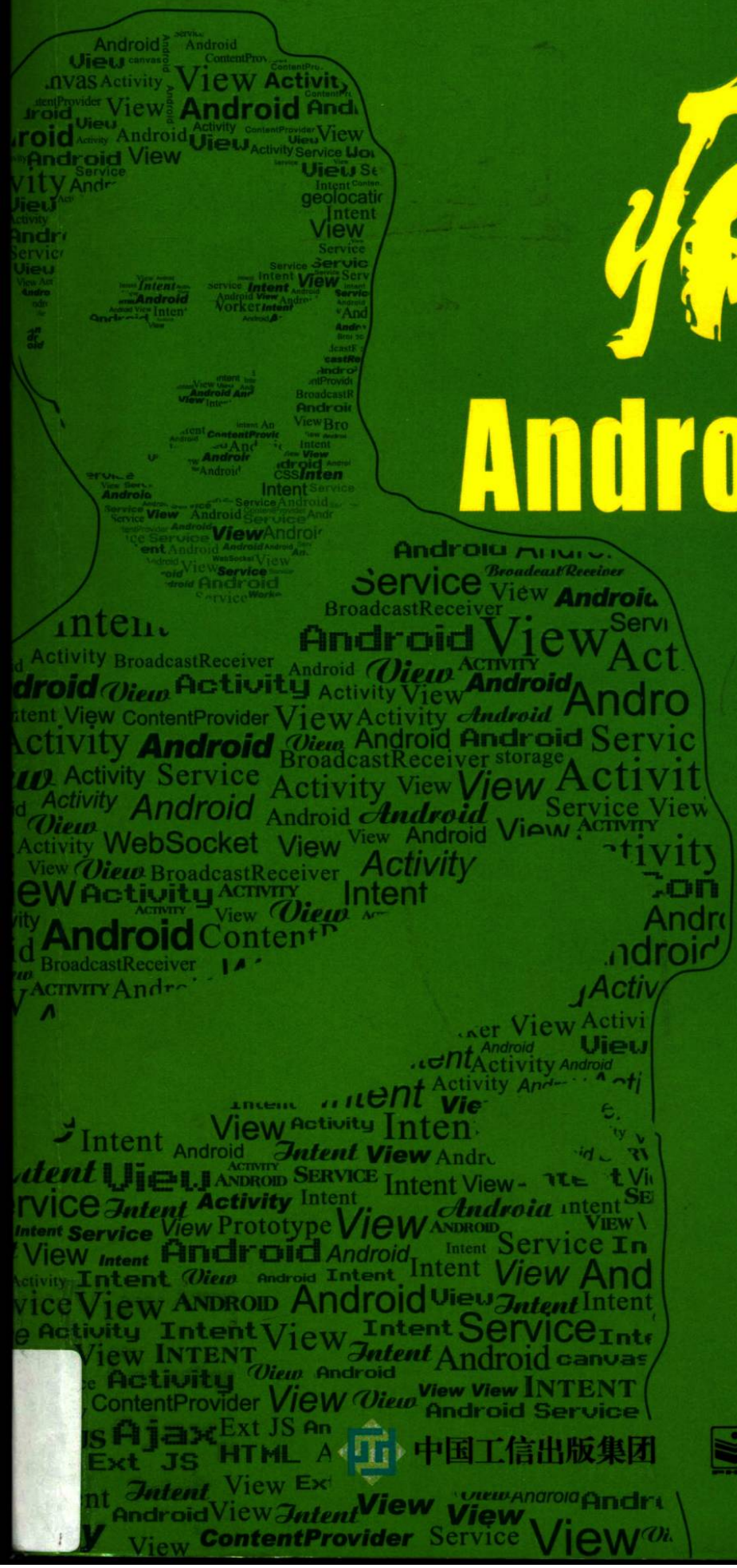
Broadview
www.broadview.com.cn

《疯狂Android讲义》自面世以来在各大网店、书城Android类图书销量中独占鳌头！

疯狂软件教育 www.fkjava.org 标准教材

疯狂Android讲义（第3版）



李刚 编著

疯狂源自梦想
技术成就辉煌

疯狂源自梦想
技术成就辉煌

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

疯狂源自梦想 技[illegible]

看得懂 学得会 做得出

1. 知识全面，覆盖面广

本书深入阐述了Android应用开发的Activity、Service、BroadcastReceiver与ContentProvider四大组件，并详细介绍了Android全部图形界面组件的功能和用法，Android各种资源的管理与用法，Android图形/图像处理，事件处理，Android输入/输出处理，音频/视频等多媒体开发，OpenGL ES开发，网络通信，传感器和GPS开发等内容，全面覆盖Android官方指南，在某些内容上更加具体、深入。

2. 内容实际，实用性强

本书并不局限于枯燥的理论介绍，而是采用了“项目驱动”的方式来讲授知识点，全书近百个实例，几乎每个知识点都可找到对应的参考实例。本书最后还提供了“合金弹头”“电子拍卖系统Android客户端”两个应用，具有极高的参考价值。

3. 讲解详细，上手容易

本书保持了“疯狂Java体系”的一贯风格：操作步骤详细，编程思路清晰，语言平实。只要读者有一定的Java编程基础，阅读本书将可以很轻松地上手Android应用开发；学习完本书最后的两个案例后，读者即可完全满足企业中实际Android应用开发的要求。

阅读此书有任何技术问题，都可以登录如下站点获得解决：
疯狂Java联盟：http://www.crazyit.org

上架建议：移动开发>Android

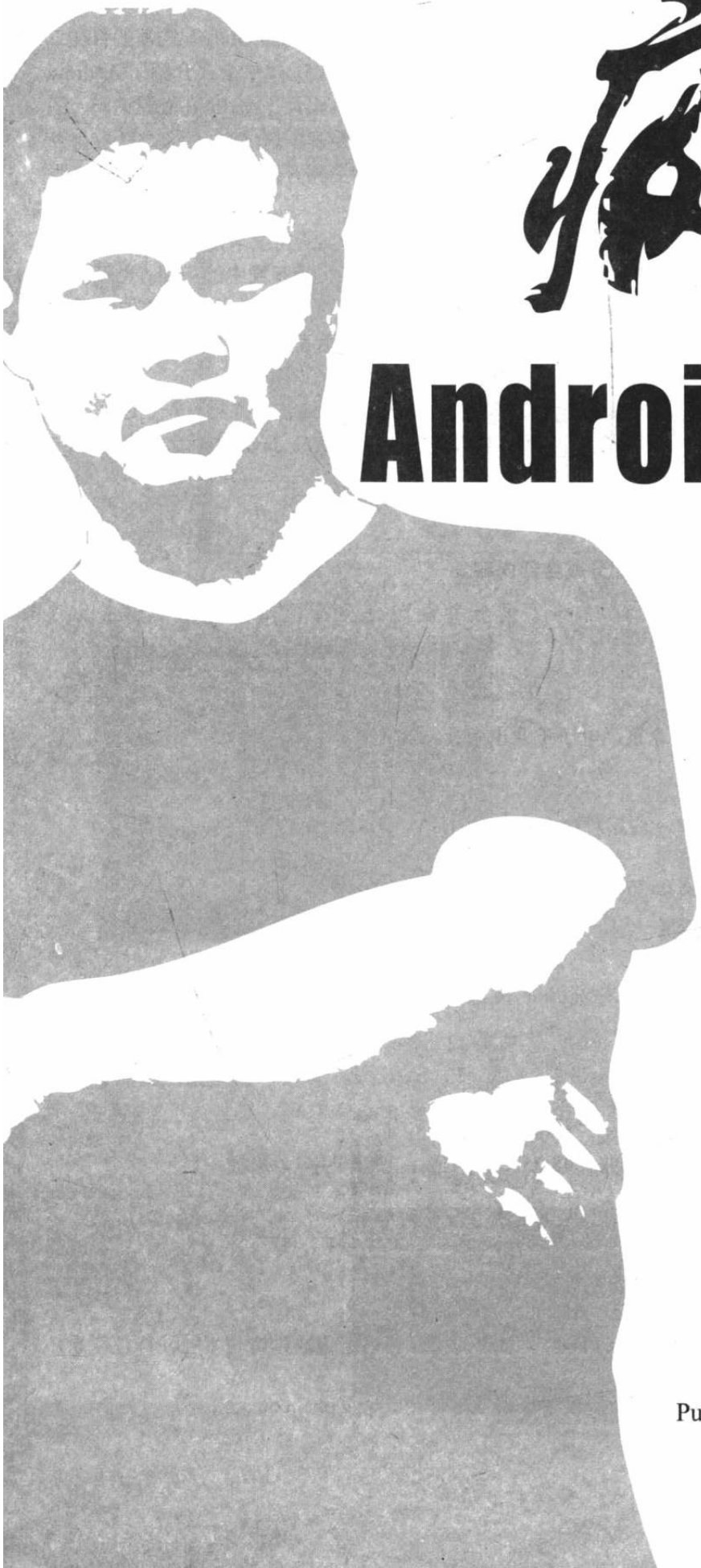
博文视点Broadview

新浪微博 weibo.com
@博文视点Broadview

策划编辑：张月萍
责任编辑：葛 娜
封面设计：李 玲

ISBN 978-7-121-25958-6

定价：108.00元（含光盘1张）



疯狂 Android讲义（第3版）

李 刚 编著

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

内 容 简 介

移动互联网已经成为当今世界发展最快、市场潜力最大、前景最诱人的业务，而 Android 则是移动互联网上市场占有率最高的平台。

本书是《疯狂 Android 讲义》的第 3 版，本书基于最新的 Android 5，并采用了 Google 推荐的 IDE：Android Studio 作为开发工具，书中每个案例、每个截图都全面升级到 Android 5。本书全面介绍了 Android 应用开发的相关知识，全书内容覆盖了 Android 用户界面编程、Android 四大组件、Android 资源访问、图形/图像处理、事件处理机制、Android 输入/输出处理、音频/视频多媒体应用开发、OpenGL 与 3D 应用开发、网络通信编程、Android 平台的 Web Service、传感器应用开发、GPS 应用开发、整合第三方 Map 服务等。

本书并不局限于介绍 Android 编程的各种理论知识，而是从“项目驱动”的角度来讲授理论，全书一共包括近百个实例，这些示范性的实例既可帮助读者更好地理解各知识点在实际开发中的应用，也可供读者在实际开发时作为参考、拿来就用。本书最后还提供了两个实用的案例：合金弹头和电子拍卖系统 Android 客户端，具有极高的参考价值。本书提供了配套的答疑网站，如果读者在阅读本书时遇到了技术问题，可以登录疯狂 Java 联盟（http://www.crazyit.org）发帖，笔者将会及时予以解答。

本书适合于有一定 Java 编程基础的读者。如果读者已熟练掌握 Java 编程语法，并具有一定的图形界面编程经验，阅读本书将十分合适；否则，在阅读本书之前建议先认真阅读疯狂 Java 体系之《疯狂 Java 讲义》。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

疯狂 Android 讲义 / 李刚编著. —3 版. —北京：电子工业出版社，2015.6
ISBN 978-7-121-25958-6

Ⅰ. ①疯… Ⅱ. ①李… Ⅲ. ①移动电话机—应用程序—程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2015）第 087649 号

策划编辑：张月萍
责任编辑：葛　娜
印　　刷：北京中新伟业印刷有限公司
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　　邮编：100036
开　　本：787×1092　1/16　　印张：48.75　　字数：1375千字　　彩插：1
版　　次：2011年7月第1版
　　2015年6月第3版
印　　次：2015年6月第1次印刷
印　　数：20000册　　定价：108.00元（含光盘1张）

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

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

服务热线：（010）88258888。

疯狂Java学习路线图（第二版）

说明：

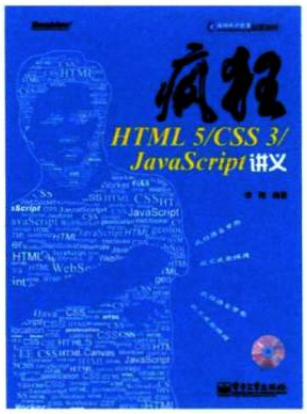
1. 没有背景色覆盖的区域稍有难度，请谨慎尝试。
2. 路线图上背景色与对应教材的封面颜色相同。
3. 已发现不少培训机构抄袭、修改该学习路线图，务请各培训机构保留对路线图的名称、引用说明。

疯狂Java体系

疯狂源自梦想 技术成就辉煌

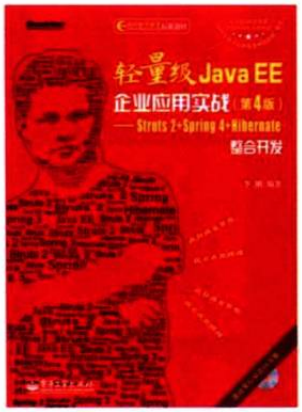
疯狂Java程序员的基本修养

作　　者：李刚
定　　价：59.00元
出版时间：2013-01
书　　号：978-7-121-19232-6



疯狂HTML 5/CSS 3/JavaScript 讲义

作　　者：李刚
定　　价：69.00元（含光盘1张）
出版时间：2012-05
书　　号：978-7-121-16863-5

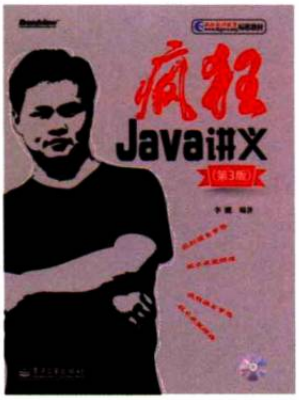


轻量级Java EE企业应用实战（第4版）——Struts 2＋Spring4＋Hibernate整合开发

作　　者：李刚
定　　价：108.00元（含光盘1张）
出版时间：2014-10
书　　号：978-7-121-24253-3

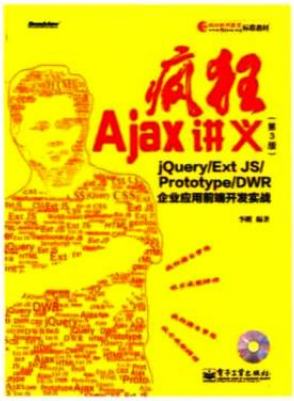
经典Java EE企业应用实战——基于WebLogic/JBoss的JSF+EJB 3+JPA整合开发

作　　者：李刚
定　　价：79.00元（含光盘1张）
出版时间：2010-08
书　　号：978-7-121-11534-9



疯狂Java讲义（第3版）

作　　者：李刚
定　　价：109.00元（含光盘1张）
出版时间：2014-07
书　　号：978-7-121-23669-3

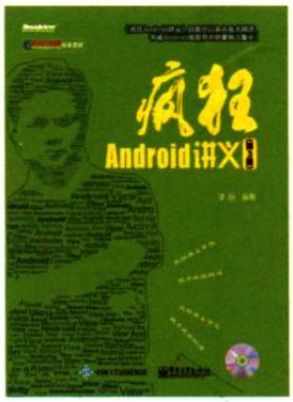


疯狂Ajax讲义（第3版）——jQuery/Ext JS/Prototype/DWR企业应用前端开发实战

作　　者：李刚
定　　价：79.00元(含光盘1张)
出版时间：2013-01
书　　号：978-7-121-19394-1

疯狂XML讲义（第2版）

作　　者：李刚
定　　价：69.00元（含光盘1张）
出版时间：2011-08
书　　号：978-7-121-14049-5



疯狂Android讲义（第3版）

作　　者：李刚
定　　价：108.00元（含光盘1张）
出版时间：2015-06
书　　号：978-7-121-25958-6

新浪微博：weibo.com/crazyjavabooks @疯狂Java体系图书

辉煌成绩

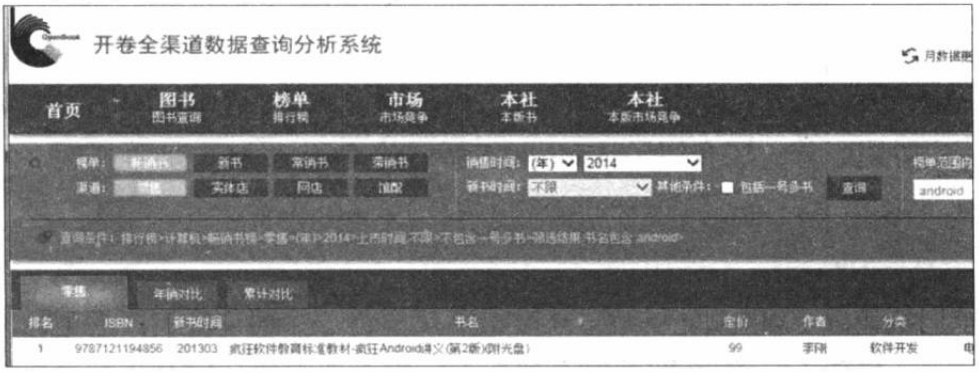
《疯狂 Android 讲义》自面市以来重印 22 次，发行量超 10 万册，并屡获殊荣！

❤曾获评 CSDN 年度最具技术影响力十大原创图书

❤多次荣获年度最畅销图书及长销图书大奖

❤在 Amazon 百万动销品种中曾取得排名第 3 的惊人成绩

❤开卷数据显示 2014 年度 Android 图书排行榜第一



前　言

移动互联网热潮在全世界引起了巨大反响，移动互联网正在改变着传统互联网的格局，全世界的IT公司争相将业务重心向移动互联网转型，移动互联网业务也成为业内最大的利润增长点。

Android系统就是一个开放式的移动互联网操作系统，Android已经成为应用最广的移动互联网平台，对于Java语言而言，Android系统给了Java一个新的机会。在过去的岁月中，Java语言作为服务器端编程语言，已经取得了极大的成功，Java EE平台发展得非常成熟，而且一直是电信、移动、银行、证券、电子商务应用的首选平台、不争的王者。但在客户端应用开发方面，Java语言一直表现不佳，虽然Java既有AWT/Swing界面开发库，也有SWT/JFace界面开发库，但对于客户端应用开发人员而言，大多不愿意选择Java语言。Android系统的出现改变了这种局面，Android是一个非常优秀的手机、平板电脑操作系统，正不断蚕食传统的桌面操作系统，而Android平台应用的开发语言就是Java，这意味着Java语言将可以在客户端应用开发上大展拳脚。

Android已经成为应用最广的手机、平板电脑操作系统，采用Java语言开发的Android应用也越来越多。不过需要指出的是，运行Android平台的硬件只是手机、平台电脑等便携式设备，这些设备的计算能力、数据存储能力都是有限的，因此不太可能在Android平台上部署大型企业级应用，因此Android应用可能以纯粹客户端应用的角色出现，然后通过网络与传统大型应用交互，充当大型企业应用的客户端，比如现在已经出现的淘宝Android客户端、赶集网Android客户端，它们都是这种发展趋势下的产物。

对于Java开发者来说，以前主要在Java EE平台上从事服务器端应用开发，但在移动互联网的趋势下，Java开发者必然面临着为这些应用开发客户端的需求。对于Java开发者来说，Android应用开发既是一个挑战，也是一个机遇——挑战是：掌握Android应用开发需要重新投入学习成本；机遇是：掌握Android开发之后将可让职业生涯达到一个新的高度，而且移动互联网与Android必然带来更多的就业机会与创业机会，这都值得当下的开发者好好把握。

本书是《疯狂Android讲义》的第3版，本书真正基于最新的Android 5，而且本书采用了Google推荐的IDE：Android Studio作为开发工具。书中每个案例、每个截图都是基于Android 5的。除此之外，本书详细介绍了Android 5新增的Material设计包含的elevation、translationZ和实时阴影等；还介绍了Android 5新增的Camera v2、屏幕捕捉、新型传感器等相关内容。

衷心感谢

疯狂Java体系图书能走到今天，广大读者的认同与支持是笔者坚持创作的最大动力。广大

读者的认同，已让疯狂Java体系图书的销量稳占国内第一。而《疯狂Android讲义》在所有Android图书销量中稳居榜首，在京东网搜索“android”关键字相关的图书，可以看到《疯狂 Android讲义》实际排在第1位（第1位并非介绍Android开发的图书），截图如下：

在当当网搜索“android”关键字相关的图书，可以看到《疯狂Android讲义》排在第1位，截图如下：

在亚马逊网搜索“android”关键字相关的图书，可以看到《疯狂 Android 讲义》排在第1位，截图如下：

诚挚地感谢广大读者的支持与爱护：你们的支持让疯狂 Java 图书没有放弃，你们的激励让疯狂 Java 图书茁壮成长，你们的反馈让疯狂 Java 图书日臻完善；同时也感谢博文视点张月萍等编辑、疯狂软件教育中心技术团队一贯的支持。

本书有什么特点

本书是一本介绍 Android 应用开发的实用图书，全面介绍了 Android 5 平台上应用开发各方面的知识。与市面上有些介绍 Android 编程的图书不同，本书并没有花太多篇幅介绍 Android 的发展历史（因为这些内容到处都是），完全没有介绍 Android 市场（因为它只是一个交易网站，与 Android 开发无关，但有些图书甚至用整整一章来介绍它），也没有介绍 JDK 安装、环境变量配置等内容——笔者假设读者已经具有一定的 Java 功底。换句话来说，如果你对 JDK 安装、Java 基本语法还不熟，本书并不适合你，建议先阅读《疯狂 Java 讲义》。

本书只用了一章来介绍如何搭建 Android 开发环境、Android 应用结构，当然也简要说明了 Android 的发展历史。可能依然会有人觉得本书篇幅很多，这是由于本书覆盖了 Android 开发绝大部分知识，而且很多知识不仅介绍了相应的理论，并通过相应的实例程序给出了示范。

需要说明的是，本书只是一本介绍 Android 实际开发的图书，而不是一本关于所谓“思想”的书，不要指望学习本书能提高你所谓的“Android 思想”，所以奉劝那些希望提高编程思想的读者不要阅读本书。

本书更不是一本看完之后可以“吹嘘、炫耀”的书——因为本书并没有堆砌一堆“深奥”的新名词、一堆“高深”的思想，本书保持了“疯狂 Java 体系”的一贯风格：操作步骤详细，编程思路清晰，语言平实。只要读者有基本的 Java 基础，阅读本书不会有任何问题，看完本书不会让你觉得自己突然“高深”了，“高深”到自己都理解不了。

认真看完本书，把书中所有示例都练习一遍，本书带给你的只是 9 个字：“看得懂、学得会、

做得出”。本书不能让你认识一堆新名词，只会让你学会实际的 Android 应用开发。

如果读者有非常扎实的 Java 基本功、良好的英文阅读能力，而且对图形用户界面编程也有丰富的经验，不管是 AWT/Swing 编程的经验，还是 SWT 编程的经验，抑或是 Windows 界面编程的经验都行，那么没有多大必要购买本书，只要花几天时间快速浏览本书即可动手编程了。如果遇到某个类、某个功能不太确定，直接查阅 Android Dev Guide 和 API 参考文档即可。

不管怎样，只要读者在阅读本书时遇到知识上的问题，都可以登录疯狂 Java 联盟（http://www.crazyit.org）与广大 Java 学习者交流，笔者也会通过该平台与大家一起交流、学习。

本书还具有如下几个特点。

1．知识全面，覆盖面广

本书深入阐述了 Android 应用开发的 Activity、Service、BroadcastReceiver 与 ContentProvider 四大组件，并详细介绍了 Android 全部图形界面组件的功能和用法，Android 各种资源的管理与用法，Android 图形/图像处理，事件处理，Android 输入/输出处理，音频/视频等多媒体开发，OpenGL-ES 开发，网络通信，传感器和 GPS 开发等内容，全面覆盖了 Android 官方指南，在某些内容上更加具体、深入。

2．内容实际，实用性强

本书并不局限于枯燥的理论介绍，而是采用了“项目驱动”的方式来讲授知识点，全书有近百个实例，几乎每个知识点都可找到对应的参考实例。本书最后还提供了“合金弹头”“电子拍卖系统 Android 客户端”两个应用，具有极高的参考价值。

3．讲解详细，上手容易

本书保持了“疯狂 Java 体系”的一贯风格：操作步骤详细，编程思路清晰，语言平实。只要读者有一定的 Java 编程基础，阅读本书将可以很轻松地上手 Android 应用开发；学习完本书最后的两个案例后，读者即可完全满足企业中实际 Android 应用开发的要求。

本书写给谁看

如果你已经具备一定的 Java 基础和 XML 基础，或已经学完了《疯狂 Java 讲义》一书，那么你阅读此书将会比较适合；如果你有不错的 Java 基础，而且有一定的图形界面编程经验，那么阅读本书将可以很快掌握 Android 应用开发。如果你对 Java 的掌握还不熟练，比如对 JDK 安装、Java 基本语法都不熟练，建议遵从学习规律，循序渐进，暂时不要购买、阅读此书。

2015 年 4 月

目 录 CONTENTS

第 1 章 Android 应用和开发环境 1
1.1 Android 的发展和历史 2
1.1.1 Android 的发展和简介 2
1.1.2 Android 5.x 平台架构及特性 3
1.2 搭建 Android 开发环境 5
1.2.1 安装 Android Studio 5
1.2.2 下载和安装 Android SDK 8
1.2.3 安装运行、调试环境 11
1.3 Android 常用开发工具的用法 16
1.3.1 在命令行创建、删除和浏览 AVD 16
1.3.2 使用 Android 模拟器（Emulator） 17
1.3.3 使用 Monitor 进行调试 18
1.3.4 Android Debug Bridge（ADB）的用法 19
1.3.5 使用 mksdcard 管理虚拟 SD 卡 20
1.4 开始第一个 Android 应用 21
1.4.1 使用 Android Studio 开发第一个 Android 应用 21
1.4.2 通过 Andorid Studio 运行 Android 应用 24
1.5 Android 应用结构分析 25
1.5.1 创建一个 Android 应用 25
1.5.2 自动生成的 R.java 28
1.5.3 res 目录说明 29
1.5.4 Android 应用的清单文件：AndroidManifest.xml 30
1.5.5 应用程序权限说明 31
1.6 Android 应用的基本组件介绍 32
1.6.1 Activity 和 View 32
1.6.2 Service 33
1.6.3 BroadcastReceiver 33
1.6.4 ContentProvider 33
1.6.5 Intent 和 IntentFilter 34
1.7 签名 Android 应用程序 35
1.7.1 使用 Android Studio 对 Android 应用签名 35
1.7.2 使用命令对 APK 包签名 36
1.8 本章小结 37

第 2 章 Android 应用的界面编程 38
2.1 界面编程与视图（View）组件 39
2.1.1 视图组件与容器组件 39
2.1.2 使用 XML 布局文件控制 UI 界面 44
2.1.3 在代码中控制 UI 界面 44
实例：用编程的方式开发 UI 界面 44
2.1.4 使用 XML 布局文件和 Java 代码混合控制 UI 界面 46
实例：简单图片浏览器 46
2.1.5 开发自定义 View 47
实例：跟随手指的小球 48
2.2 第 1 组 UI 组件：布局管理器 49
2.2.1 线性布局 50
2.2.2 表格布局 52
实例：丰富的表格布局 53
2.2.3 帧布局 55
实例：霓虹灯效果 57
2.2.4 相对布局 58
实例：梅花布局效果 59
2.2.5 网格布局 60
实例：计算器界面 61
2.2.6 绝对布局 62
实例：登录界面 63
2.3 第 2 组 UI 组件：TextView 及其子类 64
2.3.1 文本框（TextView）与编辑框（EditText）的功能和用法 64
实例：不同颜色、字体、带链接的文本 67
实例：圆角边框、渐变背景的 TextView 68
2.3.2 EditText 的功能与用法 70
实例：用户友好的输入界面 70
2.3.3 按钮（Button）组件的功能与用法 72
实例：按钮、圆形按钮、带文字的图片按钮 72
2.3.4 使用 9Patch 图片作为背景 73
2.3.5 单选钮（RadioButton）和复选框（CheckBox）的功能与用法 74
实例：利用单选钮、复选框获取用户信息 75

2.3.6 状态开关按钮（ToggleButton）和开关（Switch）的功能与用法 77
实例：动态控制布局 77
2.3.7 时钟（AnalogClock 和 TextClock）的功能与用法 79
实例：手机里的“劳力士” 79
2.3.8 计时器（Chronometer） 80
2.4 第 3 组 UI 组件：ImageView 及其子类 81
实例：图片浏览器 83
实例：强大的图片按钮 86
实例：使用 QuickContactBadge 关联联系人 87
2.5 第 4 组 UI 组件：AdapterView 及子类 88
2.5.1 列表视图（ListView）和 ListActivity 89
实例：改变分隔条、基于数组的 ListView 90
2.5.2 Adapter 接口及实现类 91
实例：使用 ArrayAdapter 创建 ListView 91
实例：基于 ListActivity 实现列表 93
实例：使用 SimpleAdapter 创建 ListView 94
实例：扩展 BaseAdapter 实现不存储列表项的 ListView 97
2.5.3 自动完成文本框（AutoCompleteTextView）的功能与用法 98
2.5.4 网格视图（GridView）的功能与用法 100
实例：带预览的图片浏览器 101
2.5.5 可展开的列表组件（ExpandableListView） 103
2.5.6 Spinner 的功能与用法 106
实例：让用户选择 106
2.5.7 AdapterViewFlipper 的功能与用法 108
实例：自动播放的图片库 108
2.5.8 StackView 的功能与用法 111
实例：叠在一起的图片 111
2.6 第 5 组 UI 组件：ProgressBar 及其子类 112
2.6.1 进度条（ProgressBar）的功能与用法 113
实例：显示在标题上的进度条 116
2.6.2 拖动条（SeekBar）的功能与用法 117
实例：通过拖动滑块来改变图片的透明度 117
2.6.3 星级评分条（RatingBar）的功能与用法 118
实例：通过星级改变图片的透明度 119
2.7 第 6 组 UI 组件：ViewAnimator 及其子类 120
2.7.1 ViewSwitcher 的功能与用法 120
实例：仿 Android 系统 Launcher 界面 120
2.7.2 图像切换器（ImageSwitcher）的功能与用法 125
实例：支持动画的图片浏览器 125
2.7.3 文本切换器（TextSwitcher）的功能与用法 127
2.7.4 ViewFlipper 的功能与用法 129
实例：自动播放的图片库 129
2.8 各种杂项组件 131
2.8.1 使用 Toast 显示提示信息框 131
实例：带图片的消息提示 131
2.8.2 日历视图（CalendarView）组件的功能和用法 133
实例：选择您的生日 133
2.8.3 日期、时间选择器（DatePicker 和 TimePicker）的功能和用法 134
实例：用户选择日期、时间 135
2.8.4 数值选择器（NumberPicker）的功能与用法 137
实例：选择您意向的价格范围 137
2.8.5 搜索框（SearchView）的功能与用法 139
实例：搜索 139
2.8.6 选项卡（TabHost）的功能和用法 141
实例：通话记录界面 141
2.8.7 滚动视图（ScrollView）的功能和用法 143
实例：可垂直和水平滚动的视图 144
2.8.8 Notification 的功能与用法 144
实例：加薪通知 145
2.9 第 7 组 UI 组件：对话框 146
2.9.1 使用 AlertDialog 创建对话框 147
实例：显示提示消息的对话框 147
实例：简单列表项对话框 149
实例：单选列表项对话框 149
实例：多选列表项对话框 150
实例：自定义列表项对话框 151
实例：自定义 View 对话框 152
2.9.2 对话框风格的窗口 154
2.9.3 使用 PopupWindow 155
2.9.4 使用 DatePickerDialog、TimePickerDialog 156
2.9.5 使用 ProgressDialog 创建进度对话框 158

2.10 菜单 160
2.10.1 选项菜单和子菜单（SubMenu） 161
2.10.2 使用监听器来监听菜单事件 164
2.10.3 创建多选菜单项和单选菜单项 164
2.10.4 设置与菜单项关联的 Activity 165
2.10.5 上下文菜单 166
2.10.6 使用 XML 文件定义菜单 167
实例：使用 XML 资源文件定义菜单 168
2.10.7 使用 PopupMenu 创建弹出式菜单 171
2.11 使用活动条（ActionBar） 173
2.11.1 启用 ActionBar 173
2.11.2 使用 ActionBar 显示选项菜单项 174
2.11.3 启用程序图标导航 176
2.11.4 添加 Action View 177
实例："标题"上的时钟 177
2.11.5 使用 ActionBar 实现 Tab 导航 178
实例：ActionBar 结合 Fragment 实现 Tab 导航 179
实例：Android 3.0 以前的 Fragment 支持 182
2.11.6 使用 ActionBar 实现下拉式导航 185
实例：ActionBar 结合 Fragment 实现下拉式导航 185
2.12 本章小结 187

第 3 章 Android 的事件处理 188
3.1 Android 事件处理概述 189
3.2 基于监听的事件处理 189
3.2.1 监听的处理模型 189
3.2.2 事件和事件监听器 192
实例：控制飞机移动 192
3.2.3 内部类作为事件监听器类 195
3.2.4 外部类作为事件监听器类 195
3.2.5 Activity 本身作为事件监听器类 197
3.2.6 匿名内部类作为事件监听器类 197
3.2.7 直接绑定到标签 198
3.3 基于回调的事件处理 199
3.3.1 回调机制与监听机制 199
3.3.2 基于回调的事件传播 201
3.3.3 重写 onTouchEvent 方法响应触摸屏事件 202
实例：通过回调实现跟随手指的小球 203
3.4 响应系统设置的事件 204
3.4.1 Configuration 类简介 204
实例：获取系统设备状态 205
3.4.2 重写 onConfigurationChanged 方法响应系统设置更改 206
实例：监听屏幕方向的改变 206
3.5 Handler 消息传递机制 208
3.5.1 Handler 类简介 208
实例：自动播放动画 209
3.5.2 Handler、Loop、MessageQueue 的工作原理 210
实例：使用新线程计算质数 211
3.6 异步任务（AsyncTask） 213
实例：使用异步任务执行下载 214
3.7 本章小结 217

第 4 章 深入理解 Activity 与 Fragment 218
4.1 建立、配置和使用 Activity 219
4.1.1 Activity 219
实例：用 LauncherActivity 开发启动 Activity 的列表 220
实例：使用 ExpandableListActivity 实现可展开的 Activity 221
实例：PreferenceActivity 结合 PreferenceFragment 实现参数设置界面 223
4.1.2 配置 Activity 227
4.1.3 启动、关闭 Activity 229
4.1.4 使用 Bundle 在 Activity 之间交换数据 231
实例：用第二个 Activity 处理注册信息 232
4.1.5 启动其他 Activity 并返回结果 235
实例：用第二个 Activity 让用户选择信息 235
4.2 Activity 的回调机制 239
4.3 Activity 的生命周期与加载模式 240
4.3.1 Activity 的生命周期演示 240
4.3.2 Activity 与 Servlet 的相似性和区别 243
4.3.3 Activity 的 4 种加载模式 244
4.4 Fragment 详解 249
4.4.1 Fragment 概述及其设计初衷 249
4.4.2 创建 Fragment 250
实例：开发显示图书详情的 Fragment 251
实例：创建 ListFragment 253
4.4.3 Fragment 与 Activity 通信 254
4.4.4 Fragment 管理与 Fragment 事务 256
实例：开发兼顾屏幕分辨率的应用 257
4.5 Fragment 的生命周期 260
4.6 本章小结 264

第 5 章　使用 Intent 和 IntentFilter 进行通信 265
5.1　Intent 对象简述 266
5.2　Intent 的属性及 intent-filter 配置 267
5.2.1　Component 属性 267
5.2.2　Action、Category 属性与 intent-filter 配置 269
5.2.3　指定 Action、Category 调用系统 Activity 273
实例：查看并获取联系人电话 274
实例：返回系统 Home 桌面 277
5.2.4　Data、Type 属性与 intent-filter 配置 ... 278
实例：使用 Action、Data 属性启动系统 Activity 284
5.2.5　Extra 属性 286
5.2.6　Flag 属性 286
5.3　使用 Intent 创建 Tab 页 287
5.4　本章小结 288
第 6 章　Android 应用的资源 289
6.1　应用资源概述 290
6.1.1　资源的类型以及存储方式 290
6.1.2　使用资源 291
6.2　字符串、颜色、尺寸资源 293
6.2.1　颜色值的定义 293
6.2.2　定义字符串、颜色、尺寸资源文件 294
6.2.3　使用字符串、颜色、尺寸资源 295
6.3　数组（Array）资源 298
6.4　使用 Drawable 资源 300
6.4.1　图片资源 300
6.4.2　StateListDrawable 资源 301
实例：高亮显示正在输入的文本框 301
6.4.3　LayerDrawable 资源 302
实例：定制拖动条的外观 303
6.4.4　ShapeDrawable 资源 304
实例：椭圆形、渐变背景的文本框 305
6.4.5　ClipDrawable 资源 306
实例：徐徐展开的风景 306
6.4.6　AnimationDrawable 资源 308
6.5　属性动画（Property Animation）资源 310
实例：不断渐变的背景色 311
6.6　使用原始 XML 资源 312
6.6.1　定义原始 XML 资源 312
6.6.2　使用原始 XML 文件 313
6.7　使用布局（Layout）资源 314
6.8　使用菜单（Menu）资源 315
6.9　样式（Style）和主题（Theme）资源 315
6.9.1　样式资源 315
6.9.2　主题资源 316
实例：给所有窗口添加边框、背景 317
6.9.3　Android 5.0 新增的 Material 主题 318
6.10　属性（Attribute）资源 318
6.11　使用原始资源 321
6.12　国际化和资源自适应 323
6.12.1　Java 国际化的思路 323
6.12.2　Java 支持的国家和语言 324
6.12.3　完成程序国际化 324
6.12.4　为 Android 应用提供国际化资源 326
6.12.5　国际化 Android 应用 327
6.13　自适应不同屏幕的资源 329
6.14　本章小结 332
第 7 章　图形与图像处理 333
7.1　使用简单图片 334
7.1.1　使用 Drawable 对象 334
7.1.2　Bitmap 和 BitmapFactory 334
7.2　绘图 337
7.2.1　Android 绘图基础：Canvas、Paint 等 337
7.2.2　Path 类 341
7.2.3　绘制游戏动画 344
实例：采用双缓冲实现画图板 344
实例：弹球游戏 348
7.3　图形特效处理 351
7.3.1　使用 Matrix 控制变换 351
实例：移动游戏背景 353
7.3.2　使用 drawBitmapMesh 扭曲图像 355
实例：可揉动的图片 356
7.3.3　使用 Shader 填充图形 358
7.4　逐帧（Frame）动画 360
7.4.1　AnimationDrawable 与逐帧动画 360
实例：在指定点爆炸 362
7.5　补间（Tween）动画 364
7.5.1　Tween 动画与 Interpolator 364
7.5.2　位置、大小、旋转度、透明度改变的补间动画 366
实例：蝴蝶飞舞 368
7.5.3　自定义补间动画 369
7.6　属性动画 373
7.6.1　属性动画的 API 373
7.6.2　使用属性动画 375
实例：大珠小珠落玉盘 379

7.7 使用 SurfaceView 实现动画 383
7.7.1 SurfaceView 的绘图机制 383
实例：基于 SurfaceView 开发示波器 386
7.8 本章小结 388

第 8 章 Android 数据存储与 IO 389
8.1 使用 SharedPreferences 390
8.1.1 SharedPreferences 与 Editor 简介 390
8.1.2 SharedPreferences 的存储位置和格式 391
实例：记录应用程序的使用次数 393
8.2 File 存储 393
8.2.1 openFileOutput 和 openFileInput 393
8.2.2 读写 SD 卡上的文件 396
实例：SD 卡文件浏览器 399
8.3 SQLite 数据库 402
8.3.1 SQLiteDatabase 简介 402
8.3.2 创建数据库和表 404
8.3.3 使用 SQL 语句操作 SQLite 数据库 404
8.3.4 使用 sqlite3 工具 406
8.3.5 使用特定方法操作 SQLite 数据库 408
8.3.6 事务 410
8.3.7 SQLiteOpenHelper 类 411
实例：英文生词本 412
8.4 手势（Gesture） 415
8.4.1 手势检测 415
实例：通过手势缩放图片 417
实例：通过手势实现翻页效果 419
8.4.2 增加手势 422
8.4.3 识别用户手势 425
8.5 自动朗读（TTS） 427
8.6 本章小结 429

第 9 章 使用 ContentProvider 实现数据共享 431
9.1 数据共享标准：ContentProvider 432
9.1.1 ContentProvider 简介 432
9.1.2 Uri 简介 433
9.1.3 使用 ContentResolver 操作数据 434
9.2 开发 ContentProvider 435
9.2.1 ContentProvider 与 ContentResolver 的关系 435
9.2.2 开发 ContentProvider 子类 436
9.2.3 配置 ContentProvider 437
9.2.4 使用 ContentResolver 调用方法 438
9.2.5 创建 ContentProvider 的说明 440
实例：使用 ContentProvider 共享生词本数据 441
9.3 操作系统的 ContentProvider 446
9.3.1 使用 ContentProvider 管理联系人 446
9.3.2 使用 ContentProvider 管理多媒体内容 452
9.4 监听 ContentProvider 的数据改变 455
9.4.1 ContentObserver 简介 455
实例：监听用户发出的短信 456
9.5 本章小结 457

第 10 章 Service 与 BroadcastReceiver 458
10.1 Service 简介 459
10.1.1 创建、配置 Service 459
10.1.2 启动和停止 Service 461
10.1.3 绑定本地 Service 并与之通信 462
10.1.4 Service 的生命周期 466
10.1.5 使用 IntentService 467
10.2 电话管理器（TelephonyManager） 470
实例：获取网络和 SIM 卡信息 470
实例：监听手机来电 472
10.3 短信管理器（SmsManager） 473
实例：发送短信 473
实例：短信群发 474
10.4 音频管理器（AudioManager） 477
10.4.1 AudioManager 简介 477
实例：使用 AudioManager 控制手机音频 477
10.5 振动器（Vibrator） 479
10.5.1 Vibrator 简介 479
10.5.2 使用 Vibrator 控制手机振动 479
10.6 手机闹钟服务（AlarmManager） 480
10.6.1 AlarmManager 简介 480
10.6.2 设置闹钟 481
实例：定时更换壁纸 483
10.7 接收广播消息 485
10.7.1 BroadcastReceiver 简介 485
10.7.2 发送广播 486
10.7.3 有序广播 488
实例：基于 Service 的音乐播放器 490
10.8 接收系统广播消息 495
实例：开机自动运行的 Service 495
实例：短信提醒 496
实例：手机电量提示 497
10.9 本章小结 498

第 11 章　多媒体应用开发 499
11.1　音频和视频的播放 500
11.1.1　使用 MediaPlayer 播放音频 500
11.1.2　音乐特效控制 503
实例：音乐的示波器、均衡、重低音和音场 505
11.1.3　使用 SoundPool 播放音效 511
11.1.4　使用 VideoView 播放视频 513
11.1.5　使用 MediaPlayer 和 SurfaceView 播放视频 515
11.2　使用 MediaRecorder 录制音频 518
实例：录制音乐 519
11.3　控制摄像头拍照 521
11.3.1　使用 Android 5.0 的 Camera v2 拍照 521
实例：拍照时自动对焦 522
11.3.2　录制视频短片 530
实例：录制生活短片 530
11.4　Android 5.0 新增的屏幕捕捉 533
11.5　本章小结 536

第 12 章　OpenGL 与 3D 开发 537
12.1　3D 图形与 3D 开发的基本知识 538
12.2　OpenGL 和 OpenGL ES 简介 539
12.3　绘制 2D 图形 540
12.3.1　在 Android 应用中使用 OpenGL ES 540
12.3.2　绘制平面上的多边形 542
12.3.3　旋转 548
12.4　绘制 3D 图形 550
12.4.1　构建 3D 图形 550
12.4.2　应用纹理贴图 554
12.5　本章小结 559

第 13 章　Android 网络应用 560
13.1　基于 TCP 协议的网络通信 561
13.1.1　TCP 协议基础 561
13.1.2　使用 ServerSocket 创建 TCP 服务器端 562
13.1.3　使用 Socket 进行通信 563
13.1.4　加入多线程 567
13.2　使用 URL 访问网络资源 573
13.2.1　使用 URL 读取网络资源 573
13.2.2　使用 URLConnection 提交请求 575
13.3　使用 HTTP 访问网络 580
13.3.1　使用 HttpURLConnection 580
实例：多线程下载 580
13.3.2　使用 Apache HttpClient 585
实例：访问被保护资源 586
13.4　Android 5.0 增强的 WebView 590
13.4.1　使用 WebView 浏览网页 590
实例：迷你浏览器 590
13.4.2　使用 WebView 加载 HTML 代码 591
13.4.3　使用 WebView 中的 JavaScript 调用 Android 方法 592
13.5　使用 Web Service 进行网络编程 595
13.5.1　Web Service 平台概述 595
13.5.2　使用 Android 应用调用 Web Service 597
实例：调用基于 CXF 的 Web Service 598
13.6　本章小结 601

第 14 章　管理 Android 手机桌面 602
14.1　管理手机桌面 603
14.1.1　删除桌面组件 603
14.1.2　添加桌面组件 603
14.2　改变手机壁纸 604
14.2.1　开发动态壁纸（Live Wallpapers） 605
实例：蜿蜒壁纸 605
14.3　通过程序添加快捷方式 609
实例：让程序占领桌面 609
14.4　管理桌面控件 611
14.4.1　开发桌面控件 611
实例：液晶时钟 614
14.4.2　显示带数据集的桌面控件 616
14.5　本章小结 620

第 15 章　传感器应用开发 621
15.1　利用 Android 的传感器 622
15.2　Android 的常用传感器 624
15.2.1　方向传感器 624
15.2.2　陀螺仪传感器 625
15.2.3　磁场传感器 625
15.2.4　重力传感器 625
15.2.5　线性加速度传感器 626
15.2.6　温度传感器 626
15.2.7　光传感器 626
15.2.8　压力传感器 626
15.2.9　Android 5.0 新增的心率传感器 629
15.3　传感器应用案例 630
实例：指南针 630
实例：水平仪 631

15.4 本章小结 636

第 16 章 GPS 应用开发 637

16.1 支持 GPS 的核心 API 638
16.2 获取 LocationProvider 640
16.2.1 获取所有可用的 LocationProvider 640
16.2.2 通过名称获得指定 LocationProvider 641
16.2.3 根据 Criteria 获得 LocationProvider 641
16.3 获取定位信息 642
16.3.1 通过模拟器发送 GPS 信息 642
16.3.2 获取定位数据 642
16.4 临近警告 644
16.5 本章小结 646

第 17 章 整合高德 Map 服务 647

17.1 调用高德 Map 服务 648
17.1.1 获取 Map API Key 648
17.1.2 高德地图入门 650
17.2 根据 GPS 信息在地图上定位 654
17.3 执行定位 660
17.3.1 地址解析与反向地址解析 660
17.3.2 执行定位 662
17.4 GPS 导航 665
17.5 本章小结 669

第 18 章 合金弹头 670

18.1 合金弹头游戏简介 671
18.2 开发游戏界面组件 671
18.2.1 游戏界面分析 671
18.2.2 实现“怪物”类 672
18.2.3 实现怪物管理类 679
18.2.4 实现“子弹”类 683
18.2.5 实现“角色”类 686
18.3 实现绘图工具类 691
18.4 加载、管理游戏图片 696
18.5 实现游戏界面 699
18.5.1 实现游戏 Activity 699
18.5.2 实现主视图 700
18.6 本章小结 710

第 19 章 电子拍卖系统 711

19.1 系统功能简介和架构设计 712
19.1.1 系统功能简介 712
19.1.2 系统架构设计 713
19.2 JSON 简介 714
19.2.1 使用 JSON 语法创建对象 715
19.2.2 使用 JSON 语法创建数组 716
19.2.3 Java 的 JSON 支持 717
19.3 发送请求的工具类 717
19.4 用户登录 719
19.4.1 处理登录的 Servlet 719
19.4.2 用户登录客户端 720
19.5 查看流拍物品 728
19.5.1 查看流拍物品的 Servlet 728
19.5.2 查看流拍物品客户端 729
19.6 管理物品种类 734
19.6.1 浏览物品种类的 Servlet 734
19.6.2 查看物品种类 735
19.6.3 添加种类的 Servlet 740
19.6.4 添加物品种类 740
19.7 管理拍卖物品 742
19.7.1 查看自己的拍卖物品的 Servlet 743
19.7.2 查看自己的拍卖物品 743
19.7.3 添加拍卖物品的 Servlet 747
19.7.4 添加拍卖物品 748
19.8 参与竞拍 753
19.8.1 选择物品种类 754
19.8.2 根据种类浏览物品的 Servlet 755
19.8.3 根据种类浏览物品 756
19.8.4 参与竞价的 Servlet 758
19.8.5 参与竞价 758
19.9 权限控制 763
19.10 本章小结 765

CHAPTER

1

第1章 Android应用和开发环境

本章要点

- Android 手机平台的发展与现状
- Android 手机平台的架构与特性
- 搭建 Android 应用的开发环境
- 管理 Android 虚拟设备
- 使用 Android 模拟器
- 安装和使用 Genymotion 模拟器
- 调试工具 Monitor 的用法
- 使用 ADB 工具复制文件、安装 APK 等
- 使用 Android Studio 开发 Android 应用
- 手动开发 Android 应用
- 掌握 Android 应用的结构
- 自动生成 Android 应用的清单文件
- Android 应用的 res 目录
- Android 应用的程序权限
- Android 应用的四大组件
- 对 Android 应用程序进行签名

Android 系统已经成为全球应用最广泛的手机操作系统，三星、华为、中兴等手机厂商早已通过 Android 阵营取得了巨大成功。目前国内对 Android 开发人才的需求也在迅速增长。而且搭载 Android 智能系统的手机越来越不像“手机”，更像是一台小型电脑。因此手机软件必将在未来 IT 行业中具有举足轻重的地位——你不可能带着一台电脑到处跑，而且时时开着机，但手机可以做到。从趋势上来看，对 Android 软件人才的需求会越来越大。

本书所介绍的平台是 Android 5 平台，该版本的 Android 平台经过几年的沉淀，不仅功能十分强大，而且十分高效、稳定。本书将会全面介绍 Android 平台的软件开发。但本章是全书的基础，将会简要介绍 Android 平台的历史、现状，重点向读者讲解如何搭建和使用 Android 应用开发环境，包括安装 Android SDK、Android 开发工具；也包括如何使用 Android 提供的 ADB、Monitor、AAPT 等工具，掌握这些工具是开发 Android 应用的基础技能。

1.1 Android 的发展和历史

Android 是由 Andy Rubin 创立的一个手机操作系统，后来被 Google 收购。Google 希望与各方共同建立一个标准化、开放式的移动电话软件平台，从而在移动产业内形成一个开放式的操作平台。

1.1.1 Android 的发展和简介

Android 并不是 Google 创造的，而是由 Android 公司所创造的，该公司的创始人是 Andy Rubin。该公司后来被 Google 收购，而 Andy Rubin 也成为 Google 公司的 Android 产品负责人。

Google 于 2007 年 11 月 5 日发布了 Android 1.0 手机操作系统，这个版本的 Android 系统还没有赢得广泛的市场支持。

2009 年 5 月，Google 发布了 Android 1.5，该版本的 Android 提供了一个非常“豪华”的用户界面，而且提供了蓝牙连接支持。这个版本的 Android 吸引了大量开发者的目光。接下来，Android 的版本更新得较快，目前最新的 Android 版本是 5.0，这也是本书所介绍的 Android 版本。

目前 Android 已经成为一个重要的手机操作系统。当前市场上常见的手机操作系统有如下这些。

- iOS：Apple 公司的手机、平板操作系统，市场占有率较高。
- Windows Phone：Microsoft 公司的手机操作系统，2014 年发布的最新版本为 Windows Phone 8.1，但局势依然不够明朗，前途依然堪忧。
- BlackBerry：即将被淘汰。
- Symbian：已被彻底淘汰，彻底退出历史舞台。

目前 Android 系统的市场占有率已经远超 iOS，而 Windows Phone 作为 Microsoft 公司最后的“赌注”，自然也是全力以赴，希望至少能与 iOS、Android 三足鼎立，但目前局势似乎并不乐观。无论从哪个角度来看，Android 已经成为最主流的手机操作系统。

提示：

事实上，Android 已经超出了手机操作系统的范畴，Android 系统已经广泛应用于 TV、手表以及各种可穿戴设备。最新发布的 Android 5.0 已经专门提供了 TV、Wear 等系统镜像。

就目前国内环境来说，已有大量手机厂商开始生产 Android 操作系统的手机，因为 Android 手机平台是一个真正开放式的平台，无须支付任何费用即可使用。出于节省研发费用的考虑，不管是对于知名的手机生产厂商，还是大量的山寨手机厂商，Android 操作平台都是一个不错的选择。

从 2008 年 9 月 22 日，T-Mobile 在纽约正式发布第一款 Android 手机——T-Mobile G1 开始，Android 系统不断地获得各个手机厂商的青睐。

2010 年 1 月 7 日，Google 在其美国总部正式向外界发布了旗下首款合作品牌手机 Nexus One（HTC G5），同时开始对外发售。

目前，已发布搭载 Android 系统的手机厂商包括：三星、HTC、索尼爱立信、LG 等；国内厂商如华为、联想、中兴、小米等也都开始发布搭载 Android 系统的手机。

1.1.2 Android 5.x 平台架构及特性

Android 系统的底层建立在 Linux 系统之上，该平台由操作系统、中间件、用户界面和应用软件 4 层组成，它采用一种被称为软件叠层（Software Stack）的方式进行构建。这种软件叠层结构使得层与层之间相互分离，明确各层的分工。这种分工保证了层与层之间的低耦合，当下层的层内或层下发生改变时，上层应用程序无须任何改变。

图 1.1 显示了 Android 系统的体系结构。

图 1.1 Android 系统的体系结构

备注：

图 1.1 以 Android 官方文档提供的附图为基础进行了修改。

从图 1.1 可以看出，Android 系统主要由 5 部分组成，下面分别对这 5 部分进行简单介绍。

1. 应用程序层

Android 系统将会包含一系列的核心应用程序，包括电子邮件客户端、SMS 程序、日历、地图、浏览器、联系人等。这些应用程序通常都是用 Java 编写的。这也是本书所介绍的主要内容：编写 Android 系统上的应用程序。

2. 应用程序框架

前面已经提到，本书所要介绍的内容就是开发 Android 应用程序，当我们开发 Android 应用程序时，就是面向底层的应用程序框架进行的。从这个意义上来看，Android 系统上的应用程序是完全平等的，不管是 Android 系统提供的程序，还是普通开发者提供的程序，都可以访问 Android 提供的 API 框架。

Android 应用程序框架提供了大量 API 供开发者使用，关于这些 API 的具体功能和用法则是本书后面要详细介绍的内容，此处不再展开阐述。

应用程序框架除了可作为应用程序开发的基础之外，也是软件复用的重要手段，任何一个应用程序都可发布它的功能模块——只要发布时遵守了框架的约定，那么其他应用程序就可使用这个功能模块。

3. 函数库

Android 包含一套被不同组件所使用的 C/C++库的集合。一般来说，Android 应用开发者不能直接调用这套 C/C++库集，但可以通过它上面的应用程序框架来调用这些库。

下面列出一些核心库。

- **系统 C 库**：一个从 BSD 系统派生出来的标准 C 系统库（libc），并且专门为嵌入式 Linux 设备调整过。
- **媒体库**：基于 PacketVideo 的 OpenCORE，这套媒体库支持播放和录制许多流行的音频和视频格式，以及查看静态图片。主要包括 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG 等多媒体格式。
- **Surface Manager**：管理对显示子系统的访问，并可以对多个应用程序的 2D 和 3D 图层提供无缝整合。
- **LibWebCore**：一个全新的 Web 浏览器引擎，该引擎为 Android 浏览器提供支持，也为 WebView 提供支持，WebView 完全可以嵌入开发者自己的应用程序中。本书后面会有关于 WebView 的介绍。
- **SGL**：底层的 2D 图形引擎。
- **3D libraries**：基于 OpenGL ES API 实现的 3D 系统，这套 3D 库既可使用硬件 3D 加速（如果硬件系统支持），也可使用高度优化的软件 3D 加速。
- **FreeType**：位图和向量字体显示。
- **SQLite**：供所有应用程序使用的功能强大的轻量级关系数据库。

4. Android 运行时

Android 运行时由两部分组成：Android 核心库集和 ART。其中核心库集提供了 Java 语言核心库所能使用的绝大部分功能，而虚拟机则负责运行 Android 应用程序。

Android 5.0 以前的 Android 运行时由 Dalvik 虚拟机和 Android 核心库集组成，但由于 Dalvik 虚拟机采用了一种被称为 JIT（Just-in-time）的解释器进行动态编译并执行，因此导致 Android App 运行时比较慢；而 ART 模式则是在用户安装 App 时进行预编译（Ahead-of-time，简称 AOT）的，将原本在程序运行时的编译动作提前到应用安装时，这样使得程序在运行时可以减少动态编译的开销，从而提升 Android App 的运行效率。

反过来，由于 ART 需要在安装 App 时进行 AOT 处理，因此 ART 需要占用更多的存储空间，应用安装和系统启动时间会延长不少。

除此之外，ART 还支持 ARM、x86 和 MIPS 架构，并且能完全兼容 64 位系统，Android 5.0

必然会带来更好的用户体验。

5. Linux 内核

Android 系统建立在 Linux 2.6 之上。Linux 内核提供了安全性、内存管理、进程管理、网络协议栈和驱动模型等核心系统服务。除此之外，Linux 内核也是系统硬件和软件叠层之间的抽象层。

1.2 搭建 Android 开发环境

在开始搭建 Android 开发环境之前，笔者假定读者已经具有一定的 Java 编程基础，像 JDK 安装、环境设置、设置 JAVA_HOME 环境变量之类的入门知识不在本书介绍范围之内。如果读者暂时还不会这些，建议先学习 Java 入门知识。

下面将从 Android SDK 的安装开始讲起，详细说明 Android 开发、调试环境的安装和使用，这些内容是 Android 开发的基础。

1.2.1 安装 Android Studio

Android Studio 是 Google 为 Android 提供的官方 IDE 工具，Google 建议广大 Android 开发者尽快从 Eclipse+ADT 的开发环境改为使用 Android Studio。

提示：

如果读者仍然希望学习使用 Eclipse+ADT 作为开发工具的开发方式，则可以选择阅读光盘根目录下的 Eclipse+ADT.pdf 电子文档，那是本书第 2 版的关于 Eclipse+ADT 安装和开发的内容。总之，无论使用何种 IDE 工具，技术才是根本，只有真正掌握了技术本身，任何工具才能用得得心应手。

Android Studio 不再基于 Eclipse，而是基于 IntelliJ IDEA 的 Android 开发环境。实际上，IntelliJ IDEA 一直都是一款非常优秀的 Java IDE 工具，只是因为 IntelliJ IDEA 是一款商业的 IDE 工具（虽然也有免费的社区交流版，但功能相当有限），因此影响了 IntelliJ IDEA 的广泛应用。现在，Google 以 IntelliJ IDEA 为基础推出的 Android Studio 同样可以免费使用，因此具有非常大的吸引力。

下载和安装 Android Studio 请按如下步骤进行。

1 登录 http://developer.android.com/sdk/index.html 页面，滚动到该页面的最下方，即可看到如图 1.2 所示的下载链接。

All Android Studio Packages

Select a specific Android Studio package for your platform. Also see the Android Studio release notes.

Platform	Package	Size	SHA-1 Checksum
Windows	android-studio-bundle-135.1641136.exe (Recommended)	868344232 bytes	1931dbaeadb52f5e0a8ba6e2ae60d9df20b2076b
	android-studio-ide-135.1641136.exe (No SDK tools included)	260272840 bytes	464d1c5497ab3d1bdef441365791ab36c89cd5ae
	android-studio-ide-135.1641136-windows.zip	246249059 bytes	6d6856aca83f6ff747ca40b10f70edfbbcccd91c
Mac OS X	android-stud... ide-1641136...	...5729073 ...tes	49506ba2cf6b56be4f7d07e6a00c4ec3ba2249d5
Linux	android-studio-ide-135.1641136-linux.zip	243917559 bytes	7c8f2d0cec21b98984cdba45ab5a25f26d67f23a

单击这个下载链接即可

图 1.2 下载 Android Studio

② 单击“android-studio-ide-135.1641136-windows.zip”链接，即可下载得到一个 android-studio-ide-135.1641136-windows.zip 压缩包。

③ 将 android-studio-ide-135.1641136-windows.zip 压缩包解压到任意盘符的根路径下，然后单击解压路径下 bin 目录下的 studio.exe 或 studio64.exe 文件，32 位系统运行 studio.exe，64 位系统运行 studio64.exe。运行该程序即可看到如图 1.3 所示的对话框。

图 1.3 所示的对话框询问用户是否需要导入 Android Studio 设置，如果以前用过 Android Studio 且保存了定制该 IDE 的设置信息，则可以选择第一个单选钮，并通过下面的文件浏览框选择 Android Studio 设置信息的存储位置；否则选择第二个单选钮。

④ 此处选择第二个单选钮，然后单击“OK”按钮，系统显示 Android Studio 的加载界面，加载完成将会看到如图 1.4 所示的配置向导对话框。

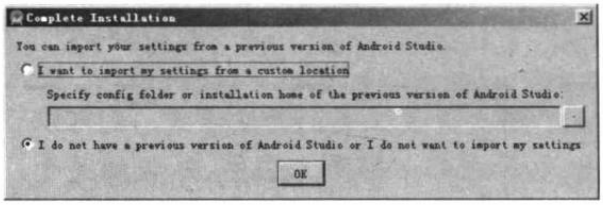


图 1.3　选择是否导入 Android Studio 设置

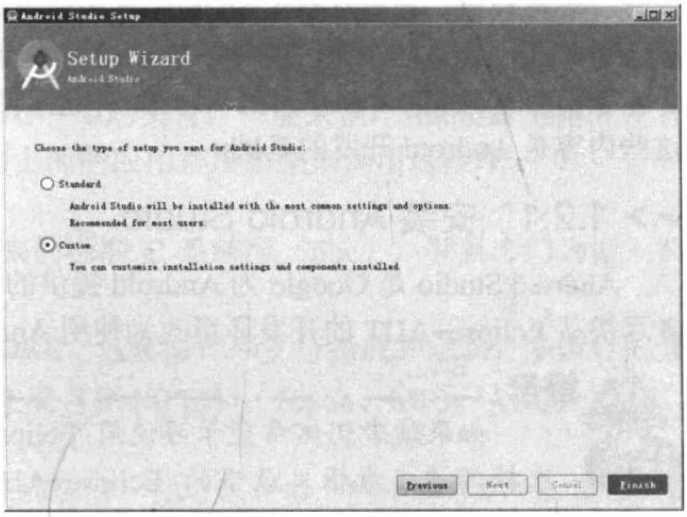


图 1.4　配置向导对话框

⑤ 图 1.4 所示的对话框用于引导用户从网络上下载 Android SDK，选择“Custom”单选钮，然后单击“Finish”按钮，即可看到如图 1.5 所示的对话框。

⑥ 单击“Finish”按钮，即可看到开始下载 Android SDK 的进度对话框。单击“Show Details”按钮，即可看到如图 1.6 所示的对话框。

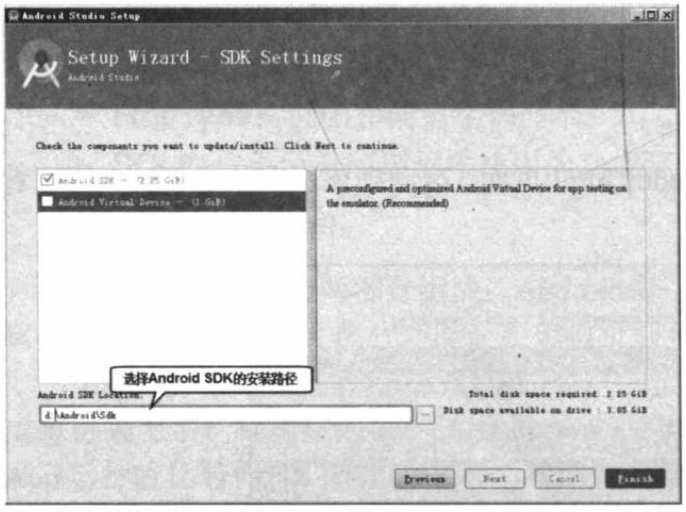


图 1.5　选择 Android SDK 的安装路径

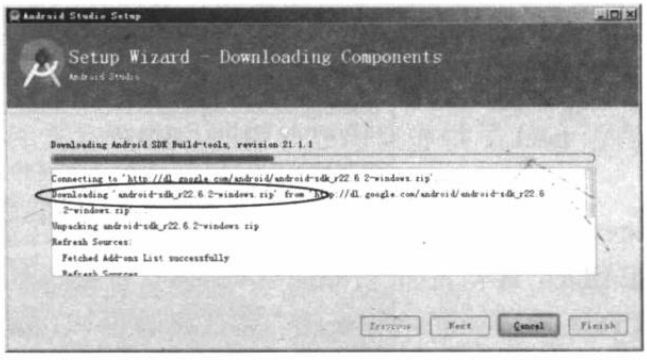


图 1.6　下载 Android SDK

⑦ 下载完成后单击“Finish”按钮，即可完成 Android Studio 的安装。安装完成后将看到如图 1.7 所示的对话框。

由于 Android Studio 是基于 IntelliJ IDEA 的 IDE 工具，因此 Android Studio 中 Project（项

目）的概念与 Eclipse 的 Project 概念不同，Android Studio 的项目相当于 Eclipse 的 Workspace（工作空间），Android Studio 的 Module（模块）才相当于 Eclipse 的项目——由此可见，Android Studio 的项目相当于一个工作空间，一个工作空间可包含多个模块，每个模块对应一个 Android 项目。因此要记住一句有些拗口的话：Andriod Studio 的项目可以包含多个 Android 项目（模块）。

⑧ 接下来即可单击图 1.7 所示对话框中的“Start a new Android Studio project”列表项来新建一个 Android Studio 项目。记住：Android Studio 的所谓项目只是一个工作空间，与 Android 项目并不对应。单击“Start a new Android Studio project”会显示如图 1.8 所示的对话框。

图 1.7 Android Studio 的开始对话框

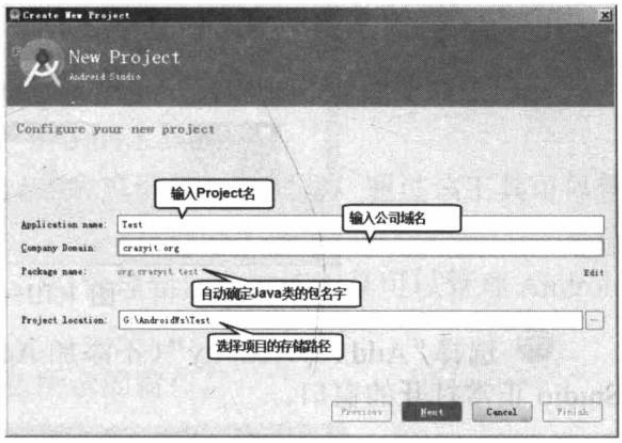


图 1.8 新建项目

⑨ 按图 1.8 所示方式输入新建项目所需的项目名、公司域名（Android Studio 将会根据项目名、公司域名自动确定项目的包名，当然开发者也可通过右边的“Edit”链接手动编辑项目的包名）和项目存储路径，然后单击“Next”按钮，接下来即可看到如图 1.9 所示的选择 Android SDK 版本的对话框。

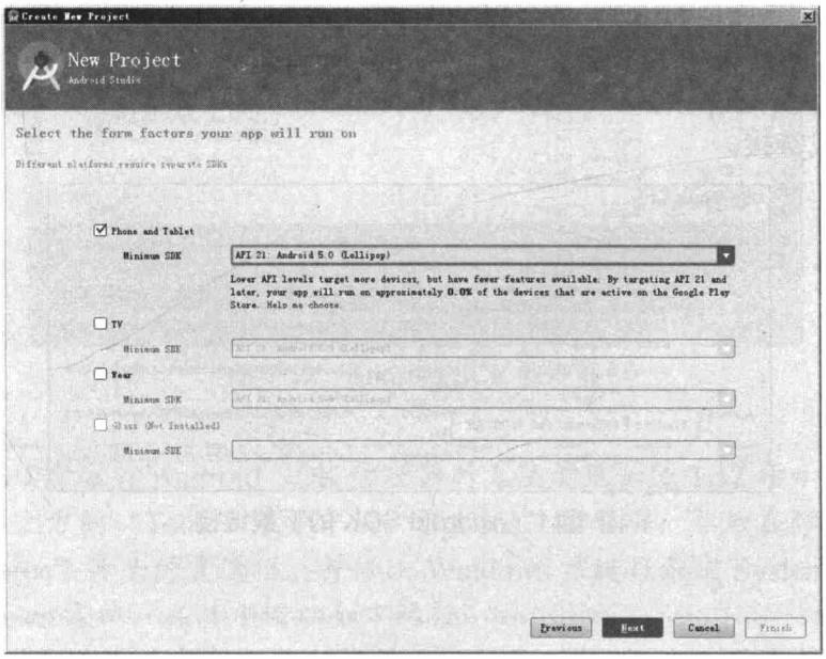


图 1.9 选择 Android SDK 版本

⑩ 根据业务需要选择 SDK 的最低版本要求——由于本书需要介绍 Android 5.0 的新功能，因此一般都会选择“API 21 Android 5.0”作为最低版本要求。选好最低的 SDK 版本要求之后，单击“Next”按钮，接下来即可看到如图 1.10 所示的添加 Activity 的对话框。

提示：

Activity是Android应用中最主要的应用组件，Activity在Android应用中是负责与用户交互的组件——大致上可以把它想象成传统界面编程中的窗口，读者此处无须过多掌握有关Activity的内容，本书后面会详细介绍它。

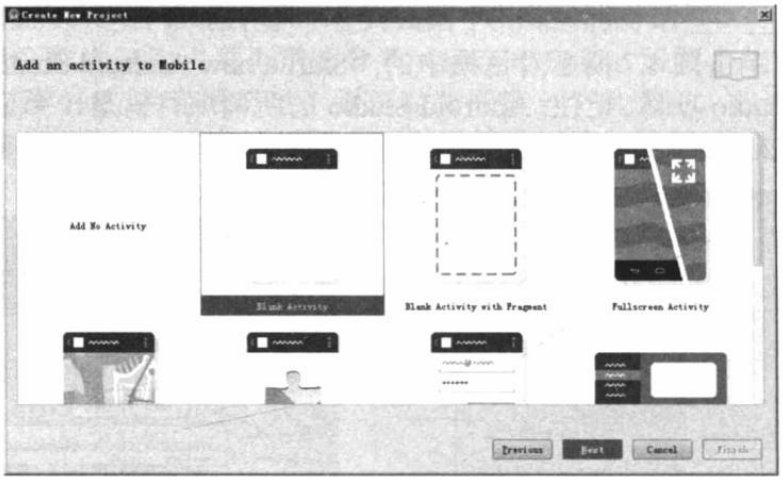


图1.10　选择添加Activity

⑪ 选择“Add No Activity”（不添加Activity），然后单击“Finish”按钮，即可看到Android Studio正常打开的窗口。

➤➤ 1.2.2　下载和安装Android SDK

虽然安装Android Studio时已经附带安装了Android SDK，但最新版的Android Studio自动下载的Android SDK往往不是最新版的，正如从图1.6所看到的，Android Studio下载的Android SDK版本为r22.6.2，但最新的Android SDK版本为r24.0.2，因此往往还需要重新下载最新的Android SDK。

Android的官方站点是http://www.android.com，登录该站点即可下载Android SDK。下载和安装Android SDK请按如下步骤进行。

① 登录http://developer.android.com/sdk/index.html页面，滚动到该页面下方，即可看到如图1.11所示的下载链接。

SDK Tools Only

If you prefer to use a different IDE or run the tools from the command line or with build scripts, you can instead download the stand-alone Android SDK Tools. These packages provide the basic SDK tools for app development, without an IDE.
Also see the SDK tools release notes.

Platform	Package	Size	SHA-1 Checksum
Windows	installer_r24.0.2-windows.exe (Recommended)	91428280 bytes	edac14e1541e97d68821fa3a709b4ea8c659e676
	android-sdk_r24.0.2-windows.zip	139473113 bytes	51269c8336f936fc9b9538f9b9ca236b78fb4e4b
Mac OS X	Windows平台的Andriod 5.0的SDK	262823 tes	3ab5e0ab0db5e7c45de9da7ff525dee6cfa97455
Linux	android-sdk_r24.0.2-linux.tgz	140097024 bytes	b6fd75e8b06b0028c2427e6da7d8a09d8f956a86

图1.11　Android SDK的下载链接

注意：

笔者使用电信ADSL无法正常访问上面的站点（该站点可能被电信封了，原因未知），如果读者也遇到这样的问题，建议设置代理服务器来访问该站点。

② 单击“android-sdk_r24.0.2-windows.zip”链接，通过该链接即可下载Android 5.0 SDK

压缩包。

提示：

不同平台下载相应的 SDK 压缩包，不管是 Windows 平台还是其他平台，都只需将下载得到的压缩包解压，再配置一些环境变量即可。

③ 下载完成后得到一个 android-sdk_r24.0.2-windows.zip 文件，将该文件解压缩到任意路径下，比如 D:\盘的根路径下。解压缩后得到一个 android-sdk-windows 文件夹，该文件夹下包含如下文件结构。

- add-ons：该目录下存放第三方公司为 Android 平台开发的附加功能系统。刚解压缩时该目录为空。
- platforms：该目录下存放不同版本的 Android 系统。刚解压缩时该目录为空。
- tools：该目录下存放了大量 Android 开发、调试的工具。
- AVD Manager.exe：该程序是 AVD（Android 虚拟设备）管理器。通过该工具可以管理 AVD。
- SDK Manager.exe：该程序就是 Android SDK 管理器。通过该工具可以管理 Android SDK。

④ 启动 SDK Manager.exe，即可看到如图 1.12 所示的窗口。

图 1.12 Android SDK 管理器

提示：

如果读者运行 Android SDK 管理器时无法看到如图 1.12 所示的列表，请单击该管理器上方的“Tools”菜单中的“Options”菜单项，然后在弹出的对话框中勾选以“Force”开头的复选框。并修改 Windows 系统目录中 System32\drivers\etc 目录下的 hosts 文件，在其中增加如下两行：

203.208.46.146 dl.google.com
203.208.46.146 dl-ssl.google.com

⑤ 在图 1.12 所示窗口的列表中勾选需要安装的平台和工具，比如 Android 5.0.1 的工具和

平台，其中 Android 文档、SDK Platform 是必选的。如果想查看 Android 官方提供的示例程序，使用 Android SDK 的源代码，则可以勾选“Samples for SDK”和“Sources for Android SDK”两个列表项（最好将 Android 5.0.1 包含的所有工具都安装上，如果无须为 Android TV、可穿戴设备开发应用，则可暂时不勾选以 Android TV、Android Wear 开头的选项）。至于是否需要安装 Android 早期版本的 SDK，则取决于读者喜好。选中所需安装的工具之后，单击“Install ××packages（其中××代表用户勾选的列表项的数量）”按钮，将看到出现如图 1.13 所示的窗口。

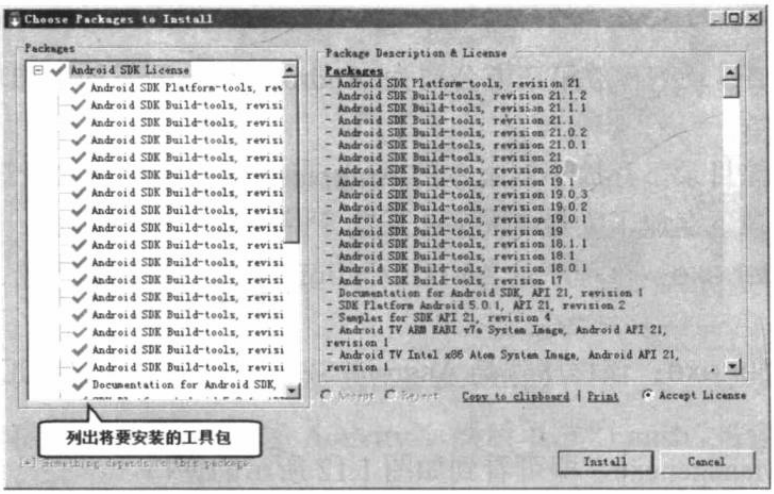


图 1.13　列出将要安装的 Android 工具包

⑥ 单击“Accept License”单选钮——确认需要安装所有的工具包，然后单击“Install”按钮，系统开始在线安装 Android SDK 及相关工具。取决于读者的网络状态及选中的工具包的数量，在线安装时间不会太短，甚至可能花费一两个小时，耐心等待即可。

⑦ 安装完成后将可以看到在 Android SDK 目录下增加了如下几个文件夹。

- docs：该文件夹下存放了 Android SDK 开发文件和 API 文档等。
- extras：该文件夹下存放了 Google 提供的 USB 驱动、Intel 提供的硬件加速等附加工具包。
- platform-tools：该文件夹下存放了 Android 平台相关工具。
- samples：该文件夹下存放了不同 Android 平台的示例程序。
- sources：该文件夹下存放了 Android 5.0 的源代码。
- system-images：该文件下存放了不同 Android 平台针对不同 CPU 架构提供的系统镜像。

⑧ 为了在命令行窗口可以使用 Android SDK 的各种工具，建议将 Android SDK 目录下的 tools 子目录、platform-tools 子目录添加到系统的 PATH 环境变量中。

成功安装了最新版本的 Android SDK 之后，还需要为 Android Studio 设置 Android SDK 的路径，单击“File”→“Other Settings”→“Default Project Structure”菜单，即可看到如图 1.14 所示的对话框。

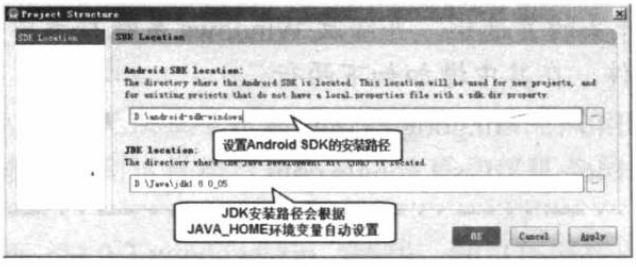


图 1.14　设置 Android SDK 的安装路径

在第一个文件浏览框中选择 Android SDK 的安装路径，然后单击“OK”按钮，为 Android Studio 设置 Android SDK 完成。如果读者还需要对 Android Studio 进行一些定制化设置，则可通过 Android Studio 的“File”→“Settings”菜单打开如图 1.15 所示的对话框。

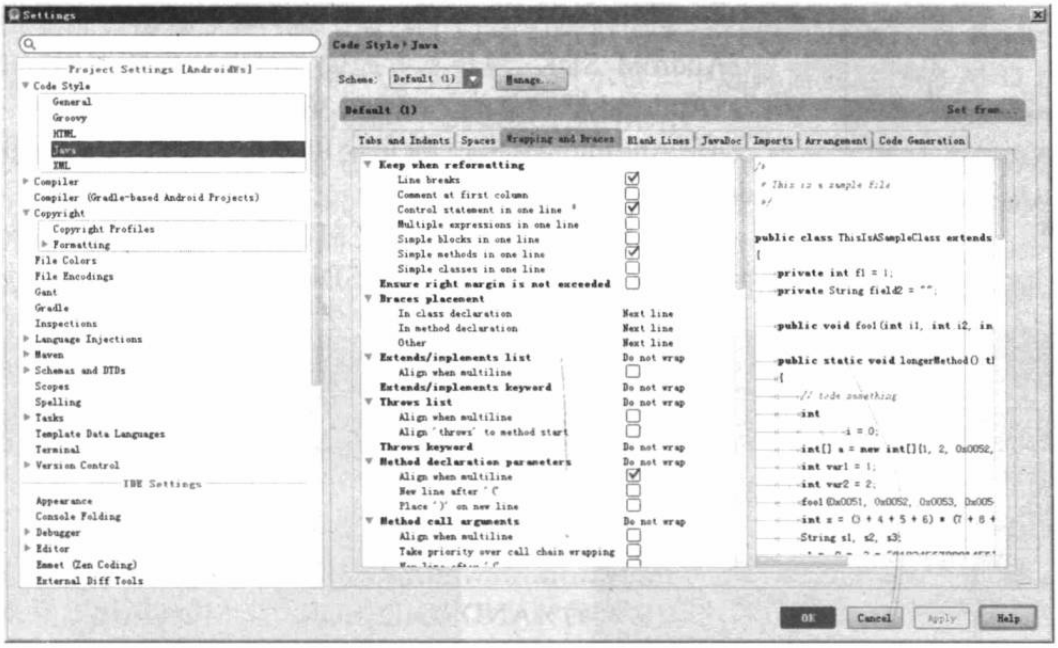


图 1.15　设置 Android Studio

➤➤ 1.2.3　安装运行、调试环境

Android 程序必须在 Android 手机上运行，因此 Android 开发时必须准备相关的运行、调试环境。准备 Android 程序的运行、调试环境有如下 3 种方式。

- 条件允许，优先考虑购买 Android 真机（真机调试的速度更快、效果更好）。
- 配置 Android 虚拟设备（即 AVD）。
- 使用第三方提供的 Genymotion 模拟器。

1. 使用真机作为运行、调试环境

使用真机作为运行、调试环境时，只要完成如下 3 步。

① 用 USB 连接线将 Android 手机连接到电脑上。

② 在电脑上为手机安装驱动，不同手机厂商的 Android 手机的驱动略有差异，请登录各手机厂商官网下载手机驱动。

注意：

通常都需要在电脑上为手机安装驱动。可能有读者会感到疑惑：Android 手机连接电脑后，电脑即可识别到 Android 的存储卡，不需要安装驱动啊？需要提醒读者的是，电脑仅能识别 Android 手机的存储卡是不够的，安装驱动才能把 Android 手机整合成运行、调试环境。

③ 打开手机的调试模式。打开手机，依次单击“Dev Tools”→“开发者选项”，进入如图 1.16 所示的设置界面，在该界面中“开启”开发者选项。

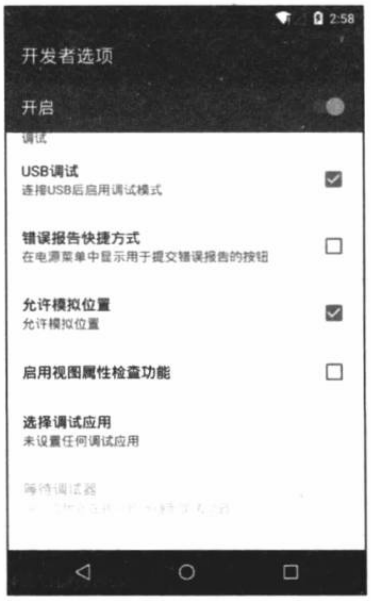


图 1.16　打开调试模式

按图 1.16 所示，勾选“Always stay awake”“USB 调试”“允许模拟位置”3 个选项即可。如果开发者还有其他需要，则可以勾选其他的开发者选项。

2. 使用 AVD 作为运行、调试环境

Android SDK 为开发者提供了可以在电脑上运行的“虚拟手机”，Android 把它称为 Android Virtual Device（AVD）。如果开发者没有 Android 手机，则完全可以在 AVD 上运行我们编写的 Android 应用。

创建、删除和浏览 AVD 之前，通常应该先为 Android SDK 设置一个环境变量：ANDROID_SDK_HOME，该环境变量的值为磁盘上一个已有的路径。如果不设置该环境变量，开发者创建的虚拟设备默认保存在 C:\Documents and Settings\<user_name>\.android 目录（以 Windows XP 为例）下；如果设置了 ANDROID_SDK_HOME 环境变量，那么虚拟设备就会保存在%ANDROID_SDK_HOME%\.android 路径下。

注意：

这里有一点非常容易混淆，此处的%ANDROID_SDK_HOME%环境变量并不是 Android SDK 的安装目录。而学习过 Java EE 的读者可能都记得 JAVA_HOME、ANT_HOME 等环境变量，它们都是指向自身的安装目录，但 Android 的%ANDROID_ SDK_HOME%不是。

在图形界面下管理 AVD 比较简单，因为可以借助于 Android SDK 和 AVD 管理器完成，完全可以在图形用户界面下操作，比较适合新上手的用户。

① 通过 Android SDK 安装目录下的 AVD Manager.exe 启动 AVD 管理器，如图 1.17 所示。单击该管理器左边的“Android Virtual Devices”选项卡，管理器列出当前已有的 AVD 设备，如图 1.17 所示。

② 单击右边的“Create...”按钮，AVD 管理器弹出如图 1.18 所示的对话框。

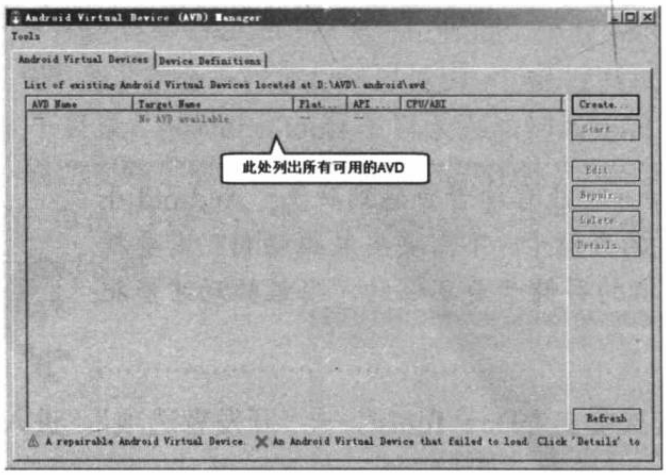


图 1.17　查看所有可用的 AVD 设备

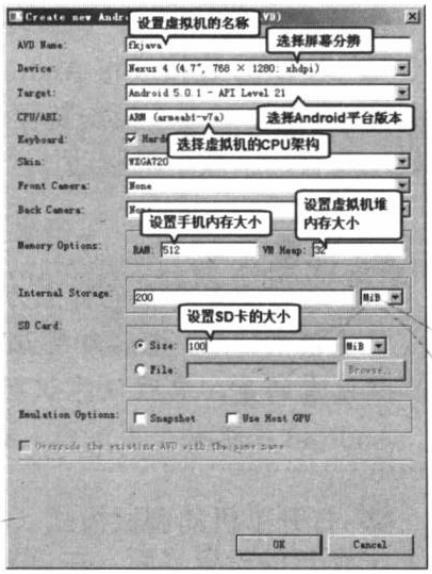


图 1.18　创建 AVD 设备

③ 填写 AVD 设备的名称、Android 平台的版本和虚拟 SD 卡的大小，然后单击“OK”

按钮，管理器即将开始创建 AVD 设备，开发者只要稍作等待即可。

创建完成后，管理器返回如图 1.17 所示的窗口，该管理器将会列出当前所有可用的 AVD 设备。如果开发者想删除某个 AVD 设备，只要在图 1.17 所示窗口中选择指定的 AVD 设备，然后单击右边的“Delete...”按钮即可。

AVD 设备创建成功之后，接下来就可以使用模拟器来运行该 AVD 了。在 Android SDK 和 AVD 管理器中运行 AVD 非常简单：①在图 1.17 所示窗口中选择需要运行的 AVD 设备；②单击“Start...”按钮即可。启动后的虚拟手机如图 1.19 所示。

图 1.19 所示的就是一个运行在电脑上的虚拟手机，用过手机的读者对这个界面应该不会陌生。现在开始使用该“虚拟手机”来模拟一些“手机操作”，读者可以花点时间来熟悉一下 Android 系统的操作习惯。

单击 Android 桌面上的“程序列表”按钮，Android 进入如图 1.20 所示的界面。

图 1.19 启动后的虚拟手机

图 1.20 Android 应用程序列表

从图 1.20 所示列表中可以看到 Android 系统默认提供的所有可用的程序，以后我们开发的 Android 程序也可以在这里找到。当包含的程序太多时，可以通过手指左右拖动来查看更多的程序。

对于国内用户来说，设置中文操作界面、设置中文输入法是两个常用的操作。设置中文操作界面可通过单击图 1.20 所示界面中的“Settings”项来进行，依次单击“Settings”→“Language & input”→“Language”，Android 系统将出现如图 1.21 所示的列表，选中其中的“中文（简体）”列表项，然后单击虚拟手机上的确认键返回。

图 1.21 设置中文操作界面

提示：

为 Android 模拟器设置了中文操作界面之后，在有些电脑上启动、运行模拟器特别慢，慢到令人难以忍受。如果遇到这种情况，请放弃使用中文操作界面。

开启中文输入法通过单击图 1.20 所示界面中的“Settings”项进行设置，依次单击“Settings”→“Language & input”，在出现的列表中勾选“谷

歌拼音输入法”列表项，然后单击虚拟手机上的确认键返回即可。

提示：

有时开发者启动了 Android 模拟器，“虚拟手机”的显示屏右上方可能提示没有网络信号，通常是因为模拟器无法访问网络的缘故。一般来说，只要运行模拟器的电脑已经处于局域网内（已接入 Internet 也可以），并且没有防火墙阻止 Android 模拟器访问网络，Android 模拟器都不应该提示没有网络信号。如果运行 Android 模拟器的机器既不在局域网内，也没有接入 Internet，则可将电脑 DNS 服务器设为与本机相同。例如，设置本机 IP 地址为 192.168.1.50，再将 DNS 服务器地址也设为 192.168.1.50 即可。

3. 安装 Genymotion 模拟器

使用 Android 自带的模拟器虽然简单、方便，但最大的问题就是慢，慢到让大部分开发者难以忍受，这时可以选择使用第三方模拟器：Genymotion，这个模拟器最大的特点是速度快，使用该模拟器可模拟出与真机媲美的速度。

提示：

如果读者的电脑性能很好，使用 Android 自带的模拟器的性能还可以承受，则完全可以跳过这部分内容。

下载和安装 Genymotion 模拟器按如下步骤进行。

① 登录 https://www.genymotion.com/#!/download 站点，看到如图 1.22 所示的页面。

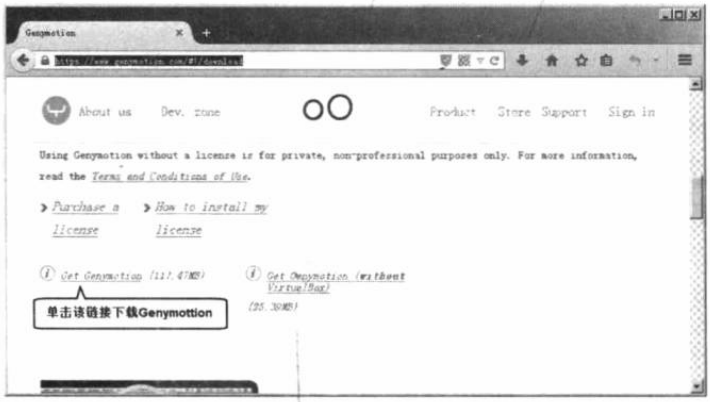


图 1.22　下载 Genymotion 模拟器

② 单击“Get Genymotion”链接，浏览器打开如图 1.23 所示的页面。

图 1.23　登录 Genymotion 官网

提示： Genymotion 有两个版本：免费的个人使用版和收费的商业版本。免费版的功能比较少，只能个人使用，但商业版本要收费，因此此处我们使用免费版。

在图 1.23 所示页面中输入用户名、密码登录 Genymotion 官网，如果读者暂时没有该网站的账户，则可单击该页面上的“Create Account”按钮注册一个。注册账户时需要输入一个有效的邮箱地址——注册完成后必须登录该邮箱来激活账户。

③ 登录成功之后就会开始下载，下载完成后得到一个 genymotion-2.3.1-vbox.exe 文件，该文件就是 Genymotion 的安装文件，双击该文件即可开始安装。安装 Genymotion 与安装其他 Windows 程序并没有什么区别，此处不再赘述。

提示： 安装 Genymotion 时会自动安装 Oracle 的 VM VirtualBox，读者只要不断地单击“Next”按钮即可完成安装。

安装完成后会自动重启电脑。

④ 从“开始”菜单中启动 Genymotion，即可看到如图 1.24 所示的窗口。

⑤ 从图 1.24 可以看出，此时还没有任何 Genymotion 模拟器，读者可通过单击该窗口上方的“Add”按钮来添加模拟器。单击“Add”按钮会再次输入用户名、密码登录系统，输入前面在 Genymotion 官网注册的用户名、密码，即可看到如图 1.25 所示的列表。

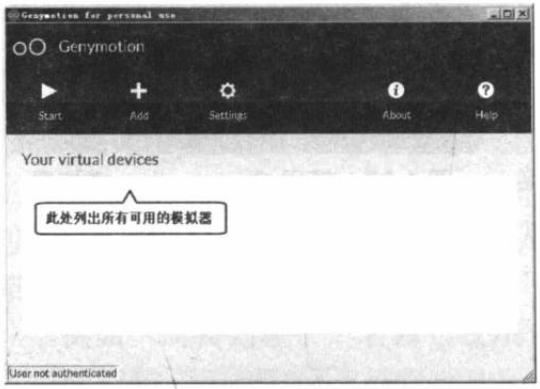


图 1.24 管理 Genymotion 模拟器

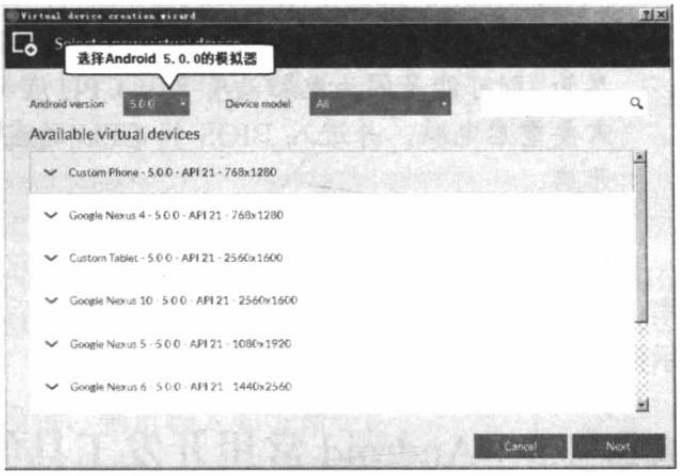


图 1.25 选择模拟器

⑥ 在图 1.25 所示列表中选择一个模拟器（此处的选择并不重要，后面还需要对此处的选择进行重新设置），此处选择第一项：Custom Phone - 5.0.0 - API 21 – 768x1280，然后单击“Next”按钮。系统将会显示一个文本框让用户输入该模拟器的名称，按自己喜欢随意输入一个名称，然后单击“Next”按钮。Genymotion 将会开始下载文件，下载完成后将会自动完成配置，配置完成后单击“Finish”按钮即可返回图 1.24 所示窗口，此时将可在该窗口中看到我们刚刚配置的 Android 模拟器。

⑦ 在图 1.24 所示窗口中右击刚刚创建的 Android 模拟器，在弹出的右键菜单中单击“Settings”菜单项，即可看到如图 1.26 所示的设置窗口。

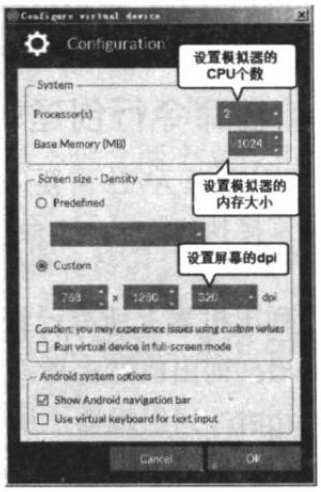


图 1.26 设置 Genymotion 模拟器的模拟参数

注意：

图 1.26 所示的内容设置了模拟器有两个 CPU，内置内存为 1024MB，读者可根据自己的电脑的性能进行适当的调整：如果读者的电脑的性能很好，则可以将模拟器的 CPU 个数、内存容量适当调大；如果读者的电脑的性能较差，则可以将模拟器的 CPU 个数、内存容量适当调小。

⑧ 在图 1.26 所示对话框中设置完成后，单击“OK”按钮即可返回图 1.24 所示窗口，在该窗口的模拟器列表中就可以看到刚刚配置的模拟器。在该窗口中选择刚刚配置的模拟器，然后单击窗口上方的“Start”按钮，即可启动 Genymotion 模拟器。

Genymotion 模拟器的启动速度比 Android 自带的模拟器的启动速度快，因此很快即可看到如图 1.27 所示的模拟器界面。

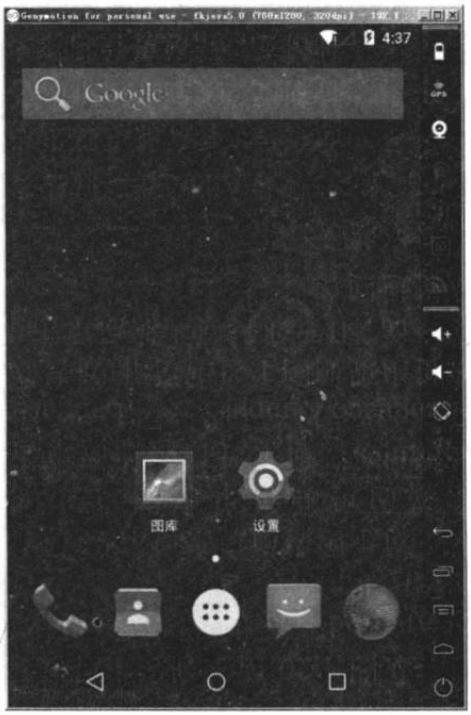


图 1.27　使用 Genymotion 模拟器

注意：

如果读者在启动 Android 5.0 版本的 Genymotion 模拟器时一直卡在黑屏界面，或者卡在 Android Logo 界面，则可能是因为电脑没有开启 CPU 虚拟化支持。需要重启电脑，并进入 BIOS 将 CPU 的虚拟化支持开启。

读者可以操作 Genymotion 模拟器，感受它的运行效率：这个模拟器的运行速度真的很流畅，本书后面大部分示例都是基于这个模拟器的。

1.3　Android 常用开发工具的用法

前面主要介绍了 Android SDK 的安装，运行、调试环境的搭建，以及 Android 开发环境 Android Studio 的安装，但这些内容只是最基本的知识，要真正掌握 Android 开发，还必须掌握 Android 开发的大量辅助工具。

1.3.1　在命令行创建、删除和浏览 AVD

在命令行下管理 AVD 需要借助于 android 命令（位于 Android SDK 安装目录的 tools 子目录下），如果直接执行 android 命令将会启动 Android SDK 管理器。除此之外，该命令还支持如下子命令。

- list：列出机器上所有已经安装的 Android 版本和 AVD 设备。
- list avd：列出机器上所有已经安装的 AVD 设备。
- list target：列出机器上所有已经安装的 Android 版本。
- create avd：创建一个 AVD 设备。
- move avd：移动或重命名一个 AVD 设备。

- delete avd：删除一个 AVD 设备。
- update avd：升级一个 AVD 设备使之符合新的 SDK 环境。
- create project：创建一个新的 Android 项目。
- update project：更新一个已有的 Android 项目。
- create test-project：创建一个新的 Android 测试项目。
- update test-project：更新一个已有的 Android 测试项目。

如果希望查看当前系统上已经安装的 Android 版本及已经安装的 AVD 设备，则运行 android list 或者 android list avd 命令即可，如图 1.28 所示。

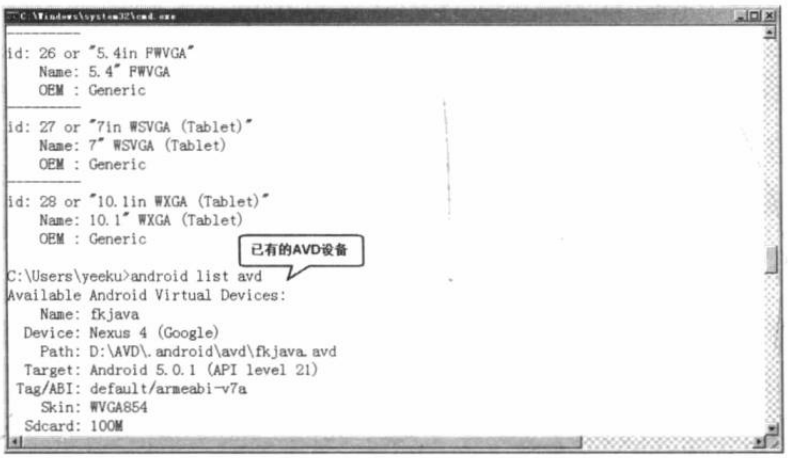


图 1.28 列出已经安装的 Android 版本和 AVD 设备

如果要创建一个全新的 AVD 设备，可执行如下命令：

```
android create avd -n <avd名称> -t <Android版本> -b <CPU架构>
-p <AVD设备保存位置> -s <选择AVD皮肤>
```

在上面的 create avd 子命令中，只有-n 和-t 选项是必需的，其余的-b 选项、-p 选项、-s 选项都是可选的。如果不设置-p 选项，创建的 AVD 设备默认保存在%ANDROID_SDK_HOME%/.android/avd 路径下。

例如，需要创建一个名为 crazyit 的 AVD 设备，则可输入如下命令：

```
android create avd -n crazyit -t 21 -b armeabi-v7a
```

上面的命令中 21 是 Android 5.0 的代号，如图 1.28 所示。执行上面的命令，系统会提醒用户是否需要定制 AVD 的硬件，开发者可以选择 yes 或 no，如果输入 no，即可直接开始创建 AVD 设备；如果输入 yes，即可开始定制 AVD 硬件的各种选项，定制完成后系统开始创建 AVD 设备。

为系统创建了两个 AVD 之后（前面通过图形界面创建了一个），即可在%ANDROID_SDK_HOME%/.android 目录下看到一个 avd 子目录，该子目录下包含两个文件和两个文件夹。

- fkjava.avd 和 fkjava.ini：fkjava AVD 的基本信息和 AVD 设备。其中 fkjava.avd 目录下有一个 userdata.img 文件，它是 AVD 中用户数据的镜像。还有一个 sdcard.img 文件，它是该 AVD 所使用的虚拟 SD 卡的镜像。
- crazyit.avd 和 crazyit.ini：crazyit AVD 的基本信息和 AVD 设备。其中 crazyit.avd 目录下有一个 userdata.img 文件，它是 AVD 中用户数据的镜像。

1.3.2 使用 Android 模拟器（Emulator）

Android 模拟器就是一个运行在电脑上的“虚拟手机”。实际上，前面我们已经使用过

Android 模拟器了，在 Android SDK 和 AVD 管理器中选择指定的 AVD 设备，然后单击“Start...”按钮就是启动模拟器来运行 Android 系统。

在 Android SDK 安装目录的 tools 子目录下有一个 emulator.exe（另外还有 emulator-arm.exe、emulator-mips.exe、mulator-x86.exe），它们都是 Android 模拟器。这个模拟器做得十分出色，几乎可以模拟真实手机的绝大部分功能（只是速度很慢），后面我们会陆续看到——当然，它只是模拟，不要指望用模拟器与你现实中的朋友“煲电话粥”。

使用 emulator.exe 启动模拟器有两种用法。

- emulator –avd <AVD 名称>
- emulator –data 镜像文件名称

第一种用法是运行指定的 AVD 设备，例如如下命令：

```
emulator -avd crazyit //运行名为 crazyit 的 AVD 设备
```

第二种用法是直接使用指定的镜像文件来运行 AVD，例如如下命令：

```
emulator -data myfile //以 myfile 作为镜像文件来运行 AVD 设备
```

1.3.3 使用 Monitor 进行调试

当 Android 应用在模拟器上运行时，我们看不到程序运行的过程，在命令行控制台也看不到程序的输出，那如何调试 Android 应用呢？

不用担心，Android 已经为我们考虑好了这个问题。Android 提供了一个 Monitor 工具，该工具可用于监视 Android 设备的运行，它是一个功能非常强大的调试环境。运行如下命令：

```
monitor.bat
```

即可看到系统启动了如图 1.29 所示的窗口。

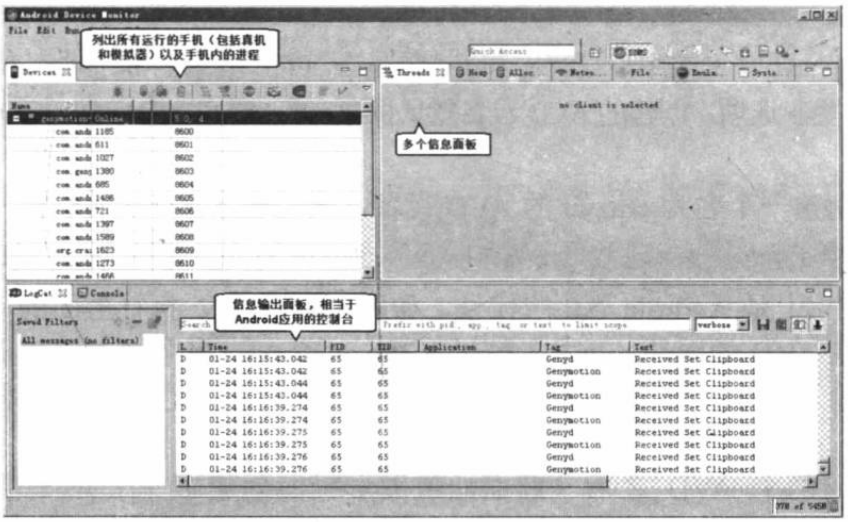


图 1.29　Monitor 的调试窗口

在图 1.29 所示的窗口中有如下几个重要的面板。

- **设备面板**：Monitor 窗口左上角的面板，该面板会列出当前所有运行的手机（包括真机和模拟器），并列出各手机内的所有进程信息。如果需要查看指定手机或指定进程信息，则应先在该面板内选中指定手机或进程。
- **信息输出面板**：该面板位于 Monitor 窗口的下方，相当于传统 Java 应用控制台，因此非常重要。
- **线程跟踪面板**：该面板可用于查看指定进程内所有正在执行的线程状态。如果需要让

该面板显示指定进程内线程的状态，则应保证下面两步：①在设备面板上选中需要查看的进程；②在设备面板上单击“Update Threads”按钮。

- **Heap 内存跟踪面板**：该面板可用于查看指定进程内堆内存的分配和回收信息。如果需要让该面板显示指定进程内 Heap 的回收和分配状态，则应保证下面两步：①在设备面板上选中需要查看的进程；②在设备面板上单击“Update Heap”按钮。
- **模拟器控制面板**：该面板用于让模拟器模拟拨打电话、发送短信等，还可以设置模拟器的虚拟位置信息等。图 1.30 显示了该面板的示意图。
- **文件管理器面板**：该面板可用于查看 Android 设备所包含的文件，也可用于将 Android 设备的文件导出到电脑上，也可将电脑中的文件导入 Android 设备，如图 1.31 所示。

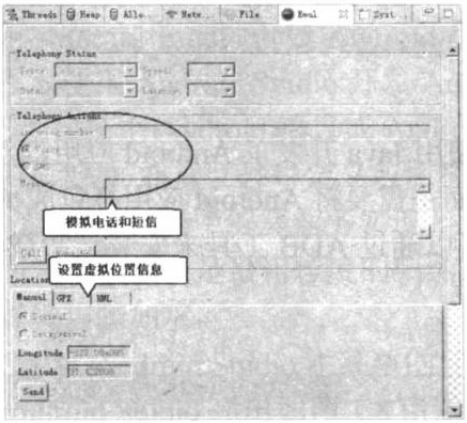


图 1.30 模拟器控制面板

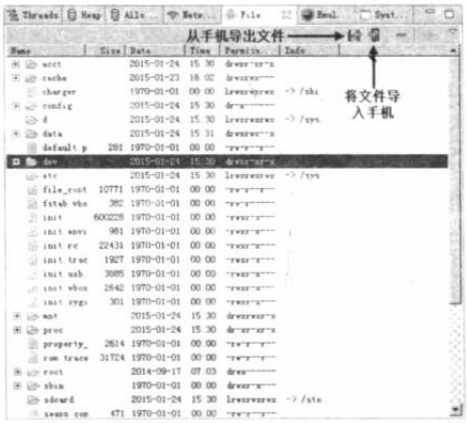


图 1.31 文件管理器面板

实际上，Android Studio 已经将 Monitor 集成进来，在 Android Studio 下方可以看到如图 1.32 所示的面板。

图 1.32 Android Studio 继承的 Monitor 面板

1.3.4 Android Debug Bridge（ADB）的用法

Android Debug Bridge（ADB）是一个功能非常强大的工具，它位于 Android SDK 安装目录的 platform-tools 子目录下。ADB 工具既可完成模拟器文件与电脑文件的相互复制，也可安装 APK 应用，甚至可以直接切换到 Android 系统中执行 Linux 命令。

ADB 工具的功能很多，此处就几个常用的功能略做说明。

1. 查看当前运行的模拟器

输入如下命令，即可查看当前运行的模拟器：

```
adb devices
```

2. 电脑与手机之间文件的相互复制

在默认情况下，ADB 工具总是操作当前正在运行的模拟器。

如果需要将电脑文件复制到模拟器中，则可使用 adb push 命令：

```
adb push d:\abc.txt /sdcard/
```

上面的命令将电脑的D:\盘根目录下的abc.txt文件复制到手机的/sdcard/目录下。

如果需要将模拟器文件复制到电脑中，则可使用adb pull命令：

```
adb pull /sdcard/xyz.txt d:\
```

上面的命令将模拟器的/sdcard/目录下的xyz.txt文件复制到电脑的D:\盘根目录下。

3. 启动模拟器的shell窗口

Android平台的内核是基于Linux的，有时开发者希望直接打开Android平台的shell窗口，这样就可以在该窗口内执行一些常用的Linux命令，如ls、mkdir、rm等。此时可考虑使用adb shell命令：

```
adb shell
```

4. 安装、卸载APK程序

APK程序就是Android程序的发布包。虽然我们使用Java开发了Android应用，但并不是直接将Java二进制文件复制到手机或模拟器上即可，而是需要将Android应用打包成APK包。

一旦将Android应用打包成APK包，接下来就可以通过ADB工具来安装、卸载APK程序了。

使用ADB安装APK程序的命令格式如下：

```
adb install [-r] [-s] <file>
```

上面的命令格式指定安装<file>代表的APK包。其中-r表示重新安装该APK包；-s表示将APK包安装到SD卡上——默认是将APK包安装到内部存储器上。例如，运行如下命令即可安装test.apk包：

```
adb install test.apk
```

如果希望从Android系统中删除指定软件包，则可使用如下命令：

```
adb uninstall [-k] <package>
```

上面的命令格式指定删除<package>代表的APK包。其中-k表示只删除该应用程序，但保留该程序所用的数据和缓存目录。

➤➤ 1.3.5 使用mksdcard管理虚拟SD卡

正如前面在Android SDK和AVD管理器中所见到的，我们可以在创建AVD设备时创建一个虚拟SD卡。实际上还可以使用mksdcard命令来单独创建一个虚拟存储卡。

mksdcard命令的语法格式如下：

```
mksdcard [-l label] <size> <file>
```

上面的命令格式中<size>指定虚拟SD卡的大小，<file>指定保存虚拟SD卡的文件镜像。

例如如下命令：

```
mksdcard 64M D:\avds\.android\avd\leegang.avd\sdcard.img
```

上面命令创建了一个大小为64MB的虚拟SD卡，该SD卡对应的镜像文件为D:\avds\.android\avd\leegang.avd\sdcard.img。

如果希望在启动模拟器时使用指定的虚拟SD卡，则在启动模拟器时增加-sdcard <file>选项，其中<file>代表虚拟SD卡的文件镜像。例如如下命令：

```
emulator -avd crazyit -sdcard d:\sdcard.img
```

到此为止，我们已经成功地安装了 Android SDK、配置了 Android 开发环境，并且对 Android 相关开发工具都有了一个大致的了解，接下来正式开始 Android 应用开发。

1.4 开始第一个 Android 应用

无须担心，Android 应用的开发十分简单！Android 应用程序建立在应用程序框架之上，所以 Android 编程就是面向应用程序框架 API 编程——这种开发方式与编写普通的 Java SE 应用程序并没有太大的区别，只是 Android 新增了一些 API 而已。

1.4.1 使用 Android Studio 开发第一个 Android 应用

使用 Android Studio 开发 Android 应用非常方便，因为 Android Studio 会为我们自动完成许多工作。使用 Android Studio 开发 Android 应用大致需要如下 3 步。

1. 创建一个 Android 项目或 Android 模块。
2. 在 XML 布局文件中定义应用程序的用户界面。
3. 在 Java 代码中编写业务实现。

上面 3 个步骤只是最粗粒度的归纳。下面以开发一个 HelloWorld 应用为例来介绍 Android 开发。详细步骤如下。

1. 通过 Android Studio 主菜单的“File”→“New Project...”菜单项，创建一个 Android Studio 项目。Android Studio 弹出如图 1.8 所示的对话框。
2. 在图 1.8 所示窗口中输入项目名、公司域名、包名和项目存储路径，然后单击“Next”按钮，接下来即可看到如图 1.9 所示的选择 Android SDK 版本的对话框。

提示： Android 应用的包名非常重要，Android 应用的包名可作为 Android 应用的唯一标识。

3. 在图 1.9 所示对话框中根据业务需要选择 SDK 的最低版本要求，此处会选择“API 21 Android 5.0”作为最低的版本要求。选好最低的 SDK 版本要求之后，单击“Next”按钮，接下来即可看到如图 1.10 所示的添加 Activity 的对话框。
4. 在图 1.10 所示对话框中选择“Blank Activity”（空 Activity，这是 Android 应用中最简单的 Activity），然后单击“Next”按钮，即可看到如图 1.33 所示的对话框。

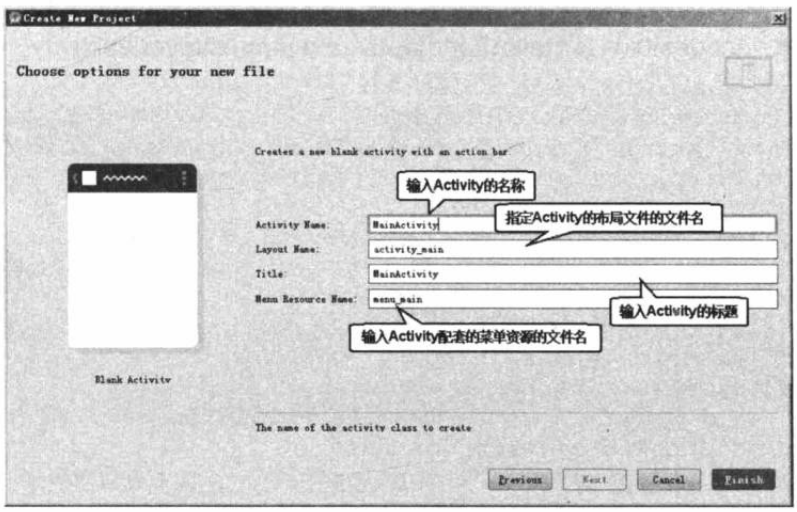


图 1.33 为创建 Activity 设置信息

5 单击“Finish”按钮，Android Studio 即成功创建了一个项目，该项目包含一个 Android 应用。

提示：

虽然 Android Studio 项目可以包含多个 Android 应用，但本书为了方便读者浏览光盘项目，将会为每个 Android 应用都创建一个 Android Studio 项目。

Android 项目创建完成后将看到如图 1.34 所示的项目结构。

6 Android 项目的 layout 目录下有一个 activity_main.xml 文件，该文件用于定义 Android 应用用户界面。在 Android Studio 工具中打开该文件，将看到如图 1.35 所示的界面。

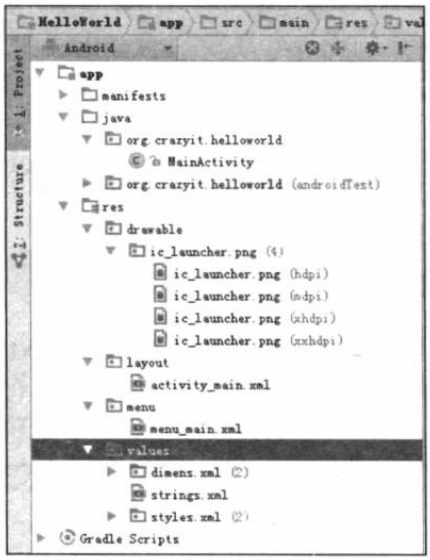


图 1.34 Android 项目结构

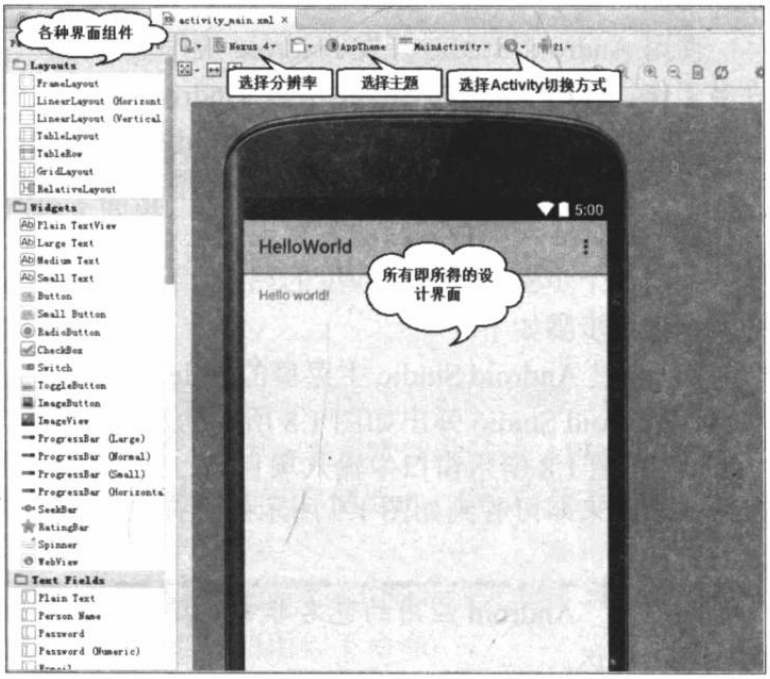


图 1.35 Android Studio 提供的界面设计工具

图 1.35 所示的“所见即所得”的设计界面十分简单，有过网页编辑经验的读者可能对 Dreamweaver 之类的工具比较熟悉，那么可以把这个界面近似地当成 Dreamweaver。

从图 1.35 所示界面的控件面板中向程序拖入一个 Button 控件（按钮），再切换到源代码编写界面，将 activity_main.xml 文件（该文件就是 MainActivity 配套的 XML 布局文件）修改为如下形式。

程序清单：\codes\01\1.4\HelloWorld\app\src\main\res\layout\activity_main.xml

```
<RelativeLayout android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <TextView
        android:id="@+id/show"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_alignParentTop="true"
        android:text="@string/hello_world" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/show"
```

```
        android:text="单击我"
        android:onClick="clickHandler"/>
</RelativeLayout>
```

上面 XML 文档的根元素是 RelativeLayout，它代表一个相对布局，在该界面布局里包含如下两个 UI 控件。

- TextView：代表一个文本框。
- Button：代表一个普通按钮。

在 Android 用户界面设计中，各种界面布局元素将会在后面进行详细介绍，不同 UI 组件也会在后面进行详细介绍。此处只想说明 UI 组件上的几个通用属性。

- android:id：指定该控件的唯一标识，在 Java 程序中可通过 findViewById("id")来获取指定的 Android 界面组件。
- android:layout_width：指定该界面组件的宽度。如果该属性值为 match_parent，则说明该组件与其父容器具有相同的宽度；如果该属性值为 wrap_content，则说明该组件的宽度取决于它的内容——能包裹它的内容即可。
- android:layout_height：指定该界面组件的高度。如果该属性值为 match_parent，则说明该组件与其父容器具有相同的高度；如果该属性值为 wrap_content，则说明该组件的高度取决于它的内容——能包裹它的内容即可。

可能有读者感到奇怪：Android 怎么采用 XML 文件来定义用户界面呢？或者有过 Swing 编程经验的读者感到不习惯：怎么不是在 Java 代码里定义用户界面，而是在 XML 文档里定义用户界面呢？

大家要接受 Android 的这种优秀设计。Android 把用户界面放在 XML 文档中定义，就可以让 XML 文档专门负责用户 UI 设置，而 Java 程序则专门负责业务实现，这样可以降低程序的耦合性。对于不习惯这种方式的读者来说，其实可以近似地把 activity_main.xml 文件当成一个 HTML 页面——它们都通过标记语言来定义用户界面。区别在于 HTML 页面使用 HTML 标签，而 activity_main.xml 文件则使用 Android 标签。

7 Android Studio 项目下的 app\src 目录是 Android 项目的源代码，该目录的 main\java\org\crazyit\helloworld 目录下有一个 MainActivity.java 文件，它就是 Android 项目的 Java 文件。打开该文件，将该文件编辑为如下形式。

程序清单：codes\01\1.4\HelloWorld\app\src\main\java\org\crazyit\helloworld\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 使用 activity_main.xml 文件定义的界面布局
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
    public void clickHandler(View source)
    {
        // 获取 UI 界面中 ID 为 R.id.show 的文本框
        TextView tv = (TextView) findViewById(R.id.show);
```

```
        // 改变文本框的文本内容
        tv.setText("Hello Android-" + new java.util.Date());
    }
}
```

对于一个有不错的 Java 基础的读者来说，上面这个程序十分简单，它只做了如下两件事情。

① 设置该 Activity 使用 activity_main.xml 文件定义的界面布局作为用户界面。

② 定义了一个 clickHandler()方法作为按钮的事件处理方法——在处理方法中改变 ID 为 R.id.show 的文本框的内容。

至此，这个 HelloWorld 级的 Android 应用已经开发完成了。

提示：

前面介绍 hello_world.xml 文件时说过，读者可以把该文件当成一份 HTML 代码，Java 程序中通过 findViewById()方法即可获取指定 ID 的界面控件，实际上读者完全可以把 findViewById()类比成 JavaScript 代码中的 getElementById()。

1.4.2 通过 Andorid Studio 运行 Android 应用

通过 Andorid Studio 来运行 Android 应用非常简单，只要如下两步即可。

① 运行 Genymotion 模拟器（或通过 Android 提供的 AVD 管理器或直接使用 emulator 命令运行指定的 AVD 虚拟机，但 Android 自带的虚拟机比较慢）。如果打算用真机作为运行、调试环境，则使用 USB 线连接手机，并打开手机的调试模式。

② 在 Android Studio 的工具条中选中 Android app，然后单击工具条中的“运行”按钮，如图 1.36 所示。

单击“运行”按钮之后，Android Studio 将会弹出如图 1.37 所示的对话框，该对话框询问用户要将该 Android 应用部署到哪个设备上？

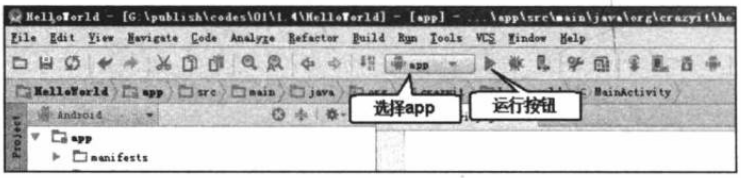


图 1.36　运行 Android 应用

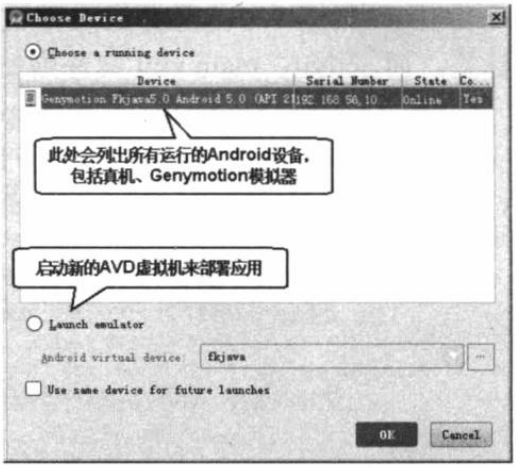


图 1.37　选择要将 Android 应用部署到哪个设备上

在图 1.37 所示对话框中选择要部署 Andriod 应用的设备，然后单击“OK”按钮，Android Studio 将会把该应用部署到指定设备上。此时打开“手机”，进入程序列表，即可看到刚刚开发的 HelloWorld 程序，如图 1.38 所示。

单击“HelloWorld”程序项，即可在“手机”上运行刚刚开发的 HelloWorld 程序，运行效果如图 1.39 所示。

图 1.38 安装成功的 Android 应用

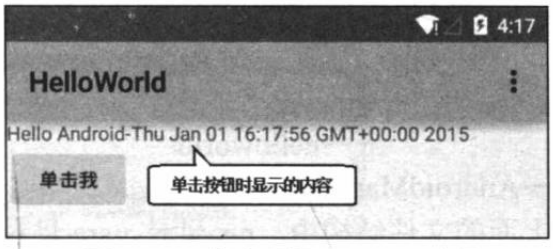


图 1.39 成功运行 Android 应用

1.5 Android 应用结构分析

使用 Android Studio 开发 Android 应用简单、方便，除了创建 Android 项目，开发者只需做两件事情：使用 activity_main.xml 文件定义用户界面；打开 Java 源代码编写业务实现。但对于一个喜欢“穷根究底”的学习者来说，这种开发方式不免让其迷惑：

- findViewById(R.id.show);代码中的 R.id.show 是什么？从哪里来？
- 为何 setContentView(R.layout.activity_main);代码设置使用 activity_main.xml 文件定义的界面布局？

……

实际上，喜欢“穷根究底”的学习者才是真正好的学习者。借用拿破仑的一句话：不想当将军的士兵不是好士兵。类似地，也可以说：不喜欢探究原理、机制的程序员不是好程序员。

每次看到那些局限于特定 IDE 工具，只能在特定 IDE 里做事，而且自我感觉良好的学习者、工作者，笔者都忍不住十分遗憾：程序员，又少了一个。

下面将带领大家“徒手”开发一个 Android 项目，这样所有的事情都是由开发者自己完成的——自然对 Android 开发的每个细节更加熟知，以后用 IDE 工具开发才能做到不仅知其然，也知其所以然。

1.5.1 创建一个 Android 应用

前面介绍 android 命令时已经提到，该命令有一个 create project 子命令，该子命令可用于“手动”创建一个 Android 应用，在命令行窗口输入如下命令：

```
android create project -n HelloWorld -t android-21 -p HelloWorld
-k org.crazyit.helloworld -a HelloWorld
```

提示：

在上面的命令中，-n 选项指定创建项目的名称；-t 选项指定项目针对的 Android 平台；-p 选项指定该项目的保存路径；-k 选项指定该项目的包名；-a 选项指定 Activity 的名称。

运行上面的命令，可以在当前目录的 HelloWorld 子目录下创建一个 Android 项目。进入该项目所在的目录，可以看到如下两个必要的文件夹：

```
HelloWorld
├─libs
├─res
│  ├─drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi
│  ├─layout
│  └─values
└─src（存放 Java 源文件）
   └─org
      └─crazyit
         └─helloworld
└─AndroidManifest.xml
```

在上面的文件结构中，res 目录、src 目录、AndroidManifest.xml 文件是 Android 项目必需的。其他目录、文件都是可选的。

- res 目录存放 Android 项目的各种资源文件，比如 layout 存放界面布局文件，values 目录下则存放各种 XML 格式的资源文件，例如字符串资源文件：strings.xml；颜色资源文件：colors.xml；尺寸资源文件：dimens.xml。drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi 这 4 个子目录则分别用于存放低分辨率、中分辨率、高分辨率、超高分辨率的 4 种图片文件。
- src 目录只是一个普通的、保存 Java 源文件的目录。
- AndroidManifest.xml 文件是 Android 项目的系统清单文件，它用于控制 Android 应用的名称、图标、访问权限等整体属性。除此之外，Andriod 应用的 Activity、Service、ContentProvider、BroadcastReceiver 这 4 大组件都需要在该文件中配置。

除此之外，还可以在 HelloWorld 目录下看到一个 build.xml 文件，这是 Android 为该项目提供的一个 Ant 生成文件。通过该生成文件，开发者可以通过 Ant 来生成、安装 Android 项目。

使用 Android Studio 开发的 Android 项目结构也与此类似，在 Android Studio 项目下包含一个 app 目录，该目录下包含如下 3 个子目录。

- build：Android Studio 自动生成的各种源文件（包括 R.java 文件也放在该目录的子目录下）。
- libs：存储 Android 项目所需的第三方 JAR 包。
- src：存储 Android 项目开发的各种源文件，包括各种 Java 源文件（放在 main\java 子目录下）、各种资源文件（放在 main\res 子目录下）和 AndroidManifest.xml 文件。除此之外，src 目录下还包含一个 androidTest 子目录，该目录下存放的是 Android 测试项目。

提示：

Ant 生成的 Android 项目（包括早期 ADT 创建的 Android 项目）主要对应于 Android Studio 创建的 Android 项目中 app/src/目录下的 main 子目录。app/src/目录下的 androidTest 子目录则用于编写单元测试的测试用例；app 目录下的 libs 目录则与 Ant 生成的 Android 项目（包括早期 ADT 创建的 Android 项目）下的 libs 目录完全相同。Android Studio 创建的 Android 项目根目录下的其他文件和文件夹，只是该项目相关的配置文件，相当于 Eclipse 工作空间下的配置文件，与具体项目无关（Android Studio 的项目对应于 Eclipse 的工作空间，Android Studio 的模块才对应于 Eclipse 的模块）。

对于 Android 开发者而言，重点关注的就是两部分。

- src 目录下（对于 Android Studio 项目的 app\src\main\java 子目录）的各种 Java 文件。
- res 目录下（对于 Android Studio 项目的 app\src\main\res 子目录）的各种资源文件。

与前面介绍的 Android 开发相似的是，此处的开发同样是先编辑 XML 格式的界面布局文件，再编辑相应的 Java 文件。编写文件的内容与前一个示例并没有太大的区别，此处不再赘述。

编辑完成后启动命令行窗口，并转入 HelloWorld 目录下，执行 ant help 命令将可以看到如图 1.40 所示的输出。

```
G:\publish\codes\01\1.5\HelloWorld>ant help
Buildfile: G:\publish\codes\01\1.5\HelloWorld\build.xml

help:
     [echo] Android Ant Build. Available targets:
     [echo]    help:      Displays this help.
     [echo]    clean:     Removes output files created by other targets.
     [echo]               This calls the same target on all dependent projects.
     [echo]               Use 'ant nodeps clean' to only clean the local project
     [echo]    debug:     Builds the application and signs it with a debug key.
     [echo]               The 'nodeps' target can be used to only build the
     [echo]               current project and ignore the libraries using:
     [echo]               'ant nodeps debug'
     [echo]    release:   Builds the application. The generated apk file must be
     [echo]               signed before it is published.
     [echo]               The 'nodeps' target can be used to only build the
     [echo]               current project and ignore the libraries using:
     [echo]               'ant nodeps release'
     [echo]    instrument:Builds an instrumented package and signs it with a
     [echo]               debug key.
     [echo]    test:      Runs the tests. Project must be a test project and
     [echo]               must have been built. Typical usage would be:
     [echo]                   ant [emma] debug install test
     [echo]    emma:      Transiently enables code coverage for subsequent
     [echo]               targets.
     [echo]    install:   Installs the newly build package. Must either be used
     [echo]               in conjunction with a build target (debug/release/
     [echo]               instrument) or with the proper suffix indicating
     [echo]               which package to install (see below).
```

图 1.40 Android 项目的生成文件

从图 1.40 可以看出，Android 项目提供的 build.xml 文件包含了如下常用的生成 target。

- clean：清除项目生成的内容——也就是恢复原来的样子。
- debug：打包一个调试用的 Android 应用的 APK 包，使用 debug key 进行签名。
- release：打包一个发布用的 Android 应用的 APK 包。
- test：运行测试。要求该项目必须是一个测试项目。
- install：将生成的调试用的 APK 包安装到模拟器上。
- uninstall：从模拟器上卸载该应用程序。

提示：

Ant 是一个非常简洁、易用的项目生成工具。对于绝大部分 Java 开发者来说，使用 Ant 应该是一项最基本的技能。考虑到有些读者对 Ant 用法不熟，此处简略介绍 Ant 的一些安装和使用方法。

① 登录 http://ant.apache.org/bindownload.cgi 站点下载 Ant 最新版，笔者使用的是 1.9.4，建议下载该版本。Windows 平台下载*.zip 压缩包，而 Linux 平台则下载.gz 压缩包。

② 将下载到的压缩文件解压缩到任意路径，例如，本书解压缩到 D:\根路径下。

③ Ant 的运行需要两个环境变量。① JAVA_HOME：该环境变量应指向 JDK 的安装路径。如果已经成功安装了 Android Studio，则该环境变量应该已经是正确的。② ANT_HOME：该环境变量应指向 Ant 的安装路径。Ant 的安装路径就是前

面释放 Ant 压缩文件的路径。Ant 安装路径下应该包含 bin、docs、etc 和 lib 四个文件夹。

④ Ant 工具的关键命令就是%ANT_HOME%/bin 路径下的 ant.bat 命令，如果读者希望操作系统可以识别该命令，还应该将%ANT_HOME%/bin 路径添加到操作系统的 PATH 环境变量之中。

经过上面 4 个步骤，即可在命令行窗口使用 ant.bat 命令了，它就是 Ant 工具。至于 Ant 更详细的用法介绍，请参阅疯狂 Java 体系的《轻量级 Java EE 企业应用实战》。

先执行 ant release 命令来发布该项目，发布完成后看到 HelloWorld 目录下出现了两个子目录。

- bin：该目录用于存放生成的目标文件，如 Java 的二进制文件、资源打包文件（.ap_后缀）、Dalvik 虚拟机的可执行性文件（.dex 后缀）等。

提示：

早期的 Android 系统需要用 Dalvik 虚拟机来运行 Android 应用。

- gen：该目录用于保存自动生成的、位于 Android 项目包下的 R.java 文件。

前面我们编写 Android 程序代码时多次使用了 R.layout.main、R.id.show、R.id.ok……现在读者应该明白这里 R 是什么了，原来它是 Android 项目自动生成的一个 Java 类。接下来将会详细介绍 R.java 文件。

1.5.2 自动生成的 R.java

打开 gen\org\crazyit\helloworld 目录下的 R.java 文件，看到如下代码。

程序清单：codes\01\1.5\HelloWorld\gen\org\crazyit\helloworld\R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package org.crazyit.helloworld;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int show=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

通过 R.java 类中的注释可以看出，R.java 文件是由 AAPT 工具根据应用中的资源文件自动生成的，因此可以把 R.java 理解成 Android 应用的资源字典。

AAPT 生成 R.java 文件的规则主要是如下两条。

- 每类资源都对应于 R 类的一个内部类。比如所有界面布局资源对应于 layout 内部类；所有字符串资源对应于 string 内部类；所有标识符资源对应于 id 内部类。
- 每个具体的资源项都对应于内部类的一个 public static final int 类型的 Field。例如，前面在界面布局文件中用到了 show 标识符，因此 R.id 类里就包含了这两个 Field；由于 drawable-xxxx 文件夹里包含了 ic_launcher.png 图片，因此 R.drawable 类里就包含了 ic_launcher Field。

随着我们不断地向 Android 项目中添加资源，R.java 文件的内容也会越来越多。后面还会详细介绍 Android 资源访问的相关内容，因此后面还会进一步说明不同资源在 R.java 文件中的表现形式。

1.5.3 res 目录说明

Android 应用的 res 目录是一个特殊的项目，该项目里存放了 Android 应用所用的全部资源，包括图片资源、字符串资源、颜色资源、尺寸资源等——后面还会进一步介绍 Android 应用中资源的用法，此处先对 res 目录的资源进行简单的归纳。

Android 按照约定，将不同的资源放在不同的文件夹内，这样可以方便地让 AAPT 工具来扫描这些资源，并为它们生成对应的资源清单类：R.java。

以/res/value/strings.xml 文件来说，该文件的内容十分简单，它只是定义了一个个的字符串常量，如以下代码所示。

程序清单：codes\01\1.5\HelloWorld\res\values\strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
</resources>
```

上面的资源文件中定义了一个字符串常量，常量的值为 HelloWorld，该字符串常量的名称为 app_name。一旦定义了这份资源文件之后，Android 项目就允许分别在 Java 代码、XML 文件中使用这份资源文件中的字符串资源。

1. 在 Java 代码中使用资源

为了在 Java 代码中使用资源，AAPT 会为 Android 项目自动生成一份 R.java 文件，R 类里为每份资源分别定义了一个内部类，其中每个资源项对应于内部类里一个 int 类型的 Field。例如，上面的字符串资源文件对应于 R.java 里的如下内容：

```
// 对应于一份资源
public static final class string {
    // 对应于一个资源项
    public static final int app_name=0x7f040000;
}
```

借助于 AAPT 自动生成 R 类的帮助，在 Java 代码中可通过 R.string.app_name 引用到"HelloWorld"字符串常量。

2. 在 XML 文件中使用资源

在 XML 文件中使用资源更加简单，只要按如下格式来访问即可：

```
@<资源对应的内部类的类名>/<资源项的名称>
```

例如，我们要访问上面的字符串资源中定义的"HelloWorld"字符串常量，则使用如下形式来引用即可：

```
@string/app_name
```

但有一种情况例外，当我们在 XML 文件中使用标识符时——这些标识符无须使用专门的资源进行定义，直接在 XML 文档中按如下格式分配标识符即可：

```
@+id/<标识符代号>
```

例如，使用如下代码为一个组件分配标识符：

```
android:id="@+id/ok"
```

上面的代码为该组件分配了一个标识符，接下来就可以在程序中引用该组件了。

如果希望在 Java 代码中获取该组件，通过调用 Activity 的 findViewById()方法即可实现。

如果希望在 XML 文件中获取该组件，则可通过资源引用的方式来引用它，语法如下：

```
@id/<标识符代号>
```

1.5.4 Android 应用的清单文件：AndroidManifest.xml

AndroidManifest.xml 清单文件是每个 Android 项目所必需的，它是整个 Android 应用的全局描述文件。AndroidManifest.xml 清单文件说明了该应用的名称、所使用的图标以及包含的组件等。

AndroidManifest.xml 清单文件通常可以包含如下信息。

- 应用程序的包名，该包名将会作为该应用的唯一标识。
- 应用程序所包含的组件，如 Activity、Service、BroadcastReceiver 和 ContentProvider 等。
- 应用程序兼容的最低版本。
- 应用程序使用系统所需的权限声明。
- 其他程序访问该程序所需的权限声明。

不管是 Android Studio 工具还是 android.bat 命令，它们所创建的 Android 项目都有一个 AndroidManifest.xml 文件。但随着不断地进行开发，可能需要对 AndroidManifest.xml 清单文件进行适当的修改。

下面是一份简单的 AndroidManifest.xml 清单文件。

程序清单：codes\01\1.5\HelloWorld\AndroidManifest.xml

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定该 Android 应用的包名，
    该包名可用于唯一地标识该应用 -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.crazyit.helloworld"
    android:versionCode="1"
    android:versionName="1.0">
    <!-- 指定 Android 应用标签、图标 -->
    <application android:label="@string/app_name"
        android:icon="@drawable/ic_launcher">
        <!-- 定义 Android 应用的一个组件：Activity，
            该 Activity 的类为 HelloWorld，并指定该 Activity 的标签-->
        <activity android:name="HelloWorld"
                android:label="@string/app_name">
            <intent-filter>
                <!-- 指定该 Activity 是程序的入口 -->
                <action android:name="android.intent.action.MAIN" />
                <!-- 指定加载该应用时运行该 Activity -->
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

```
</manifest>
```

上面这份 AndroidManifest.xml 清单文件中的注释已经大致说明了各元素的作用，故不再详细说明每个元素。上面的清单文件中有两处用到了资源。

- android:label="@string/app_name"，这说明该应用的标签（Label）为/res/value 目录下 strings.xml 文件中名为 app_name 的字符串值。
- android:icon="@drawable/ic_launcher"，这说明该应用的图标为/res/drawable-l/m/hdpi 目录下主文件名为 icon 的图片。

1.5.5 应用程序权限说明

一个 Android 应用可能需要权限才能调用 Android 系统的功能；一个 Android 应用也可能被其他应用调用，因此它也需要声明调用自身所需要的权限。

1. 声明运行该应用本身所需要的权限

通过为<manifest.../>元素添加<uses-permission.../>子元素即可为程序本身声明权限。

例如，在<manifest.../>元素里添加如下代码：

```
<!-- 声明该应用本身需要打电话的权限 -->
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

2. 声明调用该应用所需的权限

通过为应用的各组件元素，如<activity.../>元素添加<uses-permission.../>子元素即可声明调用该程序所需的权限。

例如，在<activity.../>元素里添加如下代码：

```
<!-- 声明该应用本身需要发送短信的权限 -->
<uses-permission android:name="android.permission.SEND_SMS"/>
```

通过上面的介绍可以看出，<uses-permission.../>元素的用法倒不难，但到底有多少权限呢？实际上 Android 提供了大量的权限，这些权限都位于 Manifest.permission 类中。一般来说，有如表 1.1 所示的常用权限。

表 1.1 Android 系统的常用权限

权　限	说　明
ACCESS_NETWORK_STATE	允许应用程序获取网络状态信息的权限
ACCESS_WIFI_STATE	允许应用程序获取 Wi-Fi 网络状态信息的权限
BATTERY_STATS	允许应用程序获取电池状态信息的权限
BLUETOOTH	允许应用程序连接匹配的蓝牙设备的权限
BLUETOOTH_ADMIN	允许应用程序发现匹配的蓝牙设备的权限
BROADCAST_SMS	允许应用程序广播收到短信提醒的权限
CALL_PHONE	允许应用程序拨打电话的权限
CAMERA	允许应用程序使用照相机的权限
CHANGE_NETWORK_STATE	允许应用程序改变网络连接状态的权限
CHANGE_WIFI_STATE	允许应用程序改变 Wi-Fi 网络连接状态的权限
DELETE_CACHE_FILES	允许应用程序删除缓存文件的权限
DELETE_PACKAGES	允许应用程序删除安装包的权限
FLASHLIGHT	允许应用程序访问闪光灯的权限
INTERNET	允许应用程序打开网络 Socket 的权限
MODIFY_AUDIO_SETTINGS	允许应用程序修改全局声音设置的权限
PROCESS_OUTGOING_CALLS	允许应用程序监听、控制、取消呼出电话的权限

续表

权　限	说　明
READ_CONTACTS	允许应用程序读取用户的联系人数据的权限
READ_HISTORY_BOOKMARKS	允许应用程序读取历史书签的权限
READ_OWNER_DATA	允许应用程序读取用户数据的权限
READ_PHONE_STATE	允许应用程序读取电话状态的权限
READ_PHONE_SMS	允许应用程序读取短信的权限
REBOOT	允许应用程序重启系统的权限
RECEIVE_MMS	允许应用程序接收、监控、处理彩信的权限
RECEIVE_SMS	允许应用程序接收、监控、处理短信的权限
RECORD_AUDIO	允许应用程序录音的权限
SEND_SMS	允许应用程序发送短信的权限
SET_ORIENTATION	允许应用程序旋转屏幕的权限
SET_TIME	允许应用程序设置时间的权限
SET_TIME_ZONE	允许应用程序设置时区的权限
SET_WALLPAPER	允许应用程序设置桌面壁纸的权限
VIBRATE	允许应用程序控制振动器的权限
WRITE_CONTACTS	允许应用程序写入用户联系人的权限
WRITE_HISTORY_BOOKMARKS	允许应用程序写历史书签的权限
WRITE_OWNER_DATA	允许应用程序写用户数据的权限
WRITE_SMS	允许应用程序写短信的权限

1.6 Android 应用的基本组件介绍

Android 应用通常由一个或多个基本组件组成，前面我们看到 Android 应用中最常用的组件就是 Activity。事实上 Android 应用还可能包括 Service、BroadcastReceiver、ContentProvider 等组件。本节先让读者对这些组件建立一个大致的认识，后面的章节还会对这些组件做更详细的介绍。

1.6.1 Activity 和 View

Activity 是 Android 应用中负责与用户交互的组件——大致上可以把它想象成 Swing 编程中的 JFrame 控件。不过它与 JFrame 的区别在于：JFrame 本身可以设置布局管理器，不断地向 JFrame 中添加组件，但 Activity 只能通过 setContentView(View)来显示指定组件。

View 组件是所有 UI 控件、容器控件的基类，View 组件就是 Android 应用中用户实实在在看到的部分。但 View 组件需要放到容器组件中，或者使用 Activity 将它显示出来。如果需要通过某个 Activity 把指定 View 显示出来，调用 Activity 的 setContentView()方法即可。

setContentView()方法可接受一个 View 对象作为参数，例如如下代码：

```
// 创建一个线性布局管理器
LinearLayout layout = new LinearLayout(this);
// 设置该 Activity 显示 layout
setContentView(layout);
```

上面的程序通过代码创建了一个 LinearLayout 对象（它是 ViewGroup 的子类，ViewGroup 又是 View 的子类），接着调用 Activity 的 setContentView(layout)把这个布局管理器显示出来。

setContentView()方法也可接受一个布局管理资源的 ID 作为参数，例如如下代码：

```
// 设置该 Activity 显示 main.xml 文件定义的 View
setContentView(R.layout.main);
```

从这个角度来看，大致上可以把Activity理解成Swing中的JFrame组件。当然，Activity可以完成的功能比JFrame更多，此处只是简单地类比一下。

提示：

实际上Activity是Window的容器，Activity包含一个getWindow()方法，该方法返回该Activity所包含的窗口。对于Activity而言，开发者一般不需要关心Window对象。如果应用程序不调用Activity的setContentView()来设置该窗口显示的内容，那么该程序将显示一个空窗口。

Activity为Android应用提供了可视化用户界面，如果该Android应用需要多个用户界面，那么这个Android应用将会包含多个Activity，多个Activity组成Activity栈，当前活动的Activity位于栈顶。

Activity包含了一个setTheme(int resid)方法来设置其窗口的风格。例如，我们希望窗口不显示ActionBar、以对话框形式显示窗口，都可通过该方法来实现。

1.6.2 Service

Service与Activity的地位是并列的，它也代表一个单独的Android组件。Service与Activity的区别在于：Service通常位于后台运行，它一般不需要与用户交互，因此Service组件没有图形用户界面。

与Activity组件需要继承Activity基类相似，Service组件需要继承Service基类。一个Service组件被运行起来之后，它将拥有自己独立的生命周期，Service组件通常用于为其他组件提供后台服务或监控其他组件的运行状态。

1.6.3 BroadcastReceiver

BroadcastReceiver是Android应用中另一个重要的组件，顾名思义，BroadcastReceiver代表广播消息接收器。从代码实现角度来看，BroadcastReceiver非常类似于事件编程中的监听器。与普通事件监听器不同的是，普通事件监听器监听的事件源是程序中的对象；而BroadcastReceiver监听的事件源是Android应用中的其他组件。

使用BroadcastReceiver组件接收广播消息比较简单，开发者只要实现自己的BroadcastReceiver子类，并重写onReceive(Context context, Intent intent)方法即可。当其他组件通过sendBroadcast()、sendStickyBroadcast()或sendOrderedBroadcast()方法发送广播消息时，如果该BroadcastReceiver也对该消息“感兴趣”（通过IntentFilter配置），BroadcastReceiver的onReceive(Context context, Intent intent)方法将会被触发。

开发者实现了自己的BroadcastReceiver之后，通常有两种方式来注册这个系统级的“事件监听器”。

- 在Java代码中通过Context.registReceiver()方法注册BroadcastReceiver。
- 在AndroidManifest.xml文件中使用<receiver.../>元素完成注册。

读者此处只要对BroadcastReceiver有一个大致的印象即可，本书后面的章节还会详细介绍如何开发、使用BroadcastReceiver组件。

1.6.4 ContentProvider

对于Android应用而言，它们必须相互独立，各自运行在自己的进程中，如果这些Android应用之间需要实现实时的数据交换——例如，我们开发了一个发送短信的程序，当发送短信时需

要从联系人管理应用中读取指定联系人的数据——这就需要多个应用程序之间进行数据交换。

Android 系统为这种跨应用的数据交换提供了一个标准：ContentProvider。当用户实现自己的 ContentProvider 时，需要实现如下抽象方法。

- insert(Uri, ContentValues)：向 ContentProvider 插入数据。
- delete(Uri, ContentValues)：删除 ContentProvider 中指定数据。
- update(Uri, ContentValues, String, String[])：更新 ContentProvider 中指定数据。
- query(Uri, String[], String, String[],String)：从 ContentProvider 查询数据。

通常与 ContentProvider 结合使用的是 ContentResolver，一个应用程序使用 ContentProvider 暴露自己的数据，而另一个应用程序则通过 ContentResolver 来访问数据。

1.6.5 Intent 和 IntentFilter

Intent 并不是 Android 应用的组件，但它对于 Android 应用的作用非常大——它是 Android 应用内不同组件之间通信的载体。当 Android 运行时需要连接不同的组件时，通常就需要借助于 Intent 来实现。Intent 可以启动应用中另一个 Activity，也可以启动一个 Service 组件，还可以发送一条广播消息来触发系统中的 BroadcastReceiver。也就是说，Activity、Service、BroadcastReceiver 三种组件之间的通信都以 Intent 作为载体，只是不同组件使用 Intent 的机制略有区别而已。

- 当需要启动一个 Activity 时，可调用 Context 的 startActivity (Intent intent)或 startActivityForResult(Intent intent, int requestCode)方法，这两个方法中的 Intent 参数封装了需要启动的目标 Activity 的信息。
- 当需要启动一个 Service 时，可调用 Context 的 startService(Intent intent)方法或 bindService(Intent service, ServiceConnection conn, int flags)方法，这两个方法中的 Intent 参数封装了需要启动的目标 Service 的信息。
- 当需要触发一个 BroadcastReceiver 时，可调用 Context 的 sendBroadcast(Intent intent)、sendStickyBroadcast(Intent intent) 或 sendOrderedBroadcast(Intent intent, String receiverPermission)方法来发送广播消息，这三个方法中的 Intent 参数封装了需要触发的目标 BroadcastReceiver 的信息。

通过上面的介绍不难看出，Intent 封装了当前组件需要启动或触发的目标组件的信息，因此有些资料也将 Intent 翻译为“意图”。实际上 Intent 对象里封装了大量关于目标组件的信息，本书后面还会更详细地介绍 Intent 所封装的数据，此处不再深入讲解。

当一个组件通过 Intent 表示了启动或触发另一个组件的“意图”之后，这个意图可分为两类。

- **显式 Intent**：显式 Intent 明确指定需要启动或者触发的组件的类名。
- **隐式 Intent**：隐式 Intent 只是指定需要启动或者触发的组件应满足怎样的条件。

对于显式 Intent 而言，Android 系统无须对该 Intent 做任何解析，系统直接找到指定的目标组件，启动或触发它即可。

对于隐式 Intent 而言，Android 系统需要对该 Intent 进行解析，解析出它的条件，然后再去系统中查找与之匹配的目标组件。如果找到符合条件的组件，就启动或触发它们。

那么 Android 系统如何判断被调用组件是否符合隐式 Intent 呢？这就需要靠 IntentFilter 来实现了，被调用组件可通过 IntentFilter 来声明自己所满足的条件——也就是声明自己到底能处理哪些隐式 Intent。关于 Intent 和 IntentFilter 本书后面还会有进一步阐述，此处不再深入讲解。

1.7 签名 Android 应用程序

前面已经介绍过，Android 项目以它的包名作为唯一标识。如果在同一台手机上安装两个包名相同的应用，后面安装的应用就可以覆盖前面安装的应用。为了避免这种情况发生，Android 要求对作为产品发布的应用进行签名。

签名主要有如下两个作用：

- 确定发布者的身份。由于应用开发者可以通过使用相同包名来替换已经安装的程序，因此使用签名可以避免发生这种情况。
- 确保应用的完整性。签名会对应用包中的每个文件进行处理，从而确保程序包中的文件不会被替换。

通过以上介绍不难看出，Android 应用签名的作用类似于现实生活中的签名。当开发者对 Android 应用签名时，相当于告诉外界：该应用程序是由“我”开发的，“我”会对该应用负责——因为有签名（签名有密钥），别人无法冒名顶替“我”；与此同时，“我”也无法冒名顶替别人。

提示：

在应用的开发、调试阶段，Android Studio 或 Ant 工具会自动生成调试证书对 Android 应用签名，因此部署、调试前面两个示例并没有经过签名。需要指出的是，如果要正式发布一个 Android 应用，必须使用合适的数字证书来给应用程序签名，不能使用 Android Studio 或 Ant 工具生成的调试证书来发布。

1.7.1 使用 Android Studio 对 Android 应用签名

大部分时候，开发者会直接使用 Android Studio 对 Android 应用签名。使用 Android Studio 对 Android 应用签名的步骤如下。

① 单击 Android Studio 主菜单中的“Build”→“Generate Signed APK...”菜单项，Android Studio 弹出如图 1.41 所示的对话框。

② 如果系统中还没有数字证书，则可以在图 1.41 所示窗口中单击“Create new...”按钮，并按图 1.42 所示格式填写数字证书的存储路径和密码。

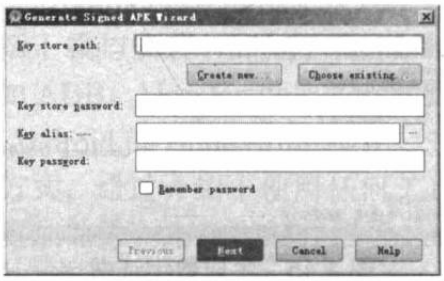


图 1.41 选择已有的 Key store 或创建新的 Key store

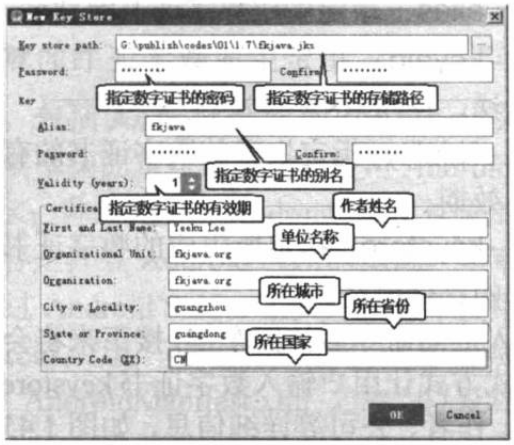


图 1.42 创建数字证书

③ 填写完成后单击“OK”按钮，Android Studio 返回图 1.41 所示的对话框，并在该对话框中使用刚刚创建的数字证书，如图 1.43 所示。

④ 单击“Next”按钮，Android Studio 将会显示如图 1.44 所示的对话框，该对话框用于

指定生成签名后的 APK 安装包的存储路径。

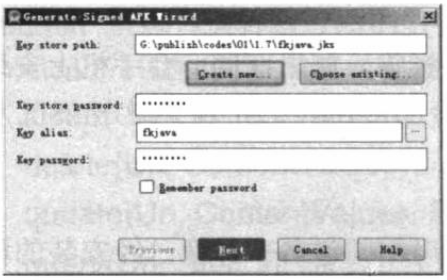


图 1.43　使用刚刚创建的数字证书

图 1.44　指定签名后的 APK 安装包的存储路径

单击“Finish”按钮，签名完成。Android Studio 将会在指定路径下生成一个签名后的 APK 安装包。

上面步骤的第 2 步用于制作新的数字证书，一旦数字证书制作完成，以后就可以直接使用该数字证书签名了。

利用已有的数字证书进行签名，请按如下步骤进行。

1 在图 1.41 所示窗口中单击“Choose existing...”按钮浏览已有的数字证书。

2 浏览到已有的数字证书之后，在该对话框下面的 Key store password、Key alias、Key password 文本框中输入已有的数字证书对应的信息，这样将会看到如图 1.43 所示的效果。剩下的事情同样是单击“Next”按钮，并选择签名 APK 的存储路径即可。

➤➤ 1.7.2　使用命令对 APK 包签名

如果不想借助于 Android Studio 对 Android 应用程序签名，或者在某些场合下，需要对一个“未签名”的 APK 包进行签名，则可通过“命令”对 Android 应用进行手动签名。

使用命令对 Android 应用签名的步骤如下。

1 创建 Key store 库。JDK 安装目录下的 bin 子目录下提供了 keytool.exe 工具来生成数字证书。在命令行窗口输入如下命令：

```
keytool -genkeypair -alias crazyit -keyalg RSA -validity 400 -keystore crazyit.jks
```

上面命令中各选项说明如下。

- –genkeypair：指定生成数字证书。
- –alias：指定生成数字证书的别名。
- –keyalg：指定生成数字证书的算法。使用 RSA 算法。
- –validity：指定生成的数字证书的有效期。
- –keystore：指定所生成的数字证书的存储路径。

输入上面命令后按回车键，接下来将会以交互式方式让用户输入数字证书 keystore 的密码、作者、公司等详细信息，如图 1.45 所示。

图 1.45　生成数字证书

提示：

第 1 步的作用是生成属于你们公司、你的数字证书，这个步骤只要做一次即可。一旦数字证书创建成功之后，只要在该证书有效期内，就可以一直重复使用该证书。

② 如果 Android 项目没有错误，在 Android Studio 的“Build”→“Make Project”即可生成未签名的 APK 安装包。在 Android Studio 项目的 app\build\outputs\apk 路径下即可找到一个 app-release-unaligned.apk 文件，该文件就是未签名的 APK 安装包。

提示：

第 2 步的作用是生成一个未签名的 APK 安装包，如果本来已有这个未签名的安装包，或者该安装包是你们委托第三方公司开发的、第三方公司负责提供该未签名的安装包，那么这个步骤是可以省略的。

③ 使用 jarsigner 命令对未签名的 APK 安装包进行签名。JDK 安装目录下的 bin 子目录下提供了 jarsigner.exe 工具进行签名。在命令行窗口输入如下命令：

```
jarsigner -verbose -keystore crazyit.jks -signedjar
 HelloWorld_crazyit.apk app-release-unaligned.apk crazyit
```

上面命令中各选项说明如下。

- –verbose：指定生成详细输出。
- –keystore：指定数字证书的存储路径。
- –signedjar：该选项的 3 个参数分别为签名后的 APK 包、未签名的 APK 包、数字证书的别名。

输入上面命令后按回车键，接下来将会以交互式方式让用户输入数字证书 keystore 的密码，如图 1.46 所示。

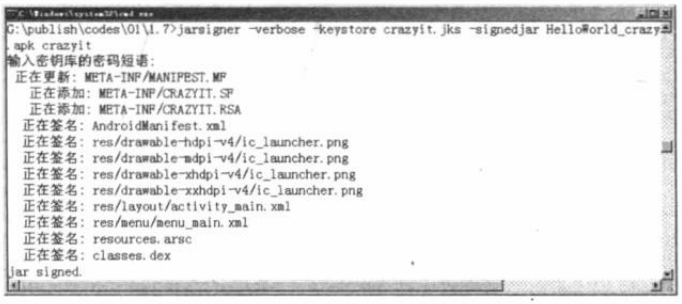


图 1.46 执行数字签名

1.8 本章小结

本章简要介绍了 Android 应用开发的背景知识，包括 Android 是什么，它是干什么的；简要介绍了 Android 的发展历史及现状。读者本章需要掌握的重点是搭建、使用 Android 开发平台，包括下载与安装 Android Studio 和 Android SDK；除此之外，Android SDK 提供的各种小工具，如 ADB、Monitor 也是需要掌握的，这些内容是开发 Android 应用的基础。本章还介绍了一个 Android 的 HelloWorld 应用，分别讲解了通过 Android Studio 工具开发和不使用工具开发两种方式；通过介绍不使用任何工具来开发 Android 应用，可以让读者对 Android 应用的程序结构更加熟悉。本章详细介绍了 Android 应用的 AndroidManifest.xml 文件，以及如何在该文件中管理程序权限。

CHAPTER

2

第2章 Android 应用的界面编程

本章要点

- Android 的程序界面与 View 组件
- View 组件与 ViewGroup 组件
- Android 控制程序界面的 3 种方式
- 通过继承 View 开发自定义 View
- Android 常见的布局管理器
- 文本框组件：TextView 和 EditText
- 按钮组件：Button
- 特殊的按钮组件：RadioButton、CheckBox、ToggleButton 和 Switch
- 时间显示组件：AnalogClock 与 TextClock
- 图片浏览组件：ImageView、ImageButton、ZoomButton 与 QuickContactBadge
- AdapterView 组件与 Adapter
- ListView 和 GridView 的功能与用法
- ExpandableListView 组件的功能与用法
- Spinner 和 Gallery 的功能与用法
- AutoCompleteTextView 组件的功能与用法
- AdapterViewFlipper 和 StackView 的功能与用法
- ProgressBar 进度条的功能与用法
- SeekBar 的功能与用法
- RatingBar 的功能与用法
- ViewAnimator 和 ViewSwitch 的功能与用法
- ImageSwitch、TextSwitch 和 ViewFlipper 的功能与用法
- 使用 Toast 创建简单提示
- CalendarView 的功能与用法
- DatePicker、TimerPicker 和 NumberPicker 的功能与用法
- SearchView 的功能与用法
- TabHost 的功能与用法
- ScrollView 的功能与用法
- 使用 Notification 发送全局通知
- 使用 AlertDialog 创建对话框
- 使用 AlertDialog 创建各种复杂的对话框
- 具有对话框风格的窗口
- 使用 PopupWindow 创建对话框
- 开发选项菜单和子菜单
- 为菜单项提供响应
- 使用 ActionBar 显示选项菜单
- 为 ActionBar 添加 ActionView
- ActionBar 结合 Fragment 实现 Tab 导航
- ActionBar 结合 Fragment 实现下拉式导航

Android 应用开发的一项内容就是用户界面的开发。不管应用实际包含的逻辑多么复杂、多么优秀，如果这个应用没有提供友好的图形用户界面（Graphics User Interface，GUI），也将很难吸引最终用户。相反，如果为应用程序提供了友好的图形用户界面，最终用户通过手指拖动、点击等动作就可以操作整个应用，这个应用程序就会受欢迎得多（实际上，Windows 之所以广为人知，其最初的吸引力就是来自于它所提供的图形用户界面）。作为一个程序设计者，必须优先考虑用户的感受，一定要让用户感到“爽”，我们的程序才会被需要、被使用，这样的程序才有价值。

Android 提供了大量功能丰富的 UI 组件，开发者只要按一定规律把这些 UI 组件组合起来——就像小朋友“搭积木”一样，把这些 UI 组件搭建在一起就可以开发出优秀的图形用户界面。为了让这些 UI 组件能响应用户的手指触摸、键盘动作，Android 也提供了事件响应机制，这样保证图形界面应用可响应用户的交互操作。

通过学习本章，读者应该能开发出漂亮的图形用户界面，这些图形用户界面是 Android 应用开发的基础，也是非常重要的组成部分。

2.1 界面编程与视图（View）组件

Android 应用是运行于手机系统上的程序，这种程序给用户的第一印象就是用户界面。从市场的角度来看，所有开发者都应充分重视 Android 应用的用户界面。Android 提供了非常丰富的用户界面组件，借助于这些用户界面组件，开发者可以非常方便地进行用户界面开发，而且可以开发出非常优秀的用户界面。

2.1.1 视图组件与容器组件

Android 应用的绝大部分 UI 组件都放在 android.widget 包及其子包、android.view 包及其子包中，Android 应用的所有 UI 组件都继承了 View 类，View 组件非常类似于 Swing 编程的 JPanel，它代表一个空白的矩形区域。

View 类还有一个重要的子类：ViewGroup，但 ViewGroup 通常作为其他组件的容器使用。

Android 的所有 UI 组件都是建立在 View、ViewGroup 基础之上的，Android 采用了“组合器”设计模式来设计 View 和 ViewGroup：ViewGroup 是 View 的子类，因此 ViewGroup 也可被当成 View 使用。对于一个 Android 应用的图形用户界面来说，ViewGroup 作为容器来盛装其他组件，而 ViewGroup 里除了可以包含普通 View 组件之外，还可以再次包含 ViewGroup 组件。

图2.1显示了Android图形用户界面的组件层次图。

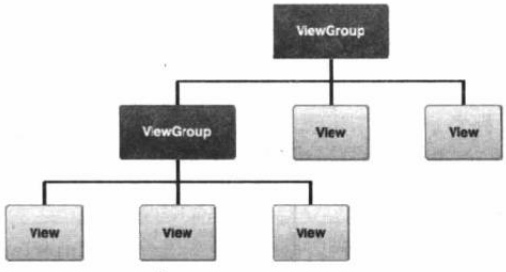


图 2.1 图形用户界面的组件层次

图 2.1 来自 Android 文档。对于每个 Android 开发者而言，Android 提供的官方文档是必看的。下面简单介绍读者应该如何查看 Android 文档——这实际上是一种学习方法。实际上，笔者常常觉得掌握学习方法比记住几个知识点更重要。

在第 1 章在线安装 Android SDK 组件时，通过图 1.12 所示窗口选择 Android 工具时勾选“Documentation for Android SDK”项，就会将 Android 文档安装到本地磁盘。一旦我们将 Android 文档安装到本地磁盘，就可以在 Android SDK 安装目录下找到 docs 子目录，打开 docs 子目录下的 index.html 页面，并单击该页面上方的“Develop”→“API Guides”（开发指南）标签页，用户将看到如图 2.2 所示的页面。

图 2.2　Android 开发指南

图 2.2 所示就是 Android 官方提供的开发指南文档，这份文档也是笔者当初学习、开发 Android 应用的重要文档。

提示：

如果具有良好的英文阅读能力，而且 Java 基本功扎实，学习 Android 完全可以不用购买任何图书，直接阅读这份 API Guides 也是很好的学习方法。

单击图 2.2 所示页面中的 Reference 标签页，接下来所看到的就是 Android 的 API 文档，如图 2.3 所示。

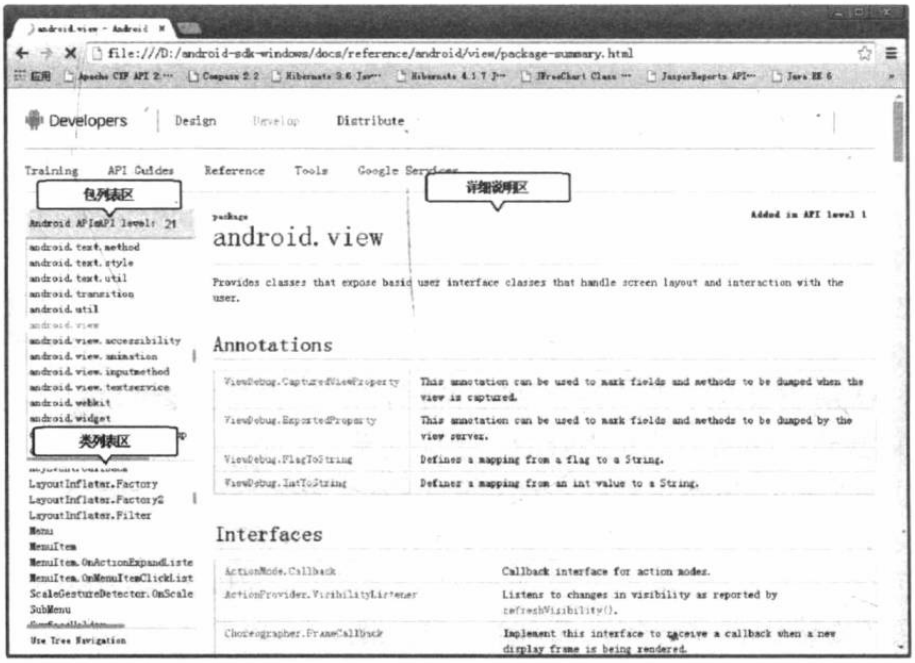


图 2.3　Android 的 API 文档

图 2.3 所示的 API 文档与我们熟悉的 API 文档大致相同，最大的区别在于 Android 的 API 文档并未在类列表区直接列出所有类，只有当开发者选择指定包之后，类列表区才会列出该包下的所有类，这给开发者带来了一些不便。建议使用 Chrome、Firefox 等浏览器查看这份 API

文档，使用 IE 查看这份 API 文档会比较慢。这份 API 文档与所有 API 文档一样，都是开发者必须时常查阅的手册。

提示：

图 2.3 所示页面中 API Guides、Reference 两个标签页的内容是 Android 文档的最重要内容，本书内容将会基本覆盖 API Guides 的内容。不仅如此，Resource 标签页中也包含了 Android 的一些简单的入门示例，也是初学者学习 Android 应用开发不错的资料。

前面介绍 Android 应用结构时已经指出，Android 推荐使用 XML 布局文件来定义用户界面，而不是使用 Java 代码来开发用户界面，因此所有组件都提供了两种方式来控制组件的行为。

- 在 XML 布局文件中通过 XML 属性进行控制。
- 在 Java 程序代码中通过调用方法进行控制。

实际上不管使用哪种方式，它们控制 Android 用户界面行为的本质是完全一样的。大部分时候，控制 UI 组件的 XML 属性还有对应的方法。

对于 View 类而言，它是所有 UI 组件的基类，因此它包含的 XML 属性和方法是所有组件都可使用的。表 2.1 所示是 View 类常用的 XML 属性、相关方法及简要说明。

表 2.1 View 类的 XML 属性、相关方法及说明

XML 属性	相关方法	说明
android:alpha	setAlpha(float)	设置该组件的透明度
android:background	setBackgroundResource(int)	设置该组件的背景颜色
android:clickable	setClickable(boolean)	设置该组件是否可以激发单击事件
android:contentDescription	setContentDescription(CharSequence)	设置该组件的内容描述信息
android:drawingCacheQuality	setDrawingCacheQuality(int)	设置该组件所使用的绘制缓存的质量
android:elevation	setElevation(float)	设置该组件“浮”起来的高度，通过设置该属性可让该组件呈现 3D 效果
android:fadeScrollbars	setScrollbarFadingEnabled(boolean)	当不使用该组件的滚动条时，是否淡出显示滚动条
android:fadingEdge	setVerticalFadingEdgeEnabled(boolean)	设置滚动该组件时组件边界是否使用淡出效果
android:fadingEdgeLength	getVerticalFadingEdgeLength()	设置淡出边界的长度
android:focusable	setFocusable(boolean)	设置该组件是否可以得到焦点
android:focusableInTouchMode	setFocusableInTouchMode(boolean)	设置该组件在触摸模式下是否可以得到焦点
android:id	setId(int)	设置该组件的唯一标识。在 Java 代码中可通过 findViewById 来获取它
android:isScrollContainer	setScrollContainer(boolean)	设置该组件是否作为可滚动容器使用
android:keepScreenOn	setKeepScreenOn(boolean)	设置该组件是否会强制手机屏幕一直打开
android:longClickable	setLongClickable(boolean)	设置该组件是否可以响应长单击事件
android:minHeight	setMinimumHeight(int)	设置该组件的最小高度
android:minWidth	etMinimumWidth(int)	设置该组件的最小宽度
android:nextFocusDown	setNextFocusDownId(int)	设置焦点在该组件上，且单击向下键时获得焦点的组件 ID
android:nextFocusLeft	setNextFocusLeftId(int)	设置焦点在该组件上，且单击向左键时获得焦点的组件 ID

续表

XML 属性	相关方法	说明
android:nextFocusRight	setNextFocusRightId(int)	设置焦点在该组件上，且单击向右键时获得焦点的组件 ID
android:nextFocusUp	setNextFocusUpId(int)	设置焦点在该组件上，且单击向上键时获得焦点的组件 ID
android:onClick		为该组件的单击事件绑定监听器
android:padding	setPadding(int,int,int,int)	在组件的四边设置填充区域
android:paddingBottom	setPadding(int,int,int,int)	在组件的下边设置填充区域
android:paddingLeft	setPadding(int,int,int,int)	在组件的左边设置填充区域
android:paddingRight	setPadding(int,int,int,int)	在组件的右边设置填充区域
android:paddingTop	setPadding(int,int,int,int)	在组件的上边设置填充区域
android:rotation	setRotation(float)	设置该组件旋转的角度
android:rotationX	setRotationX(float)	设置该组件绕 *X* 轴旋转的角度
android:rotationY	setRotationY(float)	设置该组件绕 *Y* 轴旋转的角度
android:saveEnabled	setSaveEnabled(boolean)	如果设置为 false，那么当该组件被冻结时不会保存它的状态
android:scaleX	setScaleX(float)	设置该组件在水平方向的缩放比
android:scaleY	setScaleY(float)	设置该组件在垂直方向的缩放比
android:scrollX		该组件初始化后的水平滚动偏移
android:scrollY		该组件初始化后的垂直滚动偏移
android:scrollbarAlwaysDrawHorizontalTrack		设置该组件是否总是显示水平滚动条的轨道
android:scrollbarAlwaysDrawVerticalTrack		设置该组件是否总是显示垂直滚动条的轨道
android:scrollbarDefaultDelayBeforeFade	setScrollBarDefaultDelayBeforeFade(int)	设置滚动条在淡出隐藏之前延迟多少毫秒
android:scrollbarFadeDuration	setScrollBarFadeDuration(int)	设置滚动条淡出隐藏过程需要多少秒
android:scrollbarSize	setScrollBarSize(int)	设置垂直滚动条的宽度和水平滚动条的高度
android:scrollbarStyle	setScrollBarStyle(int)	设置滚动条的风格和位置。该属性支持如下属性值： insideOverlay insideInset outsideOverlay outsideInset
android:scrollbarThumbHorizontal		设置该组件的水平滚动条的滑块对应的 Drawable 对象
android:scrollbarThumbVertical		设置该组件的垂直滚动条的滑块对应的 Drawable 对象
android:scrollbarTrackHorizontal		设置该组件的水平滚动条的轨道对应的 Drawable 对象
android:scrollbarTrackVertical		设置该组件的垂直滚动条的轨道对应的 Drawable 对象
android:scrollbars		定义该组件滚动时显示几个滚动条。该属性支持如下属性值： ➢ none：不显示滚动条 ➢ horizontal：显示水平滚动条 ➢ vertical：显示垂直滚动条

续表

XML 属性	相关方法	说明
android:soundEffectsEnabled	setSoundEffectsEnabled(boolean)	设置该组件被单击时是否使用音效。
android:tag		为该组件设置一个字符串类型的 tag 值。接下来可通过 View 的 getTag()获取该字符串，或通过 findViewWithTag()查找该组件
android:transformPivotX	setPivotX(float)	设置该组件旋转时旋转中心的 *X* 坐标
android:transformPivotY	setPivotY(float)	设置该组件旋转时旋转中心的 *Y* 坐标
android:translationX	setTranslationX(float)	设置该组件在 *X* 方向上的位移
android:translationY	setTranslationY(float)	设置该组件在 *Y* 方向上的位移
android:translationZ	setTranslationZ(float)	设置该组件在 *Z* 方向（垂直屏幕方向）上的位移
android:visibility	setVisibility(int)	设置该组件是否可见

提示：

Drawable 是 Android 提供的一个抽象基类，它代表了“可以被绘制出来的某种东西”，Drawable 包括了大量子类，比如 BitmapDrawable 代表位图 Drawable、ColorDrawable 代表颜色 Drawable、ShapeDrawable 代表几何形状 Drawable。各种 Drawable 可用于定制 UI 组件的背景等外观。本书第 7 章会详细介绍各种 Drawable 的功能与用法。

注意：

上面表格中介绍 View 组件时列出的 setElevation(float)和 setTranslationZ(float)方法都是 Android 5.0 新增的 Material Design 中的功能，它们的作用也基本相似，它们都可用于设置该组件垂直屏幕“浮”起来的效果，通过该属性可让该组件呈现 3D 效果。当该组件垂直于屏幕“浮”起来之后，组件下的实时阴影就会自动显现。

ViewGroup 继承了 View 类，当然也可以当成普通 View 来使用，但 ViewGroup 主要还是当成容器类使用。但由于 ViewGroup 是一个抽象类，因此实际使用中通常总是使用 ViewGroup 的子类来作为容器，例如各种布局管理器。

ViewGroup 容器控制其子组件的分布依赖于 ViewGroup.LayoutParams、ViewGroup.MarginLayoutParams 两个内部类。这两个内部类中都提供了一些 XML 属性，ViewGroup 容器中的子组件可以指定这些 XML 属性。

表 2.2 显示了 ViewGroup.LayoutParams 所支持的两个 XML 属性。

表 2.2 ViewGroup.LayoutParams 支持的 XML 属性

XML 属性	说明
android:layout_height	指定该子组件的布局高度
android:layout_width	指定该子组件的布局宽度

android:layout_height、android:layout_width 两个属性支持如下 3 个属性值。

- fill_parent：指定子组件的高度、宽度与父容器组件的高度、宽度相同（实际上还要减去填充的空白距离）。

- match_parent：该属性值与 fill_parent 完全相同，而且从 Android 2.2 开始就推荐使用这个属性值来代替 fill_parent。
- wrap_content：指定子组件的大小恰好能包裹它的内容即可。

读者可能对布局高度与布局宽度感到疑惑：为组件指定了高度与宽度不就够了吗？为何还要设置布局高度与布局宽度呢？这是由 Android 的布局机制决定的，Android 组件的大小不仅受它实际的宽度、高度控制，还受它的布局高度与布局宽度控制。比如设置一个组件的宽度为 30pt，如果将它的布局宽度设为 match_parent，那么该组件的宽度将会被“拉宽”到占满它所在的父容器；如果将它的布局宽度设为 wrap_content，那么该组件的宽度才会是 30pt。

ViewGroup.MarginLayoutParams 用于控制子组件周围的页边距（Margin，也就是组件四周的留白），它支持的 XML 属性如表 2.3 所示。

表 2.3 ViewGroup.MarginLayoutParams 支持的 XML 属性

XML 属性	相关方法	说明
android:layout_marginBottom	setMargins(int,int,int,int)	指定该子组件下边的页边距
android:layout_marginLeft	setMargins(int,int,int,int)	指定该子组件左边的页边距
android:layout_marginRight	setMargins(int,int,int,int)	指定该子组件右边的页边距
android:layout_marginTop	setMargins(int,int,int,int)	指定该子组件上边的页边距

后面我们还会详细介绍 ViewGroup 各子类的用法，此处不再详述。

2.1.2 使用 XML 布局文件控制 UI 界面

Android 推荐使用 XML 布局文件来控制视图，这样不仅简单、明了，而且可以将应用的视图控制逻辑从 Java 代码中分离出来，放入 XML 文件中控制，从而更好地体现 MVC 原则。

当我们在 Android 应用的 app\src\main\res/layout 目录下定义一个主文件名任意的 XML 布局文件之后（R.java 会自动收录该布局资源），Java 代码可通过如下方法在 Activity 中显示该视图：

```
setContentView(R.layout.<资源文件名字>);
```

当在布局文件中添加多个 UI 组件时，都可以为该 UI 组件指定 android:id 属性，该属性的属性值代表该组件的唯一标识。接下来如果希望在 Java 代码中访问指定 UI 组件，则可通过如下代码来访问它：

```
findViewById(R.id.<android.id 属性值>);
```

一旦在程序中获得指定 UI 组件之后，接下来就可以通过代码来控制各 UI 组件的外观行为了，包括为 UI 组件绑定事件监听器等。

2.1.3 在代码中控制 UI 界面

虽然 Android 推荐使用 XML 布局文件来控制 UI 界面，但如果开发者愿意，Android 允许开发者像开发 Swing 应用一样，完全抛弃 XML 布局文件，完全在 Java 代码中控制 UI 界面。如果希望在代码中控制 UI 界面，那么所有的 UI 组件都将通过 new 关键字创建出来，然后以合适的方式“搭建”在一起即可。

实例：用编程的方式开发 UI 界面

下面将试图开发一个完全用代码控制 UI 界面的 Android 应用。由于该应用完全采用代码来控制 UI 界面，因此可以完全抛弃 XML 布局文件。下面是通过代码控制 UI 界面的代码。

程序清单：codes\02\2.1\CodeView\app\src\main\java\org\crazyit\ui\CodeViewActivity.java

```
public class CodeViewActivity extends Activity
{
    // 当第一次创建该 Activity 时回调该方法
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 创建一个线性布局管理器
        LinearLayout layout = new LinearLayout(this);
        // 设置该 Activity 显示 layout
        super.setContentView(layout);
        layout.setOrientation(LinearLayout.VERTICAL);
        // 创建一个 TextView
        final TextView show = new TextView(this);
        // 创建一个按钮
        Button bn = new Button(this);
        bn.setText(R.string.ok);
        bn.setLayoutParams(new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
        // 向 layout 容器中添加 TextView
        layout.addView(show);
        // 向 layout 容器中添加按钮
        layout.addView(bn);
        // 为按钮绑定一个事件监听器
        bn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                show.setText("Hello , Android , " + new java.util.Date());
            }
        });
    }
}
```

从上面程序的粗体字代码可以看出，该程序中所用到的 UI 组件都是通过 new 关键字创建出来的，然后程序使用 LinearLayout 容器来“盛装”这些 UI 组件，这样就组成了图形用户界面。

从上面的程序代码中可以看出，无论创建哪种 UI 组件，都需要传入一个 this 参数，这是由于创建 UI 组件时传入一个 Context 参数，Context 代表访问 Android 应用环境的全局信息的 API。让 UI 组件持有一个 Context 参数，可以让这些 UI 组件通过该 Context 参数来获取 Android 应用环境的全局信息。

Context 本身是一个抽象类，Android 应用的 Activity、Service 都继承了 Context，因此 Activity、Service 都可直接作为 Context 使用。

在模拟器中运行上面的程序，将可以看到如图 2.4 所示的界面。

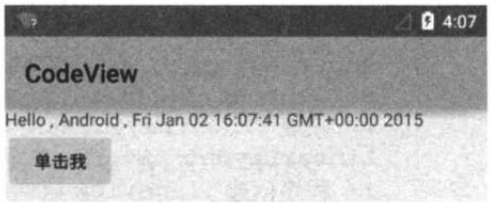


图 2.4 在代码中控制 UI 界面

从上面的程序代码中不难看出，完全在代码中控制 UI 界面不仅不利于高层次的解耦，而且由于通过 new 关键字来创建 UI 组件，需要调用方法来设置 UI 组件的行为，因此代码也显得十分臃肿；相反，如果通过 XML 布局文件来控制 UI 界面，开发者只要在 XML 布局文件中使用标签即可创建 UI 组件，而且只要配置简单的属性即可控制 UI 组件的行为，因此要简单得多。

虽然 Android 应用完全允许开发者像开发 Swing 应用一样在代码中控制 UI 界面，但这种方式不仅编程烦琐，而且不利于高层次的解耦，因此不推荐开发者使用这种方式。

2.1.4 使用 XML 布局文件和 Java 代码混合控制 UI 界面

前面已经提到，完全使用 Java 代码来控制 UI 界面不仅烦琐，而且不利于解耦；而完全利用 XML 布局文件来控制 UI 界面虽然方便、便捷，但难免有失灵活。因此有些时候，可能需要混合使用 XML 布局文件和代码来控制 UI 界面。

当混合使用 XML 布局文件和代码来控制 UI 界面时，习惯上把变化小、行为比较固定的组件放在 XML 布局文件中管理，而那些变化较多、行为控制比较复杂的组件则交给 Java 代码来管理。

实例：简单图片浏览器

例如下面的应用，我们先在布局文件中定义一个简单的线性布局容器，该布局文件的代码如下。

程序清单：codes\02\2.1\MixView\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个线性布局容器 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity"
    android:id="@+id/root"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</LinearLayout>
```

上面的布局文件只是定义了一个简单的线性布局容器。接下来我们会在程序中获取该线性布局容器，并往该容器中添加组件。下面是该实例的程序代码。

程序清单：codes\02\2.1\MixView\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义一个访问图片的数组
    int[] images = new int[] {
            R.drawable.java,
            R.drawable.javaee,
            R.drawable.swift,
            R.drawable.ajax,
            R.drawable.html, };
    int currentImg = 0;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // 获取 LinearLayout 布局容器
        LinearLayout main = (LinearLayout) findViewById(R.id.root);
        // 程序创建 ImageView 组件
        final ImageView image = new ImageView(this);
        // 将 ImageView 组件添加到 LinearLayout 布局容器中
        main.addView(image);
        // 初始化时显示第一张图片
        image.setImageResource(images[0]);
        image.setOnClickListener(new View.OnClickListener()
```

```
        {
            @Override
            public void onClick(View v)
            {
                // 改变 ImageView 里显示的图片
                image.setImageResource(images[++currentImg % images.length]);
            }
        });
    }
}
```

上面程序中第 1 行粗体字代码获取了该 Activity 所显示的 LinearLayout（线性布局容器），第 2 行和第 3 行粗体字代码用于创建一个 ImageView，并将该 ImageView 添加到 LinearLayout 容器中——其中 LinearLayout 布局容器通过 XML 布局文件管理，而 ImageView 组件则由 Java 代码管理。

除此之外，上面的程序还为 ImageView 组件添加了一个单击事件。当用户单击该组件时，ImageView 显示下一张图片。运行上面的程序，可以看到如图 2.5 所示的界面。

图 2.5 XML 布局文件和 Java 代码混合控制布局

2.1.5 开发自定义 View

前面已经提到，View 组件的作用类似于 Swing 编程中的 JPanel，它只是一个矩形的空白区域，View 组件没有任何内容。对于 Android 应用的其他 UI 组件来说，它们都继承了 View 组件，然后在 View 组件提供的空白区域上绘制外观。

基于 Android UI 组件的实现原理，开发者完全可以开发出项目定制的组件——当 Android 系统提供的 UI 组件不足以满足项目需要时，开发者可以通过继承 View 来派生自定义组件。

当开发者打算派生自己的 UI 组件时，首先定义一个继承 View 基类的子类，然后重写 View 类的一个或多个方法。通常可以被用户重写的方法如下。

- **构造器**：重写构造器是定制 View 的最基本方式，当 Java 代码创建一个 View 实例，或根据 XML 布局文件加载并构建界面时将需要调用该构造器。
- onFinishInflate()：这是一个回调方法，当应用从 XML 布局文件加载该组件并利用它来构建界面之后，该方法将会被回调。
- onMeasure(int, int)：调用该方法来检测 View 组件及其所包含的所有子组件的大小。
- onLayout(boolean, int, int, int, int)：当该组件需要分配其子组件的位置、大小时，该方法就会被回调。
- onSizeChanged(int, int, int, int)：当该组件的大小被改变时回调该方法。
- onDraw(Canvas)：当该组件将要绘制它的内容时回调该方法进行绘制。
- onKeyDown(int, KeyEvent)：当某个键被按下时触发该方法。
- onKeyUp(int, KeyEvent)：当松开某个键时触发该方法。
- onTrackballEvent(MotionEvent)：当发生轨迹球事件时触发该方法。
- onTouchEvent(MotionEvent)：当发生触摸屏事件时触发该方法。
- onFocusChanged (boolean gainFocus, int direction, Rect previouslyFocusedRect)：当该组件焦点发生改变时触发该方法。
- onWindowFocusChanged(boolean)：当包含该组件的窗口失去或得到焦点时触发该方法。

- onAttachedToWindow()：当把该组件放入某个窗口时触发该方法。
- onDetachedFromWindow()：当把该组件从某个窗口上分离时触发该方法。
- onWindowVisibilityChanged(int)：当包含该组件的窗口的可见性发生改变时触发该方法。

当需要开发自定义 View 时，开发者并不需要重写上面列出的所有方法，而是可以根据业务需要重写上面的部分方法。例如，下面的示例程序就只重写 onDraw(Canvas)方法。

实例：跟随手指的小球

为了实现一个跟随手指的小球，我们考虑开发自定义的 UI 组件，这个 UI 组件将会在指定位置绘制一个小球，这个位置可以动态改变。当用户通过手指在屏幕上拖动时，程序监听到这个手指动作，把手指动作的位置传入自定义 UI 组件，并通知该组件重绘即可。

下面是自定义组件的代码。

程序清单：codes\02\2.1\CustomView\app\src\main\java\org\crazyit\ui\DrawView.java

```
public class DrawView extends View
{
    public float currentX = 40;
    public float currentY = 50;
    // 定义并创建画笔
    Paint p = new Paint();
    public DrawView(Context context)
    {
        super(context);
    }
    public DrawView(Context context , AttributeSet set)
    {
        super(context,set);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 设置画笔的颜色
        p.setColor(Color.RED);
        // 绘制一个小圆（作为小球）
        canvas.drawCircle(currentX, currentY, 15, p);
    }
    // 为该组件的触碰事件重写事件处理方法
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 修改 currentX、currentY 两个属性
        currentX = event.getX();
        currentY = event.getY();
        // 通知当前组件重绘自己
        invalidate();
        // 返回 true 表明该处理方法已经处理该事件
        return true;
    }
}
```

上面的 DrawView 组件继承了 View 基类，并重写了 onDraw 方法——该方法负责在该组件的指定位置绘制一个小球。除此之外，该组件还重写了 onTouchEvent(MotionEvent event)方法，该方法用于处理该组件的触碰事件，当用户手指触碰该组件时将会激发该方法。当手指在

触摸屏上移动时，将会不断地触发触摸屏事件，事件监听器中负责触发事件的坐标将被传入 DrawView 组件，并通知该组件重绘——这样即可保证 DrawView 上小球跟随手指移动而移动。

有了这个自定义组件之后，接下来可以通过 Java 代码把该组件添加到指定容器中，这样就可以看到该组件的运行效果。下面是该应用的 Activity 类。

程序清单：codes\02\2.1\CustomView\app\src\main\java\org\crazyit\ui\CustomViewActivity.java

```
public class CustomViewActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取布局文件中的 LinearLayout 容器
        LinearLayout root = (LinearLayout) findViewById(R.id.root);
        // 创建 DrawView 组件
        final DrawView draw = new DrawView(this);
        // 设置自定义组件的最小宽度、高度
        draw.setMinimumWidth(300);
        draw.setMinimumHeight(500);
        root.addView(draw);
    }
}
```

上面的程序中先创建了自定义组件（DrawView）的实例，然后程序将该组件添加到 LinearLayout 容器中。

运行上面的程序，看到结果如图 2.6 所示。

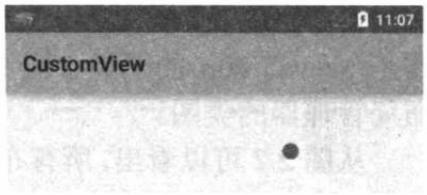


图 2.6 跟随手指的小球

该实例依然在 Java 代码中创建了 DrawView 组件的实例，并将它添加到 LinearLayout 容器中，实际上完全可以在 XML 布局文件中管理该组件，如果我们使用如下布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent "
    android:id="@+id/root">
<org.crazyit.ui.DrawView
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</LinearLayout>
```

上面的布局文件已经添加了自定义组件，因此 Java 代码中只要加载该界面布局文件即可，无须通过 Java 代码来添加该自定义组件。因此，Activity 的代码可以简化为如下形式：

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

2.2 第 1 组 UI 组件：布局管理器

Android 的界面组件比较多，如果不理顺它们内在的关系，孤立地学习、记忆这些 UI 组

件，不仅学习起来事倍功半，而且不利于掌握它们内在的关系。为了帮助读者更好地掌握Android界面组件的关系，本书将会把这些界面组件按照它们的关联分析，分为几组进行介绍。本节介绍的是第1组UI组件：以ViewGroup为基类派生的布局管理器。

为了更好地管理Android应用的用户界面里的各组件，Android提供了布局管理器。通过使用布局管理器，Android应用的图形用户界面具有良好的平台无关性。通常来说，推荐使用布局管理器来管理组件的分布、大小，而不是直接设置组件位置和大小。例如，通过如下代码定义了一个文本框（TextView）：

```
TextView hello = new TextView(this);
hello.setText("Hello Android");
```

为了让这个组件在不同的手机屏幕上都能运行良好——不同手机屏幕的分辨率、尺寸并不完全相同，如果让程序手动控制每个组件的大小、位置，则将给编程带来巨大的困难。为了解决这个问题，Android提供了布局管理器。布局管理器可以根据运行平台来调整组件的大小，程序员要做的，只是为容器选择合适的布局管理器。

与Swing界面编程不同的是，Android的布局管理器本身就是一个UI组件，所有的布局管理器都是ViewGroup的子类。图2.7显示了Android布局管理器的类图。

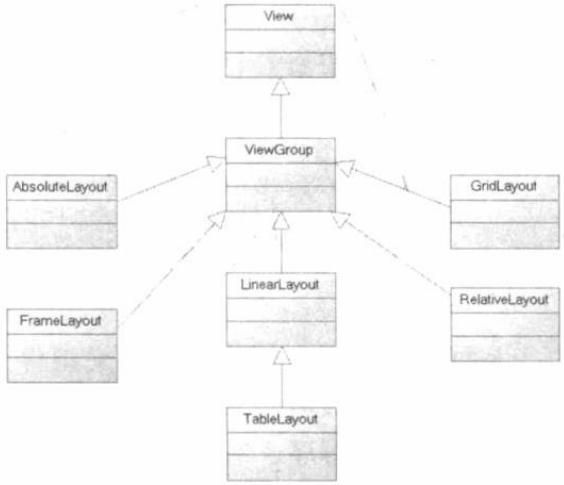


图2.7 Android布局管理器的类图

从图2.7可以看出，所有布局都可作为容器类使用，因此可以调用多个重载的addView()向布局管理器中添加组件。实际上，我们完全可以用一个布局管理器嵌套到其他布局管理器中——因为布局管理器也继承了View，也可以作为普通UI组件使用。

2.2.1 线性布局

线性布局由LinearLayout类来代表，线性布局有点像Swing编程里的Box，它们都会将容器里的组件一个挨着一个地排列起来。LinearLayout可以控制各组件横向排列（通过设置android:orientation属性控制），也可控制各组件纵向排列。

Android的线性布局不会换行，当组件一个挨着一个地排列到头之后，剩下的组件将不会被显示出来。

表2.4显示了LinearLayout支持的常用XML属性及相关方法的说明。

表2.4 LinearLayout的常用XML属性及相关方法

XML属性	相关方法	说明
android:baselineAligned	setBaselineAligned(boolean)	该属性设为false，将会阻止该布局管理器与它的子元素的基线对齐
android:divider	setDividerDrawable(Drawable)	设置垂直布局时两个按钮之间的分隔条
android:gravity	setGravity(int)	设置布局管理器内组件的对齐方式。该属性支持top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical、clip_horizontal几个属性值。也可以同时指定多种对齐方式的组合，例如left|center_vertical代表出现在屏幕左边，而且垂直居中

续表

XML 属性	相关方法	说明
android:measureWithLargestChild	setMeasureWithLargestChildEnabled(boolean)	当该属性设为 true 时，所有带权重的子元素都会具有最大子元素的最小尺寸。
android:orientation	setOrientation(int)	设置布局管理器内组件的排列方式，可以设置为 horizontal（水平排列）、vertical（垂直排列，默认值）两个值的其中之一

LinearLayout 包含的所有子元素都受 LinearLayout.LayoutParams 控制，因此 LinearLayout 包含的子元素可以额外指定如表 2.5 所示的属性。

表 2.5 LineLayout 子元素支持的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:layout_gravity		指定该子元素在 LinearLayout 中的对齐方式
android:layout_weight		指定该子元素在 LinearLayout 中所占的权重

提示：

基本上很多布局管理器都提供了相应的 LayoutParams 内部类，该内部类用于控制它们的子元素支持指定 android:layout_gravity 属性，该属性设置该子元素在父容器中的对齐方式。与 android:layout_gravity 相似的属性还有 android:gravity 属性（一般容器才支持指定该属性），android:gravity 属性用于控制它所包含的子元素的对齐方式。

例如，定义如下 XML 布局管理器。

程序清单：codes\02\2.2\LinearLayoutTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="bottom|center_horizontal">
    <Button
        android:id="@+id/bn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn1"/>
    <Button
        android:id="@+id/bn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn2"/>
    <Button
        android:id="@+id/bn3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn3"/>
    <Button
        android:id="@+id/bn4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn4"/>
    <Button
        android:id="@+id/bn5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="@string/bn5"/>
</LinearLayout>
```

上面的界面布局非常简单，它只是定义了一个简单的线性布局，并在线性布局中定义了 5 个按钮。定义线性布局时使用粗体字代码指定了垂直排列所有组件，而且所有组件对齐到容器底部并且水平居中。运行上面的程序，使用 Activity 显示界面布局，将看到如图 2.8 所示的界面。

如果将上面布局文件中的 android:gravity="bottom|center_horizontal"改为 android:gravity="right|center_vertical"——也就是所有组件水平右对齐、垂直居中，再次使用 Activity 显示该界面布局，将看到如图 2.9 所示的界面。

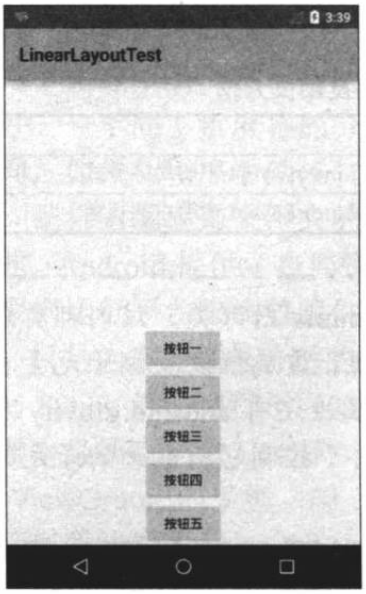


图 2.8　垂直的线性布局，底部、居中对齐

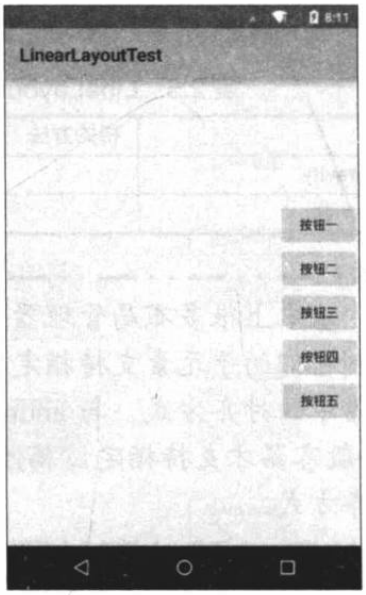


图 2.9　垂直的线性布局，水平居右、垂直居中对齐

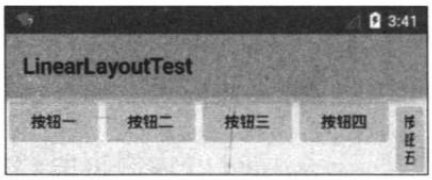


图 2.10　水平的线性布局，顶端对齐

如果我们将上面的线性布局的方向改为水平，也就是设置 android:orientation="horizontal"，并设置 gravity="top"，再次使用 Activity 来显示该界面布局，将看到如图 2.10 所示的界面。

从图 2.10 所示的运行结果可以看出，当采用线性布局来管理 5 个按钮组件时，如果这 5 个按钮组件无法在一行中同时显示出来，但 LinearLayout 不会换行显示多余的组件，因此后面可能有些按钮会显示不出来。

➤➤ 2.2.2　表格布局

表格布局由 TableLayout 所代表，TableLayout 继承了 LinearLayout，因此它的本质依然是线性布局管理器。表格布局采用行、列的形式来管理 UI 组件，TableLayout 并不需要明确地声明包含多少行、多少列，而是通过添加 TableRow、其他组件来控制表格的行数和列数。

每次向 TableLayout 中添加一个 TableRow，该 TableRow 就是一个表格行，TableRow 也是容器，因此它也可以不断地添加其他组件，每添加一个子组件该表格就增加一列。

如果直接向 TableLayout 中添加组件，那么这个组件将直接占用一行。

在表格布局中，列的宽度由该列中最宽的那个单元格决定，整个表格布局的宽度则取决于父容器的宽度（默认总是占满父容器本身）。

在表格布局管理器中，可以为单元格设置如下 3 种行为方式。

➢ Shrinkable：如果某个列被设为 Shrinkable，那么该列的所有单元格的宽度可以被收

缩，以保证该表格能适应父容器的宽度。

- Stretchable：如果某个列被设为 Stretchable，那么该列的所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间。
- Collapsed：如果某个列被设为 Collapsed，那么该列的所有单元格会被隐藏。

TableLayout 继承了 LinearLayout，因此它完全可以支持 LinearLayout 所支持的全部 XML 属性。除此之外，TableLayout 还支持如表 2.6 所示的 XML 属性。

表 2.6 TableLayout 的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:collapseColumns	setColumnCollapsed(int,boolean)	设置需要被隐藏的列的列序号。多个列序号之间用逗号隔开
android:shrinkColumns	setShrinkAllColumns(boolean)	设置允许被收缩的列的列序号。多个列序号之间用逗号隔开
android:stretchColumns	setStretchAllColumns(boolean)	设置允许被拉伸的列的列序号。多个列序号之间用逗号隔开

实例：丰富的表格布局

下面的程序示范了如何使用 TableLayout 来管理组件的布局。下面是界面布局所使用的布局文件。

程序清单：codes\02\2.2\TableLayoutTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 定义第 1 个表格布局，指定第 2 列允许收缩，第 3 列允许拉伸 -->
<TableLayout android:id="@+id/TableLayout01"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:shrinkColumns="1"
    android:stretchColumns="2">
...<!-- 表格内容接下来详细讲解 -->
</TableLayout>
<!-- 定义第 2 个表格布局，指定第 2 列隐藏-->
<TableLayout android:id="@+id/TableLayout02"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:collapseColumns="1">
...<!-- 表格内容接下来详细讲解 -->
</TableLayout>
<!-- 定义第 3 个表格布局，指定第 2 列和第 3 列可以被拉伸-->
<TableLayout android:id="@+id/TableLayout03"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1,2">
...<!-- 表格内容接下来详细讲解 -->
</TableLayout>
</LinearLayout>
```

上面页面中定义了 3 个 TableLayout，3 个 TableLayout 中粗体字代码指定了它们对各列的控制行为。

- 第 1 个 TableLayout，指定第 2 列允许收缩，第 3 列允许拉伸。
- 第 2 个 TableLayout，指定第 2 列被隐藏。
- 第 3 个 TableLayout，指定第 2 列和第 3 列允许拉伸。

接下来为布局中第 1 个 TableLayout 添加两行，第 1 行不使用 TableRow，直接添加一个

Button，那么该 Button 自己将占用整行。第 2 行先添加一个 TableRow，并为 TableRow 添加 3 个 Button，那说明该表格将包含 3 列。第 1 个表格的界面布局代码如下。

程序清单：codes\02\2.2\TableLayoutTest\app\src\main\res\layout\main.xml

```
<!-- 直接添加按钮，它自己会占一行 -->
<Button android:id="@+id/ok1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="独自一行的按钮"/>
<!-- 添加一个表格行 -->
<TableRow>
<!-- 为该表格行添加 3 个按钮 -->
<Button android:id="@+id/ok2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="普通按钮"/>
<Button android:id="@+id/ok3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="收缩的按钮"/>
<Button android:id="@+id/ok4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="拉伸的按钮"/>
</TableRow>
```

注意：

在上面的界面布局文件中我们直接把按钮上的文本写在布局文件中，这不是一种好的做法，因为 Android 推荐将这些字符串集中放到 XML 文件中管理。但此处为了编程简单，所以直接在 XML 布局文件中给出了按钮文本的字符串。

接下来为布局中第 2 个 TableLayout 添加两行，第 1 行不使用 TableRow，直接添加一个 Button，那么该 Button 自己将占用整行。第 2 行先添加一个 TableRow，并为 TableRow 添加 3 个 Button，那说明该表格将包含 3 列。第 2 个表格内容与第 1 个表格内容基本相似，但由于我们为第 2 个表格指定了 android:collapseColumns="1"，这意味着第 2 行中间的按钮将会被隐藏。

再接下来为布局中第 3 个 TableLayout 添加 3 行，第 1 行不使用 TableRow，直接添加一个 Button，那么该 Button 自己将占用整行。第 2 行先添加一个 TableRow，并为 TableRow 添加 3 个 Button，那说明该表格将包含 3 列。第 3 行也先添加一个 TableRow，并为 TableRow 添加两个按钮，这意味着表格的第 3 行只有两个单元格，第 3 个单元格为空。

第 3 个表格布局管理器的内容如下。

程序清单：codes\02\2.2\TableLayoutTest\app\src\main\res\layout\main.xml

```
<!-- 直接添加按钮，它自己会占一行 -->
<Button android:id="@+id/ok9"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="独自一行的按钮"/>
<!--定义一个表格行-->
<TableRow>
<!-- 为该表格行添加 3 个按钮 -->
<Button android:id="@+id/ok10"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:text="普通按钮"/>
    <Button android:id="@+id/ok11"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拉伸的按钮"/>
    <Button android:id="@+id/ok12"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拉伸的按钮"/>
    </TableRow>
    <!--定义一个表格行-->
    <TableRow>
    <!-- 为该表格行添加两个按钮 -->
    <Button android:id="@+id/ok13"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="普通按钮"/>
    <Button android:id="@+id/ok14"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拉伸的按钮"/>
    </TableRow>
```

使用 Activity 来显示上面的界面布局，将会看到如图 2.11 所示的界面。

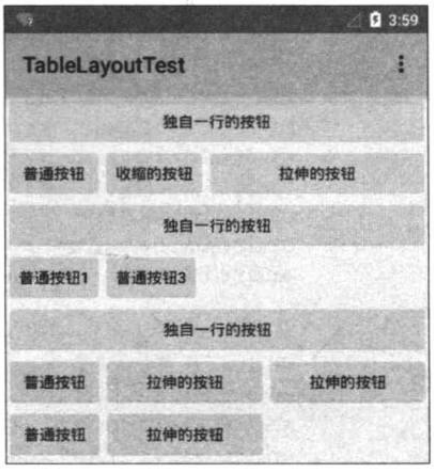


图 2.11 表格布局

2.2.3 帧布局

帧布局由 FrameLayout 所代表，FrameLayout 直接继承了 ViewGroup 组件。

帧布局容器为每个加入其中的组件创建一个空白的区域（称为一帧），每个子组件占据一帧，这些帧都会根据 gravity 属性执行自动对齐。帧布局的效果有点类似于 AWT 编程的 CardLayout，都是把组件一个个地叠加在一起。与 CardLayout 的区别在于，CardLayout 可以将下面的 Card 移上来，但 FrameLayout 则没有提供相应的方法。

表 2.7 显示了 FrameLayout 常用的 XML 属性及相关方法说明。

表 2.7 FrameLayout 的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:foreground	setForeground(Drawable)	设置该帧布局容器的前景图像
android:foregroundGravity	setForegroundGravity(int)	定义绘制前景图像的 gravity 属性

FrameLayout 包含的子元素也受 FrameLayout.LayoutParams 控制，因此它所包含的子元素也可指定 android:layout_gravity 属性，该属性控制该子元素在 FrameLayout 中的对齐方式。

下面示范了帧布局的用法，可以看到 6 个 TextView 叠加在一起，上面的 TextView 遮住下面的 TextView。下面是使用帧布局的页面定义代码。

程序清单：codes\02\2.2\FrameLayoutTest\app\src\main\res\layout\main.xml

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

```
    <!-- 依次定义 6 个 TextView，先定义的 TextView 位于底层，
        后定义的 TextView 位于上层 -->
    <TextView
        android:id="@+id/view01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="320pt"
        android:height="320pt"
        android:background="#f00"/>
    <TextView
        android:id="@+id/view02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="280pt"
        android:height="280pt"
        android:background="#0f0"/>
    <TextView
        android:id="@+id/view03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="240pt"
        android:height="240pt"
        android:background="#00f"/>
    <TextView
        android:id="@+id/view04"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="200pt"
        android:height="200pt"
        android:background="#ff0"/>
    <TextView
        android:id="@+id/view05"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="160pt"
        android:height="160pt"
        android:background="#f0f"/>
    <TextView
        android:id="@+id/view06"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="120pt"
        android:height="120pt"
        android:background="#0ff"/>
</FrameLayout>
```

上面的界面布局定义使用 FrameLayout 布局，并向该布局容器中添加了 6 个 TextView，这 6 个 TextView 的高度、宽度则逐渐减少——这样可以保证最先添加的 TextView 不会被完全遮挡；而且我们设置了 6 个 TextView 的背景色渐变。

使用 Activity 显示上面的界面布局，将看到如图 2.12 所示的效果。

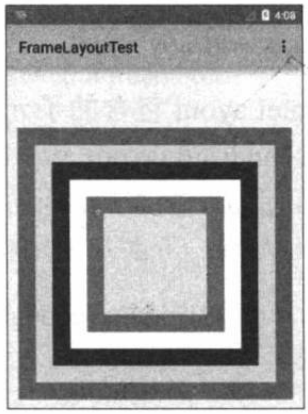


图 2.12　帧布局

实例：霓虹灯效果

如果考虑轮换改变上面的帧布局中 6 个 TextView 的背景色，就会看到上面的颜色渐变条不断地变换，就像大街上的霓虹灯一样。下面的程序还是使用上面的 FrameLayout 布局管理器，只是程序启动了一个线程来控制周期性地改变这 6 个 TextView 的背景色。下面是该主程序的代码。

程序清单：codes\02\2.2\FrameLayoutTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    private int currentColor = 0;
    // 定义一个颜色数组
    final int[] colors = new int[] {
        R.color.color1,
        R.color.color2,
        R.color.color3,
        R.color.color4,
        R.color.color5,
        R.color.color6
    };
    final int[] names = new int[] {
        R.id.view01,
        R.id.view02,
        R.id.view03,
        R.id.view04,
        R.id.view05,
        R.id.view06 };
    TextView[] views = new TextView[names.length];
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            // 表明消息来自本程序所发送的
            if (msg.what == 0x123)
            {
                for (int i = 0; i < names.length; i++)
                {
                    views[i].setBackgroundResource(colors[(i
                        + currentColor) % names.length]);
                }
                currentColor++;
            }
            super.handleMessage(msg);
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        for (int i = 0; i < names.length; i++)
        {
            views[i] = (TextView) findViewById(names[i]);
        }
        // 定义一个线程周期性地改变 currentColor 变量值
        new Timer().schedule(new TimerTask()
        {
            @Override
            public void run()
            {
```

```
                // 发送一条空消息通知系统改变 6 个 TextView 组件的背景色
                handler.sendEmptyMessage(0x123);
            }
        }, 0, 200);
    }
}
```

上面程序中的粗体字代码定义了一个每 0.2 秒执行一次的任务，该任务仅仅向 Handler 发送一条消息，通知它更新 6 个 TextView 的背景色。

可能会有读者提出疑问：为何不直接在 run()方法里直接更新 6 个 TextView 的背景色呢？这是因为 Android 的 View 和 UI 组件不是线程安全的，所以 Android 不允许开发者启动线程访问用户界面的 UI 组件。因此，程序中额外定义了一个 Handler 来处理 TextView 背景色的更新。

注意：

上面的程序中直接使用了 R.color.color1、R.color.color2、R.color.color3 等整型常量来代表颜色，这也得益于 Android 的资源访问支持，本书后面会有关于颜色资源的详细介绍。

简单地说，上面的程序通过任务调度控制了每间隔 0.2 秒轮换更新一次 6 个 TextView 的背景色，这样看上去就像大街上的霓虹灯了。

2.2.4 相对布局

相对布局由 RelativeLayout 所代表，相对布局容器内子组件的位置总是相对兄弟组件、父容器来决定的，因此这种布局方式被称为相对布局。

如果 A 组件的位置是由 B 组件的位置来决定的，Android 要求先定义 B 组件，再定义 A 组件。

RelativeLayout 可支持如表 2.8 所示的两个 XML 属性。

表 2.8　RelativeLayout 的 XML 属性及相关方法说明

XML 属性	相关方法	说明
android:gravity	setGravity(int)	设置该布局容器内各子组件的对齐方式
android:ignoreGravity	setIgnoreGravity(int)	设置哪个组件不受 gravity 属性的影响

为了控制该布局容器中各子组件的布局分布，RelativeLayout 提供了一个内部类：RelativeLayout.LayoutParams，该类提供了大量的 XML 属性来控制 RelativeLayout 布局容器中子组件的布局分布。

RelativeLayout.LayoutParams 里只能设为 true、false 的 XML 属性如表 2.9 所示。

表 2.9　RelativeLayout.LayoutParams 里只能设为 boolean 值的属性

XML 属性	说明
android:layout_centerHorizontal	控制该子组件是否位于布局容器的水平居中
android:layout_centerVertical	控制该子组件是否位于布局容器的垂直居中
android:layout_centerInParent	控制该子组件是否位于布局容器的中央位置
android:layout_alignParentBottom	控制该子组件是否与布局容器底端对齐
android:layout_alignParentLeft	控制该子组件是否与布局容器左边对齐
android:layout_alignParentRight	控制该子组件是否与布局容器右边对齐
android:layout_alignParentTop	控制该子组件是否与布局容器顶端对齐

RelativeLayout.LayoutParams 里属性值为其他 UI 组件 ID 的 XML 属性如表 2.10 所示。

表 2.10 RelativeLayout.LayoutParams 里只能设为其他 UI 组件 ID 的属性

XML 属性	说明
android:layout_toRightOf	控制该子组件位于给出 ID 组件的右侧
android:layout_toLeftOf	控制该子组件位于给出 ID 组件的左侧
android:layout_above	控制该子组件位于给出 ID 组件的上方
android:layout_below	控制该子组件位于给出 ID 组件的下方
android:layout_alignTop	控制该子组件与给出 ID 组件的上边界对齐
android:layout_alignBottom	控制该子组件与给出 ID 组件的下边界对齐
android:layout_alignLeft	控制该子组件与给出 ID 组件的左边界对齐
android:layout_alignRight	控制该子组件与给出 ID 组件的右边界对齐

除此之外，RelativeLayout.LayoutParams 还继承了 android.view.ViewGroup.MarginLayout Params，因此 RelativeLayout 布局容器中每个子组件也可指定 android.view.ViewGroup.MarginLayoutParams 所支持的各 XML 属性。

下面以一个实例来介绍相对布局的用法。

实例：梅花布局效果

相对布局容器中的子组件总是相对其他组件来决定分布位置的，可以考虑先把一个组件放在相对布局容器的中间，然后以该组件为中心，将其他组件分布在该组件的四周，这样就可以形成“梅花”布局效果。

下面是“梅花”布局效果的界面布局文件。

程序清单：codes\02\2.2\RelativeLayoutTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 定义该组件位于父容器中间 -->
<TextView
    android:id="@+id/view01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/leaf"
    android:layout_centerInParent="true"/>
<!-- 定义该组件位于 view01 组件的上方 -->
<TextView
    android:id="@+id/view02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/leaf"
    android:layout_above="@id/view01"
    android:layout_alignLeft="@id/view01"/>
<!-- 定义该组件位于 view01 组件的下方 -->
<TextView
    android:id="@+id/view03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/leaf"
    android:layout_below="@id/view01"
    android:layout_alignLeft="@id/view01"/>
<!-- 定义该组件位于 view01 组件的左边 -->
<TextView
    android:id="@+id/view04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:background="@drawable/leaf"
        android:layout_toLeftOf="@id/view01"
        android:layout_alignTop="@id/view01"/>
    <!-- 定义该组件位于view01组件的右边 -->
    <TextView
        android:id="@+id/view05"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/leaf"
        android:layout_toRightOf="@id/view01"
        android:layout_alignTop="@id/view01"/>
    </RelativeLayout>
```

上面程序中的第一行粗体字代码控制该组件位于父容器的中央，接下来定义的 4 个组件依次环绕在第一个组件四周。使用 Activity 来显示上面的布局文件，可以看到如图 2.13 所示的效果。

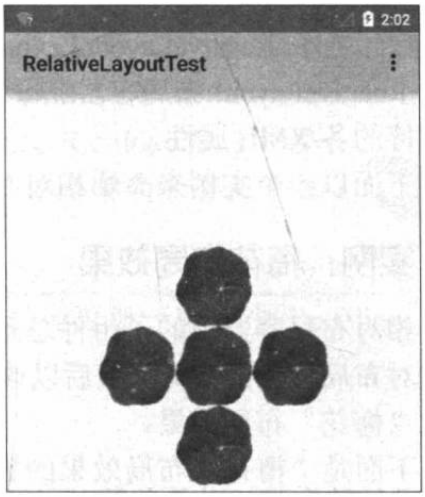


图 2.13　相对布局实现的“梅花”效果

提示：

这里例子中使用了一张图片，这也是利用了 Android 的图片资源访问策略，本书后面还会集中介绍图片资源访问。

2.2.5　网格布局

网格布局由 GridLayout 所代表，它是 Android 4.0 新增的布局管理器，因此需要在 Android 4.0 之后的版本中才能使用该布局管理器。如果希望在更早的 Android 平台上使用该布局管理器，则需要导入相应的支撑库。

GridLayout 的作用类似于 HTML 中的 table 标签，它把整个容器划分成 rows×columns 个网格，每个网格可以放置一个组件。除此之外，也可以设置一个组件横跨多少列、一个组件纵跨多少行。

GridLayout 提供了 setRowCount(int)和 setColumnCount(int)方法来控制该网格的行数量和列数量。

表 2.11 显示了 GridLayout 常用的 XML 属性及相关方法。

表 2.11　GridLayout 的 XML 属性及相关方法

XML 属性	相关方法	说明
android:alignmentMode	setAlignmentMode(int)	设置该布局管理器采用的对齐模式
android:columnCount	setColumnCount(int)	设置该网格的列数量
android:columnOrderPreserved	setColumnOrderPreserved(boolean)	设置该网格容器是否保留列序号
android:rowCount	setRowCount(int)	设置该网格的行数量
android:rowOrderPreserved	setRowOrderPreserved(boolean)	设置该网格容器是否保留行序号
android:useDefaultMargins	setUseDefaultMargins(boolean)	设置该布局管理器是否使用默认的页边距

为了控制 GridLayout 布局容器中各子组件的布局分布，GridLayout 提供了一个内部类：GridLayout.LayoutParams，该类提供了大量的 XML 属性来控制 GridLayout 布局容器中子组件的布局分布。

表 2.12 显示了 GridLayout.LayoutParams 常用的 XML 属性及相关方法。

表 2.12 GridLayout.LayoutParams 的 XML 属性及相关方法

XML 属性	相关方法	说明
android:layout_column		设置该子组件在 GridLayout 的第几列
android:layout_columnSpan		设置该子组件在 GridLayout 横向上跨几列
android:layout_gravity	setGravity(int)	设置该子组件采用何种方式占据该网格的空间
android:layout_row		设置该子组件在 GridLayout 的第几行
android:layout_rowSpan		设置该子组件在 GridLayout 纵向上跨几行

下面将会通过一个实例来示范 GridLayout 的功能和用法。

实例：计算器界面

为了实现如图 2.14 所示的计算器界面，可以考虑将该界面分解成一个 6×4 的网格，其中第一个文本框横跨 4 列，第二个按钮横跨 4 列，后面每个按钮各占一格。

为了实现该界面，考虑按如下步骤来操作。

① 在布局管理器中定义一个 GridLayout，并在该 GridLayout 中依次定义文本框、按钮，该文本框、按钮各横跨 4 列。

② 在 Java 代码中循环 16 次，依次添加 16 个按钮。

下面先定义界面布局文件。

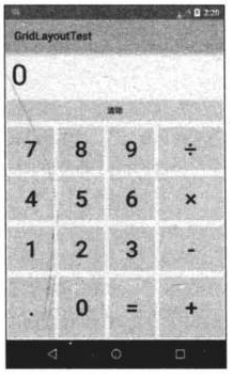


图 2.14 计算器界面

程序清单：codes\02\2.2\GridLayoutTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="6"
    android:columnCount="4"
    android:id="@+id/root">
<!-- 定义一个横跨 4 列的文本框，
并设置该文本框的前景色、背景色等属性  -->
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_columnSpan="4"
    android:textSize="50sp"
    android:layout_marginLeft="2pt"
    android:layout_marginRight="2pt"
    android:padding="3pt"
    android:layout_gravity="right"
    android:background="#eee"
    android:textColor="#000"
    android:text="0"/>
<!-- 定义一个横跨 4 列的按钮 -->
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_columnSpan="4"
    android:text="清除"/>
</GridLayout>
```

上面的界面布局文件中定义了一个 6×4 的 GridLayout，并在该布局管理器中添加了两个子组件，其中第一个子组件横跨 4 列，第二个子组件也横跨 4 列。

接下来使用如下 Java 代码采用循环控制添加 16 个按钮。

程序清单：codes\02\2.2\GridLayoutTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    GridLayout gridLayout;
    // 定义 16 个按钮的文本
    String[] chars = new String[]
        {
            "7" , "8" , "9" , "÷",
            "4" , "5" , "6" , "×",
            "1" , "2" , "3" , "-",
            "." , "0" , "=" , "+"
        };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        gridLayout = (GridLayout) findViewById(R.id.root);
        for(int i = 0 ; i < chars.length ; i++)
        {
            Button bn = new Button(this);
            bn.setText(chars[i]);
            // 设置该按钮的字号大小
            bn.setTextSize(40);
            // 设置按钮四周的空白区域
            bn.setPadding(5 , 35 , 5 , 35);
            // 指定该组件所在的行
            GridLayout.Spec rowSpec = GridLayout.spec(i / 4 + 2);
            // 指定该组件所在的列
            GridLayout.Spec columnSpec = GridLayout.spec(i % 4);
            GridLayout.LayoutParams params = new GridLayout.LayoutParams(
                    rowSpec , columnSpec);
            // 指定该组件占满父容器
            params.setGravity(Gravity.FILL);
            gridLayout.addView(bn , params);
        }
    }
}
```

上面的粗体字代码采用循环向 GridLayout 中添加了 16 个按钮，添加 16 个按钮时指定了每个按钮所在的行号、列号，并指定这些按钮将会自动填充单元格的所有空间——这样避免单元格中的大量空白。运行该程序，将可以看到如图 2.14 所示的界面。

2.2.6 绝对布局

绝对布局由 AbsoluteLayout 所代表。绝对布局就像 Java AWT 编程中的空布局，就是 Android 不提供任何布局控制，而是由开发人员自己通过 *X* 坐标、*Y* 坐标来控制组件的位置。当使用 AbsoluteLayout 作为布局容器时，布局容器不再管理子组件的位置、大小——这些都需要开发人员自己控制。

注意：

大部分时候，使用绝对布局都不是一个好思路，因为运行 Android 应用的手机往往千差万别，因此屏幕大小、分辨率都可能存在较大差异，使用绝对布局会很难兼顾不同屏幕大小、分辨率的问题。因此，AbsoluteLayout 布局管理器已经过时。

使用绝对布局时，每个子组件都可指定如下两个 XML 属性。

- layout_x：指定该子组件的 *X* 坐标。
- layout_y：指定该子组件的 *Y* 坐标。

实例：登录界面

下面介绍一个使用绝对布局开发的登录界面实例，这个登录界面中的所有组件都通过“绝对定位”的方式来指定位置。下面是该登录界面的界面布局文件。

程序清单：codes\02\2.2\AbsoluteLayoutTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 定义一个文本框，使用绝对定位 -->
<TextView
    android:layout_x="20dp"
    android:layout_y="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="用户名："/>
<!-- 定义一个文本编辑框，使用绝对定位 -->
<EditText
    android:layout_x="80dp"
    android:layout_y="15dp"
    android:layout_width="wrap_content"
    android:width="200px"
    android:layout_height="wrap_content" />
<!-- 定义一个文本框，使用绝对定位 -->
<TextView
    android:layout_x="20dp"
    android:layout_y="80dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="密　码："/>
<!-- 定义一个文本编辑框，使用绝对定位 -->
<EditText
    android:layout_x="80dp"
    android:layout_y="75dp"
    android:layout_width="wrap_content"
    android:width="200px"
    android:layout_height="wrap_content"
    android:password="true"/>
<!-- 定义一个按钮，使用绝对定位 -->
<Button
    android:layout_x="130dp"
    android:layout_y="135dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="登　录"/>
</AbsoluteLayout>
```

上面的绝对布局容器中的每个子组件都指定了 layout_x、layout_y 两个定位属性，这样才控制了每个子组件在容器中的出现位置。使用 Activity 显示上面的页面，将看到如图 2.15 所示的界面。

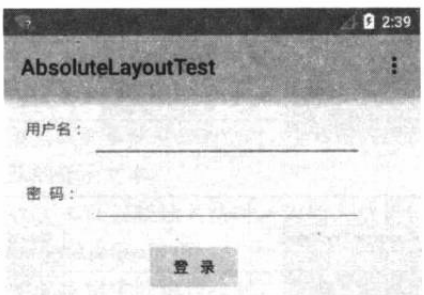


图 2.15 绝对布局实现的登录界面

不要以为笔者一下子就通过绝对布局做出了图 2.15 所示的登录界面，实际上这个登录界面是不断调整各组

件的位置，经过多次尝试之后得到的结果。当使用绝对布局来控制子组件布局时，编程要烦琐得多，而且在不同屏幕上的显示效果差异也很大。

上面的界面布局中指定各组件的 android:layout_x、android:layout_y 属性时指定了形如 20dp 这样的属性值，这是一个距离值。Android 中一般支持如下常用的距离单位。

- px（像素）：每个 px 对应屏幕上的一个点。
- dip 或 dp（device independent pixels，设备独立像素）：一种基于屏幕密度的抽象单位。在每英寸 160 点的显示器上，1dip = 1px。但随着屏幕密度的改变，dip 与 px 的换算会发生改变。
- sp（scaled pixels，比例像素）：主要处理字体的大小，可以根据用户的字体大小首选项进行缩放。
- in（英寸）：标准长度单位。
- mm（毫米）：标准长度单位。
- pt（磅）：标准长度单位，1/72 英寸。

2.3 第 2 组 UI 组件：TextView 及其子类

前面介绍了 Android 界面编程的一些基础知识，接下来将要介绍的是 Android 基本界面组件。“九层之台，起于垒土”——无论看上去多么美观的 UI 界面，开始都是先创建容器（ViewGroup 的实例），然后不断地向容器中添加界面组件，最后形成一个美观的 UI 界面。掌握这些基本用户界面组件是学好 Android 编程的基础。

2.3.1 文本框（TextView）与编辑框（EditText）的功能和用法

TextView 直接继承了 View，它还是 EditText、Button 两个 UI 组件类的父类。TextView 的作用就是在界面上显示文本——从这个意义上来看，它有点类似于 Swing 编程中的 JLabel，不过它比 JLabel 功能更强大。

从功能上来看，TextView 其实就是一个文本编辑器，只是 Android 关闭了它的文字编辑功能。如果开发者想要定义一个可以编辑内容的文本框，则可以使用它的子类：EditText，EditText 允许用户编辑文本框中的内容。

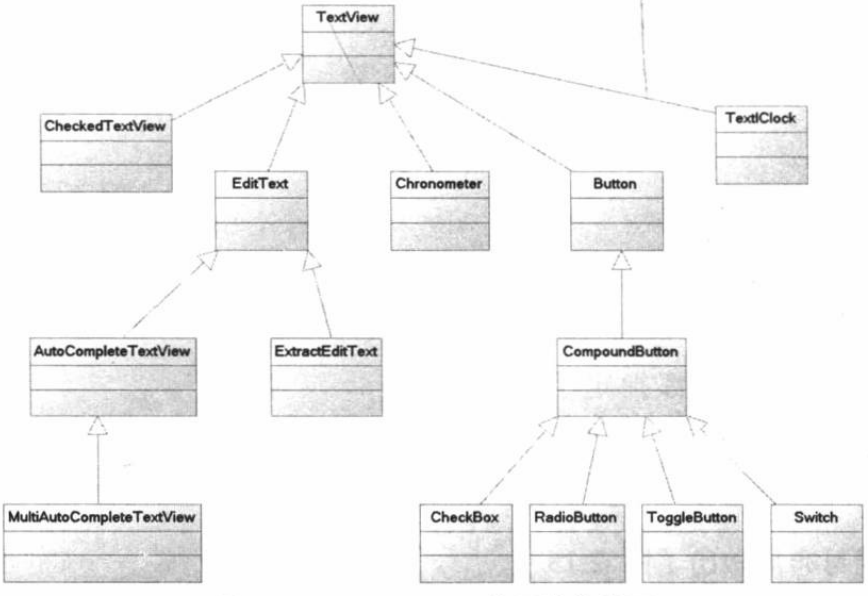


图 2.16 TextView 及其子类的类图

TextView 还派生了一个 CheckedTextView，CheckedTextView 增加了一个 checked 状态，开发者可通过 setChecked(boolean)和 isChecked()方法来改变、访问该组件的 checked 状态。除此之外，该组件还可通过 setCheckMarkDrawable()方法来设置它的勾选图标。

不仅如此，TextView 还派生出了 Button 类，TextView 及其子类的类图如图 2.16 所示。

TextView 和 EditText 具有很多相似之处，它们之间的最大区

别在于 TextView 不允许用户编辑文本内容，而 EditText 则允许用户编辑文本内容。

TextView 提供了大量的 XML 属性，这些 XML 属性大部分不仅可适用于 TextView，而且可适用于它的子类（EditText、Button 等）。表 2.13 显示了 TextView 支持的 XML 属性及相关方法。

表 2.13 TextView 支持的 XML 属性及相关方法

XML 属性	相关方法	说明
android:autoLink	setAutoLinkMask(int)	是否将符合指定格式的文本转换为可单击的超链接形式
android:autoText	setKeyListener(KeyListener)	控制是否将 URL、E-mail 地址等连接自动转换为可单击的链接
android:capitalize	setKeyListener(KeyListener)	控制是否将用户输入的文本转换为大写字母。该属性支持如下属性值： ➢ none：不转换 ➢ sentences：每个句子的首字母大写 ➢ words：每个单词的首字母大写 ➢ characters：每个字母都大写
android:cursorVisible	setCursorVisible(boolean)	设置该文本框的光标是否可见
android:digits	setKeyListener(KeyListener)	如果该属性设为 true，则该文本框对应一个数字输入方法，并且只接受那些合法字符
android:drawableBottom	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的底端绘制指定图像
android:drawableEnd		在文本框内文本的结尾处绘制指定图像
android:drawableLeft	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的左边绘制指定图像
android:drawablePadding	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	设置文本框内文本与图形之间的间距
android:drawableRight	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的右边绘制指定图像
android:drawableStart		在文本框内文本的开始处绘制指定图像
android:drawableTop	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的顶端绘制指定图像
android:editable		设置该文本是否允许编辑
android:ellipsize	setEllipsize(TextUitls.TruncateAt)	设置当显示的文本超过了 TextView 的长度时如何处理文本内容。该属性支持如下属性值： ➢ none：不做任何处理 ➢ start：在文本开始处截断，并显示省略号 ➢ middle：在文本中间处截断，并显示省略号 ➢ end：在文本结尾处截断，并显示省略号 ➢ marquee：使用 marquee 滚动动画显示文本
android:ems	setEms(int)	设置该组件的宽度，以 em 为单位
android:fontFamily	setTypeface(Typeface)	设置该文本框内文本的字体
android:gravity	setGravity(int)	设置文本框内文本的对齐方式
android:height	setHeight(int)	设置该文本框的高度（以 pixel 为单位）
android:hint	setHint(int)	设置当该文本框内容为空时，文本框内默认显示的提示文本
android:imeActionId	setImeActionLabel(CharSequence , int)	当该文本框关联输入法时，为输入法提供 EditorInfo.actionId 值
android:imeActionLabel	setImeActionLabel(CharSequence , int)	当该文本框关联输入法时，为输入法提供 EditorInfo.actionLabel 值

续表

XML 属性	相关方法	说明
android:imeOptions	setImeOptions(int)	当该文本框关联输入法时，为输入法指定额外的选项
android:includeFontPadding	setIncludeFontPadding(boolean)	设置是否为字体保留足够的空间。默认值为true
android:inputMethod	setKeyListener(KeyListener)	为该文本框指定特定的输入法。该属性值为输入法的全限定类名
android:inputType	setRawInputType(int)	指定该文本框的类型。该属性有点类似于HTML 中<input.../>元素的 type 属性。该属性支持大量属性值，不同属性值用于指定特定的输入框
android:lineSpacingExtra	setLineSpacing(float,float)	控制两行文本之间的额外间距。与android:lineSpacingMultiplier 属性结合使用
android:lineSpacingMultiplier	setLineSpacing(float,float)	控制两行文本之间的额外间距。每行文本为高度*该属性值+ android:lineSpacingExtra 属性值
android:lines	setLines(int)	设置该文本框默认占几行
android:linksClickable	setLinksClickable(boolean)	控制该文本框的 URL、E-mail 等链接是否可点击
android:marqueeRepeatLimit	setMarqueeRepeatLimit(int)	设置 marquee 动画重复的次数
android:maxEms	setMaxEms(int)	指定该文本框的最大宽度（以 em 为单位）
android:maxHeight	setMaxHeight(int)	指定该文本框的最大高度（以 pixel 为单位）
android:maxLength	setFilters(InputFilter)	设置该文本框的最大字符长度
android:maxLines	setMaxLines(int)	设置该文本框最多占几行
android:maxWidth	setMaxWidth(int)	指定该文本框的最大宽度（以 pixel 为单位）
android:minEms	setMinEms(int)	指定该文本框的最小宽度（以 em 为单位）
android:minHeight	setMinHeight(int)	指定该文本框的最小高度（以 pixel 为单位）
android:minLines	setMinLines(int)	设置该文本框最少占几行
android:minWidth	setMinWidth(int)	指定该文本框的最小宽度（以 pixel 为单位）
android:numeric	setKeyListener(KeyListener)	设置该文本框关联的数值输入法。该属性支持如下属性值： ➢ integer：指定关联整数输入法 ➢ signed：允许输入符号的数值输入法 ➢ decimal：允许输入小数点的数值输入法
android:password	setTransformationMethod(TransformationMethod)	设置该文本框是一个密码框（以点代替字符）
android:phoneNumber	setKeyListener(KeyListener)	设置该文本框只能接受电话号码
android:privateImeOptions	setPrivateImeOptions(String)	设置该文本框关联的输入法的私有选项
android:scrollHorizontally	setHorizontallyScrolling(boolean)	设置当该文本框不够显示全部内容时是否允许水平滚动
android:selectAllOnFocus	setSelectAllOnFocus(boolean)	如果文本框的内容可选择，设置是否当它获得焦点时自动选中所有文本
android:shadowColor	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影颜色
android:shadowDx	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影在水平方向的偏移
android:shadowDy	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影在垂直方向的偏移
android:shadowRadius	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影的模糊程度。该值越大，阴影越模糊
android:singleLine	setTransformationMethod	设置该文本框是否为单行模式。如果设为true，文本框不会换行
android:text	setText(CharSequence)	设置文本框内文本的内容

续表

XML 属性	相关方法	说明
android:textAllCaps	setAllCaps(boolean)	设置是否将文本框的所有字母显示为大写字母
android:textAppearance		设置该文本框的颜色、字体、大小等样式
android:textColor	setTextColor(ColorStateList)	设置文本框中文本的颜色
android:textColorHighlight	setHighlightColor(int)	设置文本框中文本被选中时的颜色
android:textColorHint	setHintTextColor(int)	设置文本框中提示文本的颜色
android:textColorLink	setLinkTextColor(int)	设置该文框中链接的颜色
android:textIsSelectable	isTextSelectedable()	设置该文本框不能编辑时，文本框内的文本是否可以被选中
android:textScaleX	setTextScaleX(float)	设置文本框内文本在水平方向上的缩放因子
android:textSize	setTextSize(float)	设置文本框内文本的字体大小
android:textStyle	setTypeface(Typeface)	设置文本框内文本的字体风格，如粗体、斜体等
android:typeface	setTypeface(Typeface)	设置文本框内文本的字体风格
android:width	setWidth(int)	设置该文本框的宽度（以 pixel 为单位）

下面通过一系列实例来介绍 TextView 和 CheckedTextView 的用法。

实例：不同颜色、字体、带链接的文本

由于 TextView 提供了大量 XML 属性，因此我们可以通过这些 XML 属性来控制 TextView 中文本的行为，例如如下界面布局文件。

程序清单：codes\02\2.3\TextViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
    <!-- 设置字号为 20pt，在文本框结尾处绘制图片 -->
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="我爱 Java"
    android:textSize="20pt"
    android:drawableEnd="@drawable/ic_launcher"/>
    <!-- 设置中间省略，所有字母大写 -->
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:text="我爱 Java 我爱 Java 我爱 Java 我爱 Java 我爱 Java 我 aaaJava"
    android:ellipsize="middle"
    android:textAllCaps="true"/>
    <!-- 对邮件、电话增加链接 -->
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:text="邮件是 kongyeeku@163.com，电话是 02088888888"
    android:autoLink="email|phone"/>
    <!-- 设置文字颜色、大小，并使用阴影 -->
    <TextView
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:text="测试文字"
    android:shadowColor="#00f"
    android:shadowDx="10.0"
    android:shadowDy="8.0"
    android:shadowRadius="3.0"
    android:textColor="#f00"
    android:textSize="18pt"/>
    <!-- 测试密码框 -->
    <TextView android:id="@+id/passwd"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:password="true"/>
    <CheckedTextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="可勾选的文本"
    android:checkMark="@drawable/ok" />
</LinearLayout>
```

上面的界面布局文件中定义了 6 个 TextView，它们指定的规则如下。

- 第 1 个 TextView 指定 android:textSize="20pt"，这就指定了字号为 20pt。而且指定了在文本框的结尾处绘制图片。
- 第 2 个 TextView 指定 android:ellipsize="middle"，这就指定了当文本多于文本框的宽度时，从中间省略文本。而且指定了 android:textAllCaps="true"，表明该文本框的所有字母大写。
- 第 3 个 TextView 指定 android:autoLink="email|phone"，这就指定了该文本框会自动为文本框内的 E-mail 地址、电话号码添加超链接。
- 第 4 个 TextView 指定一系列 android:shadowXXX 属性，这就为该文本框内的文本内容添加了阴影。
- 第 5 个 TextView 指定 android:password="true"，这就指定了该文本框会用点来代替显示所有字符。
- 第 6 个 CheckedTextView 指定 android:checkMark="@drawable/ok"，这就指定了该可勾选文本框的勾选图标。

使用 Activity 来显示上面的界面布局文件，将看到如图 2.17 所示的界面。

图 2.17 不同字体、颜色带链接的文本

实例：圆角边框、渐变背景的 TextView

在默认情况下，TextView 是不带边框的，如果想为 TextView 添加边框，只能通过“曲线救国”的方式来实现——我们可以考虑为 TextView 设置一个背景 Drawable，该 Drawable 只是一个边框，这样就实现了带边框的 TextView。

由于可以为 TextView 设置背景 Drawable 对象，因此可以在定义 Drawable 时不仅指定边框，还可以指定渐变背景，这样即可为 TextView 添加渐变背景和边框。

下面的界面布局文件中定义了两个 TextView，界面布局文件的代码如下。

程序清单：codes\02\2.3\TextViewTest2\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 通过 android:background 指定背景 -->
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="带边框的文本"
    android:textSize="24pt"
    android:background="@drawable/bg_border"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="圆角边框、渐变背景的文本"
    android:textSize="24pt"
    android:background="@drawable/bg_border2"/>
</LinearLayout>
```

上面的界面布局文件中定义了两个 TextView，其中第一个指定了背景，第二个定义文本框时指定使用圆角边框、渐变背景。第一个文本框所指定的背景是由 XML 文件定义的，将该文件放在 drawable 文件夹内，该 XML 文件也可当成 Drawable 使用。下面是该 XML 文件的代码。

程序清单：codes\02\2.3\TextViewTest2\app\src\main\res\drawable\bg_border.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置背景色为透明色 -->
    <solid android:color="#0000"/>
    <!-- 设置红色边框 -->
    <stroke android:width="4px" android:color="#f00" />
</shape>
```

第二个文本框所指定的背景是由 XML 文件定义的，将该文件放在 drawable 文件夹内，该 XML 文件也可当成 Drawable 使用。下面是该 XML 文件的代码。

程序清单：codes\02\2.3\TextViewTest2\app\src\main\res\drawable\bg_border2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- 指定圆角矩形的 4 个圆角的半径 -->
      <corners android:topLeftRadius="20px"
    android:topRightRadius="5px"
    android:bottomRightRadius="20px"
    android:bottomLeftRadius="5px"/>
      <!-- 指定边框线条的宽度和颜色 -->
    <stroke android:width="4px" android:color="#f0f" />
    <!-- 指定使用渐变背景色，使用 sweep 类型的渐变
    颜色从红色→绿色→蓝色 -->
    <gradient android:startColor="#f00"
       android:centerColor="#0f0"
       android:endColor="#00f"
       android:type="sweep"/>
</shape>
```

使用 Activity 来显示上面定义的布局页面，可以看到如图 2.18 所示的界面。

从图 2.18 不难看出，通过为 TextView 的 android:background 赋值，可以为文本框增加大量自定义外观，这种控制方式非常灵活。

需要指出的是，表面上这里只是在介绍 TextView，但由于 TextView 是 EditText、Button 等类的父类，因此此处介绍的对 TextView 控制的属性，同样适用于 EditText 与 Button。

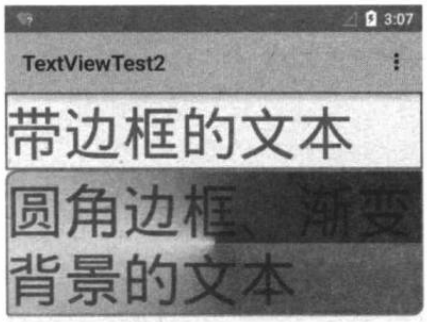


图 2.18　圆角边框、渐变背景的文本

2.3.2　EditText 的功能与用法

EditText 与 TextView 非常相似，它甚至与 TextView 共用了绝大部分 XML 属性和方法。EditText 与 TextView 的最大区别在于：EditText 可以接受用户输入。表 2.13 中介绍的与输入相关的属性主要就是为 EditText 准备的。

EditText 组件最重要的属性是 inputType，该属性相当于 HTML 的<input.../>元素的 type 属性，用于将 EditText 设置为指定类型的输入组件。inputType 能接受的属性值非常丰富，而且随着 Android 版本的升级，该属性能接受的类型还会增加。

EditText 还派生了如下两个子类。

- AutoCompleteTextView：带有自动完成功能的 EditText，实际上该组件的命名不太恰当。笔者认为该类名应该叫 AutoCompleteEditText 比较合适。由于该类通常需要与 Adapter 结合使用，因此将会在讲解 AdapteView 组件时介绍该组件的用法。
- ExtractEditText：它并不是 UI 组件，而是 EditText 组件的底层服务类，负责提供全屏输入法支持。

下面通过一个实例来介绍 EditText 的用法。

实例：用户友好的输入界面

对于一个用户友好的输入界面而言，接受用户输入的文本框内默认会提示用户如何输入；当用户把焦点切换到输入框时，输入框自动选中其中已输入的内容，避免用户删除已有内容；当用户把焦点切换到只接受电话号码的输入框时，输入法自动切换到数字键盘。

下面程序的输入界面完成了以上功能，输入界面的界面布局文件如下。

程序清单：codes\02\2.3\InputUI\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="用户名:"
    android:textSize="16sp"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="请填写登录账号"
    android:selectAllOnFocus="true"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="密码:"
```

```
    android:textSize="16sp"/>
<!-- android:inputType="numberPassword"表明只能接受数字密码 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="numberPassword"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="年龄："
    android:textSize="16sp"/>
<!-- inputType="number"表明是数值输入框 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="生日："
    android:textSize="16sp"/>
<!-- inputType="date"表明是日期输入框 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="date"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="电话号码:"
    android:textSize="16sp"/>
<!-- inputType="phone"表明是输入电话号码的输入框 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="请填写您的电话号码"
    android:selectAllOnFocus="true"
    android:inputType="phone"/>
</TableRow>
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="注册"/>
</TableLayout>
```

上面界面布局中的第一个文本框通过 android:hint 指定了文本框的提示信息：请填写登录账号——这是该文本框默认的提示。当用户还没有输入时，该文本框内默认显示这段信息；第二个输入框通过 android:inputType="numberPassword"设置这是一个密码框，而且只能接受数字密码，用户在该文本框输入的字符会以点号代替；第三个输入框通过 android:inputType="number"设置为只能接受数值的输入框；第四个输入框通过 android:inputType= "date"指定它是一个日期输入框；第五个输入框通过 android:inputType= "phone"设置为一个电话号码输入框。

使用 Activity 显示上面的界面布局文件，将可看到如图 2.19 所示的界面。

从图 2.19 可以看出，当用户把焦点定位到数字密码输入框时，系统自动显示数字输入键盘，这就是设置 android:inputType= "numberPassword"的作用。第一个文本框默认显示了“请填写登录账号”，这是由 android:hint 属性指定的。

图 2.19　友好的输入界面

➤➤ 2.3.3　按钮（Button）组件的功能与用法

Button 继承了 TextView，它主要是在 UI 界面上生成一个按钮，该按钮可以供用户单击，当用户单击按钮时，按钮会触发一个 onClick 事件。关于 onClick 事件编程的简单示例，本书前面已经见到很多，后面还会详细介绍 Android 的事件编程。

按钮使用起来比较容易，可以通过指定 android:background 属性为按钮增加背景颜色或背景图片，如果将背景图片设为不规则的背景图片，则可以开发出各种不规则形状的按钮。

如果只是使用普通的背景颜色或背景图片，那么这些背景是固定的，不会随着用户的动作而改变。如果需要让按钮的背景颜色、背景图片随用户动作动态改变，则可以考虑使用自定义 Drawable 对象来实现。

下面通过实例来开发出更强大的按钮。

实例：按钮、圆形按钮、带文字的图片按钮

为了定义图片随用户动作改变的按钮，可以考虑使用 XML 资源文件来定义 Drawable 对象，再将 Drawable 对象设为 Button 的 android:background 属性值，或设为 ImageButton 的 android:src 属性值。

例如，有如下界面布局文件。

程序清单：codes\02\2.3\ButtonTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 文字带阴影的按钮 -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="文字带阴影的按钮"
    android:textSize="12pt"
    android:shadowColor="#aa5"
    android:shadowRadius="1"
    android:shadowDx="5"
    android:shadowDy="5"/>
<!-- 普通文字按钮 -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/red"
    android:text="普通按钮"
    android:textSize="10pt"/>
<!-- 带文字的图片按钮-->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/button_selector"
```

```
        android:textSize="11px"
        android:text="带文字的图片按钮"/>
    </LinearLayout>
```

上面界面布局中的第一个按钮是一个普通按钮，但为该按钮的文字指定了阴影——配置阴影的方式与为 TextView 配置阴影的方式完全相同，这是因为 Button 的本质还是 TextView。第二个按钮通过 background 属性配置了背景图片，因此该按钮将会显示为背景图片形状的按钮。

第三个按钮有点特殊，它指定了 android:background 属性为@drawable/button_selector，该属性值引用一个 Drawable 资源，该资源对应的 XML 文件如下。

程序清单：codes\02\2.3\ButtonTest\app\src\main\res\drawable\button_selector.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 指定按钮按下时的图片 -->
    <item android: state_pressed="true"
        android:drawable="@drawable/red"
    />
    <!-- 指定按钮松开时的图片 -->
    <item android: state_pressed="false"
        android:drawable="@drawable/purple"
    />
</selector>
```

上面的资源文件使用<selector.../>元素定义了一个 StateListDrawable 对象——本书后面会详细介绍如何使用 XML 文件来定义 Drawable 的内容，此处不再深入讲解。

使用 Activity 显示上面的界面布局文件，可以看到如图 2.20 所示的界面。

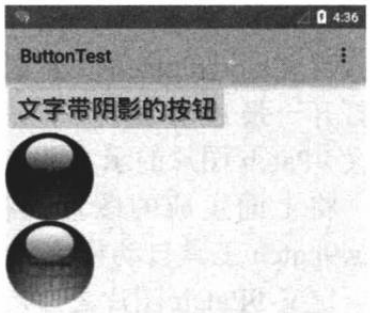


图 2.20 各种按钮

在图 2.20 所示界面的三个按钮中，前两个按钮的背景色、图片都是固定的，用户单击这两个按钮不会看到任何改变；用户单击第三个按钮时，将会看到按钮的图片被切换为红色。

以笔者的经验来看，Button 生成的按钮功能很强大，就像第三个按钮就是 Button 生成的，而且它也可以通过背景色来设置图片，因此使用 Button 生成的按钮不仅可以是普通的文字按钮，也可以定制成任意形状，并可以随用户交互动作改变外观。

2.3.4 使用 9Patch 图片作为背景

从图 2.20 所示的第三个按钮来看，当按钮的内容太多时，Android 会自动缩放整张图片，以保证背景图片能覆盖整个按钮。但这种缩放整张图片的效果可能并不好。可能存在的情况是我们只想缩放图片中某个部分，这样才能保证按钮的视觉效果。

为了实现只缩放图片中某个部分的效果，我们需要借助于 9Patch 图片来实现。9Patch 图片是一种特殊的 PNG 图片，这种图片以.9.png 结尾，它在原始图片四周各添加一个宽度为 1 像素的线条，这 4 条线就决定了该图片的缩放规则、内容显示规则。

左侧和上侧的直线共同决定了图片的缩放区域：以左边直线为左边界绘制矩形，它覆盖的区域可以在纵向上缩放；以上面直线为上边界绘制矩形，它覆盖的区域可以水平缩放；它们二者的交集可以在两个方向上缩放。图 2.21 显示了定义图片缩放区域的示意图。

右侧和下侧的直线共同决定图片的内容显示区域：以右边直线为右边界绘制矩形，以下边直线为下边界绘制矩形，它们二者的交集就是图片的内容显示区域。图 2.22 显示了定义图片

的内容显示区域的示意图。

图 2.21　定义图片缩放区域

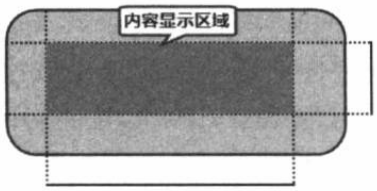


图 2.22　定义图片的内容显示区域

Android 为制作 9Patch 图片提供了 draw9patch 工具，该工具位于 Android SDK 安装路径的 tools 目录下，进入该目录双击 draw9patch.bat 文件，即可启动该工具。

提示：

如果启动该程序时提示 java.lang.NoClassDefFoundError:org/jdesktop/swingworker/SwingWorker 异常，则可能的原因是 draw9patch.bat 工具需要依赖于 SwingWorker 类，这个类在 JDK 1.5 以前需要单独下载 swing-worker 的 JAR 包，从 JDK 1.6 以后，该类已经加入了 javax.swing 包下，但 Android SDK 还在使用早期的 SwingWorker，而且 Android SDK 又删除了 swing-worker-1.1.jar 包，所以引发上面的错误。为了能正常启动 draw9patch 工具，手动将 swing-worker-1.1.jar 复制到 Android SDK 安装路径的 tools/lib 路径下即可。

启动 draw9patch 工具之后，通过该工具主菜单上的“File”→“Open 9”-“Patch”菜单项打开一张 PNG 图片，然后通过该工具定义图片的缩放区域、内容显示区域。图 2.23 显示了定义 9Patch 图片的示意图。

将上面生成的图片保存到应用的 app\src\main\res\drawable 路径下，主文件名任意，draw9patch 工具自动将该文件的后缀保存为.9.png。

定义 9Patch 图片之后，接下来在应用中定义两个按钮，分别使用原始图片和 9Patch 图片作为 Button 的背景。页面定义文件比较简单，此处不再赘述。普通图片为背景的按钮及 9Patch 图片为背景的按钮的界面如图 2.24 所示。

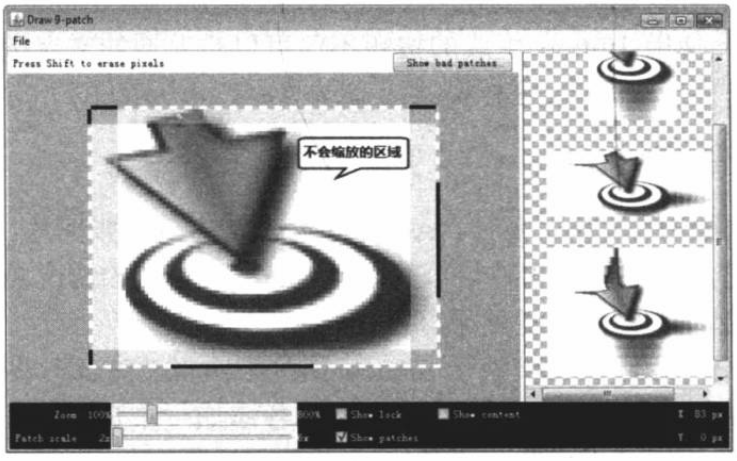


图 2.23　定义 9Patch 图片

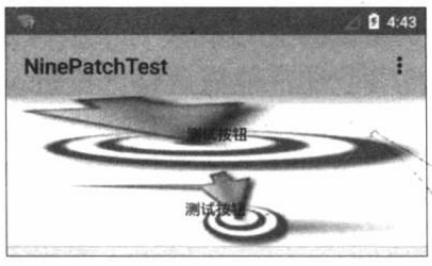


图 2.24　普通图片与 9Patch 图片作为背景的对比

从图 2.24 可以看出，普通图片作为背景时，整张图片都被缩放了；如果使用 9Patch 图片作为背景，则只有指定区域才会被缩放。

2.3.5　单选钮（RadioButton）和复选框（CheckBox）的功能与用法

单选钮（RadioButton）、复选框（CheckBox）、状态开关按钮（ToggleButton）和开关（Switch）

是用户界面中最普通的 UI 组件，它们都继承了 Button 类，因此都可直接使用 Button 支持的各种属性和方法。

RadioButton、CheckBox 与普通按钮不同的是，它们多了一个可选中的功能，因此 RadioButton、CheckBox 都可额外指定一个 android:checked 属性，该属性用于指定 RadioButton、CheckBox 初始时是否被选中。

RadioButton 与 CheckBox 的不同之处在于，一组 RadioButton 只能选中其中一个，因此 RadioButton 通常要与 RadioGroup 一起使用，用于定义一组单选钮。

下面通过实例来介绍 RadioButton 和 CheckBox 的用法。

实例：利用单选钮、复选框获取用户信息

在需要获取用户信息的界面中，有些信息不需要用户直接输入，可以考虑让用户进行选择，比如用户的性别、爱好等。下面的界面布局文件定义了一个让用户选择的输入界面。

程序清单：codes\02\2.3\CheckButtonTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<TableRow>
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="性别:"/>
<!-- 定义一组单选钮 -->
<RadioGroup android:id="@+id/rg"
    android:orientation="horizontal"
    android:layout_gravity="center_horizontal">
<!-- 定义两个单选钮 -->
<RadioButton android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/male"
    android:text="男"
    android:checked="true"/>
<RadioButton android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/female"
    android:text="女"/>
</RadioGroup>
</TableRow>
<TableRow>
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="喜欢的颜色:" />
<!-- 定义一个垂直的线性布局 -->
<LinearLayout android:layout_gravity="center_horizontal"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
<!-- 定义三个复选框 -->
<CheckBox android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="红色"
    android:checked="true"/>
<CheckBox android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="蓝色"/>
```

```
<CheckBox android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="绿色"/>
</LinearLayout>
</TableRow>
<TextView
      android:id="@+id/show"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</TableLayout>
```

上面的界面布局中定义了一组单选钮，并默认勾选了第一个单选钮，这组单选钮供用户选择性别；还定义了三个复选框，供用户选择喜欢的颜色。

注意：

如果在 XML 布局文件中默认勾选了某个单选钮，则必须为该组单选钮的每个按钮指定 android:id 属性值；否则这组单选钮不能正常工作。

为了监听单选钮、复选框的勾选状态的改变，可以为它们添加事件监听器。例如，下面 Activity 为 RadioGroup 添加了事件监听器，该监听器可以监听这组单选钮的勾选状态的改变。

程序清单：codes\02\2.3\CheckButtonTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    RadioGroup rg;
    TextView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面上 rg、show 两个组件
        rg = (RadioGroup) findViewById(R.id.rg);
        show = (TextView) findViewById(R.id.show);
        // 为 RadioGroup 组件的 OnCheckedChanged 事件绑定事件监听器
        rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId)
            {
                // 根据用户勾选的单选按钮来动态改变 tip 字符串的值
                String tip = checkedId == R.id.male ?
                        "您的性别是男人": "您的性别是女人";
                // 修改 show 组件中的文本
                show.setText(tip);
            }
        });
    }
}
```

提示：

上面代码添加事件监听器的方式采用了“委托式”事件处理机制，委托式事件处理机制的原理是，当事件源上发生事件时，该事件将会激发该事件源上的监听器的特定方法。本书第 3 章还会详细介绍这种事件处理机制。

运行上面的程序，并改变第一组单选钮的勾选状态，将看到如图 2.25 所示的界面。

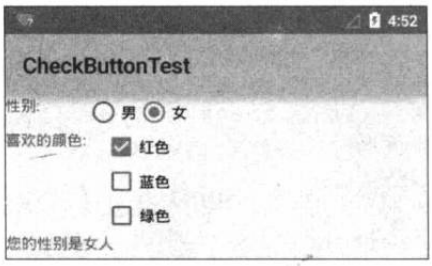


图 2.25 单选钮、复选框示意图

2.3.6 状态开关按钮（ToggleButton）和开关（Switch）的功能与用法

状态开关按钮（ToggleButton）和开关（Switch）也是由 Button 派生出来的，因此它们的本质也是按钮，Button 支持的各种属性、方法也适用于 ToggleButton 和 Switch。从功能上来看，ToggleButton、Switch 与 CheckBox 复选框非常相似，它们都可以提供两种状态。不过 ToggleButton、Switch 与 CheckBox 的区别主要体现在功能上，ToggleButton、Switch 通常用于切换程序中的某种状态。

表 2.14 显示了 ToggleButton 所支持的 XML 属性及相关方法。图 2.15 显示了 Switch 所支持的 XML 属性及相关方法。

表 2.14 ToggleButton 支持的 XML 属性及相关方法

XML 属性	相关方法	说明
android:checked	setChecked(boolean)	设置该按钮是否被选中
android:textOff		设置当该按钮的状态关闭时显示的文本
android:textOn		设置当该按钮的状态打开时显示的文本

表 2.15 Switch 支持的 XML 属性及相关方法

XML 属性	相关方法	说明
android:checked	setChecked(boolean)	设置该开关是否被选中
android:switchMinWidth	setSwitchMinWidth(int)	设置该开关的最小宽度
android:switchPadding	setSwitchPadding(int)	设置开关与标题文本之间的空白
android:switchTextAppearance	setSwitchTextAppearance(Context,int)	设置该开关图标上的文本样式
android:textOff	setTextOff(CharSequence)	设置该开关的状态关闭时显示的文本
android:textOn	setTextOn(CharSequence)	设置该开关的状态打开时显示的文本
android:textStyle	setSwitchTypeface(Typeface)	设置该开关的文本风格
android:thumb	setThumbResource(int)	指定使用自定义 Drawable 绘制该开关的开关按钮
android:track	setTrackResource(int)	指定使用自定义 Drawable 绘制该开关的开关轨道
android:typeface	setSwitchTypeface(Typeface)	设置该开关的文本的字体风格

下面通过一个动态控制布局的实例来示范 ToggleButton 与 Switch 的用法。

实例：动态控制布局

该实例的思路是在页面中增加一个 ToggleButton，随着该按钮状态的改变，界面布局中的 LinearLayout 布局的方向在水平布局、垂直布局之间切换。下面是该程序所使用的界面布局文件。

程序清单：codes\02\2.3\ToggleButtonTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
```

```
<!-- 定义一个 ToggleButton 按钮 -->
<ToggleButton android:id="@+id/toggle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="横向排列"
    android:textOn="纵向排列"
    android:checked="true"/>
<Switch android:id="@+id/switcher"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="横向排列"
    android:textOn="纵向排列"
    android:thumb="@drawable/check"
    android:checked="true"/>
<!-- 定义一个可以动态改变方向的线性布局 -->
<LinearLayout android:id="@+id/test"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 下面省略了三个按钮的定义 -->
...
</LinearLayout>
</LinearLayout>
```

上面 LinearLayout 中定义了三个按钮，该 LinearLayout 默认采用垂直方向的线性布局。接下来我们为 ToggleButton 按钮、Switch 按钮绑定监听器，当它的选中状态发生改变时，程序通过代码来改变 LinearLayout 的布局方向。

程序清单：codes\02\2.3\ToggleButtonTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    ToggleButton toggle;
    Switch switcher;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        toggle = (ToggleButton)findViewById(R.id.toggle);
        switcher = (Switch)findViewById(R.id.switcher);
        final LinearLayout test = (LinearLayout)findViewById(R.id.test);
        OnCheckedChangeListener listener = new OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(CompoundButton button
                , boolean isChecked)
            {
                if(isChecked)
                {
                    // 设置 LinearLayout 垂直布局
                    test.setOrientation(1);
                    toggle.setChecked(true);
                    switcher.setChecked(true);
                }
                else
                {
                    // 设置 LinearLayout 水平布局
                    test.setOrientation(0);
                    toggle.setChecked(false);
                    switcher.setChecked(false);
                }
            }
```

```
        };
        toggle.setOnCheckedChangeListener(listener);
        switcher.setOnCheckedChangeListener(listener);
    }
}
```

运行上面的程序，随着用户改变 ToggleButton 按钮的状态，界面布局的方向也在不断发生改变。图 2.26 显示了 ToggleButton 的界面。

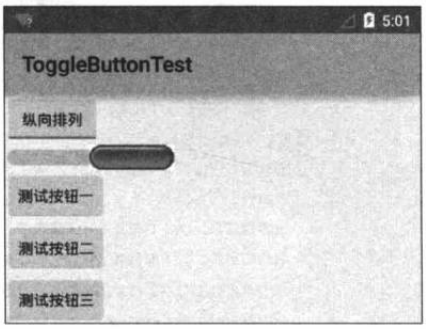


图 2.26 ToggleButton 与 Switch 的功能

2.3.7 时钟（AnalogClock 和 TextClock）的功能与用法

时钟 UI 组件是两个非常简单的组件：TextClock 本身就继承了 TextView——也就是说，它本身就是文本框，只是它里面显示的内容总是当前时间。与 TextView 不同的是，为 TextClock 设置 android:text 属性没什么作用。

TextClock 取代早期的 DigitalClock 组件，因此功能更加强大——TextClock 能以 24 小时制或 12 小时制来显示时间，而且可以由程序员来指定时间格式。表 2.17 显示了 TextClock 所支持的 XML 属性及相关方法

表 2.16 TextClock 支持的 XML 属性及相关方法

XML 属性	相关方法	说明
android:format12Hour	setFormat12Hour(CharSequence)	设置该时钟的 12 小时制的格式字符串
android:format24Hour	setFormat24Hour(CharSequence)	设置该时钟的 24 小时制的格式字符串
android:timeZone	setTimeZone(String)	设置该时钟的时区

AnalogClock 则继承了 View 组件，它重写了 View 的 OnDraw 方法，它会在 View 上绘制模拟时钟。

表 2.17 显示了 AnalogClock 所支持的 XML 属性。

表 2.17 AnalogClock 支持的 XML 属性

XML 属性	说明
android:dial	设置该模拟时钟的表盘使用的图片
android:hand_hour	设置该模拟时钟的时针使用的图片
android:hand_minute	设置该模拟时钟的分针使用的图片

TextClock 和 AnalogClock 都会显示当前时间。不同的是，TextClock 显示数字时钟，可以显示当前的秒数；AnalogClock 显示模拟时钟，不会显示当前的秒数。

下面通过实例来示范 AnalogClock 和 TextClock 的用法。

实例：手机里的“劳力士”

由于我们可以通过图片定制 AnalogClock 模拟指针的表盘、时针、分针，因此只要使用合适的图片，就可以对 AnalogClock 进行任意定制。本实例将会使用“劳力士”图片来定义模拟时钟，从而开发手机里的“劳力士”。

下面是本实例的界面布局文件。

程序清单：codes\02\2.3\ClockTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
    <!-- 定义模拟时钟 -->
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <!-- 定义数字时钟 -->
    <TextClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10pt"
        android:textColor="#f0f"
        android:format12Hour="yyyy年MM月dd日 H:mma EEEE"
        android:drawableEnd="@drawable/ic_launcher"/>
    <!-- 定义模拟时钟，并使用自定义表盘、时针图片 -->
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:dial="@drawable/watch"
        android:hand_minute="@drawable/hand"/>
</LinearLayout>
```

使用 Activity 显示上面的界面布局，将看到如图 2.27 所示的界面。

图 2.27 数字时钟和模拟时钟

正如从上面的粗体字代码中看到的，如果想控制模拟时钟显示时间的字号大小、字体颜色等，都可通过 android:textSize、android:textColor 等属性进行控制——因为 TextClock 的本质还是一个 TextView，所以它可以使用 TextView 的 XML 属性和方法。

2.3.8 计时器（Chronometer）

Android 还提供了一个计时器组件：Chronometer，该组件与 TextClock 都继承自 TextView，因此它们都会显示一段文本。但 Chronometer 并不显示当前时间，它显示的是从某个起始时间开始，一共过去了多长时间。

Chronometer 的用法也很简单，它只提供了一个 android:format 属性，用于指定计时器的计时格式。除此之外，Chronometer 还支持如下常用方法。

- setBase(long base)：设置计时器的起始时间。
- setFormat(String format)：设置显示时间的格式。
- start()：开始计时。
- stop()：停止计时。
- setOnChronometerTickListener(Chronometer.OnChronometerTickListener listener)：为计时器绑定事件监听器，当计时器改变时触发该监听器。

下面的程序简单示范了 Chronometer 的用法。该程序界面中定义了一个 Chronometer 组件和一个 Button 组件。当用户单击 Button 时系统开始计时，当计时超过 20 秒时停止计时。

程序清单：codes\02\2.3\ChronometerTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    Chronometer ch;
    Button start;
    @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取计时器组件
        ch = (Chronometer) findViewById(R.id.test);
        // 获取“开始”按钮
        start = (Button) findViewById(R.id.start);
        start.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 设置开始计时时间
                ch.setBase(SystemClock.elapsedRealtime());
                // 启动计时器
                ch.start();
                start.setEnabled(false);
            }
        });
        // 为 Chronometer 绑定事件监听器
        ch.setOnChronometerTickListener(new OnChronometerTickListener()
        {
            @Override
            public void onChronometerTick(Chronometer ch)
            {
                // 如果从开始计时到现在超过了 20s
                if (SystemClock.elapsedRealtime() - ch.getBase() > 20 * 1000)
                {
                    ch.stop();
                    start.setEnabled(true);
                }
            }
        });
    }
}
```

上面程序中的粗体字代码用于设置 Chronometer 的起始时间，并启动 Chronometer。启动计时器后可以看到该组件显示已流逝的时间，如图 2.28 所示。

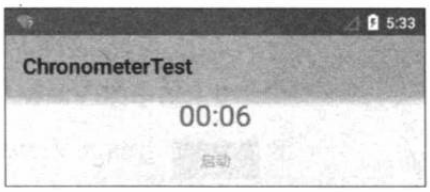


图 2.28 计时器

提示：

程序中用到的 SystemClock 类仅仅是一个获取系统时间、运行时间的工具类，用法很简单，读者自行查阅 API 文档即可。

2.4 第 3 组 UI 组件：ImageView 及其子类

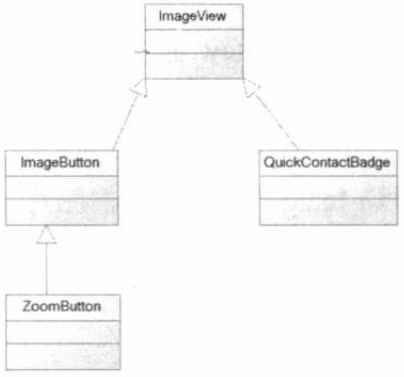


图 2.29 ImageView 及其子类的类图

ImageView 继承自 View 组件，它的主要功能是用于显示图片——实际上这个说法不太严谨，因为它能显示的不仅仅是图片，任何 Drawable 对象都可使用 ImageView 来显示。

除此之外，ImageView 还派生了 ImageButton、ZoomButton 等组件，图 2.29 显示了 ImageView 及其子类的类图。

从图 2.29 可以看出，ImageView 派生了 ImageButton、

ZoomButton 等组件，因此 ImageView 支持的 XML 属性、方法，基本上也可应用于 ImageButton、ZoomButton 等组件。

表 2.18 显示了 ImageView 支持的常用 XML 属性及相关方法。

表 2.18 ImageView 支持的 XML 属性及相关方法

XML 属性	相关方法	说明
android:adjustViewBounds	setAdjustViewBounds(boolean)	设置 ImageView 是否调整自己的边界来保持所显示图片的长宽比
android:cropToPadding	setCropToPadding(boolean)	如果将该属性设为 true，该组件将会被裁剪到保留该 ImageView 的 padding
android:maxHeight	setMaxHeight(int)	设置 ImageView 的最大高度
android:maxWidth	setMaxWidth(int)	设置 ImageView 的最大宽度
android:scaleType	setScaleType(ImageView.ScaleType)	设置所显示的图片如何缩放或移动以适应 ImageView 的大小
android:src	setImageResource(int)	设置 ImageView 所显示的 Drawable 对象的 ID

ImageView 所支持的 android:scaleType 属性可指定如下属性值。

- matrix（ImageView.ScaleType.MATRIX）：使用 matrix 方式进行缩放。
- fitXY（ImageView.ScaleType.FIT_XY）：对图片横向、纵向独立缩放，使得该图片完全适应于该 ImageView，图片的纵横比可能会改变。
- fitStart（ImageView.ScaleType.FIT_START）：保持纵横比缩放图片，直到该图片能完全显示在 ImageView 中（图片较长的边长与 ImageView 相应的边长相等），缩放完成后将该图片放在 ImageView 的左上角。
- fitCenter（ImageView.ScaleType.FIT_CENTER）：保持纵横比缩放图片，直到该图片能完全显示在 ImageView 中（图片较长的边长与 ImageView 相应的边长相等），缩放完成后将该图片放在 ImageView 的中央。
- fitEnd（ImageView.ScaleType.FIT_END）：保持纵横比缩放图片，直到该图片能完全显示在 ImageView 中（图片较长的边长与 ImageView 相应的边长相等），缩放完成后将该图片放在 ImageView 的右下角。
- center（ImageView.ScaleType.CENTER）：把图片放在 ImageView 的中间，但不进行任何缩放。
- centerCrop（ImageView.ScaleType.CENTER_CROP）：保持纵横比缩放图片，以使得图片能完全覆盖 ImageView。只要图片的最短边能显示出来即可。
- centerInside（ImageView.ScaleType.CENTER_INSIDE）：保持纵横比缩放图片，以使得 ImageView 能完全显示该图片。

为了控制 ImageView 显示的图片，ImageView 提供了如下方法。

- setImageBitmap(Bitmap bm)：使用 Bitmap 位图设置该 ImageView 显示的图片。
- setImageDrawable(Drawable drawable)：使用 Drawable 对象设置该 ImageView 显示的图片。
- setImageResource(int resId)：使用图片资源 ID 设置该 ImageView 显示的图片。
- setImageURI(Uri uri)：使用图片的 URI 设置该 ImageView 显示的图片。

ImageView 的功能比较简单，下面结合一个图片浏览器的实例来示范 ImageView 的功能和用法。

实例：图片浏览器

本例的图片浏览器可以改变所查看图片的透明度，可通过调用 ImageView 的 setImageAlpha()方法来实现。不仅如此，这个图片浏览器还可通过一个小区域来查看图片的原始大小，因此本例会定义两个 ImageView，一个用于查看图片整体，一个用于查看图片局部的细节。

下面是本例的界面布局文件。

程序清单：codes\02\2.4\ImageViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center">
<!-- 此处省略三个按钮定义 -->
...
</LinearLayout>
<!-- 定义显示图片整体的 ImageView -->
<ImageView android:id="@+id/image1"
    android:layout_width="wrap_content"
    android:layout_height="280dp"
    android:src="@drawable/shuangta"
    android:scaleType="fitCenter"/>
<!-- 定义显示图片局部细节的 ImageView -->
<ImageView android:id="@+id/image2"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:background="#00f"
    android:layout_margin="10dp"/>
</LinearLayout>
```

上面的界面布局文件中定义了两个 ImageView，其中第一个 ImageView 指定了 android:scaleType= "fitCenter"，这表明 ImageView 显示图片时会保持纵横比缩放，并将缩放后的图片放在该 ImageView 的中央。

为了能动态改变图片的透明度，接下来需要为按钮编写事件监听器，当用户单击按钮时动态改变图片的 Alpha 值。为了能动态显示图片的局部细节，程序为第一个 ImageView 添加 OnTouchListener 监听器，用户在第一个 ImageView 上发生触摸事件时，程序从原始图片中读取相应部分的图片，并将其显示在第二个 ImageView 中。下面是主程序的代码。

程序清单：codes\02\2.4\ImageViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义一个访问图片的数组
    int[] images = new int[]{
            R.drawable.lijiang,
            R.drawable.qiao,
            R.drawable.shuangta,
            R.drawable.shui,
            R.drawable.xiangbi,
```

```
};
// 定义默认显示的图片
int currentImg = 2;
// 定义图片的初始透明度
private int alpha = 255;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    final Button plus = (Button) findViewById(R.id.plus);
    final Button minus = (Button) findViewById(R.id.minus);
    final ImageView image1 = (ImageView) findViewById(R.id.image1);
    final ImageView image2 = (ImageView) findViewById(R.id.image2);
    final Button next = (Button) findViewById(R.id.next);
    // 定义查看下一张图片的监听器
    next.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            // 控制 ImageView 显示下一张图片
            image1.setImageResource(
                    images[++currentImg % images.length]);
        }
    });
    // 定义改变图片透明度的方法
    View.OnClickListener listener = new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            if (v == plus)
            {
                alpha += 20;
            }
            if (v == minus)
            {
                alpha -= 20;
            }
            if (alpha >= 255)
            {
                alpha = 255;
            }
            if (alpha <= 0)
            {
                alpha = 0;
            }
            // 改变图片的透明度
            image1.setImageAlpha(alpha);
        }
    };
    // 为两个按钮添加监听器
    plus.setOnClickListener(listener);
    minus.setOnClickListener(listener);
    image1.setOnTouchListener(new View.OnTouchListener()
    {
        @Override
        public boolean onTouch(View view, MotionEvent event)
        {
            BitmapDrawable bitmapDrawable = (BitmapDrawable) image1
                    .getDrawable();
            // 获取第一个图片显示框中的位图
            Bitmap bitmap = bitmapDrawable.getBitmap();
```

```
                // bitmap 图片实际大小与第一个 ImageView 的缩放比例
                double scale = 1.0 * bitmap.getHeight() / image1.getHeight();
                // 获取需要显示的图片的开始点
                int x = (int) (event.getX() * scale);
                int y = (int) (event.getY() * scale);
                if (x + 120 > bitmap.getWidth())
                {
                    x = bitmap.getWidth() - 120;
                }
                if (y + 120 > bitmap.getHeight())
                {
                    y = bitmap.getHeight() - 120;
                }
                // 显示图片的指定区域
                image2.setImageBitmap(Bitmap.createBitmap(bitmap
                    , x, y, 120, 120));
                image2.setImageAlpha(alpha);
                return false;
            }
        });
    }
}
```

上面程序中的第一行粗体字代码调用了 ImageView 的 setImageResoure()方法动态修改该 ImageView 所显示的图片，当用户单击“下一张”按钮时，即可控制 ImageView 显示图片数组中的下一张图片。程序中第二行粗体字代码会动态设置第一个 ImageView 的 Alpha 值，这就是动态改变该图片的透明度了。

程序中的粗体字代码段将会用于从源位图的触碰点“截取”源位图，并将截取的部分显示在第二个 ImageView 中。此时用到了 Bitmap 类，它是一个代表位图的类，调用它的 createBitmap()静态方法即可截取位图的指定部分，该方法返回截取区域生成的新位图。

运行上面的程序，将看到如图 2.30 所示的界面。

图 2.30 使用 ImageView 实现图片浏览器

ImageView 派生了如下两个子类。

- ImageButton：图片按钮。
- QuickContactBadge：显示关联到特定联系人的图片。

Button 与 ImageButton 的区别在于，Button 生成的按钮上显示文字，而 ImageButton 上则显示图片。需要指出的是，为 ImageButton 按钮指定 android:text 属性没用（ImageButton 的本质是 ImageView），即使指定了该属性，图片按钮上也不会显示任何文字。

如果考虑使用 ImageButton，图片按钮可以指定 android:src 属性，该属性既可使用静止的图片，也可使用自定义的 Drawable 对象，这样即可开发出随用户动作改变图片的按钮。

ImageButton 派生了一个 ZoomButton，ZoomButton 可以代表“放大”、“缩小”两个按钮。ZoomButton 的行为基本类似于 ImageButton，只是 Android 默认提供了 btn_minus、btn_plus 两个 Drawable 资源，只要为 ZoomButton 的 android:src 属性分别指定 btn_minus、btn_plus，即可实现“缩小”、“放大”按钮。

实际上 Android 还提供了一个 ZoomControls 组件，该组件相当于同时组合了“放大”、“缩小”两个按钮，并允许分别为两个按钮绑定不同的事件监听器。

实例：强大的图片按钮

本实例定义了多个图片按钮，并定义了两个 ZoomButton。两个 ZoomButton 的 android:src 属性分别指定为@android:drawable/btn_minus、@android:drawable/btn_plus，这样即可定义“缩小”和“放大”两个按钮。

下面是该实例的界面布局文件。

程序清单：codes\02\2.4\ImageButtonTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 普通图片按钮 -->
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/blue"/>
<!-- 按下时显示不同图片的按钮 -->
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_selector"/>
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="10sp"
    android:layout_gravity="center_horizontal">
    <!-- 分别定义两个 ZoomButton，并分别使用 btn_minus 和 btn_plus 图片 -->
    <ZoomButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btn_zoom_down"
        android:src="@android:drawable/btn_minus" />
    <ZoomButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btn_zoom_up"
        android:src="@android:drawable/btn_plus" />
</LinearLayout>
<!-- 定义 ZoomControls 组件 -->
<ZoomControls android:id="@+id/zoomControls1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"/>
</LinearLayout>
```

上面界面布局文件的开头定义了两个 ImageButton，第一个 ImageButton 的 android:src 指定为一张静态图片，这样无论用户有怎样的行为，该 ImageButton 总显示这张静态图片。第二个 ImageButton 的 android:src 指定为@drawable/button_selector，该 Drawable 组合了两张图片，可以保证用户单击该按钮时切换图片。

该布局文件中间定义了两个 ZoomButton，并分别指定了放大、缩小 Drawable。该布局文件的结尾处定义了一个 ZoomControls 组件。使用 Activity 显示该界面布局文件，可以看到如图 2.31 所示的效果。

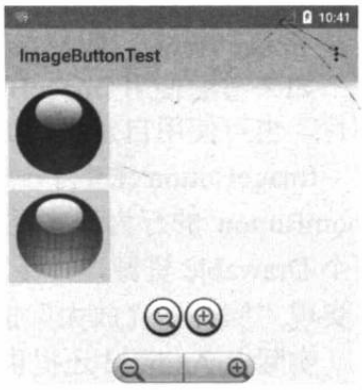


图 2.31 强大的图片按钮

对于第二个图片按钮，当用户单击第二个图片按钮时，将会看到按钮的图片被切换为红色图片。

实例：使用 QuickContactBadge 关联联系人

QuickContactBadge 继承了 ImageView，因此它的本质也是图片按钮，也可以通过 android:src 属性指定它显示的图片。QuickContactBadge 额外增加的功能是：该图片可以关联到手机中指定联系人，当用户单击该图片时，系统将会打开相应联系人的联系方式界面。

为了让 QuickContactBadge 与特定联系人关联，可以调用如下方法。

- assignContactFromEmail(String emailAddapp\src\main\ress, boolean lazyLookup)：将该图片关联到指定 E-mail 地址对应的联系人。
- assignContactFromPhone(String phoneNumber, boolean lazyLookup)：将该图片关联到指定电话号码对应的联系人。
- assignContactUri(Uri contactUri)：将该图片关联到特定 Uri 对应的联系人。

本实例示范了如何使用 QuickContactBadge 关联特定联系人。

该实例的界面布局文件如下。

程序清单：codes\02\2.4\QuickContactBadgeTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<QuickContactBadge
    android:id="@+id/badge"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/ic_launcher"/>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="16dp"
    android:text="我的偶像"/>
</LinearLayout>
```

上面的界面布局文件非常简单，它只是包含了一个 QuickContactBadge 组件与 TextView 组件。接下来在 Activity 代码中可以让 QuickContactBadge 与特定联系人建立关联。下面是该 Activity 的代码。

程序清单：codes\02\2.4\QuickContactBadgeTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    QuickContactBadge badge;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 QuickContactBadge 组件
        badge = (QuickContactBadge) findViewById(R.id.badge);
        // 将 QuickContactBadge 组件与特定电话号码对应的联系人建立关联
        badge.assignContactFromPhone("020-88888888", false);
    }
}
```

上面的粗体字代码将该 QuickContactBadge 组件与电话号码为 02088888888 的联系人建立了关联，当用户单击该图片时，系统将会打开该联系人对应的联系方式界面。

运行该实例，可以看到如图 2.32 所示的界面。

图 2.32 左边的图片就是 QuickContactBadge 组件，单击该组件，如果手机中存储有电话号码为 02088888888 的联系人，系统将会打开该联系人的联系方式界面，如图 2.33 所示。

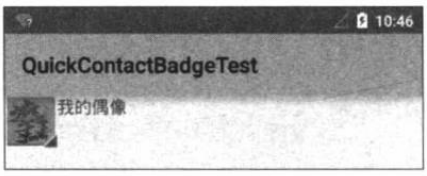


图 2.32 使用 QuickContactBadge

图 2.33 使用 QuickContactBadge 打开特定联系人

2.5 第 4 组 UI 组件：AdapterView 及子类

AdapterView 是一组重要的组件，AdapterView 本身是一个抽象基类，它派生的子类在用法上十分相似，只是显示界面有一定的区别，因此本节把它们归为一类，针对它们的共性集中讲解，并突出介绍它们的区别。

AdapterView 具有如下特征。

- AdapterView 继承了 ViewGroup，它的本质是容器。
- AdapterView 可以包括多个“列表项”，并将多个“列表项”以合适的形式显示出来。
- AdapterView 显示的多个“列表项”由 Adapter 提供。调用 AdapterView 的 setAdapter(Adapter)方法设置 Adapter 即可。

AdapterView 及其子类的继承关系图如图 2.34 所示。

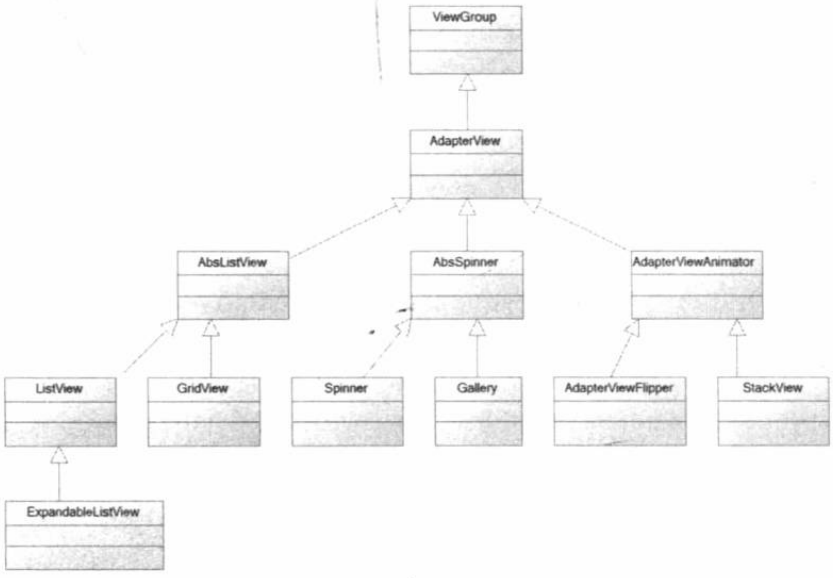


图 2.34 AdapterView 及其子类的继承关系图

从图 2.34 不难看出，AdapterView 派生了三个子类：AbsListView、AbsSpinner 和 AdapterViewAnimator，这三个子类依然是抽象的，实际使用时往往采用它们的子类。

注意：

由于 Gallery 是一个已经过时的 API，Android 推荐使用 HorizontalScrollView 来代替它，因此本书不再介绍 Gallery 的功能和用法。

2.5.1 列表视图（ListView）和 ListActivity

ListView 是手机系统中使用非常广泛的一种组件，它以垂直列表的形式显示所有列表项。生成列表视图有如下两种方式。

- 直接使用 ListView 进行创建。
- 让 Activity 继承 ListActivity（相当于该 Activity 显示的组件为 ListView）。

一旦在程序中获得了 ListView 之后，接下来就需要为 ListView 设置它要显示的列表项了。在这一点上，ListView 显示出 AdapterView 的特征：通过 setAdapter(Adapter)方法为之提供 Adapter，并由 Adapter 提供列表项即可。

提示：

ListView、GridView、Spinner、Gallery 等 AdapterView 都只是容器，而 Adapter 负责提供每个“列表项”组件，AdapterView 则负责采用合适的方式显示这些列表项。

AbsListView 提供的常用 XML 属性及相关方法如表 2.19 所示。

表 2.19 AbsListView 的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:choiceMode		设置 AbsListView 的选择行为。该属性支持如下属性值： ➢ none：不显示任何选中项 ➢ singleChoice：允许单选 ➢ multipleChoice：允许多选 ➢ multipleChoiceModal：允许多选
android:drawSelectorOnTop	setDrawSelectorOnTop(boolean)	如果该属性设置为 true，选中的列表项将会显示在上面
android:fastScrollEnabled		设置是否允许快速滚动。如果该属性设置为 true，将会显示滚动图标，并允许用户拖动该滚动图标进行快速滚动
android:listSelector	setSelector(int)	指定被选中的列表项上绘制的 Drawable
android:scrollingCache		如果设置为 true，该组件在滚动时将会使用绘制缓存
android:smoothScrollbar	setSmoothScrollbarEnabled(boolean)	如果设置为 false，则不在 header View 之后绘制分隔条
android:stackFromBottom		设置是否从底端开始排列列表项
android:textFilterEnabled		设置是否对列表项进行过滤。当该 AbsListView 对应的 Adapter 实现了 Filter 接口时该属性才会起作用
android:transcriptMode		设置该组件的滚动模式。该属性支持如下属性值： ➢ disabled：关闭滚动。这是默认值 ➢ normal：当该 AbsListView 收到数据改变通知，且最后一个列表项可见时，该 AbsListView 将会滚动到底端 ➢ alwaysScroll：该 AbsListView 总会自动滚动到底端

ListView 提供的常用 XML 属性如表 2.20 所示。

表 2.20　ListView 的常用 XML 属性

XML 属性	说明
android:divider	设置 List 列表项的分隔条（既可用颜色分隔，也可用 Drawable 分隔）
android:dividerHeight	设置分隔条的高度
android:entries	指定一个数组资源，Android 将根据该数组资源来生成 ListView
android:footerDividersEnabled	如果设置为 false，则不在 footer View 之前绘制分隔条
android:headerDividersEnabled	如果设置为 false，则不在 header View 之后绘制分隔条

下面通过几个实例来示范 ListView 的功能与用法。

实例：改变分隔条、基于数组的 ListView

下面的界面布局文件中定义了两个 ListView。

程序清单：codes\02\2.5\SimpleListViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 直接使用数组资源给出列表项 -->
<!-- 设置使用红色的分隔条 -->
<ListView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/books"
    android:divider="#f00"
    android:dividerHeight="2px"
    android:headerDividersEnabled="false"/>
</LinearLayout>
```

上面的界面布局文件中定义了一个 ListView，并通过 android:entries 指定了列表项数组，该 ListView 还通过 android:divider 改变了列表项之间的分隔条。

上面这个 ListView 指定了 android:entries="@array/books"，该属性值用到了数组资源，因此还需要在应用中定义一个名为 books 的数组，定义数组的资源文件如下。

程序清单：codes\02\2.5\SimpleListViewTest\app\src\main\res\values\arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="books">
        <item>疯狂 Java 讲义</item>
        <item>疯狂 Ajax 讲义</item>
        <item>疯狂 XML 讲义</item>
        <item>疯狂 Android 讲义</item>
    </string-array>
</resources>
```

使用 Activity 显示上面的 ListView，将可以看到如图 2.35 所示的效果。

使用数组创建 ListView 十分简单，但这种 ListView 能定制的内容很少，甚至连每个列表项的字号大小、颜色都不能改变。

如果想对 ListView 的外观、行为进行定制，就需要把 ListView 作为 AdapterView 使用，通过 Adapter 控制每个列表项的外观和行为。

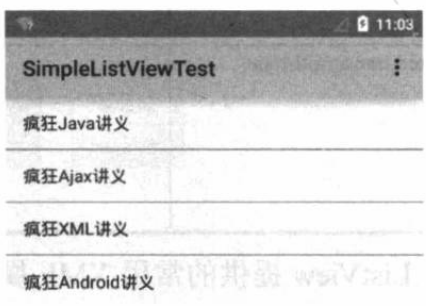


图 2.35　使用数组创建 ListView

2.5.2 Adapter 接口及实现类

Adapter 本身只是一个接口，它派生了 ListAdapter 和 SpinnerAdapter 两个子接口，其中 ListAdapter 为 AbsListView 提供列表项，而 SpinnerAdapter 为 AbsSpinner 提供列表项。Adapter 接口及其实现类的继承关系图如图 2.36 所示。

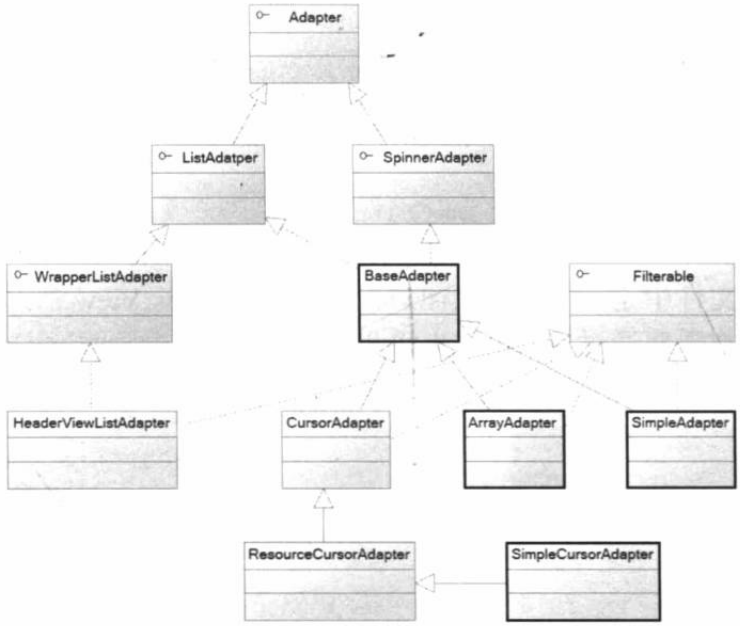


图 2.36 Adapter 及其实现类的继承关系图

图 2.36 所示继承关系类图中粗线框标出的是比较常用的 Adapter。从图 2.36 中所示的继承关系可以看出，几乎所有的 Adapter 都继承了 BaseAdapter，而 BaseAdapter 同时实现了 ListAdapter、SpinnerAdapter 两个接口，因此 BaseAdapter 及其子类可以同时为 AbsListView、AbsSpinner 提供列表项。

Adapter 常用的实现类如下。

- ArrayAdapter：简单、易用的 Adapter，通常用于将数组或 List 集合的多个值包装成多个列表项。
- SimpleAdapter：并不简单、功能强大的 Adapter，可用于将 List 集合的多个对象包装成多个列表项。
- SimpleCursorAdapter：与 SimpleAdapter 基本相似，只是用于包装 Cursor 提供的数据。
- BaseAdapter：通常用于被扩展。扩展 BaseAdapter 可以对各列表项进行最大限度的定制。

下面先通过 ArrayAdapter 来实现 ListView。

实例：使用 ArrayAdapter 创建 ListView

本实例在界面布局文件中定义了两个 ListView。

程序清单：codes\02\2.5\ArrayAdapterTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 设置使用红色的分隔条 -->
```

```
<ListView
    android:id="@+id/list1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:divider="#f00"
    android:dividerHeight="2px"
    android:headerDividersEnabled="false"/>
<!-- 设置使用绿色的分隔条 -->
<ListView
    android:id="@+id/list2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:divider="#0f0"
    android:dividerHeight="2px"
    android:headerDividersEnabled="false"/>
</LinearLayout>
```

上面的界面布局文件中定义了两个ListView，但这两个ListView都没有指定android:entries属性，这意味着它们都需要通过Adapter来提供列表项。

接下来Activity为两个ListView提供Adapter， Adapter决定ListView所显示的列表项。程序如下。

程序清单：codes\02\2.5\ArrayAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ListView list1 = (ListView) findViewById(R.id.list1);
        // 定义一个数组
        String[] arr1 = { "孙悟空", "猪八戒", "牛魔王" };
        // 将数组包装为ArrayAdapter
        ArrayAdapter<String> adapter1 = new ArrayAdapter<String>
            (this, R.layout.array_item, arr1);
        // 为ListView设置Adapter
        list1.setAdapter(adapter1);
        ListView list2 = (ListView) findViewById(R.id.list2);
        // 定义一个数组
        String[] arr2 = { "Java", "Hibernate", "Spring" , "Android" };
        // 将数组包装为ArrayAdapter
        ArrayAdapter<String> adapter2 = new ArrayAdapter<String>
            (this, R.layout.checked_item, arr2);
        // 为ListView设置Adapter
        list2.setAdapter(adapter2);
    }
}
```

上面程序中的两行粗体字代码创建了两个ArrayAdapter，创建ArrayAdapter时必须指定如下三个参数。

- Context：这个参数无须多说，它代表了访问整个Android应用的接口。几乎创建所有组件都需要传入Context对象。
- textViewResourceId：一个资源ID，该资源ID代表一个TextView，该TextView组件将作为ArrayAdapter的列表项组件。
- 数组或List：该数组或List将负责为多个列表项提供数据。

从上面的介绍不难看出，创建ArrayAdapter时传入的第二个参数控制每个列表项的组件，

第三个参数则负责为列表项提供数据。该数组或 List 包含多少个元素，就将生成多少个列表项，每个列表项都是 TextView 组件，TextView 组件显示的文本由数组或 List 的元素提供。

以上面代码中的第一个 ArrayAdapter 为例，该 ArrayAdapter 对应的数组为{ "孙悟空", "猪八戒", "牛魔王" }，该数组将会负责生成一个包含三个列表项的 ArrayAdapter，每个列表项的组件外观由 R.layout.array_item 布局文件（该布局文件只是一个 TextView 组件）控制，第一个 TextView 列表项显示的文本为“孙悟空”，第二个 TextView 列表项显示的文本为“猪八戒”……

上面程序中的 R.layout.array_item 布局文件如下。

程序清单：codes\02\2.5\ArrayAdapterTest\app\src\main\res\layout\array_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="24dp"
    android:padding="10px"
    android:shadowColor="#f0f"
    android:shadowDx="4"
    android:shadowDy="4"
    android:shadowRadius="2"/>
```

上面程序中的R.layout.checked_item布局文件与上面的布局文件类似，具体可以参考本书光盘代码。

运行上面的程序，将可以看到如图 2.37 所示的效果。

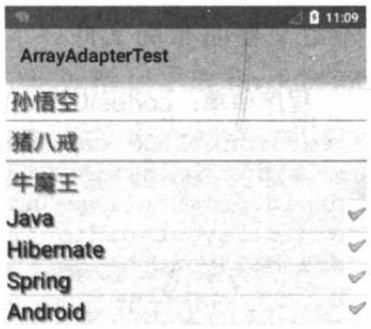


图 2.37 使用 ArrayAdatper 创建 ListView

实例：基于 ListActivity 实现列表

如果程序的窗口仅仅需要显示一个列表，则可以直接让 Activity 继承 ListActivity 来实现，ListActivity 的子类无须调用 setContentView()方法来显示某个界面，而是可以直接传入一个内容 Adapter，ListActivity 的子类就呈现出一个列表。

例如如下程序。

程序清单：codes\02\2.5\ListActivityTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends ListActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 无须使用布局文件
        String[] arr = { "孙悟空", "猪八戒", "唐僧" };
        // 创建 ArrayAdapter 对象
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_multiple_choice, arr);
        // 设置该窗口显示列表
        setListAdapter(adapter);
    }
}
```

上面程序的 Activity 继承了 ListActivity，ListActivity 无须界面布局文件——相当于它的布

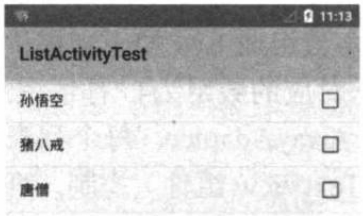


图 2.38　基于 ListActivity 实现的列表

局文件中只有一个 ListView，因此只要为 ListActivity 设置 Adapter 即可。上面程序使用 Android 提供的 R.layout.simple_list_item_multiple_choice 布局文件作为列表项组件。

运行上面的程序，将看到如图 2.38 所示的界面。

ListActivity 的默认布局是由一个位于屏幕中心的列表组成的。

实例：使用 SimpleAdapter 创建 ListView

通过 ArrayAdapter 实现 Adapter 虽然简单、易用，但 ArrayAdapter 的功能比较有限，它的每个列表项只能是 TextView。如果开发者需要实现更复杂的列表项，则可以考虑使用 SimpleAdapter。

不要被 SimpleAdapter 的名字欺骗了，SimpleAdapter 并不简单，而且它的功能非常强大。ListView 的大部分应用场景，都可以通过 SimpleAdapter 来提供列表项。

下面先定义界面布局文件。

程序清单：codes\02\2.5\SimpleAdapterTest\app\src\main\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="match_parent"
    android:layout_height="wrap_content">
<!-- 定义一个 ListView -->
<ListView android:id="@+id/mylist"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

上面的界面布局文件中仅定义了一个 ListView，该 ListView 将会显示由 SimpleAdatper 提供的列表项。

下面是 Activity 代码。

程序清单：codes\02\2.5\SimpleAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    private String[] names = new String[]
        { "虎头", "弄玉", "李清照", "李白"};
    private String[] descs = new String[]
        { "可爱的小孩", "一个擅长音乐的女孩"
            , "一个擅长文学的女性", "浪漫主义诗人"};
    private int[] imageIds = new int[]
        { R.drawable.tiger , R.drawable.nongyu
        , R.drawable.qingzhao , R.drawable.libai};
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 List 集合，List 集合的元素是 Map
        List<Map<String, Object>> listItems =
                new ArrayList<Map<String, Object>>();
        for (int i = 0; i < names.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
```

```
            listItem.put("header", imageIds[i]);
            listItem.put("personName", names[i]);
            listItem.put("desc", descs[i]);
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this, listItems,
            R.layout.simple_item,
            new String[] { "personName", "header" , "desc"},
            new int[] { R.id.name, R.id.header , R.id.desc });
        ListView list = (ListView) findViewById(R.id.mylist);
        // 为 ListView 设置 Adapter
        list.setAdapter(simpleAdapter);
    }
}
```

上面程序的关键在于粗体字代码，粗体字代码创建了一个 SimpleAdapter。使用 SimpleAdapter 的最大难点在于创建 SimpleAdapter 对象，它需要 5 个参数，其中后面 4 个参数十分关键。

- **第 2 个参数**：该参数应该是一个 List<? extends Map<String, ?>>类型的集合对象，该集合中每个 Map<String, ?>对象生成一个列表项。
- **第 3 个参数**：该参数指定一个界面布局的 ID。例如，此处指定了 R.layout.simple_ item，这意味着使用 app\src\main\res\layout\simple_item.xml 文件作为列表项组件。
- **第 4 个参数**：该参数应该是一个 String[]类型的参数，该参数决定提取 Map<String, ?>对象中哪些 key 对应的 value 来生成列表项。
- **第 5 个参数**：该参数应该是一个 int[]类型的参数，该参数决定填充哪些组件。

从上面的程序看，listItems 是一个长度为 4 的集合，这意味着它生成的 ListView 将会包含 4 个列表项，每个列表项都是 R.layout.simple_item 对应的组件（也就是一个 LinearLayout 组件）。LinearLayout 中包含了 3 个组件：ID 为 R.id.header 的 ImageView 组件、ID 为 R.id.name 和 R.id.desc 的 TextView 组件，这些组件的内容由 listItems 集合提供。

R.layout.simple_item 对应的布局文件代码如下。

程序清单：codes\02\2.5\SimpleAdapterTest\app\src\main\res\layout\simple_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<!-- 定义一个 ImageView，用于作为列表项的一部分 -->
<ImageView android:id="@+id/header"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="10dp"/>
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<!-- 定义一个 TextView，用于作为列表项的一部分 -->
<TextView android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20dp"
    android:textColor="#f0f"
    android:paddingLeft="10dp"/>
<!-- 定义一个 TextView，用于作为列表项的一部分 -->
<TextView android:id="@+id/desc"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="14dp"
    android:paddingLeft="10dp"/>
</LinearLayout>
</LinearLayout>
```

举例来说，上面 SimpleAdapter 将会生成 4 个列表项，其中第一个列表项组件是一个 LinearLayout 组件，第一个列表项的数据是{"personName"="虎头", "header"=R.id.tiger, "desc"="可爱的小孩"}Map 集合。创建 SimpleAdapter 时第 5 个参数、第 4 个参数指定使用 ID 为 R.id.name 组件显示 personName 对应的值，使用 ID 为 R.id.header 组件显示 header 对应的值，使用 ID 为 R.id.desc 的组件显示 desc 对应的值，这样第一个列表项组件所包含的三个组件都有了显示内容。后面的每个列表项依此类推。

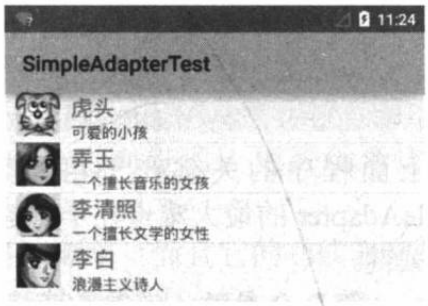


图 2.39　使用 SimpleAdapter 创建 ListView

运行上面的程序，将看到如图 2.39 所示的界面。

SimpleAdapter 同样可作为 ListActivity 的内容 Adapter，这样可以让用户方便地定制 ListActivity 所显示的列表项。

如果需要监听用户单击、选中某个列表项的事件，则可以通过 AdapterView 的 setOnItemClickListener()方法为单击事件添加监听器，或者通过 setOnItemSelectedListener()方法为列表项的选中事件添加监听器。

例如，可以在上面的 Activity 中为 ListView 通过如下代码绑定事件监听器。

程序清单：codes\02\2.5\SimpleAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
// 为 ListView 的列表项的单击事件绑定事件监听器
list.setOnItemClickListener(new OnItemClickListener()
{
    // 第 position 项被单击时激发该方法
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
            int position, long id)
    {
        System.out.println(names[position]
            + "被单击了");
    }
});
// 为 ListView 的列表项的选中事件绑定事件监听器
list.setOnItemSelectedListener(new OnItemSelectedListener()
{
    // 第 position 项被选中时激发该方法
    @Override
    public void onItemSelected(AdapterView<?> parent, View view,
            int position, long id)
    {
        System.out.println(names[position]
                + "被选中了");
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent)
    {
    }
});
```

再次运行上面程序，如果单击列表项或选中列表项，将可以看到 LogCat 控制台有如图 2.40 所示的输出。

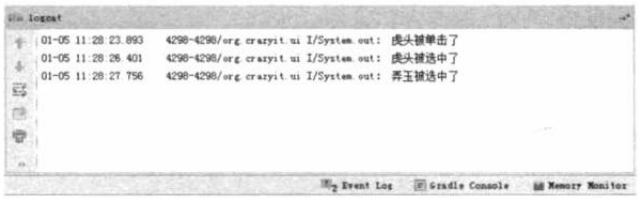


图 2.40 ListView 的事件监听

注意：

上面的程序为 ListView 的列表项单击事件、列表项选中事件绑定了事件监听器，但事件监听器只是简单地在 LogCat 控制台输出一行内容。在实际项目中我们可以在事件处理方法中做“任何”事情。不仅如此，上面绑定事件监听器的 setOnItemClickListener、setOnItemSelectedListener 方法都来自于 AdapterView，因此这种事件处理机制完全适用于 AdapterView 的其他子类。

实例：扩展 BaseAdapter 实现不存储列表项的 ListView

本实例将会通过扩展 BaseAdapter 来实现 Adapter，扩展 BaseAdapter 可以取得对 Adapter 最大的控制权：程序要创建多少个列表项，每个列表项的组件都由开发者来决定。

本实例的布局文件非常简单，布局文件中只包含一个简单的 ListView，布局文件代码如下。

程序清单：codes\02\2.5\BaseAdapterTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<ListView
    android:id="@+id/myList"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
```

该实例的 Activity 将会扩展 BaseAdapter 来实现 Adapter 对象，Activity 代码如下。

程序清单：codes\02\2.5\BaseAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    ListView myList;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myList = (ListView) findViewById(R.id.myList);
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                // 指定一共包含 40 个选项
                return 40;
            }
            @Override
            public Object getItem(int position)
            {
                return null;
```

```
            }
            // 重写该方法，该方法的返回值将作为列表项的 ID
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            // 重写该方法，该方法返回的 View 将作为列表框
            @Override
            public View getView(int position
                    , View convertView , ViewGroup parent)
            {
                // 创建一个 LinearLayout，并向其中添加两个组件
                LinearLayout line = new LinearLayout(MainActivity.this);
                line.setOrientation(0);
                ImageView image = new ImageView(MainActivity.this);
                image.setImageResource(R.drawable.ic_launcher);
                TextView text = new TextView(MainActivity.this);
                text.setText("第" + (position +1 ) + "个列表项");
                text.setTextSize(20);
                text.setTextColor(Color.RED);
                line.addView(image);
                line.addView(text);
                // 返回 LinearLayout 实例
                return line;
            }
        };
        myList.setAdapter(adapter);
    }
}
```

上面程序中的关键在于粗体字代码部分，粗体字代码创建了一个 BaseAdapter 对象，扩展该对象需要重写如下 4 个方法。

- getCount()：该方法的返回值控制该 Adapter 将会包含多少个列表项。
- getItem(int position)：该方法的返回值决定第 position 处的列表项的内容。
- getItemId(int position)：该方法的返回值决定第 position 处的列表项的 ID。
- getView(int position, View convertView , ViewGroup parent)：该方法的返回值决定第 position 处的列表项组件。

图 2.41　不存储列表项的 ListView

上面 4 个方法中最重要的是第 1 个与第 4 个。

运行上面的程序，可以看到如图 2.41 所示的效果。

到目前为止，我们已经通过 ListView 介绍了 4 种实现 Adapter 的方式。从表面上看，此处只是在介绍 ListView，但实际上这里介绍的知识完全适用于 AdapterView 的其他子类：GridView、Spinner、Gallery、AdapterViewFlipper 等。因此，后面介绍这些组件的步骤依然是如下两步。

1. 采用 4 种方式之一创建 Adapter。
2. 调用 AdapterView 的 setAdapter(Adapter)方法设置 Adapter 即可。

2.5.3　自动完成文本框（AutoCompleteTextView）的功能与用法

自动完成文本框（AutoCompleteTextView）从 EditText 派生而出，实际上它也是一个文本

编辑框，但它比普通编辑框多了一个功能：当用户输入一定字符之后，自动完成文本框会显示一个下拉菜单，供用户从中选择，当用户选择某个菜单项之后，AutoCompleteTextView 按用户选择自动填写该文本框。

AutoCompleteTextView 除了可使用 EditText 提供的 XML 属性和方法之外，还支持如表 2.20 所示的常用 XML 属性及相关方法。

表 2.21 AutoCompleteTextView 支持的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:completionHint	setCompletionHint(CharSequence)	设置下拉菜单中的提示标题
android:completionHintView		设置下拉菜单中提示标题的视图
android:completionThreshold	setThreshold(int)	设置用户至少输入几个字符才会显示提示
android:dropDownAnchor	setDropDownAnchor(int)	设置下拉菜单的定位“锚点”组件，如果没有指定该属性，将使用该 TextView 本身作为定位“锚点”组件
android:dropDownHeight	setDropDownHeight(int)	设置下拉菜单的高度
android:dropDownHorizontalOffset		设置下拉菜单与文本框之间的水平偏移。下拉菜单默认与文本框左对齐
android:dropDownVerticalOffset		设置下拉菜单与文本框之间的垂直偏移。下拉菜单默认紧跟文本框
android:dropDownWidth	setDropDownWidth(int)	设置下拉菜单的宽度
android:popupBackground	setDropDownBackgroundResource(int)	设置下拉菜单的背景

使用 AutoCompleteTextView 很简单，只要为它设置一个 Adapter 即可，该 Adapter 封装了 AutoCompleteTextView 预设的提示文本。

AutoCompleteTextView 还派生了一个子类：MultiAutoCompleteTextView，该子类的功能与 AutoCompleteTextView 基本相似，只是 MultiAutoCompleteTextView 允许输入多个提示项，多个提示项以分隔符分隔。MultiAutoCompleteTextView 提供了 setTokenizer()方法来设置分隔符。

下面的界面布局文件中包含了 AutoCompleteTextView 和 MultiAutoCompleteTextView。

程序清单：codes\02\2.5\AutoCompleteTextViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 定义一个自动完成文本框，
    指定输入一个字符后进行提示 -->
<AutoCompleteTextView
    android:id="@+id/auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:completionHint="请选择您喜欢的图书："
    android:dropDownHorizontalOffset="10dp"
    android:completionThreshold="1"/>
<!-- 定义一个 MultiAutoCompleteTextView 组件 -->
<MultiAutoCompleteTextView
    android:id="@+id/mauto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:completionThreshold="1"/>
</LinearLayout>
```

上面的界面布局文件中定义了 AutoCompleteTextView 和 MultiAutoCompleteTextView，接下来在程序中为它们绑定同一个 Adapter，这意味着两个自动完成文本框的提示项完全相同，

只是它们的表现行为略有差异。

Adapter 负责为它提供提示文本。主程序如下。

程序清单：codes\02\2.5\AutoCompleteTextViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    AutoCompleteTextView actv;
    MultiAutoCompleteTextView mauto;
    // 定义字符串数组，作为提示的文本
    String[] books = new String[]{
        "疯狂 Java 讲义",
        "疯狂 Ajax 讲义",
        "疯狂 XML 讲义",
        "疯狂 Workflow 讲义"
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 ArrayAdapter，封装数组
        ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, books);
        actv = (AutoCompleteTextView)findViewById(R.id.auto);
        // 设置 Adapter
        actv.setAdapter(aa);
        mauto = (MultiAutoCompleteTextView)findViewById(R.id.mauto);
        // 设置 Adapter
        mauto.setAdapter(aa);
        // 为 MultiAutoCompleteTextView 设置分隔符
        mauto.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
    }
}
```

上面程序中的粗体字代码负责为 AutoCompleteTextView、MultiAutoCompleteTextView 设置同一个 Adapter，并为 MultiAutoCompleteTextView 设置了分隔符。

运行上面的程序，将看到如图 2.42 所示的界面。

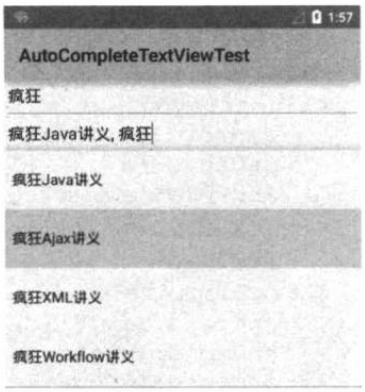


图 2.42 自动完成文本框

2.5.4 网格视图（GridView）的功能与用法

GridView 用于在界面上按行、列分布的方式来显示多个组件。

GridView 和 ListView 有共同的父类：AbsListView，因此 GridView 和 ListView 具有很高的相似性，它们都是列表项。GridView 与 ListView 的唯一区别在于：ListView 只显示一列；而 GridView 可以显示多列。从这个角度来看，ListView 相当于一种特殊的 GridView，如果让 GridView 只显示一列，那么该 GridView 就变成了 ListView。

与 ListView 类似的是，GridView 也需要通过 Adapter 来提供显示的数据：开发者可以采用上面介绍的 4 种方式中的任意一种来创建 Adapter。不管使用哪种方式，GridView 与 ListView 的用法是基本一致的。

GridView 提供的常用 XML 属性及相关方法如表 2.22 所示。

表 2.22 GridView 的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:columnWidth	setColumnWidth(int)	设置列的宽度
android:gravity	setGravity(int)	设置对齐方式
android:horizontalSpacing	setHorizontalSpacing(int)	设置各元素之间的水平间距
android:numColumns	setNumColumns(int)	设置列数
android:stretchMode	setStretchMode(int)	设置拉伸模式
android:verticalSpacing	setVerticalSpacing(int)	设置各元素之间的垂直间距

注意：

使用 GridView 时一般都应该指定 numColumns 大于 1；否则该属性的默认值为 1。如果将该属性设为 1，则意味着该 GridView 只有一列，那么 GridView 就变成了 ListView。

表 2.22 所示的 android:stretchMode 属性支持如下几个属性值。

- NO_STRETCH：不拉伸。
- STRETCH_SPACING：仅拉伸元素之间的间距。
- STRETCH_SPACING_UNIFORM：表格元素本身、元素之间的间距一起拉伸。
- STRETCH_COLUMN_WIDTH：仅拉伸表格元素本身。

下面通过一个实例来介绍 GridView 的用法，本实例采用 SimpleAdapter 为 GridView 提供数据。

实例：带预览的图片浏览器

本实例将会采用一个 GridView 以行、列的形式来组织所有图片的预览视图。然后程序用一个 ImageView 来显示图片。

下面是本实例所使用的界面布局文件。

程序清单：codes\02\2.5\GridViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<!-- 定义一个 GridView 组件 -->
<GridView
    android:id="@+id/grid01"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:horizontalSpacing="1pt"
    android:verticalSpacing="1pt"
    android:numColumns="4"
    android:gravity="center"/>
<!-- 定义一个 ImageView 组件 -->
<ImageView android:id="@+id/imageView"
    android:layout_width="240dp"
    android:layout_height="240dp"
    android:layout_gravity="center_horizontal"/>
</LinearLayout>
```

上面的界面布局文件中只是简单地定义了一个 GridView、一个 ImageView。定义 GridView

时指定了 android:numColumns="4"，这意味着该网格包含 4 列。那么该 GridView 包含多少行呢？这是动态改变的——正如 ListView 到底包含多少行是由该 ListView 对应的 Adapter 所决定的，GridView 到底包含多少行也是由 Adapter 决定的。

下面是主程序代码。

程序清单：codes\02\2.5\GridViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    GridView grid;
    ImageView imageView;
    int[] imageIds = new int[]
    {
        R.drawable.bomb5 , R.drawable.bomb6 , R.drawable.bomb7
        , R.drawable.bomb8 , R.drawable.bomb9 , R.drawable.bomb10
        , R.drawable.bomb11 , R.drawable.bomb12    , R.drawable.bomb13
        , R.drawable.bomb14 , R.drawable.bomb15 , R.drawable.bomb16
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 List 对象，List 对象的元素是 Map
        List<Map<String, Object>> listItems =
                new ArrayList<Map<String, Object>>();
        for (int i = 0; i < imageIds.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
            listItem.put("image", imageIds[i]);
            listItems.add(listItem);
        }
        // 获取显示图片的 ImageView
        imageView = (ImageView) findViewById(R.id.imageView);
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this,
                listItems
                // 使用/layout/cell.xml 文件作为界面布局
                , R.layout.cell, new String[] { "image" },
                new int[] { R.id.image1 });
        grid = (GridView) findViewById(R.id.grid01);
        // 为 GridView 设置 Adapter
        grid.setAdapter(simpleAdapter);
        // 添加列表项被选中的监听器
        grid.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
                int position, long id)
            {
                // 显示当前被选中的图片
                imageView.setImageResource(imageIds[position]);
            }
            @Override
            public void onNothingSelected(AdapterView<?> parent)
            {
            }
        });
        // 添加列表项被单击的监听器
        grid.setOnItemClickListener(new AdapterView.OnItemClickListener()
        {
            @Override
```

```
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                // 显示被单击的图片
                imageView.setImageResource(imageIds[position]);
            }
        });
    }
}
```

上面的程序同样使用了 SimpleAdapter 对象作为 GridView 的 Adapter，这个 SimpleAdapter 底层保证了一个长度为 16 的 List 集合——这意味着该 GridView 一共需要显示 16 个组件，GridView 总共有 4 列，因此该 GridView 一共包含 4 行。

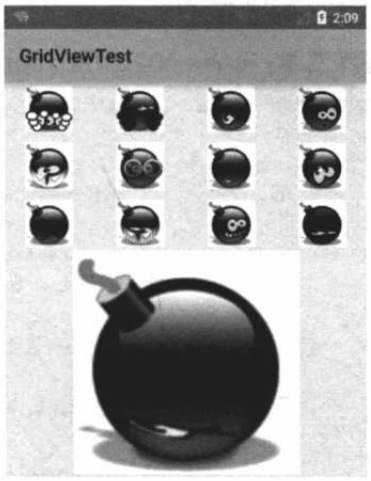


图 2.43 GridView 效果

上面的粗体字代码创建 SimpleAdapter 时指定使用 R.layout.cell 作为界面布局，因此还需要在 app\src\main\res\layout 目录下定义一个 cell.xml 界面布局文件。该文件中只包含一个简单的 ImageView 组件，此处不再给出该界面布局文件的代码。

运行上面的程序，将可以看到程序界面上方显示了 16 个图片预览视图（由 GridView 提供支持），单击任何一个图片预览视图，下面的 ImageView 将会显示对应的图片，如图 2.43 所示。

➤➤ 2.5.5 可展开的列表组件（ExpandableListView）

ExpandableListView 是 ListView 的子类，它在普通 ListView 的基础上进行了扩展，它把应用中的列表项分为几组，每组里又可包含多个列表项。

ExpandableListView 的用法与普通 ListView 的用法非常相似，只是 ExpandableListView 所显示的列表项应该由 ExpandableListAdapter 提供。ExpandableListAdapter 也是一个接口，该接口下提供的类继承关系图如图 2.44 所示。

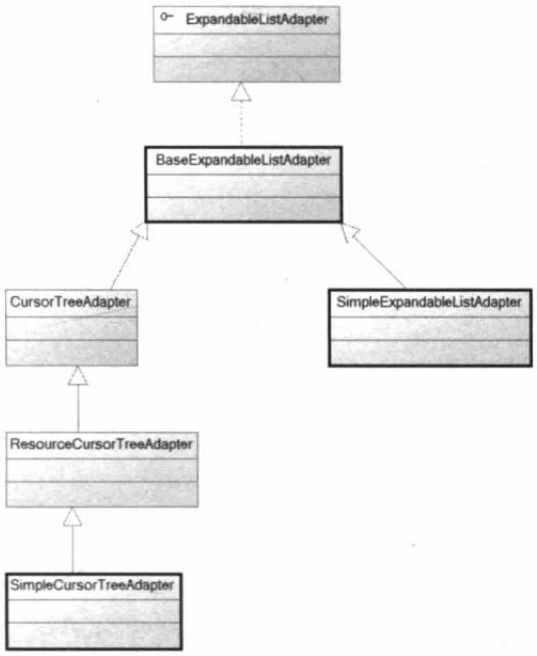


图 2.44 ExpandableListAdapter 及其子类的继承关系图

与 Adapter 类似的是，实现 ExpandableListAdapter 也有如下三种常用方式。

- 扩展 BaseExpandableListAdapter 实现 ExpandableListAdapter。
- 使用 SimpleExpandableListAdapter 将两个 List 集合包装成 ExpandableListAdapter。
- 使用 SimpleCursorTreeAdapter 将 Cursor 中的数据包装成 SimpleCursorTreeAdapter。

表 2.23 显示了 ExpandableListView 额外支持的常用 XML 属性。

表 2.23 ExpandableListView 额外支持的常用 XML 属性

XML 属性	说明
android:childDivider	指定各组内各子列表项之间的分隔条
android:childIndicator	显示在子列表项旁边的 Drawable 对象
android:groupIndicator	显示在组列表项旁边的 Drawable 对象

下面的程序示范了如何通过自定义 ExpandableListAdapter 为 ExpandableListView 提供列表项。该程序的界面布局文件非常简单，只是在 LinearLayout 内定义了一个 ExpandableListView，因此此处不再给出。

程序清单：codes\02\2.5\ExpandableListViewTest\app\src\main\java\org\crazyit\ui\MainActivity. java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 BaseExpandableListAdapter 对象
        ExpandableListAdapter adapter = new BaseExpandableListAdapter()
        {
            int[] logos = new int[]
            {
                R.drawable.p,
                R.drawable.z,
                R.drawable.t
            };
             private String[] armTypes = new String[]
                { "神族兵种", "虫族兵种", "人族兵种"};
            private String[][] arms = new String[][]
            {
                { "狂战士", "龙骑士", "黑暗圣堂", "电兵" },
                { "小狗", "刺蛇", "飞龙", "自爆飞机" },
                { "机枪兵", "护士 MM" , "幽灵" }
            };
            // 获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition, int childPosition)
            {
                return arms[groupPosition][childPosition];
            }
            @Override
            public long getChildId(int groupPosition, int childPosition)
            {
                return childPosition;
            }
            @Override
            public int getChildrenCount(int groupPosition)
            {
                return arms[groupPosition].length;
            }
            private TextView getTextView()
            {
                AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                        ViewGroup.LayoutParams.MATCH_PARENT, 64);
                TextView textView = new TextView(MainActivity.this);
                textView.setLayoutParams(lp);
                textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
                textView.setPadding(36, 0, 0, 0);
                textView.setTextSize(20);
```

```
            return textView;
        }
        // 该方法决定每个子选项的外观
        @Override
        public View getChildView(int groupPosition, int childPosition,
                boolean isLastChild, View convertView, ViewGroup parent)
        {
            TextView textView = getTextView();
            textView.setText(getChild(groupPosition, childPosition)
                    .toString());
            return textView;
        }
        // 获取指定组位置处的组数据
        @Override
        public Object getGroup(int groupPosition)
        {
            return armTypes[groupPosition];
        }
        @Override
        public int getGroupCount()
        {
            return armTypes.length;
        }
        @Override
        public long getGroupId(int groupPosition)
        {
            return groupPosition;
        }
        // 该方法决定每个组选项的外观
        @Override
        public View getGroupView(int groupPosition, boolean isExpanded,
                View convertView, ViewGroup parent)
        {
            LinearLayout ll = new LinearLayout(MainActivity.this);
            ll.setOrientation(0);
            ImageView logo = new ImageView(MainActivity.this);
            logo.setImageResource(logos[groupPosition]);
            ll.addView(logo);
            TextView textView = getTextView();
            textView.setText(getGroup(groupPosition).toString());
            ll.addView(textView);
            return ll;
        }
        @Override
        public boolean isChildSelectable(int groupPosition,
                int childPosition)
        {
            return true;
        }
        @Override
        public boolean hasStableIds()
        {
            return true;
        }
    };
    ExpandableListView expandListView = (ExpandableListView)findViewById(R.id.list);
    expandListView.setAdapter(adapter);
}
}
```

上面程序的关键代码就是扩展 BaseExpandableListAdapter 来实现 ExpandableListAdapter，当扩展 BaseExpandableListAdapter 时，关键是实现如下 4 个方法。

- getGroupCount()：该方法返回包含的组列表项的数量。

- getGroupView()：该方法返回的 View 对象将作为组列表项。
- getChildrenCount()：该方法返回特定组所包含的子列表项的数量。
- getChildView()：该方法返回的 View 对象将作为特定组、特定位置的子列表项。

上面程序中的 getChildView()方法返回了一个普通 TextView，因为每个子列表项都是一个普通文本框。而 getGroupView()方法则返回了一个 LinearLayout 对象，该 LinearLayout 对象里包含一个 ImageView 和一个 TextView。因此，每个组列表项都由图片和文本组成。

运行上面的程序，将看到如图 2.45 所示的界面。

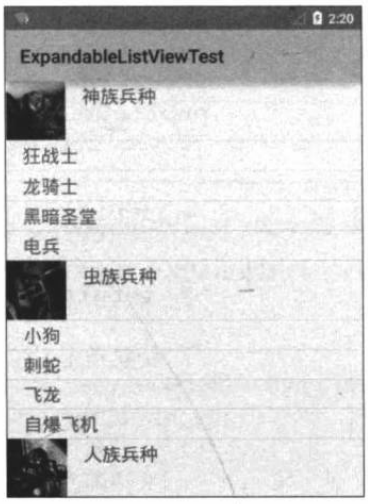


图 2.45 ExpandableListView 效果

2.5.6 Spinner 的功能与用法

Spinner 组件与 Swing 编程中的 Spinner 不同，此处的 Spinner 其实就是一个列表选择框。不过 Android 的列表选择框并不需要显示下拉列表，而是相当于弹出一个菜单供用户选择。

Spinner 与 Gallery 都继承了 AbsSpinner，AbsSpinner 继承了 AdapterView，因此它也表现出 AdapterView 的特征：只要为 AdapterView 提供 Adapter 即可。

Spinner 支持的常用 XML 属性及相关方法如表 2.24 所示。

表 2.24 Spinner 支持的常用 XML 属性及相关方法

XML 属性	相关方法	说明
android:entries		使用数组资源设置该下拉列表框的列表项
android:dropDownHorizontalOffset	setDropDownHorizontalOffset(int)	设置列表选择框的水平偏移距
android:dropDownVerticalOffset	setDropDownVerticalOffset(int)	设置列表选择框的垂直偏移距
android:dropDownWidth	setDropDownWidth(int)	设置列表选择框的宽度
android:popupBackground	android:popupBackground	设置列表选择框的背景色
android:prompt		设置该列表选择框的提示信息

注意：

android:entries 属性并不是 Spinner 定义的，而是在 AbsSpinner 中定义的，因此 Gallery（继承了 AbsSpinner）也支持该 XML 属性。

实例：让用户选择

如果开发者使用 Spinner 时已经可以确定列表选择框里的列表项，则完全不需要编写代码，只要为 Spinner 指定 android:entries 属性即可让 Spinner 正常工作；如果程序需要在运行时动态地决定 Spinner 的列表项，或者程序需要对 Spinner 的列表项进行定制，则可使用 Adapter 为 Spinner 提供列表项。

如下界面布局文件中定义了两个 Spinner 组件，其中第一个 Spinner 组件指定了 android:entries 属性，第二个 Spinner 组件没有指定 android:entries 属性，因此需要在 Activity 中为它设置 Adapter。

程序清单：codes\02\2.5\SpinnerTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 定义了一个 Spinner 组件,
    指定显示该 Spinner 组件的数组 -->
<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/books"
    android:prompt="@string/tip"/>
<Spinner
    android:id="@+id/spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:prompt="@string/tip"/>
</LinearLayout>
```

上面的界面布局文件中第一个 Spinner 使用@array/books 指定数组资源，因此需要在 app\src\main\res\value 目录下使用 XML 文件来定义一份数组资源，该数组资源文件的内容如下。

程序清单：codes\02\2.5\SpinnerTest\app\src\main\res\values\arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="books">
        <item>疯狂 Java 讲义</item>
        <item>疯狂 Ajax 讲义</item>
        <item>疯狂 XML 讲义</item>
    </string-array>
 </resources>
```

上面的界面布局文件中第二个 Spinner 没有指定 android:entries 属性，Activity 将会采用 ArrayAdapter 为该 Spinner 提供列表项。下面是 Activity 的代码。

程序清单：codes\02\2.5\SpinnerTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    Spinner spinner;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面布局文件中的 Spinner 组件
        spinner = (Spinner) findViewById(R.id.spinner);
        String[] arr = { "孙悟空", "猪八戒", "唐僧" };
        // 创建 ArrayAdapter 对象
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_multiple_choice, arr);
        // 为 Spinner 设置 Adapter
        spinner.setAdapter(adapter);
    }
}
```

上面的粗体字代码创建了一个简单的 ArrayAdapter，并调用了 Spinner 的 setAdapter(adapter)为之设置 Adpater。运行该程序，将看到如图 2.46 所示的界面。

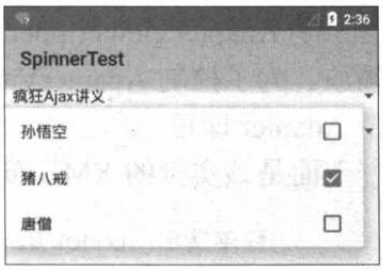


图 2.46 Spinner 的效果

Gallery 与 Spinner 组件有共同的父类：AbsSpinner，表明 Gallery 和 Spinner 都是一个列表选择框。它们之间的区别

在于，Spinner 显示的是一个垂直的列表选择框，而 Gallery 显示的是一个水平的列表选择框。Gallery 与 Spinner 还有一个区别：Spinner 的作用是供用户选择，而 Gallery 则允许用户通过拖动来查看上一个、下一个列表项。

Gallery 本身的用法非常简单——基本上与 Spinner 的用法相似，只要为它提供一个内容 Adapter 即可，该 Adapter 的 getView()方法所返回的 View 将作为 Gallery 列表的列表项。如果程序需要监控到 Gallery 选择项的改变，通过为 Gallery 添加 OnItemSelectedListener 监听器即可实现。

注意：

Android 已经不再推荐使用 Gallery 组件，而是推荐使用其他水平滚动组件如 HorizontalScrollView 和 ViewPager 来代替 Gallery 组件。故本书不再详细介绍该界面组件的用法。

2.5.7 AdapterViewFlipper 的功能与用法

AdapterViewFilpper 继承了 AdapterViewAnimator，它也会显示 Adapter 提供的多个 View 组件，但它每次只能显示一个 View 组件，程序可通过 showPrevious()和 showNext()方法控制该组件显示上一个、下一个组件。

AdapterViewFilpper 可以在多个 View 切换过程中使用渐隐渐显的动画效果。除此之外，还可以调用该组件的 startFlipping()控制它“自动播放”下一个 View 组件。

AdapterViewAnimator 支持的 XML 属性如表 2.25 所示。

表 2.25 AdapterViewAnimator 支持的 XML 属性

XML 属性	说明
android:animateFirstView	设置显示该组件的第一个 View 时是否使用动画
android:inAnimation	设置组件显示时使用的动画
android:loopViews	设置循环到最后一个组件后是否自动“转头”到第一个组件
android:outAnimation	设置组件隐藏时使用的动画

AdapterViewFilpper 额外支持的 XML 属性及相关方法如表 2.26 所示。

表 2.26 AdapterViewFilpper 支持的 XML 属性及相关方法

XML 属性	相关方法	说明
android:autoStart	startFlipping()	设置显示该组件是否自动播放
android:flipInterval	setFlipInterval(int)	设置自动播放的时间间隔

实例：自动播放的图片库

本实例示范了如何使用 AdapterViewFlipper 开发自动播放的图片库。该实例的界面上除了包含一个 AdapterViewFilpper 之外，还包含三个按钮，用于控制显示上一个、下一个图片和自动播放。为了控制 AdapterViewFilpper 要显示的多个列表项，程序为 AdapterViewFilpper 设置一个 Adapter 即可。

下面是该实例的 XML 布局文件。

程序清单：codes\02\2.5\AdapterViewFlipperTest\app\src\main\res\layout\main.xml

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

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <AdapterViewFlipper
        android:id="@+id/flipper"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:flipInterval="5000"
        android:layout_alignParentTop="true"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:onClick="prev"
        android:text="上一个"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:onClick="next"
        android:text="下一个"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:onClick="auto"
        android:text="自动播放"/>
</RelativeLayout>
```

上面的粗体字代码定义了一个 AdapterViewFlipper 组件，并为三个按钮指定了事件处理方法。该实例的 Activity 会采用扩展 BaseAdapter 的方式来实现自己的 Adapter，并为 AdapterViewFlipper 组件设置 Adapter。下面是该 Activity 的代码。

程序清单：codes\02\2.5\AdapterViewFlipperTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    int[] imageIds = new int[]
    {
        R.drawable.shuangzi, R.drawable.shuangyu,
        R.drawable.chunv, R.drawable.tiancheng, R.drawable.tianxie,
        R.drawable.sheshou, R.drawable.juxie, R.drawable.shuiping,
        R.drawable.shizi, R.drawable.baiyang, R.drawable.jinniu,
        R.drawable.mojie };
    private AdapterViewFlipper flipper;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        flipper = (AdapterViewFlipper) findViewById(R.id.flipper);
        // 创建一个BaseAdapter对象，该对象负责提供Gallery所显示的列表项
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                return imageIds.length;
            }
            @Override
            public Object getItem(int position)
```

```
            {
                return position;
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            // 该方法返回的View代表了每个列表项
            @Override
            public View getView(int position, View convertView, ViewGroup parent)
            {
                // 创建一个ImageView
                ImageView imageView = new ImageView(MainActivity.this);
                imageView.setImageResource(imageIds[position]);
                // 设置ImageView的缩放类型
                imageView.setScaleType(ImageView.ScaleType.FIT_XY);
                // 为imageView设置布局参数
                imageView.setLayoutParams(new ViewGroup.LayoutParams(
                        LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
                return imageView;
            }
        };
        flipper.setAdapter(adapter);
    }
    public void prev(View source)
    {
        // 显示上一个组件
        flipper.showPrevious();
        // 停止自动播放
        flipper.stopFlipping();
    }
    public void next(View source)
    {
        // 显示下一个组件。
        flipper.showNext();
        // 停止自动播放
        flipper.stopFlipping();
    }
    public void auto(View source)
    {
        // 开始自动播放
        flipper.startFlipping();
    }
}
```

上面程序中的粗体字代码调用了AdapterViewFlipper的showPrevious()、showNext()方法来控制该组件显示上一个、下一个组件，并调用了startFlipping()方法控制自动播放。

运行该程序，可以看到如图2.47所示的界面。

在图2.47中单击“自动播放”按钮，将可以看到AdapterViewFlipper每隔5秒更换一个图片，切换图片时会使用渐隐渐显效果。

图2.47　使用AdapterViewFlipper实现自动播放图片

2.5.8 StackView 的功能与用法

StackView 也是 AdapterViewAnimator 的子类，它也用于显示 Adapter 提供的一系列 View。StackView 将会以“堆叠（Stack）”的方式来显示多个列表项。

为了控制 StackView 显示的 View 组件，StackView 提供了如下两种控制方式。

- 拖走 StackView 中处于顶端的 View，下一个 View 将会显示出来。将上一个 View 拖进 StackView，将使之显示出来。
- 通过调用 StackView 的 showNext()、showPrevious()控制显示下一个、上一个组件。

下面的实例示范了 StackView 的功能与用法。

实例：叠在一起的图片

该实例会使用 StackView 将照片叠在一起，并允许用户通过拖动或单击按钮来显示上一个、下一个图片。该实例的布局文件如下。

程序清单：codes\02\2.5\StackViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
    <StackView
        android:id="@+id/mStackView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:loopViews="true" />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="上一个"
            android:onClick="prev"/>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="下一个"
            android:onClick="next"/>
    </LinearLayout>
</LinearLayout>
```

上面的布局文件中定义了一个 StackView 组件，该 StackView 将会以“叠”的方式显示多个 View 组件。与所有 AdapterView 类似的是，只要为 StackView 设置 Adapter 即可。

下面 Activity 将会创建一个 SimpleAdapte 作为 StackView 的 Adapter，并为布局文件中的两个按钮的 onClick 事件提供处理方法。下面是该 Activity 的代码。

程序清单：codes\02\2.5\StackViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    StackView stackView;
    int[] imageIds = new int[]
    {
        R.drawable.bomb5 , R.drawable.bomb6 , R.drawable.bomb7
        , R.drawable.bomb8 , R.drawable.bomb9 , R.drawable.bomb10
```

```
        , R.drawable.bomb11 , R.drawable.bomb12    , R.drawable.bomb13
        , R.drawable.bomb14 , R.drawable.bomb15 , R.drawable.bomb16
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        stackView = (StackView) findViewById(R.id.mStackView);
        // 创建一个 List 对象，List 对象的元素是 Map
        List<Map<String, Object>> listItems =
                new ArrayList<Map<String, Object>>();
        for (int i = 0; i < imageIds.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
            listItem.put("image", imageIds[i]);
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this,
                listItems
                // 使用/layout/cell.xml 文件作为界面布局
                , R.layout.cell, new String[] { "image" },
                new int[] { R.id.image1 });
        stackView.setAdapter(simpleAdapter);
    }
    public void prev(View view)
    {
        // 显示上一个组件
        stackView.showPrevious();
    }
    public void next(View view)
    {
        // 显示下一个组件
        stackView.showNext();
    }
}
```

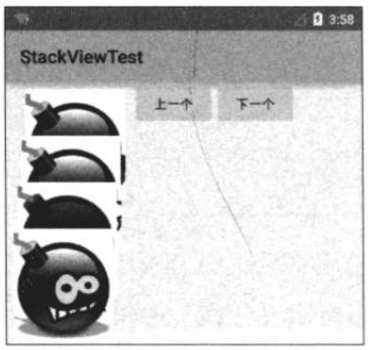


图 2.48　StackView 示例效果

上面的 Activity 代码中粗体字代码创建了一个 SimpleAdapter，并将这个 SimpleAdapter 设置为该 StackView 的 Adapter，这样该 StackView 将会显示该 SimpleAdapter 包含的一系列 View 组件。

运行该程序，并拖动 ViewStack 顶端的 View 组件，将可以看到如图 2.48 所示的效果。

图 2.48 中下方这个图片是正在被拖动的图片，这个图片显示的“动画”效果正是 StackView 支持的效果。

2.6　第 5 组 UI 组件：ProgressBar 及其子类

ProgressBar 也是一组重要的组件，ProgressBar 本身代表了进度条组件，它还派生了两个常用的组件：SeekBar 和 RatingBar。ProgressBar 及其子类在用法上十分相似，只是显示界面有一定的区别，因此本节把它们归为一类，针对它们的共性集中讲解，并突出介绍它们的区别。

ProgressBar 及其子类的继承关系图如图 2.49 所示。

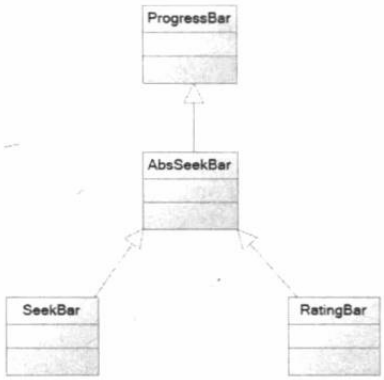


图 2.49 ProgressBar 及其子类的继承关系图

2.6.1 进度条（ProgressBar）的功能与用法

进度条也是 UI 界面中一种非常实用的组件，通常用于向用户显示某个耗时操作完成的百分比。进度条可以动态地显示进度，因此避免长时间地执行某个耗时操作时，让用户感觉程序失去了响应，从而更好地提高用户界面的友好性。

Android 支持多种风格的进度条，通过 style 属性可以为 ProgressBar 指定风格。该属性可支持如下几个属性值。

- @android:style/Widget.ProgressBar.Horizontal：水平进度条。
- @android:style/Widget.ProgressBar.Inverse：普通大小的环形进度条。
- @android:style/Widget.ProgressBar.Large：大环形进度条。
- @android:style/Widget.ProgressBar.Large.Inverse：大环形进度条。
- @android:style/Widget.ProgressBar.Small：小环形进度条。
- @android:style/Widget.ProgressBar.Small.Inverse：小环形进度条。

除此之外，ProgressBar 还支持如表 2.27 所示的常用 XML 属性。

表 2.27 ProgressBar 支持的常用 XML 属性

XML 属性	说明
android:max	设置该进度条的最大值
android:progress	设置该进度条的已完成进度值
android:progressDrawable	设置该进度条的轨道对应的 Drawable 对象
android:indeterminate	该属性设为 true，设置进度条不精确显示进度
android:indeterminateDrawable	设置绘制不显示进度的进度条的 Drawable 对象
android:indeterminateDuration	设置不精确显示进度的持续时间

表 2.27 中的 android:progressDrawable 用于指定进度条的轨道的绘制形式，该属性可指定为一个 LayerDrawable 对象（该对象可通过在 XML 文件中用<layer-list>元素进行配置）的引用。

ProgressBar 提供了如下方法来操作进度。

- setProgress(int)：设置进度的完成百分比。
- incrementProgressBy(int)：设置进度条的进度增加或减少。当参数为正数时进度增加；当参数为负数时进度减少。

下面的程序简单示范了进度条的用法。该程序的界面布局文件只是定义了几个简单的进度条，并指定 style 属性为@android:style/Widget.ProgressBar.Horizontal，即水平进度条。界面布局文件如下。

程序清单：codes\02\2.6\ProgressBarTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<!-- 定义一个大环形进度条 -->
<ProgressBar
     android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Large"/>
<!-- 定义一个中等大小的环形进度条 -->
<ProgressBar
     android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<!-- 定义一个小环形进度条 -->
<ProgressBar
     android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@android:style/Widget.ProgressBar.Small"/>
</LinearLayout>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="任务完成的进度"/>
<!-- 定义一个水平进度条 -->
<ProgressBar
    android:id="@+id/bar"
     android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    style="@android:style/Widget.ProgressBar.Horizontal"/>
<!-- 定义一个水平进度条，并改变轨道外观 -->
<ProgressBar
    android:id="@+id/bar2"
     android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progressDrawable="@drawable/my_bar"
    style="@android:style/Widget.ProgressBar.Horizontal"/>
</LinearLayout>
```

上面的界面布局文件中先定义了三个环形进度条，这种环形进度条无法显示进度，它只是显示一个不断旋转的图片。布局文件的后面定义的两个进度条的最大值为100，第一个进度条的样式为水平进度条；第二个进度条的外观被定义为@drawable/my_bar，因此还需要在drawable中定义如下文件。

程序清单：codes\02\2.6\ProgressBarTest\app\src\main\res\drawable\my_bar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 定义轨道的背景 -->
    <item android:id="@android:id/background"
        android:drawable="@drawable/no" />
    <!-- 定义轨道上已完成部分的样式 -->
    <item android:id="@android:id/progress"
        android:drawable="@drawable/ok" />
</layer-list>
```

下面的主程序用一个填充数组的任务模拟了耗时操作，并以进度条来标识任务的完成百分比。

程序清单：codes\02\2.6\ProgressBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 该程序模拟填充长度为 100 的数组
    private int[] data = new int[100];
    int hasData = 0;
    // 记录 ProgressBar 的完成进度
    int status = 0;
    ProgressBar bar , bar2;
    // 创建一个负责更新的进度的 Handler
    Handler mHandler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            // 表明消息是由该程序发送的
            if (msg.what == 0x111)
            {
                bar.setProgress(status);
                bar2.setProgress(status);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bar = (ProgressBar) findViewById(R.id.bar);
        bar2 = (ProgressBar) findViewById(R.id.bar2);
        // 启动线程来执行任务
        new Thread()
        {
            public void run()
            {
                while (status < 100)
                {
                    // 获取耗时操作的完成百分比
                    status = doWork();
                    // 发送消息
                    mHandler.sendEmptyMessage(0x111);
                }
            }
        }.start();
    }
    // 模拟一个耗时的操作
    public int doWork()
    {
        // 为数组元素赋值
        data[hasData++] = (int) (Math.random() * 100);
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
```

```
            return hasData;
        }
    }
```

上面程序中的粗体字代码用于修改进度条的完成进度。运行上面的程序，将看到如图 2.50 所示的界面。

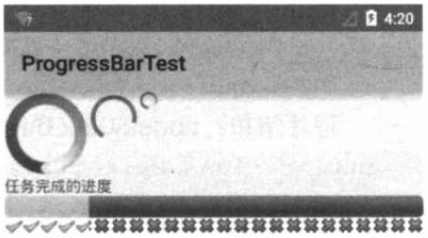


图 2.50　进度条

实例：显示在标题上的进度条

还有一种进度条，可以直接在窗口标题上显示，这种进度条甚至不需要使用 ProgressBar 组件，它是直接由 Activity 的方法启用的。为了在窗口标题上显示进度条，需要经过如下两步。

① 调用 Activity 的 requestWindowFeature()方法，该方法根据传入的参数可启用特定的窗口特征。例如，传入 Window.FEATURE_INDETERMINATE_PROGRESS 在窗口标题上显示不带进度的进度条；传入 Window.FEATURE_PROGRESS 则显示带进度的进度条。

② 调用 Activity 的 setProgressBarVisibility(boolean)或 setProgressBarIndeterminate Visibility(boolean)方法即可控制进度条的显示和隐藏。

本实例的界面布局文件中仅仅定义了两个按钮，此处不再给出界面布局文件代码。该程序的 Activity 类代码如下。

程序清单：codes\02\2.6\TitleProgressBar\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 设置窗口特征：启用显示进度的进度条
        requestWindowFeature(Window.FEATURE_PROGRESS); // ①
        // 设置窗口特征：启用不显示进度的进度条
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS); // ②
        setContentView(R.layout.main);
        Button bn1 = (Button)findViewById(R.id.bn1);
        Button bn2 = (Button)findViewById(R.id.bn2);
        bn1.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 显示不带进度的进度条
                setProgressBarIndeterminateVisibility(true);
                // 显示带进度的进度条
                setProgressBarVisibility(true);
                // 设置进度条的进度
                setProgress(4500);
            }
        });
        bn2.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 隐藏不带进度的进度条
                setProgressBarIndeterminateVisibility(false);
                // 隐藏带进度的进度条
                setProgressBarVisibility(false);
            }
```

```
        });
    }
}
```

上面程序中的①号代码控制窗口标题上显示带进度的进度条，而②号代码则控制窗口标题上显示不带进度的进度条。程序中两个按钮主要用于控制进度条的显示、隐藏。

如果程序启动窗口标题上显示带进度的进度条，则可看到如图 2.51 所示的界面。

如果程序启动窗口标题上显示不带进度的进度条，则可看到如图 2.52 所示的界面。

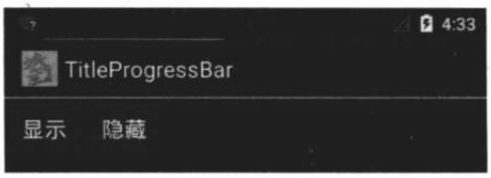


图 2.51 显示在标题上的带进度的进度条

图 2.52 显示在标题上的不带进度的进度条

提示：

可能读者已经发现，图 2.51、图 2.52 所示的界面与前面实例的界面并不相同，这是因为前面实例都使用 Android 5.0 默认的主题：Material（Android 5.0 的重要更新之一就是采用了 Material Design），而这个默认主题将看不到标题上的进度条。因此，本实例在 AndroidManifest.xml 文件中将该应用的主题设为 Holo。

2.6.2 拖动条（SeekBar）的功能与用法

拖动条和进度条非常相似，只是进度条采用颜色填充来表明进度完成的程度，而拖动条则通过滑块的位置来标识数值——而且拖动条允许用户拖动滑块来改变值，因此拖动条通常用于对系统的某种数值进行调节，比如调节音量等。

提示：

由于拖动条 SeekBar 继承了 ProgressBar，因此 ProgressBar 所支持的 XML 属性和方法完全适用于 SeekBar。

SeekBar 允许用户改变拖动条的滑块外观，改变滑块外观通过如下属性来指定。

- android:thumb：指定一个 Drawable 对象，该对象将作为自定义滑块。

为了让程序能响应拖动条滑块位置的改变，程序可以考虑为它绑定一个 OnSeekBarChangeListener 监听器。

下面通过一个实例来示范 SeekBar 的功能和用法。

实例：通过拖动滑块来改变图片的透明度

该程序的界面布局中需要两个组件：一个 ImageView 用于显示图片，一个 SeekBar 用于动态改变图片的透明度。界面布局文件如下。

程序清单：codes\02\2.6\SeekBarTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<ImageView
    android:id="@+id/image"
    android:layout_width="match_parent"
```

```
    android:layout_height="240dp"
    android:src="@drawable/lijiang"/>
<!-- 定义一个拖动条，并改变它的滑块外观 -->
<SeekBar
    android:id="@+id/seekbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="255"
    android:progress="255"
    android:thumb="@drawable/ic_launcher"/>
</LinearLayout>
```

上面程序中的粗体字代码定义了该拖动条的最大值、当前值都是 255，并通过指定 android:thumb 属性来改变拖动条上滑块的外观。

该实例的主程序比较简单，程序只要为拖动条绑定一个监听器，当滑块位置发生改变时动态改变 ImageView 的透明度即可。主程序如下。

程序清单：codes\02\2.6\SeekBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    ImageView image;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        image = (ImageView) findViewById(R.id.image);
        SeekBar seekBar = (SeekBar) findViewById(R.id.seekbar);
        seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
        {
            // 当拖动条的滑块位置发生改变时触发该方法
            @Override
            public void onProgressChanged(SeekBar arg0, int progress,
                    boolean fromUser)
            {
                // 动态改变图片的透明度
                image.setImageAlpha(progress);
            }
            @Override
            public void onStartTrackingTouch(SeekBar bar)
            {
            }
            @Override
            public void onStopTrackingTouch(SeekBar bar)
            {
            }
        });
    }
}
```

上面的粗体字代码就是监听拖动条上滑块位置发生改变的关键代码：当滑块位置发生改变时，ImageView 的透明度将变为该拖动条的当前数值。运行上面的程序，将看到如图 2.53 所示的界面。

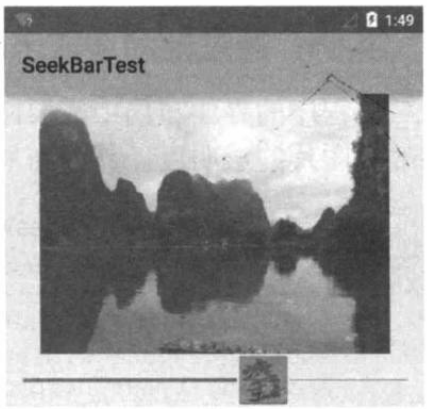


图 2.53 拖动滑块改变图片的透明度

➤➤ 2.6.3 星级评分条（RatingBar）的功能与用法

星级评分条与拖动条有相同的父类：AbsSeekBar，因

此它们十分相似。实际上星级评分条与拖动条的用法、功能都十分接近：它们都允许用户通过拖动来改变进度。RatingBar 与 SeekBar 的最大区别在于：RatingBar 通过星星来表示进度。

表 2.28 显示了 RatingBar 所支持的常见 XML 属性。

表 2.28 RatingBar 支持的常见 XML 属性

XML 属性	说明
android:isIndicator	设置该星级评分条是否允许用户改变（true 为不允许修改）
android:numStars	设置该星级评分条总共有多少个星级
android:rating	设置该星级评分条默认的星级
android:stepSize	设置每次最少需要改变多少个星级

为了让程序能响应星级评分条评分的改变，可以考虑为它绑定一个 OnRatingBarChangeListener 监听器。

下面通过一个实例来示范 RatingBar 的功能和用法。

实例：通过星级改变图片的透明度

该程序其实就是前一个程序的简单改变，只是将前一个程序中的 SeekBar 组件改为使用 RatingBar。下面是界面布局文件中关于 RatingBar 的代码片段。

程序清单：codes\02\2.6\RatingBarTest\app\src\main\res\layout\main.xml

```
<!-- 定义一个星级评分条 -->
<RatingBar
    android:id="@+id/rating"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:max="255"
    android:progress="255"
    android:stepSize="0.5"/>
```

上面的界面布局文件中指定了该星级评分条的最大值为 255，当前进度为 255——其中两个属性都来自于 ProgressBar 组件，这没有任何问题，因为 RatingBar 本来就是一个特殊的 ProgressBar。

主程序只要为 RatingBar 绑定事件监听器，即可监听星级评分条的星级改变。下面的主程序为星级评分条绑定监听器的代码。

程序清单：codes\02\2.6\RatingBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
RatingBar ratingBar = (RatingBar) findViewById(R.id.rating);
ratingBar.setOnRatingBarChangeListener(new OnRatingBarChangeListener()
{
    // 当星级评分条的评分发生改变时触发该方法
    @Override
    public void onRatingChanged(RatingBar arg0, float rating,
            boolean fromUser)
    {
        // 动态改变图片的透明度，其中 255 是星级评分条的最大值
        // 5 个星星就代表最大值 255
        image.setImageAlpha((int) (rating * 255 / 5));
    }
});
```

由于上面定义 RatingBar 时指定了 android:stepSize="0.5"，因此该星级评分条中星级的最小变化为 0.5，也就是最少要变化半个星级。运行上面的程序，将看到如图 2.54 所示的界面。

图 2.54 星级评分条改变图片的透明度

2.7 第 6 组 UI 组件：ViewAnimator 及其子类

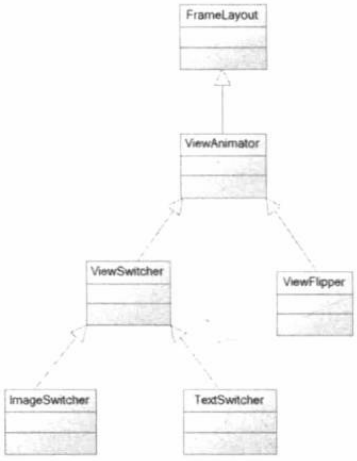


图 2.55 ViewAnimator 及其子类的继承关系图

ViewAnimator 是一个基类，它继承了 FrameLayout，因此它表现出 FrameLayout 的特征，可以将多个 View 组件“叠”在一起。ViewAnimator 额外增加的功能正如它的名字所暗示的，ViewAnimator 可以在 View 切换时表现出动画效果。

ViewAnimator 及其子类的继承关系图如图 2.55 所示。

ViewAnimator 及其子类也是一组非常重要的 UI 组件，这种组件的主要功能是增加动画效果，从而使界面更加“炫”。使用 ViewAnimator 时可以指定如表 2.29 所示的常见 XML 属性。

表 2.29 ViewAnimator 支持的常见 XML 属性

XML 属性	说明
android:animateFirstView	设置 ViewAnimator 显示第一个 View 组件时是否使用动画
android:inAnimation	设置 ViewAnimator 显示组件时所使用的动画
android:outAnimation	设置 ViewAnimator 隐藏组件时所使用的动画

在实际项目中往往会使用 ViewAnimator 的几个子类，下面一一详细介绍。

2.7.1 ViewSwitcher 的功能与用法

ViewSwitcher 代表了视图切换组件，它本身继承了 FrameLayout，因此可以将多个 View 层叠在一起，每次只显示一个组件。当程序控制从一个 View 切换到另一个 View 时，ViewSwitcher 支持指定动画效果。

为了给 ViewSwitcher 添加多个组件，一般通过调用 ViewSwitcher 的 setFactory(ViewSwitcher.ViewFactory)方法为之设置 ViewFactory，并由该 ViewFactory 为之创建 View 即可。

下面通过一个实例来介绍 ViewFactory 的用法。

实例：仿 Android 系统 Launcher 界面

Android 早期版本的 Launcher 界面是上下滚动的，Android 5.0 的 Launcher 界面已经实现

了分屏、左右滚动（可能是模仿 iOS 的操作习惯吧），本实例就是通过 ViewSwitcher 来实现 Android 5.0 的分屏、左右滚动效果。

为了实现该效果，程序主界面考虑使用 ViewSwitcher 来组合多个 GridView，每个 GridView 代表一个屏幕的应用程序，GridView 中每个单元格显示一个应用程序的图标和程序名。

该程序的主界面布局文件如下。

程序清单：codes\02\2.7\ViewSwitcherTest\app\src\main\res\layout\main.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- 定义一个 ViewSwitcher 组件 -->
    <ViewSwitcher
        android:id="@+id/viewSwitcher"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <!-- 定义滚动到上一屏的按钮 -->
    <Button
        android:id="@+id/button_prev"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:onClick="prev"
        android:text="&lt;" />
    <!-- 定义滚动到下一屏的按钮 -->
    <Button
        android:id="@+id/button_next"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:onClick="next"
        android:text="&gt;" />
</RelativeLayout>
```

上面的界面布局文件中只是定义了一个 ViewSwitcher 组件和两个按钮，这两个按钮分别用于控制该 ViewSwitcher 显示上一屏、下一屏的程序列表。

提示：

这个程序采用了按钮来控制滚动到上一屏、下一屏，这种操作方式显得不够“酷”，实际上完全可以实现通过手势来滚动屏幕。如果希望实现通过手势来滚动屏幕的效果，建议参考本书第 8 章关于手势的知识。

该实例的重点在于为该 ViewSwitcher 设置 ViewFactory 对象，并且当用户单击“<”和“>”两个按钮时控制 ViewSwitcher 显示“上一屏”和“下一屏”的应用程序。

该程序会考虑使用扩展 BaseAdapter 的方式为 GridView 提供 Adapter，而本实例的关键就是根据用户单击的按钮来动态计算该 BaseAdapter 应该显示哪些程序列表。该程序的 Activity 代码如下。

程序清单：codes\02\2.7\ViewSwitcherTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义一个常量，用于显示每屏显示的应用程序数
    public static final int NUMBER_PER_SCREEN = 12;
```

```
// 代表应用程序的内部类
public static class DataItem
{
    // 应用程序名称
    public String dataName;
    // 应用程序图标
    public Drawable drawable;
}
// 保存系统所有应用程序的 List 集合
private ArrayList<DataItem> items = new ArrayList<DataItem>();
// 记录当前正在显示第几屏的程序
private int screenNo = -1;
// 保存程序所占的总屏数
private int screenCount;
ViewSwitcher switcher;
// 创建 LayoutInflater 对象
LayoutInflater inflater;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    inflater = LayoutInflater.from(MainActivity.this);
    // 创建一个包含 40 个元素的 List 集合，用于模拟包含 40 个应用程序
    for (int i = 0; i < 40; i++)
    {
        String label = "" + i;
        Drawable drawable = getResources().getDrawable(
                R.drawable.ic_launcher);
        DataItem item = new DataItem();
        item.dataName = label;
        item.drawable = drawable;
        items.add(item);
    }
    // 计算应用程序所占的总屏数
    // 如果应用程序的数量能整除 NUMBER_PER_SCREEN，除法的结果就是总屏数
    // 如果不能整除，总屏数应该是除法的结果再加 1
    screenCount = items.size() % NUMBER_PER_SCREEN == 0 ?
            items.size()/ NUMBER_PER_SCREEN :
            items.size() / NUMBER_PER_SCREEN    + 1;
    switcher = (ViewSwitcher) findViewById(R.id.viewSwitcher);
    switcher.setFactory(new ViewSwitcher.ViewFactory()
    {
        // 实际上就是返回一个 GridView 组件
        @Override
        public View makeView()
        {
            // 加载 R.layout.slidelistview 组件，实际上就是一个 GridView 组件
            return inflater.inflate(R.layout.slidelistview, null);
        }
    });
    // 页面加载时先显示第一屏
    next(null);
}
public void next(View v)
{
    if (screenNo < screenCount - 1)
    {
        screenNo++;
        // 为 ViewSwitcher 的组件显示过程设置动画
        switcher.setInAnimation(this, R.anim.slide_in_right);
        // 为 ViewSwitcher 的组件隐藏过程设置动画
        switcher.setOutAnimation(this, R.anim.slide_out_left);
```

```
            // 控制下一屏将要显示的 GridView 对应的 Adapter
            ((GridView) switcher.getNextView()).setAdapter(adapter);
            // 单击右边按钮，显示下一屏
            // 学习手势检测后，也可通过手势检测实现显示下一屏
            switcher.showNext();  // ①
        }
    }
    public void prev(View v)
    {
        if (screenNo > 0)
        {
            screenNo--;
            // 为 ViewSwitcher 的组件显示过程设置动画
            switcher.setInAnimation(this, android.R.anim.slide_in_left);
            // 为 ViewSwitcher 的组件隐藏过程设置动画
            switcher.setOutAnimation(this, android.R.anim.slide_out_right);
            // 控制下一屏将要显示的 GridView 对应的 Adapter
            ((GridView) switcher.getNextView()).setAdapter(adapter);
            // 单击左边按钮，显示上一屏，当然可以采用手势
            // 学习手势检测后，也可通过手势检测实现显示上一屏
            switcher.showPrevious();  //②
        }
    }
    // 该 BaseAdapter 负责为每屏显示的 GridView 提供列表项
    private BaseAdapter adapter = new BaseAdapter()
    {
        @Override
        public int getCount()
        {
            // 如果已经到了最后一屏，且应用程序的数量不能整除 NUMBER_PER_SCREEN
            if (screenNo == screenCount - 1
                    && items.size() % NUMBER_PER_SCREEN != 0)
            {
                // 最后一屏显示的程序数为应用程序的数量对 NUMBER_PER_SCREEN 求余
                return items.size() % NUMBER_PER_SCREEN;
            }
            // 否则每屏显示的程序数量为 NUMBER_PER_SCREEN
            return NUMBER_PER_SCREEN;
        }
        @Override
        public DataItem getItem(int position)
        {
            // 根据 screenNo 计算第 position 个列表项的数据
            return items.get(screenNo * NUMBER_PER_SCREEN + position);
        }
        @Override
        public long getItemId(int position)
        {
            return position;
        }
        @Override
        public View getView(int position
                , View convertView, ViewGroup parent)
        {
            View view = convertView;
            if (convertView == null)
            {
                // 加载 R.layout.labelicon 布局文件
                view = inflater.inflate(R.layout.labelicon, null);
            }
            // 获取 R.layout.labelicon 布局文件中的 ImageView 组件，并为之设置图标
            ImageView imageView = (ImageView)
```

```
                view.findViewById(R.id.imageview);
            imageView.setImageDrawable(getItem(position).drawable);
            // 获取 R.layout.labelicon 布局文件中的 TextView 组件，并为之设置文本
            TextView textView = (TextView)
                view.findViewById(R.id.textview);
            textView.setText(getItem(position).dataName);
            return view;
        }
    };
}
```

上面的程序使用 screenNo 保存当前正在显示第几屏的程序列表。该程序的关键在于粗体字代码部分，该粗体字代码创建了一个 BaseAdapter 对象，这个 BaseAdapter 会根据 screenNo 动态计算该 Adapter 总共包含多少个列表项（如 getCount()方法所示），会根据 screenNo 计算每个列表项的数据（如 getItem(int position)方法所示）。

BaseAdapter 的 getView()只是简单加载了 R.layout.labelicon 布局文件，并使用当前列表项的图片数据填充 R.layout.labelicon 布局文件中的 ImageView，使用当前列表项的文本数据填充 R.layout.labelicon 布局文件中的 TextView。下面是 R.layout.labelicon 布局文件的代码。

程序清单：codes\02\2.7\ViewSwitcherTest\app\src\main\res\layout\labelicon.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个垂直的 LinearLayout，该容器中放置一个 ImageView 和一个 TextView -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">
    <ImageView
        android:id="@+id/imageview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"/>
</LinearLayout>
```

当用户单击“>”按钮时，程序的事件处理方法将会控制 ViewSwitcher 调用 showNext()方法显示下一屏的程序列表——而且此时 screenNo 被加 1，因而 Adapter 将会动态计算下一屏的程序列表，再将该 Adapter 传给 ViewSwitcher 接下来要显示的 GridView。

为了实现 ViewSwitcher 切换 View 时的动画效果，程序的事件处理方法中调用了 ViewSwitcher 的 setInAnimation()、setOutAnimation()方法来设置动画效果。本程序不仅利用了 Android 系统提供的两个动画资源，还自行提供了动画资源。

其中 R.anim.slide_in_right 动画资源对应的代码如下。

程序清单：codes\02\2.7\ViewSwitcherTest\app\src\main\res\anim\slide_in_right.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置从右边拖进来的动画，
    android:duration 指定动画持续时间 -->
    <translate
        android:fromXDelta="100%p"
        android:toXDelta="0"
        android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

R.anim.slide_out_left 动画资源对应的代码如下。

程序清单：codes\02\2.7\ViewSwitcherTest\app\src\main\res\anim\slide_out_left.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置从左边拖出去的动画，
    android:duration 指定动画持续时间 -->
    <translate
        android:fromXDelta="0"
        android:toXDelta="-100%p"
        android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

运行上面的程序，可以看到如图 2.56 所示的效果。

在图 2.56 所示界面中，当用户单击底端的“<”或“>”按钮时，将可以看到 ViewSwitcher 切换屏幕时的动画效果。

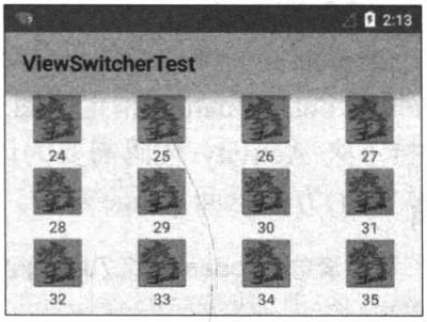


图 2.56 仿 Android 系统的 Launcher 界面

2.7.2 图像切换器（ImageSwitcher）的功能与用法

ImageSwitcher 继承了 ViewSwitcher，因此它具有与 ViewSwitcher 相同的特征：可以在切换 View 组件时使用动画效果。ImageSwitcher 继承了 ViewSwitcher，并重写了 ViewSwitcher 的 showNext()、showPrevious()方法，因此 ImageSwitcher 使用起来更加简单。使用 ImageSwitcher 只要如下两步即可。

① 为 ImageSwitcher 提供一个 ViewFactory，该 ViewFactory 生成的 View 组件必须是 ImageView。

② 需要切换图片时，只要调用 ImageSwitcher 的 setImageDrawable(Drawable drawable)、setImageResource(int resid)和 setImageURI(Uri uri)方法更换图片即可。

提示：

ImageSwitcher 与 ImageView 的功能有点相似，它们都可用于显示图片，区别在于 ImageSwitcher 的效果更炫，它可以指定图片切换时的动画效果。

下面通过一个实例来介绍 ImageSwitcher 的用法。

实例：支持动画的图片浏览器

本实例是对前面的 GridViewTest 实例的修改，该实例使用 ImageSwitcher 代替了原有的 ImageView。该实例的界面布局文件如下。

程序清单：codes\02\2.7\ImageSwitcherTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<!-- 定义一个 GridView 组件 -->
<GridView
    android:id="@+id/grid01"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:horizontalSpacing="2dp"
    android:verticalSpacing="2dp"
    android:numColumns="4"
    android:gravity="center"/>
<!-- 定义一个 ImageSwitcher 组件 -->
<ImageSwitcher android:id="@+id/switcher"
    android:layout_width="300dp"
    android:layout_height="300dp"
    android:layout_gravity="center_horizontal"
    android:inAnimation="@android:anim/fade_in"
    android:outAnimation="@android:anim/fade_out"/>
</LinearLayout>
```

上面界面布局文件中的粗体字代码定义了一个 ImageSwitcher，并通过 android:inAnimation 和 android:outAnimation 指定了图片切换时的动画效果。

接下来 Activity 代码需要为该 ImageSwitcher 设置 ViewFactory，并让该 ViewFactory 的 makeView()方法返回 ImageView。下面是该 Activity 的代码。

程序清单：codes\02\2.7\ImageSwitcherTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    int[] imageIds = new int[]
    {
        R.drawable.bomb5 , R.drawable.bomb6 , R.drawable.bomb7
        , R.drawable.bomb8 , R.drawable.bomb9 , R.drawable.bomb10
        , R.drawable.bomb11 , R.drawable.bomb12 , R.drawable.bomb13
        , R.drawable.bomb14 , R.drawable.bomb15 , R.drawable.bomb16
    };
    ImageSwitcher switcher;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 List 对象，List 对象的元素是 Map
        List<Map<String, Object>> listItems =
                new ArrayList<Map<String, Object>>();
        for (int i = 0; i < imageIds.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
            listItem.put("image", imageIds[i]);
            listItems.add(listItem);
        }
        // 获取显示图片的 ImageSwitcher
        switcher = (ImageSwitcher)
                findViewById(R.id.switcher);
        // 为 ImageSwitcher 设置图片切换的动画效果
        switcher.setFactory(new ViewFactory()
        {
            @Override
            public View makeView()
            {
                // 创建 ImageView 对象
                ImageView imageView = new ImageView(MainActivity.this);
                imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                imageView.setLayoutParams(new ImageSwitcher.LayoutParams(
                    LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
                // 返回 ImageView 对象
                return imageView;
            }
```

```
        });
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this,
            listItems
            // 使用/layout/cell.xml 文件作为界面布局
            , R.layout.cell, new String[]{"image"},
            new int[] { R.id.image1 });
        GridView grid = (GridView) findViewById(R.id.grid01);
        // 为 GridView 设置 Adapter
        grid.setAdapter(simpleAdapter);
        // 添加列表项被选中的监听器
        grid.setOnItemSelectedListener(new OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
                    int position, long id)
            {
                // 显示当前被选中的图片
                switcher.setImageResource(imageIds[position]);
            }
            @Override
            public void onNothingSelected(AdapterView<?> parent)
            {
            }
        });
        // 添加列表项被单击的监听器
        grid.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id)
            {
                // 显示被单击的图片
                switcher.setImageResource(imageIds[position]);
            }
        });
    }
}
```

上面程序中的粗体字代码重写了 ViewFactory 的 makeView()方法，该方法返回一个 ImageView 对象，这样该 ImageSwitcher 即可正常工作。该 Activity 还为 GridView 绑定了事件监听器，当用户单击 GridView 或选中 GridView 的指定单元格时，ImageSwitcher 切换为显示对应的图片。

运行上面的实例，可以看到如图 2.57 所示的界面。

在图 2.57 所示界面中，用户单击或选中 GridView 中某个图标时，下面的 ImageSwitcher 将会切换为显示对应的图片，图片切换时会使用动画效果。

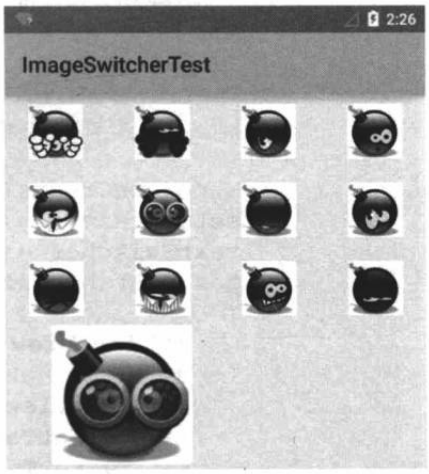


图 2.57 ImageSwitcher 测试

2.7.3 文本切换器（TextSwitcher）的功能与用法

TextSwitcher 继承了 ViewSwitcher，因此它具有与 ViewSwitcher 相同的特征：可以在切换 View 组件时使用动画效果。与 ImageSwitcher 相似的是，使用 TextSwitcher 也需要设置一个 ViewFactory。与 ImageSwitcher 不同的是，TextSwitcher 所需的 ViewFactory 的 makeView()方法必须返回一个 TextView 组件。

提示：

TextSwitcher与TextView的功能有点相似，它们都可用于显示文本内容，区别在于TextSwitcher的效果更炫，它可以指定文本切换时的动画效果。

下面的实例在界面上定义了一个TextSwitcher，界面布局文件如下。

程序清单：codes\02\2.7\TextSwitcherTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
<!-- 定义一个TextSwitcher，并指定了文本切换时的动画效果 -->
<TextSwitcher
    android:id="@+id/textSwitcher"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inAnimation="@android:anim/slide_in_left"
    android:outAnimation="@android:anim/slide_out_right"
    android:onClick="next"/>
</LinearLayout>
```

上面的界面布局文件中定义了一个TextSwitcher，并指定了文本切换时的动画效果。接下来Activity只要为该TextSwitcher设置ViewFactory，该TextSwitcher即可正常工作。

下面是该Activity的代码。

程序清单：codes\02\2.7\TextSwitcherTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    TextSwitcher textSwitcher;
    String[] strs = new String[]
        {
            "疯狂Java讲义",
            "轻量级Java EE企业应用实战",
            "疯狂Android讲义",
            "疯狂Ajax讲义"
        };
    int curStr;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textSwitcher = (TextSwitcher) findViewById(R.id.textSwitcher);
        textSwitcher.setFactory(new ViewSwitcher.ViewFactory()
        {
            public View makeView()
            {
                TextView tv = new TextView(MainActivity.this);
                tv.setTextSize(40);
                tv.setTextColor(Color.MAGENTA);
                return tv;
            }
        });
        // 调用next方法显示下一个字符串
        next(null);
    }
    // 事件处理函数，控制显示下一个字符串
    public void next(View source)
    {
```

```
            textSwitcher.setText(strs[curStr++ % strs.length]); // ①
        }
    }
```

上面的粗体字代码重写了 ViewFactory 的 makeView()方法，该方法返回了一个 TextView，这样即可让 TextSwitcher 正常工作。当程序要切换 TextSwitcher 显示文本时，调用 TextSwitcher 的 setText()方法修改文本即可——如上面的①号代码所示。

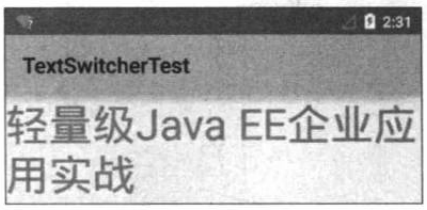


图 2.58 文本切换器

运行上面的程序，可以看到如图 2.58 所示的界面。

单击图 2.58 所示界面上的文本切换器，界面将会显示下一个文本，文本切换时将会使用动画效果。

2.7.4 ViewFlipper 的功能与用法

ViewFlipper 组件继承了 ViewAnimator，它可调用 addView(View v)添加多个组件，一旦向 ViewFlipper 中添加了多个组件之后，ViewFlipper 就可使用动画控制多个组件之间的切换效果。

ViewFlipper 与前面介绍的 AdapterViewFlipper 有较大的相似性，它们可以控制组件切换的动画效果。它们的区别是：ViewFlipper 需要开发者通过 addView(View v)添加多个 View，而 AdapterViewFlipper 则只要传入一个 Adapter，Adapter 将会负责提供多个 View。因此，ViewFlipper 可以指定与 AdapterViewFlipper 相同的 XML 属性。

实例：自动播放的图片库

该实例与前面介绍的 AdapterViewFlipper 实例非常相似，区别只是该实例直接定义了该 ViewFlipper 所包含的 View 组件。下面是该实例的界面布局文件。

程序清单：codes\02\2.7\ViewFlipperTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ViewFlipper
        android:id="@+id/details"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:flipInterval="1000">
        <ImageView
            android:src="@drawable/java"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </ImageView>
        <ImageView
            android:src="@drawable/android"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </ImageView>
        <ImageView
            android:src="@drawable/javaee"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </ImageView>
    </ViewFlipper>
    <Button
        android:text="&lt;"
```

```
        android:onClick="prev"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerInParent="true"
        android:onClick="auto"
        android:text="自动播放"/>
    <Button
        android:text="&gt;"
        android:onClick="next"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"/>
</RelativeLayout>
```

上面的界面布局文件中定义了一个 ViewFlipper，并在该 ViewFlipper 中定义了三个 ImageView，这意味着该 ViewFlipper 包含了三个子组件。接下来在 Activity 代码中即可调用 ViewFlipper 的 showPrevious()、showNext()等方法控制 ViewFlipper 显示上一个、下一个子组件。为了控制组件切换时的动画效果，还需要调用 ViewFlipper 的 setInAnimation()、setOutAnimation()方法设置动画效果。

下面是该 Activity 的代码。

程序清单：codes\02\2.7\ViewFlipperTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    private ViewFlipper viewFlipper;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        viewFlipper = (ViewFlipper) findViewById(R.id.details);
    }
    public void prev(View source)
    {
        viewFlipper.setInAnimation(this , R.anim.slide_in_right);
        viewFlipper.setOutAnimation(this , R.anim.slide_out_left);
        // 显示上一个组件
        viewFlipper.showPrevious();
        // 停止自动播放
        viewFlipper.stopFlipping();
    }
    public void next(View source)
    {
        viewFlipper.setInAnimation(this , android.R.anim.slide_in_left);
        viewFlipper.setOutAnimation(this , android.R.anim.slide_out_right);
        // 显示下一个组件
        viewFlipper.showNext();
        // 停止自动播放
        viewFlipper.stopFlipping();
    }
    public void auto(View source)
    {
        viewFlipper.setInAnimation(this , android.R.anim.slide_in_left);
        viewFlipper.setOutAnimation(this , android.R.anim.slide_out_right);
```

```
        // 开始自动播放
        viewFlipper.startFlipping();
    }
}
```

上面程序中的粗体字代码就是控制 ViewFlipper 切换组件的动画效果，以及控制 ViewFlipper 切换组件的关键代码。运行该程序，可以看到如图 2.59 所示的界面。

单击图 2.59 所示界面下方的按钮，即可看到图片上、下切换，而且组件切换时将可以看到动画切换效果。

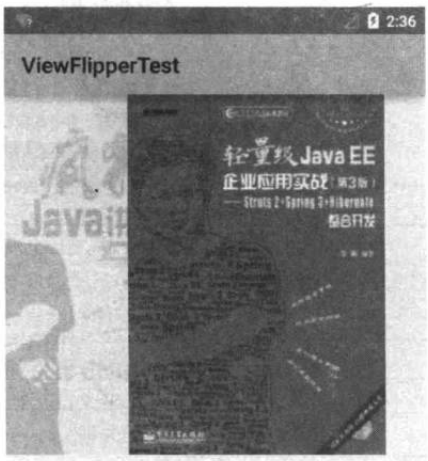


图 2.59 ViewFlipper 控制图片切换

2.8 各种杂项组件

除了前面介绍的 6 组 UI 组件之外，Android 还有如下一些常用的杂项组件，掌握这些杂项组件也是开发 Android 应用必需的技能。

2.8.1 使用 Toast 显示提示信息框

Toast 是一种非常方便的提示消息框，它会在程序界面上显示一个简单的提示信息。这个提示信息框用于向用户生成简单的提示信息。它具有如下两个特点。

- Toast 提示信息不会获得焦点。
- Toast 提示信息过一段时间会自动消失。

使用 Toast 来生成提示消息也非常简单，只要如下几个步骤即可。

1. 调用 Toast 的构造器或 makeText()静态方法创建一个 Toast 对象。
2. 调用 Toast 的方法来设置该消息提示的对齐方式、页边距等。
3. 调用 Toast 的 show()方法将它显示出来。

Toast 的功能和用法都比较简单，大部分时候它只能显示简单的文本提示；如果应用需要显示诸如图片、列表之类的复杂提示，一般建议使用对话框来完成；如果开发者确实想通过 Toast 来完成，也是可以的，此时就需要调用 Toast 构造器创建实例，再调用 setView()方法设置该 Toast 显示的 View 组件。该方法允许开发者自己定义 Toast 显示的内容。

下面以一个实例程序来示范 Toast 的用法。

实例：带图片的消息提示

本实例程序非常简单，它在用户界面上显示了两个按钮，其中一个按钮用于激发普通的 Toast 提示；另一个按钮用于激发带图片的 Toast 提示。这意味着开发者必须调用该 Toast 对象的 setView()方法来改变该 Toast 对象的内容 View。

本程序的用户界面很简单，只有两个普通按钮，故不再给出界面布局文件代码。本程序的 Java 代码如下。

程序清单：codes\02\2.8\ToastTest\app\src\main\java\org\crazyit\ui\MainActivity.java

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

```
        Button simple = (Button) findViewById(R.id.simple);
        // 为按钮的单击事件绑定事件监听器
        simple.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建一个 Toast 提示信息
                Toast toast = Toast.makeText(MainActivity.this
                    , "简单的提示信息"
                // 设置该 Toast 提示信息的持续时间
                    , Toast.LENGTH_SHORT);
                toast.show();
            }
        });
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮的单击事件绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建一个 Toast 提示信息
                Toast toast = new Toast(MainActivity.this);
                // 设置 Toast 的显示位置
                toast.setGravity(Gravity.CENTER, 0, 0);
                // 创建一个 ImageView
                ImageView image = new ImageView(MainActivity.this);
                image.setImageResource(R.drawable.tools);
                // 创建一个 LinearLayout 容器
                LinearLayout ll = new LinearLayout(MainActivity.this);
                // 向 LinearLayout 中添加图片、原有的 View
                ll.addView(image);
                // 创建一个 TextView
                TextView textView = new TextView(MainActivity.this);
                textView.setText("带图片的提示信息");
                // 设置文本框内字号的大小和字体颜色
                textView.setTextSize(24);
                textView.setTextColor(Color.MAGENTA);
                ll.addView(textView);
                // 设置 Toast 显示自定义 View
                toast.setView(ll);
                // 设置 Toast 的显示时间
                toast.setDuration(Toast.LENGTH_LONG);
                toast.show();
            }
        });
    }
}
```

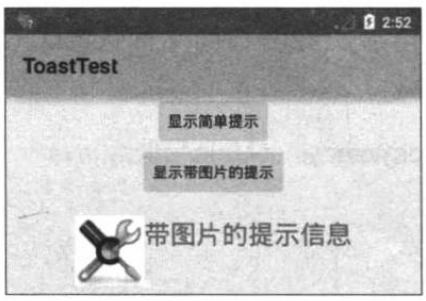


图 2.60　带图片的消息提示

上面的程序比较简单：第一个按钮被单击时，程序只是简单地创建了一个 Toast 对象，并把它显示出来，因此单击第一个按钮只是看到一个简单的 Toast 提示；当第二个按钮被单击时，程序先创建了一个 Toast 对象，并调用该 Toast 对象的 setView()方法改变了该消息提示的内容。因此单击第二个按钮时将看到带图片的消息提示，如图 2.60 所示。

2.8.2 日历视图（CalendarView）组件的功能和用法

日历视图（CalendarView）可用于显示和选择日期，用户既可选择一个日期，也可通过触摸来滚动日历。如果希望监控该组件的日期改变，则可调用 CalendarView 的 setOnDateChangeListener()方法为此组件的点击事件添加事件监听器。

使用 CalendarView 时可指定如表 2.30 所示的常见 XML 属性及相关方法。

表 2.30 CalendarView 支持的常见 XML 属性及相关方法

XML 属性	相关方法	说明
android:dateTextAppearance	setDateTextAppearance(int)	设置该日历视图的日期文字的样式
android:firstDayOfWeek	setFirstDayOfWeek(int)	设置每周的第一天，允许设置周一到周日任意一天作为每周的第一天
android:focusedMonthDateColor	setFocusedMonthDateColor(int)	设置获得焦点的月份的日期文字的颜色
android:maxDate	setMaxDate(long)	设置该日历组件支持的最大日期。以 mm/dd/yyyy 格式指定最大日期
android:minDate	setMinDate(long)	设置该日历组件支持的最小日期。以 mm/dd/yyyy 格式指定最小日期。
android:selectedDateVerticalBar	setSelectedDateVerticalBar(int)	设置绘制在选中日期两边的竖线对应的 Drawable
android:selectedWeekBackgroundColor	setSelectedWeekBackgroundColor(int)	设置被选中周的背景色
android:showWeekNumber	setShowWeekNumber(boolean)	设置是否显示第几周
android:shownWeekCount	setShownWeekCount(int)	设置该日历组件总共显示几个星期
android:unfocusedMonthDateColor	setUnfocusedMonthDateColor(int)	设置没有焦点的月份的日期文字的颜色
android:weekDayTextAppearance	setWeekDayTextAppearance(int)	设置星期几的文字样式
android:weekNumberColor	setWeekNumberColor(int)	设置显示周编号的颜色
android:weekSeparatorLineColor	setWeekSeparatorLineColor(int)	设置周分割线的颜色

下面通过实例来示范 CalendarView 组件的功能与用法。

实例：选择您的生日

本实例的重点是界面布局文件中添加了一个 CalendarView 组件。下面是该实例的界面布局文件。

程序清单：codes\02\2.8\CalendarViewTest\app\src\main\res\layout\main.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="选择您的生日："/>
    <!-- 设置以星期二作为每周第一天，
    设置该组件总共显示 4 个星期，
    并对该组件的日期时间进行了定制 -->
    <CalendarView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:firstDayOfWeek="3"
        android:shownWeekCount="4"
        android:selectedWeekBackgroundColor="#aff"
        android:focusedMonthDateColor="#f00"
```

```
        android:weekSeparatorLineColor="#ff0"
        android:unfocusedMonthDateColor="#f9f"
        android:id="@+id/calendarView" />
</LinearLayout>
```

上面的界面布局文件中粗体字代码定义了一个CalenarView组件，并设置该组件总共显示4周，以每周的星期二作为第一天。

为了监听用户选择日期的事件，本实例在Activity代码中调用该组件的setOnDateChangeListener()方法来添加事件监听器。该实例的Activity代码如下。

程序清单：codes\02\2.8\CalendarViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    CalendarView cv;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        cv = (CalendarView)findViewById(R.id.calendarView);
        // 为CalendarView组件的日期改变事件添加事件监听器
        cv.setOnDateChangeListener(new OnDateChangeListener()
        {
            @Override
            public void onSelectedDayChange(CalendarView view, int year,
                int month, int dayOfMonth)
            {
                // 使用Toast显示用户选择的日期
                Toast.makeText(MainActivity.this,
                    "你生日是" + year + "年" + month + "月"
                    + dayOfMonth + "日" ,
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

上面Activity的粗体字代码实现了onSelectedDayChange()，当用户选择的日期发生改变时将会激发该方法。运行上面的程序，当用户选择的日期发生改变时，将可以看到程序显示如图2.61所示的界面。

图2.61 通过CalendarView组件选择日期

2.8.3 日期、时间选择器（DatePicker和TimePicker）的功能和用法

DatePicker和TimePicker是两个比较易用的组件，它们都从FrameLayout派生而来，其中DatePicker供用户选择日期；而TimePicker则供用户选择时间。

DatePicker和TimePicker在FrameLayout的基础上提供了一些方法来获取当前用户所选择的日期、时间；如果程序需要获取用户选择的日期、时间，则可通过为DatePicker添加OnDateChangedListener进行监听、为TimePicker添加OnTimerChangedListener进行监听来实现。

使用 DatePicker 时可指定如表 2.31 所示的常见 XML 属性。

表 2.31 DatePicker 支持的常见 XML 属性

XML 属性	说明
android:calendarViewShown	设置该日期选择器是否显示 CalendarView 组件
android:endYear	设置日期选择器允许选择的最后一年
android:maxDate	设置该日期选择器支持的最大日期。以 mm/dd/yyyy 格式指定最大日期
android:minDate	设置该日期选择器支持的最小日期。以 mm/dd/yyyy 格式指定最小日期
android:spinnersShown	设置该日期选择器是否显示 Spinner 日期选择组件
android:startYear	设置日期选择器允许选择的第一年

下面以一个让用户选择日期、时间的实例来示范 DatePicker 和 TimePicker 的功能与用法。

实例：用户选择日期、时间

为了让用户能选择日期，本应用需要同时使用 DatePicker 和 TimePicker 两个组件，并为它们分别绑定监听器。下面是本应用的界面布局文件。

程序清单：codes\02\2.8\ChooseDate\app\src\main\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="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="选择购买本书的具体时间"/>
    <!-- 定义一个 DatePicker 组件 -->
    <DatePicker android:id="@+id/datePicker"
        android:layout_width="wrap_content"
        android:layout_height="200dp"
        android:layout_gravity="center_horizontal"
        android:startYear="2000"
        android:endYear="2016"
        android:calendarViewShown="true"
        android:spinnersShown="true"/>
    <!-- 定义一个 TimePicker 组件 -->
    <TimePicker android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_gravity="center_horizontal"/>
    <!-- 显示用户输入日期、时间的控件 -->
    <EditText android:id="@+id/show"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:editable="false"
        android:cursorVisible="false"/>
</LinearLayout>
```

上面的界面布局中添加了一个 DatePicker、一个 TimePicker，这两个组件供用户选择日期、时间。除此之外，还包含了一个 EditText，该组件用于显示用户选择的日期、时间。

本应用的主程序如下。

程序清单：codes\02\2.8\ChooseDate\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义 5 个记录当前时间的变量
```

```
    private int year;
    private int month;
    private int day;
    private int hour;
    private int minute;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        DatePicker datePicker = (DatePicker)
            findViewById(R.id.datePicker);
        TimePicker timePicker = (TimePicker)
            findViewById(R.id.timePicker);
        // 获取当前的年、月、日、小时、分钟
        Calendar c = Calendar.getInstance();
        year = c.get(Calendar.YEAR);
        month = c.get(Calendar.MONTH);
        day = c.get(Calendar.DAY_OF_MONTH);
        hour = c.get(Calendar.HOUR);
        minute = c.get(Calendar.MINUTE);
        // 初始化 DatePicker 组件，初始化时指定监听器
        datePicker.init(year, month, day, new OnDateChangedListener()
        {
            @Override
            public void onDateChanged(DatePicker arg0, int year
                    , int month, int day)
            {
                MainActivity.this.year = year;
                MainActivity.this.month = month;
                MainActivity.this.day = day;
                // 显示当前日期、时间
                showDate(year, month, day, hour, minute);
            }
        });
        // 为 TimePicker 指定监听器
        timePicker.setOnTimeChangedListener(new OnTimeChangedListener()
        {
            @Override
            public void onTimeChanged(TimePicker view
                    , int hourOfDay, int minute)
            {
                MainActivity.this.hour = hourOfDay;
                MainActivity.this.minute = minute;
                // 显示当前日期、时间
                showDate(year, month, day, hour, minute);
            }
        });
    }
    // 定义在 EditText 中显示当前日期、时间的方法
    private void showDate(int year, int month
            , int day, int hour, int minute)
    {
        EditText show = (EditText) findViewById(R.id.show);
        show.setText("您的购买日期为：" + year + "年"
                + (month + 1) + "月" + day + "日 "
                + hour + "时" + minute + "分");
    }
}
```

上面程序中的两行粗体字代码就是分别为 DatePicker、TimePicker 绑定事件监听器的代码，DatePicker 和 TimePicker 绑定监听器的方式略有不同，但本质还是一样的。一旦为 DatePicker、

TimePicker 绑定了监听器，当用户通过这两个组件来选择日期、时间时，监听器就会被触发——监听器负责使用 EditText 来显示用户选择的日期、时间。运行上面的程序，将看到如图 2.62 所示的界面。

图 2.62 选择日期、时间

提示：

对于 DatePicker、TimePicker 两个 UI 组件而言，使用 Material 风格时，虽然视觉效果非常好，但笔者测试时暂不能选择日期、时间，因此本实例同样改为使用 Holo 风格才可看到如图 2.62 所示的效果。

2.8.4 数值选择器（NumberPicker）的功能与用法

数值选择器用于让用户输入数值，用户既可以通过键盘输入数值，也可以通过拖动来选择数值。使用该组件常用如下三个方法。

- setMinValue(int minVal)：设置该组件支持的最小值。
- setMaxValue(int maxVal)：设置该组件支持的最大值。
- setValue(int value)：设置该组件的当前值。

下面通过一个实例来介绍 NumberPicker 的功能与用法。

实例：选择您意向的价格范围

在该实例中，程序将会使用两个 NumberPicker 来让用户选择价格，其中第一个 NumberPicker 用于选择低价；第二个 NumberPicker 用于选择高价。下面是该实例的界面布局文件。

程序清单：codes\02\2.8\NumberPickerTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:text="选择低价："
            android:layout_width="120dp"
            android:layout_height="wrap_content" />
        <NumberPicker
            android:id="@+id/np1"
            android:layout_width="match_parent"
            android:layout_height="80dp"
            android:focusable="true"
            android:focusableInTouchMode="true" />
    </TableRow>
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:text="选择高价："
            android:layout_width="120dp"
```

```
            android:layout_height="wrap_content" />
        <NumberPicker
            android:id="@+id/np2"
            android:layout_width="match_parent"
            android:layout_height="80dp"
            android:focusable="true"
            android:focusableInTouchMode="true" />
    </TableRow>
</TableLayout>
```

上面的界面布局文件中定义了两个 NumberPicker，接下来 Activity 代码需要为这两个 NumberPicker 设置最小值、最大值，并为它们绑定事件监听器。下面是该 Activity 的代码。

程序清单：codes\02\2.8\NumberPickerTest\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    NumberPicker np1, np2;
    // 定义最低价格、最高价格的初始值
    int minPrice = 25, maxPrice = 75;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        np1 = (NumberPicker) findViewById(R.id.np1);
        // 设置 np1 的最小值和最大值
        np1.setMinValue(10);
        np1.setMaxValue(50);
        // 设置 np1 的当前值
        np1.setValue(minPrice);
        np1.setOnValueChangedListener(new OnValueChangeListener()
        {
            // 当 NumberPicker 的值发生改变时，将会激发该方法
            @Override
            public void onValueChange(NumberPicker picker,
                int oldVal, int newVal)
            {
                minPrice = newVal;
                showSelectedPrice();
            }
        });
        np2 = (NumberPicker) findViewById(R.id.np2);
        // 设置 np2 的最小值和最大值
        np2.setMinValue(60);
        np2.setMaxValue(100);
        // 设置 np2 的当前值
        np2.setValue(maxPrice);
        np2.setOnValueChangedListener(new OnValueChangeListener()
        {
            // 当 NumberPicker 的值发生改变时，将会激发该方法
            @Override
            public void onValueChange(NumberPicker picker, int oldVal,
                    int newVal)
            {
                maxPrice = newVal;
                showSelectedPrice();
            }
        });
    }
    private void showSelectedPrice()
    {
        Toast.makeText(this, "您选择最低价格为：" + minPrice
            + ",最高价格为：" + maxPrice, Toast.LENGTH_SHORT)
```

```
            .show();
        }
    }
```

上面两段粗体字代码的控制逻辑基本是相似的，它们都调用了 NumberPicker 的 setMinValue()、setMaxValue()、setValue()来设置该数值选择器的最小值、最大值和当前值。除此之外，程序还为两个数值选择器绑定了事件监听器：当它们的值发生改变时，将会激发相应的事件处理方法。

运行该程序，并通过 NumberPicker 选择数值，将可以看到如图 2.63 所示的界面。

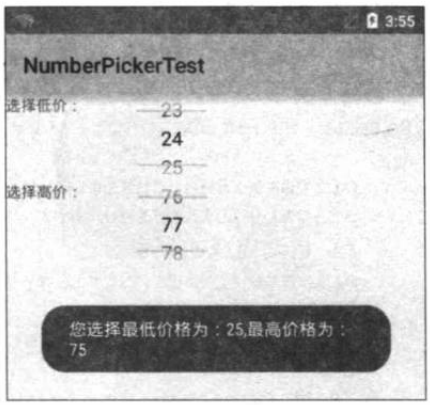


图 2.63 使用 NumberPicker 选择数值

2.8.5 搜索框（SearchView）的功能与用法

SearchView 是搜索框组件，它可以让用户在文本框内输入文字，并允许通过监听器监控用户输入，当用户输入完成后提交搜索时，也可通过监听器执行实际的搜索。

使用 SearchView 时可使用如下常用方法。

- setIconifiedByDefault (boolean iconified)：设置该搜索框默认是否自动缩小为图标。
- setSubmitButtonEnabled(boolean enabled)：设置是否显示搜索按钮。
- setQueryHint(CharSequence hint)：设置搜索框内默认显示的提示文本。
- setOnQueryTextListener (SearchView.OnQueryTextListener listener)：为该搜索框设置事件监听器。

如果为 SearchView 增加一个配套的 ListView，则可以为 SearchView 增加自动完成的功能。如下实例示范了 SearchView 的功能与用法。

实例：搜索

该实例的界面布局文件中定义了一个 SearchView 和 ListView，其中 ListView 用于为 SearchView 显示自动补齐列表。界面布局文件如下。

程序清单：codes\02\2.8\SearchViewTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <!-- 定义一个 SearchView -->
    <SearchView
        android:id="@+id/sv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <!-- 为 SearchView 定义自动完成的 ListView-->
    <ListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"/>
</LinearLayout>
```

上面的布局文件中定义了一个 SearchView 组件，并为该 SearchView 组件定义了一个 ListView 组件，该 ListView 组件用于为 SearchView 组件显示自动完成列表。

下面是该实例对应的 Activity 代码。

程序清单：codes\02\2.8\SearchViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    private SearchView sv;
    private ListView lv;
    // 自动完成的列表
    private final String[] mStrings = { "aaaaa", "bbbbbb", "cccccc" };
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        lv = (ListView) findViewById(R.id.lv);
        lv.setAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, mStrings));
        // 设置 ListView 启用过滤
        lv.setTextFilterEnabled(true);
        sv = (SearchView) findViewById(R.id.sv);
        // 设置该 SearchView 默认是否自动缩小为图标
        sv.setIconifiedByDefault(false);
        // 设置该 SearchView 显示搜索按钮
        sv.setSubmitButtonEnabled(true);
        // 设置该 SearchView 内默认显示的提示文本
        sv.setQueryHint("查找");
        // 为该 SearchView 组件设置事件监听器
        sv.setOnQueryTextListener(new OnQueryTextListener()
        {
            // 用户输入字符时激发该方法
            @Override
            public boolean onQueryTextChange(String newText)
            {
                // 如果 newText 不是长度为 0 的字符串
                if (TextUtils.isEmpty(newText))
                {
                    // 清除 ListView 的过滤
                    lv.clearTextFilter();
                }
                else
                {
                    // 使用用户输入的内容对 ListView 的列表项进行过滤
                    lv.setFilterText(newText);
                }
                return true;
            }
            // 单击搜索按钮时激发该方法
            @Override
            public boolean onQueryTextSubmit(String query)
            {
                // 实际应用中应该在该方法内执行实际查询
                // 此处仅使用 Toast 显示用户输入的查询内容
                Toast.makeText(MainActivity.this, "您的选择是:" + query
                        , Toast.LENGTH_SHORT).show();
                return false;
            }
        });
    }
}
```

上面程序中的粗体字代码就是控制 SearchView 的关键代码，第二行粗体字代码为

SeachView 设置了事件监听器，并为该 SearchView 启用了搜索按钮。接下来程序重写了 onQueryTextChange()、onQueryTextSubmit()两个方法，这两个方法用于为 SearchView 的事件提供响应。

运行该程序，将可以看到如图 2.64 所示的界面。

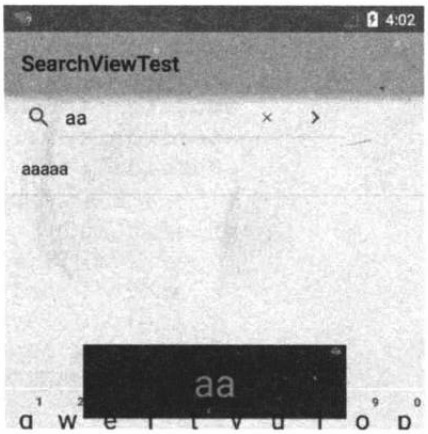


图 2.64 使用 SearchView 执行搜索

2.8.6 选项卡（TabHost）的功能和用法

TabHost 是一种非常实用的组件，TabHost 可以很方便地在窗口上放置多个标签页，每个标签页相当于获得了一个与外部容器相同大小的组件摆放区域。通过这种方式，就可以在一个容器里放置更多组件，例如许多手机系统都会在同一个窗口中定义多个标签页来显示通话记录，包括“未接电话”、“已接电话”、“呼出电话”等。

与 TabHost 结合使用的还有如下组件。

- TabWidget：代表选项卡的标题条。
- TabSpec：代表选项卡的一个 Tab 页面。

TabHost 仅仅是一个简单的容器，它提供了如下两个方法来创建、添加标签页。

- newTabSpec(String tag)：创建选项卡。
- addTab(TabHost.TabSpec tabSpec)：添加选项卡。

使用 TabHost 的一般步骤如下。

1. 在界面布局文件中定义 TabHost 组件，并为该组件定义该选项卡的内容。
2. Activity 应该继承 TabActivity。
3. 调用 TabActivity 的 getTabHost()方法获取 TabHost 对象。
4. 通过 TabHost 对象的方法来创建、添加选项卡。

除此之外，TabHost 还提供了一些方法来获取当前选项卡。获取当前 View 的方法，具体可以参考 API 文档。如果程序需要监控 TabHost 里当前标签页的改变，则可以为它设置 TabHost.OnTabChangeListener 监听器。

下面通过一个简单的实例来示范选项卡的用法。

实例：通话记录界面

本实例程序使用 TabHost 定义一个标签容器，并使用三个 LinearLayout 来定义标签页（实际上可以使用任何 View 组件来定义标签页）。该程序的界面布局文件如下。

程序清单：codes\02\2.8\TabHostTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <TabWidget
            android:id="@android:id/tabs"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <!-- 定义第一个标签页的内容 -->
            <LinearLayout
                android:id="@+id/tab01"
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
                <!-- 省略两个 TextView 的定义 -->
                ...
            </LinearLayout>
            <!-- 定义第二个标签页的内容 -->
            <LinearLayout
                android:id="@+id/tab02"
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
                <!-- 省略两个 TextView 的定义 -->
                ...
            </LinearLayout>
            <!-- 定义第三个标签页的内容 -->
            <LinearLayout
                android:id="@+id/tab03"
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:textSize="11pt">
                <!-- 省略两个 TextView 的定义 -->
                ...
            </LinearLayout>
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

请注意上面的布局文件中粗体字代码部分，从上面的布局文件可以发现，TabHost 容器内部需要组合两个组件：TabWidget 和 FrameLayout，其中 TabWidget 用于定义选项卡的标题条；FrameLayout 则用于“层叠”组合多个选项页面。不仅如此，上面的布局文件中这三个组件的 ID 也有要求。

- TabHost 的 ID 应该为@android:id/tabhost。
- TabWidget 的 ID 应该为@android:id/tabs。
- FrameLayout 的 ID 应该为@android:id/tabcontent。

上面这三个 ID 并不是开发者自己定义的，而是引用了 Android 系统已有的 ID。

接下来主程序即可加载该布局资源，并将布局文件中的三个 Tab 页面添加到该 TabHost 容器中。该 Activity 代码如下。

程序清单：codes\02\2.8\TabHostTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends TabActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取该 Activity 里面的 TabHost 组件
```

```
        TabHost tabHost = getTabHost();
        // 创建第一个 Tab 页面
        TabSpec tab1 = tabHost.newTabSpec("tab1")
            .setIndicator("已接电话") // 设置标题
            .setContent(R.id.tab01); // 设置内容
        // 添加第一个标签页
        tabHost.addTab(tab1);
        TabSpec tab2 = tabHost.newTabSpec("tab2")
            // 在标签标题上放置图标
            .setIndicator("呼出电话", getResources()
            .getDrawable(R.drawable.ic_launcher))
            .setContent(R.id.tab02);
        // 添加第二个标签页
        tabHost.addTab(tab2);
        TabSpec tab3 = tabHost.newTabSpec("tab3")
            .setIndicator("未接电话")
            .setContent(R.id.tab03);
        // 添加第三个标签页
        tabHost.addTab(tab3);
    }
}
```

上面程序中的粗体字代码就是为 TabHost 创建并添加 Tab 页面的代码。上面的程序一共添加了三个标签页，因此类似粗体字的代码一共写了三次。其中第二个标签的标题上还添加了一个图标（该主题下不会显示图标）。

运行上面的程序，将看到如图 2.65 所示的界面。

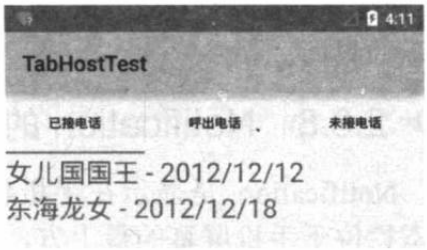


图 2.65 通话记录界面

上面的程序调用了 TabHost.TabSpec 对象的 setContent(int viewId)方法来设置标签页内容。除此之外，还可调用 setContent(Intent intent)方法来设置标签页内容，Intent 还可用于启动其他 Activity —— 这意味着 TabHost.TabSpec 可直接装载另一个 Activity。本书后面有关于 Intent 的详细介绍。

注意：

最新版本的 Android 平台已经不再推荐使用 TabActivity，而是推荐使用 Fragment 来代替 TabActivity，本书第 4 章会详细介绍 Fragment 的相关知识。

2.8.7 滚动视图（ScrollView）的功能和用法

ScrollView 由 FrameLayout 派生而出，它就是一个用于为普通组件添加滚动条的组件。ScrollView 里最多只能包含一个组件，而 ScrollView 的作用就是为该组件添加垂直滚动条。

提示：

ScrollView 的作用和 Swing 编程中的 JScrollPane 非常相似，它们甚至不能被称为真正的容器，它们只是为其他容器添加滚动条。

在默认情况下，ScrollView 只是为其他组件添加垂直滚动条，如果应用需要添加水平滚动条，则可借助于另一个滚动视图 —— HorizontalScrollView 来实现。ScrollView 与 HorizontalScrollView 的功能基本相似，只是前者添加垂直滚动条，后者添加水平滚动条。

下面以一个例子来示范 ScrollView、HorizontalScrollView 的用法。

实例：可垂直和水平滚动的视图

本实例程序通过在 ScrollView 里嵌套 HorizontalScrollView，来为应用的界面同时添加水平滚动条、垂直滚动条。下面是该应用的界面布局文件。

程序清单：codes\02\2.8\ScrollViewTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义 ScrollView，为里面的组件添加垂直滚动条 -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 定义 HorizontalScrollView，为里面的组件添加水平滚动条 -->
<HorizontalScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<LinearLayout android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 省略多个 TextView 组件 -->
...
</LinearLayout>
</HorizontalScrollView>
</ScrollView>
```

上面的界面布局实现了界面的垂直、水平同时滚动。使用 Activity 显示上面的界面布局，将看到如图 2.66 所示的界面。

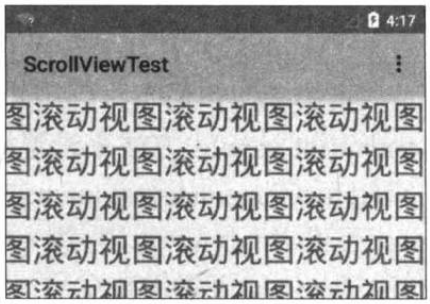


图 2.66 水平、垂直滚动

2.8.8 Notification 的功能与用法

Notification 是显示在手机状态栏的通知——手机状态栏位于手机屏幕的最上方，那里一般显示手机当前的网络状态、电池状态、时间等。Notification 所代表的是一种具有全局效果的通知，程序一般通过 NotificationManager 服务来发送 Notification。

提示：

NotificationManager 是一个重要的系统服务，该 API 位于图 1.1 中的应用程序框架层，应用程序可通过 NotificationManager 向系统发送全局通知。

Android 为 Notification 增加了 Notification.Builder 类，通过该类允许开发者更轻松地创建 Notification 对象。Notification.Builder 提供了如下常用方法。

- setDefaults()：设置通知 LED 灯、音乐、振动等。
- setAutoCancel()：设置点击通知后，状态栏自动删除通知。
- setContentTitle()：设置通知标题。
- setContentText()：设置通知内容。
- setSmallIcon()：为通知设置图标。
- setLargeIcon()：为通知设置大图标。
- setTick()：设置通知在状态栏的提示文本。
- setContentIntent()：设置点击通知后将要启动的程序组件对应的 PendingIntent。

发送 Notification 很简单，按如下步骤进行即可。

① 调用 getSystemService(NOTIFICATION_SERVICE)方法获取系统的 NotificationManager 服务。

② 通过构造器创建一个 Notification 对象。

③ 为 Notification 设置各种属性。

④ 通过 NotificationManager 发送 Notification。

下面通过实例来介绍 Notification 的功能和用法。

实例：加薪通知

本实例示范了如何通过 NotificationManager 来发送、取消 Notification。本实例的界面很简单，只是包含两个普通按钮，分别用于发送 Notification 和取消 Notification。本实例程序的 Java 代码如下。

程序清单：codes\02\2.8\NotificationTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    static final int NOTIFICATION_ID = 0x123;
    NotificationManager nm;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 NotificationManager 服务
        nm = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);
    }
    // 为发送通知的按钮的点击事件定义事件处理方法
    public void send(View source)
    {
        // 创建一个启动其他 Activity 的 Intent
        Intent intent = new Intent(MainActivity.this
                , OtherActivity.class);
        PendingIntent pi = PendingIntent.getActivity(
                MainActivity.this, 0, intent, 0);
        Notification notify = new Notification.Builder(this)
            // 设置打开该通知，该通知自动消失
            .setAutoCancel(true)
            // 设置显示在状态栏的通知提示信息
            .setTicker("有新消息")
            // 设置通知的图标
            .setSmallIcon(R.drawable.notify)
            // 设置通知内容的标题
            .setContentTitle("一条新通知")
            // 设置通知内容
            .setContentText("恭喜您，您加薪了，工资增加 20%!")
            // 设置使用系统默认的声音、默认 LED 灯
            // .setDefaults(Notification.DEFAULT_SOUND
                // |Notification.DEFAULT_LIGHTS)
            // 设置通知的自定义声音
            .setSound(Uri.parse("android.resource://org.crazyit.ui/"
                + R.raw.msg))
            .setWhen(System.currentTimeMillis())
            // 设置通知将要启动程序的 Intent
            .setContentIntent(pi)  // ①
            .build();
        // 发送通知
```

```
        nm.notify(NOTIFICATION_ID, notify);
    }
    // 为删除通知的按钮的点击事件定义事件处理方法
    public void del(View v)
    {
        // 取消通知
        nm.cancel(NOTIFICATION_ID);
    }
}
```

上面程序中的粗体字代码用于为 Notification 设置各种属性，包括 Notification 的图标、标题、发送时间等。

除此之外，上面的程序还通过 setDefaults()方法为 Notification 设置了声音提示、振动提示、闪光灯等。该属性支持如下属性值。

- DEFAULT_SOUND：设置使用默认声音。
- DEFAULT_VIBRATE：设置使用默认振动。
- DEFAULT_LIGHTS：设置使用默认闪光灯。
- ALL：设置使用默认声音、振动、闪光灯。

如果不想使用默认设置，也可以使用如下代码：

```
// 设置自定义声音
setSound(Uri.parse("file:///sdcard/click.mp3"));
// 设置自定义振动
setVibrate(new long[]{0 , 50 , 100, 150});
```

接下来的①号代码用于为该 Notification 设置事件信息，设置事件信息时传入了一个 PendingIntent 对象，该对象里封装了一个 Intent，这意味着单击该 Notification 时将会启动该 Intent 对应的程序。

图 2.67 发送通知

运行上面的程序，单击“发送 Notification”按钮，将可以看到手机屏幕上方出现了一个 Notification。将状态栏向下拖动，将可以看到 Notification 的详情，如图 2.67 所示。

图 2.67 所示的 Notification 还关联了一个 Activity：OtherActivity，因此当用户单击“普通通知”时即可启动 OtherActivity——OtherActivity 是一个十分简单的程序，故此处不再介绍。

由于上面的程序指定了该 Notification 要启动 OtherActivity，因此一定不要忘记在 AndroidManifest.xml 文件中声明该 Activity。而且上面的程序还需要访问系统闪光灯、振动器，这也需要在 AndroidManifest.xml 文件中声明权限。也就是增加如下代码片段：

```
<activity android:name=".OtherActivity" android:label="@string/other_activity">
</activity>
<!-- 添加操作闪光灯的权限 -->
<uses-permission android:name="android.permission.FLASHLIGHT"/>
<!-- 添加操作振动器的权限 -->
<uses-permission android:name="android.permission.VIBRATE"/>
```

2.9 第 7 组 UI 组件：对话框

Android 提供了丰富的对话框支持，它提供了如下 4 种常用的对话框。

- AlertDialog：功能最丰富、实际应用最广的对话框。
- ProgressDialog：进度对话框，这个对话框只是对进度条的包装。

- DatePickerDialog：日期选择对话框，这个对话框只是对 DatePicker 的包装。
- TimePickerDialog：时间选择对话框，这个对话框只是对 TimePicker 的包装。

上面 4 种对话框中功能最强、用法最灵活的就是 AlertDialog，因此应用也非常广泛，而其他三种对话框都是它的子类。图 2.68 显示了它们的继承关系图。

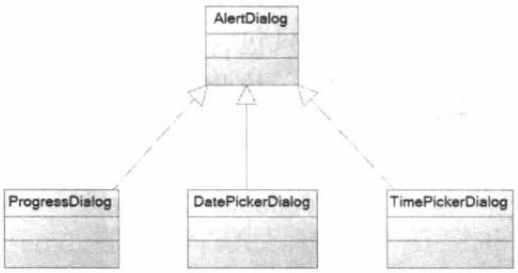


图 2.68 AlertDialog 及其子类的继承关系图

本节将会详细介绍各种对话框的功能和用法。

2.9.1 使用 AlertDialog 创建对话框

AlertDialog 的功能很强大，它可以生成各种内容的对话框。但实际上 AlertDialog 生成的对话框总有如图 2.69 所示的结构。

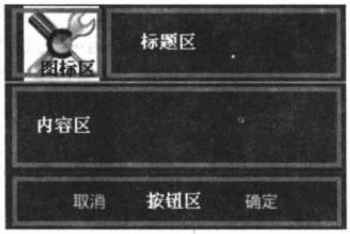


图 2.69 对话框的结构

从图 2.69 可以看出，AlertDialog 生成的对话框可分为如下 4 个区域。

- 图标区
- 标题区
- 内容区
- 按钮区

从上面对话框的结构来看，创建一个对话框需要经过如下几步。

1. 创建 AlertDialog.Builder 对象。
2. 调用 AlertDialog.Builder 的 setTitle()或 setCustomTitle()方法设置标题。
3. 调用 AlertDialog.Builder 的 setIcon()方法设置图标。
4. 调用 AlertDialog.Builder 的相关设置方法设置对话框内容。
5. 调用 AlertDialog.Builder 的 setPositiveButton()、setNegativeButton()或 setNeutralButton()方法添加多个按钮。
6. 调用 AlertDialog.Builder 的 create()方法创建 AlertDialog 对象，再调用 AlertDialog 对象的 show()方法将该对话框显示出来。

在上面的 6 个步骤中，第 4 步是最灵活的，AlertDialog 允许创建各种内容的对话框。归纳起来，AlertDialog 提供了如下 6 种方法来指定对话框的内容。

- setMessage()：设置对话内容为简单文本。
- setItems()：设置对话框内容为简单列表项。
- setSingleChoiceItems()：设置对话框内容为单选列表项。
- setMultiChoiceItems()：设置对话框内容为多选列表项。
- setAdapter()：设置对话框内容为自定义列表项。
- setView()：设置对话框内容为自定义 View。

下面通过几个实例来介绍 AlertDialog 的用法。

实例：显示提示消息的对话框

本程序的界面非常简单，程序界面上定义了 1 个文本框和 6 个按钮，每次用户单击一个按

钮时，将会显示不同类型的对话框。

本实例的界面只包含一个简单的文本框和几个按钮。当用户单击按钮时将会显示对话框。由于界面十分简单，故此处不再给出界面布局文件。

本程序将会通过上面介绍的 6 步来创建 AlertDialog 对话框。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public void simple(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("简单对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        .setMessage("对话框的测试内容\n 第二行内容");
        // 为AlertDialog.Builder 添加“确定”按钮
        setPositiveButton(builder);
        // 为AlertDialog.Builder 添加“取消”按钮
        setNegativeButton(builder)
        .create()
        .show();
}
```

上面程序中的粗体字代码为该对话框设置了图标、标题等属性。上面程序中还调用了 setPositiveButton()和 setNegativeButton()方法添加按钮。这两个方法的代码如下。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
private AlertDialog.Builder setPositiveButton(
    AlertDialog.Builder builder)
{
    // 调用 setPositiveButton 方法添加“确定”按钮
    return builder.setPositiveButton("确定", new OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog, int which)
        {
            show.setText("单击了【确定】按钮！");
        }
    });
}
private AlertDialog.Builder setNegativeButton(
    AlertDialog.Builder builder)
{
    // 调用 setNegativeButton 方法添加“取消”按钮
    return builder.setNegativeButton("取消", new OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog, int which)
        {
            show.setText("单击了【取消】按钮！");
        }
    });
}
```

除此之外，AlertDialog.Builder 还提供了如下方法来添加按钮。

- setNeutralButton(CharSequence text, DialogInterface. OnClick Listener listener)：添加一个装饰性按钮。

因此 Android 的对话框一共可以生成三个按钮。

运行上面的程序，单击“简单对话框”按钮，将看到如图 2.70 所示的界面。

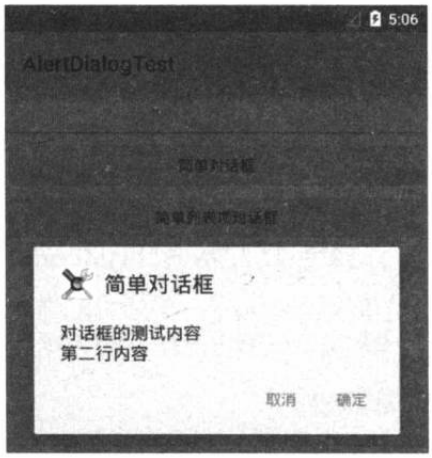


图 2.70 简单对话框

实例：简单列表项对话框

调用 AlertDialog.Builder 的 setItems()方法即可设置简单列表项对话框，调用该方法时需要传入一个数组或数组资源的资源 ID。

下面的程序调用 setItems()来设置简单列表项对话框。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public void simpleList(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("简单列表项对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置简单的列表项内容
        .setItems(items, new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                show.setText("你选中了《" + items[which] + "》");
            }
        });
        // 为 AlertDialog.Builder 添加“确定”按钮
        setPositiveButton(builder);
        // 为 AlertDialog.Builder 添加“取消”按钮
        setNegativeButton(builder)
        .create()
        .show();
}
```

上面程序中的粗体字代码调用 AlertDialog.Builder 的 setItems()方法为对话框设置了多个列表项，此处生成了 4 个普通列表项。

运行上面的程序，单击“简单列表项对话框”按钮，程序将显示如图 2.71 所示的界面。

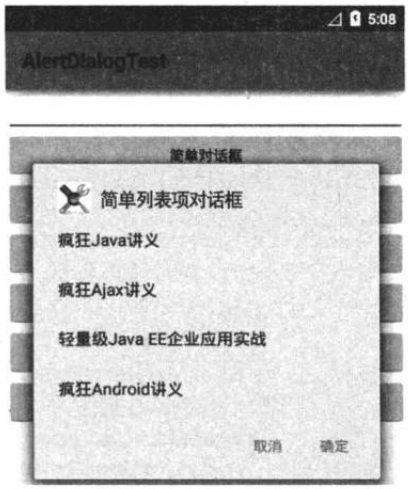


图 2.71 简单列表项对话框

实例：单选列表项对话框

只要调用 AlertDialog.Builder 的 setSingleChoiceItems()

方法即可创建带单选列表项的对话框。调用 setSingleChoiceItems()方法时既可传入数组作为参数，也可传入 Cursor（相当于数据库查询结果集）作为参数，还可传入 ListAdapter 作为参数。如果传入 ListAdapter 作为参数，则由 ListAdapter 来提供多个列表项组件。

下面的程序调用了 setSingleChoiceItems()方法来创建带单选列表项的对话框。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public void singleChoice(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("单选列表项对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置单选列表项，默认选中第二项（索引为 1）
        .setSingleChoiceItems(items, 1, new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                show.setText("你选中了《" + items[which] + "》");
            }
        });
    // 为 AlertDialog.Builder 添加“确定”按钮
    setPositiveButton(builder);
    // 为 AlertDialog.Builder 添加“取消”按钮
    setNegativeButton(builder)
        .create()
        .show();
}
```

运行上面的程序，单击“单选列表项对话框”按钮，应用显示如图 2.72 所示的界面。

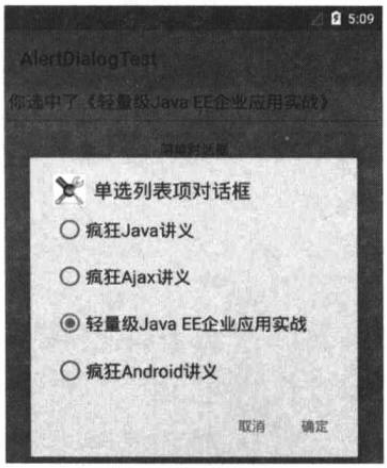


图 2.72 单选列表项对话框

实例：多选列表项对话框

只要调用 AlertDialog.Builder 的 setMultiChoiceItems()方法即可创建一个多选列表项对话框。调用 setMultiChoiceItems()方法时既可传入数组作为参数，也可传入 Cursor（相当于数据库查询结果集）作为参数。

下面的程序调用了 setMultiChoiceItems()方法来创建带多选列表项的对话框。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public void multiChoice(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("多选列表项对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置多选列表项，设置勾选第 2 项、第 4 项
        .setMultiChoiceItems(items
            , new boolean[]{false , true ,false ,true}, null);
    // 为 AlertDialog.Builder 添加“确定”按钮
```

```
        setPositiveButton(builder);
        // 为 AlertDialog.Builder 添加“取消”按钮
        setNegativeButton(builder)
            .create()
            .show();
    }
```

调用 AlertDialog.Builder 的 setMultiChoiceItems()方法添加多选列表项时，需要传入一个 boolean[]参数，该参数有两个作用：①设置初始化时选中哪些列表项；②该 boolean[]类型的参数还可用于动态地获取多选列表中列表项的选中状态。

运行上面的程序，然后单击“多选列表项对话框”按钮，程序将显示如图 2.73 所示的界面。

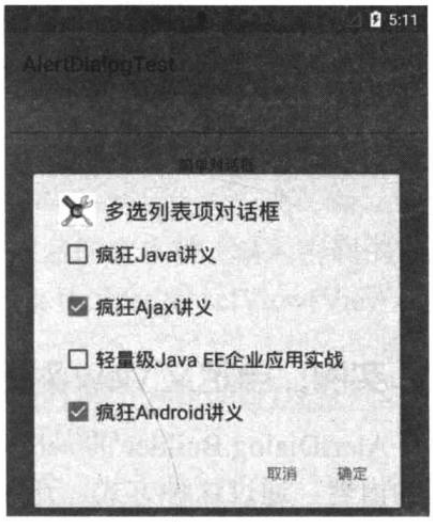


图 2.73 多选列表项对话框

实例：自定义列表项对话框

AlertDialog.Builder 提供了一个 setAdapter()方法来设置对话框的内容，该方法需要传入一个 Adapter 参数，这样即可由该 Adapter 负责提供多个列表项组件。

提示：

不仅 setAdapter()方法可以接受 Adapter 作为参数，setSingleChoice()方法也可以接受 Adapter 作为参数，这意味着调用 setSingleChoice()方法也可实现自定义列表项对话框。

下面的程序调用了 setAdapter()方法来创建自定义列表项对话框。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public void customList(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("自定义列表项对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置自定义列表项
        .setAdapter(new ArrayAdapter<String>(this
                , R.layout.array_item
                , items), null);
    // 为 AlertDialog.Builder 添加“确定”按钮
    setPositiveButton(builder);
    // 为 AlertDialog.Builder 添加“取消”按钮
    setNegativeButton(builder)
        .create()
        .show();
}
```

上面程序中的粗体字代码调用 setAdapter()方法时传入了一个 ArrayAdapter，该 ArrayAdapter 将会负责提供多个自定义列表项。如果需要，完全可以用创建 SimpleAdapter 对象或扩展 BaseAdapter 的方式来实现 Adapter，并作为参数传入 setAdapter()方法。

运行上面的程序，然后单击“自定义列表项对话框”按钮，程序将显示如图 2.74 所示的界面。

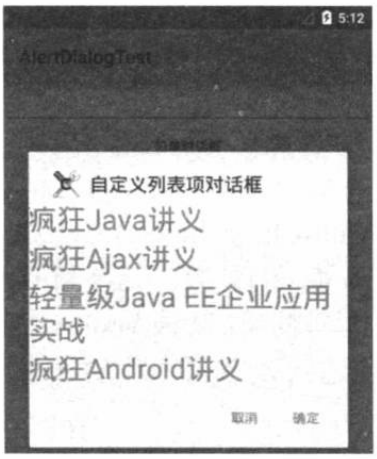


图 2.74 自定义列表项对话框

实例：自定义 View 对话框

AlertDialog.Builder 的 setView()方法可以接受一个 View 组件，该 View 组件将会作为对话框的内容，通过这种方式，开发者可以“随心所欲”地定制对话框的内容——因为在 Android 界面编程中，一切都是 View。

本实例将会定义一个登录对话框，为实现该登录对话框，先定义登录界面布局，该界面布局文件如下。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\res\layout\login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/loginForm"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="用户名:"
    android:textSize="10pt"/>
<!-- 输入用户名的文本框 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="请填写登录账号"
    android:selectAllOnFocus="true"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="密码:"
    android:textSize="10pt"/>
<!-- 输入密码的文本框 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="请填写密码"
    android:password="true"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:text="电话号码:"
    android:textSize="10pt"/>
<!-- 输入电话号码的文本框 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="请填写您的电话号码"
    android:selectAllOnFocus="true"
    android:phoneNumber="true"/>
</TableRow>
</TableLayout>
```

上面的界面布局文件定义了登录用的三个输入框：输入用户名的文本框、输入密码的密码框、输入电话号码的输入框。接下来在应用程序中调用 AlertDialog.Builder 的 setView(View view) 方法让对话框显示该输入界面。

该程序与前面介绍的列表对话框程序比较相似，只是将原来的调用 setItems()设置列表项，改为调用 setView()来设置自定义视图。下面给出该程序的关键代码。

程序清单：codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public void customView(View source)
{
    // 加载 app\src\main\res\layout\login.xml 界面布局文件
    TableLayout loginForm = (TableLayout)getLayoutInflater()
        .inflate( R.layout.login, null);
    new AlertDialog.Builder(this)
        // 设置对话框的图标
        .setIcon(R.drawable.tools)
        // 设置对话框的标题
        .setTitle("自定义 View 对话框")
        // 设置对话框显示的 View 对象
        .setView(loginForm)
        // 为对话框设置一个“确定”按钮
        .setPositiveButton("登录" , new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog,
                    int which)
            {
                // 此处可执行登录处理
            }
        })
        // 为对话框设置一个“取消”按钮
        .setNegativeButton("取消", new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog,
                    int which)
            {
                // 取消登录，不做任何事情
            }
        })
        // 创建并显示对话框
        .create()
        .show();
}
```

上面程序中的第一行粗体字代码显式加载了/app\src\main\res/layout/login.xml 文件，并返回该文件对应的 TableLayout 作为 View。接下来程序调用了 AlertDialog.Builder 的 setView()方

法来显示上一行代码所获得的 TableLayout。

运行上面的程序，然后单击“自定义 View 对话框”按钮，应用程序将显示如图 2.75 所示的界面。

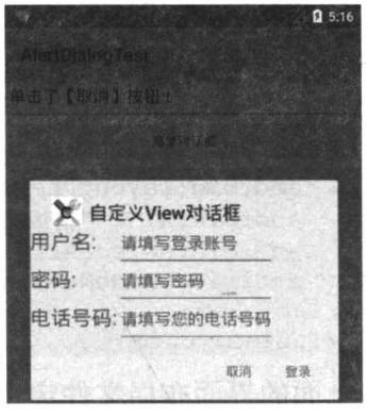


图 2.75　自定义 View 对话框

2.9.2　对话框风格的窗口

还有一种自定义对话框的方式，这种对话框本质上依然是窗口，只是把显示窗口的 Activity 的风格设为对话框风格。

下面的程序只需定义一个简单的界面布局，该界面布局里包含一个 ImageView 和一个 Button。接下来程序使用 Activity 来显示该界面布局。

程序清单：codes\02\2.9\DialogTheme\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 结束该 Activity
                finish();
            }
        });
    }
}
```

上面的程序仅仅是为界面上的按钮绑定了一个监听器，当该按钮被单击时结束该 Activity。

接下来在 AndroidManifest.xml 文件中指定该窗口以对话框风格显示，也就是在该清单文件中使用如下配置片段：

```
<activity android:name=".DialogTheme "
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Material.Dialog">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

上面的粗体字代码指定 DialogTheme 使用对话框风格进行显示。运行上面的程序，将看到如图 2.76 所示的界面。

图 2.76 对话框风格的窗口

2.9.3 使用 PopupWindow

PopupWindow 可以创建类似对话框风格的窗口，使用 PopupWindow 创建对话框风格的窗口只要如下两步即可。

1. 调用 PopupWindow 的构造器创建 PopupWindow 对象。
2. 调用 PopupWindow 的 showAsDropDown(View v)将 PopupWindow 作为 v 组件的下拉组件显示出来；或调用 PopupWindow 的 showAtLocation()方法将 PopupWindow 在指定位置显示出来。

下面的程序示范了如何使用 PopupWindow 创建对话框风格的窗口。主程序中只有一个简单的按钮，用户单击该按钮时将会显示 PopupWindow；其中 PopupWindow 负责加载并显示前一个示例的窗口界面。

程序清单：codes\02\2.9\PopupWindowTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 加载 R.layout.popup 对应的界面布局文件
        View root = this.getLayoutInflater().inflate(R.layout.popup, null);
        // 创建 PopupWindow 对象
        final PopupWindow popup = new PopupWindow(root, 560, 720);
        Button button = (Button) findViewById(R.id.bn);
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 以下拉方式显示
                // popup.showAsDropDown(v);
                // 将 PopupWindow 显示在指定位置
                popup.showAtLocation(findViewById(R.id.bn),
                    Gravity.CENTER, 20, 20);
            }
        });
        // 获取 PopupWindow 中的“关闭”按钮
        root.findViewById(R.id.close).setOnClickListener(
            new View.OnClickListener()
            {
```

```
            public void onClick(View v)
            {
                // 关闭 PopupWindow
                popup.dismiss(); // ①
            }
        });
    }
}
```

图 2.77　PopupWindow 示例效果

上面程序中的第一行粗体字代码用于创建 PopupWindow 对象，接下来的两行粗体字代码分别示范了两种显示 PopupWindow 的方式：以下拉方式显示和在指定位置显示，这就是在程序中创建并显示 PopupWindow 的关键代码。程序中①号粗体字代码负责销毁、隐藏 PopupWindow 对象——当用户单击 PopupWindow 中的“关闭”按钮时，该 PopupWindow 将会关闭。

运行上面的程序，将可以看到如图 2.78 所示的结果。

从图 2.77 所示效果不难看出，PopupWindow 非常适合显示一些需要浮动显示的内容。

2.9.4　使用 DatePickerDialog、TimePickerDialog

DatePickerDialog 与 TimePickerDialog 的功能比较简单，用法也简单，只要如下两步即可。

① 通过 new 关键字创建 DatePickerDialog、TimePickerDialog 实例，调用它们的 show() 方法即可将日期选择对话框、时间选择对话框显示出来。

② 为 DatePickerDialog、TimePickerDialog 绑定监听器，这样可以保证用户通过 DatePickerDialog、TimePickerDialog 设置事件时触发监听器，从而通过监听器来获取用户设置的事件。

下面程序的界面上定义了两个按钮，其中一个按钮用于打开日期选择对话框；另一个按钮用于打开时间选择对话框。该程序的界面布局文件非常简单，这里不再给出。该程序的 Java 代码如下。

程序清单：codes\02\2.9\DateDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button dateBn = (Button)findViewById(R.id.dateBn);
        Button timeBn = (Button)findViewById(R.id.timeBn);
        // 为“设置日期”按钮绑定监听器
        dateBn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Calendar c = Calendar.getInstance();
                // 直接创建一个 DatePickerDialog 对话框实例，并将它显示出来
                new DatePickerDialog(MainActivity.this,
                    // 绑定监听器
                    new DatePickerDialog.OnDateSetListener()
                    {
```

```
                    @Override
                    public void onDateSet(DatePicker dp, int year,
                        int month, int dayOfMonth)
                    {
                        EditText show = (EditText) findViewById(R.id.show);
                        show.setText("您选择了：" + year + "年" + (month + 1)
                            + "月" + dayOfMonth + "日");
                    }
                }
                // 设置初始日期
                , c.get(Calendar.YEAR)
                , c.get(Calendar.MONTH)
                , c.get(Calendar.DAY_OF_MONTH)).show();
            }
        });
        // 为“设置时间”按钮绑定监听器
        timeBn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Calendar c = Calendar.getInstance();
                // 创建一个 TimePickerDialog 实例，并把它显示出来
                new TimePickerDialog(MainActivity.this,
                    // 绑定监听器
                    new TimePickerDialog.OnTimeSetListener()
                    {
                        @Override
                        public void onTimeSet(TimePicker tp, int hourOfDay,
                            int minute)
                        {
                            EditText show = (EditText) findViewById(R.id.show);
                            show.setText("您选择了：" + hourOfDay + "时"
                                + minute + "分");
                        }
                    }
                    // 设置初始时间
                    , c.get(Calendar.HOUR_OF_DAY)
                    , c.get(Calendar.MINUTE)
                    // true 表示采用 24 小时制
                    , true).show();
            }
        });
    }
}
```

上面程序中的两段粗体字代码就是创建并显示 DatePickerDialog、TimePickerDialog 的关键代码。运行上面的程序，如果单击“设置时间”按钮，系统将会显示如图 2.78 所示的时间选择对话框。

如果用户单击“设置日期”按钮，系统将会显示日期选择对话框。需要指出的是，日期选择对话框、时间选择对话框只是供用户来选择日期、时间，对 Android 的系统日期、时间没有任何影响。

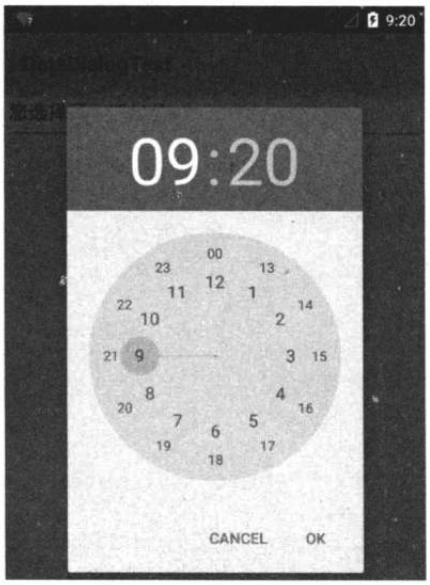


图 2.78 时间选择对话框

提示：

Android 暂时还没有公开设置系统日期、时间的 API，如果需要在程序中设置 Android 系统日期、时间，目前所知的方式都需要重新编译 Android 系统源代码，比较烦琐。

2.9.5 使用 ProgressDialog 创建进度对话框

ProgressDialog 代表了进度对话框，程序只要创建 ProgressDialog 实例，并将它显示出来就是一个进度对话框。使用 ProgressDialog 创建进度对话框有如下两种方式。

- 如果只是创建简单的进度对话框，那么调用 ProgressDialog 提供的静态 show()方法显示对话框即可。
- 创建 ProgressDialog，然后调用方法对对话框里的进度条进行设置，设置完成后将对话框显示出来即可。

为了对进度对话框里的进度条进行设置，ProgressDialog 包含了如下常用的方法。

- setIndeterminate(boolean indeterminate)：设置对话框里的进度条不显示进度值。
- setMax(int max)：设置对话框里进度条的最大值。
- setMessage(CharSequence message)：设置对话框里显示的消息。
- setProgress(int value)：设置对话框里进度条的进度值。
- setProgressStyle(int style)：设置对话框里进度条的风格。

下面的程序界面也很简单，界面上只有三个简单的按钮，当用户单击不同按钮时系统将会启动不同的进度对话框。其中第三个按钮激发的进度对话框比较复杂，该对话框使用填充数组来模拟耗时任务，随着任务进行不断更新进度对话框上进度的显示。

程序清单：codes\02\2.9\ProgressDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    final static int MAX_PROGRESS = 100;
    // 该程序模拟填充长度为 100 的数组
    private int[] data = new int[50];
    // 记录进度对话框的完成百分比
    int progressStatus = 0;
    int hasData = 0;
    ProgressDialog pd1,pd2;
    // 定义一个负责更新进度的 Handler
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            // 表明消息是由该程序发送的
            if (msg.what == 0x123)
            {
                pd2.setProgress(progressStatus);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void showSpinner(View source)
```

```
    {
        // 调用静态方法显示环形进度条
        ProgressDialog.show(this, "任务执行中"
            , "任务执行中，请等待", false, true); // ①
    }
    public void showIndeterminate(View source)
    {
        pd1 = new ProgressDialog(MainActivity.this);
        // 设置对话框的标题
        pd1.setTitle("任务正在执行中");
        // 设置对话框显示的内容
        pd1.setMessage("任务正在执行中，敬请等待...");
        // 设置对话框能用“取消”按钮关闭
        pd1.setCancelable(true);
        // 设置对话框的进度条风格
        pd1.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        // 设置对话框的进度条是否显示进度
        pd1.setIndeterminate(true);
        pd1.show(); // ②
    }
    public void showProgress(View source)
    {
        // 将进度条的完成进度重设为 0
        progressStatus = 0;
        // 重新开始填充数组
        hasData = 0;
        pd2 = new ProgressDialog(MainActivity.this);
        pd2.setMax(MAX_PROGRESS);
        // 设置对话框的标题
        pd2.setTitle("任务完成百分比");
        // 设置对话框显示的内容
        pd2.setMessage("耗时任务的完成百分比");
        // 设置对话框不能用“取消”按钮关闭
        pd2.setCancelable(false);
        // 设置对话框的进度条风格
        pd2.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        // 设置对话框的进度条是否显示进度
        pd2.setIndeterminate(false);
        pd2.show(); // ③
        new Thread()
        {
            public void run()
            {
                while (progressStatus < MAX_PROGRESS)
                {
                    // 获取耗时操作的完成百分比
                    progressStatus = MAX_PROGRESS
                        * doWork() / data.length;
                    // 发送空消息到 Handler
                    handler.sendEmptyMessage(0x123);
                }
                // 如果任务已经完成
                if (progressStatus >= MAX_PROGRESS)
                {
                    // 关闭对话框
                    pd2.dismiss();
                }
            }
        }.start();
    }
    // 模拟一个耗时的操作
```

```
    public int doWork()
    {
        // 为数组元素赋值
        data[hasData++] = (int) (Math.random() * 100);
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return hasData;
    }
}
```

上面程序为三个按钮定义了事件处理方法，其中一个按钮的事件处理方法调用了ProgressDialog的静态show()方法创建并显示对话框，如上面①号粗体字代码所示，这也是使用对话框的最简单方式；第二个按钮的事件处理方法先创建了ProgressDialog对象，再调用该对话框的show()方法将它显示出来，如上面②号粗体字代码所示；第三个按钮的事件处理方法也是先创建ProgressDialog对象，并设置其中对话框的相关属性，再调用show()方法将它显示出来，如上面③号粗体字代码所示。

单击程序中的第一个按钮，将看到如图2.79所示的界面。

单击第三个按钮，将可以看到如图2.80所示的界面。

图2.79　环形风格的进度条

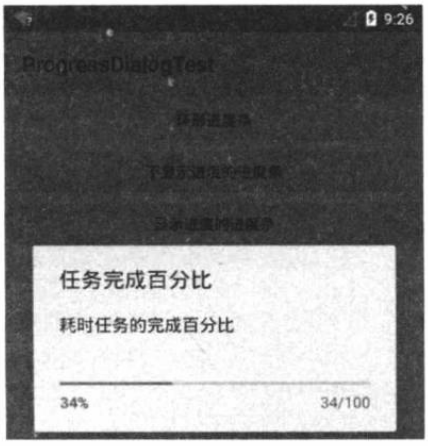


图2.80　水平风格的进度条

2.10　菜单

菜单在桌面应用中使用十分广泛，几乎所有的桌面应用都有菜单。菜单在手机应用中的使用减少了不少（主要受到手机屏幕大小制约），但依然有不少手机应用会添加菜单。

与桌面应用的菜单不同，Android应用中的菜单默认是看不见的，只有当用户按下手机上的“MENU”键（位于模拟器右边的物理键盘上）时，系统才会显示该应用关联的菜单，这种菜单叫选项菜单（Option Menu）。

注意：

从Android 3.0开始，Android并不要求手机设备上必须提供MENU按键，可能部分Android手机将不再提供MENU按键。在这样的情况下，Android推荐使用ActionBar来代替菜单。下一节会介绍Android的ActionBar支持。

Android 应用同样支持上下文菜单（ContextMenu），当用户一直按住某个组件时，该组件所关联的上下文菜单就会显示出来。

2.10.1 选项菜单和子菜单（SubMenu）

为了让读者感受 Android 应用中菜单的外观和功能，先简单看一下 Android 系统自带的选项菜单，按如下两步进行。

1. 单击模拟器右边的按键（返回桌面），系统返回桌面。
2. 单击模拟器右边的“MENU”按键，将可以在手机屏幕下方看到系统默认的选项菜单，如图 2.81 所示。

从图 2.81 可以看出，Android 的选项菜单默认是看不见的，当用户按下“MENU”按键时程序菜单将会出现在屏幕下方。

Android 系统的菜单支持主要通过 4 个接口来体现，图 2.82 显示了 Android 菜单支持的 4 个接口。

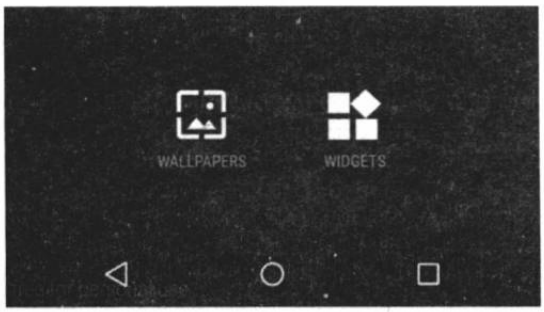


图 2.81 选项菜单

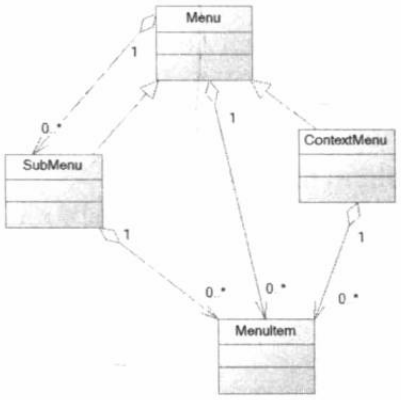


图 2.82 Android 系统的菜单支持接口

从图 2.82 可以看出，Menu 接口只是一个父接口，该接口下有如下两个子接口。

- SubMenu：它代表一个子菜单。可以包含 1～*N* 个 MenuItem（形成菜单项）。
- ContextMenu：它代表一个上下文菜单。可以包含 1～*N* 个 MenuItem（形成菜单项）。

Android 的不同菜单具有如下特征。

- **选项菜单**：选项菜单不支持勾选标记，并且只显示菜单的“浓缩（condensed）”标题。
- **子菜单（SubMenu）**：不支持菜单项图标，不支持嵌套子菜单。
- **上下文菜单（ContextMenu）**：不支持菜单快捷键和图标。

Menu 接口定义了如下方法来添加子菜单或菜单项。

- MenuItem add(int titleRes)：添加一个新的菜单项。
- MenuItem add(int groupId, int itemId, int order, CharSequence title)：添加一个新的处于 groupId 组的菜单项。
- MenuItem add(int groupId, int itemId, int order, int titleRes)：添加一个新的处于 groupId 组的菜单项。
- MenuItem add(CharSequence title)：添加一个新的菜单项。
- SubMenu addSubMenu(int titleRes)：添加一个新的子菜单。
- SubMenu addSubMenu(int groupId, int itemId, int order, int titleRes)：添加一个新的处于 groupId 组的子菜单。

- SubMenu addSubMenu(CharSequence title)：添加一个新的子菜单。
- SubMenu addSubMenu(int groupId, int itemId, int order, CharSequence title)：添加一个新的处于 groupId 组的子菜单。

上面的方法归纳起来就是两个：add()方法用于添加菜单项；addSubMenu()用于添加子菜单。这些重载方法的区别是：是否将子菜单、菜单项添加到指定菜单组中，是否使用资源文件中的字符串资源来设置标题。

SubMenu 继承了 Menu，它就代表一个子菜单，额外提供了如下常用方法。

- SubMenu setHeaderIcon(Drawable icon)：设置菜单头的图标。
- SubMenu setHeaderIcon(int iconRes)：设置菜单头的图标。
- SubMenu setHeaderTitle(int titleRes)：设置菜单头的标题。
- SubMenu setHeaderTitle(CharSequence title)：设置菜单头的标题。
- SubMenu setHeaderView(View view)：使用 View 来设置菜单头。

掌握了上面 Menu、SubMenu、MenuItem 的用法之后，接下来就可通过它们来为 Android 应用添加菜单或子菜单了。添加菜单或子菜单的步骤如下。

① 重写 Activity 的 onCreateOptionsMenu(Menu menu)方法，在该方法里调用 Menu 对象的方法来添加菜单项或子菜单。

② 如果希望应用程序能响应菜单项的单击事件，那么重写 Activity 的 onOptionsItemSelected (MenuItem mi)方法即可。

下面的程序示范了如何为 Android 应用添加菜单和子菜单。该程序的界面布局很简单，故不再给出界面布局文件。该程序的 Java 代码如下。

程序清单：codes\02\2.10\MenuTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义“字体大小”菜单项的标识
    final int FONT_10 = 0x111;
    final int FONT_12 = 0x112;
    final int FONT_14 = 0x113;
    final int FONT_16 = 0x114;
    final int FONT_18 = 0x115;
    // 定义“普通菜单项”的标识
    final int PLAIN_ITEM = 0x11b;
    // 定义“字体颜色”菜单项的标识
    final int FONT_RED = 0x116;
    final int FONT_BLUE = 0x117;
    final int FONT_GREEN = 0x118;
    private EditText edit;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        edit = (EditText) findViewById(R.id.txt);
    }
    // 当用户单击 MENU 键时触发该方法
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // -------------向 menu 中添加“字体大小”的子菜单-------------
        SubMenu fontMenu = menu.addSubMenu("字体大小");
        // 设置菜单的图标
        fontMenu.setIcon(R.drawable.font);
        // 设置菜单头的图标
```

```
        fontMenu.setHeaderIcon(R.drawable.font);
        // 设置菜单头的标题
        fontMenu.setHeaderTitle("选择字体大小");
        fontMenu.add(0, FONT_10, 0, "10 号字体");
        fontMenu.add(0, FONT_12, 0, "12 号字体");
        fontMenu.add(0, FONT_14, 0, "14 号字体");
        fontMenu.add(0, FONT_16, 0, "16 号字体");
        fontMenu.add(0, FONT_18, 0, "18 号字体");
        // -------------向 menu 中添加“普通菜单项”-------------
        menu.add(0, PLAIN_ITEM, 0, "普通菜单项");
        // -------------向 menu 中添加“字体颜色”的子菜单-------------
        SubMenu colorMenu = menu.addSubMenu("字体颜色");
        colorMenu.setIcon(R.drawable.color);
        // 设置菜单头的图标
        colorMenu.setHeaderIcon(R.drawable.color);
        // 设置菜单头的标题
        colorMenu.setHeaderTitle("选择文字颜色");
        colorMenu.add(0, FONT_RED, 0, "红色");
        colorMenu.add(0, FONT_GREEN, 0, "绿色");
        colorMenu.add(0, FONT_BLUE, 0, "蓝色");
        return super.onCreateOptionsMenu(menu);
    }
    @Override
    // 选项菜单的菜单项被单击后的回调方法
    public boolean onOptionsItemSelected(MenuItem mi)
    {
        // 判断单击的是哪个菜单项，并有针对性地做出响应
        switch (mi.getItemId())
        {
            case FONT_10:
                edit.setTextSize(10 * 2);
                break;
            case FONT_12:
                edit.setTextSize(12 * 2);
                break;
            case FONT_14:
                edit.setTextSize(14 * 2);
                break;
            case FONT_16:
                edit.setTextSize(16 * 2);
                break;
            case FONT_18:
                edit.setTextSize(18 * 2);
                break;
            case FONT_RED:
                edit.setTextColor(Color.RED);
                break;
            case FONT_GREEN:
                edit.setTextColor(Color.GREEN);
                break;
            case FONT_BLUE:
                edit.setTextColor(Color.BLUE);
                break;
            case PLAIN_ITEM:
                Toast toast = Toast.makeText(MainActivity.this
                    , "您单击了普通菜单项" , Toast.LENGTH_SHORT);
                toast.show();
                break;
        }
        return true;
    }
}
```

上面程序中的粗体字代码就是添加三个菜单的代码，三个菜单中有两个带有子菜单，而且程序还为子菜单设置了图标、标题等。运行上面的程序，单击“MENU”按键，将看到程序下方显示如图 2.83 所示的菜单。

图 2.83 所示菜单中“字体大小”包含子菜单，“普通菜单项”只是一个菜单项，“字体颜色”也包含子菜单，如果开发者单击“字体颜色”菜单，将看到屏幕上显示如图 2.84 所示的子菜单。

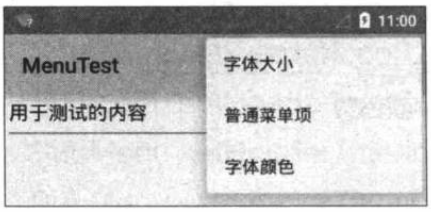


图 2.83 选项菜单

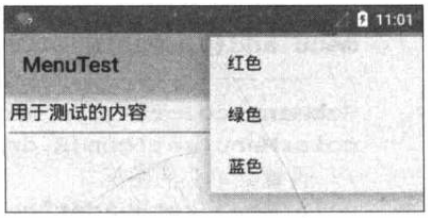


图 2.84 子菜单

提示：

Android 5.0 采用了 Material 主题，这种主题不会显示子菜单的标题，因此图 2.84 中的子菜单看不到菜单标题。

由于程序重写了 onOptionsItemSelected(MenuItem mi)方法，因此当用户单击指定菜单项时，程序可以为菜单项的单击事件提供响应。

前面已经介绍过，如果程序要监听菜单项的单击事件，可以通过重写 onOptionsItemSelected(MenuItem item)方法来实现——每当用户单击任意菜单时，该方法都会被触发。如果开发者需要针对不同菜单提供响应，就需要在 onOptionsItemSelected(MenuItem item)方法中进行判断了，如上面的程序使用 switch 语句进行了判断。由于程序需要在 onOptionsItemSelected(MenuItem item)方法中准确判断到底是哪个菜单项被单击了，因此添加菜单项时通常应为每个菜单项指定 ID。

2.10.2 使用监听器来监听菜单事件

除了重写 onOptionsItemSelected(MenuItem item)方法来为菜单单击事件编写响应之外，Android 同样允许开发者为不同菜单项分别绑定监听器。为菜单项绑定监听器的方法为：

- setOnMenuItemClickListener(MenuItem.OnMenuItemClickListener menuItemClickListener)

在这种方式下，我们可以采用简单方法来添加菜单项，无须为每个菜单项指定 ID。

一般来说，通过重写 onOptionsItemSelected(MenuItem mi)方法来使处理菜单的单击事件更加简捷，因为所有的事件处理代码都控制在该方法内，只需要判断到底单击了哪个菜单项就行。通过为每个菜单项绑定事件监听器使得代码更加臃肿，因此，一般并不推荐为每个菜单项分别绑定监听器。

2.10.3 创建多选菜单项和单选菜单项

如果希望所创建的菜单项是单选菜单项或多选菜单项，则可以调用 MenuItem 的如下方法。

- setCheckable(boolean checkable)：设置该菜单项是否可以被勾选。

调用上面的方法后，菜单项默认是多选菜单项。

如果希望将一组菜单里的菜单项都设为可勾选的菜单项，则可调用如下方法。

- setGroupCheckable(int group, boolean checkable, boolean exclusive)：设置 group 组里的所有菜单项是否可勾选；如果将 exclusive 设为 true，那么它们将是一组单选菜单项。

除此之外，Android 还为 MenuItem 提供了如下方法来设置快捷键。

- setAlphabeticShortcut(char alphaChar)：设置字母快捷键。
- setNumericShortcut(char numericChar)：设置数字快捷键。
- setShortcut(char numericChar, char alphaChar)：同时设置两种快捷键。

2.10.4 设置与菜单项关联的 Activity

有些时候，应用程序需要单击某个菜单项时启动其他 Activity（包括其他 Service）。对于这种需求，Android 甚至不需要开发者编写任何事件处理代码，只要调用 MenuItem 的 setIntent(Intent intent)方法即可——该方法把该菜单项与指定 Intent 关联到一起，当用户单击该菜单项时，该 Intent 所代表的组件将会被启动。

如下程序示范了如何通过菜单项来启动指定 Activity。该程序几乎不包含任何界面组件，因此不给出界面布局文件。该程序的 Java 文件如下。

程序清单：codes\02\2.10\ActivityMenu\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // -------------向 menu 中添加子菜单-------------
        SubMenu prog = menu.addSubMenu("启动程序");
        // 设置菜单头的图标
        prog.setHeaderIcon(R.drawable.tools);
        // 设置菜单头的标题
        prog.setHeaderTitle("选择您要启动的程序");
        // 添加菜单项
        MenuItem item = prog.add("查看 Swift");
        // 为菜单项设置关联的 Activity
        item.setIntent(new Intent(this , OtherActivity.class));
        return super.onCreateOptionsMenu(menu);
    }
}
```

上面程序中的粗体字代码表示单击指定菜单项时启动 OtherActivity，因此程序还应该定义 OtherActivity，并在 AndroidManifest.xml 文件中添加如下代码来声明 OtherActivity：

```
<activity android:name=".OtherActivity"
    android:label="查看《疯狂 Swift 讲义》">
</activity>
```

运行上面的程序，打开“启动程序”菜单，可以看到如图 2.85 所示的子菜单。

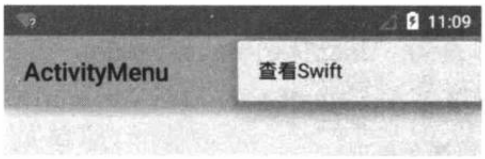


图 2.85 启动 Activity 的菜单项

单击图 2.85 所示子菜单中的“查看 Swift”，即可启动另一个 Activity：OtherActivity。

2.10.5 上下文菜单

Android 用 ContextMenu 来代表上下文菜单，为 Android 应用开发上下文菜单与开发选项菜单的方法基本相似，因为 ContextMenu 继承了 Menu，因此程序可用相同的方法为它添加菜单项。

当然，开发上下文菜单与开发选项菜单的区别在于：开发上下文菜单不是重写 onCreateOptionsMenu(Menu menu)方法，而是重写 onCreateContextMenu (ContextMenu menu, View source, ContextMenu.ContextMenuInfo menuInfo)方法。其中 source 参数代表触发上下文菜单的组件。

开发上下文菜单的步骤如下。

① 重写 Activity 的 onCreateContextMenu(ContextMenu menu, View source, ContextMenu. Context MenuInfo menuInfo)方法。

② 调用 Activity 的 registerForContextMenu(View view)方法为 view 组件注册上下文菜单。

③ 如果希望应用程序能为菜单项提供响应，则可以重写 onContextItemSelected(MenuItem mi)方法，或为指定菜单项绑定事件监听器。

ContextMenu 提供了如下方法，同样可以为上下文菜单设置图标、标题等。

- ContextMenu setHeaderIcon(Drawable icon)：为上下文菜单设置图标。
- ContextMenu setHeaderIcon(int iconRes)：为上下文菜单设置图标。
- ContextMenu setHeaderTitle(int titleRes)：为上下文菜单设置标题。

下面的程序示范了如何开发上下文菜单。该程序的用户界面同样很简单，故不再给出界面布局文件。下面是该程序的 Java 代码。

程序清单：codes\02\2.10\ContextMenuTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    // 为每个菜单定义一个标识
    final int MENU1 = 0x111;
    final int MENU2 = 0x112;
    final int MENU3 = 0x113;
    private TextView txt;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txt = (TextView) findViewById(R.id.txt);
        // 为文本框注册上下文菜单
        registerForContextMenu(txt); // ①
    }
    // 创建上下文菜单时触发该方法
    @Override
    public void onCreateContextMenu(ContextMenu menu, View source,
        ContextMenu.ContextMenuInfo menuInfo)
    {
        menu.add(0, MENU1, 0, "红色");
        menu.add(0, MENU2, 0, "绿色");
        menu.add(0, MENU3, 0, "蓝色");
        // 将三个菜单项设为单选菜单项
        menu.setGroupCheckable(0, true, true);
        // 设置上下文菜单的标题、图标
        menu.setHeaderIcon(R.drawable.tools);
        menu.setHeaderTitle("选择背景色");
```

```
    }
    // 上下文菜单的菜单项被单击时触发该方法
    @Override
    public boolean onContextItemSelected(MenuItem mi)
    {
        switch (mi.getItemId())
        {
            case MENU1:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.RED);
                break;
            case MENU2:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.GREEN);
                break;
            case MENU3:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.BLUE);
                break;
        }
        return true;
    }
}
```

上面的程序重写了 onCreateContextMenu(ContextMenu menu, View source, ContextMenu. Context MenuInfo menuInfo)方法，该方法的内部为程序创建了一个上下文菜单。

程序在①号代码处调用 registerForContextMenu(txt)为 txt 组件（一个文本框组件）注册了上下文菜单，这意味着当用户长按该组件时显示上下文菜单。上下文菜单如图 2.86 所示。

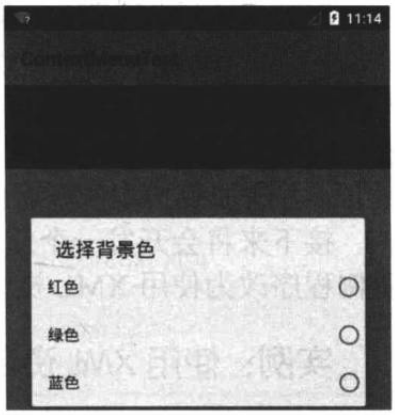


图 2.86 上下文菜单

2.10.6 使用 XML 文件定义菜单

Android 提供了两种创建菜单的方式，一种是在 Java 代码中创建，一种是使用 XML 资源文件定义。上面的示例都是在 Java 代码中创建菜单，在 Java 代码中定义菜单存在如下不足。

- 在 Java 代码中定义菜单、菜单项，必然导致程序代码臃肿。
- 需要程序员采用硬编码方式为每个菜单项分配 ID，为每个菜单组分配 ID，这种方式将导致应用可扩展性、可维护性降低。

一般推荐使用 XML 资源文件来定义菜单，这种方式可以提供更好的解耦。

提示：

使用 Android Studio 创建 Android 项目时，Android Studio 默认就会在 res 目录下新建 menu 子目录，并在该子目录下提供 menu_main.xml 菜单资源文件。由此可见，Android 推荐使用 XML 资源文件来定义菜单。

菜单资源文件通常应该放在 app\src\main\res\menu 目录下，菜单资源的根元素通常是<menu.../>，<menu.../>元素无须指定任何属性。<menu.../>元素内可包含如下子元素。

- **<item.../>元素**：定义菜单项。
- **<group.../>子元素**：将多个<item.../>定义的菜单项包装成一个菜单组。

<group.../>子元素用于控制整组菜单的行为，该元素可指定如下常用属性。

- **checkableBehavior**：指定该组菜单的选择行为。可指定为 none（不可选）、all（多

选）和 single（单选）三个值。

- menuCategory：对菜单进行分类，指定菜单的优先级。有效值为 container、system、secondary 和 alternative。
- visible：指定该组菜单是否可见。
- enable：指定该组菜单是否可用。

<item.../>元素用于定义菜单项，<item.../>元素又可包含<menu.../>元素，位于<item.../>元素内部的<menu.../>就代表了子菜单。

<item.../>元素可指定如下常用属性。

- android:id：为菜单项指定一个唯一标识。
- android:title：指定菜单项的标题。
- android:icon：指定菜单项的图标。
- android:alphabeticShortcut：为菜单项指定字符快捷键。
- android:numericShortcut：为菜单项指定数字快捷键。
- android:checkable：设置该菜单项是否可选。
- android:checked：设置该菜单项是否已选中。
- android:visible：设置该菜单项是否可见。
- android:enable：设置该菜单项是否可用。

在程序中定义了菜单资源后，接下来还是重写 onCreateOptionsMenu（用于创建选项菜单）、onCreateContextMenu（用于创建上下文菜单）方法，在这些方法中调用 MenuInflater 对象的 inflate 方法加载指定资源对应的菜单即可。

接下来将会开发一个使用 XML 资源文件定义菜单的实例，本实例将会把前面开发的菜单示例程序改为使用 XML 资源文件定义菜单。

实例：使用 XML 资源文件定义菜单

本实例包含两种菜单：选项菜单和上下文菜单，其中选项菜单对应的 XML 资源文件如下。

程序清单：codes\02\2.10\MenuResTest\app\src\main\res\menu\menu_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="@string/font_size"
        android:icon="@drawable/font">
        <menu>
            <!-- 定义一组单选菜单项 -->
            <group android:checkableBehavior="single">
                <!-- 定义多个菜单项 -->
                <item
                    android:id="@+id/font_10"
                    android:title="@string/font_10"/>
                <item
                    android:id="@+id/font_12"
                    android:title="@string/font_12"/>
                <item
                    android:id="@+id/font_14"
                    android:title="@string/font_14"/>
                <item
                    android:id="@+id/font_16"
                    android:title="@string/font_16"/>
                <item
                    android:id="@+id/font_18"
                    android:title="@string/font_18"/>
```

```
            </group>
        </menu>
    </item>
    <!-- 定义一个普通菜单项 -->
    <item android:id="@+id/plain_item"
        android:title="@string/plain_item">
    </item>
    <item android:title="@string/font_color"
        android:icon="@drawable/color">
        <menu>
            <!-- 定义一组普通菜单项 -->
            <group>
                <!-- 定义三个菜单项 -->
                <item
                    android:id="@+id/red_font"
                    android:title="@string/red_title"/>
                <item
                    android:id="@+id/green_font"
                    android:title="@string/green_title"/>
                <item
                    android:id="@+id/blue_font"
                    android:title="@string/blue_title"/>
            </group>
        </menu>
    </item>
</menu>
```

上面的菜单资源文件的<menu.../>元素里包含三个<item.../>子元素，这表明该菜单里包含三个菜单项。其中第一个、第三个菜单项都包含子菜单。

接下来为该应用定义上下文菜单的资源文件，代码如下。

程序清单：codes\02\2.10\MenuResTest\app\src\main\res\menu\context.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 定义一组单选菜单项 -->
    <group android:checkableBehavior="single">
        <!-- 定义三个菜单项 -->
        <item
            android:id="@+id/red"
            android:title="@string/red_title"
            android:alphabeticShortcut="r"/>
        <item
            android:id="@+id/green"
            android:title="@string/green_title"
            android:alphabeticShortcut="g"/>
        <item
            android:id="@+id/blue"
            android:title="@string/blue_title"
            android:alphabeticShortcut="b"/>
    </group>
</menu>
```

定义了上面两份菜单资源之后，接下来即可在 Activity 的 onCreateOptionsMenu、onCreateContextMenu 方法中加载这两份菜单资源。下面是程序中加载并显示两份菜单资源的 Java 代码。

程序清单：codes\02\2.10\MenuResTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    private TextView txt;
    @Override
```

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    txt = (TextView) findViewById(R.id.txt);
    // 为文本框注册上下文菜单
    registerForContextMenu(txt);
}
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflator = new MenuInflater(this);
    // 装填R.menu.my_menu对应的菜单，并添加到menu中
    inflator.inflate(R.menu.menu_main, menu);
    return super.onCreateOptionsMenu(menu);
}
// 创建上下文菜单时触发该方法
@Override
public void onCreateContextMenu(ContextMenu menu, View source,
    ContextMenu.ContextMenuInfo menuInfo)
{
    MenuInflater inflator = new MenuInflater(this);
    // 装填R.menu.context对应的菜单，并添加到menu中
    inflator.inflate(R.menu.context , menu);
    menu.setHeaderIcon(R.drawable.tools);
    menu.setHeaderTitle("请选择背景色");
}
// 上下文菜单中菜单项被单击时触发该方法
@Override
public boolean onContextItemSelected(MenuItem mi)
{
    // 勾选该菜单项
    mi.setChecked(true);  // ①
    switch (mi.getItemId())
    {
        case R.id.red:
            mi.setChecked(true);
            txt.setBackgroundColor(Color.RED);
            break;
        case R.id.green:
            mi.setChecked(true);
            txt.setBackgroundColor(Color.GREEN);
            break;
        case R.id.blue:
            mi.setChecked(true);
            txt.setBackgroundColor(Color.BLUE);
            break;
    }
    return true;
}
@Override
// 菜单项被单击后的回调方法
public boolean onOptionsItemSelected(MenuItem mi)
{
    if(mi.isCheckable())
    {
        // 勾选该菜单项
        mi.setChecked(true);  // ②
    }
    // 判断单击的是哪个菜单项，并有针对性地做出响应
    switch (mi.getItemId())
    {
        case R.id.font_10:
```

```
                txt.setTextSize(10 * 2);
                break;
            case R.id.font_12:
                txt.setTextSize(12 * 2);
                break;
            case R.id.font_14:
                txt.setTextSize(14 * 2);
                break;
            case R.id.font_16:
                txt.setTextSize(16 * 2);
                break;
            case R.id.font_18:
                txt.setTextSize(18 * 2);
                break;
            case R.id.red_font:
                txt.setTextColor(Color.RED);
                mi.setChecked(true);
                break;
            case R.id.green_font:
                txt.setTextColor(Color.GREEN);
                mi.setChecked(true);
                break;
            case R.id.blue_font:
                txt.setTextColor(Color.BLUE);
                mi.setChecked(true);
                break;
            case R.id.plain_item:
                Toast toast = Toast.makeText(MainActivity.this
                    , "您单击了普通菜单项" , Toast.LENGTH_SHORT);
                toast.show();
                break;
        }
        return true;
    }
}
```

上面程序中的两行粗体字代码就是加载选项菜单资源、上下文菜单资源的关键代码。

需要指出的是，可勾选菜单的勾选状态必须由程序代码来控制，所以上面程序中的①、②号代码自己控制了菜单项勾选状态的切换。这一点是笔者感到很奇怪的地方：为何 Android 系统不为我们处理这些细节？这些细节给我们编程带来了不少烦琐的处理。

从上面的程序可以看出，如果使用 XML 资源文件来定义菜单，就像使用布局文件来定义应用程序界面一样，Android 应用的 Java 代码就会简单很多，因此可维护性更好。

归纳起来，使用 XML 资源文件定义菜单有如下两个好处。

- XML 资源文件不仅负责定义应用界面，也负责定义菜单，这样可把所有界面相关的内容交给 XML 文件管理，而 Java 代码的功能更集中。
- 后期更新、维护应用时，如果需要更新、维护菜单，打开、编辑 XML 文件即可，避免对 Java 文件的修改。

该应用程序的运行效果与前面介绍的菜单示例的效果大致相同，此处不再给出。

2.10.7 使用 PopupMenu 创建弹出式菜单

PopupMenu 代表弹出式菜单，它会在指定组件上弹出 PopupMenu，在默认情况下，PopupMenu 会显示在该组件的下方或者上方。PopupMenu 可增加多个菜单项，并可为菜单项增加子菜单。

使用 PopupMenu 创建菜单的步骤非常简单，只要如下步骤即可。

① 调用 new PopupMenu(Context context, View anchor)创建下拉菜单，anchor 代表要激发

该弹出菜单的组件。

② 调用 MenuInflater 的 inflate()方法将菜单资源填充到 PopupMenu 中。

③ 调用 PopupMenu 的 show()方法显示弹出式菜单。

下面的示例示范了使用 PopupMenu 的功能和用法。这个示例的界面布局文件中仅有一个普通按钮，界面布局文件非常简单，故此处不再给出代码。

该示例的 Activity 代码如下。

程序清单：codes\02\2.10\PopupMenuTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    PopupMenu popup = null;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void onPopupButtonClick(View button)
    {
        // 创建 PopupMenu 对象
        popup = new PopupMenu(this, button);
        // 将 R.menu.popup_menu 菜单资源加载到 popup 菜单中
        getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());
        // 为 popup 菜单的菜单项单击事件绑定事件监听器
        popup.setOnMenuItemClickListener(
            new PopupMenu.OnMenuItemClickListener()
        {
            @Override
            public boolean onMenuItemClick(MenuItem item)
            {
                switch (item.getItemId())
                {
                    case R.id.exit:
                        // 隐藏该对话框
                        popup.dismiss();
                        break;
                    default:
                        // 使用 Toast 显示用户单击的菜单项
                        Toast.makeText(MainActivity.this,
                            "您单击了【" + item.getTitle() + "】菜单项"
                            , Toast.LENGTH_SHORT).show();
                }
                return true;
            }
        });
        popup.show();
    }
}
```

上面程序中的第一行粗体字代码创建了一个 PopupMenu 对象，第二行粗体字代码指定将该 R.menu.popup_menu 菜单资源填充到 PopupMenu 中，这样即可实现当用户单击界面上的按钮时弹出 popup 菜单。

运行该程序，单击程序界面上的按钮，将可以看到如图 2.87 所示的界面。

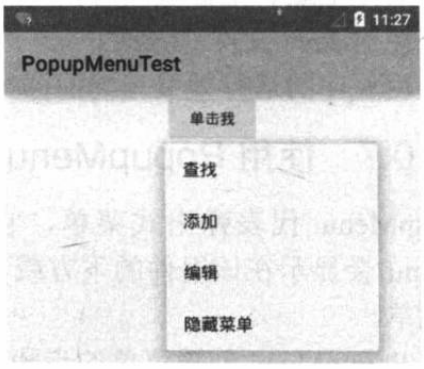


图 2.87 使用 PopupMenu 开发弹出式菜单

2.11 使用活动条（ActionBar）

活动条（ActionBar）是 Android 3.0 的重要更新之一。ActionBar 位于传统标题栏的位置，也就是显示在屏幕的顶部。ActionBar 可显示应用的图标和 Activity 标题——也就是前面应用程序的顶部显示的内容。除此之外，ActionBar 的右边还可以显示活动项（Action Item）。

归纳起来，ActionBar 提供了如下功能。

- 显示选项菜单的菜单项（将菜单项显示成 Action Item）。
- 使用程序图标作为返回 Home 主屏或向上的导航操作。
- 提供交互式 View 作为 Action View。
- 提供基于 Tab 的导航方式，可用于切换多个 Fragment。
- 提供基于下拉的导航方式。

2.11.1 启用 ActionBar

最新的 Android 版本已经默认启用了 ActionBar，因此只要在 AndroidManifest.xml 文件的 SDK 配置中指定该应用的目标版本高于 11（Android 3.0 的版本号），默认就会启用 ActionBar。例如如下配置：

```
<uses-sdk android:minSdkVersion="10"
    android:targetSdkVersion="17" />
```

指定该应用程序可以部署在 Android 4.2 平台上，同时兼容 Android 2.3.3 及更高版本。如果 Android 版本高于 3.0，该应用将会启用 ActionBar。

如果希望关闭 ActionBar，则可以设置该应用的主题为 Xxx.NoActionBar。例如如下配置片段：

```
<application android:icon="@drawable/ic_launcher"
    android:theme="@android:style/Theme.Material.NoActionBar"
    android:label="@string/app_name">
    ...
</application>
```

上面的粗体字代码指定该应用关闭 ActionBar 功能。一旦关闭了 ActionBar，该 Android 应用将不能使用 ActionBar。

在实际项目中，通常推荐使用代码来控制 ActionBar 的显示、隐藏。ActionBar 提供了如下方法来控制显示、隐藏。

- show()：显示 ActionBar。
- hide()：隐藏 ActionBar。

如下示例示范了如何通过代码来控制 ActionBar 的显示、隐藏。

该示例的界面布局文件中只定义了两个按钮，界面布局文件非常简单，故此处不再给出界面布局文件代码。该示例的 Activity 代码如下。

程序清单：codes\02\2.11\ActionBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    ActionBar actionBar;
    @Override
    public void onCreate(Bundle savedInstanceState)
```

```
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取该 Activity 的 ActionBar
        // 只有当应用主题没有关闭 ActionBar 时，该代码才能返回 ActionBar
        actionBar = getActionBar();
    }
    // 为“显示 ActionBar”按钮定义事件处理方法
    public void showActionBar(View source)
    {
        // 显示 ActionBar
        actionBar.show();
    }
    // 为“隐藏 ActionBar”按钮定义事件处理方法
    public void hideActionBar(View source)
    {
        // 隐藏 ActionBar
        actionBar.hide();
    }
}
```

上面程序中的第一行粗体字代码调用了 getActionBar()方法获取该 Activity 关联的 ActionBar。接下来就可以调用 ActionBar 的方法来控制它的显示、隐藏了。

运行该程序并单击“隐藏 ActionBar”按钮，将可以看到如图 2.88 所示的界面。

图 2.88 隐藏 ActionBar

从图 2.88 可以看出，此处程序顶端的 ActionBar 已经“消失”了。

2.11.2 使用 ActionBar 显示选项菜单项

前面介绍菜单时已经指出，Android 不再强制要求手机必须提供 MENU 按键，这样可能导致用户无法打开选项菜单。为了解决这个问题，Android 已经提供了 ActionBar 作为解决方案，ActionBar 可以将选项菜单显示成 Action Item。

从 Android 3.0 开始，MenuItem 新增了如下方法。

- setShowAsAction(int actionEnum)：该方法设置是否将该菜单项显示在 ActionBar 上，作为 Action Item。

该方法支持如下参数值。

- SHOW_AS_ACTION_ALWAYS：总是将该 MenuItem 显示在 ActionBar 上。
- SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW：将该 Action View 折叠成普通菜单项。
- SHOW_AS_ACTION_IF_ROOM：当 ActionBar 位置足够时才显示 MenuItem。
- SHOW_AS_ACTION_NEVER：不将该 MenuItem 显示在 ActionBar 上。
- SHOW_AS_ACTION_WITH_TEXT：将该 MenuItem 显示在 ActionBar 上，并显示该菜单项的文本。

正如前面看到的，实际项目推荐使用 XML 资源文件来定义菜单，因此 Android 允许在 XML 菜单资源文件中为<item.../>元素指定如下属性。

- android:showAsAction：该属性的作用类似于 setShowAsAction(int actionEnum)方法。因此该属性也能支持类似于上面的属性值。

下面的程序对前面关于菜单的示例稍作修改，只是在定义菜单资源文件时为<item.../>元素增加了 android:showAsAction 属性。下面是本示例的菜单资源文件代码。

程序清单：codes\02\2.11\ActionItemTest\app\src\main\res\menu\menu_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="@string/font_size"
        android:showAsAction="always|withText"
        android:icon="@drawable/font">
        <menu>
            <!-- 定义一组单选菜单项 -->
            <group android:checkableBehavior="single">
                <!-- 定义多个菜单项 -->
                <item
                    android:id="@+id/font_10"
                    android:title="@string/font_10"/>
                <item
                    android:id="@+id/font_12"
                    android:title="@string/font_12"/>
                <item
                    android:id="@+id/font_14"
                    android:title="@string/font_14"/>
                <item
                    android:id="@+id/font_16"
                    android:title="@string/font_16"/>
                <item
                    android:id="@+id/font_18"
                    android:title="@string/font_18"/>
            </group>
        </menu>
    </item>
    <!-- 定义一个普通菜单项 -->
    <item android:id="@+id/plain_item"
        android:showAsAction="always|withText"
        android:title="@string/plain_item">
    </item>
    <item android:title="@string/font_color"
        android:showAsAction="always"
        android:icon="@drawable/color">
        <menu>
            <!-- 定义一组允许复选的菜单项 -->
            <group>
                <!-- 定义三个菜单项 -->
                <item
                    android:id="@+id/red_font"
                    android:title="@string/red_title"/>
                <item
                    android:id="@+id/green_font"
                    android:title="@string/green_title"/>
                <item
                    android:id="@+id/blue_font"
                    android:title="@string/blue_title"/>
            </group>
        </menu>
    </item>
</menu>
```

上面三行粗体字代码为选项菜单项增加了 android:showAsAction 属性，该属性可控制将这些菜单项显示在 ActionBar 上。再次运行该示例，将可以看到如图 2.89 所示的 Action Item。

正如图 2.89 所显示的，手机顶部的 ActionBar 的空间是有限的，当选项菜单的菜单项很多时，ActionBar 无法同时显示所有的选项菜单项，Android 将会根据不同手机设备采取不同行为。

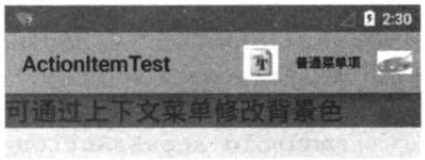


图 2.89　使用 ActionBar 显示选项菜单项

- 对于有 MENU 按键的手机，用户单击 MENU 按键即可看到剩余的选项菜单项。
- 对于没有 MENU 按键的手机，ActionBar 会在最后显示一个折叠图标，用户单击该折叠图标将会显示剩余的选项菜单项，类似于以前单击 MENU 按键后出现“More”按钮。

2.11.3　启用程序图标导航

为了将应用程序图标转变成可以点击的图标，可以调用 ActionBar 的如下方法。

- setDisplayHomeAsUpEnabled(boolean showHomeAsUp)：设置是否将应用程序图标转变成可点击的图标，并在图标上添加一个向左的箭头。
- setDisplayOptions(int options)：通过传入 int 类型常量来控制该 ActionBar 的显示选项。
- setDisplayShowHomeEnabled(boolean showHome)：设置是否显示应用程序图标。
- setHomeButtonEnabled(boolean enabled)：设置是否将应用程序图标转变成可点击的按钮。

下面的程序将该 Activity 的程序图标转变成可点击的图标，并控制单击该图标时直接返回程序的主 Activity。该程序的界面布局文件、菜单资源文件保持不变。下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionHomeTest\app\src\main\java\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity
{
    private TextView txt;
    ActionBar actionBar;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txt = (TextView) findViewById(R.id.txt);
        actionBar = getActionBar();
        // 设置是否显示应用程序图标
        actionBar.setDisplayShowHomeEnabled(true);
        // 将应用程序图标设置为可点击的按钮
        // actionBar.setHomeButtonEnabled(true);
        // 将应用程序图标设置为可点击的按钮，并在图标上添加向左的箭头
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater inflator = new MenuInflater(this);
        // 状态 R.menu.context 对应的菜单，并添加到 menu 中
```

```
        inflator.inflate(R.menu.my_menu, menu);
        return super.onCreateOptionsMenu(menu);
    }
    @Override
    // 选项菜单的菜单项被单击后的回调方法
    public boolean onOptionsItemSelected(MenuItem mi)
    {
        if(mi.isCheckable())
        {
            mi.setChecked(true);
        }
        // 判断单击的是哪个菜单项，并有针对性地做出响应
        switch (mi.getItemId())
        {
            case android.R.id.home:
                // 创建启动 FirstActivity 的 Intent
                Intent intent = new Intent(this, FirstActivity.class);
                // 添加额外的 Flag，将 Activity 栈中处于 FirstActivity 之上的 Activity 弹出
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                // 启动 intent 对应的 Activity
                startActivity(intent);
                break;
            ...
        }
        return true;
    }
}
```

上面程序中的前两行粗体字代码的任意一行都可将 ActionBar 上的应用程序图标转变成可点击的图标。接下来程序为点击事件绑定了事件监听器，当用户单击 ID 为 android.R.id.home 的 Action Item（应用程序图标的 ID）时，程序使用 Intent 返回了应用程序的 FirstActivity，如上面程序中的第二段粗体字代码所示。

提示：

上面的示例中还定义了 FirstActivity、SecondActivity 两个 Activity，关于开发、定义 Activity 的详细步骤，请参考本书第 4 章。

2.11.4 添加 Action View

ActionBar 上除了可以显示普通的 Action Item 之外，还可以显示普通的 UI 组件。为了在 ActionBar 上添加 Action View，可以来用如下两种方式。

- 定义 Action Item 时使用 android:actionViewClass 属性指定 Action View 的实现类。
- 定义 Action Item 时使用 android:actionLayout 属性指定 Action View 对应的视图资源。

实例：“标题”上的时钟

本实例将会在菜单资源文件中定义两个 Action Item，但这两个 Action Item 都是使用 Action View，而不是普通的 Action Item。资源文件代码如下。

程序清单：codes\02\2.11\ActionViewTest\app\src\main\res\menu\menu_main.xml

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

```
        android:id="@+id/search"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:actionViewClass="android.widget.SearchView"/>
    <item
        android:id="@+id/progress"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:actionLayout="@layout/clock"/>
    </menu>
```

上面资源文件中的两行粗体字代码分别采用两种方式定义了 Action View，这样就可以在 ActionBar 上显示自定义 View。其中第一行粗体字代码指定了 Action View 的实现类为 SearchView；第二行粗体字代码指定该 Action View 对应的界面布局资源为@layout/clock。该布局文件代码如下。

程序清单：codes\02\2.11\ActionViewTest\app\src\main\res\layout\clock.xml

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

该界面布局文件仅仅定义了一个 AnalogClock，这表明该ActionBar的第二个Action View只是一个模拟时钟。

运行该程序，可以看到如图 2.90 所示的界面。

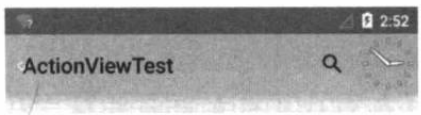


图 2.90　添加 Action View

2.11.5　使用 ActionBar 实现 Tab 导航

ActionBar 还有常用的功能：实现 Tab 导航。ActionBar 在顶端生成多个 Tab 标签，当用户点击某个 Tab 标签时，系统根据用户点击事件导航到指定 Tab 页面。

为了使用 ActionBar 实现 Tab 导航，按如下步骤进行即可。

① 调用 ActionBar 的 setNavigationMode(ActionBar.NAVIGATION_MODE_TABS)方法设置使用 Tab 导航方式。

② 调用 ActionBar 的 addTab()方法添加多个 Tab 标签，并为每个 Tab 标签添加事件监听器。

在实际项目中为了更好地展现 Tab 导航效果，ActionBar 通常会与 Fragment 结合使用，因此这里先简单介绍 Fragment 的用法。

Fragment 是 Android 3.0 新增的重要 API，Fragment 相当于 Activity 片段（Fragment 本来就是片段的意思），我们通常使用单独的 Activity 组合多个 Fragment，这样即可在一个 Activity 中创建多个用户界面。除此之外，也可让多个 Activity 复用同一个 Fragment。总之，Fragment 相当于 Activity 的模块化区域。

Fragment 拥有自己的生命周期，它也可以接收、处理属于它自身的事件，并允许 Activity 运行期间动态地添加、删除 Fragment。

Fragment 允许定义自己的布局，也可通过生命周期回调方法定义自己的行为，这一点

Fragment 非常像 Activity。

与开发 Activity 类似的是，开发者自定义的 Fragment 也需要继承 Fragment，并重写它的生命周期方法，通常会重写 Fragment 的 onCreateView()生命周期方法。

提示：

关于 Fragment 的生命周期及更详细的介绍，请参考本书第 4 章。

下面通过实例来介绍 ActionBar 结合 Fragment 实现 Tab 导航。

实例：ActionBar 结合 Fragment 实现 Tab 导航

该实例的界面布局文件非常简单，该布局文件中只是定义了一个简单的容器，该容器甚至没有太多要求，既可使用 LinearLayout，也可使用 RelativeLayout……甚至是普通 ViewGroup，该容器只是用于盛装 Fragment。该实例的界面布局文件如下。

程序清单：codes\02\2.11\ActionBar_TabNav\res\layout\main.xml

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

上面的界面布局文件只是定义了一个 LinearLayout 作为容器，接下来 Activity 将会使用该容器动态地盛装 Fragment。下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionBar_TabNav\src\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends Activity implements
    ActionBar.TabListener
{
    private static final String SELECTED_ITEM = "selected_item";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ActionBar actionBar = getActionBar();
        // 设置 ActionBar 的导航方式：Tab 导航
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        // 依次添加三个 Tab 标签，并为三个 Tab 标签添加事件监听器
        actionBar.addTab(actionBar.newTab().setText("第一页")
            .setTabListener(this));
        actionBar.addTab(actionBar.newTab().setText("第二页")
            .setTabListener(this));
        actionBar.addTab(actionBar.newTab().setText("第三页")
            .setTabListener(this));
    }
    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState)
    {
        if (savedInstanceState.containsKey(SELECTED_ITEM))
        {
            // 选中前面保存的索引对应的 Fragment 页
```

```
            getActionBar().setSelectedNavigationItem(
                savedInstanceState.getInt(SELECTED_ITEM));
        }
    }
    @Override
    public void onSaveInstanceState(Bundle outState)
    {
        // 将当前选中的 Fragment 页的索引保存到 Bundle 中
        outState.putInt(SELECTED_ITEM,
            getActionBar().getSelectedNavigationIndex());
    }
    @Override
    public void onTabUnselected(ActionBar.Tab tab,
        FragmentTransaction fragmentTransaction)
    {
    }
    // 当指定 Tab 标签被选中时激发该方法
    @Override
    public void onTabSelected(ActionBar.Tab tab,
            FragmentTransaction fragmentTransaction)
    {
        // 创建一个新的 Fragment 对象
        Fragment fragment = new DummyFragment();
        // 创建一个 Bundle 对象，用于向 Fragment 传入参数
        Bundle args = new Bundle();
        args.putInt(DummyFragment.ARG_SECTION_NUMBER,
                tab.getPosition() + 1);
        // 向 fragment 传入参数
        fragment.setArguments(args);
        // 获取 FragmentTransaction 对象
        FragmentTransaction ft = getFragmentManager()
                .beginTransaction();
        // 使用 fragment 代替该 Activity 中的 container 组件
        ft.replace(R.id.container, fragment);
        // 提交事务
        ft.commit();
    }
    @Override
    public void onTabReselected(ActionBar.Tab tab,
            FragmentTransaction fragmentTransaction)
    {
    }
}
```

上面的第一段粗体字代码设置 ActionBar 使用 Tab 导航，为该 ActionBar 添加了三个 Tab 标签，并为每个 Tab 标签都设置了事件监听器。

注意：

Android 5.0 已将 ActionBar 导航这种方式设为过时的方式，Android 5.0 推荐使用更通用的导航方式，就是前面介绍的利用 ActionBar 上的图标导航，并未提供更有效的导航方式，因此 ActionBar 依然是一种非常重要的导航方式，所以本书依然会介绍这种方式。

当用户单击 ActionBar 上的指定 Tab 标签时，系统将会激发该监听器的 onTabSelected()方

法，因此上面的第二段粗体字代码实现了 onTabSelected()方法，并在该方法中根据用户选中的 Tab 标签替换新的 Fragment。

上面的实例用到了一个 DummyFragment，这是一个简单的 Fragment，它只是显示一个简单的 TextView 组件。下面是该 DummyFragment 的代码。

程序清单：codes\02\2.11\ActionBar_TabNav\src\org\crazyit\ui\DummyFragment.java

```
public class DummyFragment extends Fragment
{
    public static final String ARG_SECTION_NUMBER = "section_number";
    // 该方法的返回值就是该 Fragment 显示的 View 组件
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState)
    {
        TextView textView = new TextView(getActivity());
        textView.setGravity(Gravity.START);
        // 获取创建该 Fragment 时传入的参数 Bundle
        Bundle args = getArguments();
        // 设置 TextView 显示的文本
        textView.setText(args.getInt(ARG_SECTION_NUMBER) + "");
        textView.setTextSize(30);
        // 返回该 TextView
        return textView;
    }
}
```

运行该实例，可以看到如图 2.91 所示的效果。

需要指出的是，Android Studio 为 ActionBar 的 Tab 导航提供了支持，当使用 Android Studio 创建 Android 项目时，Android Studio 的向导工具在创建 Tabbed Activity 时会看到如图 2.92 所示的界面。

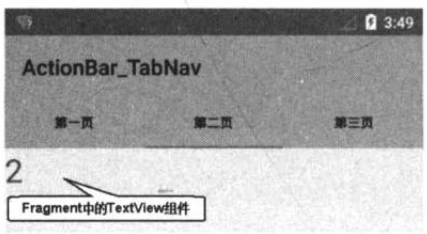


图 2.91 ActionBar 结合 Fragment 实现 Tab 导航

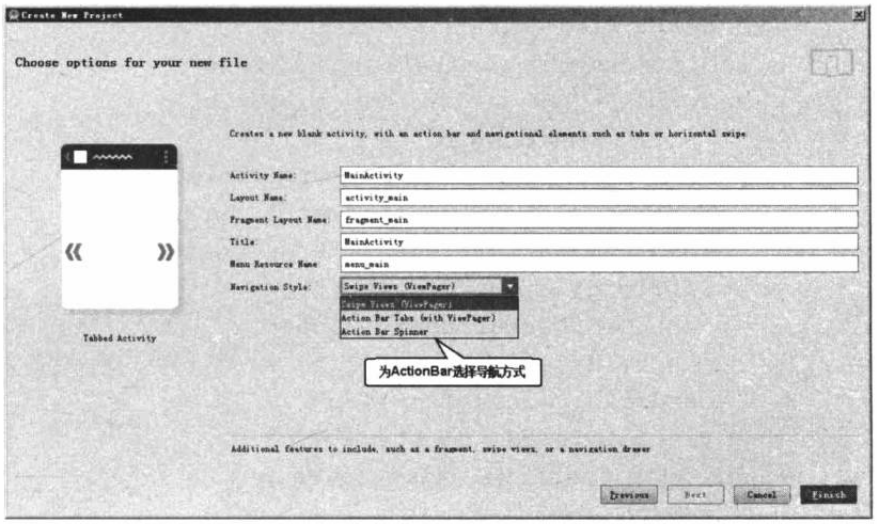


图 2.92 为 ActionBar 选择导航方式

从图 2.92 可以看出，Android Studio 为 Activity 提供了三种导航方式，其中 Action Bar Tabs 导航方式就是刚刚介绍的第一个实例。下面将会详细介绍另外两种导航方式。

实例：Android 3.0 以前的 Fragment 支持

Fragment 非常实用，Android 也为 3.0 版本以前的平台增加了 Fragment 支持，只是该 Fragment 不是继承 android.app.Fragment，而是继承 android.support.v4.app.Fragment。

除此之外，Android 还为该 android.support.v4.app.Fragment 提供了如下配套类。

- FragmentActivity：在早期版本上使用 Fragment 必须借助于 FragmentActivity 的支持，只有该支持类提供的 getSupportFragmentManager()方法才能获取 FragmentManager 管理器。
- ViewPager：它是 Fragment 容器，可以同时管理多个 Fragment，并允许多个 Fragment 切换时提供动画效果。
- FragmentPagerAdapter：Adapter 类，用于为 ViewPager 提供多个 Fragment。通常用于被扩展。

提示：

FragmentPagerAdapter 的作用有点类似于前面介绍的 Adapter，只是 Adapter 用于为 AdapterView 提供多个列表项；而 FragmentPagerAdapter 则专门为 ViewPager 提供多个 Fragment。

- PagerTitleStrip：与 ViewPager 结合使用，用于在 ViewPager 上显示“导航条”。

该实例的界面布局文件将会使用 ViewPager 容器，该容器可以盛装多个 Fragment，并为多个 Fragment 切换时提供动画支持。该实例的布局文件如下。

程序清单：codes\02\2.11\ActionBar_SwipeNav\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- 定义导航状态条组件 -->
    <android.support.v4.view.PagerTitleStrip
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#33b5e5"
        android:textColor="#fff"
        android:paddingTop="4dp"
        android:paddingBottom="4dp" />
</android.support.v4.view.ViewPager>
```

上面的布局文件中定义了一个 ViewPager 组件，并为该 ViewPager 组件定义了配套的 PagerTitleStrip 组件——它是一个导航状态条组件。

接下来 Activity 中还是需要按上面介绍的两个步骤来启用 ActionBar 的 Tab 导航支持。除此之外，为了让 ViewPager 组件能正常工作，Activity 需要为该 ViewPager 组件创建并设置 FragmentPagerAdapter。

下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionBar_SwipeNav\src\org\crazyit\ui\MainActivity.java

```
public class MainActivity extends FragmentActivity
    implements ActionBar.TabListener
{
    ViewPager viewPager;
    ActionBar actionBar;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 ActionBar 对象
        actionBar = getActionBar();
        // 获取 ViewPager
        viewPager = (ViewPager) findViewById(R.id.pager);
        // 创建一个 FragmentPagerAdapter 对象，该对象负责为 ViewPager 提供多个 Fragment
        FragmentPagerAdapter pagerAdapter = new FragmentPagerAdapter(
            getSupportFragmentManager())
        {
            // 获取第 position 位置的 Fragment
            @Override
            public Fragment getItem(int position)
            {
                Fragment fragment = new DummyFragment();
                Bundle args = new Bundle();
                args.putInt(DummyFragment.ARG_SECTION_NUMBER, position + 1);
                fragment.setArguments(args);
                return fragment;
            }
            // 该方法的返回值 i 表明该 Adapter 总共包括多少个 Fragment
            @Override
            public int getCount()
            {
                return 3;
            }
            // 该方法的返回值决定每个 Fragment 的标题
            @Override
            public CharSequence getPageTitle(int position)
            {
                switch (position)
                {
                    case 0:
                        return "第一页";
                    case 1:
                        return "第二页";
                    case 2:
                        return "第三页";
                }
                return null;
            }
        };
        // 设置 ActionBar 使用 Tab 导航方式
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        // 遍历 pagerAdapter 对象所包含的全部 Fragment
        // 每个 Fragment 对应创建一个 Tab 标签
        for (int i = 0; i < pagerAdapter.getCount(); i++)
```

```
        {
            actionBar.addTab(actionBar.newTab()
                .setText(pagerAdapter.getPageTitle(i))
                .setTabListener(this));
        }
        // 为ViewPager组件设置FragmentPagerAdapter
        viewPager.setAdapter(pagerAdapter); // ①
        // 为ViewPager组件绑定事件监听器
        viewPager.setOnPageChangeListener(
            new ViewPager.SimpleOnPageChangeListener()
            {
                // 当ViewPager显示的Fragment发生改变时激发该方法
                @Override
                public void onPageSelected(int position)
                {
                    actionBar.setSelectedNavigationItem(position);
                }
            });
    }
    @Override
    public void onTabUnselected(ActionBar.Tab tab,
        FragmentTransaction fragmentTransaction)
    {
    }
    // 当指定Tab标签被选中时激发该方法
    @Override
    public void onTabSelected(ActionBar.Tab tab,
         FragmentTransaction fragmentTransaction)
    {
        viewPager.setCurrentItem(tab.getPosition());  // ②
    }
    @Override
    public void onTabReselected(ActionBar.Tab tab,
        FragmentTransaction fragmentTransaction)
    {
    }
}
```

上面的程序采用匿名内部类的形式创建了一个FragmentPagerAdapter对象。接下来程序在①号代码处为ViewPager组件设置了该FragmentPagerAdapter对象，这样即可让该ViewPager正常工作。

注意：

上面的界面布局文件和MainActivity都使用了Android v4支持库，该支持库是位于extras\android\support\v4目录下的android-support-v4.jar文件，因此该实例需要将该JAR包复制到应用的app\libs目录下，并在Android Studio左上角的Project面板中选中该JAR，然后通过“Add as Library”右键菜单来添加该JAR包——这也是在Android Studio中为Androd应用添加第三方JAR包的方法。

为了启用ActionBar的Tab导航支持，上面的粗体字代码同样遵守前面介绍的两个步骤：先调用ActionBar的setNavigationMode(ActionBar.NAVIGATION_MODE_TABS)启用Tab导航支持；再调用ActionBar的addTab()方法添加Tab标签，并为Tab标签绑定事件监听器。

由于此处使用了 ViewPager 来管理多个 Fragment，程序代码处理 Fragment 的切换时更简单：只要调用 ViewPager 的 setCurrentItem()方法来显示指定 Fragment 即可，如以上程序中②号代码所示。

运行上面的程序，可以看到如图 2.93 所示的界面。

当用户单击图 2.93 所示界面上的 Tab 标签，或通过拖动来切换 ViewPager 显示的 Fragment 时，将可以看到 Fragment 切换时的动画效果。

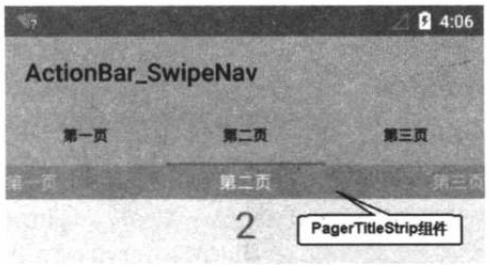


图 2.93 Android 3.0 版本以前的 Fragment 支持

2.11.6 使用 ActionBar 实现下拉式导航

ActionBar 除了可以提供 Tab 导航支持之外，还提供了下拉式（DropDown）导航方式。下拉式导航的 ActionBar 在顶端生成下拉列表框，当用户单击某个列表项时，系统根据用户单击事件导航到指定 Fragment。

为了使用 ActionBar 实现下拉式导航，按如下步骤进行即可。

① 调用 ActionBar 的 actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_ LIST)方法设置使用下拉列表的导航方式。

② 调用 ActionBar 的 setListNavigationCallbacks(SpinnerAdapter adapter, ActionBar.OnNavigationListener callback)方法添加多个列表项，并为每个列表项设置事件监听器。其中第一个参数 Adapter 负责提供多个列表项，第二个参数为事件监听器。

实例：ActionBar 结合 Fragment 实现下拉式导航

本实例的布局文件与前面介绍的 Tab 导航实例的布局文件完全相同：该布局文件只要有一个简单的容器即可，该容器用于盛装多个 Fragment。此处不再给出该界面布局文件代码。

下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionBar_DropDownNav\src\org\crazyit\ui\MainActivity java

```
public class MainActivity extends Activity implements
        ActionBar.OnNavigationListener
{
    private static final String SELECTED_ITEM = "selected_item";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ActionBar actionBar = getActionBar();
        // 设置 ActionBar 是否显示标题
        actionBar.setDisplayShowTitleEnabled(true);
        // 设置导航模式，使用 List 导航
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
        // 为 actionBar 安装 ArrayAdapter
        actionBar.setListNavigationCallbacks(
            new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1,
                android.R.id.text1, new String[]
```

```
                {"第一页","第二页","第三页" }), this);
    }
    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState)
    {
        if (savedInstanceState.containsKey(SELECTED_ITEM))
        {
            // 选中前面保存的索引对应的 Fragment 页
            getActionBar().setSelectedNavigationItem(
                savedInstanceState.getInt(SELECTED_ITEM));
        }
    }
    @Override
    public void onSaveInstanceState(Bundle outState)
    {
        // 将当前选中的 Fragment 页的索引保存到 Bundle 中
        outState.putInt(SELECTED_ITEM,
            getActionBar().getSelectedNavigationIndex());
    }
    // 当导航项被选中时激发该方法
    @Override
    public boolean onNavigationItemSelected(int position, long id)
    {
        // 创建一个新的 Fragment 对象
        Fragment fragment = new DummyFragment();
        // 创建一个 Bundle 对象，用于向 Fragment 传入参数
        Bundle args = new Bundle();
        args.putInt(DummyFragment.ARG_SECTION_NUMBER, position + 1);
        // 向 fragment 传入参数
        fragment.setArguments(args);
        // 获取 FragmentTransaction 对象
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        // 使用 fragment 代替该 Activity 中的 container 组件
        ft.replace(R.id.container, fragment);
        // 提交事务
        ft.commit();
        return true;
    }
}
```

上面程序中的第一段粗体字代码就是为 ActionBar 启用下拉导航支持的关键代码，这段代码做了上面介绍的两件事情：先调用 ActionBar 的 setNavigationMode(ActionBar.NAVIGATION_MODE_LIST)启用下拉列表导航支持；然后为 ActionBar 传入 ArrayAdapter（当然也可使用 SimpleAdapter 或扩展 BaseAdapter 对象）和监听器即可。

当用户选中指定的导航项时，将会激发该监听器的 onNavigationItemSelected()，该方法的处理逻辑与前面 Tab 导航实例中 onTabSelected()处理方法的处理逻辑完全相同，该实例所使用的 DummyFragment 与前面的 Fragment 类的代码也完全相同，此处不再赘述。

运行该实例，可以看到如图 2.94 所示的界面。

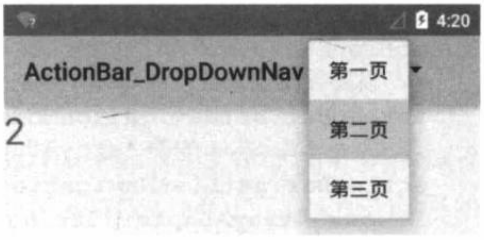


图 2.94 ActionBar 结合 Fragment 实现下拉式导航

2.12 本章小结

本章主要介绍了 Android 应用界面开发的相关知识，对于一个手机应用来说，它面临的最终用户都是不太懂软件的普通人，这批用户第一眼看到的就是软件界面，因此为 Android 系统提供一个友好的用户界面十分重要。学习本章需要重点掌握 View 与 ViewGroup 的功能和用法；Android 系统所有的基本 UI 组件、高级 UI 组件也都需要重点掌握。除此之外，用户界面少不了需要对话框与菜单，Android 为对话框提供了 AlertDialog 类，还提供了 PopupWindow、DatePickerDialog、TimePickerDialog 用于辅助开发对话框。另外，Toast 是手机系统特有的一种“准对话框”，它可用于显示简单的提示信息，而且一段时间后会自动隐藏，非常方便。Android 为菜单支持提供了 SubMenu、ContextMenu、MenuItem 等 API，读者必须掌握这些 API 的用法，并能通过它们为 Android 应用添加菜单支持。

CHAPTER

3

第3章 Android的事件处理

本章要点

- 事件处理概述与 Android 事件处理
- 基于监听的事件处理模型
- 事件与事件监听器接口
- 实现事件监听器的方式
- 基于回调的事件处理模型
- 基于回调的事件传播
- 常见的事件回调方法
- 响应系统设置事件
- 重写 onConfigurationChanged 方法响应系统设置更改
- Handler 类的功能与用法
- 使用 Handler 更新程序界面
- Handler、Looper、MessageQueue 工作原理
- 异步任务的功能与用法

与界面编程紧密相关的知识就是事件处理了，当用户在程序界面上执行各种操作时，应用程序必须为用户动作提供响应动作，这种响应动作就需要通过事件处理来完成。因此本章知识与上一章的内容衔接得非常紧密，实际上我们在介绍上一章示例时已经使用过 Android 的事件处理了。

Android 提供了两种方式的事件处理：基于回调的事件处理和基于监听的事件处理。熟悉传统图形界面编程的读者对于基于回调的事件处理可能比较熟悉；熟悉 AWT/Swing 开发方式的读者对于基于监听的事件处理可能比较熟悉。Android 系统充分利用了两种事件处理方式的优点，允许开发者采用自己熟悉的事件处理方式来为用户操作提供响应动作。

本章将会详细介绍 Android 事件处理的各种实现细节，学完本章内容之后，再结合上一章的内容，读者将可以开发出界面友好、人机交互良好的 Android 应用。

3.1 Android 事件处理概述

不管是桌面应用还是手机应用程序，面对最多的就是用户，经常需要处理的就是用户动作——也就是需要为用户动作提供响应，这种为用户动作提供响应的机制就是事件处理。

Android 提供了两套强大的事件处理机制：

- 基于监听的事件处理。
- 基于回调的事件处理。

对于 Android 基于监听的事件处理而言，主要做法就是为 Android 界面组件绑定特定的事件监听器，上一章我们已经见到大量这种事件处理的示例。

提示：

Android 还允许在界面布局文件中为 UI 组件的 android:onClick 属性指定事件监听方法，通过这种方式指定事件监听方法时，开发者需要在 Activity 中定义该事件监听方法（该方法必须有一个 View 类型的形参，该形参代表被单击的 UI 组件），当用户单击该 UI 组件时，系统将会激发 android:onClick 属性所指定的方法。

对于 Android 基于回调的事件处理而言，主要做法就是重写 Android 组件特定的回调方法，或者重写 Activity 的回调方法。Android 为绝大部分界面组件都提供了事件响应的回调方法，开发者只要重写它们即可。

一般来说，基于回调的事件处理可用于处理一些具有通用性的事件，基于回调的事件处理代码会显得比较简洁。但对于某些特定的事件，无法使用基于回调的事件处理，只能采用基于监听的事件处理。

3.2 基于监听的事件处理

基于监听的事件处理是一种更“面向对象”的事件处理，这种处理方式与 Java 的 AWT、Swing 的处理方式几乎完全相同。如果开发者有 AWT、Swing 事件处理的编程经验，基本上可以直接上手编程，甚至不需要学习。如果以前没有任何事件处理的编程经验，就需要花点时间先去理解事件监听的处理模型。

3.2.1 监听的处理模型

在事件监听的处理模型中，主要涉及如下三类对象。

- Event Source（**事件源**）：事件发生的场所，通常就是各个组件，例如按钮、窗口、菜单等。
- Event（**事件**）：事件封装了界面组件上发生的特定事情（通常就是一次用户操作）。如果程序需要获得界面组件上所发生事件的相关信息，一般通过 Event 对象来取得。
- Event Listener（**事件监听器**）：负责监听事件源所发生的事件，并对各种事件做出相应的响应。

提示：

有过 JavaScript、Visual Basic 等编程经验的读者都知道，事件响应的动作实际上就是一系列程序语句，通常以方法的形式组织起来。但 Java 是面向对象的编程语言，方法不能独立存在，所以必须以类的形式来组织这些方法，所以事件监听器的核心就是它所包含的方法——这些方法也被称为事件处理器（Event Handler）。

当用户按下一个按钮或者单击某个菜单项时，这些动作就会激发一个相应的事件，该事件就会触发事件源上注册的事件监听器（特殊的 Java 对象），事件监听器调用对应的事件处理器（事件监听器里的实例方法）来做出相应的响应。

Android 的事件处理机制是一种委派式（Delegation）事件处理方式：普通组件（事件源）将整个事件处理委托给特定的对象（事件监听器）；当该事件源发生指定的事件时，就通知所委托的事件监听器，由事件监听器来处理这个事件。

每个组件均可以针对特定的事件指定一个事件监听器，每个事件监听器也可监听一个或多个事件源。因为同一个事件源上可能发生多种事件，委派式事件处理方式可以把事件源上所有可能发生的事件分别授权给不同的事件监听器来处理；同时也可以让一类事件都使用同一个事件监听器来处理。

提示：

委派式事件处理方式明显“抄袭”了人类社会的分工协作，例如某个单位发生了火灾，该单位通常不会自己处理该事件，而是将该事件委派给消防局（事件监听器）处理；如果发生了打架斗殴事件，则委派给公安局（事件监听器）处理；而消防局、公安局也会同时监听多个单位的火灾、打架斗殴事件。这种委派式的处理方式将事件源和事件监听器分离，从而提供更好的程序模型，有利于提高程序的可维护性。

如图 3.1 所示为事件处理流程示意图。

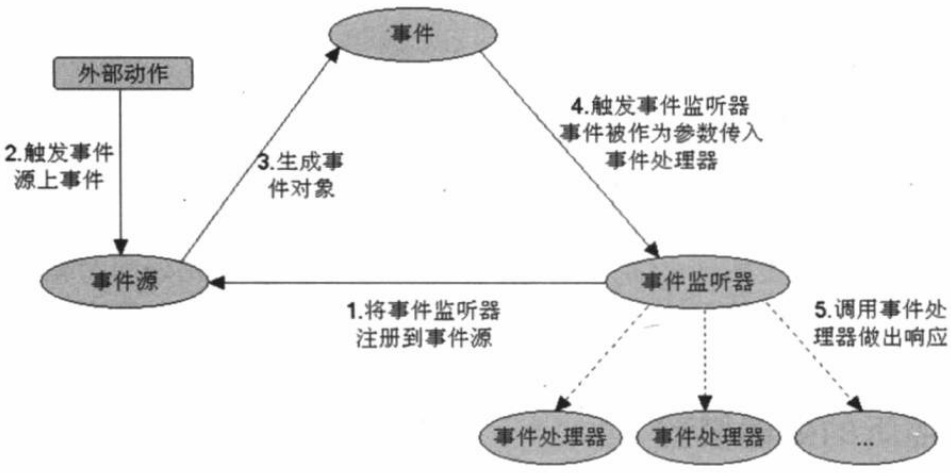


图 3.1 事件处理流程示意图

下面以一个简单的入门程序来示范基于监听的事件处理模型。

先看本程序的界面布局代码，该界面布局中只是定义了两个组件：一个文本框和一个按钮，界面布局代码如下。

程序清单：codes\03\3.2\EventQs\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<EditText
    android:id="@+id/txt"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:editable="false"
    android:cursorVisible="false"
    android:textSize="12pt"/>
<!-- 定义一个按钮，该按钮将作为事件源 -->
<Button
    android:id="@+id/bn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="单击我"/>
</LinearLayout>
```

上面程序中定义的按钮将会作为事件源，接下来程序将会为该按钮绑定一个事件监听器——监听器类必须由开发者来实现。下面是 Java 程序代码。

程序清单：codes\03\3.2\EventQs\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取应用程序中的 bn 按钮
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new MyClickListener()); // ①
    }
    // 定义一个单击事件的监听器
    class MyClickListener implements View.OnClickListener
    {
        // 实现监听器类必须实现的方法，该方法将会作为事件处理器
        @Override
        public void onClick(View v)
        {
            EditText txt = (EditText) findViewById(R.id.txt);
            txt.setText("bn 按钮被单击了！");
        }
    }
}
```

上面程序中的粗体字代码定义了一个 View.OnClickListener 实现类，这个实现类将会作为事件监听器使用。程序中①号代码用于为 bn 按钮注册事件监听器。当程序中的 bn 按钮被单击时，该处理器被触发，将看到程序中文本框内变为“bn 按钮被单击了”。

从上面的程序中可以看出，基于监听的事件处理模型的编程步骤如下。

1. 获取普通界面组件（事件源），也就是被监听的对象。
2. 实现事件监听器类，该监听器类是一个特殊的 Java 类，必须实现一个 XxxListener 接口。
3. 调用事件源的 setXxxListener 方法将事件监听器对象注册给普通组件（事件源）。

当事件源上发生指定事件时，Android 会触发事件监听器，由事件监听器调用相应的方法（事件处理器）来处理事件。

把上面的程序与图 3.1 结合起来看，可以发现基于监听的事件处理有如下规则。

- **事件源**：就是程序中的 bn 按钮，其实开发者不需要太多的额外处理，应用程序中任何组件都可作为事件源。
- **事件监听器**：就是程序中的 MyClickListener 类。监听器类必须由程序员负责实现，实现监听器类的关键就是实现处理器方法。
- **注册监听器**：只要调用事件源的 setXxxListener(XxxListener)方法即可。

对于上面三件事情，事件源可以是任何界面组件，不太需要开发者参与；注册监听器也只要一行代码即可，因此事件编程的关键就是实现事件监听器类。

3.2.2 事件和事件监听器

从图 3.1 中可以看出，当外部动作在 Android 组件上执行操作时，系统会自动生成事件对象，这个事件对象会作为参数传给事件源上注册的事件监听器。

基于监听的事件处理模型涉及三个成员：事件源、事件和事件监听器，其中事件源最容易创建，任意界面组件都可作为事件源；事件的产生无须程序员关心，它是由系统自动产生的；所以，实现事件监听器是整个事件处理的核心。

但在上面的程序中，我们并未发现事件的踪迹，这是什么原因呢？这是因为 Android 对事件监听模型做了进一步简化：如果事件源触发的事件足够简单，事件里封装的信息比较有限，那就无须封装事件对象，将事件对象传入事件监听器。

但对于键盘事件、触摸屏事件等，此时程序需要获取事件发生的详细信息。例如，键盘事件需要获取是哪个键触发的事件；触摸屏事件需要获取事件发生的位置等，对于这种包含更多信息的事件，Android 同样会将事件信息封装成 XxxEvent 对象，并把该对象作为参数传入事件处理器。

实例：控制飞机移动

下面以一个简单的飞机游戏为例来介绍键盘事件的监听。游戏中的飞机会随用户单击键的动作而移动：单击不同的键，飞机向不同的方向移动。

为了实现该程序，先开发一个自定义 View，该 View 负责绘制游戏的飞机。该 View 类的代码如下。

程序清单：codes\03\3.2\Plane\app\src\main\java\org\crazyit\event\PlainView.java

```
public class PlaneView extends View
{
    public float currentX;
    public float currentY;
    Bitmap plane;
    public PlaneView(Context context)
    {
        super(context);
        // 定义飞机图片
        plane = BitmapFactory.decodeResource(context.getResources(),
            R.drawable.plane);
```

```
        setFocusable(true);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 创建画笔
        Paint p = new Paint();
        // 绘制飞机
        canvas.drawBitmap(plane, currentX, currentY, p);
    }
}
```

上面的PlaneView足够简单，因为这个程序只需要绘制玩家自己控制的飞机，没有增加“敌机”。如果游戏中要增加“敌机”，那么还需要增加数据来控制敌机的坐标，并会在View上绘制敌机。

该游戏几乎不需要界面布局，该游戏直接使用PlaneView作为Activity显示的内容，并为该PlaneView增加键盘事件监听器即可。下面是该程序的Activity代码。

程序清单：codes\03\3.2\plane\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义飞机的移动速度
    private int speed = 10;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 去掉窗口标题
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // 全屏显示
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
        // 创建PlaneView组件
        final PlaneView planeView = new PlaneView(this);
        setContentView(planeView);
        planeView.setBackgroundResource(R.drawable.back);
        // 获取窗口管理器
        WindowManager windowManager = getWindowManager();
        Display display = windowManager.getDefaultDisplay();
        DisplayMetrics metrics = new DisplayMetrics();
        // 获得屏幕宽和高
        display.getMetrics(metrics);
        // 设置飞机的初始位置
        planeView.currentX = metrics.widthPixels / 2;
        planeView.currentY = metrics.heightPixels - 40;
        // 为planeView组件的键盘事件绑定监听器
        planeView.setOnKeyListener(new OnKeyListener()
        {
            @Override
            public boolean onKey(View source, int keyCode, KeyEvent event)
            {
                // 获取由哪个键触发的事件
                switch (event.getKeyCode())
                {
                    // 控制飞机下移
                    case KeyEvent.KEYCODE_S:
                        planeView.currentY += speed;
                        break;
                    // 控制飞机上移
```

```
                case KeyEvent.KEYCODE_W:
                    planeView.currentY -= speed;
                    break;
                // 控制飞机左移
                case KeyEvent.KEYCODE_A:
                    planeView.currentX -= speed;
                    break;
                // 控制飞机右移
                case KeyEvent.KEYCODE_D:
                    planeView.currentX += speed;
                    break;
            }
            // 通知planeView组件重绘
            planeView.invalidate();
            return true;
        }
    });
  }
}
```

上面程序中的粗体字代码就是控制飞机移动的关键代码——由于程序需要根据用户按下的键来确定飞机的移动方向，所以上面的程序先调用了 KeyEvent（事件对象）的 getKeyCode() 来获取触发事件的键，然后针对不同的键来改变游戏中飞机的坐标。

正如前面提到的，如果事件发生时有比较多的信息需要传给事件监听器，那么就需要将事件信息封装成 Event 对象，该 Event 对象将作为参数传入事件处理函数。

图 3.2　控制飞机的移动

运行上面的程序，将看到如图 3.2 所示的界面。

对于图 3.2 所示的“游戏”，当用户按下模拟器右边的 4 个方向键时，将可以看到“游戏”中的飞机可以上、下、左、右自由移动。

在基于监听的事件处理模型中，事件监听器必须实现事件监听器接口，Android 为不同的界面组件提供了不同的监听器接口，这些接口通常以内部类的形式存在。以 View 类为例，它包含了如下几个内部接口。

- View.OnClickListener：单击事件的事件监听器必须实现的接口。
- View.OnCreateContextMenuListener：创建上下文菜单事件的事件监听器必须实现的接口。
- View.onFocusChangeListener：焦点改变事件的事件监听器必须实现的接口。
- View.OnKeyListener：按键事件的事件监听器必须实现的接口。

提示：

上面这个“游戏”还只是一个“雏型”，为了增加这个游戏的可玩性，可以考虑为游戏随机地增加“敌机”，并让“敌机”在屏幕上移动。如果想增加对战效果，还可以考虑增加一个键盘监听器：监听特定按键，特定按键激发飞机射出子弹。

- View.OnLongClickListener：长按事件的事件监听器必须实现的接口。
- View.OnTouchListener：触摸事件的事件监听器必须实现的接口。

提示：

实际上可以把事件处理模型简化成如下理解。当事件源组件上发生事件时，系统将会执行该事件源组件上监听器的对应处理方法。与普通 Java 方法调用不同的是，普通 Java 程序里的方法是由程序主动调用的，事件处理中的事件处理器方法是由系统负责调用的。

通过上面的介绍不难看出，所谓事件监听器，其实就是实现了特定接口的 Java 类的实例。在程序中实现事件监听器，通常有如下几种形式。

- **内部类形式：**将事件监听器类定义成当前类的内部类。
- **外部类形式：**将事件监听器类定义成一个外部类。
- **Activity 本身作为事件监听器类：**让 Activity 本身实现监听器接口，并实现事件处理方法。
- **匿名内部类形式：**使用匿名内部类创建事件监听器对象。

3.2.3 内部类作为事件监听器类

前面两个程序中所使用的事件监听器类都是内部类形式，使用内部类可以在当前类中复用该监听器类；因为监听器类是外部类的内部类，所以可以自由访问外部类的所有界面组件。这也是内部类的两个优势。

使用内部类来定义事件监听器类的例子可以参看前面的例子程序，此处不再赘述。

3.2.4 外部类作为事件监听器类

使用外部类定义事件监听器类的形式比较少见，主要因为如下两个原因。

- 事件监听器通常属于特定的 GUI 界面，定义成外部类不利于提高程序的内聚性。
- 外部类形式的事件监听器不能自由访问创建 GUI 界面的类中的组件，编程不够简洁。

但如果某个事件监听器确实需要被多个 GUI 界面所共享，而且主要是完成某种业务逻辑的实现，则可以考虑使用外部类形式来定义事件监听器类。下面的程序定义了一个外部类作为 OnLongClickListener 类，该事件监听器实现了发送短信的功能。

程序清单：codes\03\3.2\SendSms\app\src\main\java\org\crazyit\event\SendSmsListener.java

```
public class SendSmsListener implements OnLongClickListener
{
    private Activity act;
    private EditText address;
    private EditText content;
    public SendSmsListener(Activity act, EditText address
        , EditText content)
    {
        this.act = act;
        this.address = address;
        this.content = content;
    }
    @Override
    public boolean onLongClick(View source)
    {
        String addressStr = address.getText().toString();
        String contentStr = content.getText().toString();
        // 获取短信管理器
        SmsManager smsManager = SmsManager.getDefault();
        // 创建发送短信的 PendingIntent
        PendingIntent sentIntent = PendingIntent.getBroadcast(act
```

```
            , 0, new Intent(), 0);
        // 发送文本短信
        smsManager.sendTextMessage(addressStr, null, contentStr
            , sentIntent, null);
        Toast.makeText(act, "短信发送完成", Toast.LENGTH_LONG).show();
        return false;
    }
}
```

上面的事件监听器类没有与任何 GUI 界面耦合，创建该监听器对象时需要传入两个 EditText 对象和一个 Activity 对象，其中一个 EditText 用于作为收信人号码，另一个 EditText 用于作为短信内容。

提示：

上面程序中的三行粗体字代码调用了 SmsManager、PendingIntent 来发送短信，关于 SmsManager、Intent 的用法可参看本书后面的内容。

该程序的界面布局比较简单，故不给出界面布局文件。该程序的 Java 代码如下。

程序清单：codes\03\3.2\SendSms\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText address;
    EditText content;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取页面中收件人地址、短信内容
        address = (EditText)findViewById(R.id.address);
        content = (EditText)findViewById(R.id.content);
        Button bn = (Button)findViewById(R.id.send);
        // 使用外部类的实例作为事件监听器
        bn.setOnLongClickListener(new SendSmsListener(
            this , address, content));
    }
}
```

上面程序中的粗体字代码用于为指定按钮的长单击事件绑定监听器，当用户长单击界面中的 bn 按钮时，程序将会触发 SendSmsListener 监听器，该监听器里包含的事件处理方法将会向指定手机发送短信。

运行上面的程序，将看到如图 3.3 所示的界面。

此处使用模拟器来发送短信——不要指望能给你的女朋友发送短信，否则中国移动要恼火了。该程序只能向另一台模拟器发送短信，比如执行如下命令来启动另一台模拟器（直接使用名为 fkjava 的 AVD 虚拟机）：

```
emulator @fkjava
```

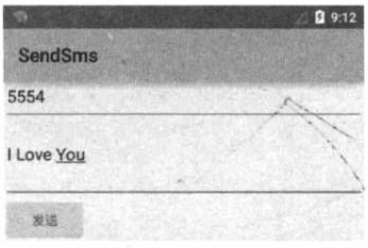


图 3.3　向指定手机发送短信

启动完成后可以在模拟器窗口的标题上看到该模拟器的号码，如图 3.4 所示。

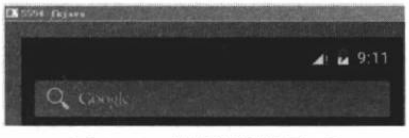


图 3.4　模拟器的号码

启动了如图 3.4 所示的模拟器之后，就可以通过图 3.3 所示的程序向指定模拟器发送短信了。

实际上不推荐将业务逻辑实现写在事件监听器中，包含

业务逻辑的事件监听器将导致程序的显示逻辑和业务逻辑耦合，从而增加程序后期的维护难度。如果确实有多个事件监听器需要实现相同的业务逻辑功能，则可以考虑使用业务逻辑组件来定义业务逻辑功能，再让事件监听器来调用业务逻辑组件的业务逻辑方法。

3.2.5 Activity本身作为事件监听器类

这种形式使用Activity本身作为监听器类，可以直接在Activity类中定义事件处理器方法。这种形式非常简洁，但这种做法有两个缺点。

- 这种形式可能造成程序结构混乱，Activity的主要职责应该是完成界面初始化工作，但此时还需包含事件处理器方法，从而引起混乱。
- 如果Activity界面类需要实现监听器接口，让人感觉比较怪异。

下面的程序使用Activity对象作为事件监听器。

程序清单：codes\03\3.2\ActivityListener\app\src\main\java\org\crazyit\event\MainActivity.java

```
// 实现事件监听器接口
public class MainActivity extends Activity
    implements OnClickListener
{
    EditText show;
    Button bn;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (EditText) findViewById(R.id.show);
        bn = (Button) findViewById(R.id.bn);
        // 直接使用Activity作为事件监听器
        bn.setOnClickListener(this);
    }
    // 实现事件处理方法
    @Override
    public void onClick(View v)
    {
        show.setText("bn按钮被单击了！");
    }
}
```

上面的程序让Activity类实现了OnClickListener事件监听器接口，从而可以在该Activity类中直接定义事件处理器方法：onClick(View v)（如上面的粗体字代码所示）。当为某个组件添加该事件监听器对象时，直接使用this作为事件监听器对象即可。

3.2.6 匿名内部类作为事件监听器类

大部分时候，事件处理器都没有什么复用价值（可复用代码通常都被抽象成了业务逻辑方法），因此大部分事件监听器只是临时使用一次，所以使用匿名内部类形式的事件监听器更合适。实际上，这种形式是目前使用最广泛的事件监听器形式。下面的程序使用匿名内部类来创建事件监听器。

程序清单：codes\03\3.2\AnonymousListener\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText show;
    Button bn;
    @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (EditText) findViewById(R.id.show);
        bn = (Button) findViewById(R.id.bn);
        // 使用匿名内部类的实例作为事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            // 实现事件处理方法
            @Override
            public void onClick(View v)
            {
                show.setText("bn 按钮被单击了！");
            }
        });
    }
}
```

上面程序中的粗体字部分使用匿名内部类创建了一个事件监听器对象，“new 监听器接口”或“new 事件适配器”的形式就是用于创建匿名内部类形式的事件监听器。

对于使用匿名内部类作为事件监听器类的形式来说，唯一的缺点就是匿名内部类的语法有点不易掌握，如果读者的 Java 基础扎实，匿名内部类的语法掌握较好，通常建议使用匿名内部类作为监听器类。

3.2.7 直接绑定到标签

Android 还有一种更简单的绑定事件监听器的方式，那就是直接在界面布局文件中为指定标签绑定事件处理方法。

对于很多 Android 界面组件标签而言，它们都支持 onClick 属性，该属性的属性值就是一个形如 xxx(View source)方法的方法名。例如如下界面布局文件。

程序清单：codes\03\3.2\BindingTag\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<EditText
    android:id="@+id/show"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:editable="false"
    android:cursorVisible="false"/>
<!-- 在标签中为按钮绑定事件处理方法 -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="单击我"
    android:onClick="clickHandler"/>
</LinearLayout>
```

上面程序中的粗体字代码用于在界面布局文件中为 Button 按钮绑定一个事件处理方法：clickHanlder，这就意味着开发者需要在该界面布局对应的 Activity 中定义一个 void clickHandler(View source)方法，该方法将会负责处理该按钮上的单击事件。下面是该界面布局对应的 Java 代码。

程序清单：codes\03\3.2\BindingTag\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    // 定义一个事件处理方法
    // 其中 source 参数代表事件源
    public void clickHandler(View source)
    {
        EditText show = (EditText) findViewById(R.id.show);
        show.setText("bn 按钮被单击了");
    }
}
```

上面程序中的粗体字代码定义了一个 clickHandler(View source)方法，当程序中的 bn 按钮被单击时，该方法将会被激发并处理 bn 按钮上的单击事件。

3.3 基于回调的事件处理

除了前面介绍的基于监听的事件处理模型之外，Android 还提供了一种基于回调的事件处理模型。从代码实现的角度来看，基于回调的事件处理模型更加简单。

3.3.1 回调机制与监听机制

如果说事件监听机制是一种委托式的事件处理，那么回调机制则恰好与之相反：对于基于回调的事件处理模型来说，事件源与事件监听器是统一的，或者说事件监听器完全消失了。当用户在 GUI 组件上激发某个事件时，组件自己特定的方法将会负责处理该事件。

为了使用回调机制类处理 GUI 组件上所发生的事件，我们需要为该组件提供对应的事件处理方法——而 Java 又是一种静态语言，我们无法为某个对象动态地添加方法，因此只能继承 GUI 组件类，并重写该类的事件处理方法来实现。

为了实现回调机制的事件处理，Android 为所有 GUI 组件都提供了一些事件处理的回调方法，以 View 为例，该类包含如下方法。

- boolean onKeyDown(int keyCode, KeyEvent event)：当用户在该组件上按下某个按键时触发该方法。
- boolean onKeyLongPress(int keyCode, KeyEvent event)：当用户在该组件上长按某个按键时触发该方法。
- boolean onKeyShortcut(int keyCode, KeyEvent event)：当一个键盘快捷键事件发生时触发该方法。
- boolean onKeyUp(int keyCode, KeyEvent event)：当用户在该组件上松开某个按键时触发该方法。
- boolean onTouchEvent(MotionEvent event)：当用户在该组件上触发触摸屏事件时触发该方法。
- boolean onTrackballEvent(MotionEvent event)：当用户在该组件上触发轨迹球事件时触发该方法。

下面的程序示范了基于回调的事件处理机制。正如前面所提到的，基于回调的事件处理机

制可通过自定义 View 来实现，自定义 View 时重写该 View 的事件处理方法即可。下面是一个自定义按钮的实现类。

程序清单：codes\03\3.3\CallbackHandler\app\src\main\java\org\crazyit\event\MyButton.java

```
public class MyButton extends Button
{
    public MyButton(Context context, AttributeSet set)
    {
        super(context, set);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        super.onKeyDown(keyCode, event);
        Log.v("-crazyit.org-", "the onKeyDown in MyButton");
        // 返回 true，表明该事件不会向外扩散
        return true;
    }
}
```

在上面自定义的 MyButton 类中，我们重写了 Button 类的 onKeyDown(int keyCode, KeyEvent event)方法，该方法将会负责处理按钮上的键盘事件。

接下来在界面布局文件中使用这个自定义 View，界面布局文件如下。

程序清单：codes\03\3.3\CallbackHandler\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 使用自定义 View 时应使用全限定类名 -->
<org.crazyit.event.MyButton
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="单击我"/>
</LinearLayout>
```

上面程序中的粗体字代码在 XML 界面布局文件中使用 MyButton 组件，接下来 Java 程序无须为该按钮绑定事件监听器——因为该按钮自己重写了 onKeyDown(int keyCode, KeyEvent event)方法，这意味着该按钮将会自己处理相应的事件。

运行上面的程序，先把焦点定位到该按钮上，如图 3.5 所示。

接着单击模拟器上的任意按键，将可以看到 Android Studio 的 LogCat 中有如图 3.6 所示的输出。

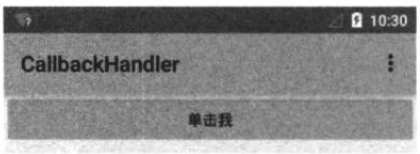


图 3.5 按钮获得焦点

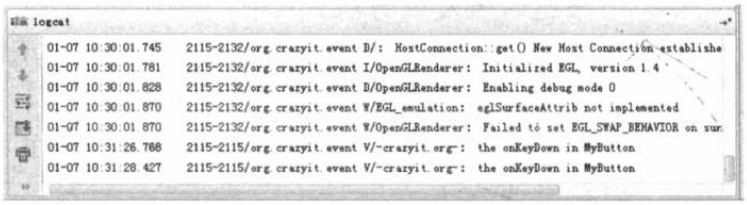


图 3.6 基于回调的事件处理

通过上面的介绍不难发现，对于基于监听的事件处理模型来说，事件源和事件监听器是分离的，当事件源上发生特定事件时，该事件交给事件监听器负责处理；对于基于回调的事件处理模型来说，事件源和事件监听器是统一的，当事件源发生特定事件时，该事件还是由事件源本身负责处理。

3.3.2 基于回调的事件传播

几乎所有基于回调的事件处理方法都有一个 boolean 类型的返回值，该返回值用于标识该处理方法是否能完全处理该事件。

- 如果处理事件的回调方法返回 true，表明该处理方法已完全处理该事件，该事件不会传播出去。
- 如果处理事件的回调方法返回 false，表明该处理方法并未完全处理该事件，该事件会传播出去。

对于基于回调的事件传播而言，某组件上所发生的事件不仅会激发该组件上的回调方法，也会触发该组件所在 Activity 的回调方法——只要事件能传播到该 Activity。

下面的程序示范了 Android 系统中的事件传播，该程序重写了 Button 类的 onKeyDown(int keyCode, KeyEvent event)方法，而且重写了该 Button 所在 Activity 的 onKeyDown(int keyCode, KeyEvent event)方法——程序没有阻止事件传播，因此程序可以看到事件从 Button 传播到 Activity 的情形。

下面是从 Button 派生而出的 MyButton 子类代码。

程序清单：codes\03\3.3\Propagation\app\src\main\java\org\crazyit\event\MyButton.java

```
public class MyButton extends Button
{
    public MyButton(Context context , AttributeSet set)
    {
        super(context , set);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        super.onKeyDown(keyCode , event);
        Log.v("-MyButton-" , "the onKeyDown in MyButton");
        // 返回 false，表明并未完全处理该事件，该事件依然向外扩散
        return false;
    }
}
```

上面的 MyButton 子类重写了 onKeyDown(int keyCode , KeyEvent event)方法，当用户在该按钮上按下某个键时将会触发该方法。但由于该方法返回了 false，这意味着该事件还会继续向外传播。

该程序也按前一个示例的方式使用自定义组件，并在 Activity 中重写 public boolean onKeyDown(int keyCode, KeyEvent event)方法，该方法也会在某个按键被按下时被回调。

看如下 Activity 类代码。

程序清单：codes\03\3.3\Propagation\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 绑定事件监听器
        bn.setOnKeyListener(new OnKeyListener()
        {
            @Override
```

```
        public boolean onKey(View source
                , int keyCode, KeyEvent event)
        {
            // 只处理按下键的事件
            if (event.getAction() == KeyEvent.ACTION_DOWN)
            {
                Log.v("-Listener-", "the onKeyDown in Listener");
            }
            // 返回 false，表明该事件会向外传播
            return false; // ①
        }
    });
}
// 重写 onKeyDown 方法，该方法可监听它所包含的所有组件的按键被按下事件
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    super.onKeyDown(keyCode , event);
    Log.v("-Activity-" , "the onKeyDown in Activity");
    // 返回 false，表明并未完全处理该事件，该事件依然向外扩散
    return false;
}
}
```

从上面的程序可以看出，粗体字代码重写了 Activity 的 onKeyDown(int keyCode, KeyEvent event)方法，当在该 Activity 包含的所有组件上按下某个键时，该方法都可能被触发——只要该组件没有完全处理该事件。

不仅如此，上面的程序还采用监听模式来处理该按钮上的按键被按下的事件。

运行上面的程序，先把焦点移动到程序界面的按钮上，然后按下模拟器右边的按键，将可以在 Android Studio 的 LogCat 中看到如图 3.7 所示的输出。

从图 3.7 不难看出，当该组件上发生某个按键被按下的事件时，Android 系统最先触发的应该是该按键上绑定的事件监听器，然后才触发该组件提供的事件回调方法，最后还会传播到该组件所在的 Activity——但如果我们让任何一个事件处理方法返回了 true，那么该事件将不会继续向外传播。例如，我们改写上面的 Activity 代码，将程序中①号代码改为 return true，然后再运行该程序，把焦点定位到该按钮上之后单击某个按键，在 Android Studio 的 LogCat 中将看到如图 3.8 所示的输出。

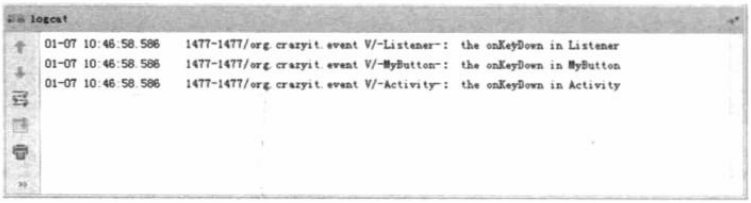


图 3.7　基于回调的事件传播

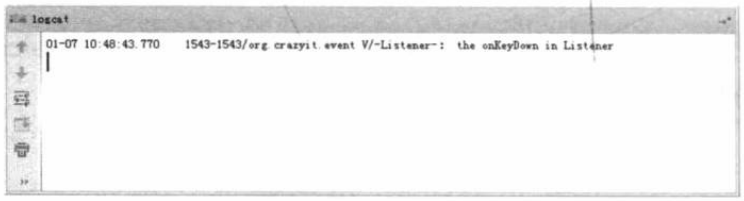


图 3.8　监听器阻止事件传播

3.3.3　重写 onTouchEvent 方法响应触摸屏事件

对比 Android 提供的两种事件处理模型，不难发现基于监听的事件处理模型具有更大的优势。

- 基于监听的事件处理模型分工更明确，事件源、事件监听器由两个类分开实现，因此具有更好的可维护性。
- Android 的事件处理机制保证基于监听的事件监听器会被优先触发。

实例：通过回调实现跟随手指的小球

在某些特定情况下，基于回调的事件处理机制会更好地提高程序的内聚性，例如上一章的“实例：跟随手指的小球”，如果改为基于回调的实现，则可以更好地提高程序的内聚性。

将该实例中的 DrawView 类改为如下形式。

程序清单：codes\03\3.3\CustomView\app\src\main\java\org\crazyit\event\DrawView.java

```
public class DrawView extends View
{
    public float currentX = 40;
    public float currentY = 50;
    // 定义、创建画笔
    Paint p = new Paint();
    public DrawView(Context context, AttributeSet set)
    {
        super(context, set);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 设置画笔的颜色
        p.setColor(Color.RED);
        // 绘制一个小圆（作为小球）
        canvas.drawCircle(currentX, currentY, 15, p);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 当前组件的 currentX、currentY 两个属性
        this.currentX = event.getX();
        this.currentY = event.getY();
        // 通知该组件重绘
        this.invalidate();
        // 返回 true 表明处理方法已经处理该事件
        return true;
    }
}
```

上面程序中的粗体字代码重写了 View 组件的 onTouchEvent(MotionEvent event)方法，这表示该组件自己就可以处理触摸屏事件，当用户手指在屏幕上移动时，该 View 上绘制的小球将会跟随用户手指。也就是说，这个自定义 View 本身就可以很好地处理屏幕事件。

接下来在程序中使用这个 DrawView 几乎不需要增加任何处理，直接将 View 放到界面布局中即可。如下面的 XML 界面布局文件所示。

程序清单：codes\03\3.3\CustomView\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 使用自定义组件 -->
<org.crazyit.event.DrawView
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
```

接下来在 Activity 类中不需要为该 View 绑定事件监听器——因为该 View 自己就可以处理

它的触摸屏事件。

通过为 View 提供事件处理的回调方法，可以很好地把事件处理方法封装在该 View 内部，从而提高程序的内聚性——基于回调的事件处理更适合于应付那种事件处理逻辑比较固定的 View，比如上面介绍的这个跟随用户手指的 View。

3.4 响应系统设置的事件

在开发 Android 应用时，有时候可能需要让应用程序随系统设置而进行调整，比如判断系统的屏幕方向、判断系统方向的方向导航设备等。除此之外，有时候可能还需要让应用程序监听系统设置的更改，对系统设置的更改做出响应。

3.4.1 Configuration 类简介

Configuration 类专门用于描述手机设备上的配置信息，这些配置信息既包括用户特定的配置项，也包括系统的动态设备配置。

程序可调用 Activity 的如下方法来获取系统的 Configuration 对象：

```
Configuration cfg = getResources().getConfiguration();
```

一旦获得了系统的 Configuration 对象，就可以使用该对象提供的如下常用属性来获取系统的配置信息。

- public float fontScale：获取当前用户设置的字体的缩放因子。
- public int keyboard：获取当前设备所关联的键盘类型。该属性可能返回 KEYBOARD_NOKEYS、KEYBOARD_QWERTY（普通电脑键盘）、KEYBOARD_12KEY（只有 12 个键的小键盘）等属性值。
- public int keyboardHidden：该属性返回一个 boolean 值用于标识当前键盘是否可用。该属性不仅会判断系统的硬件键盘，也会判断系统的软键盘（位于屏幕上）。如果该系统的硬件键盘不可用，但软键盘可用，该属性也会返回 KEYBOARDHIDDEN_NO；只有当两个键盘都不可用时才返回 KEYBOARDHIDDEN_YES。
- public Locale locale：获取用户当前的 Locale。
- public int mcc：获取移动信号的国家码。
- public int mnc：获取移动信号的网络码。
- public int navigation：判断系统上方向导航设备的类型。该属性可能返回 NAVIGATION_NONAV（无导航）、NAVIGATION_DPAD（DPAD 导航）、NAVIGATION_TRACKBALL（轨迹球导航）、NAVIGATION_WHEEL（滚轮导航）等属性值。
- public int orientation：获取系统屏幕的方向，该属性可能返回 ORIENTATION_LANDSCAPE（横向屏幕）、ORIENTATION_PORTRAIT（竖向屏幕）、ORIENTATION_SQUARE（方形屏幕）等属性值。
- public int touchscreen：获取系统触摸屏的触摸方式。该属性可能返回 TOUCHSCREEN_NOTOUCH（无触摸屏）、TOUCHSCREEN_STYLUS（触摸笔式的触摸屏）、TOUCHSCREEN_FINGER（接受手指的触摸屏）等属性值。

下面以一个实例来介绍 Configuration 的用法，该程序可以获取系统的屏幕方向、触摸屏方式等。

实例：获取系统设备状态

该程序的界面布局比较简单，程序只是提供了4个文本框来显示系统的屏幕方向、触摸屏方式等状态，故此处不再给出界面布局文件。

该程序的Java代码主要可分为两步。

1. 获取系统的Configuration对象。
2. 调用Configuration对象的属性来获取设备状态。

下面是该程序的Java代码。

程序清单：codes\03\3.4\ConfigurationTest\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText ori;
    EditText navigation;
    EditText touch;
    EditText mnc;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取应用界面中的界面组件
        ori = (EditText)findViewById(R.id.ori);
        navigation = (EditText)findViewById(R.id.navigation);
        touch = (EditText)findViewById(R.id.touch);
        mnc = (EditText)findViewById(R.id.mnc);
        Button bn = (Button)findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            // 为按钮绑定事件监听器
            @Override
            public void onClick(View source)
            {
                // 获取系统的Configuration对象
                Configuration cfg = getResources().getConfiguration();
                String screen = cfg.orientation ==
                    Configuration.ORIENTATION_LANDSCAPE
                    ? "横向屏幕": "竖向屏幕";
                String mncCode = cfg.mnc + "";
                String naviName = cfg.orientation ==
                    Configuration.NAVIGATION_NONAV
                    ? "没有方向控制" :
                    cfg.orientation == Configuration.NAVIGATION_WHEEL
                    ? "滚轮控制方向" :
                    cfg.orientation == Configuration.NAVIGATION_DPAD
                    ? "方向键控制方向" : "轨迹球控制方向";
                navigation.setText(naviName);
                String touchName = cfg.touchscreen ==
                    Configuration.TOUCHSCREEN_NOTOUCH
                    ? "无触摸屏" : "支持触摸屏";
                ori.setText(screen);
                mnc.setText(mncCode);
                touch.setText(touchName);
            }
        });
    }
}
```

上面程序中的粗体字代码用于获取系统的Configuration对象，一旦获得了系统的

Configuration 对象之后，程序就可以通过它来了解系统的设备状态了。运行上面的程序，将显示如图 3.9 所示的界面。

单击图 3.9 所示界面中的“获取手机信息”按钮，系统显示如图 3.10 所示的界面。

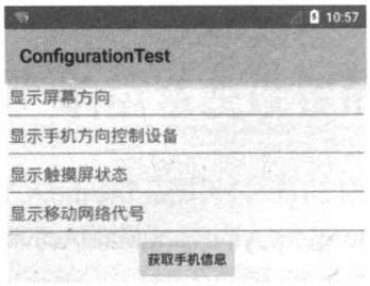


图 3.9 未获取设备状态

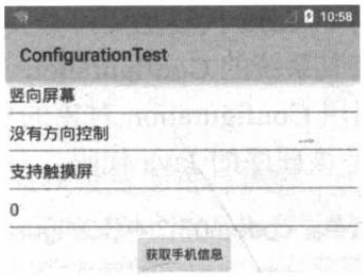


图 3.10 获取设备状态

3.4.2 重写 onConfigurationChanged 方法响应系统设置更改

如果程序需要监听系统设置的更改，则可以考虑重写 Activity 的 onConfigurationChanged (Configuration newConfig)方法，该方法是一个基于回调的事件处理方法：当系统设置发生更改时，该方法会被自动触发。

为了在程序中动态地更改系统设置，我们可调用 Activity 的 setRequestedOrientation(int)方法来修改屏幕的方向。

实例：监听屏幕方向的改变

该实例的界面布局很简单，该界面中仅包含一个普通按钮，该按钮用于动态修改系统屏幕的方向，此处不再给出系统界面布局代码。

该程序的 Java 代码主要会调用 Activity 的 setRequestedOrientation(int)方法来动态更改屏幕方向。除此之外，我们还重写了 Activity 的 onConfigurationChanged(Configuration newConfig)方法，该方法可用于监听系统设置的更改。程序代码如下。

程序清单：codes\03\3.4\ChangeCfg\app\src\main\java\org\crazyit\event\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Configuration config = getResources().getConfiguration();
                // 如果当前是横屏
                if (config.orientation == Configuration.ORIENTATION_LANDSCAPE)
                {
                    // 设为竖屏
                    MainActivity.this.setRequestedOrientation(
                        ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                }
                // 如果当前是竖屏
                if (config.orientation == Configuration.ORIENTATION_PORTRAIT)
```

```
                {
                    // 设为横屏
                    MainActivity.this.setRequestedOrientation(
                        ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                }
            }
        });
    }
    // 重写该方法，用于监听系统设置的更改，主要是监控屏幕方向的更改
    @Override
    public void onConfigurationChanged(Configuration newConfig)
    {
        super.onConfigurationChanged(newConfig);
        String screen = newConfig.orientation ==
            Configuration.ORIENTATION_LANDSCAPE ? "横向屏幕" : "竖向屏幕";
        Toast.makeText(this, "系统的屏幕方向发生改变" + "\n修改后的屏幕方向为：
            + screen, Toast.LENGTH_LONG).show();
    }
}
```

上面程序中的前两行粗体字代码用于动态地修改手机屏幕的方向，接下来的粗体字代码重写了 Activity 的 onConfigurationChanged(Configuration newConfig)方法，当系统设置发生更改时，该方法将会被自动回调。

除此之外，为了让该 Activity 能监听屏幕方向更改的事件，需要在配置该 Activity 时指定 android:configChanges 属性，该属性可以支持 mcc、mnc、locale、touchscreen、keyboard、keyboardHidden、navigation、orientation、screenLayout、uiMode、screenSize、smallestScreenSize、fontScale 属性值，其中 orientation|screenSize 属性值指定该 Activity 可以监听屏幕方向改变的事件。

因此，将应用的 AndroidManifest.xml 文件改为如下形式。

程序清单：codes\03\3.4\ChangeCfg\app\main\src\AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.crazyit.event" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <!-- 设置 Activity 可以监听屏幕方向改变的事件 -->
        <activity
            android:configChanges="orientation|screenSize"
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

上面的粗体字配置代码指定了该 Activity 可以监听屏幕方向改变的事件，这样当程序改变手机屏幕方向时，Activity 的 onConfigurationChanged()方法就会被回调。

提供上面的程序和设置之后，运行该程序，单击应用程序中的“更改屏幕方向”按钮，将可以看到如图 3.11 所示的界面。

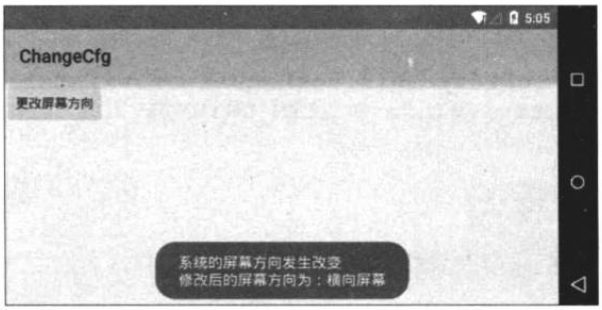


图 3.11 设置横向屏幕并响应系统设置的更改

3.5 Handler 消息传递机制

出于性能优化考虑，Android 的 UI 操作并不是线程安全的，这意味着如果有多个线程并发操作 UI 组件，则可能导致线程安全问题。为了解决这个问题，Android 制定了一条简单的规则：只允许 UI 线程修改 Activity 里的 UI 组件。

当一个程序第一次启动时，Android 会同时启动一条主线程（Main Thread），主线程主要负责处理与 UI 相关的事件，如用户的按键事件、用户接触屏幕的事件及屏幕绘图事件，并把相关的事件分发到对应的组件进行处理。所以，主线程通常又被叫作 UI 线程。

Android 的消息传递机制是另一种形式的“事件处理”，这种机制主要是为了解决 Android 应用的多线程问题——Android 平台只允许 UI 线程修改 Activity 里的 UI 组件，这样就会导致新启动的线程无法动态改变界面组件的属性值。但在实际 Android 应用开发中，尤其是涉及动画的游戏开发中，需要让新启动的线程周期性地改变界面组件的属性值，这就需要借助于 Handler 的消息传递机制来实现了。

3.5.1 Handler 类简介

Handler 类的主要作用有两个。

- 在新启动的线程中发送消息。
- 在主线程中获取、处理消息。

上面的说法看上去很简单，似乎只要分成两步即可：在新启动的线程中发送消息；然后在主线程中获取并处理消息。但这个过程涉及两个问题：新启动的线程何时发送消息呢？主线程何时去获取并处理消息呢？这个时机显然不好控制。

为了让主线程能“适时”地处理新启动的线程所发送的消息，显然只能通过回调的方式来实现——开发者只要重写 Handler 类中处理消息的方法，当新启动的线程发送消息时，消息会发送到与之关联的 MessageQueue，而 Handler 会不断地从 MessageQueue 中获取并处理消息——这将导致 Handler 类中处理消息的方法被回调。

Handler 类包含如下方法用于发送、处理消息。

- void handleMessage(Message msg)：处理消息的方法。该方法通常用于被重写。
- final boolean hasMessages(int what)：检查消息队列中是否包含 what 属性为指定值的消息。
- final boolean hasMessages(int what, Object object)：检查消息队列中是否包含 what 属性为指定值且 object 属性为指定对象的消息。
- 多个重载的 Message obtainMessage()：获取消息。
- sendEmptyMessage(int what)：发送空消息。
- final boolean sendEmptyMessageDelayed(int what, long delayMillis)：指定多少毫秒之后发送空消息。

- final boolean sendMessage(Message msg)：立即发送消息。
- final boolean sendMessageDelayed(Message msg, long delayMillis)：指定多少毫秒之后发送消息。

借助于上面这些方法，程序可以方便地利用 Handler 来进行消息传递。

实例：自动播放动画

本实例通过一个新线程来周期性地修改 ImageView 所显示的图片，通过这种方式来开发一个动画效果。该程序的界面布局代码非常简单，程序只是在界面布局中定义了 ImageView 组件，此处不再给出界面布局代码。

接下来主程序使用 java.util.Timer 来周期性地执行指定任务，程序代码如下。

程序清单：codes\03\3.5\HandlerTest\app\src\main\java\org\crazyit\handler\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义周期性显示的图片 ID
    int[] imageIds = new int[]
    {
        R.drawable.java,
        R.drawable.javaee,
        R.drawable.ajax,
        R.drawable.android,
        R.drawable.swift
    };
    int currentImageId = 0;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ImageView show = (ImageView) findViewById(R.id.show);
        final Handler myHandler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 如果该消息是本程序所发送的
                if (msg.what == 0x1233)
                {
                    // 动态地修改所显示的图片
                    show.setImageResource(imageIds[currentImageId++
                        % imageIds.length]);
                }
            }
        };
        // 定义一个计时器，让该计时器周期性地执行指定任务
        new Timer().schedule(new TimerTask()
        {
            @Override
            public void run()
            {
                // 发送空消息
                myHandler.sendEmptyMessage(0x1233);
            }
        }, 0, 1200);
    }
}
```

上面程序中的第二段粗体字代码通过 Timer 周期性地执行指定任务，Timer 对象可调度

TimerTask 对象，TimerTask 对象的本质就是启动一条新线程，由于 Android 不允许在新线程中访问 Activity 里的界面组件，因此程序只能在新线程里发送一条消息，通知系统更新 ImageView 组件。

上面程序中的第一段粗体字代码重写了 Handler 的 handleMessage(Message msg)方法，该方法用于处理消息——当新线程发送消息时，该方法会被自动回调，handleMessage(Message msg)方法依然位于主线程中，所以可以动态地修改 ImageView 组件的属性。这就实现了本程序所要达到的效果：由新线程来周期性地修改 ImageView 的属性，从而实现动画效果。运行上面的程序，可以看到应用程序中 5 张图片交替显示的动态效果。

3.5.2 Handler、Loop、MessageQueue 的工作原理

为了更好地理解 Handler 的工作原理，下面先介绍一下与 Handler 一起工作的几个组件。

- Message：Handler 接收和处理的消息对象。
- Looper：每个线程只能拥有一个 Looper。它的 loop 方法负责读取 MessageQueue 中的消息，读到信息之后就把消息交给发送该消息的 Handler 进行处理。
- MessageQueue：消息队列，它采用先进先出的方式来管理 Message。程序创建 Looper 对象时，会在它的构造器中创建 MessageQueue 对象。Looper 的构造器源代码如下：

```
private Looper()
{
    mQueue = new MessageQueue();
    mRun = true;
    mThread = Thread.currentThread();
}
```

该构造器使用了 private 修饰，表明程序员无法通过构造器创建 Looper 对象。从上面的代码不难看出，程序在初始化 Looper 时会创建一个与之关联的 MessageQueue，这个 MessageQueue 就负责管理消息。

- Handler：它的作用有两个——发送消息和处理消息，程序使用 Handler 发送消息，由 Handler 发送的消息必须被送到指定的 MessageQueue。也就是说，如果希望 Handler 正常工作，必须在当前线程中有一个 MessageQueue；否则消息就没有 MessageQueue 进行保存了。不过 MessageQueue 是由 Looper 负责管理的，也就是说，如果希望 Handler 正常工作，必须在当前线程中有一个 Looper 对象。为了保证当前线程中有 Looper 对象，可以分如下两种情况处理。
- 在主 UI 线程中，系统已经初始化了一个 Looper 对象，因此程序直接创建 Handler 即可，然后就可通过 Handler 来发送消息、处理消息了。
- 程序员自己启动的子线程，必须自己创建一个 Looper 对象，并启动它。创建 Looper 对象调用它的 prepare()方法即可。

prepare()方法保证每个线程最多只有一个 Looper 对象。prepare()方法的源代码如下：

```
public static final void prepare()
{
    if (sThreadLocal.get() != null)
    {
        throw new RuntimeException("Only one Looper may be created per thread");
    }
    sThreadLocal.set(new Looper());
}
```

接下来调用 Looper 的静态 loop()方法来启动它。loop()方法使用一个死循环不断取出

MessageQueue中的消息，并将取出的消息分给该消息对应的Handler进行处理。下面是Looper类的loop()方法的源代码。

```
for (;;)
{
    Message msg = queue.next(); // 获取消息队列中的下一个消息，如果没有消息，将会阻塞
    if (msg == null) {
        // 如果消息为null，表明消息队列正在退出
        return;
    }
    Printer logging = me.mLogging;
    if (logging != null) {
        logging.println(">>>>> Dispatching to " + msg.target + " " +
            msg.callback + ": " + msg.what);
    }
    msg.target.dispatchMessage(msg);
    if (logging != null) {
        logging.println("<<<<< Finished to " + msg.target + " " + msg.callback);
    }
    // 使用final修饰该标识符，保证在分发消息的过程中线程标识符不会被修改
    final long newIdent = Binder.clearCallingIdentity();
    if (ident != newIdent) {
    Log.wtf(TAG, "Thread identity changed from 0x"
        + Long.toHexString(ident) + " to 0x"
        + Long.toHexString(newIdent) + " while dispatching to "
        + msg.target.getClass().getName() + " "
        + msg.callback + " what=" + msg.what);
    }
    msg.recycle();
}
```

归纳起来，Looper、MessageQueue、Handler各自的作用如下。

- Looper：每个线程只有一个Looper，它负责管理MessageQueue，会不断地从MessageQueue中取出消息，并将消息分给对应的Handler处理。
- MessageQueue：由Looper负责管理。它采用先进先出的方式来管理Message。
- Handler：它能把消息发送给Looper管理的MessageQueue，并负责处理Looper分给它的消息。

在线程中使用Handler的步骤如下。

1. 调用Looper的prepare()方法为当前线程创建Looper对象，创建Looper对象时，它的构造器会创建与之配套的MessageQueue。
2. 有了Looper之后，创建Handler子类的实例，重写handleMessage()方法，该方法负责处理来自于其他线程的消息。
3. 调用Looper的loop()方法启动Looper。

下面通过一个实例来介绍Looper与Handler的用法。

实例：使用新线程计算质数

该实例允许用户输入一个数值上限，当用户单击“计算”按钮时，该应用会将该上限数值发送到新启动的线程中，让该线程来计算该范围内的所有质数。

之所以不直接在UI线程中计算该范围内的所有质数，是因为UI线程需要响应用户动作，如果在UI线程中执行一个“耗时”操作，将会导致UI线程被阻塞，从而让应用程序失去响应。比如在该实例中，如果用户输入的数值太大，系统可能需要较长时间才能计算出所有质数，这就可能导致UI线程失去响应。

提示：

尽量避免在UI线程中执行耗时操作，因为这样可能导致一个“著名”的异常：ANR异常。只要在UI线程中执行需要消耗大量时间的操作，都会引发ANR，因为这会导致Android应用程序无法响应输入事件和Broadcast。

为了将用户在UI界面输入的数值上限动态地传给新启动的线程，本实例将会在线程中创建一个Handler对象，然后UI线程的事件处理方法就可以通过该Handler向新线程发送消息了。

该实例的界面布局文件比较简单，只有一个文本框和一个按钮，故此处不再给出界面布局文件。

该实例的Activity代码如下。

程序清单：codes\03\3.5\CalPrime\app\src\main\java\org\crazyit\handler\MainActivity.java

```
public class MainActivity extends Activity
{
    static final String UPPER_NUM = "upper";
    EditText etNum;
    CalThread calThread;
    // 定义一个线程类
    class CalThread extends Thread
    {
        public Handler mHandler;
        public void run()
        {
            Looper.prepare();
            mHandler = new Handler()
            {
                // 定义处理消息的方法
                @Override
                public void handleMessage(Message msg)
                {
                    if(msg.what == 0x123)
                    {
                        int upper = msg.getData().getInt(UPPER_NUM);
                        List<Integer> nums = new ArrayList<Integer>();
                        // 计算从2开始、到upper的所有质数
                        outer:
                        for (int i = 2 ; i <= upper ; i++)
                        {
                            // 用i除以从2开始、到i的平方根的所有数
                            for (int j = 2 ; j <= Math.sqrt(i) ; j++)
                            {
                                // 如果可以整除，则表明这个数不是质数
                                if(i != 2 && i % j == 0)
                                {
                                    continue outer;
                                }
                            }
                            nums.add(i);
                        }
                        // 使用Toast显示统计出来的所有质数
                        Toast.makeText(MainActivity.this , nums.toString()
                            , Toast.LENGTH_LONG).show();
                    }
                }
            };
            Looper.loop();
        }
    }
    @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        etNum = (EditText)findViewById(R.id.etNum);
        calThread = new CalThread();
        // 启动新线程
        calThread.start();
    }
    // 为按钮的点击事件提供事件处理方法
    public void cal(View source)
    {
        // 创建消息
        Message msg = new Message();
        msg.what = 0x123;
        Bundle bundle = new Bundle();
        bundle.putInt(UPPER_NUM ,
            Integer.parseInt(etNum.getText().toString()));
        msg.setData(bundle);
        // 向新线程中的Handler发送消息
        calThread.mHandler.sendMessage(msg);
    }
}
```

上面的粗体字代码是实例的关键代码，这些粗体字代码在新线程内创建了一个 Handler。由于在新线程中创建 Handler 时必须先创建 Looper，因此程序先调用 Looper 的 prepare()方法为当前线程创建了一个 Looper 实例，并创建了配套的 MessageQueue。新线程有了 Looper 对象之后，接下来程序创建了一个 Handler 对象，该 Handler 可以处理其他线程发送过来的消息。程序最后还调用了 Looper 的 loop()方法。

运行该程序，无论用户输入多大的数值，计算该范围内的质数都将会交给新线程完成，而前台 UI 线程不会受到影响。该程序的运行效果如图 3.12 所示。

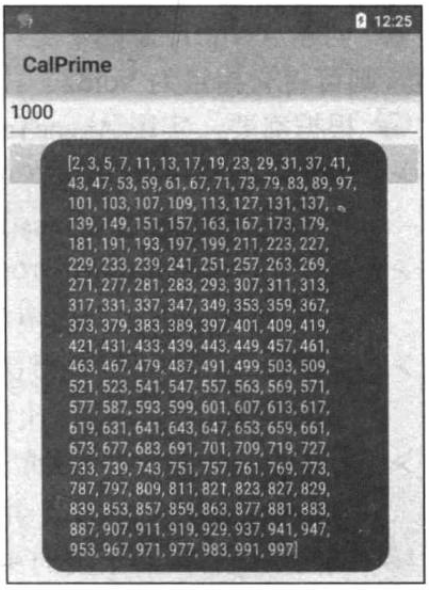


图 3.12 使用新线程计算质数

3.6 异步任务（AsyncTask）

前面已经介绍过，Android 的 UI 线程主要负责处理用户的按键事件、用户触屏事件及屏幕绘图事件等，因此开发者的其他操作不应该、也不能阻塞 UI 线程；否则 UI 界面将会变得停止响应——用户感觉非常糟糕。

提示：

Android 默认约定当 UI 线程阻塞超过 20 秒时将会引发 ANR（Application Not Responding）异常。但实际上，不要说 20 秒，即使是 5 秒甚至 2 秒，用户都会感觉十分不爽（用户是很急燥的！）。因此笔者认为，其实没必要去记这个 20 秒的时间限度。总之，开发者需要牢记：不要在 UI 线程中执行一些耗时的操作。

为了避免 UI 线程失去响应的问题，Android 建议将耗时操作放在新线程中完成，但新线程也可能需要动态更新 UI 组件，比如需要从网上获取一个网页，然后在 TextView 中将其源代码显示出来，此时就应该将连接网络、获取网络数据的操作放在新线程中完成。问题是：获取

网络数据之后，新线程不允许直接更新 UI 组件。

为了解决新线程不能更新 UI 组件的问题，Android 提供了如下几种解决方案。

- **使用 Hanlder 实现线程之间的通信。**
- Activity.runOnUiThread(Runnable)。
- View.post(Runnable)。
- View.postDelayed(Runnable, long)。

上一节已经见到了使用 Handler 的实例，后面的三种方式可能导致编程略显烦琐，而异步任务（AsyncTask）则可进一步简化这种操作。相对来说 AsyncTask 更轻量级一些，适用于简单的异步处理，不需要借助线程和 Handler 即可实现。

AsyncTask<Params, Progress, Result>是一个抽象类，通常用于被继承，继承 AsyncTask 时需要指定如下三个泛型参数。

- Params：启动任务执行的输入参数的类型。
- Progress：后台任务完成的进度值的类型。
- Result：后台执行任务完成后返回结果的类型。

使用 AsyncTask 只要如下三步即可。

1. 创建 AsyncTask 的子类，并为三个泛型参数指定类型。如果某个泛型参数不需要指定类型，则可将它指定为 Void。
2. 根据需要，实现 AsyncTask 的如下方法。
   - doInBackground(Params...)：重写该方法就是后台线程将要完成的任务。该方法可以调用 publishProgress(Progress... values)方法更新任务的执行进度。
   - onProgressUpdate(Progress... values)：在 doInBackground()方法中调用 publishProgress()方法更新任务的执行进度后，将会触发该方法。
   - onPreExecute()：该方法将在执行后台耗时操作前被调用。通常该方法用于完成一些初始化的准备工作，比如在界面上显示进度条等。
   - onPostExecute(Result result)：当 doInBackground()完成后，系统会自动调用 onPostExecute()方法，并将 doInBackground()方法的返回值传给该方法。
3. 调用 AsyncTask 子类的实例的 execute(Params... params)开始执行耗时任务。

使用 AsyncTask 时必须遵守如下规则。

- 必须在 UI 线程中创建 AsyncTask 的实例。
- 必须在 UI 线程中调用 AsyncTask 的 execute()方法。
- AsyncTask 的 onPreExecute()、onPostExecute(Result result)、doInBackground (Params... params)、onProgressUpdate(Progress... values)方法，不应该由程序员代码调用，而是由 Android 系统负责调用。
- 每个 AsyncTask 只能被执行一次，多次调用将会引发异常。

实例：使用异步任务执行下载

本实例示范如何使用异步任务下载网络资源。该实例的界面布局很简单，只包含两个组件：一个文本框用于显示从网络下载的页面代码；一个按钮用于激发下载任务。此处不再给出界面布局文件。

该程序的 Activity 代码如下。

程序清单：coder\03\3.6\AsyncTaskTest\app\src\main\java\org\crazyit\handler\MainActivity.java

```
public class MainActivity extends Activity
{
    private TextView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (TextView) findViewById(R.id.show);
    }
    // 重写该方法，为界面上的按钮提供事件响应方法
    public void download(View source) throws MalformedURLException
    {
        DownTask task = new DownTask(this);
        task.execute(new URL("http://www.crazyit.org/ethos.php"));
    }
    class DownTask extends AsyncTask<URL, Integer, String>
    {
        // 可变长的输入参数，与AsyncTask.exucute()对应
        ProgressDialog pdialog;
        // 定义记录已经读取行的数量
        int hasRead = 0;
        Context mContext;
        public DownTask(Context ctx)
        {
            mContext = ctx;
        }
        @Override
        protected String doInBackground(URL... params)
        {
            StringBuilder sb = new StringBuilder();
            try
            {
                URLConnection conn = params[0].openConnection();
                // 打开conn连接对应的输入流，并将它包装成BufferedReader
                BufferedReader br = new BufferedReader(
                    new InputStreamReader(conn.getInputStream()
                    , "utf-8"));
                String line = null;
                while ((line = br.readLine()) != null)
                {
                    sb.append(line + "\n");
                    hasRead++;
                    publishProgress(hasRead);
                }
                return sb.toString();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            return null;
        }
        @Override
        protected void onPostExecute(String result)
        {
            // 返回HTML页面的内容
            show.setText(result);
            pdialog.dismiss();
        }
        @Override
        protected void onPreExecute()
```

```
        {
            pdialog = new ProgressDialog(mContext);
            // 设置对话框的标题
            pdialog.setTitle("任务正在执行中");
            // 设置对话框显示的内容
            pdialog.setMessage("任务正在执行中，敬请等待...");
            // 设置对话框不能用“取消”按钮关闭
            pdialog.setCancelable(false);
            // 设置该进度条的最大进度值
            pdialog.setMax(202);
            // 设置对话框的进度条风格
            pdialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            // 设置对话框的进度条是否显示进度
            pdialog.setIndeterminate(false);
            pdialog.show();
        }
        @Override
        protected void onProgressUpdate(Integer... values)
        {
            // 更新进度
            show.setText("已经读取了【" + values[0] + "】行！");
            pdialog.setProgress(values[0]);
        }
    }
}
```

上面程序的download()方法很简单，它只是创建了DownTask（AsyncTask的子类）实例，并调用它的execute()方法开始执行异步任务。

该程序的重点是实现AsyncTask的子类，实现该子类时实现了如下4个方法。

- doInBackground()：该方法的代码完成实际的下载任务。
- onPreExecute()：该方法的代码负责在下载开始的时候显示一个进度条。
- onProgressUpdate()：该方法的代码负责随着下载进度的改变更新进度条的进度值。
- onPostExecute()：该方法的代码负责当下载完成后，将下载的代码显示出来。

提示：

该程序使用了网络编程从网络下载数据。关于Android网络编程的知识，请参考本书第13章的内容。除此之外，本程序需要访问网络，因此还需要在AndroidManifest.xml文件中声明如下权限：<uses-permission android:name="android.permission.INTERNET"/>。

运行该程序并单击“下载”按钮，将可以看到如图3.13所示的界面。

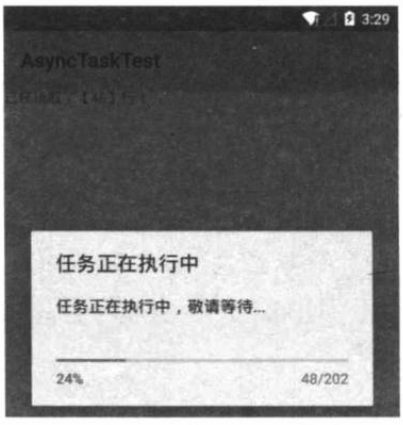


图3.13　使用异步任务下载网络资源

3.7 本章小结

本章是对上一章内容的补充：图形界面编程肯定需要与事件处理结合，当我们开发了一个界面友好的应用之后，用户在程序界面上执行操作时，程序必须为这种用户操作提供响应动作，这种响应动作就是由事件处理来完成的。学习本章的重点是掌握 Android 的两种事件处理机制：基于回调的事件处理和基于监听的事件处理；对于基于监听的事件处理来说，开发者需要掌握事件监听的处理模式，以及不同事件对应的监听器接口；对于基于回调的事件处理来说，开发者需要掌握不同事件对应的回调方法。除此之外，本章还介绍了重写 onConfigurationChanged 方法响应系统设置更改。需要指出的是，由于 Android 不允许在子线程中更新界面组件，如果想在子线程中更新界面组件，开发者需要借助于 Handler 对象来实现。本章详细介绍了 Handler、Looper 与 MessageQueue 之间的关系及工作原理。

CHAPTER

4

第4章 深入理解 Activity 与 Fragment

本章要点

- 理解 Activity 的功能与作用
- 开发普通 Activity 类
- 在 AndroidManifest.xml 中配置 Activity
- 特殊 Activity 的功能和用法
- 在程序中启动 Activity
- 关闭 Activity
- 使用 Bundle 在不同 Activity 之间交换数据
- 启动其他 Activity 并返回结果
- Activity 的回调机制
- Activity 的生命周期
- Fragment 概述及 Fragment 的设计哲学
- 开发 Fragment
- 使用 ListFragment
- 管理 Fragment，并让 Fragment 与 Activity 通信
- Fragment 的生命周期

Activity 是 Android 应用的重要组成单元之一（另外三个是 Service、BroadcastReceiver 和 ContentProvider），而 Activity 又是 Android 应用最常见的组件之一。前面看到的示例通常都只包含一个 Activity，但在实际应用中这是不大可能的，往往包括多个 Activity，不同的 Activity 向用户呈现不同的操作界面。Android 应用的多个 Activity 组成 Activity 栈，当前活动的 Activity 位于栈顶。

有 Web 开发经验的读者对 Servlet 的概念应该比较熟悉。实际上 Activity 对于 Android 应用的作用有点类似于 Servlet 对于 Web 应用的作用——一个 Web 应用通常都需要 *N* 个 Servlet 组成（JSP 的本质依然是 Servlet）；那么一个 Android 应用通常也需要 *N* 个 Activity 组成。对于 Web 应用而言，Servlet（把 JSP 也统一成 Servlet）主要负责与用户交互，并向用户呈现应用状态；对于 Android 应用而言，Activity 大致也具有相同的功能。

当 Activity 处于 Android 应用中运行时，同样受系统控制，有其自身的生命周期，本章深入介绍 Activity 的相关知识。

4.1 建立、配置和使用 Activity

Activity 是 Android 应用中最重要、最常见的应用组件（此处的组件是粗粒度的系统组成部分，并非指界面控件：widget）。Android 应用开发的一个重要组成部分就是开发 Activity，下面将会详细介绍 Activity 开发、配置的相关知识。

4.1.1 Activity

与开发 Web 应用时建立 Servlet 类相似，建立自己的 Activity 也需要继承 Activity 基类。当然，在不同应用场景下，有时也要求继承 Activity 的子类。例如，如果应用程序界面只包括列表，则可以让应用程序继承 ListActivity；如果应用程序界面需要实现标签页效果，则可以让应用程序继承 TabActivity。

图 4.1 显示了 Android 提供的 Activity 类。

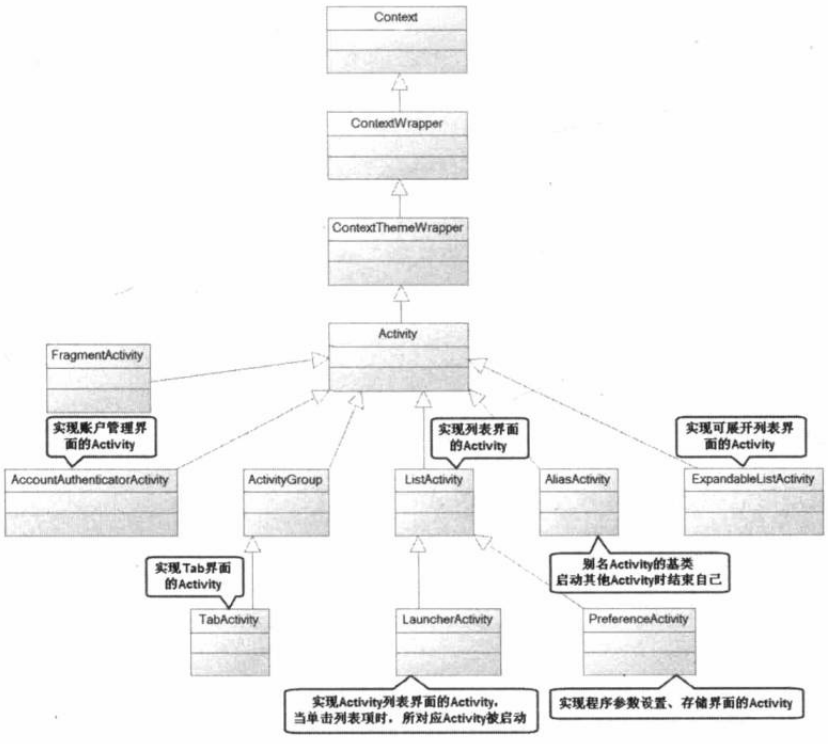


图 4.1 各种 Activity 基类

如图 4.1 所示，Activity 类间接或直接地继承了 Context、ContextWrapper、ContextThemeWrapper 等基类，因此 Activity 可以直接调用它们的方法。

与 Servlet 类似，当一个 Activity 类定义出来之后，这个 Activity 类何时被实例化、它所包含的方法何时被调用，这些都不是由开发者决定的，都应该由 Android 系统来决定。

为了让 Servlet 能响应用户请求，开发者需要重写 HttpServlet 的 doRequest(..)、doResponse(..)方法，或重写 service(..)方法。Activity 与此类似，创建一个 Activity 也需要实现一个或多个方法，其中最常见的就是实现 onCreate(Bundle status)方法，该方法将会在 Activity 创建时被回调，该方法调用 Activity 的 setContentView(View view)方法来显示要展示的 View。为了管理应用程序界面中的各组件，调用 Activity 的 findViewById(int id)方法来获取程序界面中的组件，接下来修改各组件的属性和方法即可。

实例：用 LauncherActivity 开发启动 Activity 的列表

通过前几章的实例已经介绍了 Activity、ListActivity、TabActivity 等基类的用法，接下来开发一个继承 LauncherActivity 的应用。

LauncherActivity 继承了 ListActivity，因此它本质上也是一个开发列表界面的 Activity，但它开发出来的列表界面与普通列表界面有所不同。它开发出来的列表界面中的每个列表项都对应于一个 Intent，因此当用户单击不同的列表项时，应用程序会自动启动对应的 Activity。

使用 LauncherActivity 的方法并不难，由于依然是一个 ListActivity，因此同样需要为它设置 Adapter——既可使用简单的 ArrayAdapter，也可使用 SimpleAdapter，当然还可以扩展 BaseAdapter 来实现自己的 Adapter。与使用普通 ListActivity 不同的是，继承 LauncherActivity 时通常应该重写 Intent intentForPosition(int position)方法，该方法根据不同列表项返回不同的 Intent（用于启动不同的 Activity）。

下面是 LauncherActivity 的一个子类的代码。

程序清单：codes\04\4.1\OtherActivity\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends LauncherActivity
{
    // 定义两个 Activity 的名称
    String[] names = {"设置程序参数" , "查看星际兵种"};
    // 定义两个 Activity 对应的实现类
    Class<?>[] clazzs = {PreferenceActivityTest.class
        , ExpandableListActivityTest.class};
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1 , names);
        // 设置该窗口显示的列表所需的 Adapter
        setListAdapter(adapter);
    }
    // 根据列表项返回指定 Activity 对应的 Intent
    @Override
    public Intent intentForPosition(int position)
    {
        return new Intent(MainActivity.this , clazzs[position]);
    }
}
```

上面程序中的第一行粗体字代码为该 ListActivity 设置了所需的内容 Adapter，第二段粗体

字代码则根据用户单击的列表项去启动对应的 Activity。

上面的程序还用到了如下两个 Activity。

- ExpandableListActivityTest：它是 ExpandableListActivity 的子类，用于显示一个可展开的列表窗口。
- PreferenceActivityTest：它是 PreferenceActivity 的子类，用于显示一个显示设置选项参数并进行保存的窗口。

实例：使用 ExpandableListActivity 实现可展开的 Activity

先看 ExpandableListActivityTest，它继承了 ExpandableListActivity 基类，ExpandableListActivity 的用法与前面介绍的 ExpandableListView 的用法基本相似，只要为该 Activity 传入一个 ExpandableListAdapter 对象即可，接下来 ExpandableListActivity 将会生成一个显示可展开列表的窗口。

下面是 ExpandableListActivityTest 的代码。

程序清单：codes\04\4.1\OtherActivity\app\src\main\java\org\crazyit\app\ExpandableListActivityTest.java

```
public class ExpandableListActivityTest extends ExpandableListActivity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ExpandableListAdapter adapter = new BaseExpandableListAdapter()
        {
            int[] logos = new int[]
            {
                R.drawable.p,
                R.drawable.z,
                R.drawable.t
            };
             private String[] armTypes = new String[]
                { "神族兵种", "虫族兵种", "人族兵种"};
            private String[][] arms = new String[][]
            {
                { "狂战士", "龙骑士", "黑暗圣堂", "电兵" },
                { "小狗", "刺蛇", "飞龙", "自爆飞机" },
                { "机枪兵", "护士MM" , "幽灵" }
            };
            // 获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition, int childPosition)
            {
                return arms[groupPosition][childPosition];
            }
            @Override
            public long getChildId(int groupPosition, int childPosition)
            {
                return childPosition;
            }
            @Override
            public int getChildrenCount(int groupPosition)
            {
                return arms[groupPosition].length;
            }
            private TextView getTextView()
            {
                AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                        ViewGroup.LayoutParams.MATCH_PARENT, 64);
```

```
        TextView textView = new TextView(ExpandableListActivityTest.
        this);
        textView.setLayoutParams(lp);
        textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
        textView.setPadding(36, 0, 0, 0);
        textView.setTextSize(20);
        return textView;
    }
    // 该方法决定每个子选项的外观
    @Override
    public View getChildView(int groupPosition, int childPosition,
        boolean isLastChild, View convertView, ViewGroup parent)
    {
        TextView textView = getTextView();
        textView.setText(getChild(groupPosition, childPosition).
        toString());
        return textView;
    }
    // 获取指定组位置处的组数据
    @Override
    public Object getGroup(int groupPosition)
    {
        return armTypes[groupPosition];
    }
    @Override
    public int getGroupCount()
    {
        return armTypes.length;
    }
    @Override
    public long getGroupId(int groupPosition)
    {
        return groupPosition;
    }
    // 该方法决定每个组选项的外观
    @Override
    public View getGroupView(int groupPosition, boolean isExpanded,
        View convertView, ViewGroup parent)
    {
        LinearLayout ll = new LinearLayout(
            ExpandableListActivityTest.this);
        ll.setOrientation(LinearLayout.HORIZONTAL);
        ImageView logo = new ImageView(
            ExpandableListActivityTest.this);
        logo.setImageResource(logos[groupPosition]);
        ll.addView(logo);
        TextView textView = getTextView();
        textView.setText(getGroup(groupPosition).toString());
        ll.addView(textView);
        return ll;
    }
    @Override
    public boolean isChildSelectable(int groupPosition,
        int childPosition)
    {
        return true;
    }
    @Override
    public boolean hasStableIds()
    {
        return true;
    }
};
// 设置该窗口显示列表
```

```
            setListAdapter(adapter);
        }
    }
```

上面程序中的粗体字代码为 ExpandableListActivity 设置了一个 ExpandableListAdapter 对象，即可使得该 Activity 显示可展开列表的窗口。

实例：PreferenceActivity 结合 PreferenceFragment 实现参数设置界面

PreferenceActivity 是一个非常有用的基类，当我们开发一个 Android 应用程序时，不可避免地需要进行选项设置，这些选项设置会以参数的形式保存，习惯上我们会用 Preferences 进行保存。

提示：

关于 Preferences 的介绍，请参看本书第 8 章关于 SharedPreferences 的内容。

需要指出的是，如果 Android 应用程序中包含的某个 Activity 专门用于设置选项参数，那么 Android 为这种 Activity 提供了便捷的基类：PreferenceActivity。

一旦 Activity 继承了 PreferenceActivity，那么该 Activity 完全不需要自己控制 Preferences 的读写，PreferenceActivity 会为我们处理一切。

PreferenceActivity 与普通 Activity 不同，它不再使用普通的界面布局文件，而是使用选项设置的布局文件。选项设置的布局文件以 PreferenceScreen 作为根元素——它表明定义一个参数设置的界面布局。

为了创建一个 PreferenceActivity，需要先创建一个对应的界面布局文件。从 Android 3.0 开始，Android 不再推荐直接让 PreferenceActivity 加载选项设置的布局文件，而是建议将 PreferenceActivity 与 PreferenceFragment 结合使用，其中 PreferenceActivity 只负责加载选项设置列表的布局文件，PreferenceFragment 才负责加载选项设置的布局文件。

本实例中 PreferenceActivity 加载的选项设置列表的布局文件如下。

程序清单：codes\04\4.1\OtherActivity\app\src\main\res\xml\preference_headers.xml

```
<?xml version="1.0" encoding="utf-8"?>
<preference-headers
    xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 指定启动指定 PreferenceFragment 的列表项 -->
    <header android:fragment=
        "org.crazyit.app.PreferenceActivityTest$Prefs1Fragment"
        android:icon="@drawable/ic_settings_applications"
        android:title="程序选项设置"
        android:summary="设置应用的相关选项" />
    <!-- 指定启动指定 PreferenceFragment 的列表项 -->
    <header android:fragment=
        "org.crazyit.app.PreferenceActivityTest$Prefs2Fragment"
        android:icon="@drawable/ic_settings_display"
        android:title="界面选项设置 "
        android:summary="设置显示界面的相关选项">
        <!-- 使用 extra 可向 Activity 传入额外的数据 -->
        <extra android:name="website"
            android:value="www.crazyit.org" />
    </header>
    <!-- 使用 Intent 启动指定 Activity 的列表项 -->
    <header
        android:icon="@drawable/ic_settings_display"
        android:title="使用 Intent"
```

```
        android:summary="使用 Intent 启动某个 Activity">
        <intent android:action="android.intent.action.VIEW"
            android:data="http://www.crazyit.org" />
    </header>
</preference-headers>
```

上面的布局文件中定义了三个列表项，其中前两个列表项通过 android:fragment 选项指定启动相应的 PreferenceFragment；第三个列表项通过<intent.../>子元素启动指定的 Activity。

提示：

关于 Intent 与<intent.../>的介绍，请参考本书下一章的内容。

上面的布局文件中指定使用 Prefs1Fragment、Prefs2Fragment 两个内部类，为此我们将会在 PreferenceActivityTest 类中定义这两个内部类。下面是 PreferenceActivityTest 的代码。

程序清单：codes\04\4.1\OtherActivity\app\src\main\java\org\crazyit\app\PreferenceActivityTest.java

```
public class PreferenceActivityTest extends PreferenceActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 该方法用于为该界面设置一个标题按钮
        if (hasHeaders())
        {
            Button button = new Button(this);
            button.setText("设置操作");
            // 将该按钮添加到该界面上
            setListFooter(button);
        }
    }
    // 重写该方法，负责加载界面布局文件
    @Override
    public void onBuildHeaders(List<Header> target)
    {
        // 加载选项设置列表的布局文件
        loadHeadersFromResource(R.xml.preference_headers, target);
    }
    // 重写该方法，验证各 PreferenceFragment 是否有效
    @Override
    public boolean isValidFragment(String fragmentName)
    {
        return true;
    }
    public static class Prefs1Fragment extends PreferenceFragment
    {
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);
        }
    }
    public static class Prefs2Fragment extends PreferenceFragment
    {
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.display_prefs);
            // 获取传入该 Fragment 的参数
```

```
            String website = getArguments().getString("website");
            Toast.makeText(getActivity()
                , "网站域名是: " + website , Toast.LENGTH_LONG).show();
        }
    }
}
```

上面的 Activity 重写了 PreferenceActivity 的 public void onBuildHeaders(List<Header> target) 方法，重写该方法指定加载前面定义的 preference_headers.xml 界面布局文件。

上面的 Activity 中定义了两个 PreferenceFragment，它们需要分别加载 preferences.xml、display_prefs 两个选项设置的布局文件。

建立选项设置的布局文件按如下步骤进行。

① 用鼠标右击 Android Studio 项目管理面板上的 app 节点，然后在弹出的右键菜单中单击“New”→“Android resource file”菜单，如图 4.2 所示。

② Android Studio 弹出如图 4.3 所示的窗口，选择创建“XML”类型的资源文件，该文件默认保存在/res/xml 路径下。并选择该 XML 文件的根元素为 PreferenceScreen，然后单击“Finish”按钮完成创建。

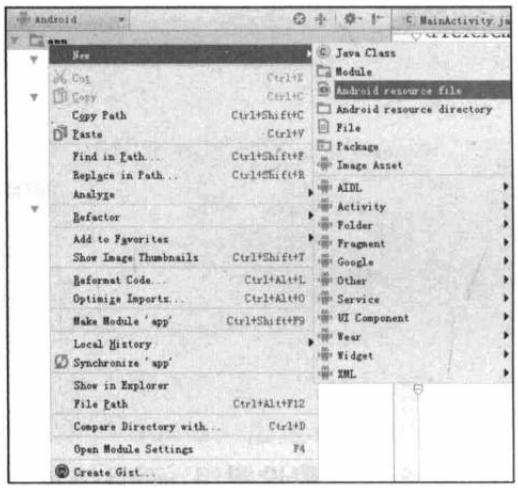


图 4.2 新建 Android 资源文件

图 4.3 创建 Android XML 文件

提示：

上面介绍的两步就是创建 Android 资源文件的通用步骤，包括界面布局文件、菜单资源文件、字符串资源等，都可通过这两步来进行创建。

接下来在 Android Studio 中打开 preferences.xml 进行编辑，Android Studio 为编辑该文件提供了良好的提示信息，该 XML 文件能接收哪些子元素、各子元素能包含哪些属性，Android Studio 中都有优秀的提示，如图 4.4 所示。

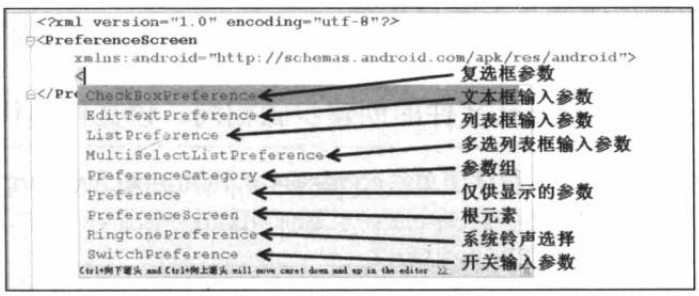


图 4.4 preferences.xml 文件中可添加的元素

③ 其中 PreferenceCategory 用于对参数选项进行分组，其他元素都用于设置相应的参数。编辑 preferences.xml 文件完成后，可以看到如下所示的界面布局文件。

程序清单：codes\04\4.1\OtherActivity\app\src\main\res\xml\preferences.xml

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

```
    xmlns:android="http://schemas.android.com/apk/res/android">
<!-- 设置系统铃声 -->
<RingtonePreference
    android:ringtoneType="all"
    android:title="设置铃声"
    android:summary="选择铃声（测试 RingtonePreference）"
    android:showDefault="true"
    android:key="ring_key"
    android:showSilent="true">
</RingtonePreference>
<PreferenceCategory android:title="个人信息设置组">
    <!-- 通过输入框填写用户名 -->
    <EditTextPreference
        android:key="name"
        android:title="填写用户名"
        android:summary="填写您的用户名（测试 EditTextPreference）"
        android:dialogTitle="您所使用的用户名为：" />
    <!-- 通过列表框选择性别 -->
    <ListPreference
        android:key="gender"
        android:title="性别"
        android:summary="选择您的性别（测试 ListPreference）"
        android:dialogTitle="ListPreference"
        android:entries="@array/gender_name_list"
        android:entryValues="@array/gender_value_list" />
</PreferenceCategory>
<PreferenceCategory android:title="系统功能设置组 ">
    <CheckBoxPreference
        android:key="autoSave"
        android:title="自动保存进度"
        android:summaryOn="自动保存：开启"
        android:summaryOff="自动保存：关闭"
        android:defaultValue="true" />
</PreferenceCategory>
</PreferenceScreen>
```

上面的界面布局文件中定义了一个参数设置界面，其中包括两个参数设置组，而且该参数设置界面全面应用了各种元素，这样方便读者以后查询。

定义了参数设置的界面布局文件之后，接下来在 PreferenceFragment 程序中使用该界面布局文件进行参数设置、保存十分简单，只要如下两步即可。

1. 让 Fragment 继承 PreferenceFragment。
2. 在 onCreate(Bundle savedInstanceState)方法中调用 addPreferencesFromResource(..)方法加载指定的界面布局文件。

上面的实例中还用到了选项设置布局文件 display_prefs.xml，该布局文件的创建步骤与 preferences.xml 文件的创建步骤相同。该文件的代码如下。

程序清单：codes\04\4.1\OtherActivity\app\src\main\res\xml\display_prefs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
<PreferenceCategory android:title="背景灯光组">
    <!-- 通过列表框选择灯光强度 -->
    <ListPreference
        android:key="light"
        android:title="灯光强度"
        android:summary="请选择灯光强度（测试 ListPreference）"
        android:dialogTitle="请选择灯光强度"
```

```
        android:entries="@array/light_strength_list"
        android:entryValues="@array/light_value_list" />
    </PreferenceCategory>
    <PreferenceCategory android:title="文字显示组 ">
        <!-- 通过 SwitchPreference 设置是否自动滚屏 -->
        <SwitchPreference
            android:key="autoScroll"
            android:title="自动滚屏"
            android:summaryOn="自动滚屏: 开启"
            android:summaryOff="自动滚屏: 关闭"
            android:defaultValue="true" />
    </PreferenceCategory>
    </PreferenceScreen>
```

至此，我们为该应用程序开发了三个 Activity 类，但这三个 Activity 还不能使用，必须在 AndroidManifest.xml 清单文件中配置 Activity 才行。

➤➤ 4.1.2 配置 Activity

Android 应用要求所有应用程序组件（Activity、Service、ContentProvider、BroadcastReceiver）都必须显式进行配置。

只要为<application.../>元素添加<activity.../>子元素即可配置 Activity。例如如下的配置片段：

```
<activity android:name=".SampleActivity"
    android:icon="@drawable/small_pic.png"
    android:label="@string/freneticLabel"
    android:exported="true"
    android:launchMode="singleInstance">
    ...
</activity>
```

从上面的配置片段可以看出，配置 Activity 时通常指定如下几个属性。

- name：指定该 Activity 的实现类的类名。
- icon：指定该 Activity 对应的图标。
- label：指定该 Activity 的标签。
- exported：指定该 Activity 是否允许被其他应用调用。如果将该属性设为 true，那么该 Activity 将可以被其他应用调用。
- launchMode：指定该 Activity 的加载模式，该属性支持 standard、singleTop、singleTask 和 singleInstance 这 4 种加载模式。本章后面会详细介绍这 4 种加载模式。

除此之外，配置 Activity 时通常还需要指定一个或多个<intent-filter.../>元素，该元素用于指定该 Activity 可响应的 Intent。

提示：

关于 Intent 和 IntentFilter 的介绍，请参看本书下一章的介绍。

为了在 AndroidManifest.xml 文件中配置、管理上面的三个 Activity，可以在清单文件的<application.../>元素中增加如下三个<activity.../>子元素。

程序清单：codes\04\4.1\OtherActivity\app\src\main\AndroidManifest.xml

```
<activity android:name=".MainActivity"
    android:label="@string/app_name">
    <!-- 指定该 Activity 是程序的入口 -->
    <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <!-- 定义两个Activity -->
    <activity android:name=".ExpandableListActivityTest"
        android:label="查看星际兵种">
    </activity>
    <activity android:name=".PreferenceActivityTest"
        android:label="设置程序参数">
    </activity>
```

上面的配置片段配置了三个 Activity，其中第一个 Activity 还配置了一个<intent-filter.../>元素，该元素指定该 Activity 作为应用程序的入口。

运行上面的应用程序，将看到如图 4.5 所示的界面。

在图 4.5 所示的程序界面中，用户单击任意列表项即可启动对应的 Activity。例如，单击“设置程序参数”将会启动 PreferenceActivityTest，单击“查看星际兵种”将会启动 ExpandableListActivityTest。单击图 4.5 所示列表的第一个列表项，将看到如图 4.6 所示的界面。

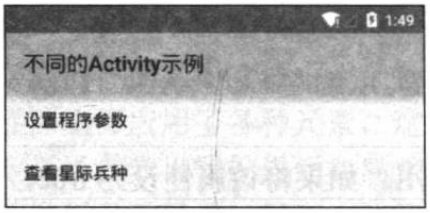


图 4.5 启动 Activity 的列表

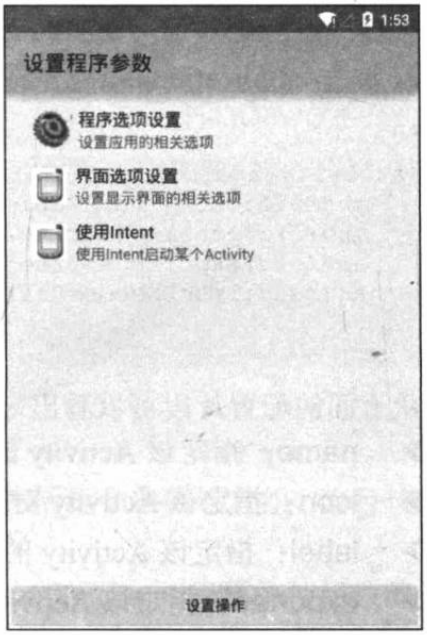


图 4.6 选项设置列表界面

图 4.6 所示界面就是利用 PreferenceActivity 生成的选项设置列表界面。这个界面只是包含三个列表项，其中前两个列表项用于启动 PreferenceFragment，最后一个列表项将会根据 Intent 启动其他 Activity。

单击图 4.6 所示界面中的第一个列表项，将可以看到如图 4.7 所示的界面。

图 4.7 所示界面就是利用 PreferenceFragment 生成的选项设置界面，这个界面非常漂亮，而且系统会自动将设置的参数永久地保存到系统中——这都得益于 PreferenceActivity。例如，我们单击“填写用户名”列表项，系统将会显示如图 4.8 所示的输入框。

在图 4.8 所示对话框中输入用户名，单击“OK”按钮，程序将会自动保存设置的选项参数。程序所设置的参数将会保存在/data/data/<应用程序包名>/shared_prefs 路径下，文件名为<应用程序包名>_preferences.xml。

运行本程序系统将会在 /data/data/org.crazyit.app/shared_prefs/ 路径下生成一个 org.crazyit.app_preferences.xml 文件。打开 DDMS 的 File Explorer 面板，进入该参数的参数文件的保存路径，将可以看到如图 4.9 所示的窗口。

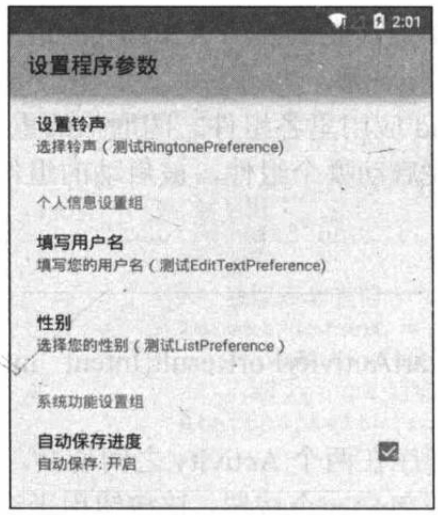


图 4.7 PreferenceFragment 生成的选项设置界面

图 4.8 使用 EditTextPreference 生成的输入框

通过图 4.9 所示窗口把其中的“org.crazyit.app_preferences.xml”文件导出来，该文件的内容为：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="name">crazyit.org</string>
    <string name="gender">male</string>
    <string name="light">medium</string>
    <boolean name="autoSave" value="true" />
    <boolean name="autoScroll" value="true" />
</map>
```

上面的文件就是程序运行设置的参数。

如果单击图 4.5 所示列表的第二个列表项，将可以看到如图 4.10 所示的界面。

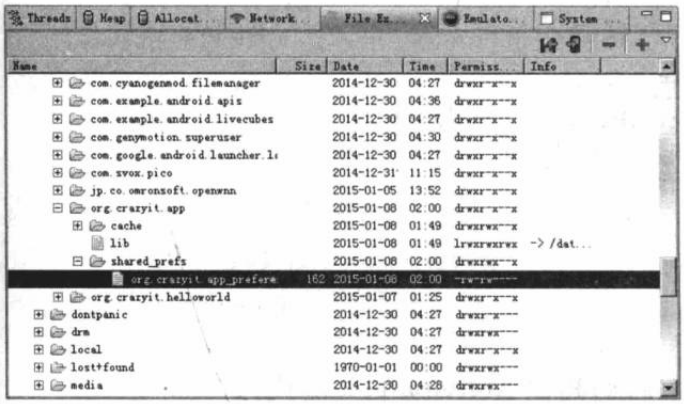


图 4.9 查看系统生成的参数文件

图 4.10 ExpandableListActivity 生成的可展开的列表界面

4.1.3 启动、关闭 Activity

正如前面所介绍的，一个 Android 应用通常都会包含多个 Activity，但只有一个 Activity 会作为程序的入口——当该 Android 应用运行时将会自动启动并执行该 Activity。至于应用中的其他 Activity，通常都由入口 Activity 启动，或由入口 Activity 启动的 Activity 启动。

Activity 启动其他 Activity 有如下两个方法。

- startActivity(Intent intent)：启动其他 Activity。
- startActivityForResult(Intent intent, int requestCode)：以指定的请求码（requestCode）启动 Activity，而且程序将会获取新启动的 Activity 返回的结果（通过重写 onActivityResult(..)方法来获取）。

启动 Activity 时可指定一个 requestCode 参数，该参数代表了启动 Activity 的请求码。这个请求码的值由开发者根据业务自行设置，用于标识请求来源。

上面两个方法都用到了 Intent 参数，Intent 是 Android 应用里各组件之间通信的重要方式，一个 Activity 通过 Intent 来表达自己“意图”——想要启动哪个组件，被启动的组件既可是 Activity 组件，也可是 Service 组件。

Android 为关闭 Activity 准备了如下两个方法。

- finish()：结束当前 Activity。
- finishActivity(int requestCode)：结束以 startActivityForResult(Intent intent, int requestCode)方法启动的 Activity。

下面的示例程序示范了如何启动 Activity，并允许程序在两个 Activity 之间切换。

程序的第一个 Activity 的界面布局很简单，该界面只包含一个按钮，该按钮用于进入第二个 Activity。此处不给出界面布局文件，Java 代码如下。

程序清单：codes\04\4.1\StartActivity\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取应用程序中的 bn 按钮
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建需要启动的 Activity 对应的 Intent
                Intent intent = new Intent(MainActivity.this,
                    SecondActivity.class);
                // 启动 intent 对应的 Activity
                startActivity(intent);
            }
        });
    }
}
```

上面程序中的粗体字代码就是在 Activity 中启动其他 Activity 的关键代码。

程序中第二个 Activity 的界面同样简单，它只包含两个按钮，其中一个按钮用于简单地返回上一个 Activity（并不关闭自己）；另一个按钮用于结束自己并返回上一个 Activity。此处不给出第二个 Activity 的界面布局文件。

下面是第二个 Activity 的 Java 代码。

程序清单：codes\04\4.1\StartActivity\app\src\main\java\org\crazyit\app\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        // 获取应用程序中的 previous 按钮
```

```
        Button previous = (Button) findViewById(R.id.previous);
        // 获取应用程序中的close按钮
        Button close = (Button) findViewById(R.id.close);
        // 为previous按钮绑定事件监听器
        previous.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取启动当前Activity的上一个Intent
                Intent intent = new Intent(SecondActivity.this,
                    MainActivity.class);
                // 启动intent对应的Activity
                startActivity(intent);
            }
        });
        // 为close按钮绑定事件监听器
        close.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取启动当前Activity的上一个Intent
                Intent intent = new Intent(SecondActivity.this,
                    MainActivity.class);
                // 启动intent对应的Activity
                startActivity(intent);
                // 结束当前Activity
                finish();
            }
        });
    }
}
```

上面程序中两个按钮的监听器里的处理代码只有一行区别：finish()，如果有这行代码，则表明该Activity会结束自己。

4.1.4 使用Bundle在Activity之间交换数据

当一个Activity启动另一个Activity时，常常会有一些数据需要传过去——这就像Web应用从一个Servlet跳到另一个Servlet时，习惯把需要交换的数据放入requestScope、sessionScope中。对于Activity而言，在Activity之间进行数据交换更简单，因为两个Activity之间本来就有一个“信使”：Intent，因此我们主要将需要交换的数据放入Intent中即可。

Intent提供了多个重载的方法来“携带”额外的数据，如下所示。

- putExtras(Bundle data)：向Intent中放入需要“携带”的数据包。
- Bundle getExtras()：取出Intent中所“携带”的数据包。
- putExtra(String name, Xxx value)：向Intent中按key-value对的形式存入数据。
- getXxxExtra(String name)：从Intent中按key取出指定类型的数据。

上面方法中的Bundle就是一个简单的数据携带包，该Bundle对象包含了多个方法来存入数据。

- putXxx(String key , Xxx data)：向Bundle中放入Int、Long等各种类型的数据。
- putSerializable(String key , Serializable data)：向Bundle中放入一个可序列化的对象。

为了取出Bundle数据携带包里的数据，Bundle提供了如下方法。

- getXxx(String key)：从Bundle中取出Int、Long等各种类型的数据。

➢ getSerializable(String key, Serializable data)：从 Bundle 中取出一个可序列化的对象。

从上面的介绍不难看出，Intent 主要通过 Bundle 对象来携带数据，因此 Intent 提供了 putExtras()和 getExtras()两个方法。除此之外，Intent 也提供了多个重载的 putExtra(String name, Xxx value)、getXxxExtra(String name)，那么这些方法存取的数据在哪里呢？其实 Intent 提供的 putExtra(String name, Xxx value)、getXxxExtra(String name)方法，只是两个便捷的方法，这些方法依然是存取 Intent 所携带的 Bundle 中的数据。

提示：

Intent 的 putExtra(String name, Xxx value)方法是“智能”的，当程序调用 Intent 的 putExtra(String name, Xxx value)方法向 Intent 中存入数据时，如果该 Intent 中已经携带了 Bundle 对象，则该方法直接向 Intent 所携带的 Bundle 中存入数据；如果 Intent 还没有携带 Bundle 对象，putExtra(String name, Xxx value)方法会先为 Intent 创建一个 Bundle，再向 Bundle 中存入数据。

下面通过一个实例应用来介绍两个 Activity 之间如何通过 Bundle 交换数据。

实例：用第二个 Activity 处理注册信息

本实例程序包含两个 Activity，其中第一个 Activity 用于收集用户的输入信息，当用户单击该 Activity 的“注册”按钮时，应用进入第二个 Activity，第二个 Activity 将会获取第一个 Activity 中的数据。

下面是第一个 Activity 的界面布局文件。

程序清单：codes\04\4.1\BundleTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="请输入您的注册信息"
    android:textSize="20sp"/>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="用户名:"
    android:textSize="16sp"/>
<!-- 定义一个 EditText，用于收集用户的账号 -->
<EditText
    android:id="@+id/name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="请填写想注册的账号"
    android:selectAllOnFocus="true"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="密码:"
    android:textSize="16sp"/>
<!-- 用于收集用户的密码 -->
```

```
<EditText
    android:id="@+id/passwd"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:password="true"
    android:selectAllOnFocus="true"/>
</TableRow>
<TableRow>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="性别:"
    android:textSize="16sp"/>
<!-- 定义一组单选钮，用于收集用户注册的性别 -->
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
<RadioButton
    android:id="@+id/male"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="男"
    android:textSize="16sp"/>
<RadioButton
    android:id="@+id/female"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="女"
    android:textSize="16sp"/>
</RadioGroup>
</TableRow>
<Button
    android:id="@+id/bn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="注册"
    android:textSize="16sp"/>
</TableLayout>
```

该界面布局对应的 Java 类代码如下。

程序清单：codes\04\4.1\BundleTest\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v)
            {
                EditText name = (EditText)findViewById(R.id.name);
                EditText passwd = (EditText)findViewById(R.id.passwd);
                RadioButton male = (RadioButton) findViewById(R.id.male);
                String gender = male.isChecked() ? "男 " : "女";
                Person p = new Person(name.getText().toString(), passwd
                        .getText().toString(), gender);
                // 创建一个 Bundle 对象
                Bundle data = new Bundle();
```

```
                data.putSerializable("person", p);
                // 创建一个 Intent
                Intent intent = new Intent(MainActivity.this,
                    ResultActivity.class);
                intent.putExtras(data);
                // 启动 intent 对应的 Activity
                startActivity(intent);
            }
        });
    }
}
```

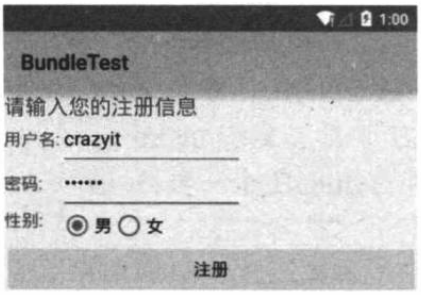


图 4.11　注册界面

上面程序中的粗体字代码根据用户输入创建了一个 Person 对象，Person 类只是一个简单的 DTO 对象，该 Person 类实现了 java.io.Serializable 接口，因此 Person 对象是可序列化的。

上面的程序创建了一个 Bundle 对象，并调用 putSerializable ("person" , p)将 Person 对象放入该 Bundle 中，然后再使用 Intent 来“携带”这个 Bundle，这样即可将 Person 对象传入第二个 Activity。

运行该程序，第一个 Activity 显示的界面如图 4.11 所示。

当用户单击“注册”按钮时，程序将会启动 ResultActivity，并将用户输入的数据传入该 Activity。下面是 ResultActivity 的界面布局文件。

程序清单：codes\04\4.1\BundleTest\res\layout\result.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
<!-- 定义三个 TextView，用于显示用户输入的数据 -->
<TextView
    android:id="@+id/name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"/>
<TextView
    android:id="@+id/passwd"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"/>
<TextView
    android:id="@+id/gender"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"/>
</LinearLayout>
```

这个 Activity 的程序将会从 Bundle 中取出前一个 Activity 传过来的数据，并将它们显示出来。该 Activity 的 Java 代码如下。

程序清单：codes\04\4.1\BundleTest\app\src\main\java\org\crazyit\app\ResultActivity.java

```
public class ResultActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.result);
        TextView name = (TextView) findViewById(R.id.name);
        TextView passwd = (TextView) findViewById(R.id.passwd);
        TextView gender = (TextView) findViewById(R.id.gender);
        // 获取启动该Activity的Intent
        Intent intent = getIntent();
        // 直接通过Intent取出它所携带的Bundle数据包中的数据
        Person p = (Person) intent.getSerializableExtra("person");
        name.setText("您的用户名为：" + p.getName());
        passwd.setText("您的密码为：" + p.getPasswd());
        gender.setText("您的性别为：" + p.getGender());
    }
}
```

上面程序中的粗体字代码用于获取前一个 Activity 所传过来的数据，至于该 Activity 获取数据之后如何处理它们，完全由开发者自己决定。本应用程序只是将获取的数据显示出来，用户在图4.11所示界面中输入注册信息之后，单击“注册”按钮，将看到如图 4.12 所示的界面。

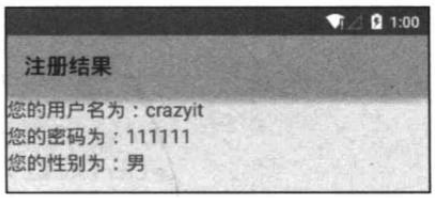


图 4.12 注册成功

4.1.5 启动其他 Activity 并返回结果

前面已经提到，Activity 还提供了一个 startActivityForResult(Intent intent, int requestCode) 方法来启动其他 Activity。该方法用于启动指定 Activity，而且期望获取指定 Activity 返回的结果。这种请求对于实际应用也是很常见的，例如应用程序第一个界面需要用户进行选择——但需要选择的列表数据比较复杂，必须启动另一个 Activity 让用户选择。当用户在第二个 Activity 中选择完成后，程序返回第一个 Activity，第一个 Activity 必须能获取并显示用户在第二个 Activity 中选择的结果。在这种应用场景下，也是通过 Bundle 进行数据交换的。

为了获取被启动的 Activity 所返回的结果，需要从两方面着手。

- 当前 Activity 需要重写 onActivityResult(int requestCode , int resultCode , Intent intent)，当被启动的 Activity 返回结果时，该方法将会被触发，其中 requestCode 代表请求码，而 resultCode 代表 Activity 返回的结果码，这个结果码也是由开发者根据业务自行设定的。
- 被启动的 Activity 需要调用 setResult()方法设置处理结果。

一个 Activity 中可能包含多个按钮，并调用多个 startActivityForResult()方法来打开多个不同的 Activity 处理不同的业务，当这些新 Activity 关闭后，系统都将回调前面 Activity 的 onActivityResult(int requestCode, int resultCode, Intent data)方法。为了知道该方法是由哪个请求的结果所触发的，可利用 requestCode 请求码；为了知道返回的数据来自于哪个新的 Activity，可利用 resultCode 结果码。

下面通过一个实例来介绍如何启动 Activity 并获取被启动 Activity 的结果。

实例：用第二个 Activity 让用户选择信息

本实例程序也包含两个 Activity，第一个 Activity 的界面布局比较简单，它只包含一个按钮和一个文本框，故此处不再给出界面布局文件。第一个 Activity 对应的 Java 代码如下。

程序清单：codes\04\4.1\ActivityForResult\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends Activity
{
```

```
    Button bn;
    EditText city;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面上的组件
        bn = (Button) findViewById(R.id.bn);
        city = (EditText) findViewById(R.id.city);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建需要对应于目标 Activity 的 Intent
                Intent intent = new Intent(MainActivity.this,
                        SelectCityActivity.class);
                // 启动指定 Activity 并等待返回的结果，其中 0 是请求码，用于标识该请求
                startActivityForResult(intent, 0);
            }
        });
    }
    ...
}
```

上面程序中的粗体字代码用于启动 ActivityForResult，并等待该 Activity 返回的结果——但问题是，该 Activity 如何获取 ActivityForResult 返回的结果呢？当前 Activity 启动 SelectCityActivity 之后，SelectCityActivity 何时返回结果是不确定的，因此当前 Activity 无法去获取 SelectCityActivity 返回的结果。

为了让当前 Activity 获取 SelectCityActivity 所返回的结果，开发者应该重写 onActivityResult()方法——当被启动的 SelectCityActivity 返回结果时，onActivityResult()方法将会被回调。

因此还需要为上面的 ActivityForResult 添加如下方法。

程序清单：codes\04\4.1\ActivityForResult\app\src\main\java\org\crazyit\activity\MainActivity.java

```
// 重写该方法，该方法以回调的方式来获取指定 Activity 返回的结果
@Override
public void onActivityResult(int requestCode
    , int resultCode, Intent intent)
{
    // 当 requestCode、resultCode 同时为 0 时，也就是处理特定的结果
    if (requestCode == 0 && resultCode == 0)
    {
        // 取出 Intent 里的 Extras 数据
        Bundle data = intent.getExtras();
        // 取出 Bundle 中的数据
        String resultCity = data.getString("city");
        // 修改 city 文本框的内容
        city.setText(resultCity);
    }
}
```

运行该程序，将看到如图 4.13 所示的界面。

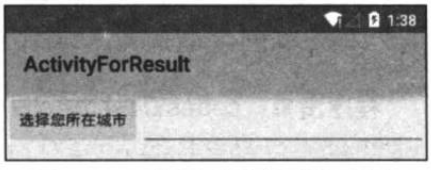


图 4.13 让用户选择所在城市

单击图 4.13 所示界面中的“选择您所在城市”按钮，系统将会启动 SelectCityActivity，该 SelectCityActivity 将会显示一个可展开的列表。该 SelectCityActivity 无须

界面布局文件。该 SelectCityActivity 的 Java 代码如下。

程序清单：codes\04\4.1\ActivityForResult\app\src\main\java\org\crazyit\app\SelectCityActivity.java

```
public class SelectCityActivity extends ExpandableListActivity
{
    // 定义省份数组
    private String[] provinces = new String[]
    { "广东", "广西", "湖南"};
    private String[][] cities = new String[][]
    {
        { "广州", "深圳", "珠海", "中山" },
        { "桂林", "柳州", "南宁", "北海" },
        { "长沙", "岳阳" , "衡阳", "株洲" }
    };
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ExpandableListAdapter adapter = new BaseExpandableListAdapter()
        {
            // 获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition, int childPosition)
            {
                return cities[groupPosition][childPosition];
            }
            @Override
            public long getChildId(int groupPosition, int childPosition)
            {
                return childPosition;
            }
            @Override
            public int getChildrenCount(int groupPosition)
            {
                return cities[groupPosition].length;
            }
            private TextView getTextView()
            {
                AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                        ViewGroup.LayoutParams.MATCH_PARENT, 64);
                TextView textView = new TextView(SelectCityActivity.this);
                textView.setLayoutParams(lp);
                textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
                textView.setPadding(36, 0, 0, 0);
                textView.setTextSize(20);
                return textView;
            }
            // 该方法决定每个子选项的外观
            @Override
            public View getChildView(int groupPosition, int childPosition,
                    boolean isLastChild, View convertView, ViewGroup parent)
            {
                TextView textView = getTextView();
                textView.setText(getChild(groupPosition, childPosition)
                        .toString());
                return textView;
            }
            // 获取指定组位置处的组数据
            @Override
            public Object getGroup(int groupPosition)
            {
                return provinces[groupPosition];
            }
            @Override
```

```
        public int getGroupCount()
        {
            return provinces.length;
        }
        @Override
        public long getGroupId(int groupPosition)
        {
            return groupPosition;
        }
        // 该方法决定每个组选项的外观
        @Override
        public View getGroupView(int groupPosition, boolean isExpanded,
                View convertView, ViewGroup parent)
        {
            LinearLayout ll = new LinearLayout(SelectCityActivity.this);
            ll.setOrientation(LinearLayout.HORIZONTAL);
            ImageView logo = new ImageView(SelectCityActivity.this);
            ll.addView(logo);
            TextView textView = getTextView();
            textView.setText(getGroup(groupPosition).toString());
            ll.addView(textView);
            return ll;
        }
        @Override
        public boolean isChildSelectable(int groupPosition,
                int childPosition)
        {
            return true;
        }
        @Override
        public boolean hasStableIds()
        {
            return true;
        }
    };
    // 设置该窗口显示列表
    setListAdapter(adapter);
    getExpandableListView().setOnChildClickListener(
        new OnChildClickListener()
        {
            @Override
            public boolean onChildClick(ExpandableListView parent,
                    View source, int groupPosition, int childPosition,
                    long id)
            {
                // 获取启动该 Activity 之前的 Activity 对应的 Intent
                Intent intent = getIntent();
                intent.putExtra("city",
                    cities[groupPosition][childPosition]);
                // 设置该 SelectCityActivity 的结果码，并设置结束之后退回的 Activity
                SelectCityActivity.this.setResult(0, intent);
                // 结束 SelectCityActivity
                SelectCityActivity.this.finish();
                return false;
            }
        });
    }
}
```

上面的 Activity 只是一个普通的显示可展开列表的 Activity，程序还为该 Activity 的各子列表项绑定了事件监听器，当用户单击子列表项时，该 Activity 将会把用户选择城市返回给上一个 Activity。

当上一个 Activity 获取 SelectCityActivity 选择城市之后，将会把该结果显示在图 4.13 所示界面右边的文本框内。

4.2 Activity 的回调机制

有 Web 开发经验的读者都知道：当一个 Servlet 开发出来之后，该 Servlet 运行于 Web 服务器中。服务器何时创建 Servlet 的实例，何时回调 Servlet 的方法向用户生成响应，程序员无法控制，这种回调由服务器自行决定。

前面已经提到，Android 应用中的 Activity 与 Web 应用中的 Servlet 有点相似，也就是说，Activity 被开发出来之后，开发者只要在 AndroidManifest.xml 文件中配置该 Activity 即可。至于该 Activity 何时被实例化，它的方法何时被调用，对开发者来说是完全透明的。

当开发者开发一个 Servlet 时，根据不同的需求场景，可能需要选择性地实现如下方法。

- init(ServletConfig config)
- destroy()
- doGet(HttpServletRequest req, HttpServletResponse resp)
- doPost(HttpServletRequest req, HttpServletResponse resp)
- service(HttpServletRequest req, HttpServletResponse resp)

当把这个 Servlet 部署在 Web 应用中之后，Web 服务器将会在特定的时刻，调用该 Servlet 上面的各种方法——这种调用就被称为回调。

所谓回调，在实现具有通用性质的应用架构时非常常见：对于一个具有通用性质的程序架构来说，程序架构完成整个应用的通用功能、流程，但在某个特定的点上，需要一段业务相关的代码——通用的程序架构无法实现这段代码，那么程序架构会在这个点上留一个“空”。

对于 Java 程序来说，程序架构在某个点上留的“空”，可以以如下两种方式存在。

- **以接口形式存在：**该接口由开发者实现，实现该接口时将会实现该接口的方法，那么通用的程序架构就会回调该方法来完成业务相关的处理。
- **以抽象方法（也可以是非抽象方法）的形式存在：**这就是 Activity 的实现形式。在这些特定的点上方法已经被定义了，如 onCreate、onActivityResult 等方法，开发者可以有选择性地重写这些方法，通用的程序架构就会回调该方法来完成业务相关的处理。

提示：

回调机制的第一种实现方式就是典型的命令者模式，关于命令者模式请参考疯狂 Java 体系的《轻量级 Java EE 企业应用实战》第 9 章。

前面介绍的事件处理也用到了回调机制：当开发者开发一个组件时，如果开发者需要该组件能响应特定的事件，则可以有选择性地实现该组件的特定方法——当用户在该组件上激发某个事件时，该组件上特定的方法就会回调。

Activity 的回调机制也与此类似，当 Activity 被部署在 Android 应用中之后，随着应用程序的运行，Activity 会不断地在不同的状态之间切换，该 Activity 中特定的方法就会被回调——开发者就可以有选择性地重写这些方法来加入业务相关的处理。

Activity 运行过程所处的不同状态也被称为生命周期，下面将详细介绍 Activity 的生命周期。

4.3 Activity的生命周期与加载模式

当Activity处于Android应用中运行时，它的活动状态由Android以Activity栈的形式管理，当前活动的Activity位于栈顶。随着不同应用的运行，每个Activity都有可能从活动状态转入非活动状态，也可能从非活动状态转入活动状态。

4.3.1 Activity的生命周期演示

归纳起来，Activity大致会经过如下4种状态。

- **运行状态**：当前Activity位于前台，用户可见，可以获得焦点。
- **暂停状态**：其他Activity位于前台，该Activity依然可见，只是不能获得焦点。
- **停止状态**：该Activity不可见，失去焦点。
- **销毁状态**：该Activity结束，或Activity所在的进程被结束。

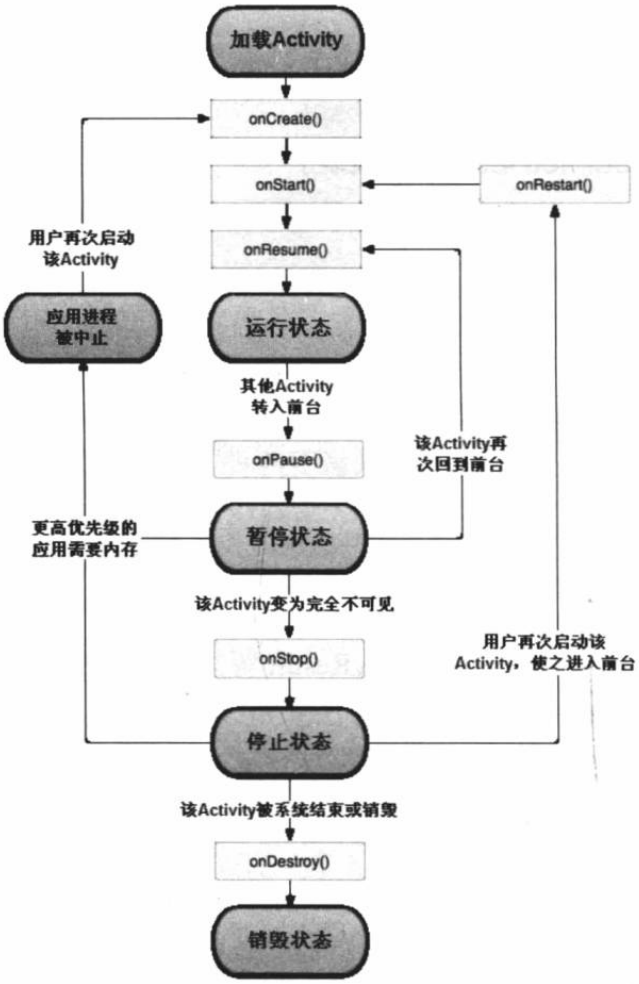


图4.14 Activity的生命周期及相关回调方法

图4.14（该图参照了Android官方文档，但细节方面比官方文档更准确）显示了Activity的生命周期及相关回调方法。

从图4.14可以看出，在Activity的生命周期中，如下方法会被系统回调。

- onCreate(Bundle savedStatus)：创建Activity时被回调。该方法只会被调用一次。
- onStart()：启动Activity时被回调。
- onRestart()：重新启动Activity时被回调。
- onResume()：恢复Activity时被回调。在onStart()方法后一定会回调onResume()方法。
- onPause()：暂停Activity时被回调。
- onStop()：停止Activity时被回调。
- onDestroy()：销毁Activity时被回调。该方法只会被调用一次。

正如开发Servlet时可以根据需要有选择性地覆盖指定方法一样，开发Activity时也可根据需要有选择性地覆盖指定方法。其中最常见的就是覆盖onCreate(Bundle saveStatus)方法——前面所有示例都覆盖了Activity的onCreate(Bundle saveStatus)方法，该方法用于对该Activity执行初始化。除此之外，覆盖onPause()方法也很常见，比如用户正在玩一个游戏，此时有电话进来，那么我们需要将当前游戏暂停，并保存该游戏的进行状态，这就可以通过覆盖onPause()方法来实现。接下来当用户再次切换到游戏状态时，onResume()方法已经被回调，因此可以通过重写onResume()方法来恢复游戏状态。

下面的Activity覆盖了上面的7个生命周期方法，并在每个方法中增加了一行记录日志代

码。该 Activity 的界面布局很简单，包含了两个按钮，其中一个用于启动一个对话框风格的 Activity；另一个用于退出该应用。此处不给出界面布局代码。该 Activity 的 Java 代码如下。

程序清单：codes\04\4.3\Lifecycle\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends Activity
{
    final String TAG = "--CrazyIt--";
    Button finish ,startActivity;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 输出日志
        Log.d(TAG, "-------onCreate------");
        finish = (Button) findViewById(R.id.finish);
        startActivity = (Button) findViewById(R.id.startActivity);
        // 为 startActivity 按钮绑定事件监听器
        startActivity.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Intent intent = new Intent(MainActivity.this
                        , SecondActivity.class);
                startActivity(intent);
            }
        });
        // 为 finish 按钮绑定事件监听器
        finish.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 结束该 Activity
                MainActivity.this.finish();
            }
        });
    }
    @Override
    public void onStart()
    {
        super.onStart();
        // 输出日志
        Log.d(TAG, "-------onStart------");
    }
    @Override
    public void onRestart()
    {
        super.onRestart();
        // 输出日志
        Log.d(TAG, "-------onRestart------");
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 输出日志
        Log.d(TAG, "-------onResume------");
    }
    @Override
    public void onPause()
```

```
    {
        super.onPause();
        // 输出日志
        Log.d(TAG, "-------onPause------");
    }
    @Override
    public void onStop()
    {
        super.onStop();
        // 输出日志
        Log.d(TAG, "-------onStop------");
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 输出日志
        Log.d(TAG, "-------onDestroy------");
    }
}
```

将该 Activity 设置成程序的入口 Activity，当程序启动时将会自动启动并执行该 Activity，此时将可以在 Android Studio 的 LogCat 窗口看到如图 4.15 所示的输出。

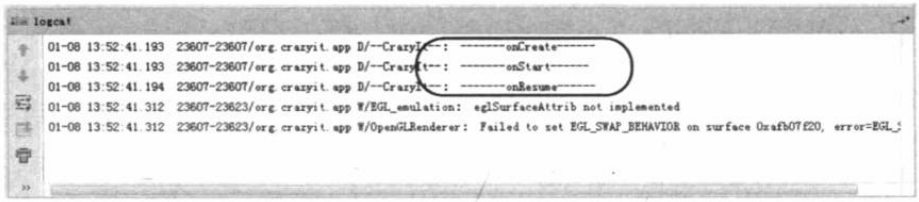


图 4.15　启动 Activity 时回调的方法

单击该程序界面上的“启动对话框风格的 Activity”按钮，对话框风格的 Activity 进入前台，虽然 MainActivity 不能获得焦点，但依然“部分可见”，此时该 Activity 进入“暂停”状态。此时将可以在 Android Studio 的 LogCat 窗口看到如图 4.16 所示的输出。

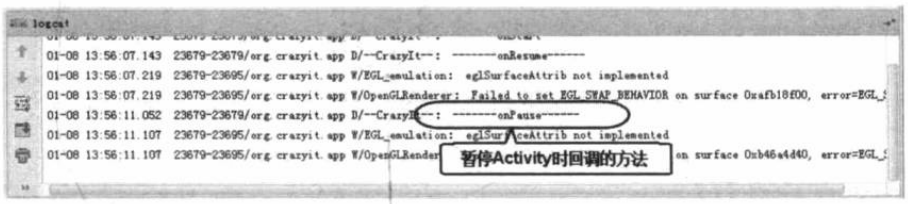


图 4.16　暂停 Activity 时回调的方法

在当前状态下，按下模拟器右边的 键，返回 Lifecycle Activity，该 Activity 再次进入“运行”状态，此时看到 LogCat 窗口有如图 4.17 所示的输出。

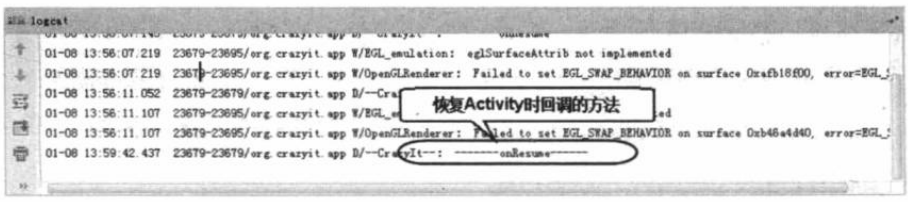


图 4.17 恢复 Activity 时回调的方法

在当前程序运行状态下，按下模拟器右边的 键，返回系统桌面，当前该 Activity 将失去焦点且不可见。但该 Activity 并未被销毁，它进入“停止”状态，此时看到 LogCat 窗口有如图 4.18 所示的输出。

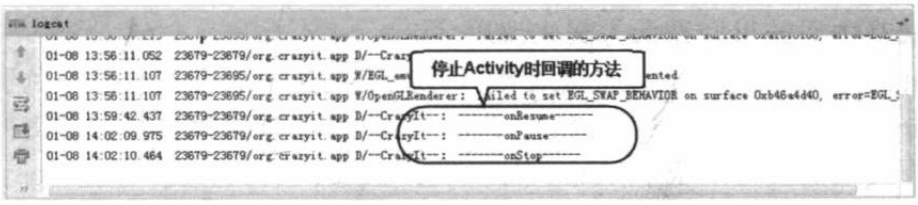


图 4.18 停止 Activity 时回调的方法

在模拟器程序列表处再次找到该应用程序并启动它，将可以看到 LogCat 窗口有如图 4.19 所示的输出。

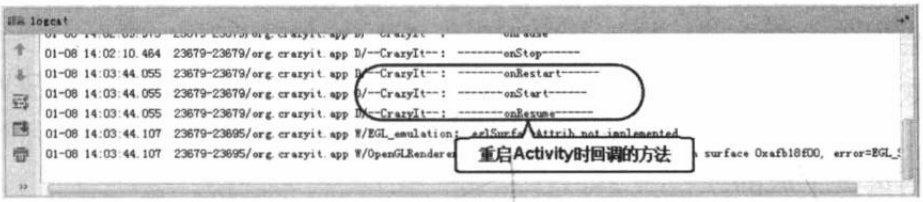


图 4.19 重新启动 Activity 时回调的方法

如果用户单击该程序界面上的“退出”按钮，该 Activity 将会结束自己，并且可以在 LogCat 窗口看到如图 4.20 所示的输出。

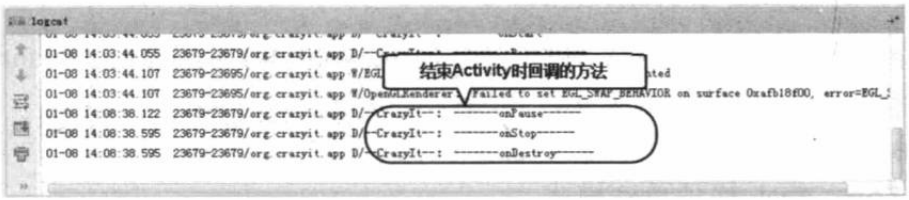


图 4.20 结束 Activity 时回调的方法

通过上面的运行过程，相信读者对 Activity 的生命周期状态及在不同状态之间切换时所回调的方法有了很清晰的认识。

4.3.2 Activity 与 Servlet 的相似性和区别

虽然很少有人会把 Activity 和 Servlet 放在一起对比，但就笔者的经验来看，Activity 与 Servlet 之间确实存在不少相似之处，如果读者已经具备一定的 Web 开发经验，通过这种“触类旁通”式的学习，相信可以更好地理解 Activity 的设计思想。

笔者一直坚信“温故而知新”是一种好的学习方式：当我们学习一门新的知识时，最好能找到新知识和已掌握知识之间的类比关系，这样既可迅速获得对新知识的直观把握，又可巩固已掌握的旧知识，避免“知识越学越多”的困扰。你的知识越积累越多，你会发现眼界越来越高，视野越来越广，看问题更容易简明、扼要地把握本质。

当然，两门知识之间除了存在相似的地方之外，也少不了差异，这部分差异正是需互相参考、互相对照的部分，以求深入理解各自的设计思想、原理。当我们学习知识时，除了掌握怎么用它之外，最好能站在设计者的角度来看：他为何要设计这个类？这个类为何要包含这些方法？方法为何要有这些形参？理解这些设计，剩下的也就简单了。

Activity 与 Servlet 的相似之处大致如下。

- Activity、Servlet 的职责都是向用户呈现界面。
- 开发者开发 Activity、Servlet 都继承系统的基类。
- Activity、Servlet 开发出来之后都需要进行配置。
- Activity 运行于 Android 应用中，Servlet 运行于 Web 应用中。
- 开发者无须创建 Activity、Servlet 的实例，无须调用它们的方法。Activity、Servlet

的方法都由系统以回调的方式来调用。

- Activity、Servlet 都有各自的生命周期，它们的生命周期都由外部负责管理。
- Activity、Servlet 都不会直接相互调用，因此都不能直接进行数据交换。Servlet 之间的数据交换需要借助于 Web 应用提供的 requestScope、sessionScope 等；Activity 之间的数据交换要借助于 Bundle。

当然，Activity 与 Servlet 之间的差别很大，因为它们本身所在场景是完全不同的，它们之间的区别也很明显。

- Activity 是 Android 窗口的容器，因此 Activity 最终以窗口的形式显示出来。而 Servlet 并不会生成应用界面，只是向浏览者生成文本响应。
- Activity 运行于 Android 应用中，因此 Activity 的本质还是通过各种界面组件来搭建界面；而 Servlet 则主要以 IO 流向浏览者生成文本响应，浏览者看到的界面其实是由浏览器负责生成的。
- Activity 之间的跳转主要通过 Intent 对象来控制；而 Servlet 之间的跳转则主要由用户请求来控制。

4.3.3 Activity 的 4 种加载模式

正如前面介绍 Activity 配置时提到的，配置 Activity 时可指定 android:launchMode 属性，该属性用于配置该 Activity 的加载模式。该属性支持如下 4 个属性值。

- standard：标准模式，这是默认的加载模式。
- singleTop：Task 栈顶单例模式。
- singleTask：Task 内单例模式。
- singleInstance：全局单例模式。

可能有读者会问：为什么要为 Activity 指定加载模式？加载模式有什么用？在讲解 Activity 的加载模式之前，先介绍 Android 对 Activity 的管理。Android 采用 Task 来管理多个 Activity，当我们启动一个应用时，Android 就会为之创建一个 Task，然后启动这个应用的入口 Activity（即<intent-filter.../>中配置为 MAIN 和 LAUNCHER 的 Activity）。

Android 的 Task 是一个有点麻烦的概念——因为 Android 并没有为 Task 提供 API，因此开发者无法真正去访问 Task，只能调用 Activity 的 getTaskId()方法来获取它所在的 Task 的 ID。事实上，我们可以把 Task 理解成 Activity 栈，Task 以栈的形式来管理 Activity：先启动的 Activity 被放在 Task 栈底，后启动的 Activity 被放在 Task 栈顶。

那么 Activity 的加载模式，就负责管理实例化、加载 Activiyt 的方式，并可以控制 Activity 与 Task 之间的加载关系。

下面详细介绍这 4 种加载模式。

1. standard 模式

每次通过这种模式来启动目标 Activity 时，Android 总会为目标 Activity 创建一个新的实例，并将该 Activity 添加到当前 Task 栈中——这种模式不会启动新的 Task，新 Activity 将被添加到原有的 Task 中。

下面的示例使用了 standard 模式来不断启动自身。

程序清单：codes\04\4.3\StandardTest\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends Activity
{
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个TextView来显示该Activity和它所在的Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity为: " + this.toString()
            + "\n" + ", Task ID为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动MainActivity");
        // 添加TextView和Button
        layout.addView(tv);
        layout.addView(button);
        // 为button添加事件监听器，当单击该按钮时启动MainActivity
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建启动MainActivity的Intent
                Intent intent = new Intent(MainActivity.this
                    , MainActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

正如上面的粗体字代码所示，每次单击按钮，程序都会再次启动 MainActivity，程序配置该 Activity 时无须指定 launchMode 属性，该 Activity 默认采用 standard 加载模式。

运行该程序，多次单击程序界面上的“启动MainActivity”按钮，程序将会不断启动新的 MainActivity 实例（不同 Activity 实例的 hashCode 值有差异），但它们所在的 Task ID 总是相同的——这表明这种加载模式不会使用全新的 Task。

standard 加载模式的示意图如图 4.21 所示。

正如图 4.21 所示，当用户单击手机的“返回”键时，系统将会“逐一”从 Activity 栈顶删除 Activity 实例。

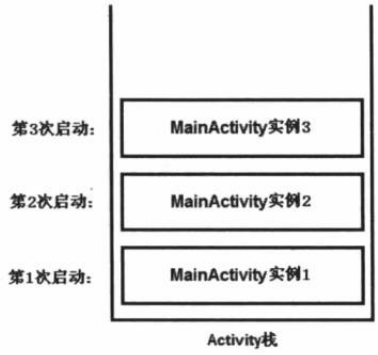


图 4.21 standard 加载模式的示意图

2. singleTop 模式

这种模式与 standard 模式基本相似，但有一点不同：当将要启动的目标 Activity 已经位于 Task 栈顶时，系统不会重新创建目标 Activity 的实例，而是直接复用已有的 Activity 实例。

如果将上面实例中 MainActivity 的加载模式改为 singleTop，那么无论用户单击多少次按钮，界面上的程序都不会有任何变化。

如果将要启动的目标 Activity 没有位于 Task 栈顶，此时系统会重新创建目标 Activity 的实例，并将它加载到 Task 栈顶——此时与 standard 模式完全相同。

3. singleTask 模式

采用这种加载模式的 Activity 在同一个 Task 内只有一个实例，当系统采用 singleTask 模式

启动目标 Activity 时，可分为如下三种情况。

- 如果将要启动的目标 Activity 不存在，系统将会创建目标 Activity 的实例，并将它加入 Task 栈顶。
- 如果将要启动的目标 Activity 已经位于 Task 栈顶，此时与 singleTop 模式的行为相同。
- 如果将要启动的目标 Activity 已经存在、但没有位于 Task 栈顶，系统将会把位于该 Activity 上面的所有 Activity 移出 Task 栈，从而使得目标 Activity 转入栈顶。

下面的示例示范了上面第三种情况。该实例包含两个 Activity，其中第一个 Activity 上显示文本框和按钮，该按钮用于启动第二个 Activity；第二个 Activity 上显示文本框和按钮，该按钮用于启动第一个 Activity。

第一个 Activity 的代码如下。

程序清单：codes\04\4.3\SingleTaskTest\app\src\main\java\org\crazyit\app\SingleTaskTest.java

```
public class SingleTaskTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在的 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为: " + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SecondActivity");
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，当单击该按钮时启动 SecondActivity
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(SingleTaskTest.this
                    , SecondActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

正如上面的粗体字代码所示，当用户单击该 Activity 上的按钮时，系统将会启动 SecondActivity。下面是 SecondActivity 的代码。

程序清单：codes\04\4.3\SingleTaskTest\app\src\main\java\org\crazyit\app\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在的 Task ID
        TextView tv = new TextView(this);
```

```
        tv.setText("Activity为: " + this.toString()
            + "\n" + ", Task ID为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动SingleTaskTest");
        // 添加TextView和Button
        layout.addView(button);
        layout.addView(tv);
        // 为button添加事件监听器，当单击该按钮时启动SingleTaskTest
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(SecondActivity.this,
                    SingleTaskTest.class);
                startActivity(intent);
            }
        });
    }
}
```

正如上面的粗体字代码所示，当用户单击该Activity上的按钮时，系统将会启动SingleTaskTest。程序将该SecondActivity的加载模式配置成singleTask。运行该示例，系统默认启动SingleTaskTest Activity，单击该界面上的按钮，系统将以singleTask模式打开SecondActivity，如图4.22所示。

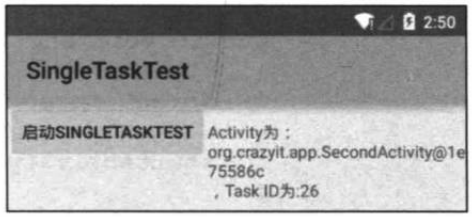


图4.22 singleTask模式

图4.22所示界面的Task栈中目前有两个Activity（从底向上）：SingleTaskTest→SecondActivity。

单击图4.22所示界面上的“启动SingleTaskTest”按钮，系统以标准模式再次加载一个新的SingleTaskTest Activity。此时Task栈中有三个Activity（从底向上）：SingleTaskTest→SecondActivity→SingleTaskTest。

在SingleTaskTest的界面上再次单击按钮，系统将会以singleTask模式再次打开SecondActivity，系统会将位于SecondActivity上面的所有Activity移出，使得SecondActivity转入栈顶。此时Task栈中只有两个Activity（从底向上）：SingleTaskTest→SecondActivity，也就是再次恢复到图4.22所示的状态。

4. singleInstance模式

在这种加载模式下，系统保证无论从哪个Task中启动目标Activity，只会创建一个目标Activity实例，并会使用一个全新的Task栈来加载该Activity实例。

当系统采用singleInstance模式启动目标Activity时，可分为如下两种情况。

- 如果将要启动的目标Activity不存在，系统会先创建一个全新的Task，再创建目标Activity的实例，并将它加入新的Task栈顶。
- 如果将要启动的目标Activity已经存在，无论它位于哪个应用程序中、位于哪个Task中，系统都会把该Activity所在的Task转到前台，从而使该Activity显示出来。

需要指出的是，采用singleInstance模式加载Activity总是位于Task栈顶，且采用singleInstance模式加载的Activity所在Task将只包含该Activity。

下面示例的SingleInstanceTest中包含一个按钮，当用户单击该按钮时，系统启动SecondActivity。程序清单如下。

程序清单：codes\04\4.3\SingleInstanceTest\app\src\main\java\org\crazyit\app\SingleInstanceTest.java

```
public class SingleInstanceTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在的 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为：" + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SecondActivity");
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，当单击该按钮时启动 SecondActivity
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(SingleInstanceTest.this
                    , SecondActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

上面的粗体字代码指定单击按钮时将会启动 SecondActivity，将该 SecondActivity 配置成 singleInstance 加载模式，并且将该 Activity 的 exported 属性配置成 true——表明该 Activity 可被其他应用启动。

配置该 Activity 的配置片段如下：

```
<activity android:name=".SecondActivity"
    android:label="@string/second"
    android:exported="true"
    android:launchMode="singleInstance">
    <intent-filter>
        <!-- 指定该 Activity 能响应 Action 为指定字符串的 Intent -->
        <action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

配置该 Activity 时，将它的 exported 属性设为 true，表明允许通过其他程序来启动该 Activity；配置该 Activity 时还配置了<intent-filter.../>子元素，这表明该 Activity 可通过隐式 Intent 启动——关于隐式 Intent 的介绍，请参考下一章的内容。

图 4.23 singleInstance 加载模式

运行该示例，系统默认显示 SingleInstanceTest，当用户单击该 Activity 界面上的按钮时，系统将会采用 singleInstance 模式加载 SecondActivity：系统启动新的 Task，并用新的 Task 加载新创建的 SecondActivity 实例，SecondActivity 总是位于该新 Task 栈顶。此时可看到如图 4.23 所示的界面。

另一个示例将采用隐式 Intent 再次启动该 SecondActivity。下面是采用隐式 Intent 启动 SecondActivity 的示例代码。

程序清单：codes\04\4.3\OtherTest\app\src\main\java\org\crazyit\other\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在的 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为: " + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SecondActivity");
        // 添加 TextView 和 Button
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，使用隐式 Intent 启动目标 Activity
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 使用隐式 Intent 启动 SecondActivity
                Intent intent = new Intent();
                intent.setAction("org.crazyit.intent.action.CRAZYIT_ACTION");
                startActivity(intent);
            }
        });
    }
}
```

运行该示例，系统默认显示 OtherTest，当用户单击该 Activity 界面上的按钮时，系统将采用隐式 Intent 来启动 SecondActivity。注意 SecondActivity 的加载模式是 singleInstance，如果前一个示例还未退出，无论 SecondActivity 所在 Task 是否位于前台，系统都将再次把 SecondActivity 所在的 Task 转入前台，从而将 SecondActivity 显示出来。也就是说，系统将会再次显示如图 4.23 所示的界面。

4.4 Fragment 详解

第 2 章介绍 ActionBar 时已经用到了 Fragment，Fragment 是 Android 3.0 引入的新 API。Fragment 代表了 Activity 的子模块，因此可以把 Fragment 理解成 Activity 片段（Fragment 本身就是片段的意思）。Fragment 拥有自己的生命周期，也可以接受它自己的输入事件。

4.4.1 Fragment 概述及其设计初衷

Fragment 必须被“嵌入”Activity 中使用，因此，虽然 Fragment 也拥有自己的生命周期，但 Fragment 的生命周期会受它所在的 Activity 的生命周期控制。例如，当 Activity 暂停时，该 Activity 内的所有 Fragment 都会暂停；当 Activity 被销毁时，该 Activity 内的所有 Fragment 都会被销毁。只有当该 Activity 处于活动状态时，程序员才可通过方法独立地操作 Fragment。

关于 Fragment，可以归纳出如下几个特征。

- Fragment 总是作为 Activity 界面的组成部分。Fragment 可调用 getActivity()方法获取它所在的 Activity，Activity 可调用 FragmentManager 的 findFragmentById()或 findFragmentByTag()方法来获取 Fragment。
- 在 Activity 运行过程中，可调用 FragmentManager 的 add()、remove()、replace()方法动态地添加、删除或替换 Fragment。
- 一个 Activity 可以同时组合多个 Fragment；反过来，一个 Fragment 也可被多个 Activity 复用。
- Fragment 可以响应自己的输入事件，并拥有自己的生命周期，但它们的生命周期直接被其所属的 Activity 的生命周期控制。

Android 3.0 引入 Fragment 的初衷是为了适应大屏幕的平板电脑，由于平板电脑的屏幕比手机屏幕更大，因此可以容纳更多的 UI 组件，且这些 UI 组件之间存在交互关系。Fragment 简化了大屏幕 UI 的设计，它不需要开发者管理组件包含关系的复杂变化，开发者使用 Fragment 对 UI 组件进行分组、模块化管理，就可以更方便地在运行过程中动态更新 Activity 的用户界面。

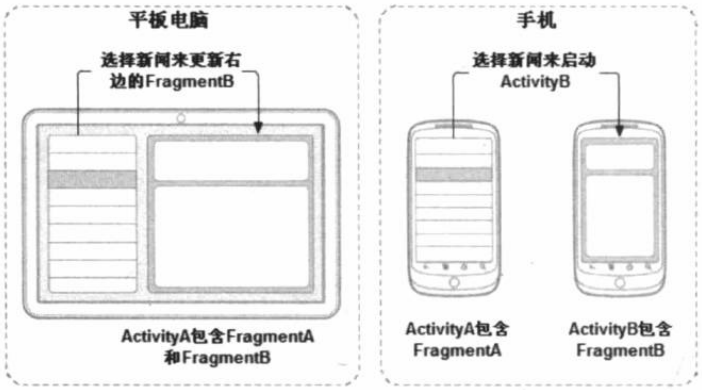


图 4.24　Fragment 的设计初衷

例如有如图 4.24 所示的新闻浏览界面，该界面需要在屏幕左边显示新闻列表，并在屏幕右边显示新闻内容，此时就可以在 Activity 中显示两个并排的 Fragment：左边的 Fragment 显示新闻列表，右边的 Fragment 显示新闻内容。由于每个 Fragment 都拥有自己的生命周期，并可响应用户输入事件，因此可以非常方便地实现：当用户单击左边列表中的指定新闻时，右边的 Fragment 就会显示相应的新闻内容。图 4.24 所示左边的“平板电脑”部分显示了这种 UI 界面。

通过使用上面的 Fragment 设计机制，可以取代传统的让一个 Activity 显示新闻列表，另一个 Activity 显示新闻内容的设计。

由于 Fragment 是可复用的组件，因此如果需要在正常尺寸的手机屏幕上运行该应用，则可以改为使用两个 Activity：ActivityA 包含 FragmentA、ActivityB 包含 FragmentB。其中 ActivityA 仅包含显示文章列表的 FragmentA，而当用户选择一篇文章时，它会启动包含新闻内容的 ActivityB，如图 4.24 所示右边的“手机”部分。由此可见，Fragment 可以很好地支持图 4.24 所示的两种设计模式。

4.4.2　创建 Fragment

与创建 Activity 类似，开发者实现的 Fragment 必须继承 Fragment 基类，Android 提供了如图 4.25 所示的 Fragment 继承体系。

开发者实现的 Fragment，可以根据需要继承图 4.25 所示的 Fragment 基类或它的任意子类。接下来，实现 Fragment 与实现 Activity

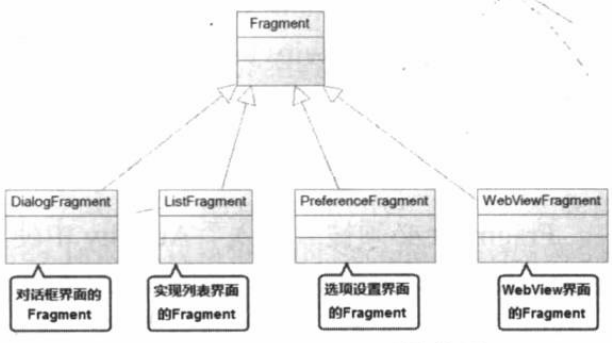


图 4.25　Fragment 继承体系

非常相似，它们都需要实现与 Activity 类似的回调方法，例如 onCreate()、onCreateView()、onStart()、onResume()、onPause()、onStop()等。

提示：

开发 Fragment 与开发 Activity 非常相似，区别只是开发 Activity 需要继承 Activity 或其子类；而开发 Fragment 需要继承 Fragment 及其子类。与此同时，只要将原来写在 Activity 回调方法中的代码“移到”Fragment 的回调方法中即可。

通常来说，创建 Fragment 需要实现如下三个方法。

- onCreate()：系统创建 Fragment 对象后回调该方法，在实现代码中只初始化想要在 Fragment 中保持的必要组件，当 Fragment 被暂停或者停止后可以恢复。
- onCreateView()：当 Fragment 绘制界面组件时会回调该方法。该方法必须返回一个 View，该 View 也就是该 Fragment 所显示的 View。
- onPause()：当用户离开该 Fragment 时将会回调该方法。

对于大部分 Fragment 而言，通常都会重写上面这三个方法。但是实际上开发者可以根据需要重写 Fragment 的任意回调方法，后面将会详细介绍 Fragment 的生命周期及其回调方法。

为了控制 Fragment 显示的组件，通常需要重写 onCreateView()方法，该方法返回的 View 将作为该 Fragment 显示的 View 组件。当 Fragment 绘制界面组件时将会回调该方法。

例如如下方法片段：

```
// 重写该方法，该方法返回的 View 将作为 Fragment 显示的组件
@Override
public View onCreateView(LayoutInflater inflater
    , ViewGroup container, Bundle savedInstanceState)
{
    // 加载/res/layout/目录下的 fragment_book_detail.xml 布局文件
    View rootView = inflater.inflate(R.layout.fragment_book_detail,
        container, false);
    if (book != null)
    {
        // 让 book_title 文本框显示 book 对象的 title 属性
        ((TextView) rootView.findViewById(R.id.book_title))
                .setText(book.title);
        // 让 book_desc 文本框显示 book 对象的 desc 属性
        ((TextView) rootView.findViewById(R.id.book_desc))
            .setText(book.desc);
    }
    return rootView;
}
```

上面方法中的第一行粗体字代码使用 LayoutInflater 加载了/res/layout/目录下的 fragment_book_detail.xml 布局文件。最后一行粗体字代码返回该布局文件对应的 View 组件，这表明该 Fragment 将会显示该 View 组件。

实例：开发显示图书详情的 Fragment

下面 Fragment 将会加载并显示一份简单的界面布局文件，并根据传入的参数来更新界面组件。该 Fragment 的代码如下。

程序清单：codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\BookDetailFragment.java

```
public class BookDetailFragment extends Fragment
{
    public static final String ITEM_ID = "item_id";
```

```
    // 保存该 Fragment 显示的 Book 对象
    BookContent.Book book;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 如果启动该 Fragment 时包含了 ITEM_ID 参数
        if (getArguments().containsKey(ITEM_ID))
        {
            book = BookContent.ITEM_MAP.get(getArguments()
                .getInt(ITEM_ID)); // ①
        }
    }
    // 重写该方法，该方法返回的 View 将作为 Fragment 显示的组件
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        // 加载/res/layout/目录下的 fragment_book_detail.xml 布局文件
        View rootView = inflater.inflate(R.layout.fragment_book_detail,
                container, false);
        if (book != null)
        {
            // 让 book_title 文本框显示 book 对象的 title 属性
            ((TextView) rootView.findViewById(R.id.book_title))
                    .setText(book.title);
            // 让 book_desc 文本框显示 book 对象的 desc 属性
            ((TextView) rootView.findViewById(R.id.book_desc))
                .setText(book.desc);
        }
        return rootView;
    }
}
```

上面的Fragment将会加载并显示res/layout/目录下的fragment_book_detail.xml界面布局文件。上面①号代码获取启动该Fragment时传入的ITEM_ID参数，并根据该ID获取BookContent的ITEM_MAP中的图书信息。

BookContent类用于模拟系统的数据模型，该模拟类的代码如下。

程序清单：codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\model\BookContent.java

```
public class BookContent
{
    // 定义一个内部类，作为系统的业务对象
    public static class Book
    {
        public Integer id;
        public String title;
        public String desc;
        public Book(Integer id, String title, String desc)
        {
            this.id = id;
            this.title = title;
            this.desc = desc;
        }
        @Override
        public String toString()
        {
            return title;
        }
    }
    // 使用 List 集合记录系统所包含的 Book 对象
```

```
    public static List<Book> ITEMS = new ArrayList<Book>();
    // 使用 Map 集合记录系统所包含的 Book 对象
    public static Map<Integer, Book> ITEM_MAP
        = new HashMap<Integer, Book>();
    static
    {
        // 使用静态初始化代码，将 Book 对象添加到 List 集合、Map 集合中
        addItem(new Book(1, "疯狂 Java 讲义"
            , "一本全面、深入的 Java 学习图书，已被多家高校选做教材。"));
        addItem(new Book(2, "疯狂 Android 讲义"
            , "Android 学习者的首选图书，常年占据京东、当当、 "
            + "亚马逊 3 大网站 Android 销量排行榜的榜首"));
        addItem(new Book(3, "轻量级 Java EE 企业应用实战"
            , "全面介绍 Java EE 开发的 Struts 2、Spring 3、Hibernate 4 框架"));
    }
    private static void addItem(Book book)
    {
        ITEMS.add(book);
        ITEM_MAP.put(book.id, book);
    }
}
```

BookDetailFragment 只是加载并显示一份简单的布局文件，这份布局文件中通过 LinearLayout 包含两个文本框。该布局文件的代码如下。

程序清单：codes\04\4.4\FragmentTest\app\src\main\res\layout\fragment_book_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
<!-- 定义一个 TextView 来显示图书标题 -->
<TextView
    style="?android:attr/textAppearanceLarge"
    android:id="@+id/book_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp"/>
<!-- 定义一个 TextView 来显示图书描述 -->
<TextView
    style="?android:attr/textAppearanceMedium"
    android:id="@+id/book_desc"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"/>
</LinearLayout>
```

实例：创建 ListFragment

如果开发 ListFragment 的子类，则无须重写 onCreateView()方法——与 ListActivity 类似的是，只要调用 ListFragment 的 setAdapter()方法为该 Fragment 设置 Adapter 即可。该 ListFragment 将会显示该 Adapter 提供的列表项。

本实例开发了一个 ListFragment 的子类。

程序清单：codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\BookListFragment.java

```
public class BookListFragment extends ListFragment
{
    private Callbacks mCallbacks;
    // 定义一个回调接口，该 Fragment 所在 Activity 需要实现该接口
```

```
    // 该 Fragment 将通过该接口与它所在的 Activity 交互
    public interface Callbacks
    {
        public void onItemSelected(Integer id);
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 为该 ListFragment 设置 Adapter
        setListAdapter(new ArrayAdapter<BookContent.Book>(getActivity(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1, BookContent.ITEMS)); //①
    }
    // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 如果 Activity 没有实现 Callbacks 接口，抛出异常
        if (!(activity instanceof Callbacks))
        {
            throw new IllegalStateException(
                "BookListFragment 所在的 Activity 必须实现 Callbacks 接口!");
        }
        // 把该 Activity 当成 Callbacks 对象
        mCallbacks = (Callbacks)activity;
    }
    // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
    @Override
    public void onDetach()
    {
        super.onDetach();
        // 将 mCallbacks 赋为 null。
        mCallbacks = null;
    }
    // 当用户单击某列表项时激发该回调方法
    @Override
    public void onListItemClick(ListView listView
        , View view, int position, long id)
    {
        super.onListItemClick(listView, view, position, id);
        // 激发 mCallbacks 的 onItemSelected 方法
        mCallbacks.onItemSelected(BookContent
            .ITEMS.get(position).id);
    }
    public void setActivateOnItemClick(boolean activateOnItemClick)
    {
        getListView().setChoiceMode(
            activateOnItemClick ? ListView.CHOICE_MODE_SINGLE
                : ListView.CHOICE_MODE_NONE);
    }
}
```

为了控制 ListFragment 显示的列表项，只要调用 ListFragment 提供的 setAdapter()方法，即可让该 ListFragment 显示该 Adapter 所提供的多个列表项。

➤➤ 4.4.3 Fragment 与 Activity 通信

为了在 Activity 中显示 Fragment，还必须将 Fragment 添加到 Activity 中。将 Fragment 添加到 Activity 中有如下两种方式。

- 在布局文件中使用<fragment.../>元素添加 Fragment，<fragment.../>元素的 android:name 属性指定 Fragment 的实现类。
- 在 Java 代码中通过 FragmentTransaction 对象的 add()方法来添加 Fragment。

Activity 的 getFragmentManager()方法可返回 FragmentManager，FragmentManager 对象的 beginTransaction()方法即可开启并返回 FragmentTransaction 对象。

下面 Activity 首先通过如下布局文件来使用前面定义的 BookListFragment。

程序清单：codes\04\4.4\FragmentTest\app\src\main\res\layout\activity_book_twopane.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个水平排列的 LinearLayout，并指定使用中等分隔条 -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:divider="?android:attr/dividerHorizontal"
    android:showDividers="middle">
    <!-- 添加一个 Fragment -->
    <fragment
        android:name="org.crazyit.app.BookListFragment"
        android:id="@+id/book_list"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />
    <!-- 添加一个 FrameLayout 容器 -->
    <FrameLayout
        android:id="@+id/book_detail_container"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />
</LinearLayout>
```

上面布局文件中的粗体字代码使用<fragment.../>元素添加了 BookListFragment，该 Activity 的左边将会显示一个 ListFragment，右边只是一个 FrameLayout 容器，该 FrameLayout 容器将会动态更新其中显示的 Fragment。下面是该 Activity 的代码。

程序清单：codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\MainActivity.java

```
public class MainActivity extends Activity implements
    BookListFragment.Callbacks
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 加载/res/layout 目录下的 activity_book_twopane.xml 布局文件
        setContentView(R.layout.activity_book_twopane);
    }
    // 实现 Callbacks 接口必须实现的方法
    @Override
    public void onItemSelected(Integer id)
    {
        // 创建 Bundle，准备向 Fragment 传入参数
        Bundle arguments = new Bundle();
        arguments.putInt(BookDetailFragment.ITEM_ID, id);
        // 创建 BookDetailFragment 对象
        BookDetailFragment fragment = new BookDetailFragment();
```

```
            // 向 Fragment 传入参数
            fragment.setArguments(arguments);
            // 使用 fragment 替换 book_detail_container 容器当前显示的 Fragment
            getFragmentManager().beginTransaction()
                .replace(R.id.book_detail_container, fragment)
                .commit(); // ①
        }
    }
```

上面程序中的①号粗体字代码就调用了 FragmentTransaction 的 replace()方法动态更新了 ID 为 book_detail_container 容器（也就是前面布局文件中的 FrameLayout 容器）中显示的 Fragment。

将 Fragment 添加到 Activity 之后，Fragment 必须与 Activity 交互信息，这就需要 Fragment 能获取它所在的 Activity，Activity 也能获取它所包含的任意的 Fragment。可按如下方式进行。

- **Fragment 获取它所在的 Activity**：调用 Fragment 的 getActivity()方法即可返回它所在的 Activity。
- **Activity 获取它包含的 Fragment**：调用 Activity 关联的 FragmentManager 的 findFragmentById(int id)或 findFragmentByTag(String tag)方法即可获取指定的 Fragment。

提示：

在界面布局文件中使用<fragment.../>元素添加 Fragment 时，可以为<fragment.../>元素指定 android:id 或 android:tag 属性，这两个属性都可用于标识该 Fragment，接下来 Activity 将可通过 findFragmentById(int id)或 findFragmentByTag (String tag)来获取该 Fragment。

除此之外，Fragment 与 Activity 可能还需要相互传递数据，可按如下方式进行。

- **Activity 向 Fragment 传递数据**：在 Activity 中创建 Bundle 数据包，并调用 Fragment 的 setArguments(Bundle bundle)方法即可将 Bundle 数据包传给 Fragment。
- **Fragment 向 Activity 传递数据或 Activity 需要在 Fragment 运行中进行实时通信**：在 Fragment 中定义一个内部回调接口，再让包含该 Fragment 的 Activity 实现该回调接口，这样 Fragment 即可调用该回调方法将数据传给 Activity。

上面示例定义了两个 Fragment，并使用一个 Activity 来“组合”这两个 Fragment，该示例的运行界面正是实现图 4.24 中示意界面的左边部分。使用大屏幕设备运行该程序，可以看到如图 4.26 所示的界面。

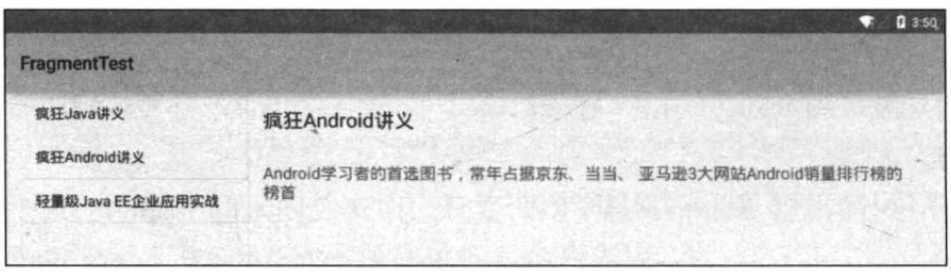


图 4.26　组合 Fragment 实现 UI 界面

4.4.4　Fragment 管理与 Fragment 事务

前面介绍了 Activity 与 Fragment 交互相关的内容，其实 Activity 管理 Fragment 主要依靠 FragmentManager。

FragmentManager 可以完成如下几方面的功能。

- 使用 findFragmentById()或 findFragmentByTag()方法来获取指定 Fragment。
- 调用 popBackStack()方法将 Fragment 从后台栈中弹出（模拟用户按下 BACK 按键）。
- 调用 addOnBackStackChangeListener()注册一个监听器，用于监听后台栈的变化。

如果需要添加、删除、替换 Fragment，则需要借助于 FragmentTransaction 对象，FragmentTransaction 代表 Activity 对 Fragment 执行的多个改变。

提示：

FragmentTransaction 也被翻译为 Fragment 事务。与数据库事务类似的是，数据库事务代表了对底层数组的多个更新操作；而 Fragment 事务则代表了 Activity 对 Fragment 执行的多个改变操作。

开发者可通过 FragmentManager 来获得 FragmentTransaction，代码片段如下：

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

每个 FragmentTransaction 可以包含多个对 Fragment 的修改，比如包含调用了多个 add()、remove()、和 replace()操作，最后调用 commit()方法提交事务即可。

在调用 commit()之前，开发者也可调用 addToBackStack()将事务添加到 Back 栈，该栈由 Activity 负责管理，这样允许用户按 BACK 按键返回到前一个 Fragment 状态。

```
// 创建一个新的 Fragment 并打开事务
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();
// 替换该界面中 fragment_container 容器内的 Fragment
transaction.replace(R.id.fragment_container, newFragment);
// 将事务添加到 Back 栈，允许用户按 BACK 按键返回到替换 Fragment 之前的状态
transaction.addToBackStack(null);
// 提交事务
transaction.commit();
```

在上面的示例代码中，newFragment 替换了当前界面布局中 ID 为 fragment_container 的容器内的 Fragment，由于程序调用 addToBackStack()将该 replace 操作添加到了 Back 栈中，因此用户可以通过按 BACK 按键返回替换之前的状态。

前面介绍的示例在大屏幕的平板电脑上运行良好，但在小屏幕的手机上运行就会有问题：运行界面非常丑陋。图 4.27 显示了在小屏幕手机上的运行效果。

图 4.27 小屏幕手机上运行组合 Fragment 实现的 UI 界面

如果希望开发的应用既可在大屏幕平板电脑上使用，也可在小屏幕手机上使用，可以考虑开发兼顾屏幕分辨率的应用。

实例：开发兼顾屏幕分辨率的应用

为了开发兼顾屏幕分辨率的应用，可以考虑在 res/目录下为大屏幕、600dpi 的屏幕建立相应的资源文件夹：values-large、values-sw600dp，在该文件夹下建立一个名为 refs.xml 的引用资源文件。该引用资源文件专门用于定义各种引用项。

本实例的引用资源文件中只有一项，下面是该引用资源文件的代码。

程序清单：codes\04\4.4\SeniorFragmentTest\app\src\main\res\values-large\refs.xml

```
<resources>
    <!-- 定义 activity_book_list 实际引用@layout/activity_book_twopane 资源 -->
    <item type="layout" name="activity_book_list">
        @layout/activity_book_twopane</item>
</resources>
```

上面引用资源文件中指定activity_book_list引用/res/layout/目录下的activity_book_twopane.xml界面布局文件。

接下来在 Activity 加载 R.layout.activity_book_list 时将会根据运行平台的屏幕大小自动选择界面布局文件：在大屏幕的平板电脑上，R.layout.activity_book_list 将会变成/res/layout/目录下的 activity_book_twopane 界面布局文件；在小屏幕的手机上，R.layout.activity_book_list 依然引用/res/layout/目录下的 activity_book_list.xml 界面布局文件。

下面是 activity_book_list.xml 界面布局文件的代码。

程序清单：codes\04\4.4\SeniorFragmentTest\app\src\main\res\layout\activity_book_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 直接使用 BookListFragment 作为界面组件 -->
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="org.crazyit.app.BookListFragment"
    android:id="@+id/book_list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"/>
```

从上面的布局文件可以看出，该布局文件仅仅显示 BookListFragment 组件，这表明该界面布局文件中只是显示图书列表。

接下来加载该布局文件的 Activity 将会针对不同的屏幕分辨率分别进行处理。下面是该 Activity 的代码。

程序清单：codes\04\4.4\SeniorFragmentTest\app\src\main\java\org\crazyit\app\BookListActivity.java

```
public class BookListActivity extends Activity implements
    BookListFragment.Callbacks
{
    // 定义一个旗标，用于标识该应用是否支持大屏幕
    private boolean mTwoPane;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 指定加载 R.layout.activity_book_list 对应的界面布局文件
        // 但实际上该应用会根据屏幕分辨率加载不同的界面布局文件
        setContentView(R.layout.activity_book_list);
        // 如果加载的界面布局文件中包含 ID 为 book_detail_container 的组件
        if (findViewById(R.id.book_detail_container) != null)
        {
            mTwoPane = true;
            ((BookListFragment) getFragmentManager()
                .findFragmentById(R.id.book_list))
                .setActivateOnItemClick(true);
        }
    }
    @Override
    public void onItemSelected(Integer id)
    {
        if (mTwoPane)
```

```
        {
            // 创建 Bundle，准备向 Fragment 传入参数
            Bundle arguments = new Bundle();
            arguments.putInt(BookDetailFragment.ITEM_ID, id);
            // 创建 BookDetailFragment 对象
            BookDetailFragment fragment = new BookDetailFragment();
            // 向 Fragment 传入参数
            fragment.setArguments(arguments);
            // 使用 fragment 替换 book_detail_container 容器当前显示的 Fragment
            getFragmentManager().beginTransaction()
                    .replace(R.id.book_detail_container, fragment).commit();
        }
        else
        {
            // 创建启动 BookDetailActivity 的 Intent
            Intent detailIntent = new Intent(this, BookDetailActivity.class);
            // 设置传给 BookDetailActivity 的参数
            detailIntent.putExtra(BookDetailFragment.ITEM_ID, id);
            // 启动 Activity
            startActivity(detailIntent);
        }
    }
}
```

由于该实例为大屏幕设备定义了引用资源文件，因此该应用将会根据大屏幕加载对应的界面布局文件，因此上面程序中第一段粗体字代码会判断界面布局中是否包含 ID 为 book_detail_container 的组件，如果包含该组件，则表明是适应大屏幕的“双屏”界面；否则，该程序界面上只包含一个简单的 BookListFragment 列表组件。此时当用户单击列表项时，程序不再是简单地更换 Fragment，而是启动 BookDetailActivity 来显示指定图书的详细信息。

BookDetailActivity 只是一个简单的封装，它将直接复用前面已有的 BookDetailFragment。BookDetailActivity 的代码如下。

程序清单：codes\04\4.4\SeniorFragmentTest\app\src\main\java\org\crazyit\app\BookDetailActivity.java

```
public class BookDetailActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 指定加载/res/layout 目录下的 activity_book_detail.xml 布局文件
        // 该界面布局文件内只定义了一个名为 book_detail_container 的 FrameLayout
        setContentView(R.layout.activity_book_detail);
        // 将 ActionBar 上的应用图标转换成可点击的按钮
        getActionBar().setDisplayHomeAsUpEnabled(true);
        if (savedInstanceState == null)
        {
            // 创建 BookDetailFragment 对象
            BookDetailFragment fragment = new BookDetailFragment();
            // 创建 Bundle 对象，
            Bundle arguments = new Bundle();
            arguments.putInt(BookDetailFragment.ITEM_ID, getIntent()
                .getIntExtra(BookDetailFragment.ITEM_ID, 0));
            // 向 Fragment 传入参数
            fragment.setArguments(arguments);
            // 将指定 fragment 添加到 book_detail_container 容器中
            getFragmentManager().beginTransaction()
                    .add(R.id.book_detail_container, fragment).commit();
        }
    }
```

```
    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        if (item.getItemId() == android.R.id.home)
        {
            // 创建启动 BookListActivity 的 Intent
            Intent intent = new Intent(this, BookListActivity.class);
            // 添加额外的 Flag，将 Activity 栈中处于 FirstActivity 之上的 Activity 弹出
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            // 启动 intent 对应的 Activity
            startActivity(intent);
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

上面的粗体字代码创建了 BookDetailFragment，并让该 Activity 显示该 Fragment 即可。

除此之外，该 Activity 还启用了 ActionBar 上的应用程序图标，允许用户点击该图标返回程序的主 Acitivity。

在大屏幕设备上运行该实例，可以看到如图 4.28 所示的界面。

单击其中一个列表项，将可以看到 4.29 所示的界面。

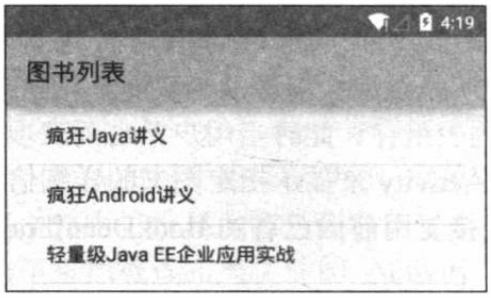


图 4.28 图书列表界面

图 4.29 图书详情界面

4.5 Fragment 的生命周期

与 Activity 类似的是，Fragment 也存在如下状态。

- **运行状态**：当前 Fragment 位于前台，用户可见，可以获得焦点。
- **暂停状态**：其他 Activity 位于前台，该 Fragment 依然可见，只是不能获得焦点。
- **停止状态**：该 Fragment 不可见，失去焦点。
- **销毁状态**：该 Fragment 被完全删除，或该 Fragment 所在的 Activity 被结束。

提示：

Android 文档只提到了 Fragment 的三个状态，官方文档没有提到 Fragment 的“销毁状态”。这也是合理的，因为处于“销毁状态”的 Fragment 基本不可用了，只能等着被回收。

图 4.30（该图参照 Android 官方文档，有些地方与文档有差异，以实际运行结果为准）显示了 Fragment 的生命周期及相关回调方法。

从图 4.30 可以看出，在 Fragment 的生命周期中，如下方法会被系统回调。

- onAttach()：当该 Fragment 被添加到 Activity 时被回调。该方法只会被调用一次。

- onCreate(Bundle savedStatus)：创建 Fragment 时被回调。该方法只会被调用一次。
- onCreateView()：每次创建、绘制该 Fragment 的 View 组件时回调该方法，Fragment 将会显示该方法返回的 View 组件。
- onActivityCreated()：当 Fragment 所在的 Activity 被启动完成后回调该方法。
- onStart()：启动 Fragment 时被回调。
- onResume()：恢复 Fragment 时被回调，在 onStart()方法后一定会回调 onResume()方法。
- onPause()：暂停 Fragment 时被回调。
- onStop()：停止 Fragment 时被回调。
- onDestroyView()：销毁该 Fragment 所包含的 View 组件时调用。
- onDestroy()：销毁 Fragment 时被回调。该方法只会被调用一次。
- onDetach()：将该 Fragment 从 Activity 中删除、替换完成时回调该方法，在 onDestroy()方法后一定会回调 onDetach()方法。该方法只会被调用一次。

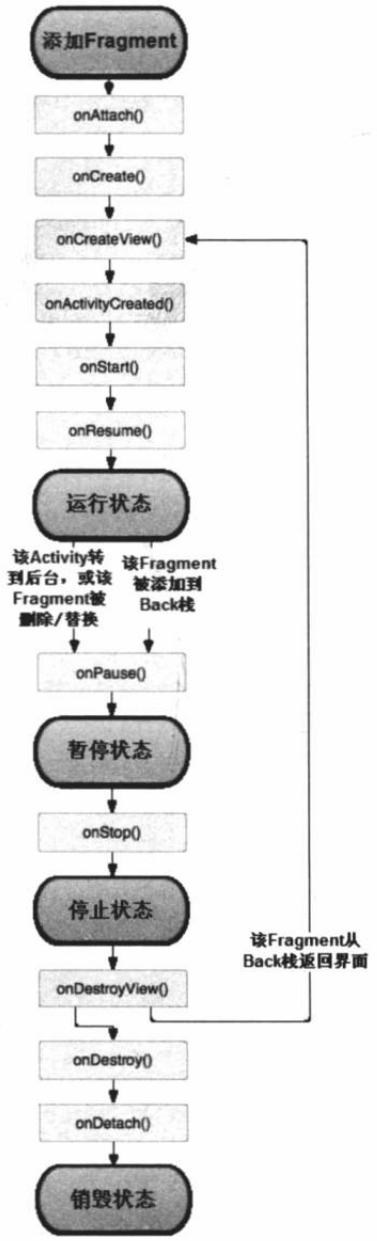


图 4.30 Fragment 的生命周期及相关回调方法

正如开发 Activity 时可以根据需要有选择性地覆盖指定方法一样，开发 Fragment 时也可根据需要有选择性地覆盖指定方法。其中最常见的就是覆盖 onCreateView()方法——该方法返回的 View 将由 Fragment 显示出来。

下面的 Fragment 覆盖了上面的 11 个生命周期方法，并在每个方法中增加了一行记录日志代码。该 Fragment 的代码如下。

程序清单：codes\04\4.5\FragmentLifecycle\app\src\main\java\org\crazyit\app\LifecycleFragment.java

```
public class LifecycleFragment extends Fragment
{
    final String TAG = "--CrazyIt--";
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 输出日志
        Log.d(TAG, "-------onAttach------");
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 输出日志
```

```
        Log.d(TAG, "-------onCreate------");
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle data)
    {
        // 输出日志
        Log.d(TAG, "-------onCreateView------");
        TextView tv = new TextView(getActivity());
        tv.setGravity(Gravity.CENTER_HORIZONTAL);
        tv.setText("测试Fragment");
        tv.setTextSize(40);
        return tv;
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState)
    {
        super.onActivityCreated(savedInstanceState);
        // 输出日志
        Log.d(TAG, "-------onActivityCreated------");
    }
    @Override
    public void onStart()
    {
        super.onStart();
        // 输出日志
        Log.d(TAG, "-------onStart------");
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 输出日志
        Log.d(TAG, "-------onResume------");
    }
    @Override
    public void onPause()
    {
        super.onPause();
        // 输出日志
        Log.d(TAG, "-------onPause------");
    }
    @Override
    public void onStop()
    {
        super.onStop();
        // 输出日志
        Log.d(TAG, "-------onStop------");
    }
    @Override
    public void onDestroyView()
    {
        super.onDestroyView();
        // 输出日志
        Log.d(TAG, "-------onDestroyView------");
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 输出日志
        Log.d(TAG, "-------onDestroy------");
    }
    @Override
```

```
    public void onDetach()
    {
        super.onDetach();
        // 输出日志
        Log.d(TAG, "-------onDetach------");
    }
}
```

包含该 Fragment 的 Activity 的布局文件很简单：该布局文件中只包含一个容器来盛装 Fragment，另外还包含几个按钮。此处不给出界面布局代码。

当使用 Activity 加载该 Fragment 时，可以在 LogCat 控制台看到如图 4.31 所示的输出。

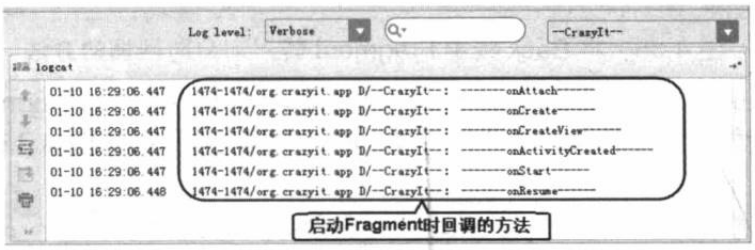


图 4.31 启动 Fragment 时回调的方法

如果单击程序中的“启动对话框风格的 Activity”按钮，将启动一个对话框风格的 Activity，当前 Activity 将会转入“暂停”状态，该 Fragment 已经进入“暂停”状态，此时可以在 LogCat 控制台看到如图 4.32 所示的输出。

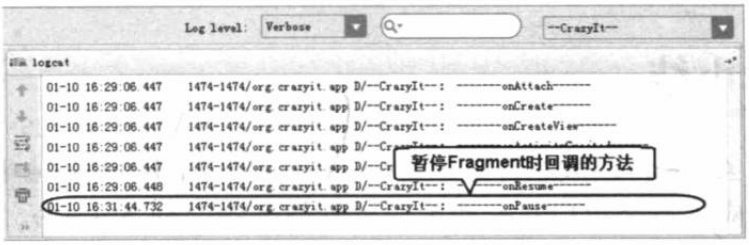


图 4.32 暂停 Fragment 时回调的方法

关闭对话框风格的 Activity，Fragment 将会再次进入“运行”状态，将可以在 LogCat 控制台看到如图 4.33 所示的输出。

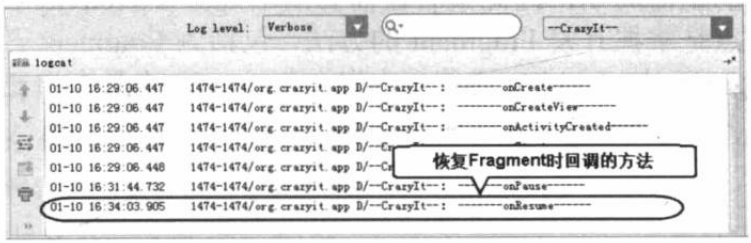


图 4.33 恢复 Fragment 时回调的方法

单击程序界面上的“替换目标 Fragment，并加入 Back 栈”，将可以在 LogCat 控制台看到如图 4.34 所示的输出。

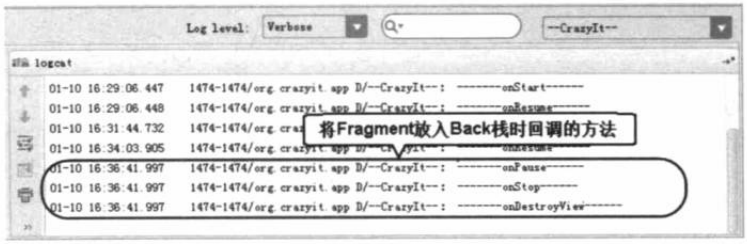


图 4.34 将 Fragment 加入 Back 栈时回调的方法

从图 4.34 可以看出，替换目标 Fragment，并将它添加到 Back 栈中，此时该 Fragment 虽然不可见，但它并未被销毁，它只是被添加到后台的 Back 栈中。当用户按下手机的 键时，该 Fragment 将会再次显示出来，此时可以在 LogCat 控制台看到如图 4.35 所示的输出。

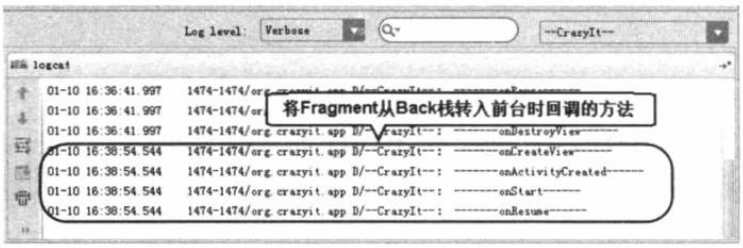


图 4.35　将 Back 栈中 Fragment 转入前台时回调的方法

单击程序界面上的“替换目标 Fragment”或“退出”按钮，该 Fragment 将会被完全结束，Fragment 将进入“销毁”状态，此时可以在 LogCat 控制台看到如图 4.36 所示的输出。

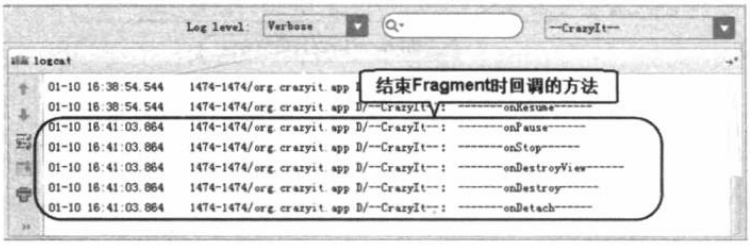


图 4.36　结束 Fragment 时回调的方法

4.6　本章小结

本章详细介绍了 Android 四大组件之一：Activity。一个 Android 应用将会包含多个 Activity，每个 Activity 通常对应于一个窗口。普通用户接触最多的就是 Activity。学习本章的重点是掌握如何开发 Activity，如何在 AndroidManifest.xml 文件中配置 Activity。不仅如此，由于 Android 系统通常会由多个 Activity 组成，因此读者还需要掌握启动其他 Activity 的方法，包括如何利用 Bundle 在不同 Activity 之间通信，启动其他 Activity 并返回结果等。Activity 在 Android 系统中运行，具有自身的生命周期，因此读者还需要清晰地掌握 Activity 的生命周期。

本章的另一个重点是掌握开发 Fragment 的方法，包括为 Fragment 开发界面、把 Fragment 添加到 Activity、Fragment 与 Activity 通信等内容。Fragment 也具有自己的生命周期，因此读者也需要清晰地掌握 Fragment 的生命周期。

CHAPTER

5

第5章 使用Intent和IntentFilter进行通信

本章要点

- 理解 Intent 对于 Android 应用的作用
- 使用 Intent 启动系统组件
- Intent 的 Component 属性的作用
- Intent 的 Action 属性的作用
- Intent 的 Category 属性的作用
- 为指定 Action、Category 的 Intent 配置对应的 intent-filter
- Intent 的 Data 属性
- Intent 的 Type 属性
- 为指定 Data、Type 的 Intent 配置对应的 intent-filter
- Intent 的 Extra 属性
- Intent 的 Flag 属性
- 使用 Intent 创建 Tab 页

前面介绍 Activity 时已经多次使用了 Intent，当一个 Activity 需要启动另一个 Activity 时，程序并没有直接告诉系统要启动哪个 Activity，而是通过 Intent 来表达自己的意图：需要启动哪个 Activity。Intent 的中文翻译就是“意图”的意思。

在这里有读者可能会产生一个疑问，假如甲 Activity 需要启动另一个 Activity，为何不直接使用一个形如 startActivity(Class activityClass)的方法呢？这样多么简单、明了啊。但实际上，这种方式虽然简洁，却明显背离了 Android 的理念，Android 使用 Intent 来封装程序的“调用意图”，不管程序想启动一个 Activity 也好，想启动一个 Service 组件也好，想启动一个 BroadcastReceiver 也好，Android 使用统一的 Intent 对象来封装这种“启动意图”，很明显使用 Intent 提供了一致的编程模型。

除此之外，使用 Intent 还有一个好处：在某些时候，应用程序只是想启动具有某种特征的组件，并不想和某个具体的组件耦合，如果使用形如 startActivity(Class activityClass)的方法来启动特定组件，势必造成一种硬编码耦合，这样也不利于高层次的解耦。

提示：

如果读者有过 Struts 2 等 MVC 框架的编程经验，一定可以很好地理解此处 Intent 的设计，当 Struts 2 的 Action 处理完用户请求之后，它并不会直接返回特定的视图页面，而是返回一个逻辑视图名，开发者可以在配置文件中把该逻辑视图映射到任意的物理视图资源；Android 系统的 Intent 设计有点类似于 Struts 2 框架中逻辑视图的设计。

总之，Intent 封装 Android 应用程序需要启动某个组件的“意图”。不仅如此，Intent 还是应用程序组件之间通信的重要媒介，正如从前面程序中所看到的，两个 Activity 可以把需要交换的数据封装成 Bundle 对象，然后使用 Intent 来携带 Bundle 对象，这样就实现了两个 Activity 之间的数据交换。

5.1 Intent 对象简述

前面介绍 Activity 时已经多次使用了 Intent，相信读者对 Intent 对象已经有了一个初步的认识，下面将对 Intent 对象进行更全面的介绍。

Android 的应用程序包含三种重要组件：Activity、Service、BroadcastReceiver，应用程序采用了一致的方式来启动它们——都是依靠 Intent 来启动的，Intent 就封装了程序想要启动程序的意图。不仅如此，Intent 还可用于与被启动组件交换信息。

表 5.1 显示了使用 Intent 启动不同组件的方法。

表 5.1 使用 Intent 启动不同组件的方法

组件类型	启动方法
Activity	startActivity(Intent intent) startActivityForResult(Intent intent, int requestCode)
Service	ComponentName startService(Intent service) boolean bindService(Intent service, ServiceConnection conn, int flags)
BroadcastReceiver	sendBroadcast(Intent intent) sendBroadcast(Intent intent, String receiverPermission) sendOrderedBroadcast(Intent intent, String receiverPermission, BroadcastReceiver resultReceiver, Handler scheduler, int initialCode, String initialData, Bundle initialExtras) sendOrderedBroadcast(Intent intent, String receiverPermission)

续表

组件类型	启动方法
BroadcastReceiver	sendStickyBroadcast(Intent intent) sendStickyOrderedBroadcast(Intent intent, BroadcastReceiver resultReceiver, Handler scheduler, int initialCode, String initialData, Bundle initialExtras)

上一章我们已经见到了如何使用 Intent 来启动 Activity 的示例，至于使用 Intent 来启动另外两种组件，本书后面章节中会有相关示例，此处暂不深入介绍。

Intent 对象大致包含 Component、Action、Category、Data、Type、Extra 和 Flag 这 7 种属性，其中 Component 用于明确指定需要启动的目标组件，而 Extra 则用于“携带”需要交换的数据。

下面详细介绍 Intent 对象各属性的作用。

5.2 Intent 的属性及 intent-filter 配置

Intent 代表了 Android 应用的启动“意图”，Android 应用将会根据 Intent 来启动指定组件，至于到底启动哪个组件，则取决于 Intent 的各属性。下面将详细介绍 Intent 的各属性值，以及 Android 如何根据不同属性值来启动相应的组件。

5.2.1 Component 属性

Intent 的 Component 属性需要接受一个 ComponentName 对象，ComponentName 对象包含如下几个构造器。

- ComponentName(String pkg, String cls)：创建 pkg 所在包下的 cls 类所对应的组件。
- ComponentName(Context pkg, String cls)：创建 pkg 所对应的包下的 cls 类所对应的组件。
- ComponentName(Context pkg, Class<?> cls)：创建 pkg 所对应的包下的 cls 类所对应的组件。

上面几个构造器的本质是相同的，这说明创建一个 ComponentName 需要指定包名和类名——这就可以唯一地确定一个组件类，这样应用程序即可根据给定的组件类去启动特定的组件。

除此之外，Intent 还包含了如下三个方法。

- setClass(Context packageContext, Class<?> cls)：设置该 Intent 将要启动的组件对应的类。
- setClassName(Context packageContext, String className)：设置该 Intent 将要启动的组件对应的类名。
- setClassName(String packageName, String className)：设置该 Intent 将要启动的组件对应的类名。

提示：

Android 应用的 Context 代表了访问该应用环境信息的接口，而 Android 应用的包名则作为应用的唯一标识，因此 Android 应用的 Context 对象与该应用的包名有一一对应的关系。上面三个 setClass()方法正是指定组件的包名（分别通过 Context 指定或 String 指定）和实现类（分别通过 Class 指定或通过 String 指定）。

指定 Component 属性的 Intent 已经明确了它将要启动哪个组件，因此这种 Intent 也被称为显式 Intent，没有指定 Component 属性的 Intent 被称为隐式 Intent——隐式 Intent 没有明确指定

要启动哪个组件，应用将会根据 Intent 指定的规则去启动符合条件的组件，但具体是哪个组件则不确定。

提示：

比如一个女孩子有找男朋友的意图，此时有两种方式来表达她的意图：第一种，她明确指出，要找“梁山伯”做男朋友，这种明确指定要找某人的方式被称为显式 Intent；第二种，她指出，要找“高”、“富”、“帅”做男朋友，至于到底是谁不重要，只要符合这三个条件即可，这种方式被称为隐式 Intent。

下面的示例程序示范了如何通过显式 Intent（指定了 Component 属性）来启动另一个 Activity。该程序的界面布局很简单，界面中只有一个按钮，用户单击该按钮将会启动第二个 Activity。此处不再给出该程序的界面布局文件。该程序的 Java 代码如下。

程序清单：codes\05\5.2\ComponentAttr\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为bn按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 创建一个ComponentName对象
                ComponentName comp = new ComponentName(MainActivity.this,
                    SecondActivity.class);
                Intent intent = new Intent();
                // 为Intent设置Component属性
                intent.setComponent(comp);
                startActivity(intent);
            }
        });
    }
}
```

上面程序中的三行粗体字代码用于创建 ComponentName 对象，并将该对象设置成 Intent 对象的 Component 属性，这样应用程序即可根据该 Intent 的“意图”去启动指定组件。

实际上，上面三行粗体字代码完全可以简化为如下形式：

```
// 根据指定组件类来创建Intent
Intent intent = new Intent(ComponentAttr.this
    , SecondActivity.class);
```

从上面的代码可以看出，当需要为 Intent 设置 Component 属性时，实际上 Intent 已经提供了一个简化的构造器，这样方便程序直接指定启动其他组件。

当程序通过 Intent 的 Component 属性（明确指定了启动哪个组件）启动特定组件时，被启动组件几乎不需要使用<intent-filter.../>元素进行配置。

程序的 SecondActivity 也很简单，它的界面布局中只包含一个简单的文本框，用于显示该 Activity 对应的 Intent 的 Component 属性的包名、类名。该 Activity 的 Java 代码如下。

程序清单：codes\05\5.2\ComponentAttr\app\src\main\java\org\crazyit\intent\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        EditText show = (EditText) findViewById(R.id.show);
        // 获取该 Activity 对应的 Intent 的 Component 属性
        ComponentName comp = getIntent().getComponent();
        // 显示该 ComponentName 对象的包名、类名
        show.setText("组件包名为：" + comp.getPackageName()
            + "\n 组件类名为：" + comp.getClassName());
    }
}
```

运行上面的程序，通过第一个 Activity 中的按钮进入第二个 Activity 中，将可以看到如图 5.1 所示的界面。

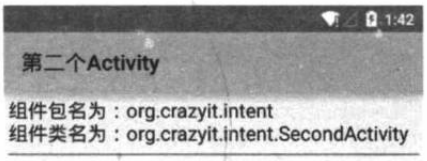


图 5.1 Intent 的 Component 属性

5.2.2 Action、Category 属性与 intent-filter 配置

Intent 的 Action、Category 属性的值都是一个普通的字符串，其中 Action 代表该 Intent 所要完成的一个抽象“动作”，而 Category 则用于为 Action 增加额外的附加类别信息。通常 Action 属性会与 Category 属性结合使用。

Action 要完成的只是一个抽象动作，这个动作具体由哪个组件（或许是 Activity，或许是 BroadcastReceiver）来完成，Action 这个字符串本身并不管。比如 Android 提供的标准 Action：Intent.ACTION_VIEW，它只表示一个抽象的查看操作，但具体查看什么、启动哪个 Activity 来查看，Intent.ACTION_VIEW 并不知道——这取决于 Activity 的<intent-filter.../>配置，只要某个 Activity 的<intent-filter.../>配置中包含了该 ACTION_VIEW，该 Actvitiy 就有可能被启动。

提示：

有过 Struts 2 开发经验的读者都知道，当 Struts 2 的 Action 处理用户请求结束后，Action 并不会直接指定“跳转”到哪个 Servlet（通常是 JSP，但 JSP 的本质就是 Servlet），Action 的处理方法只是返回一个普通字符串，然后在配置文件中配置该字符串对应到哪个 Servlet。Struts 2 之所以采用这种设计思路，就是为了把 Action 与呈现视图的 Servlet 分离开。类似的，Intent 通过指定 Action 属性（属性值其实就是一个普通字符串），就可以把该 Intent 与具体的 Activity 分离，从而提供高层次的解耦。

下面通过一个简单的示例来示范 Action 属性（就是普通字符串）的作用。下面程序的第一个 Activity 非常简单，它只包括一个普通按钮，当用户单击该按钮时，程序会“跳转”到第二个 Activity——但第一个 Activity 指定跳转的 Intent 时，并不以“硬编码”的方式指定要跳转的目标 Activity，而是为 Intent 指定 Action 属性。此处不给出界面布局的代码，第一个 Activity 的 Java 代码如下。

程序清单：codes\05\5.2\ActionAttr\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends Activity
{
```

```
    public final static String CRAZYIT_ACTION =
        "org.crazyit.intent.action.CRAZYIT_ACTION";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为bn按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 创建Intent对象
                Intent intent = new Intent();
                // 为Intent设置Action属性（属性值就是一个普通字符串）
                intent.setAction(MainActivity.CRAZYIT_ACTION);
                startActivity(intent);
            }
        });
    }
}
```

上面程序中的粗体字代码指定了根据 Intent 来启动 Activity——但该 Intent 并未以“硬编码”的方式指定要启动哪个 Activity，相信读者从上面程序中的粗体字代码无法看出该程序将要启动哪个 Activity。那么到底程序会启动哪个 Activity 呢？这取决于 Activity 配置中<intent-filter.../>元素的配置。

<intent-filter.../>元素是 AndroidManifest.xml 文件中<activity.../>元素的子元素，前面已经介绍过，<activity.../>元素用于为应用程序配置 Activity，<activity.../>的<intent-filter.../>子元素则用于配置该 Activity 所能“响应”的 Intent。

<intent-filter.../>元素里通常可包含如下子元素。

- 0～*N* 个<action.../>子元素。
- 0～*N* 个<category.../>子元素。
- 0～1 个<data.../>子元素。

提示：

<intent-filter.../>元素也可以是<service.../>、<receiver.../>两个元素的子元素，用于表明它们可以响应的 Intent。

<action.../>、<category.../>子元素的配置非常简单，它们都可指定 android:name 属性，该属性的值就是一个普通字符串。

当<activity.../>元素的<intent-filter.../>子元素里包含多个<action.../>子元素（相当于指定了多个字符串）时，就表明该 Activity 能响应 Action 属性值为其中任意一个字符串的 Intent。

提示：

还是借用前面介绍的女孩子找男朋友的例子，基本上可以把 Intent 中的 Action、Category 属性，理解为她对男朋友的要求，每个 Intent 只能指定一个 Action“要求”，但可以指定多个 Category“要求”；IntentFilter（使用<intent-filter.../>元素配置）则用于声明该组件（比如 Activity、Service、BroadcastReceiver）能满足的要求，每个组件可以声明自己满足多个 Action 要求、多个 Category 要求。只要某个组件能满足的要求大于、等于 Intent 所指定的要求，那么该 Intent 就能启动该组件。

由于上面的程序指定启动 Action 属性为 ActionAttr.CRAZYIT_ACTION 常量（常量值为 org.crazyit.intent.action.CRAZYIT_ACTION）的 Activity，也就要求被启动的 Activity 对应的配置元素的<intent-filter.../>元素里至少包括一个如下的<action.../>子元素：

```
<action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
```

需要指出的是，一个 Intent 对象最多只能包括一个 Action 属性，程序可调用 Intent 的 setAction(String str)方法来设置 Action 属性值；但一个 Intent 对象可以包括多个 Category 属性，程序可调用 Intent 的 addCategory (String str)方法来为 Intent 添加 Category 属性。当程序创建 Intent 时，该 Intent 默认启动 Category 属性值为 Intent.CATEGORY_DEFAULT 常量（常量值为 android.intent.category.DEFAULT）的组件。

因此，虽然上面程序中的粗体字代码并未指定目标 Intent 的 Category 属性，但该 Intent 已有一个值为 android.intent.category.DEFAULT 的 Category 属性值，因此被启动 Activity 对应的配置元素的<intent-filter.../>元素里至少还包括一个如下的<category.../>子元素：

```
<category android:name="android.intent.category.DEFAULT" />
```

下面是被启动的 Activity 的完整配置。

程序清单：codes\05\5.2\ActionAttr\app\src\main\AndroidManifest.xml

```
<activity android:name=".SecondActivity"
        android:label="@string/app_name">
    <intent-filter>
        <!-- 指定该Activity能响应Action属性值为指定字符串的Intent -->
        <action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
        <!-- 指定该Activity能响应Action属性值为helloWorld的Intent -->
        <action android:name="helloWorld" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

上面 Activity 配置的三行粗体字代码指定该 Activity 能响应具有指定 Action 属性值、指定 Category 属性值的 Intent。其中第二行粗体字代码只是试验用的，对于本程序没有影响——它表明该 Activity 能响应 Action 属性值为 helloWorld 字符串、Category 属性值为 android.intent.category.DEFAULT 的 Intent，但我们的程序并未尝试启动这样的 Activity，读者可以自己尝试用这样的 Intent 来启动 Activity，将会看到程序也是启动该 Activity。

上面的配置代码中配置了一个实现类为 SecondActivity 的 Activity，因此程序还应该提供这个 Activity 代码。代码如下。

程序清单：codes\05\5.2\ActionAttr\app\src\main\java\org\crazyit\intent\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        EditText show = (EditText)findViewById(R.id.show);
        // 获取该Activity对应的Intent的Action属性
        String action = getIntent().getAction();
        // 显示Action属性
        show.setText("Action为: " + action);
    }
}
```

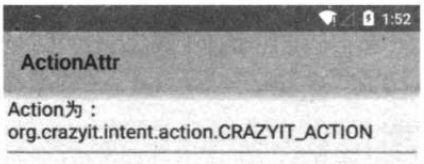


图 5.2　根据 Intent 的 Action 启动 Activity

上面的程序代码很简单，它只是在启动时把启动该 Activity 的 Intent 的 Action 属性值显示在指定文本框内。运行上面的程序，并单击程序中的“启动指定 Action、默认 Category 对应的 Activity”按钮，将看到如图 5.2 所示的界面。

接下来的示例程序将会示范 Category 属性的用法。该程序的第一个 Activity 的代码如下。

程序清单：codes\05\5.2\ActionCateAttr\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义一个 Action 常量
    final static String CRAZYIT_ACTION =
        "org.crazyit.intent.action.CRAZYIT_ACTION";
    // 定义一个 Category 常量
    final static String CRAZYIT_CATEGORY =
        "org.crazyit.intent.category.CRAZYIT_CATEGORY";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                Intent intent = new Intent();
                // 设置 Action 属性
                intent.setAction(MainActivity.CRAZYIT_ACTION);
                // 添加 Category 属性
                intent.addCategory(MainActivity.CRAZYIT_CATEGORY);
                startActivity(intent);
            }
        });
    }
}
```

上面程序中的两行粗体字代码指定了该 Intent 的 Action 属性值为 org.crazyit.intent.action.CRAZYIT_ACTION 字符串，并为该 Intent 添加了字符串为 org.crazyit.intent.category.CRAZYIT_CATEGORY 的 Category 属性。这意味着上面的程序所要启动的目标 Activity 里应该包含如下<action.../>子元素和<category.../>子元素：

```
<!-- 指定该 Activity 能响应 action 为指定字符串的 Intent -->
<action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
<!-- 指定该 Activity 能响应 category 为指定字符串的 Intent -->
<category android:name="org.crazyit.intent.category.CRAZYIT_CATEGORY" />
<!-- 指定该 Activity 能响应 category 为 android.intent.category.DEFAULT 的 Intent -->
<category android:name="android.intent.category.DEFAULT" />
```

下面是程序要启动的目标 Action 所对应的配置代码。

程序清单：codes\05\5.2\ActionCateAttr\app\src\main\AndroidManifest.xml

```
<activity android:name=".SecondActivity"
        android:label="@string/app_name">
    <intent-filter>
        <!-- 指定该 Activity 能响应 action 为指定字符串的 Intent -->
```

```
        <action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
        <!-- 指定该 Activity 能响应 category 为指定字符串的 Intent -->
        <category android:name="org.crazyit.intent.category.CRAZYIT_CATEGORY" />
        <!-- 指定该 Activity 能响应 category 为 android.intent.category.DEFAULT 的 Intent -->
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

上面配置 Activity 时也指定该 Activity 的实现类为 SecondActivity，该实现类的代码如下。

程序清单：codes\05\5.2\ActionCateAttr\app\src\main\java\org\crazyit\intent\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        EditText show = (EditText) findViewById(R.id.show);
        // 获取该 Activity 对应的 Intent 的 Action 属性
        String action = getIntent().getAction();
        // 显示 Action 属性
        show.setText("Action 为：" + action);
        EditText cate = (EditText) findViewById(R.id.cate);
        // 获取该 Activity 对应的 Intent 的 Category 属性
        Set<String> cates = getIntent().getCategories();
        // 显示 Category 属性
        cate.setText("Category 属性为：" + cates);
    }
}
```

上面的程序也很简单，它只是在启动时把启动该 Activity 的 Intent 的 Action、Catetory 属性值分别显示在不同的文本框内。运行上面的程序，并单击程序中的“启动指定 Action、指定 Category 对应的 Activity”按钮，将看到如图 5.3 所示的界面。

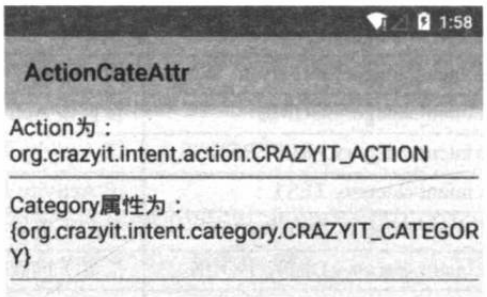


图 5.3 根据 Intent 的 Action、Category 启动 Activity

5.2.3 指定 Action、Category 调用系统 Activity

Intent 代表了启动某个程序组件的“意图”，实际上 Intent 对象不仅可以启动本应用内程序组件，也可以启动 Android 系统的其他应用的程序组件，包括系统自带的程序组件——只要权限允许。

实际上 Android 内部提供了大量标准的 Action、Catetory 常量，其中用于启动 Activity 的标准的 Action 常量及对应的字符串如表 5.2 所示。

表 5.2 启动 Activity 的标准的 Action 常量及对应的字符串

Action 常量	对应字符串	简单说明
ACTION_MAIN	android.intent.action.MAIN	应用程序入口

续表

Action 常量	对应字符串	简单说明
ACTION_VIEW	android.intent.action.VIEW	查看指定数据
ACTION_ATTACH_DATA	android.intent.action.ATTACH_DATA	指定某块数据将被附加到其他地方
ACTION_EDIT	android.intent.action.EDIT	编辑指定数据
ACTION_PICK	android.intent.action.PICK	从列表中选择某项，并返回所选的数据
ACTION_CHOOSER	android.intent.action.CHOOSER	显示一个 Activity 选择器
ACTION_GET_CONTENT	android.intent.action.GET_CONTENT	让用户选择数据，并返回所选数据
ACTION_DIAL	android.intent.action.DIAL	显示拨号面板
ACTION_CALL	android.intent.action.CALL	直接向指定用户打电话
ACTION_SEND	android.intent.action.SEND	向其他人发送数据
ACTION_SENDTO	android.intent.action.SENDTO	向其他人发送消息
ACTION_ANSWER	android.intent.action.ANSWER	应答电话
ACTION_INSERT	android.intent.action.INSERT	插入数据
ACTION_DELETE	android.intent.action.DELETE	删除数据
ACTION_RUN	android.intent.action.RUN	运行数据
ACTION_SYNC	android.intent.action.SYNC	执行数据同步
ACTION_PICK_ACTIVITY	android.intent.action.PICK_ACTIVITY	用于选择 Activity
ACTION_SEARCH	android.intent.action.SEARCH	执行搜索
ACTION_WEB_SEARCH	android.intent.action.WEB_SEARCH	执行 Web 搜索
ACTION_FACTORY_TEST	android.intent.action.FACTORY_TEST	工厂测试的入口点

标准的 Category 常量及对应的字符串如表 5.3 所示。

表 5.3 标准的 Category 常量及对应的字符串

Category 常量	对应字符串	简单说明
CATEGORY_DEFAULT	android.intent.category.DEFAULT	默认的 Category
CATEGORY_BROWSABLE	android.intent.category.BROWSABLE	指定该 Activity 能被浏览器安全调用
CATEGORY_TAB	android.intent.category.TAB	指定 Activity 作为 TabActivity 的 Tab 页
CATEGORY_LAUNCHER	android.intent.category.LAUNCHER	Activity 显示顶级程序列表
CATEGORY_INFO	android.intent.category.INFO	用于提供包信息
CATEGORY_HOME	android.intent.category.HOME	设置该 Activity 随系统启动而运行
CATEGORY_PREFERENCE	android.intent.category.PREFERENCE	该 Activity 是参数面板
CATEGORY_TEST	android.intent.category.TEST	该 Activity 是一个测试
CATEGORY_CAR_DOCK	android.intent.category.CAR_DOCK	指定手机被插入汽车底座（硬件）时运行该 Activity
CATEGORY_DESK_DOCK	android.intent.category.DESK_DOCK	指定手机被插入桌面底座（硬件）时运行该 Activity
CATEGORY_CAR_MODE	android.intent.category.CAR_MODE	设置该 Activity 可在车载环境下使用

表 5.2、表 5.3 所列出的都只是部分较为常用的 Action 常量、Category 常量。关于 Intent 所提供的全部 Action 常量、Category 常量，应参考 Android API 文档中关于 Intent 的说明。

下面将以两个实例来介绍 Intent 系统 Action、系统 Category 的用法。

实例：查看并获取联系人电话

这个实例将会在程序中提供一个按钮，用户单击该按钮时会显示系统的联系人列表，当用户单击指定联系人之后，程序将会显示该联系人的名字、电话。

这个程序非常有用，比如我们要开发一个发送短信的程序，当用户编写短信完成之后，可能需要浏览联系人列表，并从联系人列表中选出短信接收人，那就可以用到该程序了。

该程序的界面布局代码如下。

程序清单：codes\05\5.2\SysAction\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<!-- 显示联系人姓名的文本框 -->
<EditText
    android:id="@+id/show"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:editable="false"
    android:cursorVisible="false"/>
<!-- 显示联系人电话号码的文本框 -->
<EditText
    android:id="@+id/phone"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:editable="false"
    android:cursorVisible="false"/>
<Button
    android:id="@+id/bn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="查看联系人"/>
</LinearLayout>
```

上面的界面布局中包含了两个文本框、一个按钮，其中按钮用于浏览系统的联系人列表并选择其中的联系人。两个文本框分别用于显示联系人的名字和电话号码。

注意：

由于该程序会用到系统的联系人应用，因此读者在运行该程序之前应该先进入 Android 系统自带的 Contacts 程序，并通过该程序添加几个联系人。

该程序的 Java 代码如下。

程序清单：codes\05\5.2\SysAction\app\src\main\java\org\crazyit\action\MainActivity.java

```
public class MainActivity extends Activity
{
    final int PICK_CONTACT = 0;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 创建 Intent
                Intent intent = new Intent();
                // 设置 Intent 的 Action 属性
                intent.setAction(Intent.ACTION_GET_CONTENT);
```

```
            // 设置 Intent 的 Type 属性
            intent.setType("vnd.android.cursor.item/phone");
            // 启动 Activity，并希望获取该 Activity 的结果
            startActivityForResult(intent, PICK_CONTACT);
        }
    });
}
@Override
public void onActivityResult(int requestCode
    , int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode)
    {
        case (PICK_CONTACT):
            if (resultCode == Activity.RESULT_OK)
            {
                // 获取返回的数据
                Uri contactData = data.getData();
                CursorLoader cursorLoader = new CursorLoader(this
                    , contactData, null, null, null, null);
                // 查询联系人信息
                Cursor cursor = cursorLoader.loadInBackground();
                // 如果查询到指定的联系人
                if (cursor.moveToFirst())
                {
                    String contactId = cursor.getString(cursor
                        .getColumnIndex(
                        ContactsContract.Contacts._ID));
                    // 获取联系人的名字
                    String name = cursor.getString(cursor
                        .getColumnIndexOrThrow(
                        ContactsContract.Contacts.DISPLAY_NAME));
                    String phoneNumber = "此联系人暂未输入电话号码";
                    // 根据联系人查询该联系人的详细信息
                    Cursor phones = getContentResolver().query(
                        ContactsContract.CommonDataKinds.
                        Phone.CONTENT_URI, null,
                        ContactsContract.CommonDataKinds.Phone.
                                        CONTACT_ID
                            + " = " + contactId, null, null);
                    if (phones.moveToFirst())
                    {
                        // 取出电话号码
                        phoneNumber = phones
                            .getString(phones
                            .getColumnIndex(ContactsContract
                            .CommonDataKinds.Phone.NUMBER));
                    }
                    // 关闭游标
                    phones.close();
                    EditText show =(EditText)findViewById(R.id.show);
                    // 显示联系人的名称
                    show.setText(name);
                    EditText phone =(EditText)findViewById(R.id.phone);
                    // 显示联系人的电话号码
                    phone.setText(phoneNumber);
                }
                // 关闭游标
                cursor.close();
            }
```

```
                break;
            }
        }
    }
```

提示：

该实例使用了后面章节中关于 ContentProvider 的知识，如果读者一时看不懂这个程序，可以参考后面章节的讲解进行理解。

运行上面的程序，单击程序界面中的“查看联系人”按钮，程序将会显示如图 5.4 所示的界面。

在图 5.4 所示的联系人列表中单击某个联系人，系统将会自动返回上一个 Activity，程序会在上一个 Activity 中显示所选联系人的名字和电话号码，如图 5.5 所示。

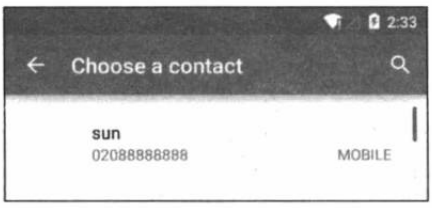


图 5.4 查看联系人

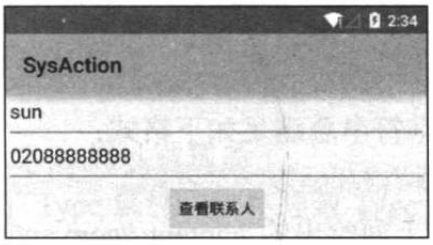


图 5.5 获取指定联系人的信息

上面的 Intent 对象除了设置 Action 属性之外，还设置了 Type 属性，关于 Intent 的 Type 属性的作用，等下将会进行更详细的介绍。

需要指出的是，由于上面的程序需要查看系统联系人信息，因此不要忘了向该应用的 AndroidManifest.xml 文件中增加相应的权限，也就是在 AndroidManifest.xml 文件中增加如下配置：

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

实例：返回系统 Home 桌面

本实例将会提供一个按钮，当用户单击该按钮时，系统将会返回 Home 桌面，就像单击模拟器右边的⌂按钮一样。这也需要通过 Intent 来实现，程序为 Intent 设置合适的 Action、Category 属性，并根据该 Intent 来启动 Activity 即可返回 Home 桌面。该实例程序如下。

程序清单：codes\05\5.2\ReturnHome\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                // 为 Intent 设置 Action、Category 属性
                intent.setAction(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
```

```
                startActivity(intent);
            }
        });
    }
}
```

上面程序中的粗体字代码设置了 Intent 的 Action 属性值为 android.intent.action.MAIN 字符串、Category 属性值为 android.intent.category.HOME 字符串，满足该 Intent 的 Activity 其实就是 Android 系统的 Home 桌面。因此，运行上面的程序时单击“返回桌面”按钮，即可返回 Home 桌面。

5.2.4 Data、Type 属性与 intent-filter 配置

Data 属性通常用于向 Action 属性提供操作的数据。Data 属性接受一个 Uri 对象，一个 Uri 对象通常通过如下形式的字符串来表示：

```
content://com.android.contacts/contacts/1
tel:123
```

Uri 字符串总满足如下格式：

```
scheme://host:port/path
```

例如上面给出的 content://com.android.contacts/contacts/1，其中 content 是 scheme 部分，com.android.contacts 是 host 部分，port 部分被省略了，/contacts/1 是 path 部分。

Type 属性用于指定该 Data 属性所指定 Uri 对应的 MIME 类型，这种 MIME 类型可以是任何自定义的 MIME 类型，只要符合 abc/xyz 格式的字符串即可。

Data 属性与 Type 属性的关系比较微妙，这两个属性会相互覆盖，例如：

- 如果为 Intent 先设置 Data 属性，后设置 Type 属性，那么 Type 属性将会覆盖 Data 属性。
- 如果为 Intent 先设置 Type 属性，后设置 Data 属性，那么 Data 属性将会覆盖 Type 属性。
- 如果希望 Intent 既有 Data 属性，也有 Type 属性，则应该调用 Intent 的 setDataAndType() 方法。

下面的示例演示了 Intent 的 Data 属性与 Type 属性互相覆盖的情形，该示例的界面布局文件很简单，只是定义了三个按钮，并为三个按钮绑定了事件处理函数。

下面是该示例的 Activity 代码。

程序清单：codes\05\5.2\DataTypeOverride\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void overrideType(View source)
    {
        Intent intent = new Intent();
        // 先为 Intent 设置 Type 属性
        intent.setType("abc/xyz");
        // 再为 Intent 设置 Data 属性，覆盖 Type 属性
        intent.setData(Uri.parse("lee://www.fkjava.org:8888/test"));
        Toast.makeText(this, intent.toString(), Toast.LENGTH_LONG).show();
    }
    public void overrideData(View source)
    {
```

```
        Intent intent = new Intent();
        // 先为 Intent 设置 Data 属性
        intent.setData(Uri.parse("lee://www.fkjava.org:8888/mypath"));
        // 再为 Intent 设置 Type 属性，覆盖 Data 属性
        intent.setType("abc/xyz");
        Toast.makeText(this, intent.toString(), Toast.LENGTH_LONG).show();
    }
    public void dataAndType(View source)
    {
        Intent intent = new Intent();
        // 同时设置 Intent 的 Data、Type 属性
        intent.setDataAndType(Uri.parse("lee://www.fkjava.org:8888/mypath"),
            "abc/xyz");
        Toast.makeText(this, intent.toString(), Toast.LENGTH_LONG).show();
    }
}
```

上面的三个事件监听方法分别为 Intent 设置了 Data、Type 属性，第一个事件监听方法先设置了 Type 属性，再设置了 Data 属性，这将导致 Data 属性覆盖 Type 属性，单击按钮激发该事件监听方法，将可以看到如图 5.6 所示的 Toast 输出。

从图 5.6 可以看出，此时的 Intent 只有 Data 属性，Type 属性被覆盖了。

第二个事件监听方法先设置了 Data 属性，再设置了 Type 属性，这将导致 Type 属性覆盖 Data 属性，单击按钮激发该事件监听方法，将可以看到如图 5.7 所示的输出。

图 5.6 Data 属性覆盖 Type 属性

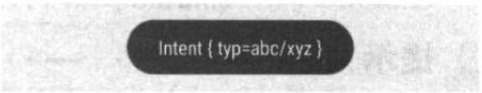


图 5.7 Type 属性覆盖 Data 属性

第三个事件监听方法同时设置了 Data、Type 属性，这样该 Intent 中才会同时具有 Data、Type 属性。

在 AndroidManifest.xml 文件中为组件声明 Data、Type 属性都通过<data.../>元素，<data.../>元素的格式如下：

```
<data android:mimeType=""
    android:scheme=""
    android:host=""
    android:port=""
    android:path=""
    android:pathPrefix=""
    android:pathPattern="" />
```

上面的<data.../>元素支持如下属性。

- mimeType：用于声明该组件所能匹配的 Intent 的 Type 属性。
- scheme：用于声明该组件所能匹配的 Intent 的 Data 属性的 scheme 部分。
- host：用于声明该组件所能匹配的 Intent 的 Data 属性的 host 部分。
- port：用于声明该组件所能匹配的 Intent 的 Data 属性的 port 部分。
- path：用于声明该组件所能匹配的 Intent 的 Data 属性的 path 部分。
- pathPrefix：用于声明该组件所能匹配的 Intent 的 Data 属性的 path 前缀。
- pathPattern：用于声明该组件所能匹配的 Intent 的 Data 属性的 path 字符串模板。

Intent 的 Type 属性也用于指定该 Intent 的要求，对应组件中<intent-filter.../>元素的<data.../>子元素的 mimeType 属性必须与此相同，才能启动该组件。

Intent 的 Data 属性则略有差异，程序员为 Intent 指定 Data 属性时，Data 属性的 Uri 对象实际上可分为 scheme、host、port 和 path 部分，此时并不要求被启动组件的<intent-filter.../>中

<data.../>子元素的 android:scheme、android:host、android:port、android:path 完全满足。

Data 属性的“匹配”过程则有些差别，它会先检查<intent-filter.../>里的<data.../>子元素，然后：

- 如果目标组件的<data.../>子元素只指定了 android:scheme 属性，那么只要 Intent 的 Data 属性的 scheme 部分与 android:scheme 属性值相同，即可启动该组件。
- 如果目标组件的<data.../>子元素只指定了 android:scheme、android:host 属性，那么只要 Intent 的 Data 属性的 scheme、host 部分与 android:scheme、android:host 属性值相同，即可启动该组件。
- 如果目标组件的<data.../>子元素指定了 android:scheme、android:host、android:port 属性，那么只要 Intent 的 Data 属性的 scheme、host、port 部分与 android:scheme、android:host、android:host 属性值相同，即可启动该组件。

提示：

如果<data.../>子元素只有 android:port 属性，没有指定 android:host 属性，那么 android:port 属性将不会起作用。

- 如果目标组件的<data.../>子元素只指定了 android:scheme、android:host、android:path 属性，那么只要 Intent 的 Data 属性的 scheme、host、path 部分与 android:scheme、android:host、android:path 属性值相同，即可启动该组件。

提示：

如果<data.../>子元素只有 android:path 属性，没有指定 android:host 属性，那么 android:path 属性将不会起作用。

- 如果目标组件的<data.../>子元素指定了 android:scheme、android:host、android:port、android:path 属性，那么就要求 Intent 的 Data 属性的 scheme、host、port、path 部分依次与 android:scheme、android:host、android:port、android:path 属性值相同，才可启动该组件。

下面的示例测试了 Intent 的 Data 属性与<data.../>元素配置的关系，该示例依次配置了如下 5 个 Activity。

程序清单：codes\05\5.2\DataTypeAttr\app\src\main\AndroidManifest.xml

```
<activity
    android:icon="@drawable/ic_scheme"
    android:name=".SchemeActivity"
    android:label="指定 scheme 的 Activity">
    <intent-filter>
        <action android:name="xx" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- 只要 Intent 的 Data 属性的 scheme 是 lee，即可启动该 Activity -->
        <data android:scheme="lee" />
    </intent-filter>
</activity>
<activity
    android:icon="@drawable/ic_host"
    android:name=".SchemeHostPortActivity"
    android:label="指定 scheme、host、port 的 Activity">
    <intent-filter>
        <action android:name="xx" />
        <category android:name="android.intent.category.DEFAULT" />
```

```
        <!-- 只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org，
        port 是 8888，即可启动该 Activity -->
        <data android:scheme="lee"
            android:host="www.fkjava.org"
            android:port="8888" />
    </intent-filter>
</activity>
<activity
    android:icon="@drawable/ic_sp"
    android:name=".SchemeHostPathActivity"
    android:label="指定 scheme、host、path 的 Activity">
    <intent-filter>
        <action android:name="xx" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- 只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org，
        path 是/mypath，即可启动该 Activity -->
        <data android:scheme="lee"
            android:host="www.fkjava.org"
            android:path="/mypath" />
    </intent-filter>
</activity>
<activity
    android:icon="@drawable/ic_path"
    android:name=".SchemeHostPortPathActivity"
    android:label="指定 scheme、host、port、path 的 Activity">
    <intent-filter>
        <action android:name="xx" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- 需要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org，
        port 是 8888，path 是/mypath，才可启动该 Activity -->
        <data android:scheme="lee"
            android:host="www.fkjava.org"
            android:port="8888"
            android:path="/mypath"/>
    </intent-filter>
</activity>
<activity
    android:icon="@drawable/ic_type"
    android:name=".SchemeHostPortPathTypeActivity"
    android:label="指定 scheme、host、port、path、type 的 Activity">
    <intent-filter>
        <action android:name="xx"/>
        <category android:name="android.intent.category.DEFAULT" />
        <!-- 需要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org，
        port 是 8888，path 是/mypath，
        type 是 abc/xyz，才可启动该 Activity -->
        <data android:scheme="lee"
            android:host="www.fkjava.org"
            android:port="8888"
            android:path="/mypath"
            android:mimeType="abc/xyz"/>
    </intent-filter>
</activity>
```

上面的配置文件中配置了 5 个 Activity，这 5 个 Activity 的实现类都非常简单，它们都仅在界面上显示一个 TextView，并不显示其他内容。关于这 5 个 Activity 的<data.../>子元素配置说明如下。

- **第 1 个 Activity**：只要 Intent 的 Data 属性的 scheme 是 lee，即可启动该 Activity。
- **第 2 个 Activity**：只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org，port 是 8888，即可启动该 Activity。

- **第3个Activity**：只要Intent的Data属性的scheme是lee，且host是www.fkjava.org，path是/mypath，即可启动该Activity。
- **第4个Activity**：需要Intent的Data属性的scheme是lee，且host是www.fkjava.org，port是8888，path是/mypath，才可启动该Activity。
- **第5个Activity**：需要Intent的Data属性的scheme是lee，且host是www.fkjava.org，port是8888，path是/mypath，type是abc/xyz，才可启动该Activity。

注意：

上面配置Activity的<intent-filter.../>元素时，<action.../>子元素的name属性是随意指定的，这是必需的。如果希望<data.../>子元素能正常起作用，至少要配置一个<action.../>子元素，但该子元素的android:name属性值可以是任意的字符串。

下面是第1个启动Activity的方法。

```
public void scheme(View source)
{
    Intent intent = new Intent();
    // 只设置Intent的Data属性
    intent.setData(Uri.parse("lee://www.crazyit.org:1234/test"));
    startActivity(intent);
}
```

上面Intent的Data属性，只有scheme为lee，也就是只有第1个Activity符合条件，因此通过该方法启动Activity时，将可以看到启动如图5.8所示的Activity。

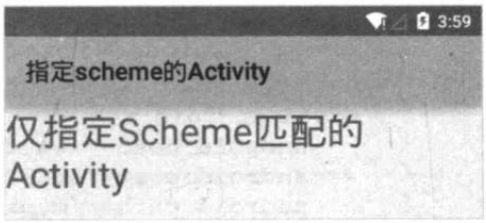


图5.8 只有scheme匹配的Activity

下面是第2个启动Activity的方法。

```
public void schemeHostPort(View source)
{
    Intent intent = new Intent();
    // 只设置Intent的Data属性
    intent.setData(Uri.parse("lee://www.fkjava.org:8888/test"));
    startActivity(intent);
}
```

上面Intent的Data属性，scheme为lee，因此第1个Activity符合条件；且该Intent的Data属性的host为www.fkjava.org，port为8888，因此第2个Activity也符合条件。通过该方法启动Activity时，将可以看到启动如图5.9所示的选择Activity界面。

下面是第3个启动Activity的方法。

```
public void schemeHostPath(View source)
{
    Intent intent = new Intent();
    // 只设置Intent的Data属性
    intent.setData(Uri.parse("lee://www.fkjava.org:1234/mypath"));
    startActivity(intent);
}
```

上面Intent的Data属性，scheme为lee，因此第1个Activity符合条件；且该Intent的Data属性的host为www.fkjava.org，path为/mypath，因此第3个Activity也符合条件。通过该方法启动Activity时，将可以看到启动如图5.10所示的选择Activity界面。

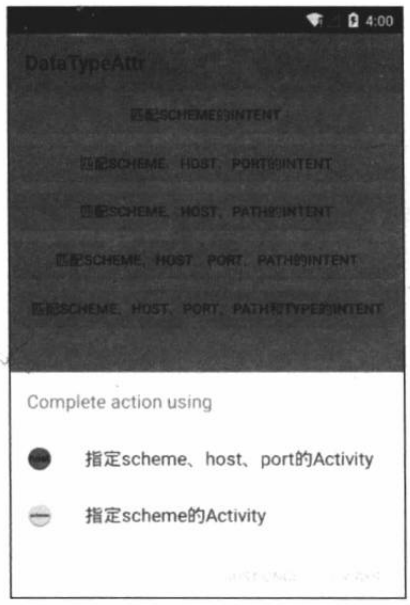


图 5.9 scheme、host、port 匹配的 Activity

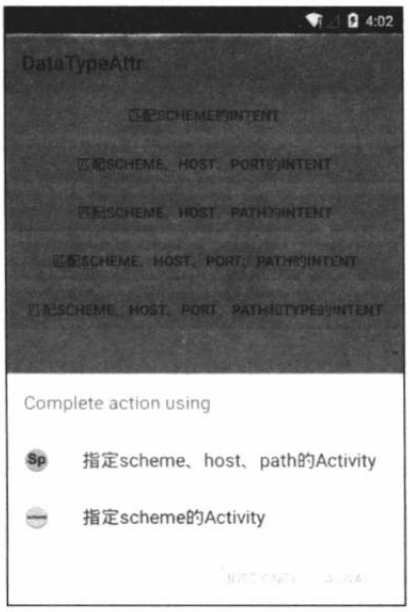


图 5.10 scheme、host、path 匹配的 Activity

下面是第 4 个启动 Activity 的方法。

```
public void schemeHostPortPath(View source)
{
    Intent intent = new Intent();
    // 只设置 Intent 的 Data 属性
    intent.setData(Uri.parse("lee://www.fkjava.org:8888/mypath"));
    startActivity(intent);
}
```

上面 Intent 的 Data 属性，scheme 为 lee，因此第 1 个 Activity 符合条件；且该 Intent 的 Data 属性的 host 为 www.fkjava.org，port 为 8888，因此第 2 个 Activity 符合条件；该 Intent 的 Data 属性的 host 为 www.fkjava.org，path 为 /mypath，因此第 3 个 Activity 符合条件；该 Intent 的 Data 属性的 host 为 www.fkjava.org，port 为 8888，path 为 /mypath，因此第 4 个 Activity 也符合条件。通过该方法启动 Activity 时，将可以看到启动如图 5.11 所示的选择 Activity 界面。

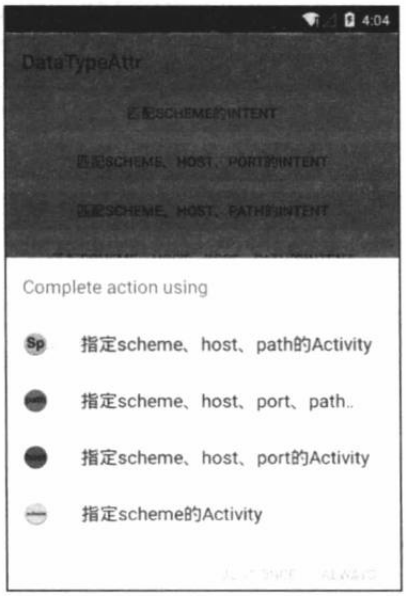


图 5.11 scheme、host、port、path 匹配的 Activity

下面是第 5 个启动 Activity 的方法。

```
public void schemeHostPortPathType(View source)
{
    Intent intent = new Intent();
    // 同时设置 Intent 的 Data、Type 属性
    intent.setDataAndType(Uri.parse("lee://www.fkjava.org:8888/mypath")
        , "abc/xyz");
    startActivity(intent);
}
```

上面的 Intent 不仅指定了 Data 属性，也指定了 Type 属性，此时符合条件的只有第 5 个

Activity。通过该方法启动 Activity 时，将可以看到启动如图 5.12 所示的 Activity。

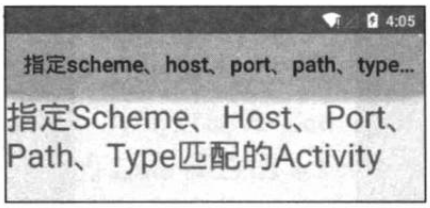


图 5.12 scheme、host、port、path、type 匹配的 Activity

实例：使用 Action、Data 属性启动系统 Activity

一旦为 Intent 同时指定了 Action、Data 属性，Android 就可根据指定的数据类型来启动特定的应用程序，并对指定数据执行相应的操作。

下面是几个 Action 属性、Data 属性的组合。

- ACTION_VIEW content://com.android.contacts/contacts/1：显示标识为 1 的联系人的信息。
- ACTION_EDIT content://com.android.contacts/contacts/1：编辑标识为 1 的联系人的信息。
- ACTION_DIAL content://com.android.contacts/contacts/1：显示向标识为 1 的联系人拨号的界面。
- ACTION_VIEW tel:123：显示向指定号码 123 拨号的界面。
- ACTION_DIAL tel:123：显示向指定号码 123 拨号的界面。
- ACTION_VIEW content://contacts/people/：显示所有联系人列表的信息，通过这种组合可以非常方便地查看系统联系人。

本实例程序示范通过同时为 Intent 指定 Action、Data 属性来启动特定程序并操作相应的数据。下面程序的界面很简单，只包含 3 个按钮，其中一个按钮用于浏览指定网页；一个按钮用于编辑指定联系人信息；另一个按钮用于呼叫指定号码。

程序清单：codes\05\5.2\ActionData\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为bn按钮添加一个监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent
                Intent intent = new Intent();
                String data = "http://www.crazyit.org";
                // 根据指定字符串解析出 Uri 对象
                Uri uri = Uri.parse(data);
                // 为 Intent 设置 Action 属性
                intent.setAction(Intent.ACTION_VIEW);
                // 设置 Data 属性
                intent.setData(uri);
                startActivity(intent);
            }
        });
```

```
        Button edit = (Button) findViewById(R.id.edit);
        // 为edit按钮添加一个监听器
        edit.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建Intent
                Intent intent = new Intent();
                // 为Intent设置Action属性（动作为：编辑）
                intent.setAction(Intent.ACTION_EDIT);
                String data = "content://com.android.contacts/contacts/1";
                // 根据指定字符串解析出Uri对象
                Uri uri = Uri.parse(data);
                // 设置Data属性
                intent.setData(uri);
                startActivity(intent);
            }
        });
        Button call = (Button) findViewById(R.id.call);
        // 为call按钮添加一个监听器
        call.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建Intent
                Intent intent = new Intent();
                // 为Intent设置Action属性（动作为：拨号）
                intent.setAction(Intent.ACTION_DIAL);
                String data = "tel:13800138000";
                // 根据指定字符串解析出Uri对象
                Uri uri = Uri.parse(data);
                // 设置Data属性
                intent.setData(uri);
                startActivity(intent);
            }
        });
    }
}
```

运行上面的程序，单击第一个按钮，该按钮单击时启动Intent（Action=Intent.ACTION_VIEW，Data=http://www.crazyit.org）对应的Activity，将看到打开www.crazyit.org的界面，如图5.13所示。

图5.13 使用Action、Data打开指定网页

单击第二个按钮，该按钮单击时启动Intent（Action=Intent.ACTION_EDIT，Data=content://com.android.contacts/ contacts/1）对应的Activity，将看到编辑标识为1的联系人界面，如图5.14所示。

单击第三个按钮，该按钮单击时启动Intent（Action=Intent.ACTION_DIAL，Data=tel:13800138000）对应的Activity，将看到程序向13800138000拨号的界面，如图5.15所示。

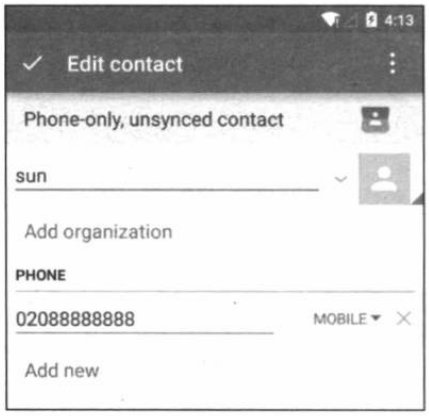


图 5.14　使用 Action、Data 属性编辑指定联系人

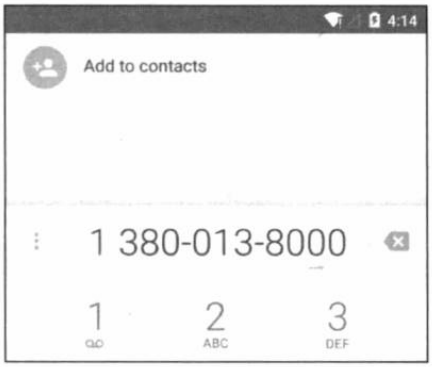


图 5.15　使用 Action、Data 属性向指定号码拨号

5.2.5　Extra 属性

Intent 的 Extra 属性通常用于在多个 Action 之间进行数据交换，Intent 的 Extra 属性值应该是一个 Bundle 对象，Bundle 对象就像一个 Map 对象，它可以存入多个 key-value 对，这样就可以通过 Intent 在不同 Activity 之间进行数据交换了。关于 Extra 属性的用法前面已有示例，此处不再赘述。

5.2.6　Flag 属性

Intent 的 Flag 属性用于为该 Intent 添加一些额外的控制旗标，Intent 可调用 addFlags()方法来添加控制旗标。

提示：

前面介绍启用 ActionBar 的程序图标返回主 Activity 时已经用到了 Flag 属性。比如前面介绍的 Intent.FLAG_ACTIVITY_CLEAR_TOP 旗标可用于清除当前 Activity 栈中的 Activity。

除此之外，Intent 还包含了如下常用的 Flag 旗标。

- FLAG_ACTIVITY_BROUGHT_TO_FRONT：如果通过该 Flag 启动的 Activity 已经存在，下次再次启动时，将只是把该 Activity 带到前台。例如，现在 Activity 栈中有 Activity A，此时以该旗标启动 Activity B（即 Activity B 是以 FLAG_ACTIVITY_BROUGHT_TO_FRONT 旗标启动的），然后在 Activity B 中启动 Activity C、D，如果此时在 Activity D 中再启动 Activity B，将直接把 Activity 栈中的 Activity B 带到前台。此时 Activity 栈中情形是 Activity A、C、D、B。
- FLAG_ACTIVITY_CLEAR_TOP：该 Flag 相当于加载模式中的 singleTask，通过这种 Flag 启动的 Activity 将会把要启动的 Activity 之上的 Activity 全部弹出 Activity 栈。例如，Activity 栈中包含 A、B、C、D 四个 Activity，如果采用该 Flag 从 Activity D 跳转到 Activity B，那么此时 Activity 栈中只包含 A、B 两个 Activity。
- FLAG_ACTIVITY_NEW_TASK：默认的启动旗标，该旗标控制重新创建一个新的 Activity。
- FLAG_ACTIVITY_NO_ANIMATION：该旗标控制启动 Activity 时不使用过渡动画。
- FLAG_ACTIVITY_NO_HISTORY：该旗标控制被启动的 Activity 将不会保留在 Activity 栈中。例如，Activity 栈中原来有 A、B、C 三个 Activity，此时在 Activity C 中以该 Flag 启动 Activity D，Activity D 再启动 Activity E，此时 Activity 栈中只有 A、B、C、E 四

个 Activity，Activity D 不会保留在 Actvity 栈中。

- FLAG_ACTIVITY_REORDER_TO_FRONT：该 Flag 控制如果当前已有 Activity，则直接将该 Activity 带到前台。例如，现在 Activity 栈中有 A、B、C、D 四个 Activity，如果使用 FLAG_ACTIVITY_REORDER_TO_FRONT 旗标来启动 Activity B，那么启动后的 Activity 栈中情形为 A、C、D、B。
- FLAG_ACTIVITY_SINGLE_TOP：该 Flag 相当于加载模式中的 singleTop 模式。例如，原来 Activity 栈中有 A、B、C、D 四个 Activity，在 Activity D 中再次启动 Activity D，Activity 栈中依然还是 A、B、C、D 四个 Activity。

Android 为 Intent 提供了大量的 Flag，每个 Flag 都有其特定的功能，具体请参考关于 Intent 的 API 文档。

5.3 使用 Intent 创建 Tab 页

前面已经介绍了如何使用 TabActivity 来创建 Activity 布局，前面添加 Tab 页时使用了 TabHost.TabSpec 提供的如下方法。

- setContent(int viewId)：直接将指定 View 组件设置成 Tab 页的 Content。

实际上 TabHost.TabSpec 还提供了如下方法。

- setContent(Intent intent)：直接将指定 Intent 对应的 Activity 设置成 Tab 页的 Content。

例如如下 Activity 代码。

程序清单：codes\05\5.3\IntentTab\app\src\main\java\org\crazyit\intent\MainActivity.java

```
public class MainActivity extends TabActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取该 Activity 里面的 TabHost 组件
        TabHost tabHost = getTabHost();
        // 使用 Intent 添加第一个 Tab 页
        tabHost.addTab(tabHost
            .newTabSpec("tab1")
            .setIndicator("已接电话",
                getResources().getDrawable(R.drawable.ic_launcher))
            .setContent(new Intent(this, BeCalledActivity.class)));
        // 使用 Intent 添加第二个 Tab 页
        tabHost.addTab(tabHost.newTabSpec("tab1")
            .setIndicator("呼出电话")
            .setContent(new Intent(this, CalledActivity.class)));
        // 使用 Intent 添加第三个 Tab 页
        tabHost.addTab(tabHost.newTabSpec("tab1")
            .setIndicator("未接电话")
            .setContent(new Intent(this, NoCallActivity.class)));
    }
}
```

上面程序中的三行粗体字代码用于为 TabHost.TabSpec 设置 Content，三行粗体字代码分别传入了三个 Intent 对象，这意味着该 TabHost 将直接以三个 Activity 类作为 Tab 页。上面的程序还需要 BeCalledActivity、CalledActivity、NoCallActivity 三个 Activity 类，不过这三个 Activity 都很简单，故此处不再给出。运行上面的程序，将看到如图 5.16 所示的界面。

图 5.16 使用 Intent 创建 Tab 页

5.4 本章小结

本章主要介绍了 Android 系统中 Intent 的功能和用法，当 Android 应用需要启动某个组件时，总需要借助于 Intent 来实现。不管是启动 Activity，还是启动 Service、BroadcastReceiver 组件，Android 系统都是由 Intent 来实现的。简单地说，Android 使用 Intent 封装了应用程序的“启动意图”，但这种“意图”并未直接与任何程序组件耦合，通过这种方式即可很好地提高系统的可扩展性和可维护性。学习本章需要重点掌握 Intent 的 Component、Action、Category、Data、Type 各属性的功能和用法，并掌握如何在 AndroidManifest.xml 文件中配置<intent-filter.../>元素。

CHAPTER

6

第6章 Android应用的资源

本章要点

- Android 应用的资源和作用
- Android 应用的资源的存储方式
- 在 XML 布局文件中使用资源
- 在 Java 程序中使用资源
- 使用字符串资源
- 使用颜色资源
- 使用尺寸资源
- 使用数组资源
- 使用图片资源
- 使用各种 Drawable 资源
- 使用原始 XML 资源
- 使用布局资源
- 使用菜单资源
- 使用样式和主题资源
- 使用属性资源
- 使用原始资源
- 使用资源进行程序国际化
- 自适应不同屏幕的资源

经过前面的介绍，相信读者对 Android 应用已有了大致的了解。如果从物理存在形式来分，Android 应用的源代码大致可分为如下三大类。

- **界面布局文件**：XML 文件，文件中每个标签都对应于相应的 View 标签。
- **Java 源文件**：应用中的 Activity、Service、BroadcastReceiver、ContentProvider 四大组件都是采用 Java 代码来实现的。
- **资源文件**：主要以各种 XML 文件为主，还可包括*.png、*.jpg、*.gif 图片资源。

在传统 Java 应用中，初学者很容易犯一个错误：直接在 Java 源代码中使用如"crazyit.org"、"hello"这样的字符串，或直接使用 123、0.9 这样的数值，而且不添加任何注释。过了一段时间后，即使自己再去看原来写的程序代码，一时之间，也无法理解其中"crazyit.org"、"hello"字符串，123、0.9 等数值的含义，这种方式就大大增加了程序的维护成本。这种直接在代码中定义的 123、0.9 等数值，也被称为“魔术数值”（就像表演魔术一样，其他人都搞不懂）。

为了改善这种情况，有经验的开发者会专门定义一个或多个接口或类，然后在其中以常量的形式来定义程序中用到的所有字符串、数值等，这些常量的名称十分明确，如 ResultSet.TYPE_FORWARD_ONLY，相信有经验的读者一看到这个常量就大致明白了它的含义。这样的方式就可以很好地提高程序的可维护性。

使用接口或类的形式来定义程序中用到的字符串、数值，虽然已经部分提高了程序的解耦，但后期维护、进一步开发时，开发人员还得去 Java“代码海”中打捞那些定义字符串常量、数值常量的位置，因此还有可以提高的地方。

Android 应用则对这种字符串常量、数值常量的定义做了进一步改进：Android 允许把应用中用到的各种资源，如字符串资源、颜色资源、数组资源、菜单资源等都集中放到 res 目录中定义，应用程序则直接使用这些资源中定义的值。

Android 应用中除了 res 目录用于存放资源之外，assets 目录也用于存放资源。一般来说，assets 目录下存放的资源代表应用无法直接访问的原生资源，应用程序需要通过 AssetManager 以二进制流的形式来读取资源。而 res 目录下的资源，Android SDK 会在编译该应用时，自动在 R.java 文件中为这些资源创建索引，程序可直接通过 R 资源清单类进行访问。

我们前面介绍的很多示例都是直接将字符串值写在界面布局文件中或 Java 程序代码中，实际上那并不是一种好的方式。只是前面还未详细介绍 Android 应用的资源，为了避免读者产生畏难心理，并未使用资源文件而已。

6.1 应用资源概述

Android 应用资源可分为两大类。

- 无法通过 R 资源清单类访问的原生资源，保存在 assets 目录下。
- 可通过 R 资源清单类访问的资源，保存在 res 目录下。

大部分时候提到 Android 应用资源时，往往都是指位于/res/目录下的应用资源，Android SDK 会在编译该应用时在 R 类中为它们创建对应的索引项。

6.1.1 资源的类型以及存储方式

Android 要求在 res 目录下用不同的子目录来保存不同的应用资源，表 6.1 大致显示了 Android 不同资源在/res/目录下的存储方式。

表 6.1 Android 应用资源的存储

目 录	存放的资源
/res/animator/	存放定义属性动画的 XML 文件
/res/anim/	存放定义补间动画的 XML 文件
/res/color/	存放定义不同状态下颜色列表的 XML 文件
/res/drawable/	存放各种位图文件（如*.png、*.9.png、*.jpg、*.gif）等。除此之外，也可能是编译成如下各种 Drawable 对象的 XML 文件： ➢ BitmapDrawable 对象 ➢ NinePatchDrawable 对象 ➢ StateListDrawable 对象 ➢ ShapeDrawable 对象 ➢ AnimationDrawable 对象 ➢ Drawable 的其他各种子类的对象
/res/layout/	存放各种用户界面的布局文件
/res/menu/	存放为应用程序定义各种菜单的资源，包括选项菜单、子菜单、上下文菜单资源
/res/raw/	存放任意类型的原生资源（比如音频文件、视频文件等）。在 Java 代码中可通过调用 Resources 对象的 openRawResource(int id)方法来获取该资源的二进制输入流。实际上，如果应用程序需要使用原生资源，也可把这些原生资源保存到/assets/目录下，然后在应用程序中使用 AssetManager 来访问这些资源
/res/values/	存放各种简单值的 XML 文件。这些简单值包括字符串值、整数值、颜色值、数组等 这些资源文件的根元素都是<resources.../>，为该<resources.../>元素添加不同的子元素则代表不同的资源，例如： ➢ string/integer/bool 子元素：代表添加一个字符串值、整数值或 boolean 值 ➢ color 子元素：代表添加一个颜色值 ➢ array 子元素或 string-array、int-array 子元素：代表添加一个数组 ➢ style 子元素：代表添加一个样式 ➢ dimen：代表添加一个尺寸 ➢ ... 由于各种简单值都可定义在/res/values/目录下的资源文件中，如果在同一份资源文件中定义各种值，势必增加程序维护的难度。为此，Android 建议使用不同的文件来存放不同类型的值，例如： ➢ arrays.xml：定义数组资源 ➢ colors.xml：定义颜色值资源 ➢ dimens.xml：定义尺寸值资源 ➢ strings.xml：定义字符串资源 ➢ styles.xml：定义样式资源
/res/xml/	存放任意的原生 XML 文件。这些 XML 文件可以在 Java 代码中使用 Resources.getXML()方法进行访问

提示：

对于 Android 5.0 版本而言，应用的/res/目录下不仅包含了 drawable 子目录，而且还提供了 drawable-ldpi（低分辨率）、drawable-mdpi（中等分辨率）、drawable-hdpi（高分辨率）、drawable-xhdpi（超高分辨率）4 个子目录，其实这 4 个子目录的作用就相当于 drawable 子目录。drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi 四个子目录下存放的图片文件的文件名完全相同，只是分辨率不同——这种做法可以让系统根据屏幕分辨率来选择相应的图片。大部分程序、系统将会选择 drawable-mdpi 目录下的图片文件，如果系统使用高分辨率屏幕，那么系统将会选择 drawable-hdpi 目录下的图片文件。

6.1.2 使用资源

在 Android 应用中使用资源可分为在 Java 代码和 XML 文件中使用资源，其中 Java 代码用于为 Android 应用定义四大组件，而 XML 文件则用于为 Android 应用定义各种资源。

1. 在 Java 代码中使用资源清单项

由于 Android SDK 会在编译应用时在 R 类中为/res/目录下所有资源创建索引项，因此在 Java 代码中访问资源主要通过 R 类来完成。其完整的语法格式为：

```
[<package_name>.]R.<resource_type>.<resource_name>
```

上面语法格式中各成分的说明如下。

- <package_name>：指定 R 类所在包，实际上就是使用全限定类名。当然，如果在 Java 程序中导入 R 类所在包，就可以省略包名。
- <resource_type>：R 类中代表不同资源类型的子类，例如 string 代表字符串资源。
- <resource_name>：指定资源的名称。该资源名称可能是无后缀的文件名（如图片资源），也可能是 XML 资源元素中由 android:name 属性所指定的名称。

例如如下代码片段：

```
// 从 drawable 资源中加载图片，并设为该窗口的背景
getWindow().setBackgroundDrawableResource(R.drawable.back) ;
// 从 string 资源中获取指定字符串资源，并设置该窗口的标题
getWindow().setTitle(getResources().getText(R.string.main_title));
// 获取指定的 TextView 组件，并设置该组件显示 string 资源中的指定字符串资源
TextView msg = (TextView) findViewById(R.id.msg);
msg.setText(R.string.hello_message);
```

2. 在 Java 代码中访问实际资源

R 资源清单类为所有的资源都定义了一个资源清单项，但这个清单项只是一个 int 类型的值，并不是实际的资源对象。在大部分情况下，Android 应用的 API 允许直接使用 int 类型的资源清单项代替应用资源。

但有些时候，程序也需要使用实际的 Android 资源，为了通过资源清单项来获取实际资源，可以借助于 Android 提供的 Resources 类。

提示：

笔者把 Resources 类称为“Android 资源访问总管家”，Resources 类提供了大量的方法来根据资源清单 ID 获取实际资源。

Resources 主要提供了如下两类方法。

- getXxx(int id)：根据资源清单 ID 来获取实际资源。
- getAssets()：获取访问/assets/目录下资源的 AssetManager 对象。

Resources 由 Context 调用 getResources()方法来获取。

下面的代码片段示范了如何通过 Resources 获取实际字符串资源。

```
// 直接调用 Activity 的 getResources()方法来获取 Resources 对象
Resources res = getResources();
// 获取字符串资源
String mainTitle = res.getText(R.string.main_title)
// 获取 Drawable 资源
Drawable logo = res.getDrawable(R.drawable.logo);
// 获取数组资源
int[] arr = res.getIntArray(R.array.books);
```

3. 在 XML 文件中使用资源

当定义 XML 资源文件时，其中的 XML 元素可能需要指定不同的值，这些值就可设置为已定义的资源项。在 XML 代码中使用资源的完整语法格式为：

```
@[<package_name>:]<resource_type>/<resource_name>
```

上面语法格式中各成分的说明如下。

- <package_name>：指定资源类所在应用的包。如果所引用的资源和当前资源位于同一个包下，则<package_name>可以省略。
- <resource_type>：R类中代表不同资源类型的子类。
- <resource_name>：指定资源的名称。该资源名称可能是无后缀的文件名（如图片资源），也可能是XML资源元素中由android:name属性所指定的名称。

例如如下代码片段在一份文件中定义了两种资源：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#ff00</color>
    <string name="hello">Hello!</string>
</resources>
```

接下来与它位于同一包中的XML资源文件就可通过如下方式来使用资源：

```
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:textColor="@color/red"
    android:text="@string/hello" />
```

下面将会对各种资源分别进行详细阐述。

6.2 字符串、颜色、尺寸资源

字符串资源、颜色资源、尺寸资源，它们对应的XML文件都将位于/res/values/目录下，它们默认的文件名以及在R类中对应的内部类如表6.2所示。

表6.2 字符串、颜色、尺寸资源表

资源类型	资源文件的默认名	对应于R类中的内部类的名称
字符串资源	/res/values/strings.xml	R.string
颜色资源	/res/values/colors.xml	R.color
尺寸资源	/res/values/dimens.xml	R.dimen

下面会通过示例来介绍字符串资源、颜色资源和尺寸资源的用法。

6.2.1 颜色值的定义

Android中的颜色值是通过红（Red）、绿（Green）、蓝（Blue）三原色以及一个透明度（Alpha）值来表示的，颜色值总是以井号（#）开头，接下来就是Alpha-Red-Green-Blue的形式。其中Alpha值可以省略，如果省略了Alpha值，那么该颜色默认是完全不透明的。

Android颜色值支持常见的4种形式。

- #RGB：分别指定红、绿、蓝三原色的值（只支持0～f这16级颜色）来代表颜色。
- #ARGB：分别指定红、绿、蓝三原色的值（只支持0～f这16级颜色）及透明度（只支持0～f这16级透明度）来代表颜色。
- #RRGGBB：分别指定红、绿、蓝三原色的值（支持00～ff这256级颜色）来代表颜色。
- #AARRGGBB：分别指定红、绿、蓝三原色的值（支持00～ff这256级颜色）以及透明度（支持00～ff这256级透明度）来代表颜色。

在上面4种形式中，A、R、G、B都代表一个十六进制的数，其中A代表透明度，R代表红色数值，G代表绿色数值，B代表蓝色数值。

提示：

关于颜色与三原色的知识，请读者自行参考光学方面的知识。此处笔者粗略介绍一下三原色理论：白色光大致可“分解”为红、绿、蓝三种光，红、绿、蓝三种光可合并成白色光。当红、绿、蓝都是最大值时，三种光合并就会变成白色光；当三种光的值相等，但不是最大值时，三种光将会合并成灰色光；如果其中一种光或两种光的值更亮，那么三种光合并就会产生彩色的光。

6.2.2 定义字符串、颜色、尺寸资源文件

字符串资源文件位于/res/values/目录下，字符串资源文件的根元素是<resources...>，该元素里每个<string.../>子元素定义一个字符串常量，其中<string.../>元素的name属性指定该常量的名称，<string.../>元素开始标签和结束标签之间的内容代表字符串值，如以下代码所示：

```
<!-- 定义一个字符串，名称为hello，字符串值为Hello World, ValuesResTest! -->
<string name="hello">Hello World, ValuesResTest!</string>
```

如下文件是该示例的字符串资源文件。

程序清单：codes\06\6.2\ValuesResTest\app\src\main\res\values\strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">字符串、颜色、尺寸资源</string>
    <string name="c1">F00</string>
    <string name="c2">0F0</string>
    <string name="c3">00F</string>
    <string name="c4">0FF</string>
    <string name="c5">F0F</string>
    <string name="c6">FF0</string>
    <string name="c7">07F</string>
    <string name="c8">70F</string>
    <string name="c9">F70</string>
</resources>
```

上面的程序代码中每个<string.../>元素定义一个字符串，其中<string.../>元素的name属性定义字符串的名称，<string>与</string>中间的内容就是该字符串的值。

颜色资源文件位于/res/values/目录下，颜色资源文件的根元素是<resources...>，该元素里每个<color.../>子元素定义一个字符串常量，其中<color.../>元素的name属性指定该颜色的名称，<color.../>元素开始标签和结束标签之间的内容代表颜色值，如以下代码所示：

```
<!-- 定义一个颜色，名称为c1，颜色为红色-->
<color name="c1">#F00</color>
```

如下文件是该示例的颜色资源文件。

程序清单：codes\06\6.2\ValuesResTest\app\src\main\res\values\colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="c1">#F00</color>
    <color name="c2">#0F0</color>
    <color name="c3">#00F</color>
    <color name="c4">#0FF</color>
    <color name="c5">#F0F</color>
    <color name="c6">#FF0</color>
```

```
    <color name="c7">#07F</color>
    <color name="c8">#70F</color>
    <color name="c9">#F70</color>
</resources>
```

上面的程序代码中每个<color.../>元素定义一个字符串，其中<color.../>元素的 name 属性定义颜色的名称，<color>与</color>中间的内容就是该颜色的值。

尺寸资源文件位于/res/values/目录下，尺寸资源文件的根元素是<resources...>，该元素里每个<dimen.../>子元素定义一个尺寸常量，其中<dimen.../>元素的 name 属性指定该尺寸的名称，<dimen.../>元素开始标签和结束标签之间的内容代表尺寸值，如以下代码所示。

程序清单：codes\06\6.2\ValuesResTest\app\src\main\res\values\dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="spacing">8dp</dimen>
    <!-- 定义 GridView 组件中每个单元格的宽度、高度 -->
    <dimen name="cell_width">60dp</dimen>
    <dimen name="cell_height">66dp</dimen>
    <!-- 定义主程序标题的字体大小 -->
    <dimen name="title_font_size">18sp</dimen>
</resources>
```

上面三份资源文件分别定义了字符串、颜色、尺寸资源，应用程序接下来既可在 XML 文件中使用这些资源，也可在 Java 代码中使用这些资源。

6.2.3 使用字符串、颜色、尺寸资源

正如前面介绍的，在 XML 文件中使用资源按如下语法格式：

```
@[<package_name>:]<resource_type>/<resource_name>
```

下面程序的界面布局中大量使用了前面定义的资源。

程序清单：codes\06\6.2\ValuesResTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<!-- 使用字符串资源、尺寸资源 -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/app_name"
    android:gravity="center"
    android:textSize="@dimen/title_font_size"/>
<!-- 定义一个 GridView 组件，使用尺寸资源中定义的长度来指定水平间距、垂直间距 -->
<GridView
    android:id="@+id/grid01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:horizontalSpacing="@dimen/spacing"
    android:verticalSpacing="@dimen/spacing"
    android:numColumns="3"
    android:gravity="center">
</GridView>
</LinearLayout>
```

上面程序中的粗体字代码就是使用字符串资源、尺寸资源的代码。

在 Java 代码中使用资源按如下语法格式：

```
[<package_name>.R.<resource_type>.<resource_name>
```

下面的 Java 程序同时使用了上面定义的三种资源。

程序清单：codes\06\6.2\ValuesResTest\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    // 使用字符串资源
    int[] textIds = new int[]
    {
        R.string.c1 , R.string.c2 , R.string.c3 ,
        R.string.c4 , R.string.c5 , R.string.c6 ,
        R.string.c7 , R.string.c8 , R.string.c9
    };
    // 使用颜色资源
    int[] colorIds = new int[]
    {
        R.color.c1 , R.color.c2 , R.color.c3 ,
        R.color.c4 , R.color.c5 , R.color.c6 ,
        R.color.c7 , R.color.c8 , R.color.c9
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 BaseAdapter 对象
        BaseAdapter ba = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                // 指定一共包含 9 个选项
                return textIds.length;
            }
            @Override
            public Object getItem(int position)
            {
                // 返回指定位置的文本
                return getResources().getText(textIds[position]);
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            // 重写该方法，该方法返回的 View 将作为 GridView 的每个格子
            @Override
            public View getView(int position
                , View convertView, ViewGroup parent)
            {
                TextView text = new TextView(MainActivity.this);
                Resources res = MainActivity.this.getResources();
                // 使用尺寸资源来设置文本框的高度、宽度
                text.setWidth((int) res.getDimension(R.dimen.cell_width));
                text.setHeight((int) res.getDimension(R.dimen.cell_height));
                // 使用字符串资源设置文本框的内容
                text.setText(textIds[position]);
                // 使用颜色资源来设置文本框的背景色
                text.setBackgroundResource(colorIds[position]);
                text.setTextSize(20);
```

```
            text.setTextSize(getResources()
                .getInteger(R.integer.font_size));
            return text;
        }
    };
    GridView grid = (GridView)findViewById(R.id.grid01);
    // 为GridView设置Adapter
    grid.setAdapter(ba);
  }
}
```

上面程序中的粗体字代码分别使用了前面定义的字符串资源、颜色资源和尺寸资源。运行上面的程序，将可以看到如图6.1所示的界面。

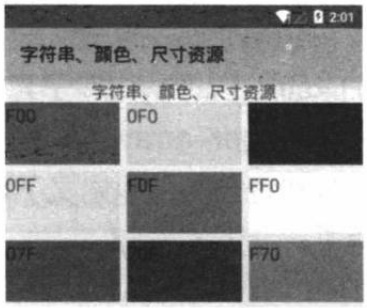


图6.1 使用字符串、颜色、尺寸资源

与定义字符串资源类似的是，Android 也允许使用资源文件来定义 boolean 常量。例如，在/res/values/目录下增加一个 bools.xml 文件，该文件的根元素也是<resources.../>，根元素内通过<bool.../>子元素来定义 boolean 常量，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="is_male">true</bool>
    <bool name="is_big">false</bool>
</resources>
```

一旦在资源文件中定义了如上所示的资源文件之后，接下来在 Java 代码中按如下语法格式访问即可：

```
[<package_name>.]R.bool.bool_name
```

在 XML 文件中按如下格式即可访问资源：

```
@[<package_name>:]bool/bool_name
```

例如，为了在 Java 代码中获取指定 boolean 变量的值，可通过如下代码来实现：

```
Resources res = getResources();
boolean is_male = res.getBoolean(R.bool.is_male);
```

与定义字符串资源类似的是，Android 也允许使用资源文件来定义整型常量。例如，在/res/values/目录下增加一个 integers.xml 文件（文件名可以自由选择），该文件的根元素也是<resources.../>，根元素内通过<integer.../>子元素来定义整型常量，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="my_size">32</integer>
    <integer name="book_numbers">12</integer>
</resources>
```

一旦在资源文件中定义了如上所示的资源文件之后，接下来在 Java 代码中按如下语法格式访问即可：

```
[<package_name>.]R.integer.integer_name
```

在 XML 文件中按如下格式即可访问资源：

```
@[<package_name>:]integer/integer_name
```

例如，为了在 Java 代码中获取指定整型变量的值，可通过如下代码来实现：

```
Resources res = getResources();
int my_size = res.getInteger(R.integer.my_size);
```

6.3 数组（Array）资源

上面的程序在 Java 代码中定义了两个数组，Android 并不推荐在 Java 代码中定义数组，因为 Android 允许通过资源文件来定义数组资源。

Android 采用位于/res/values 目录下的 arrays.xml 文件来定义数组资源，定义数组时 XML 资源文件的根元素也是<resources.../>，该元素内可包含如下三种子元素。

- **<array..../>子元素**：定义普通类型的数组，例如 Drawable 数组。
- **<string-array..../>子元素**：定义字符串数组。
- **<integer-array..../>子元素**：定义整型数组。

一旦在资源文件中定义了数组资源之后，接下来就可以在 Java 程序中通过如下形式来访问资源了：

```
[<package_name>.]R.array.array_name
```

在 XML 文件中则可通过如下形式进行访问：

```
@[<package_name>:]array/array_name
```

为了能在 Java 程序中访问到实际数组，Resources 提供了如下方法。

- **String[] getStringArray(int id)**：根据资源文件中字符串数组资源的名称来获取实际的字符串数组。
- **int[] getIntArray(int id)**：根据资源文件中整型数组资源的名称来获取实际的整型数组。
- **TypedArray obtainTypedArray(int id)**：根据资源文件中普通数组资源的名称来获取实际的普通数组。

TypedArray 代表一个通用类型的数组，该类提供了 getXxx(int index)方法来获取指定索引处的数组元素。

下面为该应用程序增加如下数组资源文件。

程序清单：codes\06\6.3\ArrayResTest\app\src\main\res\values\arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 定义一个 Drawable 数组 -->
    <array name="plain_arr">
        <item>@color/c1</item>
        <item>@color/c2</item>
        <item>@color/c3</item>
        <item>@color/c4</item>
        <item>@color/c5</item>
        <item>@color/c6</item>
        <item>@color/c7</item>
        <item>@color/c8</item>
        <item>@color/c9</item>
    </array>
    <!-- 定义字符串数组 -->
    <string-array name="string_arr">
        <item>@string/c1</item>
        <item>@string/c2</item>
        <item>@string/c3</item>
        <item>@string/c4</item>
        <item>@string/c5</item>
        <item>@string/c6</item>
        <item>@string/c7</item>
        <item>@string/c8</item>
```

```
        <item>@string/c9</item>
    </string-array>
    <!-- 定义字符串数组 -->
    <string-array name="books">
        <item>疯狂 Java 讲义</item>
        <item>疯狂 Ajax 讲义</item>
        <item>疯狂 Android 讲义</item>
    </string-array>
</resources>
```

定义了上面的数组资源之后，既可在 XML 文件中使用这些数组资源，也可在 Java 程序中使用这些数组资源。例如，如下界面布局文件中定义了一个 ListView 数组，并将 android:entries 属性值指定为一个数组。界面布局文件代码如下。

程序清单：codes\06\6.3\ArrayResTest\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<!-- 省略其他组件定义 -->
...
<!-- 定义 ListView 组件，使用了数组资源 -->
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/books"/>
</LinearLayout>
```

接下来程序中无须定义数组，程序直接使用资源文件中定义的数组。程序代码如下。

程序清单：codes\06\6.3\ArrayResTest\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    // 获取系统定义的数组资源
    String[] texts;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        texts = getResources().getStringArray(R.array.string_arr);
        // 创建一个 BaseAdapter 对象
        BaseAdapter ba = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                // 指定一共包含 9 个选项
                return texts.length;
            }
            @Override
            public Object getItem(int position)
            {
                // 返回指定位置的文本
                return texts[position];
            }
            @Override
            public long getItemId(int position)
            {
```

```
            return position;
        }
        // 重写该方法，该方法返回的 View 将作为 GridView 的每个格子
        @Override
        public View getView(int position
                , View convertView, ViewGroup parent)
        {
            TextView text = new TextView(MainActivity.this);
            Resources res = MainActivity.this.getResources();
            // 使用尺寸资源来设置文本框的高度、宽度
            text.setWidth((int) res.getDimension(R.dimen.cell_width));
            text.setHeight((int) res.getDimension(R.dimen.cell_height));
            // 使用字符串资源设置文本框的内容
            text.setText(texts[position]);
            TypedArray icons = res.obtainTypedArray(R.array.plain_arr);
            // 使用颜色资源来设置文本框的背景色
            text.setBackground(icons.getDrawable(position));
            text.setTextSize(20);
            return text;
        }
    };
    GridView grid = (GridView) findViewById(R.id.grid01);
    // 为 GridView 设置 Adapter
    grid.setAdapter(ba);
  }
}
```

上面程序中的粗体字代码就是使用数组资源的关键代码。运行上面的程序，将看到如图 6.2 所示的结果。

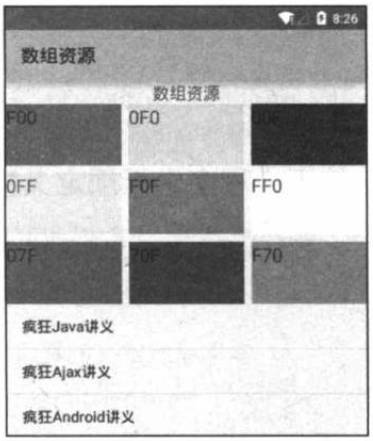


图 6.2　使用数组资源

6.4　使用 Drawable 资源

Drawable 资源是 Android 应用中使用最广泛的资源，也是 Android 应用中最灵活的资源，它不仅可以直接使用*.png、*.jpg、*.gif、*.9.png 等图片作为资源，也可以使用多种 XML 文件作为资源。只要一份 XML 文件可以被系统编译成 Drawable 子类的对象，那么这份 XML 文件即可作为 Drawable 资源。

Drawable 资源通常保存在/res/drawable 目录下，实际上可能保存在/res/drawable-ldpi、/res/drawable- mdpi、/res/drawable-hdpi、/res/drawable-xhdpi 目录下。

下面详细介绍各种 Drawable 资源。

6.4.1　图片资源

图片资源是最简单的 Drawable 资源，只要把*.png、*.jpg、*.gif 等格式的图片放入/res/drawable-xxx 目录下，Android SDK 就会在编译应用中自动加载该图片，并在 R 资源清单类中生成该资源的索引。

注意：

Android 要求图片资源的文件名必须符合 Java 标识符的命名规则，否则 Android SDK 无法为该图片在 R 类中生成资源索引。

一旦系统在 R 资源清单类中生成了指定资源的索引，接下来就可以在 Java 类中使用如下语法格式来访问该资源：

```
[<package_name>.]R.drawable.<file_name>
```

在 XML 代码中则按如下语法格式来访问该资源：

```
@[<package_name>:]drawable/file_name
```

为了在程序中获得实际的 Drawable 对象，Resources 提供了 Drawable getDrawable(int id) 方法，该方法即可根据 Drawable 资源在 R 资源清单类中的 ID 来获取实际的 Drawable 对象。

关于图片资源的示例，本书前面已有大量介绍，此处不再赘述。

6.4.2 StateListDrawable 资源

StateListDrawable 用于组织多个 Drawable 对象。当使用 StateListDrawable 作为目标组件的背景、前景图片时，StateListDrawable 对象所显示的 Drawable 对象会随目标组件状态的改变而自动切换。

定义 StateListDrawable 对象的 XML 文件的根元素为<selector.../>，该元素可以包含多个<item.../>元素，该元素可指定如下属性。

- android:color 或 android:drawable：指定颜色或 Drawable 对象。
- android:state_xxx：指定一个特定状态。

例如如下语法格式：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <!-- 指定特定状态下的颜色 -->
    <item android:color="hex_color"
        android:state_pressed=["true" | "false"] />
</selector>
```

StateListDrawable 的<item.../>元素所支持的状态有如表 6.3 所示的几种。

表 6.3 StateListDrawable 的<item.../>元素所支持的状态

属性值	含义
android:state_active	代表是否处于激活状态
android:state_checkable	代表是否处于可勾选状态
android:state_checked	代表是否处于已勾选状态
android:state_enabled	代表是否处于可用状态
android:state_first	代表是否处于开始状态
android:state_focused	代表是否处于已得到焦点状态
android:state_last	代表是否处于结束状态
android:state_middle	代表是否处于中间状态
android:state_pressed	代表是否处于已被按下状态
android:state_selected	代表是否处于已被选中状态
android:state_window_focused	代表窗口是否处于已得到焦点状态

实例：高亮显示正在输入的文本框

通过前面介绍我们知道，使用 EditText 时可指定一个 android:textColor 属性，该属性用于指定文本框的文字颜色。前面介绍该属性时总是直接给它一个颜色值，因此该文本框的文字颜色总是固定的。借助于 StateListDrawable 对象，可以让文本框中的文字颜色随文本框的状态动态改变。

为本应用提供如下 Drawable 资源文件。

程序清单：codes\06\6.4\StateListDrawableTest\app\src\main\res\drawable\my_image.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 指定获得焦点时的颜色 -->
    <item android:state_focused="true"
        android:color="#f44"/>
    <!-- 指定失去焦点时的颜色 -->
    <item android:state_focused="false"
        android:color="#ccf"/>
</selector>
```

上面的资源文件中指定了目标组件得到焦点、失去焦点时使用不同的颜色，接下来可以在定义 EditText 时使用该资源。

程序清单：codes\06\6.4\StateListDrawableTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 使用 StateListDrawable 资源 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@drawable/my_image"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@drawable/my_image"/>
</LinearLayout>
```

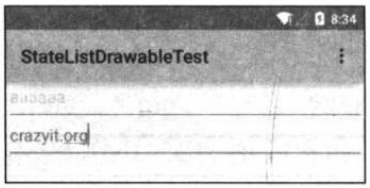


图 6.3 输入文本时高亮显示

该应用的 Java 程序代码不需要任何修改，只要显示该界面布局即可。运行该程序，将看到如图 6.3 所示的界面。

通过使用 StateListDrawable 不仅可以让文本框里文字的颜色随文本框状态的改变而切换，也可让按钮的背景图片随按钮状态的改变而切换，实际上 StateListDrawable 的功能非常灵活，它可以让各种组件的背景、前景随状态的改变而切换。

6.4.3 LayerDrawable 资源

与 StateListDrawable 有点类似，LayerDrawable 也可包含一个 Drawable 数组，因此系统将会按这些 Drawable 对象的数组顺序来绘制它们，索引最大的 Drawable 对象将会被绘制在最上面。

定义 LayerDrawable 对象的 XML 文件的根元素为<layer-list.../>，该元素可以包含多个<item.../>元素，该元素可指定如下属性。

- android:drawable：指定作为 LayerDrawable 元素之一的 Drawable 对象。
- android:id：为该 Drawable 对象指定一个标识。
- android:buttom|top|left|button：它们用于指定一个长度值，用于指定将该 Drawable 对象绘制到目标组件的指定位置。

例如如下语法格式：

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

```
    <!-- 指定一个 Drawable 元素 -->
    <item android:id="@android:id/background"
        android:drawable="@drawable/grow" />
</layer-list >
```

实例：定制拖动条的外观

通过前面介绍我们知道，使用 SeekBar 时可指定一个 android:progressDrawable 属性，该属性可改变 SeekBar 的外观，借助于 LayerDrawable 即可改变 SeekBar 的规定、已完成部分的 Drawable 对象。

例如定义如下 Drawable 资源。

程序清单：codes\06\6.4\LayerDrawableTest\app\src\main\res\drawable\my_bar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 定义轨道的背景 -->
    <item android:id="@android:id/background"
        android:drawable="@drawable/grow" />
    <!-- 定义轨道上已完成部分的外观-->
    <item android:id="@android:id/progress"
        android:drawable="@drawable/ok" />
</layer-list>
```

除此之外，该示例还定义了如下 LayerDrawable 对象。

程序清单：codes\06\6.4\LayerDrawableTest\app\src\main\res\drawable\layout_logo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
      <bitmap android:src="@drawable/ic_launcher"
        android:gravity="center" />
    </item>
    <item android:top="25dp" android:left="25dp">
      <bitmap android:src="@drawable/ic_launcher"
        android:gravity="center" />
    </item>
    <item android:top="50dp" android:left="50dp">
      <bitmap android:src="@drawable/ic_launcher"
        android:gravity="center" />
    </item>
</layer-list>
```

上面的程序中定义了三个“层叠”在一起的 Drawable 对象，接着在界面布局中使用上面的 my_bar.xml 定义的 Drawable 对象来改变 SeekBar 的外观，并通过 ImageView 来显示上面 layout_logo 的 Drawable 组件。界面布局的代码片段如下。

程序清单：codes\06\6.4\LayerDrawableTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
    <!-- 定义一个拖动条，并改变轨道外观 -->
    <SeekBar
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progressDrawable="@drawable/my_bar"/>
    <ImageView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/layout_logo"/>
</LinearLayout>
```

该程序的代码无须任何改变，直接加载、显示上面的界面布局文件即可，运行该程序出现如图 6.4 所示的界面。

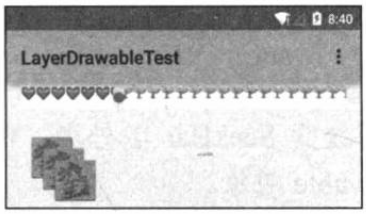


图 6.4　使用 LayerDrawable 资源

6.4.4　ShapeDrawable 资源

ShapeDrawable 用于定义一个基本的几何图形（如矩形、圆形、线条等），定义 ShapeDrawable 的 XML 文件的根元素是<shape.../>元素，该元素可指定如下属性。

- android:shape=["rectangle" | "oval" | "line" | "ring"]：指定定义哪种类型的几何图形。

定义 ShapeDrawable 对象的完整语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <!-- 定义几何图形的四个角的弧度 -->
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <!-- 定义使用渐变色填充 -->
    <gradient
        android:angle="integer"
        android:centerX="integer"
        android:centerY="integer"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=["linear" | "radial" | "sweep"]
        android:usesLevel=["true" | "false"] />
    <!-- 定义几何形状的内边距 -->
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <!-- 定义几何形状的大小 -->
    <size
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
    <!-- 定义使用单种颜色填充 -->
    <solid
        android:color="color" />
    <!-- 定义为几何形状绘制边框 -->
    <stroke
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
</shape>
```

下面通过实例来介绍 ShapeDrawable 资源的定义和使用。

实例：椭圆形、渐变背景的文本框

前面介绍 TextView 时知道该组件可指定一个 android:background 属性，该属性用于为该文本框指定背景。大部分时候，文本框的背景只是一个简单的图片，或者只是一个简单的颜色。

如果程序使用 ShapeDrawable 资源作为文本框的 android:background 属性，则可以在 Android 应用中做出各种外观的文本框。下面先定义如下的 ShapeDrawable 资源。

程序清单：codes\06\6.4\ShapeDrawableTest\app\src\main\res\drawable\my_shape_1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- 设置填充颜色 -->
    <solid android:color="#fff"/>
    <!-- 设置四周的内边距 -->
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <!-- 设置边框 -->
    <stroke android:width="3dip" android:color="#ff0" />
</shape>
```

再定义如下的 ShapeDrawable 资源。

程序清单：codes\06\6.4\ShapeDrawableTest\app\src\main\res\drawable\my_shape_2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- 定义填充渐变颜色 -->
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <!-- 设置内填充 -->
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <!-- 设置圆角矩形 -->
    <corners android:radius="8dp" />
</shape>
```

接下来定义如下的 ShapeDrawable 资源。

程序清单：codes\06\6.4\ShapeDrawableTest\app\src\main\res\drawable\my_shape_3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <!-- 定义填充渐变颜色 -->
    <gradient
        android:startColor="#ff0"
        android:endColor="#00f"
        android:angle="45"
        android:type="sweep"/>
    <!-- 设置内填充 -->
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <!-- 设置圆角矩形 -->
```

```
    <corners android:radius="8dp" />
</shape>
```

定义了上面三个 ShapeDrawable 资源之后，接下来在界面布局文件中用这三个 ShapeDrawable 资源作为文本框的背景。界面布局文件代码如下。

程序清单：codes\06\6.4\ShapeDrawableTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/my_shape_1"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/my_shape_2"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/my_shape_3"/>
</LinearLayout>
```

使用 Activity 加载、显示上面的界面布局文件，将可以看到如图 6.5 所示的界面。

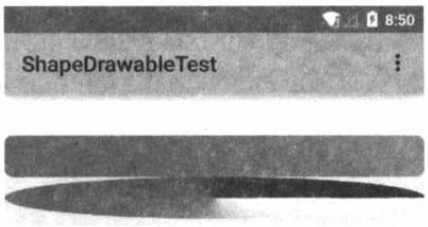


图 6.5　使用 ShapeDrawable 资源

➤➤ 6.4.5　ClipDrawable 资源

ClipDrawable 代表从其他位图上截取的一个“图片片段”。在 XML 文件中定义 ClipDrawable 对象使用<clip.../>元素，该元素的语法为：

```
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:clipOrientation=["horizontal" | "vertical"]
    android:gravity=["top" | "bottom" | "left" | "right" | "center_vertical" |
        "fill_vertical" | "center_horizontal" | "fill_horizontal" |
        "center" | "fill" | "clip_vertical" | "clip_horizontal"] />
```

上面的语法格式中可指定如下三个属性。

- android:drawable：指定截取的源 Drawable 对象。
- android:clipOrientation：指定截取方向，可设置水平截取或垂直截取。
- android:gravity：指定截取时的对齐方式。

使用 ClipDrawable 对象时可调用 setLevel(int level)方法来设置截取的区域大小，当 level 为 0 时，截取的图片片段为空；当 level 为 10000 时，截取整张图片。

下面以一个实例来说明 ClipDrawable 对象的用法。

实例：徐徐展开的风景

因为 ClipDrawable 对象可调用 setLevel(int level)控制截取图片的部分，因此本实例只要设置一个定时器，让程序不断调用 ClipDrawable 的 setLevel(int level)方法即可实现图片徐徐展开的效果。

程序先定义如下 ClipDrawable 对象。

程序清单：codes\06\6.4\ClipDrawableTest\app\src\main\res\drawable\my_clip.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/shuangta"
    android:clipOrientation="horizontal"
    android:gravity="center">
</clip>
```

上面的程序控制从中间开始截取图片，截取方向为水平截取。接下来程序将通过一个定时器来定期修改 ClipDrawable 对象的 level，即可实现图片徐徐张开的效果。

程序清单：codes\06\6.4\ClipDrawableTest\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ImageView imageview = (ImageView) findViewById(R.id.image);
        // 获取图片所显示的 ClipDrawable 对象
        final ClipDrawable drawable = (ClipDrawable)
            imageview.getDrawable();
        final Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 如果该消息是本程序所发送的
                if (msg.what == 0x1233)
                {
                    // 修改 ClipDrawable 的 level 值
                    drawable.setLevel(drawable.getLevel() + 200);
                }
            }
        };
        final Timer timer = new Timer();
        timer.schedule(new TimerTask()
        {
            @Override
            public void run()
            {
                Message msg = new Message();
                msg.what = 0x1233;
                // 发送消息，通知应用修改 ClipDrawable 对象的 level 值
                handler.sendMessage(msg);
                // 取消定时器
                if (drawable.getLevel() >= 10000)
                {
                    timer.cancel();
                }
            }
        }, 0, 300);
    }
}
```

运行上面的程序，将看到如图 6.6 所示的结果。

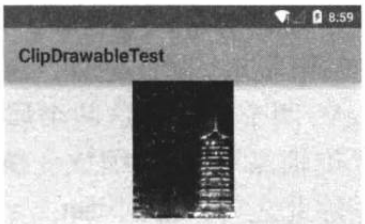


图 6.6 使用 ClipDrawable 对象

从图 6.6 所示的运行结果可以看出，通过使用这种徐徐展开的图片，用户会感觉就像进度条一样——实际上，在实

际应用中完全可以使用这种 ClipDrawable 对象来实现图片进度条。

6.4.6 AnimationDrawable 资源

AnimationDrawable 代表一个动画，关于 Android 动画的知识本书后面还有更详细的介绍，本节只是先介绍一下如何定义 AnimationDrawable 资源。Android 既支持传统的逐帧动画（类似于电影方式，一张图片、一张图片地切换），也支持通过平移、变换计算出来的补间动画。

下面以补间动画为例来介绍如何定义 AnimationDrawable 资源。定义补间动画的 XML 资源文件以<set.../>元素作为根元素，该元素内可以指定如下 4 个元素。

- alpha：设置透明度的改变。
- scale：设置图片进行缩放变换。
- translate：设置图片进行位移变换。
- rotate：设置图片进行旋转。

定义动画的 XML 资源应该放在/res/anmi/路径下，当使用 Android Studio 创建一个 Android 应用时，默认不会包含该路径，开发者需要自行创建该路径。

定义补间动画的思路很简单：设置一张图片的开始状态（包括透明度、位置、缩放比、旋转度），并设置该图片的结束状态（包括透明度、位置、缩放比、旋转度），再设置动画的持续时间，Android 系统会使用动画效果把这张图片从开始状态变换到结束状态。

设置补间动画的语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"]
    android:duration="持续时间">
    <alpha
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
</set>
```

上面语法格式中包含了大量的 fromXxx、toXxx 属性，这些属性就用于定义图片的开始状态、结束状态。除此之外，当进行缩放变换（scale）、旋转（rotate）变换时，还需要指定 pivotX、pivotY 两个属性，这两个属性用于指定变换的“中心点”——比如进行旋转变换时，需要指定“旋轴点”；进行缩放变换时，需要指定“中心点”。

除此之外，上面<set.../>、<alpha.../>、<scale.../>、<translate.../>、<rotate.../>都可指定一个 android:interpolator 属性，该属性指定动画的变化速度，可以实现匀速、正加速、负加速、无

规则变加速等，Android系统的R.anim类中包含了大量常量，它们定义了不同的动画速度，例如：

- linear_interpolator：匀速变换。
- accelerate_interpolator：加速变换。
- decelerate_interpolator：减速变换。

如果程序想让<set.../>元素下所有的变换效果使用相同的动画速度，则可指定android:shareInterpolator="true"。

例如，下面的资源文件定义了一个动画资源。

程序清单：codes\06\6.4\AnimationDrawable\app\src\main\res\anim\my_anim.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:duration="5000">
    <!-- 定义缩放变换 -->
    <scale android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="true"
        android:duration="2000"/>
    <!-- 定义位移变换 -->
    <translate android:fromXDelta="10"
        android:toXDelta="130"
        android:fromYDelta="30"
        android:toYDelta="-80"
        android:duration="2000"/>
</set>
```

上面的动画资源文件十分简单，它只指定了图片资源需要进行两种变换：缩放变换和位移变换。

一旦定义了上面的动画资源文件，接下来就可以在XML文件中按如下语法格式来访问它：

```
@[<package_name>:]anim/file_name
```

在Java代码中则按如下语法格式来访问：

```
[<package>.]R.anim.<file_name>
```

为了在Java代码中获取实际的Animation对象，可调用AnimationUtils的如下方法：

- loadAnimation(Context ctx, int resId)

下面的程序示范了如何使用AnimationDrawable资源。

程序清单：codes\06\6.4\AnimationDrawable\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ImageView image = (ImageView)findViewById(R.id.image);
        // 加载动画资源
        final Animation anim = AnimationUtils.loadAnimation(this,
            R.anim.my_anim);
        // 设置动画结束后保留结束状态
```

```
        anim.setFillAfter(true);  // ①
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 开始动画
                image.startAnimation(anim);
            }
        });
    }
}
```

图 6.7　使用 AnimationDrawable 资源

上面的程序使用了界面布局中的两个组件：一个 ImageView、一个 Button，这是两个最普通的组件，故此处不再给出界面布局代码。运行上面的程序，将看到如图 6.7 所示的动画效果。

上面的程序中①号代码设置动画结束后保留图片的变换结果。本来 Android 的 API 文档中说明可以在<alpha.../>、<scale.../>、<translate.../>、<rotate.../>等元素中指定 android:fillAfter 为 true 来实现这个效果，但实际上要为<set.../>设置 android:fillAfter 为 true 才可以。

6.5　属性动画（Property Animation）资源

Animator 代表一个属性动画，但它只是一个抽象类，通常会使用它的子类：AnimatorSet、ValueAnimator、ObjectAnimator、TimeAnimator。关于 Android 动画的知识，本书后面还有更详细的介绍，本节只是先介绍一下如何定义属性动画资源。

定义属性动画的 XML 资源文件能以如下三个元素中的任意一个作为根元素。

- **<set.../>**：它是一个父元素，用于包含<objectAnimator.../>、<animator.../>或<set.../>子元素，该元素定义的资源代表 AnimatorSet 对象。
- **<objectAnimator.../>**：用于定义 ObjectAnimator 动画。
- **<animator.../>**：用于定义 ValueAnimator 动画。

定义属性动画的语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<set android:ordering=["together" | "sequentially"]>
    <objectAnimator
        android:propertyName="string"
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:interpolator=""
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>
    <animator
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
```

```
        android:interpolator=""
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>
    <set>
        ...
    </set>
</set>
```

实例：不断渐变的背景色

该实例将会使用属性动画来控制组件背景色不断渐变。该实例所使用的属性动画资源文件如下。

程序清单：codes\06\6.5\AnimatorTest\app\src\main\res\animator\color_anim.xml

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"
    android:propertyName="backgroundColor"
    android:duration="3000"
    android:valueFrom="#FF8080"
    android:valueTo="#8080FF"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:valueType="intType">
</objectAnimator>
```

上面的代码定义了一个 ObjectAnimator 对象，接下来程序就可以通过属性动画来控制指定组件的背景色不断改变。下面是该实例的 Activity 代码。

程序清单：codes\06\6.5\AnimatorTest\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout container = (LinearLayout)
            findViewById(R.id.container);
        // 添加 MyAnimationView 组件
        container.addView(new MyAnimationView(this));
    }
    public class MyAnimationView extends View
    {
        public MyAnimationView(Context context)
        {
            super(context);
            // 加载动画资源
            ObjectAnimator colorAnim = (ObjectAnimator) AnimatorInflater
                .loadAnimator(MainActivity.this, R.animator.color_anim);
            colorAnim.setEvaluator(new ArgbEvaluator());
            // 对该 View 本身应用属性动画
            colorAnim.setTarget(this);
            // 开始指定动画
            colorAnim.start();
        }
    }
}
```

上面程序中的粗体字代码使用 AnimatorInflater 工具类加载了指定动画资源文件，并将该动画资源文件转换为 ObjectAnimator 对象。接下来程序对 MyAnimationView 本身应用该动画，

将可以看到该组件的背景色不断变化，如图 6.8 所示。

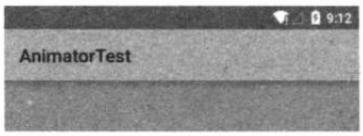


图 6.8　不断渐变的背景色

6.6　使用原始 XML 资源

在某些时候，Android 应用有一些初始化的配置信息、应用相关的数据资源需要保存，一般推荐使用 XML 文件来保存它们，这种资源就被称为原始 XML 资源。下面介绍如何定义、获取原始 XML 资源。

6.6.1　定义原始 XML 资源

原始 XML 资源一般保存在/res/xml/路径下——当使用 Android Studio 创建 Android 应用时，/res/目录下并没有包含 xml 子目录，开发者应该自行手动创建 xml 子目录。

接下来 Android 应用对原始 XML 资源没有任何特殊的要求，只要它是一份格式良好的 XML 文档即可。

一旦成功地定义了原始 XML 资源，接下来在 XML 文件中就可通过如下语法格式来访问它：

```
@[<package_name>:]xml/file_name
```

在 Java 代码中则按如下语法格式来访问：

```
[<package_name>.]R.xml.<file_name>
```

为了在 Java 程序中获取实际的 XML 文档，可以通过 Resources 的如下两个方法来实现。

- XmlResourceParser getXml(int id)：获取 XML 文档，并使用一个 XmlPullParser 来解析该 XML 文档，该方法返回一个解析器对象（XmlResourceParser 是 XmlPullParser 的子类）。
- InputStream openRawResource(int id)：获取 XML 文档对应的输入流。

大部分时候，我们可以直接调用 getXml(int id)方法来获取 XML 文档，并对该文档进行解析。Android 默认使用内置的 Pull 解析器来解析 XML 文件。

除了使用 Pull 解析之外，Java 开发者还可使用 DOM 或 SAX 对 XML 文档进行解析。一般的 Java 应用会使用 JAXP API 来解析 XML 文档。对于实际的 Java EE 项目而言，使用 JDOM 或 dom4j 进行解析可能更加简便。

提示：

Pull 解析器是一个开源项目，既可以用于 Android 应用，也可以用于 Java EE 应用。如果需要在 Java EE 应用中使用 Pull 解析器，则需要自行下载并添加 Pull 解析器的 JAR 包。不过 Android 平台已经内置了 Pull 解析器，而且 Android 系统本身也使用 Pull 解析器来解析各种 XML 文档，因此 Android 推荐开发者使用 Pull 解析器来解析 XML 文档。

Pull 解析方式有点类似于 SAX 解析，它们都采用事件驱动的方式来进行解析。当 Pull 解析器开始解析之后，开发者可不断地调用 Pull 解析器的 next()方法获取下一个解析事件（开始文档、结束文档、开始标签、结束标签等），当处于某个元素处时，可调用 XmlPullParser 的 getAttributeValue()方法来获取该元素的属性值，也可调用 XmlPullParser 的 nextText()方法来获取文本节点的值。

如果开发者希望使用 DOM、SAX 或其他解析器来解析 XML 资源，那么可调用

openRawResource (int id)方法来获取XML资源对应的输入流，这样即可自行选择解析器来解析该XML资源了。

提示：

使用其他XML解析器需要开发者自行下载并安装解析器的JAR包。关于DOM、SAX、JAXP、dom4j、JDOM的相关知识，或者读者需要获取更多关于XML的知识，请参考疯狂Java体系的《疯狂XML讲义》。

6.6.2 使用原始XML文件

下面为示例程序添加一个原始的XML文件，将该XML文件放到/res/xml目录下。该XML文件的内容很简单，如下所示。

程序清单：codes\06\6.6\XmlResTest\app\src\main\res\xml\books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <book price="109.0" 出版日期="2008年">疯狂Java讲义</book>
    <book price="108.0" 出版日期="2009年">轻量级Java EE企业应用实战</book>
    <book price="79.0" 出版日期="2009年">疯狂Ajax讲义</book>
</books>
```

接下来就可以在Java程序中获取该XML资源，并解析该XML资源中的信息了。Java程序如下。

程序清单：codes\06\6.6\XmlResTest\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取bn按钮，并为该按钮绑定事件监听器
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 根据XML资源的ID获取解析该资源的解析器
                // XmlResourceParser是XmlPullParser的子类
                XmlResourceParser xrp = getResources().getXml(R.xml.books);
                try
                {
                    StringBuilder sb = new StringBuilder("");
                    // 还没有到XML文档的结尾处
                    while (xrp.getEventType()
                        != XmlResourceParser.END_DOCUMENT)
                    {
                        // 如果遇到开始标签
                        if (xrp.getEventType() ==
                            XmlResourceParser.START_TAG)
                        {
                            // 获取该标签的标签名
                            String tagName = xrp.getName();
                            // 如果遇到book标签
                            if (tagName.equals("book"))
                            {
```

```
                        // 根据属性名来获取属性值
                        String bookName = xrp
                        .getAttributeValue(null, "price");
                        sb.append("价格：");
                        sb.append(bookName);
                        // 根据属性索引来获取属性值
                        String bookPrice = xrp
                        .getAttributeValue(1);
                        sb.append("    出版日期：");
                        sb.append(bookPrice);
                        sb.append(" 书名：");
                        // 获取文本节点的值
                        sb.append(xrp.nextText());
                    }
                    sb.append("\n");
                }
                // 获取解析器的下一个事件
                xrp.next(); // ①
            }
            EditText show = (EditText) findViewById(R.id.show);
            show.setText(sb.toString());
        }
        catch (XmlPullParserException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
});
}
}
```

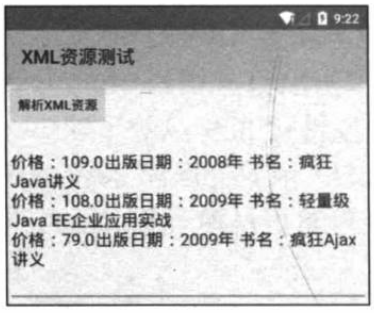


图 6.9　使用原始 XML 资源

上面程序中的①号粗体字代码用于不断地获取 Pull 解析的解析事件，程序中第一行粗体字代码只要解析事件不等于 XmlResourceParser.END_DOCUMENT（也就是还没有解析结束），程序就将一直解析下去，通过这种方式即可把整份 XML 文档的内容解析出来。

上面的程序中包含一个按钮和一个文本框，当用户单击该按钮时，程序将会解析指定 XML 文档，并把文档中的内容显示出来。运行该程序，然后单击“解析 XML 资源”按钮，程序显示如图 6.9 所示的界面。

6.7　使用布局（Layout）资源

实际上，从我们学习第一个 Android 应用开始，已经开始接触 Android 的 Layout 资源了，因此此处不会详细介绍 Android 的 Layout 资源的知识，只是对 Layout 资源进行简单的归纳。

Layout 资源文件应放在/res/layout/目录下，Layout 资源文件的根元素通常是各种布局管理器，比如 LinearLayout、TableLayout、FrameLayout 等，接着在该布局管理器中定义各种 View 组件即可。

一旦在 Android 项目中定义了 Layout 资源，接下来在 XML 文件中就可通过如下语法格式来访问它：

```
@[<package_name>:]layout/file_name
```

在 Java 代码中则按如下语法格式来访问：

```
[<package_name>.]R.layout.<file_name>
```

6.8 使用菜单（Menu）资源

前面已经介绍过 Android 的菜单支持，前面分别介绍了如何使用 Java 代码来实现菜单和使用 XML 资源文件来定义菜单。

实际上，Android 推荐使用 XML 资源文件来定义菜单，这样将会提供更好的解耦。由于前面介绍过如何使用 XML 资源文件定义菜单，因此此处不再详细介绍菜单资源文件的内容，只是对其进行简单的归纳。

Android 菜单资源文件放在/res/menu 目录下，菜单资源的根元素通常是<menu.../>元素，<menu.../>元素无须指定任何属性。

一旦在 Android 项目中定义了菜单资源，接下来在 XML 文件中就可通过如下语法格式来访问它：

```
@[<package_name>:]menu/file_name
```

在 Java 代码中则按如下语法格式来访问：

```
[<package_name>.]R.menu.<file_name>
```

6.9 样式（Style）和主题（Theme）资源

样式和主题资源都用于对 Android 应用进行“美化”，只要充分利用 Android 应用的样式和主题资源，开发者就可以开发出各种风格的 Android 应用。

6.9.1 样式资源

如果我们经常需要对某个类型的组件指定大致相似的格式，比如字体、颜色、背景色等，如果每次都要为 View 组件重复指定这些属性，无疑会有大量的工作量，而且不利于项目后期的维护。

类似于 Word，Word 也提供了样式来管理格式：一个样式等于一组格式的集合，如果设置某段文本使用某个样式，那么该样式的所有格式将会整体应用于这段文本。Android 的样式与此类似，Android 样式也包含一组格式，为一个组件设置使用某个样式时，该样式所包含的全部格式将会应用于该组件。

提示：

一个样式相当于多个格式的集合，其他 UI 组件通过 style 属性来指定样式，这就相当于把该样式包含的所有格式同时应用于该 UI 组件。

Android 的样式资源文件也放在/res/values/目录下，样式资源文件的根元素是<resources.../>元素，该元素内可包含多个<style.../>子元素，每个<style.../>子元素定义一个样式。<style.../>元素指定如下两个属性。

- name：指定样式的名称。
- parent：指定该样式所继承的父样式。当继承某个父样式时，该样式将会获得父样式中定义的全部格式。当然，当前样式也可以覆盖父样式中指定的格式。

<style.../>元素内可包含多个<item.../>子元素，每个<item.../>子元素定义一个格式项。

例如，为应用定义如下样式资源文件。

程序清单：codes\06\6.9\StyleResTest\app\src\main\res\values\my_style.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <!-- 定义一个样式，指定字体大小、字体颜色 -->
    <style name="style1">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#00d</item>
    </style>
    <!-- 定义一个样式，继承前一个颜色 -->
    <style name="style2" parent="@style/style1">
        <item name="android:background">#ee6</item>
        <item name="android:padding">8dp</item>
        <!-- 覆盖父样式中指定的属性 -->
        <item name="android:textColor">#000</item>
    </style>
</resources>
```

上面的样式资源中定义了两个样式，其中第二个样式继承了第一个样式，而且第二个样式中的 textColor 属性覆盖了父样式中的 textColor 属性。

一旦定义了上面的样式资源之后，接下来就可以在 XML 资源中按如下语法格式来使用样式了：

```
@[<package_name>:]style/file_name
```

下面是该示例中的界面布局文件，该布局文件中包含两个文本框，这两个文本框分别使用两个样式。

程序清单：codes\06\6.9\StyleResTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 指定使用 style1 的样式 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/style1"
    style="@style/style1"/>
<!-- 指定使用 style2 的样式 -->
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/style2"
    style="@style/style2"/>
</LinearLayout>
```

上面的界面布局文件中并未为两个文本框指定任何格式，只是为它们分别指定了使用 style1、style2 的样式，这两个样式包含的格式就会应用到这两个文本框。运行上面的程序，将看到如图 6.10 所示的界面。

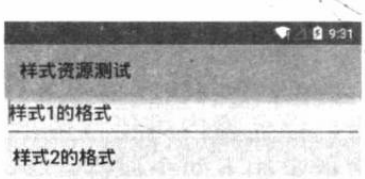


图 6.10 使用样式资源

6.9.2 主题资源

与样式资源非常相似，主题资源的 XML 文件通常也放在/res/values/目录下，主题资源的 XML 文件同样以<resources.../>元素作为根元素，同样使用<style.../>元素来定义主题。

主题与样式的区别主要体现在：

- 主题不能作用于单个的 View 组件，主体应该对整个应用中的所有 Activity 起作用，或对指定的 Activity 起作用。
- 主题定义的格式应该是改变窗口外观的格式，例如窗口标题、窗口边框等。

实例：给所有窗口添加边框、背景

下面通过一个实例来介绍主题资源的用法。为了给所有窗口都添加边框、背景，先在/res/values/my_ style.xml 文件中增加一个主题，定义主体的<style.../>片段如下：

```
<style name="CrazyTheme">
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowFullscreen">true</item>
    <item name="android:windowFrame">@drawable/window_border</item>
    <item name="android:windowBackground">@drawable/star</item>
</style>
```

上面的主题定义中使用了两个 Drawable 资源，其中@drawable/star 是一张图片；@drawable/window_border 是一个 ShapeDrawable 资源，该资源对应的 XML 文件代码如下。

程序清单：codes\06\6.9\StyleResTest\app\src\main\res\drawable\window_border.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- 设置填充颜色 -->
    <solid android:color="#0fff"/>
    <!-- 设置四周的内边距 -->
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <!-- 设置边框 -->
    <stroke android:width="10dip" android:color="#f00" />
</shape>
```

定义了上面主题之后，接下来即可在 Java 代码中使用该主题了，例如如下代码：

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

大部分时候，在 AndroidManifest.xml 文件中对指定应用、指定 Activity 应用主题更加简单。如果我们想让应用中全部窗口使用该主题，那么只要为<application.../>元素添加 android:theme 属性即可。属性值是一个主题的名字，如以下代码所示：

```
<application android:theme="@style/CrazyitTheme">
...
</application>
```

如果你只是想让程序中的某个 Activity 拥有这个主题，那么可以修改<activity.../>元素，同样通过 android:theme 指定主题即可。

本应用在 AndroidManifest.xml 文件的<application.../>元素中添加了 android:theme= "@style/CrazyTheme"属性，运行程序可以看到如图 6.11 所示的界面。

图 6.11 使用主题资源

从图6.11所示的效果可以看出，该窗口没有标题，窗口背景也被改变了，窗口全屏显示……这些都是自定义主题控制的。

提示：

可能会有读者觉得窗口边框弄得这么粗，显得很难看，其实读者可以自行控制。笔者之所以弄得这么粗，是为了让读者看到窗口边框的效果。

Android中提供了几种内置的主题资源，这些主题通过查询Android.R.style类可以看到。例如前面介绍的对话框风格的窗口，我们只要采用如下代码来定义某个Activity即可。

```
<activity android:theme="@android:style/Theme.Dialog">
...
</activtiy>
```

与样式类似的是，Android主题同样支持继承。如果开发过程中还想利用某个主题，但需要对它进行局部修改，则可通过继承系统主题来实现自定义主题。例如如下代码片段：

```
<style name="CrazyTheme" parent="@android:style/Theme.Dialog">
...
</activtiy>
```

上面定义的CrazyTheme主题继承了android.R.style.Theme.Dialog主题，那么接下来在该<style.../>元素中添加的<item.../>子元素就可覆盖系统主题的部分属性了。

6.9.3 Android 5.0新增的Material主题

Android 5.0新增了Material设计的主题，开发者只要将App的主题设为android:Theme.Material.Xxx（后面Xxx可能要根据实际选择）即可启用Material主题。

通过使用Material主题可以让App具有更自然的动态效果和过渡风格，Material Design支持包括以下几个方面。

- 系统自带Material Design主题。
- 允许通过View的setElevation()或setTranslationZ()方法让组件在屏幕上浮起来。
- 当用户点击按钮、复选框和其他可触碰组件时，将会自动显示水波动画。
- 具有Material设计风格的动画和Activity过渡效果。

实际上，前面介绍的绝大部分示例都使用了Material主题，只有第2章极个别的示例为了示范某些特殊组件的用法才改变了应用的主题，读者可通过前面示例来感受Android 5.0的Material主题的效果。

6.10 属性（Attribute）资源

前面已经介绍过自定义View组件的开发，自定义View组件与Android系统提供的View组件一样，既可在Java代码中使用，也可在XML界面布局代码中使用。

当在XML布局文件中使用Android系统提供的View组件时，开发者可以指定多个属性，这些属性可以很好地控制View组件的外观行为。如果用户开发的自定义View组件也需要指定属性，就需要属性资源的帮助了。

属性资源文件也放在/res/values目录下，属性资源文件的根元素也是<resources.../>元素，该元素包含如下两个子元素。

- **attr子元素：**定义一个属性。

➢ **declare-styleable 子元素**：定义一个 styleable 对象，每个 styleable 对象就是一组 attr 属性的集合。

当我们使用属性资源文件定义了属性之后，接下来就可以在自定义组件的构造器中通过 AttributeSet 对象来获取这些属性了。

例如，我们想开发一个默认带动画效果的图片，该图片显示时，自动从全透明变到完全不透明，为此需要开发一个自定义组件，但这个自定义组件需要指定一个额外的 duration 属性，该属性控制动画的持续时间。

为了在自定义组件中使用 duration 属性，需要先定义如下属性资源文件。

程序清单：codes\06\6.10\AttrResTest\app\src\main\res\values\attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 定义一个属性 -->
    <attr name="duration">
    </attr>
    <!-- 定义一个 styleable 对象来组合多个属性 -->
    <declare-styleable name="AlphaImageView">
        <attr name="duration"/>
    </declare-styleable>
</resources>
```

上面的属性资源文件定义了该属性之后，至于到底在哪个 View 组件中使用该属性，该属性到底能发挥什么作用，就不归属性资源文件管了。属性资源文件所定义的属性到底可以发挥什么作用，取决于自定义组件的代码实现。

例如如下自定义的 AlphaImageView，它获取了定义该组件所指定的 duration 属性之后，根据该属性来控制图片透明度的改变。程序代码如下。

提示：

在属性资源文件中定义<declare-styleable.../>元素时，也可在其内部直接使用<attr.../>定义属性，使用<attr.../>时指定一个 format 属性即可，例如上面我们可以指定<attr name="duration" format="integer"/>。

程序清单：codes\06\6.10\AttrResTest\app\src\main\java\org\crazyit\res\AlphaImageView.java

```
public class AlphaImageView extends ImageView
{
    // 图片透明度每次改变的大小
    private int alphaDelta = 0;
    // 记录图片当前的透明度
    private int curAlpha = 0;
    // 每隔多少毫秒透明度改变一次
    private final int SPEED = 300;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                // 每次增加 curAlpha 的值
                curAlpha += alphaDelta;
                if (curAlpha > 255) curAlpha = 255;
                // 修改该 ImageView 的透明度
                AlphaImageView.this.setAlpha(curAlpha);
            }
```

```
        }
    };
    public AlphaImageView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        TypedArray typedArray = context.obtainStyledAttributes(attrs,
            R.styleable.AlphaImageView);
        // 获取 duration 参数
        int duration = typedArray
            .getInt(R.styleable.AlphaImageView_duration, 0);
        // 计算图片透明度每次改变的大小
        alphaDelta = 255 * SPEED / duration;
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        this.setImageAlpha(curAlpha);
        super.onDraw(canvas);
        final Timer timer = new Timer();
        // 按固定间隔发送消息，通知系统改变图片的透明度
        timer.schedule(new TimerTask()
        {
            @Override
            public void run()
            {
                Message msg = new Message();
                msg.what = 0x123;
                if (curAlpha >= 255)
                {
                    timer.cancel();
                }
                else
                {
                    handler.sendMessage(msg);
                }
            }
        }, 0, SPEED);
    }
}
```

上面程序中的粗体字代码用于获取定义 AlphaImageView 时指定的 duration 属性，并根据该属性计算图片透明度的变化幅度，接着程序重写了 ImageView 的 onDraw(Canvas canvas)方法，该方法启动了一个任务调度来控制图片透明度的改变。

上面的粗体字代码中的 R.styleable.AlphaImageView、R.styleable.AlphaImageView_duration 都是 Android SDK 根据属性资源文件自动生成的。

接下来在界面布局文件中使用 AlphaImageView 时可以为它指定一个 duration 属性，注意该属性位于“http://schemas.android.com/apk/res/ + 项目子包”命名空间下，例如本应用的包名为 org.crazyit.res，那么 duration 属性就位于“http://schemas.android.com/apk/res/org.crazyit.res”命名空间下。

下面是该应用的界面布局文件的代码。

程序清单：codes\06\6.10\AttrResTest\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:crazyit="http://schemas.android.com/apk/res/org.crazyit.res"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 使用自定义组件，并指定属性资源文件中定义的属性 -->
```

```
    <org.crazyit.res.AlphaImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/javaee"
        crazyit:duration="60000"/>
    </LinearLayout>
```

上面程序中的第一行粗体字代码用于导入 http://schemas.android.com/apk/res/org.crazyit.res 命名空间，并指定该命名空间对应的短名前缀为 crazyit；第二行粗体字代码用于为 AlphaImageView 组件指定自定义属性 duration 的属性值为 60000。

主程序无须做任何特殊的控制，只要简单地加载并显示上面的界面布局文件，运行该程序时即可看到该图片从无到有、慢慢显示出来的效果。

6.11 使用原始资源

除了上面介绍的各种 XML 文件、图片文件之外，Android 应用可能还需要用到大量其他类型的资源，比如声音资源等。实际上，声音对于 Android 应用非常重要，选择合适的音效可以让 Android 应用增色不少。

类似于声音文件及其他各种类型的文件，只要 Android 没有为之提供专门的支持，这种资源都被称为原始资源。Android 的原始资源可以放在如下两个地方。

- 位于/res/raw/目录下，Android SDK 会处理该目录下的原始资源，Android SDK 会在 R 清单类中为该目录下的资源生成一个索引项。
- 位于/assets/目录下，该目录下的资源是更彻底的原始资源。Android 应用需要通过 AssetManager 来管理该目录下的原始资源。

Android SDK 会为位于/res/raw/目录下的资源在 R 清单类中生成一个索引项，接下来在 XML 文件中可通过如下语法格式来访问它：

```
@[<package_name>:]raw/file_name
```

在 Java 代码中则按如下语法格式来访问：

```
[<package_name>.]R.raw.<file_name>
```

通过上面的索引项，Android 应用就可以非常方便地访问/res/raw/目录下的原始资源了，至于获取原始资源之后如何处理，则完全取决于实际项目的需要。

AssetManager 是一个专门管理/assets/目录下原始资源的管理器类，AssetManager 提供了如下两个常用方法来访问 Assets 资源。

- InputStream open(String fileName)：根据文件名来获取原始资源对应的输入流。
- AssetFileDescriptor openFd(String fileName)：根据文件名来获取原始资源对应的 AssetFileDescriptor。AssetFileDescriptor 代表了一项原始资源的描述，应用程序可通过 AssetFileDescriptor 来获取原始资源。

下面的程序示范了如何使用声音。先往应用的/res/raw/目录下放入一个 bomb.mp3 文件——Android SDK 会自动处理该目录下的资源，会在 R 清单类中为它生成一个索引项：R.raw.bomb。

接下来再往/assets/目录下放入一个 shot.mp3 文件——需要通过 AssetManager 进行管理。

下面的程序中定义了两个按钮，其中一个按钮用于播放/res/raw/目录下的声音文件；另一个按钮用于播放/assets/目录下的声音文件。程序界面布局代码很简单，此处不再给出。

程序清单：codes\06\6.11\RawResTest\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    MediaPlayer mediaPlayer1 = null;
    MediaPlayer mediaPlayer2 = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 直接根据声音文件的 ID 来创建 MediaPlayer
        mediaPlayer1 = MediaPlayer.create(this, R.raw.bomb);
        // 获取该应用的 AssetManager
        AssetManager am = getAssets();
        try
        {
            // 获取指定文件对应的 AssetFileDescriptor
            AssetFileDescriptor afd = am.openFd("shot.mp3");
            mediaPlayer2 = new MediaPlayer();
            // 使用 MediaPlayer 加载指定的声音文件
            mediaPlayer2.setDataSource(afd.getFileDescriptor());
            mediaPlayer2.prepare();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        // 获取第一个按钮，并为它绑定事件监听器
        Button playRaw = (Button) findViewById(R.id.playRaw);
        playRaw.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 播放声音
                mediaPlayer1.start();
            }
        });
        // 获取第二个按钮，并为它绑定事件监听器
        Button playAsset = (Button) findViewById(R.id.playAsset);
        playAsset.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 播放声音
                mediaPlayer2.start();
            }
        });
    }
}
```

上面程序中的第一行粗体字代码用于获取/res/raw/目录下的原始资源文件；第二段粗体字代码则利用了 AssetManager 来获取/assets/目录下的原始资源文件。

上面的程序中利用了 MediaPlayer 来播放声音，MediaPlayer 是 Android 提供的一个播放声音的类，本书后面还有关于该类更详细的介绍。

6.12 国际化和资源自适应

全球化的 Internet 需要全球化的软件。全球化的软件即意味着同一种版本的产品能够容易地适用于不同地区的市场。引入国际化的目的是为了提供自适应、更友好的用户界面，并不需要改变程序的逻辑功能。国际化的英文单词是 Internationalization，因为这个单词太长了，有时也简称 I18N，其中 I 是这个单词的第一个字母，18 表示中间省略的字母个数，而 N 代表这个单词的最后一个字母。

一个国际化支持很好的应用，会随着在不同区域使用而呈现出本地语言的提示。这个过程也被称为 Localization，即本地化。类似于国际化可以称为 I18N，本地化也可以称为 L10N。

Android 所采用的资源管理方式可以非常方便地实现程序国际化。

6.12.1 Java 国际化的思路

Java 程序的国际化的思路是将程序中的标签、提示等信息放在资源文件中，程序需要支持哪些国家、语言环境，就需要提供相应的资源文件。资源文件是 key-value 对，每个资源文件中的 key 是不变的，但 value 则随不同国家、语言改变。图 6.12 显示了 Java 程序国际化的思路。

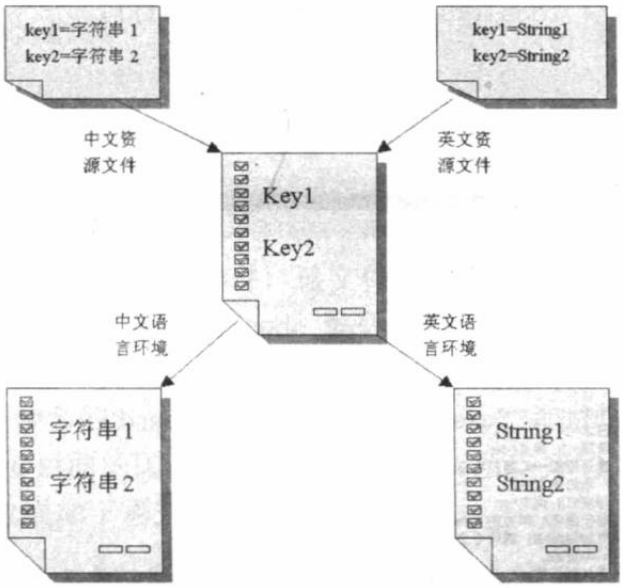


图 6.12 Java 程序国际化的思路

Java 程序的国际化主要通过如下三个类完成。

- java.util.ResourceBundle：用于加载一个国家、语言资源包。
- java.util.Locale：用于封装一个特定的国家/区域、语言环境。
- java.text.MessageFormat：用于格式化带占位符的字符串。

为了实现程序的国际化，必须先提供程序所需要的资源文件。资源文件的内容适合很多 key-value 对，其中 key 是程序使用的部分，而 value 则是程序界面的显示字符串。

资源文件的命名可以有如下三种形式。

- baseName_language_country.properties
- baseName_language.properties
- baseName.properties

其中 baseName 是资源文件的基本名，用户可以自由定义。而 language 和 country 都不可随意变化，必须是 Java 所支持的语言和国家。

6.12.2 Java 支持的国家和语言

事实上，Java 不可能支持所有的国家和语言，如需要获取 Java 所支持的国家和语言，可调用 Locale 类的 getAvailableLocales 方法获取，该方法返回一个 Locale 数组，该数组里包含了 Java 所支持的国家和语言。

下面的程序简单地示范了如何获取 Java 所支持的国家和语言。

程序清单：codes\06\6.12\LocaleList.java

```
public class LocaleList
{
    public static void main(String[] args)
    {
        // 返回 Java 所支持的全部国家和语言的数组
        Locale[] localeList = Locale.getAvailableLocales();
        // 遍历数组的每个元素，依次获取所支持的国家和语言
        for (int i = 0; i < localeList.length ; i++ )
        {
            // 打印出所支持的国家和语言
            System.out.println(localeList[i].getDisplayCountry()
                + "=" + localeList[i].getCountry()
                + " " + localeList[i].getDisplayLanguage()
                + "=" + localeList[i].getLanguage());
        }
    }
}
```

程序的运行结果如图 6.13 所示。

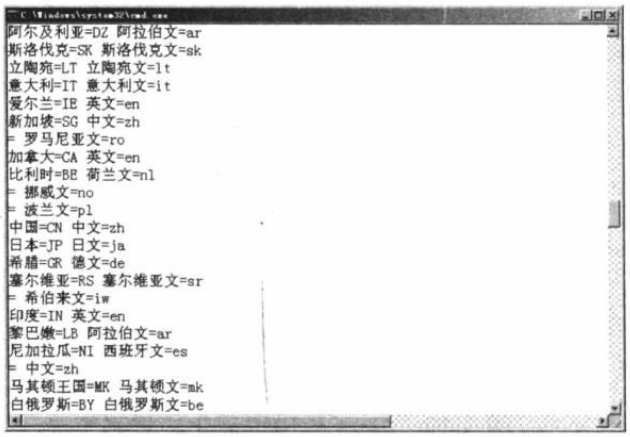


图 6.13 Java 国际化所支持的国家和语言

通过该程序，我们就可以获得 Java 程序所支持的国家/语言环境。

提示：

虽然可以通过查阅相关资料来获取 Java 语言所支持的国家/语言环境，但如果这些资料不是随手可得的，则可以通过上面的程序来获得 Java 语言所支持的国家/语言环境。

6.12.3 完成程序国际化

对于如下最简单的程序：

```
public class RawHello
{
```

```
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```

这个程序的执行结果也很简单，肯定是打印出简单的“Hello World”字符串，不管在哪里执行都不会有任何改变！为了让该程序支持国际化，则肯定不能在程序中直接输出“Hello World”字符串，这种写法直接输出一个字符串常量，永远不会有任何改变。为了让程序可以输出不同的字符串，此处绝不可使用该字符串常量。

为了让上面输出的字符串常量可以改变，我们将需要输出的各种字符串（不同国家/语言环境对应不同的字符串）定义在资源包中。

我们为上面程序提供如下两个文件。

第一个文件 mess_zh_CN.properties，该文件的内容为：

```
#资源文件的内容是 key-value 对
hello=您好！
```

第二个文件 mess_en_US.properties，该文件的内容为：

```
#资源文件的内容是 key-value 对
hello=Welcome You!
```

对于包含非西欧字符的资源文件，Java 提供了一个工具来处理该文件：native2ascii，这个工具可以在%JAVA_HOME%/bin 路径下找到。使用该工具的语法格式如下：

```
native2ascii 源资源文件 目的资源文件
```

如果我们在命令行窗口输入如下命令：

```
#使用 native2ascii 命令处理 mess_zh_CN.properties 文件，生成 aa.properties 文件
native2ascii mess_zh_CN.properties aa.properties
```

上面的命令将生成一个 aa.properties 文件，该文件才是我们需要的资源文件，该文件看上去包含很多乱码，其实是非西欧字符的 Unicode 编码方式，这完全正常。将该文件重命名为 mess_zh_CN.properties 即可。

我们看到这两份文件文件名的 baseName 是相同的：mess。前面已经介绍了资源文件的三种命名方式，其中 baseName 后面的国家、语言必须是 Java 所支持的国家、语言组合。

将上面的 Java 程序修改成如下形式。

程序清单：codes\06\6.12\Hello.java

```
public class Hello
{
    public static void main(String[] args)
    {
        // 取得系统默认的国家/语言环境
        Locale myLocale = Locale.getDefault(Locale.Category.FORMAT);
        // 根据指定国家/语言环境加载资源文件
        ResourceBundle bundle = ResourceBundle
            .getBundle("mess" , myLocale);
        // 输出从资源文件中取得的消息
        System.out.println(bundle.getString("hello"));
    }
}
```

上面程序中的打印语句不再是直接打印“Hello World”字符串，而是打印从资源包中读取的信息。如果在中文环境下运行该程序，将打印出“您好！”；如果我们在“控制面板”中将机

器的语言环境设置成美国，然后再次运行该程序，将打印出“Welcome You!”字符串。

通过上面的简单程序，我们可以体会到 Java 程序的国际化是多么简单！

从上面的程序可以看出，如果我们希望程序完成国际化，只需要将不同国家/语言（Locale）的提示信息分别以不同文件存放即可。例如，简体中文的语言资源文件就是 Xxx_zh_CN.properties 文件，而美式英语的语言资源文件就是 Xxx_en_US.properties 文件。

Java 程序国际化的关键类是 ResourceBundle，它有一个静态方法：getBundle(String baseName , Locale locale)。该方法将根据 Locale 加载资源文件，而 Locale 封装了一个国家、语言，例如简体中文的环境可以用简体中文的 Locale 代表，美式英语的环境可以用美式英语的 Locale 代表。

从上面的资源文件的命名中可以看出，不同国家、语言环境的资源文件的 baseName 是相同的，即 baseName 为 mess 的资源文件有很多个，不同国家、语言环境对应不同的资源文件。

例如，通过如下代码来加载资源文件：

```
// 根据指定国家/语言环境加载资源文件
ResourceBundle bundle = ResourceBundle.getBundle("mess" , myLocale);
```

上面的代码将会加载 baseName 为 mess 的系列资源文件的其中之一，到底加载其中的哪一个，则取决于 myLocale，对于简体中文的 Locale，则加载 mess_zh_CN.properties 文件。

一旦加载了该文件，该资源文件的内容就是多个 key-value 对，程序就根据 key 来获取指定信息，例如获取 key 为 hello 的消息，该消息是“您好！”——这就是 Java 程序国际化的过程。

对于美式英语的 Locale，则加载 mess_en_US.properties 文件，该文件中的 key 为 hello 的消息是“Welcome You!”。

Java 程序国际化的关键类是 ResourceBundle，ResourceBundle 根据不同的 Loacle 加载语言资源文件，再根据指定 key 取得已加载语言资源文件中的字符串即可。

6.12.4 为 Android 应用提供国际化资源

为 Android 程序提供国际化资源更加方便——因为 Android 本身就采用了 XML 资源文件来管理所有字符串消息，只要为各消息提供不同国家、语言对应的内容即可。

通过前面的介绍我们知道，Android 应用使用 res\values\目录下的资源文件来保存程序中用到的字符串消息，为了给这些消息提供不同国家、语言的版本，开发者需要为 values 目录添加几个不同的语言国家版本。不同 values 文件夹的命名方式为：

```
values-语言代码-r 国家代码
```

例如，如果希望下面的应用支持简体中文、美式英语两种环境，则需要在 res 目录下添加 values-zh-rCN、values-en-rUS 两个目录。

如果希望应用程序的图片也能随国家、语言环境改变，那么还需要为 drawable 目录添加几个不同的语言国家版本。不同 drawable 文件夹的命名方式为：

```
drawable-语言代码-r 国家代码
```

提示：

如果还需要为 drawable 目录按分辨率提供文件夹，则可以在后面追加分辨率后缀，例如 drawable-zh-rCN-mdpi、drawable-zh-rCN-hdpi、drawable-zh-rCN-xhdpi、drawable-en-rUS-mdpi、drawable-en-rUS-hdpi、drawable-en-rUS-xhdpi 等。

下面的程序分别为 values 目录、drawable 目录创建了简体中文、美式英语两个版本，如图 6.14 所示。

图 6.14 国际化资源文件

在\res\drawable-zh-rCN\、\res\drawable-en-rUS\目录下分别添加 logo.png 图片，这两个图片并不相同，一个是简体中文环境的图片，一个是美式英语环境的图片。

在\res\values-en-rUS\、\res\values-zh-rCN\目录下分别创建 strings.xml 文件，很明显，该文件中存放的就是字符串资源。

其中\res\values-en-rUS\目录下的 strings.xml 是美式英语的字符串资源文件，而\res\values-zh-rCN\目录下的 strings.xml 是简体中文的字符串资源文件。\res\values-en-rUS\目录下的 strings.xml 文件内容如下：

```
<resources>
    <string name="ok">OK</string>
    <string name="cancel">Cancel</string>
    <string name="msg">Hello , Android!</string>
</resources>
```

\res\values-zh-rCN\目录下的 strings.xml 文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="ok">确定</string>
    <string name="cancel">取消</string>
    <string name="msg">你好啊，可爱的小机器人！</string>
</resources>
```

正如从 Java 国际化中所看到的，不同语言的国际化资源文件中所有消息的 key 是相同的，在不同国家、语言环境下，消息资源 key 对应的 value 不同。

由于 Android 的消息资源本身就是采用 XML 文件来提供的，所以不需要额外的处理。接下来就可以在 Android 应用中使用这些国际化消息资源了。

6.12.5 国际化 Android 应用

Android 的设计本身就是国际化的，当开发者在 XML 界面布局文件、Java 代码中加载字符串资源时，Android 的国际化机制就已经起作用了。

对于下面的界面布局文件。

程序清单：codes\06\6.12\I18N\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<TextView
    android:id="@+id/show"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:lines="2"
    android:gravity="top"/>
<ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/logo"/>
<LinearLayout
    android:orientation="horizontal"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal">
<!-- 两个按钮的文本都是通过消息资源指定的 -->
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/ok"/>
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/cancel"/>
</LinearLayout>
</LinearLayout>
```

上面三行粗体字代码与前面的界面布局代码没有任何改变，它本身就没有把字符串内容“写死”在布局文件中，它本身就是去加载资源文件中的字符串值，此时 Android 的国际化机制就可发挥作用了——如果 Android 是简体中文环境，就加载\res\values-zh-rCN\strings. xml 文件中的字符串资源、加载\res\drawable-zh-rCN 目录下的 Drawable 资源；如果是美式英语环境，就加载\res\values-en-rUS\strings.xml 文件中的字符串资源、加载\res\drawable-en-rUS 目录下的 Drawable 资源。

与此类似的是，在 Java 代码中编程时，程序同样可以根据资源 ID 设置字符串内容，而不是以硬编码的方式设置为固定的字符串内容。例如如下程序代码。

程序清单：codes\06\6.12\I18N\app\src\main\java\org\crazyit\res\MainActivity.java

```
public class MainActivity extends Activity
{
    TextView tvShow;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvShow = (TextView) findViewById(R.id.show);
        // 设置文本框所显示的文本
        tvShow.setText(R.string.msg);
    }
}
```

图 6.15　美式英语环境

图 6.16　简体中文环境

上面的程序为 tvShow 设置文本内容时并未以硬编码的方式设置为固定的字符串内容，而是设置为消息资源的 name——相当于国际化消息的 key。类似的，Android 系统的国际化机制同样可以发挥作用：如果 Android 是简体中文环境，就加载\res\values-zh-rCN\strings.xml 文件中的字符串资源；如果是美式英语环境，就加载\res\values-en-rUS\strings.xml 文件中的字符串资源。

将手机设为美式英语环境（通过 Android 系统的 Settings → Language&keyboard settings → Select language → English (United States)进行设置），运行该程序即可看到如图 6.15 所示的界面。

将手机设为简体中文环境（通过 Android 系统的 Settings →Language&input→language→中文(简体)进行设置），运行该程序即可看到如图 6.16 所示的界面。

从图 6.15、图 6.16 可以看出，两个程序是同一个程序，只是由于它们的运行环境不同，Android 系统会控制程序加载不同的资源文件，因此程序界面上的图片、字符串就完全不同了。

提示：

上面程序中的标题并未改变，这是因为这个标题对应的字符串资源只有一份，而且是保存在 values 目录下的，不管是简体中文环境，还是美式英语环境，系统总是加载这份资源文件，因此程序标题是固定的。如果需要对程序标题也进行国际化，不难，只要为程序标题对应的字符串消息名（app_name）分别提供美式英语、简体中文的消息资源即可。

6.13 自适应不同屏幕的资源

开发 Android 应用有一个比较烦人的地方是：Android 设备的屏幕尺寸、分辨率差别非常大，而开发者开发的 Android 应用总希望能在所有 Android 设备上运行，因此开发 Android 应用就需要考虑不同屏幕的适应性问题。

提示：

相比之下，开发 iOS 应用要更简单，因为 iOS 只有几种特定的手机和平板电脑两种设备，它们的屏幕尺寸、分辨率都是固定的，因此需要考虑的设备更少。

前面已经提到，Android 默认把 drawable 目录（存放图片等 Drawable 资源的目录）分为 drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi 这 4 个子目录，这 4 个子目录就是为不同分辨率准备的图片。

通常来说，屏幕资源需要考虑如下两个方面。

- **屏幕尺寸：**屏幕尺寸可分为 small（小屏幕）、normal（中等屏幕）、large（大屏幕）、xlarge（超大屏幕）4 种。
- **屏幕分辨率：**屏幕分辨率可分为 ldpi（低分辨率）、mdpi（中等分辨率）、hdpi（高分辨率）、xhdpi（超高分辨率）、xxhdpi（超超高分辨率）5 种。
- **屏幕方向：**屏幕方向可分为 land（横屏）和 port（竖屏）两种。

图 6.17 显示了不同屏幕尺寸、不同分辨率对应的通用说法。

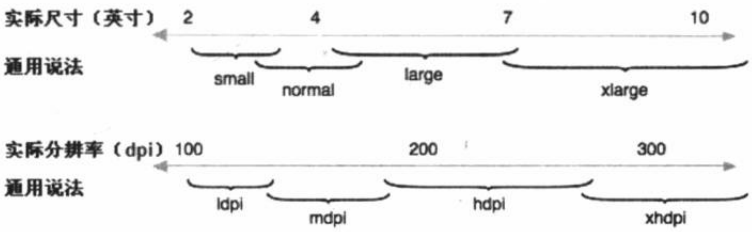


图 6.17 不同尺寸、不同分辨率的屏幕

当我们为不同尺寸的屏幕设置用户界面时，每种用户界面总有一个最低的屏幕尺寸要求（低于该屏幕尺寸就没法运行），上面这些通用说法中屏幕尺寸总有一个最低分辨率，该最低分辨率是以 dp 为单位 的，因此我们在定义界面布局时应该尽量考虑使用 dp 为单位。

下面是上面 4 种屏幕尺寸所需的最低尺寸。

- xlarge 屏幕尺寸至少需要 960dp×720dp。
- large 屏幕尺寸至少需要 640dp×480dp。
- normal 屏幕尺寸至少需要 470dp×320dp。

- small 屏幕尺寸至少需要 426dp×320dp。

通过上面的介绍不难发现，为了提供自适应不同屏幕的资源，最简单的做法就是为不同屏幕尺寸、不同屏幕分辨率提供相应的布局资源、Drawable 资源。

- **屏幕分辨率**：可以为 drawable 目录增加后缀 ldpi（低分辨率）、mdpi（中等分辨率）、hdpi（高分辨率）、xhdpi（超高分辨率），分别为不同分辨率的屏幕提供资源。
- **屏幕尺寸**：可以为 layout、values 等目录增加后缀 small、normal、large 和 xlarge，分别为不同尺寸的屏幕提供相应资源。

注意：

从 Android 3.2 开始，Android 建议直接使用真实的屏幕尺寸来定义屏幕尺寸。Android 3.2 支持在 layout、values 目录后添加 sw<*N*>dp（屏幕尺寸至少宽 *N* 个 dp 才能使用该资源）、w<*N*>dp（屏幕尺寸可用宽度为 *N* 个 dp 可使用该资源）、h<*N*>dp（屏幕尺寸可用高度为 *N* 个 dp 才能使用该资源）。例如可指定 layout-sw600dp，表明该设备屏幕的宽度大于或等于 600 个 dp 时使用该目录下的布局资源。

- **屏幕方向**：可以为 layout、values 等目录增加后缀 land 和 port，分别为横屏、竖屏提供相应的资源。

下面的示例在 layout 目录下定义了一个界面布局文件。

程序清单：codes\06\6.13\DpiTest\app\src\main\res\layout\main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/a"/>
</RelativeLayout>
```

上面的界面布局文件指定显示@drawable/a 对应的图片资源，在 hdpi 设备上运行程序时，系统将会使用/res/drawable-hdpi 目录下的图片。运行该程序，将看到如图 6.18 所示的界面。

在 xhdpi 设备上运行程序时（比如 768×1280 分辨率、320dpi），系统将会使用/res/drawable-xhdpi/目录下的图片。运行该程序，将看到如图 6.19 所示的界面。

图 6.18 hdpi 屏幕使用 drawable-hdpi 目录下的图片

图 6.19 xhdpi 屏幕使用 drawable-xhdpi 目录下的图片

下面的示例提供了两份布局文件，其中一份放在 layout-normal 目录下，一份放在 layout-large 目录下。layout-normal 目录下的布局文件代码如下。

程序清单：codes\06\6.13\ScreenSizeTest\app\src\main\res\layout-normal\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="match_parent"
    android:layout_height="match_parent">
    <Button
        android:text="第一个按钮"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:text="第二个按钮"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:text="第三个按钮"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

layout-large 目录下的布局文件代码如下。

程序清单：codes\06\6.13\ScreenSizeTest\app\src\main\res\layout-large\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="match_parent"
    android:layout_height="match_parent">
    <Button
        android:text="第一个按钮"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:text="第二个按钮"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:text="第三个按钮"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

在正常尺寸的屏幕（分辨率为 768×1280、320dpi）上运行该程序，系统将会加载 layout-normal 目录下的 main.xml 布局文件。运行该程序，将看到如图 6.20 所示的界面。

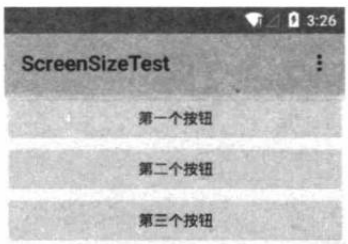


图 6.20 正常尺寸的屏幕加载 layout-normal 目录下的界面布局文件

在大尺寸的屏幕（分辨率为 720×1280、240dpi）上运行该程序，系统将会加载 layout-large

目录下的 main.xml 布局文件。运行该程序，将看到如图 6.21 所示的界面。

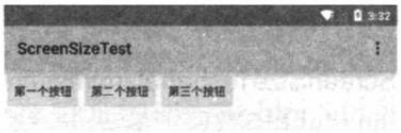


图 6.21 大尺寸的屏幕加载 layout-large 目录下的界面布局文件

可能有读者感到奇怪：720×1280 的分辨率不是比 768×1280 分辨率更低吗？怎么 Android 反而使用 layout-large 目录下的布局文件呢？这是因为手机屏幕大小不仅与分辨率有关，还与 dpi 有关。dpi 的意思是 Dots Per Inch，即每英寸包含的点数，这意味着在相同分辨率情况下，dpi 越高，屏幕反而越小；但 dpi 越高，图像显示效果就越细腻。因此 720×1280、240dpi 的屏幕其实比 768×1280、320dpi 的屏幕更大，故而 Android 在 720×1280、240dpi 的屏幕上会选择 layout-large 目录下的 main.xml 布局文件。

6.14 本章小结

本章主要介绍了 Android 应用资源的相关内容。Android 应用资源是一种非常优秀的、高解耦设计，通过使用资源文件，Android 应用可以把各种字符串、图片、颜色、界面布局等交给 XML 文件配置管理，这样就避免在 Java 代码中以硬编码的方式直接定义这些内容。学习本章需要掌握 Android 应用资源的存储方式、Android 应用资源的使用方式。除此之外，Android 应用的字符串资源、颜色资源、尺寸资源、数组资源、图片资源、各种 Drawable 资源、原始 XML 资源、布局资源、菜单资源、样式和主题资源、属性资源、原始资源等各种资源文件都需要重点掌握。

本章最后还介绍了 Android 应用的国际化。由于 Android 应用的国际化实际上是以 Java 国际化为基础的，因此本章还简单介绍了 Java 程序的国际化。这些内容都需要读者认真掌握。

CHAPTER

7

第7章 图形与图像处理

本章要点

- Android 的图形处理基础
- Bitmap 与 BitmapFactory
- 继承 View 在 Android 中绘图
- 掌握 Canvas、Paint、Path 等绘图 API
- 双缓冲机制
- 使用 Matrix 对图像进行几何变换
- 通过 drawBitmapMesh 方法扭曲图像
- 使用不同的 Shader 类渲染图形
- 逐帧动画
- 补间动画
- 属性动画
- 开发自定义补间动画
- SurfaceView 的绘图机制
- 继承 SurfaceView 开发动画

正如前面所介绍的，决定 Android 应用是否被用户接受的重要方面就是用户界面，为了提供友好的用户界面，就需要在应用中使用图片了。Android 系统提供了丰富的图片功能支持，包括处理静态图片和动画等。

Android 系统提供了 ImageView 显示普通的静态图片，也提供了 AnimationDrawable 来开发逐帧动画，还可通过 Animation 对普通图片使用补间动画。图形、图像处理不仅对 Android 系统的应用界面非常重要，而且 Android 系统上的益智类游戏、2D 游戏都需要大量的图形、图像处理。所谓游戏，本质就是提供更逼真的、能模拟某种环境的用户界面，并根据某种规则来响应用户操作。为了提供更逼真的用户界面，需要借助于图形、图像处理。

学习本章内容之后，读者应该能熟练掌握 Android 系统的图形、图像处理，这样就可以在 Android 平台上开发出俄罗斯方块、五子棋等小游戏了。本章将会通过弹球游戏、飞行游戏雏型来帮助读者掌握 Android 2D 游戏开发的入门知识。

7.1 使用简单图片

前面的 Android 应用中已经大量使用了简单图片，图片不仅可以使用 ImageView 来显示，也可作为 Button、Window 的背景。从广义的角度来看，Android 应用中的图片不仅包括*.png、*.jpg、*.gif 等各种格式的位图，也包括使用 XML 资源文件定义的各种 Drawable 对象。

7.1.1 使用 Drawable 对象

为 Android 应用增加了 Drawable 资源之后，Android SDK 会为这份资源在 R 清单文件中创建一个索引项：R.drawable.file_name。

接下来既可在 XML 资源文件中通过@drawable/file_name 访问该 Drawable 对象，也可在 Java 代码中通过 R.drawable.file_name 访问该 Drawable 对象。

需要指出的是，R.drawable.file_name 是一个 int 类型的常量，它只代表 Drawable 对象的 ID，如果 Java 程序中需要获取实际的 Drawable 对象，则可调用 Resources 的 getDrawable(int id) 方法来实现。

由于前面已经介绍了大量关于 Drawable 的示例，故此处不再给出示例。

7.1.2 Bitmap 和 BitmapFactory

Bitmap 代表一个位图，BitmapDrawable 里封装的图片就是一个 Bitmap 对象。开发者为了把一个 Bitmap 对象包装成 BitmapDrawable 对象，可以调用 BitmapDrawable 的构造器：

```
// 把一个 Bitmap 对象包装成 BitmapDrawable 对象
BitmapDrawable drawable = new BitmapDrawable(bitmap);
```

如果需要获取 BitmapDrawable 所包装的 Bitmap 对象，则可调用 BitmapDrawable 的 getBitmap()方法，如下面的代码所示：

```
// 获取 BitmapDrawable 所包装的 Bitmap 对象
Bitmap bitmap = drawable.getBitmap();
```

除此之外，Bitmap 还提供了一些静态方法来创建新的 Bitmap 对象，例如如下常用方法。

- createBitmap(Bitmap source, int x, int y, int width, int height)：从源位图 source 的指定坐标点（给定 x、y）开始，从中“挖取”宽 width、高 height 的一块出来，创建新的 Bitmap 对象。
- createScaledBitmap(Bitmap src, int dstWidth, int dstHeight, boolean filter)：对源位图

src 进行缩放，缩放成宽 dstWidth、高 dstHeight 的新位图。

- createBitmap(int width, int height, Bitmap.Config config)：创建一个宽 width、高 height 的新位图。
- createBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)：从源位图 source 的指定坐标点（给定 x、y）开始，从中“挖取”宽 width、高 height 的一块出来，创建新的 Bitmap 对象，并按 Matrix 指定的规则进行变换。

BitmapFactory 是一个工具类，它提供了大量的方法，这些方法可用于从不同的数据源来解析、创建 Bitmap 对象。BitmapFactory 包含了如下方法。

- decodeByteArray(byte[] data, int offset, int length)：从指定字节数组的 offset 位置开始，将长度为 length 的字节数据解析成 Bitmap 对象。
- decodeFile(String pathName)：从 pathName 指定的文件中解析、创建 Bitmap 对象。
- decodeFileDescriptor(FileDescriptor fd)：用于从 FileDescriptor 对应的文件中解析、创建 Bitmap 对象。
- decodeResource(Resources res, int id)：用于根据给定的资源 ID 从指定资源中解析、创建 Bitmap 对象。
- decodeStream(InputStream is)：用于从指定输入流中解析、创建 Bitmap 对象。

大部分时候，我们只要把图片放在/res/drawable/目录下，就可以在程序中通过该图片对应的资源 ID 来获取封装该图片的 Drawable 对象。但由于手机系统的内存比较小，如果系统不停地去解析、创建 Bitmap 对象，可能由于前面创建 Bitmap 所占用的内存还没有回收，而导致程序运行时引发 OutOfMemory 错误。

Android 为 Bitmap 提供了两个方法来判断它是否已回收，以及强制 Bitmap 回收自己。

- boolean isRecycled()：返回该 Bitmap 对象是否已被回收。
- void recycle()：强制一个 Bitmap 对象立即回收自己。

除此之外，如果 Android 应用需要访问其他存储路径（比如 SD 卡）里的图片，那么都需要借助于 BitmapFactory 来解析、创建 Bitmap 对象。

下面开发一个查看/assets/目录下图片的图片查看器。该程序界面十分简单，只包含一个 ImageView 和一个按钮，当用户单击该按钮时程序会自动去搜寻/assets/目录下的下一张图片。此处不再给出界面布局代码，该程序的代码如下。

程序清单：codes\07\7.1\BitmapTest\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    String[] images = null;
    AssetManager assets = null;
    int currentImg = 0;
    ImageView image;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        image = (ImageView) findViewById(R.id.image);
        try
        {
            assets = getAssets();
            // 获取/assets/目录下的所有文件
            images = assets.list("");
```

```
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        // 获取 next 按钮
        final Button next = (Button) findViewById(R.id.next);
        // 为 next 按钮绑定事件监听器，该监听器将会查看下一张图片
        next.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View sources)
            {
                // 如果发生数组越界
                if (currentImg >= images.length)
                {
                    currentImg = 0;
                }
                // 找到下一个图片文件
                while (!images[currentImg].endsWith(".png")
                        && !images[currentImg].endsWith(".jpg")
                        && !images[currentImg].endsWith(".gif"))
                {
                    currentImg++;
                    // 如果已发生数组越界
                    if (currentImg >= images.length)
                    {
                        currentImg = 0;
                    }
                }
                InputStream assetFile = null;
                try
                {
                    // 打开指定资源对应的输入流
                    assetFile = assets.open(images[currentImg++]);
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
                BitmapDrawable bitmapDrawable = (BitmapDrawable) image
                        .getDrawable();
                // 如果图片还未回收，先强制回收该图片
                if (bitmapDrawable != null
                    && !bitmapDrawable.getBitmap().isRecycled()) // ①
                {
                    bitmapDrawable.getBitmap().recycle();
                }
                // 改变 ImageView 显示的图片
                image.setImageBitmap(BitmapFactory
                    .decodeStream(assetFile)); // ②
            }
        });
    }
}
```

上面程序中的①号粗体字代码用于判断当前 ImageView 所显示的图片是否已被回收，如果该图片还未回收，则系统强制回收该图片；程序的②号粗体字代码调用了 BitmapFactory 从指定输入流解析并创建 Bitmap 对象。

7.2 绘图

除了使用已有的图片之外，Android 应用还常常需要在运行时动态地生成图片，比如一个手机游戏，游戏界面看上去丰富多彩，而且可以随着用户动作而动态改变，这就需要借助于 Android 的绘图支持了。

7.2.1 Android 绘图基础：Canvas、Paint 等

有过 Swing 编程经验的读者都知道，在 Swing 中绘图的思路是需要开发一个自定义类，该自定义类继承 JPanel，并重写 JPanel 的 paint(Graphics g)方法即可。Android 的绘图与此类似，Android 的绘图应该继承 View 组件，并重写它的 onDraw(Canvas canvas)方法即可。

重写 onDraw(Canvas canvas)方法时涉及一个绘图 API：Canvas，Canvas 代表“依附”于指定 View 的画布，它提供了如表 7.1 所示的方法来绘制各种图形。

表 7.1 Canvas 的绘制方法

方法签名	简要说明
drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)	绘制弧
drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)	在指定点绘制从源位图中“挖取”的一块
drawBitmap(Bitmap bitmap, float left, float top, Paint paint)	在指定点绘制位图
drawCircle(float cx, float cy, float radius, Paint paint)	在指定点绘制一个圆
drawLine(float startX, float startY, float stopX, float stopY, Paint paint)	绘制一条直线
drawLines(float[] pts, int offset, int count, Paint paint)	绘制多条直线
drawOval(RectF oval, Paint paint)	绘制椭圆
drawPath(Path path, Paint paint)	沿着指定 Path 绘制任意形状
drawPoint(float x, float y, Paint paint)	绘制一个点
drawPoints(float[] pts, int offset, int count, Paint paint)	绘制多个点
drawRect(float left, float top, float right, float bottom, Paint paint)	绘制矩形
drawRoundRect(RectF rect, float rx, float ry, Paint paint)	绘制圆角矩形
drawText(String text, int start, int end, Paint paint)	绘制字符串
drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint)	沿着路径绘制字符串
clipRect(float left, float top, float right, float bottom)	剪切一个矩形区域
clipRegion(Region region)	剪切指定区域

除了表 7.1 中所定义的各种方法之外，Canvas 还提供了如下方法进行坐标变换。

- rotate(float degrees, float px, float py)：对 Canvas 执行旋转变换。
- scale(float sx, float sy, float px, float py)：对 Canvas 执行缩放变换。
- skew(float sx, float sy)：对 Canvas 执行倾斜变换。
- translate(float dx, float dy)：移动 Canvas。向右移动 dx 距离（dx 为负数即向左移动）；向下移动 dy 距离（dy 为负数即向上移动）。

Canvas 提供的方法还涉及一个 API：Paint，Paint 代表 Canvas 上的画笔，因此 Paint 类主要用于设置绘制风格，包括画笔颜色、画笔笔触粗细、填充风格等。Paint 提供了如表 7.2 所示的方法。

表 7.2 Paint 的常用方法

方法签名	简要说明
setARGB(int a, int r, int g, int b)/setColor(int color)	设置颜色

续表

方法签名	简要说明
setAlpha(int a)	设置透明度
setAntiAlias(boolean aa)	设置是否抗锯齿
setColor(int color)	设置颜色
setPathEffect(PathEffect effect)	设置绘制路径时的路径效果
setShader(Shader shader)	设置画笔的填充效果
setShadowLayer(float radius, float dx, float dy, int color)	设置阴影
setStrokeWidth(float width)	设置画笔的笔触宽度
setStrokeJoin(Paint.Join join)	设置画笔转弯处的连接风格
setStyle(Paint.Style style)	设置 Paint 的填充风格
setTextAlign(Paint.Align align)	设置绘制文本时的文字对齐方式
setTextSize(float textSize)	设置绘制文本时的文字大小

在 Canvas 提供的绘制方法中还用到了一个 API：Path，Path 代表任意多条直线连接而成的任意图形，当 Canvas 根据 Path 绘制时，它可以绘制出任意的形状。

下面的程序示范了如何在 Android 应用中绘制基本的几何图形。该程序的关键在于一个自定义 View 组件，程序重写该 View 组件的 onDraw(Canvase)方法，接下来在该 Canvas 上绘制了大量几何图形。这个自定义 View 的代码如下。

程序清单：codes\07\7.2\CanvasTest\app\src\main\java\org\crazyit\image\MyView.java

```
public class MyView extends View
{
    public MyView(Context context, AttributeSet set)
    {
        super(context, set);
    }
    @Override
    // 重写该方法，进行绘图
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 把整张画布绘制成白色
        canvas.drawColor(Color.WHITE);
        Paint paint = new Paint();
        // 去锯齿
        paint.setAntiAlias(true);
        paint.setColor(Color.BLUE);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(4);
        int viewWidth = this.getWidth();
        // 绘制圆形
        canvas.drawCircle(viewWidth / 10 + 10, viewWidth / 10 + 10
            , viewWidth / 10, paint);
        // 绘制正方形
        canvas.drawRect(10 , viewWidth / 5 + 20 , viewWidth / 5 + 10
            , viewWidth * 2 / 5 + 20 , paint);
        // 绘制矩形
        canvas.drawRect(10, viewWidth * 2 / 5 + 30, viewWidth / 5 + 10
            , viewWidth / 2 + 30, paint);
        RectF re1 = new RectF(10, viewWidth / 2 + 40
            , 10 + viewWidth / 5 ,viewWidth * 3 / 5 + 40);
        // 绘制圆角矩形
        canvas.drawRoundRect(re1, 15, 15, paint);
        RectF re11 = new RectF(10, viewWidth * 3 / 5 + 50
            , 10 + viewWidth / 5 ,viewWidth * 7 / 10 + 50);
```

```
// 绘制椭圆
canvas.drawOval(re11, paint);
// 定义一个 Path 对象，封闭成一个三角形
Path path1 = new Path();
path1.moveTo(10, viewWidth * 9 / 10 + 60);
path1.lineTo(viewWidth / 5 + 10, viewWidth * 9 / 10 + 60);
path1.lineTo(viewWidth / 10 + 10, viewWidth * 7 / 10 + 60);
path1.close();
// 根据 Path 进行绘制，绘制三角形
canvas.drawPath(path1, paint);
// 定义一个 Path 对象，封闭成一个五角形
Path path2 = new Path();
path2.moveTo(10 + viewWidth / 15, viewWidth * 9 / 10 + 70);
path2.lineTo(10 + viewWidth * 2 / 15, viewWidth * 9 / 10 + 70);
path2.lineTo(10 + viewWidth / 5, viewWidth + 70);
path2.lineTo(10 + viewWidth / 10, viewWidth * 11/10 + 70);
path2.lineTo(10 , viewWidth + 70);
path2.close();
// 根据 Path 进行绘制，绘制五角形
canvas.drawPath(path2, paint);
// ----------设置填充风格后绘制----------
paint.setStyle(Paint.Style.FILL);
paint.setColor(Color.RED);
// 绘制圆形
canvas.drawCircle(viewWidth * 3 / 10 + 20, viewWidth / 10 + 10
    , viewWidth / 10, paint);
// 绘制正方形
canvas.drawRect(viewWidth / 5 + 20 , viewWidth / 5 + 20
    , viewWidth * 2 / 5 + 20 , viewWidth * 2 / 5 + 20 , paint);
// 绘制矩形
canvas.drawRect(viewWidth / 5 + 20, viewWidth * 2 / 5 + 30
    , viewWidth * 2 / 5 + 20 , viewWidth / 2 + 30, paint);
RectF re2 = new RectF(viewWidth / 5 + 20, viewWidth / 2 + 40
    , 20 + viewWidth * 2 / 5 ,viewWidth * 3 / 5 + 40);
// 绘制圆角矩形
canvas.drawRoundRect(re2, 15, 15, paint);
RectF re21 = new RectF(20 + viewWidth / 5, viewWidth * 3 / 5 + 50
    , 20 + viewWidth * 2 / 5 ,viewWidth * 7 / 10 + 50);
// 绘制椭圆
canvas.drawOval(re21, paint);
// 定义一个 Path 对象，封闭成一个三角形
Path path3 = new Path();
path3.moveTo(20 + viewWidth / 5, viewWidth * 9 / 10 + 60);
path3.lineTo(viewWidth * 2 / 5 + 20, viewWidth * 9 / 10 + 60);
path3.lineTo(viewWidth * 3 / 10 + 20, viewWidth * 7 / 10 + 60);
path3.close();
// 根据 Path 进行绘制，绘制三角形
canvas.drawPath(path3, paint);
// 定义一个 Path 对象，封闭成一个五角形
Path path4 = new Path();
path4.moveTo(20 + viewWidth *4 / 15, viewWidth * 9 / 10 + 70);
path4.lineTo(20 + viewWidth / 3, viewWidth * 9 / 10 + 70);
path4.lineTo(20 + viewWidth * 2 / 5, viewWidth + 70);
path4.lineTo(20 + viewWidth * 3 / 10, viewWidth * 11/10 + 70);
path4.lineTo(20 + viewWidth / 5 , viewWidth + 70);
path4.close();
// 根据 Path 进行绘制，绘制五角形
canvas.drawPath(path4, paint);
// ----------设置渐变器后绘制----------
// 为 Paint 设置渐变器
Shader mShader = new LinearGradient(0, 0, 40, 60
    , new int[] {Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW }
```

```
            , null , Shader.TileMode.REPEAT);
        paint.setShader(mShader);
        // 设置阴影
        paint.setShadowLayer(25 , 20 , 20 , Color.GRAY);
        // 绘制圆形
        canvas.drawCircle(viewWidth / 2 + 30, viewWidth / 10 + 10
            , viewWidth / 10, paint);
        // 绘制正方形
        canvas.drawRect(viewWidth * 2 / 5 + 30 , viewWidth / 5 + 20
            , viewWidth * 3 / 5 + 30 , viewWidth * 2 / 5 + 20 , paint);
        // 绘制矩形
        canvas.drawRect(viewWidth * 2 / 5 + 30, viewWidth * 2 / 5 + 30
            , viewWidth * 3 / 5 + 30 , viewWidth / 2 + 30, paint);
        RectF re3 = new RectF(viewWidth * 2 / 5 + 30, viewWidth / 2 + 40
            , 30 + viewWidth * 3 / 5 ,viewWidth * 3 / 5 + 40);
        // 绘制圆角矩形
        canvas.drawRoundRect(re3, 15, 15, paint);
        RectF re31 = new RectF(30 + viewWidth *2 / 5, viewWidth * 3 / 5 + 50
            , 30 + viewWidth * 3 / 5 ,viewWidth * 7 / 10 + 50);
        // 绘制椭圆
        canvas.drawOval(re31, paint);
        // 定义一个 Path 对象，封闭成一个三角形
        Path path5 = new Path();
        path5.moveTo(30 + viewWidth * 2/ 5, viewWidth * 9 / 10 + 60);
        path5.lineTo(viewWidth * 3 / 5 + 30, viewWidth * 9 / 10 + 60);
        path5.lineTo(viewWidth / 2 + 30, viewWidth * 7 / 10 + 60);
        path5.close();
        // 根据 Path 进行绘制，绘制三角形
        canvas.drawPath(path5, paint);
        // 定义一个 Path 对象，封闭成一个五角形
        Path path6 = new Path();
        path6.moveTo(30 + viewWidth * 7 / 15, viewWidth * 9 / 10 + 70);
        path6.lineTo(30 + viewWidth * 8 / 15, viewWidth * 9 / 10 + 70);
        path6.lineTo(30 + viewWidth * 3 / 5, viewWidth + 70);
        path6.lineTo(30 + viewWidth / 2, viewWidth * 11/10 + 70);
        path6.lineTo(30 + viewWidth * 2 / 5 , viewWidth + 70);
        path6.close();
        // 根据 Path 进行绘制，绘制五角形
        canvas.drawPath(path6, paint);
        // ----------设置字符大小后绘制----------
        paint.setTextSize(48);
        paint.setShader(null);
        // 绘制 7 个字符串
        canvas.drawText(getResources().getString(R.string.circle)
            , 60 + viewWidth * 3 / 5, viewWidth / 10 + 10, paint);
        canvas.drawText(getResources().getString(R.string.square)
            , 60 + viewWidth * 3 / 5, viewWidth * 3 / 10 + 20, paint);
        canvas.drawText(getResources().getString(R.string.rect)
            , 60 + viewWidth * 3 / 5, viewWidth * 1 / 2 + 20, paint);
        canvas.drawText(getResources().getString(R.string.round_rect)
            , 60 + viewWidth * 3 / 5, viewWidth * 3 / 5 + 30, paint);
        canvas.drawText(getResources().getString(R.string.oval)
            , 60 + viewWidth * 3 / 5, viewWidth * 7 / 10 + 30, paint);
        canvas.drawText(getResources().getString(R.string.triangle)
            , 60 + viewWidth * 3 / 5, viewWidth * 9 / 10 + 30, paint);
        canvas.drawText(getResources().getString(R.string.pentagon)
            , 60 + viewWidth * 3 / 5, viewWidth * 11 / 10 + 30, paint);
    }
}
```

上面的程序中大量调用了 Canvas 的方法来绘制几何图形，而且程序的第二段粗体字代码还为 Paint 画笔设置了使用渐变、阴影，因此接下来绘制的几何图形将采用渐变填充，而且具

有阴影。使用一个 Activity 来显示上面的 MyView 类，运行程序将看到如图 7.1 所示的效果。

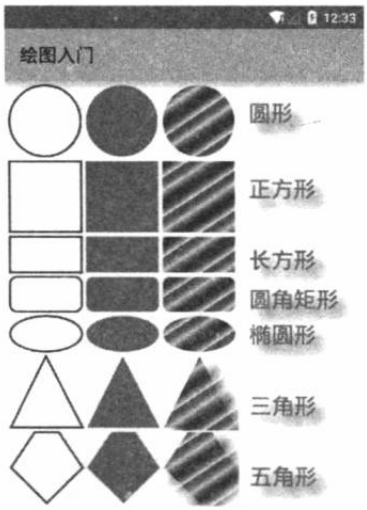


图 7.1 Android 绘图入门

Android 的 Canvas 不仅可以绘制这种简单的几何图形，还可以直接将一个 Bitmap 绘制到画布上，这样就给了开发者巨大的灵活性，只要前期美工把应用程序所需的图片制作出来，后期开发时把这些图片绘制到 Canvas 上即可。

7.2.2 Path 类

从上面的程序可以看出，Android 提供的 Path 是一个非常有用的类，它可以预先在 View 上将 *N* 个点连成一条“路径”，然后调用 Canvas 的 drawPath(path, paint)方法即可沿着路径绘制图形。实际上 Android 还为路径绘制提供了 PathEffect 来定义绘制效果，PathEffect 包含了如下子类（每个子类代表一种绘制效果）。

- ComposePathEffect
- CornerPathEffect
- DashPathEffect
- DiscretePathEffect
- PathDashPathEffect
- SumPathEffect

这些绘制效果使用语言来表述总显得有点空洞，下面通过一个程序来让读者理解这些绘制效果。该程序绘制 7 条路径，分别示范了不使用效果和使用上面 6 种效果的效果。

程序清单：codes\07\7.2\PathTest\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this));
    }
    class MyView extends View
    {
        float phase;
        PathEffect[] effects = new PathEffect[7];
        int[] colors;
        private Paint paint;
        Path path;
        public MyView(Context context)
        {
            super(context);
            paint = new Paint();
            paint.setStyle(Paint.Style.STROKE);
            paint.setStrokeWidth(4);
            // 创建并初始化 Path
            path = new Path();
            path.moveTo(0, 0);
            for (int i = 1; i <= 40; i++)
            {
                // 生成 40 个点，随机生成它们的 Y 坐标，并将它们连成一条 Path
                path.lineTo(i * 20, (float) Math.random() * 60);
```

```
            }
            // 初始化7个颜色
            colors = new int[] { Color.BLACK, Color.BLUE, Color.CYAN,
                Color.GREEN, Color.MAGENTA, Color.RED, Color.YELLOW };
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            // 将背景填充成白色
            canvas.drawColor(Color.WHITE);
            // ----------下面开始初始化7种路径效果----------
            // 不使用路径效果
            effects[0] = null;
            // 使用CornerPathEffect路径效果
            effects[1] = new CornerPathEffect(10);
            // 初始化DiscretePathEffect
            effects[2] = new DiscretePathEffect(3.0f, 5.0f);
            // 初始化DashPathEffect
            effects[3] = new DashPathEffect(new float[] { 20, 10, 5, 10 },
                    phase);
            // 初始化PathDashPathEffect
            Path p = new Path();
            p.addRect(0, 0, 8, 8, Path.Direction.CCW);
            effects[4] = new PathDashPathEffect(p, 12, phase,
                    PathDashPathEffect.Style.ROTATE);
            // 初始化ComposePathEffect
            effects[5] = new ComposePathEffect(effects[2], effects[4]);
            effects[6] = new SumPathEffect(effects[4], effects[3]);
            // 将画布移动到(8, 8)处开始绘制
            canvas.translate(8, 8);
            // 依次使用7种不同的路径效果、7种不同的颜色来绘制路径
            for (int i = 0; i < effects.length; i++)
            {
                paint.setPathEffect(effects[i]);
                paint.setColor(colors[i]);
                canvas.drawPath(path, paint);
                canvas.translate(0, 60);
            }
            // 改变phase值，形成动画效果
            phase += 1;
            invalidate();
        }
    }
}
```

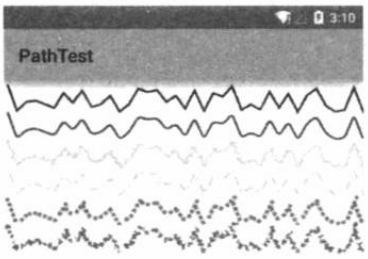


图7.2　路径效果示例

正如上面的程序中所看到的，当定义 DashPathEffect、PathDashPathEffect 时可指定一个 phase 参数，该参数用于指定路径效果的相位，当该 phase 参数改变时，绘制效果也略有变化。上面的程序不停地改变 phase 参数，并不停地重绘该 View 组件，这将产生动画效果，如图 7.2 所示。

除此之外，Android 的 Canvas 还提供了一个 drawTextOnPath (String text, Path path, float hOffset, float vOffset, Paint paint)方法，该方法可以沿着 Path 绘制文本。其中 hOffset 参数指定水平偏移，vOffset 参数指定垂直偏移。例如如下程序。

程序清单：codes\07\7.2\PathText\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
```

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(new TextView(this));
}
class TextView extends View
{
    final String DRAW_STR = "疯狂Android讲义";
    Path[] paths = new Path[3];
    Paint paint;
    public TextView(Context context)
    {
        super(context);
        paths[0] = new Path();
        paths[0].moveTo(0, 0);
        for (int i = 1; i <= 20; i++)
        {
            // 生成20个点，随机生成它们的Y坐标，并将它们连成一条Path
            paths[0].lineTo(i * 30, (float) Math.random() * 30);
        }
        paths[1] = new Path();
        RectF rectF = new RectF(0, 0, 600, 360);
        paths[1].addOval(rectF, Path.Direction.CCW);
        paths[2] = new Path();
        paths[2].addArc(rectF, 60, 180);
        // 初始化画笔
        paint = new Paint();
        paint.setAntiAlias(true);
        paint.setColor(Color.CYAN);
        paint.setStrokeWidth(1);
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        canvas.drawColor(Color.WHITE);
        canvas.translate(40, 40);
        // 设置从右边开始绘制（右对齐）
        paint.setTextAlign(Paint.Align.RIGHT);
        paint.setTextSize(20);
        // 绘制路径
        paint.setStyle(Paint.Style.STROKE);
        canvas.drawPath(paths[0], paint);
        paint.setTextSize(40);
        // 沿着路径绘制一段文本
        paint.setStyle(Paint.Style.FILL);
        canvas.drawTextOnPath(DRAW_STR, paths[0], -8, 20, paint);
        // 对Canvas进行坐标变换：画布下移60
        canvas.translate(0, 60);
        // 绘制路径
        paint.setStyle(Paint.Style.STROKE);
        canvas.drawPath(paths[1], paint);
        // 沿着路径绘制一段文本
        paint.setStyle(Paint.Style.FILL);
        canvas.drawTextOnPath(DRAW_STR, paths[1], -20, 20, paint);
        // 对Canvas进行坐标变换：画布下移360
        canvas.translate(0, 360);
        // 绘制路径
        paint.setStyle(Paint.Style.STROKE);
        canvas.drawPath(paths[2], paint);
        // 沿着路径绘制一段文本
        paint.setStyle(Paint.Style.FILL);
```

```
                canvas.drawTextOnPath(DRAW_STR, paths[2], -10, 20, paint);
            }
        }
    }
```

上面的程序三次调用了 drawTextOnPath()在 View 组件上绘制文本，此时绘制的文本并不是简单地水平排列，而是沿着指定路径绘制的。运行上面的程序，将看到如图 7.3 所示的效果。

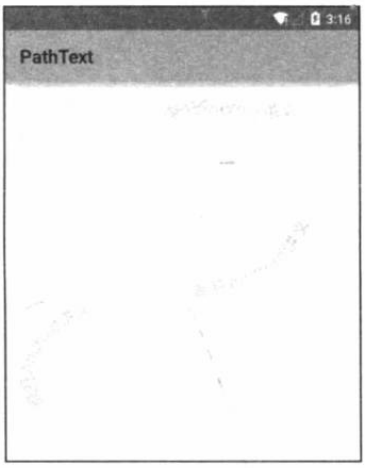


图 7.3 沿着路径绘制文本

7.2.3 绘制游戏动画

掌握了 Canvas 绘图知识之后，如果需要实现游戏动画也是非常简单的。动画其实就是不断地重复调用 View 组件的 onDraw(Canvas canvas)方法，如果每次在 View 组件上绘制的图形并不相同，那么就成了习惯上所说的动画。

为了让 View 组件上绘制的图形发生改变（无非是位置、大小、角度等发生改变），这就需要程序采用变量来“记住”这些状态数据——如果需要游戏动画随用户操作而改变，就为用户动作编写事件监听器，在监听器中修改这些数据；如果需要游戏动画“自动”改变，也就是随时间的流逝而改变，那么就需要使用定时器（Timer），让 Timer 控制这些状态数据定期改变。

不管使用哪种方式，每次 View 组件上的图形状态数据发生改变时，都应该通知 View 组件重写调用 onDraw(Canvas canvas)方法重绘该组件。通知 View 重绘可调用 invalidate（在 UI 线程中）或 postInvalidate（在非 UI 线程中）。

实例：采用双缓冲实现画图板

本实例要实现一个画图板，当用户在触摸屏上移动时，即可在屏幕上绘制任意的图形。实现手绘功能其实是一种假象：表面上看起来可以随用户在触摸屏上自由地画曲线，实际上依然利用的是 Canvas 的 drawLine()方法画直线，每条直线都是从上一次拖动事件的发生点画到本次拖动事件的发生点。当用户在触摸屏上移动时，两次拖动事件发生点的距离很小，多条极短的直线连接起来，肉眼看起来就是直线了。借助于 Android 提供的 Path 类，可以非常方便地实现这种效果。

需要指出的是，如果程序每次都只是从上次拖动事件的发生点绘一条直线到本次拖动事件的发生点，那么用户前面绘制的就会丢失。为了保留用户之前绘制的内容，程序要借助于“双缓冲”技术。

所谓双缓冲技术其实很简单：当程序需要在指定 View 上进行绘制时，程序并不直接绘制到该 View 组件上，而是先绘制到内存中的一个 Bitmap 图片（这就是缓冲区）上，等到内存中的 Bitmap 绘制好之后，再一次性地将 Bitmap 绘制到 View 组件上。

该程序需要一个自定义 View，该 View 的代码如下。

程序清单：codes\07\7.2\HandDraw\app\src\main\java\org\crazyit\image\DrawView.java

```
public class DrawView extends View
{
    // 定义记录前一个拖动事件发生点的坐标
    float preX;
    float preY;
```

```
private Path path;
public Paint paint = null;
// 定义内存中的一个图片，该图片将作为缓冲区
Bitmap cacheBitmap = null;
// 定义 cacheBitmap 上的 Canvas 对象
Canvas cacheCanvas = null;
public DrawView(Context context, int width , int height)
{
    super(context);
    // 创建一个与该 View 具有相同大小的缓冲区
    cacheBitmap = Bitmap.createBitmap(width, height,
        Bitmap.Config.ARGB_8888);
    cacheCanvas = new Canvas();
    path = new Path();
    // 设置 cacheCanvas 将会绘制到内存中的 cacheBitmap 上
    cacheCanvas.setBitmap(cacheBitmap);
    // 设置画笔的颜色
    paint = new Paint(Paint.DITHER_FLAG);
    paint.setColor(Color.RED);
    // 设置画笔风格
    paint.setStyle(Paint.Style.STROKE);
    paint.setStrokeWidth(1);
    // 反锯齿
    paint.setAntiAlias(true);
    paint.setDither(true);
}
@Override
public boolean onTouchEvent(MotionEvent event)
{
    // 获取拖动事件的发生位置
    float x = event.getX();
    float y = event.getY();
    switch (event.getAction())
    {
        case MotionEvent.ACTION_DOWN:
            // 从前一个点绘制到当前点之后，把当前点定义成下次绘制的前一个点
            path.moveTo(x, y);
            preX = x;
            preY = y;
            break;
        case MotionEvent.ACTION_MOVE:
            // 从前一个点绘制到当前点之后，把当前点定义成下次绘制的前一个点
            path.quadTo(preX, preY, x, y);
            preX = x;
            preY = y;
            break;
        case MotionEvent.ACTION_UP:
            cacheCanvas.drawPath(path, paint); // ①
            path.reset();
            break;
    }
    invalidate();
    // 返回 true 表明处理方法已经处理该事件
    return true;
}
@Override
public void onDraw(Canvas canvas)
{
    Paint bmpPaint = new Paint();
    // 将 cacheBitmap 绘制到该 View 组件上
    canvas.drawBitmap(cacheBitmap, 0, 0, bmpPaint); // ②
```

```
            // 沿着 path 绘制
            canvas.drawPath(path, paint);
        }
    }
```

上面程序中的粗体字代码为触摸屏的拖动事件提供了响应——只是简单地修改了 preX、preY 两个属性，并通知该组件重绘。

在这个自定义 View 组件中，程序重写了该 View 的 onDraw(Canvas canvas)方法，注意该方法中的②号代码，这行代码并不是调用该 View 的 Canvas 进行绘制，而是调用了缓存 Bitmap 的 Canvas 进行绘图，这表明是向缓冲区绘图。程序的②号粗体字代码将缓冲区中的 BitMap 对象绘制到 View 组件上——这就是所谓的“双缓冲”技术。

提供了上面的 DrawView 之后，接下来把该组件添加到主界面中，本程序无须使用界面布局文件，本程序直接在 Activity 中使用代码创建程序界面。

本程序还提供了菜单来设置画笔的颜色和笔触大小，该程序的菜单资源文件如下。

程序清单：codes\07\7.2\HandDraw\app\src\main\res\menu\menu_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="@string/color">
        <menu>
            <!-- 定义一组单选菜单项 -->
            <group android:checkableBehavior="single">
                <!-- 定义多个菜单项 -->
                <item android:id="@+id/red"
                    android:title="@string/color_red"/>
                <item android:id="@+id/green"
                    android:title="@string/color_green"/>
                <item android:id="@+id/blue"
                    android:title="@string/color_blue"/>
            </group>
        </menu>
    </item>
    <item android:title="@string/width">
        <menu>
            <!-- 定义一组菜单项 -->
            <group>
                <!-- 定义三个菜单项 -->
                <item android:id="@+id/width_1"
                    android:title="@string/width_1"/>
                <item android:id="@+id/width_3"
                    android:title="@string/width_3"/>
                <item android:id="@+id/width_5"
                    android:title="@string/width_5"/>
            </group>
        </menu>
    </item>
    <item android:id="@+id/blur" android:title="@string/blur"/>
    <item android:id="@+id/emboss" android:title="@string/emboss"/>
</menu>
```

主程序负责加载、显示界面布局，加载、显示上面的菜单资源。除此之外，程序还要为各菜单项编写事件响应，程序代码如下。

程序清单：codes\07\7.2\HandDraw\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    EmbossMaskFilter emboss;
    BlurMaskFilter blur;
```

```
DrawView drawView;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    LinearLayout line = new LinearLayout(this);
    DisplayMetrics displayMetrics = new DisplayMetrics();
    // 获取创建的宽度和高度
    getWindowManager().getDefaultDisplay()
        .getRealMetrics(displayMetrics);
    // 创建一个 DrawView，该 DrawView 的宽度、高度与该 Activity 保持相同
    drawView = new DrawView(this, displayMetrics.widthPixels
        , displayMetrics.heightPixels);
    line.addView(drawView);
    setContentView(line);
    emboss = new EmbossMaskFilter(new float[]
        { 1.5f, 1.5f, 1.5f }, 0.6f,    6, 4.2f);
    blur = new BlurMaskFilter(8, BlurMaskFilter.Blur.NORMAL);
}
@Override
// 负责创建选项菜单
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflator = new MenuInflater(this);
    // 加载 R.menu.my_menu 对应的菜单，并添加到 menu 中
    inflator.inflate(R.menu.menu_main, menu);
    return super.onCreateOptionsMenu(menu);
}
@Override
// 菜单项被单击后的回调方法
public boolean onOptionsItemSelected(MenuItem mi)
{
    // 判断单击的是哪个菜单项，并有针对性地做出响应
    switch (mi.getItemId())
    {
        case R.id.red:
            drawView.paint.setColor(Color.RED);
            mi.setChecked(true);
            break;
        case R.id.green:
            drawView.paint.setColor(Color.GREEN);
            mi.setChecked(true);
            break;
        case R.id.blue:
            drawView.paint.setColor(Color.BLUE);
            mi.setChecked(true);
            break;
        case R.id.width_1:
            drawView.paint.setStrokeWidth(1);
            break;
        case R.id.width_3:
            drawView.paint.setStrokeWidth(3);
            break;
        case R.id.width_5:
            drawView.paint.setStrokeWidth(5);
            break;
        case R.id.blur:
            drawView.paint.setMaskFilter(blur);
            break;
        case R.id.emboss:
            drawView.paint.setMaskFilter(emboss);
            break;
```

```
        }
        return true;
    }
}
```

上面的程序代码比较简单，当用户单击不同的菜单项之后，程序只要简单地修改DrawView组件内的Paint对象的颜色和笔触粗细即可。运行上面的程序，将看到如图7.4所示的界面。

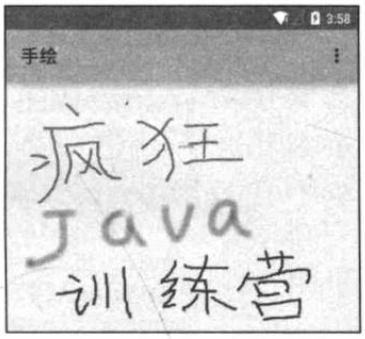


图7.4　双缓冲实现绘图板

提示：

阅读过《疯狂Java讲义》一书的读者可能对这个程序感到很“眼熟”，实际上这个程序所用的“双缓冲”技术，编程思路都来自《疯狂Java讲义》一书中11.8节的示例。正如笔者前面提到的，Android开发并不难——如果读者有扎实的Java基础，再加上一定的界面编程经验，学习Android开发将非常轻松。

实例：弹球游戏

下面的程序开发了一个简单的弹球游戏，其中小球和球拍分别以圆形区域和矩形区域代替，小球开始以随机速度向下运动，遇到边框或球拍时小球反弹；球拍则由用户控制，当用户按下向左、向右键时，球拍将会向左、向右移动。

程序清单：codes\07\7.2\PinBall\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    // 桌面的宽度
    private int tableWidth;
    // 桌面的高度
    private int tableHeight;
    // 球拍的垂直位置
    private int racketY;
    // 下面定义球拍的高度和宽度
    private final int RACKET_HEIGHT = 30;
    private final int RACKET_WIDTH = 90;
    // 小球的大小
    private final int BALL_SIZE = 16;
    // 小球纵向的运行速度
    private int ySpeed = 15;
    Random rand = new Random();
    // 返回一个-0.5~0.5的比率，用于控制小球的运行方向
    private double xyRate = rand.nextDouble() - 0.5;
    // 小球横向的运行速度
    private int xSpeed = (int) (ySpeed * xyRate * 2);
    // ballX和ballY代表小球的坐标
    private int ballX = rand.nextInt(200) + 20;
    private int ballY = rand.nextInt(10) + 20;
    // racketX代表球拍的水平位置
    private int racketX = rand.nextInt(200);
    // 游戏是否结束的旗标
    private boolean isLose = false;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 去掉窗口标题
```

```
requestWindowFeature(Window.FEATURE_NO_TITLE);
// 全屏显示
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
    WindowManager.LayoutParams.FLAG_FULLSCREEN);
// 创建 GameView 组件
final GameView gameView = new GameView(this);
setContentView(gameView);
// 获取窗口管理器
WindowManager windowManager = getWindowManager();
Display display = windowManager.getDefaultDisplay();
DisplayMetrics metrics = new DisplayMetrics();
display.getMetrics(metrics);
// 获得屏幕宽和高
tableWidth = metrics.widthPixels;
tableHeight = metrics.heightPixels;
racketY = tableHeight - 80;
final Handler handler = new Handler()
{
    public void handleMessage(Message msg)
    {
        if (msg.what == 0x123)
        {
            gameView.invalidate();
        }
    }
};
gameView.setOnKeyListener(new OnKeyListener() // ②
{
    @Override
    public boolean onKey(View source, int keyCode, KeyEvent event)
    {
        // 获取由哪个键触发的事件
        switch (event.getKeyCode())
        {
            // 控制挡板左移
            case KeyEvent.KEYCODE_A:
                if (racketX > 0) racketX -= 10;
                break;
            // 控制挡板右移
            case KeyEvent.KEYCODE_D:
                if (racketX < tableWidth - RACKET_WIDTH) racketX += 10;
                break;
        }
        // 通知 gameView 组件重绘
        gameView.invalidate();
        return true;
    }
});
final Timer timer = new Timer();
timer.schedule(new TimerTask() // ①
    {
        @Override
        public void run()
        {
            // 如果小球碰到左边边框
            if (ballX <= 0 || ballX >= tableWidth - BALL_SIZE)
            {
                xSpeed = -xSpeed;
            }
            // 如果小球高度超出了球拍位置，且横向不在球拍范围之内，游戏结束
            if (ballY >= racketY - BALL_SIZE
                    && (ballX < racketX || ballX > racketX
                            + RACKET_WIDTH))
```

```
                {
                    timer.cancel();
                    // 设置游戏是否结束的旗标为true
                    isLose = true;
                }
                // 如果小球位于球拍之内，且到达球拍位置，小球反弹
                else if (ballY <= 0
                    || (ballY >= racketY - BALL_SIZE
                        && ballX > racketX && ballX <= racketX
                        + RACKET_WIDTH))
                {
                    ySpeed = -ySpeed;
                }
                // 小球坐标增加
                ballY += ySpeed;
                ballX += xSpeed;
                // 发送消息，通知系统重绘组件
                handler.sendEmptyMessage(0x123);
            }
        }, 0, 100);
    }
    class GameView extends View
    {
        Paint paint = new Paint();
        public GameView(Context context)
        {
            super(context);
            setFocusable(true);
        }
        // 重写View的onDraw方法，实现绘画
        public void onDraw(Canvas canvas)
        {
            paint.setStyle(Paint.Style.FILL);
            // 设置去锯齿
            paint.setAntiAlias(true);
            // 如果游戏已经结束
            if (isLose)
            {
                paint.setColor(Color.RED);
                paint.setTextSize(40);
                canvas.drawText("游戏已结束", tableWidth / 2 - 100, 200, paint);
            }
            // 如果游戏还未结束
            else
            {
                // 设置颜色，并绘制小球
                paint.setColor(Color.rgb(255, 0, 0));
                canvas.drawCircle(ballX, ballY, BALL_SIZE, paint);
                // 设置颜色，并绘制球拍
                paint.setColor(Color.rgb(80, 80, 200));
                canvas.drawRect(racketX, racketY, racketX + RACKET_WIDTH,
                        racketY + RACKET_HEIGHT, paint);
            }
        }
    }
}
```

上面的程序提供了一个GameView，这个GameView很简单，它只是根据程序中小球的坐标、球拍的坐标来绘制小球和球拍。

除此之外，这个所谓的“游戏”本质上就是一个动画程序，该程序中可以“动”的内容很少，只有一个小球和一个球拍。其中小球的坐标由定时器定时修改（如程序中①号代码所示）；

球拍的坐标则由键盘动作来修改（如程序中②号代码所示）。运行上面的程序，将可以看到如图 7.5 所示的游戏界面。

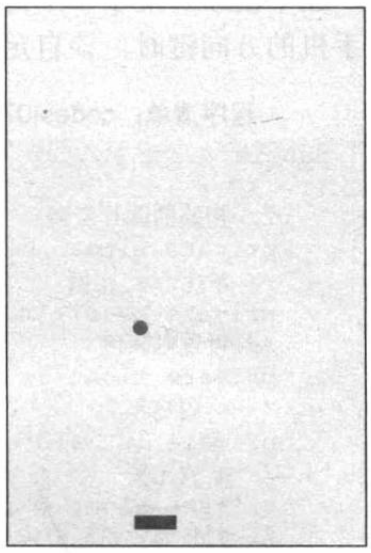


图 7.5 弹球游戏界面

虽然图 7.5 所示的弹球游戏还略嫌粗糙，但只要开发者找到一些“美丽”的图片，替换程序中的小球、球拍，再考虑在界面上方增加一些“障碍物”来增加游戏乐趣，以及为小球碰撞边界、碰撞球拍时增加音效，这个游戏将会变得“生动”起来。这个游戏的 AWT 版本同样来自于《疯狂 Java 讲义》，疯狂 Java 联盟（crazyit.org）站点上有大量已经完成的弹球游戏，读者也可利用那些游戏中的图片资源来完善这个游戏。

7.3 图形特效处理

Android 除了前面介绍的这些图形支持之外，还提供了一些额外的更高级的图形特效支持，这些图形特效支持可以让开发者开发出更绚丽的 UI 界面。

7.3.1 使用 Matrix 控制变换

Matrix 是 Android 提供的一个矩阵工具类，它本身并不能对图形或组件进行变换，但它可与其他 API 结合来控制图形、组件的变换。

使用 Matrix 控制图形或组件变换的步骤如下。

① 获取 Matrix 对象，该 Matrix 对象既可新创建，也可直接获取其他对象内封装的 Matrix（例如 Transformation 对象内部就封装了 Matrix）。

② 调用 Matrix 的方法进行平移、旋转、缩放、倾斜等。

③ 将程序对 Matrix 所做的变换应用到指定图形或组件。

提示：

从这里的介绍可以看出，Matrix 不仅可用于控制图形的平移、旋转、缩放、倾斜变换，也可控制 View 组件进行平移、旋转和缩放等。

Matrix 提供了如下方法来控制平移、旋转和缩放。

- setTranslate(float dx, float dy)：控制 Matrix 进行平移。
- setSkew(float kx, float ky, float px, float py)：控制 Matrix 以 px、py 为轴心进行倾斜。kx、ky 为 *X*、*Y* 方向上的倾斜距离。
- setSkew(float kx, float ky)：控制 Matrix 进行倾斜。kx、ky 为 *X*、*Y* 方向上的倾斜距离。
- setRotate(float degrees)：控制 Matrix 进行旋转，degrees 控制旋转的角度。
- setRotate(float degrees, float px, float py)：设置以 px、py 为轴心进行旋转，degrees 控制旋转的角度。
- setScale(float sx, float sy)：设置 Matrix 进行缩放，sx、sy 控制 *X*、*Y* 方向上的缩放比例。
- setScale(float sx, float sy, float px, float py)：设置 Matrix 以 px、py 为轴心进行缩放，sx、sy 控制 *X*、*Y* 方向上的缩放比例。

一旦对 Matrix 进行了变换，接下来就可应用该 Matrix 对图形进行控制了。例如，Canvas 就提供了一个 drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint)方法，调用该方法就可以在绘制 bitmap 时应用 Matrix 上的变换。

如下程序开发了一个自定义 View，该自定义 View 可以检测到用户的键盘事件，当用户单击手机的方向键时，该自定义 View 会用 Matrix 对绘制的图形进行旋转、倾斜变换。

程序清单：codes\07\7.3\MatrixTest\app\src\main\java\org\crazyit\image\MyView.java

```
public class MyView extends View
{
    // 初始的图片资源
    private Bitmap bitmap;
    // Matrix 实例
    private Matrix matrix = new Matrix();
    // 设置倾斜度
    private float sx = 0.0f;
    // 位图宽和高
    private int width, height;
    // 缩放比例
    private float scale = 1.0f;
    // 判断缩放还是旋转
    private boolean isScale = false;
    public MyView(Context context , AttributeSet set)
    {
        super(context , set);
        // 获得位图
        bitmap = ((BitmapDrawable) context.getResources().getDrawable(
            R.drawable.a)).getBitmap();
        // 获得位图宽
        width = bitmap.getWidth();
        // 获得位图高
        height = bitmap.getHeight();
        // 使当前视图获得焦点
        this.setFocusable(true);
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 重置 Matrix
        matrix.reset();
        if (!isScale)
        {
            // 旋转 Matrix
            matrix.setSkew(sx, 0);
        }
        else
        {
            // 缩放 Matrix
            matrix.setScale(scale, scale);
        }
        // 根据原始位图和 Matrix 创建新图片
        Bitmap bitmap2 = Bitmap.createBitmap(bitmap, 0, 0, width
            , height,matrix, true);
        // 绘制新位图
        canvas.drawBitmap(bitmap2, matrix, null);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        switch(keyCode)
        {
            // 向左倾斜
            case KeyEvent.KEYCODE_A:
                isScale = false;
```

```
                sx += 0.1;
                postInvalidate();
                break;
            // 向右倾斜
            case KeyEvent.KEYCODE_D:
                isScale = false;
                sx -= 0.1;
                postInvalidate();
                break;
            // 放大
            case KeyEvent.KEYCODE_W:
                isScale = true;
                if (scale < 2.0)
                    scale += 0.1;
                postInvalidate();
                break;
            // 缩小
            case KeyEvent.KEYCODE_S:
                isScale = true;
                if (scale > 0.5)
                    scale -= 0.1;
                postInvalidate();
                break;
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

上面程序中的粗体字代码就是通过 Matrix 控制图片倾斜、缩放的关键代码，当用户单击手机的方向键时，事件处理器负责修改程序中 sx（控制水平倾斜度）和 scale（控制缩放比）两个参数。

把上面的自定义 View 放在 Activity 中显示出来，运行该程序将看到如图 7.6 所示的界面。

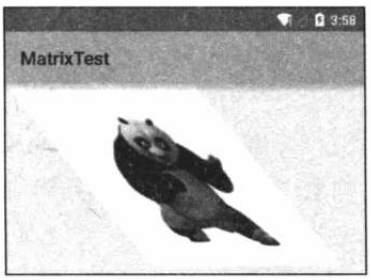


图 7.6 使用 Matrix 控制倾斜

实例：移动游戏背景

借助于 Bitmap 的 createBitmap 方法可以“挖取”源位图的其中一块，这样可以在程序中通过定时器控制不断地“挖取”源位图不同位置的块，从而给用户看到背景移动“假象”。

假设要开发经典的“雷电”飞机游戏，为了给用户造成飞机不断飞行的错觉，可以通过这种方式来控制背景图片不断下移，用户就会感觉飞机在不断地向上飞行。

例如，如下程序实现了背景图片不断“下移”的效果，用户会感觉飞机在不断地向上飞行。

程序清单：codes\07\7.3\MoveBack\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this));
    }
    class MyView extends View
    {
        // 记录背景位图的实际高度
        final int BACK_HEIGHT = 1700;
        // 背景图片
        private Bitmap back;
```

```
private Bitmap plane;
// 定义图片的宽度
final int WIDTH = 640;
final int HEIGHT = 880;
private Matrix matrix = new Matrix();
// 背景图片的开始位置
private int startY = BACK_HEIGHT - HEIGHT;
public MyView(Context context)
{
    super(context);
    back = BitmapFactory.decodeResource(context.getResources(),
            R.drawable.back_img);
    // 获取窗口管理器
    WindowManager windowManager = getWindowManager();
    Display display = windowManager.getDefaultDisplay();
    DisplayMetrics metrics = new DisplayMetrics();
    display.getMetrics(metrics);
    // 获得屏幕的宽度
    float screenWidth = metrics.widthPixels;
    // 获取图片的缩放比例
    float scale = screenWidth / WIDTH;
    matrix.setScale(scale , scale);
    plane = BitmapFactory.decodeResource(context.getResources(),
            R.drawable.plane);
    final Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                // 重新开始移动
                if (startY <= 3)
                {
                    startY = BACK_HEIGHT - HEIGHT;
                }
                else
                {
                    startY -= 3;
                }
            }
            invalidate();
        }
    };
    new Timer().schedule(new TimerTask()
    {
        @Override
        public void run()
        {
            handler.sendEmptyMessage(0x123);
        }
    }, 0, 100);
}
@Override
public void onDraw(Canvas canvas)
{
    // 根据原始位图和缩放 Matrix 创建新图片
    Bitmap bitmap2 = Bitmap
        .createBitmap(back, 0, startY, WIDTH, HEIGHT , matrix , false); // ①
    // 绘制新位图
    canvas.drawBitmap(bitmap2, 0, 0, null);
    // 绘制飞机
```

```
                canvas.drawBitmap(plane, 320, 700, null);
            }
        }
    }
```

上面程序中的①号粗体字代码根据 startY 控制“挖取”back 位图中不同位置的块，并将它绘制在当前 View 上。程序的粗体字代码则通过定时器来控制 startY 不断地改变，这样即可看到 back 图片不断下移的效果。运行该程序，读者即可看到飞机向上飞行的“错觉”。

提示：

该程序并未为用户按键事件提供监听器，实际上读者完全可以为该 View 的按键事件绑定监听器，这样用户即可控制飞机飞行。这已经是雷电游戏的雏型了。如果读者想要自己更好地完善这个雷电游戏，可以从如下三个方面进行改进。

（1）为该 View 添加一个数组来控制随机出现的敌机的坐标，当然也要通过定时器来控制敌机坐标的改变。

（2）为该 View 添加一个数组来控制飞机所发出的子弹的坐标，当然也要通过定时器来控制子弹坐标的改变。

（3）定期检查子弹是否与敌机“碰撞”，检查敌机是否与自己的飞机“碰撞”，如果发生碰撞，程序在“碰撞”点播放“爆炸”动画。本章后面会介绍在指定点播放“爆炸”动画的示例。

7.3.2 使用 drawBitmapMesh 扭曲图像

Canvas 提供了一个 drawBitmapMesh(Bitmap bitmap, int meshWidth, int meshHeight, float[] verts, int vertOffset, int[] colors, int colorOffset, Paint paint)方法，该方法可以对 bitmap 进行扭曲。这个方法非常灵活，如果用好这个方法，开发者可以在 Android 应用上开发出“水波荡漾”、“风吹旗帜”等各种扭曲效果。

drawBitmapMesh()方法的关键参数说明如下。

- bitmap：指定需要扭曲的源位图。
- meshWidth：该参数控制在横向上把该源位图划分成多少格。
- meshHeight：该参数控制在纵向上把该源位图划分成多少格。
- verts：该参数是一个长度为(meshWidth+1)*(meshHeight+1)*2 的数组，它记录了扭曲后的位图各“顶点”（图 7.7 所示网格线的交点）位置。虽然它是一个一维数组，但实际上它记录的数据是形如($x0$,$y0$)、($x1$,$y1$)、($x2$,$y2$)、…、(xN,yN)格式的数据，这些数组元素控制对 bitmap 位图的扭曲效果。
- vertOffset：控制 verts 数组中从第几个数组元素开始才对 bitmap 进行扭曲（忽略 verOffset 之前数据的扭曲效果）。

drawBitmapMesh()方法对源位图扭曲时最关键的参数是 meshWidth、meshHeight、verts，这三个参数对扭曲的控制如图 7.7 所示。

从图 7.7 可以看出，当程序希望调用 drawBitmapMesh()方法对位图进行扭曲时，关键是计算 verts 数组的值——该数组的值记录了扭曲后的位图上各“顶点”（图 7.7 所示网格线的交点）的坐标。

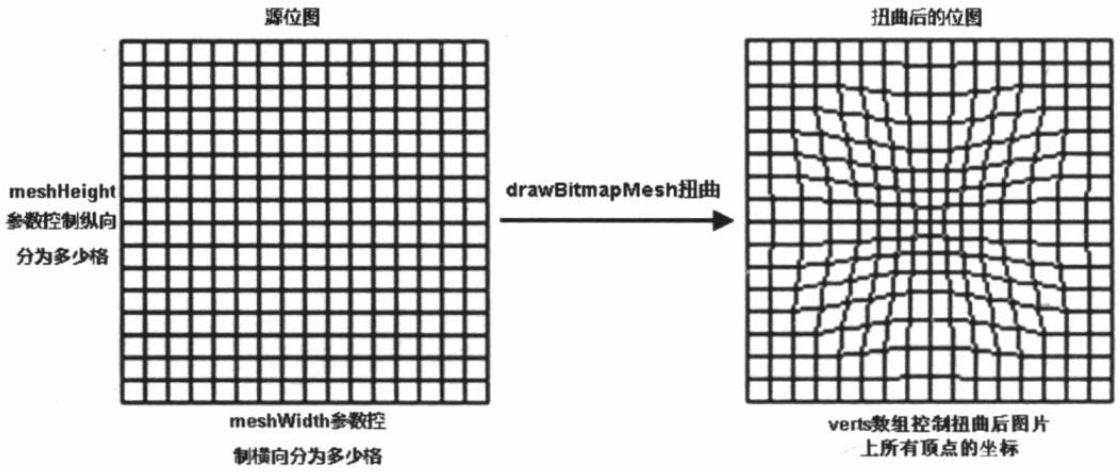


图 7.7　drawBitmapMesh 扭曲示意图

提示：

初学者往往容易对 drawBitmapMesh()方法感到不好理解，如果读者有 Photoshop 图形处理的经验，应该对图 7.7 所示的扭曲效果很熟悉——实际上该方法可以模拟 Photoshop 的扭曲“滤镜”。Android 应用面向的终端用户都是普通人群，应用界面对这些用户的影响非常大，Android 提供的 drawBitmapMesh()方法带给开发者一种非常灵活的控制。在实际开发中，笔者甚至“贪心”地希望 Android 提供更多方法，这些方法可以更好地模拟 Photoshop 的各种“滤镜”。

实例：可揉动的图片

本实例程序将会通过 drawBitmapMesh()方法来控制图片的扭曲，当用户“触摸”图片的指定点时，该图片会在这个点被用户“按”下去——就像这张图片铺在“极软的床上”一样。

为了实现这个效果，程序要在用户触摸图片的指定点时，动态地改变 verts 数组里每个元素的位置（控制扭曲后每个顶点的坐标）——这种改变也简单：程序计算图片上每个顶点与触摸点的距离，顶点与触摸点的距离越小，该顶点向触摸点移动的距离越大。

下面是该程序的代码。

程序清单：codes\07\7.3\MeshTest\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    private Bitmap bitmap;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this, R.drawable.jinta));
    }
    private class MyView extends View
    {
        // 定义两个常量，这两个常量指定该图片横向、纵向上都被划分为 20 格
        private final int WIDTH = 20;
        private final int HEIGHT = 20;
        // 记录该图片上包含 441 个顶点
        private final int COUNT = (WIDTH + 1) * (HEIGHT + 1);
        // 定义一个数组，保存 Bitmap 上的 21 * 21 个点的坐标
        private final float[] verts = new float[COUNT * 2];
        // 定义一个数组，记录 Bitmap 上的 21 * 21 个点经过扭曲后的坐标
```

```
// 对图片进行扭曲的关键就是修改该数组里元素的值
private final float[] orig = new float[COUNT * 2];
public MyView(Context context, int drawableId)
{
    super(context);
    setFocusable(true);
    // 根据指定资源加载图片
    bitmap = BitmapFactory.decodeResource(getResources()
        , drawableId);
    // 获取图片宽度、高度
    float bitmapWidth = bitmap.getWidth();
    float bitmapHeight = bitmap.getHeight();
    int index = 0;
    for (int y = 0; y <= HEIGHT; y++)
    {
        float fy = bitmapHeight * y / HEIGHT;
        for (int x = 0; x <= WIDTH; x++)
        {
            float fx = bitmapWidth * x / WIDTH;
            // 初始化 orig、verts 数组。初始化后，orig、verts
            // 两个数组均匀地保存了 21 * 21 个点的 x,y 坐标
            orig[index * 2 + 0] = verts[index * 2 + 0] = fx;
            orig[index * 2 + 1] = verts[index * 2 + 1] = fy;
            index += 1;
        }
    }
    // 设置背景色
    setBackgroundColor(Color.WHITE);
}
@Override
protected void onDraw(Canvas canvas)
{
    // 对 bitmap 按 verts 数组进行扭曲
    // 从第一个点（由第 5 个参数 0 控制）开始扭曲
    canvas.drawBitmapMesh(bitmap, WIDTH, HEIGHT, verts
        , 0, null, 0,null);
}
// 工具方法，用于根据触摸事件的位置计算 verts 数组里各元素的值
private void warp(float cx, float cy)
{
    for (int i = 0; i < COUNT * 2; i += 2)
    {
        float dx = cx - orig[i + 0];
        float dy = cy - orig[i + 1];
        float dd = dx * dx + dy * dy;
        // 计算每个坐标点与当前点（cx, cy）之间的距离
        float d = (float) Math.sqrt(dd);
        // 计算扭曲度，距离当前点（cx, cy）越远，扭曲度越小
        float pull = 80000 / ((float) (dd * d));
        // 对 verts 数组（保存 bitmap 上 21 * 21 个点经过扭曲后的坐标）重新赋值
        if (pull >= 1)
        {
            verts[i + 0] = cx;
            verts[i + 1] = cy;
        }
        else
        {
            // 控制各顶点向触摸事件发生点偏移
            verts[i + 0] = orig[i + 0] + dx * pull;
            verts[i + 1] = orig[i + 1] + dy * pull;
        }
    }
    // 通知 View 组件重绘
```

```
            invalidate();
        }
        @Override
        public boolean onTouchEvent(MotionEvent event)
        {
            // 调用 warp 方法根据触摸屏事件的坐标点来扭曲 verts 数组
            warp(event.getX(), event.getY());
            return true;
        }
    }
}
```

上面程序中的粗体字代码是关键，该方法将会根据触摸点的位置（由 cx、cy 坐标控制）动态修改 verts 数组里所有数组元素的值，这样就控制了 drawBitmapMesh()方法的扭曲效果。

运行该程序并触碰该图片的任意位置，将可以看到如图 7.8 所示的效果。

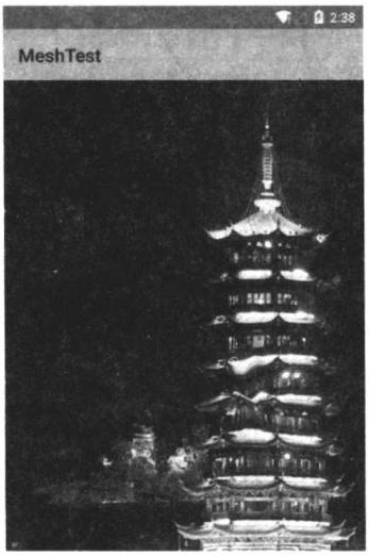


图 7.8　扭曲图像

7.3.3　使用 Shader 填充图形

前面介绍 Paint 时提到该 Shader 包含了一个 setShader(Shader s)方法，该方法控制“画笔”的渲染效果——Android 不仅可以使用颜色来填充图形（包括前面介绍的矩形、椭圆、圆形等各种几何图形），也可以使用 Shader 对象指定的渲染效果来填充图形。

Shader 本身是一个抽象类，它提供了如下实现类。

- BitmapShader：使用位图平铺的渲染效果。
- LinearGradient：使用线性渐变来填充图形。
- RadialGradient：使用圆形渐变来填充图形。
- SweepGradient：使用角度渐变来填充图形。
- ComposeShader：使用组合渲染效果来填充图形。

如果使用文字来描述这些渲染效果，不仅显得十分啰唆，而且极难讲清楚。但如果读者有“玩”Flash 的经验，则应该对位图平铺、线性渐变、圆形渐变、角度渐变等名词十分熟悉，那么此处就无须笔者多费笔墨来描述这些渲染效果了，因为它们与 Flash 提供的位图平铺、线性渐变、圆形渐变、角度渐变完全一样。如果读者没有 Flash 经验，也无须担心，运行下面的程序即可明白各种 Shader 对象的渲染效果。

下面的程序中包含了 5 个按钮，当用户按下不同的按钮时系统将会设置 Paint 使用不同的 Shader，这样读者即可看到不同 Shader 的效果。

程序清单：codes\07\7.3\ShaderTest\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
    implements OnClickListener
{
    // 声明位图渲染对象
    private Shader[] shaders = new Shader[5];
    // 声明颜色数组
    private int[] colors;
    MyView myView;
    // 自定义视图类
    @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myView = (MyView)findViewById(R.id.my_view);
        // 获得 Bitmap 实例
        Bitmap bm = BitmapFactory.decodeResource(getResources()
            , R.drawable.water);
        // 设置渐变的颜色组，也就是按红、绿、蓝的方式渐变
        colors = new int[] { Color.RED, Color.GREEN, Color.BLUE };
        // 实例化 BitmapShader，x 坐标方向重复图形，y 坐标方向镜像图形
        shaders[0] = new BitmapShader(bm, TileMode.REPEAT,
            TileMode.MIRROR);
        // 实例化 LinearGradient
        shaders[1] = new LinearGradient(0, 0, 100, 100
            , colors, null, TileMode.REPEAT);
        // 实例化 RadialGradient
        shaders[2] = new RadialGradient(100, 100, 80, colors, null,
            TileMode.REPEAT);
        // 实例化 SweepGradient
        shaders[3] = new SweepGradient(160, 160, colors, null);
        // 实例化 ComposeShader
        shaders[4] = new ComposeShader(shaders[1], shaders[2],
            PorterDuff.Mode.DARKEN);
        Button bn1 = (Button)findViewById(R.id.bn1);
        Button bn2 = (Button)findViewById(R.id.bn2);
        Button bn3 = (Button)findViewById(R.id.bn3);
        Button bn4 = (Button)findViewById(R.id.bn4);
        Button bn5 = (Button)findViewById(R.id.bn5);
        bn1.setOnClickListener(this);
        bn2.setOnClickListener(this);
        bn3.setOnClickListener(this);
        bn4.setOnClickListener(this);
        bn5.setOnClickListener(this);
    }
    @Override
    public void onClick(View source)
    {
        switch(source.getId())
        {
            case R.id.bn1:
                myView.paint.setShader(shaders[0]);
                break;
            case R.id.bn2:
                myView.paint.setShader(shaders[1]);
                break;
            case R.id.bn3:
                myView.paint.setShader(shaders[2]);
                break;
            case R.id.bn4:
                myView.paint.setShader(shaders[3]);
                break;
            case R.id.bn5:
                myView.paint.setShader(shaders[4]);
                break;
        }
        // 重绘界面
        myView.invalidate();
    }
}
```

上面程序中的第一段粗体字代码创建了多个 Shader 对象，第二段粗体字代码会根据用户按下不同的按钮来修改 MyView 对象 Paint——MyView 对象将会用该 Paint 来绘制一个矩形。

随着 MyView 的 Paint 所使用的 Shader 不断改变，MyView 对象所绘制的矩形也随之改变。

运行上面的程序，如果使用 BitmapShader 绘制矩形，将看到如图 7.9 所示的效果。

如果使用 SweepGradient 绘制矩形，将看到如图 7.10 所示的效果。

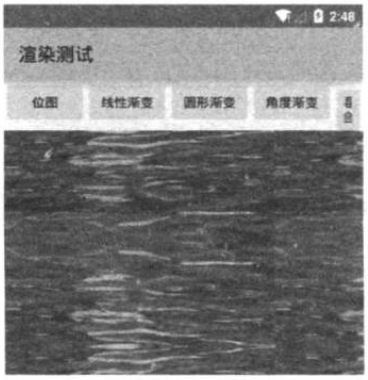


图 7.9 位图平铺的渲染效果

图 7.10 角度渐变的渲染效果

图 7.9、图 7.10 显示了 Android 提供的 BitmapShader、SweepGradient 两种 Shader 的渲染效果，读者可自行运行该程序去查看其他 Shader 的渲染效果。

7.4 逐帧（Frame）动画

逐帧（Frame）动画是最容易理解的动画，它要求开发者把动画过程的每张静态图片都收集起来，然后由 Android 来控制依次显示这些静态图片，再利用人眼“视觉暂留”的原理，给用户造成“动画”的错觉。逐帧动画的动画原理与放电影的原理完全一样。

7.4.1 AnimationDrawable 与逐帧动画

前面讲解定义 Android 资源时已经介绍了动画资源，事实上逐帧动画通常也是采用 XML 资源文件进行定义的。

定义逐帧动画非常简单，只要在<animation-list.../>元素中使用<item.../>子元素定义动画的全部帧，并指定各帧的持续时间即可。定义逐帧动画的语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
   android:oneshot=["true" | "false"] >
   <item android:drawable="@[package:]drawable/drawable_resource_name"
      android:duration="integer" />
</animation-list>
```

在上面的语法格式中，android:onshot 控制该动画是否循环播放，如果该参数指定为 true，则动画将不会循环播放；否则该动画将会循环播放。每个<item.../>子元素添加一帧。

提示：

Android 完全支持在 Java 代码中创建逐帧动画，如果开发者喜欢，则完全可以先创建 AnimationDrawable 对象，然后调用 addFrame(Drawable frame, int duration) 向该动画中添加帧，每调用一次 addFrame()方法，就向<animation-list.../>元素中添加一个<item.../>子元素。

一旦程序获取了 AnimationDrawable 对象之后，接下来就可用 ImageView 把 AnimationDrawable 显示出来——习惯上把 AnimationDrawable 设成 ImageView 的背景即可。

下面是逐帧动画的简单示例，该示例先使用如下代码定义了一个逐帧动画资源。

程序清单：codes\07\7.4\FatPo\app\src\main\res\anim\fat_po.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 指定动画循环播放 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <!-- 添加多个帧 -->
    <item android:drawable="@drawable/fat_po_f01" android:duration="60" />
    <item android:drawable="@drawable/fat_po_f02" android:duration="60" />
    <item android:drawable="@drawable/fat_po_f03" android:duration="60" />
    <!-- 下面省略了多个类似的 item 定义 -->
    ...
</animation-list>
```

上面的 fat_po.xml 文件定义了一个逐帧动画资源，接下来就可以在程序中使用 ImageView 来显示该动画了。

需要指出的是，AnimationDrawable 代表的动画默认是不播放的，必须在程序中启动动画播放才可以。AnimationDrawable 提供了如下两个方法来开始、停止动画。

- start()：开始播放动画。
- stop()：停止播放动画。

下面的程序中包含两个按钮，其中一个按钮用于播放动画；另一个按钮可停止动画。

程序清单：codes\07\7.4\FatPo\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取两个按钮
        Button play = (Button) findViewById(R.id.play);
        Button stop = (Button) findViewById(R.id.stop);
        ImageView imageView = (ImageView) findViewById(R.id.anim);
        // 获取 AnimationDrawable 动画对象
        final AnimationDrawable anim = (AnimationDrawable) imageView
                .getBackground();
        play.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 开始播放动画
                anim.start();
            }
        });
        stop.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 停止播放动画
                anim.stop();
            }
        });
    }
}
```

上面程序中的两行粗体字代码用于控制动画的播放和停止。运行该程序，单击程序界面中的“播放”按钮，将可以看到“功夫熊猫”开始表演，如图 7.11 所示。

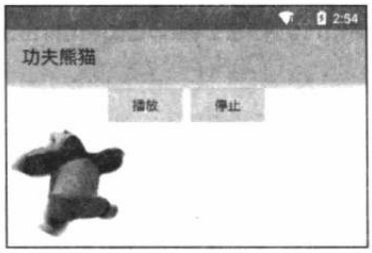


图 7.11 逐帧动画

实例：在指定点爆炸

前面介绍“雷电”飞机游戏时已经提到，当敌机与用户自己的飞机碰撞，或者自己的飞机发射的子弹与敌机碰撞时，都应该在碰撞点播放“飞机爆炸”的动画。

爆炸效果实际上是一个逐帧动画，开发者需要收集从开始爆炸到爆炸结束的所有静态图片，再将这些图片定义成一个逐帧动画，接着在碰撞点播放该逐帧动画即可。

本实例为了突出此处的“主题”——在指定点爆炸，并未增加飞机控制这些细节，只是简单地检测触摸屏事件，当用户触摸屏幕时，程序将会在触碰点“爆炸”。

该程序先使用如下代码来定义爆炸的逐帧动画资源。

程序清单：codes\07\7.4\Blast\res\anim\blast.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义动画只播放一次，不循环 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true" >
    <item android:drawable="@drawable/bom_f01" android:duration="80" />
    <item android:drawable="@drawable/bom_f02" android:duration="80" />
    <!-- 省略其他相似的 item 元素 -->
    ...
</animation-list>
```

接下来就可以利用上面的 blast.xml 所定义的动画资源了——当程序检测到用户触摸屏幕时，程序就在该触摸点播放该动画资源。程序如下。

程序清单：codes\07\7.4\Blast\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    private MyView myView;
    private AnimationDrawable anim;
    private MediaPlayer bomb;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 使用 FrameLayout 布局管理器，它允许组件自己控制位置
        FrameLayout frame = new FrameLayout(this);
        setContentView(frame);
        // 设置使用背景
        frame.setBackgroundResource(R.drawable.back);
        // 加载音效
        bomb = MediaPlayer.create(this, R.raw.bomb);
        myView = new MyView(this);
        // 设置 myView 用于显示 blast 动画
        myView.setBackgroundResource(R.anim.blast);
        // 设置 myView 默认为隐藏
        myView.setVisibility(View.INVISIBLE);
        // 获取动画对象
        anim = (AnimationDrawable) myView.getBackground();
        frame.addView(myView);
```

```
        frame.setOnTouchListener(new OnTouchListener()
        {
            @Override
            public boolean onTouch(View source, MotionEvent event)
            {
                // 只处理按下事件（避免每次产生两个动画效果）
                if (event.getAction() == MotionEvent.ACTION_DOWN)
                {
                    // 先停止动画播放
                    anim.stop();
                    float x = event.getX();
                    float y = event.getY();
                    // 控制 myView 的显示位置
                    myView.setLocation((int) y - 40, (int) x - 20);
                    myView.setVisibility(View.VISIBLE);
                    // 启动动画
                    anim.start();
                    // 播放音效
                    bomb.start();
                }
                return false;
            }
        });
    }
    // 定义一个自定义 View，该自定义 View 用于播放“爆炸”效果
    class MyView extends ImageView
    {
        public MyView(Context context)
        {
            super(context);
        }
        // 定义一个方法，该方法用于控制 MyView 的显示位置
        public void setLocation(int top, int left)
        {
            this.setFrame(left, top, left + 40, top + 40);
        }
        // 重写该方法，控制如果动画播放到最后一帧时，隐藏该 View
        @Override
        protected void onDraw(Canvas canvas) // ①
        {
            try
            {
                Field field = AnimationDrawable.class
                        .getDeclaredField("mCurFrame");
                field.setAccessible(true);
                // 获取 anim 动画的当前帧
                int curFrame = field.getInt(anim);
                // 如果已经到了最后一帧
                if (curFrame == anim.getNumberOfFrames() - 1)
                {
                    // 让该 View 隐藏
                    setVisibility(View.INVISIBLE);
                }
            }
            catch (Exception e)
            {
            }
            super.onDraw(canvas);
        }
    }
}
```

上面程序中的第一段粗体字代码就是触摸屏事件的响应代码，该程序把 MyView

（ImageView 的子类）实例移动到触摸事件的发生点，并播放动画，这就实现了在指定点爆炸的效果。

需要指出的是，程序中①号粗体字代码所定义的方法对于该程序不是必要的——即使删除该方法程序也完全正常，但这只是恰好因为爆炸效果的最后一帧是空白。换一种情况，如果爆炸动画的最后一帧不是空白，而程序又没有控制隐藏播放动画的 ImageView，用户将会看到动画结束了，但动画效果依然残留在那里。为了解决这个问题，就可借助于①号粗体字代码所定义的方法，它会自动检测动画播放到最后一帧时隐藏该 ImageView。

7.5 补间（Tween）动画

Android 除了支持逐帧动画之外，也提供了对补间（Tween）动画的支持。补间动画就是指开发者只需指定动画开始、动画结束等“关键帧”，而动画变化的“中间帧”由系统计算并补齐。这也是笔者把 Tween 动画翻译为“补间动画”的原因。

7.5.1 Tween 动画与 Interpolator

有过 Flash 设计经验的人应该对补间动画有很好的概念。就笔者的经验来看，Flash 的补间动画支持比 Android 的更简单、功能更强。对于没有 Flash 设计经验的读者，图 7.12 可作为补间动画的示意图。

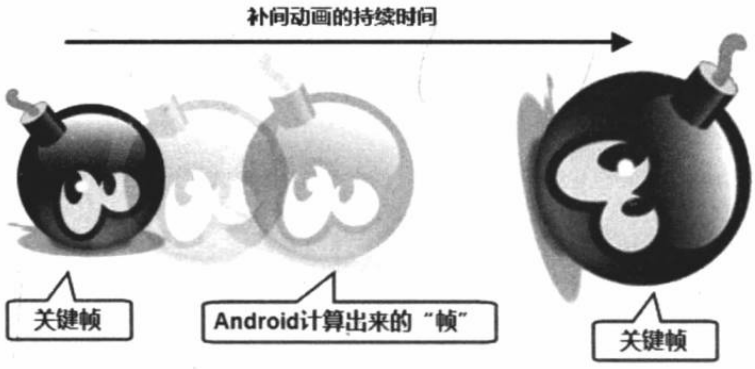


图 7.12 补间动画的示意图

从图 7.12 可以看出，对于补间动画而言，开发者无须“逐一”定义动画过程中的每一帧，他只要定义动画开始、结束的关键帧，并指定动画的持续时间即可。

从图 7.12 可以看出，补间动画所定义的开始帧、结束帧其实只是一些简单的变化，比如图形大小的缩放、旋转角度的改变等。Android 使用 Animation 代表抽象的动画类，它包括如下几个子类。

- AlphaAnimation：透明度改变的动画。创建该动画时要指定动画开始时的透明度、结束时的透明度和动画持续时间。其中透明度可从 0 变化到 1。
- ScaleAnimation：大小缩放的动画。创建该动画时要指定动画开始时的缩放比（以 *X*、*Y* 轴的缩放参数来表示）、结束时的缩放比（以 *X*、*Y* 轴的缩放参数来表示），并指定动画持续时间。由于缩放时以不同点为中心的缩放效果并不相同，因此指定缩放动画时还要通过 pivotX、pivotY 来指定“缩放中心”的坐标。
- TranslateAnimation：位移变化的动画。创建该动画时只要指定动画开始时的位置（以 *X*、*Y* 坐标来表示）、结束时的位置（以 X、Y 坐标来表示），并指定动画持续时间即可。

- RotateAnimation：旋转动画。创建该动画时只要指定动画开始时的旋转角度、结束时的旋转角度，并指定动画持续时间即可。由于旋转时以不同点为中心的旋转效果并不相同，因此指定旋转动画时还要通过 pivotX、pivotY 来指定“旋转轴心”的坐标。

一旦为补间动画指定了三个必要信息，Android 就会根据动画的开始帧、结束帧、动画持续时间计算出需要在中间“补入”多少帧，并计算所有补入帧的图形。当用户浏览补间动画时，他眼中看到的依然是“逐帧动画”。

为了控制在动画期间需要动态“补入”多少帧，具体在动画运行的哪些时刻补入帧，需要借助于 Interpolator。

提示：

Interpolator 在笔者以前念大学时看到有些资料将它翻译为“插值”，这个翻译基本还可以：补间动画定义的是动画开始、结束的关键帧，Android 需要在开始帧、结束帧之间动态地计算、插入大量的帧，而 Interpolator 用于控制“插入帧”的行为，因此翻译为插值也是合适的；现在也有些资料将 Interpolator 翻译得比较生僻，难以理解。为避免读者对各种翻译感到疑惑，本书后面笔者一律使用英文单词 Interpolator，不会强行将它翻译成中文。

Interpolator 根据特定算法计算出整个动画所需要动态插入帧的密度和位置。简单地说，Interpolator 负责控制动画的变化速度，这就使得基本的动画效果（Alpha、Scale、Translate、Rotate）能以匀速变化、加速、减速、抛物线速度等各种速度变化。

Interpolator 是一个接口，它定义了所有 Interpolator 都需要实现的方法：float getInterpolation(float input)，开发者完全可以通过实现 Interpolator 来控制动画的变化速度。

Android 为 Interpolator 提供了如下几个实现类，分别用于实现不同的动画变化速度。

- LinearInterpolator：动画以均匀的速度改变。
- AccelerateInterpolator：在动画开始的地方改变速度较慢，然后开始加速。
- AccelerateDecelerateInterpolator：在动画开始、结束的地方改变速度较慢，在中间的时候加速。
- CycleInterpolator：动画循环播放特定的次数，变化速度按正弦曲线改变。
- DecelerateInterpolator：在动画开始的地方改变速度较快，然后开始减速。

为了在动画资源文件中指定补间动画所使用的 Interpolator，定义补间动画的<set.../>元素支持一个 android:interpolator 属性，该属性的属性值可指定为 Android 默认支持的 Interpolator。例如：

- @android:anim/linear_interpolator
- @android:anim/accelerate_interpolator
- @android:anim/accelerate_decelerate_interpolator
- ……

其实上面的写法很有规律，它们就是把系统提供的 Interpolator 实现类的类名的驼峰写法改为下画线写法即可。

一旦在程序中通过 AnimationUtils 得到了代表补间动画的 Animation 之后，接下来就可调用 View 的 startAnimation(Animation anim)方法开始对该 View 执行动画了。

7.5.2 位置、大小、旋转度、透明度改变的补间动画

虽然 Android 允许在程序中创建 Animation 对象，但实际上一般都会采用动画资源文件来定义补间动画。前面已经介绍过定义补间动画资源文件的格式，此处不再赘述。

下面以一个示例来介绍补间动画。该示例包括两个动画资源文件，其中第一个动画资源文件控制图片以旋转的方式缩小。该动画资源文件如下。

程序清单：codes\07\7.5\TweenAnim\app\src\main\res\anim\anim.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 指定动画匀速改变 -->
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator">
    <!-- 定义缩放变换 -->
    <scale android:fromXScale="1.0"
        android:toXScale="0.01"
        android:fromYScale="1.0"
        android:toYScale="0.01"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="true"
        android:duration="3000"/>
    <!-- 定义透明度的变化 -->
    <alpha
        android:fromAlpha="1"
        android:toAlpha="0.05"
        android:duration="3000"/>
    <!-- 定义旋转变换 -->
    <rotate
        android:fromDegrees="0"
        android:toDegrees="1800"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="3000"/>
</set>
```

上面的动画资源指定动画匀速变化，同时进行缩放、透明度、旋转三种改变，动画持续时间为 3 秒。

第二个动画资源文件则控制图片以动画的方式恢复回来。该动画资源文件如下。

程序清单：codes\07\7.5\TweenAnim\app\src\main\res\anim\reverse.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 指定动画匀速改变 -->
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:startOffset="3000">
    <!-- 定义缩放变换 -->
    <scale android:fromXScale="0.01"
        android:toXScale="1"
        android:fromYScale="0.01"
        android:toYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="true"
        android:duration="3000"/>
    <!-- 定义透明度的变化 -->
    <alpha
        android:fromAlpha="0.05"
        android:toAlpha="1"
        android:duration="3000"/>
```

```
    <!-- 定义旋转变换 -->
    <rotate
        android:fromDegrees="1800"
        android:toDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="3000"/>
</set>
```

定义动画资源之后，接下来就可以利用 AnimationUtils 工具类来加载指定的动画资源了，加载成功后会返回一个 Animation，该对象即可控制图片或视图播放动画。

下面的程序将会负责加载动画资源，并使用 Animation 来控制图片播放动画。

程序清单：codes\07\7.5\TweenAnim\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ImageView flower = (ImageView)
            findViewById(R.id.flower);
        // 加载第一份动画资源
        final Animation anim = AnimationUtils
            .loadAnimation(this, R.anim.anim);
        // 设置动画结束后保留结束状态
        anim.setFillAfter(true);
        // 加载第二份动画资源
        final Animation reverse = AnimationUtils.loadAnimation(this
            , R.anim.reverse);
        // 设置动画结束后保留结束状态
        reverse.setFillAfter(true);
        Button bn = (Button) findViewById(R.id.bn);
        final Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                if (msg.what == 0x123)
                {
                    flower.startAnimation(reverse);
                }
            }
        };
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                flower.startAnimation(anim);
                // 设置 3.5 秒后启动第二个动画
                new Timer().schedule(new TimerTask()
                {
                    @Override
                    public void run()
                    {
                        handler.sendEmptyMessage(0x123);
                    }
                }, 3500);
            }
```

```
        });
    }
}
```

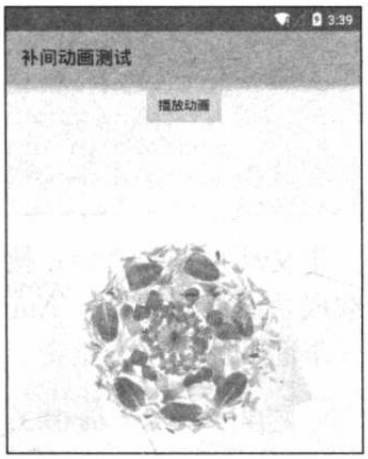


图 7.13　补间动画

正如上面两行粗体字代码所看到的，当用户单击程序中的指定按钮时，程序对 flower 图片播放第一个动画；程序使用定时器设置 3.5 秒后对 flower 图片播放第二个动画。运行该程序，单击按钮将看到程序中间的图片先旋转着缩小、变淡，然后旋转着放大，透明度也逐渐恢复正常，如图 7.13 所示。

实例：蝴蝶飞舞

很多实际的动画往往同时运行两个动画，比如我们要做一个小游戏，需要让用户控制游戏中的主角移动——当主角移动时，不仅要控制它的位置改变，还应该在它移动时播放逐帧动画来让用户感觉更“逼真”。

本实例将会结合逐帧动画和补间动画来开发一个“蝴蝶飞舞”的效果。在这个实例中，蝴蝶飞行时的振翅效果是逐帧动画；蝴蝶飞行时的位置改变是补间动画。

先为该实例定义如下动画资源。

程序清单：codes\07\7.5\Butterfly\app\src\main\res\anim\butterfly.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义动画循环播放 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/butterfly_f01" android:duration="120" />
    <item android:drawable="@drawable/butterfly_f02" android:duration="120" />
    <!-- 下面省略了相似的 item 元素 -->
    ...
</animation-list>
```

定义了上面逐帧动画的动画资源后，接下来在程序中使用一个 ImageView 显示该动画资源，即可看到蝴蝶“振翅”的效果了。由于蝴蝶飞舞主要是位移改变，接下来可以在程序中通过 TranslateAnimation 以动画的方式改变 ImageView 的位置，这样就可实现“蝴蝶飞舞”的效果了。程序如下。

程序清单：codes\07\7.5\butterfly\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    // 记录蝴蝶 ImageView 当前的位置
    private float curX = 0;
    private float curY = 30;
    // 记录蝴蝶 ImageView 下一个位置的坐标
    float nextX = 0;
    float nextY = 0;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取显示蝴蝶的 ImageView 组件
        final ImageView imageView = (ImageView)
            findViewById(R.id.butterfly);
        final Handler handler = new Handler()
        {
```

```
        @Override
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                // 横向上一直向右飞
                if (nextX > 320)
                {
                    curX = nextX = 0;
                }
                else
                {
                    nextX += 8;
                }
                // 纵向上可以随机上下
                nextY = curY + (float) (Math.random() * 10 - 5);
                // 设置显示蝴蝶的 ImageView 发生位移改变
                TranslateAnimation anim = new TranslateAnimation(
                    curX, nextX, curY, nextY);
                curX = nextX;
                curY = nextY;
                anim.setDuration(200);
                // 开始位移动画
                imageView.startAnimation(anim); // ①
            }
        }
    };
    final AnimationDrawable butterfly = (AnimationDrawable)
        imageView.getBackground();
    imageView.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            // 开始播放蝴蝶振翅的逐帧动画
            butterfly.start(); // ②
            // 通过定时器控制每 0.2 秒运行一次 TranslateAnimation 动画
            new Timer().schedule(new TimerTask()
            {
                @Override
                public void run()
                {
                    handler.sendEmptyMessage(0x123);
                }
            }, 0, 200);
        }
    });
}
}
```

上面程序中的①号粗体字代码位于 Handler 的消息处理方法内，这样程序每隔 0.2 秒即对该 ImageView 执行一次位移动画；程序的②号粗体字代码则用于播放 butterfly 动画（蝴蝶振翅效果）。运行上面的程序，单击蝴蝶，即可看到屏幕上有只蝴蝶从左向右飞舞，如图 7.14 所示。

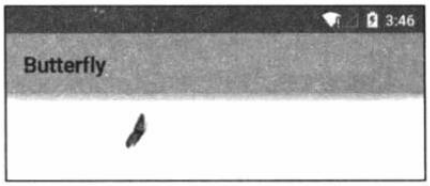


图 7.14 蝴蝶飞舞

7.5.3 自定义补间动画

Android 提供了 Animation 作为补间动画抽象基类，并且为该抽象基类提供了

AlphaAnimation、RotateAnimation、ScaleAnimation、TranslateAnimation 四个实现类，这四个实现类只是补间动画的四种基本形式：透明度改变、旋转、缩放、位移。在实际项目中可能还需要一些更复杂的动画，比如让图片在“三维”空间内进行旋转动画等，这就需要开发者自己开发补间动画了。

自定义补间动画并不难，需要继承 Animation，继承 Animation 时关键是要重写该抽象基类的 applyTransformation(float interpolatedTime, Transformation t)方法，该方法中两个参数说明如下。

- interpolatedTime：代表了动画的时间进行比。不管动画实际的持续时间如何，当动画播放时，该参数总是自动从 0 变化到 1。
- Transformation：代表了补间动画在不同时刻对图形或组件的变形程度。

从上面的介绍可以看出，实现自定义动画的关键就在于重写 applyTransformation()方法时，根据 interpolatedTime 时间来动态地计算动画对图片或视图的变形程度。

Transformation 代表了对图片或视图的变形程度，该对象里封装了一个 Matrix 对象，对它所包装的 Matrix 进行位移、倾斜、旋转等变换时，Transformation 将会控制对应的图片或视图进行相应的变换。

为了控制图片或视图进行三维空间的变换，还需要借助于 Android 提供的一个 Camera，这个 Camera 并非代表手机的摄像头，而是一个空间变换工具，作用有点类似于 Matrix，但功能更强大。

Camera 提供了如下常用的方法。

- getMatrix(Matrix matrix)：将 Camera 所做的变换应用到指定 matrix 上。
- rotateX(float deg)：使目标组件沿 *X* 轴旋转。
- rotateY(float deg)：使目标组件沿 *Y* 轴旋转。
- rotateZ(float deg)：使目标组件沿 *Z* 轴旋转。
- translate(float x, float y, float z)：使目标组件在三维空间里进行位移变换。
- applyToCanvas(Canvas canvas)：把 Camera 所做的变换应用到 Canvas 上。

从上面的方法可以看出，Camera 主要用于支持三维空间的变换，那么手机中三维空间的坐标系统是怎样的呢？图 7.15 显示了手机屏幕上的三维坐标系统。

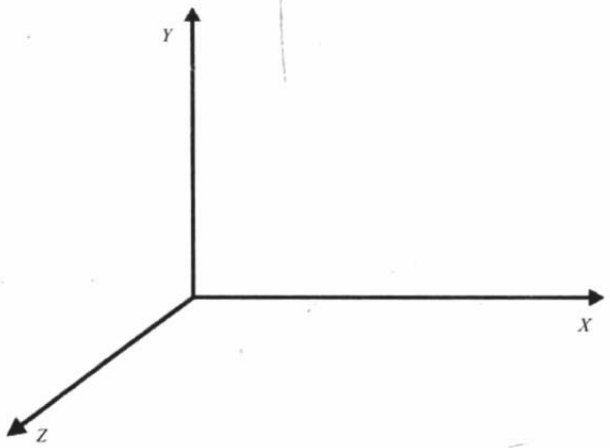


图 7.15 手机屏幕上的三维坐标系统

当 Camera 控制图片或 View 沿 *X*、*Y* 或 *Z* 轴旋转时，被旋转的图片或 View 将会呈现出三维透视的效果。图 7.16 显示了一张图片沿着 *Y* 轴旋转的效果。

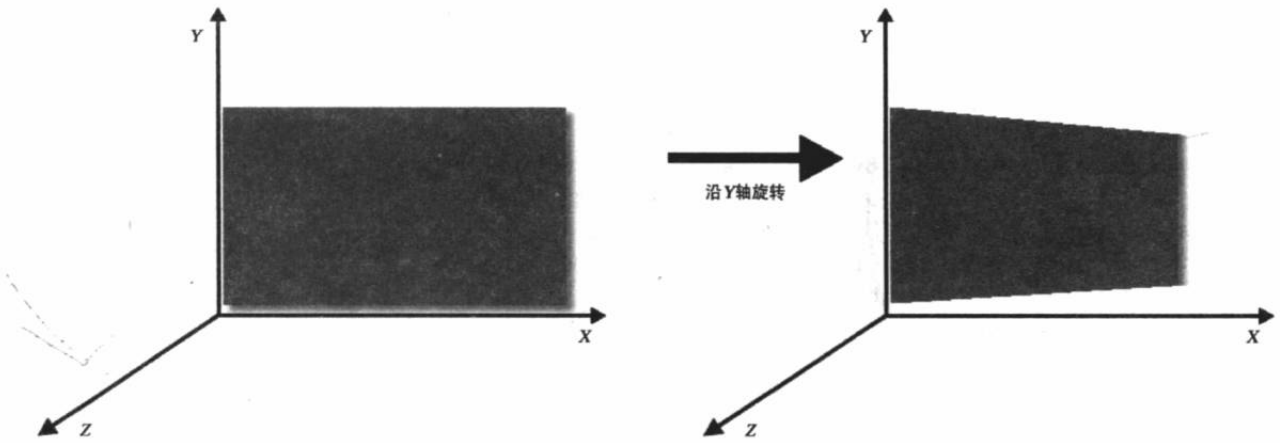


图 7.16 在三维空间内沿 Y 轴旋转的示意图

提示：

关于三维透视理论的知识，此处限于篇幅不便深入解释，如果读者有 3ds Max 或 Maya 之类的设计经验，应该很容易理解三维透视的基本理论；否则建议读者自行阅读相关内容。

下面程序将会利用 Camera 来自定义在三维空间的动画，该程序的自定义动画类的代码如下。

程序清单：\codes\07\7.5\ListViewTween\app\src\main\java\org\crazyit\image\MyAnimation.java

```
public class MyAnimation extends Animation
{
    private float centerX;
    private float centerY;
    // 定义动画的持续时间
    private int duration;
    private Camera camera = new Camera();
    public MyAnimation(float x, float y, int duration)
    {
        this.centerX = x;
        this.centerY = y;
        this.duration = duration;
    }
    @Override
    public void initialize(int width, int height
        , int parentWidth, int parentHeight)
    {
        super.initialize(width, height, parentWidth, parentHeight);
        // 设置动画的持续时间
        setDuration(duration);
        // 设置动画结束后效果保留
        setFillAfter(true);
        setInterpolator(new LinearInterpolator());
    }
    /*
     * 该方法的 interpolatedTime 代表了抽象的动画持续时间，不管动画实际持续时间多长，
     * interpolatedTime 参数总是从 0（动画开始时）变化到 1（动画结束时）
     * Transformation 参数代表了对目标组件所做的改变
     */
    @Override
    protected void applyTransformation(float interpolatedTime
        , Transformation t)
    {
        camera.save();
```

```
        // 根据 interpolatedTime 时间来控制 X、Y、Z 上的偏移
        camera.translate(100.0f - 100.0f * interpolatedTime,
                150.0f * interpolatedTime - 150,
                80.0f - 80.0f * interpolatedTime);
        // 设置根据 interpolatedTime 时间在 Y 轴上旋转不同角度
        camera.rotateY(360 * (interpolatedTime));
        // 设置根据 interpolatedTime 时间在 X 轴上旋转不同角度
        camera.rotateX((360 * interpolatedTime));
        // 获取 Transformation 参数的 Matrix 对象
        Matrix matrix = t.getMatrix();
        camera.getMatrix(matrix);
        matrix.preTranslate(-centerX, -centerY);
        matrix.postTranslate(centerX, centerY);
        camera.restore();
    }
}
```

上面的程序中自定义动画的关键就是两行粗体字代码，这两行代码将会根据动画的进行时间来控制 View 在 *X*、*Y* 轴上的旋转——这就会具有在三维空间内旋转的效果。

提供了上面的自定义动画类之后，接下来既可用该自定义动画类来控制图片，也可控制 View 组件，因为动画就是通过调用 View 的 startAnimation(Animation anim)来启动的。下面是使用 MyAnimation 执行动画的程序代码。

程序清单：codes\07\7.5\ListViewTween\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 ListView 组件
        ListView list = (ListView) findViewById(R.id.list);
        WindowManager windowManager = (WindowManager)
            getSystemService(WINDOW_SERVICE);
        Display display = windowManager.getDefaultDisplay();
        DisplayMetrics metrice = new DisplayMetrics();
        // 获取屏幕的宽和高
        display.getMetrics(metrice);
        // 设置对 ListView 组件应用动画
        list.setAnimation(new MyAnimation(metrice.xdpi / 2
            , metrice.ydpi / 2, 3500));
    }
}
```

上面程序中的粗体字代码对 ListView 使用 MyAnimation 播放动画，这意味着程序中 ListView 将会以三维变换的动画方式出现，如图 7.17 所示。

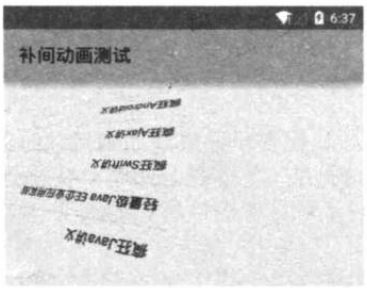


图 7.17 三维空间变换的动画

7.6 属性动画

前面介绍 Android 资源时已经提到了属性动画，从某种角度来看，属性动画是增强版的补间动画，属性动画的强大可以体现在如下两方面。

- 补间动画只能定义两个关键帧在“透明度”、“旋转”、“缩放”、“位移”4 个方面的变化，但属性动画可以定义任何属性的变化。
- 补间动画只能对 UI 组件执行动画，但属性动画几乎可以对任何对象执行动画（不管它是否显示在屏幕上）。

与补间动画类似的是，属性动画也需要定义如下几个属性。

- 动画持续时间。该属性的默认值是 300 ms。在属性动画资源文件中通过 android:duration 属性指定。
- 动画插值方式。该属性的作用与补间动画中插值属性的作用基本类似。在属性动画资源文件中通过 android:interpolator 属性指定。
- 动画重复次数。指定动画重复播放的次数。在属性动画资源文件中通过 android:repeatCount 属性指定。
- 重复行为。指定动画播放结束后、重复下次动画时，是从开始帧再次播放到结束帧，还是从结束帧反向播放到开始帧。在属性动画资源文件中通过 android:repeatMode 属性指定。
- 动画集。开发者可以将多个属性动画合并成一组，既可让这组属性动画按次序播放，也可让这组属性动画同时播放。在属性动画资源文件中通过<set..../>元素来组合，该元素的 android:ordering 属性指定该组动画是按次序播放，还是同时播放。
- 帧刷新频率。指定每隔多长时间播放一帧。该属性的默认值为 10 ms。

7.6.1 属性动画的 API

属性动画涉及的 API 如下。

- Animator：它提供了创建属性动画的基类，基本上不会直接使用该类。通常该类只用于被继承并重写它的相关方法。
- ValueAnimator：属性动画主要的时间引擎，它负责计算各个帧的属性值。它定义了属性动画的绝大部分核心功能，包括计算各帧的相关属性值，负责处理更新事件，按属性值的类型控制计算规则。属性动画主要由两方面组成：①计算各帧的相关属性值；②为指定对象设置这些计算后的值。ValueAnimator 只负责第一方面内容，因此程序员必须根据 ValueAnimator 计算并监听值更新来更新对象的相关属性值。
- ObjectAnimator：它是 ValueAnimator 的子类，允许程序员对指定对象的属性执行动画。在实际应用中，ObjectAnimator 使用起来更加简单，因此更加常用。在少数场景下，由于 ObjectAnimator 存在一些限制，可能需要考虑使用 ValueAnimator。
- AnimatorSet：它是 Animator 的子类，用于组合多个 Animator，并指定多个 Animator 是按次序播放，还是同时播放。

除此之外，属性动画还需要利用一个 Evaluator（计算器），该工具类控制属性动画如何计算属性值。Android 提供了如下 Evaluator。

- IntEvaluator：用于计算 int 类型属性值的计算器。
- FloatEvaluator：用于计算 float 类型属性值的计算器。

- ArgbEvaluator：用于计算以十六进制形式表示的颜色值的计算器。
- TypeEvaluator：它是计算器接口，开发者可以通过实现该接口来实现自定义计算器。如果需要对 int、float 或者颜色值以外类型的属性执行属性动画，可能需要实现 TypeEvaluator 接口来实现自定义计算器。

1. 使用 ValueAnimator 创建动画

使用 ValueAnimator 创建动画可按如下 4 个步骤进行。

1. 调用 ValueAnimator 的 ofInt()、ofFloat()或 ofObject()静态方法创建 ValueAnimator 实例。
2. 调用 ValueAnimator 的 setXxx()方法设置动画持续时间、插值方式、重复次数等。
3. 调用 ValueAnimator 的 start()方法启动动画。
4. 为 ValueAnimator 注册 AnimatorUpdateListener 监听器，在该监听器中可以监听 ValueAnimator 计算出来的值的改变，并将这些值应用到指定对象上。

例如如下代码片段：

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);
animation.setDuration(1000);
animation.start();
```

上面的例子实现了在 1000ms 内，值从 0 到 1 的变化。

除此之外，开发者也可以提供一个自定义的 Evaluator 计算器，例如如下代码：

```
ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(),
    startVal, endVal)
animation.setDuration(1000);
animation.start();
```

在上面的代码片段中，ValueAnimator 仅仅是计算动画过程中变化的值，并没有把这些计算出来的值应用到任何对象上，因此也不会显示任何动画。

如果希望使用 ValueAnimator 创建动画，还需要注册一个监听器：AnimatorUpdateListener，该监听器负责更新对象的属性值。在实现这个监听器时，可以通过 getAnimatedValue()方法来获取当前帧的值，并将该计算出来的值应用到指定对象上。当该对象的属性持续改变时，该对象也就呈现出动画效果了。

2. 使用 ObjectAnimator 创建动画

ObjectAnimator 继承了 ValueAnimator，因此它可以直接将 ValueAnimator 在动画过程中计算出来的值应用到指定对象的指定属性上（ValueAnimator 则需要注册一个监听器来完成这个工作）。因此使用 ObjectAnimator 就不需要注册 AnimatorUpdateListener 监听器了。

使用 ObjectAnimator 的 ofInt()、ofFloat()或 ofObject()静态方法创建 ObjectAnimator 时，需要指定具体的对象，以及对象的属性名。

例如如下代码片段：

```
ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);
anim.setDuration(1000);
anim.start();
```

与 ValueAnimator 不同的是，使用 ObjectAnimator 有如下几个注意点。

- 要为该对象对应的属性提供 setter 方法，如上例中需要为 foo 对象提供 setAlpha(float value)方法。
- 调用 ObjectAnimator 的 ofInt()、ofFloat()或 ofObject()工厂方法时，如果 values... 参数只提供了一个值（本来需要提供开始值和结束值），那么该值会被认为是结束值。该

对象应该为该属性提供一个 getter 方法，该 getter 方法的返回值将被作为开始值。

- 如果动画的对象是 View，为了能显示动画效果，可能还需要在 onAnimationUpdate()事件监听方法中调用 View.invalidate()方法来刷新屏幕的显示，比如对 Drawable 对象的 color 属性执行动画。但 View 定义的 setter 方法，如 setAlpha()和 setTranslationX()等方法，都会自动地调用 invalidate()方法，因此不需要额外地调用 invalidate()方法。

7.6.2 使用属性动画

属性动画既可作用于 UI 组件，也可作用于普通的对象（即使它没有在 UI 界面上绘制出来）。

定义属性动画有如下两种方式。

- 使用 ValueAnimator 或 ObjectAnimator 的静态工厂方法来创建动画。
- 使用资源文件来定义动画。

使用属性动画的步骤如下。

❶ 创建 ValueAnimator 或 ObjectAnimator 对象——既可从 XML 资源文件加载该动画资源，也可直接调用 ValueAnimator 或 ObjectAnimator 的静态工厂方法来创建动画。

❷ 根据需要为 Animator 对象设置属性。

❸ 如果需要监听 Animator 的动画开始事件、动画结束事件、动画重复事件、动画值改变事件，并根据事件提供相应的处理代码，则应该为 Animator 对象设置事件监听器。

❹ 如果有多个动画需要按次序或同时播放，则应使用 AnimatorSet 组合这些动画。

❺ 调用 Animator 对象的 start()方法启动动画。

下面的示例示范了如何利用属性动画来控制“小球”掉落动画。该示例会监听用户在屏幕上的“触屏”时间，程序会在屏幕的触摸点绘制一个小球，并用动画控制该小球向下掉落。

该示例的界面布局文件非常简单，界面布局文件中只有一个 LinearLayout，因此此处不再给出界面布局文件。下面是该示例的 Activity 代码。

程序清单：codes\07\7.6\AnimatorTest\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义小球大小的常量
    static final float BALL_SIZE = 50F;
    // 定义小球从屏幕上方下落到屏幕底端的总时间
    static final float FULL_TIME = 1000;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout container = (LinearLayout)
            findViewById(R.id.container);
        // 设置该窗口显示 MyAnimationView 组件
        container.addView(new MyAnimationView(this));
    }
    public class MyAnimationView extends View
        implements AnimatorUpdateListener
    {
        public final ArrayList<ShapeHolder> balls
            = new ArrayList<ShapeHolder>();
        public MyAnimationView(Context context)
        {
            super(context);
            setBackgroundColor(Color.WHITE);
```

```
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 如果触碰事件不是按下、移动事件
        if (event.getAction() != MotionEvent.ACTION_DOWN
            && event.getAction() != MotionEvent.ACTION_MOVE)
        {
            return false;
        }
        // 在事件发生点添加一个小球（用一个圆形代表）
        ShapeHolder newBall = addBall(event.getX(), event.getY());
        // 计算小球下落动画开始时的 y 坐标
        float startY = newBall.getY();
        // 计算小球下落动画结束时的 y 坐标（落到屏幕最下方，就是屏幕高度减去小球高度）
        float endY = getHeight() - BALL_SIZE;
        // 获取屏幕高度
        float h = (float) getHeight();
        float eventY = event.getY();
        // 计算动画的持续时间
        int duration = (int) (FULL_TIME * ((h - eventY) / h));
        // 定义小球“落下”的动画：
        // 让 newBall 对象的 y 属性从事件发生点变化到屏幕最下方
        ValueAnimator fallAnim = ObjectAnimator.ofFloat(
            newBall, "y", startY, endY);
        // 设置 fallAnim 动画的持续时间
        fallAnim.setDuration(duration);
        // 设置 fallAnim 动画的插值方式：加速插值
        fallAnim.setInterpolator(new AccelerateInterpolator());
        // 为 fallAnim 动画添加监听器
        // 当 ValueAnimator 的属性值发生改变时，将会激发该监听器的事件监听方法
        fallAnim.addUpdateListener(this);
        // 定义对 newBall 对象的 alpha 属性执行从 1 到 0 的动画（即定义渐隐动画）
        ObjectAnimator fadeAnim = ObjectAnimator.ofFloat(newBall
            , "alpha", 1f, 0f);
        // 设置动画持续时间
        fadeAnim.setDuration(250);
        // 为 fadeAnim 动画添加监听器
        fadeAnim.addListener(new AnimatorListenerAdapter()
        {
            // 当动画结束时
            @Override
            public void onAnimationEnd(Animator animation)
            {
                // 动画结束时将该动画关联的 ShapeHolder 删除
                balls.remove(((ObjectAnimator) animation).getTarget());
            }
        });
        // 为 fadeAnim 动画添加监听器
        // 当 ValueAnimator 的属性值发生改变时，将会激发该监听器的事件监听方法
        fadeAnim.addUpdateListener(this);
        // 定义一个 AnimatorSet 来组合动画
        AnimatorSet animatorSet = new AnimatorSet();
        // 指定在播放 fadeAnim 之前，先播放 fallAnim 动画
        animatorSet.play(fallAnim).before(fadeAnim);
        // 开始播放动画
        animatorSet.start();
        return true;
    }
    private ShapeHolder addBall(float x, float y)
    {
        // 创建一个椭圆
```

```
            OvalShape circle = new OvalShape();
            // 设置该椭圆的宽、高
            circle.resize(BALL_SIZE, BALL_SIZE);
            // 将椭圆包装成 Drawable 对象
            ShapeDrawable drawable = new ShapeDrawable(circle);
            // 创建一个 ShapeHolder 对象
            ShapeHolder shapeHolder = new ShapeHolder(drawable);
            // 设置 ShapeHolder 的 x、y 坐标
            shapeHolder.setX(x - BALL_SIZE / 2);
            shapeHolder.setY(y - BALL_SIZE / 2);
            int red = (int) (Math.random() * 255);
            int green = (int) (Math.random() * 255);
            int blue = (int) (Math.random() * 255);
            // 将 red、green、blue 三个随机数组合成 ARGB 颜色
            int color = 0xff000000 + red << 16 | green << 8 | blue;
            // 获取 drawable 上关联的画笔
            Paint paint = drawable.getPaint();
            // 将 red、green、blue 三个随机数除以 4 得到商值组合成 ARGB 颜色
            int darkColor = 0xff000000 | red / 4 << 16
                | green / 4 << 8 | blue / 4;
            // 创建圆形渐变
            RadialGradient gradient = new RadialGradient(
                37.5f, 12.5f, BALL_SIZE, color, darkColor
                , Shader.TileMode.CLAMP);
            paint.setShader(gradient);
            // 为 shapeHolder 设置 paint 画笔
            shapeHolder.setPaint(paint);
            balls.add(shapeHolder);
            return shapeHolder;
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            // 遍历 balls 集合中的每个 ShapeHolder 对象
            for (ShapeHolder shapeHolder : balls)
            {
                // 保存 canvas 的当前坐标系统
                canvas.save();
                // 坐标变换：将画布坐标系统平移到 shapeHolder 的 X、Y 坐标处
                canvas.translate(shapeHolder.getX()
                    , shapeHolder.getY());
                // 将 shapeHolder 持有的圆形绘制在 Canvas 上
                shapeHolder.getShape().draw(canvas);
                // 恢复 Canvas 坐标系统
                canvas.restore();
            }
        }
        @Override
        public void onAnimationUpdate(ValueAnimator animation)
        {
            // 指定重绘该界面
            this.invalidate(); // ①
        }
    }
}
```

上面的程序中第一段粗体字代码创建了两个 ObjectAnimator 对象，其中第一个 ObjectAnimator 控制小球的“下落”；第二个 ObjectAnimator 则用于控制该小球以“渐隐”的方式隐藏——程序为 fadeAnim 绑定了事件监听器，当 fadeAnim 动画播放完成时，程序将小球删除。

上面程序的属性动画并没有控制 UI 组件，而是对一个自定义 ShapeHolder 对象起作用。为了能让 UI 界面不断刷新，看到 UI 界面上小球的动画效果，程序为两个动画都绑定了 AnimatorUpdateListener 监听器，该监听器用于实现当目标对象（小球）的属性发生改变时，该组件会重绘界面——如以上程序中①号代码所示。

上面的程序中第三段粗体字代码用于创建一个圆形渐变，创建圆形渐变的起始颜色的 red、green、blue 值都是随机的，结束颜色值是起始颜色值的四分之一。读者可能对计算颜色值的代码有点疑惑：

```
int color = 0xff000000 + red << 16 | green << 8 | blue;
```

上面的代码其实很简单，解释如下。

- 0xff000000 代表透明度为 ff，也就是完全不透明。
- red 代表一个 0~255（0~ff）的随机整数，但这个整数要添加到 0xff**00**0000 中加粗的两个“位”上，也就是要将 red 的值左移（16 位，对应为十六进制数的 4 位），这就是 red<<16 的原因。
- green 代表一个 0~255（0~ff）的随机整数，但这个整数要添加到 0xff00**00**00 中加粗的两个“位”上，也就是要将 green 的值左移（8 位，对应为十六进制数的 2 位），这就是 red<<8 的原因。
- blue 代表一个 0~255（0~ff）的随机整数，但这个整数要添加到 0xff0000**00** 中加粗的两个“位”上，因此 blue 值就不需要位移了。

将 red、green、blue 值位移后的结果加起来就得到了实际的颜色值，但为了有更好的计算性能，本代码直接使用按位或（|）来累加这些值。

上面的示例还用到了一个 ShapeHolder 类，该类只是负责包装 ShapeDrawable 对象，并为 x、y、width、height、alpha 等属性提供 setter、getter 方法，以方便 ObjectAnimator 动画控制它。下面是 ShapeHolder 类的代码。

程序清单：codes\07\7.6\AnimatorTest\app\src\main\java\org\crazyit\image\ShapeHolder.java

```
public class ShapeHolder
{
    private float x = 0, y = 0;
    private ShapeDrawable shape;
    private int color;
    private RadialGradient gradient;
    private float alpha = 1f;
    private Paint paint;
    public ShapeHolder(ShapeDrawable s)
    {
        shape = s;
    }
    // 省略各成员变量的 setter 和 getter 方法
    ...
}
```

运行该程序，用户在屏幕上拖动手指时，将可以看到如图 7.18 所示的小球下落的动画。

如果为小球增加更多的动画，让小球落到底端时产生弹跳动画，并且再次弹起来，则可以使该示例更加完善。

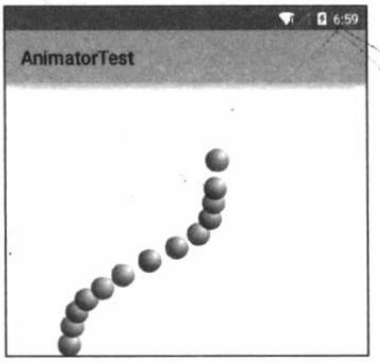


图 7.18 小球下落

实例：大珠小珠落玉盘

该实例是对上一个示例的改进，主要是为小球增加了几个动画，控制小球落到底端时小球被压扁，小球会再次弹起，这样就可以开发出“大珠小珠落玉盘”的弹跳动画。

该实例的 Activity 代码如下。

程序清单：codes\07\7.6\BouncingBalls\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义小球大小的常量
    static final float BALL_SIZE = 50F;
    // 定义小球从屏幕上方下落到屏幕底端的总时间
    static final float FULL_TIME = 1000;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout container = (LinearLayout)
            findViewById(R.id.container);
        // 设置该窗口显示 MyAnimationView 组件
        container.addView(new MyAnimationView(this));
    }
    public class MyAnimationView extends View
    {
        public final ArrayList<ShapeHolder> balls
            = new ArrayList<ShapeHolder>();
        public MyAnimationView(Context context)
        {
            super(context);
            // 加载动画资源
            ObjectAnimator colorAnim = (ObjectAnimator) AnimatorInflater
                .loadAnimator(MainActivity.this, R.animator.color_anim);
            colorAnim.setEvaluator(new ArgbEvaluator());
            // 对该 View 本身应用属性动画
            colorAnim.setTarget(this);
            // 开始指定动画
            colorAnim.start();
        }
        @Override
        public boolean onTouchEvent(MotionEvent event)
        {
            // 如果触碰事件不是按下、移动事件
            if (event.getAction() != MotionEvent.ACTION_DOWN
                && event.getAction() != MotionEvent.ACTION_MOVE)
            {
                return false;
            }
            // 在事件发生点添加一个小球（用一个圆形代表）
            ShapeHolder newBall = addBall(event.getX(), event.getY());
            // 计算小球下落动画开始时的 y 坐标
            float startY = newBall.getY();
            // 计算小球下落动画结束时的 y 坐标（落到屏幕最下方，就是屏幕高度减去小球高度）
            float endY = getHeight() - BALL_SIZE;
            // 获取屏幕高度
            float h = (float) getHeight();
            float eventY = event.getY();
            // 计算动画的持续时间
            int duration = (int) (FULL_TIME * ((h - eventY) / h));
            // 定义小球“落下”的动画
```

```
// 让 newBall 对象的 y 属性从事件发生点变化到屏幕最下方
ValueAnimator fallAnim = ObjectAnimator.ofFloat(
    newBall, "y", startY, endY);
// 设置 fallAnim 动画的持续时间
fallAnim.setDuration(duration);
// 设置 fallAnim 动画的插值方式：加速插值
fallAnim.setInterpolator(new AccelerateInterpolator());
// 定义小球“压扁”的动画：该动画控制小球的 x 坐标“左移”半个球
ValueAnimator squashAnim1 = ObjectAnimator.ofFloat(newBall
    , "x", newBall.getX(), newBall.getX() - BALL_SIZE / 2);
// 设置 squashAnim1 动画的持续时间
squashAnim1.setDuration(duration / 4);
// 设置 squashAnim1 动画重复 1 次
squashAnim1.setRepeatCount(1);
// 设置 squashAnim1 动画的重复方式
squashAnim1.setRepeatMode(ValueAnimator.REVERSE);
// 设置 squashAnim1 动画的插值方式：减速插值
squashAnim1.setInterpolator(new DecelerateInterpolator());
// 定义小球“压扁”的动画：该动画控制小球的宽度加倍
ValueAnimator squashAnim2 = ObjectAnimator.ofFloat(newBall,
    "width", newBall.getWidth()
    , newBall.getWidth() + BALL_SIZE);
// 设置 squashAnim2 动画的持续时间
squashAnim2.setDuration(duration / 4);
// 设置 squashAnim2 动画重复 1 次
squashAnim2.setRepeatCount(1);
// 设置 squashAnim2 动画的重复方式
squashAnim2.setRepeatMode(ValueAnimator.REVERSE);
// 设置 squashAnim2 动画的插值方式：减速插值
squashAnim2.setInterpolator(new DecelerateInterpolator());
// 定义小球“拉伸”的动画：该动画控制小球的 y 坐标“下移”半个球
ObjectAnimator stretchAnim1 = ObjectAnimator.ofFloat(newBall
    , "y", endY, endY + BALL_SIZE / 2);
// 设置 stretchAnim1 动画的持续时间
stretchAnim1.setDuration(duration / 4);
// 设置 stretchAnim1 动画重复 1 次
stretchAnim1.setRepeatCount(1);
// 设置 stretchAnim1 动画的重复方式
stretchAnim1.setRepeatMode(ValueAnimator.REVERSE);
// 设置 stretchAnim1 动画的插值方式：减速插值
stretchAnim1.setInterpolator(new DecelerateInterpolator());
// 定义小球“拉伸”的动画：该动画控制小球的高度减半
ValueAnimator stretchAnim2 = ObjectAnimator.ofFloat(newBall,
    "height", newBall.getHeight()
    , newBall.getHeight() - BALL_SIZE / 2);
// 设置 stretchAnim2 动画的持续时间
stretchAnim2.setDuration(duration / 4);
// 设置 squashAnim2 动画重复 1 次
stretchAnim2.setRepeatCount(1);
// 设置 squashAnim2 动画的重复方式
stretchAnim2.setRepeatMode(ValueAnimator.REVERSE);
// 设置 squashAnim2 动画的插值方式：减速插值
stretchAnim2.setInterpolator(new DecelerateInterpolator());
// 定义小球“弹起”的动画
ObjectAnimator bounceBackAnim = ObjectAnimator.ofFloat(
    newBall , "y", endY, startY);
// 设置持续时间
bounceBackAnim.setDuration(duration);
// 设置动画的插值方式：减速插值
bounceBackAnim.setInterpolator(new DecelerateInterpolator());
// 使用 AnimatorSet 按顺序播放“下落/压扁&拉伸/弹起动画
```

```
        AnimatorSet bouncer = new AnimatorSet();
        // 定义在 squashAnim1 动画之前播放 fallAnim 下落动画
        bouncer.play(fallAnim).before(squashAnim1);
        // 由于小球在“屏幕”下方弹起时，小球要被压扁
        // 即：宽度加倍、x 坐标左移半个球，高度减半、y 坐标下移半个球
        // 因此此处指定播放 squashAnim1 的同时
        // 还播放 squashAnim2、stretchAnim1、stretchAnim2
        bouncer.play(squashAnim1).with(squashAnim2);
        bouncer.play(squashAnim1).with(stretchAnim1);
        bouncer.play(squashAnim1).with(stretchAnim2);
        // 指定播放 stretchAnim2 动画之后，播放 bounceBackAnim 弹起动画
        bouncer.play(bounceBackAnim).after(stretchAnim2);
        // 定义对 newBall 对象的 alpha 属性执行从 1 到 0 的动画（即定义渐隐动画）
        ObjectAnimator fadeAnim = ObjectAnimator.ofFloat(newBall
            , "alpha", 1f, 0f);
        // 设置动画持续时间
        fadeAnim.setDuration(250);
        // 为 fadeAnim 动画添加监听器
        fadeAnim.addListener(new AnimatorListenerAdapter()
        {
            // 当动画结束时
            @Override
            public void onAnimationEnd(Animator animation)
            {
                // 动画结束时将该动画关联的 ShapeHolder 删除
                balls.remove(((ObjectAnimator) animation).getTarget());
            }
        });
        // 再次定义一个 AnimatorSet 来组合动画
        AnimatorSet animatorSet = new AnimatorSet();
        // 指定在播放 fadeAnim 之前，先播放 bouncer 动画
        animatorSet.play(bouncer).before(fadeAnim);
        // 开始播放动画
        animatorSet.start();
        return true;
    }
    private ShapeHolder addBall(float x, float y)
    {
        // 创建一个椭圆
        OvalShape circle = new OvalShape();
        // 设置该椭圆的宽、高
        circle.resize(BALL_SIZE, BALL_SIZE);
        // 将椭圆包装成 Drawable 对象
        ShapeDrawable drawable = new ShapeDrawable(circle);
        // 创建一个 ShapeHolder 对象
        ShapeHolder shapeHolder = new ShapeHolder(drawable);
        // 设置 ShapeHolder 的 x、y 坐标
        shapeHolder.setX(x - BALL_SIZE / 2);
        shapeHolder.setY(y - BALL_SIZE / 2);
        int red = (int) (Math.random() * 255);
        int green = (int) (Math.random() * 255);
        int blue = (int) (Math.random() * 255);
        // 将 red、green、blue 三个随机数组合成 ARGB 颜色
        int color = 0xff000000 + red << 16 | green << 8 | blue;
        // 获取 drawable 上关联的画笔
        Paint paint = drawable.getPaint();
        // 将 red、green、blue 三个随机数除以 4 得到商值组合成 ARGB 颜色
        int darkColor = 0xff000000 | red / 4 << 16
            | green / 4 << 8 | blue / 4;
        // 创建圆形渐变
        RadialGradient gradient = new RadialGradient(
            37.5f, 12.5f, BALL_SIZE, color, darkColor
```

```
                , Shader.TileMode.CLAMP);
            paint.setShader(gradient);
            // 为 shapeHolder 设置 paint 画笔
            shapeHolder.setPaint(paint);
            balls.add(shapeHolder);
            return shapeHolder;
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            // 遍历 balls 集合中的每个 ShapeHolder 对象
            for (ShapeHolder shapeHolder : balls)
            {
                // 保存 canvas 的当前坐标系统
                canvas.save();
                // 坐标变换：将画布坐标系统平移到 shapeHolder 的 X、Y 坐标处
                canvas.translate(shapeHolder.getX()
                    , shapeHolder.getY());
                // 将 shapeHolder 持有的圆形绘制在 Canvas 上
                shapeHolder.getShape().draw(canvas);
                // 恢复 Canvas 坐标系统
                canvas.restore();
            }
        }
    }
}
```

上面的实例对小球使用了更多的动画，因此小球下落后，会再次弹起。

该实例的 ShapeHolder 需要对 width、height 增加动画，因此该 ShapeHolder 需要增加对 width、height 的 setter 和 getter 方法。下面是该实例中 ShapeHolder 的代码。

程序清单：codes\07\7.6\BouncingBalls\app\src\main\java\org\crazyit\image\ShapeHolder.java

```
public class ShapeHolder
{
    private float x = 0, y = 0;
    private ShapeDrawable shape;
    private int color;
    private RadialGradient gradient;
    private float alpha = 1f;
    private Paint paint;
    public ShapeHolder(ShapeDrawable s)
    {
        shape = s;
    }
    public float getWidth()
    {
        return shape.getShape().getWidth();
    }
    public void setWidth(float width)
    {
        Shape s = shape.getShape();
        s.resize(width, s.getHeight());
    }
    public float getHeight()
    {
        return shape.getShape().getHeight();
    }
    public void setHeight(float height)
    {
        Shape s = shape.getShape();
        s.resize(s.getWidth(), height);
    }
```

```
    // 省略其他成员变量的 setter 和 getter 方法
    ...
}
```

运行该实例，可以看到小球在底端压扁并弹起的动画，如图 7.19 所示。

图 7.19 大珠小珠落玉盘

7.7 使用 SurfaceView 实现动画

虽然前面大量介绍了使用自定义 View 来进行绘图，但 View 的绘图机制存在如下缺陷。

- View 缺乏双缓冲机制。
- 当程序需要更新 View 上的图片时，程序必须重绘 View 上显示的整张图片。
- 新线程无法直接更新 View 组件。

由于 View 存在上述缺陷，所以通过自定义 View 来实现绘图，尤其是游戏中的绘图时性能并不好。Android 提供了一个 SurfaceView 来代替 View，在实现游戏绘图方面，SurfaceView 比 View 更加出色，因此一般推荐使用 SurfaceView。

7.7.1 SurfaceView 的绘图机制

SurfaceView 一般会与 SurfaceHolder 结合使用，SurfaceHolder 用于向与之关联的 SurfaceView 上绘图，调用 SurfaceView 的 getHolder()方法即可获取 SurfaceView 关联的 SurfaceHolder。

SurfaceHolder 提供了如下方法来获取 Canvas 对象。

- Canvas lockCanvas()：锁定整个 SurfaceView 对象，获取该 SurfaceView 上的 Canvas。
- Canvas lockCanvas(Rect dirty)：锁定 SurfaceView 上 Rect 划分的区域，获取该 SurfaceView 上的 Canvas。

当对同一个 SurfaceView 调用上面两个方法时，两个方法所返回的是同一个 Canvas 对象。但当程序调用第二个方法获取指定区域的 Canvas 时，SurfaceView 将只对 Rect 所“圈”出来的区域进行更新，通过这种方式可以提高画面的更新速度。

当通过 lockCanvas()获取了指定 SurfaceView 上的 Canvas 之后，接下来程序就可以调用 Canvas 进行绘图了，Canvas 绘图完成后通过如下方法来释放绘图、提交所绘制的图形。

- unlockCanvasAndPost(canvas)

需要指出的是，当调用 SurfaceHolder 的 unlockCanvasAndPost()方法之后，该方法之前所绘制的图形还处于缓冲区中，下一次 lockCanvas()方法锁定的区域可能会“遮挡”它。

下面的程序示范了 SurfaceView 的绘图机制。该程序将会采用 SurfaceView 绘制一个游戏动画：程序通过继承 SurfaceView 来实现一个自定义 View。下面是该自定义 View 的代码。

程序清单：codes\07\7.7\SurfaceViewTest\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class FishView extends SurfaceView
        implements SurfaceHolder.Callback
{
    private SurfaceHolder holder;
    private UpdateViewThread updateThread;
    private boolean hasSurface;
    private Bitmap back;
    private Bitmap[] fishs;
    private int fishIndex = 0; // 定义变量记录绘制第几张鱼的图片
```

```
// 下面定义两个变量，记录鱼的初始位置
private float fishX = 778;
private float fishY = 500;
private float fishSpeed = 6; // 鱼的游动速度
// 定义鱼游动的角度
private int fishAngle = new Random().nextInt(60);
Matrix matrix = new Matrix();
public FishView(Context ctx, AttributeSet set)
{
    super(ctx, set);
    // 获取该 SurfaceView 对应的 SurfaceHolder，并将该类的实例作为其 Callback
    holder = getHolder();
    holder.addCallback(this);
    hasSurface = false;
    back = BitmapFactory.decodeResource(ctx.getResources()
        , R.drawable.fishbg);
    fishs = new Bitmap[10];
    // 初始化鱼游动动画的 10 张图片
    for(int i = 0 ; i < 10 ; i++)
    {
        try
        {
            int fishId = (Integer)R.drawable.class
                .getField("fish" + i).get(null);
            fishs[i] = BitmapFactory.decodeResource(
                ctx.getResources(), fishId);
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
public void resume()
{
    // 创建和启动图像更新线程
    if (updateThread == null)
    {
        updateThread = new UpdateViewThread();
        if (hasSurface == true)
            updateThread.start();
    }
}
public void pause()
{
    // 停止图像更新线程
    if (updateThread != null)
    {
        updateThread.requestExitAndWait();
        updateThread = null;
    }
}
// 当 SurfaceView 被创建时回调该方法
@Override
public void surfaceCreated(SurfaceHolder holder)
{
    hasSurface = true;
    resume();
}
// 当 SurfaceView 将要被销毁时回调该方法
public void surfaceDestroyed(SurfaceHolder holder)
{
    hasSurface = false;
    pause();
}
```

```
// 当 SurfaceView 发生改变时回调该方法
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
    if (updateThread != null)
        updateThread.onWindowResize(w, h);
}
class UpdateViewThread extends Thread
{
    // 定义一个记录图像是否更新完成的旗标
    private boolean done;
    UpdateViewThread()
    {
        super();
        done = false;
    }
    @Override
    public void run()
    {
        SurfaceHolder surfaceHolder = holder;
        // 重复绘图循环，直到线程停止
        while (!done)
        {
            // 锁定 SurfaceView，并返回到要绘图的 Canvas
            Canvas canvas = surfaceHolder.lockCanvas(); // ①
            // 绘制背景图片
            canvas.drawBitmap(back, 0, 0, null);
            // 如果鱼“游出”屏幕之外，重新初始化鱼的位置
            if(fishX < 0)
            {
                fishX = 778;
                fishY = 500;
                fishAngle = new Random().nextInt(60);
            }
            if(fishY < 0)
            {
                fishX = 778;
                fishY = 500;
                fishAngle = new Random().nextInt(60);
            }
            // 使用 Matrix 来控制鱼的旋转角度和位置
            matrix.reset();
            matrix.setRotate(fishAngle);
            matrix.postTranslate(fishX -= fishSpeed * Math
                .cos(Math.toRadians(fishAngle))
                , fishY -= fishSpeed * Math.sin(Math.toRadians(fishAngle)));
            canvas.drawBitmap(fishs[fishIndex++ % fishs.length], matrix, null);
            // 解锁 Canvas，并渲染当前图像
            surfaceHolder.unlockCanvasAndPost(canvas); // ②
            try
            {
                Thread.sleep(60);
            }
            catch (InterruptedException e){}
        }
    }
    public void requestExitAndWait()
    {
        // 把这个线程标记为完成，并合并到主程序线程中
        done = true;
        try
        {
            join();
        }
```

```
            catch (InterruptedException ex){}
        }
        public void onWindowResize(int w, int h){
            // 处理 SurfaceView 的大小改变事件
        }
    }
}
```

上面的程序使用 FishView 自身作为 SurfaceHolder 的 Callback 实例，并实现了 Callback 中定义的如下三个方法。

- void surfaceChanged(SurfaceHolder holder, int format, int width, int height)：当一个 SurfaceView 的格式或大小发生改变时回调该方法。
- void surfaceCreated(SurfaceHolder holder)：当 SurfaceView 被创建时回调该方法。
- void surfaceDestroyed(SurfaceHolder holder)：当 SurfaceView 将要被销毁时回调该方法。

上面 FishView 所实现的 surfaceCreated()方法中调用了该类的 resume()方法，该方法将会启动一个负责更新游戏图像的线程，这样即可保证 FishView 上的动画不停地改变。当 SurfaceView 被销毁时，程序会回调它的 surfaceDestroyed()方法，FishView 所实现的该方法则负责停止更新游戏图像的线程，这样即可释放线程资源。

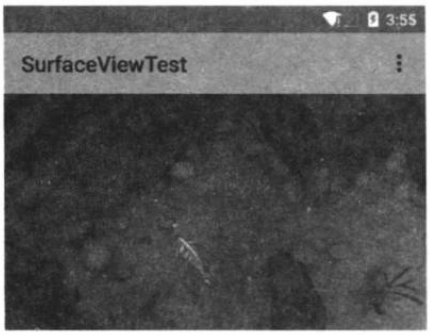


图 7.20　SurfaceView 的绘图机制

FishView 中定义了一个 UpdateViewThread 线程，这个线程可通过 SurfaceHolder 来绘制 SurfaceView 上绘制的图像（这也是 SurfaceView 与普通 View 的显著区别之一）。程序中①号粗体字代码用于锁定 SurfaceView，准备开始绘制，接下来程序即可在 SurfaceView 上执行任意绘制。程序对 SurfaceView 绘制完成之后，程序中②号粗体字代码解锁画布，并将绘制内容显示出来。

该程序的界面布局文件只是简单地显示该 FishView，主程序也很简单，此处不再给出。运行该程序，即可看到如图 7.20 所示的“小鱼游动”的界面。

实例：基于 SurfaceView 开发示波器

SurfaceView 与普通 View 还有一个重要的区别：View 的绘图必须在当前 UI 线程中进行——这也是前面程序需要更新 View 组件时总要采用 Handler 处理的原因；但 SurfaceView 就不会存在这个问题，因此 SurfaceView 的绘图是由 SurfaceHolder 来完成的。

对于 View 组件，如果程序需要花较长的时间来更新绘图，那么主 UI 线程将会被阻塞，无法响应用户的任何动作；而 SurfaceHolder 则会启用新的线程去更新 SurfaceView 的绘制，因此不会阻塞主 UI 线程。

一般来说，如果程序或游戏界面的动画元素较多，而且很多动画元素的移动都需要通过定时器来控制，就可以考虑使用 SurfaceView，而不是 View。例如，下面我们使用 SurfaceView 开发一个示波器程序，该程序将会根据用户单击的按钮在屏幕上自动绘制正弦波或余弦波。

该程序的界面布局很简单，包含两个按钮和一个 SurfaceView，程序每次绘制时只需要绘制（更新）当前点的波形，前面已经绘制的波形无须更新，这就利用了 SurfaceHolder 的 lockCanvas(Rect r)方法。

该程序的代码如下。

程序清单：codes\07\7.7\ShowWave\app\src\main\java\org\crazyit\image\MainActivity.java

```
public class MainActivity extends Activity
{
    private SurfaceHolder holder;
    private Paint paint;
    final int HEIGHT = 320;
    final int WIDTH = 768;
    final int X_OFFSET = 5;
    private int cx = X_OFFSET;
    // 实际的 Y 轴的位置
    int centerY = HEIGHT / 2;
    Timer timer = new Timer();
    TimerTask task = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final SurfaceView surface = (SurfaceView)
            findViewById(R.id.show);
        // 初始化 SurfaceHolder 对象
        holder = surface.getHolder();
        paint = new Paint();
        paint.setColor(Color.GREEN);
        paint.setStrokeWidth(3);
        Button sin = (Button)findViewById(R.id.sin);
        Button cos = (Button)findViewById(R.id.cos);
        OnClickListener listener = (new OnClickListener()
        {
            @Override
            public void onClick(final View source)
            {
                drawBack(holder);
                cx = X_OFFSET;
                if(task != null)
                {
                    task.cancel();
                }
                task = new TimerTask()
                {
                    public void run()
                    {
                        int cy = source.getId() == R.id.sin ? centerY
                            - (int)(100 * Math.sin((cx - 5) * 2
                            * Math.PI / 150))
                            : centerY - (int)(100 * Math.cos ((cx - 5)
                             * 2 * Math.PI / 150));
                        Canvas canvas = holder.lockCanvas(new Rect(cx ,
                            cy - 2  , cx + 2, cy + 2));
                        canvas.drawPoint(cx , cy , paint);
                        cx ++;
                        if (cx > WIDTH)
                        {
                            task.cancel();
                            task = null;
                        }
                        holder.unlockCanvasAndPost(canvas);
                    }
                };
                timer.schedule(task , 0 , 30);
            }
```

```
        });
        sin.setOnClickListener(listener);
        cos.setOnClickListener(listener);
        holder.addCallback(new Callback()
        {
            @Override
            public void surfaceChanged(SurfaceHolder holder, int format,
                int width, int height)
            {
                drawBack(holder);
            }
            @Override
            public void surfaceCreated(final SurfaceHolder myHolder){ }
            @Override
            public void surfaceDestroyed(SurfaceHolder holder)
            {
                timer.cancel();
            }
        });
    }
    private void drawBack(SurfaceHolder holder)
    {
        Canvas canvas = holder.lockCanvas();
        // 绘制白色背景
        canvas.drawColor(Color.WHITE);
        Paint p = new Paint();
        p.setColor(Color.BLACK);
        p.setStrokeWidth(2);
        // 绘制坐标轴
        canvas.drawLine(X_OFFSET , centerY , WIDTH , centerY , p);
        canvas.drawLine(X_OFFSET , 40 , X_OFFSET , HEIGHT , p);
        holder.unlockCanvasAndPost(canvas);
        holder.lockCanvas(new Rect(0 , 0 , 0 , 0));
        holder.unlockCanvasAndPost(canvas);
    }
}
```

从上面程序中的三行粗体字代码可以看出，当程序每次绘制正弦波、余弦波上的当前点时，程序都无须重绘整个画面，SurfaceHolder 只要锁定当前绘制点的小范围即可，系统更新画面时也只要更新这个范围即可，因此具有较好的画面性能。运行该程序，将看到如图 7.21 所示的结果。

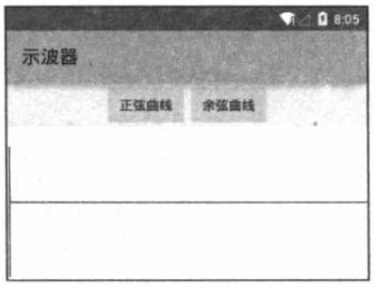


图 7.21 示波器

7.8 本章小结

本章主要介绍了 Android 的图形、图像处理，这种图形、图像处理不仅对 Android 界面开发十分重要，而且也是开发 Android 2D 游戏的基础。学习本章需要重点掌握 Android 丰富的绘图 API，包括 Canvas、Paint、Path 等类。除此之外，读者还需要掌握 Android 绘图的双缓冲机制，以及利用 Matrix 对图形进行几何变换等内容。除此之外，Android 提供了逐帧动画、补间动画、属性动画支持，尤其需要重点掌握属性动画。为了更好地开发游戏动画界面，Android 专门提供了 SurfaceView，本章详细介绍了 SurfaceView 的绘图机制，并讲解了如何继承 SurfaceView 来开发动画。

CHAPTER

8

第8章 Android 数据存储与 IO

本章要点

- SharedPreferences 的概念和作用
- 使用 SharedPreferences 保存程序的参数、选项
- 读写其他应用的 SharedPreferences
- Android 的文件 IO
- 读写 SD 卡上的文件
- 了解 SQLite 数据库
- 使用 Android 的 API 操作 SQLite 数据库
- 使用 sqlite3 工具管理 SQLite 数据库
- SQLiteOpenHelper 类的功能和用法
- Android 的手势支持
- 手势检测
- 向手势库中添加手势
- 识别用户手势

所有应用程序都必然涉及数据的输入、输出，Android 应用也不例外，应用程序的参数设置、程序运行状态数据这些都需要保存到外部存储器上，这样系统关机之后数据才不会丢失。Android 应用是使用 Java 语言来开发的，因此开发者在 Java IO 中的编程经验大部分都可“移植”到 Android 应用开发上。Android 系统还提供了一些专门的 IO API，通过这些 API 可以更有效地进行输入、输出。

如果应用程序只有少量数据需要保存，那么使用普通文件就可以了；但如果应用程序有大量数据需要存储、访问，就需要借助于数据库了。Android 系统内置了 SQLite 数据库，SQLite 数据库是一个真正轻量级的数据库，它没有后台进程，整个数据库就对应于一个文件，这样可以非常方便地在不同设备之间移植。Android 不仅内置了 SQLite 数据库，而且为访问 SQLite 数据库提供了大量便捷的 API。本章将会详细介绍如何在 Android 应用中使用 SQLite 数据库。

8.1 使用 SharedPreferences

有些时候，应用程序有少量的数据需要保存，而且这些数据的格式很简单，都是普通的字符串、标量类型的值等，比如应用程序的各种配置信息（如是否打开音效、是否使用振动效果等）、小游戏的玩家积分（如扫雷英雄榜之类的）等，对于这种数据，Android 提供了 SharedPreferences 进行保存。

8.1.1 SharedPreferences 与 Editor 简介

SharedPreferences 保存的数据主要是类似于配置信息格式的数据，因此它保存的数据主要是简单类型的 key-value 对。

SharedPreferences 接口主要负责读取应用程序的 Preferences 数据，它提供了如下常用方法来访问 SharedPreferences 中的 key-value 对。

- boolean contains(String key)：判断 SharedPreferences 是否包含特定 key 的数据。
- abstract Map<String, ?> getAll()：获取 SharedPreferences 数据里全部的 key-value 对。
- boolean getXxx(String key, xxx defValue)：获取 SharedPreferences 数据里指定 key 对应的 value。如果该 key 不存在，则返回默认值 defValue。其中 xxx 可以是 boolean、float、int、long、String 等各种基本类型的值。

SharedPreferences 接口本身并没有提供写入数据的能力，而是通过 SharedPreferences 的内部接口，SharedPreferences 调用 edit()方法即可获取它所对应的 Editor 对象。Editor 提供了如下方法来向 SharedPreferences 写入数据。

- SharedPreferences.Editor clear()：清空 SharedPreferences 里所有数据。
- SharedPreferences.Editor putXxx(String key, xxx value)：向 SharedPreferences 存入指定 key 对应的数据。其中 xxx 可以是 boolean、float、int、long、String 等各种基本类型的值。
- SharedPreferences.Editor remove(String key)：删除 SharedPreferences 里指定 key 对应的数据项。
- boolean commit()：当 Editor 编辑完成后，调用该方法提交修改。

提示：

从用法角度来看，SharedPreferences 和 SharedPreferences.Editor 组合起来非常像 Map，其中 SharedPreferences 负责根据 key 读取数据，而 SharedPreferences.Editor 则用于写入数据。

SharedPreferences 本身是一个接口，程序无法直接创建 SharedPreferences 实例，只能通过 Context 提供的 getSharedPreferences(String name , int mode)方法来获取 SharedPreferences 实例，该方法的第二个参数支持如下几个值。

- Context.MODE_PRIVATE：指定该 SharedPreferences 数据只能被本应用程序读写。
- Context.MODE_WORLD_READABLE：指定该 SharedPreferences 数据能被其他应用程序读，但不能写。
- Context.MODE_WORLD_WRITEABLE：指定该 SharedPreferences 数据能被其他应用程序读写。

提示：

从 Android 4.2 开始，Android 不再推荐使用 MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE 这两种模式，因为这两种模式允许其他应用程序来读或写本应用创建的数据，因此容易导致安全漏洞。如果应用程序确实需要把内部数据暴露出来供其他应用访问，则应该使用本书后面介绍的 ContentProvider。

下面介绍对 SharedPreferences 的简单读写。

8.1.2 SharedPreferences 的存储位置和格式

下面的程序示范了如何向 SharedPreferences 中写入、读取数据。该程序的界面很简单，它只是提供了两个按钮，其中一个用于写入数据；另一个用于读取数据，故此处不再给出界面布局文件。程序代码如下。

程序清单：codes\08\8.1\SharedPreferencesTest\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    SharedPreferences preferences;
    SharedPreferences.Editor editor;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取只能被本应用程序读写的 SharedPreferences 对象
        preferences = getSharedPreferences("crazyit", MODE_PRIVATE);
        editor = preferences.edit();
        Button read = (Button) findViewById(R.id.read);
        Button write = (Button) findViewById(R.id.write);
        read.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 读取字符串数据
                String time = preferences.getString("time", null);
                // 读取 int 类型的数据
                int randNum = preferences.getInt("random", 0);
                String result = time == null ? "您暂时还未写入数据" : "写入时间为："
                    + time + "\n 上次生成的随机数为：" + randNum;
                // 使用 Toast 提示信息
                Toast.makeText(MainActivity.this, result
                    , Toast.LENGTH_SHORT).show();
            }
        });
```

```
        write.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 "
                    + "hh:mm:ss");
                // 存入当前时间
                editor.putString("time", sdf.format(new Date()));
                // 存入一个随机数
                editor.putInt("random", (int) (Math.random() * 100));
                // 提交所有存入的数据
                editor.commit();
            }
        });
    }
}
```

上面程序中的第一段粗体字代码用于读取 SharedPreferences 数据，当程序所读取的SharedPreferences数据文件根本不存在时，程序也返回默认值，并不会抛出异常；第二段粗体字代码用于写入 SharedPreferences 数据，由于 SharedPreferences 并不支持写入 Date 类型的值，故程序使用了 SimpleDateFormat 将 Date 格式化成字符串后写入。

运行上面的程序，单击程序中“写入数据”按钮，程序将完成 SharedPreferences 写入，写入完成后打开 DDMS 的 File Explorer 面板（参考第 1 章介绍的方法来打开 DDMS 的 File Explorer 面板），然后展开文件浏览树，将看到如图 8.1 所示的窗口。

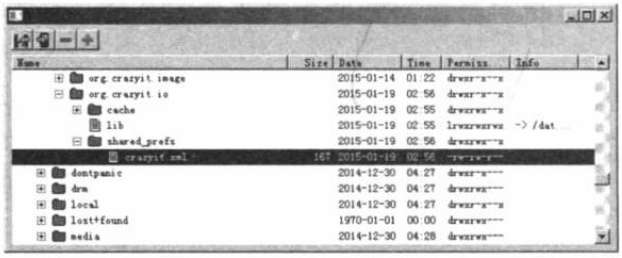


图 8.1　SharedPreferences 的存储路径

从图 8.1 可以看出，SharedPreferences 数据总是保存在/data/data/<package name>/shared_prefs 目录下，SharedPreferences 数据总是以 XML 格式保存。通过 File Explorer 面板的导出文件按钮导出该 XML 文档，打开该 XML 文档可以看到如下文件内容：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="time">2015年01月19日 02:56:01</string>
    <int name="random" value="21" />
</map>
```

从上面的文件不难看出，SharedPreferences数据文件的根元素是<map.../>元素，该元素里每个子元素代表一个 key-value 对，当 value 是整数类型时，使用<int.../>子元素；当 value 是字符串类型时，使用<string .../>子元素……依此类推。

单击程序界面中的“读取数据”按钮，程序弹出一个 Toast 对话框显示上次写入的数据，如图 8.2 所示。

图 8.2　读取 SharedPreferences 数据

实例：记录应用程序的使用次数

这个简单的实例可以记录应用程序的使用次数：当用户第一次启动该应用程序时，系统创建 SharedPreferences 来记录使用次数。用户以后启动应用程序时，系统先读取 SharedPreferences 中记录的使用次数，然后将使用次数加 1。

本实例程序的代码如下。

程序清单：codes\08\8.1\UseCount\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    SharedPreferences preferences;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        preferences = getSharedPreferences("count"
                , MODE_PRIVATE);
        // 读取 SharedPreferences 里的 count 数据
        int count = preferences.getInt("count", 0);
        // 显示程序以前使用的次数
        Toast.makeText(this, "程序以前被使用了" + count + "次。"
                , Toast.LENGTH_LONG).show();
        SharedPreferences.Editor editor = preferences.edit();
        // 存入数据
        editor.putInt("count", ++count);
        // 提交修改
        editor.commit();
    }
}
```

上面程序中的第一行粗体字代码用于读取 SharedPreferences 中记录的使用次数；第二行粗体字代码将使用次数增加 1，并再次将使用次数写入 SharedPreferences 中。

8.2 File 存储

读者学习 Java SE 的时候都知道 Java 提供了一套完整的 IO 流体系，包括 FileInputStream、FileOutputStream 等，通过这些 IO 流可以非常方便地访问磁盘上的文件内容。Android 同样支持以这种方式来访问手机存储器上的文件。

8.2.1 openFileOutput 和 openFileInput

Context 提供了如下两个方法来打开应用程序的数据文件夹里的文件 IO 流。

- FileInputStream openFileInput(String name)：打开应用程序的数据文件夹下的 name 文件对应的输入流。
- FileOutputStream openFileOutput(String name, int mode)：打开应用程序的数据文件夹下的 name 文件对应的输出流。

上面两个方法分别用于打开文件输入流、输出流，其中第二个方法的第二个参数指定打开文件的模式，该模式支持如下值。

- MODE_PRIVATE：该文件只能被当前程序读写。
- MODE_APPEND：以追加方式打开该文件，应用程序可以向该文件中追加内容。
- MODE_WORLD_READABLE：该文件的内容可以被其他程序读取。

- MODE_WORLD_WRITEABLE：该文件的内容可由其他程序读写。

提示：

与前面介绍的类似，从Android 4.2开始，Android不推荐使用MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE两种模式。

除此之外，Context还提供了如下几个方法来访问应用程序的数据文件夹。

- getDir(String name, int mode)：在应用程序的数据文件夹下获取或创建name对应的子目录。
- File getFilesDir()：获取应用程序的数据文件夹的绝对路径。
- String[] fileList()：返回应用程序的数据文件夹下的全部文件。
- deleteFile(String)：删除应用程序的数据文件夹下的指定文件。

下面的程序简单示范了如何读写应用程序数据文件夹内的文件。该程序的界面布局同样很简单，只包含两个文本框和两个按钮，其中第一组文本框和按钮用于处理写入，文本框用于接受用户输入，当用户单击“写入”按钮时，程序将会把数据写入文件；第二组文本框和按钮用于处理读取，当用户单击“读取”按钮时，该文本框显示文件中的数据。程序代码如下。

注意：

由于本书主要介绍Android应用开发的相关知识，因此Java IO支持的相关内容则不在本书介绍的范围之内，如果读者对Java IO还不熟悉，建议先阅读疯狂Java体系的《疯狂Java讲义》。

程序清单：codes\08\8.2\FileTest\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    final String FILE_NAME = "crazyit.bin";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        System.out.println(new StringBuilder("a").append("b").append("c")
            .toString());
        // 获取两个按钮
        Button read = (Button) findViewById(R.id.read);
        Button write = (Button) findViewById(R.id.write);
        // 获取两个文本框
        final EditText edit1 = (EditText) findViewById(R.id.edit1);
        final EditText edit2 = (EditText) findViewById(R.id.edit2);
        // 为write按钮绑定事件监听器
        write.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 将edit1中的内容写入文件中
                write(edit1.getText().toString());
                edit1.setText("");
            }
        });
        read.setOnClickListener(new OnClickListener()
        {
            @Override
```

```
        public void onClick(View v)
        {
            // 读取指定文件中的内容，并显示出来
            edit2.setText(read());
        }
    });
}
private String read()
{
    try
    {
        // 打开文件输入流
        FileInputStream fis = openFileInput(FILE_NAME);
        byte[] buff = new byte[1024];
        int hasRead = 0;
        StringBuilder sb = new StringBuilder("");
        // 读取文件内容
        while ((hasRead = fis.read(buff)) > 0)
        {
            sb.append(new String(buff, 0, hasRead));
        }
        // 关闭文件输入流
        fis.close();
        return sb.toString();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    return null;
}
private void write(String content)
{
    try
    {
        // 以追加方式打开文件输出流
        FileOutputStream fos = openFileOutput(FILE_NAME, MODE_APPEND);
        // 将 FileOutputStream 包装成 PrintStream
        PrintStream ps = new PrintStream(fos);
        // 输出文件内容
        ps.println(content);
        // 关闭文件输出流
        ps.close();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
```

上面程序中的第一段粗体字代码用于读取应用程序的数据文件，第二段粗体字代码用于向应用程序的数据文件中追加内容。从上面的粗体字代码可以看出，当 Android 系统调用 Context 的 openFileInput()、openFileOutput()打开文件输入流或输出流之后，接下来 IO 流的用法与 Java SE 中 IO 流的用法完全一样：直接用节点流读写也行，用包装流包装之后再处理也没问题。

当单击程序界面中的“写入”按钮时，用户在第一个文本框中输入的内容将会被保存到应用程序的数据文件中，打开 File Explorer 面板，可以看到如图 8.3 所示的界面。

从图 8.3 可以看出，应用程序的数据文件默认保存在/data/data/<package name>/files 目录下。

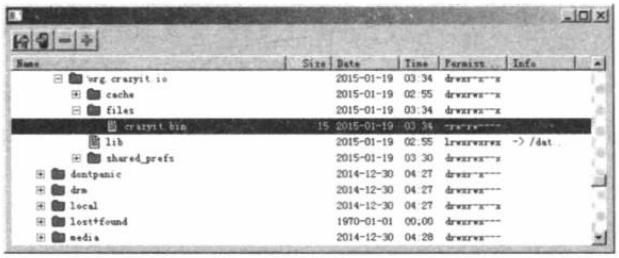


图 8.3　应用程序的数据文件

8.2.2　读写 SD 卡上的文件

当程序通过 Context 的 openFileInput()或 openFileOutput()来打开文件输入流、输出流时，程序所打开的都是应用程序的数据文件夹里的文件，这样所存储的文件大小可能比较有限——毕竟手机内置的存储空间是有限的。

为了更好地存取应用程序的大文件数据，应用程序需要读写 SD 卡上的文件。SD 卡大大扩充了手机的存储能力。

读写 SD 上的文件请按如下步骤进行。

① 调用 Environment 的 getExternalStorageState()方法判断手机上是否插入了 SD 卡，并且应用程序具有读写 SD 卡的权限。例如使用如下代码：

```
// 如果手机已插入 SD 卡，且应用程序具有读写 SD 卡的能力，下面语句返回 true
Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)
```

② 调用 Environment 的 getExternalStorageDirectory()方法来获取外部存储器，也就是 SD 卡的目录。

③ 使用 FileInputStream、FileOutputStream、FileReader 或 FileWriter 读写 SD 卡里的文件。

应用程序读写 SD 卡上的文件有如下两个注意点。

- 手机上应该已插入 SD 卡。对于模拟器来说，可通过 mksdcard 命令来创建虚拟存储卡。关于虚拟存储卡的管理请参考第 1 章。
- 为了读写 SD 卡上的数据，必须在应用程序的清单文件（AndroidManifest.xml）中添加读写 SD 卡的权限。例如如下配置：

```
<!-- 在 SD 卡中创建与删除文件权限 -->
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 向 SD 卡写入数据权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

下面的程序示范了如何读写 SD 卡上的文件。该程序的主界面与上一个程序的界面完全相同，只是该程序数据读写是基于 SD 卡的。该程序代码如下。

程序清单：codes\08\8.2\SDCardTest\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    final String FILE_NAME = "/crazyit.bin";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取两个按钮
        Button read = (Button) findViewById(R.id.read);
        Button write = (Button) findViewById(R.id.write);
        // 获取两个文本框
        final EditText edit1 = (EditText) findViewById(R.id.edit1);
```

```
        final EditText edit2 = (EditText) findViewById(R.id.edit2);
        // 为write按钮绑定事件监听器
        write.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 将edit1中的内容写入文件中
                write(edit1.getText().toString());
                edit1.setText("");
            }
        });
        read.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 读取指定文件中的内容，并显示出来
                edit2.setText(read());
            }
        });
    }
    private String read()
    {
        try
        {
            // 如果手机插入了SD卡，而且应用程序具有访问SD卡的权限
            if (Environment.getExternalStorageState().equals(
                    Environment.MEDIA_MOUNTED))
            {
                // 获取SD卡对应的存储目录
                File sdCardDir = Environment.getExternalStorageDirectory();
                // 获取指定文件对应的输入流
                FileInputStream fis = new FileInputStream(
                    sdCardDir.getCanonicalPath() + FILE_NAME);
                // 将指定输入流包装成BufferedReader
                BufferedReader br = new BufferedReader(new
                    InputStreamReader(fis));
                StringBuilder sb = new StringBuilder("");
                String line = null;
                // 循环读取文件内容
                while ((line = br.readLine()) != null)
                {
                    sb.append(line);
                }
                // 关闭资源
                br.close();
                return sb.toString();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return null;
    }
    private void write(String content)
    {
        try
        {
            // 如果手机插入了SD卡，而且应用程序具有访问SD卡的权限
            if (Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED))
```

```
            {
                // 获取 SD 卡的目录
                File sdCardDir = Environment.getExternalStorageDirectory();
                File targetFile = new File(sdCardDir
                    .getCanonicalPath() + FILE_NAME);
                // 以指定文件创建 RandomAccessFile 对象
                RandomAccessFile raf = new RandomAccessFile(
                    targetFile, "rw");
                // 将文件记录指针移动到最后
                raf.seek(targetFile.length());
                // 输出文件内容
                raf.write(content.getBytes());
                // 关闭 RandomAccessFile
                raf.close();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

上面程序中的第一段粗体字代码用于读取 SD 卡中指定文件的内容；第二段粗体字代码则使用 RandomAccessFile 向 SD 卡中的指定文件追加内容——如果使用 FileOutputStream 向指定文件写入数据，FileOutputStream 会把原有的文件内容清空，那就不是追加文件内容了。由此可见，当程序直接使用 FileOutputStream 进行输出时，比使用 Context 的 openFileOutput()方便。

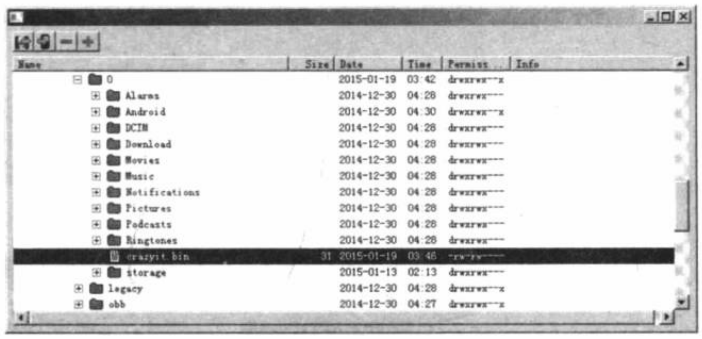


图 8.4　存入 SD 卡的数据

运行上面的程序，在第一个文本框内输入一些字符串，然后单击“写入”按钮，即可将数据写入底层 SD 卡中。打开 File Explorer，即可看到如图 8.4 所示的界面。

提示：

对于 Genymotion 模拟器而言，它模拟的 SD 卡对应的路径为/mnt/shell/emulated/0。

需要指出的是，当开发者直接在 Eclipse 中运行 Android 应用程序时，Eclipse 默认启动的模拟器是不带 SD 卡的。为了让 Eclipse 启动的模拟器带上 SD 卡，可以通过“Run As”→“Run Configurations”菜单项打开如图 8.5 所示的对话框，通过该对话框下面的附加选项来使用 SD 卡（实际上就是在启动模拟器时增加-sdcard 选项）。

提示：

如果开发者不想使用 Environment.getExternalStorageDirectory()这么复杂的语句来获取 SD 卡的路径，则完全可以使用/mnt/sdcard/路径代表 SD 卡的路径，然后程序通过判断/mnt/sdcard/路径是否存在就可知道手机是否已插入了 SD 卡。

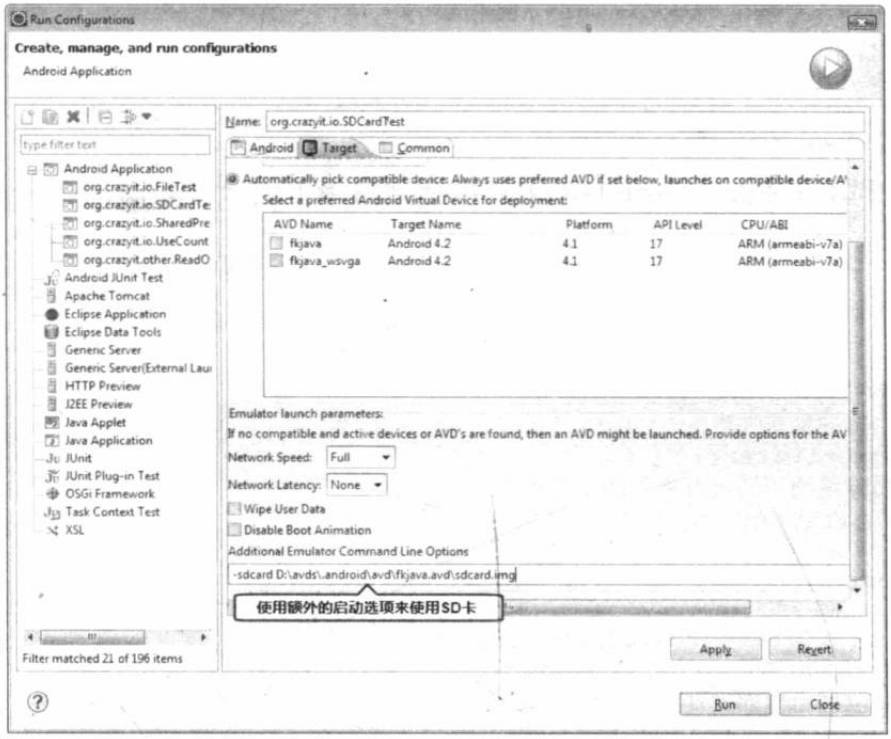


图 8.5 指定启动模拟器时使用 SD 卡

实例：SD 卡文件浏览器

下面我们将会利用 Java 的 File 类开发一个 SD 卡文件浏览器，该程序直接使用/mnt/sdcard 来访问系统的 SD 卡目录，然后通过 File 的 listFiles()方法来获取指定目录下的全部文件和文件夹。

当程序启动时，系统启动获取/mnt/sdcard/目录下的全部文件、文件夹，并使用 ListView 将它们显示出来；当用户单击 ListView 的指定列表项时，系统将会显示该列表项下的全部文件夹和文件。

该程序的界面布局文件如下。

程序清单：codes\08\8.2\SDFileExplorer\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 显示当前路径的文本框 -->
<TextView
    android:id="@+id/path"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_alignParentTop="true"/>
<!-- 列出当前路径下所有文件的 ListView -->
<ListView
    android:id="@+id/list"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:divider="#000"
    android:dividerHeight="1px"
    android:layout_below="@id/path"/>
<!-- 返回上一级目录的按钮 -->
<Button android:id="@+id/parent"
    android:layout_width="38dp"
```

```
    android:layout_height="34dp"
    android:background="@drawable/home"
    android:layout_centerHorizontal="true"
    android:layout_alignParentBottom="true"/>
</RelativeLayout>
```

该程序主要利用了 File 的 listFiles()来列出指定目录下的全部文件，程序代码如下。

程序清单：codes\08\8.2\SDFileExplorer\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    ListView listView;
    TextView textView;
    // 记录当前的父文件夹
    File currentParent;
    // 记录当前路径下的所有文件的文件数组
    File[] currentFiles;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取列出全部文件的 ListView
        listView = (ListView) findViewById(R.id.list);
        textView = (TextView) findViewById(R.id.path);
        // 获取系统的 SD 卡的目录
        File root = new File("/mnt/sdcard/");
        // 如果 SD 卡存在
        if (root.exists())
        {
            currentParent = root;
            currentFiles = root.listFiles();
            // 使用当前目录下的全部文件、文件夹来填充 ListView
            inflateListView(currentFiles);
        }
        // 为 ListView 的列表项的单击事件绑定监听器
        listView.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                // 用户单击了文件，直接返回，不做任何处理
                if (currentFiles[position].isFile()) return;
                // 获取用户单击的文件夹下的所有文件
                File[] tmp = currentFiles[position].listFiles();
                if (tmp == null || tmp.length == 0)
                {
                    Toast.makeText(MainActivity.this
                        , "当前路径不可访问或该路径下没有文件",
                        Toast.LENGTH_SHORT).show();
                }
                else
                {
                    // 获取用户单击的列表项对应的文件夹，设为当前的父文件夹
                    currentParent = currentFiles[position]; // ②
                    // 保存当前父文件夹内的全部文件和文件夹
                    currentFiles = tmp;
                    // 再次更新 ListView
                    inflateListView(currentFiles);
                }
            }
```

```
        });
        // 获取上一级目录的按钮
        Button parent = (Button) findViewById(R.id.parent);
        parent.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                try
                {
                    if (!currentParent.getCanonicalPath()
                        .equals("/mnt/sdcard"))
                    {
                        // 获取上一级目录
                        currentParent = currentParent.getParentFile();
                        // 列出当前目录下的所有文件
                        currentFiles = currentParent.listFiles();
                        // 再次更新 ListView
                        inflateListView(currentFiles);
                    }
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
    private void inflateListView(File[] files)  // ①
    {
        // 创建一个 List 集合，List 集合的元素是 Map
        List<Map<String, Object>> listItems =
            new ArrayList<Map<String, Object>>();
        for (int i = 0; i < files.length; i++)
        {
            Map<String, Object> listItem =
                new HashMap<String, Object>();
            // 如果当前 File 是文件夹，使用 folder 图标；否则使用 file 图标
            if (files[i].isDirectory())
            {
                listItem.put("icon", R.drawable.folder);
            }
            else
            {
                listItem.put("icon", R.drawable.file);
            }
            listItem.put("fileName", files[i].getName());
            // 添加 List 项
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this
            , listItems, R.layout.line
            , new String[]{ "icon", "fileName" }
            , new int[]{R.id.icon, R.id.file_name });
        // 为 ListView 设置 Adapter
        listView.setAdapter(simpleAdapter);
        try
        {
            textView.setText("当前路径为: "
                + currentParent.getCanonicalPath());
        }
        catch (IOException e)
```

```
            {
                e.printStackTrace();
            }
        }
    }
```

上面的程序中①号粗体字方法使用 File[]数组来填充 ListView，填充时程序会根据 File[]数组里的数据元素（File 对象）代表的是文件还是文件夹来选择使用文件图标或文件夹图标，如该方法中粗体字代码所示。

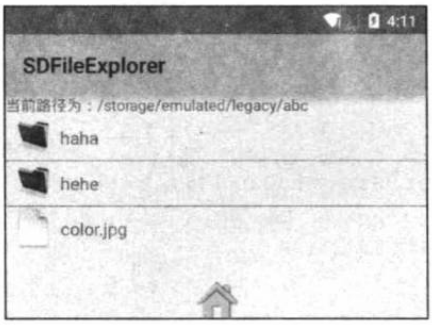


图 8.6　SD 卡文件浏览器

当用户单击 ListView 中某个列表项时，程序会将用户单击的列表项所对应的文件夹当成 currentParent 处理，并再次调用 inflateListView(File[] files)方法来列出当前文件夹下的所有文件。

运行上面的程序，将看到如图 8.6 所示的界面。

正如图 8.6 所示，该程序就像 SD 卡资源管理器一样可以非常方便地浏览 SD 卡里包含的全部文件。

提示：

从图 8.6 所示界面中可以看到/mnt/sdcard/abc 目录下包含了 haha、hehe 两个目录和 color.jpg 文件。这就要求开发者的模拟器的 SD 卡里带有这些目录和文件，为了在 SD 卡里创建目录，可先运行 adb shell 命令来启动 Android 的 Shell 窗口——由于 Android 的内核就是 Linux，因此可在该 Shell 窗口里执行 ls、mkdir 等常见的 Linux 命令。

8.3　SQLite 数据库

Android 系统集成了一个轻量级的数据库：SQLite，SQLite 并不想成为像 Oracle、MySQL 那样的数据库。SQLite 只是一个嵌入式的数据库引擎，专门适用于资源有限的设备（如手机、PDA 等）上适量数据存取。

虽然 SQLite 支持绝大部分 SQL 92 语法，也允许开发者使用 SQL 语句操作数据库中的数据，但 SQLite 并不像 Oracle、MySQL 数据库那样需要安装、启动服务器进程，SQLite 数据库只是一个文件。

提示：

从本质上来看，SQLite 的操作方式只是一种更为便捷的文件操作。后面我们会看到，当应用程序创建或打开一个 SQLite 数据库时，其实只是打开一个文件准备读写，因此有人说 SQLite 有点像 Microsoft 的 Access（实际上 SQLite 功能要强大得多）。可能有读者会问，如果实际项目中有大量数据需要读写，而且需要面临大量用户的并发存储怎么办呢？对于这种情况，本身就不应该把数据存放在手机的 SQLite 数据库里——毕竟手机还是手机，它的存储能力、计算能力都不足以让它充当服务器的角色。

8.3.1　SQLiteDatabase 简介

Android 提供了 SQLiteDatabase 代表一个数据库（底层就是一个数据库文件），一旦应用程序获得了代表指定数据库的 SQLiteDatabase 对象，接下来就可通过 SQLiteDatabase 对象来

管理、操作数据库了。

SQLiteDatabase 提供了如下静态方法来打开一个文件对应的数据库。

- static SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)：打开 path 文件所代表的 SQLite 数据库。
- static SQLiteDatabase openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)：打开或创建（如果不存在）file 文件所代表的 SQLite 数据库。
- static SQLiteDatabase openOrCreateDatabase(String path,SQLiteDatabase. CursorFactory factory)：打开或创建（如果不存在）path 文件所代表的 SQLite 数据库。

在程序中获取 SQLiteDatabase 对象之后，接下来就可调用 SQLiteDatabase 的如下方法来操作数据库了。

- execSQL(String sql, Object[] bindArgs)：执行带占位符的 SQL 语句。
- execSQL(String sql)：执行 SQL 语句。
- insert(String table, String nullColumnHack, ContentValues values)：向指定表中插入数据。
- update(String table, ContentValues values, String whereClause, String[] whereArgs)：更新指定表中的特定数据。
- delete(String table, String whereClause, String[] whereArgs)：删除指定表中的特定数据。
- Cursor query(String table, String[] columns, String whereClause, String[] whereArgs, String groupBy, String having, String orderBy)：对指定数据表执行查询。
- Cursor query(String table, String[] columns, String whereClause, String[] whereArgs, String groupBy, String having, String orderBy, String limit)：对指定数据表执行查询。limit 参数控制最多查询几条记录（用于控制分页的参数）。
- Cursor query(boolean distinct, String table, String[] columns, String whereClause, String[]whereArgs, String groupBy, String having, String orderBy, String limit)：对指定数据表执行查询。其中第一个参数控制是否去除重复值。
- rawQuery(String sql, String[] selectionArgs)：执行带占位符的 SQL 查询。
- beginTransaction()：开始事务。
- endTransaction()：结束事务。

从上面的方法不难看出，其实 SQLiteDatabase 的作用有点类似于 JDBC 的 Connection 接口，但 SQLiteDatabase 提供的方法更多，比如 insert、update、delete、query 等方法，其实这些方法完全可通过执行 SQL 语句来完成，但 Android 考虑到部分开发者对 SQL 语法不熟悉，所以提供这些方法帮助开发者以更简单的方式来操作数据表中的数据。

上面的查询方法都是返回一个 Cursor 对象，Android 中的 Cursor 类似于 JDBC 的 ResultSet，Cursor 同样提供了如下方法来移动查询结果的记录指针。

- move(int offset)：将记录指针向上或向下移动指定的行数。offset 为正数就是向下移动；为负数就是向上移动。
- boolean moveToFirst()：将记录指针移动到第一行，如果移动成功则返回 true。
- boolean moveToLast()：将记录指针移动到最后一行，如果移动成功则返回 true。
- boolean moveToNext()：将记录指针移动到下一行，如果移动成功则返回 true。
- boolean moveToPosition(int position)：将记录指针移动到指定行，如果移动成功则返回 true。
- boolean moveToPrevious()：将记录指针移动到上一行，如果移动成功则返回 true。

一旦将记录指针移动到指定行之后，接下来就可以调用 Cursor 的 getXxx()方法获取该行的指定列的数据了。

提示：

其实如果读者具有 JDBC 编程的经验，则完全可以把 SQLiteDatabase 当成 JDBC 中 Connection 和 Statement 的混合体——因为 SQLiteDatabase 既代表了与数据库的连接，也可直接用于执行 SQL 操作；而 Android 中 Cursor 则可当成 ResultSet，而且 Cursor 提供了更多便捷的方法来操作结果集。实际上对于一个 Java 程序员来说，JDBC 几乎是必备的编程基础，如果读者需要详细学习 JDBC 的相关知识，可以参考疯狂 Java 体系的《疯狂 Java 讲义》一书。

8.3.2 创建数据库和表

前面已经讲到，使用 SQLiteDatabase 的静态方法即可打开或创建数据库，例如如下代码：

```
SQLiteDatabase.openOrCreateDatabase("/mnt/db/temp.db3",null);
```

上面的代码就用于打开或创建一个 SQLite 数据库，如果/mnt/db/目录下的 temp.db3 文件（该文件就是一个数据库）存在，那么程序就是打开该数据库；如果该文件不存在，则上面的代码将会在该目录下创建 temp.db3 文件（即对应于数据库）。

上面的代码中没有指定 SQLiteDatabase.CursorFactory 参数，该参数是一个用于返回 Cursor 的工厂，如果指定该参数为 null，则意味着使用默认的工厂。

上面的代码即可返回一个 SQLiteDatabase 对象，该对象的 execSQL()方法可执行任意的 SQL 语句，因此可通过如下代码在程序中创建数据表：

```
// 定义建表语句
sql = "create table user_inf(user_id integer primary key ,"
    + " user_name varchar(255),"
    + " user_pass varchar(255))"
// 执行 SQL 语句
db.execSQL(sql);
```

在程序中执行上面的代码即可在数据库中创建一个数据表。

8.3.3 使用 SQL 语句操作 SQLite 数据库

正如前面提到的，SQLiteDatabase 的 execSQL()方法可执行任意的 SQL 语句，包括带占位符的 SQL 语句。但由于该方法没有返回值，因此一般用于执行 DDL 语句或 DML 语句；如果需要执行查询语句，则可调用 SQLiteDatabase 的 rawQuery(String sql, String[] selectionArgs)方法。

例如如下代码可用于执行 DML 语句：

```
// 执行插入语句
db.execSQL("insert into news_inf values(null , ? , ?)"
    , new String[]{title , content});
```

下面的程序示范了如何在 Android 应用中操作 SQLite 数据库。该程序提供了两个文本框，用户可以在这两个文本框中输入内容，当用户单击“插入”按钮时这两个文本框的内容将会被插入数据库。

程序清单：codes\08\8.3\DBTest\app\src\main\java\org\crazyit\db\MainActivity.java

```
public class MainActivity extends Activity
{
```

```
SQLiteDatabase db;
Button bn = null;
ListView listView;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 创建或打开数据库（此处需要使用绝对路径）
    db = SQLiteDatabase.openOrCreateDatabase(
        this.getFilesDir().toString()
        + "/my.db3", null); // ①
    listView = (ListView) findViewById(R.id.show);
    bn = (Button) findViewById(R.id.ok);
    bn.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View source)
        {
            // 获取用户输入
            String title = ((EditText) findViewById(
                R.id.title)).getText().toString();
            String content = ((EditText) findViewById(R.id.content))
                .getText().toString();
            try
            {
                insertData(db, title, content);
                Cursor cursor = db.rawQuery("select * from news_inf"
                    , null);
                inflateList(cursor);
            }
            catch (SQLiteException se)
            {
                // 执行 DDL 创建数据表
                db.execSQL("create table news_inf(_id integer"
                    + " primary key autoincrement,"
                    + " news_title varchar(50),"
                    + " news_content varchar(255))");
                // 执行 insert 语句插入数据
                insertData(db, title, content);
                // 执行查询
                Cursor cursor = db.rawQuery("select * from news_inf"
                    , null);
                inflateList(cursor);
            }
        }
    });
}
private void insertData(SQLiteDatabase db
    , String title, String content)  // ②
{
    // 执行插入语句
    db.execSQL("insert into news_inf values(null , ? , ?)"
        , new String[] {title, content });
}
private void inflateList(Cursor cursor)
{
    // 填充 SimpleCursorAdapter
    SimpleCursorAdapter adapter = new SimpleCursorAdapter(
        MainActivity.this,
        R.layout.line, cursor,
        new String[] { "news_title", "news_content" }
        , new int[] {R.id.my_title, R.id.my_content },
```

```
            CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);  // ③
        // 显示数据
        listView.setAdapter(adapter);
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 退出程序时关闭 SQLiteDatabase
        if (db != null && db.isOpen())
        {
            db.close();
        }
    }
}
```

上面的程序中①号粗体字代码用于创建或打开 SQLite 数据库。当用户单击程序中的“插入”按钮时，程序会调用②号粗体字代码向底层数据表中插入一行记录，并执行查询语句，把底层数据表中的记录查询出来，而且使用 ListView 将查询结果（Cursor）显示出来。

程序中③号粗体字代码用于将 Cursor 封装成 SimpleCursorAdapter，这个 SimpleCursorAdapter 实现了 Adapter 接口，因此可以作为 ListView 或 GridView 的内容适配器。

SimpleCursorAdapter 的构造器参数与 SimpleAdapter 的构造器参数大致相同，区别是 SimpleAdapter 负责封装集合元素为 Map 的 List，而 SimpleCursorAdapter 负责封装 Cursor——如果我们把 Cursor 里的结果集当成 List 集合，Cursor 里的每一行当成 Map 处理（以数据列的列名为 key，数据列的值为 value），那么 SimpleCursorAdapter 与 SimpleAdapter 就统一起来了。

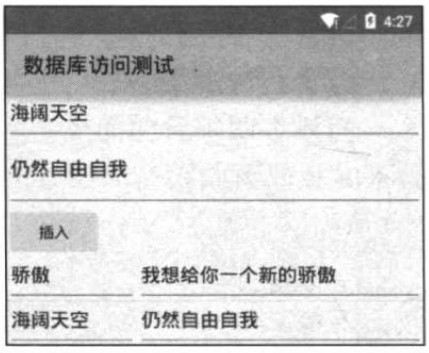


图 8.7　数据库访问示例

运行上面的程序，将看到如图 8.7 所示的界面。

正如上面的程序中所看到的，当程序不断地向底层数据表中插入数据时，程序中 ListView 将可以把底层数据表中数据显示出来。

需要指出的是，使用 SimpleCursorAdapter 封装 Cursor 时要求底层数据表的主键列的列名为_id，因为 SimpleCursorAdapter 只能识别列名为_id 的主键。因此上面的程序创建数据表时指定了主键列的列名为_id；否则就会出现 java.lang.IllegalArgumentException: column '_id' does not exist 错误。

上面的程序中我们重写了 Activity 的 onDestroy()方法，当应用程序退出该 Activity 时将会回调该方法，程序在该方法中关闭了 SQLiteDatabase——就像 JDBC 编程中需要关闭 Statement 和 Connection 一样，这里也需要关闭 SQLiteDatabase；否则可能引发资源泄漏。总结起来，使用 SQLiteDatabase 进行数据库操作的步骤如下。

1. 获取 SQLiteDatabase 对象，它代表了与数据库的连接。
2. 调用 SQLiteDatabase 的方法来执行 SQL 语句。
3. 操作 SQL 语句的执行结果，比如用 SimpleCursorAdapter 封装 Cursor。
4. 关闭 SQLiteDatabase，回收资源。

8.3.4　使用 sqlite3 工具

在 Android SDK 的 platform-tools 目录下提供了一个 sqlite3.exe 文件，它是一个简单的 SQLite 数据库管理工具，类似于 MySQL 提供的命令行窗口。有些时候，开发者利用该工具来

查询、管理数据库。

例如，我们把上面的应用程序所生成的 my.db3 导出到本地计算机上的 F:\盘根目录下，接下来可运行如下命令来启动 SQLite 数据库：

```
sqlite3 f:/my.db3
```

运行上面的命令，可以看到如图 8.8 所示的窗口。

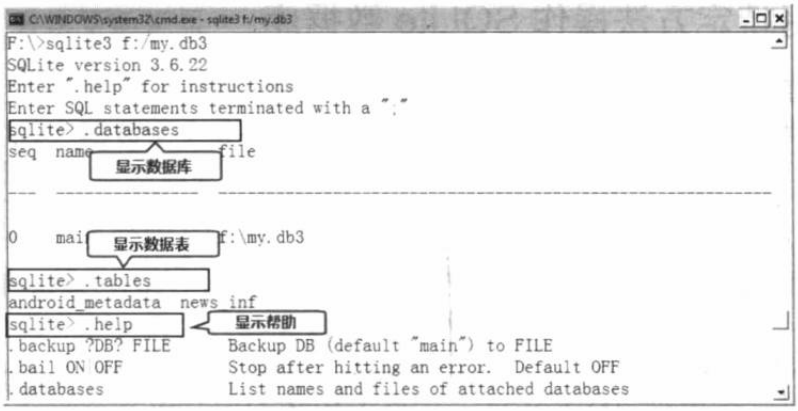


图 8.8 使用 sqlite3 工具

从图 8.8 可以看出，sqlite3 中常用的命令如下。

- .databases：查看当前数据库。
- .tables：查看当前数据库里的数据表。
- .help：查看 sqlite3 支持的命令。

当然，sqlite3 还支持一些常用的命令，当开发者在图 8.8 所示窗口中输入.help 之后，该工具将会列出这些命令。除此之外，SQLite 数据库还支持绝大部分常用的 SQL 语句，开发者可在图 8.8 所示窗口中运行各种 DDL、DML、查询语句来测试它们。

提示：

SQLite 数据库所支持的 SQL 语句与 MySQL 大致相同，开发者完全可以把已有的 MySQL 经验“移植”到 SQLite 数据库上。当然，当 Android 应用提示某条 SQL 语句有语法错误时，最好先利用 sqlite3 这个工具类测试这条语句，以保证这条 SQL 语句的语法正确。

需要指出的是，SQLite 内部只支持 NULL、INTEGER、REAL（浮点数）、TEXT（文本）和 BLOB（大二进制对象）这 5 种数据类型，但实际上 SQLite 完全可以接受 varchar(n)、char(n)、decimal(p,s) 等数据类型，只不过 SQLite 会在运算或保存时将它们转换为上面 5 种数据类型中相应的类型。

除此之外，SQLite 还有一个特点：它允许把各种类型的数据保存到任何类型字段中，开发者可以不用关心声明该字段所使用的数据类型。例如，程序可以把字符串类型的值存入 INTEGER 类型的字段中，也可以把数值类型的值存入布尔类型的字段中……但有一种情况例外：定义为 INTEGER PRIMARY KEY 的字段只能存储 64 位整数，当向这种字段中保存除整数以外的其他类型的数据时，SQLite 会产生错误。

由于 SQLite 允许存入数据时忽略底层数据列实际的数据类型，因此在编写建表语句时可以省略数据列后面的类型声明。例如，如下 SQL 语句对于 SQLite 也是正确的：

```
create table my_test
(
    _id integer primary key autoincrement,
```

```
    name ,
    pass ,
    gender
);
```

开发者可以把 MySQL 开发经验直接移植到 SQLite 数据库上，如果开发者需要了解更多关于 SQL 语法、MySQL 数据库的知识，请参考疯狂 Java 体系的《疯狂 Java 讲义》一书。

8.3.5 使用特定方法操作 SQLite 数据库

如果开发者对 SQL 语法不熟悉，甚至以前从未使用过任何数据库，Android 的 SQLiteDatabase 提供了 insert、upate、delete 或 query 语句来操作数据库。

提示：

虽然 Android 提供了这些所谓的“便捷”方法来操作 SQLite 数据库，但在笔者看来这些方法纯属“鸡肋”，对于一个程序员而言，SQL 语法可以说是基本功中的基本功——你见过不会 1+1=2 的数学工作者吗？

不过，既然 Android 提供了这些方法，这里也简单介绍一下。

1. 使用 insert 方法插入记录

SQLiteDatabase 的 insert 方法的签名为 long insert(String table, String nullColumnHack, ContentValues values)，这个插入方法的参数说明如下。

- table：代表想插入数据的表名。
- nullColumnHack：代表强行插入 null 值的数据列的列名。当 values 参数为 null 或不包含任何 key-value 对时该参数有效。
- values：代表一行记录的数据。

insert 方法插入的一行记录使用 ContentValues 存放，ContentValues 类似于 Map，它提供了 put(String key, Xxx value)（其中 key 为数据列的列名）方法用于存入数据，getAsXxx(String key) 方法用于取出数据。

例如如下语句：

```
ContentValues values = new ContentValues();
values.put("name", "孙悟空");
values.put("age", 500);
// 返回新添记录的行号，该行号是一个内部值，与主键 id 无关，发生错误返回-1
long rowid = db.insert("person_inf", null, values);
```

不管第三个参数是否包含数据，执行 insert()方法总会添加一条记录，如果第三个参数为空，则会添加一条除主键之外其他字段值都为 null 的记录。

insert()方法的底层实际上依然是通过构造 insert SQL 语句来进行插入的，因此它生成的 SQL 语句总是形如下面的语句：

```
// ContentValues 里 key-value 对的数量决定了下面的 key-value 对
insert into <表名>(key1 , key 2 ...)
values(value1 , value2 ...)
```

此时如果第三个参数为 null 或其中 key-value 对的数量为 0，由于 insert()方法还会按此方式生成一条 insert 语句，此时的 insert 语句为：

```
// ContentValues 里 key-value 对的数量决定了下面的 key-value 对
insert into <表名>()
values()
```

上面的 SQL 语句显然有问题。为了满足 SQL 语法的需要，insert 语句必须给定一个列名，如：insert into person(name) values(null)，这个 name 列名就由第二个参数来指定。由此可见，当 ContentValues 为 null 或它包含的 key-value 对的数量为 0 时，第二个参数就会起作用了。

一般来说，第二个参数指定的列名不应该是主键列的列名，也不应该是非空列的列名；否则强行往这些数据列插入 null 会引发异常。

2. 使用 update 方法更新记录

SQLiteDatabase 的 update 方法的签名为 update(String table, ContentValues values, String whereClause, String[] whereArgs)，这个更新方法的参数说明如下。

- table：代表想更新数据的表名。
- values：代表想更新的数据。
- whereClause：满足该 whereClause 子句的记录将会被更新。
- whereArgs：用于为 whereClause 子句传入参数。

该方法返回受此 update 语句影响的记录的条数。

例如，我们想更新 person_inf 表中所有主键大于 20 的人的人名，可调用如下方法：

```
ContentValues values = new ContentValues();
// 存放更新后的人名
values.put("name", "新人名");
int result= db.update("person_inf", values , "_id>?" , new Integer[]{20});
```

实际上 update 方法底层对应的 SQL 语句如下：

```
update <table>
set key1=value1 , key2=value2 ...
where <whereClause>
```

其中 whereArgs 参数用于向 whereClause 中传入参数。

3. 使用 delete 方法删除记录

SQLiteDatabase 的 delete 方法的签名为 delete(String table , String whereClause, String[] whereArgs)，这个删除方法的参数说明如下。

- table：代表想删除数据的表名。
- whereClause：满足该 whereClause 子句的记录将会被删除。
- whereArgs：用于为 whereClause 子句传入参数。

该方法返回受此 delete 语句影响的记录的条数。

例如，我们想删除 person_inf 表中所有人名以“孙”开头的记录，可调用如下方法：

```
int result= db.delete("person_inf", "person_name like ?" , new String[]{"孙_"});
```

实际上 delete 方法底层对应的 SQL 语句如下：

```
delete <table>
where <whereClause>
```

4. 使用 query 方法查询记录

SQLiteDatabase 的 query 方法的签名为 Cursor query(boolean distinct, String table, String[] columns, String whereClause, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)，这个 query 方法的参数说明如下。

- distinct：指定是否去除重复记录。
- table：执行查询数据的表名。

- columns：要查询出来的列名。相当于 select 语句 select 关键字后面的部分。
- whereClause：查询条件子句，相当于 select 语句 where 关键字后面的部分，在条件子句中允许使用占位符“?”。
- selectionArgs：用于为 whereClause 子句中的占位符传入参数值，值在数组中的位置与占位符在语句中的位置必须一致；否则就会有异常。
- groupBy：用于控制分组。相当于 select 语句 group by 关键字后面的部分
- having：用于对分组进行过滤。相当于 select 语句 having 关键字后面的部分
- orderBy：用于对记录进行排序。相当于 select 语句 order by 关键字后面的部分，如 personid desc, age asc。
- limit：用于进行分页。相当于 select 语句 limit 关键字后面的部分，如 5,10。

看到这个方法的设计，笔者忍不住想再次表达对这个方法的不满：如果读者完全不懂 SQL 语句，那需要花多少时间才能理解这个方法中这么多参数的设置。如果读者愿意花时间去理解这个方法中各参数的设置，那么所花的时间已经足够去掌握这条 select 语句的语法格式了。

当然，这个 query()方法也并非完全一无是处：当应用程序需要进行“条件不确定”的查询（即查询条件需要动态改变的查询）时，使用这个 query()方法可以避免手动拼接 SQL 语句。

例如，想查询出 person_inf 表中人名以“孙”开头的记录，可使用如下语句：

```
Cursor cursor = db.query("person_inf", new String[]{"_id,name,age"}
    , "name like ?", new String[]{"孙%"}
    , null, null, "personid desc", "5,10");
// 处理结果集
cursor.close();
```

8.3.6 事务

SQLiteDatabase 中包含如下两个方法来控制事务。

- beginTransaction()：开始事务。
- endTransaction()：结束事务。

除此之外，SQLiteDatabase 还提供了如下方法来判断当前上下文是否处于事务环境中。

- inTransaction()：如果当前上下文处于事务中，则返回 true；否则返回 false。

当程序执行 endTransaction()方法时将会结束事务——那到底是提交事务，还是回滚事务呢？这取决于 SQLiteDatabase 是否调用了 setTransactionSuccessful()方法来设置事务标志，如果程序在事务执行中调用该方法设置了事务成功则提交事务；否则程序将会回滚事务。

示例代码如下：

```
// 开始事务
db.beginTransaction();
try
{
    // 执行 DML 语句
    ...
    // 调用该方法设置事务成功；否则 endTransaction()方法将回滚事务
    db.setTransactionSuccessful();
}
finally
{
    // 由事务的标志决定是提交事务还是回滚事务
    db.endTransaction();
}
```

8.3.7 SQLiteOpenHelper 类

在上一个示例程序中，我们为了判断底层数据库是否包含 news_inf 数据表，采用的处理方法十分烦琐：程序先尝试向 news_inf 数据表中插入记录，如果程序抛出异常，在异常捕获的 catch 块中创建 news_inf 数据表，然后再插入记录。那么是否有一种更优雅的方式来处理这种问题呢？有，Android 提供了 SQLiteOpenHelper 类来处理这种问题。

在实际项目中很少使用 SQLiteDatabase 的方法来打开数据库，通常都会继承 SQLiteOpenHelper 开发子类，并通过该子类的 getReadableDatabase()、getWritableDatabase()方法打开数据库。

SQLiteOpenHelper 是 Android 提供的一个管理数据库的工具类，可用于管理数据库的创建和版本更新。一般的用法是创建 SQLiteOpenHelper 的子类，并扩展它的 onCreate (SQLiteDatabase db)和 onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)方法。

SQLiteOpenHelper 包含如下常用的方法。

- synchronized SQLiteDatabase getReadableDatabase()：以读写的方式打开数据库对应的 SQLiteDatabase 对象。
- synchronized SQLiteDatabase getWritableDatabase()：以写的方式打开数据库对应的 SQLiteDatabase 对象。
- abstract void onCreate(SQLiteDatabase db)：当第一次创建数据库时回调该方法。
- abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)：当数据库版本更新时回调该方法。
- synchronized void close()：关闭所有打开的 SQLiteDatabase 对象。

从上面的方法介绍中不难看出，SQLiteOpenHelper 提供了 getReadableDatabase()、getWritableDatabase()两个方法用于打开数据库连接，并提供了 close()方法来关闭数据库连接，而开发者需要做的就是重写它的两个抽象方法。

- onCreate(SQLiteDatabase db)：用于初次使用软件时生成数据库表。当调用 SQLiteOpenHelper 的 getWritableDatabase()或者 getReadableDatabase()方法获取用于操作数据库的 SQLiteDatabase 实例时，如果数据库不存在，Android 系统会自动生成一个数据库，接着调用 onCreate()方法，onCreate()方法在初次生成数据库时才会被调用。重写 onCreate()方法时，可以生成数据库表结构，以添加应用使用到的一些初始化数据。
- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)：用于升级软件时更新数据库表结构，此方法在数据库的版本发生变化时会被调用，该方法调用时 oldVersion 代表数据库之前的版本号，newVersion 代表数据库当前的版本号。那么在哪里指定数据库的版本号呢？当程序创建 SQLiteOpenHelper 对象时，必须指定一个 version 参数，该参数就决定了所使用的数据库的版本——也就是说，数据库的版本是由程序员控制的。只要某次创建 SQLiteOpenHelper 时指定的数据库版本号高于之前指定的版本号，系统就会自动触发 onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)方法，程序就可以在 onUpgrade()方法里面根据原版本号和目标版本号进行判断，即可根据版本号进行必需的表结构更新。

提示：

实际上，当应用程序升级表结构时，完全可能因为已有的数据导致升级失败。在这种时候程序可能需要先对数据进行转储，清空数据表中的记录，接着对数据表进行更新，当数据表更新完成后再将数据保存回来。

一旦得到了 SQLiteOpenHelper 对象之后，程序无须使用 SQLiteDatabase 的静态方法创建 SQLiteDatabase 实例，而且可以使用 getWritableDatabase()或 getReadableDatabase()方法来获取一个用于操作数据库的 SQLiteDatabase 实例。

其中 getWritableDatabase() 方法以写的方式打开数据库，一旦数据库的磁盘空间满了，数据库就只能读而不能写，倘若使用 getWritableDatabase()打开数据库就会出错。getReadableDatabase()方法先以读写方式打开数据库，如果数据库的磁盘空间满了，就会打开失败，当打开失败后会继续尝试以只读方式打开数据库。

下面以一个实例程序来说明 SQLiteOpenHelper 的功能与用法。

实例：英文生词本

该实例允许用户将自己不熟悉的单词添加到系统数据库中，当用户需要查询某个单词或解释时，只要在程序中输入相应的关键词，程序中相应的条目就会显示出来。

该程序使用了 SQLiteOpenHelper 的子类来管理数据库连接，当程序通过 SQLiteOpenHelper 的子类来获取数据库连接时，如果数据库还不存在，系统会自动调用 onCreate(SQLiteDatabase db)方法来创建数据库；如果程序创建 SQLiteOpenHelper 的子类的实例时传入的数据库版本号高于之前的版本号，系统会自动调用 onUpgrade(SQLite Database db, int oldVersion, int newVersion)方法来更新数据库。

下面是 SQLiteOpenHelper 的子类代码。

程序清单：codes\08\8.3\Dict\app\src\main\java\org\crazyit\db\MyDatabaseHelper.java

```
public class MyDatabaseHelper extends SQLiteOpenHelper
{
    final String CREATE_TABLE_SQL =
        "create table dict(_id integer primary " +
        "key autoincrement , word , detail)";
    public MyDatabaseHelper(Context context, String name, int version)
    {
        super(context, name, null, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        // 第一次使用数据库时自动建表
        db.execSQL(CREATE_TABLE_SQL);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db
        , int oldVersion, int newVersion)
    {
        System.out.println("--------onUpdate Called--------"
            + oldVersion + "--->" + newVersion);
    }
}
```

上面的 MyDatabaseHelper 继承了 SQLiteOpenHelper，并重写了其基类的 onCreate(SQLiteDatabase db)方法，该方法中执行的建表语句用于初始化系统数据表。如果用户第一次使用该程序，系统将会自动调用 onCreate(SQLiteDatabase db)方法来初始化底层数据库。

MyDatabaseHelper 工具类的作用主要是管理数据库的初始化，并允许应用程序通过该工具类来获取 SQLiteDatabase 对象。接下来的程序就可通过该工具类获取 SQLiteDatabase 对象，并利用该对象操作数据库了。

本实例的程序界面比较简单，它提供了输入框让用户添加生词，并提供输入框让用户输入要查询的关键词，用户在查询文本框中输入关键词后单击“查找”按钮即可显示相应的生词条目。该程序的界面如图 8.9 所示。

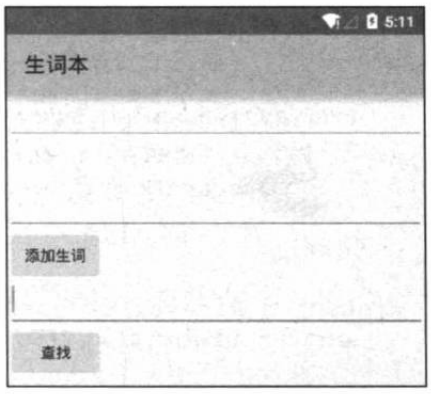


图 8.9 生词本界面

图 8.9 所示的界面中前两个文本框用于添加生词；第三个文本框用于输入想查询的关键词，当用户输入相应的关键词，并单击“查找”按钮后，系统将会查询相应的条目。下面是该生词本的主程序。

程序清单：codes\08\8.3\Dict\app\src\main\java\org\crazyit\db\MainActivity.java

```
public class MainActivity extends Activity
{
    MyDatabaseHelper dbHelper;
    Button insert = null;
    Button search = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建 MyDatabaseHelper 对象，指定数据库版本为 1，此处使用相对路径即可
        // 数据库文件会自动保存在程序的数据文件夹的 databases 目录下
        dbHelper = new MyDatabaseHelper(this, "myDict.db3", 1);
        insert = (Button) findViewById(R.id.insert);
        search = (Button) findViewById(R.id.search);
        insert.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String word = ((EditText) findViewById(R.id.word))
                    .getText().toString();
                String detail = ((EditText) findViewById(R.id.detail))
                    .getText().toString();
                // 插入生词记录
                insertData(dbHelper.getReadableDatabase(), word, detail);
                // 显示提示信息
                Toast.makeText(MainActivity.this, "添加生词成功！"
                    , Toast.LENGTH_LONG).show();
            }
        });
        search.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String key = ((EditText) findViewById(R.id.key)).getText()
                    .toString();
                // 执行查询
                Cursor cursor = dbHelper.getReadableDatabase().rawQuery(
                    "select * from dict where word like ? or detail like ?",
                    new String[] { "%" + key + "%", "%" + key + "%" });
                // 创建一个 Bundle 对象
                Bundle data = new Bundle();
```

```
            data.putSerializable("data", converCursorToList(cursor));
            // 创建一个 Intent
            Intent intent = new Intent(MainActivity.this
                , ResultActivity.class);
            intent.putExtras(data);
            // 启动 Activity
            startActivity(intent);
        }
    });
}
protected ArrayList<Map<String, String>>
    converCursorToList(Cursor cursor)
{
    ArrayList<Map<String, String>> result =
        new ArrayList<Map<String, String>>();
    // 遍历 Cursor 结果集
    while (cursor.moveToNext())
    {
        // 将结果集中的数据存入 ArrayList 中
        Map<String, String> map = new HashMap<>();
        // 取出查询记录中第 2 列、第 3 列的值
        map.put("word", cursor.getString(1));
        map.put("detail", cursor.getString(2));
        result.add(map);
    }
    return result;
}
private void insertData(SQLiteDatabase db, String word
    , String detail)
{
    // 执行插入语句
    db.execSQL("insert into dict values(null , ? , ?)"
        , new String[] {word, detail });
}
@Override
public void onDestroy()
{
    super.onDestroy();
    // 退出程序时关闭 MyDatabaseHelper 里的 SQLiteDatabase
    if (dbHelper != null)
    {
        dbHelper.close();
    }
}
}
```

上面的程序中第一行粗体字代码用于根据 SQLiteOpenHelper 来获取 SQLiteDatabase，用于执行插入数据、查询数据。程序重写了 Activity 的 onDestroy()方法，并在该方法中调用 SQLiteOpenHelper 的 close()方法来关闭数据库连接。

当程序第一次调用 getWritableDatabase() 或 getReadableDatabase() 方法后，SQLiteOpenHelper 会缓存已获得的 SQLiteDatabase 实例，在正常情况下 SQLiteDatabase 实例会维持数据库的打开状态，因此程序退出时应该关闭不再使用的 SQLiteDatabase。一旦 SQLiteOpenHelper 缓存了 SQLiteDatabase 实例之后，多次调用 getWritableDatabase()或 getReadableDatabase()方法得到的都是同一个 SQLiteDatabase 实例。

上面的程序执行查询后并未在该 Activity 中显示查询得到的条目，而是使用另一个 Activity：ResultActivity 来显示查询结果，ResultActivity 只是提供了一个 ListView 来显示查询得到的条目。为了把 dict 查询得到的条目传给 ResultActivity，程序将查询得到的条目封装成 List<Map<String，String>>后传给 ResultActivity，ResultActivity 就可利用 ListView 来显示查询

结果了。

下面是 ResultActivity 的代码。

程序清单：codes\08\8.3\Dict\app\src\main\java\org\crazyit\db\ResultActivity.java

```
public class ResultActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.popup);
        ListView listView = (ListView) findViewById(R.id.show);
        Intent intent = getIntent();
        // 获取该 intent 所携带的数据
        Bundle data = intent.getExtras();
        // 从 Bundle 数据包中取出数据
        @SuppressWarnings("unchecked")
        List<Map<String, String>> list = (List<Map<String, String>>)
            data.getSerializable("data");
        // 将 List 封装成 SimpleAdapter
        SimpleAdapter adapter = new SimpleAdapter(ResultActivity.this
            , list,
            R.layout.line, new String[] { "word", "detail" }
            , new int[] {R.id.word, R.id.detail });
        // 填充 ListView
        listView.setAdapter(adapter);
    }
}
```

上面的 ResultActivity 只是一个普通的 Activity，但我们在 AndroidManifest.xml 文件中将该 Activity 设为对话框风格的 Activity，这样就可让应用程序以对话框来显示查询结果了。查询结果如图 8.10 所示。

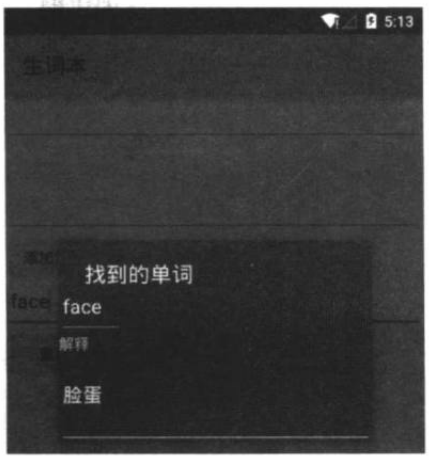


图 8.10 查询生词

8.4 手势（Gesture）

所谓手势，其实是指用户手指或触摸笔在触摸屏上的连续触碰行为，比如在屏幕上从左至右划出的一个动作，就是手势；再比如在屏幕上画出一个圆圈也是手势。手势这种连续的触碰会形成某个方向上的移动趋势，也会形成一个不规则的几何图形。Android 对两种手势行为都提供了支持。

- 对于第一种手势行为，Android 提供了手势检测，并为手势检测提供了相应的监听器。
- 对于第二种手势行为，Android 允许开发者添加手势，并提供了相应的 API 识别用户手势。

8.4.1 手势检测

Android 为手势检测提供了一个 GestureDetector 类，GestureDetector 实例代表了一个手势检测器，创建 GestureDetector 时需要传入一个 GestureDetector.OnGestureListener 实例，GestureDetector.OnGestureListener 就是一个监听器，负责对用户的手势行为提供响应。

GestureDetector.OnGestureListener 里包含的事件处理方法如下。

- boolean onDown(MotionEvent e)：当触碰事件按下时触发该方法。
- boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)：当用户手指在触摸屏上“拖过”时触发该方法。其中 velocityX、velocityY 代表“拖过”动作在横向、纵向上的速度。
- abstract void onLongPress(MotionEvent e)：当用户手指在屏幕上长按时触发该方法。
- boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)：当用户手指在屏幕上“滚动”时触发该方法。
- void onShowPress(MotionEvent e)：当用户手指在触摸屏上按下，而且还未移动和松开时触发该方法。
- boolean onSingleTapUp(MotionEvent e)：用户手指在触摸屏上的轻击事件将会触发该方法。

关于 GestureDetector.OnGestureListener 监听器里各方法的触发时机，仅从文字上表述总显得比较抽象而且难于理解，下面将以一个最简单的例子来让读者理解各方法的触发时机。

使用 Android 的手势检测只需两个步骤。

① 创建一个 GestureDetector 对象。创建该对象时必须实现一个 GestureDetector.OnGestureListener 监听器实例。

② 为应用程序的 Activity（偶尔也可为特定组件）的 TouchEvent 事件绑定监听器，在事件处理中指定把 Activity（或特定组件）上的 TouchEvent 事件交给 GestureDetector 处理。

经过上面两个步骤之后，Activity（或特定组件）上的 TouchEvent 事件就会交给 GestureDetector 处理，而 GestureDetector 就会检测是否触发了特定的手势动作。

下面的程序测试了用户的不同动作到底触发哪种手势动作。

程序清单：codes\08\8.4\GestureTest\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
    implements OnGestureListener
{
    // 定义手势检测器实例
    GestureDetector detector;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建手势检测器
        detector = new GestureDetector(this, this);
    }
    // 将该 Activity 上的触碰事件交给 GestureDetector 处理
    @Override
    public boolean onTouchEvent(MotionEvent me)
    {
        return detector.onTouchEvent(me);
    }
    @Override
    public boolean onDown(MotionEvent arg0)
    {
        Toast.makeText(this,"onDown"
            , Toast.LENGTH_SHORT).show();
        return false;
    }
    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2
```

```
        , float velocityX, float velocityY)
    {
        Toast.makeText(this , "onFling"
            , Toast.LENGTH_SHORT).show();
        return false;
    }
    @Override
    public void onLongPress(MotionEvent e)
    {
        Toast.makeText(this ,"onLongPress"
            , Toast.LENGTH_SHORT).show();
    }
    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2
        , float distanceX, float distanceY)
    {
        Toast.makeText(this ,"onScroll" ,
            Toast.LENGTH_SHORT).show();
        return false;
    }
    @Override
    public void onShowPress(MotionEvent e)
    {
        Toast.makeText(this ,"onShowPress"
            , Toast.LENGTH_SHORT).show();
    }
    @Override
    public boolean onSingleTapUp(MotionEvent e)
    {
        Toast.makeText(this ,"onSingleTapUp"
            , Toast.LENGTH_SHORT).show();
        return false;
    }
}
```

上面程序中的第一行粗体字代码创建了一个 GestureDetector 对象，创建该对象时传入了 this 作为参数，这表明该 Activity 本身将会作为 GestureDetector.OnGestureListener 监听器，所以该 Activity 实现了该接口，并实现了该接口里的全部方法。程序中第二行粗体字代码指定把 Activity 上的 TouchEvent 交给 GestureDetector 处理。

图 8.11 手势检测

运行上面的程序，当用户随意地在屏幕上触碰时，程序将会检测到用户到底执行了哪些手势，如图 8.11 所示。

掌握了用户在屏幕上的哪些动作对应于哪些手势之后，接下来再以两个实例来介绍 Android 手势检测在实际项目中的应用。

实例：通过手势缩放图片

前面介绍过图片缩放的技术实现，通过 Matrix 即可实现图片缩放，但之前的图片缩放要么通过按钮控制，要么通过 SeekBar 来控制，这些缩放方式太“传统”了。而本实例所介绍的图片缩放则“炫”得多：用户只要在图片上随意地“挥动”手指，图片就可被缩放——从左向右挥动时图片被放大，从右向左挥动时图片被缩小；挥动速度越快，缩放比越大。

该实例的程序界面布局很简单，只在界面中间定义一个 ImageView 来显示图片即可。该程序的思路是使用一个 GestureDetector 来检测用户手势，并根据用户手势在横向的速度缩放图片。该程序代码如下。

程序清单：codes\08\8.4\GestureZoom\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
    implements OnGestureListener
{
    // 定义手势检测器实例
    GestureDetector detector;
    ImageView imageView;
    // 初始的图片资源
    Bitmap bitmap;
    // 定义图片的宽、高
    int width, height;
    // 记录当前的缩放比
    float currentScale = 1;
    // 控制图片缩放的 Matrix 对象
    Matrix matrix;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建手势检测器
        detector = new GestureDetector(this, this);
        imageView = (ImageView) findViewById(R.id.show);
        matrix = new Matrix();
        // 获取被缩放的源图片
        bitmap = BitmapFactory.decodeResource(
            this.getResources(), R.drawable.flower);
        // 获得位图宽
        width = bitmap.getWidth();
        // 获得位图高
        height = bitmap.getHeight();
        // 设置 ImageView 初始化时显示的图片
        imageView.setImageBitmap(BitmapFactory.decodeResource(
            this.getResources(), R.drawable.flower));
    }
    @Override
    public boolean onTouchEvent(MotionEvent me)
    {
        // 将该 Activity 上的触碰事件交给 GestureDetector 处理
        return detector.onTouchEvent(me);
    }
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
        float velocityX, float velocityY)  // ②
    {
        velocityX = velocityX > 4000 ? 4000 : velocityX;
        velocityX = velocityX < -4000 ? -4000 : velocityX;
        // 根据手势的速度来计算缩放比，如果 velocityX>0，则放大图片；否则缩小图片
        currentScale += currentScale * velocityX / 4000.0f;
        // 保证 currentScale 不会等于 0
        currentScale = currentScale > 0.01 ? currentScale: 0.01f;
        // 重置 Matrix
        matrix.reset();
        // 缩放 Matrix
        matrix.setScale(currentScale, currentScale, 160, 200);
        BitmapDrawable tmp = (BitmapDrawable)
            imageView.getDrawable();
        // 如果图片还未回收，先强制回收该图片
        if (!tmp.getBitmap().isRecycled())  // ①
        {
```

```
            tmp.getBitmap().recycle();
        }
        // 根据原始位图和 Matrix 创建新图片
        Bitmap bitmap2 = Bitmap.createBitmap(bitmap, 0, 0
            , width, height, matrix, true);
        // 显示新的位图
        imageView.setImageBitmap(bitmap2);
        return true;
    }
    @Override
    public boolean onDown(MotionEvent arg0)
    {
        return false;
    }
    @Override
    public void onLongPress(MotionEvent event)
    {
    }
    @Override
    public boolean onScroll(MotionEvent event1
        , MotionEvent event2, float distanceX, float distanceY)
    {
        return false;
    }
    @Override
    public void onShowPress(MotionEvent event)
    {
    }
    @Override
    public boolean onSingleTapUp(MotionEvent event)
    {
        return false;
    }
}
```

上面程序的处理方式与前一个程序的处理方式如出一辙：第一行粗体字代码用于创建 GestureDetector 对象，第二行粗体字代码指定把 Activity 的 OnTouch 事件交给 GestureDetector 处理。

该程序与前一个程序的不同之处在于：程序实现 GestureDetector.OnGestureListener 监听器时，只实现了②号代码所标出的 onFling(MotionEvent event1, MotionEvent event2,float velocityX, float velocityY)方法，并在该方法内根据 velocityX 参数（横向上的拖动速度）来计算图片缩放比，这样该程序即可根据用户的“手势”来缩放图片了。

实例：通过手势实现翻页效果

本实例的手势检测思路还是一样：把 Activity 的 TouchEvent 交给 GestureDetector 处理。这个程序的特殊之处在于：该程序使用了一个 ViewFlipper 组件，ViewFlipper 可使用动画控制多个组件之间的切换效果。

本程序通过 GestureDetector 来检测用户的手势动作，并根据手势动作来控制 ViewFlipper 包含的 View 组件的切换，从而实现翻页效果。

该程序的界面布局代码如下。

程序清单：codes\08\8.4\GestureFlip\app\src\main\res\layout\main.xml

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

```
    android:orientation="vertical">
    <!-- 定义 ViewFlipper 组件 -->
<ViewFlipper android:id="@+id/flipper"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
```

上面的界面布局文件中定义了一个 ViewFlipper 组件，该组件可控制多个 View 的动画切换。该实例的程序代码如下。

程序清单：codes\08\8.4\GestureFlip\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
    implements OnGestureListener
{
    // ViewFlipper 实例
    ViewFlipper flipper;
    // 定义手势检测器实例
    GestureDetector detector;
    // 定义一个动画数组，用于为 ViewFlipper 指定切换动画效果
    Animation[] animations = new Animation[4];
    // 定义手势动作两点之间的最小距离
    final int FLIP_DISTANCE = 50;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建手势检测器
        detector = new GestureDetector(this, this);
        // 获得 ViewFlipper 实例
        flipper = (ViewFlipper) this.findViewById(R.id.flipper);
        // 为 ViewFlipper 添加 6 个 ImageView 组件
        flipper.addView(addImageView(R.drawable.java));
        flipper.addView(addImageView(R.drawable.javaee));
        flipper.addView(addImageView(R.drawable.ajax));
        flipper.addView(addImageView(R.drawable.android));
        flipper.addView(addImageView(R.drawable.html));
        flipper.addView(addImageView(R.drawable.swift));
        // 初始化 Animation 数组
        animations[0] = AnimationUtils.loadAnimation(
            this, R.anim.left_in);
        animations[1] = AnimationUtils.loadAnimation(
            this, R.anim.left_out);
        animations[2] = AnimationUtils.loadAnimation(
            this, R.anim.right_in);
        animations[3] = AnimationUtils.loadAnimation(
            this, R.anim.right_out);
    }
    // 定义添加 ImageView 的工具方法
    private View addImageView(int resId)
    {
        ImageView imageView = new ImageView(this);
        imageView.setImageResource(resId);
        imageView.setScaleType(ImageView.ScaleType.CENTER);
        return imageView;
    }
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
        float velocityX, float velocityY)
    {
        // 如果第一个触点事件的 X 坐标大于第二个触点事件的 X 坐标超过 FLIP_DISTANCE
        // 也就是手势从右向左滑
```

```
        if (event1.getX() - event2.getX() > FLIP_DISTANCE)
        {
            // 为 flipper 设置切换的动画效果
            flipper.setInAnimation(animations[0]);
            flipper.setOutAnimation(animations[1]);
            flipper.showPrevious();
            return true;
        }
        // 如果第二个触点事件的 X 坐标大于第一个触点事件的 X 坐标超过 FLIP_DISTANCE
        // 也就是手势从左向右滑
        else if (event2.getX() - event1.getX() > FLIP_DISTANCE)
        {
            // 为 flipper 设置切换的动画效果
            flipper.setInAnimation(animations[2]);
            flipper.setOutAnimation(animations[3]);
            flipper.showNext();
            return true;
        }
        return false;
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 将该 Activity 上的触碰事件交给 GestureDetector 处理
        return detector.onTouchEvent(event);
    }
    @Override
    public boolean onDown(MotionEvent arg0)
    {
        return false;
    }
    @Override
    public void onLongPress(MotionEvent event)
    {
    }
    @Override
    public boolean onScroll(MotionEvent event1
        , MotionEvent event2, float arg2, float arg3)
    {
        return false;
    }
    @Override
    public void onShowPress(MotionEvent event)
    {
    }
    @Override
    public boolean onSingleTapUp(MotionEvent event)
    {
        return false;
    }
}
```

该程序同样只是实现了 GestureDetector.OnGestureListener 的 onFling()方法。上面的程序中粗体字代码负责实现：当 event1.getX() - event2.getX()的距离大于特定距离时，即可判断用户手势为从右向左滑动，此时设置 ViewFlipper 采用动画方式切换为上一个 View；当 event2.getX() - event1.getX()的距离大于特定距离时，即可判断用户手势为从左向右滑动，此时设置 ViewFlipper 采用动画方式切换为下一个 View——这样就实现了所谓的“翻页”效果。

运行上面的程序，并在屏幕上执行“拖动”手势，即可看到 ViewFlipper 内组件切换的效果，如图 8.12 所示。

图 8.12 采用手势检测实现翻页效果

8.4.2 增加手势

Android 除了提供手势检测之外，还允许应用程序把用户手势（多个持续的触摸事件在屏幕上形成特定的形状）添加到指定文件中，以备以后使用——如果程序需要，当用户下次再次画出该手势时，系统将可识别该手势。

Android 使用 GestureLibrary 来代表手势库，并提供了 GestureLibraries 工具类来创建手势库。GestureLibraries 提供了如下 4 个静态方法从不同位置加载手势库。

- static GestureLibrary fromFile(String path)：从 path 代表的文件中加载手势库。
- static GestureLibrary fromFile(File path)：从 path 代表的文件中加载手势库。
- static GestureLibrary fromPrivateFile(Context context, String name)：从指定应用程序的数据文件夹的 name 文件中加载手势库。
- static GestureLibrary fromRawResource(Context context, int resourceId)：从 resourceId 所代表的资源中加载手势库。

一旦在程序中获得了 GestureLibrary 对象之后，该对象提供了如下方法来添加手势、识别手势。

- void addGesture(String entryName, Gesture gesture)：添加一个名为 entryName 的手势。
- Set<String> getGestureEntries()：获取该手势库中的所有手势的名称。
- ArrayList<Gesture> getGestures(String entryName)：获取 entryName 名称对应的全部手势。
- ArrayList<Prediction> recognize(Gesture gesture)：从当前手势库中识别与 gesture 匹配的全部手势。
- void removeEntry(String entryName)：删除手势库中 entryName 对应的手势。
- void removeGesture(String entryName, Gesture gesture)：删除手势库中 entryName、gesture 对应的手势。
- boolean save()：当向手势库中添加手势或从中删除手势后调用该方法保存手势库。

Android 除了提供了 GestureLibraries、GestureLibrary 来管理手势之外，还提供了一个专门的手势编辑组件：GestureOverlayView，该组件就像一个“绘图组件”，只是用户在组件上绘制的不是图形，而是手势。

为了监听 GestureOverlayView 组件上的手势事件，Android 为 GestureOverlayView 提供了

OnGestureListener、OnGesturePerformedListener、OnGesturingListener 三个监听器接口，这些监听器所包含的方法分别用于响应手势开始、结束、完成、取消等事件，开发者可根据实际需要来选择不同的监听器——一般来说，OnGesturePerformedListener 是最常用的监听器，它可用于在手势事件完成时提供响应。

下面的应用程序在界面布局中使用了 GestureOverlayView。

程序清单：codes\08\8.4\AddGesture\app\src\main\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="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/gestureTip"/>
<!-- 使用手势绘制组件 -->
<android.gesture.GestureOverlayView
    android:id="@+id/gesture"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gestureStrokeType="multiple" />
</LinearLayout>
```

由于 GestureOverlayView 并不是标准的视图组件，因此在界面布局中使用该组件时需要使用全限定类名。

上面的程序中使用 GestureOverlayView 组件时指定了一个 android:gestureStrokeType 参数，该参数控制手势是否需要多一笔完成。大部分时候，一个手势只要一笔就可以完成，此时可将该参数设为 single。如果该手势需要多笔来完成，则将该参数设为 multiple。

接下来程序将会为 GestureOverlayView 添加一个 OnGesturePerformedListener 监听器，当手势事件完成时，该监听器会打开一个对话框，让用户选择保存该手势。程序代码如下。

程序清单：codes\08\8.4\AddGesture\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText editText;
    GestureOverlayView gestureView;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取文本编辑框
        editText = (EditText) findViewById(R.id.gesture_name);
        // 获取手势编辑视图
        gestureView = (GestureOverlayView)
            findViewById(R.id.gesture);
        // 设置手势的绘制颜色
        gestureView.setGestureColor(Color.RED);
        // 设置手势的绘制宽度
        gestureView.setGestureStrokeWidth(4);
        // 为 gesture 的手势完成事件绑定事件监听器
        gestureView.addOnGesturePerformedListener(
            new OnGesturePerformedListener()
            {
                @Override
```

```
            public void onGesturePerformed(GestureOverlayView overlay,
                final Gesture gesture)
            {
                // 加载 save.xml 界面布局代表的视图
                View saveDialog = getLayoutInflater().inflate(
                    R.layout.save, null);
                // 获取 saveDialog 里的 show 组件
                ImageView imageView = (ImageView) saveDialog
                    .findViewById(R.id.show);
                // 获取 saveDialog 里的 gesture_name 组件
                final EditText gestureName = (EditText) saveDialog
                    .findViewById(R.id.gesture_name);
                // 根据 Gesture 包含的手势创建一个位图
                Bitmap bitmap = gesture.toBitmap(128,
                    128, 10, 0xffff0000);
                imageView.setImageBitmap(bitmap);
                // 使用对话框显示 saveDialog 组件
                new AlertDialog.Builder(MainActivity.this)
                .setView(saveDialog)
                .setPositiveButton("保存", new OnClickListener()
                {
                    @Override
                    public void onClick(DialogInterface dialog,
                        int which)
                    {
                        // 获取指定文件对应的手势库
                        GestureLibrary gestureLib = GestureLibraries
                            .fromFile("/mnt/sdcard/mygestures");
                        // 添加手势
                        gestureLib.addGesture(gestureName.getText()
                            .toString(), gesture);
                        // 保存手势库
                        gestureLib.save();
                    }
                }).setNegativeButton("取消", null).show();
            }
        });
    }
}
```

上面程序中的第一行粗体字代码用于为 GestureOverlayView 绑定 OnGesturePerformedListener 监听器，该监听器用于在手势完成时提供响应——它的响应就是打开一个对话框。该对话框的界面布局代码如下。

程序清单：codes\08\8.4\AddGesture\app\src\main\res\layout\save.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="8dip"
    android:text="@string/gesture_name" />
<!-- 定义一个文本框来让用户输入手势名 -->
<EditText
    android:id="@+id/gesture_name"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    </LinearLayout>
    <!-- 定义一个图片框来显示手势 -->
    <ImageView
        android:id="@+id/show"
        android:layout_width="128dp"
        android:layout_height="128dp"
        android:layout_marginTop="10dp" />
    </LinearLayout>
```

AddGesture 程序中的第二段粗体字代码是在对话框中完成的，这段粗体字代码用于从 SD 卡的指定文件中加载手势库，并添加用户刚刚输入的手势。

运行该程序，将看到如图 8.13 所示的界面。

用户可在图 8.13 所示的界面中随意地“绘制”手势，绘制完成后 OnGesturePerformedListener 监听器将会打开如图 8.14 所示的对话框。

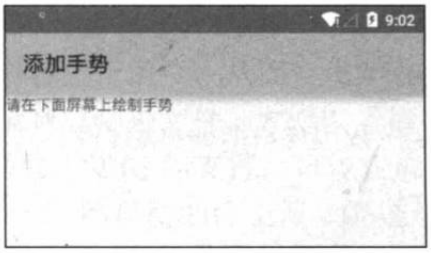


图 8.13 添加手势

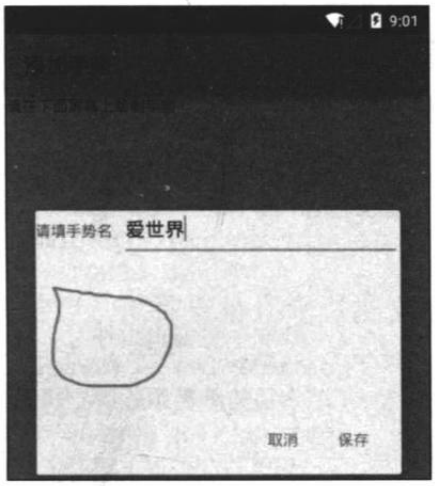


图 8.14 保存手势

当用户单击“保存”按钮后，程序将会调用 GestureLibrary 的 addGesture()方法来添加手势，并调用 save()方法来保存手势。

一旦用户通过该程序建立了自己的手势库，接下来就可在其他程序中使用该手势库了。

注意：

上面的程序需要将手势库保存在 SD 卡上，因此还需要授予该应用程序读写 SD 卡的权限。

8.4.3 识别用户手势

前面已经提到，GestureLibrary 提供了 recognize(Gesture ges)方法来识别手势，该方法将会返回该手势库中所有与 ges 匹配的手势——两个手势的图形越相似，相似度越高。

recognize(Gesture ges)方法的返回值为 ArrayList<Prediction>，其中 Prediction 封装了手势的匹配信息，Prediction 对象的 name 属性代表了匹配的手势名，score 属性代表了手势的相似度。

下面的程序将会利用前一个程序所创建的手势库来识别手势。该程序的界面很简单，只是在界面中定义了一个 GestureOverlayView 组件，允许用户在该组件上输入手势。程序为该组件绑定了 OnGesturePerformedListener 监听器，该监听器检测到用户手势完成时，就会调用手势库来识别用户输入的手势。

该程序的代码如下。

程序清单：codes\08\8.4\RecogniseGesture\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义手势编辑组件
    GestureOverlayView gestureView;
    // 记录手机上已有的手势库
    GestureLibrary gestureLibrary;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 读取上一个程序所创建的手势库
        gestureLibrary = GestureLibraries
            .fromFile("/mnt/sdcard/mygestures");
        if (gestureLibrary.load())
        {
            Toast.makeText(MainActivity.this, "手势文件装载成功！",
                Toast.LENGTH_SHORT).show();
        }
        else
        {
            Toast.makeText(MainActivity.this, "手势文件装载失败！",
                Toast.LENGTH_SHORT).show();
        }
        // 获取手势编辑组件
        gestureView = (GestureOverlayView) findViewById(R.id.gesture);
        // 为手势编辑组件绑定事件监听器
        gestureView.addOnGesturePerformedListener(
            new OnGesturePerformedListener()
            {
                @Override
                public void onGesturePerformed(GestureOverlayView
                    overlay, Gesture gesture)
                {
                    // 识别用户刚刚所绘制的手势
                    ArrayList<Prediction> predictions = gestureLibrary
                        .recognize(gesture);
                    ArrayList<String> result = new ArrayList<String>();
                    // 遍历所有找到的 Prediction 对象
                    for (Prediction pred : predictions)
                    {
                        // 只有相似度大于 2.0 的手势才会被输出
                        if (pred.score > 2.0)
                        {
                            result.add("与手势【" + pred.name + "】相似度为"
                                + pred.score);
                        }
                    }
                    if (result.size() > 0)
                    {
                        ArrayAdapter<Object> adapter = new
                            ArrayAdapter<Object>(MainActivity.this,
                            android.R.layout.simple_dropdown_item_1line
                            , result.toArray());
                        // 使用一个带 List 的对话框来显示所有匹配的手势
                        new AlertDialog.Builder(MainActivity.this)
                            .setAdapter(adapter, null)
                            .setPositiveButton("确定", null).show();
                    }
```

```
                else
                {
                    Toast.makeText(MainActivity.this
                        , "无法找到能匹配的手势！",
                        Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

上面程序中的粗体字代码就负责调用前一个程序的手势库来识别用户刚输入的手势，用户只要在屏幕上绘制一个大致与之前相似的手势，即可看到如图 8.15 所示的结果。

图 8.15 识别手势

注意：

该程序需要读取 SD 卡上的手势文件，因此该程序还需要授予读取 SD 卡的权限。

8.5 自动朗读（TTS）

Android 提供了自动朗读支持。自动朗读支持可以对指定文本内容进行朗读，从而发出声音；不仅如此，Android 的自动朗读支持还允许把文本对应的音频录制成音频文件，方便以后播放。这种自动朗读支持的英文名称为 TextToSpeech，简称 TTS。借助于 TTS 的支持，可以在应用程序中动态地增加音频输出，从而改善用户体验。

Android 的自动朗读支持主要通过 TextToSpeech 来完成，该类提供了如下一个构造器。

- TextToSpeech(Context context, TextToSpeech.OnInitListener listener)

从上面的构造器不难看出，当创建 TextToSpeech 对象时，必须先提供一个 OnInitListener 监听器——该监听器负责监听 TextToSpeech 的初始化结果。

一旦在程序中获得了 TextToSpeech 对象之后，接下来即可调用 TextToSpeech 的 setLanguage(Locale loc)方法来设置该 TTS 发声引擎应使用的语言、国家选项。

提示：

由于不同的文字，在不同的语言、国家中的发音是不同的，尤其是欧美，他们所使用的都是字母文字，因此一段文本内容，使用不同的语言、国家选项来朗读，发音效果是截然不同的。目前 Android 的 TTS 暂时不支持中文。

如果调用 setLanguage(Locale loc)的返回值是“TextToSpeech.LANG_COUNTRY_AVAILABLE”，则说明当前 TTS 系统可以支持所设置的语言、国家选项。

对 TextToSpeech 设置完成后，就可调用它的方法来朗读文本了，具体方法可参考 TextToSpeech 的 API 文档。TextToSpeech 类中最常用的方法是如下两个。

- speak(CharSequence text, int queueMode, Bundle params, String utteranceId)
- synthesizeToFile (CharSequence text, Bundle params, File file, String utteranceId)

上面两个方法都用于把 text 文字内容转换为音频，区别只是 speak()方法是播放转换的音频，而 synthesizeToFile()方法是把转换得到的音频保存成声音文件。

上面两个方法中的 params 都用于指定声音转换时的参数。speak()方法中的 queueMode 参数指定 TTS 的发音队列模式，该参数支持如下两个常量。

- TextToSpeech.QUEUE_FLUSH：如果指定该模式，当 TTS 调用 speak()方法时，它会中断当前实例正在运行的任务（也可以理解为清除当前语音任务，转而执行新的语音任务）。
- TextToSpeech.QUEUE_ADD：如果指定为该模式，当 TTS 调用 speak()方法时，会把新的发音任务添加到当前发音任务列队之后——也就是等任务队列中的发音任务执行完成后再来执行 speak()方法指定的发音任务。

当程序用完了 TextToSpeech 对象之后，可以在 Activity 的 OnDestroy()方法中调用它的 shutdown()来关闭 TextToSpeech，释放它所占用的资源。

归纳起来，使用 TextToSpeech 的步骤如下。

1. 创建 TextToSpeech 对象，创建时传入 OnInitListener 监听器监听创建是否成功。
2. 设置 TextToSpeech 所使用的语言、国家选项，通过返回值判断 TTS 是否支持该语言、国家选项。
3. 调用 speak()或 synthesizeToFile()方法。
4. 关闭 TTS，回收资源。

下面的程序示范了如何利用 TTS 来朗读用户所输入的文本内容。

程序清单：codes\08\8.5\Speech\app\src\main\java\org\crazyit\io\MainActivity.java

```
public class MainActivity extends Activity
{
    TextToSpeech tts;
    EditText editText;
    Button speech;
    Button record;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 初始化 TextToSpeech 对象
        tts = new TextToSpeech(this, new OnInitListener()
        {
            @Override
            public void onInit(int status)
            {
                // 如果装载 TTS 引擎成功
                if (status == TextToSpeech.SUCCESS)
                {
                    // 设置使用美式英语朗读
                    int result = tts.setLanguage(Locale.US);
                    // 如果不支持所设置的语言
                    if (result != TextToSpeech.LANG_COUNTRY_AVAILABLE
                        && result != TextToSpeech.LANG_AVAILABLE)
                    {
                        Toast.makeText(MainActivity.this
                            , "TTS 暂时不支持这种语言的朗读。"
                            , Toast.LENGTH_SHORT).show();
                    }
                }
            }
        });
        editText = (EditText) findViewById(R.id.txt);
        speech = (Button) findViewById(R.id.speech);
        record = (Button) findViewById(R.id.record);
        speech.setOnClickListener(new OnClickListener()
        {
```

```
        @Override
        public void onClick(View arg0)
        {
            // 执行朗读
            tts.speak(editText.getText().toString(),
                TextToSpeech.QUEUE_ADD, null , "speech");
        }
    });
    record.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View arg0)
        {
            // 将朗读文本的音频记录到指定文件
            tts.synthesizeToFile(editText.getText().toString() , null,
                new File("/mnt/sdcard/sound.wav"), "record");
            Toast.makeText(MainActivity.this, "声音记录成功！"
                , Toast.LENGTH_SHORT).show();
        }
    });
}
@Override
public void onDestroy()
{
    // 关闭 TextToSpeech 对象
    if (tts != null)
    {
        tts.shutdown();
    }
}
}
```

上面程序中的第一行粗体字代码创建了一个 TextToSpeech 对象，第二行粗体字代码设置使用美式英语进行朗读。接下来程序分别提供了两个按钮，一个按钮用于执行朗读，一个按钮用于将文本内容的朗读音频保存成声音文件，分别通过调用 TextToSpeech 对象的 speek()和 synthesizeToFile()方法来完成——如上面程序中的后两行粗体字代码所示。

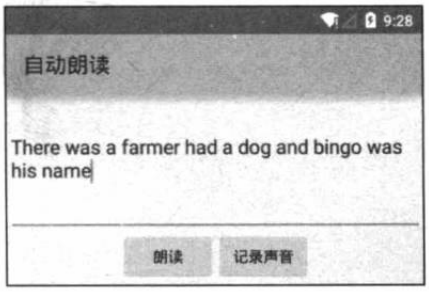


图 8.16 TTS

运行上面的程序，将可以看到如图 8.16 所示的界面。

在图 8.16 所示的界面中，当用户单击“朗读”按钮后，系统将会调用 TTS 的 speek()方法来朗读文本框中的内容；当用户单击“记录声音”按钮后，系统将会调用 synthesizeToFile()方法把文本框中的文本对应的朗读音频记录到 SD 卡的声音文件中——单击该按钮后将可以在 SD 卡的根目录下生成一个 sound.wav 文件，该文件可以被导出，在其他音频播放软件中播放。

程序重写 Activity 的 onDestroy()方法，并在该方法中关闭了 TextToSpeech 对象，回收了它的资源。

8.6 本章小结

本章主要介绍了 Android 的输入、输出支持，如果读者之前已有 Java IO 的编程经验，那么可以直接把 Java IO 的编程经验移植到 Android 上。当然 Android 为文件 IO 提供了 openFileOutput 和 openFileInput 两个便捷的方法，用起来十分方便；为记录、访问应用程序的

参数、选项提供了 SharedPreferences 工具类，这样可以非常方便地读写应用程序的参数、选项。除此之外，学习本章需要重点掌握的内容就是 SQLite 数据库，Android 系统内置了 SQLite 数据库，而且为访问 SQLite 数据库提供了大量方便的工具类，这需要读者掌握并能熟练地使用它们。

本章后面还介绍了 Android 提供的“另类”IO：手势支持和自动朗读。Android 的手势支持体现在两方面：手势检测与手势识别，前者属于事件处理方面，后者属于系统 IO 方面。熟练运用 Android 系统的手势支持可以开发出一些更新奇的应用。自动朗读则用于把文本转换成声音。

CHAPTER

9

第9章 使用 ContentProvider 实现数据共享

本章要点

- 理解 ContentProvider 的功能与意义
- ContentProvider 类的作用和常用方法
- Uri 对 ContentProvider 的作用
- 理解 ContentProvider 与 ContentResolver 的关系
- 实现自己的 ContentProvider
- 配置 ContentProvider
- 使用 ContentResolver 操作数据
- 操作系统 ContentProvider 提供的数据
- 监听 ContentProvider 的数据改变
- ContentObserver 类的作用和常用方法
- 监听系统 ContentProvider 的数据改变

当在系统中部署一个又一个 Android 应用之后，系统里将会包含多个 Android 应用，有时候就需要在不同的应用之间共享数据。比如现在有一个短信接收应用，用户想把接收到的陌生短信的发信人添加到联系人管理应用中，就需要在不同应用之间共享数据。对于这种需要在不同应用之间共享数据的需求，当然可以让一个应用程序直接去操作另一个应用程序所记录的数据，比如操作它所记录的 SharedPreferences、文件或数据库等。这种方式不仅比较麻烦，而且存在严重的安全漏洞，因此 Android 4.2 不再推荐使用这种方式，而是推荐使用本章介绍的 ContentProvider。

为了在应用程序之间交换数据，Android 提供了 ContentProvider，它是不同应用程序之间进行数据交换的标准 API，当一个应用程序需要把自己的数据暴露给其他程序使用时，该应用程序就可通过提供 ContentProvider 来实现；其他应用程序就可通过 ContentResolver 来操作 ContentProvider 暴露的数据。

ContentProvider 也是 Android 应用的四大组件之一，与 Activity、Service、BroadcastReceiver 相似，它们都需要在 AndroidManifest.xml 文件中进行配置。

一旦某个应用程序通过 ContentProvider 暴露了自己的数据操作接口，那么不管该应用程序是否启动，其他应用程序都可通过该接口来操作该应用程序的内部数据，包括增加数据、删除数据、修改数据、查询数据等。

9.1 数据共享标准：ContentProvider

ContentProvider 是不同应用程序之间进行数据交换的标准 API，ContentProvider 以某种 Uri 的形式对外提供数据，允许其他应用访问或修改数据；其他应用程序使用 ContentResolver 根据 Uri 去访问操作指定数据。

提示：

对于初学者而言，对于 ContentProvider、ContentResolver 两个核心 API 的作用可能需要花很长时间去理解。但这里有一个简单的类比，可以把 ContentProvider 当成 Android 系统内部的“网站”，这个网站以固定的 Uri 对外提供服务；而 ContentResolver 则可当成 Android 系统内部的 HttpClient，它可以向指定 Uri 发送“请求”（实际上是调用 ContentResolver 的方法），这种请求最后委托给 ContentProvider 处理，从而实现对“网站”（即 ContentProvider）内部数据进行操作。

9.1.1 ContentProvider 简介

如果把 ContentProvider 当成一个“网站”来看，那么如何对外提供数据呢？是否需要像 Java Web 开发一样编写 JSP、Servlet 之类呢？不需要。如果那样就太复杂了，毕竟 ContentProvider 只是提供数据的访问接口，并不是像一个网站一样对外提供完整的页面。

如果把 ContentProvider 当成一个“网站”来看，那么如何完整地开发一个 ContentProvider 呢？步骤其实很简单，如下所示。

① 定义自己的 ContentProvider 类，该类需要继承 Android 提供的 ContentProvider 基类。

② 向 Android 系统注册这个“网站”，也就是在 AndroidManifest.xml 文件中注册这个 ContentProvider，就像注册 Activity 一样。注册 ContentProvider 时需要为它绑定一个 Uri。

向 Android 系统中注册 ContentProvider 只要在<application.../>元素下添加如下子元素即可：

```
<!-- 下面配置中 name 属性指定 ContentProvider 类，
    authorities 就相当于为该 ContentProvider 指定域名 -->
<provider android:name=".DictProvider"
    android:authorities="org.crazyit.providers.dictprovider"
    android:exported="true"/>
```

提示：

虽然我们可以把 ContentProvider 当成一个“网站”来看，但 Android 官方文档并没有这种提法。Android 要求注册 ContentProvider 时指定 authorities 属性，该属性的值就相当于该网站的域名。但需要提醒读者：面试时千万不要这么说，因为你的面试官可能由于自身技术层次看不到这个高度，而并不认同这个说法，从而导致面试失败。

当我们通过上面配置文件注册了 DictProvider 之后，其他应用程序就可通过该 Uri 来访问 DictProvider 所暴露的数据了。

那么 DictProvider 到底如何暴露它所提供的数据呢？其实很简单，应用程序对数据的操作无非就是 CRUD 操作，因此 DictProvider 除了需要继承 ContentProvider 之外，还需要提供如下几个方法。

- public boolean onCreate()：该方法在 ContentProvider 创建后会被调用，当其他应用程序第一次访问 ContentProvider 时，该 ContentProvider 会被创建出来，并立即回调该 onCreate()方法。
- public Uri insert(Uri uri, ContentValues values)：根据该 Uri 插入 values 对应的数据。
- public int delete(Uri uri, String selection, String[] selectionArgs)：根据 Uri 删除 selection 条件所匹配的全部记录。
- public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)：根据 Uri 修改 selection 条件所匹配的全部记录。
- public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：根据 Uri 查询出 selection 条件所匹配的全部记录，其中 projection 就是一个列名列表，表明只选择出指定的数据列。
- public String getType(Uri uri)：该方法用于返回当前 Uri 所代表的数据的 MIME 类型。如果该 Uri 对应的数据可能包括多条记录，那么 MIME 类型字符串应该以 vnd.android.cursor.dir/开头；如果该 Uri 对应的数据只包含一条记录，那么 MIME 类型字符串应该以 vnd.android.cursor.item/开头。

通过介绍不难发现，对于 ContentProvider 而言，Uri 是一个非常重要的概念，下面详细介绍 Uri 的相关知识。

9.1.2 Uri 简介

在介绍 Android 系统的 Uri 之前，先来看一个最常用的互联网 URL。例如，想访问疯狂 Java 联盟的某个页面，应该在浏览器中输入如下 Uri：

```
http://www.crazyit.org:80/ethos.php
```

对于上面这个 URL，可分为如下三个部分。

- http://：URL 的协议部分，只要通过 HTTP 协议来访问网站，这个部分就是固定的。
- www.crazyit.org：域名部分。只要访问指定的网站，这个部分就是固定的。
- ethos.php：网站资源部分。当访问者需要访问不同资源时，这个部分是动态改变的。

ContentProvider 要求的 Uri 与此类似，例如如下 Uri：

```
content://org.crazyit.providers.dictprovider/words
```

它也可分为如下三个部分。

- content://：这个部分是 Android 的 ContentProvider 规定的，就像上网的协议默认是 http:// 一样。暴露 ContentProvider、访问 ContentProvider 的协议默认是 content://。
- org.crazyit.providers.dictprovider：这个部分就是 ContentProvider 的 authorities。系统就是由这个部分来找到操作哪个 ContentProvider 的。只要访问指定的 ContentProvider，这个部分就是固定的。
- words：资源部分（或者说数据部分）。当访问者需要访问不同资源时，这个部分是动态改变的。

需要指出的是，Android 的 Uri 所能表达的功能更丰富，它还可以支持如下 Uri：

```
content://org.crazyit.providers.dictprovider/word/2
```

此时它要访问的资源为 word/2，这意味着访问 word 数据中 ID 为 2 的记录。

还有如下形式：

```
content://org.crazyit.providers.dictprovider/word/2/word
```

此时它要访问的资源为 word/2/word，这意味着访问 word 数据中 ID 为 2 的记录的 word 字段。

如果想访问全部数据，即可使用下面所示的形式：

```
content://org.crazyit.providers.dictprovider/words
```

虽然大部分使用 ContentProvider 所操作的数据都来自于数据库，但有时候这些数据也可来自于文件、XML 或网络等其他存储方式，此时支持的 Uri 也可以改为如下形式：

```
content://org.crazyit.providers.dictprovider/word/detail/
```

上面的 Uri 表示操作 word 节点下的 detail 节点。

为了将一个字符串转换成 Uri，Uri 工具类提供了 parse()静态方法。例如，如下代码即可将字符串转换为 Uri：

```
Uri uri = Uri.parse("content://org.crazyit.providers.dictprovider/word/2")
```

9.1.3 使用 ContentResolver 操作数据

前面已经提到，ContentProvider 相当于一个“网站”，它的作用是暴露可供操作的数据；其他应用程序则通过 ContentResolver 来操作 ContentProvider 所暴露的数据，ContentResolver 相当于 HttpClient。

Context 提供了如下方法来获取 ContentResolver 对象。

- getContentResolver()：获取该应用默认的 ContentResolver。

一旦在程序中获得了 ContentResolver 对象之后，接下来就可调用 ContentResolver 的如下方法来操作数据了。

- insert(Uri url, ContentValues values)：向 Uri 对应的 ContentProvider 中插入 values 对应的数据。
- delete(Uri url, String where, String[] selectionArgs)：删除 Uri 对应的 ContentProvider 中 where 提交匹配的数据。
- update(Uri uri, ContentValues values, String where, String[] selectionArgs)：更新 Uri 对应的 ContentProvider 中 where 提交匹配的数据。

- query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：查询 Uri 对应的 ContentProvider 中 where 提交匹配的数据。

一般来说，ContentProvider 是单实例模式的，当多个应用程序通过 ContentResolver 来操作 ContentProvider 提供的数据时，ContentResolver 调用的数据操作将会委托给同一个 ContentProvider 处理。

9.2 开发 ContentProvider

对于许多初学者而言，理解 ContentProvider 暴露数据的方式是一个难点。ContentProvider 总是需要与 ContentResolver 结合使用，ContentProvider 负责暴露数据，因此 ContentProvider 需要与 ContentResolver 结合学习。

9.2.1 ContentProvider 与 ContentResolver 的关系

从 ContentResolver、ContentProvider 和 Uri 的关系来看，无论是 ContentResolver，还是 ContentProvider，它们所提供的 CRUD 方法的第一个参数都是 Uri。也就是说，Uri 是 ContentResolver 和 ContentProvider 进行数据交换的标识。ContentResolver 对指定 Uri 执行 CRUD 等数据操作，但 Uri 并不是真正的数据中心，因此这些 CRUD 操作会委托给该 Uri 对应的 ContentProvider 来实现。通常来说，假如 A 应用通过 ContentResolver 执行 CRUD 操作，这些 CRUD 操作都需要指定 Uri 参数，Android 系统就根据该 Uri 找到对应的 ContentProvider（该 ContentProvider 通常属于 B 应用），ContentProvider 则负责实现 CRUD 方法，完成对底层数据的增、删、改、查等操作，这样就可以让 A 应用访问、修改 B 应用的数据了。

ContentResolver、Uri、ContentProvider 三者之间的关系如图 9.1 所示。

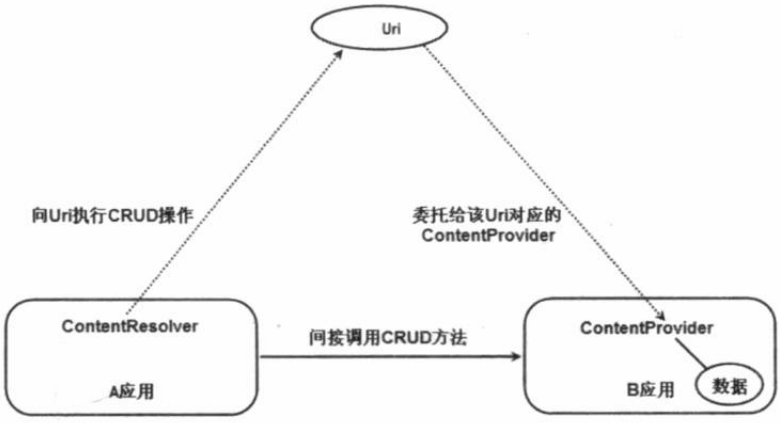


图 9.1 ContentRosolver.Uri 与 ContentProvider 的关系

从图 9.1 可以看出，以指定 Uri 为标识，ContentResolver 可以实现“间接调用”ContentProvider 的 CRUD 方法。

- 当 A 应用调用 ContentResolver 的 insert()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 insert()方法。
- 当 A 应用调用 ContentResolver 的 update()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 update()方法。
- 当 A 应用调用 ContentResolver 的 delete ()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 delete()方法。

- 当A应用调用ContentResolver的query ()方法时，实际上相当于调用了该Uri对应的ContentProvider（该ContentProvider属于B应用）的query()方法。

通过上面这种关系，即可实现让A应用访问、使用B应用底层的数据。

9.2.2 开发ContentProvider子类

开发ContentProvider只要如下两步。

① 开发一个ContentProvider子类，该子类需要实现query()、insert()、update()和delete()等方法。

② 在AndroidManifest.xml文件中注册该ContentProvider，指定android:authorities属性。

在上面两步中，ContentProvider子类实现的query()、insert()、update()和delete()方法，并不是给该应用本身调用的，而是供其他应用来调用的。正如前面提到的，当其他应用通过ContentResolver调用query()、insert()、update()和delete()方法执行数据访问时，实际上就是调用指定Uri对应的ContentProvider的query()、insert()、update()和delete()方法。

如何实现ContentProvider的query()、insert()、update()和delete()方法，完全由程序员决定——程序员想怎么暴露该应用数据，就怎么实现这4个方法。在极端情况下，开发者只对这些方法提供空实现也是可以的。

例如下面的示例ContentProvider，该ContentProvider虽然实现了query()、insert()、update()和delete()方法，但并未真正对底层数据进行访问，只是输出了一行字符串。下面是该ContentProvider的代码。

程序清单：codes\09\9.2\FirstProvider\app\src\main\java\org\crazyit\content\FirstProvider.java

```
public class FirstProvider extends ContentProvider
{
    // 第一次创建该 ContentProvider 时调用该方法
    @Override
    public boolean onCreate()
    {
        System.out.println("===onCreate 方法被调用===");
        return true;
    }
    // 该方法的返回值代表了该 ContentProvider 所提供数据的 MIME 类型
    @Override
    public String getType(Uri uri)
    {
        return null;
    }
    // 实现查询方法，该方法应该返回查询得到的 Cursor
    @Override
    public Cursor query(Uri uri, String[] projection, String where,
        String[] whereArgs, String sortOrder)
    {
        System.out.println(uri + "===query 方法被调用===");
        System.out.println("where 参数为：" + where);
        return null;
    }
    // 实现插入的方法，该方法应该返回新插入的记录的 Uri
    @Override
    public Uri insert(Uri uri, ContentValues values)
    {
        System.out.println(uri + "===insert 方法被调用===");
        System.out.println("values 参数为：" + values);
        return null;
    }
```

```
    // 实现更新方法，该方法应该返回被更新的记录条数
    @Override
    public int update(Uri uri, ContentValues values, String where,
        String[] whereArgs)
    {
        System.out.println(uri + "===update 方法被调用===");
        System.out.println("where 参数为："
            + where + ",values 参数为：" + values);
        return 0;
    }
    // 实现删除方法，该方法应该返回被删除的记录条数
    @Override
    public int delete(Uri uri, String where, String[] whereArgs)
    {
        System.out.println(uri + "===delete 方法被调用===");
        System.out.println("where 参数为：" + where);
        return 0;
    }
}
```

上面 4 个粗体字方法实现了 query()、insert()、update()和 delete()方法，这 4 个方法用于供其他应用通过 ContentRosovler 调用。

在该 ContentProvider 供其他应用调用之前，还需要先配置该 ContentProvider。

9.2.3 配置 ContentProvider

Android 应用要求所有应用程序组件（Activity、Service、ContentProvider、BroadcastReceiver）都必须显式进行配置。

只要为<application.../>元素添加了<provider.../>子元素即可配置 ContentProvider。例如如下的配置片段：

```
<provider
    android:name=".FirstProvider"
    android:authorities="org.crazyit.providers.firstprovider"
    android:exported="true" />
```

从上面的配置片段可以看出，配置 ContentProvider 时通常指定如下属性。

- name：指定该 ContentProvider 的实现类的类名。
- authorities：指定该 ContentProvider 对应的 Uri（相当于为该 ContentProvider 分配一个域名）。
- android:exported：指定该 ContentProvider 是否允许其他应用调用。如果将该属性设为 false，那么该 ContentProvider 将不允许其他应用调用。

为了配置上面的 ContentProvider，只要在<application.../>元素中添加如下子元素即可。

程序清单：codes\09\9.2\FirstProvider\app\src\main\AndroidManifest.xml

```
<application
    android:icon="@drawable/ic_launcher">
    <!-- 注册一个 ContentProvider -->
    <provider
        android:name=".FirstProvider"
        android:authorities="org.crazyit.providers.firstprovider"
        android:exported="true" />
</application>
```

上面的配置指定了该 ContentProvider 被绑定到“content://org.crazyit.providers.firstprovider”——这意味着当其他应用的 ContentResovler 向该 Uri 执行 query()、insert()、update()和 delete()方法

时，实际上是调用该 ContentProvider 的 query()、insert()、update()和 delete()方法。

- ContentResovler 调用方法时参数将会传给该 ContentProvider 的 query()、insert()、update()和 delete()方法。
- ContentResovler 调用方法的返回值，也就是 ContentProvider 执行 query()、insert()、update()和 delete()方法的返回值。

提示：

由于该应用并未提供 Activity，因此该应用没有任何界面。在 Android Studio 中运行该应用时，Android Studio 可能显示如图 9.2 所示的提示窗口，只要按图所示方式选择不加载任何 Activity 即可正常部署该应用。

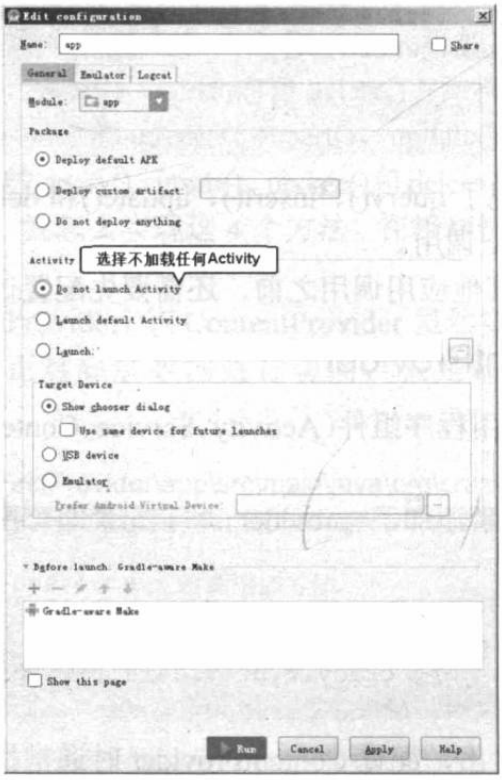


图 9.2 选择不加载任何 Activity

9.2.4 使用 ContentResolver 调用方法

Context 提供了 getContentResovler()方法，这表明 Activity、Service 等组件都可通过 getContentResovler()方法获取 ContentResolver 对象。

获取了 ContentResovler 对象之后，接下来就可调用 ContentResolver 的 query()、insert()、update()和 delete()方法了——实际上是指定 Uri 对应的 ContentProvider 的 query()、insert()、update()和 delete()。

下面示例的界面布局文件中只包含了 4 个按钮，分别用于激发调用 ContentProvider 的 query()、insert()、update()和 delete()方法。由于该示例的界面布局文件很简单，故此处不再给出代码。

下面是该示例的 Activity 代码。

程序清单：codes\09\9.2\FirstResolver\app\src\main\java\org\crazyit\resolver\MainActivity.java

```
public class MainActivity extends Activity
{
```

```
    ContentResolver contentResolver;
    Uri uri = Uri.parse("content://org.crazyit.providers.firstprovider/");
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 ContentResolver 对象
        contentResolver = getContentResolver();
    }
    public void query(View source)
    {
        // 调用 ContentResolver 的 query()方法
        // 实际返回的是该 Uri 对应的 ContentProvider 的 query()的返回值
        Cursor c = contentResolver.query(uri, null
            , "query_where", null, null);
        Toast.makeText(this, "远程 ContentProvider 返回的 Cursor 为: " + c,
            Toast.LENGTH_SHORT).show();
    }
    public void insert(View source)
    {
        ContentValues values = new ContentValues();
        values.put("name", "fkjava");
        // 调用 ContentResolver 的 insert()方法
        // 实际返回的是该 Uri 对应的 ContentProvider 的 insert()的返回值
        Uri newUri = contentResolver.insert(uri, values);
        Toast.makeText(this, "远程 ContentProvider 新插入记录的 Uri 为: "
            + newUri, Toast.LENGTH_SHORT).show();
    }
    public void update(View source)
    {
        ContentValues values = new ContentValues();
        values.put("name", "fkjava");
        // 调用 ContentResolver 的 update()方法
        // 实际返回的是该 Uri 对应的 ContentProvider 的 update()的返回值
        int count = contentResolver.update(uri, values
            , "update_where", null);
        Toast.makeText(this, "远程 ContentProvider 更新记录数为: "
            + count, Toast.LENGTH_SHORT).show();
    }
    public void delete(View source)
    {
        // 调用 ContentResolver 的 delete()方法
        // 实际返回的是该 Uri 对应的 ContentProvider 的 delete()的返回值
        int count = contentResolver.delete(uri
            , "delete_where", null);
        Toast.makeText(this, "远程 ContentProvider 删除记录数为: "
            + count, Toast.LENGTH_SHORT).show();
    }
}
```

上面的 4 行粗体字代码通过 ContentResolver 调用 query()、insert()、update()和 delete()方法，实际上就是调用 uri 参数对应的 ContentProvider 的 query()、insert()、update()和 delete()方法，也就是前面的 FirstProvider 的 query()、insert()、update()和 delete()方法。

运行上面的程序，并依次单击该应用的 4 个按钮，将可以看到 LogCat 生成如图 9.3 所示的输出。

从图 9.3 所示的输出可以看出，当 Android 应用通过 ContentResolver 调用 query()、insert()、update()和 delete()方法时，实际上就是调用 FirstProvider（该 Uri 对应的 ContentProvider）的 query()、insert()、update()和 delete()方法。

当用户单击程序界面上的“插入”按钮时，将可以在程序界面中看到如图 9.4 所示的输出。

图 9.3 ContentProvider 的方法被调用

图 9.4 通过 ContentResolver 调用远程 ContentProvider 的方法

从图 9.4 可以看出，调用 ContentResovler 的 insert()方法的返回值为 null，这是因为 FirstProvider 实现 insert()方法时返回了 null。

➤➤ 9.2.5 创建 ContentProvider 的说明

通过上面的介绍可以看出，ContentProvider 不像 Activity 存在复杂的生命周期，ContentProvider 只有一个 onCreate()生命周期方法——当其他应用通过 ContentResolver 第一次访问该 ContentProvider 时，onCreate()方法将会被回调，onCreate()方法只会被调用一次；ContentProvider 提供的 query()、insert()、update()和 delete()方法则由其他应用通过 ContentResolver 调用。

开发 ContentProvider 时所实现的 query()、insert()、update()和 delete()方法的第一个参数为 Uri，该参数由 ContentResolver 调用这些方法时传入。

前面介绍的示例由于并未真正对数据进行操作，因此 ContenProvider 并未对 Uri 参数进行任何判断。实际上，为了确定该 ContentProvider 实际能处理的 Uri，以及确定每个方法中 Uri 参数所操作的数据，Android 系统提供了 UriMatcher 工具类。

UriMatcher 工具类主要提供了如下两个方法。

- void addURI(String authority, String path, int code)：该方法用于向 UriMatcher 对象注册 Uri。其中 authority 和 path 组合成一个 Uri，而 code 则代表该 Uri 对应的标识码。
- int match(Uri uri)：根据前面注册的 Uri 来判断指定 Uri 对应的标识码。如果找不到匹配的标识码，该方法将会返回-1。

例如，我们先通过如下代码来创建 UriMatcher 对象：

```
UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
matcher.addURI("org.crazyit.providers.dictprovider", "words", 1);
matcher.addURI("org.crazyit.providers.dictprovider", "word/#", 2);
```

上面创建的 UriMatcher 对象注册了两个 Uri，其中 org.crazyit.providers.dictprovider/words 对应的标识码为 1；org.crazyit.providers.dictprovider/word/#对应的标识码为 2，#为通配符。

这就意味着如下匹配结果：

```
matcher.match(Uri.parse("content://org.crazyit.providers.dictprovider/words"));
// 返回标识码 1
matcher.match(Uri.parse("content://org.crazyit.providers.dictprovider/word/2"));
```

```
// 返回标识码 2
matcher.match(Uri.parse("content://org.crazyit.providers.dictprovider/word/10"));
// 返回标识码 2
```

至于到底需要为 UriMatcher 对象注册多少个 Uri，则取决于系统的业务需求。

对于 content://org.crazyit.providers.dictprovider/words 这个 Uri，它的资源部分为 words，这种资源通常代表了访问所有数据项；对于 content://org.crazyit.providers.dictprovider/word/2 这个 Uri，它的资源部分通常代表访问指定数据项，其中最后一个数值往往代表了该数据的 ID。

除此之外，Android 还提供了一个 ContentUris 工具类，它是一个操作 Uri 字符串的工具类，提供了如下两个工具方法。

- withAppendedId(uri, id)：用于为路径加上 ID 部分。例如：

```
Uri uri = Uri.parse("content://org.crazyit.providers.dictprovider/word")
Uri resultUri = ContentUris.withAppendedId(uri, 2);
// 生成后的 Uri 为："content://org.crazyit.providers.dictprovider/word/2"
```

- parseId(uri)：用于从指定 Uri 中解析出所包含的 ID 值。例如：

```
Uri uri = Uri.parse("content://org.crazyit.providers.dictprovider/word/2")
long wordId = ContentUris.parseId(uri); // 获取的结果为 2
```

掌握上面的知识之后，接下来我们将对前面介绍的生词本应用进行改进，增加 ContentProvider，这样即可允许其他应用程序通过 ContentResolver 来操作生词本应用的数据。

实例：使用 ContentProvider 共享生词本数据

正如前面访问系统 ContentProvider 所见到的，系统一般都会把 ContentProvider 的 Uri、数据列等信息以常量的形式公开出来，以方便访问。为此，我们也为该 ContentProvider 定义一个工具类，该类中只是包含一些 public static 的常量。该工具类的代码如下。

程序清单：codes\09\9.2\DictProvider\app\src\main\java\org\crazyit\content\Words.java

```
public final class Words
{
    // 定义该 ContentProvider 的 Authorities
    public static final String AUTHORITY
        = "org.crazyit.providers.dictprovider";
    // 定义一个静态内部类，定义该 ContentProvider 所包含的数据列的列名
    public static final class Word implements BaseColumns
    {
        // 定义 Content 所允许操作的三个数据列
        public final static String _ID = "_id";
        public final static String WORD = "word";
        public final static String DETAIL = "detail";
        // 定义该 Content 提供服务的两个 Uri
        public final static Uri DICT_CONTENT_URI = Uri
            .parse("content://" + AUTHORITY + "/words");
        public final static Uri WORD_CONTENT_URI = Uri
            .parse("content://"    + AUTHORITY + "/word");
    }
}
```

上面的工具类只是定义了一些简单的常量，这个工具类的作用就是告诉其他应用程序，访问该 ContentProvider 的一些常用入口。

提示：

实际上我们也可以不提供 Words 工具类，而是在 ContentProvider 中直接使用字符串来定义提供服务的 Uri；接下来再采用文档去告诉其他应用程序访问该 ContentProvider 的入口。但这种方式的可维护性并不好，因此在实际项目中（包括 Android 系统的 ContentProvider）都会通过采用工具类来定义各种常量的方式进行处理。

本实例只是在前面介绍的生词本应用的基础上进行修改，因此本应用可以直接使用前面所开发的 SQLiteOpenHelper 类。

接下来我们开发一个 ContentProvider 子类，并重写其中的增、删、改、查等方法。类代码如下。

程序清单：codes\09\9.2\DictProvider\app\src\main\java\org\crazyit\content\DictProvider.java

```
public class DictProvider extends ContentProvider
{
    private static UriMatcher matcher
        = new UriMatcher(UriMatcher.NO_MATCH);
    private static final int WORDS = 1;
    private static final int WORD = 2;
    private MyDatabaseHelper dbOpenHelper;
    static
    {
        // 为UriMatcher注册两个Uri
        matcher.addURI(Words.AUTHORITY, "words", WORDS);
        matcher.addURI(Words.AUTHORITY, "word/#", WORD);
    }
    // 第一次调用该DictProvider时，系统先创建DictProvider对象，并回调该方法
    @Override
    public boolean onCreate()
    {
        dbOpenHelper = new MyDatabaseHelper(this.getContext(),
            "myDict.db3", 1);
        return true;
    }
    // 返回指定Uri参数对应的数据的MIME类型
    @Override
    public String getType(Uri uri)
    {
        switch (matcher.match(uri))
        {
            // 如果操作的数据是多项记录
            case WORDS:
                return "vnd.android.cursor.dir/org.crazyit.dict";
            // 如果操作的数据是单项记录
            case WORD:
                return "vnd.android.cursor.item/org.crazyit.dict";
            default:
                throw new IllegalArgumentException("未知Uri:" + uri);
        }
    }
    // 查询数据的方法
    @Override
    public Cursor query(Uri uri, String[] projection, String where,
        String[] whereArgs, String sortOrder)
    {
        SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
        switch (matcher.match(uri))
        {
```

```
        // 如果 Uri 参数代表操作全部数据项
        case WORDS:
            // 执行查询
            return db.query("dict", projection, where,
                whereArgs, null, null, sortOrder);
        // 如果 Uri 参数代表操作指定数据项
        case WORD:
            // 解析出想查询的记录 ID
            long id = ContentUris.parseId(uri);
            String whereClause = Words.Word._ID + "=" + id;
            // 如果原来的 where 子句存在，拼接 where 子句
            if (where != null && !"".equals(where))
            {
                whereClause = whereClause + " and " + where;
            }
            return db.query("dict", projection, whereClause, whereArgs,
                null, null, sortOrder);
        default:
            throw new IllegalArgumentException("未知 Uri:" + uri);
    }
}
// 插入数据的方法
@Override
public Uri insert(Uri uri, ContentValues values)
{
    // 获得数据库实例
    SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
    switch (matcher.match(uri))
    {
        // 如果 Uri 参数代表操作全部数据项
        case WORDS:
            // 插入数据，返回插入记录的 ID
            long rowId = db.insert("dict", Words.Word._ID, values);
            // 如果插入成功返回 uri
            if (rowId > 0)
            {
                // 在已有的 Uri 的后面追加 ID
                Uri wordUri = ContentUris.withAppendedId(uri, rowId);
                // 通知数据已经改变
                getContext().getContentResolver()
                    .notifyChange(wordUri, null);
                return wordUri;
            }
            break;
        default :
            throw new IllegalArgumentException("未知 Uri:" + uri);
    }
    return null;
}
// 修改数据的方法
@Override
public int update(Uri uri, ContentValues values, String where,
    String[] whereArgs)
{
    SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
    // 记录所修改的记录数
    int num = 0;
    switch (matcher.match(uri))
    {
        // 如果 Uri 参数代表操作全部数据项
        case WORDS:
            num = db.update("dict", values, where, whereArgs);
```

```
            break;
        // 如果 Uri 参数代表操作指定数据项
        case WORD:
            // 解析出想修改的记录 ID
            long id = ContentUris.parseId(uri);
            String whereClause = Words.Word._ID + "=" + id;
            // 如果原来的 where 子句存在，拼接 where 子句
            if (where != null && !where.equals(""))
            {
                whereClause = whereClause + " and " + where;
            }
            num = db.update("dict", values, whereClause, whereArgs);
            break;
        default:
            throw new IllegalArgumentException("未知 Uri:" + uri);
    }
    // 通知数据已经改变
    getContext().getContentResolver().notifyChange(uri, null);
    return num;
}
// 删除数据的方法
@Override
public int delete(Uri uri, String where, String[] whereArgs)
{
    SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
    // 记录所删除的记录数
    int num = 0;
    // 对 uri 进行匹配
    switch (matcher.match(uri))
    {
        // 如果 Uri 参数代表操作全部数据项
        case WORDS:
            num = db.delete("dict", where, whereArgs);
            break;
        // 如果 Uri 参数代表操作指定数据项
        case WORD:
            // 解析出所需要删除的记录 ID
            long id = ContentUris.parseId(uri);
            String whereClause = Words.Word._ID + "=" + id;
            // 如果原来的 where 子句存在，拼接 where 子句
            if (where != null && !where.equals(""))
            {
                whereClause = whereClause + " and " + where;
            }
            num = db.delete("dict", whereClause, whereArgs);
            break;
        default:
            throw new IllegalArgumentException("未知 Uri:" + uri);
    }
    // 通知数据已经改变
    getContext().getContentResolver().notifyChange(uri, null);
    return num;
}
}
```

上面的 DictProvider 类很简单，它除了继承系统的 ContentProvider 之外，还实现了操作数据的增、删、改、查等方法，这样该 ContentProvider 就开发完成了。DictProvider 真正调用了 SQLiteDatabase 对底层数据执行增、删、改、查，当其他应用通过 ContentResolver 来调用该 DictProvider 的 query()、insert()、update()和 delete()方法时，就可以真正访问、操作该 ContentProvider 所在应用的底层数据了。

接下来需要在 AndroidManifest.xml 文件中注册该 ContentProvider，这就需要在 AndroidManifest.xml 文件中增加如下配置片段：

```
<!-- 注册一个 ContentProvider -->
<provider android:name=".DictProvider"
    android:authorities="org.crazyit.providers.dictprovider"
    android:exported="true"/>
```

至此，暴露生词本数据的 ContentProvider 开发完成。为了测试该 ContentProvider 的开发是否成功，接下来再开发一个应用程序，该应用程序将会通过 ContentResolver 来操作生词本中的数据。

该程序同样提供了添加生词、查询生词的功能，只是该程序自己并不保存数据，而是访问前面 DictProvider 所共享的数据。下面是使用 ContentResolver 的类的代码。

程序清单：codes\09\9.2\DictResolver\app\src\main\java\org\crazyit\resolver\MainActivity.java

```
public class MainActivity extends Activity
{
    ContentResolver contentResolver;
    Button insert = null;
    Button search = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 ContentResolver 对象
        contentResolver = getContentResolver();
        insert = (Button) findViewById(R.id.insert);
        search = (Button) findViewById(R.id.search);
        // 为 insert 按钮的单击事件绑定事件监听器
        insert.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String word = ((EditText) findViewById(R.id.word))
                    .getText().toString();
                String detail = ((EditText) findViewById(R.id.detail))
                    .getText().toString();
                // 插入生词记录
                ContentValues values = new ContentValues();
                values.put(Words.Word.WORD, word);
                values.put(Words.Word.DETAIL, detail);
                contentResolver.insert(
                    Words.Word.DICT_CONTENT_URI, values);
                // 显示提示信息
                Toast.makeText(MainActivity.this, "添加生词成功！"
                    , Toast.LENGTH_SHORT).show();
            }
        });
        // 为 search 按钮的单击事件绑定事件监听器
        search.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String key = ((EditText) findViewById(R.id.key))
                    .getText().toString();
```

```
                // 执行查询
                Cursor cursor = contentResolver.query(
                    Words.Word.DICT_CONTENT_URI, null,
                    "word like ? or detail like ?", new String[] {
                    "%" + key + "%", "%" + key + "%" }, null);
                // 创建一个 Bundle 对象
                Bundle data = new Bundle();
                data.putSerializable("data", converCursorToList(cursor));
                // 创建一个 Intent
                Intent intent = new Intent(MainActivity.this,
                    ResultActivity.class);
                intent.putExtras(data);
                // 启动 Activity
                startActivity(intent);
            }
        });
    }
    private ArrayList<Map<String, String>> converCursorToList(Cursor cursor)
    {
        ArrayList<Map<String, String>> result = new ArrayList<>();
        // 遍历 Cursor 结果集
        while (cursor.moveToNext())
        {
            // 将结果集中的数据存入 ArrayList 中
            Map<String, String> map = new HashMap<>();
            // 取出查询记录中第 2 列、第 3 列的值
            map.put(Words.Word.WORD, cursor.getString(1));
            map.put(Words.Word.DETAIL, cursor.getString(2));
            result.add(map);
        }
        return result;
    }
}
```

上面程序中的第一行粗体字代码用于向 DictProvider 所共享的数据中添加记录；第二行粗体字代码则用于查询 DictProvider 所共享的数据。

运行该程序需要先部署前面介绍的 DictProvider，因为该程序所操作的数据实际上来自于 DictProvider。当用户通过该程序来添加生词时，实际上是添加到了前面所介绍的 DictProvider 应用中；当用户通过该程序来查询生词时，实际上是查询 DictProvider 应用中的生词。

9.3 操作系统的 ContentProvider

实际上，Android 系统本身提供了大量的 ContentProvider，例如联系人信息、系统的多媒体信息等，开发者自己开发的 Android 应用也可通过 ContentResolver 来调用系统 ContentProvider 提供的 query()、insert()、update()和 delete()方法，这样开发者即可通过这些 ContentProvider 来获取 Android 内部的数据了。

使用 ContentResolver 操作系统 ContentProvider 数据的步骤依然是两步。

① 调用 Context 的 getContentResolver()获取 ContentResolver 对象。

② 根据需要调用 ContentResolver 的 insert()、delete()、update()和 query 方法操作数据。

为了操作系统提供的 ContentResolver，需要了解该 ContentProvider 的 Uri，以及该 ContentProvider 所操作的数据列的列名，可以通过查阅 Android 官方文档来获取这些信息。

9.3.1 使用 ContentProvider 管理联系人

Android 系统提供了 Contacts 应用程序来管理联系人，而且 Android 系统还为联系人管理

提供了 ContentProvider，这就允许其他应用程序以 ContentResolver 来管理联系人数据。

Android 系统用于管理联系人的 ContentProvider 的几个 Uri 如下。

- ContactsContract.Contacts.CONTENT_URI：管理联系人的 Uri。
- ContactsContract.CommonDataKinds.Phone.CONTENT_URI：管理联系人的电话的 Uri。
- ContactsContract.CommonDataKinds.Email.CONTENT_URI：管理联系人的 E-mail 的 Uri。

了解了联系人管理 ContentProvider 的 Uri 之后，接下来就可在应用程序中通过 ContentResolver 去操作系统的联系人数据了。

下面示例程序中包含两个按钮，其中一个按钮用于查询系统的联系人数据，该按钮所绑定的事件监听器代码如下。

程序清单：codes\09\9.3\ContactProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

```
search.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View source)
    {
        // 定义两个 List 来封装系统的联系人信息、指定联系人的电话号码、E-mail 等详情
        final ArrayList<String> names = new ArrayList<>();
        final ArrayList<ArrayList<String>> details = new ArrayList<>();
        // 使用 ContentResolver 查找联系人数据
        Cursor cursor = getContentResolver().query(
            ContactsContract.Contacts.CONTENT_URI, null, null,
            null, null);
        // 遍历查询结果，获取系统中所有联系人
        while (cursor.moveToNext())
        {
            // 获取联系人 ID
            String contactId = cursor.getString(cursor
                .getColumnIndex(ContactsContract.Contacts._ID));
            // 获取联系人的名字
            String name = cursor.getString(cursor.getColumnIndex(
                ContactsContract.Contacts.DISPLAY_NAME));
            names.add(name);
            // 使用 ContentResolver 查找联系人的电话号码
            Cursor phones = getContentResolver().query(
                ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                null, ContactsContract.CommonDataKinds.Phone.CONTACT_ID
                + " = " + contactId, null, null);
            ArrayList<String> detail = new ArrayList<>();
            // 遍历查询结果，获取该联系人的多个电话号码
            while (phones.moveToNext())
            {
                // 获取查询结果中电话号码列中的数据
                String phoneNumber = phones.getString(phones
                    .getColumnIndex(ContactsContract
                    .CommonDataKinds.Phone.NUMBER));
                detail.add("电话号码：" + phoneNumber);
            }
            phones.close();
            // 使用 ContentResolver 查找联系人的 E-mail 地址
            Cursor emails = getContentResolver().query(
                ContactsContract.CommonDataKinds.Email.CONTENT_URI,
                null, ContactsContract.CommonDataKinds.Email
                .CONTACT_ID + " = " + contactId, null, null);
            // 遍历查询结果，获取该联系人的多个 E-mail 地址
            while (emails.moveToNext())
```

```
        {
            // 获取查询结果中E-mail地址列中的数据
            String emailAddress = emails.getString(emails
                .getColumnIndex(ContactsContract
                .CommonDataKinds.Email.DATA));
            detail.add("邮件地址：" + emailAddress);
        }
        emails.close();
        details.add(detail);
    }
    cursor.close();
    // 加载result.xml界面布局代表的视图
    View resultDialog = getLayoutInflater().inflate(
        R.layout.result, null);
    // 获取resultDialog中ID为list的ExpandableListView
    ExpandableListView list = (ExpandableListView) resultDialog
        .findViewById(R.id.list);
    // 创建一个ExpandableListAdapter对象
    ExpandableListAdapter adapter =
        new BaseExpandableListAdapter()
        {
            // 获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition,
                int childPosition)
            {
                return details.get(groupPosition).get(childPosition);
            }
            @Override
            public long getChildId(int groupPosition,
                int childPosition)
            {
                return childPosition;
            }
            @Override
            public int getChildrenCount(int groupPosition)
            {
                return details.get(groupPosition).size();
            }
            private TextView getTextView()
            {
                AbsListView.LayoutParams lp = new AbsListView
                    .LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT
                    , 64);
                TextView textView = new TextView(
                    MainActivity.this);
                textView.setLayoutParams(lp);
                textView.setGravity(Gravity.CENTER_VERTICAL
                    | Gravity.LEFT);
                textView.setPadding(36, 0, 0, 0);
                textView.setTextSize(20);
                return textView;
            }
            // 该方法决定每个子选项的外观
            @Override
            public View getChildView(int groupPosition,
                int childPosition, boolean isLastChild,
                View convertView, ViewGroup parent)
            {
                TextView textView = getTextView();
                textView.setText(getChild(groupPosition,
                    childPosition).toString());
                return textView;
            }
```

```
            // 获取指定组位置处的组数据
            @Override
            public Object getGroup(int groupPosition)
            {
                return names.get(groupPosition);
            }
            @Override
            public int getGroupCount()
            {
                return names.size();
            }
            @Override
            public long getGroupId(int groupPosition)
            {
                return groupPosition;
            }
            // 该方法决定每个组选项的外观
            @Override
            public View getGroupView(int groupPosition,
                boolean isExpanded, View convertView,
                ViewGroup parent)
            {
                TextView textView = getTextView();
                textView.setText(getGroup(groupPosition).toString());
                return textView;
            }
            @Override
            public boolean isChildSelectable(int groupPosition,
                int childPosition)
            {
                return true;
            }
            @Override
            public boolean hasStableIds()
            {
                return true;
            }
        };
        // 为ExpandableListView设置Adapter对象
        list.setAdapter(adapter);
        // 使用对话框来显示查询结果
        new AlertDialog.Builder(MainActivity.this)
            .setView(resultDialog).setPositiveButton("确定", null)
            .show();
    }
});
```

上面程序中的第一行粗体字代码使用 ContentResolver 向 ContactsContract.Contacts.CONTENT_URI 查询数据，将可以把系统中所有联系人信息查询出来；第二行粗体字代码使用 ContentResolver 向 ContactsContract.CommonDataKinds.Phone.CONTENT_ URI 查询数据，用于查询指定联系人的电话信息；第三行粗体字代码使用 ContentResolver 向 ContactsContract.CommonDataKinds.Email.CONTENT_URI 查询数据，用于查询指定联系人的 E-mail 信息。

上面的程序通过 ContentResolver 将联系人信息、电话信息、E-mail 信息查询出来之后，使用了一个 ExpandableListView 来显示所有联系人信息。

运行该程序，并单击“查询”按钮，程序将会使用 ExpandableListView 显示所有联系人信息，如图 9.5 所示。

图 9.5　查询联系人信息

需要指出的是，上面的应用程序需要读取、添加联系人信息，因此要记得在AndroidManifest.xml 文件中为该应用程序授权。也就是在该文件的根元素中添加如下元素：

```
<!-- 授予读联系人 ContentProvider 的权限 -->
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<!-- 授予写联系人 ContentProvider 的权限 -->
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
```

该程序界面上提供了三个文本框，用户可以在这三个文本框中输入联系人名字、电话号码、E-mail 地址，然后单击“添加”按钮。该按钮绑定事件监听器的代码如下。

程序清单：codes\09\9.3\ContactProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

```
// 为 add 按钮的单击事件绑定监听器
add.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 获取程序界面中的三个文本框的内容
        String name = ((EditText) findViewById(R.id.name))
            .getText().toString();
        String phone = ((EditText) findViewById(R.id.phone))
            .getText().toString();
        String email = ((EditText) findViewById(R.id.email))
            .getText().toString();
        // 创建一个空的 ContentValues
        ContentValues values = new ContentValues();
        // 向 RawContacts.CONTENT_URI 执行一个空值插入
        // 目的是获取系统返回的 rawContactId
        Uri rawContactUri = getContentResolver().insert(
            ContactsContract.RawContacts.CONTENT_URI, values);
        long rawContactId = ContentUris.parseId(rawContactUri);
        values.clear();
        values.put(Data.RAW_CONTACT_ID, rawContactId);
        // 设置内容类型
        values.put(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE);
        // 设置联系人名字
        values.put(StructuredName.GIVEN_NAME, name);
        // 向联系人 URI 添加联系人名字
        getContentResolver().insert(android.provider.ContactsContract
            .Data.CONTENT_URI, values);
        values.clear();
        values.put(Data.RAW_CONTACT_ID, rawContactId);
        values.put(Data.MIMETYPE, Phone.CONTENT_ITEM_TYPE);
        // 设置联系人的电话号码
        values.put(Phone.NUMBER, phone);
        // 设置电话类型
```

```
            values.put(Phone.TYPE, Phone.TYPE_MOBILE);
            // 向联系人电话号码 URI 添加电话号码
            getContentResolver().insert(android.provider.ContactsContract
                .Data.CONTENT_URI, values);
            values.clear();
            values.put(Data.RAW_CONTACT_ID, rawContactId);
            values.put(Data.MIMETYPE, Email.CONTENT_ITEM_TYPE);
            // 设置联系人的 E-mail 地址
            values.put(Email.DATA, email);
            // 设置该电子邮件的类型
            values.put(Email.TYPE, Email.TYPE_WORK);
            // 向联系人 E-mail URI 添加 E-mail 数据
            getContentResolver().insert(android.provider.ContactsContract
                .Data.CONTENT_URI, values);
            Toast.makeText(MainActivity.this, "联系人数据添加成功",
                Toast.LENGTH_SHORT).show();
        }
    });
```

上面程序中的第一行粗体字代码通过 ContentResolver 添加一个联系人记录；第二行粗体字代码通过 ContentResolver 为指定联系人添加一个电话号码；第三行粗体字代码通过 ContentResolver 为指定联系人添加一个 E-mail 地址。

如果用户在程序的三个文本框中分别输入联系人姓名、电话号码、E-mail 地址，并单击“添加”按钮，将可以看到该联系人信息被添加到系统中。启动 Android 系统的 Contacts 程序，可以看到如图 9.6 所示的界面。

单击图 9.6 所示界面中的“沙和尚”列表项，系统将打开该联系人的联系方式详情，如图 9.7 所示。

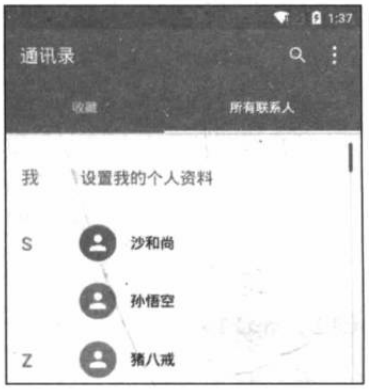


图 9.6 通过 ContentResolver 添加联系人

图 9.7 通过 ContentResolver 添加联系方式详情

从图 9.7 中可以看出，前面在程序中添加的 mobile 类型的电话号码、work 类型的 E-mail 地址都显示出来了，这就表明通过 ContentResolver 添加联系人成功。

提示：

可能有读者对 mobile 类型的电话号码、work 类型的 E-mail 地址感到迷惑，这是因为一个联系人可能有多个电话号码，比如移动电话（mobile 类型）、工作电话（work 类型）、家庭电话（home 类型）等，E-mail 地址也与此类似。实际上，好像只有早期手机上每个联系人只能添加一个电话，现在的手机应该都支持为一个联系人关联多个电话号码。

9.3.2 使用 ContentProvider 管理多媒体内容

Android 提供了 Camera 程序来支持拍照、拍摄视频，用户拍摄的照片、视频都将存放在固定的位置。有些时候，其他应用程序可能需要直接访问 Camera 所拍摄的照片、视频等，为了处理这种需求，Android 同样为这些多媒体内容提供了 ContentProvider。

Android 为多媒体提供的 ContentProvider 的 Uri 如下。

- MediaStore.Audio.Media.EXTERNAL_CONTENT_URI：存储在外部存储器（SD 卡）上的音频文件内容的 ContentProvider 的 Uri。
- MediaStore.Audio.Media.INTERNAL_CONTENT_URI：存储在手机内部存储器上的音频文件内容的 ContentProvider 的 Uri。
- MediaStore.Images.Media.EXTERNAL_CONTENT_URI：存储在外部存储器（SD 卡）上的图片文件内容的 ContentProvider 的 Uri。
- MediaStore.Images.Media.INTERNAL_CONTENT_URI：存储在手机内部存储器上的图片文件内容的 ContentProvider 的 Uri。
- MediaStore.Video.Media.EXTERNAL_CONTENT_URI：存储在外部存储器（SD 卡）上的视频文件内容的 ContentProvider 的 Uri。
- MediaStore.Video.Media.INTERNAL_CONTENT_URI：存储在手机内部存储器上的视频文件内容的 ContentProvider 的 Uri。

下面示例程序界面中包含两个按钮，一个“查看”按钮，用于查看多媒体数据中的所有图片；一个“添加”按钮，用于向多媒体数据中添加图片。

该程序中“查看”按钮所绑定的事件监听器的代码如下。

程序清单：codes\09\9.3\MediaProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

```
// 为view按钮的单击事件绑定监听器
view.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 清空names、descs、fileNames集合里原有的数据
        names.clear();
        descs.clear();
        fileNames.clear();
        // 通过ContentResolver查询所有图片信息
        Cursor cursor = getContentResolver().query(
            Media.EXTERNAL_CONTENT_URI, null, null, null, null);
        while (cursor.moveToNext())
        {
            // 获取图片的显示名
            String name = cursor.getString(cursor
                .getColumnIndex(Media.DISPLAY_NAME));
            // 获取图片的详细描述
            String desc = cursor.getString(cursor
                .getColumnIndex(Media.DESCRIPTION));
            // 获取图片的保存位置的数据
            byte[] data = cursor.getBlob(cursor
                .getColumnIndex(Media.DATA));
            // 将图片名添加到names集合中
            names.add(name);
            // 将图片描述添加到descs集合中
            descs.add(desc);
            // 将图片保存路径添加到fileNames集合中
```

```
            fileNames.add(new String(data, 0, data.length - 1));
        }
        // 创建一个 List 集合，List 集合的元素是 Map
        List<Map<String, Object>> listItems = new ArrayList<>();
        // 将 names、descs 两个集合对象的数据转换到 Map 集合中
        for (int i = 0; i < names.size(); i++)
        {
            Map<String, Object> listItem = new HashMap<>();
            listItem.put("name", names.get(i));
            listItem.put("desc", descs.get(i));
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(
            MainActivity.this, listItems ,
            R.layout.line, new String[] { "name", "desc" }
            , new int[] {R.id.name, R.id.desc });
        // 为 show ListView 组件设置 Adapter
        show.setAdapter(simpleAdapter);
    }
});
```

该程序中粗体字代码使用 ContentResolver 向 MediaStore.Audio.Images.EXTERNAL_CONTENT_URI 查询数据，这将查询出所有位于外部存储器上的图片信息。查询出图片信息之后，该程序使用了一个 ListView 来显示这些图片信息。

本应用需要读取外部存储设备中的多媒体信息，因此必须为该应用增加读取外部存储设备的权限。而且该应用还需要向外部存储设备写入图片，因此也需要增加写入外部存储设备的权限。也就是在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 授予读取外部存储设备的访问权限 -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<!-- 授予写入外部存储设备的访问权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

当用户单击“查看”按钮之后，系统将会显示如图 9.8 所示的列表。

注意图 9.8 所示列表中的列表项，该程序为该列表的列表项单击事件绑定了事件监听器，代码如下。

图 9.8 查看系统图片列表

程序清单：codes\09\9.3\MediaProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

```
// 为 show ListView 的列表项单击事件添加监听器
show.setOnItemClickListener(new OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> parent
        , View source, int position, long id)
    {
        // 加载 view.xml 界面布局代表的视图
        View viewDialog = getLayoutInflater().inflate(
            R.layout.view, null);
        // 获取 viewDialog 中 ID 为 image 的组件
        ImageView image = (ImageView) viewDialog
            .findViewById(R.id.image);
```

```
            // 设置 image 显示指定图片
            image.setImageBitmap(BitmapFactory.decodeFile(
                fileNames.get(position)));
            // 使用对话框显示用户单击的图片
            new AlertDialog.Builder(MainActivity.this)
                .setView(viewDialog).setPositiveButton("确定", null)
                .show();
        }
    });
```

正如上面的程序中粗体字代码所示，当用户单击指定列表项之后，程序会使用一个包含ImageView的对话框来显示该列表项所对应的图片。当用户单击指定列表项之后，系统将显示如图9.9所示的对话框。

图9.9　查看图片

提示：

通过该示例程序，可以非常方便地实现Android图片浏览器：Android图片浏览器并不需要程序员去遍历SD卡上的图片文件，直接查询系统多媒体ContentProvider即可获取系统中所有的图片信息。

该程序中的“添加”按钮用于将本程序中的指定图片添加到多媒体数据中——实际上完全可以把任意图片添加到多媒体数据中。程序为“添加”按钮绑定事件监听器的代码如下。

程序清单：codes\09\9.3\MediaProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

```
// 为 add 按钮的单击事件绑定监听器
add.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 创建 ContentValues 对象，准备插入数据
        ContentValues values = new ContentValues();
        values.put(Media.DISPLAY_NAME, "jinta");
        values.put(Media.DESCRIPTION, "金塔");
        values.put(Media.MIME_TYPE, "image/jpeg");
        // 插入数据，返回所插入数据对应的 Uri
        Uri uri = getContentResolver().insert(
            Media.EXTERNAL_CONTENT_URI, values);
        // 加载应用程序下的 jinta 图片
        Bitmap bitmap = BitmapFactory.decodeResource(
            MainActivity.this.getResources(),
```

```
                R.drawable.jinta);
        OutputStream os = null;
        try
        {
            // 获取刚插入的数据的 Uri 对应的输出流
            os = getContentResolver().openOutputStream(uri); // ①
            // 将 bitmap 图片保存到 Uri 对应的数据节点中
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, os);
            os.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
});
```

上面程序中的粗体字代码使用了 ContentResolver 向 MediaStore.Audio.Images.EXTERNAL_CONTENT_URI 插入一条记录，但此时还未把真正的图片数据插入进去。程序中①号粗体字代码打开了刚插入的图片的 Uri 对应的 OutputStream，接下来程序调用 Bitmap 的 compress 方法把实际的图片内容保存到 OutputStream 中，这样就会把实际图片内容存入系统。

9.4 监听 ContentProvider 的数据改变

前面介绍的是当 ContentProvider 将数据共享出来之后，ContentResolver 会根据业务需要去主动查询 ContentProvider 所共享的数据。有些时候，应用程序需要实时监听 ContentProvider 所共享数据的改变，并随着 ContentProvider 的数据的改变而提供响应，这就需要利用 ContentObserver 了。

9.4.1 ContentObserver 简介

前面介绍开发 ContentProvider 时，不管实现 insert、delete、update 方法中的哪一个，只要该方法导致 ContentProvider 数据的改变，程序就调用如下代码：

```
getContext().getContentResolver().notifyChange(uri, null);
```

这行代码可用于通知所有注册在该 Uri 上的监听者：该 ContentProvider 所共享的数据发生了改变。

为了在应用程序中监听 ContentProvider 数据的改变，需要利用 Android 提供的 ContentObserver 基类。监听 ContentProvider 数据改变的监听器需要继承 ContentObserver 类，并重写该基类所定义的 onChange(boolean selfChange)方法——当它所监听的 ContentProvider 数据发生改变时，该 onChange()方法将会被触发。

为了监听指定 ContentProvider 的数据变化，需要通过 ContentResolver 向指定 Uri 注册 ContentObserver 监听器。ContentResolver 提供了如下方法来注册监听器。

- registerContentObserver (Uri uri, boolean notifyForDescendents, ContentObserver observer)

该方法中的三个参数说明如下。

- uri：该监听器所监听的 ContentProvider 的 Uri。
- notifyForDescendents：如果该参数设为 true，假如注册监听的 Uri 为 content://abc，那么 Uri 为 content://abc/xyz、content://abc/xyz/foo 的数据改变时也会触发该监听器；

如果该参数设为 false，假如注册监听的 Uri 为 content://abc，那么只有 content://abc 的数据发生改变时才会触发该监听器。

- observer：监听器实例。

例如，如下代码片段可用于为指定 Uri 注册监听器：

```
getContentResolver().registerContentObserver(Uri.parse("content://sms")
    , true, new SmsObserver(new Handler()));
```

上面的代码中 SmsObserver 就是 ContentObserver 的子类。

实例：监听用户发出的短信

本实例通过监听 Uri 为 content://sms 的数据改变即可监听到用户短信的数据改变，并在监听器的 onChange(boolean selfChange)方法里查询 Uri 为 content://sms/outbox 的数据，这样即可获取用户正在发送的短信（用户正在发送的短信保存在发件箱内）。

该程序代码如下。

程序清单：codes\09\9.4\MonitorSms\app\src\main\java\org\crazyit\content\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 为 content://sms 的数据改变注册监听器
        getContentResolver().registerContentObserver(
            Uri.parse("content://sms"), true,
            new SmsObserver(new Handler()));
    }
    // 提供自定义的 ContentObserver 监听器类
    private final class SmsObserver extends ContentObserver
    {
        public SmsObserver(Handler handler)
        {
            super(handler);
        }
        public void onChange(boolean selfChange)
        {
            // 查询发件箱中的短信（处于正在发送状态的短信放在发件箱中）
            Cursor cursor = getContentResolver().query(
                Uri.parse("content://sms/outbox")
                , null, null, null, null);
            // 遍历查询得到的结果集，即可获取用户正在发送的短信
            while (cursor.moveToNext())
            {
                StringBuilder sb = new StringBuilder();
                // 获取短信的发送地址
                sb.append("address=").append(cursor
                    .getString(cursor.getColumnIndex("address")));
                // 获取短信的标题
                sb.append(";subject=").append(cursor
                    .getString(cursor.getColumnIndex("subject")));
                // 获取短信的内容
                sb.append(";body=").append(cursor
                    .getString(cursor.getColumnIndex("body")));
```

```
                // 获取短信的发送时间
                sb.append(";time=").append(cursor
                    .getLong(cursor.getColumnIndex("date")));
                System.out.println("发送短信：" + sb.toString());
            }
        }
    }
}
```

上面程序中的第一行粗体字代码用于监听 Uri 为 content://sms 的数据改变，就可以监听到用户短信数据的改变；第二行粗体字代码用于查询 content://sms/outbox 的全部数据，也就是查询发件箱（正在发送的短信保存在发件箱内）内的全部短信，这样就可获取用户正在发送的短信详情。

运行该程序，在不关闭该程序的情况下打开 Android 系统内置的“Messaging”程序发送短信——直接向本机号码发送即可。当用户发送短信时，可以在 LogCat 面板看到如图 9.10 所示的输出。

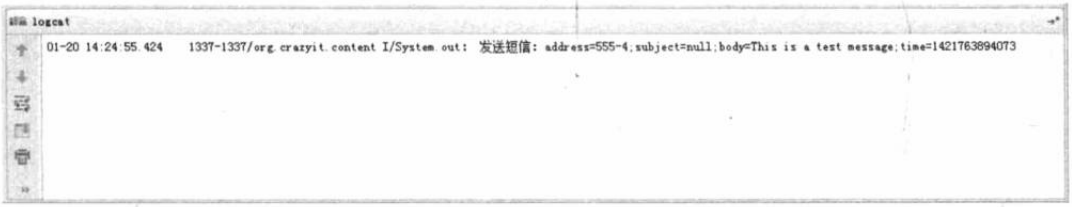


图 9.10 监听用户发送的短信

本程序需要读取系统短信的内容，因此还需要为该应用增加读取短信的权限，也就是需要在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 授予本应用读取短信的权限 -->
<uses-permission android:name="android.permission.READ_SMS"/>
```

提示：

由于 Genymotion 模拟器并未提供模拟发送短信的功能，因此本实例需要使用 Android 自带的 AVD 模拟器进行测试。

这个监听用户发送短信的程序采用 Activity 来实现并不合适——因为用户必须先主动打开该 Activity，并在保持该 Activity 不关闭的情况下，用户所发送的短信才会被监听到。这显然不太符合实际需求场景，在实际情况下，可能更希望该程序以后台进程的方式“不知不觉”地运行，这就需要利用 Android 的 Service 组件，接下来的章节将介绍 Android 提供的 Service 组件。

9.5 本章小结

本章主要介绍了 Android 系统中 ContentProvider 组件的功能和用法，ContentProvider 的本质就像一个“小网站”，它可以把应用程序的数据按照“固定规范”暴露出来，其他应用程序就可通过 ContentProvider 暴露的接口来操作内部的数据了。可以这样说，ContentProvider 是 Android 系统内不同程序之间进行数据交换的标准接口。学习本章需要重点掌握三个 API 的用法：ContentResolver、ContentProvider 和 ContentObserver，其中 Content Resolver 用于操作 ContentProvider 提供的数据；ContentObserver 用于监听 ContentProvider 的数据改变；而 ContentProvider 则是所有 ContentProvider 组件的基类。

CHAPTER

10

第10章 Service与BroadcastReceiver

本章要点

- Service 组件的作用和意义
- 创建、配置 Service
- 启动、停止 Service
- 绑定本地 Service 并与之通信
- Service 的生命周期
- IntentService 的功能和用法
- TelephonyManager 的功能和用法
- 监听手机电话
- SmsManager 的功能和用法
- 监听手机短信
- AudioManager 的功能和用法
- Vibrator 的功能和用法
- AlarmManager 的功能和用法
- BroadcastReceiver 组件的作用和意义
- 开发、配置 BroadcastReceiver 组件
- 发送广播、发送有序广播
- 使用 BroadcastReceiver 接收系统广播

Service 是 Android 四大组件中与 Activity 最相似的组件，它们都代表可执行的程序，Service 与 Activity 的区别在于：Service 一直在后台运行，它没有用户界面，所以绝不会到前台来。一旦 Service 被启动起来之后，它就与 Activity 一样，它完全具有自己的生命周期。关于程序中 Activity 与 Service 的选择标准是：如果某个程序组件需要在运行时向用户呈现某种界面，或者该程序需要与用户交互，就需要使用 Activity；否则就应该考虑使用 Service 了。

开发者开发 Service 的步骤与开发 Activity 的步骤很像，开发 Service 组件需要先开发一个 Service 子类，然后在 AndroidManifest.xml 文件中配置该 Service，配置时可通过<intent-filter.../>元素指定它可被哪些 Intent 启动。

Android 系统本身提供了大量的 Service 组件，开发者可通过这些系统 Service 来操作 Android 系统本身。

本章还将向读者介绍 BroadcastReceiver 组件。BroadcastReceiver 组件就像一个全局的事件监听器，只不过它用于监听系统发出的 Broadcast。通过使用 BroadcastReceiver，即可在不同应用程序之间通信。

10.1 Service 简介

Service 组件也是可执行的程序，它也有自己的生命周期。创建、配置 Service 与创建、配置 Activity 的过程基本相似，下面详细介绍 Android Service 的开发。

10.1.1 创建、配置 Service

就像开发 Activity 需要两个步骤：① 开发 Activity 子类；② 在 AndroidManifest.xml 文件中配置 Activity。开发 Service 也需要两个步骤。

1. 定义一个继承 Service 的子类。
2. 在 AndroidManifest.xml 文件中配置该 Service。

Service 与 Activity 还有一点相似之处，它们都是从 Context 派生出来的，因此它们都可调用 Context 里定义的如 getResources()、getContentResolver()等方法。

与 Activity 相似的是，Service 中也定义了一系列生命周期方法，如下所示。

- IBinder onBind(Intent intent)：该方法是 Service 子类必须实现的方法。该方法返回一个 IBinder 对象，应用程序可通过该对象与 Service 组件通信。
- void onCreate()：在该 Service 第一次被创建后将立即回调该方法。
- void onDestroy()：在该 Service 被关闭之前将会回调该方法。
- void onStartCommand(Intent intent, int flags, int startId)：该方法的早期版本是 void onStart (Intent intent, int startId)，每次客户端调用 startService(Intent)方法启动该 Service 时都会回调该方法。
- boolean onUnbind(Intent intent)：当该 Service 上绑定的所有客户端都断开连接时将会回调该方法。

下面的类定义了一个 Service 组件。

程序清单：codes\10\10.1\FirstService\app\src\main\java\org\crazyit\service\FirstService.java

```
public class FirstService extends Service
{
    // 必须实现的方法
    @Override
    public IBinder onBind(Intent arg0)
```

```
    {
        return null;
    }
    // Service被创建时回调该方法
    @Override
    public void onCreate()
    {
        super.onCreate();
        System.out.println("Service is Created");
    }
    // Service被启动时回调该方法
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        System.out.println("Service is Started");
        return START_STICKY;
    }
    // Service被关闭之前回调该方法
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        System.out.println("Service is Destroyed");
    }
}
```

上面这个Service什么也没干——它只是重写了Service组件的onCreate()、onStartCommand()、onDestroy()、onBind()等方法，重写这些方法时只是简单地输出了一条字符串。

提示：

虽然这个Service什么都没干，但实际上它是Service组件的框架，如果希望Service组件做某些事情，那么只要在onCreate()或onStartCommand()方法中定义相关业务代码即可。

定义了上面的Service之后，接下来需要在AndroidManifest.xml文件中配置该Service，配置Service使用<service.../>元素。与配置Activity相似的是，配置Service时也可为<service.../>元素配置<intent-filter.../>子元素，用于说明该Service可被哪些Intent启动。

在AndroidManifest.xml文件中增加如下配置片段来配置该Service：

```
<!-- 配置一个Service组件 -->
<service android:name=".FirstService">
</service>
```

从上面的配置片段不难看出，配置Service与配置Activity的差别并不大，只是配置Service使用<service.../>元素，而且无须指定android:label属性——因为Service没有界面，总是位于后台运行，为该Service指定标签没有太大的意义。

当该Service开发完成之后，接下来就可在程序中运行该Service了。在Android系统中运行Service有如下两种方式。

- **通过Context的startService()方法：**通过该方法启动Service，访问者与Service之间没有关联，即使访问者退出了，Service也仍然运行。
- **通过Context的bindService()方法：**使用该方法启动Service，访问者与Service绑定在一起，访问者一旦退出，Service也就终止了。

下面先示范第一种方式运行Service。

10.1.2 启动和停止 Service

下面的程序使用 Activity 作为 Service 的访问者，该 Activity 的界面中包含两个按钮，一个按钮用于启动 Service，一个按钮用于关闭 Service。

该 Activity 的代码如下。

程序清单：codes\10\10.1\FirstService\app\src\main\java\org\crazyit\service\MainActivity.java

```
public class MainActivity extends Activity
{
    Button start, stop;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的 start、stop 两个按钮
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        // 创建启动 Service 的 Intent
        final Intent intent = new Intent(this , FirstService.class);
        start.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 启动指定 Service
                startService(intent);
            }
        });
        stop.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 停止指定 Service
                stopService(intent);
            }
        });
    }
}
```

从上面程序中的粗体字代码不难看出，启动、关闭 Service 十分简单，调用 Context 里定义的 startService()、stopService()方法即可启动、关闭 Service。

注意：

可能有读者会注意到本程序使用了显式 Intent（直接指定要启动的目标组件的实现类）来启动 Service，但本书第 2 版使用的却是隐式 Intent 启动 Service。这是因为从 Android 5.0 开始，Google 要求必须使用显式 Intent 启动 Service 组件。

运行该程序，通过程序界面先启动 Service，再关闭 Service，将可以在 Android Studio 的 logcat 面板看到如图 10.1 所示的输出。

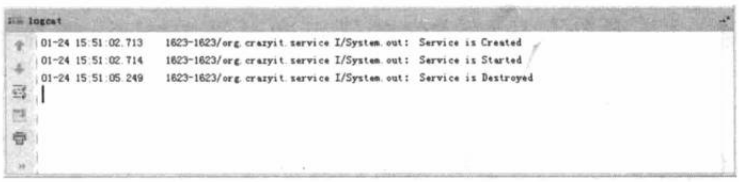


图 10.1 启动、关闭 Service

如果在不关闭 Service 的情况下，连续三次单击“启动 Service”按钮，

程序将会连续三次启动 Service，此时在 Android Studio 的 logcat 面板可以看到如图 10.2 所示的输出。

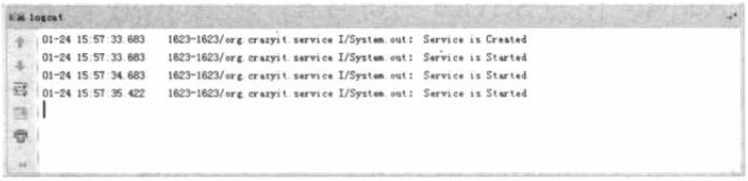


图 10.2　连续启动 Service

从图 10.2 可以看出，每当 Service 被创建时会回调 onCreate()方法，每次 Service 被启动时都会回调 onStartCommand()方法——多次启动一个已有的 Service 组件将不会再回调 onCreate()方法，但每次启动时都会回调 onStartCommand()方法。

10.1.3　绑定本地 Service 并与之通信

当程序通过 startService()和 stopService()启动、关闭 Service 时，Service 与访问者之间基本上不存在太多的关联，因此 Service 和访问者之间也无法进行通信、交换数据。

如果 Service 和访问者之间需要进行方法调用或交换数据，则应该使用 bindService()和 unbindService()方法启动、关闭 Service。

Context 的 bindService()方法的完整方法签名为：bindService(Intent service, ServiceConnection conn, int flags)，该方法的三个参数解释如下。

- service：该参数通过 Intent 指定要启动的 Service。
- conn：该参数是一个 ServiceConnection 对象，该对象用于监听访问者与 Service 之间的连接情况。当访问者与 Service 之间连接成功时将回调该 ServiceConnection 对象的 onServiceConnected(ComponentName name, IBinder service)方法；当 Service 所在的宿主进程由于异常中止或其他原因终止，导致该 Service 与访问者之间断开连接时回调该 ServiceConnection 对象的 onServiceDisconnected(ComponentName name)方法。

注意：

当调用者主动通过 unBindService()方法断开与 Service 的连接时，ServiceConnection 对象的 onServiceDisconnected (ComponentName name)方法并不会被调用。

- flags：指定绑定时是否自动创建 Service（如果 Service 还未创建）。该参数可指定为 0（不自动创建）或 BIND_AUTO_CREATE（自动创建）。

注意到 ServiceConnection 对象的 onServiceConnected()方法中有一个 IBinder 对象，该对象即可实现与被绑定 Service 之间的通信。

当开发 Service 类时，该 Service 类必须提供一个 IBinder onBind(Intent intent)方法，在绑定本地 Service 的情况下，onBind(Intent intent)方法所返回的 IBinder 对象将会传给 ServiceConnection 对象里 onServiceConnected(ComponentName name, IBinder service)方法的 service 参数，这样访问者就可通过该 IBinder 对象与 Service 进行通信了。

提示：

IBinder 对象相当于 Service 组件的内部钩子，该钩子关联到绑定的 Service 组件，当其他程序组件绑定该 Service 时，Service 将会把 IBinder 对象返回给其他程序组件，其他程序组件通过该 IBinder 对象即可与 Service 组件进行实时通信。

实际上开发时通常会采用继承 Binder（IBinder 的实现类）的方式实现自己的 IBinder 对象。

下面的程序示范了如何在 Acticity 中绑定本地 Service，并获取 Service 的运行状态。该程序的 Service 类需要“真正”实现 onBind()方法，并让该方法返回一个有效的 IBinder 对象。该 Service 类的代码如下。

程序清单：codes\10\10.1\BindService\app\src\main\java\org\crazyit\service\BindService.java

```
public class BindService extends Service
{
    private int count;
    private boolean quit;
    // 定义 onBinder 方法所返回的对象
    private MyBinder binder = new MyBinder();
    // 通过继承 Binder 来实现 IBinder 类
    public class MyBinder extends Binder  // ①
    {
        public int getCount()
        {
            // 获取 Service 的运行状态：count
            return count;
        }
    }
    // 必须实现的方法，绑定该 Service 时回调该方法
    @Override
    public IBinder onBind(Intent intent)
    {
        System.out.println("Service is Binded");
        // 返回 IBinder 对象
        return binder;
    }
    // Service 被创建时回调该方法
    @Override
    public void onCreate()
    {
        super.onCreate();
        System.out.println("Service is Created");
        // 启动一条线程，动态地修改 count 状态值
        new Thread()
        {
            @Override
            public void run()
            {
                while (!quit)
                {
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e)
                    {
                    }
                    count++;
                }
            }
        }.start();
    }
    // Service 被断开连接时回调该方法
    @Override
    public boolean onUnbind(Intent intent)
    {
        System.out.println("Service is Unbinded");
        return true;
```

```
    }
    // Service 被关闭之前回调该方法
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        this.quit = true;
        System.out.println("Service is Destroyed");
    }
}
```

上面 Service 类的粗体字代码实现了 onBind()方法，该方法返回了一个可访问该 Service 状态数据（count 值）的 IBinder 对象，该对象将被传给该 Service 的访问者。

上面程序中的①号代码通过继承 Binder 类实现了一个 IBinder 对象，这个 MyBinder 类是 Service 的内部类，这对于绑定本地 Service 并与之通信的场景是一种常见的情形。

接下来定义一个 Activity 来绑定该 Service，并在该 Activity 中通过 MyBinder 对象访问 Service 的内部状态。该 Activity 的界面上包含三个按钮，第一个按钮用于绑定 Service；第二个按钮用于解除绑定；第三个按钮则用于获取 Service 的运行状态。该 Activity 的代码如下。

程序清单：codes\10\10.1\BindService\app\src\main\java\org\crazyit\service\MainActivity.java

```
public class MainActivity extends Activity
{
    Button bind, unbind, getServiceStatus;
    // 保持所启动的 Service 的 IBinder 对象
    BindService.MyBinder binder;
    // 定义一个 ServiceConnection 对象
    private ServiceConnection conn = new ServiceConnection()
    {
        // 当该 Activity 与 Service 连接成功时回调该方法
        @Override
        public void onServiceConnected(ComponentName name
            , IBinder service)
        {
            System.out.println("--Service Connected--");
            // 获取 Service 的 onBind 方法所返回的 MyBinder 对象
            binder = (BindService.MyBinder) service; // ①
        }
        // 当该 Activity 与 Service 断开连接时回调该方法
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
            System.out.println("--Service Disconnected--");
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的 start、stop、getServiceStatus 按钮
        bind = (Button) findViewById(R.id.bind);
        unbind = (Button) findViewById(R.id.unbind);
        getServiceStatus = (Button) findViewById(R.id.getServiceStatus);
        // 创建启动 Service 的 Intent
        final Intent intent = new Intent(this, BindService.class);
        bind.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
```

```
            // 绑定指定 Service
            bindService(intent, conn, Service.BIND_AUTO_CREATE);
        }
    });
    unbind.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View source)
        {
            // 解除绑定 Service
            unbindService(conn);
        }
    });
    getServiceStatus.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View source)
        {
            // 获取并显示 Service 的 count 值
            Toast.makeText(MainActivity.this,
                "Service 的 count 值为: " + binder.getCount(),
                Toast.LENGTH_SHORT).show(); // ②
        }
    });
  }
}
```

上面程序中的①号粗体字代码用于在该 Activity 与 Service 连接成功时获取 Service 的 onBind()方法所返回的 MyBinder 对象；②号粗体字代码即可通过 MyBinder 对象来访问 Service 的运行状态了。

运行该程序，单击程序界面中的“绑定 Service”按钮，即可看到 Android Studio 的 logcat 有如图 10.3 所示的输出。

在该 Activity 中绑定 Service 之后，该 Activity 还可通过 MyBinder 对象来获取 Service 的运行状态，如果用户单击程序界面上的“获取 Service 状态”按钮，即可看到如图 10.4 所示的输出。

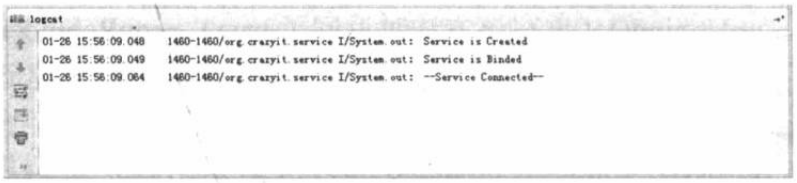


图 10.3 绑定 Service

图 10.4 访问 Service 的运行状态

从图 10.4 所示的输出可以看到，该 Activity 可以非常方便地访问到 Service 的运行状态。虽然本程序只是一个简单的示例，该 Activity 只是访问了 Service 的一个简单 count 值，但实际上完全可以让 MyBinder 去操作 Service 中更多的数据——到底需要访问 Service 的多少数据，完全取决于实际业务的需要。

提示：

对于 Service 的 onBind()方法所返回的 IBinder 对象来说，它可被当成该 Service 组件所返回的代理对象，Service 允许客户端通过该 IBinder 对象来访问 Service 内部的数据，这样即可实现客户端与 Service 之间的通信。

如果我们单击程序界面上的“解除绑定”按钮，即可在 Android Studio 的 logcat 中看到如图 10.5 所示的输出。

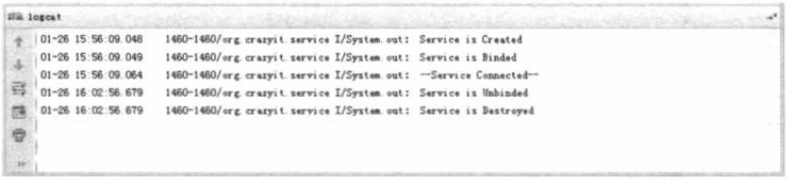


图 10.5 解除绑定

正如图 10.5 中所示，当程序调用 unbindService()方法解除对某个 Service 的绑定时，系统会先回调该 Service 的 onUnbind()方法，然后再回调 onDestroy()方法。

与多次调用 startService()方法启动 Service 不同的是，多次调用 bindService()方法并不会执行重复绑定。对于前一个示例程序，用户每单击“启动 Service”按钮一次，系统就会回调 Service 的 onStartCommand()方法一次；对于这个示例程序，不管用户单击 “绑定 Service”按钮多少次，系统只会回调 Service 的 onBind()方法一次。

➤➤ 10.1.4 Service 的生命周期

通过前面两个示例，读者应该已经大致明白 Service 的生命周期了。随着应用程序启动 Service 方式的不同，Service 的生命周期也略有差异。

如果应用程序通过 startService()方法来启动 Service，Service 的生命周期如图 10.6 左边所示。

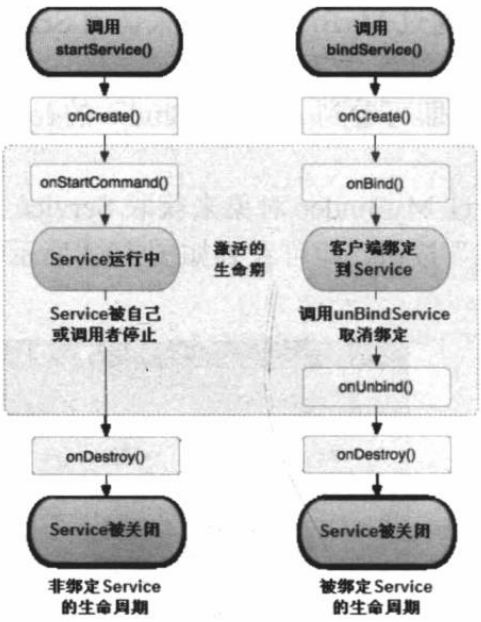


图 10.6 Service 的生命周期

如果应用程序通过 bindService()方法来启动 Service，Service 的生命周期如图 10.6 右边所示。

Service 生命周期还有一种特殊的情形——如果 Service 已由某个客户端通过 startService()方法启动了，接下来其他客户端调用 bindService()方法绑定该 Service 后，再调用 unbindService()方法解除绑定，最后又调用 bindService()方法再次绑定到 Service，这个过程所触发的生命周期方法如下。

onCreate() → onStartCommand() → onBind() → onUnbind()（重写该方法时返回了 true）→onRebind()

在上面这个触发过程中，onCreate()是创建该 Service 后立即调用的，只有当该 Service 被创建时才会被调用；onStartCommand ()方法则是由客户端调用 startService()方法时触发的。图 10.7 所示的 logcat 显示了上面生命周期的输出。

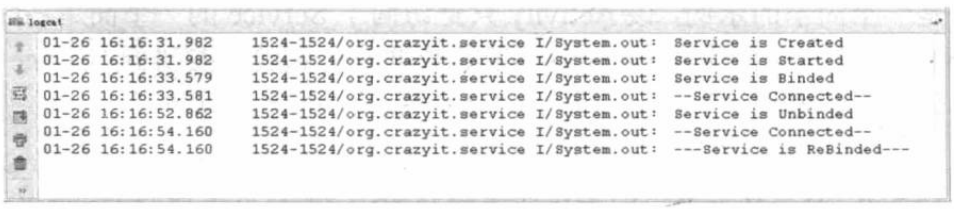


图 10.7 先启动、再绑定的 Service 生命周期

在图 10.7 所示的输出中，可以看到 Service 的 onRebind()方法被回调了。如果希望该方法被回调，除了需要该 Service 是由 Activity 的 startService()方法启动之外，还需要 Service 子类

重写 onUnbind()方法时返回 true。

注意：

在图 10.7 所示的输出中，并没有发现 Service 回调 onDestroy()方法，这是因为该 Service 并不是由 Activity 通过 bindService()方法启动的（该 Service 事先已由 Activity 通过 startService()方法启动了），因此当 Activity 调用 unBindService()方法取消与该 Service 的绑定时，该 Service 也不会终止。

由此可见，当 Activity 调用 bindService()绑定一个已启动的 Service 时，系统只是把 Service 内部 IBinder 对象传给 Activity，并不会把该 Service 生命周期完全“绑定”到该 Activity，因而当 Activity 调用 unBindService()方法取消与该 Service 的绑定时，也只是切断该 Activity 与 Service 之间的关联，并不能停止该 Service 组件。

10.1.5 使用 IntentService

IntentService 是 Service 的子类，因此它不是普通的 Service，它比普通的 Service 增加了额外的功能。

先看 Service 本身存在的两个问题。

- Service 不会专门启动一个单独的进程，Service 与它所在应用位于同一个进程中。
- Service 不是一条新的线程，因此不应该在 Service 中直接处理耗时的任务。

提示：

如果开发者需要在 Service 中处理耗时任务，建议在 Service 中另外启动一条新线程来处理该耗时任务。就像在前面 BindService 中所看到的，程序在 BindService 的 onCreate()方法中启动了一条新线程来处理耗时任务。可能有读者感到疑惑：直接在其他程序组件中启动子线程来处理耗时任务不行吗？这种方式也不可靠，由于 Activity 可能会被用户退出，而 BroadcastReceiver 的生命周期本身就很短。可能出现的情况是：在子线程还没有结束的情况下，Activity 已经被用户退出了，或者 BroadcastReceiver 已经结束了。在 Activity 已经退出、BroadcastReceiver 已经结束的情况下，此时它们所在的进程就变成了空进程（没有任何活动组件的进程），系统需要内存时可能会优先终止该进程。如果宿主进程被终止，那么该进程内的所有子线程也会被中止，这样就可能导致子线程无法执行完成。

而 IntentService 正好可以弥补 Service 的上述两个不足：IntentService 将会使用队列来管理请求 Intent，每当客户端代码通过 Intent 请求启动 IntentService 时，IntentService 会将该 Intent 加入队列中，然后开启一条新的 worker 线程来处理该 Intent。对于异步的 startService()请求，IntentService 会按次序依次处理队列中的 Intent，该线程保证同一时刻只处理一个 Intent。由于 IntentService 使用新的 worker 线程处理 Intent 请求，因此 IntentService 不会阻塞主线程，所以 IntentService 自己就可以处理耗时任务。

归纳起来，IntentService 具有如下特征。

- IntentService 会创建单独的 worker 线程来处理所有的 Intent 请求。
- IntentService 会创建单独的 worker 线程来处理 onHandleIntent()方法实现的代码，因此开发者无须处理多线程问题。
- 当所有请求处理完成后，IntentService 会自动停止，因此开发者无须调用 stopSelf()方

法来停止该 Service。

- 为 Service 的 onBind()方法提供了默认实现，默认实现的 onBind()方法返回 null。
- 为 Service 的 onStartCommand()方法提供了默认实现，该实现会将请求 Intent 添加到队列中。

从上面的介绍可以看出，扩展 IntentService 实现 Service 无须重写 onBind()、onStartCommand()方法，只要重写 onHandleIntent()方法即可。

下面的示例程序界面中包含了两个按钮，分别用于启动普通 Service 和 IntentService，两个 Service 都需要处理耗时任务。该程序的界面布局代码很简单，这里不再给出。主程序 Activity 代码如下。

程序清单：codes\10\10.1\IntentServiceTest\app\src\main\java\org\crazyit\service\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void startService(View source)
    {
        // 创建需要启动的 Service 的 Intent
        Intent intent = new Intent(this, MyService.class);
        // 启动 Service
        startService(intent);
    }
    public void startIntentService(View source)
    {
        // 创建需要启动的 IntentService 的 Intent
        Intent intent = new Intent(this, MyIntentService.class);
        // 启动 IntentService
        startService(intent);
    }
}
```

上面 Activity 的两个事件处理方法中分别启动了 MyService 和 MyIntentService，其中 MyService 是继承 Service 的子类，而 MyIntentService 则是继承 IntentService 的子类。

下面是 MyService 类的代码。

程序清单：codes\10\10.1\IntentServiceTest\app\src\main\java\org\crazyit\service\MyService.java

```
public class MyService extends Service
{
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        // 该方法内执行耗时任务可能导致 ANR (Application Not Responding) 异常
        long endTime = System.currentTimeMillis() + 20 * 1000;
        System.out.println("onStart");
        while (System.currentTimeMillis() < endTime)
        {
            synchronized (this)
            {
```

```
                try
                {
                    wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e)
                {
                }
            }
        }
        System.out.println("---耗时任务执行完成---");
        return START_STICKY;
    }
}
```

上面 MyService 在 onStartCommand()方法中使用线程暂停的方式模拟了耗时任务，该线程暂停了 20 秒，相当于该耗时任务需要执行 20 秒。由于普通 Service 的执行会阻塞主线程，因此启动该线程将会导致程序出现 ANR（Application Not Responding）异常。

下面是 MyIntentService 类的代码。

程序清单：codes\10\10.1\IntentServiceTest\app\src\main\java\org\crazyit\service\MyIntentService.java

```
public class MyIntentService extends IntentService
{
    public MyIntentService()
    {
        super("MyIntentService");
    }
    // IntentService 会使用单独的线程来执行该方法的代码
    @Override
    protected void onHandleIntent(Intent intent)
    {
        // 该方法内可以执行任何耗时任务，比如下载文件等，此处只是让线程暂停 20 秒
        long endTime = System.currentTimeMillis() + 20 * 1000;
        System.out.println("onStartCommand");
        while (System.currentTimeMillis() < endTime)
        {
            synchronized (this)
            {
                try
                {
                    wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e)
                {
                }
            }
        }
        System.out.println("---耗时任务执行完成---");
    }
}
```

从上面的代码可以看出，MyIntentService 继承了 IntentService，并不需要实现 onBind()、onStartCommand()方法，只要实现 onHandleIntent()方法即可，在该方法中定义该 Service 需要完成的任务。本示例的 onHandleIntent()方法也用线程暂停的方式模拟了耗时任务，线程同样暂停了 20 秒。但由于 IntentService 会使用单独的线程来完成该耗时任务，因此启动 MyIntentService 不会阻塞前台线程。

运行该示例，如果单击界面上的“启动普通 Service”按钮，将会激发 startService()方法，该方法将会启动 MyService 去执行耗时任务，此时将会导致程序 UI 线程被阻塞（程序界面失去响应），而且由于阻塞时间太长，因此将会看到如图 10.8 所示的 ANR 异常。

图 10.8　使用普通 Service 执行耗时任务导致 ANR 异常

相反，如果调用“启动 IntentService”来启动MyIntentService，虽然MyIntentService也需要执行耗时任务，但由于MyIntentService会使用单独的 worker 线程，因此 MyIntentService 不会阻塞前台的 UI 线程，所以程序界面不会失去响应。

10.2　电话管理器（TelephonyManager）

TelephonyManager 是一个管理手机通话状态、电话网络信息的服务类，该类提供了大量的 getXxx()方法来获取电话网络的相关信息。

在程序中获取 TelephonyManager 十分简单，只要调用如下代码即可：

```
TelephonyManager tManager =
 (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

接下来就可以通过 TelephonyManager 获取相关信息或者进行相关操作了。

实例：获取网络和 SIM 卡信息

通过 TelephonyManager 提供的一系列方法即可获取手机网络、SIM 卡的相关信息，该程序使用了一个 ListView 来显示网路和 SIM 卡的相关信息。

该程序代码如下。

程序清单：codes\10\10.2\TelephonyStatus\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    ListView showView;
    // 声明代表状态名的数组
    String[] statusNames;
    // 声明代表手机状态的集合
    ArrayList<String> statusValues = new ArrayList<>();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 TelephonyManager 对象
        TelephonyManager tManager = (TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE);
        // 获取各种状态名称的数组
        statusNames = getResources().getStringArray(R.array.statusNames);
        // 获取代表 SIM 卡状态的数组
        String[] simState = getResources()
            .getStringArray(R.array.simState);
        // 获取代表电话网络类型的数组
        String[] phoneType = getResources().getStringArray(
            R.array.phoneType);
        // 获取设备编号
```

```
        statusValues.add(tManager.getDeviceId());
        // 获取系统平台的版本
        statusValues.add(tManager.getDeviceSoftwareVersion()
            != null ? tManager.getDeviceSoftwareVersion() : "未知");
        // 获取网络运营商代号
        statusValues.add(tManager.getNetworkOperator());
        // 获取网络运营商名称
        statusValues.add(tManager.getNetworkOperatorName());
        // 获取手机网络类型
        statusValues.add(phoneType[tManager.getPhoneType()]);
        // 获取设备所在位置
        statusValues.add(tManager.getCellLocation() != null ? tManager
            .getCellLocation().toString() : "未知位置");
        // 获取 SIM 卡的国别
        statusValues.add(tManager.getSimCountryIso());
        // 获取 SIM 卡序列号
        statusValues.add(tManager.getSimSerialNumber());
        // 获取 SIM 卡状态
        statusValues.add(simState[tManager.getSimState()]);
        // 获得 ListView 对象
        showView = (ListView) findViewById(R.id.show);
        ArrayList<Map<String, String>> status = new ArrayList<>();
        // 遍历 statusValues 集合，将 statusNames、statusValues
        // 的数据封装到 List<Map<String , String>>集合中
        for (int i = 0; i < statusValues.size(); i++)
        {
            HashMap<String, String> map = new HashMap<>();
            map.put("name", statusNames[i]);
            map.put("value", statusValues.get(i));
            status.add(map);
        }
        // 使用 SimpleAdapter 封装 List 数据
        SimpleAdapter adapter = new SimpleAdapter(this, status,
            R.layout.line, new String[] { "name", "value" }
            , new int[] { R.id.name, R.id.value });
        // 为 ListView 设置 Adapter
        showView.setAdapter(adapter);
    }
}
```

由于该应用需要获取手机位置和手机状态，因此程序还需要在 AndroidManifest.xml 文件中增加如下配置片段：

```
<!-- 添加访问手机位置的权限 -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<!-- 添加访问手机状态的权限 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

运行该程序，将可以看到如图 10.9 所示的输出。

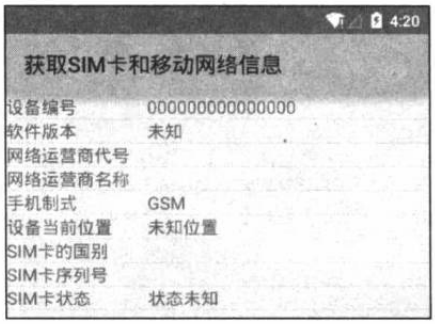


图 10.9 获取 SIM 卡信息和网络状态

TelephonyManager 除了提供一系列的 getXxx()方法来获取网络状态和 SIM 卡信息之外，还提供了一个 listen(PhoneStateListener listener, int events)方法来监听通话状态。下面通过该方法来监听手机来电信息。

实例：监听手机来电

本实例程序通过监听 TelephonyManager 的通话状态来监听手机的所有来电。该程序代码如下。

程序清单：codes\10\10.2\MonitorPhone\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    TelephonyManager tManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 取得 TelephonyManager 对象
        tManager = (TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE);
        // 创建一个通话状态监听器
        PhoneStateListener listener = new PhoneStateListener()
        {
            @Override
            public void onCallStateChanged(int state, String number)
            {
                switch (state)
                {
                    // 无任何状态
                    case TelephonyManager.CALL_STATE_IDLE:
                        break;
                    case TelephonyManager.CALL_STATE_OFFHOOK:
                        break;
                    // 来电铃响时
                    case TelephonyManager.CALL_STATE_RINGING:
                        OutputStream os = null;
                        try
                        {
                            os = openFileOutput("phoneList", MODE_APPEND);
                        }
                        catch (FileNotFoundException e)
                        {
                            e.printStackTrace();
                        }
                        PrintStream ps = new PrintStream(os);
                        // 将来电号码记录到文件中
                        ps.println(new Date() + " 来电：" + number);
                        ps.close();
                        break;
                    default:
                        break;
                }
                super.onCallStateChanged(state, number);
            }
        };
        // 监听电话通话状态的改变
        tManager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
    }
}
```

上面的程序中首先创建了一个 PhoneStateListener，它是一个通话状态监听器，该监听器可用于对 TelephonyManager 进行监听。当手机来电铃响时，程序将会把来电号码记录到文件中，如上面的粗体字代码所示。

运行上面的程序，在保证该程序运行的状态下，启动另一个模拟器呼叫该电话。接下来就可以在 Monitor 的 File Explorer 面板的 data/data/org.crazyit.manager/files 目录下看到一个 phoneList 文件，将该文件导出到计算机上，查看该文件内容即可看到如下信息：

```
Mon Jan 2 15:19:38 GMT 2015 来电：15555215556
```

从上面的文件内容可以看出，该程序记录了来自另一个模拟器的电话呼入信息。

如果我们把这段代码放在后台执行的 Service 中运行，并且设置 Service 组件随着系统开机自动运行，那么这种监听就可以“神不知鬼不觉”了。本章后面会介绍如何让 Service 随系统开机自动运行。

需要指出的是，由于该程序需要获取手机的通话状态，因此必须在 AndroidManifest.xml 文件中增加如下权限配置代码：

```
<!-- 授予该应用读取通话状态的权限 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

10.3 短信管理器（SmsManager）

SmsManager 是 Android 提供的另一个非常常见的服务，SmsManager 提供了一系列 sendXxxMessage()方法用于发送短信，不过就现在实际应用来看，短信通常都是普通的文本内容，也就是调用 sendTextMessage()方法进行发送即可。

实例：发送短信

本实例程序十分简单，程序提供一个文本框让用户输入收件人号码，一个文本框让用户输入短信内容，接下来单击“发送”按钮即可将短信发送出去。

程序代码如下。

程序清单：codes\10\10.3\SendSms\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText number, content;
    Button send;
    SmsManager sManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 SmsManager
        sManager = SmsManager.getDefault();
        // 获取程序界面上的两个文本框和按钮
        number = (EditText) findViewById(R.id.number);
        content = (EditText) findViewById(R.id.content);
        send = (Button) findViewById(R.id.send);
        // 为 send 按钮的单击事件绑定监听器
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
```

```
                // 创建一个 PendingIntent 对象
                PendingIntent pi = PendingIntent.getActivity(
                    MainActivity.this, 0, new Intent(), 0);
                // 发送短信
                sManager.sendTextMessage(number.getText().toString(),
                    null, content.getText().toString(), pi, null);
                // 提示短信发送完成
                Toast.makeText(MainActivity.this, "短信发送完成", 8000).show();
            }
        });
    }
}
```

从上面程序的粗体字代码可以看出，使用 SmsManager 发送短信十分简单，简单地调用 sendTextMessage()方法即可发送。

上面的程序中用到了一个 PendingIntent 对象，PendingIntent 是对 Intent 的包装，一般通过调用 PendingIntent 的 getActivity()、getService()、getBroadcastReceiver()静态方法来获取 PendingIntent 对象。与 Intent 对象不同的是，PendingIntent 通常会传给其他应用组件，从而由其他应用程序来执行 PendingIntent 所包装的“Intent”。

该程序需要调用 SmsManager 来发送短信，因此还需要授予该程序发送短信的权限，也就是在 AndroidManifest.xml 文件中增加如下代码：

```
<!-- 授予发送短信的权限 -->
<uses-permission android:name="android.permission.SEND_SMS"/>
```

实例：短信群发

短信群发也是一个十分实用的功能，逢年过节，很多人都喜欢通过短信群发向自己的朋友表示祝福。短信群发可以将一条短信同时向多个人发送。短信群发的实现十分简单，只要让程序遍历每个收件人号码并依次向每个收件人发送短信即可。

该程序提供了一个带列表框的对话框供用户选择收件人号码，代码如下。

程序清单：codes\10\10.3\GroupSend\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText numbers, content;
    Button select, send;
    SmsManager sManager;
    // 记录需要群发的号码列表
    ArrayList<String> sendList = new ArrayList<String>();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        sManager = SmsManager.getDefault();
        // 获取界面上的文本框、按钮组件
        numbers = (EditText) findViewById(R.id.numbers);
        content = (EditText) findViewById(R.id.content);
        select = (Button) findViewById(R.id.select);
        send = (Button) findViewById(R.id.send);
        // 为 send 按钮的单击事件绑定监听器
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
```

```
        for (String number : sendList)
        {
            // 创建一个 PendingIntent 对象
            PendingIntent pi = PendingIntent.getActivity(
                MainActivity.this, 0, new Intent(), 0);
            // 发送短信
            sManager.sendTextMessage(number, null, content
                .getText().toString(), pi, null);
        }
        // 提示短信群发完成
        Toast.makeText(MainActivity.this, "短信群发完成"
            , Toast.LENGTH_SHORT).show();
    }
});
// 为 select 按钮的单击事件绑定监听器
select.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 查询联系人的电话号码
        final Cursor cursor = getContentResolver().query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null, null, null, null);
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                return cursor.getCount();
            }
            @Override
            public Object getItem(int position)
            {
                return position;
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            @Override
            public View getView(int position, View convertView,
                ViewGroup parent)
            {
                cursor.moveToPosition(position);
                CheckBox rb = new CheckBox(MainActivity.this);
                // 获取联系人的电话号码，并去掉中间的中画线、空格
                String number = cursor
                    .getString(cursor.getColumnIndex(ContactsContract
                    .CommonDataKinds.Phone.NUMBER))
                    .replace("-", "")
                    .replace(" " , "");
                rb.setText(number);
                // 如果该号码已经被加入发送人名单，默认勾选该号码
                if (isChecked(number))
                {
                    rb.setChecked(true);
                }
                return rb;
            }
        };
        // 加载 list.xml 布局文件对应的 View
        View selectView = getLayoutInflater().inflate(
```

```
                R.layout.list, null);
            // 获取 selectView 中名为 list 的 ListView 组件
            final ListView listView = (ListView) selectView
                .findViewById(R.id.list);
            listView.setAdapter(adapter);
            new AlertDialog.Builder(MainActivity.this)
                .setView(selectView)
                .setPositiveButton("确定",
                new DialogInterface.OnClickListener()
                {
                    @Override
                    public void onClick(DialogInterface dialog,
                        int which)
                    {
                        // 清空 sendList 集合
                        sendList.clear();
                        // 遍历 listView 组件的每个列表项
                        for (int i = 0; i < listView.getCount(); i++)
                        {
                            CheckBox checkBox = (CheckBox) listView
                                .getChildAt(i);
                            // 如果该列表项被勾选
                            if (checkBox.isChecked())
                            {
                                // 添加该列表项的电话号码
                                sendList.add(checkBox.getText()
                                    .toString());
                            }
                        }
                        numbers.setText(sendList.toString());
                    }
                }).show();
            }
        });
    }
    // 判断某个电话号码是否已在群发范围内
    public boolean isChecked(String phone)
    {
        for (String s1 : sendList)
        {
            if (s1.equals(phone))
            {
                return true;
            }
        }
        return false;
    }
}
```

该程序的实现也很简单，程序提供了一个带列表框的对话框供用户选择群发短信的收件人号码，并使用了一个 ArrayLsit<String>集合来保存所有的收件人号码。为了实现群发功能，程序使用循环遍历 ArrayList<String>中的每个号码，并使用 SmsManager 依次向每个号码发送短信即可，如上面的粗体字代码所示。

注意：

上面的程序不仅需要调用 SmsManager 发送短信，也需要访问系统的联系人信息，因此不要忘记了在 AndroidManifest.xml 文件中给该程序授予相应的权限。

还有一点需要指出，该程序有一个潜在的风险：该程序直接在主线程中采用循环向多个人

发送短信，如果需要发送短信的人太多且网络延迟严重，群发短信就会变成一个耗时任务，此时可以考虑使用 IntentService 来群发短信，群发完成后通过广播通知前台 Activity。

10.4 音频管理器（AudioManager）

在某些时候，程序需要管理系统音量，或者直接让系统静音，这就可借助于 Android 提供的 AudioManager 来实现。程序一样是调用 getSystemService()方法来获取系统的音频管理器，接下来就可调用 AudioManager 的方法来控制手机音频了。

10.4.1 AudioManager 简介

在程序获取了 AudioManager 对象之后，接下来就可调用 AudioManager 的如下常用方法来控制手机音频了。

- adjustStreamVolume(int streamType, int direction, int flags)：调整手机指定类型的声音。其中第一个参数 streamType 指定声音类型，该参数可接受如下几个值。
- STREAM_ALARM：手机闹铃的声音。
- STREAM_DTMF：DTMF 音调的声音。
- STREAM_MUSIC：手机音乐的声音。
- STREAM_NOTIFICATION：系统提示的声音。
- STREAM_RING：电话铃声的声音。
- STREAM_SYSTEM：手机系统的声音。
- STREAM_VOICE_CALL：语音电话的声音。

第二个参数指定对声音进行增大还是减小；第三个参数是调整声音时的标志，例如指定 FLAG_SHOW_UI，则调整声音时显示音量进度条。

- setMicrophoneMute(boolean on)：设置是否让麦克风静音。
- setMode(int mode)：设置声音模式，可设置的值有 NORMAL、RINGTONE 和 IN_CALL。
- setRingerMode(int ringerMode)：设置手机的电话铃声模式。可支持如下几个属性值。
- RINGER_MODE_NORMAL：正常的手机铃声。
- RINGER_MODE_SILENT：手机铃声静音。
- RINGER_MODE_VIBRATE：手机振动。
- setSpeakerphoneOn(boolean on)：设置是否打开扩音器。
- setStreamMute(int streamType, boolean state)：将手机的指定类型的声音调整为静音。其中 streamType 参数与 adjustStreamVolume()方法中第一个参数的意义相同。
- setStreamVolume(int streamType, int index, int flags)：直接设置手机的指定类型的音量值。其中 streamType 参数与 adjustStreamVolume()方法中第一个参数的意义相同。

下面将通过一个简单的实例来示范 AudioManager 控制手机音频。

实例：使用 AudioManager 控制手机音频

本程序提供了一个按钮用于播放音乐，系统使用 MediaPlayer 播放音乐。程序还提供了两个按钮用于控制音乐声音量的提高、降低，并使用一个 ToggleButton 来控制是否静音。

该程序的界面比较简单，只是包含几个简单的按钮。该程序代码如下。

程序清单：codes\10\10.4\AudioTest\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    Button play, up, down;
    ToggleButton mute;
    AudioManager aManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的音频服务
        aManager = (AudioManager) getSystemService(
            Service.AUDIO_SERVICE);
        // 获取界面中三个按钮和一个 ToggleButton 控件
        play = (Button) findViewById(R.id.play);
        up = (Button) findViewById(R.id.up);
        down = (Button) findViewById(R.id.down);
        mute = (ToggleButton) findViewById(R.id.mute);
        // 为 play 按钮的单击事件绑定监听器
        play.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 初始化 MediaPlayer 对象，准备播放音乐
                MediaPlayer mPlayer = MediaPlayer.create(
                    MainActivity.this, R.raw.earth);
                // 设置循环播放
                mPlayer.setLooping(true);
                // 开始播放
                mPlayer.start();
            }
        });
        up.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 指定调节音乐的音频，增大音量，而且显示音量图形示意
                aManager.adjustStreamVolume(AudioManager.STREAM_MUSIC,
                    AudioManager.ADJUST_RAISE, AudioManager.FLAG_SHOW_UI);
            }
        });
        down.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 指定调节音乐的音频，降低音量，而且显示音量图形示意
                aManager.adjustStreamVolume(AudioManager.STREAM_MUSIC,
                    AudioManager.ADJUST_LOWER, AudioManager.FLAG_SHOW_UI);
            }
        });
        mute.setOnCheckedChangeListener(new OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(CompoundButton source,
                boolean isChecked)
            {
                // 指定调节音乐的音频，根据 isChecked 确定是否需要静音
                aManager.setStreamMute(AudioManager.STREAM_MUSIC,
                    isChecked);
```

```
            }
        });
    }
}
```

上面程序中的第一段粗体字代码使用 AudioManager 的 adjustStreamVolume()方法提高播放音乐的音量；第二段粗体字代码使用 AudioManager 的 adjustStreamVolume()方法降低播放音乐的音量；第三段粗体字代码则调用 AudioManager 的 setStreamMute()方法将音乐设为静音。

运行该程序，单击程序界面上“增大音量”或“降低音量”按钮，系统播放音乐的音量将会随之改变，如图 10.10 所示。

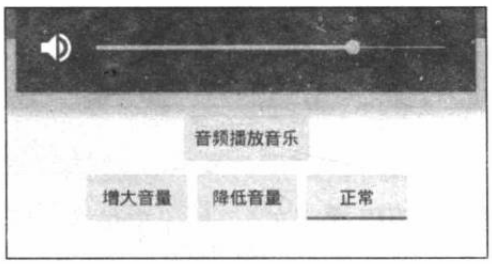


图 10.10 使用 AudioManager 管理音频

10.5 振动器（Vibrator）

在某些时候，程序需要启动系统振动器，比如手机静音时使用振动提示用户；再比如玩游戏时，当系统碰撞、爆炸时使用振动带给用户更逼真的体验等。总之，振动是除视频、声音之外的另一种“多媒体”，充分利用系统的振动器会带给用户更好的体验。

系统获取 Vibrator 也是调用 Context 的 getSystemService()方法即可，接下来就可调用 Vibrator 的方法来控制手机振动了。

10.5.1 Vibrator 简介

Vibrator 的使用比较简单，它只有三个简单的方法来控制手机振动。

- vibrate(long milliseconds)：控制手机振动 milliseconds 毫秒。
- vibrate(long[] pattern, int repeat)：指定手机以 pattern 指定的模式振动。例如指定 pattern 为 new int[400, 800, 1200, 1600]，就是指定在 400ms、800ms、1200ms、1600ms 这些时间点交替启动、关闭手机振动器；其中 repeat 指定 pattern 数组的索引，指定对 pattern 数组中从 repeat 索引开始的振动进行循环。
- cancel()：关闭手机振动。

掌握这些方法之后，接下来就可在程序中通过 Vibrator 来控制手机振动了。

10.5.2 使用 Vibrator 控制手机振动

下面这个程序十分简单，程序几乎不需要界面，重写了 Activity 的 onTouchEvent(MotionEvent event)方法，这样使得用户触碰手机触摸屏时将会启动手机振动。

程序清单：codes\10\10.5\VibratorTest\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    Vibrator vibrator;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 Vibrator 服务
        vibrator = (Vibrator) getSystemService(
            Service.VIBRATOR_SERVICE);
```

```
    }
    // 重写 onTouchEvent 方法，当用户触碰触摸屏时触发该方法
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        Toast.makeText(this, "手机振动"
            , Toast.LENGTH_SHORT).show();
        // 控制手机振动 2 秒
        vibrator.vibrate(2000);
        return super.onTouchEvent(event);
    }
}
```

上面程序中的第一行粗体字代码用于获取系统的 Vibrator 对象，第二行粗体字代码用于控制手机振动 2 秒。

需要指出的是，程序控制手机振动需要得到相应的权限，因此不要忘了在 AndroidManifest.xml 文件中增加如下授权配置代码：

```
<!-- 授予程序访问振动器的权限 -->
<uses-permission android:name="android.permission.VIBRATE"/>
```

在模拟器中运行该程序看不出振动效果，建议读者将该程序部署到真机上运行。

10.6 手机闹钟服务（AlarmManager）

AlarmManager 通常的用途就是用来开发手机闹钟，但实际上它的作用不止于此。它的本质是一个全局定时器，AlarmManager 可在指定时间或指定周期启动其他组件（包括 Activity、Service、BroadcastReceiver）。

10.6.1 AlarmManager 简介

AlarmManager 不仅可用于开发闹钟应用，还可作为一个全局定时器使用，在 Android 应用程序中也是通过 Context 的 getSystemService()方法来获取 AlarmManager 对象的。一旦程序获取了 AlarmManager 对象之后，就可调用它的如下方法来设置定时启动指定组件了。

- set(int type, long triggerAtTime, PendingIntent operation)：设置在 triggerAtTime 时间启动由 operation 参数指定的组件。其中第一个参数指定定时服务的类型，该参数可接受如下值。
  - ELAPSED_REALTIME：指定从现在开始时间过了一定时间后启动 operation 所对应的组件。
  - ELAPSED_REALTIME_WAKEUP：指定从现在开始时间过了一定时间后启动 operation 所对应的组件。即使系统处于休眠状态也会执行 operation 所对应的组件。
  - RTC：指定当系统调用 System.currentTimeMillis()方法的返回值与 triggerAtTime 相等时启动 operation 所对应的组件。
  - RTC_WAKEUP：指定当系统调用 System.currentTimeMillis()方法的返回值与 triggerAtTime 相等时启动 operation 所对应的组件。即使系统处于休眠状态也会执行 operation 所对应的组件。
- setInexactRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)：设置一个非精确的周期性任务。例如，我们设置 Alarm 每个小时启动一次，但系统并不一定总在每个小时的开始启动 Alarm 服务。

- setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)：设置一个周期性执行的定时服务。
- cancel(PendingIntent operation)：取消 AlarmManager 的定时服务。

掌握了 AlarmManager 的如上功能之后，接下来我们通过两个示例来示范 AlarmManager 在实际开发中的用途。

10.6.2 设置闹钟

这个程序比较简单，程序提供一个按钮让用户来设置闹铃时间（单击该按钮将会打开一个时间设置对话框），当用户设置好闹铃时间之后，即使退出该程序，到了预设时间 AlarmManager 也一样会启动指定组件——这是因为 AlarmManager 是一个全局定时器的缘故。

该程序的界面布局很简单，程序界面上只有一个简单的按钮。程序代码如下。

程序清单：codes\10\10.6\AlarmTest\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    Button setTime;
    Calendar currentTime = Calendar.getInstance();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面上的按钮
        setTime = (Button) findViewById(R.id.setTime);
        // 为“设置闹铃”按钮绑定监听器
        setTime.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Calendar currentTime = Calendar.getInstance();
                // 创建一个 TimePickerDialog 实例，并把它显示出来
                new TimePickerDialog(MainActivity.this, 0, // 绑定监听器
                    new TimePickerDialog.OnTimeSetListener()
                    {
                        @Override
                        public void onTimeSet(TimePicker tp,
                            int hourOfDay, int minute)
                        {
                            // 指定启动 AlarmActivity 组件
                            Intent intent = new Intent(MainActivity.this,
                                AlarmActivity.class);
                            // 创建 PendingIntent 对象
                            PendingIntent pi = PendingIntent.getActivity(
                                MainActivity.this, 0, intent, 0);
                            Calendar c = Calendar.getInstance();
                            c.setTimeInMillis(System.currentTimeMillis());
                            // 根据用户选择时间来设置 Calendar 对象
                            c.set(Calendar.HOUR, hourOfDay);
                            c.set(Calendar.MINUTE, minute);
                            // 获取 AlarmManager 对象
                            AlarmManager aManager = (AlarmManager)
                                getSystemService(ALARM_SERVICE);
                            // 设置 AlarmManager 将在 Calendar 对应的时间启动指定组件
                            aManager.set(AlarmManager.RTC_WAKEUP,
                                c.getTimeInMillis(), pi);
                            // 显示闹铃设置成功的提示信息
```

```
                        Toast.makeText(MainActivity.this, "闹铃设置成功啦"
                            , Toast.LENGTH_SHORT).show();
                    }
                }, currentTime.get(Calendar.HOUR_OF_DAY), currentTime
                .get(Calendar.MINUTE), false).show();
            }
        });
    }
}
```

上面程序中的粗体字代码控制 AlarmManager 将会在 Calendar 对应的时间启动 pi 对应的 Activity 组件，而且程序设置了 AlarmManager.RTC_WAKEUP 选项，这意味着即使系统处于关机状态，到了系统预设时间，AlarmManager 也会控制系统去执行 pi 对应的 Activity 组件。

上面的程序中 AlarmManager 需要启动的 Activity 为 AlarmActivity，它是一个非常简单的 Activity，甚至不需要程序界面，当该 Activity 加载时打开一个对话框提示闹钟时间到，并播放一段“激昂”的音乐提醒用户。AlarmActivity 代码如下。

程序清单：codes\10\10.6\AlarmTest\app\src\main\java\org\crazyit\manager\AlarmActivity.java

```
public class AlarmActivity extends Activity
{
    MediaPlayer alarmMusic;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 加载指定音乐，并为之创建 MediaPlayer 对象
        alarmMusic = MediaPlayer.create(this, R.raw.alarm);
        alarmMusic.setLooping(true);
        // 播放音乐
        alarmMusic.start();
        // 创建一个对话框
        new AlertDialog.Builder(AlarmActivity.this).setTitle("闹钟")
            .setMessage("闹钟响了,Go! Go! Go! ")
            .setPositiveButton("确定", new OnClickListener()
            {
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                    // 停止音乐
                    alarmMusic.stop();
                    // 结束该 Activity
                    AlarmActivity.this.finish();
                }
            }).show();
    }
}
```

上面 Activity 的作用只是通过一个对话框、一段音乐来提醒用户：闹钟时间到了。

不要忘记在 AndroidManifest.xml 文件中配置 AlarmActivity，如果用户忘记配置该 Activity，系统甚至不会提示用户：该 Activity 不存在！只是到了系统预设时间之后，程序将什么也不干。

运行该程序，简单地把闹钟时间设为下一分钟，即使我们退出该程序，过一分钟之后也将看到如图 10.11 所示的闹钟提示。

图 10.11 闹钟应用

需要说明的是，从 Android 4.4（API 19）开始，AlarmManager 的机制是非准确激发的，操作系统会偏移（shift）闹钟来最小化唤醒和电池消耗。不过 AlarmManager 新增了如下两个方法来支持精确激发。

- setExact(int type, long triggerAtMillis, PendingIntent operation)：设置闹钟将在精确的时间被激发。
- setWindow(int type, long windowStartMillis, long windowLengthMillis, PendingIntent operation)：设置闹钟将在精确的时间段内被激发。

对于 targetSdkVersion 在 API 19 之前应用仍将继续使用以前的行为，所有闹钟都会使用精确激发。

实例：定时更换壁纸

本实例程序将会通过 AlarmManager 来周期性地调用某个 Service，从而让系统实现定时更换壁纸的功能。

更换壁纸的 API 为 WallpaperManager，它提供了 clear()方法来清除壁纸，还提供了如下方法来设置系统的壁纸。

- setBitmap(Bitmap bitmap)：将壁纸设置为 bitmap 所代表的位图。
- setResource(int resid)：将壁纸设置为 resid 资源所代表的图片。
- setStream(InputStream data)：将壁纸设置为 data 数据所代表的图片。

该程序的界面中只有两个按钮，其中一个按钮用于启动定时更换壁纸，另一个按钮用于关闭定时更换壁纸。该程序的代码如下。

程序清单：codes\10\10.6\AlarmChangeWallpaper\app\src\main\java\org\crazyit\manager\MainActivity.java

```
public class MainActivity extends Activity
{
    Button start, stop;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        // 指定启动 ChangeService 组件
        Intent intent = new Intent(MainActivity.this, ChangeService.class);
        // 创建 PendingIntent 对象
        final PendingIntent pi = PendingIntent.getService(MainActivity.this, 0,
intent, 0);
        start.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 获取 AlarmManager 对象
                AlarmManager aManager = (AlarmManager) getSystemService(
                    Service.ALARM_SERVICE);
                // 设置每隔 5 秒执行 pi 所代表的组件一次
                aManager.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP
                    , 0, 5000, pi);
                start.setEnabled(false);
                stop.setEnabled(true);
                Toast.makeText(MainActivity.this
                    , "壁纸定时更换启动成功啦",
```

```
                    Toast.LENGTH_SHORT).show();
            }
        });
        stop.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
            start.setEnabled(true);
            stop.setEnabled(false);
            // 获取 AlarmManager 对象
            AlarmManager aManager = (AlarmManager) getSystemService(
                Service.ALARM_SERVICE);
            // 取消对 pi 的调度
            aManager.cancel(pi);
            }
        });
    }
}
```

上面程序中的粗体字代码指定程序每隔 5 秒执行一次 pi 所代表的组件。程序中 pi 代表了 ChangeService 组件，因此还需要为该应用程序提供一个 ChangeService 组件，该组件的代码如下。

程序清单：codes\10\10.6\AlarmChangeWallpaper\app\src\main\java\org\crazyit\manager\ChangeService.java

```
public class ChangeService extends Service
{
    // 定义定时更换的壁纸资源
    int[] wallpapers = new int[]{
        R.drawable.shuangta,
        R.drawable.lijiang,
        R.drawable.qiao,
        R.drawable.shui
    };
    // 定义系统的壁纸管理服务
    WallpaperManager wManager;
    // 定义当前所显示的壁纸
    int current = 0;
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        // 如果到了最后一张，系统重新开始
        if(current >= 4)
            current = 0;
        try
        {
            // 改变壁纸
            wManager.setResource(wallpapers[current++]);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return START_STICKY;
    }
    @Override
    public void onCreate()
    {
        super.onCreate();
        // 初始化 WallpaperManager
        wManager = WallpaperManager.getInstance(this);
    }
```

```
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }
}
```

上面的程序重写了 Service 的 onStartCommand()方法，这意味着程序每次启动该 Service 时都会执行 onStartCommand()方法中的代码，而 onStartCommand()方法中的粗体字代码负责更换系统的壁纸。

为了允许该程序改变壁纸，还需要在 AndroidManifest.xml 文件中为该程序授予相应的权限，也就是在 AndroidManifest.xml 文件中增加如下代码：

```
<!-- 授予用户修改壁纸的权限 -->
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
```

运行该程序，并单击程序界面上的“启动定时更换”按钮，接着退出该程序，返回到系统桌面，将可以看到系统桌面每 5 秒更换一次，如图 10.12 所示。

图 10.12 定时更换壁纸

当然，这个程序还有值得改进的地方，这个程序所更换的壁纸只是程序预设的几张图片，如果把这个程序与前面介绍的 SD 卡文件浏览器结合起来，让用户可以添加 SD 卡中的图片作为可供更换的壁纸图片，那么这个程序就算比较完善了。

10.7 接收广播消息

Android 系统的四大组件还有一种 BroadcastReceiver，这种组件本质上就是一个全局监听器，用于监听系统全局的广播消息。由于 BroadcastReceiver 是一个全局监听器，因此它可以非常方便地实现系统中不同组件之间的通信。例如，我们希望客户端程序与 startService()方法启动的 Service 之间通信，就可以借助于 BroadcastReceiver 来实现。

10.7.1 BroadcastReceiver 简介

BroadcastReceiver 用于接收程序（包括用户开发的程序和系统内建的程序）所发出的 Broadcast Intent，与应用程序启动 Activity、Service 相同的是，程序启动 BroadcastReceiver 也只需要两步。

1. 创建需要启动的 BroadcastReceiver 的 Intent。
2. 调用 Context 的 sendBroadcast()或 sendOrderedBroadcast()方法来启动指定的 BroadcastReceiver。

当应用程序发出一个 Broadcast Intent 之后，所有匹配该 Intent 的 BroadcastReceiver 都有可能被启动。

与 Activity、Service 具有完整的生命周期不同，BroadcastReceiver 本质上只是一个系统级的监听器——它专门负责监听各程序所发出的 Broadcast。

提示：

前面介绍的各种 OnXxxListener 只是程序级别的监听器，这些监听器运行在指定程序所在进程中，当程序退出时，OnXxxListener 监听器也就随之关闭了。但 BroadcastReceiver 属于系统级的监听器，它拥有自己的进程，只要存在与之匹配的 Intent 被广播出来，BroadcastReceiver 就会被激发。

由于 BroadcastReceiver 本质上属于一个监听器，因此实现 BroadcastReceiver 的方法也十分简单，只要重写 BroadcastReceiver 的 onReceive(Context context, Intent intent)方法即可。

一旦实现了 BroadcastReceiver，接下来就应该指定该 BroadcastReceiver 能匹配的 Intent，此时有两种方式。

- 使用代码进行指定，调用 BroadcastReceiver 的 Context 的 registerReceiver (BroadcastReceiver receiver, IntentFilter filter)方法指定。例如如下代码：

```
IntentFilter filter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
IncomingSMSReceiver receiver = new IncomingSMSReceiver();
registerReceiver(receiver, filter);
```

- 在 AndroidManifest.xml 文件中配置。例如如下代码：

```
<receiver android:name=".IncomingSMSReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

每次系统 Broadcast 事件发生后，系统就会创建对应的 BroadcastReceiver 实例，并自动触发它的 onReceive() 方法，onReceive()方法执行完后，BroadcastReceiver 实例就会被销毁。

提示：

与 Activity 组件不同的是，当系统通过 Intent 启动指定了 Activity 组件时，如果系统没有找到合适的 Activity 组件，则会导致程序异常中止；但系统通过 Intent 激发 BroadcastReceiver 时，如果找不到合适的 BroadcastReceiver 组件，应用不会有任何问题。

如果 BroadcastReceiver 的 onReceive()方法不能在 10 秒内执行完成，Android 会认为该程序无响应。所以不要在 BroadcastReceiver 的 onReceive()方法里执行一些耗时的操作；否则会弹出 ANR（Application No Response）对话框。

如果确实需要根据 Broadcast 来完成一项比较耗时的操作，则可以考虑通过 Intent 启动一个 Service 来完成该操作。不应考虑使用新线程去完成耗时的操作，因为 BroadcastReceiver 本身的生命周期很短，可能出现的情况是子线程可能还没有结束，BroadcastReceiver 就已经退出了。

如果 BroadcastReceiver 所在的进程结束了，虽然该进程内还有用户启动的新线程，但由于该进程内不包含任何活动组件，因此系统可能在内存紧张时优先结束该进程。这样就可能导致 BroadcastReceiver 启动的子线程不能执行完成。

10.7.2 发送广播

在程序中发送广播十分简单，只要调用 Context 的 sendBroadcast(Intent intent)方法即可，这条广播将会启动 intent 参数所对应的 BroadcastReceiver。

下面简单的程序示范了如何发送 Broadcast、使用 BroadcastReceiver 接收广播。该程序的

Activity 界面中包含一个按钮，当用户单击该按钮时程序会向外发送一条广播。该程序的代码如下。

程序清单：codes\10\10.7\Broadcast\app\src\main\java\org\crazyit\broadcast\MainActivity.java

```
public class MainActivity extends Activity
{
    Button send;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的按钮
        send = (Button) findViewById(R.id.send);
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                // 设置 Intent 的 Action 属性
                intent.setAction("org.crazyit.action.CRAZY_BROADCAST");
                intent.putExtra("msg", "简单的消息");
                // 发送广播
                sendBroadcast(intent);
            }
        });
    }
}
```

上面程序中的粗体字代码用于创建一个 Intent 对象，并使用该 Intent 对象对外发送一条广播。该程序所使用的 BroadcastReceiver 代码如下。

程序清单：codes\10\10.7\Broadcast\app\src\main\java\org\crazyit\broadcast\MyReceiver.java

```
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context,
            "接收到的 Intent 的 Action 为：" + intent.getAction()
            + "\n 消息内容是：" + intent.getStringExtra("msg")
            , Toast.LENGTH_LONG).show();
    }
}
```

正如上面的程序中看到的，当符合该 MyReceiver 的广播出现时，该 MyReceiver 的 onReceive()方法将会被触发，从而在该方法中显示广播所携带的消息。

上面发送广播的程序中指定发送广播时所用的 Intent 的 Action 为 org.crazyit.action.CRAZY_BROADCAST，这就需要配置上面的 BroadcastReceiver 应监听 Action 为该字符串的 Intent，在 AndroidManifest.xml 文件中增加如下配置即可：

```
<receiver android:name=".MyReceiver">
    <intent-filter>
        <!-- 指定该 BroadcastReceiver 所响应的 Intent 的 Action -->
        <action android:name="org.crazyit.action.CRAZY_BROADCAST" />
    </intent-filter>
</receiver>
```

运行该程序，并单击程序界面中的“发送广播”按钮，将看到程序有如图 10.13 所示的提示。

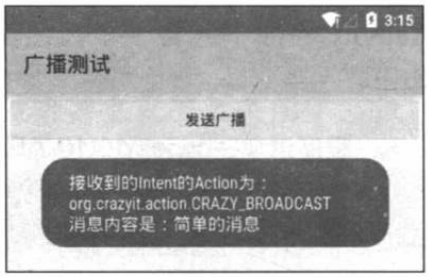


图 10.13 响应广播

10.7.3 有序广播

Broadcast 被分为如下两种。

- **Normal Broadcast（普通广播）**：Normal Broadcast 是完全异步的，可以在同一时刻（逻辑上）被所有接收者接收到，消息传递的效率比较高。但缺点是接收者不能将处理结果传递给下一个接收者，并且无法终止 Broadcast Intent 的传播。
- **Ordered Broadcast（有序广播）**：Ordered Broadcast 的接收者将按预先声明的优先级依次接收 Broadcast。比如 A 的级别高于 B、B 的级别高于 C，那么 Broadcast 先传给 A，再传给 B，最后传给 C 。优先级别声明在<intent-filter.../>元素的 android:priority 属性中，数越大优先级别越高，取值范围为-1000~1000，也可以调用 IntentFilter 对象的 setPriority()设置优先级别。Ordered Broadcast 接收者可以终止 Broadcast Intent 的传播，Broadcast Intent 的传播一旦终止，后面的接收者就无法接收到 Broadcast。另外，Ordered Broadcast 的接收者可以将数据传递给下一个接收者，比如 A 得到 Broadcast 后，可以往它的结果对象中存入数据，当 Broadcast 传给 B 时，B 可以从 A 的结果对象中得到 A 存入的数据。

Context 提供的如下两个方法用于发送广播。

- **sendBroadcast()**：发送 Normal Broadcast。
- **sendOrderedBroadcast()**：发送 Ordered Broadcast。

对于 Ordered Broadcast 而言，系统会根据接收者声明的优先级别按顺序逐个执行接收者，优先接收到 Broadcast 的接收者可以终止 Broadcast，调用 BroadcastReceiver 的 abortBroadcast()方法即可终止 Broadcast。如果 Broadcast 被前面的接收者终止，后面的广播接收者就再也无法获取到 Broadcast 了。

不仅如此，对于 Ordered Broadcast 而言，优先接收到 Broadcast 的接收者可以通过 setResultExtras(Bundle)方法将处理结果存入 Broadcast 中，然后传给下一个接收者，下一个接收者通过代码 Bundle bundle = getResultExtras(true)可以获取上一个接收者存入的数据。

提示：

系统收到短信，发出的 Broadcast 属于 Ordered Broadcast。如果想阻止用户收到短信，可以通过设置优先级，让自定义的 BroadcastReceiver 先获取到 Broadcast，然后终止 Broadcast。

接下来介绍一个发送有序广播的示例，该程序的 Activity 界面上只有一个普通按钮，用于发送一条有序广播。该程序代码如下。

程序清单：codes\10\10.7\SortedBroadcast\app\src\main\java\org\crazyit\broadcast\MainActivity.java

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

```
        // 获取程序中的 send 按钮
        send = (Button) findViewById(R.id.send);
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                intent.setAction("org.crazyit.action.CRAZY_BROADCAST");
                intent.putExtra("msg", "简单的消息");
                // 发送有序广播
                sendOrderedBroadcast(intent, null);
            }
        });
    }
}
```

上面程序中的粗体字代码指定了 Intent 的 Action 属性，再调用 sendOrderedBroadcast()方法来发送有序广播。对于有序广播而言，它会按优先级依次触发每个 BroadcastReceiver 的 onReceive()方法。

下面的程序先定义了第一个 BroadcastReceiver。

程序清单：codes\10\10.7\SortedBroadcast\app\src\main\java\org\crazyit\broadcast\MyReceiver.java

```
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context,    "接收到的 Intent 的 Action 为："
            + intent.getAction() + "\n 消息内容是："
            + intent.getStringExtra("msg")
            , Toast.LENGTH_LONG).show();
        // 创建一个 Bundle 对象，并存入数据
        Bundle bundle = new Bundle();
        bundle.putString("first", "第一个 BroadcastReceiver 存入的消息");
        // 将 bundle 放入结果中
        setResultExtras(bundle);
        // 取消 Broadcast 的继续传播
        // abortBroadcast(); // ①
    }
}
```

上面的 BroadcastReceiver 不仅处理了它所接收到的消息，而且向处理结果中存入了 key 为 first 的消息，这个消息将可以被第二个 BroadcastReceiver 解析出来。

上面程序中的①号粗体字代码用于取消广播，如果保持这条代码生效，那么优先级比 MyReceiver 低的 BroadcastReceiver 都将不会被触发。

在 AndroidManifest.xml 文件中部署该 BroadcastReceiver，并指定其优先级为 20，配置片段如下：

```
<receiver android:name=".MyReceiver">
    <intent-filter android:priority="20">
        <action android:name="org.crazyit.action.CRAZY_BROADCAST" />
    </intent-filter>
</receiver>
```

接下来为程序提供第二个 BroadcastReceiver，这个 BroadcastReceiver 将会解析前一个 BroadcastReceiver 所存入的 key 为 first 的消息。该 BroadcastReceiver 的代码如下。

程序清单：codes\10\10.7\SortedBroadcast\app\src\main\java\org\crazyit\broadcast\MyReceiver2.java

```
public class MyReceiver2 extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = getResultExtras(true);
        // 解析前一个 BroadcastReceiver 所存入的 key 为 first 的消息
        String first = bundle.getString("first");
        Toast.makeText(context, "第一个 Broadcast 存入的消息为："
            + first, Toast.LENGTH_LONG).show();
    }
}
```

上面程序中的粗体字代码用于解析前一个 BroadcastReceiver 存入结果中的 key 为 first 的消息，在 AndroidManifest.xml 文件中配置该 BroadcastReceiver，并指定其优先级为 0。配置片段如下：

```
<receiver android:name=".MyReceiver2">
    <intent-filter android:priority="0">
        <action android:name="org.crazyit.action.CRAZY_BROADCAST" />
    </intent-filter>
</receiver>
```

图 10.14 获取前一个 BroadcastReceiver 存入结果中的消息

根据上面的配置可以看出，该程序中包含两个 BroadcastReceiver，其中 MyReceiver 的优先级更高，MyReceiver2 的优先级略低。如果将程序中①号粗体字代码注释掉，那么程序中 MyReceiver2 将可以被触发，并解析得到 MyReceiver 存入结果中的 key 为 first 的消息，因此看到如图 10.18 所示的输出。

如果不注释掉程序中①号粗体字代码，这行代码将会阻止消息广播，这样消息将传不到 MyReceiver2 了。

实例：基于 Service 的音乐播放器

前面已经提到 BroadcastReceiver 是一个全局监听器，因此 BroadcastReceiver 也就提供了让不同组件之间进行通信的新思路。比如程序有一个 Activity、一个 Service，而且该 Service 是通过 startService()方法启动起来的，在正常情况下，这个 Activity 与通过 startService()方法启动的 Service 之间无法通信，但借助于 BroadcastReceiver 的帮助，程序就可以实现两者之间的通信了。

本实例程序开发了一个基于 Service 组件的音乐盒，程序的音乐将会由后台运行的 Service 组件负责播放，当后台的播放状态发生改变时，程序将会通过发送广播通知前台 Activity 更新界面；当用户单击前台 Activity 的界面按钮时，系统将通过发送广播通知后台 Service 来改变播放状态。

前台 Activity 的界面很简单，它只有两个按钮，分别用于控制播放/暂停、停止；另外还有两个文本框，用于显示正在播放的歌曲名、歌手名。前台 Activity 的代码如下。

程序清单：codes\10\10.7\MusicBox\app\src\main\java\org\crazyit\broadcast\MainActivity.java

```
public class MainActivity extends Activity implements OnClickListener
{
    // 获取界面中的显示歌曲标题、作者文本框
```

```
TextView title, author;
// 播放/暂停、停止按钮
ImageButton play, stop;
ActivityReceiver activityReceiver;
public static final String CTL_ACTION =
    "org.crazyit.action.CTL_ACTION";
public static final String UPDATE_ACTION =
    "org.crazyit.action.UPDATE_ACTION";
// 定义音乐的播放状态，0x11 代表没有播放；0x12 代表正在播放；0x13 代表暂停
int status = 0x11;
String[] titleStrs = new String[] { "心愿", "约定", "美丽新世界" };
String[] authorStrs = new String[] { "未知艺术家", "周蕙", "伍佰" };
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 获取程序界面中的两个按钮
    play = (ImageButton) this.findViewById(R.id.play);
    stop = (ImageButton) this.findViewById(R.id.stop);
    title = (TextView) findViewById(R.id.title);
    author = (TextView) findViewById(R.id.author);
    // 为两个按钮的单击事件添加监听器
    play.setOnClickListener(this);
    stop.setOnClickListener(this);
    activityReceiver = new ActivityReceiver();
    // 创建 IntentFilter
    IntentFilter filter = new IntentFilter();
    // 指定 BroadcastReceiver 监听的 Action
    filter.addAction(UPDATE_ACTION);
    // 注册 BroadcastReceiver
    registerReceiver(activityReceiver, filter);
    Intent intent = new Intent(this, MusicService.class);
    // 启动后台 Service
    startService(intent);
}
// 自定义的 BroadcastReceiver，负责监听从 Service 传回来的广播
public class ActivityReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 获取 Intent 中的 update 消息，update 代表播放状态
        int update = intent.getIntExtra("update", -1);
        // 获取 Intent 中的 current 消息，current 代表当前正在播放的歌曲
        int current = intent.getIntExtra("current", -1);
        if (current >= 0)
        {
            title.setText(titleStrs[current]);
            author.setText(authorStrs[current]);
        }
        switch (update)
        {
            case 0x11:
                play.setImageResource(R.drawable.play);
                status = 0x11;
                break;
            // 控制系统进入播放状态
            case 0x12:
                // 在播放状态下设置使用暂停图标
                play.setImageResource(R.drawable.pause);
                // 设置当前状态
                status = 0x12;
```

```
                    break;
                // 控制系统进入暂停状态
                case 0x13:
                    // 在暂停状态下设置使用播放图标
                    play.setImageResource(R.drawable.play);
                    // 设置当前状态
                    status = 0x13;
                    break;
            }
        }
    }
    @Override
    public void onClick(View source)
    {
        // 创建 Intent
        Intent intent = new Intent("org.crazyit.action.CTL_ACTION");
        switch (source.getId())
        {
            // 按下播放/暂停按钮
            case R.id.play:
                intent.putExtra("control", 1);
                break;
            // 按下停止按钮
            case R.id.stop:
                intent.putExtra("control", 2);
                break;
        }
        // 发送广播，将被 Service 组件中的 BroadcastReceiver 接收到
        sendBroadcast(intent);
    }
}
```

上面程序的关键代码就是两段粗体字代码：

- 第一段粗体字代码用于响应后台 Service 所发出的广播，该程序将会根据广播 Intent 里的消息来改变播放状态，并更新程序界面中按钮的图标：当正在播放时，显示暂停图标；当正在暂停时，显示播放图标。并根据传回来的 current 数据来更新 title、author 两个文本框所显示的文本——显示当前正在播放的歌曲的歌名和歌手。
- 第二段粗体字代码则根据用户单击的按钮发送广播，发送广播时会把所按下的按钮的标识发送出来。发送的广播将激发后台 Service 的 BroadcastReceiver，该 BroadcastReceiver 将会根据广播消息来改变播放状态。

与之对应的是，该程序的后台 Service 也一样，它会在播放状态发生改变时对外发送广播（广播将会激发前台 Activity 的 BroadcastReceiver）；它也会采用 BroadcastReceiver 监听来自前台 Activity 所发出的广播。后台 Service 的代码如下。

程序清单：codes\10\10.7\MusicBox\app\src\main\java\org\crazyit\broadcast\MusicService.java

```
public class MusicService extends Service
{
    MyReceiver serviceReceiver;
    AssetManager am;
    String[] musics = new String[] { "wish.mp3", "promise.mp3",
        "beautiful.mp3" };
    MediaPlayer mPlayer;
    // 当前的状态，0x11 代表没有播放；0x12 代表正在播放；0x13 代表暂停
    int status = 0x11;
    // 记录当前正在播放的音乐
    int current = 0;
    @Override
    public IBinder onBind(Intent intent)
```

```
{
    return null;
}
@Override
public void onCreate()
{
    super.onCreate();
    am = getAssets();
    // 创建BroadcastReceiver
    serviceReceiver = new MyReceiver();
    // 创建IntentFilter
    IntentFilter filter = new IntentFilter();
    filter.addAction(MainActivity.CTL_ACTION);
    registerReceiver(serviceReceiver, filter);
    // 创建MediaPlayer
    mPlayer = new MediaPlayer();
    // 为MediaPlayer播放完成事件绑定监听器
    mPlayer.setOnCompletionListener(new OnCompletionListener() // ①
    {
        @Override
        public void onCompletion(MediaPlayer mp)
        {
            current++;
            if (current >= 3)
            {
                current = 0;
            }
            // 发送广播通知Activity更改文本框
            Intent sendIntent = new Intent(MainActivity.UPDATE_ACTION);
            sendIntent.putExtra("current", current);
            // 发送广播，将被Activity组件中的BroadcastReceiver接收到
            sendBroadcast(sendIntent);
            // 准备并播放音乐
            prepareAndPlay(musics[current]);
        }
    });
}
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(final Context context, Intent intent)
    {
        int control = intent.getIntExtra("control", -1);
        switch (control)
        {
            // 播放或暂停
            case 1:
                // 原来处于没有播放状态
                if (status == 0x11)
                {
                    // 准备并播放音乐
                    prepareAndPlay(musics[current]);
                    status = 0x12;
                }
                // 原来处于播放状态
                else if (status == 0x12)
                {
                    // 暂停
                    mPlayer.pause();
                    // 改变为暂停状态
                    status = 0x13;
                }
                // 原来处于暂停状态
```

```
                else if (status == 0x13)
                {
                    // 播放
                    mPlayer.start();
                    // 改变状态
                    status = 0x12;
                }
                break;
            // 停止声音
            case 2:
                // 如果原来正在播放或暂停
                if (status == 0x12 || status == 0x13)
                {
                    // 停止播放
                    mPlayer.stop();
                    status = 0x11;
                }
            }
            // 广播通知 Activity 更改图标、文本框
            Intent sendIntent = new Intent(MainActivity.UPDATE_ACTION);
            sendIntent.putExtra("update", status);
            sendIntent.putExtra("current", current);
            // 发送广播，将被 Activity 组件中的 BroadcastReceiver 接收到
            sendBroadcast(sendIntent);
        }
    }
    private void prepareAndPlay(String music)
    {
        try
        {
            // 打开指定音乐文件
            AssetFileDescriptor afd = am.openFd(music);
            mPlayer.reset();
            // 使用 MediaPlayer 加载指定的声音文件。
            mPlayer.setDataSource(afd.getFileDescriptor(),
                afd.getStartOffset(), afd.getLength());
            // 准备声音
            mPlayer.prepare();
            // 播放
            mPlayer.start();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

上面程序中的粗体字代码用于接收来自前台 Activity 所发出的广播，并根据广播的消息内容改变 Service 的播放状态，当播放状态改变时，该 Service 对外发送一条广播，广播消息将会被前台 Activity 接收，前台 Activity 将会根据广播消息去更新程序界面。

除此之外，为了让该音乐播放器能按顺序依次播放每首歌曲，程序为 MediaPlayer 增加了 OnCompletionListener 监听器，当 MediaPlayer 播放完成后将让它自动播放下一首歌曲，如程序中①号代码所示。

图 10.15 音乐播放器

运行该程序，将看到如图 10.15 示的界面。

由于该程序采用了后台 Service 来播放音乐，因此即使用户退出该程序，后台也依然会播放音乐，这就是该程序的独特之处。如果用户希望停止播放，只要单击图 10.15

所示界面中的“停止”按钮，前台 Activity 就会发送广播通知后台 Service 停止播放。

该程序用到了两个 BroadcastReceiver，但已经在程序中注册了两个 BroadcastReceiver 所监听的 IntentFilter，因此无须在 AndroidManifest.xml 文件中注册这两个 BroadcastReceiver，只要注册该程序所用的前台 Activity、后台 Service 即可。

10.8 接收系统广播消息

除了接收用户发送的广播之外，BroadcastReceiver 还有一个重要的用途：接收系统广播。如果应用需要在系统特定时刻执行某些操作，就可以通过监听系统广播来实现。Android 的大量系统事件都会对外发送标准广播。下面是 Android 常见的广播 Action 常量（具体请参考 Android API 文档中关于 Intent 的说明）。

- ACTION_TIME_CHANGED：系统时间被改变。
- ACTION_DATE_CHANGED：系统日期被改变。
- ACTION_TIMEZONE_CHANGED：系统时区被改变。
- ACTION_BOOT_COMPLETED：系统启动完成。
- ACTION_PACKAGE_ADDED：系统添加包。
- ACTION_PACKAGE_CHANGED：系统的包改变。
- ACTION_PACKAGE_REMOVED：系统的包被删除。
- ACTION_PACKAGE_RESTARTED：系统的包被重启。
- ACTION_PACKAGE_DATA_CLEARED：系统的包数据被清空。
- ACTION_BATTERY_CHANGED：电池电量改变。
- ACTION_BATTERY_LOW：电池电量低。
- ACTION_POWER_CONNECTED：系统连接电源。
- ACTION_POWER_DISCONNECTED：系统与电源断开。
- ACTION_SHUTDOWN：系统被关闭。

通过使用 BroadcastReceiver 来监听特殊的广播，即可让应用随系统执行特定的操作。

实例：开机自动运行的 Service

前面已经介绍了多个程序，需要使用开机自动运行的 Service，例如监听用户来电、监听用户短信、拦截黑名单电话……为了让 Service 随系统启动自动运行，可以让 BroadcastReceiver 监听 Action 为 ACTION_BOOT_COMPLETED 常量的 Intent，然后在 BroadcastReceiver 中启动特定 Service 即可。

该程序所用的 BroadcastReceiver 代码如下。

程序清单：codes\10\10.8\LaunchService\app\src\main\java\org\crazyit\broadcast\LaunchReceiver.java

```
public class LaunchReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Intent tIntent = new Intent(context
            , LaunchService.class);
        // 启动指定 Service
        context.startService(tIntent);
    }
}
```

该LaunchReceiver的代码十分简单，只要在onReceive()方法中启动指定Service即可，关键是要让LaunchReceiver监听系统开机发出的广播，因此需要在AndroidManifest.xml文件中采用如下代码配置该BroadcastReceiver：

```
<!-- 定义一个BroadcastReceiver，监听系统开机广播 -->
<receiver android:name=".LaunchReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

除此之外，为了让程序能访问系统开机事件，还需要为应用程序增加如下权限：

```
<!-- 授予应用程序访问系统开机事件的权限 -->
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

至于程序中用到的LaunchService，则可以是用户开发的任意Service，既可以是监听用户来电的Service，也可以是监听用户短信、拦截黑名单电话等的Service，此处不再给出该Service的代码。

实例：短信提醒

前面已经提到，当系统收到短信时，系统会对外发送一个有序广播，该广播的Intent的Action为android.provider.Telephony.SMS_RECEIVED。因此，只要在程序中开发一个对应的BroadcastReceiver即可监听到系统收到的短信。

该程序对应的BroadcastReceiver代码如下。

程序清单：codes\10\10.8\MonitorSms\app\src\main\java\org\crazyit\broadcast\SmsReceiver.java

```
public class SmsReceiver extends BroadcastReceiver
{
    // 当接收到短信时被触发
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 如果是接收到短信
        if (intent.getAction().equals(
            "android.provider.Telephony.SMS_RECEIVED"))
        {
            // 取消广播（这行代码将会让系统收不到短信）
            abortBroadcast();  // ①
            StringBuilder sb = new StringBuilder();
            // 接收由SMS传过来的数据
            Bundle bundle = intent.getExtras();
            // 判断是否有数据
            if (bundle != null)
            {
                // 通过pdus可以获得接收到的所有短信消息
                Object[] pdus = (Object[]) bundle.get("pdus");
                // 构建短信对象array，并依据收到的对象长度来创建array的大小
                SmsMessage[] messages = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++)
                {
                    messages[i] = SmsMessage
                        .createFromPdu((byte[]) pdus[i]);
                }
                // 将发送来的短信合并自定义信息于StringBuilder当中
                for (SmsMessage message : messages)
                {
```

```
                sb.append("短信来源:");
                // 获得接收短信的电话号码
                sb.append(message.getDisplayOriginatingAddress());
                sb.append("\n------短信内容------\n");
                // 获得短信的内容
                sb.append(message.getDisplayMessageBody());
            }
        }
        Toast.makeText(context, sb.toString()
            , Toast.LENGTH_LONG).show();
    }
}
```

上面的程序重写了 BroadcastReceiver 的 onReceive()方法，并在该方法中获取所收到的短信内容，然后使用 Toast 显示短信内容。

为了让该程序在系统的短信接收程序之前被启动，我们将该 BroadcastReceiver 的优先级设得高一些，在 AndroidManifest.xml 文件中通过如下代码来配置该 BroadcastReceiver：

```
<receiver android:name="SmsReceiver">
    <intent-filter android:priority="1000">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

上面的配置片段中粗体字代码指定了该 BroadcastReceiver 的优先级为 1000，这样它就可以在系统短信接收程序之前被触发了。为了让该程序拥有读取短信的权限，还需要在 AndroidManifest.xml 文件中为该程序进行授权，授权的配置代码如下：

```
<!-- 授予程序接收短信的权限 -->
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

由于该 BroadcastReceiver 将会位于系统短信接收程序之前被启动，如果保持该程序中①号粗体字代码生效，那么该程序接收到短信广播之后，广播将不会被传播到系统的短信接收程序——也就是系统本身将不会收到短信。

运行该程序，然后再启动另一个模拟器向该程序所在模拟器发送短信，将可以看到如图 10.16 所示的界面。

图 10.16 短信提醒

当然，该程序也可改变成所谓的“短信窃听”程序，只要让该程序接收到短信之后并不将该短信显示出来，而是将短信存入系统中，或者再以短信的方式发送到指定手机，这样就实现了“短信窃听”功能，不过这样做是“不道德”的。

实例：手机电量提示

当手机电量发生改变时，系统会对外发送 Intent 的 Action 为 ACTION_BATTERY_CHANGED 常量的广播；当手机电量过低时，系统会对外发送 Intent 的 Action 为 ACTION_BATTERY_LOW 常量的广播。通过开发监听对应 Intent 的 BroadcastReceiver，即可让系统对手机电量进行提示。

下面的程序即可对手机电量过低进行提示。

程序清单：codes\10\10.8\MonitorBattery\app\src\main\java\org\crazyit\broadcast\BatteryReceiver.java

```
public class BatteryReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
```

```
        {
            Bundle bundle = intent.getExtras();
            // 获取当前电量
            int current = bundle.getInt("level");
            // 获取总电量
            int total = bundle.getInt("scale");
            // 如果当前电量小于总电量的15%
            if (current * 1.0 / total < 0.15)
            {
                Toast.makeText(context, "电量过低，请尽快充电！"
                    , Toast.LENGTH_LONG).show();
            }
        }
    }
```

上面程序中的粗体字代码即可通过接收到的广播消息获取系统总电量、当前电量，如果当前电量小于总电量的15%，则程序会显示电量过低的提示信息。

10.9 本章小结

前面已经介绍了 Activity 和 ContentProvider 两个组件，再加上本章所介绍的 Service、BroadcastReceiver 两个组件，这就是 Android 系统的四大组件了。学习 Service 需要重点掌握创建、配置 Service 组件，以及如何启动、停止 Service；不仅如此，如何开发 IntentService 也是需要重点掌握的内容。学习 BroadcastReceiver 需要掌握创建、配置 BroadcastReceiver 组件，还需要掌握在程序中发送 Broadcast 的方法。除此之外，本章还介绍了大量系统 Service 的功能和用法，包括 TelephonyManager、SmsManager、AudioManager、Vibrator、AlarmManager 等，需要读者熟练掌握并能熟练使用。

CHAPTER

11

第11章 多媒体应用开发

本章要点

- 音频和视频
- 使用 MediaPlayer 播放音频
- 使用 SoundPool 播放音频
- 使用 VideoView 播放视频
- 使用 MediaPlayer 与 SurfaceView 播放视频
- 使用 MediaRecorder 录制音频
- 控制摄像头拍照
- 控制摄像头录制视频短片

Android 应用面向的是普通个人用户，这些用户往往会更加关注用户体验，因此为 Android 应用增加动画、视频、音乐等多媒体功能十分必要。就目前的手机发展趋势来看，手机已经不再是单一的通信工具，已经发展成集照相机、音乐播放器、视频播放器、个人小型终端于一体的智能设备，因此为手机提供音频录制、播放，视频录制、播放的功能十分重要。

Android 提供了常见音频、视频的编码、解码机制，就像之前所用过的 MediaPlayer 类，Android 支持的音频格式有 MP3、WAV 和 3GP 等，支持的视频格式有 MP4 和 3GP 等。

借助于这些多媒体支持类，我们可以非常方便地在手机应用中播放音频、视频等，这些多媒体数据既可是来自于 Android 应用的资源文件，也可是来自于外部存储器上的文件，甚至可以是来自于网络的文件流。不仅如此，Android 也提供了对摄像头、麦克风的支持，因此也可以十分方便地从外部采集照片、视频、音频等多媒体信息。

11.1 音频和视频的播放

Android 提供了简单的 API 来播放音频、视频，下面将会详细介绍如何使用它们。

11.1.1 使用 MediaPlayer 播放音频

前面已经提及如何使用 MediaPlayer 播放音频的例子，使用 MediaPlayer 播放音频十分简单，当程序控制 MediaPlayer 对象装载音频完成之后，程序可以调用 MediaPlayer 的如下三个方法进行播放控制。

- start()：开始或恢复播放。
- stop()：停止播放。
- pause()：暂停播放。

为了让 MediaPlayer 来装载指定音频文件，MediaPlayer 提供了如下简单的静态方法。

- static MediaPlayer create(Context context, Uri uri)：从指定 Uri 来装载音频文件，并返回新创建的 MediaPlayer 对象。
- static MediaPlayer create(Context context, int resid)：从 resid 资源 ID 对应的资源文件中装载音频文件，并返回新创建的 MediaPlayer 对象。

上面两个方法用起来非常方便，但这两个方法每次都会返回新创建的 MediaPlayer 对象，如果程序需要使用 MediaPlayer 循环播放多个音频文件，使用 MediaPlayer 的静态 create()方法就不太合适了，此时可通过 MediaPlayer 的 setDataSource()方法来装载指定的音频文件。MediaPlayer 提供了如下方法来指定装载相应的音频文件。

- setDataSource(String path)：指定装载 path 路径所代表的文件。
- setDataSource(FileDescriptor fd, long offset, long length)：指定装载 fd 所代表的文件中从 offset 开始、长度为 length 的文件内容。
- setDataSource(FileDescriptor fd)：指定装载 fd 所代表的文件。
- setDataSource(Context context, Uri uri)：指定装载 uri 所代表的文件。

执行上面所示的 setDataSource()方法之后，MediaPlayer 并未真正去装载那些音频文件，还需要调用 MediaPlayer 的 prepare()方法去准备音频，所谓“准备”，就是让 MediaPlayer 真正去装载音频文件。

因此使用已有的 MediaPlayer 对象装载“下一首”歌曲的代码模板为：

```
try
{
```

```
        mPlayer.reset();
        // 装载下一首歌曲
        mPlayer.setDataSource("/mnt/sdcard/next.mp3);
        // 准备声音
        mPlayer.prepare();
        // 播放
        mPlayer.start();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
```

除此之外，MediaPlayer 还提供了一些绑定事件监听器的方法，用于监听 MediaPlayer 播放过程中所发生的特定事件。绑定事件监听器的方法如下。

- setOnCompletionListener(MediaPlayer.OnCompletionListener listener)：为 MediaPlayer 的播放完成事件绑定事件监听器。
- setOnErrorListener(MediaPlayer.OnErrorListener listener)：为 MediaPlayer 的播放错误事件绑定事件监听器。
- setOnPreparedListener(MediaPlayer.OnPreparedListener listener)：当 MediaPlayer 调用 prepare()方法时触发该监听器。
- setOnSeekCompleteListener(MediaPlayer.OnSeekCompleteListener listener)：当 MediaPlayer 调用 seek()方法时触发该监听器。

因此可以在创建一个 MediaPlayer 对象之后，通过为该 MediaPlayer 绑定监听器来监听相应的事件。例如如下代码：

```
// 为 MediaPlayer 的播放完成事件绑定事件监听器
mPlayer.setOnErrorListener(new OnErrorListener ()
{
    @Override
    onError(MediaPlayer mp, int what, int extra)
    {
        // 针对错误进行相应的处理
        ...
    }
});
// 为 MediaPlayer 的播放完成事件绑定事件监听器
mPlayer.setOnCompletionListener(new OnCompletionListener()
{
    @Override
    public void onCompletion(MediaPlayer mp)
    {
        current++;
        if (current >= 3)
        {
            current = 0;
        }
        prepareAndPlay(musics[current]);
    }
});
```

下面简单归纳一下使用 MediaPlayer 播放不同来源的音频文件。

1. 播放应用的资源文件

播放应用的资源文件需要两步即可。

① 调用 MediaPlayer 的 create(Context context, int resid)方法装载指定的资源文件。

② 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

例如如下代码：

```
MediaPlayer mPlayer = MediaPlayer.create(this, R.raw.song);
mPlayer.start();
```

提示：

音频资源文件一般放在 Android 应用的/res/raw 目录下。

2. 播放应用的原始资源文件

播放应用的资源文件按如下步骤执行。

① 调用 Context 的 getAssets()方法获取应用的 AssetManager。

② 调用 AssetManager 对象的 openFd(String name)方法打开指定的原始资源，该方法返回一个 AssetFileDescriptor 对象。

③ 调用 AssetFileDescriptor 的 getFileDescriptor()、getStartOffset()和 getLength()方法来获取音频文件的 FileDescriptor、开始位置、长度等。

④ 创建 MediaPlayer 对象（或利用已有的 MediaPlayer 对象），并调用 MediaPlayer 对象的 setDataSource(FileDescriptor fd, long offset, long length)方法来装载音频资源。

⑤ 调用 MediaPlayer 对象的 prepare()方法准备音频。

⑥ 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

注意：

虽然 MediaPlayer 提供了 setDataSource(FileDescriptor fd)方法来装载指定的音频资源，但实际使用时这个方法似乎有问题：不管程序调用 openFd(String name)方法时指定打开哪个原始资源，MediaPlayer 将总是播放第一个原始音频资源。

例如如下代码片段：

```
AssetManager am = getAssets();
// 打开指定音乐文件
AssetFileDescriptor afd = am.openFd(music);
MediaPlayer mPlayer = new MediaPlayer();
// 使用 MediaPlayer 装载指定的声音文件
mPlayer.setDataSource(afd.getFileDescriptor()
    , afd.getStartOffset()
    , afd.getLength());
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

3. 播放外部存储器上的音频文件

播放外部存储器上的音频文件按如下步骤执行。

① 创建 MediaPlayer 对象（或利用已有的 MediaPlayer 对象），并调用 MediaPlayer 对象的 setDateSource(String path)方法装载指定的音频文件。

② 调用 MediaPlayer 对象的 prepare()方法准备音频。

③ 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

例如如下代码：

```
MediaPlayer mPlayer = new MediaPlayer();
```

```
// 使用 MediaPlayer 装载指定的声音文件
mPlayer.setDataSource("/mnt/sdcard/mysong.mp3");
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

4. 播放来自网络的音频文件

播放来自网络的音频文件有两种方式：①直接使用 MediaPlayer 的静态 create(Context context, Uri uri)方法；②调用 MediaPlayer 的 setDataSource(Context context, Uri uri)方法装载指定 Uri 对应的音频文件。

以第二种方式播放来自网络的音频文件的步骤如下。

1. 根据网络上的音频文件所在的位置创建 Uri 对象。
2. 创建 MediaPlayer 对象（或利用已有的 MediaPlayer 对象），并调用 MediaPlayer 对象的 setDateSource(Context context, Uri uri)方法装载 Uri 对应的音频文件。
3. 调用 MediaPlayer 对象的 prepare()方法准备音频。
4. 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

例如如下代码片段：

```
Uri uri = Uri.parse("http://www.crazyit.org/abc.mp3");
MediaPlayer mPlayer = new MediaPlayer();
// 使用 MediaPlayer 根据 Uri 来装载指定的声音文件
mPlayer.setDataSource(this, uri);
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

MediaPlayer 除了调用 prepare()方法来准备声音之外，还可以调用 prepareAsync()来准备声音。prepareAsync()与普通 prepare()方法的区别在于，prepareAsync()是异步的，它不会阻塞当前的 UI 线程。

归纳起来，MediaPlayer 的状态图如图 11.1 所示。

由于前面已经提供了大量使用 MediaPlayer 播放声音的例子，故此处不再介绍使用 MediaPlayer 的简单方法。

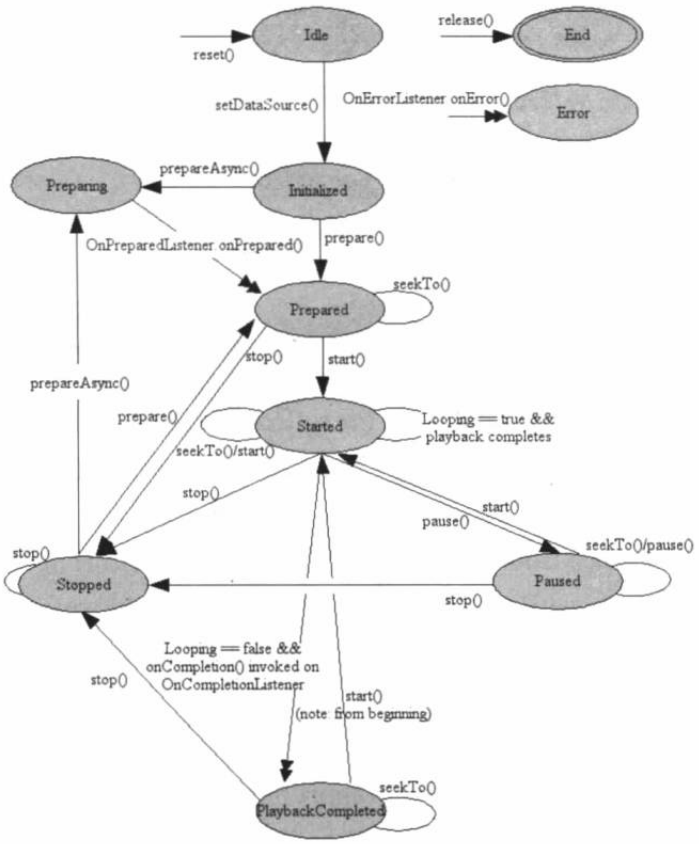


图 11.1 MediaPlayer 的状态图

11.1.2 音乐特效控制

Android 可以控制播放音乐时的均衡器、重低音、音场及显示音乐波形等，这些都是靠 AudioEffect 及其子类来完成的，它包含如下常用子类。

- AcousticEchoCanceler：取消回声控制器。
- AutomaticGainControl：自动增益控制器。
- NoiseSuppressor：噪音压制控制器。
- BassBoost：重低音控制器。
- Equalizer：均衡控制器。
- PresetReverb：预设音场控制器。
- Visualizer：示波器。

上面的子类中前三个子类的用法很简单，只要调用它们的静态 create()方法创建相应的实例，然后调用它们的 isAvailable()方法判断是否可用，再调用 setEnabled(boolean enabled)方法启用相应效果即可。

下面是启用取消回声功能的示意代码：

```
// 获取取消回声控制器
AcousticEchoCanceler canceler = AcousticEchoCanceler.create(0
    , mPlayer.getAudioSessionId())
if(canceler.isAvailable())
{
    // 启用取消回声功能
    canceler.setEnabled(true);
}
```

下面是启用自动增益功能的示意代码：

```
// 获取自动增益控制器
AutomaticGainControl ctrl = AutomaticGainControl.create(0
    , mPlayer.getAudioSessionId())
if(ctrl.isAvailable())
{
    // 启用自动增益功能
    ctrl.setEnabled(true);
}
```

下面是启用噪音压制功能的示意代码：

```
// 获取噪音压制控制器
NoiseSuppressor suppressor = NoiseSuppressor.create(0
    , mPlayer.getAudioSessionId())
if(suppressor.isAvailable())
{
    // 启用噪音压制功能
    suppressor.setEnabled(true);
}
```

BassBoost、Equalizer、PresetReverb、Visualizer 这 4 个类，都需要调用构造器来创建实例。创建实例时，同样需要传入一个 audioSession 参数，为了启用它们，同样需要调用 AudioEffect 基类的 setEnabled(true)方法。

获取 BassBoost 对象之后，可调用它的 setStrength(short strength)方法来设置重低音的强度。

获取 PresetReverb 对象之后，可调用它的 setPreset(short preset)方法设置使用预设置的音场。Equalizer 提供了 getNumberOfPresets()方法获取系统所有预设的音场，并提供了 getPresetName()方法获取预设音场名称。

获取 Equalizer 对象之后，可调用它的 getNumberOfBands()方法获取该均衡器支持的总频率数，再调用 getCenterFreq(short band)方法根据索引来获取频率。当用户想为某个频率的均衡器设置参数值时，可调用 setBandLevel(short band, short level)方法进行设置。

Visualizer 对象并不用于控制音乐播放效果，它只是显示音乐的播放波形。为了实时显示

该示波器的数据，需要为该组件设置一个 OnDataCaptureListener 监听器，该监听器将负责更新波形显示组件的界面。

下面用一个实例来示范上面 4 个特效类的用法。

实例：音乐的示波器、均衡、重低音和音场

本实例无须界面布局文件，使用一个 LinearLayout 容器来盛装一个示波器 View 组件，该示波器 View 组件将负责绘制 Visualizer 传过来的数据；LinearLayout 添加多个 SeekBar 来控制 Equalizer 支持的所有频率的均衡值；LinearLayout 还添加一个 SeekBar 来控制重低音的强度；LinearLayout 还添加一个 Spinner 让用户选择预设音场。

下面是该实例的 Activity 代码。

程序清单：codes\11\11.1\MediaPlayerTest\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义播放声音的 MediaPlayer
    private MediaPlayer mPlayer;
    // 定义系统的示波器
    private Visualizer mVisualizer;
    // 定义系统的均衡器
    private Equalizer mEqualizer;
    // 定义系统的重低音控制器
    private BassBoost mBass;
    // 定义系统的预设音场控制器
    private PresetReverb mPresetReverb;
    private LinearLayout layout;
    private List<Short> reverbNames = new ArrayList<Short>();
    private List<String> reverbVals = new ArrayList<String>();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 设置控制音乐声音
        setVolumeControlStream(AudioManager.STREAM_MUSIC);
        layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        setContentView(layout);
        // 创建 MediaPlayer 对象
        mPlayer = MediaPlayer.create(this, R.raw.beautiful);
        // 初始化示波器
        setupVisualizer();
        // 初始化均衡控制器
        setupEqualizer();
        // 初始化重低音控制器
        setupBassBoost();
        // 初始化预设音场控制器
        setupPresetReverb();
        // 开发播放音乐
        mPlayer.start();
    }
    private void setupVisualizer()
    {
        // 创建 MyVisualizerView 组件，用于显示波形图
        final MyVisualizerView mVisualizerView =
            new MyVisualizerView(this);
        mVisualizerView.setLayoutParams(new ViewGroup.LayoutParams(
            ViewGroup.LayoutParams.MATCH_PARENT,
            (int) (120f * getResources().getDisplayMetrics().density)));
```

```
        // 将MyVisualizerView组件添加到layout容器中
        layout.addView(mVisualizerView);
        // 以MediaPlayer的AudioSessionId创建Visualizer
        // 相当于设置Visualizer负责显示该MediaPlayer的音频数据
        mVisualizer = new Visualizer(mPlayer.getAudioSessionId());
        mVisualizer.setCaptureSize(Visualizer.getCaptureSizeRange()[1]);
        // 为mVisualizer设置监听器
        mVisualizer.setDataCaptureListener(
            new Visualizer.OnDataCaptureListener()
            {
                @Override
                public void onFftDataCapture(Visualizer visualizer,
                    byte[] fft, int samplingRate)
                {
                }
                @Override
                public void onWaveFormDataCapture(Visualizer visualizer,
                    byte[] waveform, int samplingRate)
                {
                    // 用waveform波形数据更新mVisualizerView组件
                    mVisualizerView.updateVisualizer(waveform);
                }
            }, Visualizer.getMaxCaptureRate() / 2, true, false);
        mVisualizer.setEnabled(true);
    }
    // 初始化均衡控制器的方法
    private void setupEqualizer()
    {
        // 以MediaPlayer的AudioSessionId创建Equalizer
        // 相当于设置Equalizer负责控制该MediaPlayer
        mEqualizer = new Equalizer(0, mPlayer.getAudioSessionId());
        // 启用均衡控制效果
        mEqualizer.setEnabled(true);
        TextView eqTitle = new TextView(this);
        eqTitle.setText("均衡器：");
        layout.addView(eqTitle);
        // 获取均衡控制器支持最小值和最大值
        final short minEQLevel = mEqualizer.getBandLevelRange()[0];
        short maxEQLevel = mEqualizer.getBandLevelRange()[1];
        // 获取均衡控制器支持的所有频率
        short brands = mEqualizer.getNumberOfBands();
        for (short i = 0; i < brands; i++)
        {
            TextView eqTextView = new TextView(this);
            // 创建一个TextView，用于显示频率
            eqTextView.setLayoutParams(new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
            eqTextView.setGravity(Gravity.CENTER_HORIZONTAL);
            // 设置该均衡控制器的频率
            eqTextView.setText((mEqualizer.getCenterFreq(i) / 1000)
                + " Hz");
            layout.addView(eqTextView);
            // 创建一个水平排列组件的LinearLayout
            LinearLayout tmpLayout = new LinearLayout(this);
            tmpLayout.setOrientation(LinearLayout.HORIZONTAL);
            // 创建显示均衡控制器最小值的TextView
            TextView minDbTextView = new TextView(this);
            minDbTextView.setLayoutParams(new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
            // 显示均衡控制器的最小值
```

```
        minDbTextView.setText((minEQLevel / 100) + " dB");
        // 创建显示均衡控制器最大值的 TextView
        TextView maxDbTextView = new TextView(this);
        maxDbTextView.setLayoutParams(new ViewGroup.LayoutParams(
            ViewGroup.LayoutParams.WRAP_CONTENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
        // 显示均衡控制器的最大值
        maxDbTextView.setText((maxEQLevel / 100) + " dB");
        LinearLayout.LayoutParams layoutParams = new
            LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT);
        layoutParams.weight = 1;
        // 定义 SeekBar 作为调整工具
        SeekBar bar = new SeekBar(this);
        bar.setLayoutParams(layoutParams);
        bar.setMax(maxEQLevel - minEQLevel);
        bar.setProgress(mEqualizer.getBandLevel(i));
        final short brand = i;
        // 为 SeekBar 的拖动事件设置事件监听器
        bar.setOnSeekBarChangeListener(new SeekBar
            .OnSeekBarChangeListener()
        {
            @Override
            public void onProgressChanged(SeekBar seekBar,
                int progress, boolean fromUser)
            {
                // 设置该频率的均衡值
                mEqualizer.setBandLevel(brand,
                    (short) (progress + minEQLevel));
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar)
            {
            }
            @Override
            public void onStopTrackingTouch(SeekBar seekBar)
            {
            }
        });
        // 使用水平排列组件的 LinearLayout 盛装三个组件
        tmpLayout.addView(minDbTextView);
        tmpLayout.addView(bar);
        tmpLayout.addView(maxDbTextView);
        // 将水平排列组件的 LinearLayout 添加到 myLayout 容器中
        layout.addView(tmpLayout);
    }
}
// 初始化重低音控制器
private void setupBassBoost()
{
    // 以 MediaPlayer 的 AudioSessionId 创建 BassBoost
    // 相当于设置 BassBoost 负责控制该 MediaPlayer
    mBass = new BassBoost(0, mPlayer.getAudioSessionId());
    // 设置启用重低音效果
    mBass.setEnabled(true);
    TextView bbTitle = new TextView(this);
    bbTitle.setText("重低音：");
    layout.addView(bbTitle);
    // 使用 SeekBar 作为重低音的调整工具
    SeekBar bar = new SeekBar(this);
    // 重低音的范围为 0~1000
    bar.setMax(1000);
```

```
        bar.setProgress(0);
        // 为SeekBar的拖动事件设置事件监听器
        bar.setOnSeekBarChangeListener(new SeekBar
            .OnSeekBarChangeListener()
        {
            @Override
            public void onProgressChanged(SeekBar seekBar
                , int progress, boolean fromUser)
            {
                // 设置重低音的强度
                mBass.setStrength((short) progress);
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar)
            {
            }
            @Override
            public void onStopTrackingTouch(SeekBar seekBar)
            {
            }
        });
        layout.addView(bar);
    }
    // 初始化预设音场控制器
    private void setupPresetReverb()
    {
        // 以MediaPlayer的AudioSessionId创建PresetReverb
        // 相当于设置PresetReverb负责控制该MediaPlayer
        mPresetReverb = new PresetReverb(0,
            mPlayer.getAudioSessionId());
        // 设置启用预设音场控制
        mPresetReverb.setEnabled(true);
        TextView prTitle = new TextView(this);
        prTitle.setText("音场");
        layout.addView(prTitle);
        // 获取系统支持的所有预设音场
        for (short i = 0; i < mEqualizer.getNumberOfPresets(); i++)
        {
            reverbNames.add(i);
            reverbVals.add(mEqualizer.getPresetName(i));
        }
        // 使用Spinner作为音场选择工具
        Spinner sp = new Spinner(this);
        sp.setAdapter(new ArrayAdapter<String>(MainActivity.this,
            android.R.layout.simple_spinner_item, reverbVals));
        // 为Spinner的列表项选中事件设置监听器
        sp.setOnItemSelectedListener(new Spinner
            .OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> arg0
                , View arg1, int arg2, long arg3)
            {
                // 设定音场
                mPresetReverb.setPreset(reverbNames.get(arg2));
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
            }
        });
        layout.addView(sp);
    }
```

```
@Override
protected void onPause()
{
    super.onPause();
    if (isFinishing() && mPlayer != null)
    {
        // 释放所有对象
        mVisualizer.release();
        mEqualizer.release();
        mPresetReverb.release();
        mBass.release();
        mPlayer.release();
        mPlayer = null;
    }
}
private static class MyVisualizerView extends View
{
    // bytes 数组保存了波形抽样点的值
    private byte[] bytes;
    private float[] points;
    private Paint paint = new Paint();
    private Rect rect = new Rect();
    private byte type = 0;
    public MyVisualizerView(Context context)
    {
        super(context);
        bytes = null;
        // 设置画笔的属性
        paint.setStrokeWidth(1f);
        paint.setAntiAlias(true);
        paint.setColor(Color.GREEN);
        paint.setStyle(Style.FILL);
    }
    public void updateVisualizer(byte[] ftt)
    {
        bytes = ftt;
        // 通知该组件重绘自己
        invalidate();
    }

    @Override
    public boolean onTouchEvent(MotionEvent me)
    {
        // 当用户触碰该组件时，切换波形类型
        if(me.getAction() != MotionEvent.ACTION_DOWN)
        {
            return false;
        }
        type ++;
        if(type >= 3)
        {
            type = 0;
        }
        return true;
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        if (bytes == null)
        {
            return;
        }
        // 绘制白色背景（主要为了印刷时好看）
```

```
            canvas.drawColor(Color.WHITE);
            // 使用 rect 对象记录该组件的宽度和高度
            rect.set(0,0,getWidth(),getHeight());
            switch(type)
            {
                // -------绘制块状的波形图-------
                case 0:
                    for (int i = 0; i < bytes.length - 1; i++)
                    {
                        float left = getWidth() * i / (bytes.length - 1);
                        // 根据波形值计算该矩形的高度
                        float top = rect.height()-(byte)(bytes[i+1]+128)
                            * rect.height() / 128;
                        float right = left + 1;
                        float bottom = rect.height();
                        canvas.drawRect(left, top, right, bottom, paint);
                    }
                    break;
                // -------绘制柱状的波形图（每隔 18 个抽样点绘制一个矩形）-------
                case 1:
                    for (int i = 0; i < bytes.length - 1; i += 18)
                    {
                        float left = rect.width()*i/(bytes.length - 1);
                        // 根据波形值计算该矩形的高度
                        float top = rect.height()-(byte)(bytes[i+1]+128)
                            * rect.height() / 128;
                        float right = left + 6;
                        float bottom = rect.height();
                        canvas.drawRect(left, top, right, bottom, paint);
                    }
                    break;
                // -------绘制曲线波形图--------
                case 2:
                    // 如果 points 数组还未初始化
                    if (points == null || points.length < bytes.length * 4)
                    {
                        points = new float[bytes.length * 4];
                    }
                    for (int i = 0; i < bytes.length - 1; i++)
                    {
                        // 计算第 i 个点的 x 坐标
                        points[i * 4] = rect.width()*i/(bytes.length - 1);
                        // 根据 bytes[i] 的值（波形点的值）计算第 i 个点的 y 坐标
                        points[i * 4 + 1] = (rect.height() / 2)
                            + ((byte) (bytes[i] + 128)) * 128
                            / (rect.height() / 2);
                        // 计算第 i+1 个点的 x 坐标
                        points[i * 4 + 2] = rect.width() * (i + 1)
                            / (bytes.length - 1);
                        // 根据 bytes[i+1] 的值（波形点的值）计算第 i+1 个点的 y 坐标
                        points[i * 4 + 3] = (rect.height() / 2)
                            + ((byte) (bytes[i + 1] + 128)) * 128
                            / (rect.height() / 2);
                    }
                    // 绘制波形曲线
                    canvas.drawLines(points, paint);
                    break;
            }
        }
    }
}
```

上面的程序中定义了 4 个方法：setupVisualizer()、setupEqualizer()、setupBassBoost()和

setupPresetReverb()，这 4 个方法分别创建 Visualizer、Equalizer、BassBoost 和 PresetReverb 对象，并调用它们的方法控制音乐特效。

上面的实例中包含了一个 MyVisualizerView 内部类，该内部类会根据 Visualizer 传过来的数据动态绘制波形效果。本例提供了三种波形：块状波形、柱状波形和曲线波形——当用户单击 MyVisualizerView 组件时将会切换波形。

运行该实例，可以看到如图 11.2 所示的效果。

如果用户单击该程序界面最上方的示波器组件，该组件将会切换一种波形显示。单击下方的音场，将可以看到系统支持的所有预设音场，如图 11.3 所示。

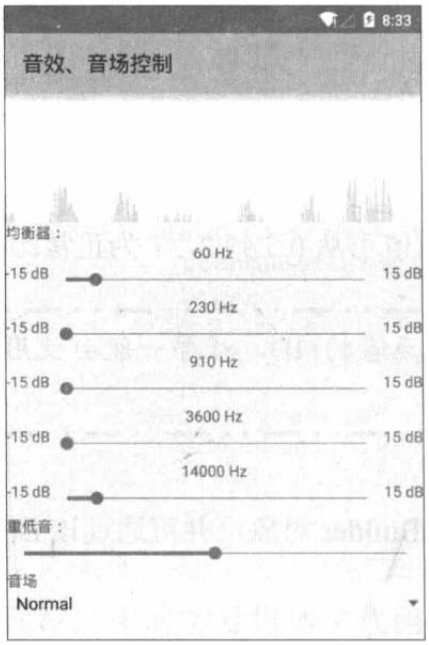


图 11.2 示波器、均衡、重低音和音场

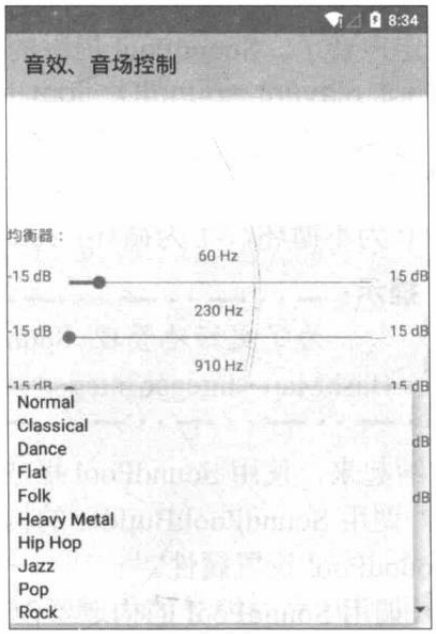


图 11.3 曲线示波器和全部预设音场

11.1.3 使用 SoundPool 播放音效

如果应用程序经常需要播放密集、短促的音效，这时还用 MediaPlayer 就显得有些不合适了。MediaPlayer 存在如下缺点：

- 资源占用量较高，延迟时间较长。
- 不支持多个音频同时播放。

除了前面介绍的使用 MediaPlayer 播放音频之外，Android 还提供了 SoundPool 来播放音效，SoundPool 使用音效池的概念来管理多个短促的音效，例如它可以开始就加载 20 个音效，以后在程序中按音效的 ID 进行播放。

SoundPool 主要用于播放一些较短的声音片段，与 MediaPlayer 相比，SoundPool 的优势在于 CPU 资源占用量低和反应延迟小。另外，SoundPool 还支持自行设置声音的品质、音量、播放比率等参数。

Android 系统 SoundPool 提供了一个 Builder 内部类，该内部类专门用于创建 SoundPool。

提示：

从 Android 5.0 开始，SoundPool 的构造器被设为过时了，因此推荐使用 SoundPool.Builder 来创建 SoundPool 对象。

一旦得到了 SoundPool 对象之后，接下来就可调用 SoundPool 的多个重载的 load()方法来

加载声音了。SoundPool 提供了如下 4 个 load()方法。

- int load(Context context, int resId, int priority)：从 resId 所对应的资源加载声音。
- int load(FileDescriptor fd, long offset, long length, int priority)：加载 fd 所对应的文件中从 offset 开始、长度为 length 的声音。
- int load(AssetFileDescriptor afd, int priority)：从 afd 所对应的文件中加载声音。
- int load(String path, int priority)：从 path 对应的文件去加载声音。

上面 4 个方法中都有一个 priority 参数，该参数目前还没有任何作用，Android 建议将该参数设为 1，保持和未来的兼容性。

上面 4 个方法加载声音之后，都会返回该声音的 ID，以后程序就可以通过该声音的 ID 来播放指定声音了。SoundPool 提供的播放指定声音的方法如下。

- int play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)：该方法的第一个参数指定播放哪个声音；leftVolume、rightVolume 指定左、右的音量；priority 指定播放声音的优先级，数值越大，优先级越高；loop 指定是否循环，0 为不循环，-1 为循环；rate 指定播放的比率，数值可从 0.5 到 2，1 为正常比率。

提示：

为了更好地管理 SoundPool 所加载的每个声音的 ID，程序一般会使用一个 HashMap <Integer,Integer>对象来管理声音。

归纳起来，使用 SoundPool 播放声音的步骤如下。

① 调用 SoundPool.Builder 的构造器创建 SoundPool.Builder 对象，并可通过该 Builder 对象为 SoundPool 设置属性。

② 调用 SoundPool 的构造器创建 SoundPool 对象。

③ 调用 SoundPool 对象的 load()方法从指定资源、文件中加载声音。最好使用 HashMap<Integer,Integer>来管理所加载的声音。

④ 调用 SoundPool 的 play()方法播放声音。

下面的程序示范了如何使用 SoundPool 来播放音效。该程序提供了三个按钮，分别用于播放不同的声音。该程序的界面十分简单，故此处不再给出界面布局代码。程序代码如下。

程序清单：codes\11\11.1\SoundPoolTest\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity implements OnClickListener
{
    Button bomb, shot, arrow;
    // 定义一个 SoundPool
    SoundPool soundPool;
    HashMap<Integer, Integer> soundMap = new HashMap<>();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bomb = (Button) findViewById(R.id.bomb);
        shot = (Button) findViewById(R.id.shot);
        arrow = (Button) findViewById(R.id.arrow);
        AudioAttributes attr = new AudioAttributes.Builder()
            .setUsage(AudioAttributes.USAGE_GAME) // 设置音效使用场景
            // 设置音效的类型
            .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
            .build();
        soundPool = new SoundPool.Builder()
```

```
            .setAudioAttributes(attr) // 设置音效池的属性
            .setMaxStreams(10) // 设置最多可容纳 10 个音频流
            .build(); // ①
        // load 方法加载指定音频文件，并返回所加载的音频 ID
        // 此处使用 HashMap 来管理这些音频流
        soundMap.put(1, soundPool.load(this, R.raw.bomb, 1)); // ②
        soundMap.put(2, soundPool.load(this, R.raw.shot, 1));
        soundMap.put(3, soundPool.load(this, R.raw.arrow, 1));
        bomb.setOnClickListener(this);
        shot.setOnClickListener(this);
        arrow.setOnClickListener(this);
    }
    // 重写 OnClickListener 监听器接口的方法
    @Override
    public void onClick(View source)
    {
        // 判断哪个按钮被单击
        switch (source.getId())
        {
            case R.id.bomb:
                soundPool.play(soundMap.get(1), 1, 1, 0, 0, 1); // ③
                break;
            case R.id.shot:
                soundPool.play(soundMap.get(2), 1, 1, 0, 0, 1);
                break;
            case R.id.arrow:
                soundPool.play(soundMap.get(3), 1, 1, 0, 0, 1);
                break;
        }
    }
}
```

上面程序中的①号粗体字代码用于创建 SoundPool 对象；②号粗体字代码用于使用 SoundPool 加载多个不同的声音；③号粗体字代码则用于根据声音 ID 来播放指定声音。这就是使用 SoundPool 播放声音的标准过程。

实际使用 SoundPool 播放声音时有如下几点需要注意：SoundPool 虽然可以一次性加载多个声音，但由于内存限制，因此应该避免使用 SoundPool 来播放歌曲或者做游戏背景音乐，只有那些短促、密集的声音才考虑使用 SoundPool 进行播放。

虽然 SoundPool 比 MediaPlayer 的效率好，但也不是绝对不存在延迟问题，尤其在那些性能不太好的手机中，SoundPool 的延迟问题会更严重。

➤➤ 11.1.4 使用 VideoView 播放视频

为了在 Android 应用中播放视频，Android 提供了 VideoView 组件，它就是一个位于 android.widget 包下的组件，它的作用与 ImageView 类似，只是 ImageView 用于显示图片，而 VideoView 用于播放视频。

使用 VideoView 播放视频的步骤如下。

1. 在界面布局文件中定义 VideoView 组件，或在程序中创建 VideoView 组件。
2. 调用 VideoView 的如下两个方法来加载指定视频。
   - setVideoPath(String path)：加载 path 文件所代表的视频。
   - setVideoURI(Uri uri)：加载 uri 所对应的视频。
3. 调用 VideoView 的 start()、stop()、pause()方法来控制视频播放。

实际上与 VideoView 一起结合使用的还有一个 MediaController 类，它的作用是提供一个友好的图形控制界面，通过该控制界面来控制视频的播放。

下面的程序示范了如何使用 VideoView 来播放视频。该程序提供了一个简单的界面，界面布局代码如下。

程序清单：codes\11\11.1\VideoViewTest\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<!-- 定义 VideoView 播放视频 -->
<VideoView
    android:id="@+id/video"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</LinearLayout>
```

上面的界面布局中定义了一个 VideoView 组件，接下来就可以在程序中使用该组件来播放视频了。播放视频时还结合了 MediaController 来控制视频的播放，该程序代码如下。

程序清单：codes\11\11.1\VideoViewTest\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity
{
    VideoView videoView;
    MediaController mController;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        getWindow().setFormat(PixelFormat.TRANSLUCENT);
        setContentView(R.layout.main);
        // 获取界面上 VideoView 组件
        videoView = (VideoView) findViewById(R.id.video);
        // 创建 MediaController 对象
        mController = new MediaController(this);
        File video = new File("/mnt/sdcard/movie.mp4");
        if(video.exists())
        {
            videoView.setVideoPath(video.getAbsolutePath()); // ①
            // 设置 videoView 与 mController 建立关联
            videoView.setMediaController(mController); // ②
            // 设置 mController 与 videoView 建立关联
            mController.setMediaPlayer(videoView); // ③
            // 让 VideoView 获取焦点
            videoView.requestFocus();
        }
    }
}
```

上面程序中的①号粗体字代码用于让 VideoView 加载指定的视频文件，接下来该 VideoView 就可以用于播放该视频了；接下来②、③号粗体字代码用于建立 VideoView 与 MediaController 之间的关联，这样就不需要开发者自己去控制视频的播放、暂停等了，让 MediaController 进行控制即可。

由于该程序需要读取 SD 卡上的视频文件，因此还需要在 AndroidManifest.xml 文件中增加如下代码片段：

```
<!-- 授予该程序读取外部存储器的权限 -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

运行该程序，在保证/mnt/sdcard/movie.mp4 视频文件存在的前提下，将可以看到如图 11.4

所示的界面。

图 11.4 使用 VedioPlayer 播放视频

图 11.4 所示界面中的快进键、暂停键、后退键以及播放进度条就是由 MediaController 所提供的。

提示：

运行该程序可能会遇到一些问题，如果读者使用了一些非标准的 MP4、3GP 文件，那么该应用程序将无法播放，因此建议读者自行使用手机录制一段兼容各种手机的标准的 MP4、3GP 视频文件。

11.1.5 使用 MediaPlayer 和 SurfaceView 播放视频

使用 VideoView 播放视频简单、方便，但有些早期的开发者还是更喜欢使用 MediaPlayer 来播放视频，但由于 MediaPlayer 主要用于播放音频，因此它没有提供图像输出界面，此时就需要借助于 SurfaceView 来显示 MediaPlayer 播放的图像输出。

使用 MediaPlayer 播放视频的步骤如下。

1. 创建 MediaPlayer 对象，并让它加载指定的视频文件。
2. 在界面布局文件中定义 SurfaceView 组件，或在程序中创建 SurfaceView 组件，并为 SurfaceView 的 SurfaceHolder 添加 Callback 监听器。
3. 调用 MediaPlayer 对象的 setDisplay(SurfaceHolder sh)方法将所播放的视频图像输出到指定的 SurfaceView 组件。
4. 调用 MediaPlayer 对象的 start()、stop()和 pause()方法控制视频的播放。

下面的程序使用了 MediaPlayer 和 SurfaceView 来播放视频，并为该程序提供了三个按钮来控制视频的播放、暂停和停止。该程序的界面布局比较简单，故此处不再给出界面布局文件。该程序的代码如下。

程序清单：codes\11\11.1\SurfaceViewPlayVideo\app\src\main\java\org\crazyit\media\MainActivity. java

```
public class MainActivity extends Activity
    implements OnClickListener
{
    SurfaceView surfaceView;
    ImageButton play, pause, stop;
    MediaPlayer mPlayer;
    // 记录当前视频的播放位置
    int position;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面中的三个按钮
        play = (ImageButton) findViewById(R.id.play);
        pause = (ImageButton) findViewById(R.id.pause);
        stop = (ImageButton) findViewById(R.id.stop);
        // 为三个按钮的单击事件绑定事件监听器
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        stop.setOnClickListener(this);
        // 创建 MediaPlayer
        mPlayer = new MediaPlayer();
        surfaceView = (SurfaceView) this.findViewById(R.id.surfaceView);
        // 设置播放时打开屏幕
        surfaceView.getHolder().setKeepScreenOn(true);
        surfaceView.getHolder().addCallback(new SurfaceListener());
    }
    @Override
    public void onClick(View source)
    {
        try
        {
            switch (source.getId())
            {
                // 播放按钮被单击
                case R.id.play:
                    play();
                    break;
                // 暂停按钮被单击
                case R.id.pause:
                    if (mPlayer.isPlaying())
                    {
                        mPlayer.pause();
                    }
                    else
                    {
                        mPlayer.start();
                    }
                    break;
                // 停止按钮被单击
                case R.id.stop:
                    if (mPlayer.isPlaying()) mPlayer.stop();
                    break;
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    private void play() throws IOException
    {
        mPlayer.reset();
        mPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
        // 设置需要播放的视频
        mPlayer.setDataSource("/mnt/sdcard/movie.3gp");
        // 把视频画面输出到 SurfaceView
        mPlayer.setDisplay(surfaceView.getHolder()); // ①
        mPlayer.prepare();
        // 获取窗口管理器
        WindowManager wManager = getWindowManager();
        DisplayMetrics metrics = new DisplayMetrics();
        // 获取屏幕大小
```

```
        wManager.getDefaultDisplay().getMetrics(metrics);
        // 设置视频保持纵横比缩放到占满整个屏幕
        surfaceView.setLayoutParams(new LayoutParams(metrics.widthPixels
            , mPlayer.getVideoHeight() * metrics.widthPixels
            / mPlayer.getVideoWidth()));
        mPlayer.start();
    }
    private class SurfaceListener implements SurfaceHolder.Callback
    {
        @Override
        public void surfaceChanged(SurfaceHolder holder, int format,
                                   int width, int height)
        {
        }
        @Override
        public void surfaceCreated(SurfaceHolder holder)
        {
            if (position > 0)
            {
                try
                {
                    // 开始播放
                    play();
                    // 并直接从指定位置开始播放
                    mPlayer.seekTo(position);
                    position = 0;
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        }
        @Override
        public void surfaceDestroyed(SurfaceHolder holder)
        {
        }
    }
    // 当其他 Activity 被打开时，暂停播放
    @Override
    protected void onPause()
    {
        if (mPlayer.isPlaying())
        {
            // 保存当前的播放位置
            position = mPlayer.getCurrentPosition();
            mPlayer.stop();
        }
        super.onPause();
    }
    @Override
    protected void onDestroy()
    {
        // 停止播放
        if (mPlayer.isPlaying()) mPlayer.stop();
        // 释放资源
        mPlayer.release();
        super.onDestroy();
    }
}
```

从上面的代码不难看出，使用 MediaPlayer 播放视频与播放音频的步骤大同小异，关键的区别在于①号粗体字代码，设置使用 SurfaceView 来显示 MediaPlayer 播放时的图像输出。当

然，由于程序需要使用 SurfaceView 来显示 MediaPlayer 的图像输出，因此程序需要一些代码来维护 SurfaceView、SurfaceHolder 对象。

该程序同样需要访问 SD 卡上的视频文件，因此也需要在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 授予该程序读取外部存储器的权限 -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

运行上面的程序，将可以看到如图 11.5 所示的播放界面。

从上面的开发过程不难看出，使用 MediaPlayer 播放视频要复杂一些，而且需要自己开发控制按钮来控制视频播放，因此一般推荐使用 VideoView 来播放视频。

图 11.5 使用 MediaPlayer 和 SurfaceView 播放视频

11.2 使用 MediaRecorder 录制音频

手机一般都提供了麦克风硬件，而 Android 系统就可以利用该硬件来录制音频了。

为了在 Android 应用中录制音频，Android 提供了 MediaRecorder 类。使用 MediaRecorder 录制音频的过程很简单，按如下步骤进行即可。

1. 创建 MediaRecorder 对象。
2. 调用 MediaRecorder 对象的 setAudioSource()方法设置声音来源，一般传入 MediaRecorder. AudioSource.MIC 参数指定录制来自麦克风的声音。
3. 调用 MediaRecorder 对象的 setOutputFormat()方法设置所录制的音频文件格式。
4. 调用 MediaRecorder 对象的 setAudioEncoder()、setAudioEncodingBitRate(int bitRate)、setAudioSamplingRate(int samplingRate)方法设置所录制的声音编码格式、编码位率、采样率等，这些参数将可以控制所录制的声音品质、文件大小。一般来说，声音品质越好，声音文件越大。
5. 调用 MediaRecorder 的 setOutputFile(String path)方法设置所录制的音频文件的保存位置。
6. 调用 MediaRecorder 的 prepare()方法准备录制。
7. 调用 MediaRecorder 对象的 start()方法开始录制。
8. 录制完成，调用 MediaRecorder 对象的 stop()方法停止录制，并调用 release()方法释放资源。

注意：

上面的步骤中第 3 步和第 4 步两个步骤千万不能搞反；否则程序将会抛出 IllegalStateException 异常。

图 11.6 显示了 MediaRecorder 的状态图。

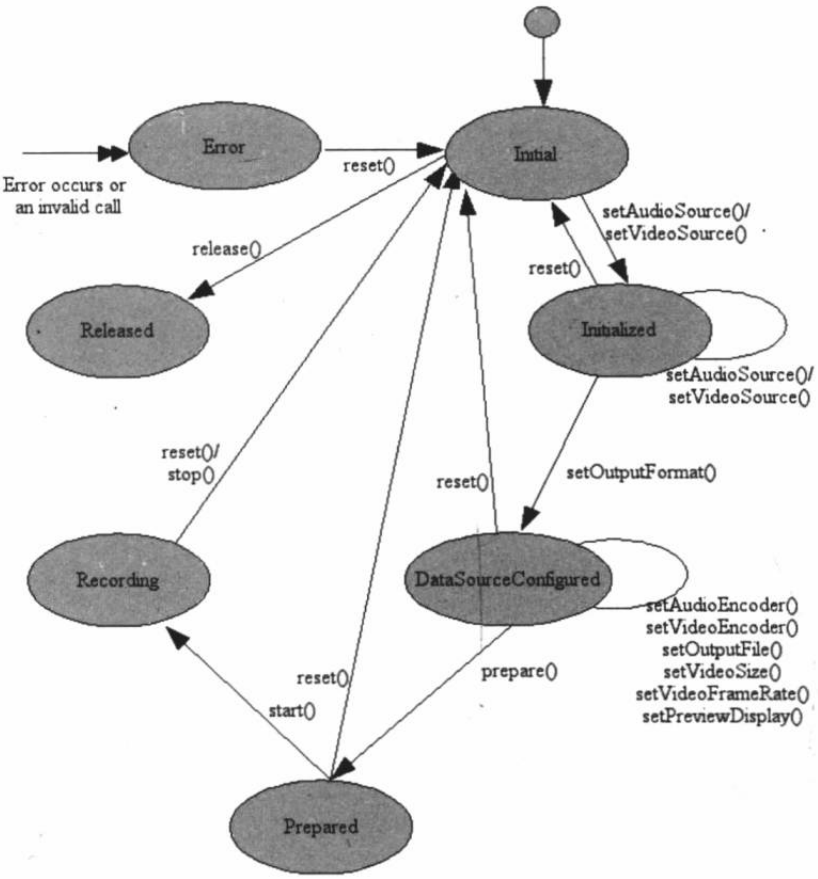


图 11.6 MediaRecorder 的状态图

实例：录制音乐

本实例程序示范了如何使用 MediaRecorder 来录制声音。该程序的界面布局很简单，只提供了两个简单的按钮来控制录音开始、停止，故此处不再给出界面布局文件。程序代码如下。

程序清单：codes\11\11.2\RecordSound\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity
    implements OnClickListener
{
    // 定义界面上的两个按钮
    ImageButton record, stop;
    // 系统的音频文件
    File soundFile;
    MediaRecorder mRecorder;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的两个按钮
        record = (ImageButton) findViewById(R.id.record);
        stop = (ImageButton) findViewById(R.id.stop);
        // 为两个按钮的单击事件绑定监听器
        record.setOnClickListener(this);
        stop.setOnClickListener(this);
    }
    @Override
    public void onDestroy()
    {
        if (soundFile != null && soundFile.exists())
        {
```

```
            // 停止录音
            mRecorder.stop();
            // 释放资源
            mRecorder.release();
            mRecorder = null;
        }
        super.onDestroy();
    }
    @Override
    public void onClick(View source)
    {
        switch (source.getId())
        {
        // 单击录音按钮
            case R.id.record:
                if (!Environment.getExternalStorageState().equals(
                    android.os.Environment.MEDIA_MOUNTED))
                {
                    Toast.makeText(MainActivity.this, "SD卡不存在，请插入SD卡！",
                        Toast.LENGTH_SHORT).show();
                    return;
                }
                try
                {
                    // 创建保存录音的音频文件
                    soundFile = new File(Environment
                        .getExternalStorageDirectory().getCanonicalFile()
                        + "/sound.amr");
                    mRecorder = new MediaRecorder();
                    // 设置录音的声音来源
                    mRecorder.setAudioSource(MediaRecorder
                        .AudioSource.MIC);
                    // 设置录制的声音的输出格式（必须在设置声音编码格式之前设置）
                    mRecorder.setOutputFormat(MediaRecorder
                        .OutputFormat.THREE_GPP);
                    // 设置声音编码格式
                    mRecorder.setAudioEncoder(MediaRecorder
                        .AudioEncoder.AMR_NB);
                    mRecorder.setOutputFile(soundFile.getAbsolutePath());
                    mRecorder.prepare();
                    // 开始录音
                    mRecorder.start(); // ①
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                break;
            // 单击停止按钮
            case R.id.stop:
                if (soundFile != null && soundFile.exists())
                {
                    // 停止录音
                    mRecorder.stop(); // ②
                    // 释放资源
                    mRecorder.release(); // ③
                    mRecorder = null;
                }
                break;
        }
    }
}
```

上面的程序中大段的粗体字代码用于设置录音的相关参数，比如输出文件的格式、声音来源等。程序中①号粗体字代码控制 MediaRecorder 开始录音；当用户单击“停止录制”按钮时，程序中②号粗体字代码控制 MediaRecorder 停止录音，③号粗体字代码用于释放资源。

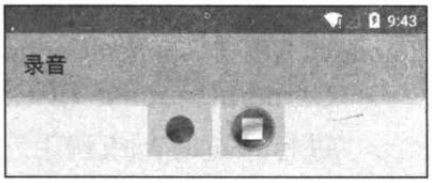


图 11.7 录音界面

运行该程序，将看到如图 11.7 所示的界面。

单击图 11.7 所示界面中第一个按钮开始录音，单击第二个按钮则可结束录音。录音完成后可以看到/mnt/sdcard/目录下生成一个 sound.amr 文件，这就是刚刚录制的音频文件——Android 模拟器将会直接使用宿主电脑上的麦克风，因此如果读者的电脑上有麦克风，那么该程序即可正常录制声音。

上面的程序需要使用系统的麦克风进行录音，因此需要向该程序授予录音的权限。除此之外，还需要授予该程序向外部存储设备写入数据的权限，也就是在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 授予该程序录制声音的权限 -->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<!-- 授予该程序向外部存储器写入数据的权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

注意：

Android 官方 API 已经指出，最新版的 MediaRecorder 目前不能在模拟器上运行，必须在真机上运行，因此读者需要在真机上测试该应用。

11.3 控制摄像头拍照

现在的手机一般都会提供相机功能，有些相机的镜头甚至支持 1000 万以上像素，有些甚至支持光学变焦，这些手机已经变成了专业数码相机。为了充分利用手机上的相机功能，Android 应用可以控制拍照和录制视频。

11.3.1 使用 Android 5.0 的 Camera v2 拍照

Android 5.0 对拍照 API 进行了全新的设计，新增了全新设计的 Camera v2 API，这些 API 不仅大幅提高了 Android 系统拍照的功能，还能支持 RAW 照片输出，甚至允许程序调整相机的对焦模式、曝光模式、快门等。

Android 5.0 的 Camera v2 主要涉及如下 API。

- CameraManager：摄像头管理器。这是一个全新的系统管理器，专门用于检测系统摄像头、打开系统摄像头。除此之外，调用 CameraManager 的 getCameraCharacteristics(String) 方法即可获取指定摄像头的相关特性。
- CameraCharacteristics：摄像头特性。该对象通过 CameraManager 来获取，用于描述特定摄像头所支持的各种特性。
- CameraDevice：代表系统摄像头。该类的功能类似于早期的 Camera 类。
- CameraCaptureSession：这是一个非常重要的 API，当程序需要预览、拍照时，都需要先通过该类的实例创建 Session。而且不管预览还是拍照，也都是由该对象的方法进行控制的，其中控制预览的方法为 setRepeatingRequest()；控制拍照的方法为 capture()。

为了监听 CameraCaptureSession 的创建过程，以及监听 CameraCaptureSession 的拍照过程，Camera v2 API 为 CameraCaptureSession 提供了 StateCallback、CaptureCallback 等内部类。

- **CameraRequest 和 CameraRequest.Builder**：当程序调用 setRepeatingRequest()方法进行预览时，或调用 capture()方法进行拍照时，都需要传入 CameraRequest 参数。CameraRequest 代表了一次捕获请求，用于描述捕获图片的各种参数设置，比如对焦模式、曝光模式……总之，程序需要对照片所做的各种控制，都通过 CameraRequest 参数进行设置。CameraRequest.Builder 则负责生成 CameraRequest 对象。

理解了上面 API 的功能和作用之后，接下来即可使用 Camera v2 API 来控制摄像头拍照了。控制拍照的步骤大致如下。

① 调用 CameraManager 的 openCamera(String cameraId, CameraDevice.StateCallback callback, Handler handler)方法打开指定摄像头。该方法的第一个参数代表要打开的摄像头 ID；第二个参数用于监听摄像头的状态；第三个参数代表执行 callback 的 Handler，如果程序希望直接在当前线程中执行 callback，则可将 handler 参数设为 null。

② 当摄像头被打开之后，程序即可获取 CameraDevice——即根据摄像头 ID 获取了指定摄像头设备，然后调用 CameraDevice 的 createCaptureSession(List<Surface> outputs, CameraCaptureSession.StateCallback callback, Handler handler)方法来创建 CameraCaptureSession。该方法的第一个参数是一个 List 集合，封装了所有需要从该摄像头获取图片的 Surface，第二个参数用于监听 CameraCaptureSession 的创建过程；第三个参数代表执行 callback 的 Handler，如果程序希望直接在当前线程中执行 callback，则可将 handler 参数设为 null。

③ 不管预览还是拍照，程序都调用 CameraDevice 的 createCaptureRequest(int templateType)方法创建 CaptureRequest.Builder，该方法支持 TEMPLATE_PREVIEW（预览）、TEMPLATE_RECORD（拍摄视频）、TEMPLATE_STILL_CAPTURE（拍照）等参数。

④ 通过第 3 步所调用方法返回的 CaptureRequest.Builder 设置拍照的各种参数，比如对焦模式、曝光模式等。

⑤ 调用 CaptureRequest.Builder 的 build()方法即可得到 CaptureRequest 对象，接下来程序可通过 CameraCaptureSession 的 setRepeatingRequest()方法开始预览，或调用 capture()方法拍照。

实例：拍照时自动对焦

本实例示范了使用 Camera v2 来进行拍照。当用户按下拍照键时，该应用会自动对焦，当对焦成功时拍下照片。该程序的界面中提供了一个自定义 TextureView 来显示预览取景，十分简单。该自定义 TextureView 类的代码如下。

程序清单：codes\11\11.3\CameraV2Test\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class AutoFitTextureView extends TextureView
{
    private int mRatioWidth = 0;
    private int mRatioHeight = 0;
    public AutoFitTextureView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
    }
    public void setAspectRatio(int width, int height)
    {
        mRatioWidth = width;
        mRatioHeight = height;
        requestLayout();
    }
```

```
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
    {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
        int width = MeasureSpec.getSize(widthMeasureSpec);
        int height = MeasureSpec.getSize(heightMeasureSpec);
        if (0 == mRatioWidth || 0 == mRatioHeight)
        {
            setMeasuredDimension(width, height);
        }
        else
        {
            if (width < height * mRatioWidth / mRatioHeight)
            {
                setMeasuredDimension(width, width * mRatioHeight / mRatioWidth);
            }
            else
            {
                setMeasuredDimension(height * mRatioWidth / mRatioHeight, height);
            }
        }
    }
}
```

接下来的 MainActivity 将会使用 CameraManager 来打开 CameraDevice，并通过 CameraDevice 创建 CameraCaptureSession，然后即可通过 CameraCaptureSession 进行预览或拍照了。该 Activity 的代码如下。

程序清单：codes\11\11.3\CameraV2Test\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity implements View.OnClickListener
{
    private static final SparseIntArray ORIENTATIONS = new SparseIntArray();
    static
    {
        ORIENTATIONS.append(Surface.ROTATION_0, 90);
        ORIENTATIONS.append(Surface.ROTATION_90, 0);
        ORIENTATIONS.append(Surface.ROTATION_180, 270);
        ORIENTATIONS.append(Surface.ROTATION_270, 180);
    }
    private AutoFitTextureView textureView;
    // 摄像头 ID（通常 0 代表后置摄像头，1 代表前置摄像头）
    private String mCameraId = "0";
    // 定义代表摄像头的成员变量
    private CameraDevice cameraDevice;
    // 预览尺寸
    private Size previewSize;
    private CaptureRequest.Builder previewRequestBuilder;
    // 定义用于预览照片的捕获请求
    private CaptureRequest previewRequest;
    // 定义 CameraCaptureSession 成员变量
    private CameraCaptureSession captureSession;
    private ImageReader imageReader;
    private final TextureView.SurfaceTextureListener mSurfaceTextureListener
        = new TextureView.SurfaceTextureListener()
    {
        @Override
        public void onSurfaceTextureAvailable(SurfaceTexture texture
            , int width, int height)
        {
            // 当 TextureView 可用时，打开摄像头
            openCamera(width, height);
        }
        @Override
```

```
        public void onSurfaceTextureSizeChanged(SurfaceTexture texture
            , int width, int height){ }
        @Override
        public boolean onSurfaceTextureDestroyed(SurfaceTexture texture) { return true; }
        @Override
        public void onSurfaceTextureUpdated(SurfaceTexture texture){}
    };
    private final CameraDevice.StateCallback stateCallback = new CameraDevice.
StateCallback()
    {
        // 摄像头被打开时激发该方法
        @Override
        public void onOpened(CameraDevice cameraDevice)
        {
            MainActivity.this.cameraDevice = cameraDevice;
            // 开始预览
            createCameraPreviewSession(); // ②
        }
        // 摄像头断开连接时激发该方法
        @Override
        public void onDisconnected(CameraDevice cameraDevice)
        {
            cameraDevice.close();
            MainActivity.this.cameraDevice = null;
        }
        // 打开摄像头出现错误时激发该方法
        @Override
        public void onError(CameraDevice cameraDevice, int error)
        {
            cameraDevice.close();
            MainActivity.this.cameraDevice = null;
            MainActivity.this.finish();
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textureView = (AutoFitTextureView) findViewById(R.id.texture);
        // 为该组件设置监听器
        textureView.setSurfaceTextureListener(mSurfaceTextureListener);
        findViewById(R.id.capture).setOnClickListener(this);
    }
    @Override
    public void onClick(View view)
    {
        captureStillPicture();
    }
    private void captureStillPicture()
    {
        try
        {
            if (cameraDevice == null)
            {
                return;
            }
            // 创建作为拍照的 CaptureRequest.Builder
            final CaptureRequest.Builder captureRequestBuilder = cameraDevice
                .createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE);
            // 将 imageReader 的 surface 作为 CaptureRequest.Builder 的目标
            captureRequestBuilder.addTarget(imageReader.getSurface());
            // 设置自动对焦模式
            captureRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
```

```
                CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
            // 设置自动曝光模式
            captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
                CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);
            // 获取设备方向
            int rotation = getWindowManager().getDefaultDisplay().getRotation();
            // 根据设备方向计算设置照片的方向
            captureRequestBuilder.set(CaptureRequest.JPEG_ORIENTATION
                , ORIENTATIONS.get(rotation));
            // 停止连续取景
            captureSession.stopRepeating();
            // 捕获静态图像
            captureSession.capture(captureRequestBuilder.build()
                , new CameraCaptureSession.CaptureCallback()  // ⑤
            {
                // 拍照完成时激发该方法
                @Override
                public void onCaptureCompleted(CameraCaptureSession session
                        , CaptureRequest request, TotalCaptureResult result)
                {
                    try
                    {
                        // 重设自动对焦模式
                        previewRequestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
                            CameraMetadata.CONTROL_AF_TRIGGER_CANCEL);
                        // 设置自动曝光模式
                        previewRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
                            CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);
                        // 打开连续取景模式
                        captureSession.setRepeatingRequest(previewRequest, null, null);
                    }
                    catch (CameraAccessException e)
                    {
                        e.printStackTrace();
                    }
                }
            }, null);
        }
        catch (CameraAccessException e)
        {
            e.printStackTrace();
        }
    }
    // 打开摄像头
    private void openCamera(int width, int height)
    {
        setUpCameraOutputs(width, height);
        CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_
SERVICE);
        try
        {
            // 打开摄像头
            manager.openCamera(mCameraId, stateCallback, null); // ①
        }
        catch (CameraAccessException e)
        {
            e.printStackTrace();
        }
    }
    private void createCameraPreviewSession()
    {
        try
        {
```

```
            SurfaceTexture texture = textureView.getSurfaceTexture();
            texture.setDefaultBufferSize(previewSize.getWidth(), previewSize.getHeight());
            // 创建作为预览的 CaptureRequest.Builder
            previewRequestBuilder = cameraDevice
                .createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
            // 将 textureView 的 surface 作为 CaptureRequest.Builder 的目标
            previewRequestBuilder.addTarget(new Surface(texture));
            // 创建 CameraCaptureSession，该对象负责管理处理预览请求和拍照请求
            cameraDevice.createCaptureSession(Arrays.asList(surface
                , imageReader.getSurface()), new CameraCaptureSession.StateCallback() // ③
                {
                    @Override
                    public void onConfigured(CameraCaptureSession cameraCaptureSession)
                    {
                        // 如果摄像头为 null，直接结束方法
                        if (null == cameraDevice)
                        {
                            return;
                        }
                        // 当摄像头已经准备好时，开始显示预览
                        captureSession = cameraCaptureSession;
                        try
                        {
                            // 设置自动对焦模式
                            previewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                                CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
                            // 设置自动曝光模式
                            previewRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
                                CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);
                            // 开始显示相机预览
                            previewRequest = previewRequestBuilder.build();
                            // 设置预览时连续捕获图像数据
                            captureSession.setRepeatingRequest(previewRequest,
                                    null, null); // ④
                        }
                        catch (CameraAccessException e)
                        {
                            e.printStackTrace();
                        }
                    }
                    @Override
                    public void onConfigureFailed(CameraCaptureSession cameraCaptureSession)
                    {
                        Toast.makeText(MainActivity.this, "配置失败！"
                            , Toast.LENGTH_SHORT).show();
                    }
                }, null
            );
        }
        catch (CameraAccessException e)
        {
            e.printStackTrace();
        }
    }
    private void setUpCameraOutputs(int width, int height)
    {
        CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_
SERVICE);
        try
        {
            // 获取指定摄像头的特性
            CameraCharacteristics characteristics
                = manager.getCameraCharacteristics(mCameraId);
```

```
            // 获取摄像头支持的配置属性
            StreamConfigurationMap map = characteristics.get(
                CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
            // 获取摄像头支持的最大尺寸
            Size largest = Collections.max(
                Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)),
                new CompareSizesByArea());
            // 创建一个 ImageReader 对象，用于获取摄像头的图像数据
            imageReader = ImageReader.newInstance(largest.getWidth(), largest.getHeight(),
                ImageFormat.JPEG, 2);
            imageReader.setOnImageAvailableListener(
                new ImageReader.OnImageAvailableListener()
                {
                    // 当照片数据可用时激发该方法
                    @Override
                    public void onImageAvailable(ImageReader reader)
                    {
                        // 获取捕获的照片数据
                        Image image = reader.acquireNextImage();
                        ByteBuffer buffer = image.getPlanes()[0].getBuffer();
                        byte[] bytes = new byte[buffer.remaining()];
                        // 使用 IO 流将照片写入指定文件
                        File file = new File(getExternalFilesDir(null), "pic.jpg");
                        buffer.get(bytes);
                        try (
                            FileOutputStream output = new FileOutputStream(file))
                        {
                            output.write(bytes);
                            Toast.makeText(MainActivity.this, "保存: "
                                + file, Toast.LENGTH_SHORT).show();
                        }
                        catch (Exception e)
                        {
                            e.printStackTrace();
                        }
                        finally
                        {
                            image.close();
                        }
                    }
                }, null);
            // 获取最佳的预览尺寸
            previewSize = chooseOptimalSize(map.getOutputSizes(
                SurfaceTexture.class), width, height, largest);
            // 根据选中的预览尺寸来调整预览组件 (TextureView) 的长宽比
            int orientation = getResources().getConfiguration().orientation;
            if (orientation == Configuration.ORIENTATION_LANDSCAPE)
            {
                textureView.setAspectRatio(previewSize.getWidth(), previewSize.
getHeight());
            }
            else
            {
                textureView.setAspectRatio(previewSize.getHeight(),
previewSize.getWidth());
            }
        }
        catch (CameraAccessException e)
        {
            e.printStackTrace();
        }
        catch (NullPointerException e)
        {
```

```
            System.out.println("出现错误。");
        }
    }
    private static Size chooseOptimalSize(Size[] choices
        , int width, int height, Size aspectRatio)
    {
        // 收集摄像头支持的大过预览 Surface 的分辨率
        List<Size> bigEnough = new ArrayList<>();
        int w = aspectRatio.getWidth();
        int h = aspectRatio.getHeight();
        for (Size option : choices)
        {
            if (option.getHeight() == option.getWidth() * h / w &&
                option.getWidth() >= width && option.getHeight() >= height)
            {
                bigEnough.add(option);
            }
        }
        // 如果找到多个预览尺寸，获取其中面积最小的
        if (bigEnough.size() > 0)
        {
            return Collections.min(bigEnough, new CompareSizesByArea());
        }
        else
        {
            System.out.println("找不到合适的预览尺寸！！！");
            return choices[0];
        }
    }
    // 为 Size 定义一个比较器 Comparator
    static class CompareSizesByArea implements Comparator<Size>
    {
        @Override
        public int compare(Size lhs, Size rhs)
        {
            // 强转为 long 保证不会发生溢出
            return Long.signum((long) lhs.getWidth() * lhs.getHeight() -
                (long) rhs.getWidth() * rhs.getHeight());
        }
    }
}
```

上面程序中的①号粗体字代码用于打开系统摄像头，openCamera()方法的第一个参数代表请求打开的摄像头 ID，此处传入的摄像头 ID 为"0"，这代表打开设备后置摄像头；如果需要打开设备指定摄像头（比如前置摄像头），可以在调用 openCamera()方法时传入相应的摄像头 ID。

提示：

CameraManager 提供了 getCameraIdList()方法来获取设备的摄像头列表，还提供了 getCameraCharacteristics(String cameraId)方法来获取指定摄像头的特性。例如如下代码：

```
// 获取设备上摄像头列表
String[] ids = CameraManager.getCameraIdList();
// 创建一个空的 CameraInfo 对象，用于获取摄像头信息
Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
for ( String id : ids)
{
    CameraCharacteristics cc = getCameraCharacteristics(id);
    // 接下来的代码就可以通过 cc 来获取该摄像头的特性了
    ...
}
```

上面程序中的①号粗体字代码打开后置摄像头时传入了一个 stateCallback 参数，该参数代表的对象可检测摄像头的状态改变，当摄像头的状态发生改变时，程序将会自动回调该对象的相应方法。该程序的关键是重写了 stateCallback 的 onOpened(CameraDevice cameraDevice)方法——当摄像头被打开时将会自动激发该方法，通过该方法的参数即可让程序获取被打开的摄像头设备。除此之外，程序在 onOpened()方法的②号粗体字代码处调用 createCameraPreviewSession()方法创建了 CameraCaptureSession，并开始预览取景。

createCameraPreviewSession()方法中的③号粗体字代码调用了 CameraDevice 的 createCaptureSession()方法来创建 CameraCaptureSession，调用该方法时也传入了一个 CameraCaptureSession.StateCallback 参数，这样即可保证当 CameraCaptureSession 被创建成功之后立即开始预览。

createCameraPreviewSession()方法的第一行粗体字代码将 texture 组件添加为 previewRequestBuilder 的 target，这意味着程序通过 previewRequestBuilder 获取的图像数据将会被显示在 texture 组件上。

程序重写了 CameraCaptureSession.StateCallback 的 onConfigured()方法——当 CameraCaptureSession 创建成功时将会自动回调该方法，该方法先通过 previewRequestBuilder 设置了预览参数，然后调用 CameraCaptureSession 对象的 setRepeatingRequest()方法开始预览。

当单击程序界面上的拍照按钮时，程序将会激发该 Activity 的 captureStillPicture()方法。该方法的实现逻辑同样很简单：程序先创建一个 CaptureRequest.Builder 对象，该方法中第一行粗体字代码将 ImageReader 添加成 CaptureRequest.Builder 的 target——这意味着当程序拍照时，图像数据将会被传给此 ImageReader。接下来程序通过 CaptureRequest.Builder 设置了拍照参数，然后通过⑤号粗体字代码调用 CameraCaptureSession 的 capture()方法拍照即可，调用该方法时也传入了 CaptureCallback 参数，这样可以保证拍照完成之后会重新开始预览。

注意：

该应用打开摄像头、创建 CameraCaptureSession、预览、拍照时都没有传入 Handler 参数，这意味着程序直接在主线程中完成相应的 Callback 任务，这样可能导致程序响应变慢。对于实际的应用，我们建议传入 Handler 参数，这样即可让 Handler 使用新线程来执行 Callback 任务，这样才可提高应用的响应速度。

由于该程序需要使用手机的摄像头，因此还需要在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 授予该程序使用摄像头的权限 -->
<uses-permission android:name="android.permission.CAMERA" />
```

在 Genymotion 模拟器上运行该程序可能看到如图 11.8 所示的预览界面——这是因为 Genymotion 模拟器可以使用宿主电脑上的摄像头作为相机摄像头。

为了让模拟器能显示图 11.8 所示的预览界面，建议读者启用 Genymotion 模拟器的摄像头支持：单击 Genymotion 模拟器右边的摄像头图标，即可看到如图 11.9 所示的对话框。按该图上标出的提示即可打开 Genymotion 模拟器的摄像头支持。

运行该程序，按下右下角的“拍照”键，程序将会把拍得的照片保存下来，界面上也会显示该照片的存储目录。

图 11.8　预览界面

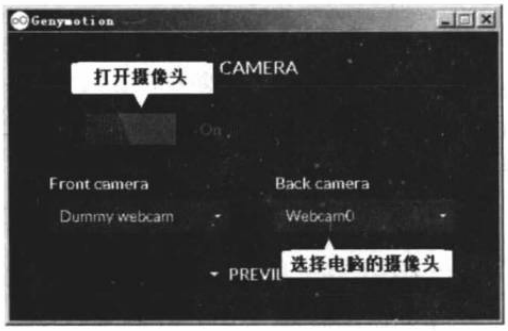


图 11.9　打开 Genymotion 模拟器的摄像头

➤➤ 11.3.2　录制视频短片

MediaRecorder 除了可用于录制音频之外，还可用于录制视频。使用 MediaRecorder 录制视频与录制音频的步骤基本相同。只是录制视频时不仅需要采集声音，还需要采集图像。为了让 MediaRecorder 录制时采集图像，应该在调用 setAudioSource(int audio_source)方法时再调用 setVideoSource(int video_source)方法来设置图像来源。

除此之外，还需要在调用 setOutputFormat()方法设置输出文件格式之后执行如下步骤。

❶ 调用 MediaRecorder 对象的 setVideoEncoder()、setVideoEncodingBitRate(int bitRate)、setVideoFrameRate()方法设置所录制的视频编码格式、编码位率、每秒多少帧等，这些参数可以控制所录制的视频品质、文件大小。一般来说，视频品质越好，视频文件越大。

❷ 调用 MediaRecorder 的 setPreviewDisplay(Surface sv)方法设置使用哪个 SurfaceView 来显示视频预览。

剩下的代码则与录制音频的代码基本相同。

实例：录制生活短片

本实例示范了如何录制视频。该程序的界面中提供了两个按钮用于控制开始、结束录制；除此之外，程序界面中还提供了一个 SurfaceView 来显示视频预览。该程序的界面布局文件如下。

程序清单：codes\11\11.3\RecordVideo\app\src\main\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 显示视频预览的 SurfaceView -->
<SurfaceView
    android:id="@+id/sView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true">
    <ImageButton
        android:id="@+id/record"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/record" />
    <ImageButton
        android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/stop" />
</LinearLayout>
</RelativeLayout>
```

提供了上面所示的界面布局文件之后，接下来就可以在程序中使用 MediaRecorder 来录制视频了。录制视频与录制音频的步骤基本相似，只是需要额外设置视频的图像来源、视频格式等。除此之外，还需要设置使用 SurfaceView 显示视频预览。录制视频的程序代码如下。

程序清单：codes\11\11.3\RecordVideo\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity
    implements OnClickListener
{
    // 程序中的两个按钮
    ImageButton record , stop;
    // 系统的视频文件
    File videoFile ;
    MediaRecorder mRecorder;
    // 显示视频预览的 SurfaceView
    SurfaceView sView;
    // 记录是否正在进行录制
    private boolean isRecording = false;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的两个按钮
        record = (ImageButton) findViewById(R.id.record);
        stop = (ImageButton) findViewById(R.id.stop);
        // 让 stop 按钮不可用
        stop.setEnabled(false);
        // 为两个按钮的单击事件绑定监听器
        record.setOnClickListener(this);
        stop.setOnClickListener(this);
        // 获取程序界面中的 SurfaceView
        sView = (SurfaceView) this.findViewById(R.id.sView);
        // 设置 Surface 不需要自己维护缓冲区
        sView.getHolder().setType(SurfaceHolder
        .SURFACE_TYPE_PUSH_BUFFERS);
        // 设置分辨率
        sView.getHolder().setFixedSize(320, 280);
        // 设置该组件让屏幕不会自动关闭
        sView.getHolder().setKeepScreenOn(true);
    }
    @Override
    public void onClick(View source)
    {
        switch (source.getId())
        {
```

```
// 单击录制按钮
case R.id.record:
    if (!Environment.getExternalStorageState().equals(
        android.os.Environment.MEDIA_MOUNTED))
    {
        Toast.makeText(MainActivity.this
            , "SD卡不存在，请插入SD卡！"
            , Toast.LENGTH_SHORT).show();
        return;
    }
    try
    {
        // 创建保存录制视频的视频文件
        videoFile = new File(Environment
            .getExternalStorageDirectory()
            .getCanonicalFile() + "/myvideo.mp4");
        // 创建MediaRecorder对象
        mRecorder = new MediaRecorder();
        mRecorder.reset();
        // 设置从麦克风采集声音
        mRecorder.setAudioSource(MediaRecorder
            .AudioSource.MIC);
        // 设置从摄像头采集图像
        mRecorder.setVideoSource(MediaRecorder
            .VideoSource.CAMERA);
        // 设置视频文件的输出格式
        // 必须在设置声音编码格式、图像编码格式之前设置
        mRecorder.setOutputFormat(MediaRecorder
            .OutputFormat.MPEG_4);
        // 设置声音编码格式
        mRecorder.setAudioEncoder(MediaRecorder
            .AudioEncoder.DEFAULT);
        // 设置图像编码格式
        mRecorder.setVideoEncoder(MediaRecorder
            .VideoEncoder.MPEG_4_SP);
        mRecorder.setVideoSize(320, 280);
        // 每秒 4 帧
        mRecorder.setVideoFrameRate(4);
        mRecorder.setOutputFile(videoFile.getAbsolutePath());
        // 指定使用SurfaceView来预览视频
        mRecorder.setPreviewDisplay(sView
            .getHolder().getSurface()); // ①
        mRecorder.prepare();
        // 开始录制
        mRecorder.start();
        System.out.println("---recording---");
        // 让record按钮不可用
        record.setEnabled(false);
        // 让stop按钮可用
        stop.setEnabled(true);
        isRecording = true;
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    break;
// 单击停止按钮
case R.id.stop:
    // 如果正在进行录制
    if (isRecording)
    {
```

```
                    // 停止录制
                    mRecorder.stop();
                    // 释放资源
                    mRecorder.release();
                    mRecorder = null;
                    // 让 record 按钮可用
                    record.setEnabled(true);
                    // 让 stop 按钮不可用
                    stop.setEnabled(false);
                }
                break;
            }
        }
    }
```

上面程序中的粗体字代码设置了视频所采集的图像来源，以及视频的压缩格式、视频分辨率等属性，程序的①号粗体字代码则用于设置使用指定 SurfaceView 显示指定视频预览。

运行该程序需要使用麦克风录制声音，需要使用摄像头采集图像，这些都需要授予相应的权限；不仅如此，由于录制视频时视频文件增大得较快，可能需要使用外部存储器，因此需要对应用程序授予相应的权限，也就是需要在 AndroidManifest.xml 文件中增加如下授权配置：

```
<!-- 授予该程序录制声音的权限 -->
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<!-- 授予该程序使用摄像头的权限 -->
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 授予使用外部存储器的权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

当在 Genymotion 模拟器上运行该程序时，由于 Genymotion 模拟器可以直接使用宿主电脑上的摄像头作为相机摄像头，因此在该模拟器上运行该程序可以看到如图 11.10 所示的界面。

图 11.10 录制视频短片

11.4 Android 5.0 新增的屏幕捕捉

在 Android 5.0 以前，如果开发者要对 Android 屏幕进行实时截图，往往被折腾得焦头烂额。Android 5.0 的出现改变了这种现状，Android 5.0 新增了 MediaProjectionManager 管理器，该管理器可以非常方便地实现屏幕捕捉功能。

使用 MediaProjectionManager 实现屏幕捕捉的步骤如下。

① 以 MEDIA_PROJECTION_SERVICE 为参数，调用 Context.getSystemService()方法即可获取 MediaProjectionManager 实例。

② 调用 MediaProjectionManager 对象的 createScreenCaptureIntent()方法创建一个屏幕捕捉的 Intent。

③ 调用 startActivityForResult()方法启动第 2 步得到的 Intent，这样即可启动屏幕捕捉的 Intent。

④ 重写 onActivityResult()方法，在该方法中通过 MediaProjectionManager 对象来获取 MediaProjection 对象，在该对象中即可获取被捕获的屏幕。

下面示例示范了 Android 5.0 的屏幕捕捉功能。该应用界面上只包含一个 ToggleButton 和一个 SurfaceView，其中 ToggleButton 用于打开屏幕捕捉功能，而 SurfaceView 则用于显示屏

幕捕捉的图像。由于程序界面非常简单，故此处不再给出界面设计文件。

下面是该应用的 Activity 代码：

程序清单：codes\11\11.4\CaptureScreen\app\src\main\java\org\crazyit\media\MainActivity.java

```
public class MainActivity extends Activity
{
    private static final int CAPTURE_CODE = 0x123;
    private MediaProjectionManager projectionManager;
    private int screenDensity;
    private int displayWidth = 360;
    private int displayHeight = 640;
    private boolean screenSharing;
    private MediaProjection mediaProjection;
    private VirtualDisplay virtualDisplay;
    private Surface surface;
    private SurfaceView surfaceView;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取屏幕分辨率
        DisplayMetrics metrics = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(metrics);
        screenDensity = metrics.densityDpi;
        // 获取应用界面上的 SurfaceView 组件
        surfaceView = (SurfaceView) findViewById(R.id.surface);
        surface = surfaceView.getHolder().getSurface();
        // 控制界面上的 SurfaceView 组件的宽度和高度
        ViewGroup.LayoutParams lp = surfaceView.getLayoutParams();
        lp.height = displayHeight;
        lp.width = displayWidth;
        surfaceView.setLayoutParams(lp);
        // 获取 MediaProjectionManager 管理器
        projectionManager = (MediaProjectionManager)
            getSystemService(Context.MEDIA_PROJECTION_SERVICE);  // ①
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        if (mediaProjection != null)
        {
            mediaProjection.stop();
            mediaProjection = null;
        }
    }
    // 当用户单击开关按钮时激发该方法
    public void onToggleScreenShare(View view)
    {
        if (((ToggleButton) view).isChecked())
        {
            shareScreen();
        }
        else
        {
            stopScreenSharing();
        }
    }
    private void shareScreen()
    {
        screenSharing = true;
```

```
        if (surface == null)
        {
            return;
        }
        if (mediaProjection == null)
        {
            Intent intent = projectionManager.createScreenCaptureIntent(); // ②
            startActivityForResult(intent, CAPTURE_CODE); // ③
            return;
        }
    }
    private void stopScreenSharing()
    {
        screenSharing = false;
        if (virtualDisplay == null)
        {
            return;
        }
        virtualDisplay.release();
    }
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        if (requestCode == CAPTURE_CODE)
        {
            // 如果 resultCode 不等于 RESULT_OK，表明用户拒绝了屏幕捕捉
            if (resultCode != RESULT_OK)
            {
                Toast.makeText(this, "用户取消了屏幕捕捉"
                    , Toast.LENGTH_SHORT).show();
                return;
            }
            mediaProjection = projectionManager
                .getMediaProjection(resultCode, data); // ④
            virtualDisplay = mediaProjection.createVirtualDisplay("屏幕捕捉",
                    displayWidth, displayHeight, screenDensity,
                    DisplayManager.VIRTUAL_DISPLAY_FLAG_AUTO_MIRROR,
                    surface, null /*Callbacks*/, null /*Handler*/);
        }
    }
}
```

上面程序中的①号粗体字代码获取了屏幕捕捉的核心 API：MediaProjectionManager，接下来程序即可通过该 API 来执行屏幕捕捉了。程序中②号粗体字代码调用 MediaProjectionManager 的 createScreenCaptureIntent()方法创建了一个屏幕捕捉的 Intent，接下来③号粗体字代码启动该 Intent 即可开始屏幕捕捉。

当程序通过启动 Activity 开始屏幕捕捉之后，接下来程序重写了 Activity 的 onActivityResult()方法，并在该方法中通过④号粗体字代码获取了 MediaProjection 对象，程序调用 MediaProjection 的方法就可以将捕捉的图像显示到指定 Surface 中——如上面程序中最后一行代码所示。

编译、运行该程序，并打开程序界面上的 ToggleButton，即可看到如图 11.11 所示的效果。

图 11.11 屏幕捕捉

11.5 本章小结

音频和视频都是非常重要的多媒体形式，Android 系统为音频、视频等多媒体的播放、录制提供了强大的支持。学习本章需要重点掌握如何使用 MediaPlayer、SoundPool 播放音频，如何使用 VideoView、MediaPlayer 播放视频。除此之外，还需要掌握通过 MediaRecorder 录制音频的方法，以及通过 Android 5.0 新增的 Camera v2 控制摄像头拍照、录制视频的方法。

CHAPTER

12

第12章 OpenGL与3D开发

本章要点

- 3D编程的概念和基本理论
- OpenGL与OpenGL ES
- Android的OpenGL ES支持
- 在Android环境中使用OpenGL ES
- 利用OpenGL ES绘制2D图形
- 旋转图形
- 从2D到3D的转换
- 利用OpenGL ES绘制3D图形
- 通过OpenGL ES对3D图形应用贴图

前面介绍过 Android 系统的图形、图像处理。通过 Android 提供的图形、图形处理 API，开发者可以非常方便地处理二维图形的处理、开发 2D 游戏等。但现在这个时代的用户显然并不满足于 2D 操作界面和 2D 游戏，3D 技术已经被广泛应用于 PC 游戏上，3D 技术下一步将要占领的肯定是手机平台。而 Android 系统已经为 3D 技术准备好了，Android 系统完全内置了 OpenGL ES（OpenGL for Embedded System）支持，也就是说，开发者可以在 Android 平台上使用 OpenGL ES API 来开发 3D 应用。

OpenGL 本身是高效、简洁的开放图形库接口，它定义了一个跨编程语言、跨平台的编程接口规范，主要用于三维图形编程。但在手机等手持终端上运行 OpenGL 有些不太合适，所以 Android 系统内置的是 OpenGL ES 支持。本章将会向读者介绍 3D 开发的基本知识，并介绍 OpenGL ES 编程的入门知识。但由于 OpenGL ES 的内容本身很多，而本书并不是一本专门介绍 OpenGL ES 编程的专著，所以并未过多地深入 OpenGL ES 编程，即使读者没有任何 3D 编程的知识，阅读本章也不会有任何障碍。

12.1　3D 图形与 3D 开发的基本知识

现在的互联网上，虽然还有一些 2D 游戏存在，但 3D 游戏已经逐渐成为主流。毕竟 3D 游戏具有更逼真的界面，能带给玩家更好的用户体验。就目前的技术来看，开发 3D 界面的各种技术已经非常成熟，为开发 3D 界面、3D 游戏提供了基础。

初学者可能会把 3D 开发想象得十分复杂。当然，3D 游戏开发肯定要比 2D 游戏开发更加复杂，毕竟开发 3D 界面需要的数据更多。

在介绍 3D 图形开发之前，先从 2D 图形开发入手。

先来看如何定义一个 2D 的三角形。对于 2D 的三角形来说，它只需要三个点，而且这三个点位于同一个平面上，因此程序只要为每个点指定 *X*、*Y* 两个坐标值即可。

接下来看如何定义一个 3D 的三棱锥。一个三棱锥需要 4 个点，而且这 4 个点都不是位于同一个平面上，因此程序需要为每个点指定 *X*、*Y*、*Z* 三个坐标值。

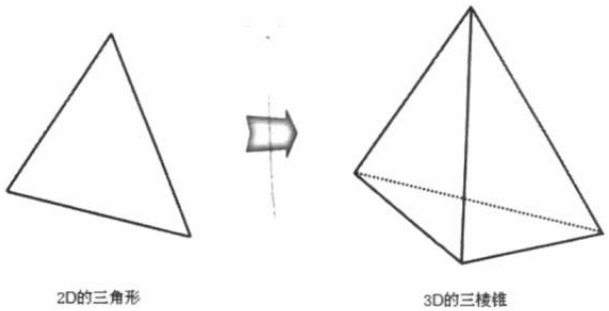


图 12.1　2D 到 3D 的转换

从图 12.1 可以看出，3D 图形需要处理的数据比 2D 图形要多得多。

从图 12.1 看到的只是最简单的 2D 图形——三角形和最简单的 3D 图形——三棱锥，但在实际应用中，应用程序需要呈现出来的 2D 图形可能由很多曲线组成；类似地，应用程序需要呈现的 3D 图形也可能是由很多曲面组成的。

前面介绍 2D 绘图时，介绍了开发随手画图的程序，当程序希望在 2D 界面上绘制一条任意的曲线时，如果把这条曲线放大了来看（放得足够大），用户将会发现这条曲线其实是由许多足够短的直线连接起来的。

对于一个 3D 图形来说，即使用户眼中看到的是一个“圆滑曲面”的 3D 图形，实际上它也

依然是由多个足够小的平面组成的，图 12.2 所示的就是从 3d max 里截取出来的一个 3D 图形。

正如图 12.2 所示，左边是用户希望看到的 3D 图形，但这个 3D 图形并不是程序员希望控制的。实际上程序员要控制的是右边的“网格图”，为了实现这个 3D 图形，程序员需要定义两方面的数据。

- 3D 图形的每个顶点（Vertex）的位置，每个顶点的位置都需要 *X*、*Y*、*Z* 三个坐标值。
- 3D 图形每个面由哪些顶点组成。

当程序给出了上面两方面的数据之后，接下来就可通知 3D 绘制接口来绘制这个 3D 图形了——就像绘制 2D 图形一样，程序员需要做的就是给出 2D 图形中各顶点的坐标。

为了定义 3D 图形每个顶点的位置，程序员需要给出 *X*、*Y*、*Z* 三个坐标值。为此，我们先要简单介绍一下 Android 的 3D 支持的三维坐标系统。

Android 的 3D 坐标系统与 2D 坐标系统完全不同，图 12.3 显示了 Android 的三维坐标系统。

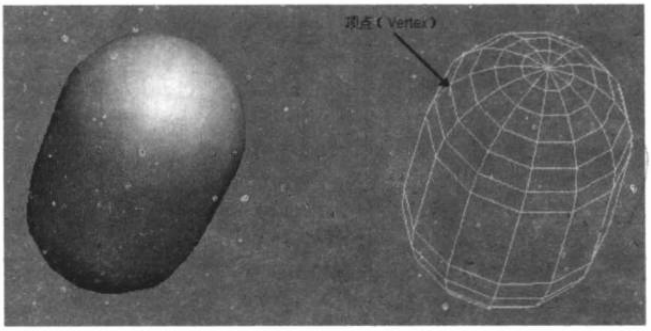


图 12.2 3D 图形的组成

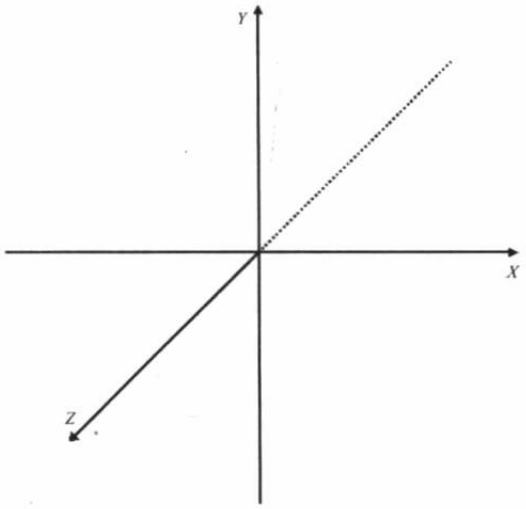


图 12.3 三维坐标系统

从图 12.3 可以看出，在 Android 的三维坐标系统中，坐标原点位于中央，*X* 轴从左向右延伸，原点左边的值为负数，右边为正数；*Y* 轴从下向上延伸，原点下边的值为负数，上边为正数；*Z* 轴从屏幕里面向外面延伸，屏幕里面为负数，外面为正数。

了解了这个三维坐标系统，我们就可以根据该三维坐标系统来定义 3D 图形每个顶点的位置了。

12.2 OpenGL 和 OpenGL ES 简介

OpenGL 的全称是 Open Graphics Library，即开放的图形库接口，它定义了一个跨编程语言、跨平台的编程接口规范，它主要用于三维图形（实际上二维图形也可以）编程。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，虽然功能强大，但是移植性不好，于是 SGI 公司便在 IRIS GL 的基础上开发了 OpenGL。

OpenGL 体系简单，而且具有跨平台的特性，它不像 Direct3D（Microsoft 开发的 3D 图形库接口，OpenGL 的最有力的竞争对手）只能在 Windows 系统上运行，因此 OpenGL 具有很广泛的适应性——它不仅适用于大型图形工作站，也适用于 PC。

在图形工作站、PC 上，OpenGL 都可以工作良好，但三维图形计算必须需要处理大量数据，因此在一些如手机之类的小型设备上，如果希望使用 OpenGL 就比较困难了。为此，Khronos

集团为 OpenGL 提供了一个子集：OpenGL ES（OpenGL for Embedded System）。

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，Khronos 协会针对手机、PDA 和游戏主机等嵌入式设备设置了 OpenGL ES。

OpenGL ES 是免费的、跨平台的、功能完善的 2D/3D 图形库接口 API，它针对多种嵌入式系统（包括控制台、移动电话、手持设备、家电设备和汽车）专门设计，它是一个精心提取出来的 OpenGL 的子集。

OpenGL ES 剔除了 OpenGl 中 glBegin/glEnd、四边形（GL_QUADS）、多边形（GL_POLYGONS）等许多非绝对必要的特性。经过多年发展，目前的 OpenGL ES 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，OpenGL ES 1.1 是以 OpenGL 1.5 规范为基础的，它们分别支持 common 和 common lite 两种 profile。common lite profile 只支持定点实数，而 common profile 既支持定点数又支持浮点数，common profile 发布于 2005 年 8 月，引入了对可编程管线的支持。

目前 Android 5.0 已经支持 OpenGL ES 3.1，而且 Android 扩展包为开发者提供了高性能的 2D 和 3D 图形 API，包括计算着色器、模板纹理、加速视觉效果、高级纹理渲染、Tessellation 着色器、几何着色器、ASTC 纹理压缩、样本缺失值插补和着色等强大功能，能够适用于不同品牌的图形处理器。

Android 专门为 OpenGL 支持提供了 android.opengl 包，在该包下提供了 GLSurfaceView、GLU、GLUtils 等工具类，通过这些工具类在 Android 应用中使用 OpenGL ES 更加方便。

12.3 绘制 2D 图形

掌握了上面关于 3D 图形开发的基本知识之后，下面可以利用 Android 的 OpenGL ES 支持来进行 3D 开发了。在真正 3D 图形之前，还需要先学习 2D 开发。

12.3.1 在 Android 应用中使用 OpenGL ES

Android 为 OpenGL ES 支持提供了 GLSurfaceView 组件，这个组件用于显示 3D 图形。GLSurfaceView 本身并不提供绘制 3D 图形的功能，而是由 GLSurfaceView.Renderer 来完成 SurfaceView 中 3D 图形的绘制。

归纳起来，在 Android 中使用 OpenGL ES 需要三个步骤。

① 创建 GLSurfaceView 组件，使用 Activity 来显示 GLSurfaceView 组件。

② 为 GLSurfaceView 组件创建 GLSurfaceView.Renderer 实例，实现 GLSurfaceView.Renderer 类时需要实现该接口里的三个方法。

- abstract void onDrawFrame(GL10 gl)：Renderer 对象调用该方法绘制 GLSurfaceView 的当前帧。
- abstract void onSurfaceChanged(GL10 gl, int width, int height)：当 GLSurfaceView 的大小改变时回调该方法。
- abstract void onSurfaceCreated(GL10 gl, EGLConfig config)：当 GLSurfaceView 被创建时回调该方法。

③ 调用 GLSurfaceView 组件的 setRenderer()方法指定 Renderer 对象，该 Renderer 对象将会完成 GLSurfaceView 里 3D 图形的绘制。

从上面的介绍不难看出，实际上绘制 3D 图形的难点不是如何使用 GLSurface 组件，而是

如何实现 Renderer 类。实现 Renderer 类时需要实现三个方法，这三个方法都有一个 GL10 形参，它就代表了 OpenGL ES 的“绘制画笔”，读者可以把它想象成 Swing 2D 绘图中的 Graphics，也可以想象成 Android 2D 绘图中的 Canvas 组件——当我们希望 Renderer 绘制 3D 图形时，实际上是调用 GL10 的方法来进行绘制的。

当 SurfaceView 被创建时，系统会回调 Renderer 对象的 onSurfaceCreated()方法，该方法可以对 OpenGL ES 执行一些无须任何改变的初始化。例如如下初始化代码：

```
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    // 关闭抗抖动
    gl.glDisable(GL10.GL_DITHER);
    // 设置系统对透视进行修正
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
    gl.glClearColor(0, 0 , 0 , 0);
    // 设置阴影平滑模式
    gl.glShadeModel(GL10.GL_SMOOTH);
    // 启用深度测试
    gl.glEnable(GL10.GL_DEPTH_TEST);
    // 设置深度测试的类型
    gl.glDepthFunc(GL10.GL_LEQUAL);
}
```

GL10 就是 OpenGL ES 的绘图接口，虽然这里看到的是一个 GL10，但实际上它也是 GLES31 的实例，读者可通过 gl instanceof GL11 判断它是否为 GL11 接口的实例。

上面的方法中用到了 GL10 的一些初始化方法，关于这些方法说明如下。

- glDisable(int cap)：用于禁用 OpenGL ES 某个方面的特性。该方法中第一行代码用于关闭抗抖动，这样可以提高性能。
- glHint (int target, int mode)：用于对 OpenGL ES 某方面进行修正。
- glclearColor(float red, float green, float blue, float alpha)：用于设置 OpenGL ES “清屏”所用的颜色，4 个参数分别设置红、绿、蓝、透明度值——0 为最小值，1 为最大值。例如设置 gl.glClearColor(0, 0 , 0 , 0)，就是用黑色“清屏”。
- glShadeModel(int mode)：用于设置 OpenGL ES 的阴影模式。此处设为阴影平滑模式。
- glEnable (int cap)：该方法与 glDisable(int cap)方法相对，用于启用 OpenGL ES 某方面的特性，此处用于启动 OpenGL ES 的深度测试。所谓深度测试，就是让 OpenGL ES 负责跟踪每个物体在 *Z* 轴上的深度，这样就可避免后面的物体遮挡前面的物体。

当 SurfaceView 组件的大小发生改变时，系统会回调 Renderer 对象的 onSurfaceChanged()方法，因此该方法通常用于初始化 3D 场景。例如如下初始化代码：

```
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    // 设置 3D 视窗的大小及位置
    gl.glViewport(0, 0, width, height);
    // 将当前矩阵模式设为投影矩阵
    gl.glMatrixMode(GL10.GL_PROJECTION);
    // 初始化单位矩阵
    gl.glLoadIdentity();
    // 计算透视视窗的宽度、高度比
    float ratio = (float)width/height;
    // 调用此方法设置透视视窗的空间大小
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
}
```

上面的方法中用到了 GL10 的一些初始化方法，关于这些方法说明如下。

- glViewport(int x, int y, int width, int height)：用于设置 3D 视窗的位置与大小。其中前两个参数指定该视窗的位置；后两个参数指定该视窗的宽、高。
- glMatrixMode(int mode)：用于设置视图的矩阵模型。通常可接受 GL10.GL_PROJECTION、GL10.GL_MODELVIEW 两个常量值。
- 当调用 gl.glMatrixMode(GL10.GL_PROJECTION);代码后，指定将屏幕设为透视图（要想看到逼真的三维物体，这是必要的），这意味着越远的东西看起来越小；当调用 gl.glMatrixMode(GL10.GL_MODELVIEW);代码后，即将当前矩阵模式设为模型视图矩阵，这意味着任何新的变换都会影响该矩阵中的所有物体。
- glLoadIdentity()：相当于 reset()方法，用于初始化单位矩阵。
- glFrustumf (float left, float right, float bottom, float top, float zNear, float zFar)：用于设置透视投影的空间大小。前两个参数用于设置 *X* 轴上的最小坐标值、最大坐标值；中间两个参数用于设置 *Y* 轴上的最小坐标值、最大坐标值；后两个参数用于设置 *Z* 轴上所能绘制的场景深度的最小值、最大值。

例如我们调用如下代码：

```
gl.glFrustumf(-0.8, 0.8, -1, 1, 1, 10);
```

这意味着如果有一个二维矩形，它的 4 个顶点的坐标分别为（-0.8, 1）、（0.8,1）、（0.8,-1）（-0.8,-1），这个矩形将会占满整个视窗。

提示：

前面已经指出，三维坐标系统与二维坐标系统并不相同，二维坐标系统上的坐标值通常就直接使用系统的像素数量；但三维坐标系统的坐标值则取决于 glFrustumf()方法的设置，当我们调用 gl.glFrustumf(-0.8, 0.8, -1, 1, 1, 10);方法时，意味着该三维坐标系统的 *X* 轴最左边的坐标值为-0.8，最右边的坐标值为 0.8；*Y* 轴最上面的坐标值为 1.0，最下面的坐标值为-1.0。

GLSurfaceView 上的所有 3D 图形都是由 Renderer 的 onDrawFrame(GL10 gl)方法绘制出来的，重写该方法时就要把所有 3D 图形都绘制出来，该方法通常以如下形式开始：

```
@Override
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT|GL10.GL_DEPTH_BUFFER_BIT);
    ...
}
```

接下来在 onDrawFrame()方法中就可以调用 GL10 的方法开始绘制了。下面会介绍 GL10 所提供的常见的绘制方法。

12.3.2 绘制平面上的多边形

前面已经说过，计算机里的 3D 图形其实是由许多个平面组合而成的。所谓“绘制 3D 图形”，其实就是通过多个平面图形形成的。下面先从绘制平面图形开始。

调用 GL10 图形绘制 2D 图形的步骤如下。

1. 调用 GL10 的 glEnableClientState(GL10.GL_VERTEX_ARRAY)方法启用顶点坐标数组。

② 调用 GL10 的 glEnableClientState(GL10.GL_COLOR_ARRAY)方法启用顶点颜色数组。

③ 调用 GL10 的 glVertexPointer(int size, int type, int stride, Buffer pointer)方法设置顶点的位置数据。这个方法中 pointer 参数用于指定顶点坐标值，但这里并未使用三维数组来指定每个顶点 *X*、*Y*、*Z* 坐标的值，pointer 依然是一个一维数组，其格式为(*x*1,*y*1,*z*1,*x*2,*y*2,*z*2,*x*3,*y*3,*z*3,…,*xN*,*yN*,*zN*)；也就是该数组里将会包含 3*N* 个数值，每 3 个值指定一个顶点的 *X*、*Y*、*Z* 坐标值。第一个参数 size 指定多少个元素指定一个顶点位置，该 size 参数通常总是 3；type 参数指定顶点坐标值的类型，如果顶点坐标值为 float 类型，则指定为 GL10.GL_FLOAT；如果顶点坐标值为整数，则指定为 GL10.GL_FIXED。

④ 调用 GL10 的 glColorPointer (int size, int type, int stride, Buffer pointer)方法设置顶点的颜色数据。这个方法中 pointer 参数用于指定顶点的颜色值，pointer 依然是一个一维数组，其格式为(*r*1,*g*1,*b*1,*a*1,*r*2,*g*2,*b*2,*a*2,*r*3,*g*3,*b*3,*a*3,…,*rN*,*gN*,*bN*,*aN*)；也就是该数组里将会包含 4*N* 个数值，每 4 个值指定一个顶点颜色的红、绿、蓝、透明度的值。第一个参数 size 指定多少个元素指定一个顶点位置，该 size 参数通常总是 4；type 参数指定顶点坐标值的类型，如果顶点坐标值为 float 类型，则指定为 GL10.GL_FLOAT；如果顶点坐标值为整数，则指定为 GL10.GL_FIXED。

⑤ 调用 GL10 的 glDrawArrays (int mode, int first, int count)方法绘制平面。该方法的第一个参数指定绘制图形类型，第二个参数指定从哪个顶点开始绘制，第三个参数指定总共绘制的顶点数量。

⑥ 绘制完成后，调用 GL10 的 glFinish()方法结束绘制，并调用 glDisableClientState(int)方法来停用顶点坐标数据、顶点颜色数据。

掌握上面的步骤之后，接下来通过示例程序来绘制几个简单的图形。

先为该程序提供一个 Renderer 实现类，该实现类的代码如下。

程序清单：codes\12\12.3\Polygon\app\src\main\java\org\crazyit\opengl\MyRenderer.java

```
public class MyRenderer implements Renderer
{
    float[] triangleData = new float[] {
        0.1f, 0.6f , 0.0f , // 上顶点
        -0.3f, 0.0f , 0.0f , // 左顶点
        0.3f, 0.1f , 0.0f  // 右顶点
    };
    int[] triangleColor = new int[] {
        65535, 0, 0, 0, // 上顶点红色
        0, 65535, 0, 0, // 左顶点绿色
        0, 0, 65535, 0 // 右顶点蓝色
    };
    float[] rectData = new float[] {
        0.4f, 0.4f , 0.0f, // 右上顶点
        0.4f, -0.4f , 0.0f, // 右下顶点
        -0.4f, 0.4f , 0.0f, // 左上顶点
        -0.4f, -0.4f , 0.0f // 左下顶点
    };
    int[] rectColor = new int[] {
        0, 65535, 0, 0, // 右上顶点绿色
        0, 0, 65535, 0, // 右下顶点蓝色
        65535, 0, 0, 0, // 左上顶点红色
        65535, 65535, 0, 0 // 左下顶点黄色
    };
    // 依然是正方形的 4 个顶点，只是顺序交换一下
    float[] rectData2 = new float[] {
```

```
        -0.4f, 0.4f , 0.0f, // 左上顶点
        0.4f, 0.4f , 0.0f, // 右上顶点
        0.4f, -0.4f , 0.0f, // 右下顶点
        -0.4f, -0.4f , 0.0f // 左下顶点
    };
    float[] pentacle = new float[]{
        0.4f , 0.4f , 0.0f,
        -0.2f , 0.3f , 0.0f,
        0.5f , 0.0f , 0f,
        -0.4f , 0.0f , 0f,
        -0.1f, -0.3f , 0f
    };
    FloatBuffer triangleDataBuffer;
    IntBuffer triangleColorBuffer;
    FloatBuffer rectDataBuffer;
    IntBuffer rectColorBuffer;
    FloatBuffer rectDataBuffer2;
    FloatBuffer pentacleBuffer;
    public MyRenderer()
    {
        // 将顶点位置数据数组转换成 FloatBuffer
        triangleDataBuffer = floatBufferUtil(triangleData);
        rectDataBuffer = floatBufferUtil(rectData);
        rectDataBuffer2 = floatBufferUtil(rectData2);
        pentacleBuffer = floatBufferUtil(pentacle);
        // 将顶点颜色数据数组转换成 IntBuffer
        triangleColorBuffer = intBufferUtil(triangleColor);
        rectColorBuffer = intBufferUtil(rectColor);
    }
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        // 关闭抗抖动
        gl.glDisable(GL10.GL_DITHER);
        // 设置系统对透视进行修正
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT
            , GL10.GL_FASTEST);
        gl.glClearColor(0, 0, 0, 0);
        // 设置阴影平滑模式
        gl.glShadeModel(GL10.GL_SMOOTH);
        // 启用深度测试
        gl.glEnable(GL10.GL_DEPTH_TEST);
        // 设置深度测试的类型
        gl.glDepthFunc(GL10.GL_LEQUAL);
    }
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        // 设置 3D 视窗的大小及位置
        gl.glViewport(0, 0, width, height);
        // 将当前矩阵模式设为投影矩阵
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // 初始化单位矩阵
        gl.glLoadIdentity();
        // 计算透视视窗的宽度、高度比
        float ratio = (float) width / height;
        // 调用此方法设置透视视窗的空间大小
        gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    }
    // 绘制图形的方法
    @Override
    public void onDrawFrame(GL10 gl)
```

```
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // 启用顶点坐标数据
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    // 启用顶点颜色数据
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    // 设置当前矩阵堆栈为模型堆栈
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    // --------------------绘制第一个图形--------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(-0.32f, 0.35f, -1.2f); // ①
    // 设置顶点的位置数据
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleDataBuffer);
    // 设置顶点的颜色数据
    gl.glColorPointer(4, GL10.GL_FIXED, 0, triangleColorBuffer);
    // 根据顶点数据绘制平面图形
    gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    // --------------------绘制第二个图形--------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(0.6f, 0.8f, -1.5f);
    // 设置顶点的位置数据
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer);
    // 设置顶点的颜色数据
    gl.glColorPointer(4, GL10.GL_FIXED, 0, rectColorBuffer);
    // 根据顶点数据绘制平面图形
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    // --------------------绘制第三个图形--------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(-0.4f, -0.5f, -1.5f);
    // 设置顶点的位置数据（依然使用之前的顶点颜色）
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer2);
    // 根据顶点数据绘制平面图形
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    // --------------------绘制第四个图形--------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(0.4f, -0.5f, -1.5f);
    // 设置使用纯色填充
    gl.glColor4f(1.0f, 0.2f, 0.2f, 0.0f); // ②
    gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
    // 设置顶点的位置数据
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, pentacleBuffer);
    // 根据顶点数据绘制平面图形
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 5);
    // 绘制结束
    gl.glFinish();
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
}
// 定义一个工具方法，将 int[]数组转换为 OpenGL ES 所需的 IntBuffer
private IntBuffer intBufferUtil(int[] arr)
{
    IntBuffer mBuffer;
    // 初始化 ByteBuffer，长度为 arr 数组的长度*4，因为一个 int 占 4 字节
    ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
    // 数组排列用 nativeOrder
    qbb.order(ByteOrder.nativeOrder());
    mBuffer = qbb.asIntBuffer();
```

```
        mBuffer.put(arr);
        mBuffer.position(0);
        return mBuffer;
    }
    // 定义一个工具方法，将float[]数组转换为OpenGL ES所需的FloatBuffer
    private FloatBuffer floatBufferUtil(float[] arr)
    {
        FloatBuffer mBuffer;
        // 初始化ByteBuffer，长度为arr数组的长度*4，因为一个int占4字节
        ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
        // 数组排列用nativeOrder
        qbb.order(ByteOrder.nativeOrder());
        mBuffer = qbb.asFloatBuffer();
        mBuffer.put(arr);
        mBuffer.position(0);
        return mBuffer;
    }
}
```

在本书的第 1 版中，为了将 int[]数组转换为 Open GL ES 所需的 IntBuffer，只要调用 IntBuffer 的 wrap()方法进行包装即可，但现在 Android 平台的要求更加严格，要求该 Buffer 必须是 native Buffer（因此使用 ByteBuffer 的 allocateDirect()方法进行创建），并且该 Buffer 必须是排序的（因此调用了 ByteBuffer 的 order()方法进行排序）。

上面程序中的粗体字代码就是使用 GL10 绘制图形的关键代码：加载顶点位置数据；加载顶点颜色数据；调用 GL10 的 glDrawArrays()方法绘制即可。由于加载顶点位置数据、顶点颜色数据时都需要 Buffer 对象，因此程序在 MyRenderer 类的构造器中把这些顶点位置数据、顶点颜色数据都包装成了相应的 FloatBuffer、IntBuffer。

上面的程序中①号代码调用了 GL10 的 glTranslatef(-0.32f, 0.35f, -1f)方法，这个 glTranslatef()方法的作用就类似于 Android 2D 绘图中 Matrix 的 setTranslate(float dx, float dy)方法，它们都用于移动绘图中心，区别只是 2D 绘图中 Matrix 的 setTranslate()方法只要指定在 *X*、*Y* 轴上的移动距离，而 GL10 的 glTranslatef()方法需要指定在 *X*、*Y*、*Z* 轴上的移动距离。在绘制图形之前，先调用 GL10 的 glTranslatef(float,float,float)方法即可保证把图形绘制在指定的中心点。

上面的程序中②号代码还调用了 glColor4f(1.0f, 0.2f, 0.2f, 0.0f)方法设置使用纯色填充。设置使用纯色填充时需要禁用顶点颜色数组，如②号代码后的一行代码。

在 Activity 中定义一个 GLSurfaceView，并使用上面的 Renderer 进行绘制，程序如下。

程序清单：codes\12\12.3\polygon\app\src\main\java\org\crazyit\opengl\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 创建一个GLSurfaceView，用于显示OpenGL绘制的图形
        GLSurfaceView glView = new GLSurfaceView(this);
        // 创建GLSurfaceView的内容绘制器
        MyRenderer myRender = new MyRenderer();
        // 为GLSurfaceView设置绘制器
        glView.setRenderer(myRender);
        setContentView(glView);
    }
}
```

运行上面的程序，可以看到如图 12.4 所示的输出。

很多读者会对图 12.4 所示绘制的图形感到奇怪：第二个图形（右上角的正方形）和第三个图形（左下角的图形）都有完全相同的 4 个坐标点，只是定义 4 个坐标点的顺序略有不同，为何绘制的图形存在这么大区别呢？

再来看 GL10 提供的 glDrawArrays(int mode, int first, int count)方法，该方法的第一个参数指定绘制模式，可指定为如下两个值。

- GL10.GL_TRIANGLES：绘制三角形。
- GL10.GL_TRIANGLE_STRIP：用多个三角形来绘制多边形。

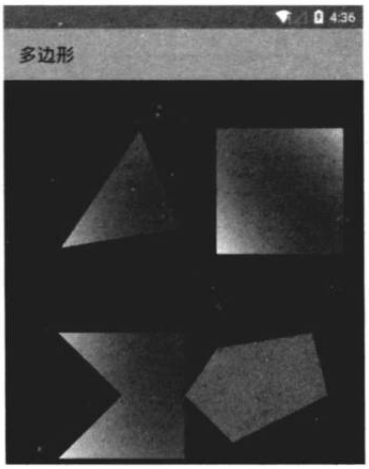


图 12.4 使用 OpenGL ES 绘制 2D 图形

提示：

前面介绍 OpenGL 与 OpenGL ES 的区别时已经指出，OpenGL ES 剔除了 OpenGl 中的四边形(GL_QUADS)、多边形(GL_POLYGONS)支持，也就是 OpenGL ES 只能绘制三角形组成 3D 图形。

当调用 glDrawArrays()方法时，如果将 mode 参数指定为 GL10.GL_TRIANGLES，则绘制简单的三角形；如果将 mode 参数指定为 GL10.GL_TRIANGLE_STRIP，那么系统将会沿着给出的顶点数据来绘制三角形。

对于上面程序中的第二个图形，程序给出 4 个顶点的顺序，如图 12.5 所示。

当指定了 glDrawArrays(int mode, int first, int count) 方法的第一个参数为 GL10.GL_TRIANGLE_STRIP 时，系统总会从 first 顶点开始，每 3 个顶点绘制一个三角形。例如调用代码：gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);，这意味着将会绘制两个三角形，分别由 0、1、2 三个顶点组成的三角形和由 1、2、3 三个顶点组成的三角形。因此，对于第二个图形而言，gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);所绘制的图形如图 12.6 所示。

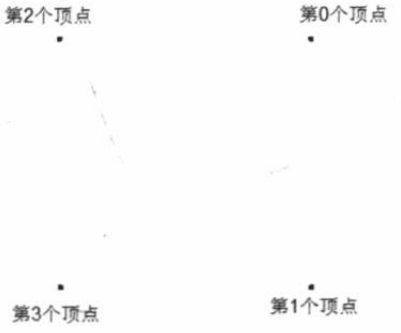


图 12.5 第二个图形的顶点数据

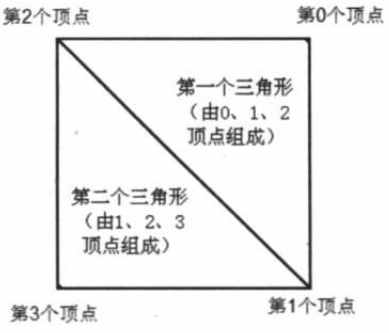


图 12.6 OpenGL ES 使用三角形绘制正方形

对于第三个图形而言，虽然它的 4 个顶点的位置与第二个图形顶点的位置完全相同，但由于 4 个顶点的顺序有所不同，所以它也是绘制两个三角形（由 0、1、2 顶点组成第一个三角形和由 1、2、3 顶点组成第二个三角形），如图 12.7 所示（浅色边界为第二个三角形的边界）。

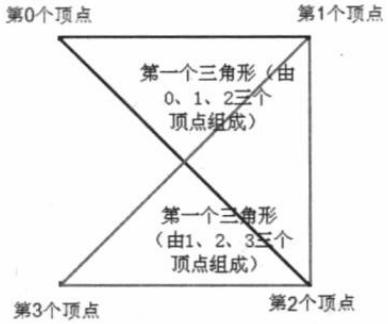


图 12.7 OpenGL ES 使用三角形绘制正方形

如图 12.7 所示的图形就与上面程序所绘制的第三个图形相同了。

讲到这里，可能有读者对 3D 开发感到害怕了——现在还只是绘制了几个简单的图形，程序员不仅要给出图形每个顶点的位置信息，还要按指定顺序来排列这些顶点，这也太复杂了吧！不要担心，现在所使用的 glDrawArrays()方法是有点复杂，实际上 3D 图形中每个顶点的坐标值不需要由程序员计算、给出；顶点的排列顺序也无须由程序员排列。

提示：

可以想象，如果 3D 场景中每个物体的所有顶点都由程序员来计算、定义，那几乎是不可想象的——如果让我们定义一个怪兽的头部，需要多少个顶点？每个顶点的位置到底应该在哪里？把一个程序员算到死都有可能！这时候通常会借助于 3d max、Maya 等三维建模工具，当我们把一个怪兽头部的模型建立出来后，这个物体的所有顶点坐标值及顶点的排列顺序都可以导出来。OpenGL 可以直接导入这些三维建模工具所建立的模型。

12.3.3 旋转

GL10 提供了一个 glRotatef(float angle, float x, float y, float z)方法，用于控制旋转。该方法中 angle 控制旋转角度；而 x、y、z 参数则共同决定了旋转轴的方向。

本质上，glRotatef(float angle, float x, float y, float z)方法的作用与 glTranslatef(float x, float y, float z)方法相似，只是 glTranslatef(float x, float y, float z)方法控制图形中心移动；而 glRotatef(float angle, float x, float y, float z)方法控制图形沿着指定旋转轴转动指定角度。

因此，只要在调用 glTranslatef()方法控制图形移动之后，再调用 glRotatef()控制图形旋转即可，如果希望看到指定图形不断旋转，则只要在 onDrawFrame(GL10 gl)方法中不断增加旋转角度即可。

下面是该程序所用的 Renderer 实现类。

程序清单：codes\12\12.3\RotatePolygon\app\src\main\java\org\crazyit\opengl\MyRenderer.java

```
public class MyRenderer implements Renderer
{
    // 省略定义顶点坐标的代码
    ...
    FloatBuffer triangleDataBuffer;
    IntBuffer triangleColorBuffer;
    FloatBuffer rectDataBuffer;
    IntBuffer rectColorBuffer;
    FloatBuffer rectDataBuffer2;
    FloatBuffer pentacleBuffer;
    // 控制旋转的角度
    private float rotate;
    public MyRenderer()
    {
        // 将顶点位置数据数组包装成 FloatBuffer
        triangleDataBuffer = floatBufferUtil(triangleData);
        rectDataBuffer = floatBufferUtil(rectData);
        rectDataBuffer2 = floatBufferUtil(rectData2);
```

```
        pentacleBuffer = floatBufferUtil(pentacle);
        // 将顶点颜色数据数组包装成 IntBuffer
        triangleColorBuffer = intBufferUtil(triangleColor);
        rectColorBuffer = intBufferUtil(rectColor);
    }
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        //省略该方法的代码
        ...
    }
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        // 省略该方法的代码
        ...
    }
    // 绘制图形的方法
    @Override
    public void onDrawFrame(GL10 gl)
    {
        // 清除屏幕缓存和深度缓存
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        // 启用顶点坐标数据
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // 启用顶点颜色数据
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        // 设置当前矩阵堆栈为模型堆栈
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        // --------------------绘制第一个图形----------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(-0.32f, 0.35f, -1.2f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleDataBuffer);
        // 设置顶点的颜色数据
        gl.glColorPointer(4, GL10.GL_FIXED, 0, triangleColorBuffer);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
        // --------------------绘制第二个图形----------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(0.6f, 0.8f, -1.5f);
        gl.glRotatef(rotate, 0f, 0f, 0.1f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer);
        // 设置顶点的颜色数据
        gl.glColorPointer(4, GL10.GL_FIXED, 0, rectColorBuffer);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        // --------------------绘制第三个图形----------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(-0.4f, -0.5f, -1.5f);
        gl.glRotatef(rotate, 0f, 0.2f, 0f);
        // 设置顶点的位置数据（依然使用之前的顶点颜色）
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer2);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        // --------------------绘制第四个图形----------------------
        // 重置当前的模型视图矩阵
```

```
        gl.glLoadIdentity();
        gl.glTranslatef(0.4f, -0.5f, -1.5f);
        // 设置使用纯色填充
        gl.glColor4f(1.0f, 0.2f, 0.2f, 0.0f);
        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, pentacleBuffer);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 5);
        // 绘制结束
        gl.glFinish();
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        // 旋转角度增加1
        rotate += 1;
    }
    // 省略 intBufferUtil 和 floatBufferUtil 两个工具方法的代码
    ...
}
```

上面程序中的粗体字代码定义了一个 rotate 变量，该变量用于控制程序中两个图形的旋转角度，其中第二个图形沿着 *Z* 轴旋转，第三个图形则沿着 *Y* 轴旋转。运行该程序，将可以看到第二个图形、第三个图形不断旋转的效果，如图 12.8 所示。

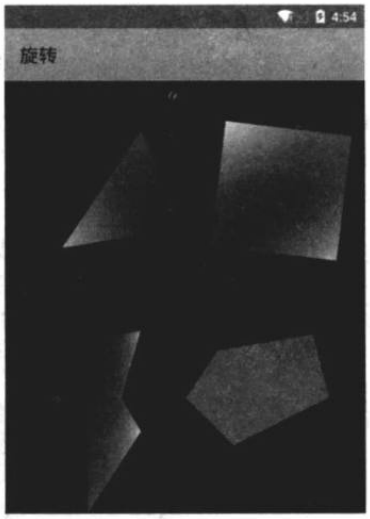


图 12.8　控制图形变换

掌握了使用 OpenGL ES 通过三角形绘制平面图形之后，接下来就可以调用 OpenGL ES 来绘制 3D 图形了——绘制 3D 图形需要定义更多的顶点数据，而且需要更好地控制哪些顶点需要组成三角形。

12.4　绘制 3D 图形

如果定义的顶点不在同一个平面上，并且使用三角形把合适的顶点连接起来，就可以绘制出 3D 图形了。

12.4.1　构建 3D 图形

正如前面所介绍的，使用 OpenGL ES 绘制 3D 图形的步骤与绘制 2D 图形的步骤大致相同，只是绘制 3D 图形需要定义更多的顶点数据，而且 3D 图形需要绘制更多的三角形。

如果还使用前面介绍的 glDrawArrays(int mode, int first, int count)进行绘制，我们将会面临巨大的考验：到底要按怎样的顺序来组织三维空间上的多个顶点，才能绘制出我们需要的三角形？为了解决这个问题，GL10 提供了如下方法。

- glDrawElements(int mode, int count, int type, Buffer indices)：根据 indices 指定的索引点来绘制三角形。该方法的第一个参数指定绘制的图形类型，可设为 GL10.GL_TRIANGLES 或 GL10.GL_TRIANGLE_STRIP；第二个参数指定一共包含多少个顶点。indices 参数最重要，它包装了一个长度为 3*N* 的数组，比如让该参数包装{0,2,3,1,4,5}数组，这意味着告诉 OpenGL ES 要绘制两个三角形，第一个三角形的三个顶点为 0、2、3 顶点，第二个三角形的三个顶点为 1、4、5 顶点。

由此可见，如果希望在程序中使用 glDrawElements(int mode, int count, int type, Buffer indices)方法来绘制 3D 图形，不仅需要指定每个 3D 图形的顶点位置信息，也需要指定 3D 图

形的每个面由哪三个顶点组成。

下面的程序使用 glDrawElements(int mode, int count, int type, Buffer indices)方法绘制了两个简单的 3D 图形：三棱锥和立方体。该程序的 Activity 依然没有变化，故下面只给出 Renderer 实现类的代码。

程序清单：codes\12\12.4\Simple3D\app\src\main\java\org\crazyit\opengl\MyRenderer.java

```
public class MyRenderer implements Renderer
{
    // 定义三棱椎的 4 个顶点
    float[] taperVertices = new float[] {
        0.0f, 0.5f, 0.0f,
        -0.5f, -0.5f, -0.2f,
        0.5f, -0.5f, -0.2f,
        0.0f, -0.2f, 0.2f
    };
    // 定义三棱椎的 4 个顶点的颜色
    int[] taperColors = new int[]{
        65535, 0, 0, 0,  // 红色
        0, 65535, 0, 0,     // 绿色
        0, 0, 65535, 0,  // 蓝色
        65535, 65535, 0, 0 //黄色
    };
    // 定义三棱椎的 4 个三角面
    private byte[] taperFacets = new byte[]{
        0, 1, 2, // 0、1、2 三个顶点组成一个面
        0, 1, 3, // 0、1、3 三个顶点组成一个面
        1, 2, 3, // 1、2、3 三个顶点组成一个面
        0, 2, 3 // 0、2、3 三个顶点组成一个面
    };
    // 定义立方体的 8 个顶点
    float[] cubeVertices = new float[] {
        // 上顶面正方形的 4 个顶点
        0.5f, 0.5f, 0.5f,
        0.5f, -0.5f, 0.5f,
        -0.5f, -0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f,
        // 下底面正方形的 4 个顶点
        0.5f, 0.5f, -0.5f,
        0.5f, -0.5f, -0.5f,
        -0.5f, -0.5f, -0.5f,
        -0.5f, 0.5f, -0.5f
    };
    // 定义立方体所需要的 6 个面（一共是 12 个三角形所需的顶点）
    private byte[] cubeFacets = new byte[]{
        0, 1, 2,
        0, 2, 3,
        2, 3, 7,
        2, 6, 7,
        0, 3, 7,
        0, 4, 7,
        4, 5, 6,
        4, 6, 7,
        0, 1, 4,
        1, 4, 5,
        1, 2, 6,
        1, 5, 6
    };
    // 定义 OpenGL ES 绘制所需要的 Buffer 对象
    FloatBuffer taperVerticesBuffer;
    IntBuffer taperColorsBuffer;
```

```
ByteBuffer taperFacetsBuffer;
FloatBuffer cubeVerticesBuffer;
ByteBuffer cubeFacetsBuffer;
// 控制旋转的角度
private float rotate;
public MyRenderer()
{
    // 将三棱椎的顶点位置数据数组包装成 FloatBuffer
    taperVerticesBuffer = floatBufferUtil(taperVertices);
    // 将三棱椎的 4 个面的数组包装成 ByteBuffer
    taperFacetsBuffer = ByteBuffer.wrap(taperFacets);
    // 将三棱椎的 4 个顶点的颜色数组包装成 IntBuffer
    taperColorsBuffer = intBufferUtil(taperColors);
    // 将立方体的顶点位置数据数组包装成 FloatBuffer
    cubeVerticesBuffer = floatBufferUtil(cubeVertices);
    // 将立方体的 6 个面（12 个三角形）的数组包装成 ByteBuffer
    cubeFacetsBuffer = ByteBuffer.wrap(cubeFacets);
}
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    // 关闭抗抖动
    gl.glDisable(GL10.GL_DITHER);
    // 设置系统对透视进行修正
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
    gl.glClearColor(0, 0, 0, 0);
    // 设置阴影平滑模式
    gl.glShadeModel(GL10.GL_SMOOTH);
    // 启用深度测试
    gl.glEnable(GL10.GL_DEPTH_TEST);
    // 设置深度测试的类型
    gl.glDepthFunc(GL10.GL_LEQUAL);
}
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    // 设置 3D 视窗的大小及位置
    gl.glViewport(0, 0, width, height);
    // 将当前矩阵模式设为投影矩阵
    gl.glMatrixMode(GL10.GL_PROJECTION);
    // 初始化单位矩阵
    gl.glLoadIdentity();
    // 计算透视视窗的宽度、高度比
    float ratio = (float) width / height;
    // 调用此方法设置透视视窗的空间大小
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
}
// 绘制图形的方法
@Override
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // 启用顶点坐标数据
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    // 启用顶点颜色数据
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    // 设置当前矩阵模式为模型视图
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    // ----------------------绘制第一个图形----------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
```

```
        gl.glTranslatef(-0.6f, 0.0f, -1.5f);
        // 沿着Y轴旋转
        gl.glRotatef(rotate, 0f, 0.2f, 0f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, taperVerticesBuffer);
        // 设置顶点的颜色数据
        gl.glColorPointer(4, GL10.GL_FIXED, 0, taperColorsBuffer);
        // 按taperFacetsBuffer指定的面绘制三角形
        gl.glDrawElements(GL10.GL_TRIANGLE_STRIP
            , taperFacetsBuffer.remaining(),
            GL10.GL_UNSIGNED_BYTE, taperFacetsBuffer);
        // --------------------绘制第二个图形--------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(0.7f, 0.0f, -2.2f);
        // 沿着Y轴旋转
        gl.glRotatef(rotate, 0f, 0.2f, 0f);
        // 沿着X轴旋转
        gl.glRotatef(rotate, 1f, 0f, 0f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, cubeVerticesBuffer);
        // 不设置顶点的颜色数据，还用以前的颜色数据
        // 按cubeFacetsBuffer指定的面绘制三角形
        gl.glDrawElements(GL10.GL_TRIANGLE_STRIP
            , cubeFacetsBuffer.remaining(),
            GL10.GL_UNSIGNED_BYTE, cubeFacetsBuffer);
        // 绘制结束
        gl.glFinish();
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        // 旋转角度增加1
        rotate += 1;
    }
    // 定义一个工具方法，将int[]数组转换为OpenGL ES所需的IntBuffer
    private IntBuffer intBufferUtil(int[] arr)
    {
        IntBuffer mBuffer;
        // 初始化ByteBuffer，长度为arr数组的长度*4，因为一个int占4字节
        ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
        // 数组排列用nativeOrder
        qbb.order(ByteOrder.nativeOrder());
        mBuffer = qbb.asIntBuffer();
        mBuffer.put(arr);
        mBuffer.position(0);
        return mBuffer;
    }
    // 定义一个工具方法，将float[]数组转换为OpenGL ES所需的FloatBuffer
    private FloatBuffer floatBufferUtil(float[] arr)
    {
        FloatBuffer mBuffer;
        // 初始化ByteBuffer，长度为arr数组的长度*4，因为一个int占4字节
        ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
        // 数组排列用nativeOrder
        qbb.order(ByteOrder.nativeOrder());
        mBuffer = qbb.asFloatBuffer();
        mBuffer.put(arr);
        mBuffer.position(0);
        return mBuffer;
    }
}
```

从上面的程序不难看出，绘制3D图形的步骤与绘制2D图形的步骤基本相似，区别只是绘制3D图形不仅需要定义各顶点位置的坐标，还需要定义3D图形的各个三角面由哪些顶点

组成，例如上面的程序中粗体字代码所示——为了定义一个立方体，除了给出这个立方体的8个顶点位置坐标之外，还需要定义组成该立方体的12个三角形。图12.9显示了该立方体上8个顶点的位置。

接下来程序需要把图12.9所示正方形的6个面（由12个三角形组成）定义出来，依次指定每个三角形包括哪三个顶点，也就是程序中粗体字代码所定义的数组。

运行上面的程序，将可以看到如图12.10所示的3D图形。

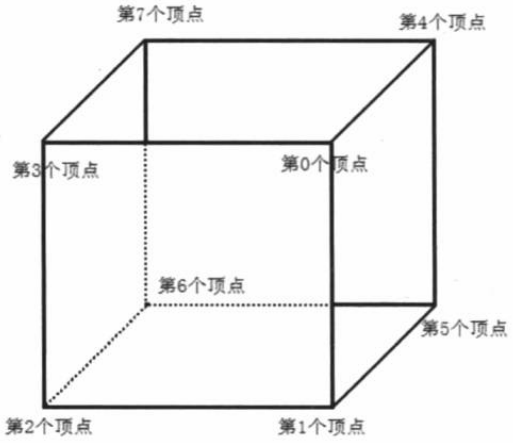


图12.9 立方体的8个顶点的位置示意图

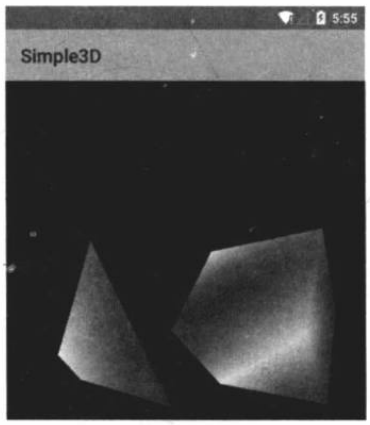


图12.10 使用OpenGL绘制3D图形

从上面的程序可以看出，为了在程序中绘制一个简单的3D立方体，程序需要定义12个三角面，这种规则的立方体还可以由程序员计算并指定，但如果遇到更复杂的3D物体，就必须借助于三维建模软件了。

12.4.2 应用纹理贴图

为了让3D图形更加逼真，我们需要为这些3D图形应用纹理贴图。比如开发一个怪兽头部，如果只是为这个头部绘制五颜六色的“皮肤”，那也太假了吧。如果考虑为这个怪兽应用“鳄鱼皮肤”的纹理贴图，或者应用“蟒蛇皮肤”的纹理贴图，这个怪兽头部就逼真多了，甚至会带给人一种恐怖的感觉。

为了在OpenGL ES中启用纹理贴图功能，可以在Renderer实现类的onSurfaceCreated(GL10 gl, EGLConfig config)方法中启用纹理贴图。例如如下代码：

```
// 启用2D纹理贴图
gl.glEnable(GL10.GL_TEXTURE_2D);
```

然后需要准备一张图片来作为纹理贴图了，建议该图片的长、宽是2的N次方，比如长、宽为256、512等。把这张准备贴图的位图放在Android项目的res/drawable/目录下，以方便应用程序加载该图片资源。

接下来程序开始加载该图片并生成对应的纹理贴图。例如如下方法：

```
// 加载位图
bitmap = BitmapFactory.decodeResource(context.getResources(),
    R.drawable.sand);
int[] textures = new int[1];
// 指定生成n个纹理（第一个参数指定生成一个纹理）
// textures数组将负责存储所有纹理的代号
gl.glGenTextures(1, textures, 0);
// 获取textures纹理数组中的第一个纹理
texture = textures[0];
```

```
// 通知 OpenGL 将 texture 纹理绑定到 GL10.GL_TEXTURE_2D 目标中
gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
// 设置纹理被缩小（距离视点很远时被缩小）时的滤波方式
gl.glTexParameterf(GL10.GL_TEXTURE_2D,
    GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
// 设置纹理被放大（距离视点很近时被放大）时的滤波方式
gl.glTexParameterf(GL10.GL_TEXTURE_2D,
    GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
// 设置在横向、纵向上都是平铺纹理
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
    GL10.GL_REPEAT);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
    GL10.GL_REPEAT);
// 加载位图生成纹理
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
```

上面的程序中用到了 GL10 的如下方法。

- glGenTextures(int n, int[] textures, int offset)：该方法指定一次性生成 *n* 个纹理。该方法所生成的纹理代号放入其中的 textures 数组中，offset 指定从第几个数组元素开始存放纹理代号。
- glBindTexture(int target, int texture)：该方法用于将 texture 纹理绑定到 target 目标上。
- glTexParameterf(int target, int pname, float param)：该方法用于为 target 纹理目标设置属性，其中第二个参数是属性名，第三个参数是属性值。

上面的程序中有 4 行代码调用了 glTexParameterf(int target, int pname, float param)方法，程序设置了当纹理被放大时使用 GL10.GL_LINEAR 滤波方式；当纹理被缩小时使用 GL10.GL_NEAREST 滤波方式。一般来说，使用 GL10.GL_LINEAR 滤波方式有较好的效果，但系统开销略微大了一些，具体采用哪种滤波方式则取决于目标机器本身的性能。

上面程序的最后一行调用了 GLUtils 工具类的方法来加载指定位图，并根据位图来生成纹理，通过上面的代码即可得到一个用于贴图的纹理了。

在 3D 绘制中进行纹理贴图也很简单，与设置顶点颜色的步骤相似，只要三步。

1. 设置启用贴图坐标数组。
2. 设置贴图坐标的数组信息。
3. 调用 GL10 的 glBindTexture(int target, int texture)方法执行贴图。

下面的程序示范了如何为一个立方体进行贴图，而且这个程序还提供了手势检测器，允许用户通过手势来改变该立方体的角度。

程序清单：codes\12\12.4\Texture3D\app\src\main\java\org\crazyit\opengl\MainActivity.java

```
public class MainActivity extends Activity
    implements OnGestureListener
{
    // 定义旋转角度
    private float anglex = 0f;
    private float angley = 0f;
    static final float ROTATE_FACTOR = 60;
    // 定义手势检测器实例
    GestureDetector detector;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 创建一个 GLSurfaceView，用于显示 OpenGL 绘制的图形
        GLSurfaceView glView = new GLSurfaceView(this);
        // 创建 GLSurfaceView 的内容绘制器
```

```
        MyRenderer myRender = new MyRenderer(this);
        // 为GLSurfaceView设置绘制器
        glView.setRenderer(myRender);
        setContentView(glView);
        // 创建手势检测器
        detector = new GestureDetector(this, this);
    }
    @Override
    public boolean onTouchEvent(MotionEvent me)
    {
        // 将该Activity上的触碰事件交给GestureDetector处理
        return detector.onTouchEvent(me);
    }
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
        float velocityX, float velocityY)
    {
        velocityX = velocityX > 2000 ? 2000 : velocityX;
        velocityX = velocityX < -2000 ? -2000 : velocityX;
        velocityY = velocityY > 2000 ? 2000 : velocityY;
        velocityY = velocityY < -2000 ? -2000 : velocityY;
        // 根据横向上的速度计算沿Y轴旋转的角度
        angley += velocityX * ROTATE_FACTOR / 4000;
        // 根据纵向上的速度计算沿X轴旋转的角度
        anglex += velocityY * ROTATE_FACTOR / 4000;
        return true;
    }
    @Override
    public boolean onDown(MotionEvent arg0)
    {
        return false;
    }
    @Override
    public void onLongPress(MotionEvent event)
    {
    }
    @Override
    public boolean onScroll(MotionEvent event1, MotionEvent event2,
        float distanceX, float distanceY)
    {
        return false;
    }
    @Override
    public void onShowPress(MotionEvent event)
    {
    }
    @Override
    public boolean onSingleTapUp(MotionEvent event)
    {
        return false;
    }
    public class MyRenderer implements Renderer
    {
        // 立方体的顶点坐标（一共是36个顶点，组成12个三角形）
        private float[] cubeVertices = { -0.6f, -0.6f, -0.6f, -0.6f, 0.6f,
            -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, -0.6f, -0.6f,
            -0.6f, -0.6f, -0.6f, -0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f,
            0.6f, 0.6f, 0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, -0.6f, -0.6f,
            0.6f, -0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, 0.6f,
            0.6f, -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, -0.6f, 0.6f,
            -0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, 0.6f, 0.6f, 0.6f,
            0.6f, 0.6f, -0.6f, 0.6f, 0.6f, -0.6f, -0.6f, 0.6f, 0.6f, -0.6f,
```

```
    -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, 0.6f,
    0.6f, 0.6f, 0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, -0.6f,
    -0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, 0.6f, 0.6f,
    -0.6f, 0.6f, -0.6f, };
// 定义立方体所需要的6个面（一共是12个三角形所需的顶点）
private byte[] cubeFacets = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
    13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
    30, 31, 32, 33, 34, 35, };
// 定义纹理贴图的坐标数据
private float[] cubeTextures = { 1.0000f, 1.0000f, 1.0000f, 0.0000f,
    0.0000f, 0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f, 1.0000f,
    1.0000f, 0.0000f, 1.0000f, 1.0000f, 1.0000f, 1.0000f, 0.0000f,
    1.0000f, 0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f, 0.0000f,
    1.0000f, 1.0000f, 1.0000f, 1.0000f, 0.0000f, 1.0000f, 0.0000f,
    0.0000f, 0.0000f, 0.0000f, 1.0000f, 0.0000f, 1.0000f, 1.0000f,
    1.0000f, 1.0000f, 0.0000f, 1.0000f, 0.0000f, 0.0000f, 0.0000f,
    0.0000f, 1.0000f, 0.0000f, 1.0000f, 1.0000f, 1.0000f, 1.0000f,
    0.0000f, 1.0000f, 0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f,
    0.0000f, 1.0000f, 1.0000f, 1.0000f, 1.0000f, 0.0000f, 1.0000f,
    0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f };
private Context context;
private FloatBuffer cubeVerticesBuffer;
private ByteBuffer cubeFacetsBuffer;
private FloatBuffer cubeTexturesBuffer;
// 定义本程序所使用的纹理
private int texture;
public MyRenderer(Context main)
{
    this.context = main;
    // 将立方体的顶点位置数据数组包装成FloatBuffer
    cubeVerticesBuffer = floatBufferUtil(cubeVertices);
    // 将立方体的6个面（12个三角形）的数组包装成ByteBuffer
    cubeFacetsBuffer = ByteBuffer.wrap(cubeFacets);
    // 将立方体的纹理贴图的坐标数据包装成FloatBuffer
    cubeTexturesBuffer = floatBufferUtil(cubeTextures);
}
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    // 该方法的大部分代码与前面的示例相同，故省略
    ...
    // 启用2D纹理贴图
    gl.glEnable(GL10.GL_TEXTURE_2D);
    // 加载纹理
    loadTexture(gl);
}
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    // 该方法的大部分代码与前面的示例相同，故省略
    ...
}
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // 启用顶点坐标数据
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    // 启用贴图坐标数组数据
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);  // ①
    // 设置当前矩阵模式为模型视图。
```

```
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        // 把绘图中心移入屏幕2个单位
        gl.glTranslatef(0f, 0.0f, -2.0f);
        // 旋转图形
        gl.glRotatef(angley, 0, 1, 0);
        gl.glRotatef(anglex, 1, 0, 0);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, cubeVerticesBuffer);
        // 设置贴图的坐标数据
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, cubeTexturesBuffer); // ②
        // 执行纹理贴图
        gl.glBindTexture(GL10.GL_TEXTURE_2D, texture); // ③
        // 按cubeFacetsBuffer指定的面绘制三角形
        gl.glDrawElements(GL10.GL_TRIANGLES, cubeFacetsBuffer.
        remaining(),
            GL10.GL_UNSIGNED_BYTE, cubeFacetsBuffer);
        // 绘制结束
        gl.glFinish();
        // 禁用顶点、纹理坐标数组
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        // 递增角度值以便每次以不同角度绘制
    }
    private void loadTexture(GL10 gl)
    {
        Bitmap bitmap = null;
        try
        {
            // 加载位图
            bitmap = BitmapFactory.decodeResource(context.getResources(),
                R.drawable.sand);
            int[] textures = new int[1];
            // 指定生成n个纹理（第一个参数指定生成一个纹理）
            // textures数组将负责存储所有纹理代号
            gl.glGenTextures(1, textures, 0);
            // 获取textures纹理数组中的第一个纹理
            texture = textures[0];
            // 通知OpenGL将texture纹理绑定到GL10.GL_TEXTURE_2D目标中
            gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
            // 设置纹理被缩小（距离视点很远时被缩小）时的滤波方式
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
            // 设置纹理被放大（距离视点很近时被放大）时的滤波方式
            gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
            // 设置在横向、纵向上都是平铺纹理
            gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_
            WRAP_S,
                GL10.GL_REPEAT);
            gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_
            WRAP_T,
                GL10.GL_REPEAT);
            // 加载位图生成纹理
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
        }
        finally
        {
            // 生成纹理之后，回收位图
            if (bitmap != null)
```

```
                bitmap.recycle();
            }
        }
    }
    // 省略 floatBufferUtil 工具方法
    ...
}
```

上面的程序中①、②、③号代码就完成了纹理贴图的三个步骤。

程序中粗体字代码用于计算用户手势在 *X* 方向、*Y* 方向上的速度，并根据 *X* 方向、*Y* 方向上的速度来改变立方体的旋转角度，这样就可以让该立方体随用户的手势而转动了。

可能有读者会发现上面这个立方体的顶点坐标看上去很“可怕”，怎么一个立方体需要这么多顶点呢？实际上是因为笔者实在不想在纸上先画出这个立方体，然后数立方体的每个三角面由哪些顶点组成，再计算每个三角面的贴图坐标——这太耗时间了。笔者直接使用 3d max 建了一个立方体模型，并从中导出了模型的每个顶点的位置信息、每个三角面由哪些顶点组成，以及贴图的坐标信息。

3d max 就不会像我们之前那样“复用”立方体的顶点——它直接计算该立方体需要 12 个三角面，每个三角面需要 3 个顶点，这样一共是 36 个顶点——其实有大量顶点的位置是相同，但 3d max 不管这些。它认为这个立方体需要 36 个顶点，每个顶点的位置需要 *X*、*Y*、*Z* 三个坐标值，因此导出这个立方体的顶点坐标信息的数组就是一个长度为 108 的数组。

提示：

由于本书只是介绍简单的 OpenGL ES 编程，因此涉及 3d max 的内容就此打住，如果读者还有更多兴趣，则可以参考 3d max 的相关资料。

运行上面的程序，用户将可以看到一个具有纹理贴图的三维立方体，用户可以通过拖动手势来旋转该立方体，程序效果如图 12.11 所示。

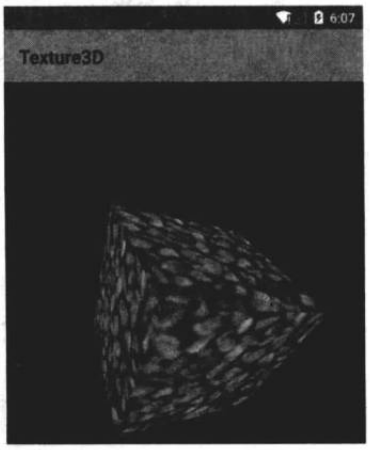


图 12.11 具有纹理贴图的立方体

需要指出的是，OpenGL ES 虽然只是 OpenGL 的一个子集，但其实它所包含的功能也很多，而本书的重点是介绍 Android 所支持的 3D 开发，限于篇幅，不可能深入介绍 OpenGL ES 编程的全部内容，如果读者希望深入了解 OpenGL ES 编程的内容，可以自行参考相关书籍和资料。

12.5 本章小结

本章主要介绍了 Android 3D 编程的入门知识，简要介绍了 2D 绘图和 3D 绘图的联系与区别，也简要介绍了 OpenGL 与 OpenGL ES。Android 系统内置了对 OpenGL ES 的支持，开发者可以在 Android 应用中通过 OpenGL ES 来绘制 3D 图形。学习本章需要掌握 3D 绘图的基本理论、三维坐标系统等。除此之外，还需要重点掌握如何在 Android 环境中使用 OpenGL ES，以及通过 OpenGL ES 绘制 2D 图形、3D 图形的方法。

CHAPTER

13

第13章 Android网络应用

本章要点

- TCP 协议的基础
- 使用 ServerSocket 建立服务器
- 使用 Socket 进行网络通信
- 在网络通信中加入多线程支持
- 使用 URL 读取网络资源
- 使用 URLConnection 提交请求
- 使用 HttpURLConnection
- Apache HttpClient 的基础知识
- 使用 HttpClient 维护用户状态、发送请求、获取响应
- 使用 WebView 浏览网页
- 使用 WebView 加载、显示 HTML 代码
- 使用 WebView 中 JavaScript 脚本调用 Android 方法
- Web Service 的基本知识
- Web Service 平台简介
- 使用 ksoap2-android 项目来调用远程 Web Service

手机本身是作为手持终端来使用的，因此它的计算能力、存储能力都是有限的。它的主要优势是携带方便，可以随时打开，而且手机通常总是处于联网状态，因此网络支持对于手机应用的重要性不言而喻。

Android完全支持JDK本身的TCP、UDP网络通信API，也可以使用ServerSocket、Socket来建立基于TCP/IP协议的网络通信还可以使用DatagramSocket、Datagrampacket、MulticastSocket来建立基于UDP协议的网络通信。如果读者有Java网络编程的经验，这些经验完全适用于Android应用的网络编程。Android也支持JDK提供的URL、URLConnection等网络通信API。

提示：

限于篇幅，本章并未涉及UDP协议编程的相关内容，如果读者需要掌握基于UDP协议，使用DatagramSocket、Datagrampacket、MulticastSocket进行网络编程的内容，可以参考疯狂Java体系的《疯狂Java讲义》。

不仅如此，Android还内置了HttpClient，这样可以非常方便地发送HTTP请求，并获取HTTP响应，通过内置HttpClient，Android简化了与网站之间的交互。令人遗憾的是，Android并未内置对Web Service的支持，为了弥补这种不足，本章将会介绍如何利用ksoap2-android项目在Android应用中调用远程Web Service。

13.1 基于TCP协议的网络通信

TCP/IP通信协议是一种可靠的网络协议，它在通信的两端各建立一个Socket，从而在通信的两端之间形成网络虚拟链路。一旦建立了虚拟的网络链路，两端的程序就可以通过虚拟链路进行通信了。Java对基于TCP协议的网络通信提供了良好的封装，Java使用Socket对象来代表两端的通信接口，并通过Socket产生IO流来进行网络通信。

13.1.1 TCP协议基础

IP协议是Internet上使用的一个关键协议，它的全称是Internet Protocol，即Internet协议，通常简称IP协议。通过使用IP协议，使Internet成为一个允许连接不同类型的计算机和不同操作系统的网络。

要使两台计算机彼此之间进行通信，两台计算机必须使用同一种“语言”，IP协议只保证计算机能发送和接收分组数据。IP协议负责将消息从一个主机传送到另一个主机，消息在传送的过程中被分割成一个个小包。

尽管计算机通过安装IP软件，保证了计算机之间可以发送和接收数据，但IP协议还不能解决数据分组在传输过程中可能出现的问题。因此，若要解决可能出现的问题，连接上Internet的计算机还需要安装TCP协议来提供可靠并且无差错的通信服务。

TCP协议被称为一种端对端协议。这是因为它为两台计算机之间的连接起了重要作用：当一台计算机需要与另一台远程计算机连接时，TCP协议会让它们建立一个连接——用于发送和接收数据的虚拟链路。

TCP协议负责收集这些信息包，并将其按适当的次序放好传送，在接收端收到后再将其正确地还原。TCP协议保证了数据包在传送中准确无误。TCP协议使用重发机制：当一个通信实体发送一个消息给另一个通信实体后，需要收到另一个通信实体的确认信息，如果没有收到另一个通信实体的确认信息，则会再次重发刚才发送的消息。

通过这种重发机制，TCP 协议向应用程序提供可靠的通信连接，使它能够自动适应网上的各种变化。即使在 Internet 暂时出现堵塞的情况下，TCP 也能够保证通信的可靠。

图 13.1 显示了 TCP 协议控制两个通信实体互相通信的示意图。

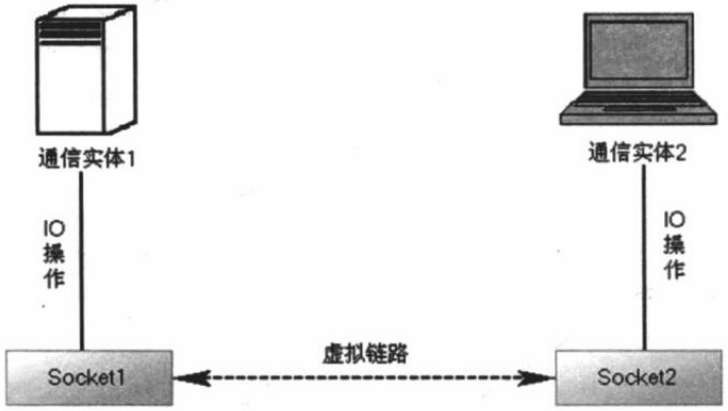


图 13.1 TCP 协议的通信示意图

综上所述，虽然 IP 和 TCP 这两个协议的功能不尽相同，也可以分开单独使用，但它们是在同一时期作为一个协议来设计的，并且在功能上也是互补的。只有两者结合，才能保证 Internet 在复杂的环境下正常运行。凡是要连接到 Internet 的计算机，都必须同时安装和使用这两个协议，因此在实际中常把这两个协议统称为 TCP/IP 协议。

13.1.2 使用 ServerSocket 创建 TCP 服务器端

从图 13.1 中看上去 TCP 通信的两个通信实体之间并没有服务器端、客户端之分，但那是两个通信实体已经建立虚拟链路之后的示意图。在两个通信实体没有建立虚拟链路之前，必须有一个通信实体先做出“主动姿态”，主动接收来自其他通信实体的连接请求。

Java 中能接收其他通信实体连接请求的类是 ServerSocket，ServerSocket 对象用于监听来自客户端的 Socket 连接，如果没有连接，它将一直处于等待状态。ServerSocket 包含一个监听来自客户端连接请求的方法。

- Socket accept()：如果接收到一个客户端 Socket 的连接请求，该方法将返回一个与连接客户端 Socket 对应的 Socket（如图 13.1 所示，每个 TCP 连接有两个 Socket）；否则该方法将一直处于等待状态，线程也被阻塞。

为了创建 ServerSocket 对象，ServerSocket 类提供了如下几个构造器。

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

当 ServerSocket 使用完毕后，应使用 ServerSocket 的 close()方法来关闭该 ServerSocket。在通常情况下，服务器不应该只接收一个客户端请求，而应该不断地接收来自客户端的所有请求，所以 Java 程序通常会通过循环不断地调用 ServerSocket 的 accept()方法，如以下代码片段所示。

```
// 创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
ServerSocket ss = new ServerSocket(30000);
// 采用循环不断接收来自客户端的请求
while (true)
```

```
{
    // 每当接收到客户端 Socket 的请求时，服务器端也对应产生一个 Socket
    Socket s = ss.accept();
    // 下面就可以使用 Socket 进行通信了
    ...
}
```

提示：

上面的程序中创建 ServerSocket 没有指定 IP 地址，则该 ServerSocket 将会绑定到本机默认的 IP 地址。程序中使用 30000 作为该 ServerSocket 的端口号，通常推荐使用 1024 以上的端口，主要是为了避免与其他应用程序的通用端口冲突。

由于手机无线上网的 IP 地址通常都是由移动运营公司动态分配的，一般不会有自己固定的 IP 地址，因此很少在手机上运行服务器端，服务器端通常运行在有固定 IP 地址的服务器上。本节所介绍的应用程序的服务器端也是运行在 PC 上的。

13.1.3 使用 Socket 进行通信

客户端通常可以使用 Socket 的构造器来连接到指定服务器，Socket 通常可提供如下两个构造器。

- Socket(InetAddress/String remoteAddress, int port)：创建连接到指定远程主机、远程端口的 Socket，该构造器没有指定本地地址、本地端口，默认使用本地主机的默认 IP 地址，默认使用系统动态分配的端口。
- Socket(InetAddress/String remoteAddress, int port, InetAddress localAddr, int localPort)：创建连接到指定远程主机、远程端口的 Socket，并指定本地 IP 地址和本地端口，适用于本地主机有多个 IP 地址的情形。

上面两个构造器中指定远程主机时既可使用 InetAddress 来指定，也可直接使用 String 对象来指定，但程序通常使用 String 对象（如 192.168.2.23）来指定远程 IP 地址。当本地主机只有一个 IP 地址时，使用第一个方法更为简单，如以下代码所示。

```
// 创建连接到 192.168.1.88、30000 端口的 Socket
Socket s = new Socket("192.168.1.88" , 30000);
// 下面就可以使用 Socket 进行通信了
...
```

当程序执行上面的粗体字代码时，该代码将会连接到指定服务器，让服务器端的 ServerSocket 的 accept()方法向下执行，于是服务器端和客户端就产生一对互相连接的 Socket。

提示：

上面的程序连接到“远程主机”的 IP 地址使用的是 192.168.1.88，这个 IP 地址就是笔者运行服务器端程序的 IP 地址。

当客户端、服务器端产生了对应的 Socket 之后，此时就到了如图 13.1 所示的通信示意图，程序无须再区分服务器、客户端，而是通过各自的 Socket 进行通信。Socket 提供了如下两个方法来获取输入流和输出流。

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

看到这两个方法返回的 InputStream 和 OutputStream，读者应该可以明白 Java 在设计 IO 体系上的苦心了：不管底层的 IO 流是怎样的节点流，文件流也好，网络 Socket 产生的流也好，程序都可以将其包装成处理流，从而提供更多方便的处理。下面以一个最简单的网络通信程序为例来介绍基于 TCP 协议的网络通信。

提示：

如果读者朋友阅读过疯狂 Java 体系的《疯狂 Java 讲义》，可能会发现本章的示例程序、文字内容与《疯狂 Java 讲义》介绍网络编程的一章有较多相同的内容。实际上也是这样的，因为 Android 网络通信还是依赖于 JDK 在 java.net、java.io 两个包下的 API。

下面的服务器端程序需要在 PC 上运行，该程序非常简单，因此不需要建立 Android 项目，直接定义一个 Java 类，并运行该 Java 类即可。它仅仅建立 ServerSocket 监听，并使用 Socket 获取输出流输出。

程序清单：codes\13\13.1\SimpleServer\SimpleServer.java

```
public class SimpleServer
{
    public static void main(String[] args)
        throws IOException
    {
        // 创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
        ServerSocket ss = new ServerSocket(30000); // ①
        // 采用循环不断接收来自客户端的请求
        while (true)
        {
            // 每当接收到客户端 Socket 的请求时，服务器端也对应产生一个 Socket
            Socket s = ss.accept();
            OutputStream os = s.getOutputStream();
            os.write("您好，您收到了服务器的新年祝福！\n"
                .getBytes("utf-8"));
            // 关闭输出流，关闭 Socket
            os.close();
            s.close();
        }
    }
}
```

上面程序中的①号粗体字代码建立了一个 ServerSocket，该 ServerSocket 在 30000 端口监听，该 ServerSocket 将会一直监听，等待客户端程序的连接。程序接下来的 4 行粗体字代码用于打开 Socket 对应的输出流，并向输出流中写入一段字符串数据。

注意：

上面的程序并未把 OutputStream 流包装成 PrintStream，然后使用 PrintStream 直接输出整个字符串，这是因为该服务器端程序运行于 Windows 主机上，当直接使用 PrintStream 输出字符串时默认使用系统平台的字符串（即 GBK）进行编码；但该程序的客户端是 Android 应用，运行于 Linux 平台（Android 是 Linux 内核的）上，因此当客户端读取网络数据时默认使用 UTF-8 字符集进行解码，这样势必引起乱码。为了保证客户端能正常解析到数据，此处手动控制字符串的编码，强行指定使用 UTF-8 字符集进行编码，这样就可以避免乱码问题了。

接下来的客户端程序也非常简单，它仅仅使用 Socket 建立与指定 IP 地址、指定端口的连接，并使用 Socket 获取输入流读取数据。该客户端程序是一个 Android 应用，因此还是需要先建立 Android 项目，该程序的界面中包含一个文本框，用于显示从服务器端读取的字符串数据。

程序清单：codes\13\13.1\SimpleClient\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (EditText) findViewById(R.id.show);
        new Thread()
        {
            @Override
            public void run()
            {
                try
                {
                    // 建立连接到远程服务器的 Socket
                    Socket socket = new Socket("192.168.1.88" , 30000); // ①
                    // 将 Socket 对应的输入流包装成 BufferedReader
                    BufferedReader br = new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
                    // 进行普通 IO 操作
                    String line = br.readLine();
                    show.setText("来自服务器的数据：" + line);
                    // 关闭输入流、socket
                    br.close();
                    socket.close();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        }.start();
    }
}
```

上面程序中的①号粗体字代码是使用 ServerSocket 和 Socket 建立网络连接的代码，接下来的几行粗体字代码是通过 Socket 获取输入流、输出流进行通信的代码。通过程序不难看出，一旦使用 ServerSocket、Socket 建立网络连接之后，程序通过网络通信与普通 IO 就没有太大的区别了。

该程序与本书第 1 版中的程序有所区别：本书第 1 版为了简化编程、突出网络编程的主题，因此直接在 UI 线程中建立网络连接，通过网络读取服务器响应数据——但在实际项目中并不推荐这么做，因为建立网络连接、网络通信是不稳定的，它所需要的时间也不确定，因此直接在 UI 线程中建立网络连接、通过网络读取数据可能阻塞 UI 线程，导致 Android 应用失去响应。在最新的 Android 平台上，Android 已经不允许在 UI 线程中建立网络连接，因此上面的示例启动了一条新线程来建立网络连接，并读取网络数据。

注意：

新版 Android 平台与 Android 2.3.x 在网络编程这一章最大的区别是，新版 Android 平台不允许直接在 UI 线程中建立网络连接、访问网络资源，因此本章后面的所有示例程序都会将建立网络连接、访问网络资源的操作放在新线程中完成。

该 Android 应用需要访问互联网，因此还需要为该应用赋予访问互联网的权限，也就是在 AndroidManifest.xml 文件中增加如下配置片段：

```
<!--授权访问互联网-->
<uses-permission android:name="android.permission.INTERNET"/>
```

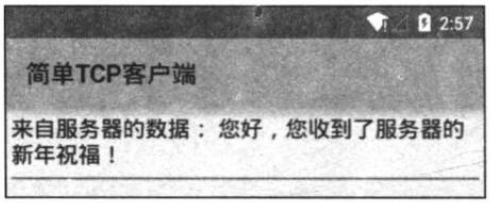


图 13.2 与远程服务器交互

先运行上面程序中的 SimpleServer 类，将看到服务器一直处于等待状态，因为服务器使用了死循环来接收来自客户端的请求；再运行客户端 AndroidClient 类，将可看到程序输出：“来自服务器的数据：您好，您收到了服务器的新年祝福！”，如图 13.2 所示，这表明客户端和服务器端通信成功。

上面的程序为了突出通过 ServerSocket 和 Socket 建立连接并通过底层 IO 流进行通信的主题，程序没有进行异常处理，也没有使用 finally 块来关闭资源。

在实际应用中，程序可能不想让执行网络连接、读取服务器数据的进程一直阻塞，而是希望当网络连接、读取操作超过合理时间之后，系统自动认为该操作失败，这个合理时间就是超时时长。Socket 对象提供了一个 setSoTimeout(int timeout)来设置超时时长，如下面的代码片段所示。

```
Socket s = new Socket("127.0.0.1" , 30000);
// 设置 10 秒之后即认为超时
s.setSoTimeout(10000);
```

为 Socket 对象指定了超时时长之后，如果使用 Socket 进行读写操作完成之前已经超出了该时间限制，那么这个方法就会抛出 SocketTimeoutException 异常，程序可以对该异常进行捕获，并进行适当处理，如以下代码所示。

```
try
{
    // 使用 Scanner 来读取网络输入流中的数据
    Scanner scan = new Scanner(s.getInputStream())
    // 读取一行字符
    String line = scan.nextLine()
    ...
}
// 捕获 SocketTimeoutException 异常
catch(SocketTimeoutException ex)
{
    // 对异常进行处理
    ...
}
```

假设程序需要为 Socket 连接服务器时指定超时时长，即经过指定时间后，如果该 Socket 还未连接到远程服务器，则系统认为该 Socket 连接超时。但 Socket 的所有构造器里都没有提供指定超时时长的参数，所以程序应该先创建一个无连接的 Socket，再调用 Socket 的 connect()方法来连接远程服务器，而 connect()方法就可以接受一个超时时长参数，如以下代码所示。

```
// 创建一个无连接的 Socket
Socket s = new Socket();
// 让该 Socket 连接到远程服务器，如果经过 10 秒还没有连接到，则认为连接超时
s.connect(new InetSocketAddress(host, port) ,10000);
```

13.1.4 加入多线程

前面服务器端和客户端只是进行了简单的通信操作：服务器接收到客户端连接之后，服务器向客户端输出一个字符串，而客户端也只是读取服务器的字符串后就退出了。在实际应用中的客户端则可能需要和服务器端保持长时间通信，即服务器需要不断地读取客户端数据，并向客户端写入数据；客户端也需要不断地读取服务器数据，并向服务器写入数据。

当使用传统 BufferedReader 的 readLine()方法读取数据时，在该方法成功返回之前，线程被阻塞，程序无法继续执行。考虑到这个原因，服务器应该为每个 Socket 单独启动一条线程，每条线程负责与一个客户端进行通信。

客户端读取服务器数据的线程同样会被阻塞，所以系统应该单独启动一条线程，该线程专门负责读取服务器数据。

下面考虑实现一个简单的 C/S 聊天室应用。服务器端应该包含多条线程，每个 Socket 对应一条线程，该线程负责读取 Socket 对应输入流的数据（从客户端发送过来的数据），并将读到的数据向每个 Socket 输出流发送一遍（将一个客户端发送的数据“广播”给其他客户端），因此需要在服务器端使用 List 来保存所有的 Socket。

下面是服务器端的实现代码。程序为服务器提供了两个类：一个是创建 ServerSocket 监听的主类；另一个是负责处理每个 Socket 通信的线程类。

程序清单：codes\13\13.1\MultiThreadServer\MyServer.java

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

上面的程序是服务器端只负责接收客户端 Socket 的连接请求，每当客户端 Socket 连接到该 ServerSocket 之后，程序将对应 Socket 加入 socketList 集合中保存，并为该 Socket 启动一条线程，该线程负责处理该 Socket 所有的通信任务，如程序中 4 行粗体字代码所示。服务器端线程类的代码如下。

程序清单：codes\13\13.1\MultiThreadServer\ServerThread.java

```
// 负责处理每条线程通信的线程类
public class ServerThread implements Runnable
{
```

```
    // 定义当前线程所处理的 Socket
    Socket s = null;
    // 该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    public ServerThread(Socket s)
        throws IOException
    {
        this.s = s;
        // 初始化该 Socket 对应的输入流
        br = new BufferedReader(new InputStreamReader(
            s.getInputStream() , "utf-8"));  // ②
    }
    public void run()
    {
        try
        {
            String content = null;
            // 采用循环不断从 Socket 中读取客户端发送过来的数据
            while ((content = readFromClient()) != null)
            {
                // 遍历 socketList 中的每个 Socket
                // 将读到的内容向每个 Socket 发送一次
                for (Iterator<Socket> it = MyServer.socketList.iterator(); it.hasNext(); )
                {
                    Socket s = it.next();
                    try{
                        OutputStream os = s.getOutputStream();
                        os.write((content + "\n").getBytes("utf-8"));
                    }
                    catch(SocketException e)
                    {
                        e.printStackTrace();
                        // 删除该 Socket
                        it.remove();
                        System.out.println(MyServer.socketList);
                    }
                }
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    // 定义读取客户端数据的方法
    private String readFromClient()
    {
        try
        {
            return br.readLine();
        }
        // 如果捕获到异常，表明该 Socket 对应的客户端已经关闭
        catch (IOException e)
        {
            e.printStackTrace();
            // 删除该 Socket
            MyServer.socketList.remove(s);  // ①
        }
        return null;
    }
}
```

上面的服务器端线程类不断读取客户端数据，程序使用 readFromClient()方法来读取客户

端数据，如果在读取数据过程中捕获到 IOException 异常，则表明该 Socket 对应的客户端 Socket 出现了问题（到底什么问题我们不管，反正不正常），程序就将该 Socket 从 socketList 中删除，如 readFromClient()方法中①号粗体字代码所示。

当服务器线程读到客户端数据之后，程序遍历 socketList 集合，并将该数据向 socketList 集合中的每个 Socket 发送一次——该服务器线程将把从 Socket 中读到的数据向 socketList 中的每个 Socket 转发一次，如 run()线程执行体中的粗体字代码所示。

提示：

上面程序中的②号粗体字代码将网络的字节输入流转换为字符输入流时，指定了转换所用的字符串：UTF-8，这也是由于客户端写过来的数据是采用 UTF-8 字符集进行编码的，所以此处的服务器端也要使用 UTF-8 字符集进行解码。当需要编写跨平台的网络通信程序时，使用 UTF-8 字符集进行编码、解码是一种较好的解决方案。

每个客户端应该包含两条线程：一条负责生成主界面，响应用户动作，并将用户输入的数据写入 Socket 对应的输出流中；另一条负责读取 Socket 对应输入流中的数据（从服务器发送过来的数据），并负责将这些数据在程序界面上显示出来。

客户端程序同样是一个 Android 应用，因此需要创建一个 Android 项目，这个 Android 应用的界面中包含两个文本框：一个用于接收用户的输入；另一个用于显示聊天信息。界面中还有一个按钮，当用户单击该按钮时，程序向服务器发送聊天信息。该程序的界面布局代码如下。

程序清单：codes\13\13.1\MultiThreadClient\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<!-- 定义一个文本框，用于接收用户的输入 -->
<EditText
    android:id="@+id/input"
    android:layout_width="280dp"
    android:layout_height="wrap_content" />
<Button
    android:id="@+id/send"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingLeft="8dp"
    android:text="@string/send"/>
</LinearLayout>
<!-- 定义一个文本框，用于显示来自服务器的信息 -->
<TextView
    android:id="@+id/show"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="top"
    android:background="#ffff"
    android:textSize="14dp"
    android:textColor="#f000"/>
</LinearLayout>
```

客户端的 Activity 负责生成程序界面，并为程序的按钮单击事件绑定事件监听器，当用户单击按钮时向服务器发送信息。客户端的 Activity 代码如下。

程序清单：codes\13\13.1\MultiThreadClient\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义界面上的两个文本框
    EditText input;
    TextView show;
    // 定义界面上的一个按钮
    Button send;
    Handler handler;
    // 定义与服务器通信的子线程
    ClientThread clientThread;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        input = (EditText) findViewById(R.id.input);
        send = (Button) findViewById(R.id.send);
        show = (TextView) findViewById(R.id.show);
        handler = new Handler() // ②
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 如果消息来自于子线程
                if (msg.what == 0x123)
                {
                    // 将读取的内容追加显示在文本框中
                    show.append("\n" + msg.obj.toString());
                }
            }
        };
        clientThread = new ClientThread(handler);
        // 客户端启动 ClientThread 线程创建网络连接，读取来自服务器的数据
        new Thread(clientThread).start(); // ①
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                try
                {
                    // 当用户单击“发送”按钮后，将用户输入的数据封装成 Message
                    // 然后发送给子线程的 Handler
                    Message msg = new Message();
                    msg.what = 0x345;
                    msg.obj = input.getText().toString();
                    clientThread.revHandler.sendMessage(msg);
                    // 清空 input 文本框
                    input.setText("");
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

当用户单击该程序界面中的“发送”按钮后，程序将会把 input 输入框中的内容发送给 clientThread 的 revHandler 对象，clientThread 负责将用户输入的内容发送给服务器。

为了避免 UI 线程被阻塞，该程序将建立网络连接、与网络服务器通信等工作都交给 ClientThread 线程完成，因此该程序在①号粗体字代码处启动 ClientThread 线程。

由于 Android 不允许子线程访问界面组件，因此上面的程序定义了一个 Handler 来处理来自子线程的消息，如程序中②号粗体字代码所示。

ClientThread 子线程负责建立与远程服务器的连接，并负责与远程服务器通信，读到数据之后便通过 Handler 对象发送一条消息；当 ClientThread 子线程收到 UI 线程发送过来的消息（消息携带了用户输入的内容）后，还负责将用户输入的内容发送给远程服务器。该子线程的代码如下。

程序清单：codes\13\13.1\MultiThreadClient\app\src\main\java\org\crazyit\net\ClientThread.java

```
public class ClientThread implements Runnable
{
    private Socket s;
    // 定义向 UI 线程发送消息的 Handler 对象
    private Handler handler;
    // 定义接收 UI 线程的消息的 Handler 对象
    public Handler revHandler;
    // 该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    OutputStream os = null;
    public ClientThread(Handler handler)
    {
        this.handler = handler;
    }
    public void run()
    {
        try
        {
            s = new Socket("192.168.1.88", 30000);
            br = new BufferedReader(new InputStreamReader(
                s.getInputStream()));
            os = s.getOutputStream();
            // 启动一条子线程来读取服务器响应的数据
            new Thread()
            {
                @Override
                public void run()
                {
                    String content = null;
                    // 不断读取 Socket 输入流中的内容
                    try
                    {
                        while ((content = br.readLine()) != null)
                        {
                            // 每当读到来自服务器的数据之后，发送消息通知
                            // 程序界面显示该数据
                            Message msg = new Message();
                            msg.what = 0x123;
                            msg.obj = content;
                            handler.sendMessage(msg);
                        }
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
```

```
            }
        }.start();
        // 为当前线程初始化 Looper
        Looper.prepare();
        // 创建 revHandler 对象
        revHandler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 接收到 UI 线程中用户输入的数据
                if (msg.what == 0x345)
                {
                    // 将用户在文本框内输入的内容写入网络
                    try
                    {
                        os.write((msg.obj.toString() + "\r\n")
                            .getBytes("utf-8"));
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                }
            }
        };
        // 启动 Looper
        Looper.loop();
    }
    catch (SocketTimeoutException e1)
    {
        System.out.println("网络连接超时！！");
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
  }
}
```

上面线程的功能也非常简单，它只是不断地获取 Socket 输入流中的内容，当读到 Socket 输入流中的内容后，便通过 Handler 对象发送一条消息，消息负责携带读到的数据，如上面程序中第一段粗体字代码所示。除此之外，该子线程还负责读取 UI 线程发送的消息，接收到消息之后，该子线程负责将消息中携带的数据发送给远程服务器，如上面程序中第二段粗体字代码所示。

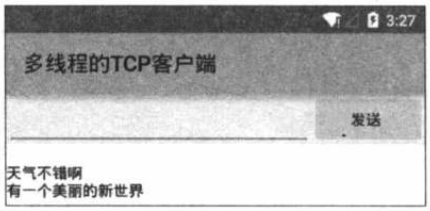


图 13.3　支持多线程的 TCP 客户端

先运行上面程序中的 MyServer 类，该类运行后只是作为服务器，看不到任何输出。接着可以运行 Android 客户端——相当于启动聊天室客户端登录该服务器，接下来在任何一个 Android 客户端输入一些内容后单击“发送”按钮，将可以看到所有客户端（包括自己）都会收到刚刚输入的内容，如图 13.3 所示，这就粗略实现了一个 C/S 结构聊天室的功能。

借助于此处介绍的网络通信机制，我们可以在 Android 平台上开发大量功能强大的网络通信程序，比如《疯狂 Java 讲义》中所介绍的网络五子棋、网络斗地主等，这些程序的网络通信部分都可按上面介绍的方式来实现，只是不同游戏可能需要在网络上交换不同类型的数据，这可能需要封装自己的网络协议。关于基于 Socket 通信的更详细介绍，读者可以对照《疯狂

Java讲义》中网络编程的相关知识，这些知识是完全相通的。

13.2 使用URL访问网络资源

URL（Uniform Resource Locator）对象代表统一资源定位器，它是指向互联网“资源”的指针。资源可以是简单的文件或目录，也可以是对更复杂的对象的引用，例如对数据库或搜索引擎的查询。就通常情况而言，URL可以由协议名、主机、端口和资源组成，即满足如下格式：

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

例如如下的URL地址：

```
http://www.crazyit.org/index.php
```

提示：

JDK中还提供了一个URI（Uniform Resource Identifiers）类，其实例代表一个统一资源标识符，Java的URI不能用于定位任何资源，它的唯一作用就是解析。与此对应的是，URL则包含一个可打开到达该资源的输入流，因此我们可以将URL理解成URI的特例。

URL类提供了多个构造器用于创建URL对象，一旦获得了URL对象之后，就可以调用如下常用方法来访问该URL对应的资源了。

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

13.2.1 使用URL读取网络资源

URL对象中前面几个方法都非常容易理解，而该对象提供的openStream()可以读取该URL资源的InputStream，通过该方法可以非常方便地读取远程资源。

下面的程序示范了如何通过URL类读取远程资源。

程序清单：codes\13\13.2\URLTest\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    ImageView show;
    // 代表从网络下载得到的图片
    Bitmap bitmap;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if(msg.what == 0x123)
```

```
            {
                // 使用 ImageView 显示该图片
                show.setImageBitmap(bitmap);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (ImageView) findViewById(R.id.show);
        new Thread()
        {
            public void run()
            {
                try
                {
                    // 定义一个 URL 对象
                    URL url = new URL("http://www.crazyit.org/"
                        + "attachments/month_1008/20100812_7763e970f"
                        + "822325bfb019ELQVym8tW3A.png");
                    // 打开该 URL 对应的资源的输入流
                    InputStream is = url.openStream();
                    // 从 InputStream 中解析出图片
                    bitmap = BitmapFactory.decodeStream(is);
                    // 发送消息，通知 UI 组件显示该图片
                    handler.sendEmptyMessage(0x123);
                    is.close();
                    // 再次打开 URL 对应的资源的输入流
                    is = url.openStream();
                    // 打开手机文件对应的输出流
                    OutputStream os = openFileOutput("crazyit.png"
                        , MODE_PRIVATE);
                    byte[] buff = new byte[1024];
                    int hasRead = 0;
                    // 将 URL 对应的资源下载到本地
                    while((hasRead = is.read(buff)) > 0)
                    {
                        os.write(buff, 0 , hasRead);
                    }
                    is.close();
                    os.close();
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        }.start();
    }
}
```

上面的程序两次调用了 URL 对象的 openStream()方法打开 URL 对应的资源的输入流，程序第一次使用 BitmapFactory 的 decodeStream(InputStream)方法来解析该输入流中的图片；第二次则使用 IO 将输入流中的图片下载到本地。

该程序同样需要访问互联网，因此需要授予该程序访问网络的权限，也就是需要在 AndroidManifest.xml 文件中增加如下授权代码：

```
<!-- 授权访问网络 -->
<uses-permission android:name="android.permission.INTERNET"/>
```

运行该程序，将可以看到如图 13.4 所示的输出。

如图 13.4 所显示的图片就是程序中 URL 对象所对应的图片，运行该程序不仅可以显示该 URL 对象所对应的图片，而且还会在手机文件系统的/data/data/org.crazyit.net/files/目录下生成 crazyit.png 图片，该图片就是通过 URL 从网络上下载的图片。

图 13.4 使用 URL 读取网络图片

13.2.2 使用 URLConnection 提交请求

URL 的 openConnection()方法将返回一个 URLConnection 对象，该对象表示应用程序和 URL 之间的通信连接。程序可以通过 URLConnection 实例向该 URL 发送请求，读取 URL 引用的资源。

通常创建一个和 URL 的连接，并发送请求、读取此 URL 引用的资源需要如下几个步骤。

1. 通过调用 URL 对象的 openConnection()方法来创建 URLConnection 对象。
2. 设置 URLConnection 的参数和普通请求属性。
3. 如果只是发送 GET 方式的请求，那么使用 connect 方法建立和远程资源之间的实际连接即可；如果需要发送 POST 方式的请求，则需要获取 URLConnection 实例对应的输出流来发送请求参数。
4. 远程资源变为可用，程序可以访问远程资源的头字段，或通过输入流读取远程资源的数据。

在建立和远程资源的实际连接之前，程序可以通过如下方法来设置请求头字段。

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

除此之外，还可以使用如下方法来设置或增加通用的头字段。

- setRequestProperty(String key, String value)：设置该 URLConnection 的 key 请求头字段的值为 value，如以下代码所示。

```
conn.setRequestProperty("accept" , "*/*")
```

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

当远程资源可用之后，程序可以使用以下方法来访问头字段和内容。

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

注意：

如果既要使用输入流读取 URLConnection 响应的内容，也要使用输出流发送请求参数，一定要先使用输出流，再使用输入流。

getHeaderField()方法用于根据响应头字段来返回对应的值。而某些头字段由于经常需要访问，所以 Java 提供了以下方法来访问特定响应头字段的值。

- getContentEncoding()：获取 content-encoding 响应头字段的值。
- getContentLength()：获取 content-length 响应头字段的值。
- getContentType()：获取 content-type 响应头字段的值。
- getDate()：获取 date 响应头字段的值。
- getExpiration()：获取 expires 响应头字段的值。
- getLastModified()：获取 last-modified 响应头字段的值。

下面的程序示范了如何向 Web 站点发送 GET 请求、POST 请求，并从 Web 站点取得响应。该程序中用到一个发送 GET、POST 请求的工具类，该工具类的代码如下。

程序清单：codes\13\13.2\GetPostTest\app\src\main\java\org\crazyit\net\GetPostUtil.java

```
public class GetPostUtil
{
    /**
     * 向指定 URL 发送 GET 方式的请求
     * @param url 发送请求的 URL
     * @param params 请求参数，请求参数应该是 name1=value1&name2=value2 的形式
     * @return URL 所代表远程资源的响应
     */
    public static String sendGet(String url, String params)
    {
        String result = "";
        BufferedReader in = null;
        try
        {
            String urlName = url + "?" + params;
            URL realUrl = new URL(urlName);
            // 打开和 URL 之间的连接
            URLConnection conn = realUrl.openConnection();
            // 设置通用的请求属性
            conn.setRequestProperty("accept", "*/*");
            conn.setRequestProperty("connection", "Keep-Alive");
            conn.setRequestProperty("user-agent",
                "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)");
            // 建立实际的连接
            conn.connect(); // ①
            // 获取所有的响应头字段
            Map<String, List<String>> map = conn.getHeaderFields();
            // 遍历所有的响应头字段
            for (String key : map.keySet())
            {
                System.out.println(key + "--->" + map.get(key));
            }
            // 定义 BufferedReader 输入流来读取 URL 的响应
            in = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
            String line;
            while ((line = in.readLine()) != null)
            {
```

```
            result += "\n" + line;
        }
    }
    catch (Exception e)
    {
        System.out.println("发送GET请求出现异常！" + e);
        e.printStackTrace();
    }
    // 使用finally块来关闭输入流
    finally
    {
        try
        {
            if (in != null)
            {
                in.close();
            }
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
    return result;
}
/**
 * 向指定URL发送POST方式的请求
 * @param url 发送请求的URL
 * @param params 请求参数，请求参数应该是name1=value1&name2=value2的形式
 * @return URL所代表远程资源的响应
 */
public static String sendPost(String url, String params)
{
    PrintWriter out = null;
    BufferedReader in = null;
    String result = "";
    try
    {
        URL realUrl = new URL(url);
        // 打开和URL之间的连接
        URLConnection conn = realUrl.openConnection();
        // 设置通用的请求属性
        conn.setRequestProperty("accept", "*/*");
        conn.setRequestProperty("connection", "Keep-Alive");
        conn.setRequestProperty("user-agent",
            "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)");
        // 发送POST请求必须设置如下两行
        conn.setDoOutput(true);
        conn.setDoInput(true);
        // 获取URLConnection对象对应的输出流
        out = new PrintWriter(conn.getOutputStream());
        // 发送请求参数
        out.print(params);  // ②
        // flush输出流的缓冲
        out.flush();
        // 定义BufferedReader输入流来读取URL的响应
        in = new BufferedReader(
            new InputStreamReader(conn.getInputStream()));
        String line;
        while ((line = in.readLine()) != null)
        {
            result += "\n" + line;
        }
```

```
        }
        catch (Exception e)
        {
            System.out.println("发送 POST 请求出现异常！" + e);
            e.printStackTrace();
        }
        // 使用 finally 块来关闭输出流、输入流
        finally
        {
            try
            {
                if (out != null)
                {
                    out.close();
                }
                if (in != null)
                {
                    in.close();
                }
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
            }
        }
        return result;
    }
}
```

从上面的程序可以看出，如果需要发送 GET 请求，只要调用 URLConnection 的 connect() 方法去建立实际的连接即可，如以上程序中①号粗体字代码所示。如果需要发送 POST 请求，则需要获取 URLConnection 的 OutputStream，然后再向网络中输出请求参数，如以上程序中②号粗体字代码所示。

提供了上面发送 GET 请求、POST 请求的工具类之后，接下来就可以在 Activity 类中通过该工具类来发送请求了。该程序的界面中包含两个按钮，一个按钮用于发送 GET 请求，一个按钮用于发送 POST 请求。程序还提供了一个 EditText 来显示远程服务器的响应。该程序的界面布局很简单，故此处不再给出界面布局文件。该程序的 Activity 代码如下。

程序清单：codes\13\13.2\GetPostTest\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    Button get , post;
    TextView show;
    // 代表服务器响应的字符串
    String response;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if(msg.what == 0x123)
            {
                // 设置 show 组件显示服务器响应
                show.setText(response);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        get = (Button) findViewById(R.id.get);
        post = (Button) findViewById(R.id.post);
        show = (TextView)findViewById(R.id.show);
        get.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        response = GetPostUtil.sendGet(
                            "http://192.168.1.88:8888/abc/a.jsp"
                            , null);
                        // 发送消息通知UI线程更新UI组件
                        handler.sendEmptyMessage(0x123);
                    }
                }.start();
            }
        });
        post.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        response = GetPostUtil.sendPost(
                            "http://192.168.1.88:8888/abc/login.jsp"
                            , "name=crazyit.org&pass=leegang");
                    }
                }.start();
                // 发送消息通知UI线程更新UI组件
                handler.sendEmptyMessage(0x123);
            }
        });
    }
}
```

上面程序中的两行粗体字代码分别用于发送GET请求、POST请求，该程序所发送的GET请求、POST请求都是向本地局域网内http://192.168.1.88:8888/abc应用下的两个页面发送的，这个应用实际上是部署在笔者本机上的Web应用。

提示：

abc这个Web应用的代码在光盘的codes\13\13.2路径下，这个Web应用需要部署在Web服务器（比如Tomcat 8.0）中才可使用。关于如何开发Web应用，如何安装、部署Web应用，可参考疯狂Java体系的《轻量级Java EE企业应用实战》一书。

在Web服务器中成功部署abc应用之后，运行上面的Android应用，单击“发送GET请求”按钮，将可以看到如图13.5所示的输出。

如果单击“发送POST请求”按钮，程序将会向abc应用下的login.jsp页面发送请求，并

提交 name=crazyit.org&pass=leegang 请求参数，此时将可以看到如图 13.6 所示的输出。

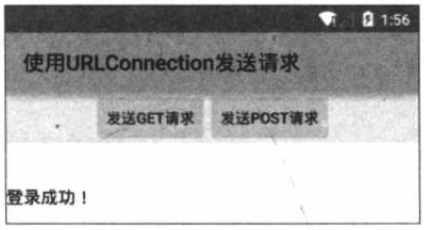


图 13.5　使用 URLConnection 对外发送 GET 请求　　图 13.6　使用 URLConnection 对外发送 POST 请求

从上面的介绍可以发现，借助于 URLConnection 类的帮助，应用程序可以非常方便地与指定站点交换信息，包括发送 GET 请求、POST 请求，并获取网站的响应等。

13.3　使用 HTTP 访问网络

前面介绍了 URLConnection 可以非常方便地与指定站点交换信息，URLConnection 还有一个子类：HttpURLConnection，HttpURLConnection 在 URLConnection 的基础上做了进一步改进，增加了一些用于操作 HTTP 资源的便捷方法。

13.3.1　使用 HttpURLConnection

HttpURLConnection 继承了 URLConnection，因此也可用于向指定网站发送 GET 请求、POST 请求。它在 URLConnection 的基础上提供了如下便捷的方法。

- int getResponseCode()：获取服务器的响应代码。
- String getResponseMessage()：获取服务器的响应消息。
- String getRequestMethod()：获取发送请求的方法。
- void setRequestMethod(String method)：设置发送请求的方法。

下面通过一个实用的示例来示范使用 HttpURLConnection 实现多线程下载。

实例：多线程下载

使用多线程下载文件可以更快地完成文件的下载，因为客户端启动多条线程进行下载就意味着服务器也需要为该客户端提供相应的服务。假设服务器同时最多服务 100 个用户，在服务器中一条线程对应一个用户，100 条线程在计算机内并发执行，也就是由 CPU 划分时间片轮流执行，如果 A 应用使用了 99 条线程下载文件，那么相当于占用了 99 个用户的资源，自然就拥有了较快的下载速度。

注意：

实际上并不是客户端并发的下载线程越多，程序的下载速度就越快，因为当客户端开启太多的并发线程之后，应用程序需要维护每条线程的开销、线程同步的开销，这些开销反而会导致下载速度降低。

为了实现多线程下载，程序可按如下步骤进行。

1. 创建 URL 对象。
2. 获取指定 URL 对象所指向资源的大小（由 getContentLength()方法实现），此处用到了 HttpURLConnection 类。
3. 在本地磁盘上创建一个与网络资源相同大小的空文件。
4. 计算每条线程应该下载网络资源的哪个部分（从哪个字节开始，到哪个字节结束）。
5. 依次创建、启动多条线程来下载网络资源的指定部分。

该程序提供的下载工具类的代码如下。

程序清单：codes\13\13.3\MultiThreadDown\app\src\main\java\org\crazyit\net\DownUtil.java

```
public class DownUtil
{
    // 定义下载资源的路径
    private String path;
    // 指定所下载的文件的保存位置
    private String targetFile;
    // 定义需要使用多少条线程下载资源
    private int threadNum;
    // 定义下载的线程对象
    private DownThread[] threads;
    // 定义下载的文件的总大小
    private int fileSize;
    public DownUtil(String path, String targetFile, int threadNum)
    {
        this.path = path;
        this.threadNum = threadNum;
        // 初始化 threads 数组
        threads = new DownThread[threadNum];
        this.targetFile = targetFile;
    }
    public void download() throws Exception
    {
        URL url = new URL(path);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setConnectTimeout(5 * 1000);
        conn.setRequestMethod("GET");
        conn.setRequestProperty(
            "Accept",
            "image/gif, image/jpeg, image/pjpeg, image/pjpeg, "
            + "application/x-shockwave-flash, application/xaml+xml, "
            + "application/vnd.ms-xpsdocument, application/x-ms-xbap, "
            + "application/x-ms-application, application/vnd.ms-excel, "
            + "application/vnd.ms-powerpoint, application/msword, */*");
        conn.setRequestProperty("Accept-Language", "zh-CN");
        conn.setRequestProperty("Charset", "UTF-8");
        conn.setRequestProperty("Connection", "Keep-Alive");
        // 得到文件大小
        fileSize = conn.getContentLength();
        conn.disconnect();
        int currentPartSize = fileSize / threadNum + 1;
        RandomAccessFile file = new RandomAccessFile(targetFile, "rw");
        // 设置本地文件的大小
        file.setLength(fileSize);
        file.close();
        for (int i = 0; i < threadNum; i++)
        {
            // 计算每条线程下载的开始位置
            int startPos = i * currentPartSize;
            // 每条线程使用一个 RandomAccessFile 进行下载
```

```
            RandomAccessFile currentPart = new RandomAccessFile(targetFile,
                "rw");
            // 定位该线程的下载位置
            currentPart.seek(startPos);
            // 创建下载线程
            threads[i] = new DownThread(startPos, currentPartSize,
                currentPart);
            // 启动下载线程
            threads[i].start();
        }
    }
    // 获取下载的完成百分比
    public double getCompleteRate()
    {
        // 统计多条线程已经下载的总大小
        int sumSize = 0;
        for (int i = 0; i < threadNum; i++)
        {
            sumSize += threads[i].length;
        }
        // 返回已经完成的百分比
        return sumSize * 1.0 / fileSize;
    }
    private class DownThread extends Thread
    {
        // 当前线程的下载位置
        private int startPos;
        // 定义当前线程负责下载的文件大小
        private int currentPartSize;
        // 当前线程需要下载的文件块
        private RandomAccessFile currentPart;
        // 定义该线程已下载的字节数
        public int length;
        public DownThread(int startPos, int currentPartSize,
            RandomAccessFile currentPart)
        {
            this.startPos = startPos;
            this.currentPartSize = currentPartSize;
            this.currentPart = currentPart;
        }
        @Override
        public void run()
        {
            try
            {
                URL url = new URL(path);
                HttpURLConnection conn = (HttpURLConnection)url
                    .openConnection();
                conn.setConnectTimeout(5 * 1000);
                conn.setRequestMethod("GET");
                conn.setRequestProperty(
                    "Accept",
                    "image/gif, image/jpeg, image/pjpeg, image/pjpeg, "
                    + "application/x-shockwave-flash, application/xaml+xml, "
                    + "application/vnd.ms-xpsdocument, application/x-ms-xbap, "
                    + "application/x-ms-application, application/vnd.ms-excel, "
                    + "application/vnd.ms-powerpoint, application/msword, */*");
                conn.setRequestProperty("Accept-Language", "zh-CN");
                conn.setRequestProperty("Charset", "UTF-8");
                InputStream inStream = conn.getInputStream();
                // 跳过 startPos 个字节，表明该线程只下载自己负责的那部分文件
                skipFully(inStream, this.startPos);
                // inStream.skip(this.startPos);
```

```
                byte[] buffer = new byte[1024];
                int hasRead = 0;
                // 读取网络数据，并写入本地文件中
                while (length < currentPartSize
                    && (hasRead = inStream.read(buffer)) > 0)
                {
                    currentPart.write(buffer, 0, hasRead);
                    // 累计该线程下载的总大小
                    length += hasRead;
                }
                currentPart.close();
                inStream.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
    // 定义一个为 InputStream 跳过 bytes 字节的方法
    public static void skipFully(InputStream in, long bytes) throws IOException
    {
        long remainning = bytes;
        long len = 0;
        while (remainning > 0)
        {
            len = in.skip(remainning);
            remainning -= len;
        }
    }
}
```

上面的 DownUtil 工具类中包括一个 DownloadThread 内部类，该内部类的 run()方法负责打开远程资源的输入流，并调用 InputStream 的 skip(int)方法跳过指定数量的字节，这样就让该线程读取由它自己负责下载的部分了。

提示：

可能有读者会发现，上面这个 DownUtil 类与《疯狂 Java 讲义》中介绍的多线程下载工具类基本相似。实际上也是这样，因此这个类并未用到任何与 Android 相关的知识，其本质就是使用 JDK 提供的 HttpURLConnection 类。

备注：

由于最新的 Android 平台上调用 InputStream 的 skip()方法时，并不能总是准确地跳过对应的字节数，因此程序专门实现了一个 skipFully()方法来跳过 InputStream 的指定字节数。上面程序中第二行粗体字代码调用了 skipFully()方法来跳过指定字节数，而被注释的第三行粗体字代码则直接调用了 InputStream 的 skip()方法——由于这个方法不太可靠，因此本程序没有直接使用该方法。

提供了上面的 DownUtil 工具类之后，接下来就可以在 Activity 中调用该 DownUtil 类来执行下载任务了。该程序界面中包含两个文本框：一个用于输入网络文件的源路径；另一个用于指定下载到本地的文件的文件名。该程序的界面比较简单，故此处不再给出界面布局代码。该程序的 Activity 代码如下。

程序清单：codes\13\13.3\MultiThreadDown\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText url;
    EditText target;
    Button downBn;
    ProgressBar bar;
    DownUtil downUtil;
    private int mDownStatus;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的三个界面控件
        url = (EditText) findViewById(R.id.url);
        target = (EditText) findViewById(R.id.target);
        downBn = (Button) findViewById(R.id.down);
        bar = (ProgressBar) findViewById(R.id.bar);
        // 创建一个 Handler 对象
        final Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                if (msg.what == 0x123)
                {
                    bar.setProgress(mDownStatus);
                }
            }
        };
        downBn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 初始化 DownUtil 对象（最后一个参数指定线程数）
                downUtil = new DownUtil(url.getText().toString(),
                    target.getText().toString(), 6);
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        try
                        {
                            // 开始下载
                            downUtil.download();
                        }
                        catch (Exception e)
                        {
                            e.printStackTrace();
                        }
                        // 定义每秒调度获取一次系统的完成进度
                        final Timer timer = new Timer();
                        timer.schedule(new TimerTask()
                        {
                            @Override
                            public void run()
                            {
                                // 获取下载任务的完成比例
                                double completeRate = downUtil
                                . getCompleteRate();
```

```
                        mDownStatus = (int) (completeRate * 100);
                        // 发送消息通知界面更新进度条
                        handler.sendEmptyMessage(0x123);
                        // 下载完全后取消任务调度
                        if (mDownStatus >= 100)
                        {
                            timer.cancel();
                        }
                    }
                }, 0, 100);
            }
        }.start();
    }
});
}
}
```

上面的 Activity 不仅使用了 DownUtil 来控制程序下载，而且程序还启动了一个定时器，该定时器控制每隔 0.1 秒查询一次下载进度，并通过程序中的进度条来显示任务的下载进度。

该程序不仅需要访问网络，还需要访问系统 SD 卡，在 SD 卡中创建文件，因此必须授予该程序访问网络、访问 SD 卡文件的权限，也就是在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 配置在SD卡中创建与删除文件权限 -->
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 配置向SD卡写入数据权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!-- 授权访问网络 -->
<uses-permission android:name="android.permission.INTERNET"/>
```

运行该程序，将看到如图 13.7 所示的下载界面。

图 13.7 多线程下载

提示：

上面的程序已经实现了多线程下载的核心代码，如果要实现断点下载，则还需要额外增加一个配置文件(读者可以发现所有的断点下载工具都会在下载开始时生成两个文件：一个是与网络资源具有相同大小的空文件；一个是配置文件)，该配置文件分别记录每条线程已经下载到了哪个字节，当网络断开后再次开始下载时，每条线程根据配置文件里记录的位置向后下载即可。

13.3.2 使用 Apache HttpClient

在一般情况下，如果只是需要向 Web 站点的某个简单页面提交请求并获取服务器响应，则完全可以使用前面所介绍的 HttpURLConnection 来完成。但在绝大部分情况下，Web 站点的网页可能没这么简单，这些页面并不是通过一个简单的 URL 就可访问的，可能需要用户登录

而且具有相应的权限才可访问该页面。在这种情况下，就需要涉及 Session、Cookie 的处理了，如果打算使用 HttpURLConnection 来处理这些细节，当然也是可能实现的，只是处理起来难度就大了。

为了更好地处理向 Web 站点请求，包括处理 Session、Cookie 等细节问题，Apache 开源组织提供了一个 HttpClient 项目，看它的名称就知道，它是一个简单的 HTTP 客户端（并不是浏览器），可以用于发送 HTTP 请求，接收 HTTP 响应。但不会缓存服务器的响应，不能执行 HTML 页面中嵌入的 JavaScript 代码；也不会对页面内容进行任何解析、处理。

提示：

简单来说，HttpClient 就是一个增强版的 HttpURLConnection，HttpURLConnection 可以做的事情 HttpClient 全部可以做；HttpURLConnection 没有提供的有些功能，HttpClient 也提供了，但它只是关注于如何发送请求、接收响应，以及管理 HTTP 连接。

Android 已经成功地集成了 HttpClient，这意味着开发人员可以直接在 Android 应用中使用 HttpClient 来提交请求、接收响应。

使用 HttpClient 发送请求、接收响应很简单，只要如下几步即可。

1. 创建 HttpClient 对象。
2. 如果需要发送 GET 请求，则创建 HttpGet 对象；如果需要发送 POST 请求，则创建 HttpPost 对象。
3. 如果需要发送请求参数，则可调用 HttpGet、HttpPost 共同的 setParams(HttpParams params)方法来添加请求参数；对于 HttpPost 对象而言，也可调用 setEntity(HttpEntity entity)方法来设置请求参数。
4. 调用 HttpClient 对象的 execute(HttpUriRequest request)方法发送请求，执行该方法返回一个 HttpResponse。
5. 调用 HttpResponse 的 getAllHeaders()、getHeaders(String name)等方法可获取服务器的响应头；调用 HttpResponse 的 getEntity()方法可获取 HttpEntity 对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。

实例：访问被保护资源

下面的 Android 应用需要向指定页面发送请求，但该页面并不是一个简单的页面，只有当用户已经登录，而且登录用户的用户名是 crazyit.org 时才可访问该页面。如果使用 HttpUrlConnection 来访问该页面，那么需要处理的细节就太复杂了。下面将会借助于 HttpClient 来访问被保护页面。

访问 Web 应用中被保护页面，如果使用浏览器则十分简单，用户通过系统提供的登录页面登录系统，浏览器会负责维护与服务器之间的 Session，如果用户登录的用户名、密码符合要求，就可以访问被保护资源了。

为了通过 HttpClient 来访问被保护页面，程序同样需要使用 HttpClient 来登录系统，只要应用程序使用同一个 HttpClient 发送请求，HttpClient 就会自动维护与服务器之间的 Session 状态。也就是说，程序第一次使用 HttpClient 登录系统后，接下来使用 HttpClient 即可访问被保护页面了。

提示：

虽然此处给出的实例只是访问被保护页面，但访问其他被保护资源也与此类似。程序只要第一次通过HttpClient登录系统，接下来即可通过该HttpClient访问被保护资源了。

下面是该程序的代码。

程序清单：codes\13\13.3\HttpClientTest\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    TextView response;
    HttpClient httpClient;
    Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
            if(msg.what == 0x123)
            {
                // 使用 response 文本框显示服务器响应
                response.append(msg.obj.toString() + "\n");
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建 DefaultHttpClient 对象
        httpClient = new DefaultHttpClient();
        response = (TextView) findViewById(R.id.response);
    }
    public void accessSecret(View v)
    {
        response.setText("");
        new Thread()
        {
            @Override
            public void run()
            {
                // 创建一个 HttpGet 对象
                HttpGet get = new HttpGet(
                    "http://192.168.1.88:8888/foo/secret.jsp");  // ①
                try
                {
                    // 发送 GET 请求
                    HttpResponse httpResponse = httpClient.execute(get);  // ②
                    HttpEntity entity = httpResponse.getEntity();
                    if (entity != null)
                    {
                        // 读取服务器响应
                        BufferedReader br = new BufferedReader(
                            new InputStreamReader(entity.getContent()));
                        String line = null;

                        while ((line = br.readLine()) != null)
                        {
                            Message msg = new Message();
                            msg.what = 0x123;
                            msg.obj = line;
                            handler.sendMessage(msg);
```

```
                }
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}.start();
}
public void showLogin(View v)
{
    // 加载登录界面
    final View loginDialog = getLayoutInflater().inflate(
        R.layout.login, null);
    // 使用对话框供用户登录系统
    new AlertDialog.Builder(MainActivity.this)
        .setTitle("登录系统")
        .setView(loginDialog)
        .setPositiveButton("登录",
        new DialogInterface.OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog,
                int which)
            {
                // 获取用户输入的用户名、密码
                final String name = ((EditText) loginDialog
                    .findViewById(R.id.name)).getText()
                    .toString();
                final String pass = ((EditText) loginDialog
                    .findViewById(R.id.pass)).getText()
                    .toString();
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        try
                        {
                            HttpPost post = new HttpPost("http://192.168"
                                + ".1.88:8888/foo/login.jsp");// ③
                            // 如果传递参数个数比较多，可以对传递的参数进行封装
                            List<NameValuePair> params = new ArrayList<>();
                            params.add(new BasicNameValuePair
                                ("name", name));
                            params.add(new BasicNameValuePair
                                ("pass", pass));
                            // 设置请求参数
                            post.setEntity(new UrlEncodedFormEntity(
                                params, HTTP.UTF_8));
                            // 发送 POST 请求
                            HttpResponse response = httpClient
                                .execute(post);  // ④
                            // 如果服务器成功地返回响应
                            if (response.getStatusLine()
                                .getStatusCode() == 200)
                            {
                                String msg = EntityUtils
                                    .toString(response.getEntity());
                                Looper.prepare();
                                // 提示登录成功
                                Toast.makeText(MainActivity.this,
```

```
                                msg, Toast.LENGTH_SHORT).show();
                            Looper.loop();
                        }
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                }
            }.start();
        }
    }).setNegativeButton("取消", null).show();
    }
}
```

上面程序中的①、②号代码先创建了一个 HttpGet 对象，接下来程序调用 HttpClient 的 execute()方法发送 GET 请求；程序中③、④号代码先创建了一个 HttpPost 对象，接下来程序调用了 HttpClient 的 execute()方法发送 POST 请求。上面的 GET 请求用于获取服务器上的被保护页面，POST 请求用于登录系统。

运行该程序，单击“访问页面”按钮，将可以看到如图 13.8 所示的页面。

提示：

运行该程序需要 Web 应用的支持，读者应该先将 codes/13/13.3 目录下的 foo 应用部署到 Web 服务器（如 Tomcat 7.0）中，然后再运行该应用。

从图 13.8 可以看出，程序直接向指定 Web 应用的被保护页面 secret.jsp 发送请求，程序将无法访问被保护页面，于是看到图 13.8 所示的页面。单击图 13.8 所示页面中的“登录系统”按钮，系统将会显示如图 13.9 所示的登录对话框。

在图 13.9 所示对话框的两个输入框中分别输入“crazyit.org”、“leegang”，然后单击“登录”按钮，系统将会向 Web 站点的 login.jsp 页面发送 POST 请求，并将用户输入的用户名、密码作为请求参数。如果用户名、密码正确，即可看到登录成功的提示。

登录成功后，HttpClient 将会自动维护与服务器之间的连接，并维护与服务器之间的 Session 状态，再次单击程序中的“访问页面”按钮，即可看到如图 13.10 所示的输出。

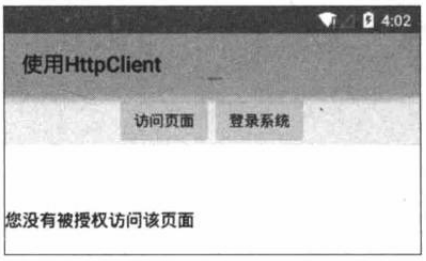


图 13.8 访问被保护页面

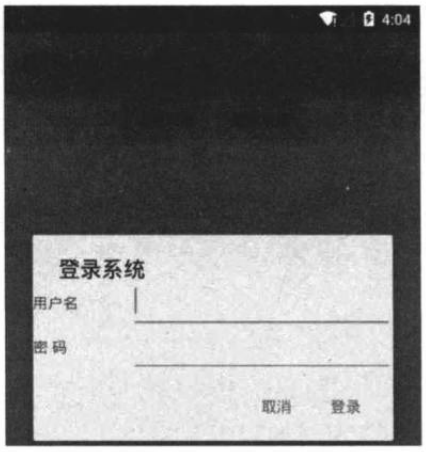


图 13.9 登录对话框

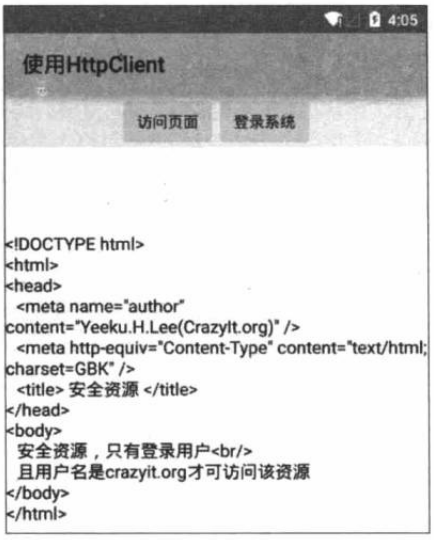


图 13.10 访问被保护资源

从图 13.10 可以看出，此时使用 HttpClient 发送 GET 请求即可正常访问被保护资源，这就是因为前面使用了 HttpClient 登录了系统，而且 HttpClient 可以维护与服务器之间的 Session

连接。

从上面的编程过程不难看出，使用 Apache 的 HttpClient 更加简单，而且它比 HttpURLConnection 提供了更多的功能。

13.4 Android 5.0 增强的 WebView

Android 提供了 WebView 组件，从表面上看，这个组件与普通 ImageView 差不多，但实际上这个组件的功能要强大得多，WebView 组件本身就是一个浏览器实现，Android 5.0 增强的 WebView 基于 Chromium M37，直接支持 WebRTC、WebAudio 和 WebGL。

Chromium M37 也包括对 Web 组件规范的原生支持，如自定义元素、阴影 DOM、HTML 导入和模板等，这意味着开发者可以直接在 WebView 中使用聚合（Polymer）和 Material 设计。

13.4.1 使用 WebView 浏览网页

WebView 的用法与普通 ImageView 组件的用法基本相似，它提供了大量方法来执行浏览器操作，例如如下常用方法。

- void goBack()：后退。
- void goForward()：前进。
- void loadUrl(String url)：加载指定 URL 对应的网页。
- boolean zoomIn()：放大网页。
- boolean zoomOut()：缩小网页。

当然，WebView 组件还包含了大量方法，具体以 Android API 文档为准。

下面的程序将基于 WebView 来开发一个简单的浏览器。

实例：迷你浏览器

该程序的界面中包含两个组件：一个文本框用于接收用户输入的 URL；一个 WebView 用于加载并显示该 URL 对应的页面。该程序的界面布局代码如下。

程序清单：codes\13\13.4\MiniBrowser\app\src\main\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="match_parent"
    android:layout_height="match_parent">
<EditText
    android:id="@+id/url"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    />
<!-- 显示页面的 WebView 组件 -->
<WebView
    android:id="@+id/show"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
</LinearLayout>
```

下面程序则主要通过 WebView 的 loadUrl(String url)来加载、显示指定 URL 对应的页面。

程序清单：codes\13\13.4\MiniBrowser\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    EditText url;
    WebView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取页面中文本框、WebView 组件
        url = (EditText) findViewById(R.id.url);
        show = (WebView) findViewById(R.id.show);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_SEARCH)
        {
            String urlStr = url.getText().toString();
            // 加载并显示 urlStr 对应的网页
            show.loadUrl(urlStr);
            return true;
        }
        return false;
    }
}
```

上面程序中的粗体字代码是该程序的关键，程序调用 WebView 的 loadUrl(String url)方法加载、显示该 URL 对应的网页——至于该 WebView 如何发送请求，如何解析服务器响应，这些细节对用户来说是透明的。

该应用需要访问互联网，同样需要在 AndroidManifest.xml 文件中增加如下配置：

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

运行该程序，在文本框中输入想访问的站点，并单击手机的“搜索”按钮，将可以看到如图 13.11 所示的输出。

图 13.11 使用 WebView 浏览指定网页

正如从图 13.11 所看到的，使用 WebView 开发浏览器十分简单，如果读者愿意多花时间对该程序界面进行美化，并为程序提供前进、后退、刷新等按钮，即可开发出一个实用的浏览器来代替 Android 系统自带的浏览器。

提示：

Android 系统自带的浏览器其实也是基于开源的 WebKit 引擎实现的。

13.4.2 使用 WebView 加载 HTML 代码

前面看到使用 EditText 显示 HTML 字符串时十分别扭，EditText 不会对 HTML 标签进行任何解析，而是直接把所有 HTML 标签都显示出来——就像用普通记事本显示一样；如果应用程序想重新对 HTML 字符串进行解析，当成 HTML 页面来显示，也是可以的。

WebView 提供了一个 loadData(String data, String mimeType, String encoding)方法，该方法

可用于加载并显示 HTML 代码。但在实际使用过程中，笔者发现这个方法有一个小问题：当它加载包含中文的 HTML 内容时，WebView 将会显示乱码。

好在 WebView 还提供了一个 loadDataWithBaseURL(String baseUrl, String data, String mimeType, String encoding, String historyUrl)方法，该方法是 loadData(String data, String mimeType, String encoding)方法的增强版，它不会产生乱码。关于该方法的几个参数简单说明一下。

- data：指定需要加载的 HTML 代码。
- mimeType：指定 HTML 代码的 MIME 类型，对于 HTML 代码可指定为 text/html。
- encoding：指定 HTML 代码编码所用的字符集。比如指定为 GBK。

下面的程序简单示范了如何使用 WebView 来加载 HTML 代码。

程序清单：codes\13\13.4\ViewHtml\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    WebView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序中的 WebView 组件
        show = (WebView) findViewById(R.id.show);
        StringBuilder sb = new StringBuilder();
        // 拼接一段 HTML 代码
        sb.append("<html>");
        sb.append("<head>");
        sb.append("<title> 欢迎您 </title>");
        sb.append("</head>");
        sb.append("<body>");
        sb.append("<h2> 欢迎您访问<a href=\"http://www.crazyit.org\">"
            + "疯狂 Java 联盟</a></h2>");
        sb.append("</body>");
        sb.append("</html>");
        // 使用简单的 loadData 方法会导致乱码，可能是 Android API 的 Bug
        // show.loadData(sb.toString() , "text/html" , "utf-8");
        // 加载并显示 HTML 代码
        show.loadDataWithBaseURL(null, sb.toString()
            , "text/html" , "utf-8", null);
    }
}
```

上面程序中的粗体字代码就是该程序的关键，这行代码负责加载指定的 HTML 页面，并将它显示出来。运行该程序，将可以看到如图 13.12 所示的输出。

图 13.12 使用 WebView 加载、显示 HTML 代码

13.4.3 使用 WebView 中的 JavaScript 调用 Android 方法

很多时候，WebView 加载的页面上是带 JavaScript 脚本的，比如页面上有一个按钮，用户单击按钮时将会弹出一个提示框，或打开一个列表框等。由于该按钮是 HTML 页面上的按钮，

它只能激发一段 JavaScript 脚本，这就需要让 JavaScript 脚本来调用 Android 方法了。

为了让 WebView 中的 JavaScript 脚本调用 Android 方法，WebView 提供了一个配套的 WebSettings 工具类，该工具类提供了大量方法来管理 WebView 的选项设置，其中它的 setJavaScriptEnabled(true)即可让 WebView 中的 JavaScript 脚本来调用 Android 方法。除此之外，为了把 Android 对象暴露给 WebView 中的 JavaScript 代码，WebView 提供了 addJavascriptInterface(Object object, String name)方法，该方法负责把 object 对象暴露成 JavaScript 中的 name 对象。

从上面的介绍可以看出，在 WebView 的 JavaScript 中调用 Android 方法只要如下三个步骤。

1. 调用 WebView 关联的 WebSettings 的 setJavaScriptEnabled(true)启用 JavaScript 调用功能。
2. 调用 WebView 的 addJavascriptInterface(Object object, String name)方法将 object 对象暴露给 JavaScript 脚本。
3. 在 JavaScript 脚本中通过刚才暴露的 name 对象调用 Android 方法。

提示：

如果读者有 DWR 开发经验，应该很好理解 Android 此处的设计，DWR 通过使用配置，可以让服务端的 Java 对象暴露给 JavaScript 脚本；Android 则通过 WebView 的 addJavascriptInterface()方法把 Android 应用中的对象暴露给 JavaScript 脚本——最后实现的效果是相同的：JavaScript 脚本可以直接调用 Java 对象的方法。

下面的示例示范了如何在 JavaScript 中调用 Android 方法。该示例的界面布局很简单，它包含了一个普通的 WebView 组件，用于显示 HTML 页面。该示例的 Activity 代码如下。

程序清单：codes\13\13.4\JsCallAndroid\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    WebView myWebView;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myWebView = (WebView) findViewById(R.id.webview);
        // 此处为了简化编程，使用 file 协议加载本地 assets 目录下的 HTML 页面
        // 如果有需要，也可使用 http 协议加载远程网站的 HTML 页面
        myWebView.loadUrl("file:///android_asset/test.html");
        // 获取 WebView 的设置对象
        WebSettings webSettings = myWebView.getSettings();
        // 开启 JavaScript 调用
        webSettings.setJavaScriptEnabled(true);
        // 将 MyObject 对象暴露给 JavaScript 脚本
        // 这样 test.html 页面中的 JavaScript 可以通过 myObj 来调用 MyObject 的方法
        myWebView.addJavascriptInterface(new MyObject(this), "myObj");
    }
}
```

上面程序中的第一行粗体字代码开启了 JavaScript 调用 Android 方法的功能，第二行粗体字代码则负责将 Androd 应用中的 MyObject 对象暴露给 JavaScript 脚本，暴露成 JavaScript 脚本中名为 myObj 的对象。

MyObject 是一个自定义的 Java 类，开发者可以根据业务需要提供任意多的方法，本示例

只为 MyObject 定义了两个方法。下面是 MyObject 类的代码。

程序清单：codes\13\13.4\JsCallAndroid\app\src\main\java\org\crazyit\net\MyObject.java

```
public class MyObject
{
    Context mContext;
    MyObject(Context c)
    {
        mContext = c;
    }
    // 该方法将会暴露给 JavaScript 脚本调用
    @JavascriptInterface
    public void showToast(String name)
    {
        Toast.makeText(mContext, name + ", 您好！"
            , Toast.LENGTH_LONG).show();
    }
    // 该方法将会暴露给 JavaScript 脚本调用
    @JavascriptInterface
    public void showList()
    {
        // 显示一个普通的列表对话框
        new AlertDialog.Builder(mContext)
            .setTitle("图书列表")
            .setIcon(R.drawable.ic_launcher)
            .setItems(new String[]{"疯狂 Java 讲义"
            , "疯狂 Android 讲义" , "轻量级 Java EE 企业应用实战"} , null)
            .setPositiveButton("确定", null)
            .create()
            .show();
    }
}
```

正如上面的代码所示，MyObject 中包含了两个方法——showToast()和 showList()方法，且这两个方法使用了@JavascriptInterface 修饰，因此这两个方法将会暴露给 JavaScript 脚本，从而允许 JavaScript 脚本通过 myObj 来调用这两个方法。下面是 HTML 页面的代码。

程序清单：codes\13\13.4\JsCallAndroid\app\src\main\assets\test.html

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title> Js 调用 Android </title>
</head>
<body>
<!-- 注意此处的 myObj 是 Android 暴露出来的对象 -->
<input type="button" value="打招呼"
    onclick="myObj.showToast('孙悟空');" />
<input type="button" value="图书列表"
    onclick="myObj.showList();" />
</body>
</html>
```

正如上面的两行粗体字代码所示，当用户单击该页面上的两个按钮时，该页面的 JavaScrip 脚本会通过 myObj 调用 Android 方法。运行该示例，单击第一个按钮，可以看到如图 13.13 所示的界面。

如果用户单击第二个按钮，该页面的 JavaScript 脚本将会通过 myObj 调用 Android 的 showList()方法，此时将看到如图 13.14 所示的对话框。

图 13.13 JS 调用 Android 方法

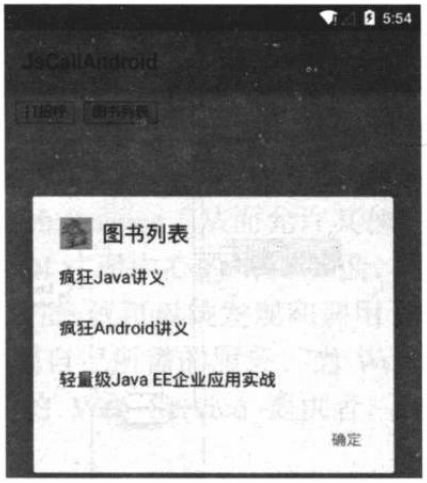


图 13.14 JS 调用 Android 方法

13.5 使用 Web Service 进行网络编程

Android 应用通常都运行在手机平台上，手机系统的硬件资源是有限的，不管是存储能力还是计算能力都有限，在 Android 系统上开发、运行一些单用户、小型应用是可能的，但对于需要进行大量的数据处理、复杂计算的应用，还是只能部署在远程服务器上，Android 应用将只是充当这些应用的客户端。

为了让 Android 应用与远程服务器进行交互，可以借助于 Java 的 RMI 技术，但这要求远程服务器程序必须采用 Java 实现；也可以借助于 CORBA 技术，但这种技术显得过于复杂。除此之外，Web Service 是一种不错的选择。

13.5.1 Web Service 平台概述

Web Service 平台主要涉及的技术有 SOAP（Simple Object Access Protocol，简单对象访问协议）、WSDL（Web Service Description Language，Web Service 描述语言）、UDDI（Universal Description，Description and Integration，统一描述、发现和整合协议）。

1. SOAP（简单对象访问协议）

SOAP 是一种具有扩展性的 XML 消息协议。SOAP 允许一个应用程序向另一个应用程序发送 XML 消息，SOAP 消息是从 SOAP 发送者传至 SOAP 接收者的单路消息，任何应用程序均可作为发送者或接收者。SOAP 仅定义消息结构和消息处理的协议，与底层的传输协议独立。因此，SOAP 协议能通过 HTTP、JMS 或 SMTP 协议传输。

SOAP 依赖于 XML 文档来构建，一条 SOAP 消息就是一份特定的 XML 文档，SOAP 消息包含如下三个主要元素。

- <Envelope.../>根元素，SOAP 消息对应的 XML 文档以该元素作为根元素。
- 可选的<Header../>元素，包含 SOAP 消息的头信息。
- 必需的<Body../>元素，包含所有的调用和响应信息。

就目前的 SOAP 消息的结构来看，<Envelope.../>根元素通常只能包含两个子元素，其中第一个子元素是可选的<Header../>元素，第二个子元素是必需的<Body../>元素。

图 13.15 显示了 SOAP 消息的基本结构。

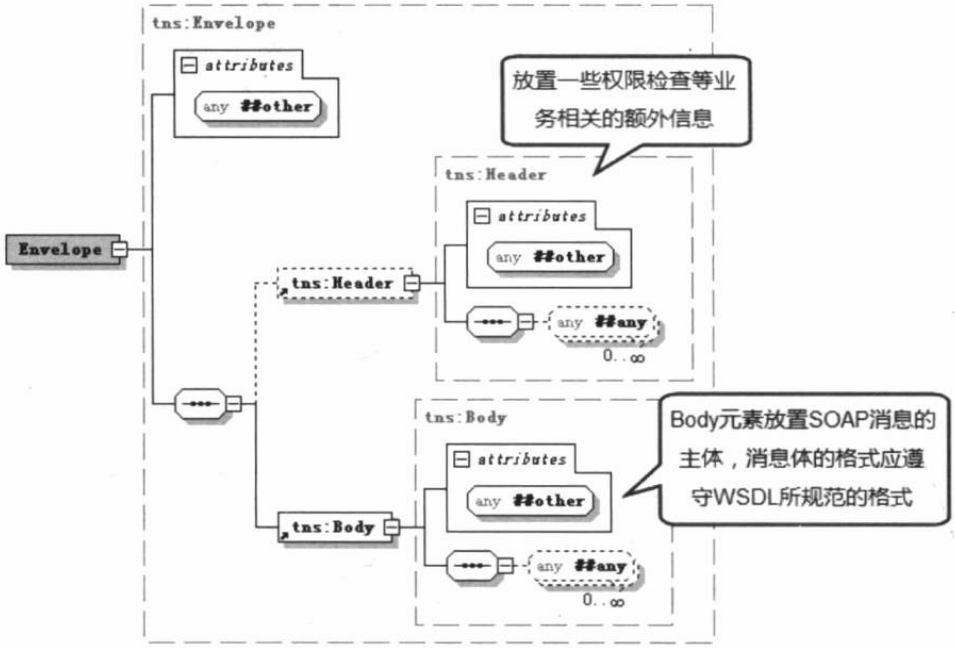


图 13.15 SOAP 消息的基本结构

2. WSDL（Web Service 描述语言）

WSDL 使用 XML 描述 Web Service，包括访问和使用 Web Service 所必需的信息，定义该 Web Service 的位置、功能及如何通信等描述信息。

一般来说，只要调用者能够获取 Web Service 对应的 WSDL，就可以从中了解它所提供的服务及如何调用 Web Service 了。因为一份 WSDL 文件清晰地定义了三个方面的内容。

- **WHAT 部分：**用于定义 Web Service 所提供的操作（或方法），也就是 Web Service 能做些什么。由 WSDL 中的<types.../>、<message.../>和<portType.../>元素定义。
- **HOW 部分：**用于定义如何访问 Web Service，包括数据格式详情和访问 Web Service 操作的必要协议。也就是定义了如何访问 Web Service。
- **WHERE 部分：**用于定义 Web Service 位于何处。该部分使用<service.../>元素定义，可在 WSDL 文件的最后部分看到<service.../>元素。

一份 WSDL 文档通常可分为两个部分。

- 第一个部分定义了服务接口，它在 WSDL 中由<message.../>和<portType.../>两个元素组成，其中<message.../>元素定义了操作的交互方式；而<portType.../>元素里则可包含任意数量的<operation.../>元素，每个<operation.../>元素代表一个允许远程调用的操作（即方法）。
- 第二个部分定义了服务实现，它在 WSDL 中由<binding.../>和<service.../>两个元素组成，其中<binding.../>元素定义使用特定的通信协议、数据编码模型和底层通信协议，将 Web Service 服务接口定义映射到具体实现；而<service.../>元素则包含一系列的<port.../>子元素，<port.../>子元素将会把绑定机制、服务访问协议和端点地址结合在一起。

图 13.16 显示了 WSDL 文档的结构模型。

3. UDDI（统一描述、发现和整合协议）

UDDI 是一套信息注册规范，它具有如下特点。

- 基于 Web。
- 分布式。

UDDI 包括一组允许企业向外注册 Web Service，以使其他企业发现访问的实现标准。UDDI 的核心组件是 UDDI 注册中心，它使用 XML 文件来描述企业及其提供的 Web Service。

通过使用 UDDI，Web Service 提供者可以对外注册 Web Service，从而允许其他企业来调用该企业注册的 Web Service。Web Service 提供者通过 UDDI 注册中心的 Web 界面，将它所提供的 Web Service 的信息加入 UDDI 注册中心，该 Web Service 就可以被发现和调用了。

Web Service 使用者也通过 UDDI 注册中心查找、发现自己所需的服务。当 Web Service 使用者找到自己所需的服务之后，可以将自己绑定到指定的 Web Service 提供者，再根据该 Web Service 对应的 WSDL 文档来调用对方的服务。

Web Service 大致的运行模式如图 13.17 所示。

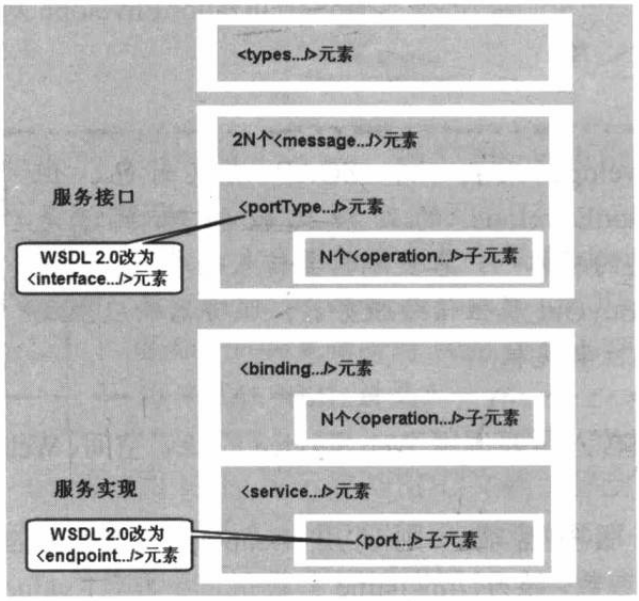


图 13.16 WSDL 文档的结构模型

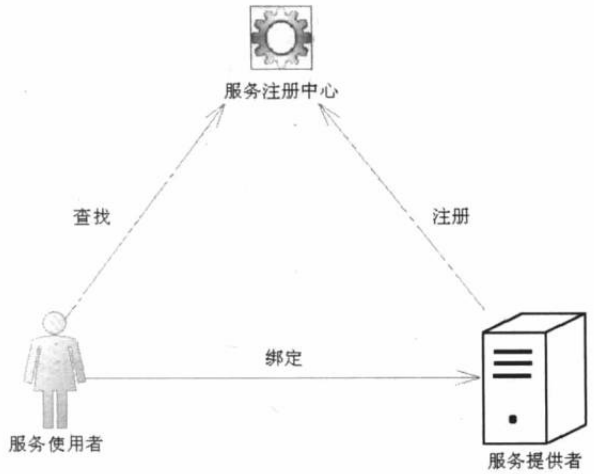


图 13.17 Web Service 运行模式示意图

13.5.2 使用 Android 应用调用 Web Service

Java 本身提供了丰富的 Web Service 支持，比如 Sun 公司制定的 JAX-WS 2 规范，还有 Apache 开源组织所提供的 Axis1、Axis2、CXF 等，这些技术不仅可以用于非常方便地对外提供 Web Service，也可以用于简化 Web Service 的客户端编程。

对于手机等小型设备而言，它们的计算资源、存储资源都十分有限，因此 Android 应用不大可能需要对外提供 Web Service，Android 应用通常只是充当 Web Service 的客户端，调用远程 Web Service。

Google 为 Android 平台开发 Web Service 客户端提供了 ksoap2-android 项目，但这个项目并未直接集成在 Android 平台中，还需要开发人员自行下载。

为 Android 应用增加 ksoap2-android 支持请按如下步骤进行。

① 登录 http://code.google.com/p/ksoap2-android/站点，该站点有介绍下载 ksoap2-android 项目的方法。

② 下载 ksoap2-android 项目的 ksoap2-android-assembly-3.4.0-jar-with-dependencies.jar 包。如果读者下载有困难，也可直接使用光盘中 codes\13\13.5 目录下的该文件。

③ 将下载得到的 JAR 包添加到 Android 项目的 libs 目录下，并通过 Project 面板选中该

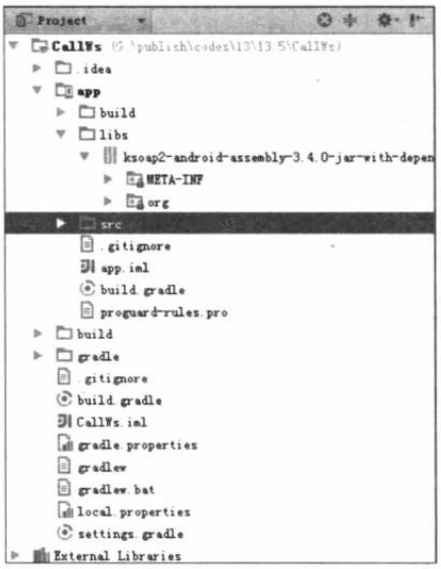


图 13.18　为 Android 项目添加 ksoap2-android 包

JAR 包后，通过右键菜单的“Add as library”菜单项添加该 JAR 包，即可在 Android Studio 左边看到如图 13.18 所示的项目管理树。

为 Android 项目添加了 ksoap2-android 包之后，接下来借助于 ksoap2-android 项目来调用 Web Service 所暴露的操作。

使用 ksoap2-android 调用 Web Service 操作的步骤如下。

① 创建 HttpTransportSE 对象，该对象用于调用 Web Service 操作。

② 创建 SoapSerializationEnvelope 对象。

提示：

从名称来看，SoapSerializationEnvelope 代表一个 SOAP 消息封包；但 ksoap2-android 项目对 SoapSerializationEnvelope 的处理比较特殊，它是 HttpTransportSE 调用 Web Service 时信息的载体——客户端需要传入的参数，需要通过 SoapSerializationEnvelope 对象的 bodyOut 属性传给服务器；服务器响应生成的 SOAP 消息也通过该对象的 bodyIn 属性来获取。

③ 创建 SoapObject 对象，创建该对象时需要传入所要调用 Web Service 的命名空间、Web Service 方法名。

④ 如果有参数需要传给 Web Service 服务器端，则调用 SoapObject 对象的 addProperty(String name , Object value)方法来设置参数，该方法的 name 参数指定参数名；value 参数指定参数值。

⑤ 调用 SoapSerializationEnvelope 的 setOutputSoapObject()方法，或者直接对 bodyOut 属性赋值，将前两步创建的 SoapObject 对象设为 SoapSerializationEnvelope 的传出 SOAP 消息体。

⑥ 调用对象的 call()方法，并以 SoapSerializationEnvelope 作为参数调用远程 Web Service。

⑦ 调用完成后，访问 SoapSerializationEnvelope 对象的 bodyIn 属性，该属性返回一个 SoapObject 对象，该对象就代表了 Web Service 的返回消息。解析该 SoapObject 对象，即可获取调用 Web Service 的返回值。

实例：调用基于 CXF 的 Web Service

下面的程序使用 CXF 开发了一个 Web Service，该 Web Service 对应的 WSDL 文档如图 13.19 所示。

提示：

使用 Ant 通过光盘中 codes\13\13.5\WsServer 目录下的 build.xml 文件运行 Web Service 服务器端——当服务器端运行起来之后才可通过浏览器看到如图 13.19 所示的 WSDL 文档。

正如图 13.19 所看到的，Web Service 客户端只要访问到 Web Service 的 WSDL 文档，即可

根据该文档来获取调用 Web Service 操作的必要信息。

图 13.19 Web Service 提供的 WSDL 文档

提示：

由于本书的重点是介绍 Android 应用调用 Web Service，因此不会涉及如何使用 CXF 开发 Web Service，以及 Web Service 的 WSDL 文档中各元素的作用、意义等知识；如果读者对 Web Service 服务器端的开发有兴趣，可以参考疯狂 Java 体系的《疯狂 XML 讲义》一书。

通过图 13.19 所示的 WSDL 文档了解到调用 Web Service 的方法名、所需的参数名之后，接下来就可以在 Android 程序中通过 ksoap2-android 调用 Web Service 操作了。

下面的程序示范了如何通过 ksoap2-android 来调用 Web Service 操作。该程序的界面很简单，界面中只定义了两个文本框来装载服务器响应，因此此处不再给出界面布局代码。该程序的 Activity 代码如下。

程序清单：codes\13\13.5\CallWs\app\src\main\java\org\crazyit\net\MainActivity.java

```
public class MainActivity extends Activity
{
    final static String SERVICE_NS = "http://ws.app.crazyit.org/";
    final static String SERVICE_URL = "http://192.168.1.88:9999/crazyit";
    private EditText txt1;
    private EditText txt2;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            switch(msg.what)
            {
                case 0x123:
                    txt1.setText(msg.obj.toString());
                    break;
                case 0x234:
                    txt2.setText(msg.obj.toString());
                    break;
            }
        }
    };
```

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    txt1 = (EditText) findViewById(R.id.txt1);
    txt2 = (EditText) findViewById(R.id.txt2);
    // 调用的方法
    String methodName = "getUserList";
    // 创建 HttpTransportSE 传输对象
    final HttpTransportSE ht = new HttpTransportSE(SERVICE_URL); // ①
    ht.debug = true;
    // 使用 SOAP 1.1 协议创建 Envelop 对象
    final SoapSerializationEnvelope envelope = new
        SoapSerializationEnvelope(SoapEnvelope.VER11); // ②
    // 实例化 SoapObject 对象
    SoapObject soapObject = new SoapObject(SERVICE_NS, methodName); // ③
    soapObject.addProperty("arg0", "客户端参数:"); // ④
    // 将 soapObject 对象设置为 SoapSerializationEnvelope 对象的传出 SOAP 消息
    envelope.bodyOut = soapObject; // ⑤
    new Thread()
    {
        public void run()
        {
            try
            {
                // 调用 Web Service
                ht.call(null, envelope); // ⑥
                if (envelope.getResponse() != null)
                {
                    // 获取服务器响应返回的 SOAP 消息
                    SoapObject result = (SoapObject) envelope.bodyIn; // ⑦
                    // 接下来就是从 SoapObject 对象中解析响应数据的过程了
                    SoapObject detail1 = (SoapObject) result
                        .getProperty(0);
                    SoapObject detail2 = (SoapObject) result
                        .getProperty(1);
                    StringBuilder person1 = new StringBuilder();
                    person1.append("用户名: ");
                    person1.append(detail1.getProperty(3));
                    person1.append("\n 密码");
                    person1.append(detail1.getProperty(0));
                    person1.append("\n 身高: ");
                    person1.append(detail1.getProperty(1));
                    Message msg = new Message();
                    msg.what = 0x123;
                    msg.obj = person1.toString();
                    handler.sendMessage(msg);
                    StringBuilder person2 = new StringBuilder();
                    person2.append("用户名: ");
                    person2.append(detail2.getProperty(3));
                    person2.append("\n 密码: ");
                    person2.append(detail2.getProperty(0));
                    person2.append("\n 身高: ");
                    person2.append(detail2.getProperty(1));
                    Message msg2 = new Message();
                    msg2.what = 0x234;
                    msg2.obj = person2.toString();
                    handler.sendMessage(msg2);
                }
            }
            catch (IOException e)
```

```
                {
                    e.printStackTrace();
                }
                catch (XmlPullParserException e)
                {
                    e.printStackTrace();
                }
            }
        }.start();
    }
}
```

上面程序中的粗体字代码就代表了使用 ksoap2-android 来调用 Web Service 操作的关键的 7 个步骤。

Web Service 服务器端运行起来之后，运行上面的 Android 应用，即可看到如图 13.20 所示的输出。

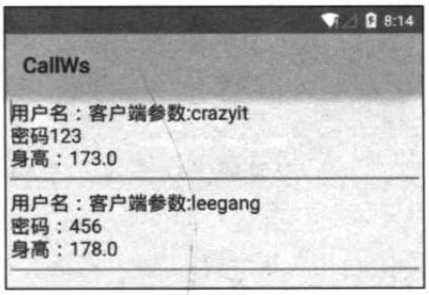


图 13.20 调用 Web Service

从图 13.20 所示的两个文本框中看到的内容就是调用 Web Service 所返回的数据。由此可见，不管远程 Web Service 提供的服务功能多么强大、业务实现多么复杂，对于 Android 客户端是完全透明的，Android 只要送出相应的请求参数，服务器就会返回包含结果的 SOAP 消息。借助于 Web Service 这个桥梁，在 Android 应用中就可以实现功能非常强大的应用——反正具体的业务逻辑由 Web Service 服务器端实现，Android 客户端只要调用 Web Service 服务即可。

不管远程 Web Service 的功能多么复杂，对于 Android 客户端而言，它只要通过 Web Service 获取对方提供的数据，并将这些数据“整合”到自己的应用中即可，因此 Web Service 为 Android 应用提供了强大的后台支持。

13.6 本章小结

本章主要介绍了 Android 网络编程的相关知识。由于 Android 完全支持 JDK 网络编程中的 ServerSocket、Socket、DatagramSocket、Datagrampacket、MulticastSocket 等 API，也支持 JDK 内置的 URL、URLConnection、HttpURLConnection 等工具类，因此如果读者已经具有网络编程的经验，这些经验完全适用于 Android 网络编程。除此之外，Android 还内置了 Apache HttpClient 支持，通过 HttpClient 可以非常方便地维持与服务器的会话状态、发送请求、获取响应等。虽然 Android 本身没有内置 Web Service 支持，但本章介绍了通过 ksoap2-android 项目调用远程 Web Service 的相关内容，这也是需要读者掌握的内容。

CHAPTER

14

第14章 管理 Android 手机桌面

本章要点

- Android 手机桌面的概念
- 改变手机壁纸的 API
- 动态壁纸的 API
- 开发动态壁纸
- 向 Android 桌面添加快捷方式
- 桌面控件的概念
- 添加桌面控件
- 带数据集的桌面控件
- RemoteViewsService 和 RemoteViewsFactory

Android 系统提供了一个桌面——也就是用户启动后第一次看到的界面，如图 14.1 所示。从图 14.1 可以看出，手机桌面的作用类似于 PC 的桌面，桌面上通常用于放置一些常用的程序和功能。

在 Android 桌面上首先看到的壁纸，也就是手机桌面上的那张图片，接着可以看到手机桌面上规则排列的多个图标，这些图标就是 Android 桌面组件，分别代表快捷方式与桌面控件；每个快捷方式只占用桌面的一个摆放位置；桌面控件则可以很大，一个桌面控件就可以占据多个摆放位置。

Android 系统提供了很好的可扩展性，开发者完全可以在程序中管理 Android 桌面，包括改变系统壁纸、管理快捷方式与创建桌面控件等。

图 14.1 Android 桌面

14.1 管理手机桌面

在默认情况下，Android 系统的桌面上除了一张壁纸之外，并不会显示如图 14.1 所示的组件。Android 系统允许普通用户动态地添加、删除桌面组件。

14.1.1 删除桌面组件

在默认情况下，Android 系统的桌面上会显示两个桌面组件：相机和模拟时钟，模拟时钟就是搜索条下面的组件。一旦熟悉了 Android 系统之后，用户可能并不希望总是看到这两个组件，此时可能会考虑删除该组件。

删除 Android 桌面组件的步骤如下。

1. 在屏幕上长按指定组件，直到桌面上方出现“删除”图标。
2. 将指定组件拖到桌面上方的“删除”图标上，如图 14.2 所示。

14.1.2 添加桌面组件

Android 桌面组件可分为快捷方式与桌面控件两种，两种组件的添加方式略有不同。

为了在 Android 桌面上添加快捷方式，可以按如下步骤进行。

1. 进入手机的程序列表界面，长按需要添加快捷方式的程序，此时可以看到程序图标会自动切换到桌面，如图 14.3 所示。

图 14.2 删除桌面组件

图 14.3 添加快捷方式

② 拖动该程序图标，将它放到指定位置即可。

为了在 Android 桌面上添加桌面控件，可以按如下步骤进行。

① 长按手机桌面，然后单击屏幕右下角的“WIDGETS”按钮，系统进入如图 14.4 所示的 Widget 列表界面。

② 长按需要添加的桌面控件，此时可以看到桌面控件会自动切换到桌面，如图 14.5 所示。

图 14.4　桌面控件列表

图 14.5　添加桌面控件

③ 拖动该桌面控件，将它放到指定位置即可。

Android 系统提供的上面的操作能让用户动态地管理手机桌面，但这些能在桌面显示的组件都是 Android 系统本身提供的，接下来将会介绍如何通过应用程序来管理桌面组件。

14.2　改变手机壁纸

前面讲解系统服务时已经介绍了一个“定时更换壁纸”的应用，在那个应用中，当用户启动相应的 Service 之后，该 Service 将会根据 AlarmManager 来定时改变手机壁纸。

Android 允许使用 WallpaperManager 来改变手机壁纸，该对象中改变手机壁纸的方法如下。

- setBitmap(Bitmap bitmap)：将壁纸设置为 bitmap 所代表的位图。
- setResource(int resid)：将壁纸设置为 resid 资源所代表的图片。
- setStream(InputStream data)：将壁纸设置为 data 数据所代表的图片。

前面介绍的这种改变手机壁纸的方式只是动态地切换不同图片作为手机壁纸，此处不再介绍调用 WallpaperManager 来改变手机壁纸的示例。

除此之外，Android 系统还提供了一种动态壁纸的功能，例如用户在手机桌面上长按，系统将会显示如图 14.6 所示的更改壁纸的方式。

在 Android 手机屏幕下方会看到大量系统内置的壁纸，其中前面几项是普通的静态图片壁纸，后面则包含了图 14.6 所示的“动态壁纸”列表项，这是 Android 系

图 14.6　动态壁纸

统默认提供的几个动态壁纸。除此之外，开发者还可以开发任意的动态壁纸。

14.2.1 开发动态壁纸（Live Wallpapers）

所谓动态壁纸，就是指手机桌面不再是简单的图片，而是运行中的动画，这个动画是由程序实时绘制的，因此被称为动态壁纸。

为了开发动态壁纸，Android 提供了 WallpaperService 基类，动态壁纸的实现类需要继承该基类。在 Android 应用中开发动态壁纸的步骤如下。

1. 开发一个子类继承 WallpaperService 基类。
2. 继承 WallpaperService 基类时必须重写 onCreateEngine() 方法，该方法返回 WallpaperService.Engine 子类对象。
3. 开发者需要实现 WallpaperService.Engine 子类，并重写其中的 public void onVisibilityChanged (boolean visible)、public void onOffsetsChanged()方法。不仅如此，由于 WallpaperService.Engine 子类采用了与 SurfaceView 相同的绘图机制，因此还可有选择性地重写 SurfaceHolder.Callback 中的三个方法。重写这些方法时可通过 SurfaceHolder 动态地绘制图形。

实例：蜿蜒壁纸

本实例将通过一个不断变换的矩形在桌面上绘制动态壁纸。下面的 LiveWallPaper 类就代表了动态壁纸服务，程序代码如下。

程序清单：codes\14\14.2\LiveWallPaper\app\src\main\java\org\crazyit\desktop\LiveWallpaper.java

```
public class LiveWallpaper extends WallpaperService
{
    // 记录用户触碰点的位图
    private Bitmap heart;
    // 实现 WallpaperService 必须实现的抽象方法
    @Override
    public Engine onCreateEngine()
    {
        // 加载心形图片
        heart = BitmapFactory.decodeResource(getResources()
            , R.drawable.heart);
        // 返回自定义的 Engine
        return new MyEngine();
    }
    class MyEngine extends Engine
    {
        // 记录程序界面是否可见
        private boolean mVisible;
        // 记录当前用户动作事件的发生位置
        private float mTouchX = -1;
        private float mTouchY = -1;
        // 记录当前需要绘制的矩形的数量
        private int count = 1;
        // 记录绘制第一个矩形所需坐标变换的 X、Y 坐标的偏移
        private int originX = 50 , originY = 50;
        // 定义画笔
        private Paint mPaint = new Paint();
        // 定义一个 Handler
        Handler mHandler = new Handler();
        // 定义一个周期性执行的任务
        private final Runnable drawTarget = new Runnable()
        {
```

```
        public void run()
        {
            drawFrame();
        }
    };
    @Override
    public void onCreate(SurfaceHolder surfaceHolder)
    {
        super.onCreate(surfaceHolder);
        // 初始化画笔
        mPaint.setARGB(76 , 0, 0 , 255);
        mPaint.setAntiAlias(true);
        mPaint.setStyle(Paint.Style.FILL);
        // 设置处理触摸事件
        setTouchEventsEnabled(true);
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 删除回调
        mHandler.removeCallbacks(drawTarget);
    }
    @Override
    public void onVisibilityChanged(boolean visible)
    {
        mVisible = visible;
        // 当界面可见时，执行 drawFrame()方法
        if (visible)
        {
            // 动态地绘制图形
            drawFrame();
        }
        else
        {
            // 如果界面不可见，删除回调
            mHandler.removeCallbacks(drawTarget);
        }
    }
    @Override
    public void onOffsetsChanged(float xOffset, float yOffset, float xStep,
        float yStep, int xPixels, int yPixels)
    {
        drawFrame();
    }
    @Override
    public void onTouchEvent(MotionEvent event)
    {
        // 如果检测到滑动操作
        if (event.getAction() == MotionEvent.ACTION_MOVE)
        {
            mTouchX = event.getX();
            mTouchY = event.getY();
        }
        else
        {
            mTouchX = -1;
            mTouchY = -1;
        }
        super.onTouchEvent(event);
    }
    // 定义绘制图形的工具方法
```

```
private void drawFrame()
{
    // 获取该壁纸的 SurfaceHolder
    final SurfaceHolder holder = getSurfaceHolder();
    Canvas c = null;
    try
    {
        // 对画布加锁
        c = holder.lockCanvas();
        if (c != null)
        {
            // 绘制背景色
            c.drawColor(0xffffffff);
            // 在触碰点绘制心形
            drawTouchPoint(c);
            // 设置画笔的透明度
            mPaint.setAlpha(76);
            c.translate(originX, originY);
            // 采用循环绘制 count 个矩形
            for (int i = 0; i < count; i++)  // ①
            {
                c.translate(80, 0);
                c.scale(0.95f, 0.95f);
                c.rotate(20f);
                c.drawRect(0, 0, 150, 75, mPaint);
            }
        }
    }
    finally
    {
        if (c != null) holder.unlockCanvasAndPost(c);
    }
    mHandler.removeCallbacks(drawTarget);
    // 调度下一次重绘
    if (mVisible)
    {
        count ++;
        if(count >= 50)
        {
            Random rand = new Random();
            count = 1;
            originX += (rand.nextInt(60) - 30);
            originY += (rand.nextInt(60) - 30);
            try
            {
                Thread.sleep(500);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
        // 指定 0.1 秒后重新执行 drawTarget 一次
        mHandler.postDelayed(drawTarget, 100);  // ②
    }
}
// 在屏幕触碰点绘制圆圈
private void drawTouchPoint(Canvas c)
{
    if (mTouchX >= 0 && mTouchY >= 0)
    {
```

```
                // 设置画笔的透明度
                mPaint.setAlpha(255);
                c.drawBitmap(heart , mTouchX, mTouchY, mPaint);
            }
        }
    }
}
```

上面程序中的粗体字代码就是实现动态壁纸 Service 的关键代码。这两段粗体字代码重写了 WallpaperService.Engine 的 onVisibilityChanged()、onOffsetsChanged()方法，并指定当桌面显示时调用 drawFrame()方法进行绘制，drawFrame()方法绘制完成后通过 Handler 对象指定 0.1 秒后重绘，如②号粗体字代码所示。

上面的程序为了实现“蜿蜒前行”的效果，使用循环控制绘制了 count 个矩形，每绘制一次，count 的值将会加 1；而且程序控制对 Canvas 进行位移、旋转两种坐标变换，通过这种方式即可实现“蜿蜒前行”的动画效果。

定义了该 Service 类之后，接下来还需要在 AndroidManifest.xml 文件中配置该 Service。配置动态壁纸 Service 与配置普通 Service 存在小小的区别，它需要指定如下两项。

- 指定运行动态壁纸，需要 android.permission.BIND_WALLPAPER 权限。
- 为动态壁纸指定 meta-data 配置。

在 AndroidManifest.xml 文件中配置动态壁纸，也就是需要增加如下配置片段。

程序清单：codes\14\14.2\LiveWallPaper\app\src\main\AndroidManifest.xml

```
<!-- 配置动态壁纸 Service -->
<service android:label="@string/app_name"
    android:name=".LiveWallpaper"
    android:permission="android.permission.BIND_WALLPAPER">
    <!-- 为动态壁纸配置 intent-filter -->
    <intent-filter>
        <action android:name="android.service.wallpaper.WallpaperService" />
    </intent-filter>
    <!-- 为动态壁纸配置 meta-data -->
    <meta-data android:name="android.service.wallpaper"
        android:resource="@xml/livewallpaper" />
</service>
```

上面的配置文件中粗体字代码就是配置动态壁纸的关键代码。上面的配置文件中指定了动态壁纸的 meta-data 放在@xml/livewallpaper 中定义，因此程序还需要在 res\xml\目录下增加一个 livewallpaper.xml 文件，该文件内容如下。

程序清单：codes\14\14.2\LiveWallPaper\app\src\main\res\xml\livewallpaper.xml

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

把上面的程序部署到模拟器上，该程序不能直接运行，需要按前面介绍的步骤来设置动态壁纸。

用户也可通过 Android 手机的 Settings 应用来设置动态壁纸，用户通过单击“Settings”→“Display”→“Wallpaper”→“Live Wallpapers”即可看到如图 14.7 所示的列表项。

单击“蜿蜒壁纸”项，系统进入预览该动态壁纸的界面，如图 14.8 所示。

在图 14.8 所示的界面中可以看到“蜿蜒前行”的动画不停执行，单击“Set wallpaper”按钮，即可应用我们刚刚所开发的动态壁纸程序。再次切换到 Android 系统桌面，将可看到桌面上显示如图 14.9 所示的效果。

图 14.7 选择动态壁纸

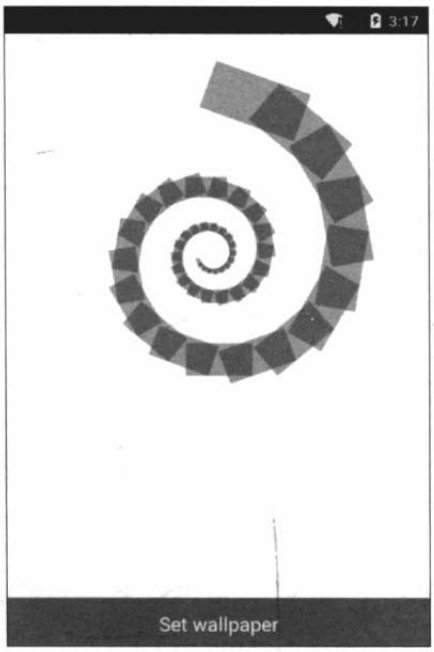


图 14.8 预览动态壁纸

图 14.9 动态壁纸效果

虽然上面的程序介绍的动态壁纸只是在桌面上绘制蜿蜒前行的矩形，但是这种动态壁纸可以提供开发者在桌面上自由绘图的能力，因此到底要在系统桌面上绘制什么，完全取决于用户自己的选择。

14.3 通过程序添加快捷方式

对于一个希望拥有更多用户的应用来说，用户桌面可以说是所有软件的必争之地，如果用户在手机桌面上建立了该软件的快捷方式，用户将会更频繁地使用该软件。因此，所有的 Android 程序都应该允许用户把软件的快捷方式添加到桌面上。

在程序中把一个软件的快捷方式添加到桌面上，只需要如下三步即可。

1. 创建一个添加快捷方式的 Intent，该 Intent 的 Action 属性值应该为 com.android.launcher.action.INSTALL_SHORTCUT。
2. 通过为该 Intent 添加 Extra 属性来设置快捷方式的标题、图标及快捷方式对应启动的程序。
3. 调用 sendBroadcast()方法发送广播即可添加快捷方式。

提示：

就像 Windows 时代，用户桌面是“兵家”必争之地，所有应用总是试图在用户桌面上创建自己的快捷方式，这样就可以让用户时时刻刻看到自己的程序，让用户经常使用自己的程序，从而依赖自己的程序。但需要注意的是，也有些用户对这种行为比较反感，他们喜欢自己的桌面简简单单，因此他们会把这些程序添加的图标删除，甚至会直接把程序卸载。所以希望读者注意，不要太激进，引起用户反感会导致程序被删除。

实例：让程序占领桌面

下面的程序是对第 7 章中一个动画程序的修改。该程序提供了一个按钮，用户单击该按钮即可在桌面上建立该程序的快捷方式。程序代码如下。

程序清单：codes\14\14.3\AddShortcut\app\src\main\java\org\crazyit\desktop\MainActivity.java

```
public class MainActivity extends Activity
{
    ImageView flower;
    // 定义两份动画资源
    Animation anim, reverse;
    final Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                flower.startAnimation(reverse);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        flower = (ImageView) findViewById(R.id.flower);
        // 加载第一份动画资源
        anim = AnimationUtils.loadAnimation(this, R.anim.anim);
        // 设置动画结束后保留结束状态
        anim.setFillAfter(true);
        // 加载第二份动画资源
        reverse = AnimationUtils.loadAnimation(this, R.anim.reverse);
        // 设置动画结束后保留结束状态
        reverse.setFillAfter(true);
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮的单击事件添加监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建添加快捷方式的 Intent
                Intent addIntent = new Intent(
                    "com.android.launcher.action.INSTALL_SHORTCUT"); // ①
                String title = getResources().getString(R.string.title);
                // 加载快捷方式的图标
                Parcelable icon = Intent.ShortcutIconResource.fromContext(
                    MainActivity.this, R.drawable.ic_launcher);
                // 创建点击快捷方式后对应的 Intent，该处指定当点击创建的快捷方式后
                // 将再次启动该程序
                Intent myIntent = new Intent(MainActivity.this,
                    MainActivity.class);
                // 设置快捷方式的标题
                addIntent.putExtra(Intent.EXTRA_SHORTCUT_NAME, title); // ②
                // 设置快捷方式的图标
                addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE
                    , icon); // ②
                // 设置快捷方式对应的 Intent
                addIntent.putExtra(Intent.EXTRA_SHORTCUT_INTENT
                    , myIntent); // ②
                // 发送广播添加快捷方式
                sendBroadcast(addIntent); // ③
            }
        });
```

```
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 开始执行动画
        flower.startAnimation(anim);
        // 设置 3.5 秒后启动第二个动画
        new Timer().schedule(new TimerTask()
        {
            @Override
            public void run()
            {
                handler.sendEmptyMessage(0x123);
            }
        }, 3500);
    }
}
```

上面程序中的粗体字代码就是为程序添加快捷方式的关键代码，其中①、②、③号粗体字代码就是前面所介绍的第 1~3 步。

在程序中添加快捷方式需要相应的权限，因此别忘了在 AndroidManifest.xml 文件中添加如下配置片段：

```
<!-- 指定添加快捷方式的权限 -->
<uses-permission android:name=
    "com.android.launcher.permission.INSTALL_SHORTCUT"/>
```

运行该程序，并单击该程序界面中的“添加快捷键”按钮，即可在桌面上添加一个快捷方式。返回桌面，即可在桌面上看到如图 14.10 所示的快捷方式。

上面的实例把添加快捷方式的代码放在事件处理方法中完成，这表明只有当用户单击该按钮时才会添加快捷方式——这种方式比较温和；如果开发者希望强行在用户桌面上添加快捷方式，则可以将上面添加快捷方式的代码放在 onCreate()方法中——这种方式容易引起用户反感。

图 14.10 快捷方式

14.4 管理桌面控件

所谓桌面控件，就是指能直接显示在 Android 系统桌面上的小程序，比如在图 14.1 中所看到的直接显示在桌面上的模拟时钟。一般来说，开发者可以把一些用户使用十分频繁的程序，比如时钟、指南针、日历等程序做成桌面控件，这样用户就可以直接在桌面上看到程序的运行界面了。

14.4.1 开发桌面控件

桌面控件是通过 BroadcastReceiver 的形式来进行控制的，因此每个桌面控件都对应于一个 BroadcastReceiver。为了简化桌面控件的开发，Android 系统提供了一个 AppWidgetProvider 类，它就是 BroadcastReceiver 的子类。也就是说，开发者开发桌面控件只要继承 AppWidgetProvider 类即可。

为了开发桌面控件，开发者只要开发一个继承 AppWidgetProvider 的子类，并重写

AppWidgetProvider不同状态的生命周期方法即可。AppWidgetProvider里提供了如下4个不同的生命周期方法。

- onUpdate()：负责更新桌面控件的方法；实现桌面控件通常会考虑重写该方法。
- onDeleted()：当一个或多个桌面控件被删除时回调该方法。
- onEnabled()：当接收到ACTION_APPWIDGET_ENABLED Broadcast时回调该方法。
- onDisabled()：当接收到ACTION_APPWIDGET_DISABLED Broadcast时回调该方法。

一般来说，开发桌面控件只需要定义一个AppWidgetProvider的子类，并重写它的onUpdate()方法即可。重写该方法按如下步骤进行。

1. 创建一个RemoteViews对象，创建该对象时可以指定加载指定的界面布局文件。
2. 如果需要改变上一步所加载的界面布局文件的内容，则可通过RemoteViews对象进行修改。

提示：

一般来说，RemoteViews所加载的界面中主要包含ImageView和TextView两种组件，RemoteViews提供了修改这两种组件的内容的方法。

3. 创建一个ComponentName对象。
4. 调用AppWidgetManager更新桌面控件。

提示：

将上面4个步骤归纳起来，其核心代码就是使用AppWidgetManager通过RemteViews来更新AppWidgetProvider的子类实例(需将它包装成ComponentName对象)。

下面的示例程序中的桌面控件十分简单，该控件中只是包含了一张简单的图片，此处不再给出界面布局文件。添加桌面控件的代码如下。

程序清单：codes\14\14.4\DesktopApp\app\src\main\java\org\crazyit\desktop\DesktopApp.java

```
public class DesktopApp extends AppWidgetProvider
{
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds)
    {
        // 加载指定界面布局文件，创建RemoteViews对象
        RemoteViews remoteViews = new RemoteViews(
            context.getPackageName(),
            R.layout.main);  // ①
        // 为show ImageView设置图片
        remoteViews.setImageViewResource(R.id.show
            , R.drawable.logo);   // ②
        // 将AppWidgetProvider的子类实例包装成ComponentName对象
        ComponentName componentName = new ComponentName(
            context, DesktopApp.class);  // ③
        // 调用AppWidgetManager将remoteViews添加到ComponentName中
        appWidgetManager.updateAppWidget(componentName
            , remoteViews);  // ④
    }
}
```

上面的程序中重写了AppWidgetProvider的onUpdate()方法，程序中①、②、③、④号粗

体字代码正好对应于上面所介绍的 4 个步骤。

由于 AppWidgetProvider 继承了 BroadcastReceiver，因此 AppWidgetProvider 的本质还是一个 BroadcastReceiver，为此需要在 AndroidManifest.xml 文件中使用<receiver.../>元素来配置它，配置该元素时需要为它指定相应的<intent-filter.../>和<meta-data.../>。

为了在系统中添加该桌面控件，还需要在 AndroidManifest.xml 文件中添加如下配置片段：

```
<receiver android:name="DesktopApp"
    android:label="@string/app_name">
    <!-- 将该 BroadcastReceiver 当成桌面控件 -->
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <!-- 指定桌面控件的 meta-data -->
    <meta-data android:name="android.appwidget.provider"
        android:resource="@xml/appwidget_provider"/>
</receiver>
```

上面配置文件中的粗体字代码指定该桌面控件使用@xml/appwidget_provider 作为 meta-data，因此还需要在应用的 res\xml 目录下添加 appwidget_provider.xml 文件。该文件的内容如下。

程序清单：codes\14\14.4\DesktopApp\app\src\main\res\xml\appwidget_provider.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 指定该桌面组件的基本配置信息：
    minWidth：桌面控件的最小宽度
    minWidth：桌面控件的最小高度
    updatePeriodMillis：更新频率
    initialLayout：初始时显示的布局 -->
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="150dip"
    android:minHeight="70dip"
    android:updatePeriodMillis="1000"
    android:initialLayout="@layout/main"/>
```

上面的 XML 配置文件中使用了<appwidget-provider.../>元素来描述桌面控件的基本信息，其中的注释已经详细说明了各属性的作用。

把该应用安装到 Android 系统上，再次进入图 14.4 所示的 Widget 列表界面，将会看到如图 14.11 所示的界面。

正如图 14.11 所示，刚才开发的桌面控件已经显示出来了，长按该列表项就会把该小控件添加到桌面上。再次返回桌面，将会看到如图 14.12 所示的桌面控件。

图 14.11 添加桌面控件

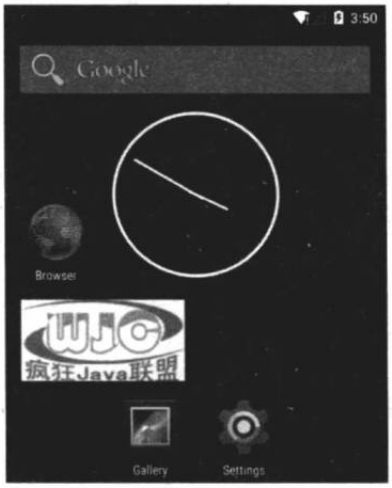


图 14.12 桌面控件

从图 14.12 可以看出，通过使用桌面控件，开发者可以把应用程序的运行界面直接放到桌面上，至于如何控制程序界面上显示的内容，完全取决于项目的业务需求。下面以开发一个液晶时钟为例，来介绍桌面控件的用法。

实例：液晶时钟

为了实现一个液晶时钟的桌面组件，开发者需要在程序界面上定义 8 个 ImageView，其中 6 个 ImageView 用于显示小时、分钟、秒钟的数字，另外两个 ImageView 用于显示小时、分钟、秒钟之间的冒号。

为了让桌面组件实时地显示当前时间，程序需要每隔 1 秒更新一次程序界面上的 6 个 ImageView，让它们显示当前小时、分钟、秒钟的数字即可。

液晶时钟的代码如下。

程序清单：codes\14\14.4\LedClock\app\src\main\java\org\crazyit\desktop\LedClock.java

```
public class LedClock extends AppWidgetProvider
{
    private Timer timer = new Timer();
    private AppWidgetManager appWidgetManager;
    private Context context;
    // 将 0~9 的液晶数字图片定义成数组
    private int[] digits = new int[]
    {
        R.drawable.su01,
        R.drawable.su02,
        R.drawable.su03,
        R.drawable.su04,
        R.drawable.su05,
        R.drawable.su06,
        R.drawable.su07,
        R.drawable.su08,
        R.drawable.su09,
        R.drawable.su10,
    };
    // 将显示小时、分钟、秒钟的 ImageView 定义成数组
    private int[] digitViews = new int[]
       {
           R.id.img01,
           R.id.img02,
           R.id.img04,
           R.id.img05,
           R.id.img07,
           R.id.img08
       };
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
           int[] appWidgetIds)
    {
        System.out.println("--onUpdate--");
        this.appWidgetManager = appWidgetManager;
        this.context = context;
        // 定义计时器
        timer = new Timer();
        // 启动周期性调度
        timer.schedule(new TimerTask()
        {
            public void run()
            {
                // 发送空消息，通知界面更新
```

```
                handler.sendEmptyMessage(0x123);
            }
        }, 0, 1000);
    }
    private Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                RemoteViews views = new RemoteViews(context
                        .getPackageName(), R.layout.main);
                // 定义 SimpleDateFormat 对象
                SimpleDateFormat df = new SimpleDateFormat(
                        "HHmmss");
                // 将当前时间格式化成 HHmmss 的形式
                String timeStr = df.format(new Date());
                for(int i = 0 ; i < timeStr.length() ;i++)
                {
                    // 将第 i 个数字字符转换为对应的数字
                    int num = timeStr.charAt(i) - 48;
                    // 将第 i 个图片设为对应的液晶数字图片
                    views.setImageViewResource(digitViews[i], digits[num]);
                }
                // 将 AppWidgetProvider 子类实例包装成 ComponentName 对象
                ComponentName componentName = new ComponentName(context,
                    LedClock.class);
                // 调用 AppWidgetManager 将 remoteViews 添加到 ComponentName 中
                appWidgetManager.updateAppWidget(componentName, views);
            }
            super.handleMessage(msg);
        }
    };
}
```

上面程序中的粗体字代码将会根据程序的时间字符串动态更新 6 个 ImageView 所显示的液晶数字图片，这样即可通过液晶数字来显示当前时间了。

在 AndroidManifest.xml 文件中添加如下代码片段来定义该桌面控件：

```
<receiver android:name=".LedClock"
    android:label="@string/app_name">
    <!-- 将该 BroadcastReceiver 当成桌面控件 -->
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <!-- 指定桌面控件的 meta-data -->
    <meta-data android:name="android.appwidget.provider"
        android:resource="@xml/my_clock"/>
</receiver>
```

上面配置文件中的粗体字代码指定了该液晶时钟的 meta-data 为@xml/my_clock，还需要在 res/xml 路径下增加一个 my_clock.xml 文件，该文件的内容与前一个桌面控件的 meta-data 文件大致相同，此处不再给出。

通过 Android 系统把该液晶时钟小控件添加到桌面上，将可以看到 Android 桌面显示如图 14.13 所示的液晶时钟。

图 14.13 液晶时钟控件

正如图 14.13 中的液晶时钟所示，这种桌面控件显示了程序的运行状态，它的运行界面可以是动态改变

的。因此还可以通过桌面控件把股票走势图之类添加到桌面上，让用户可以一眼看到。总之，这种桌面控件给了开发者更多的想象空间。

14.4.2 显示带数据集的桌面控件

Android 为 RemoteViews 提供了如下方法。

- setRemoteAdapter(int viewId, Intent intent)：该方法可以使用 Intent 更新 RemoteViews 中 viewId 对应的组件。

上面方法的 Intent 参数应该封装一个 RemoteViewsService 参数，该参数虽然继承了 Service 组件，但它的主要作用是为 RemoteViews 中 viewId 对应的组件提供列表项。

由于 Intent 参数负责提供列表项，因此 viewId 参数对应的组件可以是 ListView、GridView、StackView 和 AdapterViewFlipper 等，这些组件都是 AdapterView 的子类，由此可见，RemoteViewsService 负责提供的对象，应该是一个类似于 Adapter 的对象。

RemoteViewsService 通常用于被继承，继承该基类时需要重写它的 onGetViewFactory()方法，该方法需要返回一个类似于 Adapter 的对象——但不是 Adapter，而是 RemoteViewsFactory 对象，RemoteViewsFactory 的功能完全类似于 Adapter。

下面是本示例提供的 RemoteViewsService 类。

程序清单：codes\14\14.4\AppWidgetCollection\app\src\main\java\org\crazyit\desktop\StackWidgetService.java

```
public class StackWidgetService extends RemoteViewsService
{
    // 重写该方法，该方法返回一个 RemoteViewsFactory 对象
    // RemoteViewsFactory 对象的作用类似于 Adapter
    // 它负责为 RemoteView 中的指定组件提供多个列表项
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent)
    {
        return new StackRemoteViewsFactory(this.getApplicationContext(),
            intent); // ①
    }
    class StackRemoteViewsFactory implements
    RemoteViewsService.RemoteViewsFactory
    {
        // 定义一个数组来保存该组件生成的多个列表项
        private int[] items = null;
        private Context mContext;
        public StackRemoteViewsFactory(Context context, Intent intent)
        {
            mContext = context;
        }
        @Override
        public void onCreate()
        {
            // 初始化 items 数组
            items = new int[] { R.drawable.bomb5, R.drawable.bomb6,
                R.drawable.bomb7, R.drawable.bomb8, R.drawable.bomb9,
                R.drawable.bomb10, R.drawable.bomb11, R.drawable.bomb12,
                R.drawable.bomb13, R.drawable.bomb14, R.drawable.bomb15,
                R.drawable.bomb16
            };
        }
        @Override
        public void onDestroy()
        {
            items = null;
        }
```

```
        // 该方法的返回值控制该对象包含多少个列表项
        @Override
        public int getCount()
        {
            return items.length;
        }
        // 该方法的返回值控制各位置所显示的 RemoteViews
        @Override
        public RemoteViews getViewAt(int position)
        {
            // 创建 RemoteViews 对象，加载/res/layout 目录下的 widget_item.xml 文件
            RemoteViews rv = new RemoteViews(mContext.getPackageName(),
                R.layout.widget_item);
            // 更新 widget_item.xml 布局文件中的 widget_item 组件
            rv.setImageViewResource(R.id.widget_item,
                items[position]);
            // 创建 Intent，用于传递数据
            Intent fillInIntent = new Intent();
            fillInIntent.putExtra(StackWidgetProvider.EXTRA_ITEM, position);
            // 设置当单击该 RemoteViews 时传递 fillInIntent 包含的数据
            rv.setOnClickFillInIntent(R.id.widget_item, fillInIntent);
            // 此处让线程暂停 0.5 秒来模拟加载该组件
            try
            {
                System.out.println("加载【" + position + "】位置的组件");
                Thread.sleep(500);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            return rv;
        }
        @Override
        public RemoteViews getLoadingView()
        {
            return null;
        }
        @Override
        public int getViewTypeCount()
        {
            return 1;
        }
        @Override
        public long getItemId(int position)
        {
            return position;
        }
        @Override
        public boolean hasStableIds()
        {
            return true;
        }
        @Override
        public void onDataSetChanged()
        {
        }
    }
}
```

上面程序中的①号粗体字代码返回了一个 StackRemoteViewsFactory 对象，该对象的作用类似于 Adapter 对象，就是负责返回多个列表项。与普通 Adapter 不同的是，Adapter 返回的多个列表项只要是 View 组件即可；但 StackRemoteViewsFactory 对象返回的列表项必须是

RemoteViews 组件，因此上面程序中的粗体字代码重写 getViewAt(int position)方法时需要创建并返回一个 RemoteViews 对象。

上面程序加载的布局文件是位于/res/layout 目录下的 widget_item.xml，该文件只包含一个简单的 ImageView 组件。该布局文件代码如下。

程序清单：codes\14\14.4\AppWidgetCollection\app\src\main\res\layout\widget_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget_item"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:gravity="center"/>
```

提供了 StackWidgetService 之后，接下来开发 AppWidgetProvider 子类与开发普通 AppWidgetProvider 子类的步骤基本相同，只是此时不再调用 RemoteViews 的 setImageViewResource()或 setTextViewText()，而是调用 setRemoteAdapter(int viewId, Intent intent)方法。

下面是本示例中 AppWidgetProvider 子类的代码。

程序清单：codes\14\14.4\AppWidgetCollection\app\src\main\java\org\crazyit\desktop\StackWidgetProvider. java

```
public class StackWidgetProvider extends AppWidgetProvider
{
    public static final String TOAST_ACTION
        = "org.crazyit.desktop.TOAST_ACTION";
    public static final String EXTRA_ITEM
        = "org.crazyit.desktop.EXTRA_ITEM";
    @Override
    public void onUpdate(Context context,
        AppWidgetManager appWidgetManager, int[] appWidgetIds)
    {
        // 创建 RemoteViews 对象，加载/res/layout 目录下的 widget_layout.xml 文件
        RemoteViews rv = new RemoteViews(context.getPackageName(),
            R.layout.widget_layout);
        Intent intent = new Intent(context, StackWidgetService.class);
        // 使用 intent 更新 rv 中的 stack_view 组件（StackView）
        rv.setRemoteAdapter(R.id.stack_view, intent); // ①
        // 设置当 StackWidgetService 提供的列表项为空时，直接显示 empty_view 组件
        rv.setEmptyView(R.id.stack_view, R.id.empty_view);
        // 创建启动 StackWidgetProvider 组件（作为 BroadcastReceiver）的 Intent
        Intent toastIntent = new Intent(context,
            StackWidgetProvider.class);
        // 为该 Intent 设置 Action 属性
        toastIntent.setAction(StackWidgetProvider.TOAST_ACTION);
        // 将 Intent 包装成 PendingIntent
        PendingIntent toastPendingIntent = PendingIntent
            .getBroadcast(context, 0, toastIntent,
                PendingIntent.FLAG_UPDATE_CURRENT);
        // 将 PendingIntent 与 stack_view 进行关联
        rv.setPendingIntentTemplate(R.id.stack_view,
            toastPendingIntent);
        // 使用 AppWidgetManager 通过 RemoteViews 更新 AppWidgetProvider
        appWidgetManager.updateAppWidget(
            new ComponentName(context, StackWidgetProvider.class), rv); // ②
        super.onUpdate(context, appWidgetManager, appWidgetIds);
    }
    @Override
    public void onDeleted(Context context, int[] appWidgetIds)
    {
        super.onDeleted(context, appWidgetIds);
    }
```

```
    @Override
    public void onDisabled(Context context)
    {
        super.onDisabled(context);
    }
    @Override
    public void onEnabled(Context context)
    {
        super.onEnabled(context);
    }
    // 重写该方法，将该组件当成 BroadcastReceiver 使用
    @Override
    public void onReceive(Context context, Intent intent)
    {
        if (intent.getAction().equals(TOAST_ACTION))
        {
            // 获取 Intent 中的数据
            int viewIndex = intent.getIntExtra(EXTRA_ITEM, 0);
            // 显示 Toast 提示
            Toast.makeText(context, "点击第【" + viewIndex + "】个列表项",
                Toast.LENGTH_SHORT).show();
        }
        super.onReceive(context, intent);
    }
}
```

上面的①号粗体字代码调用 RemoteViews 的 setRemoteAdapter()方法进行设置，该方法负责为 RemoteViews 中的 StackView 设置 RemoteViewsFactory 对象——该 RemoteViewsFactory 对象负责提供多个列表项。接下来②号粗体字代码调用 AppWidgetManager 的 updateAppWidget()进行更新，这行代码与开发普通的 AppWidgetProvider 基本相同。

由于这种带数据集的桌面控件同时需要 AppWidgetProvider 和 RemoteViewsService，因此在 AndroidManifest.xml 文件中需要同时配置<receiver.../>元素和<service.../>元素。本示例的配置片段如下。

程序清单：codes\14\14.4\AppWidgetCollection\app\src\main\AndroidManifest.xml

```
<application
    android:allowBackup="true"
    android:label="@string/app_name">
    <!-- 配置 AppWidgetProvider，即配置桌面控件 -->
    <receiver android:name=".StackWidgetProvider">
        <!-- 通过该 intent-filter 指定该 Receiver 作为桌面控件 -->
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>
        <!-- 为桌面控件指定 meta-data -->
        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/stackwidgetinfo" />
    </receiver>
    <!-- 配置 RemoteViewsService
    必须指定权限为 android.permission.BIND_REMOTEVIEWS
     -->
    <service
        android:name=".StackWidgetService"
        android:permission="android.permission.BIND_REMOTEVIEWS"
        android:exported="false" />
</application>
```

该示例同样需要在/res/xml 目录下添加一个 stackwidgetinfo.xml 元数据配置文件，该文件

与前面开发普通 AppWidgetProvider 基本相似，此处不再给出。

将该应用部署到模拟器上，并将该桌面控件添加到桌面上，将可以看到如图 14.14 所示的桌面控件。

图 14.14 带数据集的桌面控件

14.5 本章小结

Android 系统的桌面是用户每天接触最多的界面，如果用户把我们开发的应用添加到系统桌面上，则可以大大增加用户对该软件的依赖。本章主要介绍了如何管理 Android 手机桌面，包括开发动态壁纸、管理手机桌面上的快捷方式、管理桌面小控件、显示带数据集的桌面小控件等，这些内容是读者需要重点掌握的。

CHAPTER

15

第15章 传感器应用开发

本章要点

- 手机传感器的概念和作用
- 定义监听器监听传感器的数据
- 加速度传感器
- 方向传感器
- 陀螺仪传感器
- 磁场传感器
- 重力传感器
- 线性加速度传感器
- 温度传感器
- 光传感器
- 压力传感器
- 心率传感器
- 使用方向传感器开发指南针
- 使用方向传感器开发水平仪

Android 系统提供了对传感器的支持，如果手机设备的硬件提供了这些传感器，Android 应用可以通过传感器来获取设备的外界条件，包括手机设备的运行状态、当前摆放方向、外界的磁场、温度和压力等。Android 系统提供了驱动程序去管理这些传感器硬件，当传感器硬件感知到外部环境发生改变时，Android 系统负责管理这些传感器数据。

对于 Android 应用开发者来说，开发传感器应用十分简单，开发者只要为指定传感器注册一个监听器即可，当外部环境发生改变时，Android 系统会通过传感器获取外部环境的数据，并将数据传给监听器的监听方法。

通过在 Android 应用中添加传感器，可以充分激发开发者、用户的想象力，可以开发出各种新奇的程序，比如电子罗盘、水平仪等。除此之外，还可以利用传感器开发各种游戏，必须通过传感器来感知用户动作，从而在游戏中提供对应的响应。

15.1 利用 Android 的传感器

在 Android 系统中开发传感器应用十分简单，因为 Android 系统为传感器支持提供了强大的管理服务。

开发传感器应用的步骤如下。

① 调用 Context 的 getSystemService(Context.SENSOR_SERVICE) 方法获取 SensorManager 对象，SensorManager 对象代表系统的传感器管理服务。

② 调用 SensorManager 的 getDefaultSensor(int type)方法来获取指定类型的传感器。

③ 通常选择在 Activity 的 onResume()方法中调用 SensorManager 的 registerListener()方法为指定传感器注册监听器，程序通过实现监听器即可获取传感器传回来的数据。

SensorManager 提供的注册传感器的方法为 registerListener(SensorEventListener listener, Sensor sensor, int rate)，该方法中三个参数说明如下。

- listener：监听传感器事件的监听器。该监听器需要实现 SensorEventListener 接口。
- sensor：传感器对象。
- rate：指定获取传感器数据的频率。

该方法中 rate 可以获取传感器数据的频率，它支持如下几个频率值。

- SensorManager.SENSOR_DELAY_FASTEST：最快。延迟最小，只有特别依赖于传感器数据的应用推荐采用这种频率，该种模式可能造成手机电量大量消耗。由于传递的为原始数据，算法处理不好将会影响应用的性能。
- SensorManager.SENSOR_DELAY_GAME：适合游戏的频率。一般有实时性要求的应用适合使用这种频率。
- SensorManager.SENSOR_DELAY_NORMAL：正常频率。一般对实时性要求不是特别高的应用适合使用这种频率。
- SensorManager.SENSOR_DELAY_UI：适合普通用户界面的频率。这种模式比较省电，而且系统开销也很小，但延迟较大，因此只适合在普通小程序中使用。

下面将会按照上面的步骤来开发一个加速度传感器应用。该程序的界面很简单，提供一个文本框来显示加速度值即可，此处不再给出界面布局代码。该应用的 Activity 代码如下。

程序清单：codes\15\15.1\AccelerometerTest\app\src\main\java\org\crazyit\sensor\MainActivity.java

```
public class MainActivity extends Activity
    implements SensorEventListener
{
```

```
    // 定义系统的 Sensor 管理器
    SensorManager sensorManager;
    EditText etTxt1;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面上的文本框组件
        etTxt1 = (EditText) findViewById(R.id.txt1);
        // 获取系统的传感器管理服务
        sensorManager = (SensorManager) getSystemService(
            Context.SENSOR_SERVICE); // ①
    }
    @Override
    protected void onResume()
    {
        super.onResume();
        // 为系统的加速度传感器注册监听器
        sensorManager.registerListener(this,
            sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_GAME); // ②
    }
    @Override
    protected void onStop()
    {
        // 取消注册
        sensorManager.unregisterListener(this);
        super.onStop();
    }
    // 以下是实现 SensorEventListener 接口必须实现的方法
    // 当传感器的值发生改变时回调该方法
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        float[] values = event.values;
        StringBuilder sb = new StringBuilder();
        sb.append("X 方向上的加速度：");
        sb.append(values[0]);
        sb.append("\nY 方向上的加速度：");
        sb.append(values[1]);
        sb.append("\nZ 方向上的加速度：");
        sb.append(values[2]);
        etTxt1.setText(sb.toString());
    }
    // 当传感器精度改变时回调该方法
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
    }
}
```

上面程序中的①号粗体字代码用于获取系统的传感器管理服务，②号粗体字代码则用于获取加速度传感器，并为该传感器注册监听器。本程序直接使用了 Activity 充当传感器监听器，因此该 Activity 实现了 SensorEventListener 接口，并实现了该接口中的两个方法。

- onSensorChanged()：当传感器的值发生改变时触发该方法。
- onAccuracyChanged()：当传感器的精度发生改变时触发该方法。

上面的程序实现 onSensorChanged()方法时通过 SensorEvent 对象的 values()方法来获取传感器的值，不同传感器所返回的值的个数是不等的。对于加速度传感器来说，它将返回三个值，

分别代表手机设备在 X、Y、Z 三个方向上的加速度。

需要指出的是，传感器的坐标系统与屏幕坐标系统不同，传感器坐标系统的 X 轴沿屏幕向右；Y 轴则沿屏幕向上，Z 轴则垂直于屏幕向外。图 15.1 显示了传感器的坐标系统。

从图 15.1 所示的坐标系统大致可以看出，当拿着手机横向左右移动时，可能产生 X 轴上的加速度；拿着手机前后移动时，可能产生 Z 轴上的加速度；当拿着手机竖向上下移动时，可能产生 Y 轴上的加速度。

运行上面的程序，将会看到如图 15.2 所示的输出。

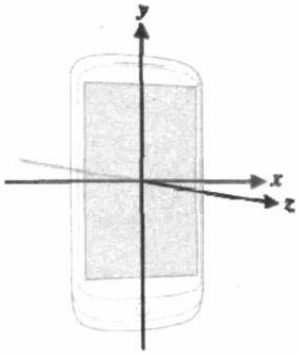


图 15.1 传感器的坐标系统

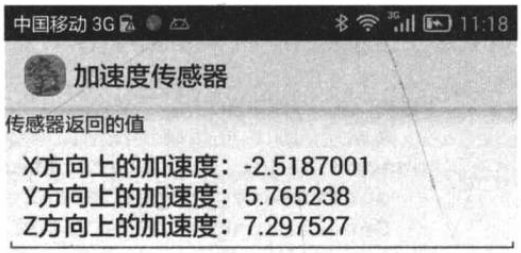


图 15.2 加速度传感器

从图 15.2 可以看出，当用户拿着手机“晃动”时，即可看到程序能检测到传感器返回的加速度值。

Android 模拟器默认并没有提供传感器支持，因此本章的所有示例程序都应使用真机进行测试，才可看到正确的运行效果。

15.2 Android 的常用传感器

上一节介绍了如何监听 Android 设备的加速度，只要通过 SensorManager 为加速度传感器注册监听器即可。实际上 Android 系统对所有类型的传感器的处理都完全一样，只是传感器的类型有所区别而已。

15.2.1 方向传感器

方向传感器用于感应手机设备的摆放状态。方向传感器可以返回三个角度，这三个角度即可确定手机的摆放状态。

关于方向传感器返回的三个角度说明如下。

- **第一个角度：**表示手机顶部朝向与正北方的夹角。当手机绕着 Z 轴旋转时，该角度值发生改变。例如当该角度为 0 度时，表明手机顶部朝向正北；该角度为 90 度时，代表手机顶部朝向正东；该角度为 180 度时，代表手机顶部朝向正南；该角度为 270 度时，代表手机顶部朝向正西。
- **第二个角度：**表示手机顶部或尾部翘起的角度。当手机绕着 X 轴倾斜时，该角度值发生变化，该角度的取值范围是-180~180 度。假设将手机屏幕朝上水平放在桌子上，如果桌子是完全水平的，该角度值应该是 0 度。假如从手机顶部开始抬起，直到将手机沿 X 轴旋转 180 度（屏幕向下水平放在桌面上），在这个旋转过程中，该角度值会从 0 度变化到-180 度。也就是说，从手机顶部抬起时，该角度的值会逐渐减小，直到等于-180 度；如果从手机底部开始抬起，直到将手机沿 X 轴旋转 180 度（屏幕向下水平

放在桌面上），该角度的值会从 0 度变化到 180 度。也就是说，从手机底部抬起时，该角度的值会逐渐增大，直到等于 180 度。

- **第三个角度：**表示手机左侧或右侧翘起的角度。当手机绕着 *Y* 轴倾斜时，该角度值发生变化。该角度的取值范围是-90～90 度。假设将手机屏幕朝上水平放在桌面上，如果桌面是完全水平的，该角度值应为 0 度。假如将手机左侧逐渐抬起，直到将手机沿 *Y* 轴旋转 90 度（手机与桌面垂直），在这个旋转过程中，该角度值会从 0 度变化到-90 度。也就是说，从手机左侧抬起时，该角度的值会逐渐减小，直到等于-90 度；如果从手机右侧开始抬起，直到将手机沿 *Y* 轴旋转 90 度（手机与桌面垂直），该角度的值会从 0 度变化到 90 度。也就是说，从手机右侧抬起时，该角度的值会逐渐增大，直到等于 90 度。

通过在应用程序中使用方向传感器，应用程序就可检测到手机设备的摆放状态，比如手机顶部的朝向，手机目前的倾斜角度等。借助于方向传感器，可以开发出指南针、水平仪等有趣的应用。

此处不单独介绍方向传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.2 陀螺仪传感器

陀螺仪传感器用于感应手机设备的旋转速度。陀螺仪传感器可以返回设备绕 *X*、*Y*、*Z* 这三个坐标轴（陀螺仪传感器的坐标系统与加速度传感器的坐标系统完全相同）的旋转速度。旋转速度的单位是弧度/秒，旋转速度为正值代表逆时针旋转，负值代表顺时针旋转。

关于陀螺仪传感器返回的三个角速度说明如下。

- **第 1 个值：**代表该设备绕 *X* 轴旋转的角速度。
- **第 2 个值：**代表该设备绕 *Y* 轴旋转的角速度。
- **第 3 个值：**代表该设备绕 *Z* 轴旋转的角速度。

此处不单独介绍陀螺仪传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.3 磁场传感器

磁场传感器主要用于读取手机设备外部的磁场强度。即使周围没有任何直接的磁场，手机设备也始终会处于地球磁场中。随着手机设备摆放状态的改变，周围磁场在手机的 *X*、*Y*、*Z* 方向上的影响会发生改变。

磁场传感器会返回三个数据，分别代表周围磁场分解到 *X*、*Y*、*Z* 三个方向上的磁场分量，磁场数据的单位是微特斯拉（μT）。

此处不单独介绍磁场传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.4 重力传感器

重力传感器会返回一个三维向量，这个三维向量可显示重力的方向和强度。重力传感器的坐标系统与加速度传感器的坐标系统是相同的。

注意：

当设备处于静止状态时，重力传感器的输出与加速度传感器的输出应该是相同的。

15.2.5 线性加速度传感器

线性加速度传感器返回一个三维向量显示设备在各方向上的加速度（不包含重力加速度）。线性加速度传感器的坐标系统与加速度传感器的坐标系统相同。

线性加速度传感器、重力传感器、加速度传感器这三者输出值的关系如下：

```
加速度传感器 = 重力传感器 + 线性加速度传感器
```

15.2.6 温度传感器

温度传感器用于获取手机设备所处环境的温度。温度传感器会返回一个数据，代表手机设备周围的温度，单位是摄氏度。

此处不单独介绍温度传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.7 光传感器

光传感器用于获取手机设备所处环境的光的强度。光传感器会返回一个数据，代表手机设备周围的光的强度，该数据的单位是勒克斯（lux）。

此处不单独介绍光传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.8 压力传感器

压力传感器用于获取手机设备所处环境的压力的大小。压力传感器会返回一个数据，代表手机设备周围的压力的大小。

此处不单独介绍压力传感器的应用，后面将通过一个示例来集中介绍所有传感器的用法。

正如前面所介绍的，Android 系统对所有传感器的处理方式完全相同，接下来通过一个示例程序来介绍上面这些传感器的用法。该程序界面只是提供了几个文本框，分别用于显示不同的传感器数据。该程序代码如下。

程序清单：codes\15\15.2\SensorTest\app\src\main\java\org\crazyit\sensor\MainActivity.java

```
public class MainActivity extends Activity
        implements SensorEventListener
{
    // 定义 Sensor 管理器
    private SensorManager mSensorManager;
    EditText etOrientation;
    EditText etGyro;
    EditText etMagnetic;
    EditText etGravity;
    EditText etLinearAcc;
    EditText etTemerature;
    EditText etLight;
    EditText etPressure;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面上的 EditText 组件
        etOrientation = (EditText) findViewById(R.id.etOrientation);
        etGyro = (EditText) findViewById(R.id.etGyro);
        etMagnetic = (EditText) findViewById(R.id.etMagnetic);
        etGravity = (EditText) findViewById(R.id.etGravity);
        etLinearAcc = (EditText) findViewById(R.id.etLinearAcc);
        etTemerature = (EditText) findViewById(R.id.etTemerature);
        etLight = (EditText) findViewById(R.id.etLight);
```

```
        etPressure = (EditText) findViewById(R.id.etPressure);
        // 获取传感器管理服务
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE); // ①
    }
    @Override
    protected void onResume()
    {
        super.onResume();
        // 为系统的方向传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的陀螺仪传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的磁场传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的重力传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的线性加速度传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的温度传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的光传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的压力传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE),
            SensorManager.SENSOR_DELAY_GAME);
    }
    @Override
    protected void onStop()
    {
        // 程序退出时取消注册传感器监听器
        mSensorManager.unregisterListener(this);
        super.onStop();
    }
    @Override
    protected void onPause()
    {
        // 程序暂停时取消注册传感器监听器
        mSensorManager.unregisterListener(this);
        super.onPause();
    }
    // 以下是实现 SensorEventListener 接口必须实现的方法
    @Override
    // 当传感器精度改变时回调该方法
    public void onAccuracyChanged(Sensor sensor, int accuracy){ }
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        float[] values = event.values;
```

```
        // 获取触发 event 的传感器类型
        int sensorType = event.sensor.getType();
        StringBuilder sb = null;
        // 判断是哪个传感器发生改变
        switch (sensorType)
        {
            // 方向传感器
            case Sensor.TYPE_ORIENTATION:
                sb = new StringBuilder();
                sb.append("绕 Z 轴转过的角度：");
                sb.append(values[0]);
                sb.append("\n 绕 X 轴转过的角度：");
                sb.append(values[1]);
                sb.append("\n 绕 Y 轴转过的角度：");
                sb.append(values[2]);
                etOrientation.setText(sb.toString());
                break;
            // 陀螺仪传感器
            case Sensor.TYPE_GYROSCOPE:
                sb = new StringBuilder();
                sb.append("绕 X 轴旋转的角速度：");
                sb.append(values[0]);
                sb.append("\n 绕 Y 轴旋转的角速度：");
                sb.append(values[1]);
                sb.append("\n 绕 Z 轴旋转的角速度：");
                sb.append(values[2]);
                etGyro.setText(sb.toString());
                break;
            // 磁场传感器
            case Sensor.TYPE_MAGNETIC_FIELD:
                sb = new StringBuilder();
                sb.append("X 轴方向上的磁场强度：");
                sb.append(values[0]);
                sb.append("\nY 轴方向上的磁场强度：");
                sb.append(values[1]);
                sb.append("\nZ 轴方向上的磁场强度：");
                sb.append(values[2]);
                etMagnetic.setText(sb.toString());
                break;
            // 重力传感器
            case Sensor.TYPE_GRAVITY:
                sb = new StringBuilder();
                sb.append("X 轴方向上的重力：");
                sb.append(values[0]);
                sb.append("\nY 轴方向上的重力：");
                sb.append(values[1]);
                sb.append("\nZ 轴方向上的重力：");
                sb.append(values[2]);
                etGravity.setText(sb.toString());
                break;
            // 线性加速度传感器
            case Sensor.TYPE_LINEAR_ACCELERATION:
                sb = new StringBuilder();
                sb.append("X 轴方向上的线性加速度：");
                sb.append(values[0]);
                sb.append("\nY 轴方向上的线性加速度：");
                sb.append(values[1]);
                sb.append("\nZ 轴方向上的线性加速度：");
                sb.append(values[2]);
                etLinearAcc.setText(sb.toString());
                break;
            // 温度传感器
```

```
            case Sensor.TYPE_AMBIENT_TEMPERATURE:
                sb = new StringBuilder();
                sb.append("当前温度为：");
                sb.append(values[0]);
                etTemerature.setText(sb.toString());
                break;
            // 光传感器
            case Sensor.TYPE_LIGHT:
                sb = new StringBuilder();
                sb.append("当前光的强度为：");
                sb.append(values[0]);
                etLight.setText(sb.toString());
                break;
            // 压力传感器
            case Sensor.TYPE_PRESSURE:
                sb = new StringBuilder();
                sb.append("当前压力为：");
                sb.append(values[0]);
                etPressure.setText(sb.toString());
                break;
        }
    }
}
```

上面的程序一样遵守前面介绍的传感器编程步骤：①号粗体字代码在 Activity 的 onCreate()方法中获取 SensorManager 对象。程序中大段的粗体字代码在 Activity 的 onResume()方法中为指定类型的传感器注册监听器，本程序为 8 种类型的传感器注册了监听器；实现 onSensorChanged(SensorEvent event)方法就实现了传感器监听器，实现监听器方法时即可获取传感器所传回来的数据。

在真机中调试该程序，运行该程序，即可看到如图 15.3 所示的结果。

从图 15.3 可以看出，该程序并不能获取温度传感器和压力传感器的值，这是因为笔者的手机不支持温度传感器与压力传感器的缘故。

图 15.3 传感器应用

➤➤ 15.2.9 Android 5.0 新增的心率传感器

心率传感器是 Android 5.0 新增的一种传感器，这种传感器可以返回佩戴该设备的人每分钟的心跳次数。该传感器返回的数据的准确性可通过 SensorEvent 的 accuracy 进行判断，如果该属性值为 SENSOR_STATUS_UNRELIABLE 或 SENSOR_STATUS_NO_CONTACT，则表明传感器返回的心率值是不太可靠的，应该被丢弃。

注意：

这种传感器需要 android.permission.BODY_SENSORS 权限，如果该程序没有获取这种权限，SensorManager.getSensorsList()、SensorManager.getDefaultSensor()都不能返回这种传感器。

15.3 传感器应用案例

对传感器的支持是 Android 系统的特性之一，通过使用传感器可以轻易开发出各种有趣的应用。下面将会通过使用方向传感器来开发指南针和水平仪两个有趣的应用。

实例：指南针

前面介绍方向传感器时已经指出，方向传感器传回来的第一个参数值就是代表手机绕 Z 轴转过的角度，也就是手机顶部与正北的夹角，通过在程序中检测该夹角就可以开发出指南针。

开发指南针的思路很简单：先准备一张指南针图片，该图片上方向指针指向北方。接下来开发一个检测方向的传感器，程序检测到手机顶部绕 Z 轴转过多少度，让指南针图片反向转过多少度即可。由此可见，指南针应用只要在界面中添加一张图片，并让图片总是反向转过方向传感器返回的第一个角度值即可。下面是该程序的代码。

程序清单：codes\15\15.3\Compass\app\src\main\java\org\crazyit\compass\MainActivity.java

```
public class MainActivity extends Activity
    implements SensorEventListener
{
    // 定义显示指南针的图片
    ImageView znzImage;
    // 记录指南针图片转过的角度
    float currentDegree = 0f;
    // 定义 Sensor 管理器
    SensorManager mSensorManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面中显示指南针的图片
        znzImage = (ImageView) findViewById(R.id.znzImage);
        // 获取传感器管理服务
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
    }
    @Override
    protected void onResume()
    {
        super.onResume();
        // 为系统的方向传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
            SensorManager.SENSOR_DELAY_GAME);
    }
    @Override
    protected void onPause()
    {
        // 取消注册
        mSensorManager.unregisterListener(this);
        super.onPause();
    }
    @Override
    protected void onStop()
    {
        // 取消注册
        mSensorManager.unregisterListener(this);
        super.onStop();
    }
```

```
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        // 获取触发 event 的传感器类型
        int sensorType = event.sensor.getType();
        switch (sensorType)
        {
            case Sensor.TYPE_ORIENTATION:
                // 获取绕 Z 轴转过的角度
                float degree = event.values[0];
                // 创建旋转动画（反向转过 degree 度）
                RotateAnimation ra = new RotateAnimation(currentDegree,
                    -degree, Animation.RELATIVE_TO_SELF, 0.5f,
                    Animation.RELATIVE_TO_SELF, 0.5f);
                // 设置动画的持续时间
                ra.setDuration(200);
                // 运行动画
                znzImage.startAnimation(ra);
                currentDegree = -degree;
                break;
        }
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
    }
}
```

指南针程序的关键代码就是程序中的粗体字代码，该程序检测到手机绕 *Z* 轴转过的角度，也就是手机 *Y* 轴与正北方向的夹角，然后让指南针图片反向转过相应的角度即可。

在真机中调试该程序，运行该程序，即可看到如图 15.4 所示的结果。

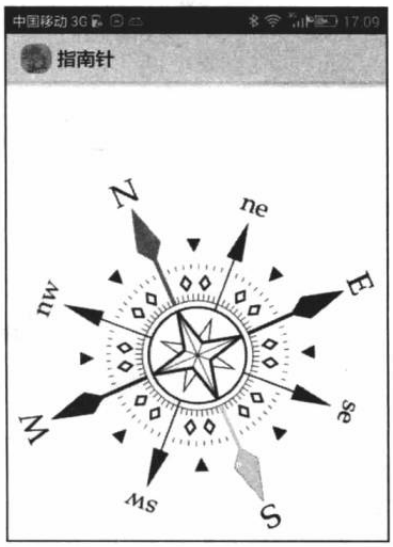


图 15.4 指南针

实例：水平仪

这里介绍的水平仪就是那种比较传统的水平仪，在一个透明的圆盘中充满某种液体，液体中留有一个气泡，当一端翘起时，该气泡将会浮向翘起的一端。

前面介绍方向传感器时已经指出，方向传感器会返回三个角度值，其中第二个角度值代表底部翘起的角度（当顶部翘起时为负值）；第三个角度值代表右侧翘起的角度（当左侧翘起时为负值）；根据这两个角度值就可开发出水平仪了。

假设我们以大透明圆盘的中心为原点，当手机顶部翘起时，气泡应该向顶部移动，也就是气泡位置的 *Y* 坐标（2D 绘图坐标系，屏幕左上角为原点）应减小；当手机底部翘起时，气泡应该向底部移动，也就是气泡位置的 *Y* 坐标应增大——假设气泡开始位于大透明圆盘的中心，气泡的 *Y* 坐标的改变正好与方向传感器返回的第二个参数代表的角度的正负相符，因此根据方向传感器返回的第二个参数来计算气泡的 *Y* 坐标即可；与此类似，当手机左侧翘起时，气泡应该向左侧移动，也就是气泡位置的 *X* 坐标（2D 绘图坐标系，屏幕左上角为原点）应减小；当手机右侧翘起时，气泡应该向右侧移动，也就是气泡位置的 *X* 坐标应增大——假设气泡开始位于大透明圆盘的中心，气泡的 *X* 坐标的改变正好与方向传感器返回的第三个参数代表的

角度的正负相符，因此根据方向传感器返回的第三个参数来计算气泡的 X 坐标即可。

通过上面介绍的方式来动态改变程序界面中气泡的位置——手机哪端翘起，水平仪中的气泡就浮向哪端，这就是水平仪的实现思想。

该程序用了一个自定义 View，该自定义 View 很简单，就是绘制透明圆盘和气泡——其中气泡的位置会动态改变。该自定义 View 的代码如下。

程序清单：codes\15\15.3\Gradienter\app\src\main\java\org\crazyit\sensor\MyView.java

```
public class MyView extends View
{
    // 定义水平仪仪表盘图片
    Bitmap back;
    // 定义水平仪中的气泡图标
    Bitmap bubble;
    // 定义水平仪中气泡的 X、Y 坐标
    int bubbleX, bubbleY;
    public MyView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        // 获取窗口管理器
        WindowManager wm = (WindowManager) context
            .getSystemService(Context.WINDOW_SERVICE);
        // 获取屏幕的宽度和高度
        Display display = wm.getDefaultDisplay();
        DisplayMetrics metrics = new DisplayMetrics();
        display.getMetrics(metrics);
        int screenWidth =  metrics.widthPixels;
        // 创建位图
        back = Bitmap.createBitmap(screenWidth , screenWidth
            , Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(back);
        Paint paint = new Paint();
        paint.setAntiAlias(true);
        // 设置绘制风格：仅填充
        paint.setStyle(Paint.Style.FILL);
        // 创建一个线性渐变来绘制线性渐变
        Shader shader = new LinearGradient(0 , screenWidth
            , screenWidth * 0.8f , screenWidth * 0.2f ,
            Color.YELLOW, Color.WHITE , Shader.TileMode.MIRROR);
        paint.setShader(shader);
        // 绘制圆形
        canvas.drawCircle(screenWidth / 2
            , screenWidth / 2,  screenWidth / 2 , paint);
        Paint paint2 = new Paint();
        paint2.setAntiAlias(true);
        // 设置绘制风格：仅绘制边框
        paint2.setStyle(Paint.Style.STROKE);
        paint2.setStrokeWidth(5);  // 设置画笔宽度
        paint2.setColor(Color.BLACK); // 设置画笔颜色
        // 绘制圆形边框
        canvas.drawCircle(screenWidth / 2
            , screenWidth / 2,  screenWidth / 2, paint2);
        // 绘制水平横线
        canvas.drawLine(0 , screenWidth / 2
            , screenWidth, screenWidth / 2, paint2);
        // 绘制垂直横线
        canvas.drawLine(screenWidth / 2 , 0
            , screenWidth / 2 , screenWidth, paint2);
        paint2.setStrokeWidth(10);  // 设置画笔宽度
        // 设置画笔颜色
```

```
        paint2.setColor(Color.RED);
        // 绘制中心的红色“十”字
        canvas.drawLine(screenWidth / 2 - 30 , screenWidth / 2
            , screenWidth / 2 + 30, screenWidth / 2, paint2);
        canvas.drawLine(screenWidth / 2 , screenWidth / 2 - 30
            , screenWidth / 2, screenWidth / 2 + 30, paint2);
        // 加载气泡图片
        bubble = BitmapFactory
            .decodeResource(getResources(), R.drawable.bubble);
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 绘制水平仪仪表盘图片
        canvas.drawBitmap(back, 0, 0, null);
        // 根据气泡坐标绘制气泡
        canvas.drawBitmap(bubble, bubbleX, bubbleY, null);
    }
}
```

正如上面程序中的粗体字代码所示，该自定义 View 会根据 bubbleX、bubbleY 动态地绘制气泡的位置，而这个 bubbleX、bubbleY 就需要根据方向传感器返回的第三个角度、第二个角度来动态计算。

下面是该程序中 Activity 的代码。

程序清单：codes\15\15.3\Gradienter\app\src\main\java\org\crazyit\sensor\MainActivity.java

```
public class MainActivity extends Activity
    implements SensorEventListener
{
    // 定义水平仪的仪表盘
    MyView show;
    // 定义水平仪能处理的最大倾斜角，超过该角度，气泡将直接位于边界
    int MAX_ANGLE = 30;
    // 定义 Sensor 管理器
    SensorManager mSensorManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取水平仪的主组件
        show = (MyView) findViewById(R.id.show);
        // 获取传感器管理服务
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 为系统的方向传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
            SensorManager.SENSOR_DELAY_GAME);
    }
    @Override
    protected void onPause()
    {
        // 取消注册
        mSensorManager.unregisterListener(this);
        super.onPause();
```

```
    }
    @Override
    protected void onStop()
    {
        // 取消注册
        mSensorManager.unregisterListener(this);
        super.onStop();
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
    }
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        float[] values = event.values;
        // 获取触发 event 的传感器类型
        int sensorType = event.sensor.getType();
        switch (sensorType)
        {
            case Sensor.TYPE_ORIENTATION:
                // 获取与 Y 轴的夹角
                float yAngle = values[1];
                // 获取与 Z 轴的夹角
                float zAngle = values[2];
                // 气泡位于中间时（水平仪完全水平），气泡的 X、Y 坐标
                int x = (show.back.getWidth() - show.bubble.getWidth()) / 2;
                int y = (show.back.getHeight() - show.bubble.getHeight()) / 2;
                // 如果与 Z 轴的倾斜角还在最大角度之内
                if (Math.abs(zAngle) <= MAX_ANGLE)
                {
                    // 根据与 Z 轴的倾斜角度计算 X 坐标的变化值
                    //（倾斜角度越大，X 坐标变化越大）
                    int deltaX = (int) ((show.back.getWidth() - show.bubble
                        .getWidth()) / 2 * zAngle / MAX_ANGLE);
                    x += deltaX;
                }
                // 如果与 Z 轴的倾斜角已经大于 MAX_ANGLE，气泡应到最左边
                else if (zAngle > MAX_ANGLE)
                {
                    x = 0;
                }
                // 如果与 Z 轴的倾斜角已经小于负的 MAX_ANGLE，气泡应到最右边
                else
                {
                    x = show.back.getWidth() - show.bubble.getWidth();
                }
                // 如果与 Y 轴的倾斜角还在最大角度之内
                if (Math.abs(yAngle) <= MAX_ANGLE)
                {
                    // 根据与 Y 轴的倾斜角度计算 Y 坐标的变化值
                    //（倾斜角度越大，Y 坐标变化越大）
                    int deltaY = (int) ((show.back.getHeight() - show.bubble
                        .getHeight()) / 2 * yAngle / MAX_ANGLE);
                    y += deltaY;
                }
                // 如果与 Y 轴的倾斜角已经大于 MAX_ANGLE，气泡应到最下边
                else if (yAngle > MAX_ANGLE)
                {
                    y = show.back.getHeight() - show.bubble.getHeight();
                }
                // 如果与 Y 轴的倾斜角已经小于负的 MAX_ANGLE，气泡应到最右边
                else
```

```
                {
                    y = 0;
                }
                // 如果计算出来的 X、Y 坐标还位于水平仪的仪表盘内，则更新水平仪的气泡坐标
                if (isContain(x, y))
                {
                    show.bubbleX = x;
                    show.bubbleY = y;
                }
                // 通知系统重绘 MyView 组件
                show.postInvalidate();
                break;
            }
        }
        // 计算 x、y 点的气泡是否处于水平仪的仪表盘内
        private boolean isContain(int x, int y)
        {
            // 计算气泡的圆心坐标 X、Y
            int bubbleCx = x + show.bubble.getWidth() / 2;
            int bubbleCy = y + show.bubble.getWidth() / 2;
            // 计算水平仪仪表盘的圆心坐标 X、Y
            int backCx = show.back.getWidth() / 2;
            int backCy = show.back.getWidth() / 2;
            // 计算气泡的圆心与水平仪仪表盘的圆心之间的距离
            double distance = Math.sqrt((bubbleCx - backCx) * (bubbleCx - backCx)
                + (bubbleCy - backCy) * (bubbleCy - backCy));
            // 若两个圆心的距离小于它们的半径差，即可认为处于该点的气泡依然位于仪表盘内
            if (distance < (show.back.getWidth() - show.bubble.getWidth()) / 2)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

该程序的关键代码就是程序中粗体字代码部分，这些代码实现了前面介绍的设计：程序检测方向传感器返回的第二个、第三个角度值，并根据第二个角度值来计算气泡的 *Y* 坐标，根据第三个角度值来计算 *X* 坐标；计算完成后通知系统重绘 MyView 组件即可。

如果需要在模拟器中调试该程序，请先启动 SensorSimulator 的 PC 端，并让 SensorSimulator 的手机端程序与 PC 端程序建立连接。然后运行该程序，即可看到如图 15.5 所示的结果。

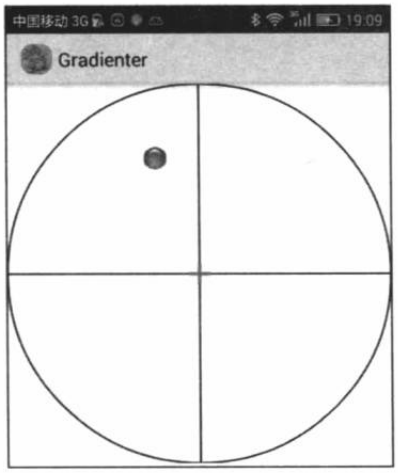


图 15.5 水平仪

从图 15.5 可以看到水平仪中间的气泡，通过该气泡所在的位置即可大致确定手机底下的支撑是否水平。当气泡位于仪表盘中的红色“+”标识处时，即可认为手机底下的支撑完全水平。

15.4 本章小结

Android 系统的特色之一就是支持传感器，通过传感器可以获取手机设备运行的外界信息，包括手机运动的加速度、手机摆放方向等。学习本章需要重点掌握 Android 传感器支持的 API，包括如何通过 SensorManager 注册传感器监听器，如何使用 SensorEventListener 监听传感器数据等。除此之外，读者还需要掌握 Android 的加速度传感器、方向传感器、陀螺仪传感器、磁场传感器、重力传感器、线性加速度传感器、温度传感器、光传感器、压力传感器和心率传感器等常见传感器的功能与用法。

CHAPTER

16

第16章 GPS应用开发

本章要点

- GPS的概念和用途
- Android提供的GPS支持
- LocationManager和LocationProvider
- 获取系统的LocationProvider列表
- 根据名字获取指定的LocationProvider
- 根据Criteria获取满足条件的LocationProvider
- 通过模拟器发送GPS定位信息
- 通过LocationListener监听GPS定位信息
- 临近警告支持

GPS 是英文 Global Positioning System（全球定位系统）的简称，GPS 是 20 世纪 70 年代由美国陆海空三军联合研制的新一代空间卫星导航定位系统。从这个介绍不难发现，GPS 的作用就是为全球的物体提供定位功能。

GPS 定位系统由三部分组成，即 GPS 卫星组成的空间部分、若干地球站组成的控制部分和普通用户手中的接收机这三个部分。对于手机用户来说，手机就是 GPS 定位系统的接收机，也就是说，GPS 定位需要手机的硬件支持 GPS 功能。

GPS 的空间部分由 GPS 卫星组成，覆盖于全球上空的 GPS 卫星星座，必须保证在各处能时时观测到高度角为 15° 以上的 4 颗卫星，这样才能保证 GPS 系统的准确定位。目前，GPS 卫星星座共有 24 颗 GPS 卫星，均匀分布在倾角为 55° 的 6 个轨道上，保证在地面的任意一点都可同时观测到 24 颗卫星。

GPS 定位系统听上去专业、高深，实际上也是一门高新技术，但对于 Android 应用开发的程序员来说，开发提供 GPS 功能的应用程序十分简单。就像 Android 为电话管理支持提供了 TelephonyManager 类、为音频管理支持提供了 AudioManager 类，Android 为支持 GPS 则提供了 LocationManager 类，通过 LocationManager 类及其他几个辅助类，开发人员可以非常方便地开发出 GPS 应用。

16.1 支持 GPS 的核心 API

Android 为 GPS 功能支持专门提供了一个 LocationManager 类，它的作用与 TelephonyManager、AudioManager 等服务类的作用相似，所有 GPS 定位相关的服务、对象都将由该对象来产生。

与程序中获取 TelephonyManager、AudioManager 的方法相似，程序并不能直接创建 LocationManager 的实例，而是通过调用 Context 的 getSystemService()方法来获取，例如如下代码：

```
LocationManager lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

一旦在程序中获得了 LocationManager 对象之后，接下来即可调用 LocationManager 的方法来获取 GPS 定位的相关服务和对象了。LocationManager 提供了如下常用的方法。

- boolean addGpsStatusListener(GpsStatus.Listener listener)：添加一个监听 GPS 状态的监听器。
- void addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)：添加一个临近警告。
- List<String> getAllProviders()：获取所有的 LocationProvider 列表。
- String getBestProvider(Criteria criteria, boolean enabledOnly)：根据指定条件返回最优的 LocationProvider 对象。
- GpsStatus getGpsStatus(GpsStatus status)：获取 GPS 状态。
- Location getLastKnownLocation(String provider)：根据 LocationProvider 获取最近一次已知的 Location。
- LocationProvider getProvider(String name)：根据名称来获取 LocationProvider。
- List<String> getProviders(Criteria criteria, boolean enabledOnly)：根据指定条件获取满足该条件的全部 LocationProvider 的名称。
- List<String> getProviders(boolean enabledOnly)：获取所有可用的 LocationProvider。

- boolean isProviderEnabled(String provider)：判断指定名称的 LocationProvider 是否可用。
- void removeGpsStatusListener(GpsStatus.Listener listener)：删除 GPS 状态监听器。
- void removeProximityAlert(PendingIntent intent)：删除一个临近警告。
- void requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)：通过指定的 LocationProvider 周期性地获取定位信息，并通过 intent 启动相应的组件。
- void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)：通过指定的 LocationProvider 周期性地获取定位信息，并触发 listener 所对应的触发器。

在上面的方法列表中涉及 GPS 定位支持的另一个重要的 API：LocationProvider（定位提供者），LocationProvider 对象就是定位组件的抽象表示，通过 LocationProvider 可以获取该定位组件的相关信息。LocationProvider 提供了如下常用方法。

- int getAccuracy()：返回该 LocationProvider 的精度。
- String getName()：返回该 LocationProvider 的名称。
- int getPowerRequirement()：获取该 LocationProvider 的电源需求。
- boolean hasMonetaryCost()：返回该 LocationProvider 是收费的还是免费的。
- boolean meetsCriteria(Criteria criteria)：判断该 LocationProvider 是否满足 Criteria 条件。
- boolean requiresCell()：判断该 LocationProvider 是否需要访问网络基站。
- boolean requiresNetwork()：判断该 LocationProvider 是否需要网络数据。
- boolean requiresSatellite()：判断该 LocationProvider 是否需要访问基于卫星的定位系统。
- boolean supportsAltitude()：判断该 LocationProvider 是否支持高度信息。
- boolean supportsBearing()：判断该 LocationProvider 是否支持方向信息。
- boolean supportsSpeed()：判断该 LocationProvider 是否支持速度信息。

除此之外，GPS 定位支持还有一个 API：Location，它就是一个代表位置信息的抽象类，提供了如下方法来获取定位信息。

- float getAccuracy()：获取定位信息的精度。
- double getAltitude()：获取定位信息的高度。
- float getBearing()：获取定位信息的方向。
- double getLatitude()：获取定位信息的纬度。
- double getLongitude()：获取定位信息的经度。
- String getProvider()：获取提供该定位信息的 LocationProvider。
- float getSpeed()：获取定位信息的速度。
- boolean hasAccuracy()：判断该定位信息是否有精度信息。
- boolean hasAltitude()：判断该定位信息是否有高度信息。
- boolean hasBearing()：判断该定位信息是否有方向信息。
- boolean hasSpeed()：判断该定位信息是否有速度信息。

上面三个 API 就是 Android GPS 定位支持的三个核心 API，使用它们来获取 GPS 定位信息的通用步骤如下。

1. 获取系统的 LocationManager 对象。
2. 使用 LocationManager，通过指定 LocationProvider 来获取定位信息，定位信息由 Location 对象来表示。

③ 从 Location 对象中获取定位信息。

下面将会使用这三个核心 API 进行系统定位。

16.2 获取 LocationProvider

通过前面的介绍可以看出，Android 的定位信息由 LocationProvider 对象来提供，该对象代表一个抽象的定位组件。在开始编程之前，需要先获得 LocationProvider 对象。

16.2.1 获取所有可用的 LocationProvider

LocationManager 提供了一个 getAllProviders() 方法来获取系统所有可用的 LocationProvider，下面的示例程序将可以列出系统所有的 LocationProvider。该程序界面很简单，界面中只提供一个 ListView 来显示所有的 LocationProvider 即可，故不再给出界面布局代码。该程序的代码如下。

程序清单：codes\16\16.2\AllProvidersTest\src\org\crazyit\gps\MainActivity.java

```
public class MainActivity extends Activity
{
    ListView providers;
    LocationManager lm;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        providers = (ListView) findViewById(R.id.providers);
        // 获取系统的 LocationManager 对象
        lm = (LocationManager)getSystemService(
            Context.LOCATION_SERVICE);
        // 获取系统所有的 LocationProvider 的名称
        List<String> providerNames = lm.getAllProviders();
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
            this,
            android.R.layout.simple_list_item_1,
            providerNames);
        // 使用 ListView 来显示所有可用的 LocationProvider
        providers.setAdapter(adapter);
    }
}
```

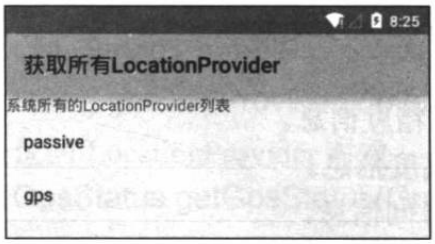


图 16.1 获取系统所有的 LocationProvider

上面程序中的粗体字代码就可获取系统中所有 LocationProvider 的名称。运行上面的程序，可以看到如图 16.1 所示的输出。

从图 16.1 所示的运行结果可以看出，当前模拟器所有可用的 LocationProvider 有如下两个。

- passive：由 LocationManager.PASSIVE_PROVIDER 常量表示。
- gps：由 LocationManager.GPS_PROVIDER 常量表示，代表通过 GPS 获取定位信息的 LocationProvider 对象。

上面列出的 LocationProvider 中最常用的是 LocationProviderGPS_PROVIDER。

提示：

除了上面列出的 passive 和 gps 两个 LocationProvider 之外，还有一个名为 network 的 LocationProvider，由 LocationManager.NETWORK_PROVIDER 常量表示，代表通过移动通信网络获取定位信息的 LocationProvider 对象。

16.2.2 通过名称获得指定 LocationProvider

程序调用 LocationManager 的 getAllProviders()方法获取所有 LocationProvider 时返回的是 List<String>集合，集合元素为 LocationProvider 的名称，为了获取实际的 LocationProvider 对象，可借助于 LocationManager 的 LocationProvider getProvider(String name)方法。

例如以下代码：

```
// 获取基于 GPS 的 LocationProvider
LocationProvider locProvider = lm.getProvider(LocationManager.GPS_PROVIDER);
```

16.2.3 根据 Criteria 获得 LocationProvider

前面的程序调用 LocationManager 的 getAllProviders()方法返回了系统所有可用的 LocationProvider，但大部分时候，应用程序可能希望得到符合指定条件的 LocationProvider，这就需要借助于 LocationManager 的 getBestProvider(Criteria criteria, boolean enabledOnly)方法来获取了。

上面方法中的 Criteria 就代表了一个“过滤”条件，该方法将只返回符合该 Criteria 的 LocationProvider。Criteria 提供了如下常用的方法来设置条件。

- setAccuracy(int accuracy)：设置对 LocationProvider 的精度要求。
- setAltitudeRequired(boolean altitudeRequired)：设置要求 LocationProvider 能提供高度信息。
- setBearingRequired(boolean bearingRequired)：设置要求 LocationProvider 能提供方向信息。
- setCostAllowed(boolean costAllowed)：设置要求 LocationProvider 是否免费。
- setPowerRequirement(int level)：设置对 LocationProvider 耗电量的要求。
- setSpeedRequired(boolean speedRequired)：设置要求 LocationProvider 能提供速度信息。

下面的程序示范了如何获取系统中免费的 LocationProvider，并且该 LocationProvider 必须能提供高度、速度信息等。

程序清单：codes\16\16.2\FreeProvidersTest\src\org\crazyit\gps\MainActivity.java

```
public class MainActivity extends Activity
{
    ListView providers;
    LocationManager lm;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        providers = (ListView) findViewById(R.id.providers);
        // 获取系统的 LocationManager 对象
        lm = (LocationManager)getSystemService(
            Context.LOCATION_SERVICE);
        // 创建一个 LocationProvider 的过滤条件
        Criteria cri = new Criteria();
        // 设置要求 LocationProvider 必须是免费的
```

```
        cri.setCostAllowed(false);
        // 设置要求 LocationProvider 能提供高度信息
        cri.setAltitudeRequired(true);
        // 设置要求 LocationProvider 能提供方向信息
        cri.setBearingRequired(true);
        // 获取系统中所有符合条件的 LocationProvider 的名称
        List<String> providerNames = lm.getProviders(cri , false);
        System.out.println(providerNames.size());
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
            this,
            android.R.layout.simple_list_item_1,
            providerNames);
        // 使用 ListView 来显示所有可用的 LocationProvider
        providers.setAdapter(adapter);
    }
}
```

上面程序中的粗体字代码创建了一个 Criteria 对象，并通过 Criteria 设置了 LocationProvider 必须满足的条件。运行该程序，即可列出所有符合 Criteria 条件的 LocationProvider。

16.3 获取定位信息

获取了 LocationManager 对象之后，接下来就可通过指定 LocationProvider 来获取定位信息了。

16.3.1 通过模拟器发送 GPS 信息

Android 模拟器本身并不能作为 GPS 接收机，因此无法得到 GPS 定位信息，但为了方便程序员测试 GPS 应用，Genymotion 模拟器可以发送模拟的 GPS 定位信息。

启动 Genymotion 模拟器之后，打开 Genymotion 模拟器右边栏的 GPS 图标即可向模拟器发送 GPS 定位信息，如图 16.2 所示。

在图 16.2 所示的“GPS”面板中输入经度值、纬度值，即可向模拟器发送 GPS 定位信息。

图 16.2　向模拟器发送 GPS 定位信息

16.3.2 获取定位数据

下面程序示范了如何通过手机实时地获取定位信息，包括用户所在的经度、纬度、高度、方向、移动速度等。该程序的界面很简单，只提供一个文本框来显示用户的定位信息即可，故此处不再给出程序界面代码。该程序代码如下。

程序清单：codes\16\16.3\LocationTest\src\org\crazyit\gps\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义 LocationManager 对象
    LocationManager locManager;
    // 定义程序界面中的 EditText 组件
    EditText show;
    @Override
```

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 获取程序界面上的 EditText 组件
    show = (EditText) findViewById(R.id.show);
    // 创建 LocationManager 对象
    locManager = (LocationManager) getSystemService
        (Context.LOCATION_SERVICE);
    // 从 GPS 获取最近的定位信息
    Location location = locManager.getLastKnownLocation(
        LocationManager.GPS_PROVIDER);
    // 使用 location 来更新 EditText 的显示
    updateView(location);
    // 设置每 3 秒获取一次 GPS 定位信息
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER
        , 3000, 8, new LocationListener() // ①
    {
        @Override
        public void onLocationChanged(Location location)
        {
            // 当 GPS 定位信息发生改变时，更新位置
            updateView(location);
        }
        @Override
        public void onProviderDisabled(String provider)
        {
            updateView(null);
        }
        @Override
        public void onProviderEnabled(String provider)
        {
            // 当 GPS LocationProvider 可用时，更新位置
            updateView(locManager
                .getLastKnownLocation(provider));
        }
        @Override
        public void onStatusChanged(String provider, int status,
            Bundle extras)
        {
        }
    });
}
// 更新 EditText 中显示的内容
public void updateView(Location newLocation)
{
    if (newLocation != null)
    {
        StringBuilder sb = new StringBuilder();
        sb.append("实时的位置信息：\n");
        sb.append("经度：");
        sb.append(newLocation.getLongitude());
        sb.append("\n 纬度：");
        sb.append(newLocation.getLatitude());
        sb.append("\n 高度：");
        sb.append(newLocation.getAltitude());
        sb.append("\n 速度：");
        sb.append(newLocation.getSpeed());
        sb.append("\n 方向：");
        sb.append(newLocation.getBearing());
        show.setText(sb.toString());
    }
    else
```

```
        {
            // 如果传入的 Location 对象为空，则清空 EditText
            show.setText("");
        }
    }
}
```

上面程序中的粗体字代码用于从 Location 中获取定位信息，包括用户的经度、纬度、高度、方向和移动速度等信息。程序中①号粗体字代码通过 LocationManager 设置了一个监听器，该监听器负责每隔 3 秒向 LocationProvider 请求一次定位信息，当 LocationProvider 可用时、不可用时或提供的定位信息发生改变时，系统会回调 updateView(Location newLocation)来更新 EditText 中显示的定位信息。

该程序需要有访问 GPS 信号的权限，因此需要在 AndroidManifest.xml 文件中增加如下授权代码片段：

```
<!-- 授权获取定位信息 -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

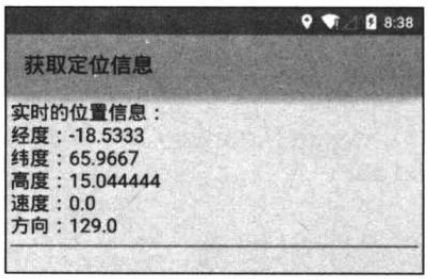


图 16.3　实时获取定位信息

运行该程序，然后通过 DDMS 的 Emulator Control 面板来发送 GPS 定位信息，即可看到该程序显示如图 16.3 所示的输出。

由于该程序每隔 3 秒就会向 GPS LocationProvider 获取一次定位信息，这样上面的程序界面上总可以实时显示该用户的定位信息。

如果把该程序与 Google Map 结合，让该程序根据 GPS 提供的信息实时地显示用户在地图上的位置，即可开发出 GPS 导航系统。下一章会介绍相关内容。

16.4　临近警告

前面介绍 LocationManager 时已经提到，该 API 提供了一个 addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)方法，该方法用于添加一个临近警告。

所谓临近警告的示意图如图 16.4 所示。

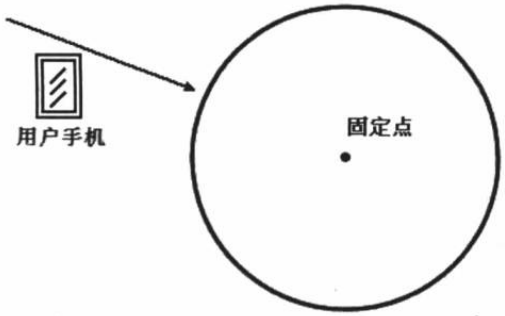


图 16.4　临近警告的示意图

也就是当用户手机不断地临近指定固定点，与该固定点的距离小于指定范围时，系统可以触发相应的处理。

添加临近警告的方法的参数说明如下。

- latitude：指定固定点的经度。
- longitude：指定固定点的纬度。
- radius：指定一个半径长度。
- expiration：指定经过多少毫秒后该临近警告就会过期失效。-1 指定永不过期。
- intent：指定临近该固定点时触发该 intent 对应的组件。

下面的程序示范了如何检测手机是否进入“疯狂软件教育中心”附近，该程序几乎没有界

面。当程序启动后，程序就会添加一个临近警告，当用户临近“疯狂软件教育中心”所在经度、纬度时，系统会显示提示。该程序代码如下。

程序清单：codes\16\16.4\ProximityTest\src\org\crazyit\gps\MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 定位服务的常量
        String locService = Context.LOCATION_SERVICE;
        // 定位服务管理器实例
        LocationManager locationManager;
        // 通过 getSystemService 方法获得 LocationManager 实例
        locationManager = (LocationManager) getSystemService(locService);
        // 定义“疯狂软件教育中心”的经度、纬度
        double longitude = 113.401863;
        double latitude = 23.132636;
        // 定义半径（5 公里）
        float radius = 5000;
        // 定义 Intent
        Intent intent = new Intent(this, ProximityAlertReciever.class);
        // 将 Intent 包装成 PendingIntent
        PendingIntent pi = PendingIntent.getBroadcast(this, -1, intent, 0);
        // 添加临近警告
        locationManager.addProximityAlert(latitude, longitude, radius, -1, pi);
    }
}
```

上面程序中的粗体字代码用于添加临近警告，当用户手机临近指定经度、纬度确定的点时，系统会启动 pi 所对应的组件。pi 对应的组件是一个 BroadcastReceiver，它的代码如下。

程序清单：codes\16\16.4\ProximityTest\src\org\crazyit\gps\ProximityAlertReciever.java

```
public class ProximityAlertReciever extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 获取是否进入指定区域
        boolean isEnter = intent.getBooleanExtra(
            LocationManager.KEY_PROXIMITY_ENTERING, false);
        if(isEnter)
        {
            // 显示提示信息
            Toast.makeText(context
                , "您已经进入疯狂软件教育中心附近"
                , Toast.LENGTH_LONG)
                .show();
        }
        else
        {
            // 显示提示信息
            Toast.makeText(context
                , "您已经离开疯狂软件教育中心附近"
```

```
                , Toast.LENGTH_LONG)
                .show();
        }
    }
}
```

该 BroadcastReceiver 被激发后的处理非常简单，程序通过 Intent 传递过来的消息判断设备是进入指定区域还是离开指定区域，并根据不同的状态提示不同的信息。

运行该程序，并通过 GPS 模拟信息面板输入疯狂软件教育中心的经度、纬度（113.39、23.13），即可看到如图 16.5 所示的提示。

如果接下来在 GPS 模拟信息面板中输入其他的经度、纬度，就意味着该设备离开了该区域，于是可以看到如图 16.6 所示的提示。

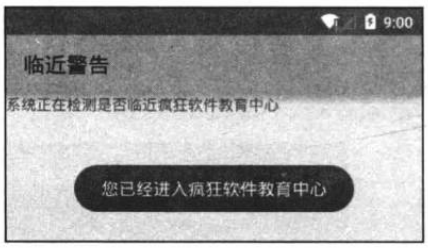


图 16.5　进入区域的提示

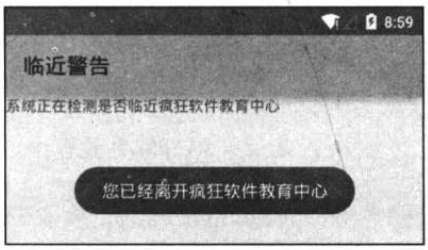


图 16.6　离开区域的提示

16.5　本章小结

本章主要介绍了 Android 提供的 GPS 定位支持，目前的绝大部分 Android 手机都提供了 GPS 硬件支持，都可以作为 GPS 定位系统的接收机，而开发者要做的就是从 Android 系统中获取 GPS 定位信息。学习本章的重点是掌握 LocationManager、LocationProvider 与 LocationListener 等 API 的功能和用法，并可以通过它们来监听、获取 GPS 定位信息。

一旦在应用程序中获取了 GPS 定位信息之后，接下来就可以通过这种定位信息在 Google Map 上进行定位、跟踪等，这就需要结合下一章所介绍的 Map 服务了。

CHAPTER

17

第17章 整合高德Map服务

本章要点

- 了解 Map 服务
- 掌握调用第三方 Map 服务的方法
- 获取第三方 Map 服务的 API Key
- 添加第三方 Map 服务的 SDK
- 根据 GPS 信息在地图上定位
- 根据 GPS 信息在地图上跟踪用户轨迹
- 调用地址解析服务
- 根据地址在地图上定位
- 调用第三方的导航服务

上一章介绍了如何使用 Android 的 GPS 来获取设备的定位信息，但这种方式得到的定位信息只不过是一些数字的经度、纬度值，如果这些经度、纬度值不能以更形象、直观的方式显示出来，对于大部分普通用户而言，这些经度、纬度数据几乎没有任何价值。

为了让上一章介绍的 GPS 信息“派上”用场，本章将会详细介绍 Android 调用第三方的 Map 服务，本书将以高德 Map 服务为例来介绍如何在 Android 应用中嵌入高德地图。由于众所周知的某些原因，现在 Google 的各种服务基本上都已经被封锁了，因此本书删除了第 2 版中调用 Google 地图服务的相关知识，改为讲授在 Android 应用中调用第三方地图服务。如果把上一章获得的 GPS 信息与本章的 Map 应用结合起来，就可以非常方便地开发出定位、导航等应用程序。

提示：

如果读者依然需要学习关于调用 Google 地图服务的相关知识，请参考本书第 2 版。

17.1 调用高德 Map 服务

国内比较常用的地图服务有高德地图服务（相对比较专业，iOS 设备也采用这种地图）、百度地图服务等，不管使用哪一家的地图服务，大致的调用步骤都基本相似，读者只要熟练掌握其中一种即可。

17.1.1 获取 Map API Key

为了在应用程序中调用第三方 Map 服务，必须先获取第三方 Map 服务的 API Key。此处以获取高德地图服务的 API Key 为例进行介绍，获取步骤如下。

① 首先找到该 App 的数字证书的 keystore 的存储路径，此时可分为两种情况。

- 如果是作为实际产品发布的签名 App，该 App 的数字证书的 keystore 将保存在用户自定义的路径中，该自定义路径也就是第 1 章中图 1.42 创建数字证书时所指定的存储路径。
- 如果是作为调试阶段的调试 App，该 App 的数字证书的 keystore 通常保存在 ANDROID_SDK_HOME 环境变量对应的路径的.android/目录下，在该目录下可以找到一个 debug.keystore 文件，该文件就是调试 App 的数字证书的 keystore 的存储路径。比如笔者的 ANDROID_SDK_HOME 环境变量值为 D:\AVD，那么对应的 keystore 的存储路径为 D:\AVD\.android\debug.keystore。

② 为了获取第三方 Map 服务的 API Key，需要先用 JDK 提供的 keytool 工具查看 keystore 的认证指纹。启动命令行窗口，输入如下命令：

```
keytool -list -v -keystore <Android keystore 的存储位置>
```

将上面命令中的<Android keystore 的存储位置>替换成 App 的数字证书的 keystore 的存储位置（如果发布 Android 应用，则应该使用本公司的 keystore 的存储路径）。

运行上面的命令，系统将会提示“输入 keystore 密码”，输入 Android 模拟器的 keystore 的默认密码：android，系统将会显示该 keystore 对应的认证指纹。图 17.1 显示了查看 keystore 的认证指纹的详细过程。

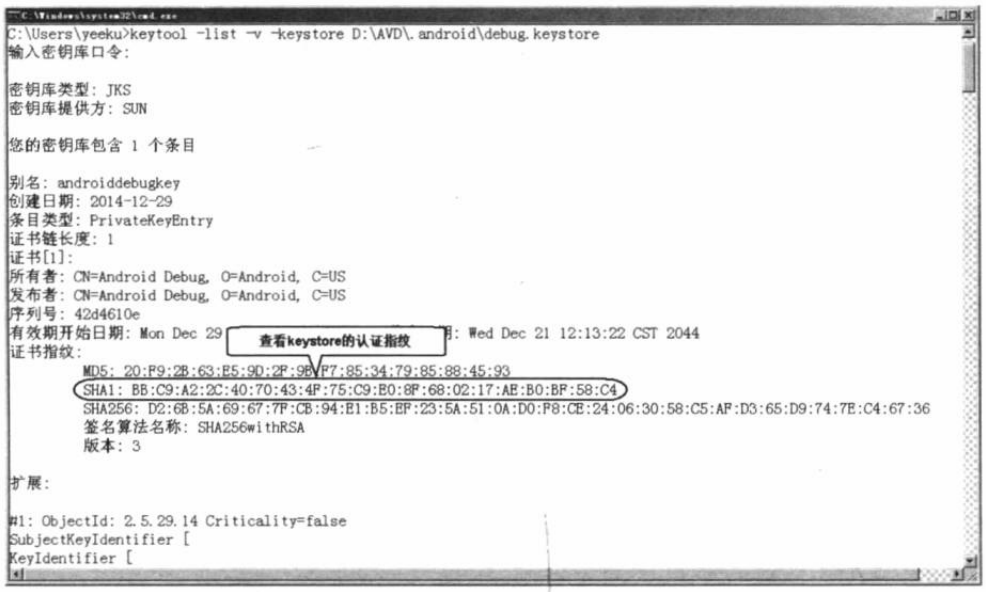


图 17.1 查看调试 App 的数字证书的 keystore 的认证指纹

提示：

如果运行 keytool 工具时系统提示“找不到该命令”，则说明还未在 PATH 环境变量中增加%JAVA_HOME%/bin 路径，其中%JAVA_HOME%代表 JDK 的安装路径——JDK 的安装路径的 bin 子目录下应该包含 java.exe、javac.exe 和 keytool.exe 工具。

③ 记住图 17.1 所显示的 SHA1 对应的认证指纹，登录 http://id.amap.com/站点，系统显示如图 17.2 所示的登录高德地图的页面。

图 17.2 登录高德地图

④ 如果读者已有高德地图的用户名、密码，那么输入用户名、密码登录即可。如果读者还没有高德地图的用户名、密码，则需要通过该页面右边的“立即免费注册”链接注册一个新的账户。

⑤ 注册完成后系统会提示用户还没有“成为开发者”，读者可通过页面中间的链接申请“成为开发者”，在申请成为开发者时，页面会提示用户输入邮箱和手机号码（请输入真实的邮箱和手机号码，因为高德会采用短信+邮箱的方式进行验证），提交请求，并通过邮件激活之后，此时就拥有了一个高德的开发者账号。

⑥ 登录 http://lbs.amap.com/console/页面，即可看到如图 17.3 所示的页面。

图 17.3　申请 Key

提示：

如果读者第一次注册、登录高德地图的网站，可能看不到上面列出的 maptest 那条记录，这条记录是本书为测试 Map 应用所获取的 Key。

⑦ 单击“获取 KEY”按钮，将显示如图 17.4 所示的输入页面。

图 17.4　填写申请 Key 的信息

⑧ 在文本框中输入 keytool 工具查看到的 SHA1 指纹，单击“获取 KEY”按钮，系统返回如图 17.3 所示的页面，即可在该页面中看到该应用（包名+SHA1 签名即可唯一地确定一个应用）对应的 API Key。

➤➤ 17.1.2　高德地图入门

一旦为指定应用申请了 API Key 之后，接下来即可非常简单地在应用中使用高德地图了。高德地图提供了一个 MapView 组件，这个 MapView 继承了 FrameLayout，因此它的本质就是一个普通的容器控件，所以开发者可以直接将该 MapView 添加到应用界面上。

MapView 只是一个容器，真正为 MapView 提供地图支持的是 AMap 类，MapView 可通过 getMap()方法来获取它所封装的 AMap 对象，AMap 对象则提供了大量的方法来控制地图。

注意：

前面在申请 API Key 时填写的应用包名是什么，那么此处新建项目的包名也必须与之相同。

为了给 Android 应用添加高德地图支持，必须为该应用添加高德地图的 SDK。添加高德地图的 SDK 请按如下步骤进行。

1 登录 http://lbs.amap.com/api/android-sdk/down/站点，即可看到如图 17.5 所示的页面。

图 17.5 下载高德地图的 SDK

2 由于本书将以高德 3D 地图为例，因此读者应该下载图 17.5 所示列表中的第 1 项（3D 地图）和第 3 项（支持地址解析和反向地址解析等功能）。下载完成后得到 Android_3DMap_V2.4.0.jar_.zip 和 AMap_Services_V2.3.1.jar_.zip 两个压缩包。

3 将上面两个压缩包解压，解压之后得到两个 JAR 包，以及 armeabi、armeabi-v7a、x86 这三个文件夹。

4 将第 3 步解压得到的两个 JAR 包复制到 Android 应用的 app/libs/目录下，再将 Android Studio 左边面板切换成 Project 面板，然后选中这两个 JAR 包并单击右键，通过右键菜单中的“Add As Library...”菜单项将这两个 JAR 包添加到该应用中。

5 在 Android 应用的 app/src/main/目录下新建一个 jniLibs 子目录，并将第 3 步解压得到的 armeabi、armeabi-v7a、x86 这三个文件夹复制到该目录下。

提示：

实际上第 4 步是通过 Android Studio 为 App 添加第三方 JAR 包的方法；第 5 步则是通过 Android Studio 为 App 添加第三方*.so 包（*.so 文件似乎是 Linux 平台的动态链接库文件，类似于 Windows 平台的*.dll 文件）的方法。如果读者使用的是 Eclipse+ADT 开发平台，那就更简单了，直接将两个 JAR 包和 armeabi、armeabi-v7a、x86 这三个文件夹复制到 Android 应用的 libs 目录下即可，Eclipse 会自动加载它们。

6 接下来打开 Android 应用的 AndroidManifest.xml 文件，在该文件的<application.../>元

素内添加如下<meta-data.../>子元素：

```
<!-- 启用高德地图服务 -->
<meta-data
    android:name="com.amap.api.v2.apikey"
    android:value="0524e572da9858235ecdb6e3893c9928"/>
```

千万不要直接照着书上输入，因为上面<meta-data.../>元素用于启用高德地图支持，而其中的 android:value 属性值应该填写前面申请得到的 API Key。

⑦ 在 Android 应用的 AndroidManifest.xml 文件中添加如下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
```

经过上面步骤，高德地图 SDK 添加完成，剩下的事情就是使用 MapView 组件了，而 MapView 组件与普通 Android 组件的区别并不大。本应用的界面设计文件如下。

程序清单：codes\17\17.1\AMapTest\app\src\main\res\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<!-- 使用高德地图提供的 MapView -->
<com.amap.api.maps.MapView
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
<ToggleButton
    android:id="@+id/tb"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="top|right"
    android:textOff="普通地图"
    android:textOn="卫星地图"
    android:checked="false"/>
</FrameLayout>
```

正如从上面粗体字代码所看到的，程序使用 MapView 的方式与使用普通 Android View 并没有太大的区别。需要指出的是，MapView 要求在其所在的 Activity 的生命周期方法中回调该 MapView 的生命周期方法。下面是该应用的 Activity 代码。

程序清单：codes\17\17.1\AMapTest\app\src\main\java\org\crazyit\map\MainActivity.java

```
public class MainActivity extends Activity {
    private MapView mapView;
    private AMap aMap;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mapView = (MapView) findViewById(R.id.map);
        // 必须回调 MapView 的 onCreate()方法
        mapView.onCreate(savedInstanceState);
        init();
```

```
        ToggleButton tb = (ToggleButton) findViewById(R.id.tb);
        tb.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
            {
                if(isChecked)
                {
                    // 设置使用卫星地图
                    aMap.setMapType(AMap.MAP_TYPE_SATELLITE);
                }
                else
                {
                    // 设置使用普通地图
                    aMap.setMapType(AMap.MAP_TYPE_NORMAL);
                }
            }
        });
    }
    // 初始化 AMap 对象
    private void init() {
        if (aMap == null) {
            aMap = mapView.getMap();
        }
    }
    @Override
    protected void onResume() {
        super.onResume();
        // 必须回调 MapView 的 onResume()方法
        mapView.onResume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        // 必须回调 MapView 的 onPause()方法
        mapView.onPause();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        // 必须回调 MapView 的 onSaveInstanceState()方法
        mapView.onSaveInstanceState(outState);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // 必须回调 MapView 的 onDestroy()方法
        mapView.onDestroy();
    }
}
```

正如从上面粗体字代码所看到的，程序只是简单地获取了界面上的 MapView 组件，并在该 Activity 的生命周期方法内回调了 MapView 的生命周期方法，整个应用就完成了。本应用为了简单示范 AMap 的功能，程序为界面上的 ToggleButton 添加了事件监听器，用于切换地图的显示方式。

运行该程序，切换到卫星地图，即可看到如图 17.6 所示的地图界面。

图 17.6 地图

17.2 根据GPS信息在地图上定位

通过MapView获取AMap对象之后，接下来即可通过AMap来控制地图的显示形式和显示外观了。AMap提供了如下常用方法来控制地图。

- setLocationSource(LocationSource locationSource)：设置定位数据源。
- setMapType(int type)：设置地图显示的类型。
- setMyLocationEnabled(boolean paramBoolean)：设置定位层是否显示。
- setMyLocationRotateAngle(float rotate)：设置定位图片旋转的角度，从正北向开始，逆时针计算。
- setMyLocationStyle(MyLocationStyle style)：设置定位（当前位置）的绘制样式。
- setMyLocationType(int type)：设置定位的类型。该方法支持三个参数值，分别为LOCATION_TYPE_LOCATE，表示只在第一次定位时移动到地图中心点；LOCATION_TYPE_MAP_FOLLOW，表示总是定位、移动到地图中心点；LOCATION_TYPE_MAP_ROTATE，表示总是定位、移动到地图中心点，跟踪并根据方向旋转地图。
- setPointToCenter(int x, int y)：设置屏幕上的某个点为地图中心点。
- setTrafficEnabled(boolean enabled)：设置是否显示交通状况。

除此之外，AMap还提供了大量的setOnXxxListener()方法，这些方法都需要传入相应的监听器实现类，通过这些监听器可以让地图响应用户操作。

如果用户希望向地图上添加自定义的图片或形状，AMap提供了如下常用方法。

- addArc(ArcOptions options)：在地图上添加一段扇形覆盖物（Arc）。
- addCircle(CircleOptions options)：在地图上添加圆形（Circle）覆盖物。
- addGroundOverlay(GroundOverlayOptions options)：在地图上添加图片。
- addMarker(MarkerOptions options)：在地图上添加标记（Marker）。
- addMarkers(java.util.ArrayList<MarkerOptions> list, boolean moveToCenter)：在地图上添加多个标记，并设置是否移动到屏幕中间。
- Polygon addPolygon(PolygonOptions options)：在地图上添加一个多边形（Polygon）。
- Polyline addPolyline(PolylineOptions options)：在地图上添加多条线段（Polyline）。
- addTileOverlay(TileOverlayOptions options)：在地图上添加tile overlay图层。

不管程序需要向地图上添加哪种东西，程序的操作步骤都大致相同，都可按如下步骤进行。

1. 创建一个XxxOptions，比如要添加Marker对象，程序就需要先创建一个MarkerOptions；比如添加Polyline，就需要创建PolylineOptions。
2. 调用XxxOptions的各种setter方法来设置属性。
3. 调用AMap对象的addXxx()方法添加即可。

提示：

高德地图的API文档本来就是用中文写成的，因此读者可以直接通过高德官网来了解AMap的各方法的功能和用法。

为了表示Map上的指定点，高德地图提供了LatLng类。LatLng十分简单，它就是对纬度、经度的封装。除此之外，高德地图还提供了一个CameraUpdate类（该类并未提供构造器，程

序应该通过 CameraUpdateFactory 来创建该类的实例），CameraUpdate 可控制地图的缩放级别、定位、倾斜角度等信息，程序调用 AMap 的 moveCamera(CameraUpdate update)方法根据 CameraUpdate 对地图进行缩放、定位和倾斜。

掌握了高德地图的上面常用的 API 之后，接下来就可以开发 Android 的 Map 应用了。

下面的程序示范了如何根据经度、纬度在地图上定位。该程序的界面提供了文本框让用户输入经度、纬度，也允许用户设置通过 GPS 信号在地图上定位。该程序的界面布局代码如下。

程序清单：codes\17\17.2\LocationMap\app\src\main\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="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal">
        <TextView android:text="@string/txtLong"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <!-- 定义输入经度值的文本框 -->
        <EditText android:id="@+id/lng"
            android:text="@string/lng"
            android:inputType="numberDecimal"
            android:layout_width="85dp"
            android:layout_height="wrap_content" />
        <TextView android:text="@string/txtLat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingLeft="8dp" />
        <!-- 定义输入纬度值的文本框 -->
        <EditText android:id="@+id/lat"
            android:text="@string/lat"
            android:inputType="numberDecimal"
            android:layout_width="85dp"
            android:layout_height="wrap_content" />
        <Button android:id="@+id/loc"
            android:text="@string/loc"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="4" />
    </LinearLayout>
    <LinearLayout android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal">
        <!-- 定义选择定位方式的单选钮组 -->
        <RadioGroup android:id="@+id/rg"
            android:orientation="horizontal"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1">
            <RadioButton
                android:text="@string/manul"
                android:id="@+id/manual"
                android:checked="true"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>
            <RadioButton
                android:text="@string/gps"
                android:id="@+id/gps"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        </RadioGroup>
    </LinearLayout>
    <!-- 使用高德地图提供的 MapView -->
    <com.amap.api.maps.MapView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

提供了上面介绍的布局之后，接下来就可以在程序中根据用户输入的经度、纬度来进行定位，也可以根据 GPS 传入的信号进行定位。根据经度、纬度在 AMap 上定位的步骤如下。

1. 根据程序获取的经度、纬度值创建 LatLng 对象。
2. 调用 CameraUpdateFactory 的 changeLatLng()方法创建改变地图中心的 Camera 对象。
3. 调用 AMap 的 moveCamera(CameraUpdate update)即可控制地图定位到指定位置。

该示例程序的代码如下。

程序清单：codes\17\17.2\LocationMap\app\src\main\java\org\crazyit\map\MainActivity.java

```
public class MainActivity extends Activity {
    private MapView mapView;
    private AMap aMap;
    private LocationManager locationManager;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        locationManager = (LocationManager) getSystemService(Context
            .LOCATION_SERVICE);
        mapView = (MapView) findViewById(R.id.map);
        // 必须回调 MapView 的 onCreate()方法
        mapView.onCreate(savedInstanceState);
        init();
        RadioButton rb = (RadioButton) findViewById(R.id.gps);
        // 为 GPS 单选钮设置监听器
        rb.setOnCheckedChangeListener(new OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(CompoundButton buttonView
                , boolean isChecked)
            {
                // 如果该单选钮已经被选中
                if(isChecked)
                {
                    // 通过监听器监听 GPS 提供的定位信息的改变
                    locationManager.requestLocationUpdates(
                        LocationManager.GPS_PROVIDER,
                        300, 8 , new LocationListener()
                        {
                            @Override
                            public void onLocationChanged(Location loc)
                            {
                                // 使用 GPS 提供的定位信息来更新位置
                                updatePosition(loc);
                            }
                            @Override
                            public void onStatusChanged(String provider
                                , int status, Bundle extras){}
                            @Override
                            public void onProviderEnabled(String provider)
```

```
                {
                    // 使用GPS提供的定位信息来更新位置
                    updatePosition(locationManager
                        .getLastKnownLocation(provider));
                }
                @Override
                public void onProviderDisabled(String provider){}
            });
        }
    }
});
Button bn = (Button) findViewById(R.id.loc);
final TextView latTv = (TextView) findViewById(R.id.lat);
final TextView lngTv = (TextView) findViewById(R.id.lng);
bn.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 获取用户输入的经度、纬度值
        String lng = lngTv.getEditableText().toString().trim();
        String lat = latTv.getEditableText().toString().trim();
        if (lng.equals("") || lat.equals(""))
        {
            Toast.makeText(MainActivity.this, "请输入有效的经度、纬度!",
                Toast.LENGTH_SHORT).show();
        }
        else
        {
            // 设置根据用户输入的地址定位
            ((RadioButton)findViewById(R.id.manual)).setChecked(true);
            double dLng = Double.parseDouble(lng);
            double dLat = Double.parseDouble(lat);
            // 将用户输入的经度、纬度封装成LatLng
            LatLng pos = new LatLng(dLat, dLng); // ①
            // 创建一个设置经纬度的CameraUpdate
            CameraUpdate cu = CameraUpdateFactory.changeLatLng(pos); // ②
            // 更新地图的显示区域
            aMap.moveCamera(cu); // ③
            // 创建MarkerOptions对象
            MarkerOptions markerOptions = new MarkerOptions();
            // 设置MarkerOptions的添加位置
            markerOptions.position(pos);
            // 设置MarkerOptions的标题
            markerOptions.title("疯狂软件教育中心");
            // 设置MarkerOptions的摘录信息
            markerOptions.snippet("专业的Java、iOS培训中心");
            // 设置MarkerOptions的图标
            markerOptions.icon(BitmapDescriptorFactory
                .defaultMarker(BitmapDescriptorFactory.HUE_RED));
            markerOptions.draggable(true);
            // 添加MarkerOptions（实际上就是添加Marker）
            Marker marker = aMap.addMarker(markerOptions);
            marker.showInfoWindow(); // 设置默认显示信息窗
            // 创建MarkerOptions，并设置它的各种属性
            MarkerOptions markerOptions1 = new MarkerOptions();
                markerOptions1.position(new LatLng(dLat + 0.001, dLng))
                // 设置标题
                .title("疯狂软件教育中心学生食堂")
                .icon(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_MAGENTA))
                .draggable(true);
```

```
                // 使用集合封装多个图标，这样可为MarkerOptions设置多个图标
                ArrayList<BitmapDescriptor> giflist = new ArrayList<>();
                giflist.add(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
                giflist.add(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_GREEN));
                giflist.add(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_YELLOW));
                // 再创建一个MarkerOptions，并设置它的各种属性
                MarkerOptions markerOptions2 = new MarkerOptions()
                    .position(new LatLng(dLat - 0.001, dLng))
                    // 为MarkerOptions设置多个图标
                    .icons(giflist)
                    .title("疯狂软件教育中心学生宿舍")
                    .draggable(true)
                    // 设置图标的切换频率
                    .period(10);
                // 使用ArrayList封装多个MarkerOptions，即可一次添加多个Marker
                ArrayList<MarkerOptions> optionList = new ArrayList<>();
                optionList.add(markerOptions1);
                optionList.add(markerOptions2);
                // 批量添加多个Marker
                aMap.addMarkers(optionList, true);
            }
        }
    });
}
private void updatePosition(Location location)
{
    LatLng pos = new LatLng(location.getLatitude(), location.getLongitude());
    // 创建一个设置经纬度的CameraUpdate
    CameraUpdate cu = CameraUpdateFactory.changeLatLng(pos);
    // 更新地图的显示区域
    aMap.moveCamera(cu);
    // 清除所有Marker等覆盖物
    aMap.clear();
    // 创建一个MarkerOptions对象
    MarkerOptions markerOptions = new MarkerOptions();
    markerOptions.position(pos);
    // 设置MarkerOptions使用自定义图标
    markerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.car));
    markerOptions.draggable(true);
    // 添加MarkerOptions（实际上是添加Marker）
    Marker marker = aMap.addMarker(markerOptions);
}
// 初始化AMap对象
private void init() {
    if (aMap == null) {
        aMap = mapView.getMap();
        // 创建一个设置放大级别的CameraUpdate
        CameraUpdate cu = CameraUpdateFactory.zoomTo(15);
        // 设置地图的默认放大级别
        aMap.moveCamera(cu);
        // 创建一个更改地图倾斜度的CameraUpdate
        CameraUpdate tiltUpdate = CameraUpdateFactory.changeTilt(30);
        // 改变地图的倾斜度
        aMap.moveCamera(tiltUpdate);
    }
}
@Override
protected void onResume() {
    super.onResume();
```

```
        // 必须回调 MapView 的 onResume()方法
        mapView.onResume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        // 必须回调 MapView 的 onPause()方法
        mapView.onPause();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        // 必须回调 MapView 的 onSaveInstanceState()方法
        mapView.onSaveInstanceState(outState);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // 必须回调 MapView 的 onDestroy()方法
        mapView.onDestroy();
    }
}
```

上面程序中的①、②、③号粗体字代码分别用于创建按 LatLng 对象，并根据 LatLng 对象创建改变地图中心点的 CameraUpdate 对象，将地图定位到指定位置点。

程序中第二段粗体字代码示范了如何向地图上添加 MarkerOptions——实际上是向地图上添加一个 Marker。

运行上面的程序，将看到如图 17.7 所示的结果。

如果读者开发该程序时一切正常，将可以看到如图 17.7 所示的地图，在应用界面上方的文本框中输入合适的经度、纬度，地图将会定位到指定的位置。

图 17.7 所示的地图界面上有 3 个 Marker，其中最下面一个 Marker 会不停地闪烁，这是因为程序给最下面一个 Marker 设置了 3 个不同颜色图标。

如果用户选中界面上的“GPS 定位”单选钮，程序将会使用 LocationManager 不断地获取 GPS 信号，并根据 GPS 信号来调整地图中心，而且程序还在地图定位点添加了一个汽车图标。运行程序可看到如图 17.8 所示的界面。

图 17.7 根据经纬度定位

图 17.8 根据 GPS 定位

17.3 执行定位

前面介绍的在地图上定位的例子需要用户输入定位点的经度、纬度才能进行定位，这种方式显然不太现实：普通用户不太可能记住某个位置的经度、纬度。对于普通用户来说，根据地址进行定位才是更有实用价值的地图。本节将会介绍如何根据地址在地图上定位。

17.3.1 地址解析与反向地址解析

通常来说，地图定位必须根据经度、纬度来完成，因此，如果需要让程序根据地址进行定位，则需要先把地址解析成经度、纬度。这里涉及如下两个基本概念。

- **地址解析**：把普通用户能看到的字符串地址转换为经度、纬度。
- **反向地址解析**：把经度、纬度值转换成普通的字符串地址。

高德地图搜索服务为地址解析提供了 GeocodeSearch 工具类，该工具类提供了如下方法来进行地址解析和反向地址解析。

- RegeocodeAddress getFromLocation(RegeocodeQuery query)：根据给定的经纬度和最大结果数返回反向地址解析的结果。
- getFromLocationAsyn(RegeocodeQuery query)：该方法与前一个方法类似，只是该方法以异步方式进行。
- java.util.List<GeocodeAddress> getFromLocationName(GeocodeQuery query)：根据给定的地理名称、城市来返回地址解析的结果列表。
- getFromLocationNameAsyn(GeocodeQuery query)：该方法与前一个方法类似，只是该方法以异步方式进行。

虽然 GeocodeSearch 工具类提供了上面方法来进行地址解析和反向地址解析，但实际上这个类还是需要调用网络上高德的查询服务。很明显，Android 平台不大可能把地球上的所有地名与经度、纬度之间的映射关系都存储在手机系统中。

由于 Android 平台要求所有的网络访问都不能直接放在 UI 线程中进行，因此 Android 应用进行地址解析或反向地址解析时都应该采用异步方式进行。当程序采用异步方式进行地址解析或反向地址解析时，程序还必须为 GeocodeSearch 设置一个监听器，当解析完成时，该监听器所包含的方法将会被触发。

借助于 GeocodeSearch 提供的地址解析、反向地址解析的功能，接下来只要按如下步骤操作即可完成地址解析和反向地址解析。

1. 创建 GeocodeSearch 对象，并为该对象设置解析监听器。
2. 如果要进行地址解析，程序先创建 GeocodeQuery 对象，该对象封装了要解析的地理名称、城市等信息；如果要进行反向地址解析，程序先创建 LatLonPoint 对象，该对象封装了经度、纬度信息。
3. 调用 GeocodeSearch 对象的方法执行地址解析或反向地址解析。

下面的示例示范了如何使用 GeocodeSearch 来进行地址解析、反向地址解析。该应用程序的 Activity 代码如下。

程序清单：codes\17\17.3\GeocoderTest\app\src\main\java\org\crazyit\map\MainActivity.java

```
public class MainActivity extends Activity
    implements OnClickListener
{
    Button parseBn, reverseBn;
```

```
EditText etLng, etLat, etAddress, etResult;
GeocodeSearch search;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 获取界面中的可视化组件
    parseBn = (Button) findViewById(R.id.parse);
    reverseBn = (Button) findViewById(R.id.reverse);
    etLng = (EditText) findViewById(R.id.lng);
    etLat = (EditText) findViewById(R.id.lat);
    etAddress = (EditText) findViewById(R.id.address);
    etResult = (EditText) findViewById(R.id.result);
    parseBn.setOnClickListener(this);
    reverseBn.setOnClickListener(this);
    // 创建GeocodeSearch对象
    search = new GeocodeSearch(this);
    // 设置解析监听器
    search.setOnGeocodeSearchListener(new OnGeocodeSearchListener()  // ①
    {
        @Override
        public void onRegeocodeSearched(RegeocodeResult
            regeocodeResult, int i)
        {
            RegeocodeAddress addr = regeocodeResult.getRegeocodeAddress();
            etResult.setText("经度:" + etLng.getText() + "、纬度:"
                + etLat.getText() + "的地址为: \n" + addr.getFormatAddress());
        }
        @Override
        public void onGeocodeSearched(GeocodeResult
            geocodeResult, int i)
        {
            GeocodeAddress addr = geocodeResult.getGeocodeAddressList().get(0);
            LatLonPoint latlng = addr.getLatLonPoint();
            etResult.setText(etAddress.getText() + "的经度是:" +
                latlng.getLongitude() + "、纬度是:" + latlng.getLatitude());
        }
    });
}
@Override
public void onClick(View source)
{
    switch (source.getId())
    {
        // 单击了“解析”按钮
        case R.id.parse:
            String address = etAddress.getText().toString().trim();
            if (address.equals(""))
            {
                Toast.makeText(this, "请输入有效的地址"
                    , Toast.LENGTH_LONG).show();
            }
            else
            {
                GeocodeQuery query = new GeocodeQuery(address, "广州");
                // 根据地理名称执行异步解析
                search.getFromLocationNameAsyn(query);  // ②
            }
            break;
        // 单击了“反向解析”按钮
        case R.id.reverse:
            String lng = etLng.getText().toString().trim();
```

```
                String lat = etLat.getText().toString().trim();
                if (lng.equals("") || lat.equals(""))
                {
                    Toast.makeText(this, "请输入有效的经度、纬度!"
                        , Toast.LENGTH_LONG)
                        .show();
                }
                else
                {
                    // 根据经纬度执行异步查询
                    search.getFromLocationAsyn(new RegeocodeQuery(
                        new LatLonPoint(Double.parseDouble(lat)
                        , Double.parseDouble(lng))
                        , 20 // 区域半径
                        , GeocodeSearch.GPS)); // ③
                }
                break;
            }
        }
    }
```

上面程序中的①号粗体字代码为 GeocodeSearch 对象设置了一个解析监听器，当程序调用该对象的方法执行异步解析完成后，程序将会自动触发该监听器包含的方法。

程序中②号粗体字代码以异步方式执行地址解析，在②号代码之前先创建了一个 GeocodeQuery 对象，该对象包含要解析的地理名称、城市等信息。

程序中③号粗体字代码以异步方式执行反向地址解析，执行③号代码时传入了一个 LatLonPoint 对象，该对象就包装了要进行反向地址解析的经度、纬度。

运行上面的程序，在文本框中输入某个地名，单击“解析”按钮，将可看到如图 17.9 所示的结果。

在图 17.9 所示的输入经度、纬度文本框中输入经度、纬度值，然后单击“反向解析”按钮，即可看到程序有如图 17.10 所示的输出。

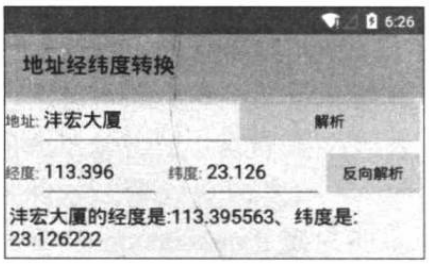


图 17.9　地址解析

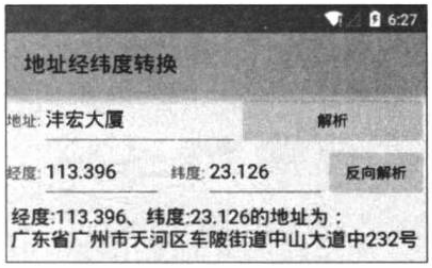


图 17.10　反向地址解析

➤➤ 17.3.2　执行定位

下面的应用程序是对 17.2 节示例程序的改进，该应用程序不需要用户输入经度、纬度值，只要用户输入目标地址，程序就会调用 GeocodeSearch 对目标地址执行地址解析，将其转换为经度、纬度值，再控制地图定位到指定地址点即可。

该程序代码如下。

程序清单：codes\17\17.3\AddrLocMap\app\src\main\java\org\crazyit\map\MainActivity.java

```
public class MainActivity extends Activity
    implements GeocodeSearch.OnGeocodeSearchListener
{
    private MapView mapView;
    private AMap aMap;
    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mapView = (MapView) findViewById(R.id.map);
    // 必须回调MapView的onCreate()方法
    mapView.onCreate(savedInstanceState);
    init();
    Button bn = (Button) findViewById(R.id.loc);
    final TextView addrTv = (TextView) findViewById(R.id.address);
    bn.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            String addr = addrTv.getText().toString();
            if (addr.equals(""))
            {
                Toast.makeText(MainActivity.this, "请输入有效的地址"
                    , Toast.LENGTH_LONG).show();
            }
            else
            {
                GeocodeSearch search = new GeocodeSearch(MainActivity.this);
                search.setOnGeocodeSearchListener(MainActivity.this);
                GeocodeQuery query = new GeocodeQuery(addr, "广州");
                // 根据地址执行异步地址解析
                search.getFromLocationNameAsyn(query); // ①
            }
        }
    });
}
// 初始化AMap对象
private void init() {
    if (aMap == null) {
        aMap = mapView.getMap();
        // 创建一个设置放大级别的CameraUpdate
        CameraUpdate cu = CameraUpdateFactory.zoomTo(16);
        // 设置地图的默认放大级别
        aMap.moveCamera(cu);
    }
}
@Override
protected void onResume() {
    super.onResume();
    // 必须回调MapView的onResume()方法
    mapView.onResume();
}
@Override
protected void onPause() {
    super.onPause();
    // 必须回调MapView的onPause()方法
    mapView.onPause();
}
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // 必须回调MapView的onSaveInstanceState()方法
    mapView.onSaveInstanceState(outState);
}
@Override
protected void onDestroy() {
    super.onDestroy();
    // 必须回调MapView的onDestroy()方法
```

```
        mapView.onDestroy();
    }
    @Override
    public void onRegeocodeSearched(RegeocodeResult
        regeocodeResult, int i){}
    @Override
    public void onGeocodeSearched(GeocodeResult geocodeResult, int i)
    {
        // 获取解析得到的第一个地址
        GeocodeAddress geo = geocodeResult.getGeocodeAddressList().get(0);
        // 获取解析得到的经纬度
        LatLonPoint pos = geo.getLatLonPoint();
        LatLng targetPos = new LatLng(pos.getLatitude(), pos.getLongitude());
        // 创建一个设置经纬度的 CameraUpdate
        CameraUpdate cu = CameraUpdateFactory.changeLatLng(targetPos);
        // 更新地图的显示区域
        aMap.moveCamera(cu);
        // 创建一个 GroundOverlayOptions（用于向地图上添加图片）
        GroundOverlayOptions options = new GroundOverlayOptions()
            // 设置 GroundOverlayOptions 包装的图片
            .image(BitmapDescriptorFactory
                .fromResource(R.drawable.ic_launcher))
            .position(targetPos , 64);
        // 添加图片
        aMap.addGroundOverlay(options);
        // 创建一个 CircleOptions（用于向地图上添加圆形）
        CircleOptions cOptions = new CircleOptions()
            .center(targetPos)  // 设置圆心
            .fillColor(0x80ffff00)  // 设置圆形的填充颜色
            .radius(80)  // 设置圆形的半径
            .strokeWidth(1)  // 设置圆形的线条宽度
            .strokeColor(0xff000000);  // 设置圆形的线条颜色
        aMap.addCircle(cOptions);
    }
    private void GroundOverlayOptions(){}
}
```

上面程序中的①号粗体字代码调用 GeocodeSearch 对象执行异步的地址解析，getFromLocationNameAsy()方法解析完成后将会触发解析监听器的 onGeocodeSearched()方法。上面程序的重点就是重写监听器的这个方法，程序在该方法中完成了如下三件事情。

- 根据地址解析得到的经度、纬度创建一个 CameraUpdate，并将地图中心改变到此处。
- 根据地址解析得到的经度、纬度添加一个 GroundOverlay（自定义图片）。
- 根据地址解析得到的经度、纬度添加一个 Circle（圆形区域）。

图 17.11 根据地址定位

上面三件事情的后两件事情的本质是一样的，因此程序的添加步骤也大同小异。编译、运行该程序，并在程序界面上的文本框中输入某个地址，然后单击“定位”按钮，即可看到如图 17.11 所示的结果。

经过前面的介绍不难发现，在 Android 应用中整合第三方 Map 服务其实比较简单，通过为 Android 应用整合第三方 Map 服务，可以开发出功能更强大的 Android

应用，比如为 SNS 系统增加地图支持，即可让用户实时查询好友的位置等。总之，通过在 Android 应用中整合第三方 Map 服务功能，就给开发 Android 应用提供了更多的可能性。

17.4 GPS 导航

高德查询服务还可让开发者查询、规划路线。高德查询服务提供了 RouteSearch 查询类，专门用于查询各种路线。该查询类支持的查询功能由如下方法实现：

- BusRouteResult calculateBusRoute(BusRouteQuery query)：根据指定参数来查询公交路线。
- void calculateBusRouteAsyn(BusRouteQuery query)：该方法与前一个方法功能类似，只是该方法以异步方式执行查询。
- DriveRouteResult calculateDriveRoute(DriveRouteQuery query)：根据指定参数来查询驾车路线。
- void calculateDriveRouteAsyn(DriveRouteQuery query)：该方法与前一个方法功能类似，只是该方法以异步方式执行查询。
- WalkRouteResult calculateWalkRoute(WalkRouteQuery query)：根据指定参数来查询步行路线。
- void calculateWalkRouteAsyn(WalkRouteQuery query)：该方法与前一个方法功能类似，只是该方法以异步方式执行查询。

与前面各种 XxxSearch 类相似的是，RouteSearch 查询的数据同样来自网络，因此在 Android 应用中调用这些查询方法，需要使用异步方式进行查询。为了使用 RouteSearch 执行异步查询，程序还需要为 RouteSearch 指定一个查询监听器（实现了 RouteSearch.OnRouteSearchListener 接口的对象）。

在最简单的情况下，程序只要通过 RouteSearch 查询、规划路线，然后实时地显示用户的当前位置，并提示用户是否正在规划路线上行驶，这样就可以实现一个最简单的导航软件了。

使用 RouteSearch 获取导航路线只要如下 3 步。

1. 创建 RouteSearch 对象，并为该对象设置对应的监听器。
2. 根据要查询路线类型的不同，分别创建对应的 XxxQuery 对象（比如查询公交路线，需要创建 BusRouteQuery；查询步行路线，创建 WalkRouteQuery；查询驾车路线，创建 DriveRouteQuery），这些 XxxQuery 中封装了相应的路线信息，比如起点、终点、途经点、绕行区域等。
3. 调用 RouteSearch 对象的方法执行查询。

当程序通过重写 RouteSearch 监听器的方法获取查询返回的路线之后，该规划路线的本质其实就是地图上的多条线段，因此程序只要通过 AMap 的 addPolyline()把这些规划路线绘制上去即可。

下面的示例示范了如何开发一个简单的 GPS 导航软件，该导航软件主要实现了两个功能：①通过 RouteSearch 查询、规划行车路线；②根据 GPS 信号显示用户的当前行驶位置。

该程序的界面设计文件比较简单，只是包含一个用于输入目的地的文本框和高德的 MapView，此处不再给出界面设计文件。下面是该程序的 Activity 类代码。

程序清单：codes\17\17.4\Navigation\app\src\main\java\org\crazyit\map\MainActivity.java

```
public class MainActivity extends Activity implements View.OnClickListener
    , OnGeocodeSearchListener, OnRouteSearchListener
```

```
{
    private MapView mapView;
    private AMap aMap;
    private LocationManager locMgr;
    GeocodeSearch search;
    RouteSearch routeSearch;
    private EditText targetAddressEt;
    private Button navBn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        targetAddressEt = (EditText) findViewById(R.id.address);
        mapView = (MapView) findViewById(R.id.map);
        // 必须回调 MapView 的 onCreate()方法
        mapView.onCreate(savedInstanceState);
        navBn = (Button) findViewById(R.id.nav);
        navBn.setOnClickListener(this);
        init();
        // 创建 GeocodeSearch 对象
        search = new GeocodeSearch(this);
        search.setOnGeocodeSearchListener(this);
        routeSearch = new RouteSearch(this);
        routeSearch.setRouteSearchListener(this);
        locMgr = (LocationManager) getSystemService(
            Context.LOCATION_SERVICE);
        // 通过监听器监听 GPS 提供的定位信息的改变
        locMgr.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            300, 8, new LocationListener()
            {
                @Override
                public void onLocationChanged(Location location)
                {
                    // 使用 GPS 提供的定位信息来更新位置
                    updatePosition(location);
                }
                @Override
                public void onStatusChanged(String provider
                        , int status, Bundle extras){ }
                @Override
                public void onProviderEnabled(String provider)
                {
                    // 使用 GPS 提供的定位信息来更新位置
                    updatePosition(locMgr.getLastKnownLocation(provider));
                }
                @Override
                public void onProviderDisabled(String provider) { }
            });
    }
    // 初始化 AMap 对象
    private void init() {
        if (aMap == null) {
            aMap = mapView.getMap();
            // 创建一个设置放大级别的 CameraUpdate
            CameraUpdate cu = CameraUpdateFactory.zoomTo(16);
            // 设置地图的默认放大级别
            aMap.moveCamera(cu);
        }
    }
    private void updatePosition(Location location)  // ①
    {
        LatLng pos = new LatLng(location.getLatitude(), location.getLongitude());
        // 创建一个设置经纬度的 CameraUpdate
```

```
        CameraUpdate cu = CameraUpdateFactory.changeLatLng(pos);
        // 更新地图的显示区域
        aMap.moveCamera(cu);
        // 清除所有 Marker 等覆盖物
        aMap.clear();
        MarkerOptions markerOptions = new MarkerOptions()
            .position(pos)
            // 设置使用自定义图标
            .icon(BitmapDescriptorFactory.fromResource(R.drawable.car))
            .draggable(true);
        // 添加 Marker
        aMap.addMarker(markerOptions);
    }
    @Override
    protected void onResume() {
        super.onResume();
        // 必须回调 MapView 的 onResume()方法
        mapView.onResume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        // 必须回调 MapView 的 onPause()方法
        mapView.onPause();
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        // 必须回调 MapView 的 onSaveInstanceState()方法
        mapView.onSaveInstanceState(outState);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // 必须回调 MapView 的 onDestroy()方法
        mapView.onDestroy();
    }
    @Override
    public void onClick(View v)
    {
        String address = targetAddressEt.getText().toString().trim();
        if (address.equals(""))
        {
            Toast.makeText(this, "请输入有效的地址"
                , Toast.LENGTH_LONG).show();
        }
        else
        {
            GeocodeQuery query = new GeocodeQuery(address, "广州");
            // 根据地址执行异步查询
            search.getFromLocationNameAsyn(query);
        }
    }
    @Override
    public void onRegeocodeSearched(RegeocodeResult
        regeocodeResult, int i){}
    @Override
    public void onGeocodeSearched(GeocodeResult geocodeResult, int i)  // ②
    {
        GeocodeAddress addr = geocodeResult.getGeocodeAddressList().get(0);
        // 获取目前的经纬度
        LatLonPoint latlng = addr.getLatLonPoint();
        // 获取用户当前的位置
```

```
        Location loc = locMgr.getLastKnownLocation(
            LocationManager.GPS_PROVIDER);
        // 创建路线规划的起始点
        RouteSearch.FromAndTo ft = new RouteSearch.FromAndTo(
            new LatLonPoint(loc.getLatitude(), loc.getLongitude()), latlng);
        // 创建自驾车的查询条件
        RouteSearch.DriveRouteQuery driveRouteQuery = new RouteSearch
            .DriveRouteQuery(ft // 定义道路规划的起始点
                , RouteSearch.DrivingDefault
                , null  // 该参数指定必须经过的多个点
                , null  // 该参数指定必须避开的多个区域
                , null  // 该参数指定必须避开的道路
            );
        routeSearch.calculateDriveRouteAsyn(driveRouteQuery);
    }
    @Override
    public void onBusRouteSearched(BusRouteResult
        busRouteResult, int i){}
    @Override
    public void onDriveRouteSearched(DriveRouteResult driveRouteResult, int i)  // ③
    {
        // 获取系统规划的第一条路线（在实际应用中可提供多条路线供用户选择）
        DrivePath drivePath = driveRouteResult.getPaths().get(0);  // ④
        // 获取该规划路线所包含的多条路段
        List<DriveStep> steps = drivePath.getSteps();
        for(DriveStep step: steps)
        {
            // 获取组成该路段的多个点
            List<LatLonPoint> points = step.getPolyline();
            List<LatLng> latLngs = new ArrayList<>();
            for(LatLonPoint point : points)
            {
                latLngs.add(new LatLng(point.getLatitude()
                    , point.getLongitude()));
            }
            // 创建一个 PolylineOptions（用于向地图添加多线段）
            PolylineOptions ployOptions = new PolylineOptions()
                // 添加多个点
                .addAll(latLngs)
                .color(0xffff0000)
                .width(8);
            aMap.addPolyline(ployOptions);
        }
    }
    @Override
    public void onWalkRouteSearched(WalkRouteResult
        walkRouteResult, int i){}
}
```

上面程序中的第一段粗体字代码分别创建了 GeocodeSearch 和 RouteSearch 对象，并为这两个对象设置了查询监听器。其中 GeocodeSearch 用于对用户输入的目的地执行地址解析；而 RouteSearch 则用于查询、规划行车路线。

当该 Activity 加载完成后，程序将会通过 LocationManager 的方法不间断地获取 GPS 信息，获取 GPS 信息之后，程序调用①号方法将用户的实时位置绘制在地图上。

当用户按下应用界面上的“规划路线”按钮时，程序将会通过 GeocodeSearch 的 getFromLocationNameAsyn()方法执行异步地址解析，因此当解析完成后程序将会自动激发②号方法，该方法中先获取了地址解析所获得的经度、纬度，然后创建了一个 DriveRouteQuery 对象（代表驾车路线查询），最后调用 RouteSearch 的方法执行异步查询即可，如该方法中的粗

体字代码所示。

但 RouteSearch 查询、规划路线完成时，程序将会自动激发③号方法，通过该方法的参数即可获取 RouteSearch 查询返回的多条规划路线。但此处为了简化程序，直接使用了第一条规划路线，然后程序通过 AMap 的 addPolyline()方法将规划路线底层封装的多条线段绘制在地图上，这样就可向用户显示规划路线了。

运行该程序，在“目标地”文本框中输入广州的某个地名，按下“规划路线”按钮即可看到如图 17.12 所示效果。

图 17.12 GPS 导航

注意：

运行该程序时一定要打开模拟器的 GPS 支持；否则程序将会因为无法获取 GPS 信号而出错。

17.5 本章小结

本章主要介绍了在 Android 系统中调用第三方 Map 的方法。学习本章需要掌握在 Android 应用中整合第三方各种服务的方法，包括如何获取第三方服务的 API Key、添加第三方服务的 SDK 等。除此之外，读者需要重点掌握根据 GPS 信息在地图上定位、跟踪的方法。不仅如此，如果结合了 Android 的地址解析服务，Android 应用还可以根据地址信息在地图上定位，这也是读者需要掌握的内容。本章也介绍了高德查询服务提供的导航支持。

CHAPTER

18

第18章 合金弹头

本章要点

- 开发射击类游戏的基本方法
- 游戏的界面分解和分析
- 游戏界面组件的分析和实现
- 怪物的移动和发射子弹
- 实现角色移动、跳跃、发射子弹等行为
- 检测子弹是否命中目标
- 实现游戏的绘图工具类
- 管理游戏资源
- 继承 SurfaceView 实现游戏主组件
- 掌握 SurfaceView 的绘图机制
- 使用多线程实现游戏动画
- 实现游戏的 Activity

本章将会介绍一款经典的射击类游戏：合金弹头，合金弹头游戏要求玩家控制自己的角色不断前行，并发射子弹去射击沿途遇到的各种怪物，同时还要躲避怪物发射的子弹。当然，由于完整的合金弹头游戏涉及的地图场景很多，而且怪物种类也很多，因此本章对该游戏进行了适当的简化。本游戏只实现了一个地图场景，并将之设置为无限地图，并且只实现了3种类型的怪物，但只要读者真正掌握了本章的内容，当然也就能从一个地图扩展为多个地图；也可以从3种类型的怪物扩展出多种类型的怪物了。

对于Android学习者来说，学习开发这个小程序难度适中，而且能很好地培养学习者的学习乐趣。开发者需要从程序员的角度来看待玩家面对的游戏界面，游戏界面上的每个怪物、每个角色、每颗子弹、每个能与玩家交互的东西，都应该在程序中通过类的形式来定义它们，这样才能更好地用面向对象的方式来解决问题。

18.1 合金弹头游戏简介

合金弹头是一款早期风靡一时的射击类游戏，这款游戏的节奏感非常强，让大部分男同胞充满童年回忆。实际上，现在也非常流行将一些早期游戏移植到手机平台上，以充分满足广大玩家的怀旧情怀。

图18.1显示了合金弹头的游戏界面。

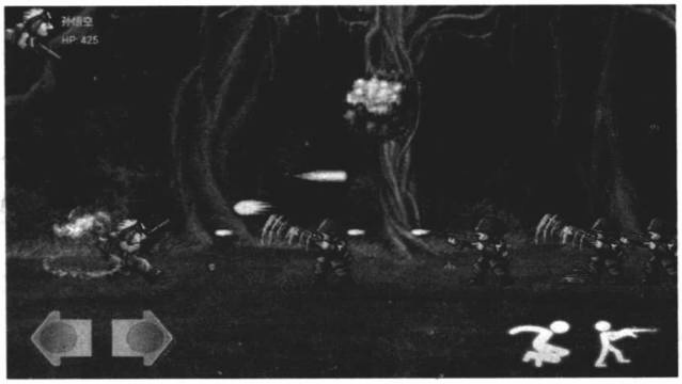


图18.1 合金弹头

这款游戏的玩法很简单，玩家控制角色不断地向右前进，角色可通过跳跃来躲避敌人（也可统称为怪物）发射的子弹和地上的炸弹，玩家也可控制角色发射子弹来打死右边的各种敌人。对于完整的合金弹头游戏，它会包含很多“关卡”，每个关卡都是一种地图，每个关卡都包含了大量不同的怪物。但本章由于篇幅关系，只做了一种地图，而且这种地图是“无限循环”的——也就是说，玩家只能一直向前去消灭不同的怪物，无法实现“通关”。

提示：

如果读者有兴趣把这款游戏改成包含很多关卡的游戏，那就需要准备大量的背景图片。然后为不同的地图加载不同的背景图片，让地图不要无限循环即可，并为不同地图使用不同的怪物。

18.2 开发游戏界面组件

在开发游戏之前，首先需要从程序员的角度来分析游戏界面，并逐步实现游戏界面上的各种组件。

18.2.1 游戏界面分析

对于图18.1所示的游戏界面，从普通玩家的角度来看，他会看到游戏界面上有受玩家控制移动、跳跃、发射子弹的角色，还有不断发射子弹的敌人、地上有炸弹、天空中有正在爆炸的飞机……乍看上去会给人眼花缭乱的感觉。

如果从程序员的角度来看，游戏界面大致可分为如下组件。

- **游戏背景**：只是一张静止的图片。
- **角色**：该角色可以站立、走动、跳跃、射击。
- **怪物**：怪物类代表了游戏界面上所有的敌人，包括拿枪的敌人、地上的炸弹、天空中的飞机……虽然这些怪物的图片不同、发射的子弹不同，攻击力也可能不同，但这些只是实例与实例之间的差异，因此程序只要为怪物定义一个类即可。
- **子弹**：不管是角色发射的子弹还是怪物发射的子弹，都可归纳为子弹类。虽然不同子弹的图片不同，攻击力不同，但这些只是实例与实例之间的差异，因此程序只要为子弹定义一个类即可。

从上面介绍不难看出，开发这款游戏，主要就是实现上面的角色、怪物和子弹3个类。

18.2.2 实现“怪物”类

由于不同怪物之间会存在各种差异，那么此处就需要为怪物类定义相应的实例变量来记录这些差异。不同怪物之间可能存在如下差异。

- 怪物的类型。
- 代表怪物位置的*X*、*Y*坐标。
- 标识怪物是否已经死亡的旗标。
- 绘制怪物图片左上角的*X*、*Y*坐标。
- 绘制怪物图片右下角的*X*、*Y*坐标。
- 怪物发射的所有子弹。（有的怪物不会发射子弹）
- 怪物未死亡时所有的动画帧图片和怪物死亡时所有的动画帧图片。

提示：

本程序并未把怪物的所有动画帧图片直接保存在怪物实例中，本程序将会专门使用一个工具类来保存所有角色、怪物的所有动画帧图片。

为了让游戏界面的角色、怪物都能“动起来”，程序的实现思路是这样的：程序会专门启动一条独立的线程，这条线程负责控制角色、怪物不断地更换新的动画帧图片——因此程序需要为怪物增加一个成员变量来记录当前游戏界面正在绘制怪物动画的第几帧，而负责动画的独立线程只要不断地调用怪物的绘制方法即可——实际上该绘制方法每次只是绘制一张静态图片（这张静态图片是怪物动画的其中一帧）。

下面是怪物类的成员变量部分。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 定义代表怪物类型的常量（如果程序还需要增加更多怪物，只需在此处添加常量即可）
public static final int TYPE_BOMB = 1;
public static final int TYPE_FLY = 2;
public static final int TYPE_MAN = 3;
// 定义怪物类型的成员变量
private int type = TYPE_BOMB;
// 定义怪物X、Y坐标的成员变量
private int x = 0;
private int y = 0;
// 定义怪物是否已经死亡的旗标
private boolean isDie = false;
// 绘制怪物图片左上角的X坐标
private int startX = 0;
```

```
// 绘制怪物图片左上角的 Y 坐标
private int startY = 0;
// 绘制怪物图片右下角的 X 坐标
private int endX = 0;
// 绘制怪物图片右下角的 Y 坐标
private int endY = 0;
// 该变量用于控制动画刷新的速度
int drawCount = 0;
// 定义当前正在绘制怪物动画的第几帧的变量
private int drawIndex = 0;
// 用于记录死亡动画只绘制一次，不需要重复绘制
// 每当怪物死亡时，该变量会被初始化为等于死亡动画的总帧数
// 当怪物的死亡动画帧播放完成时，该变量的值变为 0
private int dieMaxDrawCount = Integer.MAX_VALUE;
// 定义怪物射出的子弹
private List<Bullet> bulletList = new ArrayList<>();
```

上面的成员变量即可记录该怪物实例的各种状态。实际上以后程序要升级，比如为怪物增加更多的特征，如怪物可以拿不同的武器，怪物可以穿不同的衣服，怪物可以具有不同的攻击力……这些都可考虑定义成怪物的成员变量。

下面是怪物类的构造器，该构造器只要传入一个 type 参数即可，该 type 参数告诉系统，该怪物是哪种类型的怪物。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
public Monster(int type)
{
    this.type = type;
    // -------下面代码根据怪物类型来初始化怪物 X、Y 坐标------
    // 如果怪物是炸弹（TYPE_BOMB）或敌人（TYPE_MAN）
    // 怪物的 Y 坐标与玩家控制的角色的 Y 坐标相同
    if (type == TYPE_BOMB || type == TYPE_MAN)
    {
        y = Player.Y_DEFALUT;
    }
    // 如果怪物是飞机，根据屏幕高度随机生成怪物的 Y 坐标
    else if (type == TYPE_FLY)
    {
        y = ViewManager.SCREEN_HEIGHT * 50 / 100
            - Util.rand((int) (ViewManager.scale * 100));
    }
    // 随机计算怪物的 X 坐标。
    x = ViewManager.SCREEN_WIDTH + Util.rand(ViewManager.SCREEN_WIDTH >> 1)
        - (ViewManager.SCREEN_WIDTH >> 2);
}
```

从上面的粗体字代码可以看出，程序在创建怪物实例时，不仅负责初始化怪物的 type 成员变量，还会根据怪物类型来设置怪物的 *X*、*Y* 坐标。

- 如果怪物是炸弹和拿枪的人（都在地面上），那么它们的 *Y* 坐标与角色默认的 *Y* 坐标（在地面上）相同。如果怪物是飞机，那么怪物的 *Y* 坐标是随机计算的。
- 不管什么怪物，它的 *X* 坐标都是随机计算的。

上面程序中还用到一个 Util 工具类，该工具类仅仅包含一个计算随机数的方法。下面是该工具类的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Util.java

```
public class Util
{
```

```
    public static Random random = new Random();
    // 返回一个0～range的随机数
    public static int rand(int range)
    {
        // 如果range为0，直接返回0
        if (range == 0)
            return 0;
        // 获取一个0～range之间的随机数
        return Math.abs(random.nextInt() % range);
    }
}
```

前面已经介绍了绘制怪物动画的思路：程序将会采用后台线程来控制不断地绘制怪物动画的下一帧，但实际上每次绘制的只是怪物动画的某一帧。下面是绘制怪物的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 绘制怪物的方法
public void draw(Canvas canvas)
{
    if (canvas == null)
    {
        return;
    }
    switch (type)
    {
        case TYPE_BOMB:
            // 死亡的怪物用死亡图片
            drawAni(canvas, isDie ? ViewManager.bomb2Image : ViewManager.bombImage);
            break;
        case TYPE_FLY:
            // 死亡的怪物用死亡图片
            drawAni(canvas, isDie ? ViewManager.flyDieImage : ViewManager.flyImage);
            break;
        case TYPE_MAN:
            // 死亡的怪物用死亡图片
            drawAni(canvas, isDie ? ViewManager.manDieImage : ViewManager.manImgae);
            break;
        default:
            break;
    }
}
// 根据怪物的动画帧图片来绘制怪物动画
public void drawAni(Canvas canvas, Bitmap[] bitmapArr)
{
    if (canvas == null)
    {
        return;
    }
    if (bitmapArr == null)
    {
        return;
    }
    // 如果怪物已经死亡，且没有播放过死亡动画
    // (dieMaxDrawCount等于初始值表明未播放过死亡动画)
    if (isDie && dieMaxDrawCount == Integer.MAX_VALUE)
    {
        // 将dieMaxDrawCount设置为与死亡动画的总帧数相等
        dieMaxDrawCount = bitmapArr.length;  // ⑤
    }
    drawIndex = drawIndex % bitmapArr.length;
    // 获取当前绘制的动画帧对应的位图
    Bitmap bitmap = bitmapArr[drawIndex];  // ①
```

```
        if (bitmap == null || bitmap.isRecycled())
        {
            return;
        }
        int drawX = x;
        // 对绘制怪物动画帧位图的 X 坐标进行微调
        if (isDie)
        {
            if (type == TYPE_BOMB)
            {
                drawX = x - (int) (ViewManager.scale * 50);
            }
            else if (type == TYPE_MAN)
            {
                drawX = x + (int) (ViewManager.scale * 50);
            }
        }
        // 对绘制怪物动画帧位图的 Y 坐标进行微调
        int drawY = y - bitmap.getHeight();
        // 绘制怪物动画帧的位图
        Graphics.drawMatrixImage(canvas, bitmap, 0, 0, bitmap.getWidth(),
            bitmap.getHeight(), Graphics.TRANS_NONE, drawX, drawY, 0,
                Graphics.TIMES_SCALE);
        startX = drawX;
        startY = drawY;
        endX = startX + bitmap.getWidth();
        endY = startY + bitmap.getHeight();
        drawCount++;
        // 后面的 6、4 用于控制人、飞机发射子弹的速度
        if (drawCount >= (type == TYPE_MAN ? 6 : 4))  // ③
        {
            // 如果怪物是人，只在第 3 帧才发射子弹
            if (type == TYPE_MAN && drawIndex == 2)
            {
                addBullet();
            }
            // 如果怪物是飞机，只在最后一帧才发射子弹
            if (type == TYPE_FLY && drawIndex == bitmapArr.length - 1)
            {
                addBullet();
            }
            drawIndex++;  // ②
            drawCount = 0;  // ④
        }
        // 每播放死亡动画的一帧，dieMaxDrawCount 减 1
        // 当 dieMaxDrawCount 等于 0 时，表明死亡动画播放完成，MonsterManger 会删除该怪物
        if (isDie)
        {
            dieMaxDrawCount--;  // ⑥
        }
        // 绘制子弹
        drawBullet(canvas);
    }
```

上面代码包含两个方法，其中 draw(Canvas canvas)方法只是简单地对怪物类型进行了判断，并针对不同怪物类型使用不同的怪物动画。

draw(Canvas canvas)方法总是调用 drawAni(Canvas canvas, Bitmap[] bitmapArr)方法来绘制怪物，调用后者时根据怪物类型的不同、怪物是否死亡将会传入不同的位图数组——每个位图数组就代表一组动画帧的所有位图。

drawAni()方法中的①号粗体字代码就是根据 drawIndex 来获取当前帧对应的位图，而程序

执行 drawAni()方法时，②号粗体字代码可以控制 drawIndex 自加一次，这样即可保证下次调用 drawAni()方法时就会绘制动画的下一帧。

drawAni()方法还涉及一个 drawCount 变量，这个变量是控制动画刷新速度的计数器——程序在③号粗体字代码处进行了控制：只有当 drawCount 计数器的值大于 6（对于其他类型的怪物，该值为 4）时才会调用 drawIndex++，这意味着当怪物类型是 TYPE_MAN 时，drawAni()方法至少调用 6 次之后才会将 drawIndex 加 1（即绘制下一帧位图）；当怪物是其他类型时，drawAni()方法至少调用 4 次之后才会将 drawIndex 加 1（即绘制下一帧位图）——这是因为程序中控制动画刷新的线程的刷新频率是固定的，即如后台线程控制每隔 40ms 调用一次怪物的 drawAni()方法，但如果每隔 40ms 就更新一次动画帧的话，那么游戏界面上的所有怪物“动”的速度都是一样的（每隔 40ms 刷新一次），而且它们都动得非常快。为了解决这个问题，程序就需要使用 drawCount 来控制不同怪物实际每隔多少毫秒更新一次动画帧。对于上面代码来说，如果怪物类型是 TYPE_MAN，则只有当 drawCount 计数器大于 6 时，才会更新一次动画帧，这意味着实际上每隔 240ms 才会更新一次动画帧；如果是其他类型的怪物，那么只有当 drawCount 计数器大于 4 时，才会更新一次动画帧，这意味着实际上每隔 160ms 才会更新一次动画帧。

提示：

如果游戏中还有更多类型的怪物，且这些怪物的动画帧具有不同的更新速度，那么程序还需要进行更细致的判断。

drawAni()方法还涉及一个 dieMaxDrawCount 变量，这个变量用于控制怪物的死亡动画只会被绘制一次——在怪物临死之前，程序都必须播放怪物的死亡动画，该动画播放完成，该怪物就应该从地图上删除。当怪物已经死亡（isDie 为真）且还未绘制死亡动画的任何帧时（dieMaxDrawCount 等于初始值），程序在⑤号粗体字代码处将 dieMaxDrawCount 设置为与死亡动画的总帧数相等，程序每次调用 drawAni()方法时，⑥号粗体字代码都会把 dieMaxDrawCount 减 1，当 dieMaxDrawCount 变为 0 时，表明该怪物的死亡动画的所有帧都绘制完成，接下来程序即可将该怪物从地图上删除了——在后面的 MonsterManager 类中将会看到程序根据怪物的 dieMaxDrawCount 为 0 来从地图上删除怪物的代码。

Monster 还包含了 startX、startY、endX、endY 四个变量，这四个变量就代表了怪物当前帧所覆盖的矩形区域，因此，如果程序需要判断该怪物是否被子弹打中，只要子弹出现在该矩形区域内，即可判断怪物被子弹打中了。下面是判断怪物是否被子弹打中的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 判断怪物是否被子弹打中的方法
public boolean isHurt(int x, int y)
{
    return x >= startX && x <= endX
        && y >= startY && y <= endY;
}
```

接下来为怪物实现发射子弹的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 根据怪物类型获取子弹类型，不同怪物发射不同的子弹
// return 0 代表这种怪物不发射子弹
public int getBulletType()
{
    switch (type)
```

```
        {
            case TYPE_BOMB:
                return 0;
            case TYPE_FLY:
                return Bullet.BULLET_TYPE_3;
            case TYPE_MAN:
                return Bullet.BULLET_TYPE_2;
            default:
                return 0;
        }
    }
    // 定义发射子弹的方法
    public void addBullet()
    {
        int bulletType = getBulletType();
        // 如果没有子弹
        if (bulletType <= 0)
        {
            return;
        }
        // 计算子弹的X、Y坐标
        int drawX = x;
        int drawY = y - (int) (ViewManager.scale * 60);
        // 如果怪物是飞机，重新计算飞机发射的子弹的Y坐标
        if (type == TYPE_FLY)
        {
            drawY = y - (int) (ViewManager.scale * 30);
        }
        // 创建子弹对象
        Bullet bullet = new Bullet(bulletType, drawX, drawY, Player.DIR_LEFT);
        // 将子弹添加到该怪物发射的子弹集合中
        bulletList.add(bullet);
    }
```

怪物发射子弹的方法是 addBullet()，该方法需要调用 getBulletType()方法来判断该怪物所发射的子弹类型（不同怪物可能需要发射不同的子弹），如果 getBulletType()方法返回 0，即代表这种怪物不发射子弹。

一旦确定了这种怪物发射子弹的类型，程序就可根据不同怪物计算子弹的初始 *X*、*Y* 坐标——基本上，子弹的 *X*、*Y* 坐标保持与怪物当前的 *X*、*Y* 坐标相同，再进行适当微调即可。程序最后两行粗体字代码创建了一个 Bullet 对象（子弹实例），并将新的 Bullet 对象添加到 bulletList 集合中。

当怪物发射了子弹之后，程序还需要绘制该怪物的所有子弹。下面是绘制怪物发射的所有子弹的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
    // 更新角色的位置：将角色的X坐标减少shift距离（角色左移）
    // 更新所有子弹的位置：将所有子弹的X坐标减少shift距离（子弹左移）
    public void updateShift(int shift)
    {
        x -= shift;
        for (Bullet bullet : bulletList)
        {
            if (bullet == null)
            {
                continue;
            }
            bullet.setX(bullet.getX() - shift);
        }
    }
```

```
    // 绘制子弹的方法
    public void drawBullet(Canvas canvas)
    {
        // 定义一个 deleteList 集合，该集合保存所有需要删除的子弹
        List<Bullet> deleteList = new ArrayList<>();
        Bullet bullet = null;
        for (int i = 0; i < bulletList.size(); i++)
        {
            bullet = bulletList.get(i);
            if (bullet == null)
            {
                continue;
            }
            // 如果子弹已经越过屏幕
            if (bullet.getX() < 0 || bullet.getX() > ViewManager.SCREEN_WIDTH)
            {
                // 将需要清除的子弹添加到 deleteList 集合中
                deleteList.add(bullet);
            }
        }
        // 删除所有需要清除的子弹
        bulletList.removeAll(deleteList);  // ⑦
        // 定义代表子弹的位图
        Bitmap bitmap;
        // 遍历该怪物发射的所有子弹
        for (int i = 0; i < bulletList.size(); i++)
        {
            bullet = bulletList.get(i);
            if (bullet == null)
            {
                continue;
            }
            // 获取子弹对应的位图
            bitmap = bullet.getBitmap();
            if (bitmap == null)
            {
                continue;
            }
            // 子弹移动
            bullet.move();
            // 绘制子弹的位图
            Graphics.drawMatrixImage(canvas, bitmap, 0, 0, bitmap.getWidth(),
                bitmap.getHeight(), bullet.getDir() == Player.DIR_RIGHT ?
                Graphics.TRANS_MIRROR : Graphics.TRANS_NONE,
                bullet.getX(), bullet.getY(), 0, Graphics.TIMES_SCALE);
        }
    }
```

上面程序中的 updateShift(int shift)方法负责将怪物所有的子弹全部左移 shift 距离，这是因为界面上角色会不断地向右移动，角色会产生一个 shift 偏移，所以程序就需要将怪物（包括它的所有子弹）全部左移 shift 距离，这样才会产生逼真的效果。

上面程序中的粗体字代码使用 deleteList 集合收集所有越过屏幕的子弹，然后⑦号粗体字代码负责删除 deleteList 集合包含的所有子弹——这样即可把所有越过屏幕的子弹删除掉。

接下来程序采用循环遍历了该怪物发射的所有子弹，先获取子弹对应的位图，然后调用子弹的 move()方法控制子弹移动。上面方法中的最后一行粗体字代码负责绘制子弹位图。

Monster 类还需要定义一个方法，用于判断怪物的子弹是否打中角色，如果打中角色，则删除该子弹。下面是该方法的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 判断子弹是否与玩家控制的角色碰撞（判断子弹是否打中角色）
public void checkBullet()
{
    // 定义一个 delBulletList 集合，该集合保存打中角色的子弹，它们将要被删除
    List<Bullet> delBulletList = new ArrayList<>();
    // 遍历所有子弹
    for (Bullet bullet : bulletList)
    {
        if (bullet == null || !bullet.isEffect())
        {
            continue;
        }
        // 如果玩家控制的角色被子弹打到
        if (GameView.player.isHurt(bullet.getX(), bullet.getX()
            , bullet.getY(), bullet.getY()))
        {
            // 子弹设为无效
            bullet.setEffect(false);
            // 将玩家的生命值减 5
            GameView.player.setHp(GameView.player.getHp() - 5);
            // 将子弹添加到 delBulletList 集合中
            delBulletList.add(bullet);
        }
    }
    // 删除所有打中角色的子弹
    bulletList.removeAll(delBulletList);
}
```

18.2.3 实现怪物管理类

由于游戏界面上会出现很多个怪物，因此程序需要额外定义一个怪物管理类来专门负责管理怪物的随机产生、死亡等行为。

为了有效地管理游戏界面上所有活着的怪物和已死的怪物（保存已死的怪物是为了绘制死亡动画），为怪物管理类定义如下成员变量。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 保存所有死掉的怪物，保存它们是为了绘制死亡动画，绘制完后清除这些怪物
public static final List<Monster> dieMonsterList = new ArrayList<>();
// 保存所有活着的怪物
public static final List<Monster> monsterList = new ArrayList<>();
```

接下来在怪物管理类中定义一个随机生成怪物的工具方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 随机生成并添加怪物的方法
public static void generateMonster()
{
    if (monsterList.size() < 3 + Util.rand(3))
    {
        // 创建新怪物
        Monster monster = new Monster(1 + Util.rand(3));
        monsterList.add(monster);
    }
}
```

前面已经指出，当玩家控制游戏界面的角色不断地向右移动时，程序界面上的所有怪物、怪物的子弹都必须不断地左移，因此程序需要在 MonsterManager 类中定义一个控制所有怪物

及其子弹不断左移的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 更新怪物与子弹的坐标的方法
public static void updatePosistion(int shift)
{
    Monster monster = null;
    // 定义一个集合，保存所有将要被删除的怪物
    List<Monster> delList = new ArrayList<>();
    // 遍历怪物集合
    for (int i = 0; i < monsterList.size(); i++)
    {
        monster = monsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 更新怪物、怪物所有子弹的位置
        monster.updateShift(shift);  // ①
        // 如果怪物的 X 坐标越界，将怪物添加到 delList 集合中
        if (monster.getX() < 0)
        {
            delList.add(monster);
        }
    }
    // 删除 delList 集合中的所有怪物
    monsterList.removeAll(delList);
    delList.clear();
    // 遍历所有已死的怪物的集合
    for (int i = 0; i < dieMonsterList.size(); i++)
    {
        monster = dieMonsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 更新怪物、怪物所有子弹的位置
        monster.updateShift(shift);  // ②
        // 如果怪物的 X 坐标越界，将怪物添加到 delList 集合中
        if (monster.getX() < 0)
        {
            delList.add(monster);
        }
    }
    // 删除 delList 集合中的所有怪物
    dieMonsterList.removeAll(delList);
    // 更新玩家控制的角色的子弹坐标
    GameView.player.updateBulletShift(shift);
}
```

上面程序中的①号粗体字代码处于循环体之内，该循环将会控制把所有活着的怪物及其子弹全部都左移 shift 距离，如果移动之后的怪物的 *X* 坐标超出了屏幕范围，程序就会清除该怪物；②号粗体字代码同样处于循环体之内，②号代码的处理方式与①号代码的处理方式几乎是一样的，只是②号代码负责处理的是界面上已死的怪物。

上面程序中的最后一行粗体字代码则负责将玩家发射的所有子弹都左移 shift 距离——这也是必要的，原因与怪物发射的子弹都需要左移 shift 距离一样。

接下来要为 MonsterManager 实现一个新的方法，该方法可用于检查界面上的怪物是否将要死亡，将要死亡的怪物将会从 monsterList 集合中删除，并添加到 dieMonsterList 集合中，然

后程序将会负责绘制它们的死亡动画。

下面为 MonsterManager 类增加一个 checkMonster()方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 检查怪物是否将要死亡的方法
public static void checkMonster()
{
    // 获取玩家发射的所有子弹
    List<Bullet> bulletList = GameView.player.getBulletList();
    if (bulletList == null)
    {
        bulletList = new ArrayList<>();
    }
    Monster monster = null;
    // 定义一个 delList 集合，用于保存将要死亡的怪物
    List<Monster> delList = new ArrayList<>();
    // 定义一个 delBulletList 集合，用于保存所有将要被删除的子弹
    List<Bullet> delBulletList = new ArrayList<>();
    // 遍历所有怪物
    for (int i = 0; i < monsterList.size(); i++)
    {
        monster = monsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 如果怪物是炸弹
        if (monster.getType() == Monster.TYPE_BOMB)
        {
            // 角色被炸弹炸到
            if (GameView.player.isHurt(monster.getStartX()
                , monster.getEndX(), monster.getStartY(), monster.getEndY()))
            {
                // 将怪物设置为死亡状态
                monster.setDie(true);
                // 播放爆炸音效
                ViewManager.soundPool.play(
                    ViewManager.soundMap.get(2), 1, 1, 0, 0, 1);
                // 将怪物（爆炸的炸弹）添加到 delList 集合中
                delList.add(monster);
                // 玩家控制的角色的生命值减 10
                GameView.player.setHp(GameView.player.getHp() - 10);
            }
            continue;
        }
        // 对于其他类型的怪物，则需要遍历角色发射的所有子弹
        // 只要任何一个子弹打中怪物，即可判断怪物即将死亡
        for (Bullet bullet : bulletList)
        {
            if (bullet == null || !bullet.isEffect())
            {
                continue;
            }
            // 如果怪物被角色的子弹打到
            if (monster.isHurt(bullet.getX(), bullet.getY()))
            {
                // 将子弹设为无效
                bullet.setEffect(false);
                // 将怪物设为死亡状态
                monster.setDie(true);
                // 如果怪物是飞机
```

```
                if(monster.getType() == Monster.TYPE_FLY)
                {
                    // 播放爆炸音效
                    ViewManager.soundPool.play(
                        ViewManager.soundMap.get(2), 1, 1, 0, 0, 1);
                }
                // 如果怪物是人
                if(monster.getType() == Monster.TYPE_MAN)
                {
                    // 播放惨叫音效
                    ViewManager.soundPool.play(
                        ViewManager.soundMap.get(3), 1, 1, 0, 0, 1);
                }
                // 将怪物（被子弹打中的怪物）添加到 delList 集合中
                delList.add(monster);
                // 将打中怪物的子弹添加到 delBulletList 集合中
                delBulletList.add(bullet);
            }
        }
        // 将 delBulletList 包含的所有子弹从 bulletList 集合中删除
        bulletList.removeAll(delBulletList);
        // 检查怪物子弹是否打到角色
        monster.checkBullet();
    }
    // 将已死亡的怪物（保存在 delList 集合中）添加到 dieMonsterList 集合中
    dieMonsterList.addAll(delList);
    // 将已死亡的怪物（保存在 delList 集合中）从 monsterList 集合中删除
    monsterList.removeAll(delList);
}
```

上面这个方法的判断逻辑非常简单，程序把怪物分为两类进行处理。

- 如果怪物是地上的炸弹，只要炸弹炸到角色，炸弹也就即将死亡。上面程序中第一行粗体字代码处理了怪物是炸弹的情形。
- 对于其他类型的怪物，程序则需要遍历角色发射的子弹，只要任意一颗子弹打中了怪物，即可判断怪物即将死亡。上面程序中第二行粗体字代码正是遍历玩家所发射的子弹的代码。

最后 MonsterManager 还需要定义一个绘制所有怪物的方法。该方法的实现逻辑也非常简单，程序只要分别遍历该类的 dieMonsterList 和 monsterList 集合，并将集合中所有怪物绘制出来即可。对于 dieMonsterList 集合中的怪物，它们都是将要死亡的怪物，因此只要将它们所有的死亡动画帧都绘制一次，接下来就应该清除这些怪物了——Monster 实例的 dieMaxDrawCount 成员变量为 0 时就代表所有死亡动画帧都绘制了一次。

下面是该 drawMonster()方法的代码，该方法就负责绘制所有怪物。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 绘制所有怪物的方法
public static void drawMonster(Canvas canvas)
{
    Monster monster = null;
    // 遍历所有活着的怪物，绘制活着的怪物
    for (int i = 0; i < monsterList.size(); i++)
    {
        monster = monsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 绘制怪物
```

```
            monster.draw(canvas);
        }
        List<Monster> delList = new ArrayList<>();
        // 遍历所有已死亡的怪物，绘制已死亡的怪物
        for (int i = 0; i < dieMonsterList.size(); i++)
        {
            monster = dieMonsterList.get(i);
            if (monster == null)
            {
                continue;
            }
            // 绘制怪物
            monster.draw(canvas);
            // 当怪物的 getDieMaxDrawCount()返回 0 时，则表明该怪物已经死亡
            // 且该怪物的死亡动画所有帧都播放完成，将它们彻底删除
            if (monster.getDieMaxDrawCount() <= 0)  // ③
            {
                delList.add(monster);
            }
        }
        dieMonsterList.removeAll(delList);
    }
```

上面程序中的第一行粗体字代码负责遍历所有活着的怪物，并将它们绘制出来；第二行粗体字代码则负责遍历所有将要死亡的怪物，并将它们绘制出来。程序中③号粗体字代码检查该怪物的 dieMaxDrawCount 是否为 0，如果为 0，则表明该怪物已死亡、且该怪物的死亡动画所有帧都播放完成，应该将它们彻底删除。

18.2.4 实现“子弹”类

本游戏的子弹类比较简单，本游戏中的子弹不会产生爆炸效果。对子弹的处理思路是：只要子弹打中目标，子弹就会自动消失。正因为本游戏的子弹类比较简单，因此子弹类只需要定义如下属性即可。

- 子弹的类型。
- 子类的 *X*、*Y* 坐标。
- 子弹的射击方向（向左或向右）。
- 子弹在垂直方向（*Y* 方向）上的加速度。

基于上面分析，程序为 Bullet 类定义了如下成员变量。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

```
// 定义代表子弹类型的常量（如果程序还需要增加更多子弹，只需在此处添加常量即可）
public static final int BULLET_TYPE_1 = 1;
public static final int BULLET_TYPE_2 = 2;
public static final int BULLET_TYPE_3 = 3;
public static final int BULLET_TYPE_4 = 4;
// 定义子弹的类型
private int type;
// 子弹的 X、Y 坐标
private int x;
private int y;
// 定义子弹射击的方向
private int dir;
// 定义子弹在 Y 方向上的加速度
private int yAccelate = 0;
// 子弹是否有效
private boolean isEffect = true;
```

接下来可以为 Bullet 类定义一个有参数的构造器，该构造器用于对子弹的类型，*X*、*Y* 坐标，方向执行初始化。该构造器的代码比较简单，此处不再给出。

本游戏中不同怪物、角色发射的子弹都各不相同，因此不同类型的子弹将会采用不同的位图，这样玩家就会觉得不同怪物、角色发射的子弹都各不相同了。下面是 Bullet 类根据子弹类型来获取对应位图的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

```
// 根据子弹类型获取子弹对应的图片
public Bitmap getBitmap()
{
    switch (type)
    {
        case BULLET_TYPE_1:
            return ViewManager.bulletImage[0];
        case BULLET_TYPE_2:
            return ViewManager.bulletImage[1];
        case BULLET_TYPE_3:
            return ViewManager.bulletImage[2];
        case BULLET_TYPE_4:
            return ViewManager.bulletImage[3];
        default:
            return null;
    }
}
```

从上面程序可以看出，根据子弹类型来获取对应位图的处理方式非常简单，程序只是用一个 switch 分支语句对子弹类型进行判断，然后根据不同的子弹类型取得相应的位图即可——实际上这里只是开发者的一种约定，并没有规定哪种怪物必须发射怎样的子弹，因此读者可以修改此处的代码，从而改变角色、怪物发射子弹的外观。

接下来程序还可以计算子弹在水平方向、垂直方向上的移动速度。下面是这两个方法的代码实现。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

```
// 根据子弹类型来计算子弹在 X 方向上的速度
public int getSpeedX()
{
    // 根据玩家的方向来计算子弹方向和移动方向
    int sign = dir == Player.DIR_RIGHT ? 1 : -1;
    switch (type)
    {
        // 对于第 1 种子弹，以 12 为基数来计算它的速度
        case BULLET_TYPE_1:
            return (int) (ViewManager.scale * 12) * sign;
        // 对于第 2 种子弹，以 8 为基数来计算它的速度
        case BULLET_TYPE_2:
            return (int) (ViewManager.scale * 8) * sign;
        // 对于第 3 种子弹，以 8 为基数来计算它的速度
        case BULLET_TYPE_3:
            return (int) (ViewManager.scale * 8) * sign;
        // 对于第 4 种子弹，以 8 为基数来计算它的速度
        case BULLET_TYPE_4:
            return (int) (ViewManager.scale * 8) * sign;
        default:
            return (int) (ViewManager.scale * 8) * sign;
    }
}
// 根据子弹类型来计算子弹在 Y 方向上的速度
```

```
public int getSpeedY()
{
    // 如果 yAccelate 不为 0，则以 yAccelate 作为 Y 方向上的速度
    if (yAccelate != 0)
    {
        return yAccelate;
    }
    // 此处控制只有第 3 种子弹才有 Y 方向上的速度（子弹会斜着向下移动）
    switch (type)
    {
        case BULLET_TYPE_1:
            return 0;
        case BULLET_TYPE_2:
            return 0;
        case BULLET_TYPE_3:
            return (int) (ViewManager.scale * 6);
        case BULLET_TYPE_4:
            return 0;
        default:
            return 0;
    }
}
```

从上面代码可以看出，当程序要计算子弹在 *X* 方向上的速度时，程序首先判断该子弹的射击方向是否向右，如果子弹的射击方向为向右，那么子弹在 *X* 方向上的速度为正值（保证子弹不断地向右移动）；如果子弹的射击方向为向左，那么子弹在 *X* 方向上的速度为负值（保证子弹不断地向左移动）。

接下来程序计算子弹在 *X* 方向上的速度时就非常简单了，除了第 1 种子弹以 12 为基数来计算 *X* 方向上的速度之外，其他子弹都是以 8 为基数来计算 *X* 方向上的速度的，这意味着只有第 1 种子弹的速度是最快的。

提示：

计算子弹速度时还乘以了 ViewManager.scale 缩放因子，其中 ViewManager.scale 缩放因子代表了整个界面与屏幕之间的缩放比。

计算 *Y* 方向上的速度时，程序的计算逻辑也非常简单。如果该子弹的 yAccelate 不为 0（*Y* 方向上的加速度不为 0），则直接以 yAccelate 作为子弹在 *Y* 方向上的速度，这是因为程序设定：当玩家在跳起过程中，玩家射出的子弹应该斜向上发射；当玩家在降落的过程中，玩家射出的子弹应该斜向下发射。除此之外，程序还使用 switch 语句对子弹的类型进行了判断：当子弹为第 3 种子弹时，子弹将会具有 *Y* 方向上的速度（这意味着子弹会不断地向下移动）——这是因为本程序设定飞机发射的炮弹是第 3 种子弹，这种子弹会模拟飞机投弹斜向下移动。

程序计算出了子弹在 *X* 方向、*Y* 方向上的移动速度之后，接下来计算该子弹的移动就非常简单了，用 *X* 坐标加上 *X* 方向上的速度，*Y* 坐标加上 *Y* 方向上的速度即可。下面是控制子弹移动的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

```
// 定义控制子弹移动的方法
public void move()
{
    x += getSpeedX();
    y += getSpeedY();
}
```

18.2.5 实现“角色”类

游戏的角色类（也就是受玩家控制的那个人）和怪物类其实差不多，它们具有很多相似的地方，因此它们在类实现上有很多相似之处。不过由于角色需要受玩家控制，它的动作比较多，因此程序需要额外为角色定义一个成员变量，用于记录该角色正在执行的动作，并且需要将角色的头部、腿部分开进行处理。

下面是 Player 类的成员变量。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

```
// 定义角色的最高生命值
public static final int MAX_HP = 500;
// 定义控制角色动作的常量
// 此处控制该角色只包含站立、跑、跳等动作
public static final int ACTION_STAND_RIGHT = 1;
public static final int ACTION_STAND_LEFT = 2;
public static final int ACTION_RUN_RIGHT = 3;
public static final int ACTION_RUN_LEFT = 4;
public static final int ACTION_JUMP_RIGHT = 5;
public static final int ACTION_JUMP_LEFT = 6;
// 定义角色向右移动的常量
public static final int DIR_RIGHT = 1;
// 定义角色向左移动的常量
public static final int DIR_LEFT = 2;
// 控制角色的默认坐标
public static int X_DEFAULT = 0;
public static int Y_DEFALUT = 0;
public static int Y_JUMP_MAX = 0;
// 保存角色名字的成员变量
private String name;
// 保存角色生命值的成员变量
private int hp;
// 保存角色所使用枪的类型（以后可考虑让角色能更换不同的枪）
private int gun;
// 保存角色当前动作的成员变量（默认向右站立）
private int action = ACTION_STAND_RIGHT;
// 代表角色 X 坐标的成员变量
private int x = -1;
 // 代表角色 Y 坐标的成员变量
private int y = -1;
// 保存角色射出的所有子弹
private final List<Bullet> bulletList = new ArrayList<>();
// 定义控制角色移动的常量
// 此处控制该角色只包含站立、向右移动、向左移动三种移动方式
public static final int MOVE_STAND = 0;
public static final int MOVE_RIGHT = 1;
public static final int MOVE_LEFT = 2;
// 保存角色移动方式的成员变量
public int move = MOVE_STAND;
public static final int MAX_LEFT_SHOOT_TIME = 6;
// 控制射击状态的保留计数器
// 每当用户发射一枪时，leftShootTime 会被设为 MAX_LEFT_SHOOT_TIME
// 只有当 leftShootTime 变为 0 时，用户才能发射下一枪
private int leftShootTime = 0;
// 保存角色是否跳动的成员变量
public boolean isJump = false;
// 保存角色是否跳到最高处的成员变量
public boolean isJumpMax = false;
// 控制跳到最高处的停留时间
```

```
public int jumpStopCount = 0;
// 当前正在绘制角色腿部动画的第几帧
private int indexLeg = 0;
// 当前正在绘制角色头部动画的第几帧
private int indexHead = 0;
// 当前绘制头部图片的 X 坐标
private int currentHeadDrawX = 0;
// 当前绘制头部图片的 Y 坐标
private int currentHeadDrawY = 0;
// 当前正在绘制的腿部动画帧的图片
private Bitmap currentLegBitmap = null;
 // 当前正在绘制的头部动画帧的图片
private Bitmap currentHeadBitmap = null;
// 该变量用于控制动画刷新的速度
private int drawCount = 0;
```

上面程序中的粗体字代码成员变量正是角色类与怪物类的差别所在，由于角色有名字、生命值（hp）、动作、移动方式这些特殊的状态，因此程序为角色定义了 name、hp、action、move 这些成员变量。

上面程序还为 Player 类定义了一个 leftShootTime 变量，该变量的作用有两个。

- 当角色的 leftShootTime 不为 0 时，表明角色当前正处于射击状态，因此此时角色的头部动画必须使用射击的动画帧。
- 当角色的 leftShootTime 不为 0 时，表明角色当前正处于射击状态，因此角色不能立即发射下一枪——必须等到 leftShootTime 为 0 时，角色才能发射下一枪。这意味着：即使用户按下界面上的“射击”按钮，也必须等到角色上一枪发射完成才会发射下一枪。

上面程序中的最后 6 行粗体字代码是绘制角色位图相关的成员变量，从这些成员变量可以看出，程序把角色按头部、腿部分开处理，因此程序需要为头部、腿部分开定义相应的成员变量。

Player 类的构造器、初始化方法与 Monster 类的大致相似，此处不再介绍。

为了计算角色的方向（程序需要根据角色的方向来绘制角色），程序为 Player 提供了如下方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

```
// 获取该角色当前方向：action 成员变量为奇数代表向右
public int getDir()
{
    if (action % 2 == 1)
    {
        return DIR_RIGHT;
    }
    return DIR_LEFT;
}
```

从上面代码可以看出，程序可根据角色的 action 来计算角色的方向，只要 action 变量的值为奇数，即可判断该角色的方向为向右。

在介绍 Monster 类时已经提出，为了更好地在屏幕上绘制 Monster 对象以及所有子弹，程序需要根据角色在游戏界面上的位移来进行偏移，因此程序需要为 Player 方法来计算角色在游戏界面上的位移。下面是 Player 类中计算位移的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

```
// 返回该角色在游戏界面上的位移
public int getShift()
{
```

```
        if (x <= 0 || y <= 0)
        {
            initPosition();
        }
        return X_DEFAULT - x;
    }
```

从上面粗体字代码可以看出，程序计算角色位移的方法非常简单，只要用角色的初始 *X* 坐标减去角色当前 *X* 坐标即可。

游戏绘制角色、绘制角色动画的方法，与绘制怪物、绘制怪物动画的方法基本相似，只是程序需要分开绘制角色头部、腿部，读者可参考光盘代码来理解绘制角色、绘制角色动画的方法。

为了在游戏界面左上角绘制角色的名字、头像、生命值，Player 类提供了如下方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

```
    // 绘制左上角的角色头像、名字、生命值的方法
    public void drawHead(Canvas canvas)
    {
        if (ViewManager.head == null)
        {
            return;
        }
        // 画头像
        Graphics.drawMatrixImage(canvas, ViewManager.head, 0, 0,
            ViewManager.head.getWidth(),ViewManager.head.getHeight(),
            Graphics.TRANS_MIRROR, 0, 0, 0, Graphics.TIMES_SCALE);
        Paint p = new Paint();
        p.setTextSize(30);
        // 画名字
        Graphics.drawBorderString(canvas, 0xa33e11, 0xffde00, name,
            ViewManager.head.getWidth(), (int) (ViewManager.scale * 20), 3, p);
        // 画生命值
        Graphics.drawBorderString(canvas, 0x066a14, 0x91ff1d, "HP: " + hp,
            ViewManager.head.getWidth(), (int) (ViewManager.scale * 40), 3, p);
    }
```

上面方法的实现非常简单，第一段粗体字代码调用 Graphics 绘制头像位图，第二段、第三段粗体字代码分别调用 drawBorderString()方法来绘制包边字，包边字的内容显示了角色的名字和生命值。

角色是否被子弹打中的方法与怪物是否被子弹打中的方法基本相似：只要判断子弹出现在角色图片覆盖的区域中，即可判断子弹打中了角色。

与怪物类相似的是，Player 类同样也需要提供绘制子弹的方法，该方法负责绘制该角色发射的所有子弹，而且在绘制子弹之前，应该先判断子弹是否已越过屏幕边界，如果子弹越过屏幕边界，就应该将其清除。由于绘制子弹的方法与 Monster 类中绘制子弹的方法大致相似，此处不再赘述。

由于角色发射子弹是受玩家单击按钮控制的，但本游戏的设定是角色发射子弹之后，必须等待一定时间才能发射下一发子弹，因此程序为 Player 定义了一个 leftShootTime 计数器，只要该计数器不等于 0，角色就处于发射子弹的状态，角色不能发射下一发子弹。

下面是程序 leftShootTime 的 getter 方法与发射子弹的方法代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

```
    public int getLeftShootTime()
    {
        return leftShootTime;
    }
```

```
// 发射子弹的方法
public void addBullet()
{
    // 计算子弹的初始 X 坐标
    int bulletX = getDir() == DIR_RIGHT ? X_DEFAULT +(int)
        (ViewManager.scale * 50) : X_DEFAULT - (int)(ViewManager.scale * 50);
    // 创建子弹对象
    Bullet bullet = new Bullet(Bullet.BULLET_TYPE_1, bulletX,
        y - (int) (ViewManager.scale * 60), getDir());
    // 将子弹添加到用户发射的子弹集合中
    bulletList.add(bullet);
    // 发射子弹时，将 leftShootTime 设置为最大值
    leftShootTime = MAX_LEFT_SHOOT_TIME;
    // 播放射击音效
    ViewManager.soundPool.play(ViewManager.soundMap.get(1), 1, 1, 0, 0, 1);
}
```

正如从上面粗体字代码所看到的，程序每次发射子弹时都会将 leftShootTime 设为最大值，而 leftShootTime 会随着动画帧的绘制不断地自减，只有当 leftShootTime 为 0 时才可判断角色已结束射击状态。这样后面程序控制角色发射子弹时，也需要先判断 leftShootTime 的值：只有当 leftShootTime 的值小于、等于 0 时（角色不处于发射状态），角色才可以发射子弹。

由于玩家还可以控制界面上的角色移动、跳动，因此程序还需要实现角色移动、角色移动与跳跃之间的关系。程序为 Player 提供了如下两个方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

```
// 处理角色移动的方法
public void move()
{
    if (move == MOVE_RIGHT)
    {
        // 更新怪物的位置
        MonsterManager.updatePosistion((int) (6 * ViewManager.scale));
        // 更新角色的位置
        setX(getX() + (int) (6 * ViewManager.scale));
        if (!isJump())
        {
            // 不跳的时候，需要设置动作
            setAction(Player.ACTION_RUN_RIGHT);
        }
    }
    else if (move == MOVE_LEFT)
    {
        if (getX() - (int) (6 * ViewManager.scale) < Player.X_DEFAULT)
        {
            // 更新怪物的位置
            MonsterManager.updatePosistion(-(getX() - Player.X_DEFAULT));
        }
        else
        {
            // 更新怪物的位置
            MonsterManager.updatePosistion(-(int) (6 * ViewManager.scale));
        }
        // 更新角色的位置
        setX(getX() - (int) (6 * ViewManager.scale));
        if (!isJump())
        {
            // 不跳的时候，需要设置动作
            setAction(Player.ACTION_RUN_LEFT);
        }
    }
```

```
        else if (getAction() != Player.ACTION_JUMP_RIGHT
            && getAction() != Player.ACTION_JUMP_LEFT)
        {
            if (!isJump())
            {
                // 不跳的时候，需要设置动作
                setAction(Player.ACTION_STAND_RIGHT);
            }
        }
    }
    // 处理角色移动与跳跃的逻辑关系
    public void logic()
    {
        if (!isJump())
        {
            move();
            return;
        }
        // 如果还没有跳到最高点
        if (!isJumpMax)
        {
            setAction(getDir() == Player.DIR_RIGHT ?
                Player.ACTION_JUMP_RIGHT : Player.ACTION_JUMP_LEFT);
            // 更新Y坐标
            setY(getY() - (int) (8 * ViewManager.scale));
            // 设置子弹在Y方向上具有向上的加速度
            setBulletYAccelate(-(int) (2 * ViewManager.scale));
            // 已经达到最高点
            if (getY() <= Player.Y_JUMP_MAX)
            {
                isJumpMax = true;
            }
        }
        else
        {
            jumpStopCount--;
            // 如果在最高点停留次数已经使用完
            if (jumpStopCount <= 0)
            {
                // 更新Y坐标
                setY(getY() + (int) (8 * ViewManager.scale));
                // 设置子弹在Y方向上具有向下的加速度
                setBulletYAccelate((int) (2 * ViewManager.scale));
                // 已经掉落到最低点
                if (getY() >= Player.Y_DEFALUT)
                {
                    // 恢复Y坐标
                    setY(Player.Y_DEFALUT);
                    isJump = false;
                    isJumpMax = false;
                    setAction(Player.ACTION_STAND_RIGHT);
                }
                else
                {
                    // 未掉落到最低点，继续使用跳跃的动作
                    setAction(getDir() == Player.DIR_RIGHT ?
                        Player.ACTION_JUMP_RIGHT : Player.ACTION_JUMP_LEFT);
                }
            }
        }
        // 控制角色移动
        move();
    }
```

18.3 实现绘图工具类

上面程序中大量使用了 Graphics 绘图工具类，这是本游戏对 Android 绘图的 Canvas 提供的一个封装，Graphics 类包含了各种常见的绘制几何图形、绘制位图的方法。该工具类中用于绘制几何图形的方法如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\game\Graphics.java

```
// 定义绘制直线的方法
public static void drawLine(Canvas canvas, float startX,
    float startY, float endX, float endY, Paint paint)
{
    paint.setStyle(Style.STROKE);
    canvas.drawLine(startX, startY, endX, endY, paint);
}
// 定义绘制矩形的方法
public static void drawRect(Canvas canvas, float x,
    float y, float w, float h, Paint paint)
{
    paint.setStyle(Style.STROKE);
    RectF rect = new RectF(x, y, x + w, y + h);
    canvas.drawRect(rect, paint);
}
// 定义绘制弧的方法
public static void drawArc(Canvas canvas, float x,
    float y, float width, float height,
    float startAngle, float sweepAngle, Paint paint)
{
    paint.setStyle(Style.STROKE);
    RectF rect = new RectF(x, y, x + width, y + height);
    canvas.drawArc(rect, startAngle, sweepAngle, true, paint);
}
```

需要说明的是，由于本游戏暂时并未需要绘制直线、矩形、弧等功能，因此上面 3 个方法其实在本游戏中并未用到，这 3 个方法只是为后续开发打下基础。

Graphics 类中用于绘制字符串和包边字符串的方法如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\game\Graphics.java

```
// 定义绘制字符串的方法
public static void drawString(Canvas canvas, String text,
    float textSize, float x, float y, int anchor, Paint paint)
{
    // 动态计算字符串的对齐方式
    if ((anchor & LEFT) != 0)
    {
        paint.setTextAlign(Align.LEFT);
    }
    else if ((anchor & RIGHT) != 0)
    {
        paint.setTextAlign(Align.RIGHT);
    }
    else if ((anchor & HCENTER) != 0)
    {
        paint.setTextAlign(Align.CENTER);
    }
    else
    {
        paint.setTextAlign(Align.CENTER);
    }
    paint.setTextSize(textSize);
```

```
        // 绘制字符串
        canvas.drawText(text, x, y, paint);
        // 将文本对齐方式恢复到默认情况
        paint.setTextAlign(Align.CENTER);
    }
    /**
     * 绘制包边字符串的方法
    * @param c           绘制包边字符串的画布
     * @param borderColor 绘制包边字符串的边框颜色
     * @param textColor   绘制包边字符串的文本颜色
     * @param text        指定要绘制的字符串
     * @param x           绘制字符串的X坐标
     * @param y           绘制字符串的Y坐标
     * @param borderWidth 绘制包边字符串的边框宽度
     * @param mPaint      绘制包边字符串的画笔
     */
    public static void drawBorderString(Canvas c, int borderColor,
        int textColor, String text, int x, int y, int borderWidth, Paint mPaint)
    {
        mPaint.setAntiAlias(true);
        // 先使用STROKE风格绘制字符串的边框
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setStrokeWidth(borderWidth);
        // 设置绘制边框的颜色
        mPaint.setColor(Color.rgb((borderColor & 0xFF0000) >> 16,
            (borderColor & 0x00ff00) >> 8, (borderColor & 0x0000ff)));
        c.drawText(text, x, y, mPaint);
        // 先使用FILL风格绘制字符串
        mPaint.setStyle(Paint.Style.FILL);
        // 设置绘制文字的颜色
        mPaint.setColor(Color.rgb((textColor & 0xFF0000) >> 16,
            (textColor & 0x00ff00) >> 8, (textColor & 0x0000ff)));
        c.drawText(text, x, y, mPaint);
    }
```

关于绘制包边字符串的逻辑其实也很简单，从程序中两行粗体字代码可以看出，其实就是先采用描边方式绘制字符串，再采用填充方式绘制字符串，这样即可形成包边字符串。

除此之外，Graphics还涉及如下绘制位图、处理位图的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\game\Graphics.java

```
// 定义Android翻转参数的常量
public static final int TRANS_MIRROR = 2;
public static final int TRANS_MIRROR_ROT180 = 1;
public static final int TRANS_MIRROR_ROT270 = 4;
public static final int TRANS_MIRROR_ROT90 = 7;
public static final int TRANS_NONE = 0;
public static final int TRANS_ROT180 = 3;
public static final int TRANS_ROT270 = 6;
public static final int TRANS_ROT90 = 5;
public static final float INTERVAL_SCALE = 0.05f; // 每次缩放的梯度
// 每次缩放的梯度是0.05，所以这里乘20以后转成整数
public static final int TIMES_SCALE = 20;
private static final float[] pts = new float[8];
private static final Path path = new Path();
private static final RectF srcRect = new RectF();
public synchronized static void drawMatrixImage(Canvas canvas, Bitmap src,
    int srcX, int srcY, int width, int height, int trans, int drawX,
    int drawY, int degree, int scale)
{
    if (canvas == null)
    {
```

```
        return;
    }
    if (src == null || src.isRecycled())
    {
        return;
    }
    int srcWidth = src.getWidth();
    int srcHeight = src.getHeight();
    if (srcX + width > srcWidth)
    {
        width = srcWidth - srcX;
    }
    if (srcY + height > srcHeight)
    {
        height = srcHeight - srcY;
    }
    if (width <= 0 || height <= 0)
    {
        return;
    }
    // 设置图片在横向、纵向上缩放因子
    int scaleX = scale;
    int scaleY = scale;
    int rotate = 0;
    // 根据程序所要进行的图像变换来计算 scaleX、scaleY 以及 rotate 旋转角
    switch (trans)
    {
        case TRANS_MIRROR_ROT180:
            scaleX = -scale;
            rotate = 180;
            break;
        case TRANS_MIRROR:
            scaleX = -scale;
            break;
        case TRANS_ROT180:
            rotate = 180;
            break;
        case TRANS_MIRROR_ROT270:
            scaleX = -scale;
            rotate = 270;
            break;
        case TRANS_ROT90:
            rotate = 90;
            break;
        case TRANS_ROT270:
            rotate = 270;
            break;
        case TRANS_MIRROR_ROT90:
            scaleX = -scale;
            rotate = 90;
            break;
        default:
            break;
    }
    // 如果 rotate、degree 为 0，则表明不涉及旋转
    // 如果 scaleX 等于 TIMES_SCALE，则表明不涉及缩放
    if (rotate == 0 && degree == 0
        && scaleX == TIMES_SCALE)
    {    // 即 scale=1 无缩放， rotate=0 无旋转
        drawImage(canvas, src, drawX, drawY, srcX, srcY, width, height);
    }
    else
    {
        Matrix matrix = new Matrix();
```

```
            matrix.postScale(scaleX * INTERVAL_SCALE, scaleY * INTERVAL_SCALE);
            matrix.postRotate(rotate); // 对Matrix旋转rotate
            matrix.postRotate(degree); // 对Matrix旋转degree
            srcRect.set(srcX, srcY, srcX + width, srcY + height);
            matrix.mapRect(srcRect);
            matrix.postTranslate(drawX - srcRect.left, drawY - srcRect.top);
            pts[0] = srcX;
            pts[1] = srcY;
            pts[2] = srcX + width;
            pts[3] = srcY;
            pts[4] = srcX + width;
            pts[5] = srcY + height;
            pts[6] = srcX;
            pts[7] = srcY + height;
            matrix.mapPoints(pts);
            canvas.save();
            path.reset();
            path.moveTo(pts[0], pts[1]);
            path.lineTo(pts[2], pts[3]);
            path.lineTo(pts[4], pts[5]);
            path.lineTo(pts[6], pts[7]);
            path.close();
            canvas.clipPath(path);
            // 使用matrix变换矩阵绘制位图
            canvas.drawBitmap(src, matrix, null);
            canvas.restore();
        }
    }
    // 工具方法：绘制位图
    // 作用是将源位图image中左上角为srcX、srcY、宽width、高height的区域绘制到canvas上
    public synchronized static void drawImage(Canvas canvas, Bitmap image,
        int destX, int destY, int srcX, int srcY, int width, int height)
    {
        if (canvas == null)
        {
            return;
        }
        if (image == null || image.isRecycled())
        {
            return;
        }
        // 如果源位图的区域比需要绘制的目标区域更小
        if (srcX == 0 && srcY == 0 && image.getWidth() <= width
            && image.getHeight() <= height)
        {
            canvas.drawBitmap(image, destX, destY, null);
            return;
        }
        src.left = srcX;
        src.right = srcX + width;
        src.top = srcY;
        src.bottom = srcY + height;
        dst.left = destX;
        dst.right = destX + width;
        dst.top = destY;
        dst.bottom = destY + height;
        // 将image的src区域挖取出来，绘制在canvas的dst区域上
        canvas.drawBitmap(image, src, dst, null);
    }
    // 缩放图片，得到指定宽高的新图片
    // 使用浮点数运算，减少除法计算时的误差
    public static Bitmap scale(Bitmap img, float newWidth, float newHeight)
    {
        if (img == null || img.isRecycled())
```

```
        {
            return null;
        }
        float height = img.getHeight();
        float width = img.getWidth();
        if (height == 0 || width == 0 || newWidth == 0 || newHeight == 0)
        {
            return null;
        }
        // 创建缩放图片所需的 Matrix
        Matrix matrix = new Matrix();
        matrix.postScale(newWidth / width, newHeight / height);
        try
        {
            // 生成对 img 缩放之后的图片
            return Bitmap.createBitmap(img, 0, 0,
                (int) width, (int) height, matrix, true);
        }
        catch (Exception e)
        {
            return null;
        }
    }
    // 对原有位图执行镜像变换
    public static Bitmap mirror(Bitmap img)
    {
        if (img == null || img.isRecycled())
        {
            return null;
        }
        // 创建对图片执行镜像变换的 Matrix
        Matrix matrix = new Matrix();
        matrix.postScale(-1f, 1f);
        try
        {
            // 生成对 img 执行镜像变换之后的图片
            return Bitmap.createBitmap(img, 0, 0, img.getWidth(),
                img.getHeight(), matrix, true);
        }
        catch (Exception e)
        {
            return null;
        }
    }
```

上面程序一共定义了 4 个绘制位图和处理位图的方法。

- drawMatrixImage：该方法用于从源位图中的 srcX、srcY 点开始，挖取宽 width、高 height 的区域，并对该图片进行 trans 变换、缩放 scale（当 scale 为 20 时表示不缩放）、旋转 degree 角度后绘制到 Canvas 的 drawX、drawY 处。
- drawImage：该方法用于从源位图中的 srcX、srcY 点开始，挖取宽 width、高 height 的区域，并将这块区域的图片绘制到 Canvas 的 destX、destY 处。
- scale：该方法用于对图片进行缩放。
- mirror：该方法用于对图片进行镜像变换。

上面 4 个方法的代码实现，读者可参考本书前面介绍 Canvas、Matrix 的部分进行理解。需要说明的是，本游戏并未用到上面的 mirror 方法，这个方法只是为游戏的后续开发做准备。

18.4 加载、管理游戏图片

为了统一管理游戏中所有的图片、声音资源，本游戏开发了一个 ViewManager 工具类，该工具类除了加载、管理游戏的图片、声音资源之外，还负责绘制游戏主界面。

ViewManager 定义了如下成员变量来管理游戏涉及的图片、声音资源。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

```
// 定义一个 SoundPool
public static SoundPool soundPool;
public static HashMap<Integer, Integer> soundMap = new HashMap<>();
// 地图图片
public static Bitmap map = null;
// 保存角色站立时腿部动画帧的图片数组
public static Bitmap[] legStandImage = null;
// 保存角色站立时头部动画帧的图片数组
public static Bitmap[] headStandImage = null;
// 保存角色跑动时腿部动画帧的图片数组
public static Bitmap[] legRunImage = null;
// 保存角色跑动时头部动画帧的图片数组
public static Bitmap[] headRunImage = null;
// 保存角色跳动时腿部动画帧的图片数组
public static Bitmap[] legJumpImage = null;
// 保存角色跳动时头部动画帧的图片数组
public static Bitmap[] headJumpImage = null;
// 保存角色射击时头部动画帧的图片数组
public static Bitmap[] headShootImage = null;
// 加载所有子弹的图片
public static Bitmap[] bulletImage = null;
// 绘制角色的图片
public static Bitmap head = null;
// 保存第一种怪物（炸弹）未爆炸时动画帧的图片
public static Bitmap[] bombImage = null;
// 保存第一种怪物（炸弹）爆炸时动画帧的图片
public static Bitmap[] bomb2Image = null;
// 保存第二种怪物（飞机）的动画帧的图片
public static Bitmap[] flyImage = null;
// 保存第二种怪物（飞机）爆炸的动画帧的图片
public static Bitmap[] flyDieImage = null;
// 保存第三种怪物（人）的动画帧的图片
public static Bitmap[] manImgae = null;
// 保存第三种怪物（人）死亡时动画帧的图片
public static Bitmap[] manDieImage = null;
// 定义游戏对图片的缩放比例
public static float scale = 1f;
public static int SCREEN_WIDTH;
public static int SCREEN_HEIGHT;
```

正如上面程序中前两行粗体字代码所看到的，程序使用了 SoundPool 来管理游戏中的各种音效，比如角色发射子弹的声音、怪物被打中、爆炸的声音等。

除此之外，该类大量使用了位图数组来存储游戏需要使用的动画帧。

该类定义了一个 loadResource()方法来加载所有的图片、声音资源。该方法的代码如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

```
// 加载所有游戏图片、声音的方法
public static void loadResource()
{
    AudioAttributes attr = new AudioAttributes.Builder()
```

```
            .setUsage(AudioAttributes.USAGE_GAME)
            .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
            .build();
        soundPool = new SoundPool.Builder()
            .setAudioAttributes(attr)
            .setMaxStreams(10).build();
        // load方法加载指定音频文件，并返回所加载的音频ID
        // 此处使用HashMap来管理这些音频流
        soundMap.put(1, soundPool.load(MainActivity.mainActivity, R.raw.shot, 1));
        soundMap.put(2, soundPool.load(MainActivity.mainActivity, R.raw.bomb, 1));
        soundMap.put(3, soundPool.load(MainActivity.mainActivity, R.raw.oh, 1));
        Bitmap temp = createBitmapByID(MainActivity.res, R.drawable.map);
        if (temp != null && !temp.isRecycled())
        {
            int height = temp.getHeight();
            if (height != SCREEN_HEIGHT && SCREEN_HEIGHT != 0)
            {
                scale = (float) SCREEN_HEIGHT / (float) height;
                map = Graphics.scale(temp, temp.getWidth() * scale, height * scale);
                temp.recycle();
            }
            else
            {
                map = temp;
            }
        }
        // 加载角色站立时腿部动画帧的图片
        legStandImage = new Bitmap[1];
        legStandImage[0] = createBitmapByID(MainActivity.res, R.drawable.leg_stand, scale);
        // 加载角色站立时头部动画帧的图片
        headStandImage = new Bitmap[3];
        headStandImage[0] = createBitmapByID(MainActivity.res, R.drawable.head_stand_1, scale);
        headStandImage[1] = createBitmapByID(MainActivity.res, R.drawable.head_stand_2, scale);
        headStandImage[2] = createBitmapByID(MainActivity.res, R.drawable.head_stand_3, scale);
        ...
    }
```

上面程序中第一段粗体字代码创建了SoundPool对象，这样即可使用SoundPool来管理短促的游戏音效。接下来第二段粗体字代码大量调用createBitmapByID()工具方法来加载位图，这些代码就是根据需要为游戏中角色、怪物的动画帧加载位图。实际上，上面程序中还包含大量类似的代码，这些代码都用于加载游戏中其他动画帧的位图。

提示：

随着游戏规模的增大，游戏可能需要添加更多的怪物、更多的角色，那么此处加载动画帧的代码将会更多。

上面程序代码中加载动画帧的位图时用到一个createBitmapByID()工具方法，该方法的代码如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

```
// 工具方法：根据图片ID来获取实际的位图，并按scale进行缩放
private static Bitmap createBitmapByID(Resources res, int resID, float scale)
{
    try
    {
        InputStream is = res.openRawResource(resID);
        Bitmap bitmap = BitmapFactory.decodeStream(is, null, null);
        if (bitmap == null || bitmap.isRecycled())
        {
            return null;
```

```
        }
        if (scale <= 0 || scale == 1f)
        {
            return bitmap;
        }
        int wdith = (int) (bitmap.getWidth() * scale);
        int height = (int) (bitmap.getHeight() * scale);
        Bitmap newBitmap = Graphics.scale(bitmap, wdith, height);
        if (!bitmap.isRecycled() && !bitmap.equals(newBitmap))
        {
            bitmap.recycle();
        }
        return newBitmap;
    }
    catch (Exception e)
    {
        return null;
    }
}
```

除此之外，ViewManager 还定义了一个 drawGame()方法，该方法负责整个游戏场景。该方法的实现思路就是先绘制游戏地图，然后绘制游戏角色，最后绘制所有的怪物即可。下面是drawGame()方法的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

```
// 绘制游戏界面的方法，该方法先绘制游戏背景地图，再绘制游戏角色，最后绘制所有怪物
public static void drawGame(Canvas canvas)
{
    if (canvas == null)
    {
        return;
    }
    // 画地图
    if (map != null && !map.isRecycled())
    {
        int width = map.getWidth() + GameView.player.getShift();
        // 绘制 map 图片，也就是绘制地图
        Graphics.drawImage(canvas, map, 0, 0, -GameView.player.getShift()
            , 0 ,width, map.getHeight());
        int totalWidth = width;
        // 采用循环，保证地图前后可以拼接起来
        while (totalWidth < ViewManager.SCREEN_WIDTH)
        {
            int mapWidth = map.getWidth();
            int drawWidth = ViewManager.SCREEN_WIDTH - totalWidth;
            if (mapWidth < drawWidth)
            {
                drawWidth = mapWidth;
            }
            Graphics.drawImage(canvas, map, totalWidth, 0, 0, 0,
                    drawWidth, map.getHeight());
            totalWidth += drawWidth;
        }
    }
    // 画角色
    GameView.player.draw(canvas);
    // 画怪物
    MonsterManager.drawMonster(canvas);
}
```

上面方法中第一行粗体字代码使用 drawImage()方法来绘制背景位图，第二行粗体字代码依然使用了 drawImage()方法来绘制背景位图——这是因为当角色在地图上不断地向右移动

时，随着地图不断地向左拖动，地图就会不能完全覆盖屏幕右边，此时需要再绘制一张背景位图，这样才可以拼成完成的地图——这样就形成了无限循环的游戏地图。

18.5 实现游戏界面

至此，合金弹头游戏所需要的各种游戏元素准备完成，游戏所需的各种资源也准备完成了，接下来只要在程序界面上将这些游戏元素显示出来即可。

18.5.1 实现游戏 Activity

先看本游戏的 Activity。本游戏的 Activity 一样需要继承 Android 的 Activity 基类，并重写其生命周期方法。该 Activity 的代码如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\MainActivity.java

```
public class MainActivity extends Activity
{
    // 定义主布局内的容器：FrameLayout
    public static FrameLayout mainLayout = null;
    // 主布局的布局参数
    public static FrameLayout.LayoutParams mainLP = null;
    // 定义资源管理的核心类
    public static Resources res = null;
    public static MainActivity mainActivity = null;
    // 定义成员变量记录游戏窗口的宽度、高度
    public static int windowWidth;
    public static int windowHeight;
    // 游戏窗口的主游戏界面
    public static GameView mainView = null;
    // 播放背景音乐的 MediaPlayer
    private MediaPlayer player;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        mainActivity = this;
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // 设置全屏
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
        DisplayMetrics metric = new DisplayMetrics();
        // 获取屏幕高度、宽度
        getWindowManager().getDefaultDisplay().getMetrics(metric);
        windowHeight = metric.heightPixels;  // 屏幕高度
        windowWidth = metric.widthPixels;  // 屏幕宽度
        getWindow().setSoftInputMode(
            WindowManager.LayoutParams.SOFT_INPUT_ADJUST_PAN);
        res = getResources();
        // 加载 main.xml 界面设计文件
        setContentView(R.layout.main);
        // 获取 main.xml 界面设计文件中 ID 为 mainLayout 的组件
        mainLayout = (FrameLayout) findViewById(R.id.mainLayout);
        // 创建 GameView 组件
        mainView = new GameView(this.getApplicationContext()
            , GameView.STAGE_INIT);
        mainLP = new FrameLayout.LayoutParams(MATCH_PARENT, MATCH_PARENT);
        mainLayout.addView(mainView, mainLP);
        // 播放背景音乐
        player = MediaPlayer.create(this, R.raw.background);
```

```
        player.setLooping(true);
        player.start();
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 当游戏暂停时，暂停播放背景音乐
        if(player != null && !player.isPlaying())
        {
            player.start();
        }
    }
    @Override
    public void onPause()
    {
        super.onPause();
        // 当游戏恢复时，如果没有播放背景音乐，开始播放背景音乐
        if(player != null && player.isPlaying())
        {
            player.pause();
        }
    }
}
```

从上面 Activity 的 3 行粗体字代码可以看出，该游戏的 Activity 非常简单，程序只是加载 main.xml 界面设计文件作为游戏界面，然后程序创建了一个自定义组件：GameView 实例，并将该实例添加到 FrameLayout 容器中，其中 FrameLayout 是 main.xml 界面设计文件包含的根 UI 组件。

上面的 MainActivity 重写了 onResume()、onPause()两个生命周期方法，这样可以实现当游戏暂停时暂停播放背景音乐，当游戏恢复时继续播放背景音乐。

除此之外，上面 Activity 还设置了窗口的软键盘输入方式，而且本游戏只支持横屏方式，因此应该在 AndroidManifest.xml 文件中对该 Activity 进行如下配置。

程序清单：codes\18\MetalSlug\app\src\main\AndroidManifest.xml

```
<!-- 设置 Activity 所需的屏幕方向、软键盘输入方式 -->
<activity android:name=".MainActivity"
    android:label="@string/app_name"
    android:keepScreenOn="true"
    android:screenOrientation="landscape"
    android:configChanges="orientation|keyboardHidden|screenSize"
    android:windowSoftInputMode="adjustPan">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

从上面代码可以看出，该游戏的主界面主要由 GameView 这个自定义组件来负责提供。下面详细介绍 GameView 组件的实现。

18.5.2 实现主视图

本 GameView 为了获取更好的游戏性能，程序让 GameView 继承 SurfaceView，而不是继承普通的 View，这是因为 SurfaceView 在游戏绘图方面比普通的 View 更出色。

GameView 主要负责管理界面上的角色、当前游戏场景，以及绘图相关的 API。GameView 所需的成员变量如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
public static final Player player = new Player("孙悟空", Player.MAX_HP);
// 保存当前 Android 应用的主 Context
private Context mainContext = null;
// 画图所需要的 Paint 和 Canvas 对象
private Paint paint = null;
private Canvas canvas = null;
// SurfaceHolder 负责维护 SurfaceView 上绘制的内容
private SurfaceHolder surfaceHolder;
// 代表场景不改变的常量
public static final int STAGE_NO_CHANGE = 0;
// 代表初始化场景的常量
public static final int STAGE_INIT = 1;
// 代表登录场景的常量
public static final int STAGE_LOGIN = 2;
// 代表游戏场景的常量
public static final int STAGE_GAME = 3;
// 代表失败场景的常量
public static final int STAGE_LOSE = 4;
// 代表退出场景的常量
public static final int STAGE_QUIT = 99;
// 代表错误场景的常量
public static final int STAGE_ERROR = 255;
// 定义该游戏当前处于何种场景的变量
private int gStage = 0;
// 定义一个集合来保存该游戏已经加载的所有场景
public static final List<Integer> stageList =
    Collections.synchronizedList(new ArrayList<Integer>());
```

上面程序中第一行粗体字代码定义了该 GameView 要维护的角色，接下来三行粗体字代码定义的 Canvas、Paint、SurfaceHolder 都是绘图相关的 API，最后一行粗体字代码定义的 gStage 则用于记录当前游戏处于何种场景：这是因为本 GameView 将会负责绘制所有的游戏场景，包括游戏开始时的登录场景、游戏失败的场景、正在游戏的场景。

除此之外，GameView 还定义了一个 stageList 来保存游戏已经加载的所有场景。

GameView 的构造器主要用于对上面程序中的各种成员变量执行初始化，该构造器此处不再赘述。

由于 GameView 会负责绘制游戏的所有场景，因此程序为 GameView 提供了如下方法来处理各种场景。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
// 处理游戏场景
public int doStage(int stage, int step)
{
    int nextStage;
    switch (stage)
    {
        case STAGE_INIT:
            nextStage = doInit(step);
            break;
        case STAGE_LOGIN:
            nextStage = doLogin(step);
            break;
        case STAGE_GAME:
            nextStage = doGame(step);
            break;
        case STAGE_LOSE:
            nextStage = doLose(step);
```

```
            break;
        default:
            nextStage = STAGE_ERROR;
            break;
    }
    return nextStage;
}
```

从上面粗体字代码可以看出，doStage()方法的处理逻辑非常简单，该程序根据 GameView 所处的场景来执行相应的方法，而这些方法则分别负责绘制游戏的不同场景。

- doInit：该方法负责执行初始化。
- doLogin：该方法负责绘制游戏登录界面。
- doGame：该方法负责绘制游戏界面。
- doLose：该方法负责绘制游戏失败界面。

GameView 的 doStage()方法则由游戏线程负责调度，该线程会不断地执行 doStage()方法，这样程序只要改变 GameView 的 stage 变量，即可让 GameView 绘制不同的场景。

上面方法还涉及一个 step 参数，该参数用于控制当前场景处于哪个步骤。为了更好地管理这些步骤，GameView 还提供了如下常量：

```
// 步骤：初始化
private static final int INIT = 1;
// 步骤：逻辑
private static final int LOGIC = 2;
// 步骤：清除
private static final int CLEAN = 3;
// 步骤：绘图
private static final int PAINT = 4;
```

下面是 doInit()方法的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
// 执行初始化的方法
public int doInit(int step)
{
    // 初始化游戏图片
    ViewManager.loadResource();
    // 跳转到登录界面
    return STAGE_LOGIN;
}
```

从上面最后一行代码可以看出，doInit()方法执行完成后，将会进入登录场景。下面是 GameView 的 doLogin()方法的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
// 定义登录界面
private RelativeLayout loginView;
public int doLogin(int step)
{
    switch (step)
    {
        case INIT:
            // 初始化角色血量
            player.setHp(Player.MAX_HP);
            // 初始化登录界面
            if (loginView == null)
            {
                loginView = new RelativeLayout(mainContext);
                loginView.setBackgroundResource(R.drawable.game_back);
                // 创建按钮
```

```
            Button button = new Button(mainContext);
            // 设置按钮的背景图片
            button.setBackgroundResource(R.drawable.button_selector);
            RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
            params.addRule(RelativeLayout.CENTER_IN_PARENT);
            // 添加按钮
            loginView.addView(button, params);
            button.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    // 将游戏场景的常量添加到 stageList 集合中
                    stageList.add(STAGE_GAME);
                }
            });
            // 通过 Handler 通知主界面加载 loginView 组件
            setViewHandler.sendMessage(
                setViewHandler.obtainMessage(0, loginView)); // ①
        }
        break;
    case LOGIC:
        break;
    case CLEAN:
        // 清除登录界面
        if (loginView != null)
        {
            // 通过 Handler 通知主界面删除 loginView 组件
            delViewHandler.sendMessage(
                delViewHandler.obtainMessage(0, loginView)); // ②
            loginView = null;
        }
        break;
    case PAINT:
        break;
    }
    return STAGE_NO_CHANGE;
}
```

doLogin()方法的处理逻辑比较清晰，如果登录场景当前处于 INIT 步骤，程序将会创建一个 loginView，并向 loginView 中添加一个按钮。接下来程序在①号粗体字代码处通过 Handler 将 loginView 添加到主界面中；如果游戏场景当前处于 CLEAN 步骤，程序将会在②号粗体字代码处通过 Handler 将 loginView 从界面上删除。

下面是 setViewHandler 的创建代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
public Handler setViewHandler = new Handler()
{
    public void handleMessage(Message msg)
    {
        RelativeLayout layout = (RelativeLayout) msg.obj;
        if (layout != null)
        {
            RelativeLayout.LayoutParams params = new RelativeLayout
                .LayoutParams(LayoutParams.MATCH_PARENT,
                LayoutParams.MATCH_PARENT);
            MainActivity.mainLayout.addView(layout, params);
        }
    }
};
```

从上面粗体字代码即可看出，setViewHandler 的功能就是将接收到的消息携带的组件添加到主界面上。delViewHandler 的实现代码与此相似，只是该 Handler 负责从界面上删除组件而已。

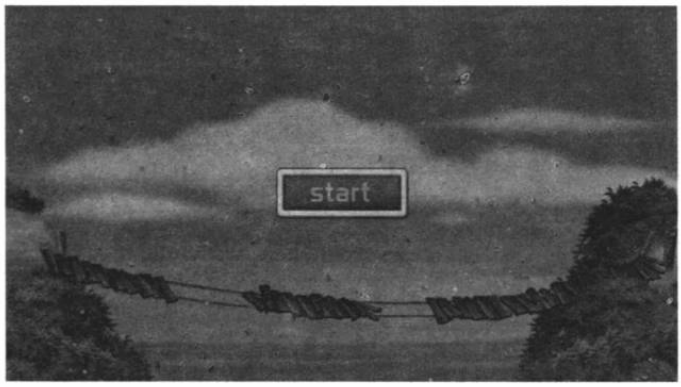


图 18.2　游戏登录界面

当游戏开始之后，游戏登录界面如图 18.2 所示。

从登录场景上按钮的事件监听器代码可以看出，当玩家按下图 18.2 所示界面上的“start”按钮时，程序将会进入游戏场景，游戏场景由 doGame()负责提供。下面是 doGame()方法的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
// 定义游戏界面
RelativeLayout gameLayout = null;
private static final int ID_LEFT = 9000000;
private static final int ID_FIRE = ID_LEFT + 1;
public int doGame(int step)
{
    switch (step)
    {
        case INIT:
            // 初始化游戏界面
            if (gameLayout == null)
            {
                gameLayout = new RelativeLayout(mainContext);
                // 添加向左移动的按钮
                Button button = new Button(mainContext);
                button.setId(ID_LEFT);
                // 设置按钮的背景图片
                button.setBackground(getResources().getDrawable(R.drawable.left));
                RelativeLayout.LayoutParams params = new RelativeLayout
                    .LayoutParams(LayoutParams.WRAP_CONTENT,
                    LayoutParams.WRAP_CONTENT);
                params.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
                params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
                params.setMargins((int) (ViewManager.scale * 20),
                    0, 0, (int) (ViewManager.scale * 10));
                // 向游戏界面上添加向左的按钮
                gameLayout.addView(button, params);
                // 为按钮添加事件监听器
                button.setOnTouchListener(new OnTouchListener()
                {
                    @Override
                    public boolean onTouch(View v, MotionEvent event)
                    {
                        switch (event.getAction())
                        {
                            case MotionEvent.ACTION_DOWN:
                                player.setMove(Player.MOVE_LEFT);
                                break;
                            case MotionEvent.ACTION_UP:
                                player.setMove(Player.MOVE_STAND);
                                break;
                            case MotionEvent.ACTION_MOVE:
                                break;
```

```
            }
            return false;
        }
    });
    // 添加向右移动的按钮
    button = new Button(mainContext);
    // 设置按钮的背景图片
    button.setBackground(getResources().getDrawable(R.drawable.right));
    params = new RelativeLayout.LayoutParams(
        LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
    params.addRule(RelativeLayout.RIGHT_OF, ID_LEFT);
    params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
    params.setMargins((int) (ViewManager.scale * 20),
        0, 0, (int) (ViewManager.scale * 10));
    // 向游戏界面上添加向右的按钮
    gameLayout.addView(button, params);
    // 为按钮添加事件监听器
    button.setOnTouchListener(new OnTouchListener()
    {
        public boolean onTouch(View v, MotionEvent event)
        {
            switch (event.getAction())
            {
                case MotionEvent.ACTION_DOWN:
                    player.setMove(Player.MOVE_RIGHT);
                    break;
                case MotionEvent.ACTION_UP:
                    player.setMove(Player.MOVE_STAND);
                    break;
                case MotionEvent.ACTION_MOVE:
                    break;
            }
            return false;
        }
    });
    // 添加射击按钮
    button = new Button(mainContext);
    button.setId(ID_FIRE);
    // 设置按钮的背景图片
    button.setBackground(getResources().getDrawable(R.drawable.fire));
    params = new RelativeLayout.LayoutParams(
        LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
    params.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
    params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
    params.setMargins(0, 0, (int) (ViewManager.scale * 20),
        (int) (ViewManager.scale * 10));
    // 向游戏界面上添加射击的按钮
    gameLayout.addView(button, params);
    // 为按钮添加事件监听器
    button.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            // 当角色的 leftShootTime 为 0 时（上一枪发射结束），才能发射下一枪
            if(player.getLeftShootTime() <= 0)
            {
                player.addBullet();
            }
        }
    });
    // 添加跳跃的按钮
    button = new Button(mainContext);
```

```
            // 设置按钮的背景图片
            button.setBackground(getResources().getDrawable(R.drawable.jump));
            params = new RelativeLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
            params.addRule(RelativeLayout.LEFT_OF, ID_FIRE);
            params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
            params.setMargins(0, 0, (int) (ViewManager.scale * 20),
                (int) (ViewManager.scale * 10));
            // 向游戏界面上添加跳跃的按钮
            gameLayout.addView(button, params);
            // 为按钮添加事件监听器
            button.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    player.setJump(true);
                }
            });
            setViewHandler.sendMessage(setViewHandler
                .obtainMessage(0, gameLayout)); // ③
        }
        break;
    case LOGIC:
        // 随机生成怪物
        MonsterManager.generateMonster();
        // 检查碰撞
        MonsterManager.checkMonster();
        // 角色跳跃与移动
        player.logic();
        // 角色死亡
        if (player.isDie())
        {
            stageList.add(STAGE_LOSE);
        }
        break;
    case CLEAN:
        // 清除游戏界面
        if (gameLayout != null)
        {
            delViewHandler.sendMessage(delViewHandler
                .obtainMessage(0, gameLayout)); // ④
            gameLayout = null;
        }
        break;
    case PAINT:
        // 画游戏元素
        ViewManager.clearScreen(canvas);
        ViewManager.drawGame(canvas);
        break;
    }
    return STAGE_NO_CHANGE;
}
```

doGame()方法稍微复杂一些，doGame()方法可分为如下几种情况。

- 当游戏处于初始化步骤时，doGame()方法会创建一个 RelativeLayout 容器，并向该容器中添加 4 个按钮（这 4 个按钮将会放在界面最下方）。最后由程序中③号粗体字代码将该 RelativeLayout 容器添加到主界面上。
- 当游戏处于逻辑步骤（LOGIC）时，doGame()方法会调用 MonsterManager、Player 的方法来管理怪物、处理角色跳跃和移动、角色死亡等事件。

- 当游戏处于清除步骤时，doGame()方法将会使用④号粗体字代码将该 RelativeLayout 容器从主界面上清除。
- 当游戏处于绘图步骤（PAINT）时，doGame()方法将会调用 ViewManager 的 drawGame()方法来绘制整个游戏界面。

从上面代码可以看出，虽然 doGame()方法的代码比较多，但其实这些代码主要就是创建了 RelativeLayout 容器，并为之添加了 4 个按钮。当游戏处于初始化步骤时，doGame()方法添加的 4 个按钮按如图 18.3 所示的方式分布。

图 18.3 游戏界面上的 4 个按钮

当游戏失败时，GameView 将会使用 doLose()方法来显示失败界面。doLose()方法的代码如下。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
// 定义游戏失败界面
private RelativeLayout loseView;
public int doLose(int step)
{
    switch (step)
    {
        case INIT:
            // 初始化失败界面
            if (loseView == null)
            {
                // 创建失败界面
                loseView = new RelativeLayout(mainContext);
                loseView.setBackgroundResource(R.drawable.game_back);
                Button button = new Button(mainContext);
                button.setBackgroundResource(R.drawable.again);
                RelativeLayout.LayoutParams params = new RelativeLayout
                    .LayoutParams(LayoutParams.WRAP_CONTENT,
                    LayoutParams.WRAP_CONTENT);
                params.addRule(RelativeLayout.CENTER_IN_PARENT);
                loseView.addView(button, params);
                button.setOnClickListener(new OnClickListener()
                {
                    @Override
                    public void onClick(View v)
                    {
                        // 跳转到继续游戏的界面
                        stageList.add(STAGE_GAME);
                        // 让角色的生命值回到最大值
                        player.setHp(Player.MAX_HP);
                    }
                });
                setViewHandler.sendMessage(setViewHandler
                    .obtainMessage(0, loseView));
            }
            break;
        case LOGIC:
            break;
        case CLEAN:
            // 清除界面
            if (loseView != null)
            {
                delViewHandler.sendMessage(delViewHandler
                    .obtainMessage(0, loseView));
```

```
                loseView = null;
            }
            break;
        case PAINT:
            break;
    }
    return STAGE_NO_CHANGE;
}
```

从上面代码可以看出，doLose()与doLogin()的实现逻辑基本相似，同样只是在界面上显示一个按钮，用于“原地复活”游戏。如图 18.4 所示是游戏失败时显示的界面。

程序最后还需要定义一条线程，该线程负责让游戏界面上的所有角色和怪物“动”起来。下面是游戏线程的代码。

图 18.4　游戏失败界面

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

```
// 两次调度之间默认的暂停时间
public static final int SLEEP_TIME = 40;
// 最小暂停时间
public static final int MIN_SLEEP = 5;
class GameThread extends Thread
{
    public SurfaceHolder surfaceHolder = null;
    public boolean needStop = false;
    public GameThread(SurfaceHolder holder)
    {
        this.surfaceHolder = holder;
    }
    public void run()
    {
        long t1, t2;
        Looper.prepare();
        synchronized (surfaceHolder)
        {
            // 游戏未退出
            while (gStage != STAGE_QUIT && needStop == false)
            {
                try
                {
                    // 处理游戏的场景逻辑
                    stageLogic();
                    t1 = System.currentTimeMillis();
                    canvas = surfaceHolder.lockCanvas();
                    if (canvas != null)
                    {
                        // 处理游戏场景
                        doStage(gStage, PAINT);
                    }
                    t2 = System.currentTimeMillis();
                    int paintTime = (int) (t2 - t1);
                    long millis = SLEEP_TIME - paintTime;
                    if (millis < MIN_SLEEP)
                    {
                        millis = MIN_SLEEP;
                    }
```

```
                // 该线程暂停 millis 毫秒后再次调用 doStage()方法
                sleep(millis);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            finally
            {
                try
                {
                    if (canvas != null)
                    {
                        surfaceHolder.unlockCanvasAndPost(canvas);
                    }
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        }
        Looper.loop();
        try
        {
            sleep(1000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

从上面线程的 run()方法中的粗体字代码可以看出，该线程所做的事情非常简单：只要游戏还未退出，该线程就会每隔 40ms 执行一次 stageLogic()方法（处理场景逻辑的方法）和 doStage()方法（根据当前场景执行绘制的方法），这意味着该游戏将会每隔 40ms 就会重绘一次游戏界面，这样即可让整个游戏界面上的角色、怪物都“动”起来。

最后程序还会在 SurfaceHolder.Callback 的 surfaceCreated()方法中启动该线程。下面是程序重写的 surfaceCreated()方法的代码。

```
@Override
public void surfaceCreated(SurfaceHolder holder)
{
    // 启动主线程执行部分
    paint.setTextSize(15);
    if (thread != null)
    {
        thread.needStop = true;
    }
    thread = new GameThread(surfaceHolder);
    thread.start();
}
```

上面粗体字代码就是创建并启动游戏线程的代码。当 SurfaceView 被加载时，surfaceCreated()方法将会被自动激发，这样整个游戏线程就被启动起来了，从而控制整个游戏界面上的角色、怪物都“动”起来。

18.6 本章小结

本章开发了一个非常经典的射击类游戏：合金弹头。开发这款流行的小游戏难度适中，而且能充分激发学习热情，对于 Android 学习者来说是一个不错的选择。学习本章可以帮助读者掌握开发手机游戏的基本功：开发者应该用面向对象的方式来定义界面上的所有角色、怪物、子弹……一些能与玩家交互的元素。除此之外，游戏的动画管理，以及动画后面的线程支持，还有游戏的发射子弹，子弹是否打中目标，玩家控制角色移动、跳跃、射击等行为的处理，都值得读者好好学习，这些都是开发手机游戏需要重点掌握的能力。

CHAPTER

19

第19章 电子拍卖系统

本章要点

- Android 应用与传统应用整合的意义
- Android 应用整合传统应用的方式
- 基于 HttpClient、JSON 数据交换的整合方式
- JSON 基本知识
- JSON 语法
- 开发服务器端生成 JSON 响应的 Servlet
- 开发 Android 客户端界面
- 使用 HttpClient 发送请求
- 使用 HttpClient 获取服务器响应
- 将服务器响应转换为 JSON 对象或数组
- 通过 Android 客户端加载服务器响应

本章介绍了一个实用的 Android 应用：电子拍卖系统。本章所介绍的应用不再是普通的 Android 项目，而是一个 Android + Struts 2 + Spring 4+ Hibernate 4 整合的应用。Struts 2 + Spring 4 + Hibernate 4 提供了一个 B/S 结构的电子拍卖系统，对于使用 PC 的用户而言，他们可以使用浏览器来访问该系统；对于使用 Android 手机的用户而言，可以选择安装 Android 客户端程序，这样即可通过手机来使用该拍卖系统了。

与一般图书中所介绍的 toy 项目不同，本应用的服务器端采用了完整的 Java EE 应用架构：在技术实现上依赖于最流行的 Struts 2 + Spring + Hibernate 组合；应用架构采用了最流行的、具有高度可扩展性的控制器层 + 业务逻辑层 + DAO 层的分层架构。Android 客户端通过网络与服务器端的控制器组件交互，整个应用具有极好的示范性。

本应用的用户界面兼顾了手机和平板电脑两种设备，因此必须为手机屏幕和平板电脑屏幕设计用户界面。但由于该 Android 应用界面大量采用了 Fragment，而 Fragment 代表了可复用的“Activity 片段”，因此兼顾两种屏幕的界面复用了共同的 Fragment。

提示：

Android 系统发展的大势是与传统服务器应用系统整合，因为手机的硬件资源毕竟是有限的——计算能力有限，存储能力也有限，所以 Android 应用更适合作为应用的客户端——手机携带方便，可以随时随地开机运行应用，而且可以随时访问网络，并通过网络与服务器端应用交互，获取服务器端的数据。在未来的几年内，运行在手机上的电子商务客户端（各种电商系统都已有了手机客户端）、证券系统客户端、金融系统客户端将会大量出现。

由于本章介绍的重点是 Android 开发，因此将会详细介绍 Android 应用的开发，以及如何让 Android 应用与远程服务器交互。本章不会全面、详细地介绍服务器端的 Spring、Hibernate 开发，这些内容不是短短的一章可以讲完的。如果读者需要详细学习 Spring、Hibernate 相关知识，请参考疯狂 Java 体系的《轻量级 Java EE 企业应用实战》一书。

19.1 系统功能简介和架构设计

本章介绍的应用不再是简单的单机小应用，而是一个 Android 与传统服务器应用整合的应用，在这个应用中，服务器端程序依然保持了良好的应用架构，而客户端则使用 Android 程序充当。

19.1.1 系统功能简介

本章所介绍的系统是一个功能不太复杂的电子拍卖系统，本系统从实际电子商务平台上抽取，只取出其中部分核心功能实现，以作为示范应用，向读者展示一种良好的程序架构。

本章的电子拍卖系统其实就是一个电子商务平台，只要将该系统部署在互联网上，全球的客户都可以在该系统上发布想售出的商品，也可以对拍卖中的商品参与竞价。整个过程无须任何人工干预，由系统自动完成。

如果系统中提供与电子银行的接口，将可以通过电子银行的操作，实现买家对卖家的自动付款。一旦付款成功，就可以利用全球物流供应系统将拍卖物品发送到买家手中。可见，这种电子拍卖系统是一种开放式的、成本极其低廉的系统，大部分工作无须人工干预，系统自动完成管理。当然，由于本系统是一个示范系统，因此不提供与电子银行的接口，只是模拟了用户添加拍卖物品，用户参与拍卖的基本行为。拍卖结束后，系统会判断物品是否被最高竞价者获得。

该电子拍卖系统模拟了淘宝系统部分功能，抽取了实际电子拍卖系统部分功能，但没有提供如个人身份认证、信用管理等细节问题，本系统主要实现了电子拍卖系统中的核心功能。

本项目的服务器端是一个完整的 Java EE 项目，服务器端采用了控制器层、业务逻辑层、DAO 层的分层架构。Android 客户端应用只负责与服务器的控制器组件交互，Android 应用采用 Apache HttpClient 向服务器端的控制器发送请求并获取服务器响应，这样即可实现 Android 系统与电子拍卖系统之间的通信。

本系统要求用户参与拍卖之前，必须登录系统。本系统提供了系统登录验证，登录验证在服务器端通过 Filter 实现，Filter 拦截用户请求，并判断 Session 中是否保存了当前用户 ID，如果保存了用户 ID，即该用户已经登录；否则没有登录。

对于物品的管理，本系统可以查询拍卖物品，添加拍卖物品，增加物品种类，竞价处理，以及发送邮件通知用户所参与的竞价。

- 注册用户可以添加拍卖物品，添加物品种类。在添加之前必须登录系统。
- 注册用户可以浏览当前拍卖中的物品，以及流拍的物品。
- 注册用户可以参与竞价，参与的竞价系统将通过邮件通知用户。

19.1.2 系统架构设计

本系统的服务器采用 Java EE 的分层结构，分为视图层、控制器层、业务逻辑层和 DAO 层。分层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再由中间层与数据库交互。

中间层采用 Struts 2+Spring 4+Hibernate 4，为了分离控制层与业务逻辑层，又可细分为：

- 控制器层，就是 MVC 模式里面的“C”（Controller），负责表现层与业务逻辑层的交互，调用业务逻辑层，并将业务数据返回给表现层来显示。MVC 框架采用流行的 Struts 2。
- Service 层（业务逻辑层），负责实现业务逻辑，对 DAO 对象进行正面模式的封装。
- DAO 层（数据访问对象层），负责与持久化对象交互，封装了数据的增、删、查、改原子操作。
- Domain Object 层（持久化对象层），通过实体/关系映射工具将关系型数据库的数据映射成对象，实现以面向对象方式操作数据库。这个系统采用 Hibernate 作为 O/R Mapping 框架。

本系统使用 MySQL 数据库存放数据。

服务器端应用的总体架构图如图 19.1 所示。

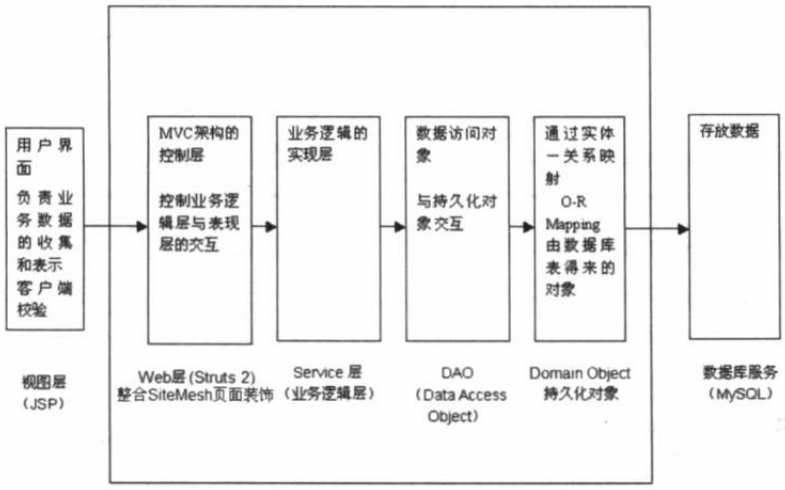


图 19.1 服务器端应用的总体架构图

当采用 Android 应用作为客户端时，Android 应用可以通过网络与服务器端交互，Android 应用将会通过 Apache HttpClient 向服务器端的控制器发送请求，并获取服务器响应，服务器响应将会采用 JSON 数据格式——可以更有效地进行数据交互。

Android 应用向服务器端的控制器发送请求，此处的控制器并不是 Struts 2 的 Action，而是直接采用 Servlet 充当。在 Servlet 3.1 规范中，开发 Servlet 已经不再需要在 web.xml 文件中进行配置了，因此开发起来十分方便。

图 19.2 显示了 Android 客户端与服务器整合的架构图。

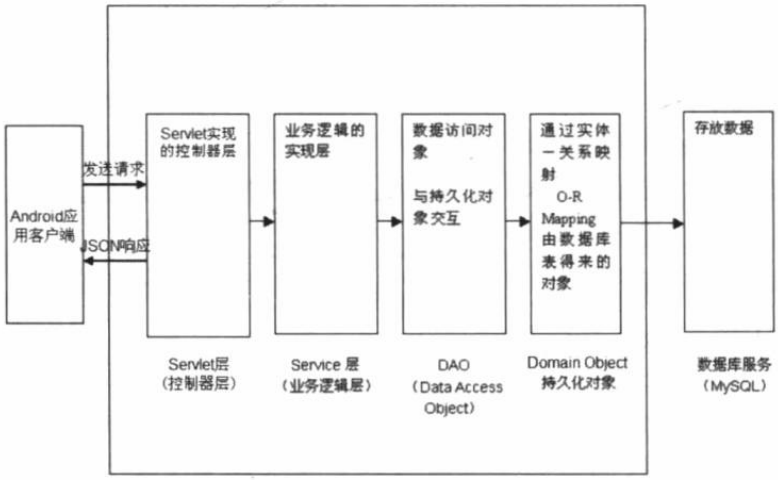


图 19.2 Android 客户端与服务器整合的架构图

细心的读者可能已经发现，图 19.1 与图 19.2 所示的结构十分相似，服务器端应用的结构基本不需要改变，只要在传统 Java EE 应用的基础上增加系列 Servlet，这些 Servlet 负责向 Android 客户端提供响应即可。

图 19.1 与图 19.2 十分相似正好说明了 Java EE 应用架构的优势：当整个应用的某一层需要改变或重构时，应用系统能最大限度地“复用”以前的应用组件，而不需要重新开发，这样才能保证以前编写的项目代码具有高度的保值性。也正是由于上面两个结构图的相似性，才可让我们非常快速地为该应用增加各种平台的客户端系统。

19.2 JSON 简介

Android 客户端与服务器端通信时需要一种合适的数据交换格式，本系统采用了 JSON 作为 Android 客户端与服务器端的数据交换格式。

提示：

实际上还可以考虑使用 Web Service 来作为 Android 应用与服务器端的通信技术，对于 Web Service 而言，底层采用 XML 作为数据交换格式。

JSON 的全称是 JavaScript Object Notation，即 JavaScript 对象符号，它是一种轻量级的数据交换格式。JSON 的数据交换格式既适合人来读/写，也适合计算机本身解析和生成。最早的时候，JSON 是 JavaScript 语言的数据交换格式，后来慢慢发展成一种语言无关的数据交换格式，这一点非常类似于 XML。

JSON 主要在类似于 C 的编程语言中广泛使用，这些语言包括 C、C++、C#、Java、JavaScript、Perl、Python 等。JSON 提供了多种语言之间完成数据交换的能力，因此，JSON 也是一种非常

理想的数据交换格式。JSON 主要有如下两种数据结构。

- 由 key-value 对组成的数据结构。这种数据结构在不同的语言中有不同的实现。例如，在 JavaScript 中是一个对象，在 Java 中是一种 Map 结构，在 C 语言中则是一个 struct，在其他语言中可能有 record、dictionary、hash table 等。
- 有序集合。这种数据结构在不同语言中可能有 list、vector、数组和序列等实现。

上面两种数据结构在不同的语言中都有对应的实现。因此，这种简便的数据表示方式完全可以实现跨语言，所以可以作为程序设计语言中通用的数据交换格式。在 JavaScript 中主要有两种 JSON 语法，其中一种用于创建对象；另一种用于创建数组。

19.2.1 使用 JSON 语法创建对象

使用 JSON 语法创建对象是一种更简单的方式，既可避免书写函数，也可避免使用 new 关键字，而是直接获取一个 JavaScript 对象。对于早期的 JavaScript 版本，如果要使用 JavaScript 创建一个对象，在通常情况下可能会这样写：

```
// 定义一个函数，同时也可作为类的构造器
function Person(name, sex)
{
    this.name = name;
    this.sex = sex;
}
// 创建一个 Person 实例
var p = new Person('yeeku', 'male');
// 输出 Person 实例
alert(p.name);
```

从 JavaScript 1.2 开始，创建对象有了一种更快捷的语法，语法如下：

```
var p = {"name": 'yeeku',
        "sex" : 'male'};
alert(p);
```

这种语法就是一种 JSON 语法。显然，使用 JSON 语法创建对象更加简捷、方便。图 19.3 显示了这种语法示意图。

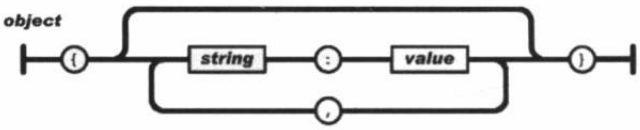


图 19.3 JSON 创建对象的语法示意图

从图 19.3 可以看出，创建对象 object 时，总以{开始，以}结束，对象的每个属性名和属性值之间以英文冒号（:）隔开，多个属性定义之间以英文逗号（,）隔开。语法格式如下：

```
object =
{
    propertyName1 : propertyValue1 ,
    propertyName2 : propertyValue2 ,
    ...
}
```

必须注意的是，并不是每个属性定义后面都有英文逗号（,），必须在后面还有属性定义时才需要逗号（,）。因此，下面的对象定义是错误的。

```
person =
{
    name: 'yeeku',
    sex: 'male',
}
```

因为 sex 属性定义后多出一个英文逗号，最后一个属性定义的后面应直接以 } 结束，不再有英文逗号（,）。

当然，使用 JSON 语法创建 JavaScript 对象时，属性值不仅可以是普通字符串，也可以是任何基本数据类型，还可以是函数、数组，甚至是另外一个 JSON 语法创建的对象。例如：

```
person =
{
    name: 'yeeku',
    sex : 'male',
    // 使用 JSON 语法为其指定一个属性
    son: {
        name:'nono',
          grade:1
    },
    // 使用 JSON 语法为 person 直接分配一个方法
    info : function()
    {
        document.writeln("姓名: " + this.name + "性别: " + this.sex);
    }
}
```

19.2.2 使用 JSON 语法创建数组

使用 JSON 语法创建数组也是非常常见的情形，在早期的 JavaScript 语法里，我们通过如下方式来创建数组。

```
// 创建数组对象
var a = new Array();
// 为数组元素赋值
a[0] = 'yeeku';
// 为数组元素赋值
a[1] = 'nono';
```

或者，通过如下方式创建数组：

```
// 创建数组对象时直接赋值
var a = new Array('yeeku', 'nono');
```

但如果我们使用 JSON 语法，则可以通过如下方式创建数组：

```
// 使用 JSON 语法创建数组
var a = ['yeeku', 'nono'];
```

如图 19.4 所示是 JSON 创建数组的语法示意图。

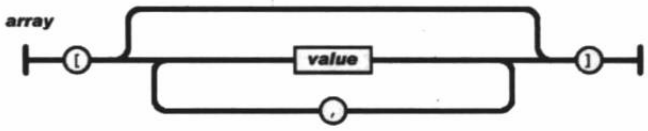


图 19.4 JSON 创建数组的语法示意图

正如从图 19.4 中所见到的，JSON 创建数组总是以英文方括号（[）开始，然后依次放入数组元素，元素与元素之间以英文逗号（,）隔开，最后一个数组元素后面不需要英文逗号，但以英文反方括号（]）结束。使用 JSON 创建数组的语法格式如下：

```
arr = [value1 , value 2 ...]
```

与使用 JSON 语法创建对象相似的是，数组的最后一个元素后面不能有英文逗号（,）。

鉴于 JSON 语法的简单易用，而且作为数据传输载体时，数据传输量更小，假设需要交换一个对象 person，其 name 属性为 yeeku，gender 属性为 male，age 属性为 29，使用 JSON 语

法可以简化为如下形式：

```
person =
{
    name:'yeeku',
    gender:'male',
    age:29
}
```

但如果使用 XML 数据交换格式，则需要采用如下格式：

```
<person>
    <name>yeeku</name>
    <gender>male</gender>
    <age>29</age>
</person>
```

对比这两种表示方式，第一种方式明显比第二种方式更加简洁，数据传输量也更小。

19.2.3 Java 的 JSON 支持

当服务器返回一个满足 JSON 格式的字符串后，我们可以利用 JSON 项目提供的工具类将该字符串转换为 JSON 对象或 JSON 数组。

幸运的是，Android 系统内置了对 JSON 的支持，在 Android SDK 的 org.json 包下提供了 JSONArray、JSONObject、JSONStringer 和 JSONExeception 等类，通过这些类即可非常方便地完成 JSON 字符串与 JSONArray、JSONObject 之间的相互转换。

JDK 默认并未提供对 JSON 的支持，不过我们可以登录 http://www.json.org/java/ index.html，在该页面可以看到 JSONArray、JSONObject、JSONStringer 和 JSONExeception 等类的源代码，读者可以自行下载这些源代码，把它们加入项目中即可使用。

提示：

笔者已经下载了这些源代码，并把它们编译成了*.class 文件，将它们打包成了 json.jar 包，该 JAR 包位于光盘的 codes\19\auction\WEB-INF\lib 目录下。读者直接把该 JAR 包添加到项目的类加载路径下即可使用。

Java 的 JSON 支持主要依赖于 JSONArray 和 JSONObject 两个类，其中：

- JSONArray 代表一个 JSON 数组，它可完成 Java 集合（集合元素可以是对象）与 JSON 字符串之间的相互转换。
- JSONObject 代表一个 JSON 对象，它可完成 Java 对象与 JSON 字符串之间的相互转换。

19.3 发送请求的工具类

本系统采用 Apache HttpClient 与远程服务器通信。为了简化 HttpClient 的用法，本系统定义了一个工具类对 HttpClient 进行封装。该工具类定义了如下两个方法来发送请求。

- getRequest()：发送 GET 请求。
- postRequest()：发送 POST 请求。

该工具类的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\util\HttpUtil.java

```
public class HttpUtil
{
    // 创建 HttpClient 对象
```

```
public static HttpClient httpClient = new DefaultHttpClient();
public static final String BASE_URL =
    "http://192.168.1.88:8888/auction/android/";
/**
 *
 * @param url 发送请求的URL
 * @return 服务器响应字符串
 * @throws Exception
 */
public static String getRequest(final String url)
    throws Exception
{
    FutureTask<String> task = new FutureTask<String>(
    new Callable<String>()
    {
        @Override
        public String call() throws Exception
        {
            // 创建HttpGet对象。
            HttpGet get = new HttpGet(url);
            // 发送GET请求
            HttpResponse httpResponse = httpClient.execute(get);
            // 如果服务器成功地返回响应
            if (httpResponse.getStatusLine()
                .getStatusCode() == 200)
            {
                // 获取服务器响应字符串
                String result = EntityUtils
                    .toString(httpResponse.getEntity());
                return result;
            }
            return null;
        }
    });
    new Thread(task).start();
    return task.get();
}
/**
 * @param url 发送请求的URL
 * @param params 请求参数
 * @return 服务器响应字符串
 * @throws Exception
 */
public static String postRequest(final String url
    , final Map<String ,String> rawParams)throws Exception
{
    FutureTask<String> task = new FutureTask<String>(
    new Callable<String>()
    {
        @Override
        public String call() throws Exception
        {
            // 创建HttpPost对象。
            HttpPost post = new HttpPost(url);
            // 如果传递参数个数比较多，可以对传递的参数进行封装
            List<NameValuePair> params =
                new ArrayList<NameValuePair>();
            for(String key : rawParams.keySet())
            {
                // 封装请求参数
                params.add(new BasicNameValuePair(key
                    , rawParams.get(key)));
            }
```

```
                // 设置请求参数
                post.setEntity(new UrlEncodedFormEntity(
                    params, "gbk"));
                // 发送 POST 请求
                HttpResponse httpResponse = httpClient.execute(post);
                // 如果服务器成功地返回响应
                if (httpResponse.getStatusLine()
                    .getStatusCode() == 200)
                {
                    // 获取服务器响应字符串
                    String result = EntityUtils
                        .toString(httpResponse.getEntity());
                    return result;
                }
                return null;
            }
        });
        new Thread(task).start();
        return task.get();
    }
}
```

上面的 HttpUtil 中两行粗体字代码定义了 getRuquest()和 postRequest()两个方法，这两个方法用于向服务器发送请求，返回服务器的响应。在 HttpUtil 中提供这两个方法之后，接下来在 Android 应用中只要调用这两个方法即可实现与服务器的通信。

19.4 用户登录

在使用该电子拍卖系统之前，用户必须先登录系统。用户在 Android 应用中输入用户名、密码，单击“登录”按钮，程序即可通过 HttpUtil 向服务器发送请求，通过服务器来验证用户所输入的用户名、密码是否正确。

19.4.1 处理登录的 Servlet

处理用户登录的 Servlet 只是前端控制器，它的作用只有三个。

- 获取请求参数。
- 调用业务逻辑组件的方法来处理用户请求。
- 根据处理结果来生成输出。

由于本项目的服务器组件都是部署在 Spring 容器中的，因此程序需要获取 Spring 容器中的业务逻辑组件，而不是自行创建业务逻辑组件。

由于所有 Servlet 都需要获取 Spring 容器中的业务逻辑组件，因此本程序提供了一个 Servlet 基类，该基类中定义了一个方法来获取 Web 应用中的 Spring 容器。该 Servlet 基类的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\base\BaseServlet.java

```
public class BaseServlet extends HttpServlet
{
    private ApplicationContext ctx;
    public void init(ServletConfig config)
        throws ServletException
    {
        super.init(config);
        // 获取 Web 应用中的 ApplicationContext 实例
        ctx = WebApplicationContextUtils
            .getWebApplicationContext(getServletContext());
    }
```

```
    // 返回 Web 应用中的 Spring 容器
    public ApplicationContext getCtx()
    {
        return this.ctx;
    }
}
```

正如从上面的粗体字代码所看到的，该 Servlet 基类中定义了一个 getCtx()方法，通过该方法即可获取 Spring 容器，这意味着应用中的所有 Servlet 都可通过该方法来获取 Spring 容器。接下来服务器端的所有应用程序只要继承该 Servlet，并通过该方法来获取 Spring 容器即可。

下面是处理用户登录的 Servlet 类代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\LoginServlet.java

```
@WebServlet(urlPatterns="/android/login.jsp")
public class LoginServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        String user = request.getParameter("user");
        String pass = request.getParameter("pass");
        // 获取系统的业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 验证用户登录
        int userId = auctionManager.validLogin(user , pass);
        response.setContentType("text/html; charset=GBK");
        // 登录成功
        if (userId > 0)
        {
            request.getSession(true).setAttribute("userId" , userId);
        }
        try
        {
            // 把验证的 userId 封装成 JSONObject
            JSONObject jsonObj = new JSONObject()
                .put("userId" , userId);
            // 输出响应
            response.getWriter().println(jsonObj.toString());
        }
        catch (JSONException ex)
        {
            ex.printStackTrace();
        }
    }
}
```

该 Servlet 中第一行粗体字代码使用@WebServlet 指定了该 Servlet 的 URL 为/android/login.jsp。第二行粗体字代码调用了业务逻辑组件的 validateLogin()方法来处理用户登录。如果用户登录成功，程序将用户的 ID 放入 HTTP session 中，方便程序跟踪用户的登录状态。

如果读者对 Spring、Hibernate 等框架不熟悉也不要紧，可以直接使用 LoginServlet 访问系统数据库，并根据用户输入的用户名、密码返回该用户的 ID，该 Android 客户端一样可以正常工作。只是采用这种方式开发的应用不太正规，后期的扩展性很差而已。

➤➤ 19.4.2 用户登录客户端

Android 客户端中用户登录界面有两个文本框，用于接收用户输入的用户名、密码信息。

下面是用户登录的界面布局 XML 文档。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="300dp"
    android:layout_height="match_parent"
    android:layout_gravity="center_horizontal"
    android:stretchColumns="1">
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitCenter"
    android:src="@drawable/logo"
    android:contentDescription="@string/hello"/>
<TextView
    android:text="@string/welcome"
    android:id="@+id/TextView"
    android:textSize="@dimen/label_font_size"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:padding="@dimen/title_padding"/>
<!-- 输入用户名的行 -->
<TableRow>
<TextView
    android:text="@string/user_name"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<EditText
    android:id="@+id/userEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="text"/>
</TableRow>
<!-- 输入密码的行 -->
<TableRow>
<TextView
    android:text="@string/user_pass"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<EditText
    android:text=""
    android:id="@+id/pwdEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPassword"/>
</TableRow>
<!-- 定义登录、取消按钮的行 -->
<LinearLayout android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center">
<Button
    android:id="@+id/bnLogin"
    android:text="@string/login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<Button
    android:id="@+id/bnCancel"
    android:text="@string/cancel"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>
</TableLayout>
```

该界面布局的效果如图 19.5 所示。

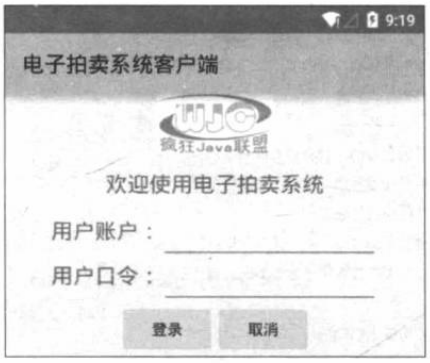


图 19.5 用户登录界面

在图 19.5 所示的界面中输入用户名、密码之后，如果用户单击“登录”按钮将会激发登录处理，也就是通过 HttpUtil 向服务器发送请求。用户登录的 Activity 代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\Login.java

```
public class Login extends Activity
{
    // 定义界面中的两个文本框
    EditText etName, etPass;
    // 定义界面中的两个按钮
    Button bnLogin, bnCancel;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        // 获取界面中的两个编辑框
        etName = (EditText) findViewById(R.id.userEditText);
        etPass = (EditText) findViewById(R.id.pwdEditText);
        // 获取界面中的两个按钮
        bnLogin = (Button) findViewById(R.id.bnLogin);
        bnCancel = (Button) findViewById(R.id.bnCancel);
        // 为bnCancal 按钮的单击事件绑定事件监听器
        bnCancel.setOnClickListener(new HomeListener(this));
        bnLogin.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 执行输入校验
                if (validate()) // ①
                {
                    // 如果登录成功
                    if (loginPro()) // ②
                    {
                        // 启动 Main Activity
                        Intent intent = new Intent(Login.this
                            , AuctionClientActivity.class);
                        startActivity(intent);
                        // 结束该 Activity
                        finish();
                    }
```

```
                else
                {
                    DialogUtil.showDialog(Login.this
                        , "用户名称或者密码错误，请重新输入！", false);
                }
            }
        }
    });
}
private boolean loginPro()
{
    // 获取用户输入的用户名、密码
    String username = etName.getText().toString();
    String pwd = etPass.getText().toString();
    JSONObject jsonObj;
    try
    {
        jsonObj = query(username, pwd);
        // 如果 userId 大于 0
        if (jsonObj.getInt("userId") > 0)
        {
            return true;
        }
    }
    catch (Exception e)
    {
        DialogUtil.showDialog(this
            , "服务器响应异常，请稍后再试！", false);
        e.printStackTrace();
    }
    return false;
}
// 对用户输入的用户名、密码进行校验
private boolean validate()
{
    String username = etName.getText().toString().trim();
    if (username.equals(""))
    {
        DialogUtil.showDialog(this, "用户账户是必填项！", false);
        return false;
    }
    String pwd = etPass.getText().toString().trim();
    if (pwd.equals(""))
    {
        DialogUtil.showDialog(this, "用户口令是必填项！", false);
        return false;
    }
    return true;
}
// 定义发送请求的方法
private JSONObject query(String username, String password)
    throws Exception
{
    // 使用 Map 封装请求参数
    Map<String, String> map = new HashMap<>();
    map.put("user", username);
    map.put("pass", password);
    // 定义发送请求的 URL
    String url = HttpUtil.BASE_URL + "login.jsp";
    // 发送请求
    return new JSONObject(HttpUtil.postRequest(url, map));
}
}
```

上面 Activity 的 query()方法中两行粗体字代码用于向指定 URL 发送请求，并将服务器响应封装成 JSONObject。

上面的程序为登录按钮的单击事件绑定了事件监听器，当用户单击“登录”按钮时，程序先执行输入校验，如①号粗体字代码所示；接下来执行登录处理，如②号粗体字代码所示——程序调用了 loginPro 来处理用户的登录请求。

如果用户输入的用户名、密码不正确，系统将会调用 DialogUtil 来显示对话框，对话框提示登录失败。由于本系统经常需要显示各种对话框，因此程序专门把它定义成一个独立的类。DialogUtil 类的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\util\DialogUtil.java

```
public class DialogUtil
{
    // 定义一个显示消息的对话框
    public static void showDialog(final Context ctx
        , String msg , boolean goHome)
    {
        // 创建一个 AlertDialog.Builder 对象
        AlertDialog.Builder builder = new AlertDialog.Builder(ctx)
            .setMessage(msg).setCancelable(false);
        if(goHome)
        {
            builder.setPositiveButton("确定", new OnClickListener()
            {
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                    Intent i = new Intent(ctx , AuctionClientActivity.class);
                    i.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                    ctx.startActivity(i);
                }
            });
        }
        else
        {
            builder.setPositiveButton("确定", null);
        }
        builder.create().show();
    }
    // 定义一个显示指定组件的对话框
    public static void showDialog(Context ctx , View view)
    {
        new AlertDialog.Builder(ctx)
            .setView(view).setCancelable(false)
            .setPositiveButton("确定", null)
            .create()
            .show();
    }
}
```

如果登录成功，系统启动 AuctionClientActivity，这个 Activity 相当于系统主界面，用户可通过该界面提供的 ListView 进入各功能。

AuctionClientActivity 的界面布局文件可分为两个：为普通手机屏幕提供的应用界面和为平板电脑屏幕提供的界面。下面先看为普通手机屏幕提供的界面布局文件。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\activity_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="match_parent"
    android:layout_height="match_parent">
    <!-- 添加一个 Fragment -->
    <fragment android:name="org.crazyit.auction.client.AuctionListFragment"
        android:id="@+id/auction_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

上面程序界面的主要组件就是 AuctionListFragment，该 Fragment 显示一个 ListView，该 ListView 中每个列表项代表一个系统功能。

下面再看为平板电脑屏幕提供的界面布局文件。

程序清单：codes\19\AuctionClient\app\src\main\res\layout-sw480dp\activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个水平排列的 LinearLayout，并指定使用中等分隔条 -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:divider="?android:attr/dividerHorizontal"
    android:showDividers="middle">
    <!-- 添加一个 Fragment -->
    <fragment android:name="org.crazyit.auction.client.AuctionListFragment"
        android:id="@+id/auction_list"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />
    <!-- 添加一个 FrameLayout 容器 -->
    <FrameLayout
        android:id="@+id/auction_detail_container"
        android:layout_width="0dp"
        android:paddingLeft="10dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />
</LinearLayout>
```

上面的程序界面中同样包含了一个 AuctionListFragment。除此之外，该程序界面中还包含了一个 FrameLayout 容器，该容器负责装载对应功能的 Fragment 组件。

由于 layout 和 layout-sw480dp 目录下分别包含了 activity_main.xml 界面布局文件，因此 AuctionClientActivity 会根据屏幕尺寸自动加载不同目录下的界面布局文件。除此之外，该 Activity 必须根据界面布局来进行处理：如果程序加载 layout 目录下的 activity_main.xml 布局文件，用户点击功能项时，系统将会启动相应的 Activity 来显示功能；如果程序加载 layout-sw480dp 目录下的 activity_main.xml 布局文件，用户点击功能项时，系统将会使用 FrameLayout 加载相应功能的 Fragment。

AuctionClientActivity 的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AuctionClientActivity.java

```
public class AuctionClientActivity extends Activity
    implements Callbacks
{
    // 定义一个旗标，用于标识该应用是否支持大屏幕
    private boolean mTwoPane;
```

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // 指定加载R.layout.activity_main对应的界面布局文件
    // 但实际上该应用会根据屏幕分辨率加载不同的界面布局文件
    setContentView(R.layout.activity_main);
    // 如果加载的界面布局文件中包含ID为auction_detail_container的组件
    if (findViewById(R.id.auction_detail_container) != null)
    {
        mTwoPane = true;
        ((AuctionListFragment) getFragmentManager()
            .findFragmentById(R.id.auction_list))
            .setActivateOnItemClick(true);
    }
}
@Override
public void onItemSelected(Integer id , Bundle bundle)
{
    if (mTwoPane)
    {
        Fragment fragment = null;
        switch ((int) id)
        {
            // 查看竞得物品
            case 0:
                // 创建ViewItemFragment
                fragment = new ViewItemFragment();
                // 创建Bundle，准备向Fragment传入参数
                Bundle arguments = new Bundle();
                arguments.putString("action", "viewSucc.jsp");
                // 向Fragment传入参数
                fragment.setArguments(arguments);
                break;
            // 浏览流拍物品
            case 1:
                // 创建ViewItemFragment
                fragment = new ViewItemFragment();
                // 创建Bundle，准备向Fragment传入参数
                Bundle arguments2 = new Bundle();
                arguments2.putString("action", "viewFail.jsp");
                // 向Fragment传入参数
                fragment.setArguments(arguments2);
                break;
            // 管理物品种类
            case 2:
                // 创建ManageKindFragment
                fragment = new ManageKindFragment();
                break;
            // 管理物品
            case 3:
                // 创建ManageItemFragment
                fragment = new ManageItemFragment();
                break;
            // 浏览拍卖物品（选择物品种类）
            case 4:
                // 创建ChooseKindFragment
                fragment = new ChooseKindFragment();
                break;
            // 查看自己的竞标
            case 5:
                // 创建ViewBidFragment
```

```
                fragment = new ViewBidFragment();
                break;
            case ManageItemFragment.ADD_ITEM:
                fragment = new AddItemFragment();
                break;
            case ManageKindFragment.ADD_KIND:
                fragment = new AddKindFragment();
                break;
            case ChooseKindFragment.CHOOSE_ITEM:
                fragment = new ChooseItemFragment();
                Bundle args = new Bundle();
                args.putLong("kindId", bundle.getLong("kindId"));
                fragment.setArguments(args);
                break;
            case ChooseItemFragment.ADD_BID:
                fragment = new AddBidFragment();
                Bundle args2 = new Bundle();
                args2.putInt("itemId", bundle.getInt("itemId"));
                fragment.setArguments(args2);
                break;
        }
        // 使用 fragment 替换 auction_detail_container 容器当前显示的 Fragment
        getFragmentManager().beginTransaction()
            .replace(R.id.auction_detail_container, fragment)
            .addToBackStack(null).commit();
    }
    else
    {
        Intent intent = null;
        switch ((int) id)
        {
            // 查看竞得物品
            case 0:
                // 启动 ViewItem Activity
                intent = new Intent(this, ViewItem.class);
                // action 属性为请求的 Servlet 地址
                intent.putExtra("action", "viewSucc.jsp");
                startActivity(intent);
                break;
            // 浏览流拍物品
            case 1:
                // 启动 ViewItem Activity
                intent = new Intent(this, ViewItem.class);
                // action 属性为请求的 Servlet 的 URL
                intent.putExtra("action", "viewFail.jsp");
                startActivity(intent);
                break;
            // 管理物品种类
            case 2:
                // 启动 ManageKind Activity
                intent = new Intent(this, ManageKind.class);
                startActivity(intent);
                break;
            // 管理物品
            case 3:
                // 启动 ManageItem Activity
                intent = new Intent(this, ManageItem.class);
                startActivity(intent);
                break;
            // 浏览拍卖物品（选择物品种类）
            case 4:
                // 启动 ChooseKind Activity
                intent = new Intent(this, ChooseKind.class);
```

```
                    startActivity(intent);
                    break;
                // 查看自己的竞标
                case 5:
                    // 启动ViewBid Activity
                    intent = new Intent(this, ViewBid.class);
                    startActivity(intent);
                    break;
            }
        }
    }
}
```

从上面的代码可以看出，如果在手机上运行，AuctionClientActivity 主要提供一个 ListView，当用户单击不同的列表项时，程序将会启动不同的 Activity。AuctionClientActivity 的运行效果如图 19.6 所示。

用户单击图 19.6 所示的“主菜单”（其实是 ListView）的任一个列表项，系统将会启动对应的 Activity，从而允许用户执行相应的操作。

但如果在平板电脑上运行，AuctionClientActivity 则提供一个 ListView 和一个 FrameLayout 容器，当用户单击不同的列表项时，程序将使用 FrameLayout 装载相应功能的 Fragment。在平板电脑上运行 AuctionClientActivity 的效果如图 19.7 所示。

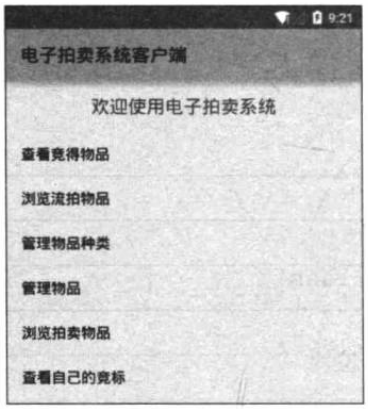


图 19.6　系统主菜单（手机）

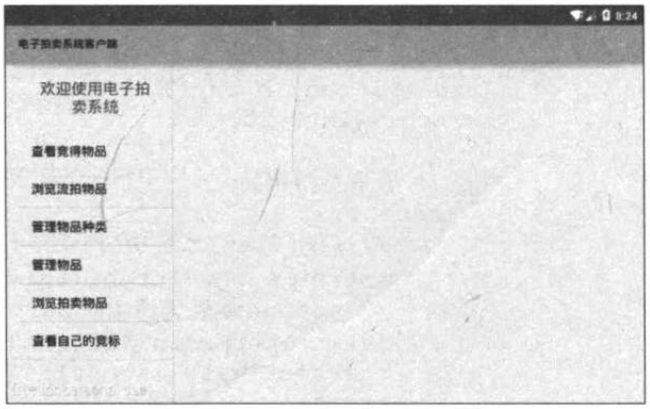


图 19.7　系统主界面（平板电脑）

用户单击图 19.7 所示的“主菜单”（其实是 ListView）的任一个列表项，系统将会使用右边的 FrameLayout 装载相应功能的 Fragment。

19.5　查看流拍物品

当用户单击“浏览流拍物品”时，程序将使用 FrameLayout 装载相应的 Fragment，启动相应的 Activity 装载相应的 Fragment，并显示系统中的流拍物品。

19.5.1　查看流拍物品的 Servlet

查看流拍物品的 Servlet 也只是前端控制器，它会调用 Spring 容器中业务逻辑组件的方法，通过该方法获取系统中的流拍物品。

该 Servlet 类的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewFailServlet.java

```
@WebServlet(urlPatterns="/android/viewFail.jsp")
public class ViewFailServlet extends BaseServlet
{
```

```
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        // 获取业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 获取系统内所有流拍的物品
        List<ItemBean> items = auctionManager.getFailItems();
        JSONArray jsonArr = new JSONArray(items);
        response.setContentType("text/html; charset=GBK");
        response.getWriter().println(jsonArr.toString()); // ①
    }
}
```

上面程序中的粗体字代码调用了 Spring 容器中业务逻辑组件的方法来获取系统中的流拍物品。程序的第二行粗体字代码把 Java 集合包装成 JSONArray 对象，程序中①号粗体字代码用于把 JSONArray 对象转换为 JSON 字符串，并作为响应输出到客户端。

如果直接向服务器的/android/viewFail.jsp 发送请求，将可以看到如图 19.8 所示的输出。

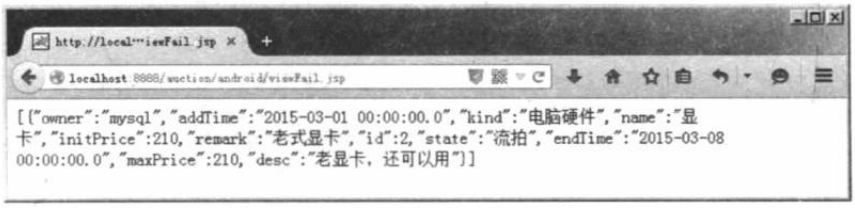


图 19.8 浏览流拍物品的 JSON 响应

图 19.8 显示的字符串就是典型的 JSON 格式字符串，Android 客户端程序只要调用 JSONArray 类的构造器即可把它转换为 JSONArray——就是一个 JSON 数组。

提示： 与前面登录的 Servlet 类似，如果读者对 Spring、Hibernate 等技术不熟，也可以直接在 Servlet 中访问数据库，并获取系统中所有的流拍物品。只要该 Servlet 能获取到所有的流拍物品，并把它们转换为图 19.8 所示的 JSON 字符串输出，该 Android 客户端就可正常工作。

19.5.2 查看流拍物品客户端

下面是查看流拍物品的程序界面布局代码。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\view_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/sub_title_margin">
<TextView
    android:id="@+id/view_titile"
    android:text="@string/view_succ"
    android:textSize="@dimen/label_font_size"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <!-- 定义返回按钮 -->
    <Button
        android:id="@+id/bn_home"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="@dimen/label_font_size"
        android:background="@drawable/home"/>
    </LinearLayout>
    <!-- 查看物品列表的 ListView -->
    <ListView
        android:id="@+id/succList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
    </LinearLayout>
```

上面界面布局中的主要组件就是一个 ListView，用于显示多个物品。查看流拍物品的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ViewItemFragment.java

```
public class ViewItemFragment extends Fragment
{
    Button bnHome;
    ListView succList;
    TextView viewTitle;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.view_item
            , container , false);
        // 获取界面上的“返回”按钮
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        succList = (ListView) rootView.findViewById(R.id.succList);
        viewTitle = (TextView) rootView.findViewById(R.id.view_titile);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        String action = getArguments().getString("action");
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + action;
        // 如果是查看流拍物品，修改标题
        if (action.equals("viewFail.jsp"))
        {
            viewTitle.setText(R.string.view_fail);
        }
        try
        {
            // 向指定 URL 发送请求，并把服务器响应转换成 JSONArray 对象
            JSONArray jsonArray = new JSONArray(HttpUtil
                .getRequest(url)); // ①
            // 将 JSONArray 包装成 Adapter
            JSONArrayAdapter adapter = new JSONArrayAdapter(getActivity()
                , jsonArray, "name", true); // ②
            succList.setAdapter(adapter);
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity(), "服务器响应异常，请稍后再试！",
                false);
            e.printStackTrace();
        }
```

```
        succList.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                // 查看指定物品的详细情况
                viewItemDetail(position); // ③
            }
        });
        return rootView;
    }
    private void viewItemDetail(int position)
    {
        // 加载 detail.xml 界面布局代表的视图
        View detailView = getActivity().getLayoutInflater()
            .inflate(R.layout.detail, null);
        // 获取 detail.xml 界面布局中的文本框
        TextView itemName = (TextView) detailView
            .findViewById(R.id.itemName);
        TextView itemKind = (TextView) detailView
            .findViewById(R.id.itemKind);
        TextView maxPrice = (TextView) detailView
            .findViewById(R.id.maxPrice);
        TextView itemRemark = (TextView) detailView
            .findViewById(R.id.itemRemark);
        // 获取被单击的列表项
        JSONObject jsonObj = (JSONObject) succList.getAdapter().getItem(
            position);
        try
        {
            // 通过文本框显示物品详情
            itemName.setText(jsonObj.getString("name"));
            itemKind.setText(jsonObj.getString("kind"));
            maxPrice.setText(jsonObj.getString("maxPrice"));
            itemRemark.setText(jsonObj.getString("desc"));
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        DialogUtil.showDialog(getActivity(), detailView);
    }
}
```

上面程序中的①号粗体字代码用于向指定 URL 发送请求，并把服务器响应包装成 JSONArray 对象，这就完成了 Android 客户端与服务器的交互。JSONArray 对象的本质就是一个数组，它提供了如下常用方法。

- length()：返回该 JSON 数组的长度。
- optJSONObject(int index)：获取指定索引处的 JSONObject 对象。
- optJSONArray(int index)：获取指定索引处的 JSONArray 对象。
- optXxx(int index)：获取指定索引处的数组元素。

对于开发者而言，当程序中获取到 JSONArray 对象之后，完全可以把它当成数组处理。

本程序提供了一个 JSONArrayAdapter 类，它的本质是一个 Adapter，该 Adapter 用于对 JSON 数组进行包装，并作为 ListView 的内容 Adapter。JSONArrayAdapter 类的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\JSONArrayAdapter.java

```
public class JSONArrayAdapter extends BaseAdapter
{
```

```
    private Context ctx;
    // 定义需要包装的 JSONArray 对象
    private JSONArray jsonArray;
    // 定义列表项显示 JSONObject 对象的哪个属性
    private String property;
    private boolean hasIcon;
    public JSONArrayAdapter(Context ctx
        , JSONArray jsonArray, String property
        , boolean hasIcon)
    {
        this.ctx = ctx;
        this.jsonArray = jsonArray;
        this.property = property;
        this.hasIcon = hasIcon;
    }
    @Override
    public int getCount()
    {
        return jsonArray.length();
    }
    @Override
    public Object getItem(int position)
    {
        return jsonArray.optJSONObject(position);
    }
    @Override
    public long getItemId(int position)
    {
        try
        {
            // 返回物品的 ID
            return ((JSONObject)getItem(position)).getInt("id");
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        return 0;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        // 定义一个线性布局管理器
        LinearLayout linear = new LinearLayout(ctx);
        // 设置为水平的线性布局管理器
        linear.setOrientation(0);
        // 创建一个 ImageView
        ImageView iv = new ImageView(ctx);
        iv.setPadding(10, 0, 20, 0);
        iv.setImageResource(R.drawable.item);
        // 将图片添加到 LinearLayout 中
        linear.addView(iv);
        // 创建一个 TextView
        TextView tv = new TextView(ctx);
        try
        {
            // 获取 JSONArray 数组元素的 property 属性
            String itemName = ((JSONObject)getItem(position))
                .getString(property);
            // 设置 TextView 所显示的内容
            tv.setText(itemName);
        }
        catch (JSONException e)
```

```
            {
                e.printStackTrace();
            }
            tv.setTextSize(20);
            if (hasIcon)
            {
                // 将 TextView 添加到 LinearLayout 中
                linear.addView(tv);
                return linear;
            }
            else
            {
                tv.setTextColor(Color.BLACK);
                return tv;
            }
        }
    }
```

上面的 Adapter 负责为 ListView 提供列表项，ListView 的列表项可以是如下两种情况。

- 当 ListView 的列表项有图标时，该 Adapter 所提供的每个列表项是一个 LinearLayout（里面包括一个图标和一个 TextView）。
- 当 ListView 的列表项没有图标时，该 Adapter 提供的每个列表项就是一个普通的 TextView。

上面的 JSONArrayAdapter 可以多次复用，因此程序把它单独定义成一个 Adapter 类，这样可以多次使用该 Adapter 来封装 JSONArray 对象。

在平板电脑上查看流拍物品，将看到如图 19.9 所示的界面。

从图 19.9 所示的列表中只能看到流拍物品的物品名，如果用户希望看到该物品的详情，则可以单击该物品对应的列表项，此时程序将会触发 ViewItemFragment 中的 viewItemDetail(position)方法，如 ViewItemFragment 类中的③号粗体字代码所示。

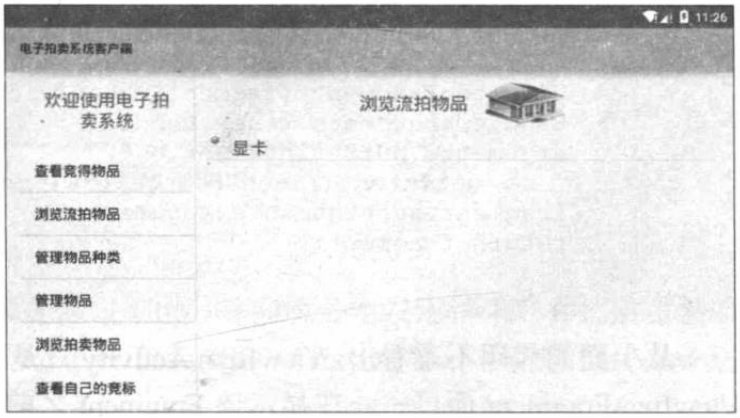


图 19.9 在平板电脑上查看流拍物品

viewItemDetail(position) 方法会使用对话框加载系统中的 detail.xml 界面布局文件，并显示该流拍物品的详情。当用户单击图 19.9 所示的列表项时，系统将会显示如图 19.10 所示的对话框。

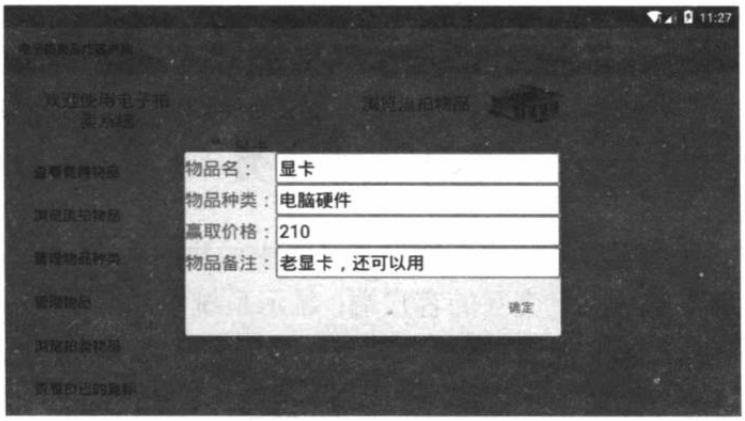


图 19.10 查看物品详情

上面的 ViewItemFragment 除了用于显示流拍物品之外，也可用于查看竞得物品，因为两个功能的前端实现基本是一致的，只是 Android 需要调用的服务器端方法不同而已。

如果在手机上使用，系统还需要使用一个 Activity 来“盛装”该 Fragment。由于该系统中有大量 Fragment 需要使用 Activity 来显示，因此本系统专门开发了一个

FragmentActivity，它只是用于显示一个 Fragment。该 Activity 的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\app\base\FragmentActivity.java

```
public abstract class FragmentActivity extends Activity
{
    private static final int ROOT_CONTAINER_ID = 0x90001;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        setContentView(layout);
        layout.setId(ROOT_CONTAINER_ID);
        getFragmentManager().beginTransaction()
            .replace(ROOT_CONTAINER_ID , getFragment())
            .commit();
    }
    protected abstract Fragment getFragment();
}
```

上面的粗体字代码定义了一个抽象方法，该抽象方法将会返回一个 Fragment，而该 Activity 的作用就是加载、显示该 Fragment。

FragmentActivity 用于被其他 Activity 继承，继承它的 Activity 只要重写 getFragment()方法即可。下面是 ViewItem Activity 的代码。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ViewItem.java

```
public class ViewItem extends FragmentActivity
{
    // 重写 getFragment()方法，该 Activity 显示该方法返回的 Fragment
    @Override
    protected Fragment getFragment()
    {
        ViewItemFragment fragment = new ViewItemFragment();
        Bundle arguments = new Bundle();
        arguments.putString("action"
            , getIntent().getStringExtra("action"));
        fragment.setArguments(arguments);
        return fragment;
    }
}
```

从上面的代码不难看出，ViewItem Activity 只是显示 ViewItemFragment 而已——在显示该 Fragment 之前，向该 Fragment 传入参数。

在普通手机上查看流拍物品，将可以看到如图 19.11 所示的界面。

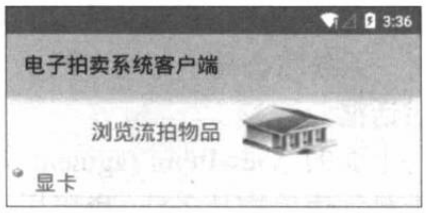


图 19.11 在手机上查看流拍物品

19.6 管理物品种类

管理物品种类包括浏览系统中的物品种类、添加物品种类两大主要功能。Android 客户端主要充当用户交互的客户端：显示系统中物品种类；添加种类时提供输入框供用户输入种类名称、种类描述等必要信息。

19.6.1 浏览物品种类的 Servlet

浏览物品种类的 Servlet 同样只是充当控制器，调用业务逻辑组件的业务方法来获取系统

中的物品种类。该 Servlet 的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewKindServlet.java

```
@WebServlet(urlPatterns="/android/viewKind.jsp")
public class ViewKindServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        // 获取业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 获取系统中所有物品种类
        List<KindBean> kinds = auctionManager.getAllKind();
        // 将所有物品种类包装成 JSONArray
        JSONArray jsonArr= new JSONArray(kinds);
        response.setContentType("text/html; charset=GBK");
        // 将 JSONArray 转换成 JSON 字符串后输出到客户端
        response.getWriter().println(jsonArr.toString());
    }
}
```

与前一个 Servlet 的实现基本相似，该 Servlet 也是调用 Spring 容器中业务逻辑组件相应的业务方法，并把方法返回的字符串包装成 JSONArray，再把 JSONArray 转换为字符串作为服务器响应。

直接使用浏览器浏览上面的 Servlet，将可以看到如图 19.12 所示的 JSON 字符串输出。

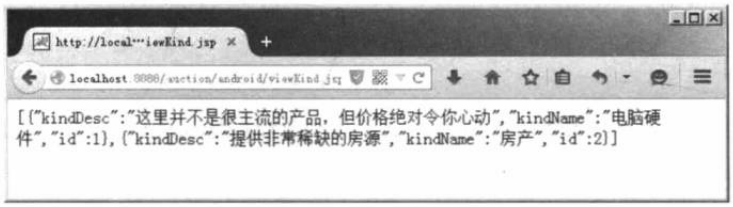


图 19.12 JSON 字符串响应

19.6.2 查看物品种类

在 Android 客户端查看物品种类，只要先通过 HttpUtil 向服务器发送请求，并把服务器响应字符串转换成 JSONArray 对象，再使用 Adapter 包装 JSONArray 对象，并使用 ListView 进行显示即可。

查看物品种类的界面布局代码如下。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\manage_kind.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/sub_title_margin">
<LinearLayout
    android:orientation="vertical"
```

```
    android:gravity="center"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
<TextView
    android:text="@string/manage_kind"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<!-- 添加种类的按钮 -->
<Button
    android:id="@+id/bnAdd"
    android:layout_width="85dp"
    android:layout_height="30dp"
    android:background="@drawable/add_kind"/>
</LinearLayout>
<Button
    android:id="@+id/bn_home"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="@dimen/label_font_size"
    android:background="@drawable/home"/>
</LinearLayout>
<!-- 显示种类列表的 ListView -->
<ListView
    android:id="@+id/kindList"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
```

上面的界面布局中定义了一个 ListView 来显示系统中的物品种类，程序只要把服务器返回的物品种类包装成 Adapter，并使用该 ListView 来显示所有的物品种类即可。下面是显示物品种类的 Fragment 代码。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageKindFragment.java

```
public class ManageKindFragment extends Fragment
{
    public static final int ADD_KIND = 0x1007;
    Button bnHome , bnAdd;
    ListView kindList;
    Callbacks mCallbacks;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.manage_kind
            , container , false);
        // 获取界面布局中的两个按钮
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        bnAdd = (Button) rootView.findViewById(R.id.bnAdd);
        kindList = (ListView) rootView.findViewById(R.id.kindList);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        // 为“添加”按钮的单击事件绑定事件监听器
        bnAdd.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 当“添加”按钮被单击时
                // 调用该 Fragment 所在 Activity 的 onItemSelected 方法
                mCallbacks.onItemSelected(ADD_KIND , null);
            }
```

```
        });
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "viewKind.jsp";
        try
        {
            // 向指定 URL 发送请求，并把响应包装成 JSONArray 对象
            final JSONArray jsonArray = new JSONArray(
                HttpUtil.getRequest(url));
            // 把 JSONArray 对象包装成 Adapter
            kindList.setAdapter(new KindArrayAdapter(jsonArray
                , getActivity()));
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity()
                , "服务器响应异常，请稍后再试！" ,false);
            e.printStackTrace();
        }
        return rootView;
    }
    // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 如果 Activity 没有实现 Callbacks 接口，抛出异常
        if (!(activity instanceof Callbacks))
        {
            throw new IllegalStateException(
                "ManageKindFragment 所在的 Activity 必须实现 Callbacks 接口!");
        }
        // 把该 Activity 当成 Callbacks 对象
        mCallbacks = (Callbacks) activity;
    }
    // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
    @Override
    public void onDetach()
    {
        super.onDetach();
        // 将 mCallbacks 赋为 null
        mCallbacks = null;
    }
}
```

上面程序的 onCreateView()方法中的粗体字代码是实现向服务器发送请求，把服务器响应转换成 JSONArray 对象，并使用 ListView 显示物品种类的核心代码。由于物品种类包含的信息量并不大，只有种类名称和种类描述两种，因此程序使用了 KindArrayAdapter 来包装 JSONArray 对象，KindArrayAdapter 提供的列表项既包括种类名称，也包括种类描述。下面是该 Adapter 类的代码。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\KindArrayAdapter.java

```
public class KindArrayAdapter extends BaseAdapter
{
    // 需要包装的 JSONArray
    private JSONArray kindArray;
    private Context ctx;
    public KindArrayAdapter(JSONArray kindArray
        ,Context ctx)
    {
        this.kindArray = kindArray;
        this.ctx = ctx;
```

```
    }
    @Override
    public int getCount()
    {
        // 返回 ListView 包含的列表项的数量
        return kindArray.length();
    }
    @Override
    public Object getItem(int position)
    {
        // 获取指定列表项所包装的 JSONObject
        return kindArray.optJSONObject(position);
    }
    @Override
    public long getItemId(int position)
    {
        try
        {
            return ((JSONObject) getItem(position)).getInt("id");
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        return -1;
    }
    @Override
    public View getView(int position, View convertView,
        ViewGroup parent)
    {
        // 定义一个线性布局管理器
        LinearLayout container = new LinearLayout(ctx);
        // 设置为水平的线性布局管理器
        container.setOrientation(1);
        // 定义一个线性布局管理器
        LinearLayout linear = new LinearLayout(ctx);
        // 设置为水平的线性布局管理器
        linear.setOrientation(0);
        // 创建一个 ImageView
        ImageView iv = new ImageView(ctx);
        iv.setPadding(10, 0, 20, 0);
        iv.setImageResource(R.drawable.item);
        // 将图片添加到 LinearLayout 中
        linear.addView(iv);
        // 创建一个 TextView
        TextView tv = new TextView(ctx);
        try
        {
            // 获取 JSONArray 数组元素的 kindName 属性
            String kindName = ((JSONObject)getItem(position))
                .getString("kindName");
            // 设置 TextView 所显示的内容
            tv.setText(kindName);
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        tv.setTextSize(20);
        // 将 TextView 添加到 LinearLayout 中
        linear.addView(tv);
        container.addView(linear);
        // 定义一个文本框来显示种类描述
```

```
        TextView descView = new TextView(ctx);
        descView.setPadding(30, 0, 0, 0);
        try
        {
            // 获取 JSONArray 数组元素的 kindDesc 属性
            String kindDesc = ((JSONObject)getItem(position))
                .getString("kindDesc");
            descView.setText(kindDesc);
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        descView.setTextSize(16);
        container.addView(descView);
        return container;
    }
}
```

从上面程序中的粗体字代码可以看出，该 Adapter 提供的每个列表项不仅包括种类名称，而且包括种类描述，这样用户查看种类时一目了然。

用户查看物品种类时将可看到如图 19.13 所示的界面。

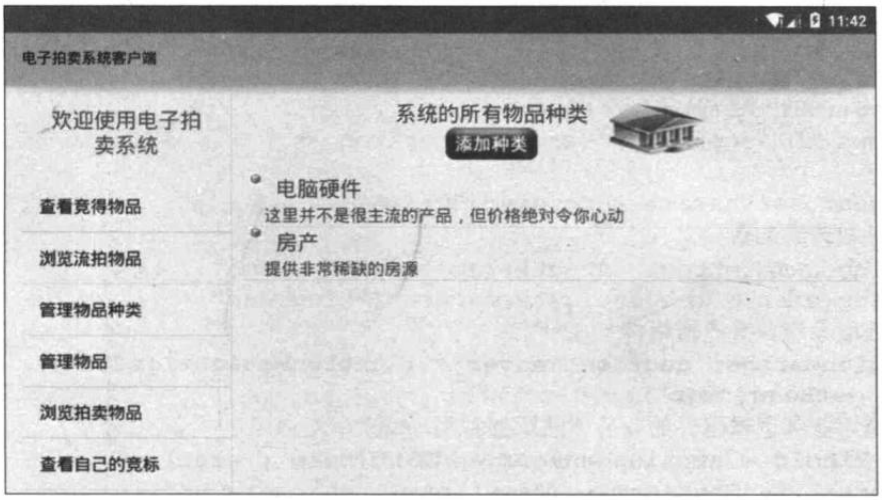


图 19.13 在平板电脑上查看物品种类

为了在手机上运行该应用，程序提供了一个 ManageKind Activity 来包装、显示该 Fragment。ManageKind 的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageKind.java

```
public class ManageKind extends FragmentActivity
    implements Callbacks
{
    @Override
    public Fragment getFragment()
    {
        return new ManageKindFragment();
    }
    @Override
    public void onItemSelected(Integer id, Bundle bundle)
    {
        // 当用户单击 ManageKindFragment 中的“添加”按钮时，启动 AddKind Activity
        Intent i = new Intent(this , AddKind.class);
        startActivity(i);
    }
}
```

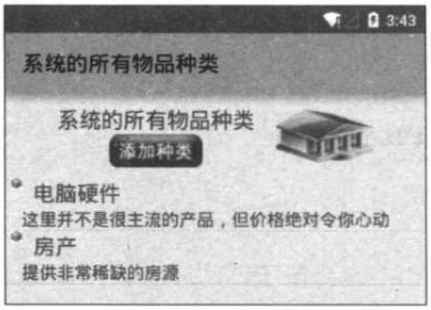


图 19.14 在手机上查看物品种类

正如上面的粗体字代码所示，ManageKind 仅仅是包装、显示 ManageKindFragment，因此在手机上运行该程序时，将可以看到如图 19.14 所示的界面。

在图 19.13、图 19.14 所示界面中有一个“添加种类”按钮，当用户单击该按钮时，Fragment 将会回调它所在 Activity 的 onItemSelected()方法。对于在平板电脑上运行的 Activity，onItemSelected()方法将会让 FrameLayout 装载 AddKindFragment；而在手机上运行的 Activity，onItemSelected()方法将会启动 AddKindActivity（该 Activity 仅仅是包装、显示 AddKindFragment）。

19.6.3 添加种类的 Servlet

添加物品种类的 Servlet 也是调用业务逻辑组件的方法来完成添加的。该 Servlet 的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddKindServlet.java

```
@WebServlet(urlPatterns="/android/addKind.jsp")
public class AddKindServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        request.setCharacterEncoding("utf-8");
        // 获取请求参数
        String name = request.getParameter("kindName");
        String desc = request.getParameter("kindDesc");
        // 获取系统业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 调用业务逻辑组件的业务方法添加种类
        int kindId = auctionManager.addKind(name , desc);
        response.setContentType("text/html; charset=GBK");
        // 添加成功
        if (kindId > 0)
        {
            response.getWriter().println("恭喜您，种类添加成功!");
        }
        else
        {
            response.getWriter().println("对不起，种类添加失败!");
        }
    }
}
```

上面 Servlet 调用业务逻辑组件的方法添加物品后，直接返回了添加结果的字符串，因此 Android 客户端只要直接显示该添加结果即可。

19.6.4 添加物品种类

添加物品种类的界面上包括两个输入框，用于接受用户输入种类名称和种类描述，并提供添加、取消两个按钮。该界面布局比较简单，此处不再给出界面布局代码。

添加物品种类的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddKindFragment.java

```
public class AddKindFragment extends Fragment
{
    // 定义界面中的两个文本框
    EditText kindName, kindDesc;
    // 定义界面中的两个按钮
    Button bnAdd, bnCancel;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.add_kind
            , container , false);
        // 获取界面中的两个编辑框
        kindName = (EditText)rootView.findViewById(R.id.kindName);
        kindDesc = (EditText)rootView.findViewById(R.id.kindDesc);
        // 获取界面中的两个按钮
        bnAdd = (Button)rootView.findViewById(R.id.bnAdd);
        bnCancel = (Button)rootView.findViewById(R.id.bnCancel);
        // 为“取消”按钮的单击事件绑定事件监听器
        bnCancel.setOnClickListener(new HomeListener(getActivity()));
        bnAdd.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 输入校验
                if (validate())
                {
                    // 获取用户输入的种类名称、种类描述
                    String name = kindName.getText().toString();
                    String desc = kindDesc.getText().toString();
                    try
                    {
                        // 添加物品种类
                        String result =  addKind(name, desc);
                        // 使用对话框来显示添加结果
                        DialogUtil.showDialog(getActivity()
                            , result , true);
                    }
                    catch (Exception e)
                    {
                        DialogUtil.showDialog(getActivity()
                            , "服务器响应异常，请稍后再试！" , false);
                        e.printStackTrace();
                    }
                }
            }
        });
        return rootView;
    }
    // 对用户输入的种类名称进行校验
    private boolean validate()
    {
        String name = kindName.getText().toString().trim();
        if (name.equals(""))
        {
            DialogUtil.showDialog(getActivity() , "种类名称是必填项！" , false);
            return false;
        }
        return true;
    }
```

```
    private String addKind(String name, String desc)
        throws Exception
    {
        // 使用 Map 封装请求参数
        Map<String , String> map = new HashMap<>();
        map.put("kindName" , name);
        map.put("kindDesc" , desc);
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "addKind.jsp";
        // 发送请求
        return HttpUtil.postRequest(url , map);
    }
}
```

上面的 Fragment 先对用户输入的种类名称、种类描述进行输入校验，然后调用 addKind() 方法来添加物品种类，该方法利用 HttpUtil 发送 POST 请求完成添加。添加成功后，可以看到系统显示如图 19.15 所示的对话框。

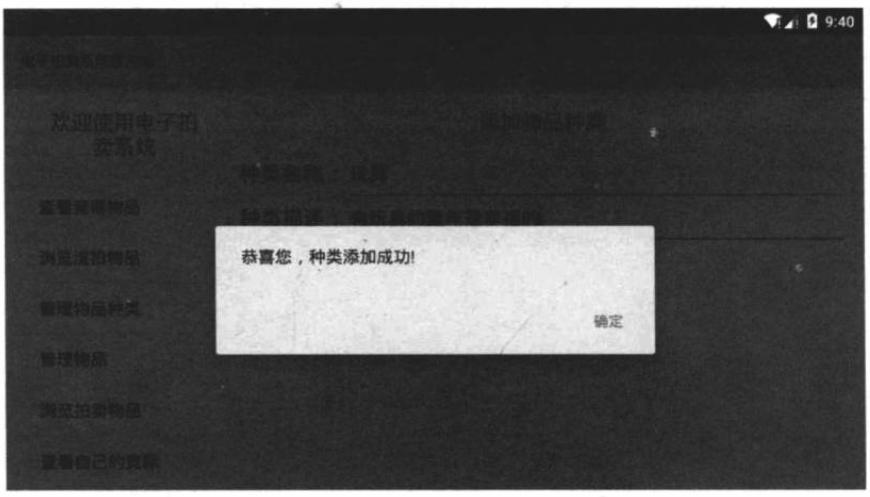


图 19.15　在平板电脑上添加物品种类

为了兼顾手机屏幕，同样需要定义 Activity 来加载、显示该 Fragment。AddKind Activity 的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddKind.java

```
public class AddKind extends FragmentActivity
{
    @Override
    public Fragment getFragment()
    {
        return new AddKindFragment();
    }
}
```

正如上面的粗体字代码所示，AddKind 什么都没干，仅仅是加载、显示 AddKindFragment。在手机上添加物品种类，如果添加成功，将看到如图 19.16 所示的对话框。

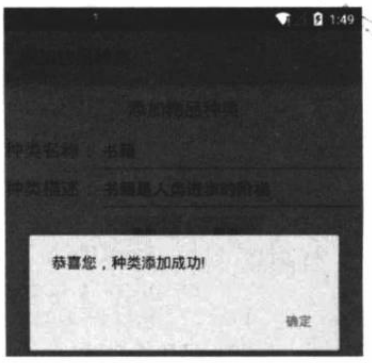


图 19.16　在手机上添加物品种类

19.7　管理拍卖物品

管理拍卖物品包括浏览自己的拍卖物品、添加拍卖物品两大主要功能。Android 客户端主要充当用户交互的客户端：显示当前用户的拍卖物品；添加拍卖物品时提

供输入框供用户输入物品名称、物品描述等必要信息。

19.7.1 查看自己的拍卖物品的 Servlet

查看自己的拍卖物品的 Servlet 调用业务逻辑组件的业务逻辑方法来获取自己的拍卖物品，接下来该 Servlet 将会把该物品列表包装成 JSONArray 对象，再转换成 JSON 字符串后输出。

下面是该 Servlet 的代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewOwnerItemServlet. java

```
@WebServlet(urlPatterns="/android/viewOwnerItem.jsp")
public class ViewOwnerItemServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        // 获取 userId
        Integer userId = (Integer)request.getSession(true)
            .getAttribute("userId");
        // 获取业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 获取该用户当前处于拍卖中的所有物品
        List<ItemBean> items = auctionManager.getItemsByOwner(userId);
        // 将查询得到的物品包装成 JSONArray 对象
        JSONArray jsonArr= new JSONArray(items);
        response.setContentType("text/html; charset=GBK");
        response.getWriter().println(jsonArr.toString());
    }
}
```

该 Servlet 中的粗体字代码调用了业务逻辑方法来获取当前用户所有处于拍卖中的物品。直接使用浏览器向该 Servlet 发送请求，将可以看到该 Servlet 输出如图 19.17 所示的 JSON 字符串。

图 19.17 JSON 字符串响应

当服务器返回如图 19.17 所示的 JSON 字符串响应之后，Android 客户端就可以把它转换成 JSONArray 对象了，并从中获取详细的物品信息。

19.7.2 查看自己的拍卖物品

查看拍卖物品的界面使用 ListView 来显示物品列表。该界面布局的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\res\manage_item.xml

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

```
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/sub_title_margin">
<LinearLayout
    android:orientation="vertical"
    android:gravity="center"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="12dp">
<TextView
    android:text="@string/manage_item"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<!-- 添加物品的按钮 -->
<Button
    android:id="@+id/bnAdd"
    android:layout_width="85dp"
    android:layout_height="30dp"
    android:background="@drawable/add_item"/>
</LinearLayout>
<Button
    android:id="@+id/bn_home"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="@dimen/label_font_size"
    android:background="@drawable/home"/>
</LinearLayout>
<!-- 显示物品列表的 ListView -->
<ListView
    android:id="@+id/itemList"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
```

上面的界面布局中定义了一个 ListView 来显示当前用户的拍卖物品。除此之外，该界面还包含了几个按钮，其中一个按钮用于启动添加物品的用户界面。

管理物品的 Fragment 只要通过 HttpUtil 向服务器发送请求，并把服务器响应转换成 JSONArray 对象，然后把 JSONArray 对象包装成 Adapter，再使用 ListView 来显示这些物品即可。管理物品的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageItemFragment.java

```
public class ManageItemFragment extends Fragment
{
    public static final int ADD_ITEM = 0x1006;;
    Button bnHome, bnAdd;
    ListView itemList;
    Callbacks mCallbacks;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.manage_item
            , container , false);
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        bnAdd = (Button) rootView.findViewById(R.id.bnAdd);
        itemList = (ListView) rootView.findViewById(R.id.itemList);
        // 为“返回”按钮的单击事件绑定事件监听器
```

```
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        bnAdd.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                mCallbacks.onItemSelected(ADD_ITEM , null);
            }
        });
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "viewOwnerItem.jsp";
        try
        {
            // 向指定 URL 发送请求
            JSONArray jsonArray = new JSONArray(HttpUtil.getRequest(url));
            // 将服务器响应包装成 Adapter
            JSONArrayAdapter adapter = new JSONArrayAdapter(getActivity()
                , jsonArray, "name", true);
            itemList.setAdapter(adapter);
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity()
                , "服务器响应异常，请稍后再试！", false);
            e.printStackTrace();
        }
        itemList.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                viewItemInBid(position);  // ①
            }
        });
        return rootView;
    }
    // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 如果 Activity 没有实现 Callbacks 接口，抛出异常
        if (!(activity instanceof Callbacks))
        {
            throw new IllegalStateException(
                "ManagerItemFragment 所在的 Activity 必须实现 Callbacks 接口！");
        }
        // 把该 Activity 当成 Callbacks 对象
        mCallbacks = (Callbacks) activity;
    }
    // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
    @Override
    public void onDetach()
    {
        super.onDetach();
        // 将 mCallbacks 赋值为 null
        mCallbacks = null;
    }
    private void viewItemInBid(int position)
    {
        // 加载 detail_in_bid.xml 界面布局代表的视图
```

```
            View detailView = getActivity().getLayoutInflater()
                .inflate(R.layout.detail_in_bid, null);
            // 获取 detail_in_bid.xml 界面中的文本框
            TextView itemName = (TextView) detailView
                .findViewById(R.id.itemName);
            TextView itemKind = (TextView) detailView
                .findViewById(R.id.itemKind);
            TextView maxPrice = (TextView) detailView
                .findViewById(R.id.maxPrice);
            TextView initPrice = (TextView) detailView
                .findViewById(R.id.initPrice);
            TextView endTime = (TextView) detailView
                .findViewById(R.id.endTime);
            TextView itemRemark = (TextView) detailView
                .findViewById(R.id.itemRemark);
            // 获取被单击列表项所包装的 JSONObject
            JSONObject jsonObj = (JSONObject) itemList.getAdapter().getItem(
                position);
            try
            {
                // 通过文本框显示物品详情
                itemName.setText(jsonObj.getString("name"));
                itemKind.setText(jsonObj.getString("kind"));
                maxPrice.setText(jsonObj.getString("maxPrice"));
                itemRemark.setText(jsonObj.getString("desc"));
                initPrice.setText(jsonObj.getString("initPrice"));
                endTime.setText(jsonObj.getString("endTime"));
            }
            catch (JSONException e)
            {
                e.printStackTrace();
            }
            DialogUtil.showDialog(getActivity(), detailView);
        }
    }
```

上面程序中的粗体字代码使用了 HttpUtil 向服务器发送请求，并使用 ListView 来显示服务器返回的物品列表。当用户在平板电脑上进入管理物品的界面时，将会看到如图 19.18 所示的界面。

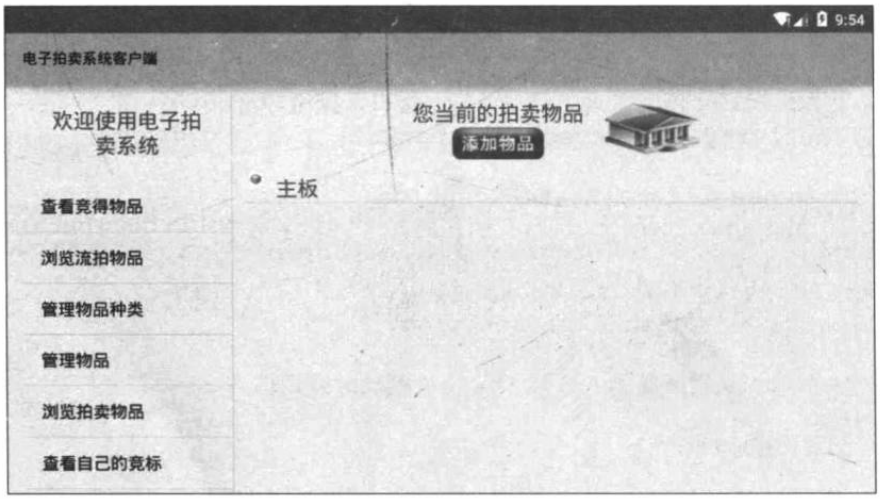


图 19.18 管理自己的拍卖物品

图 19.18 只是列出拍卖物品的物品名，如果用户需要查看该物品的详情，则可以选择单击代表该物品的列表项，程序将会触发 viewItemInBid(position)方法（如上面程序中①号粗体字代码所示），该方法将会启动一个对话框来显示该物品的详情。该对话框中定义了多个文本框，

多个文本框显示了该物品的详情。

用户单击物品列表中的指定物品时，将可以看到如图 19.19 所示的界面。

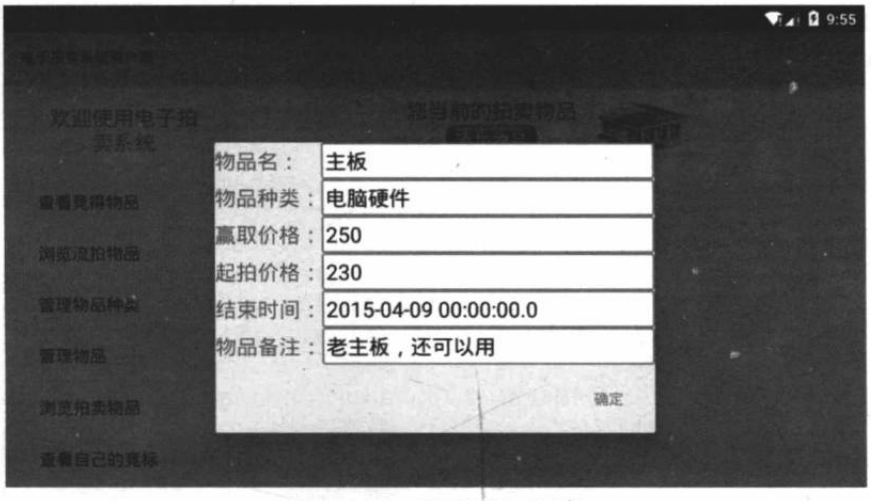


图 19.19 查看物品详情

为了兼顾手机屏幕，同样需要定义 Activity 来加载、显示该 Fragment。ManageItem Activity 的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageItem.java

```
public class ManageItem extends FragmentActivity
    implements Callbacks
{
    @Override
    public Fragment getFragment()
    {
        return new ManageItemFragment();
    }
    @Override
    public void onItemSelected(Integer id, Bundle bundle)
    {
        // 当用户单击“添加”按钮时，将会启动 AddItem Activity
        Intent i = new Intent(this , AddItem.class);
        startActivity(i);
    }
}
```

在手机上查看物品详情，将可以看到如图 19.20 所示的界面。

图 19.20 在手机上查看物品详情

在图 19.18、图 19.20 所示界面上都包含一个“添加物品”按钮，用户单击该按钮即可添加拍卖物品。

19.7.3 添加拍卖物品的 Servlet

添加拍卖物品的 Servlet 将会调用业务逻辑组件的方法来添加物品，添加物品时 Servlet 先解析请求参数，这些请求参数代表了新增物品的属性，再调用业务逻辑方法即可。该 Servlet 的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddItemServlet.java

```
@WebServlet(urlPatterns="/android/addItem.jsp")
public class AddItemServlet extends BaseServlet
{
```

```
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        // 获取 userId
        Integer userId = (Integer)request.getSession(true)
            .getAttribute("userId");
        request.setCharacterEncoding("gbk");
        // 解析请求参数
        String itemName = request.getParameter("itemName");
        String itemDesc = request.getParameter("itemDesc");
        String remark = request.getParameter("itemRemark");
        String initPrice = request.getParameter("initPrice");
        String kindId = request.getParameter("kindId");
        String avail = request.getParameter("availTime");
        // 获取业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 调用业务逻辑组件的方法来添加物品
        int itemId = auctionManager.addItem(itemName ,itemDesc , remark
            , Double.parseDouble(initPrice) , Integer.parseInt(avail)
            , Integer.parseInt(kindId) , userId);
        response.setContentType("text/html; charset=GBK");
        // 添加成功
        if (itemId > 0)
        {
            response.getWriter().println("恭喜您，物品添加成功!");
        }
        else
        {
            response.getWriter().println("对不起，物品添加失败!");
        }
    }
}
```

上面程序中的粗体字代码用于调用业务逻辑方法来添加物品，如果添加成功，该 Servlet 将会输出添加成功的响应；否则将会输出添加失败的响应。

提供该 Servlet 之后，接下来 Android 客户端就可向该 Servlet 发送请求来添加物品了。

19.7.4 添加拍卖物品

添加拍卖物品的程序界面中包含多个文本框，这些文本框用于接受用户输入的物品属性。添加拍卖物品的界面布局文件如下。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\add_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/add_item_title"
    android:textSize="@dimen/label_font_size"
    android:gravity="center"
    android:padding="@dimen/title_padding"/>
<!-- 输入物品名称的行 -->
<TableRow>
<TextView
```

```
        android:text="@string/item_name"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/itemName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>
    </TableRow>
    <!-- 输入物品描述的行 -->
    <TableRow>
    <TextView
        android:text="@string/item_desc"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/itemDesc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>
    </TableRow>
    <!-- 输入物品备注的行 -->
    <TableRow>
    <TextView
        android:text="@string/remark"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/itemRemark"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>
    </TableRow>
    <!-- 输入起拍价格的行 -->
    <TableRow>
    <TextView
        android:text="@string/init_price"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/initPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"/>
    </TableRow>
    <!-- 选择有效时间的行 -->
    <TableRow>
    <TextView
        android:text="@string/avail_time"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Spinner
        android:id="@+id/availTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/availTime"/>
    </TableRow>
    <!-- 选择物品种类的行 -->
    <TableRow>
```

```
    <TextView
        android:text="@string/item_kind"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Spinner
        android:id="@+id/itemKind"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    </TableRow>
    <!-- 定义按钮的行 -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center">
    <Button
        android:id="@+id/bnAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/add"/>
    <Button
        android:id="@+id/bnCancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cancel"/>
    </LinearLayout>
    </TableLayout>
```

上面的界面布局中包含了两个 Spinner 组件，它们定义了两个列表框供用户选择有效时间和物品种类。对于选择有效时间的 Spinner 来说，程序可以直接指定一个数组作为它的列表项；但对于选择物品种类的 Spinner 来说，程序必须加载系统中的所有物品种类来作为列表项。因此，为了在 Spinner 中加载物品种类，程序必须向/android/viewKind.jsp 发送请求，然后把响应包装成 Adapter，接下来使用 Spinner 来显示种类列表。

当用户在平板电脑上进入添加拍卖物品的界面之后，将会看到如图 19.21 所示的输入界面。

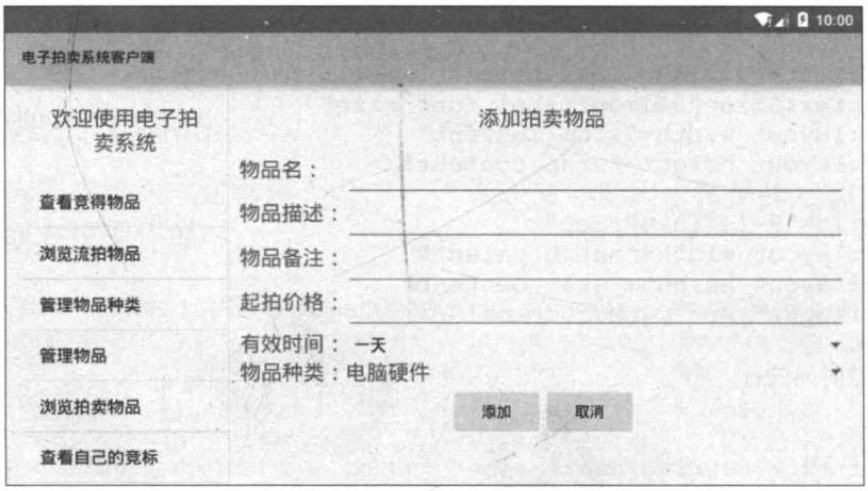


图 19.21 添加拍卖物品

从图 19.21 所示的界面可以看出，程序最下面的物品种类就是从系统中加载的。当用户填写了拍卖物品的详情之后，程序将会向/android/addItem.jsp 发送请求来添加拍卖物品。添加拍卖物品的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddItemFragment.java

```
public class AddItemFragment extends Fragment
{
```

```
// 定义界面中的文本框
EditText itemName, itemDesc,itemRemark,initPrice;
Spinner itemKind , availTime;
// 定义界面中的两个按钮
Button bnAdd, bnCancel;
@Override
public View onCreateView(LayoutInflater inflater
    , ViewGroup container, Bundle savedInstanceState)
{
    View rootView = inflater.inflate(R.layout.add_item
        , container , false);
    // 获取界面中的文本框
    itemName = (EditText) rootView.findViewById(R.id.itemName);
    itemDesc = (EditText) rootView.findViewById(R.id.itemDesc);
    itemRemark = (EditText) rootView.findViewById(R.id.itemRemark);
    initPrice = (EditText) rootView.findViewById(R.id.initPrice);
    itemKind = (Spinner) rootView.findViewById(R.id.itemKind);
    availTime = (Spinner) rootView.findViewById(R.id.availTime);
    // 定义发送请求的地址
    String url = HttpUtil.BASE_URL + "viewKind.jsp";
    JSONArray jsonArray = null;
    try
    {
        // 获取系统中所有的物品种类
        // 向执行 URL 发送请求，并把服务器响应包装成 JSONArray
        jsonArray = new JSONArray(HttpUtil.getRequest(url)); // ①
    }
    catch (Exception e1)
    {
        e1.printStackTrace();
    }
    // 将 JSONArray 包装成 Adapter
    JSONArrayAdapter adapter = new JSONArrayAdapter(
        getActivity() , jsonArray , "kindName" , false);
    // 显示物品种类列表
    itemKind.setAdapter(adapter);
    // 获取界面中的两个按钮
    bnAdd = (Button) rootView.findViewById(R.id.bnAdd);
    bnCancel = (Button) rootView.findViewById(R.id.bnCancel);
    // 为“取消”按钮的单击事件绑定事件监听器
    bnCancel.setOnClickListener(new HomeListener(getActivity()));
    bnAdd.setOnClickListener(new OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            // 执行输入校验
            if (validate())
            {
                // 获取用户输入的物品名称、物品描述等信息
                String name = itemName.getText().toString();
                String desc = itemDesc.getText().toString();
                String remark = itemRemark.getText().toString();
                String price = initPrice.getText().toString();
                JSONObject kind = (JSONObject) itemKind.getSelectedItem();
                int avail = availTime.getSelectedItemPosition();
                // 根据用户选择的有效时间选项，指定实际的有效时间
                switch(avail)
                {
                    case 5 :
                        avail = 7;
                        break;
                    case 6 :
```

```
                    avail = 30;
                    break;
                default :
                    avail += 1;
                    break;
            }
            try
            {
                // 添加物品
                String result = addItem(name, desc
                    , remark , price , kind.getInt("id") , avail);
                // 显示对话框
                DialogUtil.showDialog(getActivity()
                    , result , true);
            }
            catch (Exception e)
            {
                DialogUtil.showDialog(getActivity()
                    , "服务器响应异常，请稍后再试！" , false);
                e.printStackTrace();
            }
        }
    }
});
return rootView;
}
// 对用户输入的物品名称、起拍价格进行校验
private boolean validate()
{
    String name = itemName.getText().toString().trim();
    if (name.equals(""))
    {
        DialogUtil.showDialog(getActivity() , "物品名称是必填项！" , false);
        return false;
    }
    String price = initPrice.getText().toString().trim();
    if (price.equals(""))
    {
        DialogUtil.showDialog(getActivity() , "起拍价格是必填项！" , false);
        return false;
    }
    try
    {
        // 尝试把起拍价格转换为浮点数
        Double.parseDouble(price);
    }
    catch(NumberFormatException e)
    {
        DialogUtil.showDialog(getActivity() , "起拍价格必须是数值！" , false);
        return false;
    }
    return true;
}
private String addItem(String name, String desc
    , String remark , String initPrice , int kindId , int availTime)
    throws Exception
{
    // 使用Map封装请求参数
    Map<String , String> map = new HashMap<>();
    map.put("itemName" , name);
    map.put("itemDesc" , desc);
    map.put("itemRemark" , remark);
    map.put("initPrice" , initPrice);
    map.put("kindId" , kindId + "");
```

```
        map.put("availTime" , availTime + "");
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "addItem.jsp";
        // 发送请求
        return HttpUtil.postRequest(url , map);
    }
}
```

上面程序中的①号粗体字代码向/android/viewKind.jsp 发送请求，并把服务器响应包装成 JSONArray 对象，然后使用 Spinner 把这些物品种类显示出来即可。

当填写了拍卖物品的详情之后，用户单击“添加”按钮将先执行输入校验，然后调用 addItem()方法来添加物品，如上面程序中两行粗体字代码所示。

如果用户添加拍卖物品成功，将会看到如图 19.22 所示的对话框。

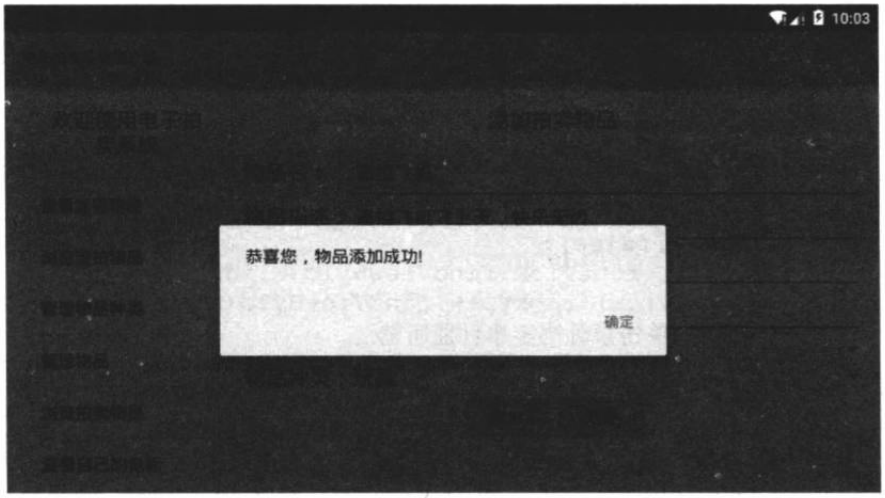


图 19.22 在平板电脑上添加拍卖物品成功

为了兼顾手机屏幕，同样需要定义 Activity 来加载、显示该 Fragment。AddItem Activity 的代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddItem.java

```
public class AddItem extends FragmentActivity
{
    @Override
    public Fragment getFragment()
    {
        return new AddItemFragment();
    }
}
```

从上面的粗体字代码不难看出，AddItem Activity 仅仅是加载并显示 AddItemFragment。在手机上添加物品成功后，可以看到如图 19.23 所示的界面。

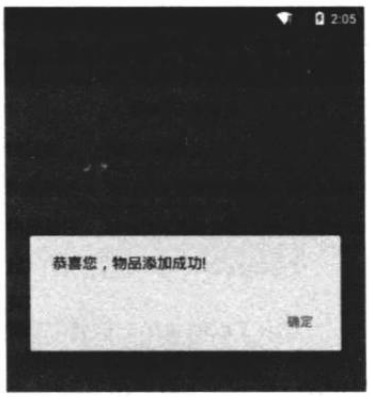


图 19.23 在手机上添加拍卖物品成功

19.8 参与竞拍

用户可以通过物品种类来浏览系统中的拍卖物品，找到合适的拍卖物品之后，可以对该物品进行竞价。

➤➤ 19.8.1 选择物品种类

前面介绍的/android/viewKind.jsp可以生成所有物品种类的响应，Android客户端只要向该Servlet发送请求即可显示物品种类供用户选择。

选择物品种类的界面上主要包含一个ListView，该ListView列出系统中全部物品种类，当用户单击某个物品种类时，程序将会显示该种类下的全部物品。

选择物品种类的Fragment代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ChooseKindFragment.java

```
public class ChooseKindFragment extends Fragment
{
    public static final int CHOOSE_ITEM = 0x1008;
    Callbacks mCallbacks;
    Button bnHome;
    ListView kindList;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.choose_kind
            , container , false);
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        kindList = (ListView) rootView.findViewById(R.id.kindList);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        // 定义发送请求的URL
        String url = HttpUtil.BASE_URL + "viewKind.jsp";
        try
        {
            // 向指定URL发送请求，并将服务器响应包装成JSONArray对象
            JSONArray jsonArray = new JSONArray(
                HttpUtil.getRequest(url)); // ①
            // 使用ListView显示所有物品种类
            kindList.setAdapter(new KindArrayAdapter(jsonArray
                , getActivity()));
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity()
                , "服务器响应异常，请稍后再试！" , false);
            e.printStackTrace();
        }
        kindList.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                Bundle bundle = new Bundle();
                bundle.putLong("kindId", id);
                mCallbacks.onItemSelected(CHOOSE_ITEM , bundle);
            }
        });
        return rootView;
    }
    // 当该Fragment被添加、显示到Activity时，回调该方法
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
```

```
        // 如果 Activity 没有实现 Callbacks 接口，抛出异常
        if (!(activity instanceof Callbacks))
        {
            throw new IllegalStateException(
                "ManageKindFragment 所在的 Activity 必须实现 Callbacks 接口!");
        }
        // 把该 Activity 当成 Callbacks 对象
        mCallbacks = (Callbacks) activity;
    }
    // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
    @Override
    public void onDetach()
    {
        super.onDetach();
        // 将 mCallbacks 赋为 null
        mCallbacks = null;
    }
}
```

程序中①号粗体字代码用于向指定 URL 发送请求，并将服务器响应转换成 JSONArray 对象，接下来程序只要把 JSONArray 对象包装成 Adapter，并使用 ListView 显示种类列表即可。

当用户单击指定物品种类时，程序将会回调该 Fragment 所在 Activity 的 onItemSelecte() 方法，并把当前物品种类作为参数传过去，而该 Fragment 所在 Activity 将会负责加载 Fragment，或启动对应的 Activity 来显示该种类下的全部拍卖物品。

➤➤ 19.8.2 根据种类浏览物品的 Servlet

根据种类浏览拍卖物品的 Servlet 调用业务逻辑方法来获取所有物品。该 Servlet 所做的就是调用业务逻辑方法来查询指定种类下的全部物品，并把这些物品包装成 JSONArray，再转换成 JSON 字符串输出。

下面是该 Servlet 的代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ItemListServlet.java

```
@WebServlet(urlPatterns="/android/itemList.jsp")
public class ItemListServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        request.setCharacterEncoding("gbk");
        // 获取物品种类 ID
        String kindId = request.getParameter("kindId");
        // 获取业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 调用业务逻辑方法来获取全部物品
        List<ItemBean> items = auctionManager
            .getItemsByKind(Integer.parseInt(kindId));
        // 将物品列表包装成 JSONArray
        JSONArray jsonArr= new JSONArray(items);
        response.setContentType("text/html; charset=GBK");
        response.getWriter().println(jsonArr.toString());
    }
}
```

上面程序中的粗体字代码调用了业务逻辑方法来获取指定种类的全部物品，接下来程序将这些物品包装成 JSONArray，再转换成 JSON 字符串输出。

直接使用浏览器访问该 Servlet，将看到如图 19.24 所示的 JSON 字符串。

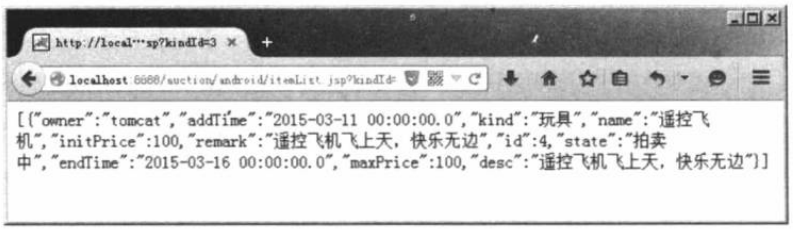


图 19.24 服务器响应的 JSON 字符串

19.8.3 根据种类浏览物品

服务器端 Servlet 生成图 19.24 所示的 JSON 字符串响应后，Android 客户端只要向该 Servlet 发送请求并把该响应转换为 JSONArray 对象，再将该 JSONArray 对象包装成 Adapter，并使用 ListView 来显示这些物品即可。

下面是 ChooseItemFragment 的代码。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ChooseItemFragment.java

```
public class ChooseItemFragment extends Fragment
{
    public static final int ADD_BID = 0x1009;
    Button bnHome;
    ListView succList;
    TextView viewTitle;
    Callbacks mCallbacks;
    // 重写该方法，该方法返回的 View 将作为 Fragment 显示的组件
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.view_item
            , container , false);
        // 获取界面中的“返回”按钮
        bnHome = (Button)rootView.findViewById(R.id.bn_home);
        succList = (ListView)rootView.findViewById(R.id.succList);
        viewTitle = (TextView)rootView.findViewById(R.id.view_titile);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        long kindId = getArguments().getLong("kindId");
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "itemList.jsp?kindId=" + kindId;
        try
        {
            // 根据种类 ID 获取该种类对应的所有物品
            JSONArray jsonArray = new JSONArray(HttpUtil.getRequest(url));
            JSONArrayAdapter adapter = new JSONArrayAdapter(
                getActivity() , jsonArray , "name" , true);
            // 使用 ListView 显示当前种类的所有物品
            succList.setAdapter(adapter);
        }
        catch (Exception e1)
        {
            DialogUtil.showDialog(getActivity() , "服务器响应异常，请稍后再试！" , false);
            e1.printStackTrace();
        }
        // 修改标题
        viewTitle.setText(R.string.item_list);
        succList.setOnItemClickListener(new OnItemClickListener()
        {
```

```
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
            int position, long id)
        {
            JSONObject jsonObj = (JSONObject) succList
                .getAdapter().getItem(position);
            Bundle bundle = new Bundle();
            try
            {
                bundle.putInt("itemId" , jsonObj.getInt("id"));
            }
            catch (JSONException e)
            {
                e.printStackTrace();
            }
            mCallbacks.onItemSelected(ADD_BID, bundle);
        }
    });
    return rootView;
}
// 当该 Fragment 被添加、显示到 Activity 时，回调该方法
@Override
public void onAttach(Activity activity)
{
    super.onAttach(activity);
    // 如果 Activity 没有实现 Callbacks 接口，抛出异常
    if (!(activity instanceof Callbacks))
    {
        throw new IllegalStateException(
            "ManagerItemFragment 所在的 Activity 必须实现 Callbacks 接口!");
    }
    // 把该 Activity 当成 Callbacks 对象
    mCallbacks = (Callbacks) activity;
}
// 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
@Override
public void onDetach()
{
    super.onDetach();
    // 将 mCallbacks 赋为 null
    mCallbacks = null;
}
}
```

上面程序中的粗体字代码用于向指定 Servlet 发送请求，并把服务器响应转换成 JSONArray 对象，再将该对象包装成 Adapter，并使用 ListView 来显示这些物品。

当用户在平板电脑上选择指定物品种类时，将会看到如图 19.25 所示的物品列表。

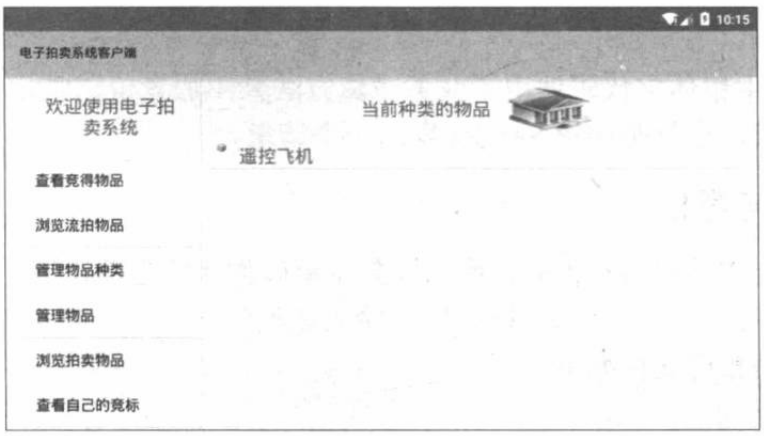


图 19.25 在平板电脑上查看指定种类的拍卖物品

图 19.25 中列出了指定种类下所有的拍卖物品，用户可以单击指定物品进入拍卖界面。

提示：

系统也为兼容手机屏幕提供了 ChooseItem Activity，该 Activity 的作用也只是包装并显示 ChooseItemFragment，此处不再给出 ChooseItem 的代码。

19.8.4 参与竞价的 Servlet

参与竞价的 Servlet 调用业务逻辑组件的业务方法来添加竞价记录，如果竞价成功，程序直接生成竞价成功的响应提示；如果竞价失败，程序生成竞价失败的响应提示。参与竞价的 Servlet 代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddBidServlet.java

```
@WebServlet(urlPatterns="/android/addBid.jsp")
public class AddBidServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        // 获取 userId
        Integer userId = (Integer)request.getSession(true)
            .getAttribute("userId");
        request.setCharacterEncoding("gbk");
        // 获取请求参数
        String itemId = request.getParameter("itemId");
        String bidPrice = request.getParameter("bidPrice");
        // 获取业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 调用业务方法来添加竞价记录
        int bidId = auctionManager.addBid(Integer.parseInt(itemId)
            , Double.parseDouble(bidPrice) , userId);
        response.setContentType("text/html; charset=GBK");
        // 竞价成功
        if (bidId > 0)
        {
            response.getWriter().println("恭喜您，竞价成功!");
        }
        else
        {
            response.getWriter().println("对不起，竞价失败!");
        }
    }
}
```

上面的 Servlet 中粗体字代码调用了业务逻辑方法来添加竞价记录，每次用户参与竞拍就是添加一条竞价记录，也就是向该 Servlet 发送一个请求。

19.8.5 参与竞价

当用户选择指定物品参与竞价时，系统将会显示该物品的当前详情，例如起拍价格、当前最高竞价等，界面下方提供一个输入框供用户输入竞拍价。

用于竞价的界面布局文件如下。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\add_bid.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1">
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/item_detail_title"
    android:textSize="@dimen/label_font_size"
    android:gravity="center"
    android:padding="@dimen/title_padding"/>
<!-- 显示物品名称的行 -->
<TableRow>
<TextView
    android:text="@string/item_name"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<TextView
    android:id="@+id/itemName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/tv_show"/>
</TableRow>
<!-- 显示物品描述的行 -->
<TableRow>
<TextView
    android:text="@string/item_desc"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<TextView
    android:id="@+id/itemDesc"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/tv_show"/>
</TableRow>
<!-- 显示物品备注的行 -->
<TableRow>
<TextView
    android:text="@string/remark"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<TextView
    android:id="@+id/itemRemark"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/tv_show"/>
</TableRow>
<!-- 显示物品种类的行 -->
<TableRow>
<TextView
    android:text="@string/item_kind"
    android:textSize="@dimen/label_font_size"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
<TextView
```

```
        android:id="@+id/itemKind"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示起拍价格的行 -->
    <TableRow>
    <TextView
        android:text="@string/init_price"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/initPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示最高竞价的行 -->
    <TableRow>
    <TextView
        android:text="@string/max_price"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/maxPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示结束时间的行 -->
    <TableRow>
    <TextView
        android:text="@string/end_time"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/endTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 输入竞拍价格的行 -->
    <TableRow>
    <TextView
        android:text="@string/you_bid"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/bidPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"/>
    </TableRow>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center">
    <Button
        android:id="@+id/bnAdd"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/bid"/>
<Button
    android:id="@+id/bnCancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/cancel"/>
</LinearLayout>
</TableLayout>
</ScrollView>
```

从上面的界面布局可以看出，当用户选择竞拍物品之后，程序会显示该物品的详情，界面下方包含一个文本框供用户填写竞拍价格。该界面如图 19.26 所示。

图 19.26 参与竞价

当用户在图 19.26 所示界面中输入竞拍价格，单击“竞价”按钮后，程序将会向 /android/addBid.jsp 发送请求。添加竞价记录。

参与竞价的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddBidFragment.java

```
public class AddBidFragment extends Fragment
{
    // 定义界面中的文本框
    TextView itemName, itemDesc,itemRemark,itemKind
        ,initPrice , maxPrice ,endTime;
    EditText bidPrice;
    // 定义界面中的两个按钮
    Button bnAdd, bnCancel;
    // 定义当前正在拍卖的物品
    JSONObject jsonObj;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.add_bid
            , container , false);
        // 获取界面中的编辑框
        itemName = (TextView)rootView.findViewById(R.id.itemName);
        itemDesc = (TextView)rootView.findViewById(R.id.itemDesc);
        itemRemark = (TextView)rootView.findViewById(R.id.itemRemark);
        itemKind = (TextView)rootView.findViewById(R.id.itemKind);
        initPrice = (TextView)rootView.findViewById(R.id.initPrice);
        maxPrice = (TextView)rootView.findViewById(R.id.maxPrice);
```

```
endTime = (TextView)rootView.findViewById(R.id.endTime);
bidPrice = (EditText)rootView.findViewById(R.id.bidPrice);
// 获取界面中的两个按钮
bnAdd = (Button)rootView.findViewById(R.id.bnAdd);
bnCancel = (Button)rootView.findViewById(R.id.bnCancel);
// 为“取消”按钮的单击事件绑定事件监听器
bnCancel.setOnClickListener(new HomeListener(getActivity()));
// 定义发送请求的 URL
String url = HttpUtil.BASE_URL + "getItem.jsp?itemId="
    + getArguments().getInt("itemId");
try
{
    // 获取指定的拍卖物品
    jsonObj = new JSONObject(HttpUtil.getRequest(url));
    // 使用文本框来显示拍卖物品的详情
    itemName.setText(jsonObj.getString("name"));
    itemDesc.setText(jsonObj.getString("desc"));
    itemRemark.setText(jsonObj.getString("remark"));
    itemKind.setText(jsonObj.getString("kind"));
    initPrice.setText(jsonObj.getString("initPrice"));
    maxPrice.setText(jsonObj.getString("maxPrice"));
    endTime.setText(jsonObj.getString("endTime"));
}
catch (Exception e1)
{
    DialogUtil.showDialog(getActivity(), "服务器响应出现异常！", false);
    e1.printStackTrace();
}
bnAdd.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        try
        {
            // 执行类型转换
            double curPrice = Double.parseDouble(
                bidPrice.getText().toString());
            // 执行输入校验
            if(curPrice < jsonObj.getDouble("maxPrice"))  // ①
            {
                DialogUtil.showDialog(getActivity(),
                    "您输入的竞价必须高于当前竞价", false);
            }
            else
            {
                // 添加竞价记录
                String result = addBid(jsonObj.getString("id")
                    , curPrice + ""); // ②
                // 显示对话框
                DialogUtil.showDialog(getActivity()
                    , result , true);
            }
        }
        catch(NumberFormatException ne)
        {
            DialogUtil.showDialog(getActivity()
                , "您输入的竞价必须是数值", false);
        }
```

```
                catch(Exception e)
                {
                    e.printStackTrace();
                    DialogUtil.showDialog(getActivity()
                        , "服务器响应出现异常，请重试！", false);
                }
            }
        });
        return rootView;
    }
    private String addBid(String itemId , String bidPrice)
        throws Exception
    {
        // 使用Map封装请求参数
        Map<String , String> map = new HashMap<String, String>();
        map.put("itemId" , itemId);
        map.put("bidPrice" , bidPrice);
        // 定义请求将会发送到addKind.jsp页面
        String url = HttpUtil.BASE_URL + "addBid.jsp";
        // 发送请求
        return HttpUtil.postRequest(url , map);
    }
}
```

从上面的程序可以看出，当用户单击“竞价”按钮后，程序将先执行输入校验，再调用addBid()方法来添加竞价记录，如上面程序中的①、②两行粗体字代码所示。

如果用户竞价成功，系统将显示如图19.27所示的对话框。

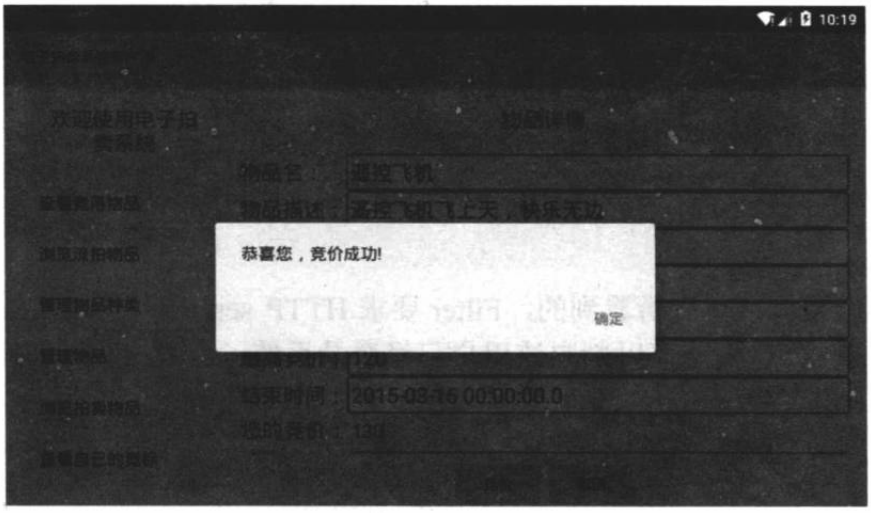


图19.27 竞价成功

提示： 系统也为兼容手机屏幕提供了AddBid Activity，该Activity的作用也只是包装并显示AddBidFragment，此处不再给出该类的代码。

19.9 权限控制

前面介绍时已经看到，服务器端程序在处理用户登录时，如果用户登录成功，系统会把用户ID放入HTTP session中，方便系统跟踪用户的登录状态。

对于Android客户端程序来说，由于Android客户端采用了Apache HttpClient来发送请求、

获取响应，因此HttpClient会自动维护与服务器之间的登录状态。在有效时间之内，服务器端程序可以跟踪到Android客户端的登录状态。

本系统要求只有登录用户才能使用系统功能，因此程序考虑在服务器端使用Filter进行控制，Filter只要拦截匹配的/android/*（匹配该URL的Servlet向Android客户端提供响应）的URL即可。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\Authority.java

```
@WebFilter(urlPatterns="/android/*")
public class Authority implements Filter
{
    public void init(FilterConfig config)
        throws ServletException{ }
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain chain)
        throws IOException , ServletException
    {
        HttpServletRequest hrequest = (HttpServletRequest)request;
        // 获取HttpSession对象
        HttpSession session = hrequest.getSession(true);
        Integer userId = (Integer)session.getAttribute("userId");
        // 如果用户已经登录，或用户正在登录
        if ((userId != null && userId > 0)
            || hrequest.getRequestURI().endsWith("/login.jsp"))
        {
            // “放行”请求
            chain.doFilter(request , response);
        }
        else
        {
            response.setContentType("text/html; charset=GBK");
            // 生成错误提示。
            response.getWriter().println("您还没有登录系统，请先登录系统！");
        }
    }
    public void destroy(){ }
}
```

正如从上面的粗体字代码所看到的，Filter要求HTTP session中的userId属性不为null，且userId属性大于0，这样就可以判定该用户已经登录系统；否则该Filter不会“放行”请求，而是直接向客户端生成提示信息。

注意：

笔者见过一些与该应用类似的Android应用的权限控制，它们并没有在服务器端进行权限控制，而是只要求用户在第一次使用时登录系统，但实际上这是不够的。如果应用程序不在服务器端进行权限控制，而是只在客户端要求用户登录，那么这种系统的安全控制十分脆弱。

对于绝大部分普通用户来说，这样的安全控制可能没有太大的问题；但对于恶意用户来说，他可以采用多种方法来绕过客户端的登录要求——最简单的方法比如反编译Android应用，让用户在启动程序时立即进入Main Activity，如果服务器不再进行权限控制，那么整个应用就“赤裸裸”地暴露出来了，这将是一件可怕的事情。所以这里系统不仅要求用户第一次使用时进行登录，而且程序还在服务器端进行了控制，这样才可保证系统的安全性。

19.10 本章小结

本章介绍了一个非常实用的 Android 应用，Android 应用充当电子拍卖系统的客户端，服务器端则采用 Struts 2 + Spring + Hibernate 的技术组合，架构上采用了控制器层、业务逻辑层、DAO 层的分层架构，保证整个项目具有极好的可扩展性和可维护性。由于本书是一本介绍 Android 开发的图书，因此本章并未介绍服务器端的业务逻辑组件、DAO 组件的实现，而是重点介绍了 Android 客户端的实现，包括为 Android 客户端提供响应的 Servlet 实现，Android 客户端的界面布局，Activity 实现等。本章的 Android 客户端通过 Apache HttpClient 与服务器交互，服务器与客户端之间采用 JSON 作为数据交换格式。读者学习本章需要重点掌握 Android 应用与传统企业应用整合的方式，二者之间通过网络进行数据交换的方式。

電子工業出版社 PUBLISHING HOUSE OF ELECTRONICS INDUSTRY http://www.phei.com.cn
Broadview WWW.BROADVIEW.COM.CN
博文视点·IT出版旗舰品牌

十载耕耘奠定专业地位

博文视点诚邀精锐作者加盟

以书为证彰显卓越品质

《C++Primer（中文版）（第5版）》、《淘宝技术这十年》、《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目，被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、昝辉Zac、李刚、曹江华……

“明星”级作者济济一堂，他们的名字熠熠生辉，与IT业的蓬勃发展紧密相连。

十年的开拓、探索和励精图治，成就**博**古通今、**文**圆质方、**视**角独特、**点**石成金之计算机图书的风向标杆：博文视点。

“凤翱翔于千仞兮，非梧不栖”，博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟，与大师并列于IT专业出版之巅。

英雄帖

江湖风云起，代有才人出。
IT界群雄并起，逐鹿中原。
博文视点诚邀天下技术英豪加入，
指点江山，激扬文字
传播信息技术，分享IT心得

• 专业的作者服务 •

博文视点自成立以来一直专注于IT专业技术图书的出版，拥有丰富的与技术图书作者合作的经验，并参照IT技术图书的特点，打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持：

善待作者——我们会把出版流程整理得清晰简明，为作者提供优厚的稿酬服务，解除作者的顾虑，安心写作，展现出最好的作品。

尊重作者——我们尊重每一位作者的技术实力和生活习惯，并会参照作者实际的工作、生活节奏，量身制定写作计划，确保合作顺利进行。

提升作者——我们打造精品图书，更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力，为作者的事业开拓带来更多的机会。

联系我们

博文视点官网：http://www.broadview.com.cn　　CSDN官方博客：http://blog.csdn.net/broadview2006/

投稿电话：010-51260888　88254368　　投稿邮箱：jsj@phei.com.cn

新浪微博 weibo.com　@博文视点Broadview　　微信公众账号　博文视点Broadview

[General Information]
书名=疯狂Android讲义
作者=李刚编著
页数=766
SS号=13805999
出版日期=2015.06
出版社=北京：电子工业出版社
ISBN号=978-7-121-25958-6
中图法分类号=TN929.53
原书定价=108.00
主题词=移动电话机-应用程序-程序设计-教材
参考文献格式=李刚编著.疯狂Android讲义.北京：电子工业出版社,2015.06.
内容提要=本书介绍了Android应用开发的相关知识，内容覆盖了Android用户界面编程、Android四大组件、Android资源访问、图形/图像处理、事件处理机制、Android输入/输出处理、OpenGL与3D应用开发、网络通信编程、传感器应用开发、GPS应用开发、GoogleMap服务等。