绍的方法编程实现。

11.2.5 细节决定成败——完善应用细节

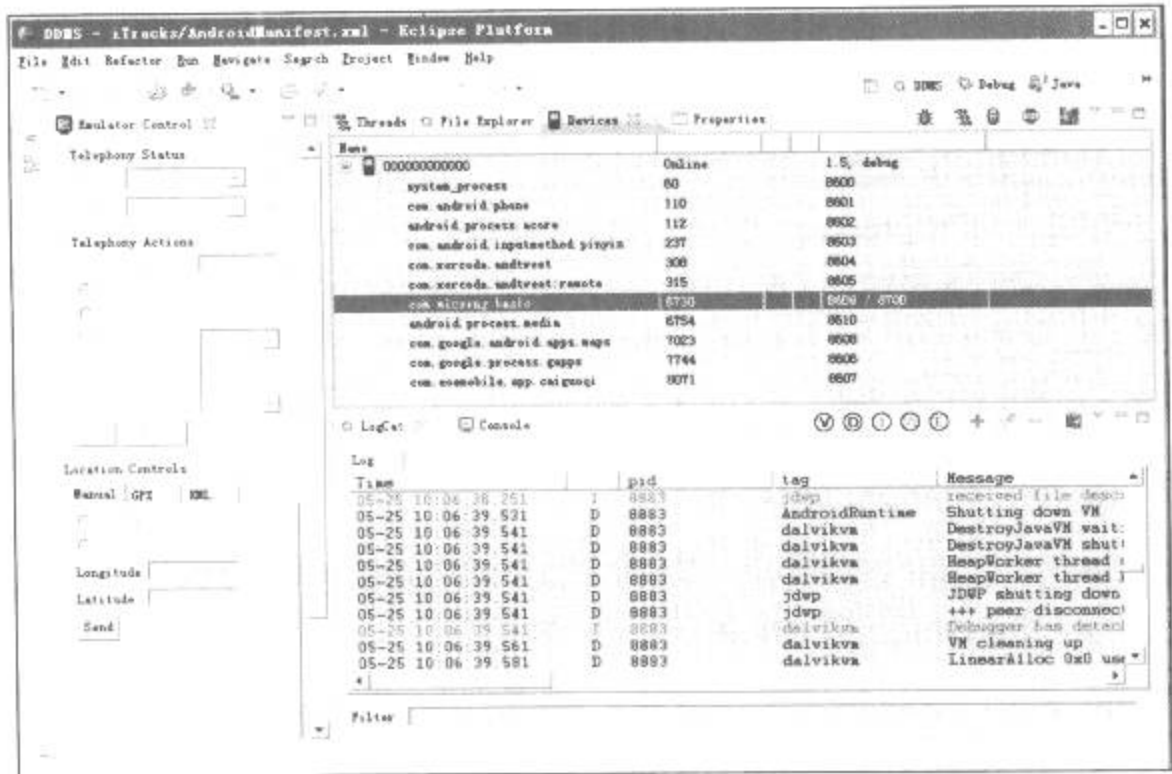
在完成大部分的功能后，最后还需要检查一些细节，如若应用需要从网络上获取数据，那么就需要在 AndroidManifest.xml 文件中添加相关的 uses-permission；如果是对老版本的更新，那么需要在 AndroidManifest.xml 文件中更新 Android:versionCode 和 Android:versionName。这些都是比较细节的东西，需要自己不断的积累，否则可能因为一个细节的缺失而浪费许多开发时间去排查错误原因。

11.3 成功就在眼前——应用测试和发布

11.3.1 只欠东风——应用测试

项目程序开发完成后，接下来需要对应用进行测试。可以选择使用模拟器或者真机模拟测试，这里不得不说一下 Android 的模拟器做的非常棒，配合其在 Eclipse 中的开发插件 DDMS 功能也非常丰富，如下图所示。

利用模拟器和其控制面板，可以模拟很多情况，如发送、接收短信，打电话，模拟 GPS 信息等，且测试出来的效果和真机上实现的非常接近。



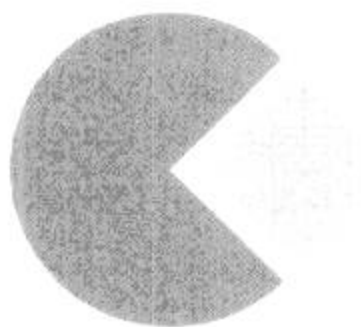
▲ 图 DDMS 的模拟器控制面板

11.3.2 可以赚钱了——发布到 Android Market

应用程序开发完，并经过测试后，就可以考虑发布了，其打包、生成签名文件的步骤和方法在本书第 4 章已经详细地介绍过了，当时可能还不是很理解，那么现在可以翻回去再看一遍了。按照第 4 章介绍的方法，完成签名后就可以上传到 Android Market 上供其他用户下载和使用了，使用的用户可以对你的应用进行评价和打分，在这里也就不再赘述了。

11.4 本章小结

本章主要介绍了开发 Android 应用的基本步骤，其中讲到的都是笔者本人实战应用的开发总结，希望对读者有一个抛砖引玉的作用。编写本章的目的就是让读者有一个应用开发的流程概念，使得在开发应用前能仔细的想一想，理清自己的思路，找到最合理的实现路线。从本章以后，将进入实例讲解阶段，每个实例差不多都是遵守这个步骤，希望通过后续章节的学习，能使读者真正成为实战角色。



第 12 章 Android 综合案例—— RSS 阅读器实例

学习目标

- 了解什么是 RSS; ☐
- 了解什么是 SAX; ☐
- 如何设计 RSS 阅读器; ☐
- RSS 阅读器的实现。 ☐

12.1 RSS 介绍

12.1.1 RSS 基础

RSS 是一个网络站点用来和其他站点之间共享内容的一种简易方式（也叫聚合内容）。RSS 是基于文本的格式。它是 XML（可扩展标识语言）的一种形式。通常 RSS 文件都是标为 XML，RSS files（通常也被称为 RSS feeds 或者 channels）通常只包含简单的项目列表。一般而言，每一个项目都含有一个标题，一段简单的介绍，还有一个 URL 链接（如一个网页的地址）。其他的信息，例如，日期、创建者的名字等，都是可以选择的。发布一个 RSS 文件（一般称为 RSS Feed）后，这个 RSS Feed 中包含的信息就能直接被其他站点调用，而且由于这些数据都是标准的 XML 格式，所以，也能在其他的终端和服务中使用，如 PDA、手机、邮件列表等。

12.1.2 RSS 的历史

那么 RSS 究竟代表什么呢？比较普遍的有两种解释，一种是“Rich Site Summary”或“RDF Site Summary”，另一种是“Really Simple Syndication”，之所以有这些分歧，需要从 RSS 发展的历史说起。

最初的 0.90 版本 RSS 是由 Netscape 公司设计的，目的是用来建立一个整合了各主要新闻站点内容的门户，但是 0.90 版本的 RSS 规范过于复杂，而一个简化的 RSS 0.91 版本也随着 Netscape 公司对该项目的放弃而于 2000 年暂停。不久，一家专门从事博客写作软件开发公司 UserLand 接手了 RSS 0.91 版本的发展，并把它作为其博客写作软件的基础功能之一继续开发，逐步推出了 0.92、0.93 和 0.94 版本。随着网络博客的流行，RSS 作为一种基本的功能也被越来越多的网站和博客软件支持。

在 UserLand 公司接手并不断开发 RSS 的同时，很多的专业人士认识到需要通过一个第三方、非商业的组织，把 RSS 发展成为一个通用的规范，并进一步标准化。于是 2001 年一个联合小组在

0.90 版本 RSS 的开发原则下，以 W3C 新一代的语义网技术 RDF (Resource Description Framework) 为基础，对 RSS 进行了重新定义，发布 RSS 1.0，并将 RSS 定义为“RDF Site Summary”。但是这项工作没有与 UserLand 公司进行有效的沟通，UserLand 公司也不承认 RSS 1.0 的有效性，并坚持按照自己的设想进一步开发出 RSS 的后续版本，到 2002 年 9 月发布了最新版本 RSS 2.0，UserLand 公司将 RSS 定义为“Really Simple Syndication”。

目前 RSS 已经分化为 RSS 0.9x/2.0 和 RSS 1.0 两个阵营，由于分歧的存在和 RSS 0.9x/2.0 的广泛应用现状，RSS 1.0 还没有成为标准化组织的真正标准。

12.1.3 RSS 语法介绍

一个 RSS 文件就是一段规范的 XML 数据，该文件一般以 RSS、XML 或者 RDF 作为后缀。下面来看一个具体的 RSS 文档，来简单地了解一下 RSS 的描述语法。

让我们看一个简单的 RSS 文档，具体代码格式如下所示：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<RSS version="2.0">

<channel>
  <title>Android Team</title>
  <link>http://www.eoemobile.com</link>
  <description>good and great</description>
  <item>
    <title>learn Android </title>
    <link>http://www.eoeAndroid.com</link>
    <description>Android tutorial</description>
  </item>
</channel>

</RSS>
```

代码解释：

文档中的第一行：XML 声明，定义了文档中使用的 XML 版本和字符编码。此例子遵守 1.0 规范，并使用 ISO-8859-1 (Latin-1/West European) 字符集。第二行是标识此文档是一个 RSS 文档的 RSS 声明（此例是 RSS version 2.0）。第三行含有<channel>元素。此元素用于描述 RSS feed。

<channel> 元素有 3 个必需的子元素：

- <title> 定义频道的标题；
- <link> 定义到达频道的超链接；
- <description> 描述此频道。

每个 <channel> 元素可拥有一个或多个 <item> 元素。每个 <item> 元素可定义 RSS feed 中的一篇文章或 "story"。

<item> 元素拥有 3 个必需的子元素：

- <title>-定义项目的标题；
- <link>-定义到达项目的超链接；
- <description>-描述此项目。

最后，后面的两行关闭 <channel> 和 <RSS> 元素。

12.2 SAX 介绍

12.2.1 SAX 基础

SAX 的全名是 Simple API for XML，同 DOM 一样也是一个访问 XML 文档的接口。像 DOM 一样，SAX 并不是一个实际可以使用的 XML 文档解析器，而是其他兼容 SAX 的解析器要实现的接口和帮助类的集合。实现了 SAX 的解析器有很多，如 Apache 的 Xerces，Oracle 的 XML Parser 等。当然在 Android 中也支持 SAX 来解析 XML，在 Android 的 jar 文件包中可以看到有两个和 SAX 相关的包，它们分别是 `org.xml.sax` 和 `org.xml.sax.helper` 包。

12.2.2 使用 SAX 的作用

解析 XML 可以采用 DOM 进行解析，也可以采用 SAX 进行解析，当然各有各的好处，根据情况不同，而用不同的解析方式。

DOM（文档对象模型），为 XML 文档的解析定义了一组接口。解析器读入整个文档，然后构建一个驻留内存的树结构，然后代码就可以使用 DOM 接口来操作整个树结构。其特点如下。

- 优点：整个文档树在内存当中，便与操作；支持删除、修改、重新排列等多种功能。
- 缺点：将整个文档调入内存（经常包括大量无用的节点），浪费时间和空间。
- 使用场合：一旦解析了文档还需要多次访问这些数据、硬件资源比较充足（如内存、CPU 等）。

为了解决 DOM 解析 XML 引起的这些问题，出现了 SAX。SAX 解析 XML 文档为事件驱动，详细地介绍我们放在后边进行。当解析器发现元素开始、元素结束，文本、文档的开始或者结束时，发送事件，程序员编写响应这些事件的代码。其特点如下。

- 优点：不用事先调入整个文档，占用资源少。尤其在嵌入式环境，如 Android，极力推荐采用 SAX 进行解析。
- 缺点：不像 DOM 一样将文档树长期驻留在内存，数据不是持久的。事件过后，若没保存数据，那么数据就丢失。
- 使用场合：机器有性能限制，尤其是在嵌入式环境，如 Android，极力推荐采用 SAX 进行解析。

12.2.3 怎样使用 SAX

SAX 的设计实现与 DOM 是完全不同的。DOM 处理 XML 文档是基于将 XML 文档解析成树状模型，放入内存进行处理。而 SAX 则是采用基于事件驱动的处理模式，它将 XML 文档转化成一系列的事件，由单独的事件处理器来决定如何处理。为了了解如何使用 SAX API 处理 XML 文档，这里先介绍一下 SAX 所使用的基于事件驱动的处理模式。这种基于事件的处理模式是一种通用的程序设计模式，被广泛应用于 GUI 设计。在 Java 的 AWT、SWING 以及 Java Beans 中就有它的身影。而 SAX 的基于事件驱动的处理模式就与上面 3 者中的非常相像。

基于事件的处理模式主要是围绕着事件源以及事件处理器（或者叫监听器）来工作的。一个可以产生事件的对象被称为事件源，而可以针对事件产生响应的对象就被叫做事件处理器。事件源和事件处理器是通过在事件源中的事件处理器注册方法连接的。这样当事件源产生事件后，调用事件

处理器相应的处理方法，一个事件就获得了处理。当然在事件源调用事件处理器中特定方法的时候，会传递给事件处理器相应事件的状态信息，这样事件处理器才能够根据事件信息来决定自己的行为。

在 SAX 接口中，事件源是 org.xml.sax 包中的 XMLReader，它通过 parse()方法来开始解析 XML 文档，并根据文档内容产生事件。而事件处理器则是 org.xml.sax 包中的 ContentHandler、DTDHandler、ErrorHandler，以及 EntityResolver 这 4 个接口。它们分别处理事件源在解析过程中产生的不同种类的事件（其中 DTDHandler 是为解析文档 DTD 时所用）。而事件源 XMLReader 和这 4 个事件处理器的连接是通过在 XMLReader 中的相应事件处理器注册方法 set***()来完成的。详细介绍如表 12-1 所示。

表 12-1 相应事件处理器描述

处理器名称	所处理事件	注册方法
org.xml.sax.ContentHandler	跟文档内容有关的所有事件 (1) 文档的开始和结束 (2) ML 元素的开始和结束 (3) 可忽略的实体 (4) 名称空间前缀映射开始和结束 (5) 处理指令 (6) 字符数据和可忽略的空格	XMLReader 中的 setContentHandler(ContentHandler handler)方法
org.xml.sax.ErrorHandler	处理 XML 文档解析时产生的错误。如果一个应用程序没有注册一个错误处理器类，会发生不可预料的解析器行为	setErrorHandler(ErrorHandler handler)
org.xml.sax.DTDHandler	处理对文档 DTD 进行解析时产生的相应事件	setDTDHandler(DTDHandler handler)
org.xml.sax.EntityResolver	处理外部实体	setEntityResolver(EntityResolver resolver)

在这 4 个处理器接口中，最重要的是 ContentHandler 接口。下面看一下对其中方法的说明，如表 12-2 所示。

表 12-2 ContentHandler 接口中方法的说明

方法名称	方法说明
public void setDocumentLocator(Locator locator)	设置一个可以定位文档内容事件发生位置的定位器对象
public void startDocument() throws SAXException	用于处理文档解析开始事件
public void endDocument() throws SAXException	用于处理文档解析结束事件
public void startElement(java.lang.String namespaceURI,java.lang.String localName,java.lang.String qName,Attributes atts) throws SAXException	处理元素开始事件，从参数中可以获得元素所在名称空间的 uri，元素名称，属性列表等信息

续表

方法名称	方法说明
<code>public void endElement(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName) throws SAXException</code>	处理元素结束事件，从参数中可以获得元素所在名称空间的 uri，元素名称等信息
<code>public void characters(char[] ch, int start, int length) throws SAXException</code>	处理元素的字符内容，从参数中可以获得内容

这里再介绍一下 `org.xml.sax.XMLReader` 中的方法。当然并不是每一个方法都会在每次解析的时候用得到，我们就在这个例子当中需要使用的方法做一个简单的介绍，具体介绍如表 12-3 所示。

表 12-3 使用的方法介绍

方法名称	方法介绍
<code>Public void setContentHandler(ContentHandler handler)</code>	注册处理 XML 文档内容解析事件的 ContentHandler
<code>public void parse(InputSource input) throws java.io.IOException, SAXException</code>	开始解析一个 XML 文档

关于具体的使用例子可以参照后续章节的例子讲解。

12.3 RSS 阅读器设计

12.3.1 RSS 阅读器功能设计

RSS 阅读器是第一个比较完整的例子，从功能上讲是比较简单，主要是从网站上获得一个 RSS 的 feed，然后解析这个 feed，并将解析完的内容，以列表的形式显示出来。单击其中的某一项后，可以看到一个简要的内容。

12.3.2 RSS 阅读器 UI 和交互流程设计

这个交互流程也是比较简单的，只有两个界面。一个界面以列表的形式显示解析出来的 RSS feed 的内容，另外一个界面显示单击其中一项后的详细内容。

1. 程序启动时的界面

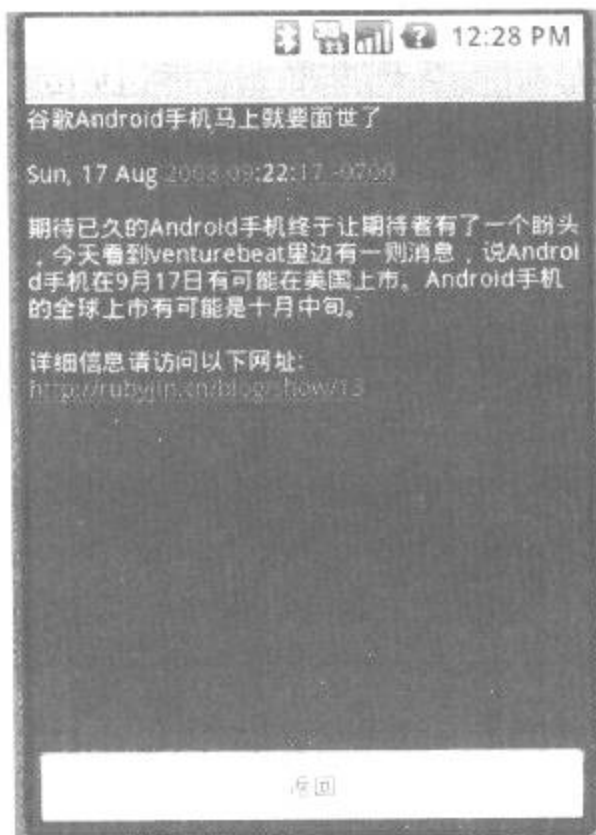
程序启动时的界面如图 12-1 所示。

2. 单击其中一项

单击其中一项后，进入如图 12-2 所示的界面。



▲ 图 12-1 程序初始化界面



▲ 图 12-2 详细信息界面

12.4 RSS 阅读器的实现

12.4.1 程序实体解析

1. 建立实体类

一个 RSS 文件可以被认为是由一个 RSS 的一些描述信息和它里边的 item 组成，如下边的 RSS 文件所示。

(1) 关于 RSS 的描述信息有：

- title，标题信息；
- link，链接信息；
- description，描述信息。

(2) item 里边包含的信息有：

- title，标题信息；
- link，链接信息；
- description，描述信息；
- pubDate，发布日期信息。

一个 RSS 文件的描述代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<RSS version="2.0" xmlns:content="http://purl.org/RSS/1.0/modules/content/" xmlns:itunes
="http://www.itunes.com/dtds/podcast-1.0.dtd"
```

```

xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:trackback="http://madskills.com/public/xml/RSS/module/trackback/">
<channel>
<title>Haiyang | Focus on Android</title>
<link>http://rubyjin.cn</link>
<description>focus on Android and startup</description>
<item>
<title>谷歌官方博客：手机的未来</title>
<link>http://rubyjin.cn/blog/show/17</link>
<description>这是描述信息</description>
<pubDate>Tue, 23 Sep 2008 06:26:54 -0700</pubDate>
</item>
</channel>
</RSS>

```

在这个项目当中我们建立两个实体类：

- RSSFeed，和一个 RSS 的完整 xml 文件相对应；
- RSSItem，和一个 RSS 中的 item 标签相对应。

在解析 RSS 文件的时候，我们就可以将文件里边的信息解析出来放进实体类里边，这样就可以直接操作实体类了。下边是两个实体类的具体代码：

RSSFeed 类：

```

package com.RSS_reader.data;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Vector;

public class RSSFeed
{
    private String title = null;
    private String pubdate = null;
    private int itemcount = 0;
    private List<RSSItem> itemlist;
    public RSSFeed()
    {
        itemlist = new Vector(0);
    }
    public int addItem(RSSItem item)
    {
        itemlist.add(item);
        itemcount++;
        return itemcount;
    }
    public RSSItem getItem(int location)
    {
        return itemlist.get(location);
    }
    public List getAllItems()
    {
        return itemlist;
    }
    public List getAllItemsForListView() {
        List<Map<String, Object>> data = new ArrayList<Map<String, Object>>();
        int size = itemlist.size();
        for(int i=0;i<size;i++){
            HashMap<String, Object> item = new HashMap<String, Object>();
            item.put(RSSItem.TITLE, itemlist.get(i).getTitle());

```

```

item.put(RSSItem.PUBDATE, itemlist.get(i).getPubDate());
data.add(item);
}
return data;
}
int getItemCount()
{
return itemcount;
}
public void setTitle(String title)
{
this.title = title;
}
public void setPubDate(String pubdate)
{
this.pubdate = pubdate;
}
public String getTitle()
{
return title;
}
public String getPubDate()
{
return pubdate;
}
}

```

代码解释:

- addItem 方法负责将一个 RSSItem 加入到 RSSFeed 类里边。
- getAllItemsForListView 方法负责从 RSSFeed 类里边生成列表 (ListView) 所需要的数据。

RSSItem 类:

```

package com.RSS_reader.data;

public class RSSItem
{
public static final String TITLE="title";
public static final String PUBDATE="pubdate";
private String title = null;
private String description = null;
private String link = null;
private String category = null;
private String pubdate = null;

public RSSItem()
{
}
public void setTitle(String title)
{
this.title = title;
}
public void setDescription(String description)
{
this.description = description;
}
public void setLink(String link)
{
this.link = link;
}
public void setCategory(String category)

```



```

{
    this.category = category;
}
public void setPubDate(String pubdate)
{
    this.pubdate = pubdate;
}
public String getTitle()
{
    return title;
}
public String getDescription()
{
    return description;
}
public String getLink()
{
    return link;
}
public String getCategory()
{
    return category;
}
public String getPubDate()
{
    return pubdate;
}
public String toString()
{
    if (title.length() > 20)
    {
        return title.substring(0, 42) + "...";
    }
    return title;
}
}

```

 代码解释:

- RSSItem 类和一个 RSS 文件中的 item 标签相对应，里边的属性也是和 item 里边的属性一样。

2. SAX 进行实体解析的步骤

建立实体类后，现在就可以用 SAX 进行解析了。将需要的内容从 RSS feed 里边读出来，然后保存到实体当中。

在 Android 中使用 SAX 是有迹可循的，完全可以遵照下面的方法就可以轻松的找到 XML 里的 tag，然后得到想要的内容。具体实现步骤如下。

(1) 第一步：新建一个工厂类 SAXParserFactory，代码如下所示：

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

(2) 第二步：让工厂类生产出一个 SAX 的解析类 SAXParser，代码如下所示：

```
SAXParser parser = factory.newSAXParser();
```

(3) 第三步：从 SAXParser 中得到一个 XMLReader 的实例，XMLReader 是一个接口，这个接口里边定义了一些解析 XML 的回调函数，代码如下所示：

```
XMLReader xmlreader = parser.getXMLReader();
```

(4) 第四步：把自己编写的 Handler 注册到 XMLReader 中去，一般最重要的就是 ContentHandler，

当然可以根据需要注册其他的 handler，代码如下所示：

```
RSSHandler RSSHandler = new RSSHandler();
xmlreader.setContentHandler(RSSHandler);
```

(5) 第五步：将一个 XML 的文档或者资源变成一个 Java 可以处理的 InputStream 流后，解析正式开始。实现代码如下：

```
xmlreader.parse(is);
```

在我们这个例子当中，关于以上 5 个步骤的完整代码为：

```
URL url = new URL(urlString);

SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
XMLReader xmlreader = parser.getXMLReader();

RSSHandler RSSHandler = new RSSHandler();
xmlreader.setContentHandler(RSSHandler);

InputSource is = new InputSource(url.openStream());
xmlreader.parse(is);
```

12.4.2 实现一个自己的 ContentHandler

在上边的几个步骤中，绝大部分都是容易理解和掌握的，但是读者可能有一点不明白，那就是在上边第四步中的 RSSHandler 是怎么实现的。对了，这也是现在要分析的，在解析一个 XML 的时候，大部分都是固定的步骤，但是关于 ContentHandler 的实现却是不一样的，一个 ContentHandler 的实现是解析 XML 最重要、最关键的部分。我们来看一下在程序当中，RSSHandler 实现的代码：

```
package com.RSS_reader.sax;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import Android.util.Log;

import com.RSS_reader.data.RSSFeed;
import com.RSS_reader.data.RSSItem;

public class RSSHandler extends DefaultHandler
{
    RSSFeed RSSFeed;
    RSSItem RSSItem;
    String lastElementName = "";
    final int RSS_TITLE = 1;
    final int RSS_LINK = 2;
    final int RSS_DESCRIPTION = 3;
    final int RSS_CATEGORY = 4;
    final int RSS_PUBDATE = 5;
    int currentstate = 0;

    public RSSHandler()
    {
    }

    public RSSFeed getFeed()
    {
        return RSSFeed;
    }
}
```

```

    }
    public void startDocument() throws SAXException
    {
        RSSFeed = new RSSFeed();
        RSSItem = new RSSItem();

    }
    public void endDocument() throws SAXException
    {
    }
    public void startElement(String namespaceURI, String localName, String qName, Attributes
    atts) throws SAXException
    {
        if (localName.equals("channel"))
        {
            currentstate = 0;
            return;
        }
        if (localName.equals("item"))
        {
            RSSItem = new RSSItem();
            return;
        }
        if (localName.equals("title"))
        {
            currentstate = RSS_TITLE;
            return;
        }
        if (localName.equals("description"))
        {
            currentstate = RSS_DESCRIPTION;
            return;
        }
        if (localName.equals("link"))
        {
            currentstate = RSS_LINK;
            return;
        }
        if (localName.equals("category"))
        {
            currentstate = RSS_CATEGORY;
            return;
        }
        if (localName.equals("pubDate"))
        {
            currentstate = RSS_PUBDATE;
            return;
        }
        currentstate = 0;
    }
    public void endElement(String namespaceURI, String localName, String qName) throws
    SAXException
    {
        //如果解析一个 item 节点结束, 就将 RSSItem 添加到 RSSFeed 中。
        if (localName.equals("item"))
        {
            RSSFeed.addItem(RSSItem);
            return;
        }
    }
}

```

```

public void characters(char ch[], int start, int length)
{
    String theString = new String(ch, start, length);
    switch (currentstate)
    {
        case RSS_TITLE:
            RSSItem.setTitle(theString);
            currentstate = 0;
            break;
        case RSS_LINK:
            RSSItem.setLink(theString);
            currentstate = 0;
            break;
        case RSS_DESCRIPTION:
            RSSItem.setDescription(theString);
            currentstate = 0;
            break;
        case RSS_CATEGORY:
            RSSItem.setCategory(theString);
            currentstate = 0;
            break;
        case RSS_PUBDATE:
            RSSItem.setPubDate(theString);
            currentstate = 0;
            break;
        default:
            return;
    }
}

```

代码解释:

结合上边的代码分析, 实现一个 `ContentHandler` 一般可以由以下几个步骤组成。

(1) 第一步: 声明一个类, 这个类继承 `DefaultHandler`。`DefaultHandler` 是一个基类, 这个类里边最简单地实现了一个 `ContentHandler`, 我们只需要重写里边的重要方法即可。

(2) 第二步: 重写 `startDocument()` 和 `endDocument()` 两个方法。一般的将正式解析之前的一些初始化的工作放到 `startDocument()` 这个方法里边进行, 例如, 在这个例子里边, 我们在此方法里边初始化了 `RSSFeed` 和 `RSSItem`, 可以将一些收尾的工作放到 `endDocument()` 方法里处理, 在这个例子当中, 没有什么工作需要做, 所以这个方法里边的方法体为空, 不执行任何操作。

(3) 第三步: 重写 `startElement (String namespaceURI, String localName, String qName, Attributes atts)` 方法。XML 解析器遇到 XML 文档流里边的 tag 的时候, 就会调用这个函数。经常在这个函数内部是通过 `localName` 这个参数进行判断而作一些处理操作, 例如, 在这个例子当中如果 `localName` 是 “Item”, 那么我们不但要标记 `currentstate` 为 Item, 而且还要新建一个 Item 的实体, 用来保存解析的 Item 的内容。

(4) 第四步: 重写 `characters (char ch[], int start, int length)` 方法。这也是一个回调方法, 当解析器执行完 `startElement()` 方法后, 解析完节点的内容后就会执行这个方法, 并且参数 `ch[]` 就是节点的内容。在这个例子当中, 我们根据 `currentstate` 的不同, 来得出当前是哪个 tag 节点的内容。并把内容放置到合适的实体类里边。

(5) 第五步: 重写 `endElement (String namespaceURI, String localName, String qName)` 方法。这个方法和 `startElement()` 方法对应。当解析完一个 tag 节点后, 执行这个方法。在这个程序中如果

解析一个 Item 节点结束，就将 RSSItem 添加到 RSSFeed 中。

12.4.3 Activity 的实现

1. ActivityMain 的实现

按照开发的惯例，我们这个程序的入口 Activity 为 ActivityMain，在这个 Activity 中得到了服务器端的 RSS feed，经过解析后将里边的内容以 ListView 的形式显示出来。具体代码如下：

```
package com.RSS_reader;

import java.net.URL;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;

import Android.app.Activity;
import Android.content.Intent;
import Android.os.Bundle;
import Android.view.View;
import Android.widget.AdapterView;
import Android.widget.AdapterView.OnItemClickListener;
import Android.widget.SimpleAdapter;

import com.RSS_reader.data.RSSFeed;
import com.RSS_reader.data.RSSItem;
import com.RSS_reader.sax.RSSHandler;

public class ActivityMain extends Activity implements OnItemClickListener
{
    public final String RSS_URL = "http://rubyjin.cn/blog/RSS";
    public final String tag = "RSSReader";
    private RSSFeed feed = null;
    /** Called when the activity is first created. */

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        feed = getFeed(RSS_URL);
        showListView();
    }

    //得到一个 RSSFeed
    private RSSFeed getFeed(String urlString)
    {
        try
        {
            URL url = new URL(urlString);

            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();
            XMLReader xmlreader = parser.getXMLReader();

            RSSHandler RSSHandler = new RSSHandler();
```



```

xmlreader.setContentHandler(RSSHandler);

InputStream is = new InputStream(url.openStream());
xmlreader.parse(is);

    return RSSHandler.getFeed();
}
catch (Exception ee)
{
    return null;
}
}

//显示列表
private void showListView()
{
    ListView itemlist = (ListView) findViewById(R.id.itemlist);
    if (feed == null)
    {
        setTitle("访问的 RSS 无效");
        return;
    }

    SimpleAdapter adapter = new SimpleAdapter(this, feed.getAllItemsForListView(),
        Android.R.layout.simple_list_item_2, new String[] { RSSItem.TITLE, RSSItem.PUBDATE },
        new int[] { Android.R.id.text1, Android.R.id.text2 });
    itemlist.setAdapter(adapter);
    itemlist.setOnItemClickListener(this);
    itemlist.setSelection(0);
}

//处理列表的单击事件
public void onItemClick(AdapterView parent, View v, int position, long id)
{
    Intent itemintent = new Intent(this, ActivityShowDescription.class);

    Bundle b = new Bundle();
    b.putString("title", feed.getItem(position).getTitle());
    b.putString("description", feed.getItem(position).getDescription());
    b.putString("link", feed.getItem(position).getLink());
    b.putString("pubdate", feed.getItem(position).getPubDate());

    itemintent.putExtra("Android.intent.extra.RSSItem", b);
    startActivityForResult(itemintent, 0);
}
}

```

代码解释:

- 在 `getFeed()` 函数里边我们从服务器端请求了 RSS feed，并且进行了解析，将解析完的内容全部都放在 `RSSFeed` 的一个实例里边。
- `showListView()` 函数负责将 `RSSFeed` 里边的内容以 `ListView` 的形式展现出来。我们让 `ListView` 和一个 `SimpleAdapter` 进行绑定。
- 由于 `ActivityMain` 实现了 `OnItemClickListener` 接口，所以处理单击 `ListView` 里边具体某项的操作都可以放到 `onItemClick()` 函数里边。在 `onItemClick()` 函数里边将 `Item` 的信息放到 `Extra` 当中，然后进行 `Activity` 的跳转。

2. ActivityShowDescription 的实现

单击列表的某一项后进入这一项的详细信息界面,这个界面由 ActivityShowDescription 来实现,具体代码如下:

```
package com.RSS_reader;

import Android.app.Activity;
import Android.os.Bundle;
import Android.widget.Button;
import Android.widget.TextView;
import Android.content.Intent;
import Android.view.*;

public class ActivityShowDescription extends Activity {
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.showdescription);
        String content = null;
        Intent startingIntent = getIntent();

        if (startingIntent != null) {
            Bundle bundle = startingIntent
                .getBundleExtra("Android.intent.extra.RSSItem");
            if (bundle == null) {
                content = "不好意思程序出错啦";
            } else {
                content = bundle.getString("title") + "\n\n"
                    + bundle.getString("pubdate") + "\n\n"
                    + bundle.getString("description").replace('\n', ' ')
                    + "\n\n 详细信息请访问以下网址:\n" + bundle.getString("link");
            }
        } else {
            content = "不好意思程序出错啦";
        }

        TextView textView = (TextView) findViewById(R.id.content);
        textView.setText(content);

        Button backbutton = (Button) findViewById(R.id.back);

        backbutton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```

 代码解释:

- 在 onCreate()函数里,我们从启动这个 Activity 的 intent 里边取出 Extra 信息,然后将里边的内容放到一个 TextVeiw 进行显示。

这个 Activity 对应的 layout 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"
```

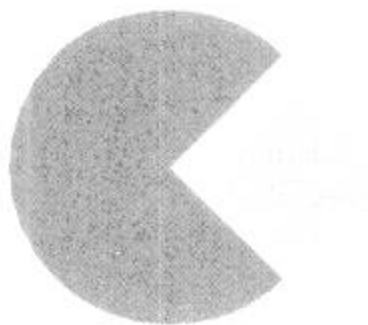
```
        Android:layout_height="fill_parent"
    >
    <TextView
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:autoLink="all"
        Android:text=""
        Android:id="@+id/content"
        Android:layout_weight="1.0"
    />
    <Button
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:text="返回"
        Android:id="@+id/back"
    />
</LinearLayout>
```

 代码解释:

- LinearLayout 有两个组件，一个是文本框，另一个是按钮。

12.5 本章小结

这个例子是我们学到的第一个和网络交互的实例。通过这个例子的学习，我们了解了什么是 RSS，什么是 SAX。并且学习了在 Android 系统中如何用 SAX 的方式解析 RSS 的数据。如果读者感兴趣，完全可以完善这个例子，做出一个真正的博客阅读器出来。



第 13 章 Android 综合案例二—— 基于 Google Map 开发个人移动地图

学习目标

- 学习 Google API 开发; ☐
- 掌握后台 Service 方式读取设备的 GPS 信息; ☐
- 掌握数据库操作; ☐
- 掌握在手机地图上显示位置信息应用; ☐
- 掌握 GPS 开发。 ☐

13.1 项目 UI 规划

通过前面基础知识的学习,对 Andriod 的结构已经熟悉了,为了更好的把前面学习的基础知识应用起来,下面将实现一些和现实结合紧密的项目应用实例。本章将逐步完成一个个人位置跟踪、监视、展示的应用程序。本实例实现的主要功能是,记录用户的移动轨迹,并随时显示在 Google Map 上。

本实例大概的实现思路是,应用程序运行后,在其主界面上可以选择创建一个新的跟踪,然后后台启动一个 Service,定时读取 GPS 数据获得用户目前所在的位置信息,将其存入数据库中;用户可以选择以往的跟踪记录,将其轨迹显示在 Map 上,由此可以记录用户去过的位置。

通过上面的思路讲解,我们可以把这个应用的实现分成如下几个步骤:

- 规划应用 UI;
- 设计存储,创建数据库、表;
- 实现跳转;
- 实现 Service;
- 完善权限设置 ;
- 程序演示;
- 程序打包、签名、发布。

下面一步一步来实现它。

规划应用 UI 其实就是对将要实现的应用做一个比较细致的梳理,实现的一般步骤可以参考本书的第 11 章,这里就不再赘述了。

针对将要实现的这个应用来看,一般应该考虑清楚需要哪些界面,也就是应用的大致工作流

程，图 13-1 所示是笔者画在纸上的项目框架草图。

main 是应用的主界面，用来在上面显示最近 10 次的跟踪记录，所以，其上需要一个列表组件 (list)，还需要添加几个菜单，菜单的作用分别是“新建跟踪”、“个人设置”、“帮助信息”。选择“新建跟踪”菜单后将出现一个新的界面，在其上需要输入跟踪的名字和一段描述，输入完成后单击确定，系统保存并开始启动跟踪服务。而在主界面的列表中，选择任何一个跟踪时，可以打开其展示界面，在其展示界面上，需要显示一副地图，并在地图上标识出其对应的跟踪点，而用其菜单可以查看此次跟踪的一些信息，还可以选择继续跟踪、删除等系列操作。除此之外，还给用户提供了一个个性化设置的界面，在此界面上，用户可以设定其 GPS 更新的频率，以及其地图显示的默认缩放等级等。最后，提供一些应用的帮助信息，例如，软件的使用帮助、作者及其联系方式等，所以，还要设置一个“帮助信息”界面，把上述规划用图描述出来，如图 13-1 所示。



▲ 图 13-1 界面规划

至此，将这个应用的功能和 workflow 梳理了一遍，接下来，需要梳理一下需要存储的数据，以及数据的存储方式。

13.2 数据存储实现

数据存储方式有几种，分别是可以把数据存储到文件系统中，或者数据库中。为了以后扩展方便，这里选择数据库为存储方式。在 Android 上，可以非常方便的使用 SQLite3 存储数据，关于 SQLite 的介绍，在前面的第 8 章已经做了介绍，读者可以参考。

依照我们的分析，这个程序需要保存 3 类数据，一类是跟踪名称，另外一类是每次跟踪过程中读取到的位置信息，还有一类是应用的配置信息。前两种比较容易理解，就不多说了，配置信息指的是一些可以配置的选项，如多久更新一次 GPS 信息、地图的默认级别等。前面两种数据会随着使用时间越来越多，这里设计将其存在数据库中；而配置信息比较少，且不是经常改变的，设计将其存储在 SharedPreferences 中。

13.2.1 设计数据库及表结构

按照上述的需要分析，数据库和表的结构设计如表 13-1 和表 13-2 所示。

表 13-1 Tracks 表的结构描述

属 性	类 型	含 义	备 注
id	INTEGER	主键 ID	
name	text	名字	
desc	text	描述	描述文字
distance	long	距离	起点到终点的距离

续表

属 性	类 型	含 义	备 注
tracked_time	Long	已经跟踪的时间	
locats_count	INTEGER	跟踪点数	
created_at	INTEGER	创建时间	
updated_at	INTEGER	更新时间	
avg_speed	long	平均速度	
max_speed	long	最大速度	

表 13-2 locats 表的结构描述

属 性	数 据 类 型	含 义	备 注
id	INTEGER	主键	
track_id	INTEGER	跟踪 ID	外键
longitude	text	维度	
latitude	TEXT	经度	
altitude	text	偏差	
created_at	INTEGER	创建时间	

13.2.2 设计 SharedPreferences 存储

关于 SharedPreferences 的使用，可以参考第 8 章，这里只规划一下需要存储的数据，针对此案例，需要存储的有 GPS 的更新频率、地图的默认缩放等级等，其他类似的数据等需要的时候再添加。

至此，基本上理清了需要的数据，并选择了各种数据的存储方式，接下来就可以实现整个应用了。

13.3 项目实施流程

前面两节讲的是项目规划，已经比较清晰整个系统的主要功能和需要的数据了，下面开始编程实现了。在本小节将实现整个系统的运行流程，使其可以在手机上运行起来。



注意

本项目实际例子中涉及的代码比较多，为了保证本书的质量，不会把全部代码放在书上，读者可以在附带光盘和笔者的网站上下载本书的全部代码。

13.3.1 创建项目工程

依照前面的讲解，这个步骤读者都很熟悉了。打开 Eclipse，依次选择 File→New→Android Project 项，输入项目名称 iTracks 和其他的辅助信息，如图 13-2 所示。

13.3.2 项目各功能及界面实现

接下来，需要依次实现上面规划的几个界面，按照应用运作的流程，下面依次实现：

- 主界面；
- 新建跟踪界面；
- 跟踪展示界面；
- 个性化设置界面；
- 帮助信息界面；
- 地图展示界面。

1. 实现主界面

按照我们前面的 UI 规划，需要在主界面展示一个“以往跟踪”的列表，首先在 `res/value` 目录下的 `strings.xml` 文件中添加一些需要用到的字符串的定义，实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">最近记录:</string>
    <string name="app_name">iTracks</string>

    <string name="menu_new">新建跟踪</string>
    <string name="menu_con">继续跟踪</string>
    <string name="menu_setting">个人设置</string>
    <string name="menu_helps">帮助信息</string>
    <string name="menu_exit">退出</string>
</resources>
```

 代码解释：

上述代码中定义了 `title`、`app_name` 等字符串，我们后面可以引用。

修改 `main` 的实现代码如下所示：

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

    <ListView android:id="@id/android:list" android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:layout_weight="0"
        android:drawSelectorOnTop="false" />
</LinearLayout>
```



▲ 图 13-2 创建项目工程

 代码解释:

上述就是 main.xml 的全部代码,其使用 LinearLayout 布局,放置一个 TextView 组件和 ListView 组件,其中 TextView 组件显示的文字引用为“@string/title”,也就是我们前面在 strings.xml 中定义的字符,其值也就是“最近记录:”;下面一个 id 为“@id/android:list”的 ListView 组件,其用来显示最近跟踪的列表。

看一下 onCreate 的实现代码,具体如下所示:

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

 代码解释:

上述代码中,也就是 onCreate 方法中,需要将以往的跟踪记录从数据库中查出,显示在列表中,我们这里调用 render_tracks()方法,这个方法负责从数据库中取出以往的记录,并更新到列表中。其实现方法会在后面完善,现在只需要记住这个方法。

 小提醒

为什么将查询数据库获取记录,更新列表的过程提取出来,主要是为了在新建、删除记录后可以很方便的调用,以便更新列表。

下面实现添加需要的菜单,关键代码如下所示:

```
//定义菜单需要的常量
private static final int MENU_NEW = Menu.FIRST + 1;
private static final int MENU_CON = MENU_NEW + 1;
private static final int MENU_SETTING = MENU_CON + 1;
private static final int MENU_HELPS = MENU_SETTING + 1;
private static final int MENU_EXIT = MENU_HELPS + 1;

// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    //如下添加几个菜单
    menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
        R.drawable.new_track).setAlphabeticShortcut('N');
    menu.add(0, MENU_CON, 0, R.string.menu_con).setIcon(
        R.drawable.con_track).setAlphabeticShortcut('C');
    menu.add(0, MENU_SETTING, 0, R.string.menu_setting).setIcon(
        R.drawable.setting).setAlphabeticShortcut('S');
    menu.add(0, MENU_HELPS, 0, R.string.menu_helps).setIcon(
        R.drawable.helps).setAlphabeticShortcut('H');
    menu.add(0, MENU_EXIT, 0, R.string.menu_exit).setIcon(
        R.drawable.exit).setAlphabeticShortcut('E');
    return true;
}

// 当一个菜单被选中的时候调用
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```

Intent intent = new Intent();
switch (item.getItemId()) {
case MENU_NEW:
intent.setClass(iTracks.this, NewTrack.class);
startActivity(intent);
return true;
case MENU_CON:
//TODO: 继续跟踪选择的记录
return true;
case MENU_SETTING:
intent.setClass(iTracks.this, Setting.class);
startActivity(intent);
return true;
case MENU_HELPS:
intent.setClass(iTracks.this, Helps.class);
startActivity(intent);
case MENU_EXIT:
finish();
break;
}
return true;
}

```

代码解释:

如上代码都是我们比较熟悉的知识,其主要用来创建菜单以及菜单被选中后的响应方法,关于菜单相关的详细介绍,可以在本书第7章找到。需要注意的是,上述代码中加粗的部分是尚未实现的,这对应着前面讲到的几个活动。按照以前添加活动的方法,依次添加 **NewTrack**、**Setting** 和 **Helps** 这几个 **Activity**。

接着还需要看一个比较重要的方法 **onItemClick**,它的作用是,当在列表中单击一个条目的时候被调用,实现代码如下所示:

```

public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
Log.i(TAG, "item clicked! [" + track.getItem(position).getName() + "]");
Intent itemintent = new Intent(this, iTrack.class); //创建一个信息的 Intent
Bundle b_track = new Bundle(); //创建一个 Bundle 用来传递数值
//然后将我们需要的数值放在 Bundle 中
b_track.putString("id", track.getItem(position).getId());
b_track.putString("name", track.getItem(position).getName());
b_track.putString("desc", track.getItem(position).getDesc());
itemintent.putExtra("android.intent.extra.album", b_track);
startActivityForResult(itemintent, 0); //启动新的活动
}

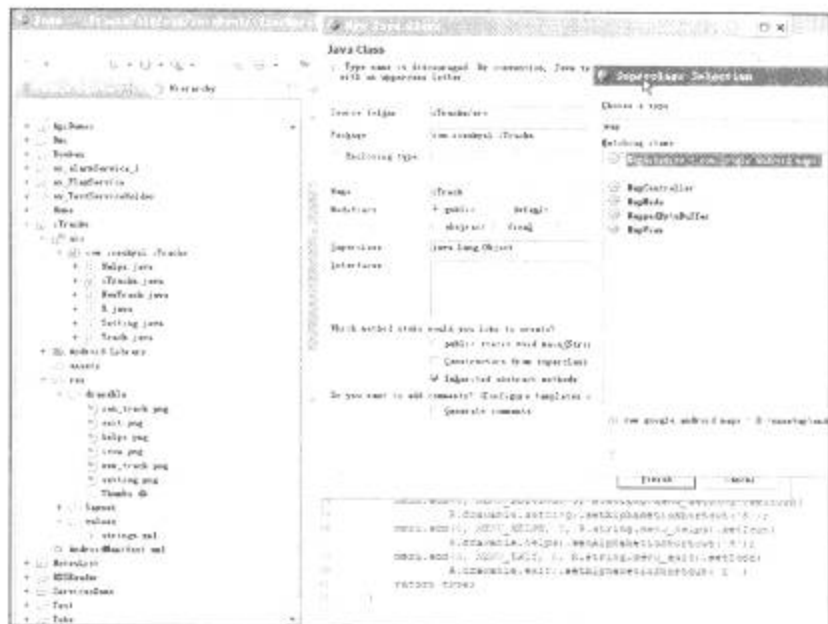
```

代码解释:

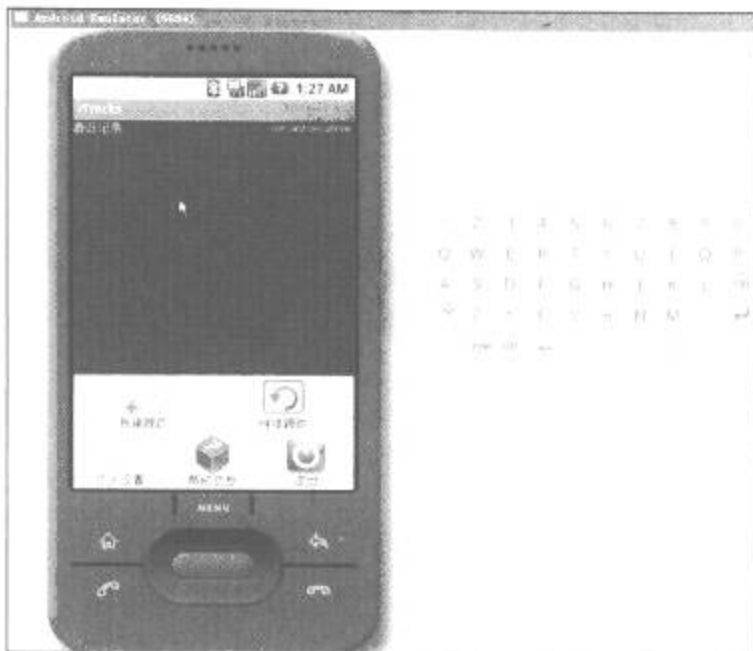
这里的处理逻辑是获取被单击条目的 ID、Name 和 Desc 属性,以绑定 (**Bundle**) 的方式传递给下一个活动,这里也就是实现单击以往跟踪记录条目时,打开一个新的活动,将路径展示在地图上。所以,这里将新建一个名字为 **iTrack** 的 **Activity**。需要注意的是,将要在这个 **Activity** 上显示一副地图,在 **Android** 中,可以非常方便的集成 **Google Map** 的功能,具体实现后面会讲到,这里要注意这个 **Activity** 需要继承自 **MapActivity**,如图 13-3 所示。

至此,基本上完成了第一个界面(主界面)的显示和跳转逻辑,运行结果如图 13-4 所示。

下面依次实现“新建跟踪”、“个人设置”、“展示跟踪”、“帮助信息”界面。



▲ 图 13-3 创建继承自 MapActivity 的 Activity



▲ 图 13-4 主界面展示

2. 实现新建跟踪界面

当在主界面的菜单选择“新建跟踪”时，将执行下面这段代码：

```
Intent intent = new Intent();
switch (item.getItemId()) {
case MENU_NEW:
intent.setClass(iTracks.this, NewTrack.class);
startActivity(intent);
return true;
}
```

 代码解释:

这段代码运行后会打开“新建跟踪”(NewTrack)对应的 Activity。按照前面的规划,在这个页面上输入跟踪的名字和一段描述后,单击确定将会创建一个新的跟踪,于是新建一个 layout 布局文件 new_track.xml, 其实现代码如下所示:

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

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

```
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/new_name" />
<EditText android:id="@+id/new_name"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="" />
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/new_desc" />
<EditText android:id="@+id/new_desc"
android:layout_width="fill_parent"
```



```

        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:scrollbars="vertical"/>
        <Button android:id="@+id/new_submit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/new_submit" />
    </LinearLayout>

```

✎ 代码解释：

在 `new_track.xml` 中，我们也是采用 `LinearLayout` 方式的布局，并放置 `TextView` 显示提示信息和 `EditText` 用来接收用户的输入，最后使用一个 `Button` 组件来提交用户输入的信息。

然后，打开 `NewTrack.java` 文件，首先申明一些后面需要用到的变量，实现代码如下所示：

```

private static final String TAG = "NewTrack";
private Button button_new;
private EditText field_new_name;
private EditText field_new_desc;

```

✎ 代码解释：

上述代码中 `TAG` 用来记录日志使用的标记，`button_new` 用来绑定模板中的“`@+id/new_submit`”，`field_new_name` 和 `field_new_desc` 用来绑定模板中输入名字和描述信息的组件。

然后在其 `onCreate` 方法中设定其关联的 `layout`，实现代码如下所示：

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.new_track);
    setTitle(R.string.menu_new);
    findViews();
    setListeners();
}

```

✎ 代码解释：

在 `onCreate()` 方法中，首先调用 `setContentView` 方法设定其对应的布局模板为 `new_track`（也就是 `new_track.xml`），然后调用 `setTitle()` 设定其标题，接着调用 `findViews()` 获取模板中的组件，并使用 `setListeners()` 对其中的按钮设置监听器。

上述代码中的 `findViews` 方法是用来查找模板中需要用到的控件的，其实现代码如下所示：

```

private void findViews() {
    Log.d(TAG, "find Views");
    field_new_name = (EditText) findViewById(R.id.new_name);
    field_new_desc = (EditText) findViewById(R.id.new_desc);
    button_new = (Button) findViewById(R.id.new_submit);
}

```

✎ 代码解释：

上述代码分别调用 `findViewById()` 方法获取名字和描述的 `EditText` 组件，并获取其提交按钮。

`onCreate` 方法中的 `setListeners` 方法用来设定监听器，这里只需要设定 `button_new` 的单击事件即可，实现代码如下所示：

```

private void setListeners() {
    Log.d(TAG, "set Listeners");
}

```

```
button_new.setOnClickListener(new_track);
}
```

 代码解释:

上述代码直接调用 Button 的 `setOnClickListener()` 方法设置其单击的响应监听器为 `new_track`。

上面设定其监听器为 `new_track`，其监听器的实现逻辑代码如下：

```
private Button.OnClickListener new_track = new Button.OnClickListener() {
    public void onClick(View v) {
        Log.d(TAG, "onClick new_track..");
        try {
            String name = (field_new_name.getText().toString());
            String desc = (field_new_desc.getText().toString());
            if (name.equals("")) {
                Toast.makeText(NewTrack.this,
                    getString(R.string.new_name_null),
                    Toast.LENGTH_SHORT).show();
            } else {
                // TODO 调用存储接口保存到数据库并启动 Service
            }
        } catch (Exception err) {
            Log.e(TAG, "error: " + err.toString());
            Toast.makeText(NewTrack.this, getString(R.string.new_fail),
                Toast.LENGTH_SHORT).show();
        }
    }
};
```

 代码解释:

这里首先定义一个 `Button.OnClickListener()` 对象，然后实现其 `onClick()` 方法，在本例中，在其 `onClick()` 方法，如果名字是空的，则使用 `Toast` 显示一段提醒文字；如果不是空的话，则要注意其中标识为 `TODO` 的地方，需要将接收到的名字和描述信息存入数据库，并启动 GPS 信息读取 `Service`，这需要等后面加以完善。其演示效果如图 13-5 所示。

3. 实现个人设置界面

当选择主界面上的“个人设置”菜单时，会相应的执行下面这段代码：

```
case MENU_SETTING:
    intent.setClass(iTracks.this, Setting.class);
    startActivity(intent);
    return true;
```

 代码解释:

上述代码在选择“`MENU_SETTING`”的时候被执行，其中设置下一个需要打开的界面是 `Setting`，并使用 `startActivity` 启动它。

上述代码也就是转向个人设置的 `Activity`，这里就需要实现这个功能。按照前面的规划，这里可以设置的是一些应用的配置信息，如 GPS 更新的速度、Map 的默认等级等。

前面讲过，这类配置数据不会很多，也不是经常改变，但是每次系统启动都需要使用。于是这



▲ 图 13-5 新建跟踪界面

里选择用 SharedPreferences 来存储。

首先，还是按照前面的 UI 规划，创建这个 Activity 需要的布局文件，我们这里将其命名为 setting.xml，然后编程实现，具体代码如下所示：

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

    <TextView android:id="@+id/setting_tips"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />

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

    <Spinner android:id="@+id/setting_gps"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="true"
        android:prompt="@string/spinner_gps_prompt"
    />

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

    <Spinner android:id="@+id/setting_map_level"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="true"
        android:prompt="@string/spinner_map_prompt"
    />

    <Button android:id="@+id/setting_submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/setting_submit" />

</LinearLayout>
```

代码解释：

上述代码对应其布局文件 setting.xml，其使用的也是 LinearLayout 布局模式，并使用 Spinner 组件提供一个供用户选择的下拉列表，最后还有一个 Button 用于提交用户选择的信息。

注意

这里使用了 Spinner，以方便用户可以选择输入，而不必自己填写，适用于选择项是可列举的情况下。

接着，编写 Setting.java 文件，先声明一些需要使用的变量，具体代码如下所示：

```
private static final String TAG = "Setting";
//定义菜单需要的常量
private static final int MENU_MAIN = Menu.FIRST + 1;
private static final int MENU_NEW = MENU_MAIN + 1;
private static final int MENU_BACK = MENU_NEW + 1;;

// 保存个性化设置
public static final String SETTING_INFOS = "SETTING_Infos";
```

```

public static final String SETTING_GPS = "SETTING_Gps";
public static final String SETTING_MAP = "SETTING_Map";
public static final String SETTING_GPS_POSITON = "SETTING_Gps_p";
public static final String SETTING_MAP_POSITON = "SETTING_Map_p";
private Button button_setting_submit;
private Spinner field_setting_gps;
private Spinner field_setting_map_level;

```

然后在 onCreate 中绑定 setting.xml 为其布局模板，实现代码如下所示：

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.setting);
    setTitle(R.string.menu_setting);
    findViews();
    setListeners();
    restorePrefs();
}

```

 代码解释：

上述代码首先使用 setContentView() 设定其对应的布局文件为 setting.xml，使用 setTitle() 设定其标题，进而调用 findViews() 找到我们需要操作的组件，并调用 setListeners() 给按钮设定单击监听器，最后调用 restorePrefs() 将默认值或者用户以前的选择值显示出来。

findViews() 用来找出需要使用的组件，实现代码如下所示：

```

private void findViews() {
    Log.d(TAG, "find Views");
    button_setting_submit = (Button) findViewById(R.id.setting_submit);
    field_setting_gps = (Spinner) findViewById(R.id.setting_gps);
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.gps, android.R.layout.simple_spinner_item);

    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    field_setting_gps.setAdapter(adapter);

    field_setting_map_level = (Spinner) findViewById(R.id.setting_map_level);
    ArrayAdapter<CharSequence> adapter2 = ArrayAdapter.createFromResource(
        this, R.array.map, android.R.layout.simple_spinner_item);
    adapter2.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    field_setting_map_level.setAdapter(adapter2);
}

```

 代码解释：

上述代码在视图中找到一个 Button 和两个 Spinner 组件，然后设定调用 Spinner 的 setDropDownViewResource 方法设定其下拉框内容，这里两个 Spinner 分别使用 R.array.gps 和 R.array.map 数据源，这两个数据源是预先定义好的，放在 value/array.xml 中的，其实现代码如下所示：

```

<resources>
    <!-- Used in View/setting.java -->
    <string-array name="gps">
        <item>5</item>
        <item>10</item>
        <item>15</item>
        <item>30</item>
        <item>60</item>
        <item>120</item>
    </string-array>

```

```

<string-array name="map">
    <item>1</item>
    <item>2</item>
    <item>3</item>
    <item>4</item>
    <item>5</item>
    <item>6</item>
</string-array>

</resources>

```

代码解释:

上述代码定义了两个列表 resources，名字分别为 gps 和 map，这两组值我们将在下拉列表中显示。
onCreate 方法中紧接着调用 setListensers 设定监听器，具体实现代码如下所示：

```

//设置按钮的单击监听器
private void setListensers() {
    Log.d(TAG, "set Listensers");
    button_setting_submit.setOnClickListener(setting_submit);
}

```

代码解释:

上述代码调用 setOnClickListener 方法给 button_setting_submit 组件设定单击的监听器为 setting_submit。
这里的监听器监听按钮被单击的事件，其实现方法如下：

```

private Button.OnClickListener setting_submit = new Button.OnClickListener() {
    public void onClick(View v) {
        Log.d(TAG, "onClick new_track..");
        try {
            String gps = (field_setting_gps.getSelectedItem().toString());
            String map = (field_setting_map_level.getSelectedItem().toString());
            if (gps.equals("") || map.equals("")) {
                Toast.makeText(Setting.this,
                    getString(R.string.setting_null),
                    Toast.LENGTH_SHORT).show();
            } else {
                //保存设定
                storePrefs();
                Toast.makeText(Setting.this,
                    getString(R.string.setting_ok),
                    Toast.LENGTH_SHORT).show();
                //跳转到主界面
                Intent intent = new Intent();
                intent.setClass(Setting.this, iTracks.class);
                startActivity(intent);
            }
        } catch (Exception err) {
            Log.e(TAG, "error: " + err.toString());
            Toast.makeText(Setting.this, getString(R.string.setting_fail),
                Toast.LENGTH_SHORT).show();
        }
    }
};

```

代码解释:

首先创建一个 Button.OnClickListener() 对象 setting_submit，进而实现其 onClick() 方法，在 onClick() 中，使用 getItem() 获得选择的条目，进行一些基本的判断，如果信息不完整，则使用 Toast 显示预先定义的提示消息。如果没有问题，则使用 storePrefs() 进行存储，storePrefs() 实现代码如下：


```
//保存个人设置
private void storePrefs() {
    Log.d(TAG, "storePrefs setting infos");
    //获取 SharedPreferences 对象
    SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0);
    //存储用户选择的值
    settings.edit()
        .putString(SETTING_GPS, field_setting_gps.getSelectedItem().toString())
        .putString(SETTING_MAP, field_setting_map_level.getSelectedItem().toString())
        .putInt(SETTING_GPS_POSITON, field_setting_gps.getSelectedItemPosition())
        .putInt(SETTING_MAP_POSITON, field_setting_map_level.getSelectedItemPosition())
        .commit();
}
```

代码解释:

上述代码开始创建一个 SharedPreferences 对象 settings，进而调用其 edit() 方法使其可以编辑，然后将用户选择的值存储进去。

这里就是使用 SharedPreferences 来存储配置信息，需要注意的是，除了保存选择的配置，还保存了其配置在列表中的位置，以方便后面使用。

在 onCreate 中还有一个过程 restorePrefs，它负责将以前的设定显示在下拉选择的默认位置上，其实现代码如下所示：

```
// 恢复 preferences 中存储的内容
private void restorePrefs() {
    SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0);
    int setting_gps_p = settings.getInt(SETTING_GPS_POSITON, 0);
    int setting_map_level_p = settings.getInt(SETTING_MAP_POSITON, 0);
    Log.d(TAG, "restorePrefs: setting_gps=" + setting_gps_p + ",setting_map_level=" +
        setting_map_level_p);
    //如果值存在，就把其设定给指定的组件
    if (setting_gps_p != 0 && setting_map_level_p != 0) {
        field_setting_gps.setSelection(setting_gps_p);
        field_setting_map_level.setSelection(setting_map_level_p);
        button_setting_submit.requestFocus();
    } else if (setting_gps_p != 0) {
        field_setting_gps.setSelection(setting_gps_p);
        field_setting_map_level.requestFocus();
    } else if (setting_map_level_p != 0) {
        field_setting_map_level.setSelection(setting_map_level_p);
        field_setting_gps.requestFocus();
    } else {
        field_setting_gps.requestFocus();
    }
}
```

代码解释:

上述代码比较容易理解，从 SharedPreferences 中取得配置信息所在其列表选择的位置，然后使用 setSelection 来预先设定其为下拉列表的默认值。

最后，还需要实现菜单和菜单对应的执行过程，实现创建菜单和选择菜单的代码如下：

```
// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(
        R.drawable.icon).setAlphabeticShortcut('M');
    menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
```

```

R.drawable.new_track).setAlphabeticShortcut('N');
menu.add(0, MENU_BACK, 0, R.string.menu_back).setIcon(
R.drawable.back).setAlphabeticShortcut('E');
return true;
}
// 当一个菜单被选中的时候调用
@Override
public boolean onOptionsItemSelected(MenuItem item) {
Intent intent = new Intent();
switch (item.getItemId()) {
case MENU_NEW: //新建跟踪
intent.setClass(Setting.this, NewTrack.class);
startActivity(intent);
return true;
case MENU_MAIN: //主界面
intent.setClass(Setting.this, iTracks.class);
startActivity(intent);
return true;
case MENU_BACK: //返回
finish();
break;
}
return true;
}

```

代码解释:

上述代码初始化 3 个菜单, 分别是跳转至“主界面”、“新建跟踪”和“返回”界面, 其各自的实现都非常容易了, 参考以上代码, 就不赘述了。最后一个小技巧, 在程序退出的时候, 也顺便保存一下设置, 实现代码如下所示:

```

@Override
protected void onStop() {
super.onStop();
Log.d(TAG, "save setting infos");
// 保存用户的个性化设置
storePrefs();
}

```

代码解释:

onStop()方法在活动停止的时候被调用, 上述代码实现在退出时调用 storePrefs()方法存储选择的值。

完成了上述代码后, 运行程序看一下实现的界面效果, 如图 13-6 所示。

4. 帮助信息界面

如果在主界面上选择了帮助信息“MENU_HELPS”项, 则对应的会执行如下这段代码:

```

case MENU_HELPS:
intent.setClass(iTracks.this, Helps.class);
startActivity(intent);

```

代码解释:

上述代码设定 Helps 为目标 Activity, 并调用 startActivity()启动这个活动。

所以, 下面继续实现 Helps 这个界面。首先创建一个名为 helps.xml 的布局文件, 按照前面的



▲ 图 13-6 个人设置页面

UI 规划, 这个界面较简单, 需要在这个界面显示版本信息、帮助信息和开发者信息。所以, 只需要放置几个 `TextView` 组件即可, 其实现代码如下所示:

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

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:autoLink="all"
        android:text="@string/helps_text"
        />

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:autoLink="all"
        android:text="@string/author_text"
        />
</LinearLayout>
```

代码解释:

使用 `LinearLayout` 布局模板, 其中放置一些用来显示帮助信息的 `TextView`, 其各个 `TextView` 显示的文字我们将在另外文件中定义。

接着新建一个 XML 文件存放这些信息, 如这里新建一个名为 `helps.xml` 的文件。其代码内容为:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="version_text">Ver 1.0.0</string>
    <string name="helps_text">
```

这个应用用来读取用户手机上的 GPS 信息, 并将其显示在 Map 上, 可以建立多个跟踪, 可以设定个人的配置信息: \n

在主界面上单击菜单，选择需要的动作...

```
</string>
<string name="author_text">Name:IceskYsl \nEmail:Iceskysl@gmail.com \nBlog:http:
//iceskysl.lsters.com</string>

</resources>
```

代码解释：

上述代码定义了我们前面需要的变量，我们将这个文件放在项目的目录 res/values 下。

接着打开 Helps.java 文件，实现其需要的代码逻辑。首先在 onCreate 方法中设定其对应的布局文件为 helps.xml，实现代码如下所示：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.helps);
    setTitle(R.string.menu_helps);
}
```

代码解释：

上述代码设定其布局文件是 helps.xml，并设定标题。

最后添加菜单和菜单功能，和前面个人设置界面实现的一样，在此不再赘述。最后看一下实现的界面，如图 13-7 所示。

5. 地图展示界面

前面实现了这个应用的其他几个界面，下面将要讲解最复杂、最有趣的一个界面的实现，这里将要实现在 Android 中展示地图，并将指定的点标记出来。Android 平台也提供了一个 map 包 (com.google.android.maps)，通过其中的 MapView 就能够方便的利用 Google 地图的资源来进行编程，但是也得按照其预定的流程操作，实现流程如下所示。

(1) 准备工作。

在利用 MapView 之前，必须要先申请一个 Android Map API Key，步骤为首选确定用户使用的 keystore 文件，然后获得其 MD5 指纹，再去 Googlemap 网站申请 API Key，详细步骤如下。

<步骤 1>找到用户的 debug.keystore 文件，开发使用的此文件通常位于下面这个目录：

C:\Documents and Settings\你的当前用户\Local Settings\Application Data\Android

<步骤 2>获取 MD5 指纹。

运行 cmd.exe，执行 keytool -list -alias androiddebugkey -keystore 命令，获取 MD5 指纹。

请输入如下命令 keytool -list -alias androiddebugkey -keystore C:\Documents and Settings\你的当前用户\Local Settings\Application Data\Android\debug.keystore

系统会提示输入 keystore 密码，这时候输入 Android，系统就会输出类似于“1D:67:43:7C:14:2E:



▲ 图 13-7 帮助信息界面

CC: CA: 35:7B:1F:93:A7:91:AD:45” 的字符串, 这就得到了 MD5 认证指纹, 如图 13-8 所示。

提醒

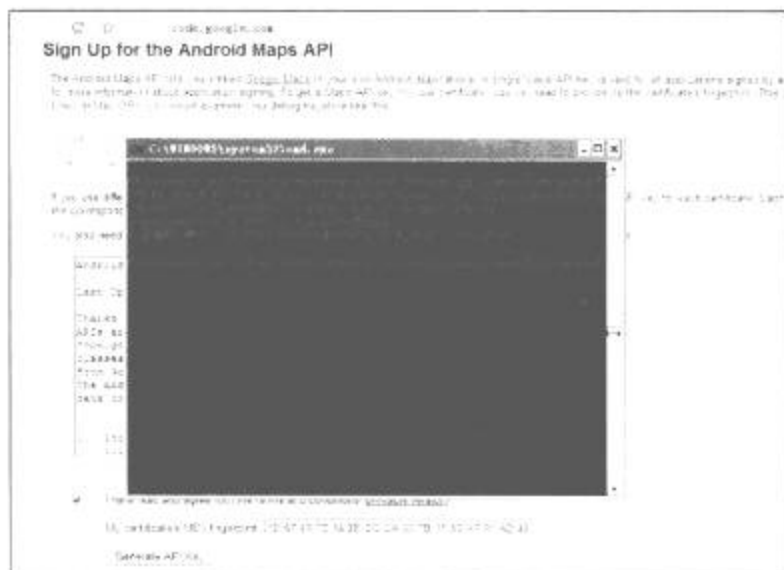
如果系统提示找不到 keytool, 则需要将 Java 的 bin 目录加入环境变量中, 或者使用 keytool 的绝对路径 (其在 Java 目录下), 如 C:\Documents and Settings\Administrator\Local Settings\Application Data\Android>

```
"C:\Program Files\Java\jdk1.6.0_10\bin\keytool.exe" -list -alias
androiddebugkey
-keystore debug.keystore -storepass android -keystore android
```

<步骤 3>申请 Android map 的 API Key。

打开浏览器, 输入网址 <http://code.google.com/intl/zh-CN/android/maps-api-signup.html>,

在 Google 的 Android map API Key 申请页面上输入步骤 2 得到的 MD5 认证指纹, 单击 Generate API Key 按钮, 即可转到图 13-9 所示的界面, 获取申请到的 API Key。



▲ 图 13-8 MD5 认证指纹



▲ 图 13-9 得到 API Key

下一小节开始将按照前面的 UI 规划来逐步利用 MapView 实现它。

(2) 编程实现。

在前面已经创建了一个基于 MapActivity 的 Activity——ShowTrack, 开发的一般思路, 首先按照最开始的 UI 设计来编写布局文件。创建一个名为 show_track.xml 的布局文件, 用来实现放大、缩小, 方向移动、地图模式切换等几个按钮。另外, 还需要有一个用来显示地图的 View 组件, 其实现代码如下所示:

```
<view android:id="@+id/mv"
    class="com.google.android.maps.MapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    android:apiKey="01Yu9W3X3vbpYT3x33chPct97U1Z6jy8WYZXNFA"
/>
```

代码解释:

需要注意, 其 class 是 com.google.android.maps.MapView, 另外, 还需要将在前面准备工作中

申请到的 API Key 的值放在 “android:apiKey” 参数中。完整的布局代码如下：

```
<?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"
    >

    <view android:id="@+id/mv"
        class="com.google.android.maps.MapView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:apiKey="01Yu9W3X3vbpYT3x33chPct97U1Z6jy8WYZXNFA"
        />

    <LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#550000ff"
        android:padding="1px"
        >

        <Button android:id="@+id/sat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="40px"
            android:text="@string/satellite" />

        <Button android:id="@+id/traffic"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/traffic" />

        <Button android:id="@+id/streetview"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/street" />

        <Button android:id="@+id/gps"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="GPS" />
    </LinearLayout>

    <LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:background="#550000ff"
        android:padding="1px"
        >

        <Button android:id="@+id/zin"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="30px"
            android:text="+" />
        <Button android:id="@+id/zout"
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="-" />

<Button android:id="@+id/pann"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="N" />

<Button android:id="@+id/pane"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="E" />

<Button android:id="@+id/panw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="W" />

<Button android:id="@+id/pans"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="S" />

</LinearLayout>

</FrameLayout>

```

代码解释：

上述代码中，我们这里选择 FrameLayout 布局，MapView 组件用来显示地图，除此之外，还放置了很多按钮组件，用来控制地图，如放大、缩小，左移、右移、上移、下移，以及卫星、地图、街区模式的切换等。

除此之外，还有需要注意的是，因为 map 包并不是 androidSDK 内置的，所以使用的时候必须申明，其方式非常简单，在项目的 manifest 文件中的 application 标签内加入 `<uses-library android:name="com.google.android.maps" />` 即可，否则，会遇到 ClassNotFound 异常。

然后看一下 ShowTrack.java 文件，首先申明一些需要用到的变量，实现代码如下所示：

```

private static final String TAG = "ShowTrack";
private static MapView mMapView;
private MapController mc;
protected MyLocationOverlay mOverlayController;
private Button mZin;
private Button mZout;
private Button mPanN;
private Button mPanE;
private Button mPanW;
private Button mPanS;
private Button mGps;
private Button mSat;
private Button mTraffic;
private Button mStreetview;
private String mDefCaption = "";
private GeoPoint mDefPoint;

private LocationManager lm;
private LocationListener locationListener;

```

接着看一下 onCreate 方法，实现代码如下所示：

```
/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.show_track);
    findViews();
}
```

 代码解释：

上述代码中 findViews 用来确定需要使用的控件，并绑定其需要响应的事件，findViews()实现的代码如下所示：

```
private void findViews() {
    Log.d(TAG, "find Views");
    // 获取视图中的组件
    mMapView = (MapView) findViewById(R.id.mv);
    mMapView.getController().setZoom(10);

    // 设置缩小按钮
    mZin = (Button) findViewById(R.id.zin);
    mZin.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
            zoomIn();
        }
    });
    // 设置放大按钮
    mZout = (Button) findViewById(R.id.zout);
    mZout.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
            zoomOut();
        }
    });
    // 设置向北移动按钮
    mPanN = (Button) findViewById(R.id.pann);
    mPanN.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
            panNorth();
        }
    });

    // 设置向东移动按钮
    mPanE = (Button) findViewById(R.id.pane);
    mPanE.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
            panEast();
        }
    });

    // 设置向西移动按钮
    mPanW = (Button) findViewById(R.id.panw);
    mPanW.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
            panWest();
        }
    });
}
```

```

});
// 设置向南移动按钮
mPanS = (Button) findViewById(R.id.pans);
mPanS.setOnClickListener(new OnClickListener() {
    // @Override
    public void onClick(View arg0) {
        panSouth();
    }
});

//设置 GPS 视图按钮
mGps = (Button) findViewById(R.id.gps);
mGps.setOnClickListener(new OnClickListener() {
    // @Override
    public void onClick(View arg0) {
        centerOnGPSPosition();
    }
});
//设置卫星模式视图按钮
mSat = (Button) findViewById(R.id.sat);
mSat.setOnClickListener(new OnClickListener() {
    // @Override
    public void onClick(View arg0) {
        toggleSatellite();
    }
});

// 设置交通模式视图按钮
mTraffic = (Button) findViewById(R.id.traffic);
mTraffic.setOnClickListener(new OnClickListener() {
    // @Override
    public void onClick(View arg0) {
        toggleTraffic();
    }
});
// 设置街景模式视图按钮
mStreetview = (Button) findViewById(R.id.streetview);
mStreetview.setOnClickListener(new OnClickListener() {
    // @Override
    public void onClick(View arg0) {
        toggleStreetView();
    }
});

//使用“位置管理器”获取 GPS 位置变化信息
lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

locationListener = new MyLocationListener();

lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
}

```

小知识

什么是 LocationManager?

Android 中 LocationManager 提供了一系列方法来处理地理位置相关的问题,包括查询上一个已知位置;注册/注销来自某个 LocationProvider 的周期性的位置更新;以及注册/注销接近某个坐标时对一个已定义 Intent 的触发等。

结合代码中的注释说明,此代码比较好理解,这里需要重点说明的是:

```
mMapView = (MapView) findViewById(R.id.mv);
mc = mMapView.getController();
mc.setZoom(10);
```

 代码解释：

第一个语句是获取布局中的 MapView，然后使用 getController 得到一个 MapController，这样就可以对这个 MapController 做一些类似于设定缩放等级（setZoom）的操作。

```
lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
```

 代码解释：

上述代码实现的比较巧妙的地方，注册一个基于 locationListener 的 MyLocationListener，使得其在 GPS 信息改变的时候能被捕获到。关于 MyLocationListener 的定义，后面会讲到。接下来就是实现这么多按钮的响应方法，先看 4 个方向键移动按钮的实现，实现代码如下所示：

```
public void panWest() {
    GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6(),
mMapView.getMapCenter().getLongitudeE6()
- mMapView.getLongitudeSpan() / 4);
    mc.setCenter(pt);
}

public void panEast() {
    GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6(),
mMapView.getMapCenter().getLongitudeE6()
+ mMapView.getLongitudeSpan() / 4);
    mc.setCenter(pt);
}

public void panNorth() {
    GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6()
+ mMapView.getLatitudeSpan() / 4, mMapView.getMapCenter()
.getLongitudeE6());
    mc.setCenter(pt);
}

public void panSouth() {
    GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6()
- mMapView.getLatitudeSpan() / 4, mMapView.getMapCenter()
.getLongitudeE6());
    mc.setCenter(pt);
}
```

 代码解释：

上述代码实现的原理都是一样的，首选是获取当前地图的中心，然后朝指定的方向移动 1/4 个距离。

接着看放大和缩小按钮的实现，具体代码如下所示：

```
public void zoomIn() {
    mc.zoomIn();
}

public void zoomOut() {
    mc.zoomOut();
}
```

 代码解释：

通过 MapController 中已经提供的方法实现。

接着是地图模式切换的实现，具体代码如下所示：

```
public void toggleSatellite() {
    mMapView.setSatellite(true);
    mMapView.setStreetView(false);
    mMapView.setTraffic(false);
}

public void toggleTraffic() {
    mMapView.setTraffic(true);
    mMapView.setSatellite(false);
    mMapView.setStreetView(false);
}

public void toggleStreetView() {
    mMapView.setStreetView(true);
    mMapView.setSatellite(false);
    mMapView.setTraffic(false);
}
```

 代码解释：

这里需要提醒的是，setStreetView(true)不是在哪里都可以展示，随着 Google Map 的不断完善，这个会越来越完善的。

接下来一个是比较有趣的方法，也就是单击“GPS”按钮的响应方法 centerOnGPSPosition，其目的是将地图定位到当前的 GPS 指定的位置，实现代码如下所示：

```
private void centerOnGPSPosition() {
    Log.d(TAG, "centerOnGPSPosition");
    String provider = "gps";
    LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

    Location loc = lm.getLastKnownLocation(provider);
    loc = lm.getLastKnownLocation(provider);

    mDefPoint = new GeoPoint((int) (loc.getLatitude() * 1000000),
        (int) (loc.getLongitude() * 1000000));
    mDefCaption = "I'm Here.";
    mc.animateTo(mDefPoint);
    mc.setCenter(mDefPoint);
    //将 Overlay 显示在地图上。
    MyOverlay mo = new MyOverlay();
    mo.onTap(mDefPoint, mMapView);
    mMapView.getOverlays().add(mo);
}
```

 代码解释：

这个方法开始使用 LocationManager 得到当前的 GPS 位置信息，然后将地图定位过去，并使用基于 Overlay 的 MyOverlay 将其显示在地图上。关于 MyOverlay 的实现，下面就会讲到。

 小提醒

由于这里使用了 GPS 和 GoogleMap，所以，需要在 manifest 中加入相应的权限，代码如下所示：

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

下面看一下前面使用到的 MyOverlay 的实现，具体代码如下所示：

```

// This is used draw an overlay on the map
protected class MyOverlay extends Overlay {
    @Override
    public void draw(Canvas canvas, MapView mv, boolean shadow) {
        Log.d(TAG, "MyOverlay::draw.mDefCaption=" + mDefCaption);
        super.draw(canvas, mv, shadow);

        if (mDefCaption.length() == 0) {
            return;
        }

        Paint p = new Paint();
        int[] scoords = new int[2];
        int sz = 5;

        // 转换为屏幕上的位置
        Point myScreenCoords = new Point();
        mMapView.getProjection().toPixels(mDefPoint, myScreenCoords);
        // 在屏幕上绘制点
        scoords[0] = myScreenCoords.x;
        scoords[1] = myScreenCoords.y;
        p.setTextSize(14);
        p.setAntiAlias(true);

        int sw = (int) (p.measureText(mDefCaption) + 0.5f);
        int sh = 25;
        int sx = scoords[0] - sw / 2 - 5;
        int sy = scoords[1] - sh - sz - 2;
        RectF rec = new RectF(sx, sy, sx + sw + 10, sy + sh);

        p.setStyle(Style.FILL);
        p.setARGB(128, 255, 0, 0);

        canvas.drawRoundRect(rec, 5, 5, p);

        p.setStyle(Style.STROKE);
        p.setARGB(255, 255, 255, 255);
        canvas.drawRoundRect(rec, 5, 5, p);

        canvas.drawText(mDefCaption, sx + 5, sy + sh - 8, p);

        p.setStyle(Style.FILL);
        p.setARGB(88, 255, 0, 0);
        p.setStrokeWidth(1);
        RectF spot = new RectF(scoords[0] - sz, scoords[1] + sz, scoords[0]
            + sz, scoords[1] - sz);
        canvas.drawOval(spot, p);

        p.setARGB(255, 255, 0, 0);
        p.setStyle(Style.STROKE);
        canvas.drawCircle(scoords[0], scoords[1], sz, p);
    }
}

```

代码解释:

Android 中的 Overlay 是个抽象类, 必须重载并实现其 draw 方法, 本例子中首先使用 toPixels 将 GPS 获得的 Geopoint 点转换为手机屏幕上的点坐标 myScreenCoords, 然后使用 Android 的 draw

系列画出需要的标识即可。

最后一个需要实现的是前面使用的 MyLocationListener，具体代码如下所示：

```
//
private class MyLocationListener implements LocationListener {
    @Override
    public void onLocationChanged(Location loc) {
        Log.d(TAG, "MyLocationListener::onLocationChanged..");
        if (loc != null) {
            Toast.makeText(
                getBaseContext(),
                "Location changed : Lat: " + loc.getLatitude()
                + " Lng: " + loc.getLongitude(),
                Toast.LENGTH_SHORT).show();
            mDefPoint = new GeoPoint((int) (loc.getLatitude() * 1000000),
                (int) (loc.getLongitude() * 1000000));
            mc.animateTo(mDefPoint);
            mc.setCenter(mDefPoint);
            //展示地图
            mDefCaption = "Lat: " + loc.getLatitude() + " Lng: " + loc.getLongitude();
            MyOverlay mo = new MyOverlay();
            mo.onTap(mDefPoint, mMapView);
            mMapView.getOverlays().add(mo);
        }
    }

    @Override
    public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
        // TODO Auto-generated method stub
    }
}
```

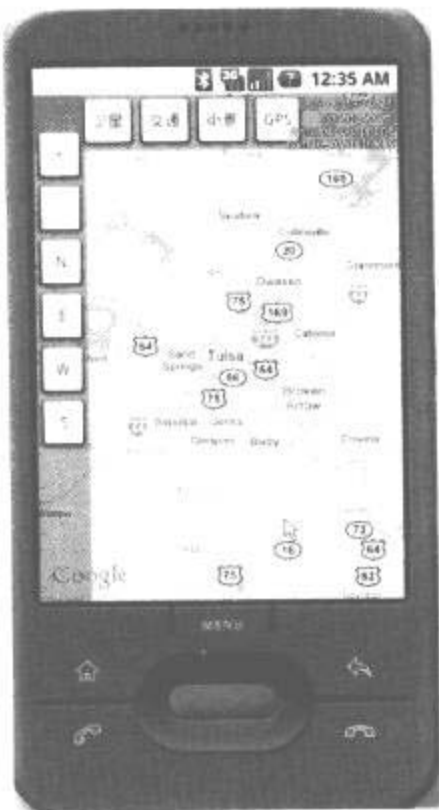
代码解释：

这里的 MyLocationListener 实现 LocationListener 定义好的接口，实现了一个 onLocationChanged 方法，当在 GPS 位置信息发生变化的时候被调用，将地图定位到新的位置，并使用 Toast 显示当前位置的坐标，并使用前面我们讲的 MyOverlay 将其显示在地图上。

（3）实例模拟演示。

上述程序完成以后，下面运行程序看看这个界面实现的效果，如果一切顺利，还需要继续往下编程，否则，就需要及时地停下来修复出现的错误。运行此应用程序，转到地图展示界面，看到界面效果如图 13-10 所示。

从图 13-10 中可以看到，界面和预想的差不多，其上摆放了一些控制按钮，可以单击界面上的“卫星”按钮，打开卫星视图，如图 13-11 所示。



▲ 图 13-10 地图展示界面



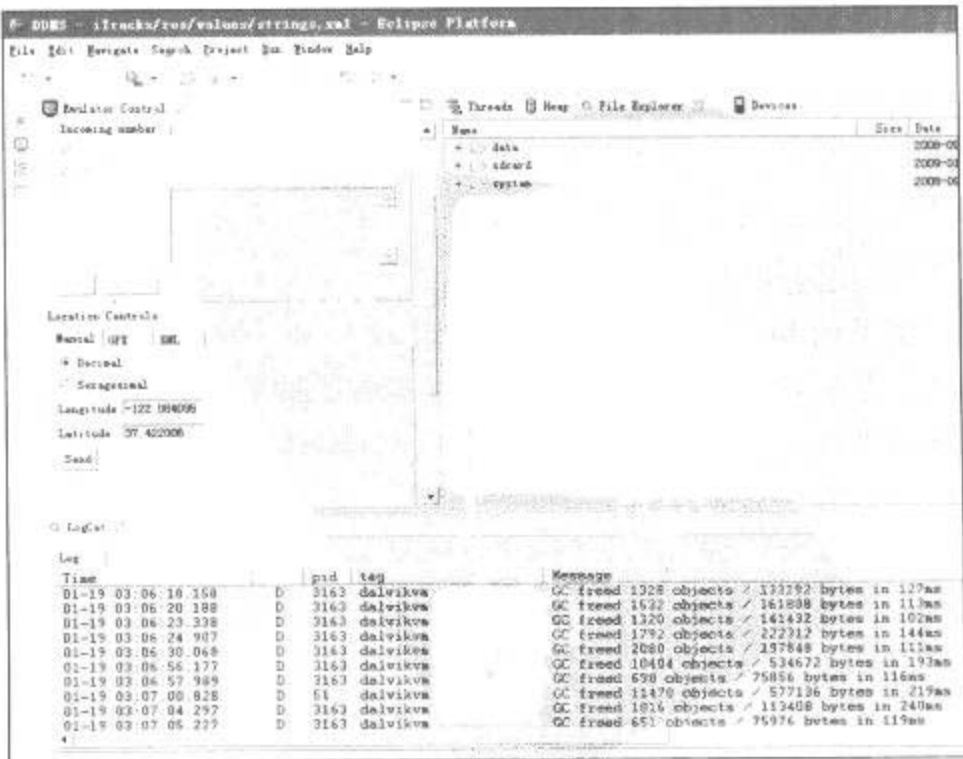
▲ 图 13-11 卫星视图界面

再单击界面上的放大按钮，其效果如图 13-12 所示。

通过运行的界面显示，以上实现的功能都达到了预期的目的，下面的工作就是需要重点实现的是，GPS 位置变化。在模拟器中模拟设定 GPS 信息，才可以应用得到该信息。比较好的是，在 Android SDK 开始，对应的 Eclipse 开发插件 DDMS 中已经可以模拟设定 GPS 信息了。打开 DDMS 的界面，在其左侧可以看到“Emulator Control”面板，面板下有“Location Controls”，在这里就可以设定 GPS 信息，如图 13-13 所示。



▲ 图 13-12 放大效果界面



▲ 图 13-13 Emulator Control 面板界面

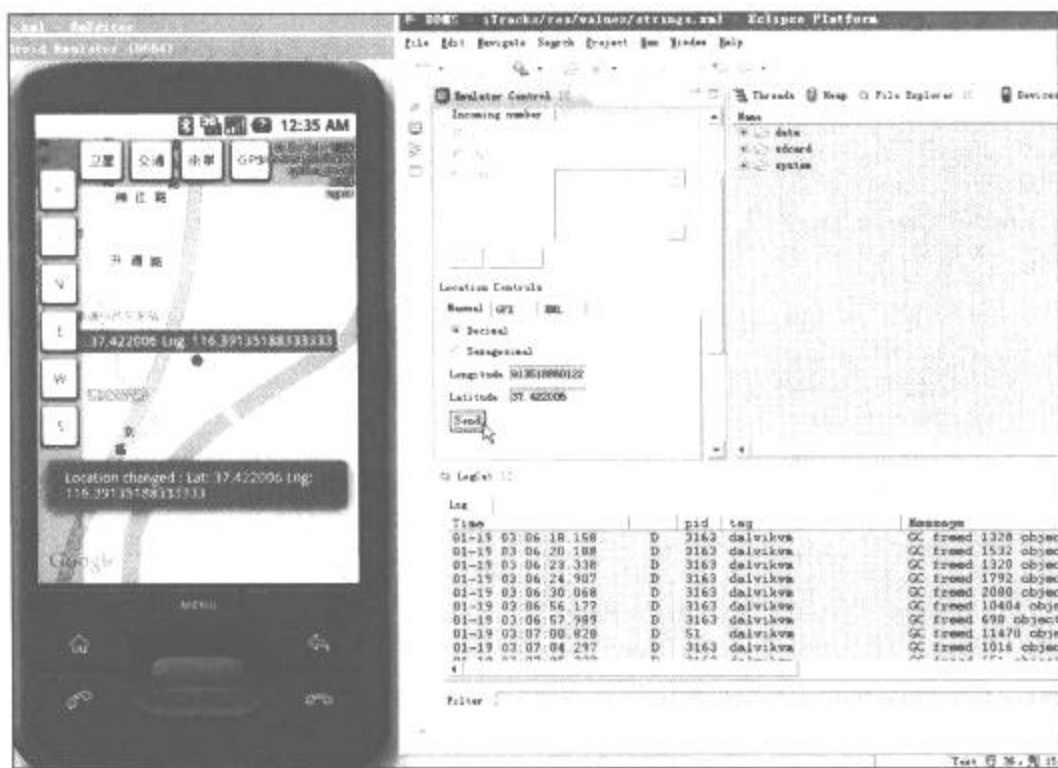
例如，我们这里设定 GPS 为北京天安门的位置 “<longitude>116.3913518860122</longitude>;<latitude>39.90528818766884</latitude>”，然后单击“Send”按钮即可。

然后试一下“GPS”按钮，按照我们预想，其将取得用户的 GPS 信息，然后将地图定位到那个信息给定的位置，并显示一个“I'm Here.”的标识，单击“GPS”按钮，效果如图 13-14 所示。

接着测试 GPS 位置发生变化时效果，输入一个不同的 GPS 信息，单击“send”按钮后，可以在界面上看到如图 13-15 所示的效果。



▲ 图 13-14 GPS 定位效果界面



▲ 图 13-15 GPS 位置改变界面

通过演示，程序顺利运行，想要的功能都可以正常实现，下面可以接着实现数据的存取。

13.3.3 实现数据存取

按照我们前面的设计规划，需要将每次跟踪时 GPS 位置的改变存放在数据库中，并能将其轨迹显示在地图上。这些都和数据的存取有关。

通过本书第 8 章的学习，应该了解到数据库操作的基本步骤和方法，在这里就不赘述了，如果有所遗忘，翻到前面章节熟悉一下。

前面已经将用户的个性化配置信息存放在 SharedPreferences 中，这里需要存取的数据是放在数据库中的。通过前面章节的学习知道，在 Android 中，存取数据库有两种方法，其一是通过 help 类继承 SQLiteDatabase 相关类绑定 SQL，其二是使用 ContentProvider 进行封装。因为实现此功能的逻辑不麻烦，可以自己封装实现。

1. 数据库准备

由于这里需要操作同一个数据库中的两个表，所以，首先创建一个名字为 DbAdapter 的类

DbAdapter.java, 在这个类中完成数据库中两个表的创建和升级等操作, 实现代码如下所示:

```
package com.iceskysl.iTracks;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DbAdapter {
    private static final String TAG = "DbAdapter";
    //设定数据文件名
    private static final String DATABASE_NAME = "iTracks.db";
    private static final int DATABASE_VERSION = 1;
    public class DatabaseHelper extends SQLiteOpenHelper {
        public DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            //编写 SQL
            String tracks_sql = "CREATE TABLE " + TrackDbAdapter.TABLE_NAME + " ("
                + TrackDbAdapter.ID + " INTEGER primary key autoincrement, "
                + TrackDbAdapter.NAME + " text not null, "
                + TrackDbAdapter.DISC + " text, "
                + TrackDbAdapter.DIST + " LONG, "
                + TrackDbAdapter.TRACKEDTIME + " LONG, "
                + TrackDbAdapter.LOCATE_COUNT + " INTEGER, "
                + TrackDbAdapter.CREATED + " text, "
                + TrackDbAdapter.AVGSPEED + " LONG, "
                + TrackDbAdapter.MAXSPEED + " LONG, "
                + TrackDbAdapter.UPDATED + " text "
                + ");";
            Log.i(TAG, tracks_sql);
            db.execSQL(tracks_sql);
            String locats_sql = "CREATE TABLE " + LocateDbAdapter.TABLE_NAME + " ("
                + LocateDbAdapter.ID + " INTEGER primary key autoincrement, "
                + LocateDbAdapter.TRACKID + " INTEGER not null, "
                + LocateDbAdapter.LON + " DOUBLE, "
                + LocateDbAdapter.LAT + " DOUBLE, "
                + LocateDbAdapter.ALT + " DOUBLE, "
                + LocateDbAdapter.CREATED + " text "
                + ");";
            Log.i(TAG, locats_sql);
            db.execSQL(locats_sql);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS " + LocateDbAdapter.TABLE_NAME + ";");
            db.execSQL("DROP TABLE IF EXISTS " + TrackDbAdapter.TABLE_NAME + ";");
            onCreate(db);
        }
    }
}
```

代码解释:

在上述代码中, 重新定义 SQLiteOpenHelper 的 onCreate 和 onUpgrade 方法, 在这两种方法中,

编写自己创建和升级数据库的脚本。

2. 数据库表的操作

下面需要实现对两个表操作的封装，因为其共用一个数据库，所以，从前面创建的 DbAdapter 中再继承出两个类 TrackDbAdapter 和 LocateDbAdapter 将要在这两个类中封装需要用到的操作，并通过接口提供出去。

首先看一下 TrackDbAdapter.java，其实现代码如下所示：

```
package com.iceskysl.iTracks;

import java.util.Calendar;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

public class TrackDbAdapter extends DbAdapter {
    //初始化一些变量
    private static final String TAG = "TrackDbAdapter";

    public static final String TABLE_NAME = "tracks";

    public static final String ID = "_id";
    public static final String KEY_ROWID = "_id";
    public static final String NAME = "name";
    public static final String DESC = "desc";
    public static final String DIST = "distance";
    public static final String TRACKEDTIME = "tracked_time";
    public static final String LOCATE_COUNT = "locats_count";
    public static final String CREATED = "created_at";
    public static final String UPDATED = "updated_at";
    public static final String AVGSPEED = "avg_speed";
    public static final String MAXSPEED = "max_speed";
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;
    private final Context mContext;

    public TrackDbAdapter(Context ctx) {
        this.mContext = ctx;
    }

    public TrackDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mContext);
        mDb = mDbHelper.getWritableDatabase();
        return this;
    }

    public void close() {
        mDbHelper.close();
    }

    //获取一个记录
    public Cursor getTrack(long rowId) throws SQLException {
        Cursor mCursor =
            mDb.query(true, TABLE_NAME, new String[] { KEY_ROWID, NAME,
                DESC, CREATED }, KEY_ROWID + "=" + rowId, null, null,
                null, null, null);
        if (mCursor != null) {
            mCursor.moveToFirst();
        }
    }
}
```

```

    }
    return mCursor;
}
//创建新的跟踪记录
public long createTrack(String name, String desc) {
    Log.d(TAG, "createTrack.");
    ContentValues initialValues = new ContentValues();
    initialValues.put(NAME, name);
    initialValues.put(DESC, desc);
    Calendar calendar = Calendar.getInstance();
    String created = calendar.get(Calendar.YEAR) + "-" + calendar.get(Calendar.MONTH) + "-"
        + calendar.get(Calendar.DAY_OF_MONTH) + " "
        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
        + calendar.get(Calendar.MINUTE) + ":" + calendar.get(Calendar.SECOND);
    initialValues.put(CREATED, created);
    initialValues.put(UPDATED, created);
    return mDb.insert(TABLE_NAME, null, initialValues);
}
// 删除
public boolean deleteTrack(long rowId) {
    return mDb.delete(TABLE_NAME, KEY_ROWID + "=" + rowId, null) > 0;
}
//获取全部跟踪记录
public Cursor getAllTracks () {
    return mDb.query(TABLE_NAME, new String[] { ID, NAME,
        DESC, CREATED }, null, null, null, null, "updated_at desc");
}
//更新已经存在的记录
public boolean updateTrack(long rowId, String name, String desc) {
    ContentValues args = new ContentValues();
    args.put(NAME, name);
    args.put(DESC, desc);
    Calendar calendar = Calendar.getInstance();
    String updated = calendar.get(Calendar.YEAR) + "-" + calendar.get(Calendar.MONTH) + "-"
        + calendar.get(Calendar.DAY_OF_MONTH) + " "
        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
        + calendar.get(Calendar.MINUTE) + ":" + calendar.get(Calendar.SECOND);
    args.put(UPDATED, updated);

    return mDb.update(TABLE_NAME, args, KEY_ROWID + "=" + rowId, null) > 0;
}
}

```

代码解释:

上述代码一开始申明一些常量，然后按照操作需要定义了几种操作，如 `createTrack` 用来创建新的跟踪记录，`updateTrack` 用来更新已有记录等。

接着看一下 `LocateDbAdapter.java` 的代码，具体如下所示：

```

package com.iceskysl.iTracks;

import java.util.Calendar;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

public class LocateDbAdapter extends DbAdapter {

```

```

private static final String TAG = "LocateDbAdapter";
public static final String TABLE_NAME = "locates";

public static final String ID = "_id";
public static final String TRACKID = "track_id";
public static final String LON = "longitude";
public static final String LAT = "latitude";
public static final String ALT = "altitude";
public static final String CREATED = "created_at";
private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;
private final Context mContext;

public LocateDbAdapter(Context ctx) {
    this.mContext = ctx;
}

public LocateDbAdapter open() throws SQLException {
    mDbHelper = new DatabaseHelper(mContext);
    mDb = mDbHelper.getWritableDatabase();
    return this;
}

public void close() {
    mDbHelper.close();
}

//获取位置
public Cursor getLocate(long rowId) throws SQLException {
    Cursor mCursor =
        mDb.query(true, TABLE_NAME, new String[] { ID, LON,
            LAT, ALT, CREATED }, ID + "=" + rowId, null, null,
            null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}

//创建一个位置记录
public long createLocate(int track_id, Double longitude, Double latitude, Double altitude)
{
    Log.d(TAG, "createLocate.");
    ContentValues initialValues = new ContentValues();
    initialValues.put(TRACKID, track_id);
    initialValues.put(LON, longitude);
    initialValues.put(LAT, latitude);
    initialValues.put(ALT, altitude);

    Calendar calendar = Calendar.getInstance();
    String created = calendar.get(Calendar.YEAR) + "-" + calendar.get(Calendar.MONTH) + "-"
        + calendar.get(Calendar.DAY_OF_MONTH) + " "
        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
        + calendar.get(Calendar.MINUTE) + ":" + calendar.get(Calendar.SECOND);
    initialValues.put(CREATED, created);
    return mDb.insert(TABLE_NAME, null, initialValues);
}

// 删除一个位置记录
public boolean deleteLocate(long rowId) {
    return mDb.delete(TABLE_NAME, ID + "=" + rowId, null) > 0;
}

//获取一个跟踪的全部记录点
public Cursor getTrackAllLocates(int trackId) {
    return mDb.query(TABLE_NAME, new String[] { ID, TRACKID, LON,
        LAT, ALT, CREATED }, "track_id=?", new String[] {String.valueOf(trackId)}, null, null,
        "created_at asc");
}

```

 代码解释：

和前面那个类的代码一样，首先定义一些常量，然后按照操作的需要定义了几种操作，在此不再赘述。

13.3.4 实现 Service

此程序还有最后一个部分功能需要完成，实现 Service。因为想要的 GPS 效果是跟踪开始后，当切换界面等操作是都不影响 GPS 跟踪，也就是希望可以在后台不间断的跟踪、记录。在这个场景下，自然需要使用 Service。

在前面章节中已经比较系统地讲过 Service 的使用，读者在那基础上，将会容易理解下面的操作。下面创建一个名字为 Track 的 Service，然后在 manifest 文件中的 application 标签内加入对其的申明，实现代码如下所示：

```
<Service android:name=".Track">
<intent-filter>
<action android:name="com.iceskysl.iTracks.START_TRACK_SERVICE" />
<category android:name="android.intent.category.default" />
</intent-filter>
</Service>
```

 代码解释：

上述代码表示添加一个名字为 Track 的 Service，并指定其名字为 “com.iceskysl.iTracks.START_TRACK_SERVICE”。然后看 Track 的代码如下所示：

```
package com.iceskysl.iTracks;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class Track extends Service {
    private static final String TAG = "Track";

    private LocationManager lm;
    private LocationListener locationListener;
    static LocateDbAdapter mlcDbHelper = null;
    private int track_id;

    @Override
    public IBinder onBind(Intent arg0) {
        Log.d(TAG, "onBind.");
        return null;
    }

    public void onStart(Intent intent, int startId) {
        Log.d(TAG, "onStart.");
        super.onStart(intent, startId);
        mlcDbHelper = new LocateDbAdapter(this);
        mlcDbHelper.open();
    }
}
```



```

Bundle extras = intent.getExtras();
if (extras != null) {
    track_id = extras.getInt(LocateDbAdapter.TRACKID);
}
Log.d(TAG, "track_id =" + track_id);
lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
}

public static LocateDbAdapter getDbHelp(){
return mlcDbHelper;
}

    public void onDestroy() {
Log.d(TAG, "onDestroy.");
    super.onDestroy();
    mlcDbHelper.close();
}
//

    protected class MyLocationListener implements LocationListener {

@Override
public void onLocationChanged(Location loc) {
Log.d(TAG, "MyLocationListener::onLocationChanged..");
if (loc != null) {
// ///////////
mlcDbHelper.createLocate(track_id, loc.getLongitude(), loc.getLatitude(),
loc.getAltitude());
}
}

@Override
public void onProviderDisabled(String provider) {
Toast.makeText(
getBaseContext(),
"ProviderDisabled.",
Toast.LENGTH_SHORT).show(); }

@Override
public void onProviderEnabled(String provider) {
Toast.makeText(
getBaseContext(),
"ProviderEnabled,provider:"+provider,
Toast.LENGTH_SHORT).show(); }

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
// TODO Auto-generated method stub
}
}
}

```

代码解释:

让 Track 继承 Android 的 Service 类, 然后在其 onStart 中连接数据库, 接收参数并设定监听器, 并使用了一个前面已经讲过的 MyLocationListener, 使得在 onLocationChanged (位置变化) 的时候, 调用前面数据存储部分已经实现的 mlcDbHelper.createLocate(track_id, loc.getLongitude(), loc.get

Latitude(), loc.getAltitude()), 将位置信息和接收到的参数写入数据库中。

进而, 就可以写出与之相应的启动和停止 Service 的方法, 实现代码如下所示:

```
private void startTrackService() { //启动服务
    Intent i = new Intent("com.iceskysl.iTracks.START_TRACK_SERVICE");
    i.putExtra(LocateDbAdapter.TRACKID, track_id);
    startService(i);
}

private void stopTrackService() { //停止服务
    stopService(new Intent("com.iceskysl.iTracks.START_TRACK_SERVICE"));
}
```

如此一来, 只有我们启动了 Service, 其就会在后台一直运行, 当位置信息发生变化时, 调用数据存储接口存入数据库。

至此, 这个应用需要的功能都已经实现了, 下面需要把这些功能串起来形成一个完整应用, 具体实现请继续看下一节。

13.3.5 应用流程整体实现

应用的功能部分实现了, 需要把各个功能按照我们的设计连接起来。在这个应用中, 需要做的主要有以下几个地方:

- (1) 继续跟踪;
- (2) 个性化设置的读取和使用;
- (3) 去掉不需要的调试信息;
- (4) 进一步美化和优化应用。

继续跟踪, 就是选择一个已有的跟踪条目, 并继续跟踪后续的位置变化, 主要就是取得需要继续跟踪的记录 ID, 在启动 Service 的时候传递进去就可以了, 示例代码如下所示:

```
private void conTrackService() {
    Intent i = new Intent("com.iceskysl.iTracks.START_TRACK_SERVICE");
    Long track_id = getListView().getSelectedItemId();
    i.putExtra(LocateDbAdapter.TRACKID, track_id.intValue());
    startService(i);
}
```

 代码解释:

上述代码中实现当选择“继续跟踪”的时候, 启动 Track 这个 service。

个性化设置现在还比较少, 只有地图的展示等级, 需要在地图展示的时候读取预先的配置, 显示的时候使用 setZoom 设定即可。示例代码如下所示:

```
mMapView = (MapView) findViewById(R.id.mv);
mc = mMapView.getController();
SharedPreferences settings = getSharedPreferences(Setting.SETTING_INFOS, 0);
int setting_gps = settings.getInt(Setting.SETTING_MAP, 10);
mc.setZoom(setting_gps);
```

 代码解释:

首先获取模板文件中的 MapView, 进而调用其 getController 获取其控制器, 就可以直接调用 setZoom() 方法将用户设置的默认地图缩放等级设定进去了。

除此之外, 还需要去掉一些不需要的调试信息, 优化代码结构和执行效率等, 这里就不一一赘

述了，本应用的全部代码可以从附带的光盘及支持网站上下载。

13.3.6 后续开发完善

到目前为止，我们很完整的完成这个 GPS 位置跟踪应用，如果读者想继续完善开发，笔者觉得应该从如下几个思路进行。

1. 实时导航

读者可能也注意到了，在创建数据库的时候，有 Avg_speed 这样的字段，通过它可以在此基础上进一步完善，取得移动的方向、最大速度、跟踪时间等。

2. 文件导出

将每次跟踪路径上的点以文件的方式导出，这是一个很好的完善。

3. 互联网方向

在个人手机上毕竟还是自己一个人用，如果放在互联网上，就可以和更多的人分享你的旅程。而在其实现上，可以将每次跟踪导出为 KML 文件，也可以每次更新位置时给指定的服务器发送数据，服务器实时记录你的位置。进而围绕着轨迹产生一系列的想法，例如，即时消息、照片展示等有趣的功能。

13.4 项目程序演示

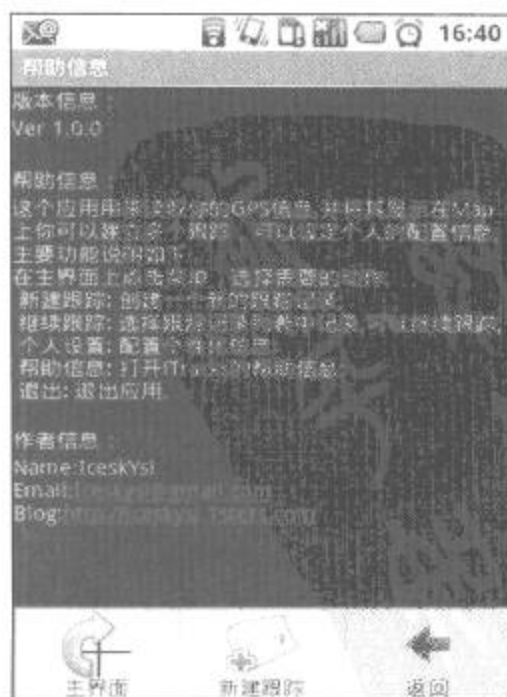
完成应用编程后，可以演示一遍，过程如下。

第一次运行程序时，看到跟踪列表是空的，并出现欢迎使用的文字，如图 13-16 所示。

然后选择“帮助信息”菜单，打开应用的帮助界面，如图 13-17 所示。



▲ 图 13-16 主界面



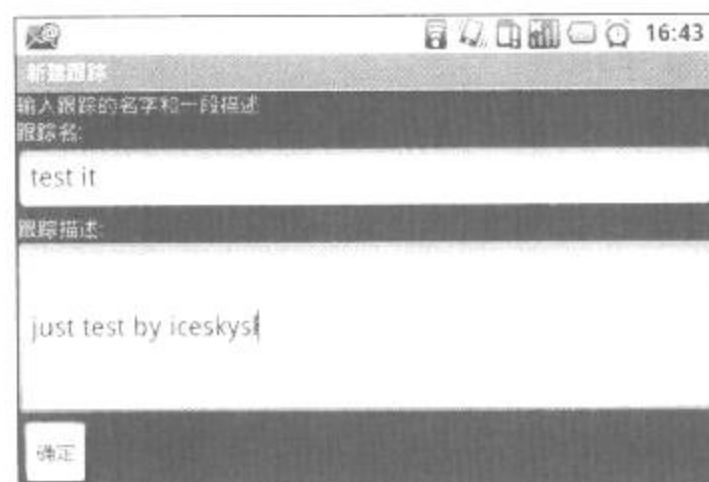
▲ 图 13-17 帮助信息界面

了解了帮助信息以后，选择菜单中的“返回”项，返回到主界面。接着选择“个人设置”菜单，打开个性化设置界面，如图 13-18 所示。

在个人设置界面中，选择需要的设置，单击“确定”按钮，保存成功后自动返回到主界面。接着在主界面选择“新建跟踪”菜单，在打开的新建跟踪界面上输入跟踪名和一段描述文字，如图 13-19 所示。



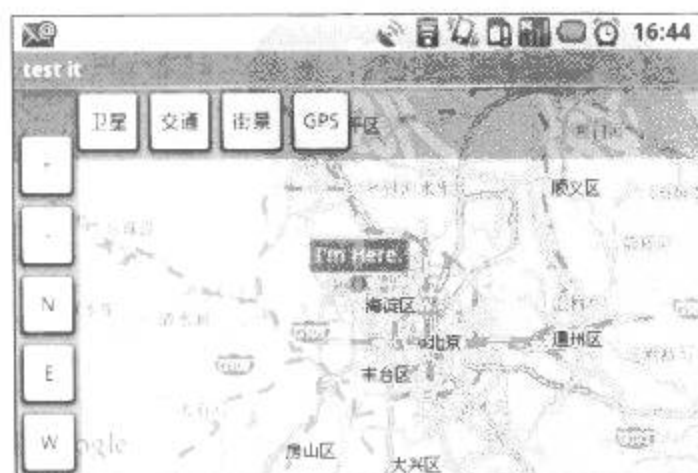
▲ 图 13-18 个人设置界面



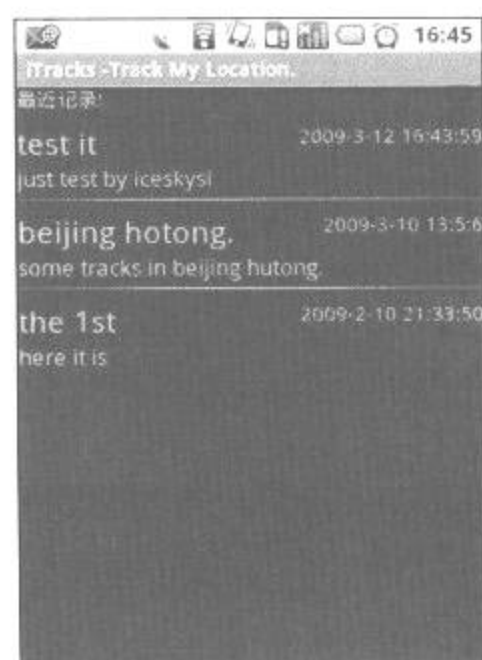
▲ 图 13-19 新建跟踪界面

单击“确定”按钮后，保存新的跟踪信息，并自动跳转到地图展示页面，并定位到 GPS 所标识的地点，如图 13-20 所示。

通过此步骤，可以创建多个跟踪，其主界面如图 13-21 所示。



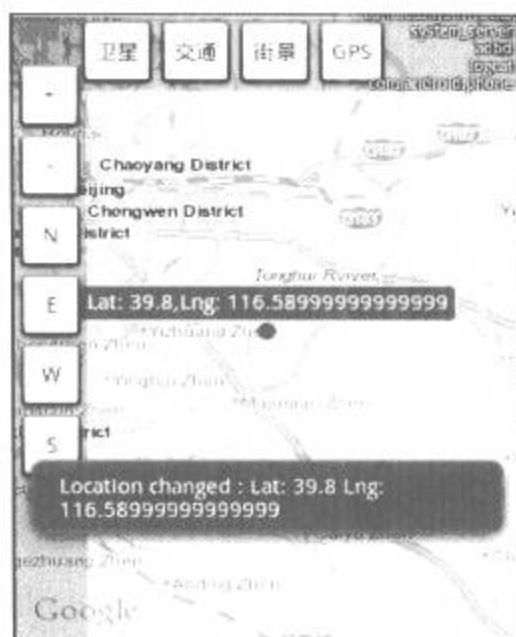
▲ 图 13-20 个人位置展示界面



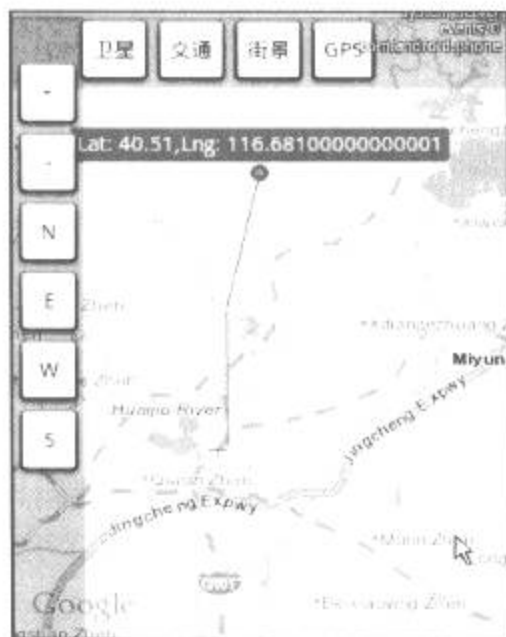
▲ 图 13-21 主界面列表显示

启动跟踪后，当 GPS 信息变化的时候，地图会随之移动，并提示当前的位置，如图 13-22 所示。

跟踪一段时间后,可以看到在地图上显示了其行动轨迹,如图 13-23 所示。



▲ 图 13-22 GPS 信息改变界面



▲ 图 13-23 轨迹图

同时可以使用旁边的缩放、移动以及地图模式切换功能,如图 13-24 和图 13-25 所示。



▲ 图 13-24 放大功能界面



▲ 图 13-25 卫星模式地图界面

至此,主要功能都演示了,达到了设计的功能,接下去需要看看如何打包和发布应用了。

13.5 项目程序签名、打包和发布

应用程序写完了,接下来想要做的就是发布出去以提供给其他人使用。但这些流程也需要规范,下面重新温习下打包、签名和发布过程的知识。

在应用打包、发布前,还需要按照前面第 4 章中的说明,对发布包进行签名,并需要对签名文

件所对应的 MD5 指纹申请一个 GoogleMap API Key，方法和前面申请 Debug 模式下的 API Key 相似。其语法是：keytool -list -alias release.keystore -keystore keys/release.keystore[release.keystore 是你的签名文件]，例如，笔者自己的就是：

```
F:\svn\book\android\keytools>"C:\Program Files\Java\jdk1.6.0_10\bin\keytool.exe"
```

```
-list -alias iceskysl.keystore -keystore keys/iceskysl.keystore
```

输入 keystore 密码：iceskysl.keystore, 2009-1-14, PrivateKeyEntry。

认证指纹 (MD5)：17:79:23:83:C1:C8:86:FE:F7:9E:08:11:E4:F6:44:F0。

使用 17:79:23:83:C1:C8:86:FE:F7:9E:08:11:E4:F6:44:F0 到 <http://code.google.com/intl/zh-CN/android/maps-api-signup.html> 申请一个 Key，使用这个 key 替换开始写在 show_track.xml 文件中的 android:apiKey 即可。

替换完成后，保存后会自动重新编译生成 APK 文件，其在项目目录下的 bin 目录下，这里就是 iTracks\bin\iTracks.apk，但是这并不是需要的打包后的文件，需要导出无签名的 APK，具体方法在第 4 章已经讲过。在 Eclipse 中选择项目，单击鼠标右键，选择“Android Tool”下的“Export Unsigned Application Package”，保存文件和签名后就可以发布了，发布命令如下所示：

```
F:\svn\book\android\keytools>"C:\Program Files\Java\jdk1.6.0_10\bin\jarsigner.exe"
```

```
-verbose -keystore keys/iceskysl.keystore -signedjar iTracks_signed.apk iTra
```

```
cks.apk iceskysl.keystore
```

输入密钥库的口令短语：

正在添加：META-INF/MANIFEST.MF

正在添加：META-INF/ICESKYSL.SF

正在添加：META-INF/ICESKYSL.RSA

正在签名：res/drawable/back.png

正在签名：res/drawable/con_track.png

正在签名：res/drawable/delete.png

正在签名：res/drawable/exit.png

正在签名：res/drawable/helps.png

正在签名：res/drawable/icon.png

正在签名：res/drawable/new_track.png

正在签名：res/drawable/setting.png

正在签名：res/layout/helps.xml

正在签名：res/layout/main.xml

正在签名：res/layout/new_track.xml

正在签名：res/layout/setting.xml

正在签名：res/layout/show_track.xml

正在签名：res/layout/track_row.xml

正在签名：AndroidManifest.xml

正在签名：resources.arsc

正在签名: classes.dex

在其目录下生成一个 iTracks_signed.apk, 它就是签名后的文件, 然后将其发给别人或者上传到 Android Market 就可以供其他人安装使用了, 如图 13-26 所示。



▲ 图 13-26 上传到 Android Market 界面

13.6 本章小结

本章比较系统地讲解了一个 GPS 位置跟踪应用的开发过程, 其中涉及很多前面讲解的知识, 是一次实际综合应用开发的范例。在开发这个应用中, 涉及 Android 自身和 GoogleMap 的众多实战技巧和注意事项, 通过本章的学习, 相信可以帮助读者掌握 Android 软件的开发流程和基本步骤, 可以写出自己满意的应用了。



第 14 章 Android 综合案例三——基于 Android 的豆瓣网（Web 2.0）移动客户端开发

学习目标

- 了解 Web 2.0 API; ☐
- 掌握开发 Web 2.0 手机客户端应用; ☐
- 掌握 Android 系统和标准的 Web2.0 网站交互; ☐
- 掌握 Web 2.0 的 OAuth 认证; ☐
- 掌握 Android 应用设计流程。 ☐

14.1 关于豆瓣网和豆瓣网 API

豆瓣网是国内 Web 2.0 中的代表网站之一，也是开放 API 比较早的优秀网站之一，之所以选择豆瓣网作为例子是因为豆瓣网不但开放了 API，而且在国内享有很高的知名度，用户群很多，当然了，作者也是豆瓣网的忠实用户之一。通过豆瓣网这个例子，读者可以以点带面地学会，如何写其他 Web 2.0 网站的客户端应用，如 twitter、facebook 或者 myspace。在这一章当中由于篇幅所限，我们只实现豆瓣网客户端的用户登录和显示“我说”部分，关于豆瓣网客户端完整的应用程序，请参照我们的光盘，或者从我们已经放在 Google 的 Market 里边找，大家也可以搜索 eoeDouban 进行免费的安装和使用。

14.1.1 豆瓣网介绍

豆瓣网是国内典型的 Web 2.0 网站，也是非常具有特色的网站之一。以往其他的 Web 2.0 网站都是模仿国外的网站，而豆瓣网可以说是一种原创，豆瓣网模式已经成为一种模式。

通过豆瓣网，可以看到使用豆瓣网的第一步：用豆瓣网中各页面里的搜索栏查找喜欢的书、电影或音乐。如一本书，找到以后单击封面或者书名，就能看介绍、评论、讨论，还有正在读这本书的其他的人。单击“在读”或者“读过”按钮，就收藏了这本书。别人也可以通过这本书找到你了；单击“我来评论”，就可以发表评论。

通过网站上面的“我读”、“我看”、“我听”，可以浏览自己添加的收藏。也可以在收藏时添加标签，用自己的方式组织收藏。有一些收藏以后，在“我读”、“我看”、“我听”里能看到豆瓣网根据网友的口味自动给出的推荐。网友的收藏越多，豆瓣网推荐会越准确。在豆瓣网上任何地方看到别的成员，都可以单击名字或头像，去看她/他的简单介绍、收藏、推荐和发表过的评论。如果觉

得这个人有意思,或者趣味相投,可以单击“把他/她加入我的友邻”。这样网友就有了一个小圈子,单击最上面“我的友邻”,可以随时进去。

14.1.2 豆瓣网 API 介绍

豆瓣网提供了丰富的 API 供第三者开发使用,以便可以开发出功能更加丰富的各种客户端。根据豆瓣网官方网站的介绍,目前豆瓣网 API 支持的功能包括:

- 搜索并查看书籍、电影、音乐信息;
- 搜索并查看用户信息,查看用户友邻信息;
- 查看用户收藏;
- 添加、更新、删除用户收藏;
- 查看评论;
- 发布、修改、删除评论;
- 查看、添加、删除用户广播;
- 查看、添加、删除用户日记;
- 搜索并查看、添加、删除活动;
- 查看、添加、删除、回复推荐。

关于豆瓣网更详细的 API 信息,大家可以参考豆瓣网官方提供的 API 参考手册,网址是:
<http://www.douban.com/service/apidoc/reference/>。

14.1.3 豆瓣网 API 认证

为了保护豆瓣网用户的数据,当第三方应用需要通过 API 访问或修改受保护的用户数据(如增、删用户收藏)时,需要通过 OAuth 认证机制来获得用户的授权。也许很多人不是很了解 OAuth 认证,那么,先简单介绍一下 OAuth 协议相关的内容,由于这部分不是本书的重点,不会花大量的篇幅进行说明,根据具体的信息,大家可以通过互联网获得。

1. 什么是 OAuth 认证

OAuth 协议致力于使网站和应用程序(统称为消费方)能够在无须用户透露其认证证书的情况下,通过 API 访问某个 Web 服务(统称为服务提供方)的受保护资源。通俗地说,OAuth 为 API 认证提供了一个可自由实现且通用的方法。一个典型的场景是某打印机提供商,希望在无须用户提供其照片存储网站密码的情况下,访问用户存储在网站上的私有照片资源。就我们这个豆瓣网版客户端来讲,Android 的豆瓣网客户端就是消费方,通过 OAuth 协议,Android 客户端不用获得用户密码的情况下就可以发表“我说”,或者访问其他有用户权限的资源。

OAuth 的诞生可以说是“豪门云集”。一出来的目标就是统一各个网络服务商的各自登录/授权/认证 API,例如 Digg、Jaiku、Flickr、Magnolia、Plaxo、Pownce、Twitter、Google 和 Yahoo 等。

2. 什么是 API Key

在介绍具体豆瓣网的 OAuth 认证之前,有必要详细地了解一下豆瓣网的 API Key。

为了保护豆瓣网用户的数据，防止 API 被滥用或恶意使用，豆瓣网要求每个 API 的使用者申请一个 API Key，而每个 API Key 惟一标识一个 API 使用者。可以在申请页面获得 API Key，在页面中填写必要信息后提交，会得到一个 API Key，例如 c4579586f41a90372f762cb65c78be5d。

在之后的 API 使用过程中，你需要在请求中包括 apikey 参数，如 `http://api.douban.com/people/1345767?apikey={apikey}`。

需要注意的是，如果只是想试验一下 API，豆瓣网也允许在不申请 API Key 的情况下进行 API 调用。不过在这种情况下，API 调用被限制为每分钟请求不超过 10 次。使用 API Key 时，对访问的限制较为宽松，为每分钟 40 次，超过限制的话会被封禁。

申请 APIKey 是非常方便的，如图 14-1 所示，大家只需要登录到网址为 `http://www.douban.com/service/apikey/apply`（具体参照豆瓣网）上，填写一些必要的信息，提交豆瓣网后，你会得到一个 API Key 和一个私钥。这两个参数在写我们这个客户端的时候是需要的。

申请豆瓣API key

你的 API Key

应用名称:

此 Douban API Key 用于:

☐ 非商业用途 ☐ 商业用途

☐ 网站应用 ☐ 桌面应用

说明你的应用

向豆瓣网管理员描述你的应用，应该包括你的产品或机构名称，该信息不会被公开。
除了该网页外，请同时提供你的 API Key 的用途。我们承诺会保护此内容不被他人看到。

应用 URL (可选)

如果你是在互联网应用里使用豆瓣 API，请提供网站的 URL

▲ 图 14-1 申请 API Key 界面

3. 认证及访问资源流程

豆瓣网 API 通过以下 4 个步骤来完成认证授权，并访问或修改受限资源的流程。

- (1) 过我们上边所申请的 APIkey 和它的私钥来获取未授权的请求令牌 (Request Token)。
- (2) 取请求令牌后请求用户授权这个请求令牌。
- (3) 如果用户授权了这个请求令牌，那么就用授权后的请求令牌换取访问令牌 (Access Token)。
- (4) 使用访问令牌访问或修改受保护资源。

4. 获取未授权的请求令牌 (Request Token)

通过访问以下网址获取未授权的 Request Token:

```
http://www.douban.com/service/auth/request_token。
```

包含的主要参数如表 14-1 所示。

表 14-1 网站显示包括的参数

参 数	意 义
oauth_consumer_key	API Key
oauth_signature_method	签名方法
oauth_signature	签名值
oauth_timestamp	时间戳
oauth_nonce	单次值，随机字符串，防止重放攻击

其中参数 oauth_consumer_key 为我们申请的 API key，oauth_signature 为和 API key 配套的私钥。

返回值包括未授权的请求令牌（Request Token）和对应的请求令牌密钥（Request Token Secret）。

5. 请求用户授权 Request Token

获得请求令牌（Request Token）之后，需要请求用户授权该请求令牌 Request Token。

你需要将浏览器跳转到如下网址（如果无法自动跳转，则需要提示用户手工跳转）。这会是一个豆瓣网上的页面，提示用户授权给的应用，以允许应用访问该用户在豆瓣网上的信息，网址为：

<http://www.douban.com/service/auth/authorize>

跳转后，用户会看到请求授权的页面如图 14-2 所示，用户可以选择同意或者拒绝授权。该请求包含如表 14-2 所示的两个参数。

API 认证授权

第三方应用 Android 豆瓣客户端 希望操作你在豆瓣上的数据：

Android 豆瓣客户端 可能进行的操作包括：

发布、修改、删除你的评论
添加、更新、删除你的收藏
查看、添加、删除你的广播
查看、添加、删除你的日记

授权后，你可以随时在我的设置里随时为 Android 豆瓣客户端 授权

注：授权操作不会导致第三方应用 Android 豆瓣客户端 获知你的密码

你是否同意向第三方应用 Android 豆瓣客户端 授权？

同意 不同意

▲ 图 14-2 请求授权的页面界面

表 14-2 包含的请求参数

参 数	意 义
oauth_token	上一步中获得的请求令牌（Request Token）
oauth_callback （可选）	如果包含这个参数，认证成功后浏览器会被重定向到形如 http://callback?oauth_token=ab3cd9j4ks73hf7g 的 url，其中 oauth_token 为 Request Token 否则需要用户手工通知第三方应用以完成授权

6. 使用授权后的 Request Token 换取 Access Token

用户完成授权后，第三方应用可以通过访问如下网址，将已授权的 Request Token 换取 Access Token。Access Token 将被用于访问或修改受限资源。网址为：

http://www.douban.com/service/auth/access_token

包含的主要参数如表 14-3 所示。

表 14-3 包含的主要参数

参 数	意 义
oauth_consumer_key	API Key
oauth_token	第一步中获得的请求令牌（Request Token）
oauth_signature_method	签名方法
oauth_signature	签名值
oauth_timestamp	时间戳
oauth_nonce	单次值

其中 oauth_consumer_key 为 API Key，oauth_token 为第一步中获得的请求令牌（Request Token），

oauth_signature 是签名值。

7. 使用 Access Token 访问或修改受保护资源

获得 Access Token 之后，用户的应用就可以使用该 Access Token 访问或修改受保护的资源。在每次操作受保护资源时，请求都必须包含如表 14-4 所示的参数。

参 数	意 义
oauth_consumer_key	API Key
oauth_token	Access Token
oauth_signature_method	签名方法
oauth_signature	签名值
oauth_timestamp	时间戳
oauth_nonce	单次值

本步骤用于签名的 secret 和 token secret 分别为，API Key Secret 和 Access Token Secret。

14.1.4 豆瓣网 API 快速入门

在这里我们只对 API 进行一个入门性的介绍，豆瓣网提供了比较完整的 API，具体详情请大家参照豆瓣网提供的 API 参考手册。下面通过一个简单的示例来显示豆瓣网 API 的使用。这个例子当中展示了使用 API 获得 ID 为 haiyangjy 用户的信息，请求的 URL(网址)如下(注意，将{yourapikey}替换为用户的 API Ke)：

```
http://api.douban.com/people/haiyangjy?apikey={yourkeyapi}。
```

返回的 XML 文档代码为：

```
<entry>
  <id>http://api.douban.com/people/2478532</id>
  <title>海阳</title>
  <link href="http://api.douban.com/people/2478532" rel="self"/>
  <link href="http://www.douban.com/people/haiyangjy/" rel="alternate"/>
  <link href="http://otho.douban.com/icon/u2478532-9.jpg" rel="icon"/>
  <content>
    Great just isn't good enough.</content>
  <db:location id="beijing">北京</db:location>
  <db:uid>haiyangjy</db:uid>
</entry>
```

通过解析这个 XML 文档，我们就可以得到所需要的用户信息，其他 API 的使用和上边这个类似。也是通过 http 请求，并返回一个标准的 XML 文档来实现和服务器的交互。

让我们简单地看一下这个 XML 文档。这段 XML 最外边的节点是<entry>节点，在豆瓣网返回的 XML 文档中，经常看到<entry>节点，<entry>代表单个信息实体，一本书或一部电影的信息也会包含在一个<entry>节点中，在这个节点当中有具体的信息，如在上边这个返回的 XML 文档中包含了：

- 用户的 id 为“haiyangjy”；
- 用户的昵称为“海阳”；

- 用户的头像的 URL 为 <http://otho.douban.com/icon/u2478532-9.jpg>;
- 用户的描述为 “Great just isn't good enough.”;
- 用户的所在地为 “北京”。

14.1.5 豆瓣网 API Java 库介绍

虽然豆瓣网已经提供了强大的 API, 但是直接使用起来还是比较复杂, 毕竟让用户把这么多的、各种各样的 XML 文档进行解析并提取有用信息是一件麻烦而且容易出错的事情。但是让人高兴的是, 网上有人开发出了豆瓣网 API 的 Java 库, 这样我们省去了大量宝贵的时间, 直接来调用和我们非常熟悉的 Java 类就可以实现 API 的访问。下边我们来介绍一个如何使用这个 Java 的 API 库, 此 Java 库为开源项目, 大家可以在网址 <http://code.google.com/p/douban-java/> 上进行下载源码, 或者编译后的 class 文件下载。

1. 授权认证

豆瓣网 API Java 客户端可以用来处理公共信息和受保护的信息。公共信息不需要授权, 它是只读的。受保护的信息是需要用户认证并授权第三方应用拥有相关权限, 授权后的客户端可以得到该用户对应的受保护的信息, 并可以新建, 修改或者删除相应的受保护的信息 (对于某些信息, 授权后的用户和未授权的用户得到的数据是不一样的, 如评论信息, 只有授权用户才能得到该用户的完整评论信息, 其他用户只能得到评论信息摘要)。

为了可以访问或者修改受保护的信息, 需要使用如下语句来实例化一个 DoubanService 对象, 具体代码如下:

```
String apiKey = "659ef56f6b705e1230dce04e42511***";
String secret = "006ba4a4899*****";
DoubanService myService = new DoubanService("ApplicationName", apiKey,
    secret);
```

 代码解释:

- DoubanService 是豆瓣 Java 库的核心类, 这个类里边提供了大量的 Java API 接口供我们使用。调用 myService 的 getAuthorizationUrl (String callback) 方法可以得到授权的 URL 链接。在通过用户的授权后, 需要调用 myService 的 getAccessToken() 来得到 Access Token。得到 Access Token 后的访问可以访问或修改受保护的资源。更多关于认证的信息, 请参考豆瓣网 API 认证授权说明。

2. 理解 feed 和 entry (条目)

feed 是由同种 entry 组成的集合, entry 对应单个信息实体, 比如 SubjectEntry, 对应一部电影, 一部音乐或者一本书, UserEntry 对应一个用户, ReviewEntry 对应一篇评论, 这些 entry 包含了它对应物的各种信息。而 SubjectFeed、UserFeed 和 ReviewFeed 则是相应的 entry 的集合。得到一个 entry 集合 (feed) 并处理相关 entry 的代码如下:

```
UserFeed userFeed = myService.findUser("haiyangjy", 10, 5);
for (UserEntry ue : userFeed.getEntries()) {
    printUserEntry(ue);
}
```

 代码解释:

- 上述代码表示以关键词 “haiyangjy” 来搜索用户, 从第 10 项开始返回 5 项。

3. 获得一个特定的 entry (条目)

如果知道条目的 URL (通常可以通过对 entry 对象调用 getId() 方法来得到该条目的 URL), 比如 <http://api.douban.com/movie/subject/3036997>, 这是电影《立春》的 API URL, 可以通过如下方式获得该 entry:

```
String entryUrl = "http://api.douban.com/movie/subject/3036997";
SubjectEntry movieEntry = myService.getBook(entryUrl);
System.out.println("Title: " + movieEntry.getTitle().getPlainText());
```

4. 得到 entry (条目) 详细信息

得到条目后, 可以调用它的各个方法来得到条目的各种信息, 例如, 对于电影、音乐和书的 entry, 可以得到摘要、作者、标题、Id 和标签等各种信息。代码示例如下:

```
if (subjectEntry.getSummary() != null)
    System.out.println("summary is "
        + subjectEntry.getSummary().getPlainText());
System.out.println("author is "
    + subjectEntry.getAuthors().get(0).getName());
System.out.println("title is " + subjectEntry.getTitle().getPlainText());
for (Attribute attr : subjectEntry.getAttributes()) {
    System.out.println(attr.getName() + " : " + attr.getContent());
}
System.out.println("id is " + subjectEntry.getId());
for (Tag tag : subjectEntry.getTags()) {
    System.out.println(tag.getName() + " : " + tag.getCount());
}
Rating rating = subjectEntry.getRating();
if (rating != null)
    System.out.println("max is " + rating.getMax() + " min is "
        + rating.getMin() + " numRaters is "
        + rating.getNumRaters() + " average is "
        + rating.getAverage());
```

对于评论 entry, 可以得到摘要 (授权用户则是得到自己评论的全文信息)、作者、标题、Id、发布时间、更新时间等各种信息。代码示例如下:

```
System.out.println("content is " + reviewEntry.getPlainTextContent());
System.out.println("id is : " + reviewEntry.getId());
System.out.println("title is : "
    + reviewEntry.getTitle().getPlainText());
if (!reviewEntry.getAuthors().isEmpty()) {
    System.out.println("author name is : "
        + reviewEntry.getAuthors().get(0).getName());
    System.out.println("author URI is : "
        + reviewEntry.getAuthors().get(0).getUri());
}
System.out.println("updated date is : " + reviewEntry.getUpdated());
System.out.println("published date is : " + reviewEntry.getPublished());
if (reviewEntry.getSummary() != null)
    System.out.println("summary is : "
        + reviewEntry.getSummary().getPlainText());
Rating rating = reviewEntry.getRating();
if (rating != null)
    System.out.println("Rating : max is " + rating.getMax()
        + " min is " + rating.getMin() + " numRaters is "
        + rating.getNumRaters() + " average is "
        + rating.getAverage() + " value is " + rating.getValue());
```

对于用户 entry, 可以得到 Id、常居地、自我介绍等信息。代码示例如下:

```
System.out.println("id is " + userEntry.getId());
System.out.println("location is " + userEntry.getLocation());
System.out.println("content is "
    + ((TextContent) userEntry.getContent()).getContent().getPlainText());
System.out.println("title is " + userEntry.getTitle().getPlainText());
```

5. 搜索 entry (条目)

对于用户、电影、音乐和书的条目, 支持搜索某个关键词的 entry 信息, 并支持从特定的条目索引返回最多某个个数的 entry。下面的代码展示搜索包含关键词 douban 的用户, 从第 10 个条目起最多返回 5 个条目, 返回的结果是一个用户 feed, 实例代码如下:

```
// q=douban&start-index=10&max-results=5
UserFeed userFeed = myService.findUser("douban", 10, 5);
for (UserEntry ue : userFeed.getEntries()) {
    printUserEntry(ue);
}
```

6. 发布修改和删除评论

发布修改和删除评论必须先进行 API 认证授权。发布评论需要使用 createReview 方法, 更新评论需要调用 updateReview 方法, 删除评论需要调用 deleteReview 方法, 代码示例如下:

```
String movieId = "3036997"; //立春(And the Spring Comes)
SubjectEntry se = myService.getMovie(movieId);
Rating rate = new Rating();
rate.setValue(4);
String content = "《立春》两个小时要讲六个人";
reviewEntry = myService.createReview(se, new PlainTextConstruct(
    "立春"), new PlainTextConstruct(content), rate);
rate.setValue(5);
reviewEntry = myService.updateReview(reviewEntry, null, null, rate);
printReviewEntry(reviewEntry);
myService.deleteReview(reviewEntry);
```

7. 添加、修改和删除收藏

添加、修改和删除收藏必须先进行 API 认证授权。添加收藏需要使用 createCollection 方法, 更新收藏需要调用 updateCollection 方法, 删除收藏需要调用 deleteCollection 方法, 代码示例如下:

```
CollectionEntry ce;

Rating rating = new Rating();
rating.setValue(4);
ArrayList<Tag> tags = new ArrayList<Tag>(2);
Tag t1 = new Tag();
t1.setName("顾长卫");
Tag t2 = new Tag();
t2.setName("李樯");
tags.add(t1);
tags.add(t2);
String movieId = "3036997"; // 立春(And the Spring Comes)
SubjectEntry se = myService.getMovie(movieId);
ce = myService.createCollection(new Status("watched"), se, tags,
```



```

        rating);
rate.setValue(5);
ce = myService.updateCollection(ce, new Status("watched"), tags, rating);
myService.deleteCollection(ce);

```

14.2 豆瓣网 (Web 2.0) 客户端设计

14.2.1 客户端功能规划设计

虽然豆瓣网没有开放全部的 API, 但是目前开放的 API 已经是非常丰富了, 这些 API 概括起来有以下是一些:

- 搜索并查看书籍、电影、音乐信息;
- 搜索并查看用户信息, 查看用户友邻信息;
- 查看用户收藏;
- 添加、更新、删除用户收藏;
- 查看评论;
- 发布、修改、删除评论;
- 查看、添加、删除用户广播;
- 查看、添加、删除用户日记;
- 搜索并查看、添加、删除活动;
- 查看、添加、删除、回复推荐。

利用这些 API 可以开发一个功能很强大的豆瓣网客户端, 但是鉴于本书的重点在于 Android 的学习, 所以, 只选择里边具有代表性且有趣的 API 进行开发讲解, 其他的读者可以举一反三的应用。在这里只选择实现广播的部分功能:

- 显示所有友邻的广播;
- 在手机上将“我说”同步到豆瓣网的服务器上。

完整的豆瓣网客户端, 大家可以上 Google 的 Market 上搜索 eoeDouban 进行下载, 或从附带光盘下载。

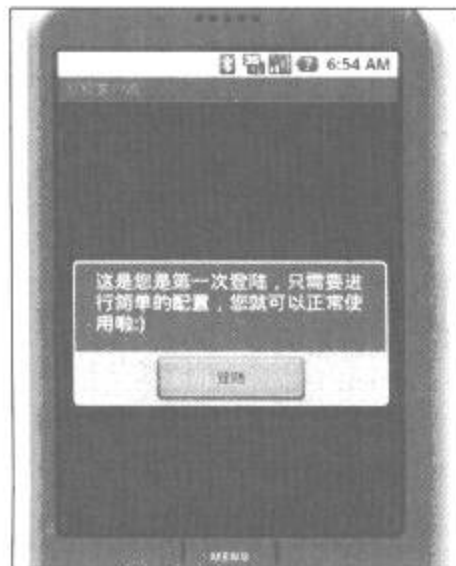
14.2.2 UI 和交互流程设计

选择好实现的功能后我们进行 UI 和交互流程设计。在实现一个应用的时候, 设计 UI 是非常重要的, 即使在草稿纸上简单的画一些流程图也是非常有益的。这个实例的主要功能如下。

(1) 用户第一次登录, 显示如图 14-3 所示的界面。当用户第一次登录的时候, 程序会弹出来一个对话框用于提示用户, 当用户单击登录按钮, 正式进入登录的过程。

(2) 跳转到 API 认证授权界面。当用浏览器打开授权认证界面并且用户没有登录时, 会显示如图 14-4 所示的界面。

(3) 用户登录界面, 如图 14-5 所示。



▲ 图 14-3 第一次登录界面



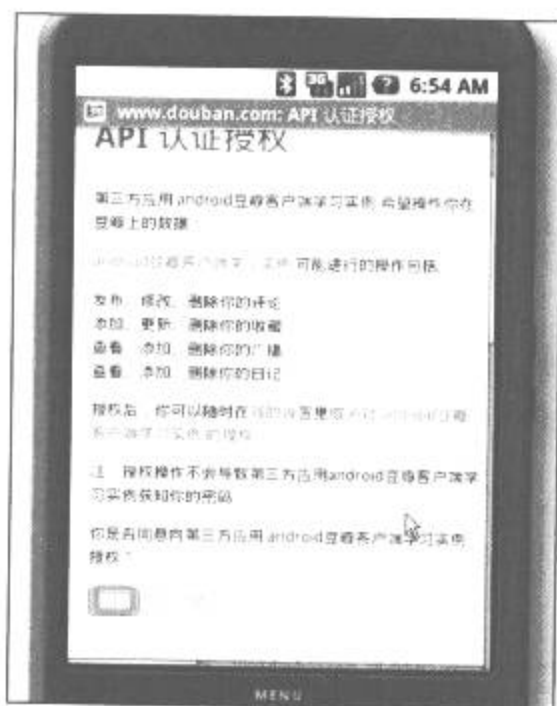
▲ 图 14-4 授权界面 (用户未登录)



▲ 图 14-5 用户登录界面

(4) 用户登录后显示 API 授权界面, 当单击确认后完成授权, 如图 14-6 所示。

(5) 完成授权后程序跳转到显示所有广播的界面, 如图 14-7 所示。



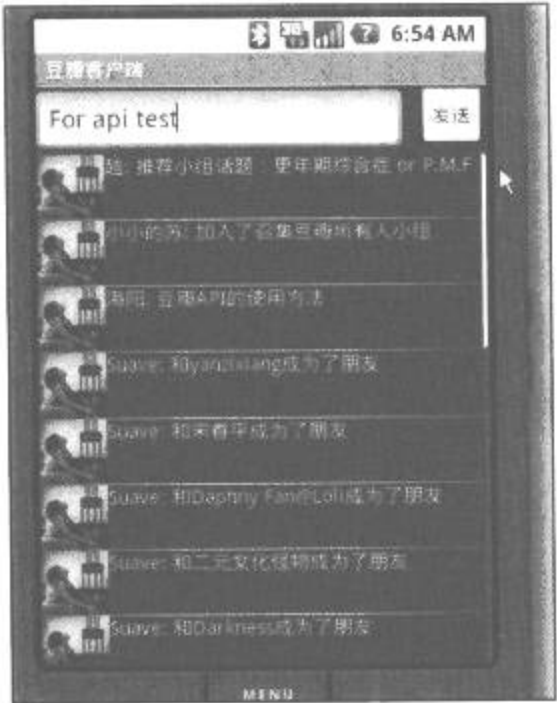
▲ 图 14-6 授权界面 (用户登录)



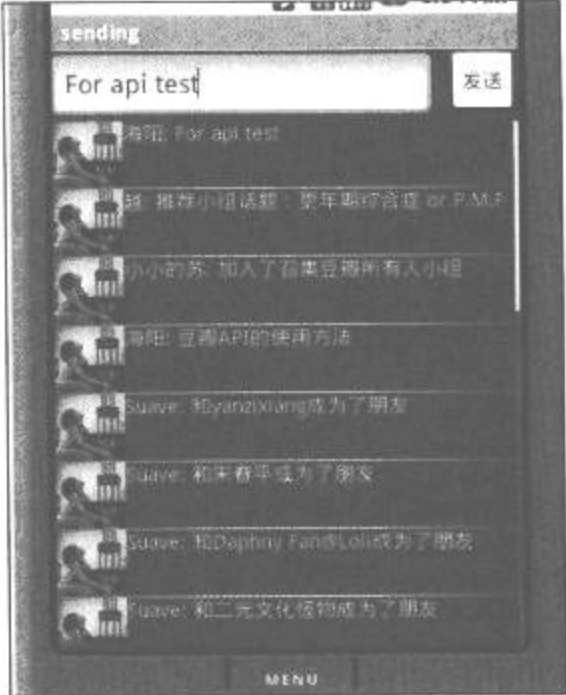
▲ 图 14-7 显示“我说”界面

(6) 在编辑框中输入“我说”的内容, 如图 14-8 所示。

(7) 单击发送后, “我说”的内容发送至服务器, 并且在屏幕上实时显示出来, 如图 14-9 所示。



▲ 图 14-8 在编辑框中输入“我说”内容（未单击发送）



▲ 图 14-9 在编辑框中输入“我说”内容（单击发送）

14.2.3 数据存储设计

在这个应用当中，我们没有涉及大量的存储，也没有涉及大规模的查询。需要存储的主要有以下几个值：

- 请求令牌的值；
- 和请求令牌对应的密钥；
- 授权令牌的值；
- 和授权令牌对应的密钥；
- 用户的 id。

对于这几个值，我们选择直接存储在 SharedPreferences 当中就可以了，事实上系统是把数据存到了个 XML 文档中，但是我们可以通过系统公布的接口进行存取和判断。下边就是从系统当中取出的 XML 文档的内容：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="oauth_token">4cc60c26310bba5ee652247293232f1a</string>
<string name="access_token_secret">bd781d2a4ed791bb</string>
<string name="access_token">d45f68eb3efe03f47d1e3ac1bcfa0bdc</string>
<string name="oauth_token_secret">2b1d963a133acaa6</string>
<string name="user_name">haiyangjy</string>
</map>
```

14.3 豆瓣网（Web 2.0）客户端实现

14.3.1 申请 API Key

在这里我们强烈建议大家在运行程序前申请一个自己的 API Key，一方面是熟悉怎么样申请一

个 API Key 和与这个 API Key 对应的密钥；另外一方面，豆瓣网对用户使用 API 访问豆瓣网的次数是有限制的。官方给出的说明是这样的：“如果只是想试验一下 API，豆瓣网也允许在不申请 API Key 的情况下进行 API 调用。不过在这种情况下，API 调用被限制为每分钟请求不超过 10 次。使用 API Key 时，对访问的限制较为宽松，为每分钟 40 次，超过限制的话会被封禁。”

在浏览器当中输入网址 `http://www.douban.com/service/apikey/apply` 进入申请 API Key 的界面，填写必要的内容后单击确定就申请到了自己的 API Key，并且看到还有一个和 API Key 对应的密钥，如图 14-10 所示。

打开 `com.douban.Android.util` 包下的 `DoubanUtil.java` 文件，将新申请的 API Key 和私钥分别替换到 `apiKey` 和 `secret` 变量当中。



▲ 图 14-10 申请 API Key

14.3.2 Activity 实现

首先我们来看一下程序的目录结构如图 14-11 所示。

在这个例子当中，我们有 5 个 Activity，其中有一个是抽象的 Activity，其余的 4 个为具体的实现。下面我们按照上一节中程序界面跳转的顺序来依次介绍每一个 Activity。

1. ActivityMain 的实现

按照开发实例的惯例，我们这个程序的入口 Activity 为 `ActivityMain`，在这个 Activity 当中实现了首次登录界面，如果用户登录过的话，会直接跳转到 `ActivityShowSaying` 中，实现代码如下所示：

```
package com.douban.Android;

import Android.app.Activity;
import Android.app.AlertDialog;
import Android.content.DialogInterface;
import Android.content.Intent;
import Android.content.SharedPreferences;
import Android.net.Uri;
import Android.os.Bundle;
import Android.util.Log;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.widget.Button;

import com.douban.Android.util.PreferencesUtil;

/**
 * @author haiyang 用户欢迎界面
 */
public class ActivityMain extends Activity {
```



▲ 图 14-11 程序的目录结构

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    if (!accessTokenIsValid())
        showAlertDialog();
    else {
        Intent intent = new Intent(ActivityMain.this,
            ActivityShowSaying.class);
        startActivity(intent);
    }
}

/*
 * 判断 SharedPreferences 中是否已经存储了 accessToken
 */
private boolean accessTokenIsValid() {
    SharedPreferences settings = getSharedPreferences(
        PreferencesUtil.preferencesDouban, 0);
    String requestToken = settings.getString(PreferencesUtil.accessToken,
        "false");
    if (requestToken.equals("false"))
        return false;
    else
        return true;
}

private void showAlertDialog() {
    final AlertDialog alert = new AlertDialog.Builder(this).create();
    alert.setMessage("这是您是第一次登录，只需要进行简单的配置，您就可以正常使用啦。");
    alert.setButton("登录", new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int whichButton) {
            alert.dismiss();
            Intent intent = new Intent(ActivityMain.this,
                ActivityAuth.class);
            startActivity(intent);
        }
    });
    alert.show();
}

protected void openWebBrowser(String url) {
    Log
        .i(this.getClass().getName(), "about to launch browser, url: "
            + url);
    Intent i = new Intent(Intent.ACTION_VIEW);
    i.setData(Uri.parse(url));
    startActivity(i);
}
}

```

代码解释:

- accessTokenIsValid()函数判断 accessToken 是否合法，如果用户是第一次登录，那么函数返回的 false，表示 accessToken 不合法。
- showAlertDialog()负责显示出用户第一次登录的提示框，单击按钮后，程序跳转到 ActivityAuth 中。

2. ActivityAuth 的实现

ActivityAuth 里主要实现了豆瓣网客户端的授权过程的前两个步骤:

- 获取未授权的请求令牌(Request Token);
- 请求用户授权 Request Token。

这两个步骤的理论讲解在本章第 1 节里有比较详细地讲解, 下面是这两个步骤的实现代码:

```
package com.douban.Android;

import Android.app.Activity;
import Android.content.Intent;
import Android.content.SharedPreferences;
import Android.net.Uri;
import Android.os.Bundle;
import Android.util.Log;
import Android.webkit.WebView;

import com.douban.Android.util.DoubanUtil;
import com.douban.Android.util.PreferencesUtil;
import com.google.gdata.client.douban.DoubanService;

/**
 * @author haiyangjy
 */
public class ActivityAuth extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        setTitle("豆瓣网");
        openWebBrowser(getRequestUrl());
    }

    private String getRequestUrl() {

        DoubanService myService = new DoubanService("subApplication",
            DoubanUtil.apiKey, DoubanUtil.secret, true);
        String url = myService.getAuthorizationUrl(DoubanUtil.callback);
        Log.i("jinyan", "url=" + url);

        SharedPreferences settings = getSharedPreferences(
            PreferencesUtil.preferencesDouban, 0);
        settings.edit().putString(PreferencesUtil.oauthToken,
            myService.getRequestToken()).putString(
            PreferencesUtil.oauthTokenSecret,
            myService.getRequestTokenSecret()).commit();

        return url;
    }

    protected void openWebBrowser(String url) {
        Log
            .i(this.getClass().getName(), "about to launch browser, url: "
                + url);
        Intent i = new Intent(Intent.ACTION_VIEW);
        i.setData(Uri.parse(url));
    }
}
```

```
startActivity(i);
}
}
```

代码解释:

- 在 `getRequestUrl()` 函数里边，我们主要得到了未授权的请求令牌 (Request Token) 和对应的请求令牌密钥 (Request Token Secret)，并把这两个值存储到 `SharedPreferences` 里边。
- 获得请求令牌 (Request Token) 之后，需要请求用户授权该请求令牌 Request Token。浏览器跳转豆瓣网的授权页面，提示用户授权给你的应用，以允许你的应用访问该用户在豆瓣网上的信息。这个过程是由 `openWebBrowser()` 函数实现的

3. ActivityAuthSuccess 的实现

当用户在豆瓣网的授权界面单击“同意”按钮后，Activity 跳转到 `ActivityAuthSuccess`，在这个 Activity 里边主要做了下边几件事情：

- 实现豆瓣网授权的第三个步骤：使用授权后的请求令牌 (Request Token) 换取访问令牌 (Access Token)；
- 保存用户的 id。

实现代码如下所示：

```
package com.douban.Android;

import java.io.IOException;
import java.util.ArrayList;

import Android.app.Activity;
import Android.content.Intent;
import Android.content.SharedPreferences;
import Android.os.Bundle;
import Android.util.Log;

import com.douban.Android.util.DoubanUtil;
import com.douban.Android.util.PreferencesUtil;
import com.google.gdata.client.douban.DoubanService;
import com.google.gdata.data.douban.UserEntry;
import com.google.gdata.util.ServiceException;

/**
 * @author haiyang 获取 token, 并且进行保存
 */
public class ActivityAuthSuccess extends AbstractActivity {
    DoubanService myService;
    SharedPreferences settings;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.auth_success);
        saveAccessToken();
        saveUserInfo();
        Intent intent = new Intent(ActivityAuthSuccess.this,
            ActivityShowSaying.class);
        startActivity(intent);
    }
}
```

```

private String saveAccessToken() {
    myService = new DoubanService("subApplication",
        DoubanUtil.apiKey, DoubanUtil.secret, true);
    String url = myService.getAuthorizationUrl(DoubanUtil.callback);

    settings = getSharedPreferences(
        PreferencesUtil.preferencesDouban, 0);
    String requestToken = settings.getString(PreferencesUtil.oauthToken,
        "false");
    String requestTokenSecret = settings.getString(PreferencesUtil.oauthTokenSecret,
        "false");
    Log.i("jinyan", "requestToken=" + requestToken);
    myService.setRequestToken(requestToken);
    myService.setRequestTokenSecret(requestTokenSecret);
    ArrayList<String> list = myService.getAccessToken();
    String accessToken = list.get(0);
    String accessTokenSecret = list.get(1);
    Log.i("jinyan", "accessToken=" + accessToken);
    Log.i("jinyan", "accessTokenSecret=" + accessTokenSecret);
    settings.edit().putString(PreferencesUtil.accessToken, accessToken)
        .putString(PreferencesUtil.accessTokenSecret, accessTokenSecret)
        .commit();

    return url;
}
/*
 * 保存用户的信息
 */
private void saveUserInfo() {
    Log.i("jinyan", "saveUserInfo");
    try {
        DoubanService Service = getAuthDoubanService();
        UserEntry userEntry = Service.getAuthorizedUser();
        settings.edit().putString(PreferencesUtil.userName, userEntry.getUid()).commit();
        Log.i("jinyan", "saveUserInfo"+userEntry.getUid());
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (ServiceException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}

```

代码解释:

- 在 saveAccessToken() 函数里边, 我们得到了访问令牌和与之对应的密钥, 并将其保存在了 SharedPreferences 当中。
- 在程序当中我们没有办法保存用户的 id 或者密码, 所以如果要得到用户的 id 信息, 必须调用豆瓣网的 API 访问服务器获得, 在方法 saveUserInfo() 里, 我们调用了豆瓣网的 API (被封装到 Java 库当中) 获得了当前登录用户的 id 信息, 并将其也保存在 SharedPreferences 中以备后面访问。

4. ActivityShowSaying 的实现

ActivityShowSaying 实现了这个例子当中最核心的两个功能:

- 豆瓣网中的“我说功能”;
- 豆瓣网中的“友邻广播功能”。

实现代码如下所示:

```
package com.douban.Android;

import java.io.IOException;
import java.util.ArrayList;

import Android.content.Context;
import Android.content.SharedPreferences;
import Android.os.Bundle;
import Android.util.Log;
import Android.view.View;
import Android.view.ViewGroup;
import Android.view.View.OnClickListener;
import Android.widget.AbsListView;
import Android.widget.BaseAdapter;
import Android.widget.Button;
import Android.widget.EditText;
import Android.widget.ImageView;
import Android.widget.LinearLayout;
import Android.widget.ListView;
import Android.widget.TextView;

import com.douban.Android.util.NetUtil;
import com.douban.Android.util.PreferencesUtil;
import com.google.gdata.client.douban.DoubanService;
import com.google.gdata.data.PlainTextConstruct;
import com.google.gdata.data.TextContent;
import com.google.gdata.data.douban.MiniblogEntry;
import com.google.gdata.data.douban.MiniblogFeed;
import com.google.gdata.util.ServiceException;

/**
 * @author haiyang 显示我说的界面
 */
public class ActivityShowSaying extends AbstractActivity {
    ListView listView;
    OnClickListener listener = null;
    EditText editText = null;
    Button button = null;
    SayingListAdapter sayingListAdapter = null;

    private class Saying {
        public String id;
        public String nickname;
        public String content;
        public String time;
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 显示列表
        setContentView(R.layout.show_saying);
        listView = (ListView) findViewById(R.id.saying_list);
        sayingListAdapter = new SayingListAdapter(this,
            getSayingArrayFromInternet());
        listView.setAdapter(sayingListAdapter);
        editText = (EditText) findViewById(R.id.content);
    }
}
```

```

listener = new OnClickListener() {
    public void onClick(View v) {
        setTitle("sending");
        DoubanService myService = getAuthDoubanService();
        try {
            sayingListAdapter.addSaying(myService
                .createSaying(new PlainTextConstruct(editText
                    .getText()).toString()));
        } catch (IOException e) {
            setTitle("sending error");
            e.printStackTrace();
        } catch (ServiceException e) {
            setTitle("sending error");
            e.printStackTrace();
        }
    }
};
button = (Button) findViewById(R.id.send);
button.setOnClickListener(listener);
}

/*
 * 得到我说的内容
 */
private ArrayList<Saying> getSayingArrayFromInternet() {
    ArrayList<Saying> sayingList = new ArrayList<Saying>();
    SharedPreferences settings = getSharedPreferences(
        PreferencesUtil.preferencesDouban, 0);

    String userId = settings.getString(PreferencesUtil.userName,
        "haiyangjy");
    MiniblogFeed mf;
    try {
        DoubanService myService = getAuthDoubanService();
        mf = myService.getContactsMiniblogs(userId, 1, 20);
        for (MiniblogEntry me : mf.getEntries()) {
            Saying saying = new Saying();
            String nickname = me.getAuthors().get(0).getName();
            String content = ((TextContent) me.getContent()).getContent()
                .getPlainText();
            saying.content = content;
            saying.nickname = nickname;
            sayingList.add(saying);
        }

    } catch (IOException e) {
        e.printStackTrace();
    } catch (ServiceException e) {
        e.printStackTrace();
    }
    return sayingList;
}

/**
 * 向服务器发送我说的内容
 *
 * @param content
 */
private void sendSaying(String content) {
    if (content == null && content.length() == 0)
        return;
    DoubanService myService = getAuthDoubanService();
    try {

```



```

MiniblogEntry miniblogEntry = myService
    .createSaying(new PlainTextConstruct(content));

} catch (IOException e) {

e.printStackTrace();
} catch (ServiceException e) {

e.printStackTrace();
}

}

public class SayingListAdapter extends BaseAdapter {
    private Context mContext;
    private ArrayList<Saying> sayingList;

    public SayingListAdapter(Context context, ArrayList<Saying> sayingList) {
        mContext = context;
        SayingListAdapter.this.sayingList = sayingList;
    }

    @Override
    public int getCount() {

        return sayingList.size();
    }

    @Override
    public Object getItem(int position) {

        return position;
    }

    @Override
    public long getItemId(int position) {

        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        SayingView sayingView;
        Saying saying = sayingList.get(position);
        if (convertView == null) {
            sayingView = new SayingView(mContext, saying);
        } else {
            sayingView = (SayingView) convertView;
            sayingView.setNickname(saying.nickname);
            sayingView.setContent(saying.content);
        }

        return sayingView;
    }

    /**
     * 实时显示增加的我说
     *
     * @param miniblogEntry
     */
    public void addSaying(MiniblogEntry miniblogEntry) {
        Saying newSaying = new Saying();
        newSaying.content = ((TextContent) miniblogEntry.getContent())
            .getContent().getPlainText();
    }

```

```

newSaying.nickname = miniblogEntry.getAuthors().get(0).getName();
sayingList.add(0, newSaying);
notifyDataSetChanged();
}

}

/**
 * 我说列表的每一个条目
 *
 * @author Administrator
 */
public class SayingView extends LinearLayout {
    private TextView tNickname;
    private TextView tContent;
    private ImageView imageView;
    private Saying saying;
    private Context mContext;

    public SayingView(Context context, Saying saying) {
        super(context);
        mContext = context;
        SayingView.this.saying = saying;
        imageView = getImageViewFromLocal();

        addView(imageView);
        tNickname = new TextView(context);
        tNickname.setText(saying.nickname+": ");
        addView(tNickname, new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT, 48));

        tContent = new TextView(context);
        tContent.setText(saying.content);
        addView(tContent, new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT, 48));
    }

    /**
     * 重新设置昵称
     *
     * @param nickname
     */
    public void setNickname(String nickname) {
        tNickname.setText(nickname+": ");
    }

    /**
     * 重新设置我说的内容
     *
     * @param content
     */
    public void setContent(String content) {
        tContent.setText(content);
    }

    /**
     * 得到本地的默认图片
     */
    private ImageView getImageViewFromLocal() {
        ImageView imageView = new ImageView(mContext);
        imageView.setImageResource(R.drawable.default_head);
        imageView.setAdjustViewBounds(true);
    }

```

```

imageView.setLayoutParams(new AbsListView.LayoutParams(
LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
return imageView;
}
}
}

```

和这个 Activity 对应的 layout 是:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <RelativeLayout android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <Button android:id="@+id/send"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10dip" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="发送" />
        <EditText android:id="@+id/content" android:maxLines="1"
            android:layout_marginTop="2dip" android:layout_width="wrap_content"
            android:ems="25" android:layout_height="wrap_content"
            android:autoText="true" android:capitalize="sentences"
            android:scrollHorizontally="false" android:layout_toLeftOf="@id/send"
            android:layout_alignTop="@id/send" />

    </RelativeLayout>

    <FrameLayout android:layout_width="fill_parent"
        android:layout_height="0dip" android:layout_weight="1">
        <ListView android:id="@+id/saying_list"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:drawSelectorOnTop="false" />
    </FrameLayout>

</LinearLayout>

```

这个 Activity 最后实现的界面如图 14-12 所示。

- 在图 14-12 中的输入框和发送按钮和布局文件中的 RelativeLayout 部分对应。

- 友邻广播列表和布局文件当中的 FrameLayout 对应。

下面我们对 ActivityShowSaying 中的源码进行分析, 看一下到底是如何实现“我的友邻广播显示”和“我说”功能。

代码解释:

- 首先我们看一下 Saying 类, 这个类是一个内部类, 它封装了和列表中一行所绑定的数据, 换句话说就是列表中的一行绑定一个 Saying 的数据类。它里边有 4 个值, 分别是用户的 id, 用户的昵称, 内容还有时间。当然, 如果还有更多信息的话, 我们也可以进行扩展。

- SayingView 是列表中某一行的视图。在我们这个例子当中, 每一行的视图包括 3 个元素, 分别是:

- 一个 ImageView, 用于显示用户的头像;
- 两个 TextView, 一个用于显示用户的昵称, 另外一个用于显示用户广播的内容。



▲ 图 14-12 显示友邻广播页面

- getSayingArrayFromInternet()函数负责调用豆瓣网的 API 从服务器取得广播的内容。
- 首先我们将 userId 从 SharedPreferences 取出来。
- 其次调用 getAuthDoubanService()函数得到一个授权的 DoubanService。
- 再次调用 DoubanService 的 getContactsMiniblogs()方法得到广播的一个列表。
- 最后我们将 MiniblogEntry 类型的“友邻广播列表”转换成我们所需要的 Saying 类型的 ArrayList。
- SayingListAdapter 是我们直接继承 BaseAdapter 而实现的,在这个类当中有 4 个方法需要重写,其中比较重要的是 getView (int position, View convertView, ViewGroup parent) 这个方法。这个方法有 3 个参数,第一个参数表示在 adapter 中的数据集里边的位置,第二个参数我们可以重用的 View,这个 View 也就是 ListView 当中的某一项。第三个参数表示包含这个 View 的父容器。
- 最后我们来看一下 onCreate()方法。在这个方法里边我们做了两件事情:
 - 通过 findViewById (R.id.saying_list) 得到一个 ListView,通过 new SayingListAdapter(this,getSayingArrayFromInternet())得到一个 SayingListAdapter。并将它们两个绑定起来,这样我们的 ListView 就可以显示出来了。
 - 通过 findViewById (R.id.send) 得到我们的发送 button,然后新建一个 OnClickListener,再将它们俩绑定在一起。这样单击发送按钮后就可以将“我说”发送至服务器。具体发送是调用 DoubanService 的 createSaying()方法来实现的。

5. AbstractActivity 的实现

AbstractActivity 是一个抽象的 Activity,其中 ActivityAuthSuccess 和 ActivityShowSaying 继承这个 AbstractActivity。AbstractActivity 里边只实现了一个函数 getAuthDoubanService(),这个函数主要是获得一个授权后的豆瓣网服务,这样就可以直接使用 Java 豆瓣网库的 API 来调用豆瓣网开放出来的 API。具体实现代码如下所示:

```
package com.douban.Android;

import Android.app.Activity;
import Android.content.SharedPreferences;

import com.douban.Android.util.DoubanUtil;
import com.douban.Android.util.PreferencesUtil;
import com.google.gdata.client.douban.DoubanService;

public abstract class AbstractActivity extends Activity {
    /**
     * 得到授权后的豆瓣网服务。
     * @return
     */
    public DoubanService getAuthDoubanService() {

        DoubanService myService = new DoubanService("subApplication",
            DoubanUtil.apiKey, DoubanUtil.secret, false);
        SharedPreferences settings = getSharedPreferences(
            PreferencesUtil.preferencesDouban, 0);

        String accessToken = settings.getString(PreferencesUtil.accessToken,
            "false");
        String accessTokenSecret = settings.getString(
            PreferencesUtil.accessTokenSecret, "false");
```

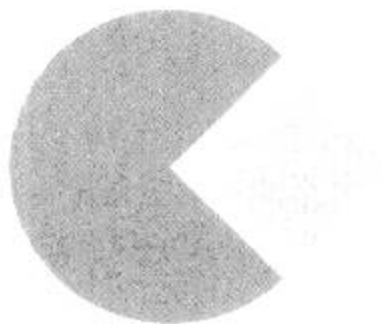
```
myService.setAccessToken(accessToken, accessTokenSecret);  
return myService;  
}  
}
```

 代码解释:

- 这个 Activity 中主要是实现了方法 `getAuthDoubanService`, 通过这个方法就可以获得一个授权后的 `DoubanService`, 通过这个 `DoubanService` 我们就可以访问豆瓣的 API。

14.4 本章小结

本实例是一个真正的 Android 系统和标准的 Web2.0 网站交互的例子。通过这个例子我们了解了豆瓣网的 API, 以及在 Android 系统如何利用这些 API 和豆瓣网进行交互。同时在这个例子当中我们也学习了 OAuth 认证, 这是一种非常重要的认证, 尤其大部分 Web 2.0 的网站, 更是采用这种认证方式。学完这个例子, 相信大家已经想跃跃欲试地准备开发应用了, 别着急, 让我们学完剩下章节, 相信更会让你如虎添翼, 游刃有余地进行移动互联应用的开发。



第 15 章 Android 综合案例四—— 在线音乐播放器

学习目标

- 掌握在线播放器的需求分析; ☐
- 了解 YOBO 及其特点; ☐
- 掌握播放器构架; ☐
- 掌握播放器的实现过程; ☐
- 掌握播放器演示。 ☐

15.1 关于 YOBO 和 YOBO API

随着移动设备的日益完善，我们大家都已经有了在手机上听自己喜欢歌曲的习惯，以往的做法是用数据线或者蓝牙等无线设备将 PC 上的歌曲复制几首到手机中，反反复复的听，等到听厌了以后，再次装一些歌曲进去，如此反复。但是这实在是麻烦。我们试想，要是能在我们的手机上在线听歌该有多好呀。但是在以前，这很不现实，困扰我们的有 3 大问题：

- 网速太慢，流量太贵；
- 没有地方可以听歌；
- 没有软件可以用。

然而，现在随着 3G 在中国以及世界的普及，Wifi 的扩展，速度问题马上即可解决；随着 YOBO 的迅速崛起和不断完善，使其成为我们网上听歌、收藏歌曲和推荐歌曲的理想选择。

本章就是要和大家一起来实现一个基于 YOBO API 的音乐播放器，使得我们可以非常方便地在手机上收听 YOBO 上的歌曲，这得益于 YOBO 提供的音乐服务和逐步完善的无线网络速度，使得我们能随时随地收听我们喜欢的歌曲的梦想成为可能。

首先需要和一些不熟悉 YOBO 的读者介绍一下，YOBO 诞生于 2006 年 12 月，是一项帮助您发现自己的音乐“DNA”、探索更多好音乐、分享音乐爱好和收藏、结交音乐知音的服务。在那里，其通过最先进的人工智能技术帮助您找到最喜爱的音乐，创建自己的个人电台，并和朋友分享！经过 2 年多的发展和完善，已经得到无数用户的喜爱，成为众多歌曲爱好者在线听歌的首选。2008 年底，YOBO 为了给更多的人提供其优质的服务，开放了 YOBO API，通过 YOBO API，使得第三方可以在自己的网站或者桌面、移动设备上集成 YOBO 的服务。而我们将要实现的这个在线播放器也正是基于 YOBO API。

15.1.1 YOBO 简介

YOBO 起源于“音乐 DNA 项目”，是目前中国较全面精准的音乐分析与个性化推荐系统。只要告诉 YOBO 一首您最喜欢的歌曲或者一名歌手，YOBO 电台就会为您建立一个流动的音乐频道，开始探索无穷的音乐世界。YOBO 可以帮助您在已经听过的基础上发现新的歌手，并通过您的音乐“DNA”找到品位和性格最接近的音乐同类。

YOBO 的个性化音乐推荐是基于其长久以来对“音乐 DNA”的研究。这是国内目前为止全面精细的音乐分析。几十名专业音乐人志愿加入了其庞大的音乐分析计划，仔细的聆听每一首歌曲，分辨记录每首歌曲的众多的音乐元素——旋律，和声，配器，节奏，人声，歌词，情感……近 50 种特征，400 多个可能的标签，一首歌需要 20 分钟完成标定和入库工作。并依据每个人独有的性格和情绪，尝试为音乐爱好者们推送他们有可能喜欢的歌曲。是一个共同的梦想激励着这群年轻的音乐人每天狂热的积蓄着这样的工作，以便在最快时间内让 YOBO 的用户得到精准的推荐，跟得上华语，日韩，欧美，以至世界乐坛的新潮流。

YOBO 只专心做一件事情：“发掘你的音乐 DNA!” YOBO 要为用户解决以下问题：

- 想听歌却不知道听什么歌；
- 工作忙碌后再也没空去找歌；
- 对着搜索引擎不知道搜什么歌；
- 挖掘排行榜外那些鲜为人知的好歌。

YOBO 知道音乐对生活的重要，他们用尽一些办法，为用户找到属于你的音乐和音乐生活！

15.1.2 YOBO 功能特点

按照我们的这个播放器的功能，需要熟悉一下 YOBO 的功能特点。YOBO 主要功能有 3 个。

(1) 音乐推荐：按照你的音乐“DNA”和以往的收听记录，音乐喜好，朋友及其朋友的音乐喜好为基础，通过数据挖掘后给你推荐的歌曲。

(2) 音乐收藏：在 YOBO 可以对自己喜欢的歌曲进行收藏，收藏后的音乐存放在自己的音乐盒中，供其他时间收听。也就是说，其给我们保存了喜欢的歌曲列表。

(3) 音乐收听：在 YOBO，你可以选择多种收听方式，你可以收听依据一个种子（比如一首歌曲或者一个艺人）收听其相似歌曲，也可以收听你创建的音乐电台，可以收听自己收藏的音乐盒获取其他的类似于排行榜之类的榜单。

而这其中，涉及诸如歌手、专辑、歌曲、音乐电台、音乐盒、榜单等多种概念以及对这些概念可能存在的一些动作（比如喜欢、讨厌、分享等），我们将要实现的是在手机上完成这些功能。

15.1.3 YOBO API 介绍

到本章写作之时，YOBO API 还在处于 Beta 阶段，其自身还在不断的发展和完善阶段，就目前其提供的功能而言，已经包含了主要的功能，包含但不限于如下列表：

- 用户鉴权和授权；
- 获取歌手列表/歌手详细信息；

- 获取专辑列表/专辑详细信息;
- 歌曲列表/歌曲详细信息;
- 音乐盒列表/音乐盒详细信息;
- 音乐电台列表/音乐电台详细信息;
- 相似艺术家列表;
- 相似歌曲列表;
- 歌曲的动作, 喜欢, 讨厌, 分享;
- 歌手的动作, 喜欢, 讨厌;
- 好友列表;
- 站内短消息。

其 API 还在不断的修订和完善, 更多的功能逐步开放出来, 就目前的需求来看, 这些现有的 API 已经够用了。

15.2 在线播放器架构设计

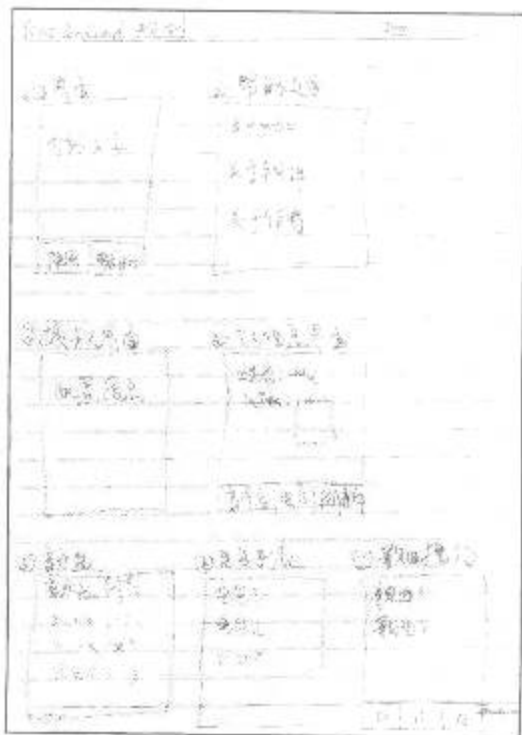
15.2.1 在线播放器功能规划设计

按照我们的需求, 可以大致规划一下这个在线播放器的主要功能和模块。

- (1) 用户注册: 使得尚未有 YOBO 账号的手机用户可以方便地注册。
- (2) 用户登录: 已经有账号的可以登录。
- (3) 音乐电台: 可以获取自己或者指定用户的音乐电台。
- (4) 音乐盒: 可以获取自己或者指定用户的音乐盒。
- (5) 歌手: 可以获取指定条件的歌手列表以及歌手的详细信息 (专辑、歌曲信息)。
- (6) 专辑: 可以获取满足一定条件的专辑列表和专辑信息。
- (7) 歌曲: 可以获取满足一定条件的歌曲列表和歌曲信息。
- (8) 好友: 可以获得自己或者指定用户的好友列表。
- (9) 短消息: 可以获得自己或者指定用户的短消息。

15.2.2 在线播放器 UI 设计

按照我们前面的功能设计, 我们将主要的 UI 设计出来, 按照前面的功能规划, 我们一步一步的勾勒出其 UI 的轮廓, 这里我还是使用我比较喜欢的用铅笔在纸上勾画。按照我们设想, 其最开始的主界面上有个简单的介绍, 像不了解 YOBO 的用户解释 YOBO 是做什么的; 进而开始使用时, 需要进行授权; 已经授权用户的音乐盒、音乐电台的各期列表及其歌曲播放几个比较重要的界面。我们在纸上勾勒出其大概如图 15-1 所示。



▲ 图 15-1 规划设计

15.2.3 在线播放器数据存储设计

设计完 UI，我们大概可以构思出整个应用将以怎样的形态呈现在使用者面前，接下来需要梳理一下哪些数据需要存储，以及采用哪种策略存储。

在这个应用中，我们需要的数据有两类，一类是一些比较简单的个性化配置信息，如登录的用户名、密码或者授权码之类的；还有一类是需要的应用数据，如歌手信息、专辑信息等。

第一类数据我们需要在用户第一次使用的时候获取存储起来备用的，而第二类数据我们是可以通过 YOBO 提供的 API 实时获取的，并不需要存储。综上所述，我们需要保存的数据还是挺少的，可以采用 SharedPreferences 来存储。关于 SharedPreferences 存储数据的例子我们在第 8 章已经讲得比较详细了，这里就不再赘述。

15.3

在线播放器的编程实现

15.3.1 申请 API Key

按照 YOBO API 的官方说明，每个需要访问 YOBO 上资源的第三方应用都需要注册，获取一个 API Key 和其对应的 API Secret，打开官方网页 (<http://api.yobo.com>)，进入注册页面，如图 15-2 所示。

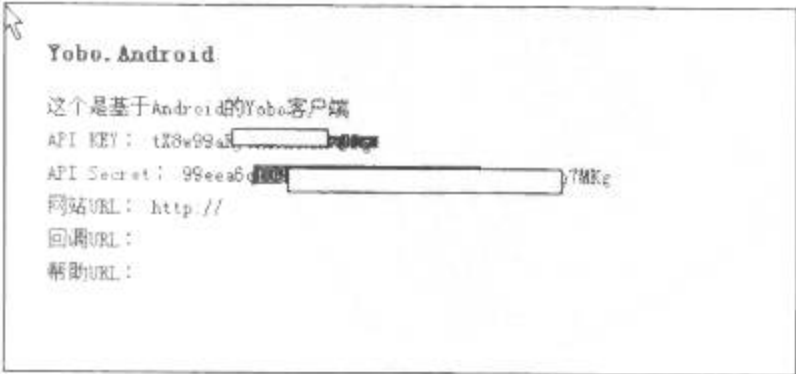


▲ 图 15-2 注册页面

在图 15-2 所示的界面中输入应用的名字和一些描述信息，注意其回调 URL (Callback_url) 对基于第三方的 Web 应用比较重要，对于我们这个运行在 Android 上的应用没有什么用处，单击“注册”后，会看到比较重要的信息，如图 15-3 所示。

图 15-3 中几个信息分别代表：

- API Key：给这个应用分配的 API Key；



▲ 图 15-3 重要的信息

- API Secret: 密钥, 需要保管好, 以后 API 签名鉴权需要它;
- 网站 URL: 第三方应用的主页地址;
- 回调 URL: 鉴权后回调的第三方应用的地址;
- 帮助 URL: 第三方应用的帮助文档地址。

15.3.2 基础功能实现

本小节用来实现该应用的基础功能点, 涉及签名算法的实现、数据的提交和接收, 数据的解析及显示等。

1. 签名算法

按照 YOBO API 的说明, 使用 API 前, 需要对需要传递的参数和自己的 API Key 和 Secret 进行签名, 按照 API 的说明, 其必须的参数及其意义分别为:

- API Key: 就是签名注册应用时分配的 Key;
- nonce: 这个是一个随机值;
- timestamp: 时间戳。

另外, 还需要对我们传递的其他参数按照键值的字母升序排列后组成<name><value>secret, 然后用 MD5 算法加密, 代码如下所示:

```
// 对需要传递的参数进行签名
// 返回签名加密串
public static String sig_params(Hashtable params) {
    Vector v = new Vector<String>(params.keySet());
    Collections.sort(v);
    // Display (sorted) hashtable.
    String result = "";
    for (Enumeration e = v.elements(); e.hasMoreElements();) {
        String key = (String) e.nextElement();
        String val = (String) params.get(key);
        result += key;
        result += val;
        Log.i("sig_params", "Key: " + key + ",Val: " + val);
    }
    Log.i("sig_params", "result:" + result);
    return result;
}
```

 代码解释:

上述代码中, 首先对键值进行升序排列, 然后依次取其对应的值, 并按照 API 签名要求拼接成一个字符串, 返回结果。

然后还有一个比较重要的方法, MD5 的实现代码如下所示:

```
public static String md5(String s) throws NoSuchAlgorithmException {
    System.out.println(System.currentTimeMillis());
    MessageDigest m = MessageDigest.getInstance("MD5");
    m.update(s.getBytes(), 0, s.length());
    String md5_str = new BigInteger(1, m.digest()).toString(16);
    Log.i("MD5", md5_str);
    return md5_str;
}
```


 代码解释:

创建一个 **MessageDigest**, 然后将传入的字符串加密成 MD5 的串。

另外, 还需要生成需要的 **timestamp** 和 **nonce**, 代码段如下所示:

```
long timestamp = System.currentTimeMillis();
String nonce = YOBUUtil.md5(String.valueOf(System
    .currentTimeMillis()) + PreferencesUtil.api_secret);
Hashtable<String, String> params = new Hashtable<String, String>();
params.put("api_key", PreferencesUtil.api_key);
params.put("nonce", nonce);
params.put("timestamp", String.valueOf(timestamp));
```

 代码解释:

这里的时间戳 (**timestamp**) 可以很方便地通过调用 **System.currentTimeMillis()** 获取, 而 **nonce** 是一个不能重复的随机值, 我们这里使用时间戳加上自己的 API 的密钥 (**api_secret**), 然后在用 MD5 编码一下。

2. 数据的提交和接收

接着看一下在 Android 中如何实现 HTTP 的 Get 方法来请求数据, 并取回结果, 实现代码如下所示:

```
public static String sendGetRequest(String endpoint,
    String requestParameters) {
    Log.i("sendGetRequest", endpoint);
    String result = null;
    if (endpoint.startsWith("http://")) {
        // 发送 get 请求
        try {
            // 构建数据
            StringBuffer data = new StringBuffer();
            // 发送数据
            String urlStr = endpoint;
            if (requestParameters != null && requestParameters.length() > 0) {
                urlStr += "?" + requestParameters;
            }
            Log.i("urlStr", urlStr);
            URL url = new URL(urlStr);
            URLConnection conn = url.openConnection();
            // 得到响应
            BufferedReader rd = new BufferedReader(new InputStreamReader(
                conn.getInputStream()));
            StringBuffer sb = new StringBuffer();
            String line;
            while ((line = rd.readLine()) != null) {
                sb.append(line);
            }
            rd.close();
            result = sb.toString();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    Log.i("sendGetRequest", result);
    return result;
}
```

 代码解释:

上述代码中, 方法 **sendGetRequest** 将 **requestParameters** 提交到 **endpoint** 的地址上, 并取得反馈信息。

3. 数据的解析

紧接着需要考虑解析返回的数据，取出需要的部分，如下是一个简单的方法，实现代码如下：

```
//解析 RequestToken 返回的 XML，得到 Token 的值
public static String paxToken(String xml) {
    String regEx = "<token>(.*?)</token>";
    Pattern p = Pattern.compile(regEx);
    Matcher m = p.matcher(xml);
    boolean rs = m.find();
    if (rs) {
        System.out.println("开始位置是：" + m.start() + "结束位置是：" + m.end()
            + "字符串是：" + m.group());
    }
    return m.group(1);
}
```

 代码解释：

上述代码中，方法 `paxToken` 用于将返回的 XML 数据中的 `token` 取出来，后面我们将会看到这个方法的使用场景。

4. 显示网络上的图片

在我们的这个例子中，我们会按照 API 中返回的数据，将类似于专辑封面之类的图片显示在手机屏幕上，在 Android 当中显示一张网络图片的时候，其实是比较麻烦的。首先得把这个网络图片转换成 Java 的 `InputStream` 流，然后再把这个流转换成一个 `Bitmap`。`Bitmap` 是可以作为参数传给 `ImageView` 的。下面的 `returnBitmap` 函数是最核心的，也是大家可以重用的，它负责把一个 URL 的网络图片变成一个本地的 `Bitmap`，代码如下所示：

```
public Bitmap returnBitmap(String url) {
    URL myFileUrl = null;
    Bitmap bitmap = null;
    try {
        myFileUrl = new URL(url);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    try {
        HttpURLConnection conn = (HttpURLConnection) myFileUrl
            .openConnection();
        conn.setDoInput(true);
        conn.connect();
        InputStream is = conn.getInputStream();
        bitmap = BitmapFactory.decodeStream(is);
        is.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return bitmap;
}
```

 代码解释：

这里获取指定地址的图片，调用 `BitmapFactory.decodeStream()` 方法获取一个 `Bitmap` 对象，就可以直接在 Android 中使用了。

15.3.3 实现 Activity

按照我们前面的功能设计和 UI 规划，我们以此实现需要的 Activity 及对应的界面。

1. 主界面

按照我们前面的设计稿，主界面比较简单，主要用来介绍该软件的用途等信息，并提供两个菜单，分别是“开始使用”和“查看帮助”，先看 onCreate 的实现，代码如下所示：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    setTitle(R.string.app_title);
}
```

 代码解释：

上述代码很简单，使用 setContentView 绑定一个名字为 main 的 layout，再使用 setTitle 设定一下标题，我们看下 main.xml 的代码，具体如下所示：

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

    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:background="@drawable/logo" />

    <TextView android:layout_width="fill_parent"
        android:paddingTop="14dip"
        android:layout_height="wrap_content" android:text="@string/app_desc"
        android:textSize="16sp" />

</LinearLayout>
```

 代码解释：

上述代码采用 LinearLayout 布局，显示一个背景图和一些文字介绍，不再赘述。

接着看生成菜单的函数 onCreateOptionsMenu，实现代码如下所示：

```
// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, MENU_START, 0, R.string.menu_start)
        .setIcon(R.drawable.menu_start)
        .setAlphabeticShortcut('S');
    menu.add(0, MENU_HELP, 0, R.string.menu_helps).setIcon(
        R.drawable.helps).setAlphabeticShortcut(
        'H');
    return true;
}
```

 代码解释：

上述代码显示，生成两个菜单，MENU_START 和 MENU_HELP，其对应的响应方法如下：

```
// 当一个菜单被选中的时候调用
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Intent intent = new Intent();
    switch (item.getItemId()) {
        case MENU_START:
            intent.setClass(Main.this, Auth.class);
            startActivity(intent);
            return true;
        case MENU_HELP:
            intent.setClass(Main.this, Helps.class);
            startActivity(intent);
            break;
    }
    return true;
}
```

代码解释:

上述代码所示, 当我们选中 MENU_START 的时候, 则启动 Auth.class 对应的 Activity; 如果选择了 MENU_HELP, 则显示 Helps.class 对应的 Activity。

主界面相对来说比较简单, 如此处理的好处是加快应用的启动速度, 让用户等待尽可能短的时间, 加入一些对软件的介绍文字, 让用户更快地了解这个应用是做什么的, 其整个效果如图 15-4 所示。



▲ 图 15-4 主界面效果图

2. 帮助界面

在前面的主界面上, 我们看到其提供了两个菜单, “开始使用”和“查看帮助”, 我们接着先看看“查看帮助”界面的实现。帮助页面和主界面差不多, 也只是文字显示, 其上主要显示该应用的一些使用方法和帮助信息, 直接看文件 helps.java 的代码, 其 onCreate 的实现代码如下所示:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.helps);
    setTitle("YOBO: 帮助信息");
}
```

代码解释:

上述代码就是绑定一个名字为 helps 的 layout, 其他的文字都是在 helps.xml 中添加的, 其代码如下所示:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="关于 YOBO:"
        android:textSize="18sp" android:paddingTop="14dip" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:autoLink="all"
        android:text="@string/helps_YOBO" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="关于 YOBO.Android:"
        android:textSize="18sp" android:paddingTop="14dip" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:autoLink="all">
```

```

android:text="@string/helps_text" />
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content" android:text="关于作者:"
android:textSize="18sp" android:paddingTop="14dip" />
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content" android:autoLink="all"
android:text="@string/author_text" />
</LinearLayout>

```

代码解释：

helps.xml 布局文件中使用一些 TextView 组件用来显示一些帮助信息，只是一些文字的绑定和文字格式的设定，其最终效果如图 15-5 所示。

3. 用户授权界面

看完“查看帮助”界面，我们紧接着看主界面上的另一个菜单“开始使用”的实现，这个界面的逻辑比较复杂，我们先整理一下开发路线。

用户使用这个软件，主要是随时随地的收听他在 YOBO 上的音乐盒或者电台歌曲，所以，我们首先需要将其和一个用户账号绑定，也就是用户给这个应用授权，使其可以访问其在 YOBO 上的个人数据（如个人资料、音乐盒、音乐台等）。按照 YOBO 的 API 文档，我们知道，其授权方式采用的是类似 OAuth 的方式，采用打开浏览器导航到授权页面，让用户给该软件授权，具体的实现方式我们后面会说到，我们一步一步结合代码来讲解。首先来看一下 onCreate 方法，实现代码如下所示：

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getSessionInPerence().equalsIgnoreCase("")) {
        setContentView(R.layout.auth);
        setTitle("YOBO: 用户授权");
        showAlertDialog();
    } else {
        setContentView(R.layout.uinfo);
        setTitle("YOBO: 用户信息");
        findViews();
        restorePrefs();
    }
}

```

代码解释：

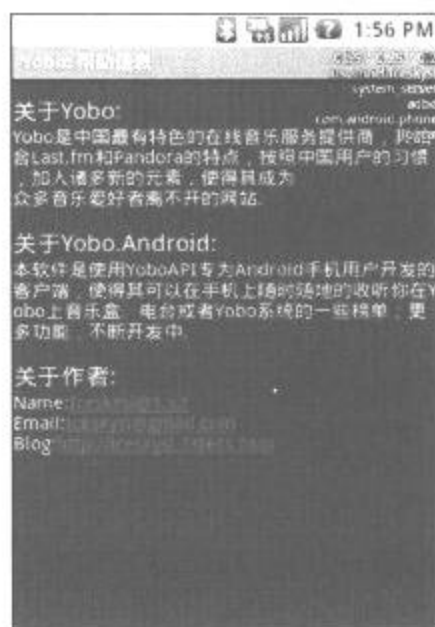
在上述代码中，我们有个逻辑，首先调用 getSessionInPerence() 看看是否已经授权，如果已经授权，就绑定 uinfo.xml 显示已经授权的用户个人信息；如果没有授权，则用 showAlertDialog() 显示一个对话框提示用户进行授权。

我们先看 getSessionInPerence() 的实现，代码如下所示：

```

// 判断 SharedPreferences 中是否已经存储了 acessToken
private String getSessionInPerence() {
    SharedPreferences settings = getSharedPreferences(
        PreferencesUtil.preferencesSetting, 0);
    String session = settings.getString(PreferencesUtil.session, "");
}

```



▲ 图 15-5 帮助界面


```

if (session.equals(""))
    return "";
else
    return session;
}

```

代码解释：

主要是查看 `SharedPreferences` 中是否已经保存了用户授权的 `session key`，如果没有则认为还没授权；如果存在，则返回这个 `key` 的值。

我们先看未被授权的情况下是怎么样的，在上面的代码中，我们看到会调用 `showAlertDialog()` 显示一个对话框，代码如下所示：

```

// 显示对话框
private void showAlertDialog() {
    final AlertDialog alert = new AlertDialog.Builder(this).create();
    alert.setTitle("配置");
    alert.setMessage(getString(R.string.app_start));
    alert.setButton("授权", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            alert.dismiss();
            try {
                openWebBrowser(Tools.getAuthorizeUrl(getToken()));
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
        }
    });
    alert.setButton2("注册", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            alert.dismiss();
            try {
                openWebBrowser(Tools.getRegUrl(getToken()));
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
        }
    });
    alert.show();
}

```

代码解释：

如上代码，将显示一个对话框，其上有两个按钮，分别是“授权”和“注册”，单击两个按钮都将调用一个 `openWebBrowser()` 方法打开到指定页面进行注册或授权，其效果如图 15-6 所示。

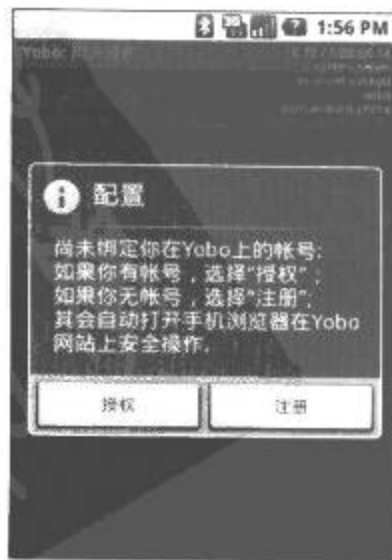
在图 15-6 中，其两个按钮是比较关键的，如果用户已经有了 YOBO 的账号，则使用“授权”给其应用访问其账户的权限即可；如果用户还没有 YOBO 的账号，则可以使用“注册”来注册一个新的账号并授权。

我们分别来看，首先是“授权”按钮的实现，在如上代码中，当单击“授权”按钮后，将执行 `openWebBrowser(Tools.getAuthorizeUrl(getToken()))` 这段代码，先看 `openWebBrowser()` 的实现，代码如下所示：

```

// 打开浏览器，定位到指定的地址
protected void openWebBrowser(String url) {
    Log.i(TAG, "about to launch browser, url: " + url);
}

```



▲ 图 15-6 未授权时的提示界面

```
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse(url));
startActivity(i);
}
```

代码解释:

上述代码实现传入一个 Url 地址, 启动一个 Activity, 载入这个地址的内容, 展示出来。我们主要看 `Tools.getAuthorizeUrl(getToken())` 这个的实现, 从外往里分析这个方法, 先看 `getAuthorizeUrl()` 的实现, 代码如下所示:

```
// 构造用户授权 URL
public static String getAuthorizeUrl(String token) {
    String url = "";
    url = PreferencesUtil.baseUrl + PreferencesUtil.authorizeUrl
    + "?token=" + token;
    return url;
}
```

代码解释:

这个方法也比较明了, 其接收一个 token 参数, 按照申请 API Key 的时候得到的一些 Url 构造授权的链接地址, 其中 `PreferencesUtil` 中定义了 YOBO 接口中的一些基本信息, 如 API Key 等。

紧接着看 `getToken()` 函数, 其主要是按照 YOBO API 的规范, 获取一个访问 token, 然后再使用这个访问 token 去授权页面, 获取一个被验证了的访问 token, 最后再使用这个被验证的访问 token 换取一个 access token, 之后将使用这个 access token 访问用户的私有数据。而这里这个 `getToken()` 函数就是获取一个访问 token, 其主要涉及 YOBO API 的签名实现和数据的发送和接收, 以及对返回结果的解析, 代码如下所示:

```
// 获取访问码
private String getToken() throws NoSuchAlgorithmException {
    String token = "";
    try {
        // 1. 构造需要的数据
        long timestamp = System.currentTimeMillis();
        String nonce = Tools.md5(String.valueOf(System.currentTimeMillis())
        + PreferencesUtil.api_secret);
        Hashtable<String, String> params = new Hashtable<String, String>();
        params.put("api_key", PreferencesUtil.api_key);
        params.put("nonce", nonce);
        params.put("timestamp", String.valueOf(timestamp));

        // 2. 对参数进行签名
        String api_sig = Tools.md5(Tools.sig_params(params)
        + PreferencesUtil.api_secret);
        // 3. 添加 api_sig
        // params.put("api_sig", api_sig);
        // 4. 转换参数格式
        String params_str = Tools.paramsToString(params) + "api_sig="
        + api_sig;
        // 5. 提交数据, 获得 RequestToken 反馈
        String url_get = PreferencesUtil.baseUrl + PreferencesUtil.tokenUrl;
        Log.i("url_get", url_get);
        String result = Tools.sendGetRequest(url_get, params_str);
        // 6. 对反馈的 XML 进行解析, 得到 requestToken 的值
        token = Tools.paxToken(result);
        // 7. 把这个值存起来
        SharedPreferences settings = getSharedPreferences(
        PreferencesUtil.preferencesSetting, 0);
    }
}
```

```

settings.edit().putString(PreferencesUtil.token, token).commit();
// // 8.构造用户验证的 URL
// url = PreferencesUtil.baseUrl + PreferencesUtil.authorizeUrl
// + "?token=" + token;

} catch (NoSuchAlgorithmException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

return token;
}

```

代码解释:

这里代码还比较清晰, 首先是构造需要的参数, 这里需要一个 timestamp, nonce 以及 api_key, 然后按 YOBO API 的规范调用 Tools.sig_params (params) 进行参数排序, 代码如下所示:

```

// 对需要传递的参数进行排序
public static String sig_params(Hashtable<String, ?> params) {
Vector<String> v = new Vector<String>(params.keySet());
Collections.sort(v);
// Display (sorted) hashtable.
String result = "";
for (Enumeration<String> e = v.elements(); e.hasMoreElements();) {
String key = (String) e.nextElement();
String val = (String) params.get(key);
result += key;
result += val;
Log.i("sig_params", "Key: " + key + ",Val: " + val);
}
Log.i("sig_params", "result:" + result);
return result;
}

```

代码解释:

对参数按照 key 的字母正序排列, 以<key1><value1><key2><value2>.....这样的方式, 并返回排序后的字符串, 紧接着调用 Tools.md5 (Tools.sig_params (params) + PreferencesUtil.api_secret) 生成签名, 代码如下所示:

```

// MD5 加密
public static String md5(String s) throws NoSuchAlgorithmException {
System.out.println(System.currentTimeMillis());
MessageDigest m = MessageDigest.getInstance("MD5");
m.update(s.getBytes(), 0, s.length());
String md5_str = new BigInteger(1, m.digest()).toString(16);
Log.i("MD5", md5_str);
if (md5_str.length() < 32) {
md5_str = "0".concat(md5_str);
}

return md5_str;
}

```

代码解释:

使用系统提供的方法, 不再赘述。接着调用 Tools.paramsToString()生成需要的参数, 代码如下所示:

```

public static String paramsToString(Hashtable<String, ?> params) {
Vector<String> v = new Vector<String>(params.keySet());
Collections.sort(v);

```

```
// Display (sorted) hashtable.
String result = "";
for (Enumeration<String> e = v.elements(); e.hasMoreElements();) {
    String key = (String) e.nextElement();
    String val = (String) params.get(key);
    result += key;
    result += "=";
    result += val;
    result += "&";
    Log.i("sig_params", "Key: " + key + ",Val: " + val);
}
Log.i("sig_params", "result:" + result);
return result;
}
```

代码解释:

遍历参数,以“&”连接起来,生成需要的参数,最后使用 `Tools.sendGetRequest(url_get, params_str)` 将如上参数发送到 `url_get` 地址上,其地址就是 YOBO API 提供的获取访问 token 的地址,发送 HTTP 请求和接收返回数据的方法前面已经说过了,这里不再赘述,其返回的内容存储在 `result` 变量中,我们只需要分析这个返回数据,得到我们需要的访问 token 即可,其解析方法代码如下所示:

```
// 解析 RequestToken 返回的 XML, 得到 Token 的值
public static String paxToken(String xml) {
    String regEx = "<token>(.*?)</token>";
    Pattern p = Pattern.compile(regEx);
    Matcher m = p.matcher(xml);
    boolean rs = m.find();
    if (rs) {
        System.out.println("开始位置是: " + m.start() + "结束位置是: " + m.end()
            + "字符串是: " + m.group());
    }
    return m.group(1);
}
```

代码解释:

上述代码用来解析接口返回的 XML 格式的数据,得到需要的 Token 的值,对于这种比较简单的 XML,使用正则匹配是非常方便的。

最后将得到的访问 token 存储起来,后面需要用到,这里直接存储在 `SharedPreferences` 中,代码如下所示:

```
// 7.把这个值存起来
SharedPreferences settings = getSharedPreferences(
    PreferencesUtil.preferencesSetting, 0);
settings.edit().putString(PreferencesUtil.token, token).commit();
```

代码解释:

这里首先获取 `SharedPreferences` 对象,进而将获取到的 token 存储起来。

以上步骤完成后,我们就取得了访问 token,接着将这个 token 附着到 YOBO API 提供的授权页面上请求用户授权,也就是前面看到的那个 `openWebBrowser()` 方法,其效果如图 15-7 所示。

如图 15-7 上显示的就是用户授权的页面,当用户单击“允许”按钮后,即授权该应用访问其在 YOBO 上的个人数据。

如果用户尚未有 YOBO 的账号,则其在“未授权的提示页面”上选择注册的时候,其逻辑是一样的,只是最后会打开注册页面,注册完成后会自动调转到上面的授权操作页面,其注册页面的

效果如图 15-8 所示。



▲ 图 15-7 用户授权



▲ 图 15-8 用户注册界面

如图 15-7 上授权操作完成后，关闭浏览器，我们还需要使用这个被验证了的访问 token 换取 access token，所以，我们还需要提供两个菜单“完成授权”和“重新授权”，其实现代码如下所示：

```
// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_DONE, 0, R.string.menu_done).setIcon(
        R.drawable.menu_start).setAlphabeticShortcut('S');
    menu.add(0, MENU_REDO, 0, R.string.menu_redo).setIcon(
        R.drawable.menu_helps).setAlphabeticShortcut('H');
}
```

🔗 代码解释：

上述代码添加两个菜单，其对应的处理逻辑代码如下所示：

```
Intent intent = new Intent();
switch (item.getItemId()) {
    case MENU_DONE:
        try {
            String session = getSessionInPerence();
            if (session == "") {
                session = getSession(getTokenInPerence());
            }
            updateUserInfos(session);
        } catch (GeneralSecurityException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return true;
    case MENU_REDO:
        openWebBrowser(Tools.getAuthorizeUrl(getTokenInPerence()));
}
```

🔗 代码解释：

上述代码中，如果选择了“完成授权”菜单，对应的会执行 `getSession(getTokenInPerence())`，使用前面获取并存储的访问 token 换取 access token，其 `getTokenInPerence()` 代码如下所示：

```
// 判断 SharedPreferences 中是否已经存储了 access token
private String getTokenInPerence() {
```



```

Log.i(TAG, "getTokenInPerence..");
SharedPreferences settings = getSharedPreferences(
    PreferencesUtil.preferencesSetting, 0);
String token = settings.getString(PreferencesUtil.token, "");
if (token.equals(""))
    return "";
else
    return token;
}

```

代码解释:

这个方法用来获取前面存储在 SharedPreferences 中的 token 的值, 然后使用 getSession() 去 YOBO API 服务器换取 access token 的值, 其代码如下所示:

```

// 获取用户鉴权后的访问码: session
public String getSession(String token) throws GeneralSecurityException {
    String session = "";
    // 1.构造需要的数据
    long timestamp = System.currentTimeMillis();
    String nonce = Tools.md5(String.valueOf(System.currentTimeMillis())
        + PreferencesUtil.api_secret);
    Hashtable<String, String> params = new Hashtable<String, String>();
    params.put("api_key", PreferencesUtil.api_key);
    params.put("nonce", nonce);
    params.put("timestamp", String.valueOf(timestamp));
    params.put("token", token);
    // 2.对参数进行签名
    String api_sig = Tools.md5(Tools.sig_params(params)
        + PreferencesUtil.api_secret);
    // 4.转换参数格式
    String params_str = Tools.paramsToString(params) + "api_sig=" + api_sig;
    // 5.提交数据, 获得 RequestToken 反馈
    String url_get = PreferencesUtil.baseUrl + PreferencesUtil.sessionUrl;
    Log.i("url_get", url_get);
    String result = Tools.sendGetRequest(url_get, params_str);
    // 6.对反馈的 XML 进行解析, 得到 requestToken 的值
    session = Tools.paxSession(result);
    Log.i(TAG, "session=" + session);
    // 7.把这个值存起来
    SharedPreferences settings = getSharedPreferences(
        PreferencesUtil.preferencesSetting, 0);
    settings.edit().putString(PreferencesUtil.session, session).commit();
    return session;
}

```

代码解释:

其主要逻辑和前面获取访问 token 的时候是差不多的, 只是这里多传递了一个参数, 也就是前面获取的访问 token 的值, 最后是使用 Http 发送请求到其置换 access token 的地址上获取 access token, 将其返回的数据使用 Tools.paxSession 解析一下, 代码如下所示:

```

// 解析获取 session 的返回数据
public static String paxSession(String xml) {
    String regEx = "<name>(.*?)</name>.*?<key>(.*?)</key>";
    Pattern p = Pattern.compile(regEx);
    Matcher m = p.matcher(xml);
    boolean rs = m.find();
    if (rs) {
        System.out.println("开始位置是: " + m.start() + "结束位置是: " + m.end()
            + "字符串是: " + m.group());
    }
}

```

```

    }
    return m.group(2);
}

```

代码解释:

这里的代码用来解析接口返回的数据, 从中提取需要的 token 值, 同样, 这里采用的还是正则匹配的方式。

最后还是一样, 将这个 session 存储在 SharedPreferences 中备用就完成了整个的授权过程。紧接着, 就是调用 updateUserInfos (session) 语句来获取该用户的个人信息了, 我们下节继续讲述。

4. 用户个人信息展示界面

前面完成用户的授权后, 就可以接着来读取用户的个人信息了。按照我们前面的处理逻辑, 获取 session 后存储在 SharedPreferences 中, 以后只需要检查 SharedPreferences 是否存在这个 session 即可了, 如果存在, 则附带着这个 session 读取用户的私人数据, 这里的个人信息就是一个例子, 我们这里主要看 updateUserInfos (session) 语句, 其代码如下所示:

```

// 读取用户的个人信息
public void updateUserInfos(String session) {
    String url_get = PreferencesUtil.baseUrl + PreferencesUtil.user_info;
    // Tools.parUinfo(url_get+"?id=825856");
    String result = Tools.sendGetRequest(url_get, "id=825856");
    Log.i(TAG, "result:" + result);
    // Tools.parUinfo(result);
    DocumentBuilderFactory domfac = DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder dombuilder = domfac.newDocumentBuilder();
        InputStream is = new ByteArrayInputStream(result.getBytes());
        Document doc = dombuilder.parse(is);
        Element root = doc.getDocumentElement();
        System.out.println(root.getAttributes().getNamedItem("status")
            .getNodeValue());
        NodeList users = root.getChildNodes();
        if (users != null) {
            SharedPreferences settings = getSharedPreferences(
                PreferencesUtil.preferencesSetting, 0);
            Editor editor = settings.edit();
            for (int i = 0; i < users.getLength(); i++) {
                Node book = users.item(i);
                if (book.getNodeType() == Node.ELEMENT_NODE) {
                    for (Node node = book.getFirstChild(); node != null; node = node
                        .getNextSibling()) {
                        if (node.getNodeType() == Node.ELEMENT_NODE) {
                            String value = node.getFirstChild()
                                .getNodeValue();
                            System.out.println(node.getNodeName() + ":"
                                + value);
                            if (node.getNodeName().trim().equalsIgnoreCase(
                                "nickname")) {
                                editor.putString(PreferencesUtil.nickname,
                                    value);
                            }
                            if (node.getNodeName().equalsIgnoreCase(
                                "loved_singers")) {
                                editor.putString(
                                    PreferencesUtil.loved_singers,
                                    value);
                            }
                        }
                    }
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

    }
    if (node.getNodeName().equalsIgnoreCase(
        "created_date")) {
        editor
            .putString(
                PreferencesUtil.created_date,
                value);
    }
    if (node.getNodeName()
        .equalsIgnoreCase("image")) {
        editor.putString(PreferencesUtil.image,
            value);
    }
    if (node.getNodeName().equalsIgnoreCase(
        "homepage")) {
        editor.putString(PreferencesUtil.homepage,
            value);
    }
}
}
}
}
editor.commit();
restorePrefs();
}
} catch (ParserConfigurationException e) {
    e.printStackTrace();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (SAXException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

❏ 代码解释：

上述代码看上去挺复杂的，其实不然，其数据获取的主要代码为：

```
String url_get = PreferencesUtil.baseUrl + PreferencesUtil.user_info;
```

String result = Tools.sendGetRequest(url_get)语句。按照其 API 的规定，其返回的数据格式如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<YOBO status="ok">
    <user>
        <id>825856</id>
        <nickname>IceskYsl</nickname>
        <url>http://www.YOBO.com/user/profile/825856</url>
        <loved_singers>beyond</loved_singers>
        <homepage>http://iceskysl.lsters.com/</homepage>

        <gender>未知</gender>
        <created_date>20071002</created_date>
        <image
size="small">http://images.YOBO.com/userimg/1122/0337278fdd9b375c858c97e7b4a8389d.50.
gif</image>
        <image
size="medium">http://images.YOBO.com/userimg/1122/0337278fdd9b375c858c97e7b4a8389d.10
0.gif</image>
        <image
size="larger">http://images.YOBO.com/userimg/1122/0337278fdd9b375c858c97e7b4a8389d.15
0.gif</image>
    </user>
</YOBO>

```

```
</user>
```

```
</YOBO>
```

接下去的代码就是解析该 XML，提取需要的数据即可，这里由于需要解析的 XML 数据不大，我们采用 DOM 方式解析，解析后将需要的数据保存在 SharedPreferences 中（因为这些数据是不怎么变化的，为了方便下次展示，将其保存起来），这里需要的数据有用户的昵称、主页地址、喜欢的歌手以及头像信息，解析后就可以展示了。

另外，在这个页面上，需要添加其他的一些菜单，当用户授权成功，获取用户信息后，需要提供用户展示自己的音乐盒和音乐电台的界面，所以，我们需要添加相关的菜单，代码如下所示：

```
menu.add(0, MENU_BOX, 0, R.string.menu_box).setIcon(
    R.drawable.music_box).setAlphabeticShortcut('B');
menu.add(0, MENU_RADIO, 0, R.string.menu_radio).setIcon(
    R.drawable.music_radio).setAlphabeticShortcut('R');
menu.add(0, MENU_REF, 0, R.string.menu_ref).setIcon(
    R.drawable.refresh).setAlphabeticShortcut('R');
```

❏ 代码解释：

上述代码实现了添加两个菜单。每个菜单对应的响应逻辑代码如下所示：

```
case MENU_REDO:
    openWebBrowser(Tools.getAuthorizeUrl(getTokenInPerence()));
    break;
case MENU_BOX:
    //我的音乐盒列表
    intent.setClass(Auth.this, BoxActivty.class);
    startActivity(intent);
    return true;
case MENU_RADIO:
    //我的电台列表
    intent.setClass(Auth.this, StationActivty.class);
    startActivity(intent);
    break;
```

❏ 代码解释：

在上述代码中，选择“我的音乐盒”将打开用户的音乐盒列表界面；用户选择“我的电台”将显示用户的电台界面；另外一个“刷新我的资料”用来刷新用户的个人信息。其整个效果如图 15-9 所示。

我们接下来要实现个人音乐盒和个人电台的获取和展示。



▲ 图 15-9 个人展示界面

5. 我的音乐盒界面

在前面完成了用户信息的获取和展示，在其展示页上，如果用户选择“我的音乐盒”菜单，也就是执行 `intent.setClass(Auth.this, BoxActivty.class)` 这段代码，其将打开一个新的界面，显示用户在 YOBO 上的个人音乐盒。按照前面的 UI 规划，我们需要在 BoxActivty 中读取用户的音乐盒列表信息，按照 YOBO API 提供的接口，实现方式如下。首先看其 OnCreate 函数实现，其代码如下所示：

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.box);
    setTitle("YOBO: 我的音乐盒");
```

```

session = getSessionInPerence();
user = new User(session);
ListView itemlist = (ListView) findViewById(R.id.itemlist);
ArrayAdapter<Box> adapter = new ArrayAdapter<Box>(this,
android.R.layout.simple_list_item_1, user.getAllBoxes());

itemlist.setAdapter(adapter);

itemlist.setOnItemClickListener(this);

itemlist.setSelection(0);
}

```

代码解释：

如上代码中，首先使用 `getSessionInPerence()` 获取前面保存的 session key，然后调用 `new User (session)` 初始化一个 User 实例变量 `user`，最后使用了 Android 提供的列表功能将其显示出来。这里比较重要的是 User (session) 的实现，其代码在 User.java 中，看一下其声明，代码如下所示：

```

public class User {
public final String TAG = "User";
private String _nickname = null;
private String _created_date = null;
private String _homepage = null;
private String _love_sigers = null;
private int _boxes_count = 0;
private int _stations_count = 0;
private List<Box> _boxes_list;
private List<Station> _stations_list;

```

代码解释：

从代码中可以看出这个 User 的结构，其他代码都是一些 set、get 的实现，比较多，这里就不全部贴出来了。这里还用到一个类 Box，其是对音乐盒的抽象，其声明代码如下所示：

```

public class Box {
private String _id = null;
private String _name = null;
private String _created_date = null;
private int _itemcount = 0;
private List<Song> _itemlist;

public Box() {
_itemlist = new Vector<Song>(0);
}

```

代码解释：

音乐盒对象的结构比较清楚，有 ID、名字、创建时间、歌曲数以及一个存放歌曲的列表。

紧接着上面说的获取用户音乐盒的逻辑，主要还是按照 API 的规则构造参数后，请求获取用户音乐盒的 API 接口，然后将返回的数据使用 DOM 解析出来，按照 API 使用说明编程，其返回的数据格式为：

```

<?xml version="1.0" encoding="UTF-8"?>
<YOB status="ok">
  <boxes>
    <box>
      <id>569144</id>
      <name>随机收藏</name>
      <songs_count>100</songs_count>
      <created_date>20081129</created_date>
    </box>

```



```

<box>
<id>559132</id>
<name>生如夏花</name>
<songs_count>14</songs_count>
<created_date>20081030</created_date>
</box>

<box>
<id>553517</id>
<name>魔杰座</name>
<songs_count>11</songs_count>
<created_date>20081015</created_date>
</box>
...

```

✎ 代码解释：

上述代码是 API 返回的数据格式，比较标准的 XML 格式，上面的这段 XML 代码对于有一些 Java 编程基础的读者来说应该非常简单了，使用 DOM 也很方便，代码在 User.java 中，感兴趣的自己找出来看一下，这里就不赘述了，其最后的运行效果如图 15-10 所示。

6. 我的音乐电台

前面一个小节讲到了如何实现用户在 YOBO 的个人音乐盒，本小节讲解如何实现显示个人电台，和前面实现的音乐盒类似，用户在 YOBO 上可以根据一个艺术家或者一首歌曲创建一个音乐电台，YOBO 通过自己的算法，将和种子歌曲或艺术家相近的歌曲推荐给用户，每个用户可以拥有多个音乐电台，在 YOBO 网站上显示效果如图 15-11 所示。



▲ 图 15-10 我的音乐盒



▲ 图 15-11 YOBO 个人电台展示

按照网站 YOBO 的 API 使用说明，其个人音乐电台的接口为。

- 我的电台。
- 说明：获取指定用户的电台信息。
- 方法名：user/stations。
- 提交方法：Get。
- 是否需要授权：是。

其返回的数据格式代码如下所示：

```
<YOBO status="ok">
  <stations>
    <station>
      <id>3282827</id>
      <name>蒲公英的约定</name>
      <created_time>2008-10-03 15:47:10 UTC</created_time>
    </station>
    <station>
      <id>3261194</id>
      <name>稻香</name>
      <created_time>2008-09-23 22:46:11 UTC</created_time>
    </station>
    <station>
      <id>3261148</id>
      <name>魔杰座</name>
      <created_time>2008-09-23 22:30:12 UTC</created_time>
    </station>
  </stations>
</YOBO>
```

 代码解释：

从上述代码可以看到，和前面的“我的音乐盒”返回的数据格式基本一致，其解析方法类似，可以非常方便地用 Java DOM 解析 XML，进而显示，运行效果如图 15-12 所示。

至此，音乐盒和音乐电台的列表都实现了，下面讲解另外一个比较关键的环节，读取每个音乐盒或音乐电台的歌曲列表。

7. 音乐盒/电台歌曲列表

前面编程已经将“我的音乐盒”的列表读取出来了，本小节继续进行另外一个任务讲解，获取指定音乐盒的歌曲列表。我们从 YOBO API 应用说明可以看到，其播放列表遵循 XSPF 规范（关于 XSPF 规范的详细说明，可以参考其官方说明，网址为 <http://www.xspf.org/>），我们只需要解析返回的 XSPF 格式的 XML 文件，生成播放列表即可。

先熟悉下 XSPF 的格式，代码如下所示：

```
<YOBO status="ok">
  -
  <playlist version="1">
    <title>随机好歌</title>
    <annotation>随机好歌</annotation>
    <creator/>
    <date>2008-08-23T11:16:01</date>
    <trackList>
      <track>
        <location>
          http://218.11.132.51/band/groovecoverage/01o/3i.mp3
        </location>
        <identifier>http://www.YOBO.com/song/index/8319</identifier>
        <title>god is a girl (album version)</title>
        <artist>Groove Coverage</artist>
        <album>Best of</album>
        <creator>Groove Coverage</creator>
        <duration>182000</duration>
      </track>
    </trackList>
  </playlist>
</YOBO>
```



▲ 图 15-12 音乐电台

```

<info>http://www.YOBO.com/song/index/8319</info>
<image>http://www.YOBO.com/artist/index/2304</image>
<extension application="http://www.YOBO.com">
<artistpage>http://www.YOBO.com/artist/index/2304</artistpage>
<albumpage>http://www.YOBO.com/album/index/14079</albumpage>
<trackpage>http://www.YOBO.com/song/index/8319</trackpage>
</extension>
</track>

```

❏ 代码解释:

上面显示的就是一段比较标准的 XSPF 格式的内容, 也不是很复杂, 已经有很多开源的类库可以直接解析 XSPF 格式的文件, 但截止到本书编写时, 尚未看到有可以使用在 Android 上的包, 我们只能自己来实现了。

实现一个通用的解析 XSPF 格式的类库可能会稍微复杂些, 我们这里只针对 YOBO API 中提供的播放列表编写其解析方法。前面已经使用 Java DOM 解析 XML 文件, 这里使用 SAX 来解析 XSPF, 关于在解析 XML 时是选择 DOM 还是 SAX, 需要根据不同的使用背景来选择。关于使用 SAX 解析 XML 文档, 比较详细的介绍请参考本书的第 12 章, 这里就不赘述了, 先生成一个继承自 DefaultHandler 的解析器 (Parser), 其中包含几个比较重要的函数构造方法, 实现代码如下所示:

```

public void parse() throws Exception {
    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser sp = spf.newSAXParser();
    XMLReader xr = sp.getXMLReader();
    xr.setContentHandler(this);
    InputStream is = null;
    URL url = new URL(playlistUrl);
    URLConnection uc = url.openConnection();
    is = uc.getInputStream();
    playlist = new Playlist();
    ready = false;
    xr.parse(new InputSource(is));
}

```

❏ 代码解释:

这段代码从指定的 URL 获取响应内容, 并调用 XMLReader 对象的 parse 解析。

另外 startElement、endElement、characters 这几个方法就不贴出来了, 其分别对应 XML 中元素开始、结束和元素内容的时候被调用, 可以参考/YOBO.Android/src/us/imnet/iceskysl/xspf 下的 Parser.java。

还需要新建一个 PlaylistActivity 来显示歌曲列表, 它首先需要接收前面一个 Activity 传入的参数, 实现方法如下:

```

// 接收传递进来的播放列表信息
private void revParams() {
    Intent startingIntent = getIntent();
    if (startingIntent != null) {
        Bundle playlist = startingIntent
            .getBundleExtra("android.intent.extra.playlist");
        if (playlist == null) {
            is_valid = false;
        } else {
            _id = playlist.getString("id");
            _name = playlist.getString("name");
            _type = playlist.getString("type");
            is_valid = true;
        }
    } else {
        is_valid = false;
    }
}

```

```

    }
}

```

代码解释:

其传递进来 3 个参数, 由于 **type** 指定是音乐盒还是音乐电台, 然后需要按照其 **type** 去获取音乐盒或者音乐电台的歌曲列表, 实现代码如下所示:

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.box);
    setTitle("YOBO: " + _name);
    revParams();
    this.bindService(new Intent(PlaylistActivity.this,
        YPRSService.class), mConnection, Context.BIND_AUTO_CREATE);
    String url = build_url(id, type);
    Parser parser = new Parser(url);
    try {
        parser.parse();
        playlist = parser.getPlaylist();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

代码解释:

parser.parse() 就是调用前面的 XML 解析器处理 XML 文件, 然后通过 **parser.getPlaylist()** 获取整个歌曲列表, 其实现效果如图 15-13 所示。

按照 YOBO API 的接口说明, 其每个电台也对应一个播放列表, 其实现方法和前面的音乐盒基本上一致, 其返回的也是 XSRF 格式的 XML 文件, 解析方法是一样的, 这里就不赘述了。下面还有更关键的事情要做, 实现音乐播放。



▲ 图 15-13 播放列表

15.3.4 实现 Service

前面已经将需要播放的歌曲列出来了, 接着需要实现对该歌曲列表的音乐播放功能了, 在 Android 中播放音频文件是非常方便的, 但是这里要更专业一些, 使用 Service 在后台播放歌曲, 使得前面的操作动作不会影响歌曲的播放。在本书的第 10 章已经介绍过 Service 的概念和使用方法, 这里将以这个实际的例子实现它的应用。

这里选择使用 AIDL(Android Interface Definition Language)来实现 Service 接口, 关于 AIDL 的介绍, 可以参考 <http://developer.android.com/guide/developing/tools/aidl.html>, 使用 AIDL 的一般步骤是编写 AIDL 文件, 实现接口中的方法, 下面将逐一讲解。

1. 编写 AIDL 文件

首先需要编写 AIDL 文件, 定义需要的接口, 这里用 Service 实现的播放器, 需要具有播放、暂停、下一首、上一首和停止等几个最典型的接口, 所以, 依照分析实现其接口定义的 AIDL 文件 (**YPRSInterface.aidl**) 代码如下所示:

```

package us.imnet.iceskysl;
interface YPRSInterface {

```

```

void clearPlaylist();
void addSongPlaylist( in String song );
void playFile( in int position );
void pause();
void stop();
void skipForward();
void skipBack();
}

```

代码解释:

如上代码定义了需要的几个接口方法, 分别是清除播放列表, 添加歌曲到播放列表, 播放, 暂停和停止。

2. 实现接口方法

前面定义好需要的接口后, AIDL 生成器会自动生成 AIDL 文件对应的 YPRSInterface.java 文件, 其中的内容不需要修改, 需要做的是其 Service 实现 YPRService.java 文件, 其关键方法代码如下所示:

```

private final YPRSInterface.Stub mBinder = new YPRSInterface.Stub() {
//播放文件
public void playFile(int position) throws DeadObjectException {
Log.i("playFile:", Integer.toString(position));
try {
currentPosition = position;
playSong(songs.get(position));
} catch (IndexOutOfBoundsException e) {
Log.e(getString(R.string.app_name), e.getMessage());
}
}
//添加歌曲的播放列表
public void addSongPlaylist(String song) throws DeadObjectException {
songs.add(song);
}
//清除播放列表
public void clearPlaylist() throws DeadObjectException {
songs.clear();
}
//下一首
public void skipBack() throws DeadObjectException {
prevSong();
}
//上一首
public void skipForward() throws DeadObjectException {
nextSong();
}
//暂停
public void pause() throws DeadObjectException {
Notification notification = new Notification();
nm.notify(NOTIFY_ID, notification);
mpr.pause();
}
//停止
public void stop() throws DeadObjectException {
nm.cancel(NOTIFY_ID);
mpr.stop();
}
};

```

代码解释:

如上代码实现了需要的几个播放器的控制动作, 这个方法在 MediaPlayer (mpr) 对象中已经存

在，在此之间调用就可以了。

另外有几个关键方法代码如下所示：

```
private void playSong(String file) {
    Log.i("playSong:", file);
    try {

        Notification notification = new Notification();
        nm.notify(NOTIFY_ID, notification);
        Log.i("playSong:", "-----");
        mpr.reset();
        mpr.setDataSource(file);
        mpr.prepare();
        mpr.start();
        mpr.setOnCompletionListener(new OnCompletionListener() {
            public void onCompletion(MediaPlayer arg0) {
                nextSong();
            }
        });
    } catch (IOException e) {
        Log.e(getString(R.string.app_name), e.getMessage());
    }
}

//下一首
private void nextSong() {
    // Check if last song or not
    if (++currentPosition >= songs.size()) {
        currentPosition = 0;
        nm.cancel(NOTIFY_ID);
    } else {
        playSong(songs.get(currentPosition));
    }
}

//上一首
private void prevSong() {
    if (mpr.getCurrentPosition() < 3000 && currentPosition >= 1) {
        playSong(songs.get(--currentPosition));
    } else {
        playSong(songs.get(currentPosition));
    }
}
```

 代码解释：

上述代码实现了播放器的主要功能，其他方法请查看/YOBO.Android/src/us/imnet/iceskysl/Service/YPRService.java 文件。

3. 把功能串起来

前面将主要的功能都实现了，然后将这个功能串起来形成一个完整的应用，主要就是实现播放器的功能，在 `PlayListActivity` 的 `onCreate` 方法中需要绑定到前面实现的 `Service`，其实现代码如下所示：

```
this.bindService(new Intent(PlayListActivity.this,
    PRService.class), mConnection, Context.BIND_AUTO_CREATE);
```

 代码解释：

在启动的时候，将 `Activity` 通过 `ServiceConnection` 与 `Service` 绑定起来。

其中 `mConnection` 的定义代码如下所示：

```
private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder Service) {
```

```

mpInterface = YPRSInterface.Stub.asInterface((IBinder) Service);
updateSongList();
}
public void onServiceDisconnected(ComponentName className) {
mpInterface = null;
}
};

```

代码解释：

如上代码实现了在绑定或者取消绑定的时候需要做的事情，其中在绑定的时候，需要将前面的播放列表加载到里面去，也就是上面代码中 `updateSongList` 实现的，其实现代码如下所示：

```

// 更新歌曲列表
public void updateSongList() {
try {
mpInterface.clearPlaylist();
if (playlist != null) {
for (Track track : playlist.getTracks()) {
songs.add(track.getTitle());
mpInterface.addSongPlaylist(track.getLocation());
Log.i("SONGS:", track.getLocation());
}
itemlist = (ListView) findViewById(R.id.itemlist);
ArrayAdapter<Track> adapter = new ArrayAdapter<Track>(this,
android.R.layout.simple_list_item_1, playlist
.getTracks());
itemlist.setAdapter(adapter);
itemlist.setOnItemClickListener(this);
itemlist.setSelection(0);
}
} catch (RemoteException e) {
Log.e(getString(R.string.app_name), e.getMessage());
}
}

```

代码解释：

上述代码比较简单，主要就是将获取的歌曲列表添加到 `YPRSInterface` 中去，在此不再赘述。

然后需要实现的功能是单击列表中的歌曲，实现歌曲的播放，其实现代码如下所示：

```

@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int position,
long arg3) {
Log.i("onListItemClick:", Integer.toString(position));
try {
player_position = position;
mpInterface.playFile(position);
itemlist.setSelector(R.color.translucent_red);
} catch (RemoteException e) {
Log.e(getString(R.string.app_name), e.getMessage());
}
}

```

代码解释：

也就是取得被单击选中的条目，然后将其歌曲文件地址传递给播放器，实现播放。

最后还需要添加多播放的控制，例如，实现“下一首”、“上一首”、“停止”等功能，选择使用菜单实现，其菜单生成和响应的代码如下所示：

```

// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {

```

```

super.onCreateOptionsMenu(menu);
menu.add(0, PLAY_NEXT, 0, R.string.player_next).setIcon(
    R.drawable.player_fwd).setAlphabeticShortcut('N');
menu.add(0, PLAY_PREV, 0, R.string.player_prev).setIcon(
    R.drawable.player_rew).setAlphabeticShortcut('P');
menu.add(0, PLAY_STOP, 0, R.string.player_stop).setIcon(
    R.drawable.player_stop).setAlphabeticShortcut('S');
return true;
}

// 当一个菜单被选中的时候调用
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case PLAY_NEXT:
            try {
                mpInterface.skipForward();
                itemlist.setSelector(R.color.translucent_red);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
            return true;
        case PLAY_PREV:
            try {
                mpInterface.skipBack();
                itemlist.setSelector(R.color.translucent_red);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        case PLAY_STOP:
            try {
                mpInterface.stop();
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        return true;
    }
}

```

代码解释:

程序实现的是当菜单被选中后, 分别调用前面定义和实现的控制接口。

如上步骤完成以后, 就可以实现歌曲的播放控制了, 运行效果如

图 15-14 所示。



▲ 图 15-14 播放器

15.3.5 后续开发展望

在前面利用 YOBO API 实现了在线音乐播放器, 具备了授权、获取个人信息、获取个人音乐盒、获取个人的音乐电台, 以及音乐盒中歌曲的播放和控制, 已经是一个比较完善的应用。但是按照 YOBO API 的介绍, 我们还可以做更多的事情, 例如, 获取指定歌手的信息、歌手的专辑列表、歌手的全部歌曲; 专辑的信息、专辑中的歌曲; 好友列表、站内短消息等一系列的功能, 这些功能也希望有兴趣的朋友可以在此基础上做出更好的完善。

15.4 在线音乐播放器应用演示

音乐播放器应用编写完后, 还需要在真实的设备上测试一遍流程, 看是否程序有问题或者需要

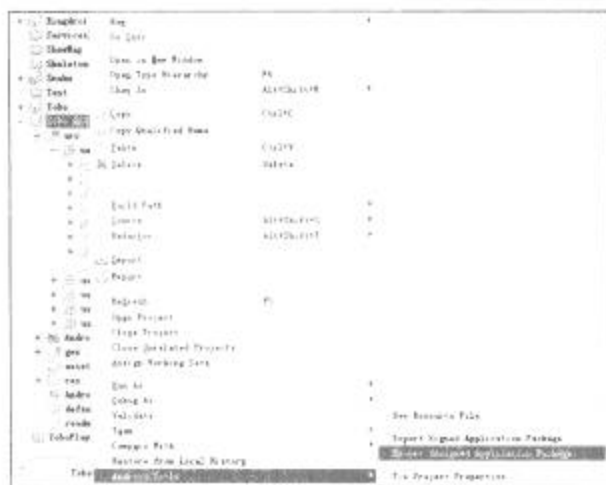
改进的地方，以便通过测试再进一步完善。值得称赞的是 Android 的模拟器，其模拟效果非常好。这里在真实设备上演示的效果和前面看到的截图是一样的，就不重复贴相应的演示图了。

15.5 在线音乐播放器应用打包、签名和发布

编写完成应用后，就可以对程序进行打包、签名和发布啦，其方法在本书第 4 章已经比较详细地介绍过了，这里就不赘述，只将操作流程演示一下，将项目导出未签名的 apk 文件包，如图 15-15 所示。

对导出的 apk 文件进行签名，如图 15-16 所示。

签名完成后，就可以上传到 Android Market 上发布了，如图 15-17 所示。



▲ 图 15-15 打包



▲ 图 15-16 签名



▲ 图 15-17 发布

15.6 本章小结

本章完整地实现了一个基于 YOBO API 的在线播放器，其涉及在 Android 上读取其他 Web Service 的数据，解析生成自己需要的内容，实现了一款在线播放器的主要功能。通过本章学习，读者应该学会了基于 API 的客户端开发过程，也帮助读者对多媒体应用的编写积累一定的经验，相信通过本章学习，就可以顺利地进入多媒体应用的开发中了。



第 16 章 Android 综合案例五——手机 信息查看助手

学习目标

- 掌握获取 CPU、内存、存储等手机硬件信息；
- 掌握获取安装的应用信息；
- 掌握获取运行的 Service、Task 和进程信息。

16.1 手机信息查看助手功能规划和可行性分析

本章的目标是编写一款查看手机系统信息的应用工具，其查看的是手机的基本信息，例如，硬件配置，软件系统，以及一些运行时的信息。通过应用的实现，将着重讲解如何通过编程的方式查看系统信息。开始之前，给这个应用起个名字为 `eoelInfosAssistant`，通过前面章节的学习，和以前重复的技术细节不再赘述。

16.1.1 手机信息查看助手功能规划

本章应用预设的目标为这个应用实现以下功能，查看设备的系统信息、硬件信息、安装的软件信息、已经运行的软件信息等。那么，可以先把上面说的功能分成这 4 个大类来组织。每个大类中需要查看的信息个数都是不一样的，例如，在查看系统信息时，会查看系统版本，系统信息及运营商信息等；而在硬件信息查看时，比较关注的是 CPU、内存、硬盘、网络信息；而在软件信息查看中，关注的是已经安装的软件应用；运行时信息中比较关注的是正在运行的后台服务、任务以及进程。按照这个功能的规划，可以大概地在纸上画出其对应的功能描述，如图 16-1 所示。

16.1.2 手机信息查看助手可行性分析

在开始进入编写程序前，需要对需求的功能做一些



▲ 图 16-1 功能规划

可行性分析，以做到有的放矢，如果发现有些无法实现的功能，要尽快调整。

这里分析一下项目需要的功能，主要是信息查看和信息收集，如版本信息、硬件信息等，这些都可以通过读取系统文件或者运行系统命令获取；而像获取安装的软件信息和运行时信息则需要通过 API 提供的接口获取。依照前面章节的学习，实现 API 接口不是什么问题。那么，需要把精力集中在如何实现运行系统命令，获取其返回的结果功能实现上。如果读者熟悉 Java 编程，那么，自己应该已经积累了不少的开发经验，下面一段代码实现这个功能，具体实现代码如下所示：

```
public class CMDExecute {

    public synchronized String run(String[] cmd, String workdirectory)
        throws IOException {
        String result = "";

        try {
            ProcessBuilder builder = new ProcessBuilder(cmd);
            // 设置一个路径
            if (workdirectory != null)
                builder.directory(new File(workdirectory));
            builder.redirectErrorStream(true);
            Process process = builder.start();
            InputStream in = process.getInputStream();
            byte[] re = new byte[1024];
            while (in.read(re) != -1) {
                System.out.println(new String(re));
                result = result + new String(re);
            }
            in.close();

        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

}
```

 代码解释：

上述代码中，CMDExecute.java 是个比较独立的类，其可以实现运行指定命令并返回结果，其使用方法比较简单，在后面具体使用到的时候再给出相关代码及说明。

16.2 手机信息查看助手功能实现

按照前面的功能规划，我们将在本节逐一实现所需要的功能。

16.2.1 手机信息查看助手主界面

按照我们的规划，将 4 类信息的查看入口放在主界面上，其布局文件为 main.xml，基本上是用一个列表组件组成的，实现代码如下所示：

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

```

    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ListView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/itemlist" />
    </LinearLayout>

```

代码解释:

这里可以看到 main.xml 中使用的是 LinearLayout 布局，其中放置了一个 ListView 组件。

然后看其 eoelInfosAssistant.java 代码，这是整个程序的入口，也就是程序的主界面，它将 4 类信息查看入口以 List（列表）的形式展示出来，先看它的 onCreate 方法，实现代码如下所示：

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    itemlist = (ListView) findViewById(R.id.itemlist);
    refreshListItems();
}

```

代码解释:

上述代码比较清晰，其中调用自定义方法 refreshListItems() 来获取列表信息，继续看这个 refreshListItems() 方法，实现代码如下所示：

```

private void refreshListItems() {
    list = buildListForSimpleAdapter();
    SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.item_row,
        new String[] { "name", "desc", "img" }, new int[] { R.id.name,
            R.id.desc, R.id.img });
    itemlist.setAdapter(notes);
    itemlist.setOnItemClickListener(this);
    itemlist.setSelection(0);
}

```

代码解释:

代码里面最开始引用 buildListForSimpleAdapter 方法填充相关数据，其实现代码如下所示：

```

private List<Map<String, Object>> buildListForSimpleAdapter() {
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(3);
    // 建立一个 HashMap
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("name", "系统信息");
    map.put("desc", "查看设备系统版本, 运营商及其系统信息.");
    map.put("img", R.drawable.system);
    list.add(map);

    map = new HashMap<String, Object>();
    map.put("name", "硬件信息");
    map.put("desc", "查看包括 CPU、硬盘、内存等硬件信息.");
    map.put("img", R.drawable.hardware);
    list.add(map);

    map = new HashMap<String, Object>();
    map.put("name", "软件信息");
    map.put("desc", "查看已经安装的软件信息.");
    map.put("img", R.drawable.software);
    list.add(map);
}

```

```

map = new HashMap<String, Object>();
map.put("name", "运行时信息");
map.put("desc", "查看设备运行时的信息.");
map.put("img", R.drawable.running);
list.add(map);
map = new HashMap<String, Object>();
map.put("name", "文件浏览器");
map.put("desc", "浏览查看文件系统.");
map.put("img", R.drawable.file_explorer);
list.add(map);
return list;
}

```

代码解释:

这个 `buildListForSimpleAdapter()` 方法构建一个 `List` (列表), 将需要的大分类的名字和描述信息填充进去。

然后 `refreshListItems()` 方法引入 `SimpleAdapter` 的适配方式, 其使用一个单独的 `item_row.xml` 为模板文件, 设置好其对应的关系, 先看 `item_row.xml` 的实现, 代码如下所示:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vw1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="4px"
    android:orientation="horizontal">

    <ImageView android:id="@+id/img"
        android:layout_width="32px"
        android:layout_margin="4px"
        android:layout_height="32px"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView android:id="@+id/name"
            android:textSize="18sp"
            android:textStyle="bold"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>

        <TextView android:id="@+id/desc"
            android:textSize="14sp"
            android:layout_width="fill_parent"
            android:paddingLeft="20px"
            android:layout_height="wrap_content"/>

    </LinearLayout>

</LinearLayout>

```

代码解释:

`item_row.xml` 模板文件用来描述主界面每行的显示方式, 这里使用示意图、名字和简介来描述一行的内容, 其运行效果如图 16-2 所示。



▲ 图 16-2 主界面效果图

然后重写单击每行的响应方法 `onItemClick`，实现代码如下所示：

```
@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
    Intent intent = new Intent();
    Log.i(TAG, "item clicked! [" + position + "]");
    switch (position) {
        case 0:
            intent.setClass(eoeInfosAssistant.this, System.class);
            startActivity(intent);
            break;
        case 1:
            intent.setClass(eoeInfosAssistant.this, Hardware.class);
            startActivity(intent);
            break;
        case 2:
            intent.setClass(eoeInfosAssistant.this, Software.class);
            startActivity(intent);
            break;
        case 3:
            intent.setClass(eoeInfosAssistant.this, Runing.class);
            startActivity(intent);
            break;
    }
}
```

 代码解释：

上述代码实现单击每行的时候，打开新的界面，从上至下依次为系统信息、硬件信息、软件信息和运行时信息。以下逐一讲解实现过程。

16.2.2 查看系统信息实现

当在运行的主界面单击第一行时，也就是“系统信息”这一行，将执行如下代码：

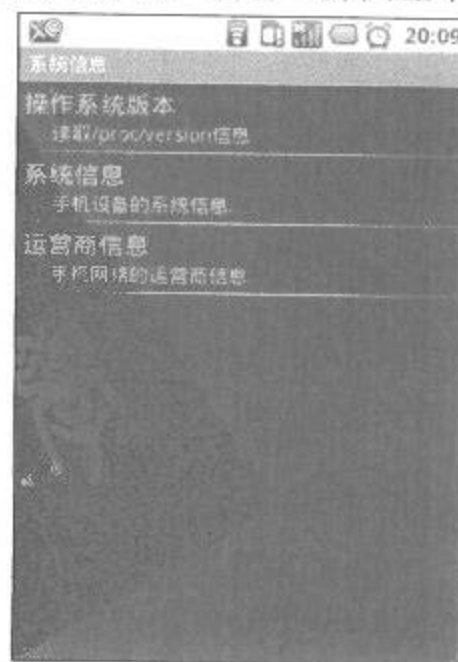
```
case 0:
    intent.setClass(eoeInfosAssistant.this, System.class);
    startActivity(intent);
    break;
```

代码运行后将显示系统（System）这个界面，这就是查看系统信息的主界面，其和主界面差不多，也就是列表显示几个需要查看的系统信息，界面运行效果如图 16-3 所示。

`System.java` 中列表填充代码如下：

```
private List<Map<String, Object>> buildListForSimpleAdapter() {
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(3);
    // 建立一个 Hash Map
    Map<String, Object> map = new HashMap<String, Object>();
    map = new HashMap<String, Object>();
    map.put("id", PreferencesUtil.VER_INFO);
    map.put("name", "操作系统版本");
    map.put("desc", "读取/proc/version 信息");
    list.add(map);

    map = new HashMap<String, Object>();
    map.put("id", PreferencesUtil.SystemProperty);
    map.put("name", "系统信息");
    map.put("desc", "手机设备的系统信息.");
}
```



▲ 图 16-3 系统信息主界面

```
// map.put("icon", R.drawable.mem);
list.add(map);

map = new HashMap<String, Object>();
map.put("id", PreferencesUtil.TEL_STATUS);
map.put("name", "运营商信息");
map.put("desc", "手机网络的运营商信息.");
list.add(map);

return list;
}
```

代码解释:

和前面介绍主界面时一样, 实现构造系统信息查看需要的功能。

当选择上述图片中的一行时, 将执行如下的代码:

```
@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
    Intent intent = new Intent();
    Log.i(TAG, "item clicked! [" + position + "]");
    Bundle info = new Bundle();
    Map<String, Object> map = list.get(position);
    info.putInt("id", (Integer) map.get("id"));
    info.putString("name", (String) map.get("name"));
    info.putString("desc", (String) map.get("desc"));
    info.putInt("position", position);
    intent.putExtra("android.intent.extra.info", info);
    intent.setClass(System.this, ShowInfo.class);
    startActivityForResult(intent, 0);
}
```

代码解释:

从上面代码可以看出, 这里将需要查看的信息及其相关参数以 **Bundle** 的方式传递到下一个活动, 这里传递了一个内部标识 ID, 需要查看的信息的名字。再看一下其显示详细信息的界面 **ShowInfo** 文件, 其主要是接收传递进来的参数, 获取相关的信息, 进而显示内容, 先看它接收参数的代码:

```
// 接收传递进来的信息
private void revParams() {
    Log.i(TAG, "revParams.");
    Intent startingIntent = getIntent();
    if (startingIntent != null) {
        Bundle infod = startingIntent
            .getBundleExtra("android.intent.extra.info");
        if (infod == null) {
            is_valid = false;
        } else {
            _id = infod.getInt("id");
            _name = infod.getString("name");
            _position = infod.getInt("position");
            is_valid = true;
        }
    } else {
        is_valid = false;
    }
    Log.i(TAG, "_name:" + _name + ", _id=" + _id);
}
```

代码解释:

上述代码接收传递进来的参数, 根据其传入的 ID 信息, 调用各自的方法获取需要查看的数据,

最后显示在界面上，这个界面比较简单，如下是其 XML 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <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="20px"
    >

    <TextView android:id="@+id/title"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:paddingBottom="8dip"
        android:text="" />
    <TextView android:id="@+id/info"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
    </LinearLayout>
</ScrollView>
```

 代码解释：

上述代码定制了一些显示样式，具体的使用方式和显示样式后面会看到。

1. 操作系统版本

单击图 16-3 所示的界面第一行“操作系统版本”项，则会打开一个新的界面，其对应的是 ShowInfo.java 文件，然后需要显示该设备的操作系统版本信息，而这个信息在 /proc/version 中有，可以直接调用。在可行性分析中给出的 CMDEXecute 类来调用系统的 cat 命令获取该文件的内容，实现代码如下所示：

```
public static String fetch_version_info() {
    String result = null;
    CMDEXecute cmdexe = new CMDEXecute();
    try {
        String[] args = { "/system/bin/cat", "/proc/version" };
        result = cmdexe.run(args, "/system/bin/");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return result;
}
```

 代码解释：

上述代码使用前面说到的 CMDEXecute 类，调用系统的“/system/bin/cat”工具，获取“/proc/version”中的内容。

其运行效果如图 16-4 所示。

从图 16-4 显示的查寻结果可以看到，这个设备的系统版本是 Linux version 2.6.25-018430-gfca26b0。



▲ 图 16-4 操作系统版本信息

2. 系统信息

继续实现“系统信息”的第二个功能，获取“系统信息”，在 Android 中获取系统信息可以调用其提供的方法 `System.getProperty (propertyStr)`，而系统信息诸如用户根目录（`user.home`）等都可以通过这个方法来获取，实现代码如下所示：

```
private static String initProperty(String description, String propertyStr) {
    if (buffer == null) {
        buffer = new StringBuffer();
    }
    buffer.append(description).append(":");
    buffer.append(System.getProperty(propertyStr)).append("\n");
    return buffer.toString();
}

public static String getSystemProperty() {
    buffer = new StringBuffer();
    initProperty("java.vendor.url", "java.vendor.url");
    initProperty("java.class.path", "java.class.path");
    initProperty("user.home", "user.home");
    initProperty("java.class.version", "java.class.version");
    initProperty("os.version", "os.version");
    initProperty("java.vendor", "java.vendor");
    initProperty("user.dir", "user.dir");
    initProperty("user.timezone", "user.timezone");
    initProperty("path.separator", "path.separator");
    initProperty("os.name", "os.name");
    initProperty("os.arch", "os.arch");
    initProperty("line.separator", "line.separator");
    initProperty("file.separator", "file.separator");
    initProperty("user.name", "user.name");
    initProperty("java.version", "java.version");
    initProperty("java.home", "java.home");
    return buffer.toString();
}
```

代码解释：

上述代码主要是通过调用系统提供的 `System.getProperty` 方法获取指定的系统信息，并合并成字符串返回。

其运行效果如图 16-5 所示。

3. 运营商信息

运营商信息中包含 IMEI、手机号码等，在 Android 中提供了运营商管理类（`TelephonyManager`），可以通过 `TelephonyManager` 来获取运营商相关的信息，实现的关键代码如下所示：

```
public static String fetch_tel_status(Context cx) {
    String result = null;
    TelephonyManager tm = (TelephonyManager) cx
        .getSystemService(Context.TELEPHONY_SERVICE);
    String str = "";
    str += "DeviceId(IMEI) = " + tm.getDeviceId() + "\n";
    str += "DeviceSoftwareVersion = " + tm.getDeviceSoftwareVersion()
```



▲ 图 16-5 系统信息界面

```

+ "\n";
str += "Line1Number = " + tm.getLine1Number() + "\n";
str += "NetworkCountryIso = " + tm.getNetworkCountryIso() + "\n";
str += "NetworkOperator = " + tm.getNetworkOperator() + "\n";
str += "NetworkOperatorName = " + tm.getNetworkOperatorName() + "\n";
str += "NetworkType = " + tm.getNetworkType() + "\n";
str += "PhoneType = " + tm.getPhoneType() + "\n";
str += "SimCountryIso = " + tm.getSimCountryIso() + "\n";
str += "SimOperator = " + tm.getSimOperator() + "\n";
str += "SimOperatorName = " + tm.getSimOperatorName() + "\n";
str += "SimSerialNumber = " + tm.getSimSerialNumber() + "\n";
str += "SimState = " + tm.getSimState() + "\n";
str += "SubscriberId(IMSI) = " + tm.getSubscriberId() + "\n";
str += "VoiceMailNumber = " + tm.getVoiceMailNumber() + "\n";

int mcc = cx.getResources().getConfiguration().mcc;
int mnc = cx.getResources().getConfiguration().mnc;
str += "IMSI MCC (Mobile Country Code) : " + String.valueOf(mcc) + "\n";
str += "IMSI MNC (Mobile Network Code) : " + String.valueOf(mnc) + "\n";
result = str;
return result;
}

```

代码解释:

上述代码中首先调用系统的 `getSystemService(Context.TELEPHONY_SERVICE)` 方法获取一个 `TelephonyManager` 对象 `tm`，进而调用其方法 `getDeviceId()` 获取 `DeviceId` 信息，调用 `getDeviceSoftwareVersion()` 获取设备的软件版本信息等。

其运行效果如图 16-6 所示。



▲ 图 16-6 运营商信息

16.2.3 查看硬件信息

前面讲解的是获取系统信息的相关方法，本节讲解如何获取手机设备的硬件信息，下面将逐一讲解获得的方法实现和运行的效果展示，硬件信息的主界面如图 16-7 所示。

1. 获取 CPU 信息

首先讲解如何获取手机设备的 CPU 信息。可以在手机设备的 `/proc/cpuinfo` 中获取 CPU 信息，和前面获取操作系统方法类似，调用 `CMDEXecute` 执行系统的 `cat` 的命令，读取 `/proc/cpuinfo` 的内容，显示的就是其 CPU 信息，实现代码如下所示：

```

public static String fetch_cpu_info() {
    String result = null;
    CMDEXecute cmdexe = new CMDEXecute();
    try {
        String[] args = { "/system/bin/cat", "/proc/cpuinfo" };
        result = cmdexe.run(args, "/system/bin/");
        Log.i("result", "result=" + result);
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return result;
}

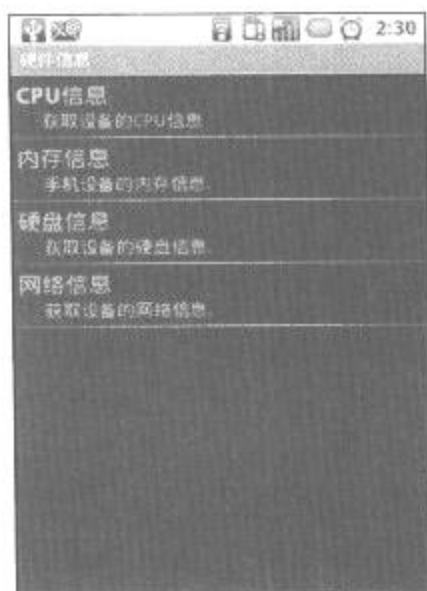
```

代码解释:

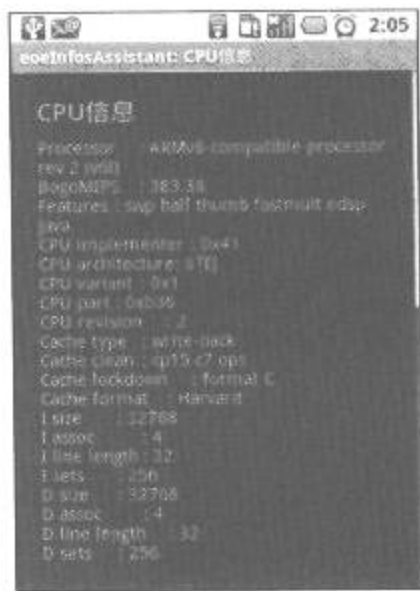
上述代码使用 CMDExecute, 调用系统中的 “/system/bin/cat” 命令查看 “/proc/cpuinfo” 中的内容, 即可得到 CPU 信息。

运行效果如图 16-8 所示。

如图 16-8 上显示的是笔者使用的 Android 手机 G1 的 CPU 信息。



▲ 图 16-7 硬件信息主界面



▲ 图 16-8 CPU 信息

2. 获取内存信息

获取内存信息的方法和获取 CPU 信息的实现差不多, 可以读取 /proc/meminfo 信息, 另外还可以通过 getSystemService(Context.ACTIVITY_SERVICE) 获取 ActivityManager.MemoryInfo 对象, 进而获取可用内存信息, 主要的代码如下所示:

```
/**
 * 系统内存情况查看
 */
public static String getMemoryInfo(Context context) {
    StringBuffer memoryInfo = new StringBuffer();
    final ActivityManager activityManager = (ActivityManager) context
        .getSystemService(Context.ACTIVITY_SERVICE);
    ActivityManager.MemoryInfo outInfo = new ActivityManager.MemoryInfo();
    activityManager.getMemoryInfo(outInfo);
    memoryInfo.append("\nTotal Available Memory :").append(
        outInfo.availMem >> 10).append("k");
    memoryInfo.append("\nTotal Available Memory :").append(
        outInfo.availMem >> 20).append("M");
    memoryInfo.append("\nIn low memory situation:").append(
        outInfo.lowMemory);
    String result = null;
    CMDExecute cmdexe = new CMDExecute();
    try {
        String[] args = { "/system/bin/cat", "/proc/meminfo" };
        result = cmdexe.run(args, "/system/bin/");
    } catch (IOException ex) {
        Log.i("fetch_process_info", "ex=" + ex.toString());
    }
    return memoryInfo.toString() + "\n\n" + result;
}
```

代码解释:

上述代码中首先通过 `ActivityManager` 对象获取其可用的内存, 然后通过查看 “`/proc/meminfo`” 内容获取更详细的信息。

程序运行后显示效果如图 16-9 所示。

如图 16-9 上可以看到, 笔者的手机总内存是 1GB, 当前可用 28MB, 以及其他有用的信息。

3. 获取磁盘信息

手机设备的磁盘信息可以通过 `df` 命令获取, 所以, 这里获取磁盘信息的方法和前面类似, 惟一不同的是, 这个是直接执行命令, 获取其命令的返回就可以了, 实现的流程和前述一样, 关键代码如下所示:

```
// 磁盘信息
public static String fetch_disk_info() {
    String result = null;
    CMDExecute cmdexe = new CMDExecute();
    try {
        String[] args = { "/system/bin/df" };
        result = cmdexe.run(args, "/system/bin/");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return result;
}
```

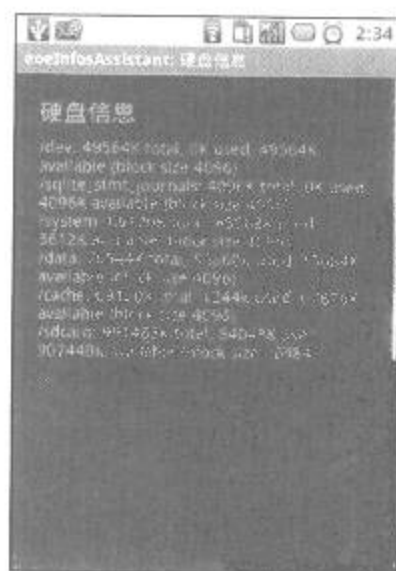
代码解释:

和前面讲解的类似, 使用 `CMDExecute` 调用系统的 “`/system/bin/df`” 命令获取其磁盘信息。程序运行后显示的效果如图 16-10 所示。

从图 16-10 可以看到, 笔者这款手机设备的磁盘和分区信息, 非常详细。



▲ 图 16-9 内存信息



▲ 图 16-10 硬盘信息

4. 获取网络信息

获取手机设备的网络信息也比较简单, 读取 `/system/bin/netcfg` 中的信息就可以了, 关键代码如下所示:


```

public static String fetch_netcfg_info() {
    Log.i("fetch_process_info", "start....");
    String result = null;
    CMDExecute cmdexe = new CMDExecute();
    try {
        String[] args = { "/system/bin/netcfg" };
        result = cmdexe.run(args, "/system/bin/");
    } catch (IOException ex) {
        Log.i("fetch_process_info", "ex=" + ex.toString());
    }
    return result;
}

```

 代码解释：

通过执行“/system/bin/netcfg”获取网络信息。

上述代码运行后显示的效果如图 16-11 所示。

5. 获取显示屏信息

除了显示手机的 CPU、内存、磁盘信息外，还有个非常重要的硬件，显示屏。下面就来讲解获取基本的显示屏信息。在 Android 中，它提供了 DisplayMetrics 类，可以通过 getApplicationContext()、getResources()、getDisplayMetrics() 初始化，进而读取其屏幕宽（widthPixels）、高（heightPixels）等信息，实现的关键代码如下所示：

```

public static String getDisplayMetrics(Context cx) {
    String str = "";
    DisplayMetrics dm = new DisplayMetrics();
    dm = cx.getApplicationContext().getResources().getDisplayMetrics();
    int screenWidth = dm.widthPixels;
    int screenHeight = dm.heightPixels;
    float density = dm.density;
    float xdpi = dm.xdpi;
    float ydpi = dm.ydpi;
    str += "The absolute width:" + String.valueOf(screenWidth) + "pixels\n";
    str += "The absolute heightin:" + String.valueOf(screenHeight)
    + "pixels\n";
    str += "The logical density of the display.:" + String.valueOf(density)
    + "\n";
    str += "X dimension :" + String.valueOf(xdpi) + "pixels per inch\n";
    str += "Y dimension :" + String.valueOf(ydpi) + "pixels per inch\n";
    return str;
}

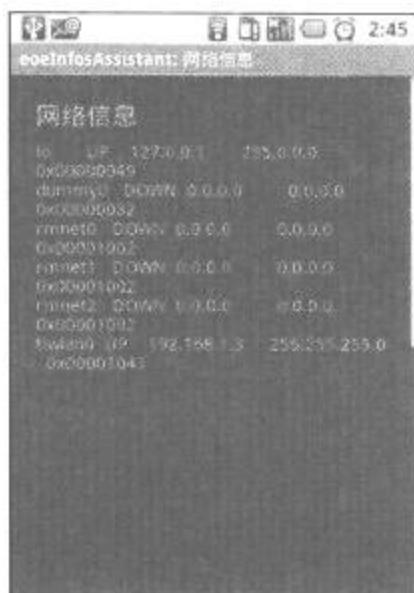
```

 代码解释：

首先初始化一个 DisplayMetrics 对象，然后通过调用 getApplicationContext().getResources().getDisplayMetrics() 获取其屏幕信息，然后读取其屏幕宽（widthPixels）、屏幕高（heightPixels）等就得到了屏幕信息。

上述代码运行后的效果如图 16-12 所示。

从图 16-12 所示可以看出，笔者当前这款手机显示屏的宽是 320，高是 480，水平分辨率是 159.3723，垂直分辨率是 320.8401。



▲ 图 16-11 网络信息



▲ 图 16-12 显示屏信息

16.2.4 查看软件信息

在 Android 上，可以在手机上随便安装自己喜欢的应用软件，下面要讲的这个功能就是收集并显示已经安装的应用软件信息。

获取已经安装的应用信息

在 Android 中，它提供了 `getPackageManager()`、`getInstalledApplications(0)` 方法，可以直接返回全部已经安装的应用列表。这个功能就是只需要获取列表，再进行显示在列表中就可以了。但是，需要注意的是，如果安装的软件比较多，那么其获取信息所花费的时间会比较多，为了更好地完善用户使用的体验，在获取列表时，需要在界面提示用户耐心等待，这就需要用到 Android 提供的另外一个功能 `Runnable`（`Runnable` 是一个抽象的接口，通过这个接口可以实现多线程）。

引入 `Runnable` 比较简单，只需要在定义类的时候实现 `Runnable` 接口就可以了，所以，这里的软件信息查看界面对应的 `Software.java` 类声明代码如下：

```
public class Software extends Activity implements Runnable {
    ...
}
```

然后需要在这个 `Activity` 启动的时候，引入进度条 `ProgressDialog` 来显示一个提示界面（关于 `ProgressDialog` 的详细介绍请查看本书第 7 章），`onCreate` 代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.softwares);
    setTitle("软件信息");
    itemList = (ListView) findViewById(R.id.itemlist);
    pd = ProgressDialog.show(this, "请稍候..", "正在收集你已经安装的软件信息...", true,
        false);
    Thread thread = new Thread(this);
    thread.start();
}
```

🔗 代码解释：

在该方法中创建了一个 `ProgressDialog`，并设定其提示信息。

然后实现其线程的 `run()` 方法，该方法实现其真正执行的逻辑，实现代码如下所示：

```
@Override
public void run() {
    fetch_installed_apps();
    handler.sendMessage(0);
}
```

 代码解释：

上述代码调用自定义的 `fetch_installed_apps()` 方法获取已经安装的应用信息，这个方法是比较消耗时间的，实现代码如下所示：

```
public List fetch_installed_apps(){
    List<ApplicationInfo> packages = getPackageManager().getInstalledApplications(0);
    list = new ArrayList<Map<String, Object>>(
        packages.size());
    Iterator<ApplicationInfo> l = packages.iterator();
    while (l.hasNext()) {
        Map<String, Object> map = new HashMap<String, Object>();
        ApplicationInfo app = (ApplicationInfo) l.next();
        String packageName = app.packageName;
        String label = "";
        try {
            label = getPackageManager().getApplicationLabel(app).toString();
        } catch (Exception e) {
            Log.i("Exception", e.toString());
        }
        map = new HashMap<String, Object>();
        map.put("name", label);
        map.put("desc", packageName);
        list.add(map);
    }
    return list;
}
```

 代码解释：

上述代码按照我们前面说到，使用 `getPackageManager().getInstalledApplications(0)` 获取已经安装的软件信息，进而构造用来显示的列表（List）对象，同时，界面通过进度条（`ProgressDialog`）显示提示信息，运行效果如图 16-13 所示。

当这个方法运行完成后，会调用 `handler.sendMessage(0)` 语句给 handler 发送一个通知消息，使其执行下面的动作，下面就是这个 handler 的实现方法，代码实现如下所示：

```
private Handler handler = new Handler() {
    public void handleMessage(Message msg) {
        refreshListItems();
        pd.dismiss();
    }
};
```

 代码解释：

上述代码中，当其接收到 `run()` 线程传递的消息后，先调用 `refreshListItems()` 方法显示列表，最后调用进度条 `ProgressDialog` 的 `dismiss` 方法使其等待提示消失。这里有必要再看一下 `refreshListItems()` 方法，其实现代码如下所示：

```
private void refreshListItems() {
    list = fetch_installed_apps();
    SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.info_row,
        new String[] { "name", "desc" }, new int[] { R.id.name,
```

```

R.id.desc ));
itemlist.setAdapter(notes);
setTitle("软件信息, 已经安装"+list.size()+"款应用.");
}

```

✎ 代码解释:

这些代码在本书前面章节已经见过类似的了, 显示已经安装的应用列表的同时, 在 Title 上显示一共安装了多少款应用, 其运行效果如图 16-14 所示。

从图 16-14 显示可以看出, 笔者的手机一共安装了 82 款应用, 列表中的每行显示一个应用的名字和其对应的包的名字。



▲ 图 16-13 等待提示信息



▲ 图 16-14 已安装应用列表

16.2.5 获取运行时信息

前面分别获取了系统信息、硬件信息和软件信息, 本小节将用来获取运行时的一些信息, 例如, 后台运行的 Service、Task, 以及进程信息, 其运行界面如图 16-15 所示。

1. 获取正在运行的 Service 信息

可以通过调用 `context.getSystemService(Context.ACTIVITY_SERVICE)` 获取 `ActivityManager`, 进而通过系统提供的方法 `getRunningServices(int maxNum)` 获取正在运行的服务列表 (`RunningServiceInfo`), 再对其结果进一步分析, 得到服务对应的进程名及其他信息, 实现的关键代码如下所示:

```

//正在运行的服务列表
public static String getRunningServicesInfo(Context context) {
    StringBuffer serviceInfo = new StringBuffer();
    final ActivityManager activityManager = (ActivityManager) context
        .getSystemService(Context.ACTIVITY_SERVICE);
    List<RunningServiceInfo> services = activityManager.getRunningServices(100);

    Iterator<RunningServiceInfo> i = services.iterator();
    while (i.hasNext()) {
        RunningServiceInfo si = (RunningServiceInfo) i.next();
        serviceInfo.append("pid: ").append(si.pid);
        serviceInfo.append("\nprocess: ").append(si.process);
        serviceInfo.append("\nservice: ").append(si.service);
        serviceInfo.append("\ncrashCount: ").append(si.crashCount);
        serviceInfo.append("\nclientCount: ").append(si.clientCount);
    }
}

```

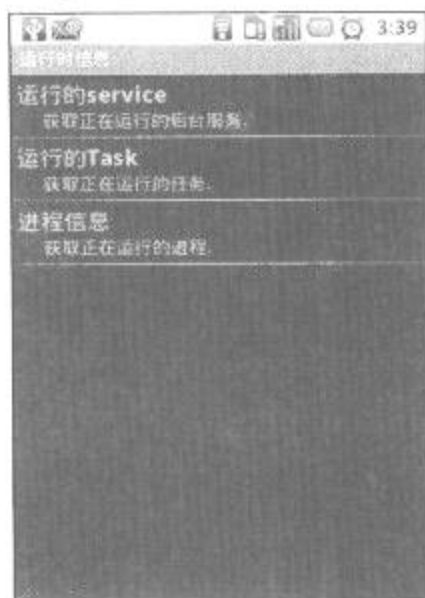
```

serviceInfo.append("\nactiveSince:").append(ToolHelper.formatData(si.activeSince));
serviceInfo.append("\nlastActivityTime:").append(ToolHelper.formatData(si.lastActivityTime));
serviceInfo.append("\n\n");
}
return serviceInfo.toString();
}

```

代码解释：

上述代码调用 `activityManager.getRunningServices(100)` 获取正在运行的服务，并依次遍历得到每个服务对应的 `pid`，进程等信息，运行效果如图 16-16 所示。



▲ 图 16-15 运行时信息主界面



▲ 图 16-16 正在运行的服务

从图 16-16 所示可以看出，后台正在运行的服务及对应的进程名的信息。（注：但是发现其 API 中提供的获取 `activeSince` 和 `lastActivityTime` 的结果不准确，可能是 Adnroid 的一个 Bug。）

2. 获取正在运行的 Task 信息

获取正在运行的 Task 信息的方式和前面实现的类似，其调用的是 `activityManager.getRunningTasks(int maxNum)` 来获取对应的正在运行的任务信息列表（`RunningTaskInfo`），进而分析、显示其任务信息，其关键代码如下所示：

```

public static String getRunningTasksInfo(Context context) {
    StringBuffer sInfo = new StringBuffer();
    final ActivityManager activityManager = (ActivityManager) context
        .getSystemService(Context.ACTIVITY_SERVICE);
    List<RunningTaskInfo> tasks = activityManager.getRunningTasks(100);
    Iterator<RunningTaskInfo> l = tasks.iterator();
    while (l.hasNext()) {
        RunningTaskInfo ti = (RunningTaskInfo) l.next();
        sInfo.append("id: ").append(ti.id);
        sInfo.append("\nbaseActivity: ").append(ti.baseActivity.flattenToString());
        sInfo.append("\nnumActivities: ").append(ti.numActivities);
        sInfo.append("\nnumRunning: ").append(ti.numRunning);
        sInfo.append("\ndescription: ").append(ti.description);
        sInfo.append("\n\n");
    }
    return sInfo.toString();
}

```


代码解释:

上述代码调用系统提供的 `activityManager.getRunningTasks(100)` 方法获取任务列表, 依次获取其对应的 `id` 等信息, 运行效果如图 16-17 所示。

从图 16-17 显示可以看出, 获取手机上正在运行的 Task 的列表和其对应的进程信息, 这对用户了解设备运行情况非常有用。

3. 获取正在运行的进程信息

下面讲解的是需要获取的正在运行的进程信息, 通过前面的讲解, 相信读者可以比较方便的实现这个功能了, 具体操作不再赘述, 其关键代码如下所示:

```
public static String fetch_process_info() {
    Log.i("fetch_process_info", "start...");
    String result = null;
    CMDEXecute cmdexe = new CMDEXecute();
    try {
        String[] args = { "/system/bin/top", "-n", "1" };
        result = cmdexe.run(args, "/system/bin/");
    } catch (IOException ex) {
        Log.i("fetch_process_info", "ex=" + ex.toString());
    }
    return result;
}
```

代码解释:

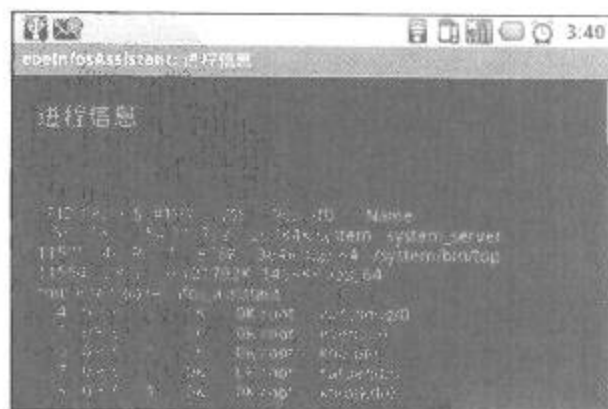
还是通过 `CMD Executo` 的方式来运行系统命令。

上述代码运行后显示的效果如图 16-18 所示。

从图 16-18 显示的界面可以看出, 通过这个功能可以非常详细地了解到正在运行的进程和各个进程所消耗的资源情况。



▲ 图 16-17 运行的 Task 信息



▲ 图 16-18 正在运行的进程

16.2.6 文件浏览器

最后一个功能是文件浏览器, 通过这个功能, 用户可以遍历浏览整个文件系统, 以便更好地了

解手机设备状况。在主界面单击最后一行，将执行下列代码：

```
case 4:
    intent.setClass(eoeInfosAssistant.this, FSExplorer.class);
    startActivity(intent);
    break;
```

将会执行 FSExplorer.java，其主界面模板和前面一样，就一个列表（List），代码如下所示：

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

    <ListView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/itemlist" />
</LinearLayout>
```

接着看 FSExplorer 的 onCreate 方法，代码如下所示：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.files);
    setTitle("文件浏览器");
    itemlist = (ListView) findViewById(R.id.itemlist);
    refreshListItems(path);
}
```

然后其调用 refreshListItems() 获取文件列表，代码如下所示：

```
private void refreshListItems(String path) {
    setTitle("文件浏览器 > "+path);
    list = buildListForSimpleAdapter(path);
    SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.file_row,
        new String[] { "name", "path", "img"}, new int[] { R.id.name,
        R.id.desc, R.id.img});
    itemlist.setAdapter(notes);
    itemlist.setOnItemClickListener(this);
    itemlist.setSelection(0);
}
```

这其中首先调用 buildListForSimpleAdapter(path) 获取文件列表，代码如下所示：

```
private List<Map<String, Object>> buildListForSimpleAdapter(String path) {
    File[] files = new File(path).listFiles();
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(files.length);
    Map<String, Object> root = new HashMap<String, Object>();
    root.put("name", "/");
    root.put("img", R.drawable.file_root);
    root.put("path", "go to root directory");
    list.add(root);
    Map<String, Object> pmap = new HashMap<String, Object>();
    pmap.put("name", "..");
    pmap.put("img", R.drawable.file_paranet);
    pmap.put("path", "go to paranet Directory");
    list.add(pmap);
    for (File file : files) {
        Map<String, Object> map = new HashMap<String, Object>();
        if (file.isDirectory()) {
            map.put("img", R.drawable.directory);
        } else {
            map.put("img", R.drawable.file_doc);
        }
    }
}
```

```

map.put("name", file.getName());
map.put("path", file.getPath());
list.add(map);
}
return lists;
}

```

代码解释:

上述代码非常清晰,使用 `File(path).listFiles()` 方法就可以获取文件和文件夹列表,最后存入 `List` (列表组件) 显示出来就可以了。然后,需要处理一下如何进入子目录,并获取和显示其内部的文件夹和文件,也就是单击每行时响应的实现,代码如下所示:

```

@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
    Log.i(TAG, "item clicked! [" + position + "]");
    if (position == 0) {
        path = "/";
        refreshListItems(path);
    } else if (position == 1) {
        goToParent();
    } else {
        path = (String) list.get(position).get("path");
        File file = new File(path);
        if (file.isDirectory())
            refreshListItems(path);
        else
            Toast.makeText(FSExplorer.this,
                getString(R.string.is_file),
                Toast.LENGTH_SHORT).show();
    }
}

```

代码解释:

这段代码的逻辑就是,如果单击的是第一行(也就是“/”这行),那么,就使用 `refreshListItems('/')` 获取根目录下的文件系统;如果单击的是第二行(也就是上一级),那么,就调用 `goToParent()` 显示其上一级的文件系统;其他入口下就获取单击的文件,判断是文件还是目录,如果是文件就给出一个提示说没有子目录,如果是目录,则进入该目录并显示它内部的文件系统。

`goToParent()` 实现的代码如下所示:

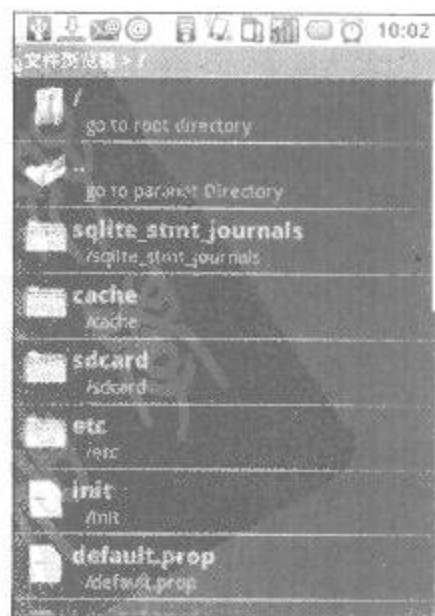
```

private void goToParent() {
    File file = new File(path);
    File str_pa = file.getParentFile();
    if (str_pa == null) {
        Toast.makeText(FSExplorer.this,
            getString(R.string.is_root_dir),
            Toast.LENGTH_SHORT).show();
        refreshListItems(path);
    } else {
        path = str_pa.getAbsolutePath();
        refreshListItems(path);
    }
}

```

代码解释:

上述代码实现跳转到上一级功能,运行效果如图 16-19、图 16-20 和图 16-21 所示。



▲ 图 16-19 文件浏览效果 (1)



▲ 图 16-20 文件浏览效果 (2)



▲ 图 16-21 文件浏览效果 (3)

16.2.7 项目的细节完善

到这里差不多需要的功能都实现了,在发布之前还需要再检查一下细节方面,尤其需要注意的,这个应用在获取运营商信息时,需要申请读取电话状态 (READ_PHONE_STATE) 权限,在获取 Task 信息时,需要申请获取任务 (GET_TASKS) 权限,所以,最后请确认在 AndroidManifest.xml 文件中添加了如下的权限需求,代码如下所示:

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

最后看一下 AndroidManifest.xml 文件代码:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eoemobile.infos_assistant"
    android:versionCode="1"
    android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".eoeInfosAssistant"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="System"></activity>
        <activity android:name="Hardware"></activity>
        <activity android:name="Software"></activity>
        <activity android:name="Runing"></activity>
        <activity android:name="ShowInfo"></activity>
        <activity android:name="FSExplorer"></activity>

    </application>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
    <uses-permission android:name="android.permission.GET_TASKS"></uses-permission>

</manifest>
```

🔗 代码解释:

上述代码中包含了所有的 Activity 和需要申请的权限 (uses-permission)。

16.3 手机信息查看助手功能展望

到这里，完成了预设的项目功能目标，整个应用差不多已经完成了，但是，可以在这里再展望一下这个应用还可以补充和完善的地方，例如，可以在原来的应用基础上加入新的功能，如电池信息、wifi 信息等的收集和查看。其他的功能和方向，还是看各自的需求了，如果在此基础上开发出更多的功能，也欢迎和笔者交流和分享。

16.4 项目的打包、签名和发布

写完手机助手应用后，就可以打包、签名和发布应用啦，其方法在本书第 4 章已经比较详细地介绍过，这里就不赘述了，只将操作流程演示一下，效果如图 16-22、图 16-23 和图 16-24 所示。



▲ 图 16-22 打包界面



▲ 图 16-23 签名界面



▲ 图 16-24 发布界面

16.5 本章小结

本章以一个手机信息查看助手的应用分析为起点,讲述了如何获取和查看手机设备自身的一些信息,例如,系统信息、硬件配置信息、软件安装情况,以及一些运行时的信息。通过本章的学习,相信能使读者在硬件操作方面积累一定的经验,可以顺利地进入手机硬件相关的应用开发。



第 17 章 芝麻开门—Android 底层 开发和移植概述

学习目标

- 能自行编译 Android 系统; ☐
- 了解 Android 移植技术的重要性; ☐
- 认识 Android 移植技术的重点; ☐
- 开始 Android 底层技术的研究。 ☐

17.1 Android 原始码下载

学习 Android 应用程序设计外,更要往底层技术探索,才能成为全方位的 Android 系统开发高手。本章将介绍搭建 Android 底层开发环境的方法。

本章将以 Neo FreeRunner 开发者手机为例,介绍取得 Android 原始码、编译、实机测试的方法,最后说明 Android 底层最关键的技术“移植”知识。

1. 安装 repo 工具

利用 repo 工具才能下载完整的 Android 原始码,因此,必须先安装 repo 工具。下载并安装 repo 工具的网站为: <http://source.android.com/download#TOC-Installing-Repo>。

2. 下载原始码

Neo FreeRunner 采用的是 s3c2443 晶片,它是 ARMv4 的架构,Openmoko 与 Koolu 公司将 Android 移植到 Neo FreeRunner 并进行分支维护,因此,访问 Koolu 网站可以获取 Neo FreeRunner 的完整 Android 原始码,具体如下所示:

```
$ mkdir mydroid
$ cd mydroid
$ repo init -u git://git.koolu.org/freerunner/platform/manifest.git -b koolu-1.0
$ repo sync
```

3. 安装 Toolchain

要将 Android 编译成 ARMv4 的格式,必须安装 cross toolchain。访问以下网站下载

Android/FreeRunner toolchain。网址为：<http://trac.koolu.org/wiki/Toolchain>。

在此网站下载 android-toolchain_20081019-koolu1_i386.deb 文件，并进行安装：

```
$ sudo dpkg --install android-toolchain_20081019-koolu1_i386.deb。
```

4. 设定 buildspec.mk

在取回的 Android 原始码根目录下，建立 buildspec.mk 文件，内容如下所示：

```
TARGET_TOOLS_PREFIX := /opt/android-toolchain/bin/armv4t-android-eabi-
```

5. 编译 Android 系统

接着在 Android 原始码根目录下执行编译命令：

```
$ make TARGET_PRODUCT=freerunner。
```

6. 制作 androidfs.jffs2 文件

编译完成后的 Android 系统存放于 out/目录下，要将 Android 放到 FreeRunner 开发手机上执行，必须将 Android 系统制作成 jffs2 的镜像文件后再烧录到 FreeRunner 手机上。访问以下网站下载 cp_android_OS 以及 mkjffs2 两个文件，网址为：<http://www.openmokoin.org/android>。

请将以上两个文件放置于 Android 原始码根目录下，并加上可执行属性：

```
$ chmod a+x ./cp_android_OS ./mkjffs2。
```

在 Android 原始码根目录下，先执行 cp_android_OS (加上 'release' 参数)，再执行 mkjffs2 命令：

```
$ ./cp_android_OS release
$ ./mkjffs2
```

执行后即可取得 androidfs.jffs2 文件。

17.2 实机测试

将上述建立好的 androidfs.jffs2 文档代码烧录至 Neo FreeRunner 开发手机中，即可进行实机测试了。

17.2.1 Neo FreeRunner 介绍

Neo FreeRunner 是 Openmoko 公司的专业研发成果，目前由 Openmoko 公司制造销售，这是一款专门针对开发者与教育用途所使用的开发平台，开发者与师生可利用此平台进行编程与技术研究，因此，很适合用于学校的实验教学，以达成理论与实践相结合的目的。

Neo FreeRunner 支持 GSM/GPRS/AGPS，也是世界上第一个 Free Hardware 的手机硬件平台，老师与学生也能由网站取得电路图、设计图与源代码，这让 Neo FreeRunner 更紧密地与学校实验课程结合。Free Software 与 Free Hardware 的专业网站为：www.openmokoin.com。Neo FreeRunner 执行 Android 实机界面如图 17-1 所示。

Neo FreeRunner 采用在学校中有影响的 s3c2443 作为处理器，并加入了 GSM/GPRS/GPS 的功能，让 Neo FreeRunner 不只是开发板，而是具备真正手机功能的平台，Neo FreeRunner 的详细硬件规格如图 17-2 所示。



图 17-1 Neo FreeRunner 执行 Android 实机界面

Neo FreeRunner (GTA02) Simplified hardware component diagram

2008 Kim Hauritz, some rights reserved - CC: A-NC-SA

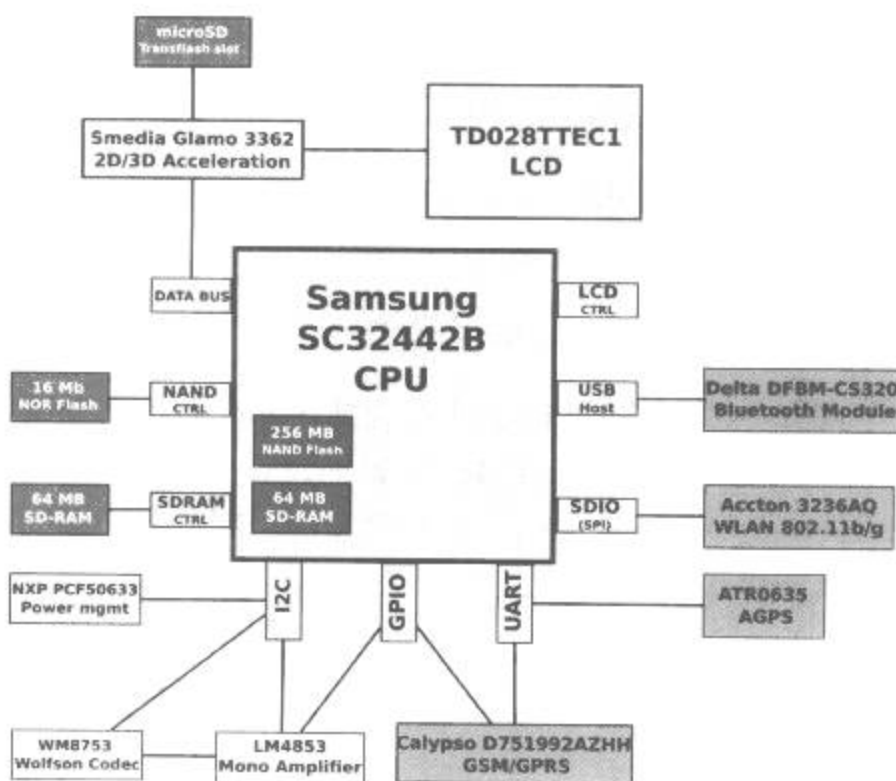


图 17-2 Neo FreeRunner 的详细硬件规格

17.2.2 烧录 androidfs.jffs2

利用 Openmoko 公司提供的 dfu-util 工具即可将 androidfs.jffs2 文档代码烧录到 Neo FreeRunner 的 NAND flash 中，实现方法如下。

1. 下载 dfu-util

由 Openmokoin.org 网站下载 dfu-util，并存放到系统的/sbin 目录下：

```
$ wget http://www.openmokoin.org/downloads/tools/dfu-util
$ chmod a+x dfu-util
$ sudo mv dfu-util /sbin
```

2. 连接手机

将手机开机至 U-Boot 模式，并以 USB 接口连接手机与计算机，如图 17-3 所示。

3. 烧录 androidfs.jffs2

执行以下指令，将新的 Android 作业系统烧录至 Neo FreeRunner：



图 17-3 将 Neo FreeRunner 开机到 U-Boot 模式

```
$ sudo ./dfu-util -a rootfs -R -D ./androidfs.jffs2
```

4. 更新 kernel

如果 Neo FreeRunner 无法开机至 Android，请下载 Openmokoin 提供的 kernel，并更新手机上的 kernel 即可。可以在 Openmokoin 网站下载最新的 kernel，下载网址为：

<http://www.openmokoin.org/downloads/android/kernel>。

将下载的 kernel 烧录至手机即可。同样先将手机开机到 U-Boot 模式，然后执行以下指令：

```
$ sudo ./dfu-util -a kernel -R -D ./ulmage-freerunner.img。
```

17.3 Android 移植技术概论

研究 Android 底层技术是为了建立移植的能力，因此，本节将针对 Android 的移植技术做一个总体性的介绍。

17.3.1 Android 底层技术的重要性

对于有志于深入 Android 系统开发的读者而言，了解 Android 的原始码目录结构，是重要且相当基本的起步工作。Android 原始码的结构非常清楚易懂，以下说明 Android 原始码的目录结构与用途。

研究 Android 作业系统底层技术的目的之一为培养 Android 移植技术的能力。任何 Android 应用程序都需要底层的支援，才能顺利运作在硬件平台上，而这也正是 Android 产品开发的核心理能力。Android 产品开发阶段，不但需要一个高品质的硬件载体，而且也需要将 Android 系统移植到该硬件上并进行调试，才能让 Android 应用程序在产品上顺利执行。

17.3.2 Android 移植项目介绍

实际进行 Android 系统移植时，需要考虑以下 6 方面。

1. Applications

Android 应用程序采用 Java 语法编写，因此，具备良好的高适配性，没有移植的问题。但是，由于 Android 每个版本之间的 API level 不相同，因此，测试 API 的兼容性是这个部份一项重要的工作。

当应用程序执行在新版本的 Android 作业系统上时，必须检查 Google 是否有更新的 API level。API level 更新时，可以参考 Google 提供的 API change 文件以进行相容性的检查，下表给出了 Android API Level 变更记录。

Android API Level 变更记录

System Image	API Level	Release
Android 1.5	3	2009.4.27
Android 1.1	2	2009.2.10
Android 1.0	1	2008.9.23

2. Product

想要在 Android 里移植一个新的硬件，若此硬件使用的处理器已经是 Android 所支持的架构时，只需建立一个新的 product 分支即可。

Android 原始码目录下的 vendor/ 目录存放不同产品的 Android 设定及其他文件。Android 的分支 (branch) 是以“产品”的概念做维护，因此，若在目前 Cupcake (Android 1.5) 能支持的平台 (architecture) 上做移植的话，是一个较简单的工作，只需要在 vendor/ 里新增自己的 product 并修改 AndroidBoard.mk、AndroidProducts.mk，以及 BoardConfig.mk 即可完成一个 Board/Product 的新分支。

3. Application Framework

Framework 层的技术重点并不在移植面，而是在 Framework 的扩充与开发技术。以大方向来看，加入新的 Library 时需要扩充 Application framework。Android Framework 以 JNI 呼叫下一层的 Library，但是 Application 不直接呼叫 Library，因此，让 Android Framework 的设计必须更加严谨才行。

Android Framework 层，部份需要考虑硬件的规格，例如，Surface Manager 需要考虑是否有 GPU 或硬件加速。但在 Android Application 建立一个 Surface holder (触摸模式) 时，若是将 Surface 的类型设定为 SURFACE_TYPE_HARDWARE，就必须针对硬体加速与 DMA 做移植工作了。

4. Libraries

Libraries 层方面需要注意，有些程序库以 assembly 为实现算法 (algorithm)，因此，针对这个部份需要做一些修改。例如，OpenCore 就包含了以下的 assembly (一种算法) 实现，在不同处理器架构间使用 OpenCore 时就需要进行移植：

```
./codecs_v2/audio/mp3/dec/src/asm/pvmp3_polyphase_filter_window_gcc.s
```

```

./codecs_v2/audio/mp3/dec/src/asm/pvmp3_mdct_18.s
./codecs_v2/audio/mp3/dec/src/asm/pvmp3_dct_9.s
./codecs_v2/audio/mp3/dec/src/asm/pvmp3_dct_9_gcc.s
./codecs_v2/audio/mp3/dec/src/asm/pvmp3_mdct_18_gcc.s
./codecs_v2/audio/mp3/dec/src/asm/pvmp3_polyphase_filter_window.s
./codecs_v2/audio/mp3/dec/src/asm/pvmp3_dct_16_gcc.s

```

另外，在 shared library 这一层里，最基础的 C library 并非采用 GNU libc，而是 Google 自行发展的 Bionic。Bionic 是 Android 专属的小型定制化 C library，主要是由 BSD C library 移植而来。支持 Linux kernel 的重要实现，如 system call、dynamic linker & loader、thread 等都是检查的重点。Bionic 里有一个 libthread_db 是基于 Linux futexes 的 thread 实现，很适合手机这样的小型系统使用，Bionic 的结构如图 17-4 所示。

Bionic 的 libc 实现，包含了依赖于处理器架构的部份，代码存放结构如图 17-5 所示。

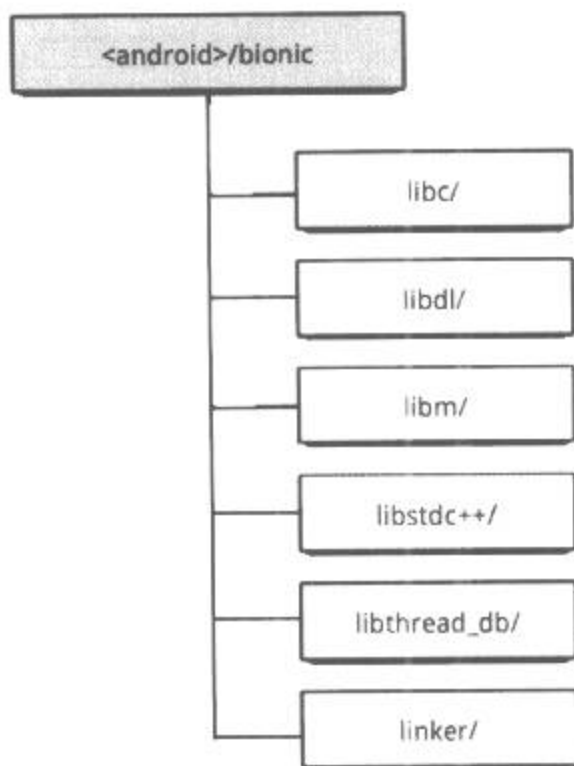


图 17-4 Bionic 的结构

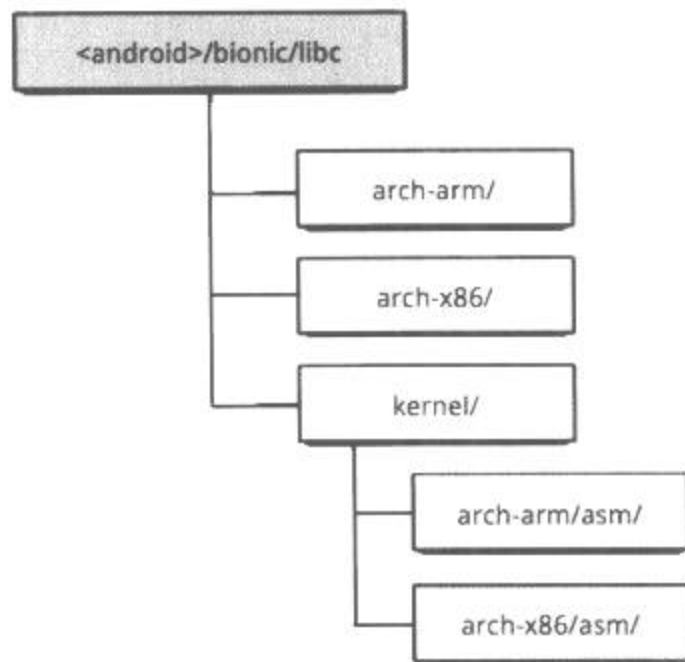


图 17-5 代码存放结构

bionic/libc/kernel/ 并非标准的 kernel header files。Android 的 kernel header 是利用工具由 Linux kernel header 所产生的新文件，主要目的是为了保留使用到的常数、资料结构给 userspace（使用场景）使用。

5. Android Runtime

Android Runtime 指的是 Dalvik VM（Android 虚拟机），Dalvik VM 在不同处理器架构间，需要考虑的移植项目有以下两项：

- （1）JNI（Java Native Interface）的实作；
- （2）Interpreter（解释器）的实作。

以 Android 1.5（Cupcake）来说，Dalvik VM 的 Interpreter 已经能支持 armv4、armv5te 以及 x86 架构，因此，这个部份的移植工作并不多，JNI 的实现中处理器架构相关部份如图 17-6 所示。

6. Linux Kernel

Linux kernel 的部份, 除了驱动的移植与开发外, Google 也加入了几个专门针对 Android 的功能到 Linux kernel:

```
-Binder;
-Ashmem (Android shared memory);
-PMEM (Processor memory allocator);
-logcat (Android Logger);
-wakelock (Android Power Management)。
```

17.3.3 Android 的技术优点

Android 操作系统本身具备许多“先天优势”, 让它在移植与 Framework 开发工作上, 都比以前传统的嵌入式 Linux 系统更有效率。

Android 操作系统的好处是“将开发者局限在应用层 (application level)”的开发, 并透过一个设计良好的 Application Framework 将 Library 层“包装起来”。传统

GNU/Linux 系统的“开源模式”是“从里到外”全面开放, 应用程序来自四面八方, 每个应用程序底层使用到的 Library 并不相同, 这让 Linux 平台的发展容易失控, 造成 Linux distribution 上虽然收录了丰富的应用程序, 但相对的也要纳入非常多的 Shared Library (共享库)。

Android 虽然也采用了其他 open source (开源) 的专业成果, 但 Android 以很聪明的方式, 解决传统 Linux 开放手机平台的“相依性”问题, 这也是过去长久以来, 使用 (leverage) 开放源代码专业开发产品的大问题。Application Framework 采用 Java 程序语言, 并软性地将开发者限制在 Application level 是 Android 解决上述技术难题的一个关键。

互联网上有着数以万计的 Free & Open Source Software 程序源代码, 而被 Android 采纳的源代码仅有约 60 个左右, 比起传统 Linux distribution 必须收录上千个套件的数量来看, Android 未来若能发扬光大, 必将扮演开源软件发展模式的推手。

传统的 Embedded Linux 系统程序基于 GNU libc 以及大量的相依程序库 (Library Dependencies), 因此, 很容易有“牵一发而动全身”的问题出现。例如, 某一个 Library 的 API 变动 (可能是函数改名或移除) 将使得其他程序库与应用程序执行错误, 这时就必须修改源代码并重新编译才能解决问题。

这个问题的主因是因为 Linux 系统是采取动态程序库 (Shared Libraries) 的机制, 程序库的变更虽然只需要换掉动态程序库文档, 但是应用程序在执行时, 才会产生“无法载入”的错误, 除非是定期进行系统重编译, 否则很难即时修正此错误。

Android 的底层并无太复杂的程序依赖性的问题, 这使得 Android 可以比较容易将系统与 IDE 开发工具做整合。在标准 C 程序库 (C library) 方面, Google 则是采用 BSD 授权实作了一份适合手机系统使用的版本, 无疑是一个值得称赞的做法。

由上述的分析来看, Android 操作系统在系统层 (library、kernel) 的移植工作将不再花费工程人员大量的时间, 同时就系统层的调试工作来看, 也能有更具体明确的调试项目。

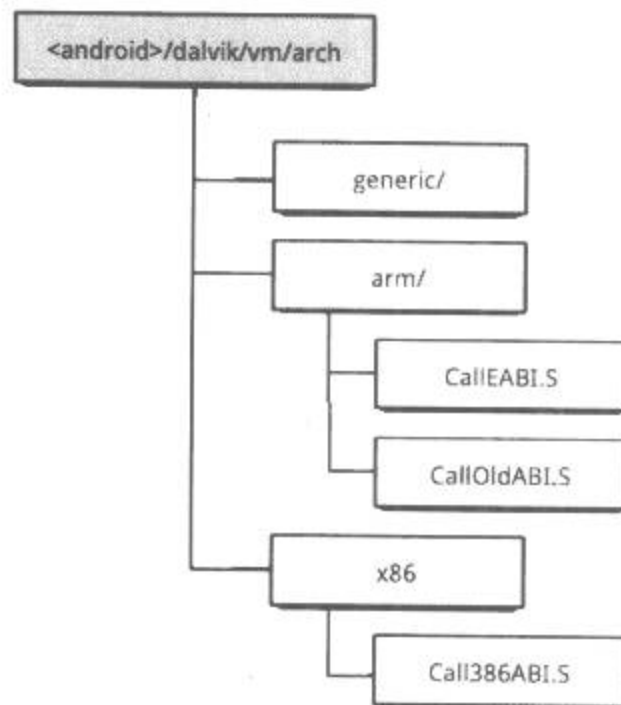


图 17-6 JNI 的实现中处理器架构相关部份