



国内首本基于**Android 2.0**的经典著作，5大专业社区一致鼎力推荐！



Android Unleashed

Android

应用开发揭秘



杨丰盛◎著



机械工业出版社
China Machine Press

Android应用开发揭秘

Windows操作系统的诞生成就了微软的霸主地位，也造就了PC时代的繁荣。然而，以Android和iPhone手机为代表的智能移动设备的发明却敲响了PC时代的丧钟！移动互联网时代（3G时代）已经来临，谁会成为这些移动设备上的主宰？毫无疑问，它就是Android——PC时代的Windows！

移动互联网还是一个新生的婴儿，各种移动设备上的操作系统群雄争霸！与Symbian、iPhone OS、Windows Mobile相比，Android有着天生的优势——完全开放和免费，对广大开发者和手机厂商而言，这是何等的诱人！此外，在Google和以其为首的Android手机联盟的大力支持和推广下，Android不仅得到了全球开发者社区的关注，而且一大批世界一流的手机厂商都已经或准备采用Android。

拥抱Android开发，拥抱移动开发的未来！

如果你也在思考下面这些问题，也许本书就是你想要的！

- Android开发与传统的J2ME开发有何相似与不同？
- 如何通过Shared Preferences、Files、Network和SQLite等方式高效实现Android数据的存储？又如何通过Content Providers轻松地实现Android数据的共享？
- 如何使用Open Core、MediaPlayer、MediaRecorder方便快速地开发出包含音频和视频等流媒体的丰富多媒体应用？
- 如何利用Android 2.0中新增的蓝牙特性开发包含蓝牙功能的应用？又如何使用蓝牙API来完善应用的网络功能？
- 如何解决Android网络通信中的乱码问题？
- 在Android中如何使用语音服务和 Google Map API？Android如何访问摄像头、传感器等硬件的API？
- 如何进行Widget开发？如何用各种Android组件来打造漂亮的UI界面？
- Android如何解析XML数据？又如何提高解析速度和减少对内存、CPU资源的消耗？
- 如何使用OpenGL ES在Android平台上开发出绚丽的3D应用？在Android平台上如何更好地设计和实现游戏引擎？
- 如何对Android应用进行优化？如何进行程序性能测试？如何实现UI、zipalign和图片优化？
- 如何通过NDK利用C、C++以及通过ASE利用Python等脚本语言开发Android应用？

作者简介

杨丰盛 Android应用开发先驱，对Android有深入研究，实战经验极其丰富。精通Java、C、C++等语言，专注于移动通信软件开发，在机顶盒软件开发和MTK平台软件开发方面有非常深厚的积累。2007年获得中国软件行业协会游戏软件分会(CGIA)认证及国际游戏开发教育联合会国际认证。曾经领导和参与《三国群英传说》、《大航海传奇》、《美少女养成计划》等经典游戏的开发。

客服热线：(010) 88378991, 88361066
 购书热线：(010) 68326294, 88379649, 68995259
 投稿热线：(010) 88379604
 读者信箱：hzsj@hzbook.com

华章网站 <http://www.hzbook.com>

网上购书：www.china-pub.com

封面设计：王建敏



上架指导：计算机 / 程序设计 / 移动开发

ISBN 978-7-111-29195-4



9 787111 291954

定价：69.00 元

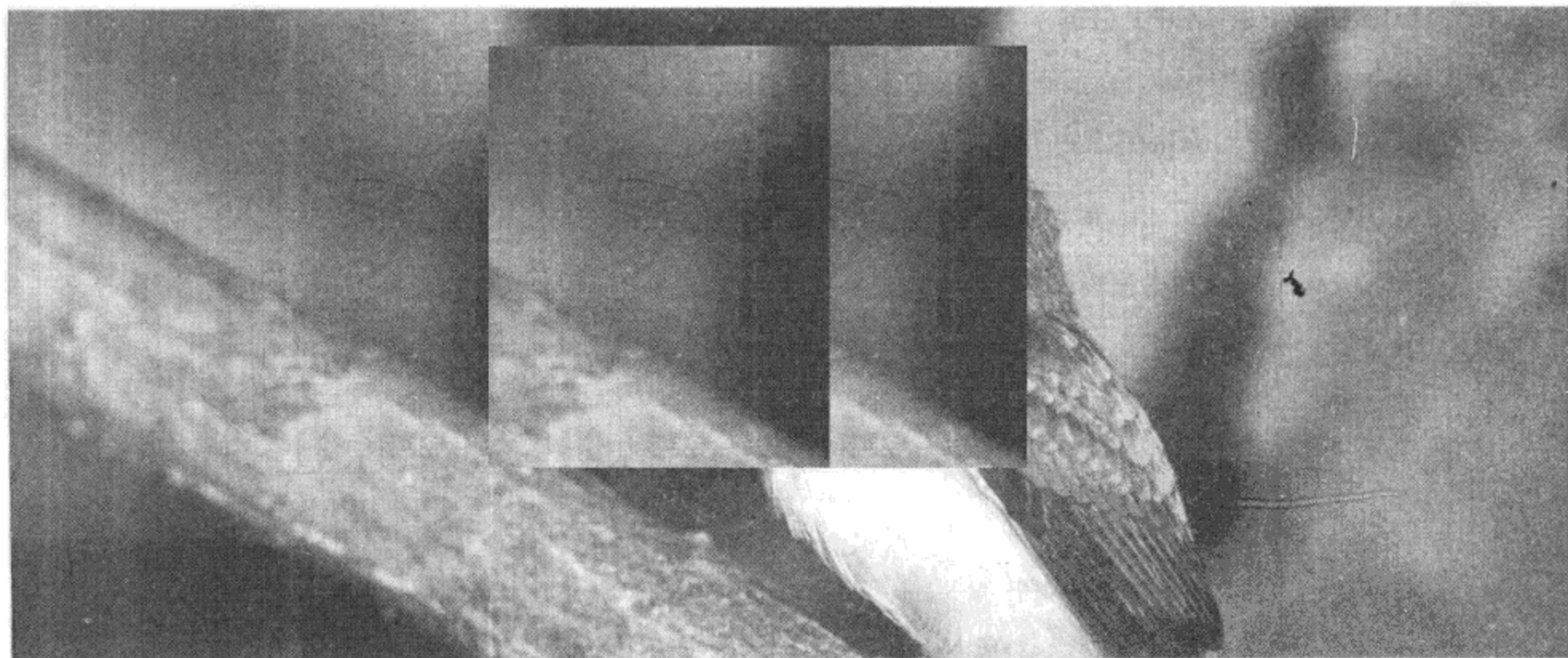
TN929.53
Y176

Android Unleashed

-32

Android

应用开发揭秘



TN 929.53
Y176

杨丰盛◎著



机械工业出版社
China Machine Press

本书赞誉

20 世纪 90 年代初，裘伯君、鲍岳桥等 IT 行业的前辈“单打独斗”就能开发出脍炙人口的应用，如今做一个项目动辄就需要数百人的大规模团队和千万级的巨额开发费用，程序员真的没有了展露个人才华和创意的机会吗？我们正站在移动技术改变人类生活方式的十字路口，而移动互联网正是这一切的关键。Android 以全新的开放平台和全球化的市场，为小团队提供了一个充分展现自己的舞台。本书为所有 Android 开发者提供了绝佳的参考，不可不读！

——Android 中文站 (<http://www.androidin.com/>)

与已经出版的所有同类书相比，本书内容更全面，几乎涵盖了 Android 开发的所有方面；实战性更强，不仅各个知识点都有翔实的范例，而且还包含多个实用的完整案例；主题更新颖，Android 2.0 中的各种最新特性一览无余……本书值得各种水平层次的 Android 应用开发者阅读，强烈推荐！

——Android 中文用户组

随着 3G 技术的成熟和智能手机的不断普及，移动应用的需求与日俱增，移动应用开发成为当下最热门的技术之一。在 Google 和 Android 手机联盟的共同推动下，Android 在众多移动应用开发平台中脱颖而出。本书的出版对于广大 Android 应用开发者来说不啻是一种福音，它将为 Android 开发者社区注入强大的活力！

——毕惠子 Android 实验室 (<http://www.androidlab.cn/>)

随着移动智能设备的普及，我国移动应用的需求即将迎来“井喷”，本书能让你轻松转型为 Android 开发者，助你笑傲移动应用开发之巅。极力推荐！

——谷奥 (<http://www.google.org.cn/>)

这是一本参考手册，内容的完整性和系统性几乎无可挑剔，可作为广大 Android 开发者的案头必备书；这是一部权威指南，基础知识部分翔实而丰富，高级知识部分深入且饱含最佳实践，能从本质上提升开发者对 Android 的理解和开发水平。尤为值得一提的是，Android 2.0 中新增了大量激动人心的新特性，不仅支持多点触摸设备、软键盘，而且还支持多账户在线管理、蓝牙……作为国内第一本基于 Android 2.0 的著作，本书可谓极具前瞻性，第一时间将这些新特性完美地呈现给了广大读者。

——安卓网 (<http://www.hiapk.com/>)

3G 牌照在国内发放后, 3G、Andriod、iPhone、Google、苹果、手机软件、移动开发等词越来越充斥于耳。随着 3G 网络的大规模建设和智能手机的迅速普及, 移动互联网时代已经微笑着迎面而来。

以创新的搜索引擎技术而一跃成为互联网巨头的 Google, 无线搜索成为 Google 进军移动互联网的一块基石。早在 2007 年, Google 中国就把无线搜索当作战略重心, 不断推出新产品, 尝试通过户外媒体推广移动搜索产品, 并积极与运营商、终端厂商、浏览器厂商等达成战略合作。

Android 操作系统是 Google 最具杀伤力的武器之一。苹果以其天才的创新, 使得 iPhone 在全球迅速拥有了数百万忠实“粉丝”, 而 Android 作为第一个完整、开放、免费的手机平台, 使开发者在为其开发程序时拥有更大的自由。与 Windows Mobile、Symbian 等厂商不同的是, Android 操作系统免费向开发人员提供, 这样可节省近三成成本, 得到了众多厂商与开发者的拥护。最早进入 Andriod 市场的宏达电已经陆续在一年内推出了 G1、Magic、Hero、Tattoo 等 4 款手机, 三星也在近期推出了 Galaxy i7500, 连摩托罗拉也推出了新款 Andorid 手机 Cliq, 中国移动也以 Android 为基础开发了 Ophone 平台。这些发展证明 Android 已经成为智能手机市场的重要发展趋势。

从技术角度而言, Android 与 iPhone 相似, 采用 WebKit 浏览器引擎, 具备触摸屏、高级图形显示和上网功能, 用户能够在手机上查收电子邮件、搜索网址和观看视频节目等。Android 手机比 iPhone 等其他手机更强调搜索功能, 界面更强大, 可以说是一种融入了全部 Web 应用的平台。Android 的版本包括 Android 1.1、Android 1.5、Android 1.6, Android 2.0 刚发布不久。随着版本的更新, 从最初的触屏到现在的多点触摸, 从普通的联系人到现在的数据同步, 从简单的 Google Map 到现在的导航系统, 从基本的网页浏览到现在的 HTML 5, 这都说明 Android 已经逐渐稳定, 而且功能越来越强大。此外, Android 平台不仅支持 Java、C、C++ 等主流的编程语言, 还支持 Ruby、Python 等脚本语言, 甚至 Google 专为 Android 的应用开发推出了 Simple 语言, 这使得 Android 有着非常广泛的开发群体。

我们都知道, 无论是产品还是技术, 商业应用是它最大的发展动力。Android 如此受厂商与开发者的青睐, 它的前景一片光明。伴随着装有 Android 操作系统的移动设备的增加, 基于 Android 的应用需求势必也会增加。Android 作为新的平台、新的技术, 国内目前介绍其技术的书籍甚少, 不能满足各个层次的开发者, 为了帮助众多开发人员和爱好者进入移动互联网领域, 并提高程序开发水平, 笔者写作了本书。

本书面向的读者

阅读本书的唯一条件是具有一定的 Java 基础, 当然扩展篇可能会涉及 C、C++ 和脚本语言的知识。

本书面向的读者群包括毫无 Android 开发经验的初学者，以及有一定的 Android 开发经验但缺乏系统学习的开发人员。

如何阅读本书

本书从基础入手，循序渐进地讲述了 Android 的主要功能和用法，使读者对其有完整的认识，掌握其结构框架。同时，从实战的角度出发，通过大量的示例程序，让读者边学习边实践，更深刻地理解 Android 系统的优点所在。

另外本书为每个功能和知识点都提供了一个示例程序，可操作性极强，建议在阅读书本书的同时，一定要结合本书所附带的示例程序（完整的示例程序源代码可登录华章网站 www.hzbook.com 下载）。本书所附的示例程序都是基于最新的 Android 2.0 的 SDK，源代码目录结构如图 1 所示，章节中每一个示例，都可以根据所在的章节及所指定的项目名称在所附源代码中找到对应的项目文件夹。每个项目文件夹都按如图 2 所示的目录结构来存放项目所需的所有源文件。

在安装了 Android 开发环境之后，可以直接将 Android 项目导入到 Eclipse 中，步骤如下：

首先，启动 Eclipse，选择“File”→“Import...”菜单，展开“General”项，选择“Existing Projects into Workspace”导入项目到工作区，如图 3 所示。



图 1 源码结构图



图 2 项目结构图

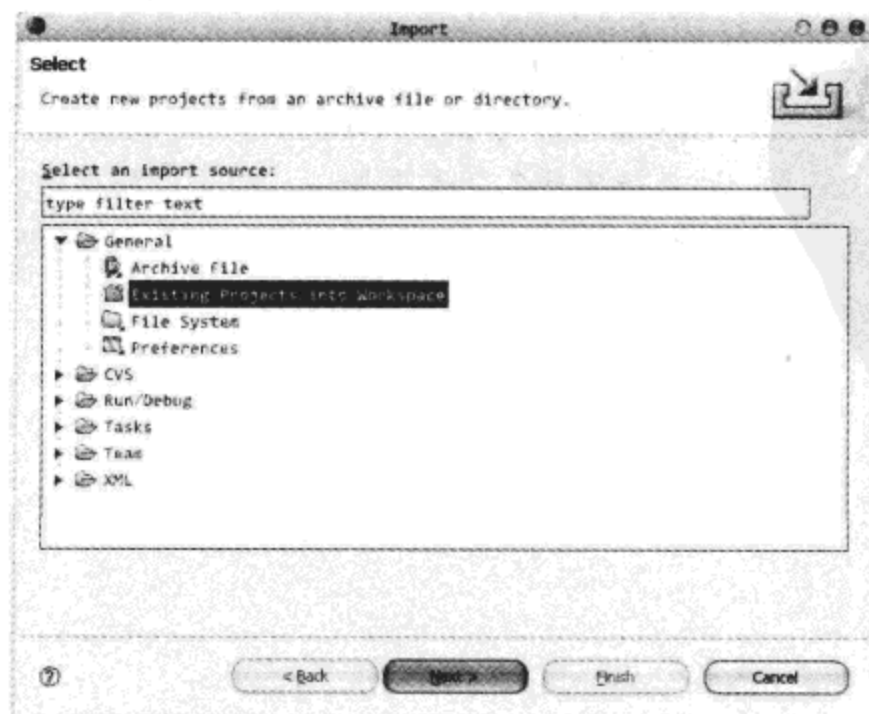


图 3 导入项目到工作区

然后，点击“Next”按钮，进入选择项目文件目录，如图 4 所示，选择好项目目录后，点击“Finish”按钮，等待导入完成即可。如果需要将项目文件一起拷贝到工作区，就需要在图 4 的界面上选择“Copy projects into workspace”复选框。

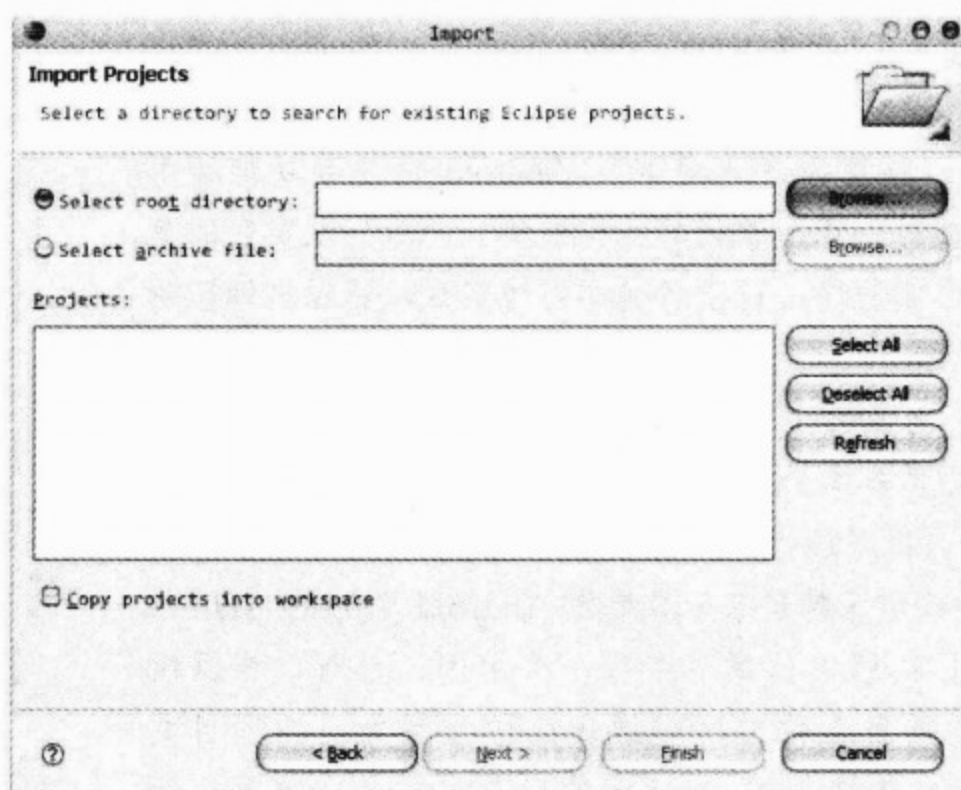


图 4 选择项目文件

致谢

感谢所有在本书写作过程中给予我指导、帮助和鼓励的朋友，尤其是本书的策划编辑杨福川，他不仅对本书提出了宝贵的写作建议，而且还和他的同事曾珊对书稿进行了仔细的审阅。

感谢一直以来信任、鼓励、支持我的父母和亲人。

最后还要感谢我的女友，正是她的爱与支持，才使我有今天的收获。

虽然我们热切地希望与广大读者朋友分享使用 Android 平台的应用开发经验，但由于时间有限，书中难免存在疏漏与错误，诚恳地希望各位读者批评、指正。如果你发现书中有任何问题，抑或是想和本书的作者和读者交流关于 Android 开发中的相关话题，你可以申请加入华章俱乐部^①，这里不仅有技术专家，还有很多志同道合的朋友，大家共同进步吧！

① <http://groups.google.com/group/HZBook-for-readers-and-authors>。

前 言

第一部分 准备篇

第 1 章 Android 开发简介	2
1.1 Android 基本概念	2
1.1.1 Android 简介	2
1.1.2 Android 的系统构架	5
1.1.3 Android 应用程序框架	7
1.2 OMS 介绍	8
1.2.1 OPhone 介绍	8
1.2.2 Widget 介绍	9
1.3 小结	9
第 2 章 Android 开发环境搭建	11
2.1 Android 开发准备工作	11
2.2 开发包及其工具的安装和配置	11
2.2.1 安装 JDK 和配置 Java 开发环境	11
2.2.2 Eclipse 的安装与汉化	12
2.2.3 SDK 和 ADT 的安装和配置	14
2.3 创建第一个 Android 项目——HelloAndroid	16
2.3.1 创建 HelloAndroid 项目	16
2.3.2 运行 HelloAndroid 及模拟器的使用	18
2.3.3 调试 HelloAndroid	22
2.4 小结	25

第二部分 基础篇

第 3 章 Android 程序设计基础 ...	28
3.1 Android 程序框架	28
3.1.1 Android 项目目录结构	28
3.1.2 Android 应用解析	32
3.2 Android 的生命周期	37
3.3 Android 程序 UI 设计	41
3.4 小结	42
第 4 章 用户界面开发	44
4.1 用户界面开发详解	44
4.1.1 用户界面简介	44
4.1.2 事件处理	45
4.2 常用控件应用	49
4.2.1 文本框 (TextView)	49
4.2.2 列表 (ListView)	50
4.2.3 提示 (Toast)	53
4.2.4 编辑框 (EditText)	55
4.2.5 单项选择 (RadioGroup、RadioButton)	56
4.2.6 多项选择 (CheckBox)	59
4.2.7 下拉列表 (Spinner)	62
4.2.8 自动提示 (AutoComplete-TextView)	64
4.2.9 日期和时间 (DatePicker、TimePicker)	65
4.2.10 按钮 (Button)	68
4.2.11 菜单 (Menu)	70
4.2.12 对话框 (Dialog)	72

4.2.13 图片视图 (ImageView) ...	75	5.2.12 获得屏幕属性	142
4.2.14 带图标的按钮 (ImageButton) ...	78	5.3 动画实现	144
4.2.15 拖动效果 (Gallery)	81	5.3.1 Tween 动画	144
4.2.16 切换图片 (ImageSwitcher) ...	83	5.3.2 Frame 动画	149
4.2.17 网格视图 (GridView)	86	5.3.3 GIF 动画播放	153
4.2.18 卷轴视图 (ScrollView)	87	5.4 小结	155
4.2.19 进度条 (ProgressBar)	90	第 6 章 Android 数据存储	156
4.2.20 拖动条 (SeekBar)	93	6.1 Android 数据存储初探	156
4.2.21 状态栏提示 (Notification、 NotificationManager)	95	6.2 数据存储之 Shared Preferences ...	157
4.2.22 对话框中的进度条 (ProgressDialog)	97	6.3 数据存储之 Files	159
4.3 界面布局	100	6.4 数据存储之 Network	162
4.3.1 垂直线性布局	101	6.5 Android 数据库编程	165
4.3.2 水平线性布局	102	6.5.1 SQLite 简介	166
4.3.3 相对布局 (RelativeLayout) ...	103	6.5.2 SQLite 编程详解	166
4.3.4 表单布局 (TableLayout) ...	104	6.5.3 SQLiteOpenHelper 应用 ...	172
4.3.5 切换卡 (TabWidget)	106	6.6 数据共享 (Content Providers) ...	177
4.4 小结	109	6.7 小结	187
第 5 章 Android 游戏开发	110	第 7 章 多媒体开发	188
5.1 Android 游戏开发框架	110	7.1 多媒体开发详解	188
5.1.1 View 类开发框架	110	7.1.1 Open Core	188
5.1.2 SurfaceView 类开发框架 ...	114	7.1.2 MediaPlayer	190
5.2 Graphics 类开发	117	7.1.3 MediaRecorder	192
5.2.1 Paint 和 Color 类介绍	117	7.2 播放音乐	194
5.2.2 Canvas 类介绍	120	7.3 播放视频	201
5.2.3 几何图形绘制	122	7.4 录制歌曲	204
5.2.4 字符串绘制	125	7.5 相机设置	208
5.2.5 图像绘制	126	7.6 闹钟设置	211
5.2.6 图像旋转	129	7.7 铃声设置	214
5.2.7 图像缩放	131	7.8 小结	219
5.2.8 图像像素操作	134	第 8 章 网络与通信	220
5.2.9 Shader 类介绍	137	8.1 网络通信基础	220
5.2.10 双缓冲技术	140	8.1.1 无线网络技术	220
5.2.11 全屏显示	142	8.1.2 Android 网络基础	222

8.2 HTTP 通信	225
8.2.1 HttpURLConnection 接口 ...	225
8.2.2 HttpClient 接口	232
8.2.3 实时更新	235
8.3 Socket 通信	238
8.3.1 Socket 基础	239
8.3.2 Socket 应用 (简易聊天室)	244
8.4 网络通信的中文乱码问题	249
8.5 WebKit 应用	250
8.5.1 WebKit 概述	251
8.5.2 WebView 浏览网页	252
8.5.3 WebView 与 Javascript	259
8.6 WiFi 介绍	261
8.7 蓝牙	266
8.8 小结	276

第 9 章 Android 特色开发

9.1 传感器	277
9.2 语音识别	280
9.3 Google Map	283
9.3.1 Google Map 概述	283
9.3.2 准备工作	285
9.3.3 Google Map API 的使用 ...	287
9.3.4 定位系统	291
9.4 桌面组件	297
9.4.1 快捷方式	297
9.4.2 实时文件夹	299
9.4.3 Widget 开发	301
9.5 账户管理	305
9.6 小结	309

第三部分 实例篇

第 10 章 Android 应用开发

实例	312
10.1 情境模式	312

10.2 文件管理器	317
10.3 通讯录	323
10.4 音乐播放器	330
10.5 天气预报	334
10.6 个人地图	342
10.7 Widget 日历	345
10.8 小结	348

第 11 章 Android 游戏开发实例...

11.1 手机游戏开发简介	349
11.2 游戏框架设计	351
11.3 地图设计	356
11.4 主角设计	358
11.5 图层管理器	363
11.6 游戏音效	367
11.7 游戏存档	369
11.8 小结	372

第四部分 高级篇

第 12 章 Android OpenGL 开发

基础	374
12.1 OpenGL 简介	374
12.2 多边形	378
12.3 颜色	380
12.4 旋转	381
12.5 3D 空间	382
12.6 纹理映射	384
12.7 光照和事件	386
12.8 混合	388
12.9 小结	390

第 13 章 Android OpenGL 综合

应用	391
13.1 移动图像	391
13.2 3D 世界	394

13.3 飘动的旗帜	398	15.1.2 编程规范	452
13.4 显示列表	400	15.2 程序性能测试	454
13.5 雾	402	15.2.1 计算性能测试	455
13.6 粒子系统	403	15.2.2 内存消耗测试	457
13.7 蒙版	407	15.3 初级优化	459
13.8 变形	411	15.4 高级优化	465
13.9 小结	415	15.5 Android 高效开发	468
第 14 章 游戏引擎实现	416	15.6 Android UI 优化	474
14.1 游戏引擎介绍	416	15.7 其他优化	480
14.1.1 什么是引擎	416	15.7.1 zipalign	480
14.1.2 引擎的进化	417	15.7.2 图片优化	481
14.1.3 常见的游戏引擎	417	15.8 小结	482
14.1.4 Android 游戏引擎	418		
14.2 游戏引擎结构	418	第五部分 扩展篇	
14.2.1 游戏引擎原理	418	第 16 章 Android NDK 开发 ...	484
14.2.2 游戏引擎定位	419	16.1 Android NDK 简介	484
14.2.3 游戏引擎框架	420	16.2 安装和配置 NDK 开发环境 ...	485
14.3 游戏引擎设计	420	16.2.1 系统和软件需求	486
14.3.1 游戏引擎结构和功能		16.2.2 NDK 开发环境搭建	487
设计	420	16.2.3 编译第一个 NDK 程序 ...	492
14.3.2 游戏引擎设计注意事项 ...	421	16.3 Android NDK 开发	493
14.4 游戏引擎实现	422	16.3.1 JNI 接口设计	493
14.4.1 Activity 类实现	422	16.3.2 使用 C/C++ 实现本地	
14.4.2 流程控制和线程	424	方法	496
14.4.3 游戏对象与对象管理	427	16.3.3 Android.mk 实现	498
14.4.4 图形引擎	428	16.3.4 Application.mk 实现	501
14.4.5 物理引擎	441	16.3.5 编译 C/C++ 代码	502
14.4.6 事件模块	443	16.4 Android NDK 中使用 OpenGL ...	503
14.4.7 工具模块	445	16.5 小结	506
14.4.8 脚本引擎、音效模块、网络			
模块	446	第 17 章 Android 脚本环境	507
14.5 小结	450	17.1 Android 脚本环境简介	507
第 15 章 优化技术	451	17.2 Android 脚本环境安装	508
15.1 优化的基本知识	451	17.3 如何编写 Android 脚本程序	511
15.1.1 如何书写出优秀代码	451	17.4 小结	515



第 1 章

Android 开发简介

在 Google 及其开放手机联盟推出基于 Linux 平台的开源手机操作系统 Android 之后, Google 又不惜重金举办了 Android 开发者大赛, 吸引了众多开发者的目光。Android 不仅功能强大, 而且具有开放和免费等先天优势, 全球范围内的电信行业、手机制造商因此毫不犹豫地加入到 Android 开放手机联盟中来。2008 年 9 月 22 日, 美国运营商 T-Mobile USA 在纽约正式发布了第一款基于 Android 的手机——T-Mobile G1。这让更多的移动设备厂商看到了 Android 的光明前景, 并纷纷加入其中, Android 甚至已经涉足上网本市场。中国移动也在 Android 的基础之上推出了自己的操作系统 OMS, 而基于 OMS 操作系统的联想 O1 手机也即将上市, 2009 年年底将会有更多的 Android 手机出现。

随着 Android 手机的普及, Android 应用的需求势必会越来越大, 这将是一个潜力巨大的市场, 会吸引无数软件开发厂商和开发者投身其中。作为程序员的我, 当然也不应该落后于人, 赶快加入到 Android 应用的开发阵营中来吧!

1.1 Android 基本概念

Android 一词本意是指“机器人”, 当然现在大家都知道它是 Google 推出的开源手机操作系统。Android 基于 Linux 平台, 由操作系统、中间件、用户界面和应用软件组成, 号称是首个为移动终端打造的真正开放和完整的移动软件。它是由一个由 30 多家科技公司和手机公司组成的“开放手机联盟”共同研发的, 这将大大降低新型手机设备的研发成本。完全整合的全移动功能性产品成为“开放手机联盟”的最终目标。

1.1.1 Android 简介

Android 作为 Google 移动互联网战略的重要组成部分, 将进一步推进“随时随地为每个人提供信息”这一企业目标的实现。Google 的目标是让移动通信不依赖于设备, 甚至是平台。出于这个目的, Android 将完善而不是替代 Google 长期以来推行的移动发展战略: 通过与全球各地的手机制造商和移动运营商成为合作伙伴, 开发既实用又有吸引力的移动服务, 并推广这些产品。

Android 平台的研发队伍阵容强大, 包括 Google、HTC (宏达电)、T-Mobile、高通、摩托罗拉、三星、LG 以及中国移动在内的 30 多家企业都将基于该平台开发手机的新型业务, 应用之间的通用性和互联性将在最大程度上得到保持。“开放手机联盟”表示,

Android 平台可以促使移动设备的创新, 让用户体验到最优质的移动服务。同时, 开发商也将得到一



图 1-1 Android G1

个新的开放级别，更方便地进行协同合作，从而保障新型移动设备的研发速度。因此 Android 是第一个完整、开放、免费的手机平台。下面我们来欣赏一下第一款基于 Android 操作系统的手机 G1，外观相当漂亮，如图 1-1 所示。

Android 系统具有如下 5 个特点：

- ❑ 开放性。Google 与开放手机联盟合作开发了 Android，Google 通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作伙伴关系，希望通过建立标准化、开放式的移动电话软件平台，在移动产业内形成一个开放式的生态系统。
- ❑ 应用程序无界限。Android 上的应用程序可以通过标准 API 访问核心移动设备功能。通过互联网，应用程序可以声明它们的功能可供其他应用程序使用。
- ❑ 应用程序是在平等的条件下创建的。移动设备上的应用程序可以被替换或扩展，即使是拨号程序或主屏幕这样的核心组件。
- ❑ 应用程序可以轻松地嵌入网络。应用程序可以轻松地嵌入 HTML、JavaScript 和样式表，还可以通过 WebView 显示网络内容。
- ❑ 应用程序可以并行运行。Android 是一种完整的多任务环境，应用程序可以在其中并行运行。在后台运行时，应用程序可以生成通知以引起注意。

为什么 Android 手机如此受用户青睐，下面我们来看看 Android 究竟有些什么功能在吸引着我们。

(1) 智能虚拟键盘。虚拟键盘的出现意味着基于 Android 1.5 或以上版本 (Android 2.0) 的移动设备可以同时支持物理键盘和虚拟键盘。不同的输入方式可满足用户在特定场景的需求。Android 虚拟键盘可以在任何应用中提供，包括 Gmail、浏览器、SMS，当然也包括大量的第三方应用，如自动校正、推荐、用户词典等。不同于其他手机平台，Android 1.5 及其以上的版本还支持第三方虚拟键盘应用的安装，如图 1-2 所示。

(2) 使用 Widget 实现桌面个性化。可以用 Widget “武装”自己的桌面。大多数小的 Web 应用都是从网络上获得实时数据并展示给用户的。Android 预装了 5 个桌面 Widget，包括数字时钟、日历、音乐播放器、相框和搜索。不同于 iPhone，Android 通过内置的应用程序库安装第三方 Widget，如图 1-3 所示。



图 1-2 虚拟键盘



图 1-3 用 Widget 实现个性化桌面

(3) 用在线文件夹快速浏览在线数据。类似于 OS X Leopard 的 QuickLook 特征, Android 的在线文件夹可显示常见的数据条目, 比如联系人、喜欢的应用、E-mail 信息、播放列表、书签、RSS 源等, 并不需要运行系统程序处理特定的数据条目。在线文件夹数据实时更新, 就像通过云或是本地创建新的数据。什么是最好的, 开发者可以拓展通用数据条目和注册新数据类型的内置支持。例如, Twitter 客户端程序可以注册 tweet 作为新数据类型, 因此可以让你从你的朋友那里创建 tweet 的在线文件。Android 可以为我们的个人桌面提供一组在线文件夹, 从而帮助我们快速、方便地浏览联系人、股市、书签等信息。

(4) 视频录制和分享。Android 还有录制和分享视频的功能, 对回放和 MPEG-4、3GP 等视频格式也有了更好的支持。可以通过 E-mail、MMS 或直接上传到 YouTube 等方式来分享视频, 使用隐私控制来决定是分享给朋友还是每个人。上传视频的同时, 可以继续使用手机, 甚至可以继续录制和上传新的视频。如图 1-4 所示, 通过 YouTube 分享录制的视频。

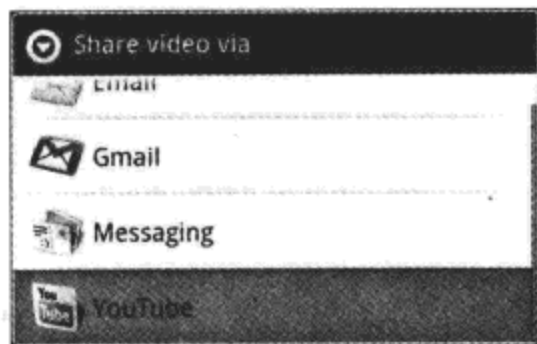


图 1-4 通过 YouTube 分享录制的视频

(5) 图片上传。在线分享图片需要的点击更少。完成照相后, 当浏览图片或选择 Google 在线图片服务 Picasa 时, 只需轻点“分享”就会拥有 1GB 的免费图片存储空间。

(6) 更快、更兼容的浏览器。Android 的基于 Webkit 内核的浏览器带来了重要的调速装置 (SpeedPumb), 这得益于新的 Webkit 渲染引擎和优化的 Java 脚本编译器 (SquirrelFish)。当使用包含大量 Java 脚本的复杂 Web 应用时, 可以体验到更佳的性能。除提高速度外, Android 的浏览器还支持 Web 页面内的复制和粘贴操作, 用户可以选中文本并复制, 然后粘贴到搜索框中进行搜索。

(7) Voice Search 语音搜索。带有语音识别技术的 Google 手机已于 2008 年 11 月面世, 它支持语音搜索功能。该功能增强了默认搜索能力, 已超过纯文本搜索。当你大声说出要搜索的内容后, Android 将上传数字信号并记录到 Google 服务器中。在服务器中, 语音识别技术能将语音转化为特定的文本搜索, 使之通过 Google 搜索引擎, 通过地理位置的筛选, 将结果反馈到手机设备。图 1-5 显示了 Google 文本和语音搜索桌面。



图 1-5 Google 文本和语音搜索桌面

(8) 立体声蓝牙和免提电话。除了增强的免提电话体验, Android 还支持立体声蓝牙 (A2DP 和 AVCRP), 并有自动配对功能。

(9) 强大的 GPS 技术。Android 内部提供了大量 GPS 组件, 我们可以很轻松地获得设备当前的位置等信息, 让导航等功能更加完美。

(10) Android 系统硬件检测。Android 可自动检测和修复 SD 卡的文件系统, 允许第三方应用显示 Android 系统的硬件特征。为了让用户下载到与自己的设备更匹配的应用, 我们可以检测用户设备的硬件信息, 让满足应用要求的设备安装该程序, 当更多的 Android 设备建立在不同的硬件上时, 这个功能会显得很实用。

1.1.2 Android 的系统构架

通过上一节的介绍，我们对 Android 的特点以及它为什么会如此受欢迎有了初步的了解。下面将讨论 Android 的系统架构，我们先来看看 Android 的体系结构，如图 1-6 所示。

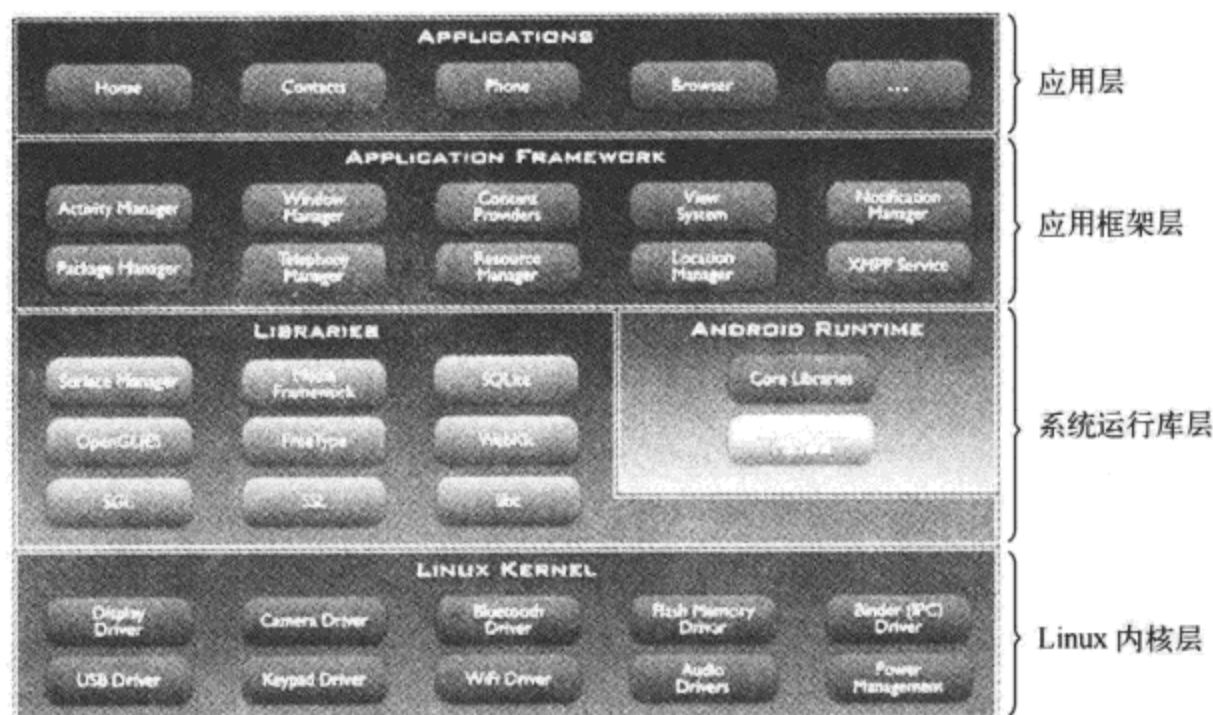


图 1-6 Android 系统结构图

从图 1-6 可以看出 Android 分为 4 层，从高到底分别是应用层、应用框架层、系统运行库层和 Linux 内核层。下面将对这 4 层进行简要的分析和介绍。

1. 应用层

应用是用 Java 语言编写的运行在虚拟机上的程序，如图 1-6 中最上层部分所示。其实，Google 最开始时就在 Android 系统中捆绑了一些核心应用，比如 E-mail 客户端、SMS 短消息程序、日历、地图、浏览器、联系人管理程序，等等。

2. 应用框架层

这一层是编写 Google 发布的核心应用时所使用的 API 框架，开发人员同样可以使用这些框架来开发自己的应用，这样便简化了程序开发的架构设计，但是必须遵守其框架的开发原则。

从图 1-6 中可以看出，Android 提供了如下一些组件。

- ❑ 丰富而又可扩展的视图 (View): 可以用来构建应用程序，它包括列表 (List)、网格 (Grid)、文本框 (Text Box)、按钮 (Button)，以及可嵌入的 Web 浏览器。
- ❑ 内容提供者 (Content Providers): 它可以让一个应用访问另一个应用的数据 (如联系人数据库)，或共享它们自己的数据。
- ❑ 资源管理器 (Resource Manager): 提供非代码资源的访问，如本地字符串、图形和布局文件 (Layout file)。
- ❑ 通知管理器 (Notification Manager): 应用可以在状态栏中显示自定义的提示信息。
- ❑ 活动管理器 (Activity Manager): 用来管理应用程序生命周期并提供常用的导航退回功能。
- ❑ 窗口管理器 (Window Manager): 管理所有的窗口程序。

❑ 包管理器 (Package Manager): Android 系统内的程序管理。

后面的章节将进一步介绍这些组件的使用。

3. 系统运行库 (C/C++库以及 Android 运行库) 层

当使用 Android 应用框架时, Android 系统会通过一些 C/C++库来支持我们使用的各个组件, 使其能更好地为我们服务。

❑ Bionic 系统 C 库: C 语言标准库, 系统最底层的库, C 库通过 Linux 系统来调用。

❑ 多媒体库 (MediaFramework): Android 系统多媒体库, 基于 PacketVideo OpenCORE, 该库支持多种常见格式的音频、视频的回放和录制, 以及图片, 比如 MPEG4、MP3、AAC、AMR、JPG、PNG 等。

❑ SGL: 2D 图形引擎库。

❑ SSL: 位于 TCP/IP 协议与各种应用层协议之间, 为数据通信提供支持。

❑ OpenGL ES 1.0: 3D 效果的支持。

❑ SQLite: 关系数据库。

❑ Webkit: Web 浏览器引擎。

❑ FreeType: 位图 (bitmap) 及矢量 (vector)。

每个 Java 程序都运行在 Dalvik 虚拟机之上。与 PC 一样, 每个 Android 应用程序都有自己的进程, Dalvik 虚拟机只执行 .dex 的可执行文件。当 Java 程序通过编译, 最后还需要通过 SDK 中的 dx 工具转化成 .dex 格式才能正常在虚拟机上执行。

Google 于 2007 年底正式发布了 Android SDK, 作为 Android 系统的重要特性, Dalvik 虚拟机也第一次进入了人们的视野。它对内存的高效使用, 以及在低速 CPU 上表现出的高性能, 确实令人刮目相看。Android 系统可以简单地完成进程隔离和线程管理。每一个 Android 应用在底层都会对应一个独立的 Dalvik 虚拟机实例, 其代码在虚拟机的解释下得以执行。

很多人认为 Dalvik 虚拟机是一个 Java 虚拟机, 因为 Android 的编程语言恰恰就是 Java 语言。但是这种说法并不准确, 因为 Dalvik 虚拟机并不是按照 Java 虚拟机的规范来实现的, 两者并不兼容。它们有两个明显的不同: Java 虚拟机运行的是 Java 字节码, 而 Dalvik 虚拟机运行的则是其专有的文件格式为 dex (Dalvik Executable) 的文件。在 Java SE 程序中的 Java 类会被编译成一个或者多个字节码文件 (.class) 然后打包到 jar 文件, 而后 Java 虚拟机会从相应的 class 文件和 jar 文件中获取相应的字节码; Android 应用虽然也是使用 Java 语言进行编程, 但是在编译成 class 文件后, 还会通过一个工具 (dx) 将应用所有的 class 文件转换成一个 dex 文件, 而后 Dalvik 虚拟机会从其中读取指令和数据。

Dalvik 虚拟机非常适合在移动终端上使用, 相对于在桌面系统和服务器系统运行的虚拟机而言, 它不需要很快的 CPU 计算速度和大量的内存空间。根据 Google 的测算, 64MB 的内存已经能够让系统正常运转了。其中 24MB 被用于底层系统的初始化和启动, 另外 20MB 被用于启动高层服务。当然, 随着系统服务的增多和应用功能的扩展, 其所消耗的内存也势必越来越大。归纳起来, Dalvik 虚拟机有如下几个主要特征:

(1) 专有的 dex 文件格式。dex 是 Dalvik 虚拟机专用的文件格式, 而为什么弃用已有的字节码文件 (.class 文件) 而采用新的格式呢? 原因如下:

❑ 每个应用中会定义很多类, 编译完成后即会有很多相应的 class 文件, class 文件中会有大量

冗余信息，而 dex 文件格式会把所有的 class 文件内容整合到一个文件中。这样，除了减少整体的文件尺寸和 I/O 操作外，也提高了类的查找速度。

- ❑ 增加了对新的操作码的支持。
- ❑ 文件结构尽量简洁，使用等长的指令，借以提高解析速度。
- ❑ 尽量扩大只读结构的大小，借以提高跨进程的数据共享。

(2) dex 的优化。dex 文件的结构是紧凑的，但是如果还想运行时的性能有进一步提高，就需要对 dex 文件进一步优化。优化主要针对以下几个方面：

- ❑ 调整所有字段的字节序 (LITTLE_ENDIAN) 和对齐结构中的每一个域。
- ❑ 验证 DEX 文件中的所有类。
- ❑ 对一些特定的类和方法里的操作码进行优化。

(3) 基于寄存器。相对于基于堆栈实现的虚拟机，基于寄存器实现的虚拟机虽然在硬件、通用性上要差一些，但是它在代码的执行效率上却更胜一筹。

(4) 一个应用，一个虚拟机实例，一个进程。每一个 Android 应用都运行在一个 Dalvik 虚拟机实例中，而每一个虚拟机实例都是一个独立的进程空间。虚拟机的线程机制、内存分配和管理、Mutex 等的实现都依赖底层操作系统。所有 Android 应用的线程都对应一个 Linux 线程，虚拟机因而可以更多地依赖操作系统的线程调度和管理机制。不同的应用在不同的进程空间里运行，对不同来源的应用都使用不同的 Linux 用户来运行，可以最大程度地保护应用的安全和独立运行。

4. Linux 内核层

Android 的核心系统服务基于 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等都依赖于该内核。Linux 内核同时也作为硬件和软件栈之间的抽象层。

Android 更多的是需要一些与移动设备相关的驱动程序，主要的驱动如下所示。

- ❑ 显示驱动 (Display Driver)：基于 Linux 的帧缓冲 (Frame Buffer) 驱动。
- ❑ 键盘驱动 (KeyBoard Driver)：作为输入设备的键盘驱动。
- ❑ Flash 内存驱动 (Flash Memory Driver)：基于 MTD 的 Flash 驱动程序。
- ❑ 照相机驱动 (Camera Driver)：常用的基于 Linux 的 v4l2 (Video for Linux) 驱动。
- ❑ 音频驱动 (Audio Driver)：常用的基于 ALSA (Advanced Linux Sound Architecture) 的高级 Linux 声音体系驱动。
- ❑ 蓝牙驱动 (Bluetooth Driver)：基于 IEEE 802.15.1 标准的无线传输技术。
- ❑ WiFi 驱动：基于 IEEE 802.11 标准的驱动程序。
- ❑ Binder IPC 驱动：Android 的一个特殊的驱动程序，具有单独的设备节点，提供进程间通信的功能。
- ❑ Power Management (电源管理)：比如电池电量等。

1.1.3 Android 应用程序框架

上一节我们对 Android 的系统构架进行了详细剖析，Android 分为应用层、应用框架层、系统运行库层和 Linux 内核层。我们在开发应用时都是通过框架来与 Android 底层进行交互，接触最多的就是应用框架层了。

什么是应用程序框架呢？框架可以说是一个应用程序的核心，是所有参与开发的程序员共同使用和遵守的约定，大家在其约定上进行必要的扩展，但程序始终保持主体结构的一致性。其作用是让程序保持清晰和一目了然，在满足不同需求的同时又不互相影响。

Android 系统提供给应用开发者的本身就是一个框架，所有的应用开发都必须遵守这个框架的原则。我们在开发应用时就是在这个框架上进行扩展，下面来看看 Android 这个框架都有些什么功能可供我们使用。

- ❑ **android.app**: 提供高层的程序模型和基本的运行环境。
- ❑ **android.content**: 包含对各种设备上的数据进行访问和发布。
- ❑ **android.database**: 通过内容提供者浏览和操作数据库。
- ❑ **android.graphics**: 底层的图形库，包含画布、颜色过滤、点、矩形，可以将它们直接绘制到屏幕上。
- ❑ **android.location**: 定位和相关服务的类。
- ❑ **android.media**: 提供一些类管理多种音频、视频的媒体接口。
- ❑ **android.net**: 提供帮助网络访问的类，超过通常的 **java.net.*** 接口。
- ❑ **android.os**: 提供了系统服务、消息传输和 IPC 机制。
- ❑ **android.opengl**: 提供 OpenGL 的工具。
- ❑ **android.provider**: 提供访问 Android 内容提供者的类。
- ❑ **android.telephony**: 提供与拨打电话相关的 API 交互。
- ❑ **android.view**: 提供基础的用户界面接口框架。
- ❑ **android.util**: 涉及工具性的方法，例如时间日期的操作。
- ❑ **android.webkit**: 默认浏览器操作接口。
- ❑ **android.widget**: 包含各种 UI 元素（大部分是可见的）在应用程序的布局中使用。

1.2 OMS 介绍

* OMS 是 Open Mobile System 的简称，即面向移动互联网的开放型移动智能终端软件平台，它包括基于 Linux 2.6 内核的移动终端下层操作系统、上层应用软件、中间件、Java 虚拟机、硬件参考设计以及基于 WebKit 的各类应用。它具有强大的兼容性、扩展性和安全性，以及简单易用、友好的人机界面等，而且具有完全自主的知识产权。在此之上，OMS 拥有开放统一的 API 开发接口、完备的集成开发环境和活跃的在线生态环境，极大地方便了移动应用的开发。

OMS 的可移植性将使该软件平台在其他领域具有广泛的应用，如航空航天、军事、制造业等。

1.2.1 OPhone 介绍

OPhone 是基于 Linux 的面向移动互联网的终端基础软件及系统解决方案。由于 OPhone 与 Android 兼容，都是基于 Java 开发的，因此可以同时用 OMS API 和 Android API 来开发 OMS 应用。任何用 Android API 开发的应用都可以在 OMS 终端上正确地运行。然而，不能在 Android 终端

上运行由扩展的 OMS API 开发的程序, 因为这些 OMS API 是 OMS 平台独有的, 而且在运行时是必需的。

OPhone 是指采用了 OMS 智能操作系统的手机。为了突破 TD 终端瓶颈, 以及促进手机终端与中国移动的网络和应用服务进行无缝对接, 中国移动在 Android 操作系统基础上自主开发了 OMS 系统, 该系统直接内置了中国移动的服务菜单、音乐随身听、手机导航、号簿管家、139 邮箱、飞信、快讯和移动梦网等特色业务, 如图 1-7 所示。

1.2.2 Widget 介绍

OMS 除了支持基于 Java 的应用, 还支持 Widget 应用开发。Widget 应用是 OMS 的精华, 而 Android 从 1.5 版本开始同样支持 Widget 应用开发, 但是所采用的标准则和 OMS 不同, 我们会在后面的章节详细讲解。

Widget 应用采用了 JIL (Joint Innovation Lab) Widget 标准。JIL Widget 是一个采用 HTML、JavaScript 和 CSS 等网络技术的应用程序。Widget 应用是在 Widget 引擎上运行的独立的应用程序。Widget 已经成为手机上非常流行的技术, 可以为用户带来良好的移动互联网体验, 随时随地获取有用的资讯, 如天气预报、股票信息、头条新闻等。从用户的角度来看, Widget 应用和 OPhone 应用没有什么区别。实际上, Widget 应用不同于 OPhone 应用。OPhone 应用是采用 Java 技术的应用程序, 而 Widget 应用则是采用 HTML、JavaScript 和 CSS 等网络技术的应用程序。相比较而言, Widget 应用的开发更加方便快捷。此外, JIL Widget 还提供了许多 JavaScript API 来扩展 Widget 应用的能力, 如访问手机电话本、手机文件系统等。Widget 应用运行效果如图 1-8 所示。



图 1-7 OPhone 系统界面



图 1-8 HelloWidget 效果预览

1.3 小结

本章主要介绍了与 Android 相关的一些基本概念, 同时分析了 Android 系统的特点及其功能,

简单介绍了目前主流的 7 个 Linux 平台手机以及中国移动的 OMS 操作系统。在介绍 Android 基本概念时重点介绍了 Android 系统架构和应用框架，其中应用层即是我们使用 Java 语言编写的一些运行在虚拟机上的程序，应用框架层是我们开发应用时接触最为紧密的一层，在开发应用程序时必须遵守其规则，才能保证所开发的应用程序能在 Android 上安全地运行。大家应着重理解这两层，这样才能开发出效率更高的应用程序。

据称今年底将会有 18 款之多的以 Android 平台为系统的 Linux 手机上市，Android 将在移动开发中具有更广阔的前景。要想成为一个优秀的 Android 手机开发者，还需要从基础做起，希望大家好好掌握本章的内容。



Android 开发环境搭建

本章讲解如何配置 Android 开发环境首先介绍 Android 开发所需要的开发包和工具，以及获得它们的方式；其次介绍如何正确安装和配置这些开发包；最后，为了测试安装的开发环境，创建了第一个 Android 项目——HelloAndroid，然后在模拟器上运行和调试该程序，并将该应用程序安装到 Android 手机上。

2.1 Android 开发准备工作

配置 Android 开发环境之前，首先需要了解 Android 对操作系统的要求。它可以使用 Windows XP 及其以上版本、Mac OS、Linux 等操作系统，本书以 Windows XP 为例进行讲解。Android 开发所需软件的版本及其下载地址如表 2-1 所示。

表 2-1 Android 开发所需软件的版本及其下载地址

软件名称	所用版本	下载地址
JDK	1.6	http://java.sun.com
Eclipse	Ganymede(3.4)	http://www.eclipse.org
Android SDK	Android SDK 2.0	http://developer.android.com/sdk/index.html
ADT	0.9.4	https://dl-ssl.google.com/android/eclipse/

2.2 开发包及其工具的安装和配置

Android 以 Java 作为开发语言，JDK 是进行 Java 开发时必需的开发包。Eclipse 是一款非常优秀的开源 IDE，在大量插件的“配合”下，完全可以满足从企业级 Java 应用到手机终端 Java 游戏的开发。Google 官方也提供了基于 Eclipse 的 Android 开发插件 ADT，所以本书选择 Eclipse 作为开发 IDE。

2.2.1 安装 JDK 和配置 Java 开发环境

很多人不能够很好地进行 Java 开发，原因就在于对 Java 运行环境不了解或是了解得不够透彻。如果连一个普通的 Java 程序运行环境都搭建不好，就更不要说理解 J2EE、J2ME 以及本书所讲的 Android 等的运行环境了。因此，这里我们先讲如何安装 JDK 以及 Java 环境的配置，教大家搭建一个学习 Java 的基础平台，让大家少走一些弯路，多学到一些小窍门。

(1) 登录 <http://java.sun.com>, 下载最新版 JDK。

(2) 安装 JDK, 安装包中包含了 JDK 和 JRE 两部分, 笔者建议将它们安装在同一个盘符下。双击安装程序, 选择安装的目录, 点击“下一步”, 等待安装程序自动完成安装即可。

(3) 右键单击“我的电脑”, 选择“属性”菜单项, 选择“高级”选项卡, 选择“环境变量”, 找到“Path”变量名(如果没有就新建一个名为“Path”的变量), 点击“编辑”按钮, 添加 JDK 安装目录中“bin”文件夹路径, 如图 2-1 所示。然后点击“确定”按钮完成。再找到“CLASSPATH”变量(如果没有, 同样可以新建), 输入 JDK 安装目录中“lib”以及“demo”的路径, 如图 2-2 所示, 单击“确定”按钮完成。

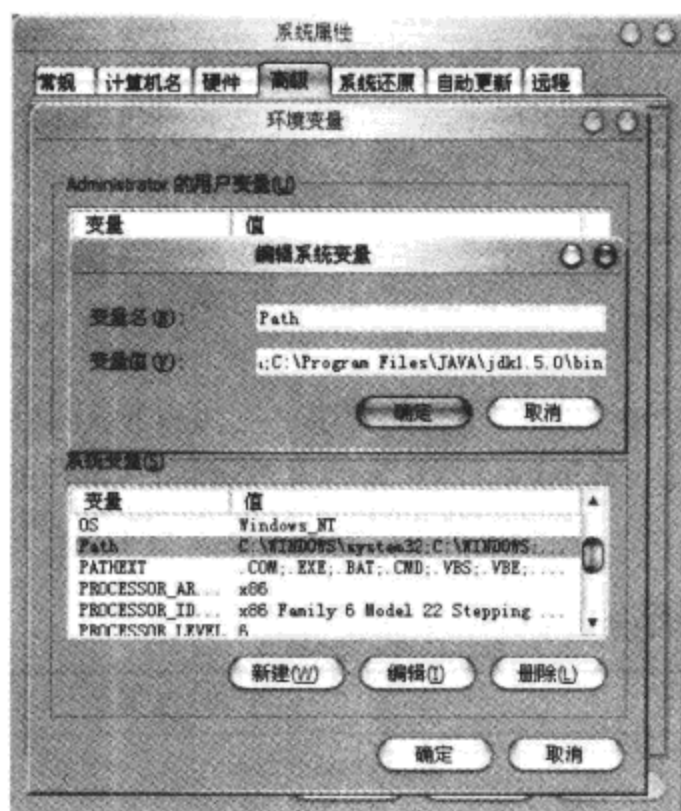


图 2-1 “Path”变量配置

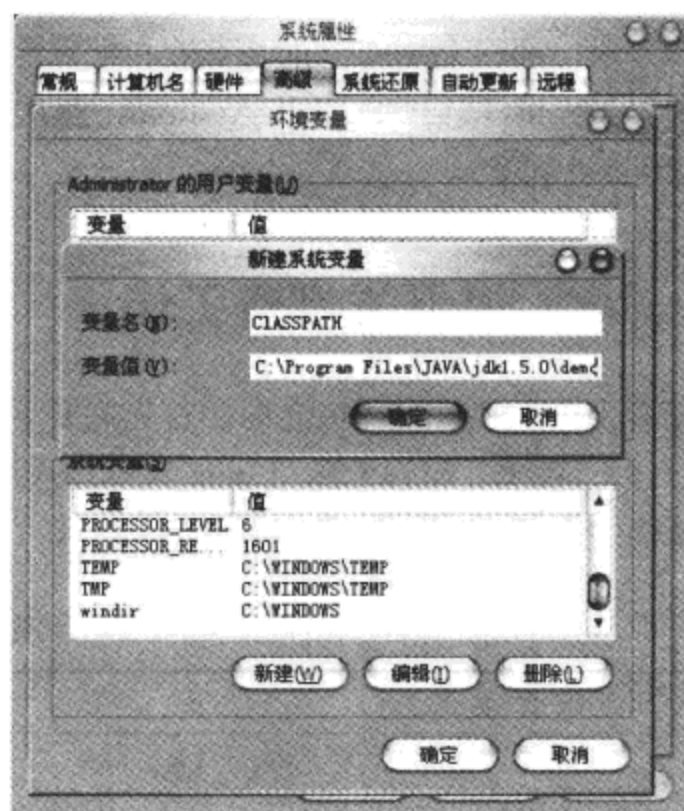


图 2-2 “CLASSPATH”变量配置

(4) 安装配置完成之后, 要测试是否安装成功。点击开始→运行, 输入“CMD”, 打开命令行模式。键入命令“java -version”, 检测 JDK 是否安装成功, 如果运行结果如图 2-3 所示, 即表示安装成功。

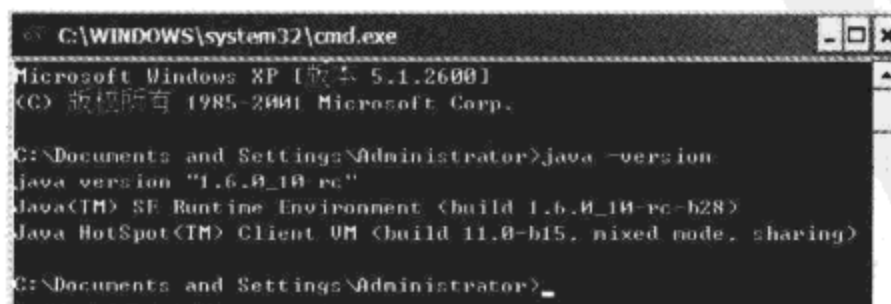


图 2-3 “java -version”测试命令

2.2.2 Eclipse 的安装与汉化

Eclipse 的安装非常简单, 直接将下载的压缩包解压即可。老版本的 Eclipse 的多国语言项目只

更新到 3.2.1 版本，以后就再也没有更新了。Eclipse 最近发布了一个名为 Babel project 的项目，这个项目就是用来解决国际化的问题，旨在为每一个插件提供独立的语言包。这样，当做 RCP 项目的时候，根据需要对语言进行打包即可！

Babel 的安装方法和步骤如下所示：

(1) 启动 Eclipse 开发工具，依次点击“Help”→选择“Software Update...”菜单命令，打开“Software Updates and Add-ons”对话框，选择“Avaliable Software”项。接着点击“Add Site...”按钮，在“Location”文本框中输入 Babel 更新地址：<http://download.eclipse.org/technology/babel/update-site/ganymede>，然后点击 OK 按钮，如图 2-4 所示。

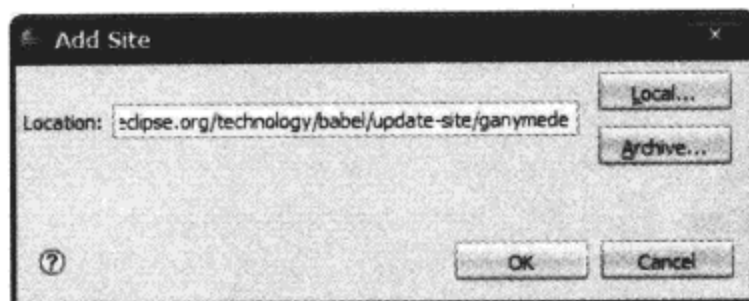


图 2-4 添加语言包更新地址

(2) “Avaliable Software”表中会多出一项 <http://download.eclipse.org/technology/babel/update-site/ganymede/>，点击该项左边的箭头，就会出现网络更新软件列表，如图 2-5 所示。

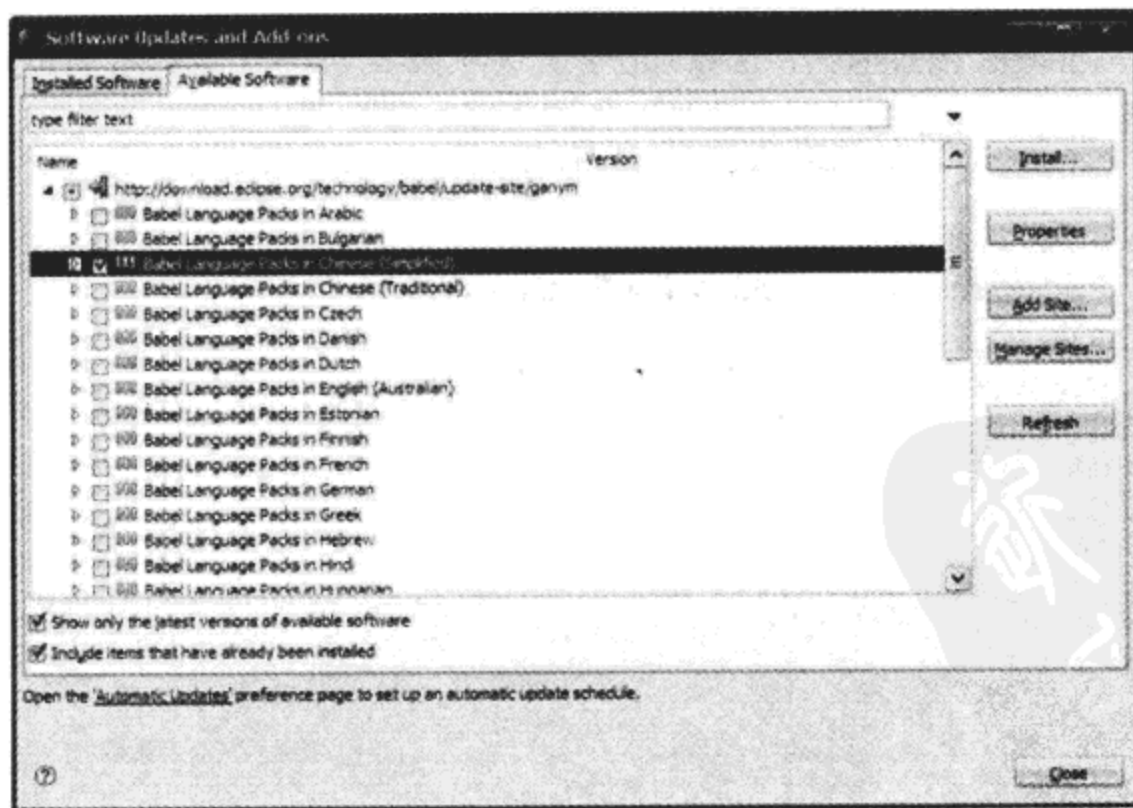


图 2-5 Avaliable Software 选择框

(3) 选择“Simplified Chinese”语言包后，点击“Install...”按钮，等待 Eclipse 处理。

处理完成后会出现“Install”对话框，这时会提示你选择要安装的语言包。根据提示，很容易完成后面的操作，这里就不再赘述了。

安装完毕后，重新启动 Eclipse 即可完成全部汉化过程。

如果重启 Eclipse 后不显示中文, 请用命令行 “eclipse.exe -nl zh_CN” 重新启动 Eclipse。

2.2.3 SDK 和 ADT 的安装和配置

安装了 JDK 和 Eclipse 后, 现在就要安装 Android SDK 和 ADT 插件了。

1. Android SDK 安装

(1) 解压缩下载好的 SDK 安装包到要安装 SDK 的路径, 然后运行 “SDK Setup.exe”。

(2) 如果遇到了消息为 “Failed to fetch URL...” 的错误提示, 那么需要将 HTTPS 方式改为 HTTP 方式, 在 “Android SDK and AVD Manager” 窗口的左侧选择 “Settings”, 选中 “Force https://... sources to be fetched using http://...” 选项 (如图 2-6 所示), 点击 “Save & Apply” 并重新运行 SDK Setup.exe。

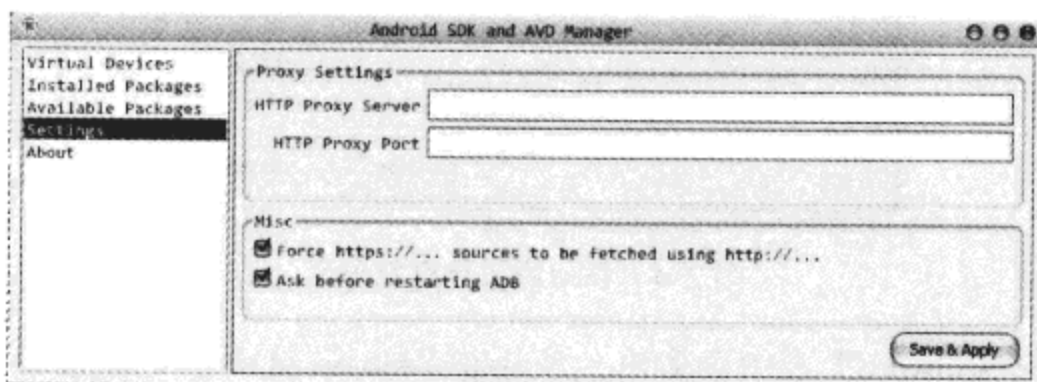


图 2-6 更改 HTTP 方式

(3) 点击 “Available Packages”, 选择要安装的 API 版本及 USB 驱动和 SDK 文档, 如图 2-7 所示。这里为了测试方便, 所以全部选择了。



图 2-7 选择 API 版本

(4) 选择好之后点击 “Install Selected” 按钮, 安装选中的软件包, 在接下来出现的界面中依次点击 “Accept All” 单选按钮和 “Install Accepted” 按钮, 开始下载所选择的安装包。

下载完成之后, 根据提示即可完成后续的安装操作。

到这里, 我们就完成了 Android SDK 的安装, 下面来配置 Android SDK。

2. Android SDK 配置

需要将 Android SDK 安装目录中的 tools 文件夹路径添加到环境变量中以便使用, 操作步骤如下:

(1) 右键点击“我的电脑”, 依次选择“属性”→“高级”→“环境变量”选项, 如图 2-8 所示。

(2) 选择“系统变量”中变量名为“Path”的项, 点击“编辑”按钮, 将 Android SDK 安装文件夹下的 tools 文件夹的路径加入到“Path”变量中, 注意用“\”隔开, 如图 2-9 所示。



图 2-8 环境变量

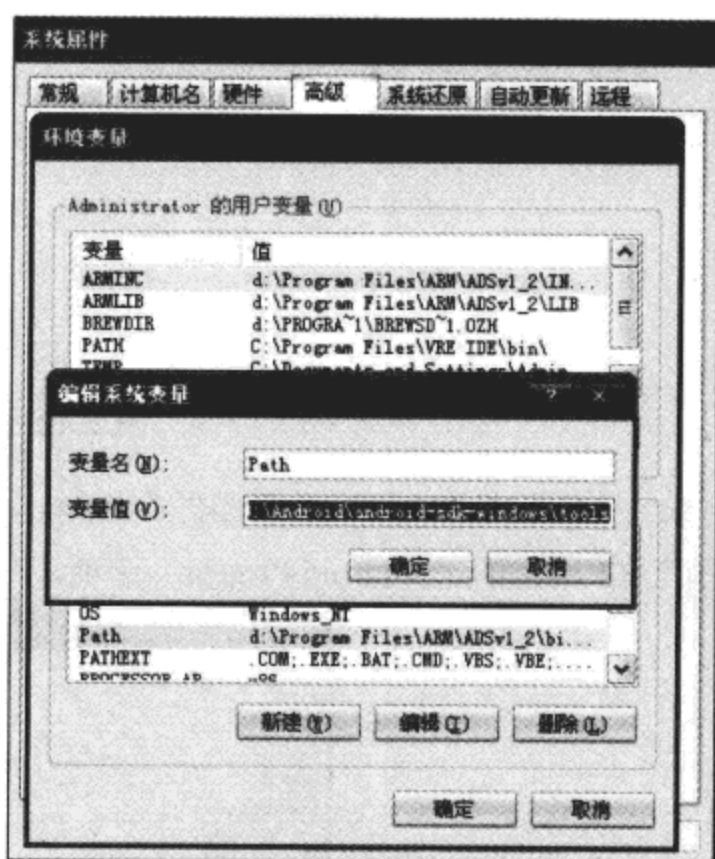


图 2-9 编辑系统环境变量

(3) 依次点击“确定”，完成环境变量配置。

3. 安装和配置 ADT

下面我们来安装和配置 ADT 插件, 步骤如下:

(1) 启动 Eclipse, 点击“Help”菜单, 依次选择“Software Update...”项和“Available Software”选项卡, 点击“Add Site...”按钮, 输入地址 <https://dl-ssl.google.com/android/eclipse/>, 结果如图 2-10 所示。

(2) 点击“OK”, 这时可能会出现如图 2-11 所示的错误。

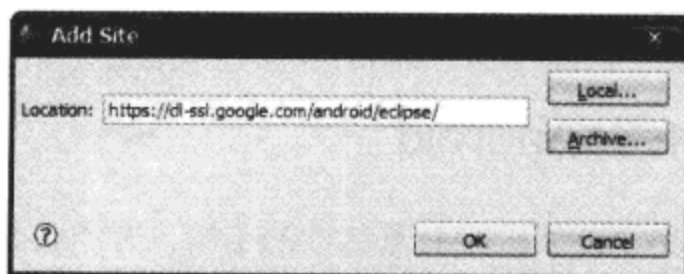


图 2-10 添加 ADT 的更新地址

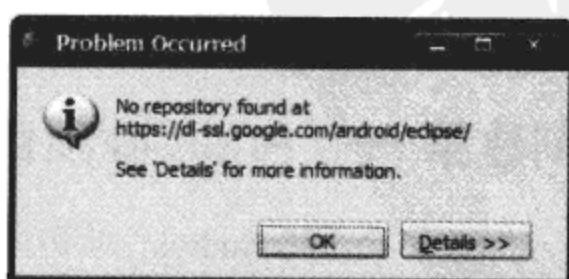


图 2-11 更新地址错误

解决这个问题的方法是: 将“<https://dl-ssl.google.com/android/eclipse/>”中的“https”更改为“http”, 在接下来的对话框中选中“Name”下的所有选项, 根据提示即可完成后续的安装过程。

(3) 打开菜单“Windows”，依次选择“Preferences”→“Android”，点击“Browse...”按钮，选择 Android SDK 的安装路径，如图 2-12 所示。



图 2-12 Eclipse 首选项

(4) 点击“OK”按钮，打开菜单“File”，依次选择“NEW”→“Project...”菜单命令，出现如图 2-13 所示的“Android Project”选项，则表示安装配置成功。

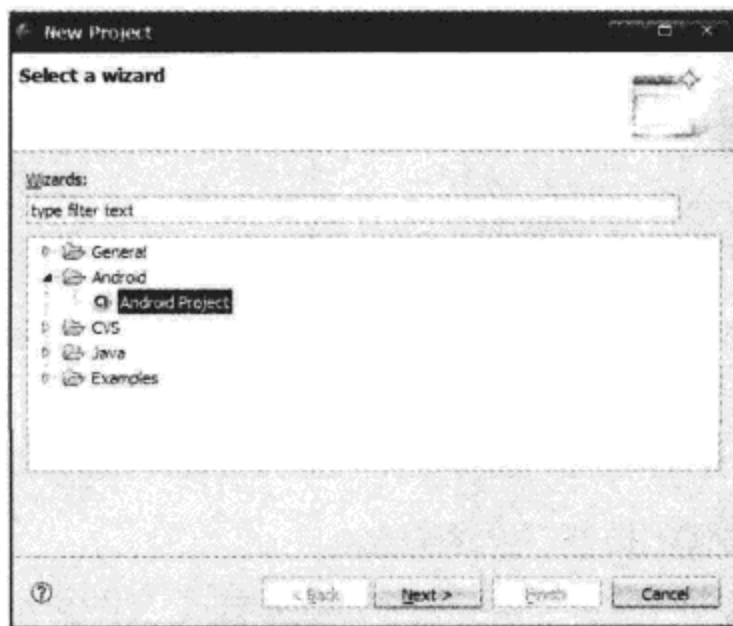


图 2-13 新建工程界面

到这里，我们的准备工作已经就绪，可以在 Android 平台上开发我们的应用了，很心动吧！神奇的 Android 之旅即将开始。

2.3 创建第一个 Android 项目——HelloAndroid

为了便于第一次开发 Android 应用的朋友能对整个开发过程有系统性的了解，并能亲自动手创建自己的应用，我们特在本书的开篇准备了一个简单的实例项目——HelloAndroid。

2.3.1 创建 HelloAndroid 项目

ADT 提供了简单的生成 Android 应用框架的功能，我们现在使用 ADT 通过 Eclipse 创建一个

Android 工程，其步骤如下。

(1) 打开 Eclipse 开发工具，新建一个项目，在弹出的“New Project”对话框的列表中展开“Android”项，然后选择“Android Project”子项，如图 2-14 所示。

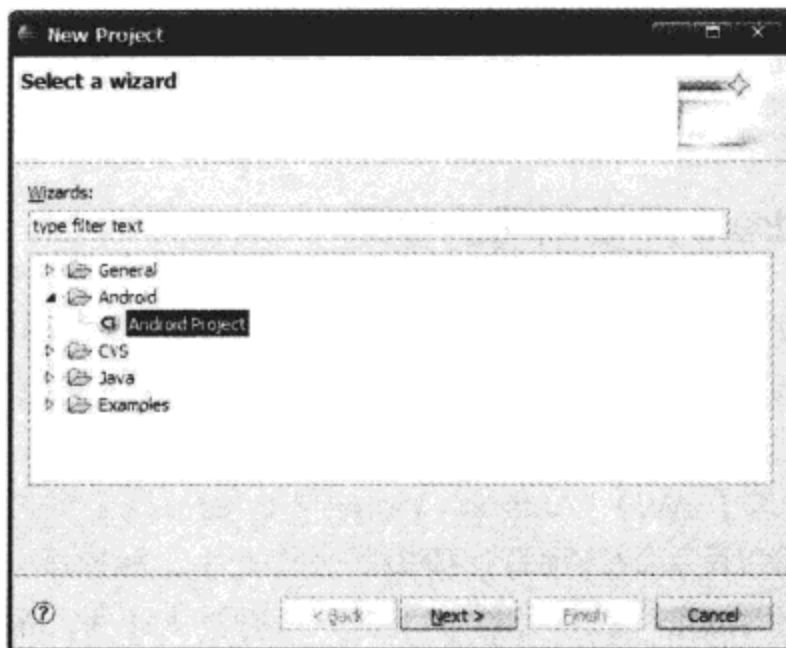


图 2-14 新建一个 Android 工程

(2) 点击“Next”按钮，在“Project name”文本框中输入“HelloAndroid”，然后在“Build Target”选项框中选择“Android SDK 1.5”，在 Application name 文本框中输入这个应用程序的名字 (HelloAndroid)，在 Package name 文本框中输入应用程序包的名字 (com.yarin.Android.HelloAndroid)，在 Create Activity 文本框中输入 Activity 的名字 (HelloAndroid)，如图 2-15 所示。

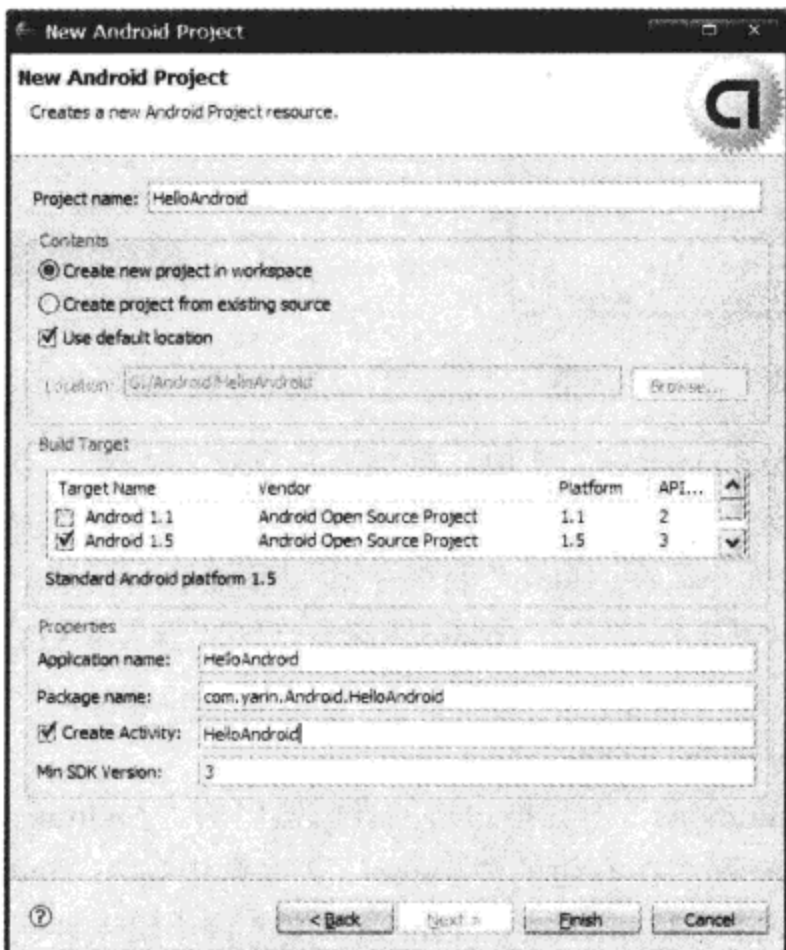


图 2-15 新建 HelloAndroid 工程

(3) 单击“Finish”按钮，此时 Eclipse 会自动完成 Android 项目的创建，这时 Eclipse 开发平台左边的导航器中显示了刚才创建的项目“HelloAndroid”。如果没有出现导航器，则可以通过单击“Window”→“Show View”→“Package Explorer”菜单命令来显示导航器，如图 2-16 所示。

到这里，HelloAndroid 项目已经创建好，而且这个项目是由我们前面安装的 ADT 插件自动生成，所以不用编写代码即可运行。下面我们将讲述如何在模拟器中运行刚刚创建的 HelloAndroid 项目。

2.3.2 运行 HelloAndroid 及模拟器的使用

上面我们已经利用 ADT 插件通过 Eclipse 创建好了第一个 Android 项目，而且没有编写任何代码，我们很想看看运行之后的结果！不要着急，在模拟器中运行该应用之前，有必要了解一下模拟器的使用和配置。

从 Android 1.5 开始引入了 AVD (Android Virtual Device) 这个概念。AVD 是一个经过配置的模拟器。在创建 AVD 时可以配置的选项有：模拟器影像大小、触摸屏、轨迹球、摄像头、屏幕分辨率、键盘、GSM、GPS、Audio 录放、SD 卡支持、缓存区大小等。配置 Android 模拟器的具体步骤如下所示。

(1) 首先打开“Android SDK and AVD Manager”，如图 2-17 所示。

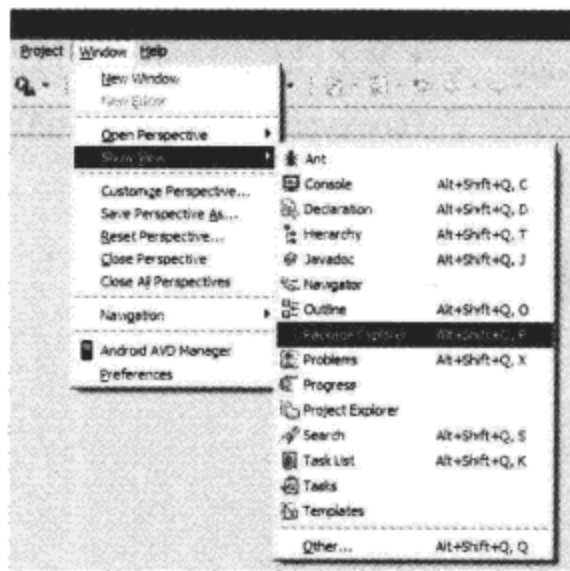


图 2-16 显示项目管理器

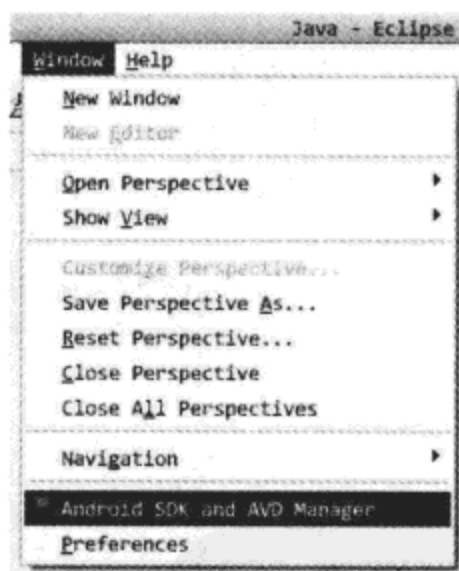


图 2-17 Android SDK and AVD Manager 菜单

(2) 点击左边的“Virtual Devices”选项，再点击右边的“New...”按钮，新建一个 AVD。

(3) 在“Name”标签处填写 AVD 的名字，在“Target”标签处选择 API 等级，在“Size”标签处填写要创建的 SD 卡的大小，在“Skin”标签中设置模拟器的风格，如图 2-18 所示。

(4) 到这里，我们便可以运行第一个 Android 项目了吗？还是不行，还需要配置模拟器运行的 AVD。操作步骤为：点击“Run”，选择“Run Configurations”菜单命令，打开“Run Configurations”对话框，如图 2-19 所示。

(5) 双击“Run Configurations”对话框左边的导航器中的“Android Application”菜单命令，创建一个 Android 项目运行配置。在右边的“Name”文本框中输入 Android 项目运行配置的名字 (HelloAndroid)，在“Android”选项卡中的“Project”文本框中输入要运行的 Android 项目，同样可以点击右边的“Browse...”按钮来选择 Android 项目，如图 2-20 所示。

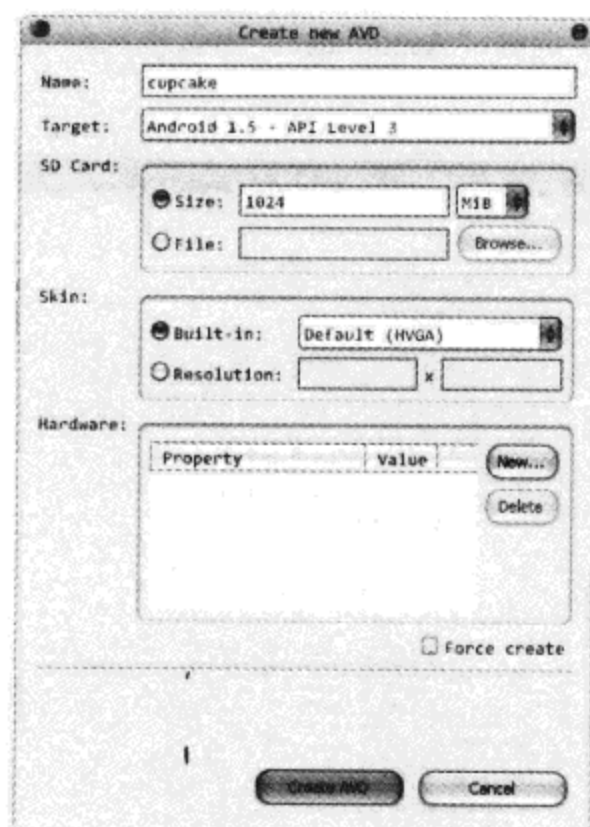


图 2-18 创建 AVD

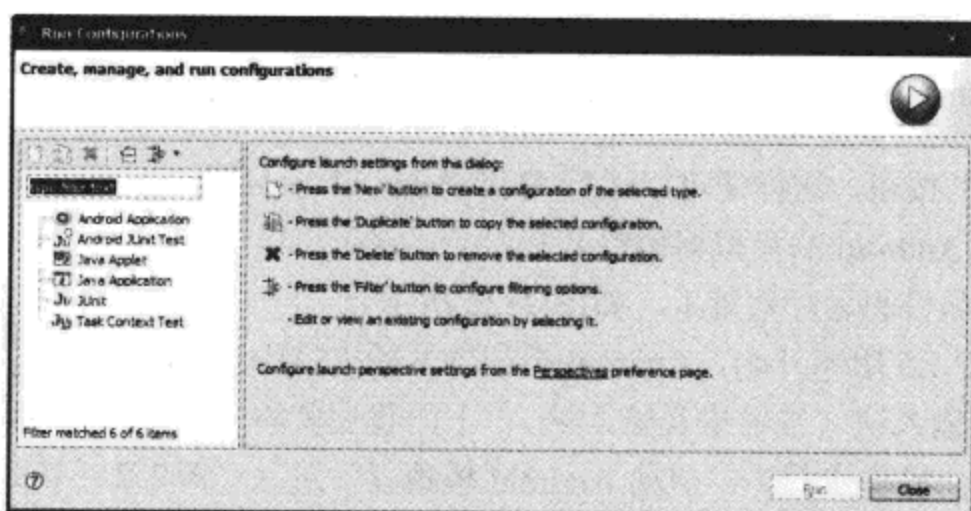


图 2-19 运行配置界面

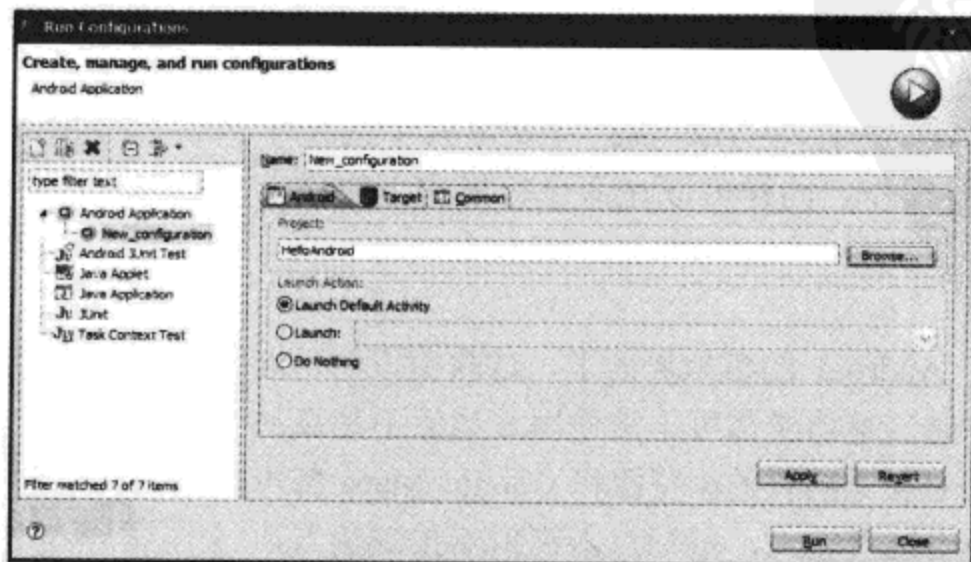


图 2-20 配置要运行的 HelloAndroid 项目

(6) 点击“Target”选项卡，选择“Automatic”单选框，然后在 AVD 列表框中选择我们刚才创建的 AVD，如图 2-21 所示。

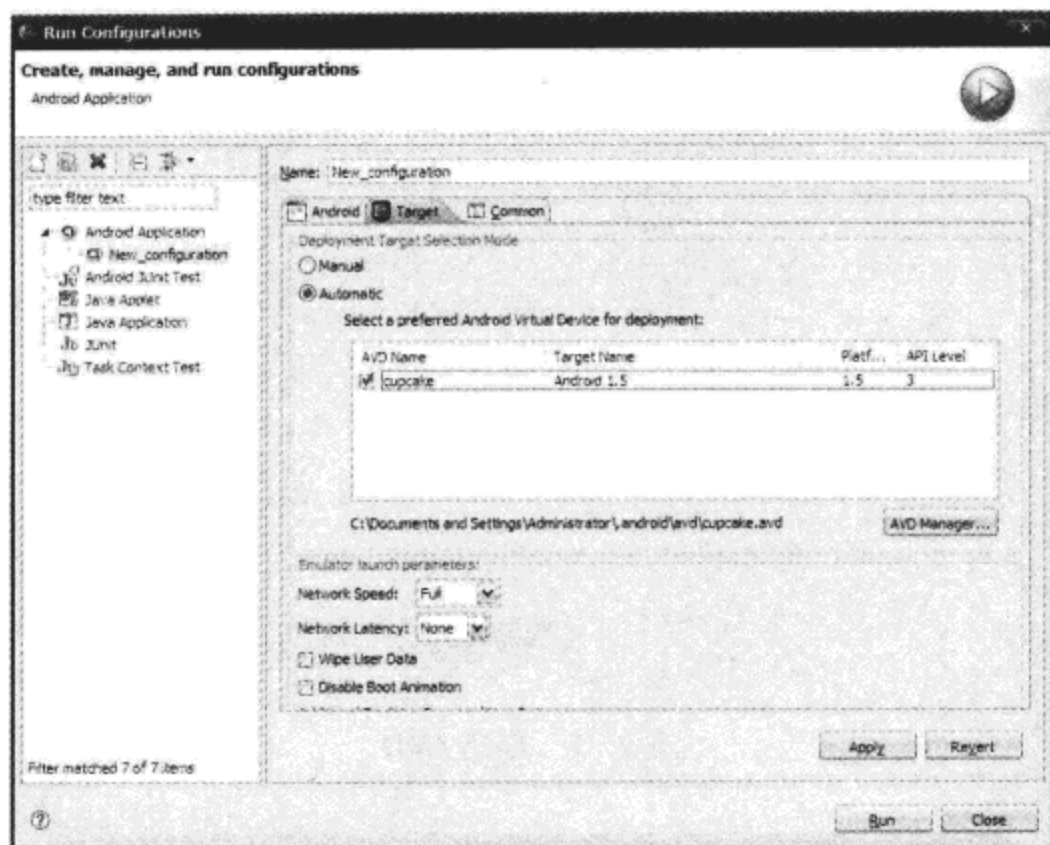


图 2-21 制定运行 HelloAndroid 项目的 AVD

(7) 点击“Run”按钮，这样便可以运行 HelloAndroid 项目了，不过 Android 模拟器启动非常慢，慢慢等吧。但是 Android 的模拟器做得非常漂亮，终于可以看到第一个 Android 项目的运行效果了，如图 2-22 所示。

从 Android SDK 1.5 版本开始，Android 模拟器开始支持中文了，也内置了中文输入法（谷歌拼音输入法），下面我们就将模拟器改为中文环境。操作步骤为：启动 Android 模拟器，进入 Android 模拟器菜单，选择“Settings”菜单项，开打“Settings”菜单，选择“Locale&text”菜单项，打开“Locale&text”菜单，依次选择“Select locale”项和“Chinese(China)”项，这样就设置为中文了，然后返回桌面，如图 2-23 所示。

上文我们使用 ADT 插件在 Eclipse 开发工具中创建了 AVD 及设置模拟器等操作，同样可以在命令行模式下完成上面的操作。

扩展学习

大家已经看到了 Android 的模拟界面了，这款模拟器功能非常齐全，电话本、通话等功能都可正常使用（当然不是真的从模拟器中打电话）。甚至其内置的浏览器和 Google Maps 都可以联网。用户可以使用键盘输入，鼠标点击模拟器按钮输入，甚至还可以使用鼠标点击、拖动屏幕进行操作。我们在开发项



图 2-22 HelloAndroid 项目在模拟器中的运行效果



图 2-23 Android 模拟器显示中文界面

目时, 这个模拟器完全可以满足我们测试的需求。下面我们列举一些常用的模拟器操作。

- ❑ 列出模拟器类型: `android list targets`。
- ❑ 创建模拟器: `android create avd --target 2 --name cupcake`, `cupcake` 为新建模拟器的名字。
- ❑ 列出自己创建的模拟器: `android list avd`。
- ❑ 切换模拟器样式: 在创建命令后面加上 “`--skin QVGA`” 即可。切换样式: Windows 操作系统按 F7 键即可。
- ❑ 删除模拟器: `android delete avd --name cupcake`, `cupcake` 为删除的模拟器的名字。
- ❑ 指定用什么模拟器启动: `emulator -debug avd_config -avd cupcake`, `cupcake` 为模拟器的名字。
- ❑ 将 apk 文件安装到 Android 模拟器。操作步骤为: 首先启动 Android 模拟器, 然后打开命令行对话框, 进入命令行模式。在命令行模式下进入 Android SDK 安装目录下面的 `tools` 文件夹, 输入 “`adb install c:\ poker80.apk`” (`c:\ poker80.apk` 是要安装的文件的路径), 这样便可以将 apk 文件安装到模拟器上, 如图 2-24 所示。

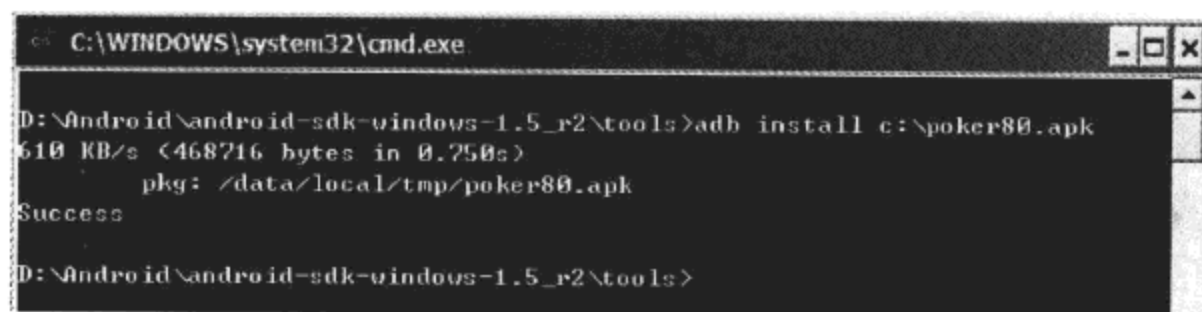


图 2-24 安装 apk 文件到模拟器

- ❑ 卸载模拟器中的 apk 文件。操作步骤为: 首先启动 Android 模拟器, 进入命令行模式。在命令行模式下进入 Android SDK 安装目录下面的 `tools` 文件夹, 然后在命令行处依次输入 “`adb shell`”、“`cd data`”、“`cd app`”、“`ls`” (主要是针对不知道包下面的文件的情况, 可以用 `ls` 命令列表显示出来)、“`rm com.fungsing.poker80.apk`” 命令 (“`com.fungsing.poker80.apk.apk`” 是你要卸载的 apk 包), 如图 2-25 所示。

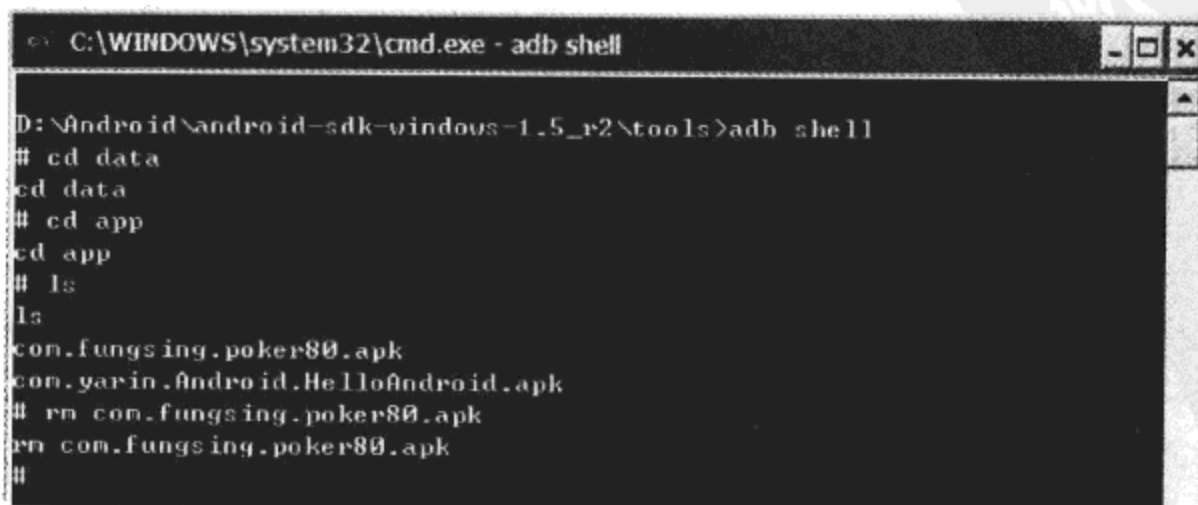


图 2-25 从 Android 模拟器卸载 apk 文件

2.3.3 调试 HelloAndroid

在 Eclipse 开发工具中调试程序的方法很多, 使用 Eclipse 调试 Android 程序时需要注意一些细节上的问题。许多刚接触 Android 的开发者, 在调试 Android 程序时总是不能迅速地找到程序的错误所在, Eclipse+ADT 的开发环境中没有直接跟踪对象内容的方法, 但是我们可以使用 Google 提供的 ADT 插件 DDMS (Dalvik Debug Monitor Service) 在 Eclipse 上轻松地调试 Android 程序。DDMS 为我们提供了很多功能, 例如: 测试设备截屏, 针对特定的进程查看正在运行的线程以及堆信息, Logcat, 广播状态信息, 模拟电话呼叫, 接收 SMS, 虚拟地理坐标等等, 下面我们通过 DDMS 来调试我们的 HelloAndroid 项目。

(1) 将 Eclipse 开发工具的工作界面切换到 DDMS 标签。首先确定 Eclipse 开发工具右上角是否有“DDMS”标签, 如果有, 则直接点击该标签即可切换到 DDMS 工作界面, 如图 2-26 所示。如果没有, 则点击“Open Perspective”按钮, 选择“Other...”命令按钮, 打开“Open Perspective”对话框, 如图 2-27 所示。在“Open Perspective”对话框中选择“DDMS”选项, 然后点击“OK”按钮, 如图 2-28 所示。

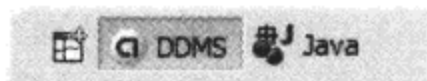


图 2-26 DDMS 工作界面切换



图 2-27 打开视图布局显示操作

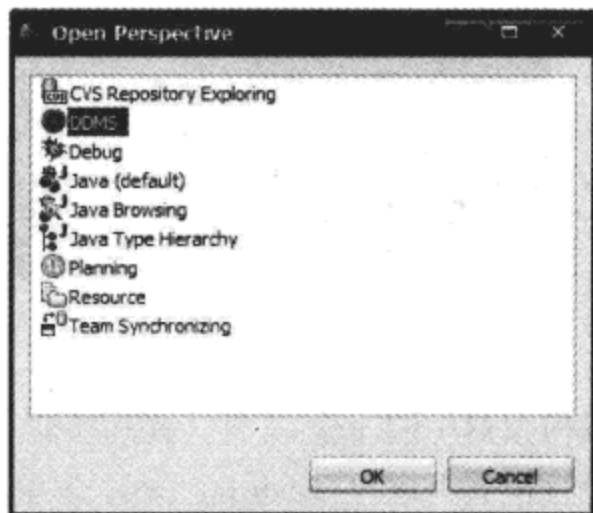


图 2-28 视图布局选择框

(2) 在“DDMS”界面中选择“Devices”标签, 查看其菜单的功能, 可以看到 Debug Process (调试进程)、Update Threads (更新线程)、Update Heap (更新堆)、Cause GC (引起垃圾回收)、Stop Process (停止进程)、Screen Capture (屏幕截图)、Reset adb (重启 Android Debug Bridge) 菜单选项, 如图 2-29 所示。

从图 2-29 中可以观察到 Android 程序运行时的各种状态, 比如进程信息、线程分析、堆内存的占用, 结束一个进程等。当然, 这些操作都是在 DDMS 框架下进行的, 日常开发的程序是无法执行调用的。如果 adb 调试桥运行不稳定, 可以选择“Reset adb”来重新启动“adb.exe”进程。下面我们介绍如何使用 DDMS 的“Logcat”来调试 Android 程序, 步骤如下:

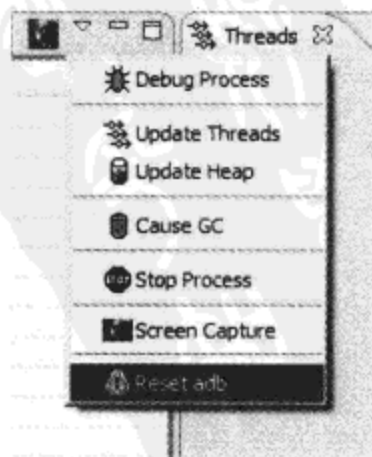


图 2-29 DDMS 操作菜单

(1) “Logcat”通过“android.util.Log”类的静态方法来查找错误和打印系统日志消息。它是一

个进行日志输出的 API，我们在 Android 程序中可以随时为某一个对象插入一个 Log，然后在 DDMS 中观察 Logcat 的输出是否正常。android.util.Log 常用的方法有以下 5 个：

- ❑ Log.v(String tag, String msg);
- ❑ Log.d(String tag, String msg);
- ❑ Log.i(String tag, String msg);
- ❑ Log.w(String tag, String msg);
- ❑ Log.e(String tag, String msg);

这 5 种方法的首字母分别对应 VERBOSE、DEBUG、INFO、WARN、ERROR。当利用 DDMS 进行调试时，它们的区别并不大，只是显示的颜色不同，可以控制要显示的某一类错误，一般如果使用“断点”方式来调试程序，则使用 Log.e 比较合适。但是根据规范建议 Log.v、Log.d 信息应当只存在于开发过程中，最终版本只可以包含 Log.i、Log.w、Log.e 这三种日志信息。下面我们对“HelloAndroid”程序进行调试，首先修改“HelloAndroid.java”如代码清单 2-1 所示。我们在代码中加入了需要输出的日志信息。

代码清单 2-1 第 2 章\HelloAndroid\src\com\yarin\Android\HelloAndroid\HelloAndroid.java

```
/* 定义 TAG 标签，这样可以很好地区分打印出来的 Log */
private static final String TAG = "HelloAndroid";

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    /* 打印出不同的 Log 信息 */
    Log.v(TAG, "VERBOSE");
    Log.d(TAG, "DEBUG");
    Log.i(TAG, "INFO");
    Log.w(TAG, "WARN");
    Log.e(TAG, "ERROR");
    setContentView(R.layout.main);
}
```

(2) 点击“Run”→“Debug”菜单命令，进入调试模式，如图 2-30 所示。

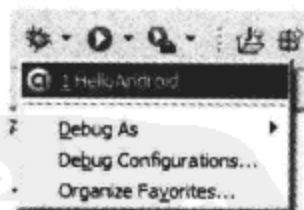


图 2-30 调试菜单命令

(3) 切换到“DDMS”界面，点击“Logcat”标签，即可查看我们刚刚在程序中打印的 Log 信息。用不同颜色表示了不同等级的信息，这样就可方便地对程序进行跟踪，使得调试 Android 程序更加方便。

在调试 Android 程序时，同样可以通过设置断点的方式来调试程序。在启动应用程序进行调试时，Eclipse 会自动切换到 Debug 透视图。毫无疑问，最常见的调试步骤是设置断点，这样可以检查条件语句或循环内的变量和值。要在 Java 透视图的 Package Explorer 视图中设置断点，双击选择的源代码文件，在一个编辑器中打开它。遍历代码，将鼠标放在可疑代码一行的标记栏（在编辑器区域的左侧）上，双击即可设置断点，如图 2-31 所示。

注意 最好不要将多条语句放在一行上，因为会无法单步执行，也不能为同一行上的多条语句设置行断点。

一旦找到错误发生的位置，你可能想知道在程序崩溃之前它在做什么。一种方法是单步执行程

序的每行语句，直到运行到可疑的那一行。有时候最好只运行一段代码，在可疑处停止运行，检查数据。另一种方法是声明条件断点，断点在表达式值发生变化时触发。如图 2-32 所示，我们设置条件“`savedInstanceState == null`”，当满足这个条件时，程序就会挂起。除此之外，在输入条件表达式时，也可以使用代码帮助。为了在 Debug 透视图的编辑器中计算表达式的值，选择设置了断点的那行代码，在上下文菜单中，通过 `Ctrl+Shift+I` 或右键单击你感兴趣的变量并选择 **Inspect** 选项。在当前堆栈框架的上下文中会计算表达式的值，在 **Display** 窗口的 **Expressions** 视图中会显示结果。

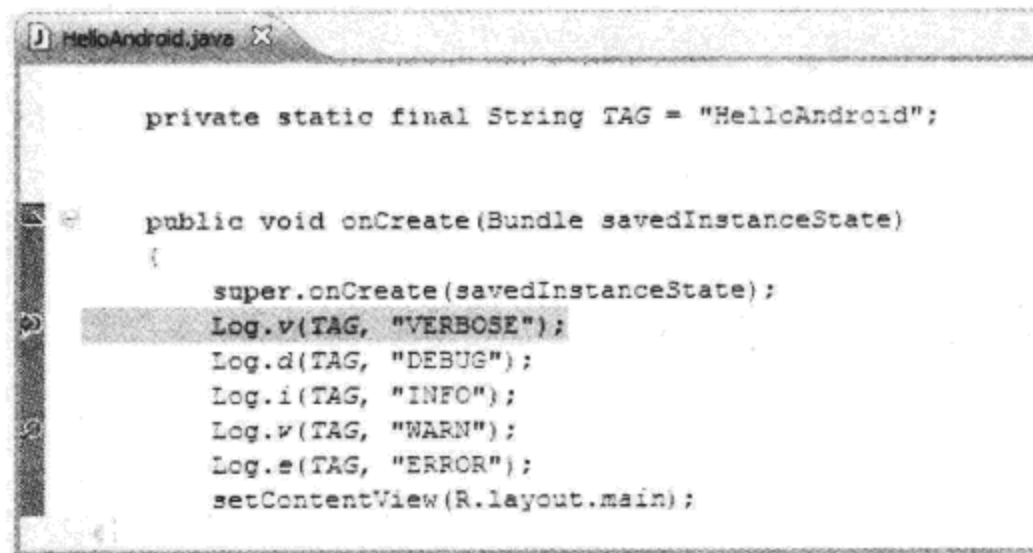


图 2-31 设置“断点”

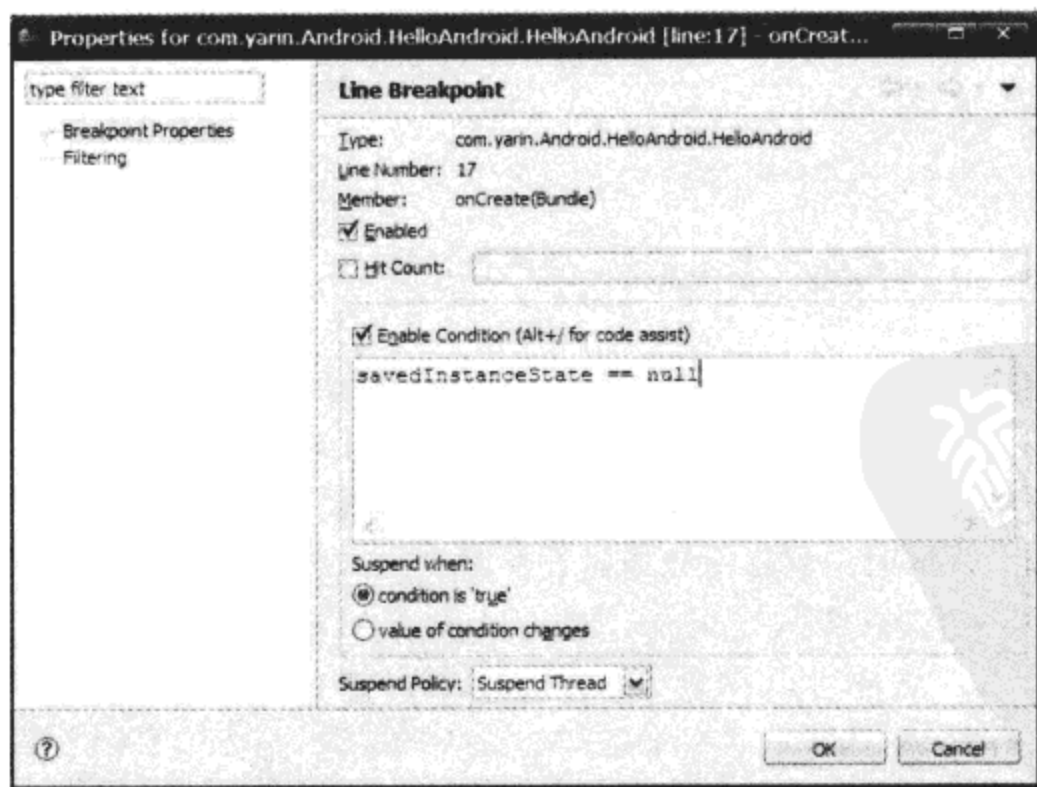


图 2-32 设置条件断点

要在 **Debug** 视图中挂起执行线程，选择一个运行线程，单击 **Debug** 视图工具栏中的 **Suspend**。该线程的当前调用堆栈就会显示出来，当前执行的代码行就会在 **Debug** 透视图中的编辑器中高亮显示。挂起一个线程时，将鼠标放在 **Java** 编辑器中的变量上，该变量的值就会在一个小

的悬停窗口中显示出来。此时，该线程的顶部堆栈框架也会自动选中，其中的可视变量也会在 Variables 视图中显示出来，可以通过单击 Variables 视图中合适的变量名来检查变量。

以上列举了一些在 Eclipse 编辑器中常用的调试方式，当然调试的方式很多，读者同样可以根据自己的需要选择不同的方式进行调试。希望读者能够根据不同的错误采取不同的方式进行调试，使错误能快速地出现在眼前。

2.4 小结

本章主要对 Android 应用开发的前期工作进行了整理，即 Android 开发工具的准备、环境的搭建及配置，最后为了测试我们的环境安装是否正确，写出了一个最经典的 HelloAndroid 程序。同时，了解了 Android 平台如何调试程序，以辅助我们后期能够快速开发出 Android 应用。本章是 Android 应用开发的基础，大家好好把握，下面我们将正式对 Android 进行系统学习。





第 3 章

Android 程序设计基础

通过上一章的学习，我们对 Eclipse+ADT 开发流程有了初步的认识和了解，对初学者来说，这一章的内容比较繁琐，但是又必须掌握，这也是进行 Android 开发必须经过的第一步，有了这个基础，我们下面将进行正式开始 Android 应用程序设计。

3.1 Android 程序框架

上一章我们建立了 HelloAndroid 项目，代码是由 ADT 插件自动生成的，我们没有对其进行编码，所以没有对其框架进行分析。其实每一个平台都有自己的结构框架，比如我们在最初学习 Java 或者 C\C++ 时，第一个程序总是 main 函数，以及文件类型和存储方式等。这一节将对 Android 平台的目录结构、文件类型及其负责的功能和 Android 平台的 main 函数进行剖析。

3.1.1 Android 项目目录结构

有了前面两章的基础，现在我们来打开上一章建立的 HelloAndroid 项目，分析其项目目录结构，对 Android 项目进一步地深入了解。首先启动 Eclipse，展开“Package Explorer”导航器中的“HelloAndroid”项目，如图 3-1 所示。

与一般的 Java 项目一样，src 文件夹是项目的所有包及源文件（.java），res 文件夹中则包含了项目中的所有资源，比如：程序图标（drawable）、布局文件（layout）、常量（values）等。下面来介绍其他 Java 项目中没有的 gen 文件夹中的 R.java 文件和每个 AndroidManifest.xml 文件。

□ R.java 是在建立项目时自动生成的，这个文件是只读模式，不能更改，R.java 文件是定义该项目所有资源的索引文件。先来看看 HelloAndroid 项目的 R.java 文件，如代码清单 3-1 所示。

代码清单 3-1 第 2 章\HelloAndroid\gen\com\yarin\Android\HelloAndroid\R.java

```
package com.yarin.Android.HelloAndroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
}
```



图 3-1 HelloAndroid 项目

```

    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}

```

可以看到这里定义了很多常量，这些常量的名字都与 `res` 文件夹中的文件名相同，这再次证明了 `R.java` 文件中所存储的是该项目所有资源的索引。有了这个文件，可以很快地找到要使用的资源，由于这个文件不能手动编辑，所以当在项目中加入了新的资源时，只需要刷新一下该项目，`R.java` 文件便自动生成了所有资源的索引。

□ `AndroidManifest.xml` 文件则包含了该项目中所使用的 `Activity`、`Service`、`Receiver`，我们先来打开 `HelloAndroid` 项目中的 `AndroidManifest.xml` 文件，如代码清单 3-2 所示。

代码清单 3-2 第2章\HelloAndroid\AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yarin.Android.HelloAndroid"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>

```

代码清单 3-2 中，`intent-filters` 描述了 `Activity` 启动的位置和时间。每当一个 `Activity`（或者操作系统）要执行一个操作时，它将创建出一个 `Intent` 的对象，这个 `Intent` 对象能承载的信息可描述你想做什么，你想处理什么数据，数据的类型，以及一些其他信息。而 `Android` 则会和每个 `Application` 所暴露的 `intent-filter` 的数据进行比较，找到最合适 `Activity` 来处理调用者所指定的数据和操作。下面我们来仔细分析 `AndroidManifest.xml` 文件，如表 3-1 所示。

表 3-1 `AndroidManifest.xml` 分析

项	说 明
<code>manifest</code>	根节点，描述了 <code>package</code> 中所有的内容
<code>xmlns:android</code>	包含命名空间的声明。 <code>xmlns:android=http://schemas.android.com/apk/res/android</code> ，使得 <code>Android</code> 中各种标准属性能在文件中使用，提供了大部分元素中的数据
<code>Package</code>	声明应用程序包
<code>application</code>	包含 <code>package</code> 中 <code>application</code> 级别组件声明的根节点。此元素也可包含 <code>application</code> 的一些全局和默认的属性，如标签、 <code>icon</code> 、主题、必要的权限，等等。一个 <code>manifest</code> 能包含零个或一个此元素（不能多余一个）
<code>android:icon</code>	应用程序图标

(续)

项	说 明
android:label	应用程序名字
Activity	用来与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 Activity 所实现，并声明在另外的 Activity 标记中。注意，每一个 Activity 必须有一个 <activity> 标记对应，无论它给外部使用或是只用于自己的 package 中。如果一个 Activity 没有对应的标记，你将不能运行它。另外，为了支持运行时查找 Activity，可包含一个或多个 <intent-filter> 元素来描述 Activity 所支持的操作
android:name	应用程序默认启动的 Activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 IntentFilter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、icon 和其他信息
action	组件支持的 Intent action
category	组件支持的 Intent Category。这里指定了应用程序默认启动的 Activity
uses-sdk	该应用程序所使用的 sdk 版本

下面我们看看资源文件中一些常量的定义，如 String.xml，如代码清单 3-3 所示。

代码清单 3-3 第 2 章\HelloAndroid\String.xml

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

这个文件很简单，就定义了两个字符串资源，因此，我们可以在代码清单 3-1 中看到如下内容，即定义了“app_name”和“hello”两个常量，分别指向代码清单 3-3 中的两个字符串资源。

```
public static final class string {
    public static final int app_name=0x7f040001;
    public static final int hello=0x7f040000;
}
```

那么如何在程序中使用我们所定义的这些资源呢？首先，通过 Context 的 getResources 实例化一个 Resources 对象，然后通过 Resources 的 getString 方法取得指定索引的字符串，代码如下：

```
Resources r = this.getContext().getResources();
String appname= ((String) r.getString(R.string.app_name));
String hello= ((String) r.getString(R.string.hello));
```

项目中所有使用的常量都可以通过这种 XML 文件的方式定义，比如，下面是我们通过 XML 文件定义的一些有关颜色的资源。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="status_idle">#cccccc</color>
    <color name="status_done">#637a47</color>
    <color name="status_sync">#cc9900</color>
    <color name="status_error">#ac4444</color>
</resources>
```

现在来分析 HelloAndroid 项目的布局文件 (layout)，首先打开 res->layout->main.xml 文件，如代码清单 3-4 所示。

代码清单 3-4 第2章\HelloAndroid\res\layout\main.xml

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

代码清单 3-4 中，有以下几个布局和参数。

- ❑ **<LinearLayout>**: 线性版面配置，在这个标签中，所有元件都是按由上到下的排列排成的。
- ❑ **android:orientation**: 表示这个介质的版面配置方式是从上到下垂直地排列其内部的视图。
- ❑ **android:orientation**: 表示这里是水平排列。
- ❑ **android:layout_width**: 定义当前视图在屏幕上所占的宽度，**fill_parent** 即填充整个屏幕。
- ❑ **android:layout_height**: 定义当前视图在屏幕上所占的高度，**fill_parent** 即填充整个屏幕。
- ❑ **wrap_content**: 随着文字栏位的不同而改变这个视图的宽度或高度。

layout_weight 用于给一个线性布局中的多个视图的重要度赋值。所有视图都有 **layout_weight** 值，默认为零，即需要显示多大的视图就占据多大的屏幕空间。如果值大于零，则将父视图中的可用空间分割，分割大小具体取决于每一个视图的 **layout_weight** 值和该值在当前屏幕布局的整体 **layout_weight** 值，以及其他视图屏幕布局的 **layout_weight** 值中所占的比例。

在这里，布局中设置了一个 **TextView**，用来配置文本标签 **Widget**，其中设置的属性 **android:layout_width** 为整个屏幕的宽度，**android:layout_height** 可以根据文字来改变高度，而 **android:text** 则设置了这个 **TextView** 要显示的文字内容，这里引用了 **@string** 中的 **hello** 字符串，即 **String.xml** 文件中的 **hello** 所代表的字符串资源。**hello** 字符串的内容“Hello World, HelloAndroid!”就是我们在 **HelloAndroid** 项目运行时看到的字符串。

最后，我们来分析 **HelloAndroid** 项目的主程序文件 **HelloAndroid.java**，如代码清单 3-5 所示。

代码清单 3-5 第2章\HelloAndroid\src\com\yarin\Android\HelloAndroid\HelloAndroid.java

```
...
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        /* 设置 Activity 要显示的布局为 (R.layout.main) */
        setContentView(R.layout.main);
    }
...
```

主程序 **HelloAndroid** 类继承自 **Activity** 类，重写了 **void onCreate(Bundle savedInstanceState)** 方法。在 **onCreate** 方法中通过 **setContentView(R.layout.main)** 设置 **Activity** 要显示的布局文件 (**layout/main.xml**)。

到这里，是不是明白了为什么我们在创建项目时没有进行编码就可以直接运行程序呢？当然，这也是 **Android** 开发的特点，这样可以很轻松地将代码和 **UI** 分开，在国际化和程序维护方面有着巨大的作用。如果你的 **Android** 程序需要适应国际化，比如说多国语言等问题，那么就可以定义不

同语言的 UI 布局，在程序装载时调用不同的布局。而且，如果我们需要修改 UI 的一些问题，就不必查看代码了，直接更改这些布局文件即可，是不是很方便？当然，这需要开发者在开发时使用这种 MVC 框架，尽量减少使用“硬编码”。笔者个人建议使用这种框架。

3.1.2 Android 应用解析

上面我们了解了 Android 应用程序的目录结构和其中每个文件的功能，要进行应用开发，还需要对 Android 应用构造进行深入分析。Android 应用程序由 4 个模块构造而成：Activity，Intent，Content Provider，Service。

当然，也不是每个 Android 应用程序都必须由这 4 部分组成，可以根据开发者需求进行组合，比如上面建立的 HelloAndroid 项目就只使用了 Activity 这一个模块。但是，任何一个应用程序都必须在 AndroidManifest.xml 文件中声明使用到的这些模块。

1. Activity

Activity 是最基本的模块，我们在 HelloAndroid 项目中已经使用过。我们称之为“活动”，在应用程序中，一个 Activity 通常就是一个单独的屏幕。每一个活动都被实现为一个独立的类，并且从活动基类中继承而来，活动类将会显示由视图控件组成的用户接口，并对事件作出响应。例如 HelloAndroid 项目中的 HelloAndroid.java 即继承了 Activity 类。大多数的应用都是由多个 Activity 显示组成，例如，对一个文本信息应用而言，第一个屏幕用来显示发送消息的联系人列表，第二个屏幕用来写文本消息和选择收件人，第三个屏幕查看消息历史或者消息设置操作等。

这里的每一个屏幕就是一个活动，很容易实现从一个屏幕到一个新的屏幕，并且完成新的活动。当一个新的屏幕打开后，前一个屏幕将会暂停，并保存在历史栈中。用户可以返回到历史栈中的前一个屏幕，当屏幕不再使用时，还可以从历史栈中删除。

简单理解，Activity 代表一个用户所能看到的屏幕，主要用于处理应用程序的整体性工作，例如，监听系统事件（按键事件、触摸屏事件等），为用户显示指定的 View，启动其他 Activity 等。所有应用的 Activity 都继承于 android.app.Activity 类，该类是 Android 提供的基层类，其他的 Activity 继承该父类后，通过父类的方法来实现各种功能，这种设计在其他领域也较为常见。

2. Intent

Android 用 Intent 这个特殊类实现在 Activity 与 Activity 之间的切换。Intent 类用于描述应用的功能。在 Intent 的描述结构中，有两个最重要的部分：动作和动作对应的数据。典型的动作类型有 MAIN、VIEW、PICK、EDIT 等，而动作对应的数据则以 URI 的形式表示。例如，要查看一个人的联系方式，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。

通过解析各种 Intent，从一个屏幕导航到另一个屏幕是很简单的。当向前导航时，Activity 将会调用 startActivity(Intent myIntent) 方法。然后，系统会在所有已安装的应用程序中定义的 IntentFilter 中查找，找到最匹配 myIntent 的 Intent 对应的 Activity。新的 Activity 接收到 myIntent 的通知后，开始运行。当 startActivity 方法被调用时，将触发解析 myIntent 的动作，该机制提供了两个关键好处：

- ❑ Activity 能够重复利用从其他组件中以 Intent 形式产生的请求。
- ❑ Activity 可以在任何时候被具有相同 IntentFilter 的新的 Activity 取代。

下面我们举例说明两个 Activity 之间的切换。运行效果：当应用程序启动时显示布局 main.xml，如图 3-2 所示，当点击“切换”按钮时，屏幕显示布局 main2.xml，如图 3-3 所示，再点击“切换”按钮，又回到如图 3-2 所示界面。就这样通过 Intent 完成了两个 Activity 之间的切换。

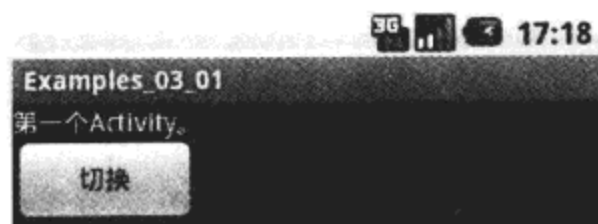


图 3-2 Activity01

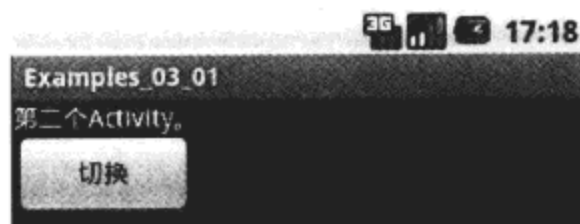


图 3-3 Activity02

下面我们来分析一下代码的具体实现，我们知道该项目是由两个 Activity 构成，在这两个 Activity 中分别显示了一个文本标签和一个按钮，关于界面的布局会在本书第 4 章进行详细讲解，要实现两个 Activity 的跳转，我们可以将要跳转的 Activity 类名绑定到 Intent 对象中，然后通过 startActivity 方法激活 Intent 对象中所指定的 Activity。关键代码如代码清单 3-6 所示。

代码清单 3-6 第 3 章\Examples_03_01\src\com\yarin\android\Examples_03_01\Activity01.java

```
/* 监听 button 的事件信息 */
button.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v)
    {
        /* 新建一个 Intent 对象 */
        Intent intent = new Intent();
        /* 指定 intent 要启动的类 */
        intent.setClass(Activity01.this, Activity02.class);
        /* 启动一个新的 Activity */
        startActivity(intent);
        /* 关闭当前的 Activity */
        Activity01.this.finish();
    }
});
```

然后，我们要从 Activity02 跳转到 Activity01 时，就只是需要在 Activity02.java 中使用同样的方法返回 Activity01 中。大家可以参考本书所附代码：第 3 章\Examples_03_01\src\com\yarin\android\Examples_03_01\ Activity02.java。值得注意的是，该项目中我们使用了两个 Activity，每一个 Activity 都需要在 AndroidManifest.xml 文件中进行声明，声明方法如代码清单 3-7 所示。

代码清单 3-7 第 3 章\Examples_03_01\AndroidManifest.xml

```
<activity android:name=".Activity01"
    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="Activity02"></activity>
```

如果希望 Android 应用能够对外部事件（如当电话呼入时，或者数据网络可用时，或者到了晚上时）做出响应，可以使用 IntentReceiver。虽然 IntentReceiver 在感兴趣的事件发生时会使用 NotificationManager 通知用户，但它并不能生成 UI。IntentReceiver 在 AndroidManifest.xml 中注册，

但也可以在代码中使用 `Context.registerReceiver()` 进行注册。当 `IntentReceiver` 被触发时，应用不必对请求调用 `IntentReceiver`，系统会在需要时启动应用。各种应用还可以通过使用 `Context.broadcastIntent()` 将它们自己的 `IntentReceiver` 广播给其他应用。

3. Content Provider

Android 应用能够将它们的数据保存到文件和 SQLite 数据库中，甚至是任何有效的设备中。当想将应用数据与其他的应用共享时，`Content Provider` 就可以发挥作用了。因为 `Content Provider` 类实现了一组标准的方法，能够让其他的应用保存或读取此内容提供器处理的各种数据类型。

数据是应用的核心。在 Android 中，默认使用鼎鼎大名的 SQLite 作为系统数据库。但是在 Android 中，使用方法有点不一样。在 Android 中，每一个应用都运行在各自的进程中，当一个应用需要访问其他应用的数据时，也就是数据需要在不同的虚拟机之间传递，这样的情况操作起来可能有些困难（正常情况下，不能读取其他应用的 db 文件），`Content Provider` 正是用来解决在不同的应用包之间共享数据的工具。

在 Android 中，`Content Provider` 是一个特殊的存储数据的类型，它提供了一套标准的接口用来获取和操作数据。并且，Android 自身也提供了现成的 `Content Provider`：`Contacts`、`Browser`、`CallLog`、`Settings`、`MediaStore`。应用可以通过唯一的 `ContentResolver` 界面来使用具体的某个 `Content Provider`，然后就可以用 `ContentResolver` 提供的方法来使用你需要的 `Content Provider` 了。其中，`ContentResolver` 提供的方法包括 `query()`、`insert()`、`update()` 等。要使用这些方法，还会涉及 URI。你可以将它理解成 string 形式的 `Content Provider` 的完全路径。

下面我们通过一个例子来学习 `Content Provider` 的使用，该例子主要通过 `Content Provider` 获得电话本中的数据，然后显示到一个 `TextView` 中，在运行程序之前我们先看看电话本中存储的电话号码，如图 3-4 所示，然后再运行程序看看我们获得的数据，如图 3-5 所示，并看看我们通过 `Content Provider` 获得的数据是否正确。



图 3-4 电话本数据

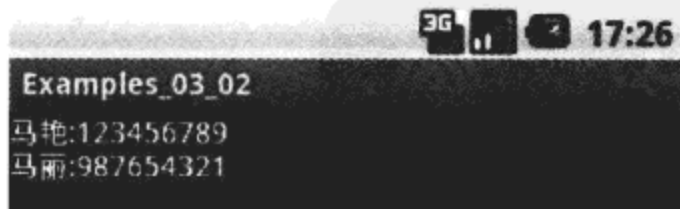


图 3-5 通过 `ContentProvider` 获得电话本数据

下面我们来分析一下如何实现通过 `ContentProvider` 取得电话本的数据，首先通过 `getContentResolver` 方法来取得一个 `ContentResolver` 对象，然后通过其 `query` 方法查询出符合标准的电话本记录，最后将这些数据都显示在一个 `TextView` 中即可，如代码清单 3-8 所示。

代码清单 3-8 第 3 章\Examples_03_02\src\com\yarin\android\Examples_03_02\Activity01.java

```
public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
```

```

        TextView tv = new TextView(this);
        String string = "";
        super.onCreate(savedInstanceState);
        //得到 ContentResolver 对象
        ContentResolver cr = getContentResolver();
        //取得电话本中开始一项的光标
        Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null,
            null, null);
        //向下移动光标
        while(cursor.moveToNext())
        {
            //取得联系人名字
            int nameFieldColumnIndex = cursor.getColumnIndex(PhoneLookup.
                DISPLAY_NAME);
            String contact = cursor.getString(nameFieldColumnIndex);
            //取得电话号码
            int numberFieldColumnIndex = cursor.getColumnIndex(PhoneLookup.
                NUMBER);
            String number = cursor.getString(numberFieldColumnIndex);

            string += (contact+":"+number+"\n");
        }
        cursor.close();
        //设置 TextView 显示的内容
        tv.setText(string);
        //显示到屏幕
        setContentView(tv);
    }
}

```

前面强调过，要使用这些模块，需要在 `AndroidManifest.xml` 声明，本例中我们使用了读取联系人的 API，因此，声明方式如下所示：

```

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

```

4. Service

Service 即“服务”的意思，既然是服务，那么 Service 将是一个生命周期长且没有用户界面的程序。比如一个正在从播放列表中播放歌曲的媒体播放器，在这个媒体播放器应用中，应该会有多个 Activity，让使用者可以选择歌曲并播放歌曲。然而，音乐重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。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。另外，还可以通过使用 `Context.bindService()` 方法连接到一个 Service 上（如果这个 Service 当前还没有处于启动状态，则将启动它）。当连接到一个 Service 之后，还可用 Service 提供的接口与它进行通信。以媒体播放器为例，我们还可以执行暂停、重播等操作。

下面通过一个例子来学习 Service 的使用，该例子通过 Service 来播放一首 MP3，如图 3-6 所示。当用户点击“开始”按钮，音乐开始播放；点击“停止”按钮，停止音乐播放。当然，这里需要在资源文件中添加一首 MP3 歌曲，如图 3-7 所示。

要实现音乐的播放，需要在界面中放置两个按钮，用来控制音乐的播放和停止。而我们的音乐

播放是通过一个服务来实现的,所以我们可以通过 `startService` 和 `stopService` 方法来开启和停止这个播放音乐的服务,如代码清单 3-9 所示。

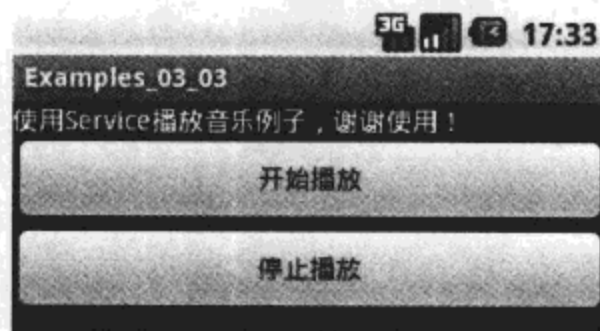


图 3-6 使用 Service 播放音乐



图 3-7 test.mp3

代码清单 3-9 第 3 章\Examples_03_03\src\com\yarin\android\Examples_03_03\Activity01.java

```
//开始按钮
private OnClickListener start = new OnClickListener()
{
    public void onClick(View v)
    {
        //开启 Service
        startService(new Intent("com.yarin.Android.MUSIC"));
    }
};
//停止按钮
private OnClickListener stop = new OnClickListener()
{
    public void onClick(View v)
    {
        //停止 Service
        stopService(new Intent("com.yarin.Android.MUSIC"));
    }
};
```

下面是该例子的核心内容。如何通过 Service 来播放音乐,其实也很简单,首先创建一个 `MusicService` 继承自 `Service`,然后通过 `start` 和 `stop` 方法来控制音乐的播放,如代码清单 3-10 所示。具体实现请参见本书所附代码:第 3 章\Examples_03_03。

代码清单 3-10 第 3 章\Examples_03_03\src\com\yarin\android\Examples_03_03\MusicService.java

```
public class MusicService extends Service
{
    //MediaPlayer 对象
    private MediaPlayer player;

    public IBinder onBind(Intent arg0)
    {
        return null;
    }
}
```

```

public void onStart(Intent intent, int startId)
{
    super.onStart(intent, startId);
    //这里可以理解为装载音乐文件
    player = MediaPlayer.create(this, R.raw.test);
    //开始播放
    player.start();
}

public void onDestroy()
{
    super.onDestroy();
    //停止音乐——停止 Service
    player.stop();
}
}

```

我们使用 Service 时同样需要在 AndroidManifest.xml 中声明，声明方式如代码清单 3-11 所示。

代码清单 3-11 第3章\Examples_03_03\AndroidManifest.xml

```

<service android:name=".MusicService">
    <intent-filter>
        <action android:name="com.yarin.Android.MUSIC" />
        <category android:name="android.intent.category.default" />
    </intent-filter>
</service>

```

3.2 Android 的生命周期

在前面的几个例子中，我们发现所有继承自 Activity 的类都重写了 onCreate 方法，程序运行就会自动进入这个方法。其实 Activity 类中还有很多类似于 onCreate 的方法，比如 onStart、onResume、onPause、onDestroy 等，而这些方法都是系统自动调用，从名字上大概就可以看出这是一些关于生命周期的方法，那么这些方法被调用的先后顺序是怎样的呢？Android 应用的生命周期又是如何呢？下面通过一个例子来进一步分析。

当应用程序启动时，进入如图 3-8 所示的 Activity01 界面，此时，点击“Activity02”按钮，进入 Activity02 界面，如图 3-9 所示。再点击“Activity01”按钮，返回 Activity01 界面，最后点击“Exit”按钮退出整个应用程序。

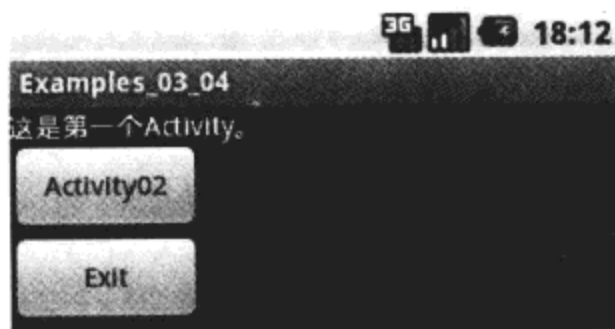


图 3-8 Activity01 界面

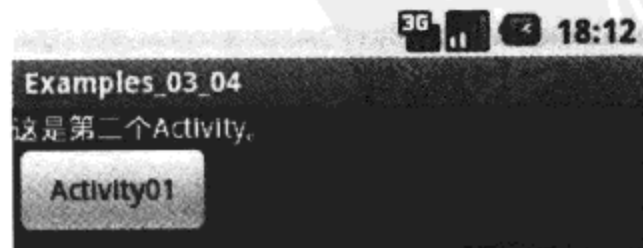


图 3-9 Activity02 界面

我们在这些类似于 onCreate 的方法中都加入了 log 函数，输出不同的信息，以便我们能更好地跟踪程序运行的过程，具体实现参见本书所附代码：第3章\Examples_03_04。

首先，我们需要在程序启动所默认的第一个界面中，加入一些 Log 函数，用于显示和输出 Log 信息，以帮助我们分析程序的执行流程，如代码清单 3-12 所示。

代码清单 3-12 第 3 章\Examples_03_04\src\com\lyarin\android\Examples_03_04\Activity01.java

```
public class Activity01 extends Activity
{
    private static final String TAG = "Activity01";

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.v(TAG, "onCreate");

        Button button1 = (Button) findViewById(R.id.button1);
        /* 监听 button 的事件信息 */
        button1.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v)
            {
                /* 新建一个 Intent 对象 */
                Intent intent = new Intent();
                /* 指定 intent 要启动的类 */
                intent.setClass(Activity01.this, Activity02.class);
                /* 启动一个新的 Activity */
                startActivity(intent);
                /* 关闭当前的 Activity */
                Activity01.this.finish();
            }
        });
        /* ***** */
        Button button3 = (Button) findViewById(R.id.button3);
        /* 监听 button 的事件信息 */
        button3.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v)
            {
                /* 关闭当前的 Activity */
                Activity01.this.finish();
            }
        });
    }

    public void onStart()
    {
        super.onStart();
        Log.v(TAG, "onStart");
    }

    public void onResume()
    {
        super.onResume();
        Log.v(TAG, "onResume");
    }

    public void onPause()
    {
        super.onPause();
        Log.v(TAG, "onPause");
    }
}
```

```

    }

    public void onStop()
    {
        super.onStop();
        Log.v(TAG, "onStop");
    }

    public void onDestroy()
    {
        super.onDestroy();
        Log.v(TAG, "onDestroy");
    }

    public void onRestart()
    {
        super.onRestart();
        Log.v(TAG, "onReStart");
    }
}

```

在第二个界面中，同第一个界面一样，加入一些可以区分的不同的 Log 信息。

同样需要在 AndroidManifest.xml 文件中声明所使用的两个 Activity 模块，如代码清单 3-13 所示。具体实现请参见本书所附代码：第 3 章\Examples_03_04。

代码清单 3-13 第 3 章\Examples_03_04\AndroidManifest.xml

```

<activity android:name=".Activity01"
    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="Activity02"></activity>

```

当在 Debug 该项目时，切换到 DDMS 标签即可以看到所打印出来的 Log 信息，这样就可以很清楚地分析程序的运行过程。

当程序第一次启动时，打印的 Log 信息如图 3-10 所示。我们看到程序的运行顺序为：Activity01 onCreate→Activity01 onStart→Activity01 onResume。这里我们可以看到，当一个 Activity 启动时，不是“创建”之后“开始”就完了，而是要经过“创建”，然后“开始”，最后“重绘”。

07-03 18:08:03.094	V	1255	Activity01	onCreate
07-03 18:08:03.135	V	1255	Activity01	onStart
07-03 18:08:03.144	V	1255	Activity01	onResume

图 3-10 第一次启动进入 Activity01 界面

当我们进入 Activity02 界面时，打印出的 Log 信息如图 3-11 所示。我们看到程序的运行顺序为：Activity01 onPause→Activity02 onCreate→Activity02 onStart→Activity02 onResume→Activity01 onStop→Activity01 onDestroy。这里我们看到，当程序从 Activity01 界面进入 Activity02 界面时，并不是马上将 Activity01 销毁，而是待 Activity02 启动之后将 Activity01 停止并销毁。

当我们返回 Activity01 界面时，打印出的 Log 信息如图 3-12 所示。我们看到程序的运行顺序为：Activity02 onPause→Activity01 onCreate→Activity01 onStart→Activity01 onResume→Activity02 onStop→Activity02 onDestroy。这里我们看到，当程序从 Activity02 界面返回 Activity01 界面时，并

不是马上将 Activity02 销毁，而是待 Activity01 启动之后将 Activity02 停止并销毁。

Time	pid	tag	Message
07-03 18:09:54.045	I 584	Activity	Starting
07-03 18:09:54.114	V 1255	Activity01	onPause
07-03 18:09:54.434	V 1255	Activity02	onCreate
07-03 18:09:54.444	V 1255	Activity02	onStart
07-03 18:09:54.463	V 1255	Activity02	onResume
07-03 18:09:54.844	I 584	Activity	Displayed
07-03 18:09:54.895	V 1255	Activity01	onStop
07-03 18:09:54.914	V 1255	Activity01	onDestroy

图 3-11 进入 Activity02 界面

Time	pid	tag	Message
07-03 18:10:51.725	I 584	Activity	Starting
07-03 18:10:51.834	V 1255	Activity02	onPause
07-03 18:10:52.244	V 1255	Activity01	onCreate
07-03 18:10:52.253	V 1255	Activity01	onStart
07-03 18:10:52.274	V 1255	Activity01	onResume
07-03 18:10:52.694	I 584	Activity	Displayed
07-03 18:10:52.714	V 1255	Activity02	onStop
07-03 18:10:52.724	V 1255	Activity02	onDestroy

图 3-12 返回 Activity01 界面

最后，当我们点击“Exit”按钮退出应用程序时，打印出的 Log 信息如图 3-13 所示。这里我们看到程序的运行顺序为：Activity01 onPause→Activity01 onStop→Activity01 onDestroy。这里我们看到当一个应用程序在退出时，并不是马上“停止”且“销毁”，而是经过“暂停”，到“停止”，然后再“销毁”。

Time	pid	tag	Message
07-03 18:11:24.835	V 1255	Activity01	onPause
07-03 18:11:25.273	V 627	InputCo	showStatus
07-03 18:11:25.315	V 1255	Activity01	onStop
07-03 18:11:25.324	V 1255	Activity01	onDestroy

图 3-13 退出应用程序

通过上面的例子，我们得出 Android 应用程序的生命周期如图 3-14 所示。

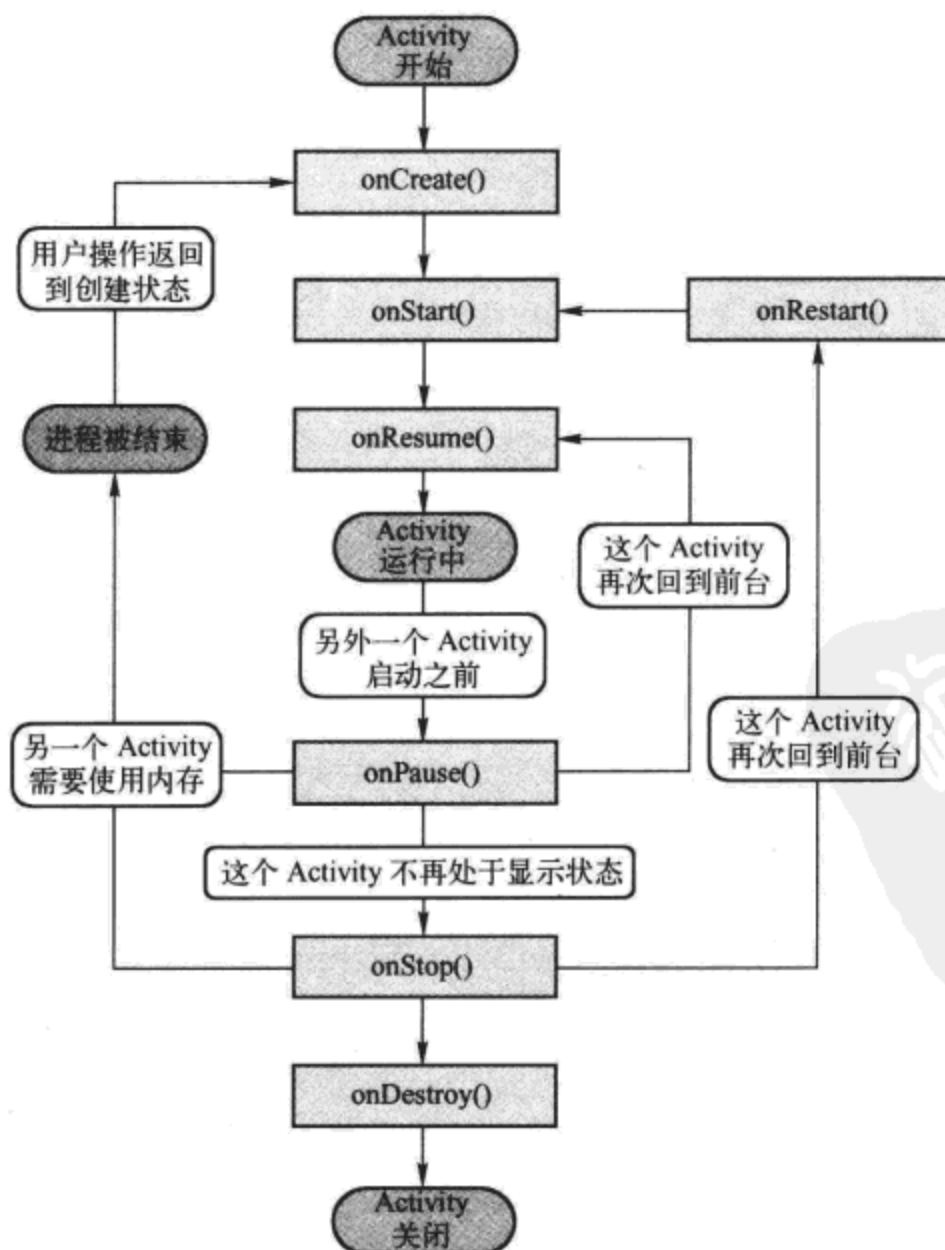


图 3-14 Android 应用的生命周期

3.3 Android 程序 UI 设计

在前面章节的例子中，我们已经接触了一些 UI 控件，比如 `TextView`、`Button` 等，其实这里所说的 UI 就是在我们所说的布局文件，UI 是一个应用程序的脸面，一个应用程序要想受用户喜爱，UI 可不能差。自从 Android SDK 1.0_r2 版本开始，ADT 提供了 UI 预览的功能。现在我们只需要打开一个 Android 项目的“/res/ layout/main.xml”并右键单击，依次选择“Open With”→“Android layout Editor”菜单命令，或者直接双击 main.xml 文件，即可以切换到 UI 设计界面，如图 3-15 所示。

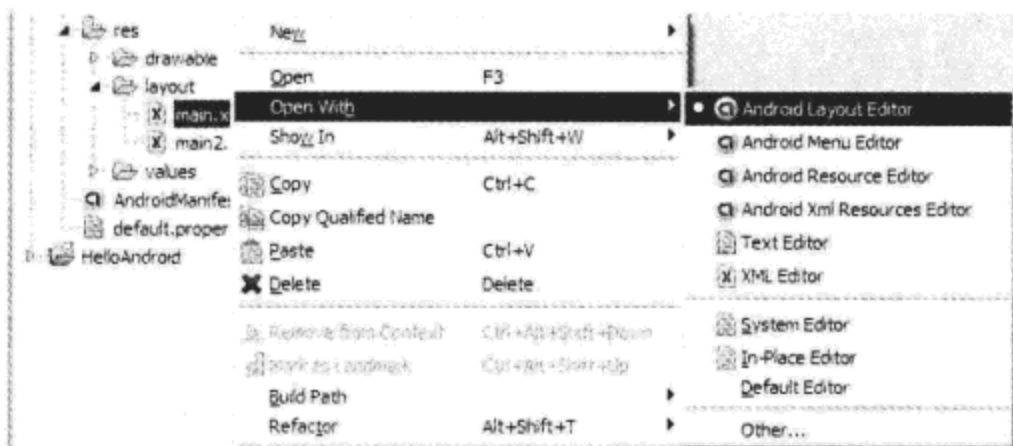


图 3-15 Android Layout Editor 命令

左边的 Layouts 标签的内容则是一些线性布局，可以使用它轻松地完成对布局的排版，比如横向或者纵向布局。Views 标签则是一些 UI 控件，可以将这些控件直接拖动到右边的窗口进行编辑，如图 3-16 所示。



图 3-16 Android Layout Editor

当然，还可以点击右下角的 main.xml 标签来切换到 XML 编辑器，对代码进行编排，如图 3-17 所示。将这些功能配合起来使用，基本可以满足开发者需求。

除了这个还不算太成熟的 UI 编辑器之外，笔者曾经使用过一个第三方的工具 DroidDraw，DroidDraw 是一个公开了源代码的 UI 设计器，可以根据自己的开发需要进行修改。www.DroidDraw.org 提供了在线使用 DroidDraw 的功能，当然也可以下载到本地来进行编辑，下载地

址: <http://code.google.com/p/droiddraw/>。

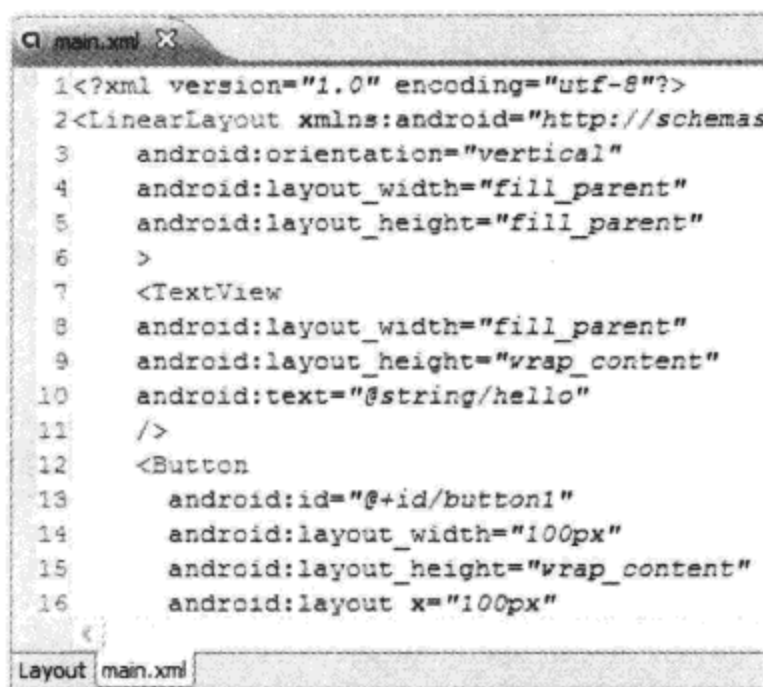


图 3-17 XML 编辑器

DroidDraw 的功能比较强大,可以直接拖动控件到窗口,然后设置其属性、参数等,这样便可以随心所欲地设计自己需要的 UI,然后点击“Generate”按钮即可生成出对应的布局代码,同时也可以点击“Load”按钮来载入已经编辑好的布局文件,如图 3-18 所示。

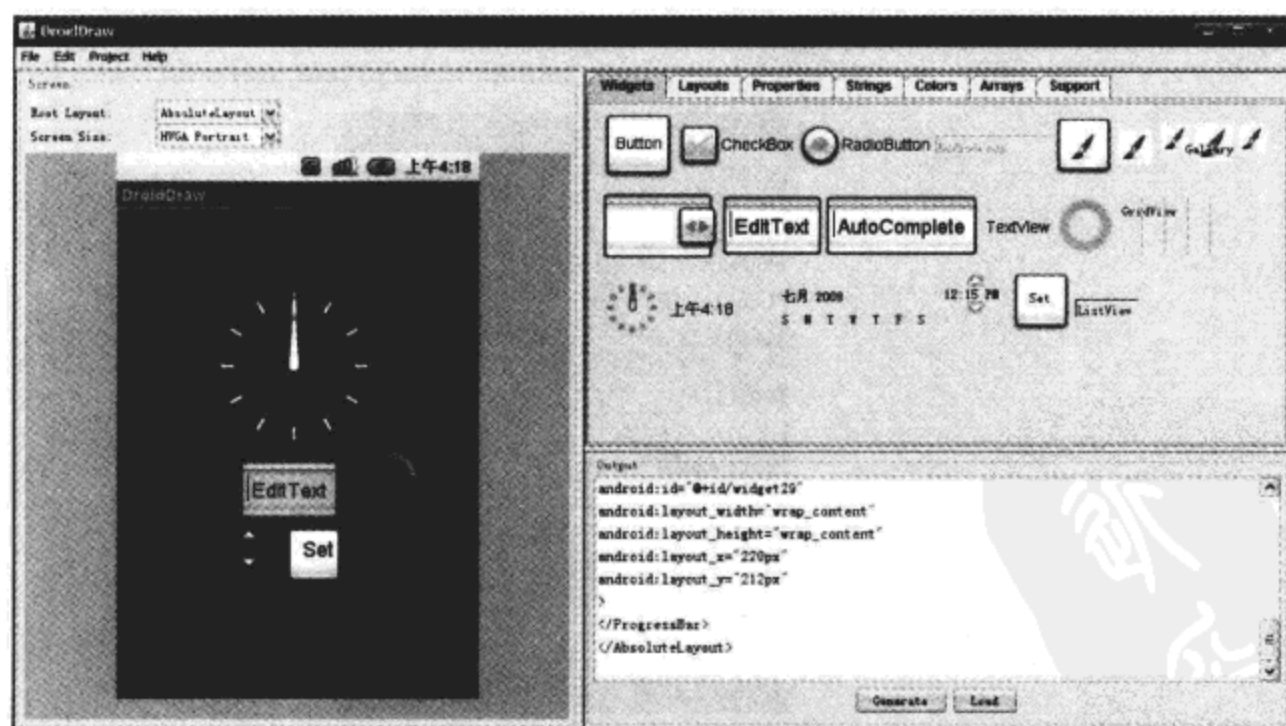


图 3-18 DroidDraw 操作界面

3.4 小结

本章主要介绍了 Android 应用程序框架及其生命周期,以及 UI 设计。首先彻底分析了上一章

的 HelloAndroid 项目，从其项目目录结构、文件功能等方面分析了 Android 应用程序的基本框架，其次逐一分析了构成 Android 应用程序的 4 个模块（Activity、Intent、Content Provider、Service），分别通过示例程序演示了其功能的运用。接着通过一个示例程序验证了 Android 应用程序的运行流程，从而得出 Android 应用程序的生命周期流程图。最后介绍了两个有关 UI 设计的工具，使得程序界面更加漂亮。

相信通过本章的学习，你已经开始“喜欢”上 Android 了，有你的这份热情和执着，加上每一章的示例，让你边学边做，理论加实践，轻轻松松学会 Android 应用开发。加油吧！后面的内容更精彩。



第4章

用户界面开发

一个好的应用界面的必备条件是：内容清楚、指示明白、屏幕美观和有亲切感。界面通常包含图形和文字。应用界面的设计是对控件进行适当的取舍及功能的选择和处理的过程。在程序设计中，需要对设计的方法反复推敲、琢磨，才能使其达到完美的境界。

Android 平台提供的控件是非常完美的，控件的使用与网页的设计类似，尽量用 `parent_width` 等抽象长度，用 `Theme` 来定制风格，抽取所有的字串等信息进行本地化设计。Android 的界面需要写在“`res/layout`”下面的布局文件中，一般情况下，一个布局文件对应一个界面。设计 Android 的界面有点像写 HTML 代码，要先给 Android 定框架，然后再往框架里面放控件。本章将详细介绍这些框架和能够放在框架中的常用控件。本章可以作为这些框架和控件的查询手册，可供你在开发过程中随时查阅。

4.1 用户界面开发详解

Android 应用程序的基础功能单元就是 `Activity` 类中的一个对象。`Activity` 可以做很多事，比如界面显示、事件处理等。Android 应用程序界面通常使用 `View` 和 `ViewGroup` 控件配 XML 样式来进行设计；而事件则包括按钮事件、触屏事件以及一些高级控件的事件监听。下面我们分别介绍 `Activity` 的界面设计、显示和事件处理。

4.1.1 用户界面简介

Android 生成屏幕有三种方式：xml 配置生成；通过用户界面接口生成；直接用代码生成。在一个 Android 应用中，用户界面是由 `View` 和 `ViewGroup` 对象构建的。`View` 与 `ViewGroup` 都有很多种类，而它们都是 `View` 类的子类。开发者可以对 `View` 和 `ViewGroup` 进行组合，来完成应用程序界面设计。下面我们分别来讨论 `View` 和 `ViewGroup`。

1. View

任何一个 `View` 对象都将继承 `android.view.View` 类。它是一个存储有屏幕上特定的一个矩形布局和内容属性的数据结构。一个 `View` 对象可以处理测距、布局、绘图、焦点变换、滚动条，以及屏幕区域自己表现的按键和手势。作为一个基类，`View` 类为 `Widget` 服务，`Widget` 则是一组用于绘制交互屏幕元素的完全实现子类。`Widget` 处理自己的测距和绘图，所以可以快速用它们去构建 UI。可用到的 `Widget` 包括 `Text`、`EditText`、`Button`、`RadioButton`、`Checkbox` 和 `ScrollView` 等。

2. ViewGroup

ViewGroup 是一个 `android.view.ViewGroup` 类的对象。顾名思义，**ViewGroup** 是一个特殊的 **View** 对象，它的功能是装载和管理一组下层的 **View** 和其他 **ViewGroup**，**ViewGroup** 可以为 UI 增加结构，并且将复杂的屏幕元素构建成一个独立的实体。作为一个基类，**ViewGroup** 为 **Layout**（布局）服务，**Layout** 则是一组提供屏幕界面通用类型的完全实现子类。**Layout** 可以为一组 **View** 构建一个结构。

图 4-1 是一个由 **View** 和 **ViewGroup** 布局的 **Activity** 界面。

从图 4-1 中可以看出，一个 **Activity** 界面可以包含多个 **ViewGroup** 和 **View**，通过这样的组合可以实现更复杂、更完美、更满足开发者需要的界面。

当 **Activity** 调用它的 `setContentView()` 方法并且传递一个参数给根节点对象时，一旦 **Android** 系统获得了根节点的参数，它就可以直接通过节点来测距和绘制树。当 **Activity** 被激活并获得焦点时，系统会通知 **Activity** 并且请求根节点测距并绘制树，根节点就会请求它的子节点去绘制它们自己。同时，每个树上的 **ViewGroup** 节点负责绘制它的直接子节点。正如之前提到的，每个 **ViewGroup** 都有测量它的有效空间、布局它的子对象并且调用每个子对象的 `Draw()` 方法去绘制它们自己。子对象可能会请求获得它们在父对象中的大小和位置，但是父对象对每个子对象的大小和位置有最终的决定权。

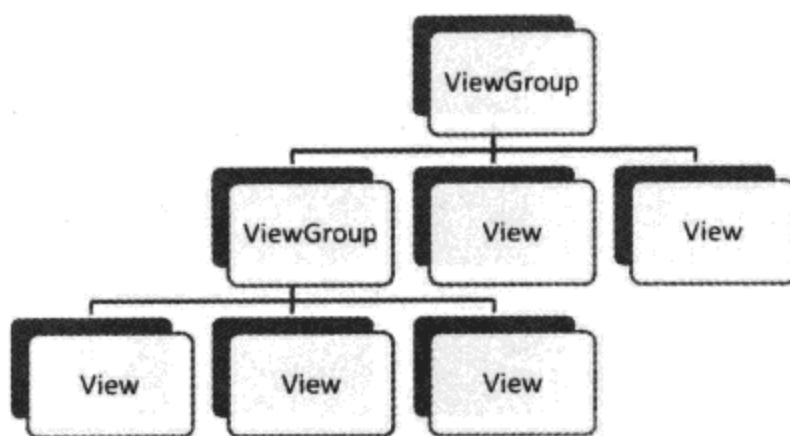


图 4-1 View 和 ViewGroup 混合布局的 **Activity** 界面

4.1.2 事件处理

什么是事件呢？事件就是用户与 UI（图形界面）交互时所触发的操作。例如，在手机键盘上按下一个键，就可以触发几个事件。键盘上的键被按下时就触发了“按下”事件，当松开按键时又会触发“弹起”事件。在 **Android** 中，这些事件都将被传送到事件处理器，它是一个专门接受事件对象并对其进行翻译和处理的方法。

在 **Java** 程序中，实现与用户交互功能的控件都需要通过事件来处理，需要指定控件所用的事件监听器。当然，**Android** 同样需要设置事件监听器。另外，在 **Android** 下，**View** 同样可以响应按键和触屏两种事件，分别如下所示。

- ❑ `boolean onKeyDown(int keyCode, KeyEvent event)` 用于响应按键按下。
- ❑ `boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)` 用于响应按键重复点击。
官方 API 指出 `onKeyMultiple` 方法总是返回 `false`，即它没有 `handle`，因此必须重写才能实现。
- ❑ `boolean onKeyUp(int keyCode, KeyEvent event)` 用于响应按键释放，`onTouchEvent(MotionEvent event)` 用于响应触摸屏事件。

下面我们通过一个示例程序（具体参见本书所附代码：第 4 章\Examples_04_01）来处理各种

事件, 主要包括: Button 控件事件监听、按键按下事件、按键弹起事件、触笔点击事件。示例成功运行后, 点击“OK”按钮后如图 4-2 所示, 按键“按下”时如图 4-3 所示, 按键“弹起”时如图 4-4 所示, 触笔点击屏幕时如图 4-5 所示。具体实现如代码清单 4-1 所示。



图 4-2 点击“OK”按钮



图 4-3 按键“按下”状态

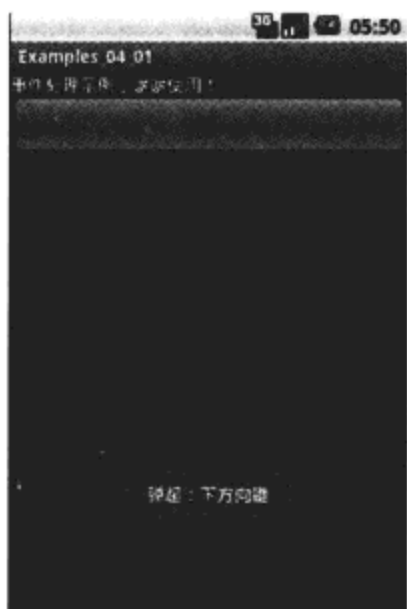


图 4-4 按键“弹起”状态



图 4-5 触笔点击事件

下面分析如何处理这些事件, 在 Android 中, 控件事件是通过设置其控件的监听器来监听并重写某些函数来处理的。具体实现请参见本书所附代码: 第 4 章\Examples_04_01。这里需要说明的是, 我们使用了

```
Toast.makeText(this, string, Toast.LENGTH_SHORT).show();
```

来显示一个短时间的提示信息。

代码清单 4-1 第 4 章\Examples_04_01\src\com\yarin\android\Examples_04_01\Activity01.java

```
/*
```

- * 控件事件通过设置其控件的监听器来监听并处理事件
- * 按键按下事件: 通过重写 onKeyDown 方法
- * 按键弹起事件: 通过重写 onKeyUp 方法

```

* 触笔点击事件: 通过实现 onTouchEvent 方法
*/
public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //获得 Button 对象
        Button button_ok = (Button) findViewById(R.id.ok);
        //设置 Button 控件监听器
        button_ok.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v)
            {
                //这里处理事件
                DisplayToast("点击了 OK 按钮");
            }
        });
    }
    /* 按键按下所触发的事件 */
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_CENTER:
                DisplayToast("按下: 中键");
                break;
            case KeyEvent.KEYCODE_DPAD_UP:
                DisplayToast("按下: 上方向键");
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                DisplayToast("按下: 下方向键");
                break;
            case KeyEvent.KEYCODE_DPAD_LEFT:
                DisplayToast("按下: 左方向键");
                break;
            case KeyEvent.KEYCODE_DPAD_RIGHT:
                DisplayToast("按下: 右方向键");
                break;
        }
        return super.onKeyDown(keyCode, event);
    }
    /* 按键弹起所触发的事件 */
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_CENTER:
                DisplayToast("弹起: 中键");
                break;
            case KeyEvent.KEYCODE_DPAD_UP:
                DisplayToast("弹起: 上方向键");
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                DisplayToast("弹起: 下方向键");

```



```

        break;
    case KeyEvent.KEYCODE_DPAD_LEFT:
        DisplayToast("弹起: 左方向键");
        break;
    case KeyEvent.KEYCODE_DPAD_RIGHT:
        DisplayToast("弹起: 右方向键");
        break;
    }
    return super.onKeyUp(keyCode, event);
}
public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
{
    return super.onKeyMultiple(keyCode, repeatCount, event);
}
/* 触笔事件 */
public boolean onTouchEvent(MotionEvent event)
{
    int iAction = event.getAction();
    if (iAction == MotionEvent.ACTION_CANCEL ||
        iAction == MotionEvent.ACTION_DOWN ||
        iAction == MotionEvent.ACTION_MOVE)
    {
        return false;
    }
    //得到触笔点击的位置
    int x = (int) event.getX();
    int y = (int) event.getY();

    DisplayToast("触笔点击坐标: (" + Integer.toString(x) + ", " + Integer.toString(y) + ")");
    return super.onTouchEvent(event);
}
/* 显示 Toast */
public void DisplayToast(String str)
{
    Toast.makeText(this, str, Toast.LENGTH_SHORT).show();
}
}

```

在 Examples_04_01 项目中, 我们分析了一些常用的事件处理方式。每个键都对应一个键值。当然, 可以根据需要来改变一些键的功能, 这就需要我们自己构建 KeyEvent 对象, 构造 KeyEvent 对象的方式有如下几种:

- ☐ KeyEvent(int action,int code);
- ☐ KeyEvent(long DownTime,long EventTime,int action,int code,int repeat);
- ☐ KeyEvent(long DownTime,long EventTime,int action,int code,int repeat,int metState);
- ☐ KeyEvent(long DownTime,long EventTime,int action,int code,int repeat,int metState,int device,int scancode);
- ☐ KeyEvent(long DownTime,long EventTime,int action,int code,int repeat,int metState,int device,int scancode,int flags);
- ☐ KeyEvent(KeyEvent origEvent,long EventTime,int newReport)。

下面我们通过一个示例 (参见本书所附代码: 第 4 章\Examples_04_02) 来改变所有按键的功能都为“返回键”的功能。这里我们将使用构造方法来构造一个 KeyEvent 对象, 让其键值为

“`KeyEvent.KEYCODE_BACK`”，这样当我们在按任意键时就都执行“退出”功能。具体实现如代码清单 4-2 所示。

代码清单 4-2 第4章\Examples_04_02\src\com\yarin\android\Examples_04_02\Activity01.java

```
public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        //这里构建 KeyEvent 对象，其功能为返回键的功能
        //因此我们按任意键都会执行返回键功能
        KeyEvent key = new KeyEvent(KeyEvent.ACTION_DOWN, KeyEvent.KEYCODE_BACK);
        //这里传入的参数就是我们自己构建的 KeyEvent 对象 key
        return super.onKeyDown(key.getKeyCode(), key);
    }
}
```

在 Examples_04_02 项目中，我们只使用 `KeyEvent(int action,int code)`这个方法来构造 `KeyEvent` 对象。当然，大家在以后的学习和使用中，可以根据需要使用不同的构造方法构造满足自己要求的 `KeyEvent` 对象。下面介绍常用 Android 组件的功能和使用方法。

4.2 常用控件应用

应用程序的人机交互界面由很多 Android 控件组成，前面章节的所有示例的界面都是由一些常用的控件构成的。Android 的控件可以说是目前所有手机平台控件中最完美的，下面我们将对这些控件进行详细介绍。Android 常用的控件如图 4-6 所示。

4.2.1 文本框 (TextView)

我们知道，前面的“HelloAndroid”工程中就是用 `TextView` 来显示一个文本。`TextView` 就是一个用来显示文本标签的控件，下面我们吧“HelloAndroid”改写成“硬编码”实现（参见本书所附代码：第4章\Examples_04_03），从而修改使用 `TextView` 显示的文本的颜色、大小等属性，运行效果如图 4-7 所示。具体实现如代码清单 4-3 所示。

代码清单 4-3 第4章\Examples_04_02\src\com\yarin\android\Examples_04_03\Activity01.java

```
public class Activity01 extends Activity
{
    /* 声明 TextView 对象 */
    private TextView textview;
    public void onCreate(Bundle savedInstanceState)
    {
```



图 4-6 Android 常用控件

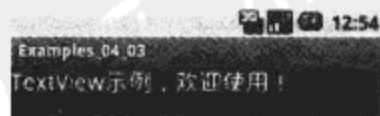


图 4-7 TextView 控件使用

```

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    /* 获得 TextView 对象 */
    textview = (TextView) this.findViewById(R.id.textview);
    String string = "TextView 示例, 欢迎使用! ";
    /* 设置文本的颜色 */
    textview.setTextColor(Color.RED);
    /* 设置字体大小 */
    textview.setTextSize(20);
    /* 设置文字背景 */
    textview.setBackgroundColor(Color.BLUE);
    /* 设置 TextView 显示的文字 */
    textview.setText(string);
}
}

```

因为在代码清单 4-3 中使用了 `findViewById` 来获得 `TextView` 对象, 因此在布局文件“main.xml”中需要指定 `TextView` 资源的 ID, 如代码清单 4-4 所示。

代码清单 4-4 第 4 章\Examples_04_03\res\layout\ main.xml

```

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

```

Examples_04_03 项目同样可以使用布局来实现, 这就交给大家自己来完成吧。下面提供了几个常用的方法在布局中定义的参数, 如表 4-1 所示。

表 4-1 TextView 对象方法与对应的 XML 属性

TextView 对象方法	XML 属性
<code>setTextColor</code>	<code>android:textColor</code>
<code>setTextSize</code>	<code>android:textSize</code>
<code>setText</code>	<code>android:text</code>
<code>setBackgroundResource</code>	<code>android:background</code>
<code>setHeight/setWidth</code>	<code>android:height/android:width</code>

4.2.2 列表 (ListView)

在 Android 中, `ListView` 用来显示一个列表的控件。当然, 用户可以选择并操作这个列表, 同时必然会触发一些事件。当鼠标滚动时会触发 `setOnItemSelectedListener` 事件, 点击时则会产生 `setOnItemClickListener` 事件。下面我们通过一个示例 (参见本书所附代码: 第 4 章\Examples_04_04) 来分析 `ListView` 的使用。

下面我们用 `ListView` 来显示电话本中的信息, 并处理鼠标滚动及点击事件。程序运行前, 我们先来查看电话本中的数据, 如图 4-8 所示; 然后运行程序之后, 我们来滚动鼠标中键, 触发 `setOnItemSelectedListener` 事件的处理, 如图 4-9 所示; 最后我们点击 `ListView` 列表中的一项来触发

setOnItemClickListener 事件, 如图 4-10 所示。



图 4-8 电话本信息



图 4-9 setOnItemSelectedListener 事件



图 4-10 setOnItemClickListener 事件

在编码实现时, 我们先创 `LinearLayout` 对象和 `ListView` 对象, `LinearLayout` 用来显示 `ListView`; 然后通过 `ListAdapter` 将获得的电话本数据与 `ListView` 连接起来; 接着将 `ListAdapter` 添加到 `ListView` 中; 最后将 `ListView` 添加到 `LinearLayout` 中, 让屏幕显示 `LinearLayout`。要处理 `ListView` 事件, 需要为 `ListView` 视图添加 `setOnItemSelectedListener` 监听以及 `setOnItemClickListener` 监听。如代码清单 4-5 所示。

代码清单 4-5 第 4 章\Examples_04_04\src\com\yarin\android\Examples_04_04\Activity01.java

```
package com.yarin.android.Examples_04_04;
```

```
public class Activity01 extends Activity
```

```
{
```

```
    LinearLayout m_LinearLayout;
```

```
    ListView      m_ListView;
```

```
    public void onCreate(Bundle savedInstanceState)
```

```
{
```

```
        super.onCreate(savedInstanceState);
```

```
        /* 创建 LinearLayout 布局对象 */
```

```
        m_LinearLayout = new LinearLayout(this);
```

```
        /* 设置布局 LinearLayout 的属性 */
```

```
        m_LinearLayout.setOrientation(LinearLayout.VERTICAL);
```

```
        m_LinearLayout.setBackgroundColor(android.graphics.Color.BLACK);
```

```
        /* 创建 ListView 对象 */
```

```
        m_ListView = new ListView(this);
```

```
        LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(LinearLayout.
            LayoutParams.FILL_PARENT, LinearLayout.LayoutParams.WRAP_CONTENT);
```

```
        m_ListView.setBackgroundColor(Color.BLACK);
```

```
        /* 添加 m_ListView 到 m_LinearLayout 布局 */
```

```
        m_LinearLayout.addView(m_ListView, param);
```

```
        /* 设置显示 m_LinearLayout 布局 */
```

```
        setContentView(m_LinearLayout);
```

```
        // 获取数据库 Phones 的 Cursor
```

```
        Cursor cur = getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
            null, null, null, null);
```

```
        startManagingCursor(cur);
```

```
        // ListAdapter 是 ListView 和后台数据的桥梁
```

```
        ListAdapter adapter = new SimpleCursorAdapter(this,
```

```
            // 定义 List 中每一行的显示模板
```

```

        // 表示每一行包含两个数据项
        android.R.layout.simple_list_item_2,
        // 数据库的 Cursor 对象
        cur,
        // 从数据库的 NAME 和 NUMBER 两列中取数据
        new String[] { PhoneLookup.DISPLAY_NAME, PhoneLookup.NUMBER },
        // 与 NAME 和 NUMBER 对应的 Views
        new int[] { android.R.id.text1, android.R.id.text2 });
    /* 将 adapter 添加到 m_ListView 中 */
    m_ListView.setAdapter(adapter);
    /* 为 m_ListView 视图添加 setOnItemSelectedListener 监听 */
    m_ListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3)
        {
            DisplayToast("滚动到第"+Long.toString(arg0.getSelectedItemId())+"项");
        }
        public void onNothingSelected(AdapterView<?> arg0)
        {
            //没有选中
        }
    });

    /* 为 m_ListView 视图添加 setOnItemClickListener 监听 */
    m_ListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3)
        {
            //对于选中的项进行处理
            DisplayToast("选中了第"+Integer.toString(arg2+1)+"项");
        }
    });
}

/* 显示 Toast */
public void DisplayToast(String str)
{
    Toast.makeText(this, str, Toast.LENGTH_SHORT).show();
}
}

```

因为在工程中我们获得了电话本数据，所以需要在 AndroidManifest.xml 中加入“<uses-permission android:name="android.permission.READ_CONTACTS" />”，如代码清单 4-6 所示。

代码清单 4-6 第 4 章\Examples_04_04\AndroidManifest.xml

```

<application
    android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Activity01"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
<uses-permission android:name="android.permission.READ_CONTACTS" />

```


4.2.3 提示 (Toast)

Toast 是 Android 提供的“快显讯息”类，Toast 类的使用非常简单，而且用途很多。比如，当退出应用程序时，可以用它来提示玩家“需要更新”，或者当在输入框中输入文本时，可以提示玩家“最多能输入 20 个字符”等。下面我们还是通过一个例子（参见本书所附代码：第 4 章\Examples_04_05）来分析 Toast 的使用，该例子实现的功能是：当收到短信时，通过 Toast 来提示用户，显示短信内容。首先运行程序，结果如图 4-11 所示；然后在 Eclipse 编辑器中切换到 DDMS 标签，这里可以给模拟器发送一条短信，如图 4-12 所示；当模拟器收到短信时，通过 Toast 来显示短信内容，如图 4-13 所示。

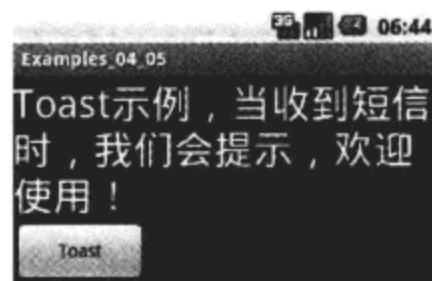


图 4-11 Toast 示例

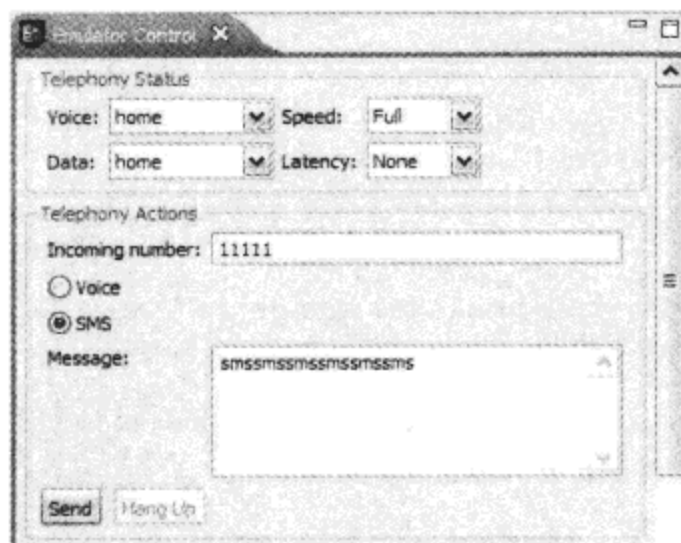


图 4-12 DDMS 中给模拟器发送短信



图 4-13 模拟器收到短信后的提示界面

在该例子中，Toast 的实现很简单，只有一行代码“`Toast.makeText(this, String, Toast.LENGTH_SHORT).show();`”，如代码清单 4-7 中的 `DisplayToast` 方法。

代码清单 4-7 第 4 章\Examples_04_05\src\com\yarin\android\Examples_04_05\Activity01.java

```
public class Activity01 extends Activity {

    private TextView textview;

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

        textview = (TextView) this.findViewById(R.id.tv1);

        String string = "Toast 示例，当收到短信时，我们会提示，欢迎使用！";
        textview.setTextSize(30);
        textview.setText(string);

        Button button = (Button) findViewById(R.id.button1);
        /* 监听 button 的事件信息 */
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v)
            {

```

```

        DisplayToast("短信内容在这里显示");
    }
});
}

/* 显示 Toast */
public void DisplayToast(String str)
{
    Toast.makeText(this, str, Toast.LENGTH_SHORT).show();
}
}

```

这里还需要创建一个专门接收短信的类 `SMSReceiver` 继承自 `BroadcastReceiver` 类, 需要重写 `onReceive` 方法。当收到短信时就会自动触发 `onReceive` 方法, 因此我们就在 `onReceive` 方法中通过 `Toast` 来显示短信的内容, 如代码清单 4-8 所示。

代码清单 4-8 第 4 章\Examples_04_05\src\com\yarin\android\Examples_04_05\SMSReceiver.java

```

public class SMSReceiver extends BroadcastReceiver
{
    /*当收到短信时, 就会触发此方法*/
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = intent.getExtras();
        Object messages[] = (Object[]) bundle.get("pdus");
        SmsMessage smsMessage[] = new SmsMessage[messages.length];
        for (int n = 0; n < messages.length; n++)
        {
            smsMessage[n] = SmsMessage.createFromPdu((byte[]) messages[n]);
        }
        //产生一个 Toast
        Toast toast = Toast.makeText(context, "短信内容: " + smsMessage[0].getMessageBody(), Toast.LENGTH_LONG);
        //设置 Toast 显示的位置
        //toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 200);
        //显示该 Toast
        toast.show();
    }
}

```

由于在 `Examples_04_05` 项目中使用了短信接口, 所以需要在 `AndroidManifest.xml` 中声明其权限, 如代码清单 4-9 所示。

代码清单 4-9 第 4 章\Examples_04_05\AndroidManifest.xml

```

<uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
<application
    android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Activity01"
        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=".SMSReceiver" android:enabled="true">
        <intent-filter>
            <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
        </intent-filter>
    </receiver>
</application>

```

```
</receiver>
</application>
```

4.2.4 编辑框 (EditText)

EditText 在开发中也是经常使用的控件, 比如, 要实现一个登录界面, 需要用户输入账号、密码等信息, 然后我们获得用户输入的内容, 把它交给服务器来判断。因此, 这一节我们需要学习如何在布局文件中实现编辑框, 然后获得编辑框的内容。我们同样通过一个示例 (参见本书所附代码: 第4章\Examples_04_06) 来向大家展示, 运行效果如图4-14所示。当用户输入信息后, 我们获得用户输入的内容, 显示在一个 TextView 上, 如图4-15所示。

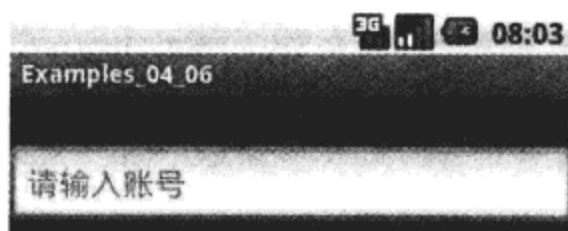


图4-14 EditText 示例

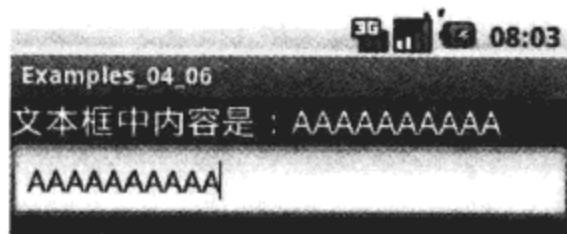


图4-15 获得 EditText 中的内容

在该示例中, 我们通过 XML 定义了一个 EditText 控件, 然后为其设置了事件监听 setOnKeyListener, 并实现了 onKey 方法, 也就是当用户在按键时便会触发这个事件, 从而可通过 getText() 方法取得用户输入的内容。当然, 在图4-14中, 在用户没有输入的时候, 我们默认在编辑框中显示了“请输入账号”的提示, 要实现这一功能很简单, 只需要“EditText.setHint(“请输入账号”);”或者在 XML 布局文件中写上“android:hint=“请输入账号””即可。下面我们来看一下具体的代码实现, 如代码清单4-10所示。

代码清单4-10 第4章\Examples_04_06\src\com\yarin\android\Examples_04_06\Activity01.java

```
public class Activity01 extends Activity
{
    private TextView m_TextView;
    private EditText m_EditText;

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

        m_TextView = (TextView) findViewById(R.id.TextView01);
        m_EditText = (EditText) findViewById(R.id.EditText01);
        m_TextView.setTextSize(20);
        /**
         * 设置当 m_EditText 中为空时提示的内容
         * 在 XML 中同样可以实现: android:hint="请输入账号"
         */
        m_EditText.setHint("请输入账号");
        /* 设置 EditText 事件监听 */
        m_EditText.setOnKeyListener(new EditText.OnKeyListener() {
            @Override
            public boolean onKey(View arg0, int arg1, KeyEvent arg2)
            {
```

```

        // 得到文字, 将其显示到 TextView 中
        m_TextView.setText("文本框中内容是: " + m_EditText.getText().toString());
        return false;
    }
});
}
}

```

该例子中界面通过 XML 布局实现, 如代码清单 4-11 所示。

代码清单 4-11 第 4 章\Examples_04_06\res\layout\main.xml

```

<?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/TextView01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <EditText
        android:id="@+id/EditText01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_x="29px"
        android:layout_y="33px"
    />
</LinearLayout>

```

4.2.5 单项选择 (RadioGroup、RadioButton)

单项选择相信大家都不陌生吧。Android 平台也提供了单项选择的组件, 可以通过 RadioGroup、RadioButton 组合起来完成一个单项选择效果。我们仍然通过一个示例 (参见本书所附代码: 第 4 章\Examples_04_07) 来说明该功能是如何实现的, 示例的运行结果如图 4-16 所示。这时用户需要选择, 如果用户选择的答案不符合题意, 结果如图 4-17 所示; 当用户选择了正确的答案之后, 结果如图 4-18 所示。



图 4-16 单项选择界面



图 4-17 回答错误



图 4-18 回答正确

其实, 要实现这样的效果并不复杂, 从运行结果可以看出, 一个单项选择由两部分组成, 分别

是前面的选择按钮和后面的“答案”。Android 平台上的选择按钮可通过 `RadioButton` 来实现，而“答案”则通过 `RadioGroup` 来实现。因此，首先要在布局文件中定义一个 `RadioGroup` 和 4 个 `RadioButton`，在定义 `RadioGroup` 时，已经将“答案”赋给了每个选项，那么如何确定用户的选择是否正确呢？这需要在用户点击时来判断用户选择的是哪一项，所以需要设置其事件监听 `setOnCheckedChangeListener`。如代码清单 4-12 所示。

代码清单 4-12 第 4 章\Examples_04_07\src\com\yarin\android\Examples_04_07\Activity01.java

```
public class Activity01 extends Activity
{
    /**
     * 创建 TextView 对象
     * 创建 RadioGroup 对象
     * 创建 4 个 RadioButton 对象
     */
    TextView    m_TextView;
    RadioGroup   m_RadioGroup;
    RadioButton m_Radio1, m_Radio2, m_Radio3, m_Radio4;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        /**
         * 获得 TextView 对象
         * 获得 RadioGroup 对象
         * 获得 4 个 RadioButton 对象
         */
        m_TextView = (TextView) findViewById(R.id.TextView01);
        m_RadioGroup = (RadioGroup) findViewById(R.id.RadioGroup01);
        m_Radio1 = (RadioButton) findViewById(R.id.RadioButton1);
        m_Radio2 = (RadioButton) findViewById(R.id.RadioButton2);
        m_Radio3 = (RadioButton) findViewById(R.id.RadioButton3);
        m_Radio4 = (RadioButton) findViewById(R.id.RadioButton4);

        /* 设置事件监听 */
        m_RadioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId)
            {
                // TODO: Auto-generated method stub
                if (checkedId == m_Radio2.getId())
                {
                    DisplayToast("正确答案: " + m_Radio2.getText() + ", 恭喜你, 回答正确!");
                }
                else
                {
                    DisplayToast("请注意, 回答错误!");
                }
            }
        });
    }

    /* 显示 Toast */
    public void DisplayToast(String str)
    {

```



```
        Toast toast = Toast.makeText(this, str, Toast.LENGTH_LONG);  
        //设置 Toast 显示的位置  
        toast.setGravity(Gravity.TOP, 0, 220);  
        //显示该 Toast  
        toast.show();  
    }  
}
```

当然，这里的题目我们是通过一个 `TextView` 来实现的，下面是实现这个单选项的整个布局文件，如代码清单 4-13 所示。

代码清单 4-13 第 4 章\Examples_04_07\res\layout\main.xml

```
<?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/TextView01"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/hello"  
    />  
    <RadioGroup  
        android:id="@+id/RadioGroup01"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:orientation="vertical"  
        android:layout_x="3px"  
        android:layout_y="54px"  
    >  
        <RadioButton  
            android:id="@+id/RadioButton1"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="@string/RadioButton1"  
        />  
        <RadioButton  
            android:id="@+id/RadioButton2"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="@string/RadioButton2"  
        />  
        <RadioButton  
            android:id="@+id/RadioButton3"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="@string/RadioButton3"  
        />  
        <RadioButton  
            android:id="@+id/RadioButton4"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="@string/RadioButton4"  
        />  
    </RadioGroup>  
</LinearLayout>
```

4.2.6 多项选择 (CheckBox)

上面我们实现了单项选择,那么多项选择又如何实现呢,多项选择与单项选择最重要的区别就在于它可以让用户选择一个以上的选项。Android 平台提供了 `CheckBox` 来实现多项选择。这里需要注意,既然用户可以选择多项,那么为了确定用户是否选择了某一项,需要对每一个选项进行事件监听。还是先来看一个例子(参见本书所附代码:第4章\Examples_04_08),其运行效果如图4-19所示。当用户选择一项之后提示用户所选择的内容,如图4-20所示;当用户选择完毕并提交之后需要给用户反馈信息,如图4-21所示。



图4-19 多项选择界面



图4-20 选择选项后反馈给用户



图4-21 选择完毕提交之后

该示例的实现过程与单项选择相类似,先在布局文件中定义 `CheckBox` 来实现多项选择,然后对每一项设置事件监听 `setOnCheckedChangeListener`,通过 `isChecked` 来判断选项是否被选中,如代码清单4-14所示。

代码清单4-14 第4章\Examples_04_08\src\com\lyarin\android\Examples_04_08\Activity01.java

```
public class Activity01 extends Activity
{
    //用来显示题目
    TextView    m_TextView1;
    //提交按钮
    Button      m_Button1;
    //4个多选题
    CheckBox    m_CheckBox1;
    CheckBox    m_CheckBox2;
    CheckBox    m_CheckBox3;
    CheckBox    m_CheckBox4;

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

        m_TextView1 = (TextView) findViewById(R.id.TextView1);
        m_Button1 = (Button) findViewById(R.id.button1);

        /* 取得每个 CheckBox 对象 */
        m_CheckBox1 = (CheckBox) findViewById(R.id.CheckBox1);
```

```

m_CheckBox2 = (CheckBox) findViewById(R.id.CheckBox2);
m_CheckBox3 = (CheckBox) findViewById(R.id.CheckBox3);
m_CheckBox4 = (CheckBox) findViewById(R.id.CheckBox4);

//对每个选项设置事件监听
m_CheckBox1.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
    {
        if(m_CheckBox1.isChecked())
        {
            DisplayToast("你选择了: "+m_CheckBox1.getText());
        }
    }
});
//////////
m_CheckBox2.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
    {
        if(m_CheckBox2.isChecked())
        {
            DisplayToast("你选择了: "+m_CheckBox2.getText());
        }
    }
});
//////////
m_CheckBox3.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
    {
        if(m_CheckBox3.isChecked())
        {
            DisplayToast("你选择了: "+m_CheckBox3.getText());
        }
    }
});
//////////
m_CheckBox4.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
    {
        if(m_CheckBox4.isChecked())
        {
            DisplayToast("你选择了: "+m_CheckBox4.getText());
        }
    }
});
//对按钮设置事件监听
m_Button1.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v)
    {
        int num = 0;
        if(m_CheckBox1.isChecked())
        {
            num++;
        }
        if(m_CheckBox2.isChecked())

```

```

        {
            num++;
        }
        if (m_CheckBox3.isChecked())
        {
            num++;
        }
        if (m_CheckBox4.isChecked())
        {
            num++;
        }
        DisplayToast("谢谢参与! 你一共选择了"+num+"项!");
    }
    });
}

/* 显示 Toast */
public void DisplayToast(String str)
{
    Toast toast = Toast.makeText(this, str, Toast.LENGTH_SHORT);
    //设置 Toast 显示的位置
    toast.setGravity(Gravity.TOP, 0, 220);
    //显示该 Toast
    toast.show();
}
}

```

通过 XML 布局来实现多项选择的界面, 如代码清单 4-15 所示。

代码清单 4-15 第4章\Examples_04_08\res\layout\main.xml

```

<?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/TextView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
    <CheckBox
        android:id="@+id/CheckBox1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/CheckBox1"
        >
    </CheckBox>
    <CheckBox
        android:id="@+id/CheckBox2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/CheckBox2"
        >
    </CheckBox>
    <CheckBox
        android:id="@+id/CheckBox3"
        android:layout_width="fill_parent"

```

```

    android:layout_height="wrap_content"
    android:text="@string/CheckBox3"
  >
</CheckBox>
<CheckBox
    android:id="@+id/CheckBox4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/CheckBox4"
  >
</CheckBox>
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="提交"
    >
</Button>
</LinearLayout>

```

所有的选项的字符串都定义在了 String.xml 文件中, 请参见本书所附代码: 第 4 章\Examples_04_08\res\values\String.xml。

4.2.7 下拉列表 (Spinner)

当我们在某个网站注册账号时, 网站可能会让我们提供性别、生日、城市等信息。网站开发人员为了方便用户, 不让用户填写这些信息, 而是提供一个下拉列表将所有可选的项列出来, 供用户选择。在 Android 上能轻松地实现这一功能, 这就是本小节要介绍的 Spinner。下面我们通过一个让用户选择自己的血型的示例来分析 Spinner 的具体用法 (参见本书所附代码: 第 4 章\Examples_04_09), 运行效果如图 4-22 所示。当用户点击下拉列表时, 如图 4-23 所示, 我们获得用户选择的内容并显示在 TextView 上。

编码实现时, 首先需要在布局中定时 Spinner 组件, 然后将可选内容通过 ArrayAdapter 和下拉列表连接起来, 最后要获得用户选择的选项, 我们需要设计事件监听 setOnItemSelectedListener 并实现 onItemSelected, 从而获得用户所选择的内容, 最后通过 setVisibility 方法设置当前的显示项, 如代码清单 4-16 所示。

代码清单 4-16 第 4 章\Examples_04_09\src\com\yarin\android\Examples_04_09\Activity01.java

```

public class Activity01 extends Activity
{
    private static final String[] m_Countries = { "O 型",
    "A 型", "B 型", "AB 型", "其他" };

    private TextView    m_TextView;
    private Spinner     m_Spinner;

```

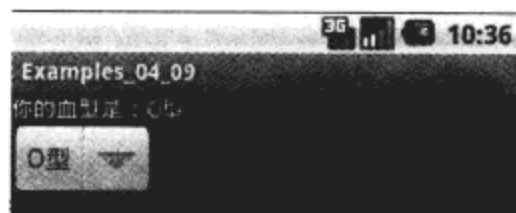


图 4-22 Spinner 下拉列表

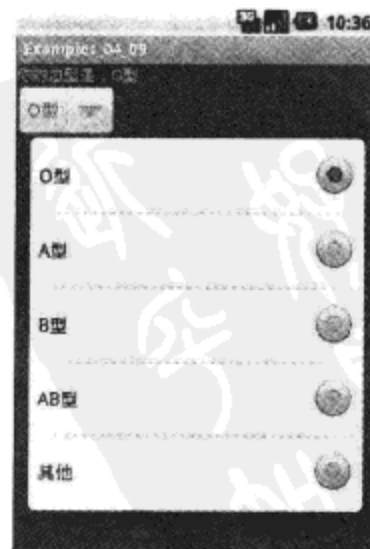


图 4-23 选择列表项


```

private ArrayAdapter<String> adapter;

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

    m_TextView = (TextView) findViewById(R.id.TextView1);
    m_Spinner = (Spinner) findViewById(R.id.Spinner1);

    //将可选内容与 ArrayAdapter 连接
    adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item,
    m_Countries);

    //设置下拉列表的风格
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    //将 adapter 添加到 m_Spinner 中
    m_Spinner.setAdapter(adapter);

    //添加 Spinner 事件监听
    m_Spinner.setOnItemSelectedListener(new Spinner.OnItemSelectedListener() {

        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3)
        {
            m_TextView.setText("你的血型是: " + m_Countries[arg2]);
            //设置显示当前选择的项
            arg0.setVisibility(View.VISIBLE);
        }

        @Override
        public void onNothingSelected(AdapterView<?> arg0)
        {
            // TODO Auto-generated method stub
        }

    });
}

```

该例子的 XML 布局文件如代码清单 4-17 所示。

代码清单 4-17 第4章\Examples_04_09\res\layout\main.xml

```

<?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/TextView1"
        android:layout_width="fill_parent"

```

```

        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
<Spinner
    android:id="@+id/Spinner1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
/>
</LinearLayout>

```

4.2.8 自动提示 (AutoCompleteTextView)

大家都对 Google 的搜索提示很熟悉吧, 当我们在搜索框中输入要搜索的内容时, Google 会自动提示很多与我们的输入接近的内容供选择。除了 Google, 我们在使用 Eclipse 时, 为了方便, 也可以设置自动提示的关键词, 当设置好之后, 在 Eclipse 中输入时, 它便会自动提示, 这样大大提高了编程的效率。这一特色功能 Android 肯定少不了, 下面我们就分别通过 AutoCompleteTextView 和 MultiAutoCompleteTextView 来实现这一功能 (参见本书所附代码: 第 4 章 \Examples_04_10), 运行效果如图 4-24 所示。当我们输入时, 自动提示效果如图 4-25 所示。

实现过程非常简单, 首先在布局文件中创建 AutoCompleteTextView, 然后通过 AutoCompleteTextView 将关键字和 AutoCompleteTextView 连接, 当我们输入时, 就会自动提示了, 具体实现如代码清单 4-18 所示。

代码清单 4-18 第 4 章 \Examples_04_10 \src \com \yarin \android \Examples_04_10 \Activity01.java

```

public class Activity01 extends Activity
{
    private static final String[] autoString= new String[]
    { "a2", "abf", "abe", "abcde", "abc2", "abcd3",
      "abcde2", "abc2", "abcd2", "abcde2" };

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

        //关联关键字
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, autoString);

        AutoCompleteTextView m_AutoCompleteTextView = (AutoCompleteTextView)
        findViewById(R.id.AutoCompleteTextView01);
    }
}

```

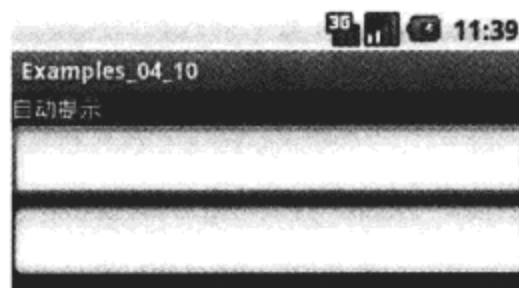


图 4-24 自动提示输入框

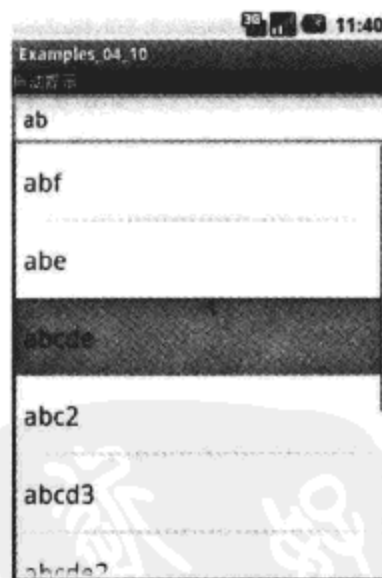


图 4-25 自动提示

```

//将 adapter 添加到 AutoCompleteTextView 中
m_AutoCompleteTextView.setAdapter(adapter);
//////////
MultiAutoCompleteTextView mm_AutoCompleteTextView = (MultiAutoCompleteTextView)
findViewById(R.id.MultiAutoCompleteTextView01);
//将 adapter 添加到 AutoCompleteTextView 中
mm_AutoCompleteTextView.setAdapter(adapter);
mm_AutoCompleteTextView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
}
}

```

AutoCompleteTextView 和 MultiAutoCompleteTextView 的 XML 布局如代码清单 4-19 所示。

代码清单 4-19 第 4 章\Examples_04_10\res\layout\main.xml

```

<?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"
        />
    <AutoCompleteTextView
        android:id="@+id/AutoCompleteTextView01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        >
    </AutoCompleteTextView>
    <MultiAutoCompleteTextView
        android:id="@+id/MultiAutoCompleteTextView01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        >
    </MultiAutoCompleteTextView>
</LinearLayout>

```

4.2.9 日期和时间 (DatePicker、TimePicker)

日期和时间太常见了，是任何手机都会有的基本功能，Android 也不例外。Android 平台用 DatePicker 来实现日期，用 TimePicker 来实现时间。Android 平台的日期、时间界面非常漂亮，本节例子（参见本书所附代码：第 4 章\Examples_04_11）效果如图 4-26 所示。设置日期界面如图 4-27 所示，设置时间界面如图 4-26 所示。

如此漂亮的界面，实现起来却非常简单。首先需要在布局文件中定义 DatePicker 和 TimePicker，然后通过 Calendar 类获得系统时间，接着通过 init 方法将日期传递给 DatePicker，并设置 OnDateChangeListener 来监听日期改变，当时间被改变时需要设置 setOnTimeChangeListener 监

听来设置时间，具体实现如代码清单 4-20 所示。



图 4-26 日期和时间界面



图 4-27 设置日期



图 4-28 设置时间

代码清单 4-20 第 4 章\Examples_04_11\src\com\yarin\android\Examples_04_11\Activity01.java

```
public class Activity01 extends Activity
{
    TextView m_TextView;
    //声明 DatePicker 对象
    DatePicker m_DatePicker;
    //声明 TimePicker 对象
    TimePicker m_TimePicker;
    Button m_dpButton;
    Button m_tpButton;
    //Java 中的 Calendar 类
    Calendar c;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        c=Calendar.getInstance();

        m_TextView= (TextView) findViewById(R.id.TextView01);
        m_dpButton = (Button) findViewById(R.id.button1);
        m_tpButton = (Button) findViewById(R.id.button2);

        //获取 DatePicker 对象
        m_DatePicker = (DatePicker) findViewById(R.id.DatePicker01);
        //将日历初始化为当前系统时间，并设置其事件监听
        m_DatePicker.init(c.get(Calendar.YEAR), c.get(Calendar.MONTH),
            c.get(Calendar.DAY_OF_MONTH), new DatePicker.OnDateChangedListener() {
                @Override
                public void onDateChanged(DatePicker view, int year, int monthOfYear,
                    int dayOfMonth)
                {
                    //当日期更改时，在这里处理
                    //c.set(year, monthOfYear, dayOfMonth);
                }
            });
    }
}
```

```

    }
    });

    //获取 TimePicker 对象
    m_TimePicker = (TimePicker) findViewById(R.id.TimePicker01);
    //设置为 24 小时制显示
    m_TimePicker.setIs24HourView(true);

    //监听时间改变
    m_TimePicker.setOnTimeChangeListener(new TimePicker.OnTimeChangeListener() {
        @Override
        public void onTimeChanged(TimePicker view, int hourOfDay, int minute)
        {
            //时间改变时处理
            //c.set(year, month, day, hourOfDay, minute, second);
        }
    });

    m_dpButton.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v)
        {
            new DatePickerDialog(Activity01.this,
                new DatePickerDialog.OnDateSetListener()
                {
                    public void onDateSet(DatePicker view, int year, int
                        monthOfYear, int dayOfMonth)
                    {
                        //设置日历
                    }
                }, c.get(Calendar.YEAR), c.get(Calendar.MONTH), c.get(Calendar.
                    DAY_OF_MONTH)).show();
        }
    });

    m_tpButton.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v)
        {
            new TimePickerDialog(Activity01.this,
                new TimePickerDialog.OnTimeSetListener()
                {
                    public void onTimeSet(TimePicker view, int hourOfDay, int minute)
                    {
                        //设置时间
                    }
                }, c.get(Calendar.HOUR_OF_DAY), c.get(Calendar.MINUTE), true).show();
        }
    });
}
}

```

日期和时间同样使用 XML 布局文件来完成, 如代码清单 4-21 所示。

代码清单 4-21 第4章\Examples_04_11\res\layout\main.xml

```

<?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/TextView01"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello"
/>
<DatePicker
android:id="@+id/DatePicker01"
android:layout_width="wrap_content"

android:layout_height="wrap_content"
>
    </DatePicker>
<TimePicker
android:id="@+id/TimePicker01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
>
    </TimePicker>
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"

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

android:text="设置时间"
>
    </Button>
</LinearLayout>

```

4.2.10 按钮 (Button)

按钮是用得最多的控件，在前面的示例中我们都已经见到了按钮。在 Android 平台中，按钮是通过 Button 来实现的，实现过程也非常简单。既然是按钮，被点击之后必定要触发事件，所以需要 对按钮设置 `setOnClickListener` 事件监听。本节示例（参见本书所附代码：第 4 章\Examples_04_12）运行效果如图 4-29 所示，当点击按钮之后所触发的按钮事件如图 4-30 所示。

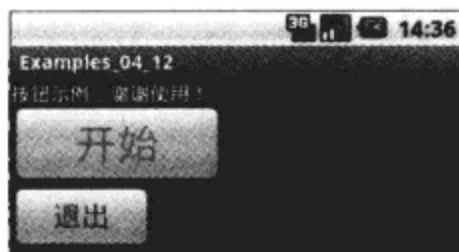


图 4-29 按钮效果



图 4-30 触发按钮事件

在效果中，我们简单地设置了按钮的大小、文本的颜色等属性，当然还可以设置按钮的其他属性，具体实现如代码清单 4-22 所示。

代码清单 4-22 第4章\Examples_04_12\Activity01.java

```
public class Activity01 extends Activity
{
    //声明按钮
    Button m_Button1,m_Button2;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //取得 Button 对象
        m_Button1=(Button)findViewById(R.id.Button1);
        m_Button2=(Button)findViewById(R.id.Button2);

        //设置按钮所显示的文字
        m_Button1.setText("开始");
        m_Button2.setText("退出");

        //设置按钮的宽度
        m_Button1.setWidth(150);
        m_Button2.setWidth(100);

        //设置文字的颜色
        m_Button1.setTextColor(Color.GREEN);
        m_Button2.setTextColor(Color.RED);

        //设置文字的大小尺寸
        m_Button1.setTextSize(30);
        m_Button2.setTextSize(20);

        //m_Button1.setBackgroundColor(Color.BLUE);

        //设置按钮的事件监听
        m_Button1.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v)
            {
                //处理按钮事件
                //产生一个 Toast
                //m_Button1.getText() 得到按钮显示的内容
                Toast toast = Toast.makeText(Activity01.this, "你点击了"+m_Button1.
                    getText()+"按钮!", Toast.LENGTH_LONG);
                //设置 Toast 显示的位置
                toast.setGravity(Gravity.TOP, 0, 150);
                //显示该 Toast
                toast.show();
            }
        });

        m_Button2.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v)
            {
```

```

        //处理按钮事件
        Activity01.this.finish();
    }
    });
}
}

```

4.2.11 菜单 (Menu)

Android 手机用一个按键“Menu”专门来显示菜单，所以，当应用程序设置了菜单，我们便可以通过该按键来操作应用程序的菜单选项。

要实现菜单功能，首先需要通过方法 `onCreateOptionsMenu` 来创建菜单，然后需要对其能够触发的事件进行监听，这样才能够在事件监听 `onOptionsItemSelected` 中根据不同的菜单选项来执行不同的任务。当然，可以通过 XML 布局来实现，也可以通过 `menu.add` 方法来实现。下面的示例分别采用了这两种不同的方法来实现（参见本书所附代码：第 4 章\Examples_04_13），效果如图 4-31 所示。点击键盘上的“Menu”

按钮之后，界面如图 4-32 所示；当选择了“关于”项时，跳转到另一个界面，如图 4-33 所示。

1. 通过 XML 布局来实现

首先在项目目录的“res”文件夹中建立“Menu”文件夹，然后在其中创建需要的菜单，如代码清单 4-23 所示。然后在 `onCreateOptionsMenu` 方法中通过 `onCreateOptionsMenu` 方法来装载这个菜单布局文件。在 `onOptionsItemSelected` 监听方法中通过 `getItemId` 方法获得当前选中的菜单的“ID”，如代码清单 4-24 所示。

代码清单 4-23 第 4 章\Examples_04_13\res\menu\menu.xml

```

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/about"
        android:title="关于" />
    <item android:id="@+id/exit"
        android:title="退出" />
</menu>

```

下面我们就需要在代码中来装载这个定义了菜单的 XML 布局文件，如代码清单 4-24 所示。

代码清单 4-24 第 4 章\Examples_04_13\src\com\yarin\android\Examples_04_13\Activity01.java

```

public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}

```

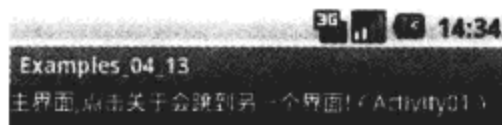


图 4-31 菜单示例



图 4-32 点击“Menu”按钮

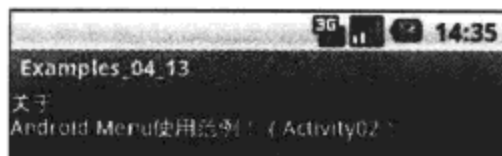


图 4-33 跳转到“关于”界面

```

}

/*创建 menu*/
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    //设置 menu 界面为 res/menu/menu.xml
    inflater.inflate(R.menu.menu, menu);
    return true;
}

/*处理菜单事件*/
public boolean onOptionsItemSelected(MenuItem item)
{
    //得到当前选中的 MenuItem 的 ID
    int item_id = item.getItemId();

    switch (item_id)
    {
        case R.id.about:
            /* 新建一个 Intent 对象 */
            Intent intent = new Intent();
            /* 指定 intent 要启动的类 */
            intent.setClass(Activity01.this, Activity02.class);
            /* 启动一个新的 Activity */
            startActivity(intent);
            /* 关闭当前的 Activity */
            Activity01.this.finish();
            break;
        case R.id.exit:
            Activity01.this.finish();
            break;
    }
    return true;
}
}

```

2. 通过 Menu.add 方法实现

即在 onCreateOptionsMenu 方法中通过 “menu.add(0, 0, 0, R.string.ok);” 来实现创建一个菜单选项, 如代码清单 4-25 所示。

代码清单 4-25 第4章\Examples_04_13\src\com\yarin\android\Examples_04_13\Activity02.java

```

public class Activity02 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        /* 设置显示 main2.xml 布局 */
        setContentView(R.layout.main2);
    }

    /*创建 Menu*/
    public boolean onCreateOptionsMenu(Menu menu)
    {
        //为 Menu 添加内容
        menu.add(0, 0, 0, R.string.ok);
        menu.add(0, 1, 1, R.string.back);
    }
}

```

```

        return true;
    }

    /*处理 Menu 的事件*/
    public boolean onOptionsItemSelected(MenuItem item)
    {
        //得到当前选中的 MenuItem 的 ID
        int item_id = item.getItemId();

        switch (item_id)
        {
            case 0:
            case 1:
                /* 新建一个 Intent 对象 */
                Intent intent = new Intent();
                /* 指定 intent 要启动的类 */
                intent.setClass(Activity02.this, Activity01.class);
                /* 启动一个新的 Activity */
                startActivity(intent);
                /* 关闭当前的 Activity */
                Activity02.this.finish();
                break;
        }
        return true;
    }
}

```

由于在项目 Examples_04_13 中使用了两个 Activity，因此需要在 AndroidManifest.xml 中声明。

4.2.12 对话框 (Dialog)

Android 中实现对话框可以使用 AlertDialog.Builder 类，还可以自定义对话框。如果对话框设置了按钮，那么需要对其设置事件监听 OnClickListener。下面我们将分析一个示例，该示例分别使用了 AlertDialog.Builder 和自定义对话框等方法（参见本书所附代码：第 4 章\Examples_04_14），效果如图 4-34 所示。点击“退出”按钮时，退出应用程序，点击“确定”按钮时，转移到我们自定义的对话框来实现登录界面，如图 4-35 所示。当输入账号、密码等信息后，程序将转移到一个登录进度对话框界面，如图 4-36 所示。



图 4-34 带两个按钮的对话框



图 4-35 自定义登录界面对话框



图 4-36 带进度条的对话框

1. 自定义对话框

首先观察图 4-35 这个对话框，它包括两个 TextView 和两个 EditText，所以需要在布局文件中定义这个对话框界面，具体实现如代码清单 4-26 所示。然后，通过 inflate 方法来创建对话框。

代码清单 4-26 第4章\Examples_04_14\res\layout\dialog.xml

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

    <TextView
        android:id="@+id/username"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"

        android:text="账号"

        android:gravity="left"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/username"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:scrollHorizontally="true"
        android:autoText="false"
        android:capitalize="none"
        android:gravity="fill_horizontal"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/password"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"

        android:text="密码"
        android:gravity="left"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/password"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip"
        android:scrollHorizontally="true"
        android:autoText="false"
        android:capitalize="none"
        android:gravity="fill_horizontal"
        android:password="true"
```

```

        android:textAppearance="?android:attr/textAppearanceMedium" />
    </LinearLayout>

```

2. 使用 AlertDialog 来创建对话框

使用 AlertDialog 创建对话框需要了解以下几个方法，完整的实现过程如代码清单 4-27 所示。

- ☐ setTitle(): 给对话框设置 title。
- ☐ setIcon(): 给对话框设置图标。
- ☐ setMessage(): 设置对话框的提示信息。
- ☐ setItems(): 设置对话框要显示的一个 list，一般用于显示几个命令时。
- ☐ setSingleChoiceItems(): 设置对话框显示一个单选的 List。
- ☐ setMultiChoiceItems(): 用来设置对话框显示一系列的复选框。
- ☐ setPositiveButton(): 给对话框添加 “Yes” 按钮。
- ☐ setNegativeButton(): 给对话框添加 “No” 按钮。

代码清单 4-27 第 4 章\Examples_04_14\src\com\yarin\android\Examples_04_14\Activity01.java

```

public class Activity01 extends Activity
{
    ProgressDialog m_Dialog;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Dialog dialog = new AlertDialog.Builder(Activity01.this)
            .setTitle("登录提示")//设置标题
            .setMessage("这里需要登录!")//设置内容
            .setPositiveButton("确定",//设置确定按钮
                new DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        //点击“确定”转向登录框

                        LayoutInflater factory = LayoutInflater.from(Activity01.this);
                        //得到自定义对话框
                        final View DialogView = factory.inflate(R.layout.dialog, null);
                        //创建对话框
                        AlertDialog dlg = new AlertDialog.Builder(Activity01.this)
                            .setTitle("登录框")
                            .setView(DialogView)//设置自定义对话框的样式
                            .setPositiveButton("确定", //设置“确定”按钮
                                new DialogInterface.OnClickListener() //设置事件监听
                                {
                                    public void onClick(DialogInterface dialog, int whichButton)
                                    {
                                        //输入完成后，点击“确定”开始登录
                                        m_Dialog = ProgressDialog.show
                                            (
                                                Activity01.this,
                                                "请等待...",
                                                "正在为你登录...",
                                                true
                                            );
                                    }
                                }
                            );
                    }
                }
            );
    }
}

```



```

        new Thread()
        {
            public void run()
            {
                try
                {
                    sleep(3000);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                finally
                {
                    //登录结束, 取消 m_Dialog 对话框
                    m_Dialog.dismiss();
                }
            }
        }.start();
    })
    .setNegativeButton("取消", //设置“取消”按钮
    new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            //点击“取消”按钮之后退出程序
            Activity01.this.finish();
        }
    })
    .create(); //创建
    dlg.show(); //显示
}
}).setNeutralButton("退出",
new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int whichButton)
    {
        //点击“退出”按钮之后退出程序
        Activity01.this.finish();
    }
})
).create(); //创建按钮

// 显示对话框
dialog.show();
}
}

```

4.2.13 图片视图 (ImageView)

我们要将一张图片显示在屏幕上, 首先需要创建一个显示图片的对象, 在 Android 中, 这个对象是 ImageView 对象, 然后通过 setImageResource 方法来设置要显示的图片资源索引。当然, 还可以对图片执行一些其他的操作, 比如设置它的 Alpha 值等。这里将直接用一个示例来分析 ImageView 的使用, 该示例中通过一个线程来不断更新 ImageView 的 Alpha 值, 实现效果如图 4-37

所示。当程序运行一段时间之后, Alpha 值逐渐变小, 效果如图 4-38 所示。具体实现参见本书所附代码: 第 4 章\Examples_04_15。



图 4-37 ImageView 视图



图 4-38 Alpha 值为 94

首先 XML 布局通过 ImageView 来显示一张图片, 通过 TextView 来显示文字, 布局方式如代码清单 4-28 所示。

代码清单 4-28 第 4 章\Examples_04_15\res\layout\main.xml

```
<?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"
    >
    <ImageView
        android:id="@+id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        >
    </ImageView>
    <TextView
        android:id="@+id/TextView01"
        android:layout_below="@id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        >
    </TextView>
</LinearLayout>
```

视图更新和线程的处理, 如代码清单 4-29 所示。

代码清单 4-29 第 4 章\Examples_04_15\res\layout\Activity01.java

```
public class Activity01 extends Activity
{
    //声明 ImageView 对象
    ImageView imageview;
    TextView textview;
    //ImageView 的 alpha 值,
    int image_alpha = 255;

    Handler mHandler = new Handler();
    //控件线程
    boolean isrun = false;

    public void onCreate(Bundle savedInstanceState)
```

```

{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    isrung = true;

    //获得 ImageView 的对象
    imageview = (ImageView) this.findViewById(R.id.ImageView01);
    textview = (TextView) this.findViewById(R.id.TextView01);

    //设置 imageview 的图片资源, 同样可以在 xml 布局中像下面这样写
    //android:src="@drawable/logo"
    imageview.setImageResource(R.drawable.logo);

    //设置 imageview 的 Alpha 值
    imageview.setAlpha(image_alpha);

    //开启一个线程来让 Alpha 值递减
    new Thread(new Runnable() {
        public void run()
        {
            while (isrung)
            {
                try
                {
                    Thread.sleep(200);
                    //更新 Alpha 值
                    updateAlpha();
                }
                catch (InterruptedException e)
                {
                    e.printStackTrace();
                }
            }
        }
    }).start();

    //接收消息之后更新 imageview 视图
    mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg)
        {
            super.handleMessage(msg);
            imageview.setAlpha(image_alpha);
            textview.setText("现在 alpha 值是: "+Integer.toString(image_alpha));
            //更新
            imageview.invalidate();
        }
    };
}

public void updateAlpha()
{
    if (image_alpha - 7 >= 0)
    {
        image_alpha -= 7;
    }
}

```

```

    }
    else
    {
        image_alpha = 0;
        isrung = false;
    }
    //发送需要更新 imageview 视图的消息
    mHandler.sendMessage(mHandler.obtainMessage());
}
}

```

4.2.14 带图标按钮 (ImageButton)

除了可以使用 Android 系统自带的 Button (按钮) 外, 在 Android 平台中, 我们还可以制作带图标的按钮, 这就需要使用本节将介绍的 ImageButton 组件。

要制作带图标的按钮, 首先要在布局文件中定义 ImageButton, 然后通过 setImageDrawable 方法来设置按钮要显示的图标。同样需要对按钮设置事件监听 setOnClickListener, 以此来捕捉事件并处理。下面我来看一个示例 (参见本书所附代码: 第 4 章\Examples_04_16), 运行效果如图 4-39 所示, 其中有三个按钮图标是我们自定义的, 另外一个系统是图标。当点击系统图标按钮时, 捕捉其点击事件, 处理后的效果如图 4-40 所示。当我们点击第三个自定义的图标按钮 (ImageButton3) 后, 效果如图 4-41 所示。最后, 将 ImageButton3 图标改变为一个系统图标, 效果如图 4-42 所示。

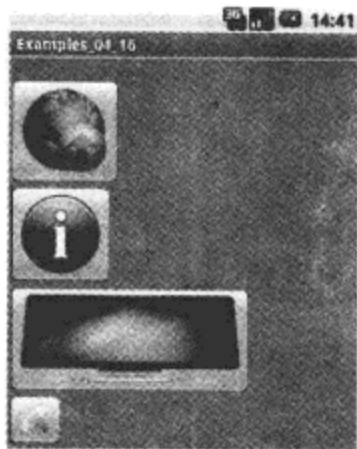


图 4-39 ImageButton 示例

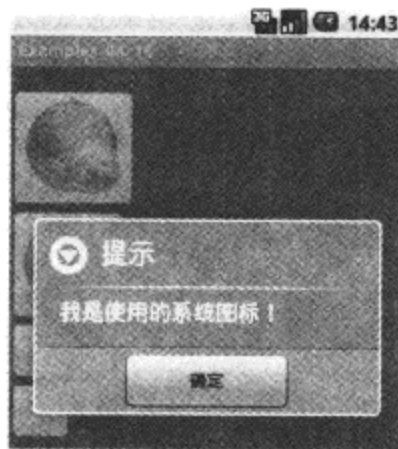


图 4-40 使用系统图标



图 4-41 ImageButton 事件处理



图 4-42 ImageButton 更改系统图标

该例子的布局文件很简单, 直接在一个线性布局中放置 4 个 ImageButton 组件即可, 如代码清单 4-30 所示。

代码清单 4-30 第4章\Examples_04_16\res\layout\main.xml

```

<?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/TextView01"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
<ImageButton
    android:id="@+id/ImageButton01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button1"
    >
...
</LinearLayout>

```

ImageButton 组件事件监听如代码清单 4-31 所示。

代码清单 4-31 第4章\Examples_04_16\src\com\yarin\android\Examples_04_16\Activity01.java

```

public class Activity01 extends Activity
{
    TextView m_TextView;
    //声明 4 个 ImageButton 对象
    ImageButton m_ImageButton1;
    ImageButton m_ImageButton2;
    ImageButton m_ImageButton3;
    ImageButton m_ImageButton4;

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

        m_TextView = (TextView) findViewById(R.id.TextView01);
        //分别取得 4 个 ImageButton 对象
        m_ImageButton1 = (ImageButton) findViewById(R.id.ImageButton01);
        m_ImageButton2 = (ImageButton) findViewById(R.id.ImageButton02);
        m_ImageButton3 = (ImageButton) findViewById(R.id.ImageButton03);
        m_ImageButton4 = (ImageButton) findViewById(R.id.ImageButton04);

        //分别设置所使用的图标
        //m_ImageButton1 是在 xml 布局中设置的, 这里就暂时不设置了
        m_ImageButton2.setImageDrawable(getResources().getDrawable(R.drawable.button2));
        m_ImageButton3.setImageDrawable(getResources().getDrawable(R.drawable.button3));
        m_ImageButton4.setImageDrawable(getResources().getDrawable(android.R.drawable.sym_call_incoming));

        //以下分别为每个按钮设置事件监听 setOnClickListener
        m_ImageButton1.setOnClickListener(new Button.OnClickListener()

```

```

{
    public void onClick(View v)
    {
        //对话框
        Dialog dialog = new AlertDialog.Builder(Activity01.this)
            .setTitle("提示")
            .setMessage("我是 ImageButton1")
            .setPositiveButton("确定",
                new DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {

                    }
                })
            .create(); //创建按钮

        dialog.show();
    }
});
m_ImageButton2.setOnClickListener(new Button.OnClickListener()
{
    public void onClick(View v)
    {
        Dialog dialog = new AlertDialog.Builder(Activity01.this)
            .setTitle("提示")
            .setMessage("我是 ImageButton2, 我要使用 ImageButton3 的图标")
            .setPositiveButton("确定",
                new DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        m_ImageButton2.setImageDrawable(getResources().getDrawable(
                            R.drawable.button3));
                    }
                })
            .create(); //创建按钮

        dialog.show();
    }
});
m_ImageButton3.setOnClickListener(new Button.OnClickListener()
{
    public void onClick(View v)
    {
        Dialog dialog = new AlertDialog.Builder(Activity01.this)
            .setTitle("提示")
            .setMessage("我是 ImageButton3, 我要使用系统打电话图标")
            .setPositiveButton("确定",
                new DialogInterface.OnClickListener()
                {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        m_ImageButton3.setImageDrawable(getResources().getDrawable(
                            android.R.drawable.sym_action_call));
                    }
                })
            .create(); //创建按钮

        dialog.show();
    }
});

```



```

    }
    });
    m_ImageButton4.setOnClickListener(new Button.OnClickListener()
    {
        public void onClick(View v)
        {
            Dialog dialog = new AlertDialog.Builder(Activity01.this)
                .setTitle("提示")
                .setMessage("我是使用的系统图标! ")
                .setPositiveButton("确定",
                    new DialogInterface.OnClickListener()
                    {
                        public void onClick(DialogInterface dialog, int whichButton)
                        {
                            //创建按钮
                        }
                    })
                .create();

            dialog.show();
        }
    });
}
}

```

4.2.15 拖动效果 (Gallery)

这是一个非常炫的效果，可以用手指直接拖动图片移动，iPhone 曾经凭借这个效果吸引了无数的苹果粉丝，在 Android 平台上也可实现这一效果。要实现这一效果，需要一个容器来存放 Gallery 显示的图片，这里使用一个继承自 `BaseAdapter` 类的派生类来装这些图片。我们需要监听其事件 `setOnItemClickListener`，从而确定用户当前选中的是哪一张图片。首先，需要将所有要显示的图片的索引存放在一个 `int` 型数组中，然后通过 `setImageResource` 方法来设置 `ImageView` 要显示的图片资源，最后将每张图片的 `ImageView` 显示在屏幕上。为了便于大家理解，这里仍然以一个示例（参见本书所附代码：第 4 章\Examples_04_17）说明，效果如图 4-43 所示。当鼠标点击某个图片时，捕捉并处理该事件，如图 4-44 所示。当然，事先需要准备一些用于显示的图片资源。



图 4-43 Gallery 效果



图 4-44 Gallery 事件处理

首先来看看布局文件如何实现 Gallery，如代码清单 4-32 所示。

代码清单 4-32 第 4 章\Examples_04_17\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Gallery
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/Gallery01"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

其次是存放这些图片的容器 ImageAdapter 的实现，如代码清单 4-33 所示。

代码清单 4-33 第 4 章\Examples_04_17\src\com\yarin\android\Examples_04_17\ImageAdapter.Java

```
public class ImageAdapter extends BaseAdapter
{
    // 定义 Context
    private Context      mContext;
    // 定义整型数组，即图片源
    private Integer[]    mImageIds =
    {
        R.drawable.img1,
        R.drawable.img2,
        R.drawable.img3,
        R.drawable.img4,
        R.drawable.img5,
        R.drawable.img6,
        R.drawable.img7,
        R.drawable.img8,
    };

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

    // 获取图片的个数
    public int getCount()
    {
        return mImageIds.length;
    }

    // 获取图片在库中的位置
    public Object getItem(int position)
    {
        return position;
    }

    // 获取图片在库中的位置
    public long getItemId(int position)
    {
        return position;
    }

    public View getView(int position, View convertView, ViewGroup parent)
    {
        ImageView imageview = new ImageView(mContext);
```

```

// 给 ImageView 设置资源
imageView.setImageResource(mImageIds[position]);
// 设置布局图片以 120*120 显示
imageView.setLayoutParams(new Gallery.LayoutParams(120, 120));
// 设置显示比例类型
imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
return imageView;
}
}

```

然后我们通过 `setAdapter` 方法把资源文件添加到 `Gallery` 中来显示，并且设置其事件处理，具体过程如代码清单 4-34 所示。

代码清单 4-34 第 4 章\Examples_04_17\src\com\yarin\android\Examples_04_17\Activity01.java

```

public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // 获得 Gallery 对象
        Gallery g = (Gallery) findViewById(R.id.Gallery01);

        // 添加 ImageAdapter 给 Gallery 对象
        g.setAdapter(new ImageAdapter(this));

        // 设置 Gallery 的背景
        g.setBackgroundResource(R.drawable.bg0);

        // 设置 Gallery 的事件监听
        g.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View v, int position, long id)
            {
                Toast.makeText(Activity01.this, "你选择了" + (position+1) + " 号图片",
                    Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

4.2.16 切换图片 (ImageSwitcher)

我们在 Windows 平台上要查看多张图片，最简单的方法就是通过“Windows 图片和传真查看器”在“下一张”和“上一张”之间切换，Android 平台上可以通过 `ImageSwitcher` 类来实现这一效果。`ImageSwitcher` 类必须设置一个 `ViewFactory`，主要用来将显示的图片 and 父窗口区分开来，因此需要实现 `ViewSwitcher.ViewFactory` 接口，通过 `makeView()` 方法来显示图片，这里会返回一个 `ImageView` 对象，而方法 `setImageResource` 用来显示指定的图片资源。我们先来看一个示例（参见本书所附代码：第 4 章\Examples_04_18），其运行效果如图 4-45 所示。当点击“上一张”或“下一张”按钮时，切换浏览另一张图片，如图 4-46 所示。具体实现过程如代码清单 4-35 所示。

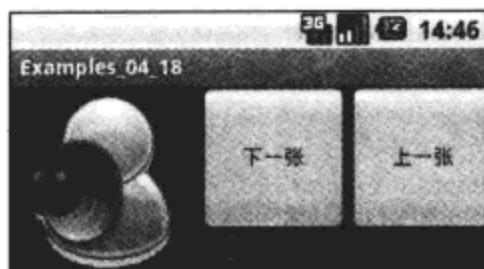


图 4-45 ImageSwitcher 视图

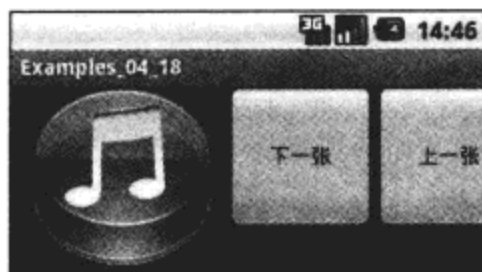


图 4-46 点击按钮可以切换图片

代码清单 4-35 第 4 章\Examples_04_18\src\com\yarin\android\Examples_04_18\Activity01.java

```
public class Activity01 extends Activity implements OnClickListener, ViewFactory
{
    /* 所有要显示的图片资源索引 */
    private static final Integer[] imagelist =
    {
        R.drawable.img1,
        R.drawable.img2,
        R.drawable.img3,
        R.drawable.img4,
        R.drawable.img5,
        R.drawable.img6,
        R.drawable.img7,
        R.drawable.img8,
    };

    //创建 ImageSwitcher 对象
    private ImageSwitcher m_Switcher;
    //索引
    private static int index = 0;

    //“下一页”按钮 ID
    private static final int BUTTON_DWON_ID = 0x123456;
    //“上一页”按钮 ID
    private static final int BUTTON_UP_ID = 0x123457;
    //ImageSwitcher 对象的 ID
    private static final int SWITCHER_ID = 0x123458;

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

        //创建一个线性布局 LinearLayout
        LinearLayout main_view = new LinearLayout(this);
        //创建 ImageSwitcher 对象
        m_Switcher = new ImageSwitcher(this);
        //在线性布局中添加 ImageSwitcher 视图
        main_view.addView(m_Switcher);
        //设置 ImageSwitcher 对象的 ID
        m_Switcher.setId(SWITCHER_ID);
        //设置 ImageSwitcher 对象的数据源
        m_Switcher.setFactory(this);
        m_Switcher.setImageResource(imagelist[index]);
    }
}
```

```

//设置显示上面创建的线性布局
setContentView(main_view);

//创建“下一张”按钮
Button next = new Button(this);
next.setId(BUTTON_DWON_ID);
next.setText("下一张");
next.setOnClickListener(this);
LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(100, 100);
main_view.addView(next, param);

//创建“上一张”按钮
Button pre = new Button(this);
pre.setId(BUTTON_UP_ID);
pre.setText("上一张");
pre.setOnClickListener(this);
main_view.addView(pre, param);
}

//事件监听、处理
public void onClick(View v)
{
    switch (v.getId())
    {
        //下一页
        case BUTTON_DWON_ID:
            index++;
            if (index >= imagelist.length)
            {
                index = 0;
            }
            //ImageSwitcher 对象资源索引
            m_Switcher.setImageResource(imagelist[index]);
            break;
        //上一页
        case BUTTON_UP_ID:
            index--;
            if (index < 0)
            {
                index = imagelist.length - 1;
            }
            //ImageSwitcher 对象资源索引
            m_Switcher.setImageResource(imagelist[index]);
            break;
        default:
            break;
    }
}

public View makeView()
{
    //将所有图片通过 ImageView 来显示
    return new ImageView(this);
}
}

```

4.2.17 网格视图 (GridView)

GridView 的视图排列方式与矩阵类似, 当屏幕上有很多元素 (文字、图片或其他元素) 需要显示时, 可以使用 GridView。既然有多个元素需要显示, 就需要使用 BaseAdapter 来存储这些元素。用户可能会选择其中一个元素进行操作, 这就需要设置事件监听 setOnItemClickListener 来捕捉和处理事件, 下面的例子 (参见本书所附代码: 第 4 章\Examples_04_19) 显示了 9 个图片, 并且捕捉了鼠标点击的事件, 运行效果如图 4-47 所示。当用户点击其中某一个元素时, 处理事件如图 4-48 所示。



图 4-47 GridView 视图

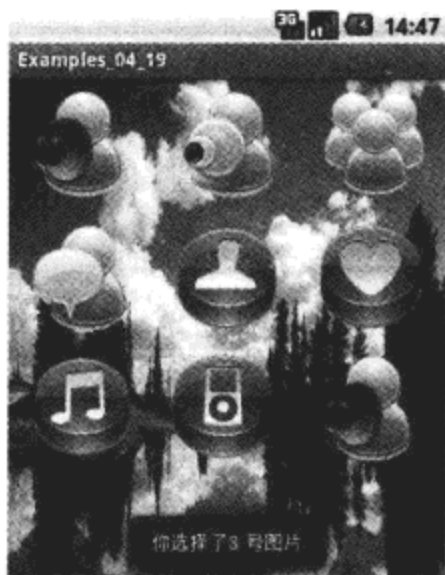


图 4-48 GridView 事件处理

首先我们来看看如何通过 XML 布局来实现这个效果, 如代码清单 4-36 所示。

代码清单 4-36 第 4 章\Examples_04_19\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:columnWidth="90dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

这里我们定义了一个 ImageAdapter 继承自 BaseAdapter, 用来存储要显示的图片, 和上一节中的 ImageAdapter.java 基本一样。

要知道用户所选择的是哪一项, 就需要设置 GridView 的事件监听, 如代码清单 4-37 所示。

代码清单 4-37 第 4 章\Examples_04_19\src\com\yarin\android\Examples_04_19\Activity01.java

```
public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```



```

setContentView(R.layout.main);

//取得 GridView 对象
GridView gridView = (GridView) findViewById(R.id.gridview);
//添加元素给 gridView
gridView.setAdapter(new ImageAdapter(this));

// 设置 Gallery 的背景
gridView.setBackgroundResource(R.drawable.bg0);

//事件监听
gridView.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v, int position, long id)
    {
        Toast.makeText(Activity01.this, "你选择了" + (position + 1) + " 号图片",
            Toast.LENGTH_SHORT).show();
    }
});
}
}

```

4.2.18 卷轴视图 (ScrollView)

卷轴视图是指当拥有很多内容，一屏显示不完时，需要通过滚动来显示的视图。比如在做阅读器的时候，文章很长，一页显示不完，那么就需要使用卷轴视图来滚动显示下一页。本节示例（参见本书所附代码：第4章\Examples_04_20）中以一个 TextView 和 Button 来实现自动滚动，根据内容的多少自动改变默认焦点。运行效果如图 4-49 所示，当我们点击“Button0”时自动产生多个类似项，如果一屏显示不完，则通过 ScrollView 来显示，如图 4-50 所示。

要实现这一效果，首先需要在布局文件中声明 ScrollView，具体实现如代码清单 4-38 所示。

代码清单 4-38 第4章\Examples_04_20\res\layout\main.xml

```

<ScrollView xmlns:android="http://schemas.android.
com/apk/res/android"
    android:id="@+id/ScrollView01"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:scrollbars="none">
    <LinearLayout
        android:id="@+id/layout"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="TextView0"/>
        <Button
            android:id="@+id/Button01"
            android:layout_width="fill_parent"

```

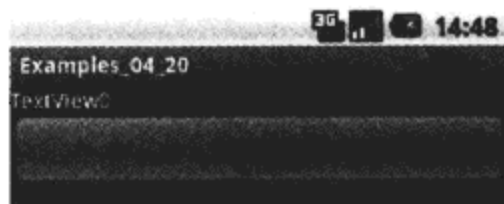


图 4-49 只有一个选项

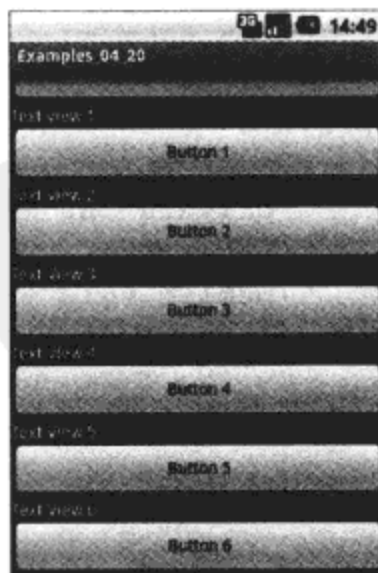


图 4-50 一屏显示不完通过 ScrollView 进行滚动

```

        android:layout_height="wrap_content"
        android:text="Button0"/>
    </LinearLayout>
</ScrollView>

```

代码清单 4-38 中定义了一个 `ScrollView`，其中包括一个线性布局，而这个线性布局则是由一个 `TextView` 和一个 `Button` 组成，因此我们在点击按钮时每次也增加这么一个线性布局。而逻辑部分则如代码清单 4-39 所示。

代码清单 4-39 第 4 章\Examples_04_20\src\com\yarin\android\Examples_04_20\Activity01.java

```

public class Activity01 extends Activity
{
    private LinearLayout mLayout;
    private ScrollView mScrollView;
    private final Handler mHandler = new Handler();

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

        //创建一个线性布局
        mLayout = (LinearLayout) findViewById(R.id.layout);
        //创建一个 ScrollView 对象
        mScrollView = (ScrollView) findViewById(R.id.ScrollView01);

        Button button = (Button) findViewById(R.id.Button01);

        button.setOnClickListener(mClickListener);
        //改变默认焦点切换
        button.setOnKeyListener(mAddButtonKeyListener);
    }

    //Button 事件监听
    //当点击按钮时，增加一个 TextView 和 Button
    private Button.OnClickListener mClickListener = new Button.OnClickListener()
    {
        private int mIndex = 1;
        @Override
        public void onClick(View arg0)
        {
            // TODO Auto-generated method stub
            TextView textView = new TextView(Activity01.this);
            textView.setText("Text View " + mIndex);
            LinearLayout.LayoutParams p = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.FILL_PARENT,
                LinearLayout.LayoutParams.WRAP_CONTENT
            );
            //增加一个 TextView 到线性布局中
            mLayout.addView(textView, p);

            Button buttonView = new Button(Activity01.this);
            buttonView.setText("Button " + mIndex++);
        }
    }
}

```

```

//增加一个 Button 到线性布局中
mLayout.addView(buttonView, p);
//改变默认焦点切换
buttonView.setOnKeyListener(mNewButtonKeyListener);
//投递一个消息进行滚动
mHandler.post(mScrollToBottom);
}
};

```

```

private Runnable mScrollToBottom = new Runnable()
{
    public void run()
    {
        // TODO Auto-generated method stub

        int off = mLayout.getMeasuredHeight() - mScrollView.getHeight();
        if (off > 0)
        {
            mScrollView.scrollTo(0, off);
        }
    }
};

```

//事件监听

```

private View.OnKeyListener mNewButtonKeyListener = new View.OnKeyListener()
{
    public boolean onKey(View v, int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN &&
            event.getAction() == KeyEvent.ACTION_DOWN &&
            v == mLayout.getChildAt(mLayout.getChildCount() - 1))
        {
            findViewById(R.id.Button01).requestFocus();
            return true;
        }
        return false;
    }
};

```

//事件监听

```

private View.OnKeyListener mAddButtonKeyListener = new Button.OnKeyListener()
{
    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event)
    {
        // TODO Auto-generated method stub
        View viewToFoucus = null;
        if (event.getAction() == KeyEvent.ACTION_DOWN)
        {
            int iCount = mLayout.getChildCount();
            switch (keyCode)

```

```

    {
        case KeyEvent.KEYCODE_DPAD_UP:
            if ( iCount > 0)
            {
                viewToFoucus = mLayout.getChildAt(iCount - 1);
            }
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            if (iCount < mLayout.getWeightSum())
            {
                viewToFoucus = mLayout.getChildAt(iCount + 1);
            }
            break;
        default:
            break;
    }
    if (viewToFoucus != null)
    {
        viewToFoucus.requestFocus();
        return true;
    }
    else
    {
        return false;
    }
}
};
}

```

代码清单 4-39 使用一个 Handler 来更新布局的高度，以此来调节默认焦点来实现自动滚动，以及对事件进行监听，当滚动到最后一项时，自动跳到第一项。

4.2.19 进度条 (ProgressBar)

当一个应用在后台执行时，前台界面就不会有什么信息，这时用户根本不知道程序是否在执行、执行进度如何、应用程序是否遇到错误终止等，这时需要使用进度条来提示用户后台程序执行的进度。Android 系统提供了两大类进度条样式，长形进度条 (progressBarStyleHorizontal) 和圆形进度条 (progressBarStyleLarge)。进度条用处很多，比如，应用程序装载资源和网络连接时，可以提示用户稍等，这一类进度条只是代表应用程序中某一部分程序的执行情况，而整个应用程序执行情况呢，则可以通过应用程序标题栏来显示一个进度条，这就需要先对窗口的显示风格进行设置 “requestWindowFeature(Window.FEATURE_PROGRESS);”，下面我们通过一个示例 (参见本书所附代码：第 4 章\Examples_04_21) 来分析圆形、长形、标题栏进度条的使用。效果如图 4-51 所示。

要实现长形和圆形进度条，我们先在布局文件中声明，如代码清单 4-40 所示。



图 4-51 各种进度条

代码清单 4-40 第4章\Examples_04_21\res\layout\main.xml

```

<?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"
    />
<ProgressBar
    android:id="@+id/ProgressBar01"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:visibility="gone"
    />
<ProgressBar
    android:id="@+id/ProgressBar02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleLarge"
    android:max="100"
    android:progress="50"
    android:secondaryProgress="70"
    android:visibility="gone"
    />
<Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="开始" />
</LinearLayout>

```

代码清单 4-40 中, ProgressBar01 为长形进度条, ProgressBar02 则为圆形进度条, 标题栏进度条没有在这里声明。“style=”表示设置进度条的风格。下面通过 setMax 方法设置进度条的最大值, setProgress 方法设置进度条的当前值, setVisibility 方法设置进度条的可见性。既然是进度条, 就有一个执行过程, 这里通过线程来模拟这个进度条。具体实现代码如代码清单 4-41 所示。

代码清单 4-41 第4章\Examples_04_21\src\com\yarin\android\Examples_04_21\Activity01.java

```

public class Activity01 extends Activity
{
    //声明 ProgressBar 对象
    private ProgressBar m_ProgressBar;
    private ProgressBar m_ProgressBar2;
    private Button mButton01;
    protected static final int GUI_STOP_NOTIFIER = 0x108;
    protected static final int GUI_THREADING_NOTIFIER = 0x109;
    public int intCounter=0;
    public void onCreate(Bundle savedInstanceState)
    {

```

```

super.onCreate(savedInstanceState);
//设置窗口模式, , 因为需要显示进度条在标题栏
requestWindowFeature(Window.FEATURE_PROGRESS);
setProgressBarVisibility(true);
setContentView(R.layout.main);

//取得 ProgressBar
m_ProgressBar = (ProgressBar) findViewById(R.id.ProgressBar01);
m_ProgressBar2= (ProgressBar) findViewById(R.id.ProgressBar02);
mButton01 = (Button)findViewById(R.id.Button01);

m_ProgressBar.setIndeterminate(false);
m_ProgressBar2.setIndeterminate(false);

//当按钮按下时开始执行
mButton01.setOnClickListener(new Button.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // TODO Auto-generated method stub

        //设置 ProgressBar 为可见状态
        m_ProgressBar.setVisibility(View.VISIBLE);
        m_ProgressBar2.setVisibility(View.VISIBLE);
        //设置 ProgressBar 的最大值
        m_ProgressBar.setMax(100);
        //设置 ProgressBar 当前值
        m_ProgressBar.setProgress(0);
        m_ProgressBar2.setProgress(0);

        //通过线程来改变 ProgressBar 的值
        new Thread(new Runnable() {
            public void run()
            {
                for (int i = 0; i < 10; i++)
                {
                    try
                    {
                        intCounter = (i + 1) * 20;
                        Thread.sleep(1000);

                        if (i == 4)
                        {
                            Message m = new Message();

                            m.what = Activity01.GUI_STOP_NOTIFIER;
                            Activity01.this.myMessageHandler.sendMessage(m);
                            break;
                        }
                    }
                    else
                    {
                        Message m = new Message();
                        m.what = Activity01.GUI_THREADING_NOTIFIER;
                        Activity01.this.myMessageHandler.sendMessage(m);
                    }
                }
            }
        });
    }
}

```



```

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    }).start();
});
}

Handler myMessageHandler = new Handler()
{
    public void handleMessage(Message msg)
    {
        switch (msg.what)
        {
            //ProgressBar 已经是最大值
            case Activity01.GUI_STOP_NOTIFIER:
                m_ProgressBar.setVisibility(View.GONE);
                m_ProgressBar2.setVisibility(View.GONE);
                Thread.currentThread().interrupt();
                break;
            case Activity01.GUI_THREADING_NOTIFIER:
                if (!Thread.currentThread().isInterrupted())
                {
                    // 改变 ProgressBar 的当前值
                    m_ProgressBar.setProgress(intCounter);
                    m_ProgressBar2.setProgress(intCounter);

                    // 设置标题栏中前景的一个进度条进度值
                    setProgress(intCounter*100);
                    // 设置标题栏中后面的一个进度条进度值
                    setSecondaryProgress(intCounter*100);
                }
                break;
        }
        super.handleMessage(msg);
    }
};
}

```

4.2.20 拖动条 (SeekBar)

拖动条类似进度条，不同的是用户可以控制，比如，应用程序中用户可以对音效进行控制，这就可以使用拖动条来实现。由于拖动条可以被用户控制，所以需要对其进行事件监听，这就需要实现 `SeekBar.OnSeekBarChangeListener` 接口。在 `SeekBar` 中共需要监听 3 个事件，分别是：数值的改变 (`onProgressChanged`)、开始拖动 (`onStartTrackingTouch`)、停止拖动 (`onStopTrackingTouch`)。在 `onProgressChanged` 中我们可以得到当前数值的大小。下面我们来看看示例的运行效果，如图 4-52 所示用户正在拖动，如图 4-53 所示用户停止调节。具体实现参见本书所附代码：第 4 章 Examples_04_22。

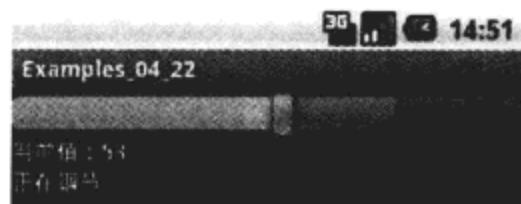


图 4-52 正在调节

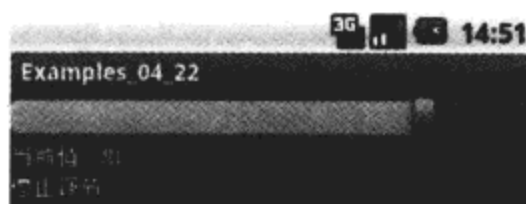


图 4-53 停止调节

首先需要在布局文件中声明 SeekBar，如代码清单 4-42 所示。

代码清单 4-42 第 4 章\Examples_04_22\res\layout\main.xml

```
<?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">

    <SeekBar android:id="@+id/seek"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="50"
        android:secondaryProgress="75" />

    <TextView android:id="@+id/progress"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <TextView android:id="@+id/tracking"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

代码清单 4-42 中通过 “android:max=“100”” 设置了最大值、“android:progress=“50”” 设置了当前值等属性。因此在使用时就只需要监听其事件并处理即可，如代码清单 4-43 所示。

代码清单 4-43 第 4 章\Examples_04_22\src\com\yarin\android\Examples_04_22\Activity01.java

```
public class Activity01 extends Activity
    implements SeekBar.OnSeekBarChangeListener
{
    //声明 SeekBar 对象
    SeekBar mSeekBar;
    TextView mProgressText;
    TextView mTrackingText;

    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        //取得 SeekBar 对象
        mSeekBar = (SeekBar) findViewById(R.id.seek);
        mSeekBar.setOnSeekBarChangeListener(this);
        mProgressText = (TextView) findViewById(R.id.progress);
        mTrackingText = (TextView) findViewById(R.id.tracking);
    }
}
```

```

//在拖动中——即值在改变
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromTouch)
{
    mProgressText.setText("当前值: "+progress);
}
public void onStartTrackingTouch(SeekBar seekBar)
{
    mTrackingText.setText("正在调节");
}
//停止拖动
public void onStopTrackingTouch(SeekBar seekBar)
{
    mTrackingText.setText("停止调节");
}
}

```

4.2.21 状态栏提示 (Notification、NotificationManager)

当有未接电话或短信时，在 Android 手机的顶部状态栏就会出现一个小图标，提示用户有没有处理的快讯，这时用触笔按住状态栏往下拖动时，就可以展开并查看这些快讯。Android 平台专门提供了 NotificationManager 来管理状态栏信息，提供了 Notification 来处理这些快讯信息。因此，我们就可以很轻松地实现这一功能。

首先通过 getSystemService 方法得到 NotificationManager 对象，我们可以对 Notification 的内容、图标、标题等属性进行设置；然后通过 notify 方法来执行一个 Notification 快讯。Android 项目 Examples_04_23 运行效果如图 4-54 所示，当用户点击一个按钮就发出一个 Notification 快讯，这时手机顶部状态栏如图 4-55 所示，向下拖动状态栏如图 4-56 所示，点击一则快讯查看如图 4-57 所示。具体实现参见本书所附代码：第 4 章\Examples_04_23。



图 4-54 Notification、NotificationManager 示例

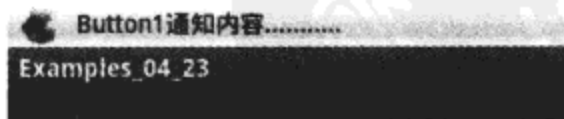


图 4-55 状态栏上的通知图标及内容

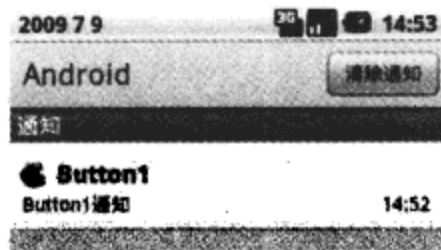


图 4-56 展开状态栏的 Notification 快讯



图 4-57 查看 Notification 快讯

布局文件很简单，就是一个 TextView 和 4 个 Button，请参见本书所附代码：第 4 章\Examples_04_23\res\layout。

在查看 Notification 快讯时需要一个界面来显示，如代码清单 4-44 所示。

代码清单 4-44 第 4 章\Examples_04_23\src\com\yarin\android\Examples_04_23\Activity02.java

```
public class Activity02 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //这里直接限制一个 TextView
        setContentView(R.layout.main2);
    }
}
```

千万不要忘记，上面我们看到本例中使用了两个 Activity，所以我们需要在 AndroidManifest.xml 中进行注册。

最后我们来分析主类的实现，如代码清单 4-45 所示。

代码清单 4-45 第 4 章\Examples_04_23\src\com\yarin\android\Examples_04_23\Activity01.java

```
public class Activity01 extends Activity
{
    Button m_Button1, m_Button2, m_Button3, m_Button4;

    //声明通知（消息）管理器
    NotificationManager m_NotificationManager;
    Intent m_Intent;
    PendingIntent m_PendingIntent;
    //声明 Notification 对象
    Notification m_Notification;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //初始化 NotificationManager 对象
        m_NotificationManager = (NotificationManager) getSystemService(
            NOTIFICATION_SERVICE);

        //获取 4 个按钮对象
        m_Button1 = (Button) findViewById(R.id.Button01);
        m_Button2 = (Button) findViewById(R.id.Button02);
        m_Button3 = (Button) findViewById(R.id.Button03);
        m_Button4 = (Button) findViewById(R.id.Button04);

        //点击通知时转移内容
        m_Intent = new Intent(Activity01.this, Activity02.class);
        //主要是设置点击通知时显示内容的类
        m_PendingIntent = PendingIntent.getActivity(Activity01.this, 0, m_Intent, 0);
        //构造 Notification 对象
        m_Notification = new Notification();

        m_Button1.setOnClickListener(new Button.OnClickListener() {
```

```

public void onClick(View v)
{
    //设置通知在状态栏显示的图标
    m_Notification.icon = R.drawable.img1;
    //当我们点击通知时显示的内容
    m_Notification.tickerText = "Button1 通知内容.....";
    //通知时发出默认的声音
    m_Notification.defaults = Notification.DEFAULT_SOUND;
    //设置通知显示的参数
    m_Notification.setLatestEventInfo(Activity01.this, "Button1",
    "Button1 通知", m_PendingIntent);
    //可以理解为执行这个通知
    m_NotificationManager.notify(0, m_Notification);
}
});
...
}
}

```

4.2.22 对话框中的进度条 (ProgressDialog)

前面学习了对话框和进度条，现在将进度条加入到对话框中，使得对话框更加完善。我们知道进度条有长形和圆形两种，这里将两种分别加入到对话框中。先来看看示例（参见本书所附代码：第4章\Examples_04_24）的运行效果如图4-58所示，当点击“圆形进度条”按钮时，如图4-59所示；当点击“长形进度条”按钮时，如图4-60所示。

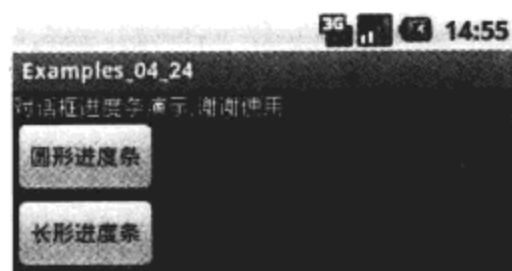


图 4-58 ProgressDialog 示例

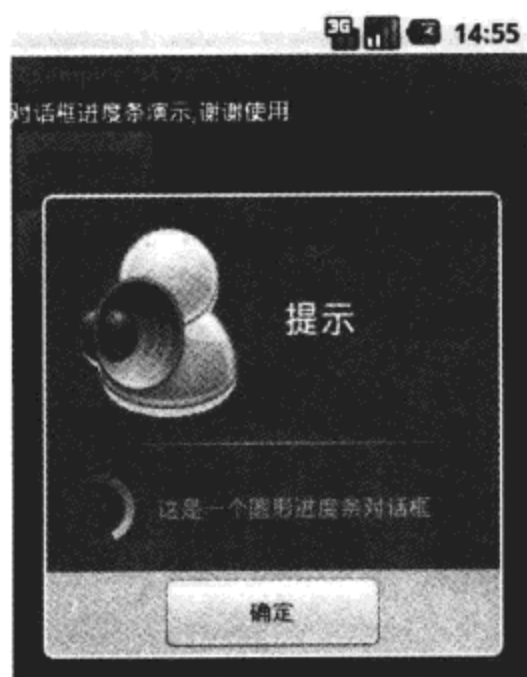


图 4-59 圆形带按钮的 ProgressDialog

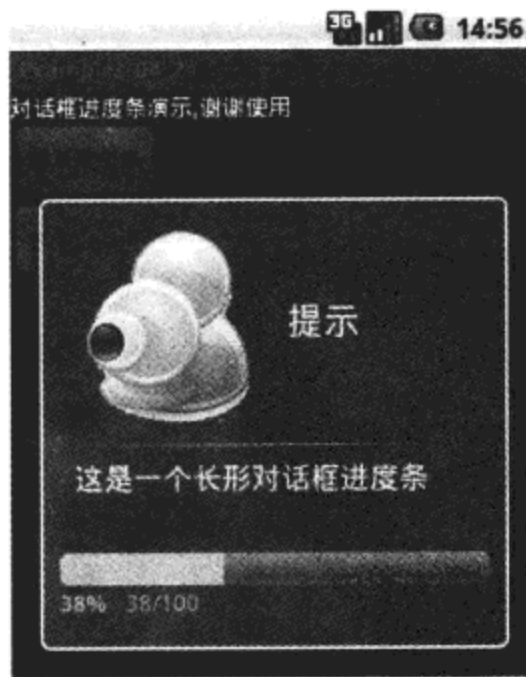


图 4-60 长形 ProgressDialog

首先我们需要创建 ProgressDialog 对象，当然这里同样使用了线程来控制进度条显示，另外可以使用以下方法来设置 ProgressDialog。

❑ **setProgressStyle**: 设置进度条风格，风格为圆形，旋转的；

- ❑ setTitle: 设置 ProgressDialog 标题;
- ❑ setMessage: 设置 ProgressDialog 提示信息;
- ❑ setIcon: 设置 ProgressDialog 标题图标;
- ❑ setIndeterminate: 设置 ProgressDialog 的进度条是否不明确;
- ❑ setCancelable: 设置 ProgressDialog 是否可以按返回按钮取消;
- ❑ setButton: 设置 ProgressDialog 的一个 Button (需要监听 Button 事件);
- ❑ show: 显示 ProgressDialog。

首先我们看看如何通过 XML 布局文件来实现, 如代码清单 4-46 所示。

代码清单 4-46 第 4 章\Examples_04_24\res\layout\main.xml

```
<?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/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="圆形进度条" />
<Button
    android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="长形进度条" />
</LinearLayout>
```

同样我们需要监听这些按钮的事件, 如代码清单 4-47 所示。

代码清单 4-47 第 4 章\Examples_04_24\src\com\yarin\android\Examples_04_24\Activity01.java

```
public class Activity01 extends Activity
{
    private Button mButton01,mButton02;

    int m_count = 0;
    //声明进度条对话框
    ProgressDialog m_pDialog;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //得到按钮对象
        mButton01 = (Button)findViewById(R.id.Button01);
        mButton02 = (Button)findViewById(R.id.Button02);
```



```

//设置 mButton01 的事件监听
mButton01.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        // TODO Auto-generated method stub

        //创建 ProgressDialog 对象
        m_pDialog = new ProgressDialog(Activity01.this);

        // 设置进度条风格, 风格为圆形, 旋转的
        m_pDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);

        // 设置 ProgressDialog 标题
        m_pDialog.setTitle("提示");

        // 设置 ProgressDialog 提示信息
        m_pDialog.setMessage("这是一个圆形进度条对话框");

        // 设置 ProgressDialog 标题图标
        m_pDialog.setIcon(R.drawable.img1);

        // 设置 ProgressDialog 的进度条是否不明确
        m_pDialog.setIndeterminate(false);

        // 设置 ProgressDialog 是否可以按返回按钮取消
        m_pDialog.setCancelable(true);

        // 设置 ProgressDialog 的一个 Button
        m_pDialog.setButton("确定", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int i)
            {
                //点击“确定”按钮取消对话框
                dialog.cancel();
            }
        });

        // 让 ProgressDialog 显示
        m_pDialog.show();
    }
});

//设置 mButton02 的事件监听
mButton02.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        // TODO Auto-generated method stub

        m_count = 0;

        // 创建 ProgressDialog 对象
        m_pDialog = new ProgressDialog(Activity01.this);

        // 设置进度条风格, 风格为长形
        m_pDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    }
});

```

```

// 设置 ProgressDialog 标题
m_pDialog.setTitle("提示");

// 设置 ProgressDialog 提示信息
m_pDialog.setMessage("这是一个长形对话框进度条");

// 设置 ProgressDialog 标题图标
m_pDialog.setIcon(R.drawable.img2);

// 设置 ProgressDialog 进度条进度
m_pDialog.setProgress(100);

// 设置 ProgressDialog 的进度条是否不明确
m_pDialog.setIndeterminate(false);

// 设置 ProgressDialog 是否可以按返回按钮取消
m_pDialog.setCancelable(true);

// 让 ProgressDialog 显示
m_pDialog.show();

new Thread()
{
    public void run()
    {
        try
        {
            while (m_count <= 100)
            {
                // 由线程来控制进度
                m_pDialog.setProgress(m_count++);
                Thread.sleep(100);
            }
            m_pDialog.cancel();
        }
        catch (InterruptedException e)
        {
            m_pDialog.cancel();
        }
    }
}.start();
});
}
}

```

4.3 界面布局

上一节我们学习了 Android 系统的一些常用组件，一个完整的 UI 界面需要将这些组件按照一定的样式进行布局，这就要使用 AndroidXML 布局文件来完成了。AndroidXML 布局文件的大体结构很简单。它是一个标签的树，每个标签就是 View 类的名字。可以使用任何继承自 View 类的名字

作为标签的名字，包括代码中自定义的 View 类。通过这个结构可以很容易地构建界面，它比你在源代码中使用的结构和语法更简单。这个模式的设计灵感来自于 Web 开发，就是可以将界面和应用程序逻辑分离的模式。这一节我们将分析几个常用的布局框架。

- ❑ **LinearLayout**: 线性布局，可以使用垂直线性布局，也可以使用水平线性布局，在 **LinearLayout** 里面可以放多个控件，但是一行（列）只能放一个控件。
- ❑ **RelativeLayout**: 相对布局。**RelativeLayout** 里面可以放多个控件，不过控件的位置都是相对位置。
- ❑ **TableLayout**: 表单布局。这要和 **TableRow** 配合使用，很像 HTML 里面的 Table。
- ❑ **TabWidget**: 切换卡。这是一个特殊的框架，通过继承 **TabActivity** 而来，实现标签切换的功能。

当然还有一些其他的框架，比如，**FrameLayout** 里面只可以有一个控件，并且不能设计这个控件的位置，控件会放到左上角；**AbsoluteLayout** 里面可以放多个控件，并且可以自己定义控件的 *x*、*y* 的位置。下面我们就来学习这些布局的使用。

4.3.1 垂直线性布局

上面说过，**LinearLayout** 是一行（列）只能放置一个控件的线性布局，所以当有很多控件需要在一个界面中列出来时，可以使用 **LinearLayout** 布局，下面我们先来看一个通过 **LinearLayout** 布局的例子（参见本书所附代码：第4章\Examples_04_25），运行效果如图4-61所示，实现过程如代码清单4-48所示。

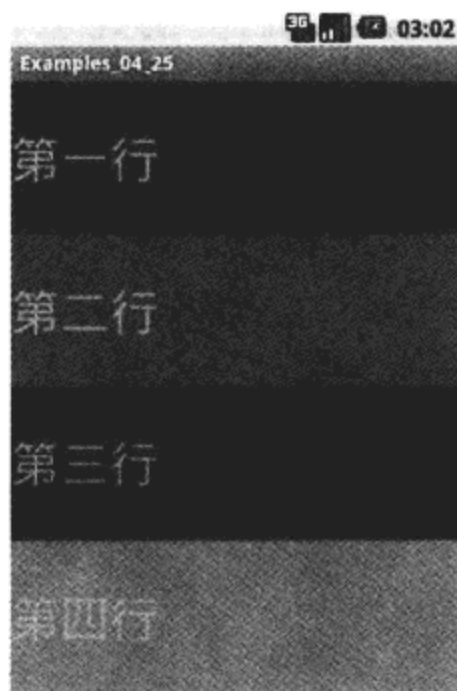


图 4-61 LinearLayout 垂直线性布局

代码清单 4-48 第4章\Examples_04_25\res\layout\main.xml

```
<?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:text="第一行"
        android:gravity="center_vertical"
        android:textSize="15pt"
        android:background="#aa0000"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

    <TextView
        android:text="第二行"
        android:textSize="15pt"
        android:gravity="center_vertical"
        android:background="#00aa00"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

    <TextView
        android:text="第三行"
        android:textSize="15pt"
        android:gravity="center_vertical"
        android:background="#0000aa"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="第四行"
        android:textSize="15pt"
        android:gravity="center_vertical"
        android:background="#aaaa00"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
</LinearLayout>

```

代码清单 4-54 中，我们通过 `android:orientation="vertical"` 声明了这个线性布局以垂直方式排版；通过 `android:layout_width="fill_parent"` 定义当前视图在屏幕上可以使用的宽度，`fill_parent` 即填充整个屏幕；`android:layout_height="wrap_content"` 表示随着文字栏位的不同而改变这个视图的宽度或者高度，类似自动设置宽度或者高度。`layout_weight` 用于给一个线性布局中的诸多视图的重要度赋值，所有的视图都有一个“`layout_weight`”值，默认为 0，意思是需要显示多大的视图就占据多大的屏幕空间。若赋一个大于 0 的值，则将父视图中的可用空间分割，具体分割大小取决于每一个视图的 `layout_weight` 值以及该值在当前屏幕布局的整体 `layout_weight` 值和在其他视图屏幕布局的 `layout_weight` 值中所占的比率而定。

4.3.2 水平线性布局

上一节我们学习了 `LinearLayout` 中的垂直线性布局，水平线性布局每一列只能放置一个控件，与垂直线性布局不同的地方就只是通过 `android:orientation="horizontal"` 来声明是水平布局即可，具体示例（参见本书所附代码：第 4 章\Examples_04_26）运行效果如图 4-62 所示，实现过程如代码清单 4-49 所示。

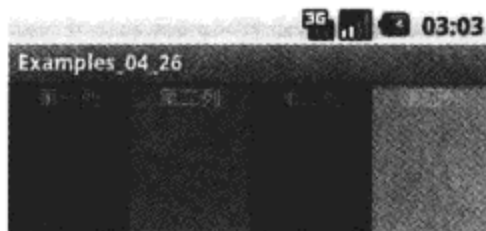


图 4-62 LinearLayout 水平线性布局

代码清单 4-49 第4章\Examples_04_26\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:text="第一列"
        android:gravity="center_horizontal"
        android:background="#aa0000"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="第二列"
        android:gravity="center_horizontal"
        android:background="#00aa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="第三列"
        android:gravity="center_horizontal"
        android:background="#0000aa"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="第四列"
        android:gravity="center_horizontal"
        android:background="#aaaa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
</LinearLayout>
```

代码清单 4-49 中, `android:gravity="center_horizontal"` 表示水平居中对齐; `android:background="#aaaa00"` 设置了 `TextView` 的背景颜色。

4.3.3 相对布局 (RelativeLayout)

在很多时候, 线性布局还不能满足我们的需求, 比如, 我们要在一行 (列) 上显示多个控件, 这就需要使用 `RelativeLayout` 来进行相对布局, `RelativeLayout` 允许子元素指定它们相对于其他元素或父元素的位置 (通过 ID 指定)。因此, 你可以以右对齐、上下或置于屏幕中央的形式来排列两个

元素。元素按顺序排列，因此如果第一个元素在屏幕的中央，那么相对于这个元素的其他元素将以屏幕中央的相对位置来排列。如果使用 XML 来指定这个布局，在你定义它之前，被关联的元素必须定义。**RelativeLayout** 视图显示了屏幕元素的类名称，下面是每个元素的属性列表。这些属性一部分由元素直接提供，另一部分由容器的 **LayoutParams** 成员（**RelativeLayout** 的子类）提供。**RelativeLayout** 参数有 **Width**、**Height**、**Below**、**AlignTop**、**ToLeft**、**Padding** 和 **MarginLeft**。注意，这些参数中的一部分，其值是相对于其他子元素而言的，所以才称为 **RelativeLayout** 布局。这些参数包括 **ToLeft**、**AlignTop** 和 **Below**，用来指定相对于其他元素的左、上和下的位置。下面我们通过 **RelativeLayout** 布局来实现一个带按钮的输入框（参见本书所附代码：第 4 章\Examples_04_27），运行效果如图 4-63 所示。具体实现如代码清单 4-50 所示。

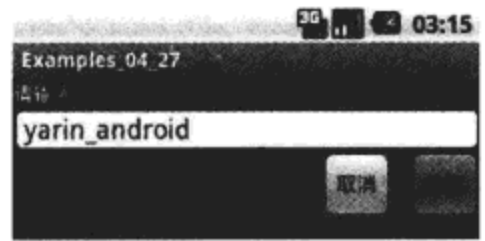


图 4-63 RelativeLayout 布局

代码清单 4-50 第 4 章\Examples_04_27\res\layout\main.xml

```
<?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">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请输入:"/>
    <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"/>
    <Button
        android:id="@+id/ok"
        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="确定" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="取消" />
</RelativeLayout>
```

代码清单 4-58 中，在 **EditText** 中，**android:layout_below="@id/label"** 设置了 **EditText** 处于 **TextView** 下方；在 **Button** 中 **android:layout_below="@id/entry"** 设置该 **Button** 位于 **EditText** 下方。

4.3.4 表单布局 (TableLayout)

TableLayout 将子元素的位置分配到行或列中。Android 的一个 **TableLayout** 由许多的 **TableRow**

组成, 每个 `TableRow` 都会定义一个 `Row`。 `TableLayout` 容器不会显示 `Row`、`Column` 或 `Cell` 的边框线。每个 `Row` 拥有 0 个或多个的 `Cell`, 每个 `Cell` 拥有一个 `View` 对象。表格由列和行组成许多单元格, 允许单元格为空。单元格不能跨列, 这与 HTML 中不一样。列可以被隐藏, 也可以被设置为伸展的, 从而填充可利用的屏幕空间, 也可以被设置为强制列收缩, 直到表格匹配屏幕大小。下面我们来看一个通过 `TableLayout` 布局的例子 (参见本书所附代码: 第 4 章\Examples_04_28), 运行效果如图 4-64 所示。

从图 4-64 我们可以看出这个 `TableLayout` 分成了 3 个部分, 其中每个部分分别由 `TableRow` 组成, `TableRow` 中包括了 `TextView` 控件, 具体实现如代码清单 4-51 所示。

代码清单 4-51 第 4 章\Examples_04_28\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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:layout_column="1"
            android:text="打开..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="保存..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="另存为..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="#FF909090" />
    <TableRow>
```

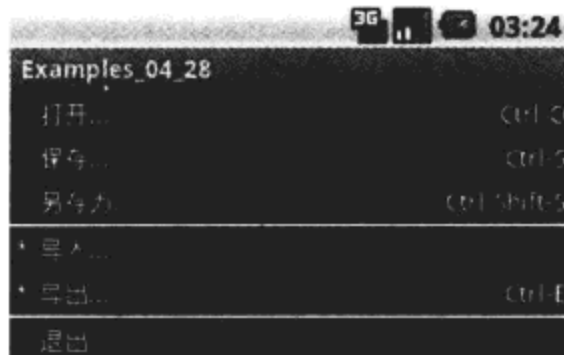


图 4-64 `TableLayout` 视图

```

        <TextView
            android:text="*"
            android:padding="3dip" />
        <TextView
            android:text="导入..."
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="*"
            android:padding="3dip" />
        <TextView
            android:text="导出..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-E"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="#FF909090" />

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="退出"
            android:padding="3dip" />
    </TableRow>
</TableLayout>

```

4.3.5 切换卡 (TabWidget)

TabWidget 类似于 Android 中查看电话簿的界面，通过多个标签切换显示不同的内容。要实现这一效果，首先要了解 TabHost，它是一个用来存放多个 Tab 标签的容器。每一个 Tab 都可以对应自己的布局，比如，电话簿中的 Tab 布局就是一个 List 的线性布局了。

要使用 TabHost，首先需要通过 `getTabHost` 方法来获取 TabHost 的对象，然后通过 `addTab` 方法来向 TabHost 中添加 Tab。当然每个 Tab 在切换时都会产生一个事件，要捕捉这个事件需要设置 TabActivity 的事件监听 `setOnTabChangeListener`。下面我们来看一个具体示例（参见本书所附代码：第 4 章\Examples_04_29），运行效果如图 4-65 所示。当我们点击一个 Tab 切换时，捕捉事件如图 4-66 所示。

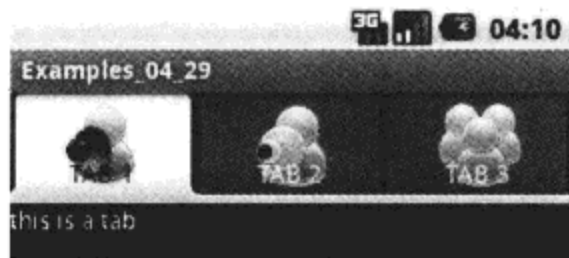


图 4-65 TabWidget 视图



图 4-66 TabWidget 事件处理

首先我们来看布局中如何实现，如代码清单 4-52 所示。其中 Tab 对应的布局通过 `FrameLayout` 作为根布局，然后分别放置了 3 个 `TextView` 控件来显示每个 Tab 标签的内容。

代码清单 4-52 第 4 章\Examples_04_29\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <TextView
                android:id="@+id/textview1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="this is a tab" />
            <TextView
                android:id="@+id/textview2"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="this is another tab" />
            <TextView
                android:id="@+id/textview3"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="this is a third tab" />
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

TabWidget 的使用需要继承自 TabActivity 类, 并实现 setOnTabChangeListener 的 onTabChanged 方法来监听 Tab 的改变, 如代码清单 4-53 所示。

代码清单 4-53 第 4 章\Examples_04_29\src\com\yarin\android\Examples_04_29\Activity01.java

```
public class Activity01 extends TabActivity
{
    //声明 TabHost 对象
    TabHost mTabHost;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //取得 TabHost 对象
        mTabHost = getTabHost();

        /* 为 TabHost 添加标签 */
        //新建一个 newTabSpec(newTabSpec)
        //设置其标签和图标(setIndicator)
        //设置内容(setContent)
        mTabHost.addTab(mTabHost.newTabSpec("tab_test1")
            .setIndicator("TAB 1",getResources().getDrawable(R.drawable.img1))
            .setContent(R.id.textview1));
        mTabHost.addTab(mTabHost.newTabSpec("tab_test2")
            .setIndicator("TAB 2",getResources().getDrawable(R.drawable.img2))
            .setContent(R.id.textview2));
        mTabHost.addTab(mTabHost.newTabSpec("tab_test3")
            .setIndicator("TAB 3",getResources().getDrawable(R.drawable.img3))
            .setContent(R.id.textview3));

        //设置 TabHost 的背景颜色
        mTabHost.setBackgroundColor(Color.argb(150, 22, 70, 150));
        //设置 TabHost 的背景图片资源
        //mTabHost.setBackgroundResource(R.drawable.bg0);

        //设置当前显示哪一个标签
        mTabHost.setCurrentTab(0);

        //标签切换事件处理, setOnTabChangeListener
        mTabHost.setOnTabChangeListener(new OnTabChangeListener()
        {
            // TODO Auto-generated method stub
            @Override
            public void onTabChanged(String tabId)
            {
                Dialog dialog = new AlertDialog.Builder(Activity01.this)
                    .setTitle("提示")
                    .setMessage("当前选中: "+tabId+"标签")
                    .setPositiveButton("确定",
                        new DialogInterface.OnClickListener()
                        {
                            public void onClick(DialogInterface dialog, int whichButton)
                            {
                                dialog.cancel();
                            }
                        }
                    )
                    .create();
                dialog.show();
            }
        });
    }
}
```

```
        }  
        }).create();//创建按钮  
  
        dialog.show();  
    }  
});  
}
```

4.4 小结

本章针对 Android 系统一些常用的组件及界面布局进行了详细的介绍，每个组件都列举了一个简单的例子，通过示例代码可以加深对各个组件功能的认识，让大家能够巧妙地运用这些组件来完成 Android 开发的 UI 设计。这里的很多组件都能实现与用户进行交互的功能，因此必须对需要交互的组件设置事件监听，进而捕捉用户所触发的事件，从而进行相应的处理。

最后，开发者可以进行自定义一些控件来满足自己开发的需要，比如，我们讲述的用对话框来实现登录框效果等。本章内容看似比较繁多，但都是一些相对独立的内容，希望大家好好掌握，后面的开发中会经常使用。



第 5 章

Android 游戏开发

前面介绍了 Android 系统的一些组件和布局，利用这些知识基本可以完成一些应用程序的界面设计。但是，如果要在 Android 平台上开发游戏，这些组件还远远不能满足我们的需求。大家都知道，游戏界面是由大量的美工资源图片构成的，因此，在设计游戏界面时，千万不要使用 Layout 来实现布局。如果像这样将游戏中的对象当成一个组件来处理，那么在开发过程中将会出现很多麻烦的事情，比如界面刷新、事件消息处理等。

其实游戏就是通过状态机让 Canvas 不断地在 View 上画你想要的东西，而这个状态机不仅包括游戏内部的执行，还包括外部的输入。我们还是从游戏开发中比较经典的 MVC 模式说起吧。首先，需要一个用来显示界面的视图，Android 中提供了 View 和 Surfaceview 来实现这个视图；其次，需要控制游戏的整体结构，即 MVC 中的 Control，在 Android 中可以通过 Activity 来实现；最后，需要实现的是一个逻辑类，专门用来处理游戏的逻辑计算等。游戏中的另外一个重要环节就是处理游戏界面和用户交互所发生的事件，比如用户按键、触笔点击等。在 Android 中，View 类提供了 onKeyDown、onKeyUp、onTouchEvent 等方法来捕捉并处理这些事件。准备好了吗？让我们通过本章学会在 Android 平台上开发属于自己的游戏吧！

5.1 Android 游戏开发框架

Borland JBuilder、Eclipse、BEA Workbench 等企业开发环境不外乎都是基于 Apache Struts、Spring、Hibernate 等开发框架，而这些框架几乎都遵循 MVC 设计模式。业务逻辑和描述被分开，通过一个逻辑流控制器来协调源自客户端的请求和服务上将采取的行动。这一途径逐渐成为了开发者们共同遵守的一个不成文的标准。每个框架的内在机制固然是不同的，但是开发者们使用它来设计和实现应用软件的结构是类似的。因此，我们重点来分析 Android 平台提供的 View、Surfaceview 类作为 MVC 中视图类的基类的开发框架。

5.1.1 View 类开发框架

View 类是 Android 的一个超类，这个类几乎包含了所有的屏幕类型。每一个 View 都有一个用于绘画的画布，这个画布可以进行任意扩展。第 4 章中讲解了可以使用 Android 的 XML 来布局视图，在游戏开发中当然也可以自定义视图（View），让这个画布的功能更能满足我们在游戏开发中的需要。在 Android 中，任何一个 View 类都只需要重写 onDraw 方法来实现界面显示，自定义的视图可以是复杂的 3D 实现，也可以是非常简单的文本形式等。

游戏中最重要的就是需要与玩家交互，比如键盘输入、触笔点击等事件，我们如何来处理这些事件呢？Android 中提供了 `onKeyUp`、`onKeyDown`、`onKeyMultiple`、`onKeyPreIme`、`onTouchEvent`、`onTrackballEvent` 等方法，可以轻松地处理游戏中的事件消息。所以，在继承 `View` 时，需要重载这几个方法，当有按键按下或弹起等事件时，按键代码会自动会传输给这些相应的方法来处理。

游戏的核心是不断地绘图和刷新界面，图我们已经通过 `onDraw` 方法绘制了，下面来分析如何刷新界面。Android 中提供了 `invalidate` 方法来实现界面刷新，注意，`invalidate` 不能直接在线程中调用，因为它违背了单线程模型：Android UI 操作并不是线程安全的，并且这些操作必须在 UI 线程中执行，因此 Android 中最常用的方法是利用 `Handler` 来实现 UI 线程的更新。

下面通过实现一个在屏幕上不停变换颜色的矩形来分析 Android 中界面的更新。先来看看运行效果，如图 5-1 所示；通过线程改变颜色之后，如图 5-2 所示；当我们在键盘上按上下键，矩形就会上下移动，如图 5-3 所示。



图 5-1 此时为绿色矩形

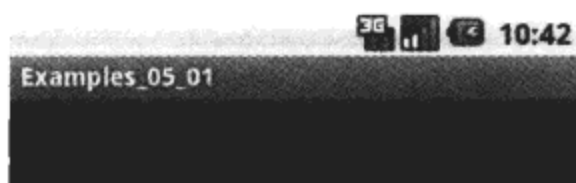


图 5-2 invalidate 方法更新

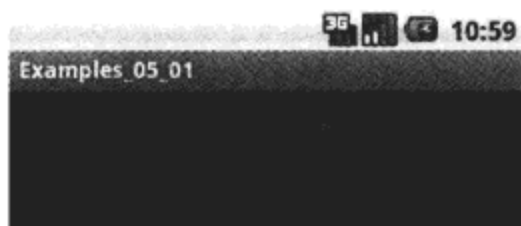


图 5-3 按键处理

该示例中，我们通过实例化 `Handler` 对象并重写 `handleMessage` 方法实现了一个消息接收器，然后在线程中通过 `sendMessage` 方法发送更新界面的消息，当接收器收到更新界面的消息时，便执行 `invalidate` 方法更新屏幕显示，如代码清单 5-1 所示。

代码清单 5-1 第 5 章\Examples_05_01\src\com\lyarin\android\Examples_05_01\GameView.java

```
public class GameView extends View
{
    int miCount = 0;
    int y = 0;
    public GameView(Context context)
    {
        super(context);
    }
    //具体绘图内容我们紧接着就会讲
    public void onDraw(Canvas canvas)
    {
        if (miCount < 100)
        {
            miCount++;
        }
        else
        {

```

```

        miCount = 0;
    }
    //绘图
    Paint mPaint = new Paint();
    switch (miCount%4)
    {
        case 0:
            mPaint.setColor(Color.BLUE);
            break;
        case 1:
            mPaint.setColor(Color.GREEN);
            break;
        case 2:
            mPaint.setColor(Color.RED);
            break;
        case 3:
            mPaint.setColor(Color.YELLOW);
            break;
        default:
            mPaint.setColor(Color.WHITE);
            break;
    }
    //绘制矩形——后面我们将详细讲解
    canvas.drawRect((320-80)/2, y, (320-80)/2+80, y+40, mPaint);
}
}

```

代码清单 5-1 主要用来绘制界面，我们还需要一个用来控制整个应用的操作，比如事件处理、更新频率等，这就需要实现一个 Activity 类，如代码清单 5-2 所示。

代码清单 5-2 第 5 章\Examples_05_01\src\com\yarin\android\Examples_05_01\Activity01.java

```

public class Activity01 extends Activity
{
    private static final int    REFRESH            = 0x000001;
    /* 声明 GameView 类对象 */
    private GameView            mGameView= null;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        /* 实例化 GameView 对象 */
        this.mGameView = new GameView(this);
        //设置显示为我们自定义的 View(GameView)
        setContentView(mGameView);
        //开启线程
        new Thread(new GameThread()).start();
    }
    Handler myHandler= new Handler()
    {
        //接收到消息后处理
        public void handleMessage(Message msg)
        {
            switch (msg.what)
            {
                case Activity01.REFRESH:
                    mGameView.invalidate();
                    break;
            }
        }
    }
}

```

```

        }
        super.handleMessage(msg);
    }
};
class GameThread implements Runnable
{
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            Message message = new Message();
            message.what = Activity01.REFRESH;
            //发送消息
            Activity01.this.myHandler.sendMessage(message);
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
        }
    }
}
/**
 * 当然可以将 GameThread 类这样写
 * 同样可以更新界面, 并且不再需要
 * Handler 接收消息
class GameThread implements Runnable
{
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            mGameView.postInvalidate();
        }
    }
}
*/
//详细事件处理见第 3 章
//当然这些事件也可以写在 GameView 中
//触笔事件
public boolean onTouchEvent(MotionEvent event)
{
    return true;
}
//按键按下事件
public boolean onKeyDown(int keyCode, KeyEvent event)

```

```

{
    return true;
}
//按键弹起事件
public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        //上方向键
        case KeyEvent.KEYCODE_DPAD_UP:
            mGameView.y-=3;
            break;
        //下方向键
        case KeyEvent.KEYCODE_DPAD_DOWN:
            mGameView.y+=3;
            break;
    }
    return false;
}

public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
{
    return true;
}
}

```

在 Android 中还提供了一个更新界面的方法 `postInvalidate`，该方法使用起来更加简单，不再需要 `Handler`，可以直接在线程中更新，如代码清单 5-3 中注释了的部分。希望大家能同时掌握这两种方法，并熟练运用。下面将介绍 Android 中的另一个视图 `SurfaceView`，它功能更加强大，给开发者的空间和自由度更大。

5.1.2 SurfaceView 类开发框架

上一节讲解了 `View` 类游戏开发框架，但是，当需要开发复杂的游戏，而且对程序的执行效率要求很高时，`View` 类就不能满足需求了，这时必须用 `SurfaceView` 类进行开发。例如，对速度要求很高的游戏，可以使用双缓冲来显示。游戏中的背景、人物、动画等都需要绘制在一个画布（`Canvas`）上，而 `SurfaceView` 可以直接访问一个画布，`SurfaceView` 是提供给需要直接画像素而不是使用窗体部件的应用使用的。Android 图形系统中的一个重要概念是 `Surface`，`View` 及其子类（如 `TextView` 和 `Button`）要画在 `Surface` 上。每个 `Surface` 创建一个 `Canvas` 对象（但属性时常改变），用来管理 `View` 在 `Surface` 上的绘图操作。

在使用 `SurfaceView` 开发时需要注意的是，使用它绘图时，一般都是出现在最顶层。使用时还需要对其进行创建、销毁，情况改变时进行监视，这就要实现 `SurfaceHolder.Callback` 接口，如果要对被绘制的画布进行裁剪，控制其大小时都需要使用 `SurfaceHolder` 来完成处理。在程序中，`SurfaceHolder` 对象需要通过 `getHolder` 方法来获得，同时还需要 `addCallback` 方法来添加“回调函数”。

- ❑ `surfaceChanged`：在 `Surface` 的大小发生改变时激发。
- ❑ `surfaceCreated`：在创建 `Surface` 时激发。
- ❑ `surfaceDestroyed`：在销毁 `Surface` 时激发。

- ❑ **addCallback**: 给 **SurfaceView** 添加一个回调函数。
- ❑ **lockCanvas**: 锁定画布, 绘图之前必须锁定画布才能得到当前画布对象。
- ❑ **unlockCanvasAndPost**: 开始绘制时锁定了画布, 绘制完成之后解锁画布。
- ❑ **removeCallback**: 从 **SurfaceView** 中移除回调函数。

SurfaceView 和 **View** 的明显不同之处在于, **SurfaceView** 不需要通过线程来更新视图, 但在绘制之前必须使用 **lockCanvas** 方法锁定画布, 并得到画布, 然后在画布上绘制; 当绘制完成后, 使用 **unlockCanvasAndPost** 方法来解锁画布, 于是才能显示在屏幕之上。**SurfaceView** 类的事件处理则和 **View** 一样, 因此这里不再重复了。

下面通过一个示例来分析使用 **SurfaceView** 开发的视图类, 该程序实现了绘制一个不断变换颜色的圆, 并且实现了 **SurfaceView** 的事件处理, 其运行效果如图 5-4 所示。当我们按上下键时, 图形移动之后效果如图 5-5 所示。

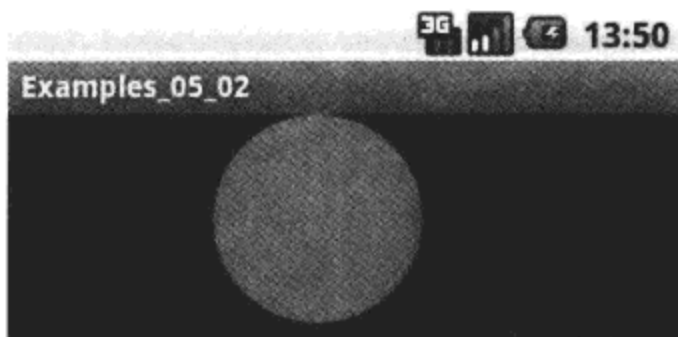


图 5-4 **SurfaceView** 示例

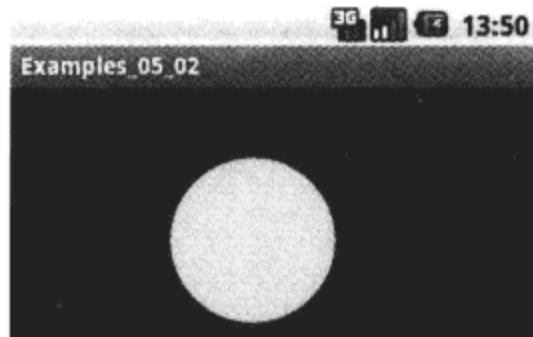


图 5-5 **SurfaceView** 事件处理

本例中通过 **SurfaceView** 实现一个游戏框架, 其控制整个应用流程的 **Activity** 类和上一节差不多, 下面我们主要分析如何实现 **SurfaceView** 框架。

如代码清单 5-3 所示。

代码清单 5-3 第 5 章\Examples_05_02\src\com\yarin\android\Examples_05_02\GameSurfaceView.java

```
public class GameSurfaceView extends SurfaceView
    implements SurfaceHolder.Callback, Runnable
{
    //控制循环
    boolean mbLoop = false;

    //定义 SurfaceHolder 对象
    SurfaceHolder mSurfaceHolder = null;
    int miCount = 0;
    int y = 50;

    public GameSurfaceView(Context context)
    {
        super(context);

        //实例化 SurfaceHolder
        mSurfaceHolder = this.getHolder();

        //添加回调
        mSurfaceHolder.addCallback(this);
        this.setFocusable(true);
    }
}
```

```

        mbLoop = true;
    }

    // 在 Surface 的大小发生改变时激发
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int
height)
    {

    }

    // 在 Surface 创建时激发
    public void surfaceCreated(SurfaceHolder holder)
    {
        //开启绘图线程
        new Thread(this).start();
    }

    // 在 Surface 销毁时激发
    public void surfaceDestroyed(SurfaceHolder holder)
    {
        // 停止循环
        mbLoop = false;
    }

    // 绘图循环
    public void run()
    {
        while (mbLoop)
        {
            try
            {
                Thread.sleep(200);
            }
            catch (Exception e)
            {

            }
            synchronized( mSurfaceHolder )
            {
                Draw();
            }
        }
    }

    // 绘图方法
    public void Draw()
    {
        //锁定画布, 得到 canvas
        Canvas canvas= mSurfaceHolder.lockCanvas();

        if (mSurfaceHolder==null || canvas == null )
        {
            return;
        }

        if (miCount < 100)
        {

```



```

        miCount++;
    }
    else
    {
        miCount = 0;
    }
    // 绘图
    Paint mPaint = new Paint();
    mPaint.setAntiAlias(true);
    mPaint.setColor(Color.BLACK);
    //绘制矩形——清屏作用
    canvas.drawRect(0, 0, 320, 480, mPaint);
    switch (miCount % 4)
    {
    case 0:
        mPaint.setColor(Color.BLUE);
        break;
    case 1:
        mPaint.setColor(Color.GREEN);
        break;
    case 2:
        mPaint.setColor(Color.RED);
        break;
    case 3:
        mPaint.setColor(Color.YELLOW);
        break;
    default:
        mPaint.setColor(Color.WHITE);
        break;
    }
    // 绘制矩形——后面我们将详细讲解
    canvas.drawCircle((320 - 25) / 2, y, 50, mPaint);
    // 绘制后解锁，绘制后必须解锁才能显示
    mSurfaceHolder.unlockCanvasAndPost(canvas);
}
}

```

到这里，我们已经分析了 Android 中游戏开发常用的两个视图类的开发框架，下面学习如何用 Android 中 Graphics 类绘制图形。

5.2 Graphics 类开发

要开发游戏，必须在屏幕上绘制 2D 图形，而在 Android 中需要通过 graphics 类来显示 2D 图形。graphics 中包括了 Canvas（画布）、Paint（画笔）、Color（颜色）、Bitmap（图像）、2D 几何图形等常用类。Graphics 具有绘制点、线、颜色、图像处理、2D 几何图形等功能，下面我们分别介绍 Graphics 的各个类的使用。

5.2.1 Paint 和 Color 类介绍

要绘图，首先得调整画笔，待画笔调整好之后，再将图像绘制到画布上，这样才可以显示在手机屏幕上。Android 中的画笔是 Paint 类，Paint 中包含了很多方法对其属性进行设置，主要方法

(没有全部列出，大家可以查看官方文档) 如下：

- ❑ `setAntiAlias`: 设置画笔的锯齿效果。
- ❑ `setColor`: 设置画笔的颜色。
- ❑ `setARGB`: 设置画笔的 a, r, p, g 值。
- ❑ `setAlpha`: 设置 Alpha 值。
- ❑ `setTextSize`: 设置字体尺寸。
- ❑ `setStyle`: 设置画笔的风格，空心或者实心。
- ❑ `setStrokeWidth`: 设置空心的边框宽度。
- ❑ `getColor`: 得到画笔的颜色。
- ❑ `getAlpha`: 得到画笔的 Alpha 值。

`Color` 类则更加简单，主要定义了一些颜色常量，以及对颜色的转换等。主要的 12 种颜色如表 5-1 所示。

表 5-1 `Color` 类的 12 种颜色

颜色常量	含 义	颜色常量	含 义
<code>Color.BLACK</code>	黑色	<code>Color.GREEN</code>	绿色
<code>Color.BLUE</code>	蓝色	<code>Color.LTGRAY</code>	浅灰色
<code>Color.CYAN</code>	青绿色	<code>Color.MAGENTA</code>	红紫色
<code>Color.DKGRAY</code>	灰黑色	<code>Color.RED</code>	红色
<code>Color.YELLOW</code>	黄色	<code>Color.TRANSPARENT</code>	透明
<code>Color.GRAY</code>	灰色	<code>Color.WHITE</code>	白色

同时还提供了 `Color.rgb` 方法将整型的颜色转换成 `Color` 类型，如 `Color.red` 方法可以提取出红色的值。下面我们来看一个示例，分别说明这些方法的使用，运行效果如图 5-6 所示。具体实现参见本书所附代码：第 5 章\Examples_05_03。

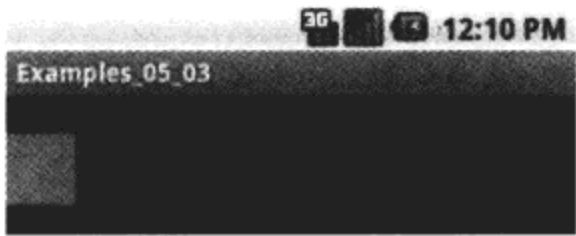


图 5-6 `Paint` 和 `Color` 的使用

首先来看绘图类是如何绘制出矩形图形的，如代码清单 5-4 所示。

代码清单 5-4 第 5 章\Examples_05_03\src\com\yarin\android\Examples_05_03\GameView.java

```
public class GameView extends View implements Runnable
{
    public final static String TAG = "Examples_05_03_GameView"; //
    /* 声明 Paint 对象 */
    private Paint mPaint = null;
```

```

public GameView(Context context)
{
    super(context);
    /* 构建对象 */
    mPaint = new Paint();
    /* 开启线程 */
    new Thread(this).start();
}
public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
    /* 设置 Paint 为无锯齿 */
    mPaint.setAntiAlias(true);
    /* 设置 Paint 的颜色 */
    mPaint.setColor(Color.RED);
    mPaint.setColor(Color.BLUE);
    mPaint.setColor(Color.YELLOW);
    mPaint.setColor(Color.GREEN);
    /* 同样是设置颜色 */
    mPaint.setColor(Color.rgb(255, 0, 0));
    /* 提取颜色 */
    Color.red(0xcccccc);
    Color.green(0xcccccc);
    /* 设置 Paint 的颜色和 Alpha 值(a,r,g,b) */
    mPaint.setARGB(255, 255, 0, 0);
    /* 设置 Paint 的 Alpha 值 */
    mPaint.setAlpha(220);
    /* 这里可以设置为另外一个 Paint 对象 */
    // mPaint.set(new PAINT());
    /* 设置字体的尺寸 */
    mPaint.setTextSize(14);

    // 设置 Paint 的风格为“空心”
    // 当然也可以设置为“实心” (Paint.Style.FILL)
    mPaint.setStyle(Paint.Style.STROKE);
    // 设置“空心”的外框的宽度
    mPaint.setStrokeWidth(5);
    /* 得到 Paint 的一些属性 */
    Log.i(TAG, "paint 的颜色: " + mPaint.getColor());
    Log.i(TAG, "paint 的 Alpha: " + mPaint.getAlpha());
    Log.i(TAG, "paint 的外框的宽度: " + mPaint.getStrokeWidth());
    Log.i(TAG, "paint 的字体尺寸: " + mPaint.getTextSize());
    /* 绘制一个矩形 */
    // 肯定是一个空心的矩形
    canvas.drawRect((320 - 80) / 2, 20, (320 - 80) / 2 + 80, 20 + 40, mPaint);
    /* 设置风格为实心 */
    mPaint.setStyle(Paint.Style.FILL);
    mPaint.setColor(Color.GREEN);
    /* 绘制绿色实心矩形 */
    canvas.drawRect(0, 20, 40, 20 + 40, mPaint);
}
public void run()
{
    while (!Thread.currentThread().isInterrupted())

```

```

    {
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
            Thread.currentThread().interrupt();
        }
        // 使用 postInvalidate 可以直接在线程中更新界面
        postInvalidate();
    }
}

```

在绘制出图形之后还需要通过 Activity 类的 `setContentView` 方法来设置要显示的具体 View 类。

5.2.2 Canvas 类介绍

画笔调整好之后，现在需要绘制到画布上，这就得用到 Canvas 类了。在 Android 中，既然把 Canvas 当作画布，那么就可以在画布上绘制我们想要的东西。除了在画布上绘制之外，还需要设置一些关于画布的属性，比如，画布的颜色、尺寸等。下面来分析 Android 中 Canvas 有哪些功能。首先，Canvas 提供了如下一些方法：

- ❑ `Canvas()`：创建一个空的画布，可以使用 `setBitmap()` 方法来设置绘制的具体画布。
- ❑ `Canvas (Bitmap bitmap)`：以 bitmap 对象创建一个画布，则将内容都绘制在 bitmap 上，因此 bitmap 不得为 NULL。
- ❑ `Canvas (GL gl)`：在绘制 3D 效果时使用，与 OpenGL 相关。
- ❑ `drawColor`：设置 Canvas 的背景颜色。
- ❑ `setBitmap`：设置具体画布。
- ❑ `clipRect`：设置显示区域，即设置裁剪区。
- ❑ `isOpaque`：检测是否支持透明。
- ❑ `rotate`：旋转画布。
- ❑ `setViewport`：设置画布中显示窗口。
- ❑ `skew`：设置偏移量。

上面列举了几个常用的方法。在游戏开发中，我们可能需要对某个精灵执行旋转、缩放和一些其他操作。我们可以通过旋转画布来实现，但是旋转画布时会旋转画布上的所有对象，而我们只是需要旋转其中的一个，这时就需要用到 `save` 方法来锁定需要操作的对象，在操作之后通过 `restore` 方法来解除锁定，运行效果如图 5-7 所示。具体代码请参见本书所附代码：第 5 章\Examples_05_04。我们对左边的矩形执行了旋转操作，而没有旋转右边的矩形，由于我们设置了裁剪区域，因此左边的矩形只能看到一部分，具体代码实现见代码清单 5-5 所示。

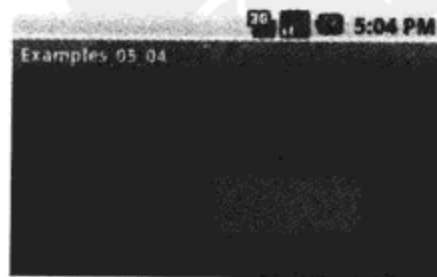


图 5-7 Canvas 基本操作

代码清单 5-5 第5章\Examples_05_04\src\com\yarin\android\Examples_05_04\GameView.java

```

public class GameView extends View implements Runnable
{
    /* 声明 Paint 对象 */
    private Paint mPaint = null;
    public GameView(Context context)
    {
        super(context);
        /* 构建对象 */
        mPaint = new Paint();
        /* 开启线程 */
        new Thread(this).start();
    }
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        /* 设置画布的颜色 */
        canvas.drawColor(Color.BLACK);
        /* 设置取消锯齿效果 */
        mPaint.setAntiAlias(true);
        /* 设置裁剪区域 */
        canvas.clipRect(10, 10, 280, 260);
        /* 锁定画布 */
        canvas.save();
        /* 旋转画布 */
        canvas.rotate(45.0f);
        /* 设置颜色及绘制矩形 */
        mPaint.setColor(Color.RED);
        canvas.drawRect(new Rect(15, 15, 140, 70), mPaint);
        /* 解除画布的锁定 */
        canvas.restore();
        /* 设置颜色及绘制另一个矩形 */
        mPaint.setColor(Color.GREEN);
        canvas.drawRect(new Rect(150, 75, 260, 120), mPaint);
    }

    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            // 使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}

```

5.2.3 几何图形绘制

前两节调整好了画笔，设置好了画布，现在需要在画布上绘制内容了。其实前面我们已经看到了在屏幕上显示的矩形、圆形、三角形等几何图形，下面我们来看看在 Android 中可以绘制出哪些几何图形，如表 5-2 所示。

表 5-2 Android 中可以绘制的几何图形

方 法	说 明
drawRect	绘制矩形
drawCircle	绘制圆形
drawOval	绘制椭圆
drawPath	绘制任意多边形
drawLine	绘制直线
drawPoint	绘制点

下面我们通过一个示例来看看如何绘制这些几何图形，运行效果如图 5-8 所示，完整源码请参见本书所附代码：第 5 章\Examples_05_05。示例中分别绘制了空心 and 实心的几何图形，具体实现见代码清单 5-6 所示。



图 5-8 几何图形绘制

代码清单 5-6 第 5 章\Examples_05_05\src\com\yarin\android\Examples_05_05\GameView.java

```
public class GameView extends View implements Runnable
{
    /* 声明 Paint 对象 */
    private Paint mPaint = null;
    private GameView2 mGameView2 = null;
    public GameView(Context context)
    {
        super(context);
        /* 构建对象 */
        mPaint = new Paint();
        mGameView2 = new GameView2(context);
        /* 开启线程 */
    }
}
```



```

        new Thread(this).start();
    }
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        /* 设置画布为黑色背景 */
        canvas.drawColor(Color.BLACK);
        /* 取消锯齿 */
        mPaint.setAntiAlias(true);
        mPaint.setStyle(Paint.Style.STROKE);
        {
            /* 定义矩形对象 */
            Rect rect1 = new Rect();
            /* 设置矩形大小 */
            rect1.left = 5;
            rect1.top = 5;
            rect1.bottom = 25;
            rect1.right = 45;
            mPaint.setColor(Color.BLUE);
            /* 绘制矩形 */
            canvas.drawRect(rect1, mPaint);

            mPaint.setColor(Color.RED);
            /* 绘制矩形 */
            canvas.drawRect(50, 5, 90, 25, mPaint);

            mPaint.setColor(Color.YELLOW);
            /* 绘制圆形(圆心 x, 圆心 y, 半径 r,p) */
            canvas.drawCircle(40, 70, 30, mPaint);
            /* 定义椭圆对象 */
            RectF rectf1 = new RectF();
            /* 设置椭圆大小 */
            rectf1.left = 80;
            rectf1.top = 30;
            rectf1.right = 120;
            rectf1.bottom = 70;
            mPaint.setColor(Color.LTGRAY);
            /* 绘制椭圆 */
            canvas.drawOval(rectf1, mPaint);
            /* 绘制多边形 */
            Path path1 = new Path();

            /*设置多边形的点*/
            path1.moveTo(150+5, 80-50);
            path1.lineTo(150+45, 80-50);
            path1.lineTo(150+30, 120-50);
            path1.lineTo(150+20, 120-50);
            /* 使这些点构成封闭的多边形 */
            path1.close();
            mPaint.setColor(Color.GRAY);
            /* 绘制多边形 */
            canvas.drawPath(path1, mPaint);

            mPaint.setColor(Color.RED);
            mPaint.setStrokeWidth(3);
            /* 绘制直线 */
            canvas.drawLine(5, 110, 315, 110, mPaint);
        }
    }
}

```

```

    }
    //下面绘制实心几何体

    ...

    /* 通过 ShapeDrawable 来绘制几何图形 */
    mGameView2.DrawShape(canvas);
}

public void run()
{
    while (!Thread.currentThread().isInterrupted())
    {
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
            Thread.currentThread().interrupt();
        }
        //使用 postInvalidate 可以直接在线程中更新界面
        postInvalidate();
    }
}
}

```

扩展学习

当然，在 Android 中还可以通过 ShapeDrawable 来绘制图像，ShapeDrawable 可以设置画笔的形状。通过 getPaint 方法可以得到 Paint 对象，可以像前面一样设置这个画笔的颜色、尺寸等属性。然而，在 ShapeDrawable 中提供了 setBounds 方法来设置图形显示的区域，最后通过 ShapeDrawable 的 Draw 方法将图形显示到屏幕上，具体实现如代码清单 5-7 所示。完整源码请参见本书所附代码：第 5 章\Examples_05_05。

代码清单 5-7 第 5 章\Examples_05_05\src\com\yarin\android\Examples_05_05\GameView2.java

```

//通过 ShapeDrawable 来绘制几何图形
public class GameView2 extends View
{
    /* 声明 ShapeDrawable 对象 */
    ShapeDrawable mShapeDrawable = null;

    public GameView2(Context context)
    {
        super(context);
    }

    public void DrawShape(Canvas canvas)
    {
        /* 实例化 ShapeDrawable 对象并说明是绘制一个矩形 */
        mShapeDrawable = new ShapeDrawable(new RectShape());

        //得到画笔 Paint 对象并设置其颜色
        mShapeDrawable.getPaint().setColor(Color.RED);
    }
}

```

```

Rect bounds = new Rect(5, 250, 55, 280);

/* 设置图像显示的区域 */
mShapeDrawable.setBounds(bounds);

/* 绘制图像 */
mShapeDrawable.draw(canvas);
/*=====*/
/* 实例化 ShapeDrawable 对象并说明是绘制一个椭圆 */
mShapeDrawable = new ShapeDrawable(new OvalShape());

//得到画笔 Paint 对象并设置其颜色
mShapeDrawable.getPaint().setColor(Color.GREEN);

/* 设置图像显示的区域 */
mShapeDrawable.setBounds(70, 250, 150, 280);

/* 绘制图像 */
mShapeDrawable.draw(canvas);

Path path1 = new Path();
/*设置多边形的点*/
path1.moveTo(150+5, 80+80-50);
path1.lineTo(150+45, 80+80-50);
path1.lineTo(150+30, 80+120-50);
path1.lineTo(150+20, 80+120-50);
/* 使这些点构成封闭的多边形 */
path1.close();

//PathShape 后面两个参数分别是宽度和高度
mShapeDrawable = new ShapeDrawable(new PathShape(path1,150,150));

//得到画笔 Paint 对象并设置其颜色
mShapeDrawable.getPaint().setColor(Color.BLUE);

/* 设置图像显示的区域 */
mShapeDrawable.setBounds(100, 170, 200, 280);

/* 绘制图像 */
mShapeDrawable.draw(canvas);
}
}

```

5.2.4 字符串绘制

在游戏开发中，我们不可能全部用图片来显示，很多时候需要绘制字符串，比如开发一个 RPG 的游戏，有大量的对话内容，这时如果全部用图片来显示肯定不行。下面我们就看看如何在 Canvas 画布中绘制字符串吧！

Android 中提供了一系列的 `drawText` 方法来绘制字符串，在绘制字符串之前需要设置画笔对象，包括字符串的尺寸、颜色等属性。使用 `FontMetrics` 来规划字体的属性，可以通过 `getFontMetrics` 方法来获得系统字体的相关内容。下面来看一个大量文本自动换行、翻页的示例是如何操作字符串的，运行效果如图 5-9 所示，当我们按下方向键翻页时，如图 5-10 所示。

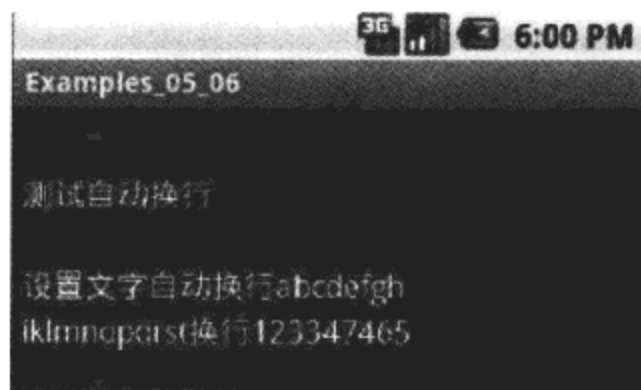


图 5-9 字符串绘制

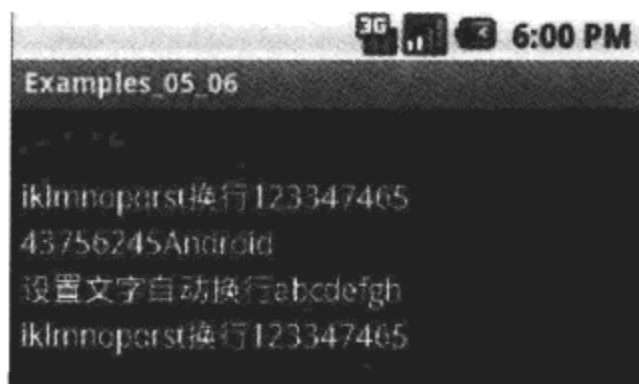


图 5-10 字符串自动换行翻页

关于字符串自动换行的完整代码请参见本书所附代码：第 5 章\Examples_05_06\src\com\yarin\android\Examples_05_06\TextUtil.java。下面我们列出一些在字符串处理时常用的方法：

setTextSize: 设置字符串的尺寸；

setARGB: 设置颜色 (ARGB)；

getTextWidths: 取得字符串的宽度；

setFlags (Paint.ANTI_ALIAS_FLAG): 消除锯齿。

当然要取得字符串的宽度还可以使用 **measureText** 方法，得到字符串宽度，可以使用如下代码：

```
FontMetrics fm = m_paint.getFontMetrics();
m_iFontHeight = (int) Math.ceil(fm.descent - fm.top) + 2;
```

到这里，我们已经可以绘制出一些几何图形和字符串了。一个游戏当然不是这些简简单单的几何图形加上字符串就能组成的，还需要一些美工资源的支持，界面才能更加的漂亮。下面将介绍如何在 Canvas 中绘制外部图片资源。

5.2.5 图像绘制

在 Android 中，项目目录下的“res\drawable”用来放置该项目的一些图片资源，那么如何来显示这些图片资源呢？Android 中提供了 **Bitmap** 来存放这些资源，如下代码可以通过一个资源索引获得其图像对象 **Bitmap**。

```
((BitmapDrawable) getResources().
getDrawable(资源索引)).getBitmap();
```

然后使用 **drawBitmap** 方法将图片显示到屏幕上。如下代码将一个名为 **bitmap** 的 **Bitmap** 对象显示在 (x,y) 坐标上。

```
canvas.drawBitmap(bitmap, x, y, null);
```

Bitmap 还提供了一些方法，比如 **getHeight** 方法和 **getWidth** 方法可以获得这个图片的高度和宽度，我们将在下面的示例中分析如何使用这些方法，并将图片显示在手机屏幕上，运行效果如图 5-11 所示。

图 5-11 中显示了两张图片，用户可以通过左右键来控制图片左右移动，具体实现如代码清单 5-8 所示。完整源码请参见本书所附代码：第 5 章\



图 5-11 图片显示

Examples_05_07。

代码清单 5-8 第5章\Examples_05_07\src\com\yarin\android\Examples_05_07\GameView.java

```
public class GameView extends View implements Runnable
{
    /* 声明 Paint 对象 */
    private Paint mPaint = null;

    /* 创建两个图片对象 */
    Bitmap mBitQQ = null;
    Bitmap mBitDestTop = null;

    int mIDTX = 0;

    public GameView(Context context)
    {
        super(context);

        mPaint = new Paint();

        mIDTX = 0;

        /* 从资源文件中装载图片 */
        //getResources()->得到 Resources
        //getDrawable()->得到资源中的 Drawable 对象, 参数为资源索引 ID
        //getBitmap()->得到 Bitmap
        mBitQQ = ((BitmapDrawable) getResources().getDrawable(
            R.drawable.qq)).getBitmap();

        mBitDestTop = ((BitmapDrawable) getResources().getDrawable(
            R.drawable.desktop)).getBitmap();

        new Thread(this).start();
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);

        /* 清屏效果 */
        canvas.drawColor(Color.GRAY);

        /* 在屏幕(0,0)处绘制图片 mBitQQ */
        GameView.drawImage(canvas, mBitQQ, 0, 0);

        /* 在指定位置按指定裁剪的区域进行绘制 */
        //getWidth()->得到图片的宽度
        //getHeight()->得到图片的高度
        GameView.drawImage(canvas, mBitDestTop, mIDTX, mBitQQ.getHeight(),
            mBitDestTop.getWidth(), mBitDestTop.getHeight()/2, 0, 0);
    }

    // 触笔事件
    public boolean onTouchEvent(MotionEvent event)
    {
        return true;
    }
}
```

```

    }
    // 按键按下事件
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        //左方向键
        if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT)
        {
            if (miDTX > 0)
            {
                miDTX--;
            }
        }
        //右方向键
        else if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT)
        {
            if (miDTX+mBitDestTop.getWidth() < 320)
            {
                miDTX++;
            }
        }
        return true;
    }

    /**
     * 线程处理
     */
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }

    /**-----
     * 绘制图片
     *
     * @param      x 屏幕上的 x 坐标
     * @param      y 屏幕上的 y 坐标
     * @param      w 要绘制的图片的宽度
     * @param      h 要绘制的图片的高度
     * @param      bx 图片上的 x 坐标
     * @param      by 图片上的 y 坐标
     *
     * @return      null
     *-----*/
    public static void drawImage(Canvas canvas, Bitmap blt, int x, int y, int w,
    int h, int bx, int by)

```



```

    {
        Rect src = new Rect();// 图片裁剪区域
        Rect dst = new Rect();// 屏幕裁剪区域

        src.left = bx;
        src.top = by;
        src.right = bx + w;
        src.bottom = by + h;

        dst.left = x;
        dst.top = y;
        dst.right = x + w;
        dst.bottom = y + h;
        canvas.drawBitmap(blt, src, dst, null);

        src = null;
        dst = null;
    }

    /**
     * 绘制一个 Bitmap
     * @param canvas 画布
     * @param bitmap 图片
     * @param x      屏幕上的 x 坐标
     * @param y      屏幕上的 y 坐标
     */
    public static void drawImage(Canvas canvas, Bitmap bitmap, int x, int y)
    {
        /* 绘制图像 */
        canvas.drawBitmap(bitmap, x, y, null);
    }
}

```

代码清单 5-8 中我们封装了两个绘制图片的方法 `drawImage`，实现了在指定图片裁剪区域进行绘制。我们在做游戏地图时，肯定不会是将一整张大图绘制上去，而是通过小块组合成一张大的地图，这时就需要对图片进行裁剪。为了节省资源，我们还需要对图片执行旋转和缩放等操作，下面将讲述图片的旋转操作。

5.2.6 图像旋转

在 Android 中进行图像旋转需要使用 `Matrix`，它包含一个 3×3 的矩阵，专门用于进行图像变换匹配。`Matrix` 没有结构体，它必须被初始化，通过 `reset` 方法或 `set` 方法来实现。通过 `setRotate` 设置旋转角度，用 `createBitmap` 创建一个经过旋转等处理的 `Bitmap` 对象，然后将 `Bitmap` 绘制到屏幕之上，于是就实现了旋转操作。下面我们通过一个示例来说明 `Matrix` 的使用以及旋转的方式，运行效果如图 5-12 所示，具体实现如代码清单 5-9 所示。可以通过左右键来旋转图像。完整源码请参见本书所附代码：第 5



图 5-12 图像旋转

章\Examples_05_08。

代码清单 5-9 第 5 章\Examples_05_08\src\com\yarin\android\Examples_05_08\GameView.java

```
public class GameView extends View implements Runnable
{
    /* 声明 Bitmap 对象 */
    Bitmap mBitQQ = null;
    int BitQQwidth = 0;
    int BitQQheight = 0;

    float Angle= 0.0f;

    /* 构建 Matrix 对象 */
    Matrix mMatrix = new Matrix();

    public GameView(Context context)
    {
        super(context);

        /* 装载资源 */
        mBitQQ = ((BitmapDrawable) getResources().getDrawable(
            R.drawable.qq)).getBitmap();

        /* 得到图像的宽度和高度 */
        BitQQwidth = mBitQQ.getWidth();
        BitQQheight = mBitQQ.getHeight();

        /* 开启线程 */
        new Thread(this).start();
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);

        /* 重置 mMatrix */
        mMatrix.reset();

        /* 设置旋转 */
        mMatrix.setRotate(Angle);

        /* 按 mMatrix 的旋转构建新的 Bitmap */
        Bitmap mBitQQ2 = Bitmap.createBitmap(mBitQQ, 0, 0, BitQQwidth,
            BitQQheight, mMatrix, true);

        /* 绘制旋转之后的图像 */
        GameView.drawImage(canvas, mBitQQ2, (320-BitQQwidth)/2, 10);

        mBitQQ2 = null;
    }

    // 按键按下事件
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {

```

```

        //左方向键
        if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT)
        {
            Angle--;
        }
        //右方向键
        else if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT)
        {
            Angle++;
        }
        return true;
    }

    /**
     * 线程处理
     */
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }

    /**
     * 绘制一个 Bitmap
     * @param canvas 画布
     * @param bitmap 图片
     * @param x      屏幕上的 x 坐标
     * @param y      屏幕上的 y 坐标
     */
    public static void drawImage(Canvas canvas, Bitmap bitmap, int x, int y)
    {
        /* 绘制图像 */
        canvas.drawBitmap(bitmap, x, y, null);
    }
}

```

5.2.7 图像缩放

上面讲了如何利用 Matrix 来旋转图像，那么又如何来缩放图像呢？其实和旋转一样，不同的是需要使用 Matrix 的 postScale 方法来设置图像缩放的倍数。下面的示例可以通过上下键来缩小和放大图像。图 5-13 所示是经过缩小的图像，图 5-14 则是经过放大的图像。完整源码请参见本书所

附代码：第 5 章\Examples_05_09。



图 5-13 缩小图像



图 5-14 放大图像

下面我们来看看如何实现旋转和缩放功能，具体如代码清单 5-10 所示。

代码清单 5-10 第 5 章\Examples_05_09\src\com\yarin\android\Examples_05_09\GameView.java

```
public class GameView extends View implements Runnable
{
    /* 声明 Bitmap 对象 */
    Bitmap mBitQQ = null;
    int BitQQwidth = 0;
    int BitQQheight = 0;

    float Scale= 1.0f;

    /* 构建 Matrix 对象 */
    Matrix mMatrix = new Matrix();

    public GameView(Context context)
    {
        super(context);

        /* 装载资源 */
        mBitQQ = ((BitmapDrawable) getResources().getDrawable(R.drawable.qq)).
            getBitmap();

        /* 得到图像的宽度和高度 */
        BitQQwidth = mBitQQ.getWidth();
        BitQQheight = mBitQQ.getHeight();

        /* 开启线程 */
        new Thread(this).start();
    }

    public void onDraw(Canvas canvas)
    {

```

```

    super.onDraw(canvas);

    /* 重置 mMatrix */
    mMatrix.reset();

    /* 设置缩放 */
    mMatrix.postScale(Scale, Scale);

    /* 按 mMatrix 的旋转构建新的 Bitmap */
    Bitmap mBitQQ2 = Bitmap.createBitmap(mBitQQ, 0, 0, BitQQwidth,
    BitQQheight, mMatrix, true);

    /* 绘制旋转之后的图像 */
    GameView.drawImage(canvas, mBitQQ2, (320-BitQQwidth)/2, 10);

    mBitQQ2 = null;
}

// 按键按下事件
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    //左方向键
    if (keyCode == KeyEvent.KEYCODE_DPAD_UP)
    {
        if ( Scale > 0.3 )
        {
            Scale-=0.1;
        }
    }
    //右方向键
    else if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN)
    {
        if ( Scale < 1.5 )
        {
            Scale+=0.1;
        }
    }
    return true;
}

/**
 * 线程处理
 */
public void run()
{
    while (!Thread.currentThread().isInterrupted())

```

```

        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }

    /**
     * 绘制一个 Bitmap
     * @param canvas 画布
     * @param bitmap 图片
     * @param x      屏幕上的 x 坐标
     * @param y      屏幕上的 y 坐标
     */
    public static void drawImage(Canvas canvas, Bitmap bitmap, int x, int y)
    {
        /* 绘制图像 */
        canvas.drawBitmap(bitmap, x, y, null);
    }
}

```

5.2.8 图像像素操作

我们在玩游戏时经常会看到一些图像的特效，比如半透明等效果。要实现这些效果并不难，只需要对图像本身的像素执行操作。Android 中 `Bitmap` 同样提供了操作像素的方法，可以通过 `getPixels` 方法来获得该图像的像素并放到一个数组中，我们处理这个像素数组就可以了，最后通过 `setPixels` 设置这个像素数组到 `Bitmap` 中。

在 Android 中，每一个图像像素通过一个 4 字节整数来展现：最高位字节用作 Alpha 通道，即用来实现透明与不透明控制，255 代表完全不透明，0 则代表完全透明；接下来的一个字节是 Red 红色通道，255 代表完全是红色。依次类推，接下来的两个字节相应地实现绿色和蓝色通道。

下面的示例通过对图像像素的操作来模拟水纹效果，如图 5-15 所示，具体实现如代码清单 5-11 所示。完整源码请参见本书所附代码：第 5 章\Examples_05_10。



图 5-15 水纹效果

代码清单 5-11 第5章\Examples_05_10\src\com\yarin\android\Examples_05_10\GameView.java

```

public class GameView extends View implements Runnable
{
    int BACKWIDTH;

    int BACKHEIGHT;

    short[] buf2;

    short[] buf1;

    int[] Bitmap2;

    int[] Bitmap1;

    public GameView(Context context)
    {
        super(context);

        /* 装载图像 */
        Bitmap image = BitmapFactory.decodeResource(this.getResources(),
            R.drawable.qq);
        BACKWIDTH = image.getWidth();
        BACKHEIGHT = image.getHeight();

        buf2 = new short[BACKWIDTH * BACKHEIGHT];
        buf1 = new short[BACKWIDTH * BACKHEIGHT];
        Bitmap2 = new int[BACKWIDTH * BACKHEIGHT];
        Bitmap1 = new int[BACKWIDTH * BACKHEIGHT];

        /* 加载图像的像素到数组中 */
        image.getPixels(Bitmap1, 0, BACKWIDTH, 0, 0, BACKWIDTH, BACKHEIGHT);

        new Thread(this).start();
    }

    void DropStone(int x, // x 坐标
        int y, // y 坐标
        int stonesize, // 波源半径
        int stoneweight) // 波源能量
    {
        for (int posx = x - stonesize; posx < x + stonesize; posx++)
            for (int posy = y - stonesize; posy < y + stonesize; posy++)
                if ((posx - x) * (posx - x) + (posy - y) * (posy - y) <
                    stonesize * stonesize)
                    buf1[BACKWIDTH * posy + posx] = (short) -stoneweight;
    }

    void RippleSpread()
    {
        for (int i = BACKWIDTH; i < BACKWIDTH * BACKHEIGHT - BACKWIDTH; i++)
        {
            // 波能扩散
            buf2[i] = (short) (((buf1[i - 1] + buf1[i + 1] + buf1[i - BACKWIDTH]
                + buf1[i + BACKWIDTH]) >> 1) - buf2[i]);
        }
    }
}

```



```

        // 波能衰减
        buf2[i] -= buf2[i] >> 5;
    }

    // 交换波能数据缓冲区
    short[] ptmp = buf1;
    buf1 = buf2;
    buf2 = ptmp;
}

/* 渲染水纹效果 */
void render()
{
    int xoff, yoff;
    int k = BACKWIDTH;
    for (int i = 1; i < BACKHEIGHT - 1; i++)
    {
        for (int j = 0; j < BACKWIDTH; j++)
        {
            // 计算偏移量
            xoff = buf1[k - 1] - buf1[k + 1];
            yoff = buf1[k - BACKWIDTH] - buf1[k + BACKWIDTH];

            // 判断坐标是否在窗口范围内
            if ((i + yoff) < 0)
            {
                k++;
                continue;
            }
            if ((i + yoff) > BACKHEIGHT)
            {
                k++;
                continue;
            }
            if ((j + xoff) < 0)
            {
                k++;
                continue;
            }
            if ((j + xoff) > BACKWIDTH)
            {
                k++;
                continue;
            }

            // 计算出偏移像素和原始像素的内存地址偏移量
            int pos1, pos2;
            pos1 = BACKWIDTH * (i + yoff) + (j + xoff);
            pos2 = BACKWIDTH * i + j;
            Bitmap2[pos2++] = Bitmap1[pos1++];
            k++;
        }
    }
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
}

```

```

        /* 绘制经过处理的图片效果 */
        canvas.drawBitmap(Bitmap2, 0, BACKWIDTH, 0, 0, BACKWIDTH, BACKHEIGHT,
            false, null);
    }

    /**
     * 线程处理
     */
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(50);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            RippleSpread();
            render();
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}

```

5.2.9 Shader 类介绍

Android 中提供了 Shader 类专门用来渲染图像以及一些几何图形, Shader 下面包括几个直接子类, 分别是 BitmapShader、ComposeShader、LinearGradient、RadialGradient、SweepGradient。BitmapShader 主要用来渲染图像, LinearGradient 用来进行线性渲染, RadialGradient 用来进行环形渲染, SweepGradient 用来进行梯度渲染, ComposeShader 则是一个混合渲染, 可以和其他几个子类组合起来使用。下面的示例将分别分析它们的使用方法。

如果要将一张图片裁剪成椭圆或圆形等形状展示给用户, 这时需要使用 BitmapShader 类来裁剪, 我们随时可以看到一些渐变的效果, 而在 Android 中可以使用 LinearGradient、SweepGradient、RadialGradient 很轻松地来完成。首先来看看示例的运行效果, 如图 5-16 所示, 每个图形分别使用了不同的渲染。

Shader 类的使用, 都需要先构建 Shader 对象, 然后通过 Paint 的 setShader 方法来设置渲染对象, 然后在绘制时使用这个 Paint 对



图 5-16 Shader 类渲染

象即可。当然，用不同的渲染时需要构建不同的对象，如代码清单 5-12 所示。具体实现参见本书所附代码：第 5 章\Examples_05_11。

代码清单 5-12 第 5 章\Examples_05_11\src\com\yarin\android\Examples_05_11\GameView.java

```
public class GameView extends View implements Runnable
{
    /* 声明 Bitmap 对象 */
    Bitmap mBitQQ = null;
    int BitQQwidth = 0;
    int BitQQheight = 0;

    Paint mPaint = null;

    /* Bitmap 渲染 */
    Shader mBitmapShader = null;

    /* 线性渐变渲染 */
    Shader mLinearGradient = null;

    /* 混合渲染 */
    Shader mComposeShader = null;

    /* 唤醒渐变渲染 */
    Shader mRadialGradient = null;

    /* 梯度渲染 */
    Shader mSweepGradient = null;

    ShapeDrawable mShapeDrawableQQ = null;

    public GameView(Context context)
    {
        super(context);

        /* 装载资源 */
        mBitQQ = ((BitmapDrawable) getResources().getDrawable(R.drawable.qq)).
            getBitmap();

        /* 得到图片的宽度和高度 */
        BitQQwidth = mBitQQ.getWidth();
        BitQQheight = mBitQQ.getHeight();

        /* 创建 BitmapShader 对象 */
        mBitmapShader = new BitmapShader(mBitQQ, Shader.TileMode.REPEAT,
            Shader.TileMode.MIRROR);

        /* 创建 LinearGradient 并设置渐变的颜色数组 */
        mLinearGradient = new LinearGradient(0,0,100,100,
            new int[] {Color.RED,Color.GREEN,Color.BLUE,Color.WHITE},null,
            Shader.TileMode.REPEAT);

        /* 这里笔者理解为“混合渲染”——大家可以有自己的理解，能明白这个意思就好 */
        mComposeShader = new ComposeShader(mBitmapShader,mLinearGradient,
            PorterDuff.Mode.DARKEN);

        /* 构建 RadialGradient 对象，设置半径的属性 */
    }
}
```

```

//这里使用了 BitmapShader 和 LinearGradient 进行混合
//当然也可以使用其他的组合
//混合渲染的模式很多, 可以根据自己的需要来选择
mRadialGradient = new RadialGradient(50, 200, 50, new int[] {Color.GREEN,
Color.RED, Color.BLUE, Color.WHITE}, null, Shader.TileMode.REPEAT);
/* 构建 SweepGradient 对象 */
mSweepGradient = new SweepGradient(30, 30, new int[] {Color.GREEN, Color.RED,
Color.BLUE, Color.WHITE}, null);

mPaint = new Paint();

/* 开启线程 */
new Thread(this).start();
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    //将图片裁剪为椭圆形
    /* 构建 ShapeDrawable 对象并定义形状为椭圆 */
    mShapeDrawableQQ = new ShapeDrawable(new OvalShape());

    /* 设置要绘制的椭圆形的东西为 ShapeDrawable 图片 */
    mShapeDrawableQQ.getPaint().setShader(mBitmapShader);

    /* 设置显示区域 */
    mShapeDrawableQQ.setBounds(0, 0, BitQQwidth, BitQQheight);

    /* 绘制 ShapeDrawableQQ */
    mShapeDrawableQQ.draw(canvas);

    //绘制渐变的矩形
    mPaint.setShader(mLinearGradient);
    canvas.drawRect(BitQQwidth, 0, 320, 156, mPaint);

    //显示混合渲染效果
    mPaint.setShader(mComposeShader);
    canvas.drawRect(0, 300, BitQQwidth, 300+BitQQheight, mPaint);

    //绘制环形渐变
    mPaint.setShader(mRadialGradient);
    canvas.drawCircle(50, 200, 50, mPaint);

    //绘制梯度渐变
    mPaint.setShader(mSweepGradient);
    canvas.drawRect(150, 160, 300, 300, mPaint);
}

/**
 * 线程处理
 */
public void run()
{

```

```

        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }
}

```

5.2.10 双缓冲技术

双缓冲技术是游戏开发中的一个重要技术。主要原理：当一个动画争先显示时，程序又在改变它，前画面还没有显示完，程序又请求重新绘制，这样屏幕就会不停闪烁。为了避免闪烁，可以使用双缓冲技术，将要处理的图片都在内存中处理好之后，再将其显示到屏幕上。这样显示出来的总是完整的图像，不会出现闪烁现象。

通过前面的章节我们可以看出，Android 中的 `SurfaceView` 类其实就是一个双缓冲机制。因此，开发游戏时可能会比较多地使用 `SurfaceView` 类来完成，这样效率更高，而且 `SurfaceView` 类的功能也更加完善。下面将主要介绍用 `View` 类实现双缓冲技术。先来看看通过双缓冲技术绘制的图像效果，如图 5-17 所示。

双缓冲的核心技术就是先通过 `setBitmap` 方法将要绘制的所有图形绘制到一个 `Bitmap` 上，然后再来调用 `drawBitmap` 方法绘制出这个 `Bitmap`，显示在屏幕上。具体实现如代码清单 5-13 所示。完整源码请参见本书所附代码：第 5 章\Examples_05_12。



图 5-17 双缓冲技术

代码清单 5-13 第 5 章\Examples_05_12\src\com\yarin\android\Examples_05_12\GameView.java

```

public class GameView extends View implements Runnable
{
    /* 声明 Bitmap 对象 */
    Bitmap mBitQQ = null;

    Paint mPaint = null;

    /* 创建一个缓冲区 */
    Bitmap mSCBitmap = null;

    /* 创建 Canvas 对象 */
    Canvas mCanvas = null;

    public GameView(Context context)
    {
        super(context);
    }
}

```

```

    /* 装载资源 */
    mBitQQ = ((BitmapDrawable) getResources().getDrawable(R.drawable.qq)).
    getBitmap();

    /* 创建屏幕大小的缓冲区 */
    mSCBitmap=Bitmap.createBitmap(320, 480, Config.ARGB_8888);

    /* 创建 Canvas */
    mCanvas = new Canvas();

    /* 设置将内容绘制在 mSCBitmap 上 */
    mCanvas.setBitmap(mSCBitmap);

    mPaint = new Paint();

    /* 将 mBitQQ 绘制到 mSCBitmap 上 */
    mCanvas.drawBitmap(mBitQQ, 0, 0, mPaint);

    /* 开启线程 */
    new Thread(this).start();
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    /* 将 mSCBitmap 显示到屏幕上 */
    canvas.drawBitmap(mSCBitmap, 0, 0, mPaint);
}

// 触笔事件
public boolean onTouchEvent(MotionEvent event)
{
    return true;
}

/**
 * 线程处理
 */
public void run()
{
    while (!Thread.currentThread().isInterrupted())
    {
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
            Thread.currentThread().interrupt();
        }
        //使用 postInvalidate 可以直接在线程中更新界面
        postInvalidate();
    }
}
}

```

5.2.11 全屏显示

我们前面所写的程序在运行时屏幕顶部都带有标题栏和系统信息栏，但是在开发游戏时，可能会因为界面美观等因素不需要显示这些多余的界面，而将更多的区域留给游戏界面来控制，那么就需要把屏幕的现实模式设置为全屏模式。通过 `requestWindowFeature` 方法可以设置标题栏是否显示，通过 `setFlags` 方法可以设置全屏模式，具体运行效果如图 5-18 所示。



图 5-18 全屏模式

图 5-23 中取消了状态栏和应用程序标题栏的显示，实现了全屏显示效果，程序默认运行都是竖屏显示的，但是我们可以通过 `setRequestedOrientation` 方法将显示模式设置为横屏显示，如代码清单 5-14 所示。完整源码请参见本书所附代码：第 5 章\Examples_05_13。

代码清单 5-14 第 5 章\Examples_05_13\src\com\yarin\android\Examples_05_13\Activity01.java

```
public class Activity01 extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        /* 设置为无标题栏 */
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        /* 设置为全屏模式 */
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);

        /* 设置为横屏 */
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        setContentView(R.layout.main);
    }
}
```

5.2.12 获得屏幕属性

市面上的手机样式非常多，而且屏幕大小不一，大家肯定都希望自己开发的应用程序能自动适应各种屏幕大小的手机。这就需要在开发时尽量不要把一些坐标都设置为定值，而是通过计算得出。这样应用程序不管安装到什么手机上，都不需要做很大的改动就能正常显示。要计算坐标等变量的值，肯定需要一个值作为参考，这里最好的参考值就是屏幕的宽度和高度了。下面我们使用

程序来获得屏幕的宽度和高度。

Android 中的 `DisplayMetrics` 定义了屏幕的一些属性，可以通过 `getMetrics` 方法得到当前屏幕的 `DisplayMetrics` 属性，从而取得屏幕的宽和高。下面我们将获得的屏幕的宽和高显示在屏幕上，运行效果如图 5-19 所示。具体实现如代码清单 5-15 所示。完整源码请参见本书所附代码：第 5 章\Examples_05_14。



图 5-19 获得屏幕的宽和高

代码清单 5-15 第 5 章\Examples_05_14\src\com\lyarin\android\Examples_05_14\Activity01.java

```
public class Activity01 extends Activity
{
    TextView mTextView = null;

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

        /* 定义 DisplayMetrics 对象 */
        DisplayMetrics dm = new DisplayMetrics();

        /* 取得窗口属性 */
        getWindowManager().getDefaultDisplay().getMetrics(dm);

        /* 窗口的宽度 */
        int screenWidth = dm.widthPixels;

        /* 窗口的高度 */
        int screenHeight = dm.heightPixels;

        mTextView = (TextView) findViewById(R.id.TextView01);

        mTextView.setText("屏幕宽度: " + screenWidth + "\n屏幕高度: " +
            screenHeight);
    }
}
```

扩展学习

玩过 G1 朋友都知道它内置了加速感应器，可以让应用程序自动适应屏幕的模式，比如当前是竖屏模式，当我们将手机横放时，应用程序会自动变为横向模式。我们可以很简单地实现这个效果，在 Eclipse 中双击 `AndroidManifest.xml` 文件，选择 `Application` 选项卡，选中 `Activity` 类，然后在右边找到“Screen orientation”选项，选择“sensor”，最后保存即可，这时在真机上就能感觉到效果了。

当然也可以直接在 `AndroidManifest.xml` 文件中修改“`<activity android:name=".Activity01" android:label="@string/app_name" android:screenOrientation="sensor">`”，可以实现相同的效果，代码如下：

```

<activity android:name=".Activity01"
    android:label="@string/app_name"
    android:screenOrientation="sensor">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

```

5.3 动画实现

Android 平台提供了两类动画，一类是 Tween 动画，即通过对场景里的对象不断进行图像变换（平移、缩放、旋转）来产生动画效果；第二类是 Frame 动画，即顺序播放事先做好的图像，和电影类似。本节我们将分别学习这两种动画的实现原理，最后笔者会结合 Java ME 开发经验讲解 GIF 动画的实现原理。

5.3.1 Tween 动画

首先来分析 Tween 动画，Tween 动画通过对 View 的内容完成一系列的图形变换（包括平移、缩放、旋转、改变透明度）来实现动画效果。它主要包括以下四种动画效果：

- ❑ Alpha: 渐变透明度动画效果。
- ❑ Scale: 渐变尺寸伸缩动画效果。
- ❑ Translate: 画面转换位置移动动画效果。
- ❑ Rotate: 画面转移旋转动画效果。

具体来讲，Tween 动画是通过预先定义一组指令，这些指令指定了图形变换的类型、触发时间、持续时间。程序沿着时间线执行这些指令就可以实现动画效果。因此我们首先需要定义 Animation 动画对象，然后设置该动画的一些属性，最后通过 `startAnimation` 方法来开始动画。各个动画的属性设置如下：

`AlphaAnimation(float fromAlpha, float toAlpha)`

功能：构建一个渐变透明度动画

参数：fromAlpha 为动画起始时透明度；toAlpha 为动画结束时透明度（0.0 表示完全透明，1.0 表示完全不透明）。

`ScaleAnimation(float fromX, float toX, float fromY, float toY, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue)`

功能：构建一个渐变尺寸伸缩动画。

参数：fromX、toX 分别是起始和结束时 x 坐标上的伸缩尺寸。fromY、toY 分别是起始和结束时 y 坐标上的伸缩尺寸。pivotXValue、pivotYType 分别为 x、y 的伸缩模式。pivotXValue、pivotYValue 分别为伸缩动画相对于 x、y 的坐标开始位置。

`TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float toYDelta)`

功能：构建一个画面转换位置移动动画。

参数：fromXDelta、fromYDelta 分别为起始坐标；toXDelta、toYDelta 分别为结束坐标。

```
RotateAnimation(float fromDegrees, float toDegrees, int pivotXType, float
pivotXValue, int pivotYType, float pivotYValue)
```

功能：构建一个旋转画面的动画。

参数：fromDegrees 为开始的角度；toDegrees 为结束的角度。pivotXValue、pivotYType 分别为 x、y 的伸缩模式。pivotXValue、pivotYValue 分别为伸缩动画相对于 x、y 的坐标开始位置。

```
setDuration(long durationMillis)
```

功能：设置动画显示的时间。

参数：durationMillis 为动画显示时间的长短，以毫秒为单位。

```
startAnimation(Animation animation)
```

功能：开始播放动画。

参数：animation 为要播放的动画。

下面我们通过具体代码来看这些动画是如何实现的，AlphaAnimation 动画如图 5-20 所示，ScaleAnimation 动画如图 5-21 所示，TranslateAnimation 动画如图 5-22 所示，RotateAnimation 如图 5-23 所示，具体代码如代码清单 5-16 所示。分别通过操作上、下、左、右四个方向键来显示动画效果。完整源码请参见本书所附代码：第 5 章\Examples_05_15。

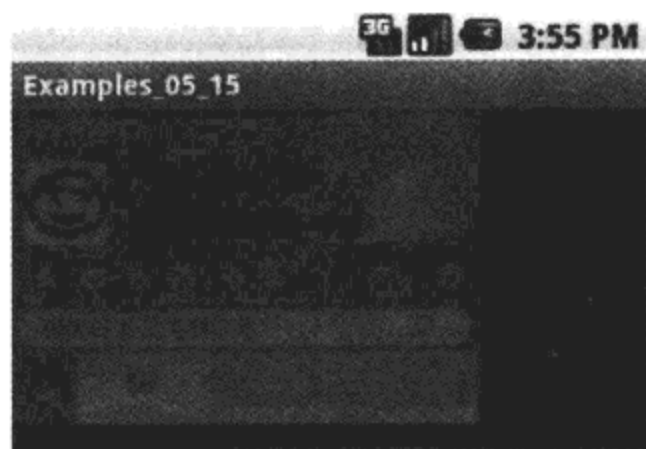


图 5-20 渐变透明度动画



图 5-21 渐变尺寸伸缩动画



图 5-22 画面位置移动动画

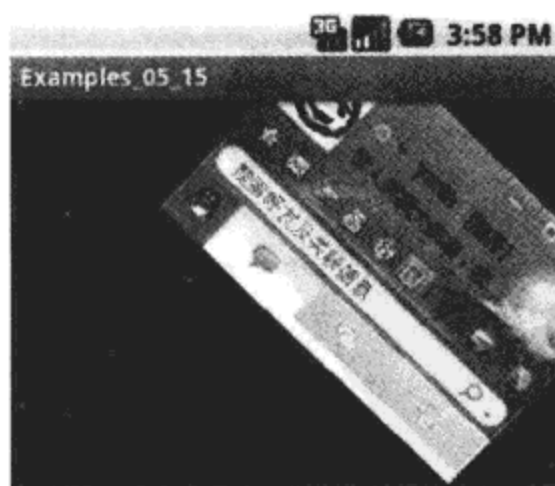


图 5-23 画面旋转动画

代码清单 5-16 \第 5 章\Examples_05_15\src\com\yarin\android\Examples_05_15 \GameView.java

```
public class GameView extends View
{
```

```

/* 定义 Alpha 动画 */
private Animation mAnimationAlpha = null;

/* 定义 Scale 动画 */
private Animation mAnimationScale = null;

/* 定义 Translate 动画 */
private Animation mAnimationTranslate = null;

/* 定义 Rotate 动画 */
private Animation mAnimationRotate = null;

/* 定义 Bitmap 对象 */
Bitmap mBitQQ = null;

public GameView(Context context)
{
    super(context);

    /* 装载资源 */
    mBitQQ = ((BitmapDrawable) getResources().getDrawable(R.drawable.qq)).
        getBitmap();
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    /* 绘制图片 */
    canvas.drawBitmap(mBitQQ, 0, 0, null);
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            /* 创建 Alpha 动画 */
            mAnimationAlpha = new AlphaAnimation(0.1f, 1.0f);
            /* 设置动画的时间 */
            mAnimationAlpha.setDuration(3000);
            /* 开始播放动画 */
            this.startAnimation(mAnimationAlpha);
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            /* 创建 Scale 动画 */
            mAnimationScale = new ScaleAnimation(0.0f, 1.0f, 0.0f, 1.0f,
                Animation.RELATIVE_TO_SELF, 0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
            /* 设置动画的时间 */
            mAnimationScale.setDuration(500);
            /* 开始播放动画 */
            this.startAnimation(mAnimationScale);
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            /* 创建 Translate 动画 */
            mAnimationTranslate = new TranslateAnimation(10, 100, 10, 100);
            /* 设置动画的时间 */

```

```

        mAnimationTranslate.setDuration(1000);
        /* 开始播放动画 */
        this.startAnimation(mAnimationTranslate);
        break;
    case KeyEvent.KEYCODE_DPAD_RIGHT:
        /* 创建 Rotate 动画 */
        mAnimationRotate=new RotateAnimation(0.0f, +360.0f, Animation.
            RELATIVE_TO_SELF,0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
        /* 设置动画的时间 */
        mAnimationRotate.setDuration(1000);
        /* 开始播放动画 */
        this.startAnimation(mAnimationRotate);
        break;
    }
    return true;
}
}
}

```

扩展学习

上面我们分别实现了 4 种不同的 Tween 动画，但都是通过编写 Java 代码实现的。其实，在 Android 中使用 XML 布局文件实现动画更加简单。下面我们分别将这 4 种动画改用 XML 布局文件来实现。

<alpha>动画布局中，fromAlpha 表示起始透明度，toAlpha 表示结束透明度，duration 表示动画的时间长短。具体实现如代码清单 5-17 所示。

代码清单 5-17 第 5 章\Examples_05_16\res\anim\alpha_animation.xml

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <alpha
        android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="2000"
    />
</set>

```

在<scale>动画布局中，fromXScale 为动画起始时 x 坐标上的伸缩尺寸，toXScale 为动画结束时 x 坐标上的伸缩尺寸，fromYScale 为动画起始时 y 坐标上的伸缩尺寸，toYScale 为动画结束时 y 坐标上的伸缩尺寸；pivotX 为动画相对于物件的 x 坐标的开始位置，pivotY 为动画相对于物件的 y 坐标的开始位置；duration 表示动画的时间长短。具体代码如代码清单 5-18 所示。

代码清单 5-18 第 5 章\Examples_05_16\res\anim\scale_animation.xml

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="0.0"
        android:toXScale="1.0"
        android:fromYScale="0.0"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="500" />
</set>

```

在<translate>动画布局中, fromXDelta 为动画起始时 x 坐标上的位置, toXDelta 为动画结束时 x 坐标上的位置, fromYDelta 为动画起始时 y 坐标上的位置, toYDelta 为动画结束时 y 坐标上的位置; duration 表示动画的时间长短。具体实现如代码清单 5-19 所示。

代码清单 5-19 第 5 章\Examples_05_16\res\anim\translate_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<translate
  android:fromXDelta="10"
  android:toXDelta="100"
  android:fromYDelta="10"
  android:toYDelta="100"
  android:duration="1000"
/>
</set>
```

在<rotate>动画布局中, interpolator 指定了一个动画插入器, fromDegrees 为动画起始时物件的角度, toDegrees 为动画结束时物件旋转的角度可以大于 360 度, pivotX 为动画相对于物件的 x 坐标的开始位置, pivotY 为动画相对于物件的 y 坐标的开始位置, duration 表示动画的时间长短, 具体实现如代码清单 5-20 所示。

代码清单 5-20 第 5 章\Examples_05_16\res\anim\rotate_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<rotate
  android:interpolator="@android:anim/accelerate_decelerate_interpolator"
  android:fromDegrees="0"
  android:toDegrees="+360"
  android:pivotX="50%"
  android:pivotY="50%"
  android:duration="1000" />
</set>
```

现在我们完成了几种动画的 XML 布局, 下面只需要在程序中装载这些布局, 然后开始播放动画即可, 具体实现如代码清单 5-21 所示。完整源码请参见本书所附代码: 第 5 章\Examples_05_16。

代码清单 5-21 第 5 章\Examples_05_16\src\com\yarin\android\Examples_05_16\GameView.java

```
public class GameView extends View
{
    /* 定义 Alpha 动画 */
    private Animation mAnimationAlpha = null;
    /* 定义 Scale 动画 */
    private Animation mAnimationScale = null;
    /* 定义 Translate 动画 */
    private Animation mAnimationTranslate = null;
    /* 定义 Rotate 动画 */
    private Animation mAnimationRotate = null;
    /* 定义 Bitmap 对象 */
    Bitmap mBitQQ = null;

    Context mContext = null;
    public GameView(Context context)
    {
        super(context);
```

```

mContext = context;

/* 装载资源 */
mBitQQ = ((BitmapDrawable) getResources().getDrawable(R.drawable.qq)).
getBitmap();
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    /* 绘制图片 */
    canvas.drawBitmap(mBitQQ, 0, 0, null);
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            /* 装载动画布局 */
            mAnimationAlpha = AnimationUtils.loadAnimation(mContext, R.anim.
            alpha_animation);
            /* 开始播放动画 */
            this.startAnimation(mAnimationAlpha);
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            /* 装载动画布局 */
            mAnimationScale = AnimationUtils.loadAnimation(mContext, R.anim.
            scale_animation);
            /* 开始播放动画 */
            this.startAnimation(mAnimationScale);
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            /* 装载动画布局 */
            mAnimationTranslate = AnimationUtils.loadAnimation(mContext,
            R.anim.translate_animation);
            /* 开始播放动画 */
            this.startAnimation(mAnimationTranslate);
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            /* 装载动画布局 */
            mAnimationRotate = AnimationUtils.loadAnimation(mContext,
            R.anim.rotate_animation);
            /* 开始播放动画 */
            this.startAnimation(mAnimationRotate);
            break;
    }
    return true;
}
}

```

5.3.2 Frame 动画

大家见得最多的应该就是 Frame 动画了，Android 中当然也少不了它。它的使用更加简单，只

需要创建一个 `AnimationDrawable` 对象来表示 `Frame` 动画，然后通过 `addFrame` 方法把每一帧要显示的内容添加进去，最后通过 `start` 方法就可以播放这个动画了，同时还可以通过 `setOneShot` 方法来设置该动画是否重复播放。下面我们就通过 `Frame` 动画来模拟日全食的效果。先看看程序运行的效果，如图 5-24 和图 5-25 所示。



图 5-24 Frame 动画效果一



图 5-25 Frame 动画效果二

按上方向键开始欣赏日全食效果。下面我们来看看这是如何实现的，具体实现如代码清单 5-22 所示。完整代码请参见本书所附代码：第 5 章\Examples_05_17。

代码清单 5-22 第 5 章\Examples_05_17\src\com\yarin\android\Examples_05_17\GameView.java

```
public class GameView extends View
{
    /* 定义 AnimationDrawable 动画 */
    private AnimationDrawable frameAnimation = null;
    Context mContext = null;

    /* 定义一个 Drawable 对象 */
    Drawable mBitAnimation = null;
    public GameView(Context context)
    {
        super(context);

        mContext = context;

        /* 实例化 AnimationDrawable 对象 */
        frameAnimation = new AnimationDrawable();

        /* 装载资源 */
        //这里用一个循环装载所有名字类似的资源
        //如 "a1.....15.png" 的图片
        //这个方法用处非常大
        for (int i = 1; i <= 15; i++)
        {
            int id = getResources().getIdentifier("a" + i, "drawable", mContext.
```

```

        getPackageName());
        mBitAnimation = getResources().getDrawable(id);
        /* 为动画添加一帧 */
        //参数 mBitAnimation 是该帧的图片
        //参数 500 是该帧显示的时间,按毫秒计算
        frameAnimation.addFrame(mBitAnimation, 500);
    }

    /* 设置播放模式是否循环, false 表示循环, true 表示不循环 */
    frameAnimation.setOneShot( false );

    /* 设置本类将要显示这个动画 */
    this.setBackgroundDrawable(frameAnimation);
}

public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch ( keyCode )
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            /* 开始播放动画 */
            frameAnimation.start();
            break;
    }
    return true;
}
}

```

扩展学习

当然,如果使用 XML 布局文件来实现 Frame 动画,基本上就不用写多少代码了,只需要在 XML 文件中通过 oneshot 设置动画是否重复播放,然后将所有帧都列出来即可,如代码清单 5-23 所示。最后在程序中通过 setBackgroundResource 方法加载这个 XML 动画布局文件,通过用 getBackground 方法得到动画,通过用 setBackgroundDrawable 方法设置要显示的动画,通过用 start 开始播放动画即可欣赏和上面一样的日全食效果了,具体代码如代码清单 5-24 所示。完整源码参见本书所附代码:第 5 章\Examples_05_18。

代码清单 5-23 第 5 章\Examples_05_18\res\anim\frameanimation.xml

```

<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/a1" android:duration="500" />
    <item android:drawable="@drawable/a2" android:duration="500" />
    <item android:drawable="@drawable/a3" android:duration="500" />
    <item android:drawable="@drawable/a4" android:duration="500" />
    <item android:drawable="@drawable/a5" android:duration="500" />
    <item android:drawable="@drawable/a6" android:duration="500" />
    <item android:drawable="@drawable/a7" android:duration="500" />
    <item android:drawable="@drawable/a8" android:duration="500" />
    <item android:drawable="@drawable/a9" android:duration="500" />

```

```

<item android:drawable="@drawable/a10" android:duration="500" />
<item android:drawable="@drawable/a11" android:duration="500" />
<item android:drawable="@drawable/a12" android:duration="500" />
<item android:drawable="@drawable/a13" android:duration="500" />
<item android:drawable="@drawable/a14" android:duration="500" />
<item android:drawable="@drawable/a15" android:duration="500" />
</animation-list>

```

代码清单 5-24 第 5 章\Examples_05_18\src\com\yarin\android\Examples_05_18\GameView.java

```
package com.yarin.android.Examples_05_18;
```

```

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.drawable.AnimationDrawable;
import android.view.KeyEvent;
import android.view.View;
import android.widget.ImageView;

```

```
public class GameView extends View
```

```

{
    /* 定义 AnimationDrawable 动画对象 */
    private AnimationDrawable frameAnimation = null;
    Context mContext = null;
    public GameView(Context context)
    {
        super(context);

        mContext = context;

        /* 定义一个 ImageView 用来显示动画 */
        ImageView img = new ImageView(mContext);

        /* 装载动画布局文件 */
        img.setBackgroundResource(R.anim.frameanimation);

        /* 构建动画 */
        frameAnimation = (AnimationDrawable) img.getBackground();

        /* 设置是否循环 */
        frameAnimation.setOneShot(false);

        /* 设置该类显示的动画 */
        this.setBackgroundDrawable(frameAnimation);
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
    }
}

```

```

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch ( keyCode )
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            /* 开始播放动画 */
            frameAnimation.start();
            break;
    }
    return true;
}
}

```

5.3.3 GIF 动画播放

GIF 动画大家都很熟悉了吧，GIF 格式的图片本身就可以做成动画效果，大家是否在想，要是能直接播放 GIF 动画，那么做游戏界面不是很轻松了吗？当然，程序员可以不用自己去做动画，只需要直接播放美工做好的 GIF 动画就可以了。不光是你在想，我也在想呢，全世界的程序员都在想，目前已经有很多程序员已经实现了这一功能，也出现了一些开源工程。笔者这里把自己在做 Java ME 开发时使用的播放 GIF 的类移植到了 Android 平台上，下面我们先来看看播放 GIF 动画的效果，如图 5-26 所示。



图 5-26 播放 GIF 动画

图 5-26 中是播放的一张 GIF 的动画，要想播放 GIF 动画，首先需要对 GIF 图像进行解码，然后将 GIF 中的每一帧分别提取出来保存到一个容器中，然后根据需要连续绘制每一帧，这样就可以轻松地实现 GIF 动画的播放。这个逻辑过程十分简单，首先需要创建一个 GIF 图像解码类（请参见本书所附代码：第 5 章\Examples_05_19\src\com\yarin\android\Examples_05_19\ GifDecoder.java）。然后需要一个 GIF 帧管理器来管理 GIF 的每一帧的图片（请参见本书所附代码：请参见本书所附代码：第 5 章\Examples_05_19\src\com\yarin\android\Examples_05_19\ GifFrame.java）。最后，使用这两个类播放一张 GIF 动画，如代码清单 5-25 所示。完整源码请参见本书所附代码：第 5 章\Examples_05_19。

最后，我们还需要在 GameView 中解析 GIF 动画并设置显示，如代码清单 5-25 所示。

代码清单 5-25 第 5 章\Examples_05_19\src\com\yarin\android\Examples_05_19、GameView.java

```

public class GameView extends View implements Runnable
{
    Context mContext = null;

    /* 声明 GifFrame 对象 */
    GifFrame mGifFrame = null;

    public GameView(Context context)
    {
        super(context);
    }
}

```

```

        mContext = context;
        /* 解析 GIF 动画 */
        mGifFrame=GifFrame.CreateGifImage(fileConnect(this.getResources().
        openRawResource(R.drawable.gif1)));
        /* 开启线程 */
        new Thread(this).start();
    }

    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        /* 下一帧 */
        mGifFrame.nextFrame();
        /* 得到当前帧的图片 */
        Bitmap b=mGifFrame.getImage();

        /* 绘制当前帧的图片 */
        if(b!=null)
            canvas.drawBitmap(b,10,10,null);
    }

    /**
     * 线程处理
     */
    public void run()
    {
        while (!Thread.currentThread().isInterrupted())
        {
            try
            {
                Thread.sleep(100);
            }
            catch (InterruptedException e)
            {
                Thread.currentThread().interrupt();
            }
            //使用 postInvalidate 可以直接在线程中更新界面
            postInvalidate();
        }
    }

    /* 读取文件 */
    public byte[] fileConnect(InputStream is)
    {
        try
        {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            int ch = 0;
            while( (ch = is.read()) != -1)
            {
                baos.write(ch);
            }
            byte[] datas = baos.toByteArray();
            baos.close();
            baos = null;
            is.close();
        }
    }

```

```
        is = null;
        return datas;
    }
    catch(Exception e)
    {
        return null;
    }
}
```

5.4 小结

本章首先介绍了 Android 中的游戏开发框架，然后介绍了图形图像的绘制和处理方法，最后着重介绍了 Android 中的两种动画的实现原理，以及在 Android 平台中如何播放 GIF 动画。这些知识基本能满足游戏开发的需求，当然，游戏开发还需要进行大量的图形图像处理，以及动画的实现。每一节都提供了一个实例供大家参考，希望大家在理解理论内容的同时，学会实际操作。对于复杂的游戏来说，还需要更高级的处理以及优化，后面的章节会专门讨论这方面的话题，希望大家保持高度的热情继续后面的学习！



Android 数据存储

数据存储是应用程序最基本的问题，任何企业系统、应用软件都必须解决这一问题，数据存储必须以某种方式保存，不能丢失，并且能够有效、简便地使用和更新这些数据。本章将详细介绍 Android 中数据存储方式以及数据共享。

6.1 Android 数据存储初探

在 Android 中一共提供了 4 种数据存储方式，但是由于存储的这些数据都是其应用程序私有的，所以如果需要在其他应用程序中使用这些数据，就要使用 Android 提供的 Content Providers（数据共享）。Android 中 4 种数据存储方式分别介绍如下。

- ❑ **Shared Preferences:** 用来存储“key-value paires”格式的数据，它是一个轻量级的键值存储机制，只可以存储基本数据类型。
- ❑ **Files:** 它通过 `FileInputStream` 和 `FileOutputStream` 对文件进行操作。但是在 Android 中，文件是一个应用程序私有的，一个应用程序无法读写其他应用程序的文件。
- ❑ **SQLite:** Android 提供的一个标准的数据库，支持 SQL 语句。
- ❑ **Network:** 通过网络来存储和获得数据。

Shared Preferences 主要是针对系统配置信息的保存，比如给程序界面设置了音效，想在下一次启动时还能够保留上次设置的音效。由于 Android 系统的界面是采用 Activity 栈的形式，所以在系统资源不足时会收回一些界面，因此，有些操作需要在不活动时保留下来，以便等再次激活时能够显示出来。

Files 就是把需要保存的东西通过文件的形式记录下来，当需要这些数据时，通过读取这个文件来获得这些数据即可。因为 Android 采用了 Linux 核心，所以在 Android 系统中，文件也是 Linux 的形式。

SQLite 是一个开源的关系型数据库，与普通关系型数据库一样，也具有 ACID 的特性。它可以用来存储大量的数据，并且能够很容易地对数据进行使用、更新、维护等操作。但是操作规范肯定比前两种复杂。

Network 用于将数据存储于网络，还需要使用 `java.net.*` 和 `android.net.*` 这些类。在后面的章节中，我们会通过实例讲述具体操作。

我们在第 3 章解析应用程序时，就简单地分析过 Content Providers（数据共享），因为在 Android 中，很多数据不只是供一个应用程序使用，可能要被多个应用程序共同使用。由于 Shared Preferences 存储的是应用程序系统配置相关的数据，所以通过 Shared Preferences 存储的数据只能供

本应用程序使用，要想让数据共享，就需要使用 Files、SQLite 以及网络存储等方式来存储数据。

了解了这些数据存储方式之后，就可以根据应用程序的需要来选择一种或几种最佳的数据存储方式了。

6.2 数据存储之 Shared Preferences

Shared Preferences 类似于我们常用的 ini 文件，用来保存应用程序的一些属性设置，在 Android 平台常用于存储较简单的参数设置。例如，可以通过它保存上一次用户所做的修改或者自定义参数设定，当再次启动程序后依然保持原有的设置。通过 `getPreferences()` 方法来获得 Preferences 对象，通过 “`SharedPreferences.Editor editor = uiState.edit();`” 取得编辑对象，然后通过 “`editor.put…()`” 方法添加数据，最后通过 `commit()` 方法保存这些数据，如果不需要与其他模块共享数据，可以使用 `Activity.getPreferences()` 方法保持数据私有。需要着重强调的一点是，我们无法直接在多个程序间共享 Preferences 数据。下面我们通过一个例子来保存应用程序当前是否播放音效的状态。首先运行应用程序如图 6-1 所示，当前音效状态处于关闭状态时，我们按“上”方向键来开启音效，然后退出程序，我们再次启动应用程序，现在音效的状态就处于开的状态，如图 6-2 所示。



图 6-1 音效状态“关”



图 6-2 音效状态“开”

在这个应用程序中，我们在退出应用程序时把音效的状态保存到 Preferences 中，因此在启动时就读取出了上次保存的数据，所以程序启动就开始播放音乐。下面我们来看看具体实现代码，如代码清单 6-1 所示（具体实现参见本书所附代码：第 6 章\Examples_06_01）。

代码清单 6-1 第 6 章\Examples_06_01\src\com\lyarin\android\Examples_06_01\Activity01.java

```
public class Activity01 extends Activity
{
    private MIDIPlayer mMIDIPlayer = null;
    private boolean mbMusic = false;
    private TextView mTextView = null;

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

        mTextView = (TextView) this.findViewById(R.id.TextView01);

        mMIDIPlayer = new MIDIPlayer(this);

        /* 装载数据 */
        // 取得活动的 Preferences 对象
        SharedPreferences settings = getPreferences(Activity.MODE_PRIVATE);
        // 取得值
```

```

        mbMusic = settings.getBoolean("bmusic", false);

        if (mbMusic)
        {
            mTextView.setText("当前音乐状态: 开");
            mbMusic = true;
            mMIDIPlayer.PlayMusic();
        }
        else
        {
            mTextView.setText("当前音乐状态: 关");
        }
    }

    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_UP:
                mTextView.setText("当前音乐状态: 开");
                mbMusic = true;
                mMIDIPlayer.PlayMusic();
                break;
            case KeyEvent.KEYCODE_DPAD_DOWN:
                mTextView.setText("当前音乐状态: 关");
                mbMusic = false;
                mMIDIPlayer.FreeMusic();
                break;
        }
        return true;
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_BACK)
        {
            /* 这里我们在退出应用程序时保存数据 */
            // 取得活动的 preferences 对象
            SharedPreferences uiState = getPreferences(0);
            // 取得编辑对象
            SharedPreferences.Editor editor = uiState.edit();
            // 添加值
            editor.putBoolean("bmusic", mbMusic);

            // 提交保存
            editor.commit();
            if (mbMusic)
            {
                mMIDIPlayer.FreeMusic();
            }
            this.finish();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}

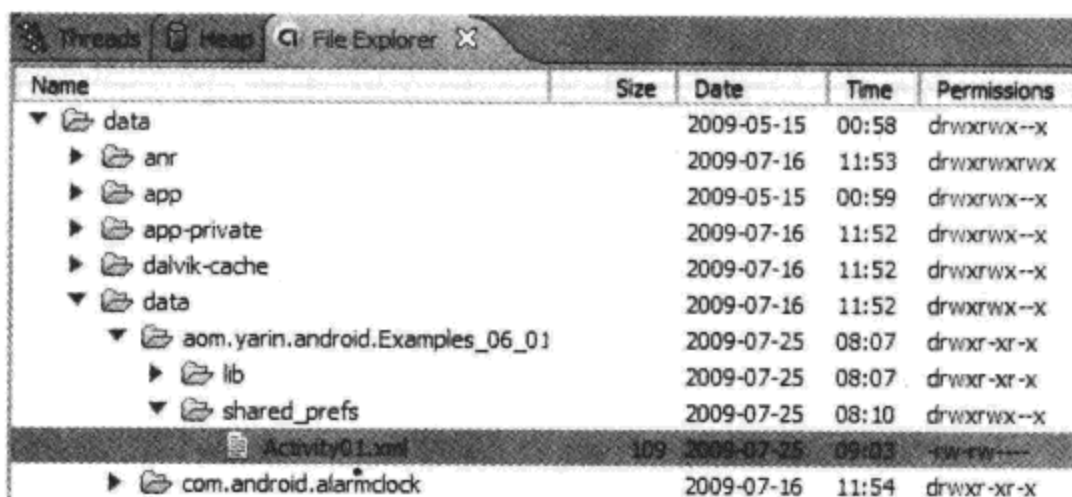
```

扩展学习

现在我们已经实现通过 Preferences 来存取数据了, 那么这些数据究竟被存放到什么地方了呢? 其实每安装一个应用程序时, 在 /data/data 目录下都会产生一个文件夹, 如果应用程序中使用了 Preferences, 那么便会在该文件夹下产生一个 shared_prefs 文件夹, 其中就是我们保存的数据。请查看步骤:

- (1) 启动模拟器, 启动 Eclipse。
- (2) 在 Eclipse 中切换到 DDMS 视图, 选择 File Explorer 标签。
- (3) 找到 /data/data 目录中对应的项目文件夹下的 shared_prefs 文件夹。

例如本节的 Examples_06_01 项目中用 Preferences 来存取的数据保存在 Activity01.xml 文件中, 如图 6-3 所示。



Name	Size	Date	Time	Permissions
data		2009-05-15	00:58	drwxrwx--x
anr		2009-07-16	11:53	drwxrwxrwx
app		2009-05-15	00:59	drwxrwx--x
app-private		2009-07-16	11:52	drwxrwx--x
dalvik-cache		2009-07-16	11:52	drwxrwx--x
data		2009-07-16	11:52	drwxrwx--x
aom.yarin.android.Examples_06_01		2009-07-25	08:07	drwxr-xr-x
lib		2009-07-25	08:07	drwxr-xr-x
shared_prefs		2009-07-25	08:10	drwxrwx--x
Activity01.xml	109	2009-07-25	09:03	-rw-rw----
com.android.alarmclock		2009-07-16	11:54	drwxr-xr-x

图 6-3 Preferences 数据存储目录

6.3 数据存储之 Files

Android 中可以在设备本身的存储设备或者外接的存储设备中创建用于保存数据的文件。同样, 在默认的状态下, 文件是不能在不同的程序间共享的。用文件来存储数据可以通过 openFileOutput 方法打开一个文件 (如果这个文件不存在就自动创建这个文件), 通过 load 方法来获取文件中的数据, 通过 deleteFile 方法可以删除一个指定的文件。

现在我们就可以用文件的方式来实现上一节中保存音乐状态的例子。首先, 在退出应用程序之前, 通过我们自己实现的一个 save 方法来保存这些数据 (音效状态), 然后, 在应用程序启动时, 通过一个 load 方法来读取这些数据, 如代码清单 6-2 所示 (具体实现参见本书所附代码: 第 6 章 \Examples_06_02)。

代码清单 6-2 第 6 章 \Examples_06_02 \src \com \yarin \android \Examples_06_02 \Activity01.java

```
public class Activity01 extends Activity
{
    private MIDIPlayer mMIDIPlayer = null;
    private boolean mbMusic = false;
    private TextView mTextView = null;
    public void onCreate(Bundle savedInstanceState)
```

```

{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mTextView = (TextView) this.findViewById(R.id.TextView01);

    mMIDIPlayer = new MIDIPlayer(this);
    /* 读取文件数据 */
    load();

    if (mbMusic)
    {
        mTextView.setText("当前音乐状态: 开");
        mbMusic = true;
        mMIDIPlayer.PlayMusic();
    }
    else
    {
        mTextView.setText("当前音乐状态: 关");
    }
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            mTextView.setText("当前音乐状态: 开");
            mbMusic = true;
            mMIDIPlayer.PlayMusic();
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            mTextView.setText("当前音乐状态: 关");
            mbMusic = false;
            mMIDIPlayer.FreeMusic();
            break;
    }
    return true;
}

public boolean onKeyDown(int keyCode, KeyEvent event)
{
    if (keyCode == KeyEvent.KEYCODE_BACK)
    {
        /* 退出应用程序时保存数据 */
        save();

        if (mbMusic)
        {
            mMIDIPlayer.FreeMusic();
        }
        this.finish();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}

```

```

/* 装载、读取数据 */
void load()
{
    /* 构建 Properties 对象 */
    Properties properties = new Properties();
    try
    {
        /* 开发文件 */
        FileInputStream stream = this.openFileInput("music.cfg");
        /* 读取文件内容 */
        properties.load(stream);
    }
    catch (FileNotFoundException e)
    {
        return;
    }
    catch (IOException e)
    {
        return;
    }
    /* 取得数据 */
    mbMusic = Boolean.valueOf(properties.get("bmusic").toString());
}

/* 保存数据 */
boolean save()
{
    Properties properties = new Properties();

    /* 将数据打包成 Properties */
    properties.put("bmusic", String.valueOf(mbMusic));
    try
    {
        FileOutputStream stream = this.openFileOutput("music.cfg",
            Context.MODE_WORLD_WRITEABLE);

        /* 将打包好的数据写入文件中 */
        properties.store(stream, "");
    }
    catch (FileNotFoundException e)
    {
        return false;
    }
    catch (IOException e)
    {
        return false;
    }
    return true;
}
}

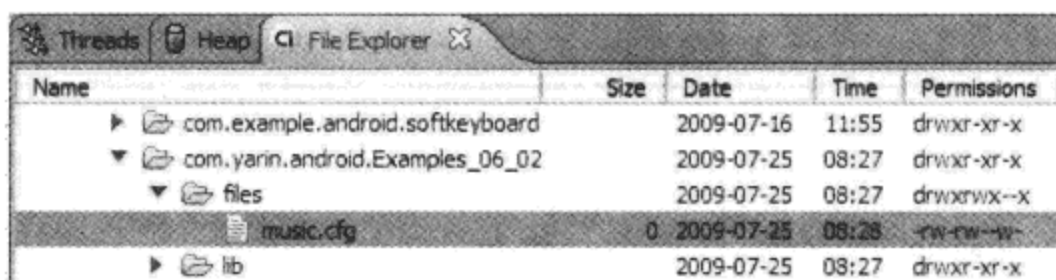
```

注意 如果使用绝对路径来存储文件，那么在其他应用程序中一样不能通过这个绝对路径来访问和操作该文件。

扩展学习

同样，我们知道了通过 Preferences 存储的数据保存在 shared_prefs 文件夹下，那么如果我们没

有指定路径的文件存储方式，数据又保存在什么地方呢？如果我们使用了文件存储数据的方式，系统就会在和 `shared_prefs` 相同的目录中产生一个名为 `files` 的文件夹，它下面的文件就是我们通过 `Files` 存储数据的文件。那么本节示例的 `Examples_06_02` 项目所存储的数据保存在如图 6-4 所示的文件目录中。



Name	Size	Date	Time	Permissions
com.example.android.softkeyboard		2009-07-16	11:55	drwxr-xr-x
com.yarin.android.Examples_06_02		2009-07-25	08:27	drwxr-xr-x
files		2009-07-25	08:27	drwxrwx--x
music.cfg	0	2009-07-25	08:28	-rw-rw-rw-
lib		2009-07-25	08:27	drwxr-xr-x

图 6-4 Files 方式存储数据的文件目录

如果在开发一个应用程序时，需要通过加载一个文件的内容来初始化程序，就可以在编译程序之前，在 `res/raw/tempFile` 中建立一个 `static` 文件，这样可以在程序中通过 `Resources.openRawResource(R.raw.文件名)` 方法同样返回一个 `InputStream` 对象，直接读取文件内容。

6.4 数据存储之 Network

通过网络来获取和保存数据资源，这个方法需要设备保持网络连接状态，所以相对存在一些限制。将数据存储到网络上的方法很多，比如将要保存的数据以文件的方式上传到服务器、发送邮件等等。本节我们将在应用程序退出时，将数据发送到电子邮件中备份，要发送电子邮件首先需要在模拟器中配置电子邮件账户。下面我们就一步一步来配置电子邮件账户。

(1) 启动模拟器，打开“菜单”，选择“电子邮件”项，在启动之后，填写好邮件地址和密码，选中“默认情况下从该账户发送电子邮件”。如图 6-5 所示。

(2) 点击“下一步”按钮，程序将自动配置电子邮件的相关信息。如图 6-6 所示。



图 6-5 设置电子邮件

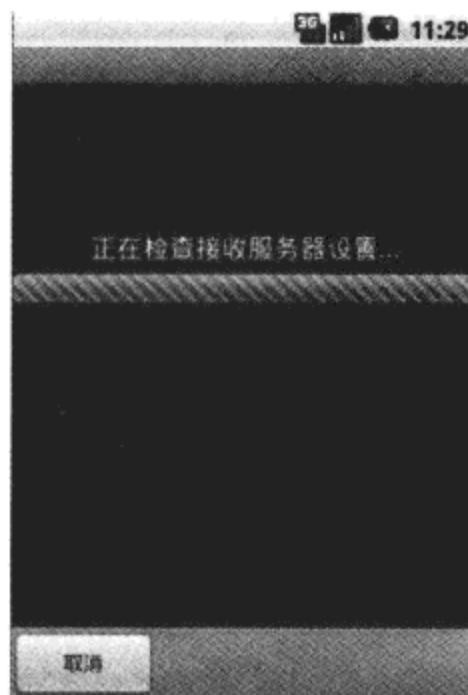
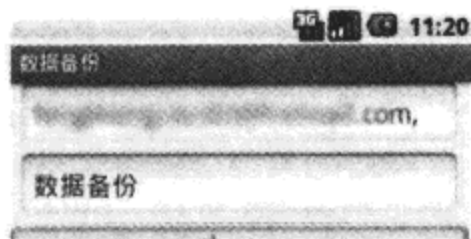
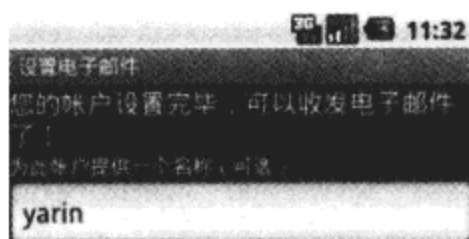


图 6-6 自动检测电子邮件

(3) 等待自动配置完成后, 设置好名称等相关信息, 如图 6-7 所示。点击“完成”, 电子邮件配置成功。

配好电子邮件账户之后, 我们就可以通过程序来调用模拟器的电子邮件客户端发送邮件了。Android 中发送电子邮件是通过 `startActivity` 方法来调用要发送的邮件数据的 `Intent`。我们可以通过 `putExtra` 方法来设置邮件的主题、内容、附件等等 (具体实现参见本书所附代码: 第 6 章 \Examples_06_03)。

运行效果如图 6-8 所示, 当我们发送邮件之后, 到邮箱中去查看, 则收到一份我们在程序中发送的邮件, 如图 6-9 所示。




```

        setContentView(R.layout.main);
        miCount=1000;
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_BACK)
        {
            //退出应用程序时保存数据
            /* 发送邮件的地址 */
            Uri uri = Uri.parse("mailto:xxxxx@gmail.com");
            /* 创建 Intent */
            Intent it = new Intent(Intent.ACTION_SENDTO, uri);
            /* 设置邮件的主题 */
            it.putExtra(android.content.Intent.EXTRA_SUBJECT,
                "数据备份");
            /* 设置邮件的内容 */
            it.putExtra(android.content.Intent.EXTRA_TEXT, "本次计数:
                "+miCount);
            /* 开启 */
            startActivity(it);

            this.finish();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}

```

既然可以将数据发送到网络上保存，当然也可以获得网络上的数据，下面我们通过网络来读取一个文件的内容，然后将其显示到 `TextView` 中，这就需要一个网络服务器来存储我们的文件，为了方便起见，笔者在本机架设了一个 `Web` 服务器。在本节中会使用一些有关网络的知识，大家可以参考本书第 8 章的内容。首先我们看看要读取的文件是什么内容，如图 6-10 所示。而通过程序读取的 `http://192.168.1.110:8080/android.txt` 的内容如图 6-11 所示（具体实现参见本书所附代码：第 6 章\Examples_06_04）。

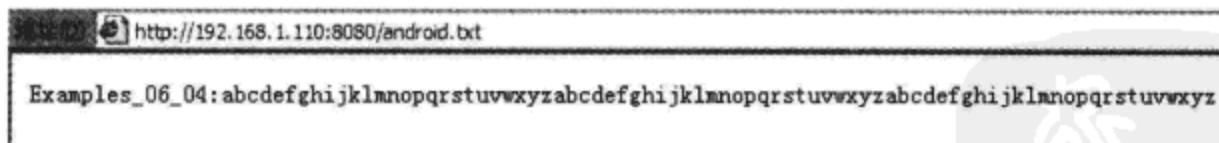


图 6-10 通过网页访问我们要读取的 `android.txt` 文件内容

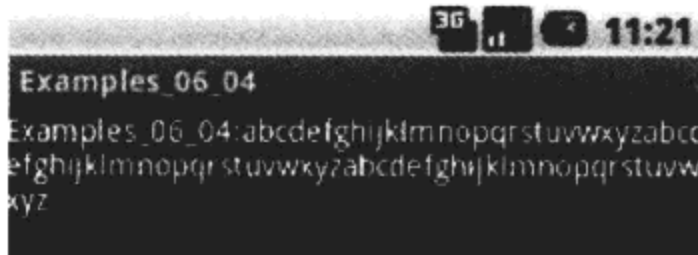


图 6-11 读取 `android.txt` 文件的内容

下面我们来看看程序如何实现读取网络文件，如代码清单 6-4 所示。

代码清单 6-4 第6章\Examples_06_04\src\com\yarin\android\Examples_06_04\Activity01.java

```

public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        TextView tv = new TextView(this);

        String myString = null;

        try
        {
            /* 定义我们要访问的地址 url */
            URL uri = new URL("http://192.168.1.110:8080/android.txt");

            /* 打开这个 url 链接 */
            URLConnection ucon = uri.openConnection();

            /* 从上面的链接中取得 InputStream */
            InputStream is = ucon.getInputStream();

            BufferedInputStream bis = new BufferedInputStream(is);
            ByteBuffer baf = new ByteBuffer(100);
            int current = 0;
            /* 一直读到文件结束 */
            while ((current = bis.read()) != -1)
            {
                baf.append((byte) current);
            }
            myString = new String(baf.toByteArray());
        }
        catch (Exception e)
        {
            myString = e.getMessage();
        }
        /* 将信息设置到 TextView */
        tv.setText(myString);

        /* 将 TextView 显示到屏幕上 */
        this.setContentView(tv);
    }
}

```

由于在程序中访问了外部网络，我们需要在 AndroidManifest.xml 文件中给予权限，代码如下：

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

6.5 Android 数据库编程

前文已经讲述了在 Android 平台中数据存储的 3 种方式，可以看出这 3 种方式都是存储一些简单的、数据量较小的数据，如果需要对大量的数据进行存储、管理以及升级维护（比如实现一个理

财工具),可能就需要随时添加数据、查看数据、更新数据,这时如果使用前面的3种方式就几乎不能满足需要了,而Google也考虑到了这些问题,因此提供了SQLite数据库,专门用来处理数据量较大的数据;它在数据的存储、管理、维护等各个方面都更加合理,功能更加强大。很多系统自带的应用都使用了SQLite数据库,比如电话本。本节我们就将详细地介绍SQLite数据库在Android中的使用。

6.5.1 SQLite 简介

SQLite 第一个 Alpha 版本诞生于 2000 年 5 月,它是一款轻型数据库,它的设计目标是嵌入式的,而且目前已经在很多嵌入式产品中使用,它占用的资源非常少,在嵌入式设备中,可能只需要几百 KB 的内存就够了。也许这正是 Android 系统要采用 SQLite 数据库的原因之一吧。

SQLite 数据库是 D. Richard Hipp 用 C 语言编写的开源嵌入式数据库,支持的数据库大小为 2TB。它具有如下特征。

1. 轻量级

SQLite 和 C/S 模式的数据库软件不同,它是进程内的数据库引擎,因此不存在数据库的客户端和服务端。使用 SQLite 一般只需要带上它的一个动态库,就可以享受它的全部功能。而且那个动态库的尺寸也相当小。

2. 独立性

SQLite 数据库的核心引擎本身不依赖第三方软件,使用它也不需要“安装”。所以在部署的时候能够省去不少麻烦。

3. 隔离性

SQLite 数据库中所有的信息(比如表、视图、触发器等)都包含在一个文件内,方便管理和维护。

4. 跨平台

SQLite 数据库支持大部分操作系统,除了我们在电脑上使用的操作系统之外,很多手机操作系统同样可以运行,比如 Android、Windows Mobile、Symbian、Palm 等。

5. 多语言接口

SQLite 数据库支持很多语言编程接口,比如 C/C++、Java、Python、dotNet、Ruby、Perl 等,得到更多开发者的喜爱。

6. 安全性

SQLite 数据库通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据,但只有一个可以写入数据。在某个进程或线程向数据库执行写操作之前,必须获得独占锁定。在发出独占锁定后,其他的读或写操作将不会再发生。

有关 SQLite 数据库的优点太多了,由于篇幅关系这里就列举这些。如果需要了解更多请参考 SQLite 官方网站(<http://www.sqlite.org/>)。下面我们将介绍在 Android 中如何使用 SQLite 数据库。

6.5.2 SQLite 编程详解

SQLite 数据库功能非常强大,使用起来也非常方便,SQLite 数据库的一般操作包括:创建数

数据库、打开数据库、创建表、向表中添加数据、从表中删除数据、修改表中的数据、关闭数据库、删除指定表、删除数据库和查询表中的某条数据。下面我们分别来学习这些基本操作。

1. 创建和打开数据库

在 Android 中创建和打开一个数据库都可以使用 `openOrCreateDatabase` 方法来实现，因为它会自动去检测是否存在这个数据库，如果存在则打开，如果不存在则创建一个数据库；创建成功则返回一个 `SQLiteDatabase` 对象，否则抛出异常 `FileNotFoundException`。下面我们来创建一个名为“Examples_06_05.db”的数据库，并返回一个 `SQLiteDatabase` 对象 `mSQLiteDatabase`。

```
mSQLiteDatabase =
    this.openOrCreateDatabase("Examples_06_05.db", MODE_PRIVATE, null);
```

2. 创建表

一个数据库中可以包含多个表，我们的每一条数据都保存在一个指定的表中，要创建表可以通过 `execSQL` 方法来执行一条 SQL 语句。`execSQL` 能够执行大部分的 SQL 语句，下面我们来创建一个名为 `table1` 且包含 3 个字段的表。具体代码如下：

```
String CREATE_TABLE
    = "CREATE TABLE table1 (_id INTEGER PRIMARY KEY,num INTERGER,data TEXT)";
mSQLiteDatabase.execSQL(CREATE_TABLE);
```

3. 向表中添加一条数据

可以使用 `insert` 方法来添加数据，但是 `insert` 方法要求把数据都打包到 `ContentValues` 中，`ContentValues` 其实就是一个 `Map`，`Key` 值是字段名称，`Value` 值是字段的值。通过 `ContentValues` 的 `put` 方法就可以把数据放到 `ContentValues` 对象中，然后插入到表中去。具体实现如下：

```
ContentValues cv = new ContentValues();
cv.put(TABLE_NUM, 1);
cv.put(TABLE_DATA, "测试数据库数据");
mSQLiteDatabase.insert(TABLE_NAME, null, cv);
```

这里同样可以使用 `execSQL` 方法来执行一条“插入”的 SQL 语句，代码如下：

```
String INSERT_DATA
    = "INSERT INTO table1 (_id,num,data) values(1,1,'通过 SQL 语句插入')";
mSQLiteDatabase.execSQL(INSERT_DATA);
```

4. 从表中删除数据

要删除数据可以使用 `delete` 方法，下面我们删除字段“`_id`”等于 1 的数据，具体代码如下：

```
mSQLiteDatabase.delete("Examples_06_05.db", " WHERE _id="+0, null);
```

通过 `execSQL` 方法执行 SQL 语句删除数据如下：

```
String DELETE_DATA = "DELETE FROM table1 WHERE _id=1";
mSQLiteDatabase.execSQL(DELETE_DATA);
```

5. 修改表中的数据

如果在添加了数据后发现数据有误，这时需要修改这个数据，可以使用 `update` 方法来更新一条数据。下面我们来修改“`num`”值为 0 的数据，具体代码如下：

```
ContentValues cv = new ContentValues();
cv.put(TABLE_NUM, 3);
cv.put(TABLE_DATA, "修改后的数据");
mSQLiteDatabase.update("table1", cv, "num " + "=" + Integer.toString(0), null);
```

6. 关闭数据库

关闭数据库很重要，也是大家经常容易忘记的。关闭的方法很简单，直接使用 `SQLiteDatabase` 的 `close` 方法。具体代码如下：

```
mSQLiteDatabase.close();
```

7. 删除指定表

这里我们使用 `execSQL` 方法来实现，具体代码如下：

```
mSQLiteDatabase.execSQL("DROP TABLE table1");
```

8. 删除数据库

要删除一个数据库，直接使用 `deleteDatabase` 方法即可，具体代码如下：

```
this.deleteDatabase("Examples_06_05.db");
```

9. 查询表中的某条数据

在 Android 中查询数据是通过 `Cursor` 类来实现的，当我们使用 `SQLiteDatabase.query()` 方法时，会得到一个 `Cursor` 对象，`Cursor` 指向的就是每一条数据。它提供了很多有关查询的方法，具体方法如表 6-1 所示。

表 6-1 `Cursor` 类常见方法

方 法	说 明
<code>move</code>	以当前位置为参考，将 <code>Cursor</code> 移动到指定的位置，成功返回 <code>true</code> ，失败返回 <code>false</code>
<code>moveToPosition</code>	将 <code>Cursor</code> 移动到指定的位置，成功返回 <code>true</code> ，失败返回 <code>false</code>
<code>moveToNext</code>	将 <code>Cursor</code> 向前移动一个位置，成功返回 <code>true</code> ，失败返回 <code>false</code>
<code>moveToLast</code>	将 <code>Cursor</code> 向后移动一个位置，成功返回 <code>true</code> ，失败返回 <code>false</code>
<code>moveToFirst</code>	将 <code>Cursor</code> 移动到第一行，成功返回 <code>true</code> ，失败返回 <code>false</code>
<code>isBeforeFirst</code>	返回 <code>Cursor</code> 是否指向第一项数据之前
<code>isAfterLast</code>	返回 <code>Cursor</code> 是否指向最后一项数据之后
<code>isClosed</code>	返回 <code>Cursor</code> 是否关闭
<code>isFirst</code>	返回 <code>Cursor</code> 是否指向第一项数据
<code>isLast</code>	返回 <code>Cursor</code> 是否指向最后一项数据
<code>isNull</code>	返回指定位置的值是否为 <code>Null</code>
<code>getCount</code>	返回总的的数据项数
<code>getInt</code>	返回当前行中指定索引的数据

下面我们就使用 `Cursor` 来查询数据库中的数据，具体代码如下：

```
Cursor cur = mSQLiteDatabase.rawQuery("SELECT * FROM table", null);
if (cur != null) {
    if (cur.moveToFirst()) {
        do {
            int numColumn = cur.getColumnIndex("num");
            int num = cur.getInt(numColumn);
        } while (cur.moveToNext());
    }
}
```

注意 使用 SQLiteDatabase 数据库后要及时关闭 (close), 否则可能会抛出 SQLiteException 异常。

上面我们分别讲述了 SQLite 数据库的基本操作, 现在来实现一个完整的数据库操作示例。首先运行项目, 初始化数据库 (创建数据库、创建表); 然后点击左方向键向表中插入一条数据, 按右方向键删除一条数据, 如图 6-12 所示; 按数字键 1 修改表中指定的一条数据, 如图 6-13 所示; 最后可以通过数字键 2 删除一个表, 数字键 3 删除数据库。

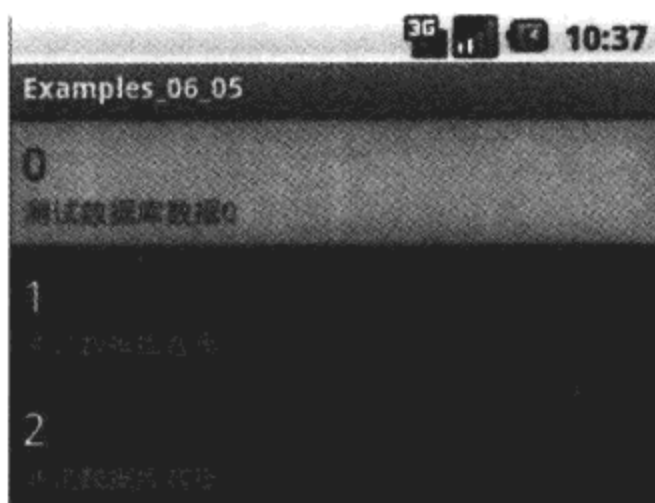


图 6-12 添加到表中的数据

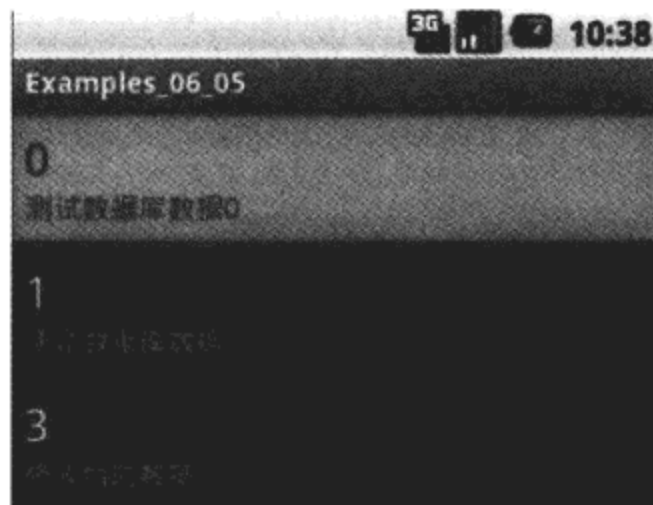


图 6-13 修改后的数据

在图 6-12 中我们通过 ListView 显示数据库中的数据, 该例子中我们创建了一个具有 3 个字段的表, 分别是 “_id”、“num”、“data”, 其中的 “_id” 是通过系统自动赋值的 (INTEGER PRIMARY KEY), 其他两个字段的值通过 ContentValues 使用 insert 来插入。如代码清单 6-5 所示 (具体实现参见本书所附代码: 第 6 章\Examples_06_05)。

代码清单 6-5 第 6 章\Examples_06_05\src\com\yarin\android\Examples_06_05\Activity01.java

```
public class Activity01 extends Activity
{
    private static int miscount = 0;
    /* 数据库对象 */
    private SQLiteDatabase mSQLiteDatabase = null;
    /* 数据库名 */
    private final static String DATABASE_NAME = "Examples_06_05.db";
    /* 表名 */
    private final static String TABLE_NAME = "table1";

    /* 表中的字段 */
    private final static String TABLE_ID = "_id";
    private final static String TABLE_NUM = "num";
    private final static String TABLE_DATA = "data";

    /* 创建表的 sql 语句 */
    private final static String CREATE_TABLE = "CREATE TABLE "
        + TABLE_NAME
        + " (" + TABLE_ID
        + " INTEGER PRIMARY KEY, "
        + TABLE_NUM + "
        + " INTEGER, " + TABLE_DATA + " TEXT)";

    /* 线性布局 */
    LinearLayout m_LinearLayout = null;
```

```

/* 列表视图-显示数据库中的数据 */
ListView m_ListView = null;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    /* 创建LinearLayout 布局对象 */
    m_LinearLayout = new LinearLayout(this);
    /* 设置布局LinearLayout 的属性 */
    m_LinearLayout.setOrientation(LinearLayout.VERTICAL);
    m_LinearLayout.setBackgroundColor(android.graphics.Color.BLACK);
    /* 创建ListView 对象 */
    m_ListView = new ListView(this);
    LinearLayout.LayoutParams param = new LinearLayout.
    LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
    LinearLayout.LayoutParams.WRAP_CONTENT);
    m_ListView.setBackgroundColor(Color.BLACK);
    /* 添加m_ListView 到m_LinearLayout 布局 */
    m_LinearLayout.addView(m_ListView, param);
    /* 设置显示m_LinearLayout 布局 */
    setContentView(m_LinearLayout);
    // 打开已经存在的数据库
    mSQLiteDatabase = this.openOrCreateDatabase(DATABASE_NAME, MODE_PRIVATE,
    null);
    // 获取数据库 Phones 的 Cursor
    try
    {
        /* 在数据库 mSQLiteDatabase 中创建一个表 */
        mSQLiteDatabase.execSQL(CREATE_TABLE);
    }
    catch (Exception e)
    {
        UpdataAdapter();
    }
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_LEFT:
            AddData();
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            DeleteData();
            break;
        case KeyEvent.KEYCODE_1:
            UpData();
            break;
        case KeyEvent.KEYCODE_2:
            DeleteTable();
            break;
        case KeyEvent.KEYCODE_3:
            DeleteDataBase();
            break;
    }
}

```



```

        return true;
    }

    /* 删除数据库 */
    public void DeleteDataBase()
    {
        this.deleteDatabase(DATABASE_NAME);
        this.finish();
    }

    /* 删除一个表 */
    public void DeleteTable()
    {
        mSQLiteDatabase.execSQL("DROP TABLE " + TABLE_NAME);
        this.finish();
    }

    /* 更新一条数据 */
    public void UpData()
    {
        ContentValues cv = new ContentValues();
        cv.put(TABLE_NUM, miCount);
        cv.put(TABLE_DATA, "修改后的数据" + miCount);
        /* 更新数据 */
        mSQLiteDatabase.update(TABLE_NAME, cv, TABLE_NUM + "=" + Integer.
            toString(miCount - 1), null);

        UpdataAdapter();
    }

    /* 向表中添加一条数据 */
    public void AddData()
    {
        ContentValues cv = new ContentValues();
        cv.put(TABLE_NUM, miCount);
        cv.put(TABLE_DATA, "测试数据库数据" + miCount);
        /* 插入数据 */
        mSQLiteDatabase.insert(TABLE_NAME, null, cv);
        miCount++;
        UpdataAdapter();
    }

    /* 从表中删除指定的一条数据 */
    public void DeleteData()
    {
        /* 删除数据 */
        mSQLiteDatabase.execSQL("DELETE FROM " + TABLE_NAME + " WHERE _id=" +
            Integer.toString(miCount));

        miCount--;
        if (miCount < 0)
        {
            miCount = 0;
        }
        UpdataAdapter();
    }

    /* 更新视图显示 */
    public void UpdataAdapter()
    {
        // 获取数据库 Phones 的 Cursor

```

```

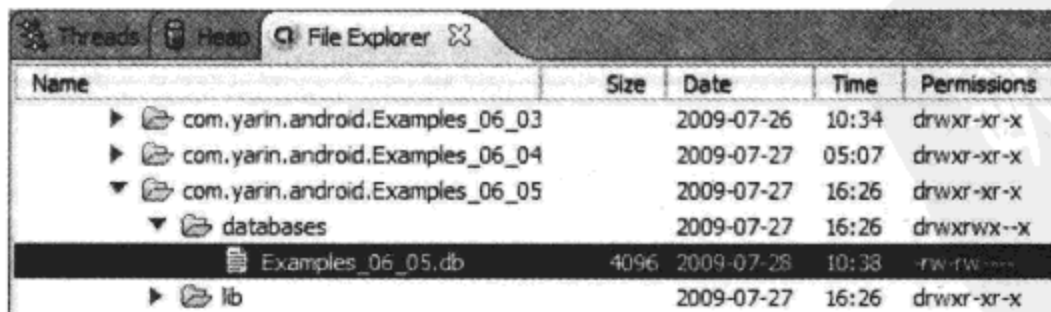
Cursor cur = mSQLiteDatabase.query(TABLE_NAME, new String[] { TABLE_ID,
TABLE_NUM, TABLE_DATA }, null, null, null, null, null);

miCount = cur.getCount();
if (cur != null && cur.getCount() >= 0)
{
    // ListAdapter 是 ListView 和后台数据的桥梁
    ListAdapter adapter = new SimpleCursorAdapter(this,
    // 定义 List 中每一行的显示模板
    // 表示每一行包含两个数据项
    android.R.layout.simple_list_item_2,
    // 数据库的 Cursor 对象
    cur,
    // 从数据库的 TABLE_NUM 和 TABLE_DATA 两列中取数据
    new String[] { TABLE_NUM, TABLE_DATA },
    // 与 NAME 和 NUMBER 对应的 Views
    new int[] { android.R.id.text1, android.R.id.text2 });

    /* 将 adapter 添加到 m_ListView 中 */
    m_ListView.setAdapter(adapter);
}
}
/* 按键事件处理 */
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    if (keyCode == KeyEvent.KEYCODE_BACK)
    {
        /* 退出时, 不要忘记关闭 */
        mSQLiteDatabase.close();
        this.finish();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
}

```

通过这个例子的学习, 相信大家对 SQLite 数据库的使用有了一定的了解! 还有一点需要大家了解——数据库存储的地址。我们来看看 Examples_06_05 项目中数据库存储的地址, 如图 6-14 所示。



Name	Size	Date	Time	Permissions
com.yarin.android.Examples_06_03		2009-07-26	10:34	drwxr-xr-x
com.yarin.android.Examples_06_04		2009-07-27	05:07	drwxr-xr-x
com.yarin.android.Examples_06_05		2009-07-27	16:26	drwxr-xr-x
databases		2009-07-27	16:26	drwxrwx--x
Examples_06_05.db	4096	2009-07-28	10:38	rw-rw-r--
lib		2009-07-27	16:26	drwxr-xr-x

图 6-14 数据库存储地址

由图 6-14 可以得出, SQLite 数据库存储的地址是: /data/data/<package_name>/databases/。

6.5.3 SQLiteOpenHelper 应用

前面我们已经学习了 SQLite 编程基础, 但是在实际开发中, 为了更好地管理和维护数据

库，我们会封装一个继承自 SQLiteOpenHelper 类的数据库操作类。SQLiteOpenHelper 的构造方法中分别需要传入 Context、数据库名称、CursorFactory（一般传入 null，否则为默认数据库）、数据库的版本号（不能为负数）。同样在 SQLiteOpenHelper 中首先执行的是 onCreate 方法（当数据库第一次被创建时）。当然，在构造函数时并没有真正创建数据库，而是在调用 getWritableDatabase 或者 getReadableDatabase 方法时才真正去创建数据库，并且返回一个 SQLiteDatabase 对象。因此，我们就可以很轻松地修改上一节中的 Examples_06_05 项目，使其更方便管理、升级维护等操作（具体实现参见本书所附代码：第6章\Examples_06_06）。

首先来分析我们自己封装的数据库管理类 MyDataBaseAdapter，如代码清单 6-6 所示。

代码清单 6-6 第6章\Examples_06_06\src\com\yarin\android\Examples_06_06\myDataBaseAdapter.java

```
public class MyDataBaseAdapter
{
    // 用于打印 log
    private static final String TAG = "MyDataBaseAdapter";
    // 表中一条数据的名称
    public static final String KEY_ID = "_id";
    // 表中一条数据的内容
    public static final String KEY_NUM = "num";
    // 表中一条数据的 id
    public static final String KEY_DATA = "data";
    // 数据库名称为 data
    private static final String DB_NAME = "Examples_06_06.db";
    // 数据库表名
    private static final String DB_TABLE = "table1";
    // 数据库版本
    private static final int DB_VERSION = 1;
    // 本地 Context 对象
    private Context mContext = null;
    // 创建一个表
    private static final String DB_CREATE = "CREATE TABLE "
        + DB_TABLE + " ("
        + KEY_ID
        + " INTEGER PRIMARY KEY, "
        + KEY_NUM + " INTERGER, "
        + KEY_DATA + " TEXT) ";

    // 执行 open() 打开数据库时，保存返回的数据库对象
    private SQLiteDatabase mSQLiteDatabase = null;
    // 由 SQLiteOpenHelper 继承过来
    private DatabaseHelper mDatabaseHelper = null;
    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        /* 构造函数-创建一个数据库 */
        DatabaseHelper(Context context)
        {
            // 当调用 getWritableDatabase()
            // 或 getReadableDatabase() 方法时
            // 则创建一个数据库
            super(context, DB_NAME, null, DB_VERSION);
        }
        /* 创建一个表 */
        @Override
        public void onCreate(SQLiteDatabase db)
```

```

    {
        // 数据库没有表时创建一个
        db.execSQL(DB_CREATE);
    }
    /* 升级数据库 */
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        db.execSQL("DROP TABLE IF EXISTS notes");
        onCreate(db);
    }
}

/* 构造函数-取得 Context */
public MyDataBaseAdapter(Context context)
{
    mContext = context;
}
// 打开数据库, 返回数据库对象
public void open() throws SQLException
{
    mDatabaseHelper = new DatabaseHelper(mContext);
    mSQLiteDatabase = mDatabaseHelper.getWritableDatabase();
}
// 关闭数据库
public void close()
{
    mDatabaseHelper.close();
}
/* 插入一条数据 */
public long insertData(int num, String data)
{
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_NUM, num);
    initialValues.put(KEY_DATA, data);
    return mSQLiteDatabase.insert(DB_TABLE, KEY_ID, initialValues);
}
/* 删除一条数据 */
public boolean deleteData(long rowId)
{
    return mSQLiteDatabase.delete(DB_TABLE, KEY_ID + "=" + rowId, null) > 0;
}
/* 通过 Cursor 查询所有数据 */
public Cursor fetchAllData()
{
    return mSQLiteDatabase.query(DB_TABLE, new String[] { KEY_ID, KEY_NUM,
        KEY_DATA }, null, null, null, null, null);
}
/* 查询指定数据 */
public Cursor fetchData(long rowId) throws SQLException
{
    Cursor mCursor =
        mSQLiteDatabase.query(true, DB_TABLE, new String[] { KEY_ID, KEY_NUM,
            KEY_DATA }, KEY_ID + "=" + rowId, null, null, null, null);

    if (mCursor != null)
    {
        mCursor.moveToFirst();
    }
}

```

```

    }
    return mCursor;
}
/* 更新一条数据 */
public boolean updateData(long rowId, int num, String data)
{
    ContentValues args = new ContentValues();
    args.put(KEY_NUM, num);
    args.put(KEY_DATA, data);
    return mSQLiteDatabase.update(DB_TABLE, args, KEY_ID + "=" + rowId, null) > 0;
}
}

```

接下来我们看看如何使用 MyDataBaseAdapter 类。首先需要通过构造方法传入 Context 对象，然后通过 open 方法来创建并获得数据库对象。通过 close 方法可以关闭数据库，最后我们就可以很轻松地操作数据库了，具体代码请参见代码清单 6-7。

代码清单 6-7 第6章\Examples_06_06\src\com\yarin\android\Examples_06_06\Activity01.java

```

public class Activity01 extends Activity
{
    private static int miCount = 0;

    /* 线性布局 */
    LinearLayout m_LinearLayout = null;
    /* 列表视图-显示数据库中的数据 */
    ListView m_ListView = null;

    MyDataBaseAdapter m_MyDataBaseAdapter;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        /* 创建 LinearLayout 布局对象 */
        m_LinearLayout = new LinearLayout(this);
        /* 设置布局 LinearLayout 的属性 */
        m_LinearLayout.setOrientation(LinearLayout.VERTICAL);
        m_LinearLayout.setBackgroundColor(android.graphics.Color.BLACK);

        /* 创建 ListView 对象 */
        m_ListView = new ListView(this);
        LinearLayout.LayoutParams param = new
        LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT);
        m_ListView.setBackgroundColor(Color.BLACK);
        /* 添加 m_ListView 到 m_LinearLayout 布局 */
        m_LinearLayout.addView(m_ListView, param);

        /* 设置显示 m_LinearLayout 布局 */
        setContentView(m_LinearLayout);
        /* 构造 MyDataBaseAdapter 对象 */
        m_MyDataBaseAdapter = new MyDataBaseAdapter(this);
        /* 取得数据库对象 */
        m_MyDataBaseAdapter.open();

        UpdataAdapter();
    }
}

```

```

    }

    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_LEFT:
                AddData();
                break;
            case KeyEvent.KEYCODE_DPAD_RIGHT:
                DeleteData();
                break;
            case KeyEvent.KEYCODE_1:
                UpData();
                break;
        }
        return true;
    }

    /* 更新一条数据 */
    public void UpData()
    {
        m_MyDataBaseAdapter.updateData(miCount - 1, miCount, "修改后的数据" +
            miCount);

        UpdataAdapter();
    }

    /* 向表中添加一条数据 */
    public void AddData()
    {
        m_MyDataBaseAdapter.insertData(miCount, "测试数据库数据" + miCount);
        miCount++;
        UpdataAdapter();
    }

    /* 从表中删除指定的一条数据 */
    public void DeleteData()
    {
        /* 删除数据 */
        m_MyDataBaseAdapter.deleteData(miCount);
        miCount--;
        if (miCount < 0)
        {
            miCount = 0;
        }
        UpdataAdapter();
    }

    /* 按键事件处理 */
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_BACK)
        {
            /* 退出时, 不要忘记关闭 */
            m_MyDataBaseAdapter.close();
            this.finish();
            return true;
        }
    }

```

```

    }
    return super.onKeyDown(keyCode, event);
}

/* 更新视图显示 */
public void UpdataAdapter()
{
    // 获取数据库 Phones 的 Cursor
    Cursor cur = m_MyDataBaseAdapter.fetchAllData();

    miCount = cur.getCount();
    if (cur != null && cur.getCount() >= 0)
    {
        // ListAdapter 是 ListView 和后台数据的桥梁
        ListAdapter adapter = new SimpleCursorAdapter(this,
        // 定义 List 中每一行的显示模板
        // 表示每一行包含两个数据项
        android.R.layout.simple_list_item_2,
        // 数据库的 Cursor 对象
        cur,
        // 从数据库的 TABLE_NUM 和 TABLE_DATA 两列中取数据
        new String[] {MyDataBaseAdapter.KEY_NUM,
        MyDataBaseAdapter.KEY_DATA },
        // 与 NAME 和 NUMBER 对应的 Views
        new int[] { android.R.id.text1, android.R.id.text2 });

        /* 将 adapter 添加到 m_ListView 中 */
        m_ListView.setAdapter(adapter);
    }
}
}

```

6.6 数据共享 (Content Providers)

Content Providers 是所有应用程序之间数据存储和检索的一个桥梁，它的作用就是使得各个应用程序之间实现数据共享。在 Android 中，Content Providers 是一个特殊的存储数据的类型，它提供了一套标准的接口用来获取、操作数据。系统也提供了音频、视频、图像、个人联系信息等几个常用的 Content Providers。如果你想公开自己的私有数据，可以通过创建自己的 Content Providers 类，或者当你对这些数据拥有控制、写入的权限时将这些数据添加到 Content Providers 中来实现共享。本节我们将学习如何使用 Content Providers。

1. ContentResolver

所有的 Content Providers 都会实现一些共同的接口，包括数据的查询、添加、更改和删除。应用可以通过一个唯一的 ContentResolver 接口来使用具体的某个 Content Providers。可以通过 getContentResolver() 方法来取得一个 ContentResolver 对象，然后就可以用 ContentResolver 提供的方法来操作你需要的 Content Provider 了。

```
ContentResolver cr = getContentResolver();
```

查询初始化时，Android 系统会确定查询目标的那个 ContentProvider，并且确保它处于启动运行状态。系统会把所有 ContentProvider 对象实例化，用户不需要自己做这些。实际上，用户根本不

需要同 `ContentProvider` 对象直接打交道。通常，对于每一种类型的 `ContentProvider` 只有一个实例，但是这个实例可以与在不同的程序或进程中的多个 `ContentResolver` 对象进行通信。进程之间的互动是由 `ContentResolver` 和 `ContentProvider` 类处理的。

`Content Provider` 把它的数作为数据库模型上的单一数据表提供出来，在数据表中，每一行是一条记录，每一列代表某一特定类型的值。例如，联系人信息及其电话号码会像下面这样输出，如表 6-2 所示。

表 6-2 `Content Provider` 的数据模型

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

2. URI

每个 `Content Providers` 都会对外提供一个公共的 URI（包装成 `Uri` 对象），如果应用程序有数据需要共享时，就需要使用 `Content Providers` 为这些数据定义一个 URI，然后其他的应用程序就可以通过 `Content Providers` 传入这个 URI 来对数据进行操作。URI 由 3 个部分组成：“content://”、数据的路径、标识 ID（可选）。例如以下是系统的一些 URI：

`content://media/internal/images` （这个 URI 将返回设备上存储的所有图片）

`content://contacts/people/5` （联系人信息中 ID 为 5 的联系人记录）

`content://contacts/people/` （这个 URI 将返回设备上的所有联系人信息）

为了使得大家更容易理解，在 `android.provider` 包下提供了一系列的辅助类，其中包含了很多以类变量形式给出的查询字符串，因此我们可以将上面的第 2 个 URI 改写成以下方式：

```
Uri person = ContentUris.withAppendedId(People.CONTENT_URI, 5);
```

所有与其 `Content Providers` 互动的过程中都可以使用这个 URI 常量。每个 `ContentResolver` 都把 URI 作为其第一个参数，它决定了 `ContentResolver` 将与哪一个 `Provider` 对话。

3. 查询数据

要查询 `Content Providers` 的内容，可以使用 `ContentResolver.query()` 方法或者 `Activity.managedQuery()` 方法，它们都将返回一个 `Cursor` 对象，不同的是 `managedQuery` 包含了 `Cursor` 生命周期，当 `Activity` 被暂停后，如果要使用 `Cursor` 就必须通过 `Activity.startManagingCursor()` 方法来重新启动。`ContentResolver.query()` 方法或者 `Activity.managedQuery()` 都需要传入一个 URI 来确定当前要操作的数据，下面我们来查询联系人信息中 ID 为 5 的联系人记录，代码如下：

```
Cursor cur = managedQuery(person, null, null, null);
```

当然还可以得到更详细的内容，下面我们将取得联系人的姓名、电话，并按名字的升序来排列，代码如下：

```
String[] projection = new String[] {
    People._ID,
```

```

        People._COUNT,
        People.NAME,
        People.NUMBER
    };
    Uri contacts = People.CONTENT_URI;
    Cursor managedCursor = managedQuery(contacts,
        projection, // 返回指定列的数据
        null,        // 返回所有行的数据
        null,        // 可选参数
        People.NAME + " ASC"); // 按名字的升序排列

```

一条查询返回的是一组数据的集合。列名、缺省排序、数据类型对每个 Content Provider 是特定的，但是每个 Provider 都有一个 `_id` 列，它为每条记录保存一个数字 ID 值。每个 Provider 都可以把返回的记录数量作为 `_count` 列报告出来，这个值对所有的行来说是相同的。那么我们查询的结果就如表 6-3 所示。

表 6-3 查询后返回的数据集合

_ID	_COUNT	NAME	NUMBER
44	3	Alan Vain	212 555 1234
13	3	Bully Pulpit	425 555 6677
53	3	Rex Cars	201 555 4433

读取查询的数据由 `Cursor` 对象的方法来操作，`Cursor` 对象用于在结果集中前向或后向列举数据，`Cursor` 对象只能用来读数据。增加、修改和删除数据必须使用 `ContentResolver` 对象。代码如下：

```

private void getColumnData(Cursor cur) {
    if (cur.moveToFirst()) {
        String name;
        String phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        String imagePath;
        do {
            name = cur.getString(nameColumn);
            phoneNumber = cur.getString(phoneColumn);
        } while (cur.moveToNext());
    }
}

```

如果是通过 ID 来查询某一特定记录的，那么这个结果只有一个值，否则，它将包含多个值（如果没有匹配的值，将返回空）。我们可以读取记录的特定字段，但是必须知道字段的类型，这是因为 `Cursor` 对象读每种数据类型都有各自的方法，例如，`getString()`、`getInt()`、和 `getFloat()` 方法（当然，对于大多数类型，如果调用读字符串的函数，`Cursor` 对象将会返回一个代表这个数据的字符串。）`Cursor` 可以通过列的索引得到列名，或者通过列名得到列索引。

4. 修改数据

我们可以使用 `ContentResolver.update()` 方法来修改数据。下面我们创建一个 `updateRecord` 方法专门用来修改数据，代码如下：

```

private void updateRecord(int id, int num) {

```

```

Uri uri = ContentUris.withAppendedId(People.CONTENT_URI, id);
ContentValues values = new ContentValues();
values.put(People.NUMBER, num);
getContentResolver().update(uri, values, null, null);
}

```

5. 添加数据

我们可以调用 `ContentResolver.insert()` 方法来增加一条记录，该方法接受一个要增加的记录的目标 URI，以及一个包含了新记录值的 `Map` 对象，调用后的返回值是新记录的 URI（包含记录号）。具体代码如下：

```

import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;
ContentValues values = new ContentValues();
values.put(People.NAME, "Abraham Lincoln");
values.put(People.STARRED, 1);
Uri uri = getContentResolver().insert(People.CONTENT_URI, values);

```

下面我们来为刚刚添加的记录加入新的数据，这就得使用上面 `insert` 方法所返回的 `Uri`。具体代码如下。

```

Uri phoneUri = null;
Uri emailUri = null;
phoneUri = Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);
values.clear();
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, "1233214567");
getContentResolver().insert(phoneUri, values);
emailUri = Uri.withAppendedPath(uri, People.ContactMethods.CONTENT_DIRECTORY);
values.clear();
values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);
values.put(People.ContactMethods.DATA, "test@example.com");
values.put(People.ContactMethods.TYPE, People.ContactMethods.TYPE_HOME);
getContentResolver().insert(emailUri, values);

```

如果一个查询返回诸如图像和声音的二进制数据，那么在数据表中可以被直接存储或者数据的表入口是指定一个 `Content` 的字符串：URI，通过这个 URI 可以得到数据。一般说来，小数量的数据（20K 到 50K 或者更少）常常在表中直接存储，通过 `Cursor.getBlob()` 函数读取，这个函数返回的是一个二进制数组。如果数据表入口是一个 `Content`：URI，就不要直接打开和读取文件（一个原因是权限问题），而是通过调用 `ContentResolver.openInputStream()` 函数来得到一个可以用来读数据的 `InputStream` 对象，具体代码如下：

```

ContentValues values = new ContentValues(3);
values.put(Media.DISPLAY_NAME, "road_trip_1");
values.put(Media.DESCRPTION, "Day 1, trip to Los Angeles");
values.put(Media.MIME_TYPE, "image/jpeg");
Uri uri = getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, values);
try {
    OutputStream outputStream = getContentResolver().openOutputStream(uri);
    sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outputStream);
    outputStream.close();
} catch (Exception e) {}

```

6. 删除数据

删除记录则相当简单了，只需要调用 `ContentResolver.update()` 方法即可，代码如下：

```
Uri uri = People.CONTENT_URI;
getContentResolver().delete(uri, null, null);
//指定 WHERE 条件语句来删除
getContentResolver().delete(uri, "NAME=" + "'xxx'", null);
```

7. 创建 Content Provider

上面我们已经学会了如何使用 Content Provider，现在让我们来看下如何自己创建一个 Content Provider。在我们创建的 Content Provider 类中定义一个公共的、静态的常量 CONTENT_URI 来代表这个 URI 地址，该地址必须是唯一的。

```
public static final Uri CONTENT_URI =
Uri.parse("content://com.example.codelab.transporationprovider");
```

必须定义你要返回给客户端的数据列名。如果你正在使用 Android 数据库，则数据列的使用方式就和你以往所熟悉的其他数据库一样。但是，你必须为其定义一个叫 `_id` 的列，它用来表示每条记录的唯一性。模式使用“INTEGER PRIMARY KEY AUTOINCREMENT”自动更新。

如果你要处理的数据类型是一种比较新的类型，你就必须先定义一个新的 MIME 类型，以供 ContentProvider.getType(url) 来返回。MIME 类型有两种形式：一种是为指定的单个记录的，还有一种是为多条记录的。这里给出一种常用的格式：

单个记录的 MIME 类型：

```
vnd.android.cursor.item/vnd.yourcompanyname.contenttype
```

比如，一个请求列车信息的 URI，如 `content://com.example.transportationprovider/trains/122`，可能就会返回 `typevnd.android.cursor.item/vnd.example.rail` 这样一个 MIME 类型。

多个记录的 MIME 类型：

```
vnd.android.cursor.dir/vnd.yourcompanyname.contenttype
```

比如，一个请求所有列车信息的 URI，如 `content://com.example.transportationprovider/trains`，可能就会返回 `vnd.android.cursor.dir/vnd.example.rail` 这样一个 MIME 类型。

在 AndroidManifest.xml 中使用 `<provider>` 标签来设置 Content Provider。如果我们创建的 ContentProvider 类名为 MyContentProvider，则需要在 AndroidManifest.xml 中配置如下：

```
<provider android:name="MyContentProvider"
android:authorities="com.wissen.MyContentProvider" />
```

还要通过 `setReadPermission()` 和 `setWritePermission()` 来设置其操作权限。当然也可以在上面的代码中加入 `android:readPermission` 或者 `android:writePermission` 属性来控制其权限。

最后，我们要在其他应用程序中使用这个 ContentProvider，这就需要将 MyContentProvider 加入到项目中，也可以将我们定义了静态字段的文件打包成 jar 文件，加入到要使用的工程中。通过 `import` 来带入其中。

注意 因为 ContentProvider 可能被不同的进程和线程调用，所以这些方法必须是线程安全的。

现在我们来修改 SDK 中的 Notes 例子，来学习 ContentProvider 的创建及简单的使用（具体实现参见本书所附代码：第6章\Examples_06_07）。

首先创建 ContentProvider 的 CONTENT_URI 和一些字段数据，字段类可以继承自 BaseColumns 类，它包括了一些基本的字段，比如：_id 等，如代码清单 6-8 所示。

代码清单 6-8 第 6 章\Examples_06_07\src\com\yarin\android\Examples_06_07\NotePad.java

```

public class NotePad
{
    //ContentProvider 的 uri
    public static final String AUTHORITY = "com.google.provider.NotePad";
    private NotePad() {}
    // 定义基本字段
    public static final class Notes implements BaseColumns
    {
        private Notes() {}
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY +
            "/notes");
        // 新的 MIME 类型-多个
        public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/vnd.google.note";

        // 新的 MIME 类型-单个
        public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/vnd.google.note";

        public static final String DEFAULT_SORT_ORDER = "modified DESC";
        // 字段
        public static final String TITLE = "title";
        public static final String NOTE = "note";
        public static final String CREATEDDATE = "created";
        public static final String MODIFIEDDATE = "modified";
    }
}

```

然后我们需要来创建自己的 ContentProvider 类 NotePadProvider, 它包括了查询、添加、删除、更新等操作以及打开和创建数据库, 具体实现如代码清单 6-9 所示。

代码清单 6-9 第 6 章\Examples_06_07\src\com\yarin\android\Examples_06_07\NotePadProvider.java

```

public class NotePadProvider extends ContentProvider
{
    private static final String TAG = "NotePadProvider";
    // 数据库名
    private static final String DATABASE_NAME = "note_pad.db";
    private static final int DATABASE_VERSION = 2;
    // 表名
    private static final String NOTES_TABLE_NAME = "notes";
    private static HashMap<String, String> sNotesProjectionMap;
    private static final int NOTES = 1;
    private static final int NOTE_ID = 2;
    private static final UriMatcher sUriMatcher;
    private DatabaseHelper mOpenHelper;
    // 创建表 SQL 语句
    private static final String CREATE_TABLE = "CREATE TABLE "
        + NOTES_TABLE_NAME
        + " (" + Notes._ID
        + " INTEGER PRIMARY KEY, "
        + Notes.TITLE
        + " TEXT, "
        + Notes.NOTE
        + " TEXT, "
        + Notes.CREATEDDATE

```

```

+ " INTEGER,"
+ Notes.MODIFIEDDATE
+ " INTEGER" + ");";

static
{
    sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    sUriMatcher.addURI(NotePad.AUTHORITY, "notes", NOTES);
    sUriMatcher.addURI(NotePad.AUTHORITY, "notes/#", NOTE_ID);

    sNotesProjectionMap = new HashMap<String, String>();
    sNotesProjectionMap.put(Notes._ID, Notes._ID);
    sNotesProjectionMap.put(Notes.TITLE, Notes.TITLE);
    sNotesProjectionMap.put(Notes.NOTE, Notes.NOTE);
    sNotesProjectionMap.put(Notes.CREATEDDATE, Notes.CREATEDDATE);
    sNotesProjectionMap.put(Notes.MODIFIEDDATE, Notes.MODIFIEDDATE);
}

private static class DatabaseHelper extends SQLiteOpenHelper
{
    //构造函数-创建数据库
    DatabaseHelper(Context context)
    {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    //创建表
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(CREATE_TABLE);
    }
    //更新数据库
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        db.execSQL("DROP TABLE IF EXISTS notes");
        onCreate(db);
    }
}

public boolean onCreate()
{
    mOpenHelper = new DatabaseHelper(getContext());
    return true;
}

//查询操作
public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
{
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    switch (sUriMatcher.match(uri))
    {
        case NOTES:
            qb.setTables(NOTES_TABLE_NAME);
            qb.setProjectionMap(sNotesProjectionMap);
            break;

        case NOTE_ID:
            qb.setTables(NOTES_TABLE_NAME);
            qb.setProjectionMap(sNotesProjectionMap);
            qb.appendWhere(Notes._ID + "=" +

```



```

        uri.getPathSegments().get(1));
        break;

        default:
            throw new IllegalArgumentException("Unknown URI"+uri);
    }
    String orderBy;
    if (TextUtils.isEmpty(sortOrder))
    {
        orderBy = NotePad.Notes.DEFAULT_SORT_ORDER;
    }
    else
    {
        orderBy = sortOrder;
    }
    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
    Cursor c = qb.query(db,projection, selection, selectionArgs, null,
        null, orderBy);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}

// 如果有自定义类型, 必须实现该方法
public String getType(Uri uri)
{
    switch (sUriMatcher.match(uri))
    {
        case NOTES:
            return Notes.CONTENT_TYPE;

        case NOTE_ID:
            return Notes.CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Unknown URI
            " + uri);
    }
}

// 插入数据库
public Uri insert(Uri uri, ContentValues initialValues)
{
    if (sUriMatcher.match(uri) != NOTES)
    {
        throw new IllegalArgumentException("Unknown URI " + uri);
    }
    ContentValues values;
    if (initialValues != null)
    {
        values = new ContentValues(initialValues);
    }
    else
    {
        values = new ContentValues();
    }
    Long now = Long.valueOf(System.currentTimeMillis());
    if (values.containsKey(NotePad.Notes.CREATEDDATE) == false)
    {
        values.put(NotePad.Notes.CREATEDDATE, now);
    }
}

```



```

    }
    if (values.containsKey(NotePad.Notes.MODIFIEDDATE) == false)
    {
        values.put(NotePad.Notes.MODIFIEDDATE, now);
    }
    if (values.containsKey(NotePad.Notes.TITLE) == false)
    {
        Resources r = Resources.getSystem();
        values.put(NotePad.Notes.TITLE,
            r.getString(android.R.string.untitled));
    }
    if (values.containsKey(NotePad.Notes.NOTE) == false)
    {
        values.put(NotePad.Notes.NOTE, "");
    }
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    long rowId = db.insert(NOTES_TABLE_NAME, Notes.NOTE, values);
    if (rowId > 0)
    {
        Uri noteUri = ContentUris.WithAppendedId
            (NotePad.Notes.CONTENT_URI, rowId);
        getContext().getContentResolver().notifyChange(noteUri, null);
        return noteUri;
    }
    throw new SQLException("Failed to insert row into " + uri);
}

//删除数据
public int delete(Uri uri, String where, String[] whereArgs)
{
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri))
    {
        case NOTES:
            count = db.delete(NOTES_TABLE_NAME, where, whereArgs);
            break;

        case NOTE_ID:
            String noteId = uri.getPathSegments().get(1);
            count = db.delete(NOTES_TABLE_NAME, Notes._ID +
                "=" + noteId + (!TextUtils.isEmpty(where) ? "
                AND (" + where + ')': ""), whereArgs);
            break;

        default:
            throw new IllegalArgumentException("Unknown URI
                " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}

//更新数据
public int update(Uri uri, ContentValues values, String where, String[] whereArgs)
{
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri))
    {
        case NOTES:

```

```

        count = db.update(NOTES_TABLE_NAME, values,
            where, whereArgs);
        break;

    case NOTE_ID:
        String noteId = uri.getPathSegments().get(1);
        count = db.update(NOTES_TABLE_NAME, values,
            Notes._ID + "=" + noteId + (!TextUtils.isEmpty
            (where)? "AND (" + where + ')': ""), whereArgs);
        break;

    default:
        throw new IllegalArgumentException("Unknown URI
            " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
}

```

下面我们要来使用自己创建的 `NotePadProvider` 类，首先向其中插入两条数据，然后通过 `Toast` 来显示数据库中的所有数据，如代码清单 6-10 所示，运行效果如图 6-15 所示。

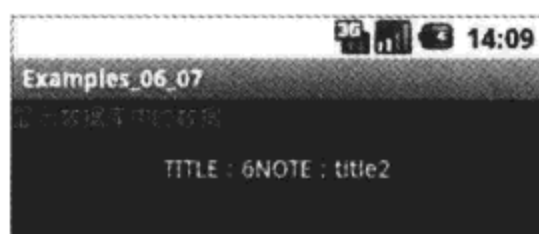


图 6-15 通过 Toast 显示数据库中的数据

代码清单 6-10 第 6 章\Examples_06_07\src\com\yarin\android\Examples_06_07\Activity01.java

```

public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        /* 插入数据 */
        ContentValues values = new ContentValues();
        values.put(NotePad.Notes.TITLE, "title1");
        values.put(NotePad.Notes.NOTE, "NOTENOTE1");
        getContentResolver().insert(NotePad.Notes.CONTENT_URI, values);
        values.clear();
        values.put(NotePad.Notes.TITLE, "title2");
        values.put(NotePad.Notes.NOTE, "NOTENOTE2");
        getContentResolver().insert(NotePad.Notes.CONTENT_URI, values);
        /* 显示 */
        displayNote();
    }
    /* 显示数据 */
    private void displayNote()
    {
        String columns[] = new String[]{NotePad.Notes._ID,
            NotePad.Notes.TITLE,
            NotePad.Notes.NOTE,
            NotePad.Notes.CREATEDDATE,

```

```

                                NotePad.Notes.MODIFIEDDATE});
Uri myUri = NotePad.Notes.CONTENT_URI;
Cursor cur = managedQuery(myUri, columns, null, null, null);
if (cur.moveToFirst())
{
    String id = null;
    String title = null;
    do
    {
        id = cur.getString(cur.getColumnIndex
            (NotePad.Notes._ID));
        title = cur.getString(cur.getColumnIndex
            (NotePad.Notes.TITLE));
        Toast toast=Toast.makeText(this, "TITLE: "+id +
            "NOTE: " + title, Toast.LENGTH_LONG);
        toast.setGravity(Gravity.TOP|Gravity.CENTER, 0, 40);
        toast.show();
    }
    while (cur.moveToNext());
}
}
}

```

最后不要忘记在 AndroidManifest.xml 文件中声明我们使用的 ContentProvider，具体代码如代码清单 6-11 所示。

代码清单 6-11 第 6 章\Examples_06_07\AndroidManifest.xml

```

<provider android:name="NotePadProvider"
    android:authorities="com.google.provider.NotePad"/>
<activity android:name=".Activity01" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
    <intent-filter>
        <data android:mimeType="vnd.android.cursor.dir/vnd.google.note"/>
    </intent-filter>
    <intent-filter>
        <data android:mimeType="vnd.android.cursor.item/vnd.google.note"/>
    </intent-filter>
</activity>

```

6.7 小结

本章主要介绍了 Android 中 4 种数据存储的方式，它们分别是：Shared Preferences、Files、Network、SQLite，着重介绍了使用最多、应用最广的 SQLite 数据库的使用。由于 Android 中数据基本是私有的，因此最后我们介绍了通过 Content Providers 来实现各个不同应用程序之间数据的传递和共享。每种存储方式我们都通过一个示例来告诉大家如何使用及其功能、作用，让大家在开发的过程中能够合理地选择数据的存储方式，提高开发效率。同时我们还复习了前面所学的 UI 界面设计等知识，让大家在学习新内容的同时，又温习了旧的知识。让大家逐渐更深入地理解 Android 系统的整体框架。

第 7 章

多媒体开发

自 1983 年世界上第一款商用手机发布到现在, 只经过了仅仅不到 30 年的时间, 全球手机用户已达 23 亿之多, 手机已成为人们必不可少的通信工具。从最初没有显示屏幕到黑白显示屏, 再到现在的彩色显示屏, 手机显示技术已经完成 3 次大的飞跃。随着达到高清电视全屏显示效果的新一代显示屏出现, 手机业将掀起新一轮革新风暴。消费者对手机产品在商务和娱乐方面的要求越来越高, 让手机日益成为便携式的商务或娱乐终端。在这种趋势下, 显示功能更加清晰, 色彩更加丰富, 成为手机产品发展的必然趋势。在硬件的推动下, 用户对手机软件的功能需求也越来越高。而 Android 系统也在不断更新以满足需求, 因此手机多媒体技术的开发也就不可避免成为热点。本章我们就将介绍 Android 平台上的多媒体开发。

7.1 多媒体开发详解

Android 的核心代码在今年 3 月底有了一次比较重大的改动, 尤其是多媒体方面的改动比较大, 主要目的是为了更好地实现 Camcorder, 以及进一步实现 Video Telephony。这些改动也是软硬件协调、结构和性能折中的一个结果。本节将介绍 Android 中多媒体开发基础。

7.1.1 Open Core

Open Core 是 Android 多媒体框架的核心, 所有 Android 平台的音频、视频的采集以及播放等操作都是通过它来实现。它也被称为 PV (Packet Video), Packet Video 是一家专门提供多媒体解决方案的公司。程序员可以通过 Open Core 方便快速地开发出想要的多媒体应用程序, 例如: 录音、播放、回放、视频会议、流媒体播放等等。下面是 Open Core 的框架图, 如图 7-1 所示。

从图 7-1 可以看出 Open Core 支持的格式包括: MPEG4、H.264、MP3、AAC、AMR、JPG、PNG、GIF 等。Open Core 多媒体框架有一套通用可扩展的接口, 针对第三方的多媒体编解码器, 输入、输出设备等等。具体功能如下:

- ❑ 多媒体文件的播放、下载, 包括: 3GPP, MPEG-4, AAC 和 MP3 containers。
- ❑ 流媒体文件的下载、实时播放, 包括: 3GPP, HTTP 和 RTSP/RTP。
- ❑ 动态视频和静态图像的编码、解码, 例如: MPEG-4, H.263 和 AVC (H.264), JPEG。
- ❑ 语音编码格式: AMR-NB 和 AMR-WB。
- ❑ 音乐编码格式: MP3, AAC, AAC+。
- ❑ 视频和图像格式: 3GPP, MPEG-4 和 JPEG。
- ❑ 视频会议: 基于 H324-M 标准。

- ❑ pvplayer: pvplayer 库文件的 Android.mk 文件，没有源文件。
- ❑ pvauthor: pvauthor 库文件的 Android.mk 文件，没有源文件。
- ❑ tools_v2: 编译工具以及一些可注册的模块。

在实际开发中我们并不会过多地研究 Open Core 的实现，Android 提供了上层的 Media API 给开发人员使用，图 7-2 所示为使用过程中的调用关系图。

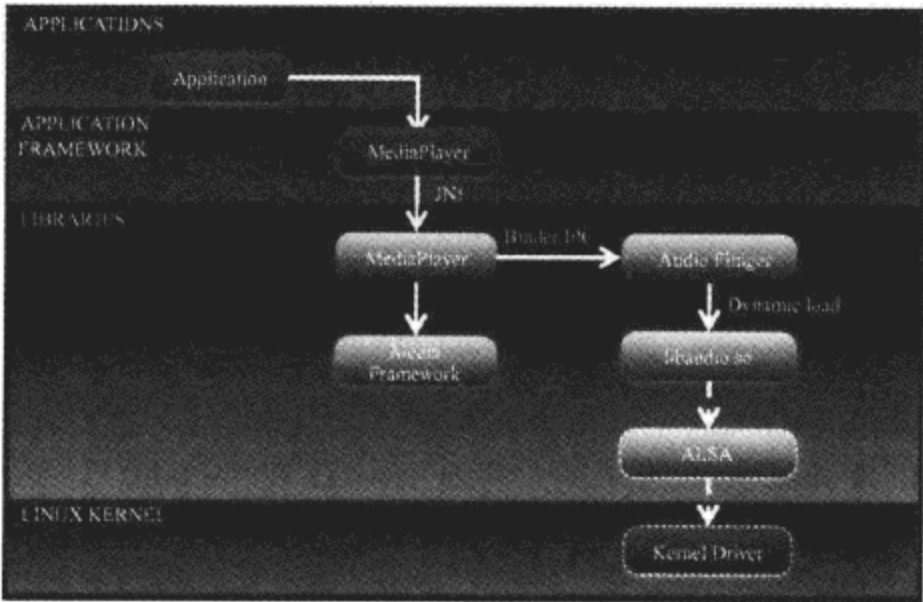


图 7-2 Android 中使用 Open Core 的调用关系图

有了对 Open Core 的了解，下面来看看 Android 中为我们提供了哪些接口来处理多媒体，Android 中多媒体系统相关的 Java 代码在./frameworks/base/media/java/android/media/目录下，主要有如表 7-1 所示文件。

表 7-1 Android 中多媒体接口

文 件	说 明
AudioManager.java	为上层应用提供了声音设置管理接口
AudioService.java	它在 SystemServer 中启动，为所有的音频相关的设置提供服务。在 AudioService 中定义了一个 AudioSystemThread 的类，用来监控音频控制相关的信号，当有请求时，它会通过调用 AudioSystem 的接口实现音频的控制，这里的消息处理是异步的。此外在 AudioService 还抽象出了一套发送音频控制信号的接口为 AudioManager 提供支持
AudioSystem.java	提供了音频系统的基本类型定义以及基本操作的接口。它对应于 frameworks/base/core/jni/android_media_AudioSystem.cpp
Ringtone.java	为铃声、闹钟等提供了快速的播放以及管理接口
RingtoneManager.java	直接为 PCM 数据提供支持，对应于 frameworks/base/core/jni/android_media_AudioTrack.cpp
AudioTrack.java	提供了为引用播放声音的接口，在加载文件等方面做了优化
SoundPool.java	提供了播放 DTMF tones 的支持，如电话的拨号音，对应于直接为 PCM 数据提供支持，对应于 frameworks/base/core/jni/android_media_ToneGenerator.cpp
ToneGenerator.java	这个是音频系统对外的录制接口，对应于 frameworks/base/core/jni/android_media_AudioRecord.cpp
AudioRecord.java	

7.1.2 MediaPlayer

MediaPlayer 类可以用来播放音频、视频和流媒体，MediaPlayer 包含了 Audio 和 Video 的播放功能，在 Android 的界面上，Music 和 Video 两个应用程序都是调用 MediaPlayer 实现的。首先我们来看看 MediaPlayer 的生命周期，如图 7-3 所示。

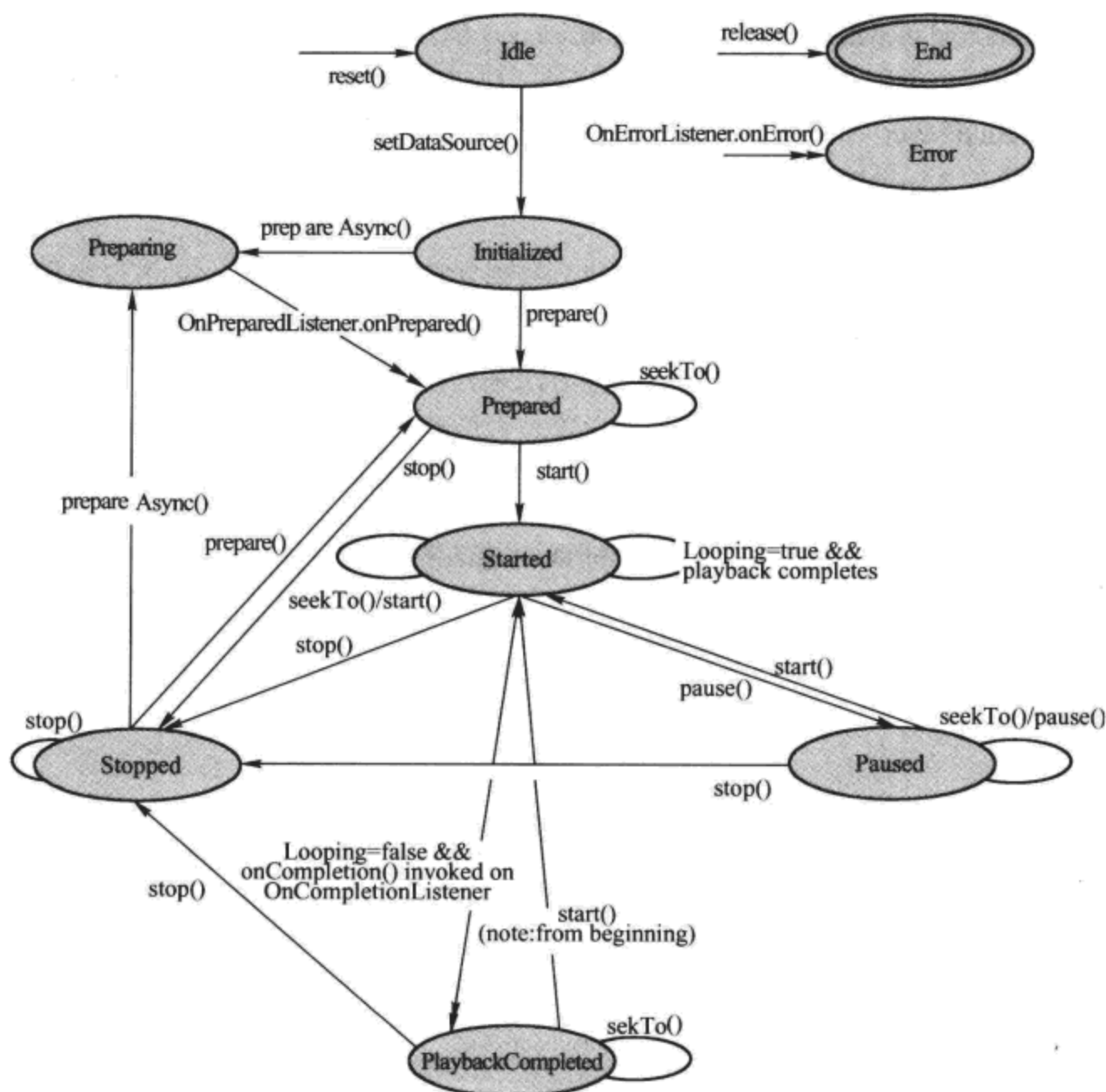


图 7-3 MediaPlayer 的生命周期

从图 7-3 中我们可以得到以下结论：

(1) 当一个 MediaPlayer 对象被新建或者调用 reset()方法之后，它处于空闲状态，在调用 release()方法后，才会处于结束状态。

□ 一个新建的 MediaPlayer 对象在调用 getCurrentPosition()、getDuration()、getVideoHeight()、getVideoWidth()、setAudioStreamType(int)、setLooping(boolean)、setVolume(float, float)、pause()、start()、stop()、seekTo(int)、prepare()、prepareAsync()方法时，不会触发 OnErrorListener.onError()事件，但是 MediaPlayer 对象如果调用了 reset()方法后，再使用这些方法则会触发 OnErrorListener.onError()事件。

□ 当 MediaPlayer 对象不再被使用时，最好通过 release()方法来释放，使其处于结束状态，以免造成不必要的错误。当 MediaPlayer 对象处于结束状态时，便不能再使用。

□ MediaPlayer 对象被新建时处于空闲状态，如果通过 create()方法创建之后便处于准备状态。

(2) 一般情况下，一些常用的播放控制操作可能因为音频、视频的格式不被支持或者质量较差以及流超时，也有可能由于开发者的疏忽使得 MediaPlayer 对象处于无效状态等而导致错误。这时可以通过注册

`setOnErrorListener(android.media.MediaPlayer.OnErrorListener)`方法实现 `OnErrorListener.onError()` 方法来监控这些错误。如果发生了错误, `MediaPlayer` 对象将处于错误状态, 可以使用 `reset()`方法来恢复错误。

(3) 任何 `MediaPlayer` 对象都必须先处于准备状态, 然后才开始播放。

(4) 要开始播放 `MediaPlayer` 对象都必须成功调用 `start()`方法。可以通过 `isPlaying()`方法来检测当前是否正在播放。

(5) 当 `MediaPlayer` 对象在播放时, 可以进行暂停和停止等操作, `pause()`方法暂停播放, `stop()`方法停止播放。处于暂停状态时可以通过 `start()`方法来恢复播放, 但是处于停止状态时则必须先调用 `pause()`方法处于准备状态, 然后再通过 `start()`方法来开始播放。

(6) 可以通过 `setLooping(boolean)`方法来设置是否循环播放。

要学习使用 `MediaPlayer` 类来播放多媒体, 首先看看 `MediaPlayer` 提供的常用方法, 如表 7-2 所示。

表 7-2 `MediaPlayer` 类的常用方法

方 法	说 明
<code>MediaPlayer</code>	构造方法
<code>create</code>	创建一个要播放的多媒体
<code>getCurrentPosition</code>	得到当前播放位置
<code>getDuration</code>	得到文件的时间
<code>getVideoHeight</code>	得到视频的高度
<code>getVideoWidth</code>	得到视频的宽度
<code>isLooping</code>	是否循环播放
<code>isPlaying</code>	是否正在播放
<code>pause</code>	暂停
<code>prepare</code>	准备(同步)
<code>prepareAsync</code>	准备(异步)
<code>release</code>	释放 <code>MediaPlayer</code> 对象
<code>reset</code>	重置 <code>MediaPlayer</code> 对象
<code>seekTo</code>	指定播放的位置(以毫秒为单位的时间)
<code>setAudioStreamType</code>	设置流媒体的类型
<code>setDataSource</code>	设置多媒体数据来源
<code>setDisplay</code>	设置用 <code>SurfaceHolder</code> 来显示多媒体
<code>setLooping</code>	设置是否循环播放
<code>setOnBufferingUpdateListener</code>	网络流媒体的缓冲监听
<code>setOnErrorListener</code>	设置错误信息监听
<code>setOnVideoSizeChangedListener</code>	视频尺寸监听
<code>setScreenOnWhilePlaying</code>	设置是否使用 <code>SurfaceHolder</code> 来显示
<code>setVolume</code>	设置音量
<code>start</code>	开始播放
<code>stop</code>	停止播放

因此我们可以得出, 在 Android 中通过 `MediaPlayer` 来播放音乐的步骤如下:

```
MediaPlayer mp = new MediaPlayer(); //构建 MediaPlayer 对象
mp.setDataSource("/sdcard/test.mp3"); //设置文件路径
mp.prepare(); //准备
mp.start(); //开始播放
```

7.1.3 MediaRecorder

`MediaRecorder` 类用来进行媒体采样, 包括音频和视频。`MediaRecorder` 作为状态机运行。需要设置不同的参数, 比如源设备和格式。设置后, 可执行任何时间长度的录制, 直到用户停止。还是先来看看 `MediaRecorder` 类的工作流程吧, 如图 7-4 所示。

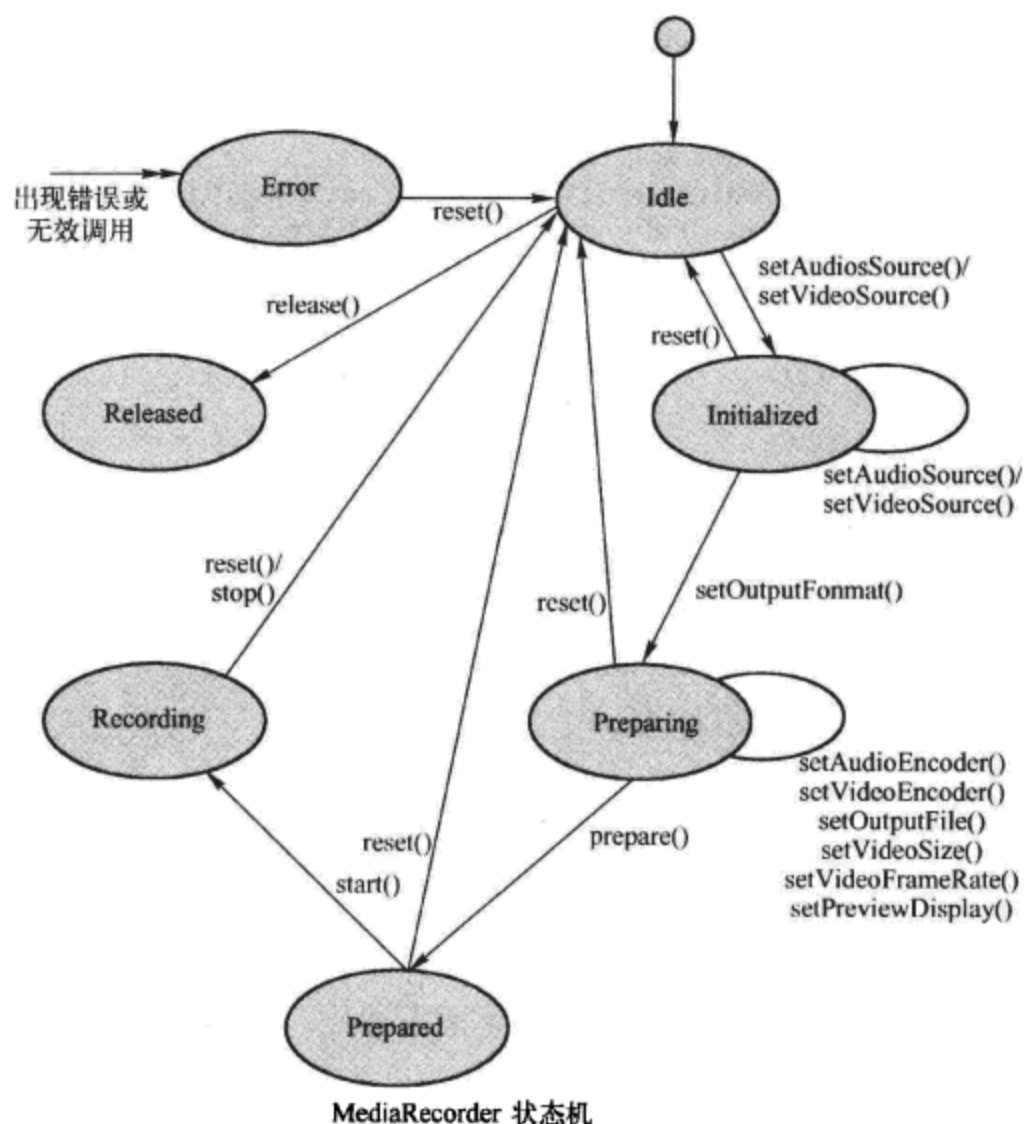


图 7-4 MediaRecorder 类工作原理

MediaRecorder 在底层同样是通过 Open Core 来实现的，但是要开发应用时需要使用 Android 为我们提供的 Java 接口，下面我们看看 MediaRecorder 提供了什么功能，如表 7-3 所示。

表 7-3 MediaRecorder 类的常用方法

方 法	说 明
MediaRecorder	构造方法
getMaxAmplitude	得到目前为止最大的幅度
prepare	准备录音机
release	释放 MediaRecorder 对象
reset	重置 MediaRecorder 对象，使其为空闲状态
setAudioEncoder	设置音频编码
setAudioSource	设置音频源
setCamera	设置摄像机
setMaxDuration	设置最大期限
setMaxFileSize	设置文件的最大尺寸
setOnErrorListener	错误监听
setOutputFile	设置输出文件
setOutputFormat	设置输出文件的格式
setPreviewDisplay	设置预览
setVideoEncoder	设置视频编码
setVideoFrameRate	设置视频帧的频率
setVideoSize	设置视频的宽度和高度（分辨率）
setVideoSource	设置视频源
start	开始录制
stop	停止录制

有了表 7-3 中的这些接口，在 Android 中实现录音就非常简单，下面是一段最简单的录音代码示例：

```
MediaRecorder recorder = new MediaRecorder();
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
recorder.setOutputFile(PATH_NAME);
recorder.prepare();
recorder.start();
//录音中...
recorder.stop();
recorder.reset();
recorder.release();
```

这里我们只是先了解一下 Android 中多媒体开发的原理以及一些简单的底层实现，读者若有兴趣可以看看多媒体开发的源代码，则能更深入地理解多媒体开发。在后面的章节中，我们会通过示例来详细讲述如何开发音频、视频等应用。

7.2 播放音乐

多媒体中最常见的就是音乐的播放了吧！前面我们曾讲述过使用 Server 来播放一个音乐的例子，但是这个音乐是在后台播放，并且操作也不是很人性化，我们在听音乐时不管使用什么播放器都不只是有暂停、停止等操作，通常都有上一首、下一首等按钮来切换歌曲，还有快进、播放列表、播放进度（时间）等很多的操作。有了上一节的基础，现在我们要做个简单的播放器就很简单了（具体实现参见本书所附代码：第 7 章\Examples_07_01）。

首先我们的播放器可能不只是希望播放一种格式的音乐，而且还需要自动识别指定的路径中哪些是可以播放的音频文件，哪些是不支持的格式。我们可以通过下面的代码来过滤文件类型：

```
class MusicFilter implements FilenameFilter
{
    public boolean accept(File dir, String name)
    {
        //这里还可以设置其他格式的音乐文件
        return (name.endsWith(".mp3"));
    }
}
```

现在我们已经得到了能够播放的音频文件，如何来实现一个播放列表的界面呢？本例我们使用 ListActivity 布局来实现。但是我们还需要几个按钮来表示开始、暂停等操作，我们使用 ImageButton 来实现，下面是我们的播放器的界面，请参见本书所附代码：第 7 章\Examples_07_01\res\layout\main.xml。

具体界面运行效果如图 7-5 所示。当然，大家可以设计出自己喜欢的、更漂亮的界面。

现在我们需要指定音乐文件的位置，本例我们设置路径为“/sdcard/”，因此我们需要向 SD 卡中添加一些音频文件。步骤如下：启动模拟器，在 Eclipse 上选择 DDMS 窗口，选择 File Explorer 标签。如图 7-6 所示，向左、向右箭头分别是拷贝进、拷贝出指定位置，减号则是删除选中的文件。

现在我们只需要设置这些 ImageButton 按钮的事件监听来处理相应的动作即可要播放音乐，首先调用 reset()，这样做将重置 MediaPlayer 到它的正常状态，这是必须的。如果正在放一首歌曲，同时又

想去改变这个数据源, 这个 `reset()` 功能也将停止任何正在播放的歌曲。因此, 如果一首歌曲是播放状态, 同时我们又要选择另一首歌曲, 那么它将在启动下一首歌曲之前停止这首正在播放的歌曲。然后我们通过路径找到这首歌曲(`setDataSource(String)`)同时调用 `prepare()` 和 `start()`。在这里指向 `MediaPlayer` 将启动播放你的歌曲。接下来的工作就是去装备一个 `OnCompletionListener` (`setOnCompletionListener(newOnCompletionListener())`)。在歌曲结束的时候, `onCompletion(MediaPlayer)` 这个功能将被调用。比如我们可以在这里设置自动播放下一首。下面我们来看看具体的实现代码, 如代码清单 7-1 所示。

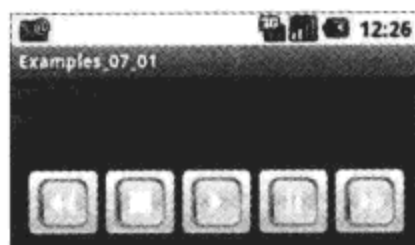


图 7-5 简易播放器界面

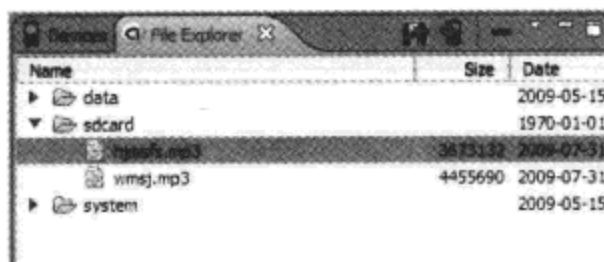


图 7-6 拷贝文件到模拟器中

代码清单 7-1 第7章\Examples_07_01\src\com\lyarin\android\Examples_07_01\Activity01.java

```
public class Activity01 extends ListActivity
{
    /* 几个操作按钮 */
    private ImageButton mFrontImageButton = null;
    private ImageButton mStopImageButton = null;
    private ImageButton mStartImageButton = null;
    private ImageButton mPauseImageButton = null;
    private ImageButton mNextImageButton = null;
    /* MediaPlayer 对象 */
    public MediaPlayer mMediaPlayer = null;
    /* 播放列表 */
    private List<String> mMusicList = new ArrayList<String>();
    /* 当前播放歌曲的索引 */
    private int currentListItem = 0;
    /* 音乐的路径 */
    private static final String MUSIC_PATH = new String("/sdcard/");
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        /* 更新显示播放列表 */
        musicList();
        /* 构建 MediaPlayer 对象 */
        mMediaPlayer = new MediaPlayer();

        mFrontImageButton = (ImageButton) findViewById(R.id.LastImageButton);
        mStopImageButton = (ImageButton) findViewById(R.id.StopImageButton);
        mStartImageButton = (ImageButton) findViewById(R.id.StartImageButton);
        mPauseImageButton = (ImageButton) findViewById(R.id.PauseImageButton);
        mNextImageButton = (ImageButton) findViewById(R.id.NextImageButton);

        // 停止按钮
        mStopImageButton.setOnClickListener(new ImageButton.OnClickListener()
        {
            public void onClick(View v)
```

```

        {
            /* 是否正在播放 */
            if (mMediaPlayer.isPlaying())
            {
                //重置 MediaPlayer 到初始状态
                mMediaPlayer.reset();
            }
        }
    });

    //开始按钮
    mStartImageButton.setOnClickListener(new ImageButton.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            playMusic(MUSIC_PATH + mMusicList.get(
                currentListItme));
        }
    });

    //暂停
    mPauseImageButton.setOnClickListener(new
    ImageButton.OnClickListener()
    {
        public void onClick(View view)
        {
            if (mMediaPlayer.isPlaying())
            {
                /* 暂停 */
                mMediaPlayer.pause();
            }
            else
            {
                /* 开始播放 */
                mMediaPlayer.start();
            }
        }
    });

    //下一首
    mNextImageButton.setOnClickListener(new
    ImageButton.OnClickListener()
    {
        public void onClick(View arg0)
        {
            nextMusic();
        }
    });

    //上一首
    mFrontImageButton.setOnClickListener(new
    ImageButton.OnClickListener()
    {
        public void onClick(View arg0)
        {
            FrontMusic();
        }
    });

```

```

        });
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_BACK)
        {
            mediaPlayer.stop();
            mediaPlayer.release();
            this.finish();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
    /* 当点击列表时, 播放被点击的音乐 */
    protected void onItemClick(ListView l, View v, int position, long id)
    {
        currentListItem = position;
        playMusic(MUSIC_PATH + mMUSICLIST.get(position));
    }

    /* 播放列表 */
    public void musicList()
    {
        //取得指定位置的文件设置显示到播放列表
        File home = new File(MUSIC_PATH);
        if (home.listFiles(new MusicFilter()).length > 0)
        {
            for (File file : home.listFiles(new MusicFilter()))
            {
                mMUSICLIST.add(file.getName());
            }
            ArrayAdapter<String> musicList = new ArrayAdapter<String>
                (Activity01.this, R.layout.musicitem, mMUSICLIST);
            setListAdapter(musicList);
        }
    }

    private void playMusic(String path)
    {
        try
        {
            /* 重置 MediaPlayer */
            mediaPlayer.reset();
            /* 设置要播放的文件的途径 */
            mediaPlayer.setDataSource(path);
            /* 准备播放 */
            mediaPlayer.prepare();
            /* 开始播放 */
            mediaPlayer.start();
            mediaPlayer.setOnCompletionListener(new
                OnCompletionListener()
            {
                public void onCompletion(MediaPlayer arg0)
                {
                    //播放完成一首之后进行下一首
                    nextMusic();
                }
            });
        }
    }

```

```

        });
    } catch (IOException e) {}
}
/* 下一首 */
private void nextMusic()
{
    if (++currentListItme >= mMusicList.size())
    {
        currentListItme = 0;
    }
    else
    {
        playMusic(MUSIC_PATH + mMusicList.get(currentListItme));
    }
}

/* 上一首 */
private void FrontMusic()
{
    if (--currentListItme >= 0)
    {
        currentListItme = mMusicList.size();
    }
    else
    {
        playMusic(MUSIC_PATH + mMusicList.get(currentListItme));
    }
}

}

/* 过滤文件类型 */
class MusicFilter implements FilenameFilter
{
    public boolean accept(File dir, String name)
    {
        //这里还可以设置其他格式的音乐文件
        return (name.endsWith(".mp3"));
    }
}

```

扩展学习

现在我们已经知道如何来播放指定 SD 卡上的音乐了，但是大家是否想到，我们在应用时（比如游戏中的音效），肯定是需要和我们的程序一起打包发布的，这样音乐就没有存在于 SD 卡上了，一般情况下会放在应用目录下的“res/raw\”中。那么如何来播放这样的音乐呢，其实很简单，只需要把上面例子中的 `setDataSource` 方法改成由 `create` 方法来创建一个指定资源索引的 `MediaPlayer` 对象，其他操作基本一样。下面我们来看看如何实现播放应用程序下“res/raw\”文件夹的音乐，如代码清单 7-2 所示（具体实现参见本书所附代码：第 7 章\Examples_07_02）。

代码清单 7-2 第 7 章\Examples_07_02\src\com\yarin\android\Examples_07_02\Activity01.java

```

public class Activity01 extends Activity
{
    private ImageButton    mStopImageButton;
    private ImageButton    mStartImageButton;
    private ImageButton    mPauseImageButton;

```



```

private TextView          mTextView;

private boolean           bIsReleased      = false;
private boolean           bIsPaused        = false;
private boolean           bIsPlaying       = false;

public MediaPlayer mMediaPlayer = new MediaPlayer(); ;
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mStopImageButton = (ImageButton) findViewById(R.id.StopImageButton);
    mStartImageButton = (ImageButton) findViewById(R.id.StartImageButton);
    mPauseImageButton = (ImageButton) findViewById(R.id.PauseImageButton);

    mTextView = (TextView) findViewById(R.id.TextView01);
    //停止按钮
    mStopImageButton.setOnClickListener(new ImageButton.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            if (bIsPlaying == true)
            {
                if ( !bIsReleased )
                {
                    mMediaPlayer.stop();
                    mMediaPlayer.release();
                    bIsReleased = true;
                }
                bIsPlaying = false;
                mTextView.setText("当前处于停止状态, 请按开
                始进行音乐播放!");
            }
        }
    });

    //开始按钮
    mStartImageButton.setOnClickListener(new ImageButton.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            try
            {
                if ( !bIsPlaying )
                {
                    bIsPlaying = true;
                    /* 装载资源中的音乐 */
                    mMediaPlayer = MediaPlayer.create
                    (Activity01.this, R.raw.test);
                    bIsReleased = false;
                    /* 设置是否循环 */
                    mMediaPlayer.setLooping(true);
                }
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    });
}

```

```

        mMediaPlayer.prepare();
    }
    catch (IllegalStateException e)
    {
    }
    catch (IOException e)
    {
    }
    mMediaPlayer.start();
    mTextView.setText("当前正在播放
    音乐!");
}
}
catch (IllegalStateException e)
{
    e.printStackTrace();
}
mMediaPlayer.setOnCompletionListener(new
OnCompletionListener()
{
    public void onCompletion(MediaPlayer arg0)
    {
        mMediaPlayer.release();
    }
});
});

//暂停
mPauseImageButton.setOnClickListener(new ImageButton.OnClickListener()
{
    public void onClick(View view)
    {
        if (mMediaPlayer != null)
        {
            if (bIsReleased == false)
            {
                if (bIsPaused == false)
                {
                    mMediaPlayer.pause();
                    bIsPaused = true;
                    bIsPlaying = true;
                    mTextView.setText("已经暂停, 请再次按暂停按钮恢复播放!");
                }
                else if (bIsPaused == true)
                {
                    mMediaPlayer.start();
                    bIsPaused = false;
                    mTextView.setText("音乐恢复播放!");
                }
            }
        }
    }
});
}
}

```

```

public boolean onKeyDown(int keyCode, KeyEvent event)
{
    if ( keyCode == KeyEvent.KEYCODE_BACK)
    {
        if ( !bIsReleased )
        {
            mMediaPlayer.stop();
            mMediaPlayer.release();
            bIsReleased = true;
        }
        this.finish();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
}

```

细心的读者已经发现 `create` 方法不仅可以加载应用程序中资源的索引，还可以加载一个 URI 地址，当然这就说明我们同样可以很轻松地播放网络音乐哦。我们是不是可以做个网络音乐播放器呢？留给大家自己实现吧，核心代码如下：

```

private MediaPlayer mMediaPlayer;
Uri uri = Uri.parse("http://www.xxxx.com/xxx.mp3");
mMediaPlayer = MediaPlayer.create(this, uri);
mMediaPlayer.start();

```

7.3 播放视频

前面我们学习了如何播放音频文件，那么视频文件又如何播放呢？由于 Android 平台由 Google 自己封装、设计，提供的 Java Dalvik 在算法处理效率上无法与 C/C++ 或 ARM ASM 相提并论，在描述或移植一些本地语言的解码器上显得无能为力，目前整个平台仅支持 MP4 的 H.264、3GP 和 WMV 视频的解析。本节我们将在 Android 中实现一个 MP4 视频文件的播放。

Android 内置的 `VideoView` 类可以快速制作一个系统播放器，`VideoView` 主要用来显示一个视频文件，我们先来看看 `VideoView` 类的一些基本方法。如表 7-4 所示。

表 7-4 `VideoView` 类的常用方法

方 法	说 明
<code>getBufferPercentage</code>	得到缓冲的百分比
<code>getCurrentPosition</code>	得到当前播放的位置
<code>getDuration</code>	得到视频文件的时间
<code>isPlaying</code>	是否正在播放
<code>pause</code>	暂停
<code>resolveAdjustedSize</code>	调整视频显示大小
<code>seekTo</code>	指定播放位置
<code>setMediaController</code>	设置播放控制器模式（播放进度条）
<code>setOnCompletionListener</code>	当媒体文件播放完成时触发事件
<code>setOnErrorListener</code>	错误监听
<code>setVideoPath</code>	设置视频源路径
<code>setVideoURI</code>	设置视频源地址
<code>start</code>	开始播放

有了这些方法，我们要播放一个视频文件就很简单了（具体实现参见本书所附代码：第 7 章\Examples_07_03），运行效果如图 7-7 所示。

首先在布局文件中创建 VideoView 布局，并且创建几个按钮（Button）来实现对视频的操作，请参见本书所附代码：第 7 章\Examples_07_03\res\layout\main.xml。

当我们点击“装载”按钮时，将指定视频文件的路径，如下列代码所示：

```
/* 设置路径 */
videoView.setVideoPath("/sdcard/test.mp4");
/* 设置模式-播放进度条 */
videoView.setMediaController(new MediaController(Activity01.this));
videoView.requestFocus();
```

装载之后便可以通过 start、pause 方法来播放和暂停，具体实现见代码清单 7-3 所示。

代码清单 7-3 第 7 章\Examples_07_03\src\com\yarin\android\Examples_07_03\Activity01.java

```
public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        /* 创建 VideoView 对象 */
        final VideoView videoView = (VideoView) findViewById(R.id.VideoView01);

        /* 操作播放的三个按钮 */
        Button PauseButton = (Button) this.findViewById(R.id.PauseButton);
        Button LoadButton = (Button) this.findViewById(R.id.LoadButton);
        Button PlayButton = (Button) this.findViewById(R.id.PlayButton);

        /* 装载按钮事件 */
        LoadButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View arg0)
            {
                /* 设置路径 */
                videoView.setVideoPath("/sdcard/test.mp4");
                /* 设置模式-播放进度条 */
                videoView.setMediaController(new
                    MediaController(Activity01.this));
                videoView.requestFocus();
            }
        });

        /* 播放按钮事件 */
        PlayButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View arg0)
            {
                /* 开始播放 */
                videoView.start();
            }
        });
    }
}
```



图 7-7 在 Android 中播放 MP4 视频

```

        /* 暂停按钮 */
        PauseButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View arg0)
            {
                /* 暂停 */
                videoView.pause();
            }
        });
    }
}

```

扩展学习

上面我们通过系统自带的 VideoView 播放了一个 MP4 视频文件，但是我们知道 VideoView 类实际上继承自 SurfaceView 类，且实现了 MediaController.MediaPlayerControl 接口，那么我们就可以写一个用来播放视频文件的类，下面我们就来实现一个视频播放器，同样可以使用 SurfaceView 类来显示视频（具体实现参见本书所附代码：第 7 章\Examples_07_04）。首先在布局文件中需要创建一个 SurfaceView，请参见本书所附代码：第 7 章\Examples_07_04\res\layout\main.xml。

当然需要通过 SurfaceHolder 来控制 SurfaceView，因此需要通过 getHolder 方法来得到 SurfaceView 的 SurfaceHolder 对象。具体代码如代码清单 7-4 所示。

代码清单 7-4 第 7 章\Examples_07_04\src\com\yarin\android\Examples_07_04\Activity01.java

```

public class Activity01 extends Activity implements
    OnBufferingUpdateListener,
    OnCompletionListener,
    MediaPlayer.OnPreparedListener,
    SurfaceHolder.Callback{
    TAG = "Activity01";

    private static final String
    private int          mVideoWidth;
    private int          mVideoHeight;
    private MediaPlayer mMediaPlayer;
    private SurfaceView mPreview;
    private SurfaceHolder holder;
    private String path;
    public void onCreate(Bundle icicle)
    {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        mPreview = (SurfaceView) findViewById(R.id.surface);
        /* 得到 SurfaceHolder 对象 */
        holder = mPreview.getHolder();
        /* 设置回调函数 */
        holder.addCallback(this);
        /* 设置风格 */
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
    private void playVideo()
    {
        try
        {
            path = "/sdcard/test.mp4";
            /* 构建 MediaPlayer 对象 */
            mMediaPlayer = new MediaPlayer();
            /* 设置媒体文件路径 */

```

```

        mMediaPlayer.setDataSource(path);
        /* 设置通过 SurfaceView 来显示画面 */
        mMediaPlayer.setDisplay(holder);
        /* 准备 */
        mMediaPlayer.prepare();
        /* 设置事件监听 */
        mMediaPlayer.setOnBufferingUpdateListener(this);
        mMediaPlayer.setOnCompletionListener(this);
        mMediaPlayer.setOnPreparedListener(this);

mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    }
    catch (Exception e)
    {
        Log.e(TAG, "error: " + e.getMessage(), e);
    }
}

public void onPrepared(MediaPlayer mediaplayer)
{
    Log.d(TAG, "onPrepared called");
    mVideoWidth = mMediaPlayer.getVideoWidth();
    mVideoHeight = mMediaPlayer.getVideoHeight();
    if (mVideoWidth != 0 && mVideoHeight != 0)
    {
        /* 设置视频的宽度和高度 */
        holder.setFixedSize(mVideoWidth, mVideoHeight);
        /* 开始播放 */
        mMediaPlayer.start();
    }
}

/* 当 SurfaceHolder 创建时触发 */
public void surfaceCreated(SurfaceHolder holder)
{
    Log.d(TAG, "surfaceCreated called");
    playVideo();
}

/* 销毁 */
protected void onDestroy()
{
    super.onDestroy();
    if (mMediaPlayer != null)
    {
        mMediaPlayer.release();
        mMediaPlayer = null;
    }
}
}

```

7.4 录制歌曲

现在几乎每个手机都有录音功能，其用途也很广，比如我们需要记录某些重要的内容时，手头又没有笔之类的工具，这时我们就可以通过录音功能来将通话内容录制下来。而且我们通常可以将

自己的录音设置为个性铃声等。本节内容我们将学习在 Android 如何进行录音。

前面我们对 `MediaRecorder` 类有了简单了解,现在我们要实现录音功能就得使用 `MediaRecorder` 类。这里我们将实现一个简易的录音机,首先我们在界面上放置一个 `ListView` 来显示录音文件的列表、一个“开始”按钮、和一个“停止”按钮,如图 7-8 所示。当我们点击录音文件时就播放这段录音,如图 7-9 所示(具体实现参见本书所附代码:第 7 章\Examples_07_05)。

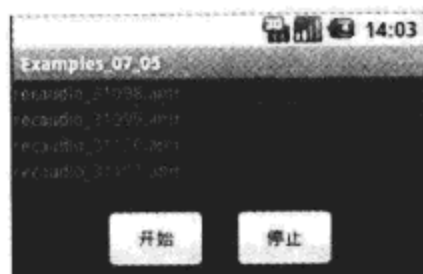


图 7-8 简易录音机界面

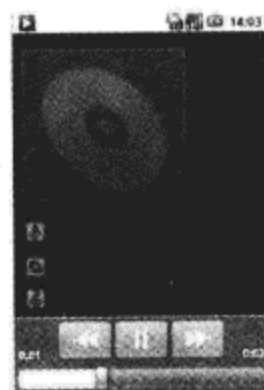


图 7-9 播放录音

当点击“开始”按钮后就构建 `MediaRecorder` 对象,并且设置声音的来源(`setAudioSource`)、输出文件的格式(`setOutputFormat`)、音频文件的编码(`setAudioEncoder`)、输出文件的路径(`setOutputFile`)等;然后准备开始录音(`prepare`),开始录音(`start`)。当点击“停止”按钮时,停止录音(`stop`),然后释放 `MediaRecorder` 对象(`release`),完成录音。具体代码如代码清单 7-5 所示。

代码清单 7-5 第 7 章\Examples_07_05\src\com\yarin\android\Examples_07_05\Activity01.java

```
public class Activity01 extends ListActivity
{
    /* 按钮 */
    private Button StartButton;
    private Button StopButton;
    /* 录制的音频文件 */
    private File mRecAudioFile;
    private File mRecAudioPath;
    /* MediaRecorder 对象 */
    private MediaRecorder mMediaRecorder;
    /* 录音文件列表 */
    private List<String> mMusicList = new ArrayList<String>();
    /* 临时文件的前缀 */
    private String strTempFile = "recaudio_";

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        /* 取得按钮 */
        StartButton = (Button) findViewById(R.id.StartButton);
        StopButton = (Button) findViewById(R.id.StopButton);

        /* 检测是否存在 SD 卡 */
        if (Environment.getExternalStorageState().equals(android.os.Environment.
            MEDIA_MOUNTED))
        {
            /* 得到 SD 卡的路径 */
            mRecAudioPath = Environment.getExternalStorageDirectory();
        }
    }
}
```



```

        /* 更新所有录音文件到 List 中 */
        musicList();
    }
    else
    {
        Toast.makeText(Activity01.this, "没有SD卡", Toast.
            LENGTH_LONG).show();
    }
    /* 开始按钮事件监听 */
    StartButton.setOnClickListener(new Button.OnClickListener()
    {
        @Override
        public void onClick(View arg0)
        {
            try
            {
                /* 创建录音文件 */
                mRecAudioFile = File.createTempFile
                    (strTempFile, ".amr", mRecAudioPath);
                /* 实例化 MediaRecorder 对象 */
                mMediaRecorder = new MediaRecorder();
                /* 设置麦克风 */

                mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
                /* 设置输出文件的格式 */

                mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
                /* 设置音频文件的编码 */

                mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
                /* 设置输出文件的路径 */

                mMediaRecorder.setOutputFile(mRecAudioFile.getAbsolutePath());
                /* 准备 */
                mMediaRecorder.prepare();
                /* 开始 */
                mMediaRecorder.start();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    });
    /* 停止按钮事件监听 */
    StopButton.setOnClickListener(new Button.OnClickListener()
    {
        @Override
        public void onClick(View arg0)
        {
            // TODO Auto-generated method stub
            if (mRecAudioFile != null)
            {
                /* 停止录音 */
                mMediaRecorder.stop();
                /* 将录音文件添加到 List 中 */
                mMusicList.add(mRecAudioFile.getName());
            }
        }
    });

```

```

        ArrayAdapter<String> musicList = new
        ArrayAdapter<String>(Activity01.this,
        R.layout.list, mMusicList);
        setListAdapter(musicList);
        /* 释放 MediaRecorder */
        mMediaRecorder.release();
        mMediaRecorder = null;
    }
}

});

}

/* 播放录音文件 */
private void playMusic(File file)
{
    Intent intent = new Intent();
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    intent.setAction(android.content.Intent.ACTION_VIEW);
    /* 设置文件类型 */
    intent.setDataAndType(Uri.fromFile(file), "audio");
    startActivity(intent);
}

/* 当我们点击列表时, 播放被点击的音乐 */
protected void onItemClick(ListView l, View v, int position, long id)
{
    /* 得到被点击的文件 */
    File playfile = new File(mRecAudioPath.getAbsolutePath() +
    File.separator + mMusicList.get(position));
    /* 播放 */
    playMusic(playfile);
}

/* 播放列表 */
public void musicList()
{
    // 取得指定位置的文件设置显示到播放列表
    File home = mRecAudioPath;
    if (home.listFiles(new MusicFilter()).length > 0)
    {
        for (File file : home.listFiles(new MusicFilter()))
        {
            mMusicList.add(file.getName());
        }
        ArrayAdapter<String> musicList = new
        ArrayAdapter<String>(Activity01.this, R.layout.list,
        mMusicList);
        setListAdapter(musicList);
    }
}

/* 过滤文件类型 */
class MusicFilter implements FilenameFilter
{
    public boolean accept(File dir, String name)
    {
        return (name.endsWith(".amr"));
    }
}

```

代码清单 7-5 中通过“Environment.getExternalStorageDirectory()”可以取得 SD 卡路径。最后

将录音保存到了 SD 卡, 图 7-10 是录音后保存在 SD 卡中的 amr 文件。
通过这个例子的学习, 我们可以得出要实现录音的一般步骤:

- ❑ 实例化 MediaRecorder: `mr=new MediaRecorder();`
- ❑ 初始化 mr: `mr.setAudioSource(MIC)/setVideoSource(CAMERA)`, 必须在配置 DataSource 之前调用;
- ❑ 配置 DataSource: 设置输出文件格式/路径, 编码器等;
- ❑ 准备录制: `mr.prepare();`
- ❑ 开始录制: `mr.start();`
- ❑ 停止录制: `mr.stop();`
- ❑ 释放资源: `mr.release();`

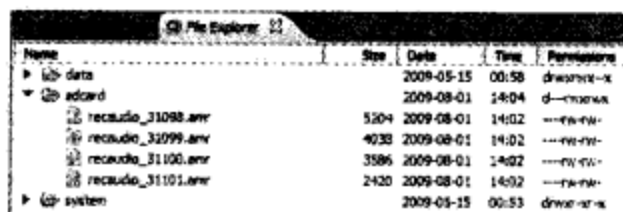


图 7-10 录音文件存储的位置

7.5 相机设置

现在的手机几乎都能实现照相机功能了, 而且在硬件的支持下像素也越来越高, 在现实生活中的用途也越来越广, 而在 Android 中专门提供了 Camera 来处理相机相关的事件, Camera 是一个专门用来连接和断开相机服务的类, Camera 下面包括如下几个事件:

- ❑ Camera.AutoFocusCallback: 自动调焦功能;
- ❑ Camera.ErrorCallback: 错误信息捕捉;
- ❑ Camera.Parameters: 相机的属性参数;
- ❑ Camera.PictureCallback: 拍照、产生图片时触发;
- ❑ Camera.PreviewCallback: 相机预览设置;
- ❑ Camera.ShutterCallback: 快门设置;
- ❑ Camera.Size: 图片的尺寸。

要在 Android 中使用相机服务很简单, Camera 没有构造方法, 我们要使用它直接通过 `open()` 方法来打开相机设备, 然后通过 `Camera.Parameters` 对相机的一些属性进行设置, 比如输出图片的格式、大小等等。使用 Camera 可进行的操作, 如表 7-5 所示。

表 7-5 Camera 类的方法

方 法	说 明
<code>autoFocus</code>	设置自动对焦
<code>getParameters</code>	得到相机的参数
<code>open</code>	启动相机服务
<code>release</code>	释放 Camera 服务
<code>setParameters</code>	设置预览的参数
<code>setPreviewDisplay</code>	设置预览
<code>startPreview</code>	开始预览
<code>stopPreview</code>	停止预览
<code>takePicture</code>	拍照

这里重点说明一下拍照的方法和使用，takePicture 方法要实现 3 个回调函数，分别是：Camera.ShutterCallback（快门）和两个 Camera.PictureCallback（图像数据）。这里我们在拍照之后要取得图像数据就需要实现 Camera.PictureCallback 的 onPictureTaken 方法。onPictureTaken 中第一个参数就是图像数据，第二个参数则是相机。

下面看一个通过相机拍照的例子。相机需要一个界面来预览当前拍摄的效果，这里可以使用 SurfaceView 类。先来看看在模拟器上的运行效果，如图 7-11 所示（具体实现参见本书所附代码：第 7 章 \Examples_07_06）。

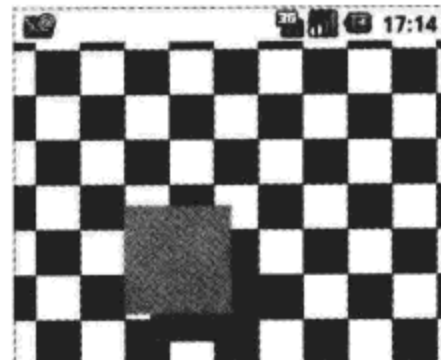


图 7-11 Camera 预览效果

下面我们来看具体如何使用相机服务预览效果，最后按导航键中键来拍照，将照片保存到 SD 卡中（当然首先要确认有 SD 卡插入哦）。如代码清单 7-6 所示。

代码清单 7-6 第 7 章 \Examples_07_06 \src \com \yarin \android \Examples_07_06 \Activity01.java

```
public class Activity01 extends Activity
{
    private Preview mPreview;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        //创建预览视图并设置它作为 Activity 要显示的内容
        mPreview = new Preview(this);
        setContentView(mPreview);
    }
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_DPAD_CENTER:
                mPreview.takePicture();
                break;
        }
        return true;
    }
}

/* Preview-显示 Preview */
class Preview extends SurfaceView implements SurfaceHolder.Callback
{
    SurfaceHolder mHolder;
    Camera mCamera;
    Bitmap CameraBitmap;

    Preview(Context context)
    {
        super(context);
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    /* SurfaceView 创建时触发 */
```

```

public void surfaceCreated(SurfaceHolder holder)
{
    /* 启动 Camera */
    mCamera = Camera.open();
    try
    {
        mCamera.setPreviewDisplay(holder);
    }
    catch (IOException exception)
    {
        /* 释放 mCamera */
        mCamera.release();
        mCamera = null;
    }
}

public void surfaceDestroyed(SurfaceHolder holder)
{
    /* 停止预览 */
    mCamera.stopPreview();
    mCamera = null;
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
    /* 构建 Camera.Parameters 对相机的参数进行设置 */
    Camera.Parameters parameters = mCamera.getParameters();
    /* 设置拍照的图片格式 */
    parameters.setPictureFormat(PixelFormat.JPEG);
    /* 设置 Preview 的尺寸 */
    parameters.setPreviewSize(320, 480);
    /* 设置图像分辨率 */
    //parameters.setPictureSize(320, 480);
    /* 设置相机采用 parameters */
    mCamera.setParameters(parameters);
    /* 开始预览 */
    mCamera.startPreview();
}

/* 拍照片 */
public void takePicture()
{
    if (mCamera != null)
    {
        mCamera.takePicture(null, null, jpegCallback);
    }
}

/* 拍照后输出图片 */
private PictureCallback jpegCallback = new PictureCallback()
{
    public void onPictureTaken(byte[] _data, Camera _camera)
    {
        CameraBitmap = BitmapFactory.decodeByteArray(_data, 0, _data.length);
        File myCaptureFile = new File("/sdcard/camera1.jpg");
        try
        {
            BufferedOutputStream bos = new BufferedOutputStream(new
                FileOutputStream(myCaptureFile));

```

```

        CameraBitmap.compress(Bitmap.CompressFormat.JPEG, 80, bos);
        bos.flush();
        bos.close();
        /* 将拍到的图片绘制出来 */
        Canvas canvas= mHolder.lockCanvas();
        canvas.drawBitmap(CameraBitmap, 0, 0, null);
        mHolder.unlockCanvasAndPost(canvas);
    }
    catch (Exception e)
    {
        e.getMessage();
    }
}
};
}

```

7.6 闹钟设置

闹钟在生活中最常见了，大家说不定天天都在用呢！在 Android 中可以通过 `AlarmManager` 来实现闹钟，`AlarmManager` 类是专门用来设定在某个指定的时间去完成指定的事件。`AlarmManager` 提供了访问系统警报的服务，只要在程序中设置了警报服务，`AlarmManager` 就会通过 `onReceive()` 方法去执行这些事件，就算系统处于待机状态，同样不会影响运行。可以通过 `Context.getSystemService` 方法来获得该服务。`AlarmManager` 中的方法很少，如表 7-6 所示。

表 7-6 `AlarmManager` 的方法

方 法	说 明
<code>cancel</code>	取消 <code>AlarmManager</code> 服务
<code>set</code>	设置 <code>AlarmManager</code> 服务
<code>setInexactRepeating</code>	设置不精确周期
<code>setRepeating</code>	设置精确周期
<code>setTimeZone</code>	设置时区

要实现闹钟，首先需要创建一个继承自 `BroadcastReceiver` 的类，实现 `onReceive` 方法来接收这个 `Alarm` 服务，然后通过建立 `Intent` 和 `PendingIntent` 连接来调用 `Alarm` 组件，示例运行效果如图 7-12 所示。当我们点击“设置闹钟”按钮时，通过 `TimePickerDialog` 来设置时间，如图 7-13 所示，当时间到我们指定的时间后 `onReceive` 方法接收到 `Alarm` 服务后的界面如图 7-14 所示（具体实现参见本书所附代码：第 7 章\Examples_07_07）。

首先看看我们实现的接收 `Alarm` 服务的 `AlarmReceiver` 类，很简单，就是在收到消息后用 `Toast` 来提示用户，如代码清单 7-7 所示。

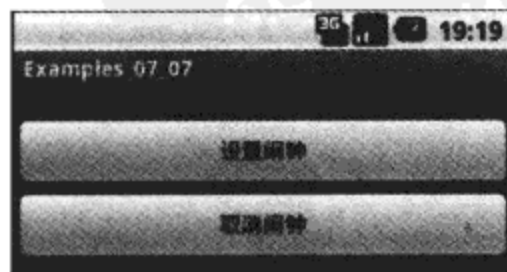


图 7-12 未设置闹钟之前

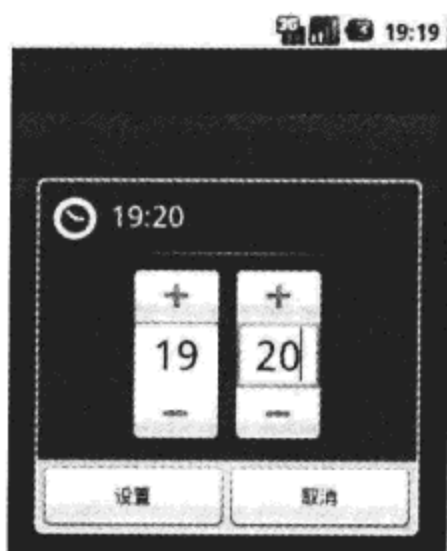


图 7-13 设置闹钟的时间



图 7-14 闹钟时间到之后的提示

代码清单 7-7 第 7 章\Examples_07_07\src\com\yarin\android\Examples_07_07\AlarmReceiver.java

```
public class AlarmReceiver extends BroadcastReceiver
{
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context, "你设置的闹钟时间到了", Toast.LENGTH_LONG).show();
    }
}
```

由于使用了 `BroadcastReceiver` 服务，因此需要在 `AndroidManifest.xml` 中进行声明，代码如下：

```
<receiver android:name=".AlarmReceiver" android:process=":remote" />
```

然后就只需要对“设置闹钟”、“取消闹钟”的事件进行监听了，代码如代码清单 7-8 所示。

代码清单 7-8 第 7 章\Examples_07_07\src\com\yarin\android\Examples_07_07\Activity01.java

```
public class Activity01 extends Activity
{
    Button mButton1;
    Button mButton2;

    TextView mTextView;

    Calendar calendar;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        calendar=Calendar.getInstance();

        mTextView=(TextView) findViewById(R.id.TextView01);
        mButton1=(Button) findViewById(R.id.Button01);
        mButton2=(Button) findViewById(R.id.Button02);

        mButton1.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                calendar.setTimeInMillis(System.currentTimeMillis());
            }
        });
    }
}
```



```

int mHour=calendar.get(Calendar.HOUR_OF_DAY);
int mMinute=calendar.get(Calendar.MINUTE);
new TimePickerDialog(Activity01.this,
    new TimePickerDialog.OnTimeSetListener()
    {
        public void onTimeSet(TimePicker view,int hourOfDay,int minute)
        {
            calendar.setTimeInMillis(System.currentTimeMillis());
            calendar.set(Calendar.HOUR_OF_DAY,hourOfDay);
            calendar.set(Calendar.MINUTE,minute);
            calendar.set(Calendar.SECOND,0);
            calendar.set(Calendar.MILLISECOND,0);
            /* 建立 Intent 和 PendingIntent 来调用目标组件 */
            Intent intent = new Intent(Activity01.this,AlarmReceiver.class);
            PendingIntent pendingIntent=PendingIntent.getBroadcast
                (Activity01.this,0, intent, 0);
            AlarmManager am;
            /* 获取闹钟管理的实例 */
            am = (AlarmManager)getSystemService(ALARM_SERVICE);
            /* 设置闹钟 */
            am.set(AlarmManager.RTC_WAKEUP,calendar.getTimeInMillis(), pendingIntent);
            /* 设置周期闹钟 */
            am.setRepeating(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()+
                (10*1000), (24*60*60*1000), pendingIntent);
            String tmpS="设置闹钟时间为"+format(hourOfDay)+":"+format(minute);
            mTextView.setText(tmpS);
        }
    },mHour,mMinute,true).show();
});

mButton2.setOnClickListener(new View.OnClickListener()
{
    public void onClick(View v)
    {
        Intent intent = new Intent(Activity01.this, AlarmReceiver.class);
        PendingIntent pendingIntent=PendingIntent.getBroadcast
            (Activity01.this,0, intent, 0);
        AlarmManager am;
        /* 获取闹钟管理的实例 */
        am =(AlarmManager)getSystemService(ALARM_SERVICE);
        /* 取消 */
        am.cancel(pendingIntent);
        mTextView.setText("闹钟已取消!");
    }
});

/* 格式化字符串(7:3->07:03) */
private String format(int x)
{
    String s = "" + x;
    if (s.length() == 1)
        s = "0" + s;
    return s;
}
}

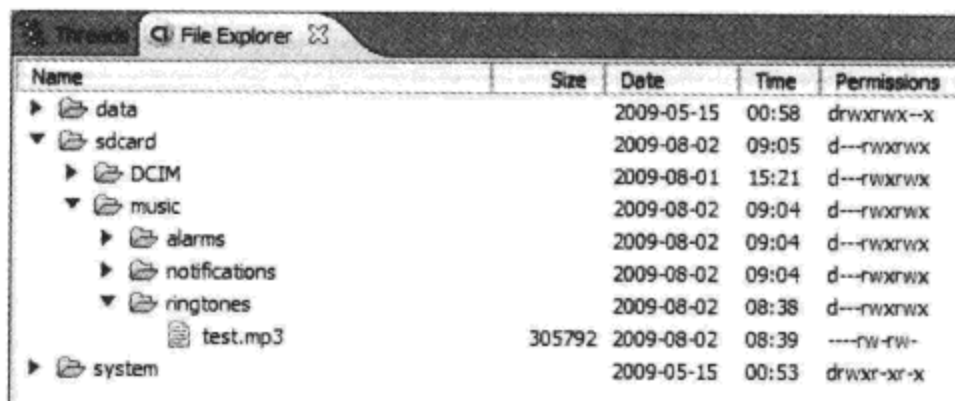
```

7.7 铃声设置

可以从网上下载很多自己喜欢的铃声，并设置成我们手机默认的铃声。Android 提供了 `RingtoneManager` 类专门来操作各种铃声，比如常见的来电铃声、闹钟铃声以及一些警告或通知铃声。Android 自带的系统铃声都放置在“`/system/medio/audio/`”文件夹中，而自己下载的铃声一般都放置在 SD 卡中，那么就需要在 SD 卡下面建立如下文件目录：

- ❑ `/sdcard/music/ringtones`：一般的铃声（比如来电铃声）；
- ❑ `/sdcard/music/alarms`：闹钟铃声；
- ❑ `/sdcard/music/notifications`：警告或通知铃声。

然后，将下载的铃声按自己的需要放置到这些文件夹中，如图 7-15 所示，我们在 `ringtones` 文件夹中放置了一个 `test.mp3` 来作为铃声。



Name	Size	Date	Time	Permissions
data		2009-05-15	00:58	drwxrwx--x
sdcard		2009-08-02	09:05	d--rwxrwx
DCIM		2009-08-01	15:21	d--rwxrwx
music		2009-08-02	09:04	d--rwxrwx
alarms		2009-08-02	09:04	d--rwxrwx
notifications		2009-08-02	09:04	d--rwxrwx
ringtones		2009-08-02	08:38	d--rwxrwx
test.mp3	305792	2009-08-02	08:39	---rwxrwx
system		2009-05-15	00:53	drwxr-xr-x

图 7-15 SD 卡中铃声目录结构

现在我们要做一个铃声设置的应用，首先需要设置一个界面来让用户选择需要设置的铃声的类型，这里我们通过 3 个按钮（`Button`）分别代表来电铃声、闹钟铃声、通知铃声，如图 7-16 所示。当点击这些按钮后，分别调用系统的铃声设置界面，如图 7-17、图 7-18、图 7-19 所示，当设置完成之后又返回如图 7-16 所示界面，这里就需要使用 `startActivityForResult` 来开启一个 `Activity`，并且 3 个按钮分别设置不同的铃声类型：来电铃声（`TYPE_RINGTONE`）、闹钟铃声（`TYPE_ALARM`）、通知铃声（`TYPE_NOTIFICATION`）（具体实现参见本书所附代码：第 7 章 \Examples_07_08）。

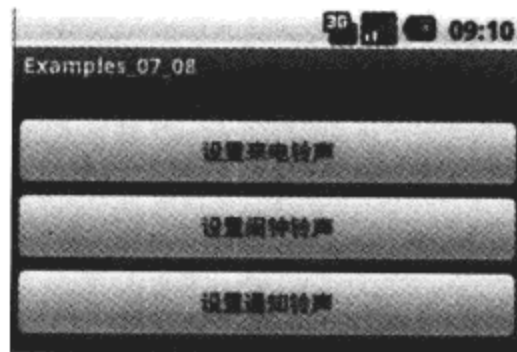


图 7-16 铃声设置界面

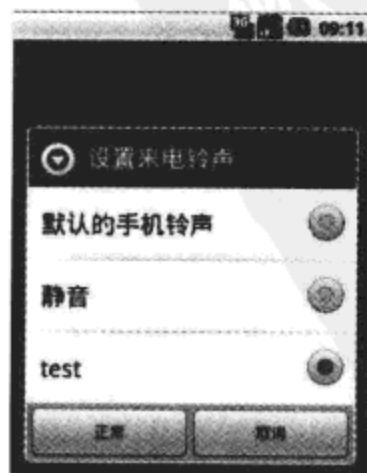


图 7-17 设置来电铃声



图 7-18 设置闹钟铃声



图 7-19 设置通知铃声

RingtoneManager 类提供的操作铃声的方法如表 7-7 所示。

表 7-7 RingtoneManager 类的常用方法

方 法	说 明
getActualDefaultRingtoneUri	取得指定类型当前默认的铃声
getCursor	返回所有可用铃声的游标
getDefaultType	得到指定 URI 默认的铃声类型
getDefaultUri	返回指定类型的默认铃声的 URI
getRingtone	返回指定铃声的 URI
getRingtone	获得当前游标所指定的铃声
getRingtonePosition	得到我们指定铃声的位置
getRingtoneUri	获得得到指定位置的铃声的 URI
getValidRingtoneUri	得到一个可用铃声的 URI
isDefault	得到指定的 URI 是否是默认的铃声
setActualDefaultRingtoneUri	设置默认的铃声

当选择了需要设置为铃声的音乐后，系统会调用 `onActivityResult` 方法来处理我们所进行的设置，因此需要重写 `onActivityResult`，并根据设置的不同类型的铃声来告诉系统我们的设置。具体实现参见代码清单 7-9。

代码清单 7-9 第 7 章\Examples_07_08\src\com\yarin\android\Examples_07_08\Activity01.java

```
public class Activity01 extends Activity
{
    /* 3 个按钮 */
    private Button mButtonRingtone;
    private Button mButtonAlarm;
    private Button mButtonNotification;

    /* 自定义的类型 */
    public static final int ButtonRingtone = 0;
    public static final int ButtonAlarm = 1;
    public static final int ButtonNotification = 2;
    /* 铃声文件夹 */
    private String strRingtoneFolder = "/sdcard/music/ringtones";
    private String strAlarmFolder = "/sdcard/music/alarms";
    private String strNotificationFolder = "/sdcard/music/notifications";
```

```

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

    mButtonRingtone = (Button) findViewById(R.id.ButtonRingtone);
    mButtonAlarm = (Button) findViewById(R.id.ButtonAlarm);
    mButtonNotification = (Button) findViewById(
        R.id.ButtonNotification);
    /* 设置来电铃声 */
    mButtonRingtone.setOnClickListener(new Button.OnClickListener()
    {
        public void onClick(View arg0)
        {
            if (bFolder(strRingtoneFolder))
            {
                //打开系统铃声设置
                Intent intent = new Intent(Ringtone
                    Manager.ACTION_RINGTONE_PICKER);
                //类型为来电 RINGTONE

                intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
                    RingtoneManager.TYPE_RINGTONE);

                //设置显示的 title

                intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设置来电铃声");
                //当设置完成之后返回到当前的 Activity
                startActivityForResult(intent,
                    ButtonRingtone);
            }
        }
    });
    /* 设置闹钟铃声 */
    mButtonAlarm.setOnClickListener(new Button.OnClickListener()
    {
        @Override
        public void onClick(View arg0)
        {
            if (bFolder(strAlarmFolder))
            {
                //打开系统铃声设置
                Intent intent = new Intent(Ringtone
                    Manager.ACTION_RINGTONE_PICKER);
                //设置铃声类型和 title

                intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE, RingtoneManager.
                    TYPE_ALARM);

                intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设置闹钟铃声");
                //当设置完成之后返回到当前的 Activity
                startActivityForResult(intent,
                    ButtonAlarm);
            }
        }
    });
    /* 设置通知铃声 */

```

```

mButtonNotification.setOnClickListener(new Button.OnClickListener()
{
    @Override
    public void onClick(View arg0)
    {
        if (bFolder(strNotificationFolder))
        {
            //打开系统铃声设置
            Intent intent = new Intent(Ringtone
            Manager.ACTION_RINGTONE_PICKER);
            //设置铃声类型和title

            intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
            RingtoneManager.TYPE_NOTIFICATION);

            intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设置通知铃声");
            //当设置完成之后返回到当前的 Activity
            startActivityForResult(intent,
            ButtonNotification);
        }
    }
});
}
/* 当设置铃声之后的回调函数 */
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    // TODO Auto-generated method stub
    if (resultCode != RESULT_OK)
    {
        return;
    }
    switch (requestCode)
    {
        case ButtonRingtone:
            try
            {
                //得到我们选择的铃声
                Uri pickedUri = data.getParcelable
                Extra(RingtoneManager.EXTRA_
                RINGTONE_PICKED_URI);
                //将我们选择的铃声设置成为默认
                if (pickedUri != null)
                {
                    RingtoneManager.setActualDefaultRingtoneUri(Activity01.this,
                    RingtoneManager.TYPE_RINGTONE, pickedUri);
                }
            }
            catch (Exception e)
            {
            }
            break;
        case ButtonAlarm:
            try
            {
                //得到我们选择的铃声
                Uri pickedUri = data.getParcelableExtra
                (RingtoneManager.EXTRA_RINGTONE_PICKED_URI);
            }
            catch (Exception e)
            {
            }
            break;
    }
}

```

```

//将我们选择的铃声设置成为默认
if (pickedUri != null)
{
    RingtoneManager.setActualDefaultRingtoneUri(Activity01.this,
    RingtoneManager.TYPE_ALARM, pickedUri);
}
}
catch (Exception e)
{
}
break;
case ButtonNotification:
    try
    {
        //得到我们选择的铃声
        Uri pickedUri = data.getParcelableExtra
        (RingtoneManager.EXTRA_RINGTONE_PICKED_URI);
        //将我们选择的铃声设置成为默认
        if (pickedUri != null)
        {
            RingtoneManager.setActualDefaultRingtoneUri(Activity01.this,
            RingtoneManager.TYPE_NOTIFICATION, pickedUri);
        }
    }
    catch (Exception e)
    {
    }
    break;
}
super.onActivityResult(requestCode, resultCode, data);
}
//检测是否存在指定的文件夹
//如果不存在则创建
private boolean bFolder(String strFolder)
{
    boolean btmp = false;
    File f = new File(strFolder);
    if (!f.exists())
    {
        if (f.mkdirs())
        {
            btmp = true;
        }
        else
        {
            btmp = false;
        }
    }
    else
    {
        btmp = true;
    }
    return btmp;
}
}

```

因为我们在程序中操作了系统文件夹，所以这里需要在 `AndroidManifest.xml` 文件中声明其权限，代码如下：

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

7.8 小结

本章首先介绍了 Android 平台多媒体开发的框架，让大家对 Open Core 有了更深入的了解，然后介绍了 Google 封装的 `MediaPlayer`、`MediaRecorder` 来实现对多媒体的操作，最后分别从音乐播放、视频播放、录音、照相、闹钟、铃声等几个具体模块进行了实现，通过每个示例程序，使大家对 Android 中多媒体开发有进一步的了解。当然，1.5 以后的 SDK 开始支持流媒体的播放了，大家可以自己写流媒体然后在 Android 播放，使得多媒体开发更具特色。随着手机硬件的提高以及 3G 技术的普及，手机音乐、手机购物、手机电视、视频通话等功能将逐渐进入人们的生活中，这就需要大家对多媒体开发有更深入的研究，以满足用户的需求；这也必然是一个具有很大发展空间的技术。希望大家好好把握。

资源分享
PDG

手机的确给我们带来了不少方便, 1995 年问世的第一代 (1G) 模拟制式手机还只能进行语音通话; 1996~1997 年出现的第二代 (2G) GSM、CDMA 等数字制式手机便增加了接收数据 (如接收电子邮件或网页) 等功能; 而 2009 年诞生的 3G, 其功能更是让人眼花缭乱, 比如高速的无线宽带上网、视频通话、无线搜索、手机音乐、手机网游等。无线网络发展的速度也非常迅猛, 有了无线网络的支持, 我们就不必受时间和空间的限制, 可以随时随地进行数据交换, 浏览 Internet, 第一时间获得新闻。随着人们知识水平的提高, 生活圈也越来越广, 人们更需要网络的帮助来处理一些事务, 比如手机炒股、手机证券、手机银行、手机地图等等, 而在 Android 中, 掌握了网络通信便可以开发出这些优秀的网络应用。

8.1 网络通信基础

无线网络的产生为我们提供了不少的方便, 有了无线网络我们几乎可以从任何地方接入网络: 办公桌、会议室、咖啡厅、企业园区或校园中的办公楼。同时无线网络可以为员工提供最大限度的灵活性、生产率、效率, 大幅度地加强他们与同事、业务伙伴和客户之间的合作。如今, 每一天大约有 25 万人成为新的无线用户, 全球范围内的无线用户数量已经超过 2 亿, 这些人包括大学教授、仓库管理员、护士、商店负责人、办公室经理和卡车司机。他们使用无线技术的方式和他们自身的工作一样都在不断地更新。无线网络技术涵盖的范围很广, 既包括允许用户建立远距离无线连接的全球语音和数据网络, 也包括为近距离无线连接进行优化的红外线技术及射频技术。通常用于无线网络的设备包括便携式计算机、台式计算机、手持计算机、个人数字助理 (PDA)、移动电话、笔式计算机和寻呼机。无线技术用于多种实际用途, 例如, 手机用户可以使用移动电话查看电子邮件。使用便携式计算机的旅客可以通过安装在机场、火车站和其他公共场所的基站连接到 Internet。在家中, 用户可以连接桌面设备来同步数据和发送文件。因此, 我们要在 Android 上进行网络开发, 就先来了解一下无线网络通信的相关技术。

8.1.1 无线网络技术

所谓无线网络, 即采用无线传输媒介 (如无线电波、红外线等) 的网络。既包括允许用户建立远距离无线连接的全球语音和数据网络, 也包括为近距离无线连接进行优化的红外线技术及射频技术。无线网络与有线网络的用途十分类似, 最大的不同在于传输媒介的不同, 利用无线电技术取代网线。无线网络可以和有线网络互为备份。

1. 无线网络通信标准

为了降低成本、保证互操作性并促进无线技术的广泛应用,许多组织(例如电气电子工程师协会(IEEE)、Internet 工程任务组(IETF)、无线以太网兼容性联盟(WECA)和国际电信联盟(ITU))都参与了若干主要的标准化工作。例如,IEEE 工作组正在定义如何将信息从一台设备传送到另一台设备(例如,是使用无线电波还是使用红外光波),以及怎样、何时使用传输介质进行通信。在开发无线网络标准时,有些组织(如 IEEE)着重于电源管理、带宽、安全性和其他无线网络特有的问题。在开发无线网络之前定义标准非常重要。

2. 无线网络类型

与有线网络一样,无线网络可根据数据传输的距离分为几种不同类型。根据国际上所采用的通信技术种类可将无线网络划分为无线广域网、无线城域网、无线局域网、无线个域网、低速率无线个域网。

□ 无线广域网(WWAN)。WWAN 技术可使用户通过远程公用网络或专用网络建立无线网络连接。通过使用由无线服务提供商负责维护的若干天线基站或卫星系统,这些连接可以覆盖广大的地理区域,例如若干城市或者国家(地区)。虽然 3G 时代已经开始,目前的 WWAN 技术还主要为第二代(2G)系统。2G 系统主要包括移动通信全球系统(GSM)、蜂窝式数字分组数据(CDPD)和码分多址(CDMA)。第三代(3G)技术将执行全球标准,并提供全球漫游功能。在新一代技术(3G)刚推出市场之后,更高的技术应用已经在实验室进行研发。目前日本的 NTT DoCoMo 公司已经表示,4G 通信的试验网络已经部署在公司的神奈川县横须贺研发工业园内,该网络集结了试验基站和移动终端,同时 NTT DoCoMo 公司还表示,4G 通信服务将于 2010 年推出,网络的下载速度可以达到 100Mbps,上载速度为 20Mbps。我们期待 4G 带来更美好的生活。

□ 无线城域网(WMAN)。WMAN 技术使用户可以在城区的多个场所之间创建无线连接(例如,在一个城市或大学校园的多个办公楼之间),而不必花费高昂的费用铺设光缆、铜质电缆和租用线路。此外,当有线网络的主要租赁线路不能使用时,WMAN 还可以作为备用网络使用。WMAN 使用无线电波或红外光波传送数据。用户对于高速 Internet 接入的宽带无线接入网络的需求量正日益增长。

□ 无线局域网(WLAN)。WLAN 技术可以使用户在本地创建无线连接(例如在公司或校园的大楼里,或在某个公共场所,如机场)。WLAN 可用于临时办公室或其他无法大范围布线的场所,或者用于增强现有的 LAN,使用户可以在不同时间、不同地方工作。WLAN 以两种不同方式运行。在基础结构 WLAN 中,无线站(具有无线网卡或外置调制解调器的设备)连接到无线接入点,后者在无线站与现有网络中枢之间起桥梁作用。在点对点(临时)WLAN 中,有限区域(例如会议室)内的几个用户可以在不需要访问网络资源时建立临时网络,而无需使用接入点。

□ 无线个域网(WPAN)。WPAN 技术使用户能够为个人操作空间(POS)设备(如 PDA、移动电话和笔记本电脑等)创建临时无线通信。POS 指的是以个人为中心、最大距离为 10 米的一个空间范围。目前,两个主要的 WPAN 技术是 Bluetooth 和红外线。Bluetooth 是一种电缆替代技术,可以在 30 英尺以内使用无线电波传送数据。Bluetooth 数据可以穿过墙壁、口袋和公文包进行传输。如果要更近距离(一米以内)连接设备,用户还可以创建红外连接。

- 低速率无线个域网 (LR-WPAN)。LR-WPAN 适用于工业监测、办公和家庭自动化以及农作物监测等。在工业应用方面, 主要用于建立传感器网络、紧急状况监测、机器检测; 在办公和家庭自动化方面, 用于提供无线办公解决方案, 建立类似传感器的疲劳程度监测系统, 用无线替代有线连接 VCR (盒式磁带像机)、PC 外设、游戏机、安全系统、照明和空调系统; 在农作物监测方面, 用于建立数千个 LR-WPAN 装置构成的网状网, 收集土地信息和气象信息, 农民利用这些信息可获取较高的农作物产量。

3. 3G

3G 指第三代移动通信技术。相对第一代模拟制式手机和第二代 GSM、CDMA 等数字手机, 3G 是将无线通信与国际互联网等多媒体通信结合的新一代移动通信系统, 支持高速数据传输的蜂窝移动通信技术。3G 服务能够同时传送声音 (通话) 及数据信息 (电子邮件、即时通信等), 代表特征是提供高速数据业务。它与 2G 的主要区别是在传输声音和数据时速度上的提升, 它能够在全球范围内更好地实现无线漫游, 并处理图像、音乐、视频流等多种媒体形式, 提供包括网页浏览、电话会议、电子商务等多种信息服务, 同时也要考虑与已有第二代系统的良好兼容性。为了提供这种服务, 无线网络必须能够支持不同的数据传输速度, 也就是说在室内、室外和行车的环境中能够分别支持至少 2Mbps、384kbps 以及 144kbps 的传输速度 (此数值根据网络环境会发生变化)。

3G 共有 3 个标准, 它们分别是 WCDMA (欧洲版)、CDMA2000 (美国版) 和 TD-SCDMA (中国版)。

WCDMA 全称为 Wideband CDMA, 也称为 CDMA Direct Spread, 意为宽频分码多重存取, 这是基于 GSM 网发展出来的 3G 技术规范, 是欧洲提出的宽带 CDMA 技术, 它与日本提出的宽带 CDMA 技术基本相同, 目前正在进一步融合。

CDMA2000 是由窄带 CDMA (CDMA IS95) 技术发展而来的宽带 CDMA 技术, 也称为 CDMA Multi-Carrier, 它由美国高通北美公司主导提出, 摩托罗拉、Lucent 和后来加入的韩国三星都有参与, 韩国现在成为该标准的主导者。

TD-SCDMA 全称为 Time Division - Synchronous CDMA (时分同步 CDMA), 该标准是由中国大陆独自制定的 3G 标准, 1999 年 6 月 29 日, 中国原邮电部电信科学技术研究院 (大唐电信) 向 ITU 提出, 但技术发明始祖于西门子公司, TD-SCDMA 具有辐射低的特点, 被誉为绿色 3G。

4. Android 的 3G 时代

随着 3G 时代的来临, 无论是上网、娱乐, 还是办公、学习, 智能手机将是用户的首选工具。而 Android 是一个以 Google 为首的由 30 多家科技公司和手机公司组成的开放手机联盟。Android 在使用目前已经非常成熟的 Java 作为主要开发语言的同时, 也支持 C/C++、常用的脚本语言以及 Simple 语言, 所以越来越多的开发者加入进来开发自己的应用。当然 Android 手机也受到广大用户的喜爱。Android 的另一个优势是丰富的应用程序, 特别是和网络应用集成紧密, 使得 3G 手机除了可以进行高质量通话外, 还给我们带来更好的 3G 体验 (快速上网、移动定位、可视电话、视频互动游戏、手机钱包、电子购物、家庭监控等)。下面我们将进入 Android 的网络世界。

8.1.2 Android 网络基础

前面我们已经看到了 Android 出色的界面设计和强大的数据管理功能, 其实 Android 在网络通

信方面也非常优秀，我们可以很轻松地使用 Android 自带的浏览器来访问网页，如图 8-1 所示，还可以通过自带的电子邮件程序来收取邮件，如图 8-2 所示。Android 平台浏览器采用了 Webkit 引擎，这款名为 Chrome Lite 的 Web 浏览器拥有强大扩展特性，每个开发者都可以编写自己的插件，使得浏览器的功能更加完美。



图 8-1 Android 自带浏览器访问网页



图 8-2 Android 自带电子邮件客户端

Android 基于 Linux 内核，它包含一组优秀的联网功能，当然这些只是 Android 自带的一些功能，它们是开源软件，大家可以拿来学习研究。目前，Android 平台有 3 种网络接口可以使用，它们分别是：`java.net.*`（标准 Java 接口）、`org.apache`（Apache 接口）和 `android.net.*`（Android 网络接口）。下面我们分别来简单介绍一下这些接口的功能及其作用。

1. 标准 Java 接口

`java.net.*`（标准 Java 接口）提供与联网有关的类，包括流和数据包套接字、Internet 协议、常见 HTTP 处理。比如：创建 URL 以及 `URLConnection` / `HttpURLConnection` 对象、设置连接参数、连接到服务器、向服务器写数据、从服务器读取数据等通信。下面是常见的使用 `java.net` 包的 HTTP 例子，如代码清单 8-1 所示。

代码清单 8-1 使用 `java.net.*` 包连接网络

```
//...
try
{
    //定义地址
    URL url = new URL("http://www.google.com");
    //打开连接
    HttpURLConnection http = (HttpURLConnection) url.openConnection();
    //得到连接状态
    int nRC = http.getResponseCode();
    if (nRC == HttpURLConnection.HTTP_OK)
    {
        //取得数据
        InputStream is = http.getInputStream();
        //处理数据...
    }
}
```

```

}
catch (Exception e)
{
}
//...

```

注意 由于是连接网络，不免会出现一些异常，所以必须处理这些异常。

2. Apache 接口

HTTP 协议可能是现在 Internet 上使用得最多、最重要的通信协议了，越来越多的 Java 应用程序需要通过 HTTP 协议来访问网络资源。虽然在 JDK 的 `java.net` 包中已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，JDK 库本身提供的功能远远不够。这时就需要 Android 提供的 Apache HttpClient 了。它是一个开源项目，功能更加完善，为客户端的 HTTP 编程提供高效、最新、功能丰富的工具包支持。Android 平台引入了 Apache HttpClient 的同时还提供了对它的一些封装和扩展，例如设置缺省的 HTTP 超时和缓存大小等。Android 使用的是目前最新的 HttpClient 4.0(org.apache.http.*)，可以将 Apache 视为目前流行的开源 Web 服务器，主要包括创建 HttpClient 以及 Get/Post、HttpRequest 等对象，设置连接参数，执行 HTTP 操作，处理服务器返回结果等功能。下面是一个使用 `android.net.http.*` 包的例子，如代码清单 8-2 所示。

代码清单 8-2 使用 `android.net.http.*` 连接网络

```

//...
try
{
    //创建 HttpClient
    //这里使用 DefaultHttpClient 表示默认属性
    HttpClient hc = new DefaultHttpClient();
    //HttpGet 实例
    HttpGet get = new HttpGet("http://www.google.com");
    //连接
    HttpResponse rp = hc.execute(get);
    if (rp.getStatusLine().getStatusCode() == HttpStatus.SC_OK)
    {
        InputStream is = rp.getEntity().getContent();
        //处理数据...
    }
}
catch (IOException e)
{
}
//...

```

3. Android 网络接口

`android.net.*` 包实际上是通过封装 Apache 中 HttpClient 来实现的一个 HTTP 编程接口，同时还提供了 HTTP 请求队列管理以及 HTTP 连接池管理，以提高并发请求情况下（如转载网页时）的处理效率，除此之外还有网络状态监视等接口、网络访问的 Socket、常用的 Uri 类以及有关 WiFi 相关的类等等。下面我们看最简单的 Socket 连接，如代码清单 8-3 所示。

代码清单 8-3 Android 中的 Socket 连接

```

//...
try

```



```

{
    //IP 地址
    InetAddress inetAddress = InetAddress.getByName("192.168.1.110");
    //端口
    Socket client = new Socket(inetAddress, 61203, true);
    //取得数据
    InputStream in = client.getInputStream();
    OutputStream out = client.getOutputStream();
    // 处理数据.....
    out.close();
    in.close();
    client.close();
}
catch (UnknownHostException e)
{
}
catch (IOException e)
{
}
}
//...

```

8.2 HTTP 通信

HTTP (Hyper Text Transfer Protocol, 超文本传输协议) 用于传送 WWW 方式的数据。HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求, 请求头包含了请求的方法、URI、协议版本, 以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。服务器以一个状态行作为响应, 响应的内容包括消息协议的版本、成功或者错误编码, 还包含服务器信息、实体元信息以及可能的实体内容。它是一个属于应用层的面向对象的协议, 由于其简洁、快速, 它适用于分布式超媒体信息系统。它于 1990 年提出, 经过几年的使用与发展, 得到不断完善和扩展。许多 HTTP 通信是由一个用户代理初始化的, 并且包括一个申请在源服务器上资源的请求。最简单的情况可能是在用户代理和服务器之间通过一个单独的连接来完成。在 Internet 上, HTTP 通信通常发生在 TCP/IP 连接之上, 缺省端口是 TCP 80, 但其他的端口也是可用的。这并不预示着 HTTP 协议在 Internet 或其他网络的其他协议之上才能完成, HTTP 只预示着一个可靠的传输。Android 提供了 `URLConnection` 和 `HttpClient` 接口来开发 HTTP 程序, 本节我们将分别介绍这两种方式。

8.2.1 HttpURLConnection 接口

HTTP 通信中使用最多的就是 Get 和 Post, Get 请求可以获取静态页面, 也可以把参数放在 URL 字符串后面, 传递给服务器。Post 与 Get 的不同之处在于 Post 的参数不是放在 URL 字符串里面, 而是放在 http 请求数据中。`URLConnection` 是 Java 的标准类, 继承自 `URLConnection` 类, `URLConnection` 与 `URLConnection` 都是抽象类, 无法直接实例化对象。其对象主要通过 URL 的 `openConnection` 方法获得, 创建一个 `URLConnection` 连接的代码如下所示:

```

URL url = new URL("http://www.google.com/");
URLConnection urlConn = (URLConnection) url.openConnection();

```

`openConnection` 方法只创建 `URLConnection` 或者 `URLConnection` 实例, 但是并不进行真正

的连接操作。并且，每次 `openConnection` 都将创建一个新的实例。因此，在连接之前我们可以对其一些属性进行设置，比如超时时间等。下面是对 `HttpURLConnection` 实例的属性设置：

```
//设置输入（输出）流
connection.setDoOutput(true);
connection.setDoInput(true);
//设置方式为 POST
connection.setRequestMethod("POST");
//Post 请求不能使用缓存
connection.setUseCaches(false);
```

当然在连接完成之后可以关闭这个连接，代码如下：

```
//关闭 HttpURLConnection 连接
urlConn.disconnect();
```

有了这些基础，我们将分别使用 `Get` 和 `Post` 方式来获取一个网页内容。首先，我们在服务器上建立一个 `http1.jsp` 文件，见代码清单 8-4 所示。

代码清单 8-4 `http1.jsp`

```
<HTML>
<HEAD>
<TITLE>Http Test</TITLE>
</HEAD>
<BODY>
<%
out.println("<h1>HTTP TEST<br>http test</h1>");
%>
</BODY>
</HTML>
```

`http1.jsp` 在浏览器中的访问效果如图 8-3 所示。

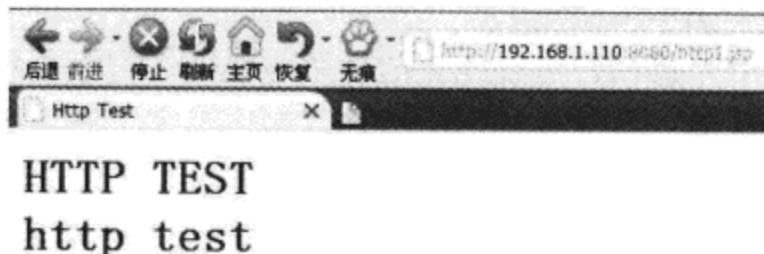


图 8-3 `http1.jsp` 效果

再创建一个使用 `Get` 和 `Post` 来传递参数的网页 `httpget.jsp`，如代码清单 8-5 所示。

代码清单 8-5 `httpget.jsp`

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<HTML>
<HEAD>
<TITLE>Http Test</TITLE>
</HEAD>
<BODY>
<%
String type = request.getParameter("par");
String result = new String(type.getBytes("iso-8859-1"), "gb2312");
out.println("<h1>parameters:"+result+"</h1>");
%>
</BODY>
</HTML>
```


httpget.jsp 中要求我们传递一个 par 参数，然后在网页中显示 “parameters: par”，在浏览器中我们输入 “httpget.jsp??par=abcdefg” 来访问，效果如图 8-4 所示。

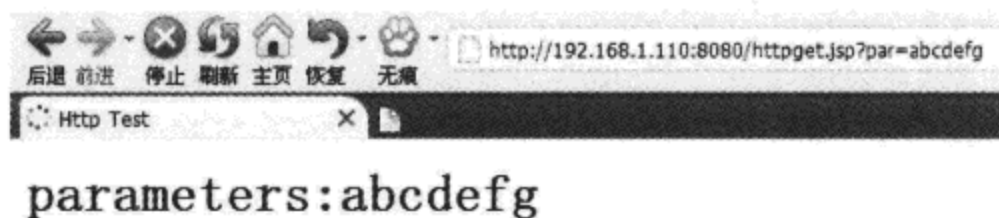


图 8-4 “httpget.jsp?par=abcdefg” 在浏览器中的效果

下面我们通过一个 Android 程序分别以不同的方式访问这两个页面，程序主界面如图 8-5 所示，不需要传递参数的网页 http1.jsp 被访问的效果如图 8-6 所示，通过 Get 方式访问 httpget.jsp 的效果如图 8-7 所示，通过 Post 访问 httpget.jsp 的效果如图 8-8 所示。



图 8-5 程序主界面



图 8-6 访问 http1.jsp



图 8-7 以 Get 方式访问 httpget.jsp

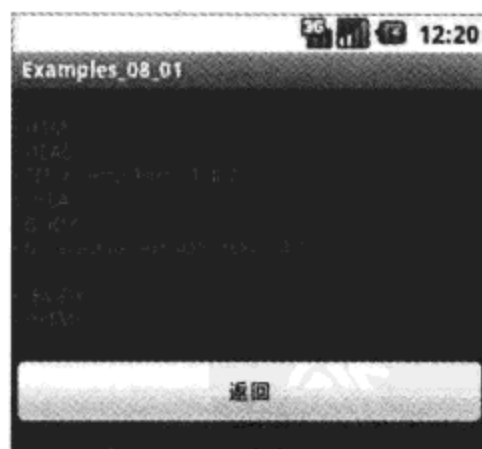


图 8-8 以 Post 方式访问 httpget.jsp

具体实现参见本书所附代码：第 8 章\Examples_08_01。

首先，我们来看看访问不需要传递参数的网页如何实现。非常简单！只需要打开一个 HttpURLConnection 连接，然后取得流中的数据，完成之后关闭这个连接，如代码清单 8-6 所示。

代码清单 8-6 第 8 章\Examples_08_01\src\com\yarin\android\Examples_08_01\Activity02.java

//直接获取数据

```
public class Activity02 extends Activity
{
    private final String DEBUG_TAG = "Activity02";
    public void onCreate(Bundle savedInstanceState)
```

```

{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.http);
    TextView mTextView = (TextView)this.findViewById(R.id.TextView_HTTP);
    //http 地址
    String httpUrl = "http://192.168.1.110:8080/http1.jsp";
    //获得的数据
    String resultData = "";
    URL url = null;
    try
    {
        //构造一个 URL 对象
        url = new URL(httpUrl);
    }
    catch (MalformedURLException e)
    {
        Log.e(DEBUG_TAG, "MalformedURLException");
    }
    if (url != null)
    {
        try
        {
            //使用 HttpURLConnection 打开连接
            HttpURLConnection urlConn = (HttpURLConnection)
            url.openConnection();
            //得到读取的内容(流)
            InputStreamReader in = new InputStreamReader
            (urlConn.getInputStream());
            // 为输出创建 BufferedReader
            BufferedReader buffer = new BufferedReader(in);
            String inputLine = null;
            //使用循环来读取获得的数据
            while (((inputLine = buffer.readLine()) != null))
            {
                //我们在每一行后面加上一个"\n"来换行
                resultData += inputLine + "\n";
            }
            //关闭 InputStreamReader
            in.close();
            //关闭 http 连接
            urlConn.disconnect();
            //设置显示取得的内容
            if ( resultData != null )
            {
                mTextView.setText(resultData);
            }
            else
            {
                mTextView.setText("读取的内容为 NULL");
            }
        }
        catch (IOException e)
        {
            Log.e(DEBUG_TAG, "IOException");
        }
    }
    else

```

```

    {
        Log.e(DEBUG_TAG, "Url NULL");
    }
    //设置按键事件监听
    Button button_Back = (Button) findViewById(R.id.Button_Back);
    /* 监听 button 的事件信息 */
    button_Back.setOnClickListener(new Button.OnClickListener()
    {
        public void onClick(View v)
        {
            /* 新建一个 Intent 对象 */
            Intent intent = new Intent();
            /* 指定 intent 要启动的类 */
            intent.setClass(Activity02.this, Activity01.class);
            /* 启动一个新的 Activity */
            startActivity(intent);
            /* 关闭当前的 Activity */
            Activity02.this.finish();
        }
    });
}
}

```

以 Get 方式传递参数也很简单, 对于 Activity02.java 只需要修改网页地址, 加上要传递的参数即可。代码如下 (其中, “?par=abcdefg” 则是我们传递的参数 par)。

```

String httpUrl = "http://192.168.1.110:8080/httpget.jsp?par=abcdefg";
URL url = new URL(httpUrl);
URLConnection urlConn = (URLConnection) url.openConnection();

```

由于 HttpURLConnection 默认使用 Get 方式, 所以如果我们要使用 Post 方式, 则需要 setRequestMethod 设置。然后将我们要传递的参数内容通过 writeBytes 方法写入数据流, 具体实现如代码清单 8-7 所示。

代码清单 8-7 第 8 章\Examples_08_01\src\com\yarin\android\Examples_08_01\Activity04.java

```

// 以 Post 方式上传参数
public class Activity04 extends Activity
{
    private final String DEBUG_TAG = "Activity04";
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.http);

        TextView mTextView = (TextView) this.findViewById(R.id.TextView_HTTP);
        //http 地址"?par=abcdefg"是我们上传的参数
        String httpUrl = "http://192.168.1.110:8080/httpget.jsp";
        //获得的数据
        String resultData = "";
        URL url = null;
        try
        {
            //构造一个 URL 对象
            url = new URL(httpUrl);
        }
        catch (MalformedURLException e)
        {

```

```

        Log.e(DEBUG_TAG, "MalformedURLException");
    }
    if (url != null)
    {
        try
        {
            // 使用 HttpURLConnection 打开连接
            HttpURLConnection urlConn = (HttpURLConnection)
            url.openConnection();
            //因为这个是 post 请求,需要设置为 true
            urlConn.setDoOutput(true);
            urlConn.setDoInput(true);
            // 设置以 Post 方式
            urlConn.setRequestMethod("POST");
            // Post 请求不能使用缓存
            urlConn.setUseCaches(false);
            urlConn.setInstanceFollowRedirects(true);
            // 配置本次连接的 Content-type, 配置为 application/x-www-form-urlencoded
            urlConn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            // 连接, 从 postUrl.openConnection() 至此的配置必须要在 connect 之前完成
            // 要注意的是 connection.getOutputStream 会隐含地进行 connect.
            urlConn.connect();
            //DataOutputStream 流
            DataOutputStream out = new DataOutputStream
            (urlConn.getOutputStream());
            //要上传的参数
            String content = "par=" + URLEncoder.encode("ABCDEFGH", "gb2312");
            //将要上传的内容写入流中
            out.writeBytes(content);
            //刷新、关闭
            out.flush();
            out.close();
            //获取数据
            BufferedReader reader = new BufferedReader(new
            InputStreamReader(urlConn.getInputStream()));
            String inputLine = null;
            //使用循环来读取获得的数据
            while (((inputLine = reader.readLine()) != null))
            {
                //我们在每一行后面加上一个"\n"来换行
                resultData += inputLine + "\n";
            }
            reader.close();
            //关闭 http 连接
            urlConn.disconnect();
            //设置显示取得的内容
            if (resultData != null )
            {
                mTextView.setText(resultData);
            }
            else
            {
                mTextView.setText("读取的内容为 NULL");
            }
        }
        catch (IOException e)
    }

```

```

        {
            Log.e(DEBUG_TAG, "IOException");
        }
    }
    else
    {
        Log.e(DEBUG_TAG, "Url NULL");
    }

    Button button_Back = (Button) findViewById(R.id.Button_Back);
    /* 监听 button 的事件信息 */
    button_Back.setOnClickListener(new Button.OnClickListener()
    {
        public void onClick(View v)
        {
            /* 新建一个 Intent 对象 */
            Intent intent = new Intent();
            /* 指定 intent 要启动的类 */
            intent.setClass(Activity04.this, Activity01.class);
            /* 启动一个新的 Activity */
            startActivity(intent);
            /* 关闭当前的 Activity */
            Activity04.this.finish();
        }
    });
}
}

```

扩展学习

上面我们学习了通过 `HttpURLConnection` 来完成网络通信，那么如果是网络上的一张图片，我们如何来显示呢？也很简单！连接方式和前面相同，只需要将连接之后得到的数据流转换成 `Bitmap` 就可以了。下面是一个显示网络图片的方法，如代码清单 8-8 所示。

代码清单 8-8 `GetNetBitmap` 方法

```

//取得网络上的图片
//url: 图片地址
public Bitmap GetNetBitmap(String url)
{
    URL imageUrl = null;
    Bitmap bitmap = null;
    try
    {
        imageUrl = new URL(url);
    }
    catch (MalformedURLException e)
    {
        Log.e(DEBUG_TAG, e.getMessage());
    }
    try
    {
        HttpURLConnection conn = (HttpURLConnection)imageUrl.openConnection();
        conn.setDoInput(true);
        conn.connect();
        //将得到的数据转换成 InputStream
        InputStream is = conn.getInputStream();
        //将 InputStream 转换成 Bitmap
    }
}

```

```

        bitmap = BitmapFactory.decodeStream(is);
        is.close();
    }
    catch (IOException e)
    {
        Log.e(DEBUG_TAG, e.getMessage());
    }
    return bitmap;
}

```

8.2.2 HttpClient 接口

上一节我们学习了通过标准 Java 接口来实现 Android 应用的联网操作。上面都只是简单地进行网络的访问，但是在实际开发中，可能会运用到更复杂的联网操作。Apache 提供了 HttpClient，它对 java.net 中的类做了封装和抽象，更适合我们在 Android 上开发联网应用。要使用 HttpClient，需要了解一些类。

1. ClientConnectionManager 接口

ClientConnectionManager 是客户端连接管理器接口，它提供以下几个抽象方法，如表 8-1 所示。

表 8-1 ClientConnectionManager 的抽象方法

方 法	说 明
ClientConnectionManager	关闭所有无效、超时的连接
closeIdleConnections	关闭空闲的连接
releaseConnection	释放一个连接
requestConnection	请求一个新的连接
shutdown	关闭管理器并释放资源

2. DefaultHttpClient

DefaultHttpClient 是默认的一个 HTTP 客户端，我们可以使用它创建一个 HTTP 连接。代码如下：

```
HttpClient httpClient = new DefaultHttpClient();
```

3. HttpResponse

HttpResponse 是一个 HTTP 连接响应，当执行一个 HTTP 连接后，就会返回一个 HttpResponse，可以通过 HttpResponse 获得一些响应的信息。下面是请求一个 HTTP 连接并获得该请求是否成功的代码。

```

HttpResponse httpResponse = httpClient.execute(httpRequest);
if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK)
{
    //连接成功
}

```

通过上面几个类的了解，下面我们将分别使用 Get 和 Post 方式请求一个网页（具体实现参见本书所附代码：第 8 章\Examples_08_02）。

我们先来看看 HttpClient 中如何使用 Get 方式获取数据，这里需要使用 HttpGet 来构建一个 Get 方式的 Http 请求，然后通过 HttpClient 来执行这个请求，HttpResponse 在接收这个请求后给出响

应,最后通过“`HttpResponse.getStatusLine().getStatusCode()`”来判断请求是否成功,并处理。具体实现见代码清单 8-9 所示。

代码清单 8-9 第 8 章\Examples_08_02\src\com\yarin\android\Examples_08_02\Activity02.java

```
public class Activity02 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.http);
        TextView mTextView = (TextView) this.findViewById(R.id.TextView_HTTP);
        // http 地址
        String httpUrl = "http://192.168.1.110:8080/httpget.jsp?par=Http
Client_android_Get";
        //HttpGet 连接对象
        HttpGet httpRequest = new HttpGet(httpUrl);
        try
        {
            //取得 HttpClient 对象
            HttpClient httpClient = new DefaultHttpClient();
            //请求 HttpClient, 取得 HttpResponse
            HttpResponse httpResponse = httpClient.execute(httpRequest);
            //请求成功
            if (httpResponse.getStatusLine().getStatusCode() == Http
Status.SC_OK)
            {
                //取得返回的字符串
                String strResult = EntityUtils.toString
(httpResponse.getEntity());
                mTextView.setText(strResult);
            }
            else
            {
                mTextView.setText("请求错误!");
            }
        }
        catch (ClientProtocolException e)
        {
            mTextView.setText(e.getMessage().toString());
        }
        catch (IOException e)
        {
            mTextView.setText(e.getMessage().toString());
        }
        catch (Exception e)
        {
            mTextView.setText(e.getMessage().toString());
        }

        //设置按键事件监听
        Button button_Back = (Button) findViewById(R.id.Button_Back);
        /* 监听 button 的事件信息 */
        button_Back.setOnClickListener(new Button.OnClickListener()
        {
            public void onClick(View v)
            {
                /* 新建一个 Intent 对象 */
            }
        });
    }
}
```



```

        Intent intent = new Intent();
        /* 指定 intent 要启动的类 */
        intent.setClass(Activity02.this, Activity01.class);
        /* 启动一个新的 Activity */
        startActivity(intent);
        /* 关闭当前的 Activity */
        Activity02.this.finish();
    }
});
}
}

```

Post 方式则比 Get 方式稍微复杂一点。首先使用 `HttpPost` 来构建一个 Post 方式的请求。代码如下：

```

String httpUrl = "http://192.168.1.110:8080/httpget.jsp";
HttpPost httpRequest = new HttpPost(httpUrl);

```

需要使用 `NameValuePair` 来保存要传递的参数，这里可以使用 `BasicNameValuePair` 来构造一个要被传递的参数，然后通过 `add` 方法添加这个参数到 `NameValuePair` 中，代码如下：

```

//使用 NameValuePair 来保存要传递的 Post 参数
List<NameValuePair> params = new ArrayList<NameValuePair>();
//添加要传递的参数
params.add(new BasicNameValuePair("par", "HttpClient_android_Post"));

```

Post 方式需要设置所使用的字符集，最后就和 Get 方式一样通过 `HttpClient` 来请求这个连接，返回响应并处理。下面是一个完整的通过 Post 方式请求的例子，如代码清单 8-10 所示。

代码清单 8-10 第 8 章\Examples_08_02\src\com\yarin\android\Examples_08_02\Activity03.java

```

public class Activity03 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.http);
        TextView mTextView = (TextView) this.findViewById(R.id.TextView_HTTP);
        // http 地址
        String httpUrl = "http://192.168.1.110:8080/httpget.jsp";
        //HttpPost 连接对象
        HttpPost httpRequest = new HttpPost(httpUrl);
        //使用 NameValuePair 来保存要传递的 Post 参数
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        //添加要传递的参数
        params.add(new BasicNameValuePair("par", "HttpClient_android_Post"));
        try
        {
            //设置字符集
            HttpEntity httpentity=new UrlEncodedFormEntity(params,"gb2312");
            //请求 httpRequest
            httpRequest.setEntity(httpentity);
            //取得默认的 HttpClient
            HttpClient httpclient = new DefaultHttpClient();
            //取得 HttpResponse
            HttpResponse httpResponse = httpclient.execute(httpRequest);
            //HttpStatus.SC_OK 表示连接成功
            if (httpResponse.getStatusLine().getStatusCode() ==

```

```

        HttpStatus.SC_OK)
    {
        //取得返回的字符串
        String strResult = EntityUtils.toString(
            httpResponse.getEntity());
        mTextView.setText(strResult);
    }
    else
    {
        mTextView.setText("请求错误!");
    }
}
catch (ClientProtocolException e)
{
    mTextView.setText(e.getMessage().toString());
}
catch (IOException e)
{
    mTextView.setText(e.getMessage().toString());
}
catch (Exception e)
{
    mTextView.setText(e.getMessage().toString());
}
//设置按键事件监听
Button button_Back = (Button) findViewById(R.id.Button_Back);
/* 监听 button 的事件信息 */
button_Back.setOnClickListener(new Button.OnClickListener()
{
    public void onClick(View v)
    {
        /* 新建一个 Intent 对象 */
        Intent intent = new Intent();
        /* 指定 intent 要启动的类 */
        intent.setClass(Activity03.this, Activity01.class);
        /* 启动一个新的 Activity */
        startActivity(intent);
        /* 关闭当前的 Activity */
        Activity03.this.finish();
    }
});
}
}

```

8.2.3 实时更新

前面两节我们学习了 Android 中两种 HTTP 通信方式，两个示例都只是简单地一次性获取网页数据，而在实际开发中更多的是需要我们实时获取最新数据，比如道路流量、实时天气信息等等，这时就需要通过一个线程来控制视图的更新，本例中我们首先创建一个网页来显示系统当前的时间，然后在 Android 程序中每隔 5 秒钟刷新一次视图，以达到实时更新的效果。

首先，创建一个显示当前系统时间的 jsp 网页文件，如代码清单 8-11 所示。

代码清单 8-11 date.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
```

```

<HTML>
<HEAD>
<TITLE>Date Test</TITLE>
</HEAD>
<BODY>
<%
java.text.SimpleDateFormat formatter=new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
java.util.Date currentTime = new java.util.Date();
String str_date1 = formatter.format(currentTime);
String str_date2 = currentTime.toString();
out.println("<h1>Date:"+str_date2+"</h1>");
%>
</BODY>
</HTML>

```

然后，我们浏览这个网页文件，如图 8-9 所示。

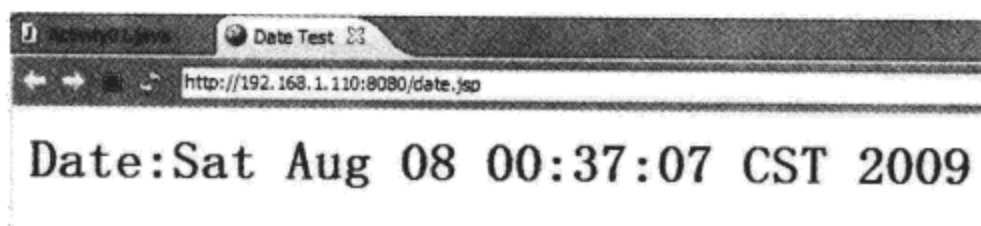


图 8-9 date.jsp 运行效果

下面我们通过 HTTP 连接来读取网页，如图 8-10 所示，每隔 5 秒后，程序自己刷新，如图 8-11 所示。我们可以看到图 8-10 和图 8-11 显示的内容不同，说明程序可以实时地从网络获取数据（具体实现参见本书所附代码：第 8 章\Examples_08_03）。

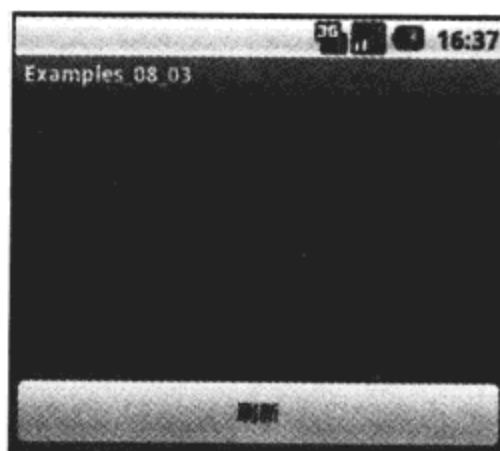


图 8-10 第一次更新视图

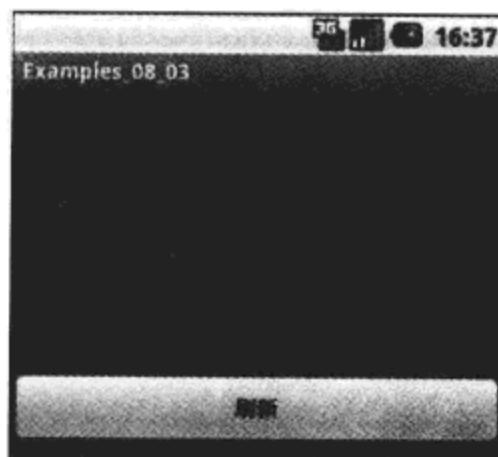


图 8-11 第二次更新视图

要实现实时地从网络获取数据，其实就是把获取网络数据的代码写到线程中，不停地进行更新。这里需要说明一点，Android 中更新视图不能直接在线程中进行，所以这里需要使用 Handler 来实现更新，具体代码如代码清单 8-12 所示。

代码清单 8-12 第 8 章\Examples_08_03\src\com\yarin\android\Examples_08_03\Activity01.java

```

public class Activity01 extends Activity
{
    private final String DEBUG_TAG = "Activity02";
    private TextView mTextView;
    private Button mButton;
    public void onCreate(Bundle savedInstanceState)
    {

```

```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mTextView = (TextView) this.findViewById(R.id.TextView01);
        mButton = (Button) this.findViewById(R.id.Button01);
        mButton.setOnClickListener(new Button.OnClickListener()
        {
            public void onClick(View arg0)
            {
                refresh();
            }
        });

        //开启线程
        new Thread(mRunnable).start();
    }
    //刷新网页显示
    private void refresh()
    {
        String httpUrl = "http://192.168.1.110:8080/date.jsp";
        String resultData = "";
        URL url = null;
        try
        {
            // 构造一个 URL 对象
            url = new URL(httpUrl);
        }
        catch (MalformedURLException e)
        {
            Log.e(DEBUG_TAG, "MalformedURLException");
        }
        if (url != null)
        {
            try
            {
                // 使用 HttpURLConnection 打开连接
                HttpURLConnection urlConn = (HttpURLConnection)
                    url.openConnection();
                // 得到读取的内容(流)
                InputStreamReader in = new InputStreamReader
                    (urlConn.getInputStream());
                // 为输出创建 BufferedReader
                BufferedReader buffer = new BufferedReader(in);
                String inputLine = null;
                // 使用循环来读取获得的数据
                while (((inputLine = buffer.readLine()) != null))
                {
                    // 我们在每一行后面加上一个"\n"来换行
                    resultData += inputLine + "\n";
                }
                // 关闭 InputStreamReader
                in.close();
                // 关闭 http 连接
                urlConn.disconnect();
                // 设置显示取得的内容
                if (resultData != null)
                {
                    mTextView.setText(resultData);
                }
            }
        }
    }

```

```

        else
        {
            mTextView.setText("读取的内容为 NULL");
        }
    }
    catch (IOException e)
    {
        Log.e(DEBUG_TAG, "IOException");
    }
}
else
{
    Log.e(DEBUG_TAG, "Url NULL");
}
}

private Runnable mRunnable = new Runnable()
{
    public void run()
    {
        while (true)
        {
            try
            {
                Thread.sleep(5 * 1000);
                //发送消息
                mHandler.sendMessage(mHandler.obtainMessage());
            } catch (InterruptedException e)
            {
                Log.e(DEBUG_TAG, e.toString());
            }
        }
    }
};

Handler mHandler = new Handler()
{
    public void handleMessage(Message msg)
    {
        super.handleMessage(msg);
        //刷新
        refresh();
    }
};
}

```

8.3 Socket 通信

8.2 节我们学习了 Android 中的 HTTP 通信，HTTP 通信中客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。要保持客户端程序的在线状态，需要不断地向服务器发起连接请求。通常的做法是即使不需要获得任何数据，客户端也保持每隔一段固定的时间向服务器发送一次“保持连接”的请求，服务器在收到该请求后对客户端进行回复，表明知道客户端“在线”。若服务器长时间无法收到客户端的请求，则认为客户端“下线”，若客户端长时间无法收到服务器的回复，则认为网络已经断开。很多情况

下, 需要服务器端主动向客户端发送数据, 保持客户端与服务器数据的实时与同步。若双方建立的是 HTTP 连接, 则服务器需要等到客户端发送一次请求后才能将数据传回给客户端, 因此, 客户端定时向服务器端发送连接请求, 不仅可以保持在线, 同时也是在“询问”服务器是否有新的数据, 如果有就将数据传给客户端。如果要开发一款多人联网的游戏, 那么 HTTP 已经不能很好地满足需求了! 这时就需要使用 Socket 通信。

8.3.1 Socket 基础

首先我们应该了解什么是 Socket, Socket 通常也称做“套接字”, 用于描述 IP 地址和端口, 是一个通信链的句柄。应用程序通常通过“套接字”向网络发出请求或者应答网络请求。它是通信的基石, 是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示, 包含进行网络通信必需的 5 种信息: 连接使用的协议、本地主机的 IP 地址、本地进程的协议端口、远地主机的 IP 地址和远地进程的协议端口。

1. Socket 传输模式

Socket 有两种主要的操作方式: 面向连接的和无连接的。面向连接的 Socket 操作就像一部电话, 必须建立一个连接和一个呼叫。所有的事情到达时的顺序与它们出发时的顺序是一样的。无连接的 Socket 操作就像是一个邮件投递, 没有什么保证, 多个邮件到达时的顺序可能与出发时的顺序不一样。到底用哪种模式是由应用程序的需要决定的。如果可靠性更重要的话, 用面向连接的操作会好一些。比如文件服务器需要数据的正确性和有序性, 如果一些数据丢失了, 系统的有效性将会失去。一些服务器, 比如间歇性地发送一些数据块, 如果数据丢了的话, 服务器并不想要再重新发一次, 因为当数据到达的时候, 它可能已经过时了。确保数据的有序性和正确性需要额外的操作, 这会带来内存消耗, 额外的费用将会降低系统的回应速率。

无连接的操作使用数据报协议。一个数据报是一个独立的单元, 它包含了这次投递的所有信息。把它想象成一个信封吧, 它有目的地址和要发送的内容。这个模式下的 Socket 不需要连接一个目的 Socket, 它只是简单地投出数据报。无连接的操作是快速的和高效的, 但是数据安全性不佳。

面向连接的操作使用 TCP 协议。一个这个模式下的 Socket 必须在发送数据之前与目的地的 Socket 取得连接。一旦连接建立了, Socket 就可以使用一个流接口进行打开、读、写、关闭操作。所有发送的信息都会在另一端以同样的顺序被接收。面向连接的操作比无连接操作的效率更低, 但是数据的安全性更高。

2. Socket 编程原理

(1) Socket 构造。

Java 在包 `java.net` 中提供了两个类 `Socket` 和 `ServerSocket`, 分别用来表示双向连接的客户端和服务端。这是两个封装得非常好的类, 使用很方便。其构造方法如下:

```
Socket(InetAddress address, int port);
Socket(InetAddress address, int port, boolean stream);
Socket(String host, int port);
Socket(String host, int port, boolean stream);
Socket(SocketImpl impl);
Socket(String host, int port, InetAddress localAddr, int localPort);
Socket(InetAddress address, int port, InetAddress localAddr, int localPort);
ServerSocket(int port);
```

```
ServerSocket(int port, int backlog);
ServerSocket(int port, int backlog, InetAddress bindAddr);
```

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

```
Socket client = new Socket("192.168.1.110", 54321);
ServerSocket server = new ServerSocket(54321);
```

注意, 在选择端口时必须小心。每一个端口提供一种特定的服务, 只有给出正确的端口, 才能获得相应的服务。0~1023 的端口号为系统所保留, 例如 `http` 服务的端口号为 80, `telnet` 服务的端口号为 21, `ftp` 服务的端口号为 23, 所以我们在选择端口号时, 最好选择一个大于 1023 的数, 防止发生冲突。在创建 `Socket` 时, 如果发生错误, 将产生 `IOException`, 在程序中必须对其进行处理。所以在创建 `Socket` 或 `ServerSocket` 时必须捕获或抛出异常。

(2) 客户端 Socket。

要想使用 `Socket` 来与一个服务器通信, 就必须先在客户端创建一个 `Socket`, 并指出需要连接的服务器的 IP 地址和端口, 这也是使用 `Socket` 通信的第一步, 代码如下:

```
try
{
    //192.168.1.110 是 IP 地址, 54321 是端口
    Socket socket=new Socket("192.168.1.110",54321);
}
catch(IOException e){}
```

(3) ServerSocket。

先看一个创建服务器端 `ServerSocket` 的过程, 代码如下:

```
ServerSocket server=null;
try {
    server=new ServerSocket(54321);
}catch(IOException e){}
try {
    Socket socket=server.accept();
}catch(IOException e){}
```

通过以上程序我们创建一个 `ServerSocket` 在端口 54321 监听客户请求, 它是 `Server` 的典型工作模式, 在这里 `Server` 只能接收一个请求, 接收后 `Server` 就退出了。实际的应用中总是让它不停地循环接收, 一旦有客户请求, `Server` 总是会创建一个服务线程来服务新来的客户, 而自己继续监听。程序中 `accept()` 是一个阻塞函数, 所谓阻塞性方法就是说该方法被调用后将等待客户的请求, 直到有一个客户启动并请求连接到相同的端口, 然后 `accept()` 返回一个对应于客户的 `Socket`。这时, 客户方和服务方都建立了用于通信的 `Socket`, 接下来就是由各个 `Socket` 分别打开各自的输入、输出流。

(4) 输入 (出) 流。

`Socket` 提供了方法 `getInputStream()` 和 `getOutputStream()` 来得到对应的输入 (出) 流以进行读 (写) 操作, 这两个方法分别返回 `InputStream` 和 `OutputStream` 类对象。为了便于读 (写) 数据, 我们可以在返回的输入/输出流对象上建立过滤流, 如 `DataInputStream`、`DataOutputStream` 或 `PrintStream` 类对象, 对于文本方式流对象, 可以采用 `InputStreamReader` 和 `OutputStreamWriter`、

PrintWriter 等处理。代码如下:

```
PrintStream os=new PrintStream(new BufferedOutputStream(socket.getOutputStream()));
DataInputStream is=new DataInputStream(socket.getInputStream());
PrintWriter out=new PrintWriter(socket.getOutputStream(),true);
BufferedReader in=new ButfferedReader(new InputSteramReader(Socket.getInputStream()));
```

(5) 关闭 Socket 和流。

每一个 Socket 存在时都将占用一定的资源,在 Socket 对象使用完毕时,要使其关闭,关闭 Socket 可以调用 Socket 的 close()方法。在关闭 Socket 之前,应与 Socket 相关的所有输入(出)流全部关闭,以释放所有的资源。而且要注意关闭的顺序,与 Socket 相关的所有的输入/输出应首先关闭,然后再关闭 Socket。尽管 Java 有自动回收机制,网络资源最终是会被释放的,但是为了有效利用资源,建议读者按照合理的顺序主动释放资源。

```
os.close();
is.close();
socket.close();
```

3. Android Socket 编程

在 Android 中完全可以使用 Java 标准 API 来开发网络应用,下面我们将实现一个服务器和客户端通信,客户端发送数据并接收服务器发回的数据。客户端界面如图 8-12 所示,编辑数据并点击“发送”按钮后,得到服务器回发的数据并显示,如图 8-13 所示,服务器端接收到的数据则如图 8-14 所示(具体实现参见本书所附代码:第 8 章\Examples_08_04)。

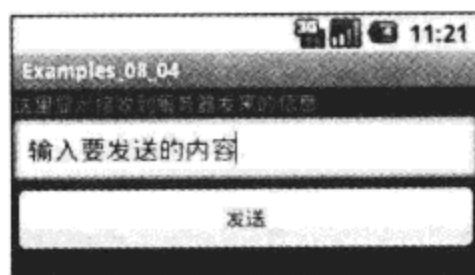


图 8-12 客户端界面

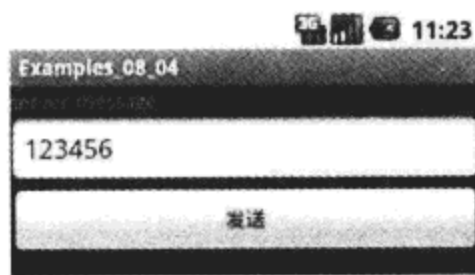


图 8-13 读取服务端数据并显示

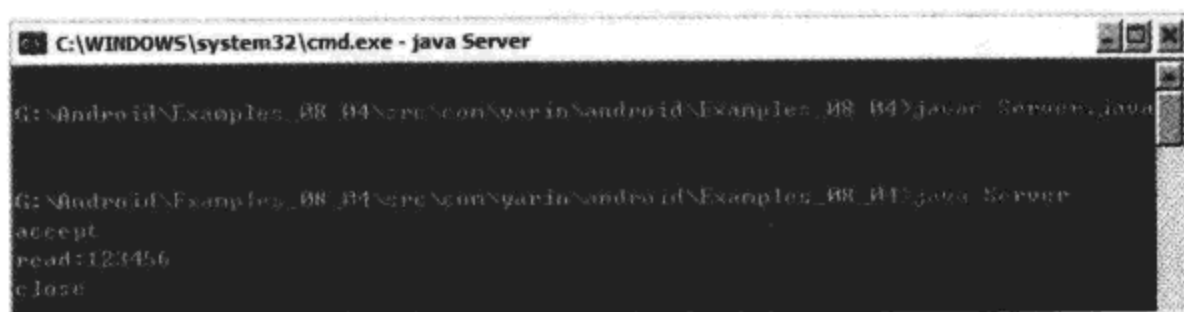


图 8-14 服务器接收到的数据

(1) 服务器实现。

首先来看看我们这里实现的服务器程序,如代码清单 8-13 所示。注意该程序需要单独编译,并在命令行模式下启动。

代码清单 8-13 第 8 章\Examples_08_04\src\com\yarin\android\Examples_08_04\Server.java

```
public class Server implements Runnable
{
    public void run()
    {
        try
```

```

        {
            //创建 ServerSocket
            ServerSocket serverSocket = new ServerSocket(54321);
            while (true)
            {
                //接收客户端请求
                Socket client = serverSocket.accept();
                System.out.println("accept");
                try
                {
                    //接收客户端消息
                    BufferedReader in = new BufferedReader
                        (new InputStreamReader(client.get
                            InputStream()));
                    String str = in.readLine();
                    System.out.println("read:" + str);
                    //向服务器发送消息
                    PrintWriter out = new PrintWriter( new
                        BufferedWriter( new OutputStreamWriter
                            (client.getOutputStream()), true));
                    out.println("server message");
                    //关闭流
                    in.close();
                    out.close();
                }
                catch (Exception e)
                {
                    System.out.println(e.getMessage());
                    e.printStackTrace();
                }
                finally
                {
                    //关闭
                    client.close();
                    System.out.println("close");
                }
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
    //main 函数, 开启服务器
    public static void main(String a[])
    {
        Thread desktopServerThread = new Thread(new Server());
        desktopServerThread.start();
    }
}

```

代码清单 8-13 中使用了 `java.net` 和 `java.io`。`java.net` 包提供了我们需要的 `Socket` 工具。`java.io` 包提供对流进行读写的工具。我们设置了服务器的端口为 54321, 开启了一个线程, 通过 `accept` 方法使服务器开始监听客户端的连接, 并取得客户端的 `Socket` 对象 `client`, 通过 `BufferedReader` 对象来接收客户端的输入流, 如果要向客户端发送数据, 可以使用 `PrintWriter` 来实现, 但是需要通过 `Socket` 对象来取得其输出流, 最后, 不要忘了关闭流和 `Socket`。`main` 函数用来开启服务器。

下面我们总结一下创建服务器的步骤：

- ☐ 指定端口实例化一个 `ServerSocket`。
- ☐ 调用 `ServerSocket` 的 `accept()` 以在等待连接期间造成阻塞。
- ☐ 获取位于该底层 `Socket` 的流以进行读写操作。
- ☐ 将数据封装成流。
- ☐ 对 `Socket` 进行读写。
- ☐ 关闭打开的流。

注意 不要在关闭 `Writer` 之前关闭 `Reader`。

(2) 客户端实现。

客户端实现如代码清单 8-14 所示。

代码清单 8-14 第 8 章\Examples_08_04\src\com\yarin\android\Examples_08_04\Activity01.java

```
public class Activity01 extends Activity
{
    private final String DEBUG_TAG = "Activity01";

    private TextView mTextView=null;
    private EditText mEditText=null;
    private Button mButton=null;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mButton = (Button)findViewById(R.id.Button01);
        mTextView=(TextView)findViewById(R.id.TextView01);
        mEditText=(EditText)findViewById(R.id.EditText01);
        //登录
        mButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v)
            {
                Socket socket = null;
                String message= mEditText.getText().toString()+"\r\n";
                try
                {
                    //创建 Socket
                    socket = new Socket("192.168.1.110",54321);
                    //向服务器发送消息
                    PrintWriter out = new PrintWriter( new
                    BufferedWriter( new OutputStreamWriter
                    (socket.getOutputStream()),true);
                    out.println(message);

                    //接收来自服务器的消息
                    BufferedReader br = new BufferedReader(new
                    InputStreamReader(socket.getInputStream()));
                    String msg = br.readLine();

                    if ( msg != null )
                    {
                        mTextView.setText(msg);
                    }
                }
            }
        });
    }
}
```

```

    }
    else
    {
        mTextView.setText("数据错误!");
    }
    //关闭流
    out.close();
    br.close();
    //关闭 Socket
    socket.close();
}
catch (Exception e)
{
    Log.e(DEBUG_TAG, e.toString());
}
});
}
}

```

代码清单 8-14 中我们监听了一个按钮事件，在按钮事件中通过“`socket = new Socket("192.168.1.110",54321);`”来请求连接服务器。和服务器一样，通过 `PrintWriter` 和 `BufferedReader` 来接收和发送消息。在接收到消息后，更新显示到 `TextView` 中。下面我们总结一下使用 `Socket` 来实现客户端的步骤：

- ☐ 通过 IP 地址和端口实例化 `Socket`，请求连接服务器。
- ☐ 获取 `Socket` 上的流以进行读写。
- ☐ 把流包装进 `BufferedReader/PrintWriter` 的实例。
- ☐ 对 `Socket` 进行读写。
- ☐ 关闭打开的流。

8.3.2 Socket 应用（简易聊天室）

前面的例子中只是实现了一个客户端和一个服务端的单独通信，并且只能一次通信，在实际应用中，往往是在服务器上运行一个永久的程序，它可以接收来自其他多个客户端的请求，并提供相应的服务。为了实现在服务器方给多个客户提供服务的功能，需要对上面的程序进行改造，利用多线程实现多客户机制。服务器总是在指定的端口上监听是否有客户请求，一旦监听到客户请求，服务器就会启动一个专门的服务线程来响应该客户的请求，而服务器本身在启动完线程之后马上又进入监听状态，等待下一个客户的到来。

下面使用 `Socket` 通信实现一个简单的聊天室程序（参见本书所附代码：第 8 章 \Examples_08_05）。首先来看看程序运行的效果，这里我们需要启动两个客户端来同时连接服务器，一个客户端是 `Android` 程序，另外一个为 `Java` 程序。首先启动服务器→启动 `Android` 客户端→启动另一个 `pc` 客户端→`Android` 客户端发送消息“11111”→`pc` 客户端发送消息“22222”→`Android` 客户端发送消息“33333”，现在服务器显示如图 8-15 所示，`Android` 客户端显示如图 8-16 所示，另外一个 `Java` 客户端显示如图 8-17 所示。



图 8-15 服务器界面



图 8-16 android 客户端界面

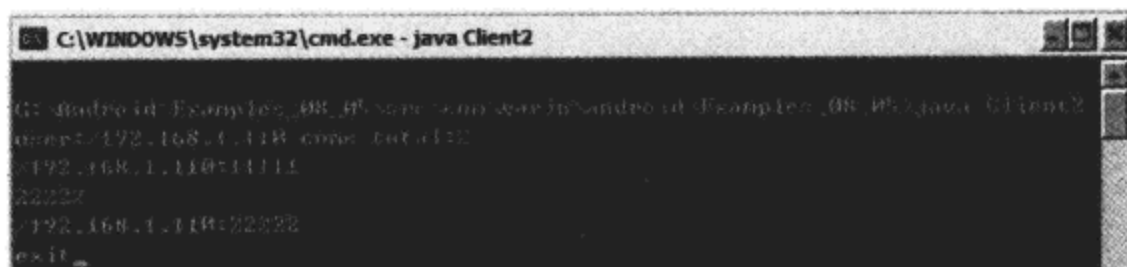


图 8-17 Java 客户端界面

可以看出，当一个客户端连接服务器（或者发送消息）之后，服务器将向所有客户端发送一个消息，这就需要服务器和客户端都一直处于监听状态，当有消息触发时进行相应的处理。下面我们来分析其实现过程。

首先来看服务器的实现，在服务器中我们通过一个 List 来存储所有连接进来的客户端的 Socket 对象，当然，用 CopyOnWriteArrayList 来存储也许会更好，这个交给大家自己来实现吧。与第 7 章的服务器程序不同的是，这里我们为每一个客户端都开启了一个线程，这样就可以同时对每个客户端的通信进行监听，具体实现如代码清单 8-15 所示。注意，该程序同上一示例一样，Server.java 和 Client2.java 都需要单独编译并在命令行模式下启动测试。

代码清单 8-15 第 8 章\Examples_08_05\src\com\yarin\android\Examples_08_05\Server.java

```
public class Server
{
    //服务器端口
    private static final int SERVERPORT = 54321;
    //客户端连接
    private static List<Socket> mClientList = new ArrayList<Socket>();
    //线程池
    private ExecutorService mExecutorService;
    //ServerSocket 对象
    private ServerSocket mServerSocket;
    //开启服务器
    public static void main(String[] args)
    {
        new Server();
    }
    public Server()
    {
        try
        {
            //设置服务器端口
            mServerSocket = new ServerSocket(SERVERPORT);
            //创建一个线程池
            mExecutorService = Executors.newCachedThreadPool();
            System.out.println("start...");
```

```

        //用来临时保存客户端连接的 Socket 对象
        Socket client = null;
        while (true)
        {
            //接收客户连接并添加到 List 中
            client = mServerSocket.accept();
            mClientList.add(client);
            //开启一个客户端线程
            mExecutorService.execute(new ThreadServer(client));
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
//每个客户端单独开启一个线程
static class ThreadServer implements Runnable
{
    private Socket mSocket;
    private BufferedReader mBufferedReader;
    private PrintWriter mPrintWriter;
    private String mStrMSG;

    public ThreadServer(Socket socket) throws IOException
    {
        this.mSocket = socket;
        mBufferedReader = new BufferedReader(new InputStreamReader
            (socket.getInputStream()));
        mStrMSG = "user:" + this.mSocket.getInetAddress() + " come
            total:" + mClientList.size();
        sendMessage();
    }

    public void run()
    {
        try
        {
            while((mStrMSG = mBufferedReader.readLine()) != null)
            {
                if (mStrMSG.trim().equals("exit"))
                {
                    //当一个客户端退出时
                    mClientList.remove(mSocket);
                    mBufferedReader.close();
                    mPrintWriter.close();
                    mStrMSG = "user:" + this.mSocket.
                        getInetAddress() + "exit total:"
                        + mClientList.size();
                    mSocket.close();
                    sendMessage();
                    break;
                }
                else
                {
                    mStrMSG = mSocket.getInet
                        Address() + ":" + mStrMSG;
                    sendMessage();
                }
            }
        }
    }
}

```

```

        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
//发送消息给所有客户端
private void sendMessage() throws IOException
{
    System.out.println(mStrMSG);
    for (Socket client : mClientList)
    {
        mPrintWriter = new PrintWriter(client.getOutputStream(), true);
        mPrintWriter.println(mStrMSG);
    }
}
}
}

```

客户端同样需要与服务器始终保持通信状态，这里需要注意的是，由于 Android 中线程是安全的，所以不能直接在线程中更新视图，所以我们需要使用 **Handler** 来更新界面的现实。当我们点击“登录”按钮时，连接服务器，并取得需要操作的流，点击“发送”按钮时我们取出输入框中的内容发送向服务器。由服务器来转发给其他每个客户端。具体实现如代码清单 8-16 所示。

代码清单 8-16 第 8 章\Examples_08_05\src\com\yarin\android\Examples_08_05\Activity01.java

```

public class Activity01 extends Activity
{
    private final String DEBUG_TAG = "Activity01";
    //服务器 IP、端口
    private static final String SERVERIP = "192.168.1.110";
    private static final int SERVERPORT = 54321;
    private Thread mThread = null;
    private Socket mSocket = null;
    private Button mButton_In = null;
    private Button mButton_Send = null;
    private EditText mEditText01 = null;
    private EditText mEditText02 = null;
    private BufferedReader mBufferedReader = null;
    private PrintWriter mPrintWriter = null;
    private String mStrMSG = "";
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mButton_In = (Button)findViewById(R.id.Button_In);
        mButton_Send = (Button)findViewById(R.id.Button_Send);
        mEditText01 = (EditText)findViewById(R.id.EditText01);
        mEditText02 = (EditText)findViewById(R.id.EditText02);
        //登录
        mButton_In.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v)
            {
                try
                {

```



```

        //连接服务器
        mSocket = new Socket(SERVERIP, SERVERPORT);
        //取得输入、输出流
        mBufferedReader = new BufferedReader
            (new InputStreamReader(mSocket.getInputStream()));
        mPrintWriter=new PrintWriter(mSocket.getOutputStream(), true);
    }
    catch (Exception e)
    {
        // TODO: handle exception
        Log.e(DEBUG_TAG, e.toString());
    }
}

});
//发送消息
mButton_Send.setOnClickListener(new OnClickListener()
{
    public void onClick(View v)
    {
        try
        {
            //取得编辑框中我们输入的内容
            String str = mEditText02.getText().toString() + "\n";
            //发送给服务器
            mPrintWriter.print(str);
            mPrintWriter.flush();
        }
        catch (Exception e)
        {
            // TODO: handle exception
            Log.e(DEBUG_TAG, e.toString());
        }
    }
});

mThread = new Thread(mRunnable);
mThread.start();
}
//线程:监听服务器发来的消息
private Runnable mRunnable= new Runnable()
{
    public void run()
    {
        while (true)
        {
            try
            {
                if ( (mStrMSG = mBufferedReader.readLine()) != null )
                {
                    //消息换行
                    mStrMSG+="\n";

                    mHandler.sendMessage(mHandler.obtainMessage());
                }
                // 发送消息
            }

```

```

    }
    catch (Exception e)
    {
        Log.e(DEBUG_TAG, e.toString());
    }
}

};
Handler mHandler = new Handler()
{
    public void handleMessage(Message msg)
    {
        super.handleMessage(msg);
        // 刷新
        try
        {
            //将聊天记录添加进来
            mEditText01.append(mStrMSG);
        }
        catch (Exception e)
        {
            Log.e(DEBUG_TAG, e.toString());
        }
    }
};
}

```

8.4 网络通信的中文乱码问题

在实际开发过程中，可能经常会遇到中文显示乱码的问题，那么这究竟是因为什么呢？要想解决 Java 中文乱码问题，首先需要了解字符、字符集、编码的概念以及常用的编码方式。

- 字符：是文字与符号的总称，包括文字、图形符号、数学符号等。
- 字符集：就是一组抽象字符的集合。字符集常常和一种具体的语言文字对应起来，该文字中的所有字符或者大部分常用字符就构成了该文字的字符集，比如英文字符集、繁体汉字字符集、日文字符集等等。
- 字符编码：计算机要处理各种字符，就需要将字符和二进制内码对应起来，这种对应关系就是字符编码。要制定编码首先要确定字符集，并将字符集内的字符排序，然后和二进制数字对应起来。根据字符集内字符的多少，确定用几个字节来编码。

ASCII 编码是目前计算机中用得最广泛的字符集及其编码。ISO-8859-1 可以表示的是西欧语言，它看起来很单一，为什么还在使用呢？由于是单字节编码，与计算机最基础的表示单位一致，所以很多时候，仍旧使用 ISO-8859-1 编码来表示，而且在很多协议上默认使用该编码。Unicode 编码（统一码、万国码、单一码）是一种在计算机上使用的字符编码，通常我们所遇到的 UTF-8 就是 Unicode 编码的实现方式。GB2312 字集是简体字集；BIG5 字集是台湾繁体字集；GBK 字集是简繁体字集，包括了 GB 字集、BIG5 字集和一些符号。GB18030 是国家制定的一个强制性大字符集标准，它的推出使汉字集有了统一的标准。Linux 系统默认使用的是 ISO-8859-1 编码，Win32 系统默认使用的是 GB2312 编码。

网络通信中，产生乱码的原因主要是通信过程中使用了不同的编码方式：服务器中的编码方式，传输过程中的编码方式，传输到达终端设备的编码方式。因此在传输过程中就需要至少两次编

码转换：首先从服务器编码转换为网络编码，再从网络编码转换为终端设备编码。在转换过程中发生任何情况都可能引起编码混乱，一般情况下我们可以通过以下两种方式来解决这个问题。

一种方式是：由于大部分终端设备都支持 Unicode 字符集，所以在连接网页时，我们希望网页数据在网络传输时使用 UTF-8 方式传输，这样就可以很简单地将 UTF-8 转换成 Unicode 字符集了。下面我们将通信过程中得到的流先转换为字节，然后再将字节按 GB2312 的方式进行转换得到字符串，代码如下：

```
InputStream is = conn.getInputStream();
BufferedInputStream bis = new BufferedInputStream(is);
byte bytearray[] = new byte[1024];
int current = -1;
int i = 0;
while ((current = bis.read()) != -1)
{
    bytearray[i] = (byte) current;
    i++;
}
resultData = new String(bytearray, "GB2312");
```

因此，通过上面的转换，“resultData”字符串便可以显示中文效果了。

另一种方式是在数据传递过程中使用 ISO-8859-1 字符集，这样就是直接使用 ASCII 编码方式，当然在传递到终端设备时，需要将其数据反转才能够正常显示。下面我们将一个字符串按 ISO-8859-1 字符集进行转换，代码如下：

```
public static String FormatStr(String str){
    if(str == null || str.length() == 0){
        return "";
    }
    try {
        return new String(str.getBytes("ISO-8859-1"), "bgk");
    } catch (UnsupportedEncodingException ex) {
        return str;
    }
}
```

归根结底，解决中文乱码只需要两个步骤：

- ❑ 使用 `getBytes("编码方式")`：来对汉字进行重编码，得到它的字节数组。
- ❑ 再使用 `new String(Bytes[], "解码方式")`：来对字节数组进行相应的解码。

只要理解了编码和解码的含义，并掌握了什么时候应该编码，什么时候应该解码，怎么编码及怎么解码，就不再害怕中文乱码问题了。

8.5 WebKit 应用

我们已经知道，Android 浏览器的内核是 Webkit 引擎，WebKit 的前身是 KDE 小组的 KHTML。Apple 将 KHTML 发扬光大，推出了装备 KHTML 的改进型的 WebKit 引擎的浏览器 Safari，获得了非常好的反响。WebKit 内核在手机上的应用十分广泛，例如 Google 的手机 Gphone、Apple 的 iPhone、Nokia 的 Series 60 browser 等所使用的 Browser 内核引擎，都是基于 WebKit。随着计算机、手机及连网装置的普及，未来终端运算都会在云端执行，目前云计算技术在网络服务中已经随处可见，例如搜寻引擎、网络信箱等，使用者只要输入简单指令即能得到大量信息。未来的手机、

GPS 等行动装置都可以透过云计算技术, 发展出更多的应用服务。因此人们只要拥有一个功能强大的浏览器, 就能满足平时工作生活的需要。本节我们将通过 Android 中的 WebKit 包来实现这些需求。

8.5.1 WebKit 概述

WebKit 是一个开源浏览器网页排版引擎, 与之相应的引擎有 Gecko (Mozilla、Firefox 等使用的排版引擎) 和 Trident (也称为 MSHTML, 是 IE 使用的排版引擎)。同时, WebKit 也是苹果 Mac OS X 系统引擎框架版本的名称, 主要用于 Safari、Dashboard、Mail 和其他一些 Mac OS X 程序。WebKit 所包含的 WebCore 排版引擎和 JSCore 引擎来自于 KDE 的 KHTML 和 KJS, 当年苹果比较了 Gecko 和 KHTML 后, 仍然选择了后者, 就因为它拥有清晰的源码结构、极快的渲染速度。而今 Android 系统也毫不犹豫地选择了 WebKit。它具备触摸屏、高级图形显示和上网功能, 用户能够在手机上查看电子邮件、搜索网址和观看视频节目等。可以看出这是一个非常强大的 Web 应用平台。

WebKit 由 3 个模块组成: JavaScriptCore、WebCore 和 WebKit, 其结构图如图 8-18 所示。

- WebKit: 整个项目的名称。
- JavaScriptCore: JavaScript 解释器。
- WebCore: 整个项目的核心, 用来实现 Render 引擎, 解析 Web 页面, 生成一个 DOM 树和一个 Render 树。

WebCore 的主要功能有:

- Page——与外框相关的内容 (Frame, Page, History, Focus, Window)。
- Loader——加载资源及 Cache。
- HTML——DOM HTML 内容及解析。
- DOM——DOM CORE 内容。
- XML——XML 内容及解析。
- Render——排版功能。
- CSS——DOM CSS 内容。
- Binding——DOM 与 JavaScriptCore 绑定的功能。
- Editing——所有与编辑相关的功能。

JavaScriptCore 的主要功能有:

- API——基本 Javascript 功能。
- Binding——与其他功能绑定的功能, 如 DOM、C、JNI。
- DerviedSource——自动产生的代码。
- ForwordHeads——头文件, 无实际意义。
- PCRE——Perl-Compatible Regular Expressions (Perd 兼容的规则表达式)。
- KJS——Javascript 内核。
- WTF——KDE 的 C++模板库

对 WebKit 各个模块的功能有了了解, 下面我们看看 WebKit 的解析过程是怎样的。流程如下:

- CURL 获得网站的 stream。

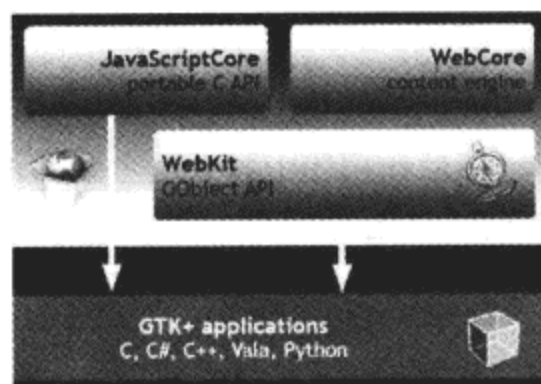


图 8-18 WebKit 结构图

- 解析划分字符串。
- 通过 DOM Builder 按合法的 HTML 规范生成 DOM 树。
- 如果有 Javascript, JSEngine 就通过 ECMA-262 标准完善 DOM 树。
- 把 DOM 传给 LayoutEngine 进行布局, 如果有 CSS 样式, 就通过 CSSParser 解析。
- 最后 Rendering 渲染出来。

而 Google 对 WebKit 进行了封装, 为开发者提供了丰富的 Java 接口, 其中最重要的便是 android.webkit.WebView 控件。下面我们重点学习 WebView 控件的使用。

8.5.2 WebView 浏览网页

Android 提供了 WebView 控件专门用来浏览网页, 和其他控件一样, 它使用起来非常简单。首先我们需要在 XML 布局文件中定义一个 WebView 控件, 代码如下:

```
<WebView
    android:id="@+id/WebView01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"/>
```

然后在程序中装载这个控件, 设置其属性, 比如: 颜色、字体、要访问的网址等 (当然也可以全部在 XML 中定义), 下面的代码通过 loadUrl 方法设置了当前 WebView 需要访问的网址:

```
mWebView = (WebView) findViewById(R.id.WebView01);
mWebView.loadUrl("http://www.google.com");
```

在 Android 中专门通过 WebSettings 来设置 WebView 的一些属性、状态等。在创建 WebView 时, 系统有一个默认的设置, 我们可以通过 WebView.getSettings 来得到这个设置。代码如下:

```
//取得 mWebView 的 WebSettings 对象
WebSettings webSettings = mWebView.getSettings();
```

WebSettings 和 WebView 都在同一个生命周期中存在, 当 WebView 被销毁后, 如果再使用 WebSettings 则会抛出 IllegalStateException 异常。下面是设置一些常用属性、状态的方法, 如表 8-2 所示。

表 8-2 WebSettings 的常用方法

方 法	说 明
setAllowFileAccess	启用或禁止 WebView 访问文件数据
setBlockNetworkImage	是否显示网络图像
setBuiltInZoomControls	设置是否支持缩放
setCacheMode	设置缓冲的模式
setDefaultFontSize	设置默认的字大小
setDefaultTextEncodingName	设置在解码时使用的默认编码
setFixedFontFamily	设置固定使用的字体
setJavaScriptEnabled	设置是否支持 Javascript
setLayoutAlgorithm	设置布局方式
setLightTouchEnabled	设置用鼠标激活被选项
setSupportZoom	设置是否支持变焦

将表 8-2 中的大部分方法中的 set 改为 get, 即可得到 WebView 的一些状态和属性。方法太多了, 这里我们仅列举了一些常用的方法, 大家可以参见官方 API。这样就可以浏览网页了, 但是通过调用系统浏览器来完成的。那么如何才能能在应用程序中自定义网页浏览程序呢? 可以使用 WebViewClient 来完成这个功能。

WebViewClient 就是专门辅助 WebView 处理各种通知、请求等事件的类。我们可以通过 WebView 的 setWebViewClient 方法来指定一个 WebViewClient 对象。WebViewClient 提供了如表 8-3 所示的一些方法，我们可以覆盖这些方法来辅助 WebView 浏览网页，代码如下（我们设置覆盖 shouldOverrideUrlLoading 方法，使得当有新连接时，使用当前的 WebView 来显示）：

```
public boolean shouldOverrideUrlLoading(WebView view, String url)
{
    view.loadUrl(url);
    return true;
}
```

表 8-3 WebViewClient 的常用方法

方 法	说 明
doUpdateVisitedHistory	更新历史记录
onFormResubmission	应用程序重新请求网页数据
onLoadResource	加载指定地址提供的资源
onPageFinished	网页加载完毕
onPageStarted	网页开始加载
onReceivedError	报告错误信息
onScaleChanged	WebView 发生改变
shouldOverrideUrlLoading	控制新的连接在当前 WebView 中打开

通过 WebViewClient 可以浏览网页的大部分内容了，但是现在很多网页中使用了 Javascript 脚本语言，比如用 Javascript 实现的对话框，这时我们如何处理这些对话框呢？Android 中还提供了一个重要的类 WebChromeClient，专门用来辅助 WebView 处理 Javascript 的对话框、网站图标、网站 Title、加载进度等。WebChromeClient 中的方法不是很多，其功能如表 8-4 所示。下面我们来实现 onReceivedTitle 方法，用于更改应用程序的 Title，代码如下：

```
public void onReceivedTitle(WebView view, String title)
{
    Activity01.this.setTitle(title);
    super.onReceivedTitle(view, title);
}
```

表 8-4 WebChromeClient 的常用方法

方 法	说 明
onCloseWindow	关闭 WebView
onCreateWindow	创建 WebView
onJsAlert	处理 Javascript 中的 Alert 对话框
onJsConfirm	处理 Javascript 中的 Confirm 对话框
onJsPrompt	处理 Javascript 中的 Prompt 对话框
onProgressChanged	加载进度条改变
onReceivedIcon	网页图标更改
onReceivedTitle	网页 Title 更改
onRequestFocus	WebView 显示焦点

有了这些基础，下面来制作一个浏览器，对这些知识进行综合应用（参见本书所附代码：第 8 章\Examples_08_06），界面中使用 EditText 来输入网址，用 Button 来确认连接，用 WebView 来显示网页内容，效果如图 8-19 所示，当输入网址时浏览界面如图 8-20 所示。

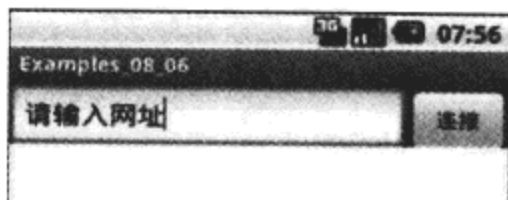


图 8-19 浏览器界面



图 8-20 浏览网页界面

这里使用了 `WebViewClient` 来辅助 `WebView` 处理一些事件，因此我们显示的网页都是在一个 `WebView` 控件中。当我们按返回键时，可以通过 `goBack` 和 `goForward` 方法来设置其前进和后退，在使用之前我们通过 `canGoBack` 和 `canGoForward` 方法来检测是否能够后退和前进。代码如下：

```
if ((keyCode == KeyEvent.KEYCODE_BACK) && mWebView.canGoBack())
{
    //返回前一个页面
    mWebView.goBack();
    return true;
}
```

这里我们重点学习如何处理 Javascript 的常用对话框，首先准备一个包含 Javascript 对话框的网页文件，效果如图 8-21 所示，图中显示了 3 种不同的对话框（参见本书所附代码：第 8 章 \Examples_08_06）。

现在我们在自己制作的浏览器中来访问这个 HTML 文件（笔者放置文件的地址为：`http://192.168.1.110:8080/dialog.html`），当然也可以使用“`file:///android_asset/dialog.html`”来访问这个文件，浏览效果如图 8-22 所示。当我们点击第一个按钮时，如图 8-23 所示；点击第二个按钮时，如图 8-24 所示；当我们再次点击确定按钮时，跳转到另一个页面，如图 8-25 所示。最后当我们点击第三个按钮时，如图 8-26 所示。

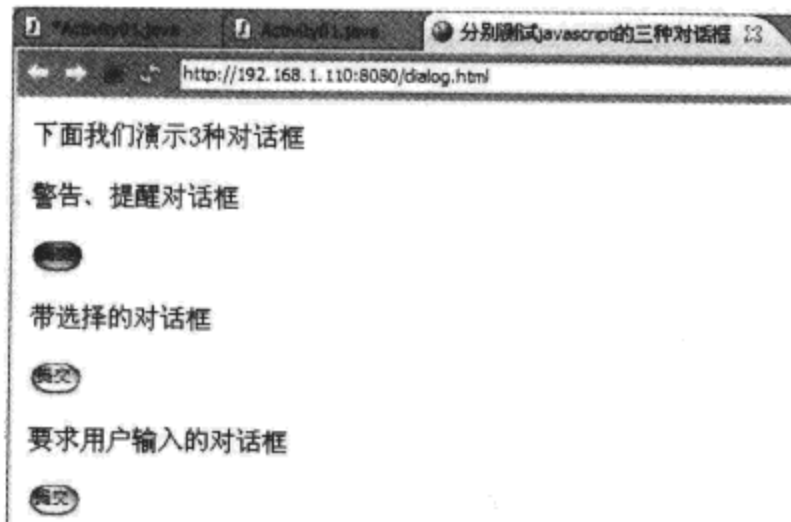


图 8-21 Javascript 的常用对话框演示



图 8-22 浏览 dialog.html

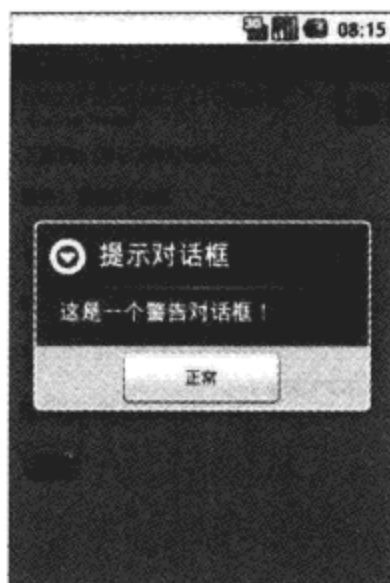


图 8-23 Javascript 的 alert 对话框

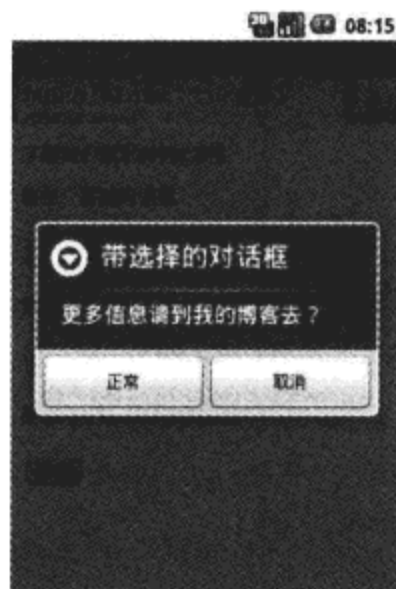


图 8-24 Javascript 的 confirm 对话框



图 8-25 Javascript 指定跳转的页面



图 8-26 Javascript 的 prompt 对话框

这里我们通过 `setWebChromeClient` 方法为 `WebView` 设置了一个 `WebChromeClient` 对象，来辅助 `WebView` 处理 Javascript 对话框等，图 8-26 是我们自定义的一个带输入的对话框，在图 8-23 和图 8-24 中我们都只需要监听这个按钮的事件，然后通过 `confirm` 和 `cancel` 方法将我们的操作传递给 Javascript 处理。具体实现参见代码清单 8-17 所示。

代码清单 8-17 第 8 章\Examples_08_06\src\com\yarin\android\Examples_08_06\Activity01.java

```
public class Activity01 extends Activity
{
    private final String DEBUG_TAG = "Activity01";
    private Button mButton;
    private EditText mEditText;
    private WebView mWebView;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

```

mButton = (Button) findViewById(R.id.Button01);
mEditText = (EditText) findViewById(R.id.EditText01);
mWebView = (WebView) findViewById(R.id.WebView01);
//设置支持 JavaScript 脚本
WebSettings webSettings = mWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
//设置可以访问文件
webSettings.setAllowFileAccess(true);
//设置支持缩放
webSettings.setBuiltInZoomControls(true);
//设置 WebViewClient
mWebView.setWebViewClient(new WebViewClient()
{
    public boolean shouldOverrideUrlLoading(WebView view, String url)
    {
        view.loadUrl(url);
        return true;
    }

    public void onPageFinished(WebView view, String url)
    {
        super.onPageFinished(view, url);
    }

    public void onPageStarted(WebView view, String url, Bitmap favicon)
    {
        super.onPageStarted(view, url, favicon);
    }
});
//设置 WebChromeClient
mWebView.setWebChromeClient(new WebChromeClient() {
    @Override
    //处理 Javascript 中的 alert
    public boolean onJsAlert(WebView view, String url, String message,
        final JsResult result)
    {
        //构建一个 Builder 来显示网页中的对话框
        Builder builder = new Builder(Activity01.this);
        builder.setTitle("提示对话框");
        builder.setMessage(message);
        builder.setPositiveButton(android.R.string.ok,
            new AlertDialog.OnClickListener() {
                public void onClick
                    (DialogInterface dialog,
                     int which) {
                    //点击确定按钮之后,继续执行网页中的操作
                    result.confirm();
                }
            });
        builder.setCancelable(false);
        builder.create();
        builder.show();
        return true;
    }
});

//处理 Javascript 中的 confirm
public boolean onJsConfirm(WebView view, String url,

```

```

String message,
        final JsResult result)
{
    Builder builder = new Builder(Activity01.this);
    builder.setTitle("带选择的对话框");
    builder.setMessage(message);
    builder.setPositiveButton(android.R.string.ok,
        new AlertDialog.OnClickListener() {
            public void onClick
                (DialogInterface dialog,
                 int which) {

result.confirm();

            }

        });

    builder.setNegativeButton(android.R.string.cancel,
        new DialogInterface.OnClickListener
            () {
                public void onClick
                    (DialogInterface dialog,
                     int which) {

result.cancel();

                }

            });
    builder.setCancelable(false);
    builder.create();
    builder.show();
    return true;
};
@Override
//处理 Javascript 中的 prompt
//message 为网页中对话框的提示内容
//defaultValue 为没有输入时默认显示的内容
public boolean onJsPrompt(WebView view, String url, String message,
    String defaultValue, final JsPromptResult result) {
    //自定义一个带输入的对话框由 TextView 和 EditText 构成
    final LayoutInflater factory =
        LayoutInflater.from(Activity01.this);
    final View dialogview =
        factory.inflate(R.layout.prom_dialog, null);
    //设置 TextView 对应网页中的提示信息
    ((TextView) dialogview.findViewById
        (R.id.TextView_PROM)).setText(message);
    //设置 EditText 对应网页中的输入框
    ((EditText) dialogview.findViewById
        (R.id.EditText_PROM)).setText(defaultValue);

    Builder builder = new Builder(Activity01.this);
    builder.setTitle("带输入的对话框");
    builder.setView(dialogview);
    builder.setPositiveButton(android.R.string.ok,
        new AlertDialog.OnClickListener
            () {
                public void onClick
                    (DialogInterface dialog,
                     int which) {

```

```

//点击确定之后,取得输入的值,传给网页处理
String
value = ((EditText) dialogview.findViewById(R.id.EditText_PROM)).getText().
toString();

result.confirm(value);

});

builder.setNegativeButton(android.R.string.cancel,
    new DialogInterface.OnClickListener() {
        public void onClick
            (DialogInterface dialog,
             int which) {

result.cancel();

        }

    });
builder.setOnCancelListener(new DialogInterface.
    OnCancelListener() {

        public void onCancel
            (DialogInterface dialog){

result.cancel();

        }

    });
builder.show();
return true;
};
@Override
//设置网页加载的进度条
public void onProgressChanged(WebView view, int newProgress)
{
    Activity01.this.getWindow().setFeatureInt(Window.FEATURE_PROGRESS,
        newProgress * 100);
    super.onProgressChanged(view, newProgress);
}

@Override
//设置应用程序的标题 title
public void onReceivedTitle(WebView view, String title)
{
    Activity01.this.setTitle(title);
    super.onReceivedTitle(view, title);
}

});
//连接按钮事件监听
mButton.setOnClickListener(new OnClickListener()
{
    public void onClick(View v)
    {
        try
        {
            //取得编辑框中我们输入的内容
            String url = mEditText.getText().toString();
            //判断输入的内容是不是网址

```

```

        if ( URLUtil.isNetworkUrl(url) )
        {
            //装载网址
            mWebView.loadUrl(url);
        }
        else
        {
            mEditText.setText("输入网址错误, 请重新输入");
        }
    }
    catch (Exception e)
    {
        Log.e(DEBUG_TAG, e.toString());
    }
    }
    });
}
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    if ((keyCode == KeyEvent.KEYCODE_BACK) &&
        mWebView.canGoBack())
    {
        //返回前一个页面
        mWebView.goBack();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
}

```

8.5.3 WebView 与 Javascript

WebView 不但可以运行一段 HTML 代码, 而且还有一个最重要的特点, 就是 WebView 可以同 Javascript 互相调用。有了它我们就可以用 HTML 和 Javascript 来编写 Android 应用。下面我们通过一个例子(参见本书所附代码: 第 8 章\Examples_08_07)来学习 WebView 如何和 Javascript 互相调用, 该例子将实现从 Android 应用中调出个人资料, 然后通过 Javascript 显示出来。首先在 Android 中定义一个 PersonalData 类, 用来保存个人资料, 并且定义得到这些数据的成员函数, 供 Javascript 调用。首先看看运行效果, 如图 8-27 所示。

这里需要通过 addJavascriptInterface(Object obj, String interfaceName) 方法将一个 Java 对象绑定到一个 Javascript 对象中, Javascript 对象名就是 interfaceName, 作用域是 Global, 这样便可以扩展 Javascript 的 API, 获取 Android 的数据。具体实现如代码清单 8-18 所示。

代码清单 8-18 第 8 章\Examples_08_07\src\com\yarin\android\Examples_08_07\Activity01.java

```

public class Activity01 extends Activity
{
    private WebView mWebView;

```

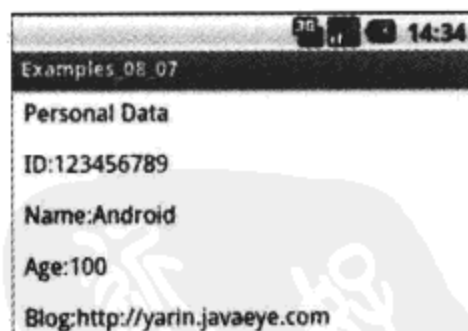


图 8-27 Javascript 调用显示
Android 应用中的数据

```

private PersonalData mPersonalData;
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mPersonalData = new PersonalData();
    mWebView = (WebView) this.findViewById(R.id.WebView01);
    //设置支持 JavaScript
    mWebView.getSettings().setJavaScriptEnabled(true);
    //把本类的一个实例添加到 Javascript 的全局对象 window 中,
    //这样就可以使用 window.PersonalData 来调用它的方法
    mWebView.addJavascriptInterface(this, "PersonalData");
    //加载网页
    mWebView.loadUrl("file:///android_asset/PersonalData.html");
}

//在 Javascript 脚本中调用得到 PersonalData 对象
public PersonalData getPersonalData()
{
    return mPersonalData;
}

//Javascript 脚本中调用显示的资料
class PersonalData
{
    String mID;
    String mName;
    String mAge;
    String mBlog;
    public PersonalData()
    {
        this.mID="123456789";
        this.mName="Android";
        this.mAge="100";
        this.mBlog="http://yarin.javaeye.com";
    }
    public String getID()
    {
        return mID;
    }
    public String getName()
    {
        return mName;
    }
    public String getAge()
    {
        return mAge;
    }
    public String getBlog()
    {
        return mBlog;
    }
}
}

```

这样初始化 WebView 后, 在 Javascript 中就可以使用 `window.PersonalData.getPersonalData()` 来取得 Java 对象, 下面是 Javascript 中访问 Android 应用的代码, 如代码清单 8-19 所示。

代码清单 8-19 第8章\Examples_08_07\assets\test.js

```

window.onload= function(){
    //取得 Java 对象
    var personaldata = window.PersonalData.getPersonalData();
    if(personaldata)
    {
        var Personaldata = document.getElementById("Personaldata");
        pnode = document.createElement("p");
        //通过 Java 对象访问其数据
        tnode = document.createTextNode("ID:" + personaldata.getID());
        pnode.appendChild(tnode);
        Personaldata.appendChild(pnode);
        pnode = document.createElement("p");
        tnode = document.createTextNode("Name:" + personaldata.getName());
        pnode.appendChild(tnode);
        Personaldata.appendChild(pnode);
        pnode = document.createElement("p");
        tnode = document.createTextNode("Age:" + personaldata.getAge());
        pnode.appendChild(tnode);
        Personaldata.appendChild(pnode);
        pnode = document.createElement("p");
        tnode = document.createTextNode("Blog:" + personaldata.getBlog());
        pnode.appendChild(tnode);
        Personaldata.appendChild(pnode);
    }
}

```

同时，在 Java 代码中也可以直接调用 Javascript 方法，这样就可以互相调用取得数据了，代码如下：

```
WebView.loadUrl("javascript:方法名()");
```

为了让 WebView 从 Apk 文件中加载 Assets，Android SDK 提供了一个 Schema，前缀为“file:///android_asset/”。WebView 遇到这样的 Schema，就去当前包中的 Assets 目录中找内容。所以我们使用了如下代码来加载一个网页：

```
mWebView.loadUrl("file:///android_asset/PersonalData.html");
```

8.6 WiFi 介绍

WiFi 全称 Wireless Fidelity，又称 802.11b 标准，它的最大优点就是传输速度较高，可以达到 11Mbps；另外它的有效距离也很长，同时也与已有的各种 802.11DSSS 设备兼容。WiFi 是一个无线网络通信技术的品牌，由 Wi-Fi 联盟（Wi-Fi Alliance）拥有，目的是改善基于 IEEE 802.11 标准的无线网络产品之间的互通性。Wi-Fi 联盟成立于 1999 年，当时的名称叫做 Wireless Ethernet Compatibility Alliance (WECA)，2002 年 10 月正式改名为 Wi-Fi Alliance。

由于 WiFi 的频段在世界范围内是无需任何电信运营执照的免费频段，因此 WLAN 无线设备提供了一个世界范围内可用的、费用极其低廉且数据带宽极高的无线空中接口。用户可以在 WiFi 覆盖区域内快速浏览网页，随时随地接听、拨打电话。而其他一些基于 WLAN 的宽带数据应用，如流媒体、网络游戏等功能更是值得用户期待。有了 WiFi 功能，我们打长途电话（包括国际长途）、浏览网页、收发电子邮件、音乐下载、数码照片传递等，再也无需担心速度慢和花费高的问题。现

在 WiFi 在国内的覆盖范围越来越广泛——高级宾馆、豪华住宅区、飞机场以及咖啡厅之类的场所都有 WiFi 接口。当我们去旅游、办公时，就可以在这些场所使用我们的掌上设备尽情网上冲浪了。

T-Mobile G1 自带了 Wi-Fi 无线网卡，可以在功能表中找到 Settings 设置图标，进入后选择 Wireless controls (无线控制)，其中就是 WiFi 的设置项。Android 也提供了 android.net.wifi 包供我们操作，WiFi 应用程序接口为应用程序和底层无线栈进行交流提供途径，而底层无线栈又为 WiFi 提供接入点。基本上来自请求端的信息都是可见的，包括已经连接网络的连接速度、IP 地址、完成状态和更多的信息，比如：可以到达的其他网络的信息，应用程序接口的特性（包括扫描、添加、保存、终止和开始与 WiFi 的连接），当然由于 WiFi 和硬件设备以及驱动等有关，所以不代表每个厂商的 Android 设备都具有 WiFi 功能。下面我们来实现一些常见的操作，主要包括以下几个类和接口。

1. ScanResult

主要用来描述已经检测出的接入点，包括接入点的地址、接入点的名称、身份认证、频率、信号强度等信息。

2. WifiConfiguration

WiFi 网络的配置，包括安全配置等。

3. WifiInfo

WiFi 无线连接的描述，包括接入点、网络连接状态、隐藏的接入点、IP 地址、连接速度、MAC 地址、网络 ID、信号强度等信息。

4. WifiManager

提供了管理 WiFi 连接的大部分 API，它主要包括如下内容：

- ❑ 已经配置好的网络的清单。这个清单可以查看和修改，而且可以修改个别记录的属性。
- ❑ 当连接中有活动的 Wi-Fi 网络时，可以建立或者关闭这个连接，并且可以查询有关网络状态的动态信息。
- ❑ 对接入点的扫描结果包含足够的信息来决定需要与什么接入点建立连接。
- ❑ 还定义了许多常量来表示 Wi-Fi 状态的改变。

5. WifiManager.WifiLock

允许应用程序一直使用 WiFi 无线网络，通常情况下当用户在一段时间内没有任何操作时，WiFi 网络就会自动关闭。我们可以通过 WifiLock 来锁定 WiFi 网络，使其一直保持连接，直到这个锁定被释放。当有多个应用程序时可能会有多个 WifiLock，在无线电设备中必须保证任何一个应用中都没有使用 WifiLock 时，才可以关闭它，在使用一个 WiFi 锁之前，必须确定应用需要 WiFi 的接入，或者能够在移动网络下工作。如果一个应用程序需要下载很大的文件，就需要保持 WiFi 锁，来确保应用程序有充足的时间下载完成。但是如果一个应用程序只是偶尔需要网络，就不应该保持 WiFi 锁，以防止对电池寿命的影响。注意 WiFi 锁不能超越客户端的 Wi-Fi Enabled 设置，也没有飞行模式，此时它们仍然认为 WiFi 在运行，但是设备处于空闲状态。

下面我们分别来学习如何使用这些类来控制应用程序中 WiFi 的连接。

要管理 WiFi 连接，首先来看看 WifiManager 如何使用。我们可以通过如下代码来获得 WifiManager 对象：

```
WifiManager wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

得到了这个对象，现在可以使用这个对象来操作 WiFi 连接。下面我们看看可以进行什么样的操作，WifiManager 的常用方法如表 8-5 所示。

表 8-5 WifiManager 的常用方法

方 法	说 明
addNetwork	添加一个配置好的网络连接
calculateSignalLevel	计算信号的强度
compareSignalLevel	比较两个信号的强度
createWifiLock	创建一个 WiFi 锁
disableNetwork	取消一个配置好的网络，即使其不可用
disconnect	从接入点断开
enableNetwork	允许指定的网络连接
getConfiguredNetworks	得到客户端所有已经配置好的网络列表
getConnectionInfo	得到正在使用的连接的动态信息
getDhcpInfo	得到最后一次成功的 DHCP 请求的 DHCP 地址
getScanResults	得到被扫描到的接入点
getWifiState	得到可用 WiFi 的状态
isWifiEnabled	得到 WiFi 是否可用
pingSupplicant	检查客户端对请求的反应
reassociate	从当前接入点重新连接
removeNetwork	从已经配置好的网络列表中删除指定 ID 的网络
saveConfiguration	保存当前配置好的网络列表
setWifiEnabled	设置 WiFi 是否可用
startScan	扫描存在的接入点
updateNetwork	更新已经配置好的网络

最后我们来实现一个管理 WiFi 的类（参见本书所附代码：第 8 章\Examples_08_08），并完整地学习 WiFi 的相关内容。首先，我们的 WifiAdmin 需要打开（关闭）WIFI、锁定（释放）WifiLock、创建 WifiLock、取得配置好的网络、扫描、连接、断开、获取网络连接的信息等基本操作。如代码清单 8-20 所示。

代码清单 8-20 第 8 章\Examples_08_08\src\com\yarin\android\Examples_08_08\WifiAdmin.java

```
public class WifiAdmin
{
    //定义 WifiManager 对象
    private WifiManager mWifiManager;
    //定义 WifiInfo 对象
    private WifiInfo mWifiInfo;
    //扫描出的网络连接列表
    private List<ScanResult> mWifiList;
    //网络连接列表
    private List<WifiConfiguration> mWifiConfiguration;
    //
    WifiLock mWifiLock;
    //构造器
    public WifiAdmin(Context context)
    {
        //取得 WifiManager 对象
        mWifiManager = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
        //取得 WifiInfo 对象
```

```
        mWifiInfo = mWifiManager.getConnectionInfo();
    }
    //打开 WiFi
    public void OpenWifi()
    {
        if (!mWifiManager.isWifiEnabled())
        {
            mWifiManager.setWifiEnabled(true);
        }
    }
    //关闭 WiFi
    public void CloseWifi()
    {
        if (!mWifiManager.isWifiEnabled())
        {
            mWifiManager.setWifiEnabled(false);
        }
    }
    //锁定 WifiLock
    public void AcquireWifiLock()
    {
        mWifiLock.acquire();
    }
    //解锁 WifiLock
    public void ReleaseWifiLock()
    {
        //判断时候锁定
        if (mWifiLock.isHeld())
        {
            mWifiLock.acquire();
        }
    }
    //创建一个 WifiLock
    public void CreatWifiLock()
    {
        mWifiLock = mWifiManager.createWifiLock("Test");
    }
    //得到配置好的网络
    public List<WifiConfiguration> GetConfiguration()
    {
        return mWifiConfiguration;
    }
    //指定配置好的网络进行连接
    public void ConnectConfiguration(int index)
    {
        //索引大于配置好的网络索引返回
        if (index > mWifiConfiguration.size())
        {
            return;
        }
        //连接配置好的指定 ID 的网络
        mWifiManager.enableNetwork(mWifiConfiguration.get(index).networkId, true);
    }
    public void StartScan()
    {
        mWifiManager.startScan();
    }
}
```

```

        //得到扫描结果
        mWifiList = mWifiManager.getScanResults();
        //得到配置好的网络连接
        mWifiConfiguration = mWifiManager.getConfiguredNetworks();
    }
    //得到网络列表
    public List<ScanResult> GetWifiList()
    {
        return mWifiList;
    }
    //查看扫描结果
    public StringBuilder LookUpScan()
    {
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < mWifiList.size(); i++)
        {
            stringBuilder.append("Index_"+new Integer(i+1).toString() + ":");
            //将 ScanResult 信息转换成一个字符串包
            //其中包括 BSSID、SSID、capabilities、frequency 和 level
            stringBuilder.append(mWifiList.get(i).toString());
            stringBuilder.append("\n");
        }
        return stringBuilder;
    }
    //得到 MAC 地址
    public String GetMacAddress()
    {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.getMacAddress();
    }
    //得到接入点的 BSSID
    public String GetBSSID()
    {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.getBSSID();
    }
    //得到 IP 地址
    public int GetIPAddress()
    {
        return (mWifiInfo == null) ? 0 : mWifiInfo.getIpAddress();
    }
    //得到连接的 ID
    public int GetNetworkId()
    {
        return (mWifiInfo == null) ? 0 : mWifiInfo.getNetworkId();
    }
    //得到 WifiInfo 的所有信息包
    public String GetWifiInfo()
    {
        return (mWifiInfo == null) ? "NULL" : mWifiInfo.toString();
    }
    //添加一个网络并连接
    public void AddNetwork(WifiConfiguration wcg)
    {
        int wcgID = mWifiManager.addNetwork(wcg);
        mWifiManager.enableNetwork(wcgID, true);
    }
    //断开指定 ID 的网络
    public void DisconnectWifi(int netId)

```

```

        {
            mWifiManager.disableNetwork(netId);
            mWifiManager.disconnect();
        }
    }
}

```

当然了，我们既然使用了 WiFi，就需要在 AndroidManifest.xml 中加入对权限的声明，代码如下：

```

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_CHECKIN_PROPERTIES"></uses-permission>
<uses-permission android:name="android.permission.WAKE_LOCK"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.MODIFY_PHONE_STATE"></uses-permission>

```

由于 WiFi 需要设备和 WiFi 网络的支持，所以在模拟器上不能看到效果，因此必须在真机上测试，并且附近要有 WiFi 的接入点，大家尽情地去体验 WiFi 带来的乐趣吧！

8.7 蓝牙

具体说来，“蓝牙”是一种短距离的无线连接技术标准的代称，蓝牙的实质内容就是要建立通用的无线电空中接口及其控制软件的公开标准。蓝牙”计划主要面向网络中各类数据及语音设备，如 PC 机、笔记本电脑、打印机、传真机、数码相机、移动电话、家电设备等，使用无线微波的方式将它们连成一个微微网，多个微微网之间也可以互连，从而方便快速地实现各类设备之间的通信。蓝牙采用分散式网络结构以及快跳频和短包技术，支持点对点及点对多点通信，工作在全球通用的 2.4GHz ISM（即工业、科学、医学）频段，其数据速率为 1Mbps，采用时分双工传输方案实现全双工传输。

蓝牙协议可以分为 4 层，即核心协议层、电缆替代协议层、电话控制协议层和采纳的其他协议层。由于篇幅的限制，这里只向读者介绍核心协议。

蓝牙的核心协议包括基带、链路管理、逻辑链路控制与适应协议四部分。

链路管理（LMP）负责蓝牙组件间连接的建立。通过连接的发起、交换、核实，进行身份鉴别和加密等安全方面的操作；通过协商确定基带数据分组大小；它还控制无线单元的电源模式和工作周期，以及微微网内蓝牙组件的连接状态。

逻辑链路控制与适应协议（L2CAP）位于基带协议层之上，属于数据链路层，是一个为高层传输和应用层协议屏蔽基带协议的适配协议。其完成数据的拆装、基带与高协议间的适配，并通过协议复用、分用及重组操作为高层提供数据业务和分类提取，它允许高层协议和应用接收或发送长于 64KB 的 L2CAP 数据包。

业务搜寻协议（SDP）是极其重要的部分，它是所使用模式的基础。通过 SDP，可以查询设备信息、业务及业务特征，并在查询之后建立两个或多个蓝牙设备间的连接。SDP 支持 3 种查询方式：按业务类别搜寻、按业务属性搜寻和业务浏览。

蓝牙作为一个全球公开的无线应用标准，通过把各种语音和数据设备用无线链路连接起来，使人们能随时随地进行数据信息的交换与传输。无疑，它将在人们的日常生活和工作中扮演重要的角色，其市场潜力巨大，正成为目前的投资热点。很高兴的是 Android 在 2.0 版本中终于开始支持蓝

牙了，本节我们就来学习 Android 平台上的蓝牙开发。

注意 蓝牙是 Android 2.0 的包，所以我们在建立工程时，一定要将 API Level 等级设置为 5，即 Android 2.0。

有关蓝牙的类和接口位于 `android.bluetooth` 包中，首先我们看看蓝牙包中提供了哪些功能，如表 8-6 所示。

表 8-6 蓝牙功能包

功 能 包	说 明
<code>BluetoothAdapter</code>	蓝牙适配器（代表本地蓝牙适配器）
<code>BluetoothClass</code>	蓝牙类（主要包括服务和设备）
<code>BluetoothClass.Device</code>	蓝牙设备类
<code>BluetoothClass.Device.Major</code>	蓝牙设备管理
<code>BluetoothClass.Service</code>	有关蓝牙服务的类
<code>BluetoothDevice</code>	蓝牙设备（主要指远程蓝牙设备）
<code>BluetoothServerSocket</code>	监听蓝牙连接的类
<code>BluetoothSocket</code>	蓝牙连接类

这些蓝牙 API 允许应用程序连接和断开蓝牙耳机、扫描仪和其他蓝牙设备，包括编写和修改本地服务的 SDP 协议数据库和查询其他蓝牙设备上的 SDP 协议数据库，在 Android 上建立 RFCOMM 协议的连接并连接到其他指定设备上。

下面我们依然通过一个示例来学习 Android 上的蓝牙包的使用，该示例主要演示了如图 8-28 所示的功能。

每一个按钮代表了一个不同的蓝牙功能，下面我们就一来学习这些功能。具体源代码请参见本书所附代码：第 8 章\Examples_08_09。

首先，我们要使用蓝牙 API，并且会对蓝牙进行操作，所以我们就必须先先在 `AndroidManifest.xml` 中声明其权限，如代码清单 8-21 所示。

代码清单 8-21 `AndroidManifest.xml` 片段

```
/* API 等级要为 5 */
<uses-sdk android:minSdkVersion="5" />
/* 蓝牙权限 */
<uses-permission android:name="android.permission.BLUETOOTH" />
/* 蓝牙管理、操作 */
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
/* 读联系人-该示例会读取联系人 */
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

其次，要使用蓝牙，必须先取得蓝牙适配器，这里我们可以通过 `BluetoothAdapter` 的 `getDefaultAdapter()` 方法来取得本地蓝牙适配器，但是要取得远程的蓝牙适配器就需要使用 `BluetoothDevice` 类，在后面我们会详细讲解。代码如下：

```
/* 取得蓝牙适配器 */
BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
```

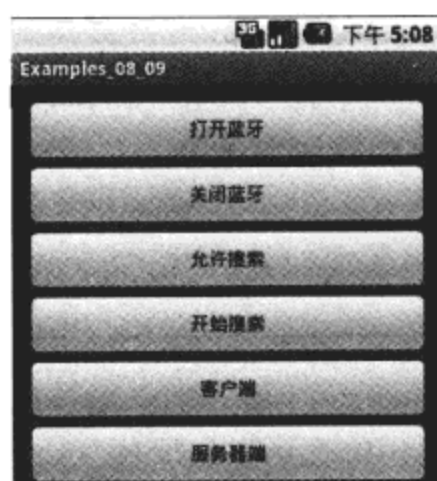


图 8-28 蓝牙功能示例

这些都准备好了，我们就先来学习如何打开、关闭本地蓝牙和如何才能使别的设备能够搜索到自己的设备。首先我们定义两个常量，分别用来代表请求打开和请求能够被搜索，代码如下：

```
/* 请求打开蓝牙 */
private static final int REQUEST_ENABLE = 0x1;
/* 请求能够被搜索 */
private static final int REQUEST_DISCOVERABLE = 0x2;
```

1. 请求开启蓝牙

取得了蓝牙适配器，要开启蓝牙就很简单了，代码如下：

```
// 用户请求打开蓝牙
Intent enabler = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enabler, REQUEST_ENABLE);
```

这里需要说明的就是 `BluetoothAdapter.ACTION_REQUEST_ENABLE`，代表请求系统允许用户打开蓝牙，有了这个例子，我们可以想象出请求能够被搜索该如何书写了。

2. 请求能够被搜索

同样，我们可以在 `BluetoothAdapter` 类中找到常量 `ACTION_REQUEST_DISCOVERABLE`，即请求系统允许蓝牙设备被搜索，代码如下：

```
// 使设备能够被搜索
Intent enabler = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
startActivityForResult(enabler, REQUEST_DISCOVERABLE);
```

其实在 `BluetoothAdapter` 类中还存在着许多这样的常量，用来执行某个活动、服务或者动作，具体内容请参见表 8-7。

表 8-7 BluetoothAdapter 中的动作常量

动作常量	说明
<code>ACTION_DISCOVERY_FINISHED</code>	已完成蓝牙搜索
<code>ACTION_DISCOVERY_STARTED</code>	已经开始搜索蓝牙设备
<code>ACTION_LOCAL_NAME_CHANGED</code>	更改蓝牙的名字
<code>ACTION_REQUEST_DISCOVERABLE</code>	请求能够被搜索
<code>ACTION_REQUEST_ENABLE</code>	请求启动蓝牙
<code>ACTION_SCAN_MODE_CHANGED</code>	扫描模式已改变
<code>ACTION_STATE_CHANGED</code>	状态已改变

3. 打开蓝牙

打开蓝牙功能，可以使用 `BluetoothAdapter` 类中的 `enable` 方法，代码如下：

```
// 关闭蓝牙
_bluetooth.enable();
```

4. 关闭蓝牙

关闭蓝牙很简单，直接使用 `BluetoothAdapter` 类中的 `disable` 方法即可，代码如下：

```
// 关闭蓝牙
_bluetooth.disable();
```

当然，`BluetoothAdapter` 类也不只是提供了表 8-7 所示的常量和关闭方法，还可以通过该类来操作蓝牙的其他一些属性，具体方法如表 8-8 所示。

表 8-8 BluetoothAdapter 中的常用方法

方 法	说 明
cancelDiscovery	取消当前设备搜索的过程
checkBluetoothAddress	检查蓝牙地址是否正确。如“00:43:A8:23:10:F0”字母字符必须是大写才能有效
disable	关闭蓝牙适配器
enable	打开蓝牙适配器
getAddress	取得本地蓝牙的硬件适配器地址
getDefaultAdapter	得到默认的蓝牙适配器
getName	得到蓝牙的名字
getRemoteDevice	取得指定蓝牙硬件地址的 BluetoothDevice 对象
getScanMode	得到扫描模式
getState	得到状态
isDiscovering	是否允许被搜索
isEnabled	是否打开
setName	设置名字
startDiscovery	开始搜索

学习了蓝牙的基本功能操作之后，现在开始学习如何搜索网络上的设备并显示。

5. 搜索蓝牙设备

前面我们提到，搜索远程蓝牙设备需要使用 BluetoothDevice 类，首先我们可以使用 BluetoothAdapter 类的 getRemoteDevice 方法来得到一个指定地址的 BluetoothDevice。BluetoothDevice 类实际上是一个蓝牙硬件地址簿，该类对象是不可改变的。其操作都是远程蓝牙硬件地址使用 BluetoothAdapter 来创建一个 BluetoothDevice 对象。下面我们来看看搜索蓝牙设备的具体流程，如图 8-29 所示。

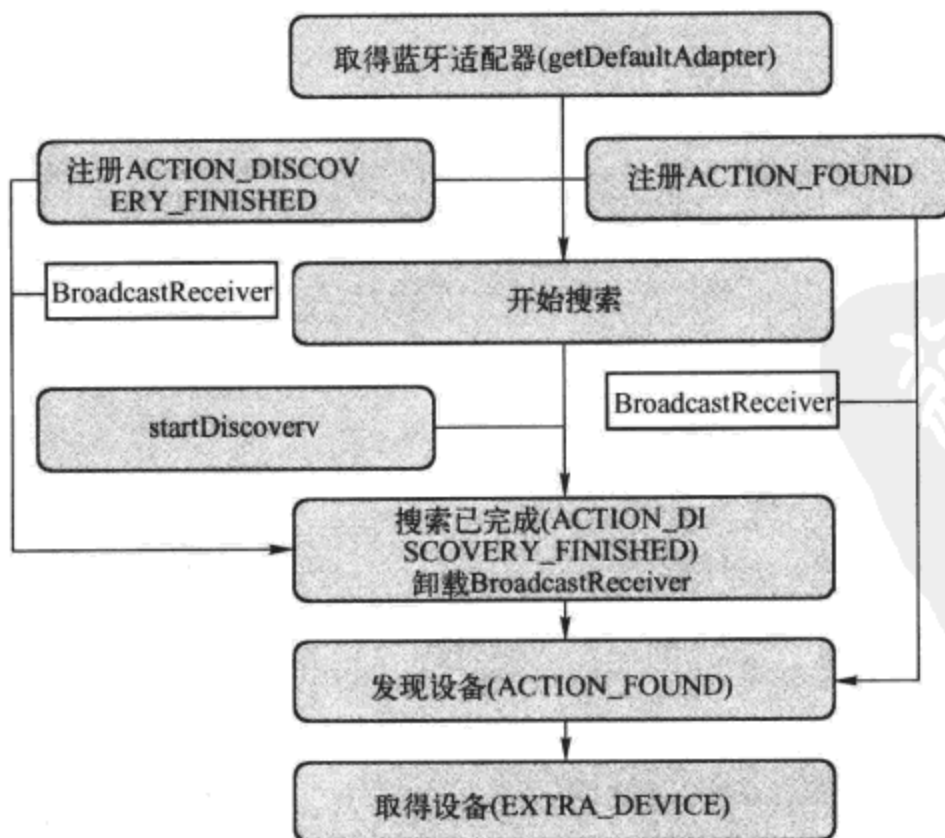


图 8-29 搜索蓝牙设备流程图

通过该流程图，我们就可以完成搜索蓝牙的类，如代码清单 8-22 所示。

代码清单 8-22 第 8 章\Examples_08_09\src\com\lyarin\android\Examples_08_09\DiscoveryActivity.java

```
public class DiscoveryActivity extends ListActivity
{
    private Handler _handler = new Handler();
    /* 取得默认的蓝牙适配器 */
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    /* 用来存储搜索到的蓝牙设备 */
    private List<BluetoothDevice> _devices = new ArrayList<BluetoothDevice>();
    /* 是否完成搜索 */
    private volatile boolean _discoveryFinished;
    private Runnable _discoveryWorkder = new Runnable() {
        public void run()
        {
            /* 开始搜索 */
            _bluetooth.startDiscovery();
            for (;;)
            {
                if (_discoveryFinished)
                {
                    break;
                }
                try
                {
                    Thread.sleep(100);
                }
                catch (InterruptedException e){}
            }
        }
    };
    /**
     * 接收器
     * 当搜索蓝牙设备完成时调用
     */
    private BroadcastReceiver _foundReceiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            /* 从 intent 中取得搜索结果数据 */
            BluetoothDevice device = intent
                .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            /* 将结果添加到列表中 */
            _devices.add(device);
            /* 显示列表 */
            showDevices();
        }
    };
    private BroadcastReceiver _discoveryReceiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent)
        {
            /* 卸载注册的接收器 */
            unregisterReceiver(_foundReceiver);
            unregisterReceiver(this);
            _discoveryFinished = true;
        }
    };
    protected void onCreate(Bundle savedInstanceState)
```

```

{
    super.onCreate(savedInstanceState);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
        WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
    setContentView(R.layout.discovery);
    /* 如果蓝牙适配器没有打开, 则结束 */
    if (!_bluetooth.isEnabled())
    {
        finish();
        return;
    }
    /* 注册接收器 */
    IntentFilter discoveryFilter = new IntentFilter(BluetoothAdapter.
        ACTION_DISCOVERY_FINISHED);
    registerReceiver(_discoveryReceiver, discoveryFilter);
    IntentFilter foundFilter = new IntentFilter(BluetoothDevice.
        ACTION_FOUND);
    registerReceiver(_foundReceiver, foundFilter);
    /* 显示一个对话框, 正在搜索蓝牙设备 */
    SamplesUtils.indeterminate(DiscoveryActivity.this, _handler, "Scanning...",
        _discoveryWorkder, new OnDismissListener() {
            public void onDismiss(DialogInterface dialog)
            {
                for (; _bluetooth.isDiscovering(); )
                {
                    _bluetooth.cancelDiscovery();
                }
                _discoveryFinished = true;
            }
        }, true);
}
/* 显示列表 */
protected void showDevices()
{
    List<String> list = new ArrayList<String>();
    for (int i = 0, size = _devices.size(); i < size; ++i)
    {
        StringBuilder b = new StringBuilder();
        BluetoothDevice d = _devices.get(i);
        b.append(d.getAddress());
        b.append('\n');
        b.append(d.getName());
        String s = b.toString();
        list.add(s);
    }
    final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, list);
    _handler.post(new Runnable() {
        public void run()
        {
            setListAdapter(adapter);
        }
    });
}
protected void onListItemClick(ListView l, View v, int position, long id)
{
    Intent result = new Intent();
    result.putExtra(BluetoothDevice.EXTRA_DEVICE,

```

```

        _devices.get(position));
        setResult(RESULT_OK, result);
        finish();
    }
}

```

代码清单 8-22 中我们首先用 `getDefaultAdapter` 方法取得了默认的蓝牙适配器，并且创建了一个用来存储搜索到的蓝牙设备 `BluetoothDevice` 的 `List`。然后在程序开始时注册了搜索已完成 (`BluetoothAdapter.ACTION_DISCOVERY_FINISHED`) 和发现设备 (`BluetoothDevice.ACTION_FOUND`) 两个接收器 `BroadcastReceiver`。然后通过一个线程来控制蓝牙设备的搜索 (`startDiscovery`)，当搜索中有触发上面两个接收器的事件，就直接传递给接收器进行保存。最后将保存在 `List` 中的 `BluetoothDevice` 显示在一个 `ListView` 中。由于模拟器上没有蓝牙设备，所以这里就不贴图演示了，大家可以在支持 Android 2.0 的真机上测试该程序。蓝牙同样可以实现 `Socket` 连接，下面我们将学习通过蓝牙 API 来实现一个 `Socket` 连接的服务器和客户端。

6. Socket 服务端与客户端

在 Android 的蓝牙 API 中我们可以看到 `BluetoothServerSocket` 和 `BluetoothSocket` 类，在建 `Socket` 连接时需要用到它们。首先我们来看如何实现一个蓝牙 `Socket` 服务器。

蓝牙的服务器端很简单，就是通过线程来注册一个具有名称和唯一识别的 `UUID` 号的 `BluetoothServerSocket`，然后就一直监听客户端 (`BluetoothSocket`) 的请求，并对这些请求做出相应的处理。首先我们来看看如何注册蓝牙服务器，代码如下：

```

/* 取得默认的蓝牙适配器 */
private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
/* 注册蓝牙服务器 */
private BluetoothServerSocket _serverSocket = _bluetooth.listenUsingRfcommWithServiceRecord(
    PROTOCOL_SCHEME_RFCOMM, UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));

```

其中值得注意的是 `listenUsingRfcommWithServiceRecord` 可以返回一个蓝牙服务器 (`BluetoothServerSocket`) 对象，其参数 `PROTOCOL_SCHEME_RFCOMM` 是一个 `String` 类型的常量，代表蓝牙服务器的名称，而 `UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666")` 则是该蓝牙服务器唯一标识 `UUID`。在客户端连接这个服务器时需要使用该 `UUID`。

连接之后就可以通过 `BluetoothServerSocket` 的 `accept` 方法来接收客户端的请求，并做出相应的处理，该方法会返回一个 `BluetoothSocket` 对象代表客户端，代码如下：

```

/* 接收客户端的连接请求 */
BluetoothSocket socket = _serverSocket.accept();
/* 处理请求内容 */
if (socket != null) {}

```

最后，在关闭蓝牙服务器时，可以直接通过 `BluetoothServerSocket` 的 `close` 方法即可，代码如下：

```

/* 关闭服务器 */
_serverSocket.close();

```

下面我们来看看本例中一个蓝牙服务器类的完整实现，如代码清单 8-23 所示。

代码清单 8-23 第 8 章\Examples_08_09\src\com\yarin\android\Examples_08_09\ServerSocketActivity.java

```

public class ServerSocketActivity extends ListActivity
{

```

```

/* 一些常量, 代表服务器的名称 */
public static final String PROTOCOL_SCHEME_L2CAP = "bt12cap";
public static final String PROTOCOL_SCHEME_RFCOMM = "btspp";
public static final String PROTOCOL_SCHEME_BT_OBEX = "btgoep";
public static final String PROTOCOL_SCHEME_TCP_OBEX = "tcpobex";
private static final String TAG = ServerSocketActivity.class.getSimpleName();
private Handler _handler = new Handler();
/* 取得默认的蓝牙适配器 */
private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
/* 蓝牙服务器 */
private BluetoothServerSocket _serverSocket;
/* 线程-监听客户端的连接 */
private Thread _serverWorker = new Thread() {
    public void run() {
        listen();
    };
};
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
        WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
    setContentView(R.layout.server_socket);
    if (!_bluetooth.isEnabled()) {
        finish();
        return;
    }
    /* 开始监听 */
    _serverWorker.start();
}
protected void onDestroy() {
    super.onDestroy();
    shutdownServer();
}
protected void finalize() throws Throwable {
    super.finalize();
    shutdownServer();
}
/* 停止服务器 */
private void shutdownServer() {
    new Thread() {
        public void run() {
            _serverWorker.interrupt();
            if (_serverSocket != null) {
                try {
                    /* 关闭服务器 */
                    _serverSocket.close();
                } catch (IOException e) {
                    Log.e(TAG, "", e);
                }
                _serverSocket = null;
            }
        };
    }.start();
}
public void onClicked(View view) {
    shutdownServer();
}
protected void listen() {

```

```

try
{
    /* 创建一个蓝牙服务器
    * 参数分别为服务器名称、UUID
    */
    _serverSocket = _bluetooth.listenUsingRfcommWithServiceRecord (PROTOCOL_
    _SCHEME_RFCOMM, UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));
    /* 客户端连接列表 */
    final List<String> lines = new ArrayList<String>();
    _handler.post(new Runnable() {
        public void run() {
            lines.add("serverstarted...");
            ArrayAdapter<String> adapter = new ArrayAdapter
            <String>(ServerSocketActivity.this, android.R.
            layout.simple_list_item_1, lines);
            setListAdapter(adapter);
        }
    });
    /* 接收客户端的连接请求 */
    BluetoothSocket socket = _serverSocket.accept();
    /* 处理请求内容 */
    if (socket != null) {
        InputStream inputStream = socket.getInputStream();
        int read = -1;
        final byte[] bytes = new byte[2048];
        for (; (read = inputStream.read(bytes)) > -1;) {
            final int count = read;
            _handler.post(new Runnable() {
                public void run() {
                    StringBuilder b = new StringBuilder();
                    for (int i = 0; i < count; ++i) {
                        if (i > 0) {
                            b.append(' ');
                        }
                        String s = Integer.toHexString(bytes[i] & 0xFF);
                        if (s.length() < 2) {
                            b.append('0');
                        }
                        b.append(s);
                    }
                    String s = b.toString();
                    lines.add(s);
                    ArrayAdapter<String> adapter = new ArrayAdapter<String>(Server
                    SocketActivity.this, android.R.layout.simple_list_item_1, lines);
                    setListAdapter(adapter);
                }
            });
        }
    }
} catch (IOException e) {
    Log.e(TAG, "", e);
} finally {}
}
}

```

代码清单 8-23 中我们设置了一个蓝牙服务器，然后将连接进来的所有客户端都放进一个 List 中，并显示了服务器进行事务处理的日志文件，以便查看。最后当我们点击按钮“Stop Server”

时,就关闭服务器。完成了蓝牙服务器的搭建,下面我们学习如何建立一个客户端连接。

客户端的连接就更加简单了,我们只需要选择一个服务器(搜索出来的蓝牙设备),通过其唯一标识 UUID 就可以创建一个 BluetoothSocket 对象,然后通过 BluetoothSocket 的 connect 方法进行连接,在需要关闭时同样调用 BluetoothSocket 的 close 方法即可。下面我们先来看一个客户端连接的示例,如代码清单 8-24 所示。

代码清单 8-24 第8章\Examples_08_09\src\com\yarin\android\Examples_08_09\ClientSocketActivity.java

```
public class ClientSocketActivity extends Activity
{
    private static final String TAG = ClientSocketActivity.class.getSimpleName();
    private static final int REQUEST_DISCOVERY = 0x1;;
    private Handler _handler = new Handler();
    private BluetoothAdapter _bluetooth = BluetoothAdapter.getDefaultAdapter();
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
            WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
        setContentView(R.layout.client_socket);
        if (!_bluetooth.isEnabled()) {
            finish();
            return;
        }
        Intent intent = new Intent(this, DiscoveryActivity.class);
        /* 提示选择一个要连接的服务器 */
        Toast.makeText(this, "select device to connect", Toast.LENGTH_SHORT).show();
        /* 跳转到搜索的蓝牙设备列表区,进行选择 */
        startActivityForResult(intent, REQUEST_DISCOVERY);
    }
    /* 选择了服务器之后进行连接 */
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode != REQUEST_DISCOVERY) {
            return;
        }
        if (resultCode != RESULT_OK) {
            return;
        }
        final BluetoothDevice device = data.getParcelableExtra(Bluetooth
            Device.EXTRA_DEVICE);
        new Thread() {
            public void run() {
                /* 连接 */
                connect(device);
            }
        }.start();
    }
    protected void connect(BluetoothDevice device) {
        BluetoothSocket socket = null;
        try {
            //创建一个 Socket 连接: 只需要服务器在注册时的 UUID 号
            // socket = device.createRfcommSocketToServiceRecord
            (BluetoothProtocols.OBEX_OBJECT_PUSH_PROTOCOL_UUID);
            socket = device.createRfcommSocketToServiceRecord
            (UUID.fromString("a60f35f0-b93a-11de-8a39-08002009c666"));
            //连接
            socket.connect();
        }
    }
}
```



```

    } catch (IOException e) {
        Log.e(TAG, "", e);
    } finally {
        if (socket != null) {
            try {
                socket.close();
            } catch (IOException e) {
                Log.e(TAG, "", e);
            }
        }
    }
}
}
}

```

代码清单 8-24 中，首先判断了本地蓝牙是否处于启动状态，因为在连接时首先要确保本地蓝牙已经启动，然后通过代码

```
BluetoothDevice device=Intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
```

取得要连接的蓝牙服务设备，进而通过 **BluetoothDevice** 的 **createRfcommSocketToServiceRecord** 方法传入唯一的标识 **UUID** 创建了一个 **BluetoothSocket** 对象，最后通过 **connect** 方法与服务器进行了连接。整个过程虽然很简单，但是在蓝牙联网应用开发中必定会用到。

8.8 小结

本章主要学习了 **Android** 平台上网络与通信的开发，主要包括：无线网络技术的分析，网络通信中常见的 **HTTP** 和 **Socket** 通信，通信过程中的中文乱码处理方案，**WebView** 组件，**WiFi** 开发基础，蓝牙开发。在最常用的 **HTTP** 通信中分别讲述了 **HttpURLConnection** 和 **HttpClient** 接口的使用，学习 **Socket** 通信时我们通过一个聊天室程序对 **Socket** 进行了综合运用，学习 **WebView** 组件时我们自己制作了一个简易的浏览器，并介绍了与 **Javascript** 之间的互相调用，然后对 **WiFi** 的常用操作进行了综合，完成了一个 **WiFi** 管理器，最后我们对蓝牙进行了详细介绍，包括蓝牙实现的一个 **Socket** 服务器与客户端，每个知识点我们都举例分析了其功能及使用方法。

3G 网络在国外已经成熟，在国内也正在如火如荼地发展。不久 **WiFi** 在国内也将得到认可，两种网络技术在移动通信技术发展中将实现局部融合，各自发挥优势、扬长避短，必然会带来更多的网络应用。

Android 特色开发

Android 是一个面向应用程序开发的丰富平台，它拥有许多具有吸引力的用户界面元素、数据管理和网络应用等优秀的功能。Android 还提供了很多颇具特色的接口。本章我们将分别介绍这些吸引开发者眼球的特色开发，主要包括：传感器系统（Sensor）、语音识别技术（RecognizerIntent）、Google Map 和用来开发桌面的插件（Widget）。通过本章的学习，读者将对 Android 有一个更深入的了解，可以开发出一些有特色、有创意的应用程序。

9.1 传感器

据调查，2008 年全球传感器销售额为 506 亿美元，预计到 2010 年全球传感器销售额可达 600 亿美元以上。调查显示，东欧、亚太区和加拿大成为传感器市场增长最快的地区，而美国、德国、日本依旧是传感器市场分布最大的地区。就世界范围而言，传感器市场增长最快的领域依旧是汽车，占第二位的是过程控制，当然现在也被广泛应用于通信。那么，传感器的定义是什么呢？有哪些种类的传感器呢？Android 中提供了哪些传感器呢？

1. 传感器的定义

传感器是一种物理装置或生物器官，能够探测、感受外界的信号、物理条件（如光、热、湿度）或化学组成（如烟雾），并将探知的信息传递给其他装置或器官。国家标准 GB7665—87 对传感器的定义是：“能感受规定的被测量并按照一定的规律转换成可用信号的器件或装置，通常由敏感元件和转换元件组成”。传感器是一种检测装置，能感受被测量的信息，并能将检测的感受到的信息，按一定规律变换成为电信号或其他所需形式的信息输出，以满足信息的传输、处理、存储、显示、记录和控制等要求。它是实现自动检测和自动控制的首要环节。

2. 传感器的种类

可以从不同的角度对传感器进行分类：转换原理（传感器工作的基本物理或化学效应）；用途；输出信号类型以及制作材料和工艺等。

根据工作原理，传感器可分为物理传感器和化学传感器两大类。

物理传感器应用的是物理效应，诸如压电效应，磁致伸缩现象，离化、极化、热电、光电、磁电等效应。被测信号量的微小变化都将转换成电信号。

化学传感器包括那些以化学吸附、电化学反应等现象为因果关系的传感器，被测信号量的微小变化也将转换成电信号。

大多数传感器是以物理原理为基础运作的。化学传感器的技术问题较多，例如可靠性问题、规模生产的可能性、价格问题等，解决了这些问题，化学传感器的应用将会有巨大增长。而有些传感

器既不能划分为物理类，也不能划分为化学类。

3. Android 中传感器的种类

Google Android 操作系统中内置了很多传感器，比如 G1 自带了一个非常实用的加速感应器（微型陀螺仪），有了它，G1 手机就支持重力感应、方向判断等功能，在部分游戏或软件中可以自动识别屏幕的横屏、竖屏方向来改变屏幕显示布局。下面是 Android 中支持的几种传感器：

- ☐ Sensor.TYPE_ACCELEROMETER：加速度传感器。
- ☐ Sensor.TYPE_GYROSCOPE：陀螺仪传感器。
- ☐ Sensor.TYPE_LIGHT：亮度传感器。
- ☐ Sensor.TYPE_MAGNETIC_FIELD：地磁传感器。
- ☐ Sensor.TYPE_ORIENTATION：方向传感器。
- ☐ Sensor.TYPE_PRESSURE：压力传感器。
- ☐ Sensor.TYPE_PROXIMITY：近程传感器。
- ☐ Sensor.TYPE_TEMPERATURE：温度传感器。

下面我们通过一个例子来分析 Android 中传感器的使用（具体实现参见本书所附代码：第 9 章 Examples_09_01），这里分析的是方向传感器（TYPE_ORIENTATION）。

4. Android 中传感器的功能

要在 Android 中使用传感器，首先需要了解 `SensorManager` 和 `SensorEventListener`。顾名思义，`SensorManager` 就是所有传感器的一个综合管理类，包括了传感器的种类、采样率、精准度等。我们可以通过 `getSystemService` 方法来取得一个 `SensorManager` 对象。代码如下：

```
SensorManager mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

取得 `SensorManager` 对象之后，可以通过 `getSensorList` 方法来获得我们需要的传感器类型，保存到一个传感器列表中。通过如下代码可以得到一个方向传感器：

```
List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
```

要与传感器交互，应用程序必须注册以侦听与一个或多个传感器相关的活动。Android 中提供了 `registerListener` 来注册一个传感器，并提供了 `unregisterListener` 来卸载一个传感器。`registerListener` 方法包括 3 个参数：第 1 个参数，接收信号的 `Listener` 实例；第 2 个参数，想接收的传感器类型的列表（即上一步创建的 `List<Sensor>` 对象）；第 3 个参数，接收频度。调用之后返回一个布尔值，`true` 表示成功，`false` 表示失败。当然，之后不再使用时，我们还需要卸载。代码如下：

```
//注册传感器
Boolean mRegisteredSensor = mSensorManager.registerListener(this, sensor,
    SensorManager.SENSOR_DELAY_FASTEST);
//卸载传感器
mSensorManager.unregisterListener(this);
```

其中，`SensorEventListener` 是使用传感器的核心部分，包括以下两个方法必须实现：

- ☐ `onSensorChanged (SensorEvent event)` 方法在传感器值更改时调用。该方法只由受此应用程序监视的传感器调用。该方法的参数包括一个 `SensorEvent` 对象，该对象主要包括一组浮点数，表示传感器获得的方向、加速度等信息。例如，以下代码可以取得其值：

```
float x = event.values[SensorManager.DATA_X];
float y = event.values[SensorManager.DATA_Y];
float z = event.values[SensorManager.DATA_Z];
```

□ `onAccuracyChanged (Sensor sensor,int accuracy)` 方法在传感器的精准度发生改变时调用。其参数包括两个整数：一个表示传感器，另一个表示该传感器新的准确值。
具体实现如代码清单 9-1 所示。

代码清单 9-1 第9章\Examples_09_01\src\com\yarin\android\Examples_09_01\Activity01.java

```
public class Activity01 extends Activity implements SensorEventListener
{
    private boolean                mRegisteredSensor;
    //定义 SensorManager
    private SensorManager          mSensorManager;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mRegisteredSensor = false;
        //取得 SensorManager 实例
        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    }
    protected void onResume()
    {
        super.onResume();

        //接收 SensorManager 的一个列表 (Listener)
        //这里我们指定类型为 TYPE_ORIENTATION (方向传感器)
        List<Sensor> sensors = mSensorManager.getSensorList(
            Sensor.TYPE_ORIENTATION);

        if (sensors.size() > 0)
        {
            Sensor sensor = sensors.get(0);
            //注册 SensorManager
            //this->接收 sensor 的实例
            //接收传感器类型的列表
            //接收的频率
            mRegisteredSensor = mSensorManager.registerListener(this,
                sensor, SensorManager.SENSOR_DELAY_FASTEST);
        }
    }
    protected void onPause()
    {
        if (mRegisteredSensor)
        {
            //如果调用了 registerListener
            //这里我们需要 unregisterListener 来卸载/取消注册
            mSensorManager.unregisterListener(this);
            mRegisteredSensor = false;
        }
        super.onPause();
    }
    //当精准度发生改变时
    //sensor->传感器
    //accuracy->精准度
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {

```

```

        //处理精准度改变
    }
    // 当传感器在被改变时触发
    public void onSensorChanged(SensorEvent event)
    {
        // 接收方向传感器的类型
        if (event.sensor.getType() == Sensor.TYPE_ORIENTATION)
        {
            //这里我们可以得到数据, 然后根据需要来处理
            //由于模拟器上面无法测试效果, 因此我们暂时不处理数据
            float x = event.values[SensorManager.DATA_X];
            float y = event.values[SensorManager.DATA_Y];
            float z = event.values[SensorManager.DATA_Z];
        }
    }
}

```

上面的例子中演示了如何获得方向传感器的方向、加速度等信息, 我们可以根据得到的数值与上一次得到的数值之间的关系来进行需要的操作。SensorManager 中还有很多常量和一些常用的方法, 如下:

- ❑ getDefaultSensor: 得到默认的传感器对象。
- ❑ getInclination: 得到地磁传感器倾斜角的弧度值。
- ❑ getOrientation: 得到设备旋转的方向。
- ❑ getSensorList: 得到指定传感器的列表。

9.2 语音识别

语音识别技术在手机上应用得相当广泛, 我们日常最频繁的沟通方式是语音, 在手机应用中, 大部分是通过硬件手动输入, 目前这依然是主要与手机互动的方式, 然而对于像手机这种小巧的移动设备来说, 使用键盘甚至是虚拟键盘打字是一件非常不爽的事情。于是, Google 推出了强大的语音搜索业务。2008 年 11 月, Google 的语音搜索已经在 iPhone 平台上线, 而 Android 在 1.5 SDK 版本中也加强了语音识别功能, 并应用到了搜索功能上, 这的确是一个非常让人惊喜的更新。我们只需要点击搜索框旁边的那个小话筒形状的按钮, 如图 9-1 所示, Android 就可以通过语音识别你要搜索的内容。如果你的语音不够清晰, Android 也可以通过大体的意思来提供一些选择, 其宗旨是最大限度地改善人机交互的便捷性。相信很快会有更多人性化的功能出现在 Android 平台上, 比如我们在玩游戏时, 可以通过语音来控制操作, 让我们期待每一次革新带给我们的便捷吧!

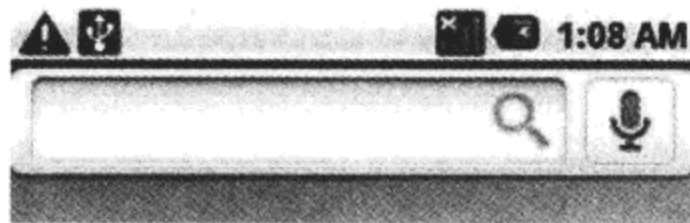


图 9-1 Android 语音识别按钮

Android 中主要通过 RecognizerIntent 来实现语音识别, 它主要包括一些常量来表示语音的模式等, 如表 9-1 所示。

表 9-1 RecognizerIntent 包括的常量

常 量	描 述
ACTION_RECOGNIZE_SPEECH	开启语音活动
ACTION_WEB_SEARCH	开启网络语音模式，结果将以网页搜索显示
EXTRA_LANGUAGE	可以理解为设置一个语言库
EXTRA_LANGUAGE_MODEL	语音识别模式
EXTRA_MAX_RESULTS	返回的最大结果
EXTRA_PROMPT	提示用户可以开始语音了
EXTRA_RESULTS	将字符串返回到一个 ArrayList 中
LANGUAGE_MODEL_FREE_FORM	在一种语言模式上自由语音
LANGUAGE_MODEL_WEB_SEARCH	使用语言模型在 Web 上搜索
RESULT_AUDIO_ERROR	返回结果时，音频遇到错误
RESULT_CLIENT_ERROR	返回结果时，客户端遇到错误
RESULT_NETWORK_ERROR	返回结果时，网络遇到错误
RESULT_NO_MATCH	没有检测到语音的错误
RESULT_SERVER_ERROR	返回结果时，服务器遇到错误

这里我们只需要通过 Intent 来传递一个动作以及一些属性，然后通过 startActivityForResult 来开始语音，代码如下：

```
Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "开始语音");
startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
```

当然，如果找不到设置，就会抛出异常 `ActivityNotFoundException`，所以我们需要捕捉这个异常。当然，另外需要实现 `onActivityResult` 方法，当语音结束时，会触发来获得语音的字符序列。下面我们通过一个例子来学习语音识别（参见本书所附代码：第 9 章\Examples_09_02），当我们点击“开始使用语音识别”按钮时，开始语音，然后在 `onActivityResult` 方法中取得结果并显示出来，运行效果如图 9-2 所示。由于在模拟器上没有设备，所以显示了 `ActivityNotFoundException` 异常，当我们在真机上测试、开始语音时，如图 9-3 所示，语音结束后取出的字符序列如图 9-4 所示。



图 9-2 ActivityNotFoundException 异常



图 9-3 开始语音



图 9-4 获取的字符序列

该例子很简单，具体实现如代码清单 9-2 所示。

代码清单 9-2 第 9 章\Examples_09_02\src\com\yarin\android\Examples_09_02\Activity01.java

```
public class Activity01 extends Activity
{
    private static final int VOICE_RECOGNITION_REQUEST_CODE = 4321;

    private ListView mList;

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

        mList = (ListView) findViewById(R.id.ListView01);

        Button button = (Button) findViewById(R.id.Button01);
        button.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                try
                {
                    //通过 Intent 传递语音识别的模式, 开启语音
                    Intent intent = new Intent
                    (RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
                    //语言模式和自由形式的语音识别
                    intent.putExtra(RecognizerIntent.EXTRA_
                    LANGUAGE_MODEL, RecognizerIntent.
                    LANGUAGE_MODEL_FREE_FORM);
                    //提示语音开始
                    intent.putExtra(RecognizerIntent.EXTRA_
                    PROMPT, "开始语音");
                    //开始执行我们的 Intent、语音识别
                    startActivityForResult(intent,
                    VOICE_RECOGNITION_REQUEST_CODE);
                }
                catch (ActivityNotFoundException e)
                {
                    //找不到语音设备装置
                    Toast.makeText(Activity01.this,
                    "ActivityNotFoundException",
                    Toast.LENGTH_LONG).show();
                }
            }
        });
    }
    //当语音结束时的回调函数 onActivityResult
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        // 判断是否是我们执行的语音识别
        if(requestCode==VOICE_RECOGNITION_REQUEST_CODE&&resultCode==RESULT_OK)
        {
            // 取得语音的字符
            ArrayList<String> results = data.getStringArrayListExtra
            RecognizerIntent.EXTRA_RESULTS);
            //设置视图更新
        }
    }
}
```



```

        //mList.setAdapter(new ArrayAdapter<String>(this, android.
        R.layout.simple_list_item_1, results));
        String resultsString = "";
        for (int i = 0; i < results.size(); i++)
        {
            resultsString += results.get(i);
        }
        Toast.makeText(this, resultsString, Toast.LENGTH_LONG).show();
        super.onActivityResult(requestCode, resultCode, data);
    }
}

```

9.3 Google Map

提起 Google Map (Google 地图), 大家无不想到其姊妹产品 Google Earth (Google 地球)。全新的免费地图服务让 Google 在 2005 年震惊了整个互联网界。此后, 各大门户纷纷推出自己的地图服务, 不少门户还和 Google 一样提供了二次开发的 API。目前, 基于地图服务的各种应用已如雨后春笋般到处萌发了。当然, 对于 Google 的 Android 系统来说, 地图肯定也是必不可少的特色。

9.3.1 Google Map 概述

Google Map 是 Google 公司提供的电子地图服务, 包括局部详细的卫星照片。它能提供三种视图: 一是矢量地图 (传统地图), 可提供政区和交通以及商业信息; 二是不同分辨率的卫星照片 (俯视地图, 与 Google Earth 上的卫星照片基本一样); 三是后来加上的地形视图, 可以用以显示地形和等高线。它的姊妹产品是 Google Earth——一个桌面应用程序, 在三维模型上提供街景和更多的卫星视图及 GPS 定位的功能。

Google 公司于 2004 年 11 月收购了美国 Keyhole 公司, 推出了 <http://maps.google.com>, 令人耳目一新。但 Google 并未就此止步, 在 2005 年 6 月底推出了桌面工具 Google Earth, 把“地球”放到了每个人的桌面上, 让你坐在电脑前, 就可以在名川大山间漫步, 在摩天楼群中俯瞰。

当然, 随着 Google Map 和 Google Earth 的诞生, 也出现了很多非常有趣的应用, 比如下面两个典型的基于 Google Earth 和 Google Maps 的小游戏非常有创意, 吸引了不少玩家。

如图 9-5 所示, 我们可以在全球任何地方甚至海底模拟开飞机或者潜水艇, 来漫游整个世界, 更多游戏请参见 <http://www.sea-seek.com/>。



图 9-5 模拟飞行

如图 9-6 所示，我们可以在地球上任何地方开着自己喜欢的车奔跑，非常有意思的是，可以在电脑面前开着车在自己周围以及熟悉的地方模拟驾驶。详细信息请参见 <http://geoquake.jp/en/webgame/DrivingSimulatorGM/>。



图 9-6 模拟驾驶

类似的应用还有很多，这里我们只介绍这两款，有兴趣的朋友可以自己去试试。现在 Google Map 已经被应用到很多手机上了，这更加方便了大家的生活。下面我们来看看手机上如何应用 Google Map，如图 9-7 所示。

它包括如下功能：

- ❑ 我的位置（测试版）：“我的位置”在地图上显示你的当前位置（通常在 1000 米范围之内）。即使没有 GPS，你也可以确定自己的位置。谷歌手机地图还支持内置 GPS，也可以链接到蓝牙 GPS 传感器，以便更准确地确定用户的位置。“我的位置”功能是通过识别你附近无线发射塔的信息广播而确定你的位置的。
- ❑ 地图和卫星视图：谷歌手机地图向你提供所查看地区的地图和卫星视图，其界面的使用感觉与你在台式机上相同。可沿其中一个方向滚动，以查看地图上的更多内容；或使用快捷键进行缩放。
- ❑ 商户列表：借助于 Google 的本地搜索引擎，可以按名称（如“星巴克”）或类型（如“咖啡”）搜索商家，查看商店的营业时间和评分，然后，只需点击一下即可拨通感兴趣的商家的电话。有了“我的位置”功能，甚至都不需要输入当前位置即可方便地找到附近的



图 9-7 Google Map 手机版

商家。

- ❑ 驾车路线：可以很方便地获得驾车路线，其中会清楚地标明每次转弯。有了“我的位置”功能，甚至都不需要输入出发点。
- ❑ 公交换乘：查看公交和地铁线路，确定转车路线，制定你在全球 80 多个城市的出行计划。“公交换乘”功能目前适用于黑莓、Windows Mobile、S60 和其他支持 Java 的手机。
- ❑ 路况信息：Google 地图中的公路会根据实时路况数据，以绿色、黄色或红色显示。
- ❑ 收藏夹：为你常去的地方加上书签，以便能在地图上非常方便地返回到这些地方。

大家不要认为这些功能在手机上很难实现，尤其是在我们要学习的 Android 平台中，要实现这些功能是非常简单的，只需要使用 Android Maps API（地图 API）和 Android Location API（定位 API）即可。下面我们将学习如何来使用这些 API 开发自己的地图应用。

9.3.2 准备工作

在 Android SDK 1.5 预装的 add-on 中提供了一个 Map 扩展库 com.google.android.maps，利用它就可以给 android 应用程序加上强大的地图功能了。这个库的位置是“Android SDK 路径”\add-ons\google_apis-3\libs。需要说明的是，这个库并不是标准的 Android sdk 的内容，可以自己从这个位置下载，并放到你的 sdk 中，这样就可以为你新建的应用或者已有的应用加上地图功能了。在使用 Android Map API 之前，还需要申请一个 Android Map API Key。

1. 申请 Android Map API Key

为了能顺利地申请 Android Map API Key，必须要准备 Google 的账号和系统的证明书。一般 Google 发布 Key 都需要 Google 的账号，Google 的账号是通用的，Gmail 的账号就可以了（没有的话可以到 <http://www.google.com/> 去申请一个）。当一个应用程序发布时必须要有证明书，证明书其实就是 MD5。我们这里不是发布，而只是为了测试，可以使用 Debug 版的证明书。下面我们来学习如何申请 Android Map API Key。

步骤 1：找到你的 debug.keystore 文件。

证书的一般路径为：C:\Documents and Settings\当前用户\Local Settings\Application Data\Android\debug.keystore。当然我们使用 Eclipse 开发，便可以打开 Eclipse 选择 Windows→Preference→Android→Build，其中 Default debug keystore 的值便是 debug.keystore 的路径，如图 9-8 所示。

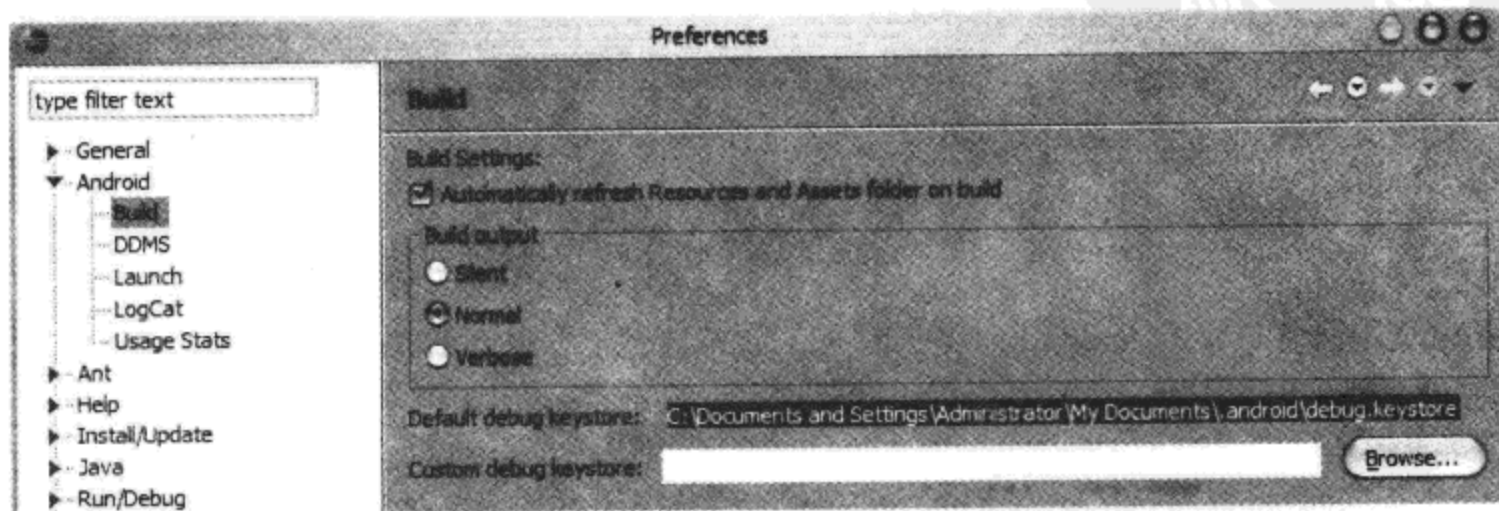


图 9-8 debug.keystore 文件的位置

步骤 2: 取得 debug.keystore 的 MD5 值。

首先在命令提示符下进入 debug.keystore 文件所在的路径, 执行命令: `keytool -list -keystore debug.keystore`, 这时可能会提示你输入密码, 这里输入默认的密码 “android”, 即可取得 MD5 值, 如图 9-9 所示。



图 9-9 取得 debug.keystore 的 MD5 值

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>, 登录 Google 账号, 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, 选中 “I have read and agree with the terms and conditions” 选项, 如图 9-10 所示, 按下 “Generate API Key” 按钮, 即可得到我们申请到的 API Key。

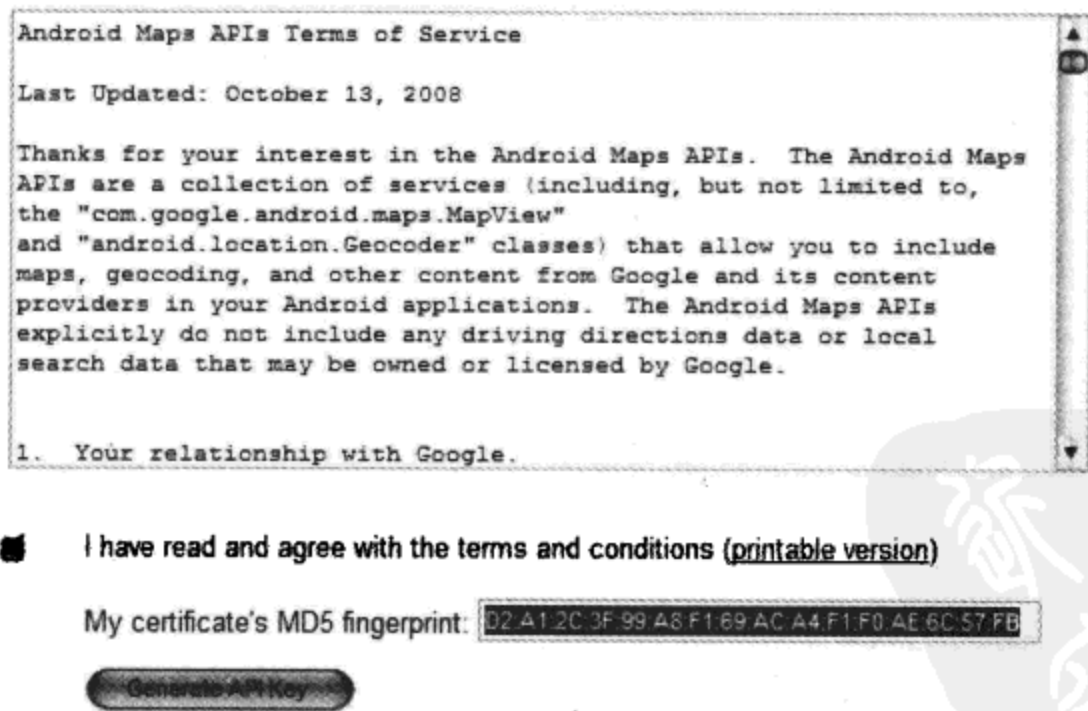


图 9-10 Android Map API Key 申请页面

到这里我们便完成了 Android Map API Key 的申请, 记下申请的 Android Map API Key 值, 在后面的应用程序中会用到它。下面我们还将创建一个基于 Google APIs 的 AVD。

2. 创建基于 Google APIs 的 AVD

在 Eclipse 中打开 AVD 管理界面, 在 “Create AVD” 部分的 Name 处填写 AVD 的名字, 在 Target 处选择 “Google APIs-1.5”, 如图 9-11 所示, 点击 “Create AVD” 按钮完成创建。

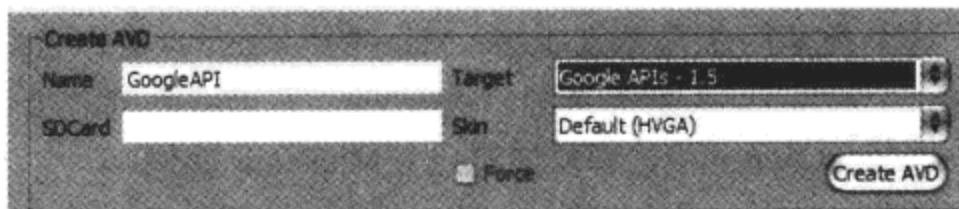


图 9-11 创建 AVD

3. 创建基于 Google APIs 的工程

这里需要注意的是，由于我们需要使用 Google APIs，所以在创建工程时，在 Build Target 处需要选择 Google APIs，如图 9-12 所示，其他选项和以前一样。当然，在运行工程时也就需要选择我们刚刚创建的基于 Google APIs 的 AVD 来运行。

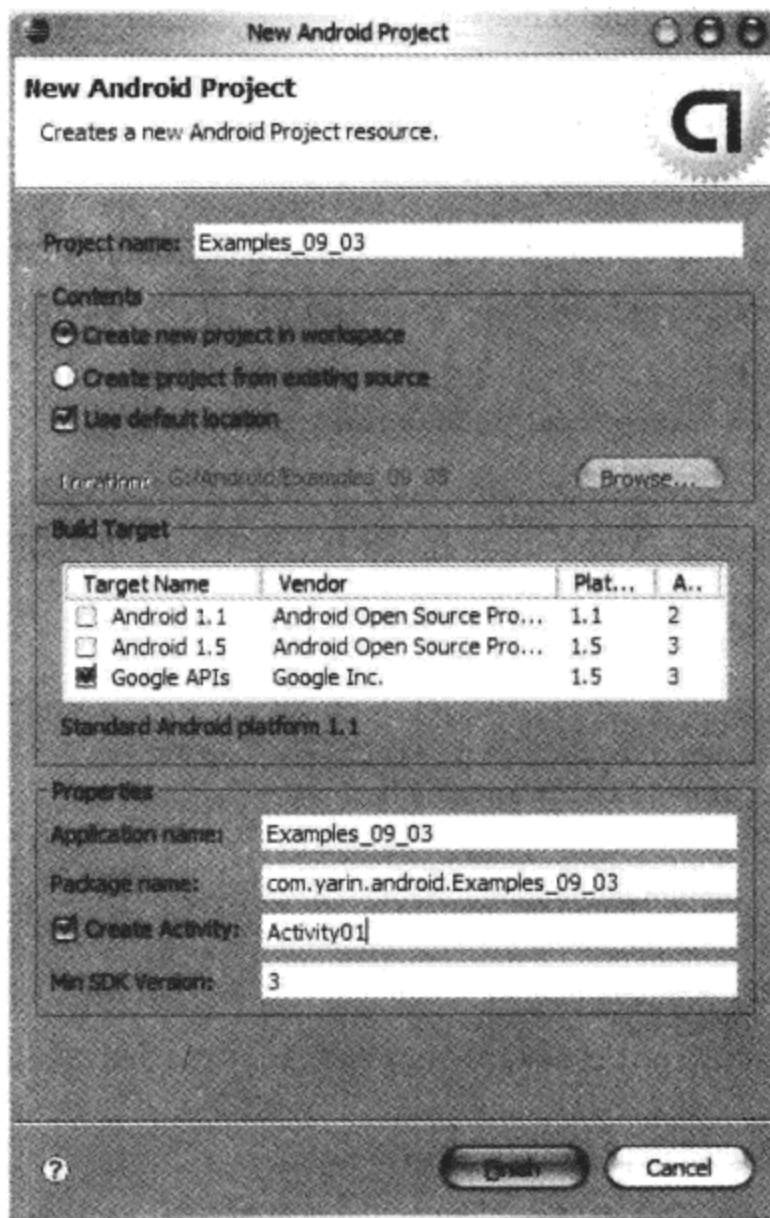


图 9-12 创建基于 Google APIs 的工程

到这里，我们基本完成了准备工作，下面我们将开始学习如何使用 Google API 来开发地图应用程序。

9.3.3 Google Map API 的使用

Android 中定义了一个名为 `com.google.android.maps` 的包，其中包含了一系列用于在 Google Map 上显示、控制和层叠信息的功能类，以下是该包中最重要的几个类：

- ❑ **MapActivity**: 这个类是用于显示 Google Map 的 Activity 类, 它需要连接底层网络。MapActivity 是一个抽象类, 任何想要显示 MapView 的 activity 都需要派生自 MapActivity, 并且在其派生类的 onCreate() 中, 都要创建一个 MapView 实例。
- ❑ **MapView**: MapView 是用于显示地图的 View 组件。它派生自 android.view.ViewGroup。它必须和 MapActivity 配合使用, 而且只能被 MapActivity 创建, 这是因为 MapView 需要通过后台的线程来连接网络或者文件系统, 而这些线程要由 MapActivity 来管理。
- ❑ **MapController**: MapController 用于控制地图的移动、缩放等。
- ❑ **Overlay**: 这是一个可显示于地图之上的可绘制的对象。
- ❑ **GeoPoint**: 这是一个包含经纬度位置的对象。

下面我们将使用 com.google.android.maps 包来实现一个地图浏览程序 (见本书所附代码: 第 9 章\Examples_09_03)。

步骤 1: 创建工程, 注意要选择的 Build Target 为 “Google APIs”。

步骤 2: 修改 AndroidManifest.xml 文件。

由于我们要使用 Google Map API, 所以必须先先在 AndroidManifest.xml 中定义如下信息: `<uses-library android:name="com.google.android.maps">`, 当然要从网络获取地图数据, 还需要添加应用程序访问网络的权限。代码如下:

```
<uses-library android:name="com.google.android.maps" />
```

步骤 3: 创建 MapView。

要显示地图, 需要创建一个 MapView, 在 XML 文件中的布局如代码清单 9-3 所示。其中 android:apiKey 的值便是我们申请的 Android Map API Key。

代码清单 9-3 main.xml

```
<?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"
    >
    <com.google.android.maps.MapView
        android:id="@+id/MapView01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:apiKey="0dYpmIXGIIdwiVm-HEpzuUW2fjNYsFQ9EvYirlsg"/>
    </RelativeLayout>
```

当然, 可以在程序中通过如下代码创建 MapView。

```
MapView map = new MapView(this, "[Android Maps API Key]");
```

步骤 4: 实现 MapActivity。

MapView 需要由 MapActivity 来管理, 所以程序部分应该继承自 MapActivity 类, 必须实现 isRouteDisplayed 方法。MapView 提供了 3 种模式的地图, 分别可以通过以下方式设置采用什么模式来显示地图。另外, 可以通过 setBuiltInZoomControls 方法设置地图是否支持缩放。

```
//设置为交通模式
//mMapView.setTraffic(true);
//设置为卫星模式
//mMapView.setSatellite(true);
//设置为街景模式
```

```
//mMapView.setStreetView(false);
```

步骤 5: MapController 的使用。

如果要设置地图显示的地点以及放大的倍数等,就需要使用 MapController 来控制地图。可以通过如下代码获得 MapController 对象:

```
mMapController = mMapView.getController();
```

要定位地点,需要构建一个 GeoPoint 来表示地点的经度和纬度,然后使用 animateTo 方法将地图定位到指定的 GeoPoint 上,代码如下:

```
//设置起点为成都
mGeoPoint=new GeoPoint((int)(30.659259*1000000),(int)(104.065762*1000000));
//定位到成都
mMapController.animateTo(mGeoPoint);
```

步骤 6: Overlay 的使用。

如果需要在地图上标注一些图标文字等信息,就需要使用 Overlay。这里我们首先要将地图上的经度和纬度转换成屏幕上实际的坐标,才能将信息绘制上去。Map API 中提供了 Projection.toPixels(GeoPoint in, Point out)方法,可以将经度和纬度转换成屏幕上的坐标。首先需要实现 Overlay 中的 draw 方法才能在地图上绘制信息,代码如下:

```
class MyLocationOverlay extends Overlay
{
    public boolean draw(Canvas canvas,MapView mapView,boolean shadow,long when)
    {
        //...
    }
}
```

下面是示例运行效果,图 9-13 以交通模式显示地图,图 9-14 以卫星模式显示地图,它们都在屏幕上显示了一个图标,并标明了位置。



图 9-13 交通模式地图

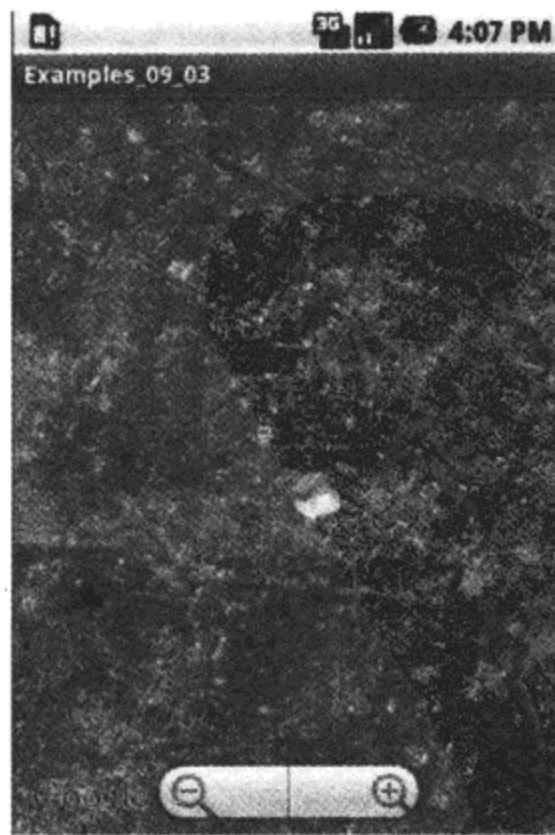


图 9-14 卫星模式地图

该示例显示了成都市区的地图，标注了天府广场的大概位置，具体实现如代码清单 9-4 所示。

代码清单 9-4 第 9 章\Examples_09_03\src\com\yarin\android\Examples_09_03\Activity01.java

```
public class Activity01 extends MapActivity
{
    private MapView    mMapView;
    private MapController mMapController;
    private GeoPoint mGeoPoint;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mMapView = (MapView) findViewById(R.id.MapView01);
        //设置为交通模式
        //mMapView.setTraffic(true);
        //设置为卫星模式
        mMapView.setSatellite(true);
        //设置为街景模式
        //mMapView.setStreetView(false);
        //取得 MapController 对象(控制 MapView)
        mMapController = mMapView.getController();
        mMapView.setEnabled(true);
        mMapView.setClickable(true);
        //设置地图支持缩放
        mMapView.setBuiltInZoomControls(true);
        //设置起点为成都
        mGeoPoint=new GeoPoint((int) (30.659259*1000000), (int) (104.065762*1000000));
        //定位到成都
        mMapController.animateTo(mGeoPoint);
        //设置倍数(1-21)
        mMapController.setZoom(12);
        //添加 Overlay, 用于显示标注信息
        MyLocationOverlay myLocationOverlay = new MyLocationOverlay();
        List<Overlay> list = mMapView.getOverlays();
        list.add(myLocationOverlay);
    }
    protected boolean isRouteDisplayed()
    {
        return false;
    }
    class MyLocationOverlay extends Overlay
    {
        public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)
        {
            super.draw(canvas, mapView, shadow);
            Paint paint = new Paint();
            Point myScreenCoords = new Point();
            // 将经纬度转换成实际屏幕坐标
            mapView.getProjection().toPixels(mGeoPoint, myScreenCoords);
            paint.setStrokeWidth(1);
            paint.setARGB(255, 255, 0, 0);
            paint.setStyle(Paint.Style.STROKE);
            Bitmap bmp = BitmapFactory.decodeResource(getResources(),
                R.drawable.home);
            canvas.drawBitmap(bmp, myScreenCoords.x, myScreenCoords.y, paint);
            canvas.drawText("天府广场", myScreenCoords.x, myScreenCoords.y,
```

```

        paint);
        return true;
    }
}

```

9.3.4 定位系统

全球定位系统 (Global Positioning System, GPS) 又称为全球卫星定位系统, 是一个中距离圆型轨道卫星导航系统, 它可以为地球表面的绝大部分地区 (98%) 提供准确的定位、测速和高精度的时间标准。该系统由美国国防部研制和维护, 可满足位于全球任何地方或近地空间的军事用户连续、精确地确定三维位置、三维运动和时间的需要。该系统包括太空中的 24 颗 GPS 卫星, 地面上的 1 个主控站、3 个数据注入站和 5 个监测站及作为用户端的 GPS 接收机。最少只需其中 3 颗卫星, 就能迅速确定用户端在地球上所处的位置及海拔高度。所能连接到的卫星数越多, 解码出来的位置就越精确。GPS 广泛应用于军事、物流、地理、移动电话、数码相机、航空等领域, 具有非常强大的功能, 主要包括:

- ☐ 精确定时: 广泛应用在天文台、通信系统基站、电视台中。
- ☐ 工程施工: 道路、桥梁、隧道的施工中大量采用 GPS 设备进行工程测量。
- ☐ 勘探测绘: 野外勘探及城区规划中都有用到。
- ☐ 导航。
 - ◆ 武器导航: 精确制导导弹、巡航导弹。
 - ◆ 车辆导航: 车辆调度、监控系统。
 - ◆ 船舶导航: 远洋导航、港口/内河引水。
 - ◆ 飞机导航: 航线导航、进场着陆控制。
 - ◆ 星际导航: 卫星轨道定位。
 - ◆ 个人导航: 个人旅游及野外探险。
- ☐ 定位。
 - ◆ 车辆防盗系统。
 - ◆ 手机、PDA、PPC 等通信移动设备防盗以及电子地图、定位系统。
 - ◆ 儿童及特殊人群的防走失系统。
- ☐ 精准农业: 农机具导航、自动驾驶以及土地高精度平整。

Android 支持地理定位服务的 API。该地理定位服务可以用来获取当前设备的地理位置, 应用程序可以定时请求更新设备当前的地理定位信息。比如应用程序可以借助一个 Intent 接收器来实现如下功能: 以经纬度和半径划定一个区域, 当设备出入该区域时, 发出提醒信息, 还可以和 Google Map API 一起使用, 完成更多的任务。关于地理定位系统的 API 全部位于 android.location 包内, 其中包括以下几个重要的功能类:

- ☐ LocationManager: 本类提供访问定位服务的功能, 也提供获取最佳定位提供者的功能。另外, 临近警报功能也可以借助该类来实现。
- ☐ LocationProvider: 该类是定位提供者的抽象类。定位提供者具备周期性报告设备地理位置的功能。

- ❑ **LocationListener**: 提供定位信息发生改变时的回调功能。必须事先在定位管理器中注册监听器对象。
- ❑ **Criteria**: 该类使得应用能够通过向 **LocationProvider** 中设置的属性来选择合适的定位提供者。
- ❑ **Geocoder**: 用于处理地理编码和反向地理编码的类。地理编码是指将地址或其他描述转变为经度和纬度, 反向地理编码则是将经度和纬度转变为地址或描述语言, 其中包含了两个构造函数, 需要传入经度和纬度的坐标。**getFromLocation** 方法可以得到一组关于地址的数组。

要使用地理定位, 首先需要取得 **LocationManager** 的实例, 在 Android 中, 获得 **LocationManager** 的唯一方法是通过 **getSystemService()** 方法的调用。通过使用 **LocationManager**, 我们可以获得一个位置提供者的列表。在一个真实的手持设备中, 这个列表包含了一些 GPS 服务。我们也可以选择更强大、更精确、不带有其他附加服务的 GPS。代码如下:

```
LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

取得 **LocationManager** 对象之后, 我们还需要注册一个周期性的更新视图, 代码如下:

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 0, locationManager);
```

其中第一个参数是设置服务提供者, 第二个参数是周期, 这里需要重点说明一下最后一个参数 **locationListener**, 它用来监听定位信息的改变, 所以我们必须实现以下几个方法:

- ❑ **onLocationChanged(Location location)**: 当坐标改变时触发此函数, 如果 **Provider** 传进相同的坐标, 它就不会被触发。
- ❑ **onProviderDisabled(String provider)**: **Provider** 禁用时触发此函数, 比如 GPS 被关闭。
- ❑ **onProviderEnabled(String provider)**: **Provider** 启用时触发此函数, 比如 GPS 被打开。
- ❑ **onStatusChanged(String provider, int status, Bundle extras)**: **Provider** 的转态在可用、暂时不可用和无服务三个状态直接切换时触发此函数。

下面我们通过更改上一节的例子 (本书所附代码: 第 9 章 \Examples_09_04) 来实现自动通过定位系统获取用户当前的坐标, 然后加载并显示地图, 将坐标信息显示在一个 **TextView** 中, 运行效果如图 9-15 所示。

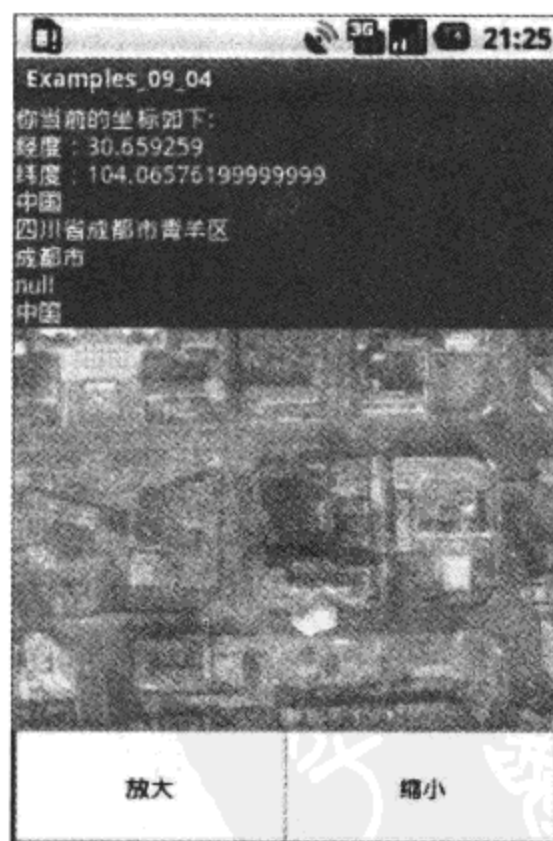


图 9-15 地图定位

要使用定位的 API, 首先需要在 **AndroidManifest.xml** 文件中添加其权限, 具体代码如代码清单 9-5 所示。

代码清单 9-5 第 9 章 \Examples_09_04 \AndroidManifest.xml

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

```

package="com.yarin.android.Examples_09_04"
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon" android:label="@string/app_name">
  <uses-library android:name="com.google.android.maps" />
  <activity android:name=".Activity01"
    android:label="@string/app_name">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
</application>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-sdk android:minSdkVersion="5" />
</manifest>

```

由于我们在模拟器上测试，所以需要人为设置一个坐标。可以通过两种方法来设置一个模拟的坐标值。第一种方法是通过 DDMS，我们可以在 Eclipse 的 ADT 插件中使用这种方法，只要启动 Eclipse，选择“Window”->“Show View”，打开“Emulator Control”界面即可看到如下的设置窗口，我们可以手动或者通过 KML 和 GPX 文件来设置一个坐标。如图 9-16 所示。

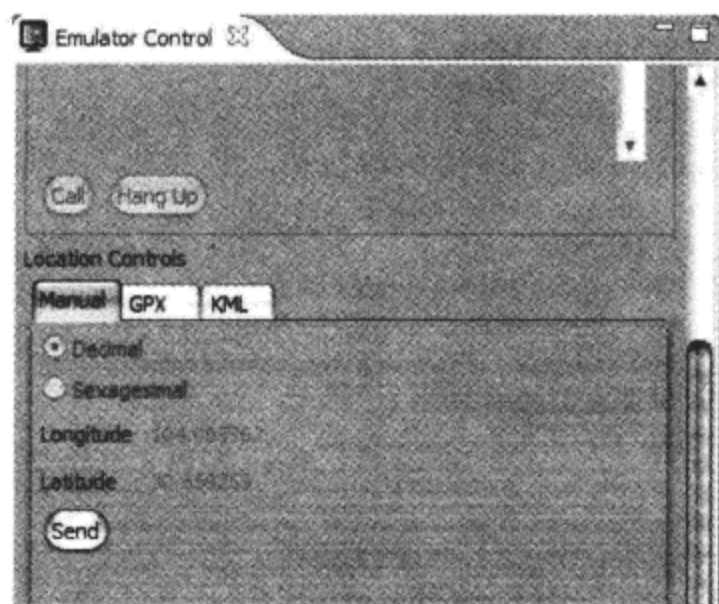


图 9-16 设置显示的坐标

另一种方法是使用 geo 命令，我们需要 telnet 到本机的 5554 端口，然后在命令行下输入类似于 geo fix-121.45356 46.51119 4392 这样的命令，后面 3 个参数分别代表了经度、纬度和（可选的）海拔。设置之后在 Android 模拟器屏幕上便多出一个如图 9-17 所示的标志，表示模拟了 GPS 权限。

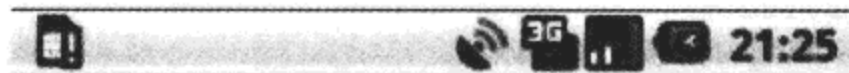


图 9-17 GPS 使用标志

现在我们可以使用位置管理器（LocationManager）和位置提供者进行 getFromLocation 的调用。这个方法返回本机当前位置的一个快照，这个快照将以 Location 对象形式提供。在手持设备中，我们可以获得当前位置的经度和纬度；调用 getFromLocationName 方法可能返回一个数据，表

示一个地方的名称。该例中我们还创建了一个菜单用来缩放地图，这时就使用地图控制器（MapController）的 zoomIn 和 zoomOut 方法来放大和缩小视图，具体实现如代码清单 9-6 所示。

代码清单 9-6 第 9 章\Examples_09_04\src\com\yarin\android\Examples_09_04\Activity01.java

```
public class Activity01 extends MapActivity
{
    public MapController mapController;
    public MyLocationOverlay myPosition;
    public MapView myMapView;
    private static final int ZOOM_IN=Menu.FIRST;
    private static final int ZOOM_OUT=Menu.FIRST+1;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //取得 LocationManager 实例
        LocationManager locationManager;
        String context=Context.LOCATION_SERVICE;
        locationManager=(LocationManager) getSystemService(context);
        myMapView=(MapView) findViewById(R.id.MapView01);
        //取得 MapController 实例，控制地图
        mapController=myMapView.getController();
        //设置显示模式
        myMapView.setSatellite(true);
        myMapView.setStreetView(true);
        //设置缩放控制，这里我们自己实现缩放菜单
        myMapView.displayZoomControls(false);
        //设置使用 MyLocationOverlay 来绘图
        mapController.setZoom(17);
        myPosition=new MyLocationOverlay();
        List<Overlay> overlays=myMapView.getOverlays();
        overlays.add(myPosition);
        //设置 Criteria（服务商）的信息
        Criteria criteria =new Criteria();
        //经度要求
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(false);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        //取得效果最好的 criteria
        String provider=locationManager.getBestProvider(criteria, true);
        //得到坐标相关的信息
        Location location=locationManager.getLastKnownLocation(provider);
        //更新坐标
        updateWithNewLocation(location);
        //注册一个周期性的更新，3000ms 更新一次
        //locationListener 用来监听定位信息的改变
        locationManager.requestLocationUpdates(provider, 3000, 0,locationListener);
    }
    private void updateWithNewLocation(Location location)
    {
        String latLongString;
        TextView myLocationText = (TextView) findViewById(R.id.TextView01);
```

```

String addressString="没有找到地址\n";

if(location!=null)
{
    //为绘制标志的类设置坐标
    myPosition.setLocation(location);
    //取得经度和纬度
    Double geoLat=location.getLatitude()*1E6;
    Double geoLng=location.getLongitude()*1E6;
    //将其转换为 int 型
    GeoPoint point=new GeoPoint(geoLat.intValue(),geoLng.intValue());
    //定位到指定坐标
    mapController.animateTo(point);
    double lat=location.getLatitude();
    double lng=location.getLongitude();
    latLongString="经度: "+lat+"\n 纬度: "+lng;

    double latitude=location.getLatitude();
    double longitude=location.getLongitude();
    //根据地理环境来确定编码
    Geocoder gc=new Geocoder(this,Locale.getDefault());
    try
    {
        //取得地址相关的一些信息、经度、纬度
        List<Address> addresses=gc.getFromLocation(latitude, longitude,1);
        StringBuilder sb=new StringBuilder();
        if(addresses.size()>0)
        {
            Address address=addresses.get(0);
            for(int i=0;i<address.getMaxAddressLineIndex();i++)
                sb.append(address.getAddressLine(i)).append("\n");

            sb.append(address.getLocality()).append("\n");
            sb.append(address.getPostalCode()).append("\n");
            sb.append(address.getCountryName());
            addressString=sb.toString();
        }
    }catch(IOException e){}
}
else
{
    latLongString="没有找到坐标.\n";
}
//显示
myLocationText.setText("你当前的坐标如下:\n"+latLongString+"\n"+addressString);
}
private final LocationListener locationListener=new LocationListener()
{
    //当坐标改变时触发此函数
    public void onLocationChanged(Location location)
    {
        updateWithNewLocation(location);
    }
    //Provider 禁用时触发此函数, 比如 GPS 被关闭
    public void onProviderDisabled(String provider)
    {
        updateWithNewLocation(null);
    }
}

```



```

    }
    //Provider 启用时触发此函数, 比如 GPS 被打开
    public void onProviderEnabled(String provider){}
    //Provider 的转态在可用、暂时不可用和无服务三个状态直接切换时触发此函数
    public void onStatusChanged(String provider,int status,Bundle extras){}
};
protected boolean isRouteDisplayed()
{
    return false;
}
//为应用程序添加菜单
public boolean onCreateOptionsMenu(Menu menu)
{
    super.onCreateOptionsMenu(menu);
    menu.add(0, ZOOM_IN, Menu.NONE, "放大");
    menu.add(0, ZOOM_OUT, Menu.NONE, "缩小");
    return true;
}
public boolean onOptionsItemSelected(MenuItem item)
{
    super.onOptionsItemSelected(item);
    switch (item.getItemId())
    {
        case (ZOOM_IN):
            //放大
            mapController.zoomIn();
            return true;
        case (ZOOM_OUT):
            //缩小
            mapController.zoomOut();
            return true;
    }
    return true;
}

class MyLocationOverlay extends Overlay
{
    Location mLocation;
    //在更新坐标时, 设置该坐标, 以便画图
    public void setLocation(Location location)
    {
        mLocation = location;
    }
    @Override
    public boolean draw(Canvas canvas,MapView mapView,boolean shadow,long when)
    {
        super.draw(canvas, mapView, shadow);
        Paint paint = new Paint();
        Point myScreenCoords = new Point();
        // 将经纬度转换成实际屏幕坐标
        GeoPoint tmpGeoPoint = new GeoPoint((int)(mLocation.
            getLatitude()*1E6), (int)(mLocation.getLongitude()*1E6));
        mapView.getProjection().toPixels(tmpGeoPoint,myScreenCoords);
        paint.setStrokeWidth(1);
        paint.setARGB(255, 255, 0, 0);
        paint.setStyle(Paint.Style.STROKE);
        Bitmap bmp = BitmapFactory.decodeResource(getResources(),

```



```

        R.drawable.home);
        canvas.drawBitmap(bitmap, myScreenCoords.x, myScreenCoords.y, paint);
        canvas.drawText("Here am I", myScreenCoords.x, myScreenCoords.y, paint);
        return true;
    }
}

```

9.4 桌面组件

第一次启动 Android 模拟器时，可以看到在桌面上有很多图标，如图 9-18 所示的 Google 搜索框、时钟、联系人、浏览器等，点击这些图标，系统就会执行相应的程序，与 PC 操作系统桌面上的快捷方式很像，但是它不完全是快捷方式，还包括了实时文件夹（Live Folder）和桌面插件（Widget），这样既美观又方便用户操作。本节将学习这每一种桌面组件的开发，让我们自己的应用程序也能轻松地放置到桌面上。

9.4.1 快捷方式

首先我们学习最基本的桌面组件快捷方式，它和 PC 上的快捷方式一样，用于启动某一应用程序的某个组件（如 Activity、Service 等）。其实要在桌面上添加一个快捷方式很简单，只需要长按桌面或者点击“Menu”按钮（如图 9-19 所示），就可以弹出添加桌面组件的选项，如图 9-20 所示，“Shortcuts”为添加快捷方式，“Widgets”为 Widget 开发的桌面插件，“Folders”为实时文件夹，进入相应的选项后即可添加相应的桌面组件。



图 9-18 Android 桌面组件

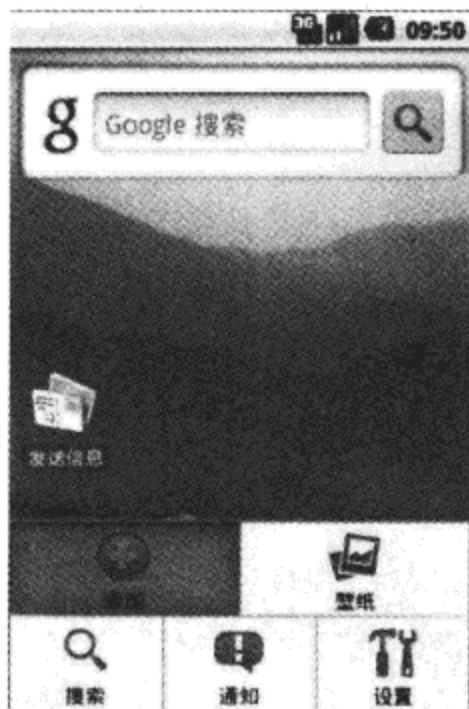


图 9-19 Menu 菜单



图 9-20 添加桌面组件

本小节重点介绍在应用程序中通过代码来将一个应用程序添加到图 9-20 的 Shortcuts 列表中，这里添加一个发送邮件的应用到快捷方式列表上去（参见本书所附代码：第 9 章 \Examples_09_05）。

首先需要在 Activity 注册时添加一个 Action 为 `android.intent.action.CREATE_SHORTCUT` 的 `IntentFilter`，如代码清单 9-7 所示，添加之后列表中就会出现该应用的图标和名字了。

代码清单 9-7 第 9 章 \Examples_09_05 \AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yarin.android.Examples_09_05"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Activity01"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                <action android:name="android.intent.action.CREATE_SHORTCUT"/>
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>
```

接下来还要为快捷方式设置名字、图标、事件等属性。`Intent.EXTRA_SHORTCUT_NAME` 对应快捷方式的名字；`Intent.EXTRA_SHORTCUT_ICON_RESOURCE` 对应快捷方式的图标；`Intent.EXTRA_SHORTCUT_INTENT` 对应快捷方式执行的事件。需要说明的是，Android 专门提供了 `Intent.ShortcutIconResource.fromContext` 来创建快捷方式的图标，最后通过 `setResult` 来返回，构建一个快捷方式，如代码清单 9-8 所示。

代码清单 9-8 第 9 章 \Examples_09_05 \src \com \yarin \android \Examples_09_05 \Activity01.java

```
public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //要添加的快捷方式的 Intent
        Intent addShortcut;
        //判断是否要添加快捷方式
        if (getIntent().getAction().equals(Intent.ACTION_CREATE_SHORTCUT))
        {
            addShortcut = new Intent();
            //设置快捷方式的名字
            addShortcut.putExtra(Intent.EXTRA_SHORTCUT_NAME, "发送邮件");
            //构建快捷方式中专门的图标
            Parcelable icon = Intent.ShortcutIconResource.fromContext(
                this, R.drawable.mail_edit);
            //添加快捷方式图标
            addShortcut.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, icon);
            //构建快捷方式执行的 Intent
            Intent mailto=new Intent(Intent.ACTION_SENDTO, Uri.parse(
                "mailto:xxx@xxx.com" ));
```

```

        //添加快捷方式 Intent
        addShortcut.putExtra(Intent.EXTRA_SHORTCUT_INTENT,mailto);
        //正常
        setResult(RESULT_OK,addShortcut);
    }
    else
    {
        //取消
        setResult(RESULT_CANCELED);
    }
    //关闭
    finish();
}
}

```

现在我们启动模拟器，就可以在 Shortcuts 列表中找到所添加的快捷方式，将其添加到桌面，如图 9-21 所示。



图 9-21 桌面快捷方式

9.4.2 实时文件夹

在 Android 1.5 中，Live Folders 无疑是一个备受关注的功能。简单地说，Live Folders 就是一个查看你的手机中所有电子书、电子邮件、rss 订阅、播放列表的快捷方式，并且这些内容是实时更新的。比如你不再需要单独打开电子邮件软件查看邮件，打开通讯录找联系人等。Live Folders 自带了列出所有联系人、所有有电话号码的联系人以及 Starred 联系人的功能，我们还可以使用 Live Folders API 开发出更多的新颖应用。

由于 Live Folders 本身不存储任何信息，都是以映射的方式查看其 ContentProvider 所指向的数据信息，并可以自定义显示格式，所以当源数据发生改变后，Live Folders 可以实时更新显示内容。那么在开发时，我们要确保所指定数据信息 URI 的 ContentProvider 支持实时文件夹的查询。其添加方式和添加快捷方式一样，只是在选择时要选择“Folders”。本小节我们通过 Live Folders 调用电话本中的信息，当点击其中一条信息时，便执行呼叫该联系人的动作（本书所附代码：第 9 章\Examples_09_06）。

和创建快捷方式一样，我们需要在 Activity 注册时添加一个 Action 动作为 android.intent.action.CREATE_LIVE_FOLDER 的 IntentFilter。代码如下：

```

<intent-filter>
<action android:name= "android.intent.action.CREATE_LIVE_FOLDER" />
<category android:name= "android.intent.category.DEFAULT" />
</intent-filter>

```

我们需要在程序中设置该实时文件夹的数据源、图标、名字的信息。可以通过 `intent.setData` 方法来设置要读取的数据源，该例中我们设置数据源为“`content://contacts/live_folders/people`”，即联系人信息。其他信息的设置如表 9-2 所示。

表 9-2 Live Folders 的常用属性

属 性	描 述
EXTRA_LIVE_FOLDER_BASE_INTENT	选中选项之后执行的事件
EXTRA_LIVE_FOLDER_NAME	实时文件夹的名字
EXTRA_LIVE_FOLDER_ICON	实时文件夹的图标
EXTRA_LIVE_FOLDER_DISPLAY_MODE	实时文件夹的显示模式（列表和宫格）

在设置图标时，Android 专门提供了 `Intent.ShortcutIconResource.fromContext` 来设置实时文件夹的图标。下面我们将实时文件夹添加到桌面（如图 9-22 所示），运行效果如图 9-23 所示。



图 9-22 “电话本”实时文件夹

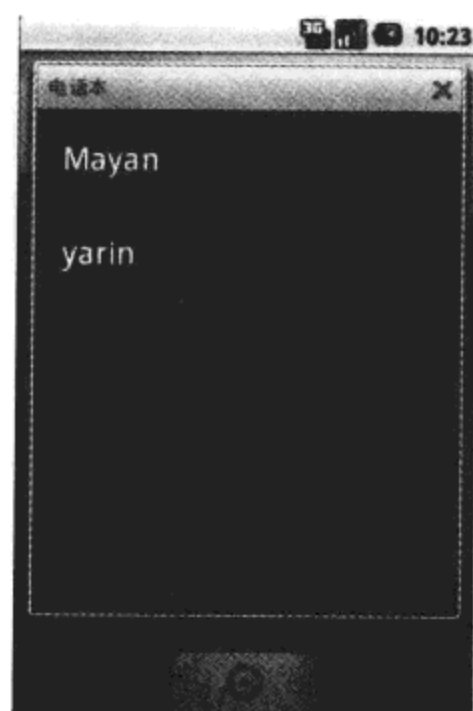


图 9-23 实时文件夹运行效果

下面需要在 `onCreate` 方法中将实时文件夹的相关信息装入 `Intent` 对象，并通过 `setResult` 方法设置为结果 `Intent`，最后调用 `finish` 方法结束 `Activity`，把结果返回给 `Home` 应用程序，以添加实时文件夹，如代码清单 9-9 所示。

代码清单 9-9 第 9 章\Examples_09_06\src\com\yarin\android\Examples_09_06\Activity01.java

```

public class Activity01 extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // setContentView(R.layout.main);
    }
}

```

```

// 判断是否创建实时文件夹
if (getIntent().getAction().equals(LiveFolders.ACTION_CREATE_LIVE_FOLDER))
{
    Intent intent = new Intent();
    // 设置数据地址
    intent.setData(Uri.parse("content://contacts/live_folders/people"));
    // 设置单击之后的事件, 这里单击一个联系人后, 呼叫
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_BASE_INTENT,
        new Intent(Intent.ACTION_CALL, Contacts.People.CONTENT_URI));
    // 设置实时文件夹的名字
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, "电话本");
    // 设置实施文件夹的图标
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON, Intent.
        ShortcutIconResource.fromContext(this, R.drawable.contacts));
    // 设置显示模式为列表
    intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE,
        LiveFolders.DISPLAY_MODE_LIST);
    // 完成
    setResult(RESULT_OK, intent);
}
else
{
    setResult(RESULT_CANCELED);
}
finish();
}
}

```

9.4.3 Widget 开发

Widget 是一种很小的应用程序, 主要作为 Web 2.0 服务或互联网内容的前端。Web 设计人员与开发者可以使用 Widget 来创造最受欢迎的互联网体验。在 Android 1.5 中加入了 AppWidget framework 框架, 开发者可以使用该框架开发 Widget, 这些 Widget 可以拖到用户的桌面并且可以交互。Widget 可以提供一个 full-featured apps 的预览, 例如可以显示即将到来的日历事件, 或者一首后台播放的歌曲的详细信息。当 Widget 被拖到桌面上时, 指定一个保留的空间来显示应用提供的自定义内容。用户可以通过这个 Widget 来和应用交互, 例如暂停或切换歌曲。如果你有一个后台服务, 可以按照你自己的 Schedule 更新你的 Widget, 或者使用 AppWidget framework 提供一个自动的更新机制。

每个 Widget 就是一个 BroadcastReceiver, 它们用 XML metadata 来描述 Widget 的细节。AppWidget framework 通过 Broadcast intents 和 Widget 通信, Widget 的更新使用 RemoteViews 来发送。RemoteViews 被包装成一个 layout 和特定内容来显示到桌面上。下面我们通过一个示例来学习 Widget 开发 (本书所附代码: 第 9 章\Examples_09_07)。

首先需要在 res\layout 目录下创建桌面组件的布局文件 appwidget_provider.xml, 用来显示桌面布局, 这里我们创建一个 TextView 用来显示一段文字, 如代码清单 9-10 所示。

代码清单 9-10 第 9 章\Examples_09_07\res\layout\appwidget_provider.xml

```

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

```

```

        android:id="@+id/appwidget_text"
        android:textColor="#ff000000"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />

```

然后需要创建一个描述这个桌面组件属性的文件，存放到 `res/xml` 文件夹下，如代码清单 9-11 所示。

代码清单 9-11 第 9 章\Examples_09_07\res\xml\appwidget_provider.xml

```

<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="100dp"
    android:minHeight="50dp"
    android:updatePeriodMillis="86400000"
    android:initialLayout="@layout/appwidget_provider"
    android:configure="com.yarin.android.Examples_09_07.Activity01"
    >
</appwidget-provider>

```

其中 `android:minWidth` 和 `android:minHeight` 分别指定了桌面组件的最小宽度和最小高度，其值可以根据期望的单元格数量并使用前面介绍过的公式来计算（最小尺寸 = （单元格数 × 74） - 2），`android:updatePeriodMillis` 是自动更新的时间间隔，`android:initialLayout` 是 Widget 的界面描述文件。`Android:configure` 是可选的，如果你的 Widget 需要在启动前先启动一个 Activity，则需要设定该项为你的 Activity。这里我们需要先输入一段文字，然后显示在 Widget 上。

然后要建立一个 Widget，创建一个类，让其继承类 `AppWidgetProvider`。在 `AppWidgetProvider` 中有许多方法，包括 `onUpdate`（周期更新时调用）、`onDeleted`（删除组件时调用）、`onEnabled`（当第一个组件创建时调用）、`onDisabled`（当最后一个组件删除时调用），如代码清单 9-12 所示。

代码清单 9-12 第 9 章\Examples_09_07\src\com\yarin\android\Examples_09_07\ExampleAppWidgetProvider.java

```

public class ExampleAppWidgetProvider extends AppWidgetProvider
{
    //周期更新时调用
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[]
appWidgetIds)
    {
        final int N = appWidgetIds.length;
        for (int i = 0; i < N; i++)
        {
            int appWidgetId = appWidgetIds[i];
            String titlePrefix=Activity01.loadTitlePref(context,appWidgetId);
            updateAppWidget(context, appWidgetManager, appWidgetId,
titlePrefix);
        }
    }
    //当桌面组件删除时调用
    public void onDeleted(Context context, int[] appWidgetIds)
    {
        //删除 appWidget
        final int N = appWidgetIds.length;

```



```

        for (int i = 0; i < N; i++)
        {
            Activity01.deleteTitlePref(context, appWidgetIds[i]);
        }
    }
    //当 AppWidgetProvider 提供的第一个组件创建时调用
    public void onEnabled(Context context)
    {
        PackageManager pm = context.getPackageManager();
        pm.setComponentEnabledSetting(new ComponentName("com.yarin.android.
            Examples_09_07", ".ExampleBroadcastReceiver"),
            PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
            PackageManager.DONT_KILL_APP);
    }
    //当 AppWidgetProvider 提供的最后一个组件删除时调用
    public void onDisabled(Context context)
    {
        PackageManager pm = context.getPackageManager();
        pm.setComponentEnabledSetting(new ComponentName("com.yarin.
            android.Examples_09_07", ".ExampleBroadcastReceiver"),
            PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
            PackageManager.DONT_KILL_APP);
    }
    //更新
    static void updateAppWidget(Context context, AppWidgetManager
        appWidgetManager, int appWidgetId, String titlePrefix)
    {
        //构建 RemoteViews 对象来对桌面组件进行更新
        RemoteViews views = new RemoteViews(context.getPackageName(),
            R.layout.appwidget_provider);
        //更新文本内容, 指定布局的组件
        views.setTextViewText(R.id.appwidget_text, titlePrefix);
        //将 RemoteViews 的更新传入 AppWidget 进行更新
        appWidgetManager.updateAppWidget(appWidgetId, views);
    }
}

```

其中, 在 `updateAppWidget` 方法中我们构建了一个 `RemoteViews` 对象来对桌面组件进行更新, 通过 `setTextViewText` 方法来更新一个文本的显示, 然后通过 `updateAppWidget` 方法来将更新提供给 `AppWidget` 使其更新到桌面。在 `onDisabled` 和 `onEnabled` 方法中我们用 `ComponentName` 来表示应用程序中某个组件的完整名字。

最后, 创建一个 `BroadcastReceiver` 类来接收更新的信息, 在收到更新的信息之后就更新这个桌面 `Widget` 组件, 如代码清单 9-13 所示。

代码清单 9-13 第 9 章\Examples_09_07\src\com\yarin\android\Examples_09_07\ExampleBroadcastReceiver.java

```

public class ExampleBroadcastReceiver extends BroadcastReceiver
{
    public void onReceive(Context context, Intent intent)
    {
        //通过 BroadcastReceiver 来更新 AppWidget
        String action = intent.getAction();
        if (action.equals(Intent.ACTION_TIMEZONE_CHANGED) || action.equals
            (Intent.ACTION_TIME_CHANGED))
        {

```



```

AppWidgetManager gm = AppWidgetManager.getInstance(context);
ArrayList<Integer> appWidgetIds = new ArrayList<Integer>();
ArrayList<String> texts = new ArrayList<String>();
Activity01.loadAllTitlePrefs(context, appWidgetIds, texts);
//更新所有 AppWidget
final int N = appWidgetIds.size();
for (int i = 0; i < N; i++)
{
    ExampleAppWidgetProvider.updateAppWidget(context,
                                              gm, appWidgetIds.get(i), texts.get(i));
}
}
}

```

接下来，处理 `Android:configure` 指定的类，用来输入信息，在该类中我们监听这个按钮，当点击按钮之后，创建一个 `AppWidgetManager` 实例，然后调用 `ExampleAppWidgetProvider.updateAppWidget` 方法来更新这个 Widget，通过以下代码可以取得一个 `AppWidgetManager` 实例：

```
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

注意，还需要在 `AndroidManifest.xml` 中注册 `AppWidget`、`BroadcastReceiver` 和用来输入信息的 `Activity`，如代码清单 9-14 所示。

代码清单 9-14 第 9 章\Examples_09_07\AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yarin.android.Examples_09_07"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver android:name=".ExampleAppWidgetProvider">
            <meta-data android:name="android.appwidget.provider"
                android:resource="@xml/appwidget_provider" />
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
        </receiver>
        <activity android:name=".Activity01">
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
            </intent-filter>
        </activity>
        <receiver android:name=".ExampleBroadcastReceiver" android:enabled="false">
            <intent-filter>
                <action android:name="android.intent.ACTION_TIMEZONE_CHANGED" />
                <action android:name="android.intent.ACTION_TIME" />
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>

```

下面将该 Widget 添加到桌面上，和添加快捷方式一样，如图 9-24 所示，然后输入要显示的文字，如图 9-25 所示，点击“确定”按钮之后，桌面即显示我们输入的信息，如图 9-26 所示。



图 9-24 添加 Widget 到桌面

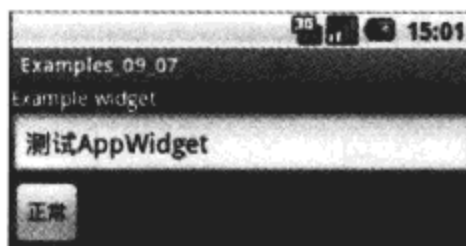


图 9-25 输入要显示的信息



图 9-26 桌面显示 Widget

9.5 账户管理

Android 2.0 中加入了一个新的包 `android.accounts`，该包主要包括了集中式的账户管理 API，用以安全地存储和访问认证的令牌和密码，比如，我们的手机存在多个账户，每个账户下面都有不同的信息，甚至每个账户都可以与不同的服务器之间进行数据同步（例如，手机账户中的联系人可以是一个 Gmail 账户中的通讯录，可联网进行同步更新）。下面首先来看看 `android.accounts` 包提供了哪些功能，如表 9-3 所示。

表 9-3 `android.accounts` 包的功能描述

功 能	描 述
<code>AccountManagerCallback</code>	账户管理回调接口
<code>AccountManagerFuture</code>	代表一个 <code>AccountManager</code> 的异步调用结果
<code>OnAccountsUpdateListener</code>	账户监听回调接口
<code>AbstractAccountAuthenticator</code>	创建 <code>AccountAuthenticators</code> （账户验证）的一个基类
<code>Account</code>	<code>AccountManager</code> 中的一个账户信息及类型
<code>AccountAuthenticatorActivity</code>	用于实现一个 <code>AbstractAccountAuthenticator</code> 的活动
<code>AccountAuthenticatorResponse</code>	账户验证响应
<code>AccountManager</code>	账户管理器
<code>AuthenticatorDescription</code>	一个 <code>Parcelable</code> 类型的值包括了账户验证的信息

光看这些介绍，也许会难以理解，下面我们结合一个示例程序来学习 `android.accounts` 包中各功能的使用。该示例实现了账户添加功能，可以添加多个账户来集中管理，程序运行界面如图 9-27 所示，点击“新建账户”按钮后，就可以添加账户的相关信息，如图 9-28 所示。程序的具体实现请参见本书所附代码：第 9 章\Examples_09_08。

该示例中一共新建了 4 个账户，因此在退出程序、点击新建联系人时，会出现如图 9-29 所示的界面来提示用户选择在哪一个账户中创建联系人，这样使得每个账户独立隔开，又统一管理，非常方便。

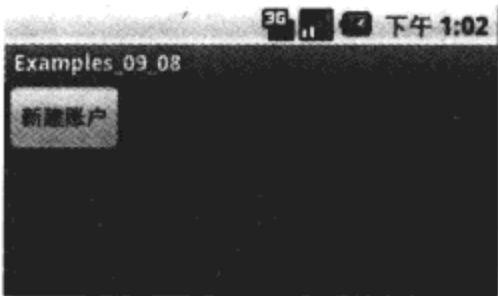


图 9-27 账户列表



图 9-28 新建账户界面



图 9-29 新建联系人

由于在该示例中对用户账户信息进行了操作，因此首先要确保在 `AndroidManifest.xml` 文件中对操作权限进行声明，以及确定 API 等级为 5，代码如下：

```
<uses-sdk android:minSdkVersion="5"/>
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS"/></uses-permission>
<uses-permission android:name="android.permission.ACCOUNT_MANAGER"/></uses-permission>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/></uses-permission>
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"/></uses-permission>
```

该示例的 UI 界面布局设计很简单，大家可以参考本书第 4 章的内容。首先来看一下如图 9-27 所示的 `Activity` 类的实现，我们需要通过 `AccountManager` 类的 `get` 方法来取得 `AccountManager` 对象，代码如下：

```
//取得 AccountManager 对象
AccountManager_am = AccountManager.get(this);
```

在 `AccountManager` 中提供了很多方法来供我们管理这些账户，常用方法如表 9-4 所示。

表 9-4 AccountManager 中的常用方法

方 法	描 述
addAccount	添加一个账户
addOnAccountsUpdatedListener	添加一个账户更新的监听器
removeOnAccountsUpdatedListener	取消一个账户更新的监听器
clearPassword	清除指定账户的密码数据
getAccounts	得到所有的账户信息
getAccountsByType	得到指定类型的账户信息
getPassword	得到指定账户的密码
setUserData	设置指定账户的信息
setPassword	设置指定账户的密码
removeAccount	删除一个账户

通过这些方法就可以很轻松地操作这些账户数据，比如，将指定类型的账户信息全部列出来，代码如下：

```
/* 显示出所有账户 */
private void listAccounts()
{
    /* 得到指定类型的账户 */
    Account[] accounts = _am.getAccountsByType(getString(R.string.ACCOUNT_TYPE));
    _accountList.setText("账户列表: ");
    for (Account account : accounts)
    {
        _accountList.setText(_accountList.getText().toString() + '\n' +
            account.name + " - " + account.type);
    }
}
```

下面我们重点来学习如何将账户信息添加到账户管理器中。首先，需要实现一个 `AccountAuthenticatorActivity` 类来供用户输入账户信息，即 `AbstractAccountAuthenticator` 的一个 `Activity`，如代码清单 9-15 所示。

代码清单 9-15 第 9 章\Examples_09_08\src\com\yarin\android\Examples_09_08\auth\SleepyAccountAuthenticatorActivity.java

```
public class SleepyAccountAuthenticatorActivity extends AccountAuthenticatorActivity
{
    protected void onCreate(Bundle icle)
    {
        super.onCreate(icle);
        setContentView(R.layout.new_account);
        final Button done = (Button) findViewById(R.id.new_account_done);
        final EditText server = (EditText) findViewById(R.id.new_account_server);
        final EditText username = (EditText) findViewById(R.id.new_account_username);
        final EditText password = (EditText) findViewById(R.id.new_account_password);
        final Activity self = this;
        done.setOnClickListener(new OnClickListener() {
            public void onClick(View v)
            {
                //Account--指定账户名和账户类型
                Account account=new Account(username.getText().toString(), getString(R.string.ACCOUNT_TYPE));
                //服务器数据
                Bundle userdata = new Bundle();
                userdata.putString("SERVER", server.getText().toString());
                //取得 AccountManager
                AccountManager am = AccountManager.get(self);
                //添加一个账户
                if (am.addAccountExplicitly(account, password.getText().toString(), userdata))
                {
                    Bundle result = new Bundle();
                    result.putString(AccountManager.KEY_ACCOUNT_NAME, username.getText().toString());
                    result.putString(AccountManager.KEY_ACCOUNT_
```

```

        TYPE, getString(R.string.ACCOUNT_TYPE));
        setAccountAuthenticatorResult(result);
    }
    finish();
}
});
}
}

```

在上述代码清单中,我们先通过账户名及其类型构建一个 `Account` 对象,然后将服务器数据通过 `Bundle` 方式加入进来,最后通过 `AccountManager` 的 `addAccountExplicitly` 方法向账户管理器中添加一个账户信息。

接下来需要添加一个账户服务 (`Service`) 和一个验证器 (`AbstractAccountAuthenticator`)。

首先,构建一个 `authenticator.xml`,如代码清单 9-16 所示。

代码清单 9-16 第 9 章\Examples_09_08\res\xml\ authenticator.xml

```

<?xml version="1.0" encoding="utf-8"?>
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="com.yarin.AccountType"
    android:icon="@drawable/icon"
    android:smallIcon="@drawable/icon"
    android:label="@string/ACCOUNT_LABEL"
    android:accountPreferences="@xml/account_preferences"
/>

```

然后,在 `AndroidManifest.xml` 文件中开启一个账户管理服务,加入如下代码:

```

<service android:name="SleepyAccountsService">
<intent-filter>
<action android:name="android.accounts.AccountAuthenticator" ></action>
</intent-filter>
<meta-data
    android:name="android.accounts.AccountAuthenticator"
    android:resource="@xml/authenticator">
</meta-data>
</service>

```

账户服务类的实现很简单,就是在 `intent.getAction()` 的动作为 `android.accounts.AccountManager.ACTION_AUTHENTICATOR_INTENT` 时,通过 `AccountAuthenticator` 的 `getIBinder` 方法返回一个 `IBinder`,如代码清单 9-17 所示。

代码清单 9-17 第 9 章\Examples_09_08\src\com\yarin\android\Examples_09_08\ SleepyAccountsService.java

```

public class SleepyAccountsService extends Service
{
    private SleepyAccountAuthenticator _saa;
    public IBinder onBind(Intent intent)
    {
        IBinder ret = null;
        if (intent.getAction().equals(android.accounts.AccountManager.
            ACTION_AUTHENTICATOR_INTENT))
            ret = getSleepyAuthenticator().getIBinder();
        return ret;
    }
    private SleepyAccountAuthenticator getSleepyAuthenticator()
    {

```

```

        if (_saa == null)
            _saa = new SleepyAccountAuthenticator(this);
        return _saa;
    }
}

```

最后,最重要的是 `AbstractAccountAuthenticator` 类的实现,因为在添加、操作账户信息时会通过 `AbstractAccountAuthenticator` 实现异步调用。下面是实现的 `addAccount` 方法,如代码清单 9-18 所示。

代码清单 9-18 `addAccount` 方法

```

/* 添加账户 */
public Bundle addAccount(AccountAuthenticatorResponse response, String
accountType, String authTokenType, String[] requiredFeatures, Bundle
options)throws NetworkErrorException
{
    Log.d(_tag, accountType + " - " + authTokenType);
    Bundle ret = new Bundle();
    Intent intent=new Intent(_context,SleepyAccountAuthenticatorActivity.class);
    intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE, response);
    ret.putParcelable(AccountManager.KEY_INTENT, intent);
    return ret;
}

```

有关账户管理的内容,可能比较难以理解,建议大家一定要结合本节的示例程序进行学习,同时可以参考 `Android 2.0 SDK` 文档。

9.6 小结

本章内容之间的联系不是很紧密,都是一些 `Android` 中的特色功能,也正是这些功能吸引了不少开发者和用户。这些特色功能主要包括:`Android` 中传感器的使用、语音识别技术、`Google Map API` 在 `Android` 中的使用和出色的桌面组件开发,最后学习了 `Android` 中账户管理功能的简单实现。这些功能在日常生活中运用得也比较广泛,比如 `GPS` 导航、路径规划等,希望大家着重理解本章的内容,开发出具有创意的应用。





第三部分

实例篇

新
学
社
PDG

第 10 章

Android 应用开发实例

黑莓和 iPhone 都提供了受欢迎的、高容量的移动平台，但是却分别针对两个不同的消费群体。黑莓是企业用户的不二选择。但是，作为一种消费设备，它在易用性和“新奇特”方面难以和 iPhone 抗衡。Android 则是一个年轻的、有待开发的平台，它具有潜力同时涵盖移动电话的两个不同消费群体，甚至可能缩小工作和娱乐之间的差别。前面我们已经学习了 Android 平台应用开发的基础知识，本章将通过一些实例来对所学知识进行综合运用，包括：应用开发中重要的 UI 设计和 Android 应用程序组件（活动、服务、数据库、广播）。通过本章的学习，我们将对 Android SDK 进行移动应用开发有全面深入的了解，掌握 Android 编程的基本模式，理解 Android 编程的关键技术，能够完成功能较全面的 Android 程序。

10.1 情境模式

情景模式是我们日常生活中使用最多的软件，用户可以方便地对自己的情景模式进行修改，一般分为标准、家庭、会议、户外、寻呼机等。用户可以在手机的情景模式之间切换，每个情景模式中都包含不同的振铃音量、振铃音类型。可以针对不同环境设置不同的情景模式。例如，可能需要创建一个静音模式、会议模式（适合正式场合的振铃音）和户外模式（适合于嘈杂的室外环境）。在每个情景模式中都可以为语音呼入和信息告警等设置各种铃声。本节我们将实现一个 Android 平台情景模式的应用，主要包括：普通情景模式、定时情景模式、自定义情景模式。具体实现参见本书所附代码：第 10 章\RingProfile。

1. UI 设计

由于这个情景模式包括了 3 个大的选项，所以可以考虑使用 TabHost 来实现主界面的布局，然后在 TabHost 中使用不同的布局，完成实际功能。在普通模式下，可以设置一组单选项来切换不同的情景模式，如图 10-1 所示；定时情景模式在普通模式下还需要加上一个 TimePicker 来显示并设置时间，如图 10-2 所示；而自定义情景模式则需要自己定义声音的大小，如图 10-3 所示。具体实现参见本书所附代码：第 10 章\RingProfile\res\layout\main.xml。



图 10-1 普通情景模式界面



图 10-2 定时情景模式界面

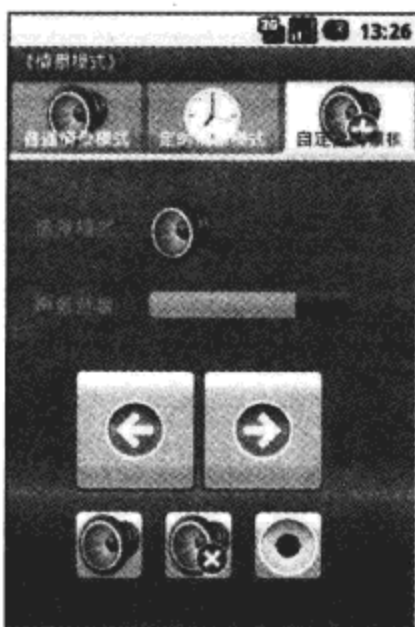


图 10-3 自定义情景模式界面

2. 音频管理器 (AudioManager)

既然是情景模式，肯定需要调节声音的大小、静音、振动等。这就需要使用 AudioManager，它主要用来控制和访问铃声。可以使用如下代码获得一个 AudioManager 实例：

```
Context.getSystemService(Context.AUDIO_SERVICE)
```

AudioManager 中提供了很多常量来供我们操作铃声的属性，这里列举一些常用的信息，如表 10-1 所示。

表 10-1 AudioManager 的常用信息

信 息	描 述
ADJUST_LOWER	减小铃声音量
ADJUST_RAISE	增加铃声音量
ADJUST_SAME	保持原来的铃声音量
EXTRA_RINGER_MODE	新的铃声模式
EXTRA_VIBRATE_SETTING	新的特殊的振动设置
EXTRA_VIBRATE_TYPE	振动类型
FLAG_PLAY_SOUND	当调节音量时，是否播放（预览效果）
FLAG_VIBRATE	是否振动
MODE_CURRENT	当前音频模式
MODE_INVALID	无效音频模式
MODE_IN_CALL	在调用的音频模式
MODE_NORMAL	普通音频模式
MODE_RINGTONE	响铃模式

有了表 10-1 的常量，还需要一些方法来设置这些模式，AudioManager 提供了表 10-2 中的一些方法。

表 10-2 AudioManager 方法

方 法	描 述
adjustVolume	调整音量大小
getMode	得到当前音频模式
getRingerMode	得到当前铃声模式
getStreamMaxVolume	得到最大的音频指数
getStreamVolume	得到当前音频指数
isSpeakerphoneOn	扬声器是否打开
loadSoundEffects	加载声音效果
playSoundEffect	播放声音效果
setMicrophoneMute	设置麦克风静音和关闭
setMode	设置音频模式
setRingerMode	设置铃声模式
setSpeakerphoneOn	设置扬声器打开和关闭
setVibrateSetting	当铃声模式改变时，设置振动类型

如下代码设置了既铃声又振动的模式：

```
audio.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER,
    AudioManager.VIBRATE_SETTING_ON);
audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_NOTIFICATION,
    AudioManager.VIBRATE_SETTING_ON);
```

如下代码设置了静音模式：

```
audio.setRingerMode(AudioManager.RINGER_MODE_SILENT);
audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER, AudioManager.VIBRATE_
    SETTING_OFF);
audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_NOTIFICATION, AudioManager.
    VIBRATE_SETTING_OFF);
```

3. AlarmManager 与 PendingIntent

由于使用了定时情景模式，肯定需要使用 AlarmManager 来控制定时器，我们可以通过 set(int type, long triggerAtTime, PendingIntent operation)来注册一个定时器。在注册定时、发送定时事件时，需要一个重要的参数 PendingIntent，它将负责传递定时信息，既要信息发出去，也要对发送后的状态进行处理，我们可以通过调用 getActivity(Context, int, Intent, int), getBroadcast(Context, int, Intent, int), getService(Context, int, Intent, int)函数来得到一个 PendingIntent 实例。通过 getBroadcast 获得的 PendingIntent 将实现广播的功能，就像调用 Context.sendBroadcast()函数一样。当系统通过它发送一个 intent 时要采用广播的形式，并且在该 intent 中会包含相应的 intent 接收对象，当然可以在创建 PendingIntent 时指定这个对象，也可以通过 ACTION 和 CATEGORY 等描述让系统自动找到该行为处理对象。比如，下面就是我们处理一个定时器的方法：

```
protected void ring(){
    Intent intent = new Intent(RingBroadcastReceiver.RING_CHANGED);
    intent.putExtra("checkedId", R.id.ring02);
```

```

PendingIntent alarmIntent = PendingIntent.getBroadcast(this,
RingBroadcastReceiver.REQUEST_CODE, intent, 0);
mAlarmManager.set(AlarmManager.RTC_WAKEUP, getTime(), alarmIntent);
}

```

4. BroadcastReceiver (广播接收器)

当应用程序所设置的时间到了后, 系统中的 `AlarmManagerService` 就会从系统底层获取一个闹铃事件, 并从自己维护的队列中取出与其匹配的信息, 然后通过其应用注册的 `PendingIntent` 把该闹铃事件发回给应用。当应用收到该 `Intent` 后就会对事件做出相应的处理, 比如, 更改情景模式为振动、铃声等。因此, 需要在应用程序中创建一个接收广播的类 `RingBroadcastReceiver` 来处理这些事件, 见代码清单 10-1 所示。

代码清单 10-1 第 10 章\RingProfile\src\com\yarin\android\RingProfile\RingBroadcastReceiver.java

```

public class RingBroadcastReceiver extends BroadcastReceiver
{
    private static final String TAG = "RingBroadcastReceiver";
    public static final String VIBRATE_CHANGED =
        "com.yarin.android.RingProfile.VIBRATE_CHANGED";
    public static final String SILENT_CHANGED =
        "com.yarin.android.RingProfile.SILENT_CHANGED";
    public static final String RV_CHANGED =
        "com.yarin.android.RingProfile.RV_CHANGED";
    public static final String RING_CHANGED =
        "com.yarin.android.RingProfile.RING_CHANGED";
    public static final int REQUEST_CODE = 0;
    public void onReceive(Context context, Intent intent)
    {
        AudioManager audio=(AudioManager)context.getSystemService(Context.
        AUDIO_SERVICE);
        int checkedId = intent.getIntExtra("checkedId", 0);
        Log.e(TAG, checkedId + intent.getAction());
        // 切换情景模式
        switch (checkedId)
        {
            case R.id.ring_and_vibrate01:
            case R.id.ring_and_vibrate02:
                ringAndVibrate(audio);
                break;
            case R.id.vibrate01:
            case R.id.vibrate02:
                vibrate(audio);
                break;
            case R.id.silent01:
            case R.id.silent02:
                silent(audio);
                break;
            default:
                ring(audio);
                break;
        }
    }
    // 铃声和振动
    protected void ringAndVibrate(AudioManager audio)
    {
        audio.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER,

```

```

        AudioManager.VIBRATE_SETTING_ON);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_NOTIFICATION,
        AudioManager.VIBRATE_SETTING_ON);
    }
    // 铃声
    protected void ring(AudioManager audio)
    {
        audio.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER,
        AudioManager.VIBRATE_SETTING_OFF);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_NOTIFICATION,
        AudioManager.VIBRATE_SETTING_OFF);
    }
    // 振动
    protected void vibrate(AudioManager audio)
    {
        audio.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER,
        AudioManager.VIBRATE_SETTING_ON);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_NOTIFICATION,
        AudioManager.VIBRATE_SETTING_ON);
    }
    // 静音
    protected void silent(AudioManager audio)
    {
        audio.setRingerMode(AudioManager.RINGER_MODE_SILENT);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_RINGER,
        AudioManager.VIBRATE_SETTING_OFF);
        audio.setVibrateSetting(AudioManager.VIBRATE_TYPE_NOTIFICATION,
        AudioManager.VIBRATE_SETTING_OFF);
    }
}

```

5. UI 事件处理

接下来只需要处理 UI 中定义的按钮、选项的事件即可。如代码清单 10-2 所示的代码片段就是我们在 RingProfile.java 中处理一个定时情景模式的事件。

代码清单 10-2 第 10 章\RingProfile\src\com\yarin\android\RingProfile\RingProfile.java 片段

```

//定时情景模式处理
RadioGroup group2 = (RadioGroup) findViewById(R.id.RadioGroup02);
group2.setOnCheckedChangeListener(new OnCheckedChangeListener()
{
    public void onCheckedChanged(RadioGroup group, int checkedId)
    {
        if (isChange)
            return;
        switch (checkedId)
        {
            case R.id.ring_and_vibrate02: ringAndVibrate(); break;
            case R.id.ring02: ring(); break;
            case R.id.vibrate02: vibrate(); break;
            case R.id.silent02: silent(); break;
        }
        RadioButton radio = (RadioButton) findViewById(checkedId);
        if (radio != null)
            radio.setTextSize(30);
    }
});

```


6. 权限处理

最后还需要对应用程序中使用的 `BroadcastReceiver` 以及所使用的一些 `Action` 进行注册和声明, 代码如下:

```
<receiver android:name="RingBroadcastReceiver">
    <intent-filter>
        <action android:name="com.yarin.android.RingProfile.RV_CHANGED" />
        <action android:name="com.yarin.android.RingProfile.RING_CHANGED" />
        <action android:name="com.yarin.android.RingProfile.VIBRATE_CHANGED" />
        <action android:name="com.yarin.android.RingProfile.SILENT_CHANGED" />
    </intent-filter>
</receiver>
```

到这里情景模式应用已基本完成, 该实例主要运用了 UI 设计、`AudioManager` (音频管理器)、`AlarmManager` (定时器)、`PendingIntent` 和 `BroadcastReceiver` (广播信息发送和接收器)。当然, 大家还可以根据自己的创意和需求做出更好的情景模式, 比如加入 Google 的地图定位, 当我们到达某个地点时就自动切换情景模式, 这样就可以避免在开会时出现铃声的尴尬局面了。

10.2 文件管理器

文件管理器就是能够浏览和管理我们手机中的文件、文件夹和存储卡的工具, 和 Windows 系统提供的资源管理工具很相似, 我们可以用它查看本机的所有资源、文件系统结构, 以便更清楚、直观地查看和操作手机的文件和文件夹。当然, 可以包括重命名、删除、新建、复制、粘贴等简单的文件操作。大家都知道第一款 Android 手机 G1 中并没有自带这样的文件管理器, 这使得用户操作文件极不方便。因此, 通过本节内容的学习, 我们将自己来完成一个 Android 平台的文件管理器。

下面先来看看这个文件管理器需要实现什么样的功能, 采用什么样的界面设计。首先需要浏览文件 (文件夹)、显示路径以及对目录的一些常用操作 (比如新建、删除、粘贴等), 如图 10-4 所示; 当选中一个文件时, 需要提示用户有关文件操作的菜单, 比如打开、复制、剪切、重命名、删除等操作, 如图 10-5 所示; 在重命名或者新建文件夹时还需要弹出对话框来供用户输入文件名, 如图 10-6 所示; 在操作文件, 比如重命名、粘贴等, 遇到该目录中有相同的文件名时, 提示用户是否需要覆盖, 如图 10-7 所示; 很重要的一点是, 当用户删除文件时, 提示用户是否确定删除该文件等, 如图 10-8 所示; 当然, 在删除时需要判断删除的是文件还是整个目录, 然后进行相应的操作。具体实现参见本书所附代码: 第 10 章\FileManager。

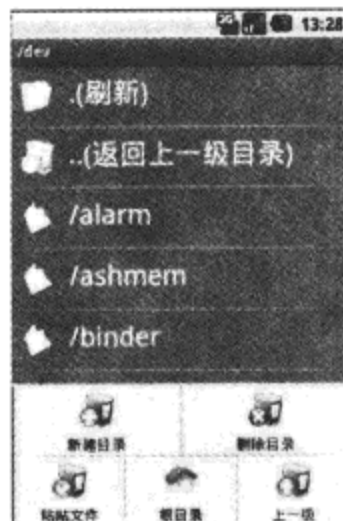


图 10-4 文件浏览

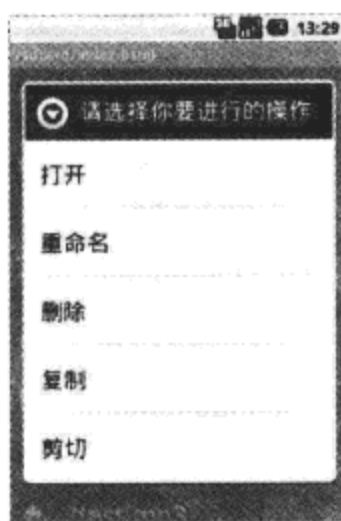


图 10-5 菜单操作



图 10-6 重命名

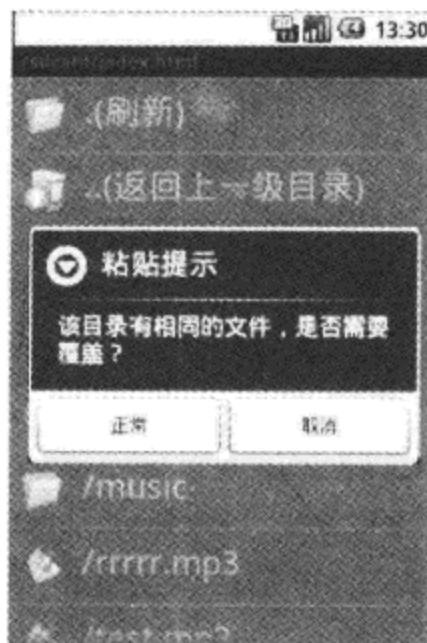


图 10-7 覆盖提示



图 10-8 删除提示

1. UI 设计

由于需要浏览大量的文件（文件夹），所以需要使用一个 `ListView` 来显示这些文件（文件夹）的列表，但是大家从上面的图中可以看到，我们是根据不同的文件类型在前面设置了不同的图标，因此 `Android` 自带的 `ListView` 可能并不能满足需要，所以这里我们需要自己来定义一个用于显示文件（文件夹）的列表视图。

前面使用了 `xml` 布局文件来设计界面，这里我们学习一下通过程序来定义布局文件。那么在布局之前我们需要设计一个实例来显示每一个列表，这个列表必须包括一个图标和一个字符串，因此定义一个 `IconifiedText` 类专门用来显示每一行数据的信息。该类中至少包含以上两个对象，这里我们使用 `Drawable` 来显示图标，使用 `String` 来显示文件名。另外附加一个 `boolean` 来表示该行是否被选中。为了能够操作每一类数据，我们要完成一些该类的成员方法，比如设置文件名、设置图标、获得文件名、获得图标等，见代码清单 10-3 所示。

代码清单 10-3 第 10 章\FileManager\src\com\yarin\android\FileManager\IconifiedText.java

```
public class IconifiedText implements Comparable<IconifiedText>
{
    /* 文件名 */
    private String mText = "";
    /* 文件的图标 ICNO */
    private Drawable mIcon = null;
    /* 能否选中 */
    private boolean mSelectable = true;
    public IconifiedText(String text, Drawable bullet)
    {
        mIcon = bullet;
        mText = text;
    }
    //是否可以选中
    public boolean isSelectable()
    {
        return mSelectable;
    }
    //设置是否可以选中
```

```

public void setSelectable(boolean selectable)
{
    mSelectable = selectable;
}
//得到文件名
public String getText()
{
    return mText;
}
//设置文件名
public void setText(String text)
{
    mText = text;
}
//设置图标
public void setIcon(Drawable icon)
{
    mIcon = icon;
}
//得到图标
public Drawable getIcon()
{
    return mIcon;
}
//比较文件名是否相同
public int compareTo(IconifiedText other)
{
    if (this.mText != null)
        return this.mText.compareTo(other.getText());
    else
        throw new IllegalArgumentException();
}
}

```

有了每一行的数据信息，要将该数据显示在界面之上，就需要实现一个界面布局的类 `IconifiedTextView`，在该类中可以使用 `TextView` 来显示文件名，使用 `ImageView` 来显示图标，由于我们的布局是图标和文件名在同一行上面，可以使用线性布局来完成这个界面，所以这里的 `IconifiedTextView` 就继承一个线性布局 `LinearLayout`。具体实现见代码清单 10-4 所示。

代码清单 10-4 第 10 章\FileManager\src\com\yarin\android\FileManager\IconifiedTextView.java

```

public class IconifiedTextView extends LinearLayout
{
    //一个文件包括文件名和图标
    //采用一个垂直线性布局
    private TextView mText = null;
    private ImageView mIcon = null;
    public IconifiedTextView(Context context, IconifiedText aIconifiedText)
    {
        super(context);
        //设置布局方式
        this.setOrientation(HORIZONTAL);
        mIcon = new ImageView(context);
        //设置 ImageView 为文件的图标
        mIcon.setImageDrawable(aIconifiedText.getIcon());
        //设置图标在该布局中的填充位置
        mIcon.setPadding(8, 12, 6, 12);
    }
}

```

```

//将 ImageView (即图标) 添加到该布局中
addView(mIcon, new LinearLayout.LayoutParams(
    LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
//设置文件名、填充方式、字体大小
mText = new TextView(context);
mText.setText(aIconifiedText.getText());
mText.setPadding(8, 6, 6, 10);
mText.setTextSize(26);
//将文件名添加到布局中
addView(mText, new LinearLayout.LayoutParams(
    LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
}
//设置文件名
public void setText(String words)
{
    mText.setText(words);
}
//设置图标
public void setIcon(Drawable bullet)
{
    mIcon.setImageDrawable(bullet);
}
}

```

2. BaseAdapter (数据容器)

既然是列表视图，那么肯定需要使用 `BaseAdapter` 来存储这些列表的数据。`BaseAdapter` 可以用来为一个列表提供数据源，所以这里我们创建一个继承自 `BaseAdapter` 类的 `IconifiedTextListAdapter` 类，专门用来显示列表中的数据。只不过这里 `List` 的内容是我们上面实现的 `IconifiedText` 类。具体实现如代码清单 10-5 所示。

代码清单 10-5 第 10 章\FileManager\src\com\yarin\android\FileManager\IconifiedTextListAdapter.java

```

//使用 BaseAdapter 来存储取得的文件
public class IconifiedTextListAdapter extends BaseAdapter
{
    private Context mContext = null;
    // 用于显示文件的列表
    private List<IconifiedText> mItems = new ArrayList<IconifiedText>();
    public IconifiedTextListAdapter(Context context)
    {
        mContext = context;
    }
    //添加一项 (一个文件)
    public void addItem(IconifiedText it) { mItems.add(it); }
    //设置文件列表
    public void setListItems(List<IconifiedText> lit) { mItems = lit; }
    //得到文件的数目, 列表的个数
    public int getCount() { return mItems.size(); }
    //得到一个文件
    public Object getItem(int position) { return mItems.get(position); }
    //能否全部选中
    public boolean areAllItemsSelectable() { return false; }
    //判断指定文件是否被选中
    public boolean isSelectable(int position)
    {
        return mItems.get(position).isSelectable();
    }
}

```

```

    }
    //得到一个文件的 ID
    public long getItemId(int position) { return position; }
    //重写 getView 方法来返回一个 IconifiedTextView (我们自定义的文件布局) 对象
    public View getView(int position, View convertView, ViewGroup parent) {
        IconifiedTextView btv;
        if (convertView == null)
        {
            btv=new IconifiedTextView(mContext, mItems.get(position));
        }
        else
        {
            btv = (IconifiedTextView) convertView;
            btv.setText(mItems.get(position).getText());
            btv.setIcon(mItems.get(position).getIcon());
        }
        return btv;
    }
}

```

3. 事件处理

所有的准备工作到这里都完成了，下面就需要获取手机中的目录以及其中的文件（文件夹），然后显示到 ListView 中展现给用户。文件的操作和标准的 Java 一样。而这里我们需要学习的是以浮动的方式显示菜单和对话框。如代码清单 10-6 所示，这是一个重命名的浮动对话框。

代码清单 10-6 第 10 章\FileManager\src\com\yarin\android\FileManager\FileManager.java 片段

```

//自定义一个带输入的对话框，由 TextView 和 EditText 构成
final LayoutInflater factory = LayoutInflater.from(FileManager.this);
final View dialogview = factory.inflate(R.layout.rename, null);
//设置 TextView 的提示信息
((TextView) dialogview.findViewById(R.id.TextView01)).setText("重命名");
//设置 EditText 输入框初始值
((EditText) dialogview.findViewById(R.id.EditText01)).setText(file.getName());

Builder builder = new Builder(FileManager.this);
builder.setTitle("重命名");
builder.setView(dialogview);
builder.setPositiveButton(android.R.string.ok,
    new AlertDialog.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            //点击确定之后
            String value = GetCurDirectory()+"/"+((EditText) dialogview.findViewById(
                R.id.EditText01)).getText().toString();
            if(new File(value).exists())
            {
                Builder builder = new Builder(FileManager.this);
                builder.setTitle("重命名");
                builder.setMessage("文件名重复，是否需要覆盖？");
                builder.setPositiveButton(android.R.string.ok,
                    new AlertDialog.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {
                            String str2 = GetCurDirectory()+"/"+((EditText) dialogview.
                                findViewById(R.id.EditText01)).getText().toString();
                            file.renameTo(new File(str2));
                            browseTo(new File(GetCurDirectory()));
                        }
                    }
                );
            }
        }
    }
);

```

```

    });
    builder.setNegativeButton(android.R.string.cancel,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
    builder.setCancelable(false);
    builder.create();
    builder.show();
}
else
{
    //重命名
    file.renameTo(new File(value));
    browseTo(new File(GetCurDirectory()));
}
});
builder.setNegativeButton(android.R.string.cancel,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });
builder.setOnCancelListener(new DialogInterface.OnCancelListener()
{
    public void onCancel(DialogInterface dialog) {
        dialog.cancel();
    }
});
builder.show();

```

当我们点击打开菜单之后，要求能够根据不同的文件类型来启动相应的 Activity，控制文件的操作等。下面是 FileManager.java 文件中的一个打开文件的方法，先通过 File 来确定一个 URI，然后通过 Intent 来设置 URI 的属性，最后通过 startActivity 方法来开启一个 Activity，如代码清单 10-7 所示。

代码清单 10-7 openFile 方法

```

//打开指定文件
protected void openFile(File aFile)
{
    Intent intent = new Intent();
    intent.setAction(android.content.Intent.ACTION_VIEW);
    File file = new File(aFile.getAbsolutePath());
    // 取得文件名
    String fileName = file.getName();
    // 根据不同的文件类型来打开文件
    if (checkEndsWithInStringArray(fileName, getResources().getStringArray(
        R.array.fileEndingImage)))
    {
        intent.setDataAndType(Uri.fromFile(file), "image/*");
    }
    else if (checkEndsWithInStringArray(fileName, getResources().
        getStringArray(R.array.fileEndingAudio)))
    {
        intent.setDataAndType(Uri.fromFile(file), "audio/*");
    }
}

```

```

    }
    else if (checkEndsWithInStringArray(fileName, getResources().
        getStringArray(R.array.fileEndingVideo)))
    {
        intent.setDataAndType(Uri.fromFile(file), "video/*");
    }
    startActivity(intent);
}

```

4. AndroidManifest.xml (权限设置)

由于我们在应用程序中使用了文件操作，所以需要在 `AndroidManifest.xml` 中添加 “`<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>`” 声明访问权限，代码如下：

```

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

```

在补充完各个菜单对应的处理之后，该文件浏览器的例子就完成了。同样，可以加入一些更复杂的操作，和其他应用程序一起使用，比如，为音乐播放器指定一个歌曲的路径，这里可以使用该浏览器来浏览手机上的文件（文件夹），并做出简单的操作。

10.3 通讯录

通讯录作为手机的基本功能之一，每天我们都在频繁地使用着。根据手机功能使用调查显示，有八成以上的消费者使用手机通讯录功能。但各款手机在通讯录这一功能上，有着千差万别的变化。各个手机生产厂商在不断开发手机新功能的同时，也没有忘记加强和完善手机通讯录的功能。手机通讯录对于人们的意义，已不仅仅像记事簿一样显示通讯地址，而是向着个性化、人性化的方向发展。通讯录从无到有，从英文到中文，经过了十几年的发展历程，而今后的发展趋势就是从通讯录发展成为名片夹，也就是在一个人名下，可以存储座机、手机、宅电、单位、传真、地址、电子邮件等多项内容，并对不同的人分别设置个性化铃声和多种图片。这种名片夹是在电话簿的基础上，大大丰富了内容，同时结构也发生了革命性变化，而且随着手机的发展，相信更优秀的通讯录会越来越受到社会各层人士的宠爱。

本节我们将在 `Android` 上完成一个通讯录的功能。首先我们将确定要实现的这个通讯录有什么样的功能，比如：浏览联系人、添加联系人、删除联系人、编辑联系人、查看联系人，当找到一个联系人之后，可以呼叫或者发送短信息给该联系人。确定了这些功能之后，再来思考需要使用哪些知识，比如：要存储很多联系人，可以使用数据库，以方便管理和维护。在确定之后，就可以开始新建工程了。具体实现参见本书所附代码：第 10 章 `MyContacts`。

1. UI 设计

在设计界面时，需要根据所定的功能来设计，本节中我们浏览联系人是通过一个 `List` 来展示给用户的，如图 10-9 所示；用户需要操作，所以需要设计供用户操作的菜单（`Menu`），本节我们将学习如何动态地添加菜单选项，图 10-9 和图 10-10 分别是用户没有选中联系人的菜单和用户选中一个联系人的菜单；当然，为了完整，还可以设置用户点击一列信息的事件和用户长按一列信息的事件，图 10-11 即长按之后产生的菜单。



图 10-9 浏览联系人



图 10-10 选中联系人的操作菜单

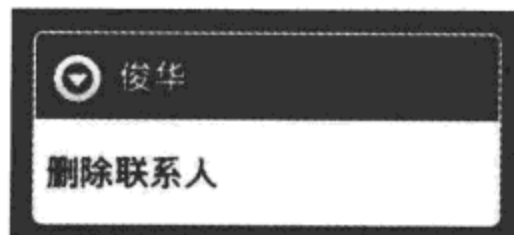


图 10-11 长按联系人的菜单

在完成了这些基本功能的设计之后，需要设计添加和修改联系人的界面。这样的布局很简单，可以通过 `TextView` 来显示一个标签，比如：姓名、电话等。既然要编辑肯定需要接收用户的输入，所以这里使用了 `EditText` 来供用户输入信息，如图 10-12 所示；在修改之后需要查看联系人的信息，这样的界面可以使用 `TableRow` 来显示，如图 10-13 所示。当然实现这些功能的方法很多，这里并不是唯一的方法。



图 10-12 编辑联系人



图 10-13 查看联系人

2. 数据库设计

对于联系人信息的存储，可以使用 Android 中提供的数据库。要设计数据库，首先要确定数据都是什么内容，为了方便管理、维护和共享，首先将数据库中要使用的数据全部定义到 `ContactColumn` 类，该例中定义的数据信息如代码清单 10-8 所示。

代码清单 10-8 第10章\MyContacts\src\com\yarin\android\MyContacts\ContactColumn.java

//定义数据

public class ContactColumn **implements** BaseColumns

{

public ContactColumn() {}

//列名

public static final String NAME = "name"; //姓名 **public static final** String MOBILENUM = "mobileNumber"; //移动电话 **public static final** String HOMENUM = "homeNumber"; //家庭电话 **public static final** String ADDRESS = "address"; //地址 **public static final** String EMAIL = "email"; //邮箱 **public static final** String BLOG = "blog"; //博客

//列索引值

public static final int _ID_COLUMN = 0; **public static final** int NAME_COLUMN = 1; **public static final** int MOBILENUM_COLUMN = 2; **public static final** int HOMENUM_COLUMN = 3; **public static final** int ADDRESS_COLUMN = 4; **public static final** int EMAIL_COLUMN = 5; **public static final** int BLOG_COLUMN = 6;

//查询结果

public static final String[] PROJECTION = {

_ID,

NAME,

MOBILENUM,

HOMENUM,

ADDRESS,

EMAIL,

BLOG,

};

}

Android 中的 `android.database.sqlite.SQLiteOpenHelper` 类是一个专门用于数据库创建和版本管理的辅助类。因此,为了更好地管理数据库,这里我们创建一个继承自 `SQLiteOpenHelper` 的辅助类 `DBHelper` 来维护和更新数据库,如代码清单 10-9 所示。

代码清单 10-9 第10章\MyContacts\src\com\yarin\android\MyContacts\DBHelper.java

public class DBHelper **extends** SQLiteOpenHelper

{

public static final String DATABASE_NAME = "mycontacts.db"; //数据库名 **public static final** int DATABASE_VERSION = 2; //版本 **public static final** String CONTACTS_TABLE = "contacts"; //表名

//创建表的 SQL 语句

private static final String DATABASE_CREATE =

"CREATE TABLE " + CONTACTS_TABLE + " ("

+ ContactColumn._ID+" integer primary key autoincrement,"

+ ContactColumn.NAME+" text,"

+ ContactColumn.MOBILENUM+" text,"

+ ContactColumn.HOMENUM+" text,"

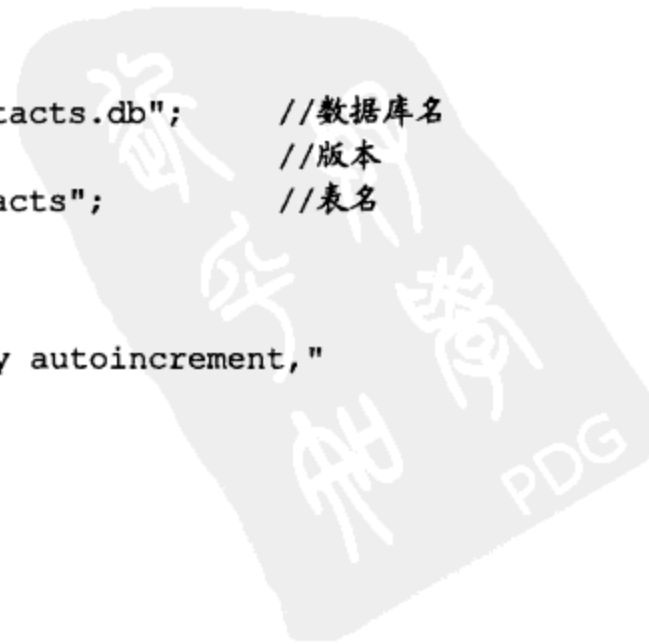
+ ContactColumn.ADDRESS+" text,"

+ ContactColumn.EMAIL+" text,"

+ ContactColumn.BLOG+" text);"

public DBHelper(Context context)

{

super(context, DATABASE_NAME, null, DATABASE_VERSION);

```

    }
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(DATABASE_CREATE);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        db.execSQL("DROP TABLE IF EXISTS " + CONTACTS_TABLE);
        onCreate(db);
    }
}

```

在 Android 中, `ContentProvider` 类提供了一种多应用间数据共享的方式, 比如: 联系人信息可以被多个应用程序访问。 `ContentProvider` 是一个实现了一组用于提供其他应用程序存取数据的标准方法的类, 所以可以创建一个继承自 `ContentProvider` 类的派生类来操作数据库, 比如: 查询、修改、添加、删除等操作。下面就来分别实现这些操作。

这里我们定义了一种新的类型, 因此得到 URI 时, 就可以通过如下方法来转换为我们需要的类型。其中, `CONTACTS` 表示多行数据, `CONTACT_ID` 表示单行数据, 当然这些类型都需要在 `AndroidManifest.xml` 中声明权限。

```

// URI 类型转换
public String getType(Uri uri) {
    switch (uriMatcher.match(uri))
    {
        case CONTACTS:
            return "vnd.android.cursor.dir/vnd.yarin.android.mycontacts";
        case CONTACT_ID:
            return "vnd.android.cursor.item/vnd.yarin.android.mycontacts";
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}

```

定义好了类型, 还需要在 `ContentProvider` 中实现对数据库的一些常用操作, 下面是删除一条指定的数据, 如代码清单 10-10 所示。

代码清单 10-10 delete 方法

```

public int delete(Uri uri, String where, String[] selectionArgs)
{
    int count;
    switch (uriMatcher.match(uri))
    {
        case CONTACTS:
            count = contactsDB.delete(CONTACTS_TABLE, where, selectionArgs);
            break;
        case CONTACT_ID:
            String contactID = uri.getPathSegments().get(1);
            count = contactsDB.delete(CONTACTS_TABLE,
                ContactColumn._ID
                + "=" + contactID
                + (!TextUtils.isEmpty(where) ? " AND (" + where + ")" : ""),
                selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}

```

```

        .getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

```

下面是插入一条数据，如代码清单 10-11 所示。

代码清单 10-11 insert 方法

```

public Uri insert(Uri uri, ContentValues initialValues){
    if (uriMatcher.match(uri) != CONTACTS){
        throw new IllegalArgumentException("Unknown URI " + uri);
    }
    ContentValues values;
    if (initialValues != null){
        values = new ContentValues(initialValues);
    }else {
        values = new ContentValues();
    }
    // 设置默认值
    ...
    long rowId = contactsDB.insert(CONTACTS_TABLE, null, values);
    if (rowId > 0) {
        Uri noteUri = ContentUris.withAppendedId(CONTENT_URI, rowId);
        getContext().getContentResolver().notifyChange(noteUri, null);
        Log.e(TAG + "insert", noteUri.toString());
        return noteUri;
    }
    throw new SQLException("Failed to insert row into " + uri);
}

```

下面是查询数据，如代码清单 10-12 所示。

代码清单 10-12 query 方法

```

public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder){
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(CONTACTS_TABLE);

    switch (uriMatcher.match(uri))
    {
        case CONTACT_ID:
            qb.appendWhere(ContactColumn._ID + "=" + uri.getPathSegments().get(1));
            break;
    }
    String orderBy;
    if (TextUtils.isEmpty(sortOrder)) {
        orderBy = ContactColumn._ID;
    }else{
        orderBy = sortOrder;
    }
    Cursor c=qb.query(contactsDB,projection,selection,selectionArgs,null,null,orderBy);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}

```

下面是更新数据库，如代码清单 10-13 所示。

代码清单 10-13 update 方法

```

public int update(Uri uri,ContentValues values,String where,String[] selectionArgs){
    int count;
    switch (uriMatcher.match(uri))

```

```

{
    case CONTACTS:
        count = contactsDB.update(CONTACTS_TABLE, values, where, selectionArgs);
        break;
    case CONTACT_ID:
        String contactID = uri.getPathSegments().get(1);
        count = contactsDB.update(CONTACTS_TABLE,
            values,
            ContactColumn._ID + "=" +
            contactID
            + (!TextUtils.isEmpty(where) ?
            " AND (" + where + ")" : ""),
            selectionArgs);

        break;
    default:
        throw new IllegalArgumentException("Unsupported URI: " + uri);
}
getContext().getContentResolver().notifyChange(uri, null);
return count;
}

```

3. 动态菜单

在浏览联系人时，如果没有选中一条联系人信息，那么就不能使用删除、修改、查看等菜单，但是当选中一条数据时，就可以执行这些功能，所以我们要使用动态菜单，根据不同的状态显示不同的菜单。动态菜单要在 Android 中使用 Intent 来设置 ACTION，然后根据不同的动作来启动不同的界面 Activity。下面是添加动态菜单“编辑”、“查看”的方法，如代码清单 10-14 所示。

代码清单 10-14 onPrepareOptionsMenu 方法

```

public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);
    final boolean haveItems = getListAdapter().getCount() > 0;
    if (haveItems) {
        Uri uri = ContentUris.withAppendedId(getIntent().getData(), getSelectedItemId());
        Intent[] specifics = new Intent[2];
        specifics[0] = new Intent(Intent.ACTION_EDIT, uri);
        specifics[1] = new Intent(Intent.ACTION_VIEW, uri);
        MenuItem[] items = new MenuItem[2];
        //添加满足条件的菜单
        Intent intent = new Intent(null, uri);
        intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
        menu.addIntentOptions(Menu.CATEGORY_ALTERNATIVE, 0, 0, null, specifics, intent, 0, items);
        if (items[0] != null) {
            //编辑联系人
            items[0].setShortcut('1', 'e').setIcon(R.drawable.edituser).setTitle(
                R.string.editor_user);
        }
        if (items[1] != null) {
            //查看联系人
            items[1].setShortcut('2', 'f').setTitle(R.string.view_user).setIcon(
                R.drawable.viewuser);
        }
    } else {
        menu.removeGroup(Menu.CATEGORY_ALTERNATIVE);
    }
    return true;
}

```

当我们长按列表项时会触发 `onCreateContextMenu` 事件，这时可以设置能够进行操作的菜单，通过 `onContextItemSelected` 方法来监听长按菜单的事件处理。本例中长按列表项时会弹出删除该条记录的菜单，如图 10-11 所示。

4. 权限设置

本例中创建了新的类型，所以需要在 `AndroidManifest.xml` 中定义、声明。另外还设置了直接呼叫联系人和发送短信，这都需要在 `AndroidManifest.xml` 中注册，如代码清单 10-15 所示。

代码清单 10-15 `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yarin.android.MyContacts"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <provider android:name="ContactsProvider"
            android:authorities="com.yarin.android.provider.ContactsProvider"/>
        <activity android:name=".MyContacts"
            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=".ContactEditor"
            android:label="@string/editor_user">
            <intent-filter>
                <action android:name="android.intent.action.EDIT" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.item/vnd.yarin.android.mycontacts" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.INSERT" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.dir/vnd.yarin.android.mycontacts" />
            </intent-filter>
        </activity>
        <activity android:name=".ContactView"
            android:label="@string/view_user">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.item/vnd.yarin.android.mycontacts" />
            </intent-filter>
            <intent-filter>
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.dir/vnd.yarin.android.mycontacts" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
    <uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
```

```
<uses-sdk android:minSdkVersion="5" />
</manifest>
```

最后完成查看和修改界面的功能、一些菜单的事件处理,就可以制作一个自己的通讯录了。大家可以在学习之后将该通讯录更加完善和优化,使其成为一个优秀的通讯录软件。通过该实例,我们主要学习了 UI 设计、数据库的综合操作、动态菜单的使用以及各种权限的注册。通过本章的学习,大家应该对 Android 平台的数据库操作有了更进一步的理解,同时也对 Android 系统有了更深入的了解。

10.4 音乐播放器

音频数字化目前是较为成熟的技术,多媒体声卡就是采用此技术而设计的,数字音响也是采用此技术取代传统的模拟方式而达到了理想的音响效果。音频采样包括两个重要的参数,即采样频率和采样数据位数。采样频率就是对声音每秒钟采样的次数,人耳听觉上限在 20kHz 左右,目前常用的采样频率为 11kHz、22kHz 和 44kHz 三种。采样频率越高,音质越好,存储数据量越大。CD 唱片的采样频率为 44.1kHz,目前达到了最好的听觉效果。采样数据位数就是每个采样点的数据表示范围,目前常用的有 8 位、12 位和 16 位三种。不同的采样数据位数决定了不同的音质,采样位数越高,存储数据量越大,音质也越好。CD 唱片采用了双声道 16 位采样,采样频率为 44.1kHz,因而达到了专业级水平。通过前面的学习,我们知道 Android 平台提供了完整的多媒体解决方案,为开发者提供了统一、简单、易用的开发接口,让我们可以很轻松地开发出一个多媒体播放器。本节我们将通过构建一个音乐播放器来更进一步地学习 Android 平台的多媒体开发技术,具体实现参见本书所附代码:第 10 章\MediaPlayer。

1. UI 设计

现在很多音乐播放器的界面都做得非常漂亮,本节我们就使用一个简单的 ListView 来显示播放列表,如图 10-14 所示;当点击一首曲目时,出现几个按钮来控制音乐的暂停、停止、播放等操作,如图 10-15 所示。具体布局文件参见本书所附代码:第 10 章\MediaPlayer\res\layout\list_layout.xml,当然大家可以根据自己的创意来完成最优秀的 UI 设计。



图 10-14 播放列表



图 10-15 播放操作

2. 媒体信息的管理

为了使用户更方便地管理曲目,我们将在启动播放器的同时自动获取手机和 SD 卡上的音频文件,并显示到 ListView 视图中。Android 系统提供了 MediaScanner、MediaProvider、MediaStore 等接口,并且提供了一套数据库表,通过 Content Provider 方式提供给用户。当手机开机或者有 SD 卡插拔等事件发生时,系统将会自动扫描 SD 卡和手机内存上的媒体文件,如音频、视频、图片等,将相

应的信息放到定义好的数据库表中。**MediaStore** 中定义了一系列的数据表，通过 **ContentResolver** 提供的查询接口，可以得到各种需要的信息。下面我们来学习如何获得这些信息及其使用方法。

ContentResolver 提供了如下查询接口：

```
Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder);
```

参数说明如下：

- ❑ **uri**：指明要查询的数据库名称加上表的名称。
- ❑ **projection**：指定查询数据库表中的哪几列，返回的游标中将包括相应的信息。**null** 则返回所有信息。
- ❑ **selection**：指定查询条件。
- ❑ **selectionArgs**：参数 **selection** 里有 “?” 这个符号，这里可以以实际值代替这个问号。如果 **selection** 里没有 “?” 的话，那么这个 **String** 数组可以为 **null**。
- ❑ **sortOrder**：指定查询结果的排列顺序。

例如，要查询所有的歌曲：

```
Cursor cursor = query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, null,
null, MediaStore.Audio.Media.DEFAULT_SORT_ORDER);
```

该方法将返回所有外部存储卡上的音乐文件的信息，我们可以通过一些常量来获得歌曲的具体信息，常用信息如表 10-3 所示。

表 10-3 多媒体的常用信息

信 息	描 述
<code>MediaStore.Audio.Media._ID</code>	歌曲 ID
<code>MediaStore.Audio.Media.TITLE</code>	歌曲的名称
<code>MediaStore.Audio.Media.ALBUM</code>	歌曲的专辑名
<code>MediaStore.Audio.Media.ARTIST</code>	歌曲的歌手名
<code>MediaStore.Audio.Media.DATA</code>	歌曲文件的路径
<code>MediaStore.Audio.Media.DURATION</code>	歌曲的总播放时长
<code>MediaStore.Audio.Media.SIZE</code>	歌曲文件的大小
<code>MediaStore.Audio.Playlists.DATE_MODIFIED</code>	修改时间
<code>MediaStore.Audio.Playlists.DATE_ADDED</code>	添加时间
<code>MediaStore.Audio.Playlists.NAME</code>	播放列表名称
<code>MediaStore.Audio.Playlists._ID</code>	播放列表 ID
<code>MediaStore.Audio.Albums.NUMBER_OF_SONGS</code>	该专辑共有多少歌曲
<code>MediaStore.Audio.Albums.ALBUM</code>	专辑名称
<code>MediaStore.Audio.Albums._ID</code>	专辑 ID
<code>MediaStore.Audio.Artists.NUMBER_OF_TRACKS</code>	共有多少该歌手的歌曲
<code>MediaStore.Audio.Artists.NUMBER_OF_ALBUMS</code>	共有多少该歌手的专辑
<code>MediaStore.Audio.Artists.ARTIST</code>	歌手姓名
<code>MediaStore.Audio.Artists._ID</code>	歌手 ID

这些常用信息使我们可以通过相应的 `cursor.get` 方法来取得这些信息，代码如下：

```
int id =
cursor.getInt(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media._ID));
String tilte =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.TITLE));
String album =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.ALBUM));
String artist =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.ARTIST));
int size =
cursor.getLong(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.SIZE));
int duration =
cursor.getInt(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DURATION));
```

查询歌手信息，返回所有外部存储卡上的歌手信息，代码如下：

```
Cursor cursor = query(MediaStore.Audio.Artists.EXTERNAL_CONTENT_URI, null, null,
null, MediaStore.Audio.Artists.DEFAULT_SORT_ORDER);
```

查询专辑，返回所有外部存储卡上的专辑信息，代码如下：

```
Cursor cursor = query(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI, null,
null, null, MediaStore.Audio.Albums.DEFAULT_SORT_ORDER);
```

查询播放列表，返回所有外部存储卡上的专辑信息，代码如下：

```
Cursor cursor = query(MediaStore.Audio.Playlists.EXTERNAL_CONTENT_URI, null,
null, null, MediaStore.Audio.Playlists.DATE_ADDED + "yarin");
```

通过组合这些查询结果，指定查询条件，用户可以很方便地查询指定的媒体信息，比如：查询属于指定歌手（歌手 ID 为 `aid`）的歌曲：

```
query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, MediaStore.Audio.Media.
ARTIST_ID + "=" + aid, null, MediaStore.Audio.Media.TITLE);
```

查询属于指定专辑（专辑 ID 为 `aid`）的歌曲：

```
query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, MediaStore.Audio.Media.
ALBUM_ID + "=" + aid, null, MediaStore.Audio.Media.TITLE);
```

有了这些之后，就可以先定义一个用来管理音乐信息的类 `MusicInfoController`，它主要有如下两个方法，分别是查询指定的信息和查询所有歌曲信息。

```
private Cursor query(Uri uri, String[] prjs, String selections, String[]
selectArgs, String order){
    ContentResolver resolver = pApp.getContentResolver();
    if (resolver == null){
        return null;
    }
    return resolver.query(uri, prjs, selections, selectArgs, order);
}
public Cursor getAllSongs(){
    return query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, null, null,
MediaStore.Audio.Media.DEFAULT_SORT_ORDER);
}
```

3. 播放音乐

音乐文件的播放功能是由 `MediaPlayer` 类实现的，`MediaPlayer` 提供了常用的接口，比如播放、暂停、停止、快速定位等。我们需要考虑如何来播放这些媒体库中的文件。我们希望当用户退出这个程序界面后，程序仍然能够继续播放歌曲，比如用户在读邮件时可以听听音乐。为了达到后台播放的效果，需要使用 `Service`。当程序的所有 `Activity` 都退出后，`Service` 仍然可以在后台运行。在这

个示例中我们使用 Local Service，它与应用程序运行在同一个进程中。

因此，需要先创建一个 MusicPlayerService 类，它继承自 android.app.Service，重载 onBind 方法，返回自定义的 LocalBinder，通过 LocalBinder 的 getService() 方法就可以获得 MusicPlayerService 的实例了。代码如下：

```
private final IBinder mBinder = new LocalBinder();
public class LocalBinder extends Binder{
    public MusicPlayerService getService(){
        return MusicPlayerService.this;
    }
}
public IBinder onBind(Intent intent){
    return mBinder;
}
```

再在该类中创建一个 MediaPlayer 用来播放音乐，在 onCreate() 函数中对其进行初始化，并分别实现播放、暂停、停止等操作。

接下来需要实现 MusicList 来显示播放列表并开启服务 (Service)，在后台播放音乐。在我们没有选择要播放歌曲的情况下，播放、暂停等按钮处于不可见状态，当用户选择一首曲目时，要显示这些控制按钮，当这个状态改变时，通过 Intent 的形式，将消息广播 (BroadcastReceiver) 出去，给 mediaPlayer 添加 MediaPlayer.OnPreparedListener 和 MediaPlayer.OnCompletionListener，监听准备完毕和播放结束的消息。

首先在 MusicPlayerService 中，当更改状态时，需要发送出一个广播，如代码清单 10-16 所示。

代码清单 10-16 发送 BroadcastReceiver

```
.....
MediaPlayer.OnCompletionListener mCompleteListener=new MediaPlayer.OnCompletionListener()
{
    public void onCompletion(MediaPlayer mp)
    {
        broadcastEvent(PLAY_COMPLETED);
    }
};
MediaPlayer.OnPreparedListener mPrepareListener=new MediaPlayer.OnPreparedListener()
{
    public void onPrepared(MediaPlayer mp)
    {
        broadcastEvent(PLAYER_PREPARE_END);
    }
};
private void broadcastEvent(String what)
{
    Intent i = new Intent(what);
    sendBroadcast(i);
}
.....
```

当然在发送了这些广播之后，需要在 MusicList 中处理这些接收到的广播，实现对界面的更新。如代码清单 10-17 所示。

代码清单 10-17 BroadcastReceiver 处理

```
protected BroadcastReceiver mPlayerEvtReceiver = new BroadcastReceiver()
{
    public void onReceive(Context context, Intent intent)
```

```

{
    String action = intent.getAction();
    if (action.equals(MusicPlayerService.PLAYER_PREPARE_END))
    {
        mTextView.setVisibility(View.INVISIBLE);
        mPlayPauseButton.setVisibility(View.VISIBLE);
        mStopButton.setVisibility(View.VISIBLE);
        mPlayPauseButton.setText(R.string.pause);
    }
    else if (action.equals(MusicPlayerService.PLAY_COMPLETED))
    {
        mPlayPauseButton.setText(R.string.play);
    }
}
};

```

4. AndroidManifest.xml (权限设置)

由于我们在应用中使用了 Service 来播放音乐，所以需要在 AndroidManifest.xml 中进行注册，另外程序中 MusicInfoController 采用单例模式，使程序中只有唯一的实例。我们传入 MediaPlayerApp 作为 Context 生成 ContentResolver，用来查询媒体库。因此需要修改 AndroidManifest.xml 文件，代码如下：

```

<service android:name=".MusicPlayerService" android:exported="true" >
</service>

```

上面分析了这个音乐播放器中的核心代码，剩下的就是完成一些按钮的事件处理，即可完成这个简易的播放器。大家可以参见第 7 章，以便更好地理解本节的相关内容。

10.5 天气预报

天气预报就是对未来时期内天气变化的预先估计和预告。它是根据大气科学的基本理论和技术对某一地区未来的天气作出分析和预测，这是大气科学为国民经济建设和人民生活服务的重要手段，准确及时的天气预报对于经济建设和国防建设的趋利避害、保障人民生命财产安全等有极大的社会和经济效益。按照天气预报的时限分：1~2 天为短期天气预报，3~15 天为中期天气预报，月、季为长期天气预报，1~6 小时之内则为短临预报（临近预报）。目前的天气学方法以天气图为主，配合气象卫星云图、雷达等资料；数值天气预报以计算机为工具，通过求解由流体力学、热力学、动力气象学组成的预报方程，来制作天气预报；统计预报以概率论数理统计为手段来制作天气预报。本节我们将学习如何在 Android 上完成一个天气预报的应用，以便随时随地获取最新的天气信息。具体实现参见本书所附代码：第 10 章\CityWeather。

1. 信息来源

要实现一个天气预报的应用，肯定需要找到一个提供天气信息的 Web 服务，当然现在提供天气预报的网站很多，方式也是多种多样。这里为了讲述 Android 平台有关 XML 解析的方法，选择了 Google 提供的天气预报，其获得方式有两种：

(1) 通过经纬度来定位获得该地区的天气信息。

在浏览器中输入 <http://www.google.com/ig/api?hl=zh-cn&weather=,,,30670000,104019996> (30670000 和 104019996 分别表示经度和纬度的数据)，即可获得一个 XML 形式的数据，其中包括了天气信

息,如图 10-16 所示。通过网络获得这些信息之后,只需将这些 XML 信息进行解析就可以得到我们需要的天气信息了。



图 10-16 有关天气信息的 XML 属性

(2) 通过城市的名字来获取天气信息。

在浏览器中输入 http://www.google.com/ig/api?hl=zh_cn&weather=chengdu (chengdu 代表城市的名字,这里输入的是成都),就可以获得与第一种方式一样的数据信息。可以看出,这两种方式获得的数据都包含了 forecast_information,表示当前的一些信息,比如城市、时间等,current_conditions 表示当前的实时天气预报,还有 4 个 forecast_conditions 表示预报的后 4 天的天气信息。

2. UI 设计

上面我们找到了天气预报的信息来源,也看到了是什么内容的信息。下面需要根据信息的内容来设计适合我们程序的界面。从图 10-16 中可以看到这些信息包括最高(低)温度、一个 ICNO 图标(表示天气信息的小图标)、一些附加的描述。也就是说,我们至少需要一个 ImageView 来显示这些图标,一个 TextView 来显示另外的温度和附加信息,因此先创建一个 SingleWeatherInfoView 继承自 LinearLayout 来自定义一个线性布局以显示所获取的信息。其中包括一个 ImageView 和一个

TextView。只是需要注意这个 ImageView 是通过网络获取的图片，所以在 SingleWeatherInfoView 类中定义 setWeatherIcon 方法专门来设置图片，代码如下：

```
public void setWeatherIcon(URL aURL){
    try{
        URLConnection conn = aURL.openConnection();
        conn.connect();
        InputStream is = conn.getInputStream();
        BufferedInputStream bis = new BufferedInputStream(is);
        Bitmap bm = BitmapFactory.decodeStream(bis);
        bis.close();
        is.close();
        this.myWeatherImageView.setImageBitmap(bm);
    }catch (Exception e){}
```

该例中我们将分别使用两种方式来获得数据信息，因此需要一个 Spinner 来显示一些城市供用户选择，还需要一个 EditText 来供用户输入城市名字，具体布局如图 10-17 所示。

然后我们将获得天气预报信息，并通过上面我们自定义的 SingleWeatherInfoView 视图显示出来，如图 10-18 所示。

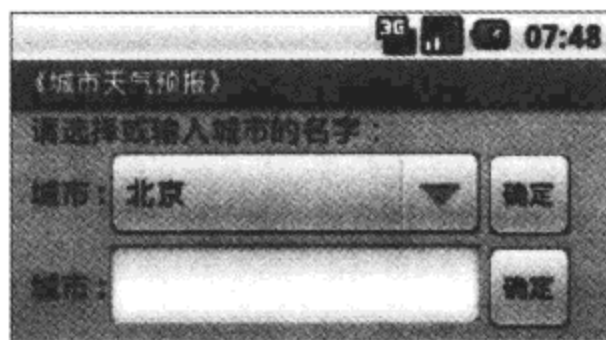


图 10-17 选择和输入城市

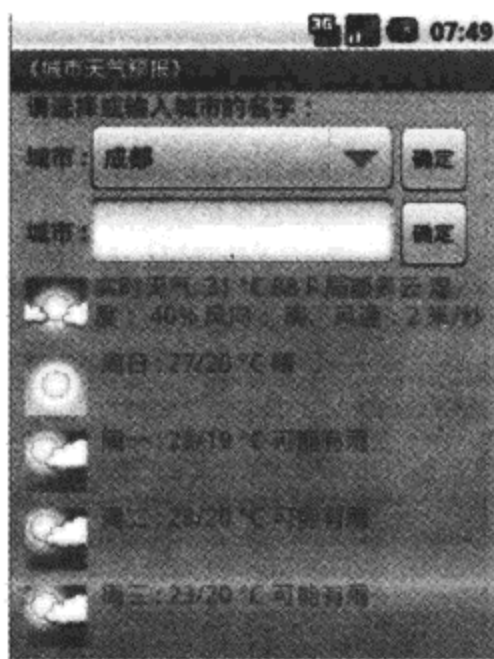


图 10-18 取得的天气预报信息

3. 解析 XML

该实例的核心内容就在于如何解析 XML。Android 平台最大的一个优势在于它利用了 Java 编程语言。Android SDK 并未向标准 Java Runtime Environment (JRE) 提供一切可用功能，但它支持其中很大一部分功能。Java 平台支持通过许多不同的方式来使用 XML，并且大多数与 XML 相关的 Java API 在 Android 上得到了完全支持。举例来说，Java 的 Simple API for XML (SAX) 和 Document Object Model (DOM) 在 Android 上都是可用的。这些 API 多年以来一直都是 Java 技术的一部分，已经是相当稳定的技术。较新的 Streaming API for XML (StAX) 在 Android 中并不可用。但是，Android 提供了一个功能相当的库。最后，Java XML Binding API 在 Android 中也不可用。这个 API 已确定可以在 Android 中实现，但是，它更倾向于是一个重量级的 API，需要使用许多不同类的实例来表示 XML 文档。因此，对于受限的环境，比如说 Android 针对的手持设备，不太理想。下面我们将先介绍 Android 中 3 种解析 XML 的方法，然后选择一种来完成天气预

报程序。

(1) 使用 DOM 方式来解析 XML。

DOM 是 Document Object Model 的缩写，即文档对象模型。XML 将数据组织为一棵树，所以 DOM 就是对这棵树的一个对象描述。通俗地说，就是通过解析 XML 文档，为 XML 文档在逻辑上建立一个树模型，树的节点是一个个对象。我们通过存取这些对象就能够存取 XML 文档的内容。DOM 解析器是通过将 XML 文档解析成树状模型并将其放入内存来完成解析工作的，而后对文档的操作都是在这个树状模型上完成。这个在内存中的文档树将是文档实际大小的几倍。这样做的好处是结构清楚、操作方便，而带来的麻烦就是极其耗费系统资源。

要使用 DOM 方式来解析 XML，就需要引入如下两个包：

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
```

其中 javax.xml.parsers 包含有 DOM 解析器和 SAX 解析器的具体实现。org.w3c.dom 包中定义了 W3C 所制定的 DOM 接口。

要将 XML 的内容解析到一个个的 Java 对象中去供程序使用，首先需要创建一个 DocumentBuilderFactory，代码如下：

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

使用 DocumentBuilderFactory 是为了创建与具体解析器无关的程序，当 DocumentBuilderFactory 类的静态方法 newInstance() 被调用时，它根据一个系统变量来决定具体使用哪一个解析器。又因为所有的解析器都服从于 JAXP 所定义的接口，所以无论具体使用哪一个解析器，代码都是一样的。所以当在不同的解析器之间进行切换时，只需要更改系统变量的值，而不用更改任何代码。这就是工厂对象所带来的好处。

当获得一个工厂对象后，使用它的静态方法 newDocumentBuilder() 可以获得一个 DocumentBuilder 对象，这个对象代表了具体的 DOM 解析器。代码如下：

```
DocumentBuilder db = dbf.newDocumentBuilder();
```

现在我们就可以利用这个解析器来对 XML 文档进行解析了，DocumentBuilder 的 parse() 方法接收一个 XML 文档名作为输入参数，返回一个 Document 对象，这个 Document 对象就代表了一个 XML 文档的树模型。以后所有对 XML 文档的操作，都与解析器无关，直接在这个 Document 对象上进行操作就可以了。而对 Document 操作的具体方法，就是由 DOM 所定义的了。Document 对象的 normalize() 可以去掉 XML 文档中作为格式化内容的空白而映射在 DOM 树中的不必要的 Text Node 对象，否则得到的 DOM 树可能并不如你所想象的那样。特别是在输出的时候，这个 normalize() 更为有用。

```
Document doc = db.parse("xxxx.xml");
doc.normalize();
```

现在可以通过 Document 的 getElementsByTagName 方法来取得一个节点的 NodeList 对象，代码如下：

```
Nodelist nodelist = doc.getElementsByTagName("节点");
```

然后用 item() 方法提取想要的元素，并输出每个元素的数据，代码如下：

```
for (int i=0; i<nodelist.getLength(); i++) {
    Element element = (Element)nodelist.item(i);
```

```
String str=element.getElementsByTagName("元素名称").item(0).getFirstChild().
getNodeValue();
}
```

这样就可以得到我们需要的数据了，当然还可以修改一个 DOM 树中的值，然后重新写入到一个 XML 文件中去。Document 对象中的 createElement 方法可以创建一个元素，如下代码即是创建一个节点，并对其进行赋值。

```
Element element=doc.createElement("节点");
Element elementtext=doc.createElement("text");
String textseg=doc.createTextNode(text);
elementtext.appendChild(textseg);
element.appendChild(elementtext);
```

在创建好节点之后，就可以将修改的内容写入到 XML 文件中。Transformer 类的 transform 方法接收两个参数、一个数据源 Source 和一个输出目标 Result，这里数据源和输出目标分别使用的是 DOMSource 和 StreamResult，这样就能够把 DOM 的内容输出到一个输出流中，当这个输出流是一个文件的时候，DOM 的内容就被写入到文件中去了。代码如下：

```
TransformerFactory tFactory =TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new java.io.File("text.xml"));
transformer.transform(source, result);
```

(2) 使用 XmlPullParser 来解析 XML。

Android 并未提供对 Java SAX API 的支持，但是，Android 确实附带了一个 XmlPullParser 解析器，其工作方式类似于 SAX。它允许应用程序代码从解析器中获取事件，这与 SAX 解析器自动将事件推入处理程序相反。XmlPullParser 是 Android 平台标准的 XML 解析器，它的正确名字应该是“XML PULL PARSER”，这项技术来自一个开源的 XML 解析 API 项目 XMLPULL。下面是使用 XmlPullParser 来解析一个 XML 文件的方法，如代码清单 10-18 所示。

代码清单 10-18 getXML 方法

```
Public void getXML(String url) throws
XmlPullParserException, IOException, URISyntaxException{
    XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
    factory.setNamespaceAware(true);
    XmlPullParser parser = factory.newPullParser();
    parser.setInput(new InputStreamReader(getUrlData(url)));
    XmlUtils.beginDocument(parser,"results");
    int eventType = parser.getEventType();
    do{
        XmlUtils.nextElement(parser);
        parser.next();
        eventType = parser.getEventType();
        if(eventType == XmlPullParser.TEXT){
            String str= "";
            str+=parser.getText();
        }
    } while (eventType != XmlPullParser.END_DOCUMENT);
}
```

XmlPullParser 解析器的运行方式与 SAX 解析器相似。它提供了类似的事件（开始元素和结束元素），但需要使用 parser.next()方法来提取它们。事件将作为数值代码被发送，因此可以根据不同的事

件代码值来进行不同的处理。代码清单 10-21 中通过 `parser.getEventType()` 方法来取得事件的代码值 (如 `XmlPullParser.START_DOCUMENT`、`XmlPullParser.START_TAG`、`XmlPullParser.END_TAG`)，解析并未像 SAX 解析那样监听元素的结束，而是在开始处完成了大部分处理。当某个元素开始时，可以调用 `parser.nextText()` 从 XML 文档中提取所有的字符数据。

同样，还可以使用 `XmlPullParser` 来创建 XML 文件，要创建 XML 文件需要使用一个 `StringBuilder` 来创建 XML 字符串，代码清单 10-19 就是用 `XmlPullParser` 解析器来创建一个 XML 文件并写入数据。

代码清单 10-19 writeXml 方法

```
private String writeXml(List<Message> messages){
    XmlSerializer serializer = Xml.newSerializer();
    StringWriter writer = new StringWriter();
    try {
        serializer.setOutput(writer);
        serializer.startDocument("UTF-8", true);
        serializer.startTag("", "messages");
        serializer.attribute("", "number", String.valueOf(messages.size()));
        for (Message msg: messages){
            serializer.startTag("", "message");
            serializer.attribute("", "date", msg.getDate());
            serializer.startTag("", "title");
            serializer.text(msg.getTitle());
            serializer.endTag("", "title");
            serializer.startTag("", "url");
            serializer.text(msg.getLink().toExternalForm());
            serializer.endTag("", "url");
            serializer.startTag("", "body");
            serializer.text(msg.getDescription());
            serializer.endTag("", "body");
            serializer.endTag("", "message");
        }
        serializer.endTag("", "messages");
        serializer.endDocument();
        return writer.toString();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

(3) 使用 SAX 来解析 XML 文件。

在 Java 环境中，当需要一个速度快的解析器并且希望最大限度地减少应用程序的内存占用时，通常可以使用 SAX API，这非常适用于运行 Android 的移动设备。可以在 Java 环境中照原样使用 SAX API，不需要做任何修改地在 Android 上运行它。本例中我们将采用 SAX 来解析 XML 文件，获取天气预报信息。

SAX 是 Simple API for XML 的缩写，它并不是由 W3C 官方提出的标准。实际上，它是一种社区讨论的产物。虽然如此，在 XML 中对 SAX 的应用丝毫不比 DOM 少，几乎所有的 XML 解析器都支持它。与 DOM 比较而言，SAX 是一种轻量型的方法。我们知道，在处理 DOM 的时候，需要读入整个 XML 文档，然后在内存中创建 DOM 树，生成 DOM 树上的每个 Node 对象。当文档比较小的时候，这不会造成什么问题，但是一旦文档大起来，处理 DOM 就会变得相当费时、费力。特别是其对于内存的需求，也将是成倍地增长，以至于在某些应用中（比如在 applet 中）使用 DOM

是一件很不划算的事。这时候，一个较好的替代解决方法就是 SAX。

我们创建一个继承于 `DefaultHandler` 的 `GoogleWeatherHandler` 类，要使用 SAX 来解析 XML，首先需要引入以下几个包：

```
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.*;
import java.io.*;
```

当遇到一个开始标签的时候，在 `startElement()` 方法中可以根据不同的标签来取得不同的数据信息存放到一个我们自定义的用来保存获取的天气信息的 `WeatherSet` 类中，具体实现见代码清单 10-20 所示。

代码清单 10-20 `startElement` 方法

```
public void startElement(String uri, String localName, String name, Attributes
attributes) throws SAXException
{
    if (localName.equals(CURRENT_CONDITIONS))
    { // 实时天气
        Log.i("localName+CURRENT", localName);
        this.myWeatherSet.setMyCurrentCondition(new WeatherCurrentCondition());
        Log.i("localName+CURRENT+1", localName);
        this.is_Current_Conditions = true;
    }
    else if (localName.equals(FORECAST_CONDITIONS))
    { // 预报天气
        this.myWeatherSet.getMyForecastConditions().add(new WeatherForecastCondition());
        this.is_Forecast_Conditions = true;
    }
    else
    {
        // 分别将得到的信息设置到指定的对象中
        if (localName.equals(CURRENT_CONDITIONS))
        {
            Log.i("localName+CURRENT", localName);
        }
        String dataAttribute = attributes.getValue("data");
        if (localName.equals("icon"))
        {
            if (this.is_Current_Conditions)
            {
                this.myWeatherSet.getMyCurrentCondition().setIcon(dataAttribute);
            }
            else if (this.is_Forecast_Conditions)
            {
                this.myWeatherSet.getLastForecastCondition().setIcon(dataAttribute);
            }
        }
        else if (localName.equals("condition"))
        {
            if (this.is_Current_Conditions)
            {
                this.myWeatherSet.getMyCurrentCondition().setCondition(dataAttribute);
            }
            else if (this.is_Forecast_Conditions)
            {
                this.myWeatherSet.getLastForecastCondition().setCondition(dataAttribute);
            }
        }
    }
}
```

```

        {
            this.myWeatherSet.getLastForecastCondition().setCondition(dataAttribute);
        }
    }
    else if (localName.equals("temp_c"))
    {
        this.myWeatherSet.getMyCurrentCondition().setTemp_celcius(dataAttribute);
    }
    else if (localName.equals("temp_f"))
    {
        this.myWeatherSet.getMyCurrentCondition().setTemp_fahrenheit(dataAttribute);
    }
    else if (localName.equals("humidity"))
    {
        this.myWeatherSet.getMyCurrentCondition().setHumidity(dataAttribute);
    }
    else if (localName.equals("wind_condition"))
    {
        this.myWeatherSet.getMyCurrentCondition().setWind_condition(dataAttribute);
    }
    else if (localName.equals("day_of_week"))
    {
        this.myWeatherSet.getLastForecastCondition().setDay_of_week(dataAttribute);
    }
    else if (localName.equals("low"))
    {
        this.myWeatherSet.getLastForecastCondition().setLow(dataAttribute);
    }
    else if (localName.equals("high"))
    {
        this.myWeatherSet.getLastForecastCondition().setHigh(dataAttribute);
    }
}
}
}

```

以上使用了同 DOM 一样的设计技巧，在创建 SAXParser 对象的时候，通过一个 SAXParserFactory 类来创建具体的 SAXParser 对象，这样，当需要使用不同解析器的时候，要改变的只是一个环境变量的值，而程序的代码可以保持不变。这就是 FactoryMethod 模式的思想。代码如下：

```
SAXParserFactory spf = SAXParserFactory.newInstance();
```

在获得了 SAXParserFactory 对象之后，要解析 XML 还需要一个 SAXParser 或者 XMLReader，SAXParser 是 JAXP 中对 XMLReader 的一个封装类，而 XMLReader 是定义在 SAX 2.0 中的一个用来解析文档的接口。你可以同样调用 SAXParser 或者 XMLReader 中的 parser() 方法来解析文档，效果是完全一样的。不过 SAXParser 中的 parser() 方法接收更多的参数，可以对不同的 XML 文档数据来源进行解析，因而使用起来要比 XMLReader 方便一些。

```
SAXParser sp = spf.newSAXParser();
XMLReader xr = sp.getXMLReader();
```

在创建了 XMLReader 之后，就可以使用上一个步骤创建的 GoogleWeatherHandler 来解析 XML，具体操作代码如下：

```
GoogleWeatherHandler gwh = new GoogleWeatherHandler();
xr.setContentHandler(gwh);
InputStreamReader isr = new InputStreamReader(url.openStream(), "GBK");
```

```

    InputSource is = new InputSource(isr);
    xr.parse(is);

```

4. AndroidManifest.xml (权限设置)

由于我们的信息是通过网络获取的，所以需要在 `AndroidManifest.xml` 中注册访问网络的权限“`<uses-permission android:name="android.permission.INTERNET"></uses-permission>`”。

最后将解析 XML 得到的数据更新到我们自定义的视图上，展示给用户即可完成从网络获取天气信息的程序。本节的重点内容在于如何解析 XML 文件，该例只是使用了 Android 中的其中一种方式，大家可以根据我们所讲述的其他两种方法来完成这个天气预报程序。这三种方式都有各自的优势，究竟使用哪一种方式，还需要根据应用程序的实际需要来确定。大多数情况下，使用 SAX 是比较安全的，并且 Android 提供了一种传统的 SAX 使用方法以及一个便捷的 SAX 包装器。如果文档比较小，那么 DOM 可能是一种比较简单的方法。如果文档比较大，但只需要文档的一部分，则 XML PULL 解析器可能是更为有效的方法。

10.6 个人地图

利用 Google 的地图服务，可以很轻松地获得周边的一些商铺、交通状况等信息。上一章我们学习了 Android 的特色开发，其中之一就是使用 Google Map API 来实现的地图服务，其中分别学习了通过 Android 提供的 `MapView` 来浏览地图程序和使用 `Location` 来实现定位功能。本节我们将实现一个移动版的个人地图，具体实现参见本书所附代码：第 10 章\MobileMap。

1. UI 设计

任何一个应用程序都需要设计出漂亮的 UI，这才是吸引用户的第一因素，但是在地图开发中 UI 设计很简单，只需要一个 `MapView` 来显示地图即可，如代码清单 10-21 所示。然后再加入一些功能菜单，使应用程序更加完美，如图 10-19 所示；Android 提供了 3 种地图模式，所以需要设置一个菜单来切换地图的模式，如图 10-20 所示；也可以设置一些城市名字来供用户选择，如图 10-21 所示；当然也许用户没有自己要选择的的城市，所以再设置一个供用户输入城市名字的输入框，如图 10-22 所示。

代码清单 10-21 main.xml

```

<?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"
    >
    <view class="com.google.android.maps.MapView"
        android:id="@+id/MapView01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="0dYpmIXGI dwiVm-HEpzuUW2fjNYsFQ9EvYirlsg"
        />
</LinearLayout>

```



图 10-19 地图浏览界面

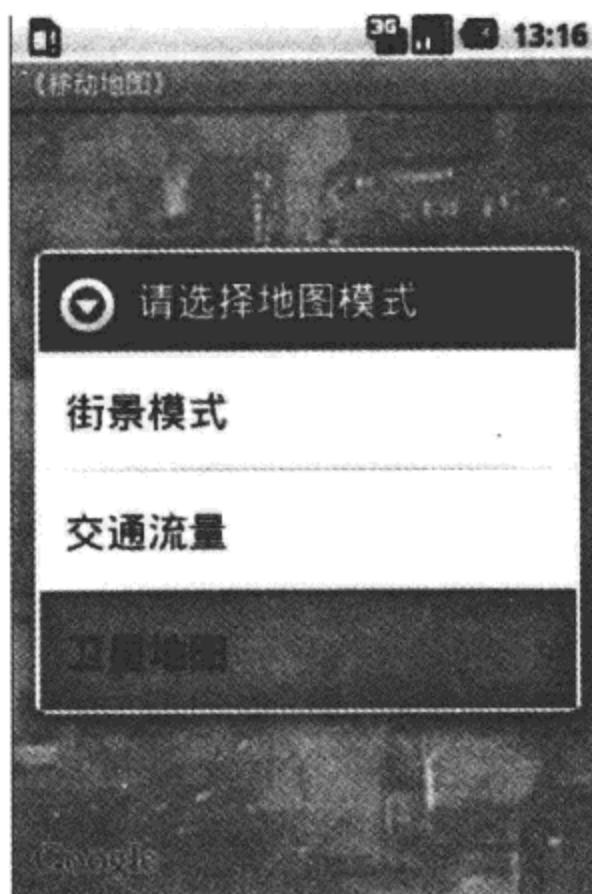


图 10-20 地图模式切换

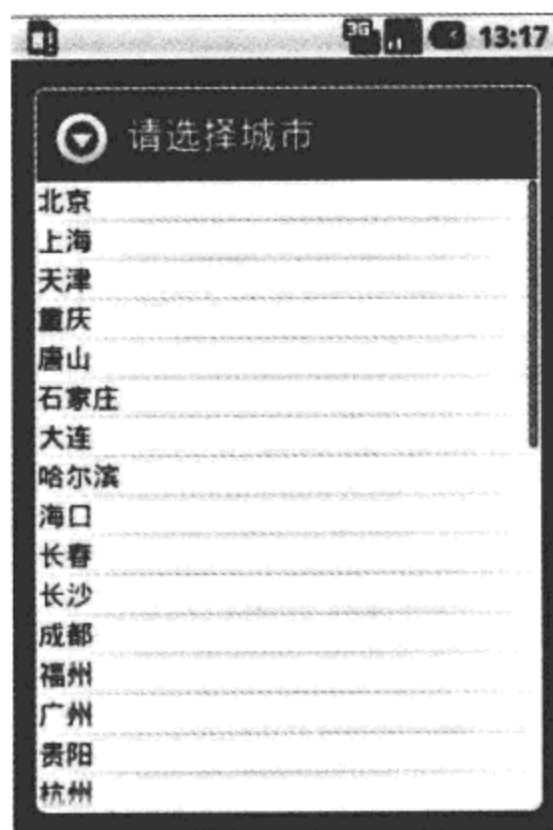


图 10-21 城市选择界面



图 10-22 输入城市名字界面

2. Overlay

该例中，我们在定位之后显示出当前地点的地址以及一个标示，所以需要使用 Overlay 来绘制地图上的信息。在绘制之前需要得到当前地点的地址信息，Geocoder 为我们提供了 `getFromLocation` 方法来获得地址等相关信息，因此可以使用如代码清单 10-22 所示的方法来获取当前的地址。

代码清单 10-22 getAddresses()方法

```

public String getAddresses()
{
    String addressString="没有找到地址";
    Geocoder gc=new Geocoder(mMobileMap,Locale.getDefault());
    try
    {
        List<Address>addresses=gc.getFromLocation(mLocation.getLatitude(),mLocation.
        getLongitude(),1);
        StringBuilder sb=new StringBuilder();
        if(addresses.size()>0)
        {
            Address address = addresses.get(0);
            sb.append("地址: ");
            for(int i = 0; i < address.getMaxAddressLineIndex(); i++)
            {
                sb.append(address.getAddressLine(i));
            }
            addressString=sb.toString();
        }
    }catch(IOException e){}
    return addressString;
}

```

要在 MapView 上绘制信息,需要重写 Overlay 的 Draw 方法,可以通过 toPixels 方法将经度和纬度的信息转换成屏幕上对象的像素坐标信息,绘制方法和一般的绘制图形一样,这里可以通过 paint.setFlags(Paint.ANTI_ALIAS_FLAG)来消除字体的锯齿效果。

3. MapActivity

MapActivity 用于浏览地图,定位信息处理以及一些菜单事件的处理,我们的程序中定义了一些城市的名字供用户选择,在选择之后将通过该城市所对应的经度和纬度的数据来进行定位。当用户输入城市名字的时候,可以通过 Geocoder.getFromLocationName 方法将名字转换成地址,然后找到最合适的城市进行定位,如代码清单 10-23 所示。

代码清单 10-23 searchName 方法

```

public void searchName(String nameStr)
{
    try
    {
        List<Address> addresses= mGeocoder.getFromLocationName(nameStr, 1);
        if (addresses.size() != 0)
        {
            Address address = addresses.get(0);
            GeoPoint geoPoint=new GeoPoint((int)(address.getLatitude()
            * 1E6), (int)(address.getLongitude() * 1E6));
            mLocation.setLatitude(address.getLatitude());
            mLocation.setLongitude(address.getLongitude());
            mLocationOverlay.setLocation(mLocation);
            mMapController.animateTo(geoPoint);
        }
    }
    catch (IOException e){}
}

```


4. AndroidManifest.xml (权限设置)

由于我们使用了 `com.google.android.maps`，所以需要在 `AndroidManifest.xml` 中声明。在使用地图服务的同时我们还使用了网络和定位信息，都需要注册，代码如下：

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

由此可以看出，要在 Android 上开发地图程序其实很简单，但是需要注意的是，如果应用程序需要发布，那么需要申请 Google 的 API Key，因为发布的程序不能使用这个测试用的 APK Key。我们还可以在这个地图上实现轨迹的计算和一些常用的信息查询，请大家自行完成这些功能，做出一个更加完整的个人地图。

10.7 Widget 日历

通过上一章对桌面组件的学习，我们了解了 Widget 开发的基本知识，本节将实现一个 Widget 日历，用来显示当前的日期、时间等信息（具体实现参见本书所附代码：第 10 章 \TodayDate），将有关 Widget 开发的知识进行总结和综合运用。来看看我们实现的 Widget 日历的效果，如图 10-23 所示。要综合学习 Widget 开发，首先需要彻底了解 AppWidget 的框架结构。AppWidget 是 Android 1.5 平台推出的一种崭新的应用程序框架。基于该框架，开发者可以在 OPhone 及模拟器上开发“外形”类似传统 Widget 的小应用程序，并将其嵌入到其他应用中。下面我们分析 AppWidget 框架的主要内容。



图 10-23 Widget 日历效果

1. AppWidget 框架

AppWidget 框架主要包括以下类：

- **AppWidgetProvider**: 继承自 `BroadcastReceiver`，在 AppWidget 应用 `update`、`enable`、`disable` 和 `deleted` 时接收通知。其中，`onUpdate`、`onReceive` 是最常用到的方法，它们接收更新通知。
- **AppWidgetProviderInfo**: 描述 AppWidget 的大小、更新频率和初始界面等信息，以 XML 文件形式存在于应用的 `res/xml/` 目录下。
- **AppWidgetManager**: 负责管理 AppWidget，向 AppWidgetProvider 发送通知。
- **RemoteViews**: 一个可以在其他应用进程中运行的类，是构造 AppWidget 的核心。

2. Widget 开发流程

在学习了 AppWidget 框架之后，我们就可以完成一个 Widget 应用了。下面我们以实现如图 10-23 所示的日历为例，来学习 Widget 开发的具体流程。

我们都知道 Widget 是一个实现在桌面的小应用程序，既然是显示在桌面，对 UI 的设计要求就非常高，其中包括大小的计算和界面是否适合屏幕的布局等。因此我们首先建立 RemoteViews 对应的布局文件，本例中需要 3 个 `TextView` 来显示日期。在完成了 RemoteViews 布局之后，在 `res/xml/`

目录下添加一个能够反映出 AppWidgetProviderInfo 信息的布局文件。AppWidget 的像素大小取决于它所占的方块多少，其计算公式是(块数*74) - 2，因此，我们取高度是 146 像素，宽度是 146 像素。android:initialLayout 设置了 AppWidget 的布局文件，即刚刚创建的 RemoteViews 对应的布局文件 widget_layout.xml，如代码清单 10-24 所示。

代码清单 10-24 widget.xml

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialLayout="@layout/widget_layout"
    android:minWidth="146dip"android:minHeight="146dip"android:updatePeriodMillis="3600000">
</appwidget-provider>
```

接下来需要创建一个继承自 AppWidgetProvider 的类，当我们添加 AppWidget 应用或自动定时更新时，AppWidgetManager 会广播动作名字是“android.appwidget.action.APPWIDGET_UPDATE”的 Intent，当 onReceive()方法没有被重载时，onUpdate 方法会接收到这些广播的 Intent。类似于普通 BroadcastReceiver 类，我们可以重载 AppWidgetProvider 的 onReceive 方法，并在其中指定想要接收的 Intent。本例所实现的 TodayDate 继承自 AppWidgetProvider，具体代码如代码清单 10-25 所示。

代码清单 10-25 第 10 章\TodayDate\src\com\lyarin\android\TodayDate\TodayDate.java

```
public class TodayDate extends AppWidgetProvider
{
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
    {
        RemoteViews updateView = buildUpdate(context);
        appWidgetManager.updateAppWidget(appWidgetIds, updateView);
        super.onUpdate(context, appWidgetManager, appWidgetIds);
    }

    private String[] months = {"一月", "二月", "三月", "四月",
                                "五月", "六月", "七月", "八月",
                                "九月", "十月", "十一月", "十二月"};
    private String[] days = {"星期日", "星期一", "星期二", "星期三",
                              "星期四", "星期五", "星期六"};
    private RemoteViews buildUpdate(Context context)
    {
        RemoteViews updateView = null;
        Time time = new Time();
        time.setToNow();
        String month = months[time.month] + " " + time.year;
        updateView = new RemoteViews(context.getPackageName(),
            R.layout.widget_layout);
        updateView.setTextViewText(R.id.Date, new Integer(time.monthDay).toString());
        updateView.setTextViewText(R.id.Month, month);
        updateView.setTextViewText(R.id.WeekDay, days[time.weekDay]);
        Intent launchIntent = new Intent();
        launchIntent.setComponent(new ComponentName("com.android.calendar",
            "com.android.calendar.LaunchActivity"));
        launchIntent.setAction(Intent.ACTION_MAIN);
        launchIntent.addCategory(Intent.CATEGORY_LAUNCHER);
        launchIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_
            ACTIVITY_RESET_TASK_IF_NEEDED);
        PendingIntent intent = PendingIntent.getActivity(context, 0,
            launchIntent, 0);
        updateView.setOnClickPendingIntent(R.id.Base, intent);
    }
}
```

```

        return updateView;
    }
}

```

3. AndroidManifest.xml (权限设置)

我们使用了 receiver 及其 “android.appwidget.action.APPWIDGET_UPDATE”，所以需要在 AndroidManifest.xml 中进行声明，如代码清单 10-26 所示。

代码清单 10-26 AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.yarin.android.TodayDate"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver android:label="@string/app_name" android:name=".TodayDate">
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE"></action>
            </intent-filter>
            <meta-data android:name="android.appwidget.provider" android:resource=
                "@xml/widget"></meta-data>
        </receiver>
        <receiver android:label="@string/app_name_small" android:name=".TodayDateSmall">
            <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE"></action>
            </intent-filter>
            <meta-data android:name="android.appwidget.provider" android:resource=
                "@xml/widget_small"></meta-data>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>

```

4. Widget 事件处理

当然本例中只是更新显示了一个界面，在 Widget 中同样可以通过 setOnClickPendingIntent 方法来监听一个事件的处理，如代码清单 10-27 所示。当我们点击 “widget_button” 按钮时，会弹出一个 Toast 提示。

代码清单 10-27 Widget 按钮事件处理

```

public class widget extends AppWidgetProvider
{
    public void onReceive(Context context, Intent intent)
    {
        super.onReceive(context, intent);
        if(intent.getAction().equals("com.android.myapp.widget.CLICK"))
        {
            Toast.makeText(context, "It works!!", Toast.LENGTH_SHORT).show();
        }
    }
    public void onUpdate(Context context, AppWidgetManager mgr, int[] appWidgetIds)
    {
        final int N = appWidgetIds.length;
        for (int i=0; i<N; i++)
        {
            int[] appWidgetId = appWidgetIds;
            RemoteViews views=new RemoteViews(context.getPackageName(),R.layout.main);
            Intent clickintent=new Intent("com.android.myapp.widget.CLICK");

```

```

        PendingIntent pendingIntentClick=PendingIntent.getBroadcast(context,
        0, clickintent, 0);
        views.setOnClickListener(R.id.widget_button, pendingIntentClick);
        mgr.updateAppWidget(appWidgetId, views);
    }
}

```

5. Widget 启动 Activity

在 Widget 中同样可以启动一个 Activity，比如 Android 自带的音乐播放器，当我们点击时就会开启一个 Activity 来显示播放音乐的界面。通过如下代码即可启动一个 Activity：

```

RemoteViews rv = new RemoteViews(context.getPackageName(), R.layout.mywidget);
Intent intentActivity = new Intent(context,MusicPlayer.class);
PendingIntent pendingIntentActivity=PendingIntent.getActivity(context,0,intentActivity,0);
rv.setOnClickListener(R.id.button, pendingIntentActivity);
appWidgetManager.updateAppWidget(appWidgetId, rv);

```

当我们点击“button”按钮时就可以启动指定的 Activity 了，代码如下：

```
context.startActivity(intentActivity);
```

6. Widget 启动 Service

Widget 还可以启动一个服务，还是以 Android 自带的音乐播放器为例。要播放音乐肯定需要使用 Service 在后台播放，所以就需要在 Widget 中启动一个 Service。使用下面的方法可以在 Widget 中启动一个 Service。

```

RemoteViews rv = new RemoteViews(context.getPackageName(), R.layout.mywidget);
Intent intentService = new Intent("Service名");
context.startService(intentService);
appWidgetManager.updateAppWidget(appWidgetId, rv);

```

Widget 开发其实很简单，只是需要设计好 UI，然后根据需要来完成更新。如果程序不需要使用定时更新策略来更新 AppWidget，则建议不要使用定时更新，因为它会增加电量和 CPU 资源的开销。我们可以自定义一个线程，当需要更新的时候才更新，这样可以让 Widget 达到最高效率。通过本节的学习，相信大家对 Widget 的开发有了更深入的了解，应该可以自己动手来开发一些 Widget 应用了。

10.8 小结

本章是以实例来综合前面所学的知识，使大家能够更加深入地掌握 Android 平台应用开发。当然 Android 平台的应用是丰富多彩的，而本章所安排的实例几乎包括了前面所学的全部基础知识要点。在巩固旧知识的同时，又插入了新的内容，比如前面没有的 Android 平台关于 XML 的解析。每个实例都是按照标准的开发流程进行开发讲解，既让大家能更好地接受，又使大家明白了 Android 应用开发的流程和重点，同时也让大家养成了很好的习惯，在开发过程中不会因为流程的混乱而影响开发的效率和质量等。希望通过本章的学习，大家都能独立开发出自己的 Android 应用。

Android 游戏开发实例

手机游戏被业内人士称为继短信之后的又一座“金矿”。从 2005 年开始，全球手机游戏市场一直以每年 40% 的速度增长，2006 年，手机游戏掘金 9 亿美元，成为超过无线音乐和娱乐内容服务的最大应用。据预测，2010 年中国手机游戏市场总销售额将达到 95.27 亿元，从 2003 年到 2010 年的市场销售额年均复合增长率为 62.4%。可以看出，手机游戏行业将经历一个黄金发展时期。3G 的普及、手机显示性能的提高、手机屏幕的扩大和较高的分辨率，解决了传统手机游戏行业的很多弊端，同时 Android 也采用了键盘形式的按键，可以完成更复杂的操作。Android 虽然基于 Java，但是其虚拟机是基于寄存器的（这与普通的 Java 基于堆栈不同），速度得到了很大的提升，这为 Android 平台上开发游戏提供了坚强的后盾。

11.1 手机游戏开发简介

在这里首先要说的是大家不要认为开发游戏很难，开发游戏其实比开发一些应用更简单，为什么呢？因为游戏的本质就是在屏幕上不断地显示和更新图片，只不过不是胡乱地更新，而是根据程序逻辑来控制，比如要实现一个主角在地图上移动，那么只需要将编辑好的地图先绘制到屏幕上去，然后再将主角绘制上去，剩下的便是逻辑判断了。再如，要让主角移动，当我们按下了前进键时，只需要将主角绘制的位置向前移动就可以达到目的了（在碰撞检测之后主角能够移动的情况下）。但同时也不要就因为这些就认为开发一款游戏很简单，一款完整的游戏需要多方面的知识，比如游戏的创意、背景、故事情节、游戏音效、游戏风格、游戏类型、运行速度、适配机型等。而且，游戏开发是需要策划、美工、程序、测试的协同工作和默契配合完成的。下面我们将详细地介绍游戏开发的流程。

1. 游戏策划

一款游戏的策划是最重要的，它将直接决定该游戏的后期制作和玩家的喜好程度。在早期的游戏开发过程中，或许只是一个或者多个代码工作者将自己封闭起来狂写一段时间的代码就可以产生一款很不错的游戏，但现在如果还是这样，写出来的游戏多半不会受到用户的欢迎。目前游戏行业飞速发展，市面上各种各样的游戏简直是数不胜数，如何才能开发一款受广大用户喜爱的游戏呢？这就需要在策划之前确定市场的需求，比如，将很有中国特色的棋牌游戏“斗地主”拿到美国去销售，就算你做得很好，也许美国人根本就不玩这种游戏。再比如，国家明明禁止了成人游戏和赌博游戏，我们要是开发出这样的游戏能行吗？肯定不能，所以首先要确定目标客户。

在明确了目标客户之后，是否需要考虑游戏的类型呢？从大的方面来说，游戏是单机的还是联网的？是单人的还是多人的？是动作类型的还是角色扮演类型的？等等。在确定了游戏的类型之后，还

需要考虑游戏的可玩性，总不可能说游戏玩家十多分钟就通关了，这就需要包括游戏的难度设置、关卡控制以及后期的版本控制。

现在我们就可以根据上面的目标客户和游戏类型来定义游戏的风格了，比如想以三国为题材做一个即时战略游戏，就不可能定义为现代风格，游戏中就不会出现坦克、飞机的道具。同时这也体现了游戏的真实性，但是游戏本身是一个虚幻的东西，玩家就是需要将自己放到虚拟的世界中，所以也不能过度真实，让玩家觉得枯燥乏味。风格确定之后，还可以根据游戏的风格来配置游戏音效。

由于玩家经常会接触到很多游戏，所以他在玩游戏时会对一些没有新意的游戏感到厌倦，“反正我玩过类似的游戏，没什么好玩的”。如果得到玩家这样的评论，不用我说，大家都知道这款游戏的成败了。虽然游戏创新并不是一件很容易的事情，但是为了吸引玩家，我们不得不大胆地创新。

这些问题都解决了，就可以准备写策划案了。在写策划案的同时还需要考虑到美工和程序在技术上的实现以及硬件的支持，不能设计技术达不到的效果。了解了这些问题，开始编写策划案。

其实策划是一个非常广泛的领域，有很多东西需要自己在实践中证明，这里只是列举了常用的、值得注意的地方。

2. 美工

美工在拿到策划文档之后，需要根据策划文档的描述来发挥自己的想象力，在保持和策划文档一致的情况下，进行创新。同样，为了保证美工的图片适合程序的要求，还需要多和程序员进行交流，以确保程序员能够很清楚地理解自己的设计，使游戏效果达到最好。

3. 程序

程序员在得到文档和资源后并不能马上打开编辑器，新建工程开始写代码，而是要仔细查看文档和资源，根据这些来确定所要使用的知识和所要实现的功能，然后构建一个整体的框架。这个整体的框架很重要，一个优秀的程序员会在框架的设计上花很多时间，因为一个好的框架可以使后面的开发、调试等更加简单，同时一个好的框架还能提高游戏的运行效率。为了保证质量，每个程序员写的程序都有 Bug，所以我们需要不断地测试、修改，再测试、再修改，从而给玩家一个最好的体验，希望大家都能养成这个很好的习惯。

到目前为止，Android 的游戏还很少。Android Market 从 2008 年 10 月 22 日开始运营，从 Google 公布的数字来看，Android Market 中的应用程序数量达到了 5000 个，和 App Store 比起来还是有明显的差距。下面是 Android Market 上面几款游戏的效果，如图 11-1 所示。

在本书第 5 章我们学习了 Android 平台开发游戏的基础知识，本章我们将通过在 Android 平台完成一个风靡全球的经典游戏《魔塔》来学习 Android 游戏开发。该游戏包括了游戏开发中的大部分技术，包括地图、精灵、图层、音效、存档等。下面是本章将完成的游戏在 Android 上的运行效果，如图 11-2 所示。



图 11-1 Android 平台游戏

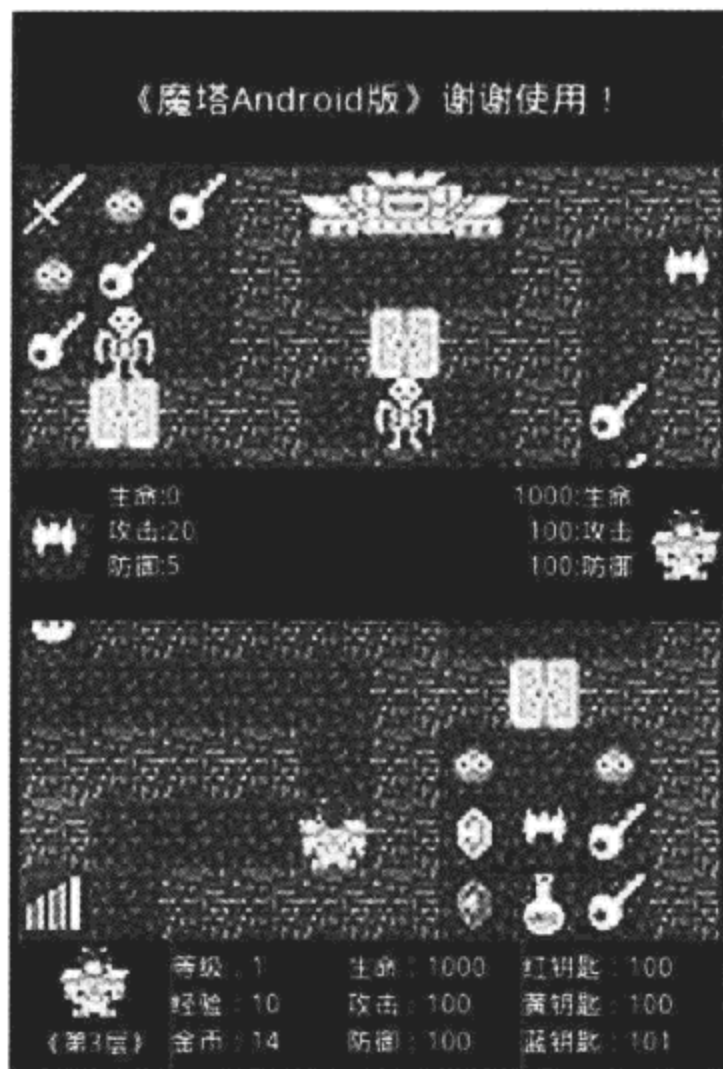


图 11-2 《魔塔》游戏效果

11.2 游戏框架设计

前面已经介绍了框架在游戏开发中的重要地位，如何才能实现一个适合该游戏的框架呢？首先我们需要了解游戏的内容，从图 11-2 中可以大致看出游戏中包括了地图、主角、整个屏幕界面，显示了地图和主角的属性，地图上还有道具，至少需要一个视图来显示，并且需要更新界面的显示和一个控制游戏逻辑及事件的类。下面我们来构建该游戏的整体框架。

我们知道，在 Android 中要显示一个视图类就必须继承自 View 类，View 类中包括一个最主要的绘制方法 onDraw 和一些事件的处理，比如 onKeyDown、onKeyUp 等。当然在构建这个视图类时还可以加入自己的一些抽象方法，比如资源回收 (reCycle)、刷新 (refurbish) 等。有了这些内容，下面构建一个用于显示游戏界面的视图类 GameView，当然该游戏也不只是这么一个界面，还有菜单、道具商店、战斗等，所以这个类应该是一个抽象类 (abstract)，可以被各个界面使用。GameView 类的代码如代码清单 11-1 所示。

代码清单 11-1 第 11 章\MagicTower\src\com\lyarin\android\MagicTower GameView.java 片段

```
public abstract class GameView extends View
{
    public GameView(Context context)
    {
        super(context);
    }
}
```

```

    }
    /**
     * 绘图
     *
     * @param          N/A
     *
     * @return          null
     */
    protected abstract void onDraw(Canvas canvas);
    /**
     * 按键按下
     *
     * @param          N/A
     *
     * @return          null
     */
    public abstract boolean onKeyDown(int keyCode);
    /**
     * 按键弹起
     *
     * @param          N/A
     *
     * @return          null
     */
    public abstract boolean onKeyUp(int keyCode);
    /**
     * 回收资源
     */
    protected abstract void reCycle();

    /**
     * 刷新
     */
    protected abstract void refurbish();
}

```

有了视图类来显示界面，还要控制当前屏幕显示哪一个界面，甚至对界面进行一些逻辑上的处理，这时就可以构建一个整个游戏逻辑的 `MainGame` 类，在该类中需要根据不同的游戏状态来设置屏幕需要显示的视图，如代码清单 11-2 所示，这是 `MainGame` 类中控制界面显示的一段代码。

代码清单 11-2 `controlView` 方法

```

GameView m_GameView = null;
...
public void controlView(int status)
{
    //当状态改变时，释放上一个状态的资源
    if(m_status != status)
    {
        if(m_GameView != null)
        {
            m_GameView.reCycle();
            System.gc();
        }
    }
    //释放当前视图对象
    freeGameView(m_GameView);
    //根据不同的游戏状态判断当前屏幕显示的界面
}

```



```

//当然这些界面都需要继承自我们上面所创建的 GameView
switch (status)
{
case yarin.GAME_SPLASH:
    m_GameView = new SplashScreen(m_Context,this);
    break;
case yarin.GAME_MENU:
    m_GameView = new MainMenu(m_Context,this);
    break;
case yarin.GAME_HELP:
    m_GameView = new HelpScreen(m_Context,this);
    break;
case yarin.GAME_ABOUT:
    m_GameView = new AboutScreen(m_Context,this);
    break;
case yarin.GAME_RUN:
    m_GameView = new GameScreen(m_Context,m_MagicTower,this,true);
    break;
case yarin.GAME_CONTINUE:
    m_GameView=new GameScreen(m_Context,m_MagicTower,this,false);
    break;
}
//更改游戏状态
setStatus(status);
}

```

同样，还需要得到当前要显示的视图对象，以便被其他类使用，因此这里我们实现一个 `getMainView` 方法来取得当前视图对象。代码如下：

```

//得到当前需要显示的对象
public static GameView getMainView()
{
    return m_GameView;
}

```

在创建和控制了视图显示之后，要让游戏能够动起来，需要开启一个线程来实时更新视图显示界面并刷新视图。下面我们将为游戏开启一个主线程，可以通过 `MainGame.getMainView()` 方法来取得当前显示的视图界面，然后根据不同的界面来进行游戏更新，本例中使用了 `postInvalidate()` 方法来刷新一个界面，如代码清单 11-3 所示。

代码清单 11-3 第 11 章\MagicTower\src\com\yarin\android\MagicTower\ThreadCanvas.java 片段

```

public class ThreadCanvas extends View implements Runnable
{
    private String m_Tag = "ThreadCanvas_Tag";
    public ThreadCanvas(Context context) {
        super(context);
    }
    //绘制界面
    protected void onDraw(Canvas canvas) {
        if (MainGame.getMainView() != null) {
            MainGame.getMainView().onDraw(canvas);
        } else {
            Log.i(m_Tag, "null");
        }
    }
    //开启线程
    public void start() {

```

```

        Thread t = new Thread(this);
        t.start();
    }
    // 刷新界面
    public void refurbish(){
        if (MainGame.getMainView() != null) {
            MainGame.getMainView().refurbish();
        }
    }
    public void run(){
        while (true) {
            try{
                Thread.sleep(yarin.GAME_LOOP);
            }catch (Exception e) {
                e.printStackTrace();
            }
            refurbish(); // 更新显示
            postInvalidate(); // 刷新屏幕
        }
    }
    // 按键处理(按键按下)
    boolean onKeyDown(int keyCode) {
        if (MainGame.getMainView() != null) {
            MainGame.getMainView().onKeyDown(keyCode);
        }else{
            Log.i(m_Tag, "null");
        }
        return true;
    }
    // 按键弹起
    boolean onKeyUp(int keyCode) {
        if (MainGame.getMainView() != null) {
            MainGame.getMainView().onKeyUp(keyCode);
        }else{
            Log.i(m_Tag, "null");
        }
        return true;
    }
}

```

在完成了这些模块之后,就需要通知一个 Activity 来显示界面,这里我们是在 ThreadCanvas 中控制界面的显示,所以使用 setContentView 方法来显示一个 ThreadCanvas 类对象即可,当然按键事件的处理也就可以调用 ThreadCanvas 类来处理,ThreadCanvas 类会和 MainGame 一起配合来找到我们所指定的界面,代码清单 11-4 为 MagicTower 类的处理。

代码清单 11-4 第 11 章\MagicTower\src\com\yarin\android\MagicTower\MagicTower.java

```

public class MagicTower extends Activity
{
    private ThreadCanvas mThreadCanvas = null;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //设置全屏风格
        setTheme(android.R.style.Theme_Black_NoTitleBar_Fullscreen);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
        new MainGame(this);
    }
}

```

```

        mThreadCanvas = new ThreadCanvas(this);
        setContentView(mThreadCanvas);
    }
    protected void onPause() {
        super.onPause();
    }
    protected void onResume() {
        super.onResume();
        mThreadCanvas.requestFocus();
        mThreadCanvas.start();
    }
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        mThreadCanvas.onKeyDown(keyCode);
        return false;
    }
    public boolean onKeyUp(int keyCode, KeyEvent event) {
        mThreadCanvas.onKeyUp(keyCode);
        return false;
    }
}

```

到这里我们基本完成了一个游戏的整体框架，后面的所有视图类界面显示都只需要继承自我们自定义的抽象类 `GameView`，然后在 `MainGame` 中判断和更改当前的游戏状态，程序便自动找到我们需要更新和释放的视图类进行操作。下面我们将通过该游戏的菜单显示界面来熟悉该框架的使用，菜单界面如图 11-3 所示。

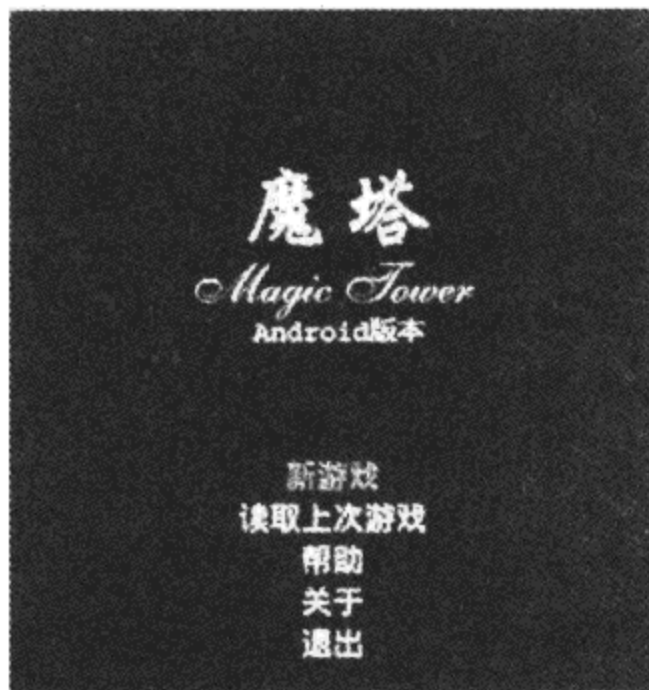


图 11-3 《魔塔》菜单界面

要通过 `GameView` 类来实现该菜单界面，首先需要创建一个 `MainMenu` 类来继承自 `GameView` 类，代码如下：

```

public class MainMenu extends GameView
{
    //...
}

```

既然继承自 `GameView`，就需要实现其中的抽象方法。绘制界面很简单，大家可以参考本书第 5 章的内容，完成界面的绘制，在绘制好界面和处理好按键事件之后，只需要在 `MainGame` 中判断

当前状态即可。如果是菜单界面，那么将 GameView 对象构建为 MainMenu 类型，ThreadCanvas 便可以通过 `MainGame.getMainView()` 取得当前的 MainMenu 界面来进行更新，当有事件触发时便自动找到 MainMenu 中事件相关的方法进行处理。这里需要说明的是，当我们选择了一个菜单项之后，应用程序的界面将跳转到另一个界面，比如选择了“关于”界面，将跳转到“关于”界面中，这时就需要通过“`mMainGame.controlView(yarin.GAME_ABOUT);`”来改变当前状态为“关于”状态，当然以后任何界面的设计都可以继承自 GameView 这样来进行。

图 11-4 是游戏“关于”界面的效果，具体实现参见本书所附源代码：第 11 章\MagicTower。

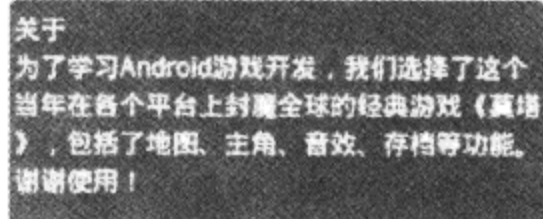


图 11-4 《魔塔》“关于”界面

当然，一个好的游戏框架需要根据具体的游戏类型和内容来进行设计，以方便开发，提高开发效率，使得开发的游戏运行效果更好。现在我们可以点击菜单界面上的“新游戏”来进入游戏的主题开发了。大家可能都知道，J2ME 中在 MDIP2.0 后面的版本多出了一个专门为开发游戏提供的 Game 包，本章我们将通过在 Android 中实现并使用这个包来完成《魔塔》。笔者在这里先实现了 Game 包中的几个重要的类，包括 Layer、TiledLayer、Sprite、LayerManager。这样就可以使用这个包来轻松地完成以后的游戏开发了。没有学习过 J2ME 的读者不用着急，下面我们将介绍如何使用这个包开发游戏。具体实现参见本书所附源代码：第 11 章\MagicTower\src\javax\microedition\lcdui\game。

11.3 地图设计

地图的设计在游戏中非常重要，我们不可能让美工做出一张完整的大地图，然后在程序中直接使用，这样做出来的游戏将会很大，一般的手机不一定能成功地装载这么大的图片。通常游戏中的地图是多个小块组成的一个完整的大地图，而组成这些小块的数据一般可以使用一个二维数组来存储，然后通过程序以最快的方式将这些地图数据对应的小块映射到屏幕上组成一幅完整的地图。当然，这些数据也不是我们从键盘上一个一个地输入进去的，一般情况下先由程序员做一个地图编辑器，在这个地图编辑器中用鼠标点击再保存，或者是从网络上下载一些成熟的编辑器，比如用 mappy 这样的工具生成地图，再用脚本语言为 mappy 写一个应该保存成什么格式的程序。通常地图分为 45 度角、侧视角和俯视角等。如图 11-5 所示，是编辑地图时的一个截图。

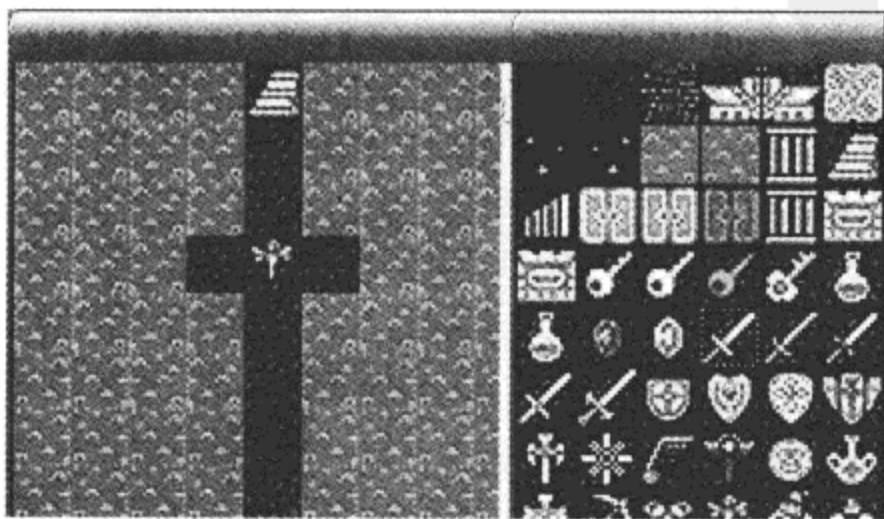


图 11-5 地图编辑

编辑好地图之后，可以得到一个保存了地图数据的二维数组，如下面代码所示：

```
private byte[][] map={
    6,11,6,6,
    6,1,6,6,
    6,6,1,6,
    6,6,1,6};
```

其中，6、1、11 这些代表了不同的图块，我们只需要将这些数据所对应的图块绘制到屏幕上即可。这里将使用前面所实现的 TiledLayer 类来显示地图。

1. 首先创建一个 TiledLayer 对象

创建 TiledLayer 对象需要使用其构造函数：

```
public TiledLayer( int columns, int rows, Bitmap image,
                  int tileWidth, int tileHeight)
```

参数 columns 和 rows 分别表示地图的列数和行数，即我们构建的地图数组的行数和列数；参数 image 表示地图所有图块的一张图片，即我们在编辑地图时所使用的图片；参数 tileWidth 和 tileHeight 分别表示每个图片的宽度和高度。

2. 设置地图数据

创建了 TiledLayer 对象后，可以通过如下方法来设置该地图使用的地图数据：

```
public void setCell(int col, int row, int tileIndex)
```

参数 col 和 row 分别表示在屏幕上要显示的列数和行数；参数 tileIndex 则表示在地图上显示图块的值。比如，如下代码通过一个循环设置了所有要显示的行和列的图块索引值：

```
floorMap=new TiledLayer(TILE_NUM_COL,TILE_NUM_ROW,bmap,TILE_WIDTH, TILE_HEIGHT);
for (int i = 0; i < TILE_NUM; i++)
{
    int[] colrow = getColRow(i);
    floorMap.setCell(colrow[0], colrow[1], floorArray[curFloorNum][i]);
}
```

3. 绘制地图

通过上面的设置，绘制地图就非常简单了，只需要调用 TiledLayer 中的 paint 方法就可以将地图绘制到屏幕上。代码如下：

```
floorMap.paint(canvas);
```

下面我们来看看 TiledLayer 中的 paint 方法是如何实现的，如代码清单 11-5 所示。

代码清单 11-5 第 11 章\MagicTower\src\com\yarin\android\MagicTower\TiledLayer.java 片段

```
public final void paint(Canvas canvas) {
    if (canvas == null) {
        throw new NullPointerException();
    }
    if (visible) {
        int tileIndex = 0;
        int ty = this.y;
        for (int row = 0; row < cellMatrix.length; row++, ty += cellHeight) {
            int tx = this.x;
            int totalCols = cellMatrix[row].length;
            for (int column = 0; column < totalCols; column++, tx += cellWidth) {
                tileIndex = cellMatrix[row][column];
                if (tileIndex == 0) { // transparent tile
                    continue;
                } else if (tileIndex < 0) {
```

```

        tileIndex = getAnimatedTile(tileIndex);
    }
    drawImage(canvas, tx, ty, sourceImage,
        tileSetX[tileIndex], tileSetY[tileIndex], cellWidth, cellHeight);
    }
}
}
//绘制一个图片
private void drawImage(Canvas canvas, int x, int y,
    Bitmap bsrc, int sx, int sy, int w, int h) {
    Rect rect_src = new Rect();
    rect_src.left = sx;
    rect_src.right = sx + w;
    rect_src.top = sy;
    rect_src.bottom = sy + h;
    Rect rect_dst = new Rect();
    rect_dst.left = x;
    rect_dst.right = x + w;
    rect_dst.top = y;
    rect_dst.bottom = y + h;
    //按指定的裁剪矩形来绘制图片
    //分别是屏幕上的矩形和图片上要绘制的矩形裁剪
    canvas.drawBitmap(bsrc, rect_src, rect_dst, null);
    rect_src = null;
    rect_dst = null;
}

```

代码清单 11-5 就是用一个循环来将我们设置的地图数组和图片进行对应，绘制到对应的屏幕上，在 Android 中按指定矩形裁剪图片绘制的方法如下：

```
void android.graphics.Canvas.drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)
```

参数 `bitmap` 是要绘制的图片，参数 `src` 和 `dst` 分别是图片上的裁剪区域和屏幕上对应的裁剪区域，参数 `paint` 设置了画笔的属性。更多方法可以参见我们实现的 `Game` 包中的 `TiledLayer` 类。

11.4 主角设计

游戏中的主角在这里称为“精灵”，当然精灵包括的范围很广，不仅仅是主角，还有 NPC、道具等。既然是精灵，必然有很多动画，比如，主角在向 4 个方向移动时分别对应 4 个不同的动画，动画本身就是将图片一帧一帧地连接起来，循环地播放每一帧形成的。在一些大的游戏中同样可以使用自己写的精灵编辑器去编辑精灵，将精灵拆成很多部分，然后再组合起来，这样可以节省大量的空间。这里可以使用我们实现的 `Sprite` 类来完成《魔塔》中的主角，`Sprite` 是个用来显示图像类，该类和 `TiledLayer` 的区别是：`Sprite` 是由一个图像（可以有好几帧，但是一次只有一个显示）组成的（当然 `Sprite` 还有其他的特性，每次只能使用一个图像而不是多个图像来填充屏幕是它的最主要特征），因而 `Sprite` 被用来定义一些小的有动作的游戏对象（比如飞船和小行星相撞），`TiledLayer` 更常被用来构造生动的背景。下面我们看看如何使用这个 `Sprite` 类。

1. 构建 `Sprite` 对象

在该 `Sprite` 类中提供了 3 个构造方法，分别是：

```
public Sprite(Bitmap image)
```



```
public Sprite(Bitmap image, int frameWidth, int frameHeight)
public Sprite(Sprite s)
```

参数 `image` 为精灵的图片，参数 `frameWidth` 和 `frameHeight` 分别设置了精灵图片的每一帧的宽度和高度，参数 `s` 表示通过一个精灵来创建另一个精灵。构造 `Sprite` 类的时候需要指定精灵的高度和宽度（像素值）。图像的高度和宽度必须分别是精灵的高度和宽度的整数倍。换句话说，要能正好让电脑把图像按照精灵的大小划分成几个类，通过上面的例子也看到了，这些帧是横着排还是竖着排，抑或横竖都有，排成一个方阵，都无所谓。接着就可以指定帧数了，左上方是编号 0，然后从左到右、从上到下依次排列。可以使用 `setFrame(int sequenceIndex)` 选择哪一帧被显示，只要把它的编号作为参数传递即可。

2. Sprite 属性

我们使用的 `TiledLayer` 类可以自动根据精灵的位置来判断地图绘制的位置，因此可以使用如下方法来设置精灵的位置：

```
public void setRefPixelPosition(int x, int y)
```

参数 `x` 和 `y` 是精灵的位置。更多的方法可以参考我们实现的 `Game` 包中的 `Sprite` 类。

注意 `Sprite` 的编号是从 0 开始的，但是 `TiledLayer` 却是从 1 开始的（笔者开始因为没注意得到了一个 `IndexOutOfBoundsException` 异常）。在 `TiledLayer` 中，序号 0 表示一个空白的元素（比如在某个位置你什么都不想画，那就把它设置成 0）。`Sprite` 只由一个单元组成，所以如果你想让它不显示这个单元，简单地设置成 `setVisible(false)` 就可以了，因而 `Sprite` 不需要一个特殊的编号来表示空白的单元。

3. 碰撞检测

精灵类提供了以下 3 个碰撞检测函数：

```
public final boolean collidesWith(TiledLayer t, boolean pixelLevel)
public final boolean collidesWith(Sprite s, boolean pixelLevel)
public final boolean collidesWith(Bitmap image, int x, int y, boolean pixelLevel)
```

上面的 3 个函数分别表示精灵和 `TiledLayer` 的碰撞、精灵和精灵的碰撞、精灵和图片的碰撞。参数 `pixelLevel` 表示使用像素检测还是矩形检测。矩形检测只需要将精灵对应成相应的矩形范围来进行检查，这种检测速度很快，但不是很准确，对于碰撞要求不高的游戏可以使用它。同时还可以将一个 `Sprite` 分解成很多矩形来使用矩形检测以提高准确性。而像素检测则比较准确，但是速度必然会减慢。

4. 精灵旋转和镜像

一般在做游戏时，都需要使用旋转和镜像。比如，在做一个飞机游戏时，飞机有几个方向的图片，这样会增加游戏开发出来的包的大小，所以可以制作一个方向的图片，然后通过精灵的旋转方法来将图片在各个方向进行旋转。这里我们只实现了 90° 的倍数旋转，分别是我们在 `Sprite` 类中定义的常量：`TRANS_NONE`、`TRANS_ROT90`、`TRANS_ROT180`、`TRANS_ROT270`、`TRANS_MIRROR`、`TRANS_MIRROR_ROT90`、`TRANS_MIRROR_ROT180`、`TRANS_MIRROR_ROT270`。我们可以通过 `setTransform` 方法来传输上面这些常量，设置 `Sprite` 的旋转和镜像。当然，可以查看该方法的具体实现来更深入地理解 `Sprite` 的旋转和镜像。

5. 主角对话

《魔塔》中的主角可能和 NPC 对话来获得一些信息，进行游戏。《魔塔》中勇士和仙子的对话如图 11-6 所示。



图 11-6 勇士和仙子的对话

从图 11-6 中可以看出，它是通过一个浮动的对话框来显示对话内容，这个对话框只是一个矩形框，然后在右边绘制出对应的 NPC 的头像即可。这里的对话内容可以通过前面介绍的 TextUtil 类来实现自动换行。如代码清单 11-6 所示，即可实现一个游戏中的对话框的效果。

代码清单 11-6 dialog 方法

```
public void dialog()
{
    int x, y, w, h;
    w = yarin.SCREENW;
    h = yarin.MessageBoxH;
    x = 0;
    y = (yarin.SCREENH - yarin.MessageBoxH) / 2;
    if (task.curTask2 % 2 == 0)
    {
        drawDialogBox(IMAGE_DIALOG_HERO, x, y, w, h);
    }
    else
    {
        drawDialogBox(curDialogImg, x, y, w, h);
    }
    tu.DrawText(mcanvas);
}

private void drawDialogBox(int imgType, int x, int y, int w, int h)
{
    Paint ptmPaint = new Paint();
```

```

ptmPaint.setARGB(255,Color.red(BACK_COLOR), Color.green(BACK_COLOR), Color.
blue(BACK_COLOR));
yarin.fillRect(mcanvas, x, y, w, h, ptmPaint);
Bitmap img = getImage(imgType);
yarin.drawRect(mcanvas, x, y, w, h, ptmPaint);
if (img != null)
{
    if (imgType == IMAGE_DIALOG_HERO)
    {
        yarin.drawImage(mcanvas, img, x, y - 64);
    }
    else
    {
        yarin.drawImage(mcanvas, img, yarin.SCREENW - 40, y - 64);
    }
}
ptmPaint = null;
}

```

6. 战斗界面

当主角和怪物发生碰撞时,就会发生战斗,这时需要一个界面来显示战斗效果,首先看看《魔塔》中的战斗界面的效果,如图 11-7 所示。

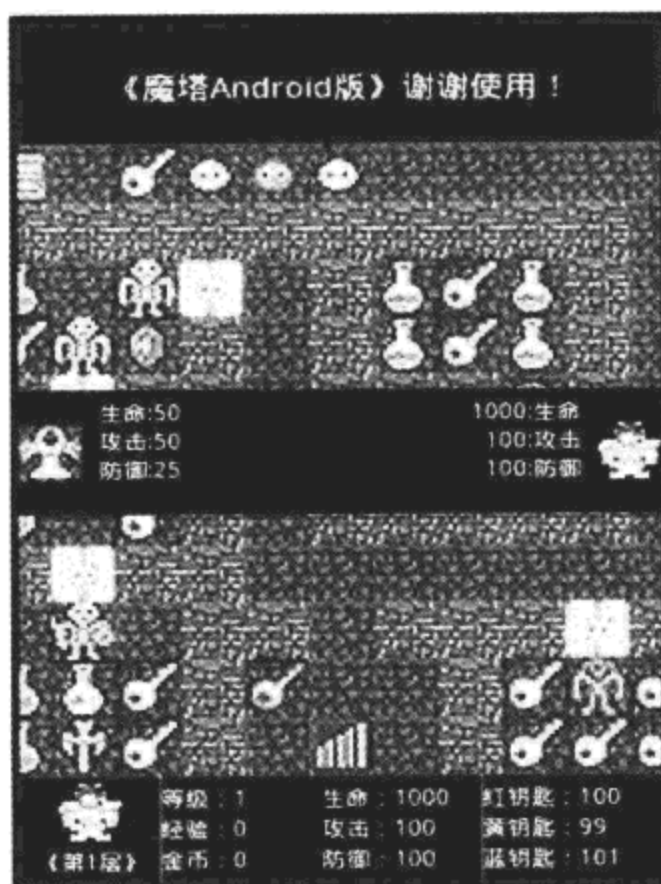


图 11-7 战斗界面

在这里,战斗界面很简单,就是分别显示玩家和怪物的头像以及属性,包括生命、攻击、防御。下面我们来看看战斗界面的绘制,如代码清单 11-7 所示。

代码清单 11-7 第 11 章\MagicTower\src\com\yarin\android\MagicTower\FightScreen.java 片段

```

protected void onDraw(Canvas canvas)
{
    mcanvas = canvas;
    int tx, ty, tw, th;
    tw = yarin.SCREENW;

```

```

        th = yarin.MessageBoxH;
        tx = 0;
        ty = (yarin.SCREENH - yarin.MessageBoxH) / 2;
        showMessage();
        if (!isFighting)
        {
            tu.DrawText(mcanvas);
        }
        else
        {
            yarin.drawImage(canvas, orgeImage, 0, ty + (th - GameMap.TILE_
            WIDTH)/2, GameMap.TILE_WIDTH, GameMap.TILE_WIDTH, orgeSrcX, orgeSrcY);
            yarin.drawImage(canvas, heroImage, (tw - GameMap.TILE_WIDTH), ty +
            (th - GameMap.TILE_WIDTH) / 2, GameMap.TILE_WIDTH, GameMap.TILE_
            WIDTH, 0, 0);
            paint.setColor(Color.WHITE);
            // 怪物
            {
                tx = 40;
                ty = (yarin.SCREENH - yarin.MessageBoxH) / 2 + 5;
                yarin.drawString(canvas, "生命:" + orgeHp, tx, ty, paint);
                yarin.drawString(canvas, "攻击:" + orgeAttack, tx, ty +
                yarin.TextSize, paint);
                yarin.drawString(canvas, "防御:" + orgeDefend, tx, ty + 2
                * yarin.TextSize, paint);
            }
            // 英雄
            {
                String string = "";
                ty = (yarin.SCREENH - yarin.MessageBoxH) / 2 + 5;
                string = hero.getHp() + ":生命";
                yarin.drawString(canvas, string, (tw-40-paint.measureText
                (string)), ty, paint);
                string = hero.getAttack() + ":攻击";
                yarin.drawString(canvas, string, (tw - 40 - paint.
                measureText(string)), ty + yarin.TextSize, paint);
                string = hero.getDefend() + ":防御";
                yarin.drawString(canvas, string, (tw-40-paint.measureText
                (string)), ty + 2 * yarin.TextSize, paint);
            }
        }

        tick();
    }
    //绘制一个矩形框
    public void showMessage()
    {
        int x = 0;
        int y = (yarin.SCREENH - yarin.MessageBoxH) / 2;
        int w = yarin.SCREENW;
        int h = yarin.MessageBoxH;
        Paint ptmPaint = new Paint();
        ptmPaint.setARGB(255, 0, 0, 0);

        yarin.fillRect(mcanvas, x, y, w, h, ptmPaint);

        ptmPaint = null;
    }
}

```

7. Sprite 绘制

上面设置了精灵的一些属性，要将 Sprite 显示在屏幕上同样很简单，下面我们看看 Sprite 类中是如何将图片绘制到屏幕上的，具体实现如代码清单 11-8 所示。

代码清单 11-8 Sprite.java 片段

```
public final void paint(Canvas canvas)
{
    if (canvas == null)
    {
        throw new NullPointerException();
    }
    //是否显示
    if (visible)
    {
        drawImage(canvas, this.x, this.y, sourceImage,
            frameCoordsX[frameSequence[sequenceIndex]],
            frameCoordsY[frameSequence[sequenceIndex]],
            srcFrameWidth,
            srcFrameHeight);
    }
}
```

代码清单 11-8 中通过 visible 来检测是否需要显示当前精灵，如果需要显示我们才绘制，绘制时只需要指定要绘制精灵的位置、精灵的图片、当前帧在图片上的偏移量、帧的宽度和高度。

本节介绍了 Sprite 的使用，以及如何在 Android 平台实现 Sprite 类。该类中还有很多方法来帮助我们开发，大家可以参考本书所附源代码：第 11 章\MagicTower\src\javax\microedition\lcdui\game\Sprite.java。

11.5 图层管理器

前面学习了使用 TiledLayer 来绘制地图，使用 Sprite 来绘制主角，可以分别使用它们的 paint 方法来显示地图和精灵，但是我们是否发现 TiledLayer 和 Sprite 类都继承自一个抽象类 Layer（图层）呢？也就是说，不管是地图还是精灵，都包含在图层这个类中，所以为了方便管理和维护这些图层，我们构建一个专门用来管理图层的图层管理器 LayerManager。

抽象类 Layer 的实现很简单，只是包括了图层的位置（x, y）、图层的宽度和高度（width, height）以及一个控制是否显示图层的布尔变量 visible。下面我们来看看 Layer 类的实现，如代码清单 11-9 所示。

代码清单 11-9 第 11 章\MagicTower\src\com\yarin\android\MagicTower\Layer.java

```
public abstract class Layer {
    //位置和宽度
    int x; // = 0;
    int y; // = 0;
    int width; // = 0;
    int height; // = 0;
    //是否显示
    boolean visible = true;
    //构造方法（宽度，高度）
    Layer(int width, int height) {
        setWidthImpl(width);
    }
}
```

```

        setHeightImpl(height);
    }
    //设置位置
    public void setPosition(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //移动图层
    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
    //得到 x
    public final int getX() {
        return x;
    }
    //得到 y
    public final int getY() {
        return y;
    }
    //得到宽度
    public final int getWidth() {
        return width;
    }
    //得到高度
    public final int getHeight() {
        return height;
    }
    //设置是否显示
    public void setVisible(boolean visible) {
        this.visible = visible;
    }
    //得到是否显示
    public final boolean isVisible() {
        return visible;
    }
    //绘制图层的一个抽象方法
    public abstract void paint(Canvas canvas);
    //设置宽度
    void setWidthImpl(int width) {
        if (width < 0) {
            throw new IllegalArgumentException();
        }
        this.width = width;
    }
    //设置高度
    void setHeightImpl(int height) {
        if (height < 0) {
            throw new IllegalArgumentException();
        }
        this.height = height;
    }
}

```

代码清单 11-9 中的方法都很简单，就是设置（获得）图层的一些属性，下面我们来创建一个图层管理器 **LayerManager**。首先确定图层管理器需要包括的成员变量：一个视图的 **x** 和 **y**，宽度和

高度, 一个 Layer 的数组 (用来保存所有的图层), 一个变量 (用来保存实际图层的数量)。下面我们来看看这个图层管理器的具体实现, 如代码清单 11-10 所示。

代码清单 11-10 第 11 章\MagicTower\src\com\yarin\android\MagicTower\LayerManager.java

```
public class LayerManager {
    public LayerManager() {
        setViewWindow(0, 0, Integer.MAX_VALUE, Integer.MAX_VALUE);
    }
    //添加一个图层
    public void append(Layer l) {
        removeImpl(l);
        addImpl(l, nlayers);
    }
    //在指定索引处设置图层
    public void insert(Layer l, int index) {
        if ((index < 0) || (index > nlayers)) {
            throw new IndexOutOfBoundsException();
        }
        removeImpl(l);
        addImpl(l, index);
    }
    //得到指定索引的图层
    public Layer getLayerAt(int index) {
        if ((index < 0) || (index >= nlayers)) {
            throw new IndexOutOfBoundsException();
        }
        return component[index];
    }
    //得到图层的数量
    public int getSize() {
        return nlayers;
    }
    //删除指定的图层
    public void remove(Layer l) {
        removeImpl(l);
    }
    //在指定位置绘制所有的图层
    public void paint(Canvas canvas, int x, int y) {
        canvas.translate(x - viewX, y - viewY);
        //设置裁剪区域
        canvas.clipRect(viewX, viewY, viewX+viewWidth, viewY+viewHeight);
        for (int i = nlayers; --i >= 0;) {
            Layer comp = component[i];
            if (comp.visible) {
                comp.paint(canvas);
            }
        }
        canvas.restore();
        canvas.translate(-x + viewX, -y + viewY);
    }
    //设置视图显示的位置
    public void setViewWindow(int x, int y, int width, int height) {
        if (width < 0 || height < 0) {
            throw new IllegalArgumentException();
        }
        viewX = x;
```

```

        viewY = y;
        viewWidth = width;
        viewHeight = height;
    }
    //添加一个图层(图层, 索引)
    private void addImpl(Layer layer, int index) {
        if (nlayers == component.length) {
            Layer newcomponents[] = new Layer[nlayers + 4];
            System.arraycopy(component, 0, newcomponents, 0, nlayers);
            System.arraycopy(component, index, newcomponents, index + 1,
                nlayers - index);
            component = newcomponents;
        } else {
            System.arraycopy(component, index, component, index + 1, nlayers
                - index);
        }
        component[index] = layer;
        nlayers++;
    }
    //删除一个指定图层
    private void removeImpl(Layer l) {
        if (l == null) {
            throw new NullPointerException();
        }
        for (int i = nlayers; --i >= 0;) {
            if (component[i] == l) {
                remove(i);
            }
        }
    }
    //删除一个指定索引的图层
    private void remove(int index) {
        System.arraycopy(component, index + 1, component, index, nlayers
            - index - 1);
        component[--nlayers] = null;
    }
    //实际的图层数
    private int nlayers; // = 0;
    //这里我们考虑性能的优化, 最多设置了4层
    private Layer component[] = new Layer[4];
    //窗口视图(x,y,w,h)
    private int viewX, viewY, viewWidth, viewHeight; // = 0;
}

```

从代码清单 11-10 中可以看出, 只需要将所有图层(包括地图、主角)一起添加到图层管理器中, 然后设置视图查看时的位置及大小, 调用图层管理器的 `paint` 方法就可以绘制出图层。绘制的顺序是按添加的反顺序, 即先添加的后绘制, 大家一定要注意这一点, 以免图层被覆盖之后显示不出来。因此, 我们只需要在《魔塔》的 `GameScreen` 类中创建一个图层管理器(`LayerManager`), 将地图和主角都添加进去, 代码如下:

```

LayerManager layerManager = new LayerManager();
layerManager.append(hero); //添加主角
layerManager.append(gameMap.getFloorMap()); //添加地图

```

然后在要绘制的地方设置视窗的位置和大小, 再调用 `paint` 方法就可以绘制出所有的图层, 代码如下:


```
layerManager.setViewWindow(scrollX, scrollY, winWidth, winHeight);
layerManager.paint(canvas, borderX, borderY);
```

到这里我们基本完成了前面所说的 Game 包的内容, 简单地说就是一个图层和一个图层管理器。如果我们使用这个 Game 来开发游戏, 就可以节省很多的时间, 提高开发效率。同样, 我们的实现方式是按 J2ME 中 MIDP2.0 的 Game 包来进行的, 所以大家在移植 J2ME 的程序时可以很轻松地解决两个平台不同的绘图方式。大家还可以实现一个 MIDP2.0 中的 GameCanvas 来使移植 J2ME 的游戏更加方便。将两个甚至多个平台的优点融合到一起, 可以让我们开发出更优秀的、更受欢迎的游戏。

11.6 游戏音效

音效在游戏开发中占据重要地位, 这一点可以通过经典的 RPG 游戏《仙剑奇侠传》来说明, 该游戏之所以能够成功就在于故事情节的逼真, 让玩家很容易将自己融入其中, 随着游戏的节奏喜怒哀乐。游戏开发的最高境界就是能带动玩家的情绪, 我们设想一下, 要是《仙剑奇侠传》没有音效, 会是一个什么样的情况呢? 可能总是感觉缺少什么一样, 玩家不会如此轻易地进入游戏的情节。

开发游戏时, 人们常常忽视游戏的音效。开发者往往把主要精力花费在游戏的图像和动画等方面, 而忽视了背景音乐和声音效果。当他们意识到这一点时, 通常为时已晚。这种做法显然是不正确的, 因为好的游戏音效和音乐可以使玩家融入游戏世界, 产生共鸣。音效的作用还不仅限于此。如果没有高超的游戏音效的映衬, 再好的图像技巧也无法使游戏的表现摆脱平庸, 对玩家也没有足够的吸引力。首先我们将游戏中的音效分为如下几类: 背景音乐、剧情音乐、音效(动作的音效、使用道具音效、辅助音效)等。背景音乐一般需要一直播放, 而剧情音乐则只需要在剧情需要的时候播放, 音效则是很短小的一段, 比如挥刀的声音、怪物叫声等。下面我们为《魔塔》加入两个背景音乐, 一个是菜单背景音乐, 一个是游戏中的背景音乐。

(1) 首先准备两个符合游戏剧情的背景音乐放到 res/raw 文件夹下面。

(2) 创建一个 CMIDIPlayer 类, 控制音乐播放。

我们知道在 Android 中是通过 MediaPlayer 来播放音乐的, 所以在 CMIDIPlayer 类中需要构建一个 MediaPlayer 对象, 通过 MediaPlayer.create 来装载音乐文件, 具体实现如代码清单 11-11 所示。

代码清单 11-11 第 11 章\MagicTower\src\com\yarin\android\MagicTower\CMIDIPlayer.java

```
public class CMIDIPlayer
{
    public MediaPlayer    playerMusic;
    public MagicTower     magicTower    = null;
    public CMIDIPlayer(MagicTower magicTower)
    {
        this.magicTower = magicTower;
    }
    // 播放音乐
    public void PlayMusic(int ID)
    {
```

```
FreeMusic();
switch (ID)
{
case 1:
    //装载音乐
    playerMusic = MediaPlayer.create(magicTower, R.raw.menu);
    //设置循环
    playerMusic.setLooping(true);
    try
    {
        //准备
        playerMusic.prepare();
    }
    catch (IllegalStateException e)
    {
        e.printStackTrace();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    //开始
    playerMusic.start();
    break;
case 2:
    playerMusic = MediaPlayer.create(magicTower, R.raw.run);
    playerMusic.setLooping(true);
    try
    {
        playerMusic.prepare();
    }
    catch (IllegalStateException e)
    {
        e.printStackTrace();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    playerMusic.start();
    break;
}
// 退出释放资源
public void FreeMusic()
{
    if (playerMusic != null)
    {
        playerMusic.stop();
        playerMusic.release();
    }
}
// 停止播放
public void StopMusic()
{
    if (playerMusic != null)
    {
        playerMusic.stop();
    }
}
```

```

    }
}
}

```

从代码清单 11-11 中可以看出, 通过 `PlayMusic` 加上 ID 来确定播放什么音乐, 通过 `StopMusic` 来停止正在播放的音乐, 当程序退出时调用 `FreeMusic` 方法来释放播放音乐产生的资源。

(3) 创建“是否开启音效”界面。

游戏中我们不可能强制玩家接受要播放的音乐, 所以需要设计一个界面来供玩家选择是否开启音乐, 这个界面这里设置得很简单, 就是绘制了一段字符串, 如图 11-8 所示。



图 11-8 音效选择界面

点击了“是”(OK 键)就是需要播放音乐, 而点击“否”则直接进入游戏, 因此点击了“是”需要开启音乐, 然后进入菜单界面, 播放音乐代码如下:

```
mCMIDIPlayer.PlayMusic(1);
```

`mCMIDIPlayer` 是我们构建的一个 `CMIDIPlayer` 类的对象。在进入游戏时, 又需要播放另一首背景音乐, 和上面的代码一样, 只需要通过参数的 ID 来设置要播放的音乐。

(4) 释放资源。

当退出游戏时, 需要释放资源, 这时调用 `CMIDIPlayer` 类的 `FreeMusic` 方法来释放资源, 代码如下:

```
mCMIDIPlayer.FreeMusic();
```

当然, 在一般的游戏中, 控制音效的界面也不只是这一个, 可以在主菜单界面里设置音效, 同样还可以在游戏中的一个弹出菜单等, 来控制音效, 但是实现方式都相同, 大家可以自己将其完善, 尽可能地方使用户随时控制音乐开关。

11.7 游戏存档

手机游戏最大的优点就是不受时间和地点等因素的限制, 玩家随时随地掏出手机即可玩游戏, 但会由于一些突发的事件让玩家不得不终止游戏, 但是玩家又正玩得高兴, 不想在下次进入游戏时又重新开始, 这就体现了游戏存档的重要性。游戏存档就是将玩家当前游戏的进度等信息存储下来, 在玩家再次进入游戏时可以通过读取上次的存档来接着上次的进度继续游戏。我们知道, 在应用程序中可以通过一个变量或者其他容器来存储一个值供我们使用, 但是退出了游戏这个值也就随之消失了。所以我们需要一个能够持久地存储数据的功能。实际上, 在本书第 6 章已经介绍了如何让数据能够持久地存储, 我们在开发游戏存档时可以选择一种适合游戏的存储方式来存储当前的游戏进度。下面我们将为《魔塔》添加一个游戏存档的功能。

1. 明确需要存储的数据

既然要存储数据, 首先就需要确定游戏中哪些数据是需要存储的, 下面我们来分析《魔塔》中需要存储的数据。首先, 为了再次游戏能够顺利地装载上次的进度, 需要保存主角的一些属性(包

括位置、生命、攻击、防御等), 还需要保存当前地图的一些属性(比如行、列、当前层数), 同样还需要保存对话的相关内容, 最后需要保存游戏的整个地图数据(每一层)。不要忘了, 还有当前的音乐状态也需要存储。当然, 每个游戏需要存储的数据肯定是不一样的, 我们必须在保证存储完整性的前提下尽可能地存储少量的数据。

2. 保存数据

可以根据数据的多少和操作的难度来选择一个合适的存储方式, 这里我们选择了存储到一个文件中, 在下一次进入游戏之后通过读取这个文件的内容来装载上次的进度。存储过程是: 获取要存储的数据→将数据打包到 `Properties` 中→将 `Properties` 写入到文件中。代码清单 11-12 存储了《魔塔》游戏中的数据。

代码清单 11-12 save 方法

```
boolean save()
{
    //取得需要存储的数据
    int col = hero.getRefPixelX() / GameMap.TILE_WIDTH;
    int row = hero.getRefPixelY() / GameMap.TILE_HEIGHT;
    byte[] r1 = hero.encode();
    byte[] r2 = {(byte) gameMap.curFloorNum, (byte) gameMap.reachedHighest, (byte)
    row, (byte) col, (byte) hero.getFrame() };
    byte[] r3 = task.getTask();
    //构建 Properties 对象
    Properties properties = new Properties();
    //将要存储的数据进行打包
    properties.put("music", String.valueOf(mMainGame.mbMusic));
    properties.put("r1l", String.valueOf(r1.length));
    properties.put("r2l", String.valueOf(r2.length));
    properties.put("r3l", String.valueOf(r3.length));
    for (int i = 0; i < r1.length; i++)
    {
        properties.put("r1_" + i, String.valueOf(r1[i]));
    }
    for (int i = 0; i < r2.length; i++)
    {
        properties.put("r2_" + i, String.valueOf(r2[i]));
    }
    for (int i = 0; i < r3.length; i++)
    {
        properties.put("r3_" + i, String.valueOf(r3[i]));
    }
    //所有的地图数据
    for (int i = 0; i < GameMap.FLOOR_NUM; i++)
    {
        byte map[] = gameMap.getFloorArray(i);
        for (int j = 0; j < map.length; j++)
        {
            properties.put("map_" + i + "_" + j, String.valueOf(map[j]));
        }
    }
    //存储数据
    try
    {
        //打开一个名为 save 的文件
        FileOutputStream stream = magicTower.openFileOutput("save", Context.
```

```

        MODE_WORLD_WRITEABLE);
        //将 properties 中的内容写入到该文件流中
        properties.store(stream, "");
    }
    catch (FileNotFoundException e)
    {
        return false;
    }
    catch (IOException e)
    {
        return false;
    }
    return true;
}

```

3. 装载数据

通过上面的方式，我们将所有的数据存储到了一个名为“save”的文件中，那么再次进入游戏时，就需要读取这个文件，获取上次的游戏进度。读取文件的流程如下：打开文件→将文件流装载进 properties 中→通过 properties.get 方法得到指定标签的数据→将得到的数据赋值给应用程序中对应的变量。代码清单 11-13 是我们读取《魔塔》的存档文件 save 的方法 load。

代码清单 11-13 load 方法

```

boolean load()
{
    //构建 Properties 对象
    Properties properties = new Properties();
    try
    {
        //打开名为"save"的文件
        FileInputStream stream = magicTower.openFileInput("save");
        //将文件流装载进 properties 中
        properties.load(stream);
    }
    catch (FileNotFoundException e)
    {
        return false;
    }
    catch (IOException e)
    {
        return false;
    }
    //将取得的数据赋值给指定的变量
    mMainGame.mbMusic = Byte.valueOf(properties.get("music").toString());
    byte[] r1 = new byte[Byte.valueOf(properties.get("r1l").toString())];
    byte[] r2 = new byte[Byte.valueOf(properties.get("r2l").toString())];
    byte[] r3 = new byte[Byte.valueOf(properties.get("r3l").toString())];
    for (int i = 0; i < r1.length; i++)
    {
        r1[i] = Byte.valueOf(properties.get("r1_" + i).toString());
    }
    for (int i = 0; i < r2.length; i++)
    {
        r2[i] = Byte.valueOf(properties.get("r2_" + i).toString());
    }
    for (int i = 0; i < r3.length; i++)
    {

```

```

        r3[i] = Byte.valueOf(properties.get("r3_" + i).toString());
    }
    hero.decode(r1);
    gameMap.curFloorNum = r2[0];
    gameMap.reachedHighest = r2[1];
    hero.setFrame(r2[4]);
    task.setTask(r3);
    for (int i = 0; i < GameMap.FLOOR_NUM; i++)
    {
        byte[] map = new byte[GameMap.TILE_NUM];
        for (int j = 0; j < map.length; j++)
        {
            map[j]=Byte.valueOf(properties.get("map_"+i+"_"+j).toString());
        }

        gameMap.setFloorArray(i, map);
    }
    gameMap.setMap(gameMap.curFloorNum);
    hero.setRefPixelPosition(r2[3]*GameMap.TILE_WIDTH + GameMap.TILE_WIDTH / 2,
    r2[2] * GameMap.TILE_HEIGHT + GameMap.TILE_HEIGHT / 2);
    return true;
}

```

上面两个方法分别实现了保存和装载，现在就要在需要保存游戏的地方调用 `save()` 方法，然后在需要读取的地方调用 `load()` 方法。我们可以看出，一个游戏的存档并不是很难，关键在于找到需要保存的数据，选择存储的方式。为了方便玩家，我们尽可能地每个游戏实现存档功能，并且在退出游戏时，不管玩家是否保存都将自动保存下来，以避免游戏进度的丢失，为玩家提供更好、更多、更方便的服务。

11.8 小结

本章我们通过实现一个较为典型的游戏学习了 Android 平台游戏开发的相关知识。相信通过本章的学习，大家都明白了我们在本章开始时所说的“其实游戏开发并不难”。由于在前面的章节中我们介绍了有关图形绘制和操作的一些知识，本章把重点放在游戏框架、如何实现以及游戏开发流程上。关于该游戏的具体实现大家可以参考本书所附源代码，这个游戏能够运行，但是并不是一个完整的游戏，因为我们将游戏中的道具商城系统、跳跃、查看怪物属性等功能留给大家学习之后练习。本章所讲述的内容基本上包括了游戏开发中经常使用的技术，大家在学习了本章后一定能够顺利地完成任务的小部分。当然这里需要说明一下，游戏开发重在创新，玩家需要挑战，所以在开发游戏之前大家一定要选择一个非常具有创意的题材。



第 12 章

Android OpenGL 开发基础

早期的手机游戏以贪食蛇、五子棋、黑白棋等画面简单的益智类游戏为主，这些游戏容易让人乏味。此后，技术进步催生了大量的动作、网络等类型的游戏，画面也随之变得愈发精细。如今，手机游戏更加向着高端的 3D 游戏发展。

随着全球 3G 的快速发展，手机 3D 游戏应用也在近两年成为业界关注的热点。由于技术厂商的不断创新，市场上以硬件加速 3D 图形功能为特色的手机早已不是新鲜玩意，并已经逐步具备了与专业游戏设备相媲美的画面质量。面对这一市场机遇，越来越多的游戏开发商和发行商加入到了手机 3D 游戏的队伍中，合纵连横形成了多方位的产业链合作体系，这将进一步推动手机 3D 游戏市场的发展。随着技术的不断成熟，想要在手机 3D 游戏市场上有所作为的厂商开始了多个层次的合作，以期取长补短。这是一个全新的领域，从游戏功能到操作感觉、从支撑软件到游戏内容，都需要产业链各环节的紧密合作，所以形成一个完善的产业链至关重要。游戏内容是未来手机 3D 游戏市场健康发展的关键，众多的游戏开发商和发行商也在积极寻求与底层技术厂商的合作，在技术上推陈出新并合作推广炫目的移动 3D 游戏。随着移动价值链上所有环节的领先厂商进一步加深彼此之间的合作，3D 游戏必将获得更快的发展。在 Android 系统中提供了 `android.opengl` 包，专门用于 3D 的加速和渲染等，本章我们将学习 OpenGL 的基础知识。

12.1 OpenGL 简介

OpenGL (Open Graphics Library) 定义了一个跨编程语言、跨平台的编程接口的规格，是一个性能卓越的三维图形标准。OpenGL 是一个专业的图形程序接口，是一个功能强大、调用方便的底层图形库。OpenGL 的前身是 SGI 公司为其图形工作站开发的 IRIS GL。IRIS GL 是一个工业标准的 3D 图形软件接口，功能虽然强大但是移植性不好，于是 SGI 公司便在 IRIS GL 的基础上开发了 OpenGL。虽然 DirectX 在家用市场全面领先，但在专业高端绘图领域，OpenGL 是不二的主角。

1. OpenGL 的发展历程

1992 年 7 月，SGI 公司发布了 OpenGL 1.0 版本，随后又与微软公司共同开发了 Windows NT 版本的 OpenGL，从而使一些原来必须在高档图形工作站上运行的大型 3D 图形处理软件也可以在微机上运用。

1995 年，OpenGL 1.1 版本面世，该版本较 1.0 性能提高了许多，并加入了一些新的功能（包括提高顶点位置、法线、颜色、色彩指数、纹理坐标、多边形边缘标识的传输速度），引入了新的纹理特性等。

1997 年，Windows 95 下的 3D 游戏大量涌现，游戏开发公司迫切需要一个功能强大、兼容性好

的 3D 图形接口,而当时微软公司自己的 3D 图形接口 DirectX 3.0 的功能却很糟糕。因而,以制作《雷神之锤》等经典 3D 射击游戏而著名的公司同其他一些游戏开发公司一同强烈要求微软在 Windows 95 中加入对 OpenGL 的支持。微软公司最终在 Windows 95 的 OSR2 版和后来的 Windows 版本中加入了对 OpenGL 的支持。这样,不但许多支持 OpenGL 的电脑 3D 游戏得到广泛应用,而且许多 3D 图形设计软件也可以采用支持 OpenGL 标准的 3D 加速卡,大大提高了其 3D 图形的处理速度。

2003 年 7 月 28 日,SGI 和 ARB 公布了 OpenGL 1.5。OpenGL 1.5 中包括 OpenGL ARB 的正式扩展规格绘制语言“OpenGL Shading Language”。OpenGL 1.5 的新功能包括顶点 Buffer Object、Shadow 功能、隐蔽查询、非乘方纹理等。

2004 年 8 月,OpenGL 2.0 版本发布。OpenGL 2.0 标准的主要制定者并非原来的 SGI,而是逐渐在 ARB 中占据主动地位的 3Dlabs。OpenGL 2.0 支持 OpenGL Shading Language、新的 shader 扩展特性以及其他多项增强特性。

2008 年 8 月初,Khronos 工作组在 Siggraph 2008 大会上宣布了 OpenGL 3.0 图形接口规范,GLSL 1.30 shader 语言和其他新增功能将再次为未来 3D 接口的发展指明方向。

OpenGL 3.0 API 的开发代号为 Longs Peak,和以往一样,OpenGL 3.0 仍然作为一个开放性和跨平台的 3D 图形接口标准,在 shader 语言盛行的今天,OpenGL 3.0 增加了新版本的 shader 语言:GLSL 1.30,可以充分发挥当前可编程图形硬件的潜能。同时,OpenGL 3.0 还引入了一些新的功能,例如顶点矩阵对象、全帧缓存对象功能、32 位浮点纹理和渲染缓存、基于阻塞队列的条件渲染、紧凑行半浮点顶点和像素数据、四个新压缩机制等。

2009 年 3 月,又公布了升级版新规范 OpenGL 3.1,也是这套跨平台免费 API 有史以来的第 9 次更新。OpenGL 3.1 将此前引入的 OpenGL 着色语言“GLSL”从 1.30 版升级到了 1.40 版,通过改进程序增强了对最新可编程图形硬件的访问,还有更高效的顶点处理、扩展的纹理功能、更弹性的缓冲管理等。宽泛地讲,OpenGL 3.1 在 3.0 版的基础上对整个 API 模型体系进行了简化,可大幅提高软件开发效率。

2. OpenGL 与 OpenGL ES 的区别

笔者在编写本书时,发现网上有很多人分不清楚 OpenGL 与 OpenGL ES 之间的关系,因此这里对它们进行详细的介绍。OpenGL ES 是专为内嵌和移动设备设计的一个 2D/3D 轻量图形库,它是基于 OpenGL API 设计的,是 OpenGL 三维图形 API 的子集,针对手机、PDA 和游戏主机等嵌入式设备而设计。该 API 由 Khronos 集团定义、推广,Khronos 是一个图形软硬件行业协会,该协会主要关注图形和多媒体方面的开放标准。OpenGL ES 是从 OpenGL 裁剪定制而来的,去除了 glBegin/glEnd、四边形(GL_QUADS)、多边形(GL_POLYGONS)等许多非绝对必要的特性。经过多年的发展,OpenGL ES 现在主要有两个版本:OpenGL ES 1.x 针对固定管线硬件,OpenGL ES 2.x 针对可编程管线硬件。OpenGL ES 1.0 是以 OpenGL 1.3 规范为基础的,而 OpenGL ES 1.1 是以 OpenGL 1.5 规范为基础的,它们又分别支持 common 和 common lite 两种 profile。lite profile 只支持定点实数,而 common profile 既支持定点数又支持浮点数。OpenGL ES 2.0 则是参照 OpenGL 2.0 规范定义的。下面列举了一些 OpenGL ES 相对于 OpenGL 删减的功能供大家参考。

(1) glBegin/glEnd。

(2) glVertexElement。

- (3) 显示列表。
- (4) 求值器。
- (5) 索引色模式。
- (6) 自定义裁剪平面。
- (7) `glRect`。
- (8) 图像处理 (这个一般显卡没有, FireGL/Quadro 显卡有)。
- (9) 反馈缓冲。
- (10) 选择缓冲。
- (11) 累积缓冲。
- (12) 边界标志。
- (13) `glPolygonMode`。
- (14) `GL_QUADS`、`GL_QUAD_STRIP`、`GL_POLYGON`。
- (15) `glPushAttrib`、`glPopAttrib`、`glPushClientAttrib`、`glPopClientAttrib`。
- (16) `TEXTURE_1D`、`TEXTURE_3D`、`TEXTURE_RECT`、`TEXTURE_CUBE_MAP`。
- (17) `GL_COMBINE`。
- (18) 自动纹理坐标生成。
- (19) 纹理边界。
- (20) `GL_CLAMP`、`GL_CLAMP_TO_BORDER`。
- (21) 消失纹理代表。
- (22) 纹理 LOD 限定。
- (23) 纹理偏好限定。
- (24) 纹理自动压缩、解压缩。
- (25) `glDrawPixels`、`glPixelTransfer`、`glPixelZoom`。
- (26) `glReadBuffer`、`glDrawBuffer`、`glCopyPixels`。

3. Android OpenGL ES

目前 Android SDK 1.5 只支持 OpenGL ES 1.0 和 OpenGL ES 1.1 的大部分功能, 对于 OpenGL ES 2.0 的支持我们都很期待。但是目前 Android 平台上已经也有不少的 3D 游戏了, 如图 12-1 所示是 Omnigsoft 公司的几款 3D 游戏, 分别是: Super-G Stunt、City Stage、Snow Rally Canada、Nine Hole Golf。

首先, 我们来学习如何在 Android 构建一个 3D 开发的基本框架, 这里直接在工程中导入如下库:

```
import javax.microedition.khronos.opengles.GL10;
```

其次, 直接实例化 `OpenGLContext` 的对象 `mOpenGLContext` 就创建了一个 3D 的支持接口, 通过创建 `OpenGLContext` 对象, 直接实例化 `GL10`, 这里 10 代表 1.0 版本。代码如下:

```
GL10 gl = (GL10) (mOpenGLContext.getGL());
```

再次, 在 `onDraw` 中处理 `canvas` 需要在开始和结束分别调用 `canvas` 绘制的 `mOpenGL Context`。

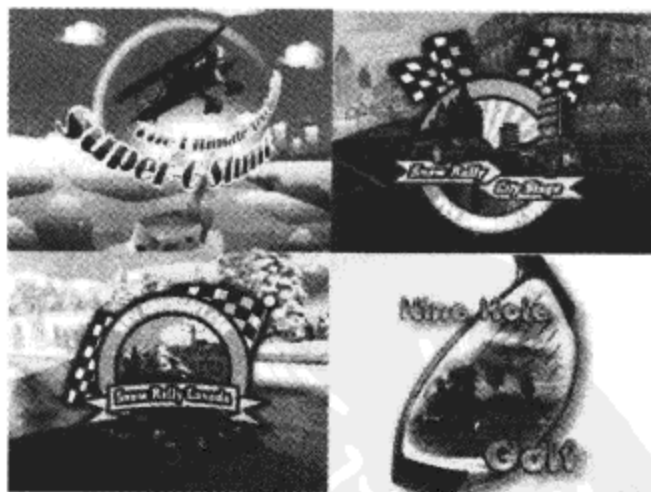


图 12-1 Android 上的几款 3D 游戏

waitNative 和 waitGL()作为开始和结束。

通过上面的方法我们可以操作 3D 接口,但是 Android 为我们提供了 GLSurfaceView 来更好地显示 OpenGL 视图,而 GLSurfaceView 中则包含了一个专门用于渲染 3D 的接口 Renderer。所以这里我们先来构建一个自己的 Renderer 类,首先需要引入如下接口:

```
import android.opengl.GLSurfaceView.Renderer;
```

然后创建一个 GLRender 类实现 Renderer 接口,代码如下:

```
public class GLRender implements Renderer
{
}
```

在 GLRender 中我们必须实现下面的 3 个抽象方法:

```
public void onDrawFrame(GL10 gl){}
public void onSurfaceChanged(GL10 gl, int width, int height){}
public void onSurfaceCreated(GL10 gl, EGLConfig config){}
```

当窗口被创建时需要调用 onSurfaceCreated,所以我们可以里面对 OpenGL 做一些初始化工作,例如,如下代码:

```
// 告诉系统对透视进行修正
gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
// 黑色背景
gl.glClearColor(0, 0, 0, 0);
// 启用阴影平滑
gl.glShadeModel(GL10.GL_SMOOTH);
```

其中,glHint 用于告诉 OpenGL 我们希望进行最好的透视修正,这会轻微地影响性能,但会使得透视图看起来好一点。glClearColor 设置清除屏幕时所用的颜色,色彩值的范围从 0.0f~1.0f,0.0f 代表最黑的情况,1.0f 就是最亮的情况。glClearColor 后的第一个参数是 Red Intensity (红色分量),第二个参数是绿色分量,第三个参数是蓝色分量,最大值也是 1.0f,代表特定颜色分量的最亮情况。最后一个参数是 Alpha 值。通过混合三种原色(红、绿、蓝),可以得到不同的色彩。因此,当使用 glClearColor(0.0f,0.0f,1.0f,0.0f)时,将用亮蓝色来清除屏幕。如果用 glClearColor(0.5f,0.0f,0.0f,0.0f)的话,将使用中红色来清除屏幕。要得到白色背景,应该将所有的颜色设成最亮(1.0f)。要得到黑色背景,应该将所有的颜色设为最暗(0.0f)。glShadeModel 用于启用 smooth shading (阴影平滑)。阴影平滑通过多边形精细地混合色彩,并对外部光进行平滑。

最后我们还需要做的一个最重要的步骤是关于 depth buffer (深度缓存)的。将深度缓存设想为屏幕后面的层,它不断地对物体进入屏幕内部的深度进行跟踪。本节的程序其实没有真正使用深度缓存,但几乎所有在屏幕上显示 3D 场景的 OpenGL 程序都使用了深度缓存,它的排序决定哪个物体先画。这样你就不会将一个圆形后面的正方形画到圆形上来。深度缓存是 OpenGL 十分重要的部分,代码如下:

```
gl.glClearDepthf(1.0f);           // 设置深度缓存
gl.glEnable(GL10.GL_DEPTH_TEST);  // 启用深度测试
gl.glDepthFunc(GL10.GL_LEQUAL);   // 所做深度测试的类型
```

当窗口的大小发生改变时调用 onSurfaceChanged 方法,当然,不管窗口的大小是否已经改变。它在程序开始时至少运行一次,所以我们在该方法中设置 OpenGL 场景的大小,这里将 OpenGL 场景设置成它显示时所在窗口的大小,代码如下:

```
//设置 OpenGL 场景的大小
gl.glViewport(0, 0, width, height);
```

下面我们将为屏幕设置透视图，这意味着越远的东西看起来越小。这么做创建了一个显示外观的场景。`gl.glMatrixMode(GL10.GL_PROJECTION)`指明接下来的代码将影响 `projection matrix`（投影矩阵），投影矩阵负责为场景增加透视。`gl.glLoadIdentity()`近似于重置，它将所选的矩阵状态恢复成原始状态，调用 `gl.glLoadIdentity()`之后为场景设置透视图。`gl.glMatrixMode(GL10.GL_MODELVIEW)`指明任何新的变换将会影响 `modelview matrix`（模型观察矩阵）。模型观察矩阵中存放了物体信息。最后重置模型观察矩阵，代码如下：

```
gl.glMatrixMode(GL10.GL_PROJECTION);           //设置投影矩阵
gl.glLoadIdentity();                           //重置投影矩阵
gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);     //设置视口的大小
gl.glMatrixMode(GL10.GL_MODELVIEW);
gl.glLoadIdentity();
```

这里需要说明一下 `glFrustumf` 方法，前面 4 个参数用于确定窗口的大小，而后面两个参数分别是在场景中所能绘制深度的起点和终点。

最后所有的绘图操作都在 `onDrawFrame` 方法中进行，在绘图之前，需要将屏幕清除成前面所指定的颜色，清除深度缓存并且重置场景，然后就可以进行绘图了。代码如下：

```
// 清除屏幕和深度缓存
gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
gl.glLoadIdentity();
```

最后只需要调用 `GLSurfaceView` 类的 `setRenderer` 方法将我们自己构建的 `GLRender` 类设置为默认的 `Renderer`，通过 `setContentView` 方法使 `Activity` 显示一个 `GLSurfaceView` 即可，代码如下：

```
Renderer render = new GLRender();
GLSurfaceView glView = new GLSurfaceView(this);
glView.setRenderer(render);
setContentView(glView);
```

到这里，我们基本完成了 `Android` 平台 `OpenGL` 开发的基本框架。从下一节开始，我们将在 `Android` 中使用 `OpenGL` 来进行 3D 开发。本节所讲的内容都是最基本的，但同时也是最重要的，后面的章节我们将按照这个框架来进行开发，希望大家好好掌握。

12.2 多边形

在 `OpenGL` 中绘制的任何模型都会被分解为三角形和四边形这两种简单的图形，因此我们首先来学习如何使用 `OpenGL` 来绘制一个三角形和一个四边形。有了上一节的框架，现在要绘制三角形和四边形就很简单了，只需要在 `GLRender` 类的 `onDrawFrame` 方法中添加绘制代码即可。但是在绘制之前，我们先要了解一下 `OpenGL` 中的坐标系。简单地说，当调用 `glLoadIdentity()`之后，实际上是将当前点移到了屏幕中心，`X` 坐标轴从左至右，`Y` 坐标轴从下至上，`Z` 坐标轴从里至外。`OpenGL` 屏幕中心的坐标值是 `X` 和 `Y` 轴上的 `0.0f` 点。中心左边的坐标值是负值，右边是正值。移向屏幕顶端是正值，移向屏幕底端是负值。移入屏幕深处是负值，移出屏幕则是正值。

我们都知道三角形由 3 个顶点组成，所以需要先定义出三角形和四边形的顶点坐标。这里三角形不再处于 2D 平面中，而是在 3D 空间中，所以每一个坐标点由 `(X,Y,Z)` 组成，例如定义三角形

的顶点数组如下:

```
int one = 0x10000;
//三角形三个顶点
private IntBuffer triggerBuffer = IntBuffer.wrap(new int[]{
    0, one, 0,           //上顶点
    -one, -one, 0,       //左下点
    one, -one, 0,        //右下点
});
```

四边形的顶点数组当然由 4 个顶点组成, 这里需要在左边绘制一个三角形, 在右边绘制一个四边形, 所以需要使用 `glTranslatef(X, Y, Z)` 将当前的中点移至要绘制三角形的位置, 例如如下代码将沿着 X 轴左移 1.5 个单位, Y 轴不动(0.0f), 最后移入屏幕 6.0f 个单位。

```
gl.glTranslatef(-1.5f, 0.0f, -6.0f);
```

现在我们已经移到了屏幕的左半部分, 并且将视图推入屏幕背后足够的距离以便可以看见全部的场景, 这里需要注意的是屏幕内移动的单位数必须小于前面我们通过 `glFrustumf` 方法所设置的最远距离, 否则显示不出来。我们要为 OpenGL 设置一个顶点数组, 所以需要告诉 OpenGL 要设置顶点这个功能, 通过如下代码可以开启顶点设置功能:

```
gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
```

要绘制三角形, 还需要告诉 OpenGL 所要绘制的三角形的顶点数据。可以通过 `glVertexPointer(int size, int type, int stride, Buffer pointer)` 方法来设置顶点数据, 该方法中的参数 `size` 用于描述顶点的尺寸, 本例中使用的是 X,Y,Z 坐标系, 所以是 3; `type` 描述了顶点的类型, 本例中的数据是固定的, 所以使用了 `GL_FIXED` 表示固定的顶点; `stride` 描述了步长; `pointer` 则是顶点缓存, 即我们创建的顶点数组。如下代码是设置三角形的顶点数组。

```
gl.glVertexPointer(3, GL10.GL_FIXED, 0, triggerBuffer);
```

设置好了顶点缓冲, 我们就可以通过 `glDrawArrays(int mode, int first, int count)` 方法将这些顶点绘制出来, 参数 `mode` 描述了绘制的模式, 我们使用 `GL_TRIANGLES` 来表示绘制三角形, 后面两个参数分别是开始位置和要绘制的顶点计数。如下是我们绘制一个三角形的方法:

```
gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
```

绘制完三角形之后要绘制四边形, 由于在绘制三角形时我们将中点移动到了屏幕左边, 所以需要先重置矩阵, 将中点移动到三角形的右边, 然后设置顶点, 完成绘制。完整的绘制如代码清单 12-1 所示。

代码清单 12-1 onDrawFrame 方法

```
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // 重置当前的模型观察矩阵
    gl.glLoadIdentity();
    // 左移 1.5 单位, 并移入屏幕 6.0
    gl.glTranslatef(-1.5f, 0.0f, -6.0f);
    // 允许设置顶点
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    // 设置三角形
    gl.glVertexPointer(3, GL10.GL_FIXED, 0, triggerBuffer);
    // 绘制三角形
    gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    // 重置当前的模型观察矩阵
```

```

gl.glLoadIdentity();
// 左移 1.5 单位, 并移入屏幕 6.0
gl.glTranslatef(1.5f, 0.0f, -6.0f);
// 设置和绘制正方形
gl.glVertexPointer(3, GL10.GL_FIXED, 0, quaterBuffer);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
// 取消顶点设置
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
}

```

注意 我们通过 `glEnableClientState` 方法开启了顶点设置功能, 在使用完成之后, 需要通过 `gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);` 方法关闭 (取消) 顶点设置功能。

最后, 图 12-2 是我们所绘制的三角形和四边形的效果图。具体实现参见本书所附代码: 第 12 章\Examples_12_01。



图 12-2 OpenGL 绘制的多边形

通过本节的学习我们了解了 OpenGL 绘图的流程、坐标以及如何在 OpenGL 中绘制一些简单的几何图形, 大家可以试着更改坐标的数据和 `glTranslatef` 方法移动中心的位置, 以便更深入地了解 OpenGL 的工作窗口。

12.3 颜色

通过上一节的学习我们可以使用 OpenGL 通过顶点数据绘制出一些图形, 但是可以看出我们绘制的图形都是白色的, 这是因为没有对图形进行色彩渲染。本节我们将学习如何对图形进行着色, 并学习两种不同的着色方式, 分别是: 光滑着色和平面着色。先来看看经过着色之后的图形, 如图 12-3 所示。

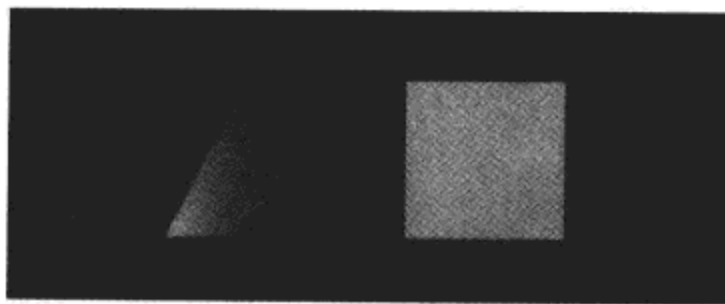


图 12-3 经过着色的多边形

如图 12-3 所示的三角形使用了 **Smooth coloring** (平滑着色) 将三角形的三个顶点的不同颜色混合在一起, 创建出漂亮的色彩混合。而四边形则使用了 **Flat coloring** (单调着色) 涂上一种固定

的颜色。

首先我们学习 Smooth coloring (平滑着色)。先为三角形的每个顶点设置一个颜色, 这个颜色和我们前面所说的清屏颜色一样, 如下代码是为三角形定义的颜色数组:

```
int one = 0x10000;
//三角形的顶点颜色值(r,g,b,a)
private IntBuffer colorBuffer = IntBuffer.wrap(new int[]{
    one, 0, 0, one,
    0, one, 0, one,
    0, 0, one, one, });
```

同样, 需要使用如下代码开启颜色渲染功能:

```
gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
```

Android 中提供了 glColorPointer 方法来设置颜色, 其中参数类型和 glVertexPointer 类似, 下面是为三角形设置定义好的颜色:

```
gl.glColorPointer(4, GL10.GL_FIXED, 0, colorBuffer);
```

最后, 绘制方法和前面一样。需要注意的是, 使用完颜色之后不要忘记关闭, 在 OpenGL 中使用 glEnableClientState 之后都需要 glDisableClientState 来关闭或取消对应的功能, 代码如下:

```
gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
```

在学会了平滑着色之后, 单调着色就简单了, 单调着色实际上是设置当前所使用的颜色, 即设置当前颜色之后绘制的所有内容都使用当前颜色。即使是在完全采用纹理贴图的时候, 仍然可以用来调节纹理的色调 (纹理我们在后面会讲解)。

现在我们只需将颜色一次性地设为想采用的颜色(本例采用蓝色), 然后绘制场景。这时绘制的每个顶点都是蓝色的, 因为我们没有告诉 OpenGL 要改变顶点的颜色。这样就可以绘制出一个蓝色的正方形, 设置当前颜色代码如下:

```
//颜色(r,g,b,a)
gl.glColor4f(0.5f, 0.5f, 1.0f, 1.0f);
```

本节内容很简单, 绘制方式和上一节一样, 只是在绘制之前设置了绘制要使用的颜色。需要注意的是, 这里颜色的取值范围都是 0~1, 可能大家刚接触这种方式, 有点不适应, 建议多更改一下颜色的数值, 然后观察运行的效果, 慢慢便熟悉了。具体实现参见本书所附代码: 第 12 章\Examples_12_02。

12.4 旋转

上一节中我们给三角形和四边形进行了着色, 这一节将学习如何将彩色对象绕着坐标轴旋转。本节中我们将三角形沿 Y 轴旋转, 四边形沿 X 轴旋转。运行效果如图 12-4 所示, 具体实现参见本书所附代码: 第 12 章\Examples_12_03。



图 12-4 多边形旋转

要实现旋转很简单，首先增加两个变量来控制这两个对象的旋转。它们是浮点类型的变量，使得我们能够非常精确地旋转对象。浮点数包含小数位置，这意味着无需使用 1、2、3... 的角度。浮点数是 OpenGL 编程的基础。新变量中的 rotateTri 用来旋转三角形，rotateQuad 用来旋转四边形，代码如下：

```
float          rotateTri; // 用于三角形的角度
float          rotateQuad; // 用于四边形的角度
```

下面就是本节内容的核心部分了——旋转函数。在 Android 中用 glRotatef (float angle, float X, float Y, float Z) 方法将某个物体沿指定的轴旋转，其中参数 angle 通常是一个变量，代表对象转过的角度。X、Y 和 Z 三个参数则共同决定旋转轴的方向。比如 (1,0,0) 所描述的矢量经过 X 坐标轴的 1 个单位处并且方向向右，(-1,0,0) 所描述的矢量经过 X 坐标轴的 1 个单位处但方向向左。下面是用于旋转的三角形和四边形的代码，我们可以看出三角形沿 Y 轴旋转 rotateTri 角度，四边形沿 X 轴旋转 rotateQuad 角度。

```
//旋转三角形
gl.glRotatef(rotateTri, 0.0f, 1.0f, 0.0f);
//旋转四边形
gl.glRotatef(rotateQuad, 1.0f, 0.0f, 0.0f);
```

其实到这里我们已经完成了对物体的旋转，为了让物体能够不停地旋转，在 onDrawFrame 方法中不断改变角度的变量，如下代码分别是改变三角形和四边形的旋转角度。

```
//改变旋转的角度
rotateTri += 0.5f;
rotateQuad -= 0.5f;
```

和前面一样，大家可以试着更改旋转的轴和数值来观察旋转的效果，这可以更好地理解 3D 的空间坐标轴。这也是大家刚学习 3D 时的难点。

12.5 3D 空间

前面几节我们所学习的都是 3D 中的 2D 物体，本节将开始接触真正的 3D 物体。本节我们将给三角形增加一个左侧面、一个右侧面和一个后侧面来生成一个金字塔（四棱锥），给正方形增加左、右、上、下及背面生成一个立方体。我们混合金字塔上的颜色，创建一个平滑着色的对象；给立方体的每一面加上不同的颜色。运行效果如图 12-5 所示，具体实现参见本书所附代码：第 12 章 \Examples_12_04。

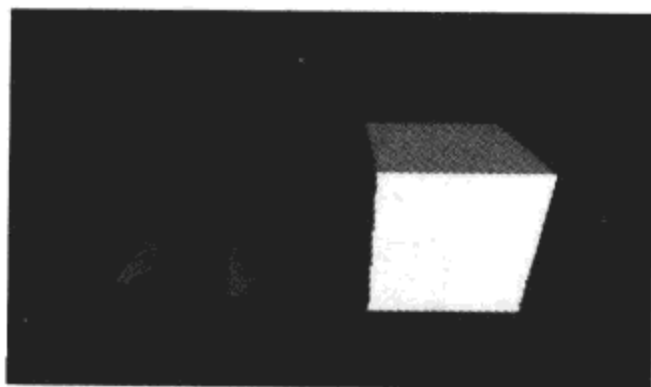


图 12-5 3D 空间

要绘制图 12-5 的效果很简单，基本上和绘制三角形一样，重点在于构建四棱锥和立方体的顶点坐标，这里需要注意：在构建这些顶点坐标时，要让对象绕自身的轴旋转，必须让对象的中心坐标总是 (0.0f,0.0f,0.0f)，并且三角形都是按逆时针次序绘制的。所以在构建顶点坐标时，需要按照逆时针的顺序来绘制，千万不能既有逆时针的方向又有顺时针的方向。下面是我们构建的金字塔，金字塔的上顶点高出原点一个单位，底面中心低于原点一个单位。上顶点在底面的投影位于底面的中心。如代码清单 12-2 所示。

代码清单 12-2 triggerBuffer

```
int one = 0x10000;
private IntBuffer triggerBuffer = IntBuffer.wrap(new int[]{
    0, one, 0,
    -one, -one, 0,
    one, -one, one,

    0, one, 0,
    one, -one, one,
    one, -one, -one,

    0, one, 0,
    one, -one, -one,
    -one, -one, -one,

    0, one, 0,
    -one, -one, -one,
    -one, -one, one
});
```

由于所有的面都共享上顶点，所以将这个点在所有三角形中都设置为红色。底边上的两个顶点的颜色则是互斥的。前侧面的左下顶点是绿色的，右下顶点是蓝色的。这样相邻右侧面的左下顶点是蓝色的，右下顶点是绿色的。所以，四棱锥的顶点颜色定义如代码清单 12-3 所示。

代码清单 12-3 colorBuffer

```
private IntBuffer colorBuffer = IntBuffer.wrap(new int[]{
    //tri 4 face
    one, 0, 0, one,
    0, one, 0, one,
    0, 0, one, one,

    one, 0, 0, one,
    0, one, 0, one,
    0, 0, one, one,

    one, 0, 0, one,
    0, one, 0, one,
    0, 0, one, one,

    one, 0, 0, one,
    0, one, 0, one,
    0, 0, one, one,
});
```

创建好了顶点和颜色的相关数据，现在我们就开始绘制这个四棱锥。由于四棱锥不再是一个三角形面，而是由 4 个三角形面组成，所以这里需要分别绘制出 4 个三角形。如下代码是我们绘制四棱锥的代码：

```
//绘制四棱锥
for(int i=0; i<4; i++)
{
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, i*3, 3);
}
```

其他颜色和顶点的设置与三角形一样，当然绘制立方体也一样，只不过立方体是由 6 个正方形面组成的，所以需要分别绘制出 6 个正方形，代码如下：

```
//绘制立方体
for(int i=0; i<6; i++)
{
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, i*4, 4);
}
```

这样就可以绘制出逼真的 3D 物体了。刚接触 3D 时，对于 3D 的空间感非常重要，大家可以通过绘制不同形状的物体来练习，加强自己的 3D 空间感。

12.6 纹理映射

在上一节中，我们已学习了在 3D 空间创建对象的方法。必须将 OpenGL 屏幕想象成一张很大的画纸，后面还带着许多透明的层，大致就是由大量的点组成的立方体，这些点从左至右、从上至下、从前至后地布满了这个立方体。如果你能想象出屏幕的深度方向，在设计新 3D 对象时应该没有任何问题。大家设想一下，光靠一些基本的几何体和一些颜色组成的游戏，能受到玩家的喜爱吗？肯定不能，所以本节我们学习如何将纹理映射到立方体上去，运行效果如图 12-6 所示。具体实现参见本书所附代码：第 12 章\Examples_12_05。

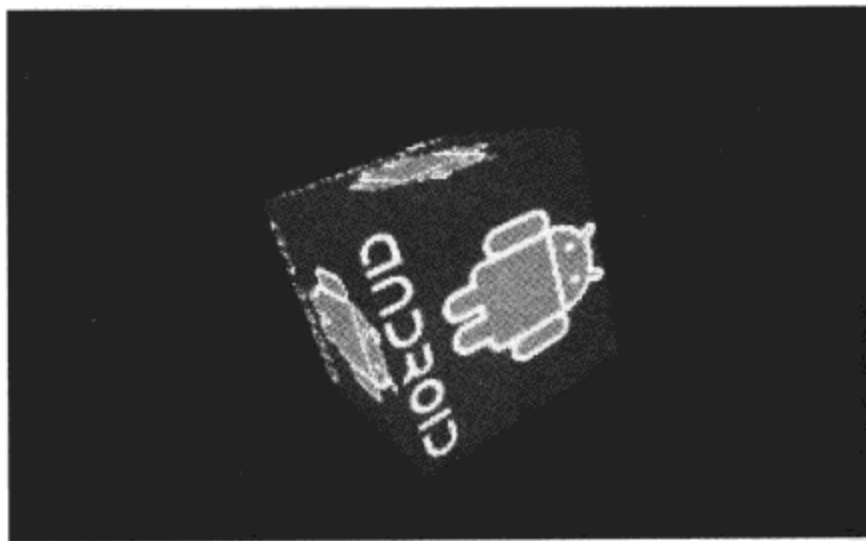


图 12-6 纹理映射

从图 12-6 可以看出，我们为立方体的每一个面都贴上了一张图片，要实现这个效果，需要创建一个纹理，并使用图片来生成一个纹理。可以通过如下代码创建一个纹理：

```
IntBuffer intBuffer = IntBuffer.allocate(1);
// 创建纹理
gl.glGenTextures(1, intBuffer);
texture = intBuffer.get();
// 设置要使用的纹理
gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
```

上面代码中的 `glGenTextures (int n, IntBuffer textures)` 用于通知 OpenGL 我们想生成一个纹理的名字。如果想载入多个纹理，可以在参数 `n` 处设置，参数 `textures` 描述了纹理的名字（代号）。`glBindTexture (int target, int texture)` 方法用于通知 OpenGL 将纹理名字 `texture` 绑定到纹理目标上。2D 纹理只有高度（在 Y 轴上）和宽度（在 X 轴上）。

下面我们将使用图片来生成一个纹理。在 Android 中我们使用 `GLUtils` 中的一个静态方法 `texImage2D (int target, int level, Bitmap bitmap, int border)` 来生成一个纹理，参数 `target` 描述了纹理的类型，参数 `level` 描述了纹理的详细程度，一般情况下设置为 0，参数 `bitmap` 是用于纹理贴图的图像数据，参数 `border` 描述了边框效果。该方法使用如下代码：

```
//生成纹理
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, GLImage.mBitmap, 0);
```

到这里，我们成功地创建了一个纹理。为了能有更好的效果，还需要设置在 OpenGL 显示图像时，它放大得比原始纹理大（`GL_TEXTURE_MAG_FILTER`）或缩小得比原始纹理小（`GL_TEXTURE_MIN_FILTER`）时 OpenGL 所采用的滤波方式。通常这两种情况下都采用 `GL_LINEAR`，这使得纹理从很远处到离屏幕很近时都平滑显示。使用 `GL_LINEAR` 需要 CPU 和显卡做更多的运算。考虑到效率问题，这里采用 `GL_NEAREST`。过滤的纹理在放大的时候，看起来有很多类似于马赛克的东西。当然也可以结合这两种滤波方式，在近处时使用 `GL_LINEAR`，在远处时使用 `GL_NEAREST`。下面是我们设置的线性滤波。

```
// 线性滤波
gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
```

在设置好纹理之后，暂时还不能绘制，还要绑定当前需要使用的纹理。在 Android 中可以使用 `glBindTexture (int target, int texture)` 方法来绑定纹理，参数 `target` 描述了要绑定的纹理的类型，参数 `texture` 描述了纹理的名字（代号），如果在场景中使用多个纹理，则应该使用 `glBindTexture` 方法来选择要绑定的纹理。当你想改变纹理时，应该绑定新的纹理。需要注意的是，绑定纹理操作必须在绘图之前。例如如下代码绑定了上面创建的纹理。

```
// 绑定纹理
gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
```

纹理的使用与颜色一样，需要使用 `glEnableClientState` 方法来开启纹理（`GL_TEXTURE_COORD_ARRAY`），使用完之后需要使用 `glDisableClientState` 来关闭，代码如下：

```
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
//.....
gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
```

为了将纹理正确地映射到四边形上，必须将纹理的右上角映射到四边形的右上角，纹理的左上角映射到四边形的左上角，纹理的右下角映射到四边形的右下角，纹理的左下角映射到四边形的左下角。如果映射错误的话，图像显示时可能上下颠倒，侧向一边或者什么都没有。下面是我们对立方体的每一个面所设置的纹理映射数据。

```
IntBuffer texCoords = IntBuffer.wrap(new int[] {
    one, 0, 0, 0, 0, one, one, one,
    0, 0, 0, one, one, one, one, 0,
    one, one, one, 0, 0, 0, 0, one,
    0, one, one, one, one, 0, 0, 0,
    0, 0, 0, one, one, one, one, 0,
```

```
one, 0, 0, 0, 0, one, one, one, });
```

设置好这些映射数据之后，可以通过 `glTexCoordPointer` 将纹理绑定到要绘制的物体上。代码如下，其中，第一个参数是纹理的坐标类型，这里使用的是 2D 纹理，只有 X 和 Y 两个坐标，所以是 2；第二个参数描述了我们所使用的是固定的纹理数据；第三个参数是步长；第四个参数是我们定义的纹理映射数据。

```
//指定纹理映射
gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, texCoords);
```

最后，和绘制多边形一样将其绘制到屏幕上即可。这里还可以使用 `glDrawElements` 方法来绘制多边形，该方法只是多了一个数据类型的参数，用来描述所使用的数据类型，如下绘制四边形所使用的是 `GL_UNSIGNED_BYTE`。

```
gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, 24, GL10.GL_UNSIGNED_BYTE, indices);
```

这样我们就轻松实现了本节开始所展示的效果，如果你觉得这个立方体的每一个面都贴上了相同的纹理不好看，那么可以试着将每个面贴上不同的纹理，这样也有助于你更好地理解纹理映射的相关知识。希望通过本章的学习能够激发大家对 OpenGL 的兴趣。

12.7 光照和事件

上一节我们给立方体贴上了纹理，为了使程序效果更加美观、逼真，还可以用程序来模拟光照的效果，本节我们将学习如何为程序添加光照和通过按键来控制物体的旋转等操作。首先需要绘制一个立方体并贴上纹理，当我们按上下左右键时，该立方体向指定的方向旋转；当我们按 OK 键来控制光照的开关并打开光照时（如图 12-7 所示），离光源位置较远的地方光线较暗，当光照关闭时，屏幕上什么都看不到。

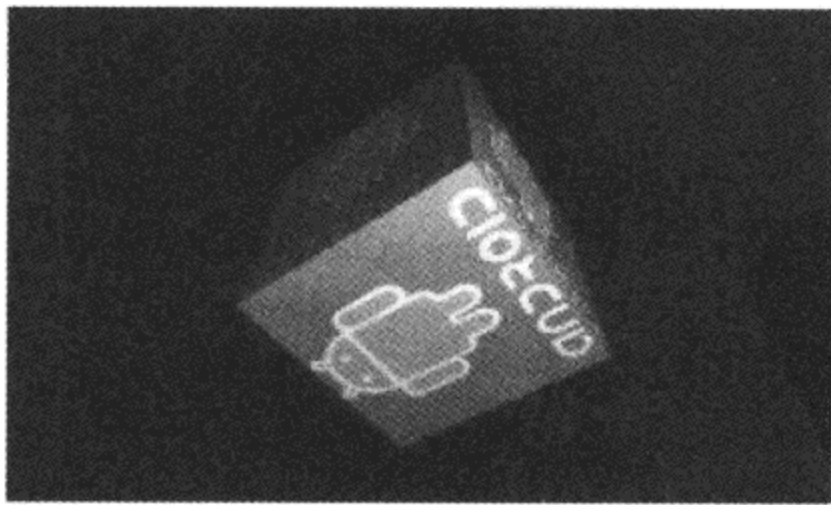


图 12-7 物体光照效果

1. 光照

要为程序添加光照效果，首先需要创建光源的数组。这里我们将使用两种不同的光。第一种称为环境光。环境光来自于四面八方，所有场景中的对象都处于环境光的照射中。第二种称为漫射光。漫射光由特定的光源产生，并在场景中的对象表面产生反射。处于漫射光直接照射下的任何对象表面都变得很亮，而几乎未被照射到的区域就显得要暗一些。这样，在我们所创建的木板箱的棱

边上就会产生很不错的阴影效果。创建光源的过程和创建颜色的过程完全一致。前 3 个参数分别是 RGB 三色分量, 最后一个是 **alpha** 通道参数。因此, 下面的代码得到的是半亮(0.5f)的白色环境光。如果没有环境光, 未被漫射光照到的地方会变得十分黑暗。

```
//定义环境光(r,g,b,a)
FloatBuffer lightAmbient = FloatBuffer.wrap(new float[]{0.5f,0.5f,0.5f,1.0f});
```

我们只需要定义最亮的漫射光, 所有的参数值都取成最大值 1.0f。代码如下:

```
//定义漫射光
FloatBuffer lightDiffuse = FloatBuffer.wrap(new float[]{1.0f,1.0f,1.0f,1.0f});
```

设置好了光源数组, 还需要定义光源在场景中所处的位置。光源的位置由 4 个数值组成, 前 3 个参数和 `glTranslate` 中的一样, 分别是 X、Y、Z 轴上的位移。由于我们想要光线直接照射在木箱的正面, 所以 XY 轴上的位移都是 0.0f。第三个值是 Z 轴上的位移。为了保证光线总在木箱的前面, 所以将光源的位置朝着观察者挪出屏幕。通常将屏幕(也就是显示器的屏幕玻璃)所处的位置称为 Z 轴的 0.0f 点, 所以 Z 轴上的位移最后定为 2.0f。假如你能够看见光源的话, 它就浮在显示器的前方。当然, 如果木箱不在显示器的屏幕玻璃后面的话, 你也无法看见箱子。最后一个参数取 1.0f, 这将告诉 OpenGL 这里指定的坐标就是光源的位置。本例中设置的光源位置如下代码所示:

```
//光源的位置
FloatBuffer lightPosition = FloatBuffer.wrap(new float[]{0.0f,0.0f,2.0f,1.0f});
```

到这里我们已经准备好了光源所需要的数据, 下面开始设置光源。我们可以通过 `glLightfv` 方法来设置一个光源, 如下代码分别是设置一个环境光和一个漫射光。第一个参数可以理解为光源的 ID, 当程序中包含多个光源时, 就可以通过这个 ID 来区分这些光源; 第二个参数描述了光源的类型, `GL_AMBIENT` 为环境光, `GL_DIFFUSE` 为漫射光; 第三个参数则是光源数组。

```
//设置环境光
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lightAmbient);
//设置漫射光
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lightDiffuse);
```

设置了光源数组之后, 还需要设置光源的位置, 光源的位置同样使用 `glLightfv` 方法来设置, 只是将第二个参数更改为 `GL_POSITION`, 表示设置光源的位置。代码如下:

```
//设置光源的位置
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, lightPosition);
```

最后还需要开启光源。我们还没有启用 `GL_LIGHTING`, 所以看不见任何光线。如果只对光源进行设置、定位甚至启用, 光源不会工作, 除非启用 `GL_LIGHTING`。如下代码开启上面所设置的光源:

```
//启用一号光源
gl.glEnable(GL10.GL_LIGHT1);
```

当然, 在不需要光源时, 可以将光源关闭掉, 关闭一个光源的代码如下:

```
//禁用一号光源
gl.glDisable(GL10.GL_LIGHT1);
```

2. 事件

事件的处理和以前一样, 通过 `onKeyUp` 和 `onKeyDown` 分别处理按键弹起和按下, 本例中的按键事件处理如代码清单 12-4 所示。具体实现参见本书所附代码: 第 12 章\Examples_12_06。

代码清单 12-4 按键事件

```

public boolean onKeyDown(int keyCode, KeyEvent event)
{
    switch ( keyCode )
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            key = true;
            xspeed=-step;
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            key = true;
            xspeed=step;
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            key = true;
            yspeed=-step;
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            key = true;
            yspeed=step;
            break;
        case KeyEvent.KEYCODE_DPAD_CENTER:
            light = !light;//光源开关
            break;
    }
    return false;
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    key = false;
    return false;
}

```

12.8 混合

相信大家在游戏中都看到过透明的效果，我们在纹理的基础上加上了混合，使它看起来具有透明的效果。下面先看看经过混合之后的立方体效果，如图 12-8 所示。具体实现参见本书所附代码：第 12 章\Examples_12_07。

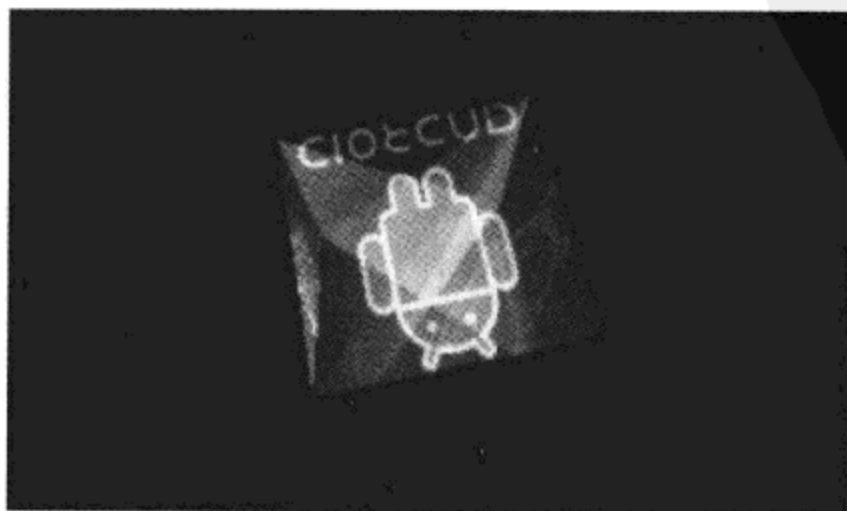


图 12-8 混合效果的立方体

1. 简单的透明

OpenGL 中的绝大多数特效都与某些类型的（色彩）混合有关。混色的定义为：将某个像素的颜色和已绘制在屏幕上与其对应的像素颜色相互结合。至于如何结合这两个颜色则依赖于颜色的 alpha 通道的分量值，以及/或者所使用的混色函数。alpha 通常是位于颜色值末尾的第 4 个颜色组成分量。前面我们都是用 GL_RGB 来指定颜色的 3 个分量。相应地，GL_RGBA 可以指定 alpha 分量的值。更进一步，可以使用 glColor4f() 来代替 glColor3f()。绝大多数人都认为 alpha 分量代表材料的透明度。这就是说，alpha 值为 0.0 时所代表的材料是完全透明的，alpha 值为 1.0 时所代表的材料是完全不透明的。

2. 混色公式

混色公式如下：

$$(R_s S_r + R_d D_r, G_s S_g + G_d D_g, B_s S_b + B_d D_b, A_s S_a + A_d D_a)$$

OpenGL 按照上面的公式计算两个像素的混色效果。小写的 s 和 r 分别代表源像素和目标像素，大写的 S 和 D 则是相应的混色因子。这些决定了如何对两个像素混色。大多数情况下，各颜色通道的 alpha 混色值大小相同，这样对源像素有 (As, As, As, As)，对目标像素则有 (1, 1, 1, 1) - (As, As, As, As)。上面的公式变成：

$$(R_s A_s + R_d (1 - A_s), G_s A_s + G_d (1 - A_s), B_s A_s + B_d (1 - A_s), A_s A_s + A_d (1 - A_s))$$

3. OpenGL 中的混色

在 OpenGL 中实现混色的步骤类似于我们以前提到的过程。接着设置公式，并在绘制透明对象时关闭写深度缓存。这不是正确的混色方法，但大多数时候这种做法在简单的项目中都工作得很好。考虑对两个多边形（1 和 2）进行 alpha 混合，不同的绘制次序会得到不同的结果这里假定多边形 1 离观察者较近，那么正确的过程应该是先画多边形 2，再画多边形 1。正如你在现实中所见到的那样，从这两个“透明的”多边形背后照射来的光线总是先穿过多边形 2，再穿过多边形 1，最后才到达观察者的眼睛。

在深度缓存启用时，应该将透明图形按照深度进行排序，并在全部场景绘制完毕之后再绘制透明物体，否则将得到不正确的结果。某些时候这样做是很令人痛苦的，但这是正确的方法。下面我们来看看如何在程序中使用混合实现透明效果。

首先，在 onSurfaceCreated 方法中加入如下代码：

```
//设置光线, 1.0f 为全光线, a=50%
gl.glColor4f(1.0f, 1.0f, 1.0f, 0.5f);
//基于源像素 alpha 通道值的半透明混合函数 gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE);
```

其中，glColor4f 以全亮度绘制此物体，并对其进行 50% 的 alpha 混合（半透明）。当混合选项打开时，此物体将会产生 50% 的透明效果，alpha 通道的值为 0.0 意味着物体材质是完全透明的，为 1.0 则意味着完全不透明。glBlendFunc 设置了混合的类型，即基于源像素 alpha 通道值的半透明混合。然后可以在程序中控制是否开启混合，需要开启混合时关闭深度测试，在关闭混合时打开深度测试，这样效果会更好。开启和关闭混合如代码清单 12-5 所示。

代码清单 12-5 开启（关闭）混合

```
//混合开关
if (key)
{
```

```
        // 打开混合
        gl.glEnable(GL10.GL_BLEND);
        // 关闭深度测试
        gl.glDisable(GL10.GL_DEPTH_TEST);
    }
    else
    {
        // 关闭混合
        gl.glDisable(GL10.GL_BLEND);
        // 打开深度测试
        gl.glEnable(GL10.GL_DEPTH_TEST);
    }
}
```

但是怎样才能在使用纹理贴图的时候指定混合时的颜色呢?很简单,在调整贴图模式时,纹理贴图的每个像素点的颜色都是由 **alpha** 通道参数与当前的像素颜色相乘所得到的。比如,绘制的颜色是 (0.5, 0.6, 0.4), 我们会把颜色相乘得到 (0.5, 0.6, 0.4, 0.2), 在没有指定 **alpha** 参数时,默认为零。通过本章的学习,相信大家都能 OpenGL 中实现 **alpha** 混合。

12.9 小结

通过本章的学习,我们已经学会了设置一个 OpenGL 窗口的每个细节,学会了在旋转的物体上贴图并加上光线以及混色(透明)处理。OpenGL 是一个与硬件无关的软件接口,可以在不同的平台(如 Windows 95、Windows NT、UNIX、Linux、MacOS、OS/2)之间进行移植。因此,支持 OpenGL 的软件具有很好的移植性,可以获得非常广泛的应用。由于 OpenGL 是 3D 图形的底层图形库,没有提供几何实体图元,所以不能直接用于描述场景。但是,通过一些转换程序,可以很方便地将 AutoCAD、3DS 等 3D 图形设计软件制作的 DFX 和 3DS 模型文件转换成 OpenGL 的顶点数组。因此目前 OpenGL 的应用非常广泛,按照 Andoid 目前的发展趋势,后期必定会支持更高版本的 OpenGL ES,让我们能够开发出更好的 3D 游戏。但是我们目前所学习的仅仅是 Android OpenGL ES 的基础知识,要想开发很好的 3D 产品,还需要进入下一章学习更高级的 OpenGL 技术。



Android OpenGL 综合应用

在三维绘图蓬勃发展的过程中，计算机公司推出了大量的三维绘图软件包，其中，SGI 公司推出了 OpenGL，它作为一个性能优越的图形应用程序设计界面（API）异军突起，取得了很大的成就。它以高性能的交互式三维图形建模能力和易于编程开发的特点，得到了 Microsoft、IBM、DEC、Sun、HP 等大公司的认同。因此，OpenGL 已经成为一种三维图形开发标准，是从事三维图形开发工作的必要工具。上一章我们已经学会了如何设置一个 OpenGL 窗口的细节，学会了在旋转的物体上贴图并加上光线以及混色、透明处理。但是仅凭这些基础知识要想做出优秀的 3D 游戏还很难，所以还需要继续学习。本章我们将在综合前面所学知识的同时，学习一些新的渲染技术。

13.1 移动图像

在学习了前面的知识后，大家是否在想：3D 空间中如何移动物体？如何在屏幕上绘制一个图像，从而让图像的背景色变为透明？如何实现一个简单的动画？那么，本节我们就学习在 3D 场景中移动图像，并去除图像上的黑色像素（使用混色）。接着为黑白纹理上色，最后学习创建丰富的色彩，并把上过不同色彩的纹理相互混合，得到简单的动画效果。

本节需要准备一张白色的图像，用于纹理贴图，如图 13-1 所示；然后使用该图像来实现很多颗星星自动绕成一个螺旋形的图案（动画效果），每颗星星都随机生成一种颜色，实现一个五颜六色的螺旋形图案，如图 13-2 所示。



图 13-1 纹理贴图

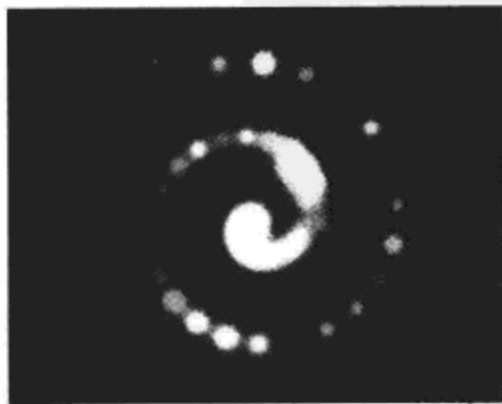


图 13-2 七彩星星效果

首先需要实现一个星星的类 `Star`，主要包括星星的颜色（`r`，`g`，`b`）。此外，每个星星离屏幕中心的距离不同，而且可以是以屏幕中心为原点的任意一个角度，因此，需要定义一个变量来保存星星距离中心的距离 `dist`。既然要旋转，还需要一个变量来表示当前星星所处的角度 `angle`。代码如下：

```

class Star
{
    int    r, g, b;           // 星星的颜色
    float  dist;              // 星星距离中心的距离
    float  angle = 0.0f;      // 当前星星所处的角度
}

```

由于一个屏幕上要显示很多星星，所以通过一个数组来保存这些星星，代码如下：

```

public static final int num = 50;           // 星星数目
public Star[] star = new Star[num];        // 存放星星的数组

```

首先需要将纹理贴图文件装载到程序中，并创建纹理，具体装载方式可以参考上一章的内容。需要注意的是，在设置 OpenGL 的渲染方式时，不要使用深度测试，而需要使用混色来启用纹理映射。代码如下：

```

gl.glEnable(GL10.GL_TEXTURE_2D);           // 启用纹理映射
gl.glShadeModel(GL10.GL_SMOOTH);           // 启用阴影平滑
gl.glClearColor(0.0f, 0.0f, 0.0f, 0.5f);   // 黑色背景
// 真正精细的透视修正
gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE); // 设置混色函数取得半透明效果
gl.glEnable(GL10.GL_BLEND);                // 启用混色

```

下面我们来初始化一个星星，设置每颗星星的起始角度、距离和颜色。如代码清单 13-1 所示，通过一个循环来初始化星星的属性，第 i 颗星星离中心的距离是将 i 的值除以星星的总颗数，然后乘上 $5.0f$ ，这样使得后一颗星星比前一颗星星离中心更远一点。当 i 为 50 时（最后一颗星星）， i 除以星星的总颗数（ num ）正好是 $1.0f$ 。之所以乘以 $5.0f$ ，是因为 $1.0f * 5.0f$ 就是 $5.0f$ 。 $5.0f$ 已经很接近屏幕边缘，不想让星星飞出屏幕， $5.0f$ 是最好的选择了。当然，如果将场景设置得更深入屏幕里面的话，也许可以使用大于 $5.0f$ 的数值，但星星看起来会更小一些（都是透视的缘故）。每颗星星的颜色都是 $0 \sim 255$ 之间的一个随机数。也许你会奇怪为何这里的颜色取值范围不是 OpenGL 通常的 $0.0f \sim 1.0f$ 。这是因为这里我们使用的颜色设置函数是 `glColor4ub`，而不是以前的 `glColor4f`。 `ub` 意味着参数是 Unsigned Byte 型的，一个 byte 的取值范围是 $0 \sim 255$ 。这里使用 byte 值取随机整数比取一个浮点的随机数更容易一些。

代码清单 13-1 初始化星星

```

// 创建循环设置全部星星
for (int i=0; i<num; i++)
{
    Star starTMP = new Star();
    // 所有星星都从零角度开始
    starTMP.angle=0.0f;
    // 计算星星离中心的距离
    starTMP.dist=((float) i)/(float) num)*5.0f;
    // 为 star[loop]设置随机红色分量
    starTMP.r=random.nextInt(256);
    // 为 star[loop]设置随机绿色分量
    starTMP.g=random.nextInt(256);
    // 为 star[loop]设置随机蓝色分量
    starTMP.b=random.nextInt(256);
    star[i] = starTMP;
}

```

初始化了星星的属性之后，要形成动画效果，需要移动星星。星星开始时位于屏幕的中心。我们要做的第一件事是把场景沿 Y 轴旋转。如果旋转 90 度的话，X 轴不再是自左至右的，而将由里向外穿出屏幕。为了让大家更清楚些，下面举例说明。假设你站在房子中间，你左侧的墙上写着 -X，前面的墙上写着 -Z，右侧的墙上就是 +X，身后的墙上则是 +Z。假如整个房子向右转 90 度，但你没有动，那么前面的墙上将是 -X 而不再是 -Z 了。所有其他的墙也都跟着移动。-Z 出现在右侧，+Z 出现在左侧，+X 出现在你身后。有点错乱了吧？通过旋转场景，我们改变了 X 和 Z 平面的方向。第二行代码沿 X 轴移动一个正值，通常 X 轴上的正值代表移向了屏幕的右侧（也就是通常的 X 轴的正向），但这里由于我们绕 Y 轴旋转了坐标系，X 轴的正向可以是任意方向。如果转 180 度的话，屏幕的左右侧就镜像成反向了。因此，当我们沿 X 轴正向移动时，可能向左、向右、向前或向后。代码如下：

```
gl.glRotatef(star[i].angle,0.0f,1.0f,0.0f);    // 旋转至当前所画星星的角度
gl.glTranslatef(star[i].dist,0.0f,0.0f);        // 沿 X 轴正向移动
```

接下来的代码有点小技巧。星星实际上是一个平面的纹理。现在在屏幕中心画了个平面的四边形，然后贴上纹理。但是当你沿着 Y 轴转 90 度的话，纹理在屏幕上就只剩右侧和左侧的两条边朝着你，看起来就是一条细线。这不是我们想要的，我们希望星星永远正面朝着我们，而不管屏幕如何旋转或倾斜。我们通过在绘制星星之前，抵消对星星所做的任何旋转来实现这个效果。可以采用逆序来抵消旋转。当我们倾斜屏幕时，实际上是以当前角度旋转了星星。通过逆序，我们又以当前角度反向旋转星星，也就是以当前角度的负值来旋转星星。这就是说，如果将星星旋转了 10 度的话，又将其旋转 -10 度来使星星在那个轴上重新面对屏幕。下面的第一行抵消了沿 Y 轴的旋转。然后，还需要抵消掉沿 X 轴的屏幕倾斜。要做到这一点，只需要将屏幕再旋转 -tilt 角度。在抵消掉 X 和 Y 轴的旋转后，星星又完全面对着我们了。

```
gl.glRotatef(-star[i].angle,0.0f,1.0f,0.0f);    // 取消当前星星的角度
gl.glRotatef(-tilt,1.0f,0.0f,0.0f);            // 取消屏幕倾斜
```

程序中我们定义了一个 **twinkle** 变量来控制星星的闪烁，如果 **twinkle** 为 **true**，在屏幕上先画一次不旋转的星星：将星星总数（**num**）减去当前的星星数（**loop**）再减去 1，来提取每颗星星的不同颜色（这么做是因为循环范围从 0~**num**-1）。举例来说，结果为 10 时，就使用 10 号星星的颜色，这样相邻星星的颜色总是不同的。最后一个值是 **alpha** 通道分量，这个值越小，这颗星星就越暗。由于启用了 **twinkle**，每颗星星最后会被绘制两遍。程序运行起来会慢一些，这要看机器性能如何了。但两遍绘制的星星颜色相互融合，会产生很棒的效果。同时由于第一遍的星星没有旋转，启用 **twinkle** 后的星星看起来有一种动画效果（可以先参考程序的运行效果）。值得注意的是，给纹理上色是件很容易的事。尽管纹理本身是黑白的，但它将变成我们在绘制它之前选定的任意颜色。此外，同样值得注意的是，这里使用的颜色值是 **byte** 型的，而不是通常的浮点数，甚至 **alpha** 通道分量也是如此。如代码清单 13-2 所示。

代码清单 13-2 绘制星星

```
// 启用闪烁效果
if (twinkle)
{
    // 使用 byte 型数值指定一个颜色
    gl.glColor4f((float)star[(num-i)-1].r/255.0f,(float)star[(num-i)-1].g/255.0f,
        (float)star[(num-i)-1].b/255.0f,1.0f);
```



```

gl.glVertexPointer(3, GL10.GL_FIXED, 0, vertexs);
gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, coord);
{
    coord.position(0);
    vertexs.position(0);
    indices.position(0);
    //绘制
    gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, 4, GL10.GL_UNSIGNED_BYTE, indices);
}
//绘制结束
gl.glFinish();
}

```

现在绘制第二遍的星星。唯一和前面的代码不同的是这一遍的星星肯定会被绘制，并且这次的星星绕着 Z 轴旋转。然后，增加 `spin` 的值来旋转所有的星星（公转）。接下来，将每颗星星的自转角度增加 `loop/num`，这使离中心更远的星星转得更快。最后，减少每颗星星离屏幕中心的距离，这样看起来星星好像被不断地吸入屏幕的中心。代码如下：

```

spin+=0.01f;           // 星星的公转
star[i].angle+=(float) (i)/(float) num;    / 改变星星的自转角度
star[i].dist-=0.01f;   // 改变星星离中心的距离

```

接着检查星星是否碰到了屏幕中心。当星星碰到屏幕中心时，为它赋一个新颜色，然后往外移 5 个单位，代码如下：

```

if (star[i].dist<0.0f) // 星星到达中心了吗
{
    star[i].dist+=5.0f; // 往外移 5 个单位
    star[i].r=random.nextInt(256); // 赋一个新红色分量
    star[i].g=random.nextInt(256); // 赋一个新绿色分量
    star[i].b=random.nextInt(256); // 赋一个新蓝色分量
}

```

到这里我们就已经实现了本节开始时所展示的效果了，具体实现参见本书所附代码：第 13 章 \Examples_13_01。本节基本上包括了前面所学的基础知识，可以参考本书第 12 章的内容。在学习了本节之后，大家可以自己独立完成一些简单的 Demo 了，加油，希望大家都能学好 OpenGL。

13.2 3D 世界

前面的章节中我们都是绘制一个 3D 的物体，在立体空间控制一个 3D 对象，大家是否在想：如何构建一个 3D 的场景呢？现在我们已经可以创建一个旋转的立方体或一群星星了，但是仅凭这些还难以完成一个 3D 大作，还需要一个大一点的、更复杂些的、动态的 3D 世界，它带有空间的六自由度和花哨的效果（如镜像、入口、扭曲等），当然还要有更快的帧显示速度。本节我们将完成一个 3D 世界的场景，如图 13-3 所示，可以通过上下左右键来控制场景的移动等操作。

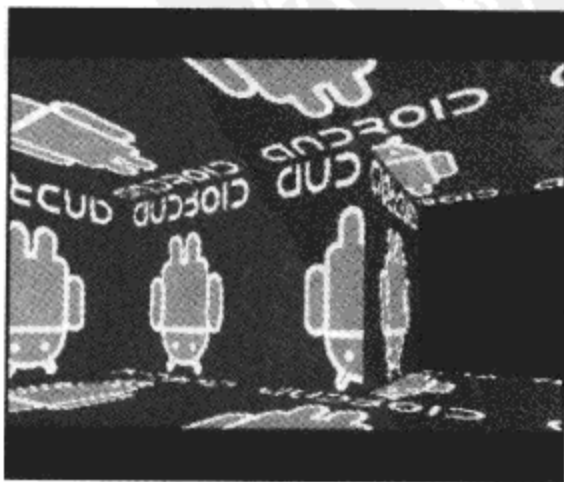


图 13-3 3D 世界

我们都知道，任何一个复杂的对象都是由一些简单的三角形构成的，所以在创建一个复杂的 3D 场景之前，首先定义一个场景的数据结构。三角形本质上是由一些（两个以上）顶点组成的多边形，顶点是最基本的分类单位，它包含了 OpenGL 真正有用的数据。我们用 3D 空间中的坐标值 (x,y,z) 以及它们的纹理坐标(u,v)来定义三角形的每个顶点。当然每个对象都不只是由一个三角形构成的，因此可以通过一个 List 来存储这些三角形。数据结构如代码清单 13-3 所示。

代码清单 13-3 ScData.java

```
package com.yarin.android.Examples_13_02;
import java.util.ArrayList;
import java.util.List;
//VERTEX 顶点结构
class VERTEX
{
    float x, y, z; // 3D 坐标
    float u, v; // 纹理坐标
    public VERTEX(float x, float y, float z, float u, float v)
    {
        this.x = x;
        this.y = y;
        this.z = z;
        this.u = u;
        this.v = v;
    }
}
//TRIANGLE 三角形结构
class TRIANGLE
{
    // VERTEX 矢量数组，大小为 3
    VERTEX[] vertex = new VERTEX[3];
}
//SECTOR 区段结构
class SECTOR
{
    // SECTOR 中的三角形个数
    int numtriangles;
    // 三角形的 List
    List<TRIANGLE> triangle = new ArrayList<TRIANGLE>();
}
```

当然一个场景必然由很多个顶点组成，由于这些顶点的数据量过大，如果写在程序之中，程序会显得很臃肿，所以这里将这些顶点存放到一个和游戏一起打包的文件中，然后在程序中通过装载这个文件来取得数据。本例中，我们首先将顶点数据存放到“assets/data/ world.txt”文件中，其数据存储方式如下（当然可以根据自己的需要按其他方式来存储数据，甚至一些加密算法等）：

```
X1 Y1 Z1 U1 V1 //x, y, z 坐标, u, v 纹理坐标
X2 Y2 Z2 U2 V2
X3 Y3 Z3 U3 V3
```

然后在程序中读取文件，具体代码如代码清单 13-4 所示。

代码清单 13-4 SetupWorld 方法

```
//读取资源数据
public void SetupWorld()
{
```

```

BufferedReader br = new BufferedReader(new InputStreamReader(GLFile.getFile
("data/world.txt")));
TRIANGLE triangle = new TRIANGLE();
int vertexIndex = 0;
try {
    String line = null;
    while((line = br.readLine()) != null){
        if(line.trim().length() <= 0 || line.startsWith("/")){
            continue;
        }
        String part[] = line.trim().split("\\s+");
        float x = Float.valueOf(part[0]);
        float y = Float.valueOf(part[1]);
        float z = Float.valueOf(part[2]);
        float u = Float.valueOf(part[3]);
        float v = Float.valueOf(part[4]);
        VERTEX vertex = new VERTEX(x, y, z, u, v);
        triangle.vertex[vertexIndex] = vertex;
        vertexIndex++;
        //3 个点构成一个三角形
        if(vertexIndex == 3){
            vertexIndex = 0;
            sector1.triangle.add(triangle);
            triangle = new TRIANGLE();
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
}

```

现在我们去掉了每个三角形的顶点数据，然后就是装载纹理贴图，将这些数据绘制到屏幕上就构建了所指定的场景样式了。代码清单 13-5 就是绘制这个场景的三角形的 List 的代码。

代码清单 13-5 绘制场景

```

for(TRIANGLE triangle : sector1.triangle)
{
    vertexPointer.clear();
    texCoordPointer.clear();
    gl.glNormal3f(0.0f, 0.0f, 1.0f);
    for(int i=0; i<3; i++)
    {
        VERTEX vt = triangle.vertex[i];
        vertexPointer.put(vt.x);
        vertexPointer.put(vt.y);
        vertexPointer.put(vt.z);
        texCoordPointer.put(vt.u);
        texCoordPointer.put(vt.v);
    }
    gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 4);
}

```

之前我们所做过的都是些简单的旋转和平移，我们的镜头始终位于原点 (0, 0, 0) 处，任何一个 3D 引擎都会允许用户在这个世界中遍历，这里也一样。实现这个功能的一种途径是直接移动镜头并绘制以镜头为中心的 3D 环境，这样做会很慢并且不易用代码实现，解决方法如下：

- ❑ 根据用户的指令旋转并变换镜头位置。
- ❑ 围绕原点，以与镜头相反的方向来旋转世界。

□ 以与镜头平移相反的方式来平移世界。

下面我们将通过按键事件来处理镜头的移动和旋转，当按右方向键时，只需将场景向左旋转，反之当按左方向键时，需要将场景向右旋转。当向下或向上方向键时，可以更改 x 和 z 平面的平移来达到场景移动的效果。本例中按键处理方法如代码清单 13-6 所示。

代码清单 13-6 按键处理

```
public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_LEFT:
            yrot -= 1.5f;
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            yrot += 1.5f;
            break;
        case KeyEvent.KEYCODE_DPAD_UP:
            // 沿游戏者所在的 X 平面移动
            xpos -= (float)Math.sin(heading*piover180) * 0.05f;
            // 沿游戏者所在的 Z 平面移动
            zpos -= (float)Math.cos(heading*piover180) * 0.05f;
            if (walkbiasangle >= 359.0f) // 如果 walkbiasangle 大于 359 度
            {
                walkbiasangle = 0.0f; // 将 walkbiasangle 设为 0
            }
            else
            {
                walkbiasangle += 10; // 如果 walkbiasangle < 359，则增加 10
            }
            // 使游戏者产生跳跃感
            walkbias = (float)Math.sin(walkbiasangle * piover180)/20.0f;
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            // 沿游戏者所在的 X 平面移动
            xpos += (float)Math.sin(heading*piover180) * 0.05f;
            // 沿游戏者所在的 Z 平面移动
            zpos += (float)Math.cos(heading*piover180) * 0.05f;
            // 如果 walkbiasangle 小于 1 度
            if (walkbiasangle <= 1.0f)
            {
                walkbiasangle = 359.0f; // 使 walkbiasangle 等于 359
            }
            else
            {
                walkbiasangle -= 10; // 如果 walkbiasangle > 1，则减去 10
            }
            // 使游戏者产生跳跃感
            walkbias = (float)Math.sin(walkbiasangle * piover180)/20.0f;
            break;
    }
    return false;
}
```

通过这里的学习相信大家已经喜欢上 OpenGL 了，体会到了 3D 世界的乐趣。该示例的具体实现参见本书所附代码：第 13 章\Examples_13_02。建议大家结合示例程序来阅读，这样可

以更深刻地理解 3D 场景。同时，大家可以考虑使用鼠标或者触笔来实现对场景的缩放。在创作一款游戏时，场景中可能不仅仅有一个数据结构，而且考虑到程序运行的效率，可以减少处理镜头背面（观察者视线所不能看到的地方）的三角形的绘制，这样会使程序运行得更加流畅。

13.3 飘动的旗帜

本节要实现一个在微风中飘动的旗帜的效果。前面我们学习了纹理映射和混合等基础知识，本节将对这些知识进行综合运用，并实现一个以正弦波方式运动的图像，看上去就像一面旗帜在风中飘动。所以，还需要使用数学函数 \sin 和 \cos 。首先来看看我们要实现的效果，如图 13-4 所示。具体实现参见本书所附代码：第 13 章\Examples_13_03。

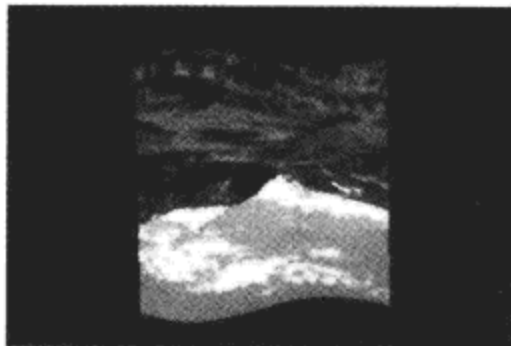


图 13-4 飘动的旗帜

我们将使用 `vertex` 数组来存放网格各顶点独立的 x , y , z 坐标。这里网格由 45×45 点形成，换句话说，也就是由 44×44 格的小方格依次组成的；`wiggle_count` 用来指定旗形波浪的运动速度；变量 `hold` 将存放一个用来对旗形波浪进行光滑的浮点数。代码如下：

```
float vertex[][][] = new float[45][45][3]; // Points 网格顶点数组 (45, 45, 3)
int wiggle_count = 0;                      // 指定旗形波浪的运动速度
float hold;                                // 临时变量
```

下面我们将初始化这个网格上的点，可以通过两个循环来实现，这里使用整数循环消除由于浮点运算取整所造成的脉冲锯齿。我们将 x 和 y 变量都除以 5，再减去 4.5，这样使得波浪可以“居中”。点 `[x][y][2]` 最后的值就是一个 \sin 函数计算的结果。 $\sin()$ 函数需要一个弧度参变量。将 `float_x` 乘以 $40.0f$ ，得到角度值。然后除以 $360.0f$ 再乘以 PI ，乘以 2，就转换为弧度了。初始化代码如下：

```
// 沿 X 平面循环
for(int x=0; x<45; x++)
{
    // 沿 Y 平面循环
    for(int y=0; y<45; y++)
    {
        // 向表面添加波浪效果
        vertex[x][y][0] = ((float)x/5.0f) - 4.5f;
        vertex[x][y][1] = ((float)y/5.0f) - 4.5f;
        vertex[x][y][2] = (float) (Math.sin((((float)x/5.0f)*40.0f)/360.0f)*
            3.141592654*2.0f));
    }
}
```

其他 OpenGL 设置和以前一样，下面开始绘制图形。同样，我们需要使用一个双循环来处理每个面的顶点和纹理，然后绘制。首先使用 4 个变量存放纹理坐标。每个多边形（网格之间的四边形）分别映射了纹理的 $1/44 \times 1/44$ 部分。循环首先确定左下顶点的值，据此得到其他三点的值，然后赋给顶点缓冲进行绘制。如代码清单 13-7 所示。

代码清单 13-7 绘制飘动的旗帜

```

for( x = 0; x < 44; x++ ){
    for( y = 0; y < 44; y++ ) {
        float_x = (float)(x)/44.0f; // 生成X浮点值
        float_y = (float)(y)/44.0f; // 生成Y浮点值
        float_xb = (float)(x+1)/44.0f; // X浮点值+0.0227f
        float_yb = (float)(y+1)/44.0f; // Y浮点值+0.0227f
        texCoord.clear();
        texCoord.put(float_x);
        texCoord.put(float_y);
        texCoord.put(float_x);
        texCoord.put(float_yb);
        texCoord.put(float_xb);
        texCoord.put(float_yb);
        texCoord.put(float_xb);
        texCoord.put(float_y);
        points.clear();
        points.put(vertex[x][y][0]);
        points.put(vertex[x][y][1]);
        points.put(vertex[x][y][2]);
        points.put(vertex[x][y+1][0]);
        points.put(vertex[x][y+1][1]);
        points.put(vertex[x][y+1][2]);
        points.put(vertex[x+1][y+1][0]);
        points.put(vertex[x+1][y+1][1]);
        points.put(vertex[x+1][y+1][2]);
        points.put(vertex[x+1][y][0]);
        points.put(vertex[x+1][y][1]);
        points.put(vertex[x+1][y][2]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);
    }
}

```

由于需要产生波浪效果,所以每绘制两次场景,循环一次 \sin 值,以产生运动效果。我们可以先存储每一行的第一个值,然后将波浪左移一下,使图像产生波浪效果。存储的数值移到末端以产生一个永无止境的波浪纹理效果,然后重置计数器 `wiggle_count` 以保持动画的进行。具体实现如代码清单 13-8 所示。

代码清单 13-8 产生波浪效果

```

if( wiggle_count == 2 ) // 用来降低波浪速度(每隔 2 帧一次)
{
    for( y = 0; y < 45; y++ )// 沿 Y 平面循环
    {
        hold=vertex[0][y][2]; // 存储当前左侧波浪值
        for( x = 0; x < 44; x++ )// 沿 X 平面循环
        {
            // 当前波浪值等于其右侧的波浪值
            vertex[x][y][2] = vertex[x+1][y][2];
        }
        vertex[44][y][2]=hold; // 刚才的值成为最左侧的波浪值
    }
    wiggle_count = 0; // 计数器清零
}

```

接下来实现图像的旋转。到这里我们就成功地实现了一个飘动的旗帜效果,回顾一下本节所使

用的知识，除了数学三角函数计算之外，最主要的就是纹理映射和混合。所以掌握基础知识很重要，掌握了基础知识后，大家可以根据自己的开发需要来设计出漂亮的 3D 效果。

13.4 显示列表

一般情况下要在一个场景中绘制多个类似的物体，最简单的方式就是通过循环来绘制，但是这样既增加了代码，也使得程序运行速度减慢，本节我们将学习如何使用显示列表。显示列表将加快程序的速度，而且可以减少代码的长度。比如，当我们制作游戏里的小行星场景时，每一层至少需要两个行星，可以用 OpenGL 中的多边形来构造每一个行星。较好的做法是做一个循环，每个循环画出行星的一个面，最终用几十条语句画出了一个行星。每次把行星画到屏幕上都是很困难的，当遇到更复杂的物体时，这个工程量就会大大增加。那么，解决的办法是什么呢？利用显示列表，只需一次性建立物体，将数据保存到列表中，就可以贴图、着色，想怎么用就怎么用。任何时候想在屏幕上画出行星，只需通过循环将这些数据绘制到屏幕上即可。因为小行星已经在显示列表里建造好了，所以 OpenGL 不会再计算如何构造它。这将大大降低 CPU 的使用，让程序运行得更快。本节我们将在屏幕上画 15 个立方体，每个立方体都由一个盒子和一个顶部构成，顶部是一个单独的显示列表，盒子没有顶。首先看看运行效果，如图 13-5 所示。

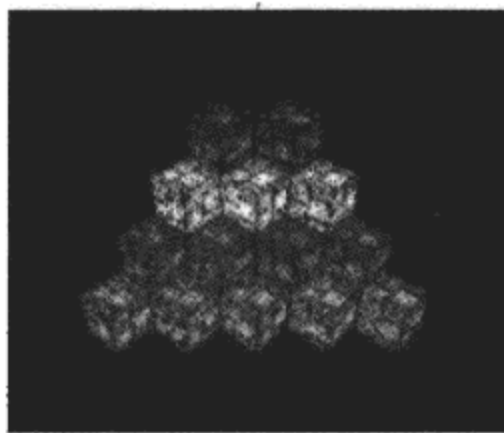


图 13-5 显示立方体列表

很容易实现图 13-5 所示的效果，装载纹理贴图和 OpenGL 设置与以前一样，不需要改变，只需要定义几个 FloatBuffer 类型的变量来保存列表的数据，包括顶点和贴图等。代码如下：

```
// 保存盒子的显示列表
FloatBuffer boxVertices = FloatBuffer.allocate(60);
FloatBuffer boxTexCoords = FloatBuffer.allocate(40);
// 保存盒子顶部的显示列表
FloatBuffer topVertices = FloatBuffer.allocate(12);
FloatBuffer topTexCoords = FloatBuffer.allocate(8);
```

图 13-5 中每个盒子的颜色都不同，因此需要定义两个数组来分别存储这些立方体盒子的颜色，如代码清单 13-9 所示。

代码清单 13-9 颜色数据

```
// 顶部的颜色数组
float[][] boxcol = {
    {1.0f, 0.0f, 0.0f},
    {1.0f, 0.5f, 0.0f},
    {1.0f, 1.0f, 0.0f},
    {0.0f, 1.0f, 0.0f},
    {0.0f, 1.0f, 1.0f},
};
// 盒子的颜色数组
float[][] topcol = {
    {0.5f, 0.0f, 0.0f},
    {0.5f, 0.25f, 0.0f},
```

```

        {0.5f, 0.5f, 0.0f},
        {0.0f, 0.5f, 0.0f},
        {0.0f, 0.5f, 0.5f},
    };

```

接下来我们将初始化这个列表数据，这也是本节的核心内容。其实就是保存了一个立方体的顶点和贴图数据，在绘制的时候循环绘制这些立方体即可，构建列表的具体内容如代码清单 13-10 所示。

代码清单 13-10 BuildLists 方法

```

public void BuildLists(GL10 gl)
{
    boxTexCoords.put(new float[]{1.0f, 1.0f,0.0f, 1.0f,0.0f, 0.0f,1.0f,0.0f});
    boxVertices.put(new float[]{-1.0f, -1.0f, -1.0f,1.0f, -1.0f, -1.0f,1.0f,
    -1.0f, 1.0f,-1.0f, -1.0f, 1.0f});

    boxTexCoords.put(new float[]{0.0f, 0.0f,1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f});
    boxVertices.put(new float[]{-1.0f, -1.0f, 1.0f,1.0f, -1.0f, 1.0f,1.0f,
    1.0f, 1.0f,-1.0f, 1.0f, 1.0f});

    boxTexCoords.put(new float[]{1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f,0.0f, 0.0f});
    boxVertices.put(new float[]{-1.0f, -1.0f, -1.0f,-1.0f, 1.0f, -1.0f,1.0f,
    1.0f, -1.0f,1.0f, -1.0f, -1.0f});

    boxTexCoords.put(new float[]{1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f,0.0f, 0.0f});
    boxVertices.put(new float[]{1.0f, -1.0f, -1.0f,1.0f, 1.0f, -1.0f,1.0f, 1.0f,
    1.0f,1.0f, -1.0f, 1.0f});

    boxTexCoords.put(new float[]{0.0f, 0.0f,1.0f, 0.0f,1.0f, 1.0f,0.0f, 1.0f});
    boxVertices.put(new float[]{-1.0f, -1.0f, -1.0f,-1.0f, -1.0f, 1.0f,-1.0f,
    1.0f, 1.0f,-1.0f, 1.0f, -1.0f});

    topTexCoords.put(new float[]{0.0f, 1.0f,0.0f, 0.0f,1.0f, 0.0f,1.0f, 1.0f});
    topVertices.put(new float[]{-1.0f, 1.0f, -1.0f,-1.0f, 1.0f, 1.0f,1.0f,
    1.0f, 1.0f,1.0f, 1.0f, -1.0f});
}

```

构建了一个立方体的数据之后，就剩下绘制了。下面我们将绘制出 15 个立方体盒子，并将其堆积成如图 13-5 所示的效果。这其实就是在绘制每一个立方体之前将其设置到相应的位置，组合成图形，如代码清单 13-11 所示。

代码清单 13-11 绘制列表

```

for (yloop=1;yloop<6;yloop++)
{
    for (xloop=0;xloop<yloop;xloop++)
    {
        gl.glLoadIdentity();// 重置模型变化矩阵
        // 设置盒子的位置
        gl.glTranslatef(1.4f+((float) (xloop)*2.8f)-((float) (yloop)*1.4f),((6.0f-
        (float) (yloop))*2.4f)-7.0f,-20.0f);

        gl.glRotatef(45.0f-(2.0f*yloop)+xrot,1.0f,0.0f,0.0f);
        gl.glRotatef(45.0f+yrot,0.0f,1.0f,0.0f);
        gl.glColor4f(boxcol[yloop-1][0], boxcol[yloop-1][1], boxcol[yloop-1][2], 1.0f);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, boxVertices);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, boxTexCoords);
        gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);
    }
}

```



```

gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 4, 4);
gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 8, 4);
gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 12, 4);
gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 16, 4);
/* Select The Top Color */
gl.glColor4f(topcol[yloop-1][0], topcol[yloop-1][1], topcol[yloop-1][2], 1.0f);
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, topVertices);
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, topTexCoords);
gl.glDrawArrays(GL10.GL_TRIANGLE_FAN, 0, 4);
}
}

```

最后我们来回顾一下有关灯光的知识，为场景中添加灯光，代码如下：

```

gl.glEnable(GL10.GL_LIGHT0); // 使用默认的 0 号灯
gl.glEnable(GL10.GL_LIGHTING); // 使用灯光
gl.glEnable(GL10.GL_COLOR_MATERIAL); // 使用颜色材质

```

Light0 一般来说是在显卡中预先定义过的，如果 Light0 不工作，那么后面的两行代码将不起作用。GL_COLOR_MATERIAL 使我们可以用颜色来贴纹理，如果没有这行代码，纹理将始终保持原来的颜色，glColor3f(r,g,b)就没有用了。

到这里本节内容就结束了，我们也在屏幕上绘制出了多个物体的列表，从严格意义上来说，这还不算是列表，列表是标准的 OpenGL 中的概念，可以使用 glNewList 方法来创建一个列表，创建结束之后通过调用 glEndList 方法来保存列表，然后在绘制的地方直接通过 glCallList 方法进行绘制。在学习本章的同时，大家可以去了解一下标准的 OpenGL 中的列表，这样有助于更好地理解本节的内容。

13.5 雾

在生活中经常能见到雾，大家是否在想：如何在 OpenGL 中实现雾的效果呢？本节我们就将学习三种不同的雾的计算方法，以及怎样设置雾的颜色和雾的范围。由于本节的内容非常简单，所以我们先来看看示例程序的运行效果和实现，然后再解释这些代码。运行效果如图 13-6 所示。

这里我们将实现三种雾的效果，因此首先将这三种雾的效果保存到一个数组中，并且定义好雾的颜色，代码如下：

```

// 雾气的模式
int fogMode[] = { GL10.GL_EXP, GL10.GL_EXP2, GL10.GL_LINEAR };
// 雾的颜色设为白色
float fogColor[] = { 0.5f, 0.5f, 0.5f, 1.0f };

```

下面将对雾效果的一些属性进行设置，然后在场景中开启雾效果，具体实现代码如代码清单 13-12 所示。

代码清单 13-12 雾效果

```

gl.glFogi(GL10.GL_FOG_MODE, fogMode[fogfilter]); // 设置雾气的模式
gl.glFogfv(GL10.GL_FOG_COLOR, fogColor, 0); // 设置雾的颜色

```



图 13-6 雾效果

```
gl.glFogf(GL10.GL_FOG_DENSITY, 0.35f); // 设置雾的密度
gl.glHint(GL10.GL_FOG_HINT, GL10.GL_DONT_CARE); // 设置系统如何计算雾气
gl.glFogf(GL10.GL_FOG_START, 1.0f); // 雾气的开始位置
gl.glFogf(GL10.GL_FOG_END, 5.0f); // 雾气的结束位置
gl.glEnable(GL10.GL_FOG); // 使用雾气
```

“glFogi(GL_FOG_MODE, fogMode[fogfilter]);”建立雾的过滤模式。之前我们声明了数组 fogMode, 它保存了值 GL_EXP、GL_EXP2 和 GL_LINEAR。下面是每一种雾效果的计算方式。

- GL_EXP: 充满整个屏幕的基本渲染的雾。它能在较老的 PC 上工作, 因此并不是特别像雾。
- GL_EXP2: 比 GL_EXP 更进一步。它也是充满整个屏幕, 但它使屏幕看起来更有深度。
- GL_LINEAR: 最好的渲染模式, 物体淡入淡出的效果更自然。

“glFogfv(GL_FOG_COLOR, fogcolor);”设置雾的颜色。之前我们已将变量 fogcolor 设为 (0.5f,0.5f,0.5f,1.0f), 所以这里直接使用 fogColor。“glFogf(GL_FOG_DENSITY, 0.35f);”设置雾的密度, 增大数字会让雾更密, 减小数字则雾更稀。“glHint (GL_FOG_HINT, GL_DONT_CARE);”设置修正, 这里使用了 GL_DONT_CARE 是不关心它的值。“glFogf(GL_FOG_START, 1.0f);”设定雾效果距屏幕多远开始, 可以根据需要随意改变这个值。“glFogf(GL_FOG_END, 5.0f);”告诉 OpenGL 程序雾效果持续到距屏幕多远。

本节的内容很简单, 但是很实用, 开发游戏时可能会经常遇到, 大家可以通过更改上面几个方法的数值来观察产生的雾的效果和每一种模式的雾的特点, 这样在开发过程中就可以根据不同的环境和需要选择使用不同模式和效果的雾, 使得游戏更加逼真。

13.6 粒子系统

看到本节的标题, 大家可能会眼前一亮, 因为粒子系统是很多读者最喜爱也是最受游戏玩家欢迎的一种效果。比如游戏中的爆炸、喷泉、流星、火焰之类的效果, 甚至一些法术技能的特效, 都可以通过粒子系统来完成。在学习本节之前我们先来了解一个概念——三角形带 (GL_TRIANGLE_STRIP), 简单地说, 就是画很多三角形来组合成我们要的形状。其实前面在绘图时已经解除过三角形带, 它非常容易使用, 其重要作用就是画很多三角形的时候能加快程序的运行速度。本节我们将学习使用三角形带来绘制一个复杂的微粒程序。首先我们需要了解微粒程序的原理, 可以假设赋予每个粒子生命、任意变化颜色、速度、重力影响等来适应环境的变化, 把每个粒子看成单一的从一个点运动到另一个点的颗粒。这样我们就可以很简单地创建出火、烟、喷泉、法术等特效。在写程序之前, 可以先运行本节的示例程序来了解一下我们究竟要实现一个什么样的功能和效果, 运行效果如图 13-7 所示。



图 13-7 粒子系统

它看上去很简单, 就是由一些点组成的图案。我们都知道, 在 OpenGL 中, 图像都是由许多三角形组成的, 所以只需将很多三角形按一定的规律组合成我们想要的效果即可。为了方便维护程序,

可以把每一个“点”抽象成一个物体，然后赋予这个物体一些属性。下面是本节示例程序中定义的一个专门表示这些“点”的类 `particle`，如代码清单 13-13 所示。

代码清单 13-13 `particle.java`

```
package com.yarin.android.Examples_13_06;
public class particle
{
    boolean active;    /* 是否激活 */
    float life;        /* 粒子生命 */
    float fade;        /* 衰减速度 */
    /* 颜色值(r,g,b) */
    float r;
    float g;
    float b;
    /* 每个粒子的位置 */
    float x;
    float y;
    float z;
    /* 每个粒子的方向 */
    float xi;
    float yi;
    float zi;
    /* 每个粒子的重力加速度 */
    float xg;    /* X Gravity */
    float yg;    /* Y Gravity */
    float zg;    /* Z Gravity */
}
```

其中，我们从布尔型变量 `active` 开始。如果变量 `active` 为 `true`，粒子为活跃的；如果为 `false` 则粒子为死的，此时就删除它。这里程序中没有使用活跃的。变量 `life` 和 `fade` 分别用来控制粒子显示多久以及显示时的亮度。随着 `life` 数值的降低，`fade` 的数值也相应降低，这将导致一些粒子比其他粒子燃烧的时间长。变量 `r`、`g` 和 `b` 分别用来表示粒子的红色强度、绿色强度和蓝色强度。当 `r` 的值变成 `1.0f` 时，粒子将会很红，当三个变量全为 `1.0f` 时，粒子将变成白色。变量 `x`、`y` 和 `z` 控制粒子在屏幕上显示的位置。`x` 表示粒子在 `x` 轴上的位置，`y` 表示粒子在 `y` 轴上的位置，`z` 表示粒子在 `z` 轴上的位置。`xi`、`yi` 和 `zi` 这三个变量控制粒子在每个轴上移动的快慢和方向。如果 `xi` 是负值粒子将会向左移动，正值粒子将会向右移动。如果 `yi` 是负值粒子将会向下移动，正值粒子将会向上移动。最后，如果 `zi` 是负值粒子将会向屏幕内部移动，正值粒子将向观察者移动。`xg`、`yg` 和 `zg` 可被看成加速度。如果 `xg` 是正值，粒子将会被拉向右边，负值将拉向左边。所以，如果粒子向左移动（负的）而给它一个正的加速度，粒子速度将变慢，最后将向反方向移动（高中物理所学）。`yg` 将粒子拉向下边或上边，`zg` 将粒子拉进或拉出屏幕。

另外，为了更好地操作和控制这些微粒，我们加入了如下一些变量：

```
public final static int MAX_PARTICLES =1000; //粒子的最大数量
float slowdown=2.0f; // 减速粒子
float xspeed;        // X 方向的速度
float yspeed;        // Y 方向的速度
float zoom=-40.0f;   // 沿 Z 轴缩放
int loop;            // 循环变量
int col;             // 当前的颜色
int delay;           // 彩虹效果延迟
```

其中, 变量 `slowdown` 控制粒子移动的快慢, 数值越高, 移动越慢, 数值越低, 移动越快, 如果数值降低, 粒子将快速地移动。粒子的速度影响其在屏幕中移动的距离。记住, 速度慢的粒子不会移动很远。变量 `xspeed` 和 `yspeed` 控制尾部的方向。`xspeed` 将会增加粒子在 x 轴上的速度。如果 `xspeed` 是正值, 粒子将会向右边移动多。如果 `xspeed` 是负值, 粒子将会向左边移动多。`yspeed` 与此类似, 只是在 y 轴上移动。因为有其他的因素影响粒子的运动, 所以这里用了“多”。`xspeed` 和 `yspeed` 有助于我们在想要的方向上移动粒子。变量 `zoom` 用来移入或移出屏幕。`col` 用来赋予粒子不同的颜色, `delay` 用来控制在彩虹模式中圆的颜色变化。

最后, 设定一个存储空间 (粒子纹理)。这里使用纹理而不使用点的重要原因是, 点的速度慢而且很麻烦。我们创建一个数组 `particle`。该数组存储 `MAX_PARTICLES` 个元素, 也就是说创建 1000(`MAX_PARTICLES`) 个粒子, 存储空间为每个粒子提供相应的信息。代码如下:

```
particle particles[] = new particle[MAX_PARTICLES];
```

在颜色数组上减少一些代码来存储 12 种不同的颜色, 对每一种颜色从 0 到 11 存储亮红、亮绿和亮蓝。下面的颜色表里包含从红色到紫罗兰色共 12 种渐变颜色。

```
static float colors[][] =
{
    {1.0f, 0.5f, 0.5f}, {1.0f, 0.75f, 0.5f}, {1.0f, 1.0f, 0.5f},
    {0.75f, 1.0f, 0.5f}, {0.5f, 1.0f, 0.5f}, {0.5f, 1.0f, 0.75f},
    {0.5f, 1.0f, 1.0f}, {0.5f, 0.75f, 1.0f}, {0.5f, 0.5f, 1.0f},
    {0.75f, 0.5f, 1.0f}, {1.0f, 0.5f, 1.0f}, {1.0f, 0.5f, 0.75f}
};
```

然后就是装载纹理贴图, 装载纹理贴图很简单, 大家可以参考本书第 12 章的内容, 初始化如代码清单 13-14 所示。

代码清单 13-14 ResetParticle 方法

```
public void ResetParticle(int num, int color, float xDir, float yDir, float zDir)
{
    particle tmp = new particle();
    tmp.active=true;
    tmp.life=1.0f;
    tmp.fade=(float) (rand()%100)/1000.0f+0.003f;
    tmp.r=colors[color][0];
    tmp.g=colors[color][1];
    tmp.b=colors[color][2];
    tmp.x=0.0f;
    tmp.y=0.0f;
    tmp.z=0.0f;
    tmp.xi=xDir;
    tmp.yi=yDir;
    tmp.zi=zDir;
    tmp.xg=0.0f;
    tmp.yg=-0.5f;
    tmp.zg=0.0f;
    particles[num] = tmp;
    return;
}
```

初始化每个粒子时, 从活跃的粒子开始, 如果粒子不活跃, 无论它有多少 `life`, 都不会出现在屏幕上。使粒子活跃之后, 给它 `life`。给粒子生命和颜色渐变是否是最好的方法呢? 当它运行一次后, 效果很好! `life` 的最大值是 1.0f, 这将给粒子完整的光亮。我们通过给定的值来设定粒子褪色

的快慢。每次粒子被拉的时候, life 随着 fade 而减小, 结束的数值将是 0~99 中的任意一个, 然后平分 1000 份来得到一个很小的浮点数。最后把结果加上 0.003f 来使 fade 速度值不为 0。既然粒子是活跃的, 而且又给它生命, 下面将给它颜色数值。一开始, 想给每个粒子不同的颜色。怎么做才能使每个粒子与前面颜色箱里的颜色一一对应呢? 用 loop 变量乘以箱子中颜色的数目与粒子最大值 (MAX_PARTICLES) 的余数即可做到, 这样可以防止最后的颜色数值大于最大的颜色数值 (12), 比如: $900 * (12 / 900) = 12$, $1000 * (12 / 1000) = 12$ 。关于每个粒子移动的方向和速度, 通过将结果乘以 10.0f 来创建开始时的爆炸效果。我们将会以任意一个正值或负值结束, 这个数值将以任意速度、任意方向移动粒子。最后, 设定加速度的数值。不像一般的加速度仅仅把事物拉下, 这里的加速度能拉出、拉下、拉左、拉右、拉前和拉后粒子。开始需要强大的向下加速度, 为了达到这样的效果, 将 xg 设为 0.0f, 即在 x 方向没有拉力, 而将 yg 设为 -0.8f 来产生一个向下的拉力。如果值为正, 则向上拉。我们不希望粒子拉近或远离我们, 所以将 zg 设为 0.0f。

现在我们来绘制这些粒子, 首先取得当前粒子的 x 位置, 然后把 x 运动速度加上粒子被减速 1000 倍后的值, 所以如果粒子在 x 轴 (0) 上屏幕中心的位置, 运动值 (xi) 为 x 轴方向+10 (移动方向为右), 而 slowdown 等于 1, 将以 $10 / (1 * 1000)$ 或 0.01f 速度移向右边。如果增加 slowdown 值到 2, 我们只移动 0.005f。在计算出下一步粒子移到哪里后, 开始考虑重力和阻力。将阻力 (xg) 和移动速度 (xi) 相加, 移动速度是 10, 阻力是 1。每时每刻粒子都在抵抗阻力。第二次画粒子时, 阻力开始作用, 移动速度将会从 10 降到 9。第三次画粒子时, 阻力再一次作用, 移动速度降到 8。如果画粒子超过 10 次, 它将会结束, 并向相反方向移动, 因为移动速度变成负值。阻力同样作用于 y 和 z 移动速度。

在粒子重新设置之后, 将赋给它新的移动速度/方向。注意: 增加最大值和最小值, 粒子移动速度为 50~60 之间的任意值, 但是这次我们没有将移动速度乘以 10, 因为这次不想要一个爆发的效果, 而是比较慢地移动粒子。还要注意: 把 xspeed 和 x 轴移动速度相加, y 轴移动速度和 yspeed 相加, 以此控制粒子的移动方向。

最后, 分配给粒子一种新的颜色。变量 col 保存一个从 0 到 11 (12 种颜色) 的数字, 用这个变量去找红、绿、蓝亮度 (在颜色箱里面), 下面第一行表示红色的强度, 数值保存在 colors[col][0]。所以如果 col 是 0, 红色的亮度就是 1.0f。绿色的值和蓝色的值用相同的方法读取。绘制粒子的方法如代码清单 13-15 所示。

代码清单 13-15 onDrawFrame 方法

```
public void onDrawFrame(GL10 gl)
{
    FloatBuffer vertices = FloatBuffer.wrap(new float[12]);
    FloatBuffer texcoords = FloatBuffer.wrap(new float[8]);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertices);
    gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texcoords);
    gl.glLoadIdentity();
    for (loop = 0; loop < MAX_PARTICLES; loop++)
    {
        if (particles[loop].active)
        {
            float x = particles[loop].x;
```



```

float y = particles[loop].y;
float z = particles[loop].z + zoom;
gl.glColor4f(particles[loop].r, particles[loop].g, particles[loop].b,
particles[loop].life);
texcoords.clear();
vertices.clear();
texcoords.put(1.0f);
texcoords.put(1.0f);
vertices.put(x + 0.5f);
vertices.put(y + 0.5f);
vertices.put(z);
texcoords.put(0.0f);
texcoords.put(1.0f);
vertices.put(x - 0.5f);
vertices.put(y + 0.5f);
vertices.put(z);
texcoords.put(1.0f);
texcoords.put(0.0f);
vertices.put(x + 0.5f);
vertices.put(y - 0.5f);
vertices.put(z);
texcoords.put(0.0f);
texcoords.put(0.0f);
vertices.put(x - 0.5f);
vertices.put(y - 0.5f);
vertices.put(z);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
particles[loop].x += particles[loop].xi / (slowdown * 1000);
particles[loop].y += particles[loop].yi / (slowdown * 1000);
particles[loop].z += particles[loop].zi / (slowdown * 1000);
particles[loop].xi += particles[loop].xg;
particles[loop].yi += particles[loop].yg;
particles[loop].zi += particles[loop].zg;
particles[loop].life -= particles[loop].fade;
if (particles[loop].life < 0.0f)
{
    float xi, yi, zi;
    xi = xspeed + (float) ((rand() % 60) - 32.0f);
    yi = yspeed + (float) ((rand() % 60) - 30.0f);
    zi = (float) ((rand() % 60) - 30.0f);
    ResetParticle(loop, col, xi, yi, zi);
}
}
}
gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glFinish();
}

```

到这里我们已经实现了一个复杂的粒子效果，大家还可以为程序加上按键、触摸等事件来控制这些粒子的速度、大小、光线等操作，这样可以更好地理解 OpenGL 中的粒子效果。另外，大家可以根据自身的情况和学习的进度，将本程序进行简单的修改，从而实现一个漂亮的烟花效果。

13.7 蒙版

到目前为止，大家已经学习了如何使用 alpha 混合把一个透明物体渲染到屏幕上，但有的并不

是很符合我们的要求。本节我们将使用蒙版技术，按照蒙版的位置精确地进行绘制。如图 13-8 所示是本节例子的运行效果，从图中可以看出网状物体下面绘制了一些图像。

蒙版是一种控制深度在蒙版后面的物体，在蒙版区域的某个位置是否显示的机制。这个功能可以使用混色通过控制 **alpha** 通道的值来完成，但是这样做的效果并不好，会使得蒙版上的物体比较虚。使用蒙版以后，可以使得透过深度较靠前的物体看到深度较靠后的物体的一部分。这个机制可以用两种方式来实现。

第一种方式是使用模拟的方式，它的原理是使用混色当中的像素叠加操作。所以在操作的一开始必须要打开混色开关“`gl.glEnable(GL10.GL_BLEND);`”，然后在需要的地方放置蒙版，蒙版是一个黑白贴图，这个贴图在需要透视后方物体的地方使用白色，而在需要遮挡的地方使用黑色。在载入并绑定蒙版贴图以后，控制混色方案为“`gl.glBlendFunc(GL10.GL_DST_COLOR, GL10.GL_ZERO);`”，这样贴图的黑色部分被贴至指定位置，而白色部分则没有贴图。接着载入并绑定与蒙版对应的贴图，并且更换混色方案为“`gl.glBlendFunc(GL10.GL_ONE, GL10.GL_ONE);`”，然后贴图，这样做的效果是仅仅将原先蒙版贴到场景中的部分贴上图。至此，一个蒙版的使用完成。

第二种方式使用了 OpenGL 当中的蒙版缓存。使用这种机制的方法是首先在创建场景时打开蒙版缓存，然后在需要的时候用 `glEnable` 允许蒙版缓存的使用。在完成以上的初始化工作以后，可以开始使用蒙版缓存，在正式使用之前需要用蒙版缓存指定接下来允许进行绘图的地方。因此需要向蒙版缓存当中写入关于这个区域的数据，而写入的过程类似于绘制的过程。在写入这些数据之前，为了保证写入的过程不会改写深度缓存和颜色缓存，必须指定所有的颜色“掩码”为 `false` 来保证没有任何颜色写入到颜色缓存，而对于深度缓存的操作则非常简单，直接用 `glDisable` 关闭之就可以了。然后，开始写入蒙版缓存。

下面我们来学习具体的实现方法。

首先装载纹理贴图，如下代码是装载纹理贴图的一部分。

```
gl.glBindTexture(GL10.GL_TEXTURE_2D, texture[0]);
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, GLImage.mBitmapLogo, 0);
gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
```

`onSurfaceCreated` 和 `onSurfaceChanged` 方法与以前的示例一样，不用改变。下面我们开始本节最核心的部分——绘制方法 `onDrawFrame`。

有人说 OpenGL 是一个基于顶点的图形系统，你设置的大部分参数是作为顶点的属性而记录的，纹理坐标就是这样一种属性。只要简单地设置各个顶点的纹理坐标，OpenGL 就会通过一个插值的过程自动帮你把多边形内部填充纹理。

像前面几章一样，假定四边形面对我们，并把纹理坐标 (0,0) 绑定到左下角，(1,0) 绑定到右下角，(1,1) 绑定到右上角，那么四边形中间对应的纹理坐标为 (0.5,0.5)，但我们并没有设置此处的纹理坐标！



图 13-8 蒙版技术

可以通过设置纹理坐标达到滚动纹理的目的。纹理坐标是被归一化的，它的范围为 0.0~1.0，值 0 被映射到纹理的一边，值 1 被映射到纹理的另一边。超过 1 的值，纹理可以按照不同的方式被映射，这里设置为它将回绕到另一边并重复纹理。如果使用这样的映射方式，(0.3,0.5) 和 (1.3,0.5) 将被映射到同一个纹理坐标。在本节中，我们将尝试一种无缝填充的效果。

我们使用 roll 变量设置纹理坐标，当它为 0 时，把纹理的左下角映射到四边形的左下角，如图 13-9 所示。当它大于 0 时，把纹理的左上角映射到四边形的左下角，如图 13-10 所示，看起来的效果就是纹理沿四边形向上滚动。

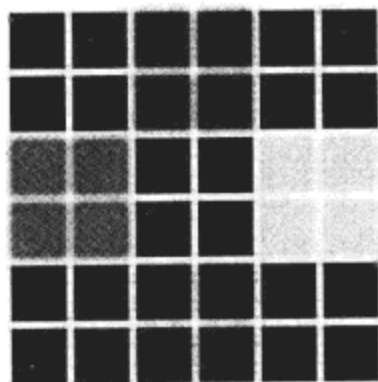


图 13-9 初始纹理

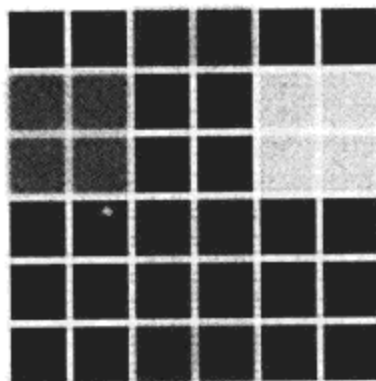


图 13-10 变换之后的纹理

然后启用混合和禁用深度测试，代码如下：

```
gl.glEnable(GL10.GL_BLEND);
gl.glDisable(GL10.GL_DEPTH_TEST);
```

接下来，需要根据 **masking** 的值设置是否使用“掩码”，如果是，则需要设置相应的混合系数。如果启用了“掩码”，则要设置“掩码”的混合系数。一个“掩码”只是一幅绘制到屏幕的纹理图片，但只有黑色和白色，白色的部分代表透明，黑色的部分代表不透明。下面这个混合系数使得任何对应“掩码”黑色的部分会变为黑色，白色的部分会保持原来的颜色。代码如下：

```
// 使用黑白“掩码”混合屏幕颜色
gl.glBlendFunc(GL10.GL_DST_COLOR, GL10.GL_ZERO);
```

现在我们检查绘制哪一个层，如果 **scene** 为 **true** 绘制第二个层，否则绘制第一个层。为了不使它看起来显得非常大，我们把它移入屏幕一个单位，并把它按 **roll** 变量的值进行旋转（沿 Z 轴）。

```
gl.glTranslatef(0.0f, 0.0f, -1.0f);
gl.glRotatef(roll*360, 0.0f, 0.0f, 1.0f);
```

如果启用了“掩码”，我们会把掩码绘制到屏幕。完成这个操作时，将会看到一个镂空的纹理出现在屏幕上，如代码清单 13-16 所示。

代码清单 13-16 绘制掩码

```
if (masking)
{
    gl.glBindTexture(GL10.GL_TEXTURE_2D, texture[3]);

    texcoords.clear();
    texcoords.put(0.0f); texcoords.put(1.0f);
    texcoords.put(1.0f); texcoords.put(1.0f);
    texcoords.put(1.0f); texcoords.put(0.0f);
    texcoords.put(0.0f); texcoords.put(0.0f);

    vertices.clear();
```

```

        vertices.put(-1.1f); vertices.put(-1.1f); vertices.put(0.0f);
        vertices.put(1.1f); vertices.put(-1.1f); vertices.put(0.0f);
        vertices.put(1.1f); vertices.put(1.1f); vertices.put(0.0f);
        vertices.put(-1.1f); vertices.put(1.1f); vertices.put(0.0f);

        gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, 4, GL10.GL_UNSIGNED_BYTE, indices);
    }

```

当我们把“掩码”绘制到屏幕上后，接着变换混合系数。这次我们告诉 OpenGL 把任何黑色部分对应的像素绘制到屏幕，这样看起来纹理就像镂空地贴在屏幕上一样。注意，变换了混合模式后再选择纹理。如果没有使用“掩码”，图像将与屏幕颜色混合。如代码清单 13-17 所示。

代码清单 13-17 绘制纹理贴图

```

gl.glBlendFunc(GL10.GL_ONE, GL10.GL_ONE);
gl.glBindTexture(GL10.GL_TEXTURE_2D, texture[4]);

texcoords.clear();
texcoords.put(0.0f); texcoords.put(1.0f);
texcoords.put(1.0f); texcoords.put(1.0f);
texcoords.put(1.0f); texcoords.put(0.0f);
texcoords.put(0.0f); texcoords.put(0.0f);

vertices.clear();
vertices.put(-1.1f); vertices.put(-1.1f); vertices.put(0.0f);
vertices.put(1.1f); vertices.put(-1.1f); vertices.put(0.0f);
vertices.put(1.1f); vertices.put(1.1f); vertices.put(0.0f);
vertices.put(-1.1f); vertices.put(1.1f); vertices.put(0.0f);

gl.glDrawElements(GL10.GL_TRIANGLE_STRIP, 4, GL10.GL_UNSIGNED_BYTE, indices);

```

绘制完成之后启用深度测试，禁用混合。最后增加 roll 变量，如果大于 1，把它的值减 1。

```

gl.glEnable(GL10.GL_DEPTH_TEST);
gl.glDisable(GL10.GL_BLEND);
roll+=0.002f;
if (roll>1.0f)
{
    roll-=1.0f;
}

```

到这里我们已经学会了 OpenGL 中的蒙版技术，利用它能高效地绘制一部分纹理而不使用 alpha 值。具体实现参见本书所附代码：第 13 章\Examples_13_07。蒙版就好像隔着网状物往里面看的效果一样。下面我们总结一下使用蒙版技术的具体步骤：

(1) 网状物是一个纹理，网线为彩色，镂空地方为黑色，预先为它生成一个一样的黑白纹理，使镂空的地方为白色，网线为黑色。

(2) 绘制网内的纹理。

(3) 用“gl.glBlendFunc(GL 10.GL_DST_COLOR, GL 10.GL_ZERO);”画黑白纹理，其中 GL_DST_COLOR 对应的混合因子为 (Rd, Gd, Bd, Ad)，也就是帧缓存中原像素的颜色。把黑白纹理与 GL_DST_COLOR 相乘。网线为黑色，则结果为黑色，镂空为白色，则结果为原像素的颜色。

(4) 用“gl.glBlendFunc(GL GL_ONE, GL GL_ONE);”画网状物的纹理，镂空的地方为黑色，则使用原像素的颜色。上步渲染中，镂空地方为原像素的颜色。网线部分为现在颜色+第 3 步后的颜色。第 3 步网线为黑色，即值为 0，则网线部分为现在颜色 + 0 = 现在颜色。

蒙版技术在游戏开发过程中使用颇多，希望大家好好掌握。

13.8 变形

这一节的内容非常有趣，对于初学 OpenGL 的人来说，算是一个很大的进步。本节我们将学习如何从文件加载 3D 模型，并且平滑地从一个模型变换为另一个模型。需要注意的是，各个变形的模型必须有相同的顶点，才能一一对应，并应用变形。先来看看我们的示例程序的变形效果，起初为球体形状，如图 13-11 所示，变形过程中的效果如图 13-12 所示，变形为一个圆柱体的效果如图 13-13 所示，变形为一个圆形弹簧的效果如图 13-14 所示。



图 13-11 球体形状

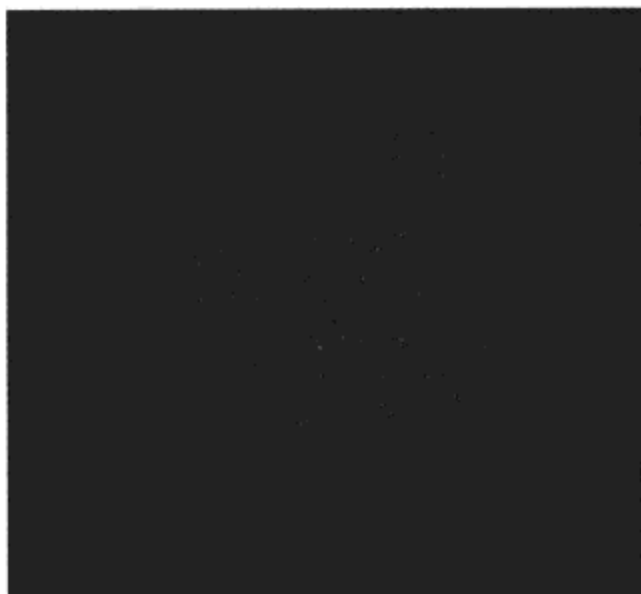


图 13-12 变形中



图 13-13 圆柱体



图 13-14 圆形弹簧

下面我们将一步一步地学习如何实现这些变形的效果。我们需要添加几个旋转变量，用来记录旋转的信息，并使用 `cx`、`cy`、`cz` 设置物体在屏幕上的位置。这里使用 `step` 来设置相邻变形之间的中间步骤。比如 `step` 为 200，则需要 200 次才能把一个物体变为另一个物体。最后用 `morph` 变量来设置是否使用变形。代码如下：

```
float xrot, yrot, zrot,
xspeed, yspeed, zspeed,
cx, cy, cz=-15;
boolean morph=false;
int step=0, steps=200;
```

由于三维空间中的每一个点都由 `x`、`y`、`z` 三个值来确定，下面我们定义一个三维顶点的类：

```

class VERTEX{
    float x, y, z;
    public VERTEX(float x,float y,float z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
}

```

一个物体（模型）必然由很多个点组成，因此需要定义一个类 **OBJECT** 来表示三维空间中的物体，该类中需要包括模型的顶点数 **verts** 和每一个物体的顶点列表，如下代码所示：

```

class OBJECT
{
    int     verts;
    List<VERTEX> points  = new ArrayList<VERTEX>();
}

```

既然要变形，就需要准备几个供变形的模型文件，模型文件保存了模型中的各个顶点、颜色等数据。本示例中的模型数据如下所示，当然大家可以根据自己的需要定义模型数据。

```

//Vertices
486 (顶点数)
-0.106    1.593    2.272 //顶点(x,y,z).

```

然后在程序中加载这些模型。下面我们来学习如何加载模型文件。首先实现一个方法 **readstr** 来读取一行数据，并判断是否是注释（/）或者换行（\n），如果是则重新读取一行，如代码清单 13-18 所示。

代码清单 13-18 readstr 方法

```

public String readstr(BufferedReader br){
    String str = "";
    try{
        do {
            str=br.readLine();
        } while ((str.charAt(0) == '/') || (str.charAt(0)=='\n'));
    }catch (Exception e){}
    return str;
}

```

然后通过一个循环来把所有的顶点都装载到我们定义的顶点 **List** 变量 **points** 中。具体实现如代码清单 13-19 所示。

代码清单 13-19 objload 方法

```

void objload(String name, OBJECT k)
{
    int    ver = 0;
    String oneline;
    int    i;
    BufferedReader br = new BufferedReader(new InputStreamReader(GLFile.getFile(name)));
    //读出顶点数
    oneline = readstr(br);
    ver=Integer.valueOf(oneline).intValue();
    k.verts = ver;
    for (i=0; i<ver; i++){
        oneline=readstr(br);
        String part[] = oneline.trim().split("\\s+");
        float x = Float.valueOf(part[0]);

```

```

        float y = Float.valueOf(part[1]);
        float z = Float.valueOf(part[2]);
        VERTEX vertex = new VERTEX(x, y, z);
        k.points.add(vertex);
    }
    if (ver>maxver){
        maxver=ver;
    }
}

```

然后初始化 OpenGL, 将混合模式设置为半透明。接下来装载一开始准备好的 3 个模型文件, 第 4 个模型文件用于变形中的效果, 在 $(-7, -7, -7) \sim (7, 7, 7)$ 之间随机生成模型点, 它和我们载入的模型一样, 都具有 486 个顶点。具体实现如代码清单 13-20 所示。

代码清单 13-20 onSurfaceCreated 方法

```

public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    int i = 0;
    gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE);
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    gl.glClearDepthf(1.0f);
    gl.glDepthFunc(GL10.GL_LESS);
    gl.glEnable(GL10.GL_DEPTH_TEST);
    gl.glShadeModel(GL10.GL_SMOOTH);
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);

    maxver=0;
    objload("sphere.txt",morph1);
    objload("torus.txt",morph2);
    objload("tube.txt",morph3);

    for(i=0; i<486; i++)
    {
        float xx=((float)(rand()%14000)/1000)-7;
        float yy=((float)(rand()%14000)/1000)-7;
        float zz=((float)(rand()%14000)/1000)-7;
        morph4.points.add(new VERTEX(xx,yy,zz));
    }
    objload("sphere.txt", helper);
    sour=dest=morph1;
}

```

最后将模型文件绘制到屏幕上即可, 通过按键事件来改变需要绘制的模型文件, 并实现变形效果, 绘制方法如代码清单 13-21 所示。

代码清单 13-21 onDrawFrame 方法

```

public void onDrawFrame(GL10 gl)
{
    int i;
    float tx, ty, tz;
    VERTEX q = new VERTEX(0,0,0);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl.glLoadIdentity();
    gl.glTranslatef(cx, cy, cz);
    gl.glRotatef(xrot, 1, 0, 0);
    gl.glRotatef(yrot, 0, 1, 0);
    gl.glRotatef(zrot, 0, 0, 1);
    xrot+=xspeed; yrot+=yspeed; zrot+=zspeed;
}

```

```

gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertices);
gl.glColorPointer(4, GL10.GL_FLOAT, 0, colors);
gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
colors.clear();
vertices.clear();
for(i=0; i<morph1.verts; i++)
{
    /* If morph Is True Calculate Movement Otherwise Movement=0 */
    if (morph)
    {
        q=calculate(i);
    }
    else
    {
        q.x=q.y=q.z=0;
    }
    helper.points.get(i).x-=q.x;
    helper.points.get(i).y-=q.y;
    helper.points.get(i).z-=q.z;
    tx=helper.points.get(i).x;
    ty=helper.points.get(i).y;
    tz=helper.points.get(i).z;

    colors.put(0.0f); colors.put(1.0f); colors.put(1.0f); colors.put(1.0f);
    vertices.put(tx); vertices.put(ty); vertices.put(tz);

    colors.put(0.0f); colors.put(0.5f); colors.put(1.0f); colors.put(1.0f);
    tx-=2*q.x; ty-=2*q.y; ty-=2*q.y;
    vertices.put(tx); vertices.put(ty); vertices.put(tz);

    colors.put(0.0f); colors.put(0.0f); colors.put(1.0f); colors.put(1.0f);
    tx-=2*q.x; ty-=2*q.y; ty-=2*q.y;
    vertices.put(tx); vertices.put(ty); vertices.put(tz);
}

gl.glDrawArrays(GL10.GL_POINTS, 0, morph1.verts*3);

gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisableClientState(GL10.GL_COLOR_ARRAY);

if (morph && step<=steps)
{
    step++;
}
else
{
    morph=false;
    sour=dest;
    step=0;
}
gl.glFinish();
}

```

到这里，我们就将模型文件绘制到屏幕上了，有关按键事件的处理，可参见本书所附代码：第13章\Examples_13_08，这里需要提醒一下，我们的模型文件放在工程目录下的 assets 文件夹中。

13.9 小结

本章我们学习了很多 3D 游戏开发中很实用的技术，主要包括移动图像、3D 世界、飘动的旗帜、显示列表、雾、粒子系统、蒙版、变形等，相信大家对 OpenGL 有了更深入的了解。大家在以后的学习中，需要同时考虑时间和空间上的优缺点（时间即速度，运行的效率；空间则是程序所占用的内存空间和硬盘空间），才能开发出更优秀的 OpenGL 程序。目前国内在移动手持设备上的 3D 技术还处于发展阶段，需要我们大家的共同努力。3D 世界其实丰富多彩，同时也有更多的东西需要我们去研究、去发现。



第 14 章

游戏引擎实现

最早的手机游戏出现于 1997 年，经过 10 年的发展，随着手机终端和移动通信网络的不断进步，手机游戏也正在经历由简单到复杂的进化过程。从全球来看，手机娱乐服务被公认为带动移动数据业务快速发展的重要力量。近年来，伴随着移动网络和移动终端性能的不断提高与完善，作为手机娱乐服务的重要内容之一的手机游戏业务呈现快速增长的势头，成为一座名副其实的“金矿”。而市场的发展导致了需求量的迅猛增长，如何缩短手机游戏的开发周期，并且降低成本和风险，成为游戏开发者亟待解决的问题。

国外媒体报道，美国市场研究公司 Gartner 预计，谷歌 Android 到 2012 年将超越 iPhone、Windows Mobile 和黑莓，成为继 Symbian 之后全球第二大智能手机操作系统，市场份额达到 14.5%。本章通过对 Android 平台游戏的程序结构进行研究，分析了代码的相似性，并据此设计了一个游戏引擎，它可以有效减少游戏开发者编写游戏程序时的冗余劳动，同时增强游戏的可移植性。

14.1 游戏引擎介绍

在阅读各种游戏软件的介绍时，我们常常会碰见“引擎”(Engine)这个单词，引擎在游戏中究竟起着什么样的作用？它的进化对于游戏的发展产生了哪些影响？本节我们将对游戏引擎做一个全面的介绍。

14.1.1 什么是引擎

我们可以把游戏的引擎比作赛车的引擎，大家知道，引擎是赛车的核心，决定着赛车的性能和稳定性。赛车的速度、操纵感这些直接与车手相关的指标都建立在引擎的性能基础之上。游戏也是如此，玩家所体验到的剧情、关卡、美工、音乐、操作等内容都是由游戏的引擎直接控制的，它扮演着发动机的角色，把游戏中的所有元素捆绑在一起，在后台指挥它们同时、有序地工作。简单地说，引擎就是“用于控制所有游戏功能的主程序，从计算碰撞、物理系统和物体的相对位置，到接受玩家的输入，以及按照正确的音量输出声音等。”

可见，引擎并不是什么神秘的东西，无论是 2D 游戏还是 3D 游戏，无论是角色扮演游戏、即时策略游戏、冒险解谜游戏，或是动作射击游戏，甚至一个只有几十 KB 的小游戏，都有这样一段起控制作用的代码。经过不断进化，如今的游戏引擎已经发展为一套由多个子系统共同构成的复杂系统，从建模、动画到光影、粒子特效，从物理系统、碰撞检测到文件管理、网络特性，还有专业的编辑工具和插件，几乎涵盖了开发过程中的所有重要环节。

引擎还有一个重要的职责就是负责玩家与电脑之间的沟通,处理来自键盘、鼠标、摇杆和其他外设的信号。如果游戏支持联网特性,那么网络代码也会被集成在引擎中,用于管理客户端与服务端之间的通信。

通过上面这些枯燥的介绍我们至少可以了解到一点:引擎相当于游戏的框架。框架打好后,关卡设计师、建模师、动画师只要往里填充内容就可以了。因此,在 3D 游戏的开发过程中,引擎的制作往往会占用非常多的时间,《马科斯-佩恩》的 MAX-FX 引擎从最初的雏形 Final Reality 到最终的成品共花了四年多时间,LithTech 引擎的开发共花了整整五年时间,耗资 700 万美元,Monolith 公司(LithTech 引擎的开发者)的老板詹森·霍尔甚至不无懊悔地说:“如果当初意识到制作自己的引擎要付出这么大的代价,我们根本就不可能去做这种傻事。没有人会预料得到五年后的市场究竟是怎样的。”

正是出于节约成本、缩短周期和降低风险这三方面的考虑,越来越多的开发者倾向于使用第三方的现成引擎制作自己的游戏,一个庞大的引擎授权市场已经形成。

14.1.2 引擎的进化

曾经有一段时期,游戏开发者关心的只是如何尽量多地开发出新的游戏并把它们推销给玩家。尽管那时的游戏大多简单粗糙,但每款游戏的平均开发周期也要达到 8 个月以上,这一方面是由于技术的原因,另一方面则是因为几乎每款游戏都要从头编写代码,造成了大量的重复劳动。渐渐地,一些有经验的开发者摸索出了一条捷径,他们借用上一款类似题材的游戏中的部分代码作为新游戏的基本框架,以节省开发时间和开发费用。根据马老先生的生产力学说,单位产品的成本因生产力水平的提高而降低,自动化程度较高的手工业者最终将把那些生产力低下的手工业者淘汰出局,引擎的概念就是在这种机器化作业的背景下诞生的。

14.1.3 常见的游戏引擎

每一款游戏都有自己的引擎,但真正能获得他人认可并成为标准引擎的并不多。纵观九年多的发展历程,我们可以看出引擎最大的驱动力来自于 3D 游戏,尤其是 3D 射击游戏。尽管像 Infinity 这样的 2D 引擎也有着相当久远的历史,从《博德之门》(Baldur's Gate)系列到《异域镇魂曲》(Planescape: Torment)、《冰风谷》(Icewind Dale)、《冰风谷 2》,但它的应用范围毕竟局限于“龙与地下城”风格的角色扮演游戏,包括颇受期待的《夜在绝冬城》(Neverwinter Nights)所使用的 Aurora 引擎,它们都有着十分特殊的使用目的,很难对整个引擎技术的发展起到推动作用,这也是为什么体育模拟游戏、飞行模拟游戏和即时策略游戏的引擎很少进入授权市场的原因,开发者即便使用第三方引擎也很难获得理想的效果,采用《帝国时代 2》(Age of Empires)引擎制作的《星球大战:银河战场》(Star Wars: Galactic Battleground)就是一个最好的例子。

□ 天堂 2

正是通过《天堂 2》,很多玩家才认识了韩国著名的游戏开发商 Ncsoft,当时《天堂 2》依靠真实感十足的画面赢得了无数玩家的厚爱,甚至有许多玩家为了这款游戏更换电脑。不过很可惜,由于外挂问题,《天堂 2》在中国未能取得理想的市场效果,目前,《天堂 2》由 Ncsoft 中国分公司独立运营。说起《天堂 2》的引擎就不得不说在游戏界叱咤风云的 Epic 公司,成立于 1991 年由 Epic

公司开发的 Unreal（虚幻）引擎系列拥有很高的声誉，而《天堂 2》所采用的正是 Unreal 2 引擎，这也让《天堂 2》成为了当时最华丽的游戏之一。

□ 完美时空

完美时空依靠 Element3D 游戏引擎，在中国的网游市场上一炮走红，并先后依靠它开发了《完美世界》、《武林外传》、《诛仙》、《赤壁》等多款网游。完美时空的游戏给人最显著的感觉就是：游戏画面细致出色，渲染能力强，但是战斗时明显缺乏打击感。

□ 魔兽世界

如果说你会玩网游，却从没听过《魔兽世界》，估计会被不少人耻笑，暴雪第一次踏足网游业的同时也宣告了网络游戏新时代的开始，《魔兽世界》也成了无数玩家心中难得一见的神作，而长达 10 年的开发周期和天文数字一般的开发费也足以成为网游史上的里程碑。这样网游史上的佳作肯定带来了一款强大的“游戏引擎”，《魔兽世界》正是凭借强大的引擎，支持着庞大的无缝连接场景、众多的副本、柔和自然的场景表现、复杂的技能属性运算，勾勒出了一个唯美的艾泽拉斯大陆。

14.1.4 Android 游戏引擎

Android 因其充分的开放性、架构与功能的完整性，一经推出，便迅速成长为手机、上网本及其他各种移动互联网设备（MID）领域重要的应用软件平台。可见 Android 的应用领域非常广泛，我们不排除将来会把 Android 应用到 PC 上，所以由于其广泛的应用领域，就决定了 Android 平台的应用程序也需要多样化，当然游戏就更需要满足不同领域的需求了，因此我们在开发一款基于 Android 平台的游戏引擎时，就需要根据其应用领域的不同进行不同的偏好设置。

当然，不管应用于什么领域，其本质都是 Android 系统，所以接下来，我们就根据 Android 系统的特点来学习如何开发一个完整的 Android 平台游戏引擎。

14.2 游戏引擎结构

游戏引擎是一个为运行某一类游戏的机器设计的能够被机器识别的代码（指令）集合。它像一个发动机，控制着游戏的运行。一个游戏作品可以分为游戏引擎和游戏资源两大部分。游戏资源包括：图像、声音、动画等，由此列出一个公式，如图 14-1 所示。游戏引擎则是按游戏设计的要求顺序地调用这些资源。



图 14-1 游戏构成公式

14.2.1 游戏引擎原理

游戏开发以及任何软件在设计时，都需要建立一个稳定的引擎结构体来作为软件设计的基本架

构，它是软件体系结构的核心支撑框架，一切功能的实现以及扩展都在这个基础架构之上来完成。一个游戏的设计包括：游戏控制、角色、游戏场景、道具、游戏声效以及游戏过程事件监听机制等元素，因此建立良好的游戏引擎能对以上各元素进行有效地控制。

游戏引擎是为运行某类游戏的机器所设计的，能够被该机器所识别的代码集合，它定义了该类游戏的代码框架，负责按照开发人员设计的游戏执行逻辑依次调用各种游戏资源（如图像、声音等），指挥它们有序地协同工作，是控制游戏运行的发动机。如同人们能够使用相同的引擎，制造出拥有不同外壳的各款汽车一样，开发人员可以在游戏引擎框架上根据需要添加相应内容（包括体现游戏情节的执行逻辑和游戏中使用的各种多媒体资源）来设计不同的游戏，这一方面简化了游戏的开发流程，使开发人员能够将把精力集中在游戏的情节设计上，另一方面也提高了代码的可靠性和游戏的开发速度。

14.2.2 游戏引擎定位

要开发出一款优秀的游戏引擎，首先就需要了解一下一款优秀的游戏具有什么特征？手机游戏的盈利主要是由于它们的涉及面很广。手机已经与现代生活紧密地结合在一起。它们是最普及的个人携带品之一，仅次于钥匙和钱包。传统的台式机游戏将目标锁定在低级趣味的人和青少年身上，而手机游戏能被每个人随时随地访问。尽管每个手机游戏都不贵，但是巨大的使用量（如：每人每星期玩一个新游戏）将使得这个市场商机无限并且有利可图。而对于开发者来说，将控制台游戏迁移到手机游戏的工程很大，因为它们所面向的对象、生活方式和分布式模型都有着极大的区别。一个成功的手机游戏大多具有以下特征。

- ❑ 易于学习：既然手机游戏面向的是普通消费者而不是计算机高手，那么他们不可能深入地学习游戏技巧。消费者不会花几个小时去研究一个 3 美元的手动操作的游戏。所以游戏必须是一下载就可以玩的。保持游戏的简单是最基本的要求。
- ❑ 可中断性：多任务处理是手机生活方式的基本特征。手机用户常常在任务（如等一个电子邮件或者等车）之间有一小段时间，而游戏、日历管理、通讯和工作数据访问使用的是同一个设备。所以一个好的手机游戏应该提供短时间的娱乐功能，并且允许用户在游戏和工作模式之间顺利切换。
- ❑ 基于订阅：手机游戏的盈利取决于它们巨大的使用量。起初开发和设计每个游戏都是昂贵的。如果一个手机游戏开发者要赚钱的话，重要的是：同一个游戏引擎，多个标题，基本的故事情节类似。基于订阅的游戏是不断产生收入的最好方法。
- ❑ 丰富的社会交互：不管一个游戏设计得多好，只要玩家找到了它的根本模式或者迅速玩完了所有的游戏关卡，他就会厌烦这个游戏。对于一个基于订阅的游戏，重要的是与别的玩家合作以增强所玩游戏的智力性和随机性。在今天纷繁复杂的多玩家游戏中，具有丰富社会交互的游戏被证明是成功的。
- ❑ 利用手机技术的优点：巨额的手机技术研发费用都花在提高设备和网络的可用性和可靠性上面。因此，手机设备硬件和网络协议与桌面/控制台世界（如 GPS 扩展、条形码扫描仪、和 SMS/MMS 通讯）有着非常大的差别。好的手机游戏应该利用那些更新的设备特征和网络基础设备的优点。

- 无违法内容：既然所有年龄/性别的人群都玩手机游戏，并且常常在公共/工作场合，你应该避免明显的暴力或者色情内容。

另外，要开发一款引擎首先要分析游戏的类型，因为根据不同的游戏类型，可以确定游戏引擎要实现的模块和重点实现。

14.2.3 游戏引擎框架

Borland Jbuilder、Eclipse 和 BEA Workbench 等企业开发环境不外乎都是基于 Apache Struts、Spring、Hibernate 等开发框架，而这些框架几乎都遵循了模型-视图-控制（MVC）设计模式，即商业逻辑和描述被分开，由一个逻辑流控制器来协调来自客户端的请求和服务上将采取的行动。这条途径逐渐成为了开发者们共同遵守的一个不成文的标准。每个框架的内在的机制固然是不同的，但是开发者们使用它来设计和实现他们的应用软件的结构是很类似的。而在游戏开发中，MVC 框架也有着举足轻重的地位，所以我们在开发游戏引擎时也同样可以使用这个框架，将游戏引擎分为三大部分，如图 14-2 所示。

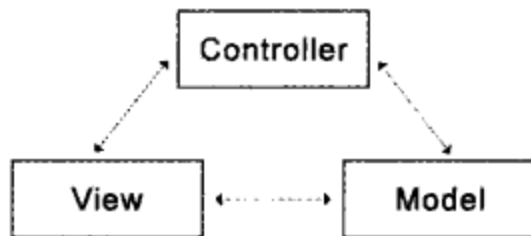


图 14-2 MVC 框架

- 视图类（View）：主要负责游戏的界面绘制。
- 控制类（Controller）：主要处理工作线程的创建和中止，处理虚拟时间的流逝，读取用户输入，并交由其他类处理。它还要处理各类定时器事件。
- 逻辑类（Model）：主要包含游戏的模型，游戏本身的各种功能以及游戏中的所有逻辑计算等。

当然这只是一个大的框架，在具体设计时还会根据具体的需求来设计一系列的子系统，比如：物力系统、碰撞检测系统，离子系统以及游戏实体对象类等。同时你也可以不选择 MVC 框架而使用自己的特殊框架来完成引擎的设计。

14.3 游戏引擎设计

游戏引擎的设计主要包括引擎的结构设计、引擎的功能设计以及设计时所需要注意的问题。

14.3.1 游戏引擎结构和功能设计

作为游戏设计的基础，游戏引擎也是游戏和一些交互式实时图形应用程序设计的核心组件，它的主要工作是设计游戏中的物体成像、物理演算、碰撞运算、玩家角色的操作以及播放正确的音量和声音输出等必要功能。大部分游戏引擎都是由图形引擎、音效引擎、物理引擎、事件模块、网络模块、工具模块、引擎脚本等部分组成，如图 14-3 所示。图形引擎用于产生游戏里的角色及周边场景的图形，把读取的所有数据即时转化成屏幕显示的图形，即可视化；音效引擎用于控制游戏中各种音乐效果输出的部分；物理引擎用于在游戏中准确地实现物理模拟，让游戏更逼真地展现物体的碰撞、翻滚、反弹等效果；事件模块负责游戏中与玩家交互部分（比如按键、触笔事件等）和游

戏内部事件（比如游戏状态的改变等）的设计；网络模块负责让玩家通过联网进行对战任务；工具模块主要是针对不同的游戏来定的（比如 RPG 游戏的地图编辑器、精灵编辑器等）；脚本引擎的功能非常强大，它可以让用户对游戏引擎的代码进行修改。

在这些引擎模块中，图形引擎是游戏引擎的关键，其性能直接影响游戏的可玩性和可操作性，决定了游戏整体质量的层次和今后的市场走势。其中 2D 图形引擎主要使用在 2D 游戏中，是绘制图形并向外部表达图形的系统。在 3D 游戏中，也会使用 2D 图形引擎来绘制界面以及一些二维元素。

有了这样一个大的结构，我们就可以根据引擎具体所要实现的功能来进行更详细的分析和设计。进行游戏引擎的设计，需要分析游戏的运行机制并提取各款游戏的基本构成元素。每个平台游戏的运行机制都稍有不同，那么下面我们分析 Android 平台游戏的运行机制。

Android 游戏主要包括一个 Activity 类、一个流程控制类、一个游戏线程类和若干个游戏对象类。其中，Activity 类是游戏的基本执行单元，负责游戏生命周期的控制，如游戏的启动、暂停、退出等。流程控制类提供了在游戏的多个界面（如启动画面、主菜单、游戏场景、帮助信息等）之间切换的方法，使用户能控制游戏的运行。游戏线程类不断地循环检测可能发生的各种事件，计算游戏状态，并刷新屏幕。手机游戏程序处理的事件可以分为两类：一类由硬件装置产生（如键盘被按下），另一类由游戏中的内部对象产生（如游戏对象发生碰撞）。游戏中所有可见的东西都是游戏对象，它们是游戏中实际运动的实体。游戏对象需要定义该类对象对应的图片和能够执行的动作。当指定事件发生时，游戏对象会根据运行逻辑执行相应的动作。为了增强运行效果，游戏中有时会使用播音器、定时器、对话框等控件。比较复杂的游戏还可以按情节划分为若干个逻辑单元，每个单元称为一个游戏场景。由此，设计出如图 14-4 所示的 Android 平台游戏引擎结构图。

从图 14-4 可以看出，Android 平台游戏引擎就是将游戏的每个对象进行对比、分类、封装，在不同的状态时，每个对象展示着不同的内容（比如：在一个 RPG 游戏中，主角与 NPC 在战斗状态和在对话状态时图形、音效、环境都各不相同，这些则是游戏引擎所需要实现的），这样就形成了游戏，因此游戏引擎主要就是针对游戏对象来进行设计的。当然这些游戏对象就和游戏的类型有很大的关系了，比如：RPG 游戏中的游戏对象可以是一个场景，可以是一个 NPC，还可以是一个道具，甚至任何类型的游戏中的任何一种显示或隐藏的游戏元素都可以是游戏对象，因此我们在设计引擎时也可以根据游戏的类型来进行更深入的设计，这样更能节省开发时间，缩短开发周期。



图 14-3 游戏引擎结构



图 14-4 Android 平台游戏引擎结构图

14.3.2 游戏引擎设计注意事项

作为手机游戏发展的最大优势，便携性同时也为手机游戏设计带来许多限制，例如功耗方面，

手机的电池资源有限，不能运行能耗大的游戏；由于硬件限制，目前大部分手机设备的运算速度也不尽如人意，导致游戏动画不够流畅，动画帧数甚至达不到 10 帧/秒，在实际游戏中容易造成玩家视觉疲劳；色彩数量、明亮度和声音支持有限，屏幕大小直接导致游戏中各种角色造型的大小及表现力，影响玩家的感受和游戏的可玩性；内存空间大小受限；此外，手机游戏还受到网络响应速度等因素的限制。

从总体上看，目前的手机游戏引擎基本上都是由软件完成绘图、音效播放等全部的功能设计。用软件实现绘图的全部操作，特别是便携式设备中游戏的绘图操作，速度慢、完成相应任务所需消耗的能量高，很难达到游戏对性能的要求及便携式设备对低功耗的要求。而且，在手机游戏设计中，为了追求透明光影的效果，通常都会使用到 Alpha 混合、马赛克、淡入淡出、缩放旋转等颜色特效，处理这些颜色特效需消耗大量 CPU 资源，功耗过大会导致电池使用时间短，给使用者带来很大的不便。同时，功耗的增加还会带来一系列问题，例如电路参数漂移、可靠性下降、芯片封装成本增加等。虽然在手机游戏设计中进行了软件代码优化，改进了游戏引擎的质量，但是面对当今手机便携式设备对显示性能日益提高的要求，仅对软件进行优化处理是不够的，还必须提供硬件加速引擎。

14.4 游戏引擎实现

在 14.3 节中，我们对游戏引擎进行了设计，那么本节我们将按照上一节所设计的模块，来实现一个简单的游戏引擎。本节和上一节的内容结合得非常紧密，在实现过程时大家可以回顾上一节的内容。

14.4.1 Activity 类实现

我们都知道，要在 Android 中要实现一个应用程序界面，就必须有一个继承自 Activity 的类，可以将其理解为应用程序的主类、入口程序。在 Activity 类中我们将对应用程序的生命周期进行控制，所以我们首先需要实现如下几个与 Activity 生命周期相关的方法。

- ❑ onCreate(): 当 Activity 第一次被创建的时候调用。
- ❑ onRestart(): 当 Activity 从停止运行的状态到开始运行的状态的时候被调用。
- ❑ onStart(): 当 Activity 展示给用户看到的时候被调用。如果本 Activity 处于 Activity 栈的顶部，则 onStart 方法紧随着 onResume 方法的调用而调用。
- ❑ onResume(): 当 Activity 开始将与用户进行交互时被调用。
- ❑ onPause(): 当其他 Activity 被激活时，当前 Activity 的 onPause 将被调用。
- ❑ onStop(): 当前 Activity 不再对用户可见的时候被调用（即可能是一个新的 Activity 被激活，或者是当前的 Activity 被销毁）。
- ❑ onDestroy(): 当前 Activity 被销毁之前被调用。要销毁一个 Activity，可以调用 finish() 方法。

同时，大家不要忘记，Activity 也提供了事件处理函数，比如：按键按下、按键弹起、触笔等，为了能更好地展示出该类的功能，我们用图 14-5 来展示。



图 14-5 Activity 功能结构图

Activity 类的实现很简单，当然要根据具体的游戏来进行操作，比如：销毁的时候需要释放资源，中断时需要特殊处理等。具体实现参见代码清单 14-1。

代码清单 14-1 第 14 章\GameEngine\src\com\yarin\android\GameEngine\GameActivity.java

```
public class GameActivity extends Activity
{
    /* 创建 */
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    /* 创建菜单 */
    public boolean onCreateOptionsMenu(Menu menu)
    {
        return super.onCreateOptionsMenu(menu);
    }
    /* 销毁 */
    protected void onDestroy()
    {
        super.onDestroy();
    }
    /* 按键按下 */
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        return super.onKeyDown(keyCode, event);
    }
    /* 按键重复按下 */
    public boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)
    {
        return super.onKeyMultiple(keyCode, repeatCount, event);
    }
    /* 按键释放 */
    public boolean onKeyUp(int keyCode, KeyEvent event)
    {
        return super.onKeyUp(keyCode, event);
    }
    /* 菜单事件 */

```

```

    public boolean onOptionsItemSelected(int featureId, MenuItem item)
    {
        return super.onOptionsItemSelected(featureId, item);
    }
    /* 暂停 */
    protected void onPause()
    {
        super.onPause();
    }
    /* 重新开始 */
    protected void onRestart()
    {
        super.onRestart();
    }
    /* 重新激活 */
    protected void onResume()
    {
        super.onResume();
    }
    /* 开始 */
    protected void onStart()
    {
        super.onStart();
    }
    /* 停止 */
    protected void onStop()
    {
        super.onStop();
    }
    /* 触笔事件 */
    public boolean onTouchEvent(MotionEvent event)
    {
        return super.onTouchEvent(event);
    }
    /* 设置标题 */
    //使用 MVC 框架让资源文件通过 R.java 文件读取
    public void setTitle(int titleId)
    {
        super.setTitle(titleId);
    }
}

```

这里列出了大部分需要实现的方法，如果需要更加完善，可以参考 Activity 类来进行设计，当然笔者建议使用 MVC 框架，让资源文件通过 R.java 文件读取，比如：setTitle 和 onCreateOptionsMenu 方法等。关于 Activity 类就写到这里，下面我们将实现流程控制类。

14.4.2 流程控制和线程

流程控制类主要用于根据游戏状态的不同来控制不同界面之间的切换，所以我们需要继承自 View 类或者 Surfaceview 类。为了简化，并且让大家更好地理解流程控制，我们选择继承自 View 类，由于前面我们讲述过这两个类（参考第 5 章），这里我们要使代码结构看上去很清楚明白，所以使用了状态机制。状态机制，就是对游戏定义了很多个状态，每一个状态所显示的界面都不同，当我们在绘制的时候就根据状态的不同来选择绘制不同的界面。因此首先我们可以定义一系列的游

戏状态,如代码清单 14-2 所示,它分别定义了游戏中常用的集中状态。这里笔者建议大家可以将游戏中需要使用的所有常量都定义在一个独立的类中,方便管理。

代码清单 14-2 第 14 章\GameEngine\src\com\yarin\android\GameEngine\GameDefinition.java

```
public class GameDefinition
{
    /* 游戏状态 */
    public static final int Game_Logo = 0;
    public static final int Game_MainMenu = 1;
    public static final int Game_Set = 2;
    public static final int Game_Help = 3;
    public static final int Game_About = 4;
    public static final int Game_Over = 5;
    //显示下一条信息(用于对话模式)
    public static final int PAGEDOWN_KEYPRESS = 11;
    //确认(用于对话模式)
    public static final int OK_KEYPRESS = 12;
    //取消(用于对话模式)
    public static final int CANCEL_KEYPRESS = 13;
}
```

由于线程类中我们需要随时监视事件的触发,进而改变状态,更新界面显示,所示通常情况下我们需要把流程控制类和线程类合并到一起来处理,此时我们就需要继承自 View 类并且实现 Runnable 接口。同时需要定义一个整型变量来记录当前的游戏状态, onDraw 方法控制整个游戏的绘图,而在线程中使用 postInvalidate 方法来更新界面的显示,然后在 onKeyDown、onKeyUp、onTouchEvent 等方法中处理外部事件的介入,从而改变界面的绘制,具体实现参见代码清单 14-3。

代码清单 14-3 第 14 章\GameEngine\src\com\yarin\android\GameEngine\GameControl.java

```
public class GameControl extends View implements Runnable
{
    //游戏状态
    public int mIGameStatus = -1;
    //是否开启线程
    public boolean mBLoop = false;
    public GameControl(Context context)
    {
        super(context);
    }
    //初始化游戏
    public void initGame()
    {
        mBLoop = true;
        mIGameStatus = GameDefinition.Game_Logo;

        Thread t = new Thread(this);
        t.start();
    }
    //绘制游戏界面
    protected void onDraw(Canvas canvas)
    {
        switch ( mIGameStatus )
        {
            case GameDefinition.Game_Logo:
                //显示 logo
                break;
        }
    }
}
```

```

        case GameDefinition.Game_MainMenu:
            //显示主菜单
            break;
        case GameDefinition.Game_Help:
            //显示帮助
            break;
        default:
            break;
    }
}
//线程控制
public void run()
{
    while (mBLoop)
    {
        try
        {
            Thread.sleep(500);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        ;
        postInvalidate(); // 刷新屏幕
    }
}
/* 按键按下 */
boolean onKeyDown(int keyCode)
{
    switch (mIGameStatus)
    {
        case GameDefinition.Game_Logo:
            //处理 logo 状态的按键事件
            break;
        case GameDefinition.Game_MainMenu:
            //处理主菜单状态的按键事件
            break;
        case GameDefinition.Game_Help:
            ///处理帮助状态的按键事件
            break;
        default:
            break;
    }
    return true;
}
/* 按键弹起 */
boolean onKeyUp(int keyCode)
{
    return true;
}
/* 触笔事件 */
public boolean onTouchEvent(MotionEvent event) {
    int iAction = event.getAction();
    //根据获得的不同事件进行处理
    if ( iAction == MotionEvent.ACTION_CANCEL )
    {

```

```

    }
    else if (iAction == MotionEvent.ACTION_DOWN)
    {
    }
    else if( iAction == MotionEvent.ACTION_MOVE )
    {
    }
    int x = (int) event.getX();
    int y = (int) event.getY();
    switch (mIGameStatus)
    {
        case GameDefinition.Game_Logo:
            //处理 logo 状态的触笔事件
            break;
        case GameDefinition.Game_MainMenu:
            //处理主菜单状态的触笔事件
            break;
        case GameDefinition.Game_Help:
            ///处理帮助状态的触笔事件
            break;
        default:
            break;
    }
    return true;
}
}

```

14.4.3 游戏对象与对象管理

在引擎中，我们将游戏的任何一个元素都抽象为对象，这些对象可以使用其他几个模块的元素，先来看看我们所抽象的游戏对象类（GameObject）的实现，如代码清单 14-4 所示。

代码清单 14-4 第 14 章\GameEngine\src\com\yarin\android\GameEngine\GameObject.java

```

public abstract class GameObject
{
    //对象的 ID
    private String id=null;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public abstract void loadProperties(Vector v);
    public String toString(){
        return ("id="+id);
    }
}

```

在这个游戏对象类中，我们为每个对象定义了一个 ID，以便更好地区分每个对象，因为每个对象都会有很多不同的属性，我们将这些属性全部放置到一个 Vector 中，在使用的时候再读取出具体的数据，所以我们定义了一个抽象方法 loadProperties 来装载游戏对象的各个属性。

我们都知道，任何一个游戏中都会存在很多这样的游戏对象，为了能够方便地管理和使用这些对象，我们将所有的对象存储到一个队列中来统一管理，下面看看我们实现的游戏对象管理类，如

代码清单 14-5 所示。

代码清单 14-5 第 14 章\GameEngine\src\com\yarin\android\GameEngine\GameObjectQueue.java

//对象的一个队列

```
public class GameObjectQueue extends Hashtable
{
    private String id=null;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public Object find(String gameIdId){
        if (this.containsKey(gameIdId)){
            return this.get(gameIdId);
        }
        else{
            return null;
        }
    }
    public Object[] list(){
        Object[] result=new Object[this.size()];
        Enumeration enumer=this.elements();
        int i=0;
        while(enumer.hasMoreElements()){
            result[i++]=enumer.nextElement();
        }
        return result;
    }
}
```

从这里可以看出我们继承了 Hashtable 类，我们可以通过 find 方法来查询指定 ID 的对象，并且设置这些对象的 ID。同时我们还可以通过 list 方法来获得所有对象的列表。这样就让我们可以将众多的对象管理得有条不紊。

14.4.4 图形引擎

图形引擎是游戏引擎中一个最重要的模块，包含的内容也非常丰富，包括：地图（场景）、摄像机、动画、主角、NPC、道具等，如图 14-6 所示。

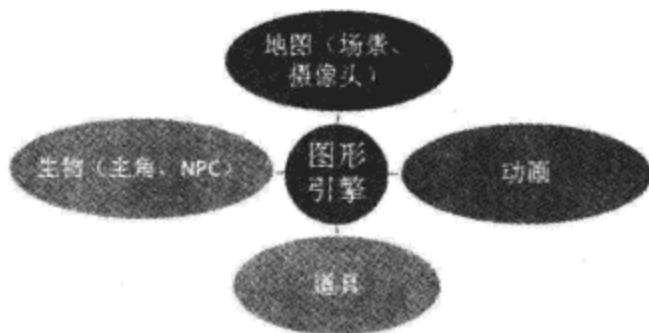


图 14-6 图形引擎结构

首先我们来分析图形引擎中的地图管理，这里我们对于地图的管理使用了 MIDP2.0 中 Game 包中的图层、图层管理器、精灵来实现，使得游戏开发更加简单，大家可以参考本书第 11 章游戏

开发实例中 Game 包的使用。由于游戏引擎的代码量比较大,大家可以参考本书所附代码:第14章\GameEngine。这里我们来看一个简单的地图层,如代码清单14-6所示。

代码清单 14-6 第14章\GameEngine\src\com\yarin\android\GameEngine\Screen\SimpleLayer.java

```
//简单地图层: 只显示一屏大小
public class SimpleLayer extends GameObject
{
    //行走路径类型图层
    public static int WALKARENA=1;
    //非行走路径类型图层
    public static int NO_WALKARENA=2;
    //地图数据
    private int[] mapData=null;
    //瓷砖的宽度
    private int tileWidth=0;
    //瓷砖的高度
    private int tileHeight=0;
    //瓷砖的列数
    private int tileCols=0;
    //瓷砖的行数
    private int tileRows=0;
    //图层
    private TiledLayer layer=null;
    //图层种类: 主角的行走路径、非行走路径
    private int type=0;
    //图片 URL
    private String imgURL=null;

    public SimpleLayer(){
        super();
    }
    /**
     * 将字符串转换为数组
     * @param s 被转换的字符串
     * @return 转换后的整型数组
     */
    private int[] StringToIntArray(String s){
        //首先去掉字符串中的格式化字符
        s=StringExtension.removeToken(s,new String[]{"\t"," ","\r","\n"});
        Object[] objArr=StringExtension.split(new StringBuffer(s),",",StringExtension.
            INTEGER_ARRAY,false);
        return StringExtension.objectArrayBatchToIntArray(objArr);
    }
    public void loadProperties(Vector v){
        this.setId((String)v.elementAt(0));
        this.tileWidth=Integer.parseInt((String)v.elementAt(1));
        this.tileHeight=Integer.parseInt((String)v.elementAt(2));
        this.tileCols=Integer.parseInt((String)v.elementAt(3));
        this.tileRows=Integer.parseInt((String)v.elementAt(4));
        this.type=Integer.parseInt((String)v.elementAt(5));
        this.imgURL=(String)v.elementAt(6);
        this.mapData=StringToIntArray((String)v.elementAt(7));
    }
    public TiledLayer getLayer(){
        return layer;
    }
}
```



```

    }
    public int[] getMapData() {
        return mapData;
    }
    public int getTileCols() {
        return tileCols;
    }
    public int getTileHeight() {
        return tileHeight;
    }
    public int getTileRows() {
        return tileRows;
    }
    public int getTileWidth() {
        return tileWidth;
    }
    public int getType() {
        return type;
    }
    public void setLayer(TiledLayer layer) {
        this.layer = layer;
    }
    public String getImgURL() {
        return imgURL;
    }
    public String toString(){
        return "id="+super.getId()
            +" tileWidth="+this.tileWidth
            +" tileHeight="+this.tileHeight
            +" tileCols="+this.tileCols
            +" tileRows="+this.tileRows
            +" type="+this.type
            +" imgURL="+this.imgURL
            +" mapData Size="+this.mapData.length;
    }
}

```

从代码清单 14-6 中我们看出，图层也是游戏对象的范围，所以同样继承了 `GameObject` 类，在这里我们实现了 `loadProperties` 类装载了地图的一些数据，比如：地图的宽度、地图的高度、图片资源、地图数组等。当然这只是一个图层，一个完整的地图可能由多个图层组成，下面我们来看一个简单的地图，如代码清单 14-7 所示。

代码清单 14-7 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\ SimpleMap.java

```

//简单地图：由主角的行走路径和非行走路径组成，还有 NPC
public class SimpleMap extends GameObject
{
    //图层集合(图层的顺序按照数组的顺序)
    private GameObjectQueue layerSet=null;
    //此图层中的 NPC
    private GameObjectQueue npcSet=null;
    //本地图与前后地图的连接器，一个 map 至少可以有一个
    private GameObjectQueue mapLink=null;
    //地图的宽度
    private int width=0;
    //地图的高度
    private int height=0;
}

```

```

//地图名字
private String name=null;

public String getName() {
    return name;
}
public SimpleMap(){
    super();
}
public void loadProperties(Vector v){
    this.setId((String)v.elementAt(0));
    this.name=(String)v.elementAt(1);
    this.width=Integer.parseInt((String)v.elementAt(2));
    this.height=Integer.parseInt((String)v.elementAt(3));
}
public GameObjectQueue getLayerSet() {
    return layerSet;
}
public GameObjectQueue getMapLink() {
    return mapLink;
}
public GameObjectQueue getNpcSet() {
    return npcSet;
}
public int getHeight() {
    return height;
}
public int getWidth() {
    return width;
}
public void setLayerSet(GameObjectQueue layerSet) {
    this.layerSet = layerSet;
}
public void setMapLink(GameObjectQueue mapLink) {
    this.mapLink = mapLink;
}
public void setNpcSet(GameObjectQueue npcSet) {
    this.npcSet = npcSet;
}
public String toString(){
    return "id="+super.toString()
    +" name="+this.name
    +" width="+this.width
    +" height="+this.height
    +" layerSet size="+this.layerSet.size()
    +" MapLink size="+this.mapLink.size()
    +" NpcSet size="+this.npcSet.size();
}
}

```

该地图类的实现同样很简单，就是将地图的名字、宽度、高度、图层、相连接的地图、NPC进行了整合，使其成为一个整体，构成一个完整的地图，在使用时就会更方便。关于地图的内容我们就简单地介绍到这里，大家一定要参考着完整的代码进行理解。下面我们来实现角色动画部分，如代码清单 14-8 所示。

代码清单 14-8 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\animation\Animator.java

//动画角色类

```

public class Animator extends Sprite
{
    private Calculagraph cal=null;

    public Animator(Bitmap img,
        int frameWidth,int frameHeight,
        int loopTime){
        super(img,frameWidth,frameHeight);
        cal=new Calculagraph(loopTime);
    }
    public Animator(Sprite s,int loopTime){
        super(s);
        cal=new Calculagraph(loopTime);
    }
    /**
     * 播放动画
     */
    public void PlayAnimation(){
        if (cal.getLoopTime()>0){
            //如果超时，则重新计时并播放下一 Frame
            if (cal.isTimeout()){
                cal.reset();
                this.nextFrame();
            }
            //否则继续计时
            else{
                cal.calculate();
            }
        }
    }
    /**
     * 停止播放动画
     */
    public void StopAnimation(){
        cal.reset();
    }
    /**
     * 刷新动画的位置
     * @param x 阿尔法坐标系的 x 轴位置
     * @param y 阿尔法坐标系的 y 轴位置
     */
    public void flushPosition(int x,int y){
        setRefPixelPosition(x,y);
    }
}

```

该类整体上很简单，但是一个重要的问题就是它继承自 `Sprite` 类，只需要将动画的资源文件传递给 `Sprite` 类，设置动画播放的位置，然后在播放函数中不停地调用 `nextFrame`（下一帧），这样就可以将动画按照我们的要求完整地播放出来。

下面我们来看一个很重要的、在游戏中必不可少的模块，即实体对象，主要包括：主角、NPC、怪物、道具等，我们都知道这些对象都具有一些共同的特征，因此我们可以按照如图 14-7 所示的结构来设计这些对象。

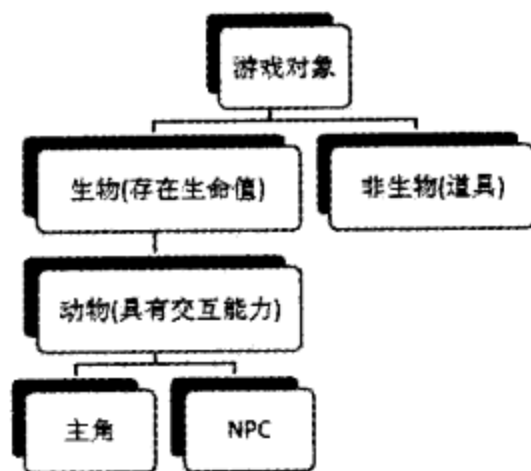


图 14-7 生物结构

从图 14-7 我们可以看出，最高层次是生物，即带有生命的动植物，所以我们首先就需要抽象一个生物的接口，该接口中我们主要定义了生物生命的最大、最小极限，以及生命减少或增加，如代码清单 14-9 所示。

代码清单 14-9 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\elements\biology\Biology.

```

java
package com.yarin.android.GameEngine.Screen.elements.biology;
//生物接口：定义了具有生命力的动/植物行为
public interface Biology
{
    //最大生命力
    public static final int MAXLIFE=100;
    //最小生命力
    public static final int MINLIFE=1;
    /**
     * 由于受到伤害或者其他因素减少主角的生命力，如果生命值小于 MINLIFE，则表明主角已死亡。
     * @param detaLife 生命力减少量
     * @return 返回主角的生命力
     */
    public int decreaseLife(int detaLife);
    /**
     * 由于使用物品或其他原因，主角的生命力增加
     * @param detaLife 生命力增加量
     * @return 主角的生命力
     */
    public int increaseLife(int detaLife);
}
  
```

当然，生物的概念很大，在游戏中（无论是主角还是 NPC），它们都具有一种可以区分其他生物的能力，那就是它们都具有交互能力，所以我们就将这一类生物封装到一个类中，很显然该类需要继承自生物类（Biology）接口，如代码清单 14-10 所示。

代码清单 14-10 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\elements\biology\

```

Animal.java
//实现 Biology 接口的动物类
public class Animal extends GameObject implements Biology
{
    //名称
    private String name=null;
  
```

```

//生命值
private int life=0;
//攻击值
private int attack=0;
//防御值
private int defence=0;
//主角是否生存
private boolean alive=true;
//图片/动画 URL
private String imgURL=null;
//脸部图片 URL
private String faceURL=null;
//事件对列: 当主角与 NPC 发生碰撞时, 使用事件 ID 调用事件对列中的某个事件。
private EventQueue eventQueue=null;
//坐标
private Coordinates co=null;
//运动方向和速度
private Movement movement=null;
//动画对象
private Animator ani=null;

public Animal(){
    super();
}

public void loadProperties(Vector v){
    this.setId((String)v.elementAt(0));
    this.name=(String)v.elementAt(1);
    this.life=Integer.parseInt((String)v.elementAt(2));
    this.attack=Integer.parseInt((String)v.elementAt(3));
    this.defence=Integer.parseInt((String)v.elementAt(4));
    this.imgURL=(String)v.elementAt(5);
    this.faceURL=(String)v.elementAt(6);
    int col=Integer.parseInt((String)v.elementAt(7));
    int row=Integer.parseInt((String)v.elementAt(8));
    this.co=new Coordinates(col,row);
    int stepSpeed=Integer.parseInt((String)v.elementAt(9));
    int moveDirection=Integer.parseInt((String)v.elementAt(10));
    this.movement=new Movement(stepSpeed,moveDirection);
    this.alive=true;
    int animationLoopTime=Integer.parseInt((String)v.elementAt(11));
    int frameWidth=Integer.parseInt((String)v.elementAt(12));
    int frameHeight=Integer.parseInt((String)v.elementAt(13));
    try{
        Bitmap img=BitmapFactory.decodeResource(GameActivity.mContext.
            getResources(), Integer.parseInt(this.imgURL));
        this.ani=new Animator(img,frameWidth,frameHeight,animationLoopTime);
        this.ani.setRefPixelPosition(col*frameWidth,row*frameHeight);
    }
    catch(Exception ex){}
}

//修改名字
public void changeName(String name){
    this.name=name;
}

//返回动物的名字
public String getName(){
    return this.name;
}

```

```

}
//修改动物的图片/动画 URL
public void changeImgURL(String imgURL){
    this.imgURL=imgURL;
}
//返回图片 URL
public String getImgURL(){
    return this.imgURL;
}
/**
 * 由于受到伤害或者其他因素减少动物的生命力, 如果生命值小于 MINLIFE, 则表明动物已死亡.
 * @param detaLife 生命力减少量
 * @return 返回动物的生命力
 */
public int decreaseLife(int detaLife){
    if (alive){
        if (life>MINLIFE){
            life-=detaLife;
            if (life<MINLIFE){
                alive=false;
            }
        }
    }
    return life;
}
/**
 * 由于使用物品或其他原因, 动物的生命力增加
 * @param detaLife 生命力增加量
 * @return 动物的生命力
 */
public int increaseLife(int detaLife){
    if (alive){
        if ((life+detaLife)<=MAXLIFE){
            life+=detaLife;
        }
        else{
            life=MAXLIFE;
        }
    }
    return life;
}
//判断动物是否生存
public boolean isAlive(){
    return alive;
}
//获得运动速度和方向
public Movement getMovement() {
    return movement;
}
//设置运动速度和方向
public void setMovement(Movement movement) {
    this.movement = movement;
}
//获得动物当前的坐标
public Coordinates getCoordinates() {
    return co;
}

```

```

    public int getAttack() {
        return attack;
    }
    public int increaseAttack(int detaAttack){
        this.attack+=detaAttack;
        return this.attack;
    }
    public int getDefence() {
        return defence;
    }
    public int increaseDefence(int detaDefence){
        this.defence+=detaDefence;
        return this.defence;
    }
    public EventQueue getEventQueue() {
        return eventQueue;
    }
    public String getFaceURL() {
        return faceURL;
    }
    public void setCoordinate(Coordinates co) {
        this.co = co;
    }
    public Animator getAnimator() {
        return ani;
    }
    public void setEventQueue(EventQueue eventQueue) {
        this.eventQueue = eventQueue;
    }
    public int getLife(){
        return this.life;
    }
    public String toString(){
        return super.toString()
            +" name="+name+" life="+life
            +" attack="+attack+" defence="+defence
            +" imgURL="+imgURL+" faceURL="+faceURL
            +" col="+co.getX()+" row="+co.getY()
            +" speed="+movement.getStepSpeed()+" direction="+movement.getMoveDirection()
            +" eventQueue="+eventQueue+" animator="+ani;
    }
}

```

很明显，动物都具有生命、攻击力、防御力、图片资源、动画、位置等属性，所以该类将动物的这些属性进行封装，方法都很简单，就是修改和读取这些属性。

我们都知道，在游戏中，主角和 NPC 肯定不一样，主角需要玩家来控制，而 NPC 则是由程序控制，所以形成了图 14-7 中的第 3 层，包括了主角和 NPC，在这两个类中主要实现了行走、运动等功能。值得注意的一点是，主角需要更具游戏状态的不同进行不同的设置，而 NPC 有很多类型，我们需要对每个 NPC 进行类型的设置和区分。首先我们来看看主角类如何实现，如代码清单 14-11 所示。

代码清单 14-11 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\elements\biology\Actor.java

```

public class Actor extends Animal
{
    //道具箱
    private PropertyManager propBox=null;

```



```

//游戏模式
private GameModel gameModel=null;
//帧切换数组,对应着主角的上、下、左、右四个方向的帧图片
private int[] frameSwitchSequence=null;

public Actor(){
    super();
    gameModel=new GameModel();
    gameModel.setModelType(GameModel.FREEMOVE_MODEL);
}
//重载父类的loadProperties,读取帧切换序列
public void loadProperties(Vector v){
    super.loadProperties(v);
    Object[] objArray=StringExtension.split(new StringBuffer((String)v.elementAt(14)),
        ",",StringExtension.INTEGER_ARRAY,false);
    if (objArray!=null){
        frameSwitchSequence=StringExtension.objectArrayBatchToIntArray(objArray);
    }
    else{
        frameSwitchSequence=null;
    }
}
//获得主角的道具箱
public PropertyManager getPropertyBox(){
    return propBox;
}
//设置主角的道具箱
public void setPropertyBox(PropertyManager propBox){
    this.propBox=propBox;
}
//装备道具,从道具箱中注销
public void equipProperty(Property prop){
    switch(prop.getType()){
        case Property.MEDICINE_PROP:
            increaseLife(prop.getLifeEffect());
            break;
        case Property.ATTACK_WEAPON_PROP:
            increaseAttack(prop.getAttackEffect());
            break;
        case Property.DEFENCE_WEAPON_PROP:
            increaseDefence(prop.getDefenceEffect());
    }
    propBox.unregisterProperty(prop.getId());
}
//返回当前的游戏模式
public GameModel getGameModel(){
    return gameModel;
}
//根据运动的方向和每步速度移动,每次移动一个主角的动画帧的位置
public void move(){
    int x=this.getAnimator().getX();
    int y=this.getAnimator().getY();
    //移动动画位置,每次移动一个主角的动画帧的位置
    switch(this.getMovement().getMoveDirection()){
        case Movement.LEFT_MOVE:
            x-=this.getMovement().getStepSpeed();
            getAnimator().setFrame(frameSwitchSequence[2]);

```

```

        getAnimator().flushPosition(x,y);
        break;
    case Movement.RIGHT_MOVE:
        x+=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[3]);
        getAnimator().flushPosition(x,y);
        break;
    case Movement.UP_MOVE:
        y-=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[0]);
        getAnimator().flushPosition(x,y);
        break;
    case Movement.DOWN_MOVE:
        y+=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[1]);
        getAnimator().flushPosition(x,y);
        break;
    }
}

//根据运动的方向和每步速度反向移动,用于做前面取消移动效果的目的
public void undoMove(){
    int x=this.getAnimator().getX();
    int y=this.getAnimator().getY();
    //移动动画位置
    switch(this.getMovement().getMoveDirection()){
    case Movement.LEFT_MOVE:
        x+=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[2]);
        getAnimator().flushPosition(x,y);
        break;
    case Movement.RIGHT_MOVE:
        x-=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[3]);
        getAnimator().flushPosition(x,y);
        break;
    case Movement.UP_MOVE:
        y+=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[0]);
        getAnimator().flushPosition(x,y);
        break;
    case Movement.DOWN_MOVE:
        y-=this.getMovement().getStepSpeed();
        getAnimator().setFrame(frameSwitchSequence[1]);
        getAnimator().flushPosition(x,y);
        break;
    }
}

public String toString(){
    if (frameSwitchSequence!=null){
        StringBuffer tmp=new StringBuffer();
        for(int i=0;i<frameSwitchSequence.length;i++){
            if (i<frameSwitchSequence.length-1){
                tmp.append(frameSwitchSequence[i]+",");
            }
            else{
                tmp.append(frameSwitchSequence[i]);
            }
        }
    }
}

```

```

        return super.toString()+" frameSwitichSequence="+tmp.toString();
    }
    else{
        return super.toString();
    }
}
}

```

NPC 类和主角类很类似，这里就不再给出代码了，大家可以参考“com.yarin.android.GameEngine.Screen.elements.biology.NPC”类。最后介绍道具类的实现了，道具不外乎就是情节类（帮助主角完成剧情）、恢复类（比如：增加生命值）、功能类（实现游戏的某些功能）等类型。当然，道具的来源主要包括：购买、拾取、交易等；道具的属性则比较多，比如：名称、价格、效果、功能、时间等。下面我们来看看道具类的实现，如代码清单 14-12 所示。

代码清单 14-12 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\elements\Property\Property.java

```

//道具类
public class Property extends GameObject
{
    //治疗类道具:用于恢复主角生命力
    public static final int MEDICINE_PROP=1;
    //攻击武器类道具:增加主角的攻击力
    public static final int ATTACK_WEAPON_PROP=2;
    //防御武器类道具:增加主角的防御力
    public static final int DEFENCE_WEAPON_PROP=3;
    //情节类道具: 作为用于过关的条件, 如钥匙等
    public static final int SCENARIO_PROP=4;

    //道具名称
    private String name=null;
    //道具的描述
    private String description=null;
    //道具买进价格(玩家买进)
    private int buyPrice=0;
    //道具卖出价格(玩家卖出)
    private int salePrice=0;
    //治疗效果(对生命值的影响)
    private int lifeEffect=0;
    //攻击力增加效果(对攻击值的影响)
    private int attackEffect=0;
    //防御力增加效果(对防御值的影响)
    private int defenceEffect=0;
    //耐用次数, 如药品只能用一次, 而小刀能用许多次
    private int useTimes=0;
    //道具类型
    private int type=0;

    public Property(){
        super();
    }

    public void loadProperties(Vector v){
        this.setId((String)v.elementAt(0));
        this.name=((String)v.elementAt(1));
        this.description=((String)v.elementAt(2));
    }
}

```

```

        this.buyPrice=(Integer.parseInt((String)v.elementAt(3)));
        this.salePrice=(Integer.parseInt((String)v.elementAt(4)));
        this.lifeEffect=(Integer.parseInt((String)v.elementAt(5)));
        this.attackEffect=(Integer.parseInt((String)v.elementAt(6)));
        this.defenceEffect=(Integer.parseInt((String)v.elementAt(7)));
        this.useTimes=(Integer.parseInt((String)v.elementAt(8)));
        this.type=(Integer.parseInt((String)v.elementAt(9)));
    }
    public int getBuyPrice() {
        return buyPrice;
    }
    public String getDescription() {
        return description;
    }
    public int getLifeEffect() {
        return lifeEffect;
    }
    public String getName() {
        return name;
    }
    public int getSalePrice() {
        return salePrice;
    }
    public int getUseTimes() {
        return useTimes;
    }
    public int getAttackEffect() {
        return attackEffect;
    }
    public int getDefenceEffect() {
        return defenceEffect;
    }
    public int getType() {
        return type;
    }
    public String toString(){
        return super.toString()
            +" name="+name+" description="+description
            +" buyPrice="+buyPrice+" salePrice="+salePrice
            +" lifeEffect="+lifeEffect+" attackEffect="+attackEffect
            +" defenceEffect="+defenceEffect+" useTimes="+useTimes
            +" type="+type;
    }
}

```

由于任何一个游戏都会存在很多道具，为了方便管理我们还是将所有的道具放置到一个队列中。这个队列就主要包括道具的添加、获得、注销等功能，如代码清单 14-13 所示。

代码清单 14-13 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Screen\elements\Property\PropertyManager.java

```

//道具管理：装载一定数量的道具
public class PropertyManager extends GameObjectQueue
{
    public PropertyManager(){
        super();
    }
    //尝试将道具放入道具箱
    public void putIntoBox(Property prop){

```

```

        this.put(prop.getId(),prop);
    }
    //取出道具
    public Property takeFromBox(String propID){
        try{
            return (Property)this.get(propID);
        }
        catch(Exception ex){
            return null;
        }
    }
    //注销道具(当使用完道具或者丢弃道具时使用)
    public boolean unregisterProperty(String propID){
        try{
            this.remove(propID);
            return true;
        }
        catch(Exception ex){
            return false;
        }
    }
    //返回道具数组列表
    public Property[] getPropertyList(){
        Property[] prop=new Property[this.size()];
        Enumeration enu=this.elements();
        int i=0;
        while(enu.hasMoreElements()){
            prop[i++]=(Property)enu.nextElement();
        }
        return prop;
    }
}

```

14.4.5 物理引擎

物理引擎通过为刚性物体赋予真实的物理属性的方式来计算运动、旋转和碰撞反映。没必要为每个游戏使用物理引擎——简单的“牛顿”物理（比如加速和减速）也可以在一定程度上通过编程或编写脚本来实现。然而，当游戏需要比较复杂的物体碰撞、滚动、滑动或者弹跳的时候（比如赛车类游戏或者保龄球游戏），通过编程的方法就比较困难了。物理引擎使用对象属性（动量、扭矩或者弹性）来模拟刚体行为，这不仅可以得到更加真实的结果，对于开发人员来说也比编写行为脚本更加容易掌握。好的物理引擎允许有复杂的机械装置，像轮子、气缸、铰链等。有些也支持非刚性体的物理属性，比如流体。物理引擎可以从另外的厂商购买，而一些游戏开发系统具备完整的物理引擎。但是要注意，虽然有的系统在其特性列表中说它们有物理引擎，但其实是一些简单的加速和碰撞检测属性而已。

说了这么多，那么物理引擎究竟能够实现一些什么样的效果呢？

□ 粒子效果

流体运动更多的则是展现水从水管内喷出及水冲击到物体后物体的表现。比如木箱被冲翻时，那些被冲翻的木箱翻倒的方向每次都各不相同；又比如在一个 NVIDIA Logo 形状的玻璃容器中，用鼠标来控制玻璃容器的位置，让容器里的液体流动，这种流动的效果和现实

中的效果已经相当接近了。

□ 流体效果

在 NVIDIA 展示的演示画面中, 还有一个则是“食人花”的场景, 讲述的是一朵巨大的食人花 (也可以说是花形状的怪兽), 玩家用激光去攻击食人花, 然后这朵“花”就会有相应的反应。这种柔软的身躯扭动起来非常像现实生活中看到的鼻涕虫之类的虫在蠕动, 令人感觉相当恶心。

□ 软体效果

至于在关节和布料方面的应用, 在文章的开始处已经提到了, 美女们在 T 型台上用夸张的姿势行走, 虽然动作夸张, 却丝毫没有虚假的成分在里面, 这在传统的 3D 游戏里是很难做到的。而身上的裙子也跟着关节的移动而摆动。

可以看出, 所谓物理效果, 都是在游戏中模仿现实中真实物理世界的运动方式。在游戏中, 大家甚至能感受到箱子、石头、布料以及那些恶心的无脊椎动物的触感是怎样的。游戏之所以能实现如此多的动态效果, 都要归功于物理运算。

关于这些效果的实现, 大家可以参考目前主流的开源物理引擎 Havok、NovodeX、Bullet、ODE 等。另外的物理引擎还包括了一些不可见的实现, 比如重力环境模拟、碰撞检测、游戏 AI 等。本节我们主要讲述一些物理计算上的内容, 比如一个简单的边界检查方法的实现, 如代码清单 14-14 所示。

代码清单 14-14 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Material\Measure.java

```
package com.yarin.android.GameEngine.Material;
//边界测量: 测量物体是否出边界
public class Measure
{
    //测量类型: 在矩形中检测
    public static final int RECTANGLE_MEASURE=1;
    /**
     * 判断是否出边界
     * @param x 物体的 x 坐标
     * @param y 物体的 y 坐标
     * @param border 边界对象
     * @param type 测量类型
     * @param superposition 判断边界时是否包含与边界重合的情况
     * @return 是否出边界
     */
    public static boolean isOutOfBorder(int x,int y,Border border,int type,boolean
    superposition){
        boolean result=false;

        switch(type){
            case RECTANGLE_MEASURE:
                //如果包含与边界重合的情况
                if (superposition){
                    if ((x>border.getMaxX())||
                        (x<border.getMinX())||
                        (y>border.getMaxY())||
                        (y<border.getMinY())){
                        result=true;
                    }
                }
                else{
                    result=false;
                }
            default:
                result=false;
        }
    }
}
```

```

    }
    }
    else{
        if ((x>=border.getMaxX())||
            (x<=border.getMinX())||
            (y>=border.getMaxY())||
            (y<=border.getMinY())){
            result=true;
        }
        else{
            result=false;
        }
    }
    break;
}
return result;
}
}

```

在代码清单 14-14 中，我们只是简单地介绍了如何检测是否出边界，本节内容主要用来告诉大家物理引擎应该包含的功能，目前已经有高手将 Box2d（2D 物理引擎）移植到了 Android 平台，大家可以参考，但是物理引擎根据计算的复杂程度会影响游戏的效率，所以建议大家考虑将 Box2d 通过 NDK（Android 原生开发，参考本书扩展篇）来实现，这样效率会得到很大的提高。当然物理引擎一般要视游戏功能而使用，这里我们介绍的是一个通用的游戏引擎框架，关于物理引擎我们就介绍到这里，更多内容大家可以参考“com.yarin.android.GameEngine.Materia”包的内容。

14.4.6 事件模块

事件模块是游戏引擎中必不可少的一部分，任何游戏都必须有事件的触发才能持续下去。事件可以分为外部事件（比如：键盘、触笔）和内部事件（游戏流程中触发）两类，外部事件由 Android 系统内部提供的方法来处理，这里我们主要讨论内部事件在游戏引擎中的处理。

既然是事件，那么肯定就有事件的类型、发起者、被发起者、事件参数等属性，因此我们在实现这个事件类的时候就需要考虑到这些因素。为了方便大家理解，我们将事件分为主角对话事件和主角作战事件（完整的引擎会考虑其他很多的事件），值得注意的是事件的参数，因为不同的事件所需参数的复杂程度不同，所以在传递事件参数时一定要考虑清楚。下面我们看一个事件处理类的例子，如代码清单 14-15 所示。

代码清单 14-15 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Events\Event.java

```

//事件类:对于对话类事件，一个事件对应着一个消息队列 MessageQueue
public class Event extends GameObject
{
    //对话类型事件：当需要主角和 NPC 对话时触发此事件
    public static final int TALK_EVENT=1;
    //战斗类型事件：当需要主角和 NPC 作战时触发此事件
    public static final int FIGHT_EVENT=2;
    //事件发起者
    private String invoker=null;
    //事件响应者
    private String responder=null;
    //事件类型
}

```



```

    private int type=0;
    //参数
    private String parameter=null;
    public Event(){
        super();
    }
    public void loadProperties(Vector attrValueSet){
        this.setId((String)attrValueSet.elementAt(0));
        this.invoker=(String)attrValueSet.elementAt(1);
        this.responser=(String)attrValueSet.elementAt(2);
        this.type=Integer.parseInt((String)attrValueSet.elementAt(3));

        this.parameter=(String)attrValueSet.elementAt(4);
    }
    public String getInvoker() {
        return invoker;
    }
    public String getResponser() {
        return responser;
    }
    public int getType() {
        return type;
    }
    public String getParameter() {
        return parameter;
    }
    public String toString() {
        return super.toString()
            +" invoker="+invoker
            +" responser="+responser
            +" type="+type
            +" parameter="+parameter;
    }
}

```

当然，为了方便管理，我们同样需要将所有的事件都放置到一个队列中去，具体实现大家可以参见“com.yarin.android.GameEngine.Events.EventQueue”类，对于事件的传递，我们可以使用消息机制。当一个事件被触发时，我们就发送一个消息给事件的响应者，当事件的响应者收到消息之后，根据消息的内容来判断，并处理相应的事件。但是由于那都是一些内部的消息，所以不需要网络来传送，因此要实现该消息类也很简单，如代码清单 14-16 所示。

代码清单 14-16 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Events\ Message.java

```

//消息类
public class Message extends GameObject
{
    private String msgContent=null;
    public Message(){
        super();
    }
    public Message(String msgID,String msgContent){
        super();
        this.msgContent=msgContent;
        this.setId(msgID);
    }
    public void loadProperties(Vector valueSet){
        this.msgContent=(String)valueSet.elementAt(0);
        this.setId((String)valueSet.elementAt(1));
    }
}

```

```

    }
    public String getMsgContent() {
        return msgContent;
    }
    public String toString(){
        return super.toString()
            +" msgContent="+msgContent;
    }
}

```

另外，我们同样可以实现一个管理消息的类，管理类和其他模块实现一样，大家可以参考“com.yarin.android.GameEngine.Events.MessageQueue”类，实现了这些类，我们对于事件的处理就会很简单，让游戏引擎也发挥了自己的作用。关于事件处理模块我们就介绍到这里，下面我们将学习游戏引擎中的工具模块。

14.4.7 工具模块

工具模块的内容比较散乱，可以随时根据自己的需要来添加，具体工具依游戏平台和游戏类型而定，这里我们实现了坐标类、计时器、随机数类、文件操作类、字符串扩展类。可以看出这都是一些需要经常使用的工具，只是将其封装成一个完整的类，让我们使用起来更加方便，下面我们来看看计时器类的实现，如代码清单 14-17 所示。

代码清单 14-17 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Util\Calculagraph.java

```

package com.yarin.android.GameEngine.Util;
//计时器类
public class Calculagraph
{
    //循环间隔（以 ms 为单位）
    private int loopTime=0;
    //是否是第一次计时
    private boolean isFirstTime=true;
    //开始时间
    private long startTime=0;
    //运行时间
    private long runTime=0;

    public Calculagraph(int loopTime){
        this.loopTime=loopTime;
        isFirstTime=true;
        runTime=0;
        startTime=0;
    }
    //计时
    public void calculate(){
        if(isFirstTime){
            startTime=System.currentTimeMillis();
            isFirstTime=false;
        }
        else{
            runTime=System.currentTimeMillis();
        }
    }
    //是否超时

```

```

    public boolean isTimeout(){
        return ((runTime-startTime)>loopTime);
    }
    //重置计时器
    public void reset(){
        startTime=0;
        runTime=0;
        isFirstTime=true;
    }
    //得到循环时间
    public int getLoopTime() {
        return loopTime;
    }
}

```

从代码清单 14-17 可以看出，这些工具类的实现都很简单，但是使用起来却变得更加简单，而且也更加完善。当然，其他工具类的实现在这里就不一一列举了，大家可以参考“com.yarin.android.GameEngine.Util”包。

14.4.8 脚本引擎、音效模块、网络模块

本节我们将分析游戏引擎中最后 3 个模块，它们分别是：脚本引擎、音效模块、网络模块，我们首先来分析脚本引擎的实现。

脚本也是游戏开发中很重要的部分，脚本可以处理的事情也很多，比如：用脚本技术来处理游戏剧情。要学习脚本引擎，首先需要明白脚本解释器的概念。

1. 脚本解释器

脚本解释器是这个系统的核心，首先是确定设计目标，脚本语言本身和 BASIC 语言类似，包括关键字及语句执行形式，这是一个面向过程的语言，同时支持某些面向对象的特性。

在设计一个脚本引擎时需要考虑脚本语言的基本结构、数据类型、表达式求值、变量有效范围、基本控制语句（自顶向下执行，循环，分支）、过程调用（也称方法调用）、出错处理、执行效率及内存使用、装载及预处理。

2. 脚本语言基本结构

脚本语言的基本结构由一个主程序和多个子程序构成，引擎本身提供一些基本的过程（方法）支持，以及可以二次开发扩充其过程库。主过程是脚本中的第一个过程，子过程紧随其后。

（1）数据类型

数据类型的设计，因为是在 Java 平台上实现，所以要尽量简化设计，支持 int 型数字、String 型字符串、boolean 类型以及数组。并且，每种数据在脚本引擎层面都是以对象的方式存在。

引擎用 java.lang.Integer 对象表示数值类型的数据，数值本身占 4 个字节，其数值范围同 Java 中的 int 型一样，取值范围是-2147483648~2147483647，可以进行加减乘除的基本操作。字符串数据以 java.lang.String 表示，初期版本无法使用双引号字符等。可以进行加操作，即连接两个字符串。布尔型数据用 java.lang.Boolean 对象表示，在逻辑表达式中使用。数组在 Java 中，本身就是一个对象，只不过是个特殊的对象。数组的数据类型仅限于数值及字符串型，且是一个方阵，超过二维的数组可以有不等长子数组成员。可以动态定义数组长度，不限制其数组维数。

(2) 表达式求值

表达式求值可以说是整个脚本引擎中最复杂的一件工作，也是最影响效率的脚本处理器。表达式分为数值表达式、字符串表达式和逻辑表达式。

(3) 变量有效范围

变量的有效范围可以参考 BASIC，其中主程序中定义的变量是全局变量，在主过程及子过程中都可以使用，变量的定义就是在主过程的任意地方进行一个赋值操作即可，且第一次赋值就定义。

(4) 控制语句

控制程序运行的主要方式是通过行号来进行处理的，即当脚本载入内存后，会把脚本存放在一个 `String[]` 数组中。

(5) 过程

过程调用的代码解释器是脚本引擎的核心，所有的代码都在这个解释器中执行，主过程的执行方式和子过程的执行方式类似。只不过，主过程没有过程标识符而已，在这个执行体中，要考虑的有变量的有效范围、参数的传递和代码的判断。

(6) 出错处理

为使效率更高，脚本引擎对错误类型的探测不是很细，最低要求是指出在某行出错，只有当某些较常见的错误发生时，可以详细提示错误类型，比如变量未找到、方法未找到、数据类型错误、过程调用参数个数错误、参数数据类型错误、代码未装载等。

(7) 执行效率及内存使用

为了提高效率，可以预存已分解的指令行，而不是每次执行时再去解析，或者通过编译方式，把脚本进行预编译，而不是通过解析字符串的方式来执行命令等。通过一些优化方式可以达到提高效率的目的，相应地，如果预存已解析指令方式，则会占用更多内存，因为每条指令存一个字符串对象比存数十个对象要节约许多内存。如果使用预编译的方式，无疑会增加引擎的复杂度，整个引擎的尺寸也会更大，可以视不同的应用进行不同方式的优化，从而达到预期的效果。

(8) 装载及预处理

装载时，按字节流方式读入资源文件，或者通过网络等其他方式读入。读入后，以 `0x0a`, `0x0d` 为分隔符，把脚本分行存入 `String[]` 中。预处理主要是找到小于 `0x20` 的字符，替成空格，去掉行首行尾的空格。另外一项较重要的预处理工作是，在整个脚本中，把所有子过程的起始行号找到，并存入一个散列表 `subAddr` 中，以便于在以后调用过程时，不需要在整个脚本中去查找过程的起始行号。

通过上面的介绍，我们明白了实现一个脚本引擎时要注意的地方，而本节我们实现了一个简单的 XML 播放器和一个自定义文件类型的读取处理。由于代码比较多，在这里我们就不列出了，大家可以参考“`com.yarin.android.GameEngine.Script`”包的内容。下面我们将介绍音效模块的实现。

游戏音效本身就比较简单，不外乎就是背景音效和动作音效。要在 Android 平台播放一个音乐很简单，这里我们就不再介绍如何播放了，只是将游戏中的音效进行一个综合的处理，如代码清单 14-18 所示。

代码清单 14-18 第 14 章\GameEngine\src\com\yarin\android\GameEngine\Music\Music.java

```
//音乐类: 定义音乐资源 ID、资源 URL、播放方式、播放循环次数
public class Music extends GameObject
{
    //播放方式: 无限循环
```

```
public static final int INFINITE_LOOP=1;
//播放方式: 有限次数播放
public static final int FINITE_LOOP=2;
//资源 URL
private String resURL=null;
//音乐类型
private String musicType=null;
//播放方式
private int playModel=0;
//循环次数, 在有限次数播放时有效
private int loopNumber=0;
//音乐播放器
private MediaPlayer musicPlayer=null;
//当前播放次数
private int currentPlayTimes=0;
public Music(){
    super();
    currentPlayTimes=0;
}
public void loadProperties(Vector v){
    this.setId((String)v.elementAt(0));
    this.resURL=(String)v.elementAt(1);
    this.musicType=(String)v.elementAt(2);
    this.playModel=Integer.parseInt((String)v.elementAt(3));
    this.loopNumber=Integer.parseInt((String)v.elementAt(4));
    try{
musicPlayer=MediaPlayer.create(GameActivity.mContext, Integer.parseInt(resURL));
musicPlayer.prepare();
    }
    catch(Exception ex){
        ex.printStackTrace();
    }
}
//是否播放完毕, 对于有限播放次数的播放方式有效
public boolean isPlayEnd(){
    if (playModel==FINITE_LOOP){
        return (currentPlayTimes>=loopNumber);
    }
    else{
        return false;
    }
}
//增加播放次数
public void increasePlayTimes(){
    currentPlayTimes++;
}
public int getLoopNumber() {
    return loopNumber;
}
public int getPlayModel() {
    return playModel;
}
public String getResURL() {
    return resURL;
}
public String getMusicType() {
    return musicType;
}
}
```

```

public MediaPlayer getMediaPlayer() {
    return musicPlayer;
}
public String toString(){
    return super.toString()
    +" resURL="+this.resURL
    +" musicType="+this.musicType
    +" playModel="+this.playModel
    +" loopNumber="+this.loopNumber;
}
}

```

为了方便管理，我们还是会所有的音效资源放置到一个队列中，请参见“com.yarin.android.GameEngine.Music.Musician”类。

如果游戏需要进行网络连接，那么就需要引擎能支持网络，这就是我们这个引擎中最后一个模块——网络模块，网络连接主要分为：互联网和局域网，互联网要考虑网络的传输协议，而局域网的链接方式就很多，比如：蓝牙、红外等。

网络模块主要用于客户端和服务端的连接和数据传递，如图 14-8 为服务器与客户端通信的详细流程图。

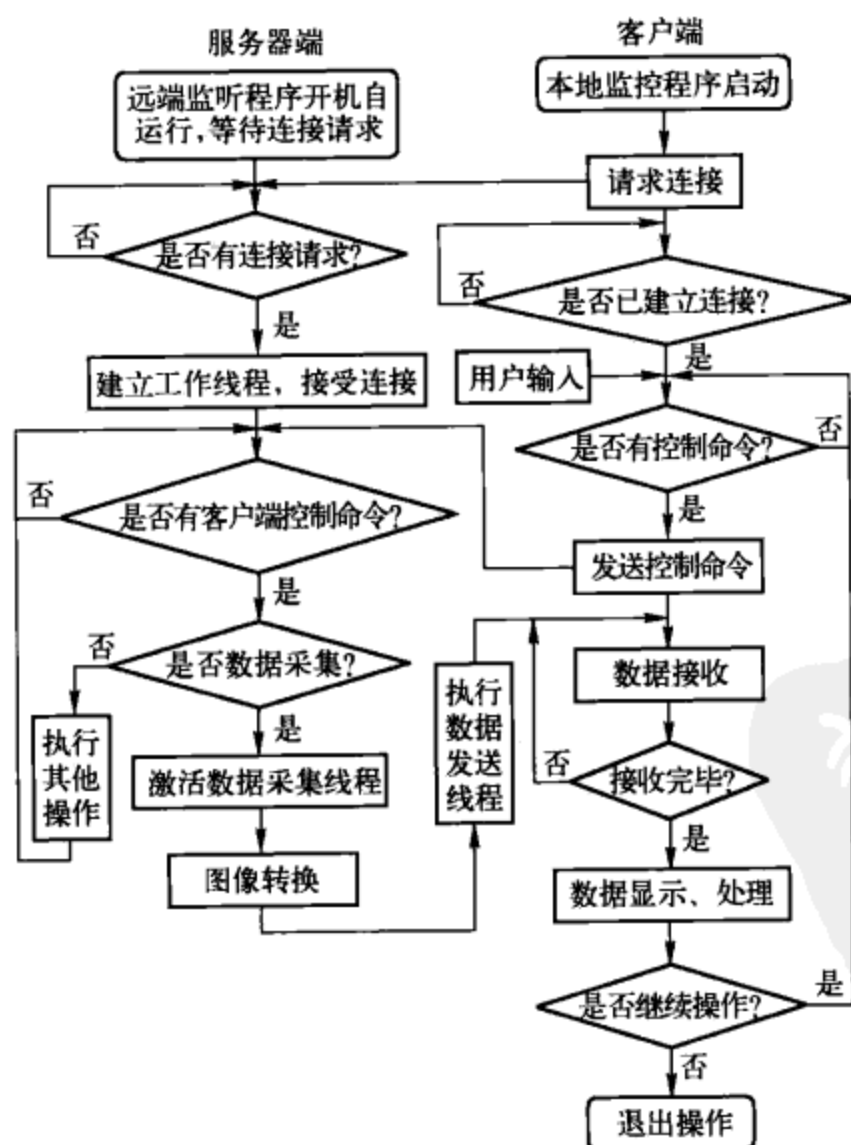


图 14-8 服务器与客户端通信流程图

因此无论使用什么方式或者什么协议联网，我们可以按照如图 14-8 所示的流程图来实现这个引擎的网络模块，因此网络模块的实现就变得更加简单，这也是唯一留给大家实现的一个模块，相信大家经过学习都能实现，期待大家都能实现自己的游戏引擎。

14.5 小结

到这里，本章的内容就结束了，下面回顾一下我们本章的内容。首先我们对游戏引擎进行了介绍，了解了什么是游戏引擎以及常用的游戏引擎，然后对游戏引擎的框架结构进行了分析，明白了游戏引擎的原理和引擎中最常用的框架，进而设计了一个简单的游戏引擎，最后在设计的基础上简单地实现了该引擎的各个模块，使大家对游戏引擎的开发流程有了一个详细的了解。

游戏引擎开发是一个比较热门的职业，它强调学科间的相互合作，诸如几何学、色彩理论、物理学、计算机科学等，主要集中在视觉科学领域。因此大家要开发游戏引擎，需要对这些学科进行更深入地研究。一些公司甚至以开发引擎为乐趣和技术追求，而不是用来卖钱。比如 Crystal Space 就是一个比较知名的免费开放源代码多平台游戏引擎。笔者很欣赏这种开源的精神。



看到这一章的内容，也许有人会说这是多余，但是我们还是要在这一章详细地讲解，因为很多人能够很轻松地写出代码，但是代码的运行效率却不是很理想，那么我们如何解决这些问题呢？这就是本章将要学习的优化技术。我们都知道 Android 目前除了 NDK 以外，开发应用所使用的都是 Java 语言，以前也听到很多人说 Java 语言的效率本身就很低，对于这样的说法，笔者不能否认，但是也不赞同，Java 属于高级语言，给编程人员提供了极大的方便，那么它肯定就没有一些低级语言效率高，那么为什么我们不使用汇编语言来编写应用程序呢？总还是有一定的原因吧！从目前的情况来看，我们还得使用 Java 来开发应用，大部分程序员想必也不想使用汇编等语言的开发吧！既然我们改变不了 Java 语言，那么我们就尽量养成好的代码编写习惯，掌握优化技术，提高程序的性能。大家不要把优化想象得很难，优化其实非常有趣，让我们带着兴奋的心情开始本章的学习吧。

15.1 优化的基本知识

为了减少 Java 平台的硬件环境对游戏性能的限制，在开发阶段必须对游戏的程序进行改良化设计和采用优化算法。传统手机软件是由手机开发厂商固化在手机中的。在 SUN 公司推出 J2ME 后，各大手机厂商很快就推出了支持 J2ME 的手机，它可运行由第三方提供的基于 J2ME 开发的软件，使手机的扩展功能得到极大增强。当然现在也有很多类似于 J2ME 的解决方案，它们可能没有使用 Java 作为开发语言，但是不管用什么语言进行开发，所有的优秀代码都会展示出共有的经典品质：简练、可读性强、模块化、层次性、设计良好、高效、优雅、清晰等。本节我们将从这些特征出发，来学习如何书写出优秀的代码，以及一些常用的编程规范。

15.1.1 如何书写出优秀代码

本节我们将对优秀代码所应具备的上述品质一一进行讲解。

1. 简练

这意味着能用五行代码解决的问题，绝不用十行代码；这也意味着，必须花费更多的精力来保证代码的简练，而不是生产令人费解的代码；这还意味着，你所厌恶的事情，是连篇累牍的开放性代码和函数。简练——即结构好、可执行、设计优秀——可以让你的代码更易于理解，也减少了错误的发生。

2. 可读性强

正如其字，这说明你的代码应该更能让其他人看懂。要做到可读性强，就得多写注释，符合大

众习惯，不要自作聪明地命名变量，比如，用 a、b、c 等。

3. 模块化

这意味着你得像宇宙的构成一样来开发程序。世界是由分子构成的，而这些分子又由原子、电子、核子、夸克和连线（如果你相信的话）组成。类似地，优秀的程序也是由小模块建成大的系统，而这些小模块又由更小的部分来组成。你完全可以只使用移动、插入和删除这三个简单的应用，来编写一个文本编辑器。就像原子的构成方式一样，软件的组件也应该具有复用性。

4. 层次性

程序得像蛋糕内部一样层次分明。应用运行在框架之上，框架运行在系统之上，而系统又运行在硬件之上。就算是应用程序的内部，也应该有层次。高级层访问低级层，而且事件正好相反（向下访问，向上返回），低级层不应该知道高级层在干些什么。事件/反馈的基本特性就是向上发出无指向性通知。如果你的文档直接访问了框架，那可就有危险了。模块和层次由 API 来定义，这样就限定了各自的运行范围。因此，设计就变得尤为重要了。

5. 设计良好

在开发程序之前，要先花些时间来设计你的程序，因为思考的代价要小于调试。优秀的开发准则就是，花一半时间来设计。你得写一份功能说明（这个程序是用来干嘛的）和一个深层蓝图，而 API 的功能也应该整理成文。

6. 高效

程序不但要运行快，而且要节省资源，它不能牵扯到文档、数据连接等。程序只做它该做的事，不能胡乱地装载和撤销线程。在运行层上，你可以在测试之后才优化程序；但是在高级层，你必须先计划好执行过程。

7. 优雅

优雅和漂亮是一个概念，它难以描述却显而易见。优雅综合了简练、高效和简明等概念，同时还能让人感受到高贵的气质。举个例子，优雅就是使用表格，或者是用递归来编写程序。

```
{  
    return n==0 ? 1 : n * factorial(n-1);  
}
```

8. 清晰

清晰是优秀代码的基本，也是其他要素的品质保证。相对于普通机械，计算机建立更为复杂的系统的能力要强得多。对于编程来说，最基础的挑战就是减少复杂度。简单、可读性强、模块化、层次、设计良好、高效、优雅，都是久经考验达成简练代码的方法，这些可以有效地减少代码的复杂度。

清晰的代码，良好的设计，明确的目标，你必须真正地了解自己在每个步骤所要做的事情，否则就会一事无成。差劲的程序，往往不是因为编程技术水平的问题，而是因为没有一个明确的目标。所以，设计是开发的关键，能让自己保持目标。如果不能写出设计计划，不能向其他人解释开发目标，其实说明你连自己在做什么都不知道。

15.1.2 编程规范

Java 程序员能够依据 Java 编程规范养成良好的编程习惯，是编写良好 Java 程序的先决条件。

对于 Java 编程规范首先要准确理解。例如，每行声明一个局部变量，不仅仅要知道是 Java 编程规范的要求，更重要的是要理解这样增加了代码的易懂性。理解好 Java 编程规范是发挥规范作用的基础。理解规范中每一个原则仅仅是开始，进一步需要相信这些规范是编码的最好方法，并且在编程过程中坚持应用。另外，应该在编程过程中坚持遵循这些规范，培养成习惯，这样才能够保证开发出干净代码，使开发和维护工作都更简单。从一开始就写干净的代码，可以在程序开发过程中以及程序维护阶段不断受益。下面是一些常见的编程规范，大家可以参考一下：

1. 基本要求

- ☐ 程序结构清晰，简单易懂，单个函数的程序行数不得超过 100 行。
- ☐ 打算干什么，要简单，直接了当，代码精简，避免垃圾程序。
- ☐ 尽量使用标准库函数和公共函数。
- ☐ 不要随意定义全局变量，尽量使用局部变量。
- ☐ 使用括号以避免二义性。

2. 可读性要求

- ☐ 可读性第一，效率第二。
 - ☐ 保持注释与代码完全一致。
 - ☐ 每个源程序文件，都有文件头说明，说明规格见规范。
 - ☐ 每个函数，都有函数头说明，说明规格见规范。
 - ☐ 主要变量定义或引用时，注释能反映其含义。
 - ☐ 常量定义有相应说明。
 - ☐ 处理过程的每个阶段都有相关注释说明。
 - ☐ 在典型算法前都有注释。
 - ☐ 利用缩进来显示程序的逻辑结构，缩进量一致并以 Tab 键为单位，定义 Tab 为 6 个字节。
 - ☐ 循环、分支层次不要超过五层。
- 注释可以与语句在同一行，也可以在上行。
- ☐ 空行和空白字符也是一种特殊注释。
 - ☐ 一目了然的语句不加注释。
 - ☐ 注释的作用范围可以为：定义、引用、条件分支以及一段代码。
 - ☐ 注释行数（不包括程序头和函数头说明部分）应占总行数的 1/5~1/3。

3. 结构化要求

- ☐ 禁止出现两条等价的支路。
- ☐ 用 CASE 实现多路分支。
- ☐ 避免从循环引出多个出口。
- ☐ 函数只有一个出口。
- ☐ 不使用条件赋值语句。
- ☐ 避免不必要的分支。
- ☐ 不要轻易用条件分支去替换逻辑表达式。

4. 正确性与容错性要求

- ☐ 程序首先是正确，其次是优美。

- ❑ 无法证明你的程序没有错误，因此在编写完一段程序后，应先回头检查。
- ❑ 改一个错误时可能产生新的错误，因此在修改前首先考虑对其他程序的影响。
- ❑ 所有变量在调用前必须被初始化。
- ❑ 对所有的用户输入，必须进行合法性检查。
- ❑ 不要比较浮点数的相等，如： $10.0 * 0.1 = 1.0$ 。
- ❑ 程序与环境或状态发生关系时，必须主动去处理发生的意外事件，如文件能否逻辑锁定、打印机是否联机等。
- ❑ 单元测试也是编程的一部份，提交联调测试的程序必须通过单元测试。

5. 可重用性要求

- ❑ 重复使用的完成相对独立功能的算法或代码应抽象为公共控件或类。
- ❑ 公共控件或类应考虑面向对象思想，减少外界联系，考虑独立性或封装性。

15.2 程序性能测试

要想对程序进行优化，肯定需要先对程序的性能进行测试，找出需要优化的部分，进行对症下药地优化。本节将讲述如何利用 Java 语言本身提供的方法在开发中进行性能测试，找到系统瓶颈，进而改进设计；并且在尽量不修改测试对象的情况下进行测试。

面向对象编程通过抽象继承采用模块化的思想来求解问题域，但是模块化不能很好地解决所有问题。有时，这些问题可能在多个模块中都出现（比如，日志模块），为了记录每个方法进入和离开时的信息，你不得不在每个方法里添加 `log("in some method")` 等信息。如何解决这类问题呢？将这些解决问题的功能点散落在多个模块中会使冗余增大，并且当多个功能点出现在一个模块中时，代码会变得很难维护。因此，AOP（Aspect Oriented Programming，面向切面编程）应运而生。如果说 OOP（Object Oriented Programming，面向对象编程）关注的是一个类的垂直结构，那么 AOP 就是从水平角度来看待问题的。

动态代理类可以在运行时实现若干接口，每一个动态代理类都有一个 `Invocation handler` 对象与之对应，这个对象实现了 `InvocationHandler` 接口，通过动态代理的接口对动态代理对象的方法调用会转而调用 `Invocation handler` 对象的 `invoke` 方法，通过动态代理实例、方法对象和参数对象可以执行调用并返回结果。

说到 AOP，大家首先会想到的是日志记录、权限检查和事务管理。是的，AOP 是解决这些问题的好办法。本文根据 AOP 的思想，通过动态代理来解决一类新的问题——性能测试（performance testing）。性能测试主要包括以下几个方面：

- ❑ 计算性能：可能是人们首先关心的，简单地说就是执行一段代码所用的时间。
- ❑ 内存消耗：程序运行所占用的内存大小。
- ❑ 启动时间：从你启动程序到程序正常运行的时间。
- ❑ 可伸缩性(scalability)：指应用程序如何应对增长的流量。说到可伸缩性的时候，我们通常指向上可伸缩（scaling up），以便应对更大的负载。可伸缩性经常等价于水平可伸缩性（horizontal scalability）即向上伸缩到服务器集群来提高吞吐量。我们也可以通过把应用转移到更强的服务器上来提高吞吐量。后者要简单得多，但显然并不能让应用更牢固，也只

能得到有限的提高。另一种选择是垂直伸缩 (vertical scaling) 即在每台服务器上运行多份服务。“垂直伸缩”这个术语被 Fowler 用来指“为单台服务器增加更多的计算能力”，比如添加额外的 CPU 或者内存。

- 用户察觉性能(Perceived Performance): 不是程序实际运行有多快, 而是用户感觉程序运行有多快。

15.2.1 计算性能测试

Java 为我们提供了 `System.currentTimeMillis()` 方法, 可以得到毫秒级的当前时间, 我们在以前的程序当中一定也写过类似的代码来计算执行某一段代码所消耗的时间。

```
long start=System.currentTimeMillis();
...
long end=System.currentTimeMillis();
System.out.println("time lasts "+(end-start)+"ms");
```

但是, 在每个方法里面都写上这么一段代码是一件很枯燥的事情, 我们通过 Java 的 `java.lang.reflect.Proxy` 和 `java.lang.reflect.InvocationHandler` 利用动态代理来很好地解决上面的问题。

我们要测试的例子是 `java.util.LinkedList` 和 `java.util.ArrayList` 的 `get(int index)` 方法, 显然 `ArrayList` 要比 `LinkedList` 高效, 因为前者是随机访问, 而后者需要顺序访问。具体实现参见本书所附代码: 第 15 章\Examples_05_01, 操作步骤如下:

首先我们创建一个接口。

```
package com.yarin.android.Examples_15_01;
public interface Testing
{
    public void testArrayList();
    public void testLinkedList();
}
```

然后我们创建测试对象实现这个接口 `TestingImpl`, 如代码清单 15-1 所示。

代码清单 15-1 第 15 章\Examples_15_01\src\com\yarin\android\Examples_15_01\TestingImpl.java

```
public class TestingImpl implements Testing
{
    private List link = new LinkedList();
    private List array = new ArrayList();
    public TestingImpl()
    {
        for (int i = 0; i < 10000; i++)
        {
            array.add(new Integer(i));
            link.add(new Integer(i));
        }
    }
    public void testArrayList()
    {
        for (int i = 0; i < 10000; i++)
            array.get(i);
    }
    public void testLinkedList()
    {
        for (int i = 0; i < 10000; i++)
            link.get(i);
    }
}
```



```

    }
}

```

接下来我们要做关键的一步——实现 `InvocationHandler` 接口，如代码清单 15-2 所示。

代码清单 15-2 第 15 章\Examples_15_01\src\com\yarin\android\Examples_15_01\Handler.java

```

public class Handler implements InvocationHandler
{
    private Object obj;
    public Handler(Object obj)
    {
        this.obj = obj;
    }
    public static Object newInstance(Object obj)
    {
        Object result = Proxy.newProxyInstance(obj.getClass().getClassLoader(), obj.
            getClass().getInterfaces(), new Handler(obj));
        return result;
    }
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
    {
        Object result;
        try
        {
            Log.i("Handler", "begin method " + method.getName());

            long start = System.currentTimeMillis();
            result = method.invoke(obj, args);
            long end = System.currentTimeMillis();

            Log.i("Handler", "the method " + method.getName() + " lasts " + (end
                - start) + "ms");
        }
        catch (InvocationTargetException e)
        {
            throw e.getTargetException();
        }
        catch (Exception e)
        {
            throw new RuntimeException("unexpected invocation exception: " +
                e.getMessage());
        }
        finally
        {
            Log.i("Handler", "end method " + method.getName());
        }
        return result;
    }
}

```

最后创建测试程序，在 `Activity01.java` 中加入如下代码：

```

try
{
    Testing testing = (Testing) Handler.newInstance(new TestingImpl());
    testing.testArrayList();
    testing.testLinkedList();
} catch (Exception e) { e.printStackTrace();}

```

运行程序后，在 DDMS 中可以看到如图 15-1 所示程序输出的 Log 信息。我们便可以很清楚地

看出每个方法执行的时间，从而选择效率更高的方法。

10-18 10:4	I	721	Handler	begin method testArrayList
10-18 10:4	I	721	Handler	the method testArrayList lasts 39ms
10-18 10:4	I	721	Handler	end method testArrayList
10-18 10:4	I	721	Handler	begin method testLinkedList
10-18 10:4	W	584	Activity	Launch timeout has expired, giving up w
10-18 10:4	W	584	Activity	Activity idle timeout for HistoryRecord
10-18 10:4	I	721	Handler	the method testLinkedList lasts 9902ms
10-18 10:4	I	721	Handler	end method testLinkedList

图 15-1 Examples_15_01 项目的 Log 信息

使用动态代理的好处是你不必修改原有代码 `FooImpl`，但是有一个缺点——如果你的类原来没有实现接口，你不得不写一个接口。

上面的例子演示了利用动态代理比较两个方法的执行时间，有时候通过一次简单的测试进行比较是片面的，因此可以进行多次执行测试对象，从而计算出最差、最好和平均性能。这样，我们才能“加快经常执行的程序的速度，尽量少调用速度慢的程序”。

15.2.2 内存消耗测试

当一个 Java 应用程序运行时，有很多需要消耗内存的因素存在，如对象、加载类、线程等。在这里只考虑程序中的对象所消耗的虚拟机堆空间，这样我们就可以利用 `Runtime` 类的 `freeMemory()` 和 `totalMemory()` 方法。下面我们将学习如何来检测内存的使用，具体实现参见本书所附代码：第 15 章\Examples_05_02，实现步骤如下：

同样地，我们先创建一个接口 `MemoConsumer`，如下代码所示。

```
package com.yarin.android.Examples_15_02;
public interface MemoConsumer
{
    public void creatArray();
    public void creatHashMap();
}
```

然后我们创建测试对象实现这个接口 `MemoConsumerImpl`，如代码清单 15-1 所示。比较一个长度为 1000 的 `ArrayList` 和 `HashMap` 所占空间的大小，接口、实现如代码清单 15-3 所示。

代码清单 15-3 第 15 章\Examples_15_02\src\com\yarin\android\Examples_15_02\Memo ConsumerImpl.java

```
public class MemoConsumerImpl implements MemoConsumer
{
    ArrayList arr = null;
    HashMap hash = null;
    public void creatArray()
    {
        arr = new ArrayList(1000);
    }
    public void creatHashMap()
    {
        hash = new HashMap(1000);
    }
}
```

为了方便起见，我们首先添加一个类计算当前的内存消耗，如代码清单 15-4 所示。

代码清单 15-4 第 15 章\Examples_15_02\src\com\yarin\android\Examples_15_02\Memory.java

```
package com.yarin.android.Examples_15_02;
public class Memory
{
    public static long used()
    {
        long total = Runtime.getRuntime().totalMemory();
        long free = Runtime.getRuntime().freeMemory();
        return (total - free);
    }
}
```

然后修改 Handler 类的 invoke() 方法, 如代码清单 15-5 所示。

代码清单 15-5 第 15 章\Examples_15_02\src\com\yarin\android\Examples_15_02\Handler.java

```
public class Handler implements InvocationHandler
{
    private Object obj;
    public Handler(Object obj)
    {
        this.obj = obj;
    }
    public static Object newInstance(Object obj)
    {
        Object result = Proxy.newProxyInstance(obj.getClass().getClassLoader(),
        obj.getClass().getInterfaces(), new Handler(obj));
        return (result);
    }
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
    {
        Object result;
        try
        {
            Log.i("Handler", "begin method " + method.getName());

            long start = Memory.used();
            result = method.invoke(obj, args);
            long end = Memory.used();

            Log.i("Handler", "memory increased by " + (end - start) + "bytes");
        }
        catch (InvocationTargetException e)
        {
            throw e.getTargetException();
        }
        catch (Exception e)
        {
            throw new RuntimeException("unexpected invocation exception: " + e.
            getMessage());
        }
        finally
        {
            Log.i("Handler", "end method " + method.getName());
        }
        return result;
    }
}
```

最后我们在 Activity01.java 中加入如下代码, 用来测试程序运行的内存情况。

```
MemoConsumer arrayMemo=(MemoConsumer)Handler.newInstance(new MemoConsumerImpl ());
arrayMemo.creatArray();
arrayMemo.creatHashMap();
```

运行程序，在 DDMS 中查看 Log 信息，如图 15-2 所示。

10-18 11:0	I	727	Handler	begin method creatArray
10-18 11:0	I	727	Handler	memory increased by 4056bytes
10-18 11:0	I	727	Handler	end method creatArray
10-18 11:0	I	727	Handler	begin method creatHashMap
10-18 11:0	I	727	Handler	memory increased by 4064bytes
10-18 11:0	I	727	Handler	end method creatHashMap

图 15-2 Examples_15_02 项目内存使用信息

AOP 通过分解关注点和 OOP 相得益彰，使程序更加简洁易懂，通过 Java 语言本身提供的动态代理帮助我们很容易分解关注点，取得了较好的效果。不过测试对象必须实现接口在一定程度上限制了动态代理的使用，可以借鉴 Spring 中使用的 CGLib 来为没有实现任何接口的类创建动态代理。当然检测程序性能的方法很多，比如 ADT 和 Eclipse 等都提供了测试的工具，大家可以根据自己项目的具体情况来选择测试所使用的方法。找到了程序运行效率低的原因之后，后面的章节我们就可以来更改这些低效率、耗内存的方法，从而提高程序的性能。

15.3 初级优化

我们都知道可供程序利用的资源（内存、CPU 时间、网络带宽等）是有限的，优化的目的就是让程序用尽可能少的资源完成预定的任务。优化通常包含两方面的内容：减小代码的体积，提高代码的运行效率。本文讨论的主要是如何提高代码的效率。

在 Java 程序中，性能问题的大部分原因并不在于 Java 语言，而是在于程序本身。养成好的代码编写习惯非常重要，比如正确地、巧妙地运用 `java.lang.String` 类和 `java.util.Vector` 类，它能够显著地提高程序的性能。下面我们就来具体地分析一下这方面的问题。

(1) 尽量指定类的 `final` 修饰符。带有 `final` 修饰符的类是不可派生的。在 Java 核心 API 中，有许多应用 `final` 的例子，例如 `java.lang.String`。为 `String` 类指定 `final` 防止了人们覆盖 `length()` 方法。另外，如果指定一个类为 `final`，则该类所有的方法都是 `final`。Java 编译器会寻找机会内联 (`inline`) 所有的 `final` 方法（这和具体的编译器实现有关）。此举能够使性能平均提高 50%。

(2) 尽量重用对象。特别是 `String` 对象的使用中，出现字符串连接情况时应用 `StringBuffer` 代替。由于系统不仅要花时间生成对象，以后可能还需花时间对这些对象进行垃圾回收和处理。因此，生成过多的对象将会给程序的性能带来很大的影响。

(3) 尽量使用局部变量。调用方法时传递的参数以及在调用中创建的临时变量都保存在栈 (`Stack`) 中，速度较快。其他变量，如静态变量、实例变量等，都在堆 (`Heap`) 中创建，速度较慢。

(4) 不要重复初始化变量。默认情况下，调用类的构造函数时，Java 会把变量初始化成确定的值，所有的对象被设置成 `null`，整数变量 (`byte`、`short`、`int`、`long`) 设置成 0，`float` 和 `double` 变量设置成 0.0，逻辑值设置成 `false`。当一个类从另一个类派生时，这一点尤其应该注意，因为用 `new` 关键词创建一个对象时，构造函数链中的所有构造函数都会被自动调用。

(5) 在 Java+Oracle 的应用系统开发中, Java 中内嵌的 SQL 语句尽量使用大写的形式, 以减轻 Oracle 解析器的解析负担。

(6) Java 编程过程中, 进行数据库连接、I/O 流操作时务必小心, 在使用完毕后, 及时关闭以释放资源。因为对这些大对象的操作会造成系统大的开销, 稍有不慎, 会导致严重的后果。

(7) 由于 JVM 的有其自身的 GC 机制, 不需要程序开发者的过多考虑, 从一定程度上减轻了开发者负担, 但同时也遗漏了隐患, 过分的创建对象会消耗系统的大量内存, 严重时会导致内存泄露, 因此, 保证过期对象的及时回收具有重要意义。JVM 回收垃圾的条件是: 对象不再被引用; 然而, JVM 的 GC 并非十分地机智, 即使对象满足了垃圾回收的条件也不一定会被立即回收。所以, 建议在对象使用完毕后, 手动设置成 null。

(8) 在使用同步机制时, 应尽量使用方法同步代替代码块同步。

(9) 尽量减少对变量的重复计算。

例如:

```
for(int i = 0; i < list.size; i++) {
    ...
}
```

应替换为:

```
for(int i = 0, int len = list.size(); i < len; i++) {
    ...
}
```

(10) 尽量采用 lazy loading 的策略, 在需要的时候才开始创建。

例如:

```
String str = "aaa";
if(i == 1) {
    list.add(str);
}
```

应替换为:

```
if(i == 1) {
    String str = "aaa";
    list.add(str);
}
```

(11) 慎用异常, 异常对性能不利。抛出异常首先要创建一个新的对象。Throwable 接口的构造函数调用名为 fillInStackTrace() 的本地方法, fillInStackTrace() 方法检查栈, 收集调用跟踪信息。只要有异常被抛出, VM 就必须调整调用栈, 因为在处理过程中创建了一个新的对象。异常只能用于错误处理, 不应该用来控制程序流程。

(12) 不要在循环中使用 Try/Catch 语句, 应将其放置在最外层。

(13) StringBuffer 的使用, StringBuffer 表示了可变的、可写的字符串。有三个构造方法。

```
StringBuffer (); //默认分配 16 个字符的空间
StringBuffer (int size); //分配 size 个字符的空间
StringBuffer (String str); //分配 16 个字符+str.length() 个字符空间
```

你可以通过 StringBuffer 的构造函数来设定它的初始化容量, 这样可以明显地提升性能。这里提到的构造函数是 StringBuffer(int length), length 参数表示当前的 StringBuffer 能保持的字符数量。

你也可以使用 `ensureCapacity(int minimumcapacity)` 方法在 `StringBuffer` 对象创建之后设置它的容量。首先我们看看 `StringBuffer` 的默认行为，然后再找出一条更好地提升性能的途径。

`StringBuffer` 在内部维护一个字符数组，当你使用缺省的构造函数来创建 `StringBuffer` 对象时，因为没有设置初始化字符长度，`StringBuffer` 的容量被初始化为 16 个字符，也就是说缺省容量就是 16 个字符。当 `StringBuffer` 达到最大容量时，它会将自身容量增加到当前的 2 倍再加 2，也就是 $(2 \times \text{旧值} + 2)$ 。如果你使用缺省值，初始化之后接着往里面追加字符，在你追加到第 16 个字符的时候，它会将容量增加到 34 $(2 \times 16 + 2)$ ，当追加到 34 个字符的时候，它就会将容量增加到 70 $(2 \times 34 + 2)$ 。无论何时，只要 `StringBuffer` 到达它的最大容量，它就不得不创建一个新的字符数组，然后重新将旧字符和新字符都复制一遍——这也太昂贵了点。所以，应给 `StringBuffer` 设置一个合理的初始化容量值，这样会带来立竿见影的性能增益。

`StringBuffer` 初始化过程的调整作用由此可见一斑。所以，使用一个合适的容量值来初始化 `StringBuffer` 永远都是一个最佳的建议。

(14) 合理的使用 Java 类 `java.util.Vector`。简单地说，一个 `Vector` 就是一个 `java.lang.Object` 实例的数组。`Vector` 与数组相似，它的元素可以通过整数形式的索引访问。但是，`Vector` 类型的对象在创建之后，对象的大小依元素的增加或者删除而定。请考虑下面这个向 `Vector` 加入元素的例子。

```
Object obj = new Object();
Vector v = new Vector(100000);
for(int I=0;
I<100000; I++) { v.add(0,obj); }
```

除非有绝对充足的理由要求每次都把新元素插入到 `Vector` 的前面，否则上面的代码对性能不利。在默认构造函数中，`Vector` 的初始存储能力是 10 个元素，如果新元素加入时的存储能力不足，则以后存储能力每次加倍。`Vector` 类就像 `StringBuffer` 类一样，每次扩展存储能力时，所有现有的元素都要复制到新的存储空间之中。下面的代码片段要比前面的例子快几个数量级：

```
Object obj = new Object();
Vector v = new Vector(100000);
for(int I=0; I<100000; I++) { v.add(obj); }
```

同样的规则也适用于 `Vector` 类的 `remove()` 方法。由于 `Vector` 中各个元素之间不能含有“空隙”，删除除最后一个元素之外的任意其他元素都导致被删除元素之后的元素向前移动。也就是说，从 `Vector` 删除最后一个元素要比删除第一个元素“开销”低好几倍。

假设要从前面的 `Vector` 删除所有元素，我们可以使用以下代码：

```
for(int I=0; I<100000; I++){
    v.remove(0);
}
```

但是，与下面的代码相比，前面的代码要慢几个数量级：

```
for(int I=0; I<100000; I++){
    v.remove(v.size()-1);
}
```

从 `Vector` 类型的对象 `v` 删除所有元素的最好方法是：

```
v.removeAllElements();
```

假设 `Vector` 类的对象 `v` 包含字符串“Hello”。考虑下面的代码，它要从这个 `Vector` 中删除“Hello”字符串：

```
String s = "Hello";
int i = v.indexOf(s);
if(I != -1) v.remove(s);
```

这些代码看起来没什么错误，但它同样对性能不利。在这段代码中，`indexOf()`方法对 `v` 进行顺序搜索寻找字符串“Hello”，`remove(s)`方法也要进行同样的顺序搜索。改进之后的版本是：

```
String s = "Hello";
int i = v.indexOf(s);
if(I != -1) v.remove(i);
```

这个版本中我们直接在 `remove()`方法中给出带删除元素的精确索引位置，从而避免了第二次搜索。一个更好的版本是：

```
String s = "Hello"; v.remove(s);
```

最后，我们再来看一个有关 `Vector` 类的代码片段：

```
for(int I=0; I++;I < v.length)
```

如果 `v` 包含 100 000 个元素，这个代码片段将调用 `v.size()`方法 100 000 次。虽然 `size` 方法是一个简单的方法，但它仍旧需要一次方法调用的开销，至少 JVM 需要为它配置以及清除栈环境。在这里，`for` 循环内部的代码不会以任何方式修改 `Vector` 类对象 `v` 的大小，因此上面的代码最好改写成下面这种形式：

```
int size = v.size();
for(int I=0; I++;I<size)
```

虽然这是一个简单的改动，但它仍旧赢得了性能。毕竟，每一个 CPU 周期都是宝贵的。

(15) 当复制大量数据时，使用 `System.arraycopy()`命令。

(16) 代码重构：增强代码的可读性。

例如：

```
public class ShopCart {
    private List carts ;
    public void add (Object item) {
        if(carts == null) {
            carts = new ArrayList();
        }
        carts.add(item);
    }
    public void remove(Object item) {
        if(carts.contains(item)) {
            carts.remove(item);
        }
    }
    public List getCarts() {
        //返回只读列表
        return Collections.unmodifiableList(carts);
    }
    //不推荐这种方式
    //this.getCarts().add(item);
}
```

(17) 不用 `new` 关键词创建类的实例。

用 `new` 关键词创建类的实例时，构造函数链中的所有构造函数都会被自动调用。但如果一个对象实现了 `Cloneable` 接口，我们可以调用它的 `clone()`方法。`clone()`方法不会调用任何类构造

函数。

在使用设计模式 (Design Pattern) 的场合, 如果用 Factory 模式创建对象, 则改用 clone() 方法创建新的对象实例非常简单。例如, 下面是 Factory 模式的一个典型实现。

```
public static Credit getNewCredit() {
    return new Credit();
}
```

改进后的代码使用 clone() 方法, 如下所示。

```
private static Credit BaseCredit = new Credit();
public static Credit getNewCredit() {
    return (Credit) BaseCredit.clone();
}
```

上面的思路对于数组处理同样很有用。

(18) 乘法和除法。

考虑下面的代码:

```
for (val = 0; val < 100000; val +=5) {
    alterX = val * 8; myResult = val * 2;
}
```

用移位操作替代乘法操作可以极大地提高性能。下面是修改后的代码。

```
for (val = 0; val < 100000; val += 5) {
    alterX = val << 3; myResult = val << 1;
}
```

修改后的代码不再做乘以 8 的操作, 而是改用等价的左移 3 位操作, 每左移 1 位相当于乘以 2。相应地, 右移 1 位操作相当于除以 2。值得一提的是, 虽然移位操作速度快, 但这可能使代码比较难于理解, 所以最好加上一些注释。

(19) 不要将数组声明为: public static final。

(20) 讨论 HashMap 的遍历效率。经常遇到对 HashMap 中的 key 和 value 值的遍历操作, 有如下两种方法:

```
Map<String, String[]> paraMap = new HashMap<String, String[]>();
//第一个循环
Set<String> appFieldDefIds = paraMap.keySet();
for (String appFieldDefId : appFieldDefIds) {
    String[] values = paraMap.get(appFieldDefId);
}
//第二个循环
for(Entry<String, String[]> entry : paraMap.entrySet()){
    String appFieldDefId = entry.getKey();
    String[] values = entry.getValue();
}
```

第一种实现的效率明显不如第二种实现。分析如下:

Set<String> appFieldDefIds = paraMap.keySet();

第一个是先将 HashMap 中取得 keySet 值, 第二个则是每次循环时都取值, 增加了 CPU 要处理的任务。代码如下:

```
public Set<K> keySet() {
    Set<K> ks = keySet;
    return (ks != null ? ks : (keySet = new KeySet()));
}
```



```
private class KeySet extends AbstractSet<K> {
    public Iterator<K> iterator() {
        return newKeyIterator();
    }
    public int size() {
        return size;
    }
    public boolean contains(Object o) {
        return containsKey(o);
    }
    public boolean remove(Object o) {
        return HashMap.this.removeEntryForKey(o) != null;
    }
    public void clear() {
        HashMap.this.clear();
    }
}
```

其实就是返回一个私有类 `KeySet`，它是从 `AbstractSet` 继承而来，实现了 `Set` 接口。再来看看 `for/in` 循环的语法。

```
for(declaration : expression)
    statement
```

在执行阶段被翻译成如下格式：

```
for(Iterator<E> #i = (expression).iterator(); #i.hasNext();) {
    declaration = #i.next();
    statement
}
```

因此，在第一个 `for` 语句 `for (String appFieldDefId : appFieldDefIds)` 中调用了 `HashMap.keySet().iterator()`，而这个方法调用了 `newKeyIterator()`。

```
Iterator<K> newKeyIterator() {
    return new KeyIterator();
}
private class KeyIterator extends HashIterator<K> {
    public K next() {
        return nextEntry().getKey();
    }
}
```

在第二个循环 `for(Entry<String, String[]> entry : paraMap.entrySet())` 中使用的 `Iterator` 是如下的一个内部类：

```
private class EntryIterator extends HashIterator<Map.Entry<K,V>> {
    public Map.Entry<K,V> next() {
        return nextEntry();
    }
}
```

此时第一个循环得到 `key`，第二个循环得到 `HashMap` 的 `Entry` 效率就是从循环里面体现出来的，第二个循环此时可以直接取 `key` 和 `value` 值，而第一个循环还是得再利用 `HashMap` 的 `get(Object key)` 来取 `value` 值。

现在看看 `HashMap` 的 `get(Object key)` 方法。

```
public V get(Object key) {
    Object k = maskNull(key);
    int hash = hash(k);
    int i = indexFor(hash, table.length);
```



```

Entry<K,V> e = table;
while (true) {
    if (e == null)
        return null;
    if (e.hash == hash && eq(k, e.key))
        return e.value;
    e = e.next;
}
}

```

其实就是再次利用散列值取出相应的 Entry 做比较得到结果，所以使用第一种循环相当于两次进入 HashMap 的 Entry 中，而第二个循环取得 Entry 的值之后直接取 key 和 value，效率比第一个循环高。按照 Map 的概念来看，也应该是用第二个循环好一点，它本来就是 key 和 value 的值对，将 key 和 value 分开操作在这里不是个好选择。

(21) array (数组) 和 ArrayList 的使用。array ([])：最高效；但是其容量固定且无法动态改变；ArrayList：容量可动态增长；但牺牲效率。

基于效率和类型检验，应尽可能使用 array，无法确定数组大小时，才使用 ArrayList。ArrayList 是 Array 的复杂版本，ArrayList 内部封装了一个 Object 类型的数组，从一般的意义来说，它和数组没有本质的差别，甚至于 ArrayList 的许多方法，如 Index、IndexOf、Contains、Sort 等都是在内部数组的基础上直接调用 Array 的对应方法。ArrayList 存入对象时，抛弃类型信息，所有对象屏蔽为 Object，编译时不检查类型，但是运行时会报错（JDK5 中加入了泛型的支持，已经可以在使用 ArrayList 时进行类型检查）。从这一点上看来，ArrayList 与数组的区别主要就是因为动态扩容的效率问题了。

(22) 尽量使用 HashMap 和 ArrayList，除非必要，否则不推荐使用 Hashtable 和 Vector，后者由于使用同步机制，而导致了性能的开销。

(23) StringBuffer 和 StringBuilder 的区别在于：java.lang.StringBuffer 线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但不能修改。StringBuilder 与该类相比，通常应该优先使用 java.lang.StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。为了获得更好的性能，在构造 StringBuffer 或 StringBuilder 时应尽可能指定它的容量。当然，如果你操作的字符串长度不超过 16 个字符时就不用了。相同情况下，使用 StringBuilder 比使用 StringBuffer 仅能获得 10%~15% 左右的性能提升，但却要冒多线程不安全的风险。而在现实的模块化编程中，负责某一模块的程序员不一定能清晰地判断该模块是否会放入多线程的环境中运行，因此，除非你能确定你的系统的瓶颈是在 StringBuffer 上，并且确定你的模块不会运行在多线程模式下，否则还是用 StringBuffer 吧。

15.4 高级优化

上一节我们从 Java 语言的角度，了解了一些初级的优化方式，而本节我们将从程序运行性能的角度出发，分析各种常用方案的不足，并给出了对象池技术、基本数据类型替换法、屏蔽函数计算三种能够节省资源开销和处理器时间以提高游戏运行性能优化策略。

目前被普遍采用的优化方案有：

- 优化循环，通过重新组织重复的子表达式来提高循环体的运行性能。
- 减少使用对象的数量来提高运行性能。
- 缩减网络传输数据来缩短等待时间等。

而本节将学习另外三种性能优化的策略：

- 采用对象池技术，提高对象的利用率。
- 局部使用基本数据类型代替对象，节省资源开销。
- 用简单的数值计算代替复杂的函数计算，节省处理器时间。

1. 采用对象池技术，提高对象的利用效率

Java 是面向对象的编程语言，创建和释放对象会占用相当大的资源，而在 Java 里不用对象在很多情况下又无法实现。本文提出一种对象池技术，将有效解决创建和释放对象带来的性能损失问题。

例如，游戏中敌机的处理方式：一种解决方案是游戏在载入关卡的时候，为每架敌机创建一个对象，随着游戏的行进，按照游戏进程显示不同的敌机。这种方案中创建对象的资源开销巨大，因此严重影响手机游戏的运行性能。虽然在敌机被击毁的时候可以将对应的对象设置为 `null` 并由 `System.gc()` 回收，但在下一关卡载入的时候还要重新创建相关的对象，增加了用户的等待时间。另一种方案是在游戏的进程中，根据需要动态创建敌机对象，被击毁后将对象设置为 `null` 并由 `System.gc()` 回收。这种方案虽然能减少游戏载入时间，但是频繁地创建和释放对象的资源开销使游戏变得不流畅，对于射击游戏这类对实时性要求很高的游戏而言，这一点是不可接受的。

从研究数据来看，游戏性能损耗主要源于创建和释放对象，而不创建对象又无法实现逻辑功能，因此要尽量避免对象的创建和释放。问题的重点就转到怎样有效利用已有的对象上。本文提出的对象池技术，就是根据需求先创建一定量的对象，在需要创建对象的时候从池中申请空闲对象，释放对象时把对象释放回池中，以有效避免由创建和释放对象带来的性能损失。

分析游戏需求发现同时显示的敌机数量最多不过 5 架，采用对象池技术可以先定义一个对象池，容量为同时显示的敌机的最大数量。

```
Enemy[5] enemy = new Enemy[5];
for (int i = 0; i < 5; i++) {
    enemy[i] = new Enemy();
}
```

在类 `Enemy` 里增加标志属性 `used` 和带参数的 `reset` 方法使对象可重置到初始状态，在载入游戏关卡的时候初始化对象池，在需要创建对象的时候从对象池获取一个未被使用的对象并使用 `reset` 方法初始化，需要释放对象的时候只需将标志位修改以供下次使用。与第一种解决方案相比，使用对象池减少了相当大一部分的资源开销；与第二种解决方案相比，使用对象池避免了频繁创建对象的额外资源开销。

2. 尽可能地使用基本数据类型代替对象

对象虽然屏蔽了细节实现，但是是以牺牲存储空间来实现的。使用基本数据类型则仅需要少量的存储空间。虽然对象在逻辑的实现上具有优势，但是在绝大多数情况下，使用基本数据类型比使用对象更高效，所以在局部不影响逻辑的情况下可以考虑用基本数据类型代替对象实现。

在游戏中飞机要发射子弹，基于面向对象的设计模式可将子弹抽象成 `Bullet` 类，然后定义代表

子弹的对象池，在发射子弹的时候，在对象池中查找可用对象并重置其位置为飞机所在的位置，即在每一个 gameloop 中将 Bullet 对象的纵坐标值减少一定数值（屏幕坐标系 Y 轴坐标为自上而下递增）。由于对象操作的效率低于对基本数据类型的操作，由此可以想到由基本数据类型来代替对象实现。代码如下：

```
int[][] bullet=new int[2][10]:    //假设同屏同时显示10颗子弹
for(int i=0; i<10; i++){
bullet[0][i]=999;                //对应Bullet对象的xposition属性
bullet[1][i]=999;                //对应Bullet对象的yposition属性
}
```

在每一个 gameloop 中：

```
while(run){
    for(int i=0; i<10; i++){
        if(bullet[1][i]==999){
            //,重置为发射子弹飞机当前的位置
            bullet[0][i]==myPlane.getXposition();
            bullet[1][i]==myPlane.getYposition();
        }else{
            //,当前使用的子弹让其纵坐标减少
            bullet[1][i]-=10;
        }
    }
}
```

3. 用简单的数值计算代替复杂的函数计算

在任何一款游戏里，都不可避免地要进行函数运算，而大量复杂的函数计算势必会占用大量的空间和处理器时间，进而影响游戏性能。减少函数计算复杂度或者减少复杂函数的调用次数甚至避免使用复杂函数计算，将有利于游戏性能的提高。与复杂函数运算相比，简单的数值计算无论从时间上还是空间上都有极大的优势。

在游戏中敌机也会发射子弹，不同于我机子弹，敌弹的运行轨迹可以朝着任意一个方向。计算敌弹运行轨迹常用的方法是使用三角函数，根据飞行方向和横坐标轴正方向的夹角来计算敌弹的位移量。虽然提供了三角函数的类库，但是在每个 gameloop 中计算每颗敌弹的运行轨迹势必会有大量的三角函数运算而导致性能损失，当子弹数目变大时这种影响更加明显。

由基本的数学原理可知，当角度一定时其对应的三角函数值也是一定的，又因为三角函数是周期函数，因此本文的优化算法是用 $0^\circ \sim 45^\circ$ 的函数值通过简单的四则运算来模拟任意函数值。实现方式如下。首先定义如下数组：

```
public static int iAngle[][]=new int[][]{
    //根据游戏需要定义不同的精度，可以从0连续定义到45°。
    //因为 SinX=Cos(90-X)，所以不必重复定义46°到90°的函数值。
    {0, 1000, 0},
    {5, 996, 87},
    {10, 985, 174},
    {40, 766, 643},
    {45, 707, 707}
}
```

无论子弹朝着哪个方向飞行，假设飞行速度一定，其横纵坐标位移量均为该方向对应的余弦值

和正弦值,所以可以重新定义敌机子弹数组,将位移量包含到数组中。本游戏设定每颗子弹的飞行方向始终是固定不变的,即不会出现弧型的飞行轨迹。

```
int[][] enemyBullet=new int[4][20];           //假设同屏同时显示 20 颗敌机子弹
int size=enemyBullet.length;
for(int i=0; i<size; i++) {
    enemyBullet[0][i]=999;           //对应敌机子弹对象的x坐标
    enemyBullet[1][i]=999;           //对应敌机子弹对象的y坐标
    enemyBullet[2][i]=999;           //对应敌机子弹对象的运行时的x坐标位移量
    enemyBullet[3][i]=999;           //对应敌机子弹对象的运行时的y坐标位移量
}
```

按照程序的需要,从数组 `iAngle` 中把特定的数值赋值到 `enemyBullet[2][i]` 和 `enemyBullet[3][i]`,而在每一个 `gameloop` 中只需要执行如下代码:

```
int size=enemyBullet.length;
for(int i=0; i<size; i++) {
    if(enemyBullet[0][i]!=999&&enemyBullet[1][i]!=999) {
        enemyBullet[0][i]+=enemyBullet[2][i];
        enemyBullet[1][i]+=enemyBullet[3][i];
    }
}
```

以上过程避免了每个 `gameloop` 中使用三角函数计算子弹的位移,为了保证足够的精度,这里相关函数值取 3 位小数,在处理时可以将横纵坐标值 `enemyBullet[0][i]` 和 `enemyBullet[1][i]` 定义成坐标值的 1000 倍,然后与坐标值 `enemyBullet[2][i]` 和 `enemyBullet[3][i]` 相加,显示的时候将 `enemyBullet[0][i]` 和 `enemyBullet[1][i]` 的值除以 1000 取整。

当前游戏性能的瓶颈在于有限的存储资源和偏弱的处理器速度,而游戏优化的关键之处在于节省资源开销、节省冗余计算和计算简化,所以牺牲部分设计上的结构化(即面向对象)换来性能的提升,在当前情况下仍然是非常有必要的。本文提出的优化方法为在较低的硬件条件下实现流畅的画面,提供了可行的解决方案。

15.5 Android 高效开发

毫无疑问,基于 Android 平台的设备一定是嵌入式设备。现代的手持设备不仅仅是一部电话那么简单,它还是一个小型的笔记本电脑,但是,即使是最快的最高端的手持设备也远远比不上一个中等性能的桌面机。

这就是为什么在编写 Android 程序时要时刻考虑执行的效率,这些系统不是想象中的那么快,并且你还要考虑它电池的续航能力。这就意味着没有多少剩余空间给你去浪费了,因此,在编写 Android 程序时,要尽可能地使代码优化,从而提高效率。

本节我们将介绍几种可以使 Android 程序运行得更加有效率的方法。对于如何判断一个系统是否合理,这里有两个基本的原则:

- ☐ 不要做不必要做的事情。
- ☐ 尽可能地节省内存的使用。

Android 的成功在于开发程序提供给用户的体验,然而用户体验的好坏又决定于你的代码是否能及时地响应而不至于慢得让人崩溃。因为所有的程序都会在同一设备上运行,所以我们要把它

们作为一个整体来考虑。本文就像你考驾照需要学习的交通规则一样：如果所有人遵守，事情就会很流畅；但当你不遵守时，你就会撞车。下面的所有方法都是基于这个原则的。

1. 尽可能避免创建对象 (Object)

对象的创建并不是没有代价的。一个带有线程分配池的 `generational` 的内存管理机制会使创建临时对象的代价减少，并不是分配内存总不如不分配好。

如果你在一个用户界面的循环中分配一个对象，你不得不强制地进行内存回收，那么就会使用户体验出现稍微“打嗝”的现象。因此，如果没有必要你就不应该创建对象实例。下面是一个有帮助的例子。

❑ 当从原始的输入数据中提取字符串时，试着从原始字符串返回一个子字符串，而不是创建一份复本。你将会创建一个新的字符串对象，但是它和你的原始数据共享数据空间。

❑ 如果你有一个返回字符串的方法，你应该知道无论如何返回的结果是 `StringBuffer`，改变你的函数的定义和执行，让函数直接返回而不是通过创建一个临时的对象。

一个比较激进的方法就是把一个多维数组分割成几个平行的一维数组：

❑ 一个 `Int` 类型的数组要比一个 `Integer` 类型的数组要好，但着同样也可以归纳于这样一个原则，两个 `Int` 类型的数组要比一个 `(int, int)` 对象数组的效率高得多。对于其他原始数据类型，这个原则同样适用。

❑ 如果你需要创建一个包含一系列 `Foo` 和 `Bar` 对象的容器 (`container`) 时，记住：两个平行的 `Foo[]` 和 `Bar[]` 要比一个 `(Foo, Bar)` 对象数组的效率高得多（这个例子也有一个例外，即当你设计其他代码的接口 `API` 时；在这种情况下，速度上的一点损失就不用考虑了。但是，在你的代码里面，你应该尽可能的编写高效代码）。

一般来说，我们应该尽可能地避免创建短期的临时对象。越少的对象创建意味着越少的垃圾回收，这会提高你程序的用户体验质量。

2. 使用自身方法

当处理字符串的时候，不要犹豫，尽可能多地使用诸如 `String.indexOf()`、`String.lastIndexOf()` 这样对象自身带有的方法。因为这些方法使用 C/C++ 来实现的，要比在一个 `Java` 循环中做同样的事情快 10~100 倍。

3. 使用虚拟优于使用接口

假设你有一个 `HashMap` 对象，你可以声明它是一个 `HashMap` 或者只是一个 `Map`：

```
Map myMap1 = new HashMap();
HashMap myMap2 = new HashMap();
```

哪一个更好呢？

一般来说，明智的做法是使用 `Map`，因为它能够允许你改变 `Map` 接口执行上面的任何东西，但是这种“明智”的方法只是适用于常规的编程，对于嵌入式系统并不适合。相对于通过具体的引用进行虚拟函数的调用，通过接口引用来调用会花费 2 倍以上的时间。

如果你选择使用一个 `HashMap`，因为它更适合于你的编程，那么使用 `Map` 会毫无价值。假定你有一个能重构代码的集成编码环境，那么调用 `Map` 没有什么用处，即使你不确定你的程序从哪开始。

4. 使用静态优于使用虚拟

如果你没有必要去访问对象的外部，那么就使你的方法成为静态方法。它会被更快地调用，因为它不需要一个虚拟函数导向表。这同时也是一个很好的实践，因为它告诉你如何区分方法的性质 (signature)，调用这个方法不会改变对象的状态。

5. 尽可能避免使用内在的 Get、Set 方法

像 C++ 编程语言，我们通常会使用 Get 方法 (例如 `i=getCount()`) 去取代直接访问这个属性 (`i=mCount`)。这在 C++ 编程里面是一个很好的习惯，因为编译器会把访问方式设置为 `Inline`，并且如果想约束或调试属性访问，你只需要在任何时候添加一些代码。

在 Android 编程中，虚方法的调用会产生很多代价，比实例属性查询的代价还要多。我们应该在外部调用时使用 Get 和 Set 函数，但是在内部调用时，我们应该直接调用。

6. 缓冲属性调用

```
for (int i = 0; i < this.mCount; i++)
    dumpItem(this.mItems[i]);
```

而是应该这样写：

```
int count = this.mCount;
Item[] items = this.mItems;
for (int i = 0; i < count; i++)
    dumpItems(items[i]);
```

一个相似的原则就是：绝不在一个 For 语句中第二次调用一个类的方法。例如，下面的代码就会一次又一次地执行 `getCount()` 方法，相比与你把它直接隐藏到一个 `Int` 变量中，这是一个极大地浪费。

```
for (int i = 0; i < this.getCount(); i++)
    dumpItems(this.getItem(i));
```

这是一个比较好的办法，当你不止一次地调用某个实例时，直接本地化这个实例，把这个实例中的某些值赋给一个本地变量。例如：

```
protected void drawHorizontalScrollBar(Canvas canvas, int width, int height) {
    if (isHorizontalScrollBarEnabled()) {
        int size = mScrollBar.getSize(false);
        if (size <= 0) {
            size = mScrollBarSize;
        }
        mScrollBar.setBounds(0, height - size, width, height);
        mScrollBar.setParams(
            computeHorizontalScrollRange(),
            computeHorizontalScrollOffset(),
            computeHorizontalScrollExtent(), false);
        mScrollBar.draw(canvas);
    }
}
```

这里有四次 `mScrollBar` 的属性调用，把 `mScrollBar` 缓冲到一个栈变量之中，四次成员属性的调用就会变成四次栈的访问，这样就会提高效率。

7. 声明 Final 常量

我们可以看看下面一个类顶部的声明：

```
static int intVal = 100;
static String strVal = "Hello, world!";
```

当一个类第一次使用时，编译器会调用一个类初始化方法<clinit>，这个方法将 100 存入变量 `intVal`，并且为 `strVal` 在类文件字符串常量表中提取一个引用，当这些值在后面引用时，就会直接属性调用。我们可以用关键字“`final`”来改进代码：

```
static final int intVal = 100;
static final String strVal = "Hello, world!";
```

这个类将不会调用 <clinit>方法，因为这些常量直接写入了类文件静态属性初始化中，这个初始化直接由虚拟机来处理。代码访问 `intVal` 将会使用 `Integer` 类型的 100，访问 `strVal` 将使用相对节省的“字符串常量”来替代一个属性调用。

将一个类或者方法声明为“`final`”并不会带来任何执行上的好处，它能够进行一定的最优化处理。例如，如果编译器知道一个 `Get` 方法不能被子类重载，那么它就把该函数设置成 `Inline`。

同时，你也可以把本地变量声明为 `final` 变量。但是，这毫无意义。作为一个本地变量，使用 `final` 只能使代码更加清晰（或者在匿名访问内联类时，你不得不用）。

8. 慎重使用增强型 For 循环语句

增强型 `For` 循环（也就是常说的“`For-each` 循环”）经常用于 `Iterable` 接口的继承收集接口上面。在这些对象里面，一个 `iterator` 被分配给对象去调用它的 `hasNext()`和 `next()`方法。在一个数组列表里面，你可以直接地敷衍它，在其他的收集器里面，增强型的 `for` 循环将相当于 `iterator` 的使用。

尽管如此，下面的源代码给出了一个可以接受的增强型 `for` 循环的例子，如代码清单 15-6 所示。

代码清单 15-6 `for` 循环例子

```
public class Foo {
    int mSplat;
    static Foo mArray[] = new Foo[27];
    public static void zero() {
        int sum = 0;
        for (int i = 0; i < mArray.length; i++) {
            sum += mArray[i].mSplat;
        }
    }
    public static void one() {
        int sum = 0;
        Foo[] localArray = mArray;
        int len = localArray.length;

        for (int i = 0; i < len; i++) {
            sum += localArray[i].mSplat;
        }
    }
    public static void two() {
        int sum = 0;
        for (Foo a: mArray) {
            sum += a.mSplat;
        }
    }
}
```

`zero()` 函数在每一次的循环中重新得到静态属性两次，获得数组长度一次。`one()` 函数把所有的东西都变为本地变量，避免类查找属性调用。`two()` 函数使用 Java 语言的 1.5 版本中的 `for` 循环语句，编辑者产生的源代码考虑到了复制数组的引用和数组的长度到本地变量，是遍历数组比较好

的方法，它的主循环中确实产生了一个额外的载入和储存过程（显然保存了“a”），相比函数 `one()` 来说，它有一点比特上的减慢和 4 字节的增长。

总结之后，我们可以得到：增强的 `for` 循环在数组里面表现很好，但是当和 `Iterable` 对象一起使用时要谨慎，因为这里多了一个对象的创建。

9. 避免列举类型

列举类型非常好用，当考虑到尺寸和速度时，就会显得代价很高，例如：

```
public class Foo {
    public enum Shrubbery { GROUND, CRAWLING, HANGING }
}
```

这会转变成为一个 900 字节的 `class` 文件 (`Foo$Shrubbery.class`)。第一次使用时，类的初始化要在独享上面调用方法去描述列举的每一项，每一个对象都要有它自身的静态空间，整个被储存在一个数组里面（一个叫做“\$VALUE”的静态数组）。那是一大堆的代码和数据，仅仅是为了三个整数值。

```
Shrubbery shrub = Shrubbery.GROUND;
```

这会引起一个静态属性的调用，如果 `GROUND` 是一个静态的 `Final` 变量，编译器会把它当作一个常数嵌套在代码里面。

还有一点要说的，通过列举，你可以得到更好的 `API` 和一些编译时间上的检查。因此，一种比较平衡的做法就是：你应该尽一切方法在你的公用 `API` 中使用列举型变量，当处理问题时就尽量避免。

在一些环境下面，通过 `ordinal()` 方法获取一个列举变量的整数值是很有用的，例如：把下面代码：

```
for (int n = 0; n < list.size(); n++) {
    if (list.items[n].e == MyEnum.VAL_X)
        // do stuff 1
    else if (list.items[n].e == MyEnum.VAL_Y)
        // do stuff 2
}
```

替换为：

```
int valX = MyEnum.VAL_X.ordinal();
int valY = MyEnum.VAL_Y.ordinal();
int count = list.size();
MyItem items = list.items();
for (int n = 0; n < count; n++) {
    int valItem = items[n].e.ordinal();
    if (valItem == valX)
        // do stuff 1
    else if (valItem == valY)
        // do stuff 2
}
```

在一些条件下，这会执行得更快，虽然没有保障。

10. 通过内联类使用包空间

我们看下面的类声明：

```
public class Foo {
    private int mValue;
    public void run() {
```



```

        Inner in = new Inner();
        mValue = 27;
        in.stuff();
    }
    private void doStuff(int value) {
        System.out.println("Value is " + value);
    }
    private class Inner {
        void stuff() {
            Foo.this.doStuff(Foo.this.mValue);
        }
    }
}

```

这里我们要注意的，我们定义了一个内联类，它调用了外部类的私有方法和私有属性。这是合法的调用，代码应该会显示“Value is 27”。

问题是 `Foo$Inner` 在理论上（后台运行上）应该是一个完全独立的类，它违规地调用了 `Foo` 的私有成员。为了弥补这个缺陷，编译器产生了一对合成的方法。

```

/*package*/ static int Foo.access$100(Foo foo) {
    return foo.mValue;
}
/*package*/ static void Foo.access$200(Foo foo, int value) {
    foo.doStuff(value);
}

```

当内联类需要从外部访问“`mValue`”和调用“`doStuff`”时，内联类就会调用这些静态的方法，这就意味着你不是直接访问类成员，而是通过公共的方法来访问的。前面我们谈过间接访问要比直接访问慢，因此这是一个按语言习惯无形执行的例子。

让拥有包空间的内联类直接声明需要访问的属性和方法，我们就可以避免这个问题，这里是包空间而不是私有空间。这运行得更快并且去除了生成函数前面东西（缺点就是：它同时也意味着该属性也能够被相同包下面的其他的类直接访问，这违反了标准的面向对象的使所有属性私有的原则。同样，如果是设计公共的 API，你就要仔细地考虑这种优化的用法）。

11. 避免浮点类型的使用

在奔腾 CPU 发布之前，游戏作者尽可能得使用 `Integer` 类型的数学函数是很正常的。在奔腾处理器里面，浮点数的处理成为它一个突出的特点，并且浮点数与整数的交互使用相比单独使用整数来说，前者会使你的游戏运行得更快，而一般地，在桌面电脑上面我们可以自由地使用浮点数。

不幸的是，嵌入式的处理器通常并不支持浮点数的处理，因此所有的“`float`”和“`double`”操作都是通过软件进行的，一些基本的浮点数的操作就需要花费毫秒级的时间。

同时，即使是整数，一些芯片也只有乘法而没有除法。在这些情况下，整数的除法和取模操作都是通过软件实现。当创建一个散列表或者进行大量的数学运算时，这都是要考虑的。

12. 一些标准操作的时间比较

为了进一步说明我们的观点，表 15-1 给出了包括一些基本操作所使用的大概时间。注意，这些时间并不是绝对的时间，绝对时间要考虑到 CPU 和时钟频率。系统不同，时间的长短也会有所差别。当然，这也是一种有意义的比较方法，我们可以比较不同操作花费的相对时间。例如，添加一个成员变量的时间是添加一个本地变量的 4 倍。

表 15-1 一些标准操作的时间比较

操 作	时间 (ms)
添加一个局部变量	1
添加一个成员变量	4
调用 String.length()	5
调用一个空的静态方法	5
调用一个空的静态方法	12
调用一个空的虚方法	12.5
调用一个空的接口方法	15
调用 HashMap 中的 Iterator.next()方法	165
调用 HashMap 中的 put()方法	600
从 xml 文件中读取一个 View	22000
读取一个 LinearLayout 中包含 1 个 TextView 对象	25000
读取一个 LinearLayout 包含 6 个 View 对象	100000
读取一个 LinearLayout 包含 6 个 TextView 对象	135000
启动一个空的 Activity	3000000

写高效的嵌入式程序的最好方法就是要搞清楚你写的程序究竟做了些什么。如果你真地想分配一个 iterator 类, 尽一切方法在一个 List 中使用增强型的 for 循环, 使它成为一个有意而为之的做法, 而不是一个无意的疏漏而产生负面影响。

有备无患, 搞清楚你在做什么! 你可以给自己拟定一些行为准则, 但是一定要注意你的代码正在做什么, 然后开始寻找方法去优化它。

15.6 Android UI 优化

当我们更多地关注应用程序或者游戏所达到的结果时, 往往非常容易忽视一些优化的问题, 例如内存优化、线程优化、媒体优化和 UI 优化等等。不同的模块都存在更为巧妙的方式来对待一般性问题, 所以每当我们实现一个行为后, 稍微多花一些时间来考虑目前所做的工作是否存在更为高效的解决办法。本节我们将对常用的 Layout 进行优化。

在 Android 中, 最常用 LinearLayout 表示 UI 的框架, 而且也是最直观和方便的方法。例如, 创建一个 UI 用于展现 Item 的基本内容, 如图 15-3 所示。我们将图形用一个线框的形式表示, 如图 15-4 所示。



图 15-3 LinearLayout 布局



图 15-4 线框图

可以直接通过 LinearLayout 快速地实现这个 UI 的排列, 如代码清单 15-7 所示。

代码清单 15-7 LinearLayout 布局

```
<LinearLayout
    xmlns:
        android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="?android:attr/listPreferredItemHeight"
```

```

        android:padding="6dip"
    >
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="6dip"
        android:src="@drawable/icon"
    />
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="0dip"
        android:layout_weight="1"
        android:layout_height="fill_parent"
    >
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="My Application"
        />
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:singleLine="true"
            android:ellipsize="marquee"
            android:text="Simple application that shows how to use RelativeLayout"
        />
    </LinearLayout>
</LinearLayout>

```

尽管可以通过 `LinearLayout` 实现我们所预想的结果，但是这里存在一个优化的问题，尤其是针对大量 `Items`。比较 `RelativeLayout` 和 `LinearLayout`，在资源利用上，前者会占用更少的资源而达到相同的效果，如代码清单 15-8 所示，是用 `RelativeLayout` 实现的代码。

代码清单 15-8 `RelativeLayout` 布局

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip"
>
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentBottom="true"
        android:layout_marginRight="6dip"
        android:src="@drawable/icon"
    />
    <TextView
        android:id="@+id/secondLine"
        android:layout_width="fill_parent"
        android:layout_height="26dip"
        android:layout_toRightOf="@id/icon"
    />

```

```

        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:singleLine="true"
        android:ellipsize="marquee"
        android:text="Simple application that shows how to use RelativeLayout"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/icon"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:layout_above="@id/secondLine"
        android:layout_alignWithParentIfMissing="true"
        android:gravity="center_vertical"
        android:text="My Application"
    />
</RelativeLayout>

```

针对 **RelativeLayout** 有一点需要注意，因为它内部是通过多个 **View** 之间的关系而确定的框架，那么当其中某一个 **View** 因为某些需要调用 **GONE** 来完全隐藏掉后，会影响与其相关联的 **Views**。**Android** 为我们提供了一个属性 **alignWithParentIfMissing** 用于解决类似问题，当某一个 **View** 无法找到与其相关联的 **Views** 后，它将依据 **alignWithParentIfMissing** 的设定判断是否与父级 **View** 对齐。图 15-5 是两种不同 layout 在 **Hierarchy Viewer** 中的层级关系图：

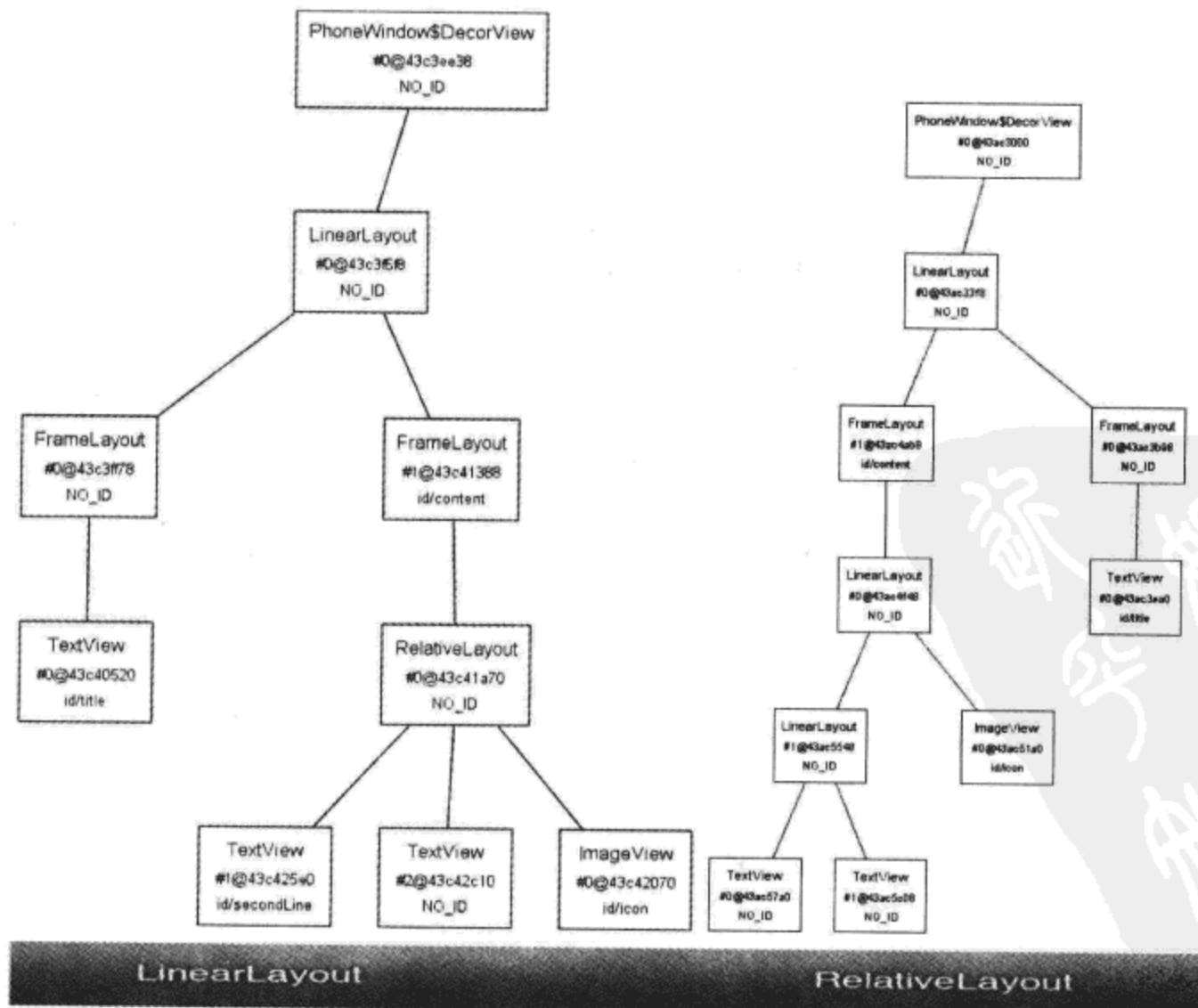


图 15-5 两种不同 layout 在 Hierarchy Viewer 中的层级关系

简单或复杂的问题都需要时常考虑如何优化资源的分配。比如一个功能很简单的应用程序，它会调用一些我们常用的对话框或者输入面板，这需要采用统一的方式来针对不同的应用程序制定统一标准。

当我们面对 Android UI 优化时，有必要继续考虑资源复用。手机开发给我们的直观感觉是运行其上的软件应该尽可能地达到资源高效利用的极致，而不能像开发 PC 机那样，似乎有用之不尽的资源。

定义 Android Layout(XML)时，有 4 个比较特别的标签是非常重要的，其中有 3 个与资源复用有关，它们分别是 `<viewStub />`、`<requestFocus />`、`<merge />` 和 `<include />`。可是以往我们所接触的实例或者官方文档的例子都没有着重去介绍这些标签的重要性。

- `<viewStub />`: 此标签可以使 UI 在特殊情况下，直观效果类似于设置 View 的不可见性，但是其更大的(R)意义在于被这个标签所包裹的 View 在默认状态下不会占用任何内存空间。`viewStub` 通过 `include` 从外部导入 View 元素。用法是通过 `android:layout` 来指定所包含的内容。默认情况下，ViewStub 所包含的标签都属于 `visibility=GONE`。`viewStub` 通过方法 `inflate()` 来召唤系统加载其内部的 View。

```
<ViewStub
    android:id="@+id/stub"
    android:inflatedId="@+id/subTree"
    android:layout="@layout/mySubTree"
    android:layout_width="120dip"
    android:layout_height="40dip"/>
```

- `<include />`: 可以通过这个标签直接加载外部的 xml 到当前结构中，是复用 UI 资源的常用标签。用法是将需要复用 xml 文件路径赋予 `include` 标签的 `Layout` 属性。

```
<include
    android:id="@+id/cell1"
    layout="@layout/ar01"/>
<include
    android:layout_width="fill_parent"
    layout="@layout/ar02"/>
```

- `<requestFocus />`: 标签用于指定屏幕内的焦点 View。用法是将标签置于 View 标签内部。

```
<EditText
    id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="0"
    android:paddingBottom="4">
    <requestFocus/>
</EditText>
```

- `<merge />` 标签: 优化 UI 结构时起到很重要的作用。目的是通过删减多余或者额外的层级，从而优化整个 Android Layout 的结构。

将通过一个例子来了解这个标签实际产生的作用，这样可以更直观地了解 `<merge/>` 的用法。建立一个简单的 Layout，其中包含两个 View 元素: `ImageView` 和 `TextView`。默认状态下我们将这两个元素放在 `FrameLayout` 中，其效果是在主视图中全屏显示一张图片，之后将标题显示在图片上，并位于视图的下方，如代码清单 15-9 所示。

代码清单 15-9 \第 15 章\Examples_15_03\res\layout\main.xml

```

<?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"
    >
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="center"
        android:src="@drawable/golden_gate"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="20dip"
        android:layout_gravity="center_horizontal|bottom"
        android:padding="12dip"
        android:background="#AA000000"
        android:textColor="#ffffff"
        android:text="Golden Gate"
        />
</FrameLayout>

```

运行效果如图 15-6 所示。启动 Android SDK 目录下的 tools 文件夹下的 hierarchyviewer.bat 工具，如图 15-7 所示；查看当前 UI 结构视图，如图 15-8 所示。



图 15-6 ImageView 和 TextView 运行效果

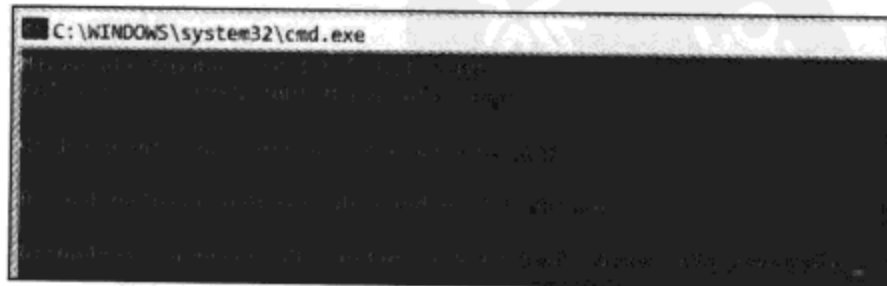


图 15-7 运行 hierarchyviewer.bat 工具

我们可以很明显地看到，图 15-8 中由线框所包含的结构出现了两个 `frameLayout` 节点，很明显这两个意义完全相同的节点造成了资源浪费（这里可以提醒大家在开发工程中可以习惯性地通过 `hierarchyViewer` 查看当前 UI 资源的分配情况），那么如何才能解决这种问题呢（就当前例子是如何去掉多余的 `frameLayout` 节点）？这时候就要用到 `<merge />` 标签来处理类似的问题了。我们将上边

xml 代码中的 `framLayout` 替换成 `merge`，如代码清单 15-10 所示。

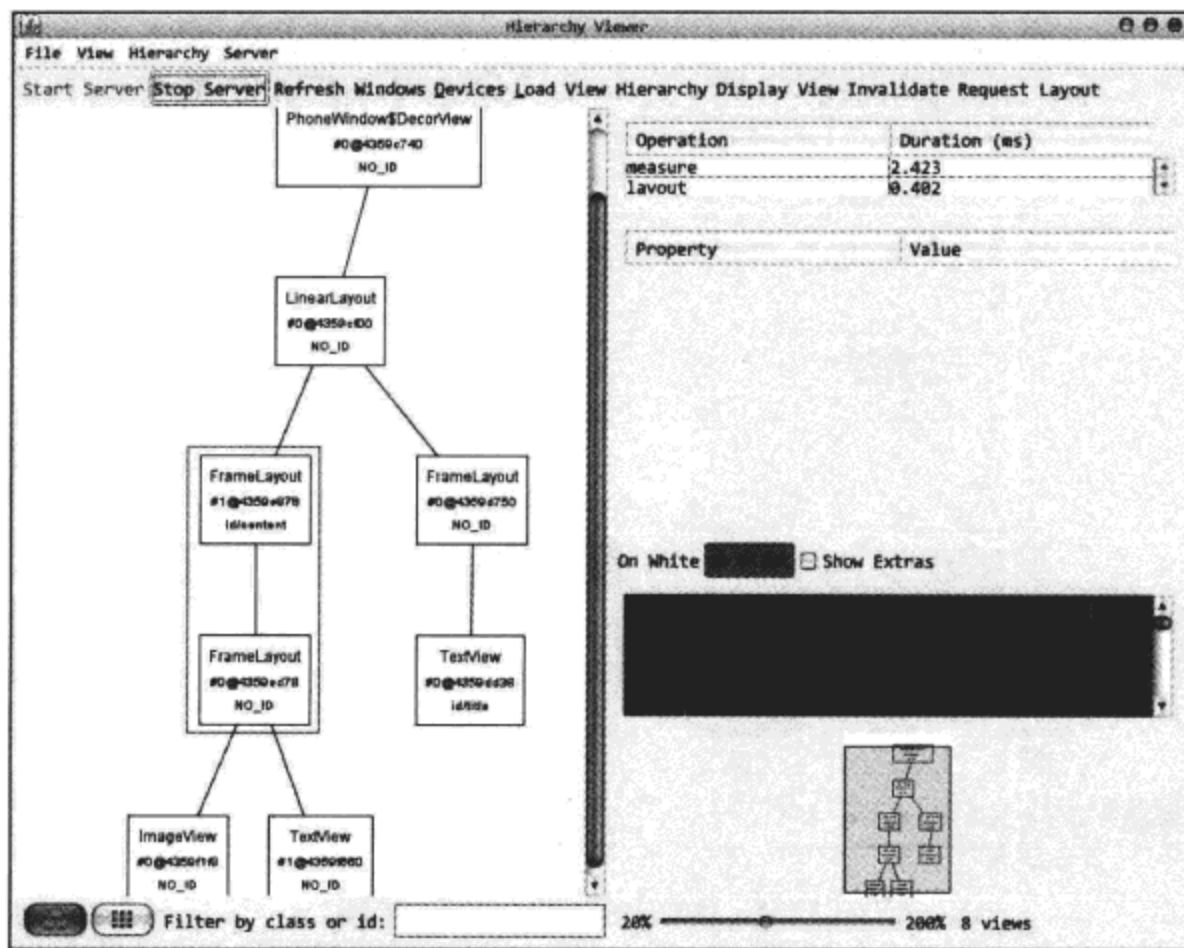


图 15-8 main.xml 的 UI 结构视图

代码清单 15-10 \第 15 章\Examples_15_03\res\layout\main2.xml

```
<merge xmlns:android="http://schemas.android.com/apk/res/android">
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="center"
        android:src="@drawable/golden_gate"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="20dip"
        android:layout_gravity="center_horizontal|bottom"
        android:padding="12dip"
        android:background="#AA000000"
        android:textColor="#ffffff"
        android:text="Golden Gate"/>
</merge>
```

运行程序后在 Emulator 中显示的效果和图 15-6 是一样的，可是通过 `hierarchyviewer` 查看的 UI 结构是有变化的，如图 15-9 所示，当初多余的 `frameLayout` 节点被合并在一起了，或者可以理解为将 `merge` 标签中的子集直接加到 Activity 的 `frameLayout` 根节点下（这里需要提醒大家注意：所有的 Activity 视图的根节点都是 `frameLayout`）。如果你所创建的 Layout 并不是把 `framLayout` 作为根节点（而是应用 `LinerLayout` 等定义 `root` 标签），就不能应用上边的例子通过 `merge` 来优化 UI 结构。

除了上边的例子外，`meger` 还有另外一个用法，当应用 `Include` 或者 `ViewStub` 标签从外部导入 xml 结构时，可以将被导入的 xml 用 `merge` 作为根节点表示，这样当被嵌入父级结构中后可以很好

地将它所包含的子集融合到父级结构中，而不会出现冗余的节点。

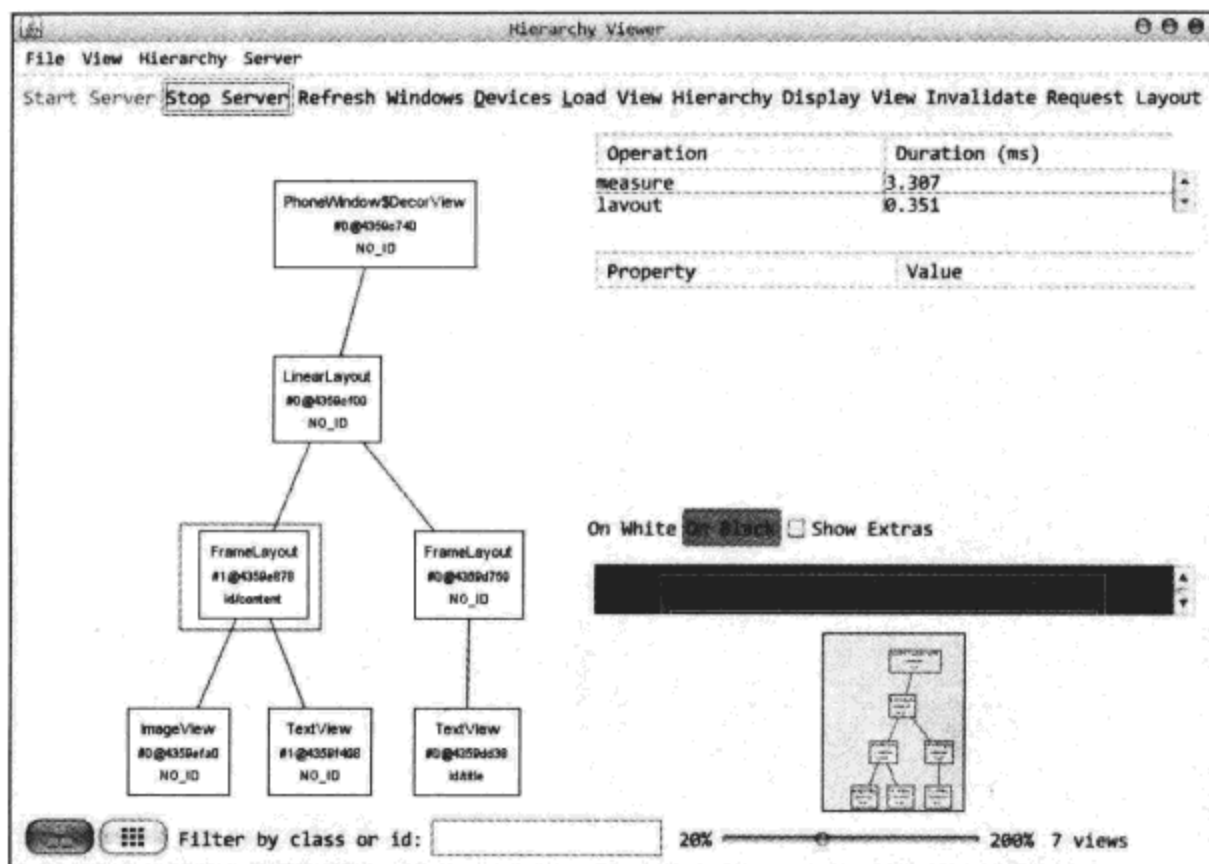


图 15-9 main2.xml 的 UI 结构视图

注意 <merge />只可以作为 xml layout 的根节点。当需要扩充的 xml layout 本身是由 merge 作为根节点的话，需要将导入的 xml layout 置于 viewGroup 中，同时需要设置 attachToRoot 为 True。

15.7 其他优化

15.7.1 zipalign

在 Android 1.6 SDK 中包含一个工具，称为 zipalign，它优化了应用程序的打包方式。这样做使 Android 与你的应用程序更加有效和简便地交互，有可能提高应用程序和整个系统的运行速度。我们强烈建议你在 Android apk 程序中使用 zipalign。

在 Android 程序中，每个应用程序的数据文件存储在软件系统中，可以由多个进程访问：安装程序读取 manifest 文件处理与该应用程序相关联的权限，主应用程序读取资源文件来获取应用程序的名称和图标，系统服务器为程序的某些请求来读取资源（例如：显示该应用程序的通知），另外的资源文件，由应用程序本身使用。

我们在程序中可以很方便地校准整理资源。

1. 使用 ADT 命令

ADT (版本 0.9.3 及以后) 可以选择自动校验程序文件资源，下面来看看如果使用它：首先鼠标右键点击 Eclipse 中项目名称 >>Android Tools>>Export Signed Application Package，剩下的工作就交给 Eclipse 来为你完成。

2. 使用 Ant 命令

Android 1.6 (API 级别为 4) 中的 Ant 命令可以自动为你校验应用程序包，但是不可以自动为

Android 旧版本的程序校验，所以你需要手动校验。在默认选项下，Ant 命令构建的 Package 不包含任何的签署内容。

3. 手动

为了手动检验包，zipalign 位于 Android1.6 SDK tools 文件夹内。它可以用来申请针对任何一个机器人版本的软件包。它应运行在签署 apk 文件，使用以下命令：

```
zipalign -v 4 source.apk dest.apk
```

4. 验证校对

下面这个命令可以验证.apk 应用程序文件：

```
zipalign -c -v 4 app.apk
```

zipalign 命令参数：

- ☐ -f: 覆盖现有的输出文件。
- ☐ -v: 详细输出。
- ☐ -c: 确认给定的文件校准。

15.7.2 图片优化

在 Android 平台中，二维图像处理库 BitmapFactory 做得比较智能，为了减少文件体积和效率，常常不用很多资源文件，而是把很多小图片放在一个图片中，用切片的方式来完成，在 J2ME 中我们这样是为了减少文件头从而解决 MIDP 设备的问题，从 Android G1 硬件配置参数中我们可以看出 Android 中虽然机型硬件配置都比较高，但是当资源多时这样的运行效率还是不太令人满意，至少 Dalvik 优化得还不是很好。因此我们在开发过程中同样需要注意美工在图片上的优化，以及尽可能地降低对高质量图片的使用。

1. 图片压缩

手机的屏幕大小因不同厂商、不同型号而不尽相同。因此，同一张图片怎样在不同大小屏幕的手机上合理地显示出来对一款游戏来说是相当重要的。图片缩小的操作是删除部分像素点，小图放大则要人为地添加一些像素点。如把一个 2×2 的图片放大成 4×4 的图片。图片放大并不总是成倍的，比如从 2×2 放大到 3×3。在移植到不同的手机时可以先用菜单画布界面 getWidth()及 getHeight()取得当前手机屏幕的宽度和高度，如要当前图片不够全屏显示时就对图片进行放大，相反就进行缩小，也可自定义屏幕显示图片的大小，如在屏幕上显示多张图片的缩略图，使图片方式的显示更加丰富。灵活使用 J2ME 的图形处理技术，可以优化手机游戏图形显示，减少游戏对手机资源的占用等，降低了手机游戏对手机硬件配置的要求，既丰富了手机游戏的图像效果，又增加了手机游戏的可移植。

2. 减少图片容量

将多张图片集成到一张图片上：如果游戏有 10 张图片，每张 7×11，可以把它集成到一张 7×110 或者 110×7 或者其他尺寸的图片上去。这张大图片的容量比 10 张图片的总容量小很多。这是因为省去了 10 张图片的文件头、文件结束数据块等，而且合并了调色板（如果 10 张图片的调色板恰好相同，则省去了 10 张图片的调色板所占的容量，这是个不小的数字。）

减少图片的颜色数：如果游戏完成后发现容量超出，此时再用优化工具减少颜色，虽然能降低

图片容量，但图片效果可能就差一些。所以，在美工画图时就要确定使用的颜色数，手机游戏使用的是像素图，即一个像素一个像素点出来的图像，所以预先规定调色板颜色数量是可以办到的。例如将图片的色深减少一半，图片的体积会减小到原来的 1/3。

15.8 小结

Google 设计 Android 平台时选择了 Java，Java 语言是一种解释型语言，需要 Java VM 实时解析运营，虽然有 JNI 机制但是似乎 Google 并不希望 Android 平台出现很多安全问题，提供程序运行效率的方法就显得尤为重要了，尤其是性能敏感的游戏设计。本章我们从优化的基本知识开始，讲述了为什么要优化，优化的重要性，如何找到需要优化的程序，如何优化等各方面内容。也对 Android 平台的特殊优化进行了重点介绍。优化的方式很多，本文只是尽可能地介绍了常用的方法，读者可以根据实际需求来进行选择性的使用或研究出更先进的方法。当然笔者最后还要提醒一下，一个优秀的程序员必须具备良好的编程规范。





Android NDK 开发

我们都知道 Android SDK 在最初发布时，Google 官方将 Java 作为第三方应用的开发语言，这就使得很多 C 语言开发人员被拒之门外，但是并没有完全拒绝 C 语言开发人员参与开发，因为在 Android 发布初期，Google 就表明其虚拟机 Dalvik 支持 JNI 编程方式，也就是第三方应用完全可以通过 JNI 调用自己的 C 动态库，但是 Google 官方并没有明确表示支持开发者使用这种方法。终于在 2009 年 6 月 26 日，Google Android 发布了 NDK (Native Development Kit, 原生态开发包)，引起了很多开发人员的兴趣。它支持开发者用 C/C++ 语言开发 Android 程序。它将作为 Android SDK 的一个附加组件提供，开发者须先安装 Android SDK 方可使用 NDK。NDK 的目的是为了增加代码的重用性及加快程序的运行速度，这有利于开发者从其他系统上移植软件到 Android 平台，同时也为喜欢使用 C 来开发的朋友们提供了方便。本章我们就将学习如何使用 Android NDK 来参与应用开发，开始之前先说明一下，本章的内容要求开发者具有一定的 Linux 和 C/C++ 编程的知识。

16.1 Android NDK 简介

要学习 NDK，首先我们必须知道 NDK 是什么？用它能够做什么？

在 NDK 发布之前，Android 平台的第三方应用程序均是依靠基于 Java 的 Dalvik 特制虚拟机进行开发的。原生 SDK 的公布可以让开发者更加直接地接触 Android 系统资源，并使用传统的 C 或 C++ 语言编写程序，并在程序封包文件 (.apks) 中直接嵌入原生库文件。不过，Google 也表示，使用原生 SDK 编程相比于 Dalvik 虚拟机也存在一些劣势，比如程序更加复杂，兼容性难以保障，无法访问 API 框架，调试难度更大等。开发者需要自行斟酌使用。虽然有了 NDK，但还是不能只通过 NDK 来开发 Android 应用，因为 NDK 并没有提供各种系统事件处理支持，也没有提供应用程序生命周期维护。此外，在本次发布的 NDK 中，应用程序 UI 方面的 API 也没有提供。所以我们要开发应用程序还需要使用 Java，通过 JIN 来调用所实现的 C 动态库。

Android NDK 是配合 Android SDK 的工具，NDK 的目的不是为了取代 Android SDK，并且也不可能完全取代，它只是作为 Android SDK 的一个补充。用来编译应用的原生代码。它只能与 Android SDK 配合使用，因此下载 NDK 之前请先安装 Android 1.5 SDK 及以上版本。Android 应用运行在 Dalvik 虚拟机上。NDK 允许开发者用原生代码 (C 或 C++) 实现应用的一部分。这将给某些应用带来好处，这种方式可重用代码，而且在某些情况下可加快运行速度。NDK 提供了以下资源。

- ❑ 将 C 和 C++ 源代码生成原生代码库的工具和文件。
- ❑ 将原生库嵌入 apk 文件的方法。

- ❑ 兼容 1.5 版本以上的原生系统头文件和库。
- ❑ 文档，示例和指引。

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

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

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

1. NDK 是一系列工具的集合

- ❑ NDK 包含了一套交叉编译工具，它可生成 Linux，MAC 和 Windows（用 Cygwin）上的原生 ARM 的二进制码。
- ❑ NDK 提供了一系列的工具，帮助开发者快速开发 C 或 C++ 的动态库，并能自动将 so 和 Java 应用一起打包成 apk。这些工具对开发者的帮助是巨大的。
- ❑ NDK 集成了交叉编译器，并提供了相应的 mk 文件隔离 CPU、平台、ABI 等差异，开发人员只需要简单修改 mk 文件（指出“哪些文件需要编译”、“编译特性要求”等），就可以创建出 so。

2. NDK 提供了一套原生 API 的系统头文件

- ❑ libc (C library) headers: C 标准库
- ❑ libm (math library) headers: 标准数学库
- ❑ JNI interface headers: JNI 接口
- ❑ libz (Zlib compression) headers: 压缩库
- ❑ liblog (Android logging) header: Log 库
- ❑ A Minimal set of headers for C++ support: 一部分 C++ 库

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

3. NDK 的前景

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

16.2 安装和配置 NDK 开发环境

通过上一节，我们对 Android NDK 有了一个初步的了解，但是要真正使用 NDK 来开发，还需

要一系列的准备工作，包括系统的要求、环境的安装与配置等，本节我们就将详细地介绍如何在 Windows XP 中搭建 Android NDK 开发环境。

16.2.1 系统和软件需求

在安装和配置开发环境之前，我们先了解所需要的系统及软件要求，下面是 Google 官方所推荐的系统及软件。

1. Android SDK

需要完整安装 Android SDK。需要注意 Android SDK 版本为 1.5 或以上版本，本书使用 1.6 版本的 SDK。

2. Android NDK 需要安装完整的 Android NDK

大家可以到如表 16-1 所示的网站下载适合自己的 NDK 版本，本书将使用 Windows 版本的 NDK。

表 16-1 Android 1.6 NDK 下载地址

操作系统	文件名	下载地址
Windows	android-ndk-1.6_r1-windows.zip	http://dl.google.com/android/ndk/android-ndk-1.6_r1-windows.zip
Mac OS X (Intel)	android-ndk-1.6_r1-darwin-x86.zip	http://dl.google.com/android/ndk/android-ndk-1.6_r1-darwin-x86.zip
Linux 32/64-bit (x86)	android-ndk-1.6_r1-linux-x86.zip	http://dl.google.com/android/ndk/android-ndk-1.6_r1-linux-x86.zip

3. 支持的操作系统

- ☐ Windows XP (32bit) 或 Vista (32bit 或 64bit)
- ☐ Mac OS X 10.4.8 或 x86
- ☐ Linux (32bit 或 64bit, 在 Linux Ubuntu Dapper Drake 上测试成功)

4. 需要的开发工具

在所有操作系统上，都需要安装 GNU Make 3.81 或更高版本。早期的版本也许可用，但未经测试验证。在 Windows 上需要安装较新版本的 Cygwin，包括 gmake 和 gcc 包。

Android 是基于 Linux 的操作系统，处理器是 ARM 的，所以要在 Linux 或 Windows 等 x86 系统上编译 Android 能运行的程序，你需要一个交叉编译器。在 Linux 下面，你可以自己编译一个交叉编译环境，在 Windows 下面，我们可以通过 Cygwin 编译一个交叉编译环境。当然可以选择下载一个现成的交叉编译环境，比如 CodeSourcery 编译器等。本文中我们将选择 Windows 操作系统，然后安装 Cygwin，Cygwin 下载地址为 <http://www.cygwin.com/>。

先进入 www.cygwin.com，点击其中的 Install or update now 链接（如图 16-1 所示），将下载一个名为 setup.exe 的安装文件。到这里我们就准备好了所需要的全部软件，下一节我们将分步讲解如何在 Windows 下搭建 NDK 开发环境。

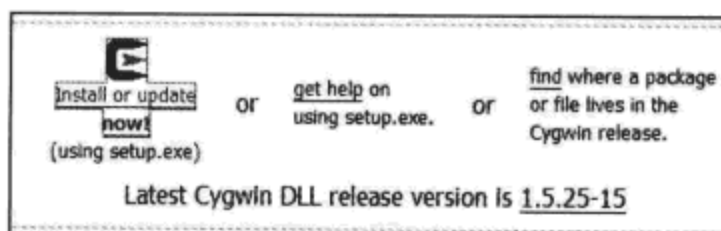


图 16-1 Cygwin 下载

16.2.2 NDK 开发环境搭建

上一节我们已经知道，要开发 NDK 就需要一个交叉编译器和 Android NDK，首先准备好上一节中我们列出的所需要的软件，本节我们就将来学习如何安装 Cygwin 环境以及配置 Android NDK。NDK 编译需要用到 Cygwin 中的 `make` 和 `gcc`，所以我们首先安装 Cygwin。

1. 安装 Cygwin

(1) 运行上一节中我们所下载的 Cygwin 的安装文件 `setup.exe`，如图 16-2 所示。

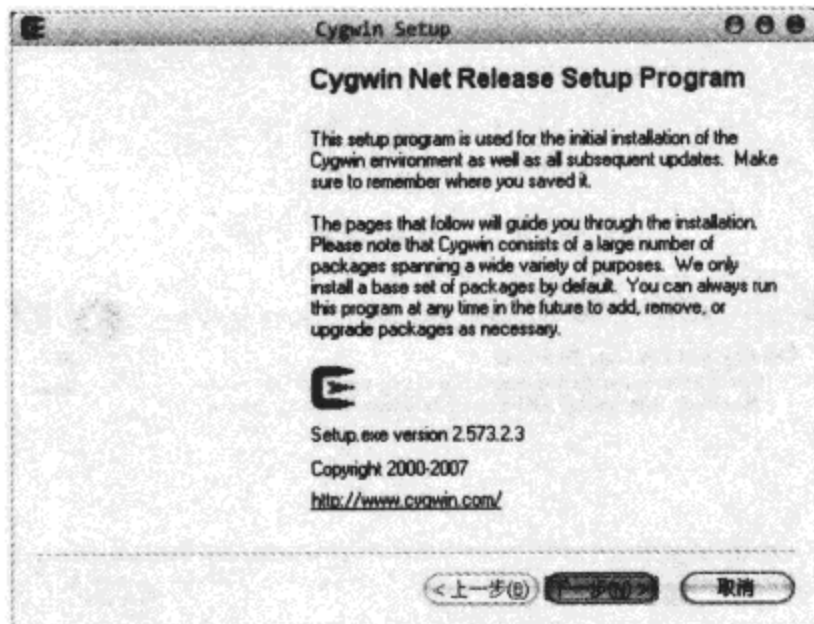


图 16-2 Cygwin 安装界面

(2) 单击“下一步”按钮，进入选择安装方式界面，如图 16-3 所示。Cygwin 提供了两种安装方式，一种是本地安装，这里采用另一种在线安装方式。

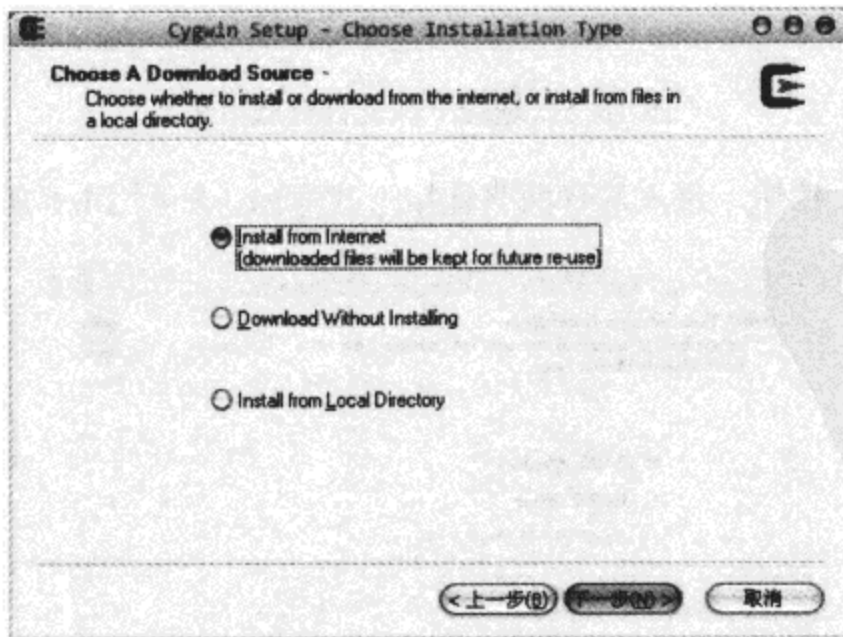


图 16-3 选择安装方式

(3) 单击“下一步”按钮，进入选择安装目录界面，如图 16-4 所示。

(4) 单击“下一步”按钮，进入选择下载文件存放目录界面，如图 16-5 所示。

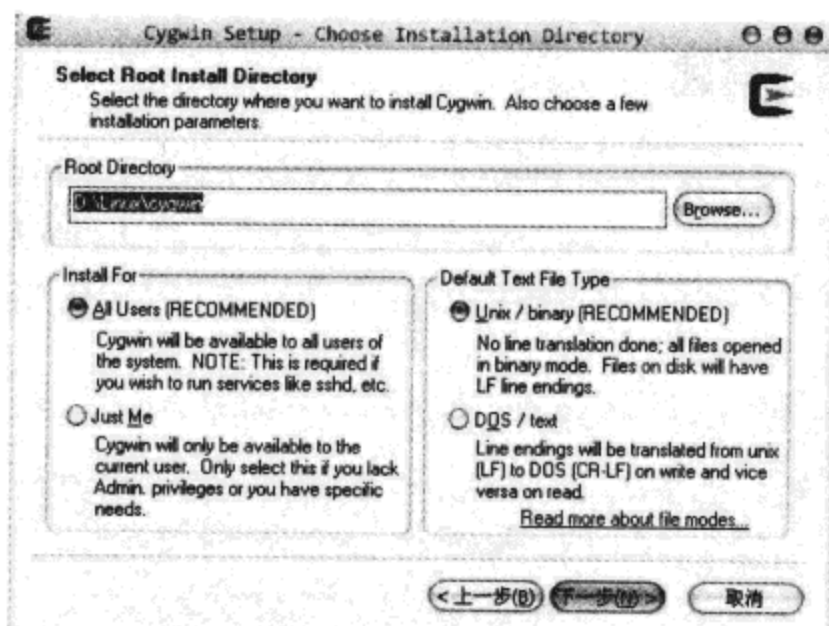


图 16-4 选择安装目录

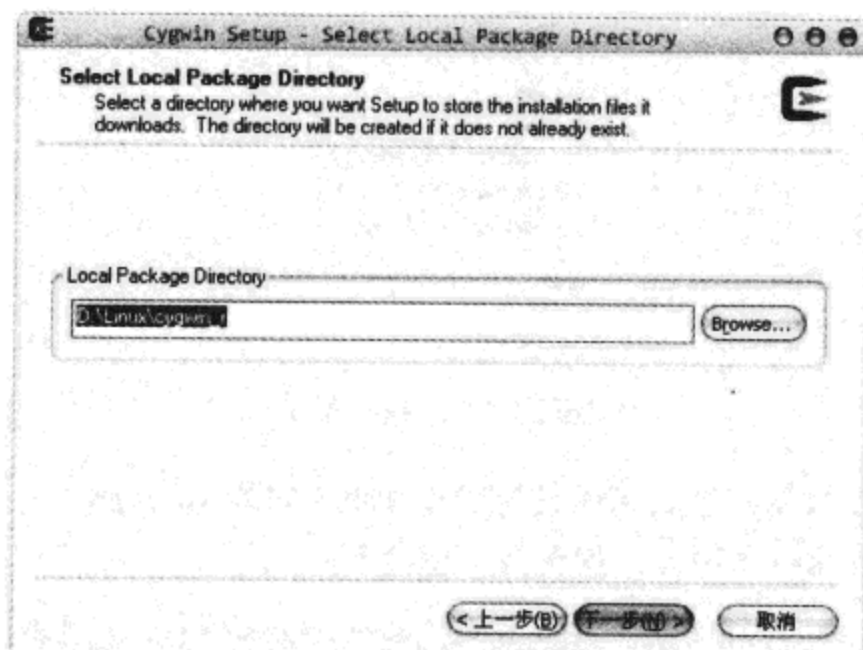


图 16-5 选择下载文件存放目录

(5) 单击“下一步”按钮，进入选择网络连接方式界面，如图 16-6 所示。

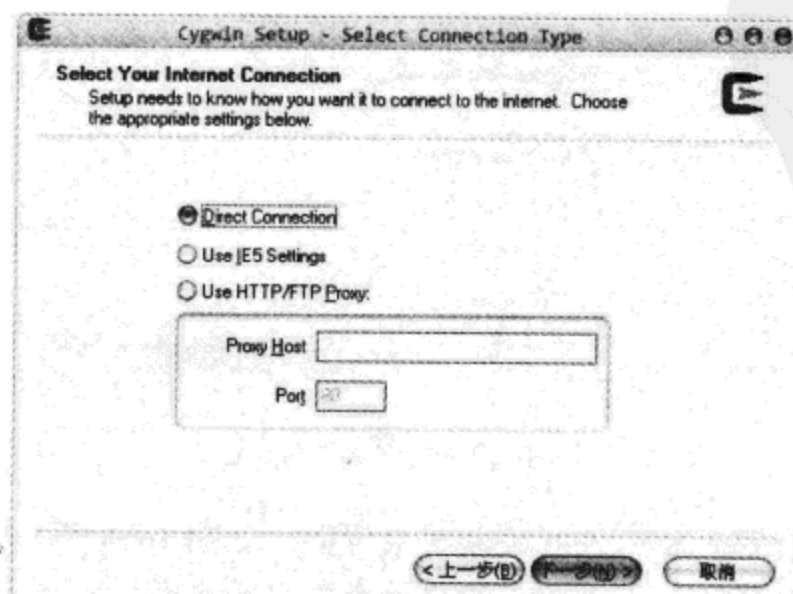


图 16-6 选择网络连接方式

(6) 单击“下一步”按钮，进入选择镜像下载站点界面，如图 16-7 所示。笔者建议选择离自己较近的镜像站点，否则下载速度会很慢，笔者这里选择了 `ftp://mirrors.kernel.org/`。

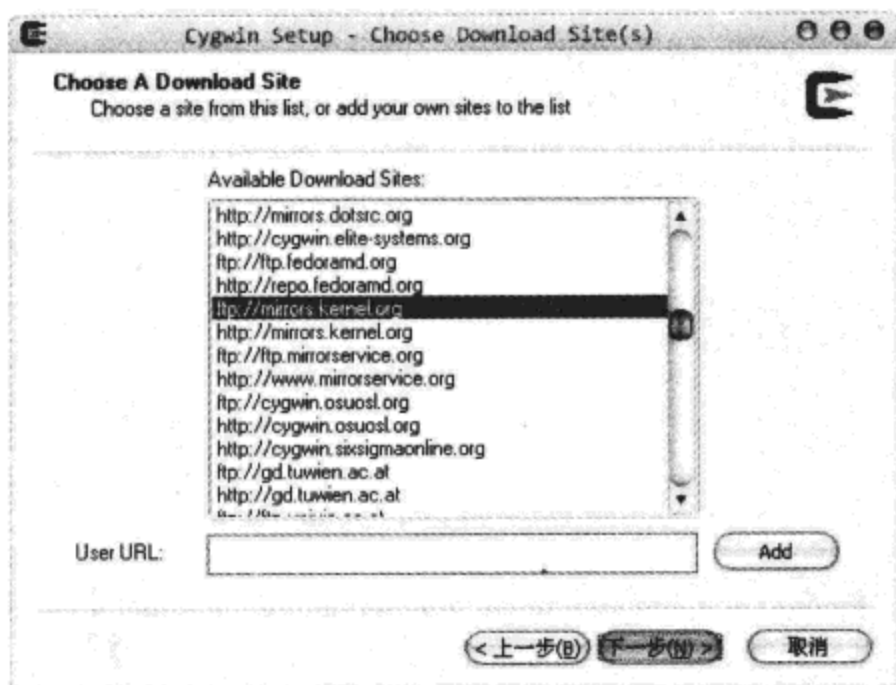


图 16-7 选择镜像下载站点

(7) 单击“下一步”按钮，开始下载，如图 16-8 所示。

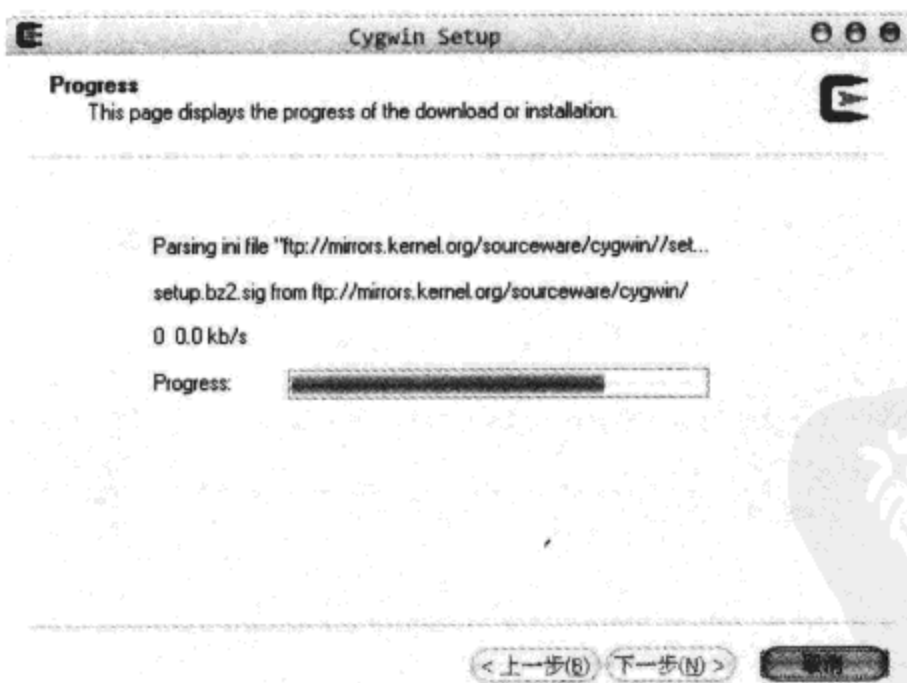


图 16-8 开始下载

(8) 等待下载完成之后，进入选择安装界面，如图 16-9 所示，我们编译 NDK 需要使用的 `make` 和 `gcc`。可以点击循环箭头图标，将 `Default` 改变为 `Install`。

(9) 单击“下一步”按钮，开始下载图 16-9 中所选择的要安装项，如图 16-10 所示。

(10) 等待下载完毕之后，程序自动安装完成，如图 16-11 所示，点击“完成”按钮，完成安装。

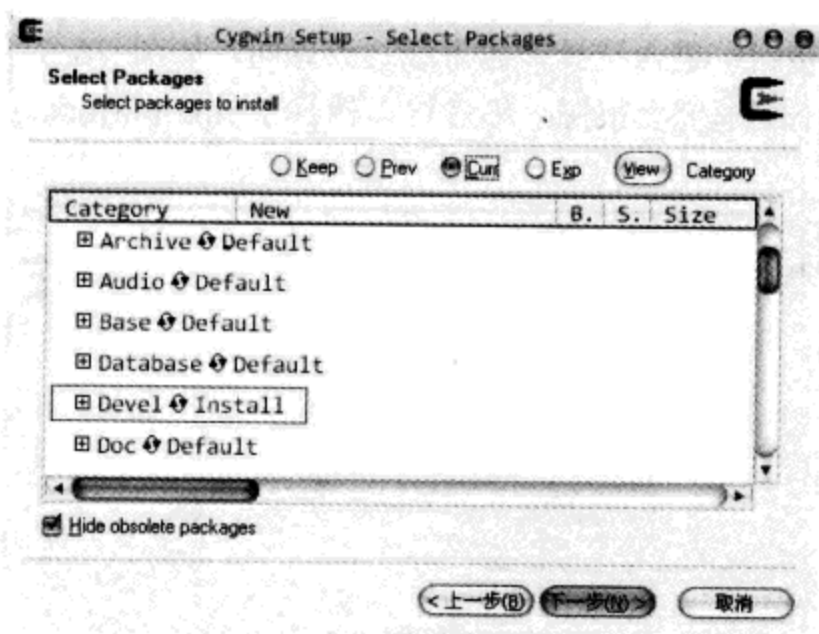


图 16-9 选择安装项

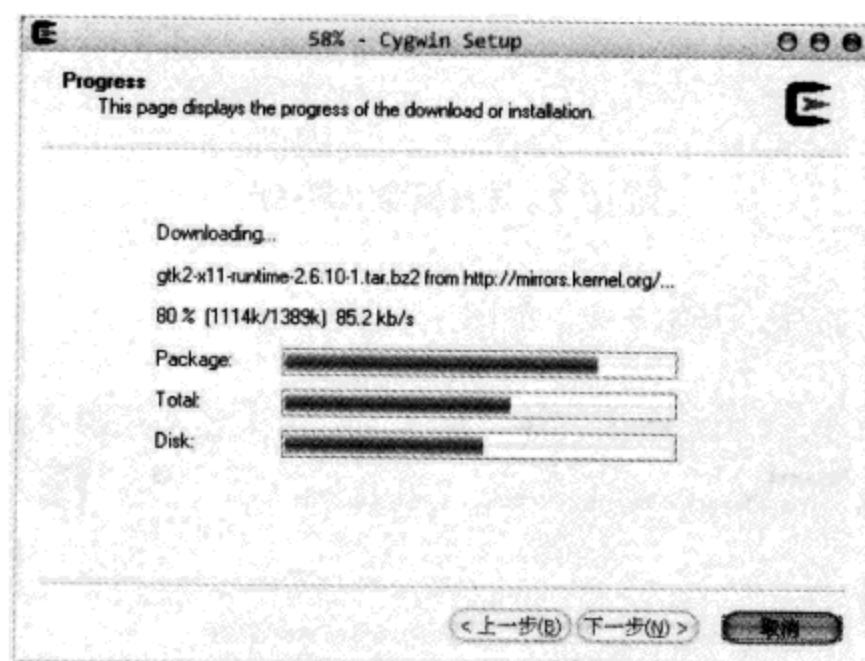


图 16-10 下载安装项

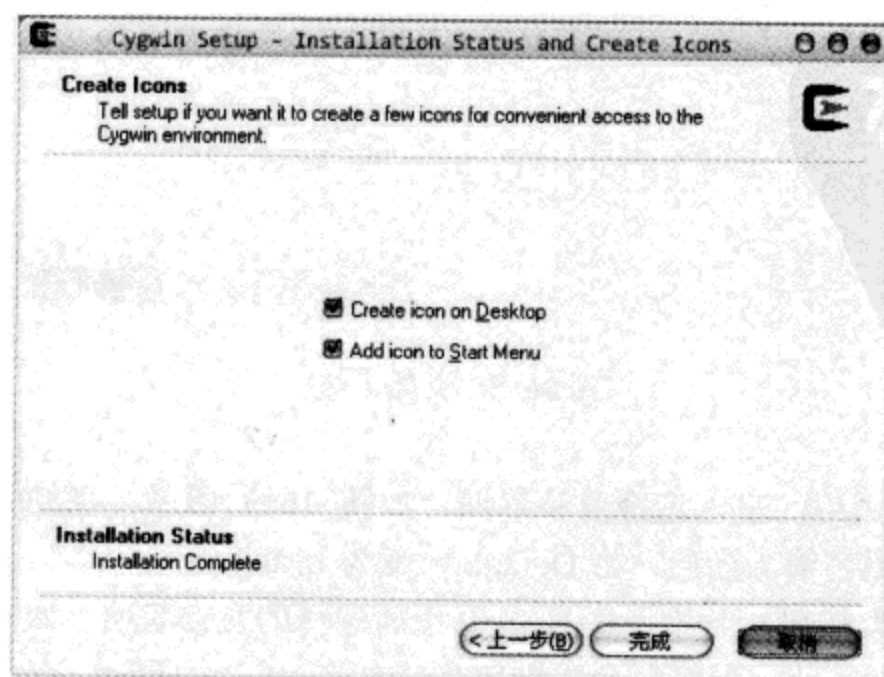


图 16-11 完成安装

(11) 运行 Cygwin, 输入 “make -v” 和 “gcc -v” 来检测是否安装成功, 如图 16-12 所示, 表示 make 和 gcc 安装成功。

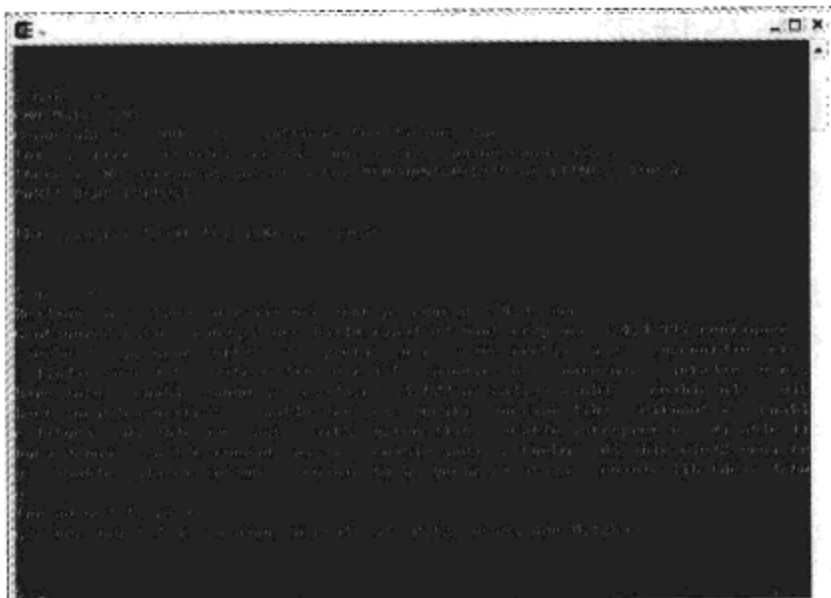


图 16-12 make 和 gcc 版本信息

到这里我们已经成功的安装了 Cygwin, 要开发 NDK, 我们还需要来安装 NDK, 下面就开始安装 NDK。

2. 安装 NDK

安装 NDK 非常简单, 只需要把下载的文件解压到指定目录即可, 本文中我们将 NDK 安装到 D:\Android\android-ndk-1.6_r1 目录中, 下面我们来配置 NDK, 运行 Cygwin, 进入 NDK 目录下的 build 文件夹下, 然后运行 “/host-setup.sh” 命令, 提示我们需要设定 NDKROOT, 如图 16-13 所示。

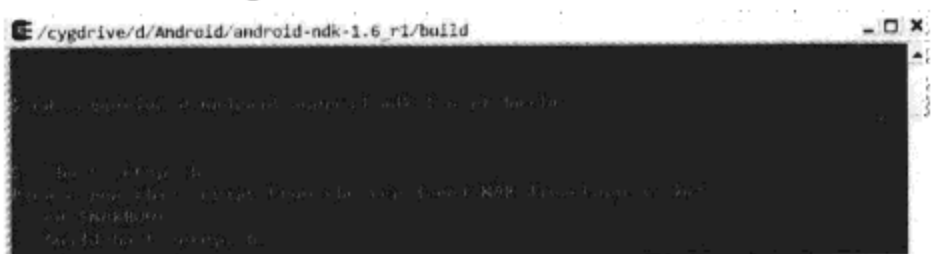


图 16-13 提示设置 NDKROOT

修改 Cygwin 目录 (/home/username) 下的 .bash_profile 文件, 在文件尾部加入如下代码, 然后重新启动 cygwin。

```
NDKROOT=/cygdrive/d/Android/android-ndk-1.6_r1 //NDK 的路径
export NDKROOT
```

进入 NDKROOT, 然后输入 “build/host-setup.sh” 命令, 若出现如图 16-14 所示画面, 则表示安装配置成功。



图 16-14 配置 NDK

到这里我们已经完成了 NDK 开发环境的搭建，下面我们将来学习编译第一个 NDK 程序。

16.2.3 编译第一个 NDK 程序

在成功安装了 NDK 之后，我们可以在 NDK 安装目录下找到 `apps` 文件夹，其下是 NDK 自带的几个例子，这里我们来学习编译第一个 HelloJni 程序，位于 NDK 根目录下 `apps\hello-jni` 文件夹中。

(1) 启动 Cygwin，进入 NDK 的根目录，输入“`make APP=hello-jni`”命令，`hello-jni` 为 `apps` 下 `hello-jni` 文件夹的名称，这个命令会先找到 `apps\hello-jni` 下的 `Application.mk` 文件，然后找到 `source\samples\hello-jni` 这个目录，并找到 `Android.mk` 这个文件中的配置信息进行编译。这里我们已经编译过了，所以使用“`make APP=hello-jni -B`”命令来重新编译，如果出现如图 16-15 所示画面，则表示编译成功。SO 文件在 `apps/hello-jni/project/libs/armeabi` 文件夹下。

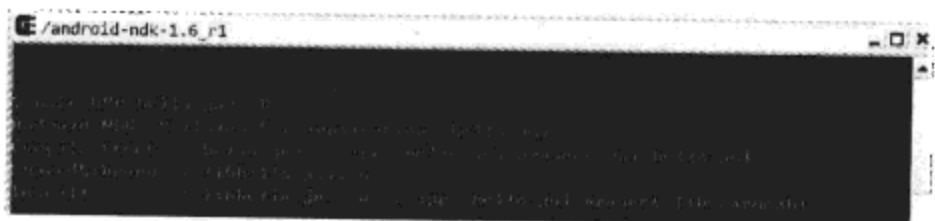


图 16-15 编译 hello-jni

(2) 将 `hello-jni` 工程导入到 Eclipse 中，如图 16-16 所示。

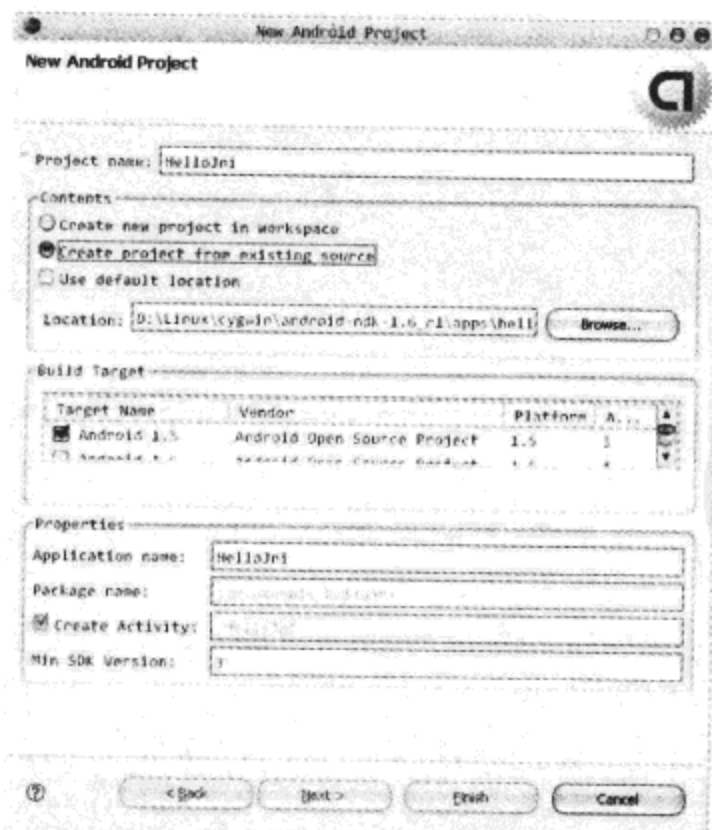


图 16-16 导入 hello-jni 到 Eclipse

(3) 导入之后，展开工程，如图 16-17 所示。

(4) 然后运行这个项目，出现如图 16-18 所示效果，则表示我们第一个 NDK 程序编译运行成功。



图 16-17 hellojni 项目

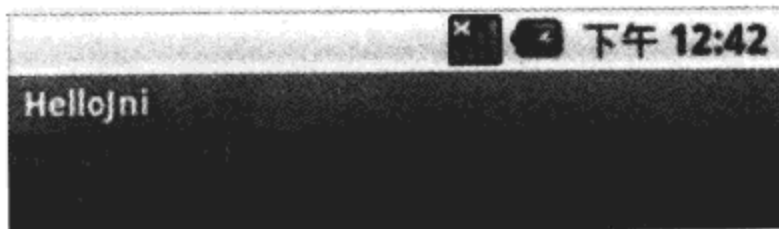


图 16-18 hellojni 运行效果

到此为止，我们可以成功地在 Android 运行第一个 NDK 程序，很有趣吧，接下来我们将使用 NDK 来开发更有趣的程序。

16.3 Android NDK 开发

Java 以其跨平台的特性深受人们喜爱，而又正由于它跨平台的目的，使得它和本地机器的各种内部联系变得很少，约束了它的功能。解决 Java 对本地操作的一种方法就是 JNI。通过前面的学习，我们知道 Android NDK 开发其实就是使用 JNI 来调用本地的方法或者库来将 Java 程序和 Native 程序结合起来。JNI 明确分开了 Java 代码与本机代码(C/C++)的执行，定义了一个清晰的 API 在这两者之间进行通信。从很大程度上说，它避免了本机代码对 JVM 的直接内存引用，从而确保本机代码只需编写一次，并且可以跨不同的 JVM 实现或版本运行。因此 Android NDK 开发一般有以下步骤：

- (1) JNI 接口设计。
- (2) 使用 C/C++实现本地方法。
- (3) 生成动态链接库。
- (4) 将动态链接库复制到 Java 工程，运行 Java 程序。

本章我们就将来学习如何使用 Android NDK 开发原生程序。

16.3.1 JNI 接口设计

Java 通过 JNI 调用本地方法，而本地方法是以库文件的形式存放的（在 WINDOWS 平台上是 DLL 文件形式，在 UNIX 机器上是 SO 文件形式）。通过调用本地的库文件的内部方法，使 Java 可以实现和本地机器的紧密联系，调用系统级的各接口方法。在 Android 中，同样也是使用 SO 文件的形式来存放本地库文件。本章我们将学习如何设计这些连接 Java 和 C/C++的接口。步骤如下：

(1) 按“android-ndk-1.6_r1\apps\HelloNDK\project”目录创建一个 NDK 工程，其中 HelloNDK 为 NDK 工程文件夹，project 存放 Java 工程和 C/C++代码。

(2) 创建 Android 工程，并指定创建到“android-ndk-1.6_r1\apps\HelloNDK\project”目录，如

图 16-19 所示。

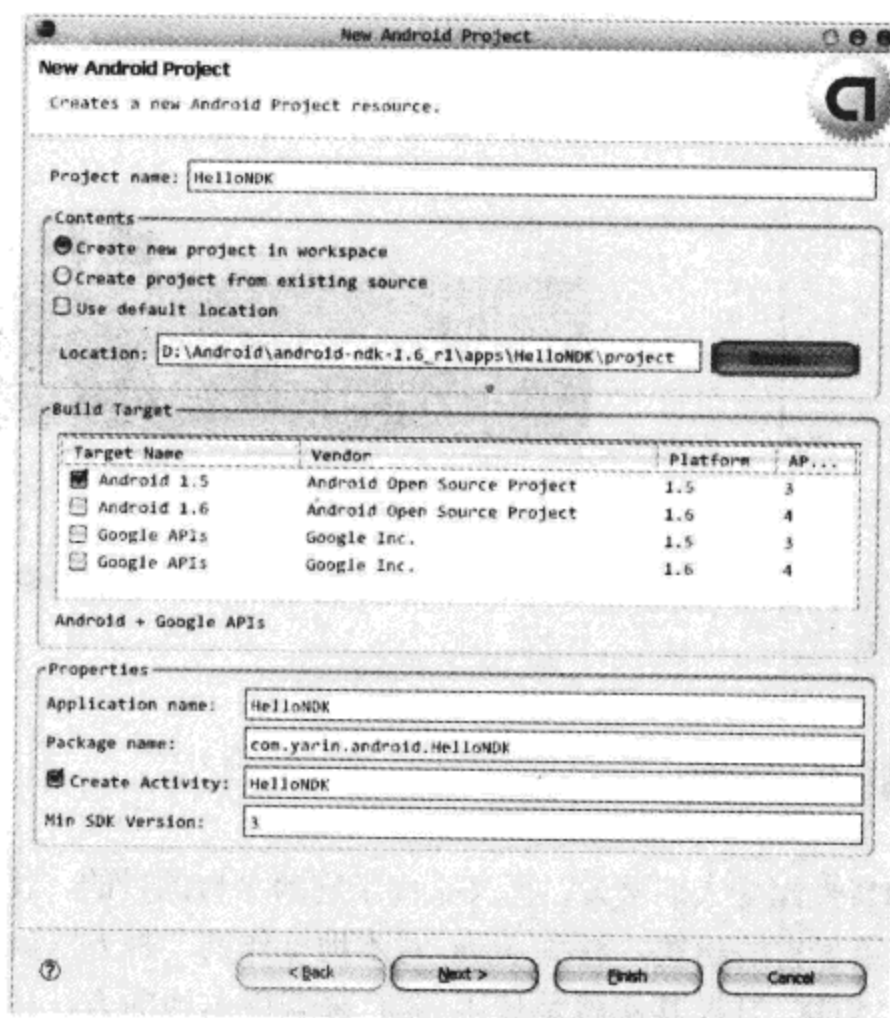


图 16-19 创建 NDK 工程

(3) 书写 Java 类。

首先创建一个 JNI 的 Java 类，如图 16-20 所示，用来声明要调用的本地方法，关键字 `native`。在这里只需要声明，不需要具体实现。

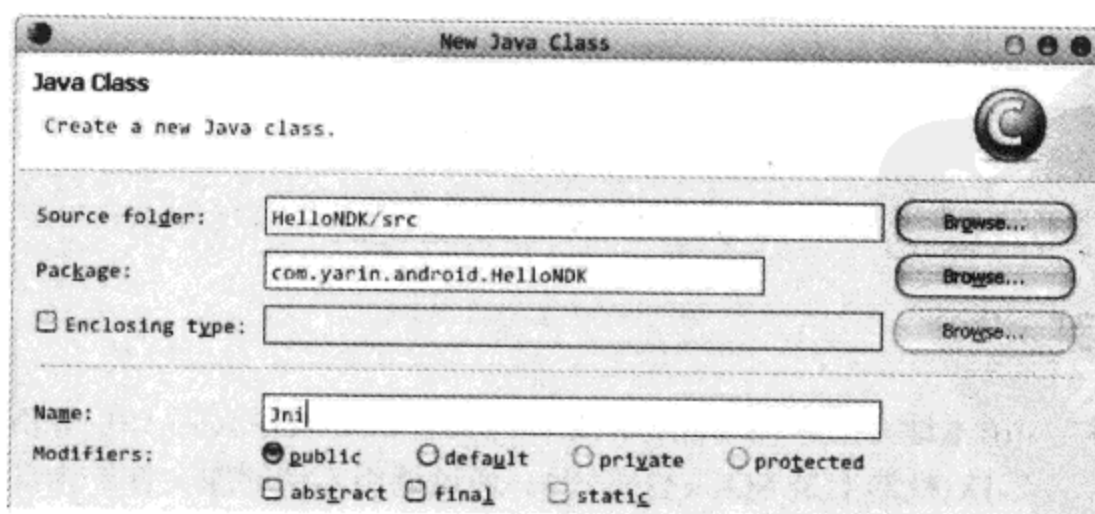


图 16-20 创建 JNI 类

代码清单 16-1 第 16 章\HelloNDK\project\src\com\yarin\android\HelloNDK\Jni.java

```
package com.yarin.android.HelloNDK;
public class Jni
{
```

```

/*声明本地方法*/
//得到一个 int 型数据
public native int getCInt();
//得到一个字符串数据
public native String getCString();
}

```

(4) 将工程目录下“src\com\yarin\android\HelloNDK”目录中的 Jni.java 文件（如图 16-21 所示）复制到工程目录中 bin 文件夹下面（如图 16-22 所示）。

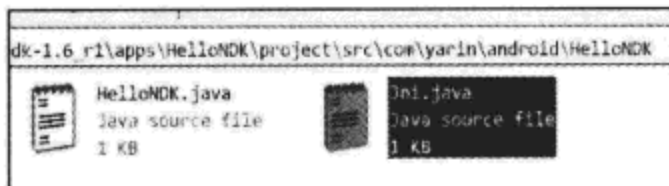


图 16-21 jni.java 文件路径

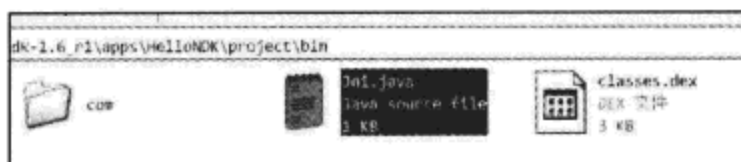


图 16-22 工程 bin 目录

(5) 从命令行进入该工程的 bin 目录，输入“javac jni.java”命令，如图 16-23 所示，生成 Jni.class 文件。

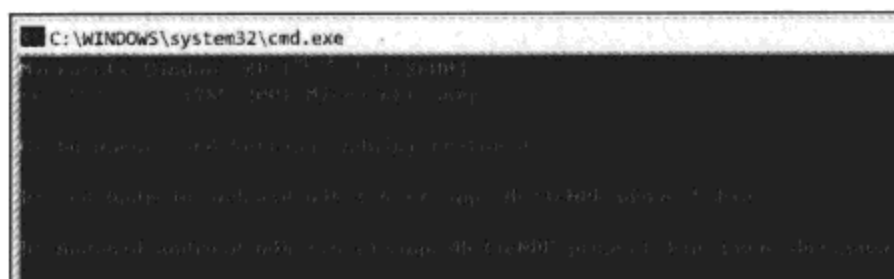


图 16-23 javac 命令

(6) 复制上一步生成的 Jni.class 文件到“project\bin\com\yarin\android\HelloNDK”目录，覆盖以前 Jni.class 文件，如图 16-24 所示。

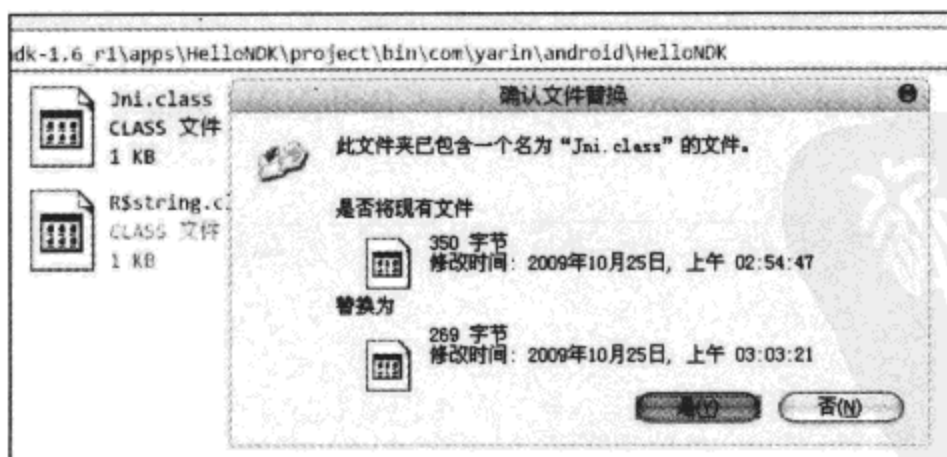


图 16-24 覆盖 Jni.class 文件

(7) 再次从命令行进入该工程的 bin 目录，输入“javah -jni com.yarin.android.HelloNDK.Jni”命令，如图 16-25 所示，此时将在 bin 目录中生成一个名为 com_yarin_android_HelloNDK_Jni.h 的 C 头文件。

(8) 在该工程的根目录中创建一个 jni 文件夹，将上一步生成的 om_yarin_android_HelloNDK_

Jni.h 文件复制到新建的 jni 文件夹下, 如图 16-26 所示。该 jni 文件夹专门用来储存 C/C++ 文件。

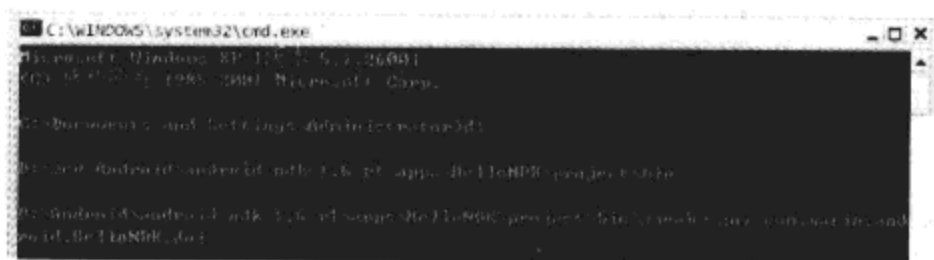


图 16-25 javah 命令

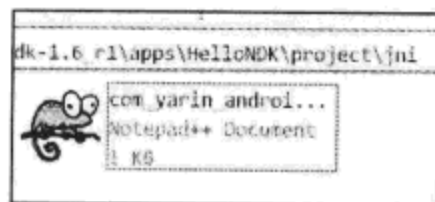


图 16-26

到这里我们已经完成了 NDK 开发中的 JNI 接口设计, 要在 Android 上运行 C/C++ 程序还需要继续学习, 下面我们将用 C/C++ 来完成这些本地方法的实现。

16.3.2 使用 C/C++ 实现本地方法

上一节我们设计好了 Java 程序中要调用的 C/C++ 的本地方法, 但是没有实现, 本节内容我们将实现这些原生的方法, 所以本节我们将接着上一节的内容继续学习。

(1) 如果你的 Eclipse 安装了 CDT 支持开发 C/C++ 程序, 那么你可以建立一个 C 工程, 指定目录到上一节中我们创建的 jni 文件夹下面, 如图 16-27 所示, 这样可以方便代码的书写, 当然你也可以不建立该工程, 直接使用文本编辑器来完成 C/C++ 代码的书写。

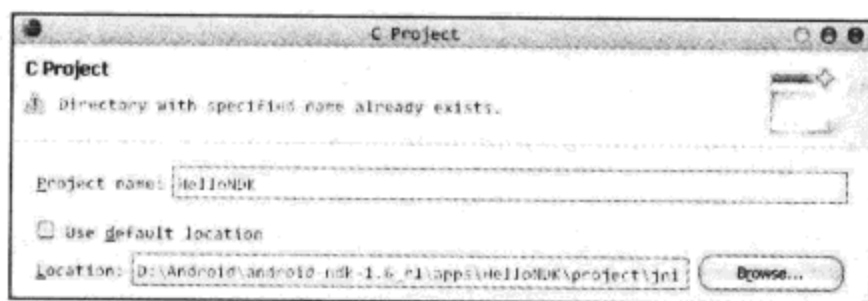
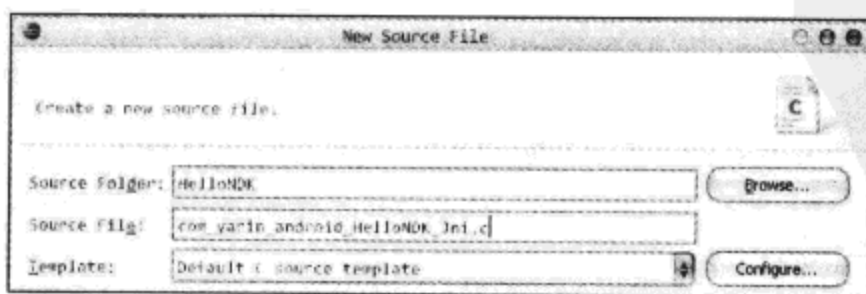


图 16-27 在 Eclipse 中建立 C 工程

(2) 在该 C 工程中新建一个 `com_yarin_android_HelloNDK_Jni.c` 文件, 用来实现这些原生方法, 如图 16-28 所示。

图 16-28 新建 `com_yarin_android_HelloNDK_Jni.c` 文件

(3) 首先打开 `com_yarin_android_HelloNDK_Jni.h` 文件, 其中包括了两个原生方法的声明, 没有实现, 如代码清单 16-2 所示。

代码清单 16-2 第 16 章\HelloNDK\project\jni\com_yarin_android_HelloNDK_Jni.h

```
#include <jni.h>
```

```

#ifndef _Included_com_yarin_android_HelloNDK_Jni
#define _Included_com_yarin_android_HelloNDK_Jni
#ifdef __cplusplus
extern "C" {
JNIEXPORT jint JNICALL Java_com_yarin_android_HelloNDK_Jni_getCInt
    (JNIEnv *env, jobject thiz);
JNIEXPORT jstring JNICALL Java_com_yarin_android_HelloNDK_Jni_getCString
    (JNIEnv *env, jobject thiz);

#ifdef __cplusplus
}
#endif
#endif

```

(4) 大家可以看出, 代码清单 16-2 中首先 Included 了<jni.h>文件, 该文件主要用来转化 Java 和 C/C++中的一些差别, 比如说数据类型等, 下面我们打开<jni.h>文件, 其中代码片段如代码清单 16-3 所示。

代码清单 16-3 jni.h

```

#ifdef HAVE_INTTYPES_H
# include <inttypes.h> /* C99 */
typedef uint8_t      jboolean; /* unsigned 8 bits */
typedef int8_t       jbyte;    /* signed 8 bits */
typedef uint16_t     jchar;    /* unsigned 16 bits */
typedef int16_t      jshort;   /* signed 16 bits */
typedef int32_t      jint;     /* signed 32 bits */
typedef int64_t      jlong;    /* signed 64 bits */
typedef float        jfloat;   /* 32-bit IEEE 754 */
typedef double       jdouble;  /* 64-bit IEEE 754 */
#else
typedef unsigned char jboolean; /* unsigned 8 bits */
typedef signed char   jbyte;    /* signed 8 bits */
typedef unsigned short jchar;   /* unsigned 16 bits */
typedef short         jshort;   /* signed 16 bits */
typedef int           jint;     /* signed 32 bits */
typedef long long     jlong;    /* signed 64 bits */
typedef float         jfloat;   /* 32-bit IEEE 754 */
typedef double        jdouble;  /* 64-bit IEEE 754 */
#endif
typedef jint          jsize;
class _jobject {};
class _jclass : public _jobject {};
class _jstring : public _jobject {};
class _jarray : public _jobject {};
class _jobjectArray : public _jarray {};
class _jbooleanArray : public _jarray {};
class _jbyteArray : public _jarray {};
class _jcharArray : public _jarray {};
class _jshortArray : public _jarray {};
class _jintArray : public _jarray {};
class _jlongArray : public _jarray {};
class _jfloatArray : public _jarray {};
class _jdoubleArray : public _jarray {};
class _jthrowable : public _jobject {};

```

看到 jni.h 文件大家应该明白 Java 和 C/C++是如何来取得联系并交换数据了, 因此我们在书写 C 代码的时候就必须按照这些规定来, 否则 Java 不会识别。代码清单 16-2 中, 就按照该规则声明

了两个需要被 Java 调用的方法，其中 JNIEXPORT 和 JNICALL 都是 JNI 的关键字，表示此函数是要被 JNI 调用的。而 jint 是以 JNI 为中介使 Java 的 int 类型与本地的 int 沟通的一种类型，我们可以视而不见，就当做 int 使用，jstring 也就同 jint 一样。函数的名称是由 Java_加上 Java 程序的 package 路径再加函数名组成的。参数中，我们也只需要关心在 Java 程序中存在的参数，至于 JNIEnv* 和 jclass 我们一般没有必要去碰它。

(5) 然后我们打开 com_yarin_android_HelloNDK_Jni.c 文件来书写 C 代码，并实现 com_yarin_android_HelloNDK_Jni.h 中声明的方法，如代码清单 16-4 所示，这里都是一些 C 的代码，大家有不明白的可以查看 C 编程，由于不在本书的范围之内，所以就不再详细解释了。

代码清单 16-4 第 16 章\HelloNDK\project\jni\com_yarin_android_HelloNDK_Jni.c

```
#include <stdio.h>
#include <stdlib.h>
#include "com_yarin_android_HelloNDK_Jni.h"
int add2()
{
    int x,y;
    x = 1000;
    y = 8989;
    x+=y;
    return x;
}
JNIEXPORT jint JNICALL Java_com_yarin_android_HelloNDK_Jni_getCInt
(JNIEnv *env, jobject thiz)
{
    return add2();
}
JNIEXPORT jstring JNICALL Java_com_yarin_android_HelloNDK_Jni_getCString
(JNIEnv *env, jobject thiz)
{
    (*env)->NewStringUTF(env, " HelloNDK---->> ");
}
```

到这里我们已经完成了 C/C++ 代码的书写，我们知道 jni 是通过调用动态链接库（dll 或者 so 文件）来实现 C/C++ 方法的调用，因此我们现在就需要将上面所书写的 C 代码编译成 SO 文件，下面我们将详细地介绍如何编译。

16.3.3 Android.mk 实现

要将我们书写出来的 C 代码编译成 SO 文件，就需要使用带 Android.mk 编译脚本，本节我们就将详细分析 Android.mk 文件的使用以及如何书写。

首先我们来看看 Android.mk 文件所存在的位置，如图 16-29 所示。

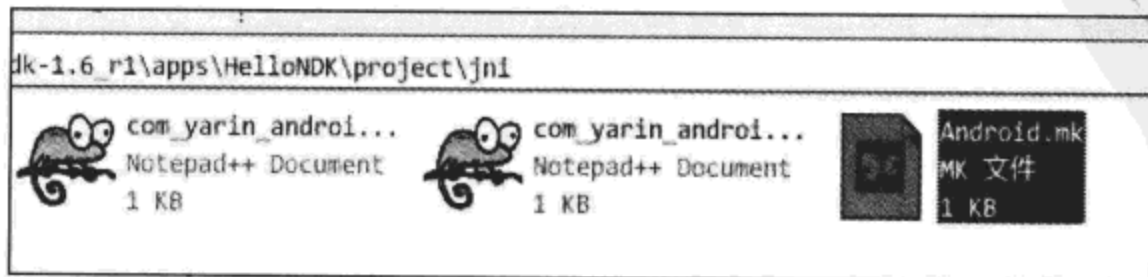


图 16-29 Android.mk 文件的位置

从图 16-29 可以看出, `Android.mk` 文件应该放置到我们的 C 工程目录, 它主要用来指定我们要编译的 SO 文件所包含的内容, 该 SO 文件包含的都是一些模块, 每个模块可以是一个静态库也可以是一个动态链接库, 只有动态链接库被安装进你的应用程序包后, 静态库才能被动态链接库使用。我们可以在每一个 `Android.mk` 文件中定义一个或多个模块, 你也可以在不同的模块中使用相同的库。编译系统会为你处理很多细节。举个例子, 不需要列出头文件以及产生在 `Android.mk` 文件中的详细属性。NDK 编译系统会自动帮你计算。这就意味着, 当有新的 NDK 版本发布后, 你可以直接从新的工具链和平台上得到支持, 而不用动手去修改你的 `Android.mk` 文件。

那么首先我们需要在上一节的工程中建立一个 `Android.mk` 文件, 如图 16-29 所示, 在其中添加如代码清单 16-5 所示的代码。

代码清单 16-5 第 16 章\HelloNDK\project\jni\Android.mk

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

include $(CLEAR_VARS)

LOCAL_MODULE := HelloNDK
LOCAL_SRC_FILES := com_yarin_android_HelloNDK_Jni.c

include $(BUILD_SHARED_LIBRARY)
```

此时 `com_yarin_android_HelloNDK_Jni.c` 中实现了两个本地方法, 将作为 JNI 的动态链接库的源文件。下面我们来分析为什么要像代码清单 16-5 所示来书写这个 `Android.mk` 文件, 可以换一种方式来写吗?

❑ `LOCAL_PATH := $(call my-dir)`

`LOCAL_PATH` 表示此时位于工程目录的根目录中, `(call my-dir)` 的功能由编译器提供, 被用来返回当前目录的地址 (这里的当前目录里包括 `Android.mk` 这个文件本身)。

❑ `include $(CLEAR_VARS)`

`CLEAR_VARS` 这个变量由编译系统提供, 并且指明了一个 GNU makefile 文件, 这个功能会清理掉所有以 `LOCAL_` 开头的内容 (例如 `LOCAL_MODULE`、`LOCAL_SRC_FILES`、`LOCAL_STATIC_LIBRARIES` 等), 除了 `LOCAL_PATH`。这句话是必须的, 因为如果所有的变量都是全局的, 所有的可控的编译文件都需要在一个单独的 GNU 中被解析并执行。

❑ `LOCAL_MODULE := HelloNDK`

`LOCAL_MODULE` 变量必须被定义, 用来区分 `Android.mk` 中的每一个模块。文件名必须是唯一的, 不能有空格。注意, 这里编译器会为你自动加上一些前缀和后缀, 来保证文件是一致的。比如: 这里表明一个动态链接库模块被命名成 “HelloNDK”, 但是最后会生成为 “lib HelloNDK.so” 文件。但是在 Java 中装载这个库时还是使用 “HelloNDK” 名称。当然如果我们使用 “IMPORTANT NOTE:”, 编译系统就不会给你加上前缀, 但是为了支持 Android 平台源码中的 `Android.mk` 文件, 也同样会生成 `libHelloNDK.so` 这样的文件。

❑ `LOCAL_SRC_FILES := com_yarin_android_HelloNDK_Jni.c`

`LOCAL_SRC_FILES` 变量必须包含一个 C 和 C++ 源文件的列表, 这些会被编译并聚合到一个模块中。

注意 这里并不需要列出头文件和被包含的文件，因为编译系统会自动为你计算相关的属性，源代码中的列表会直接传递给编译器。

C++默认文件的扩展名是“.cpp”。我们可以通过定义一个 `LOCAL_DEFAULT_CPP_EXTENSION` 变量来定义一个不同的 C 文件。不要忘记初始化前面的“.”（点），（也就是说“.cpp”可以正常工作，但是“cpp”不行）。

❑ `include $(BUILD_SHARED_LIBRARY)`

`BUILD_SHARED_LIBRARY` 这个变量是由系统提供的，并且指定给 GNU Makefile 的脚本，它可以收集所有你定义的“`include $(CLEAR_VARS)`”中以 `LOCAL_` 开头的变量，并且决定哪些要被编译，哪些应该做得更加准确。我们同样也可以使用 `BUILD_STATIC_LIBRARY` 来生成一个静态的库。如果使用 `BUILD_STATIC_LIBRARY` 编译系统便会生成一个以“`lib$(LOCAL_MODULE).a`”为文件名的文件来供动态库调用。

当然这些是我们上一个例子中所用到的变量，还有如下一些没有被用到。

❑ `TARGET_ARCH`

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

❑ `TARGET_PLATFORM`

Android 平台的名字在 `Android.mk` 文件中被解析，比如：“`android-1.5`”。

❑ `TARGET_ARCH_ABI`

`TARGET_ARCH_ABI` 指 CPU+ABI 的名字，目前，只有 Arm 平台被支持，这也就意味着目前只有 Arm5 以及更高版本的 CPU，并只支持“`softfloat`”或者其他的 ABIS。未来的 NDK 版本中得到支持，它们会有一个不同的名字，注意所有基于 ARM 的 ABI 都会有一个“`TARGET_ARCH`”被定义给 `arm`，但也有可能有不同的“`TARGET_ARCH_ABI`”。

❑ `TARGET_ABI`

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

下面都是 GNU 编译出来的宏，而且它们都必须使用“`$(call <function>)`”才能返回文字化的信息。

❑ `all-subdir-makefiles`

`all-subdir-makefiles` 返回一个 `Android.mk` 文件所在位置的列表，以及当前的 `my-dir` 的路径。比如：`include $(call all-subdir-makefiles)`。

❑ `this-makefile`

返回当前 Makefile 的路径（也就是那个功能被调用了）。

❑ `parent-makefile`

返回 Makefile 的包含树，也就是包含 Makefile 当前的文件。

这里我们分析了 `Android.mk` 文件中常用的一些宏以及变量，当然还不只是这些，我们同样可以自己定义要使用的类型，大家可以参考 NDK 自带例子中 `Android.mk` 文件的定义。

16.3.4 Application.mk 实现

要将 C/C++ 代码编译为 SO 文件，光有 `Android.mk` 文件还不行，还需要一个 `Application.mk` 文件，本节我们就将学习如何完成 `Application.mk` 文件。

同样我们还是先看看 `Application.mk` 文件所存放的位置，如图 16-30 所示。

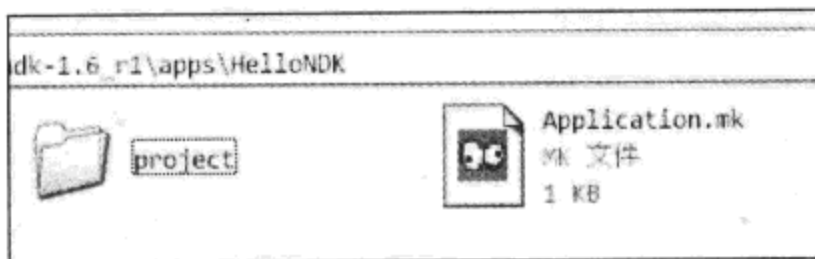


图 16-30 `Application.mk` 文件存放位置

从图 16-30 中可以看出，`Application.mk` 文件存放的位置处于 NDK 工程的根目录，`Application.mk` 就是用来描述应用程序中所需要的原生的模块（也就是静态库和动态链接库）。每一个 `Application.mk` 都必须放在应用目录下的子目录，例如：`$NDK/apps/HelloNDK/Application.mk`，`HelloNDK` 是一个 NDK 编译系统，而描述应用程序的简短的名称（在最终生成的动态链接库中，不需要包含这样的名称）。作为 GNU Makefile 的一部分，`Application.mk` 必须要定义以下部分：

1. APP_MODULES

`APP_MODULES` 变量是强制性的，并且会列出所有你的应用所需要的模块（通过 `Android.mk` 文件来详细地描述）。

2. APP_PROJECT_PATH

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

下面我们在 `HelloNDK` 工程中创建一个 `Application.mk` 文件，其内容如代码清单 16-6 所示。

代码清单 16-6 第 16 章\HelloNDK\Application.mk

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

代码清单 16-5 中定义了工程的路径为 `$(call my-dir)/project`，而要编译的模块则是 `HelloNDK`，因此这就需要和 `Android.mk` 文件中 `LOCAL_MODULE:= HelloNDK` 有一个对应的关系，这样编译系统才会找到我们要编译的库的源文件。

另外该文件中还可以存在 `APP_OPTIM` 变量，这个可选的变量可以定义为“发布版”或者是“测试版”。这是用来在你编译项目模块的时候，改进你的优化等级。`Release` 模式时，也会默认生成更加优化的二进制文件。“调试模式”会生成一些没有优化过的二进制文件，但是更容易检测出一些 Bug。这里需要注意一下，其实两种模式都能检测出 Bug。但是相对来说 `Release` 模式提供的信息较少。在我们清除 Bug 的过程中，有一些变量被优化，或者根本就无法被检测出来，代码的重新排序会让这些代码变得更加难以阅读，并且让这些轨迹更加不可靠。`APP_CFLAGS` 则是当要编译模块中有任何 C 文件的时候，C 编译器的信号就会被发出。这里可以在你的应用中需要这些模块时，进行编译的调整，这样就不用直接更改 `Android.mk` 文件本身了。

16.3.5 编译 C\C++ 代码

到这里我们已经都准备好了，现在可以来编译这些 C\C++ 代码，生成可以供 Java 调用的 SO 文件了。编译非常简单，首先启动 Cygwin，进入 NDK 根目录，输入“`make APP=HelloNDK`”命令，出现如图 16-31 所示画面则表示编译成功。同样由于笔者已经编译过了，所以使用了“`make APP=HelloNDK -B`”命令强制重新编译。



图 16-31 编译 SO 文件

从图 16-31 可以看出，生成了名为“`libHelloNDK.so`”的文件，并安装到工程目录下“`libs\armeabi`”目录中，如图 16-32 所示。

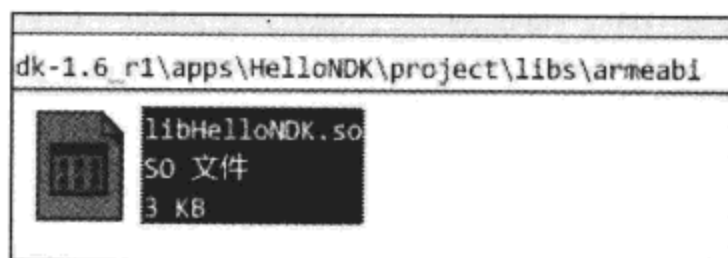


图 16-32 libHelloNDK.so 文件

最后在 Java 工程中加入如代码清单 16-7 所示代码，装载动态库“`libHello.so`”，并调用本地原生方法来取得 C\C++ 中的数据，并显示在一个 `TextView` 中。

代码清单 16-7 第 16 章\HelloNDK\project\src\com\yarin\android\HelloNDK\HelloNDK.java

```
public class HelloNDK extends Activity
{
    //装载动态库“libHello.so”
    static
    {
        System.loadLibrary("HelloNDK");
    }
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Jni jni = new Jni();
        TextView textView = new TextView(this);
        //调用原生方法
        textView.setText(jni.getCString()+Integer.toString(jni.getCInt()));
        setContentView(textView);
    }
}
```

注意 如代码清单 16-7 所示, 在装载动态库时, 不需要加后缀名, 由系统自动判断, 并且不需要加入前缀 lib, 如: `System.loadLibrary("HelloNDK")`。

运行该工程, 效果如图 16-33 所示, 成功调用了 libHelloNDK, so 中的 `getCString()` 和 `getCInt()` 方法。到这里我们已经完成了一个简单的 NDK 原生开发工程。



图 16-33 HelloNDK 效果

16.4 Android NDK 中使用 OpenGL

Android 1.6 NDK 中我们可使用 OpenGL ES 1.1 的库, 这说明我们可以用 NDK 来开发 OpenGL 的 3D 项目了, 前面我们用 Android SDK 也开发过 OpenGL 程序, 但是相信利用 NDK 开发的原生程序在速度、效率等方面肯定要比 SDK 开发得好。本节我们将用 OpenGL ES 2005 编程挑战赛的一个项目 San angeles 来演示 NDK 中的 OpenGL 开发。San angeles 是一个 OpenGL ES 的自运行演示程序, 也是一个很好的测试程序, 运行效果如图 16-34 所示。



图 16-34 San angeles 效果

下面我们来看看该项目是如何实现的, 在 Android 1.6 NDK 中可以找到 San-angeles 项目, 具体实现请参见本书所附源码: 第 16 章\San-angeles。

首先要显示一个 OpenGL 视图就需要实现一个 GLSurfaceView 类, 而 GLSurfaceView 中需要实

现一个 Render 接口, 另外要使用原生方法, 我们需要先声明, 如代码清单 16-8 所示。

代码清单 16-8 第 16 章\san-angeles\project\src\com\example\SanAngeles\DemoActivity.java

```
public class DemoActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mGLView = new DemoGLSurfaceView(this);
        setContentView(mGLView);
    }
    protected void onPause() {
        super.onPause();
        mGLView.onPause();
    }
    protected void onResume() {
        super.onResume();
        mGLView.onResume();
    }
    private GLSurfaceView mGLView;
    //装在动态链接库
    static {
        System.loadLibrary("sanangeles");
    }
}

class DemoGLSurfaceView extends GLSurfaceView {
    public DemoGLSurfaceView(Context context) {
        super(context);
        mRenderer = new DemoRenderer();
        setRenderer(mRenderer);
    }
    public boolean onTouchEvent(final MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            nativePause();
        }
        return true;
    }
    DemoRenderer mRenderer;
    private static native void nativePause();
}

class DemoRenderer implements GLSurfaceView.Renderer {
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        nativeInit();
    }
    public void onSurfaceChanged(GL10 gl, int w, int h) {
        //gl.glViewport(0, 0, w, h);
        nativeResize(w, h);
    }
    public void onDrawFrame(GL10 gl) {
        nativeRender();
    }
    //声明原生方法
    private static native void nativeInit();
    private static native void nativeResize(int w, int h);
    private static native void nativeRender();
    private static native void nativeDone();
}
```

在这个 Java 工程中很简单, 就这样便可以了, 下面重要的就是如何实现这几个原生的方法, 根据上一节所讲的, 我们将 C 代码放入工程目录上的 `Jni` 文件夹下。如代码清单 16-9 所示, 是该工程中几个原生方法的实现。

代码清单 16-9 第16章\san-angeles\project\jni\app-android.c

```
void
Java_com_example_SanAngeles_DemoRenderer_nativeInit( JNIEnv* env )
{
    importGLInit();
    appInit();
    gAppAlive    = 1;
    sDemoStopped = 0;
    sTimeOffsetInit = 0;
}
void
Java_com_example_SanAngeles_DemoRenderer_nativeResize( JNIEnv* env, jobject thiz,
jint w, jint h )
{
    sWindowWidth  = w;
    sWindowHeight = h;
    __android_log_print(ANDROID_LOG_INFO, "SanAngeles", "resize w=%d h=%d", w, h);
}
void
Java_com_example_SanAngeles_DemoRenderer_nativeDone( JNIEnv* env )
{
    appDeinit();
    importGLDeinit();
}
void
Java_com_example_SanAngeles_DemoGLSurfaceView_nativePause( JNIEnv* env )
{
    sDemoStopped = !sDemoStopped;
    if (sDemoStopped) {
        sTimeStopped = _getTime();
    } else {
        sTimeOffset -= _getTime() - sTimeStopped;
    }
}
void
Java_com_example_SanAngeles_DemoRenderer_nativeRender( JNIEnv* env )
{
    long curTime;
    if (sDemoStopped) {
        curTime = sTimeStopped + sTimeOffset;
    } else {
        curTime = _getTime() + sTimeOffset;
        if (sTimeOffsetInit == 0) {
            sTimeOffsetInit = 1;
            sTimeOffset      = -curTime;
            curTime          = 0;
        }
    }
    appRender(curTime, sWindowWidth, sWindowHeight);
}
```

剩下的事情就是通过 C 语言来实现 OpenGL 编程了, 大家可以参见该工程下的 `demo.c` 文件, 最后我们来编译生成这个 SO 文件, 首先我们看看如何编写 `Android.mk` 文件, 如代码清单 16-10

所示。

代码清单 16-10 第 16 章\san-angeles\project\jni\Android.mk

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

include $(CLEAR_VARS)

LOCAL_MODULE := sanangeles

LOCAL_CFLAGS := -DANDROID_NDK \
                -DDISABLE_IMPORTGL

LOCAL_SRC_FILES := \
    importgl.c \
    demo.c \
    app-android.c \

LOCAL_LDLIBS := -lGLESv1_CM -ldl -llog
```

```
include $(BUILD_SHARED_LIBRARY)
```

最后还需要完成 **Application.mk** 文件，该文件非常简单，编写如下代码即可。

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

下面就是编译这些 C 代码生成 SO 文件，具体操作步骤如图 16-35 所示。



图 16-35 编译 san-angeles

编译完成之后，在模拟器中运行该工程，成功的话就会看到本节开始时的图 16-34。Android NDK 中有了 OpenGL 库，不久在 Android 上必然会出现大量的 3D 游戏，所以 OpenGL 是 NDK 开发中较为重要的，也是非常有用的，希望大家都能好好掌握。

16.5 小结

本章我们主要讲述了 Android NDK 的相关知识，从 NDK 的简单介绍到环境的安装以及 NDK 的完整开发流程，最后通过一个 San-angeles 项目演示了 NDK 中的 OpenGL 开发。很显然，NDK 开发可以提高程序的效率，但是同时也大大增加了开发的难度，因此大家在策划时就应该慎重考虑是否使用 NDK，比如项目中包含了大量的逻辑计算或者 3D 特效，通过 Android SDK 开发可能会严重影响程序运行的速度，这时便可以考虑使用 NDK 原生开发。目前 NDK 还处于初级阶段，官方也表示后期将会支持更多的库，所以我们就从初级开始，把握其特点，开发出更多优秀的应用程序。

Android 脚本环境

上一章我们学习了 NDK，我们可以使用 NDK 在 Android 应用中通过 JNI 方式来实现原生开发，这就使得众多的 C 爱好者得到了满足，但是 Android 的目标并不只是这些，因为还有很大一部分人都喜爱脚本语言，并且脚本语言比 C/C++、Java 更加简单，就算没有太多经验的人都可以使用，所以 Android 同样也支持了脚本环境，这就是我们本章即将介绍的 Android ScriptingEnvironment（简称 ASE）项目。

17.1 Android 脚本环境简介

脚本语言(Scripting language)是电脑程式语言，因此也能让开发者借以编写出让电脑听命行事的程式。以简单的方式快速完成某些复杂的事情通常是创造脚本语言的重要原则，基于这项原则，使得脚本语言通常比C语言、C++语言或 Java 之类的系统程式语言要简单容易，也让脚本语言另有一些属于脚本语言的特性：

- ☐ 语法和结构通常比较简单。
- ☐ 学习和使用通常比较简单。
- ☐ 通常以容易修改程式的“直译”作为执行方式，而不需要“编译”。
- ☐ 程式的开发产能优于执行效能。

一个脚本可以使得本来要用键盘进行的交互式操作自动化。一个 Shell 脚本主要由原本需要在命令行输入的命令组成，或在一个文本编辑器中，用户可以使用脚本来把一些常用的操作组合成一组序列，用来书写这种脚本的语言叫做脚本语言。很多脚本语言实际上已经超过简单的用户命令序列的指令，还可以编写更复杂的程序，使用脚本语言可以使得很多程序变得简单化。而 Android 平台同样也支持大部分脚本语言，这个 ASE 项目就支持了目前主流的脚本语言，包括：Python、Lua、BeanShell、Perl 等，当然后续版本还会支持 Ruby 和 JavaScript，图 17-1 便是我们在 Android 上编写的一段 Python 脚本语言片段。

ASE 为 Android 系统带来了脚本语言的支持，通过它我们可以编辑和执行脚本，甚至和脚本解释器做出交互。脚本可以访问多数的 Android API，有了这个大大简化了的接口，我们可以很方便的实现以下功能：

- ☐ 操作 Intents
- ☐ 启动 Activities
- ☐ 拨打电话
- ☐ 发送短信

- ❑ 扫描条形码
- ❑ 获取地理位置和传感器数据
- ❑ 使用 Text-To-Speech(TTS)

当然还有更多的功能，这里不一一列举。脚本可以在终端中来交互地执行，或者作为后台 Service 启动，也可以通过 Locale 命令来启动。这里我们要明白一个事实，因为在网上看到很多人说“这下我们不用学 Java 了，直接用脚本语言来开发应用”，其实这样想并不正确，虽然脚本可以用几行指令完成一个特殊的操作，几乎会点编程的人就能用，但是真正复杂的应用包括逻辑计算、UI 界面，这些都需要通过 Java 来实现，大家可以把脚本语言想象成 Android 上的一些插件，可以简化我们很多工作，为我们的应用程序服务，共同来完成一项功能，使得我们的应用更加完善。



```

sayweather.py

"""Speak the weather."""
__author__ = 'T.V. Raman <raman@google.com>'
__copyright__ = 'Copyright (c) 2009, Google Inc.'
__license__ = 'Apache License, Version 2.0'

import android
import weather

def say_weather(droid):
    """Speak the weather at the current location."""
    print 'Finding ZIP code.'
    location = droid.
    getLastKnownLocation()[0]['result']
    addresses = droid.
    geocode(location['latitude'],
    location['longitude'])
    zip =
    addresses['result'][0]['postal_code']
    if zip is None:
        msg = 'Failed to find location.'

```

图 17-1 Python 脚本语言片段

目前脚本语言的分类也比较多，但是任何一种语言都有自己的优点，同时也存在一些不足，当然了，Android 平台上的 ASE 最主要的优点就是灵活方便，因为 Android 的开发环境非常易于使用，但是你必须要在电脑前完成所有工作。而 ASE 则使你在任何所需情况下，都能快速地在设备上使用高级脚本语言尝试各种想法，不受时间、地点的限制，只要有手机（或者其他设备）我们就可以编写出脚本程序，同时由于 ASE 还处于初步发展的阶段，还不能支持 Android 中所有的 API，这给编写程序带来了一定的难度，但是 ASE 项目的可扩展性很高，大家可以根据自己的需要来扩展功能更加完善的 API 接口。

17.2 Android 脚本环境安装

通过上一节的学习，我们对 ASE 有了一定的了解，这一节我们将讲述如何在 Android 上安装这样一个脚本环境，ASE 的安装非常简单，下面我们就一步一步地来学习如何安装 ASE 的环境。

由于 ASE 是一个开源的项目，所以我们可以到 Google Code 上下载它的安装包或者源文件，首先进入 ASE 项目的 SVN 地址“<http://code.google.com/p/android-scripting/>”，如果要下载 ASE 的安装包以及一些例子程序我们可以点击“Downloads”标签，选择“ase_r13.apk”文件（ASE 环境安装包）下载，如图 17-2 所示。如果需要下载 ASE 项目的源文件，那么可以点击“Source”标签来下载，如图 17-3 所示。这里需要使用 SVN 工具才可以下载。

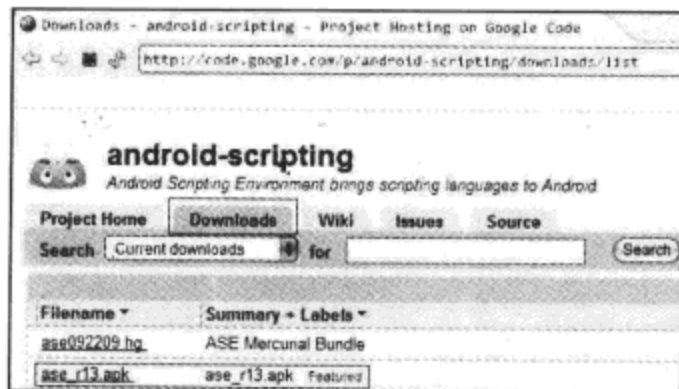


图 17-2 ase_r13.apk 下载

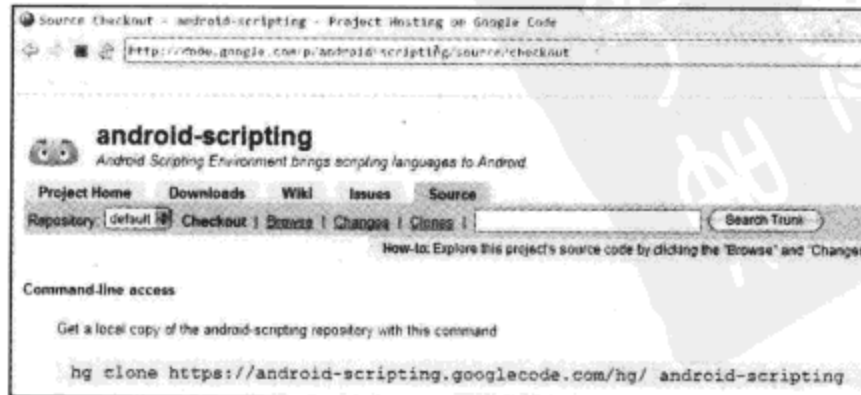


图 17-3 ASE 源文件下载

下载之后我们就可以安装了，ASE 的安装和一般的应用程序安装一样，这里我们首先启动模拟器，然后从命令行进入 Android SDK 目录下的 tools 目录，然后输入“adb install”命令进行安装，如图 17-4 所示，安装完成之后会在模拟器上面出现如图 17-5 所示的图标，这表示安装成功，如果没有安装成功，可以根据错误的提示内容重新安装。



图 17-4 安装 ASE

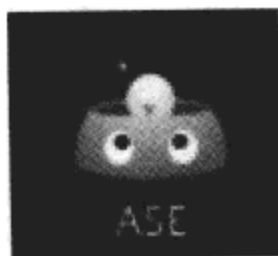


图 17-5 ASE 图标

到这里我们已经成功地安装了 ASE，这时我们点击如图 17-5 所示的图标，启动 ASE，此时启动之后里面是空白的，什么东西都没有，如图 17-6 所示，提示我们可以按菜单键来添加脚本环境和终端以及解释器等操作。

我们要想在 Android 上运行脚本程序，首先需要安装解释器了，点击“Menu”键，选择“Add Interpreter”菜单，如图 17-7 所示，选择添加脚本语言的解释器。

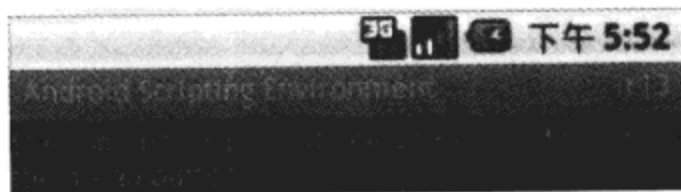


图 17-6 初次启动 ASE



图 17-7 添加解释器

当选择脚本解释器之后，程序自动开始从网站上下载并安装，正常情况下会出现如图 17-8 所示的界面。

安装之后就出现了很多 ASE 项目自带的例子，如图 17-9 所示，根据不同的后缀名我们可以区分出是哪一种脚本语言。

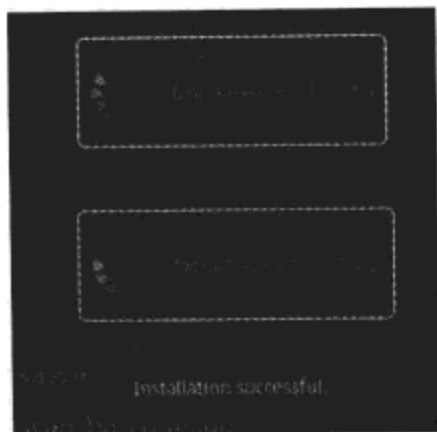


图 17-8 安装解释器



图 17-9 脚本程序列表

当然这些自带的脚本程序都是可以运行的，比如我们点击“hello_user.lua”就会要求用户输入一个用户名，如图 17-10 所示。输入之后就会在屏幕上以 Toast 方式显示一个“Hello xxx”的字符串，如图 17-11 所示。



图 17-10 输入用户名



图 17-11 显示 Toast 信息

到这里，说明我们的脚本程序已经能够运行，脚本环境已经安装成功。当然细心的朋友可能已经注意到如何编写和修改 ASE 自带的脚本程序呢？要修改一个脚本程序，我们就点击它，它出现终端界面时，我们点击“Menu”键，就会弹出菜单，大家可以去试试这些菜单的作用，这里我们选择“Exit&Edit”菜单来编辑该脚本程序，如图 17-12 所示。

下面我们看看如何编辑一个新的脚本程序，同样很简单，在主界面上点击“Menu”键出现菜单，选择“Add Script”菜单，然后弹出界面选择要添加的脚本程序的种类，然后写入脚本程序的名字及内容，点击“Menu”键，选择“Save”或者“Save&Run”菜单来保存或者保持并运行，如图 17-13 所示。

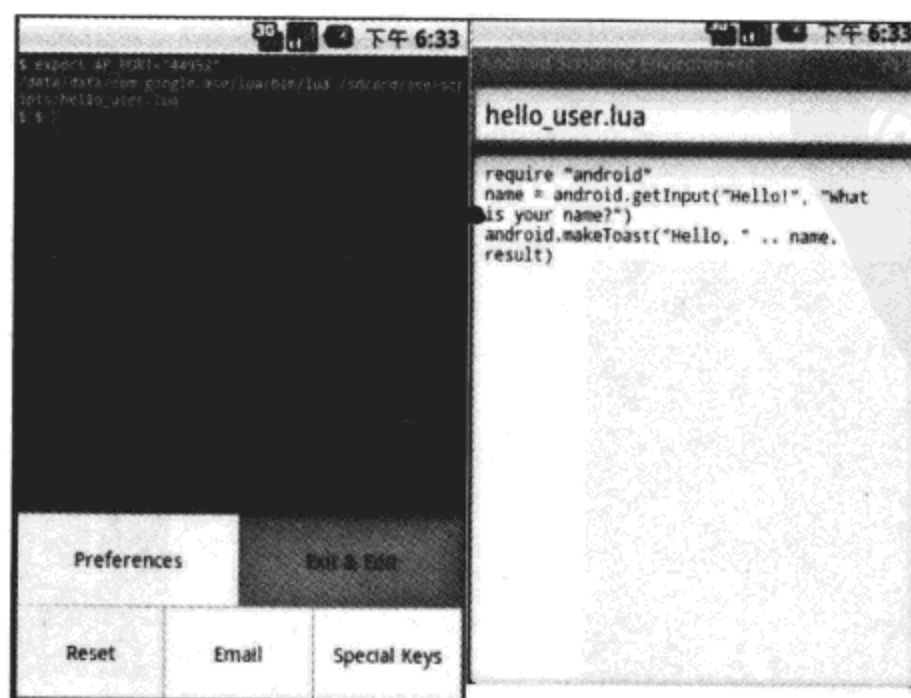


图 17-12 修改脚本程序



图 17-13 新建脚本程序

我们已经完成了 ASE 环境的搭建，也熟悉了 ASE 的操作方式，下面我们就将学习如何在 Android 上利用 ASE 来开发脚本程序。

17.3 如何编写 Android 脚本程序

通过前两节的学习，我们对 Android 脚本环境有了一定的了解，脚本程序可以直接在手机上写，而不需要使用电脑等工具。那么 ASE 是如何来调用 Android API 的呢？继续学习本节内容你就会得到答案。

ASE 通过两种方式来访问 Android API。目前我们可以通过 ASE 使用 Python 及 Lua 来运行脚本，这些都是本地应用，它们通过 JSON-RPC 来访问 Android API。

另外由于 Android 本身基于 Java，因此我们还可以运行基于 JVM 的语言。目前可以使用的是 BeanShell——Java 语言的动态版本。基于 JVM 的解释器就在 ASE 内部加载，无需间接的 RPC 调用。

当然 ASE 也不仅仅满足当前，还计划让更多的 JVM 语言运行在 Android 上。虽然 JRuby 早就能运行在 Android 上了，但却存在一些问题，首当其冲的就是 Android 使用了 Dalvik VM（Android 应用被编译成 Dalvik VM 字节码）。由于 ASE 是一个开源项目，所以如果大家感兴趣也可以来扩展这个项目，使其功能更加强大。

对于 BeanShell 这样的脚本语言，则可以直接访问 Android 的 Java API。出于简化的目的，ASE 提供了 AndroidFacade 类。而对其他像 Python 和 Lua 这样的语言，API 是通过使用 JSON RPC 调用在代理上暴露出来的。当然，这意味着只有被 AndroidFacade 和 AndroidProxy 封装了的那部分 API 才能被 Python 和 Lua 的交叉编译解释器所使用。值得庆幸的是，AndroidFacade 和 AndroidProxy 类都很容易扩展。因此我们要编写脚本语言就可以先查看 AndroidFacade 和 AndroidProxy 类，AndroidFacade 和 AndroidProxy 类位于源代码中“com\google\ase”目录下，如图 17-14 所示。

打开 AndroidFacade 类我们可以看到里面封装了很多方法，这些方法的实现都很简单，全部调用了 Android 的 API 来实现功能，而这些方法正是我们在编写脚本语言时所使用的方法。有了这

些方法我们就可以通过任何一种目前所支持的脚本语言来编写一些 Android 的应用，下面我们来看看 AndroidFacade 中究竟提供了什么方法，我们可以使用 AndroidFacade 来编写什么样的应用程序？如表 17-1 所示。

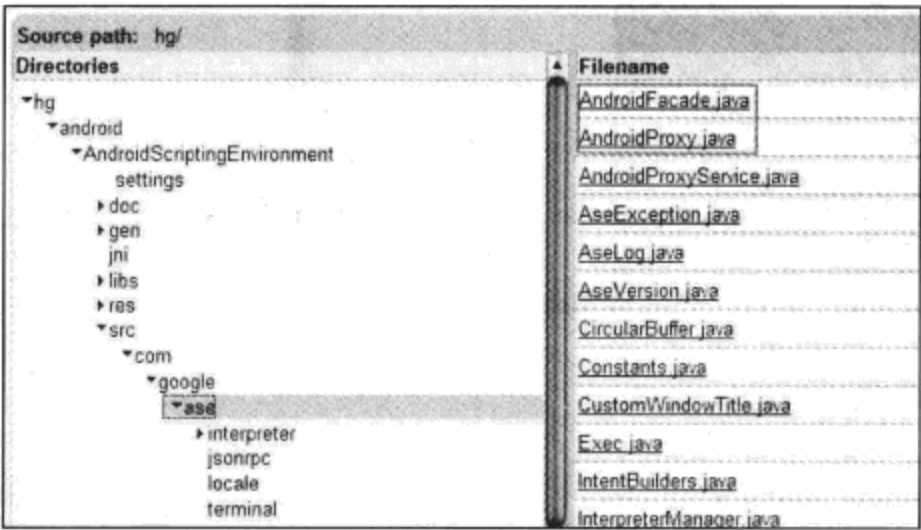


图 17-14 ASE 项目目录结构

表 17-1 AndroidFacade 方法

方 法 名	功 能
speak	通过 TTS 提供语音信息
startTrackingPhoneState	开始跟踪电话的状态
stopTrackingPhoneState	停止跟踪电话的状态
readPhoneState	查看电话当前的状态——返回当前电话的状态和来电号码
startSensing	启动传感器
stopSensing	停止传感器
ReadSensors	查看传感器所得到的信息
startLocating	开始定位
stopLocating	停止定位
readLocation	取得定位信息
getLastKnownLocation	得到最后一次成功定位的信息
geocode	更具经度和纬度等数据得到地址信息
getRingerVolume	得到铃声的音量
setRingerVolume	设置铃声音量
startActivityResult\startActivity	启动一个 Activity，和 Android API 中的一样
sendTextMessage	发送短信息
vibrate	震动的频率
setRingerSilent	设置铃声为静音
setWifiEnabled	启用 wifi
makeToast	设置一个 Toast

(续)

方 法 名	功 能
onActivityResult	结束一个 Activity
getInput	得到输入的内容, 会调用一个输入对话框
notify	产生一个通知
dial	拨号盘
dialNumber	拨号
call	调用一个地址
callNumber	呼叫号码
view	开启一个 View
map	调用一个 Map
showContacts	显示联系方式
email	调用邮件客户端
scanBarcode	扫描条码
captureImage	截图 (捕获图像)
webSearch	网页搜索
exit	取消、退出
exitWithResultOk	点击了确定键
exitWithResultCanceled	点击了取消键

当然这些也不是全部的方法, 大家可以参考该类的其他方法, 这几个方法还不够用, 但是大家看到 ASE 项目源代码之后就会明白, `AndroidFacade` 类都是很好扩展的, 大家可以随意扩展。有了这些方法我们下面就来看一段脚本程序, 当手机正面朝下放置时禁用铃声。

首先我们来看看 Lua 脚本语言如何实现这一功能, 如代码清单 17-1 所示。

代码清单 17-1 Lua 脚本示例

```
require "android"
--开启传感器
android.startSensing()
android.sleep(1)
silent = false
while true do
    --得到传感器信息
    s = android.readSensors()
    --取得传感器信息后判断是否正面朝下
    facedown = s.result and s.result.zforce and s.result.zforce > 9
    --如果是正面朝下
    if facedown and not silent then
        android.vibrate()
        --设置铃声模式静音
        android.setRingerSilent(true)
        silent = true
    --如果正面朝上
```



```

        elseif not facedown and silent then
            android.setRingerSilent(false)
            silent = false
        end
        android.sleep(1)
    end
end

```

代码清单 17-1 中主要用了我们上面所说的传感器和设置铃音模式等方法, 这样很简单的几行程序就可以实现我们所需要的——当手机正面朝下时自动更改铃声模式为静音。下面我们再来看看使用 Python 语言如何来实现一个简易的聊天程序, 如代码清单 17-2 所示。

代码清单 17-2 Python 脚本示例

```

import android, xmpp
"""设置服务器, 端口"""
_SERVER = 'talk.google.com', 5223
class SayChat(object):
    def __init__(self):
        self.droid = android.Android()
        """取得输入的用户名和密码"""
        username = self.droid.getInput('Username')['result']
        password = self.droid.getInput('Password')['result']
        jid = xmpp.protocol.JID(username)
        self.client = xmpp.Client(jid.getDomain(), debug=[])
        """连接服务器"""
        self.client.connect(server=_SERVER)
        """注册回调"""
        self.client.RegisterHandler('message', self.message_cb)
        if not self.client:
            """连接失败"""
            print 'Connection failed!'
            return
        auth = self.client.auth(jid.getNode(), password, 'botty')
        if not auth:
            print 'Authentication failed!'
            return
        self.client.sendInitPresence()
    def message_cb(self, session, message):
        jid = xmpp.protocol.JID(message.getFrom())
        username = jid.getNode()
        text = message.getBody()
        """通过 TTS 来实现语音对话"""
        self.droid.speak('%s says %s' % (username, text))
    def run(self):
        try:
            while True:
                self.client.Process(1)
        except KeyboardInterrupt:
            pass
        saychat = SayChat()
        saychat.run()

```

同样, Python 脚本语言也通过这么几行代码就实现了一个简易的聊天程序, 的确比我们通过 Java 来实现简单了很多, 所以脚本语言给我们开发 Android 应用带来了很大的方便。但是前提是你需要学会这些脚本语言。

17.4 小结

到这里，本章的内容就结束了，如果你熟悉脚本语言的编程，那么就可以顺利地利用 ASE 在 Android 上实现脚本编程。由于 ASE 支持目前大部分的脚本语言，所以这里我们就不再一一讲解了，这也不属于本书的内容，大家可以在 17.2 节所说的 ASE 项目的网站上下载到每一种脚本语言的示例程序，祝大家编程愉快。

